

ChibiOS/HAL

5.0.0

Reference Manual

Sun Jun 4 2017 15:00:02

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	21
7.1.4.6	_adc_timeout_isr	22

7.1.4.7	<code>_adc_isr_half_code</code>	22
7.1.4.8	<code>_adc_isr_full_code</code>	23
7.1.4.9	<code>_adc_isr_error_code</code>	23
7.1.4.10	<code>PLATFORM_ADC_USE_ADC1</code>	24
7.1.5	Typedef Documentation	24
7.1.5.1	<code>adcsample_t</code>	24
7.1.5.2	<code>adc_channels_num_t</code>	24
7.1.5.3	<code>ADCDriver</code>	24
7.1.5.4	<code>adccallback_t</code>	24
7.1.5.5	<code>adcerrorcallback_t</code>	24
7.1.6	Enumeration Type Documentation	24
7.1.6.1	<code>adcstate_t</code>	25
7.1.6.2	<code>adcerror_t</code>	25
7.1.7	Function Documentation	25
7.1.7.1	<code>adcInit(void)</code>	25
7.1.7.2	<code>adcObjectInit(ADCDriver *adcp)</code>	25
7.1.7.3	<code>adcStart(ADCDriver *adcp, const ADCConfig *config)</code>	26
7.1.7.4	<code>adcStop(ADCDriver *adcp)</code>	26
7.1.7.5	<code>adcStartConversion(ADCDriver *adcp, const ADCConversionGroup *grpp, adcsample_t *samples, size_t depth)</code>	27
7.1.7.6	<code>adcStartConversionI(ADCDriver *adcp, const ADCConversionGroup *grpp, adcsample_t *samples, size_t depth)</code>	27
7.1.7.7	<code>adcStopConversion(ADCDriver *adcp)</code>	28
7.1.7.8	<code>adcStopConversionI(ADCDriver *adcp)</code>	29
7.1.7.9	<code>adcConvert(ADCDriver *adcp, const ADCConversionGroup *grpp, adcsample_t *samples, size_t depth)</code>	29
7.1.7.10	<code>adcAcquireBus(ADCDriver *adcp)</code>	30
7.1.7.11	<code>adcReleaseBus(ADCDriver *adcp)</code>	31
7.1.7.12	<code>adc_lld_init(void)</code>	31
7.1.7.13	<code>adc_lld_start(ADCDriver *adcp)</code>	31
7.1.7.14	<code>adc_lld_stop(ADCDriver *adcp)</code>	31
7.1.7.15	<code>adc_lld_start_conversion(ADCDriver *adcp)</code>	32
7.1.7.16	<code>adc_lld_stop_conversion(ADCDriver *adcp)</code>	32
7.1.8	Variable Documentation	32
7.1.8.1	<code>ADCD1</code>	32
7.2	CAN Driver	33
7.2.1	Detailed Description	33
7.2.2	Driver State Machine	33
7.2.3	Macro Definition Documentation	36
7.2.3.1	<code>CAN_LIMIT_WARNING</code>	36
7.2.3.2	<code>CAN_LIMIT_ERROR</code>	36

7.2.3.3	CAN_BUS_OFF_ERROR	36
7.2.3.4	CAN_FRAMING_ERROR	36
7.2.3.5	CAN_OVERFLOW_ERROR	36
7.2.3.6	CAN_ANY_MAILBOX	36
7.2.3.7	CAN_USE_SLEEP_MODE	36
7.2.3.8	CAN_MAILBOX_TO_MASK	36
7.2.3.9	canTransmit	36
7.2.3.10	canReceive	36
7.2.3.11	CAN_TX_MAILBOXES	37
7.2.3.12	CAN_RX_MAILBOXES	37
7.2.3.13	PLATFORM_CAN_USE_CAN1	37
7.2.4	Typedef Documentation	37
7.2.4.1	canmbx_t	37
7.2.5	Enumeration Type Documentation	37
7.2.5.1	canstate_t	37
7.2.6	Function Documentation	37
7.2.6.1	canInit(void)	37
7.2.6.2	canObjectInit(CANDriver *canp)	38
7.2.6.3	canStart(CANDriver *canp, const CANConfig *config)	38
7.2.6.4	canStop(CANDriver *canp)	39
7.2.6.5	canTryTransmit(CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp)	39
7.2.6.6	canTryReceive(CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp)	40
7.2.6.7	canTransmitTimeout(CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp, systime_t timeout)	41
7.2.6.8	canReceiveTimeout(CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp, systime_t timeout)	41
7.2.6.9	canSleep(CANDriver *canp)	42
7.2.6.10	canWakeup(CANDriver *canp)	43
7.2.6.11	can_lld_init(void)	43
7.2.6.12	can_lld_start(CANDriver *canp)	44
7.2.6.13	can_lld_stop(CANDriver *canp)	44
7.2.6.14	can_lld_is_tx_empty(CANDriver *canp, canmbx_t mailbox)	44
7.2.6.15	can_lld_transmit(CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp)	45
7.2.6.16	can_lld_is_rx_nonempty(CANDriver *canp, canmbx_t mailbox)	45
7.2.6.17	can_lld_receive(CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp)	46
7.2.6.18	can_lld_sleep(CANDriver *canp)	46
7.2.6.19	can_lld_wakeup(CANDriver *canp)	46
7.2.7	Variable Documentation	46
7.2.7.1	CAND1	46
7.3	DAC Driver	47

7.3.1	Detailed Description	47
7.3.2	Macro Definition Documentation	49
7.3.2.1	DAC_USE_WAIT	49
7.3.2.2	DAC_USE_MUTUAL_EXCLUSION	49
7.3.2.3	_dac_wait_s	49
7.3.2.4	_dac_reset_i	50
7.3.2.5	_dac_reset_s	50
7.3.2.6	_dac_wakeup_isr	50
7.3.2.7	_dac_timeout_isr	51
7.3.2.8	_dac_isr_half_code	51
7.3.2.9	_dac_isr_full_code	51
7.3.2.10	_dac_isr_error_code	52
7.3.2.11	DAC_MAX_CHANNELS	53
7.3.2.12	PLATFORM_DAC_USE_DAC1	53
7.3.3	Typedef Documentation	53
7.3.3.1	dacchannel_t	53
7.3.3.2	DACDriver	53
7.3.3.3	dacsample_t	53
7.3.3.4	daccallback_t	53
7.3.3.5	dacerrorcallback_t	53
7.3.4	Enumeration Type Documentation	54
7.3.4.1	dacstate_t	54
7.3.4.2	dacerror_t	54
7.3.5	Function Documentation	54
7.3.5.1	dacInit(void)	54
7.3.5.2	dacObjectInit(DACDriver *dACP)	55
7.3.5.3	dacStart(DACDriver *dACP, const DACConfig *config)	55
7.3.5.4	dacStop(DACDriver *dACP)	56
7.3.5.5	dacPutChannelX(DACDriver *dACP, dacchannel_t channel, dacsample_t sample)	56
7.3.5.6	dacStartConversion(DACDriver *dACP, const DACConversionGroup *grpp, dacsample_t *samples, size_t depth)	57
7.3.5.7	dacStartConversionI(DACDriver *dACP, const DACConversionGroup *grpp, dacsample_t *samples, size_t depth)	57
7.3.5.8	dacStopConversion(DACDriver *dACP)	58
7.3.5.9	dacStopConversionI(DACDriver *dACP)	58
7.3.5.10	dacConvert(DACDriver *dACP, const DACConversionGroup *grpp, dacsample_t *samples, size_t depth)	59
7.3.5.11	dacAcquireBus(DACDriver *dACP)	60
7.3.5.12	dacReleaseBus(DACDriver *dACP)	60
7.3.5.13	dac_Ild_init(void)	60
7.3.5.14	dac_Ild_start(DACDriver *dACP)	61

7.3.5.15	<code>dac_ild_stop(DACDriver *dACP)</code>	61
7.3.5.16	<code>dac_ild_put_channel(DACDriver *dACP, dacchannel_t channel, dacsample_t sample)</code>	61
7.3.5.17	<code>dac_ild_start_conversion(DACDriver *dACP)</code>	62
7.3.5.18	<code>dac_ild_stop_conversion(DACDriver *dACP)</code>	62
7.3.6	Variable Documentation	62
7.3.6.1	DACD1	62
7.4	EXT Driver	63
7.4.1	Detailed Description	63
7.4.2	Driver State Machine	63
7.4.3	EXT Operations	63
7.4.4	Macro Definition Documentation	65
7.4.4.1	EXT_CH_MODE_EDGES_MASK	65
7.4.4.2	EXT_CH_MODE_DISABLED	65
7.4.4.3	EXT_CH_MODE_RISING_EDGE	65
7.4.4.4	EXT_CH_MODE_FALLING_EDGE	65
7.4.4.5	EXT_CH_MODE_BOTH_EDGES	66
7.4.4.6	EXT_CH_MODE_LOW_LEVEL	66
7.4.4.7	EXT_CH_MODE_AUTOSTART	66
7.4.4.8	<code>extChannelEnable</code>	66
7.4.4.9	<code>extChannelDisable</code>	66
7.4.4.10	<code>extSetChannelMode</code>	66
7.4.4.11	EXT_MAX_CHANNELS	67
7.4.4.12	PLATFORM_EXT_USE_EXT1	67
7.4.5	Typedef Documentation	67
7.4.5.1	EXTDriver	67
7.4.5.2	expchannel_t	67
7.4.5.3	extcallback_t	67
7.4.6	Enumeration Type Documentation	67
7.4.6.1	extstate_t	68
7.4.7	Function Documentation	68
7.4.7.1	<code>extInit(void)</code>	68
7.4.7.2	<code>extObjectInit(EXTDriver *extp)</code>	68
7.4.7.3	<code>extStart(EXTDriver *extp, const EXTConfig *config)</code>	68
7.4.7.4	<code>extStop(EXTDriver *extp)</code>	69
7.4.7.5	<code>extChannelEnable(EXTDriver *extp, expchannel_t channel)</code>	69
7.4.7.6	<code>extChannelDisable(EXTDriver *extp, expchannel_t channel)</code>	70
7.4.7.7	<code>extSetChannelModel(EXTDriver *extp, expchannel_t channel, const EXTChannelConfig *extcp)</code>	70
7.4.7.8	<code>ext_ild_init(void)</code>	71

