

ChibiOS/HAL

3.0.2

Reference Manual

Sat Jan 23 2016 09:51:35

Contents

1 ChibiOS/HAL	1
1.1 Copyright	1
1.2 Introduction	1
1.3 Related Documents	1
2 Deprecated List	3
3 Module Index	5
3.1 Modules	5
4 Hierarchical Index	7
4.1 Class Hierarchy	7
5 Data Structure Index	9
5.1 Data Structures	9
6 File Index	13
6.1 File List	13
7 Module Documentation	17
7.1 ADC Driver	17
7.1.1 Detailed Description	17
7.1.2 Driver State Machine	17
7.1.3 ADC Operations	18
7.1.3.1 ADC Conversion Groups	18
7.1.3.2 ADC Conversion Modes	18
7.1.3.3 ADC Callbacks	18
7.1.4 Macro Definition Documentation	21
7.1.4.1 ADC_USE_WAIT	21
7.1.4.2 ADC_USE_MUTUAL_EXCLUSION	21
7.1.4.3 _adc_reset_i	21
7.1.4.4 _adc_reset_s	21
7.1.4.5 _adc_wakeup_isr	22
7.1.4.6 _adc_timeout_isr	22

7.1.4.7	_adc_isr_half_code	22
7.1.4.8	_adc_isr_full_code	23
7.1.4.9	_adc_isr_error_code	23
7.1.4.10	PLATFORM_ADC_USE_ADC1	24
7.1.5	Typedef Documentation	24
7.1.5.1	adcsample_t	24
7.1.5.2	adc_channels_num_t	24
7.1.5.3	ADCDriver	24
7.1.5.4	adccallback_t	24
7.1.5.5	adcerrorcallback_t	24
7.1.6	Enumeration Type Documentation	25
7.1.6.1	adcstate_t	25
7.1.6.2	adcerror_t	25
7.1.7	Function Documentation	25
7.1.7.1	adclInit	25
7.1.7.2	adcObjectInit	26
7.1.7.3	adcStart	26
7.1.7.4	adcStop	27
7.1.7.5	adcStartConversion	28
7.1.7.6	adcStartConversionl	29
7.1.7.7	adcStopConversion	30
7.1.7.8	adcStopConversionl	30
7.1.7.9	adcConvert	31
7.1.7.10	adcAcquireBus	31
7.1.7.11	adcReleaseBus	32
7.1.7.12	adc_lld_init	32
7.1.7.13	adc_lld_start	33
7.1.7.14	adc_lld_stop	33
7.1.7.15	adc_lld_start_conversion	33
7.1.7.16	adc_lld_stop_conversion	33
7.1.8	Variable Documentation	34
7.1.8.1	ADCD1	34
7.2	CAN Driver	35
7.2.1	Detailed Description	35
7.2.2	Driver State Machine	35
7.2.3	Macro Definition Documentation	37
7.2.3.1	CAN_LIMIT_WARNING	37
7.2.3.2	CAN_LIMIT_ERROR	37
7.2.3.3	CAN_BUS_OFF_ERROR	38
7.2.3.4	CAN_FRAMING_ERROR	38

7.2.3.5	CAN_OVERFLOW_ERROR	38
7.2.3.6	CAN_ANY_MAILBOX	38
7.2.3.7	CAN_USE_SLEEP_MODE	38
7.2.3.8	CAN_MAILBOX_TO_MASK	38
7.2.3.9	CAN_TX_MAILBOXES	38
7.2.3.10	CAN_RX_MAILBOXES	38
7.2.3.11	PLATFORM_CAN_USE_CAN1	38
7.2.4	Typedef Documentation	38
7.2.4.1	canmbx_t	38
7.2.5	Enumeration Type Documentation	39
7.2.5.1	canstate_t	39
7.2.6	Function Documentation	39
7.2.6.1	canInit	39
7.2.6.2	canObjectInit	39
7.2.6.3	canStart	40
7.2.6.4	canStop	40
7.2.6.5	canTransmit	41
7.2.6.6	canReceive	42
7.2.6.7	canSleep	43
7.2.6.8	canWakeup	44
7.2.6.9	can_lld_init	45
7.2.6.10	can_lld_start	45
7.2.6.11	can_lld_stop	46
7.2.6.12	can_lld_is_tx_empty	46
7.2.6.13	can_lld_transmit	46
7.2.6.14	can_lld_is_rx_nonempty	46
7.2.6.15	can_lld_receive	47
7.2.6.16	can_lld_sleep	47
7.2.6.17	can_lld_wakeup	47
7.2.7	Variable Documentation	48
7.2.7.1	CAND1	48
7.3	DAC Driver	49
7.3.1	Detailed Description	49
7.3.2	Macro Definition Documentation	51
7.3.2.1	DAC_USE_WAIT	51
7.3.2.2	DAC_USE_MUTUAL_EXCLUSION	51
7.3.2.3	_dac_wait_s	51
7.3.2.4	_dac_reset_i	52
7.3.2.5	_dac_reset_s	52
7.3.2.6	_dac_wakeup_isr	52

7.3.2.7	_dac_timeout_isr	53
7.3.2.8	_dac_isr_half_code	53
7.3.2.9	_dac_isr_full_code	54
7.3.2.10	_dac_isr_error_code	54
7.3.2.11	DAC_MAX_CHANNELS	55
7.3.2.12	PLATFORM_DAC_USE_DAC1	55
7.3.3	Typedef Documentation	55
7.3.3.1	dacchannel_t	55
7.3.3.2	DACDriver	55
7.3.3.3	dacsample_t	55
7.3.3.4	daccallback_t	55
7.3.3.5	dacerrorcallback_t	55
7.3.4	Enumeration Type Documentation	56
7.3.4.1	dacstate_t	56
7.3.4.2	dacerror_t	56
7.3.5	Function Documentation	56
7.3.5.1	dacInit	56
7.3.5.2	dacObjectInit	57
7.3.5.3	dacStart	57
7.3.5.4	dacStop	58
7.3.5.5	dacPutChannelIX	58
7.3.5.6	dacStartConversion	59
7.3.5.7	dacStartConversionI	59
7.3.5.8	dacStopConversion	60
7.3.5.9	dacStopConversionI	61
7.3.5.10	dacConvert	61
7.3.5.11	dacAcquireBus	62
7.3.5.12	dacReleaseBus	63
7.3.5.13	dac_lld_init	63
7.3.5.14	dac_lld_start	63
7.3.5.15	dac_lld_stop	64
7.3.5.16	dac_lld_put_channel	64
7.3.5.17	dac_lld_start_conversion	64
7.3.5.18	dac_lld_stop_conversion	65
7.3.6	Variable Documentation	65
7.3.6.1	DACD1	65
7.4	EXT Driver	66
7.4.1	Detailed Description	66
7.4.2	Driver State Machine	66
7.4.3	EXT Operations.	66

7.4.4	Macro Definition Documentation	68
7.4.4.1	EXT_CH_MODE_EDGES_MASK	68
7.4.4.2	EXT_CH_MODE_DISABLED	68
7.4.4.3	EXT_CH_MODE_RISING_EDGE	68
7.4.4.4	EXT_CH_MODE_FALLING_EDGE	69
7.4.4.5	EXT_CH_MODE_BOTH_EDGES	69
7.4.4.6	EXT_CH_MODE_AUTOSTART	69
7.4.4.7	extChannelEnable	69
7.4.4.8	extChannelDisable	69
7.4.4.9	extSetChannelMode	69
7.4.4.10	EXT_MAX_CHANNELS	70
7.4.4.11	PLATFORM_EXT_USE_EXT1	70
7.4.5	Typedef Documentation	70
7.4.5.1	EXTDriver	70
7.4.5.2	expchannel_t	70
7.4.5.3	extcallback_t	70
7.4.6	Enumeration Type Documentation	70
7.4.6.1	extstate_t	70
7.4.7	Function Documentation	71
7.4.7.1	extInit	71
7.4.7.2	extObjectInit	71
7.4.7.3	extStart	71
7.4.7.4	extStop	72
7.4.7.5	extChannelEnable	72
7.4.7.6	extChannelDisable	73
7.4.7.7	extSetChannelModel	74
7.4.7.8	ext_lld_init	74
7.4.7.9	ext_lld_start	74
7.4.7.10	ext_lld_stop	75
7.4.7.11	ext_lld_channel_enable	75
7.4.7.12	ext_lld_channel_disable	75
7.4.8	Variable Documentation	75
7.4.8.1	EXTD1	75
7.5	GPT Driver	76
7.5.1	Detailed Description	76
7.5.2	Driver State Machine	76
7.5.3	GPT Operations.	76
7.5.4	Macro Definition Documentation	79
7.5.4.1	gptChangel intervall	79
7.5.4.2	gptGetIntervalX	79

7.5.4.3	gptGetCounterX	79
7.5.4.4	PLATFORM_GPT_USE_GPT1	81
7.5.4.5	gpt_lld_change_interval	81
7.5.5	Typedef Documentation	82
7.5.5.1	GPTDriver	82
7.5.5.2	gptcallback_t	82
7.5.5.3	gptfreq_t	82
7.5.5.4	gptcnt_t	82
7.5.6	Enumeration Type Documentation	82
7.5.6.1	gptstate_t	82
7.5.7	Function Documentation	82
7.5.7.1	gptInit	82
7.5.7.2	gptObjectInit	83
7.5.7.3	gptStart	83
7.5.7.4	gptStop	84
7.5.7.5	gptChangeInterval	85
7.5.7.6	gptStartContinuous	86
7.5.7.7	gptStartContinuousl	87
7.5.7.8	gptStartOneShot	88
7.5.7.9	gptStartOneShotl	88
7.5.7.10	gptStopTimer	89
7.5.7.11	gptStopTimerl	89
7.5.7.12	gptPolledDelay	90
7.5.7.13	gpt_lld_init	90
7.5.7.14	gpt_lld_start	91
7.5.7.15	gpt_lld_stop	91
7.5.7.16	gpt_lld_start_timer	91
7.5.7.17	gpt_lld_stop_timer	91
7.5.7.18	gpt_lld_polled_delay	92
7.5.8	Variable Documentation	92
7.5.8.1	GPTD1	92
7.6	HAL Driver	93
7.6.1	Detailed Description	93
7.6.2	Macro Definition Documentation	94
7.6.2.1	_CHIBIOS_HAL_	94
7.6.2.2	CH_HAL_STABLE	94
7.6.2.3	HAL_VERSION	94
7.6.2.4	CH_HAL_MAJOR	94
7.6.2.5	CH_HAL_MINOR	94
7.6.2.6	CH_HAL_PATCH	94

7.6.3	Function Documentation	94
7.6.3.1	hallInit	94
7.6.3.2	hal_lld_init	95
7.7	Abstract I/O Channel	96
7.7.1	Detailed Description	96
7.7.2	Macro Definition Documentation	97
7.7.2.1	_base_channel_methods	97
7.7.2.2	_base_channel_data	97
7.7.2.3	chnPutTimeout	97
7.7.2.4	chnGetTimeout	98
7.7.2.5	chnWrite	98
7.7.2.6	chnWriteTimeout	99
7.7.2.7	chnRead	99
7.7.2.8	chnReadTimeout	100
7.7.2.9	CHN_NO_ERROR	100
7.7.2.10	CHN_CONNECTED	100
7.7.2.11	CHN_DISCONNECTED	100
7.7.2.12	CHN_INPUT_AVAILABLE	100
7.7.2.13	CHN_OUTPUT_EMPTY	100
7.7.2.14	CHN_TRANSMISSION_END	100
7.7.2.15	_base_asynchronous_channel_methods	100
7.7.2.16	_base_asynchronous_channel_data	101
7.7.2.17	chnGetEventSource	101
7.7.2.18	chnAddFlags!	101
7.8	Abstract Files	102
7.8.1	Detailed Description	102
7.8.2	Macro Definition Documentation	103
7.8.2.1	FILE_OK	103
7.8.2.2	FILE_ERROR	103
7.8.2.3	FILE_EOF	103
7.8.2.4	_file_stream_methods	103
7.8.2.5	_file_stream_data	103
7.8.2.6	fileStreamWrite	103
7.8.2.7	fileStreamRead	104
7.8.2.8	fileStreamPut	104
7.8.2.9	fileStreamGet	105
7.8.2.10	fileStreamClose	105
7.8.2.11	fileStreamGetError	106
7.8.2.12	fileStreamGetSize	106
7.8.2.13	fileStreamGetPosition	106

7.8.2.14	fileStreamSeek	107
7.8.3	Typedef Documentation	107
7.8.3.1	fileoffset_t	107
7.9	Abstract I/O Block Device	108
7.9.1	Detailed Description	108
7.9.2	Driver State Machine	108
7.9.3	Macro Definition Documentation	109
7.9.3.1	_base_block_device_methods	109
7.9.3.2	_base_block_device_data	110
7.9.3.3	blkGetDriverState	110
7.9.3.4	blkIsTransferring	110
7.9.3.5	blkIsInserted	111
7.9.3.6	blkIsWriteProtected	111
7.9.3.7	blkConnect	111
7.9.3.8	blkDisconnect	112
7.9.3.9	blkRead	112
7.9.3.10	blkWrite	113
7.9.3.11	blkSync	113
7.9.3.12	blkGetInfo	113
7.9.4	Enumeration Type Documentation	114
7.9.4.1	blkstate_t	114
7.10	I/O Queues	115
7.10.1	Detailed Description	115
7.10.2	Macro Definition Documentation	117
7.10.2.1	Q_OK	117
7.10.2.2	Q_TIMEOUT	117
7.10.2.3	Q_RESET	117
7.10.2.4	Q_EMPTY	117
7.10.2.5	Q_FULL	117
7.10.2.6	qSizeX	117
7.10.2.7	qSpaceL	118
7.10.2.8	qGetLink	118
7.10.2.9	iqGetFulll	118
7.10.2.10	iqGetEmptyl	119
7.10.2.11	iqIsEmptyl	119
7.10.2.12	iqIsFulll	119
7.10.2.13	iqGet	120
7.10.2.14	_INPUTQUEUE_DATA	120
7.10.2.15	INPUTQUEUE_DECL	121
7.10.2.16	oqGetFulll	121

7.10.2.17 oqGetEmpty	121
7.10.2.18 oqlsEmpty	122
7.10.2.19 oqlsFull	122
7.10.2.20 oqPut	123
7.10.2.21 _OUTPUTQUEUE_DATA	123
7.10.2.22 OUTPUTQUEUE_DECL	123
7.10.3 Typedef Documentation	124
7.10.3.1 io_queue_t	124
7.10.3.2 qnotify_t	124
7.10.3.3 input_queue_t	124
7.10.3.4 output_queue_t	124
7.10.4 Function Documentation	124
7.10.4.1 iqObjectInit	124
7.10.4.2 iqReset	125
7.10.4.3 iqPut	126
7.10.4.4 iqGetTimeout	127
7.10.4.5 iqReadTimeout	128
7.10.4.6 oqObjectInit	129
7.10.4.7 oqReset	130
7.10.4.8 oqPutTimeout	130
7.10.4.9 oqGet	131
7.10.4.10 oqWriteTimeout	132
7.11 Abstract Streams	134
7.11.1 Detailed Description	134
7.11.2 Macro Definition Documentation	134
7.11.2.1 _base_sequential_stream_methods	134
7.11.2.2 _base_sequential_stream_data	135
7.11.2.3 streamWrite	135
7.11.2.4 streamRead	135
7.11.2.5 streamPut	135
7.11.2.6 streamGet	136
7.12 I2C Driver	137
7.12.1 Detailed Description	137
7.12.2 Driver State Machine	137
7.12.3 Macro Definition Documentation	139
7.12.3.1 I2C_NO_ERROR	139
7.12.3.2 I2C_BUS_ERROR	140
7.12.3.3 I2C_ARBITRATION_LOST	140
7.12.3.4 I2C_ACK_FAILURE	140
7.12.3.5 I2C_OVERRUN	140

7.12.3.6 I2C_PEC_ERROR	140
7.12.3.7 I2C_TIMEOUT	140
7.12.3.8 I2C_SMB_ALERT	140
7.12.3.9 I2C_USE_MUTUAL_EXCLUSION	140
7.12.3.10 _i2c_wakeup_isr	140
7.12.3.11 _i2c_wakeup_error_isr	141
7.12.3.12 i2cMasterTransmit	141
7.12.3.13 i2cMasterReceive	141
7.12.3.14 PLATFORM_I2C_USE_I2C1	141
7.12.3.15 i2c_lld_get_errors	141
7.12.4 TYPEDOC Documentation	142
7.12.4.1 i2caddr_t	142
7.12.4.2 i2cflags_t	142
7.12.4.3 I2CDriver	142
7.12.5 ENUMERATION Documentation	142
7.12.5.1 i2cstate_t	142
7.12.6 FUNCTION Documentation	142
7.12.6.1 i2cInit	142
7.12.6.2 i2cObjectInit	143
7.12.6.3 i2cStart	143
7.12.6.4 i2cStop	144
7.12.6.5 i2cGetErrors	144
7.12.6.6 i2cMasterTransmitTimeout	145
7.12.6.7 i2cMasterReceiveTimeout	146
7.12.6.8 i2cAcquireBus	147
7.12.6.9 i2cReleaseBus	147
7.12.6.10 i2c_lld_init	148
7.12.6.11 i2c_lld_start	148
7.12.6.12 i2c_lld_stop	148
7.12.6.13 i2c_lld_master_receive_timeout	149
7.12.6.14 i2c_lld_master_transmit_timeout	149
7.12.7 VARIABLE Documentation	150
7.12.7.1 I2CD1	150
7.13 I2S Driver	151
7.13.1 Detailed Description	151
7.13.2 Driver State Machine	151
7.13.3 Macro Definition Documentation	152
7.13.3.1 i2sStartExchangel	152
7.13.3.2 i2sStopExchangel	153
7.13.3.3 _i2s_isr_half_code	153

7.13.3.4	_i2s_isr_full_code	153
7.13.3.5	PLATFORM_I2S_USE_I2S1	154
7.13.4	Typedef Documentation	154
7.13.4.1	I2SDriver	154
7.13.4.2	i2scallback_t	154
7.13.5	Enumeration Type Documentation	154
7.13.5.1	i2sstate_t	154
7.13.6	Function Documentation	155
7.13.6.1	i2sInit	155
7.13.6.2	i2sObjectInit	155
7.13.6.3	i2sStart	155
7.13.6.4	i2sStop	156
7.13.6.5	i2sStartExchange	156
7.13.6.6	i2sStopExchange	157
7.13.6.7	i2s_lld_init	157
7.13.6.8	i2s_lld_start	158
7.13.7	Variable Documentation	158
7.13.7.1	I2SD1	158
7.14	ICU Driver	159
7.14.1	Detailed Description	159
7.14.2	Driver State Machine	159
7.14.3	ICU Operations	159
7.14.4	Macro Definition Documentation	162
7.14.4.1	icuStartCaptureI	162
7.14.4.2	icuStopCaptureI	162
7.14.4.3	icuEnableNotificationsI	163
7.14.4.4	icuDisableNotificationsI	163
7.14.4.5	icuAreNotificationsEnabledX	163
7.14.4.6	icuGetWidthX	164
7.14.4.7	icuGetPeriodX	164
7.14.4.8	_icu_isr_invoke_width_cb	165
7.14.4.9	_icu_isr_invoke_period_cb	165
7.14.4.10	_icu_isr_invoke_overflow_cb	165
7.14.4.11	PLATFORM_ICU_USE_ICU1	166
7.14.4.12	icu_lld_get_width	166
7.14.4.13	icu_lld_get_period	166
7.14.4.14	icu_lld_are_notifications_enabled	166
7.14.5	Typedef Documentation	167
7.14.5.1	ICUDriver	167
7.14.5.2	icucallback_t	167

7.14.5.3	icufreq_t	167
7.14.5.4	icucnt_t	167
7.14.6	Enumeration Type Documentation	167
7.14.6.1	icustate_t	167
7.14.6.2	icumode_t	168
7.14.7	Function Documentation	168
7.14.7.1	icuInit	168
7.14.7.2	icuObjectInit	168
7.14.7.3	icuStart	168
7.14.7.4	icuStop	169
7.14.7.5	icuStartCapture	170
7.14.7.6	icuWaitCapture	171
7.14.7.7	icuStopCapture	172
7.14.7.8	icuEnableNotifications	172
7.14.7.9	icuDisableNotifications	173
7.14.7.10	icu_lld_init	174
7.14.7.11	icu_lld_start	174
7.14.7.12	icu_lld_stop	174
7.14.7.13	icu_lld_start_capture	175
7.14.7.14	icu_lld_wait_capture	175
7.14.7.15	icu_lld_stop_capture	175
7.14.7.16	icu_lld_enable_notifications	176
7.14.7.17	icu_lld_disable_notifications	176
7.14.8	Variable Documentation	176
7.14.8.1	ICUD1	176
7.15	MAC Driver	177
7.15.1	Detailed Description	177
7.15.2	Macro Definition Documentation	179
7.15.2.1	MAC_USE_ZERO_COPY	179
7.15.2.2	MAC_USE_EVENTS	179
7.15.2.3	macGetReceiveEventSource	179
7.15.2.4	macWriteTransmitDescriptor	179
7.15.2.5	macReadReceiveDescriptor	180
7.15.2.6	macGetNextTransmitBuffer	180
7.15.2.7	macGetNextReceiveBuffer	181
7.15.2.8	MAC_SUPPORTS_ZERO_COPY	181
7.15.2.9	PLATFORM_MAC_USE_MAC1	181
7.15.3	Typedef Documentation	181
7.15.3.1	MACDriver	181
7.15.4	Enumeration Type Documentation	181

7.15.4.1	macstate_t	181
7.15.5	Function Documentation	182
7.15.5.1	macInit	182
7.15.5.2	macObjectInit	182
7.15.5.3	macStart	182
7.15.5.4	macStop	183
7.15.5.5	macWaitTransmitDescriptor	184
7.15.5.6	macReleaseTransmitDescriptor	185
7.15.5.7	macWaitReceiveDescriptor	186
7.15.5.8	macReleaseReceiveDescriptor	187
7.15.5.9	macPollLinkStatus	188
7.15.5.10	mac_lld_init	188
7.15.5.11	mac_lld_start	189
7.15.5.12	mac_lld_stop	189
7.15.5.13	mac_lld_get_transmit_descriptor	189
7.15.5.14	mac_lld_release_transmit_descriptor	190
7.15.5.15	mac_lld_get_receive_descriptor	190
7.15.5.16	mac_lld_release_receive_descriptor	190
7.15.5.17	mac_lld_poll_link_status	190
7.15.5.18	mac_lld_write_transmit_descriptor	191
7.15.5.19	mac_lld_read_receive_descriptor	191
7.15.5.20	mac_lld_get_next_transmit_buffer	192
7.15.5.21	mac_lld_get_next_receive_buffer	192
7.15.6	Variable Documentation	192
7.15.6.1	ETHD1	192
7.16	HAL	193
7.16.1	Detailed Description	193
7.16.2	HAL Device Drivers Architecture	193
7.16.3	HAL Normal Device Drivers	193
7.16.3.1	Diagram	194
7.16.4	HAL Complex Device Drivers	194
7.16.5	HAL Interfaces	194
7.16.6	HAL Inner Code	194
7.17	Configuration	196
7.17.1	Detailed Description	196
7.17.2	Macro Definition Documentation	198
7.17.2.1	HAL_USE_PAL	198
7.17.2.2	HAL_USE_ADC	198
7.17.2.3	HAL_USE_CAN	198
7.17.2.4	HAL_USE_DAC	198

7.17.2.5 HAL_USE_EXT	198
7.17.2.6 HAL_USE_GPT	198
7.17.2.7 HAL_USE_I2C	198
7.17.2.8 HAL_USE_I2S	198
7.17.2.9 HAL_USE_ICU	198
7.17.2.10 HAL_USE_MAC	198
7.17.2.11 HAL_USE_MMC_SPI	199
7.17.2.12 HAL_USE_PWM	199
7.17.2.13 HAL_USE_RTC	199
7.17.2.14 HAL_USE_SDC	199
7.17.2.15 HAL_USE_SERIAL	199
7.17.2.16 HAL_USE_SERIAL_USB	199
7.17.2.17 HAL_USE_SPI	199
7.17.2.18 HAL_USE_UART	199
7.17.2.19 HAL_USE_USB	199
7.17.2.20 ADC_USE_WAIT	199
7.17.2.21 ADC_USE_MUTUAL_EXCLUSION	199
7.17.2.22 CAN_USE_SLEEP_MODE	200
7.17.2.23 I2C_USE_MUTUAL_EXCLUSION	200
7.17.2.24 MAC_USE_ZERO_COPY	200
7.17.2.25 MAC_USE_EVENTS	200
7.17.2.26 MMC_NICE_WAITING	200
7.17.2.27 SDC_INIT_RETRY	200
7.17.2.28 SDC_MMC_SUPPORT	200
7.17.2.29 SDC_NICE_WAITING	200
7.17.2.30 SERIAL_DEFAULT_BITRATE	200
7.17.2.31 SERIAL_BUFFERS_SIZE	201
7.17.2.32 SERIAL_USB_BUFFERS_SIZE	201
7.17.2.33 SPI_USE_WAIT	201
7.17.2.34 SPI_USE_MUTUAL_EXCLUSION	201
7.18 Normal Drivers	202
7.18.1 Detailed Description	202
7.19 Complex Drivers	203
7.19.1 Detailed Description	203
7.20 Interfaces	204
7.20.1 Detailed Description	204
7.21 Inner Code	205
7.21.1 Detailed Description	205
7.22 OSAL	206
7.22.1 Detailed Description	206

7.22.2 Macro Definition Documentation	211
7.22.2.1 OSAL_ST_RESOLUTION	211
7.22.2.2 OSAL_ST_FREQUENCY	211
7.22.2.3 OSAL_ST_MODE	211
7.22.2.4 OSAL_IRQ_PRIORITY_LEVELS	211
7.22.2.5 OSAL_IRQ_MAXIMUM_PRIORITY	211
7.22.2.6 OSAL_DBG_ENABLE_ASSERTS	211
7.22.2.7 OSAL_DBG_ENABLE_CHECKS	211
7.22.2.8 osalDbgAssert	211
7.22.2.9 osalDbgCheck	212
7.22.2.10 osalDbgCheckClassI	212
7.22.2.11 osalDbgCheckClassS	212
7.22.2.12 OSAL_IRQ_IS_VALID_PRIORITY	213
7.22.2.13 OSAL_IRQ_PROLOGUE	213
7.22.2.14 OSAL_IRQ_EPILOGUE	213
7.22.2.15 OSAL_IRQ_HANDLER	213
7.22.2.16 OSAL_S2ST	213
7.22.2.17 OSAL_MS2ST	213
7.22.2.18 OSAL_US2ST	214
7.22.2.19 OSAL_S2RTC	214
7.22.2.20 OSAL_MS2RTC	215
7.22.2.21 OSAL_US2RTC	215
7.22.2.22 osalThreadSleepSeconds	216
7.22.2.23 osalThreadSleepMilliseconds	216
7.22.2.24 osalThreadSleepMicroseconds	216
7.22.3 Typedef Documentation	216
7.22.3.1 syssts_t	216
7.22.3.2 msg_t	217
7.22.3.3 systime_t	217
7.22.3.4 rtcnt_t	217
7.22.3.5 thread_reference_t	217
7.22.3.6 event_source_t	217
7.22.3.7 eventcallback_t	217
7.22.3.8 eventflags_t	217
7.22.3.9 mutex_t	217
7.22.4 Function Documentation	217
7.22.4.1 osallInit	217
7.22.4.2 osalSysHalt	218
7.22.4.3 osalSysPolledDelayX	218
7.22.4.4 osalOsTimerHandlerI	218

7.22.4.5 osalOsRescheduleS	219
7.22.4.6 osalOsGetSystemTimeX	219
7.22.4.7 osalThreadSleepS	219
7.22.4.8 osalThreadSleep	219
7.22.4.9 osalThreadSuspendS	220
7.22.4.10 osalThreadSuspendTimeoutS	220
7.22.4.11 osalThreadResumel	221
7.22.4.12 osalThreadResumeS	221
7.22.4.13 osalThreadEnqueueTimeoutS	221
7.22.4.14 osalThreadDequeueNextl	222
7.22.4.15 osalThreadDequeueAlll	222
7.22.4.16 osalEventBroadcastFlagsl	222
7.22.4.17 osalEventBroadcastFlagsl	223
7.22.4.18 osalEventSetCallback	223
7.22.4.19 osalMutexLock	224
7.22.4.20 osalMutexUnlock	224
7.22.4.21 osalSysDisable	224
7.22.4.22 osalSysEnable	225
7.22.4.23 osalSysLock	225
7.22.4.24 osalSysUnlock	225
7.22.4.25 osalSysLockFromISR	225
7.22.4.26 osalSysUnlockFromISR	225
7.22.4.27 osalSysGetStatusAndLockX	226
7.22.4.28 osalSysRestoreStatusX	226
7.22.4.29 osalOsIsTimeWithinX	226
7.22.4.30 osalThreadQueueObjectInit	227
7.22.4.31 osalEventObjectInit	227
7.22.4.32 osalMutexObjectInit	227
7.22.5 Variable Documentation	227
7.22.5.1 osal_halt_msg	227
7.22.5.2 osal_halt_msg	228
7.23 MMC over SPI Driver	229
7.23.1 Detailed Description	229
7.23.2 Driver State Machine	229
7.23.3 Driver Operations	229
7.23.4 Macro Definition Documentation	231
7.23.4.1 MMC_NICE_WAITING	231
7.23.4.2 _mmc_driver_methods	231
7.23.4.3 mmclsCardInserted	231
7.23.4.4 mmclsWriteProtected	231

7.23.5 Function Documentation	232
7.23.5.1 crc7	232
7.23.5.2 wait	232
7.23.5.3 send_hdr	233
7.23.5.4 recv1	233
7.23.5.5 recv3	234
7.23.5.6 send_command_R1	235
7.23.5.7 send_command_R3	235
7.23.5.8 read_CxD	236
7.23.5.9 sync	237
7.23.5.10 mmcInit	238
7.23.5.11 mmcObjectInit	238
7.23.5.12 mmcStart	238
7.23.5.13 mmcStop	238
7.23.5.14 mmcConnect	239
7.23.5.15 mmcDisconnect	240
7.23.5.16 mmcStartSequentialRead	241
7.23.5.17 mmcSequentialRead	242
7.23.5.18 mmcStopSequentialRead	243
7.23.5.19 mmcStartSequentialWrite	244
7.23.5.20 mmcSequentialWrite	245
7.23.5.21 mmcStopSequentialWrite	246
7.23.5.22 mmcSync	247
7.23.5.23 mmcGetInfo	248
7.23.5.24 mmcErase	248
7.23.6 Variable Documentation	249
7.23.6.1 mmc_vmt	249
7.23.6.2 crc7_lookup_table	249
7.24 MMC/SD Block Device	251
7.24.1 Detailed Description	251
7.24.2 Macro Definition Documentation	255
7.24.2.1 MMCSD_BLOCK_SIZE	255
7.24.2.2 MMCSD_R1_ERROR_MASK	255
7.24.2.3 MMCSD_CMD8_PATTERN	255
7.24.2.4 MMCSD_CSD_MMC_CSD_STRUCTURE_SLICE	255
7.24.2.5 MMCSD_CID_SDC_CRC_SLICE	255
7.24.2.6 _mmcisd_block_device_methods	256
7.24.2.7 _mmcisd_block_device_data	256
7.24.2.8 MMCSD_R1_ERROR	256
7.24.2.9 MMCSD_R1_STS	256

7.24.2.10 MMCSD_R1_IS_CARD_LOCKED	256
7.24.2.11 mmcisdGetCardCapacity	256
7.24.3 Function Documentation	257
7.24.3.1 _mmcisd_get_slice	257
7.24.3.2 _mmcisd_get_capacity	257
7.24.3.3 _mmcisd_get_capacity_ext	258
7.24.3.4 _mmcisd_unpack_sdc_cid	258
7.24.3.5 _mmcisd_unpack_mmc_cid	258
7.24.3.6 _mmcisd_unpack_csd_mmc	259
7.24.3.7 _mmcisd_unpack_csd_v10	259
7.24.3.8 _mmcisd_unpack_csd_v20	260
7.25 PAL Driver	261
7.25.1 Detailed Description	261
7.25.2 Implementation Rules	261
7.25.2.1 Writing on input pads	261
7.25.2.2 Reading from output pads	261
7.25.2.3 Writing unused or unimplemented port bits	261
7.25.2.4 Reading from unused or unimplemented port bits	262
7.25.2.5 Reading or writing on pins associated to other functionalities	262
7.25.3 Macro Definition Documentation	265
7.25.3.1 PAL_MODE_RESET	265
7.25.3.2 PAL_MODE_UNCONNECTED	265
7.25.3.3 PAL_MODE_INPUT	265
7.25.3.4 PAL_MODE_INPUT_PULLUP	265
7.25.3.5 PAL_MODE_INPUT_PULLDOWN	265
7.25.3.6 PAL_MODE_INPUT_ANALOG	265
7.25.3.7 PAL_MODE_OUTPUT_PUSH_PULL	265
7.25.3.8 PAL_MODE_OUTPUT_OPENDRAIN	265
7.25.3.9 PAL_LOW	265
7.25.3.10 PAL_HIGH	265
7.25.3.11 PAL_PORT_BIT	266
7.25.3.12 PAL_GROUP_MASK	267
7.25.3.13 _IOBUS_DATA	267
7.25.3.14 IOBUS_DECL	267
7.25.3.15 pallnit	267
7.25.3.16 palReadPort	268
7.25.3.17 palReadLatch	268
7.25.3.18 palWritePort	268
7.25.3.19 palSetPort	269
7.25.3.20 palClearPort	269

7.25.3.21	palTogglePort	269
7.25.3.22	palReadGroup	270
7.25.3.23	palWriteGroup	270
7.25.3.24	palSetGroupMode	271
7.25.3.25	palReadPad	271
7.25.3.26	palWritePad	272
7.25.3.27	palSetPad	272
7.25.3.28	palClearPad	273
7.25.3.29	palTogglePad	273
7.25.3.30	palSetPadMode	273
7.25.3.31	PAL_IOPORTS_WIDTH	274
7.25.3.32	PAL_WHOLE_PORT	274
7.25.3.33	IOPORT1	274
7.25.3.34	pal_lld_init	274
7.25.3.35	pal_lld_readport	274
7.25.3.36	pal_lld_readdlatch	275
7.25.3.37	pal_lld_writeport	275
7.25.3.38	pal_lld_setport	275
7.25.3.39	pal_lld_clearport	276
7.25.3.40	pal_lld_toggleport	276
7.25.3.41	pal_lld_readgroup	277
7.25.3.42	pal_lld_writegroup	277
7.25.3.43	pal_lld_setgroupmode	278
7.25.3.44	pal_lld_readpad	278
7.25.3.45	pal_lld_writepad	279
7.25.3.46	pal_lld_setpad	279
7.25.3.47	pal_lld_clearpad	279
7.25.3.48	pal_lld_togglepad	280
7.25.3.49	pal_lld_setpadmode	280
7.25.4	Typedef Documentation	281
7.25.4.1	ioportmask_t	281
7.25.4.2	iomode_t	281
7.25.4.3	ioportid_t	281
7.25.5	Function Documentation	281
7.25.5.1	palReadBus	281
7.25.5.2	palWriteBus	282
7.25.5.3	palSetBusMode	282
7.25.5.4	_pal_lld_init	282
7.25.5.5	_pal_lld_setgroupmode	283
7.26	PWM Driver	285

7.26.1	Detailed Description	285
7.26.2	Driver State Machine	285
7.26.3	PWM Operations.	285
7.26.4	Macro Definition Documentation	288
7.26.4.1	PWM_OUTPUT_MASK	288
7.26.4.2	PWM_OUTPUT_DISABLED	288
7.26.4.3	PWM_OUTPUT_ACTIVE_HIGH	288
7.26.4.4	PWM_OUTPUT_ACTIVE_LOW	288
7.26.4.5	PWM_FRACTION_TO_WIDTH	289
7.26.4.6	PWM_DEGREES_TO_WIDTH	289
7.26.4.7	PWM_PERCENTAGE_TO_WIDTH	289
7.26.4.8	pwmChangePeriodl	290
7.26.4.9	pwmEnableChannell	290
7.26.4.10	pwmDisableChannell	291
7.26.4.11	pwmlsChannelEnabledl	292
7.26.4.12	pwmEnablePeriodicNotificationl	292
7.26.4.13	pwmDisablePeriodicNotificationl	292
7.26.4.14	pwmEnableChannelNotificationl	293
7.26.4.15	pwmDisableChannelNotificationl	293
7.26.4.16	PWM_CHANNELS	294
7.26.4.17	PLATFORM_PWM_USE_PWM1	294
7.26.4.18	pwm_lld_change_period	294
7.26.5	Typedef Documentation	294
7.26.5.1	PWMDriver	294
7.26.5.2	pwmcallback_t	294
7.26.5.3	pwmmode_t	295
7.26.5.4	pwmcchannel_t	295
7.26.5.5	pwmcchnmsk_t	295
7.26.5.6	pwmcnt_t	295
7.26.6	Enumeration Type Documentation	295
7.26.6.1	pwmstate_t	295
7.26.7	Function Documentation	295
7.26.7.1	pwmInit	295
7.26.7.2	pwmObjectInit	296
7.26.7.3	pwmStart	296
7.26.7.4	pwmStop	297
7.26.7.5	pwmChangePeriod	297
7.26.7.6	pwmEnableChannel	298
7.26.7.7	pwmDisableChannel	299
7.26.7.8	pwmEnablePeriodicNotification	300

7.26.7.9	pwmDisablePeriodicNotification	300
7.26.7.10	pwmEnableChannelNotification	301
7.26.7.11	pwmDisableChannelNotification	302
7.26.7.12	pwm_lld_init	302
7.26.7.13	pwm_lld_start	303
7.26.7.14	pwm_lld_stop	303
7.26.7.15	pwm_lld_enable_channel	303
7.26.7.16	pwm_lld_disable_channel	304
7.26.7.17	pwm_lld_enable_periodic_notification	304
7.26.7.18	pwm_lld_disable_periodic_notification	305
7.26.7.19	pwm_lld_enable_channel_notification	305
7.26.7.20	pwm_lld_disable_channel_notification	305
7.26.8	Variable Documentation	306
7.26.8.1	PWMD1	306
7.27	RTC Driver	307
7.27.1	Detailed Description	307
7.27.2	Macro Definition Documentation	309
7.27.2.1	RTC_BASE_YEAR	309
7.27.2.2	RTC_SUPPORTS_CALLBACKS	309
7.27.2.3	RTC_ALARMS	309
7.27.2.4	RTC_HAS_STORAGE	309
7.27.2.5	PLATFORM_RTC_USE_RTC1	309
7.27.2.6	_rtc_driver_methods	309
7.27.3	Typedef Documentation	310
7.27.3.1	RTCDriver	310
7.27.3.2	rtcalarm_t	310
7.27.3.3	rtccb_t	310
7.27.4	Enumeration Type Documentation	310
7.27.4.1	rtcevent_t	310
7.27.5	Function Documentation	310
7.27.5.1	rtcInit	310
7.27.5.2	rtcObjectInit	310
7.27.5.3	rtcSetTime	311
7.27.5.4	rtcGetTime	311
7.27.5.5	rtcSetAlarm	312
7.27.5.6	rtcGetAlarm	312
7.27.5.7	rtcSetCallback	313
7.27.5.8	rtcConvertDateTimeToStructTm	313
7.27.5.9	rtcConvertStructTmToDateTm	314
7.27.5.10	rtcConvertDateTimeToFAT	314

7.27.5.11 <code>rtc_lld_init</code>	314
7.27.5.12 <code>rtc_lld_set_time</code>	315
7.27.5.13 <code>rtc_lld_get_time</code>	315
7.27.5.14 <code>rtc_lld_set_alarm</code>	315
7.27.5.15 <code>rtc_lld_get_alarm</code>	316
7.27.6 Variable Documentation	316
7.27.6.1 <code>RTCD1</code>	316
7.28 SDC Driver	317
7.28.1 Detailed Description	317
7.28.2 Driver State Machine	317
7.28.3 Driver Operations	317
7.28.4 Macro Definition Documentation	320
7.28.4.1 <code>SDC_INIT_RETRY</code>	320
7.28.4.2 <code>SDC_MMC_SUPPORT</code>	321
7.28.4.3 <code>SDC_NICE_WAITING</code>	321
7.28.4.4 <code>sdclsCardInserted</code>	321
7.28.4.5 <code>sdclsWriteProtected</code>	321
7.28.4.6 <code>PLATFORM_SDC_USE_SDC1</code>	322
7.28.4.7 <code>_sdc_driver_methods</code>	322
7.28.5 Typedef Documentation	322
7.28.5.1 <code>sdcmode_t</code>	322
7.28.5.2 <code>sdcflags_t</code>	322
7.28.5.3 <code>SDCDriver</code>	322
7.28.6 Enumeration Type Documentation	322
7.28.6.1 <code>mmc_switch_t</code>	322
7.28.6.2 <code>sd_switch_t</code>	322
7.28.6.3 <code>sd_switch_function_t</code>	323
7.28.6.4 <code>sdcbusmode_t</code>	323
7.28.6.5 <code>sdcbusclk_t</code>	323
7.28.7 Function Documentation	323
7.28.7.1 <code>mode_detect</code>	323
7.28.7.2 <code>mmc_init</code>	323
7.28.7.3 <code>sdc_init</code>	324
7.28.7.4 <code>mmc_cmd6_construct</code>	325
7.28.7.5 <code>sdc_cmd6_construct</code>	325
7.28.7.6 <code>sdc_cmd6_extract_info</code>	326
7.28.7.7 <code>sdc_cmd6_check_status</code>	326
7.28.7.8 <code>sdc_detect_bus_clk</code>	326
7.28.7.9 <code>mmc_detect_bus_clk</code>	327
7.28.7.10 <code>detect_bus_clk</code>	328

7.28.7.11 <code>sdc_set_bus_width</code>	329
7.28.7.12 <code>mmc_set_bus_width</code>	330
7.28.7.13 <code>_sdc_wait_for_transfer_state</code>	331
7.28.7.14 <code>sdcInit</code>	332
7.28.7.15 <code>sdcObjectInit</code>	332
7.28.7.16 <code>sdcStart</code>	332
7.28.7.17 <code>sdcStop</code>	333
7.28.7.18 <code>sdcConnect</code>	334
7.28.7.19 <code>sdcDisconnect</code>	335
7.28.7.20 <code>sdcRead</code>	336
7.28.7.21 <code>sdcWrite</code>	337
7.28.7.22 <code>sdcGetAndClearErrors</code>	338
7.28.7.23 <code>sdcSync</code>	338
7.28.7.24 <code>sdcGetInfo</code>	339
7.28.7.25 <code>sdcErase</code>	339
7.28.7.26 <code>sdc_lld_init</code>	340
7.28.7.27 <code>sdc_lld_start</code>	340
7.28.7.28 <code>sdc_lld_stop</code>	340
7.28.7.29 <code>sdc_lld_start_clk</code>	341
7.28.7.30 <code>sdc_lld_set_data_clk</code>	341
7.28.7.31 <code>sdc_lld_stop_clk</code>	341
7.28.7.32 <code>sdc_lld_set_bus_mode</code>	341
7.28.7.33 <code>sdc_lld_send_cmd_none</code>	342
7.28.7.34 <code>sdc_lld_send_cmd_short</code>	342
7.28.7.35 <code>sdc_lld_send_cmd_short_crc</code>	342
7.28.7.36 <code>sdc_lld_send_cmd_long_crc</code>	343
7.28.7.37 <code>sdc_lld_read</code>	343
7.28.7.38 <code>sdc_lld_write</code>	344
7.28.7.39 <code>sdc_lld_sync</code>	344
7.28.8 Variable Documentation	345
7.28.8.1 <code>sdc_vmt</code>	345
7.28.8.2 <code>SDCD1</code>	345
7.29 Serial Driver	346
7.29.1 Detailed Description	346
7.29.2 Driver State Machine	346
7.29.3 Macro Definition Documentation	348
7.29.3.1 <code>SD_PARITY_ERROR</code>	348
7.29.3.2 <code>SD_FRAMING_ERROR</code>	349
7.29.3.3 <code>SD_OVERRUN_ERROR</code>	349
7.29.3.4 <code>SD_NOISE_ERROR</code>	349

7.29.3.5	SD_BREAK_DETECTED	349
7.29.3.6	SERIAL_DEFAULT_BITRATE	349
7.29.3.7	SERIAL_BUFFERS_SIZE	349
7.29.3.8	_serial_driver_methods	349
7.29.3.9	sdPut	349
7.29.3.10	sdPutTimeout	350
7.29.3.11	sdGet	350
7.29.3.12	sdGetTimeout	350
7.29.3.13	sdWrite	351
7.29.3.14	sdWriteTimeout	351
7.29.3.15	sdAsynchronousWrite	351
7.29.3.16	sdRead	352
7.29.3.17	sdReadTimeout	352
7.29.3.18	sdAsynchronousRead	352
7.29.3.19	PLATFORM_SERIAL_USE_USART1	352
7.29.3.20	_serial_driver_data	353
29.4	Typedef Documentation	353
29.4.1	SerialDriver	353
29.5	Enumeration Type Documentation	353
29.5.1	sdstate_t	353
29.6	Function Documentation	353
29.6.1	sdInit	353
29.6.2	sdObjectInit	354
29.6.3	sdStart	354
29.6.4	sdStop	355
29.6.5	sdIncomingData	356
29.6.6	sdRequestData	356
29.6.7	sdPutWouldBlock	357
29.6.8	sdGetWouldBlock	358
29.6.9	sd_lld_init	359
29.6.10	sd_lld_start	359
29.6.11	sd_lld_stop	359
29.7	Variable Documentation	359
29.7.1	SD1	359
29.7.2	default_config	360
30	Serial over USB Driver	361
30.1	Detailed Description	361
30.2	Driver State Machine	361
30.3	Macro Definition Documentation	364
30.3.1	SERIAL_USB_BUFFERS_SIZE	364

7.30.3.2 _serial_usb_driver_data	364
7.30.3.3 _serial_usb_driver_methods	364
7.30.4 TYPEDOC Documentation	364
7.30.4.1 SerialUSBDriver	364
7.30.5 ENUMERATION Documentation	364
7.30.5.1 sdustate_t	364
7.30.6 FUNCTION Documentation	364
7.30.6.1 inotify	364
7.30.6.2 onotify	365
7.30.6.3 sduInit	365
7.30.6.4 sduObjectInit	366
7.30.6.5 sduStart	366
7.30.6.6 sduStop	367
7.30.6.7 sduConfigureHookI	368
7.30.6.8 sduRequestsHook	368
7.30.6.9 sduDataTransmitted	369
7.30.6.10 sduDataReceived	369
7.30.6.11 sduInterruptTransmitted	370
7.31 SPI Driver	371
7.31.1 DETAILED DESCRIPTION	371
7.31.2 DRIVER STATE MACHINE	371
7.31.3 MACRO DEFINITION Documentation	374
7.31.3.1 SPI_USE_WAIT	374
7.31.3.2 SPI_USE_MUTUAL_EXCLUSION	374
7.31.3.3 spiSelectl	374
7.31.3.4 spiUnselectl	374
7.31.3.5 spiStartIgnoreI	375
7.31.3.6 spiStartExchangeI	375
7.31.3.7 spiStartSendI	376
7.31.3.8 spiStartReceiveI	377
7.31.3.9 spiPolledExchange	377
7.31.3.10 _spi_wakeup_isr	378
7.31.3.11 _spi_isr_code	378
7.31.3.12 PLATFORM_SPI_USE_SPI1	379
7.31.4 TYPEDOC Documentation	379
7.31.4.1 SPIDriver	379
7.31.4.2 spicallback_t	379
7.31.5 ENUMERATION Documentation	379
7.31.5.1 spistate_t	379
7.31.6 FUNCTION Documentation	379

7.31.6.1	spilinit	379
7.31.6.2	spiObjectInit	380
7.31.6.3	spiStart	380
7.31.6.4	spiStop	381
7.31.6.5	spiSelect	382
7.31.6.6	spiUnselect	383
7.31.6.7	spiStartIgnore	383
7.31.6.8	spiStartExchange	384
7.31.6.9	spiStartSend	385
7.31.6.10	spiStartReceive	386
7.31.6.11	spilgnore	387
7.31.6.12	spiExchange	387
7.31.6.13	spiSend	388
7.31.6.14	spiReceive	389
7.31.6.15	spiAcquireBus	390
7.31.6.16	spiReleaseBus	390
7.31.6.17	spi_lld_init	391
7.31.6.18	spi_lld_start	391
7.31.6.19	spi_lld_stop	391
7.31.6.20	spi_lld_select	392
7.31.6.21	spi_lld_unselect	392
7.31.6.22	spi_lld_ignore	392
7.31.6.23	spi_lld_exchange	392
7.31.6.24	spi_lld_send	393
7.31.6.25	spi_lld_receive	393
7.31.6.26	spi_lld_polled_exchange	394
7.31.7	Variable Documentation	394
7.31.7.1	SPID1	394
7.32	ST Driver	395
7.32.1	Detailed Description	395
7.32.2	Macro Definition Documentation	395
7.32.2.1	stGetCounter	395
7.32.2.2	stIsAlarmActive	396
7.32.3	Function Documentation	396
7.32.3.1	stInit	396
7.32.3.2	stStartAlarm	397
7.32.3.3	stStopAlarm	397
7.32.3.4	stSetAlarm	397
7.32.3.5	stGetAlarm	398
7.32.3.6	st_lld_init	398

7.32.3.7	st_lld_get_counter	399
7.32.3.8	st_lld_start_alarm	399
7.32.3.9	st_lld_stop_alarm	399
7.32.3.10	st_lld_set_alarm	399
7.32.3.11	st_lld_get_alarm	399
7.32.3.12	st_lld_is_alarm_active	400
7.33	UART Driver	401
7.33.1	Detailed Description	401
7.33.2	Driver State Machine	401
7.33.2.1	Transmitter sub State Machine	402
7.33.2.2	Receiver sub State Machine	402
7.33.3	Macro Definition Documentation	404
7.33.3.1	UART_NO_ERROR	404
7.33.3.2	UART_PARITY_ERROR	405
7.33.3.3	UART_FRAMING_ERROR	405
7.33.3.4	UART_OVERRUN_ERROR	405
7.33.3.5	UART_NOISE_ERROR	405
7.33.3.6	UART_BREAK_DETECTED	405
7.33.3.7	PLATFORM_UART_USE_UART1	405
7.33.4	Typedef Documentation	405
7.33.4.1	uartflags_t	405
7.33.4.2	UARTDriver	405
7.33.4.3	uartcb_t	405
7.33.4.4	uartccb_t	405
7.33.4.5	uarteccb_t	406
7.33.5	Enumeration Type Documentation	406
7.33.5.1	uartstate_t	406
7.33.5.2	uarttxstate_t	406
7.33.5.3	uartrxstate_t	406
7.33.6	Function Documentation	406
7.33.6.1	uartInit	406
7.33.6.2	uartObjectInit	407
7.33.6.3	uartStart	407
7.33.6.4	uartStop	408
7.33.6.5	uartStartSend	408
7.33.6.6	uartStartSendl	409
7.33.6.7	uartStopSend	410
7.33.6.8	uartStopSendl	410
7.33.6.9	uartStartReceive	411
7.33.6.10	uartStartReceive1	412

7.33.6.11 uartStopReceive	412
7.33.6.12 uartStopReceive1	413
7.33.6.13 uart_lld_init	414
7.33.6.14 uart_lld_start	414
7.33.6.15 uart_lld_stop	414
7.33.6.16 uart_lld_start_send	415
7.33.6.17 uart_lld_stop_send	415
7.33.6.18 uart_lld_start_receive	415
7.33.6.19 uart_lld_stop_receive	416
7.33.7 Variable Documentation	416
7.33.7.1 UARTD1	416
7.34 USB Driver	417
7.34.1 Detailed Description	417
7.34.2 Driver State Machine	417
7.34.3 USB Operations	417
7.34.3.1 USB Implementation	417
7.34.3.2 USB Endpoints	418
7.34.3.3 USB Callbacks	419
7.34.4 Macro Definition Documentation	424
7.34.4.1 USB_DESC_INDEX	424
7.34.4.2 USB_DESC_BYTE	424
7.34.4.3 USB_DESC_WORD	424
7.34.4.4 USB_DESC_BCD	424
7.34.4.5 USB_DESC_DEVICE	424
7.34.4.6 USB_DESC_CONFIGURATION_SIZE	425
7.34.4.7 USB_DESC_CONFIGURATION	425
7.34.4.8 USB_DESC_INTERFACE_SIZE	425
7.34.4.9 USB_DESC_INTERFACE	425
7.34.4.10 USB_DESC_INTERFACE_ASSOCIATION_SIZE	426
7.34.4.11 USB_DESC_INTERFACE_ASSOCIATION	426
7.34.4.12 USB_DESC_ENDPOINT_SIZE	426
7.34.4.13 USB_DESC_ENDPOINT	426
7.34.4.14 USB_EP_MODE_TYPE	426
7.34.4.15 USB_EP_MODE_TYPE_CTRL	427
7.34.4.16 USB_EP_MODE_TYPE_ISOC	427
7.34.4.17 USB_EP_MODE_TYPE_BULK	427
7.34.4.18 USB_EP_MODE_TYPE_INTR	427
7.34.4.19 USB_EP_MODE_LINEAR_BUFFER	427
7.34.4.20 USB_EP_MODE_QUEUE_BUFFER	427
7.34.4.21 usbGetDriverStatel	427

7.34.4.22 <code>usbConnectBus</code>	427
7.34.4.23 <code>usbDisconnectBus</code>	428
7.34.4.24 <code>usbGetFrameNumber</code>	429
7.34.4.25 <code>usbGetTransmitStatus</code>	429
7.34.4.26 <code>usbGetReceiveStatus</code>	429
7.34.4.27 <code>usbGetReceiveTransactionSize</code>	430
7.34.4.28 <code>usbSetupTransfer</code>	430
7.34.4.29 <code>usbReadSetup</code>	431
7.34.4.30 <code>_usb_isr_invoke_event_cb</code>	431
7.34.4.31 <code>_usb_isr_invoke_sof_cb</code>	432
7.34.4.32 <code>_usb_isr_invoke_setup_cb</code>	433
7.34.4.33 <code>_usb_isr_invoke_in_cb</code>	433
7.34.4.34 <code>_usb_isr_invoke_out_cb</code>	433
7.34.4.35 <code>USB_MAX_ENDPOINTS</code>	434
7.34.4.36 <code>USB_EP0_STATUS_STAGE</code>	434
7.34.4.37 <code>USB_SET_ADDRESS_MODE</code>	434
7.34.4.38 <code>USB_SET_ADDRESS_ACK_HANDLING</code>	434
7.34.4.39 <code>PLATFORM_USB_USE_USB1</code>	434
7.34.4.40 <code>usb_lld_get_frame_number</code>	434
7.34.4.41 <code>usb_lld_get_transaction_size</code>	435
7.34.4.42 <code>usb_lld_connect_bus</code>	435
7.34.4.43 <code>usb_lld_disconnect_bus</code>	435
7.34.5 Typedef Documentation	435
7.34.5.1 <code>USBDriver</code>	435
7.34.5.2 <code>usbep_t</code>	435
7.34.5.3 <code>usbcallback_t</code>	435
7.34.5.4 <code>usbepcallback_t</code>	436
7.34.5.5 <code>usbeventcb_t</code>	436
7.34.5.6 <code>usbreqhandler_t</code>	436
7.34.5.7 <code>usbgetdescriptor_t</code>	436
7.34.6 Enumeration Type Documentation	436
7.34.6.1 <code>usbstate_t</code>	436
7.34.6.2 <code>usbepstatus_t</code>	437
7.34.6.3 <code>usbep0state_t</code>	437
7.34.6.4 <code>usbevent_t</code>	437
7.34.7 Function Documentation	437
7.34.7.1 <code>set_address</code>	437
7.34.7.2 <code>default_handler</code>	438
7.34.7.3 <code>usbInit</code>	439
7.34.7.4 <code>usbObjectInit</code>	439

7.34.7.5	usbStart	440
7.34.7.6	usbStop	440
7.34.7.7	usbInitEndpoint1	441
7.34.7.8	usbDisableEndpoints1	441
7.34.7.9	usbPrepareReceive	442
7.34.7.10	usbPrepareTransmit	443
7.34.7.11	usbPrepareQueuedReceive	443
7.34.7.12	usbPrepareQueuedTransmit	444
7.34.7.13	usbStartReceive1	445
7.34.7.14	usbStartTransmit1	445
7.34.7.15	usbStallReceive1	446
7.34.7.16	usbStallTransmit1	447
7.34.7.17	_usb_reset	448
7.34.7.18	_usb_suspend	449
7.34.7.19	_usb_wakeup	449
7.34.7.20	_usb_ep0setup	449
7.34.7.21	_usb_ep0in	450
7.34.7.22	_usb_ep0out	451
7.34.7.23	usb_lld_init	452
7.34.7.24	usb_lld_start	452
7.34.7.25	usb_lld_stop	453
7.34.7.26	usb_lld_reset	454
7.34.7.27	usb_lld_set_address	454
7.34.7.28	usb_lld_init_endpoint	454
7.34.7.29	usb_lld_disable_endpoints	455
7.34.7.30	usb_lld_get_status_out	456
7.34.7.31	usb_lld_get_status_in	456
7.34.7.32	usb_lld_read_setup	456
7.34.7.33	usb_lld_prepare_receive	457
7.34.7.34	usb_lld_prepare_transmit	457
7.34.7.35	usb_lld_start_out	457
7.34.7.36	usb_lld_start_in	457
7.34.7.37	usb_lld_stall_out	458
7.34.7.38	usb_lld_stall_in	458
7.34.7.39	usb_lld_clear_out	458
7.34.7.40	usb_lld_clear_in	458
7.34.8	Variable Documentation	459
7.34.8.1	USBD1	459
7.34.8.2	ep0_state	459
7.34.8.3	in	459

7.34.8.4	out	459
7.34.8.5	ep0config	459
8	Data Structure Documentation	461
8.1	ADCCConfig Struct Reference	461
8.1.1	Detailed Description	461
8.2	ADCConversionGroup Struct Reference	462
8.2.1	Detailed Description	462
8.2.2	Field Documentation	463
8.2.2.1	circular	463
8.2.2.2	num_channels	463
8.2.2.3	end_cb	463
8.2.2.4	error_cb	463
8.3	ADCDriver Struct Reference	463
8.3.1	Detailed Description	464
8.3.2	Field Documentation	465
8.3.2.1	state	465
8.3.2.2	config	465
8.3.2.3	samples	465
8.3.2.4	depth	465
8.3.2.5	grpp	465
8.3.2.6	thread	465
8.3.2.7	mutex	465
8.4	BaseAsynchronousChannel Struct Reference	465
8.4.1	Detailed Description	467
8.4.2	Field Documentation	467
8.4.2.1	vmt	467
8.5	BaseAsynchronousChannelVMT Struct Reference	467
8.5.1	Detailed Description	469
8.6	BaseBlockDevice Struct Reference	469
8.6.1	Detailed Description	471
8.6.2	Field Documentation	471
8.6.2.1	vmt	471
8.7	BaseBlockDeviceVMT Struct Reference	471
8.7.1	Detailed Description	472
8.8	BaseChannel Struct Reference	472
8.8.1	Detailed Description	474
8.8.2	Field Documentation	474
8.8.2.1	vmt	474
8.9	BaseChannelVMT Struct Reference	474

8.9.1	Detailed Description	476
8.10	BaseSequentialStream Struct Reference	476
8.10.1	Detailed Description	477
8.10.2	Field Documentation	477
8.10.2.1	vmt	477
8.11	BaseSequentialStreamVMT Struct Reference	477
8.11.1	Detailed Description	478
8.12	BlockDeviceInfo Struct Reference	478
8.12.1	Detailed Description	479
8.12.2	Field Documentation	479
8.12.2.1	blk_size	479
8.12.2.2	blk_num	479
8.13	CANConfig Struct Reference	479
8.13.1	Detailed Description	480
8.14	CANDriver Struct Reference	480
8.14.1	Detailed Description	481
8.14.2	Field Documentation	481
8.14.2.1	state	481
8.14.2.2	config	481
8.14.2.3	txqueue	481
8.14.2.4	rxqueue	481
8.14.2.5	rxfull_event	481
8.14.2.6	txempty_event	482
8.14.2.7	error_event	482
8.14.2.8	sleep_event	482
8.14.2.9	wakeup_event	482
8.15	CANRxFrame Struct Reference	482
8.15.1	Detailed Description	483
8.15.2	Field Documentation	483
8.15.2.1	FMI	483
8.15.2.2	TIME	483
8.15.2.3	DLC	483
8.15.2.4	RTR	483
8.15.2.5	IDE	484
8.15.2.6	SID	484
8.15.2.7	EID	484
8.15.2.8	data8	484
8.15.2.9	data16	484
8.15.2.10	data32	484
8.16	CANTxFrame Struct Reference	484

8.16.1	Detailed Description	485
8.16.2	Field Documentation	485
8.16.2.1	DLC	485
8.16.2.2	RTR	485
8.16.2.3	IDE	485
8.16.2.4	SID	485
8.16.2.5	EID	485
8.16.2.6	data8	486
8.16.2.7	data16	486
8.16.2.8	data32	486
8.17	cdc_linecoding_t Struct Reference	486
8.17.1	Detailed Description	486
8.18	DACConfig Struct Reference	486
8.18.1	Detailed Description	487
8.19	DACConversionGroup Struct Reference	487
8.19.1	Detailed Description	488
8.19.2	Field Documentation	488
8.19.2.1	num_channels	488
8.19.2.2	end_cb	488
8.19.2.3	error_cb	488
8.20	DACDriver Struct Reference	488
8.20.1	Detailed Description	489
8.20.2	Field Documentation	490
8.20.2.1	state	490
8.20.2.2	grpp	490
8.20.2.3	samples	490
8.20.2.4	depth	490
8.20.2.5	config	490
8.20.2.6	thread	490
8.20.2.7	mutex	490
8.21	event_source Struct Reference	490
8.21.1	Detailed Description	491
8.21.2	Field Documentation	491
8.21.2.1	flags	491
8.21.2.2	cb	491
8.21.2.3	param	491
8.22	EXTChannelConfig Struct Reference	491
8.22.1	Detailed Description	492
8.22.2	Field Documentation	492
8.22.2.1	mode	492

8.22.2.2	cb	492
8.23	EXTConfig Struct Reference	493
8.23.1	Detailed Description	493
8.23.2	Field Documentation	493
8.23.2.1	channels	493
8.24	EXTDriver Struct Reference	494
8.24.1	Detailed Description	494
8.24.2	Field Documentation	494
8.24.2.1	state	494
8.24.2.2	config	495
8.25	FileStream Struct Reference	495
8.25.1	Detailed Description	496
8.25.2	Field Documentation	496
8.25.2.1	vmt	496
8.26	FileStreamVMT Struct Reference	496
8.26.1	Detailed Description	497
8.27	GPTConfig Struct Reference	497
8.27.1	Detailed Description	498
8.27.2	Field Documentation	498
8.27.2.1	frequency	498
8.27.2.2	callback	498
8.28	GPTDriver Struct Reference	499
8.28.1	Detailed Description	499
8.28.2	Field Documentation	499
8.28.2.1	state	499
8.28.2.2	config	499
8.29	I2CConfig Struct Reference	499
8.29.1	Detailed Description	500
8.30	I2CDriver Struct Reference	500
8.30.1	Detailed Description	501
8.30.2	Field Documentation	501
8.30.2.1	state	501
8.30.2.2	config	501
8.30.2.3	errors	501
8.31	I2SConfig Struct Reference	501
8.31.1	Detailed Description	502
8.31.2	Field Documentation	502
8.31.2.1	tx_buffer	502
8.31.2.2	rx_buffer	503
8.31.2.3	size	503

8.31.2.4	end_cb	503
8.32	I2SDriver Struct Reference	503
8.32.1	Detailed Description	504
8.32.2	Field Documentation	504
8.32.2.1	state	504
8.32.2.2	config	504
8.33	ICUConfig Struct Reference	504
8.33.1	Detailed Description	505
8.33.2	Field Documentation	505
8.33.2.1	mode	505
8.33.2.2	frequency	505
8.33.2.3	width_cb	505
8.33.2.4	period_cb	505
8.33.2.5	overflow_cb	505
8.34	ICUDriver Struct Reference	505
8.34.1	Detailed Description	506
8.34.2	Field Documentation	506
8.34.2.1	state	506
8.34.2.2	config	506
8.35	io_queue Struct Reference	506
8.35.1	Detailed Description	507
8.35.2	Field Documentation	508
8.35.2.1	q_waiting	508
8.35.2.2	q_counter	508
8.35.2.3	q_buffer	508
8.35.2.4	q_top	508
8.35.2.5	q_wptr	508
8.35.2.6	q_rptr	508
8.35.2.7	q_notify	508
8.35.2.8	q_link	508
8.36	IOBus Struct Reference	508
8.36.1	Detailed Description	509
8.36.2	Field Documentation	509
8.36.2.1	portid	509
8.36.2.2	mask	509
8.36.2.3	offset	509
8.37	MACConfig Struct Reference	510
8.37.1	Detailed Description	510
8.37.2	Field Documentation	510
8.37.2.1	mac_address	510

8.38 MACDriver Struct Reference	510
8.38.1 Detailed Description	511
8.38.2 Field Documentation	511
8.38.2.1 state	511
8.38.2.2 config	511
8.38.2.3 tdqueue	511
8.38.2.4 rdqueue	512
8.38.2.5 rdevent	512
8.39 MACReceiveDescriptor Struct Reference	512
8.39.1 Detailed Description	512
8.39.2 Field Documentation	512
8.39.2.1 offset	512
8.39.2.2 size	512
8.40 MACTransmitDescriptor Struct Reference	513
8.40.1 Detailed Description	513
8.40.2 Field Documentation	513
8.40.2.1 offset	513
8.40.2.2 size	513
8.41 MMCConfig Struct Reference	513
8.41.1 Detailed Description	514
8.41.2 Field Documentation	514
8.41.2.1 spip	514
8.41.2.2 lscfg	515
8.41.2.3 hscfg	515
8.42 MMCDriver Struct Reference	515
8.42.1 Detailed Description	516
8.42.2 Field Documentation	516
8.42.2.1 vmt	516
8.42.2.2 config	517
8.43 MMCDriverVMT Struct Reference	517
8.43.1 Detailed Description	518
8.44 MMCSDBlockDevice Struct Reference	518
8.44.1 Detailed Description	520
8.44.2 Field Documentation	520
8.44.2.1 vmt	520
8.45 MMCSDBlockDeviceVMT Struct Reference	520
8.45.1 Detailed Description	521
8.46 PALConfig Struct Reference	522
8.46.1 Detailed Description	522
8.47 PWMChannelConfig Struct Reference	522

8.47.1 Detailed Description	523
8.47.2 Field Documentation	523
8.47.2.1 mode	523
8.47.2.2 callback	523
8.48 PWMConfig Struct Reference	524
8.48.1 Detailed Description	525
8.48.2 Field Documentation	525
8.48.2.1 frequency	525
8.48.2.2 period	525
8.48.2.3 callback	525
8.48.2.4 channels	525
8.49 PWMDriver Struct Reference	525
8.49.1 Detailed Description	526
8.49.2 Field Documentation	526
8.49.2.1 state	527
8.49.2.2 config	527
8.49.2.3 period	527
8.49.2.4 enabled	527
8.49.2.5 channels	527
8.50 RTCAlarm Struct Reference	527
8.50.1 Detailed Description	527
8.51 RTCDateTime Struct Reference	527
8.51.1 Detailed Description	528
8.51.2 Field Documentation	528
8.51.2.1 year	528
8.51.2.2 month	528
8.51.2.3 dstflag	528
8.51.2.4 dayofweek	529
8.51.2.5 day	529
8.51.2.6 millisecond	529
8.52 RTCDriver Struct Reference	529
8.52.1 Detailed Description	530
8.52.2 Field Documentation	530
8.52.2.1 vmt	530
8.53 RTCDriverVMT Struct Reference	530
8.53.1 Detailed Description	531
8.54 SDCCConfig Struct Reference	531
8.54.1 Detailed Description	532
8.54.2 Field Documentation	532
8.54.2.1 scratchpad	532

8.54.2.2	bus_width	532
8.55	SDCDriver Struct Reference	532
8.55.1	Detailed Description	533
8.55.2	Field Documentation	533
8.55.2.1	vmt	533
8.55.2.2	config	534
8.55.2.3	cardmode	534
8.55.2.4	errors	534
8.55.2.5	rca	534
8.56	SDCDriverVMT Struct Reference	534
8.56.1	Detailed Description	535
8.57	SerialConfig Struct Reference	535
8.57.1	Detailed Description	536
8.57.2	Field Documentation	536
8.57.2.1	speed	536
8.58	SerialDriver Struct Reference	536
8.58.1	Detailed Description	537
8.58.2	Field Documentation	537
8.58.2.1	vmt	537
8.59	SerialDriverVMT Struct Reference	537
8.59.1	Detailed Description	539
8.60	SerialUSBConfig Struct Reference	539
8.60.1	Detailed Description	539
8.60.2	Field Documentation	540
8.60.2.1	usbp	540
8.60.2.2	bulk_in	540
8.60.2.3	bulk_out	540
8.60.2.4	int_in	540
8.61	SerialUSBDriver Struct Reference	540
8.61.1	Detailed Description	542
8.61.2	Field Documentation	542
8.61.2.1	vmt	542
8.62	SerialUSBDriverVMT Struct Reference	542
8.62.1	Detailed Description	544
8.63	SPIConfig Struct Reference	544
8.63.1	Detailed Description	544
8.63.2	Field Documentation	544
8.63.2.1	end_cb	544
8.64	SPIDriver Struct Reference	545
8.64.1	Detailed Description	545

8.64.2 Field Documentation	545
8.64.2.1 state	545
8.64.2.2 config	546
8.64.2.3 thread	546
8.64.2.4 mutex	546
8.65 threads_queue_t Struct Reference	546
8.65.1 Detailed Description	546
8.66 UARTConfig Struct Reference	546
8.66.1 Detailed Description	547
8.66.2 Field Documentation	547
8.66.2.1 txend1_cb	547
8.66.2.2 txend2_cb	548
8.66.2.3 rxend_cb	548
8.66.2.4 rxchar_cb	548
8.66.2.5 rxerr_cb	548
8.67 UARTDriver Struct Reference	548
8.67.1 Detailed Description	549
8.67.2 Field Documentation	549
8.67.2.1 state	549
8.67.2.2 txstate	549
8.67.2.3 rxstate	549
8.67.2.4 config	549
8.68 unpacked_mmc_cid_t Struct Reference	549
8.68.1 Detailed Description	550
8.69 unpacked_mmc_csd_t Struct Reference	550
8.69.1 Detailed Description	550
8.70 unpacked_sdc_cid_t Struct Reference	551
8.70.1 Detailed Description	551
8.71 unpacked_sdc_csd_10_t Struct Reference	551
8.71.1 Detailed Description	552
8.72 unpacked_sdc_csd_20_t Struct Reference	552
8.72.1 Detailed Description	553
8.73 USBConfig Struct Reference	553
8.73.1 Detailed Description	555
8.73.2 Field Documentation	555
8.73.2.1 event_cb	555
8.73.2.2 get_descriptor_cb	555
8.73.2.3 requests_hook_cb	555
8.73.2.4 sof_cb	555
8.74 USBDescriptor Struct Reference	555

8.74.1 Detailed Description	556
8.74.2 Field Documentation	556
8.74.2.1 <code>ud_size</code>	556
8.74.2.2 <code>ud_string</code>	556
8.75 USBDriver Struct Reference	556
8.75.1 Detailed Description	557
8.75.2 Field Documentation	557
8.75.2.1 <code>state</code>	557
8.75.2.2 <code>config</code>	557
8.75.2.3 <code>transmitting</code>	557
8.75.2.4 <code>receiving</code>	557
8.75.2.5 <code>epc</code>	558
8.75.2.6 <code>in_params</code>	558
8.75.2.7 <code>out_params</code>	558
8.75.2.8 <code>ep0state</code>	558
8.75.2.9 <code>ep0next</code>	558
8.75.2.10 <code>ep0n</code>	558
8.75.2.11 <code>ep0endcb</code>	558
8.75.2.12 <code>setup</code>	558
8.75.2.13 <code>status</code>	558
8.75.2.14 <code>address</code>	558
8.75.2.15 <code>configuration</code>	558
8.76 USBEndpointConfig Struct Reference	559
8.76.1 Detailed Description	560
8.76.2 Field Documentation	560
8.76.2.1 <code>ep_mode</code>	560
8.76.2.2 <code>setup_cb</code>	560
8.76.2.3 <code>in_cb</code>	560
8.76.2.4 <code>out_cb</code>	561
8.76.2.5 <code>in_maxsize</code>	561
8.76.2.6 <code>out_maxsize</code>	561
8.76.2.7 <code>in_state</code>	561
8.76.2.8 <code>out_state</code>	561
8.77 USBIInEndpointState Struct Reference	561
8.77.1 Detailed Description	562
8.77.2 Field Documentation	563
8.77.2.1 <code>txqueued</code>	563
8.77.2.2 <code>txsize</code>	563
8.77.2.3 <code>txcnt</code>	563
8.77.2.4 <code>txbuf</code>	563

8.77.2.5 txqueue	563
8.78 USBOutEndpointState Struct Reference	563
8.78.1 Detailed Description	564
8.78.2 Field Documentation	565
8.78.2.1 rxqueued	565
8.78.2.2 rxsize	565
8.78.2.3 rxcnt	565
8.78.2.4 rxbuf	565
8.78.2.5 rxqueue	565
9 File Documentation	567
9.1 adc.c File Reference	567
9.1.1 Detailed Description	567
9.2 adc.h File Reference	568
9.2.1 Detailed Description	569
9.3 adc_lld.c File Reference	569
9.3.1 Detailed Description	569
9.4 adc_lld.h File Reference	569
9.4.1 Detailed Description	570
9.5 can.c File Reference	571
9.5.1 Detailed Description	571
9.6 can.h File Reference	571
9.6.1 Detailed Description	572
9.7 can_lld.c File Reference	572
9.7.1 Detailed Description	573
9.8 can_lld.h File Reference	573
9.8.1 Detailed Description	574
9.9 dac.c File Reference	574
9.9.1 Detailed Description	575
9.10 dac.h File Reference	575
9.10.1 Detailed Description	576
9.11 dac_lld.c File Reference	577
9.11.1 Detailed Description	577
9.12 dac_lld.h File Reference	577
9.12.1 Detailed Description	578
9.13 ext.c File Reference	579
9.13.1 Detailed Description	579
9.14 ext.h File Reference	579
9.14.1 Detailed Description	580
9.15 ext_lld.c File Reference	580

9.15.1	Detailed Description	581
9.16	ext_lld.h File Reference	581
9.16.1	Detailed Description	582
9.17	gpt.c File Reference	582
9.17.1	Detailed Description	583
9.18	gpt.h File Reference	583
9.18.1	Detailed Description	584
9.19	gpt_lld.c File Reference	584
9.19.1	Detailed Description	584
9.20	gpt_lld.h File Reference	584
9.20.1	Detailed Description	585
9.21	hal.c File Reference	585
9.21.1	Detailed Description	586
9.22	hal.h File Reference	586
9.22.1	Detailed Description	587
9.23	hal_channels.h File Reference	587
9.23.1	Detailed Description	588
9.24	hal_files.h File Reference	588
9.24.1	Detailed Description	589
9.25	hal_ioblock.h File Reference	589
9.25.1	Detailed Description	590
9.26	hal_lld.c File Reference	591
9.26.1	Detailed Description	591
9.27	hal_lld.h File Reference	591
9.27.1	Detailed Description	591
9.28	hal_mmcisd.c File Reference	591
9.28.1	Detailed Description	592
9.29	hal_mmcisd.h File Reference	592
9.29.1	Detailed Description	596
9.30	hal_queues.c File Reference	596
9.30.1	Detailed Description	597
9.31	hal_queues.h File Reference	597
9.31.1	Detailed Description	599
9.32	hal_streams.h File Reference	599
9.32.1	Detailed Description	599
9.33	halconf.h File Reference	599
9.33.1	Detailed Description	601
9.34	i2c.c File Reference	601
9.34.1	Detailed Description	602
9.35	i2c.h File Reference	602

9.35.1 Detailed Description	603
9.36 i2c_lld.c File Reference	604
9.36.1 Detailed Description	604
9.37 i2c_lld.h File Reference	604
9.37.1 Detailed Description	605
9.38 i2s.c File Reference	605
9.38.1 Detailed Description	606
9.39 i2s.h File Reference	606
9.39.1 Detailed Description	607
9.40 i2s_lld.c File Reference	607
9.40.1 Detailed Description	607
9.41 i2s_lld.h File Reference	607
9.41.1 Detailed Description	608
9.42 icu.c File Reference	608
9.42.1 Detailed Description	608
9.43 icu.h File Reference	609
9.43.1 Detailed Description	610
9.44 icu_lld.c File Reference	610
9.44.1 Detailed Description	611
9.45 icu_lld.h File Reference	611
9.45.1 Detailed Description	612
9.46 mac.c File Reference	612
9.46.1 Detailed Description	612
9.47 mac.h File Reference	613
9.47.1 Detailed Description	614
9.48 mac_lld.c File Reference	614
9.48.1 Detailed Description	615
9.49 mac_lld.h File Reference	615
9.49.1 Detailed Description	616
9.50 mmc_spi.c File Reference	616
9.50.1 Detailed Description	617
9.51 mmc_spi.h File Reference	617
9.51.1 Detailed Description	618
9.52 osal.c File Reference	619
9.52.1 Detailed Description	620
9.53 osal.h File Reference	620
9.53.1 Detailed Description	624
9.54 pal.c File Reference	624
9.54.1 Detailed Description	624
9.55 pal.h File Reference	624

9.55.1	Detailed Description	626
9.56	pal_Ild.c File Reference	626
9.56.1	Detailed Description	626
9.57	pal_Ild.h File Reference	626
9.57.1	Detailed Description	628
9.58	pwm.c File Reference	628
9.58.1	Detailed Description	628
9.59	pwm.h File Reference	628
9.59.1	Detailed Description	630
9.60	pwm_Ild.c File Reference	630
9.60.1	Detailed Description	631
9.61	pwm_Ild.h File Reference	631
9.61.1	Detailed Description	632
9.62	rtc.c File Reference	632
9.62.1	Detailed Description	633
9.63	rtc.h File Reference	633
9.63.1	Detailed Description	634
9.64	rtc_Ild.c File Reference	634
9.64.1	Detailed Description	634
9.65	rtc_Ild.h File Reference	635
9.65.1	Detailed Description	636
9.66	sdc.c File Reference	636
9.66.1	Detailed Description	637
9.67	sdc.h File Reference	637
9.67.1	Detailed Description	639
9.68	sdc_Ild.c File Reference	639
9.68.1	Detailed Description	640
9.69	sdc_Ild.h File Reference	640
9.69.1	Detailed Description	641
9.70	serial.c File Reference	641
9.70.1	Detailed Description	642
9.71	serial.h File Reference	642
9.71.1	Detailed Description	644
9.72	serial_Ild.c File Reference	644
9.72.1	Detailed Description	644
9.73	serial_Ild.h File Reference	644
9.73.1	Detailed Description	645
9.74	serial_usb.c File Reference	645
9.74.1	Detailed Description	645
9.75	serial_usb.h File Reference	646

9.75.1 Detailed Description	648
9.76 spi.c File Reference	648
9.76.1 Detailed Description	648
9.77 spi.h File Reference	649
9.77.1 Detailed Description	650
9.78 spi_lld.c File Reference	650
9.78.1 Detailed Description	651
9.79 spi_lld.h File Reference	651
9.79.1 Detailed Description	652
9.80 st.c File Reference	652
9.80.1 Detailed Description	652
9.81 st.h File Reference	652
9.81.1 Detailed Description	653
9.82 st_lld.c File Reference	653
9.82.1 Detailed Description	653
9.83 st_lld.h File Reference	653
9.83.1 Detailed Description	654
9.84 uart.c File Reference	654
9.84.1 Detailed Description	654
9.85 uart.h File Reference	655
9.85.1 Detailed Description	656
9.86 uart_lld.c File Reference	656
9.86.1 Detailed Description	656
9.87 uart_lld.h File Reference	656
9.87.1 Detailed Description	657
9.88 usb.c File Reference	658
9.88.1 Detailed Description	659
9.89 usb.h File Reference	659
9.89.1 Detailed Description	662
9.90 usb_lld.c File Reference	662
9.90.1 Detailed Description	663
9.90.2 Variable Documentation	663
9.90.2.1 in	663
9.90.2.2 out	663
9.91 usb_lld.h File Reference	663
9.91.1 Detailed Description	665

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

globalScope> Global **sdGetWouldBlock** (SerialDriver *sdp)

globalScope> Global **sdPutWouldBlock** (SerialDriver *sdp)

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

HAL	193
Configuration	196
Normal Drivers	202
ADC Driver	17
CAN Driver	35
DAC Driver	49
EXT Driver	66
GPT Driver	76
HAL Driver	93
I2C Driver	137
I2S Driver	151
ICU Driver	159
MAC Driver	177
PAL Driver	261
PWM Driver	285
RTC Driver	307
SDC Driver	317
Serial Driver	346
SPI Driver	371
ST Driver	395
UART Driver	401
USB Driver	417
Complex Drivers	203
MMC over SPI Driver	229
Serial over USB Driver	361
Interfaces	204
Abstract I/O Channel	96
Abstract Files	102
Abstract I/O Block Device	108
Abstract Streams	134
Inner Code	205
I/O Queues	115
MMC/SD Block Device	251
OSAL	206

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

ADCConfig	461
ADCConversionGroup	462
ADCDriver	463
BaseBlockDevice	469
MMCSDBlockDevice	518
MMCDriver	515
BaseBlockDeviceVMT	471
MMCSDBlockDeviceVMT	520
MMCDriverVMT	517
SDCDriverVMT	534
BaseSequentialStream	476
BaseChannel	472
BaseAsynchronousChannel	465
SerialDriver	536
SerialUSBDriver	540
FileStream	495
RTCDriverVMT	530
BaseSequentialStreamVMT	477
BaseChannelVMT	474
BaseAsynchronousChannelVMT	467
SerialDriverVMT	537
SerialUSBDriverVMT	542
FileStreamVMT	496
BlockDeviceInfo	478
CANConfig	479
CANDriver	480
CANRxFrame	482
CANTxFrame	484
cdc_linecoding_t	486
DACConfig	486
DACConversionGroup	487
DACDriver	488
event_source	490
EXTChannelConfig	491
EXTConfig	493
EXTDriver	494

GPTConfig	497
GPTDriver	499
I2CConfig	499
I2CDriver	500
I2SConfig	501
I2SDriver	503
ICUConfig	504
ICUDriver	505
io_queue	506
IOBus	508
MACConfig	510
MACDriver	510
MACReceiveDescriptor	512
MACTransmitDescriptor	513
MMCConfig	513
PALConfig	522
PWMChannelConfig	522
PWMConfig	524
PWMDriver	525
RTCAlarm	527
RTCDateTime	527
RTCDriver	529
SDCConfig	531
SDCDriver	532
SerialConfig	535
SerialUSBConfig	539
SPIConfig	544
SPIDriver	545
threads_queue_t	546
UARTConfig	546
UARTDriver	548
unpacked_mmc_cid_t	549
unpacked_mmc_csd_t	550
unpacked_sdc_cid_t	551
unpacked_sdc_csd_10_t	551
unpacked_sdc_csd_20_t	552
USBConfig	553
USBDescriptor	555
USBDriver	556
USBEndpointConfig	559
USBInEndpointState	561
USBOutEndpointState	563

Chapter 5

Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

ADCConfig	Driver configuration structure	461
ADCConversionGroup	Conversion group configuration structure	462
ADCDriver	Structure representing an ADC driver	463
BaseAsynchronousChannel	Base asynchronous channel class	465
BaseAsynchronousChannelVMT	BaseAsynchronousChannel virtual methods table	467
BaseBlockDevice	Base block device class	469
BaseBlockDeviceVMT	BaseBlockDevice virtual methods table	471
BaseChannel	Base channel class	472
BaseChannelVMT	BaseChannel virtual methods table	474
BaseSequentialStream	Base stream class	476
BaseSequentialStreamVMT	BaseSequentialStream virtual methods table	477
BlockDeviceInfo	Block device info	478
CANConfig	Driver configuration structure	479
CANDriver	Structure representing an CAN driver	480
CANRxFrame	CAN received frame	482
CANTxFrame	CAN transmission frame	484
cdc_linecoding_t	Type of Line Coding structure	486
DACConfig	Driver configuration structure	486
DACConversionGroup	DAC Conversion group structure	487

DACDriver	Structure representing a DAC driver	488
event_source	Events source object	490
EXTChannelConfig	Channel configuration structure	491
EXTConfig	Driver configuration structure	493
EXTDriver	Structure representing an EXT driver	494
FileStream	Base file stream class	495
FileStreamVMT	FileStream virtual methods table	496
GPTConfig	Driver configuration structure	497
GPTDriver	Structure representing a GPT driver	499
I2CConfig	Type of I2C driver configuration structure	499
I2CDriver	Structure representing an I2C driver	500
I2SConfig	Driver configuration structure	501
I2SDriver	Structure representing an I2S driver	503
ICUConfig	Driver configuration structure	504
ICUDriver	Structure representing an ICU driver	505
io_queue	Generic I/O queue structure	506
IOBus	I/O bus descriptor	508
MACConfig	Driver configuration structure	510
MACDriver	Structure representing a MAC driver	510
MACReceiveDescriptor	Structure representing a receive descriptor	512
MACTransmitDescriptor	Structure representing a transmit descriptor	513
MMCConfig	MMC/SD over SPI driver configuration structure	513
MMCDriver	Structure representing a MMC/SD over SPI driver	515
MMCDriverVMT	MMCDriver virtual methods table	517
MMCSDBlockDevice	MCC/SD block device class	518
MMCSDBlockDeviceVMT	MMCSDBlockDevice virtual methods table	520
PALConfig	Generic I/O ports static initializer	522
PWMChannelConfig	Type of a PWM driver channel configuration structure	522
PWMConfig	Type of a PWM driver configuration structure	524

PWMDriver	Structure representing a PWM driver	525
RTCAlarm	Type of a structure representing an RTC alarm time stamp	527
RTCDateTime	Type of a structure representing an RTC date/time stamp	527
RTCDriver	Structure representing an RTC driver	529
RTCDriverVMT	RTCDriver virtual methods table	530
SDCConfig	Driver configuration structure	531
SDCDriver	Structure representing an SDC driver	532
SDCDriverVMT	SDCDriver virtual methods table	534
SerialConfig	PLATFORM Serial Driver configuration structure	535
SerialDriver	Full duplex serial driver class	536
SerialDriverVMT	SerialDriver virtual methods table	537
SerialUSBConfig	Serial over USB Driver configuration structure	539
SerialUSBDriver	Full duplex serial driver class	540
SerialUSBDriverVMT	SerialDriver virtual methods table	542
SPIConfig	Driver configuration structure	544
SPIDriver	Structure representing an SPI driver	545
threads_queue_t	Type of a thread queue	546
UARTConfig	Driver configuration structure	546
UARTDriver	Structure representing an UART driver	548
unpacked_mmc_cid_t	Unpacked CID register from MMC	549
unpacked_mmc_csd_t	Unpacked CSD register from MMC	550
unpacked_sdc_cid_t	Unpacked CID register from SDC	551
unpacked_sdc_csd_10_t	Unpacked CSD v1.0 register from SDC	551
unpacked_sdc_csd_20_t	Unpacked CSD v2.0 register from SDC	552
USBConfig	Type of an USB driver configuration structure	553
USBDescriptor	Type of an USB descriptor	555
USBDriver	Structure representing an USB driver	556
USBEndpointConfig	Type of an USB endpoint configuration structure	559
USBInEndpointState	Type of an IN endpoint state structure	561

USBOutEndpointState	Type of an OUT endpoint state structure	563
---------------------	---	-----

Chapter 6

File Index

6.1 File List

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

adc.c	ADC Driver code	567
adc.h	ADC Driver macros and structures	568
adc_lld.c	PLATFORM ADC subsystem low level driver source	569
adc_lld.h	PLATFORM ADC subsystem low level driver header	569
can.c	CAN Driver code	571
can.h	CAN Driver macros and structures	571
can_lld.c	PLATFORM CAN subsystem low level driver source	572
can_lld.h	PLATFORM CAN subsystem low level driver header	573
dac.c	DAC Driver code	574
dac.h	DAC Driver macros and structures	575
dac_lld.c	PLATFORM DAC subsystem low level driver source	577
dac_lld.h	PLATFORM DAC subsystem low level driver header	577
ext.c	EXT Driver code	579
ext.h	EXT Driver macros and structures	579
ext_lld.c	PLATFORM EXT subsystem low level driver source	580
ext_lld.h	PLATFORM EXT subsystem low level driver header	581
gpt.c	GPT Driver code	582
gpt.h	GPT Driver macros and structures	583
gpt_lld.c	PLATFORM GPT subsystem low level driver source	584

gpt_ll.h	PLATFORM GPT subsystem low level driver header	584
hal.c	HAL subsystem code	585
hal.h	HAL subsystem header	586
hal_channels.h	I/O channels access	587
hal_files.h	Data files	588
hal_ioblock.h	I/O block devices access	589
hal_lld.c	PLATFORM HAL subsystem low level driver source	591
hal_lld.h	PLATFORM HAL subsystem low level driver header	591
hal_mmcisd.c	MMC/SD cards common code	591
hal_mmcisd.h	MMC/SD cards common header	592
hal_queues.c	I/O Queues code	596
hal_queues.h	I/O Queues macros and structures	597
hal_streams.h	Data streams	599
halconf.h	HAL configuration header	599
i2c.c	I2C Driver code	601
i2c.h	I2C Driver macros and structures	602
i2c_lld.c	PLATFORM I2C subsystem low level driver source	604
i2c_lld.h	PLATFORM I2C subsystem low level driver header	604
i2s.c	I2S Driver code	605
i2s.h	I2S Driver macros and structures	606
i2s_lld.c	PLATFORM I2S subsystem low level driver source	607
i2s_lld.h	PLATFORM I2S subsystem low level driver header	607
icu.c	ICU Driver code	608
icu.h	ICU Driver macros and structures	609
icu_lld.c	PLATFORM ADC subsystem low level driver source	610
icu_lld.h	PLATFORM ICU subsystem low level driver header	611
mac.c	MAC Driver code	612
mac.h	MAC Driver macros and structures	613
mac_lld.c	PLATFORM MAC subsystem low level driver source	614

mac_lld.h	PLATFORM MAC subsystem low level driver header	615
mmc_spi.c	MMC over SPI driver code	616
mmc_spi.h	MMC over SPI driver header	617
osal.c	OSAL module code	619
osal.h	OSAL module header	620
pal.c	I/O Ports Abstraction Layer code	624
pal.h	I/O Ports Abstraction Layer macros, types and structures	624
pal_lld.c	PLATFORM PAL subsystem low level driver source	626
pal_lld.h	PLATFORM PAL subsystem low level driver header	626
pwm.c	PWM Driver code	628
pwm.h	PWM Driver macros and structures	628
pwm_lld.c	PLATFORM PWM subsystem low level driver source	630
pwm_lld.h	PLATFORM PWM subsystem low level driver header	631
rtc.c	RTC Driver code	632
rtc.h	RTC Driver macros and structures	633
rtc_lld.c	PLATFORM RTC subsystem low level driver source	634
rtc_lld.h	PLATFORM RTC subsystem low level driver header	635
sdc.c	SDC Driver code	636
sdc.h	SDC Driver macros and structures	637
sdc_lld.c	PLATFORM SDC subsystem low level driver source	639
sdc_lld.h	PLATFORM SDC subsystem low level driver header	640
serial.c	Serial Driver code	641
serial.h	Serial Driver macros and structures	642
serial_lld.c	PLATFORM serial subsystem low level driver source	644
serial_lld.h	PLATFORM serial subsystem low level driver header	644
serial_usb.c	Serial over USB Driver code	645
serial_usb.h	Serial over USB Driver macros and structures	646
spi.c	SPI Driver code	648
spi.h	SPI Driver macros and structures	649

spi_lld.c	PLATFORM SPI subsystem low level driver source	650
spi_lld.h	PLATFORM SPI subsystem low level driver header	651
st.c	ST Driver code	652
st.h	ST Driver macros and structures	652
st_lld.c	PLATFORM ST subsystem low level driver source	653
st_lld.h	PLATFORM ST subsystem low level driver header	653
uart.c	UART Driver code	654
uart.h	UART Driver macros and structures	655
uart_lld.c	PLATFORM UART subsystem low level driver source	656
uart_lld.h	PLATFORM UART subsystem low level driver header	656
usb.c	USB Driver code	658
usb.h	USB Driver macros and structures	659
usb_lld.c	PLATFORM USB subsystem low level driver source	662
usb_lld.h	PLATFORM USB subsystem low level driver header	663

Chapter 7

Module Documentation

7.1 ADC Driver

Generic ADC Driver.

7.1.1 Detailed Description

Generic ADC Driver.

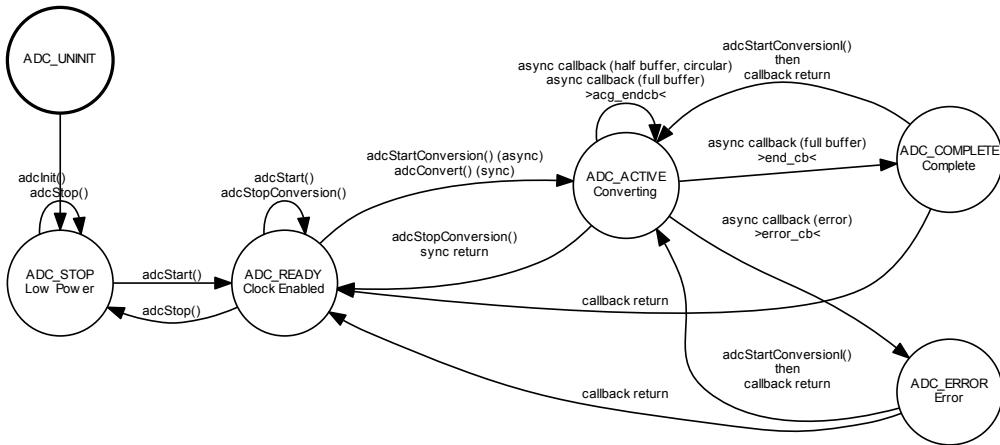
This module implements a generic ADC (Analog to Digital Converter) driver supporting a variety of buffer and conversion modes.

Precondition

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

7.1.2 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).



7.1.3 ADC Operations

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

7.1.3.1 ADC Conversion Groups

The [ADCConversionGroup](#) is the objects that specifies a physical conversion operation. This structure contains some standard fields and several implementation-dependent fields.

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

7.1.3.2 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.

7.1.3.3 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.

ADC configuration options

- `#define ADC_USE_WAIT TRUE`
Enables synchronous APIs.
- `#define ADC_USE_MUTUAL_EXCLUSION TRUE`
Enables the `adcAcquireBus()` and `adcReleaseBus()` APIs.

Low level driver helper macros

- `#define _adc_reset_i(adcp) osalThreadResumel(&(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 uint16_t adcsample_t`
ADC sample data type.
- `typedef uint16_t adc_channels_num_t`
Channels number in a conversion group.
- `typedef struct ADCDriver ADCDriver`
Type of a structure representing an ADC driver.
- `typedef void(* adccallback_t) (ADCDriver *adcp, adcsample_t *buffer, size_t n)`
ADC notification callback type.
- `typedef void(* adcerrorcallback_t) (ADCDriver *adcp, adcerror_t err)`
ADC error callback type.

Data Structures

- struct [ADCConversionGroup](#)
Conversion group configuration structure.
- struct [ADCConfig](#)
Driver configuration structure.
- struct [ADCDriver](#)
Structure representing an ADC driver.

Functions

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

Enumerations

- enum `adcstate_t` {
 `ADC_UNINIT` = 0, `ADC_STOP` = 1, `ADC_READY` = 2, `ADC_ACTIVE` = 3,
`ADC_COMPLETE` = 4, `ADC_ERROR` = 5 }

Driver state machine possible states.
- enum `adcerror_t` { `ADC_ERR_DMAFAILURE` = 0, `ADC_ERR_OVERFLOW` = 1, `ADC_ERR_AWD` = 2 }

Possible ADC failure causes.

Variables

- `ADCDriver ADCD1`
ADC1 driver identifier.

7.1.4 Macro Definition Documentation

7.1.4.1 `#define ADC_USE_WAIT TRUE`

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.1.4.2 `#define ADC_USE_MUTUAL_EXCLUSION TRUE`

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

Note

Disabling this option saves both code and data space.

7.1.4.3 `#define _adc_reset_i(adcp) osalThreadResumel(&(adcp)->thread, MSG_RESET)`

Resumes a thread waiting for a conversion completion.

Parameters

in	<code>adcp</code>	pointer to the <code>ADCDriver</code> object
----	-------------------	--

Function Class:

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

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

Resumes a thread waiting for a conversion completion.

Parameters

in	<i>adcp</i>	pointer to the ADCDriver object
----	-------------	---

Function Class:

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

7.1.4.5 #define _adc_wakeup_isr(*adcp*)**Value:**

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

Wakes up the waiting thread.

Parameters

in	<i>adcp</i>	pointer to the ADCDriver object
----	-------------	---

Function Class:

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

7.1.4.6 #define _adc_timeout_isr(*adcp*)**Value:**

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

Wakes up the waiting thread with a timeout message.

Parameters

in	<i>adcp</i>	pointer to the ADCDriver object
----	-------------	---

Function Class:

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

7.1.4.7 #define _adc_isr_half_code(*adcp*)**Value:**

```
{
    if ((adcp)->grpp->end_cb != NULL) {
        (adcp)->grpp->end_cb(adcp, (adcp)->samples, (adcp)->depth / 2);
    }
}
```

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	<i>adcp</i>	pointer to the ADCDriver object
----	-------------	---

Function Class:

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

7.1.4.8 #define _adc_isr_full_code(*adcp*)

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

in	<i>adcp</i>	pointer to the ADCDriver object
----	-------------	---

Function Class:

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

7.1.4.9 #define _adc_isr_error_code(*adcp*, *err*)**Value:**

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

Common ISR code, error event.

This code handles the portable part of the ISR code:

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

Note

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

Parameters

in	<i>adcp</i>	pointer to the ADCDriver object
in	<i>err</i>	platform dependent error code

Function Class:

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

7.1.4.10 #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.

7.1.5 Typedef Documentation**7.1.5.1 typedef uint16_t adcsample_t**

ADC sample data type.

7.1.5.2 typedef uint16_t adc_channels_num_t

Channels number in a conversion group.

7.1.5.3 typedef struct ADCDriver ADCDriver

Type of a structure representing an ADC driver.

7.1.5.4 typedef void(* adccallback_t) (ADCDriver *adcp, adcsample_t *buffer, size_t n)

ADC notification callback type.

Parameters

in	<i>adcp</i>	pointer to the ADCDriver object triggering the callback
in	<i>buffer</i>	pointer to the most recent samples data
in	<i>n</i>	number of buffer rows available starting from <i>buffer</i>

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

ADC error callback type.

Parameters

in	<i>adcp</i>	pointer to the ADC Driver object triggering the callback
in	<i>err</i>	ADC error code

7.1.6 Enumeration Type Documentation

7.1.6.1 enum adcstate_t

Driver state machine possible states.

Enumerator

- ADC_UNINIT** Not initialized.
- ADC_STOP** Stopped.
- ADC_READY** Ready.
- ADC_ACTIVE** Converting.
- ADC_COMPLETE** Conversion complete.
- ADC_ERROR** Conversion complete.

7.1.6.2 enum adcerror_t

Possible ADC failure causes.

Note

Error codes are architecture dependent and should not relied upon.

Enumerator

- ADC_ERR_DMAFAILURE** DMA operations failure.
- ADC_ERR_OVERFLOW** ADC overflow condition.
- ADC_ERR_AWD** Analog watchdog triggered.

7.1.7 Function Documentation

7.1.7.1 void adclnit(void)

ADC Driver initialization.

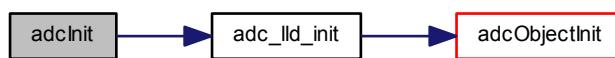
Note

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:



7.1.7.2 void adcObjectInit (ADCDriver * *adcp*)

Initializes the standard part of a [ADCDriver](#) structure.

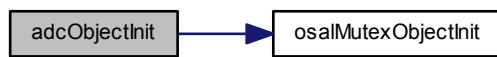
Parameters

out	<i>adcp</i>	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:



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

Configures and activates the ADC peripheral.

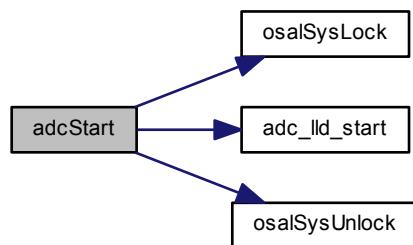
Parameters

in	<i>adcp</i>	pointer to the ADCDriver object
in	<i>config</i>	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:



7.1.7.4 void adcStop (**ADCDriver** * *adcp*)

Deactivates the ADC peripheral.