7.4.7.9	<code>ext_lld_start(EXTDriver *extp)</code>	71
7.4.7.10	<code>ext_lld_stop(EXTDriver *extp)</code>	71
7.4.7.11	<code>ext_lld_channel_enable(EXTDriver *extp, expchannel_t channel)</code>	72
7.4.7.12	<code>ext_lld_channel_disable(EXTDriver *extp, expchannel_t channel)</code>	72
7.4.8	Variable Documentation	72
7.4.8.1	EXTD1	72
7.5	Abstract NOR Flash Class	73
7.5.1	Driver State Machine	73
7.5.2	Flash Operations.	73
7.6	GPT Driver	75
7.6.1	Detailed Description	75
7.6.2	Driver State Machine	75
7.6.3	GPT Operations.	75
7.6.4	Macro Definition Documentation	78
7.6.4.1	<code>gptChangeIntervall</code>	78
7.6.4.2	<code>gptGetIntervalX</code>	78
7.6.4.3	<code>gptGetCounterX</code>	78
7.6.4.4	PLATFORM_GPT_USE_GPT1	79
7.6.4.5	<code>gpt_lld_change_interval</code>	79
7.6.5	Typedef Documentation	80
7.6.5.1	GPTDriver	80
7.6.5.2	<code>gptcallback_t</code>	80
7.6.5.3	<code>gptfreq_t</code>	80
7.6.5.4	<code>gptcnt_t</code>	80
7.6.6	Enumeration Type Documentation	80
7.6.6.1	<code>gptstate_t</code>	80
7.6.7	Function Documentation	80
7.6.7.1	<code>gptInit(void)</code>	80
7.6.7.2	<code>gptObjectInit(GPTDriver *gptp)</code>	81
7.6.7.3	<code>gptStart(GPTDriver *gptp, const GPTConfig *config)</code>	81
7.6.7.4	<code>gptStop(GPTDriver *gptp)</code>	82
7.6.7.5	<code>gptChangeInterval(GPTDriver *gptp, gptcnt_t interval)</code>	82
7.6.7.6	<code>gptStartContinuous(GPTDriver *gptp, gptcnt_t interval)</code>	83
7.6.7.7	<code>gptStartContinuousI(GPTDriver *gptp, gptcnt_t interval)</code>	83
7.6.7.8	<code>gptStartOneShot(GPTDriver *gptp, gptcnt_t interval)</code>	84
7.6.7.9	<code>gptStartOneShotI(GPTDriver *gptp, gptcnt_t interval)</code>	84
7.6.7.10	<code>gptStopTimer(GPTDriver *gptp)</code>	84
7.6.7.11	<code>gptStopTimerI(GPTDriver *gptp)</code>	85
7.6.7.12	<code>gptPolledDelay(GPTDriver *gptp, gptcnt_t interval)</code>	85
7.6.7.13	<code>gpt_lld_init(void)</code>	86

7.6.7.14	gpt_lld_start(GPTDriver *gptp)	86
7.6.7.15	gpt_lld_stop(GPTDriver *gptp)	87
7.6.7.16	gpt_lld_start_timer(GPTDriver *gptp, gptcnt_t interval)	87
7.6.7.17	gpt_lld_stop_timer(GPTDriver *gptp)	87
7.6.7.18	gpt_lld_polled_delay(GPTDriver *gptp, gptcnt_t interval)	87
7.6.8	Variable Documentation	88
7.6.8.1	GPTD1	88
7.7	HAL Driver	89
7.7.1	Detailed Description	89
7.7.2	Macro Definition Documentation	90
7.7.2.1	_CHIBIOS_HAL_	90
7.7.2.2	CH_HAL_STABLE	90
7.7.2.3	HAL_VERSION	90
7.7.2.4	CH_HAL_MAJOR	90
7.7.2.5	CH_HAL_MINOR	90
7.7.2.6	CH_HAL_PATCH	90
7.7.3	Function Documentation	90
7.7.3.1	hallnit(void)	90
7.7.3.2	hal_lld_init(void)	91
7.8	I/O Buffers Queues	92
7.8.1	Detailed Description	92
7.8.2	Macro Definition Documentation	94
7.8.2.1	BQ_BUFFER_SIZE	94
7.8.2.2	bqSizeX	94
7.8.2.3	bqSpaceI	94
7.8.2.4	bqGetLinkX	94
7.8.2.5	bqIsSuspendedX	95
7.8.2.6	bqSuspendI	95
7.8.2.7	bqResumeX	96
7.8.2.8	ibqIsEmptyI	96
7.8.2.9	ibqIsFullI	97
7.8.2.10	obqIsEmptyI	97
7.8.2.11	obqIsFullI	98
7.8.3	Typedef Documentation	98
7.8.3.1	io_buffers_queue_t	98
7.8.3.2	bqnotify_t	98
7.8.3.3	input_buffers_queue_t	98
7.8.3.4	output_buffers_queue_t	98
7.8.4	Function Documentation	99

7.8.4.1	ibqObjectInit(input_buffers_queue_t *ibqp, bool suspended, uint8_t *bp, size_t size, size_t n, bqnotify_t infy, void *link)	99
7.8.4.2	ibqResetI(input_buffers_queue_t *ibqp)	99
7.8.4.3	ibqGetEmptyBufferI(input_buffers_queue_t *ibqp)	99
7.8.4.4	ibqPostFullBufferI(input_buffers_queue_t *ibqp, size_t size)	100
7.8.4.5	ibqGetFullBufferTimeout(input_buffers_queue_t *ibqp, systime_t timeout)	100
7.8.4.6	ibqGetFullBufferTimeoutS(input_buffers_queue_t *ibqp, systime_t timeout)	101
7.8.4.7	ibqReleaseEmptyBuffer(input_buffers_queue_t *ibqp)	102
7.8.4.8	ibqReleaseEmptyBufferS(input_buffers_queue_t *ibqp)	102
7.8.4.9	ibqGetTimeout(input_buffers_queue_t *ibqp, systime_t timeout)	103
7.8.4.10	ibqReadTimeout(input_buffers_queue_t *ibqp, uint8_t *bp, size_t n, systime_t timeout)	103
7.8.4.11	obqObjectInit(output_buffers_queue_t *obqp, bool suspended, uint8_t *bp, size_t size, size_t n, bqnotify_t onfy, void *link)	104
7.8.4.12	obqResetI(output_buffers_queue_t *obqp)	105
7.8.4.13	obqGetFullBufferI(output_buffers_queue_t *obqp, size_t *sizep)	105
7.8.4.14	obqReleaseEmptyBufferI(output_buffers_queue_t *obqp)	106
7.8.4.15	obqGetEmptyBufferTimeout(output_buffers_queue_t *obqp, systime_t timeout)	106
7.8.4.16	obqGetEmptyBufferTimeoutS(output_buffers_queue_t *obqp, systime_t timeout)	107
7.8.4.17	obqPostFullBuffer(output_buffers_queue_t *obqp, size_t size)	107
7.8.4.18	obqPostFullBufferS(output_buffers_queue_t *obqp, size_t size)	108
7.8.4.19	obqPutTimeout(output_buffers_queue_t *obqp, uint8_t b, systime_t timeout)	108
7.8.4.20	obqWriteTimeout(output_buffers_queue_t *obqp, const uint8_t *bp, size_t n, systime_t timeout)	109
7.8.4.21	obqTryFlushI(output_buffers_queue_t *obqp)	110
7.8.4.22	obqFlush(output_buffers_queue_t *obqp)	111
7.9	Abstract I/O Channel	112
7.9.1	Detailed Description	112
7.9.2	Macro Definition Documentation	113
7.9.2.1	_base_channel_methods	113
7.9.2.2	_base_channel_data	113
7.9.2.3	chnPutTimeout	113
7.9.2.4	chnGetTimeout	114
7.9.2.5	chnWrite	115
7.9.2.6	chnWriteTimeout	115
7.9.2.7	chnRead	115
7.9.2.8	chnReadTimeout	116
7.9.2.9	CHN_NO_ERROR	116
7.9.2.10	CHN_CONNECTED	116
7.9.2.11	CHN_DISCONNECTED	116
7.9.2.12	CHN_INPUT_AVAILABLE	116

7.9.2.13	CHN_OUTPUT_EMPTY	117
7.9.2.14	CHN_TRANSMISSION_END	117
7.9.2.15	_base_asynchronous_channel_methods	117
7.9.2.16	_base_asynchronous_channel_data	117
7.9.2.17	chnGetEventSource	117
7.9.2.18	chnAddFlagsI	117
7.10	Abstract Files	119
7.10.1	Detailed Description	119
7.10.2	Macro Definition Documentation	120
7.10.2.1	FILE_OK	120
7.10.2.2	FILE_ERROR	120
7.10.2.3	FILE_EOF	120
7.10.2.4	_file_stream_methods	120
7.10.2.5	_file_stream_data	120
7.10.2.6	fileStreamWrite	120
7.10.2.7	fileStreamRead	121
7.10.2.8	fileStreamPut	121
7.10.2.9	fileStreamGet	122
7.10.2.10	fileStreamClose	122
7.10.2.11	fileStreamGetError	123
7.10.2.12	fileStreamGetSize	123
7.10.2.13	fileStreamGetPosition	124
7.10.2.14	fileStreamSeek	124
7.10.3	Typedef Documentation	124
7.10.3.1	fileoffset_t	124
7.11	Abstract I/O Block Device	125
7.11.1	Detailed Description	125
7.11.2	Driver State Machine	125
7.11.3	Macro Definition Documentation	126
7.11.3.1	_base_block_device_methods	126
7.11.3.2	_base_block_device_data	126
7.11.3.3	blkGetDriverState	127
7.11.3.4	blkIsTransferring	127
7.11.3.5	blkIsInserted	128
7.11.3.6	blkIsWriteProtected	128
7.11.3.7	blkConnect	128
7.11.3.8	blkDisconnect	129
7.11.3.9	blkRead	129
7.11.3.10	blkWrite	130
7.11.3.11	blkSync	130

7.11.3.12 blkGetInfo	131
7.11.4 Enumeration Type Documentation	131
7.11.4.1 blkstate_t	131
7.12 I/O Bytes Queues	132
7.12.1 Detailed Description	132
7.12.2 Macro Definition Documentation	133
7.12.2.1 Q_OK	133
7.12.2.2 Q_TIMEOUT	134
7.12.2.3 Q_RESET	134
7.12.2.4 Q_EMPTY	134
7.12.2.5 Q_FULL	134
7.12.2.6 qSizeX	134
7.12.2.7 qSpacel	134
7.12.2.8 qGetLink	135
7.12.2.9 iqGetFullI	135
7.12.2.10 iqGetEmptyI	136
7.12.2.11 iqIsEmptyI	136
7.12.2.12 iqIsFullI	136
7.12.2.13 iqGet	138
7.12.2.14 oqGetFullI	138
7.12.2.15 oqGetEmptyI	139
7.12.2.16 oqIsEmptyI	139
7.12.2.17 oqIsFullI	140
7.12.2.18 oqPut	140
7.12.3 Typedef Documentation	141
7.12.3.1 io_queue_t	141
7.12.3.2 qnotify_t	141
7.12.3.3 input_queue_t	141
7.12.3.4 output_queue_t	141
7.12.4 Function Documentation	141
7.12.4.1 iqObjectInit(input_queue_t *iqp, uint8_t *bp, size_t size, qnotify_t infy, void *link)	141
7.12.4.2 iqResetI(input_queue_t *iqp)	142
7.12.4.3 iqPutI(input_queue_t *iqp, uint8_t b)	142
7.12.4.4 iqGetTimeout(input_queue_t *iqp, systime_t timeout)	143
7.12.4.5 iqReadTimeout(input_queue_t *iqp, uint8_t *bp, size_t n, systime_t timeout)	143
7.12.4.6 oqObjectInit(output_queue_t *oqp, uint8_t *bp, size_t size, qnotify_t onfy, void *link)	144
7.12.4.7 oqResetI(output_queue_t *oqp)	144
7.12.4.8 oqPutTimeout(output_queue_t *oqp, uint8_t b, systime_t timeout)	145
7.12.4.9 oqGetI(output_queue_t *oqp)	145