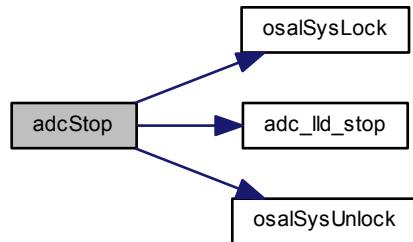
Parameters

in	<i>adcp</i>	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:



7.1.7.5 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.

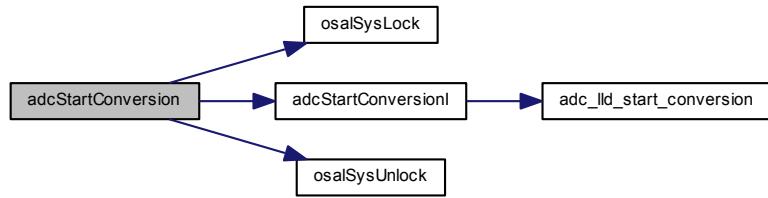
Parameters

in	<i>adcp</i>	pointer to the ADCDriver object
in	<i>grpp</i>	pointer to a ADCConversionGroup object
out	<i>samples</i>	pointer to the samples buffer
in	<i>depth</i>	buffer depth (matrix rows number). The buffer depth must be one or an even number.

Function Class:

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

Here is the call graph for this function:



7.1.7.6 void adcStartConversionl (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

in	<i>adcp</i>	pointer to the <code>ADCDriver</code> object
in	<i>grpp</i>	pointer to a <code>ADCConversionGroup</code> object
out	<i>samples</i>	pointer to the samples buffer
in	<i>depth</i>	buffer depth (matrix rows number). The buffer depth must be one or an even number.

Function Class:

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

Here is the call graph for this function:



7.1.7.7 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.

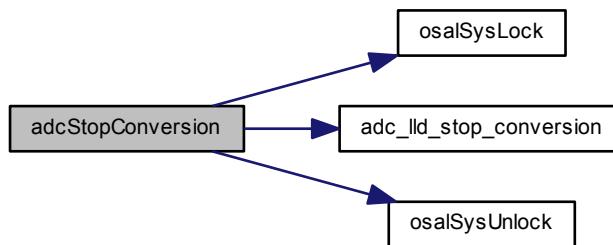
Parameters

in	<i>adcp</i>	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:



7.1.7.8 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	<i>adcp</i>	pointer to the ADCDriver object
----	-------------	---

Function Class:

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

Here is the call graph for this function:



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

Performs an ADC conversion.

Performs a synchronous conversion operation.

Note

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

Parameters

in	<code>adcp</code>	pointer to the <code>ADCDriver</code> object
in	<code>grpp</code>	pointer to a <code>ADCConversionGroup</code> object
out	<code>samples</code>	pointer to the samples buffer
in	<code>depth</code>	buffer depth (matrix rows number). The buffer depth must be one or an even number.

Returns

The operation result.

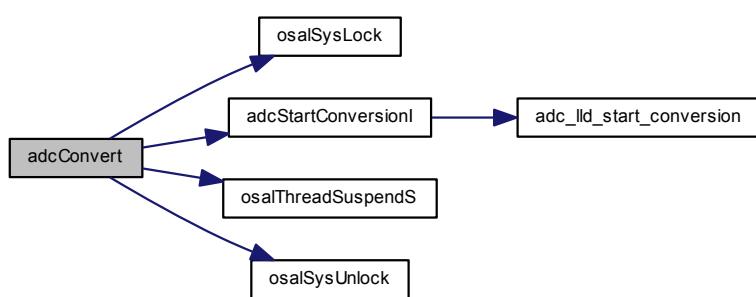
Return values

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

Function Class:

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

Here is the call graph for this function:



7.1.7.10 `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	<i>adcp</i>	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:

**7.1.7.11 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	<i>adcp</i>	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:

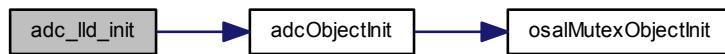
**7.1.7.12 void adc_lld_init (void)**

Low level ADC driver initialization.

Function Class:

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

Here is the call graph for this function:

**7.1.7.13 void adc_lld_start (ADCDriver * adcp)**

Configures and activates the ADC peripheral.

Parameters

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

Function Class:

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

7.1.7.14 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.

7.1.7.15 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.

7.1.7.16 void adc_lld_stop_conversion (ADCDriver * adcp)

Stops an ongoing conversion.

Parameters

in	<i>adcp</i>	pointer to the ADCDriver object
----	-------------	---

Function Class:

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

7.1.8 Variable Documentation

7.1.8.1 ADCDriver ADCD1

ADC1 driver identifier.

7.2 CAN Driver

Generic CAN Driver.

7.2.1 Detailed Description

Generic CAN Driver.

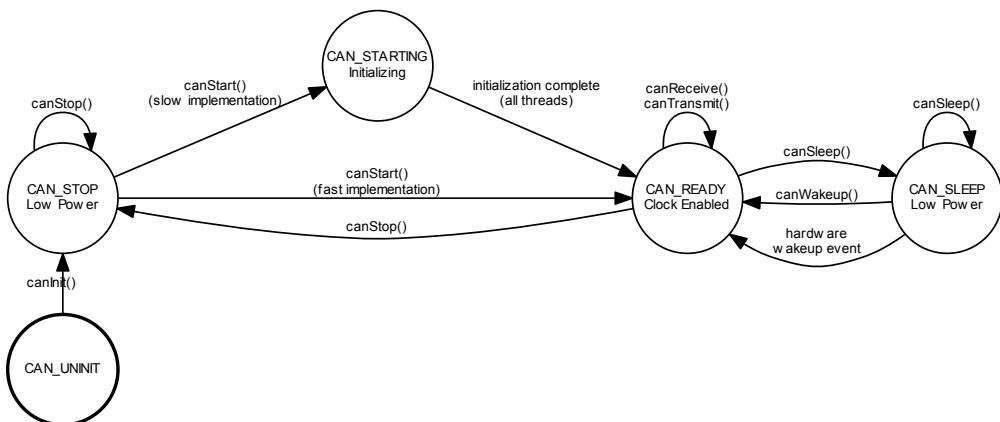
This module implements a generic CAN (Controller Area Network) driver allowing the exchange of information at frame level.

Precondition

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

7.2.2 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).



Macros

- `#define CAN_ANY_MAILBOX 0`
Special mailbox identifier.
- `#define CAN_TX_MAILBOXES 1`
Number of transmit mailboxes.
- `#define CAN_RX_MAILBOXES 1`
Number of receive mailboxes.

CAN status flags

- `#define CAN_LIMIT_WARNING 1`

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

CAN configuration options

- `#define CAN_USE_SLEEP_MODE TRUE`
Sleep mode related APIs inclusion switch.

Macro Functions

- `#define CAN_MAILBOX_TO_MASK(mbx) (1 << ((mbx) - 1))`
Converts a mailbox index to a bit mask.

PLATFORM configuration options

- `#define PLATFORM_CAN_USE_CAN1 FALSE`
CAN1 driver enable switch.

Typedefs

- `typedef uint32_t canmbx_t`
Type of a transmission mailbox index.

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 *canc)`
Initializes the standard part of a `CANDriver` structure.
- `void canStart (CANDriver *canc, const CANConfig *config)`
Configures and activates the CAN peripheral.

- void `canStop (CANDriver *canp)`
Deactivates the CAN peripheral.
- `msg_t canTransmit (CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp, systime_t timeout)`
Can frame transmission.
- `msg_t canReceive (CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp, systime_t timeout)`
Can frame receive.
- void `canSleep (CANDriver *canp)`
Enters the sleep mode.
- void `canWakeup (CANDriver *canp)`
Enforces leaving the sleep mode.
- void `can_lld_init (void)`
Low level CAN driver initialization.
- void `can_lld_start (CANDriver *canp)`
Configures and activates the CAN peripheral.
- void `can_lld_stop (CANDriver *canp)`
Deactivates the CAN peripheral.
- bool `can_lld_is_tx_empty (CANDriver *canp, canmbx_t mailbox)`
Determines whether a frame can be transmitted.
- void `can_lld_transmit (CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp)`
Inserts a frame into the transmit queue.
- bool `can_lld_is_rx_nonempty (CANDriver *canp, canmbx_t mailbox)`
Determines whether a frame has been received.
- void `can_lld_receive (CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp)`
Receives a frame from the input queue.
- void `can_lld_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_READY = 3,`
`CAN_SLEEP = 4 }`
Driver state machine possible states.

Variables

- `CANDriver CAND1`
CAN1 driver identifier.

7.2.3 Macro Definition Documentation

7.2.3.1 #define CAN_LIMIT_WARNING 1

Errors rate warning.

7.2.3.2 #define CAN_LIMIT_ERROR 2

Errors rate error.

7.2.3.3 #define CAN_BUS_OFF_ERROR 4

Bus off condition reached.

7.2.3.4 #define CAN_FRAMING_ERROR 8

Framing error of some kind on the CAN bus.

7.2.3.5 #define CAN_OVERFLOW_ERROR 16

Overflow in receive queue.

7.2.3.6 #define CAN_ANY_MAILBOX 0

Special mailbox identifier.

7.2.3.7 #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_SUPPORTS_SLEEP` exported by the underlying implementation.

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

Converts a mailbox index to a bit mask.

7.2.3.9 #define CAN_TX_MAILBOXES 1

Number of transmit mailboxes.

7.2.3.10 #define CAN_RX_MAILBOXES 1

Number of receive mailboxes.

7.2.3.11 #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`.

7.2.4 Typedef Documentation**7.2.4.1 typedef uint32_t canmbx_t**

Type of a transmission mailbox index.

7.2.5 Enumeration Type Documentation

7.2.5.1 enum canstate_t

Driver state machine possible states.

Enumerator

CAN_UNINIT Not initialized.

CAN_STOP Stopped.

CAN_STARTING Starting.

CAN_READY Ready.

CAN_SLEEP Sleep state.

7.2.6 Function Documentation

7.2.6.1 void canInit(void)

CAN Driver initialization.

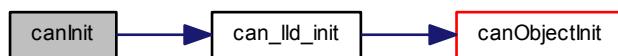
Note

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:



7.2.6.2 void canObjectInit(CANDriver * canp)

Initializes the standard part of a [CANDriver](#) structure.

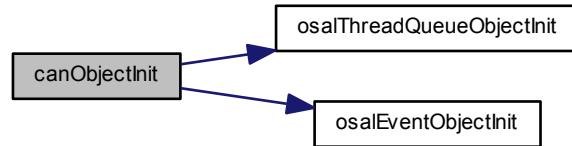
Parameters

out	<i>canp</i>	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:



7.2.6.3 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\(\)](#).

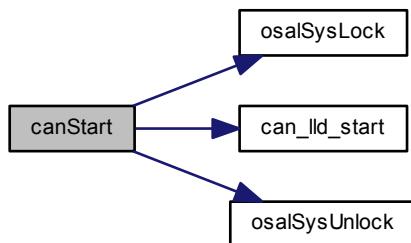
Parameters

in	<i>canp</i>	pointer to the CANDriver object
in	<i>config</i>	pointer to the CANConfig 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:



7.2.6.4 void canStop (CANDriver * *canp*)

Deactivates the CAN peripheral.

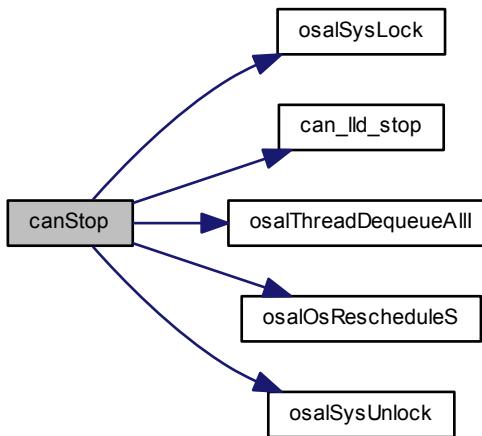
Parameters

in	<i>canp</i>	pointer to the CANDriver object
----	-------------	---

Function Class:

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

Here is the call graph for this function:



7.2.6.5 msg_t canTransmit (CANDriver * *canp*, canmbx_t *mailbox*, const CANTxFrame * *ctfp*, systime_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

in	<i>canp</i>	pointer to the CANDriver object
in	<i>mailbox</i>	mailbox number, CAN_ANY_MAILBOX for any mailbox
in	<i>ctfp</i>	pointer to the CAN frame to be transmitted
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <i>TIME_IMMEDIATE</i> immediate timeout. • <i>TIME_INFINITE</i> no timeout.

Returns

The operation result.

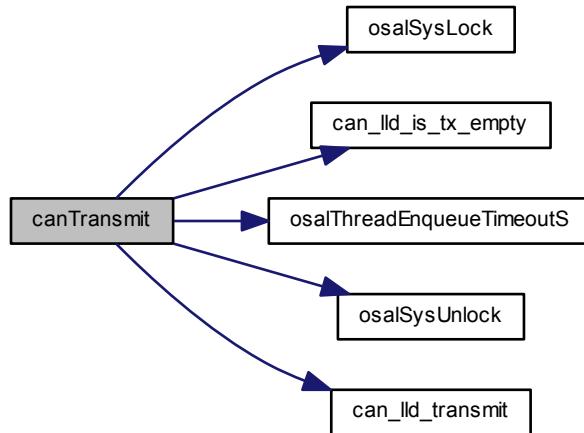
Return values

<i>MSG_OK</i>	the frame has been queued for transmission.
<i>MSG_TIMEOUT</i>	The operation has timed out.
<i>MSG_RESET</i>	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:



7.2.6.6 `msg_t canReceive(CANDriver * canp, canmbx_t mailbox, CANRxFrame * crfp, systime_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

in	<i>canp</i>	pointer to the CANDriver object
in	<i>mailbox</i>	mailbox number, <code>CAN_ANY_MAILBOX</code> for any mailbox
out	<i>crfp</i>	pointer to the buffer where the CAN frame is copied
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout (useful in an event driven scenario where a thread never blocks for I/O). • <code>TIME_INFINITE</code> no timeout.

Returns

The operation result.

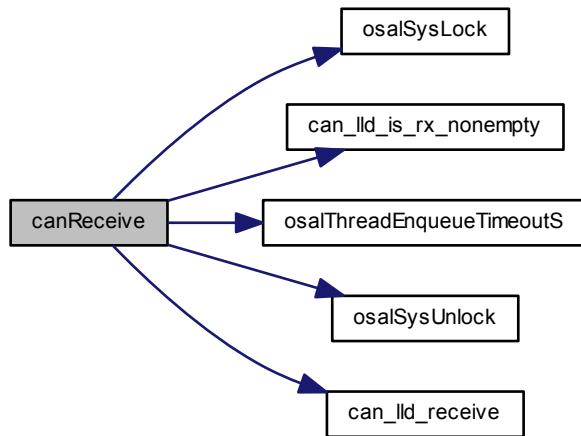
Return values

<i>MSG_OK</i>	a frame has been received and placed in the buffer.
<i>MSG_TIMEOUT</i>	The operation has timed out.
<i>MSG_RESET</i>	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:

**7.2.6.7 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_SUPPORTS_SLEEP` mode must be supported by the low level driver.

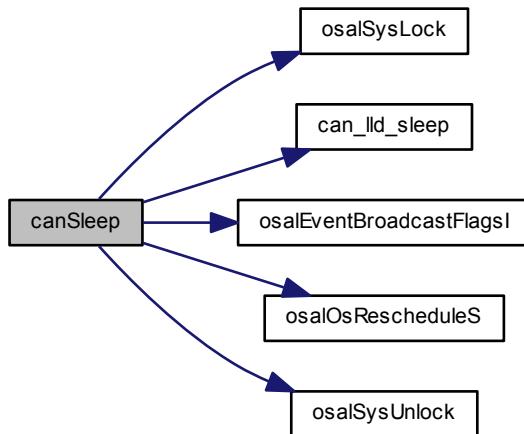
Parameters

in	<i>canp</i>	pointer to the CANDriver object
----	-------------	---

Function Class:

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

Here is the call graph for this function:



7.2.6.8 void canWakeup (**CANDriver** * *canp*)

Enforces leaving the sleep mode.

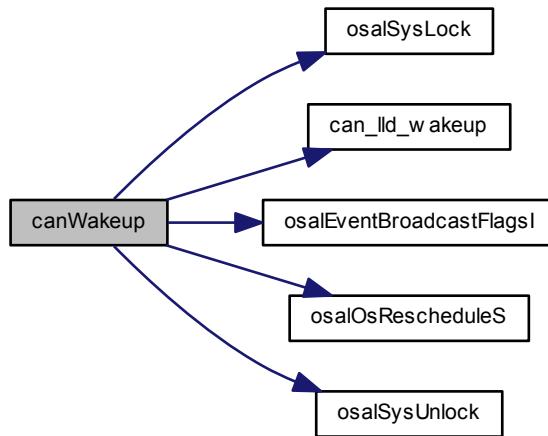
Note

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

Parameters

in	<i>canp</i>	pointer to the CANDriver object
----	-------------	--

Here is the call graph for this function:



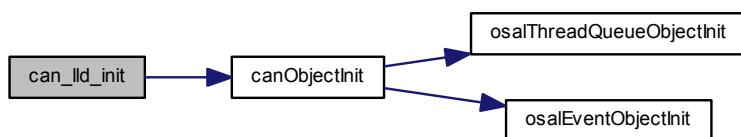
7.2.6.9 void can_Ild_init(void)

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:



7.2.6.10 void can_Ild_start(CANDriver * canp)

Configures and activates the CAN peripheral.

Parameters

in	<code>canp</code>	pointer to the <code>CANDriver</code> object
----	-------------------	--

Function Class:

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

7.2.6.11 void can_lld_stop (CANDriver * *canp*)

Deactivates the CAN peripheral.

Parameters

in	<i>canp</i>	pointer to the CANDriver object
----	-------------	---

Function Class:

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

7.2.6.12 bool can_lld_is_tx_empty (CANDriver * *canp*, canmbx_t *mailbox*)

Determines whether a frame can be transmitted.

Parameters

in	<i>canp</i>	pointer to the CANDriver object
in	<i>mailbox</i>	mailbox number, CAN_ANY_MAILBOX for any mailbox

Returns

The queue space availability.

Return values

<i>FALSE</i>	no space in the transmit queue.
<i>TRUE</i>	transmit slot available.

Function Class:

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

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

Inserts a frame into the transmit queue.

Parameters

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

Function Class:

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

7.2.6.14 bool can_lld_is_rx_nonempty (CANDriver * *canp*, canmbx_t *mailbox*)

Determines whether a frame has been received.

Parameters

in	<i>canp</i>	pointer to the CANDriver object
in	<i>mailbox</i>	mailbox number, CAN_ANY_MAILBOX for any mailbox

Returns

The queue space availability.

Return values

<i>FALSE</i>	no space in the transmit queue.
<i>TRUE</i>	transmit slot available.

Function Class:

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

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

Receives a frame from the input queue.

Parameters

in	<i>canp</i>	pointer to the CANDriver object
in	<i>mailbox</i>	mailbox number, CAN_ANY_MAILBOX for any mailbox
out	<i>crfp</i>	pointer to the buffer where the CAN frame is copied

Function Class:

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

7.2.6.16 void can_lld_sleep (**CANDriver * *canp*)**

Enters the sleep mode.

Parameters

in	<i>canp</i>	pointer to the CANDriver object
----	-------------	--

Function Class:

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

7.2.6.17 void can_lld_wakeup (**CANDriver * *canp*)**

Enforces leaving the sleep mode.

Parameters

in	<i>canp</i>	pointer to the CANDriver object
----	-------------	--

Function Class:

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

7.2.7 Variable Documentation**7.2.7.1 CANDriver CAND1**

CAN1 driver identifier.

7.3 DAC Driver

Generic DAC driver.

7.3.1 Detailed Description

Generic DAC driver.

This module implements a generic DAC (Digital to Analog Converter) driver.

Precondition

In order to use the MAC 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.

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 _dac_wait_s(dacp) osalThreadSuspendS(&(dacp)->thread)`
Waits for operation completion.
- `#define _dac_reset_i(dacp) osalThreadResumeI(&(dacp)->thread, MSG_RESET)`
Resumes a thread waiting for a conversion completion.
- `#define _dac_reset_s(dacp) osalThreadResumeS(&(dacp)->thread, MSG_RESET)`
Resumes a thread waiting for a conversion completion.
- `#define _dac_wakeup_isr(dacp)`
Wakes up the waiting thread.
- `#define _dac_timeout_isr(dacp)`
Wakes up the waiting thread with a timeout message.
- `#define _dac_isr_half_code(dacp)`
Common ISR code, half buffer event.
- `#define _dac_isr_full_code(dacp)`
Common ISR code, full buffer event.
- `#define _dac_isr_error_code(dacp, err)`
Common ISR code, error event.

Configuration options

- `#define PLATFORM_DAC_USE_DAC1 FALSE`
DAC1 CH1 driver enable switch.

Typedefs

- `typedef uint32_t dacchannel_t`
Type of a DAC channel index.
- `typedef struct DACDriver DACDriver`
Type of a structure representing an DAC driver.
- `typedef uint16_t dacsample_t`
Type representing a DAC sample.
- `typedef void(* daccallback_t) (DACDriver *dacp, const dacsample_t *buffer, size_t n)`
DAC notification callback type.
- `typedef void(* dacerrorcallback_t) (DACDriver *dacp, dacerror_t err)`
ADC error callback type.

Data Structures

- `struct DACConversionGroup`
DAC Conversion group structure.
- `struct DACConfig`
Driver configuration structure.
- `struct DACDriver`
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, const dacsample_t *samples, size_t depth)`
Starts a DAC conversion.
- `void dacStartConversionl (DACDriver *dacp, const DACConversionGroup *grpp, const dacsample_t *samples, size_t depth)`
Starts a DAC conversion.
- `void dacStopConversion (DACDriver *dacp)`
Stops an ongoing conversion.
- `void dacStopConversionl (DACDriver *dacp)`
Stops an ongoing conversion.
- `msg_t dacConvert (DACDriver *dacp, const DACConversionGroup *grpp, const dacsample_t *samples, size_t depth)`
Performs a DAC conversion.
- `void dacAcquireBus (DACDriver *dacp)`
Gains exclusive access to the DAC bus.
- `void dacReleaseBus (DACDriver *dacp)`
Releases exclusive access to the DAC bus.

- void `dac_lld_init` (void)
Low level DAC driver initialization.
- void `dac_lld_start` (`DACDriver` *`dacp`)
Configures and activates the DAC peripheral.
- void `dac_lld_stop` (`DACDriver` *`dacp`)
Deactivates the DAC peripheral.
- void `dac_lld_put_channel` (`DACDriver` *`dacp`, `dacchannel_t` `channel`, `dacsample_t` `sample`)
Outputs a value directly on a DAC channel.
- void `dac_lld_start_conversion` (`DACDriver` *`dacp`)
Starts a DAC conversion.
- void `dac_lld_stop_conversion` (`DACDriver` *`dacp`)
Stops an ongoing conversion.

Enumerations

- enum `dacstate_t` {
`DAC_UNINIT` = 0, `DAC_STOP` = 1, `DAC_READY` = 2, `DAC_ACTIVE` = 3,
`DAC_COMPLETE` = 4, `DAC_ERROR` = 5 }
Driver state machine possible states.
- enum `dacerror_t` { `DAC_ERR_DMAFAILURE` = 0, `DAC_ERR_UNDERFLOW` = 1 }
Possible DAC failure causes.

Variables

- `DACDriver DACD1`
DAC1 driver identifier.

7.3.2 Macro Definition Documentation

7.3.2.1 #define DAC_USE_WAIT TRUE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.3.2.2 #define DAC_USE_MUTUAL_EXCLUSION TRUE

Enables the `dacAcquireBus()` and `dacReleaseBus()` APIs.

Note

Disabling this option saves both code and data space.

7.3.2.3 #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	<i>dacp</i>	pointer to the DACDriver object
----	-------------	---

Function Class:

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

7.3.2.4 #define _dac_reset_i(*dacp*) osalThreadResumeI(&(*dacp*)>thread, MSG_RESET)

Resumes a thread waiting for a conversion completion.

Parameters

in	<i>dacp</i>	pointer to the DACDriver object
----	-------------	---

Function Class:

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

7.3.2.5 #define _dac_reset_s(*dacp*) osalThreadResumeS(&(*dacp*)>thread, MSG_RESET)

Resumes a thread waiting for a conversion completion.

Parameters

in	<i>dacp</i>	pointer to the DACDriver object
----	-------------	---

Function Class:

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

7.3.2.6 #define _dac_wakeup_isr(*dacp*)**Value:**

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

Wakes up the waiting thread.

Parameters

in	<i>dacp</i>	pointer to the DACDriver object
----	-------------	---

Function Class:

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

7.3.2.7 #define _dac_timeout_isr(*dacp*)**Value:**

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

Wakes up the waiting thread with a timeout message.

Parameters

in	<i>dacp</i>	pointer to the DACDriver object
----	-------------	---

Function Class:

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

7.3.2.8 #define _dac_isr_half_code(*dacp*)**Value:**

```
{
    if ((dacp)->grpp->end_cb != NULL) {
        (dacp)->grpp->end_cb(dacp, (dacp)->samples, (dacp)->depth / 2);
    }
}
```

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	<i>dacp</i>	pointer to the DACDriver object
----	-------------	---

Function Class:

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

7.3.2.9 #define _dac_isr_full_code(*dacp*)**Value:**

```
{
    if ((dacp)->grpp->end_cb != NULL) {
        if ((dacp)->depth > 1) {
            /* Invokes the callback passing the 2nd half of the buffer.*/
            size_t half = (dacp)->depth / 2;
            size_t half_index = half * (dacp)->grpp->num_channels;
            (dacp)->grpp->end_cb(dacp, (dacp)->samples + half_index, half);
        }
        else {
            /* Invokes the callback passing the whole buffer.*/
            (dacp)->grpp->end_cb(dacp, (dacp)->samples, (dacp)->depth);
        }
    }
}
```

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

in	<i>dacp</i>	pointer to the DACDriver object
----	-------------	---

Function Class:

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

7.3.2.10 #define _dac_isr_error_code(*dacp, err*)**Value:**

```
{
    dac_lld_stop_conversion(dacp);
    if ((dacp)->grpp->error_cb != NULL) {
        (dacp)->state = DAC\_ERROR;
        (dacp)->grpp->error_cb(dacp, err);
        if ((dacp)->state == DAC\_ERROR)
            (dacp)->state = DAC\_READY;
    }
    (dacp)->grpp = NULL;
    \_dac\_timeout\_isr(dacp);
}
```

Common ISR code, error event.

This code handles the portable part of the ISR code:

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

Note

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

Parameters

in	<i>dacp</i>	pointer to the DACDriver object
in	<i>err</i>	platform dependent error code

Function Class:

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

7.3.2.11 #define DAC_MAX_CHANNELS 2

Maximum number of DAC channels per unit.

7.3.2.12 #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.

7.3.3 Typedef Documentation**7.3.3.1 typedef uint32_t dacchannel_t**

Type of a DAC channel index.

7.3.3.2 typedef struct DACDriver DACDriver

Type of a structure representing an DAC driver.

7.3.3.3 typedef uint16_t dacsample_t

Type representing a DAC sample.

7.3.3.4 typedef void(* daccallback_t) (DACDriver *dacp, const dacsample_t *buffer, size_t n)

DAC notification callback type.

Parameters

in	<i>dacp</i>	pointer to the DACDriver object triggering the
in	<i>buffer</i>	pointer to the next semi-buffer to be filled
in	<i>n</i>	number of buffer rows available starting from <i>buffer</i> callback

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

ADC error callback type.

Parameters

in	<i>dacp</i>	pointer to the DACDriver object triggering the callback
in	<i>err</i>	ADC error code

7.3.4 Enumeration Type Documentation

7.3.4.1 enum dacstate_t

Driver state machine possible states.

Enumerator

- DAC_UNINIT** Not initialized.
- DAC_STOP** Stopped.
- DAC_READY** Ready.
- DAC_ACTIVE** Exchanging data.
- DAC_COMPLETE** Asynchronous operation complete.
- DAC_ERROR** Error.

7.3.4.2 enum dacerror_t

Possible DAC failure causes.

Note

Error codes are architecture dependent and should not relied upon.

Enumerator

- DAC_ERR_DMAFAILURE** DMA operations failure.
- DAC_ERR_UNDERFLOW** DAC overflow condition.

7.3.5 Function Documentation

7.3.5.1 void dacInit(void)

DAC Driver initialization.

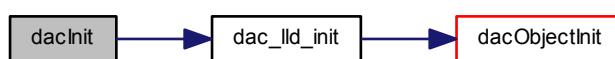
Note

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:



7.3.5.2 void dacObjectInit (DACDriver * *dacp*)

Initializes the standard part of a [DACDriver](#) structure.

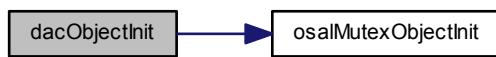
Parameters

out	<i>dacp</i>	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:



7.3.5.3 void dacStart (DACDriver * *dacp*, const DACConfig * *config*)

Configures and activates the DAC peripheral.

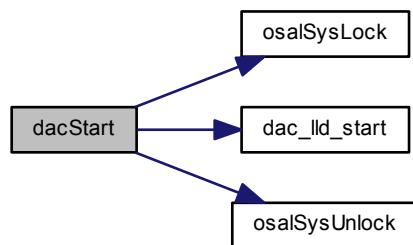
Parameters

in	<i>dacp</i>	pointer to the DACDriver object
in	<i>config</i>	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:



7.3.5.4 void dacStop (**DACDriver** * *dacp*)

Deactivates the DAC peripheral.

Note

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

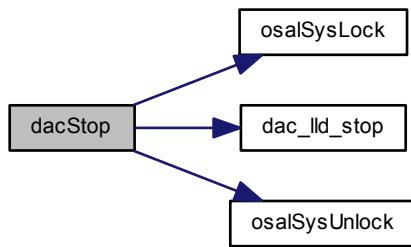
Parameters

in	<i>dacp</i>	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:



7.3.5.5 void dacPutChannelX (**DACDriver** * *dacp*, **dacchannel_t** *channel*, **dacsample_t** *sample*)

Outputs a value directly on a DAC channel.

Parameters

in	<i>dacp</i>	pointer to the DACDriver object
in	<i>channel</i>	DAC channel number
in	<i>sample</i>	value to be output

Function Class:

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

Here is the call graph for this function:



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

Starts a DAC conversion.

Starts an asynchronous conversion operation.

Note

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

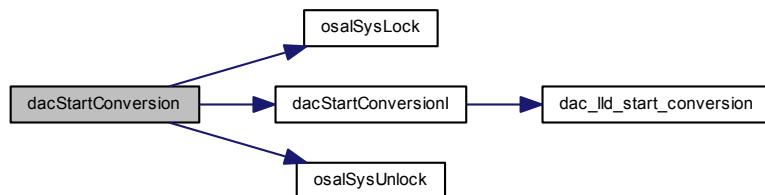
Parameters

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

Function Class:

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

Here is the call graph for this function:



7.3.5.7 void dacStartConversionI (DACDriver * *dacp*, const DACConversionGroup * *grpp*, const dacsample_t * *samples*, size_t *depth*)

Starts a DAC conversion.

Starts an asynchronous conversion operation.

Postcondition

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

Note

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

Parameters

in	<i>dacp</i>	pointer to the <code>DACDriver</code> object
in	<i>grpp</i>	pointer to a <code>DACConversionGroup</code> object
in	<i>samples</i>	pointer to the samples buffer
in	<i>depth</i>	buffer depth (matrix rows number). The buffer depth must be one or an even number.

Function Class:

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

Here is the call graph for this function:



7.3.5.8 void dacStopConversion (`DACDriver` * *dacp*)

Stops an ongoing conversion.

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

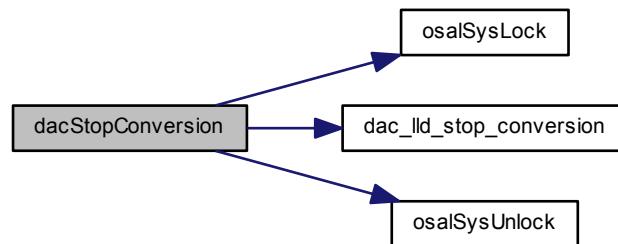
Parameters

in	<i>dacp</i>	pointer to the <code>DACDriver</code> 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:



7.3.5.9 void dacStopConversion(**DACDriver** * *dacp*)

Stops an ongoing conversion.

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

Parameters

in	<i>dacp</i>	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:



7.3.5.10 msg_t dacConvert(**DACDriver** * *dacp*, const **DACConversionGroup** * *grpp*, const **dacsample_t** * *samples*, size_t *depth*)

Performs a DAC conversion.

Performs a synchronous conversion operation.

Note

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

Parameters

in	<i>dacp</i>	pointer to the DACDriver object
in	<i>grpp</i>	pointer to a DACConversionGroup object
out	<i>samples</i>	pointer to the samples buffer
in	<i>depth</i>	buffer depth (matrix rows number). The buffer depth must be one or an even number.

Returns

The operation result.

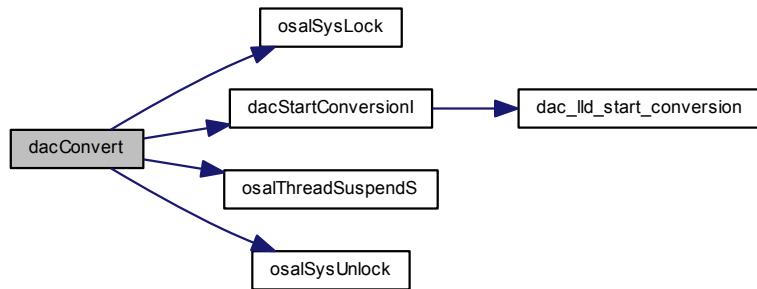
Return values

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

Function Class:

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

Here is the call graph for this function:

**7.3.5.11 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	<i>dacp</i>	pointer to the <code>DACDriver</code> 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:



7.3.5.12 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	<i>dacp</i>	pointer to the <code>DACDriver</code> 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:



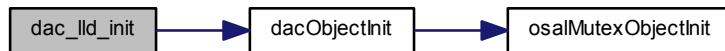
7.3.5.13 void dac_lld_init (void)

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:



7.3.5.14 void dac_lld_start (DACDriver * *dacp*)

Configures and activates the DAC peripheral.

Parameters

in	<i>dacp</i>	pointer to the DACDriver object
----	-------------	---

Function Class:

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

7.3.5.15 void dac_lld_stop (DACDriver * *dacp*)

Deactivates the DAC peripheral.

Parameters

in	<i>dacp</i>	pointer to the DACDriver object
----	-------------	---

Function Class:

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

7.3.5.16 void dac_lld_put_channel (DACDriver * *dacp*, dacchannel_t *channel*, dacsample_t *sample*)

Outputs a value directly on a DAC channel.

Parameters

in	<i>dacp</i>	pointer to the DACDriver object
in	<i>channel</i>	DAC channel number
in	<i>sample</i>	value to be output

Function Class:

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

7.3.5.17 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	<i>dacp</i>	pointer to the DACDriver object
----	-------------	---

Function Class:

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

7.3.5.18 void dac_lld_stop_conversion(DACDriver * *dacp*)

Stops an ongoing conversion.

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

Parameters

in	<i>dacp</i>	pointer to the <code>DACDriver</code> 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.

7.3.6 Variable Documentation

7.3.6.1 DACDriver DACD1

DAC1 driver identifier.

7.4 EXT Driver

Generic EXT Driver.

7.4.1 Detailed Description

Generic EXT Driver.

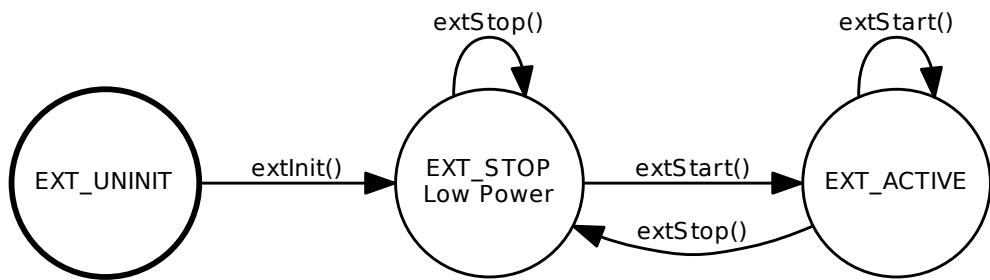
This module implements a generic EXT (EXTernal) driver.

Precondition

In order to use the EXT driver the `HAL_USE_EXT` 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).



7.4.3 EXT Operations.

This driver abstracts generic external interrupt sources, a callback is invoked when a programmable transition is detected on one of the configured channels. Several channel modes are possible.

- **`EXT_CH_MODE_DISABLED`**, channel not used.
- **`EXT_CH_MODE_RISING_EDGE`**, callback on a rising edge.
- **`EXT_CH_MODE_FALLING_EDGE`**, callback on a falling edge.
- **`EXT_CH_MODE_BOTH_EDGES`**, callback on a both edges.

Macros

- `#define EXT_MAX_CHANNELS 20`

Available number of EXT channels.

EXT channel modes

- `#define EXT_CH_MODE_EDGES_MASK 3U`
Mask of edges field.
- `#define EXT_CH_MODE_DISABLED 0U`
Channel disabled.
- `#define EXT_CH_MODE_RISING_EDGE 1U`
Rising edge callback.
- `#define EXT_CH_MODE_FALLING_EDGE 2U`
Falling edge callback.
- `#define EXT_CH_MODE_BOTH_EDGES 3U`
Both edges callback.
- `#define EXT_CH_MODE_AUTOSTART 4U`
Channel started automatically on driver start.

Macro Functions

- `#define extChannelEnable(extp, channel) ext_lld_channel_enable(extp, channel)`
Enables an EXT channel.
- `#define extChannelDisable(extp, channel) ext_lld_channel_disable(extp, channel)`
Disables an EXT channel.
- `#define extSetChannelMode(extp, channel, extcp)`
Changes the operation mode of a channel.

PLATFORM configuration options

- `#define PLATFORM_EXT_USE_EXT1 FALSE`
EXT driver enable switch.

Typedefs

- `typedef struct EXTDriver EXTDriver`
Type of a structure representing a EXT driver.
- `typedef uint32_t expchannel_t`
EXT channel identifier.
- `typedef void(* extcallback_t) (EXTDriver *extp, expchannel_t channel)`
Type of an EXT generic notification callback.

Data Structures

- `struct EXTChannelConfig`
Channel configuration structure.
- `struct EXTConfig`
Driver configuration structure.
- `struct EXTDriver`
Structure representing an EXT driver.

Functions

- void `extInit` (void)

EXT Driver initialization.
- void `extObjectInit` (EXTDriver *extp)

Initializes the standard part of a `EXTDriver` structure.
- void `extStart` (EXTDriver *extp, const EXTConfig *config)

Configures and activates the EXT peripheral.
- void `extStop` (EXTDriver *extp)

Deactivates the EXT peripheral.
- void `extChannelEnable` (EXTDriver *extp, expchannel_t channel)

Enables an EXT channel.
- void `extChannelDisable` (EXTDriver *extp, expchannel_t channel)

Disables an EXT channel.
- void `extSetChannelModel` (EXTDriver *extp, expchannel_t channel, const EXTChannelConfig *extcp)

Changes the operation mode of a channel.
- void `ext_lld_init` (void)

Low level EXT driver initialization.
- void `ext_lld_start` (EXTDriver *extp)

Configures and activates the EXT peripheral.
- void `ext_lld_stop` (EXTDriver *extp)

Deactivates the EXT peripheral.
- void `ext_lld_channel_enable` (EXTDriver *extp, expchannel_t channel)

Enables an EXT channel.
- void `ext_lld_channel_disable` (EXTDriver *extp, expchannel_t channel)

Disables an EXT channel.

Enumerations

- enum `extstate_t` { `EXT_UNINIT` = 0, `EXT_STOP` = 1, `EXT_ACTIVE` = 2 }

Driver state machine possible states.

Variables

- EXTDriver `EXTD1`

EXT1 driver identifier.

7.4.4 Macro Definition Documentation

7.4.4.1 #define EXT_CH_MODE_EDGES_MASK 3U

Mask of edges field.

7.4.4.2 #define EXT_CH_MODE_DISABLED 0U

Channel disabled.

7.4.4.3 #define EXT_CH_MODE_RISING_EDGE 1U

Rising edge callback.

7.4.4.4 #define EXT_CH_MODE_FALLING_EDGE 2U

Falling edge callback.

7.4.4.5 #define EXT_CH_MODE_BOTH_EDGES 3U

Both edges callback.

7.4.4.6 #define EXT_CH_MODE_AUTOSTART 4U

Channel started automatically on driver start.

7.4.4.7 #define extChannelEnable(extp, channel) ext_llid_channel_enable(extp, channel)

Enables an EXT channel.

Parameters

in	<i>extp</i>	pointer to the <code>EXTDriver</code> object
in	<i>channel</i>	channel to be enabled

Function Class:

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

7.4.4.8 #define extChannelDisable(extp, channel) ext_llid_channel_disable(extp, channel)

Disables an EXT channel.

Parameters

in	<i>extp</i>	pointer to the <code>EXTDriver</code> object
in	<i>channel</i>	channel to be disabled

Function Class:

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

7.4.4.9 #define extSetChannelMode(extp, channel, extcp)

Value:

```
{
    \osalSysLock();
    \
    extSetChannelModeI(extp, channel, extcp);
    \
    \osalSysUnlock();
}
```

Changes the operation mode of a channel.

Note

This function attempts to write over the current configuration structure that must have been not declared constant. This violates the `const` qualifier in `extStart()` but it is intentional. This function cannot be used if the configuration structure is declared `const`.

Parameters

in	<i>extp</i>	pointer to the <code>EXTDriver</code> object
in	<i>channel</i>	channel to be changed
in	<i>extcp</i>	new configuration for the channel

Function Class:

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

7.4.4.10 #define EXT_MAX_CHANNELS 20

Available number of EXT channels.

7.4.4.11 #define PLATFORM_EXT_USE_EXT1 FALSE

EXT driver enable switch.

If set to `TRUE` the support for EXT1 is included.

Note

The default is `FALSE`.

7.4.5 Typedef Documentation**7.4.5.1 typedef struct EXTDriver EXTDriver**

Type of a structure representing a EXT driver.

7.4.5.2 typedef uint32_t expchannel_t

EXT channel identifier.

7.4.5.3 typedef void(* extcallback_t)(EXTDriver *extp, expchannel_t channel)

Type of an EXT generic notification callback.

Parameters

in	<i>extp</i>	pointer to the <code>EXTDriver</code> object triggering the callback
----	-------------	--

7.4.6 Enumeration Type Documentation**7.4.6.1 enum extstate_t**

Driver state machine possible states.

Enumerator

`EXT_UNINIT` Not initialized.

`EXT_STOP` Stopped.

`EXT_ACTIVE` Active.

7.4.7 Function Documentation

7.4.7.1 void extInit(void)

EXT Driver initialization.

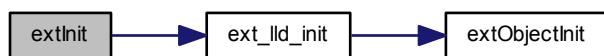
Note

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:



7.4.7.2 void extObjectInit(EXTDriver * extp)

Initializes the standard part of a `EXTDriver` structure.

Parameters

out	<code>extp</code>	pointer to the <code>EXTDriver</code> object
-----	-------------------	--

Function Class:

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

7.4.7.3 void extStart(EXTDriver * extp, const EXTConfig * config)

Configures and activates the EXT peripheral.

Postcondition

After activation all EXT channels are in the disabled state, use `extChannelEnable()` in order to activate them.

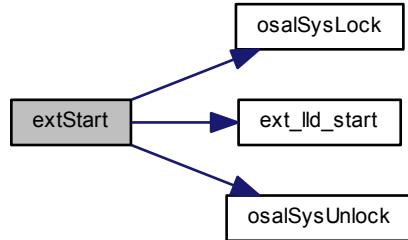
Parameters

in	<code>extp</code>	pointer to the <code>EXTDriver</code> object
in	<code>config</code>	pointer to the <code>EXTConfig</code> object

Function Class:

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

Here is the call graph for this function:



7.4.7.4 void extStop (EXTDriver * extp)

Deactivates the EXT peripheral.

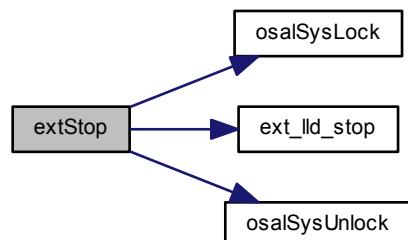
Parameters

in	<code>extp</code>	pointer to the <code>EXTDriver</code> object
----	-------------------	--

Function Class:

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

Here is the call graph for this function:



7.4.7.5 void extChannelEnable (EXTDriver * extp, expchannel_t channel)

Enables an EXT channel.

Precondition

The channel must not be in `EXT_CH_MODE_DISABLED` mode.

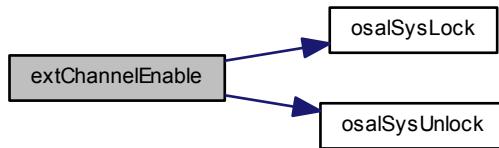
Parameters

in	<i>extp</i>	pointer to the <code>EXTDriver</code> object
in	<i>channel</i>	channel to be enabled

Function Class:

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

Here is the call graph for this function:

**7.4.7.6 void extChannelDisable (`EXTDriver` * *extp*, `expchannel_t` *channel*)**

Disables an EXT channel.

Precondition

The channel must not be in `EXT_CH_MODE_DISABLED` mode.

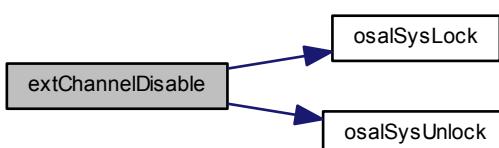
Parameters

in	<i>extp</i>	pointer to the <code>EXTDriver</code> object
in	<i>channel</i>	channel to be disabled

Function Class:

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

Here is the call graph for this function:



7.4.7.7 void extSetChannelModel (EXTDriver * extp, expchannel_t channel, const EXTChannelConfig * extcp)

Changes the operation mode of a channel.

Note

This function attempts to write over the current configuration structure that must have been not declared constant. This violates the `const` qualifier in `extStart()` but it is intentional.

This function cannot be used if the configuration structure is declared `const`.

The effect of this function on constant configuration structures is not defined.

Parameters

in	<code>extp</code>	pointer to the <code>EXTDriver</code> object
in	<code>channel</code>	channel to be changed
in	<code>extcp</code>	new configuration for the channel

Function 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:



7.4.7.8 void ext_lld_init (void)

Low level EXT driver initialization.

Function Class:

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

Here is the call graph for this function:



7.4.7.9 void ext_lld_start (EXTDriver * extp)

Configures and activates the EXT peripheral.

Parameters

in	<i>extp</i>	pointer to the EXTDriver object
----	-------------	---

Function Class:

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

7.4.7.10 void ext_lld_stop (EXTDriver * *extp*)

Deactivates the EXT peripheral.

Parameters

in	<i>extp</i>	pointer to the EXTDriver object
----	-------------	---

Function Class:

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

7.4.7.11 void ext_lld_channel_enable (EXTDriver * *extp*, expchannel_t *channel*)

Enables an EXT channel.

Parameters

in	<i>extp</i>	pointer to the EXTDriver object
in	<i>channel</i>	channel to be enabled

Function Class:

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

7.4.7.12 void ext_lld_channel_disable (EXTDriver * *extp*, expchannel_t *channel*)

Disables an EXT channel.

Parameters

in	<i>extp</i>	pointer to the EXTDriver object
in	<i>channel</i>	channel to be disabled

Function Class:

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

7.4.8 Variable Documentation**7.4.8.1 EXTDriver EXTD1**

EXT1 driver identifier.

7.5 GPT Driver

Generic GPT Driver.

7.5.1 Detailed Description

Generic GPT Driver.

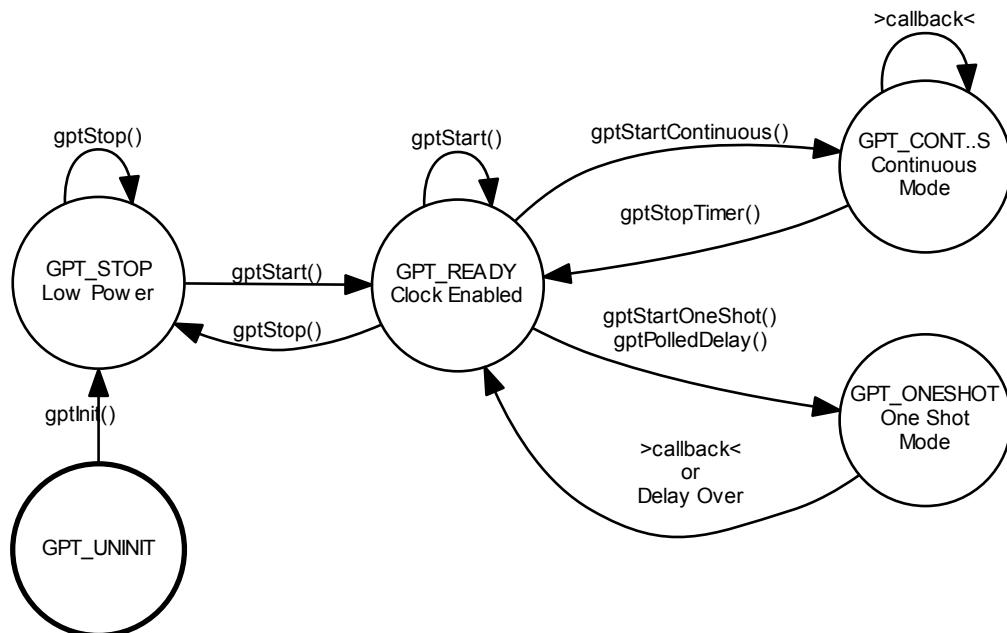
This module implements a generic GPT (General Purpose Timer) driver. The timer can be programmed in order to trigger callbacks after a specified time period or continuously with a specified interval.

Precondition

In order to use the GPT driver the `HAL_USE_GPT` option must be enabled in `halconf.h`.

7.5.2 Driver State Machine

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



7.5.3 GPT Operations.

This driver abstracts a generic timer composed of:

- A clock prescaler.
- A main up counter.

- A comparator register that resets the main counter to zero when the limit is reached. A callback is invoked when this happens.

The timer can operate in three different modes:

- **Continuous Mode**, a periodic callback is invoked until the driver is explicitly stopped.
- **One Shot Mode**, a callback is invoked after the programmed period and then the timer automatically stops.
- **Delay Mode**, the timer is used for inserting a brief delay into the execution flow, no callback is invoked in this mode.

Macros

- `#define gptChangeInterval(gptp, interval)`
Changes the interval of GPT peripheral.
- `#define gptGetIntervalX(gptp) gpt_lld_get_interval(gptp)`
Returns the interval of GPT peripheral.
- `#define gptGetCounterX(gptp) gpt_lld_get_counter(gptp)`
Returns the counter value of GPT peripheral.
- `#define gpt_lld_change_interval(gptp, interval)`
Changes the interval of GPT peripheral.

PLATFORM configuration options

- `#define PLATFORM_GPT_USE_GPT1 FALSE`
GPTD1 driver enable switch.

Typedefs

- `typedef struct GPTDriver GPTDriver`
Type of a structure representing a GPT driver.
- `typedef void(* gptcallback_t) (GPTDriver *gptp)`
GPT notification callback type.
- `typedef uint32_t gptfreq_t`
GPT frequency type.
- `typedef uint16_t gptcnt_t`
GPT counter type.

Data Structures

- `struct GPTConfig`
Driver configuration structure.
- `struct GPTDriver`
Structure representing a GPT driver.

Functions

- void `gptInit` (void)

GPT Driver initialization.
- void `gptObjectInit` (GPTDriver *gptp)

Initializes the standard part of a `GPTDriver` structure.
- 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 `gptStartContinuousl` (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 `gptStartOneShotl` (GPTDriver *gptp, gptcnt_t interval)

Starts the timer in one shot mode.
- void `gptStopTimer` (GPTDriver *gptp)

Stops the timer.
- void `gptStopTimerl` (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.

7.5.4 Macro Definition Documentation

7.5.4.1 #define gptChangeInterval(*gptp*, *interval*)

Value:

```
{
    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

in	<i>gptp</i>	pointer to a GPTDriver object
in	<i>interval</i>	new cycle time in timer 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.

7.5.4.2 #define gptGetIntervalX(*gptp*) gpt_lld_get_interval(*gptp*)

Returns the interval of GPT peripheral.

Precondition

The GPT unit must be running in continuous mode.

Parameters

in	<i>gptp</i>	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.

7.5.4.3 #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	<i>gptp</i>	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.

7.5.4.4 #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.

7.5.4.5 #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

in	<i>gptp</i>	pointer to a GPTDriver object
in	<i>interval</i>	new cycle time in timer ticks

Function Class:

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

7.5.5 Typedef Documentation

7.5.5.1 `typedef struct GPTDriver GPTDriver`

Type of a structure representing a GPT driver.

7.5.5.2 `typedef void(* gptcallback_t)(GPTDriver *gptp)`

GPT notification callback type.

Parameters

in	<i>gptp</i>	pointer to a GPTDriver object
----	-------------	---

7.5.5.3 `typedef uint32_t gptfreq_t`

GPT frequency type.

7.5.5.4 `typedef uint16_t gptcnt_t`

GPT counter type.

7.5.6 Enumeration Type Documentation

7.5.6.1 `enum gptstate_t`

Driver state machine possible states.

Enumerator

GPT_UNINIT Not initialized.

GPT_STOP Stopped.

GPT_READY Ready.

GPT_CONTINUOUS Active in continuous mode.

GPT_ONESHOT Active in one shot mode.

7.5.7 Function Documentation

7.5.7.1 `void gptInit(void)`

GPT Driver initialization.

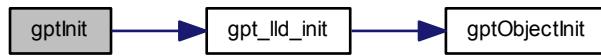
Note

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:



7.5.7.2 void gptObjectInit (GPTDriver * *gptp*)

Initializes the standard part of a `GPTDriver` structure.

Parameters

out	<i>gptp</i>	pointer to the <code>GPTDriver</code> object
-----	-------------	--

Function Class:

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

7.5.7.3 void gptStart (GPTDriver * *gptp*, const GPTConfig * *config*)

Configures and activates the GPT peripheral.

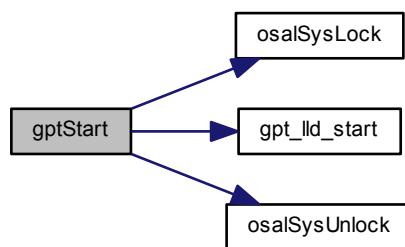
Parameters

in	<i>gptp</i>	pointer to the <code>GPTDriver</code> object
in	<i>config</i>	pointer to the <code>GPTConfig</code> object

Function Class:

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

Here is the call graph for this function:



7.5.7.4 void gptStop (GPTDriver * *gptp*)

Deactivates the GPT peripheral.

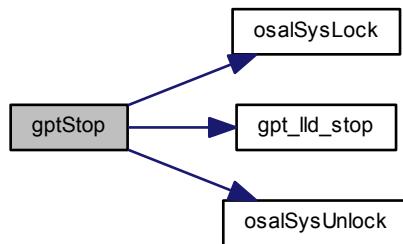
Parameters

in	<i>gptp</i>	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:



7.5.7.5 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.

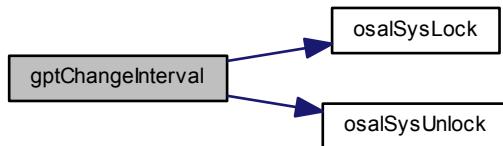
Parameters

in	<i>gptp</i>	pointer to a GPTDriver object
in	<i>interval</i>	new cycle time in timer ticks

Function Class:

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

Here is the call graph for this function:

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

Starts the timer in continuous mode.

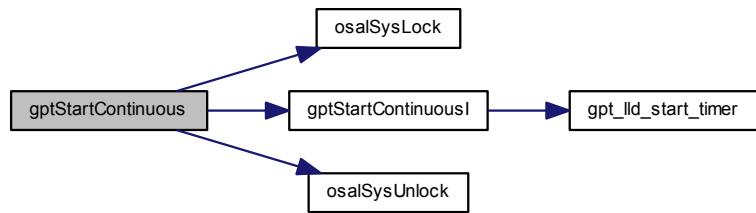
Parameters

in	<i>gptp</i>	pointer to the GPTDriver object
in	<i>interval</i>	period in ticks

Function Class:

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

Here is the call graph for this function:



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

Starts the timer in continuous mode.

Parameters

in	<i>gptp</i>	pointer to the GPTDriver object
in	<i>interval</i>	period 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.

Here is the call graph for this function:



7.5.7.8 void gptStartOneShot ([GPTDriver](#) * *gptp*, *gptcnt_t* *interval*)

Starts the timer in one shot mode.

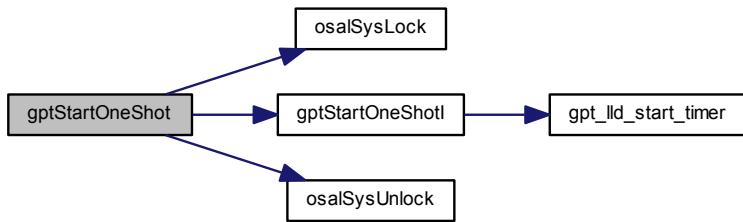
Parameters

in	<i>gptp</i>	pointer to the GPTDriver object
in	<i>interval</i>	time interval in ticks

Function Class:

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

Here is the call graph for this function:



7.5.7.9 void gptStartOneShotI ([GPTDriver](#) * *gptp*, *gptcnt_t* *interval*)

Starts the timer in one shot mode.

Parameters

in	<i>gptp</i>	pointer to the GPTDriver object
in	<i>interval</i>	time interval in ticks

Function Class:

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

Here is the call graph for this function:

**7.5.7.10 void gptStopTimer (GPTDriver * *gptp*)**

Stops the timer.

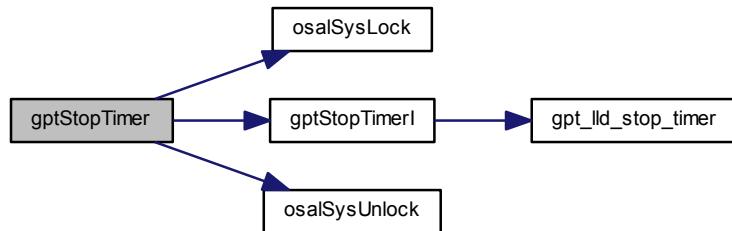
Parameters

in	<i>gptp</i>	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:

**7.5.7.11 void gptStopTimerI (GPTDriver * *gptp*)**

Stops the timer.

Parameters

in	<i>gptp</i>	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:

**7.5.7.12 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

in	<i>gptp</i>	pointer to the GPTDriver object
in	<i>interval</i>	time interval in ticks

Function Class:

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

Here is the call graph for this function:

**7.5.7.13 void gpt_lld_init (void)**

Low level GPT driver initialization.

Function Class:

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

Here is the call graph for this function:

**7.5.7.14 void gpt_lld_start (GPTDriver * gptp)**

Configures and activates the GPT peripheral.

Parameters

in	<i>gptp</i>	pointer to the GPTDriver object
----	-------------	---

Function Class:

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

7.5.7.15 void gpt_lld_stop (GPTDriver * gptp)

Deactivates the GPT peripheral.

Parameters

in	<i>gptp</i>	pointer to the GPTDriver object
----	-------------	---

Function Class:

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

7.5.7.16 void gpt_lld_start_timer (GPTDriver * gptp, gptcnt_t interval)

Starts the timer in continuous mode.

Parameters

in	<i>gptp</i>	pointer to the GPTDriver object
in	<i>interval</i>	period in ticks

Function Class:

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

7.5.7.17 void gpt_lld_stop_timer (GPTDriver * gptp)

Stops the timer.

Parameters

in	<i>gptp</i>	pointer to the GPTDriver object
----	-------------	---

Function Class:

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

7.5.7.18 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

in	<i>gptp</i>	pointer to the GPTDriver object
in	<i>interval</i>	time interval in ticks

Function Class:

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

7.5.8 Variable Documentation**7.5.8.1 GPTDriver GPTD1**

GPTD1 driver identifier.

7.6 HAL Driver

Hardware Abstraction Layer.

7.6.1 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 "3.0.2"`
HAL version string.
- `#define CH_HAL_MAJOR 3`
HAL version major number.
- `#define CH_HAL_MINOR 0`
HAL version minor number.
- `#define CH_HAL_PATCH 2`
HAL version patch number.

Return codes

- `#define HAL_SUCCESS false`
- `#define HAL_FAILED true`

Platform identification macros

- `#define PLATFORM_NAME "templates"`

Functions

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

7.6.2 Macro Definition Documentation

7.6.2.1 `#define _CHIBIOS_HAL_`

ChibiOS/HAL identification macro.

7.6.2.2 `#define CH_HAL_STABLE 1`

Stable release flag.

7.6.2.3 `#define HAL_VERSION "3.0.2"`

HAL version string.

7.6.2.4 `#define CH_HAL_MAJOR 3`

HAL version major number.

7.6.2.5 `#define CH_HAL_MINOR 0`

HAL version minor number.

7.6.2.6 `#define CH_HAL_PATCH 2`

HAL version patch number.

7.6.3 Function Documentation

7.6.3.1 `void halInit(void)`

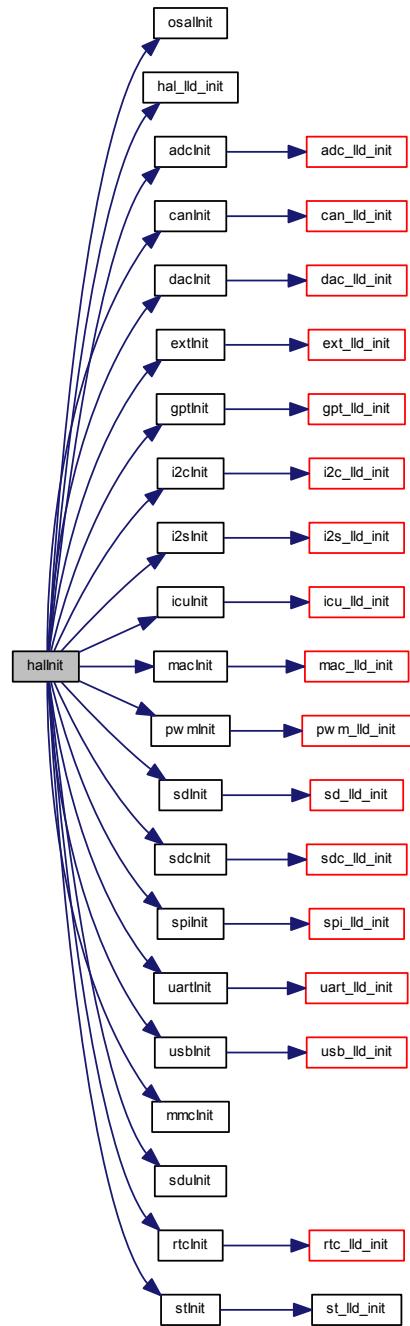
HAL initialization.

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

Function Class:

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

Here is the call graph for this function:



7.6.3.2 void hal_lld_init (void)

Low level HAL driver initialization.

Function Class:

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

7.7 Abstract I/O Channel

7.7.1 Detailed Description

This module defines an abstract interface for I/O channels by extending the `BaseSequentialStream` interface. Note that no code is present, I/O channels are just abstract interface like structures, you should look at the systems as to a set of abstract C++ classes (even if written in C). Specific device drivers can use/extend the interface and implement them.

This system has the advantage to make the access to channels independent from the implementation logic.

Macros

- `#define _base_channel_methods`
`BaseChannel specific methods.`
- `#define _base_channel_data _base_sequential_stream_data`
`BaseChannel specific data.`
- `#define _base_asynchronous_channel_methods _base_channel_methods \`
`BaseAsynchronousChannel specific methods.`
- `#define _base_asynchronous_channel_data`
`BaseAsynchronousChannel specific data.`

Macro Functions (BaseChannel)

- `#define chnPutTimeout(ip, b, time) ((ip)->vmt->putt(ip, b, time))`
`Channel blocking byte write with timeout.`
- `#define chnGetTimeout(ip, time) ((ip)->vmt->gett(ip, time))`
`Channel blocking byte read with timeout.`
- `#define chnWrite(ip, bp, n) streamWrite(ip, bp, n)`
`Channel blocking write.`
- `#define chnWriteTimeout(ip, bp, n, time) ((ip)->vmt->writet(ip, bp, n, time))`
`Channel blocking write with timeout.`
- `#define chnRead(ip, bp, n) streamRead(ip, bp, n)`
`Channel blocking read.`
- `#define chnReadTimeout(ip, bp, n, time) ((ip)->vmt->readt(ip, bp, n, time))`
`Channel blocking read with timeout.`

I/O status flags added to the event listener

- `#define CHN_NO_ERROR (eventflags_t)0`
`No pending conditions.`
- `#define CHN_CONNECTED (eventflags_t)1`
`Connection happened.`
- `#define CHN_DISCONNECTED (eventflags_t)2`
`Disconnection happened.`
- `#define CHN_INPUT_AVAILABLE (eventflags_t)4`
`Data available in the input queue.`
- `#define CHN_OUTPUT_EMPTY (eventflags_t)8`
`Output queue empty.`
- `#define CHN_TRANSMISSION_END (eventflags_t)16`
`Transmission end.`

Macro Functions (`BaseAsynchronousChannel`)

- `#define chnGetEventSource(ip) (&((ip)->event))`
Returns the I/O condition event source.
- `#define chnAddFlagsI(ip, flags)`
Adds status flags to the listeners'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.`

7.7.2 Macro Definition Documentation

7.7.2.1 `#define _base_channel_methods`

Value:

```
_base_sequential_stream_methods
/* Channel put method with timeout specification.*/
msg_t (*putt)(void *instance, uint8_t b, systime_t time);
/* Channel get method with timeout specification.*/
msg_t (*gett)(void *instance, systime_t time);
/* Channel write method with timeout specification.*/
size_t (*writet)(void *instance, const uint8_t *bp,
                  size_t n, systime_t time);
/* Channel read method with timeout specification.*/
size_t (*readt)(void *instance, uint8_t *bp, size_t n, systime_t time);
```



`BaseChannel` specific methods.

7.7.2.2 `#define _base_channel_data _base_sequential_stream_data`

`BaseChannel` specific data.

Note

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

7.7.2.3 `#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

in	<i>ip</i>	pointer to a BaseChannel or derived class
in	<i>b</i>	the byte value to be written to the channel
in	<i>time</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <i>TIME_IMMEDIATE</i> immediate timeout. • <i>TIME_INFINITE</i> no timeout.

Returns

The operation status.

Return values

<i>STM_OK</i>	if the operation succeeded.
<i>STM_TIMEOUT</i>	if the specified time expired.
<i>STM_RESET</i>	if the channel associated queue (if any) was reset.

Function Class:

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

7.7.2.4 #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

in	<i>ip</i>	pointer to a BaseChannel or derived class
in	<i>time</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <i>TIME_IMMEDIATE</i> immediate timeout. • <i>TIME_INFINITE</i> no timeout.

Returns

A byte value from the queue.

Return values

<i>STM_TIMEOUT</i>	if the specified time expired.
<i>STM_RESET</i>	if the channel associated queue (if any) has been reset.

Function Class:

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

7.7.2.5 #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

in	<i>ip</i>	pointer to a BaseChannel or derived class
out	<i>bp</i>	pointer to the data buffer
in	<i>n</i>	the maximum amount of data to be transferred

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.

7.7.2.6 #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

in	<i>ip</i>	pointer to a BaseChannel or derived class
out	<i>bp</i>	pointer to the data buffer
in	<i>n</i>	the maximum amount of data to be transferred
in	<i>time</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none">• <i>TIME_IMMEDIATE</i> immediate timeout.• <i>TIME_INFINITE</i> 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.

7.7.2.7 #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

in	<i>ip</i>	pointer to a BaseChannel or derived class
in	<i>bp</i>	pointer to the data buffer
in	<i>n</i>	the maximum amount of data to be transferred

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.

7.7.2.8 #define chnReadTimeout(*ip*, *bp*, *n*, *time*) ((*ip*)>vmt->readt(*ip*, *bp*, *n*, *time*))

Channel blocking read with timeout.

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

Parameters

in	<i>ip</i>	pointer to a BaseChannel or derived class
in	<i>bp</i>	pointer to the data buffer
in	<i>n</i>	the maximum amount of data to be transferred
in	<i>time</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <i>TIME_IMMEDIATE</i> immediate timeout. • <i>TIME_INFINITE</i> 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.

7.7.2.9 #define CHN_NO_ERROR (eventflags_t)0

No pending conditions.

7.7.2.10 #define CHN_CONNECTED (eventflags_t)1

Connection happened.

7.7.2.11 #define CHN_DISCONNECTED (eventflags_t)2

Disconnection happened.

7.7.2.12 #define CHN_INPUT_AVAILABLE (eventflags_t)4

Data available in the input queue.

7.7.2.13 #define CHN_OUTPUT_EMPTY (eventflags_t)8

Output queue empty.

7.7.2.14 #define CHN_TRANSMISSION_END (eventflags_t)16

Transmission end.

7.7.2.15 #define _base_asynchronous_channel_methods _base_channel_methods \

[BaseAsynchronousChannel](#) specific methods.

7.7.2.16 #define _base_asynchronous_channel_data

Value:

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

BaseAsynchronousChannel specific data.

7.7.2.17 #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.

7.7.2.18 #define chnAddFlagsI(ip, flags)

Value:

```
{
  osalEventBroadcastFlagsI(&(ip)->event, flags);
}
```

Adds status flags to the listeners's flags mask.

This function is usually called from the I/O ISRs in order to notify I/O conditions such as data events, errors, signal changes etc.

Parameters

in	ip	pointer to a BaseAsynchronousChannel or derived class
in	flags	condition flags to be added to the listener flags mask

Function Class:

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

7.8 Abstract Files

7.8.1 Detailed Description

This module define 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) ((ip)->vmt->getsize(ip))`
Returns the current file size.
- `#define fileStreamGetPosition(ip) ((ip)->vmt->getposition(ip))`
Returns the current file pointer position.
- `#define fileStreamSeek(ip, offset) ((ip)->vmt->lseek(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.

7.8.2 Macro Definition Documentation

7.8.2.1 #define FILE_OK STM_OK

No error return code.

7.8.2.2 #define FILE_ERROR STM_TIMEOUT

Error code from the file stream methods.

7.8.2.3 #define FILE_EOF STM_RESET

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

7.8.2.4 #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);
/* File get current position method.*/
msg_t (*getposition)(void *instance);
/* File seek method.*/
msg_t (*lseek)(void *instance, fileoffset_t offset);
```



[FileStream](#) specific methods.

7.8.2.5 #define _file_stream_data _base_sequential_stream_data

[FileStream](#) specific data.

Note

It is empty because [FileStream](#) is only an interface without implementation.

7.8.2.6 #define fileStreamWrite(ip, bp, n) streamWrite(ip, bp, n)

File stream write.

The function writes data from a buffer to a file stream.

Parameters

in	<i>ip</i>	pointer to a FileStream or derived class
in	<i>bp</i>	pointer to the data buffer
in	<i>n</i>	the maximum amount of data to be transferred

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.

7.8.2.7 #define fileStreamRead(*ip*, *bp*, *n*) streamRead(*ip*, *bp*, *n*)

File stream read.

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

Parameters

in	<i>ip</i>	pointer to a FileStream or derived class
out	<i>bp</i>	pointer to the data buffer
in	<i>n</i>	the maximum amount of data to be transferred

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.

7.8.2.8 #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

in	<i>ip</i>	pointer to a FileStream or derived class
in	<i>b</i>	the byte value to be written to the channel

Returns

The operation status.

Return values

<i>FILE_OK</i>	if the operation succeeded.
<i>FILE_ERROR</i>	operation failed.
<i>FILE_EOF</i>	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.

7.8.2.9 #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	<i>ip</i>	pointer to a FileStream or derived class
----	-----------	--

Returns

A byte value from the queue.

Return values

<i>FILE_ERROR</i>	operation failed.
<i>FILE_EOF</i>	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.

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

File Stream close.

The function closes a file stream.

Parameters

in	<i>ip</i>	pointer to a FileStream or derived class
----	-----------	--

Returns

The operation status.

Return values

<i>FILE_OK</i>	no error.
<i>FILE_ERROR</i>	operation failed.

Function Class:

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

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

Returns an implementation dependent error code.

Precondition

The previously called function must have returned FILE_ERROR.

Parameters

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

Returns

Implementation dependent error code.

Function Class:

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

7.8.2.12 #define fileStreamGetSize(ip) ((ip)->vmt->getsize(ip))

Returns the current file size.

Parameters

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

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.

7.8.2.13 #define fileStreamGetPosition(ip) ((ip)->vmt->getposition(ip))

Returns the current file pointer position.

Parameters

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

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.

7.8.2.14 #define fileStreamSeek(*ip*, *offset*) ((*ip*)->vmt->lseek(*ip*, *offset*))

Moves the file current pointer to an absolute position.

Parameters

in	<i>ip</i>	pointer to a FileStream or derived class
in	<i>offset</i>	new absolute position

Returns

The operation status.

Return values

<i>FILE_OK</i>	no error.
<i>FILE_ERROR</i>	operation failed.

Function Class:

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

7.8.3 Typedef Documentation

7.8.3.1 `typedef uint32_t fileoffset_t`

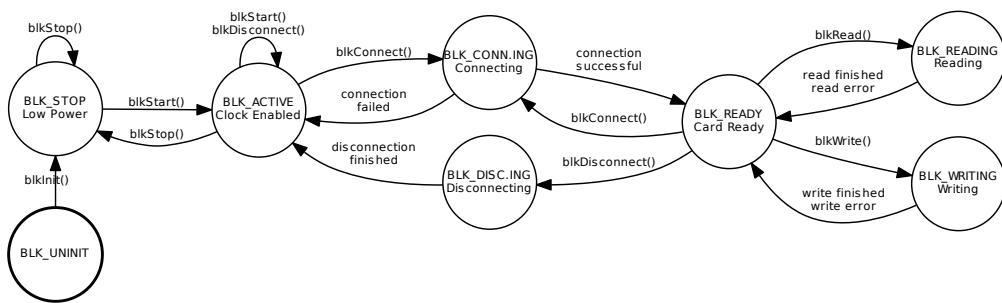
File offset type.

7.9 Abstract I/O Block Device

7.9.1 Detailed Description

7.9.2 Driver State Machine

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



This module defines an abstract interface for accessing generic block devices.

Note that no code is present, just abstract interfaces-like structures, you should look at the system as to a set of abstract C++ classes (even if written in C). This system has then advantage to make the access to block devices independent from the implementation logic.

Macros

- `#define _base_block_device_methods`
BaseBlockDevice specific methods.
- `#define _base_block_device_data`
BaseBlockDevice specific data.

Macro Functions (BaseBlockDevice)

- `#define blkGetDriverState(ip) ((ip)->state)`
Returns the driver state.
- `#define blkIsTransferring(ip)`
Determines if the device is transferring data.
- `#define blkIsInserted(ip) ((ip)->vmt->is_inserted(ip))`
Returns the media insertion status.
- `#define blkIsWriteProtected(ip) ((ip)->vmt->is_protected(ip))`
Returns the media write protection status.
- `#define blkConnect(ip) ((ip)->vmt->connect(ip))`
Performs the initialization procedure on the block device.
- `#define blkDisconnect(ip) ((ip)->vmt->disconnect(ip))`

- Terminates operations on the block device.*
- #define `blkRead(ip, startblk, buf, n)` ((ip)->vmt->read(ip, startblk, buf, n))
Reads one or more blocks.
 - #define `blkWrite(ip, startblk, buf, n)` ((ip)->vmt->write(ip, startblk, buf, n))
Writes one or more blocks.
 - #define `blkSync(ip)` ((ip)->vmt->`sync(ip)`)
Ensures write synchronization.
 - #define `blkGetInfo(ip, bdip)` ((ip)->vmt->get_info(ip, bdip))
Returns a media information structure.

Data Structures

- struct `BlockDeviceInfo`
Block device info.
- struct `BaseBlockDeviceVMT`
BaseBlockDevice virtual methods table.
- struct `BaseBlockDevice`
Base block device class.

Enumerations

- enum `blkstate_t` {
 `BLK_UNINIT` = 0, `BLK_STOP` = 1, `BLK_ACTIVE` = 2, `BLK_CONNECTING` = 3,
`BLK_DISCONNECTING` = 4, `BLK_READY` = 5, `BLK_READING` = 6, `BLK_WRITING` = 7,
`BLK_SYNCING` = 8 }

Driver state machine possible states.

7.9.3 Macro Definition Documentation

7.9.3.1 #define _base_block_device_methods

Value:

```
/* 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.

7.9.3.2 #define _base_block_device_data

Value:

```
/* Driver state */
blkstate_t state;
```

BaseBlockDevice specific data.

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

Returns the driver state.

Note

Can be called in ISR context.

Parameters

in	<i>ip</i>	pointer to a BaseBlockDevice or derived class
----	-----------	---

Returns

The driver state.

Function Class:

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

7.9.3.4 #define blkIsTransferring(ip)

Value:

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

Determines if the device is transferring data.

Note

Can be called in ISR context.

Parameters

in	<i>ip</i>	pointer to a BaseBlockDevice or derived class
----	-----------	---

Returns

The driver state.

Return values

<i>FALSE</i>	the device is not transferring data.
<i>TRUE</i>	the device not transferring data.

Function Class:

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

7.9.3.5 #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	<i>ip</i>	pointer to a BaseBlockDevice or derived class
----	-----------	---

Returns

The media state.

Return values

<i>FALSE</i>	media not inserted.
<i>TRUE</i>	media inserted.

Function Class:

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

7.9.3.6 #define blkIsWriteProtected(*ip*) ((ip)->vmt->is_protected(ip))

Returns the media write protection status.

Parameters

in	<i>ip</i>	pointer to a BaseBlockDevice or derived class
----	-----------	---

Returns

The media state.

Return values

<i>FALSE</i>	writable media.
<i>TRUE</i>	non writable media.

Function Class:

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

7.9.3.7 #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	<i>ip</i>	pointer to a BaseBlockDevice or derived class
----	-----------	---

Returns

The operation status.

Return values

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

Function Class:

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

7.9.3.8 #define blkDisconnect(*ip*) ((ip)->vmt->disconnect(ip))

Terminates operations on the block device.

This operation safely terminates operations on the block device.

Parameters

in	<i>ip</i>	pointer to a BaseBlockDevice or derived class
----	-----------	---

Returns

The operation status.

Return values

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

Function Class:

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

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

Reads one or more blocks.

Parameters

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

Returns

The operation status.

Return values

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

Function Class:

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

7.9.3.10 #define blkWrite(*ip*, *startblk*, *buf*, *n*) ((*ip*)->vmt->write(*ip*, *startblk*, *buf*, *n*))

Writes one or more blocks.

Parameters

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

Returns

The operation status.

Return values

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

Function Class:

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

7.9.3.11 #define blkSync(*ip*) ((*ip*)->vmt->sync(*ip*))

Ensures write synchronization.

Parameters

in	<i>ip</i>	pointer to a BaseBlockDevice or derived class
----	-----------	---

Returns

The operation status.

Return values

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

Function Class:

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

7.9.3.12 #define blkGetInfo(*ip*, *bdip*) ((*ip*)->vmt->get_info(*ip*, *bdip*))

Returns a media information structure.

Parameters

in	<i>ip</i>	pointer to a BaseBlockDevice or derived class
out	<i>bdipl</i>	pointer to a BlockDeviceInfo structure

Returns

The operation status.

Return values

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

Function Class:

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

7.9.4 Enumeration Type Documentation**7.9.4.1 enum blkstate_t**

Driver state machine possible states.

Enumerator

BLK_UNINIT Not initialized.

BLK_STOP Stopped.

BLK_ACTIVE Interface active.

BLK_CONNECTING Connection in progress.

BLK_DISCONNECTING Disconnection in progress.

BLK_READY Device ready.

BLK_READING Read operation in progress.

BLK_WRITING Write operation in progress.

BLK_SYNCING Sync. operation in progress.

7.10 I/O Queues

7.10.1 Detailed Description

Queues are mostly used in serial-like device drivers. Serial device drivers are usually designed to have a lower side (lower driver, it is usually an interrupt service routine) and an upper side (upper driver, accessed by the application threads).

There are several kind of queues:

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

Macros

- `#define _INPUTQUEUE_DATA(name, buffer, size, inotify, link)`
Data part of a static input queue initializer.
- `#define INPUTQUEUE_DECL(name, buffer, size, inotify, link) input_queue_t name = _INPUTQUEUE_DA←
TA(name, buffer, size, inotify, link)`
Static input queue initializer.
- `#define _OUTPUTQUEUE_DATA(name, buffer, size, onotify, link)`
Data part of a static output queue initializer.
- `#define OUTPUTQUEUE_DECL(name, buffer, size, onotify, link) output_queue_t name = _OUTPUTQUE←
UE_DATA(name, buffer, size, onotify, link)`
Static output queue initializer.

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_t)-3`
Queue empty.
- `#define Q_FULL (msg_t)-4`
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 iqGetFullI(iqp) qSpaceI(iqp)`

- `#define iqGetEmptyI(iqp) (qSizeX(iqp) - qSpaceI(iqp))`

Returns the filled space into an input queue.
- `#define iqIsEmptyI(iqp) ((bool)(qSpaceI(iqp) == 0U))`

Returns the empty space into an input queue.
- `#define iqIsFullI(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 oqIsEmptyI(oqp)`

Evaluates to true if the specified output queue is empty.
- `#define oqIsFullI(oqp) ((bool)(qSpaceI(oqp) == 0U))`

Evaluates to true if the specified output queue is full.
- `#define oqPut(oqp, b) oqPutTimeout(oqp, b, TIME_INFINITE)`

Output queue write.

Typedefs

- `typedef struct io_queue io_queue_t`

Type of a generic I/O queue structure.
- `typedef void(* qnotify_t) (io_queue_t *qp)`

Queue notification callback type.
- `typedef io_queue_t input_queue_t`

Type of an input queue structure.
- `typedef io_queue_t output_queue_t`

Type of an output queue structure.

Data Structures

- `struct io_queue`

Generic I/O queue structure.

Functions

- `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 iqGetTimeout (input_queue_t *iqp, systime_t timeout)`

Input queue read with timeout.
- `size_t iqReadTimeout (input_queue_t *iqp, uint8_t *bp, size_t n, systime_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 oqPutTimeout (output_queue_t *oqp, uint8_t b, systime_t timeout)`
Output queue write with timeout.
- `msg_t oqGetI (output_queue_t *oqp)`
Output queue read.
- `size_t oqWriteTimeout (output_queue_t *oqp, const uint8_t *bp, size_t n, systime_t timeout)`
Output queue write with timeout.

7.10.2 Macro Definition Documentation

7.10.2.1 #define Q_OK MSG_OK

Operation successful.

7.10.2.2 #define Q_TIMEOUT MSG_TIMEOUT

Timeout condition.

7.10.2.3 #define Q_RESET MSG_RESET

Queue has been reset.

7.10.2.4 #define Q_EMPTY (msg_t)-3

Queue empty.

7.10.2.5 #define Q_FULL (msg_t)-4

Queue full.,

7.10.2.6 #define qSizeX(qp)

Value:

```
/*lint -save -e9033 [10.8] The cast is safe.*/
((size_t)((qp)->q_top - (qp)->q_buffer)) \
/*lint -restore*/
```

Returns the queue's buffer size.

Parameters

in	qp	pointer to a <code>io_queue_t</code> structure
----	----	--

Returns

The buffer size.

Function Class:

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

7.10.2.7 #define qSpaceI(*qp*) ((*qp*)->q_counter)

Queue space.

Returns the used space if used on an input queue or the empty space if used on an output queue.

Parameters

in	<i>qp</i>	pointer to a <code>io_queue_t</code> structure
----	-----------	--

Returns

The buffer space.

Function Class:

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

7.10.2.8 #define qGetLink(*qp*) ((*qp*)->q_link)

Returns the queue application-defined link.

Note

This function can be called in any context.

Parameters

in	<i>qp</i>	pointer to a <code>io_queue_t</code> structure
----	-----------	--

Returns

The application-defined link.

Function Class:

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

7.10.2.9 #define iqGetFull(*iqp*) qSpaceI(*iqp*)

Returns the filled space into an input queue.

Parameters

in	<i>iqp</i>	pointer to an <code>input_queue_t</code> 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.

```
7.10.2.10 #define iqGetEmpty( iqp ) (qSizeX(iqp) - qSpace(iqp))
```

Returns the empty space into an input queue.

Parameters

in	<i>iqp</i>	pointer to an <code>input_queue_t</code> 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.

7.10.2.11 #define iqIsEmpty(*iqp*) ((bool)(qSpace(iqp) == 0U))

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

Parameters

in	<i>iqp</i>	pointer to an <code>input_queue_t</code> structure
----	------------	--

Returns

The queue status.

Return values

<i>false</i>	if the queue is not empty.
<i>true</i>	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.

7.10.2.12 #define iqIsFull(*iqp*)**Value:**

```
((bool)((((iwp)->q_wptr == (iwp)->q_rptr) && \
          ((iwp)->q_counter != 0U)))
```

Evaluates to `true` if the specified input queue is full.

Parameters

in	<i>iqp</i>	pointer to an <code>input_queue_t</code> structure
----	------------	--

Returns

The queue status.

Return values

<i>false</i>	if the queue is not full.
<i>true</i>	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.

7.10.2.13 #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

<i>in</i>	<i>iqp</i>	pointer to an <code>input_queue_t</code> structure
-----------	------------	--

Returns

A byte value from the queue.

Return values

<i>Q_RESET</i>	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.

7.10.2.14 #define _INPUTQUEUE_DATA(*name*, *buffer*, *size*, *inotify*, *link*)**Value:**

```
{
    \\\
    NULL,
    0U,
    (uint8_t *) (buffer),
    (uint8_t *) (buffer) + (size),
    (uint8_t *) (buffer),
    (uint8_t *) (buffer),
    (inotify),
    (link)
}
```

Data part of a static input queue initializer.

This macro should be used when statically initializing an input queue that is part of a bigger structure.

Parameters

<i>in</i>	<i>name</i>	the name of the input queue variable
<i>in</i>	<i>buffer</i>	pointer to the queue buffer area
<i>in</i>	<i>size</i>	size of the queue buffer area
<i>in</i>	<i>inotify</i>	input notification callback pointer
<i>in</i>	<i>link</i>	application defined pointer

```
7.10.2.15 #define INPUTQUEUE_DECL( name, buffer, size, inotify, link ) input_queue_t name =
    _INPUTQUEUE_DATA(name, buffer, size, inotify, link)
```

Static input queue initializer.

Statically initialized input queues require no explicit initialization using `iqInit()`.

Parameters

in	<i>name</i>	the name of the input queue variable
in	<i>buffer</i>	pointer to the queue buffer area
in	<i>size</i>	size of the queue buffer area
in	<i>inotify</i>	input notification callback pointer
in	<i>link</i>	application defined pointer

```
7.10.2.16 #define oqGetFull( oqp ) (qSizeX(oqp) - qSpaceI(oqp))
```

Returns the filled space into an output queue.

Parameters

in	<i>oqp</i>	pointer to an <code>output_queue_t</code> 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.

```
7.10.2.17 #define oqGetEmpty( oqp ) qSpaceI(oqp)
```

Returns the empty space into an output queue.

Parameters

in	<i>oqp</i>	pointer to an <code>output_queue_t</code> 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.

7.10.2.18 #define oqIsEmpty(*oqp*)**Value:**

```
((bool) (((oqp)->q_wptr == (oqp)->q_rptr) && \
           ((oqp)->q_counter != 0U)))
```

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

Parameters

in	<i>oqp</i>	pointer to an <code>output_queue_t</code> structure
----	------------	---

Returns

The queue status.

Return values

<i>false</i>	if the queue is not empty.
<i>true</i>	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.

7.10.2.19 #define oqIsFull(*oqp*) ((bool)(qSpace(oqp) == 0U))

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

Parameters

in	<i>oqp</i>	pointer to an <code>output_queue_t</code> structure
----	------------	---

Returns

The queue status.

Return values

<i>false</i>	if the queue is not full.
<i>true</i>	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.

7.10.2.20 #define oqPut(*oqp*, *b*) oqPutTimeout(*oqp*, *b*, TIME_INFINITE)

Output queue write.

This function writes a byte value to an output queue. If the queue is full then the calling thread is suspended until there is space in the queue.

Parameters

in	<i>oqp</i>	pointer to an <code>output_queue_t</code> structure
in	<i>b</i>	the byte value to be written in the queue

Returns

The operation status.

Return values

<i>Q_OK</i>	if the operation succeeded.
<i>Q_RESET</i>	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.

7.10.2.21 #define _OUTPUTQUEUE_DATA(*name*, *buffer*, *size*, *onotify*, *link*)**Value:**

```
{
    NULL, \
    (size),
    (uint8_t *) (buffer),
    (uint8_t *) (buffer) + (size),
    (uint8_t *) (buffer),
    (uint8_t *) (buffer),
    (onotify),
    (link)
}
```

Data part of a static output queue initializer.

This macro should be used when statically initializing an output queue that is part of a bigger structure.

Parameters

in	<i>name</i>	the name of the output queue variable
in	<i>buffer</i>	pointer to the queue buffer area
in	<i>size</i>	size of the queue buffer area
in	<i>onotify</i>	output notification callback pointer
in	<i>link</i>	application defined pointer

**7.10.2.22 #define OUTPUTQUEUE_DECL(*name*, *buffer*, *size*, *onotify*, *link*) output_queue_t *name* =
_OUTPUTQUEUE_DATA(*name*, *buffer*, *size*, *onotify*, *link*)**

Static output queue initializer.

Statically initialized output queues require no explicit initialization using `oqInit()`.

Parameters

in	<i>name</i>	the name of the output queue variable
in	<i>buffer</i>	pointer to the queue buffer area
in	<i>size</i>	size of the queue buffer area
in	<i>onotify</i>	output notification callback pointer
in	<i>link</i>	application defined pointer

7.10.3 Typedef Documentation

7.10.3.1 `typedef struct io_queue io_queue_t`

Type of a generic I/O queue structure.

7.10.3.2 `typedef void(* qnotify_t)(io_queue_t *qp)`

Queue notification callback type.

Parameters

in	<i>qp</i>	the queue pointer.
----	-----------	--------------------

7.10.3.3 `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.

7.10.3.4 `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.

7.10.4 Function Documentation

7.10.4.1 `void iqObjectInit(input_queue_t * iqp, uint8_t * bp, size_t size, qnotify_t infy, void * link)`

Initializes an input queue.

A Semaphore is internally initialized and works as a counter of the bytes contained in the queue.

Note

The callback is invoked from within the S-Locked system state.

Parameters

out	<i>iqp</i>	pointer to an <code>input_queue_t</code> structure
in	<i>bp</i>	pointer to a memory area allocated as queue buffer
in	<i>size</i>	size of the queue buffer
in	<i>infy</i>	pointer to a callback function that is invoked when data is read from the queue. The value can be <code>NULL</code> .
in	<i>link</i>	application defined pointer

Function Class:

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

Here is the call graph for this function:



7.10.4.2 void iqResetl (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 Q_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	<i>iqp</i>	pointer to an <code>input_queue_t</code> 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:



7.10.4.3 msg_t iqPutl (input_queue_t * iqp, uint8_t b)

Input queue write.

A byte value is written into the low end of an input queue.

Parameters

in	<i>iqp</i>	pointer to an <code>input_queue_t</code> structure
in	<i>b</i>	the byte value to be written in the queue

Returns

The operation status.

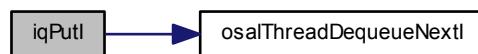
Return values

<i>Q_OK</i>	if the operation has been completed with success.
<i>Q_FULL</i>	if the queue is full and the operation cannot be completed.

Function 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:



7.10.4.4 `msg_t iqGetTimeout(input_queue_t * iqp, systime_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 before reading the character from the buffer or before entering the state THD_STA←TE_WTQUEUE.

Parameters

in	<i>iqp</i>	pointer to an <code>input_queue_t</code> structure
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

A byte value from the queue.

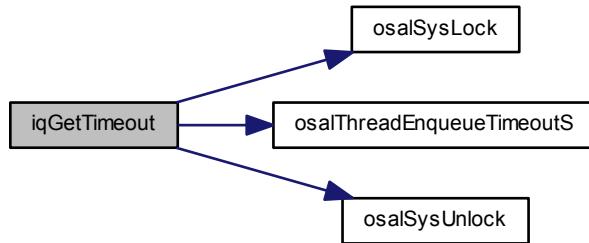
Return values

<i>Q_TIMEOUT</i>	if the specified time expired.
<i>Q_RESET</i>	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:



7.10.4.5 `size_t iqReadTimeout (input_queue_t * iqp, uint8_t * bp, size_t n, systime_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 before reading each character from the buffer or before entering the state THD_ST←ATE_WTQUEUE.

Parameters

in	<i>iqp</i>	pointer to an <code>input_queue_t</code> structure
out	<i>bp</i>	pointer to the data buffer
in	<i>n</i>	the maximum amount of data to be transferred, the value 0 is reserved
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

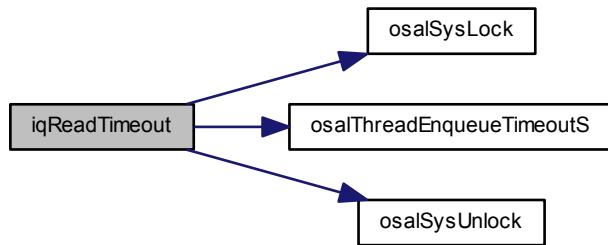
Returns

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:

**7.10.4.6 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

out	<i>oqp</i>	pointer to an <code>output_queue_t</code> structure
in	<i>bp</i>	pointer to a memory area allocated as queue buffer
in	<i>size</i>	size of the queue buffer
in	<i>onfy</i>	pointer to a callback function that is invoked when data is written to the queue. The value can be <code>NULL</code> .
in	<i>link</i>	application defined pointer

Function Class:

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

Here is the call graph for this function:



7.10.4.7 void oqResetl (`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 Q_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	<code>oqp</code>	pointer to an <code>output_queue_t</code> 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:



7.10.4.8 msg_t oqPutTimeout (`output_queue_t * oqp, uint8_t b, systime_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 writing the character into the buffer.

Parameters

in	<code>oqp</code>	pointer to an <code>output_queue_t</code> structure
in	<code>b</code>	the byte value to be written in the queue
in	<code>timeout</code>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

The operation status.

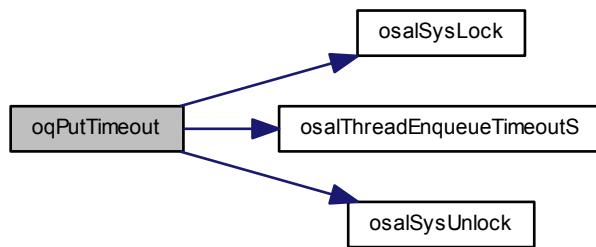
Return values

<i>Q_OK</i>	if the operation succeeded.
<i>Q_TIMEOUT</i>	if the specified time expired.
<i>Q_RESET</i>	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:

7.10.4.9 `msg_t oqGetl(output_queue_t * oqp)`

Output queue read.

A byte value is read from the low end of an output queue.

Parameters

in	<i>oqp</i>	pointer to an <code>output_queue_t</code> structure
----	------------	---

Returns

The byte value from the queue.

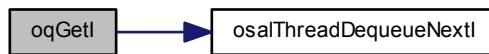
Return values

<i>Q_EMPTY</i>	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:



7.10.4.10 size_t oqWriteTimeout(output_queue_t * *oqp*, const uint8_t * *bp*, size_t *n*, systime_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 writing each character into the buffer.

Parameters

in	<i>oqp</i>	pointer to an <code>output_queue_t</code> structure
in	<i>bp</i>	pointer to the data buffer
in	<i>n</i>	the maximum amount of data to be transferred, the value 0 is reserved
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <i>TIME_IMMEDIATE</i> immediate timeout. • <i>TIME_INFINITE</i> no timeout.

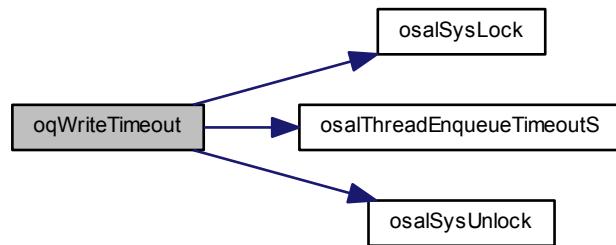
Returns

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:



7.11 Abstract Streams

7.11.1 Detailed Description

This module define an abstract interface for generic data streams. Note that no code is present, just abstract interfaces-like structures, you should look at the system as to a set of abstract C++ classes (even if written in C). This system has then advantage to make the access to data streams independent from the implementation logic. The stream interface can be used as base class for high level object types such as files, sockets, serial ports, pipes etc.

Macros

- `#define _base_sequential_stream_methods`
BaseSequentialStream specific methods.
- `#define _base_sequential_stream_data`
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.

7.11.2 Macro Definition Documentation

7.11.2.1 `#define _base_sequential_stream_methods`

Value:

```
/* 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.

7.11.2.2 #define _base_sequential_stream_data

[BaseSequentialStream](#) specific data.

Note

It is empty because [BaseSequentialStream](#) is only an interface without implementation.

7.11.2.3 #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

in	ip	pointer to a BaseSequentialStream or derived class
in	bp	pointer to the data buffer
in	n	the maximum amount of data to be transferred

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.

7.11.2.4 #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

in	ip	pointer to a BaseSequentialStream or derived class
out	bp	pointer to the data buffer
in	n	the maximum amount of data to be transferred

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.

7.11.2.5 #define streamPut(ip, b) ((ip)->vmt->put(ip, b))

Sequential Stream blocking byte write.

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

Parameters

in	<i>ip</i>	pointer to a BaseChannel or derived class
in	<i>b</i>	the byte value to be written to the channel

Returns

The operation status.

Return values

<i>STM_OK</i>	if the operation succeeded.
<i>STM_RESET</i>	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.

7.11.2.6 #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	<i>ip</i>	pointer to a BaseChannel or derived class
----	-----------	---

Returns

A byte value from the queue.

Return values

<i>STM_RESET</i>	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.

7.12 I2C Driver

Generic I2C Driver.

7.12.1 Detailed Description

Generic I2C Driver.

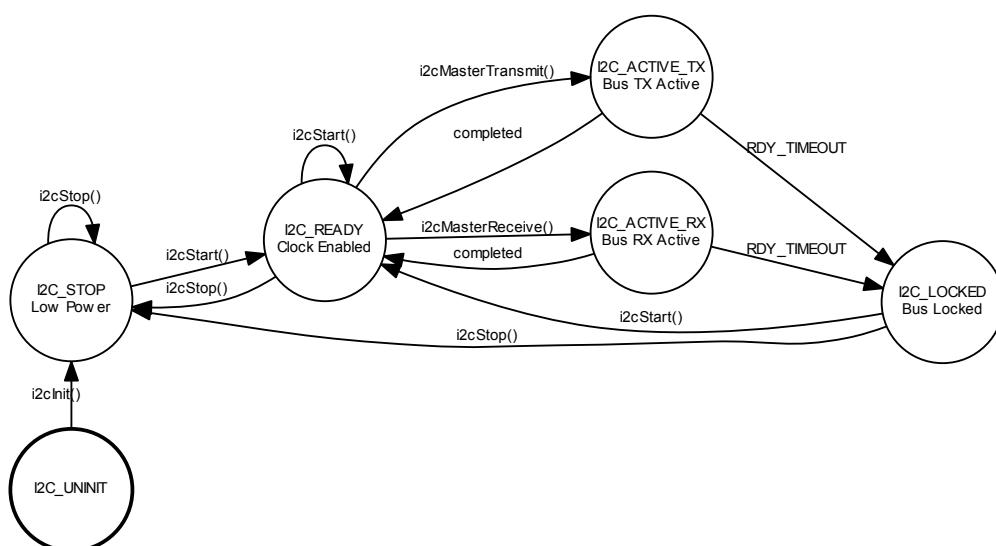
This module implements a generic I2C (Inter-Integrated Circuit) driver.

Precondition

In order to use the I2C driver the `HAL_USE_I2C` option must be enabled in `halconf.h`.

7.12.2 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).



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)`

- `#define i2cMasterTransmit(i2cp, addr, txbuf, txbytes, rxbuf, rxbytes)`
Wrap i2cMasterTransmit function with TIME_INFINITE timeout.
- `#define i2cMasterReceive(i2cp, addr, rxbuf, rxbytes) (i2cMasterReceiveTimeout(i2cp, addr, rxbuf, rxbytes, TIME_INFINITE))`
Wrap i2cMasterReceiveTimeout function with TIME_INFINITE timeout.
- `#define i2c_lld_get_errors(i2cp) ((i2cp)->errors)`
Get errors from I2C driver.

I2C bus error conditions

- `#define I2C_NO_ERROR 0x00`
No error.
- `#define I2C_BUS_ERROR 0x01`
Bus Error.
- `#define I2C_ARBITRATION_LOST 0x02`
Arbitration Lost.
- `#define I2C_ACK_FAILURE 0x04`
Acknowledge Failure.
- `#define I2C_OVERRUN 0x08`
Overrun/Underrun.
- `#define I2C_PEC_ERROR 0x10`
PEC Error in reception.
- `#define I2C_TIMEOUT 0x20`
Hardware timeout.
- `#define I2C_SMB_ALERT 0x40`
SMBus Alert.

PLATFORM configuration options

- `#define PLATFORM_I2C_USE_I2C1 FALSE`
I2C1 driver enable switch.

Typedefs

- `typedef uint16_t i2caddr_t`
Type representing an I2C address.
- `typedef uint32_t i2cflags_t`
Type of I2C Driver condition flags.
- `typedef struct 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 i2cMasterTimeout (I2CDriver *i2cp, i2caddr_t addr, const uint8_t *txbuf, size_t txbytes, uint8_t *rdbuf, size_t rxbytes, systime_t timeout)`
Sends data via the I2C bus.
- `msg_t i2cMasterReceiveTimeout (I2CDriver *i2cp, i2caddr_t addr, uint8_t *rdbuf, size_t rxbytes, systime_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 *rdbuf, size_t rxbytes, systime_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 *rdbuf, size_t rxbytes, systime_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`
}

Driver state machine possible states.

Variables

- `I2CDriver I2CD1`
I2C1 driver identifier.

7.12.3 Macro Definition Documentation

7.12.3.1 #define I2C_NO_ERROR 0x00

No error.

7.12.3.2 #define I2C_BUS_ERROR 0x01

Bus Error.

7.12.3.3 #define I2C_ARBITRATION_LOST 0x02

Arbitration Lost.

7.12.3.4 #define I2C_ACK_FAILURE 0x04

Acknowledge Failure.

7.12.3.5 #define I2C_OVERRUN 0x08

Overrun/Underrun.

7.12.3.6 #define I2C_PEC_ERROR 0x10

PEC Error in reception.

7.12.3.7 #define I2C_TIMEOUT 0x20

Hardware timeout.

7.12.3.8 #define I2C_SMB_ALERT 0x40

SMBus Alert.

7.12.3.9 #define I2C_USE_MUTUAL_EXCLUSION TRUE

Enables the mutual exclusion APIs on the I2C bus.

7.12.3.10 #define _i2c_wakeup_isr(i2cp)

Value:

```
do {
    osalSysLockFromISR();
    \
    osalThreadResumeI(&(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.

7.12.3.11 #define _i2c_wakeup_error_isr(*i2cp*)

Value:

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

Wakes up the waiting thread notifying errors.

Parameters

in	<i>i2cp</i>	pointer to the I2CDriver object
----	-------------	---

Function Class:

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

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

Value:

```
(i2cMasterTransmitTimeout(i2cp, addr, txbuf, txbytes, rxbuf, rxbytes,
TIME_INFINITE)) \
```

Wrap i2cMasterTransmitTimeout function with TIME_INFINITE timeout.

Function Class:

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

7.12.3.13 #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.

7.12.3.14 #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.

7.12.3.15 #define i2c_lld_get_errors(*i2cp*) ((i2cp)->errors)

Get errors from I2C driver.

Parameters

in	<i>i2cp</i>	pointer to the I2CDriver object
----	-------------	---

Function Class:

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

7.12.4 TYPEDOC Documentation**7.12.4.1 `typedef uint16_t i2caddr_t`**

Type representing an I2C address.

7.12.4.2 `typedef uint32_t i2cflags_t`

Type of I2C Driver condition flags.

7.12.4.3 `typedef struct I2CDriver I2CDriver`

Type of a structure representing an I2C driver.

7.12.5 ENUMERATION Documentation**7.12.5.1 `enum i2cstate_t`**

Driver state machine possible states.

Enumerator

I2C_UNINIT Not initialized.

I2C_STOP Stopped.

I2C_READY Ready.

I2C_ACTIVE_TX Transmitting.

I2C_ACTIVE_RX Receiving.

7.12.6 FUNCTION Documentation**7.12.6.1 `void i2clinit(void)`**

I2C Driver initialization.

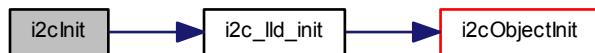
Note

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:



7.12.6.2 void i2cObjectInit (I2CDriver * *i2cp*)

Initializes the standard part of a [I2CDriver](#) structure.

Parameters

out	<i>i2cp</i>	pointer to the I2CDriver object
-----	-------------	---

Function Class:

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

Here is the call graph for this function:



7.12.6.3 void i2cStart (I2CDriver * *i2cp*, const I2CConfig * *config*)

Configures and activates the I2C peripheral.

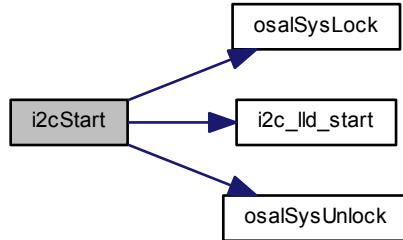
Parameters

in	<i>i2cp</i>	pointer to the I2CDriver object
in	<i>config</i>	pointer to the I2CConfig 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:



7.12.6.4 void i2cStop (I2CDriver * i2cp)

Deactivates the I2C peripheral.

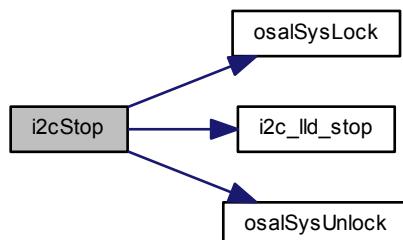
Parameters

in	<i>i2cp</i>	pointer to the I2CDriver object
----	-------------	---

Function Class:

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

Here is the call graph for this function:



7.12.6.5 i2cflags_t i2cGetErrors (I2CDriver * i2cp)

Returns the errors mask associated to the previous operation.

Parameters

in	<i>i2cp</i>	pointer to the I2CDriver 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.

7.12.6.6 *msg_t i2cMasterTransmitTimeout(I2CDriver *i2cp, i2caddr_t addr, const uint8_t *txbuf, size_t txbytes, uint8_t *rxbuf, size_t rxbytes, systime_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

in	<i>i2cp</i>	pointer to the I2CDriver object
in	<i>addr</i>	slave device address (7 bits) without R/W bit
in	<i>txbuf</i>	pointer to transmit buffer
in	<i>txbytes</i>	number of bytes to be transmitted
out	<i>rxbuf</i>	pointer to receive buffer
in	<i>rxbytes</i>	number of bytes to be received, set it to 0 if you want transmit only
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <i>TIME_INFINITE</i> no timeout.

Returns

The operation status.

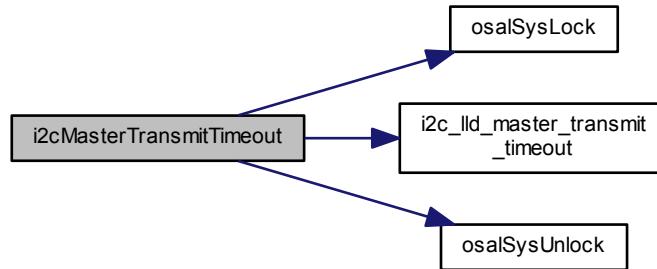
Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more I2C errors occurred, the errors can be retrieved using i2cGetErrors() .
<i>MSG_TIMEOUT</i>	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.

Here is the call graph for this function:



7.12.6.7 `msg_t i2cMasterReceiveTimeout(I2CDriver * i2cp, i2caddr_t addr, uint8_t * rdbuf, size_t rxbytes, systime_t timeout)`

Receives data from the I2C bus.

Parameters

in	<code>i2cp</code>	pointer to the I2CDriver object
in	<code>addr</code>	slave device address (7 bits) without R/W bit
out	<code>rdbuf</code>	pointer to receive buffer
in	<code>rxbytes</code>	number of bytes to be received
in	<code>timeout</code>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_INFINITE</code> no timeout.

Returns

The operation status.

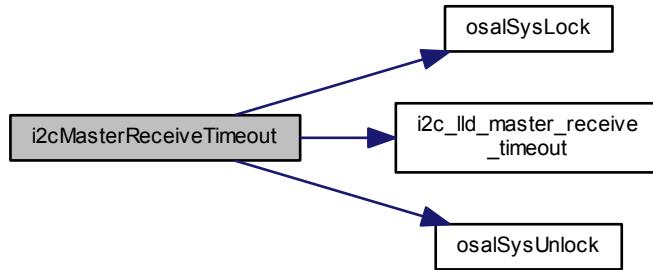
Return values

<code>MSG_OK</code>	if the function succeeded.
<code>MSG_RESET</code>	if one or more I2C errors occurred, the errors can be retrieved using i2cGetErrors() .
<code>MSG_TIMEOUT</code>	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.

Here is the call graph for this function:



7.12.6.8 void i2cAcquireBus (I2CDriver * i2cp)

Gains exclusive access to the I2C bus.

This function tries to gain ownership to the I2C bus, if the bus is already being used then the invoking thread is queued.

Precondition

In order to use this function the option `I2C_USE_MUTUAL_EXCLUSION` must be enabled.

Parameters

in	<code>i2cp</code>	pointer to the <code>I2CDriver</code> 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:



7.12.6.9 void i2cReleaseBus (I2CDriver * i2cp)

Releases exclusive access to the I2C bus.

Precondition

In order to use this function the option `I2C_USE_MUTUAL_EXCLUSION` must be enabled.

Parameters

in	<i>i2cp</i>	pointer to the I2CDriver object
----	-------------	---

Function Class:

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

Here is the call graph for this function:

**7.12.6.10 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:

**7.12.6.11 void i2c_lld_start(I2CDriver * i2cp)**

Configures and activates the I2C peripheral.

Parameters

in	<i>i2cp</i>	pointer to the I2CDriver object
----	-------------	---

Function Class:

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

7.12.6.12 void i2c_lld_stop(I2CDriver * i2cp)

Deactivates the I2C peripheral.

Parameters

in	<i>i2cp</i>	pointer to the I2CDriver object
----	-------------	---

Function Class:

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

7.12.6.13 msg_t i2c_lld_master_receive_timeout (I2CDriver * *i2cp*, i2caddr_t *addr*, uint8_t * *rxbuf*, size_t *rxbytes*, systime_t *timeout*)

Receives data via the I2C bus as master.

Parameters

in	<i>i2cp</i>	pointer to the I2CDriver object
in	<i>addr</i>	slave device address
out	<i>rxbuf</i>	pointer to the receive buffer
in	<i>rxbytes</i>	number of bytes to be received
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <i>TIME_INFINITE</i> no timeout.

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more I2C errors occurred, the errors can be retrieved using i2cGetErrors() .
<i>MSG_TIMEOUT</i>	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.

7.12.6.14 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*, systime_t *timeout*)

Transmits data via the I2C bus as master.

Parameters

in	<i>i2cp</i>	pointer to the I2CDriver object
in	<i>addr</i>	slave device address
in	<i>txbuf</i>	pointer to the transmit buffer
in	<i>txbytes</i>	number of bytes to be transmitted
out	<i>rxbuf</i>	pointer to the receive buffer

in	<i>rxbytes</i>	number of bytes to be received
in	<i>timeout</i>	<p>the number of ticks before the operation timeouts, the following special values are allowed:</p> <ul style="list-style-type: none"> • <i>TIME_INFINITE</i> no timeout.

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the function succeeded.
<i>MSG_RESET</i>	if one or more I2C errors occurred, the errors can be retrieved using <i>i2cGetErrors()</i> .
<i>MSG_TIMEOUT</i>	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.

7.12.7 Variable Documentation

7.12.7.1 I2CDriver I2CD1

I2C1 driver identifier.

7.13 I2S Driver

Generic I2S Driver.

7.13.1 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`.

7.13.2 Driver State Machine

I2S modes

- `#define I2S_MODE_SLAVE 0`
- `#define I2S_MODE_MASTER 1`

Macro Functions

- `#define i2sStartExchange(i2sp)`
Starts a I2S data exchange.
- `#define i2sStopExchange(i2sp)`
Stops the ongoing data exchange.
- `#define _i2s_isr_half_code(i2sp)`
Common ISR code, half buffer event.
- `#define _i2s_isr_full_code(i2sp)`
Common ISR code.

PLATFORM configuration options

- `#define PLATFORM_I2S_USE_I2S1 FALSE`
I2SD1 driver enable switch.

Typedefs

- `typedef struct I2SDriver I2SDriver`
Type of a structure representing an I2S driver.
- `typedef void(* i2scallback_t) (I2SDriver *i2sp, size_t offset, size_t n)`
I2S notification callback type.

Data Structures

- `struct I2SConfig`
Driver configuration structure.
- `struct I2SDriver`
Structure representing an I2S driver.

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.

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.

7.13.3 Macro Definition Documentation

7.13.3.1 #define `i2sStartExchange`(`i2sp`)

Value:

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

Starts a I2S data exchange.

Parameters

in	<code>i2sp</code>	pointer to the <code>I2SDriver</code> object
----	-------------------	--

Function Class:

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

7.13.3.2 #define i2sStopExchange(*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	<i>i2sp</i>	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.

7.13.3.3 #define _i2s_isr_half_code(*i2sp*)**Value:**

```
{
    if ((i2sp)>config->end_cb != NULL) {
        (i2sp)>config->end_cb(i2sp, 0, (i2sp)>config->size / 2);
    }
}
```

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	<i>i2sp</i>	pointer to the I2CDriver object
----	-------------	---

Function Class:

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

7.13.3.4 #define _i2s_isr_full_code(*i2sp*)**Value:**

```
{
    if ((i2sp)>config->end_cb) {
        (i2sp)>state = I2S_COMPLETE;
        (i2sp)>config->end_cb(i2sp,
            (i2sp)>config->size / 2,
            (i2sp)>config->size / 2);
        if ((i2sp)>state == I2S_COMPLETE)
            (i2sp)>state = I2S_READY;
    }
    else
        (i2sp)>state = I2S_READY;
}
```

Common ISR code.

This code handles the portable part of the ISR code:

- Callback invocation.
- Driver state transitions.

Note

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

Parameters

in	<i>i2sp</i>	pointer to the I2CDriver object
----	-------------	---

Function Class:

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

7.13.3.5 #define PLATFORM_I2S_USE_I2S1 FALSE

I2SD1 driver enable switch.

If set to TRUE the support for I2S1 is included.

Note

The default is FALSE.

7.13.4 Typedef Documentation

7.13.4.1 typedef struct I2SDriver I2SDriver

Type of a structure representing an I2S driver.

7.13.4.2 typedef void(* i2scallback_t) (I2SDriver *i2sp, size_t offset, size_t n)

I2S notification callback type.

Parameters

in	<i>i2sp</i>	pointer to the I2SDriver object
in	<i>offset</i>	offset in buffers of the data to read/write
in	<i>n</i>	number of samples to read/write

7.13.5 Enumeration Type Documentation

7.13.5.1 enum i2sstate_t

Driver state machine possible states.

Enumerator

I2S_UNINIT Not initialized.

I2S_STOP Stopped.

I2S_READY Ready.

I2S_ACTIVE Active.

I2S_COMPLETE Transmission complete.

7.13.6 Function Documentation

7.13.6.1 void i2sInit(void)

I2S Driver initialization.

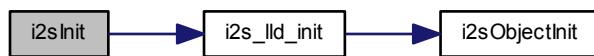
Note

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:



7.13.6.2 void i2sObjectInit(I2SDriver * i2sp)

Initializes the standard part of a `I2SDriver` structure.

Parameters

out	<i>i2sp</i>	pointer to the <code>I2SDriver</code> object
-----	-------------	--

Function Class:

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

7.13.6.3 void i2sStart(I2SDriver * i2sp, const I2SConfig * config)

Configures and activates the I2S peripheral.

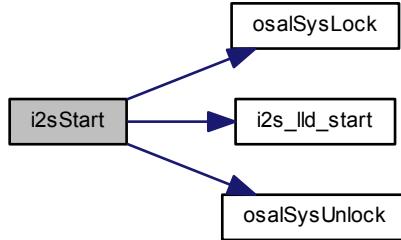
Parameters

in	<i>i2sp</i>	pointer to the <code>I2SDriver</code> object
in	<i>config</i>	pointer to the <code>I2SConfig</code> 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:



7.13.6.4 void i2sStop (I2SDriver * i2sp)

Deactivates the I2S peripheral.

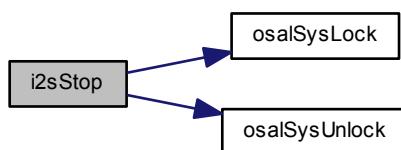
Parameters

in	<i>i2sp</i>	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:



7.13.6.5 void i2sStartExchange (I2SDriver * i2sp)

Starts a I2S data exchange.

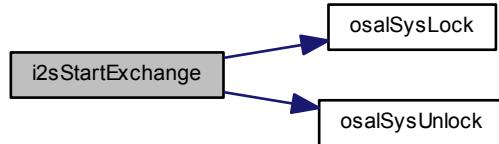
Parameters

in	<i>i2sp</i>	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:



7.13.6.6 void i2sStopExchange (I2SDriver * i2sp)

Stops the ongoing data exchange.

The ongoing data exchange, if any, is stopped, if the driver was not active the function does nothing.

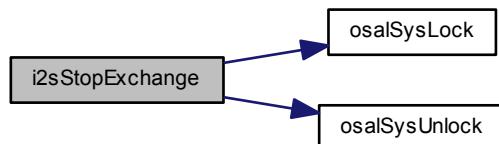
Parameters

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

Function Class:

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

Here is the call graph for this function:



7.13.6.7 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:



7.13.6.8 void i2s_lld_start (I2SDriver * i2sp)

Configures and activates the I2S peripheral.

Parameters

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

Function Class:

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

Deactivates the I2S peripheral.

Parameters

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

Function Class:

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

Starts a I2S data exchange.

Parameters

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

Function Class:

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

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.

7.13.7 Variable Documentation

7.13.7.1 I2SDriver I2SD1

I2S2 driver identifier.

7.14 ICU Driver

Generic ICU Driver.

7.14.1 Detailed Description

Generic ICU Driver.

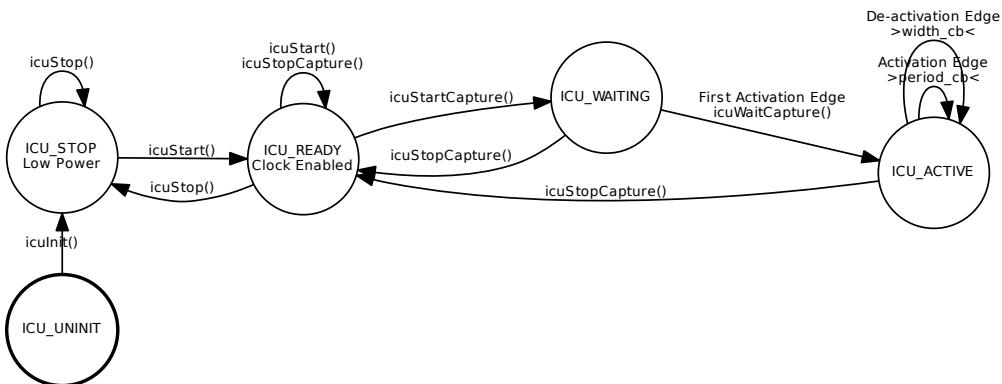
This module implements a generic ICU (Input Capture Unit) driver. The purpose of the driver is to measure period and duty cycle of an input digital signal (PWM input).

Precondition

In order to use the ICU driver the `HAL_USE_ICU` option must be enabled in `halconf.h`.

7.14.2 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).



7.14.3 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 icuStartCapture1(icup)`
Starts the input capture.
- `#define icuStopCapture1(icup)`
Stops the input capture.
- `#define icuEnableNotifications1(icup) icu_lld_enable_notifications(icup)`
Enables notifications.
- `#define icuDisableNotifications1(icup) icu_lld_disable_notifications(icup)`
Disables notifications.
- `#define icuAreNotificationsEnabledX(icup) icu_lld_are_notifications_enabled(icup)`
Check on notifications status.
- `#define icuGetWidthX(icup) icu_lld_get_width(icup)`
Returns the width of the latest pulse.
- `#define icuGetPeriodX(icup) icu_lld_get_period(icup)`
Returns the width of the latest cycle.

Low level driver helper macros

- `#define _icu_isr_invoke_width_cb(icup)`
Common ISR code, ICU width event.
- `#define _icu_isr_invoke_period_cb(icup)`
Common ISR code, ICU period event.
- `#define _icu_isr_invoke_overflow_cb(icup)`
Common ISR code, ICU timer overflow event.

PLATFORM configuration options

- `#define PLATFORM_ICU_USE_ICU1 FALSE`
ICUD1 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.
- **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 `icemode_t` { `ICU_INPUT_ACTIVE_HIGH` = 0, `ICU_INPUT_ACTIVE_LOW` = 1 }

ICU driver mode.

Variables

- `ICUDriver` `ICUD1`
ICUD1 driver identifier.

7.14.4 Macro Definition Documentation

7.14.4.1 #define `icuStartCapture`(`icup`)

Value:

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

Starts the input capture.

Parameters

in	<code>icup</code>	pointer to the <code>ICUDriver</code> 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.

7.14.4.2 #define `icuStopCapture`(`icup`)

Value:

```
do {
    icu_lld_stop_capture(icup);
    (icup)>state = ICU_READY;
} while (false)
```

Stops the input capture.

Parameters

in	<i>icup</i>	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.

7.14.4.3 #define icuEnableNotifications(*icup*) icu_lld_enable_notifications(*icup*)

Enables notifications.

Precondition

The ICU unit must have been activated using [icuStart\(\)](#).

Note

If the notification is already enabled then the call has no effect.

Parameters

in	<i>icup</i>	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.

7.14.4.4 #define icuDisableNotifications(*icup*) icu_lld_disable_notifications(*icup*)

Disables notifications.

Precondition

The ICU unit must have been activated using [icuStart\(\)](#).

Note

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

Parameters

in	<i>icup</i>	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.

7.14.4.5 #define icuAreNotificationsEnabledX(*icup*) icu_lld_are_notifications_enabled(*icup*)

Check on notifications status.

Parameters

in	<i>icup</i>	pointer to the ICUDriver object
----	-------------	---

Returns

The notifications status.

Return values

<i>false</i>	if notifications are not enabled.
<i>true</i>	if notifications are enabled.

Function Class:

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

7.14.4.6 #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	<i>icup</i>	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.

7.14.4.7 #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	<i>icup</i>	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.

7.14.4.8 #define _icu_isr_invoke_width_cb(*icup*)**Value:**

```
do {
    if (((icup)->state == ICU_ACTIVE) &&
        ((icup)->config->width_cb != NULL))
        (icup)->config->width_cb(icup);
} while (0)
```

Common ISR code, ICU width event.

Parameters

in	<i>icup</i>	pointer to the ICUDriver object
----	-------------	---

Function Class:

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

7.14.4.9 #define _icu_isr_invoke_period_cb(*icup*)**Value:**

```
do {
    if (((icup)->state == ICU_ACTIVE) &&
        ((icup)->config->period_cb != NULL))
        (icup)->config->period_cb(icup);
    (icup)->state = ICU_ACTIVE;
} while (0)
```

Common ISR code, ICU period event.

Note

A period event brings the driver into the `ICU_ACTIVE` state.

Parameters

in	<i>icup</i>	pointer to the ICUDriver object
----	-------------	---

Function Class:

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

7.14.4.10 #define _icu_isr_invoke_overflow_cb(*icup*)**Value:**

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

Common ISR code, ICU timer overflow event.

Note

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

Parameters

in	<i>icup</i>	pointer to the ICUDriver object
----	-------------	---

Function Class:

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

7.14.4.11 #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.

7.14.4.12 #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	<i>icup</i>	pointer to the ICUDriver object
----	-------------	---

Returns

The number of ticks.

Function Class:

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

7.14.4.13 #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	<i>icup</i>	pointer to the ICUDriver object
----	-------------	---

Returns

The number of ticks.

Function Class:

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

7.14.4.14 #define icu_lld_are_notifications_enabled(*icup*) false

Check on notifications status.

Parameters

in	<i>icup</i>	pointer to the ICUDriver object
----	-------------	---

Returns

The notifications status.

Return values

<i>false</i>	if notifications are not enabled.
<i>true</i>	if notifications are enabled.

Function Class:

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

7.14.5 Typedef Documentation**7.14.5.1 `typedef struct ICUDriver ICUDriver`**

Type of a structure representing an ICU driver.

7.14.5.2 `typedef void(* icucallback_t)(ICUDriver *icup)`

ICU notification callback type.

Parameters

in	<i>icup</i>	pointer to a ICUDriver object
----	-------------	---

7.14.5.3 `typedef uint32_t icufreq_t`

ICU frequency type.

7.14.5.4 `typedef uint32_t icucnt_t`

ICU counter type.

7.14.6 Enumeration Type Documentation**7.14.6.1 `enum icustate_t`**

Driver state machine possible states.

Enumerator

ICU_UNINIT Not initialized.

ICU_STOP Stopped.

ICU_READY Ready.

ICU_WAITING Waiting for first front.

ICU_ACTIVE First front detected.

7.14.6.2 enum icumode_t

ICU driver mode.

Enumerator

ICU_INPUT_ACTIVE_HIGH Trigger on rising edge.

ICU_INPUT_ACTIVE_LOW Trigger on falling edge.

7.14.7 Function Documentation

7.14.7.1 void iculinit (void)

ICU Driver initialization.

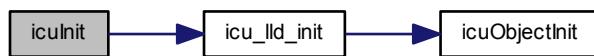
Note

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:



7.14.7.2 void icuObjectInit (ICUDriver * *icup*)

Initializes the standard part of a [ICUDriver](#) structure.

Parameters

out	<i>icup</i>	pointer to the ICUDriver object
-----	-------------	---

Function Class:

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

7.14.7.3 void icuStart (ICUDriver * *icup*, const ICUConfig * *config*)

Configures and activates the ICU peripheral.

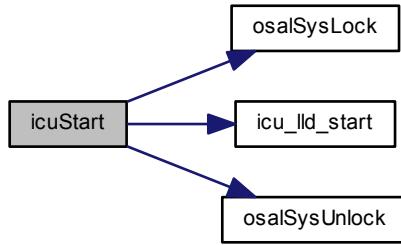
Parameters

in	<i>icup</i>	pointer to the ICUDriver object
in	<i>config</i>	pointer to the ICUConfig 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:

**7.14.7.4 void icuStop (ICUDriver * *icup*)**

Deactivates the ICU peripheral.

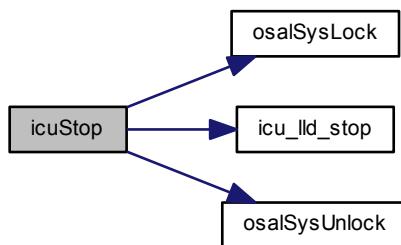
Parameters

in	<i>icup</i>	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:



7.14.7.5 void icuStartCapture (**ICUDriver** * *icup*)

Starts the input capture.

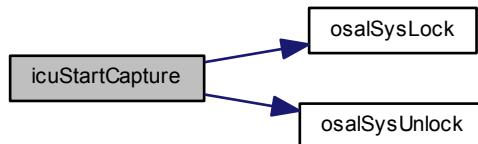
Parameters

in	<i>icup</i>	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:

**7.14.7.6 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	<i>icup</i>	pointer to the ICUDriver object
----	-------------	---

Returns

The capture status.

Return values

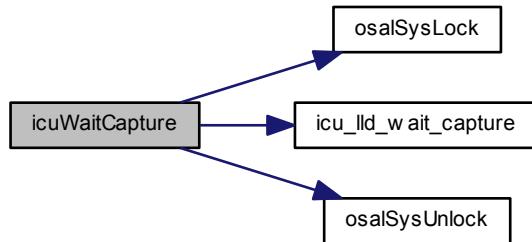
<i>false</i>	if the capture is successful.
--------------	-------------------------------

<code>true</code>	if a timer overflow occurred.
-------------------	-------------------------------

Function Class:

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

Here is the call graph for this function:

**7.14.7.7 void icuStopCapture (ICUDriver * *icup*)**

Stops the input capture.

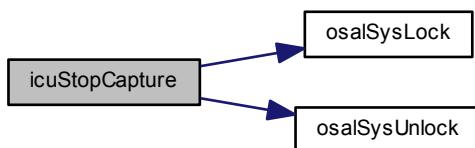
Parameters

in	<i>icup</i>	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:

**7.14.7.8 void icuEnableNotifications (ICUDriver * *icup*)**

Enables notifications.

Precondition

The ICU unit must have been activated using `icuStart()`.

Note

If the notification is already enabled then the call has no effect.

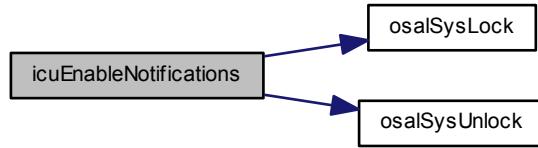
Parameters

in	<i>icup</i>	pointer to the <code>ICUDriver</code> 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:

**7.14.7.9 void icuDisableNotifications (ICUDriver * *icup*)**

Disables notifications.

Precondition

The ICU unit must have been activated using `icuStart()`.

Note

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

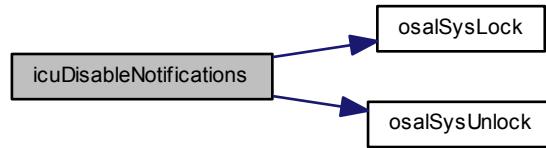
Parameters

in	<i>icup</i>	pointer to the <code>ICUDriver</code> 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:



7.14.7.10 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:



7.14.7.11 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.

7.14.7.12 void icu_lld_stop(ICUDriver * icup)

Deactivates the ICU peripheral.

Parameters

in	<i>icup</i>	pointer to the ICUDriver object
----	-------------	---

Function Class:

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

7.14.7.13 void icu_lld_start_capture (ICUDriver * *icup*)

Starts the input capture.

Parameters

in	<i>icup</i>	pointer to the ICUDriver object
----	-------------	---

Function Class:

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

7.14.7.14 bool icu_lld_wait_capture (ICUDriver * *icup*)

Waits for a completed capture.

Note

The operation is performed in polled mode.

In order to use this function notifications must be disabled.

Parameters

in	<i>icup</i>	pointer to the ICUDriver object
----	-------------	---

Returns

The capture status.

Return values

<i>false</i>	if the capture is successful.
<i>true</i>	if a timer overflow occurred.

Function Class:

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

7.14.7.15 void icu_lld_stop_capture (ICUDriver * *icup*)

Stops the input capture.

Parameters

in	<i>icup</i>	pointer to the ICUDriver object
----	-------------	---

Function Class:

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

7.14.7.16 void icu_lld_enable_notifications (ICUDriver * *icup*)

Enables notifications.

Precondition

The ICU unit must have been activated using [icuStart \(\)](#).

Note

If the notification is already enabled then the call has no effect.

Parameters

in	<i>icup</i>	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.

7.14.7.17 void icu_lld_disable_notifications (ICUDriver * *icup*)

Disables notifications.

Precondition

The ICU unit must have been activated using [icuStart \(\)](#).

Note

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

Parameters

in	<i>icup</i>	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.

7.14.8 Variable Documentation**7.14.8.1 ICUDriver ICUD1**

ICUD1 driver identifier.

Note

The driver ICUD1 allocates the complex timer TIM1 when enabled.

7.15 MAC Driver

Generic MAC driver.

7.15.1 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)`
Returns the received frames event source.
- `#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, **systime_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, **systime_t** timeout)
Waits for a received frame.
- void **macReleaseReceiveDescriptor** (**MACReceiveDescriptor** *rdp)
Releases a receive descriptor.
- bool **macPollLinkStatus** (**MACDriver** *macp)
Updates and returns the link status.
- void **mac_lld_init** (void)
Low level MAC initialization.
- void **mac_lld_start** (**MACDriver** *macp)
Configures and activates the MAC peripheral.
- void **mac_lld_stop** (**MACDriver** *macp)
Deactivates the MAC peripheral.
- **msg_t** **mac_lld_get_transmit_descriptor** (**MACDriver** *macp, **MACTransmitDescriptor** *tdp)
Returns a transmission descriptor.
- void **mac_lld_release_transmit_descriptor** (**MACTransmitDescriptor** *tdp)
Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.
- **msg_t** **mac_lld_get_receive_descriptor** (**MACDriver** *macp, **MACReceiveDescriptor** *rdp)
Returns a receive descriptor.
- void **mac_lld_release_receive_descriptor** (**MACReceiveDescriptor** *rdp)
Releases a receive descriptor.
- bool **mac_lld_poll_link_status** (**MACDriver** *macp)
Updates and returns the link status.
- **size_t** **mac_lld_write_transmit_descriptor** (**MACTransmitDescriptor** *tdp, **uint8_t** *buf, **size_t** size)
Writes to a transmit descriptor's stream.
- **size_t** **mac_lld_read_receive_descriptor** (**MACReceiveDescriptor** *rdp, **uint8_t** *buf, **size_t** size)
Reads from a receive descriptor's stream.

- `uint8_t * mac_lld_get_next_transmit_buffer (MACTransmitDescriptor *tdp, size_t size, size_t *sizep)`
Returns a pointer to the next transmit buffer in the descriptor chain.
- `const uint8_t * mac_lld_get_next_receive_buffer (MACReceiveDescriptor *rdp, size_t *sizep)`
Returns a pointer to the next receive buffer in the descriptor chain.

Enumerations

- enum `macstate_t { MAC_UNINIT = 0, MAC_STOP = 1, MAC_ACTIVE = 2 }`
Driver state machine possible states.

Variables

- `MACDriver ETHD1`
MAC1 driver identifier.

7.15.2 Macro Definition Documentation

7.15.2.1 `#define MAC_USE_ZERO_COPY FALSE`

Enables an event sources for incoming packets.

7.15.2.2 `#define MAC_USE_EVENTS TRUE`

Enables an event sources for incoming packets.

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

Returns the received frames event source.

Parameters

in	<code>macp</code>	pointer to the <code>MACDriver</code> 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.

7.15.2.4 `#define macWriteTransmitDescriptor(tdp, buf, size) mac_lld_write_transmit_descriptor(tdp, buf, size)`

Writes to a transmit descriptor's stream.

Parameters

in	<code>tdp</code>	pointer to a <code>MACTransmitDescriptor</code> structure
in	<code>buf</code>	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.

7.15.2.5 #define macReadReceiveDescriptor(*rdp*, *buf*, *size*) mac_lld_read_receive_descriptor(*rdp*, *buf*, *size*)

Reads from a receive descriptor's stream.

Parameters

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

Returns

The number of bytes read from the descriptor's stream, this value can be less than the amount specified in the parameter `size` if there are no more bytes to read.

Function Class:

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

7.15.2.6 #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

in	<i>tdp</i>	pointer to a MACTransmitDescriptor structure
in	<i>size</i>	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.
out	<i>sizep</i>	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.

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.

7.15.2.7 #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

<i>in</i>	<i>rdp</i>	pointer to a MACReceiveDescriptor structure
<i>out</i>	<i>sizep</i>	pointer to variable receiving the buffer size, it is zero when the last buffer has already been returned.

Returns

Pointer to the returned buffer.

Return values

<i>NULL</i>	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.

7.15.2.8 #define MAC_SUPPORTS_ZERO_COPY TRUE

This implementation supports the zero-copy mode API.

7.15.2.9 #define PLATFORM_MAC_USE_MAC1 FALSE

MAC driver enable switch.

If set to TRUE the support for MAC1 is included.

Note

The default is FALSE.

7.15.3 Typedef Documentation

7.15.3.1 `typedef struct MACDriver MACDriver`

Type of a structure representing a MAC driver.

7.15.4 Enumeration Type Documentation

7.15.4.1 `enum macstate_t`

Driver state machine possible states.

Enumerator

MAC_UNINIT Not initialized.

MAC_STOP Stopped.

MAC_ACTIVE Active.

7.15.5 Function Documentation

7.15.5.1 void macInit(void)

MAC Driver initialization.

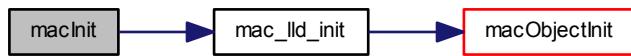
Note

This function is implicitly invoked by [halInit\(\)](#), there is no need to explicitly initialize the driver.

Function Class:

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

Here is the call graph for this function:



7.15.5.2 void macObjectInit(MACDriver * macp)

Initialize the standard part of a [MACDriver](#) structure.

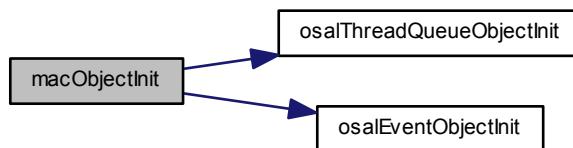
Parameters

out	<i>macp</i>	pointer to the MACDriver object
-----	-------------	---

Function Class:

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

Here is the call graph for this function:



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

Configures and activates the MAC peripheral.

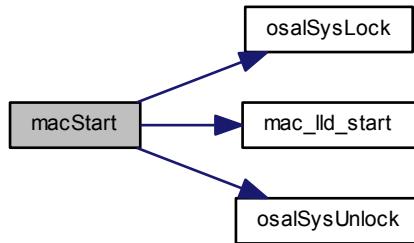
Parameters

in	<i>macp</i>	pointer to the MACDriver object
in	<i>config</i>	pointer to the MACConfig 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:

**7.15.5.4 void macStop (MACDriver * *macp*)**

Deactivates the MAC peripheral.

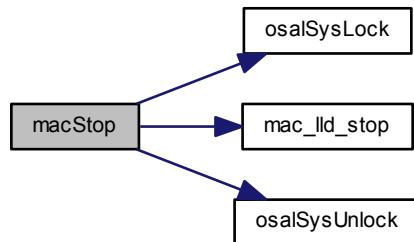
Parameters

in	<i>macp</i>	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.

Here is the call graph for this function:



7.15.5.5 `msg_t macWaitTransmitDescriptor(MACDriver * macp, MACTransmitDescriptor * tdp, systime_t timeout)`

Allocates a transmission descriptor.

One of the available transmission descriptors is locked and returned. If a descriptor is not currently available then the invoking thread is queued until one is freed.

Parameters

in	<i>macp</i>	pointer to the <code>MACDriver</code> object
out	<i>tdp</i>	pointer to a <code>MACTransmitDescriptor</code> structure
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

The operation status.

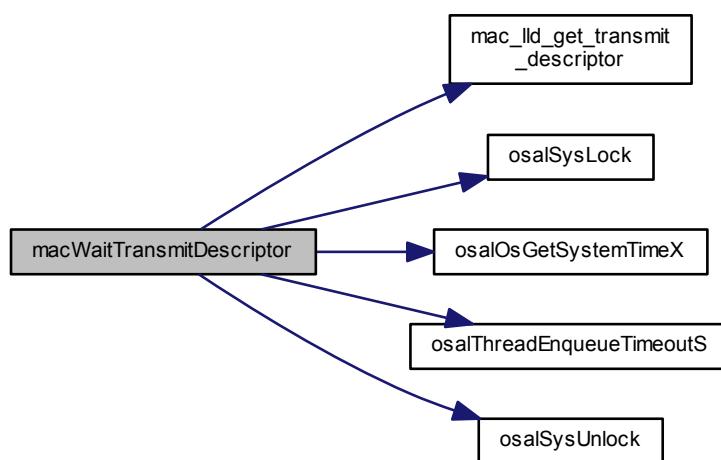
Return values

<code>MSG_OK</code>	the descriptor was obtained.
<code>MSG_TIMEOUT</code>	the operation timed out, descriptor not initialized.

Function Class:

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

Here is the call graph for this function:



7.15.5.6 void macReleaseTransmitDescriptor (MACTransmitDescriptor * *tdp*)

Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.

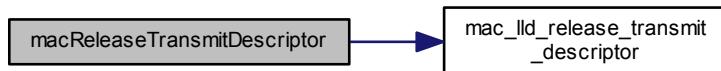
Parameters

in	<i>tdp</i>	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:



7.15.5.7 `msg_t macWaitReceiveDescriptor (MACDriver * macp, MACReceiveDescriptor * rdp, systime_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

in	<i>macp</i>	pointer to the MACDriver object
out	<i>rdp</i>	pointer to a MACReceiveDescriptor structure
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <i>TIME_IMMEDIATE</i> immediate timeout. • <i>TIME_INFINITE</i> no timeout.

Returns

The operation status.

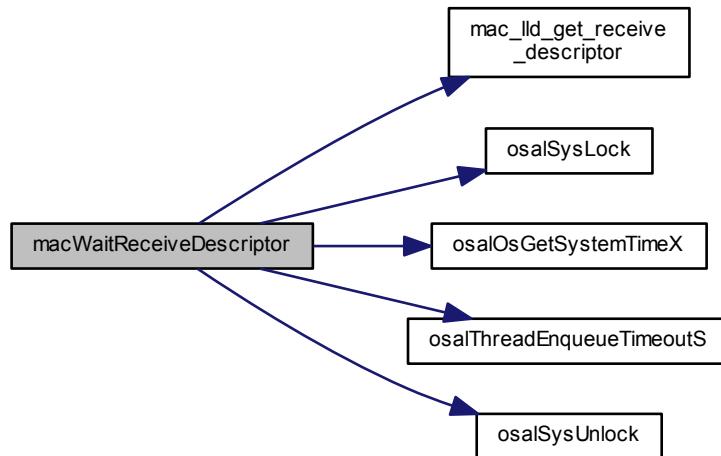
Return values

<i>MSG_OK</i>	the descriptor was obtained.
<i>MSG_TIMEOUT</i>	the operation timed out, descriptor not initialized.

Function Class:

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

Here is the call graph for this function:

**7.15.5.8 void macReleaseReceiveDescriptor (**MACReceiveDescriptor** * *rdp*)**

Releases a receive descriptor.

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

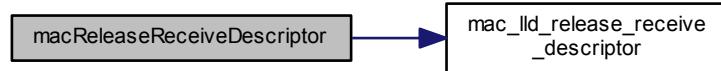
Parameters

in	<i>rdp</i>	the pointer to the <code>MACReceiveDescriptor</code> 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:



7.15.5.9 bool macPollLinkStatus (MACDriver * macp)

Updates and returns the link status.

Parameters

in	<i>macp</i>	pointer to the MACDriver object
----	-------------	---

Returns

The link status.

Return values

<i>true</i>	if the link is active.
<i>false</i>	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:



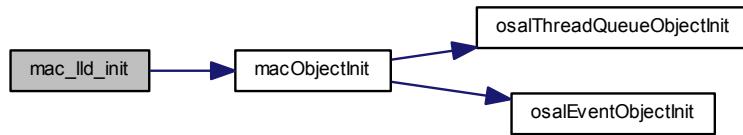
7.15.5.10 void mac_lld_init (void)

Low level MAC initialization.

Function Class:

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

Here is the call graph for this function:



7.15.5.11 void mac_lld_start (**MACDriver** * *macp*)

Configures and activates the MAC peripheral.

Parameters

in	<i>macp</i>	pointer to the MACDriver object
----	-------------	--

Function Class:

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

7.15.5.12 void mac_lld_stop (**MACDriver** * *macp*)

Deactivates the MAC peripheral.

Parameters

in	<i>macp</i>	pointer to the MACDriver object
----	-------------	--

Function Class:

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

7.15.5.13 **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	<i>macp</i>	pointer to the MACDriver object
out	<i>tdp</i>	pointer to a MACTransmitDescriptor structure

Returns

The operation status.

Return values

<i>MSG_OK</i>	the descriptor has been obtained.
<i>MSG_TIMEOUT</i>	descriptor not available.

Function Class:

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

7.15.5.14 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

<i>in</i>	<i>tdp</i>	the pointer to the MACTransmitDescriptor structure
-----------	------------	---

Function Class:

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

7.15.5.15 msg_t mac_lld_get_receive_descriptor (**MACDriver * *macp*, **MACReceiveDescriptor** * *rdp*)**

Returns a receive descriptor.

Parameters

<i>in</i>	<i>macp</i>	pointer to the MACDriver object
<i>out</i>	<i>rdp</i>	pointer to a MACReceiveDescriptor structure

Returns

The operation status.

Return values

<i>MSG_OK</i>	the descriptor has been obtained.
<i>MSG_TIMEOUT</i>	descriptor not available.

Function Class:

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

7.15.5.16 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

<i>in</i>	<i>rdp</i>	the pointer to the MACReceiveDescriptor structure
-----------	------------	--

Function Class:

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

7.15.5.17 bool mac_lld_poll_link_status (**MACDriver * *macp*)**

Updates and returns the link status.

Parameters

in	<i>macp</i>	pointer to the MACDriver object
----	-------------	---

Returns

The link status.

Return values

<i>true</i>	if the link is active.
<i>false</i>	if the link is down.

Function Class:

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

7.15.5.18 size_t mac_lld_write_transmit_descriptor ([MACTransmitDescriptor](#) * *tdp*, uint8_t * *buf*, size_t *size*)

Writes to a transmit descriptor's stream.

Parameters

in	<i>tdp</i>	pointer to a MACTransmitDescriptor structure
in	<i>buf</i>	pointer to the buffer containing the data to be written
in	<i>size</i>	number of bytes to be written

Returns

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

Function Class:

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

7.15.5.19 size_t mac_lld_read_receive_descriptor ([MACReceiveDescriptor](#) * *rdp*, uint8_t * *buf*, size_t *size*)

Reads from a receive descriptor's stream.

Parameters

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

Returns

The number of bytes read from the descriptor's stream, this value can be less than the amount specified in the parameter *size* if there are no more bytes to read.

Function Class:

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

7.15.5.20 `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

in	<i>tdp</i>	pointer to a <code>MACTransmitDescriptor</code> structure
in	<i>size</i>	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.
out	<i>sizep</i>	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.

Returns

Pointer to the returned buffer.

Return values

<code>NULL</code>	if the buffer chain has been entirely scanned.
-------------------	--

Function Class:

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

7.15.5.21 `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

in	<i>rdp</i>	pointer to a <code>MACReceiveDescriptor</code> structure
out	<i>sizep</i>	pointer to variable receiving the buffer size, it is zero when the last buffer has already been returned.

Returns

Pointer to the returned buffer.

Return values

<code>NULL</code>	if the buffer chain has been entirely scanned.
-------------------	--

Function Class:

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

7.15.6 Variable Documentation

7.15.6.1 MACDriver ETHD1

MAC1 driver identifier.

7.16 HAL

Hardware Abstraction Layer.

7.16.1 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.

7.16.2 HAL Device Drivers Architecture

The HAL contains several kind of modules:

- Normal Device Drivers
- Complex Device Drivers
- Interfaces
- Inner Code

7.16.3 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:

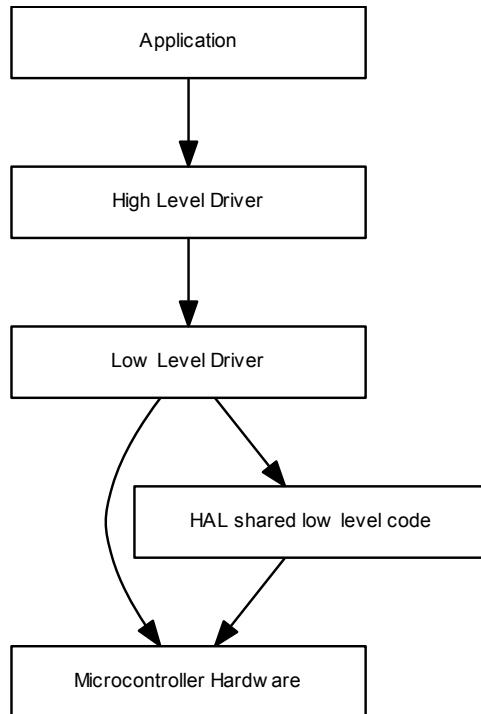
- <driver>.c, the HLD implementation file. This file must be included in the Makefile in order to use the driver.
- <driver>.h, the HLD header file. This file is implicitly included by the HAL header file `hal.h`.

- Low Level Device Driver (**LLD**). This layer contains the platform dependent part of the driver.

A LLD is composed by two files:

- <driver>_lld.c, the LLD implementation file. This file must be included in the Makefile in order to use the driver.
- <driver>_lld.h, the LLD header file. This file is implicitly included by the HLD header file.

7.16.3.1 Diagram



7.16.4 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.

7.16.5 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.

7.16.6 HAL Inner Code

Some modules are shared among multiple device drivers and are not necessarily meant to be used by the application layer.

Modules

- [Configuration](#)
HAL Configuration.
- [Normal Drivers](#)

HAL Normal Drivers.

- [Complex Drivers](#)

HAL Complex Drivers.

- [Interfaces](#)

HAL Interfaces.

- [Inner Code](#)

HAL Inner Code.

- [OSAL](#)

Operating System Abstraction Layer.

7.17 Configuration

HAL Configuration.

7.17.1 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.

Drivers enable switches

- `#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_DAC FALSE`
Enables the DAC subsystem.
- `#define HAL_USE_EXT TRUE`
Enables the EXT 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_SPI TRUE`
Enables the SPI subsystem.
- `#define HAL_USE_UART TRUE`
Enables the UART subsystem.
- `#define HAL_USE_USB TRUE`
Enables the USB subsystem.

ADC driver related setting

- #define ADC_USE_WAIT TRUE
Enables synchronous APIs.
- #define ADC_USE_MUTUAL_EXCLUSION TRUE
Enables the `adcAcquireBus()` and `adcReleaseBus()` APIs.

CAN driver related setting

- #define CAN_USE_SLEEP_MODE TRUE
Sleep mode related APIs inclusion switch.

I2C driver related setting

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

MAC driver related setting

- #define MAC_USE_ZERO_COPY TRUE
Enables an event sources for incoming packets.
- #define MAC_USE_EVENTS TRUE
Enables an event sources for incoming packets.

MMC_SPI driver related setting

- #define MMC_NICE_WAITING TRUE
Delays insertions.

SDC driver related setting

- #define SDC_INIT_RETRY 100
Number of initialization attempts before rejecting the card.
- #define SDC_MMIC_SUPPORT TRUE
Include support for MMC cards.
- #define SDC_NICE_WAITING TRUE
Delays insertions.

SERIAL driver related setting

- #define SERIAL_DEFAULT_BITRATE 38400
Default bit rate.
- #define SERIAL_BUFFERS_SIZE 16
Serial buffers size.

SERIAL_USB driver related setting

- #define SERIAL_USB_BUFFERS_SIZE 256
Serial over USB buffers size.

SPI driver related setting

- `#define SPI_USE_WAIT TRUE`
Enables synchronous APIs.
- `#define SPI_USE_MUTUAL_EXCLUSION TRUE`
Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.

7.17.2 Macro Definition Documentation

7.17.2.1 `#define HAL_USE_PAL TRUE`

Enables the PAL subsystem.

7.17.2.2 `#define HAL_USE_ADC TRUE`

Enables the ADC subsystem.

7.17.2.3 `#define HAL_USE_CAN TRUE`

Enables the CAN subsystem.

7.17.2.4 `#define HAL_USE_DAC FALSE`

Enables the DAC subsystem.

7.17.2.5 `#define HAL_USE_EXT TRUE`

Enables the EXT subsystem.

7.17.2.6 `#define HAL_USE_GPT TRUE`

Enables the GPT subsystem.

7.17.2.7 `#define HAL_USE_I2C TRUE`

Enables the I2C subsystem.

7.17.2.8 `#define HAL_USE_I2S TRUE`

Enables the I2S subsystem.

7.17.2.9 `#define HAL_USE_ICU TRUE`

Enables the ICU subsystem.

7.17.2.10 `#define HAL_USE_MAC TRUE`

Enables the MAC subsystem.

7.17.2.11 `#define HAL_USE_MMC_SPI TRUE`

Enables the MMC_SPI subsystem.

7.17.2.12 `#define HAL_USE_PWM TRUE`

Enables the PWM subsystem.

7.17.2.13 `#define HAL_USE_RTC TRUE`

Enables the RTC subsystem.

7.17.2.14 `#define HAL_USE_SDC TRUE`

Enables the SDC subsystem.

7.17.2.15 `#define HAL_USE_SERIAL TRUE`

Enables the SERIAL subsystem.

7.17.2.16 `#define HAL_USE_SERIAL_USB TRUE`

Enables the SERIAL over USB subsystem.

7.17.2.17 `#define HAL_USE_SPI TRUE`

Enables the SPI subsystem.

7.17.2.18 `#define HAL_USE_UART TRUE`

Enables the UART subsystem.

7.17.2.19 `#define HAL_USE_USB TRUE`

Enables the USB subsystem.

7.17.2.20 `#define ADC_USE_WAIT TRUE`

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.17.2.21 `#define ADC_USE_MUTUAL_EXCLUSION TRUE`

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

Note

Disabling this option saves both code and data space.

7.17.2.22 #define CAN_USE_SLEEP_MODE TRUE

Sleep mode related APIs inclusion switch.

7.17.2.23 #define I2C_USE_MUTUAL_EXCLUSION TRUE

Enables the mutual exclusion APIs on the I2C bus.

7.17.2.24 #define MAC_USE_ZERO_COPY TRUE

Enables an event sources for incoming packets.

7.17.2.25 #define MAC_USE_EVENTS TRUE

Enables an event sources for incoming packets.

7.17.2.26 #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.

7.17.2.27 #define SDC_INIT_RETRY 100

Number of initialization attempts before rejecting the card.

Note

Attempts are performed at 10mS intervals.

7.17.2.28 #define SDC_MMCC_SUPPORT TRUE

Include support for MMC cards.

Note

MMC support is not yet implemented so this option must be kept at FALSE.

7.17.2.29 #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.

7.17.2.30 #define SERIAL_DEFAULT_BITRATE 38400

Default bit rate.

Configuration parameter, this is the baud rate selected for the default configuration.

7.17.2.31 #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 64 bytes for both the transmission and receive buffers.

7.17.2.32 #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 64 bytes for both the transmission and receive buffers.

7.17.2.33 #define SPI_USE_WAIT TRUE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.17.2.34 #define SPI_USE_MUTUAL_EXCLUSION TRUE

Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.

Note

Disabling this option saves both code and data space.

7.18 Normal Drivers

HAL Normal Drivers.

7.18.1 Detailed Description

HAL Normal Drivers.

Modules

- [ADC Driver](#)
Generic ADC Driver.
- [CAN Driver](#)
Generic CAN Driver.
- [DAC Driver](#)
Generic DAC driver.
- [EXT Driver](#)
Generic EXT Driver.
- [GPT Driver](#)
Generic GPT Driver.
- [HAL Driver](#)
Hardware Abstraction Layer.
- [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](#)
Real Time Clock Abstraction Layer.
- [SDC Driver](#)
Generic SD Card Driver.
- [Serial Driver](#)
Generic Serial Driver.
- [SPI Driver](#)
Generic SPI Driver.
- [ST Driver](#)
Generic System Tick driver.
- [UART Driver](#)
Generic UART Driver.
- [USB Driver](#)
Generic USB Driver.

7.19 Complex Drivers

HAL Complex Drivers.

7.19.1 Detailed Description

HAL Complex Drivers.

Modules

- [MMC over SPI Driver](#)
Generic MMC driver.
- [Serial over USB Driver](#)
Serial over USB Driver.

7.20 Interfaces

HAL Interfaces.

7.20.1 Detailed Description

HAL Interfaces.

Modules

- [Abstract I/O Channel](#)
- [Abstract Files](#)
- [Abstract I/O Block Device](#)
- [Abstract Streams](#)

7.21 Inner Code

HAL Inner Code.

7.21.1 Detailed Description

HAL Inner Code.

Modules

- [I/O Queues](#)
- [MMC/SD Block Device](#)

7.22 OSAL

Operating System Abstraction Layer.

7.22.1 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_RESET (msg_t)-1`
- `#define MSG_TIMEOUT (msg_t)-2`

Special time constants

- `#define TIME_IMMEDIATE ((systime_t)0)`
- `#define TIME_INFINITE ((systime_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_S2ST(sec) ((systime_t)((uint32_t)(sec) * (uint32_t)OSAL_ST_FREQUENCY))`
Seconds to system ticks.
- `#define OSAL_MS2ST(msec)`
Milliseconds to system ticks.
- `#define OSAL_US2ST(usec)`
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(sec) osalThreadSleep(OSAL_S2ST(sec))`
Delays the invoking thread for the specified number of seconds.
- `#define osalThreadSleepMilliseconds(msec) osalThreadSleep(OSAL_MS2ST(msec))`
Delays the invoking thread for the specified number of milliseconds.
- `#define osalThreadSleepMicroseconds(usec) osalThreadSleep(OSAL_US2ST(usec))`
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 rtcnt_t`
Type of realtime counter.
- `typedef void *thread_reference_t`
Type of a thread reference.
- `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 eventflags_t`
Type of an event flags mask.
- `typedef uint32_t mutex_t`
Type of a mutex.

Data Structures

- `struct event_source`
Events source object.
- `struct threads_queue_t`
Type of a thread queue.

Functions

- `void osallinit (void)`
OSAL module initialization.
- `void 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 (systime_t time)**
Suspends the invoking thread for the specified time.
- void **osalThreadSleep (systime_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, systime_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, systime_t timeout)**
Enqueues the caller thread.
- void **osalThreadDequeueNextI (threads_queue_t *tqp, msg_t msg)**
Dequeues and wakes up one thread from the queue, if any.
- void **osalThreadDequeueAllI (threads_queue_t *tqp, msg_t msg)**
Dequeues and wakes up all threads from the queue.
- void **osalEventBroadcastFlagsI (event_source_t *esp, eventflags_t flags)**
Add flags to an event source object.
- void **osalEventBroadcastFlags (event_source_t *esp, eventflags_t flags)**
Add flags to an event source object.
- void **osalEventSetCallback (event_source_t *esp, eventcallback_t cb, void *param)**
Event callback setup.
- void **osalMutexLock (mutex_t *mp)**
Locks the specified mutex.
- void **osalMutexUnlock (mutex_t *mp)**
Unlocks the specified mutex.
- static void **osalSysDisable (void)**
Disables interrupts globally.
- static void **osalSysEnable (void)**
Enables interrupts globally.
- static void **osalSysLock (void)**
Enters a critical zone from thread context.
- static void **osalSysUnlock (void)**
Leaves a critical zone from thread context.
- static void **osalSysLockFromISR (void)**
Enters a critical zone from ISR context.
- static void **osalSysUnlockFromISR (void)**
Leaves a critical zone from ISR context.
- static syssts_t **osalSysGetStatusAndLockX (void)**
Returns the execution status and enters a critical zone.
- static void **osalSysRestoreStatusX (syssts_t sts)**
Restores the specified execution status and leaves a critical zone.
- static bool **osalOslsTimeWithinX (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 flags object.
- static void **osalMutexObjectInit (mutex_t *mp)**
Initializes a mutex_t object.

Variables

- const char * **osal_halt_msg**
Pointer to a halt error message.
- const char * **osal_halt_msg**
Pointer to a halt error message.

7.22.2 Macro Definition Documentation

7.22.2.1 #define OSAL_ST_RESOLUTION 32

Size in bits of the `systick_t` type.

7.22.2.2 #define OSAL_ST_FREQUENCY 1000

Required systick frequency or resolution.

7.22.2.3 #define OSAL_ST_MODE OSAL_ST_MODE_PERIODIC

Systick mode required by the underlying OS.

7.22.2.4 #define OSAL_IRQ_PRIORITY_LEVELS 16U

Total priority levels.

Implementation not mandatory.

7.22.2.5 #define OSAL_IRQ_MAXIMUM_PRIORITY 0U

Highest IRQ priority for HAL drivers.

Implementation not mandatory.

7.22.2.6 #define OSAL_DBG_ENABLE_ASSERTS FALSE

Enables OSAL assertions.

7.22.2.7 #define OSAL_DBG_ENABLE_CHECKS FALSE

Enables OSAL functions parameters checks.

7.22.2.8 #define osalDbgAssert(c, remark)

Value:

```
do {
    /*lint -save -e506 -e774 [2.1, 14.3] Can be a constant by design.*/
    if (OSAL_DBG_ENABLE_ASSERTS != FALSE) {
        if (!(c)) {
            /*lint -restore*/
            osalSysHalt(__func__);
        }
    }
} while (false)
```

Condition assertion.

If the condition check fails then the OSAL panics with a message and halts.

Note

The condition is tested only if the `OSAL_ENABLE_ASSERTIONS` switch is enabled.

The remark string is not currently used except for putting a comment in the code about the assertion.

Parameters

in	<i>c</i>	the condition to be verified to be true
in	<i>remark</i>	a remark string

Function Class:

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

7.22.2.9 #define osalDbgCheck(*c*)

Value:

```
do {
    /*lint -save -e506 -e774 [2.1, 14.3] Can be a constant by design.*/
    if (OSAL_DBG_ENABLE_CHECKS != FALSE) {
        if (!(c)) {
            /*lint -restore*/
            osalSysHalt (__func__);
        }
    }
} while (false)
```

Function parameters check.

If the condition check fails then the OSAL panics and halts.

Note

The condition is tested only if the `OSAL_ENABLE_CHECKS` switch is enabled.

Parameters

in	<i>c</i>	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.

7.22.2.10 #define osalDbgCheckClassI()

I-Class state check.

Note

Implementation is optional.

7.22.2.11 #define osalDbgCheckClassS()

S-Class state check.

Note

Implementation is optional.

7.22.2.12 #define OSAL_IRQ_IS_VALID_PRIORITY(*n*) (((*n*) >= OSAL_IRQ_MAXIMUM_PRIORITY) && ((*n*) < OSAL_IRQ_PRIORITY_LEVELS))

Priority level verification macro.

7.22.2.13 #define OSAL_IRQ_PROLOGUE()

IRQ prologue code.

This macro must be inserted at the start of all IRQ handlers.

7.22.2.14 #define OSAL_IRQ_EPILOGUE()

IRQ epilogue code.

This macro must be inserted at the end of all IRQ handlers.

7.22.2.15 #define OSAL_IRQ_HANDLER(*id*) void id(void)

IRQ handler function declaration.

This macro hides the details of an ISR function declaration.

Parameters

in	<i>id</i>	a vector name as defined in vectors.s
----	-----------	---------------------------------------

7.22.2.16 #define OSAL_S2ST(*sec*) ((*systime_t*)((*uint32_t*)(*sec*) * (*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	<i>sec</i>	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.

7.22.2.17 #define OSAL_MS2ST(*msec*)

Value:

```
((systime_t)((((uint32_t)(msec) * \
    (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	msec	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.

7.22.2.18 #define OSAL_US2ST(*usec*)**Value:**

```
((systime_t) (((((uint32_t)(usec)) *
    ((uint32_t)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	usec	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.

7.22.2.19 #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

in	<i>freq</i>	clock frequency, in Hz, of the realtime counter
in	<i>sec</i>	number of seconds

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.

7.22.2.20 #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

in	<i>freq</i>	clock frequency, in Hz, of the realtime counter
in	<i>msec</i>	number of milliseconds

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.

7.22.2.21 #define OSAL_US2RTC(freq, usec) (rtcnt_t)((((freq) + 999999UL) / 1000000UL) * (usec))

Microseconds to realtime counter.

Converts from microseconds to realtime counter cycles.

Note

The result is rounded upward to the next microsecond boundary.

The macro assumes that *freq* ≥ 1000000 .

Parameters

in	<i>freq</i>	clock frequency, in Hz, of the realtime counter
in	<i>usec</i>	number of microseconds

Returns

The number of cycles.

Function Class:

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

7.22.2.22 #define osalThreadSleepSeconds(sec) osalThreadSleep(OSAL_S2ST(sec))

Delays the invoking thread for the specified number of seconds.

Note

The specified time is rounded up to a value allowed by the real system tick clock.
The maximum specifiable value is implementation dependent.

Parameters

in	sec	time in seconds, must be different from zero
----	-----	--

Function Class:

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

7.22.2.23 #define osalThreadSleepMilliseconds(msec) osalThreadSleep(OSAL_MS2ST(msec))

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	msec	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.

7.22.2.24 #define osalThreadSleepMicroseconds(usec) osalThreadSleep(OSAL_US2ST(usec))

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	usec	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.

7.22.3 Typedef Documentation

7.22.3.1 typedef uint32_t syssts_t

Type of a system status word.

7.22.3.2 `typedef int32_t msg_t`

Type of a message.

7.22.3.3 `typedef uint32_t systime_t`

Type of system time counter.

7.22.3.4 `typedef uint32_t rtcnt_t`

Type of realtime counter.

7.22.3.5 `typedef void* thread_reference_t`

Type of a thread reference.

7.22.3.6 `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.

7.22.3.7 `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.

7.22.3.8 `typedef uint32_t eventflags_t`

Type of an event flags mask.

7.22.3.9 `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.

7.22.4 Function Documentation

7.22.4.1 `void osallinit(void)`

OSAL module initialization.

Function Class:

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

7.22.4.2 void osalSysHalt (const char * *reason*)

System halt with error message.

Parameters

in	<i>reason</i>	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:

**7.22.4.3 void osalSysPolledDelayX (rtcnt_t *cycles*)**

Polled delay.

Note

The real delay is always few cycles in excess of the specified value.

Parameters

in	<i>cycles</i>	number of cycles
----	---------------	------------------

Function Class:

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

7.22.4.4 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.

7.22.4.5 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.

7.22.4.6 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.

7.22.4.7 void osalThreadSleepS (systime_t time)

Suspends the invoking thread for the specified time.

Parameters

in	time	the delay in system ticks, the special values are handled as follow: <ul style="list-style-type: none">• <code>TIME_INFINITE</code> is allowed but interpreted as a normal time specification.• <code>TIME_IMMEDIATE</code> this value is not allowed.
----	------	---

Function Class:

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

7.22.4.8 void osalThreadSleep (systime_t time)

Suspends the invoking thread for the specified time.

Parameters

in	<i>time</i>	the delay in system ticks, the special values are handled as follow: <ul style="list-style-type: none"> • <i>TIME_INFINITE</i> is allowed but interpreted as a normal time specification. • <i>TIME_IMMEDIATE</i> this value is not allowed.
----	-------------	---

Function Class:

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

7.22.4.9 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	<i>trp</i>	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.

7.22.4.10 msg_t osalThreadSuspendTimeoutS (thread_reference_t * *trp*, systime_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

in	<i>trp</i>	a pointer to a thread reference object
in	<i>timeout</i>	the timeout in system ticks, the special values are handled as follow: <ul style="list-style-type: none"> • <i>TIME_INFINITE</i> the thread enters an infinite sleep state. • <i>TIME_IMMEDIATE</i> the thread is not enqueued and the function returns MSG_TIMEOUT as if a timeout occurred.

Returns

The wake up message.

Return values

<code>MSG_TIMEOUT</code>	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.

7.22.4.11 `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

in	<i>trp</i>	a pointer to a thread reference object
in	<i>msg</i>	the message code

Function Class:

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

7.22.4.12 `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

in	<i>trp</i>	a pointer to a thread reference object
in	<i>msg</i>	the message code

Function Class:

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

7.22.4.13 `msg_t osalThreadEnqueueTimeoutS (threads_queue_t * tqp, systime_t timeout)`

Enqueues the caller thread.

The caller thread is enqueued and put to sleep until it is dequeued or the specified timeouts expires.

Parameters

in	<i>tqp</i>	pointer to the threads queue object
in	<i>timeout</i>	<p>the timeout in system ticks, the special values are handled as follow:</p> <ul style="list-style-type: none"> • <i>TIME_INFINITE</i> the thread enters an infinite sleep state. • <i>TIME_IMMEDIATE</i> the thread is not enqueued and the function returns <code>MSG_TIMEOUT</code> as if a timeout occurred.

Returns

The message from `osalQueueWakeupOneI()` or `osalQueueWakeupAllI()` functions.

Return values

<code>MSG_TIMEOUT</code>	if the thread has not been dequeued within the specified timeout or if the function has been invoked with <code>TIME_IMMEDIATE</code> 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.

7.22.4.14 void osalThreadDequeueNextI(threads_queue_t *tqp, msg_t msg)

Dequeues and wakes up one thread from the queue, if any.

Parameters

in	<i>tqp</i>	pointer to the threads queue object
in	<i>msg</i>	the message code

Function Class:

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

7.22.4.15 void osalThreadDequeueAllI(threads_queue_t *tqp, msg_t msg)

Dequeues and wakes up all threads from the queue.

Parameters

in	<i>tqp</i>	pointer to the threads queue object
in	<i>msg</i>	the message code

Function Class:

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

7.22.4.16 void osalEventBroadcastFlagsI(event_source_t *esp, eventflags_t flags)

Add flags to an event source object.

Parameters

in	<i>esp</i>	pointer to the event flags object
in	<i>flags</i>	flags to be ORed to the flags mask

Function Class:

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

7.22.4.17 void osalEventBroadcastFlags(event_source_t *esp, eventflags_t flags)

Add flags to an event source object.

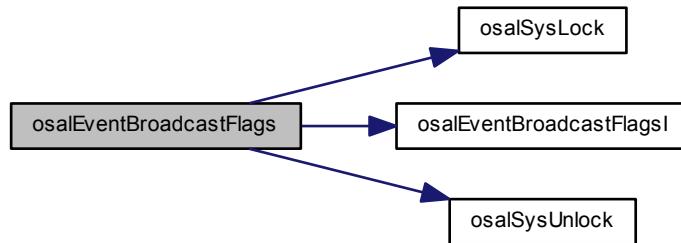
Parameters

in	<i>esp</i>	pointer to the event flags object
in	<i>flags</i>	flags to be ORed to the flags mask

Function Class:

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

Here is the call graph for this function:



7.22.4.18 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

in	<i>esp</i>	pointer to the event flags object
in	<i>cb</i>	pointer to the callback function
in	<i>param</i>	parameter to be passed to the callback function

Function Class:

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

7.22.4.19 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	<i>mp</i>	pointer to the <code>mutex_t</code> object
---------	-----------	--

Function Class:

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

7.22.4.20 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	<i>mp</i>	pointer to the <code>mutex_t</code> object
---------	-----------	--

Function Class:

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

7.22.4.21 static void osalSysDisable (void) [inline], [static]

Disables interrupts globally.

Function Class:

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

7.22.4.22 static void osalSysEnable (void) [inline], [static]

Enables interrupts globally.

Function Class:

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

7.22.4.23 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.

7.22.4.24 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.

7.22.4.25 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.

7.22.4.26 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.

7.22.4.27 static syssts_t osalSysGetStatusAndLockX (void) [inline], [static]

Returns the execution status and enters a critical zone.

This functions enters into a critical zone and can be called from any context. Because its flexibility it is less efficient than chSysLock () which is preferable when the calling context is known.

Postcondition

The system is in a critical zone.

Returns

The previous system status, the encoding of this status word is architecture-dependent and opaque.

Function Class:

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

7.22.4.28 static void osalSysRestoreStatusX(syssts_t sts) [inline], [static]

Restores the specified execution status and leaves a critical zone.

Note

A call to `chSchRescheduleS()` is automatically performed if exiting the critical zone and if not in ISR context.

Parameters

in	<i>sts</i>	the system status to be restored.
----	------------	-----------------------------------

Function Class:

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

7.22.4.29 static bool osalOsIsTimeWithinX (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 true because the whole time range is specified.
This function can be called from any context.

Parameters

in	<i>time</i>	the time to be verified
in	<i>start</i>	the start of the time window (inclusive)
in	<i>end</i>	the end of the time window (non inclusive)

Return values

<i>true</i>	current time within the specified time window.
<i>false</i>	current time not within the specified time window.

Function Class:

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

7.22.4.30 static void osalThreadQueueObjectInit (threads_queue_t * *tqp*) [inline], [static]

Initializes a threads queue object.

Parameters

out	<i>tqp</i>	pointer to the threads queue object
-----	------------	-------------------------------------

Function Class:

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

7.22.4.31 static void osalEventObjectInit (event_source_t * *esp*) [inline], [static]

Initializes an event flags object.

Parameters

out	<i>esp</i>	pointer to the event flags object
-----	------------	-----------------------------------

Function Class:

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

7.22.4.32 static void osalMutexObjectInit(mutex_t * *mp*) [inline], [static]

Initializes a mutex_t object.

Parameters

out	<i>mp</i>	pointer to the <code>mutex_t</code> object
-----	-----------	--

Function Class:

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

7.22.5 Variable Documentation

7.22.5.1 `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.

7.22.5.2 `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.

7.23 MMC over SPI Driver

Generic MMC driver.

7.23.1 Detailed Description

Generic MMC driver.

This module implements a portable MMC/SD driver that uses a SPI driver as physical layer. Hot plugging and removal are supported through kernel events.

Precondition

In order to use the MMC_SPI driver the HAL_USE_MMC_SPI and HAL_USE_SPI options must be enabled in `halconf.h`.

7.23.2 Driver State Machine

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

7.23.3 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`
MMCDriver specific methods.

MMC_SPI configuration options

- `#define MMC_NICE_WAITING TRUE`
Delays insertions.

Macro Functions

- `#define mmclsCardInserted(mmcp) mmc_lld_is_card_inserted(mmcp)`
Returns the card insertion status.
- `#define mmclsWriteProtected(mmcp) mmc_lld_is_write_protected(mmcp)`
Returns the write protect status.

Data Structures

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

Functions

- static uint8_t `crc7` (uint8_t crc, const uint8_t *buffer, size_t len)
Calculate the MMC standard CRC-7 based on a lookup table.
- static void `wait` (MMCDriver *mmcp)
Waits an idle condition.
- static void `send_hdr` (MMCDriver *mmcp, uint8_t cmd, uint32_t arg)
Sends a command header.
- static uint8_t `recv1` (MMCDriver *mmcp)
Receives a single byte response.
- static uint8_t `recv3` (MMCDriver *mmcp, uint8_t *buffer)
Receives a three byte response.
- static uint8_t `send_command_R1` (MMCDriver *mmcp, uint8_t cmd, uint32_t arg)
Sends a command and 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 MMCCConfig *config)
Configures and activates the MMC peripheral.
- void `mmcStop` (MMCDriver *mmcp)
Disables the MMC peripheral.
- bool `mmcConnect` (MMCDriver *mmcp)
Performs the initialization procedure on the inserted card.
- bool `mmcDisconnect` (MMCDriver *mmcp)
Brings the driver in a state safe for card removal.
- bool `mmcStartSequentialRead` (MMCDriver *mmcp, uint32_t startblk)
Starts a sequential read.
- bool `mmcSequentialRead` (MMCDriver *mmcp, uint8_t *buffer)
Reads a block within a sequential read operation.
- bool `mmcStopSequentialRead` (MMCDriver *mmcp)
Stops a sequential read gracefully.
- bool `mmcStartSequentialWrite` (MMCDriver *mmcp, uint32_t startblk)
Starts a sequential write.
- bool `mmcSequentialWrite` (MMCDriver *mmcp, const uint8_t *buffer)
Writes a block within a sequential write operation.
- bool `mmcStopSequentialWrite` (MMCDriver *mmcp)
Stops a sequential write gracefully.
- bool `mmcSync` (MMCDriver *mmcp)
Waits for card idle condition.
- bool `mmcGetInfo` (MMCDriver *mmcp, BlockDeviceInfo *bdip)
Returns the media info.
- bool `mmcErase` (MMCDriver *mmcp, uint32_t startblk, uint32_t endblk)
Erases blocks.

Variables

- static const struct [MMCDriverVMT](#) mmc_vmt
Virtual methods table.
- static const uint8_t [crc7_lookup_table](#) [256]
Lookup table for CRC-7 (based on polynomial $x^7 + x^3 + 1$).

7.23.4 Macro Definition Documentation

7.23.4.1 #define MMC_NICE_WAITING TRUE

Delays insertions.

If enabled this option inserts delays into the MMC waiting routines releasing some extra CPU time for the threads with lower priority, this may slow down the driver a bit however. This option is recommended also if the SPI driver does not use a DMA channel and heavily loads the CPU.

7.23.4.2 #define _mmc_driver_methods _mmcsd_block_device_methods

[MMCDriver](#) specific methods.

7.23.4.3 #define mmclsCardInserted(*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	<i>mmcp</i>	pointer to the MMCDriver object
----	-------------	---

Returns

The card state.

Return values

<i>FALSE</i>	card not inserted.
<i>TRUE</i>	card inserted.

Function Class:

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

7.23.4.4 #define mmclsWriteProtected(*mmcp*) mmc_lld_is_write_protected(*mmcp*)

Returns the write protect status.

Parameters

in	<i>mmcp</i>	pointer to the MMCDriver object
----	-------------	---

Returns

The card state.

Return values

<i>FALSE</i>	card not inserted.
<i>TRUE</i>	card inserted.

Function Class:

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

7.23.5 Function Documentation**7.23.5.1 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

in	<i>crc</i>	start value for CRC
in	<i>buffer</i>	pointer to data buffer
in	<i>len</i>	length of data

Returns

Calculated CRC

7.23.5.2 static void wait(MMCDriver * *mmcp*) [static]

Waits an idle condition.

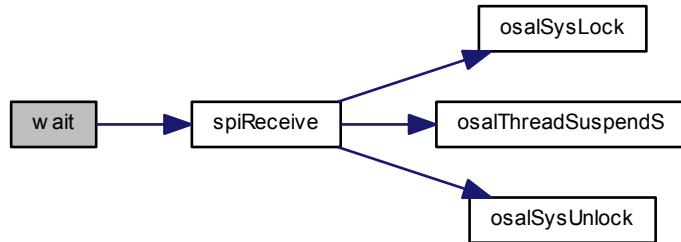
Parameters

in	<i>mmcp</i>	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:



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

Sends a command header.

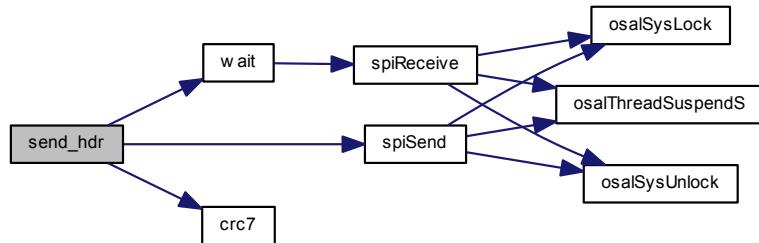
Parameters

in	<i>mmcp</i>	pointer to the MMCDriver object
in	<i>cmd</i>	the command id
in	<i>arg</i>	the command argument

Function Class:

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

Here is the call graph for this function:



7.23.5.4 static uint8_t recv1 (**MMCDriver** * *mmcp*) [static]

Receives a single byte response.

Parameters

in	<i>mmcp</i>	pointer to the MMCDriver object
----	-------------	---

Returns

The response as an `uint8_t` value.

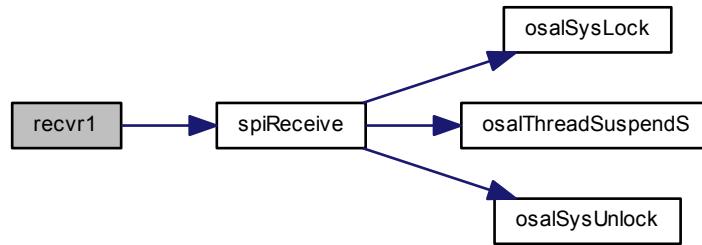
Return values

<i>0xFF</i>	timed out.
-------------	------------

Function Class:

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

Here is the call graph for this function:

**7.23.5.5 static uint8_t recv3 ([MMCDriver](#) * *mmcp*, `uint8_t` * *buffer*) [static]**

Receives a three byte response.

Parameters

in	<i>mmcp</i>	pointer to the MMCDriver object
out	<i>buffer</i>	pointer to four bytes wide buffer

Returns

First response byte as an `uint8_t` value.

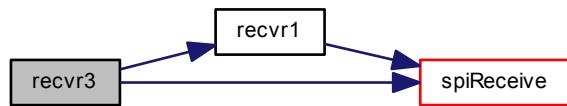
Return values

<i>0xFF</i>	timed out.
-------------	------------

Function Class:

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

Here is the call graph for this function:



7.23.5.6 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

in	<i>mmcp</i>	pointer to the <code>MMCDriver</code> object
in	<i>cmd</i>	the command id
in	<i>arg</i>	the command argument

Returns

The response as an `uint8_t` value.

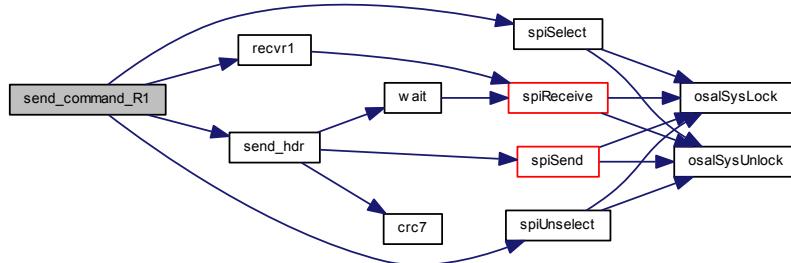
Return values

0xFF | timed out.

Function Class:

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

Here is the call graph for this function:



```
7.23.5.7 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

in	<i>mmcp</i>	pointer to the MMCDriver object
in	<i>cmd</i>	the command id
in	<i>arg</i>	the command argument
out	<i>response</i>	pointer to four bytes wide uint8_t buffer

Returns

The first byte of the response (R1) as an `uint8_t` value.

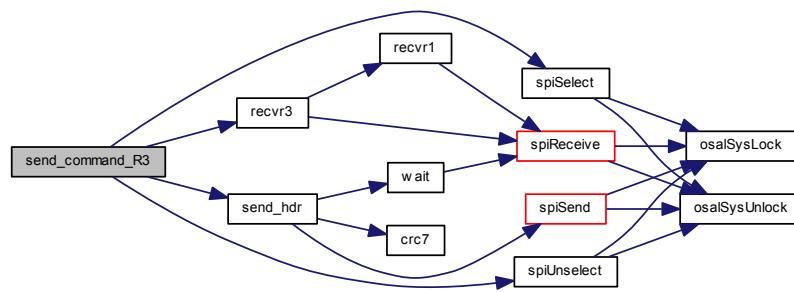
Return values

<code>0xFF</code>	timed out.
-------------------	------------

Function Class:

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

Here is the call graph for this function:

**7.23.5.8 static bool read_CxD ([MMCDriver](#) * *mmcp*, `uint8_t` *cmd*, `uint32_t` *cxd[4]*) [static]**

Reads the CSD.

Parameters

in	<i>mmcp</i>	pointer to the MMCDriver object
out	<i>cmd</i>	command
out	<i>cxd</i>	pointer to the CSD/CID buffer

Returns

The operation status.

Return values

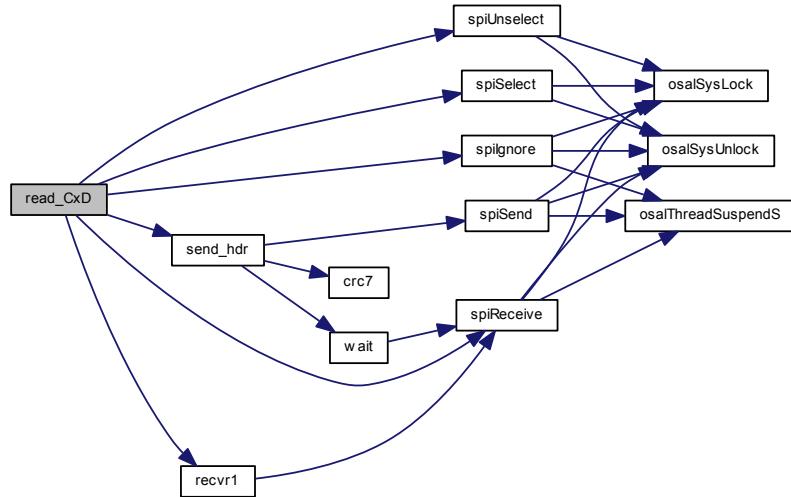
<code>HAL_SUCCESS</code>	the operation succeeded.
--------------------------	--------------------------

<code>HAL_FAILED</code>	the operation failed.
-------------------------	-----------------------

Function Class:

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

Here is the call graph for this function:

**7.23.5.9 static void sync (MMCDriver * mmcp) [static]**

Waits that the card reaches an idle state.

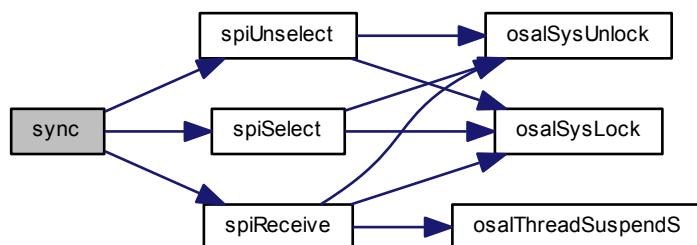
Parameters

in	<code>mmcp</code>	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:



7.23.5.10 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.

7.23.5.11 void mmcObjectInit(MMCDriver * mmcp)

Initializes an instance.

Parameters

out	<i>mmcp</i>	pointer to the MMCDriver object
-----	-------------	---

Function Class:

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

7.23.5.12 void mmcStart(MMCDriver * mmcp, const MMCCConfig * config)

Configures and activates the MMC peripheral.

Parameters

in	<i>mmcp</i>	pointer to the MMCDriver object
in	<i>config</i>	pointer to the MMCCConfig object.

Function Class:

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

7.23.5.13 void mmcStop(MMCDriver * mmcp)

Disables the MMC peripheral.

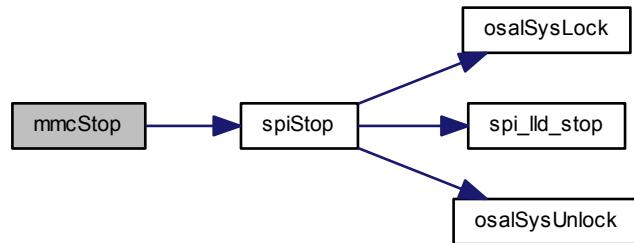
Parameters

in	<i>mmcp</i>	pointer to the MMCDriver object
----	-------------	---

Function Class:

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

Here is the call graph for this function:



7.23.5.14 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

<code>in</code>	<code>mmcp</code>	pointer to the MMCDriver object
-----------------	-------------------	---

Returns

The operation status.

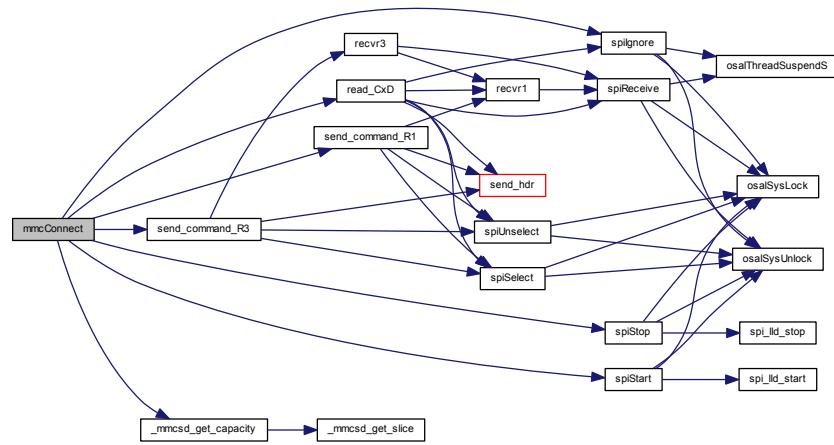
Return values

<code>HAL_SUCCESS</code>	the operation succeeded and the driver is now in the <code>MMC_READY</code> state.
<code>HAL_FAILED</code>	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:



7.23.5.15 bool mmcDisconnect ([MMCDriver](#) * *mmcp*)

Brings the driver in a state safe for card removal.

Parameters

in	<i>mmcp</i>	pointer to the MMCDriver object
----	-------------	---

Returns

The operation status.

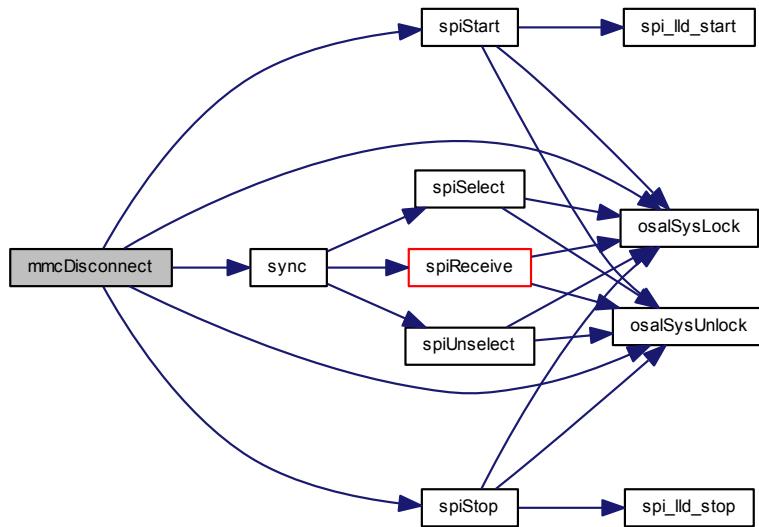
Return values

<i>HAL_SUCCESS</i>	the operation succeeded and the driver is now in the MMC_INSERTED state.
<i>HAL_FAILED</i>	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:



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

Starts a sequential read.

Parameters

<code>in</code>	<code>mmcp</code>	pointer to the <code>MMCDriver</code> object
<code>in</code>	<code>startblk</code>	first block to read

Returns

The operation status.

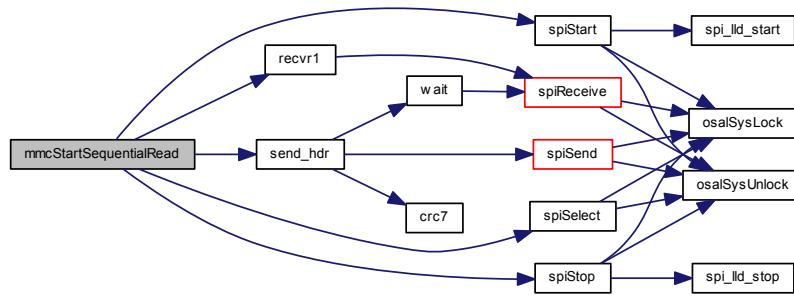
Return values

<code>HAL_SUCCESS</code>	the operation succeeded.
<code>HAL_FAILED</code>	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:



7.23.5.17 bool mmcSequentialRead (MMCDriver * mmcp, uint8_t * buffer)

Reads a block within a sequential read operation.

Parameters

in	<code>mmcp</code>	pointer to the <code>MMCDriver</code> object
out	<code>buffer</code>	pointer to the read buffer

Returns

The operation status.

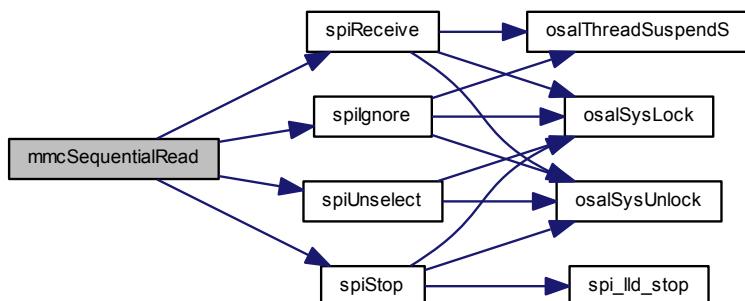
Return values

<code>HAL_SUCCESS</code>	the operation succeeded.
<code>HAL_FAILED</code>	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:



7.23.5.18 bool mmcStopSequentialRead (**MMCDriver * *mmcp*)**

Stops a sequential read gracefully.

Parameters

in	<i>mmcp</i>	pointer to the MMCDriver object
----	-------------	---

Returns

The operation status.

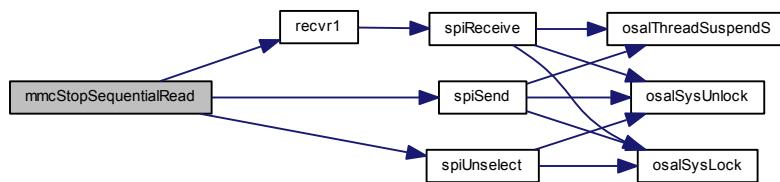
Return values

<i>HAL_SUCCESS</i>	the operation succeeded.
<i>HAL_FAILED</i>	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:

**7.23.5.19 bool mmcStartSequentialWrite([MMCDriver](#) * *mmcp*, uint32_t *startblk*)**

Starts a sequential write.

Parameters

in	<i>mmcp</i>	pointer to the MMCDriver object
in	<i>startblk</i>	first block to write

Returns

The operation status.

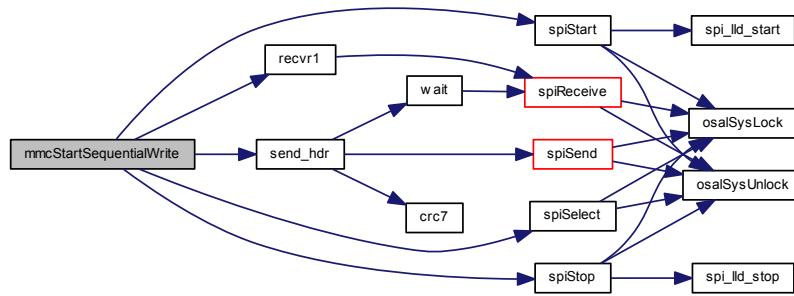
Return values

<i>HAL_SUCCESS</i>	the operation succeeded.
<i>HAL_FAILED</i>	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:



7.23.5.20 bool mmcSequentialWrite (MMCDDriver * mmcp, const uint8_t * buffer)

Writes a block within a sequential write operation.

Parameters

in	<i>mmcp</i>	pointer to the MMCDDriver object
out	<i>buffer</i>	pointer to the write buffer

Returns

The operation status.

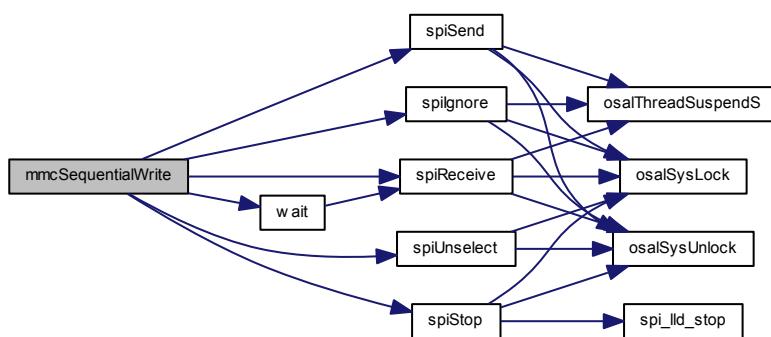
Return values

<i>HAL_SUCCESS</i>	the operation succeeded.
<i>HAL_FAILED</i>	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:



7.23.5.21 bool mmcStopSequentialWrite (**MMCDriver** * *mmcp*)

Stops a sequential write gracefully.

Parameters

in	<i>mmcp</i>	pointer to the MMCDriver object
----	-------------	---

Returns

The operation status.

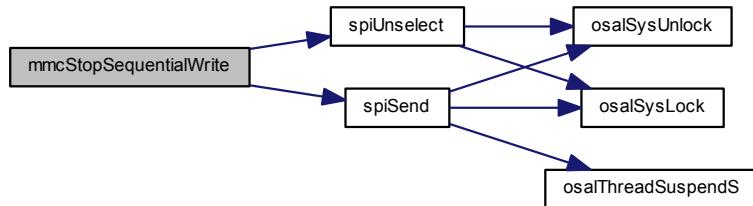
Return values

<i>HAL_SUCCESS</i>	the operation succeeded.
<i>HAL_FAILED</i>	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:

**7.23.5.22 bool mmcSync (MMCDriver * *mmcp*)**

Waits for card idle condition.

Parameters

in	<i>mmcp</i>	pointer to the MMCDriver object
----	-------------	---

Returns

The operation status.

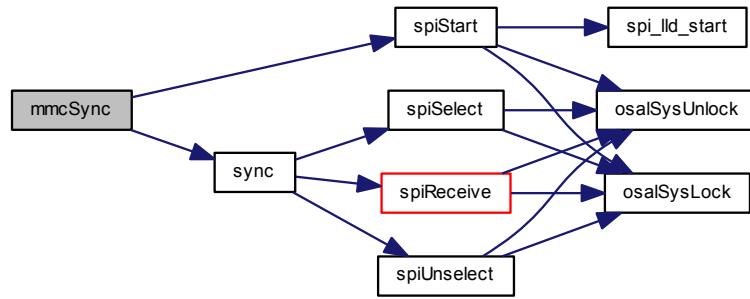
Return values

<i>HAL_SUCCESS</i>	the operation succeeded.
<i>HAL_FAILED</i>	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:



7.23.5.23 bool mmcGetInfo (**MMCDriver** * *mmcp*, **BlockDeviceInfo** * *bdip*)

Returns the media info.

Parameters

in	<i>mmcp</i>	pointer to the MMCDriver object
out	<i>bdip</i>	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.

7.23.5.24 bool mmcErase (**MMCDriver** * *mmcp*, **uint32_t** *startblk*, **uint32_t** *endblk*)

Erases blocks.

Parameters

in	<i>mmcp</i>	pointer to the MMCDriver object
in	<i>startblk</i>	starting block number
in	<i>endblk</i>	ending block number

Returns

The operation status.