7.12.4.10 oqWriteTimeout(output_queue_t *oqp, const uint8_t *bp, size_t n, systime_t timeout)	146
7.13 Abstract Streams	147
7.13.1 Detailed Description	147
7.13.2 Macro Definition Documentation	147
7.13.2.1 _base_sequential_stream_methods	147
7.13.2.2 _base_sequential_stream_data	148
7.13.2.3 streamWrite	148
7.13.2.4 streamRead	148
7.13.2.5 streamPut	148
7.13.2.6 streamGet	149
7.14 I2C Driver	150
7.14.1 Detailed Description	150
7.14.2 Driver State Machine	150
7.14.3 Macro Definition Documentation	152
7.14.3.1 I2C_NO_ERROR	152
7.14.3.2 I2C_BUS_ERROR	152
7.14.3.3 I2C_ARBITRATION_LOST	153
7.14.3.4 I2C_ACK_FAILURE	153
7.14.3.5 I2C_OVERRUN	153
7.14.3.6 I2C_PEC_ERROR	153
7.14.3.7 I2C_TIMEOUT	153
7.14.3.8 I2C_SMB_ALERT	153
7.14.3.9 I2C_USE_MUTUAL_EXCLUSION	153
7.14.3.10 _i2c_wakeup_isr	153
7.14.3.11 _i2c_wakeup_error_isr	153
7.14.3.12 i2cMasterTransmit	154
7.14.3.13 i2cMasterReceive	154
7.14.3.14 PLATFORM_I2C_USE_I2C1	154
7.14.3.15 i2c_lld_get_errors	154
7.14.4 Typedef Documentation	155
7.14.4.1 i2caddr_t	155
7.14.4.2 i2cflags_t	155
7.14.4.3 I2CDriver	155
7.14.5 Enumeration Type Documentation	155
7.14.5.1 i2cstate_t	155
7.14.6 Function Documentation	155
7.14.6.1 i2cInit(void)	155
7.14.6.2 i2cObjectInit(I2CDriver *i2cp)	156
7.14.6.3 i2cStart(I2CDriver *i2cp, const I2CConfig *config)	156

7.14.6.4	i2cStop(I2CDriver *i2cp)	156
7.14.6.5	i2cGetErrors(I2CDriver *i2cp)	157
7.14.6.6	i2cMasterTransmitTimeout(I2CDriver *i2cp, i2caddr_t addr, const uint8_t *txbuf, size_t txbytes, uint8_t *rxbuf, size_t rxbytes, systime_t timeout)	157
7.14.6.7	i2cMasterReceiveTimeout(I2CDriver *i2cp, i2caddr_t addr, uint8_t *rxbuf, size_t rxbytes, systime_t timeout)	158
7.14.6.8	i2cAcquireBus(I2CDriver *i2cp)	159
7.14.6.9	i2cReleaseBus(I2CDriver *i2cp)	159
7.14.6.10	i2c_lld_init(void)	159
7.14.6.11	i2c_lld_start(I2CDriver *i2cp)	160
7.14.6.12	i2c_lld_stop(I2CDriver *i2cp)	160
7.14.6.13	i2c_lld_master_receive_timeout(I2CDriver *i2cp, i2caddr_t addr, uint8_t *rxbuf, size_t rxbytes, systime_t timeout)	160
7.14.6.14	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)	161
7.14.7	Variable Documentation	162
7.14.7.1	I2CD1	162
7.15	I2S Driver	163
7.15.1	Detailed Description	163
7.15.2	Driver State Machine	163
7.15.3	Macro Definition Documentation	164
7.15.3.1	i2sStartExchangeI	164
7.15.3.2	i2sStopExchangeI	165
7.15.3.3	_i2s_isr_half_code	165
7.15.3.4	_i2s_isr_full_code	165
7.15.3.5	PLATFORM_I2S_USE_I2S1	166
7.15.4	Typedef Documentation	166
7.15.4.1	I2SDriver	166
7.15.4.2	i2scallback_t	166
7.15.5	Enumeration Type Documentation	167
7.15.5.1	i2sstate_t	167
7.15.6	Function Documentation	167
7.15.6.1	i2sInit(void)	167
7.15.6.2	i2sObjectInit(I2SDriver *i2sp)	167
7.15.6.3	i2sStart(I2SDriver *i2sp, const I2SConfig *config)	168
7.15.6.4	i2sStop(I2SDriver *i2sp)	168
7.15.6.5	i2sStartExchange(I2SDriver *i2sp)	168
7.15.6.6	i2sStopExchange(I2SDriver *i2sp)	169
7.15.6.7	i2s_lld_init(void)	169
7.15.6.8	i2s_lld_start(I2SDriver *i2sp)	169
7.15.6.9	i2s_lld_stop(I2SDriver *i2sp)	170

7.15.6.10	i2s_lld_start_exchange(I2SDriver *i2sp)	170
7.15.6.11	i2s_lld_stop_exchange(I2SDriver *i2sp)	170
7.15.7	Variable Documentation	170
7.15.7.1	I2SD1	170
7.16	ICU Driver	171
7.16.1	Detailed Description	171
7.16.2	Driver State Machine	171
7.16.3	ICU Operations.	171
7.16.4	Macro Definition Documentation	174
7.16.4.1	icuStartCaptureI	174
7.16.4.2	icuStopCaptureI	174
7.16.4.3	icuEnableNotificationsI	175
7.16.4.4	icuDisableNotificationsI	175
7.16.4.5	icuAreNotificationsEnabledX	175
7.16.4.6	icuGetWidthX	176
7.16.4.7	icuGetPeriodX	176
7.16.4.8	_icu_isr_invoke_width_cb	177
7.16.4.9	_icu_isr_invoke_period_cb	177
7.16.4.10	_icu_isr_invoke_overflow_cb	177
7.16.4.11	PLATFORM_ICU_USE_ICU1	178
7.16.4.12	icu_lld_get_width	178
7.16.4.13	icu_lld_get_period	178
7.16.4.14	icu_lld_are_notifications_enabled	179
7.16.5	Typedef Documentation	179
7.16.5.1	ICUDriver	179
7.16.5.2	icucallback_t	179
7.16.5.3	icufreq_t	179
7.16.5.4	icucnt_t	179
7.16.6	Enumeration Type Documentation	180
7.16.6.1	icustate_t	180
7.16.6.2	icumode_t	180
7.16.7	Function Documentation	180
7.16.7.1	icuInit(void)	180
7.16.7.2	icuObjectInit(ICUDriver *icup)	180
7.16.7.3	icuStart(ICUDriver *icup, const ICUConfig *config)	181
7.16.7.4	icuStop(ICUDriver *icup)	181
7.16.7.5	icuStartCapture(ICUDriver *icup)	182
7.16.7.6	icuWaitCapture(ICUDriver *icup)	182
7.16.7.7	icuStopCapture(ICUDriver *icup)	183
7.16.7.8	icuEnableNotifications(ICUDriver *icup)	183

7.16.7.9	icuDisableNotifications(ICUDriver *icup)	184
7.16.7.10	icu_lld_init(void)	184
7.16.7.11	icu_lld_start(ICUDriver *icup)	184
7.16.7.12	icu_lld_stop(ICUDriver *icup)	185
7.16.7.13	icu_lld_start_capture(ICUDriver *icup)	185
7.16.7.14	icu_lld_wait_capture(ICUDriver *icup)	185
7.16.7.15	icu_lld_stop_capture(ICUDriver *icup)	186
7.16.7.16	icu_lld_enable_notifications(ICUDriver *icup)	186
7.16.7.17	icu_lld_disable_notifications(ICUDriver *icup)	186
7.16.8	Variable Documentation	187
7.16.8.1	ICUD1	187
7.17	MAC Driver	188
7.17.1	Detailed Description	188
7.17.2	Macro Definition Documentation	190
7.17.2.1	MAC_USE_ZERO_COPY	190
7.17.2.2	MAC_USE_EVENTS	190
7.17.2.3	macGetReceiveEventSource	190
7.17.2.4	macWriteTransmitDescriptor	190
7.17.2.5	macReadReceiveDescriptor	191
7.17.2.6	macGetNextTransmitBuffer	191
7.17.2.7	macGetNextReceiveBuffer	192
7.17.2.8	MAC_SUPPORTS_ZERO_COPY	192
7.17.2.9	PLATFORM_MAC_USE_MAC1	192
7.17.3	Typedef Documentation	192
7.17.3.1	MACDriver	192
7.17.4	Enumeration Type Documentation	192
7.17.4.1	macstate_t	192
7.17.5	Function Documentation	193
7.17.5.1	macInit(void)	193
7.17.5.2	macObjectInit(MACDriver *macp)	193
7.17.5.3	macStart(MACDriver *macp, const MACConfig *config)	193
7.17.5.4	macStop(MACDriver *macp)	194
7.17.5.5	macWaitTransmitDescriptor(MACDriver *macp, MACTransmitDescriptor *tdp, system_t timeout)	194
7.17.5.6	macReleaseTransmitDescriptor(MACTransmitDescriptor *tdp)	195
7.17.5.7	macWaitReceiveDescriptor(MACDriver *macp, MACReceiveDescriptor *rdp, system_t timeout)	195
7.17.5.8	macReleaseReceiveDescriptor(MACReceiveDescriptor *rdp)	196
7.17.5.9	macPollLinkStatus(MACDriver *macp)	197
7.17.5.10	mac_lld_init(void)	197

7.17.5.11	mac_lld_start(MACDriver *macp)	198
7.17.5.12	mac_lld_stop(MACDriver *macp)	198
7.17.5.13	mac_lld_get_transmit_descriptor(MACDriver *macp, MACTransmitDescriptor *tdp)	198
7.17.5.14	mac_lld_release_transmit_descriptor(MACTransmitDescriptor *tdp)	199
7.17.5.15	mac_lld_get_receive_descriptor(MACDriver *macp, MACReceiveDescriptor *rdp)	199
7.17.5.16	mac_lld_release_receive_descriptor(MACReceiveDescriptor *rdp)	199
7.17.5.17	mac_lld_poll_link_status(MACDriver *macp)	200
7.17.5.18	mac_lld_write_transmit_descriptor(MACTransmitDescriptor *tdp, uint8_t *buf, size_t size)	200
7.17.5.19	mac_lld_read_receive_descriptor(MACReceiveDescriptor *rdp, uint8_t *buf, size_t size)	200
7.17.5.20	mac_lld_get_next_transmit_buffer(MACTransmitDescriptor *tdp, size_t size, size_t *sizep)	201
7.17.5.21	mac_lld_get_next_receive_buffer(MACReceiveDescriptor *rdp, size_t *sizep)	201
7.17.6	Variable Documentation	202
7.17.6.1	ETHD1	202
7.18	HAL	203
7.18.1	Detailed Description	203
7.18.2	HAL Device Drivers Architecture	203
7.18.3	HAL Normal Device Drivers	203
7.18.3.1	Diagram	204
7.18.4	HAL Complex Device Drivers	204
7.18.5	HAL Interfaces	204
7.18.6	HAL Inner Code	204
7.19	Configuration	206
7.19.1	Detailed Description	206
7.19.2	Macro Definition Documentation	208
7.19.2.1	HAL_USE_PAL	208
7.19.2.2	HAL_USE_ADC	208
7.19.2.3	HAL_USE_CAN	208
7.19.2.4	HAL_USE_DAC	208
7.19.2.5	HAL_USE_EXT	208
7.19.2.6	HAL_USE_GPT	209
7.19.2.7	HAL_USE_I2C	209
7.19.2.8	HAL_USE_I2S	209
7.19.2.9	HAL_USE_ICU	209
7.19.2.10	HAL_USE_MAC	209
7.19.2.11	HAL_USE_MMC_SPI	209
7.19.2.12	HAL_USE_PWM	209
7.19.2.13	HAL_USE_QSPI	209
7.19.2.14	HAL_USE_RTC	209