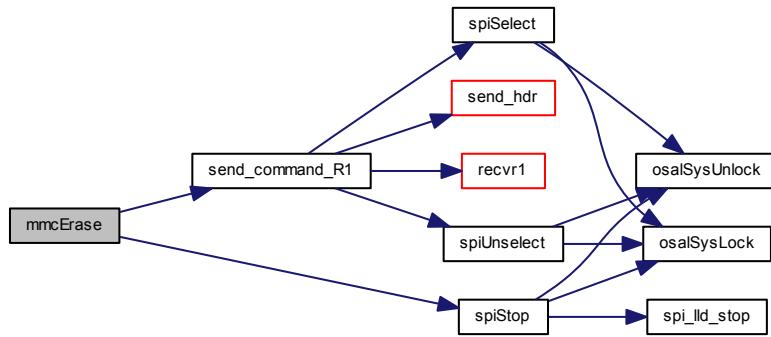
Return values

<i>HAL_SUCCESS</i>	the operation succeeded.
<i>HAL_FAILED</i>	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:



7.23.6 Variable Documentation

7.23.6.1 const struct MMCDriverVMT mmc_vmt [static]

Initial value:

```
= {
    (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.

7.23.6.2 const uint8_t crc7_lookup_table[256] [static]

Initial value:

```
= {
    0x00, 0x09, 0x12, 0x1b, 0x24, 0x2d, 0x36, 0x3f, 0x48, 0x41, 0x5a, 0x53,
    0x6c, 0x65, 0x7e, 0x77, 0x19, 0x10, 0xb, 0x02, 0x3d, 0x34, 0x2f, 0x26,
    0x51, 0x58, 0x43, 0xa, 0x75, 0x7c, 0x67, 0x6e, 0x32, 0x3b, 0x20, 0x29,
    0x16, 0x1f, 0x04, 0xd, 0x7a, 0x73, 0x68, 0x61, 0x5e, 0x57, 0x4c, 0x45,
    0x2b, 0x22, 0x39, 0x30, 0xf, 0x06, 0x1d, 0x14, 0x63, 0x6a, 0x71, 0x78,
    0x47, 0x4e, 0x55, 0x5c, 0x64, 0x6d, 0x76, 0x7f, 0x40, 0x49, 0x52, 0x5b,
    0x2c, 0x25, 0x3e, 0x37, 0x08, 0x01, 0x1a, 0x13, 0x7d, 0x74, 0x6f, 0x66,
    0x59, 0x50, 0x4b, 0x42, 0x35, 0x3c, 0x27, 0x2e, 0x11, 0x18, 0x03, 0xa,
    0x56, 0x5f, 0x44, 0x4d, 0x72, 0x7b, 0x60, 0x69, 0x1e, 0x17, 0x0c, 0x05,
    0x3a, 0x33, 0x28, 0x21, 0x4f, 0x46, 0x5d, 0x54, 0x6b, 0x62, 0x79, 0x70,
    0x07, 0xe, 0x15, 0x1c, 0x23, 0x2a, 0x31, 0x38, 0x41, 0x48, 0x53, 0x5a,
```

```
0x65, 0x6c, 0x77, 0x7e, 0x09, 0x00, 0x1b, 0x12, 0x2d, 0x24, 0x3f, 0x36,
0x58, 0x51, 0x4a, 0x43, 0x7c, 0x75, 0x6e, 0x67, 0x10, 0x19, 0x02, 0x0b,
0x34, 0x3d, 0x26, 0x2f, 0x73, 0x7a, 0x61, 0x68, 0x57, 0x5e, 0x45, 0x4c,
0x3b, 0x32, 0x29, 0x20, 0x1f, 0x16, 0x0d, 0x04, 0x6a, 0x63, 0x78, 0x71,
0x4e, 0x47, 0x5c, 0x55, 0x22, 0x2b, 0x30, 0x39, 0x06, 0x0f, 0x14, 0x1d,
0x25, 0x2c, 0x37, 0x3e, 0x01, 0x08, 0x13, 0x1a, 0x6d, 0x64, 0x7f, 0x76,
0x49, 0x40, 0x5b, 0x52, 0x3c, 0x35, 0x2e, 0x27, 0x18, 0x11, 0x0a, 0x03,
0x74, 0x7d, 0x66, 0x6f, 0x50, 0x59, 0x42, 0x4b, 0x17, 0x1e, 0x05, 0x0c,
0x33, 0x3a, 0x21, 0x28, 0x5f, 0x56, 0x4d, 0x44, 0x7b, 0x72, 0x69, 0x60,
0x0e, 0x07, 0x1c, 0x15, 0x2a, 0x23, 0x38, 0x31, 0x46, 0x4f, 0x54, 0x5d,
0x62, 0x6b, 0x70, 0x79
}
```

Lookup table for CRC-7 (based on polynomial $x^7 + x^3 + 1$).

7.24 MMC/SD Block Device

7.24.1 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 0xFDFE008U`
Mask of error bits in R1 responses.
- `#define MMCSD_CMD8_PATTERN 0x000001AAU`
Fixed pattern for CMD8.
- `#define _mmcsd_block_device_methods _base_block_device_methods`
MMCSDBlockDevice specific methods.
- `#define _mmcsd_block_device_data`
MMCSDBlockDevice specific data.

SD/MMC status conditions

- `#define MMCSD_STS_IDLE 0U`
- `#define MMCSD_STS_READY 1U`
- `#define MMCSD_STS_IDENT 2U`
- `#define MMCSD_STS_STBY 3U`
- `#define MMCSD_STS_TRAN 4U`
- `#define MMCSD_STS_DATA 5U`
- `#define MMCSD_STS_RCV 6U`
- `#define MMCSD_STS_PRG 7U`
- `#define MMCSD_STS_DIS 8U`

SD/MMC commands

- `#define MMCSD_CMD_GO_IDLE_STATE 0U`
- `#define MMCSD_CMD_INIT 1U`
- `#define MMCSD_CMD_ALL_SEND_CID 2U`
- `#define MMCSD_CMD_SEND_RELATIVE_ADDR 3U`
- `#define MMCSD_CMD_SET_BUS_WIDTH 6U`
- `#define MMCSD_CMD_SWITCH MMCSD_CMD_SET_BUS_WIDTH`
- `#define MMCSD_CMD_SEL_DESEL_CARD 7U`
- `#define MMCSD_CMD_SEND_IF_COND 8U`
- `#define MMCSD_CMD_SEND_EXT_CSD MMCSD_CMD_SEND_IF_COND`
- `#define MMCSD_CMD_SEND_CSD 9U`
- `#define MMCSD_CMD_SEND_CID 10U`
- `#define MMCSD_CMD_STOP_TRANSMISSION 12U`
- `#define MMCSD_CMD_SEND_STATUS 13U`
- `#define MMCSD_CMD_SET_BLOCKLEN 16U`
- `#define MMCSD_CMD_READ_SINGLE_BLOCK 17U`
- `#define MMCSD_CMD_READ_MULTIPLE_BLOCK 18U`
- `#define MMCSD_CMD_SET_BLOCK_COUNT 23U`
- `#define MMCSD_CMD_WRITE_BLOCK 24U`

- #define **MMCSD_CMD_WRITE_MULTIPLE_BLOCK** 25U
- #define **MMCSD_CMD_ERASE_RW_BLK_START** 32U
- #define **MMCSD_CMD_ERASE_RW_BLK_END** 33U
- #define **MMCSD_CMD_ERASE** 38U
- #define **MMCSD_CMD_APP_OP_COND** 41U
- #define **MMCSD_CMD_LOCK_UNLOCK** 42U
- #define **MMCSD_CMD_APP_CMD** 55U
- #define **MMCSD_CMD_READ_OCR** 58U

CSD record offsets

- #define **MMCSD_CSD_MMC_CSD_STRUCTURE_SLICE** 127U,126U
Slice position of values in CSD register.
- #define **MMCSD_CSD_MMC_SPEC_VERS_SLICE** 125U,122U
- #define **MMCSD_CSD_MMC_TAAC_SLICE** 119U,112U
- #define **MMCSD_CSD_MMC_NSAC_SLICE** 111U,104U
- #define **MMCSD_CSD_MMC_TRAN_SPEED_SLICE** 103U,96U
- #define **MMCSD_CSD_MMC_CCC_SLICE** 95U,84U
- #define **MMCSD_CSD_MMC_READ_BL_LEN_SLICE** 83U,80U
- #define **MMCSD_CSD_MMC_READ_BL_PARTIAL_SLICE** 79U,79U
- #define **MMCSD_CSD_MMC_WRITE_BLK_MISALIGN_SLICE** 78U,78U
- #define **MMCSD_CSD_MMC_READ_BLK_MISALIGN_SLICE** 77U,77U
- #define **MMCSD_CSD_MMC_DSR_IMP_SLICE** 76U,76U
- #define **MMCSD_CSD_MMC_C_SIZE_SLICE** 73U,62U
- #define **MMCSD_CSD_MMC_VDD_R_CURR_MIN_SLICE** 61U,59U
- #define **MMCSD_CSD_MMC_VDD_R_CURR_MAX_SLICE** 58U,56U
- #define **MMCSD_CSD_MMC_VDD_W_CURR_MIN_SLICE** 55U,53U
- #define **MMCSD_CSD_MMC_VDD_W_CURR_MAX_SLICE** 52U,50U
- #define **MMCSD_CSD_MMC_C_SIZE_MULT_SLICE** 49U,47U
- #define **MMCSD_CSD_MMC_ERASE_GRP_SIZE_SLICE** 46U,42U
- #define **MMCSD_CSD_MMC_ERASE_GRP_MULT_SLICE** 41U,37U
- #define **MMCSD_CSD_MMC_WP_GRP_SIZE_SLICE** 36U,32U
- #define **MMCSD_CSD_MMC_WP_GRP_ENABLE_SLICE** 31U,31U
- #define **MMCSD_CSD_MMC_DEFAULT_ECC_SLICE** 30U,29U
- #define **MMCSD_CSD_MMC_R2W_FACTOR_SLICE** 28U,26U
- #define **MMCSD_CSD_MMC_WRITE_BL_LEN_SLICE** 25U,22U
- #define **MMCSD_CSD_MMC_WRITE_BL_PARTIAL_SLICE** 21U,21U
- #define **MMCSD_CSD_MMC_CONTENT_PROT_APP_SLICE** 16U,16U
- #define **MMCSD_CSD_MMC_FILE_FORMAT_GRP_SLICE** 15U,15U
- #define **MMCSD_CSD_MMC_COPY_SLICE** 14U,14U
- #define **MMCSD_CSD_MMC_PERM_WRITE_PROTECT_SLICE** 13U,13U
- #define **MMCSD_CSD_MMC_TMP_WRITE_PROTECT_SLICE** 12U,12U
- #define **MMCSD_CSD_MMC_FILE_FORMAT_SLICE** 11U,10U
- #define **MMCSD_CSD_MMC_ECC_SLICE** 9U,8U
- #define **MMCSD_CSD_MMC_CRC_SLICE** 7U,1U
- #define **MMCSD_CSD_20_CRC_SLICE** 7U,1U
- #define **MMCSD_CSD_20_FILE_FORMAT_SLICE** 11U,10U
- #define **MMCSD_CSD_20_TMP_WRITE_PROTECT_SLICE** 12U,12U
- #define **MMCSD_CSD_20_PERM_WRITE_PROTECT_SLICE** 13U,13U
- #define **MMCSD_CSD_20_COPY_SLICE** 14U,14U
- #define **MMCSD_CSD_20_FILE_FORMAT_GRP_SLICE** 15U,15U
- #define **MMCSD_CSD_20_WRITE_BL_PARTIAL_SLICE** 21U,21U
- #define **MMCSD_CSD_20_WRITE_BL_LEN_SLICE** 25U,12U
- #define **MMCSD_CSD_20_R2W_FACTOR_SLICE** 28U,26U

- #define **MMCSD_CSD_20_WP_GRP_ENABLE_SLICE** 31U,31U
- #define **MMCSD_CSD_20_WP_GRP_SIZE_SLICE** 38U,32U
- #define **MMCSD_CSD_20_ERASE_SECTOR_SIZE_SLICE** 45U,39U
- #define **MMCSD_CSD_20_ERASE_BLK_EN_SLICE** 46U,46U
- #define **MMCSD_CSD_20_C_SIZE_SLICE** 69U,48U
- #define **MMCSD_CSD_20_DSR_IMP_SLICE** 76U,76U
- #define **MMCSD_CSD_20_READ_BLK_MISALIGN_SLICE** 77U,77U
- #define **MMCSD_CSD_20_WRITE_BLK_MISALIGN_SLICE** 78U,78U
- #define **MMCSD_CSD_20_READ_BL_PARTIAL_SLICE** 79U,79U
- #define **MMCSD_CSD_20_READ_BL_LEN_SLICE** 83U,80U
- #define **MMCSD_CSD_20_CCC_SLICE** 95U,84U
- #define **MMCSD_CSD_20_TRANS_SPEED_SLICE** 103U,96U
- #define **MMCSD_CSD_20_NSAC_SLICE** 111U,104U
- #define **MMCSD_CSD_20_TAAC_SLICE** 119U,112U
- #define **MMCSD_CSD_20_CSD_STRUCTURE_SLICE** 127U,126U
- #define **MMCSD_CSD_10_CRC_SLICE** MMCSD_CSD_20_CRC_SLICE
- #define **MMCSD_CSD_10_FILE_FORMAT_SLICE** MMCSD_CSD_20_FILE_FORMAT_SLICE
- #define **MMCSD_CSD_10_TMP_WRITE_PROTECT_SLICE** MMCSD_CSD_20_TMP_WRITE_PROTECT_SLICE
- #define **MMCSD_CSD_10_PERM_WRITE_PROTECT_SLICE** MMCSD_CSD_20_PERM_WRITE_PROTECT_SLICE
- #define **MMCSD_CSD_10_COPY_SLICE** MMCSD_CSD_20_COPY_SLICE
- #define **MMCSD_CSD_10_FILE_FORMAT_GRP_SLICE** MMCSD_CSD_20_FILE_FORMAT_GRP_SLICE
- #define **MMCSD_CSD_10_WRITE_BL_PARTIAL_SLICE** MMCSD_CSD_20_WRITE_BL_PARTIAL_SLICE
- #define **MMCSD_CSD_10_WRITE_BL_LEN_SLICE** MMCSD_CSD_20_WRITE_BL_LEN_SLICE
- #define **MMCSD_CSD_10_R2W_FACTOR_SLICE** MMCSD_CSD_20_R2W_FACTOR_SLICE
- #define **MMCSD_CSD_10_WP_GRP_ENABLE_SLICE** MMCSD_CSD_20_WP_GRP_ENABLE_SLICE
- #define **MMCSD_CSD_10_WP_GRP_SIZE_SLICE** MMCSD_CSD_20_WP_GRP_SIZE_SLICE
- #define **MMCSD_CSD_10_ERASE_SECTOR_SIZE_SLICE** MMCSD_CSD_20_ERASE_SECTOR_SIZE_SLICE
- #define **MMCSD_CSD_10_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
Slice position of values in CID register.
- #define **MMCSD_CID_SDC_MDT_M_SLICE** 11U,8U
- #define **MMCSD_CID_SDC_MDT_Y_SLICE** 19U,12U
- #define **MMCSD_CID_SDC_PSN_SLICE** 55U,24U
- #define **MMCSD_CID_SDC_PRV_M_SLICE** 59U,56U
- #define **MMCSD_CID_SDC_PRV_N_SLICE** 63U,60U
- #define **MMCSD_CID_SDC_PNM0_SLICE** 71U,64U
- #define **MMCSD_CID_SDC_PNM1_SLICE** 79U,72U
- #define **MMCSD_CID_SDC_PNM2_SLICE** 87U,80U
- #define **MMCSD_CID_SDC_PNM3_SLICE** 95U,88U
- #define **MMCSD_CID_SDC_PNM4_SLICE** 103U,96U
- #define **MMCSD_CID_SDC_OID_SLICE** 119U,104U
- #define **MMCSD_CID_SDC_MID_SLICE** 127U,120U
- #define **MMCSD_CID_MMCCRC_SLICE** 7U,1U
- #define **MMCSD_CID_MMCMDTY_SLICE** 11U,8U
- #define **MMCSD_CID_MMCMDTM_SLICE** 15U,12U
- #define **MMCSD_CID_MMCPSON_SLICE** 47U,16U
- #define **MMCSD_CID_MMCPRM_SLICE** 51U,48U
- #define **MMCSD_CID_MMCPRN_SLICE** 55U,52U
- #define **MMCSD_CID_MMCPNM0_SLICE** 63U,56U
- #define **MMCSD_CID_MMCPNM1_SLICE** 71U,64U
- #define **MMCSD_CID_MMCPNM2_SLICE** 79U,72U
- #define **MMCSD_CID_MMCPNM3_SLICE** 87U,80U
- #define **MMCSD_CID_MMCPNM4_SLICE** 95U,88U
- #define **MMCSD_CID_MMCPNM5_SLICE** 103U,96U
- #define **MMCSD_CID_MMCOID_SLICE** 119U,104U
- #define **MMCSD_CID_MMCMID_SLICE** 127U,120U

R1 response utilities

- #define **MMCSD_R1_ERROR**(r1) (((r1) & **MMCSD_R1_ERROR_MASK**) != 0U)
Evaluates to TRUE if the R1 response contains error flags.
- #define **MMCSD_R1_STS**(r1) (((r1) >> 9U) & 15U)
Returns the status field of an R1 response.
- #define **MMCSD_R1_IS_CARD_LOCKED**(r1) (((((r1) >> 21U) & 1U) != 0U)
Evaluates to TRUE if the R1 response indicates a locked card.

Macro Functions

- #define **mmcsdGetCardCapacity**(ip) ((ip)->capacity)
Returns the card capacity in blocks.

Data Structures

- struct **MMCSDBlockDeviceVMT**
MMCSDBlockDevice virtual methods table.
- struct **MMCSDBlockDevice**
MCC/SD block device class.
- struct **unpacked_sdc_cid_t**

- struct [unpacked_mmc_cid_t](#)
Unpacked CID register from SDC.
- 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.

7.24.2 Macro Definition Documentation

7.24.2.1 #define MMCSD_BLOCK_SIZE 512U

Fixed block size for MMC/SD block devices.

7.24.2.2 #define MMCSD_R1_ERROR_MASK 0xFDFFE008U

Mask of error bits in R1 responses.

7.24.2.3 #define MMCSD_CMD8_PATTERN 0x0000001AAU

Fixed pattern for CMD8.

7.24.2.4 #define MMCSD_CSD_MMC_CSD_STRUCTURE_SLICE 127U,126U

Slice position of values in CSD register.

7.24.2.5 #define MMCSD_CID_SDC_CRC_SLICE 7U,1U

Slice position of values in CID register.

7.24.2.6 #define _mmcsd_block_device_methods _base_block_device_methods

`MMCSDBlockDevice` specific methods.

7.24.2.7 #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.

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

Evaluates to TRUE if the R1 response contains error flags.

Parameters

in	r1	the r1 response
----	----	-----------------

7.24.2.9 #define MMCSD_R1_STS(r1) (((r1) >> 9U) & 15U)

Returns the status field of an R1 response.

Parameters

in	r1	the r1 response
----	----	-----------------

7.24.2.10 #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
----	----	-----------------

7.24.2.11 #define mmcsdGetCardCapacity(ip) ((ip)->capacity)

Returns the card capacity in blocks.

Parameters

in	ip	pointer to a <code>MMCSDBlockDevice</code> 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.

7.24.3 Function Documentation**7.24.3.1 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

in	<i>data</i>	pointer to the words array
in	<i>end</i>	bit offset of the last bit of the field, inclusive
in	<i>start</i>	bit offset of the first bit of the field, inclusive

Returns

The bits field value, left aligned.

Function Class:

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

7.24.3.2 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

in	<i>csd</i>	the CSD record
----	------------	----------------

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:



7.24.3.3 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	<i>ext_csd</i>	the extended CSD record
----	----------------	-------------------------

Returns

The card capacity.

Function Class:

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

7.24.3.4 void _mmcsd_unpack_sdc_cid (const MMCSDBlockDevice * sdc, unpacked_sdc_cid_t * cidsdc)

Unpacks SDC CID array in structure.

Parameters

in	<i>sdc</i>	pointer to the MMCSDBlockDevice object
out	<i>cidsdc</i>	pointer to the unpacked_sdc_cid_t object

Function Class:

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

Here is the call graph for this function:



7.24.3.5 void _mmcsd_unpack_mmc_cid (const MMCSDBlockDevice * sdc, unpacked_mmc_cid_t * cidmmc)

Unpacks MMC CID array in structure.

Parameters

in	<i>sdcp</i>	pointer to the MMCSDBlockDevice object
out	<i>cidmmc</i>	pointer to the unpacked_mmc_cid_t object

Function Class:

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

Here is the call graph for this function:

**7.24.3.6 void _mmcsd_unpack_csd_mmc (const MMCSDBlockDevice * *sdcp*, unpacked_mmc_csd_t * *csdmmc*)**

Unpacks MMC CSD array in structure.

Parameters

in	<i>sdcp</i>	pointer to the MMCSDBlockDevice object
out	<i>csdmmc</i>	pointer to the unpacked_mmc_csd_t object

Function Class:

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

Here is the call graph for this function:

**7.24.3.7 void _mmcsd_unpack_csd_v10 (const MMCSDBlockDevice * *sdcp*, unpacked_sdc_csd_10_t * *csd10*)**

Unpacks SDC CSD v1.0 array in structure.

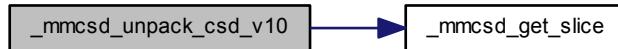
Parameters

in	<i>sdcp</i>	pointer to the MMCSDBlockDevice object
out	<i>csd10</i>	pointer to the unpacked_sdc_csd_10_t object

Function Class:

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

Here is the call graph for this function:



7.24.3.8 void _mmcsd_unpack_csd_v20(const MMCSDBlockDevice * *sdcp*, unpacked_sdc_csd_20_t * *csd20*)

Unpacks SDC CSD v2.0 array in structure.

Parameters

in	<i>sdcp</i>	pointer to the MMCSDBlockDevice object
out	<i>csd20</i>	pointer to the unpacked_sdc_csd_20_t object

Function Class:

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

Here is the call graph for this function:



7.25 PAL Driver

I/O Ports Abstraction Layer.

7.25.1 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](#) 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`.

7.25.2 Implementation Rules

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

7.25.2.1 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.

7.25.2.2 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.

7.25.2.3 Writing unused or unimplemented port bits

The behavior is not specified.

7.25.2.4 Reading from unused or unimplemented port bits

The behavior is not specified.

7.25.2.5 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 PAL_IOPORTS_WIDTH 32U`
Width, in bits, of an I/O port.
- `#define PAL_WHOLE_PORT ((ioportmask_t)0xFFFFFFFFU)`
Whole port mask.
- `#define IOPORT1 0`
First I/O port identifier.
- `#define pal_lld_init(config) _pal_lld_init(config)`
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.

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.

Macro Functions

- `#define pallInit(config) pal_lld_init(config)`
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))`

- `#define palWriteGroup(port, mask, offset, bits)`

Reads a group of bits.
- `#define palSetGroupMode(port, mask, offset, mode)`

Writes a group of bits.
- `#define palReadPad(port, pad) ((palReadPort(port) >> (pad)) & 1U)`

Pads group mode setup.
- `#define palWritePad(port, pad, bit)`

Reads an input pad logical state.
- `#define palSetPad(port, pad) palSetPort(port, PAL_PORT_BIT(pad))`

Writes a logical state on an output pad.
- `#define palClearPad(port, pad) palClearPort(port, PAL_PORT_BIT(pad))`

Sets a pad logical state to PAL_HIGH.
- `#define palTogglePad(port, pad) palTogglePort(port, PAL_PORT_BIT(pad))`

Clears a pad logical state to PAL_LOW.
- `#define palSetPadMode(port, pad, mode) palSetGroupMode(port, PAL_PORT_BIT(pad), 0U, mode)`

Toggles a pad logical state.
- `#define palSetPadMode(port, pad, mode) palSetGroupMode(port, PAL_PORT_BIT(pad), 0U, mode)`

Pad mode setup.

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 ioportid_t`

Port Identifier.

Data Structures

- `struct IOBus`

I/O bus descriptor.
- `struct PALConfig`

Generic I/O ports static initializer.

Functions

- `ioportmask_t palReadBus (IOBus *bus)`

Read from an I/O bus.
- `void palWriteBus (IOBus *bus, ioportmask_t bits)`

Write to an I/O bus.
- `void palSetBusMode (IOBus *bus, iomode_t mode)`

Programs a bus with the specified mode.
- `void _pal_lld_init (const PALConfig *config)`

STM32 I/O ports configuration.
- `void _pal_lld_setgroupmode (ioportid_t port, ioportmask_t mask, iomode_t mode)`

Pads mode setup.

7.25.3 Macro Definition Documentation

7.25.3.1 #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.

7.25.3.2 #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_PUSH_PULL` for example.

7.25.3.3 #define PAL_MODE_INPUT 2U

Regular input high-Z pad.

7.25.3.4 #define PAL_MODE_INPUT_PULLUP 3U

Input pad with weak pull up resistor.

7.25.3.5 #define PAL_MODE_INPUT_PULLDOWN 4U

Input pad with weak pull down resistor.

7.25.3.6 #define PAL_MODE_INPUT_ANALOG 5U

Analog input mode.

7.25.3.7 #define PAL_MODE_OUTPUT_PUSH_PULL 6U

Push-pull output pad.

7.25.3.8 #define PAL_MODE_OUTPUT_OPENDRAIN 7U

Open-drain output pad.

7.25.3.9 #define PAL_LOW 0U

Logical low state.

7.25.3.10 #define PAL_HIGH 1U

Logical high state.

```
7.25.3.11 #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	<i>n</i>	bit position within the port
----	----------	------------------------------

Returns

The bit mask.

7.25.3.12 #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	<i>width</i>	group width
----	--------------	-------------

Returns

The group mask.

7.25.3.13 #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	<i>name</i>	name of the IOBus variable
in	<i>port</i>	I/O port descriptor
in	<i>width</i>	bus width in bits
in	<i>offset</i>	bus bit offset within the port

7.25.3.14 #define IOBUS_DECL(*name*, *port*, *width*, *offset*) IOBus name = _IOBUS_DATA(*name*, *port*, *width*, *offset*)

Static I/O bus initializer.

Parameters

in	<i>name</i>	name of the IOBus variable
in	<i>port</i>	I/O port descriptor
in	<i>width</i>	bus width in bits
in	<i>offset</i>	bus bit offset within the port

7.25.3.15 #define pallInit(*config*) pal_lld_init(*config*)

PAL subsystem initialization.

Note

This function is implicitly invoked by [halInit\(\)](#) , there is no need to explicitly initialize the driver.

Parameters

in	<i>config</i>	pointer to an architecture specific configuration structure. This structure is defined in the low level driver header.
----	---------------	--

Function Class:

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

7.25.3.16 #define palReadPort(*port*) ((void)(port), 0U)

Reads the physical I/O port states.

Note

The default implementation always return zero and computes the parameter eventual side effects.
The function can be called from any context.

Parameters

in	<i>port</i>	port identifier
----	-------------	-----------------

Returns

The port logical states.

Function Class:

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

7.25.3.17 #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 default implementation always return zero and computes the parameter eventual side effects.
The function can be called from any context.

Parameters

in	<i>port</i>	port identifier
----	-------------	-----------------

Returns

The latched logical states.

Function Class:

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

7.25.3.18 #define palWritePort(*port*, *bits*) ((void)(port), (void)(bits))

Writes a bits mask on a I/O port.

Note

The default implementation does nothing except computing the parameters eventual side effects.
The function can be called from any context.

Parameters

in	<i>port</i>	port identifier
in	<i>bits</i>	bits to be written on the specified port

Function Class:

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

7.25.3.19 #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 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

in	<i>port</i>	port identifier
in	<i>bits</i>	bits to be ORed on the specified port

Function Class:

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

7.25.3.20 #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 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

in	<i>port</i>	port identifier
in	<i>bits</i>	bits to be cleared on the specified port

Function Class:

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

7.25.3.21 #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 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

in	<i>port</i>	port identifier
in	<i>bits</i>	bits to be XORed on the specified port

Function Class:

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

7.25.3.22 #define `palReadGroup(port, mask, offset) ((palReadPort(port) >> (offset)) & (mask))`

Reads a group of bits.

Note

The function can be called from any context.

Parameters

in	<i>port</i>	port identifier
in	<i>mask</i>	group mask, a logical AND is performed on the input data
in	<i>offset</i>	group bit offset within the port

Returns

The group logical states.

Function Class:

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

7.25.3.23 #define `palWriteGroup(port, mask, offset, bits)`

Value:

```
palWritePort(port, (palReadLatch(port) & ~((mask) << (offset))) | \
((bits) & (mask)) << (offset))) \
```

Writes a group of bits.

Note

The function can be called from any context.

Parameters

in	<i>port</i>	port identifier
in	<i>mask</i>	group mask, a logical AND is performed on the output data
in	<i>offset</i>	group bit offset within the port
in	<i>bits</i>	bits to be written. Values exceeding the group width are masked.

Function Class:

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

7.25.3.24 #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

Programming an unknown or unsupported mode is silently ignored.

The function can be called from any context.

Parameters

in	<i>port</i>	port identifier
in	<i>mask</i>	group mask
in	<i>offset</i>	group bit offset within the port
in	<i>mode</i>	group mode

Function Class:

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

7.25.3.25 #define palReadPad(*port*, *pad*)((palReadPort(*port*) >> (*pad*)) & 1U)

Reads an input pad logical state.

Note

The default implementation not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

The default implementation internally uses the [palReadPort\(\)](#).

The function can be called from any context.

Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port

Returns

The logical state.

Return values

<i>PAL_LOW</i>	low logical state.
<i>PAL_HIGH</i>	high logical state.

Function Class:

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

7.25.3.26 #define palWritePad(*port*, *pad*, *bit*)**Value:**

```
palWritePort(port, (palReadLatch(port) & ~PAL_PORT_BIT(pad)) |
    (((bit) & 1U) << pad))
```

Writes a logical 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 default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

The default implementation internally uses the *palReadLatch()* and *palWritePort()*.

The function can be called from any context.

Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port
in	<i>bit</i>	logical value, the value must be <i>PAL_LOW</i> or <i>PAL_HIGH</i>

Function Class:

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

7.25.3.27 #define palSetPad(*port*, *pad*) palSetPort(*port*, PAL_PORT_BIT(*pad*))

Sets a pad logical 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 default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

The default implementation internally uses the *palSetPort()*.

The function can be called from any context.

Parameters

in	<i>port</i>	port identifier
----	-------------	-----------------

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port

Function Class:

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

7.25.3.28 #define palClearPad(*port*, *pad*) palClearPort(*port*, PAL_PORT_BIT(*pad*))

Clears a pad logical 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 default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

The default implementation internally uses the `palClearPort()`.

The function can be called from any context.

Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port

Function Class:

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

7.25.3.29 #define palTogglePad(*port*, *pad*) palTogglePort(*port*, PAL_PORT_BIT(*pad*))

Toggles a pad logical 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 default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

The default implementation internally uses the `palTogglePort()`.

The function can be called from any context.

Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port

Function Class:

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

7.25.3.30 #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 default implementation not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

Programming an unknown or unsupported mode is silently ignored.

The function can be called from any context.

Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port
in	<i>mode</i>	pad mode

Function Class:

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

7.25.3.31 #define PAL_IOPORTS_WIDTH 32U

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

7.25.3.32 #define PAL_WHOLE_PORT ((ioportmask_t)0xFFFFFFFFU)

Whole port mask.

This macro specifies all the valid bits into a port.

7.25.3.33 #define IOPORT1 0

First I/O port identifier.

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

7.25.3.34 #define pal_lld_init(config) _pal_lld_init(config)

Low level PAL subsystem initialization.

Parameters

in	<i>config</i>	architecture-dependent ports configuration
----	---------------	--

Function Class:

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

7.25.3.35 #define pal_lld_readport(port) 0U

Reads the physical I/O port states.

Parameters

in	<i>port</i>	port identifier
----	-------------	-----------------

Returns

The port bits.

Function Class:

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

7.25.3.36 #define pal_lld_readlatch(*port*) 0U

Reads the output latch.

The purpose of this function is to read back the latched output value.

Parameters

in	<i>port</i>	port identifier
----	-------------	-----------------

Returns

The latched logical states.

Function Class:

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

7.25.3.37 #define pal_lld_writeport(*port*, *bits*)**Value:**

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

Writes a bits mask on a I/O port.

Parameters

in	<i>port</i>	port identifier
in	<i>bits</i>	bits to be written on the specified port

Function Class:

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

7.25.3.38 #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

in	<i>port</i>	port identifier
in	<i>bits</i>	bits to be ORed on the specified port

Function Class:

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

7.25.3.39 #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

in	<i>port</i>	port identifier
in	<i>bits</i>	bits to be cleared on the specified port

Function Class:

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

7.25.3.40 #define pal_lld_toggleport(*port*, *bits*)**Value:**

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

Toggles a bits mask on a I/O port.

Note

The [PAL Driver](#) provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Parameters

in	<i>port</i>	port identifier
in	<i>bits</i>	bits to be XORed on the specified port

Function Class:

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

7.25.3.41 #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

in	<i>port</i>	port identifier
in	<i>mask</i>	group mask
in	<i>offset</i>	group bit offset within the port

Returns

The group logical states.

Function Class:

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

7.25.3.42 #define pal_lld_writegroup(*port*, *mask*, *offset*, *bits*)

Value:

```
do {
    (void)port;
    (void)mask;
    (void)offset;
    (void)bits;
} while (false)
```



Writes a group of bits.

Note

The [PAL Driver](#) provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Parameters

in	<i>port</i>	port identifier
in	<i>mask</i>	group mask
in	<i>offset</i>	group bit offset within the port
in	<i>bits</i>	bits to be written. Values exceeding the group width are masked.

Function Class:

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

7.25.3.43 #define pal_lld_setgroupmode(*port*, *mask*, *offset*, *mode*) _pal_lld_setgroupmode(*port*, *mask* << *offset*, *mode*)

Pads group mode setup.

This function programs a pads group belonging to the same port with the specified mode.

Note

Programming an unknown or unsupported mode is silently ignored.

Parameters

in	<i>port</i>	port identifier
in	<i>mask</i>	group mask
in	<i>offset</i>	group bit offset within the port
in	<i>mode</i>	group mode

Function Class:

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

7.25.3.44 #define pal_lld_readpad(*port*, *pad*) PAL_LOW

Reads a logical state from an I/O pad.

Note

The [PAL Driver](#) provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port

Returns

The logical state.

Return values

<i>PAL_LOW</i>	low logical state.
<i>PAL_HIGH</i>	high logical state.

Function Class:

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

7.25.3.45 #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

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port
in	<i>bit</i>	logical value, the value must be PAL_LOW or PAL_HIGH

Function Class:

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

7.25.3.46 #define pal_lld_setpad(*port*, *pad*)

Value:

```
do {
    (void)port;
    (void)pad;
} while (false)
```

Sets a pad logical state to PAL_HIGH.

Note

The [PAL Driver](#) provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port

Function Class:

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

7.25.3.47 #define pal_lld_clearpad(*port*, *pad*)

Value:

```
do {
    (void)port;
    (void)pad;
} while (false)
```

Clears a pad logical state to PAL_LOW.

Note

The [PAL Driver](#) provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port

Function Class:

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

7.25.3.48 #define pal_lld_togglepad(*port*, *pad*)

Value:

```
do {
    (void)port;
    (void)pad;
} while (false)
```

Toggles a pad logical state.

Note

The [PAL Driver](#) provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port

Function Class:

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

7.25.3.49 #define pal_lld_setpadmode(*port*, *pad*, *mode*)

Value:

```
do {
    (void)port;
    (void)pad;
    (void)mode;
} while (false)
```

Pad mode setup.

This function programs a pad with the specified mode.

Note

The [PAL Driver](#) provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Programming an unknown or unsupported mode is silently ignored.

Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port
in	<i>mode</i>	pad mode

Function Class:

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

7.25.4 Typedef Documentation

7.25.4.1 **typedef uint32_t ioportmask_t**

Digital I/O port sized unsigned type.

7.25.4.2 `typedef uint32_t iomode_t`

Digital I/O modes.

7.25.4.3 `typedef uint32_t ioportid_t`

Port Identifier.

This type can be a scalar or some kind of pointer, do not make any assumption about it, use the provided macros when populating variables of this type.

7.25.5 Function Documentation

7.25.5.1 `ioportmask_t palReadBus (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

<code>in</code>	<code>bus</code>	the I/O bus, pointer to a <code>IOBus</code> structure
-----------------	------------------	--

Returns

The bus logical states.

Function Class:

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

7.25.5.2 `void palWriteBus (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

<code>in</code>	<code>bus</code>	the I/O bus, pointer to a <code>IOBus</code> structure
<code>in</code>	<code>bits</code>	the bits to be written on the I/O bus. Values exceeding the bus width are masked so most significant bits are lost.

Function Class:

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

7.25.5.3 void palSetBusMode (**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

in	<i>bus</i>	the I/O bus, pointer to a IOBus structure
in	<i>mode</i>	the mode

Function Class:

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

7.25.5.4 void _pal_lid_init (const **PALConfig** * *config*)

STM32 I/O ports configuration.

Ports A-D(E, F, G, H) clocks enabled.

Parameters

in	<i>config</i>	the STM32 ports configuration
----	---------------	-------------------------------

Function Class:

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

7.25.5.5 void _pal_lid_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

in	<i>port</i>	the port identifier
in	<i>mask</i>	the group mask
in	<i>mode</i>	the mode

Function Class:

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

7.26 PWM Driver

Generic PWM Driver.

7.26.1 Detailed Description

Generic PWM Driver.

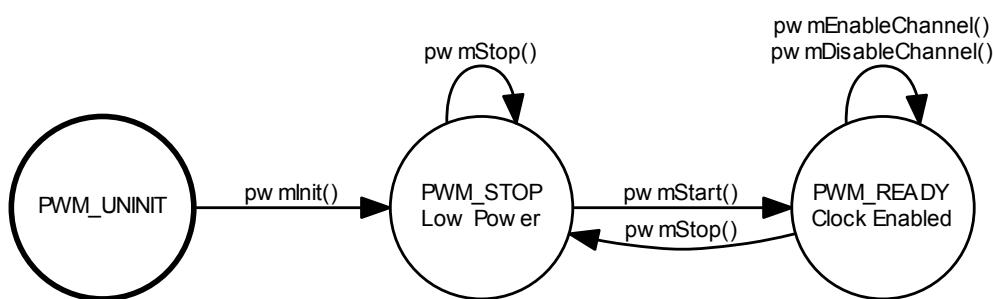
This module implements a generic PWM (Pulse Width Modulation) driver.

Precondition

In order to use the PWM driver the `HAL_USE_PWM` option must be enabled in `halconf.h`.

7.26.2 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).



7.26.3 PWM Operations.

This driver abstracts a generic PWM timer composed of:

- A clock prescaler.
- A main up counter.
- A comparator register that resets the main counter to zero when the limit is reached. An optional callback can be generated when this happens.
- An array of `PWM_CHANNELS` PWM channels, each channel has an output, a comparator and is able to invoke an optional callback when a comparator match with the main counter happens.

A PWM channel output can be in two different states:

- **IDLE**, when the channel is disabled or after a match occurred.
- **ACTIVE**, when the channel is enabled and a match didn't occur yet in the current PWM cycle.

Note that the two states can be associated to both logical zero or one in the `PWMChannelConfig` structure.

Macros

- `#define PWM_CHANNELS 4`
Number of PWM channels per PWM driver.
- `#define pwm_lld_change_period(pwmp, period)`
Changes the period the PWM peripheral.

PWM output mode macros

- `#define PWM_OUTPUT_MASK 0x0FU`
Standard output modes mask.
- `#define PWM_OUTPUT_DISABLED 0x00U`
Output not driven, callback only.
- `#define PWM_OUTPUT_ACTIVE_HIGH 0x01U`
Positive PWM logic, active is logic level one.
- `#define PWM_OUTPUT_ACTIVE_LOW 0x02U`
Inverse PWM logic, active is logic level zero.

PWM duty cycle conversion

- `#define PWM_FRACTION_TO_WIDTH(pwmp, denominator, numerator)`
Converts from fraction to pulse width.
- `#define PWM_DEGREES_TO_WIDTH(pwmp, degrees) PWM_FRACTION_TO_WIDTH(pwmp, 36000, degrees)`
Converts from degrees to pulse width.
- `#define PWM_PERCENTAGE_TO_WIDTH(pwmp, percentage) PWM_FRACTION_TO_WIDTH(pwmp, 10000, percentage)`
Converts from percentage to pulse width.

Macro Functions

- `#define pwmChangePeriodI(pwmp, value)`
Changes the period the PWM peripheral.
- `#define pwmEnableChannelI(pwmp, channel, width)`
Enables a PWM channel.
- `#define pwmDisableChannelI(pwmp, channel)`
Disables a PWM channel.
- `#define pwmlsChannelEnabledI(pwmp, channel) (((pwmp)->enabled & ((pwmchnmsk_t)1U << (pwmchnmsk_t)(channel))) != 0U)`
Returns a PWM channel status.
- `#define pwmEnablePeriodicNotificationI(pwmp) pwm_lld_enable_periodic_notification(pwmp)`
Enables the periodic activation edge notification.
- `#define pwmDisablePeriodicNotificationI(pwmp) pwm_lld_disable_periodic_notification(pwmp)`
Disables the periodic activation edge notification.
- `#define pwmEnableChannelNotificationI(pwmp, channel) pwm_lld_enable_channel_notification(pwmp, channel)`
Enables a channel de-activation edge notification.
- `#define pwmDisableChannelNotificationI(pwmp, channel) pwm_lld_disable_channel_notification(pwmp, channel)`
Disables a channel de-activation edge notification.

PLATFORM configuration options

- `#define PLATFORM_PWM_USE_PWM1 FALSE`
PWMD1 driver enable switch.

Typedefs

- `typedef struct PWMDriver PWMDriver`
Type of a structure representing a PWM driver.
- `typedef void(* pwmcallback_t) (PWMDriver *pwmp)`
Type of a PWM notification callback.
- `typedef uint32_t pwmmodemode_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 pwcnt_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, pwcnt_t period)`
Changes the period the PWM peripheral.
- `void pwmEnableChannel (PWMDriver *pwmp, pwmchannel_t channel, pwcnt_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)`

- **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, pwcmt_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.

7.26.4 Macro Definition Documentation

7.26.4.1 #define PWM_OUTPUT_MASK 0x0FU

Standard output modes mask.

7.26.4.2 #define PWM_OUTPUT_DISABLED 0x00U

Output not driven, callback only.

7.26.4.3 #define PWM_OUTPUT_ACTIVE_HIGH 0x01U

Positive PWM logic, active is logic level one.

7.26.4.4 #define PWM_OUTPUT_ACTIVE_LOW 0x02U

Inverse PWM logic, active is logic level zero.

7.26.4.5 #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

in	<i>pwmp</i>	pointer to a <code>PWMDriver</code> object
in	<i>denominator</i>	denominator of the fraction
in	<i>numerator</i>	numerator of the fraction

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.

7.26.4.6 #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

in	<i>pwmp</i>	pointer to a <code>PWMDriver</code> object
in	<i>degrees</i>	degrees as an integer between 0 and 36000

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.

7.26.4.7 #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	<i>pwmp</i>	pointer to a PWMDriver object
in	<i>percentage</i>	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.

7.26.4.8 #define pwmChangePeriod(*pwmp*, *value*)**Value:**

```
{
    (pwmp) ->period = (value);
    pwm\_lld\_change\_period(pwmp, value);
}
```

Changes the period the PWM peripheral.

This function changes the period of a PWM unit that has already been activated using [pwmStart\(\)](#).

Precondition

The PWM unit must have been activated using [pwmStart\(\)](#).

Postcondition

The PWM unit period is changed to the new value.

Note

If a period is specified that is shorter than the pulse width programmed in one of the channels then the behavior is not guaranteed.

Parameters

in	<i>pwmp</i>	pointer to a PWMDriver object
in	<i>value</i>	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.

7.26.4.9 #define pwmEnableChannel(*pwmp*, *channel*, *width*)**Value:**

```
do {
    (pwmp) ->enabled |= ((pwmchnmsk\_t)1U << (pwmchnmsk\_t) (channel));
    pwm\_lld\_enable\_channel(pwmp, channel, width);
} while (false)
```

Enables a PWM channel.

Precondition

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

Postcondition

The channel is active using the specified configuration.

Note

Depending on the hardware implementation this function has effect starting on the next cycle (recommended implementation) or immediately (fallback implementation).

Parameters

in	<i>pwmp</i>	pointer to a <code>PWMDriver</code> object
in	<i>channel</i>	PWM channel identifier (0...channels-1)
in	<i>width</i>	PWM pulse width as clock pulses number

Function Class:

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

7.26.4.10 #define pwmDisableChannel(*pwmp*, *channel*)

Value:

```
do {
    (pwmp) ->enabled &= ~((pwmchnmsk_t)1U << (pwmchnmsk_t)(channel));
    \pwm_lld_disable_channel(pwmp, channel);
} while (false) \
```

Disables a PWM channel.

Precondition

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

Postcondition

The channel is disabled and its output line returned to the idle state.

Note

Depending on the hardware implementation this function has effect starting on the next cycle (recommended implementation) or immediately (fallback implementation).

Parameters

in	<i>pwmp</i>	pointer to a <code>PWMDriver</code> object
in	<i>channel</i>	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.

```
7.26.4.11 #define pwmIsChannelEnabled( pwmp, channel ) (((pwmp)>enabled & ((pwmchnmsk_t)1U << (pwmchnmsk_t)(channel))) != 0U)
```

Returns a PWM channel status.

Precondition

The PWM unit must have been activated using [pwmStart \(\)](#).

Parameters

in	<i>pwmp</i>	pointer to a PWMDriver object
in	<i>channel</i>	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.

```
7.26.4.12 #define pwmEnablePeriodicNotification( pwmp ) pwm_lld_enable_periodic_notification(pwmp)
```

Enables the periodic activation edge notification.

Precondition

The PWM unit must have been activated using [pwmStart \(\)](#).

Note

If the notification is already enabled then the call has no effect.

Parameters

in	<i>pwmp</i>	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.

```
7.26.4.13 #define pwmDisablePeriodicNotification( pwmp ) pwm_lld_disable_periodic_notification(pwmp)
```

Disables the periodic activation edge notification.

Precondition

The PWM unit must have been activated using [pwmStart \(\)](#).

Note

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

Parameters

in	<i>pwmp</i>	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.

7.26.4.14 #define pwmEnableChannelNotification(*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	<i>pwmp</i>	pointer to a PWMDriver object
in	<i>channel</i>	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.

7.26.4.15 #define pwmDisableChannelNotification(*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	<i>pwmp</i>	pointer to a PWMDriver object
in	<i>channel</i>	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.

7.26.4.16 #define PWM_CHANNELS 4

Number of PWM channels per PWM driver.

7.26.4.17 #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.

7.26.4.18 #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

in	<i>pwmp</i>	pointer to a PWMDriver object
in	<i>period</i>	new cycle time in ticks

Function Class:

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

7.26.5 Typedef Documentation**7.26.5.1 typedef struct PWMDriver PWMDriver**

Type of a structure representing a PWM driver.

7.26.5.2 typedef void(* pwmcallback_t)(PWMDriver *pwmp)

Type of a PWM notification callback.

Parameters

in	<i>pwmp</i>	pointer to a PWMDriver object
----	-------------	---

7.26.5.3 `typedef uint32_t pwmmode_t`

Type of a PWM mode.

7.26.5.4 `typedef uint8_t pwmchannel_t`

Type of a PWM channel.

7.26.5.5 `typedef uint32_t pwmchnmsk_t`

Type of a channels mask.

7.26.5.6 `typedef uint32_t pwcnt_t`

Type of a PWM counter.

7.26.6 Enumeration Type Documentation**7.26.6.1 `enum pwmstate_t`**

Driver state machine possible states.

Enumerator

PWM_UNINIT Not initialized.

PWM_STOP Stopped.

PWM_READY Ready.

7.26.7 Function Documentation**7.26.7.1 `void pwmlInit(void)`**

PWM Driver initialization.

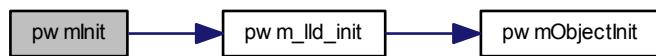
Note

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:

**7.26.7.2 void pwmObjectInit (PWMDriver * pwmp)**

Initializes the standard part of a `PWMDriver` structure.

Parameters

<code>out</code>	<code>pwmp</code>	pointer to a <code>PWMDriver</code> object
------------------	-------------------	--

Function Class:

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

7.26.7.3 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.

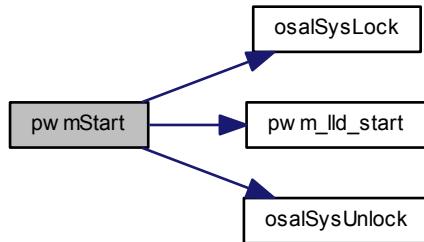
Parameters

<code>in</code>	<code>pwmp</code>	pointer to a <code>PWMDriver</code> object
<code>in</code>	<code>config</code>	pointer to a <code>PWMConfig</code> 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:



7.26.7.4 void pwmStop (PWMDriver * pwmp)

Deactivates the PWM peripheral.

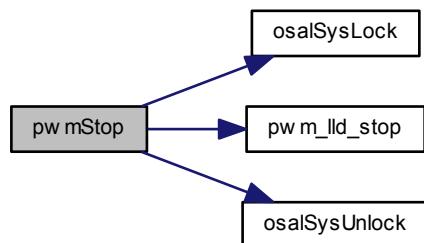
Parameters

in	<i>pwmp</i>	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:



7.26.7.5 void pwmChangePeriod (PWMDriver * pwmp, pwcnt_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.

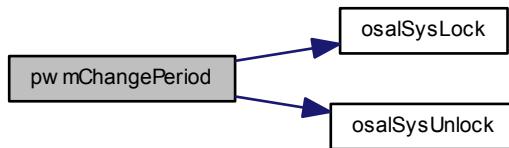
Parameters

in	<i>pwmp</i>	pointer to a <code>PWMDriver</code> object
in	<i>period</i>	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:

**7.26.7.6 void pwmEnableChannel (PWMDriver * *pwmp*, pwmchannel_t *channel*, pwcnt_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).

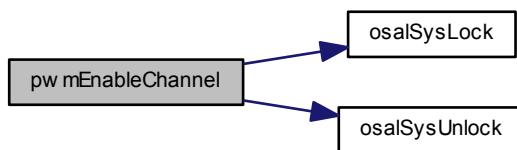
Parameters

in	<i>pwmp</i>	pointer to a PWMDriver object
in	<i>channel</i>	PWM channel identifier (0...channels-1)
in	<i>width</i>	PWM pulse width as clock pulses number

Function Class:

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

Here is the call graph for this function:



7.26.7.7 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).

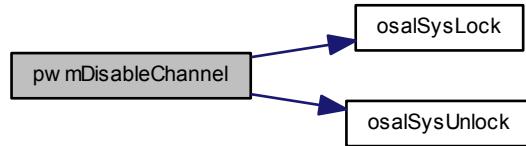
Parameters

in	<i>pwmp</i>	pointer to a PWMDriver object
in	<i>channel</i>	PWM channel identifier (0...channels-1)

Function Class:

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

Here is the call graph for this function:



7.26.7.8 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.

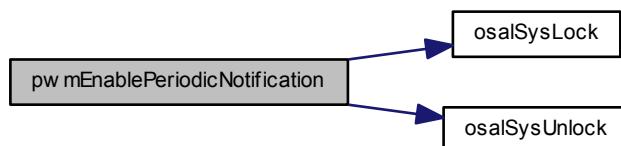
Parameters

in	<i>pwmp</i>	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:



7.26.7.9 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.

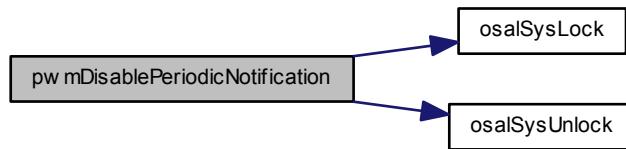
Parameters

in	<i>pwmp</i>	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:

**7.26.7.10 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.

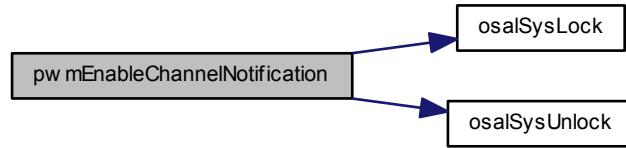
Parameters

in	<i>pwmp</i>	pointer to a PWMDriver object
in	<i>channel</i>	PWM channel identifier (0...channels-1)

Function Class:

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

Here is the call graph for this function:



7.26.7.11 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.

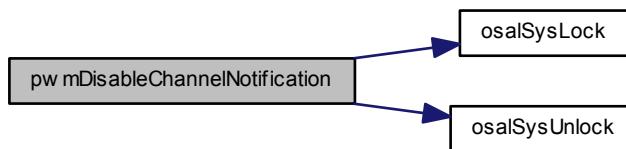
Parameters

in	<code>pwmp</code>	pointer to a <code>PWMDriver</code> object
in	<code>channel</code>	PWM channel identifier (0...channels-1)

Function Class:

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

Here is the call graph for this function:



7.26.7.12 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:

**7.26.7.13 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	<code>pwmp</code>	pointer to a <code>PWMDriver</code> object
----	-------------------	--

Function Class:

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

7.26.7.14 void pwm_lld_stop (PWMDriver * pwmp)

Deactivates the PWM peripheral.

Parameters

in	<code>pwmp</code>	pointer to a <code>PWMDriver</code> object
----	-------------------	--

Function Class:

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

7.26.7.15 void pwm_lld_enable_channel (PWMDriver * pwmp, pwmchannel_t channel, pwcnt_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

in	<i>pwmp</i>	pointer to a PWMDriver object
in	<i>channel</i>	PWM channel identifier (0...channels-1)
in	<i>width</i>	PWM pulse width as clock pulses number

Function Class:

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

7.26.7.16 void pwm_ll_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

in	<i>pwmp</i>	pointer to a PWMDriver object
in	<i>channel</i>	PWM channel identifier (0...channels-1)

Function Class:

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

7.26.7.17 void pwm_ll_enable_periodic_notification (PWMDriver * *pwmp*)

Enables the periodic activation edge notification.

Precondition

The PWM unit must have been activated using [pwmStart \(\)](#).

Note

If the notification is already enabled then the call has no effect.

Parameters

in	<i>pwmp</i>	pointer to a PWMDriver object
----	-------------	---

Function Class:

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

7.26.7.18 void pwm_lld_disable_periodic_notification (**PWM**Driver * *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	<i>pwmp</i>	pointer to a PWMDriver object
----	-------------	---

Function Class:

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

7.26.7.19 void pwm_lld_enable_channel_notification (**PWM**Driver * *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

in	<i>pwmp</i>	pointer to a PWMDriver object
in	<i>channel</i>	PWM channel identifier (0...channels-1)

Function Class:

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

7.26.7.20 void pwm_lld_disable_channel_notification (**PWM**Driver * *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

in	<i>pwmp</i>	pointer to a PWMDriver object
in	<i>channel</i>	PWM channel identifier (0...channels-1)

Function Class:

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

7.26.8 Variable Documentation

7.26.8.1 PWMDriver PWMD1

PWMD1 driver identifier.

Note

The driver PWMD1 allocates the complex timer TIM1 when enabled.

7.27 RTC Driver

Real Time Clock Abstraction Layer.

7.27.1 Detailed Description

Real Time Clock Abstraction Layer.

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 _file_stream_methods`
FileStream 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`

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 uint32_t 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 RTCAlarm`
Type of a structure representing an RTC alarm time stamp.
- `struct RTCDriverVMT`
RTCDriver virtual methods table.
- `struct RTCDriver`
Structure representing an RTC driver.

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 rtcConvertStructTmToDateTm (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)`
Enable access to registers.
- `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.

Variables

- RTCDriver `RTCD1`
RTC driver identifier.

7.27.2 Macro Definition Documentation

7.27.2.1 #define RTC_BASE_YEAR 1980U

Base year of the calendar.

7.27.2.2 #define RTC_SUPPORTS_CALLBACKS TRUE

Callback support int the driver.

7.27.2.3 #define RTC_ALARMS 2

Number of alarms available.

7.27.2.4 #define RTC_HAS_STORAGE FALSE

Presence of a local persistent storage.

7.27.2.5 #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.

7.27.2.6 #define _rtc_driver_methods _file_stream_methods

FileStream specific methods.

7.27.3 Typedef Documentation

7.27.3.1 `typedef struct RTCDriver RTCDriver`

Type of a structure representing an RTC driver.

7.27.3.2 `typedef uint32_t rtcalarm_t`

Type of an RTC alarm number.

7.27.3.3 `typedef void(* rtccb_t) (RTCDriver *rtcp, rtcevent_t event)`

Type of a generic RTC callback.

7.27.4 Enumeration Type Documentation

7.27.4.1 `enum rtcevent_t`

Type of an RTC event.

7.27.5 Function Documentation

7.27.5.1 `void rtcInit(void)`

RTC Driver initialization.

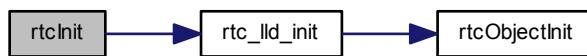
Note

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

Function Class:

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

Here is the call graph for this function:



7.27.5.2 `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	<i>rtcp</i>	pointer to RTC driver structure
-----	-------------	---------------------------------

Function Class:

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

7.27.5.3 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	<i>rtcp</i>	pointer to RTC driver structure
in	<i>timespec</i>	pointer to a RTCDateTime structure

Function Class:

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

Here is the call graph for this function:

**7.27.5.4 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

in	<i>rtcp</i>	pointer to RTC driver structure
----	-------------	---------------------------------

out	<i>timespec</i>	pointer to a RTCDateTime structure
-----	-----------------	--

Function Class:

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

Here is the call graph for this function:

**7.27.5.5 void rtcSetAlarm (RTCDriver * *rtcp*, rtcalarm_t *alarm*, const RTCAlarm * *alarmspec*)**

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

in	<i>rtcp</i>	pointer to RTC driver structure
in	<i>alarm</i>	alarm identifier
in	<i>alarmspec</i>	pointer to a RTCAlarm structure or NULL

Function Class:

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

Here is the call graph for this function:

**7.27.5.6 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

in	<i>rtcp</i>	pointer to RTC driver structure
in	<i>alarm</i>	alarm identifier
out	<i>alarmspec</i>	pointer to a RTCAlarm structure

Function Class:

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

Here is the call graph for this function:

**7.27.5.7 void rtcSetCallback (RTCDriver * *rtcp*, rtccb_t *callback*)**

Enables or disables RTC callbacks.

This function enables or disables the callback, use a `NULL` pointer in order to disable it.

Note

This function can be called from any context but limitations could be imposed by the low level implementation.
 It is guaranteed that the function can be called from thread context.
 The function can be reentrant or not reentrant depending on the low level implementation.

Parameters

in	<i>rtcp</i>	pointer to RTC driver structure
in	<i>callback</i>	callback function pointer or <code>NULL</code>

Function Class:

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

7.27.5.8 void rtcConvertDateTimeToStructTm (const RTCDateTime * *timespec*, struct tm * *tmp*, uint32_t * *tv_msec*)

Convert [RTCDateTime](#) to broken-down time structure.

Parameters

in	<i>timespec</i>	pointer to a RTCDateTime structure
out	<i>tmp</i>	pointer to a broken-down time structure
out	<i>tv_msec</i>	pointer to milliseconds value or NULL

Function Class:

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

7.27.5.9 void rtcConvertStructTmToDateTIme (const struct tm * *tmp*, uint32_t *tv_msec*, RTCDateTime * *timespec*)

Convert broken-down time structure to [RTCDateTime](#).

Parameters

in	<i>tmp</i>	pointer to a broken-down time structure
in	<i>tv_msec</i>	milliseconds value
out	<i>timespec</i>	pointer to a RTCDateTime structure

Function Class:

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

7.27.5.10 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	<i>timespec</i>	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.

7.27.5.11 void rtc_lld_init (void)

Enable access to registers.

Function Class:

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

Here is the call graph for this function:



7.27.5.12 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

in	<i>rtcp</i>	pointer to RTC driver structure
in	<i>timespec</i>	pointer to a RTCDateTime structure

Function Class:

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

7.27.5.13 void rtc_lld_get_time (RTCDriver * *rtcp*, RTCDateTime * *timespec*)

Get current time.

Note

The function can be called from any context.

Parameters

in	<i>rtcp</i>	pointer to RTC driver structure
out	<i>timespec</i>	pointer to a RTCDateTime structure

Function Class:

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

7.27.5.14 void rtc_lld_set_alarm (RTCDriver * *rtcp*, rtcalarm_t *alarm*, const RTCAlarm * *alarmspec*)

Set alarm time.

Note

Default value after BKP domain reset for both comparators is 0.
Function does not performs any checks of alarm time validity.
The function can be called from any context.

Parameters

in	<i>rtcp</i>	pointer to RTC driver structure.
in	<i>alarm</i>	alarm identifier. Can be 1 or 2.
in	<i>alarmspec</i>	pointer to a RTCAalarm structure.

Function Class:

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

7.27.5.15 void rtc_lld_get_alarm (RTCDriver * *rtcp*, rtcalarm_t *alarm*, RTCAalarm * *alarmspec*)

Get alarm time.

Note

The function can be called from any context.

Parameters

in	<i>rtcp</i>	pointer to RTC driver structure
in	<i>alarm</i>	alarm identifier
out	<i>alarmspec</i>	pointer to a RTCAalarm structure

Function Class:

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

7.27.6 Variable Documentation**7.27.6.1 RTCDriver RTCD1**

RTC driver identifier.

7.28 SDC Driver

Generic SD Card Driver.

7.28.1 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`.

7.28.2 Driver State Machine

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

7.28.3 Driver Operations

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

Macros

- `#define _sdc_driver_methods _mmcsd_block_device_methods`
SDC driver 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.

Macro Functions

- `#define sdclsCardInserted(sdc) (sdc_lld_is_card_inserted(sdc))`
Returns the card insertion status.
- `#define sdclsWriteProtected(sdc) (sdc_lld_is_write_protected(sdc))`
Returns the write protect status.

PLATFORM configuration options

- `#define PLATFORM_SDC_USE_SDC1 FALSE`
PWMD1 driver enable switch.

Typedefs

- `typedef uint32_t sdcmode_t`
Type of card flags.
- `typedef uint32_t sdcflags_t`
SDC Driver condition flags type.
- `typedef struct SDCDriver SDCDriver`
Type of a structure representing an SDC driver.

Data Structures

- `struct SDCCConfig`
Driver configuration structure.
- `struct SDCDriverVMT`
SDCDriver virtual methods table.
- `struct SDCDriver`
Structure representing an SDC driver.

Functions

- `static bool mode_detect (SDCDriver *sdc)`
Detects card mode.
- `static bool mmc_init (SDCDriver *sdc)`
Init procedure for MMC.
- `static bool sdc_init (SDCDriver *sdc)`
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)`

- static uint16_t **sdc_cmd6_extract_info** (sd_switch_function_t function, const uint8_t *buf)

Extracts information from CMD6 answer.
- static bool **sdc_cmd6_check_status** (sd_switch_function_t function, const uint8_t *buf)

Checks status after switching using CMD6.
- static bool **sdc_detect_bus_clk** (SDCDriver *sdcp, sdcbusclk_t *clk)

Reads supported bus clock and switch SDC to appropriate mode.
- static bool **mmc_detect_bus_clk** (SDCDriver *sdcp, sdcbusclk_t *clk)

Reads supported bus clock and switch MMC to appropriate mode.
- static bool **detect_bus_clk** (SDCDriver *sdcp, sdcbusclk_t *clk)

Reads supported bus clock and switch card to appropriate mode.
- static bool **sdc_set_bus_width** (SDCDriver *sdcp)

Sets bus width for SDC.
- static bool **mmc_set_bus_width** (SDCDriver *sdcp)

Sets bus width for MMC.
- bool **_sdc_wait_for_transfer_state** (SDCDriver *sdcp)

Wait for the card to complete pending operations.
- void **sdcInit** (void)

SDC Driver initialization.
- void **sdcObjectInit** (SDCDriver *sdcp)

*Initializes the standard part of a *SDCDriver* structure.*
- void **sdcStart** (SDCDriver *sdcp, const SDCCConfig *config)

Configures and activates the SDC peripheral.
- void **sdcStop** (SDCDriver *sdcp)

Deactivates the SDC peripheral.
- bool **sdcConnect** (SDCDriver *sdcp)

Performs the initialization procedure on the inserted card.
- bool **sdcDisconnect** (SDCDriver *sdcp)

Brings the driver in a state safe for card removal.
- bool **sdcRead** (SDCDriver *sdcp, uint32_t startblk, uint8_t *buf, uint32_t n)

Reads one or more blocks.
- bool **sdcWrite** (SDCDriver *sdcp, uint32_t startblk, const uint8_t *buf, uint32_t n)

Writes one or more blocks.
- **sdcflags_t sdcGetAndClearErrors** (SDCDriver *sdcp)

Returns the errors mask associated to the previous operation.
- bool **sdcSync** (SDCDriver *sdcp)

Waits for card idle condition.
- bool **sdcGetInfo** (SDCDriver *sdcp, BlockDeviceInfo *bdip)

Returns the media info.
- bool **sdcErase** (SDCDriver *sdcp, uint32_t startblk, uint32_t endblk)

Erases the supplied blocks.
- 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.

7.28.4 Macro Definition Documentation

7.28.4.1 #define SDC_INIT_RETRY 100

Number of initialization attempts before rejecting the card.

Note

Attempts are performed at 10mS intervals.

7.28.4.2 #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.

7.28.4.3 #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.

7.28.4.4 #define sdclIsCardInserted(*sdcp*) (sdc_lld_is_card_inserted(*sdcp*))

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	<i>sdcp</i>	pointer to the <code>SDCDriver</code> object
----	-------------	--

Returns

The card state.

Return values

<i>FALSE</i>	card not inserted.
<i>TRUE</i>	card inserted.

Function Class:

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

7.28.4.5 #define sdclIsWriteProtected(*sdcp*) (sdc_lld_is_write_protected(*sdcp*))

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	<code>sdcp</code>	pointer to the <code>SDCDriver</code> object
----	-------------------	--

Returns

The card state.

Return values

<i>FALSE</i>	not write protected.
<i>TRUE</i>	write protected.

Function Class:

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

7.28.4.6 #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.

7.28.4.7 #define _sdc_driver_methods _mmcblk_driver_methods

`SDCDriver` specific methods.

7.28.5 Typedef Documentation**7.28.5.1 typedef uint32_t sdcmode_t**

Type of card flags.

7.28.5.2 typedef uint32_t sdcflags_t

SDC Driver condition flags type.

7.28.5.3 typedef struct SDCDriver SDCDriver

Type of a structure representing an SDC driver.

7.28.6 Enumeration Type Documentation**7.28.6.1 enum mmc_switch_t**

MMC switch mode.

7.28.6.2 enum sd_switch_t

SDC switch mode.

7.28.6.3 enum sd_switch_function_t

SDC switch function.

7.28.6.4 enum sdcbusmode_t

Type of SDIO bus mode.

7.28.6.5 enum sdcbusclk_t

Max supported clock.

7.28.7 Function Documentation

7.28.7.1 static bool mode_detect (SDCDriver * *sdcp*) [static]

Detects card mode.

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
----	-------------	---

Returns

The operation status.

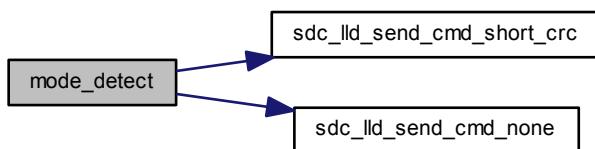
Return values

HAL_SUCCESS	operation succeeded.
HAL_FAILED	operation failed.

Function Class:

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

Here is the call graph for this function:



7.28.7.2 static bool mmc_init (SDCDriver * *sdcp*) [static]

Init procedure for MMC.

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
----	-------------	---

Returns

The operation status.

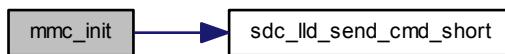
Return values

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

Function Class:

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

Here is the call graph for this function:

**7.28.7.3 static bool sdc_init(SDCHal *sdc) [static]**

Init procedure for SDC.

Parameters

in	<i>sdc</i>	pointer to the SDCHal object
----	------------	--

Returns

The operation status.

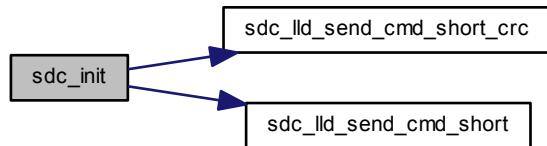
Return values

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

Function Class:

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

Here is the call graph for this function:



7.28.7.4 static uint32_t mmc_cmd6_construct (mmc_switch_t access, uint32_t idx, uint32_t value, uint32_t cmd_set) [static]

Constructs CMD6 argument for MMC.

Parameters

in	<i>access</i>	EXT_CSD access mode
in	<i>idx</i>	EXT_CSD byte number
in	<i>value</i>	value to be written in target field
in	<i>cmd_set</i>	switch current command set

Returns

CMD6 argument.

Function Class:

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

7.28.7.5 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

in	<i>mode</i>	switch/test mode
in	<i>function</i>	function number to be switched
in	<i>value</i>	value to be written in target function

Returns

CMD6 argument.

Function Class:

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

7.28.7.6 static uint16_t sdc_cmd6_extract_info (*sd_switch_function_t function*, *const uint8_t * buf*) [static]

Extracts information from CMD6 answer.

Parameters

in	<i>function</i>	function number to be switched
in	<i>buf</i>	buffer with answer

Returns

extracted answer.

Function Class:

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

7.28.7.7 static bool sdc_cmd6_check_status (*sd_switch_function_t function*, *const uint8_t * buf*) [static]

Checks status after switching using CMD6.

Parameters

in	<i>function</i>	function number to be switched
in	<i>buf</i>	buffer with answer

Returns

The operation status.

Return values

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

Function Class:

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

7.28.7.8 static bool sdc_detect_bus_clk (*SDCDriver * sdcp*, *sdcbusclk_t * clk*) [static]

Reads supported bus clock and switch SDC to appropriate mode.

Parameters

in	<i>sdcp</i>	pointer to the <code>SDCDriver</code> object
out	<i>clk</i>	pointer to clock enum

Returns

The operation status.

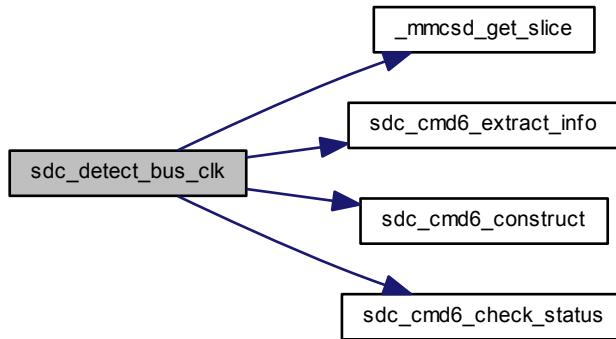
Return values

<code>HAL_SUCCESS</code>	operation succeeded.
<code>HAL_FAILED</code>	operation failed.

Function Class:

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

Here is the call graph for this function:

**7.28.7.9 static bool mmc_detect_bus_clk(`SDCDriver` * *sdcp*, `sdcbusclk_t` * *clk*) [static]**

Reads supported bus clock and switch MMC to appropriate mode.

Parameters

in	<i>sdcp</i>	pointer to the <code>SDCDriver</code> object
out	<i>clk</i>	pointer to clock enum

Returns

The operation status.

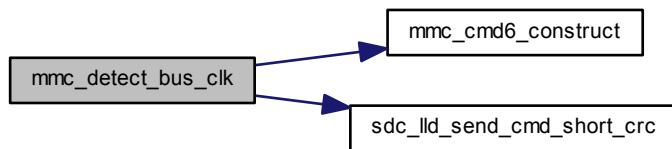
Return values

<code>HAL_SUCCESS</code>	operation succeeded.
<code>HAL_FAILED</code>	operation failed.

Function Class:

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

Here is the call graph for this function:



7.28.7.10 static bool detect_bus_clk (`SDCDriver` * `sdcpc`, `sdcbusclk_t` * `clk`) [static]

Reads supported bus clock and switch card to appropriate mode.

Parameters

in	<code>sdcpc</code>	pointer to the <code>SDCDriver</code> object
out	<code>clk</code>	pointer to clock enum

Returns

The operation status.

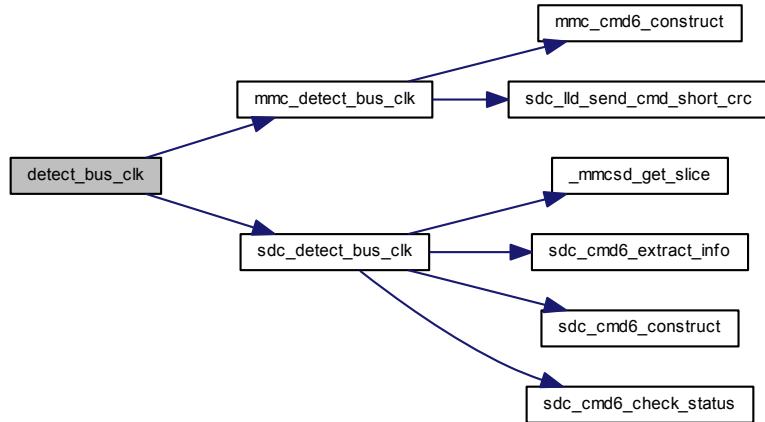
Return values

<code>HAL_SUCCESS</code>	operation succeeded.
<code>HAL_FAILED</code>	operation failed.

Function Class:

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

Here is the call graph for this function:



7.28.7.11 static bool sdc_set_bus_width (**SDCDriver** * *sdc*) [static]

Sets bus width for SDC.

Parameters

in	<i>sdc</i>	pointer to the SDCDriver object
----	------------	--

Returns

The operation status.

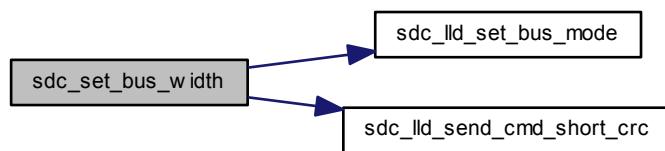
Return values

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

Function Class:

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

Here is the call graph for this function:



7.28.7.12 static bool mmc_set_bus_width(**SDCDriver * *sdcp*) [static]**

Sets bus width for MMC.

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
----	-------------	---

Returns

The operation status.

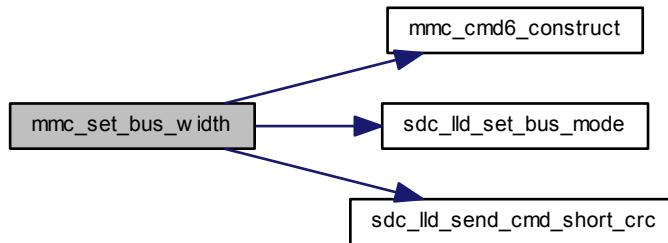
Return values

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

Function Class:

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

Here is the call graph for this function:

**7.28.7.13 bool _sdc_wait_for_transfer_state ([SDCDriver](#) * *sdc*)**

Wait for the card to complete pending operations.

Parameters

in	<i>sdc</i>	pointer to the SDCDriver object
----	------------	---

Returns

The operation status.

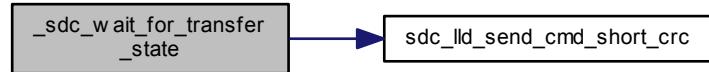
Return values

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

Function Class:

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

Here is the call graph for this function:



7.28.7.14 void sdcInit(void)

SDC Driver initialization.

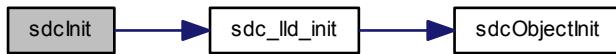
Note

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:



7.28.7.15 void sdcObjectInit(SDCDriver * sdc)

Initializes the standard part of a [SDCDriver](#) structure.

Parameters

out	<i>sdc</i>	pointer to the SDCDriver object
-----	------------	---

Function Class:

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

7.28.7.16 void sdcStart(SDCDriver * sdc, const SDCCConfig * config)

Configures and activates the SDC peripheral.

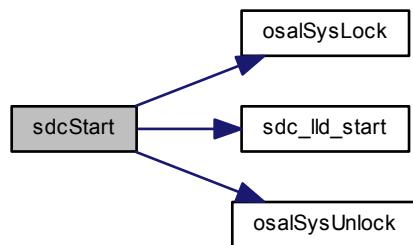
Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
in	<i>config</i>	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:

**7.28.7.17 void sdcStop (SDCDriver * *sdcp*)**

Deactivates the SDC peripheral.

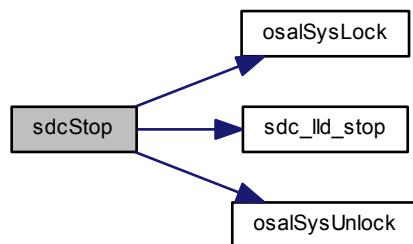
Parameters

in	<i>sdcp</i>	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:



7.28.7.18 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	<i>sdcp</i>	pointer to the <i>SDCDriver</i> object
----	-------------	--

Returns

The operation status.

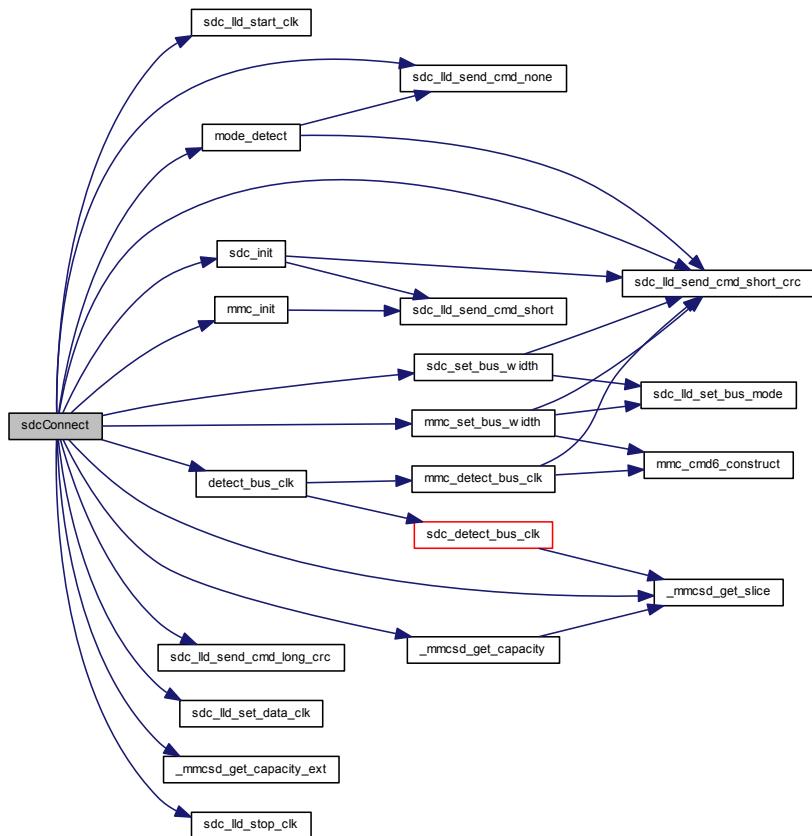
Return values

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	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:



7.28.7.19 bool sdcDisconnect (**SDCDriver * *sdcp*)**

Brings the driver in a state safe for card removal.

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
----	-------------	---

Returns

The operation status.

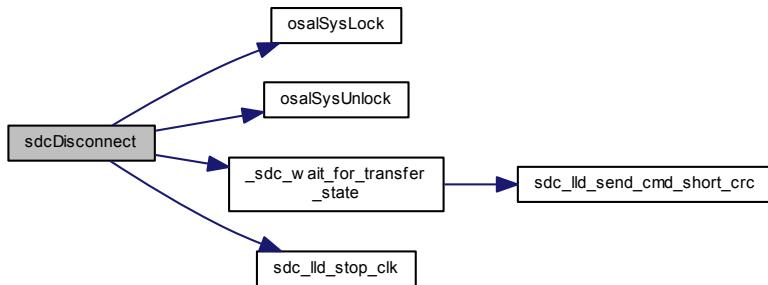
Return values

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	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:

**7.28.7.20 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

in	<i>sdcp</i>	pointer to the SDCDriver object
in	<i>startblk</i>	first block to read
out	<i>buf</i>	pointer to the read buffer
in	<i>n</i>	number of blocks to read

Returns

The operation status.

Return values

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	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:



7.28.7.21 bool sdcWrite (SDCDriver * *sdc*, 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

in	<i>sdc</i>	pointer to the <code>SDCDriver</code> object
in	<i>startblk</i>	first block to write
out	<i>buf</i>	pointer to the write buffer
in	<i>n</i>	number of blocks to write

Returns

The operation status.

Return values

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	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:



7.28.7.22 `sdcflags_t sdcGetAndClearErrors (SDCDriver * sdc)`

Returns the errors mask associated to the previous operation.

Parameters

in	<code>sdc</code>	pointer to the <code>SDCDriver</code> object
----	------------------	--

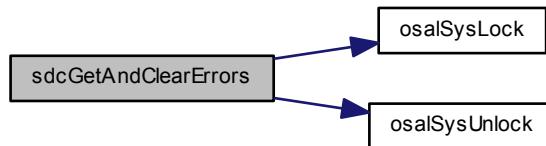
Returns

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:



7.28.7.23 `bool sdcSync (SDCDriver * sdc)`

Waits for card idle condition.

Parameters

in	<code>sdc</code>	pointer to the <code>SDCDriver</code> object
----	------------------	--

Returns

The operation status.

Return values

<code>HAL_SUCCESS</code>	the operation succeeded.
<code>HAL_FAILED</code>	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:



7.28.7.24 bool sdcGetInfo (SDCDriver * *sdcp*, BlockDeviceInfo * *b dip*)

Returns the media info.

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
out	<i>b dip</i>	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.

7.28.7.25 bool sdcErase (SDCDriver * *sdcp*, uint32_t *startblk*, uint32_t *endblk*)

Erases the supplied blocks.

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
in	<i>startblk</i>	starting block number
in	<i>endblk</i>	ending block number

Returns

The operation status.

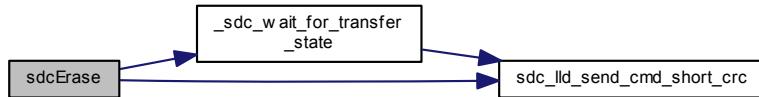
Return values

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:



7.28.7.26 void sdcIldInit(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:



7.28.7.27 void sdcIldStart(SDCDriver * sdcp)

Configures and activates the SDC peripheral.

Parameters

in	<code>sdcp</code>	pointer to the <code>SDCDriver</code> object
----	-------------------	--

Function Class:

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

7.28.7.28 void sdcIldStop(SDCDriver * sdcp)

Deactivates the SDC peripheral.

Parameters

in	<code>sdcp</code>	pointer to the <code>SDCDriver</code> object
----	-------------------	--

Function Class:

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

7.28.7.29 void sdc_lld_start_clk(SDCDriver * sdc)

Starts the SDIO clock and sets it to init mode (400kHz or less).

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
----	-------------	---

Function Class:

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

7.28.7.30 void sdc_llid_set_data_clk (SDCDriver * *sdcp*, sdcbusclk_t *clk*)

Sets the SDIO clock to data mode (25MHz or less).

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
in	<i>clk</i>	the clock mode

Function Class:

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

7.28.7.31 void sdc_llid_stop_clk (SDCDriver * *sdcp*)

Stops the SDIO clock.

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
----	-------------	---

Function Class:

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

7.28.7.32 void sdc_llid_set_bus_mode (SDCDriver * *sdcp*, sdcbusmode_t *mode*)

Switches the bus to 4 bits mode.

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
in	<i>mode</i>	bus mode

Function Class:

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

7.28.7.33 void sdc_llid_send_cmd_none (SDCDriver * *sdcp*, uint8_t *cmd*, uint32_t *arg*)

Sends an SDIO command with no response expected.

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
in	<i>cmd</i>	card command
in	<i>arg</i>	command argument

Function Class:

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

7.28.7.34 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.

Note

The CRC is not verified.

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
in	<i>cmd</i>	card command
in	<i>arg</i>	command argument
out	<i>resp</i>	pointer to the response buffer (one word)

Returns

The operation status.

Return values

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

Function Class:

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

7.28.7.35 bool sdc_lld_send_cmd_short_crc (SDCDriver * *sdcp*, uint8_t *cmd*, uint32_t *arg*, uint32_t * *resp*)

Sends an SDIO command with a short response expected and CRC.

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
in	<i>cmd</i>	card command
in	<i>arg</i>	command argument
out	<i>resp</i>	pointer to the response buffer (one word)

Returns

The operation status.

Return values

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

Function Class:

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

7.28.7.36 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

in	<i>sdcp</i>	pointer to the <code>SDCDriver</code> object
in	<i>cmd</i>	card command
in	<i>arg</i>	command argument
out	<i>resp</i>	pointer to the response buffer (four words)

Returns

The operation status.

Return values

<code>HAL_SUCCESS</code>	operation succeeded.
<code>HAL_FAILED</code>	operation failed.

Function Class:

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

7.28.7.37 bool sdc_lld_read (`SDCDriver` * *sdcp*, `uint32_t` *startblk*, `uint8_t` * *buf*, `uint32_t` *n*)

Reads one or more blocks.

Parameters

in	<i>sdcp</i>	pointer to the <code>SDCDriver</code> object
in	<i>startblk</i>	first block to read
out	<i>buf</i>	pointer to the read buffer
in	<i>n</i>	number of blocks to read

Returns

The operation status.

Return values

<code>HAL_SUCCESS</code>	operation succeeded.
<code>HAL_FAILED</code>	operation failed.

Function Class:

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

7.28.7.38 bool sdc_lld_write (`SDCDriver` * *sdcp*, `uint32_t` *startblk*, `const uint8_t` * *buf*, `uint32_t` *n*)

Writes one or more blocks.

Parameters

in	<i>sdcp</i>	pointer to the <code>SDCDriver</code> object
in	<i>startblk</i>	first block to write
out	<i>buf</i>	pointer to the write buffer
in	<i>n</i>	number of blocks to write

Returns

The operation status.

Return values

<i>HAL_SUCCESS</i>	operation succeeded.
<i>HAL_FAILED</i>	operation failed.

Function Class:

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

7.28.7.39 bool sdc_lld_sync (SDCDriver * sdc)

Waits for card idle condition.

Parameters

in	<i>sdc</i>	pointer to the SDCDriver object
----	------------	---

Returns

The operation status.

Return values

<i>HAL_SUCCESS</i>	the operation succeeded.
<i>HAL_FAILED</i>	the operation failed.

Function Class:

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

7.28.8 Variable Documentation**7.28.8.1 const struct SDCDriverVMT sdc_vmt [static]****Initial value:**

```
= {
    (bool (*)(void *))sdc_lld_is_card_inserted,
    (bool (*)(void *))sdc_lld_is_write_protected,
    (bool (*)(void *))sdcConnect,
    (bool (*)(void *))sdcDisconnect,
    (bool (*)(void *, uint32_t, uint8_t *, uint32_t))sdcRead,
    (bool (*)(void *, uint32_t, const uint8_t *, uint32_t))sdcWrite,
    (bool (*)(void *))sdcSync,
    (bool (*)(void *, BlockDeviceInfo *))sdcGetInfo
}
```

Virtual methods table.

7.28.8.2 SDCDriver SDCD1

SDCD1 driver identifier.

7.29 Serial Driver

Generic Serial Driver.

7.29.1 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.

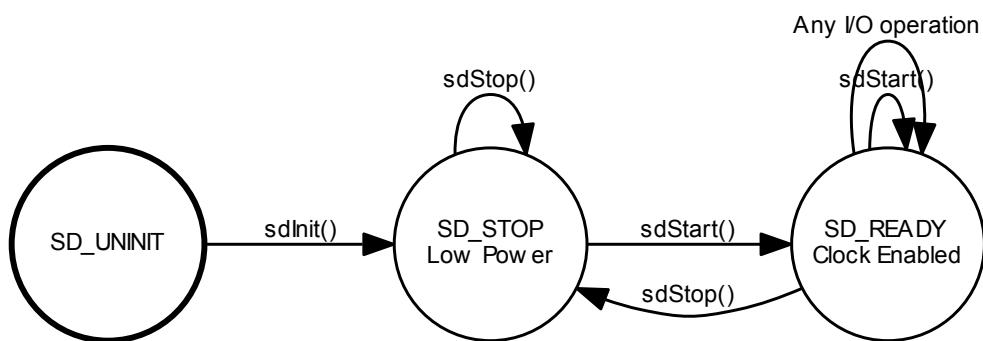
The module also contains functions that make the implementation of the interrupt service routines much easier.

Precondition

In order to use the SERIAL driver the `HAL_USE_SERIAL` option must be enabled in `halconf.h`.

7.29.2 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_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`

- `#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.

Serial configuration options

- `#define SERIAL_DEFAULT_BITRATE 38400`
Default bit rate.
- `#define SERIAL_BUFFERS_SIZE 16`
Serial buffers size.

Macro Functions

- `#define sdPut(sdp, b) oqPut(&(sdp)->oqueue, b)`
Direct write to a `SerialDriver`.
- `#define sdPutTimeout(sdp, b, t) oqPutTimeout(&(sdp)->oqueue, b, t)`
Direct write to a `SerialDriver` with timeout specification.
- `#define sdGet(sdp) iqGet(&(sdp)->iqueue)`
Direct read from a `SerialDriver`.
- `#define sdGetTimeout(sdp, t) iqGetTimeout(&(sdp)->iqueue, t)`
Direct read from a `SerialDriver` with timeout specification.
- `#define sdWrite(sdp, b, n) oqWriteTimeout(&(sdp)->oqueue, b, n, TIME_INFINITE)`
Direct blocking write to a `SerialDriver`.
- `#define sdWriteTimeout(sdp, b, n, t) oqWriteTimeout(&(sdp)->oqueue, b, n, t)`
Direct blocking write to a `SerialDriver` with timeout specification.
- `#define sdAsynchronousWrite(sdp, b, n) oqWriteTimeout(&(sdp)->oqueue, b, n, TIME_IMMEDIATE)`
Direct non-blocking write to a `SerialDriver`.
- `#define sdRead(sdp, b, n) iqReadTimeout(&(sdp)->iqueue, b, n, TIME_INFINITE)`
Direct blocking read from a `SerialDriver`.
- `#define sdReadTimeout(sdp, b, n, t) iqReadTimeout(&(sdp)->iqueue, b, n, t)`
Direct blocking read from a `SerialDriver` with timeout specification.
- `#define sdAsynchronousRead(sdp, b, n) iqReadTimeout(&(sdp)->iqueue, b, n, TIME_IMMEDIATE)`
Direct non-blocking read from a `SerialDriver`.

PLATFORM configuration options

- `#define PLATFORM_SERIAL_USE_USART1 FALSE`
USART1 driver enable switch.

Typedefs

- `typedef struct SerialDriver SerialDriver`
Structure representing a serial driver.

Data Structures

- struct `SerialDriverVMT`
`SerialDriver` virtual methods table.
- struct `SerialDriver`
`Full duplex serial driver class.`
- struct `SerialConfig`
`PLATFORM Serial Driver configuration structure.`

Functions

- void `sdInit` (void)
`Serial Driver initialization.`
- void `sdObjectInit` (`SerialDriver` *`sdp`, `qnotify_t` `inotify`, `qnotify_t` `onotify`)
`Initializes a generic full duplex driver object.`
- void `sdStart` (`SerialDriver` *`sdp`, const `SerialConfig` *`config`)
`Configures and starts the driver.`
- void `sdStop` (`SerialDriver` *`sdp`)
`Stops the driver.`
- void `sdIncomingData` (`SerialDriver` *`sdp`, `uint8_t` `b`)
`Handles incoming data.`
- `msg_t` `sdRequestData` (`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.`
- void `sd_lld_init` (void)
`Low level serial driver initialization.`
- void `sd_lld_start` (`SerialDriver` *`sdp`, const `SerialConfig` *`config`)
`Low level serial driver configuration and (re)start.`
- void `sd_lld_stop` (`SerialDriver` *`sdp`)
`Low level serial driver stop.`

Enumerations

- enum `sdstate_t` { `SD_UNINIT` = 0, `SD_STOP` = 1, `SD_READY` = 2 }
`Driver state machine possible states.`

Variables

- `SerialDriver SD1`
`USART1 serial driver identifier.`
- static const `SerialConfig default_config`
`Driver default configuration.`

7.29.3 Macro Definition Documentation

7.29.3.1 #define SD_PARITY_ERROR (eventflags_t)32

Parity.

7.29.3.2 #define SD_FRAMING_ERROR (eventflags_t)64

Framing.

7.29.3.3 #define SD_OVERRUN_ERROR (eventflags_t)128

Overflow.

7.29.3.4 #define SD_NOISE_ERROR (eventflags_t)256

Line noise.

7.29.3.5 #define SD_BREAK_DETECTED (eventflags_t)512

LIN Break.

7.29.3.6 #define SERIAL_DEFAULT_BITRATE 38400

Default bit rate.

Configuration parameter, this is the baud rate selected for the default configuration.

7.29.3.7 #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.

7.29.3.8 #define _serial_driver_methods _base_asynchronous_channel_methods

[SerialDriver](#) specific methods.

7.29.3.9 #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.

See also

[chnPutTimeout\(\)](#)

Function Class:

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

```
7.29.3.10 #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.

See also

[chnPutTimeout\(\)](#)

Function Class:

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

```
7.29.3.11 #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.

See also

[chnGetTimeout\(\)](#)

Function Class:

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

```
7.29.3.12 #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.

See also

[chnGetTimeout\(\)](#)

Function Class:

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

```
7.29.3.13 #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.

See also

[chnWrite\(\)](#)

Function Class:

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

```
7.29.3.14 #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.

See also

[chnWriteTimeout\(\)](#)

Function Class:

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

```
7.29.3.15 #define sdAsynchronousWrite( sdp, b, n ) oqWriteTimeout(&(sdp)->oqueue, b, n, TIME_IMMEDIATE)
```

Direct non-blocking write to a [SerialDriver](#).

Note

This function bypasses the indirect access to the channel and writes directly to the output queue. This is faster but cannot be used to write to different channels implementations.

See also

[chnWriteTimeout\(\)](#)

Function Class:

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

```
7.29.3.16 #define sdRead( sdp, b, n ) iqReadTimeout(&(sdp)->iqueue, b, n, TIME_INFINITE)
```

Direct blocking read from a [SerialDriver](#).

Note

This function bypasses the indirect access to the channel and reads directly from the input queue. This is faster but cannot be used to read from different channels implementations.

See also

[chnRead\(\)](#)

Function Class:

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

```
7.29.3.17 #define sdReadTimeout( sdp, b, n, t ) iqReadTimeout(&(sdp)->iqueue, 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.

See also

[chnReadTimeout\(\)](#)

Function Class:

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

```
7.29.3.18 #define sdAsynchronousRead( sdp, b, n ) iqReadTimeout(&(sdp)->iqueue, 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.

See also

[chnReadTimeout\(\)](#)

Function Class:

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

```
7.29.3.19 #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.

7.29.3.20 #define _serial_driver_data

Value:

```
_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.

7.29.4 Typedef Documentation

7.29.4.1 typedef struct SerialDriver SerialDriver

Structure representing a serial driver.

7.29.5 Enumeration Type Documentation

7.29.5.1 enum sdstate_t

Driver state machine possible states.

Enumerator

SD_UNINIT Not initialized.

SD_STOP Stopped.

SD_READY Ready.

7.29.6 Function Documentation

7.29.6.1 void sdlInit(void)

Serial Driver initialization.

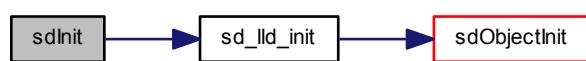
Note

This function is implicitly invoked by [halInit\(\)](#), there is no need to explicitly initialize the driver.

Function Class:

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

Here is the call graph for this function:



7.29.6.2 void sdObjectInit (*SerialDriver* * *sdp*, *qnotify_t* *inotify*, *qnotify_t* *onotify*)

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.

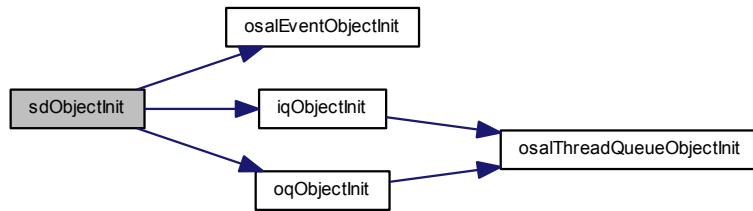
Parameters

out	<i>sdp</i>	pointer to a <i>SerialDriver</i> structure
in	<i>inotify</i>	pointer to a callback function that is invoked when some data is read from the Queue. The value can be NULL.
in	<i>onotify</i>	pointer to a callback function that is invoked when some data is written in the Queue. The value can be NULL.

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:



7.29.6.3 void sdStart (*SerialDriver* * *sdp*, const *SerialConfig* * *config*)

Configures and starts the driver.

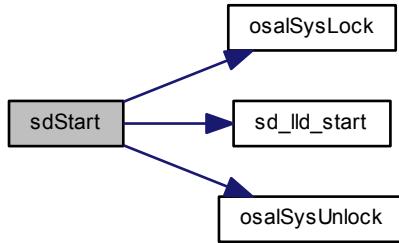
Parameters

in	<i>sdp</i>	pointer to a <i>SerialDriver</i> object
in	<i>config</i>	the architecture-dependent serial driver configuration. If this parameter is set to NULL then a default configuration is used.

Function Class:

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

Here is the call graph for this function:



7.29.6.4 void sdStop (SerialDriver * sdp)

Stops the driver.

Any thread waiting on the driver's queues will be awakened with the message Q_RESET.

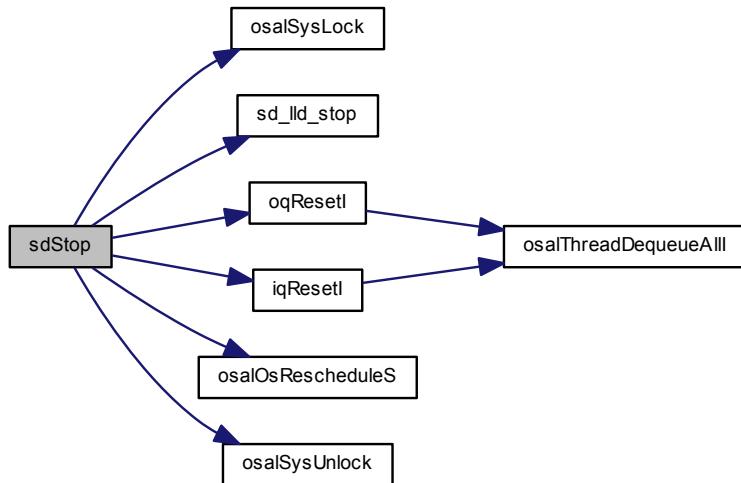
Parameters

in	<code>sdp</code>	pointer to a <code>SerialDriver</code> 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:



7.29.6.5 void sdlIncomingData (*SerialDriver* * *sdp*, *uint8_t* *b*)

Handles incoming data.

This function must be called from the input interrupt service routine in order to enqueue incoming data and generate the related events.

Note

The incoming data event is only generated when the input queue becomes non-empty.

In order to gain some performance it is suggested to not use this function directly but copy this code directly into the interrupt service routine.

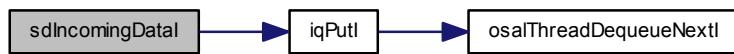
Parameters

in	<i>sdp</i>	pointer to a <i>SerialDriver</i> structure
in	<i>b</i>	the byte to be written in the driver's Input Queue

Function Class:

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

Here is the call graph for this function:



7.29.6.6 msg_t sdrequestData (*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	<i>sdp</i>	pointer to a <i>SerialDriver</i> structure
----	------------	--

Returns

The byte value read from the driver's output queue.

Return values

<code>Q_EMPTY</code>	if the queue is empty (the lower driver usually disables the interrupt source when this happens).
----------------------	---

Function Class:

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

Here is the call graph for this function:



7.29.6.7 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	<code>sdp</code>	pointer to a <code>SerialDriver</code> structure
----	------------------	--

Returns

The queue status.

Return values

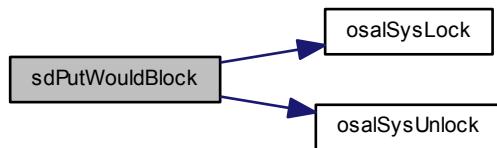
<code>false</code>	if the next write operation would not block.
<code>true</code>	if the next write operation would block.

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:



7.29.6.8 bool sdGetWouldBlock (*SerialDriver* * *sdp*)

Direct input check on a [SerialDriver](#).

Note

This function bypasses the indirect access to the channel and checks directly the input queue. This is faster but cannot be used to check different channels implementations.

Parameters

in	<i>sdp</i>	pointer to a SerialDriver structure
----	------------	---

Returns

The queue status.

Return values

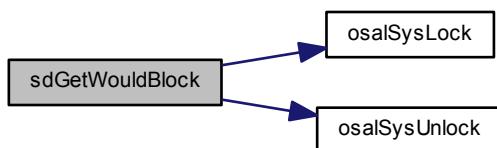
<i>false</i>	if the next write operation would not block.
<i>true</i>	if the next write operation would block.

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:



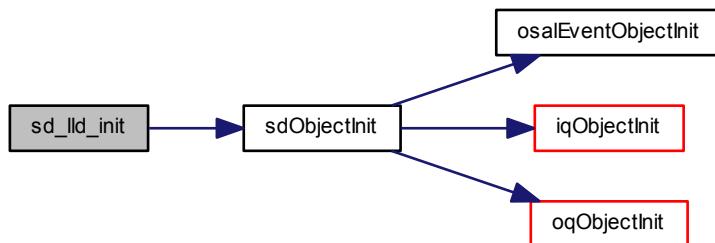
7.29.6.9 void sd_lld_init(void)

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:



7.29.6.10 void sd_lld_start(SerialDriver * sdp, const SerialConfig * config)

Low level serial driver configuration and (re)start.

Parameters

in	<i>sdp</i>	pointer to a <code>SerialDriver</code> object
in	<i>config</i>	the architecture-dependent serial driver configuration. If this parameter is set to <code>NULL</code> then a default configuration is used.

Function Class:

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

7.29.6.11 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	<i>sdp</i>	pointer to a <code>SerialDriver</code> object
----	------------	---

Function Class:

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

7.29.7 Variable Documentation

7.29.7.1 SerialDriver SD1

USART1 serial driver identifier.

7.29.7.2 const SerialConfig default_config [static]**Initial value:**

```
= {  
    38400  
}
```

Driver default configuration.

7.30 Serial over USB Driver

Serial over USB Driver.

7.30.1 Detailed Description

Serial over USB Driver.

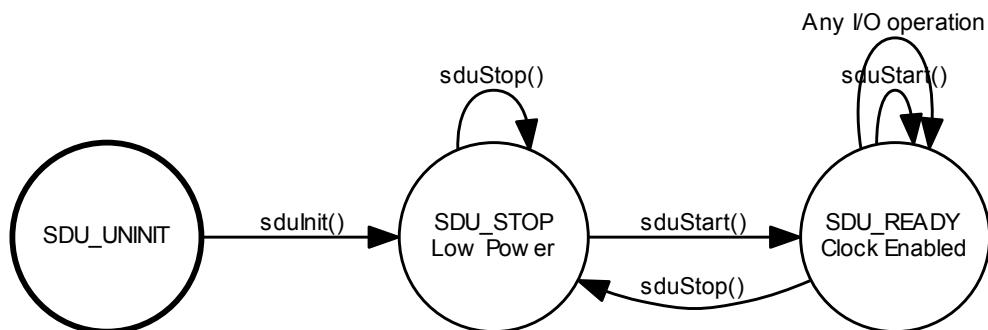
This module implements an USB Communication Device Class (CDC) as a normal serial communication port accessible from the device application.

Precondition

In order to use the USB over Serial driver the `HAL_USE_SERIAL_USB` option must be enabled in `halconf.h`.

7.30.2 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.

CDC specific messages.

- `#define CDC_SEND_ENCAPSULATED_COMMAND 0x00`
- `#define CDC_GET_ENCAPSULATED_RESPONSE 0x01`

- #define **CDC_SET_COMM_FEATURE** 0x02
- #define **CDC_GET_COMM_FEATURE** 0x03
- #define **CDC_CLEAR_COMM_FEATURE** 0x04
- #define **CDC_SET_AUX_LINE_STATE** 0x10
- #define **CDC_SET_HOOK_STATE** 0x11
- #define **CDC_PULSE_SETUP** 0x12
- #define **CDC_SEND_PULSE** 0x13
- #define **CDC_SET_PULSE_TIME** 0x14
- #define **CDC_RING_AUX_JACK** 0x15
- #define **CDC_SET_LINE_CODING** 0x20
- #define **CDC_GET_LINE_CODING** 0x21
- #define **CDC_SET_CONTROL_LINE_STATE** 0x22
- #define **CDC_SEND_BREAK** 0x23
- #define **CDC_SET_RINGER_PARMS** 0x30
- #define **CDC_GET_RINGER_PARMS** 0x31
- #define **CDC_SET_OPERATION_PARMS** 0x32
- #define **CDC_GET_OPERATION_PARMS** 0x33

CDC classes

- #define **CDC_COMMUNICATION_INTERFACE_CLASS** 0x02
- #define **CDC_DATA_INTERFACE_CLASS** 0x0A

CDC subclasses

- #define **CDC_ABSTRACT_CONTROL_MODEL** 0x02

CDC descriptors

- #define **CDC_CS_INTERFACE** 0x24

CDC subdescriptors

- #define **CDC_HEADER** 0x00
- #define **CDC_CALL_MANAGEMENT** 0x01
- #define **CDC_ABSTRACT_CONTROL_MANAGEMENT** 0x02
- #define **CDC_UNION** 0x06

Line Control bit definitions.

- #define **LC_STOP_1** 0
- #define **LC_STOP_1P5** 1
- #define **LC_STOP_2** 2
- #define **LC_PARITY_NONE** 0
- #define **LC_PARITY_ODD** 1
- #define **LC_PARITY_EVEN** 2
- #define **LC_PARITY_MARK** 3
- #define **LC_PARITY_SPACE** 4

SERIAL_USB configuration options

- #define **SERIAL_USB_BUFFERS_SIZE** 256
Serial over USB buffers size.

Typedefs

- `typedef struct SerialUSBDriver SerialUSBDriver`

Structure representing a serial over USB driver.

Data Structures

- `struct cdc_linecoding_t`
Type of Line Coding structure.
- `struct SerialUSBConfig`
Serial over USB Driver configuration structure.
- `struct SerialUSBDriverVMT`
SerialDriver virtual methods table.
- `struct SerialUSBDriver`
Full duplex serial driver class.

Functions

- `static void inotify (io_queue_t *qp)`
Notification of data removed from the input queue.
- `static void onotify (io_queue_t *qp)`
Notification of data inserted into the output 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 sduConfigureHookI (SerialUSBDriver *sdup)`
USB device configured handler.
- `bool sduRequestsHook (USBDriver *usbp)`
Default requests hook.
- `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.

Enumerations

- `enum sdustate_t { SDU_UNINIT = 0, SDU_STOP = 1, SDU_READY = 2 }`
Driver state machine possible states.

7.30.3 Macro Definition Documentation

7.30.3.1 #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.

7.30.3.2 #define _serial_usb_driver_data

Value:

```
_base_asynchronous_channel_data
/* Driver state.*/
sdustate_t           state;
/* Input queue.*/
input_queue_t         iqueue;
/* Output queue.*/
output_queue_t        oqueue;
/* Input buffer.*/
uint8_t               ib[SERIAL_USB_BUFFERS_SIZE];
/* Output buffer.*/
uint8_t               ob[SERIAL_USB_BUFFERS_SIZE];
/* End of the mandatory fields.*/
/* Current configuration data.*/
const SerialUSBConfig *config;
```



`SerialDriver` specific data.

7.30.3.3 #define _serial_usb_driver_methods _base_asynchronous_channel_methods

`SerialUSBDriver` specific methods.

7.30.4 Typedef Documentation

7.30.4.1 typedef struct SerialUSBDriver SerialUSBDriver

Structure representing a serial over USB driver.

7.30.5 Enumeration Type Documentation

7.30.5.1 enum sdustate_t

Driver state machine possible states.

Enumerator

SDU_UNINIT Not initialized.

SDU_STOP Stopped.

SDU_READY Ready.

7.30.6 Function Documentation

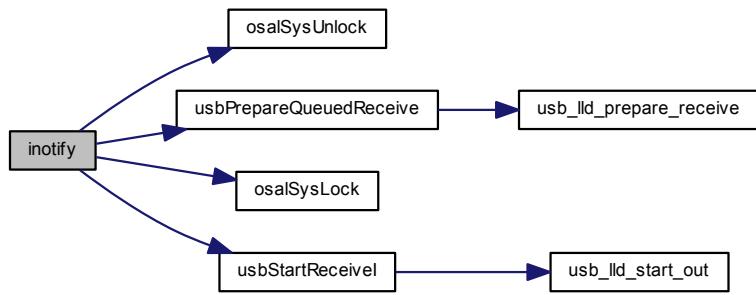
7.30.6.1 static void inotify(io_queue_t *qp) [static]

Notification of data removed from the input queue.

Parameters

in	<i>qp</i>	the queue pointer.
----	-----------	--------------------

Here is the call graph for this function:



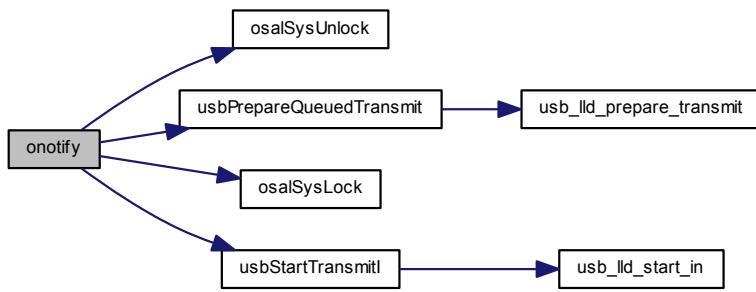
7.30.6.2 static void onnotify(io_queue_t *qp) [static]

Notification of data inserted into the output queue.

Parameters

in	<i>qp</i>	the queue pointer.
----	-----------	--------------------

Here is the call graph for this function:



7.30.6.3 void sdulinit(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.

7.30.6.4 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.

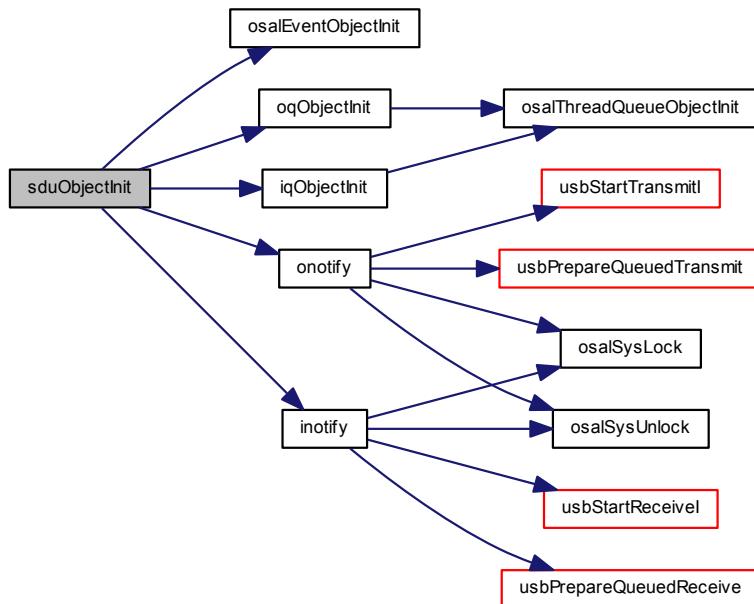
Parameters

<code>out</code>	<code>sdup</code>	pointer to a <code>SerialUSBDriver</code> structure
------------------	-------------------	---

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:

**7.30.6.5 void sduStart (`SerialUSBDriver *sdup`, `const SerialUSBConfig *config`)**

Configures and starts the driver.

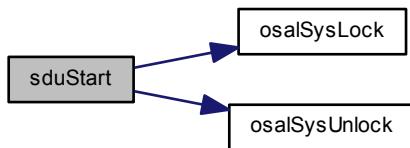
Parameters

in	<i>sduP</i>	pointer to a SerialUSBDriver object
in	<i>config</i>	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:

**7.30.6.6 void sduStop (SerialUSBDriver * sduP)**

Stops the driver.

Any thread waiting on the driver's queues will be awakened with the message Q_RESET.

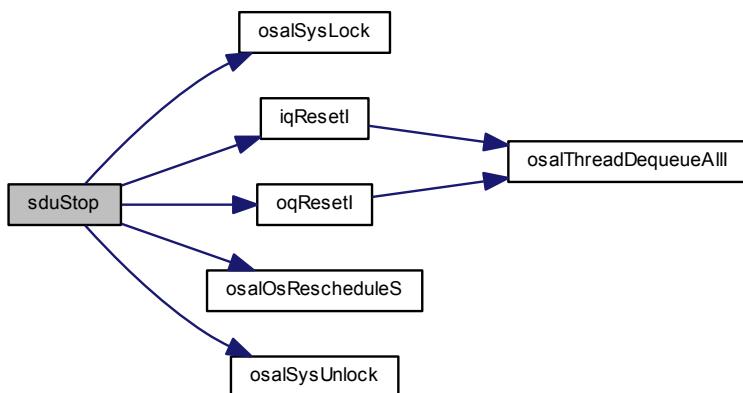
Parameters

in	<i>sduP</i>	pointer to a SerialUSBDriver object
----	-------------	---

Function Class:

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

Here is the call graph for this function:



7.30.6.7 void sduConfigureHook([SerialUSBDriver](#) * *sdup*)

USB device configured handler.

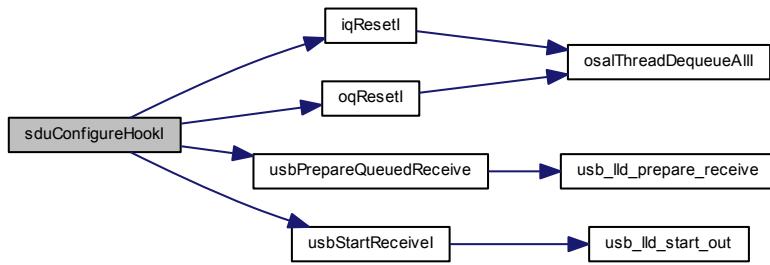
Parameters

in	<i>sdu</i>	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:



7.30.6.8 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	<i>usbp</i>	pointer to the USBDriver object
----	-------------	---

Returns

The hook status.

Return values

<i>true</i>	Message handled internally.
-------------	-----------------------------

<i>false</i>	Message not handled.
--------------	----------------------

7.30.6.9 void sduDataTransmitted (**USBDriver** * *usbp*, **usbep_t** *ep*)

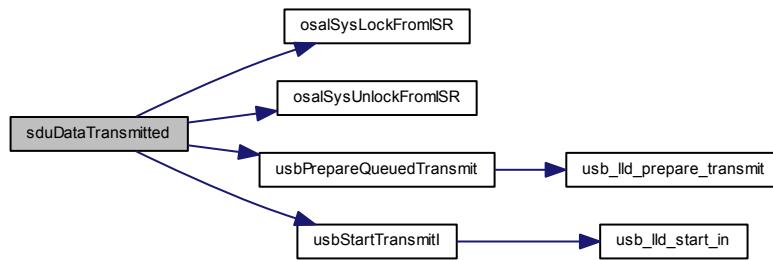
Default data transmitted callback.

The application must use this function as callback for the IN data endpoint.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Here is the call graph for this function:



7.30.6.10 void sduDataReceived (**USBDriver** * *usbp*, **usbep_t** *ep*)

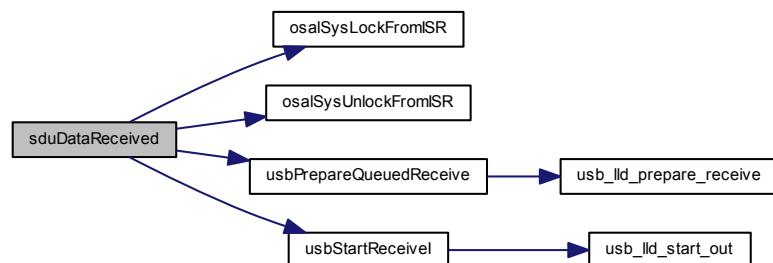
Default data received callback.

The application must use this function as callback for the OUT data endpoint.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Here is the call graph for this function:



7.30.6.11 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

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

7.31 SPI Driver

Generic SPI Driver.

7.31.1 Detailed Description

Generic SPI Driver.

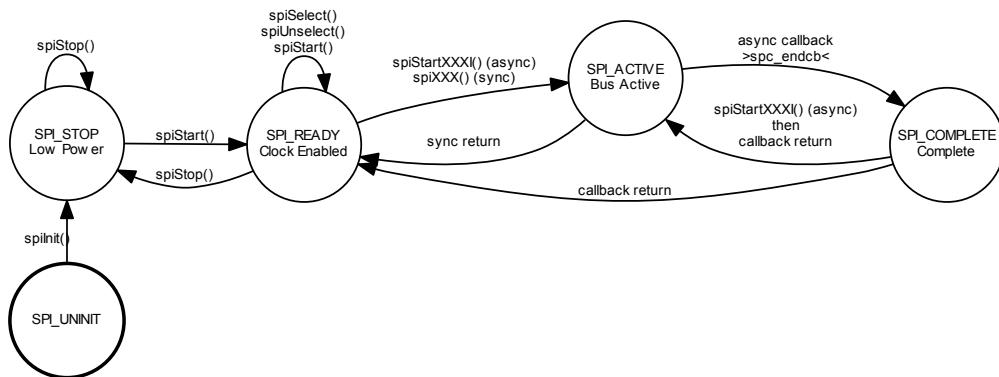
This module implements a generic SPI (Serial Peripheral Interface) driver allowing bidirectional and monodirectional transfers, complex atomic transactions are supported as well.

Precondition

In order to use the SPI driver the `HAL_USE_SPI` option must be enabled in `halconf.h`.

7.31.2 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.

SPI configuration options

- `#define SPI_USE_WAIT TRUE`
Enables synchronous APIs.
- `#define SPI_USE_MUTUAL_EXCLUSION TRUE`
Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.

Macro Functions

- `#define spiSelectI(spip)`

- **#define spiUnselectl(spi)**
Asserts the slave select signal and prepares for transfers.
- **#define spiStartIgnoreL(spi, n)**
Deasserts the slave select signal.
- **#define spiStartExchangeL(spi, n, txbuf, rxbuf)**
Ignores data on the SPI bus.
- **#define spiStartSendL(spi, n, txbuf)**
Exchanges data on the SPI bus.
- **#define spiStartReceiveL(spi, n, rxbuf)**
Sends data over the SPI bus.
- **#define spiPolledExchange(spi, frame) spi_lld_polled_exchange(spi, frame)**
Receives data from the SPI bus.
- **#define spiPolledExchange(spi, frame) spi_lld_polled_exchange(spi, frame)**
Exchanges one frame using a polled wait.

Low level driver helper macros

- **#define _spi_wakeup_isr(spi)**
Wakes up the waiting thread.
- **#define _spi_isr_code(spi)**
Common ISR code.

PLATFORM configuration options

- **#define PLATFORM_SPI_USE_SPI1 FALSE**
SPI1 driver enable switch.

Typedefs

- **typedef struct SPIDriver SPIDriver**
Type of a structure representing an SPI driver.
- **typedef void(* spicallback_t) (SPIDriver *spip)**
SPI notification callback type.

Data Structures

- **struct SPIConfig**
Driver configuration structure.
- **struct SPIDriver**
Structure representing an SPI driver.

Functions

- **void spiInit (void)**
SPI Driver initialization.
- **void spiObjectInit (SPIDriver *spip)**
Initializes the standard part of a `SPIDriver` structure.
- **void spiStart (SPIDriver *spip, const SPIConfig *config)**
Configures and activates the SPI peripheral.
- **void spiStop (SPIDriver *spip)**

- **Deactivates the SPI peripheral.**
- void **spiSelect (SPIDriver *spip)**
Asserts the slave select signal and prepares for transfers.
- void **spiUnselect (SPIDriver *spip)**
Deasserts the slave select signal.
- void **spiStartIgnore (SPIDriver *spip, size_t n)**
Ignores data on the SPI bus.
- void **spiStartExchange (SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)**
Exchanges data on the SPI bus.
- void **spiStartSend (SPIDriver *spip, size_t n, const void *txbuf)**
Sends data over the SPI bus.
- void **spiStartReceive (SPIDriver *spip, size_t n, void *rxbuf)**
Receives data from the SPI bus.
- void **spignore (SPIDriver *spip, size_t n)**
Ignores data on the SPI bus.
- void **spiExchange (SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)**
Exchanges data on the SPI bus.
- void **spiSend (SPIDriver *spip, size_t n, const void *txbuf)**
Sends data over the SPI bus.
- void **spiReceive (SPIDriver *spip, size_t n, void *rxbuf)**
Receives data from the SPI bus.
- void **spiAcquireBus (SPIDriver *spip)**
Gains exclusive access to the SPI bus.
- void **spiReleaseBus (SPIDriver *spip)**
Releases exclusive access to the SPI bus.
- void **spi_lld_init (void)**
Low level SPI driver initialization.
- void **spi_lld_start (SPIDriver *spip)**
Configures and activates the SPI peripheral.
- void **spi_lld_stop (SPIDriver *spip)**
Deactivates the SPI peripheral.
- void **spi_lld_select (SPIDriver *spip)**
Asserts the slave select signal and prepares for transfers.
- void **spi_lld_unselect (SPIDriver *spip)**
Deasserts the slave select signal.
- void **spi_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.
- 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.

7.31.3 Macro Definition Documentation

7.31.3.1 #define SPI_USE_WAIT TRUE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.31.3.2 #define SPI_USE_MUTUAL_EXCLUSION TRUE

Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.

Note

Disabling this option saves both code and data space.

7.31.3.3 #define spiSelect(spip)

Value:

```
{                                     \
    spi_lld_select(spip);           \
}
```

Asserts the slave select signal and prepares for transfers.

Parameters

in	<code>spip</code>	pointer to the <code>SPIDriver</code> 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.

7.31.3.4 #define spiUnselect(spip)

Value:

```
{                                     \
    spi_lld_unselect(spip);           \
}
```

Deasserts the slave select signal.

The previously selected peripheral is unselected.

Parameters

in	<i>spip</i>	pointer to the SPIDriver object
----	-------------	---

Function Class:

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

7.31.3.5 #define spiStartIgnore(*spip*, *n*)**Value:**

```
{
  (spip)->state = SPI_ACTIVE;
  \
  spi_llld_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	<i>spip</i>	pointer to the SPIDriver object
in	<i>n</i>	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.

7.31.3.6 #define spiStartExchange(*spip*, *n*, *txbuf*, *rxbuf*)**Value:**

```
{
  (spip)->state = SPI_ACTIVE;
  \
  spi_llld_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

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>n</i>	number of words to be exchanged
in	<i>txbuf</i>	the pointer to the transmit buffer
out	<i>rxbuf</i>	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.

7.31.3.7 #define spiStartSend(*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

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>n</i>	number of words to send
in	<i>txbuf</i>	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.

7.31.3.8 #define spiStartReceive(*spip*, *n*, *rxbuf*)

Value:

```
{
  (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

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>n</i>	number of words to receive
out	<i>rxbuf</i>	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.

7.31.3.9 #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

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>frame</i>	the data frame to send over the SPI bus

Returns

The received data frame from the SPI bus.

7.31.3.10 #define _spi_wakeup_isr(*spip*)**Value:**

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

Wakes up the waiting thread.

Parameters

in	<i>spip</i>	pointer to the SPIDriver object
----	-------------	---

Function Class:

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

7.31.3.11 #define _spi_isr_code(*spip*)**Value:**

```
{
    if ((spip)->config->end_cb) {
        (spip)->state = SPI_COMPLETE;
        (spip)->config->end_cb(spip);
        if ((spip)->state == SPI_COMPLETE)
            (spip)->state = SPI_READY;
    }
    else
        (spip)->state = SPI_READY;
    \
    _spi_wakeup_isr(spip);
}
```

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	<i>spip</i>	pointer to the SPIDriver object
----	-------------	---

Function Class:

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

7.31.3.12 #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.

7.31.4 Typedef Documentation

7.31.4.1 typedef struct SPIDriver SPIDriver

Type of a structure representing an SPI driver.

7.31.4.2 typedef void(* spicallback_t)(SPIDriver *spip)

SPI notification callback type.

Parameters

in	spip	pointer to the SPIDriver object triggering the callback
----	------	---

7.31.5 Enumeration Type Documentation

7.31.5.1 enum spistate_t

Driver state machine possible states.

Enumerator

SPI_UNINIT Not initialized.

SPI_STOP Stopped.

SPI_READY Ready.

SPI_ACTIVE Exchanging data.

SPI_COMPLETE Asynchronous operation complete.

7.31.6 Function Documentation

7.31.6.1 void spilinit(void)

SPI Driver initialization.

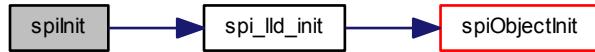
Note

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:



7.31.6.2 void spiObjectInit ([SPIDriver](#) * *spip*)

Initializes the standard part of a [SPIDriver](#) structure.

Parameters

out	<i>spip</i>	pointer to the SPIDriver object
-----	-------------	---

Function Class:

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

Here is the call graph for this function:



7.31.6.3 void spiStart ([SPIDriver](#) * *spip*, const [SPIConfig](#) * *config*)

Configures and activates the SPI peripheral.

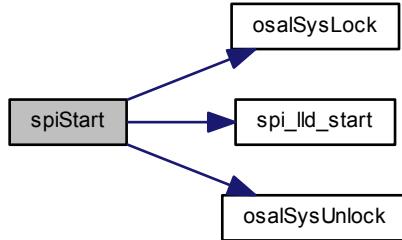
Parameters

in	<i>spip</i>	pointer to the SPIDriver object
in	<i>config</i>	pointer to the SPIConfig object

Function Class:

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

Here is the call graph for this function:



7.31.6.4 void spiStop (SPIDriver * spip)

Deactivates the SPI peripheral.

Note

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

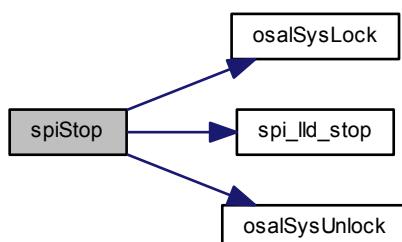
Parameters

in	<i>spip</i>	pointer to the <code>SPIDriver</code> 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:



7.31.6.5 void spiSelect (SPIDriver * spip)

Asserts the slave select signal and prepares for transfers.

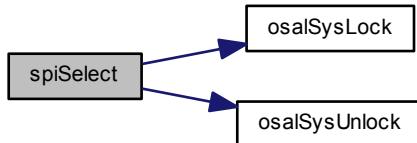
Parameters

in	<i>spip</i>	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:

**7.31.6.6 void spiUnselect (SPIDriver * *spip*)**

Deasserts the slave select signal.

The previously selected peripheral is unselected.

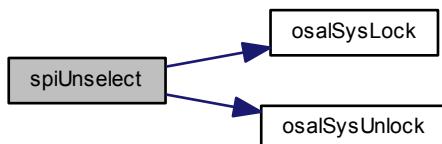
Parameters

in	<i>spip</i>	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:

**7.31.6.7 void spiStartIgnore (SPIDriver * *spip*, size_t *n*)**

Ignores data on the SPI bus.

This asynchronous function starts the transmission of a series of idle words on the SPI bus and ignores the received data.

Precondition

A slave must have been selected using `spiSelect()` or `spiSelectI()`.

Postcondition

At the end of the operation the configured callback is invoked.

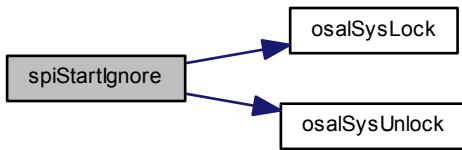
Parameters

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>n</i>	number of words to be ignored

Function Class:

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

Here is the call graph for this function:

**7.31.6.8 void spiStartExchange (SPIDriver * spip, size_t n, const void * txbuf, void * rdbuf)**

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.

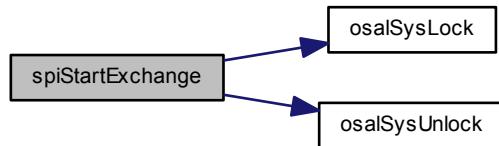
Parameters

in	<i>spip</i>	pointer to the SPIDriver object
in	<i>n</i>	number of words to be exchanged
in	<i>txbuf</i>	the pointer to the transmit buffer
out	<i>rxbuf</i>	the pointer to the receive 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:

**7.31.6.9 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.

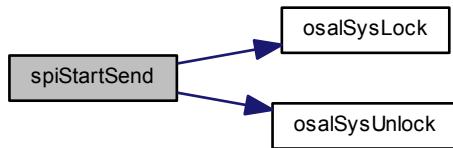
Parameters

in	<i>spip</i>	pointer to the SPIDriver object
in	<i>n</i>	number of words to send
in	<i>txbuf</i>	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:



7.31.6.10 void spiStartReceive (*SPIDriver* * *spip*, *size_t* *n*, *void* * *rxbuf*)

Receives data from the SPI bus.

This asynchronous function starts a receive operation.

Precondition

A slave must have been selected using `spiSelect()` or `spiSelectI()`.

Postcondition

At the end of the operation the configured callback is invoked.

Note

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

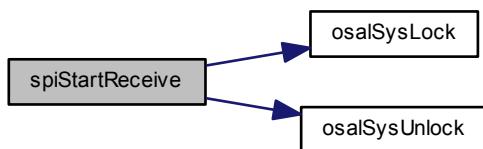
Parameters

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>n</i>	number of words to receive
out	<i>rxbuf</i>	the pointer to the receive 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:



7.31.6.11 void spilgnore (*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).

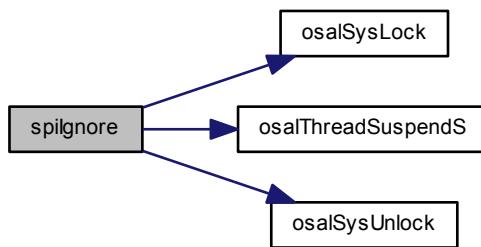
Parameters

in	<i>spip</i>	pointer to the <i>SPIDriver</i> object
in	<i>n</i>	number of words to be ignored

Function Class:

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

Here is the call graph for this function:



7.31.6.12 void spiExchange (*SPIDriver * spip*, *size_t n*, *const void * txbuf*, *void * rdbuf*)

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.

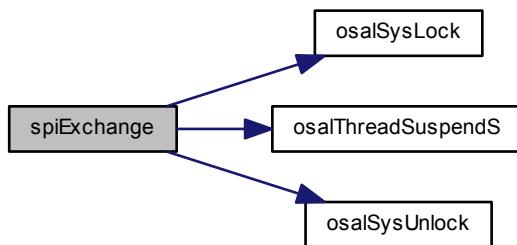
Parameters

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>n</i>	number of words to be exchanged
in	<i>txbuf</i>	the pointer to the transmit buffer
out	<i>rxbuf</i>	the pointer to the receive 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:

**7.31.6.13 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.

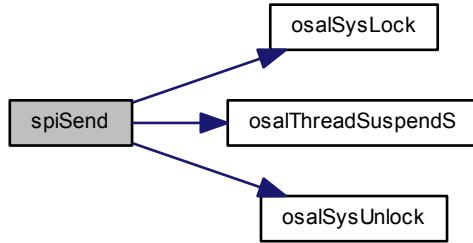
Parameters

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>n</i>	number of words to send
in	<i>txbuf</i>	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:



7.31.6.14 void spiReceive (`SPIDriver * spip`, `size_t n`, `void * rdbuf`)

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.

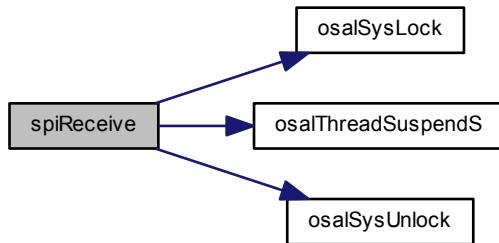
Parameters

in	<code>spip</code>	pointer to the <code>SPIDriver</code> object
in	<code>n</code>	number of words to receive
out	<code>rdbuf</code>	the pointer to the receive 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:



7.31.6.15 void spiAcquireBus (SPIDriver * spip)

Gains exclusive access to the SPI bus.

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

Precondition

In order to use this function the option SPI_USE_MUTUAL_EXCLUSION must be enabled.

Parameters

in	<i>spip</i>	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:



7.31.6.16 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	<i>spip</i>	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:

**7.31.6.17 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:

**7.31.6.18 void spi_lld_start(SPIDriver * spip)**

Configures and activates the SPI peripheral.

Parameters

in	<i>spip</i>	pointer to the SPIDriver object
----	-------------	---

Function Class:

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

7.31.6.19 void spi_lld_stop(SPIDriver * spip)

Deactivates the SPI peripheral.

Parameters

in	<i>spip</i>	pointer to the SPIDriver object
----	-------------	---

Function Class:

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

7.31.6.20 void spi_lld_select ([SPIDriver](#) * *spip*)

Asserts the slave select signal and prepares for transfers.

Parameters

in	<i>spip</i>	pointer to the SPIDriver object
----	-------------	---

Function Class:

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

7.31.6.21 void spi_lld_unselect ([SPIDriver](#) * *spip*)

Deasserts the slave select signal.

The previously selected peripheral is unselected.

Parameters

in	<i>spip</i>	pointer to the SPIDriver object
----	-------------	---

Function Class:

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

7.31.6.22 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

in	<i>spip</i>	pointer to the SPIDriver object
in	<i>n</i>	number of words to be ignored

Function Class:

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

7.31.6.23 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

in	<i>spip</i>	pointer to the SPIDriver object
in	<i>n</i>	number of words to be exchanged
in	<i>txbuf</i>	the pointer to the transmit buffer
out	<i>rxbuf</i>	the pointer to the receive buffer

Function Class:

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

7.31.6.24 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

in	<i>spip</i>	pointer to the SPIDriver object
in	<i>n</i>	number of words to send
in	<i>txbuf</i>	the pointer to the transmit buffer

Function Class:

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

7.31.6.25 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

in	<i>spip</i>	pointer to the SPIDriver object
in	<i>n</i>	number of words to receive
out	<i>rdbuf</i>	the pointer to the receive buffer

Function Class:

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

7.31.6.26 uint16_t spi_lld_polled_exchange ([SPIDriver](#) * *spip*, uint16_t *frame*)

Exchanges one frame using a polled wait.

This synchronous function exchanges one frame using a polled synchronization method. This function is useful when exchanging small amount of data on high speed channels, usually in this situation is much more efficient just wait for completion using polling than suspending the thread waiting for an interrupt.

Parameters

in	<i>spip</i>	pointer to the SPIDriver object
in	<i>frame</i>	the data frame to send over the SPI bus

Returns

The received data frame from the SPI bus.

7.31.7 Variable Documentation**7.31.7.1 SPIDriver SPID1**

SPI1 driver identifier.

7.32 ST Driver

Generic System Tick driver.

7.32.1 Detailed Description

Generic System Tick driver.

This module implements a system tick timer in order to support the underlying operating system.

Macro Functions

- `#define stGetCounter() st_lld_get_counter()`
Returns the time counter value.
- `#define stIsAlarmActive() st_lld_is_alarm_active()`
Determines if the alarm is active.

Functions

- `void stInit (void)`
ST Driver initialization.
- `void stStartAlarm (systime_t abstime)`
Starts the alarm.
- `void stStopAlarm (void)`
Stops the alarm interrupt.
- `void stSetAlarm (systime_t abstime)`
Sets the alarm time.
- `systime_t stGetAlarm (void)`
Returns the current alarm 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.

7.32.2 Macro Definition Documentation

7.32.2.1 #define stGetCounter() st_lld_get_counter()

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.

7.32.2.2 #define stIsAlarmActive() st_lld_is_alarm_active()

Determines if the alarm is active.

Returns

The alarm status.

Return values

<i>false</i>	if the alarm is not active.
<i>true</i>	is the alarm is active

Function Class:

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

7.32.3 Function Documentation**7.32.3.1 void stlInit(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.

Here is the call graph for this function:



7.32.3.2 void stStartAlarm (*systime_t abstime*)

Starts the 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

in	<i>abstime</i>	the time to be set for the first alarm
----	----------------	--

Function Class:

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

Here is the call graph for this function:



7.32.3.3 void stStopAlarm (void)

Stops the alarm 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:



7.32.3.4 void stSetAlarm (*systime_t abstime*)

Sets the alarm time.

Note

This functionality is only available in free running mode, the behavior in periodic mode is undefined.

Parameters

in	<i>abstime</i>	the time to be set for the next alarm
----	----------------	---------------------------------------

Function Class:

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

Here is the call graph for this function:

**7.32.3.5 systime_t stGetAlarm(void)**

Returns the current alarm time.

Note

This functionality is only available in free running mode, the behavior in periodic mode is undefined.

Returns

The currently set alarm time.

Function Class:

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

Here is the call graph for this function:

**7.32.3.6 void st_lld_init(void)**

Low level ST driver initialization.

Function Class:

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

7.32.3.7 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.

7.32.3.8 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.

7.32.3.9 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.

7.32.3.10 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.

7.32.3.11 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.

7.32.3.12 static bool st_lld_is_alarm_active(void) [inline], [static]

Determines if the alarm is active.

Returns

The alarm status.

Return values

<i>false</i>	if the alarm is not active.
<i>true</i>	is the alarm is active

Function Class:

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

7.33 UART Driver

Generic UART Driver.

7.33.1 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.

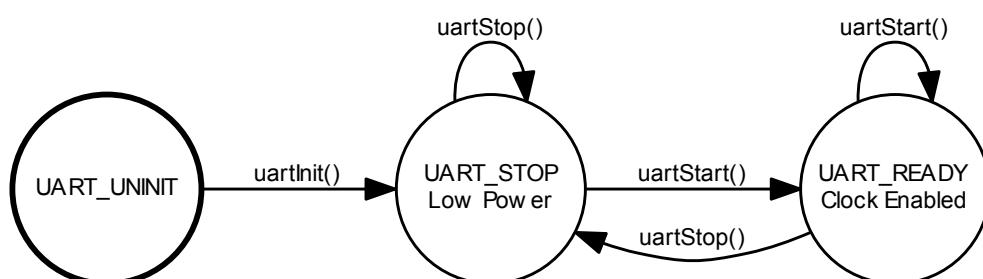
If your application requires a synchronous buffered driver then the [Serial Driver](#) should be used instead.

Precondition

In order to use the UART driver the `HAL_USE_UART` option must be enabled in [`halconf.h`](#).

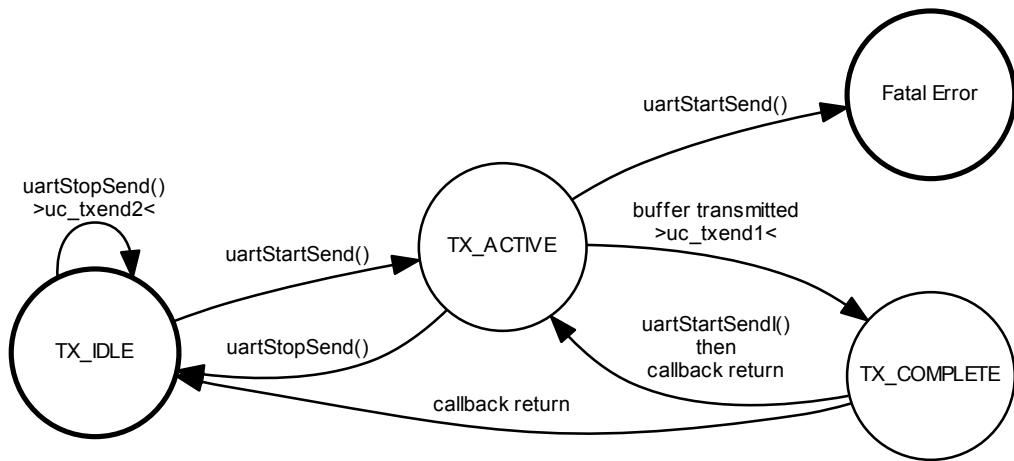
7.33.2 Driver State Machine

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).



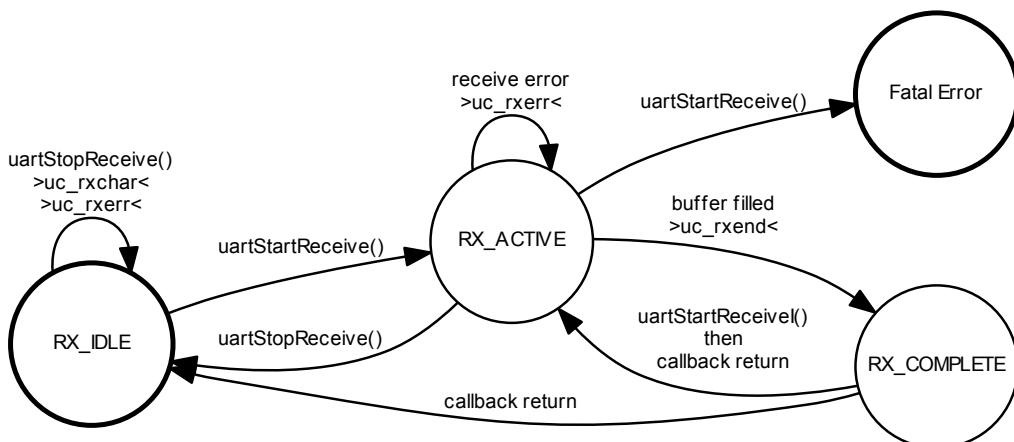
7.33.2.1 Transmitter sub State Machine

The follow diagram describes the transmitter state machine, this diagram is valid while the driver is in the `UART←T_READY` state. This state machine is automatically reset to the `TX_IDLE` state each time the driver enters the `UART_READY` state.



7.33.2.2 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.



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.

PLATFORM configuration options

- #define **PLATFORM_UART_USE_UART1** FALSE
UART driver enable switch.

Typedefs

- typedef uint32_t **uartflags_t**
UART driver condition flags type.
- typedef struct **UARTDriver** **UARTDriver**
Type of structure representing an UART driver.
- typedef void(* **uartcb_t**) (**UARTDriver** *uartp)
Generic UART notification callback type.
- typedef void(* **uartccb_t**) (**UARTDriver** *uartp, uint16_t c)
Character received UART notification callback type.
- typedef void(* **uartecb_t**) (**UARTDriver** *uartp, **uartflags_t** e)
Receive error UART notification callback type.

Data Structures

- struct **UARTConfig**
Driver configuration structure.
- struct **UARTDriver**
Structure representing an UART driver.

Functions

- void **uartInit** (void)
UART Driver initialization.
- void **uartObjectInit** (**UARTDriver** *uartp)
*Initializes the standard part of a **UARTDriver** structure.*
- void **uartStart** (**UARTDriver** *uartp, const **UARTConfig** *config)
Configures and activates the UART peripheral.
- void **uartStop** (**UARTDriver** *uartp)
Deactivates the UART peripheral.

- void `uartStartSend (UARTDriver *uartp, size_t n, const void *txbuf)`
Starts a transmission on the UART peripheral.
- void `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 `uartStartReceivel (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 `uartStopReceivel (UARTDriver *uartp)`
Stops any ongoing receive operation.
- void `uart_lld_init (void)`
Low level UART driver initialization.
- void `uart_lld_start (UARTDriver *uartp)`
Configures and activates the UART peripheral.
- void `uart_lld_stop (UARTDriver *uartp)`
Deactivates the UART peripheral.
- void `uart_lld_start_send (UARTDriver *uartp, size_t n, const void *txbuf)`
Starts a transmission on the UART peripheral.
- size_t `uart_lld_stop_send (UARTDriver *uartp)`
Stops any ongoing transmission.
- void `uart_lld_start_receive (UARTDriver *uartp, size_t n, void *rxbuf)`
Starts a receive operation on the UART peripheral.
- size_t `uart_lld_stop_receive (UARTDriver *uartp)`
Stops any ongoing receive operation.

Enumerations

- enum `uartstate_t { UART_UNINIT = 0, UART_STOP = 1, UART_READY = 2 }`
Driver state machine possible states.
- enum `uartxstate_t { UART_TX_IDLE = 0, UART_TX_ACTIVE = 1, UART_TX_COMPLETE = 2 }`
Transmitter state machine states.
- enum `uartrxstate_t { UART_RX_IDLE = 0, UART_RX_ACTIVE = 1, UART_RX_COMPLETE = 2 }`
Receiver state machine states.

Variables

- `UARTDriver UARTD1`
UART1 driver identifier.

7.33.3 Macro Definition Documentation

7.33.3.1 #define `UART_NO_ERROR 0`

No pending conditions.

7.33.3.2 `#define UART_PARITY_ERROR 4`

Parity error happened.

7.33.3.3 `#define UART_FRAMING_ERROR 8`

Framing error happened.

7.33.3.4 `#define UART_OVERRUN_ERROR 16`

Overflow happened.

7.33.3.5 `#define UART_NOISE_ERROR 32`

Noise on the line.

7.33.3.6 `#define UART_BREAK_DETECTED 64`

Break detected.

7.33.3.7 `#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.

7.33.4 TYPEDOC Documentation

7.33.4.1 `typedef uint32_t uartflags_t`

UART driver condition flags type.

7.33.4.2 `typedef struct UARTDriver UARTDriver`

Type of structure representing an UART driver.

7.33.4.3 `typedef void(* uartcb_t) (UARTDriver *uartp)`

Generic UART notification callback type.

Parameters

in	<code>uartp</code>	pointer to the <code>UARTDriver</code> object
----	--------------------	---

7.33.4.4 `typedef void(* uartccb_t) (UARTDriver *uartp, uint16_t c)`

Character received UART notification callback type.

Parameters

in	<i>uartp</i>	pointer to the UARTDriver object triggering the callback
in	<i>c</i>	received character

7.33.4.5 `typedef void(* uartcb_t) (UARTDriver *uartp, uartflags_t e)`

Receive error UART notification callback type.

Parameters

in	<i>uartp</i>	pointer to the UARTDriver object triggering the callback
in	<i>e</i>	receive error mask

7.33.5 Enumeration Type Documentation**7.33.5.1 `enum uartstate_t`**

Driver state machine possible states.

Enumerator

UART_UNINIT Not initialized.

UART_STOP Stopped.

UART_READY Ready.

7.33.5.2 `enum uarttxstate_t`

Transmitter state machine states.

Enumerator

UART_TX_IDLE Not transmitting.

UART_TX_ACTIVE Transmitting.

UART_TX_COMPLETE Buffer complete.

7.33.5.3 `enum uartrxstate_t`

Receiver state machine states.

Enumerator

UART_RX_IDLE Not receiving.

UART_RX_ACTIVE Receiving.

UART_RX_COMPLETE Buffer complete.

7.33.6 Function Documentation**7.33.6.1 `void uartInit(void)`**

UART Driver initialization.

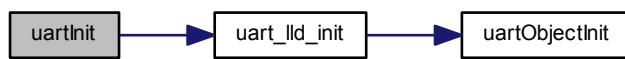
Note

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:

**7.33.6.2 void uartObjectInit (**UARTDriver** * *uartp*)**

Initializes the standard part of a `UARTDriver` structure.

Parameters

<code>out</code>	<code>uartp</code>	pointer to the <code>UARTDriver</code> object
------------------	--------------------	---

Function Class:

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

7.33.6.3 void uartStart (**UARTDriver * *uartp*, const **UARTConfig** * *config*)**

Configures and activates the UART peripheral.

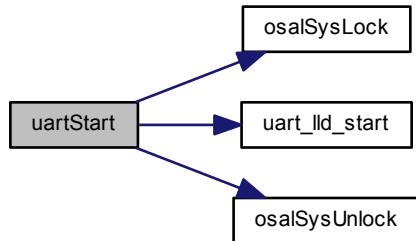
Parameters

<code>in</code>	<code>uartp</code>	pointer to the <code>UARTDriver</code> object
<code>in</code>	<code>config</code>	pointer to the <code>UARTConfig</code> 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:



7.33.6.4 void uartStop (`UARTDriver * uartp`)

Deactivates the UART peripheral.

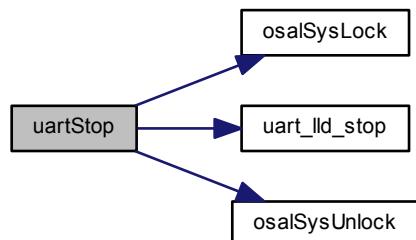
Parameters

in	<code>uartp</code>	pointer to the <code>UARTDriver</code> 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:



7.33.6.5 void uartStartSend (`UARTDriver * uartp`, `size_t n`, `const void * txbuf`)

Starts a transmission on the UART peripheral.

Note

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

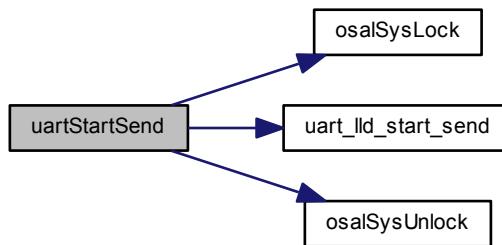
Parameters

in	<i>uartp</i>	pointer to the UARTDriver object
in	<i>n</i>	number of data frames to send
in	<i>txbuf</i>	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:



7.33.6.6 void uartStartSendl ([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

in	<i>uartp</i>	pointer to the UARTDriver object
in	<i>n</i>	number of data frames to send
in	<i>txbuf</i>	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.

Here is the call graph for this function:



7.33.6.7 size_t uartStopSend (**UARTDriver** * *uartp*)

Stops any ongoing transmission.

Note

Stopping a transmission also suppresses the transmission callbacks.

Parameters

in	<i>uartp</i>	pointer to the UARTDriver object
----	--------------	---

Returns

The number of data frames not transmitted by the stopped transmit operation.

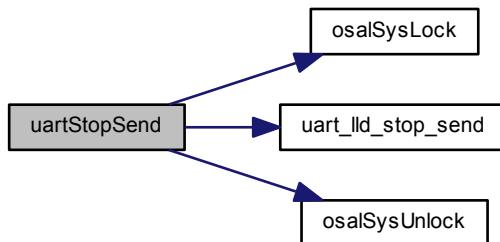
Return values

0	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:



7.33.6.8 size_t uartStopSendl (**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	<i>uartp</i>	pointer to the UARTDriver object
----	--------------	--

Returns

The number of data frames not transmitted by the stopped transmit operation.

Return values

0	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:



7.33.6.9 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.

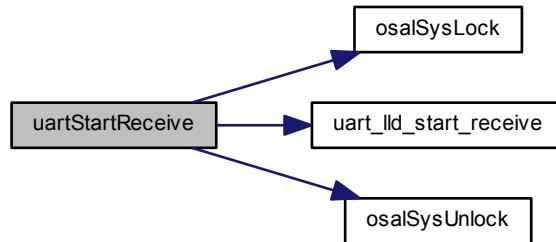
Parameters

in	<i>uartp</i>	pointer to the UARTDriver object
in	<i>n</i>	number of data frames to receive
in	<i>rxbuf</i>	the pointer to the receive 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:



7.33.6.10 void uartStartReceive(`UARTDriver *uartp`, `size_t n`, `void *rdbuf`)

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

in	<code>uartp</code>	pointer to the <code>UARTDriver</code> object
in	<code>n</code>	number of data frames to receive
out	<code>rdbuf</code>	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.

Here is the call graph for this function:



7.33.6.11 `size_t uartStopReceive(UARTDriver *uartp)`

Stops any ongoing receive operation.

Note

Stopping a receive operation also suppresses the receive callbacks.

Parameters

in	<i>uartp</i>	pointer to the UARTDriver object
----	--------------	--

Returns

The number of data frames not received by the stopped receive operation.

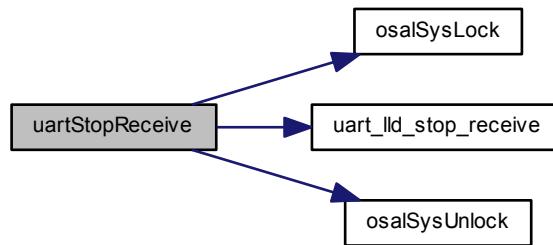
Return values

0	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:

**7.33.6.12 size_t uartStopReceive([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	<i>uartp</i>	pointer to the UARTDriver object
----	--------------	--

Returns

The number of data frames not received by the stopped receive operation.

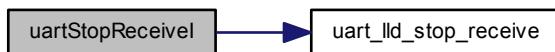
Return values

0	There was no receive operation in progress.
---	---

Function Class:

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

Here is the call graph for this function:



7.33.6.13 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:

7.33.6.14 void uart_lld_start(**UARTDriver** * *uartp*)

Configures and activates the UART peripheral.

Parameters

in	<i>uartp</i>	pointer to the UARTDriver object
----	--------------	--

Function Class:

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

7.33.6.15 void uart_lld_stop(**UARTDriver** * *uartp*)

Deactivates the UART peripheral.

Parameters

in	<i>uartp</i>	pointer to the UARTDriver object
----	--------------	--

Function Class:

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

7.33.6.16 void uart_lld_start_send ([UARTDriver](#) * *uartp*, [size_t](#) *n*, const void * *txbuf*)

Starts a transmission on the UART peripheral.

Note

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

Parameters

in	<i>uartp</i>	pointer to the UARTDriver object
in	<i>n</i>	number of data frames to send
in	<i>txbuf</i>	the pointer to the transmit buffer

Function Class:

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

7.33.6.17 [size_t](#) uart_lld_stop_send ([UARTDriver](#) * *uartp*)

Stops any ongoing transmission.

Note

Stopping a transmission also suppresses the transmission callbacks.

Parameters

in	<i>uartp</i>	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.

7.33.6.18 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	<i>uartp</i>	pointer to the UARTDriver object
in	<i>n</i>	number of data frames to send
out	<i>rxbuf</i>	the pointer to the receive buffer

Function Class:

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

7.33.6.19 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	<i>uartp</i>	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.

7.33.7 Variable Documentation**7.33.7.1 [UARTDriver](#) UARTD1**

UART1 driver identifier.

7.34 USB Driver

Generic USB Driver.

7.34.1 Detailed Description

Generic USB Driver.

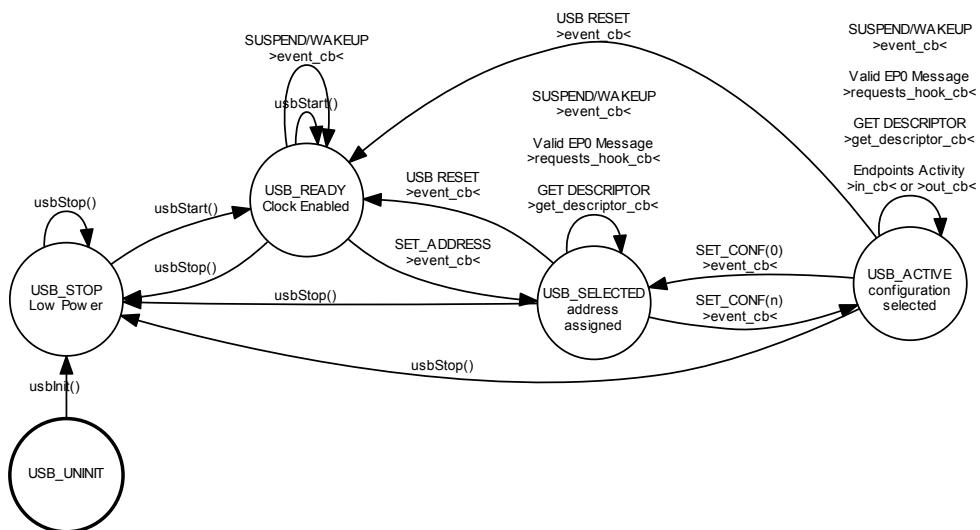
This module implements a generic USB (Universal Serial Bus) driver supporting device-mode operations.

Precondition

In order to use the USB driver the `HAL_USE_USB` option must be enabled in `halconf.h`.

7.34.2 Driver State Machine

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



7.34.3 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.

7.34.3.1 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

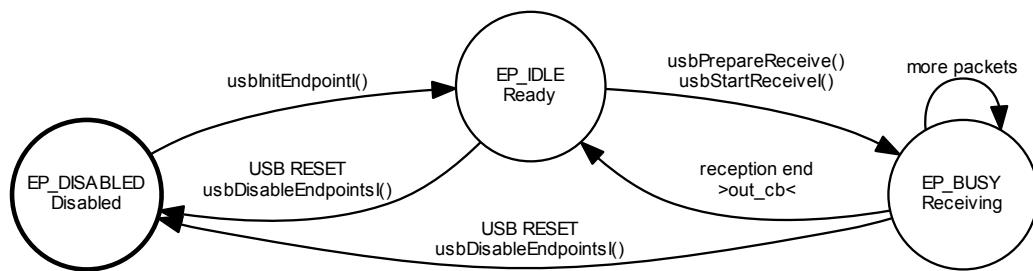
required by the USB device class to be implemented and of the handling of the specific messages sent over the endpoint zero. Standard messages are handled internally to the driver. The application can use hooks in order to handle custom messages or override the handling of the default handling of standard messages.

7.34.3.2 USB Endpoints

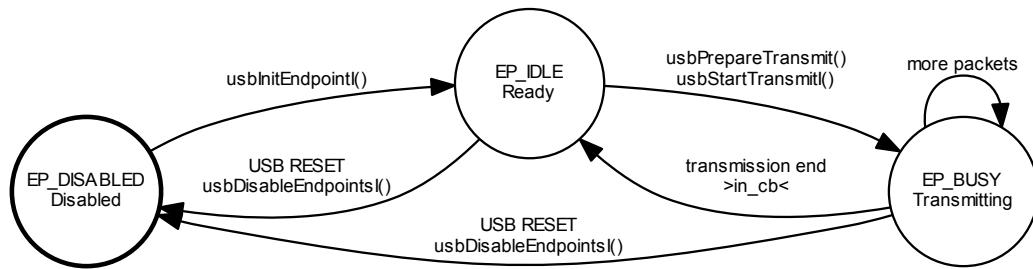
USB endpoints are the objects that the application uses to exchange data with the host. There are two kind of endpoints:

- **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:



7.34.3.3 USB Callbacks

The USB driver uses callbacks in order to interact with the application. There are several kinds of callbacks to be handled:

- Driver events callback. As example errors, suspend event, reset event etc.
- Messages Hook callback. This hook allows the application to implement handling of custom messages or to override the default handling of standard messages on endpoint zero.
- Descriptor Requested callback. When the driver endpoint zero handler receives a GET DESCRIPTOR message and needs to send a descriptor to the host it queries the application using this callback.
- Start of Frame callback. This callback is invoked each time a SOF packet is received.
- Endpoint callbacks. Each endpoint informs the application about I/O conditions using those callbacks.

Macros

- `#define USB_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_ll_get_frame_number(usbp) 0`
Returns the current frame number.
- `#define usb_ll_get_transaction_size(usbp, ep) ((usbp)->epc[ep]->out_state->rxcnt)`
Returns the exact size of a receive transaction.
- `#define usb_ll_connect_bus(usbp)`
Connects the USB device.
- `#define usb_ll_disconnect_bus(usbp)`
Disconnect the USB device.

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)`

- **USB_DESC_INTERFACE_SIZE** 9U
Interface Descriptor size.
- **USB_DESC_INTERFACE**(bInterfaceNumber, bAlternateSetting, bNumEndpoints, bInterfaceClass, bInterfaceSubClass, bInterfaceProtocol, iInterface)
Interface Descriptor helper macro.
- **USB_DESC_INTERFACE_ASSOCIATION_SIZE** 8U
Interface Association Descriptor size.
- **USB_DESC_INTERFACE_ASSOCIATION**(bFirstInterface, bInterfaceCount, bFunctionClass, bFunctionSubClass, bFunctionProtocol, iInterface)
Interface Association Descriptor helper macro.
- **USB_DESC_ENDPOINT_SIZE** 7U
Endpoint Descriptor size.
- **USB_DESC_ENDPOINT**(bEndpointAddress, bmAttributes, wMaxPacketSize, bInterval)
Endpoint Descriptor helper macro.

Endpoint types and settings

- **USB_EP_MODE_TYPE** 0x0003U
- **USB_EP_MODE_TYPE_CTRL** 0x0000U
- **USB_EP_MODE_TYPE_ISOC** 0x0001U
- **USB_EP_MODE_TYPE_BULK** 0x0002U
- **USB_EP_MODE_TYPE_INTR** 0x0003U
- **USB_EP_MODE_LINEAR_BUFFER** 0x0000U
- **USB_EP_MODE_QUEUE_BUFFER** 0x0010U

Macro Functions

- **usbGetDriverState**(usbp) ((usbp)->state)
Returns the driver state.
- **usbConnectBus**(usbp) **usb_lld_connect_bus**(usbp)
Connects the USB device.
- **usbDisconnectBus**(usbp) **usb_lld_disconnect_bus**(usbp)
Disconnect the USB device.
- **usbGetFrameNumber**(usbp) **usb_lld_get_frame_number**(usbp)
Returns the current frame number.
- **usbGetTransmitStatus**(usbp, ep) (((usbp)->transmitting & (uint16_t)((unsigned)1U << (unsigned)(ep))) != 0U)
Returns the status of an IN endpoint.
- **usbGetReceiveStatus**(usbp, ep) (((usbp)->receiving & (uint16_t)((unsigned)1U << (unsigned)(ep))) != 0U)
Returns the status of an OUT endpoint.
- **usbGetReceiveTransactionSize**(usbp, ep) **usb_lld_get_transaction_size**(usbp, ep)
Returns the exact size of a receive transaction.
- **usbSetupTransfer**(usbp, buf, n, endcb)
Request transfer setup.
- **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(* usbeppcallback_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.
- `typedef const USBDescriptor *(* usbgetdescriptor_t) (USBDriver *usbp, uint8_t dtype, uint8_t dindex, uint16_t lang)`
Type of an USB descriptor-retrieving 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 `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 `usbInitEndpoint` (`USBDriver` *`usbp`, `usbep_t` `ep`, const `USBEndpointConfig` *`epcp`)

Enables an endpoint.
- void `usbDisableEndpoints` (`USBDriver` *`usbp`)

Disables all the active endpoints.
- void `usbPrepareReceive` (`USBDriver` *`usbp`, `usbep_t` `ep`, `uint8_t` *`buf`, `size_t` `n`)

Prepares for a receive transaction on an OUT endpoint.
- void `usbPrepareTransmit` (`USBDriver` *`usbp`, `usbep_t` `ep`, const `uint8_t` *`buf`, `size_t` `n`)

Prepares for a transmit transaction on an IN endpoint.
- void `usbPrepareQueuedReceive` (`USBDriver` *`usbp`, `usbep_t` `ep`, `input_queue_t` *`iqp`, `size_t` `n`)

Prepares for a receive transaction on an OUT endpoint.
- void `usbPrepareQueuedTransmit` (`USBDriver` *`usbp`, `usbep_t` `ep`, `output_queue_t` *`oqp`, `size_t` `n`)

Prepares for a transmit transaction on an IN endpoint.
- bool `usbStartReceivel` (`USBDriver` *`usbp`, `usbep_t` `ep`)

Starts a receive transaction on an OUT endpoint.
- bool `usbStartTransmitl` (`USBDriver` *`usbp`, `usbep_t` `ep`)

Starts a transmit transaction on an IN endpoint.
- bool `usbStallReceivel` (`USBDriver` *`usbp`, `usbep_t` `ep`)

Stalls an OUT endpoint.
- bool `usbStallTransmitl` (`USBDriver` *`usbp`, `usbep_t` `ep`)

Stalls an IN endpoint.
- 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)`
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.

Enumerations

- enum `usbstate_t` {
 `USB_UNINIT` = 0, `USB_STOP` = 1, `USB_READY` = 2, `USB_SELECTED` = 3,
`USB_ACTIVE` = 4, `USB_SUSPENDED` = 5 }

Type of a driver state machine possible states.
- enum `usbepstatus_t` { `EP_STATUS_DISABLED` = 0, `EP_STATUS_STALLED` = 1, `EP_STATUS_ACTIVE` = 2 }

Type of an endpoint status.
- enum `usbep0state_t` {
 `USB_EP0_WAITING_SETUP`, `USB_EP0_TX`, `USB_EP0_WAITING_TX0`, `USB_EP0_WAITING_STS`,
`USB_EP0_RX`, `USB_EP0_SENDING_STS`, `USB_EP0_ERROR` }

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_SUSPEND` = 3,
`USB_EVENT_WAKEUP` = 4, `USB_EVENT_STALLED` = 5 }

Type of an enumeration of the possible USB events.

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.

7.34.4 Macro Definition Documentation

7.34.4.1 #define **USB_DESC_INDEX(i)** ((**uint8_t**)(i))

Helper macro for index values into descriptor strings.

7.34.4.2 #define **USB_DESC_BYTE(b)** ((**uint8_t**)(b))

Helper macro for byte values into descriptor strings.

7.34.4.3 #define **USB_DESC_WORD(w)**

Value:

```
(uint8_t)((w) & 255U),                                     \
  (uint8_t)(((w) >> 8) & 255U)
```

Helper macro for word values into descriptor strings.

7.34.4.4 #define **USB_DESC_BCD(bcd)**

Value:

```
(uint8_t)((bcd) & 255U),                                     \
  (uint8_t)(((bcd) >> 8) & 255U)
```

Helper macro for BCD values into descriptor strings.

7.34.4.5 #define **USB_DESC_DEVICE(bcdUSB, bDeviceClass, bDeviceSubClass, bDeviceProtocol, bMaxPacketSize, idVendor, idProduct, bcdDevice, iManufacturer, iProduct, iSerialNumber, bNumConfigurations)**

Value:

```
USB_DESC_BYTE(USB_DESC_DEVICE_SIZE),  

  \  

  USB_DESC_BYTE(USB_DESCRIPTOR_DEVICE),  

  \  

  USB_DESC_BCD(bcdUSB),  

  \  

  USB_DESC_BYTE(bDeviceClass),  

  \
```

```

USB_DESC_BYTE (bDeviceSubClass),
  \
  USB_DESC_BYTE (bDeviceProtocol),
    \
    USB_DESC_BYTE (bMaxPacketSize),
      \
      USB_DESC_WORD (idVendor),
        \
        USB_DESC_WORD (idProduct),
          \
          USB_DESC_BCD (bcdDevice),
            \
            USB_DESC_INDEX (iManufacturer),
              \
              USB_DESC_INDEX (iProduct),
                \
                USB_DESC_INDEX (iSerialNumber),
                  \
                  USB_DESC_BYTE (bNumConfigurations)

```

Device Descriptor helper macro.

7.34.4.6 #define USB_DESC_CONFIGURATION_SIZE 9U

Configuration Descriptor size.

7.34.4.7 #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.

7.34.4.8 #define USB_DESC_INTERFACE_SIZE 9U

Interface Descriptor size.

7.34.4.9 #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.

7.34.4.10 #define USB_DESC_INTERFACE_ASSOCIATION_SIZE 8U

Interface Association Descriptor size.

7.34.4.11 #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.

7.34.4.12 #define USB_DESC_ENDPOINT_SIZE 7U

Endpoint Descriptor size.

7.34.4.13 #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.

7.34.4.14 #define USB_EP_MODE_TYPE 0x0003U

Endpoint type mask.

7.34.4.15 #define USB_EP_MODE_TYPE_CTRL 0x0000U

Control endpoint.

7.34.4.16 #define USB_EP_MODE_TYPE_ISOC 0x0001U

Isochronous endpoint.

7.34.4.17 #define USB_EP_MODE_TYPE_BULK 0x0002U

Bulk endpoint.

7.34.4.18 #define USB_EP_MODE_TYPE_INTR 0x0003U

Interrupt endpoint.

7.34.4.19 #define USB_EP_MODE_LINEAR_BUFFER 0x0000U

Linear buffer mode.

7.34.4.20 #define USB_EP_MODE_QUEUE_BUFFER 0x0010U

Queue buffer mode.

7.34.4.21 #define usbGetDriverState(*usbp*) ((*usbp*)>state)

Returns the driver state.

Parameters

in	<i>usbp</i>	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.

7.34.4.22 #define usbConnectBus(*usbp*) [usb_ll_connect_bus](#)(*usbp*)

Connects the USB device.

Parameters

in	<i>usbp</i>	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.

7.34.4.23 #define usbDisconnectBus(*usbp*) usb_lld_disconnect_bus(*usbp*)

Disconnect the USB device.

Parameters

in	<i>usbp</i>	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.

7.34.4.24 #define usbGetFrameNumber(*usbp*) *usb_lld_get_frame_number(usbp)*

Returns the current frame number.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
----	-------------	---

Returns

The current frame number.

Function Class:

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

7.34.4.25 #define usbGetTransmitStatus(*usbp*, *ep*) (((*usbp*)>transmitting & (uint16_t)((unsigned)1U << (unsigned)(*ep*))) != 0U)

Returns the status of an IN endpoint.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Returns

The operation status.

Return values

<i>false</i>	Endpoint ready.
<i>true</i>	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.

7.34.4.26 #define usbGetReceiveStatus(*usbp*, *ep*) (((*usbp*)>receiving & (uint16_t)((unsigned)1U << (unsigned)(*ep*))) != 0U)

Returns the status of an OUT endpoint.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Returns

The operation status.

Return values

<i>false</i>	Endpoint ready.
<i>true</i>	Endpoint receiving.

Function Class:

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

7.34.4.27 #define **usbGetReceiveTransactionSize(*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

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Returns

Received data size.

Function Class:

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

7.34.4.28 #define **usbSetupTransfer(*usbp*, *buf*, *n*, *endcb*)**

Value:

```
{
  (usbp)->ep0next  = (buf);
  (usbp)->ep0n      = (n);
  (usbp)->ep0endcb = (endcb);
}
```

Request transfer setup.

This macro is used by the request handling callbacks in order to prepare a transaction over the endpoint zero.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>buf</i>	pointer to a buffer for the transaction data
in	<i>n</i>	number of bytes to be transferred
in	<i>endcb</i>	callback to be invoked after the transfer or NULL

Function Class:

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

7.34.4.29 #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

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number
out	<i>buf</i>	buffer where to copy the packet data

Function Class:

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

7.34.4.30 #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

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>evt</i>	USB event code

Function Class:

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

7.34.4.31 #define _usb_isr_invoke_sof_cb(*usbp*)

Value:

```
{\n    if (((usbp)->config->saf_cb) != NULL) {\n        (usbp)->config->saf_cb(usbp);\n    }\n}
```

Common ISR code, SOF callback.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
----	-------------	---

Function Class:

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

7.34.4.32 #define _usb_isr_invoke_setup_cb(*usbp*, *ep*)**Value:**

```
{
  (usbp) ->epc[ep] ->setup_cb(usbp, ep);
}
```

Common ISR code, setup packet callback.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

7.34.4.33 #define _usb_isr_invoke_in_cb(*usbp*, *ep*)**Value:**

```
{
  (usbp) ->transmitting &= ~(1 << (ep));
  (usbp) ->epc[ep] ->in_cb(usbp, ep);
}
```

Common ISR code, IN endpoint callback.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

7.34.4.34 #define _usb_isr_invoke_out_cb(*usbp*, *ep*)**Value:**

```
{
  (usbp) ->receiving &= ~(1 << (ep));
  (usbp) ->epc[ep] ->out_cb(usbp, ep);
}
```

Common ISR code, OUT endpoint event.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

7.34.4.35 #define USB_MAX_ENDPOINTS 4

Maximum endpoint address.

7.34.4.36 #define USB_EP0_STATUS_STAGE USB_EP0_STATUS_STAGE_SW

Status stage handling method.

7.34.4.37 #define USB_SET_ADDRESS_MODE USB_LATE_SET_ADDRESS

The address can be changed immediately upon packet reception.

7.34.4.38 #define USB_SET_ADDRESS_ACK_HANDLING USB_SET_ADDRESS_ACK_SW

Method for set address acknowledge.

7.34.4.39 #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.

7.34.4.40 #define usb_ll_get_frame_number(*usbp*) 0

Returns the current frame number.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
----	-------------	---

Returns

The current frame number.

Function Class:

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

```
7.34.4.41 #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

in	<code>usbp</code>	pointer to the <code>USBDriver</code> object
in	<code>ep</code>	endpoint number

Returns

Received data size.

Function Class:

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

```
7.34.4.42 #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.

```
7.34.4.43 #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.

7.34.5 Typedef Documentation

7.34.5.1 `typedef struct USBDriver USBDriver`

Type of a structure representing an USB driver.

```
7.34.5.2 typedef uint8_t usbep_t
```

Type of an endpoint identifier.

```
7.34.5.3 typedef void(* usbcallback_t) (USBDriver *usbp)
```

Type of an USB generic notification callback.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object triggering the callback
----	-------------	---

7.34.5.4 [typedef void\(* usbepcallback_t\)\(USBDriver *usbp, usbep_t ep\)](#)

Type of an USB endpoint callback.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object triggering the callback
in	<i>ep</i>	endpoint number

7.34.5.5 [typedef void\(* usbeventcb_t\)\(USBDriver *usbp, usbevent_t event\)](#)

Type of an USB event notification callback.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object triggering the callback
in	<i>event</i>	event type

7.34.5.6 [typedef bool\(* usbreqhandler_t\)\(USBDriver *usbp\)](#)

Type of a requests handler callback.

The request is encoded in the `usb_setup` buffer.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object triggering the callback
----	-------------	---

Returns

The request handling exit code.

Return values

<i>false</i>	Request not recognized by the handler.
<i>true</i>	Request handled.

7.34.5.7 [typedef const USBDescriptor*\(* usbgetdescriptor_t\)\(USBDriver *usbp, uint8_t dtype, uint8_t dindex, uint16_t lang\)](#)

Type of an USB descriptor-retrieving callback.

7.34.6 Enumeration Type Documentation**7.34.6.1 enum [usbstate_t](#)**

Type of a driver state machine possible states.

Enumerator

[USB_UNINIT](#) Not initialized.

- USB_STOP** Stopped.
- USB_READY** Ready, after bus reset.
- USB_SELECTED** Address assigned.
- USB_ACTIVE** Active, configuration selected.
- USB_SUSPENDED** Suspended, low power mode.

7.34.6.2 enum usbepstatus_t

Type of an endpoint status.

Enumerator

- EP_STATUS_DISABLED** Endpoint not active.
- EP_STATUS_STALLED** Endpoint opened but stalled.
- EP_STATUS_ACTIVE** Active endpoint.

7.34.6.3 enum usbep0state_t

Type of an endpoint zero state machine states.

Enumerator

- USB_EP0_WAITING_SETUP** Waiting for SETUP data.
- USB_EP0_TX** Transmitting.
- USB_EP0_WAITING_TX0** Waiting transmit 0.
- USB_EP0_WAITING_STS** Waiting status.
- USB_EP0_RX** Receiving.
- USB_EP0_SENDING_STS** Sending status.
- USB_EP0_ERROR** Error, EP0 stalled.

7.34.6.4 enum usbevent_t

Type of an enumeration of the possible USB events.

Enumerator

- USB_EVENT_RESET** Driver has been reset by host.
- USB_EVENT_ADDRESS** Address assigned.
- USB_EVENT_CONFIGURED** Configuration selected.
- USB_EVENT_SUSPEND** Entering suspend mode.
- USB_EVENT_WAKEUP** Leaving suspend mode.
- USB_EVENT_STALLED** Endpoint 0 error, stalled.

7.34.7 Function Documentation

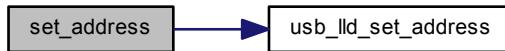
7.34.7.1 static void set_address (**USBDriver** * *usbp*) [static]

SET ADDRESS transaction callback.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
----	-------------	---

Here is the call graph for this function:



7.34.7.2 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	<i>usbp</i>	pointer to the USBDriver object
----	-------------	---

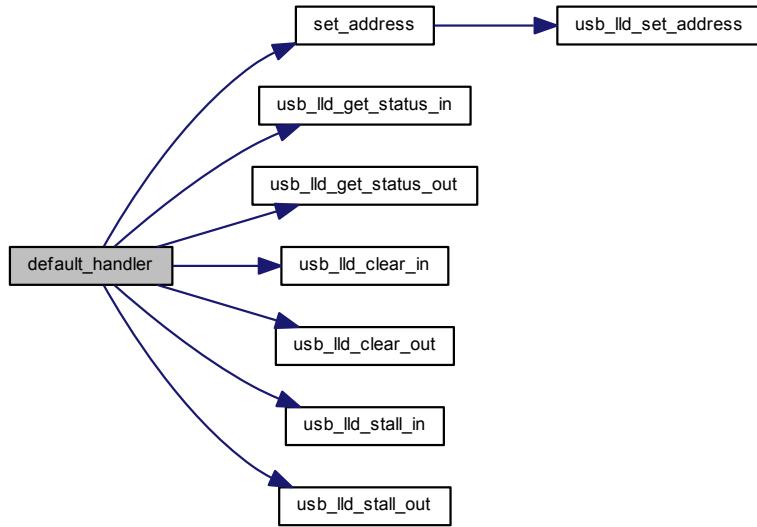
Returns

The request handling exit code.

Return values

<i>false</i>	Request not recognized by the handler or error.
<i>true</i>	Request handled.

Here is the call graph for this function:



7.34.7.3 void usblInit(void)

USB Driver initialization.

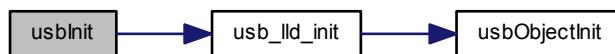
Note

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:



7.34.7.4 void usbObjectInit(USBDriver * usbp)

Initializes the standard part of a [USBDriver](#) structure.

Parameters

out	<i>usbp</i>	pointer to the USBDriver object
-----	-------------	---

Function Class:

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

7.34.7.5 void usbStart ([USBDriver](#) * *usbp*, const [USBConfig](#) * *config*)

Configures and activates the USB peripheral.

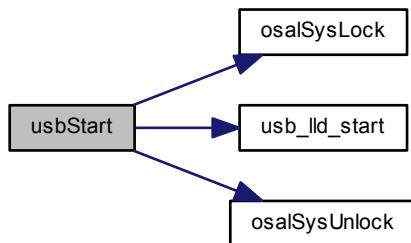
Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>config</i>	pointer to the USBConfig 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:

**7.34.7.6 void usbStop ([USBDriver](#) * *usbp*)**

Deactivates the USB peripheral.

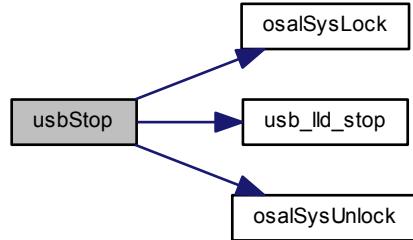
Parameters

in	<i>usbp</i>	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.

Here is the call graph for this function:



7.34.7.7 void usbInitEndpointl (**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

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number
in	<i>epcp</i>	the endpoint configuration

Function 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:



7.34.7.8 void usbDisableEndpointsl (**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	<i>usbp</i>	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:



7.34.7.9 void usbPrepareReceive ([USBDriver](#) * *usbp*, *usbep_t* *ep*, *uint8_t* * *buf*, *size_t* *n*)

Prepares for a receive transaction on an OUT endpoint.

Postcondition

The endpoint is ready for [usbStartReceiveI\(\)](#).

Note

This function can be called both in ISR and thread context.

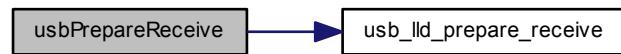
Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number
out	<i>buf</i>	buffer where to copy the received data
in	<i>n</i>	transaction size

Function Class:

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

Here is the call graph for this function:



7.34.7.10 void usbPrepareTransmit (**USBDriver** * *usbp*, **usbep_t** *ep*, const **uint8_t** * *buf*, **size_t** *n*)

Prepares for a transmit transaction on an IN endpoint.

Postcondition

The endpoint is ready for [usbStartTransmitI \(\)](#).

Note

This function can be called both in ISR and thread context.

The queue must contain at least the amount of data specified as transaction size.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number
in	<i>buf</i>	buffer where to fetch the data to be transmitted
in	<i>n</i>	transaction size

Function Class:

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

Here is the call graph for this function:



7.34.7.11 void usbPrepareQueuedReceive (**USBDriver** * *usbp*, **usbep_t** *ep*, **input_queue_t** * *iqp*, **size_t** *n*)

Prepares for a receive transaction on an OUT endpoint.

Postcondition

The endpoint is ready for [usbStartReceiveI \(\)](#).

Note

This function can be called both in ISR and thread context.

The queue must have enough free space to accommodate the specified transaction size rounded to the next packet size boundary. For example if the transaction size is 1 and the packet size is 64 then the queue must have space for at least 64 bytes.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number
in	<i>iqp</i>	input queue to be filled with incoming data
in	<i>n</i>	transaction size

Function Class:

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

Here is the call graph for this function:



7.34.7.12 void usbPrepareQueuedTransmit ([USBDriver](#) * *usbp*, [usbep_t](#) *ep*, [output_queue_t](#) * *oqp*, [size_t](#) *n*)

Prepares for a transmit transaction on an IN endpoint.

Postcondition

The endpoint is ready for [usbStartTransmitI\(\)](#).

Note

This function can be called both in ISR and thread context.

The transmit transaction size is equal to the data contained in the queue.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number
in	<i>oqp</i>	output queue to be fetched for outgoing data
in	<i>n</i>	transaction size

Function Class:

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

Here is the call graph for this function:



7.34.7.13 bool usbStartReceive(**USBDriver** * *usbp*, **usbep_t** *ep*)

Starts a receive transaction on an OUT endpoint.

Postcondition

The endpoint callback is invoked when the transfer has been completed.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Returns

The operation status.

Return values

<i>false</i>	Operation started successfully.
<i>true</i>	Endpoint busy, operation not started.

Function 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:



7.34.7.14 bool usbStartTransmit(**USBDriver** * *usbp*, **usbep_t** *ep*)

Starts a transmit transaction on an IN endpoint.

Postcondition

The endpoint callback is invoked when the transfer has been completed.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Returns

The operation status.

Return values

<i>false</i>	Operation started successfully.
<i>true</i>	Endpoint busy, operation not started.

Function 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:



7.34.7.15 bool usbStallReceive(USBDriver * usbp, usbep_t ep)

Stalls an OUT endpoint.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Returns

The operation status.

Return values

<i>false</i>	Endpoint stalled.
<i>true</i>	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:



7.34.7.16 bool usbStallTransmit(**USBDriver** * *usbp*, **usbep_t** *ep*)

Stalls an IN endpoint.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Returns

The operation status.

Return values

<i>false</i>	Endpoint stalled.
<i>true</i>	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:

**7.34.7.17 void _usb_reset(USBDriver * *usbp*)**

USB reset routine.

This function must be invoked when an USB bus reset condition is detected.

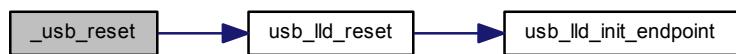
Parameters

in	<i>usbp</i>	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:



7.34.7.18 void _usb_suspend (**USBDriver * *usbp*)**

USB suspend routine.

This function must be invoked when an USB bus suspend condition is detected.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
----	-------------	--

Function Class:

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

7.34.7.19 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	<i>usbp</i>	pointer to the USBDriver object
----	-------------	--

Function Class:

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

7.34.7.20 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.

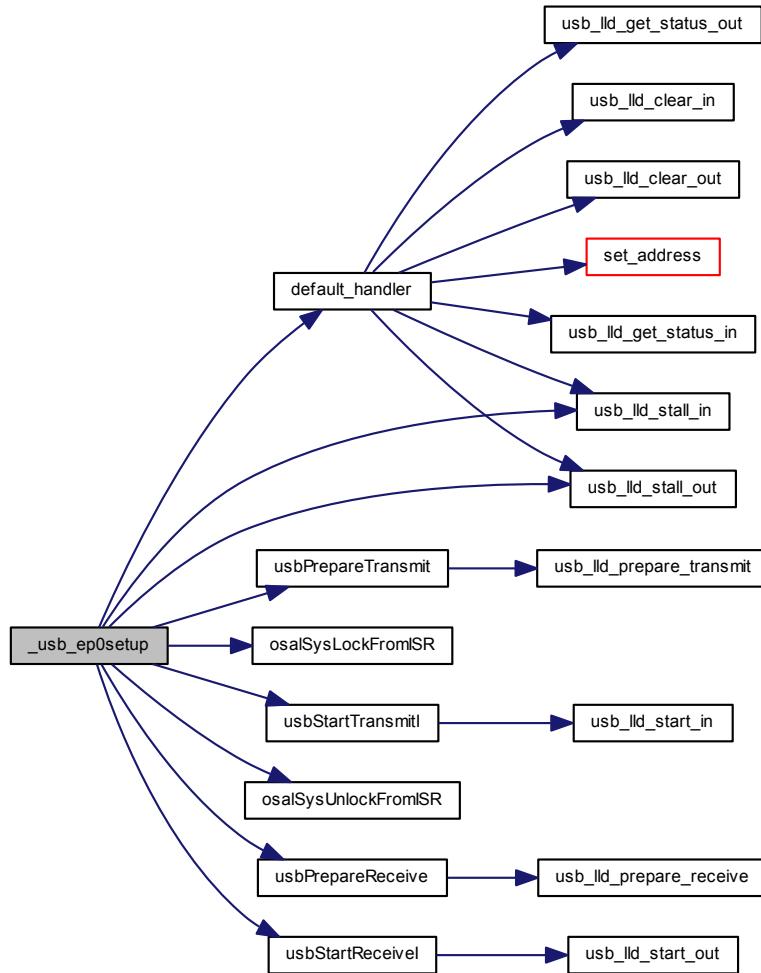
Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number, always zero

Function Class:

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

Here is the call graph for this function:



7.34.7.21 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.

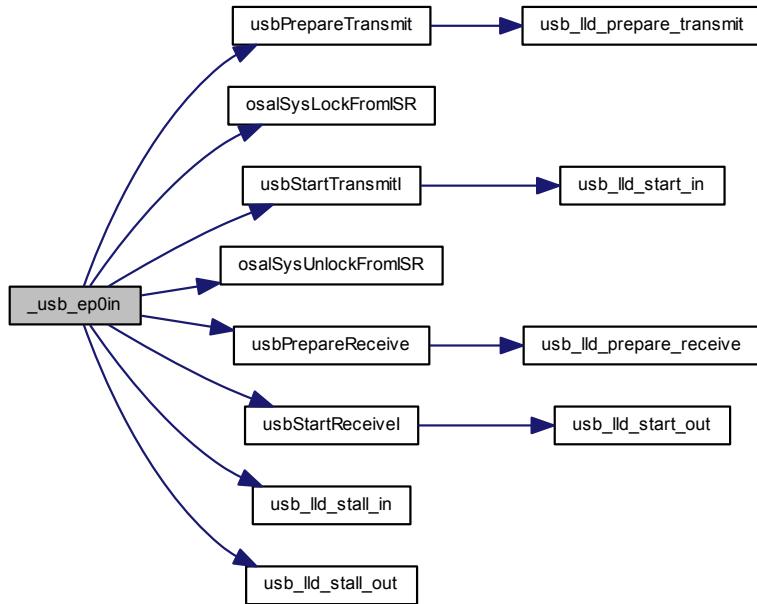
Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number, always zero

Function Class:

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

Here is the call graph for this function:



7.34.7.22 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.

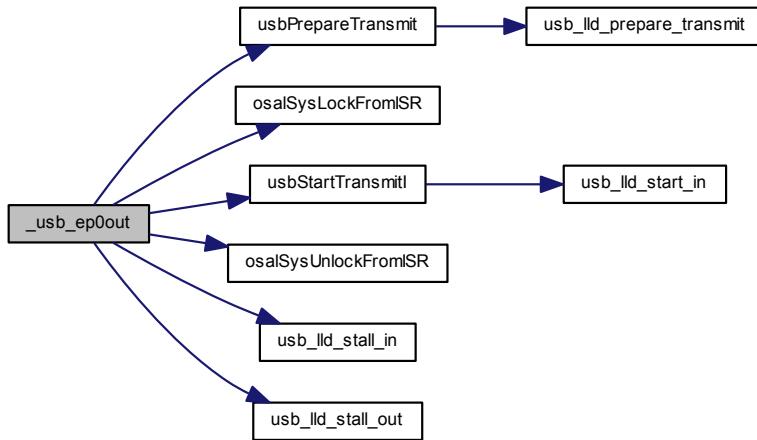
Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number, always zero

Function Class:

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

Here is the call graph for this function:



7.34.7.23 void `usb_lld_init` (void)

Low level USB driver initialization.

Function Class:

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

Here is the call graph for this function:



7.34.7.24 void `usb_lld_start` (`USBDriver * usbp`)

Configures and activates the USB peripheral.

Parameters

in	<code>usbp</code>	pointer to the <code>USBDriver</code> object
----	-------------------	--

Function Class:

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

7.34.7.25 void `usb_lld_stop(USBDriver * usb)`

Deactivates the USB peripheral.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
----	-------------	---

Function Class:

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

7.34.7.26 void usb_lld_reset (USBDriver * *usbp*)

USB low level reset routine.

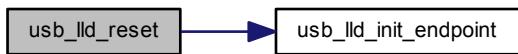
Parameters

in	<i>usbp</i>	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:

**7.34.7.27 void usb_lld_set_address (USBDriver * *usbp*)**

Sets the USB address.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
----	-------------	---

Function Class:

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

7.34.7.28 void usb_lld_init_endpoint (USBDriver * *usbp*, usbep_t *ep*)

Enables an endpoint.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

7.34.7.29 void **usb_lld_disable_endpoints**(**USBDriver** * *usbp*)

Disables all the active endpoints except the endpoint zero.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
----	-------------	---

Function Class:

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

7.34.7.30 usbepstatus_t usb_lld_get_status_out (USBDriver * *usbp*, usbep_t *ep*)

Returns the status of an OUT endpoint.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Returns

The endpoint status.

Return values

<i>EP_STATUS_DISABLED</i>	The endpoint is not active.
<i>EP_STATUS_STALLED</i>	The endpoint is stalled.
<i>EP_STATUS_ACTIVE</i>	The endpoint is active.

Function Class:

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

7.34.7.31 usbepstatus_t usb_lld_get_status_in (USBDriver * *usbp*, usbep_t *ep*)

Returns the status of an IN endpoint.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Returns

The endpoint status.

Return values

<i>EP_STATUS_DISABLED</i>	The endpoint is not active.
<i>EP_STATUS_STALLED</i>	The endpoint is stalled.
<i>EP_STATUS_ACTIVE</i>	The endpoint is active.

Function Class:

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

7.34.7.32 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

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number
out	<i>buf</i>	buffer where to copy the packet data

Function Class:

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

7.34.7.33 void usb_lld_prepare_receive ([USBDriver](#) * *usbp*, [usbep_t](#) *ep*)

Prepares for a receive operation.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

7.34.7.34 void usb_lld_prepare_transmit ([USBDriver](#) * *usbp*, [usbep_t](#) *ep*)

Prepares for a transmit operation.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

7.34.7.35 void usb_lld_start_out ([USBDriver](#) * *usbp*, [usbep_t](#) *ep*)

Starts a receive operation on an OUT endpoint.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

7.34.7.36 void usb_lld_start_in ([USBDriver](#) * *usbp*, [usbep_t](#) *ep*)

Starts a transmit operation on an IN endpoint.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

7.34.7.37 void usb_lld_stall_out(USBDriver * *usbp*, usbep_t *ep*)

Brings an OUT endpoint in the stalled state.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

7.34.7.38 void usb_lld_stall_in(USBDriver * *usbp*, usbep_t *ep*)

Brings an IN endpoint in the stalled state.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

7.34.7.39 void usb_lld_clear_out(USBDriver * *usbp*, usbep_t *ep*)

Brings an OUT endpoint in the active state.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

7.34.7.40 void usb_lld_clear_in(USBDriver * *usbp*, usbep_t *ep*)

Brings an IN endpoint in the active state.

Parameters

in	<i>usbp</i>	pointer to the <code>USBDriver</code> object
in	<i>ep</i>	endpoint number

Function Class:

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

7.34.8 Variable Documentation**7.34.8.1 `USBDriver USBD1`**

USB1 driver identifier.

7.34.8.2 `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.

7.34.8.3 `USBInEndpointState { ... } in`

IN EP0 state.

7.34.8.4 `USBOutEndpointState { ... } out`

OUT EP0 state.

7.34.8.5 `const USBEndpointConfig ep0config [static]`**Initial value:**

```
= {
    USB_EP_MODE_TYPE_CTRL,
    _usb_ep0setup,
    _usb_ep0in,
    _usb_ep0out,
    0x40,
    0x40,
    &ep0_state.in,
    &ep0_state.out
}
```

EP0 initialization structure.

Chapter 8

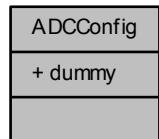
Data Structure Documentation

8.1 ADCConfig Struct Reference

Driver configuration structure.

```
#include <adc_lld.h>
```

Collaboration diagram for ADCConfig:



8.1.1 Detailed Description

Driver configuration structure.

Note

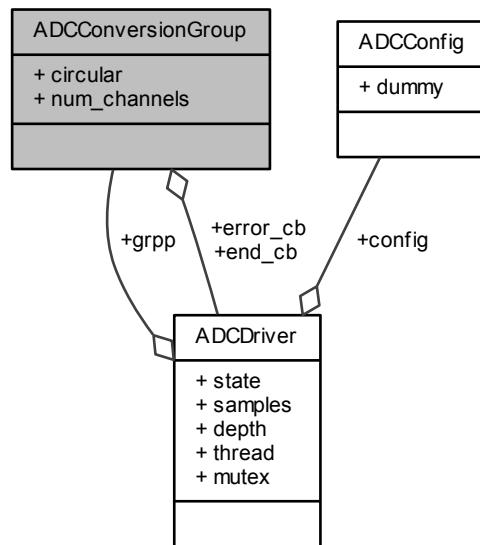
It could be empty on some architectures.

8.2 ADCConversionGroup Struct Reference

Conversion group configuration structure.

```
#include <adc_lld.h>
```

Collaboration diagram for ADCConversionGroup:



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`.

8.2.1 Detailed Description

Conversion group configuration structure.

This implementation-dependent structure describes a conversion operation.

Note

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

8.2.2 Field Documentation

8.2.2.1 bool ADCConversionGroup::circular

Enables the circular buffer mode for the group.

8.2.2.2 adc_channels_num_t ADCConversionGroup::num_channels

Number of the analog channels belonging to the conversion group.

8.2.2.3 adccallback_t ADCConversionGroup::end_cb

Callback function associated to the group or NULL.

8.2.2.4 adcerrorcallback_t ADCConversionGroup::error_cb

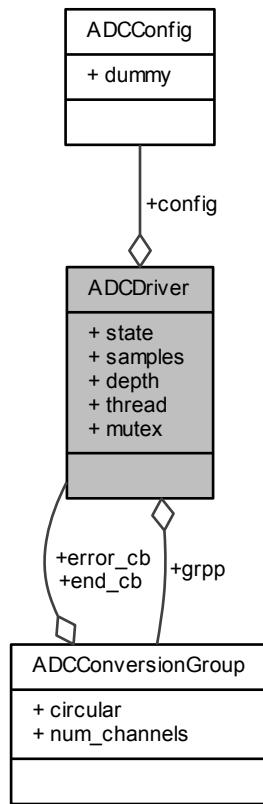
Error callback or NULL.

8.3 ADCDriver Struct Reference

Structure representing an ADC driver.

```
#include <adc_lld.h>
```

Collaboration diagram for ADCDriver:



Data Fields

- `adcstate_t state`
Driver state.
- `const ADCCConfig * config`
Current configuration data.
- `adosample_t * samples`
Current samples buffer pointer or `NULL`.
- `size_t depth`
Current samples buffer depth or 0.
- `const ADCCConversionGroup * grpp`
Current conversion group pointer or `NULL`.
- `thread_reference_t thread`
Waiting thread.
- `mutex_t mutex`
Mutex protecting the peripheral.

8.3.1 Detailed Description

Structure representing an ADC driver.

8.3.2 Field Documentation

8.3.2.1 `adcstate_t` ADCDriver::state

Driver state.

8.3.2.2 `const ADCConfig*` ADCDriver::config

Current configuration data.

8.3.2.3 `adcsample_t*` ADCDriver::samples

Current samples buffer pointer or NULL.

8.3.2.4 `size_t` ADCDriver::depth

Current samples buffer depth or 0.

8.3.2.5 `const ADCConversionGroup*` ADCDriver::grpp

Current conversion group pointer or NULL.

8.3.2.6 `thread_reference_t` ADCDriver::thread

Waiting thread.

8.3.2.7 `mutex_t` ADCDriver::mutex

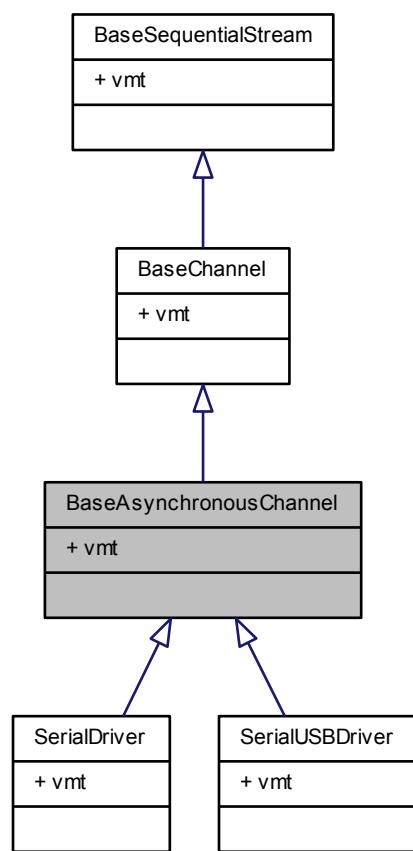
Mutex protecting the peripheral.

8.4 BaseAsynchronousChannel Struct Reference

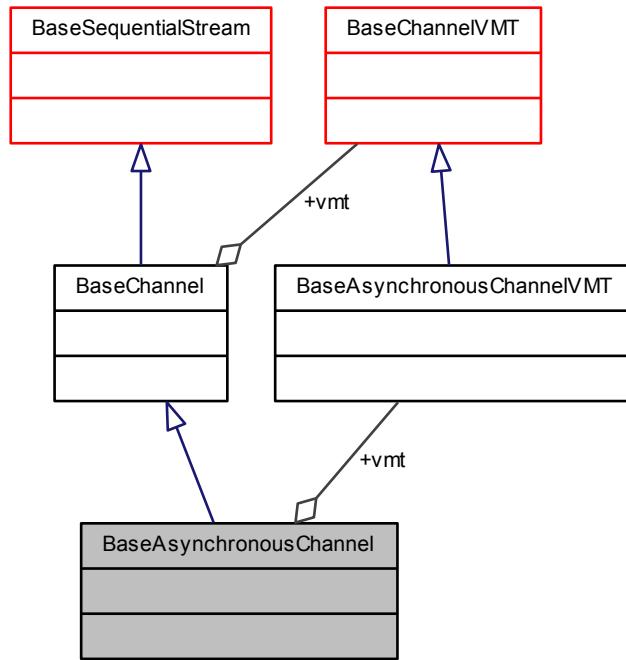
Base asynchronous channel class.

```
#include <hal_channels.h>
```

Inheritance diagram for BaseAsynchronousChannel:



Collaboration diagram for BaseAsynchronousChannel:



Data Fields

- const struct [BaseAsynchronousChannelVMT](#) * vmt

Virtual Methods Table.

8.4.1 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.

8.4.2 Field Documentation

8.4.2.1 const struct [BaseAsynchronousChannelVMT](#)* [BaseAsynchronousChannel::vmt](#)

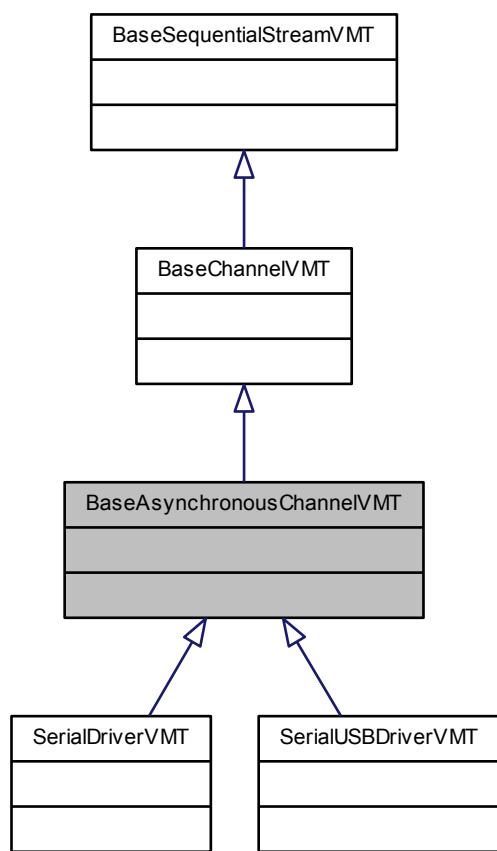
Virtual Methods Table.

8.5 BaseAsynchronousChannelVMT Struct Reference

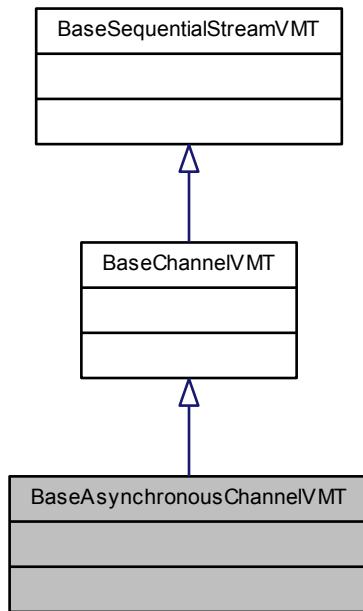
[BaseAsynchronousChannel](#) virtual methods table.

```
#include <hal_channels.h>
```

Inheritance diagram for BaseAsynchronousChannelVMT:



Collaboration diagram for BaseAsynchronousChannelVMT:



8.5.1 Detailed Description

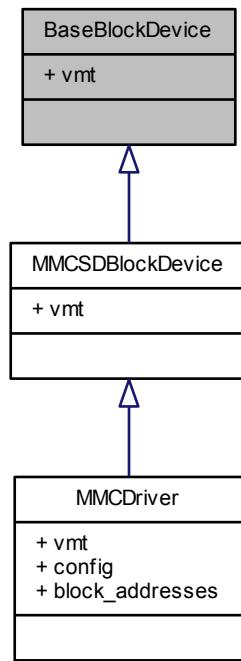
[BaseAsynchronousChannel](#) virtual methods table.

8.6 BaseBlockDevice Struct Reference

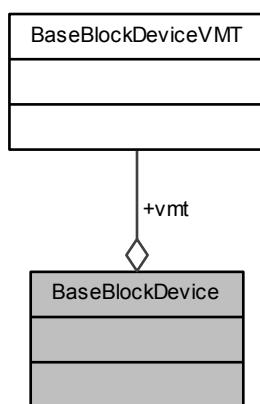
Base block device class.

```
#include <hal_ioblock.h>
```

Inheritance diagram for BaseBlockDevice:



Collaboration diagram for BaseBlockDevice:



Data Fields

- const struct `BaseBlockDeviceVMT` * `vmt`

Virtual Methods Table.

8.6.1 Detailed Description

Base block device class.

This class represents a generic, block-accessible, device.

8.6.2 Field Documentation

8.6.2.1 const struct BaseBlockDeviceVMT* BaseBlockDevice::vmt

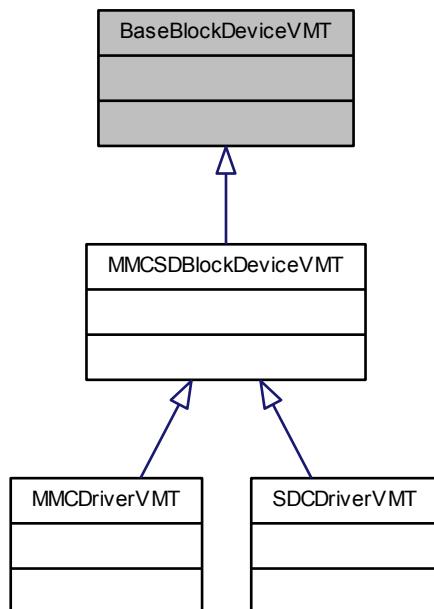
Virtual Methods Table.

8.7 BaseBlockDeviceVMT Struct Reference

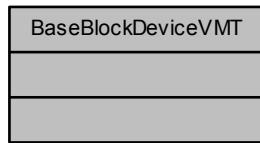
[BaseBlockDevice](#) virtual methods table.

```
#include <hal_ioblock.h>
```

Inheritance diagram for BaseBlockDeviceVMT:



Collaboration diagram for BaseBlockDeviceVMT:



8.7.1 Detailed Description

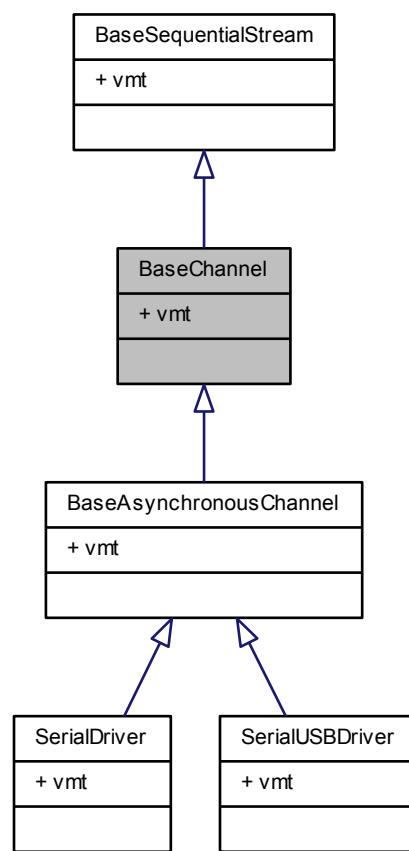
`BaseBlockDevice` virtual methods table.

8.8 BaseChannel Struct Reference

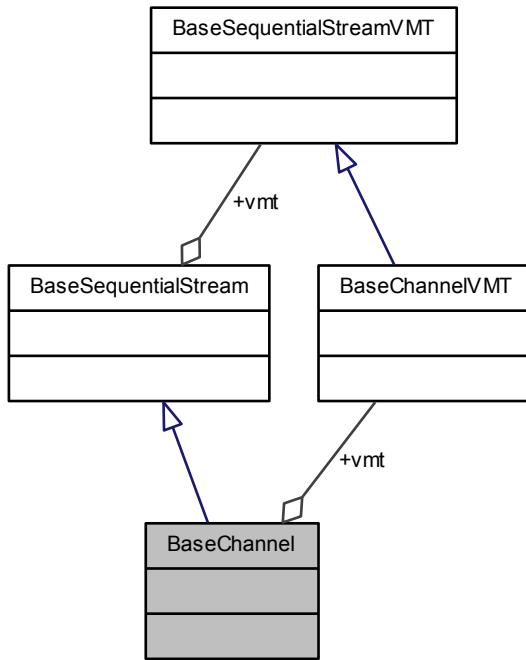
Base channel class.