7.19.2.15 HAL_USE_SDC	209
7.19.2.16 HAL_USE_SERIAL	209
7.19.2.17 HAL_USE_SERIAL_USB	209
7.19.2.18 HAL_USE_SPI	210
7.19.2.19 HAL_USE_UART	210
7.19.2.20 HAL_USE_USB	210
7.19.2.21 HAL_USE_WDG	210
7.19.2.22 ADC_USE_WAIT	210
7.19.2.23 ADC_USE_MUTUAL_EXCLUSION	210
7.19.2.24 CAN_USE_SLEEP_MODE	210
7.19.2.25 I2C_USE_MUTUAL_EXCLUSION	210
7.19.2.26 MAC_USE_ZERO_COPY	210
7.19.2.27 MAC_USE_EVENTS	210
7.19.2.28 MMC_NICE_WAITING	211
7.19.2.29 SDC_INIT_RETRY	211
7.19.2.30 SDC_MMC_SUPPORT	211
7.19.2.31 SDC_NICE_WAITING	211
7.19.2.32 SERIAL_DEFAULT_BITRATE	211
7.19.2.33 SERIAL_BUFFERS_SIZE	211
7.19.2.34 SERIAL_USB_BUFFERS_SIZE	211
7.19.2.35 SERIAL_USB_BUFFERS_NUMBER	212
7.19.2.36 SPI_USE_WAIT	212
7.19.2.37 SPI_USE_MUTUAL_EXCLUSION	212
7.19.2.38 UART_USE_WAIT	212
7.19.2.39 UART_USE_MUTUAL_EXCLUSION	212
7.19.2.40 USB_USE_WAIT	212
7.20 Normal Drivers	213
7.20.1 Detailed Description	213
7.21 Complex Drivers	214
7.21.1 Detailed Description	214
7.22 Interfaces	215
7.22.1 Detailed Description	215
7.23 Inner Code	216
7.23.1 Detailed Description	216
7.24 Support Code	217
7.24.1 Detailed Description	217
7.25 OSAL	218
7.26 MII/RMII Header	220
7.26.1 Detailed Description	220
7.26.2 Macro Definition Documentation	222

7.26.2.1	MII_BMCR	222
7.26.2.2	MII_BMSR	222
7.26.2.3	MII_PHYSID1	222
7.26.2.4	MII_PHYSID2	222
7.26.2.5	MII_ADVERTISE	222
7.26.2.6	MII_LPA	222
7.26.2.7	MII_EXPANSION	223
7.26.2.8	MII_ANNPTR	223
7.26.2.9	MII_CTRL1000	223
7.26.2.10	MII_STAT1000	223
7.26.2.11	MII_ESTATUS	223
7.26.2.12	MII_PHYSTS	223
7.26.2.13	MII_MICR	223
7.26.2.14	MII_DCOUNTER	223
7.26.2.15	MII_FCSCOUNTER	223
7.26.2.16	MII_NWAYTEST	223
7.26.2.17	MII_RERRCOUNTER	223
7.26.2.18	MII_SREVISION	223
7.26.2.19	MII_RESV1	224
7.26.2.20	MII_LBRERROR	224
7.26.2.21	MII_PHYADDR	224
7.26.2.22	MII_RESV2	224
7.26.2.23	MII_TPISTATUS	224
7.26.2.24	MII_NCONFIG	224
7.26.2.25	BMCR_RESV	224
7.26.2.26	BMCR_CTST	224
7.26.2.27	BMCR_FULLDPLX	224
7.26.2.28	BMCR_ANRESTART	224
7.26.2.29	BMCR_ISOLATE	224
7.26.2.30	BMCR_PDOWN	224
7.26.2.31	BMCR_ANENABLE	225
7.26.2.32	BMCR_SPEED100	225
7.26.2.33	BMCR_LOOPBACK	225
7.26.2.34	BMCR_RESET	225
7.26.2.35	BMSR_ERCAP	225
7.26.2.36	BMSR_JCD	225
7.26.2.37	BMSR_LSTATUS	225
7.26.2.38	BMSR_ANEGCAPABLE	225
7.26.2.39	BMSR_RFAULT	225
7.26.2.40	BMSR_ANEGCOMPLETE	225

7.26.2.41 BMSR_MFPRESUPPCAP	225
7.26.2.42 BMSR_RESV	225
7.26.2.43 BMSR_10HALF	226
7.26.2.44 BMSR_10FULL	226
7.26.2.45 BMSR_100HALF	226
7.26.2.46 BMSR_100FULL	226
7.26.2.47 BMSR_100BASE4	226
7.26.2.48 ADVERTISE_SLCT	226
7.26.2.49 ADVERTISE_CSMA	226
7.26.2.50 ADVERTISE_10HALF	226
7.26.2.51 ADVERTISE_10FULL	226
7.26.2.52 ADVERTISE_100HALF	226
7.26.2.53 ADVERTISE_100FULL	226
7.26.2.54 ADVERTISE_100BASE4	226
7.26.2.55 ADVERTISE_PAUSE_CAP	227
7.26.2.56 ADVERTISE_PAUSE_ASYM	227
7.26.2.57 ADVERTISE_RESV	227
7.26.2.58 ADVERTISE_RFAULT	227
7.26.2.59 ADVERTISE_LPACK	227
7.26.2.60 ADVERTISE_NPAGE	227
7.26.2.61 LPA_SLCT	227
7.26.2.62 LPA_10HALF	227
7.26.2.63 LPA_10FULL	227
7.26.2.64 LPA_100HALF	227
7.26.2.65 LPA_100FULL	227
7.26.2.66 LPA_100BASE4	227
7.26.2.67 LPA_PAUSE_CAP	228
7.26.2.68 LPA_PAUSE_ASYM	228
7.26.2.69 LPA_RESV	228
7.26.2.70 LPA_RFAULT	228
7.26.2.71 LPA_LPACK	228
7.26.2.72 LPA_NPAGE	228
7.26.2.73 EXPANSION_NWAY	228
7.26.2.74 EXPANSION_LCWP	228
7.26.2.75 EXPANSION_ENABLENPAGE	228
7.26.2.76 EXPANSION_NPCAPABLE	228
7.26.2.77 EXPANSION_MFAULTS	228
7.26.2.78 EXPANSION_RESV	228
7.26.2.79 NWAYTEST_RESV1	229
7.26.2.80 NWAYTEST_LOOPBACK	229

7.26.2.81 NWAYTEST_RESV2	229
7.27 MMC over SPI Driver	230
7.27.1 Detailed Description	230
7.27.2 Driver State Machine	230
7.27.3 Driver Operations	230
7.27.4 Macro Definition Documentation	232
7.27.4.1 MMC_NICE_WAITING	232
7.27.4.2 _mmc_driver_methods	232
7.27.4.3 mmclsCardInserted	232
7.27.4.4 mmclsWriteProtected	232
7.27.5 Function Documentation	233
7.27.5.1 crc7(uint8_t crc, const uint8_t *buffer, size_t len)	233
7.27.5.2 wait(MMCDriver *mmcp)	233
7.27.5.3 send_hdr(MMCDriver *mmcp, uint8_t cmd, uint32_t arg)	234
7.27.5.4 recvr1(MMCDriver *mmcp)	234
7.27.5.5 recvr3(MMCDriver *mmcp, uint8_t *buffer)	235
7.27.5.6 send_command_R1(MMCDriver *mmcp, uint8_t cmd, uint32_t arg)	236
7.27.5.7 send_command_R3(MMCDriver *mmcp, uint8_t cmd, uint32_t arg, uint8_t *response)	237
7.27.5.8 read_CxD(MMCDriver *mmcp, uint8_t cmd, uint32_t cxd[4])	237
7.27.5.9 sync(MMCDriver *mmcp)	238
7.27.5.10 mmcInit(void)	239
7.27.5.11 mmcObjectInit(MMCDriver *mmcp)	239
7.27.5.12 mmcStart(MMCDriver *mmcp, const MMCCConfig *config)	239
7.27.5.13 mmcStop(MMCDriver *mmcp)	240
7.27.5.14 mmcConnect(MMCDriver *mmcp)	240
7.27.5.15 mmcDisconnect(MMCDriver *mmcp)	241
7.27.5.16 mmcStartSequentialRead(MMCDriver *mmcp, uint32_t startblk)	242
7.27.5.17 mmcSequentialRead(MMCDriver *mmcp, uint8_t *buffer)	243
7.27.5.18 mmcStopSequentialRead(MMCDriver *mmcp)	244
7.27.5.19 mmcStartSequentialWrite(MMCDriver *mmcp, uint32_t startblk)	245
7.27.5.20 mmcSequentialWrite(MMCDriver *mmcp, const uint8_t *buffer)	246
7.27.5.21 mmcStopSequentialWrite(MMCDriver *mmcp)	247
7.27.5.22 mmcSync(MMCDriver *mmcp)	248
7.27.5.23 mmcGetInfo(MMCDriver *mmcp, BlockDeviceInfo *bdip)	249
7.27.5.24 mmcErase(MMCDriver *mmcp, uint32_t startblk, uint32_t endblk)	249
7.27.6 Variable Documentation	250
7.27.6.1 mmc_vmt	250
7.27.6.2 crc7_lookup_table	250
7.28 MMC/SD Block Device	252

7.28.1	Detailed Description	252
7.28.2	Macro Definition Documentation	256
7.28.2.1	MMCSDBLOCK_SIZE	256
7.28.2.2	MMCSDBLOCK_R1_ERROR_MASK	256
7.28.2.3	MMCSDBLOCK_CMD8_PATTERN	256
7.28.2.4	MMCSDBLOCK_CSD_MMC_CSD_STRUCTURE_SLICE	256
7.28.2.5	MMCSDBLOCK_CID_SDC_CRC_SLICE	256
7.28.2.6	_mmc_sdblock_device_methods	257
7.28.2.7	_mmc_sdblock_device_data	257
7.28.2.8	MMCSDBLOCK_R1_ERROR	257
7.28.2.9	MMCSDBLOCK_R1_STS	257
7.28.2.10	MMCSDBLOCK_R1_IS_CARD_LOCKED	257
7.28.2.11	mmc_sdblock_get_card_capacity	257
7.28.3	Function Documentation	258
7.28.3.1	_mmc_sdblock_get_slice(const uint32_t *data, uint32_t end, uint32_t start)	258
7.28.3.2	_mmc_sdblock_get_capacity(const uint32_t *csd)	258
7.28.3.3	_mmc_sdblock_get_capacity_ext(const uint8_t *ext_csd)	259
7.28.3.4	_mmc_sdblock_unpack_sdc_cid(const MMCSDBlockDevice *sdc, unpacked_sdc_cid_t *cid_sdc)	259
7.28.3.5	_mmc_sdblock_unpack_mmc_cid(const MMCSDBlockDevice *sdc, unpacked_mmc_cid_t *cid_mmc)	260
7.28.3.6	_mmc_sdblock_unpack_csd_mmc(const MMCSDBlockDevice *sdc, unpacked_mmc_csd_t *csd_mmc)	260
7.28.3.7	_mmc_sdblock_unpack_csd_v10(const MMCSDBlockDevice *sdc, unpacked_sdc_csd_10_t *csd_10)	261
7.28.3.8	_mmc_sdblock_unpack_csd_v20(const MMCSDBlockDevice *sdc, unpacked_sdc_csd_20_t *csd_20)	261
7.29	PAL Driver	263
7.29.1	Detailed Description	263
7.29.2	Implementation Rules	263
7.29.2.1	Writing on input pads	263
7.29.2.2	Reading from output pads	263
7.29.2.3	Writing unused or unimplemented port bits	263
7.29.2.4	Reading from unused or unimplemented port bits	264
7.29.2.5	Reading or writing on pins associated to other functionalities	264
7.29.3	Macro Definition Documentation	267
7.29.3.1	PAL_MODE_RESET	267
7.29.3.2	PAL_MODE_UNCONNECTED	268
7.29.3.3	PAL_MODE_INPUT	268
7.29.3.4	PAL_MODE_INPUT_PULLUP	268
7.29.3.5	PAL_MODE_INPUT_PULLDOWN	268