```
#include <hal_channels.h>
```

Inheritance diagram for BaseChannel:



Collaboration diagram for BaseChannel:



Data Fields

- const struct `BaseChannelVMT` * `vmt`

Virtual Methods Table.

8.8.1 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.

8.8.2 Field Documentation

8.8.2.1 const struct `BaseChannelVMT`* `BaseChannel::vmt`

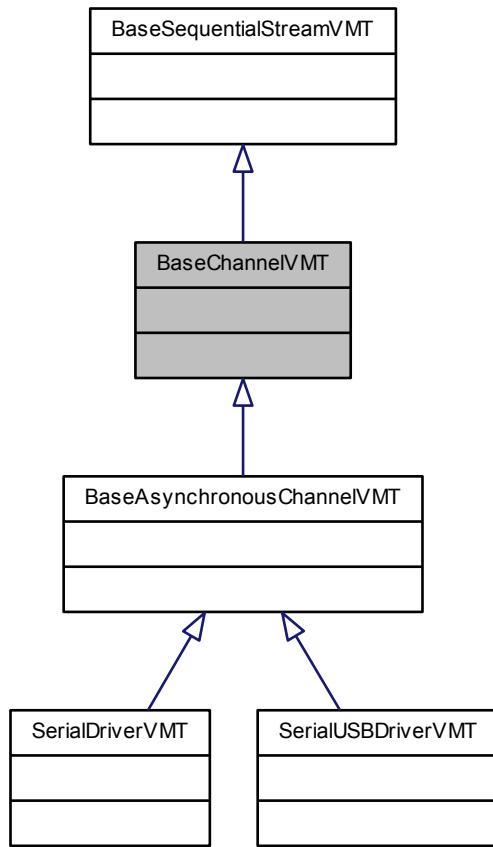
Virtual Methods Table.

8.9 BaseChannelVMT Struct Reference

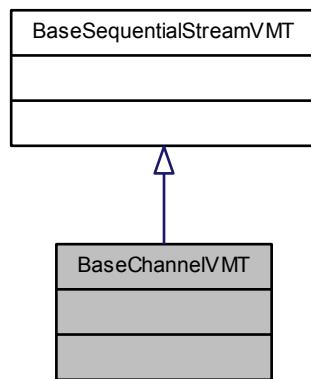
`BaseChannel` virtual methods table.

```
#include <hal_channels.h>
```

Inheritance diagram for BaseChannelVMT:



Collaboration diagram for BaseChannelVMT:



8.9.1 Detailed Description

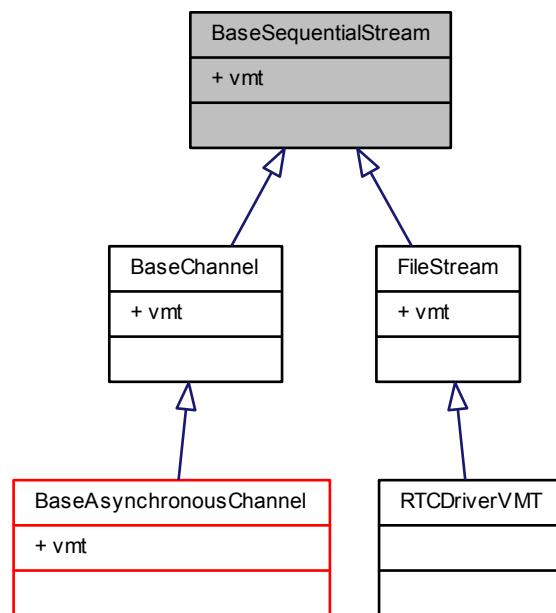
[BaseChannel](#) virtual methods table.

8.10 BaseSequentialStream Struct Reference

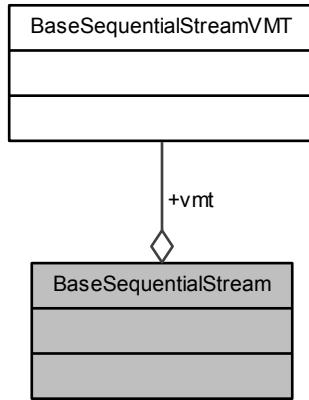
Base stream class.

```
#include <hal_streams.h>
```

Inheritance diagram for BaseSequentialStream:



Collaboration diagram for BaseSequentialStream:



Data Fields

- const struct [BaseSequentialStreamVMT](#) * vmt

Virtual Methods Table.

8.10.1 Detailed Description

Base stream class.

This class represents a generic blocking unbuffered sequential data stream.

8.10.2 Field Documentation

8.10.2.1 const struct [BaseSequentialStreamVMT](#)* [BaseSequentialStream::vmt](#)

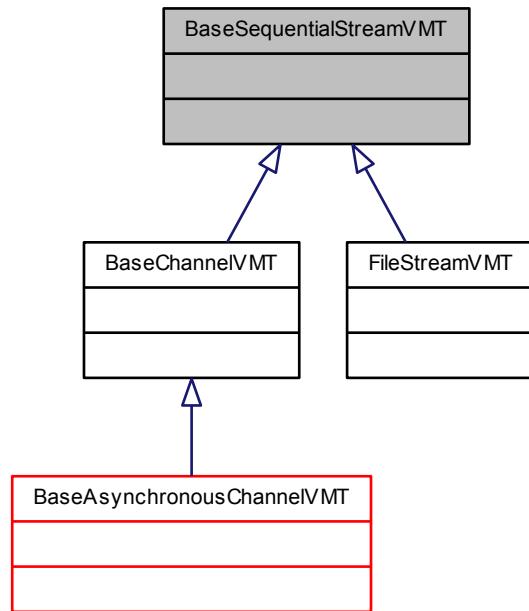
Virtual Methods Table.

8.11 BaseSequentialStreamVMT Struct Reference

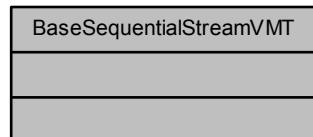
[BaseSequentialStream](#) virtual methods table.

```
#include <hal_streams.h>
```

Inheritance diagram for BaseSequentialStreamVMT:



Collaboration diagram for BaseSequentialStreamVMT:



8.11.1 Detailed Description

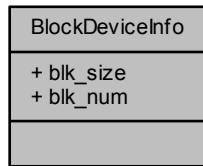
[BaseSequentialStream](#) virtual methods table.

8.12 BlockDeviceInfo Struct Reference

Block device info.

```
#include <hal_ioblock.h>
```

Collaboration diagram for BlockDeviceInfo:



Data Fields

- `uint32_t blk_size`
Block size in bytes.
- `uint32_t blk_num`
Total number of blocks.

8.12.1 Detailed Description

Block device info.

8.12.2 Field Documentation

8.12.2.1 `uint32_t BlockDeviceInfo::blk_size`

Block size in bytes.

8.12.2.2 `uint32_t BlockDeviceInfo::blk_num`

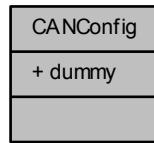
Total number of blocks.

8.13 CANConfig Struct Reference

Driver configuration structure.

```
#include <can_lld.h>
```

Collaboration diagram for CANConfig:



8.13.1 Detailed Description

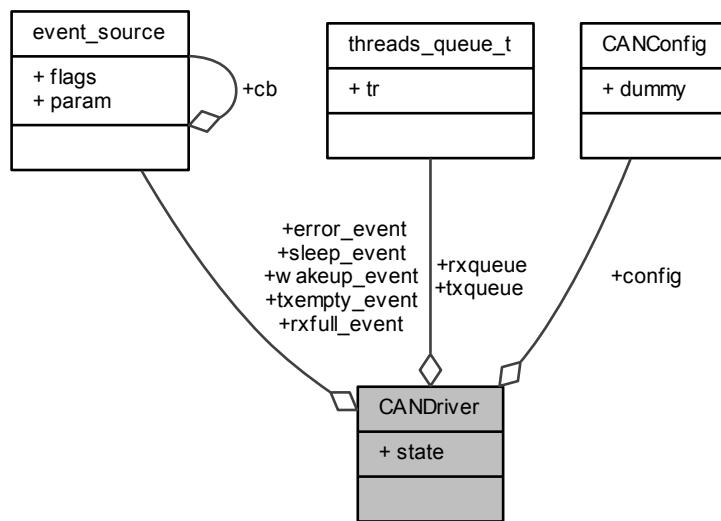
Driver configuration structure.

8.14 CANDriver Struct Reference

Structure representing an CAN driver.

```
#include <can_lld.h>
```

Collaboration diagram for CANDriver:



Data Fields

- `canstate_t state`
Driver state.

- `const CANConfig * config`
Current configuration data.
- `threads_queue_t txqueue`
Transmission threads queue.
- `threads_queue_t rxqueue`
Receive threads queue.
- `event_source_t rxfull_event`
One or more frames become available.
- `event_source_t txempty_event`
One or more transmission mailbox become available.
- `event_source_t error_event`
A CAN bus error happened.
- `event_source_t sleep_event`
Entering sleep state event.
- `event_source_t wakeup_event`
Exiting sleep state event.

8.14.1 Detailed Description

Structure representing an CAN driver.

8.14.2 Field Documentation

8.14.2.1 canstate_t CANDriver::state

Driver state.

8.14.2.2 const CANConfig* CANDriver::config

Current configuration data.

8.14.2.3 threads_queue_t CANDriver::txqueue

Transmission threads queue.

8.14.2.4 threads_queue_t CANDriver::rxqueue

Receive threads queue.

8.14.2.5 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.

8.14.2.6 event_source_t CANDriver::txempty_event

One or more transmission mailbox become available.

Note

The flags associated to the listeners will indicate which transmit mailboxes become empty.

8.14.2.7 event_source_t CANDriver::error_event

A CAN bus error happened.

Note

The flags associated to the listeners will indicate the error(s) that have occurred.

8.14.2.8 event_source_t CANDriver::sleep_event

Entering sleep state event.

8.14.2.9 event_source_t CANDriver::wakeup_event

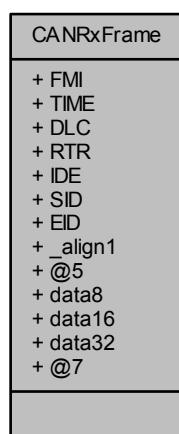
Exiting sleep state event.

8.15 CANRxFrame Struct Reference

CAN received frame.

```
#include <can_lld.h>
```

Collaboration diagram for CANRxFrame:



Data Fields

- `uint8_t FMI`
Filter id.
- `uint16_t TIME`
Time stamp.
- `uint8_t DLC:4`
Data length.
- `uint8_t RTR:1`
Frame type.
- `uint8_t IDE:1`
Identifier type.
- `uint32_t SID:11`
Standard identifier.
- `uint32_t EID:29`
Extended identifier.
- `uint8_t data8 [8]`
Frame data.
- `uint16_t data16 [4]`
Frame data.
- `uint32_t data32 [2]`
Frame data.

8.15.1 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.

8.15.2 Field Documentation

8.15.2.1 `uint8_t CANRxFrame::FMI`

Filter id.

8.15.2.2 `uint16_t CANRxFrame::TIME`

Time stamp.

8.15.2.3 `uint8_t CANRxFrame::DLC`

Data length.

8.15.2.4 `uint8_t CANRxFrame::RTR`

Frame type.

8.15.2.5 uint8_t CANRxFrame::IDE

Identifier type.

8.15.2.6 uint32_t CANRxFrame::SID

Standard identifier.

8.15.2.7 uint32_t CANRxFrame::EID

Extended identifier.

8.15.2.8 uint8_t CANRxFrame::data8[8]

Frame data.

8.15.2.9 uint16_t CANRxFrame::data16[4]

Frame data.

8.15.2.10 uint32_t CANRxFrame::data32[2]

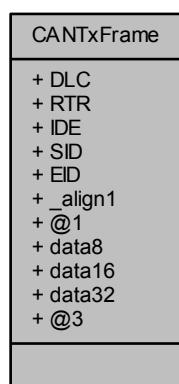
Frame data.

8.16 CANTxFrame Struct Reference

CAN transmission frame.

```
#include <can_ll.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]`
Frame data.
- `uint16_t data16 [4]`
Frame data.
- `uint32_t data32 [2]`
Frame data.

8.16.1 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.

8.16.2 Field Documentation

8.16.2.1 `uint8_t CANTxFrame::DLC`

Data length.

8.16.2.2 `uint8_t CANTxFrame::RTR`

Frame type.

8.16.2.3 `uint8_t CANTxFrame::IDE`

Identifier type.

8.16.2.4 `uint32_t CANTxFrame::SID`

Standard identifier.

8.16.2.5 `uint32_t CANTxFrame::EID`

Extended identifier.

8.16.2.6 uint8_t CANTxFrame::data8[8]

Frame data.

8.16.2.7 uint16_t CANTxFrame::data16[4]

Frame data.

8.16.2.8 uint32_t CANTxFrame::data32[2]

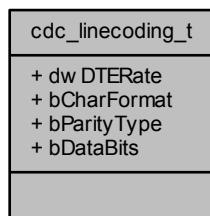
Frame data.

8.17 cdc_linecoding_t Struct Reference

Type of Line Coding structure.

```
#include <serial_usb.h>
```

Collaboration diagram for cdc_linecoding_t:



8.17.1 Detailed Description

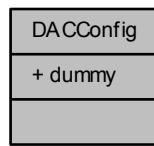
Type of Line Coding structure.

8.18 DACConfig Struct Reference

Driver configuration structure.

```
#include <dac_lld.h>
```

Collaboration diagram for DACConfig:



8.18.1 Detailed Description

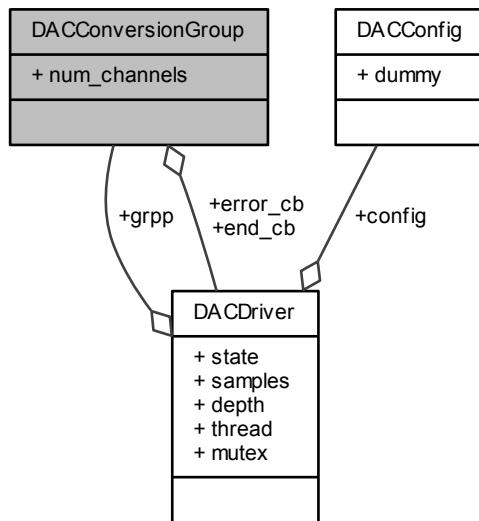
Driver configuration structure.

8.19 DACConversionGroup Struct Reference

DAC Conversion group structure.

```
#include <dac_lld.h>
```

Collaboration diagram for DACConversionGroup:



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.

8.19.1 Detailed Description

DAC Conversion group structure.

8.19.2 Field Documentation

8.19.2.1 `uint32_t DACConversionGroup::num_channels`

Number of DAC channels.

8.19.2.2 `daccallback_t DACConversionGroup::end_cb`

Operation complete callback or NULL.

8.19.2.3 `dacerrorcallback_t DACConversionGroup::error_cb`

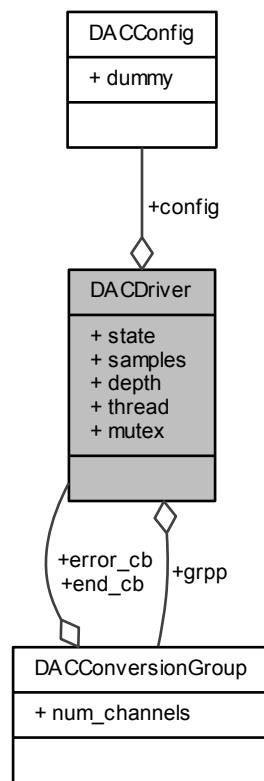
Error handling callback or NULL.

8.20 DACDriver Struct Reference

Structure representing a DAC driver.

```
#include <dac_lld.h>
```

Collaboration diagram for DACDriver:



Data Fields

- `dacstate_t state`
Driver state.
- `const DACConversionGroup * grpp`
Conversion group.
- `const dacsample_t * samples`
Samples buffer pointer.
- `uint16_t depth`
Samples buffer size.
- `const DACConfig * config`
Current configuration data.
- `thread_reference_t thread`
Waiting thread.
- `mutex_t mutex`
Mutex protecting the bus.

8.20.1 Detailed Description

Structure representing a DAC driver.

8.20.2 Field Documentation

8.20.2.1 `dacstate_t` DACDriver::state

Driver state.

8.20.2.2 `const DACConversionGroup*` DACDriver::grpp

Conversion group.

8.20.2.3 `const dacsample_t*` DACDriver::samples

Samples buffer pointer.

8.20.2.4 `uint16_t` DACDriver::depth

Samples buffer size.

8.20.2.5 `const DACConfig*` DACDriver::config

Current configuration data.

8.20.2.6 `thread_reference_t` DACDriver::thread

Waiting thread.

8.20.2.7 `mutex_t` DACDriver::mutex

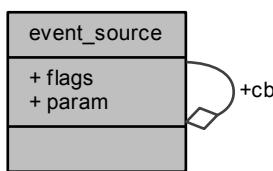
Mutex protecting the bus.

8.21 event_source Struct Reference

Events source object.

```
#include <osal.h>
```

Collaboration diagram for `event_source`:



Data Fields

- volatile `eventflags_t` `flags`

Stored event flags.

- `eventcallback_t` `cb`

Event source callback.

- `void *` `param`

User defined field.

8.21.1 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.

8.21.2 Field Documentation

8.21.2.1 volatile `eventflags_t` `event_source::flags`

Stored event flags.

8.21.2.2 `eventcallback_t` `event_source::cb`

Event source callback.

8.21.2.3 `void*` `event_source::param`

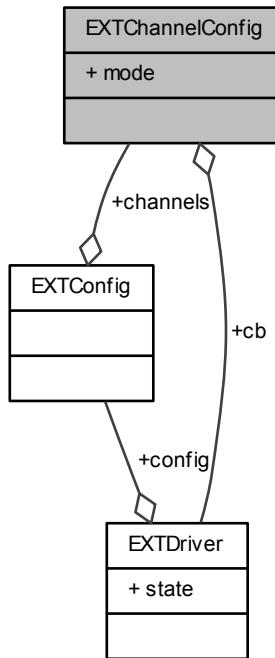
User defined field.

8.22 EXTChannelConfig Struct Reference

Channel configuration structure.

```
#include <ext_lld.h>
```

Collaboration diagram for EXTChannelConfig:



Data Fields

- `uint32_t mode`
Channel mode.
- `extcallback_t cb`
Channel callback.

8.22.1 Detailed Description

Channel configuration structure.

8.22.2 Field Documentation

8.22.2.1 `uint32_t EXTChannelConfig::mode`

Channel mode.

8.22.2.2 `extcallback_t EXTChannelConfig::cb`

Channel callback.

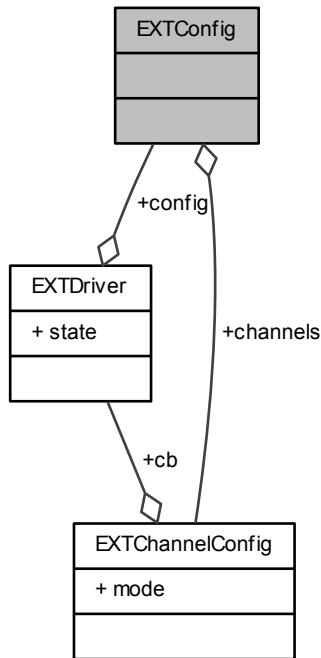
In the STM32 implementation a `NULL` callback pointer is valid and configures the channel as an event sources instead of an interrupt source.

8.23 EXTConfig Struct Reference

Driver configuration structure.

```
#include <ext_lld.h>
```

Collaboration diagram for EXTConfig:



Data Fields

- [EXTChannelConfig channels \[EXT_MAX_CHANNELS\]](#)

Channel configurations.

8.23.1 Detailed Description

Driver configuration structure.

Note

It could be empty on some architectures.

8.23.2 Field Documentation

8.23.2.1 EXTChannelConfig EXTConfig::channels[EXT_MAX_CHANNELS]

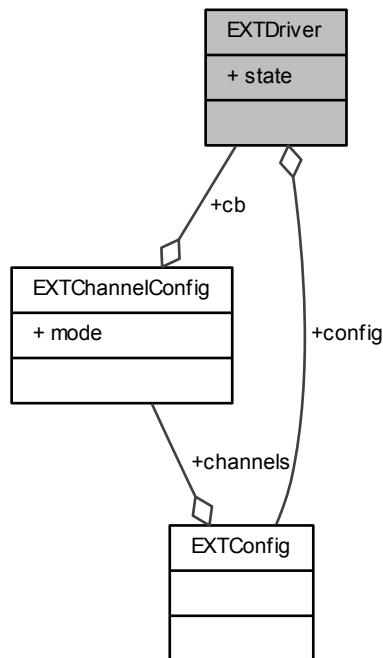
Channel configurations.

8.24 EXTDriver Struct Reference

Structure representing an EXT driver.

```
#include <ext_lld.h>
```

Collaboration diagram for EXTDriver:



Data Fields

- `extstate_t state`
Driver state.
- `const EXTConfig * config`
Current configuration data.

8.24.1 Detailed Description

Structure representing an EXT driver.

8.24.2 Field Documentation

8.24.2.1 `extstate_t EXTDriver::state`

Driver state.

8.24.2.2 const EXTConfig* EXTDriver::config

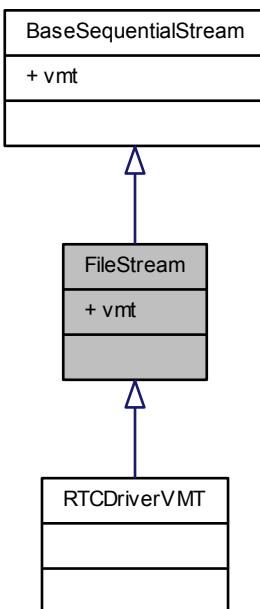
Current configuration data.

8.25 FileStream Struct Reference

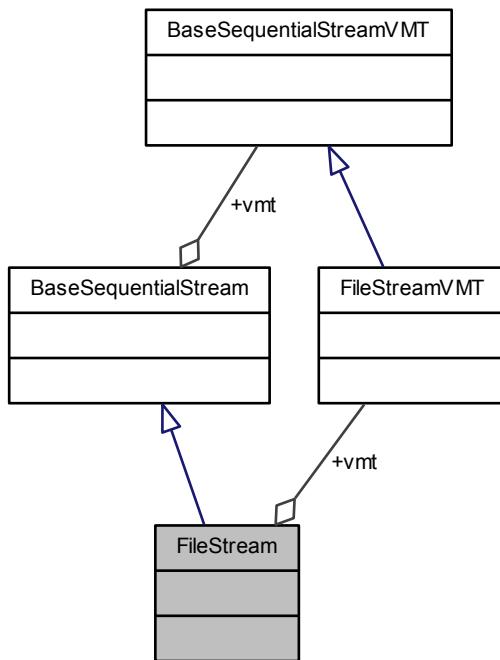
Base file stream class.

```
#include <hal_files.h>
```

Inheritance diagram for FileStream:



Collaboration diagram for FileStream:



Data Fields

- const struct `FileStreamVMT` * `vmt`

Virtual Methods Table.

8.25.1 Detailed Description

Base file stream class.

This class represents a generic file data stream.

8.25.2 Field Documentation

8.25.2.1 const struct `FileStreamVMT`* `FileStream::vmt`

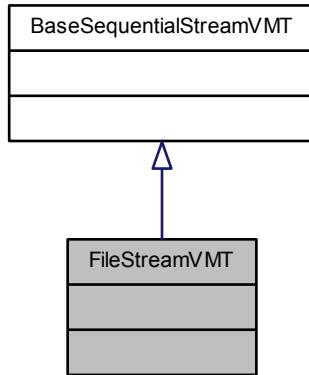
Virtual Methods Table.

8.26 FileStreamVMT Struct Reference

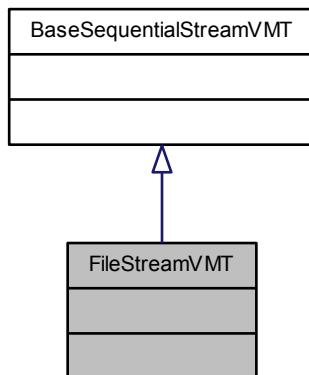
`FileStream` virtual methods table.

```
#include <hal_files.h>
```

Inheritance diagram for FileStreamVMT:



Collaboration diagram for FileStreamVMT:



8.26.1 Detailed Description

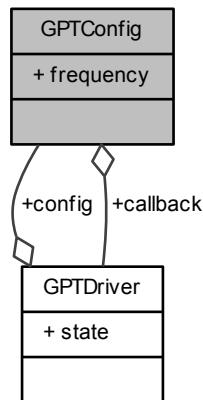
[FileStream](#) virtual methods table.

8.27 GPTConfig Struct Reference

Driver configuration structure.

```
#include <gpt_lld.h>
```

Collaboration diagram for GPTConfig:



Data Fields

- `gptfreq_t frequency`
Timer clock in Hz.
- `gptcallback_t callback`
Timer callback pointer.

8.27.1 Detailed Description

Driver configuration structure.

Note

It could be empty on some architectures.

8.27.2 Field Documentation

8.27.2.1 `gptfreq_t GPTConfig::frequency`

Timer clock in Hz.

Note

The low level can use assertions in order to catch invalid frequency specifications.

8.27.2.2 `gptcallback_t GPTConfig::callback`

Timer callback pointer.

Note

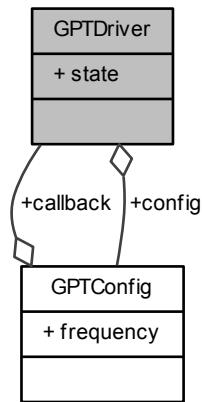
This callback is invoked on GPT counter events.

8.28 GPTDriver Struct Reference

Structure representing a GPT driver.

```
#include <gpt_lld.h>
```

Collaboration diagram for GPTDriver:



Data Fields

- [gptstate_t state](#)
Driver state.
- [const GPTConfig * config](#)
Current configuration data.

8.28.1 Detailed Description

Structure representing a GPT driver.

8.28.2 Field Documentation

8.28.2.1 gptstate_t GPTDriver::state

Driver state.

8.28.2.2 const GPTConfig* GPTDriver::config

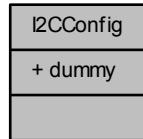
Current configuration data.

8.29 I2CConfig Struct Reference

Type of I2C driver configuration structure.

```
#include <i2c_lld.h>
```

Collaboration diagram for I2CConfig:



8.29.1 Detailed Description

Type of I2C driver configuration structure.

Note

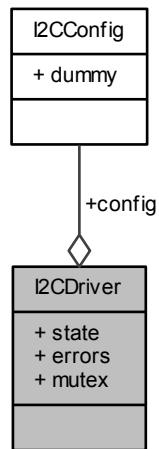
Implementations may extend this structure to contain more, architecture dependent, fields.

8.30 I2CDriver Struct Reference

Structure representing an I2C driver.

```
#include <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.

8.30.1 Detailed Description

Structure representing an I2C driver.

8.30.2 Field Documentation

8.30.2.1 i2cstate_t I2CDriver::state

Driver state.

8.30.2.2 const I2CConfig* I2CDriver::config

Current configuration data.

8.30.2.3 i2cflags_t I2CDriver::errors

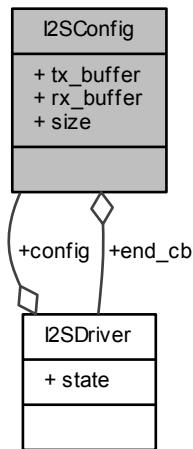
Error flags.

8.31 I2SConfig Struct Reference

Driver configuration structure.

```
#include <i2s_ll.h>
```

Collaboration diagram for I2SConfig:



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.

8.31.1 Detailed Description

Driver configuration structure.

Note

It could be empty on some architectures.

8.31.2 Field Documentation

8.31.2.1 `const void* I2SConfig::tx_buffer`

Transmission buffer pointer.

Note

Can be `NULL` if TX is not required.

8.31.2.2 void* I2SConfig::rx_buffer

Receive buffer pointer.

Note

Can be NULL if RX is not required.

8.31.2.3 size_t I2SConfig::size

TX and RX buffers size as number of samples.

8.31.2.4 i2scallback_t I2SConfig::end_cb

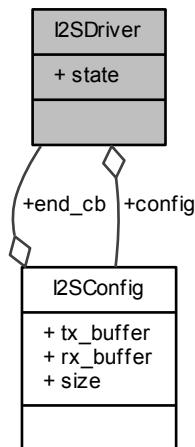
Callback function called during streaming.

8.32 I2SDriver Struct Reference

Structure representing an I2S driver.

```
#include <i2s_llld.h>
```

Collaboration diagram for I2SDriver:



Data Fields

- [i2sstate_t state](#)
Driver state.
- const [I2SConfig * config](#)
Current configuration data.

8.32.1 Detailed Description

Structure representing an I2S driver.

8.32.2 Field Documentation

8.32.2.1 i2sstate_t I2SDriver::state

Driver state.

8.32.2.2 const I2SConfig* I2SDriver::config

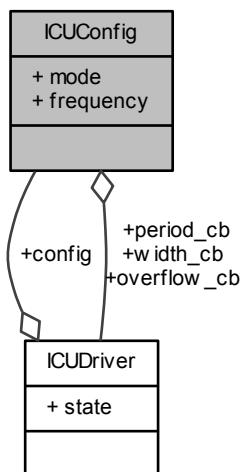
Current configuration data.

8.33 ICUConfig Struct Reference

Driver configuration structure.

```
#include <icu_lld.h>
```

Collaboration diagram for ICUConfig:



Data Fields

- [icemode_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.

8.33.1 Detailed Description

Driver configuration structure.

Note

It could be empty on some architectures.

8.33.2 Field Documentation

8.33.2.1 `icemode_t ICUConfig::mode`

Driver mode.

8.33.2.2 `icufreq_t ICUConfig::frequency`

Timer clock in Hz.

Note

The low level can use assertions in order to catch invalid frequency specifications.

8.33.2.3 `icucallback_t ICUConfig::width_cb`

Callback for pulse width measurement.

8.33.2.4 `icucallback_t ICUConfig::period_cb`

Callback for cycle period measurement.

8.33.2.5 `icucallback_t ICUConfig::overflow_cb`

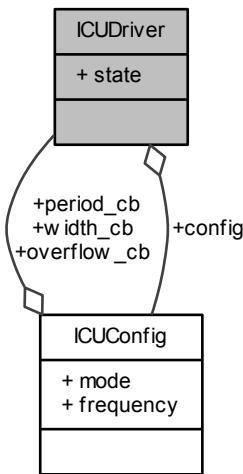
Callback for timer overflow.

8.34 ICUDriver Struct Reference

Structure representing an ICU driver.

```
#include <icu_lld.h>
```

Collaboration diagram for ICUDriver:



Data Fields

- `icustate_t state`
Driver state.
- `const ICUConfig * config`
Current configuration data.

8.34.1 Detailed Description

Structure representing an ICU driver.

8.34.2 Field Documentation

8.34.2.1 `icustate_t ICUDriver::state`

Driver state.

8.34.2.2 `const ICUConfig* ICUDriver::config`

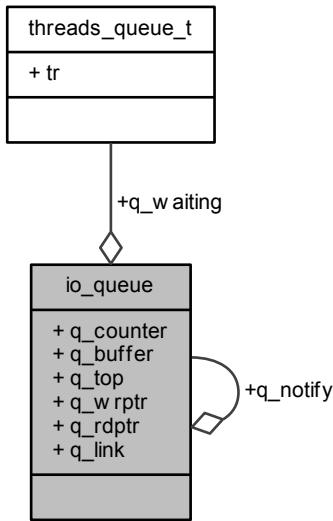
Current configuration data.

8.35 io_queue Struct Reference

Generic I/O queue structure.

```
#include <hal_queues.h>
```

Collaboration diagram for io_queue:



Data Fields

- `threads_queue_t q_waiting`
Waiting thread.
- 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_wptr`
Write pointer.
- uint8_t * `q_rptr`
Read pointer.
- qnotify_t `q_notify`
Data notification callback.
- void * `q_link`
Application defined field.

8.35.1 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.

8.35.2 Field Documentation

8.35.2.1 `threads_queue_t io_queue::q_waiting`

Waiting thread.

8.35.2.2 `volatile size_t io_queue::q_counter`

Resources counter.

8.35.2.3 `uint8_t* io_queue::q_buffer`

Pointer to the queue buffer.

8.35.2.4 `uint8_t* io_queue::q_top`

Pointer to the first location after the buffer.

8.35.2.5 `uint8_t* io_queue::q_wptr`

Write pointer.

8.35.2.6 `uint8_t* io_queue::q_rdptr`

Read pointer.

8.35.2.7 `qnotify_t io_queue::q_notify`

Data notification callback.

8.35.2.8 `void* io_queue::q_link`

Application defined field.

8.36 IOBus Struct Reference

I/O bus descriptor.

```
#include <pal.h>
```

Collaboration diagram for IOBus:



Data Fields

- **ioprtid_t portid**
Port identifier.
- **ioprtmask_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.

8.36.1 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.

8.36.2 Field Documentation

8.36.2.1 ioprtid_t IOBus::portid

Port identifier.

8.36.2.2 ioprtmask_t IOBus::mask

Bus mask aligned to port bit 0.

Note

The bus mask implicitly define the bus width. A logical AND is performed on the bus data.

8.36.2.3 uint_fast8_t IOBus::offset

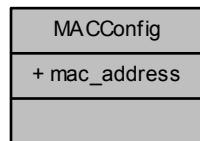
Offset, within the port, of the least significant bit of the bus.

8.37 MACConfig Struct Reference

Driver configuration structure.

```
#include <mac_lld.h>
```

Collaboration diagram for MACConfig:



Data Fields

- `uint8_t * mac_address`

MAC address.

8.37.1 Detailed Description

Driver configuration structure.

8.37.2 Field Documentation

8.37.2.1 `uint8_t* MACConfig::mac_address`

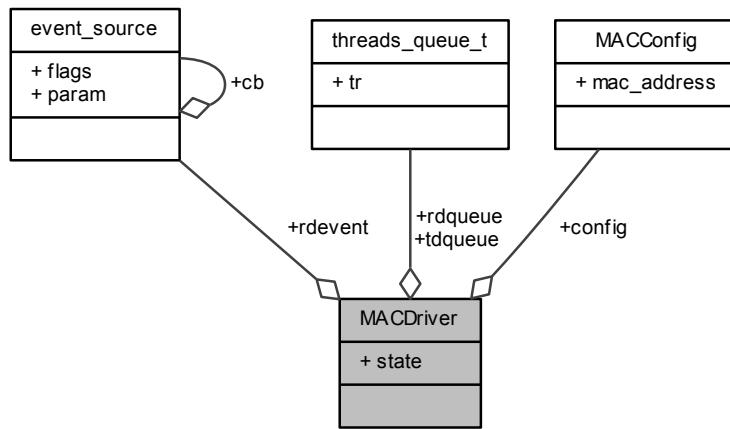
MAC address.

8.38 MACDriver Struct Reference

Structure representing a MAC driver.

```
#include <mac_lld.h>
```

Collaboration diagram for MACDriver:



Data Fields

- **macstate_t state**
Driver state.
- **const MACConfig * config**
Current configuration data.
- **threads_queue_t tdqueue**
Transmit semaphore.
- **threads_queue_t rdqueue**
Receive semaphore.
- **event_source_t rdevent**
Receive event.

8.38.1 Detailed Description

Structure representing a MAC driver.

8.38.2 Field Documentation

8.38.2.1 **macstate_t MACDriver::state**

Driver state.

8.38.2.2 **const MACConfig* MACDriver::config**

Current configuration data.

8.38.2.3 **threads_queue_t MACDriver::tdqueue**

Transmit semaphore.

8.38.2.4 threads_queue_t MACDriver::rdqueue

Receive semaphore.

8.38.2.5 event_source_t MACDriver::rdevent

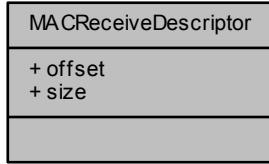
Receive event.

8.39 MACReceiveDescriptor Struct Reference

Structure representing a receive descriptor.

```
#include <mac_lld.h>
```

Collaboration diagram for MACReceiveDescriptor:



Data Fields

- size_t **offset**

Current read offset.

- size_t **size**

Available data size.

8.39.1 Detailed Description

Structure representing a receive descriptor.

8.39.2 Field Documentation

8.39.2.1 size_t MACReceiveDescriptor::offset

Current read offset.

8.39.2.2 size_t MACReceiveDescriptor::size

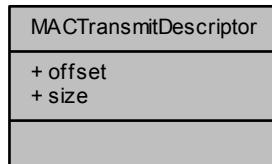
Available data size.

8.40 MACTransmitDescriptor Struct Reference

Structure representing a transmit descriptor.

```
#include <mac_lld.h>
```

Collaboration diagram for MACTransmitDescriptor:



Data Fields

- `size_t offset`
Current write offset.
- `size_t size`
Available space size.

8.40.1 Detailed Description

Structure representing a transmit descriptor.

8.40.2 Field Documentation

8.40.2.1 `size_t MACTransmitDescriptor::offset`

Current write offset.

8.40.2.2 `size_t MACTransmitDescriptor::size`

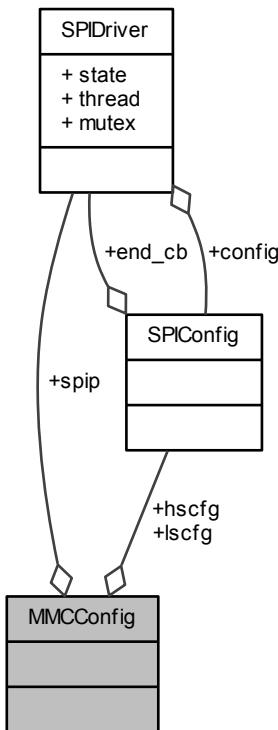
Available space size.

8.41 MMCConfig Struct Reference

MMC/SD over SPI driver configuration structure.

```
#include <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.

8.41.1 Detailed Description

MMC/SD over SPI driver configuration structure.

8.41.2 Field Documentation

8.41.2.1 `SPIDriver* MMCConfig::spip`

SPI driver associated to this MMC driver.

8.41.2.2 const SPIConfig* MMCConfig::lscfg

SPI low speed configuration used during initialization.

8.41.2.3 const SPIConfig* MMCConfig::hscfg

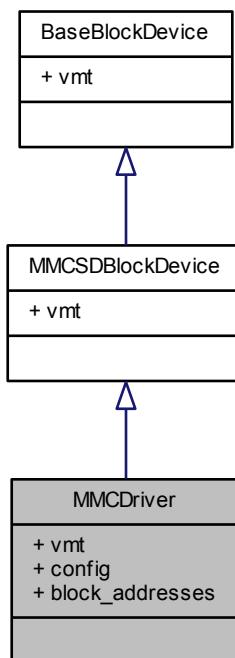
SPI high speed configuration used during transfers.

8.42 MMCDriver Struct Reference

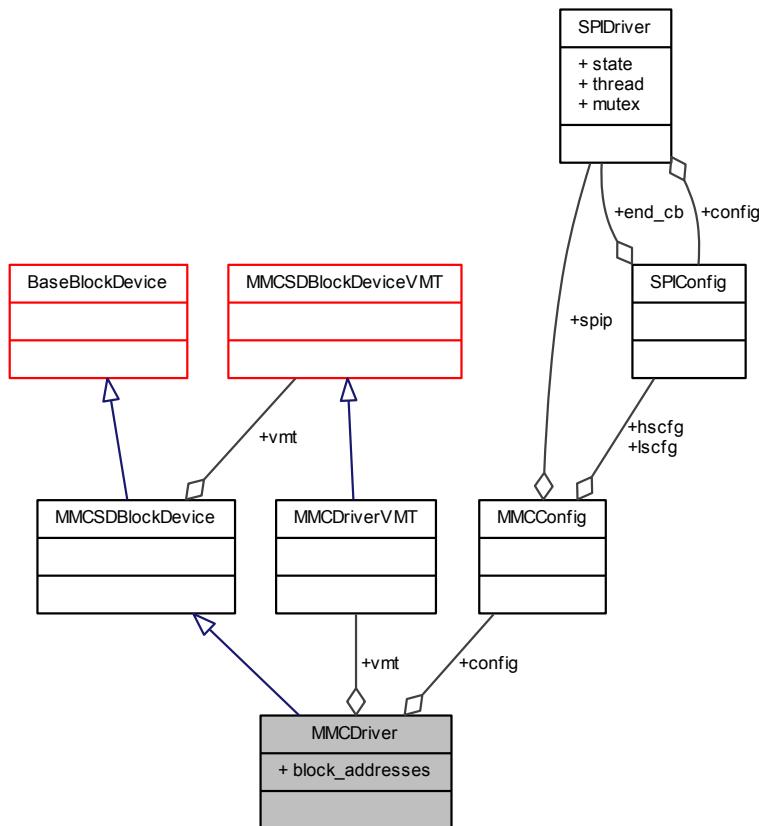
Structure representing a MMC/SD over SPI driver.

```
#include <mmc_spi.h>
```

Inheritance diagram for MMCDriver:



Collaboration diagram for MMCDDriver:



Data Fields

- const struct `MMCDriverVMT` * `vmt`
Virtual Methods Table.
- `_mmcsd_block_device_data` const `MMCConfig` * `config`
Current configuration data.

8.42.1 Detailed Description

Structure representing a MMC/SD over SPI driver.

8.42.2 Field Documentation

8.42.2.1 const struct `MMCDriverVMT`* `MMCDriver::vmt`

Virtual Methods Table.

8.42.2.2 _mmcsd_block_device_data const MMCConfig* MMCDriver::config

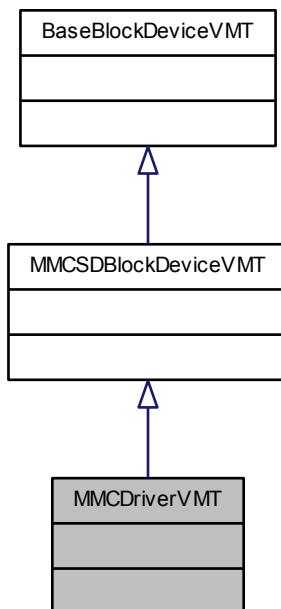
Current configuration data.

8.43 MMCDriverVMT Struct Reference

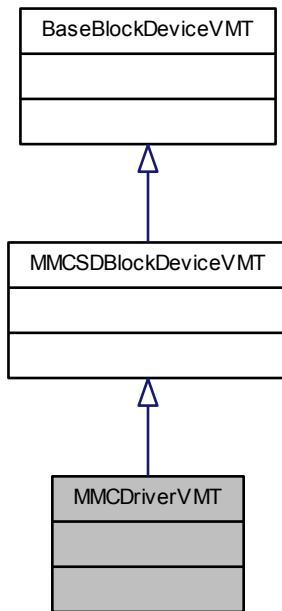
[MMCDriver](#) virtual methods table.

```
#include <mmc_spi.h>
```

Inheritance diagram for MMCDriverVMT:



Collaboration diagram for MMCBlockDriverVMT:



8.43.1 Detailed Description

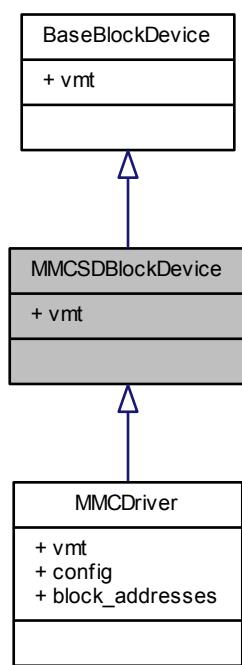
[MMCBlockDriver](#) virtual methods table.

8.44 MMCSDBlockDevice Struct Reference

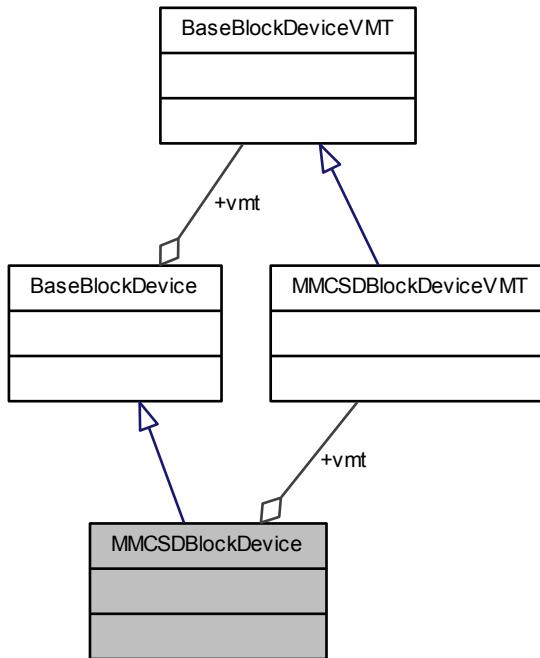
MCC/SD block device class.

```
#include <hal_mmcblk.h>
```

Inheritance diagram for MMCSDBlockDevice:



Collaboration diagram for MMCSDBlockDevice:



Data Fields

- const struct `MMCSDBlockDeviceVMT` * `vmt`

Virtual Methods Table.

8.44.1 Detailed Description

MCC/SD block device class.

This class represents a, block-accessible, MMC/SD device.

8.44.2 Field Documentation

8.44.2.1 const struct `MMCSDBlockDeviceVMT`* `MMCSDBlockDevice::vmt`

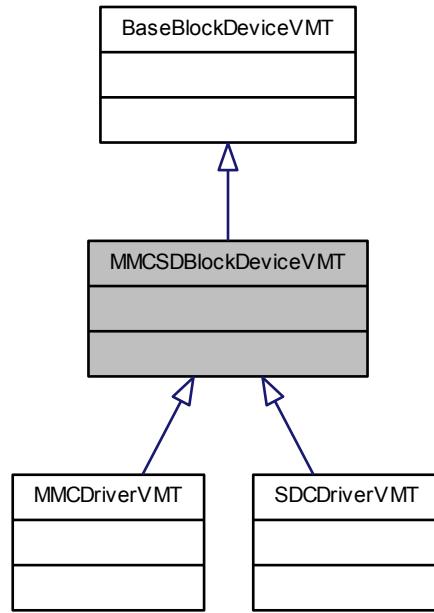
Virtual Methods Table.

8.45 MMCSDBlockDeviceVMT Struct Reference

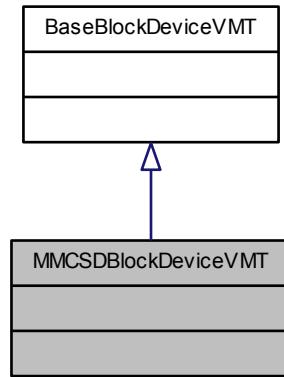
`MMCSDBlockDevice` virtual methods table.

```
#include <hal_mmcisd.h>
```

Inheritance diagram for MMCSDBlockDeviceVMT:



Collaboration diagram for MMCSDBlockDeviceVMT:



8.45.1 Detailed Description

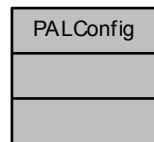
[MMCSDBlockDevice](#) virtual methods table.

8.46 PALConfig Struct Reference

Generic I/O ports static initializer.

```
#include <pal_lld.h>
```

Collaboration diagram for PALConfig:



8.46.1 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 initialize the digital I/O subsystem. This represents only the initial setup, specific pads or whole ports can be reprogrammed at later time.

Note

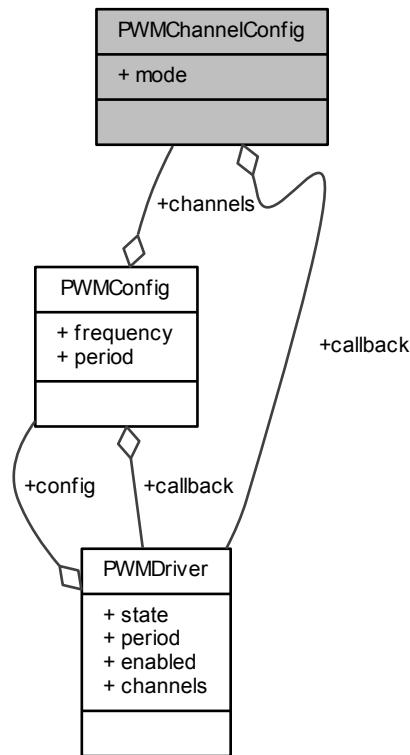
Implementations may extend this structure to contain more, architecture dependent, fields.

8.47 PWMChannelConfig Struct Reference

Type of a PWM driver channel configuration structure.

```
#include <pwm_lld.h>
```

Collaboration diagram for PWMChannelConfig:



Data Fields

- `pwmmode_t mode`
Channel active logic level.
- `pwmcallback_t callback`
Channel callback pointer.

8.47.1 Detailed Description

Type of a PWM driver channel configuration structure.

8.47.2 Field Documentation

8.47.2.1 `pwmmode_t PWMChannelConfig::mode`

Channel active logic level.

8.47.2.2 `pwmcallback_t PWMChannelConfig::callback`

Channel callback pointer.

Note

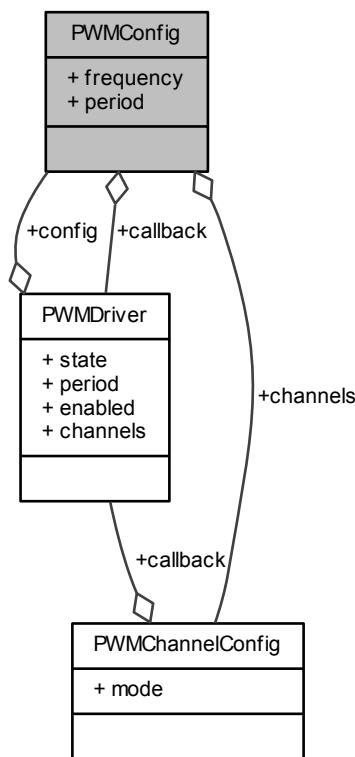
This callback is invoked on the channel compare event. If set to NULL then the callback is disabled.

8.48 PWMConfig Struct Reference

Type of a PWM driver configuration structure.

```
#include <pwm_ll.h>
```

Collaboration diagram for PWMConfig:



Data Fields

- `uint32_t frequency`
Timer clock in Hz.
- `pwmcnt_t period`
PWM period in ticks.
- `pwmcallback_t callback`
Periodic callback pointer.
- `PWMChannelConfig channels [PWM_CHANNELS]`
Channels configurations.

8.48.1 Detailed Description

Type of a PWM driver configuration structure.

8.48.2 Field Documentation

8.48.2.1 uint32_t PWMConfig::frequency

Timer clock in Hz.

Note

The low level can use assertions in order to catch invalid frequency specifications.

8.48.2.2 pwmcnt_t PWMConfig::period

PWM period in ticks.

Note

The low level can use assertions in order to catch invalid period specifications.

8.48.2.3 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.

8.48.2.4 PWMChannelConfig PWMConfig::channels[PWM_CHANNELS]

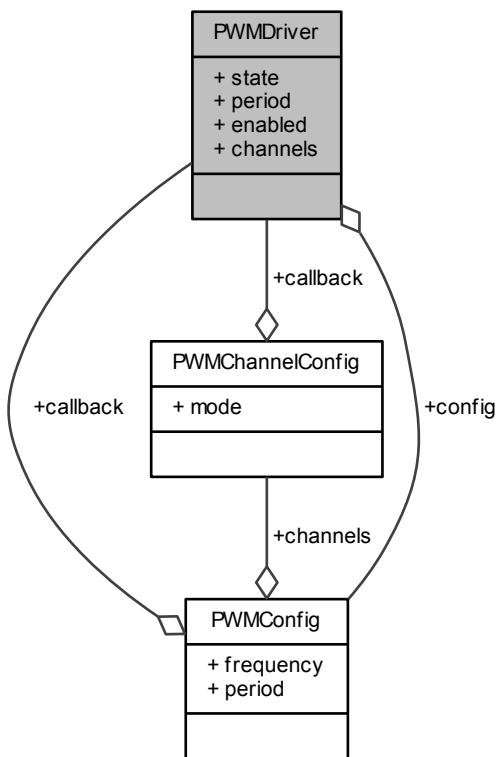
Channels configurations.

8.49 PWMDriver Struct Reference

Structure representing a PWM driver.

```
#include <pwm_lld.h>
```

Collaboration diagram for PWMDriver:



Data Fields

- `pwmstate_t state`
Driver state.
- `const PWMConfig * config`
Current driver configuration data.
- `pwmcnt_t period`
Current PWM period in ticks.
- `pwmchnmsk_t enabled`
Mask of the enabled channels.
- `pwmchannel_t channels`
Number of channels in this instance.

8.49.1 Detailed Description

Structure representing a PWM driver.

8.49.2 Field Documentation

8.49.2.1 pwmstate_t PWMDriver::state

Driver state.

8.49.2.2 const PWMConfig* PWMDriver::config

Current driver configuration data.

8.49.2.3 pwmcnt_t PWMDriver::period

Current PWM period in ticks.

8.49.2.4 pwmchnmsk_t PWMDriver::enabled

Mask of the enabled channels.

8.49.2.5 pwmchannel_t PWMDriver::channels

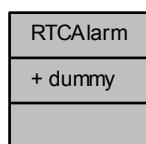
Number of channels in this instance.

8.50 RTCAlarm Struct Reference

Type of a structure representing an RTC alarm time stamp.

```
#include <rtc_llld.h>
```

Collaboration diagram for RTCAlarm:



8.50.1 Detailed Description

Type of a structure representing an RTC alarm time stamp.

8.51 RTCDateTime Struct Reference

Type of a structure representing an RTC date/time stamp.

```
#include <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.

8.51.1 Detailed Description

Type of a structure representing an RTC date/time stamp.

8.51.2 Field Documentation

8.51.2.1 `uint32_t RTCDateTime::year`

Years since 1980.

8.51.2.2 `uint32_t RTCDateTime::month`

Months 1..12.

8.51.2.3 `uint32_t RTCDateTime::dstflag`

DST correction flag.

8.51.2.4 uint32_t RTCDateTime::dayofweek

Day of week 1..7.

8.51.2.5 uint32_t RTCDateTime::day

Day of the month 1..31.

8.51.2.6 uint32_t RTCDateTime::millisecond

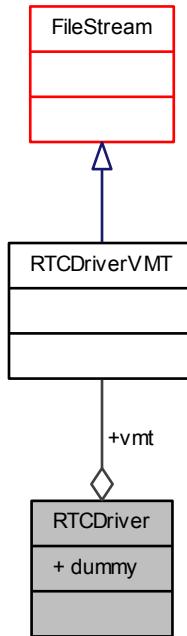
Milliseconds since midnight.

8.52 RTCDriver Struct Reference

Structure representing an RTC driver.

```
#include <rtc_llld.h>
```

Collaboration diagram for RTCDriver:



Data Fields

- const struct [RTCDriverVMT](#) * vmt

Virtual Methods Table.

8.52.1 Detailed Description

Structure representing an RTC driver.

8.52.2 Field Documentation

8.52.2.1 const struct RTCDriverVMT* RTCDriver::vmt

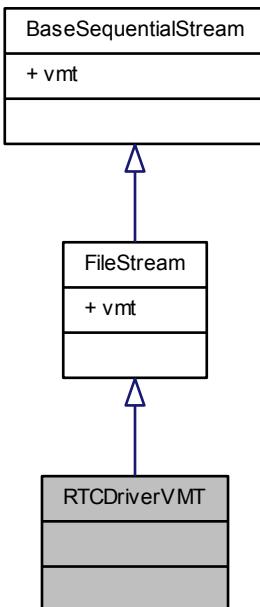
Virtual Methods Table.

8.53 RTCDriverVMT Struct Reference

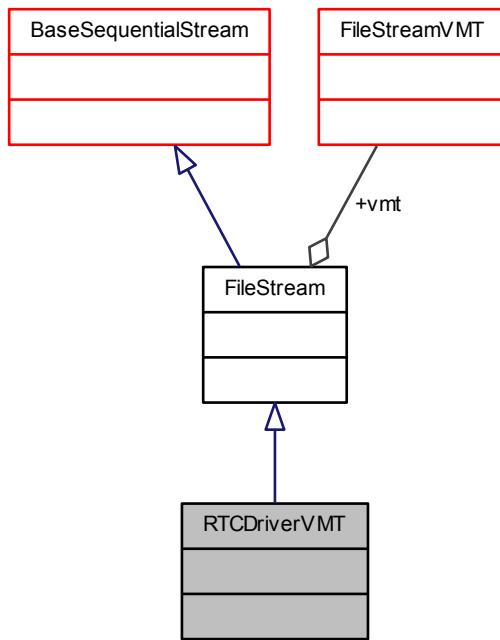
[RTCDriver](#) virtual methods table.

```
#include <rtc_lld.h>
```

Inheritance diagram for RTCDriverVMT:



Collaboration diagram for RTCDriverVMT:



Additional Inherited Members

8.53.1 Detailed Description

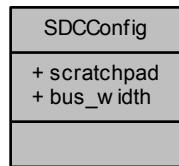
[RTCDriver](#) virtual methods table.

8.54 SDCCConfig Struct Reference

Driver configuration structure.

```
#include <sdc_lld.h>
```

Collaboration diagram for SDCCConfig:



Data Fields

- `uint8_t * scratchpad`
Working area for memory consuming operations.
- `sdcbusmode_t bus_width`
Bus width.

8.54.1 Detailed Description

Driver configuration structure.

Note

It could be empty on some architectures.

8.54.2 Field Documentation

8.54.2.1 `uint8_t* SDCCConfig::scratchpad`

Working area for memory consuming operations.

Note

It is mandatory for detecting MMC cards bigger than 2GB else it can be NULL.
Memory pointed by this buffer is only used by `sdcConnect ()`, afterward it can be reused for other purposes.

8.54.2.2 `sdcbusmode_t SDCCConfig::bus_width`

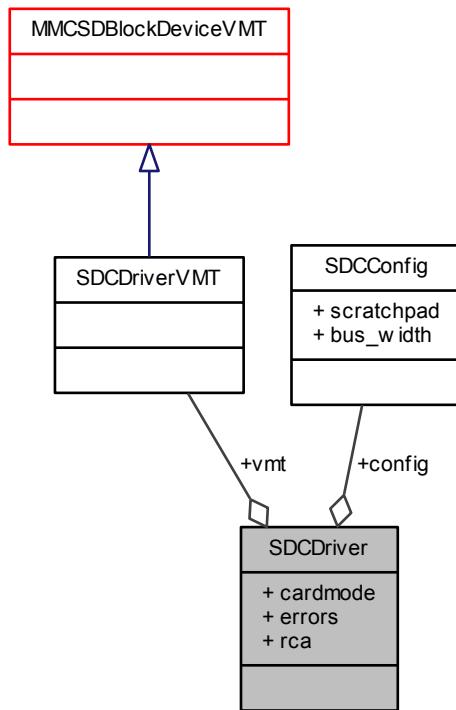
Bus width.

8.55 SDCDriver Struct Reference

Structure representing an SDC driver.

```
#include <sdc_lld.h>
```

Collaboration diagram for SDCDriver:



Data Fields

- const struct [SDCDriverVMT](#) * vmt
Virtual Methods Table.
- [_mmcsd_block_device_data](#) const [SDCCConfig](#) * config
Current configuration data.
- [sdcmode_t](#) cardmode
Various flags regarding the mounted card.
- [sdcflags_t](#) errors
Errors flags.
- uint32_t rca
Card RCA.

8.55.1 Detailed Description

Structure representing an SDC driver.

8.55.2 Field Documentation

8.55.2.1 const struct [SDCDriverVMT](#)* SDCDriver::vmt

Virtual Methods Table.

8.55.2.2 `_mmcsd_block_device_data const SDCCConfig* SDCDriver::config`

Current configuration data.

8.55.2.3 `sdcmode_t SDCDriver::cardmode`

Various flags regarding the mounted card.

8.55.2.4 `sdcflags_t SDCDriver::errors`

Errors flags.

8.55.2.5 `uint32_t SDCDriver::rca`

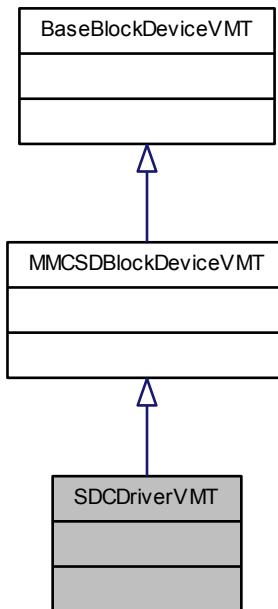
Card RCA.

8.56 SDCDriverVMT Struct Reference

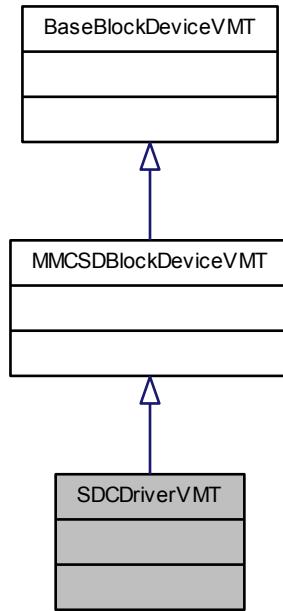
`SDCDriver` virtual methods table.

```
#include <sdc_lld.h>
```

Inheritance diagram for SDCDriverVMT:



Collaboration diagram for SDCDriverVMT:



8.56.1 Detailed Description

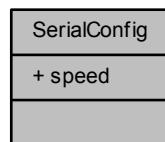
[SDCDriver](#) virtual methods table.

8.57 SerialConfig Struct Reference

PLATFORM Serial Driver configuration structure.

```
#include <serial_lld.h>
```

Collaboration diagram for SerialConfig:



Data Fields

- `uint32_t speed`

Bit rate.

8.57.1 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.

8.57.2 Field Documentation

8.57.2.1 `uint32_t SerialConfig::speed`

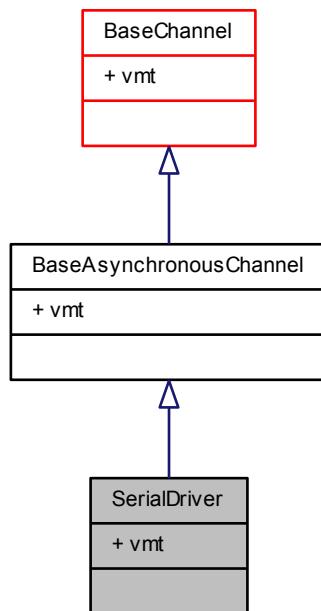
Bit rate.

8.58 SerialDriver Struct Reference

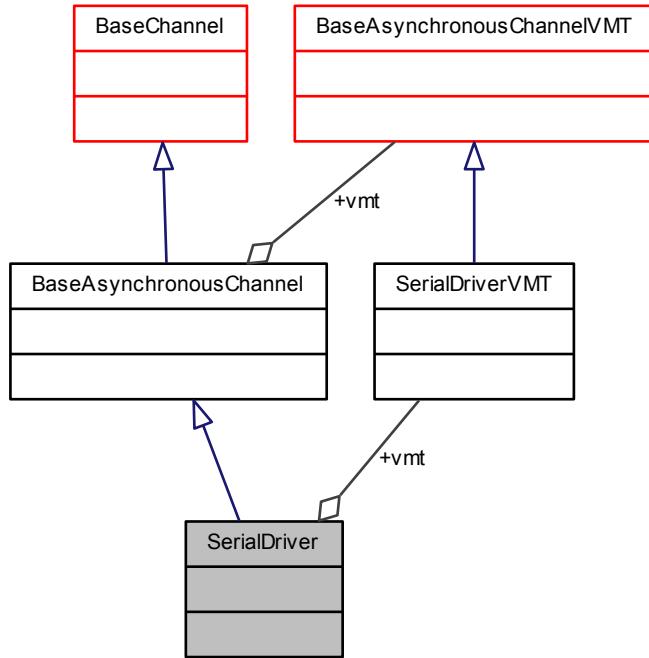
Full duplex serial driver class.

```
#include <serial.h>
```

Inheritance diagram for SerialDriver:



Collaboration diagram for SerialDriver:



Data Fields

- const struct [SerialDriverVMT](#) * vmt

Virtual Methods Table.

8.58.1 Detailed Description

Full duplex serial driver class.

This class extends [BaseAsynchronousChannel](#) by adding physical I/O queues.

8.58.2 Field Documentation

8.58.2.1 const struct [SerialDriverVMT](#)* SerialDriver::vmt

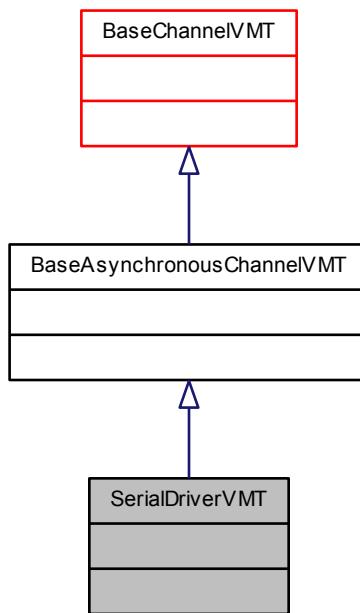
Virtual Methods Table.

8.59 SerialDriverVMT Struct Reference

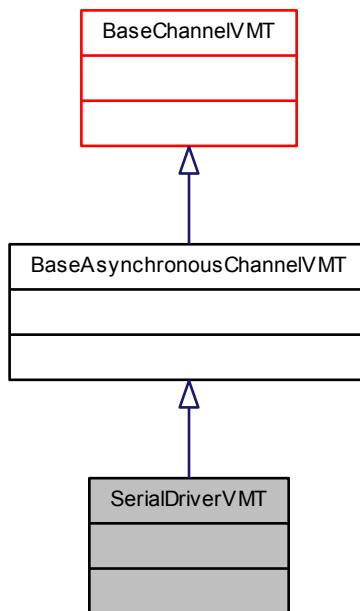
[SerialDriver](#) virtual methods table.

```
#include <serial.h>
```

Inheritance diagram for SerialDriverVMT:



Collaboration diagram for SerialDriverVMT:



8.59.1 Detailed Description

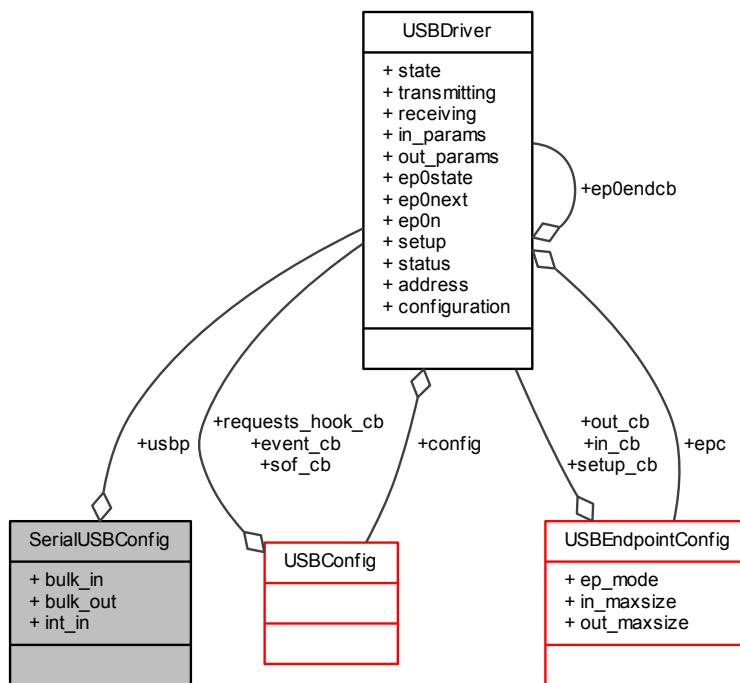
[SerialDriver](#) virtual methods table.

8.60 SerialUSBConfig Struct Reference

Serial over USB Driver configuration structure.

```
#include <serial_usb.h>
```

Collaboration diagram for SerialUSBConfig:



Data Fields

- [USBDriver * usbp](#)
USB driver to use.
- [usbep_t bulk_in](#)
Bulk IN endpoint used for outgoing data transfer.
- [usbep_t bulk_out](#)
Bulk OUT endpoint used for incoming data transfer.
- [usbep_t int_in](#)
Interrupt IN endpoint used for notifications.

8.60.1 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.

8.60.2 Field Documentation

8.60.2.1 `USBDriver* SerialUSBConfig::usbp`

USB driver to use.

8.60.2.2 `usbep_t SerialUSBConfig::bulk_in`

Bulk IN endpoint used for outgoing data transfer.

8.60.2.3 `usbep_t SerialUSBConfig::bulk_out`

Bulk OUT endpoint used for incoming data transfer.

8.60.2.4 `usbep_t SerialUSBConfig::int_in`

Interrupt IN endpoint used for notifications.

Note

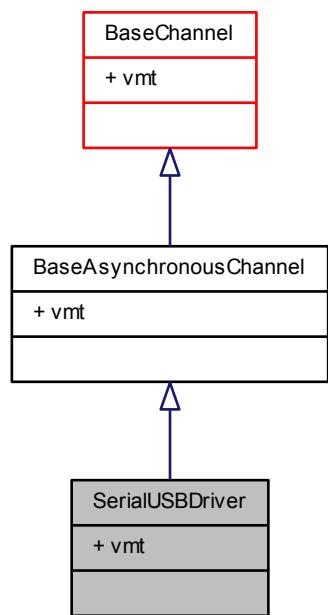
If set to zero then the INT endpoint is assumed to be not present, USB descriptors must be changed accordingly.

8.61 SerialUSBDriver Struct Reference

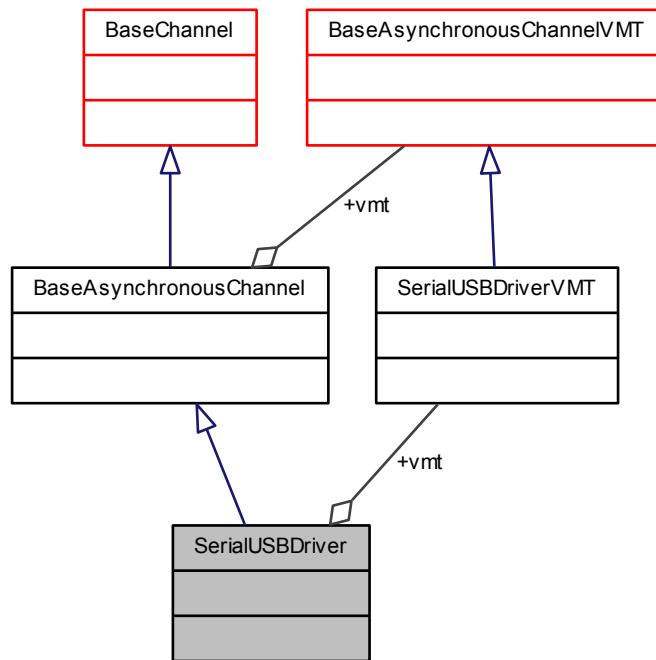
Full duplex serial driver class.

```
#include <serial_usb.h>
```

Inheritance diagram for SerialUSBDriver:



Collaboration diagram for SerialUSBDriver:



Data Fields

- const struct [SerialUSBDriverVMT](#) * vmt

Virtual Methods Table.

8.61.1 Detailed Description

Full duplex serial driver class.

This class extends [BaseAsynchronousChannel](#) by adding physical I/O queues.

8.61.2 Field Documentation

8.61.2.1 const struct [SerialUSBDriverVMT](#)* [SerialUSBDriver::vmt](#)

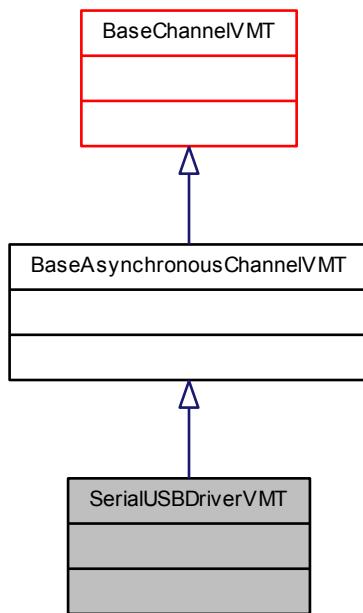
Virtual Methods Table.

8.62 [SerialUSBDriverVMT](#) Struct Reference

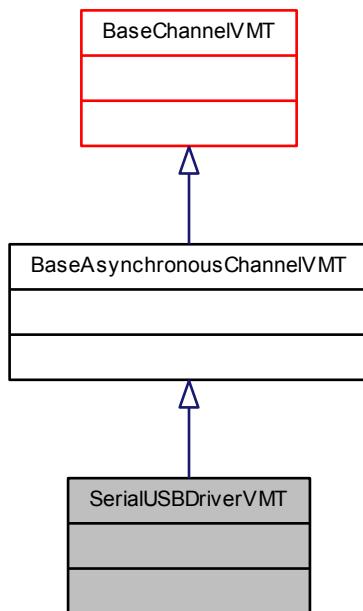
[SerialDriver](#) virtual methods table.

```
#include <serial_usb.h>
```

Inheritance diagram for SerialUSBDriverVMT:



Collaboration diagram for SerialUSBDriverVMT:



8.62.1 Detailed Description

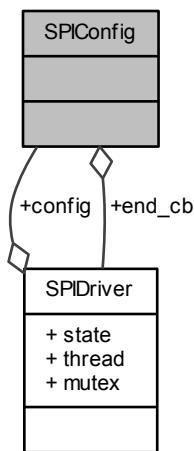
`SerialDriver` virtual methods table.

8.63 SPIConfig Struct Reference

Driver configuration structure.

```
#include <spi_lld.h>
```

Collaboration diagram for SPIConfig:



Data Fields

- `spicallback_t end_cb`

Operation complete callback or NULL.

8.63.1 Detailed Description

Driver configuration structure.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.

8.63.2 Field Documentation

8.63.2.1 `spicallback_t SPIConfig::end_cb`

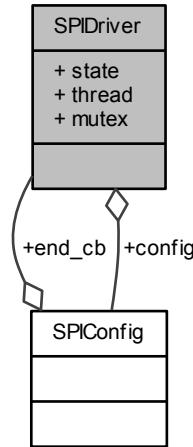
Operation complete callback or NULL.

8.64 SPIDriver Struct Reference

Structure representing an SPI driver.

```
#include <spi_llld.h>
```

Collaboration diagram for SPIDriver:



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.

8.64.1 Detailed Description

Structure representing an SPI driver.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.

8.64.2 Field Documentation

8.64.2.1 spistate_t SPIDriver::state

Driver state.

8.64.2.2 const SPIConfig* SPIDriver::config

Current configuration data.

8.64.2.3 thread_reference_t SPIDriver::thread

Waiting thread.

8.64.2.4 mutex_t SPIDriver::mutex

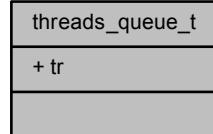
Mutex protecting the peripheral.

8.65 threads_queue_t Struct Reference

Type of a thread queue.

```
#include <osal.h>
```

Collaboration diagram for threads_queue_t:



8.65.1 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

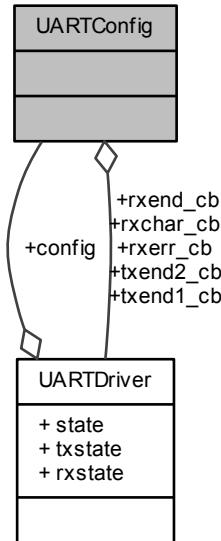
If the OSAL is implemented on a bare metal machine without RTOS then the queue can be implemented as a single thread reference.

8.66 UARTConfig Struct Reference

Driver configuration structure.

```
#include <uart_lld.h>
```

Collaboration diagram for UARTConfig:



Data Fields

- [uartcb_t txend1_cb](#)
End of transmission buffer callback.
- [uartcb_t txend2_cb](#)
Physical end of transmission callback.
- [uartcb_t rxend_cb](#)
Receive buffer filled callback.
- [uartccb_t rxchar_cb](#)
Character received while out if the `UART_RECEIVE` state.
- [uartecb_t rxerr_cb](#)
Receive error callback.

8.66.1 Detailed Description

Driver configuration structure.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.

8.66.2 Field Documentation

8.66.2.1 [uartcb_t](#) `UARTConfig::txend1_cb`

End of transmission buffer callback.

8.66.2.2 `uartcb_t` `UARTConfig::txend2_cb`

Physical end of transmission callback.

8.66.2.3 `uartcb_t` `UARTConfig::rxend_cb`

Receive buffer filled callback.

8.66.2.4 `uartccb_t` `UARTConfig::rxchar_cb`

Character received while out if the `UART_RECEIVE` state.

8.66.2.5 `uartccb_t` `UARTConfig::rxerr_cb`

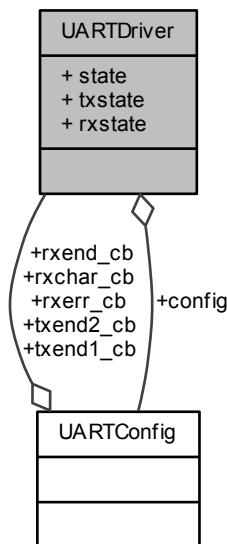
Receive error callback.

8.67 `UARTDriver` Struct Reference

Structure representing an UART driver.

```
#include <uart_lld.h>
```

Collaboration diagram for `UARTDriver`:



Data Fields

- `uartstate_t state`

Driver state.

- `uarttxstate_t txstate`

Transmitter state.

- `uartrxstate_t rxstate`

Receiver state.

- const `UARTConfig * config`

Current configuration data.

8.67.1 Detailed Description

Structure representing an UART driver.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.

8.67.2 Field Documentation

8.67.2.1 `uartstate_t` `UARTDriver::state`

Driver state.

8.67.2.2 `uarttxstate_t` `UARTDriver::txstate`

Transmitter state.

8.67.2.3 `uartrxstate_t` `UARTDriver::rxstate`

Receiver state.

8.67.2.4 const `UARTConfig*` `UARTDriver::config`

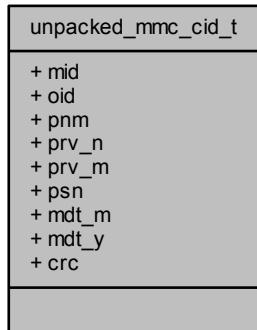
Current configuration data.

8.68 unpacked_mmc_cid_t Struct Reference

Unpacked CID register from MMC.

```
#include <hal_mmcsd.h>
```

Collaboration diagram for unpacked_mmc_cid_t:



8.68.1 Detailed Description

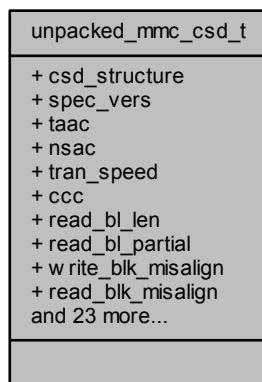
Unpacked CID register from MMC.

8.69 unpacked_mmc_csd_t Struct Reference

Unpacked CSD register from MMC.

```
#include <hal_mmc_csd.h>
```

Collaboration diagram for unpacked_mmc_csd_t:



8.69.1 Detailed Description

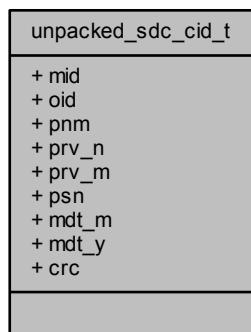
Unpacked CSD register from MMC.

8.70 unpacked_sdc_cid_t Struct Reference

Unpacked CID register from SDC.

```
#include <hal_mmc.h>
```

Collaboration diagram for unpacked_sdc_cid_t:



8.70.1 Detailed Description

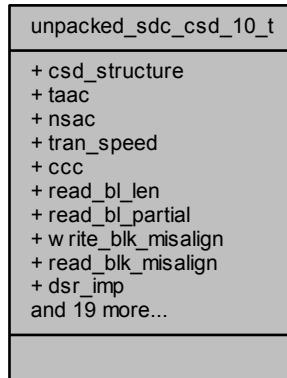
Unpacked CID register from SDC.

8.71 unpacked_sdc_csd_10_t Struct Reference

Unpacked CSD v1.0 register from SDC.

```
#include <hal_mmc.h>
```

Collaboration diagram for unpacked_sdc_csd_10_t:



8.71.1 Detailed Description

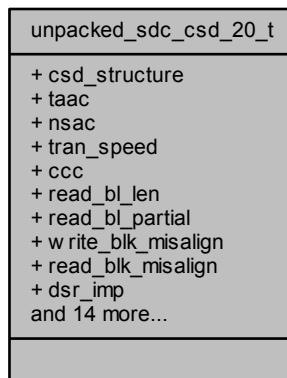
Unpacked CSD v1.0 register from SDC.

8.72 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:



8.72.1 Detailed Description

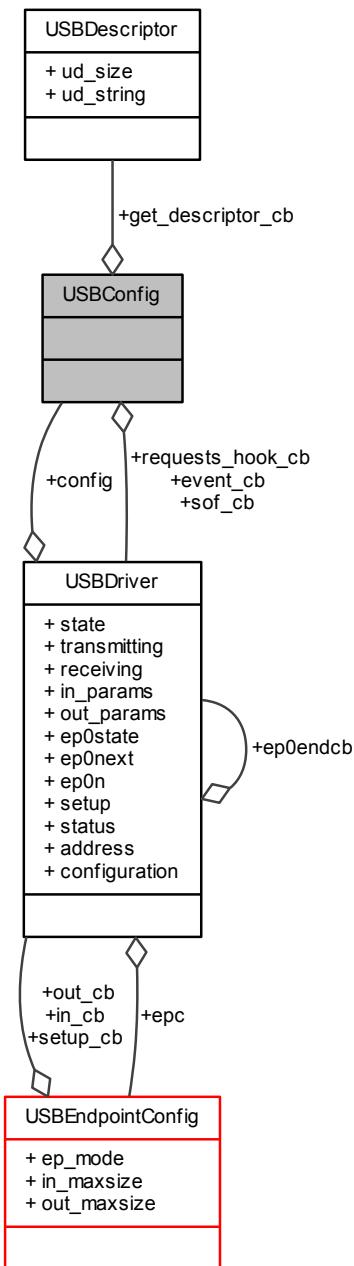
Unpacked CSD v2.0 register from SDC.

8.73 USBConfig Struct Reference

Type of an USB driver configuration structure.

```
#include <usb_lld.h>
```

Collaboration diagram for USBConfig:



Data Fields

- **usbeventcb_t event_cb**
USB events callback.
- **usbgetdescriptor_t get_descriptor_cb**
Device GET_DESCRIPTOR request callback.
- **usbreqhandler_t requests_hook_cb**

Requests hook callback.

- **usbcallback_t sof_cb**
Start Of Frame callback.

8.73.1 Detailed Description

Type of an USB driver configuration structure.

8.73.2 Field Documentation

8.73.2.1 usbeventcb_t USBConfig::event_cb

USB events callback.

This callback is invoked when an USB driver event is registered.

8.73.2.2 usbgetdescriptor_t USBConfig::get_descriptor_cb

Device GET_DESCRIPTOR request callback.

Note

This callback is mandatory and cannot be set to NULL.

8.73.2.3 usbreqhandler_t USBConfig::requests_hook_cb

Requests hook callback.

This hook allows to be notified of standard requests or to handle non standard requests.

8.73.2.4 usbcallback_t USBConfig::sof_cb

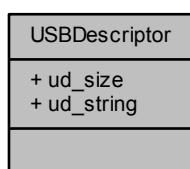
Start Of Frame callback.

8.74 USBDescriptor Struct Reference

Type of an USB descriptor.

```
#include <usb.h>
```

Collaboration diagram for USBDescriptor:



Data Fields

- `size_t ud_size`
Descriptor size in unicode characters.
- `const uint8_t * ud_string`
Pointer to the descriptor.

8.74.1 Detailed Description

Type of an USB descriptor.

8.74.2 Field Documentation

8.74.2.1 `size_t USBDescriptor::ud_size`

Descriptor size in unicode characters.

8.74.2.2 `const uint8_t* USBDescriptor::ud_string`

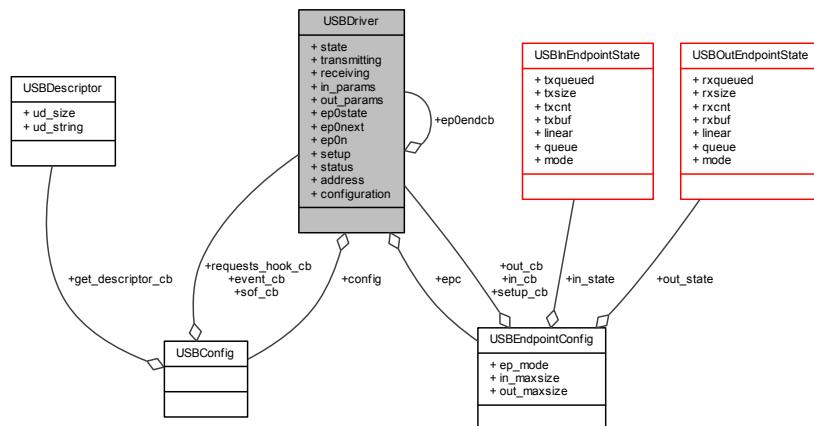
Pointer to the descriptor.

8.75 USBDriver Struct Reference

Structure representing an USB driver.

```
#include <usb_ll.h>
```

Collaboration diagram for USBDriver:



Data Fields

- `usbstate_t state`
Driver state.
- `const USBConfig * config`

- **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.

8.75.1 Detailed Description

Structure representing an USB driver.

8.75.2 Field Documentation

8.75.2.1 **usbstate_t USBDriver::state**

Driver state.

8.75.2.2 **const USBConfig* USBDriver::config**

Current configuration data.

8.75.2.3 **uint16_t USBDriver::transmitting**

Bit map of the transmitting IN endpoints.

8.75.2.4 **uint16_t USBDriver::receiving**

Bit map of the receiving OUT endpoints.

8.75.2.5 `const USBEndpointConfig* USBDriver::epc[USB_MAX_ENDPOINTS+1]`

Active endpoints configurations.

8.75.2.6 `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.

8.75.2.7 `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.

8.75.2.8 `usbep0state_t USBDriver::ep0state`

Endpoint 0 state.

8.75.2.9 `uint8_t* USBDriver::ep0next`

Next position in the buffer to be transferred through endpoint 0.

8.75.2.10 `size_t USBDriver::ep0n`

Number of bytes yet to be transferred through endpoint 0.

8.75.2.11 `usbcallback_t USBDriver::ep0endcb`

Endpoint 0 end transaction callback.

8.75.2.12 `uint8_t USBDriver::setup[8]`

Setup packet buffer.

8.75.2.13 `uint16_t USBDriver::status`

Current USB device status.

8.75.2.14 `uint8_t USBDriver::address`

Assigned USB address.

8.75.2.15 `uint8_t USBDriver::configuration`

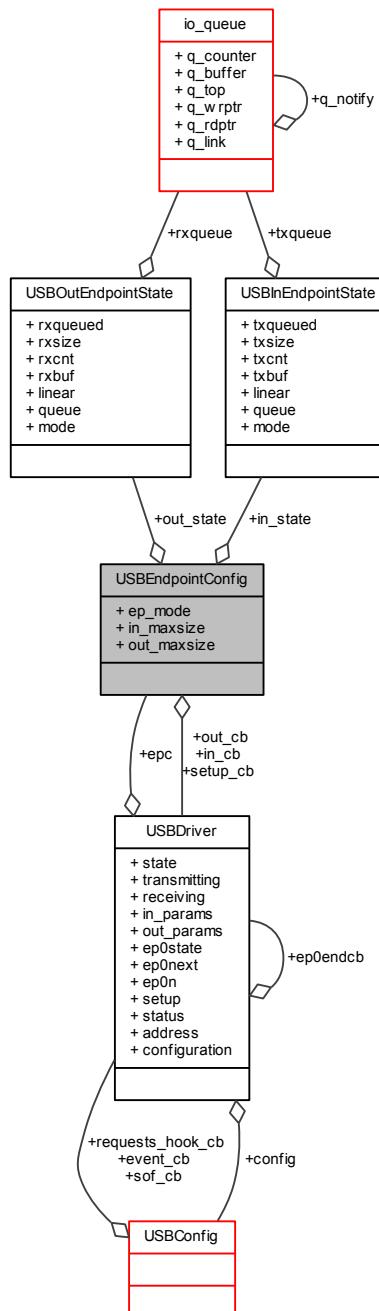
Current USB device configuration.

8.76 USBEndpointConfig Struct Reference

Type of an USB endpoint configuration structure.

```
#include <usb_llld.h>
```

Collaboration diagram for USBEndpointConfig:



Data Fields

- `uint32_t ep_mode`
Type and mode of the endpoint.
- `usbepcallback_t setup_cb`
Setup packet notification callback.
- `usbepcallback_t in_cb`
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`
USBInEndpointState associated to the IN endpoint.
- `USBOutEndpointState * out_state`
USBOutEndpointState associated to the OUT endpoint.

8.76.1 Detailed Description

Type of an USB endpoint configuration structure.

Note

Platform specific restrictions may apply to endpoints.

8.76.2 Field Documentation

8.76.2.1 `uint32_t USBEndpointConfig::ep_mode`

Type and mode of the endpoint.

8.76.2.2 `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.

8.76.2.3 `usbepcallback_t USBEndpointConfig::in_cb`

IN endpoint notification callback.

This field must be set to `NULL` if the IN endpoint is not used.

8.76.2.4 usbepcallback_t USBEndpointConfig::out_cb

OUT endpoint notification callback.

This field must be set to `NULL` if the OUT endpoint is not used.

8.76.2.5 uint16_t USBEndpointConfig::in_maxsize

IN endpoint maximum packet size.

This field must be set to zero if the IN endpoint is not used.

8.76.2.6 uint16_t USBEndpointConfig::out_maxsize

OUT endpoint maximum packet size.

This field must be set to zero if the OUT endpoint is not used.

8.76.2.7 USBInEndpointState* USBEndpointConfig::in_state

USBEndpointState associated to the IN endpoint.

This structure maintains the state of the IN endpoint.

8.76.2.8 USBOutEndpointState* USBEndpointConfig::out_state

USBEndpointState associated to the OUT endpoint.

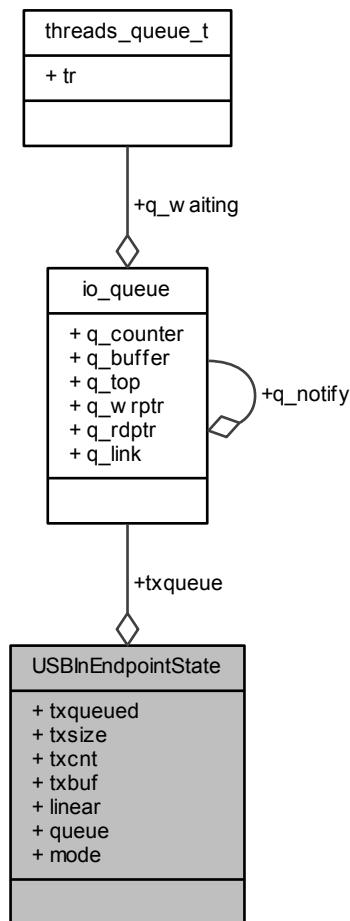
This structure maintains the state of the OUT endpoint.

8.77 USBInEndpointState Struct Reference

Type of an IN endpoint state structure.

```
#include <usb_lld.h>
```

Collaboration diagram for USBInEndpointState:



Data Fields

- bool `txqueued`
Buffer mode, queue or linear.
- size_t `txsize`
Requested transmit transfer size.
- size_t `txcnt`
Transmitted bytes so far.
- const uint8_t * `txbuf`
Pointer to the transmission linear buffer.
- output_queue_t * `txqueue`
Pointer to the output queue.

8.77.1 Detailed Description

Type of an IN endpoint state structure.

8.77.2 Field Documentation

8.77.2.1 bool USBInEndpointState::txqueued

Buffer mode, queue or linear.

8.77.2.2 size_t USBInEndpointState::txsize

Requested transmit transfer size.

8.77.2.3 size_t USBInEndpointState::txcnt

Transmitted bytes so far.

8.77.2.4 const uint8_t* USBInEndpointState::txbuf

Pointer to the transmission linear buffer.

8.77.2.5 output_queue_t* USBInEndpointState::txqueue

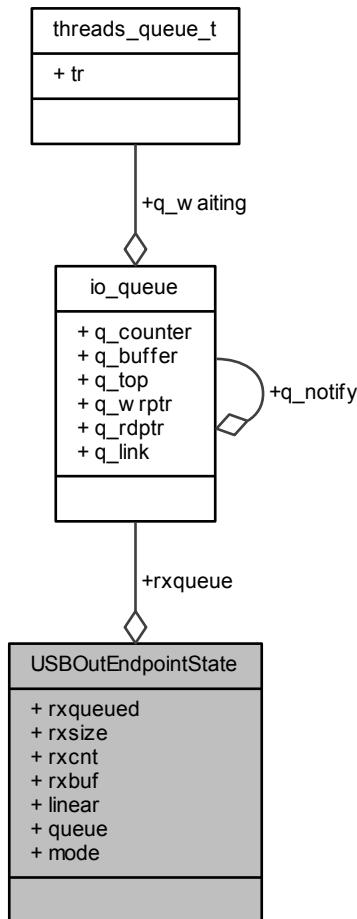
Pointer to the output queue.

8.78 USBOutEndpointState Struct Reference

Type of an OUT endpoint state structure.

```
#include <usb_lld.h>
```

Collaboration diagram for USBOutEndpointState:



Data Fields

- `bool rxqueued`
Buffer mode, queue or linear.
- `size_t rxsized`
Requested receive transfer size.
- `size_t rcnt`
Received bytes so far.
- `uint8_t * rbuf`
Pointer to the receive linear buffer.
- `input_queue_t * queue`
Pointer to the input queue.

8.78.1 Detailed Description

Type of an OUT endpoint state structure.

8.78.2 Field Documentation

8.78.2.1 `bool USBOutEndpointState::rxqueued`

Buffer mode, queue or linear.

8.78.2.2 `size_t USBOutEndpointState::rxsize`

Requested receive transfer size.

8.78.2.3 `size_t USBOutEndpointState::rcnt`

Received bytes so far.

8.78.2.4 `uint8_t* USBOutEndpointState::rxbuf`

Pointer to the receive linear buffer.

8.78.2.5 `input_queue_t* USBOutEndpointState::rxqueue`

Pointer to the input queue.

Chapter 9

File Documentation

9.1 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 `ADCCConfig` *config)
Configures and activates the ADC peripheral.
- void `adcStop` (`ADCDriver` *adcp)
Deactivates the ADC peripheral.
- void `adcStartConversion` (`ADCDriver` *adcp, const `ADCCConversionGroup` *grpp, `adcsample_t` *samples, `size_t` depth)
Starts an ADC conversion.
- void `adcStartConversionI` (`ADCDriver` *adcp, const `ADCCConversionGroup` *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 `ADCCConversionGroup` *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.

9.1.1 Detailed Description

ADC Driver code.

9.2 adc.h File Reference

ADC Driver macros and structures.

```
#include "adc_lld.h"
```

Macros

ADC configuration options

- #define `ADC_USE_WAIT` TRUE
Enables synchronous APIs.
- #define `ADC_USE_MUTUAL_EXCLUSION` TRUE
Enables the `adcAcquireBus()` and `adcReleaseBus()` APIs.

Low level driver helper macros

- #define `_adc_reset_i`(adcp) `osalThreadResumel(&(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.

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 `adclInit` (void)
ADC Driver initialization.
- void `adcObjectInit` (`ADCDriver` *adcp)
Initializes the standard part of a `ADCDriver` structure.
- void `adcStart` (`ADCDriver` *adcp, const `ADCCConfig` *config)
Configures and activates the ADC peripheral.
- void `adcStop` (`ADCDriver` *adcp)
Deactivates the ADC peripheral.
- void `adcStartConversion` (`ADCDriver` *adcp, const `ADCCConversionGroup` *grpp, `adcsample_t` *samples, `size_t` depth)
Starts an ADC conversion.
- void `adcStartConversionI` (`ADCDriver` *adcp, const `ADCCConversionGroup` *grpp, `adcsample_t` *samples, `size_t` depth)

- void `adcStopConversion` (ADCDriver *adcp)

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.

9.2.1 Detailed Description

ADC Driver macros and structures.

9.3 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.

9.3.1 Detailed Description

PLATFORM ADC subsystem low level driver source.

9.4 adc_lld.h File Reference

PLATFORM ADC subsystem low level driver header.

Data Structures

- struct [ADCConversionGroup](#)
Conversion group configuration structure.
- struct [ADCCConfig](#)
Driver configuration structure.
- struct [ADCDriver](#)
Structure representing an ADC driver.

Macros

PLATFORM configuration options

- #define [PLATFORM_ADC_USE_ADC1](#) FALSE
ADC1 driver enable switch.

Typedefs

- typedef uint16_t [adcsample_t](#)
ADC sample data type.
- typedef uint16_t [adc_channels_num_t](#)
Channels number in a conversion group.
- typedef struct [ADCDriver](#) [ADCDriver](#)
Type of a structure representing an ADC driver.
- typedef void(* [adccallback_t](#)) ([ADCDriver](#) *adcp, [adcsample_t](#) *buffer, size_t n)
ADC notification callback type.
- typedef void(* [adcerrorcallback_t](#)) ([ADCDriver](#) *adcp, [adcerror_t](#) err)
ADC error callback type.

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.

9.4.1 Detailed Description

PLATFORM ADC subsystem low level driver header.

9.5 can.c File Reference

CAN Driver code.

```
#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.
- **msg_t** **canTransmit** (**CANDriver** *canp, **canmbx_t** mailbox, const **CANTxFrame** *ctfp, **systime_t** timeout)
Can frame transmission.
- **msg_t** **canReceive** (**CANDriver** *canp, **canmbx_t** mailbox, **CANRxFrame** *crfp, **systime_t** timeout)
Can frame receive.
- void **canSleep** (**CANDriver** *canp)
Enters the sleep mode.
- void **canWakeup** (**CANDriver** *canp)
Enforces leaving the sleep mode.

9.5.1 Detailed Description

CAN Driver code.

9.6 can.h File Reference

CAN Driver macros and structures.

```
#include "can_lld.h"
```

Macros

- #define **CAN_ANY_MAILBOX** 0
Special mailbox identifier.

CAN status flags

- #define **CAN_LIMIT_WARNING** 1
Errors rate warning.
- #define **CAN_LIMIT_ERROR** 2
Errors rate error.
- #define **CAN_BUS_OFF_ERROR** 4
Bus off condition reached.
- #define **CAN_FRAMING_ERROR** 8
Framing error of some kind on the CAN bus.

- `#define CAN_OVERFLOW_ERROR 16`
Overflow in receive queue.

CAN configuration options

- `#define CAN_USE_SLEEP_MODE TRUE`
Sleep mode related APIs inclusion switch.

Macro Functions

- `#define CAN_MAILBOX_TO_MASK(mbx) (1 << ((mbx) - 1))`
Converts a mailbox index to a bit mask.

Enumerations

- enum `canstate_t` {

`CAN_UNINIT = 0, CAN_STOP = 1, CAN_STARTING = 2, CAN_READY = 3,`

`CAN_SLEEP = 4 }`

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.
- `msg_t canTransmit (CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp, systime_t timeout)`
Can frame transmission.
- `msg_t canReceive (CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp, systime_t timeout)`
Can frame receive.

9.6.1 Detailed Description

CAN Driver macros and structures.

9.7 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_sleep` (`CANDriver` *canp)
Enters the sleep mode.
- void `can_lld_wakeup` (`CANDriver` *canp)
Enforces leaving the sleep mode.

Variables

- `CANDriver CAND1`
CAN1 driver identifier.

9.7.1 Detailed Description

PLATFORM CAN subsystem low level driver source.

9.8 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 uint32_t canmbx_t`
Type of a transmission mailbox index.

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

9.8.1 Detailed Description

PLATFORM CAN subsystem low level driver header.

9.9 dac.c File Reference

DAC Driver code.

```
#include "hal.h"
```

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`, const `dacsample_t` *`samples`, `size_t` `depth`)

Starts a DAC conversion.
- void `dacStartConversionI` (`DACDriver` *`dacp`, const `DACConversionGroup` *`grpp`, const `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`, const `dacsample_t` *`samples`, `size_t` `depth`)

Performs a DAC conversion.
- void `dacAcquireBus` (`DACDriver` *`dacp`)

Gains exclusive access to the DAC bus.
- void `dacReleaseBus` (`DACDriver` *`dacp`)

Releases exclusive access to the DAC bus.

9.9.1 Detailed Description

DAC Driver code.

9.10 dac.h File Reference

DAC Driver macros and structures.

```
#include "dac_lld.h"
```

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 _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.

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 *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, const dacsample_t *samples, size_t depth)`
Starts a DAC conversion.
- `void dacStartConversionI (DACDriver *dacp, const DACConversionGroup *grpp, const 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.

9.10.1 Detailed Description

DAC Driver macros and structures.

9.11 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](#) *dapc)
Configures and activates the DAC peripheral.
- void [dac_lld_stop](#) ([DACDriver](#) *dapc)
Deactivates the DAC peripheral.
- void [dac_lld_put_channel](#) ([DACDriver](#) *dapc, [dacchannel_t](#) channel, [dacsample_t](#) sample)
Outputs a value directly on a DAC channel.
- void [dac_lld_start_conversion](#) ([DACDriver](#) *dapc)
Starts a DAC conversion.
- void [dac_lld_stop_conversion](#) ([DACDriver](#) *dapc)
Stops an ongoing conversion.

Variables

- [DACDriver DACD1](#)
DAC1 driver identifier.

9.11.1 Detailed Description

PLATFORM DAC subsystem low level driver source.

9.12 dac_lld.h File Reference

PLATFORM DAC subsystem low level driver header.

Data Structures

- struct [DACConversionGroup](#)
DAC Conversion group structure.
- struct [DACConfig](#)
Driver configuration structure.
- struct [DACDriver](#)
Structure representing a DAC driver.

Macros

- `#define DAC_MAX_CHANNELS 2`
Maximum number of DAC channels per unit.

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 struct DACDriver DACDriver`
Type of a structure representing an DAC driver.
- `typedef uint16_t dacsample_t`
Type representing a DAC sample.
- `typedef void(* daccallback_t) (DACDriver *dACP, const dacsample_t *buffer, size_t n)`
DAC notification callback type.
- `typedef void(* dacerrorcallback_t) (DACDriver *dACP, dacerror_t err)`
ADC error callback type.

Enumerations

- `enum dacerror_t { DAC_ERR_DMAFAILURE = 0, DAC_ERR_UNDERFLOW = 1 }`
Possible DAC failure causes.

Functions

- `void dac_llD_init (void)`
Low level DAC driver initialization.
- `void dac_llD_start (DACDriver *dACP)`
Configures and activates the DAC peripheral.
- `void dac_llD_stop (DACDriver *dACP)`
Deactivates the DAC peripheral.
- `void dac_llD_put_channel (DACDriver *dACP, dacchannel_t channel, dacsample_t sample)`
Outputs a value directly on a DAC channel.
- `void dac_llD_start_conversion (DACDriver *dACP)`
Starts a DAC conversion.
- `void dac_llD_stop_conversion (DACDriver *dACP)`
Stops an ongoing conversion.

9.12.1 Detailed Description

PLATFORM DAC subsystem low level driver header.

9.13 ext.c File Reference

EXT Driver code.

```
#include "hal.h"
```

Functions

- void `extInit` (void)
EXT Driver initialization.
- void `extObjectInit` (EXTDriver *extp)
Initializes the standard part of a `EXTDriver` structure.
- void `extStart` (EXTDriver *extp, const EXTConfig *config)
Configures and activates the EXT peripheral.
- void `extStop` (EXTDriver *extp)
Deactivates the EXT peripheral.
- void `extChannelEnable` (EXTDriver *extp, expchannel_t channel)
Enables an EXT channel.
- void `extChannelDisable` (EXTDriver *extp, expchannel_t channel)
Disables an EXT channel.
- void `extSetChannelModel` (EXTDriver *extp, expchannel_t channel, const EXTChannelConfig *extcp)
Changes the operation mode of a channel.

9.13.1 Detailed Description

EXT Driver code.

9.14 ext.h File Reference

EXT Driver macros and structures.

```
#include "ext_lld.h"
```

Macros

EXT channel modes

- #define `EXT_CH_MODE_EDGES_MASK` 3U
Mask of edges field.
- #define `EXT_CH_MODE_DISABLED` 0U
Channel disabled.
- #define `EXT_CH_MODE_RISING_EDGE` 1U
Rising edge callback.
- #define `EXT_CH_MODE_FALLING_EDGE` 2U
Falling edge callback.
- #define `EXT_CH_MODE_BOTH_EDGES` 3U
Both edges callback.
- #define `EXT_CH_MODE_AUTOSTART` 4U
Channel started automatically on driver start.

Macro Functions

- #define `extChannelEnable`(extp, channel) `ext_lld_channel_enable(extp, channel)`
Enables an EXT channel.
- #define `extChannelDisable`(extp, channel) `ext_lld_channel_disable(extp, channel)`
Disables an EXT channel.
- #define `extSetChannelMode`(extp, channel, extcp)
Changes the operation mode of a channel.

Typedefs

- typedef struct `EXTDriver` `EXTDriver`
Type of a structure representing a EXT driver.

Enumerations

- enum `extstate_t` { `EXT_UNINIT` = 0, `EXT_STOP` = 1, `EXT_ACTIVE` = 2 }
- Driver state machine possible states.*

Functions

- void `extInit` (void)
EXT Driver initialization.
- void `extObjectInit` (`EXTDriver` *extp)
Initializes the standard part of a `EXTDriver` structure.
- void `extStart` (`EXTDriver` *extp, const `EXTConfig` *config)
Configures and activates the EXT peripheral.
- void `extStop` (`EXTDriver` *extp)
Deactivates the EXT peripheral.
- void `extChannelEnable` (`EXTDriver` *extp, `expchannel_t` channel)
Enables an EXT channel.
- void `extChannelDisable` (`EXTDriver` *extp, `expchannel_t` channel)
Disables an EXT channel.
- void `extSetChannelModel` (`EXTDriver` *extp, `expchannel_t` channel, const `EXTChannelConfig` *extcp)
Changes the operation mode of a channel.

9.14.1 Detailed Description

EXT Driver macros and structures.

9.15 ext_lld.c File Reference

PLATFORM EXT subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void `ext_ll_init` (void)
Low level EXT driver initialization.
- void `ext_ll_start` (EXTDriver *extp)
Configures and activates the EXT peripheral.
- void `ext_ll_stop` (EXTDriver *extp)
Deactivates the EXT peripheral.
- void `ext_ll_channel_enable` (EXTDriver *extp, expchannel_t channel)
Enables an EXT channel.
- void `ext_ll_channel_disable` (EXTDriver *extp, expchannel_t channel)
Disables an EXT channel.

Variables

- EXTDriver EXTD1
EXT1 driver identifier.

9.15.1 Detailed Description

PLATFORM EXT subsystem low level driver source.

9.16 ext_ll.h File Reference

PLATFORM EXT subsystem low level driver header.

Data Structures

- struct `EXTChannelConfig`
Channel configuration structure.
- struct `EXTConfig`
Driver configuration structure.
- struct `EXTDriver`
Structure representing an EXT driver.

Macros

- #define `EXT_MAX_CHANNELS` 20
Available number of EXT channels.

PLATFORM configuration options

- #define `PLATFORM_EXT_USE_EXT1` FALSE
EXT driver enable switch.

Typedefs

- typedef uint32_t `expchannel_t`
EXT channel identifier.
- typedef void(* `extcallback_t`) (EXTDriver *extp, expchannel_t channel)
Type of an EXT generic notification callback.

Functions

- void `ext_lld_init` (void)
Low level EXT driver initialization.
- void `ext_lld_start` (EXTDriver *extp)
Configures and activates the EXT peripheral.
- void `ext_lld_stop` (EXTDriver *extp)
Deactivates the EXT peripheral.
- void `ext_lld_channel_enable` (EXTDriver *extp, expchannel_t channel)
Enables an EXT channel.
- void `ext_lld_channel_disable` (EXTDriver *extp, expchannel_t channel)
Disables an EXT channel.

9.16.1 Detailed Description

PLATFORM EXT subsystem low level driver header.

9.17 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 `gptStartContinuousl` (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 `gptStartOneShotl` (GPTDriver *gptp, gptcnt_t interval)
Starts the timer in one shot mode.
- void `gptStopTimer` (GPTDriver *gptp)
Stops the timer.
- void `gptStopTimerl` (GPTDriver *gptp)
Stops the timer.
- void `gptPolledDelay` (GPTDriver *gptp, gptcnt_t interval)
Starts the timer in one shot mode and waits for completion.

9.17.1 Detailed Description

GPT Driver code.

9.18 gpt.h File Reference

GPT Driver macros and structures.

```
#include "gpt_lld.h"
```

Macros

- `#define gptChangeInterval(gptp, interval)`
Changes the interval of GPT peripheral.
- `#define gptGetIntervalX(gptp) gpt_lld_get_interval(gptp)`
Returns the interval of GPT peripheral.
- `#define gptGetCounterX(gptp) gpt_lld_get_counter(gptp)`
Returns the counter value of GPT peripheral.

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 `gptStartOneShotl (GPTDriver *gptp, gptcnt_t interval)`
Starts the timer in one shot mode.
- void `gptStopTimer (GPTDriver *gptp)`
Stops the timer.
- void `gptStopTimerl (GPTDriver *gptp)`
Stops the timer.
- void `gptPolledDelay (GPTDriver *gptp, gptcnt_t interval)`
Starts the timer in one shot mode and waits for completion.

9.18.1 Detailed Description

GPT Driver macros and structures.

9.19 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.

9.19.1 Detailed Description

PLATFORM GPT subsystem low level driver source.

9.20 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.

9.20.1 Detailed Description

PLATFORM GPT subsystem low level driver header.

9.21 hal.c File Reference

HAL subsystem code.

```
#include "hal.h"
```

Functions

- void `halInit` (void)

HAL initialization.

9.21.1 Detailed Description

HAL subsystem code.

9.22 hal.h File Reference

HAL subsystem header.

```
#include "osal.h"
#include "board.h"
#include "halconf.h"
#include "hal_lld.h"
#include "hal_streams.h"
#include "hal_channels.h"
#include "hal_files.h"
#include "hal_ioblock.h"
#include "hal_mmcspi.h"
#include "hal_queues.h"
#include "pal.h"
#include "adc.h"
#include "can.h"
#include "dac.h"
#include "ext.h"
#include "gpt.h"
#include "i2c.h"
#include "i2s.h"
#include "icu.h"
#include "mac.h"
#include "mii.h"
#include "pwm.h"
#include "rtc.h"
#include "serial.h"
#include "sdc.h"
#include "spi.h"
#include "uart.h"
#include "usb.h"
#include "mmc_spi.h"
#include "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 "3.0.2"`
HAL version string.
- `#define CH_HAL_MAJOR 3`
HAL version major number.
- `#define CH_HAL_MINOR 0`
HAL version minor number.
- `#define CH_HAL_PATCH 2`
HAL version patch number.

Return codes

- `#define HAL_SUCCESS false`
- `#define HAL_FAILED true`

Functions

- `void halInit (void)`
HAL initialization.

9.22.1 Detailed Description

HAL subsystem header.

9.23 hal_channels.h File Reference

I/O channels access.

Data Structures

- struct `BaseChannelVMT`
BaseChannel virtual methods table.
- struct `BaseChannel`
Base channel class.
- struct `BaseAsynchronousChannelVMT`
BaseAsynchronousChannel virtual methods table.
- struct `BaseAsynchronousChannel`
Base asynchronous channel class.

Macros

- `#define _base_channel_methods`
BaseChannel specific methods.
- `#define _base_channel_data _base_sequential_stream_data`
BaseChannel specific data.
- `#define _base_asynchronous_channel_methods _base_channel_methods \`
BaseAsynchronousChannel specific methods.
- `#define _base_asynchronous_channel_data`
BaseAsynchronousChannel specific data.

Macro Functions (BaseChannel)

- #define `chnPutTimeout(ip, b, time)` ((ip)->vmt->putt(ip, b, time))
Channel blocking byte write with timeout.
- #define `chnGetTimeout(ip, time)` ((ip)->vmt->gett(ip, time))
Channel blocking byte read with timeout.
- #define `chnWrite(ip, bp, n)` `streamWrite(ip, bp, n)`
Channel blocking write.
- #define `chnWriteTimeout(ip, bp, n, time)` ((ip)->vmt->writet(ip, bp, n, time))
Channel blocking write with timeout.
- #define `chnRead(ip, bp, n)` `streamRead(ip, bp, n)`
Channel blocking read.
- #define `chnReadTimeout(ip, bp, n, time)` ((ip)->vmt->readt(ip, bp, n, time))
Channel blocking read with timeout.

I/O status flags added to the event listener

- #define `CHN_NO_ERROR` (`eventflags_t`)0
No pending conditions.
- #define `CHN_CONNECTED` (`eventflags_t`)1
Connection happened.
- #define `CHN_DISCONNECTED` (`eventflags_t`)2
Disconnection happened.
- #define `CHN_INPUT_AVAILABLE` (`eventflags_t`)4
Data available in the input queue.
- #define `CHN_OUTPUT_EMPTY` (`eventflags_t`)8
Output queue empty.
- #define `CHN_TRANSMISSION_END` (`eventflags_t`)16
Transmission end.

Macro Functions (BaseAsynchronousChannel)

- #define `chnGetEventSource(ip)` (&((ip)->event))
Returns the I/O condition event source.
- #define `chnAddFlags1(ip, flags)`
Adds status flags to the listeners's flags mask.

9.23.1 Detailed Description

I/O channels access.

This header defines an abstract interface useful to access generic I/O serial devices in a standardized way.

9.24 hal_files.h File Reference

Data files.

Data Structures

- struct `FileStreamVMT`
FileStream virtual methods table.
- struct `FileStream`
Base file stream class.

Macros

- `#define _file_stream_methods`
FileStream specific methods.
- `#define _file_stream_data_base_sequential_stream_data`
FileStream specific data.

Files return codes

- `#define FILE_OK STM_OK`
No error return code.
- `#define FILE_ERROR STM_TIMEOUT`
Error code from the file stream methods.
- `#define FILE_EOF STM_RESET`
End-of-file condition for file get/put methods.

Macro Functions (FileStream)

- `#define fileStreamWrite(ip, bp, n) streamWrite(ip, bp, n)`
File stream write.
- `#define fileStreamRead(ip, bp, n) streamRead(ip, bp, n)`
File stream read.
- `#define fileStreamPut(ip, b) streamPut(ip, b)`
File stream blocking byte write.
- `#define fileStreamGet(ip) streamGet(ip)`
File stream blocking byte read.
- `#define fileStreamClose(ip) ((ip)->vmt->close(ip))`
File Stream close.
- `#define fileStreamGetError(ip) ((ip)->vmt->geterror(ip))`
Returns an implementation dependent error code.
- `#define fileStreamGetSize(ip) ((ip)->vmt->getsize(ip))`
Returns the current file size.
- `#define fileStreamGetPosition(ip) ((ip)->vmt->getposition(ip))`
Returns the current file pointer position.
- `#define fileStreamSeek(ip, offset) ((ip)->vmt->lseek(ip, offset))`
Moves the file current pointer to an absolute position.

Typedefs

- `typedef uint32_t fileoffset_t`
File offset type.

9.24.1 Detailed Description

Data files.

This header defines abstract interfaces useful to access generic data files in a standardized way.

9.25 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.

9.25.1 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.

9.26 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.

9.26.1 Detailed Description

PLATFORM HAL subsystem low level driver source.

9.27 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.

9.27.1 Detailed Description

PLATFORM HAL subsystem low level driver header.

9.28 hal_mmc.c File Reference

MMC/SD cards common code.

```
#include "hal.h"
```

Functions

- uint32_t [_mmc_get_slice](#) (const uint32_t *data, uint32_t end, uint32_t start)
Gets a bit field from a words array.
- uint32_t [_mmc_get_capacity](#) (const uint32_t *csd)
Extract card capacity from a CSD.
- uint32_t [_mmc_get_capacity_ext](#) (const uint8_t *ext_csd)

- Extract MMC card capacity from EXT_CSD.
- void `_mmcsd_unpack_sdc_cid` (const `MMCSDBlockDevice` *`sdc`, `unpacked_sdc_cid_t` *`cidsdc`)
Unpacks SDC CID array in structure.
- void `_mmcsd_unpack_mmc_cid` (const `MMCSDBlockDevice` *`sdc`, `unpacked_mmc_cid_t` *`cidmmc`)
Unpacks MMC CID array in structure.
- void `_mmcsd_unpack_csd_mmc` (const `MMCSDBlockDevice` *`sdc`, `unpacked_mmc_csd_t` *`csdmmc`)
Unpacks MMC CSD array in structure.
- void `_mmcsd_unpack_csd_v10` (const `MMCSDBlockDevice` *`sdc`, `unpacked_sdc_csd_10_t` *`csd10`)
Unpacks SDC CSD v1.0 array in structure.
- void `_mmcsd_unpack_csd_v20` (const `MMCSDBlockDevice` *`sdc`, `unpacked_sdc_csd_20_t` *`csd20`)
Unpacks SDC CSD v2.0 array in structure.

9.28.1 Detailed Description

MMC/SD cards common code.

9.29 hal_mmcsd.h File Reference

MMC/SD cards common header.

Data Structures

- struct `MMCSDBlockDeviceVMT`
`MMCSDBlockDevice` virtual methods table.
- struct `MMCSDBlockDevice`
MCC/SD block device class.
- struct `unpacked_sdc_cid_t`
Unpacked CID register from SDC.
- struct `unpacked_mmc_cid_t`
Unpacked CID register from MMC.
- struct `unpacked_sdc_csd_10_t`
Unpacked CSD v1.0 register from SDC.
- struct `unpacked_sdc_csd_20_t`
Unpacked CSD v2.0 register from SDC.
- struct `unpacked_mmc_csd_t`
Unpacked CSD register from MMC.

Macros

- #define `MMCSD_BLOCK_SIZE` 512U
Fixed block size for MMC/SD block devices.
- #define `MMCSD_R1_ERROR_MASK` 0xFDFFE008U
Mask of error bits in R1 responses.
- #define `MMCSD_CMD8_PATTERN` 0x000001AAU
Fixed pattern for CMD8.
- #define `_mmcsd_block_device_methods` `_base_block_device_methods`
`MMCSDBlockDevice` specific methods.
- #define `_mmcsd_block_device_data`
`MMCSDBlockDevice` specific data.

SD/MMC status conditions

- #define **MMCSD_STS_IDLE** 0U
- #define **MMCSD_STS_READY** 1U
- #define **MMCSD_STS_IDENT** 2U
- #define **MMCSD_STS_STBY** 3U
- #define **MMCSD_STS_TRAN** 4U
- #define **MMCSD_STS_DATA** 5U
- #define **MMCSD_STS_RCV** 6U
- #define **MMCSD_STS_PRG** 7U
- #define **MMCSD_STS_DIS** 8U

SD/MMC commands

- #define **MMCSD_CMD_GO_IDLE_STATE** 0U
- #define **MMCSD_CMD_INIT** 1U
- #define **MMCSD_CMD_ALL_SEND_CID** 2U
- #define **MMCSD_CMD_SEND_RELATIVE_ADDR** 3U
- #define **MMCSD_CMD_SET_BUS_WIDTH** 6U
- #define **MMCSD_CMD_SWITCH** MMCSD_CMD_SET_BUS_WIDTH
- #define **MMCSD_CMD_SEL_DESEL_CARD** 7U
- #define **MMCSD_CMD_SEND_IF_COND** 8U
- #define **MMCSD_CMD_SEND_EXT_CSD** MMCSD_CMD_SEND_IF_COND
- #define **MMCSD_CMD_SEND_CSD** 9U
- #define **MMCSD_CMD_SEND_CID** 10U
- #define **MMCSD_CMD_STOP_TRANSMISSION** 12U
- #define **MMCSD_CMD_SEND_STATUS** 13U
- #define **MMCSD_CMD_SET_BLOCKLEN** 16U
- #define **MMCSD_CMD_READ_SINGLE_BLOCK** 17U
- #define **MMCSD_CMD_READ_MULTIPLE_BLOCK** 18U
- #define **MMCSD_CMD_SET_BLOCK_COUNT** 23U
- #define **MMCSD_CMD_WRITE_BLOCK** 24U
- #define **MMCSD_CMD_WRITE_MULTIPLE_BLOCK** 25U
- #define **MMCSD_CMD_ERASE_RW_BLK_START** 32U
- #define **MMCSD_CMD_ERASE_RW_BLK_END** 33U
- #define **MMCSD_CMD_ERASE** 38U
- #define **MMCSD_CMD_APP_OP_COND** 41U
- #define **MMCSD_CMD_LOCK_UNLOCK** 42U
- #define **MMCSD_CMD_APP_CMD** 55U
- #define **MMCSD_CMD_READ_OCR** 58U

CSD record offsets

- #define **MMCSD_CSD_MMC_CSD_STRUCTURE_SLICE** 127U,126U
Slice position of values in CSD register.
- #define **MMCSD_CSD_MMC_SPEC_VERS_SLICE** 125U,122U
- #define **MMCSD_CSD_MMC_TAAC_SLICE** 119U,112U
- #define **MMCSD_CSD_MMC_NSAC_SLICE** 111U,104U
- #define **MMCSD_CSD_MMC_TRAN_SPEED_SLICE** 103U,96U
- #define **MMCSD_CSD_MMC_CCC_SLICE** 95U,84U
- #define **MMCSD_CSD_MMC_READ_BL_LEN_SLICE** 83U,80U
- #define **MMCSD_CSD_MMC_READ_BL_PARTIAL_SLICE** 79U,79U
- #define **MMCSD_CSD_MMC_WRITE_BLK_MISALIGN_SLICE** 78U,78U
- #define **MMCSD_CSD_MMC_READ_BLK_MISALIGN_SLICE** 77U,77U
- #define **MMCSD_CSD_MMC_DSR_IMP_SLICE** 76U,76U
- #define **MMCSD_CSD_MMC_C_SIZE_SLICE** 73U,62U
- #define **MMCSD_CSD_MMC_VDD_R_CURR_MIN_SLICE** 61U,59U
- #define **MMCSD_CSD_MMC_VDD_R_CURR_MAX_SLICE** 58U,56U
- #define **MMCSD_CSD_MMC_VDD_W_CURR_MIN_SLICE** 55U,53U
- #define **MMCSD_CSD_MMC_VDD_W_CURR_MAX_SLICE** 52U,50U
- #define **MMCSD_CSD_MMC_C_SIZE_MULT_SLICE** 49U,47U
- #define **MMCSD_CSD_MMC_ERASE_GRP_SIZE_SLICE** 46U,42U
- #define **MMCSD_CSD_MMC_ERASE_GRP_MULT_SLICE** 41U,37U

- #define **MMCSD_CSD_MMC_WP_GRP_SIZE_SLICE** 36U,32U
- #define **MMCSD_CSD_MMC_WP_GRP_ENABLE_SLICE** 31U,31U
- #define **MMCSD_CSD_MMC_DEFAULT_ECC_SLICE** 30U,29U
- #define **MMCSD_CSD_MMC_R2W_FACTOR_SLICE** 28U,26U
- #define **MMCSD_CSD_MMC_WRITE_BL_LEN_SLICE** 25U,22U
- #define **MMCSD_CSD_MMC_WRITE_BL_PARTIAL_SLICE** 21U,21U
- #define **MMCSD_CSD_MMC_CONTENT_PROT_APP_SLICE** 16U,16U
- #define **MMCSD_CSD_MMC_FILE_FORMAT_GRP_SLICE** 15U,15U
- #define **MMCSD_CSD_MMC_COPY_SLICE** 14U,14U
- #define **MMCSD_CSD_MMC_PERM_WRITE_PROTECT_SLICE** 13U,13U
- #define **MMCSD_CSD_MMC_TMP_WRITE_PROTECT_SLICE** 12U,12U
- #define **MMCSD_CSD_MMC_FILE_FORMAT_SLICE** 11U,10U
- #define **MMCSD_CSD_MMC_ECC_SLICE** 9U,8U
- #define **MMCSD_CSD_MMC_CRC_SLICE** 7U,1U
- #define **MMCSD_CSD_20_CRC_SLICE** 7U,1U
- #define **MMCSD_CSD_20_FILE_FORMAT_SLICE** 11U,10U
- #define **MMCSD_CSD_20_TMP_WRITE_PROTECT_SLICE** 12U,12U
- #define **MMCSD_CSD_20_PERM_WRITE_PROTECT_SLICE** 13U,13U
- #define **MMCSD_CSD_20_COPY_SLICE** 14U,14U
- #define **MMCSD_CSD_20_FILE_FORMAT_GRP_SLICE** 15U,15U
- #define **MMCSD_CSD_20_WRITE_BL_PARTIAL_SLICE** 21U,21U
- #define **MMCSD_CSD_20_WRITE_BL_LEN_SLICE** 25U,12U
- #define **MMCSD_CSD_20_R2W_FACTOR_SLICE** 28U,26U
- #define **MMCSD_CSD_20_WP_GRP_ENABLE_SLICE** 31U,31U
- #define **MMCSD_CSD_20_WP_GRP_SIZE_SLICE** 38U,32U
- #define **MMCSD_CSD_20_ERASE_SECTOR_SIZE_SLICE** 45U,39U
- #define **MMCSD_CSD_20_ERASE_BLK_EN_SLICE** 46U,46U
- #define **MMCSD_CSD_20_C_SIZE_SLICE** 69U,48U
- #define **MMCSD_CSD_20_DSR_IMP_SLICE** 76U,76U
- #define **MMCSD_CSD_20_READ_BLK_MISALIGN_SLICE** 77U,77U
- #define **MMCSD_CSD_20_WRITE_BLK_MISALIGN_SLICE** 78U,78U
- #define **MMCSD_CSD_20_READ_BL_PARTIAL_SLICE** 79U,79U
- #define **MMCSD_CSD_20_READ_BL_LEN_SLICE** 83U,80U
- #define **MMCSD_CSD_20_CCC_SLICE** 95U,84U
- #define **MMCSD_CSD_20_TRANS_SPEED_SLICE** 103U,96U
- #define **MMCSD_CSD_20_NSAC_SLICE** 111U,104U
- #define **MMCSD_CSD_20_TAAC_SLICE** 119U,112U
- #define **MMCSD_CSD_20_CSD_STRUCTURE_SLICE** 127U,126U
- #define **MMCSD_CSD_10_CRC_SLICE** MMCSD_CSD_20_CRC_SLICE
- #define **MMCSD_CSD_10_FILE_FORMAT_SLICE** MMCSD_CSD_20_FILE_FORMAT_SLICE
- #define **MMCSD_CSD_10_TMP_WRITE_PROTECT_SLICE** MMCSD_CSD_20_TMP_WRITE_PROTECT_SLICE
- #define **MMCSD_CSD_10_PERM_WRITE_PROTECT_SLICE** MMCSD_CSD_20_PERM_WRITE_PROTECT_SLICE
- #define **MMCSD_CSD_10_COPY_SLICE** MMCSD_CSD_20_COPY_SLICE
- #define **MMCSD_CSD_10_FILE_FORMAT_GRP_SLICE** MMCSD_CSD_20_FILE_FORMAT_GRP_SLICE
- #define **MMCSD_CSD_10_WRITE_BL_PARTIAL_SLICE** MMCSD_CSD_20_WRITE_BL_PARTIAL_SLICE
- #define **MMCSD_CSD_10_WRITE_BL_LEN_SLICE** MMCSD_CSD_20_WRITE_BL_LEN_SLICE
- #define **MMCSD_CSD_10_R2W_FACTOR_SLICE** MMCSD_CSD_20_R2W_FACTOR_SLICE
- #define **MMCSD_CSD_10_WP_GRP_ENABLE_SLICE** MMCSD_CSD_20_WP_GRP_ENABLE_SLICE
- #define **MMCSD_CSD_10_WP_GRP_SIZE_SLICE** MMCSD_CSD_20_WP_GRP_SIZE_SLICE
- #define **MMCSD_CSD_10_ERASE_SECTOR_SIZE_SLICE** MMCSD_CSD_20_ERASE_SECTOR_SIZE_SLICE
- #define **MMCSD_CSD_10_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_MISALIG↔
N_SLICE
- #define **MMCSD_CSD_10_WRITE_BLK_MISALIGN_SLICE** MMCSD_CSD_20_WRITE_BLK_MISALI↔
GN_SLICE
- #define **MMCSD_CSD_10_READ_BL_PARTIAL_SLICE** MMCSD_CSD_20_READ_BL_PARTIAL_SLI↔
CE
- #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
Slice position of values in CID register.
- #define **MMCSD_CID_SDC_MDT_M_SLICE** 11U,8U
- #define **MMCSD_CID_SDC_MDT_Y_SLICE** 19U,12U
- #define **MMCSD_CID_SDC_PSN_SLICE** 55U,24U
- #define **MMCSD_CID_SDC_PRV_M_SLICE** 59U,56U
- #define **MMCSD_CID_SDC_PRV_N_SLICE** 63U,60U
- #define **MMCSD_CID_SDC_PNM0_SLICE** 71U,64U
- #define **MMCSD_CID_SDC_PNM1_SLICE** 79U,72U
- #define **MMCSD_CID_SDC_PNM2_SLICE** 87U,80U
- #define **MMCSD_CID_SDC_PNM3_SLICE** 95U,88U
- #define **MMCSD_CID_SDC_PNM4_SLICE** 103U,96U
- #define **MMCSD_CID_SDC_OID_SLICE** 119U,104U
- #define **MMCSD_CID_SDC_MID_SLICE** 127U,120U
- #define **MMCSD_CID_MM_CRC_SLICE** 7U,1U
- #define **MMCSD_CID_MM_MDT_Y_SLICE** 11U,8U
- #define **MMCSD_CID_MM_MDT_M_SLICE** 15U,12U
- #define **MMCSD_CID_MM_PSN_SLICE** 47U,16U
- #define **MMCSD_CID_MM_PRV_M_SLICE** 51U,48U
- #define **MMCSD_CID_MM_PRV_N_SLICE** 55U,52U
- #define **MMCSD_CID_MM_PNM0_SLICE** 63U,56U
- #define **MMCSD_CID_MM_PNM1_SLICE** 71U,64U
- #define **MMCSD_CID_MM_PNM2_SLICE** 79U,72U
- #define **MMCSD_CID_MM_PNM3_SLICE** 87U,80U
- #define **MMCSD_CID_MM_PNM4_SLICE** 95U,88U
- #define **MMCSD_CID_MM_PNM5_SLICE** 103U,96U
- #define **MMCSD_CID_MM_OID_SLICE** 119U,104U
- #define **MMCSD_CID_MM_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.

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.

9.29.1 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.

9.30 hal_queues.c File Reference

I/O Queues code.

```
#include "hal.h"
```

Functions

- `void iqObjectInit (input_queue_t *iqp, uint8_t *bp, size_t size, qnotify_t infy, void *link)`
Initializes an input queue.
- `void iqResetl (input_queue_t *iqp)`
Resets an input queue.
- `msg_t iqPutl (input_queue_t *iqp, uint8_t b)`
Input queue write.
- `msg_t iqGetTimeout (input_queue_t *iqp, systime_t timeout)`
Input queue read with timeout.
- `size_t iqReadTimeout (input_queue_t *iqp, uint8_t *bp, size_t n, systime_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 oqResetl (output_queue_t *oqp)`
Resets an output queue.
- `msg_t oqPutTimeout (output_queue_t *oqp, uint8_t b, systime_t timeout)`
Output queue write with timeout.
- `msg_t oqGetl (output_queue_t *oqp)`

- `size_t oqWriteTimeout (output_queue_t *oqp, const uint8_t *bp, size_t n, systime_t timeout)`
Output queue write with timeout.

9.30.1 Detailed Description

I/O Queues code.

9.31 hal_queues.h File Reference

I/O Queues macros and structures.

Data Structures

- `struct io_queue`
Generic I/O queue structure.

Macros

- `#define _INPUTQUEUE_DATA(name, buffer, size, inotify, link)`
Data part of a static input queue initializer.
- `#define INPUTQUEUE_DECL(name, buffer, size, inotify, link) input_queue_t name = _INPUTQUEUE_DA←
 TA(name, buffer, size, inotify, link)`
Static input queue initializer.
- `#define _OUTPUTQUEUE_DATA(name, buffer, size, onotify, link)`
Data part of a static output queue initializer.
- `#define OUTPUTQUEUE_DECL(name, buffer, size, onotify, link) output_queue_t name = _OUTPUTQUE←
 UE_DATA(name, buffer, size, onotify, link)`
Static output queue initializer.

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_t)-3`
Queue empty.
- `#define Q_FULL (msg_t)-4`
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 iqGetFullI(iqp) qSpaceI(iqp)`

- `#define iqGetEmpty(iqp) (qSizeX(iqp) - qSpaceL(iqp))`

Returns the filled space into an input queue.
- `#define iqIsEmpty(iqp) ((bool)(qSpaceL(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) - qSpaceL(oqp))`

Returns the filled space into an output queue.
- `#define oqIsEmpty(oqp) qSpaceL(oqp)`

Returns the empty space into an output queue.
- `#define oqIsFull(oqp)`

Evaluates to true if the specified output queue is empty.
- `#define oqGetTimeout(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 iqResetl (input_queue_t *iqp)`

Resets an input queue.
- `msg_t iqPutl (input_queue_t *iqp, uint8_t b)`

Input queue write.
- `msg_t iqGetTimeout (input_queue_t *iqp, systime_t timeout)`

Input queue read with timeout.
- `size_t iqReadTimeout (input_queue_t *iqp, uint8_t *bp, size_t n, systime_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 oqResetl (output_queue_t *oqp)`

Resets an output queue.
- `msg_t oqPutTimeout (output_queue_t *oqp, uint8_t b, systime_t timeout)`

Output queue write with timeout.
- `msg_t oqGetl (output_queue_t *oqp)`

Output queue read.
- `size_t oqWriteTimeout (output_queue_t *oqp, const uint8_t *bp, size_t n, systime_t timeout)`

Output queue write with timeout.

9.31.1 Detailed Description

I/O Queues macros and structures.

9.32 hal_streams.h File Reference

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`
`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.

9.32.1 Detailed Description

Data streams.

This header defines abstract interfaces useful to access generic data streams in a standardized way.

9.33 halconf.h File Reference

HAL configuration header.

```
#include "mcuconf.h"
```

Macros

Drivers enable switches

- #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_DAC` FALSE
Enables the DAC subsystem.
- #define `HAL_USE_EXT` TRUE
Enables the EXT 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_SPI` TRUE
Enables the SPI subsystem.
- #define `HAL_USE_UART` TRUE
Enables the UART subsystem.
- #define `HAL_USE_USB` TRUE
Enables the USB subsystem.

ADC driver related setting

- #define `ADC_USE_WAIT` TRUE
Enables synchronous APIs.
- #define `ADC_USE_MUTUAL_EXCLUSION` TRUE
Enables the `adcAcquireBus()` and `adcReleaseBus()` APIs.

CAN driver related setting

- #define `CAN_USE_SLEEP_MODE` TRUE
Sleep mode related APIs inclusion switch.

I2C driver related setting

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

MAC driver related setting

- #define `MAC_USE_ZERO_COPY` TRUE
Enables an event sources for incoming packets.
- #define `MAC_USE_EVENTS` TRUE
Enables an event sources for incoming packets.

MMC_SPI driver related setting

- #define `MMC_NICE_WAITING` TRUE
Delays insertions.

SDC driver related setting

- #define `SDC_INIT_RETRY` 100
Number of initialization attempts before rejecting the card.
- #define `SDC_MMCC SUPPORT` TRUE
Include support for MMC cards.
- #define `SDC_NICE_WAITING` TRUE
Delays insertions.

SERIAL driver related setting

- #define `SERIAL_DEFAULT_BITRATE` 38400
Default bit rate.
- #define `SERIAL_BUFFERS_SIZE` 16
Serial buffers size.

SERIAL_USB driver related setting

- #define `SERIAL_USB_BUFFERS_SIZE` 256
Serial over USB buffers size.

SPI driver related setting

- #define `SPI_USE_WAIT` TRUE
Enables synchronous APIs.
- #define `SPI_USE_MUTUAL_EXCLUSION` TRUE
Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.

9.33.1 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.

9.34 i2c.c File Reference

I2C Driver code.

```
#include "hal.h"
```

Functions

- void `i2cInit` (void)

I2C Driver initialization.
- void `i2cObjectInit` (`I2CDriver` *`i2cp`)

Initializes the standard part of a `I2CDriver` structure.
- void `i2cStart` (`I2CDriver` *`i2cp`, const `I2CConfig` *`config`)

Configures and activates the I2C peripheral.
- void `i2cStop` (`I2CDriver` *`i2cp`)

Deactivates the I2C peripheral.
- `i2cflags_t i2cGetErrors` (`I2CDriver` *`i2cp`)

Returns the errors mask associated to the previous operation.
- `msg_t i2cMasterTransmitTimeout` (`I2CDriver` *`i2cp`, `i2caddr_t` `addr`, const `uint8_t` *`txbuf`, `size_t` `txbytes`, `uint8_t` *`rxbuf`, `size_t` `rxbytes`, `systime_t` `timeout`)

Sends data via the I2C bus.
- `msg_t i2cMasterReceiveTimeout` (`I2CDriver` *`i2cp`, `i2caddr_t` `addr`, `uint8_t` *`rxbuf`, `size_t` `rxbytes`, `systime_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.

9.34.1 Detailed Description

I2C Driver code.

9.35 i2c.h File Reference

I2C Driver macros and structures.

```
#include "i2c_lld.h"
```

Macros

- `#define I2C_USE_MUTUAL_EXCLUSION TRUE`

Enables the mutual exclusion APIs on the I2C bus.
- `#define _i2c_wakeup_isr(i2cp)`

Wakes up the waiting thread notifying no errors.
- `#define _i2c_wakeup_error_isr(i2cp)`

Wakes up the waiting thread notifying errors.
- `#define i2cMasterTransmit(i2cp, addr, txbuf, txbytes, rxbuf, rxbytes)`

Wrap i2cMasterTransmit function with TIME_INFINITE timeout.
- `#define i2cMasterReceive(i2cp, addr, rxbuf, rxbytes) (i2cMasterReceiveTimeout(i2cp, addr, rxbuf, rxbytes, TIME_INFINITE))`

Wrap i2cMasterReceive function with TIME_INFINITE timeout.

I2C bus error conditions

- `#define I2C_NO_ERROR 0x00`

- `#define I2C_BUS_ERROR 0x01`
No error.
- `#define I2C_ARBITRATION_LOST 0x02`
Bus Error.
- `#define I2C_ACK_FAILURE 0x04`
Arbitration Lost.
- `#define I2C_OVERRUN 0x08`
Acknowledge Failure.
- `#define I2C_PEC_ERROR 0x10`
Overrun/Underrun.
- `#define I2C_TIMEOUT 0x20`
PEC Error in reception.
- `#define I2C_SMB_ALERT 0x40`
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 }`

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 *rdbuf, size_t rxbytes, systime_t timeout)`
Sends data via the I2C bus.
- `msg_t i2cMasterReceiveTimeout (I2CDriver *i2cp, i2caddr_t addr, uint8_t *rdbuf, size_t rxbytes, systime_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 macros and structures.

9.36 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 *rdbuf, size_t rxbytes, systime_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 *rdbuf, size_t rxbytes, systime_t timeout)
Transmits data via the I2C bus as master.

Variables

- I2CDriver `I2CD1`
I2C1 driver identifier.

9.36.1 Detailed Description

PLATFORM I2C subsystem low level driver source.

9.37 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 *rdbuf, size_t rxbytes, systime_t timeout)`
Transmits data via the I2C bus as master.
- `msg_t i2c_lld_master_receive_timeout (I2CDriver *i2cp, i2caddr_t addr, uint8_t *rdbuf, size_t rxbytes, systime_t timeout)`
Receives data via the I2C bus as master.

9.37.1 Detailed Description

PLATFORM I2C subsystem low level driver header.

9.38 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.

9.38.1 Detailed Description

I2S Driver code.

9.39 i2s.h File Reference

I2S Driver macros and structures.

```
#include "i2s_llld.h"
```

Macros

I2S modes

- #define **I2S_MODE_SLAVE** 0
- #define **I2S_MODE_MASTER** 1

Macro Functions

- #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.

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.

9.39.1 Detailed Description

I2S Driver macros and structures.

9.40 i2s_lld.c File Reference

PLATFORM I2S subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void [i2s_lld_init](#) (void)
Low level I2S driver initialization.
- void [i2s_lld_start](#) (I2SDriver *i2sp)
Configures and activates the I2S peripheral.

Variables

- I2SDriver I2SD1
I2S2 driver identifier.

9.40.1 Detailed Description

PLATFORM I2S subsystem low level driver source.

9.41 i2s_lld.h File Reference

PLATFORM I2S subsystem low level driver header.

Data Structures

- struct [I2SConfig](#)
Driver configuration structure.
- struct [I2SDriver](#)
Structure representing an I2S driver.

Macros

PLATFORM configuration options

- #define [PLATFORM_I2S_USE_I2S1](#) FALSE
I2SD1 driver enable switch.

Typedefs

- **typedef struct I2SDriver I2SDriver**
Type of a structure representing an I2S driver.
- **typedef void(* i2scallback_t) (I2SDriver *i2sp, size_t offset, size_t n)**
I2S notification callback type.

Functions

- **void i2s_lld_init (void)**
Low level I2S driver initialization.
- **void i2s_lld_start (I2SDriver *i2sp)**
Configures and activates the I2S peripheral.

9.41.1 Detailed Description

PLATFORM I2S subsystem low level driver header.

9.42 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.

9.42.1 Detailed Description

ICU Driver code.

9.43 icu.h File Reference

ICU Driver macros and structures.

```
#include "icu_lld.h"
```

Macros

Macro Functions

- #define `icuStartCaptureI(icup)`
Starts the input capture.
- #define `icuStopCaptureI(icup)`
Stops the input capture.
- #define `icuEnableNotificationsI(icup) icu_lld_enable_notifications(icup)`
Enables notifications.
- #define `icuDisableNotificationsI(icup) icu_lld_disable_notifications(icup)`
Disables notifications.
- #define `icuAreNotificationsEnabledX(icup) icu_lld_are_notifications_enabled(icup)`
Check on notifications status.
- #define `icuGetWidthX(icup) icu_lld_get_width(icup)`
Returns the width of the latest pulse.
- #define `icuGetPeriodX(icup) icu_lld_get_period(icup)`
Returns the width of the latest cycle.

Low level driver helper macros

- #define `_icu_isr_invoke_width_cb(icup)`
Common ISR code, ICU width event.
- #define `_icu_isr_invoke_period_cb(icup)`
Common ISR code, ICU period event.
- #define `_icu_isr_invoke_overflow_cb(icup)`
Common ISR code, ICU timer overflow event.

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`)

- 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 macros and structures.

9.44 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.

9.44.1 Detailed Description

PLATFORM ADC subsystem low level driver source.

9.45 icu_ll.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_ll_get_width**(icup) 0
Returns the width of the latest pulse.
- #define **icu_ll_get_period**(icup) 0
Returns the width of the latest cycle.
- #define **icu_ll_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_ll_init** (void)
Low level ICU driver initialization.
- void **icu_ll_start** (ICUDriver *icup)
Configures and activates the ICU peripheral.
- void **icu_ll_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.

9.45.1 Detailed Description

PLATFORM ICU subsystem low level driver header.

9.46 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, systime_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, systime_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.

9.46.1 Detailed Description

MAC Driver code.

9.47 mac.h File Reference

MAC Driver macros and structures.

```
#include "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)
Returns the received frames event source.
- #define `macWriteTransmitDescriptor`(tdp, buf, size) `mac_lld_write_transmit_descriptor`(tdp, buf, size)
Writes to a transmit descriptor's stream.
- #define `macReadReceiveDescriptor`(rdp, buf, size) `mac_lld_read_receive_descriptor`(rdp, buf, size)
Reads from a receive descriptor's stream.
- #define `macGetNextTransmitBuffer`(tdp, size, sizep) `mac_lld_get_next_transmit_buffer`(tdp, size, sizep)
Returns a pointer to the next transmit buffer in the descriptor chain.
- #define `macGetNextReceiveBuffer`(rdp, sizep) `mac_lld_get_next_receive_buffer`(rdp, sizep)
Returns a pointer to the next receive buffer in the descriptor chain.

Typedefs

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

Enumerations

- enum `macstate_t` { `MAC_UNINIT` = 0, `MAC_STOP` = 1, `MAC_ACTIVE` = 2 }
Driver state machine possible states.

Functions

- void `macInit` (void)
MAC Driver initialization.
- void `macObjectInit` (`MACDriver` *macp)
Initialize the standard part of a `MACDriver` structure.
- void `macStart` (`MACDriver` *macp, const `MACConfig` *config)
Configures and activates the MAC peripheral.
- void `macStop` (`MACDriver` *macp)
Deactivates the MAC peripheral.
- `msg_t` `macWaitTransmitDescriptor` (`MACDriver` *macp, `MACTransmitDescriptor` *tdp, `systime_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, systime_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.

9.47.1 Detailed Description

MAC Driver macros and structures.

9.48 mac_lld.c File Reference

PLATFORM MAC subsystem low level driver source.

```
#include <string.h>
#include "hal.h"
#include "mii.h"
```

Functions

- `void mac_lld_init (void)`
Low level MAC initialization.
- `void mac_lld_start (MACDriver *macp)`
Configures and activates the MAC peripheral.
- `void mac_lld_stop (MACDriver *macp)`
Deactivates the MAC peripheral.
- `msg_t mac_lld_get_transmit_descriptor (MACDriver *macp, MACTransmitDescriptor *tdp)`
Returns a transmission descriptor.
- `void mac_lld_release_transmit_descriptor (MACTransmitDescriptor *tdp)`
Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.
- `msg_t mac_lld_get_receive_descriptor (MACDriver *macp, MACReceiveDescriptor *rdp)`
Returns a receive descriptor.
- `void mac_lld_release_receive_descriptor (MACReceiveDescriptor *rdp)`
Releases a receive descriptor.
- `bool mac_lld_poll_link_status (MACDriver *macp)`
Updates and returns the link status.
- `size_t mac_lld_write_transmit_descriptor (MACTransmitDescriptor *tdp, uint8_t *buf, size_t size)`
Writes to a transmit descriptor's stream.
- `size_t mac_lld_read_receive_descriptor (MACReceiveDescriptor *rdp, uint8_t *buf, size_t size)`
Reads from a receive descriptor's stream.
- `uint8_t * mac_lld_get_next_transmit_buffer (MACTransmitDescriptor *tdp, size_t size, size_t *sizep)`
Returns a pointer to the next transmit buffer in the descriptor chain.
- `const uint8_t * mac_lld_get_next_receive_buffer (MACReceiveDescriptor *rdp, size_t *sizep)`
Returns a pointer to the next receive buffer in the descriptor chain.

Variables

- `MACDriver ETHD1`
MAC1 driver identifier.

9.48.1 Detailed Description

PLATFORM MAC subsystem low level driver source.

9.49 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)

- `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.

9.49.1 Detailed Description

PLATFORM MAC subsystem low level driver header.

9.50 mmc_spi.c File Reference

MMC over SPI driver code.

```
#include <string.h>
#include "hal.h"
```

Functions

- `static uint8_t crc7 (uint8_t crc, const uint8_t *buffer, size_t len)`
Calculate the MMC standard CRC-7 based on a lookup table.
- `static void wait (MMCDriver *mmcp)`
Waits an idle condition.
- `static void send_hdr (MMCDriver *mmcp, uint8_t cmd, uint32_t arg)`
Sends a command header.
- `static uint8_t 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 and 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 mmclInit (void)`
MMC over SPI driver initialization.
- `void mmcObjectInit (MMCDriver *mmcp)`
Initializes an instance.
- `void mmcStart (MMCDriver *mmcp, const MMConfig *config)`
Configures and activates the MMC peripheral.
- `void mmcStop (MMCDriver *mmcp)`
Disables the MMC peripheral.
- `bool mmcConnect (MMCDriver *mmcp)`
Performs the initialization procedure on the inserted card.
- `bool mmcDisconnect (MMCDriver *mmcp)`
Brings the driver in a state safe for card removal.

- bool `mmcStartSequentialRead (MMCDriver *mmcp, uint32_t startblk)`
Starts a sequential read.
- bool `mmcSequentialRead (MMCDriver *mmcp, uint8_t *buffer)`
Reads a block within a sequential read operation.
- bool `mmcStopSequentialRead (MMCDriver *mmcp)`
Stops a sequential read gracefully.
- bool `mmcStartSequentialWrite (MMCDriver *mmcp, uint32_t startblk)`
Starts a sequential write.
- 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 $x^7 + x^3 + 1$).

9.50.1 Detailed Description

MMC over SPI driver code.

9.51 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(mmc) mmc_lld_is_card_inserted(mmc)`
Returns the card insertion status.
- `#define mmcIsWriteProtected(mmc) mmc_lld_is_write_protected(mmc)`
Returns the write protect status.

Functions

- `void mmcInit (void)`
MMC over SPI driver initialization.
- `void mmcObjectInit (MMCDriver *mmc)`
Initializes an instance.
- `void mmcStart (MMCDriver *mmc, const MMConfig *config)`
Configures and activates the MMC peripheral.
- `void mmcStop (MMCDriver *mmc)`
Disables the MMC peripheral.
- `bool mmcConnect (MMCDriver *mmc)`
Performs the initialization procedure on the inserted card.
- `bool mmcDisconnect (MMCDriver *mmc)`
Brings the driver in a state safe for card removal.
- `bool mmcStartSequentialRead (MMCDriver *mmc, uint32_t startblk)`
Starts a sequential read.
- `bool mmcSequentialRead (MMCDriver *mmc, uint8_t *buffer)`
Reads a block within a sequential read operation.
- `bool mmcStopSequentialRead (MMCDriver *mmc)`
Stops a sequential read gracefully.
- `bool mmcStartSequentialWrite (MMCDriver *mmc, uint32_t startblk)`
Starts a sequential write.
- `bool mmcSequentialWrite (MMCDriver *mmc, const uint8_t *buffer)`
Writes a block within a sequential write operation.
- `bool mmcStopSequentialWrite (MMCDriver *mmc)`
Stops a sequential write gracefully.
- `bool mmcSync (MMCDriver *mmc)`
Waits for card idle condition.
- `bool mmcGetInfo (MMCDriver *mmc, BlockDeviceInfo *bdip)`
Returns the media info.
- `bool mmcErase (MMCDriver *mmc, uint32_t startblk, uint32_t endblk)`
Erases blocks.

9.51.1 Detailed Description

MMC over SPI driver header.

9.52 osal.c File Reference

OSAL module code.

```
#include "osal.h"
```

Functions

- void **osallInit** (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** (systime_t time)

Suspends the invoking thread for the specified time.
- void **osalThreadSleep** (systime_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, systime_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, systime_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.

9.52.1 Detailed Description

OSAL module code.

9.53 osal.h File Reference

OSAL module header.

```
#include <stddef.h>
#include <stdint.h>
#include <stdbool.h>
```

Data Structures

- struct **event_source**
Events source object.
- struct **threads_queue_t**
Type of a thread queue.

Macros

- #define **OSAL_DBG_ENABLE_ASSERTS** FALSE
Enables OSAL assertions.
- #define **OSAL_DBG_ENABLE_CHECKS** FALSE
Enables OSAL functions parameters checks.

Common constants

- #define **FALSE** 0
- #define **TRUE** 1
- #define **OSAL_SUCCESS** false
- #define **OSAL_FAILED** true

Messages

- #define **MSG_OK** (**msg_t**)0
- #define **MSG_RESET** (**msg_t**)-1
- #define **MSG_TIMEOUT** (**msg_t**)-2

Special time constants

- #define **TIME_IMMEDIATE** ((**systime_t**)0)
- #define **TIME_INFINITE** ((**systime_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_S2ST(sec) ((systime_t)((uint32_t)(sec) * (uint32_t)OSAL_ST_FREQUENCY))`
Seconds to system ticks.
- `#define OSAL_MS2ST(msec)`
Milliseconds to system ticks.
- `#define OSAL_US2ST(usec)`
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(sec) osalThreadSleep(OSAL_S2ST(sec))`
Delays the invoking thread for the specified number of seconds.
- `#define osalThreadSleepMilliseconds(msec) osalThreadSleep(OSAL_MS2ST(msec))`
Delays the invoking thread for the specified number of milliseconds.
- `#define osalThreadSleepMicroseconds(usec) osalThreadSleep(OSAL_US2ST(usec))`
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 rtcnt_t`
Type of realtime counter.
- `typedef void *thread_reference_t`
Type of a thread reference.
- `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 eventflags_t`
Type of an event flags mask.
- `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 (systime_t time)`
Suspends the invoking thread for the specified time.
- `void osalThreadSleep (systime_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, systime_t timeout)`

- Sends the current thread sleeping and sets a reference variable.
- void **osalThreadResumel** (**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, **systime_t** timeout)

Enqueues the caller thread.
- void **osalThreadDequeueNextl** (**threads_queue_t** *tqp, **msg_t** msg)

Dequeues and wakes up one thread from the queue, if any.
- void **osalThreadDequeueAlll** (**threads_queue_t** *tqp, **msg_t** msg)

Dequeues and wakes up all threads from the queue.
- void **osalEventBroadcastFlagsI** (**event_source_t** *esp, **eventflags_t** flags)

Add flags to an event source object.
- void **osalEventBroadcastFlags** (**event_source_t** *esp, **eventflags_t** flags)

Add flags to an event source object.
- void **osalEventSetCallback** (**event_source_t** *esp, **eventcallback_t** cb, void *param)

Event callback setup.
- void **osalMutexLock** (**mutex_t** *mp)

Locks the specified mutex.
- void **osalMutexUnlock** (**mutex_t** *mp)

Unlocks the specified mutex.
- static void **osalSysDisable** (void)

Disables interrupts globally.
- static void **osalSysEnable** (void)

Enables interrupts globally.
- static void **osalSysLock** (void)

Enters a critical zone from thread context.
- static void **osalSysUnlock** (void)

Leaves a critical zone from thread context.
- static void **osalSysLockFromISR** (void)

Enters a critical zone from ISR context.
- static void **osalSysUnlockFromISR** (void)

Leaves a critical zone from ISR context.
- static **syssts_t** **osalSysGetStatusAndLockX** (void)

Returns the execution status and enters a critical zone.
- static void **osalSysRestoreStatusX** (**syssts_t** sts)

Restores the specified execution status and leaves a critical zone.
- static bool **osalOsisTimeWithinX** (**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 flags object.
- static void **osalMutexObjectInit** (**mutex_t** *mp)

Initializes a mutex_t object.

Variables

- const char * **osal_halt_msg**

Pointer to a halt error message.

9.53.1 Detailed Description

OSAL module header.

9.54 pal.c File Reference

I/O Ports Abstraction Layer code.

```
#include "hal.h"
```

Functions

- `ioportmask_t palReadBus (IOBus *bus)`
Read from an I/O bus.
- `void palWriteBus (IOBus *bus, ioportmask_t bits)`
Write to an I/O bus.
- `void palSetBusMode (IOBus *bus, iomode_t mode)`
Programs a bus with the specified mode.

9.54.1 Detailed Description

I/O Ports Abstraction Layer code.

9.55 pal.h File Reference

I/O Ports Abstraction Layer macros, types and structures.

```
#include "pal_lld.h"
```

Data Structures

- `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`

- `#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.

Macro Functions

- `#define palInit(config) pal_lld_init(config)`
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 logical state.
- `#define palWritePad(port, pad, bit)`
Writes a logical state on an output pad.
- `#define palSetPad(port, pad) palSetPort(port, PAL_PORT_BIT(pad))`
*Sets a pad logical state to **PAL_HIGH**.*
- `#define palClearPad(port, pad) palClearPort(port, PAL_PORT_BIT(pad))`
*Clears a pad logical state to **PAL_LOW**.*
- `#define palTogglePad(port, pad) palTogglePort(port, PAL_PORT_BIT(pad))`
Toggles a pad logical state.
- `#define palSetPadMode(port, pad, mode) palSetGroupMode(port, PAL_PORT_BIT(pad), 0U, mode)`
Pad mode setup.

Functions

- `ioportmask_t palReadBus (IOBus *bus)`
Read from an I/O bus.
- `void palWriteBus (IOBus *bus, ioportmask_t bits)`
Write to an I/O bus.
- `void palSetBusMode (IOBus *bus, iomode_t mode)`
Programs a bus with the specified mode.

9.55.1 Detailed Description

I/O Ports Abstraction Layer macros, types and structures.

9.56 pal_lld.c File Reference

PLATFORM PAL subsystem low level driver source.

```
#include "hal.h"
```

Functions

- `void _pal_lld_init (const PALConfig *config)`
STM32 I/O ports configuration.
- `void _pal_lld_setgroupmode (ioportid_t port, ioportmask_t mask, iomode_t mode)`
Pads mode setup.

9.56.1 Detailed Description

PLATFORM PAL subsystem low level driver source.

9.57 pal_lld.h File Reference

PLATFORM PAL subsystem low level driver header.

Data Structures

- `struct PALConfig`
Generic I/O ports static initializer.

Macros

- `#define PAL_IOPORTS_WIDTH 32U`
Width, in bits, of an I/O port.
- `#define PAL_WHOLE_PORT ((ioportmask_t)0xFFFFFFFFU)`
Whole port mask.
- `#define IOPORT1 0`
First I/O port identifier.
- `#define pal_lld_init(config) _pal_lld_init(config)`

- **#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.

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 ioportid_t**
Port Identifier.

Functions

- **void _pal_lld_init (const PALConfig *config)**
STM32 I/O ports configuration.
- **void _pal_lld_setgroupmode (ioportid_t port, ioportmask_t mask, iomode_t mode)**
Pads mode setup.

9.57.1 Detailed Description

PLATFORM PAL subsystem low level driver header.

9.58 pwm.c File Reference

PWM Driver code.

```
#include "hal.h"
```

Functions

- void `pwmInit` (void)

PWM Driver initialization.
- void `pwmObjectInit` (`PWMDriver` *`pwmp`)

Initializes the standard part of a `PWMDriver` structure.
- void `pwmStart` (`PWMDriver` *`pwmp`, const `PWMConfig` *`config`)

Configures and activates the PWM peripheral.
- void `pwmStop` (`PWMDriver` *`pwmp`)

Deactivates the PWM peripheral.
- void `pwmChangePeriod` (`PWMDriver` *`pwmp`, `pwmcnt_t` `period`)

Changes the period the PWM peripheral.
- void `pwmEnableChannel` (`PWMDriver` *`pwmp`, `pwmchannel_t` `channel`, `pwmcnt_t` `width`)

Enables a PWM channel.
- void `pwmDisableChannel` (`PWMDriver` *`pwmp`, `pwmchannel_t` `channel`)

Disables a PWM channel and its notification.
- void `pwmEnablePeriodicNotification` (`PWMDriver` *`pwmp`)

Enables the periodic activation edge notification.
- void `pwmDisablePeriodicNotification` (`PWMDriver` *`pwmp`)

Disables the periodic activation edge notification.
- void `pwmEnableChannelNotification` (`PWMDriver` *`pwmp`, `pwmchannel_t` `channel`)

Enables a channel de-activation edge notification.
- void `pwmDisableChannelNotification` (`PWMDriver` *`pwmp`, `pwmchannel_t` `channel`)

Disables a channel de-activation edge notification.

9.58.1 Detailed Description

PWM Driver code.

9.59 pwm.h File Reference

PWM Driver macros and structures.

```
#include "pwm_lld.h"
```

Macros

PWM output mode macros

- #define `PWM_OUTPUT_MASK` 0x0FU
Standard output modes mask.
- #define `PWM_OUTPUT_DISABLED` 0x00U
Output not driven, callback only.
- #define `PWM_OUTPUT_ACTIVE_HIGH` 0x01U
Positive PWM logic, active is logic level one.
- #define `PWM_OUTPUT_ACTIVE_LOW` 0x02U
Inverse PWM logic, active is logic level zero.

PWM duty cycle conversion

- #define `PWM_FRACTION_TO_WIDTH`(pwmp, denominator, numerator)
Converts from fraction to pulse width.
- #define `PWM_DEGREES_TO_WIDTH`(pwmp, degrees) `PWM_FRACTION_TO_WIDTH`(pwmp, 36000, degrees)
Converts from degrees to pulse width.
- #define `PWM_PERCENTAGE_TO_WIDTH`(pwmp, percentage) `PWM_FRACTION_TO_WIDTH`(pwmp, 10000, percentage)
Converts from percentage to pulse width.

Macro Functions

- #define `pwmChangePeriodI`(pwmp, value)
Changes the period the PWM peripheral.
- #define `pwmEnableChannelI`(pwmp, channel, width)
Enables a PWM channel.
- #define `pwmDisableChannelI`(pwmp, channel)
Disables a PWM channel.
- #define `pwmIsChannelEnabledI`(pwmp, channel) (((pwmp)->enabled & ((pwmchnmsk_t)1U << (pwmchnmsk_t)(channel))) != 0U)
Returns a PWM channel status.
- #define `pwmEnablePeriodicNotificationI`(pwmp) `pwm_lld_enable_periodic_notification`(pwmp)
Enables the periodic activation edge notification.
- #define `pwmDisablePeriodicNotificationI`(pwmp) `pwm_lld_disable_periodic_notification`(pwmp)
Disables the periodic activation edge notification.
- #define `pwmEnableChannelNotificationI`(pwmp, channel) `pwm_lld_enable_channel_notification`(pwmp, channel)
Enables a channel de-activation edge notification.
- #define `pwmDisableChannelNotificationI`(pwmp, channel) `pwm_lld_disable_channel_notification`(pwmp, channel)
Disables a channel de-activation edge notification.

Typedefs

- typedef struct `PWMDriver` `PWMDriver`
Type of a structure representing a PWM driver.
- typedef void(* `pwmcallback_t`) (`PWMDriver` *pwmp)
Type of a PWM notification callback.

Enumerations

- enum `pwmstate_t` { `PWM_UNINIT` = 0, `PWM_STOP` = 1, `PWM_READY` = 2 }
Driver state machine possible states.

Functions

- void `pwmInit` (void)

PWM Driver initialization.
- void `pwmObjectInit` (`PWMDriver` *`pwmp`)

Initializes the standard part of a `PWMDriver` structure.
- void `pwmStart` (`PWMDriver` *`pwmp`, const `PWMConfig` *`config`)

Configures and activates the PWM peripheral.
- void `pwmStop` (`PWMDriver` *`pwmp`)

Deactivates the PWM peripheral.
- void `pwmChangePeriod` (`PWMDriver` *`pwmp`, `pwmcnt_t` `period`)

Changes the period the PWM peripheral.
- void `pwmEnableChannel` (`PWMDriver` *`pwmp`, `pwmchannel_t` `channel`, `pwmcnt_t` `width`)

Enables a PWM channel.
- void `pwmDisableChannel` (`PWMDriver` *`pwmp`, `pwmchannel_t` `channel`)

Disables a PWM channel and its notification.
- void `pwmEnablePeriodicNotification` (`PWMDriver` *`pwmp`)

Enables the periodic activation edge notification.
- void `pwmDisablePeriodicNotification` (`PWMDriver` *`pwmp`)

Disables the periodic activation edge notification.
- void `pwmEnableChannelNotification` (`PWMDriver` *`pwmp`, `pwmchannel_t` `channel`)

Enables a channel de-activation edge notification.
- void `pwmDisableChannelNotification` (`PWMDriver` *`pwmp`, `pwmchannel_t` `channel`)

Disables a channel de-activation edge notification.

9.59.1 Detailed Description

PWM Driver macros and structures.

9.60 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.

9.60.1 Detailed Description

PLATFORM PWM subsystem low level driver source.

9.61 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, pwcnt_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.

9.61.1 Detailed Description

PLATFORM PWM subsystem low level driver header.

9.62 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, rtalarm_t alarm, const RTCAlarm *alarmspec)
Set alarm time.
- void `rtcGetAlarm` (RTCDriver *rtcp, rtalarm_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 *tmp, uint32_t *tv_msec)
Convert `RTCDateTime` to broken-down time structure.

- void `rtcConvertStructTmToDateTIme` (const struct tm *tmp, 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.

9.62.1 Detailed Description

RTC Driver code.

9.63 rtc.h File Reference

RTC Driver macros and structures.

```
#include <time.h>
#include "rtc_lld.h"
```

Data Structures

- struct `RTCDateTime`
Type of a structure representing an RTC date/time stamp.

Macros

- #define `RTC_BASE_YEAR` 1980U
Base year of the calendar.

Date/Time bit masks for FAT format

- #define `RTC_FAT_TIME_SECONDS_MASK` 0x00000001FU
- #define `RTC_FAT_TIME_MINUTES_MASK` 0x0000007E0U
- #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.

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 `rtcConvertStructTmToDateTm` (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.

9.63.1 Detailed Description

RTC Driver macros and structures.

9.64 rtc_lld.c File Reference

PLATFORM RTC subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void `rtc_lld_init` (void)
Enable access to registers.
- 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.

Variables

- `RTCDriver RTCD1`
RTC driver identifier.

9.64.1 Detailed Description

PLATFORM RTC subsystem low level driver source.

9.65 rtc_ll.h File Reference

PLATFORM RTC subsystem low level driver header.

Data Structures

- struct **RTCAlarm**
Type of a structure representing an RTC alarm time stamp.
- struct **RTCDriverVMT**
RTCDriver virtual methods table.
- struct **RTCDriver**
Structure representing an RTC driver.

Macros

- #define **_rtc_driver_methods_file_stream_methods**
FileStream specific methods.

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 uint32_t **rtcalarm_t**
Type of an RTC alarm number.
- 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_ll_init** (void)
Enable access to registers.
- void **rtc_ll_set_time** (**RTCDriver** *rtcp, const **RTCDateTime** *timespec)
Set current time.
- void **rtc_ll_get_time** (**RTCDriver** *rtcp, **RTCDateTime** *timespec)

- *Get current time.*
- void `rtc_lld_set_alarm (RTCDriver *rtcp, rtcalarm_t alarm, const RTCAlarm *alarmspec)`
Set alarm time.
- void `rtc_lld_get_alarm (RTCDriver *rtcp, rtcalarm_t alarm, RTCAlarm *alarmspec)`
Get alarm time.

9.65.1 Detailed Description

PLATFORM RTC subsystem low level driver header.

9.66 sdc.c File Reference

SDC Driver code.

```
#include <string.h>
#include "hal.h"
```

Enumerations

- enum `mmc_switch_t`
MMC switch mode.
- enum `sd_switch_t`
SDC switch mode.
- enum `sd_switch_function_t`
SDC switch function.

Functions

- static bool `mode_detect (SDCDriver *sdcp)`
Detects card mode.
- static bool `mmc_init (SDCDriver *sdcp)`
Init procedure for MMC.
- static bool `sdc_init (SDCDriver *sdcp)`
Init procedure for SDC.
- static uint32_t `mmc_cmd6_construct (mmc_switch_t access, uint32_t idx, uint32_t value, uint32_t cmd_set)`
Constructs CMD6 argument for MMC.
- static uint32_t `sdc_cmd6_construct (sd_switch_t mode, sd_switch_function_t function, uint32_t value)`
Constructs CMD6 argument for SDC.
- static uint16_t `sdc_cmd6_extract_info (sd_switch_function_t function, const uint8_t *buf)`
Extracts information from CMD6 answer.
- static bool `sdc_cmd6_check_status (sd_switch_function_t function, const uint8_t *buf)`
Checks status after switching using CMD6.
- static bool `sdc_detect_bus_clk (SDCDriver *sdcp, sdcbusclk_t *clk)`
Reads supported bus clock and switch SDC to appropriate mode.
- static bool `mmc_detect_bus_clk (SDCDriver *sdcp, sdcbusclk_t *clk)`
Reads supported bus clock and switch MMC to appropriate mode.
- static bool `detect_bus_clk (SDCDriver *sdcp, sdcbusclk_t *clk)`
Reads supported bus clock and switch card to appropriate mode.
- static bool `sdc_set_bus_width (SDCDriver *sdcp)`

- static bool `mmc_set_bus_width (SDCDriver *sdcp)`

Sets bus width for SDC.
- bool `_sdc_wait_for_transfer_state (SDCDriver *sdcp)`

Wait for the card to complete pending operations.
- void `sdcInit (void)`

SDC Driver initialization.
- void `sdcObjectInit (SDCDriver *sdcp)`

Initializes the standard part of a `SDCDriver` structure.
- void `sdcStart (SDCDriver *sdcp, const SDCCConfig *config)`

Configures and activates the SDC peripheral.
- void `sdcStop (SDCDriver *sdcp)`

Deactivates the SDC peripheral.
- bool `sdcConnect (SDCDriver *sdcp)`

Performs the initialization procedure on the inserted card.
- bool `sdcDisconnect (SDCDriver *sdcp)`

Brings the driver in a state safe for card removal.
- bool `sdcRead (SDCDriver *sdcp, uint32_t startblk, uint8_t *buf, uint32_t n)`

Reads one or more blocks.
- bool `sdcWrite (SDCDriver *sdcp, uint32_t startblk, const uint8_t *buf, uint32_t n)`

Writes one or more blocks.
- `sdcflags_t sdcGetAndClearErrors (SDCDriver *sdcp)`

Returns the errors mask associated to the previous operation.
- bool `sdcSync (SDCDriver *sdcp)`

Waits for card idle condition.
- bool `sdcGetInfo (SDCDriver *sdcp, BlockDeviceInfo *bdip)`

Returns the media info.
- bool `sdcErase (SDCDriver *sdcp, uint32_t startblk, uint32_t endblk)`

Erases the supplied blocks.

Variables

- static const struct `SDCDriverVMT sdc_vmt`

Virtual methods table.

9.66.1 Detailed Description

SDC Driver code.

9.67 sdc.h File Reference

SDC Driver macros and structures.

```
#include "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.

Macro Functions

- #define **sdclsCardInserted**(sdcp) (**sdc_lld_is_card_inserted**(sdcp))
Returns the card insertion status.
- #define **sdclsWriteProtected**(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.

9.67.1 Detailed Description

SDC Driver macros and structures.

9.68 sdc_ll.c File Reference

PLATFORM SDC subsystem low level driver source.

```
#include "hal.h"
```

Functions

- **void sdc_ll_init (void)**
Low level SDC driver initialization.
- **void sdc_ll_start (SDCDriver *sdcp)**
Configures and activates the SDC peripheral.
- **void sdc_ll_stop (SDCDriver *sdcp)**
Deactivates the SDC peripheral.
- **void sdc_ll_start_clk (SDCDriver *sdcp)**
Starts the SDIO clock and sets it to init mode (400kHz or less).
- **void sdc_ll_set_data_clk (SDCDriver *sdcp, sdcbusclk_t clk)**
Sets the SDIO clock to data mode (25MHz or less).
- **void sdc_ll_stop_clk (SDCDriver *sdcp)**
Stops the SDIO clock.
- **void sdc_ll_set_bus_mode (SDCDriver *sdcp, sdcbusmode_t mode)**
Switches the bus to 4 bits mode.
- **void sdc_ll_send_cmd_none (SDCDriver *sdcp, uint8_t cmd, uint32_t arg)**
Sends an SDIO command with no response expected.
- **bool sdc_ll_send_cmd_short (SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp)**
Sends an SDIO command with a short response expected.

- bool `sdc_lld_send_cmd_short_crc (SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp)`
Sends an SDIO command with a short response expected and CRC.
- bool `sdc_lld_send_cmd_long_crc (SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp)`
Sends an SDIO command with a long response expected and CRC.
- bool `sdc_lld_read (SDCDriver *sdcp, uint32_t startblk, uint8_t *buf, uint32_t n)`
Reads one or more blocks.
- bool `sdc_lld_write (SDCDriver *sdcp, uint32_t startblk, const uint8_t *buf, uint32_t n)`
Writes one or more blocks.
- bool `sdc_lld_sync (SDCDriver *sdcp)`
Waits for card idle condition.

Variables

- `SDCDriver SDCD1`
SDCD1 driver identifier.

9.68.1 Detailed Description

PLATFORM SDC subsystem low level driver source.

9.69 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.

9.69.1 Detailed Description

PLATFORM SDC subsystem low level driver header.

9.70 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` onnotify)
Initializes a generic full duplex driver object.
- void `sdStart` (SerialDriver *sdp, const `SerialConfig` *config)
Configures and starts the driver.

- void `sdStop (SerialDriver *sdp)`
Stops the driver.
- void `sdIncomingData (SerialDriver *sdp, uint8_t b)`
Handles incoming data.
- `msg_t sdRequestData (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`.

9.70.1 Detailed Description

Serial Driver code.

9.71 serial.h File Reference

Serial Driver macros and structures.

```
#include "serial_lld.h"
```

Data Structures

- struct `SerialDriverVMT`
`SerialDriver` virtual methods table.
- struct `SerialDriver`
Full duplex serial driver class.

Macros

- `#define _serial_driver_methods _base_asynchronous_channel_methods`
`SerialDriver` specific methods.

Serial status flags

- `#define SD_PARITY_ERROR (eventflags_t)32`
Parity.
- `#define SD_FRAMING_ERROR (eventflags_t)64`
Framing.
- `#define SD_OVERRUN_ERROR (eventflags_t)128`
Overflow.
- `#define SD_NOISE_ERROR (eventflags_t)256`
Line noise.
- `#define SD_BREAK_DETECTED (eventflags_t)512`
LIN Break.

Serial configuration options

- `#define SERIAL_DEFAULT_BITRATE 38400`
Default bit rate.
- `#define SERIAL_BUFFERS_SIZE 16`

Serial buffers size.

Macro Functions

- `#define sdPut(sdp, b) oqPut(&(sdp)->oqueue, b)`
Direct write to a `SerialDriver`.
- `#define sdPutTimeout(sdp, b, t) oqPutTimeout(&(sdp)->oqueue, b, t)`
Direct write to a `SerialDriver` with timeout specification.
- `#define sdGet(sdp) iqGet(&(sdp)->iqueue)`
Direct read from a `SerialDriver`.
- `#define sdGetTimeout(sdp, t) iqGetTimeout(&(sdp)->iqueue, t)`
Direct read from a `SerialDriver` with timeout specification.
- `#define sdWrite(sdp, b, n) oqWriteTimeout(&(sdp)->oqueue, b, n, TIME_INFINITE)`
Direct blocking write to a `SerialDriver`.
- `#define sdWriteTimeout(sdp, b, n, t) oqWriteTimeout(&(sdp)->oqueue, b, n, t)`
Direct blocking write to a `SerialDriver` with timeout specification.
- `#define sdAsynchronousWrite(sdp, b, n) oqWriteTimeout(&(sdp)->oqueue, b, n, TIME_IMMEDIATE)`
Direct non-blocking write to a `SerialDriver`.
- `#define sdRead(sdp, b, n) iqReadTimeout(&(sdp)->iqueue, b, n, TIME_INFINITE)`
Direct blocking read from a `SerialDriver`.
- `#define sdReadTimeout(sdp, b, n, t) iqReadTimeout(&(sdp)->iqueue, b, n, t)`
Direct blocking read from a `SerialDriver` with timeout specification.
- `#define sdAsynchronousRead(sdp, b, n) iqReadTimeout(&(sdp)->iqueue, b, n, TIME_IMMEDIATE)`
Direct non-blocking read from a `SerialDriver`.

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 full duplex driver object.
- `void sdStart (SerialDriver *sdp, const SerialConfig *config)`
Configures and starts the driver.
- `void sdStop (SerialDriver *sdp)`
Stops the driver.
- `void sdIncomingData (SerialDriver *sdp, uint8_t b)`
Handles incoming data.
- `msg_t sdRequestData (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`.

9.71.1 Detailed Description

Serial Driver macros and structures.

9.72 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.

9.72.1 Detailed Description

PLATFORM serial subsystem low level driver source.

9.73 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.

9.73.1 Detailed Description

PLATFORM serial subsystem low level driver header.

9.74 serial_usb.c File Reference

Serial over USB Driver code.

```
#include "hal.h"
```

Functions

- static void **inotify** (io_queue_t *qp)
Notification of data removed from the input queue.
- static void **onotify** (io_queue_t *qp)
Notification of data inserted into the output 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 **sduConfigureHookI** (SerialUSBDriver *sdup)
USB device configured handler.
- bool **sduRequestsHook** (USBDriver *usbp)
Default requests hook.
- 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.

9.74.1 Detailed Description

Serial over USB Driver code.

9.75 serial_usb.h File Reference

Serial over USB Driver macros and structures.

Data Structures

- struct `cdc_linecoding_t`
Type of Line Coding structure.
- 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.

CDC specific messages.

- `#define CDC_SEND_ENCAPSULATED_COMMAND 0x00`
- `#define CDC_GET_ENCAPSULATED_RESPONSE 0x01`
- `#define CDC_SET_COMM_FEATURE 0x02`
- `#define CDC_GET_COMM_FEATURE 0x03`
- `#define CDC_CLEAR_COMM_FEATURE 0x04`
- `#define CDC_SET_AUX_LINE_STATE 0x10`
- `#define CDC_SET_HOOK_STATE 0x11`
- `#define CDC_PULSE_SETUP 0x12`
- `#define CDC_SEND_PULSE 0x13`
- `#define CDC_SET_PULSE_TIME 0x14`
- `#define CDC_RING_AUX_JACK 0x15`
- `#define CDC_SET_LINE_CODING 0x20`
- `#define CDC_GET_LINE_CODING 0x21`
- `#define CDC_SET_CONTROL_LINE_STATE 0x22`
- `#define CDC_SEND_BREAK 0x23`
- `#define CDC_SET_RINGER_PARMS 0x30`
- `#define CDC_GET_RINGER_PARMS 0x31`
- `#define CDC_SET_OPERATION_PARMS 0x32`
- `#define CDC_GET_OPERATION_PARMS 0x33`

CDC classes

- `#define CDC_COMMUNICATION_INTERFACE_CLASS 0x02`
- `#define CDC_DATA_INTERFACE_CLASS 0x0A`

CDC subclasses

- `#define CDC_ABSTRACT_CONTROL_MODEL 0x02`

CDC descriptors

- `#define CDC_CS_INTERFACE 0x24`

CDC subdescriptors

- #define **CDC_HEADER** 0x00
- #define **CDC_CALL_MANAGEMENT** 0x01
- #define **CDC_ABSTRACT_CONTROL_MANAGEMENT** 0x02
- #define **CDC_UNION** 0x06

Line Control bit definitions.

- #define **LC_STOP_1** 0
- #define **LC_STOP_1P5** 1
- #define **LC_STOP_2** 2
- #define **LC_PARITY_NONE** 0
- #define **LC_PARITY_ODD** 1
- #define **LC_PARITY_EVEN** 2
- #define **LC_PARITY_MARK** 3
- #define **LC_PARITY_SPACE** 4

SERIAL_USB configuration options

- #define **SERIAL_USB_BUFFERS_SIZE** 256
Serial over USB buffers size.

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 **sduConfigureHookI** (**SerialUSBDriver** *sdup)
USB device configured handler.
- bool **sduRequestsHook** (**USBDriver** *usbp)
Default requests hook.
- 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.

9.75.1 Detailed Description

Serial over USB Driver macros and structures.

9.76 spi.c File Reference

SPI Driver code.

```
#include "hal.h"
```

Functions

- void **spilinit** (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 **spilgnore** (SPIDriver *spip, size_t n)

Ignores data on the SPI bus.
- void **spiExchange** (SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)

Exchanges data on the SPI bus.
- 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.

9.76.1 Detailed Description

SPI Driver code.

9.77 spi.h File Reference

SPI Driver macros and structures.

```
#include "spi_lld.h"
```

Macros

SPI configuration options

- #define SPI_USE_WAIT TRUE
Enables synchronous APIs.
- #define SPI_USE_MUTUAL_EXCLUSION TRUE
Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.

Macro Functions

- #define `spiSelectl`(spip)
Asserts the slave select signal and prepares for transfers.
- #define `spiUnselectl`(spip)
Deasserts the slave select signal.
- #define `spiStartIgnorel`(spip, n)
Ignores data on the SPI bus.
- #define `spiStartExchangeL`(spip, n, txbuf, rxbuf)
Exchanges data on the SPI bus.
- #define `spiStartSendl`(spip, n, txbuf)
Sends data over the SPI bus.
- #define `spiStartReceivel`(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.

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 SPICConfig *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.

9.77.1 Detailed Description

SPI Driver macros and structures.

9.78 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.
- uint16_t **spi_lld_polled_exchange** (SPIDriver *spip, uint16_t frame)
Exchanges one frame using a polled wait.

Variables

- **SPIDriver SPID1**
SPI1 driver identifier.

9.78.1 Detailed Description

PLATFORM SPI subsystem low level driver source.

9.79 spi_lld.h File Reference

PLATFORM SPI subsystem low level driver header.

Data Structures

- struct **SPIConfig**
Driver configuration structure.
- struct **SPIDriver**
Structure representing an SPI driver.

Macros

PLATFORM configuration options

- #define **PLATFORM_SPI_USE_SPI1 FALSE**
SPI1 driver enable switch.

Typedefs

- typedef struct **SPIDriver SPIDriver**
Type of a structure representing an SPI driver.
- typedef void(* **spicallback_t**) (**SPIDriver** *spip)
SPI notification callback type.

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.*
- uint16_t **spi_lld_polled_exchange** (SPIDriver *spip, uint16_t frame)
 - Exchanges one frame using a polled wait.*

9.79.1 Detailed Description

PLATFORM SPI subsystem low level driver header.

9.80 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.*
- void **stStopAlarm** (void)
 - Stops the alarm interrupt.*
- void **stSetAlarm** (systime_t abstime)
 - Sets the alarm time.*
- systime_t **stGetAlarm** (void)
 - Returns the current alarm time.*

9.80.1 Detailed Description

ST Driver code.

9.81 st.h File Reference

ST Driver macros and structures.

```
#include "st_lld.h"
```

Macros

Macro Functions

- #define **stGetCounter()** **st_lld_get_counter()**
 - Returns the time counter value.*
- #define **stIsAlarmActive()** **st_lld_is_alarm_active()**
 - Determines if the alarm is active.*

Functions

- void `stInit` (void)
ST Driver initialization.
- void `stStartAlarm` (`systime_t` abstime)
Starts the alarm.
- void `stStopAlarm` (void)
Stops the alarm interrupt.
- void `stSetAlarm` (`systime_t` abstime)
Sets the alarm time.
- `systime_t` `stGetAlarm` (void)
Returns the current alarm time.

9.81.1 Detailed Description

ST Driver macros and structures.

This header is designed to be include-able without having to include other files from the HAL.

9.82 st_lld.c File Reference

PLATFORM ST subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void `st_lld_init` (void)
Low level ST driver initialization.

9.82.1 Detailed Description

PLATFORM ST subsystem low level driver source.

9.83 st_lld.h File Reference

PLATFORM ST subsystem low level driver header.

Functions

- void `st_lld_init` (void)
Low level ST driver initialization.
- static `systime_t` `st_lld_get_counter` (void)
Returns the time counter value.
- static void `st_lld_start_alarm` (`systime_t` abstime)
Starts the alarm.
- static void `st_lld_stop_alarm` (void)
Stops the alarm interrupt.

- static void [st_lld_set_alarm](#) (systime_t abstime)
Sets the alarm time.
- static systime_t [st_lld_get_alarm](#) (void)
Returns the current alarm time.
- static bool [st_lld_is_alarm_active](#) (void)
Determines if the alarm is active.

9.83.1 Detailed Description

PLATFORM ST subsystem low level driver header.

This header is designed to be include-able without having to include other files from the HAL.

9.84 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 [uartStartReceive1](#) (UARTDriver *uartp, size_t n, void *rxbuf)
Starts a receive operation on the UART peripheral.
- size_t [uartStopReceive](#) (UARTDriver *uartp)
Stops any ongoing receive operation.
- size_t [uartStopReceive1](#) (UARTDriver *uartp)
Stops any ongoing receive operation.

9.84.1 Detailed Description

UART Driver code.

9.85 uart.h File Reference

UART Driver macros and structures.

```
#include "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.

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 **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 *rdbuf)
Starts a receive operation on the UART peripheral.

- void `uartStartReceive1 (UARTDriver *uartp, size_t n, void *rdbuf)`
Starts a receive operation on the UART peripheral.
- size_t `uartStopReceive (UARTDriver *uartp)`
Stops any ongoing receive operation.
- size_t `uartStopReceive1 (UARTDriver *uartp)`
Stops any ongoing receive operation.

9.85.1 Detailed Description

UART Driver macros and structures.

9.86 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 *rdbuf)`
Starts a receive operation on the UART peripheral.
- size_t `uart_lld_stop_receive (UARTDriver *uartp)`
Stops any ongoing receive operation.

Variables

- `UARTDriver UARTD1`
UART1 driver identifier.

9.86.1 Detailed Description

PLATFORM UART subsystem low level driver source.

9.87 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_ll_init** (void)
Low level UART driver initialization.
- void **uart_ll_start** (**UARTDriver** *uartp)
Configures and activates the UART peripheral.
- void **uart_ll_stop** (**UARTDriver** *uartp)
Deactivates the UART peripheral.
- void **uart_ll_start_send** (**UARTDriver** *uartp, size_t n, const void *txbuf)
Starts a transmission on the UART peripheral.
- size_t **uart_ll_stop_send** (**UARTDriver** *uartp)
Stops any ongoing transmission.
- void **uart_ll_start_receive** (**UARTDriver** *uartp, size_t n, void *rdbuf)
Starts a receive operation on the UART peripheral.
- size_t **uart_ll_stop_receive** (**UARTDriver** *uartp)
Stops any ongoing receive operation.

9.87.1 Detailed Description

PLATFORM UART subsystem low level driver header.

9.88 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 **usbInitEndpointl** (USBDriver *usbp, usbep_t ep, const USBEndpointConfig *epcp)

Enables an endpoint.
- void **usbDisableEndpointsl** (USBDriver *usbp)

Disables all the active endpoints.
- void **usbPrepareReceive** (USBDriver *usbp, usbep_t ep, uint8_t *buf, size_t n)

Prepares for a receive transaction on an OUT endpoint.
- void **usbPrepareTransmit** (USBDriver *usbp, usbep_t ep, const uint8_t *buf, size_t n)

Prepares for a transmit transaction on an IN endpoint.
- void **usbPrepareQueuedReceive** (USBDriver *usbp, usbep_t ep, input_queue_t *iqp, size_t n)

Prepares for a receive transaction on an OUT endpoint.
- void **usbPrepareQueuedTransmit** (USBDriver *usbp, usbep_t ep, output_queue_t *oqp, size_t n)

Prepares for a transmit transaction on an IN endpoint.
- bool **usbStartReceivel** (USBDriver *usbp, usbep_t ep)

Starts a receive transaction on an OUT endpoint.
- bool **usbStartTransmitl** (USBDriver *usbp, usbep_t ep)

Starts a transmit transaction on an IN endpoint.
- bool **usbStallReceivel** (USBDriver *usbp, usbep_t ep)

Stalls an OUT endpoint.
- bool **usbStallTransmitl** (USBDriver *usbp, usbep_t ep)

Stalls an IN endpoint.
- 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.

9.88.1 Detailed Description

USB Driver code.

9.89 usb.h File Reference

USB Driver macros and structures.

```
#include "usb_lld.h"
```

Data Structures

- struct **USBDescriptor**
Type of an USB descriptor.

Macros

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
- #define `USB_EP_MODE_LINEAR_BUFFER` 0x0000U
- #define `USB_EP_MODE_QUEUE_BUFFER` 0x0010U

Macro Functions

- #define `usbGetDriverState`(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 `usbGetFrameNumber`(usbp) `usb_lld_get_frame_number`(usbp)
Returns the current frame number.
- #define `usbGetTransmitStatus`(usbp, ep) (((usbp)->transmitting & (uint16_t)((unsigned)1U << (unsigned)(ep))) != 0U)
Returns the status of an IN endpoint.
- #define `usbGetReceiveStatus`(usbp, ep) (((usbp)->receiving & (uint16_t)((unsigned)1U << (unsigned)(ep))) != 0U)
Returns the status of an OUT endpoint.
- #define `usbGetReceiveTransactionSize`(usbp, ep) `usb_lld_get_transaction_size`(usbp, ep)
Returns the exact size of a receive transaction.
- #define `usbSetupTransfer`(usbp, buf, n, endcb)
Request transfer setup.
- #define `usbReadSetup`(usbp, ep, buf) `usb_lld_read_setup`(usbp, ep, buf)
Reads a setup packet from the dedicated packet buffer.

Low level driver helper macros

- #define `_usb_isr_invoke_event_cb`(usbp, evt)
Common ISR code, usb event callback.
- #define `_usb_isr_invoke_sof_cb`(usbp)
Common ISR code, SOF callback.
- #define `_usb_isr_invoke_setup_cb`(usbp, ep)
Common ISR code, setup packet callback.
- #define `_usb_isr_invoke_in_cb`(usbp, ep)
Common ISR code, IN endpoint callback.
- #define `_usb_isr_invoke_out_cb`(usbp, ep)
Common ISR code, OUT endpoint event.

Typedefs

- typedef struct `USBDriver` `USBDriver`
Type of a structure representing an USB driver.
- typedef uint8_t `usbep_t`
Type of an endpoint identifier.
- typedef void(* `usbcallback_t`) (`USBDriver` *usbp)
Type of an USB generic notification callback.
- typedef void(* `usbepcallback_t`) (`USBDriver` *usbp, `usbep_t` ep)
Type of an USB endpoint callback.
- typedef void(* `usbeventcb_t`) (`USBDriver` *usbp, `usbevent_t` event)
Type of an USB event notification callback.
- typedef bool(* `usbreqhandler_t`) (`USBDriver` *usbp)
Type of a requests handler callback.
- typedef const `USBDescriptor` *(* `usbgetdescriptor_t`) (`USBDriver` *usbp, uint8_t dtype, uint8_t dindex, uint16_t lang)
Type of an USB descriptor-retrieving callback.

Enumerations

- enum `usbstate_t` {

 `USB_UNINIT` = 0, `USB_STOP` = 1, `USB_READY` = 2, `USB_SELECTED` = 3,

 `USB_ACTIVE` = 4, `USB_SUSPENDED` = 5 }

 Type of a driver state machine possible states.
- enum `usbepstatus_t` { `EP_STATUS_DISABLED` = 0, `EP_STATUS_STALLED` = 1, `EP_STATUS_ACTIVE` = 2 }

 Type of an endpoint status.
- enum `usbep0state_t` {

 `USB_EP0_WAITING_SETUP`, `USB_EP0_TX`, `USB_EP0_WAITING_TX0`, `USB_EP0_WAITING_STS`,

 `USB_EP0_RX`, `USB_EP0_SENDING_STS`, `USB_EP0_ERROR` }

 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_SUSPEND` = 3,

 `USB_EVENT_WAKEUP` = 4, `USB_EVENT_STALLED` = 5 }

 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 `usbInitEndpoint` (`USBDriver` *`usbp`, `usbep_t` `ep`, const `USBEndpointConfig` *`epcp`)

 Enables an endpoint.
- void `usbDisableEndpoints` (`USBDriver` *`usbp`)

 Disables all the active endpoints.
- void `usbPrepareReceive` (`USBDriver` *`usbp`, `usbep_t` `ep`, `uint8_t` *`buf`, `size_t` `n`)

 Prepares for a receive transaction on an OUT endpoint.
- void `usbPrepareTransmit` (`USBDriver` *`usbp`, `usbep_t` `ep`, const `uint8_t` *`buf`, `size_t` `n`)

 Prepares for a transmit transaction on an IN endpoint.
- void `usbPrepareQueuedReceive` (`USBDriver` *`usbp`, `usbep_t` `ep`, `input_queue_t` *`iqp`, `size_t` `n`)

 Prepares for a receive transaction on an OUT endpoint.
- void `usbPrepareQueuedTransmit` (`USBDriver` *`usbp`, `usbep_t` `ep`, `output_queue_t` *`oqp`, `size_t` `n`)

 Prepares for a transmit transaction on an IN endpoint.
- bool `usbStartReceivel` (`USBDriver` *`usbp`, `usbep_t` `ep`)

 Starts a receive transaction on an OUT endpoint.
- bool `usbStartTransmitl` (`USBDriver` *`usbp`, `usbep_t` `ep`)

 Starts a transmit transaction on an IN endpoint.
- bool `usbStallReceivel` (`USBDriver` *`usbp`, `usbep_t` `ep`)

 Stalls an OUT endpoint.
- bool `usbStallTransmitl` (`USBDriver` *`usbp`, `usbep_t` `ep`)

 Stalls an IN endpoint.
- void `_usb_reset` (`USBDriver` *`usbp`)

 USB reset routine.
- void `_usb_suspend` (`USBDriver` *`usbp`)

- void `_usb_wakeup (USBDriver *usbp)`
USB suspend routine.
- void `_usb_ep0setup (USBDriver *usbp, usbep_t ep)`
USB wake-up routine.
- void `_usb_ep0in (USBDriver *usbp, usbep_t ep)`
Default EP0 SETUP callback.
- void `_usb_ep0out (USBDriver *usbp, usbep_t ep)`
Default EP0 IN callback.
- void `_usb_ep0out (USBDriver *usbp, usbep_t ep)`
Default EP0 OUT callback.

9.89.1 Detailed Description

USB Driver macros and structures.

9.90 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_ll_stall_out** (**USBDriver** *usbp, **usbep_t** ep)

Brings an OUT endpoint in the stalled state.
- void **usb_ll_stall_in** (**USBDriver** *usbp, **usbep_t** ep)

Brings an IN endpoint in the stalled state.
- void **usb_ll_clear_out** (**USBDriver** *usbp, **usbep_t** ep)

Brings an OUT endpoint in the active state.
- void **usb_ll_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.

9.90.1 Detailed Description

PLATFORM USB subsystem low level driver source.

9.90.2 Variable Documentation

9.90.2.1 **USBInEndpointState** in

IN EP0 state.

9.90.2.2 **USBOutEndpointState** out

OUT EP0 state.

9.91 usb_ll.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 [USBCConfig](#)
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.

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_ll_read_setup (USBDriver *usbp, usbep_t ep, uint8_t *buf)`
Reads a setup packet from the dedicated packet buffer.
- void `usb_ll_prepare_receive (USBDriver *usbp, usbep_t ep)`
Prepares for a receive operation.
- void `usb_ll_prepare_transmit (USBDriver *usbp, usbep_t ep)`
Prepares for a transmit operation.
- void `usb_ll_start_out (USBDriver *usbp, usbep_t ep)`
Starts a receive operation on an OUT endpoint.
- void `usb_ll_start_in (USBDriver *usbp, usbep_t ep)`
Starts a transmit operation on an IN endpoint.
- void `usb_ll_stall_out (USBDriver *usbp, usbep_t ep)`
Brings an OUT endpoint in the stalled state.
- void `usb_ll_stall_in (USBDriver *usbp, usbep_t ep)`
Brings an IN endpoint in the stalled state.
- void `usb_ll_clear_out (USBDriver *usbp, usbep_t ep)`
Brings an OUT endpoint in the active state.
- void `usb_ll_clear_in (USBDriver *usbp, usbep_t ep)`
Brings an IN endpoint in the active state.

9.91.1 Detailed Description

PLATFORM USB subsystem low level driver header.