7.29.3.6	PAL_MODE_INPUT_ANALOG	268
7.29.3.7	PAL_MODE_OUTPUT_PUSHPULL	268
7.29.3.8	PAL_MODE_OUTPUT_OPENDRAIN	268
7.29.3.9	PAL_LOW	268
7.29.3.10	PAL_HIGH	268
7.29.3.11	PAL_EVENT_MODE_EDGES_MASK	268
7.29.3.12	PAL_EVENT_MODE_DISABLED	268
7.29.3.13	PAL_EVENT_MODE_RISING_EDGE	268
7.29.3.14	PAL_EVENT_MODE_FALLING_EDGE	269
7.29.3.15	PAL_EVENT_MODE_BOTH_EDGES	269
7.29.3.16	PAL_PORT_BIT	269
7.29.3.17	PAL_GROUP_MASK	269
7.29.3.18	_IOBUS_DATA	269
7.29.3.19	IOBUS_DECL	270
7.29.3.20	palInit	270
7.29.3.21	palReadPort	270
7.29.3.22	palReadLatch	270
7.29.3.23	palWritePort	271
7.29.3.24	palSetPort	271
7.29.3.25	palClearPort	272
7.29.3.26	palTogglePort	272
7.29.3.27	palReadGroup	272
7.29.3.28	palWriteGroup	273
7.29.3.29	palSetGroupMode	273
7.29.3.30	palReadPad	274
7.29.3.31	palWritePad	274
7.29.3.32	palSetPad	275
7.29.3.33	palClearPad	275
7.29.3.34	palTogglePad	275
7.29.3.35	palSetPadMode	276
7.29.3.36	palPadEnableEventI	276
7.29.3.37	palPadDisableEventI	277
7.29.3.38	palReadLine	277
7.29.3.39	palWriteLine	277
7.29.3.40	palSetLine	278
7.29.3.41	palClearLine	278
7.29.3.42	palToggleLine	279
7.29.3.43	palSetLineMode	279
7.29.3.44	palLineEnableEventI	279
7.29.3.45	palLineDisableEventI	280

7.29.3.46	PAL_IOPORTS_WIDTH	280
7.29.3.47	PAL_WHOLE_PORT	280
7.29.3.48	PAL_LINE	280
7.29.3.49	PAL_PORT	280
7.29.3.50	PAL_PAD	280
7.29.3.51	PAL_NOLINE	280
7.29.3.52	IOPORT1	280
7.29.3.53	pal_lld_init	280
7.29.3.54	pal_lld_readport	281
7.29.3.55	pal_lld_readlatch	281
7.29.3.56	pal_lld_writeport	281
7.29.3.57	pal_lld_setport	282
7.29.3.58	pal_lld_clearport	282
7.29.3.59	pal_lld_toggleport	283
7.29.3.60	pal_lld_readgroup	283
7.29.3.61	pal_lld_writegroup	284
7.29.3.62	pal_lld_setgroupmode	284
7.29.3.63	pal_lld_readpad	284
7.29.3.64	pal_lld_writepad	285
7.29.3.65	pal_lld_setpad	286
7.29.3.66	pal_lld_clearpad	286
7.29.3.67	pal_lld_togglepad	286
7.29.3.68	pal_lld_setpadmode	287
7.29.4	Typedef Documentation	287
7.29.4.1	palcallback_t	287
7.29.4.2	ioportmask_t	288
7.29.4.3	iomode_t	288
7.29.4.4	ioline_t	288
7.29.4.5	ioportid_t	288
7.29.5	Function Documentation	288
7.29.5.1	palReadBus(IOBus *bus)	288
7.29.5.2	palWriteBus(IOBus *bus, ioportmask_t bits)	288
7.29.5.3	palSetBusMode(IOBus *bus, iomode_t mode)	289
7.29.5.4	_pal_lld_init(const PALConfig *config)	289
7.29.5.5	_pal_lld_setgroupmode(ioportid_t port, ioportmask_t mask, iomode_t mode)	290
7.30	PWM Driver	291
7.30.1	Detailed Description	291
7.30.2	Driver State Machine	291
7.30.3	PWM Operations.	291
7.30.4	Macro Definition Documentation	294

7.30.4.1	PWM_OUTPUT_MASK	294
7.30.4.2	PWM_OUTPUT_DISABLED	294
7.30.4.3	PWM_OUTPUT_ACTIVE_HIGH	294
7.30.4.4	PWM_OUTPUT_ACTIVE_LOW	294
7.30.4.5	PWM_FRACTION_TO_WIDTH	295
7.30.4.6	PWM_DEGREES_TO_WIDTH	295
7.30.4.7	PWM_PERCENTAGE_TO_WIDTH	296
7.30.4.8	pwmChangePeriodI	296
7.30.4.9	pwmEnableChannelI	297
7.30.4.10	pwmDisableChannelI	297
7.30.4.11	pwmIsChannelEnabledI	298
7.30.4.12	pwmEnablePeriodicNotificationI	298
7.30.4.13	pwmDisablePeriodicNotificationI	299
7.30.4.14	pwmEnableChannelNotificationI	299
7.30.4.15	pwmDisableChannelNotificationI	300
7.30.4.16	PWM_CHANNELS	300
7.30.4.17	PLATFORM_PWM_USE_PWM1	300
7.30.4.18	pwm_lld_change_period	300
7.30.5	Typedef Documentation	301
7.30.5.1	PWMDriver	301
7.30.5.2	pwmcallback_t	301
7.30.5.3	pwmmode_t	301
7.30.5.4	pwmchannel_t	301
7.30.5.5	pwmchnmsk_t	301
7.30.5.6	pwmcnt_t	301
7.30.6	Enumeration Type Documentation	302
7.30.6.1	pwmstate_t	302
7.30.7	Function Documentation	302
7.30.7.1	pwmInit(void)	302
7.30.7.2	pwmObjectInit(PWMDriver *pwmp)	302
7.30.7.3	pwmStart(PWMDriver *pwmp, const PWMConfig *config)	302
7.30.7.4	pwmStop(PWMDriver *pwmp)	303
7.30.7.5	pwmChangePeriod(PWMDriver *pwmp, pwmcnt_t period)	303
7.30.7.6	pwmEnableChannel(PWMDriver *pwmp, pwmchannel_t channel, pwmcnt_t width)	304
7.30.7.7	pwmDisableChannel(PWMDriver *pwmp, pwmchannel_t channel)	304
7.30.7.8	pwmEnablePeriodicNotification(PWMDriver *pwmp)	305
7.30.7.9	pwmDisablePeriodicNotification(PWMDriver *pwmp)	305
7.30.7.10	pwmEnableChannelNotification(PWMDriver *pwmp, pwmchannel_t channel)	306
7.30.7.11	pwmDisableChannelNotification(PWMDriver *pwmp, pwmchannel_t channel)	306
7.30.7.12	pwm_lld_init(void)	307

7.30.7.13	pwm_ll_start(PWMDriver *pwmp)	307
7.30.7.14	pwm_ll_stop(PWMDriver *pwmp)	307
7.30.7.15	pwm_ll_enable_channel(PWMDriver *pwmp, pwmchannel_t channel, pwmcnt← _t width)	307
7.30.7.16	pwm_ll_disable_channel(PWMDriver *pwmp, pwmchannel_t channel)	308
7.30.7.17	pwm_ll_enable_periodic_notification(PWMDriver *pwmp)	308
7.30.7.18	pwm_ll_disable_periodic_notification(PWMDriver *pwmp)	309
7.30.7.19	pwm_ll_enable_channel_notification(PWMDriver *pwmp, pwmchannel_t channel)	309
7.30.7.20	pwm_ll_disable_channel_notification(PWMDriver *pwmp, pwmchannel_t channel)	310
7.30.8	Variable Documentation	310
7.30.8.1	PWMD1	310
7.31	QSPI Driver	311
7.31.1	Detailed Description	311
7.31.2	Macro Definition Documentation	314
7.31.2.1	QSPI_USE_WAIT	314
7.31.2.2	QSPI_USE_MUTUAL_EXCLUSION	314
7.31.2.3	qspiStartCommandI	314
7.31.2.4	qspiStartSendI	314
7.31.2.5	qspiStartReceiveI	315
7.31.2.6	qspiMapFlashI	316
7.31.2.7	qspiUnmapFlashI	316
7.31.2.8	_qspi_wakeup_isr	316
7.31.2.9	_qspi_isr_code	317
7.31.2.10	PLATFORM_QSPI_USE_QSPI1	317
7.31.3	Typedef Documentation	317
7.31.3.1	QSPIDriver	317
7.31.3.2	qspicallback_t	318
7.31.4	Enumeration Type Documentation	318
7.31.4.1	qspistate_t	318
7.31.5	Function Documentation	318
7.31.5.1	qspiInit(void)	318
7.31.5.2	qspiObjectInit(QSPIDriver *qspip)	318
7.31.5.3	qspiStart(QSPIDriver *qspip, const QSPICongig *config)	319
7.31.5.4	qspiStop(QSPIDriver *qspip)	319
7.31.5.5	qspiStartCommand(QSPIDriver *qspip, const qspi_command_t *cmdp)	320
7.31.5.6	qspiStartSend(QSPIDriver *qspip, const qspi_command_t *cmdp, size_t n, const uint8_t *txbuf)	320
7.31.5.7	qspiStartReceive(QSPIDriver *qspip, const qspi_command_t *cmdp, size_t n, uint8_t *rxbuf)	320
7.31.5.8	qspiCommand(QSPIDriver *qspip, const qspi_command_t *cmdp)	321