Index

_CHIBIOS_HAL_
 HAL Driver, 94
_INPUTQUEUE_DATA
 I/O Queues, 120
_IOBUS_DATA
 PAL Driver, 267
_OUTPUTQUEUE_DATA
 I/O Queues, 123
_adc_isr_error_code
 ADC Driver, 23
_adc_isr_full_code
 ADC Driver, 23
_adc_isr_half_code
 ADC Driver, 22
_adc_reset_i
 ADC Driver, 21
_adc_reset_s
 ADC Driver, 21
_adc_timeout_isr
 ADC Driver, 22
_adc_wakeup_isr
 ADC Driver, 22
_base_asynchronous_channel_data
 Abstract I/O Channel, 100
_base_asynchronous_channel_methods
 Abstract I/O Channel, 100
_base_block_device_data
 Abstract I/O Block Device, 109
_base_block_device_methods
 Abstract I/O Block Device, 109
_base_channel_data
 Abstract I/O Channel, 97
_base_channel_methods
 Abstract I/O Channel, 97
_base_sequential_stream_data
 Abstract Streams, 135
_base_sequential_stream_methods
 Abstract Streams, 134
_dac_isr_error_code
 DAC Driver, 54
_dac_isr_full_code
 DAC Driver, 53
_dac_isr_half_code
 DAC Driver, 53
_dac_reset_i
 DAC Driver, 52
_dac_reset_s
 DAC Driver, 52
_dac_timeout_isr
 DAC Driver, 53
_dac_wait_s
 DAC Driver, 51
_dac_wakeup_isr
 DAC Driver, 52
_file_stream_data
 Abstract Files, 103
_file_stream_methods
 Abstract Files, 103
_i2c_wakeup_error_isr
 I2C Driver, 140
_i2c_wakeup_isr
 I2C Driver, 140
_i2s_isr_full_code
 I2S Driver, 153
_i2s_isr_half_code
 I2S Driver, 153
_icu_isr_invoke_overflow_cb
 ICU Driver, 165
_icu_isr_invoke_period_cb
 ICU Driver, 165
_icu_isr_invoke_width_cb
 ICU Driver, 164
_mmc_driver_methods
 MMC over SPI Driver, 231
_mmcsd_block_device_data
 MMC/SD Block Device, 256
_mmcsd_block_device_methods
 MMC/SD Block Device, 255
_mmcsd_get_capacity
 MMC/SD Block Device, 257
_mmcsd_get_capacity_ext
 MMC/SD Block Device, 258
_mmcsd_get_slice
 MMC/SD Block Device, 257
_mmcsd_unpack_csd_mmc
 MMC/SD Block Device, 259
_mmcsd_unpack_csd_v10
 MMC/SD Block Device, 259
_mmcsd_unpack_csd_v20
 MMC/SD Block Device, 260
_mmcsd_unpack_mmc_cid
 MMC/SD Block Device, 258
_mmcsd_unpack_sdc_cid
 MMC/SD Block Device, 258
_pal_lld_init
 PAL Driver, 282
_pal_lld_setgroupmode
 PAL Driver, 282

_rtc_driver_methods
 RTC Driver, 309
 _sdc_driver_methods
 SDC Driver, 322
 _sdc_wait_for_transfer_state
 SDC Driver, 331
 _serial_driver_data
 Serial Driver, 352
 _serial_driver_methods
 Serial Driver, 349
 _serial_usb_driver_data
 Serial over USB Driver, 364
 _serial_usb_driver_methods
 Serial over USB Driver, 364
 _spi_isr_code
 SPI Driver, 378
 _spi_wakeup_isr
 SPI Driver, 378
 _usb_ep0in
 USB Driver, 450
 _usb_ep0out
 USB Driver, 451
 _usb_ep0setup
 USB Driver, 449
 _usb_isr_invoke_event_cb
 USB Driver, 431
 _usb_isr_invoke_in_cb
 USB Driver, 433
 _usb_isr_invoke_out_cb
 USB Driver, 433
 _usb_isr_invoke_setup_cb
 USB Driver, 433
 _usb_isr_invoke_sof_cb
 USB Driver, 431
 _usb_reset
 USB Driver, 448
 _usb_suspend
 USB Driver, 448
 _usb_wakeup
 USB Driver, 449

ADC Driver, 17
 _adc_isr_error_code, 23
 _adc_isr_full_code, 23
 _adc_isr_half_code, 22
 _adc_reset_i, 21
 _adc_reset_s, 21
 _adc_timeout_isr, 22
 _adc_wakeup_isr, 22
 ADC_ACTIVE, 25
 ADC_COMPLETE, 25
 ADC_ERR_AWD, 25
 ADC_ERR_DMAFAILURE, 25
 ADC_ERR_OVERFLOW, 25
 ADC_ERROR, 25
 ADC_READY, 25
 ADC_STOP, 25
 ADC_UNINIT, 25
 ADC_USE_MUTUAL_EXCLUSION, 21

ADC_USE_WAIT, 21
 ADCD1, 34
 ADCDriver, 24
 adc_channels_num_t, 24
 adc_lld_init, 32
 adc_lld_start, 33
 adc_lld_start_conversion, 33
 adc_lld_stop, 33
 adc_lld_stop_conversion, 33
 adcAcquireBus, 31
 adcConvert, 30
 adcInit, 25
 adcObjectInit, 26
 adcReleaseBus, 32
 adcStart, 26
 adcStartConversion, 28
 adcStartConversionI, 29
 adcStop, 26
 adcStopConversion, 29
 adcStopConversionI, 30
 adccallback_t, 24
 adcerror_t, 25
 adcerrorcallback_t, 24
 adcsample_t, 24
 adcstate_t, 25
 PLATFORM_ADC_USE_ADC1, 24

ADC_ACTIVE
 ADC Driver, 25
 ADC_COMPLETE
 ADC Driver, 25
 ADC_ERR_AWD
 ADC Driver, 25
 ADC_ERR_DMAFAILURE
 ADC Driver, 25
 ADC_ERR_OVERFLOW
 ADC Driver, 25
 ADC_ERROR
 ADC Driver, 25
 ADC_READY
 ADC Driver, 25
 ADC_STOP
 ADC Driver, 25
 ADC_UNINIT
 ADC Driver, 25
 ADC_USE_MUTUAL_EXCLUSION
 ADC Driver, 21
 Configuration, 199
 ADC_USE_WAIT
 ADC Driver, 21
 Configuration, 199
 ADCConfig, 461
 ADCConversionGroup, 462
 circular, 463
 end_cb, 463
 error_cb, 463
 num_channels, 463

ADCD1
 ADC Driver, 34

ADCDriver, 463
ADC Driver, 24
config, 465
depth, 465
grpp, 465
mutex, 465
samples, 465
state, 465
thread, 465
Abstract Files, 102
 _file_stream_data, 103
 _file_stream_methods, 103
FILE_EOF, 103
FILE_ERROR, 103
FILE_OK, 103
fileStreamClose, 105
fileStreamGet, 105
fileStreamGetError, 106
fileStreamGetPosition, 106
fileStreamGetSize, 106
fileStreamPut, 104
fileStreamRead, 104
fileStreamSeek, 107
fileStreamWrite, 103
fileoffset_t, 107
Abstract I/O Block Device, 108
 _base_block_device_data, 109
 _base_block_device_methods, 109
BLK_ACTIVE, 114
BLK_CONNECTING, 114
BLK_DISCONNECTING, 114
BLK_READING, 114
BLK_READY, 114
BLK_STOP, 114
BLK_SYNCING, 114
BLK_UNINIT, 114
BLK_WRITING, 114
blkConnect, 111
blkDisconnect, 112
blkGetDriverState, 110
blkGetInfo, 113
blkIsInserted, 111
blkIsTransferring, 110
blkIsWriteProtected, 111
blkRead, 112
blkSync, 113
blkWrite, 113
blkstate_t, 114
Abstract I/O Channel, 96
 _base_asynchronous_channel_data, 100
 _base_asynchronous_channel_methods, 100
 _base_channel_data, 97
 _base_channel_methods, 97
CHN_CONNECTED, 100
CHN_DISCONNECTED, 100
CHN_INPUT_AVAILABLE, 100
CHN_NO_ERROR, 100
CHN_OUTPUT_EMPTY, 100
 CHN_TRANSMISSION_END, 100
 chnAddFlagsI, 101
 chnGetEventSource, 101
 chnGetTimeout, 98
 chnPutTimeout, 97
 chnRead, 99
 chnReadTimeout, 99
 chnWrite, 98
 chnWriteTimeout, 99
Abstract Streams, 134
 _base_sequential_stream_data, 135
 _base_sequential_stream_methods, 134
 streamGet, 136
 streamPut, 135
 streamRead, 135
 streamWrite, 135
adc.c, 567
adc.h, 568
adc_channels_num_t
 ADC Driver, 24
adc_lld.c, 569
adc_lld.h, 569
adc_lld_init
 ADC Driver, 32
adc_lld_start
 ADC Driver, 33
adc_lld_start_conversion
 ADC Driver, 33
adc_lld_stop
 ADC Driver, 33
adc_lld_stop_conversion
 ADC Driver, 33
adcAcquireBus
 ADC Driver, 31
adcConvert
 ADC Driver, 30
adclInit
 ADC Driver, 25
adcObjectInit
 ADC Driver, 26
adcReleaseBus
 ADC Driver, 32
adcStart
 ADC Driver, 26
adcStartConversion
 ADC Driver, 28
adcStartConversionl
 ADC Driver, 29
adcStop
 ADC Driver, 26
adcStopConversion
 ADC Driver, 29
adcStopConversionl
 ADC Driver, 30
adccallback_t
 ADC Driver, 24
adcerror_t
 ADC Driver, 25

adcerrorcallback_t
 ADC Driver, 24

adcsample_t
 ADC Driver, 24

adcstate_t
 ADC Driver, 25

address
 USBDriver, 558

BLK_ACTIVE
 Abstract I/O Block Device, 114

BLK_CONNECTING
 Abstract I/O Block Device, 114

BLK_DISCONNECTING
 Abstract I/O Block Device, 114

BLK_READING
 Abstract I/O Block Device, 114

BLK_READY
 Abstract I/O Block Device, 114

BLK_STOP
 Abstract I/O Block Device, 114

BLK_SYNCING
 Abstract I/O Block Device, 114

BLK_UNINIT
 Abstract I/O Block Device, 114

BLK_WRITING
 Abstract I/O Block Device, 114

BaseAsynchronousChannel, 465
 vmt, 467

BaseAsynchronousChannelVMT, 467

BaseBlockDevice, 469
 vmt, 471

BaseBlockDeviceVMT, 471

BaseChannel, 472
 vmt, 474

BaseChannelVMT, 474

BaseSequentialStream, 476
 vmt, 477

BaseSequentialStreamVMT, 477

blk_num
 BlockDeviceInfo, 479

blk_size
 BlockDeviceInfo, 479

blkConnect
 Abstract I/O Block Device, 111

blkDisconnect
 Abstract I/O Block Device, 112

blkGetDriverState
 Abstract I/O Block Device, 110

blkGetInfo
 Abstract I/O Block Device, 113

blkIsInserted
 Abstract I/O Block Device, 111

blkIsTransferring
 Abstract I/O Block Device, 110

blkIsWriteProtected
 Abstract I/O Block Device, 111

blkRead
 Abstract I/O Block Device, 112

blkSync
 Abstract I/O Block Device, 113

blkWrite
 Abstract I/O Block Device, 113

blkstate_t
 Abstract I/O Block Device, 114

BlockDeviceInfo, 478
 blk_num, 479
 blk_size, 479

bulk_in
 SerialUSBConfig, 540

bulk_out
 SerialUSBConfig, 540

bus_width
 SDCConfig, 532

CAN Driver, 35
 CAN_ANY_MAILBOX, 38
 CAN_BUS_OFF_ERROR, 37
 CAN_FRAMING_ERROR, 38
 CAN_LIMIT_ERROR, 37
 CAN_LIMIT_WARNING, 37
 CAN_MAILBOX_TO_MASK, 38
 CAN_OVERFLOW_ERROR, 38
 CAN_READY, 39
 CAN_RX_MAILBOXES, 38
 CAN_SLEEP, 39
 CAN_STARTING, 39
 CAN_STOP, 39
 CAN_TX_MAILBOXES, 38
 CAN_UNINIT, 39
 CAN_USE_SLEEP_MODE, 38
 CAND1, 48
 can_lld_init, 45
 can_lld_is_rx_nonempty, 46
 can_lld_is_tx_empty, 46
 can_lld_receive, 47
 can_lld_sleep, 47
 can_lld_start, 45
 can_lld_stop, 46
 can_lld_transmit, 46
 can_lld_wakeup, 47
 canInit, 39
 canObjectInit, 39
 canReceive, 42
 canSleep, 43
 canStart, 40
 canStop, 40
 canTransmit, 41
 canWakeUp, 44
 canmbx_t, 38
 canstate_t, 39
 PLATFORM_CAN_USE_CAN1, 38

CAN_ANY_MAILBOX
 CAN Driver, 38

CAN_BUS_OFF_ERROR
 CAN Driver, 37

CAN_FRAMING_ERROR
 CAN Driver, 38

CAN_LIMIT_ERROR
 CAN Driver, 37
CAN_LIMIT_WARNING
 CAN Driver, 37
CAN_MAILBOX_TO_MASK
 CAN Driver, 38
CAN_OVERFLOW_ERROR
 CAN Driver, 38
CAN_READY
 CAN Driver, 39
CAN_RX_MAILBOXES
 CAN Driver, 38
CAN_SLEEP
 CAN Driver, 39
CAN_STARTING
 CAN Driver, 39
CAN_STOP
 CAN Driver, 39
CAN_TX_MAILBOXES
 CAN Driver, 38
CAN_UNINIT
 CAN Driver, 39
CAN_USE_SLEEP_MODE
 CAN Driver, 38
 Configuration, 199
CANConfig, 479
CAND1
 CAN Driver, 48
CANDriver, 480
 config, 481
 error_event, 482
 rxfull_event, 481
 rxqueue, 481
 sleep_event, 482
 state, 481
 txempty_event, 481
 txqueue, 481
 wakeup_event, 482
CANRxFrame, 482
 DLC, 483
 data16, 484
 data32, 484
 data8, 484
 EID, 484
 FMI, 483
 IDE, 483
 RTR, 483
 SID, 484
 TIME, 483
CANTxFrame, 484
 DLC, 485
 data16, 486
 data32, 486
 data8, 485
 EID, 485
 IDE, 485
 RTR, 485
 SID, 485
CH_HAL_MAJOR
 HAL Driver, 94
CH_HAL_MINOR
 HAL Driver, 94
CH_HAL_PATCH
 HAL Driver, 94
CH_HAL_STABLE
 HAL Driver, 94
CHN_CONNECTED
 Abstract I/O Channel, 100
CHN_DISCONNECTED
 Abstract I/O Channel, 100
CHN_INPUT_AVAILABLE
 Abstract I/O Channel, 100
CHN_NO_ERROR
 Abstract I/O Channel, 100
CHN_OUTPUT_EMPTY
 Abstract I/O Channel, 100
CHN_TRANSMISSION_END
 Abstract I/O Channel, 100
callback
 GPTConfig, 498
 PWMChannelConfig, 523
 PWMConfig, 525
can.c, 571
can.h, 571
can_lld.c, 572
can_lld.h, 573
can_lld_init
 CAN Driver, 45
can_lld_is_rx_nonempty
 CAN Driver, 46
can_lld_is_tx_empty
 CAN Driver, 46
can_lld_receive
 CAN Driver, 47
can_lld_sleep
 CAN Driver, 47
can_lld_start
 CAN Driver, 45
can_lld_stop
 CAN Driver, 46
can_lld_transmit
 CAN Driver, 46
can_lld_wakeup
 CAN Driver, 47
canInit
 CAN Driver, 39
canObjectInit
 CAN Driver, 39
canReceive
 CAN Driver, 42
canSleep
 CAN Driver, 43
canStart
 CAN Driver, 40
canStop
 CAN Driver, 40

canTransmit
 CAN Driver, 41

canWakeup
 CAN Driver, 44

canmbx_t
 CAN Driver, 38

canstate_t
 CAN Driver, 39

cardmode
 SDCDriver, 534

cb
 EXTChannelConfig, 492
 event_source, 491

cdc_linecoding_t, 486

channels
 EXTConfig, 493
 PWMConfig, 525
 PWMDriver, 527

chnAddFlagsI
 Abstract I/O Channel, 101

chnGetEventSource
 Abstract I/O Channel, 101

chnGetTimeout
 Abstract I/O Channel, 98

chnPutTimeout
 Abstract I/O Channel, 97

chnRead
 Abstract I/O Channel, 99

chnReadTimeout
 Abstract I/O Channel, 99

chnWrite
 Abstract I/O Channel, 98

chnWriteTimeout
 Abstract I/O Channel, 99

circular
 ADCConversionGroup, 463

Complex Drivers, 203

config
 ADCDriver, 465
 CANDriver, 481
 DACDriver, 490
 EXTDriver, 494
 GPTDriver, 499
 I2CDriver, 501
 I2SDriver, 504
 ICUDriver, 506
 MACDriver, 511
 MMCDriver, 516
 PWMDriver, 527
 SDCDriver, 533
 SPIDriver, 545
 UARTDriver, 549
 USBDriver, 557

Configuration, 196
 ADC_USE_MUTUAL_EXCLUSION, 199
 ADC_USE_WAIT, 199
 CAN_USE_SLEEP_MODE, 199
 HAL_USE_ADC, 198
 HAL_USE_CAN, 198
 HAL_USE_DAC, 198
 HAL_USE_EXT, 198
 HAL_USE_GPT, 198
 HAL_USE_I2C, 198
 HAL_USE_I2S, 198
 HAL_USE_ICU, 198
 HAL_USE_MAC, 198
 HAL_USE_MMC_SPI, 198
 HAL_USE_PAL, 198
 HAL_USE_PWM, 199
 HAL_USE_RTC, 199
 HAL_USE_SDC, 199
 HAL_USE_SERIAL, 199
 HAL_USE_SERIAL_USB, 199
 HAL_USE_SPI, 199
 HAL_USE_UART, 199
 HAL_USE_USB, 199
 I2C_USE_MUTUAL_EXCLUSION, 200
 MAC_USE_EVENTS, 200
 MAC_USE_ZERO_COPY, 200
 MMC_NICE_WAITING, 200
 SDC_INIT_RETRY, 200
 SDC_MMCSUPPORT, 200
 SDC_NICE_WAITING, 200
 SERIAL_BUFFERS_SIZE, 200
 SERIAL_DEFAULT_BITRATE, 200
 SERIAL_USB_BUFFERS_SIZE, 201
 SPI_USE_MUTUAL_EXCLUSION, 201
 SPI_USE_WAIT, 201

configuration
 USBDriver, 558

crc7
 MMC over SPI Driver, 232

crc7_lookup_table
 MMC over SPI Driver, 249

DAC Driver, 49
 _dac_isr_error_code, 54
 _dac_isr_full_code, 53
 _dac_isr_half_code, 53
 _dac_reset_i, 52
 _dac_reset_s, 52
 _dac_timeout_isr, 53
 _dac_wait_s, 51
 _dac_wakeup_isr, 52
 DAC_ACTIVE, 56
 DAC_COMPLETE, 56
 DAC_ERR_DMAFAILURE, 56
 DAC_ERR_UNDERFLOW, 56
 DAC_ERROR, 56
 DAC_MAX_CHANNELS, 55
 DAC_READY, 56
 DAC_STOP, 56
 DAC_UNINIT, 56
 DAC_USE_MUTUAL_EXCLUSION, 51
 DAC_USE_WAIT, 51
 DACP1, 65
 DACDriver, 55

dac_lld_init, 63
dac_lld_put_channel, 64
dac_lld_start, 63
dac_lld_start_conversion, 64
dac_lld_stop, 64
dac_lld_stop_conversion, 64
dacAcquireBus, 62
dacConvert, 61
dacInit, 56
dacObjectInit, 56
dacPutChannelX, 58
dacReleaseBus, 62
dacStart, 57
dacStartConversion, 58
dacStartConversionl, 59
dacStop, 57
dacStopConversion, 60
dacStopConversionl, 60
daccallback_t, 55
dacchannel_t, 55
dacerror_t, 56
dacerrorcallback_t, 55
dacsample_t, 55
dacstate_t, 56
PLATFORM_DAC_USE_DAC1, 55
DAC_ACTIVE
 DAC Driver, 56
DAC_COMPLETE
 DAC Driver, 56
DAC_ERR_DMAFAILURE
 DAC Driver, 56
DAC_ERR_UNDERFLOW
 DAC Driver, 56
DAC_ERROR
 DAC Driver, 56
DAC_MAX_CHANNELS
 DAC Driver, 55
DAC_READY
 DAC Driver, 56
DAC_STOP
 DAC Driver, 56
DAC_UNINIT
 DAC Driver, 56
DAC_USE_MUTUAL_EXCLUSION
 DAC Driver, 51
DAC_USE_WAIT
 DAC Driver, 51
DACConfig, 486
DACConversionGroup, 487
 end_cb, 488
 error_cb, 488
 num_channels, 488
DACD1
 DAC Driver, 65
DACDriver, 488
 config, 490
 DAC Driver, 55
 depth, 490
 grpp, 490
 mutex, 490
 samples, 490
 state, 490
 thread, 490
DLC
 CANRxFrame, 483
 CANTxFrame, 485
 dac.c, 574
 dac.h, 575
 dac_lld.c, 577
 dac_lld.h, 577
 dac_lld_init
 DAC Driver, 63
 dac_lld_put_channel
 DAC Driver, 64
 dac_lld_start
 DAC Driver, 63
 dac_lld_start_conversion
 DAC Driver, 64
 dac_lld_stop
 DAC Driver, 64
 dac_lld_stop_conversion
 DAC Driver, 64
 dacAcquireBus
 DAC Driver, 62
 dacConvert
 DAC Driver, 61
 dacInit
 DAC Driver, 56
 dacObjectInit
 DAC Driver, 56
 dacPutChannelX
 DAC Driver, 58
 dacReleaseBus
 DAC Driver, 62
 dacStart
 DAC Driver, 57
 dacStartConversion
 DAC Driver, 58
 dacStartConversionl
 DAC Driver, 59
 dacStop
 DAC Driver, 57
 daccallback_t
 DAC Driver, 55
 dacchannel_t
 DAC Driver, 55
 dacerror_t
 DAC Driver, 56
 dacerrorcallback_t
 DAC Driver, 55
 dacsample_t
 DAC Driver, 55

dacstate_t
 DAC Driver, 56
 data16
 CANRxFrame, 484
 CANTxFrame, 486
 data32
 CANRxFrame, 484
 CANTxFrame, 486
 data8
 CANRxFrame, 484
 CANTxFrame, 485
 day
 RTCDateTime, 529
 dayofweek
 RTCDateTime, 528
 default_config
 Serial Driver, 359
 default_handler
 USB Driver, 438
 depth
 ADCDriver, 465
 DACDriver, 490
 detect_bus_clk
 SDC Driver, 328
 dstflag
 RTCDateTime, 528
 EID
 CANRxFrame, 484
 CANTxFrame, 485
 EP_STATUS_ACTIVE
 USB Driver, 437
 EP_STATUS_DISABLED
 USB Driver, 437
 EP_STATUS_STALLED
 USB Driver, 437
 ETHD1
 MAC Driver, 192
 EXT Driver, 66
 EXT_ACTIVE, 70
 EXT_CH_MODE_AUTOSTART, 69
 EXT_CH_MODE_BOTH_EDGES, 69
 EXT_CH_MODE_DISABLED, 68
 EXT_CH_MODE_EDGES_MASK, 68
 EXT_CH_MODE_FALLING_EDGE, 68
 EXT_CH_MODE_RISING_EDGE, 68
 EXT_MAX_CHANNELS, 70
 EXT_STOP, 70
 EXT_UNINIT, 70
 EXTD1, 75
 EXTDriver, 70
 expchannel_t, 70
 ext_lld_channel_disable, 75
 ext_lld_channel_enable, 75
 ext_lld_init, 74
 ext_lld_start, 74
 ext_lld_stop, 75
 extChannelDisable, 73
 extChannelDisablel, 69
 extChannelEnable, 72
 extChannelEnablel, 69
 extInit, 71
 extObjectInit, 71
 extSetChannelMode, 69
 extSetChannelModel, 73
 extStart, 71
 extStop, 72
 extcallback_t, 70
 extstate_t, 70
 PLATFORM_EXT_USE_EXT1, 70
 EXT_ACTIVE
 EXT Driver, 70
 EXT_CH_MODE_AUTOSTART
 EXT Driver, 69
 EXT_CH_MODE_BOTH_EDGES
 EXT Driver, 69
 EXT_CH_MODE_DISABLED
 EXT Driver, 68
 EXT_CH_MODE_EDGES_MASK
 EXT Driver, 68
 EXT_CH_MODE_FALLING_EDGE
 EXT Driver, 68
 EXT_CH_MODE_RISING_EDGE
 EXT Driver, 68
 EXT_MAX_CHANNELS
 EXT Driver, 70
 EXT_STOP
 EXT Driver, 70
 EXT_UNINIT
 EXT Driver, 70
 EXTChannelConfig, 491
 cb, 492
 mode, 492
 EXTConfig, 493
 channels, 493
 EXTD1
 EXT Driver, 75
 EXTDriver, 494
 config, 494
 EXT Driver, 70
 state, 494
 enabled
 PWMDriver, 527
 end_cb
 ADCConversionGroup, 463
 DACConversionGroup, 488
 I2SConfig, 503
 SPIConfig, 544
 ep0_state
 USB Driver, 459
 ep0config
 USB Driver, 459
 ep0endcb
 USBDriver, 558
 ep0n
 USBDriver, 558
 ep0next

USBDriver, 558
ep0state
 USBDriver, 558
ep_mode
 USBEndpointConfig, 560
epc
 USBDriver, 557
error_cb
 ADCConversionGroup, 463
 DACConversionGroup, 488
error_event
 CANDriver, 482
errors
 I2CDriver, 501
 SDCDriver, 534
event_cb
 USBConfig, 555
event_source, 490
 cb, 491
 flags, 491
 param, 491
event_source_t
 OSAL, 217
eventcallback_t
 OSAL, 217
eventflags_t
 OSAL, 217
expchannel_t
 EXT Driver, 70
ext.c, 579
ext.h, 579
ext_lld.c, 580
ext_lld.h, 581
ext_lld_channel_disable
 EXT Driver, 75
ext_lld_channel_enable
 EXT Driver, 75
ext_lld_init
 EXT Driver, 74
ext_lld_start
 EXT Driver, 74
ext_lld_stop
 EXT Driver, 75
extChannelDisable
 EXT Driver, 73
extChannelDisableI
 EXT Driver, 69
extInit
 EXT Driver, 71
extObjectInit
 EXT Driver, 71
extSetChannelMode
 EXT Driver, 69
extSetChannelModel
 EXT Driver, 73
extStart
 EXT Driver, 71
extStop
 EXT Driver, 72
extcallback_t
 EXT Driver, 70
extstate_t
 EXT Driver, 70
FILE_EOF
 Abstract Files, 103
FILE_ERROR
 Abstract Files, 103
FILE_OK
 Abstract Files, 103
FMI
 CANRxFrame, 483
FileStream, 495
 vmt, 496
fileStreamClose
 Abstract Files, 105
fileStreamGet
 Abstract Files, 105
fileStreamGetPosition
 Abstract Files, 106
fileStreamGetSize
 Abstract Files, 106
fileStreamPut
 Abstract Files, 104
fileStreamRead
 Abstract Files, 104
fileStreamSeek
 Abstract Files, 107
FileStreamVMT, 496
fileStreamWrite
 Abstract Files, 103
fileoffset_t
 Abstract Files, 107
flags
 event_source, 491
frequency
 GPTConfig, 498
 ICUConfig, 505
 PWMConfig, 525
GPT Driver, 76
 GPT_CONTINUOUS, 82
 GPT_ONESHOT, 82
 GPT_READY, 82
 GPT_STOP, 82
 GPT_UNINIT, 82
 GPTD1, 92
 GPTDriver, 82
 gpt_lld_change_interval, 81
 gpt_lld_init, 90
 gpt_lld_polled_delay, 92

gpt_lld_start, 91
gpt_lld_start_timer, 91
gpt_lld_stop, 91
gpt_lld_stop_timer, 91
gptChangeInterval, 85
gptChangeIntervall, 79
gptGetCounterX, 79
gptGetIntervalX, 79
gptInit, 82
gptObjectInit, 83
gptPolledDelay, 90
gptStart, 83
gptStartContinuous, 86
gptStartContinuousl, 86
gptStartOneShot, 88
gptStartOneShotl, 88
gptStop, 83
gptStopTimer, 89
gptStopTimerl, 89
gptcallback_t, 82
gptcnt_t, 82
gptfreq_t, 82
gptstate_t, 82
PLATFORM_GPT_USE_GPT1, 81
GPT_CONTINUOUS
 GPT Driver, 82
GPT_ONESHOT
 GPT Driver, 82
GPT_READY
 GPT Driver, 82
GPT_STOP
 GPT Driver, 82
GPT_UNINIT
 GPT Driver, 82
GPTConfig, 497
 callback, 498
 frequency, 498
GPTD1
 GPT Driver, 92
GPTDriver, 499
 config, 499
 GPT Driver, 82
 state, 499
get_descriptor_cb
 USBConfig, 555
gpt.c, 582
gpt.h, 583
gpt_lld.c, 584
gpt_lld.h, 584
gpt_lld_change_interval
 GPT Driver, 81
gpt_lld_init
 GPT Driver, 90
gpt_lld_polled_delay
 GPT Driver, 92
gpt_lld_start
 GPT Driver, 91
apt_lld_start_timer

GPT Driver, 91
gpt_lld_stop
 GPT Driver, 91
gpt_lld_stop_timer
 GPT Driver, 91
gptChangeInterval
 GPT Driver, 85
gptChangeIntervall
 GPT Driver, 79
gptGetCounterX
 GPT Driver, 79
gptGetIntervalX
 GPT Driver, 79
gptInit
 GPT Driver, 82
gptObjectInit
 GPT Driver, 83
gptPolledDelay
 GPT Driver, 90
gptStart
 GPT Driver, 83
gptStartContinuous
 GPT Driver, 86
gptStartContinuousl
 GPT Driver, 86
gptStartOneShot
 GPT Driver, 88
gptStartOneShotl
 GPT Driver, 88
gptStop
 GPT Driver, 83
gptStopTimer
 GPT Driver, 89
gptStopTimerl
 GPT Driver, 89
gptcallback_t
 GPT Driver, 82
gptcnt_t
 GPT Driver, 82
gptfreq_t
 GPT Driver, 82
gptstate_t
 GPT Driver, 82
grpp
 ADCDriver, 465
 DACDriver, 490
HAL, 193
HAL Driver, 93
 _CHIBIOS_HAL_, 94
 CH_HAL_MAJOR, 94
 CH_HAL_MINOR, 94
 CH_HAL_PATCH, 94
 CH_HAL_STABLE, 94
 HAL_VERSION, 94
 hal_lld_init, 95
 hallnit, 94
HAL_USE_ADC
 Configuration, 198

HAL_USE_CAN
 Configuration, 198
HAL_USE_DAC
 Configuration, 198
HAL_USE_EXT
 Configuration, 198
HAL_USE_GPT
 Configuration, 198
HAL_USE_I2C
 Configuration, 198
HAL_USE_I2S
 Configuration, 198
HAL_USE_ICU
 Configuration, 198
HAL_USE_MAC
 Configuration, 198
HAL_USE_MMCSPI
 Configuration, 198
HAL_USE_PAL
 Configuration, 198
HAL_USE_PWM
 Configuration, 199
HAL_USE_RTC
 Configuration, 199
HAL_USE_SDC
 Configuration, 199
HAL_USE_SERIAL
 Configuration, 199
HAL_USE_SERIAL_USB
 Configuration, 199
HAL_USE_SPI
 Configuration, 199
HAL_USE_UART
 Configuration, 199
HAL_USE_USB
 Configuration, 199
HAL_VERSION
 HAL Driver, 94
hal.c, 585
hal.h, 586
hal_channels.h, 587
hal_files.h, 588
hal_ioblock.h, 589
hal_lld.c, 591
hal_lld.h, 591
hal_lld_init
 HAL Driver, 95
hal_mmcsd.c, 591
hal_mmcsd.h, 592
hal_queues.c, 596
hal_queues.h, 597
hal_streams.h, 599
hallInit
 HAL Driver, 94
halconf.h, 599
hscfg
 MMCConfig, 515
I/O Queues, 115
_INPUTQUEUE_DATA, 120
_OUTPUTQUEUE_DATA, 123
INPUTQUEUE_DECL, 121
input_queue_t, 124
io_queue_t, 124
iqGet, 120
iqGetEmptyI, 119
iqGetFullI, 118
iqGetTimeout, 127
iqIsEmptyI, 119
iqIsFullI, 119
iqObjectInit, 124
iqPutI, 125
iqReadTimeout, 128
iqResetI, 125
OUTPUTQUEUE_DECL, 123
oqGetEmptyI, 121
oqGetFullI, 121
oqGetI, 131
oqlIsEmptyI, 122
oqlIsFullI, 122
oqObjectInit, 129
oqPut, 122
oqPutTimeout, 130
oqResetI, 129
oqWriteTimeout, 132
output_queue_t, 124
Q_EMPTY, 117
Q_FULL, 117
Q_OK, 117
Q_RESET, 117
Q_TIMEOUT, 117
qGetLink, 118
qSizeX, 117
qSpacel, 117
qnotify_t, 124
I2C Driver, 137
 _i2c_wakeup_error_isr, 140
 _i2c_wakeup_isr, 140
 I2C_ACK_FAILURE, 140
 I2C_ACTIVE_RX, 142
 I2C_ACTIVE_TX, 142
 I2C_ARBITRATION_LOST, 140
 I2C_BUS_ERROR, 139
 I2C_NO_ERROR, 139
 I2C_OVERRUN, 140
 I2C_PEC_ERROR, 140
 I2C_READY, 142
 I2C_SMB_ALERT, 140
 I2C_STOP, 142
 I2C_TIMEOUT, 140
 I2C_UNINIT, 142
 I2C_USE_MUTUAL_EXCLUSION, 140
 I2CD1, 150
 I2CDriver, 142
 i2c_lld_get_errors, 141
 i2c_lld_init, 148
 i2c_lld_master_receive_timeout, 149

i2c_lld_master_transmit_timeout, 149
 i2c_lld_start, 148
 i2c_lld_stop, 148
 i2cAcquireBus, 147
 i2cGetErrors, 144
 i2cInit, 142
 i2cMasterReceive, 141
 i2cMasterReceiveTimeout, 146
 i2cMasterTransmit, 141
 i2cMasterTransmitTimeout, 145
 i2cObjectInit, 143
 i2cReleaseBus, 147
 i2cStart, 143
 i2cStop, 144
 i2caddr_t, 142
 i2cflags_t, 142
 i2cstate_t, 142
 PLATFORM_I2C_USE_I2C1, 141
I2C_ACK_FAILURE
 I2C Driver, 140
I2C_ACTIVE_RX
 I2C Driver, 142
I2C_ACTIVE_TX
 I2C Driver, 142
I2C_ARBITRATION_LOST
 I2C Driver, 140
I2C_BUS_ERROR
 I2C Driver, 139
I2C_NO_ERROR
 I2C Driver, 139
I2C_OVERRUN
 I2C Driver, 140
I2C_PEC_ERROR
 I2C Driver, 140
I2C_READY
 I2C Driver, 142
I2C_SMB_ALERT
 I2C Driver, 140
I2C_STOP
 I2C Driver, 142
I2C_TIMEOUT
 I2C Driver, 140
I2C_UNINIT
 I2C Driver, 142
I2C_USE_MUTUAL_EXCLUSION
 Configuration, 200
 I2C Driver, 140
I2CConfig, 499
I2CD1
 I2C Driver, 150
I2CDriver, 500
 config, 501
 errors, 501
 I2C Driver, 142
 state, 501
I2S_Driver, 151
 _i2s_isr_full_code, 153
 _i2s_isr_half_code, 153
 I2S_ACTIVE, 154
 I2S_COMPLETE, 154
 I2S_READY, 154
 I2S_STOP, 154
 I2S_UNINIT, 154
I2SD1, 158
I2SDriver, 154
 i2s_lld_init, 157
 i2s_lld_start, 158
 i2sInit, 155
 i2sObjectInit, 155
 i2sStart, 155
 i2sStartExchange, 156
 i2sStartExchangel, 152
 i2sStop, 156
 i2sStopExchange, 157
 i2sStopExchangel, 152
 i2scallback_t, 154
 i2sstate_t, 154
 PLATFORM_I2S_USE_I2S1, 154
I2S_ACTIVE
 I2S Driver, 154
I2S_COMPLETE
 I2S Driver, 154
I2S_READY
 I2S Driver, 154
I2S_STOP
 I2S Driver, 154
I2S_UNINIT
 I2S Driver, 154
I2SConfig, 501
 end_cb, 503
 rx_buffer, 502
 size, 503
 tx_buffer, 502
I2SD1
 I2S Driver, 158
I2SDriver, 503
 config, 504
 I2S Driver, 154
 state, 504
 i2c.c, 601
 i2c.h, 602
 i2c_lld.c, 604
 i2c_lld.h, 604
 i2c_lld_get_errors
 I2C Driver, 141
 i2c_lld_init
 I2C Driver, 148
 i2c_lld_master_receive_timeout
 I2C Driver, 149
 i2c_lld_master_transmit_timeout
 I2C Driver, 149
 i2c_lld_start
 I2C Driver, 148
 i2c_lld_stop
 I2C Driver, 148
 i2cAcquireBus

I2C Driver, 147
i2cGetErrors
 I2C Driver, 144
i2cInit
 I2C Driver, 142
i2cMasterReceive
 I2C Driver, 141
i2cMasterReceiveTimeout
 I2C Driver, 146
i2cMasterTransmit
 I2C Driver, 141
i2cMasterTransmitTimeout
 I2C Driver, 145
i2cObjectInit
 I2C Driver, 143
i2cReleaseBus
 I2C Driver, 147
i2cStart
 I2C Driver, 143
i2cStop
 I2C Driver, 144
i2caddr_t
 I2C Driver, 142
i2cflags_t
 I2C Driver, 142
i2cstate_t
 I2C Driver, 142
i2s.c, 605
i2s.h, 606
i2s_lld.c, 607
i2s_lld.h, 607
i2s_lld_init
 I2S Driver, 157
i2s_lld_start
 I2S Driver, 158
i2sInit
 I2S Driver, 155
i2sObjectInit
 I2S Driver, 155
i2sStart
 I2S Driver, 155
i2sStartExchange
 I2S Driver, 156
i2sStartExchangel
 I2S Driver, 152
i2sStop
 I2S Driver, 156
i2sStopExchange
 I2S Driver, 157
i2sStopExchangel
 I2S Driver, 152
i2scallback_t
 I2S Driver, 154
i2sstate_t
 I2S Driver, 154
ICU Driver, 159
 _icu_isr_invoke_overflow_cb, 165
 _icu_isr_invoke_period_cb, 165
 _icu_isr_invoke_width_cb, 164
 ICU_ACTIVE, 167
 ICU_INPUT_ACTIVE_HIGH, 168
 ICU_INPUT_ACTIVE_LOW, 168
 ICU_READY, 167
 ICU_STOP, 167
 ICU_UNINIT, 167
 ICU_WAITING, 167
 ICUD1, 176
 ICUDriver, 167
 icu_lld_are_notifications_enabled, 166
 icu_lld_disable_notifications, 176
 icu_lld_enable_notifications, 176
 icu_lld_get_period, 166
 icu_lld_get_width, 166
 icu_lld_init, 174
 icu_lld_start, 174
 icu_lld_start_capture, 175
 icu_lld_stop, 174
 icu_lld_stop_capture, 175
 icu_lld_wait_capture, 175
 icuAreNotificationsEnabledX, 163
 icuDisableNotifications, 173
 icuDisableNotificationsI, 163
 icuEnableNotifications, 172
 icuEnableNotificationsI, 163
 icuGetPeriodX, 164
 icuGetWidthX, 164
 icuInit, 168
 icuObjectInit, 168
 icuStart, 168
 icuStartCapture, 169
 icuStartCaptureI, 162
 icuStop, 169
 icuStopCapture, 172
 icuStopCaptureI, 162
 icuWaitCapture, 171
 icucallback_t, 167
 icucnt_t, 167
 icufreq_t, 167
 icumode_t, 167
 icustate_t, 167
 PLATFORM_ICU_USE_ICU1, 166
 ICU_ACTIVE
 ICU Driver, 167
 ICU_INPUT_ACTIVE_HIGH
 ICU Driver, 168
 ICU_INPUT_ACTIVE_LOW
 ICU Driver, 168
 ICU_READY
 ICU Driver, 167
 ICU_STOP
 ICU Driver, 167
 ICU_UNINIT
 ICU Driver, 167
 ICU_WAITING
 ICU Driver, 167
 ICUConfig, 504

frequency, 505
 mode, 505
 overflow_cb, 505
 period_cb, 505
 width_cb, 505
ICUD1
 ICU Driver, 176
ICUDriver, 505
 config, 506
 ICU Driver, 167
 state, 506
IDE
 CANRxFrame, 483
 CANTxFrame, 485
INPUTQUEUE_DECL
 I/O Queues, 121
IOBUS_DECL
 PAL Driver, 267
IOBus, 508
 mask, 509
 offset, 509
 portid, 509
IOPORT1
 PAL Driver, 274
icu.c, 608
icu.h, 609
icu_lld.c, 610
icu_lld.h, 611
icu_lld_are_notifications_enabled
 ICU Driver, 166
icu_lld_disable_notifications
 ICU Driver, 176
icu_lld_enable_notifications
 ICU Driver, 176
icu_lld_get_period
 ICU Driver, 166
icu_lld_get_width
 ICU Driver, 166
icu_lld_init
 ICU Driver, 174
icu_lld_start
 ICU Driver, 174
icu_lld_start_capture
 ICU Driver, 175
icu_lld_stop
 ICU Driver, 174
icu_lld_stop_capture
 ICU Driver, 175
icu_lld_wait_capture
 ICU Driver, 175
icuAreNotificationsEnabledX
 ICU Driver, 163
icuDisableNotifications
 ICU Driver, 173
icuDisableNotificationsI
 ICU Driver, 163
icuEnableNotifications
 ICU Driver, 172
icuEnableNotificationsI
 ICU Driver, 163
icuGetPeriodX
 ICU Driver, 164
icuGetWidthX
 ICU Driver, 164
icuInit
 ICU Driver, 168
icuObjectInit
 ICU Driver, 168
icuStart
 ICU Driver, 168
icuStartCapture
 ICU Driver, 169
icuStartCaptureI
 ICU Driver, 162
icuStop
 ICU Driver, 169
icuStopCapture
 ICU Driver, 172
icuStopCaptureI
 ICU Driver, 162
icuWaitCapture
 ICU Driver, 171
icucallback_t
 ICU Driver, 167
icucnt_t
 ICU Driver, 167
icufreq_t
 ICU Driver, 167
icumode_t
 ICU Driver, 167
icustate_t
 ICU Driver, 167
in
 USB Driver, 459
usb_lld.c, 663
in_cb
 USBEndpointConfig, 560
in_maxsize
 USBEndpointConfig, 561
in_params
 USBDriver, 558
in_state
 USBEndpointConfig, 561
Inner Code, 205
inotify
 Serial over USB Driver, 364
input_queue_t
 I/O Queues, 124
int_in
 SerialUSBConfig, 540
Interfaces, 204
io_queue, 506
 q_buffer, 508
 q_counter, 508
 q_link, 508
 q_notify, 508

q_rdptr, 508
q_top, 508
q_waiting, 508
q_wptr, 508
io_queue_t
 I/O Queues, 124
iomode_t
 PAL Driver, 281
ioportid_t
 PAL Driver, 281
ioportmask_t
 PAL Driver, 281
iqGet
 I/O Queues, 120
iqGetEmpty()
 I/O Queues, 119
iqGetFull()
 I/O Queues, 118
iqGetTimeout()
 I/O Queues, 127
iqIsEmpty()
 I/O Queues, 119
iqIsFull()
 I/O Queues, 119
iqObjectInit()
 I/O Queues, 124
iqPut()
 I/O Queues, 125
iqReadTimeout()
 I/O Queues, 128
iqReset()
 I/O Queues, 125
lscfg
 MMCConfig, 514
MAC Driver, 177
 ETHD1, 192
 MAC_ACTIVE, 181
 MAC_STOP, 181
 MAC_SUPPORTS_ZERO_COPY, 181
 MAC_UNINIT, 181
 MAC_USE_EVENTS, 179
 MAC_USE_ZERO_COPY, 179
MACDriver, 181
mac_lld_get_next_receive_buffer, 192
mac_lld_get_next_transmit_buffer, 191
mac_lld_get_receive_descriptor, 190
mac_lld_get_transmit_descriptor, 189
mac_lld_init, 188
mac_lld_poll_link_status, 190
mac_lld_read_receive_descriptor, 191
mac_lld_release_receive_descriptor, 190
mac_lld_release_transmit_descriptor, 190
mac_lld_start, 189
mac_lld_stop, 189
mac_lld_write_transmit_descriptor, 191
macGetNextReceiveBuffer, 180
macGetNextTransmitBuffer, 180
macGetReceiveEventSource, 179
macInit, 182
macObjectInit, 182
macPollLinkStatus, 188
macReadReceiveDescriptor, 180
macReleaseReceiveDescriptor, 187
macReleaseTransmitDescriptor, 184
macStart, 182
macStop, 183
macWaitReceiveDescriptor, 186
macWaitTransmitDescriptor, 183
macWriteTransmitDescriptor, 179
macstate_t, 181
 PLATFORM_MAC_USE_MAC1, 181
MAC_ACTIVE
 MAC Driver, 181
MAC_STOP
 MAC Driver, 181
MAC_SUPPORTS_ZERO_COPY
 MAC Driver, 181
MAC_UNINIT
 MAC Driver, 181
MAC_USE_EVENTS
 Configuration, 200
 MAC Driver, 179
MAC_USE_ZERO_COPY
 Configuration, 200
 MAC Driver, 179
MACConfig, 510
 mac_address, 510
MACDriver, 510
 config, 511
 MAC Driver, 181
 rdevent, 512
 rdqueue, 511
 state, 511
 tdqueue, 511
MACReceiveDescriptor, 512
 offset, 512
 size, 512
MACTransmitDescriptor, 513
 offset, 513
 size, 513
MMC over SPI Driver, 229
 mmc_driver_methods, 231
 crc7, 232
 crc7_lookup_table, 249
 MMC_NICE_WAITING, 231
 mmc_vmt, 249
 mmcConnect, 239
 mmcDisconnect, 240
 mmcErase, 248
 mmcGetInfo, 248
 mmcInit, 238
 mmclsCardInserted, 231
 mmclsWriteProtected, 231
 mmcObjectInit, 238
 mmcSequentialRead, 242

mmcSequentialWrite, 245
 mmcStart, 238
 mmcStartSequentialRead, 241
 mmcStartSequentialWrite, 244
 mmcStop, 238
 mmcStopSequentialRead, 242
 mmcStopSequentialWrite, 245
 mmcSync, 247
 read_CxD, 236
 recvr1, 233
 recvr3, 234
 send_command_R1, 235
 send_command_R3, 235
 send_hdr, 233
 sync, 237
 wait, 232
MMC/SD Block Device, 251
 _mmcsd_block_device_data, 256
 _mmcsd_block_device_methods, 255
 _mmcsd_get_capacity, 257
 _mmcsd_get_capacity_ext, 258
 _mmcsd_get_slice, 257
 _mmcsd_unpack_csd_mmc, 259
 _mmcsd_unpack_csd_v10, 259
 _mmcsd_unpack_csd_v20, 260
 _mmcsd_unpack_mmc_cid, 258
 _mmcsd_unpack_sdc_cid, 258
MMCSD_BLOCK_SIZE, 255
MMCSD_CID_SDC_CRC_SLICE, 255
MMCSD_CMD8_PATTERN, 255
MMCSD_CSD_MMC_CSD_STRUCTURE_SLICE, 255
MMCSD_R1_ERROR, 256
MMCSD_R1_ERROR_MASK, 255
MMCSD_R1_IS_CARD_LOCKED, 256
MMCSD_R1_STS, 256
 mmcsdGetCardCapacity, 256
MMC_NICE_WAITING
 Configuration, 200
 MMC over SPI Driver, 231
MMCConfig, 513
 hscfg, 515
 lscfg, 514
 spip, 514
MMCDriver, 515
 config, 516
 vmt, 516
MMCDriverVMT, 517
MMCSD_BLOCK_SIZE
 MMC/SD Block Device, 255
MMCSD_CID_SDC_CRC_SLICE
 MMC/SD Block Device, 255
MMCSD_CMD8_PATTERN
 MMC/SD Block Device, 255
MMCSD_CSD_MMC_CSD_STRUCTURE_SLICE
 MMC/SD Block Device, 255
MMCSD_R1_ERROR
 MMC/SD Block Device, 256
MMCSD_R1_ERROR_MASK
 MMC/SD Block Device, 255
MMCSD_R1_IS_CARD_LOCKED
 MMC/SD Block Device, 256
MMCSD_R1_STS
 MMC/SD Block Device, 256
MMCSDBlockDevice, 518
 vmt, 520
MMCSDBlockDeviceVMT, 520
mac.c, 612
mac.h, 613
mac_address
 MACConfig, 510
mac_Ild.c, 614
mac_Ild.h, 615
mac_Ild_get_next_receive_buffer
 MAC Driver, 192
mac_Ild_get_next_transmit_buffer
 MAC Driver, 191
mac_Ild_get_receive_descriptor
 MAC Driver, 190
mac_Ild_get_transmit_descriptor
 MAC Driver, 189
mac_Ild_init
 MAC Driver, 188
mac_Ild_poll_link_status
 MAC Driver, 190
mac_Ild_read_receive_descriptor
 MAC Driver, 191
mac_Ild_release_receive_descriptor
 MAC Driver, 190
mac_Ild_release_transmit_descriptor
 MAC Driver, 190
mac_Ild_start
 MAC Driver, 189
mac_Ild_stop
 MAC Driver, 189
mac_Ild_write_transmit_descriptor
 MAC Driver, 191
macGetNextReceiveBuffer
 MAC Driver, 180
macGetNextTransmitBuffer
 MAC Driver, 180
macGetReceiveEventSource
 MAC Driver, 179
macInit
 MAC Driver, 182
macObjectInit
 MAC Driver, 182
macPollLinkStatus
 MAC Driver, 188
macReadReceiveDescriptor
 MAC Driver, 180
macReleaseReceiveDescriptor
 MAC Driver, 187
macReleaseTransmitDescriptor
 MAC Driver, 184
macStart

MAC Driver, 182
macStop
 MAC Driver, 183
macWaitReceiveDescriptor
 MAC Driver, 186
macWaitTransmitDescriptor
 MAC Driver, 183
macWriteTransmitDescriptor
 MAC Driver, 179
macstate_t
 MAC Driver, 181
mask
 IOBus, 509
millisecond
 RTCDateTime, 529
mmc_cmd6_construct
 SDC Driver, 325
mmc_detect_bus_clk
 SDC Driver, 327
mmc_init
 SDC Driver, 323
mmc_set_bus_width
 SDC Driver, 329
mmc_spi.c, 616
mmc_spi.h, 617
mmc_switch_t
 SDC Driver, 322
mmc_vmt
 MMC over SPI Driver, 249
mmcConnect
 MMC over SPI Driver, 239
mmcDisconnect
 MMC over SPI Driver, 240
mmcErase
 MMC over SPI Driver, 248
mmcGetInfo
 MMC over SPI Driver, 248
mmclInit
 MMC over SPI Driver, 238
mmclsCardInserted
 MMC over SPI Driver, 231
mmclsWriteProtected
 MMC over SPI Driver, 231
mmcObjectInit
 MMC over SPI Driver, 238
mmcSequentialRead
 MMC over SPI Driver, 242
mmcSequentialWrite
 MMC over SPI Driver, 245
mmcStart
 MMC over SPI Driver, 238
mmcStartSequentialRead
 MMC over SPI Driver, 241
mmcStartSequentialWrite
 MMC over SPI Driver, 244
mmcStop
 MMC over SPI Driver, 238
mmcStopSequentialRead
 MMC over SPI Driver, 242
MMC over SPI Driver, 242
mmcStopSequentialWrite
 MMC over SPI Driver, 245
mmcSync
 MMC over SPI Driver, 247
mmcSdGetCardCapacity
 MMC/SD Block Device, 256
mode
 EXTChannelConfig, 492
 ICUConfig, 505
 PWMChannelConfig, 523
mode_detect
 SDC Driver, 323
month
 RTCDateTime, 528
msg_t
 OSAL, 216
mutex
 ADCDriver, 465
 DACDriver, 490
 SPIDriver, 546
mutex_t
 OSAL, 217
Normal Drivers, 202
num_channels
 ADCConversionGroup, 463
 DACConversionGroup, 488
OSAL, 206
 event_source_t, 217
 eventcallback_t, 217
 eventflags_t, 217
 msg_t, 216
 mutex_t, 217
 OSAL_DBG_ENABLE_ASSERTS, 211
 OSAL_DBG_ENABLE_CHECKS, 211
 OSAL_IRQ_EPILOGUE, 213
 OSAL_IRQ_HANDLER, 213
 OSAL_IRQ_IS_VALID_PRIORITY, 212
 OSAL_IRQ_MAXIMUM_PRIORITY, 211
 OSAL_IRQ_PRIORITY_LEVELS, 211
 OSAL_IRQ_PROLOGUE, 213
 OSAL_MS2RTC, 215
 OSAL_MS2ST, 213
 OSAL_S2RTC, 214
 OSAL_S2ST, 213
 OSAL_ST_FREQUENCY, 211
 OSAL_ST_MODE, 211
 OSAL_ST_RESOLUTION, 211
 OSAL_US2RTC, 215
 OSAL_US2ST, 214
 osal_halt_msg, 227
 osalDbgAssert, 211
 osalDbgCheck, 212
 osalDbgCheckClassI, 212
 osalDbgCheckClassS, 212
 osalEventBroadcastFlags, 223
 osalEventBroadcastFlagsI, 222

osalEventObjectInit, 227
 osalEventSetCallback, 223
 osalInit, 217
 osalMutexLock, 224
 osalMutexObjectInit, 227
 osalMutexUnlock, 224
 osalOsGetSystemTimeX, 219
 osalOsIsTimeWithinX, 226
 osalOsRescheduleS, 218
 osalOsTimerHandlerI, 218
 osalSysDisable, 224
 osalSysEnable, 224
 osalSysGetStatusAndLockX, 225
 osalSysHalt, 218
 osalSysLock, 225
 osalSysLockFromISR, 225
 osalSysPolledDelayX, 218
 osalSysRestoreStatusX, 226
 osalSysUnlock, 225
 osalSysUnlockFromISR, 225
 osalThreadDequeueAllI, 222
 osalThreadDequeueNextI, 222
 osalThreadEnqueueTimeoutS, 221
 osalThreadQueueObjectInit, 227
 osalThreadResumeI, 221
 osalThreadResumeS, 221
 osalThreadSleep, 219
 osalThreadSleepMicroseconds, 216
 osalThreadSleepMilliseconds, 216
 osalThreadSleepS, 219
 osalThreadSleepSeconds, 215
 osalThreadSuspendS, 220
 osalThreadSuspendTimeoutS, 220
 rtcnt_t, 217
 syssts_t, 216
 systime_t, 217
 thread_reference_t, 217
 OSAL_DBG_ENABLE_ASSERTS
 OSAL, 211
 OSAL_DBG_ENABLE_CHECKS
 OSAL, 211
 OSAL_IRQ_EPILOGUE
 OSAL, 213
 OSAL_IRQ_HANDLER
 OSAL, 213
 OSAL_IRQ_IS_VALID_PRIORITY
 OSAL, 212
 OSAL_IRQ_MAXIMUM_PRIORITY
 OSAL, 211
 OSAL_IRQ_PRIORITY_LEVELS
 OSAL, 211
 OSAL_IRQ_PROLOGUE
 OSAL, 213
 OSAL_MS2RTC
 OSAL, 215
 OSAL_MS2ST
 OSAL, 213
 OSAL_S2RTC
 OSAL, 214
 OSAL_S2ST
 OSAL, 213
 OSAL_ST_FREQUENCY
 OSAL, 211
 OSAL_ST_MODE
 OSAL, 211
 OSAL_ST_RESOLUTION
 OSAL, 211
 OSAL_US2RTC
 OSAL, 215
 OSAL_US2ST
 OSAL, 214
 OUTPUTQUEUE_DECL
 I/O Queues, 123
 offset
 IOBus, 509
 MACReceiveDescriptor, 512
 MACTransmitDescriptor, 513
 onnotify
 Serial over USB Driver, 365
 oqGetEmptyI
 I/O Queues, 121
 oqGetFullI
 I/O Queues, 121
 oqGetI
 I/O Queues, 131
 oqlIsEmptyI
 I/O Queues, 122
 oqlIsFullI
 I/O Queues, 122
 oqObjectInit
 I/O Queues, 129
 oqPut
 I/O Queues, 122
 oqPutTimeout
 I/O Queues, 130
 oqResetI
 I/O Queues, 129
 oqWriteTimeout
 I/O Queues, 132
 osal.c, 619
 osal.h, 620
 osal_halt_msg
 OSAL, 227
 osalDbgAssert
 OSAL, 211
 osalDbgCheck
 OSAL, 212
 osalDbgCheckClassI
 OSAL, 212
 osalDbgCheckClassS
 OSAL, 212
 osalEventBroadcastFlags
 OSAL, 223
 osalEventBroadcastFlagsI
 OSAL, 222
 osalEventObjectInit

OSAL, 227
osalEventSetCallback
 OSAL, 223
osalInit
 OSAL, 217
osalMutexLock
 OSAL, 224
osalMutexObjectInit
 OSAL, 227
osalMutexUnlock
 OSAL, 224
osalOsGetSystemTimeX
 OSAL, 219
osalOsIsTimeWithinX
 OSAL, 226
osalOsRescheduleS
 OSAL, 218
osalOsTimerHandlerI
 OSAL, 218
osalSysDisable
 OSAL, 224
osalSysEnable
 OSAL, 224
osalSysGetStatusAndLockX
 OSAL, 225
osalSysHalt
 OSAL, 218
osalSysLock
 OSAL, 225
osalSysLockFromISR
 OSAL, 225
osalSysPolledDelayX
 OSAL, 218
osalSysRestoreStatusX
 OSAL, 226
osalSysUnlock
 OSAL, 225
osalSysUnlockFromISR
 OSAL, 225
osalThreadDequeueAll
 OSAL, 222
osalThreadDequeueNextI
 OSAL, 222
osalThreadEnqueueTimeoutS
 OSAL, 221
osalThreadQueueObjectInit
 OSAL, 227
osalThreadResumeI
 OSAL, 221
osalThreadResumeS
 OSAL, 221
osalThreadSleep
 OSAL, 219
osalThreadSleepMicroseconds
 OSAL, 216
osalThreadSleepMilliseconds
 OSAL, 216
osalThreadSleepS
 OSAL, 219
osalThreadSleepSeconds
 OSAL, 215
osalThreadSuspendS
 OSAL, 220
osalThreadSuspendTimeoutS
 OSAL, 220
out
 USB Driver, 459
 usb_lld.c, 663
out_cb
 USBEndpointConfig, 560
out_maxsize
 USBEndpointConfig, 561
out_params
 USBDriver, 558
out_state
 USBEndpointConfig, 561
output_queue_t
 I/O Queues, 124
overflow_cb
 ICUConfig, 505

PAL Driver, 261
 _IOBUS_DATA, 267
 _pal_lld_init, 282
 _pal_lld_setgroupmode, 282
 IOBUS_DECL, 267
 IOPORT1, 274
 iomode_t, 281
 iportid_t, 281
 iportmask_t, 281
 PAL_GROUP_MASK, 267
 PAL_HIGH, 265
 PAL_IOPORTS_WIDTH, 274
 PAL_LOW, 265
 PAL_MODE_INPUT, 265
 PAL_MODE_INPUT_ANALOG, 265
 PAL_MODE_INPUT_PULLDOWN, 265
 PAL_MODE_INPUT_PULLUP, 265
 PAL_MODE_OUTPUT_OPENDRAIN, 265
 PAL_MODE_OUTPUT_PUSHPULL, 265
 PAL_MODE_RESET, 265
 PAL_MODE_UNCONNECTED, 265
 PAL_PORT_BIT, 265
 PAL_WHOLE_PORT, 274
 pal_lld_clearpad, 279
 pal_lld_clearport, 276
 pal_lld_init, 274
 pal_lld_readgroup, 277
 pal_lld_readlatch, 275
 pal_lld_readpad, 278
 pal_lld_readport, 274
 pal_lld_setgroupmode, 278
 pal_lld_setpad, 279
 pal_lld_setpadmode, 280
 pal_lld_setport, 275
 pal_lld_togglepad, 280
 pal_lld_toggleport, 276

pal_lld_writegroup, 277
 pal_lld_writepad, 278
 pal_lld_writeport, 275
 palClearPad, 273
 palClearPort, 269
 pallInit, 267
 palReadBus, 281
 palReadGroup, 270
 palReadLatch, 268
 palReadPad, 271
 palReadPort, 268
 palSetBusMode, 282
 palSetGroupMode, 271
 palSetPad, 272
 palSetPadMode, 273
 palSetPort, 269
 palTogglePad, 273
 palTogglePort, 269
 palWriteBus, 281
 palWriteGroup, 270
 palWritePad, 272
 palWritePort, 268
PAL_GROUP_MASK
 PAL Driver, 267
PAL_HIGH
 PAL Driver, 265
PAL_IOPORTS_WIDTH
 PAL Driver, 274
PAL_LOW
 PAL Driver, 265
PAL_MODE_INPUT
 PAL Driver, 265
PAL_MODE_INPUT_ANALOG
 PAL Driver, 265
PAL_MODE_INPUT_PULLDOWN
 PAL Driver, 265
PAL_MODE_INPUT_PULLUP
 PAL Driver, 265
PAL_MODE_OUTPUT_OPENDRAIN
 PAL Driver, 265
PAL_MODE_OUTPUT_PUSH_PULL
 PAL Driver, 265
PAL_MODE_RESET
 PAL Driver, 265
PAL_MODE_UNCONNECTED
 PAL Driver, 265
PAL_PORT_BIT
 PAL Driver, 265
PAL_WHOLE_PORT
 PAL Driver, 274
PALConfig, 522
PLATFORM_ADC_USE_ADC1
 ADC Driver, 24
PLATFORM_CAN_USE_CAN1
 CAN Driver, 38
PLATFORM_DAC_USE_DAC1
 DAC Driver, 55
PLATFORM_EXT_USE_EXT1
 EXT Driver, 70
PLATFORM_GPT_USE_GPT1
 GPT Driver, 81
PLATFORM_I2C_USE_I2C1
 I2C Driver, 141
PLATFORM_I2S_USE_I2S1
 I2S Driver, 154
PLATFORM_ICU_USE_ICU1
 ICU Driver, 166
PLATFORM_MAC_USE_MAC1
 MAC Driver, 181
PLATFORM_PWM_USE_PWM1
 PWM Driver, 294
PLATFORM_RTC_USE_RTC1
 RTC Driver, 309
PLATFORM_SDC_USE_SDC1
 SDC Driver, 322
PLATFORM_SERIAL_USE_USART1
 Serial Driver, 352
PLATFORM_SPI_USE_SPI1
 SPI Driver, 379
PLATFORM_UART_USE_UART1
 UART Driver, 405
PLATFORM_USB_USE_USB1
 USB Driver, 434
PWM Driver, 285
 PLATFORM_PWM_USE_PWM1, 294
 PWM_CHANNELS, 293
 PWM_DEGREES_TO_WIDTH, 289
 PWM_FRACTION_TO_WIDTH, 288
 PWM_OUTPUT_ACTIVE_HIGH, 288
 PWM_OUTPUT_ACTIVE_LOW, 288
 PWM_OUTPUT_DISABLED, 288
 PWM_OUTPUT_MASK, 288
 PWM_PERCENTAGE_TO_WIDTH, 289
 PWM_READY, 295
 PWM_STOP, 295
 PWM_UNINIT, 295
 PWMD1, 306
 PWMDriver, 294
 pwm_lld_change_period, 294
 pwm_lld_disable_channel, 304
 pwm_lld_disable_channel_notification, 305
 pwm_lld_disable_periodic_notification, 304
 pwm_lld_enable_channel, 303
 pwm_lld_enable_channel_notification, 305
 pwm_lld_enable_periodic_notification, 304
 pwm_lld_init, 302
 pwm_lld_start, 303
 pwm_lld_stop, 303
 pwmChangePeriod, 297
 pwmChangePeriodl, 290
 pwmDisableChannel, 299
 pwmDisableChannell, 291
 pwmDisableChannelNotification, 302
 pwmDisableChannelNotificationl, 293
 pwmDisablePeriodicNotification, 300
 pwmDisablePeriodicNotificationl, 292

pwmEnableChannel, 298
pwmEnableChannell, 290
pwmEnableChannelNotification, 301
pwmEnableChannelNotificationl, 293
pwmEnablePeriodicNotification, 300
pwmEnablePeriodicNotificationl, 292
pwmInit, 295
pwmIsChannelEnabledl, 292
pwmObjectInit, 296
pwmStart, 296
pwmStop, 297
pwmcallback_t, 294
pwmchannel_t, 295
pwmchnmsk_t, 295
pwmcnt_t, 295
pwmemode_t, 295
pwmstate_t, 295
PWM_CHANNELS
 PWM Driver, 293
PWM_DEGREES_TO_WIDTH
 PWM Driver, 289
PWM_FRACTION_TO_WIDTH
 PWM Driver, 288
PWM_OUTPUT_ACTIVE_HIGH
 PWM Driver, 288
PWM_OUTPUT_ACTIVE_LOW
 PWM Driver, 288
PWM_OUTPUT_DISABLED
 PWM Driver, 288
PWM_OUTPUT_MASK
 PWM Driver, 288
PWM_PERCENTAGE_TO_WIDTH
 PWM Driver, 289
PWM_READY
 PWM Driver, 295
PWM_STOP
 PWM Driver, 295
PWM_UNINIT
 PWM Driver, 295
PWMChannelConfig, 522
 callback, 523
 mode, 523
PWMConfig, 524
 callback, 525
 channels, 525
 frequency, 525
 period, 525
PWMD1
 PWM Driver, 306
PWMDriver, 525
 channels, 527
 config, 527
 enabled, 527
 PWM Driver, 294
 period, 527
 state, 526
pal.c, 624
pal.h, 624
pal_lld.c, 626
pal_lld.h, 626
pal_lld_clearpad
 PAL Driver, 279
pal_lld_clearport
 PAL Driver, 276
pal_lld_init
 PAL Driver, 274
pal_lld_readgroup
 PAL Driver, 277
pal_lld_readlatch
 PAL Driver, 275
pal_lld_readpad
 PAL Driver, 278
pal_lld_readport
 PAL Driver, 274
pal_lld_setgroupmode
 PAL Driver, 278
pal_lld_setpad
 PAL Driver, 279
pal_lld_setpadmode
 PAL Driver, 280
pal_lld_setport
 PAL Driver, 275
pal_lld_togglepad
 PAL Driver, 280
pal_lld_toggleport
 PAL Driver, 276
pal_lld_writegroup
 PAL Driver, 277
pal_lld_writepad
 PAL Driver, 278
pal_lld_writeport
 PAL Driver, 275
palClearPad
 PAL Driver, 273
palClearPort
 PAL Driver, 269
pallnit
 PAL Driver, 267
palReadBus
 PAL Driver, 281
palReadGroup
 PAL Driver, 270
palReadLatch
 PAL Driver, 268
palReadPad
 PAL Driver, 271
palReadPort
 PAL Driver, 268
palSetBusMode
 PAL Driver, 282
palSetGroupMode
 PAL Driver, 271
palSetPad
 PAL Driver, 272
palSetPadMode
 PAL Driver, 273

palSetPort
 PAL Driver, 269
 palTogglePad
 PAL Driver, 273
 palTogglePort
 PAL Driver, 269
 palWriteBus
 PAL Driver, 281
 palWriteGroup
 PAL Driver, 270
 palWritePad
 PAL Driver, 272
 palWritePort
 PAL Driver, 268
 param
 event_source, 491
 period
 PWMConfig, 525
 PWMDriver, 527
 period_cb
 ICUConfig, 505
 portid
 IOBus, 509
 pwm.c, 628
 pwm.h, 628
 pwm_lld.c, 630
 pwm_lld.h, 631
 pwm_lld_change_period
 PWM Driver, 294
 pwm_lld_disable_channel
 PWM Driver, 304
 pwm_lld_disable_channel_notification
 PWM Driver, 305
 pwm_lld_disable_periodic_notification
 PWM Driver, 304
 pwm_lld_enable_channel
 PWM Driver, 303
 pwm_lld_enable_channel_notification
 PWM Driver, 305
 pwm_lld_enable_periodic_notification
 PWM Driver, 304
 pwm_lld_init
 PWM Driver, 302
 pwm_lld_start
 PWM Driver, 303
 pwm_lld_stop
 PWM Driver, 303
 pwmChangePeriod
 PWM Driver, 297
 pwmChangePeriodl
 PWM Driver, 290
 pwmDisableChannel
 PWM Driver, 299
 pwmDisableChannell
 PWM Driver, 291
 pwmDisableChannelNotification
 PWM Driver, 302
 pwmDisableChannelNotificationl
 PWM Driver, 293
 pwmDisablePeriodicNotification
 PWM Driver, 300
 pwmDisablePeriodicNotificationl
 PWM Driver, 292
 pwmEnableChannel
 PWM Driver, 298
 pwmEnableChannell
 PWM Driver, 290
 pwmEnableChannelNotification
 PWM Driver, 301
 pwmEnableChannelNotificationl
 PWM Driver, 293
 pwmEnablePeriodicNotification
 PWM Driver, 300
 pwmEnablePeriodicNotificationl
 PWM Driver, 292
 pwmlInit
 PWM Driver, 295
 pwmlsChannelEnabledl
 PWM Driver, 292
 pwmObjectInit
 PWM Driver, 296
 pwmStart
 PWM Driver, 296
 pwmStop
 PWM Driver, 297
 pwmcallback_t
 PWM Driver, 294
 pwmchannel_t
 PWM Driver, 295
 pwmchnmsk_t
 PWM Driver, 295
 pwcmt_t
 PWM Driver, 295
 pwmmode_t
 PWM Driver, 295
 pwmstate_t
 PWM Driver, 295
 Q_EMPTY
 I/O Queues, 117
 Q_FULL
 I/O Queues, 117
 Q_OK
 I/O Queues, 117
 Q_RESET
 I/O Queues, 117
 Q_TIMEOUT
 I/O Queues, 117
 q_buffer
 io_queue, 508
 q_counter
 io_queue, 508
 q_link
 io_queue, 508
 q_notify
 io_queue, 508
 q_rptr

io_queue, 508
q_top
 io_queue, 508
q_waiting
 io_queue, 508
q_wptr
 io_queue, 508
qGetLink
 I/O Queues, 118
qSizeX
 I/O Queues, 117
qSpaceI
 I/O Queues, 117
qnotify_t
 I/O Queues, 124

RTC Driver, 307
 rtc_driver_methods, 309
 PLATFORM_RTC_USE_RTC1, 309
 RTC_ALARMS, 309
 RTC_BASE_YEAR, 309
 RTC_HAS_STORAGE, 309
 RTC_SUPPORTS_CALLBACKS, 309
 RTCD1, 316
 RTCDriver, 310
 rtc_lld_get_alarm, 316
 rtc_lld_get_time, 315
 rtc_lld_init, 314
 rtc_lld_set_alarm, 315
 rtc_lld_set_time, 315
 rtcConvertDateTimeToFAT, 314
 rtcConvertDateTimeToStructTm, 313
 rtcConvertStructTmToDateTm, 314
 rtcGetAlarm, 312
 rtcGetTime, 311
 rtcInit, 310
 rtcObjectInit, 310
 rtcSetAlarm, 312
 rtcSetCallback, 313
 rtcSetTime, 311
 rtcalarm_t, 310
 rtccb_t, 310
 rtcevent_t, 310

 RTC_ALARMS
 RTC Driver, 309
 RTC_BASE_YEAR
 RTC Driver, 309
 RTC_HAS_STORAGE
 RTC Driver, 309
 RTC_SUPPORTS_CALLBACKS
 RTC Driver, 309

 RTCAlarm, 527
 RTCD1
 RTC Driver, 316
 RTCDateTime, 527
 day, 529
 dayofweek, 528
 dstflag, 528
 millisecond, 529

 month, 528
 year, 528
 RTCDriver, 529
 RTC Driver, 310
 vmt, 530
 RTCDriverVMT, 530
 RTR
 CANRxFrame, 483
 CANTxFrame, 485

 rca
 SDCDriver, 534
 rdevent
 MACDriver, 512
 rdqueue
 MACDriver, 511
 read_CxD
 MMC over SPI Driver, 236
 receiving
 USBDriver, 557
 recvr1
 MMC over SPI Driver, 233
 recvr3
 MMC over SPI Driver, 234
 requests_hook_cb
 USBConfig, 555
 rtc.c, 632
 rtc.h, 633
 rtc_lld.c, 634
 rtc_lld.h, 635
 rtc_lld_get_alarm
 RTC Driver, 316
 rtc_lld_get_time
 RTC Driver, 315
 rtc_lld_init
 RTC Driver, 314
 rtc_lld_set_alarm
 RTC Driver, 315
 rtc_lld_set_time
 RTC Driver, 315
 rtcConvertDateTimeToFAT
 RTC Driver, 314
 rtcConvertDateTimeToStructTm
 RTC Driver, 313
 rtcConvertStructTmToDateTm
 RTC Driver, 314
 rtcGetAlarm
 RTC Driver, 312
 rtcGetTime
 RTC Driver, 311
 rtcInit
 RTC Driver, 310
 rtcObjectInit
 RTC Driver, 310
 rtcSetAlarm
 RTC Driver, 312
 rtcSetCallback
 RTC Driver, 313
 rtcSetTime

RTC Driver, 311
 rtcalarm_t
 RTC Driver, 310
 rtccb_t
 RTC Driver, 310
 rtcevent_t
 RTC Driver, 310
 rtcnt_t
 OSAL, 217
 rx_buffer
 I2SConfig, 502
 rdbuf
 USBOutEndpointState, 565
 rxchar_cb
 UARTConfig, 548
 rxcnt
 USBOutEndpointState, 565
 rxend_cb
 UARTConfig, 548
 rxerr_cb
 UARTConfig, 548
 rxfull_event
 CANDriver, 481
 rxqueue
 CANDriver, 481
 USBOutEndpointState, 565
 rxqueued
 USBOutEndpointState, 565
 rxsize
 USBOutEndpointState, 565
 rxstate
 UARTDriver, 549

SD1
 Serial Driver, 359
SD_BREAK_DETECTED
 Serial Driver, 349
SD_FRAMING_ERROR
 Serial Driver, 348
SD_NOISE_ERROR
 Serial Driver, 349
SD_OVERRUN_ERROR
 Serial Driver, 349
SD_PARITY_ERROR
 Serial Driver, 348
SD_READY
 Serial Driver, 353
SD_STOP
 Serial Driver, 353
SD_UNINIT
 Serial Driver, 353
SDC Driver, 317
 _sdc_driver_methods, 322
 _sdc_wait_for_transfer_state, 331
 detect_bus_clk, 328
 mmc_cmd6_construct, 325
 mmc_detect_bus_clk, 327
 mmc_init, 323
 mmc_set_bus_width, 329
 mmc_switch_t, 322
 mode_detect, 323
 PLATFORM_SDC_USE_SDC1, 322
 SDC_INIT_RETRY, 320
 SDC_MMC_SUPPORT, 320
 SDC_NICE_WAITING, 321
 SDCD1, 345
 SDCDriver, 322
 sd_switch_function_t, 322
 sd_switch_t, 322
 sdc_cmd6_check_status, 326
 sdc_cmd6_construct, 325
 sdc_cmd6_extract_info, 326
 sdc_detect_bus_clk, 326
 sdc_init, 324
 sdc_lld_init, 340
 sdc_lld_read, 343
 sdc_lld_send_cmd_long_crc, 343
 sdc_lld_send_cmd_none, 342
 sdc_lld_send_cmd_short, 342
 sdc_lld_send_cmd_short_crc, 342
 sdc_lld_set_bus_mode, 341
 sdc_lld_set_data_clk, 341
 sdc_lld_start, 340
 sdc_lld_start_clk, 341
 sdc_lld_stop, 340
 sdc_lld_stop_clk, 341
 sdc_lld_sync, 344
 sdc_lld_write, 344
 sdc_set_bus_width, 329
 sdc_vmt, 345
 sdcConnect, 333
 sdcDisconnect, 334
 sdcErase, 339
 sdcGetAndClearErrors, 337
 sdcGetInfo, 339
 sdclInit, 332
 sdclsCardInserted, 321
 sdclsWriteProtected, 321
 sdcObjectInit, 332
 sdcRead, 336
 sdcStart, 332
 sdcStop, 333
 sdcSync, 338
 sdcWrite, 337
 sdcbusclk_t, 323
 sdcbusmode_t, 323
 sdcflags_t, 322
 sdcmode_t, 322
SDC_INIT_RETRY
 Configuration, 200
 SDC Driver, 320
SDC_MMC_SUPPORT
 Configuration, 200
 SDC Driver, 320
SDC_NICE_WAITING
 Configuration, 200
 SDC Driver, 321

SDDCConfig, 531
bus_width, 532
scratchpad, 532
SDCD1
 SDC Driver, 345
SDCDriver, 532
 cardmode, 534
 config, 533
 errors, 534
 rca, 534
 SDC Driver, 322
 vmt, 533
SDCDriverVMT, 534
SDU_READY
 Serial over USB Driver, 364
SDU_STOP
 Serial over USB Driver, 364
SDU_UNINIT
 Serial over USB Driver, 364
SERIAL_BUFFERS_SIZE
 Configuration, 200
 Serial Driver, 349
SERIAL_DEFAULT_BITRATE
 Configuration, 200
 Serial Driver, 349
SERIAL_USB_BUFFERS_SIZE
 Configuration, 201
 Serial over USB Driver, 364
SID
 CANRxFrame, 484
 CANTxFrame, 485
SPI Driver, 371
 _spi_isr_code, 378
 _spi_wakeup_isr, 378
 PLATFORM_SPI_USE_SPI1, 379
 SPI_ACTIVE, 379
 SPI_COMPLETE, 379
 SPI_READY, 379
 SPI_STOP, 379
 SPI_UNINIT, 379
 SPI_USE_MUTUAL_EXCLUSION, 374
 SPI_USE_WAIT, 374
 SPID1, 394
 SPIDriver, 379
 spi_lld_exchange, 392
 spi_lld_ignore, 392
 spi_lld_init, 391
 spi_lld_polled_exchange, 394
 spi_lld_receive, 393
 spi_lld_select, 392
 spi_lld_send, 393
 spi_lld_start, 391
 spi_lld_stop, 391
 spi_lld_unselect, 392
 spiAcquireBus, 390
 spiExchange, 387
 spilgnore, 386
 spilnIt, 379
 spiObjectInit, 380
 spiPolledExchange, 377
 spiReceive, 389
 spiReleaseBus, 390
 spiSelect, 381
 spiSelectl, 374
 spiSend, 388
 spiStart, 380
 spiStartExchange, 384
 spiStartExchangel, 375
 spiStartIgnore, 383
 spiStartIgnorel, 375
 spiStartReceive, 386
 spiStartReceiveI, 376
 spiStartSend, 385
 spiStartSendl, 376
 spiStop, 381
 spiUnselect, 383
 spiUnselectl, 374
 spicallback_t, 379
 spistate_t, 379
 SPI_ACTIVE
 SPI Driver, 379
 SPI_COMPLETE
 SPI Driver, 379
 SPI_READY
 SPI Driver, 379
 SPI_STOP
 SPI Driver, 379
 SPI_UNINIT
 SPI Driver, 379
 SPI_USE_MUTUAL_EXCLUSION
 Configuration, 201
 SPI Driver, 374
 SPI_USE_WAIT
 Configuration, 201
 SPI Driver, 374
 SPICConfig, 544
 end_cb, 544
SPID1
 SPI Driver, 394
SPIDriver, 545
 config, 545
 mutex, 546
 SPI Driver, 379
 state, 545
 thread, 546
ST Driver, 395
 st_lld_get_alarm, 399
 st_lld_get_counter, 398
 st_lld_init, 398
 st_lld_is_alarm_active, 399
 st_lld_set_alarm, 399
 st_lld_start_alarm, 399
 st_lld_stop_alarm, 399
 stGetAlarm, 398
 stGetCounter, 395
 stlInit, 396

stIsAlarmActive, 396
 stSetAlarm, 397
 stStartAlarm, 396
 stStopAlarm, 397
 samples
 ADCDriver, 465
 DACDriver, 490
 scratchpad
 SDCConfig, 532
 sd_lld_init
 Serial Driver, 358
 sd_lld_start
 Serial Driver, 359
 sd_lld_stop
 Serial Driver, 359
 sd_switch_function_t
 SDC Driver, 322
 sd_switch_t
 SDC Driver, 322
 sdAsynchronousRead
 Serial Driver, 352
 sdAsynchronousWrite
 Serial Driver, 351
 sdGet
 Serial Driver, 350
 sdGetTimeout
 Serial Driver, 350
 sdGetWouldBlock
 Serial Driver, 358
 sdIncomingDataI
 Serial Driver, 355
 sdInit
 Serial Driver, 353
 sdObjectInit
 Serial Driver, 354
 sdPut
 Serial Driver, 349
 sdPutTimeout
 Serial Driver, 349
 sdPutWouldBlock
 Serial Driver, 357
 sdRead
 Serial Driver, 351
 sdReadTimeout
 Serial Driver, 352
 sdRequestDataI
 Serial Driver, 356
 sdStart
 Serial Driver, 354
 sdStop
 Serial Driver, 355
 sdWrite
 Serial Driver, 350
 sdWriteTimeout
 Serial Driver, 351
 sdc.c, 636
 sdc.h, 637
 sdc_cmd6_check_status
 SDC Driver, 326
 sdc_cmd6_construct
 SDC Driver, 325
 sdc_cmd6_extract_info
 SDC Driver, 326
 sdc_detect_bus_clk
 SDC Driver, 326
 sdc_init
 SDC Driver, 324
 sdc_lld.c, 639
 sdc_lld.h, 640
 sdc_lld_init
 SDC Driver, 340
 sdc_lld_read
 SDC Driver, 343
 sdc_lld_send_cmd_long_crc
 SDC Driver, 343
 sdc_lld_send_cmd_none
 SDC Driver, 342
 sdc_lld_send_cmd_short
 SDC Driver, 342
 sdc_lld_send_cmd_short_crc
 SDC Driver, 342
 sdc_lld_set_bus_mode
 SDC Driver, 341
 sdc_lld_set_data_clk
 SDC Driver, 341
 sdc_lld_start
 SDC Driver, 340
 sdc_lld_start_clk
 SDC Driver, 341
 sdc_lld_stop
 SDC Driver, 340
 sdc_lld_stop_clk
 SDC Driver, 341
 sdc_lld_sync
 SDC Driver, 344
 sdc_lld_write
 SDC Driver, 344
 sdc_set_bus_width
 SDC Driver, 329
 sdc_vmt
 SDC Driver, 345
 sdcConnect
 SDC Driver, 333
 sdcDisconnect
 SDC Driver, 334
 sdcErase
 SDC Driver, 339
 sdcGetAndClearErrors
 SDC Driver, 337
 sdcGetInfo
 SDC Driver, 339
 sdclInit
 SDC Driver, 332
 sdclsCardInserted
 SDC Driver, 321
 sdclsWriteProtected

SDC Driver, 321
sdcObjectInit
 SDC Driver, 332
sdcRead
 SDC Driver, 336
sdcStart
 SDC Driver, 332
sdcStop
 SDC Driver, 333
sdcSync
 SDC Driver, 338
sdcWrite
 SDC Driver, 337
sdcbusclk_t
 SDC Driver, 323
sdcbusmode_t
 SDC Driver, 323
sdcfags_t
 SDC Driver, 322
sdemode_t
 SDC Driver, 322
sdstate_t
 Serial Driver, 353
sduConfigureHookl
 Serial over USB Driver, 367
sduDataReceived
 Serial over USB Driver, 369
sduDataTransmitted
 Serial over USB Driver, 369
sduInit
 Serial over USB Driver, 365
sduInterruptTransmitted
 Serial over USB Driver, 369
sduObjectInit
 Serial over USB Driver, 366
sduRequestsHook
 Serial over USB Driver, 368
sduStart
 Serial over USB Driver, 366
sduStop
 Serial over USB Driver, 367
sdustate_t
 Serial over USB Driver, 364
send_command_R1
 MMC over SPI Driver, 235
send_command_R3
 MMC over SPI Driver, 235
send_hdr
 MMC over SPI Driver, 233
Serial Driver, 346
 _serial_driver_data, 352
 _serial_driver_methods, 349
 default_config, 359
 PLATFORM_SERIAL_USE_USART1, 352
 SD1, 359
 SD_BREAK_DETECTED, 349
 SD_FRAMING_ERROR, 348
 SD_NOISE_ERROR, 349
 SD_OVERRUN_ERROR, 349
 SD_PARITY_ERROR, 348
 SD_READY, 353
 SD_STOP, 353
 SD_UNINIT, 353
 SERIAL_BUFFERS_SIZE, 349
 SERIAL_DEFAULT_BITRATE, 349
 sd_lld_init, 358
 sd_lld_start, 359
 sd_lld_stop, 359
 sdAsynchronousRead, 352
 sdAsynchronousWrite, 351
 sdGet, 350
 sdGetTimeout, 350
 sdGetWouldBlock, 358
 sdIncomingData1, 355
 sdInit, 353
 sdObjectInit, 354
 sdPut, 349
 sdPutTimeout, 349
 sdPutWouldBlock, 357
 sdRead, 351
 sdReadTimeout, 352
 sdRequestData1, 356
 sdStart, 354
 sdStop, 355
 sdWrite, 350
 sdWriteTimeout, 351
 sdstate_t, 353
 SerialDriver, 353
 Serial over USB Driver, 361
 _serial_usb_driver_data, 364
 _serial_usb_driver_methods, 364
 inotify, 364
 onotify, 365
 SDU_READY, 364
 SDU_STOP, 364
 SDU_UNINIT, 364
 SERIAL_USB_BUFFERS_SIZE, 364
 sduConfigureHookl, 367
 sduDataReceived, 369
 sduDataTransmitted, 369
 sduInit, 365
 sduInterruptTransmitted, 369
 sduObjectInit, 366
 sduRequestsHook, 368
 sduStart, 366
 sduStop, 367
 sdustate_t, 364
 SerialUSBDriver, 364
 serial.c, 641
 serial.h, 642
 serial_lld.c, 644
 serial_lld.h, 644
 serial_usb.c, 645
 serial_usb.h, 646
 SerialConfig, 535
 speed, 536

SerialDriver, 536
 Serial Driver, 353
 vmt, 537
 SerialDriverVMT, 537
 SerialUSBConfig, 539
 bulk_in, 540
 bulk_out, 540
 int_in, 540
 usbp, 540
 SerialUSBDriver, 540
 Serial over USB Driver, 364
 vmt, 542
 SerialUSBDriverVMT, 542
 set_address
 USB Driver, 437
 setup
 USBDriver, 558
 setup_cb
 USBEndpointConfig, 560
 size
 I2SConfig, 503
 MACReceiveDescriptor, 512
 MACTransmitDescriptor, 513
 sleep_event
 CANDriver, 482
 sof_cb
 USBConfig, 555
 speed
 SerialConfig, 536
 spi.c, 648
 spi.h, 649
 spi_lld.c, 650
 spi_lld.h, 651
 spi_lld_exchange
 SPI Driver, 392
 spi_lld_ignore
 SPI Driver, 392
 spi_lld_init
 SPI Driver, 391
 spi_lld_polled_exchange
 SPI Driver, 394
 spi_lld_receive
 SPI Driver, 393
 spi_lld_select
 SPI Driver, 392
 spi_lld_send
 SPI Driver, 393
 spi_lld_start
 SPI Driver, 391
 spi_lld_stop
 SPI Driver, 391
 spi_lld_unselect
 SPI Driver, 392
 spiAcquireBus
 SPI Driver, 390
 spiExchange
 SPI Driver, 387
 spilgnore
 SPI Driver, 386
 spilInit
 SPI Driver, 379
 spiObjectInit
 SPI Driver, 380
 spiPolledExchange
 SPI Driver, 377
 spiReceive
 SPI Driver, 389
 spiReleaseBus
 SPI Driver, 390
 spiSelect
 SPI Driver, 381
 spiSelectl
 SPI Driver, 374
 spiSend
 SPI Driver, 388
 spiStart
 SPI Driver, 380
 spiStartExchange
 SPI Driver, 384
 spiStartExchangel
 SPI Driver, 375
 spiStartIgnore
 SPI Driver, 383
 spiStartIgnorel
 SPI Driver, 375
 spiStartReceive
 SPI Driver, 386
 spiStartReceive1
 SPI Driver, 376
 spiStartSend
 SPI Driver, 385
 spiStartSendl
 SPI Driver, 376
 spiStop
 SPI Driver, 381
 spiUnselect
 SPI Driver, 383
 spiUnselectl
 SPI Driver, 374
 spicallback_t
 SPI Driver, 379
 spip
 MMCConfig, 514
 spistate_t
 SPI Driver, 379
 st.c, 652
 st.h, 652
 st_lld.c, 653
 st_lld.h, 653
 st_lld_get_alarm
 ST Driver, 399
 st_lld_get_counter
 ST Driver, 398
 st_lld_init
 ST Driver, 398
 st_lld_is_alarm_active

ST Driver, 399
st_lld_set_alarm
 ST Driver, 399
st_lld_start_alarm
 ST Driver, 399
st_lld_stop_alarm
 ST Driver, 399
stGetAlarm
 ST Driver, 398
stGetCounter
 ST Driver, 395
stInit
 ST Driver, 396
stIsAlarmActive
 ST Driver, 396
stSetAlarm
 ST Driver, 397
stStartAlarm
 ST Driver, 396
stStopAlarm
 ST Driver, 397
state
 ADCDriver, 465
 CANDriver, 481
 DACDriver, 490
 EXTDriver, 494
 GPTDriver, 499
 I2CDriver, 501
 I2SDriver, 504
 ICUDriver, 506
 MACDriver, 511
 PWMDriver, 526
 SPIDriver, 545
 UARTDriver, 549
 USBDriver, 557
status
 USBDriver, 558
streamGet
 Abstract Streams, 136
streamPut
 Abstract Streams, 135
streamRead
 Abstract Streams, 135
streamWrite
 Abstract Streams, 135
sync
 MMC over SPI Driver, 237
syssts_t
 OSAL, 216
system_t
 OSAL, 217
TIME
 CANRxFrame, 483
tdqueue
 MACDriver, 511
thread
 ADCDriver, 465
 DACDriver, 490
 SPIDriver, 546
 thread_reference_t
 OSAL, 217
 threads_queue_t, 546
 transmitting
 USBDriver, 557
 tx_buffer
 I2SConfig, 502
 txbuf
 USBInEndpointState, 563
 txcnt
 USBInEndpointState, 563
 txempty_event
 CANDriver, 481
 txend1_cb
 UARTConfig, 547
 txend2_cb
 UARTConfig, 547
 txqueue
 CANDriver, 481
 USBInEndpointState, 563
 txqueued
 USBInEndpointState, 563
 txsize
 USBInEndpointState, 563
 txstate
 UARTDriver, 549
 UART Driver, 401
 PLATFORM_UART_USE_UART1, 405
 UART_BREAK_DETECTED, 405
 UART_FRAMING_ERROR, 405
 UART_NO_ERROR, 404
 UART_NOISE_ERROR, 405
 UART_OVERRUN_ERROR, 405
 UART_PARITY_ERROR, 404
 UART_READY, 406
 UART_RX_ACTIVE, 406
 UART_RX_COMPLETE, 406
 UART_RX_IDLE, 406
 UART_STOP, 406
 UART_TX_ACTIVE, 406
 UART_TX_COMPLETE, 406
 UART_TX_IDLE, 406
 UART_UNINIT, 406
 UARTD1, 416
 UARTDriver, 405
 uart_lld_init, 414
 uart_lld_start, 414
 uart_lld_start_receive, 415
 uart_lld_start_send, 415
 uart_lld_stop, 414
 uart_lld_stop_receive, 416
 uart_lld_stop_send, 415
 uartInit, 406
 uartObjectInit, 407
 uartStart, 407
 uartStartReceive, 411
 uartStartReceive1, 412

uartStartSend, 408
 uartStartSendl, 409
 uartStop, 408
 uartStopReceive, 412
 uartStopReceive1, 413
 uartStopSend, 410
 uartStopSendl, 410
 uartcb_t, 405
 uartccb_t, 405
 uar tecb_t, 406
 uartflags_t, 405
 uartrxstate_t, 406
 uartstate_t, 406
 uarttxstate_t, 406
UART_BREAK_DETECTED
 UART Driver, 405
UART_FRAMING_ERROR
 UART Driver, 405
UART_NO_ERROR
 UART Driver, 404
UART_NOISE_ERROR
 UART Driver, 405
UART_OVERRUN_ERROR
 UART Driver, 405
UART_PARITY_ERROR
 UART Driver, 404
UART_READY
 UART Driver, 406
UART_RX_ACTIVE
 UART Driver, 406
UART_RX_COMPLETE
 UART Driver, 406
UART_RX_IDLE
 UART Driver, 406
UART_STOP
 UART Driver, 406
UART_TX_ACTIVE
 UART Driver, 406
UART_TX_COMPLETE
 UART Driver, 406
UART_TX_IDLE
 UART Driver, 406
UART_UNINIT
 UART Driver, 406
UARTConfig, 546
 rxchar_cb, 548
 rxend_cb, 548
 rxerr_cb, 548
 txend1_cb, 547
 txend2_cb, 547
UARTD1
 UART Driver, 416
UARTDriver, 548
 config, 549
 rxstate, 549
 state, 549
 txstate, 549
 UART Driver, 405

USB Driver, 417
 _usb_ep0in, 450
 _usb_ep0out, 451
 _usb_ep0setup, 449
 _usb_isr_invoke_event_cb, 431
 _usb_isr_invoke_in_cb, 433
 _usb_isr_invoke_out_cb, 433
 _usb_isr_invoke_setup_cb, 433
 _usb_isr_invoke_sof_cb, 431
 _usb_reset, 448
 _usb_suspend, 448
 _usb_wakeup, 449
 default_handler, 438
 EP_STATUS_ACTIVE, 437
 EP_STATUS_DISABLED, 437
 EP_STATUS_STALLED, 437
 ep0_state, 459
 ep0config, 459
 in, 459
 out, 459
 PLATFORM_USB_USE_USB1, 434
 set_address, 437
 USB_ACTIVE, 437
 USB_DESC_BCD, 424
 USB_DESC_BYTE, 424
 USB_DESC_CONFIGURATION, 425
 USB_DESC_CONFIGURATION_SIZE, 425
 USB_DESC_DEVICE, 424
 USB_DESC_ENDPOINT, 426
 USB_DESC_ENDPOINT_SIZE, 426
 USB_DESC_INDEX, 424
 USB_DESC_INTERFACE, 425
 USB_DESC_INTERFACE_ASSOCIATION, 426
 USB_DESC_INTERFACE_ASSOCIATION_SIZE, 426
 USB_DESC_INTERFACE_SIZE, 425
 USB_DESC_WORD, 424
 USB_EP0_ERROR, 437
 USB_EP0_RX, 437
 USB_EP0_SENDING_STS, 437
 USB_EP0_STATUS_STAGE, 434
 USB_EP0_TX, 437
 USB_EP0_WAITING_SETUP, 437
 USB_EP0_WAITING_STS, 437
 USB_EP0_WAITING_TX0, 437
 USB_EP_MODE_LINEAR_BUFFER, 427
 USB_EP_MODE_QUEUE_BUFFER, 427
 USB_EP_MODE_TYPE, 426
 USB_EP_MODE_TYPE_BULK, 427
 USB_EP_MODE_TYPE_CTRL, 426
 USB_EP_MODE_TYPE_INTR, 427
 USB_EP_MODE_TYPE_ISOC, 427
 USB_EVENT_ADDRESS, 437
 USB_EVENT_CONFIGURED, 437
 USB_EVENT_RESET, 437
 USB_EVENT_STALLED, 437
 USB_EVENT_SUSPEND, 437
 USB_EVENT_WAKEUP, 437

USB_MAX_ENDPOINTS, 434
USB_READY, 437
USB_SELECTED, 437
USB_SET_ADDRESS_ACK_HANDLING, 434
USB_SET_ADDRESS_MODE, 434
USB_STOP, 436
USB_SUSPENDED, 437
USB_UNINIT, 436
USBD1, 459
USBDriver, 435
usb_lld_clear_in, 458
usb_lld_clear_out, 458
usb_lld_connect_bus, 435
usb_lld_disable_endpoints, 454
usb_lld_disconnect_bus, 435
usb_lld_get_frame_number, 434
usb_lld_get_status_in, 456
usb_lld_get_status_out, 456
usb_lld_get_transaction_size, 434
usb_lld_init, 452
usb_lld_init_endpoint, 454
usb_lld_prepare_receive, 457
usb_lld_prepare_transmit, 457
usb_lld_read_setup, 456
usb_lld_reset, 454
usb_lld_set_address, 454
usb_lld_stall_in, 458
usb_lld_stall_out, 458
usb_lld_start, 452
usb_lld_start_in, 457
usb_lld_start_out, 457
usb_lld_stop, 452
usbConnectBus, 427
usbDisableEndpointsl, 441
usbDisconnectBus, 427
usbGetDriverStatel, 427
usbGetFrameNumber, 429
usbGetReceiveStatusl, 429
usbGetReceiveTransactionSizeI, 430
usbGetTransmitStatusl, 429
usbInit, 439
usbInitEndpointl, 441
usbObjectInit, 439
usbPrepareQueuedReceive, 443
usbPrepareQueuedTransmit, 444
usbPrepareReceive, 442
usbPrepareTransmit, 442
usbReadSetup, 431
usbSetupTransfer, 430
usbStallReceivel, 446
usbStallTransmitl, 446
usbStart, 440
usbStartReceivel, 444
usbStartTransmitl, 445
usbStop, 440
usbcallback_t, 435
usbep0state_t, 437
usbep_t, 435
usbepcallback_t, 436
usbepstatus_t, 437
usbevent_t, 437
usbeventcb_t, 436
usbgetdescriptor_t, 436
usbreqhandler_t, 436
usbstate_t, 436
USB_ACTIVE
 USB Driver, 437
USB_DESC_BCD
 USB Driver, 424
USB_DESC_BYTEx
 USB Driver, 424
USB_DESC_CONFIGURATION
 USB Driver, 425
USB_DESC_CONFIGURATION_SIZE
 USB Driver, 425
USB_DESC_DEVICE
 USB Driver, 424
USB_DESC_ENDPOINT
 USB Driver, 426
USB_DESC_ENDPOINT_SIZE
 USB Driver, 426
USB_DESC_INDEX
 USB Driver, 424
USB_DESC_INTERFACE
 USB Driver, 425
USB_DESC_INTERFACE_ASSOCIATION
 USB Driver, 426
USB_DESC_INTERFACE_ASSOCIATION_SIZE
 USB Driver, 426
USB_DESC_INTERFACE_SIZE
 USB Driver, 425
USB_DESC_WORD
 USB Driver, 424
USB_EP0_ERROR
 USB Driver, 437
USB_EP0_RX
 USB Driver, 437
USB_EP0_SENDING_STS
 USB Driver, 437
USB_EP0_STATUS_STAGE
 USB Driver, 434
USB_EP0_TX
 USB Driver, 437
USB_EP0_WAITING_SETUP
 USB Driver, 437
USB_EP0_WAITING_STS
 USB Driver, 437
USB_EP0_WAITING_TX0
 USB Driver, 437
USB_EP_MODE_LINEAR_BUFFER
 USB Driver, 427
USB_EP_MODE_QUEUE_BUFFER
 USB Driver, 427
USB_EP_MODE_TYPE
 USB Driver, 426
USB_EP_MODE_TYPE_BULK

USB Driver, 427
USB_EP_MODE_TYPE_CTRL
 USB Driver, 426
USB_EP_MODE_TYPE_INTR
 USB Driver, 427
USB_EP_MODE_TYPE_ISOC
 USB Driver, 427
USB_EVENT_ADDRESS
 USB Driver, 437
USB_EVENT_CONFIGURED
 USB Driver, 437
USB_EVENT_RESET
 USB Driver, 437
USB_EVENT_STALLED
 USB Driver, 437
USB_EVENT_SUSPEND
 USB Driver, 437
USB_EVENT_WAKEUP
 USB Driver, 437
USB_MAX_ENDPOINTS
 USB Driver, 434
USB_READY
 USB Driver, 437
USB_SELECTED
 USB Driver, 437
USB_SET_ADDRESS_ACK_HANDLING
 USB Driver, 434
USB_SET_ADDRESS_MODE
 USB Driver, 434
USB_STOP
 USB Driver, 436
USB_SUSPENDED
 USB Driver, 437
USB_UNINIT
 USB Driver, 436
USBConfig, 553
 event_cb, 555
 get_descriptor_cb, 555
 requests_hook_cb, 555
 sof_cb, 555
USBD1
 USB Driver, 459
USBDescriptor, 555
 ud_size, 556
 ud_string, 556
USBDriver, 556
 address, 558
 config, 557
 configuration, 558
 ep0endcb, 558
 ep0n, 558
 ep0next, 558
 ep0state, 558
 epc, 557
 in_params, 558
 out_params, 558
 receiving, 557
 setup, 558
 state, 557
 status, 558
 transmitting, 557
 USB Driver, 435
USBEndpointConfig, 559
 ep_mode, 560
 in_cb, 560
 in_maxsize, 561
 in_state, 561
 out_cb, 560
 out_maxsize, 561
 out_state, 561
 setup_cb, 560
USBInEndpointState, 561
 txbuf, 563
 txcnt, 563
 txqueue, 563
 txqueued, 563
 txsize, 563
USBOutEndpointState, 563
 rxbuf, 565
 rcnt, 565
 rxqueue, 565
 rxqueued, 565
 rsize, 565
 uart.c, 654
 uart.h, 655
 uart_lld.c, 656
 uart_lld.h, 656
 uart_lld_init
 UART Driver, 414
 uart_lld_start
 UART Driver, 414
 uart_lld_start_receive
 UART Driver, 415
 uart_lld_start_send
 UART Driver, 415
 uart_lld_stop
 UART Driver, 414
 uart_lld_stop_receive
 UART Driver, 416
 uart_lld_stop_send
 UART Driver, 415
 uartInit
 UART Driver, 406
 uartObjectInit
 UART Driver, 407
 uartStart
 UART Driver, 407
 uartStartReceive
 UART Driver, 411
 uartStartReceive1
 UART Driver, 412
 uartStartSend
 UART Driver, 408
 uartStartSend1
 UART Driver, 409
 uartStop

UART Driver, 408
uartStopReceive
 UART Driver, 412
uartStopReceive1
 UART Driver, 413
uartStopSend
 UART Driver, 410
uartStopSend1
 UART Driver, 410
uartcb_t
 UART Driver, 405
uartccb_t
 UART Driver, 405
uartecb_t
 UART Driver, 406
uartflags_t
 UART Driver, 405
uartrxstate_t
 UART Driver, 406
uartstate_t
 UART Driver, 406
uarttxstate_t
 UART Driver, 406
ud_size
 USBDescriptor, 556
ud_string
 USBDescriptor, 556
unpacked_mmc_cid_t, 549
unpacked_mmc_csd_t, 550
unpacked_sdc_cid_t, 551
unpacked_sdc_csd_10_t, 551
unpacked_sdc_csd_20_t, 552
usb.c, 658
usb.h, 659
usb_lld.c, 662
 in, 663
 out, 663
usb_lld.h, 663
usb_lld_clear_in
 USB Driver, 458
usb_lld_clear_out
 USB Driver, 458
usb_lld_connect_bus
 USB Driver, 435
usb_lld_disable_endpoints
 USB Driver, 454
usb_lld_disconnect_bus
 USB Driver, 435
usb_lld_get_frame_number
 USB Driver, 434
usb_lld_get_status_in
 USB Driver, 456
usb_lld_get_status_out
 USB Driver, 456
usb_lld_get_transaction_size
 USB Driver, 434
usb_lld_init
 USB Driver, 452
usb_lld_init_endpoint
 USB Driver, 454
usb_lld_prepare_receive
 USB Driver, 457
usb_lld_prepare_transmit
 USB Driver, 457
usb_lld_read_setup
 USB Driver, 456
usb_lld_reset
 USB Driver, 454
usb_lld_set_address
 USB Driver, 454
usb_lld_stall_in
 USB Driver, 458
usb_lld_stall_out
 USB Driver, 458
usb_lld_start
 USB Driver, 452
usb_lld_start_in
 USB Driver, 457
usb_lld_start_out
 USB Driver, 457
usb_lld_stop
 USB Driver, 452
usbConnectBus
 USB Driver, 427
usbDisableEndpoints
 USB Driver, 441
usbDisconnectBus
 USB Driver, 427
usbGetDriverStateI
 USB Driver, 427
usbGetFrameNumber
 USB Driver, 429
usbGetReceiveStatusI
 USB Driver, 429
usbGetReceiveTransactionSizeI
 USB Driver, 430
usbGetTransmitStatusI
 USB Driver, 429
usbInit
 USB Driver, 439
usbInitEndpointI
 USB Driver, 441
usbObjectInit
 USB Driver, 439
usbPrepareQueuedReceive
 USB Driver, 443
usbPrepareQueuedTransmit
 USB Driver, 444
usbPrepareReceive
 USB Driver, 442
usbPrepareTransmit
 USB Driver, 442
usbReadSetup
 USB Driver, 431
usbSetupTransfer
 USB Driver, 430

usbStallReceive()
 USB Driver, 446
usbStallTransmit()
 USB Driver, 446
usbStart()
 USB Driver, 440
usbStartReceive()
 USB Driver, 444
usbStartTransmit()
 USB Driver, 445
usbStop()
 USB Driver, 440
usbcallback_t
 USB Driver, 435
usbep0state_t
 USB Driver, 437
usbep_t
 USB Driver, 435
usbepcallback_t
 USB Driver, 436
usbepstatus_t
 USB Driver, 437
usbevent_t
 USB Driver, 437
usbeventcb_t
 USB Driver, 436
usbgetdescriptor_t
 USB Driver, 436
usbp
 SerialUSBConfig, 540
usbreqhandler_t
 USB Driver, 436
usbstate_t
 USB Driver, 436

vmt
 BaseAsynchronousChannel, 467
 BaseBlockDevice, 471
 BaseChannel, 474
 BaseSequentialStream, 477
 FileStream, 496
 MMCDriver, 516
 MMCSDBlockDevice, 520
 RTCDriver, 530
 SDCDriver, 533
 SerialDriver, 537
 SerialUSBDriver, 542

wait
 MMC over SPI Driver, 232

wakeup_event
 CANDriver, 482

width_cb
 ICUConfig, 505

year
 RTCDateTime, 528