7.31.5.9	qspiSend(QSPIDriver *qspip, const qspi_command_t *cmdp, size_t n, const uint8_t *txbuf)	321
7.31.5.10	qspiReceive(QSPIDriver *qspip, const qspi_command_t *cmdp, size_t n, uint8_t *rxbuf)	322
7.31.5.11	qspiMapFlash(QSPIDriver *qspip, const qspi_command_t *cmdp, uint8_t **addrp)	322
7.31.5.12	qspiUnmapFlash(QSPIDriver *qspip)	322
7.31.5.13	qspiAcquireBus(QSPIDriver *qspip)	323
7.31.5.14	qspiReleaseBus(QSPIDriver *qspip)	323
7.31.5.15	qspi_llc_init(void)	323
7.31.5.16	qspi_llc_start(QSPIDriver *qspip)	324
7.31.5.17	qspi_llc_stop(QSPIDriver *qspip)	324
7.31.5.18	qspi_llc_command(QSPIDriver *qspip, const qspi_command_t *cmdp)	324
7.31.5.19	qspi_llc_send(QSPIDriver *qspip, const qspi_command_t *cmdp, size_t n, const uint8_t *txbuf)	325
7.31.5.20	qspi_llc_receive(QSPIDriver *qspip, const qspi_command_t *cmdp, size_t n, uint8_t *rxbuf)	325
7.31.5.21	qspi_llc_map_flash(QSPIDriver *qspip, const qspi_command_t *cmdp, uint8_t **addrp)	325
7.31.5.22	qspi_llc_unmap_flash(QSPIDriver *qspip)	326
7.31.6	Variable Documentation	326
7.31.6.1	QSPID1	326
7.32	RTC Driver	327
7.32.1	Detailed Description	327
7.32.2	Macro Definition Documentation	329
7.32.2.1	RTC_BASE_YEAR	329
7.32.2.2	RTC_SUPPORTS_CALLBACKS	329
7.32.2.3	RTC_ALARMS	329
7.32.2.4	RTC_HAS_STORAGE	329
7.32.2.5	PLATFORM_RTC_USE_RTC1	329
7.32.2.6	_rtc_driver_methods	329
7.32.3	Typedef Documentation	329
7.32.3.1	RTCDriver	329
7.32.3.2	rtcalarm_t	329
7.32.3.3	rtccb_t	330
7.32.4	Enumeration Type Documentation	330
7.32.4.1	rtcevent_t	330
7.32.5	Function Documentation	330
7.32.5.1	rtcInit(void)	330
7.32.5.2	rtcObjectInit(RTCDriver *rtcp)	330
7.32.5.3	rtcSetTime(RTCDriver *rtcp, const RTCDateTime *timespec)	330
7.32.5.4	rtcGetTime(RTCDriver *rtcp, RTCDateTime *timespec)	331

7.32.5.5	<code>rtcSetAlarm(RTCDriver *rtcp, rtcalarm_t alarm, const RTCArm *alarmspec)</code>	332
7.32.5.6	<code>rtcGetAlarm(RTCDriver *rtcp, rtcalarm_t alarm, RTCArm *alarmspec)</code>	332
7.32.5.7	<code>rtcSetCallback(RTCDriver *rtcp, rtccb_t callback)</code>	333
7.32.5.8	<code>rtcConvertDateTimeToStructTm(const RTCDatetime *timespec, struct tm *timp, uint32_t *tv_msec)</code>	333
7.32.5.9	<code>rtcConvertStructTmToDateTime(const struct tm *timp, uint32_t tv_msec, RTCDatetime *timespec)</code>	334
7.32.5.10	<code>rtcConvertDateTimeToFAT(const RTCDatetime *timespec)</code>	334
7.32.5.11	<code>rtc_llid_init(void)</code>	334
7.32.5.12	<code>rtc_llid_set_time(RTCDriver *rtcp, const RTCDatetime *timespec)</code>	335
7.32.5.13	<code>rtc_llid_get_time(RTCDriver *rtcp, RTCDatetime *timespec)</code>	335
7.32.5.14	<code>rtc_llid_set_alarm(RTCDriver *rtcp, rtcalarm_t alarm, const RTCArm *alarmspec)</code>	335
7.32.5.15	<code>rtc_llid_get_alarm(RTCDriver *rtcp, rtcalarm_t alarm, RTCArm *alarmspec)</code>	336
7.33	SDC Driver	337
7.33.1	Detailed Description	337
7.33.2	Driver State Machine	337
7.33.3	Driver Operations	337
7.33.4	Macro Definition Documentation	340
7.33.4.1	<code>SDC_INIT_RETRY</code>	340
7.33.4.2	<code>SDC_MMC_SUPPORT</code>	340
7.33.4.3	<code>SDC_NICE_WAITING</code>	341
7.33.4.4	<code>SDC_INIT_OCR_V20</code>	341
7.33.4.5	<code>SDC_INIT_OCR</code>	341
7.33.4.6	<code>sdclsCardInserted</code>	341
7.33.4.7	<code>sdclsWriteProtected</code>	341
7.33.4.8	<code>PLATFORM_SDC_USE_SDC1</code>	342
7.33.4.9	<code>_sdc_driver_methods</code>	342
7.33.5	Typedef Documentation	342
7.33.5.1	<code>sdcmode_t</code>	342
7.33.5.2	<code>sdcflags_t</code>	342
7.33.5.3	<code>SDCDriver</code>	342
7.33.6	Enumeration Type Documentation	342
7.33.6.1	<code>mmc_switch_t</code>	342
7.33.6.2	<code>sd_switch_t</code>	343
7.33.6.3	<code>sd_switch_function_t</code>	343
7.33.6.4	<code>sdcbusmode_t</code>	343
7.33.6.5	<code>sdcbusclk_t</code>	343
7.33.7	Function Documentation	343
7.33.7.1	<code>mode_detect(SDCDriver *sdcp)</code>	343
7.33.7.2	<code>mmc_init(SDCDriver *sdcp)</code>	344

7.33.7.3	sd_init(SDCDriver *sdcp)	344
7.33.7.4	mmc_cmd6_construct(mmc_switch_t access, uint32_t idx, uint32_t value, uint32_t cmd_set)	345
7.33.7.5	sd_cmd6_construct(sd_switch_t mode, sd_switch_function_t function, uint32_t value)	345
7.33.7.6	sd_cmd6_extract_info(sd_switch_function_t function, const uint8_t *buf)	346
7.33.7.7	sd_cmd6_check_status(sd_switch_function_t function, const uint8_t *buf)	346
7.33.7.8	sd_detect_bus_clk(SDCDriver *sdcp, sdcbusclk_t *clk)	346
7.33.7.9	mmc_detect_bus_clk(SDCDriver *sdcp, sdcbusclk_t *clk)	347
7.33.7.10	detect_bus_clk(SDCDriver *sdcp, sdcbusclk_t *clk)	348
7.33.7.11	sd_set_bus_width(SDCDriver *sdcp)	349
7.33.7.12	mmc_set_bus_width(SDCDriver *sdcp)	350
7.33.7.13	_sd_wait_for_transfer_state(SDCDriver *sdcp)	351
7.33.7.14	sdInit(void)	351
7.33.7.15	sdObjectInit(SDCDriver *sdcp)	352
7.33.7.16	sdStart(SDCDriver *sdcp, const SDConfig *config)	352
7.33.7.17	sdStop(SDCDriver *sdcp)	352
7.33.7.18	sdConnect(SDCDriver *sdcp)	353
7.33.7.19	sdDisconnect(SDCDriver *sdcp)	354
7.33.7.20	sdRead(SDCDriver *sdcp, uint32_t startblk, uint8_t *buf, uint32_t n)	355
7.33.7.21	sdWrite(SDCDriver *sdcp, uint32_t startblk, const uint8_t *buf, uint32_t n)	356
7.33.7.22	sdGetAndClearErrors(SDCDriver *sdcp)	356
7.33.7.23	sdSync(SDCDriver *sdcp)	357
7.33.7.24	sdGetInfo(SDCDriver *sdcp, BlockDeviceInfo *bdip)	357
7.33.7.25	sdErase(SDCDriver *sdcp, uint32_t startblk, uint32_t endblk)	358
7.33.7.26	sd_ll_init(void)	358
7.33.7.27	sd_ll_start(SDCDriver *sdcp)	359
7.33.7.28	sd_ll_stop(SDCDriver *sdcp)	359
7.33.7.29	sd_ll_start_clk(SDCDriver *sdcp)	359
7.33.7.30	sd_ll_set_data_clk(SDCDriver *sdcp, sdcbusclk_t clk)	360
7.33.7.31	sd_ll_stop_clk(SDCDriver *sdcp)	360
7.33.7.32	sd_ll_set_bus_mode(SDCDriver *sdcp, sdcbusmode_t mode)	360
7.33.7.33	sd_ll_send_cmd_none(SDCDriver *sdcp, uint8_t cmd, uint32_t arg)	360
7.33.7.34	sd_ll_send_cmd_short(SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp)	361
7.33.7.35	sd_ll_send_cmd_short_crc(SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp)	361
7.33.7.36	sd_ll_send_cmd_long_crc(SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp)	362
7.33.7.37	sd_ll_read(SDCDriver *sdcp, uint32_t startblk, uint8_t *buf, uint32_t n)	362
7.33.7.38	sd_ll_write(SDCDriver *sdcp, uint32_t startblk, const uint8_t *buf, uint32_t n)	363

7.33.7.39	<code>sdc_llid_sync(SDCDriver *sdcp)</code>	363
7.33.8	Variable Documentation	363
7.33.8.1	<code>sdc_vmt</code>	364
7.33.8.2	<code>SDCD1</code>	364
7.34	Serial Driver	365
7.34.1	Detailed Description	365
7.34.2	Driver State Machine	365
7.34.3	Macro Definition Documentation	367
7.34.3.1	<code>SD_PARITY_ERROR</code>	367
7.34.3.2	<code>SD_FRAMING_ERROR</code>	368
7.34.3.3	<code>SD_OVERRUN_ERROR</code>	368
7.34.3.4	<code>SD_NOISE_ERROR</code>	368
7.34.3.5	<code>SD_BREAK_DETECTED</code>	368
7.34.3.6	<code>SD_QUEUE_FULL_ERROR</code>	368
7.34.3.7	<code>SERIAL_DEFAULT_BITRATE</code>	368
7.34.3.8	<code>SERIAL_BUFFERS_SIZE</code>	368
7.34.3.9	<code>_serial_driver_methods</code>	368
7.34.3.10	<code>sdPut</code>	368
7.34.3.11	<code>sdPutTimeout</code>	369
7.34.3.12	<code>sdGet</code>	369
7.34.3.13	<code>sdGetTimeout</code>	369
7.34.3.14	<code>sdWrite</code>	370
7.34.3.15	<code>sdWriteTimeout</code>	370
7.34.3.16	<code>sdAsynchronousWrite</code>	370
7.34.3.17	<code>sdRead</code>	371
7.34.3.18	<code>sdReadTimeout</code>	371
7.34.3.19	<code>sdAsynchronousRead</code>	371
7.34.3.20	<code>PLATFORM_SERIAL_USE_USART1</code>	371
7.34.3.21	<code>_serial_driver_data</code>	372
7.34.4	Typedef Documentation	372
7.34.4.1	<code>SerialDriver</code>	372
7.34.5	Enumeration Type Documentation	372
7.34.5.1	<code>sdstate_t</code>	372
7.34.6	Function Documentation	372
7.34.6.1	<code>sdInit(void)</code>	372
7.34.6.2	<code>sdObjectInit(SerialDriver *sdp, qnotify_t inotify, qnotify_t onotify)</code>	373
7.34.6.3	<code>sdStart(SerialDriver *sdp, const SerialConfig *config)</code>	373
7.34.6.4	<code>sdStop(SerialDriver *sdp)</code>	374
7.34.6.5	<code>sdIncomingData(SerialDriver *sdp, uint8_t b)</code>	374
7.34.6.6	<code>sdRequestData(SerialDriver *sdp)</code>	376

7.34.6.7	<code>sdPutWouldBlock(SerialDriver *sdp)</code>	377
7.34.6.8	<code>sdGetWouldBlock(SerialDriver *sdp)</code>	377
7.34.6.9	<code>sd_lld_init(void)</code>	378
7.34.6.10	<code>sd_lld_start(SerialDriver *sdp, const SerialConfig *config)</code>	378
7.34.6.11	<code>sd_lld_stop(SerialDriver *sdp)</code>	379
7.34.7	Variable Documentation	379
7.34.7.1	<code>SD1</code>	379
7.34.7.2	<code>default_config</code>	379
7.35	Serial over USB Driver	380
7.35.1	Detailed Description	380
7.35.2	Driver State Machine	380
7.35.3	Macro Definition Documentation	382
7.35.3.1	<code>SERIAL_USB_BUFFERS_SIZE</code>	382
7.35.3.2	<code>SERIAL_USB_BUFFERS_NUMBER</code>	382
7.35.3.3	<code>_serial_usb_driver_data</code>	382
7.35.3.4	<code>_serial_usb_driver_methods</code>	382
7.35.4	Typedef Documentation	382
7.35.4.1	<code>SerialUSBDriver</code>	382
7.35.5	Enumeration Type Documentation	383
7.35.5.1	<code>sdustate_t</code>	383
7.35.6	Function Documentation	383
7.35.6.1	<code>ibnotify(io_buffers_queue_t *bqp)</code>	383
7.35.6.2	<code>obnotify(io_buffers_queue_t *bqp)</code>	383
7.35.6.3	<code>sduInit(void)</code>	383
7.35.6.4	<code>sduObjectInit(SerialUSBDriver *sdup)</code>	384
7.35.6.5	<code>sduStart(SerialUSBDriver *sdup, const SerialUSBConfig *config)</code>	384
7.35.6.6	<code>sduStop(SerialUSBDriver *sdup)</code>	385
7.35.6.7	<code>sduSuspendHookI(SerialUSBDriver *sdup)</code>	385
7.35.6.8	<code>sduWakeupHookI(SerialUSBDriver *sdup)</code>	386
7.35.6.9	<code>sduConfigureHookI(SerialUSBDriver *sdup)</code>	386
7.35.6.10	<code>sduRequestsHook(USBDriver *usbp)</code>	386
7.35.6.11	<code>sduSOFHookI(SerialUSBDriver *sdup)</code>	387
7.35.6.12	<code>sduDataTransmitted(USBDriver *usbp, usbep_t ep)</code>	388
7.35.6.13	<code>sduDataReceived(USBDriver *usbp, usbep_t ep)</code>	388
7.35.6.14	<code>sduInterruptTransmitted(USBDriver *usbp, usbep_t ep)</code>	388
7.36	SPI Driver	391
7.36.1	Detailed Description	391
7.36.2	Driver State Machine	391
7.36.3	Macro Definition Documentation	394
7.36.3.1	<code>SPI_USE_WAIT</code>	394

7.36.3.2	SPI_USE_MUTUAL_EXCLUSION	394
7.36.3.3	spiSelectl	394
7.36.3.4	spiUnselectl	394
7.36.3.5	spiStartIgnorel	395
7.36.3.6	spiStartExchangel	395
7.36.3.7	spiStartSendl	396
7.36.3.8	spiStartReceiveI	397
7.36.3.9	spiPolledExchange	397
7.36.3.10	_spi_wakeup_isr	398
7.36.3.11	_spi_isr_code	398
7.36.3.12	PLATFORM_SPI_USE_SPI1	399
7.36.4	Typedef Documentation	399
7.36.4.1	SPIDriver	399
7.36.4.2	spicallback_t	399
7.36.5	Enumeration Type Documentation	399
7.36.5.1	spistate_t	399
7.36.6	Function Documentation	399
7.36.6.1	spiInit(void)	399
7.36.6.2	spiObjectInit(SPIDriver *spip)	400
7.36.6.3	spiStart(SPIDriver *spip, const SPIConfig *config)	400
7.36.6.4	spiStop(SPIDriver *spip)	401
7.36.6.5	spiSelect(SPIDriver *spip)	401
7.36.6.6	spiUnselect(SPIDriver *spip)	402
7.36.6.7	spiStartIgnore(SPIDriver *spip, size_t n)	402
7.36.6.8	spiStartExchange(SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)	402
7.36.6.9	spiStartSend(SPIDriver *spip, size_t n, const void *txbuf)	404
7.36.6.10	spiStartReceive(SPIDriver *spip, size_t n, void *rxbuf)	404
7.36.6.11	spiIgnore(SPIDriver *spip, size_t n)	405
7.36.6.12	spiExchange(SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)	405
7.36.6.13	spiSend(SPIDriver *spip, size_t n, const void *txbuf)	406
7.36.6.14	spiReceive(SPIDriver *spip, size_t n, void *rxbuf)	406
7.36.6.15	spiAcquireBus(SPIDriver *spip)	407
7.36.6.16	spiReleaseBus(SPIDriver *spip)	407
7.36.6.17	spi_lld_init(void)	407
7.36.6.18	spi_lld_start(SPIDriver *spip)	408
7.36.6.19	spi_lld_stop(SPIDriver *spip)	408
7.36.6.20	spi_lld_select(SPIDriver *spip)	408
7.36.6.21	spi_lld_unselect(SPIDriver *spip)	409
7.36.6.22	spi_lld_ignore(SPIDriver *spip, size_t n)	409
7.36.6.23	spi_lld_exchange(SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)	409

7.36.6.24 spi_llc_send(SPIDriver *spip, size_t n, const void *txbuf)	410
7.36.6.25 spi_llc_receive(SPIDriver *spip, size_t n, void *rxbuf)	410
7.36.6.26 spi_llc_polled_exchange(SPIDriver *spip, uint16_t frame)	411
7.36.7 Variable Documentation	411
7.36.7.1 SPID1	411
7.37 ST Driver	412
7.37.1 Detailed Description	412
7.37.2 Macro Definition Documentation	412
7.37.2.1 stGetCounter	412
7.37.2.2 stIsAlarmActive	413
7.37.3 Function Documentation	413
7.37.3.1 stInit(void)	413
7.37.3.2 stStartAlarm(systime_t abstime)	414
7.37.3.3 stStopAlarm(void)	414
7.37.3.4 stSetAlarm(systime_t abstime)	415
7.37.3.5 stGetAlarm(void)	415
7.37.3.6 st_llc_init(void)	416
7.37.3.7 st_llc_get_counter(void)	416
7.37.3.8 st_llc_start_alarm(systime_t abstime)	416
7.37.3.9 st_llc_stop_alarm(void)	416
7.37.3.10 st_llc_set_alarm(systime_t abstime)	416
7.37.3.11 st_llc_get_alarm(void)	417
7.37.3.12 st_llc_is_alarm_active(void)	417
7.38 UART Driver	418
7.38.1 Detailed Description	418
7.38.2 Driver State Machine	418
7.38.2.1 Transmitter sub State Machine	419
7.38.2.2 Receiver sub State Machine	419
7.38.3 Macro Definition Documentation	422
7.38.3.1 UART_NO_ERROR	422
7.38.3.2 UART_PARITY_ERROR	422
7.38.3.3 UART_FRAMING_ERROR	422
7.38.3.4 UART_OVERRUN_ERROR	422
7.38.3.5 UART_NOISE_ERROR	422
7.38.3.6 UART_BREAK_DETECTED	423
7.38.3.7 UART_USE_WAIT	423
7.38.3.8 UART_USE_MUTUAL_EXCLUSION	423
7.38.3.9 _uart_wakeup_tx1_isr	423
7.38.3.10 _uart_wakeup_tx2_isr	423
7.38.3.11 _uart_wakeup_rx_complete_isr	424

7.38.3.12	<code>_uart_wakeup_rx_error_isr</code>	424
7.38.3.13	<code>_uart_wakeup_rx_timeout_isr</code>	424
7.38.3.14	<code>_uart_tx1_isr_code</code>	425
7.38.3.15	<code>_uart_tx2_isr_code</code>	425
7.38.3.16	<code>_uart_rx_complete_isr_code</code>	426
7.38.3.17	<code>_uart_rx_error_isr_code</code>	427
7.38.3.18	<code>_uart_rx_idle_code</code>	427
7.38.3.19	<code>_uart_timeout_isr_code</code>	428
7.38.3.20	<code>PLATFORM_UART_USE_UART1</code>	428
7.38.4	Typedef Documentation	428
7.38.4.1	<code>uartflags_t</code>	428
7.38.4.2	<code>UARTDriver</code>	429
7.38.4.3	<code>uartcb_t</code>	429
7.38.4.4	<code>uartccb_t</code>	429
7.38.4.5	<code>uartecb_t</code>	429
7.38.5	Enumeration Type Documentation	429
7.38.5.1	<code>uartstate_t</code>	429
7.38.5.2	<code>uarttxstate_t</code>	429
7.38.5.3	<code>uartrxstate_t</code>	430
7.38.6	Function Documentation	430
7.38.6.1	<code>uartInit(void)</code>	430
7.38.6.2	<code>uartObjectInit(UARTDriver *uartp)</code>	430
7.38.6.3	<code>uartStart(UARTDriver *uartp, const UARTConfig *config)</code>	431
7.38.6.4	<code>uartStop(UARTDriver *uartp)</code>	431
7.38.6.5	<code>uartStartSend(UARTDriver *uartp, size_t n, const void *txbuf)</code>	431
7.38.6.6	<code>uartStartSendI(UARTDriver *uartp, size_t n, const void *txbuf)</code>	432
7.38.6.7	<code>uartStopSend(UARTDriver *uartp)</code>	433
7.38.6.8	<code>uartStopSendI(UARTDriver *uartp)</code>	433
7.38.6.9	<code>uartStartReceive(UARTDriver *uartp, size_t n, void *rxbuf)</code>	434
7.38.6.10	<code>uartStartReceiveI(UARTDriver *uartp, size_t n, void *rxbuf)</code>	435
7.38.6.11	<code>uartStopReceive(UARTDriver *uartp)</code>	435
7.38.6.12	<code>uartStopReceiveI(UARTDriver *uartp)</code>	436
7.38.6.13	<code>uartSendTimeout(UARTDriver *uartp, size_t *np, const void *txbuf, systime_t timeout)</code>	437
7.38.6.14	<code>uartSendFullTimeout(UARTDriver *uartp, size_t *np, const void *txbuf, systime_t timeout)</code>	438
7.38.6.15	<code>uartReceiveTimeout(UARTDriver *uartp, size_t *np, void *rxbuf, systime_t timeout)</code>	439
7.38.6.16	<code>uartAcquireBus(UARTDriver *uartp)</code>	440
7.38.6.17	<code>uartReleaseBus(UARTDriver *uartp)</code>	440
7.38.6.18	<code>uart_lld_init(void)</code>	440

7.38.6.19	uart_lld_start(UARTDriver *uartp)	441
7.38.6.20	uart_lld_stop(UARTDriver *uartp)	441
7.38.6.21	uart_lld_start_send(UARTDriver *uartp, size_t n, const void *txbuf)	441
7.38.6.22	uart_lld_stop_send(UARTDriver *uartp)	442
7.38.6.23	uart_lld_start_receive(UARTDriver *uartp, size_t n, void *rxbuf)	442
7.38.6.24	uart_lld_stop_receive(UARTDriver *uartp)	442
7.38.7	Variable Documentation	443
7.38.7.1	UARTD1	443
7.39	USB Driver	444
7.39.1	Detailed Description	444
7.39.2	Driver State Machine	444
7.39.3	USB Operations	444
7.39.3.1	USB Implementation	444
7.39.3.2	USB Endpoints	445
7.39.3.3	USB Callbacks	446
7.39.4	Macro Definition Documentation	450
7.39.4.1	USB_DESC_INDEX	450
7.39.4.2	USB_DESC_BYTE	451
7.39.4.3	USB_DESC_WORD	451
7.39.4.4	USB_DESC_BCD	451
7.39.4.5	USB_DESC_DEVICE	451
7.39.4.6	USB_DESC_CONFIGURATION_SIZE	451
7.39.4.7	USB_DESC_CONFIGURATION	452
7.39.4.8	USB_DESC_INTERFACE_SIZE	452
7.39.4.9	USB_DESC_INTERFACE	452
7.39.4.10	USB_DESC_INTERFACE_ASSOCIATION_SIZE	452
7.39.4.11	USB_DESC_INTERFACE_ASSOCIATION	452
7.39.4.12	USB_DESC_ENDPOINT_SIZE	453
7.39.4.13	USB_DESC_ENDPOINT	453
7.39.4.14	USB_EP_MODE_TYPE	453
7.39.4.15	USB_EP_MODE_TYPE_CTRL	453
7.39.4.16	USB_EP_MODE_TYPE_ISOC	453
7.39.4.17	USB_EP_MODE_TYPE_BULK	453
7.39.4.18	USB_EP_MODE_TYPE_INTR	453
7.39.4.19	USB_USE_WAIT	454
7.39.4.20	usbGetDriverStatel	454
7.39.4.21	usbConnectBus	454
7.39.4.22	usbDisconnectBus	454
7.39.4.23	usbGetFrameNumberX	454
7.39.4.24	usbGetTransmitStatusl	455

7.39.4.25	usbGetReceiveStatusI	455
7.39.4.26	usbGetReceiveTransactionSizeX	456
7.39.4.27	usbSetupTransfer	456
7.39.4.28	usbReadSetup	457
7.39.4.29	_usb_isr_invoke_event_cb	457
7.39.4.30	_usb_isr_invoke_sof_cb	457
7.39.4.31	_usb_isr_invoke_setup_cb	458
7.39.4.32	_usb_isr_invoke_in_cb	458
7.39.4.33	_usb_isr_invoke_out_cb	458
7.39.4.34	USB_MAX_ENDPOINTS	459
7.39.4.35	USB_EP0_STATUS_STAGE	459
7.39.4.36	USB_SET_ADDRESS_MODE	459
7.39.4.37	USB_SET_ADDRESS_ACK_HANDLING	459
7.39.4.38	PLATFORM_USB_USE_USB1	459
7.39.4.39	usb_lld_get_frame_number	459
7.39.4.40	usb_lld_get_transaction_size	460
7.39.4.41	usb_lld_connect_bus	460
7.39.4.42	usb_lld_disconnect_bus	460
7.39.5	Typedef Documentation	460
7.39.5.1	USBDriver	460
7.39.5.2	usbep_t	461
7.39.5.3	usbcallback_t	461
7.39.5.4	usbepcallback_t	461
7.39.5.5	usbeventcb_t	461
7.39.5.6	usbreqhandler_t	461
7.39.5.7	usbgetdescriptor_t	462
7.39.6	Enumeration Type Documentation	462
7.39.6.1	usbstate_t	462
7.39.6.2	usbepstatus_t	462
7.39.6.3	usbep0state_t	462
7.39.6.4	usbevent_t	462
7.39.7	Function Documentation	463
7.39.7.1	set_address(USBDriver *usbp)	463
7.39.7.2	default_handler(USBDriver *usbp)	463
7.39.7.3	usbInit(void)	464
7.39.7.4	usbObjectInit(USBDriver *usbp)	464
7.39.7.5	usbStart(USBDriver *usbp, const USBConfig *config)	465
7.39.7.6	usbStop(USBDriver *usbp)	465
7.39.7.7	usbInitEndpointI(USBDriver *usbp, usbep_t ep, const USBEndpointConfig *epcp)	466
7.39.7.8	usbDisableEndpointsI(USBDriver *usbp)	466

7.39.7.9	usbStartReceive(USBDriver *usbp, usbep_t ep, uint8_t *buf, size_t n)	467
7.39.7.10	usbStartTransmit(USBDriver *usbp, usbep_t ep, const uint8_t *buf, size_t n)	468
7.39.7.11	usbReceive(USBDriver *usbp, usbep_t ep, uint8_t *buf, size_t n)	468
7.39.7.12	usbTransmit(USBDriver *usbp, usbep_t ep, const uint8_t *buf, size_t n)	469
7.39.7.13	usbStallReceive(USBDriver *usbp, usbep_t ep)	470
7.39.7.14	usbStallTransmit(USBDriver *usbp, usbep_t ep)	470
7.39.7.15	_usb_reset(USBDriver *usbp)	471
7.39.7.16	_usb_suspend(USBDriver *usbp)	471
7.39.7.17	_usb_wakeup(USBDriver *usbp)	472
7.39.7.18	_usb_ep0setup(USBDriver *usbp, usbep_t ep)	472
7.39.7.19	_usb_ep0in(USBDriver *usbp, usbep_t ep)	473
7.39.7.20	_usb_ep0out(USBDriver *usbp, usbep_t ep)	474
7.39.7.21	usb_lld_init(void)	475
7.39.7.22	usb_lld_start(USBDriver *usbp)	475
7.39.7.23	usb_lld_stop(USBDriver *usbp)	475
7.39.7.24	usb_lld_reset(USBDriver *usbp)	475
7.39.7.25	usb_lld_set_address(USBDriver *usbp)	476
7.39.7.26	usb_lld_init_endpoint(USBDriver *usbp, usbep_t ep)	476
7.39.7.27	usb_lld_disable_endpoints(USBDriver *usbp)	476
7.39.7.28	usb_lld_get_status_out(USBDriver *usbp, usbep_t ep)	477
7.39.7.29	usb_lld_get_status_in(USBDriver *usbp, usbep_t ep)	477
7.39.7.30	usb_lld_read_setup(USBDriver *usbp, usbep_t ep, uint8_t *buf)	477
7.39.7.31	usb_lld_prepare_receive(USBDriver *usbp, usbep_t ep)	478
7.39.7.32	usb_lld_prepare_transmit(USBDriver *usbp, usbep_t ep)	478
7.39.7.33	usb_lld_start_out(USBDriver *usbp, usbep_t ep)	478
7.39.7.34	usb_lld_start_in(USBDriver *usbp, usbep_t ep)	479
7.39.7.35	usb_lld_stall_out(USBDriver *usbp, usbep_t ep)	479
7.39.7.36	usb_lld_stall_in(USBDriver *usbp, usbep_t ep)	479
7.39.7.37	usb_lld_clear_out(USBDriver *usbp, usbep_t ep)	480
7.39.7.38	usb_lld_clear_in(USBDriver *usbp, usbep_t ep)	480
7.39.8	Variable Documentation	480
7.39.8.1	USBD1	480
7.39.8.2	ep0_state	480
7.39.8.3	in	480
7.39.8.4	out	480
7.39.8.5	ep0config	480
7.40	USB CDC Header	482
7.40.1	Detailed Description	482
7.41	WDG Driver	484
7.41.1	Detailed Description	484

7.41.2	Macro Definition Documentation	485
7.41.2.1	wdgResetI	485
7.41.2.2	PLATFORM_WDG_USE_WDG1	485
7.41.3	Typedef Documentation	485
7.41.3.1	WDGDriver	485
7.41.4	Enumeration Type Documentation	485
7.41.4.1	wdgstate_t	485
7.41.5	Function Documentation	485
7.41.5.1	wdgInit(void)	485
7.41.5.2	wdgStart(WDGDriver *wdgp, const WDGConfig *config)	486
7.41.5.3	wdgStop(WDGDriver *wdgp)	486
7.41.5.4	wdgReset(WDGDriver *wdgp)	487
7.41.5.5	wdg_lld_init(void)	487
7.41.5.6	wdg_lld_start(WDGDriver *wdgp)	487
7.41.5.7	wdg_lld_stop(WDGDriver *wdgp)	487
7.41.5.8	wdg_lld_reset(WDGDriver *wdgp)	488
8	Data Structure Documentation	489
8.1	ADCCConfig Struct Reference	489
8.1.1	Detailed Description	489
8.2	ADCConversionGroup Struct Reference	490
8.2.1	Detailed Description	490
8.2.2	Field Documentation	491
8.2.2.1	circular	491
8.2.2.2	num_channels	491
8.2.2.3	end_cb	491
8.2.2.4	error_cb	491
8.3	ADCDriver Struct Reference	491
8.3.1	Detailed Description	492
8.3.2	Field Documentation	493
8.3.2.1	state	493
8.3.2.2	config	493
8.3.2.3	samples	493
8.3.2.4	depth	493
8.3.2.5	grpp	493
8.3.2.6	thread	493
8.3.2.7	mutex	493
8.4	BaseAsynchronousChannel Struct Reference	493
8.4.1	Detailed Description	495
8.4.2	Field Documentation	495

8.4.2.1	vmt	495
8.5	BaseAsynchronousChannelVMT Struct Reference	495
8.5.1	Detailed Description	497
8.6	BaseBlockDevice Struct Reference	497
8.6.1	Detailed Description	499
8.6.2	Field Documentation	499
8.6.2.1	vmt	499
8.7	BaseBlockDeviceVMT Struct Reference	499
8.7.1	Detailed Description	500
8.8	BaseChannel Struct Reference	500
8.8.1	Detailed Description	502
8.8.2	Field Documentation	502
8.8.2.1	vmt	502
8.9	BaseChannelVMT Struct Reference	502
8.9.1	Detailed Description	504
8.10	BaseSequentialStream Struct Reference	504
8.10.1	Detailed Description	505
8.10.2	Field Documentation	505
8.10.2.1	vmt	505
8.11	BaseSequentialStreamVMT Struct Reference	505
8.11.1	Detailed Description	506
8.12	BlockDeviceInfo Struct Reference	506
8.12.1	Detailed Description	507
8.12.2	Field Documentation	507
8.12.2.1	blk_size	507
8.12.2.2	blk_num	507
8.13	CANConfig Struct Reference	507
8.13.1	Detailed Description	508
8.14	CANDriver Struct Reference	508
8.14.1	Detailed Description	509
8.14.2	Field Documentation	509
8.14.2.1	state	509
8.14.2.2	config	509
8.14.2.3	txqueue	509
8.14.2.4	rxqueue	509
8.14.2.5	rxfull_event	509
8.14.2.6	txempty_event	510
8.14.2.7	error_event	510
8.14.2.8	sleep_event	510
8.14.2.9	wakeup_event	510

8.15 CANRxFrame Struct Reference	510
8.15.1 Detailed Description	511
8.15.2 Field Documentation	511
8.15.2.1 FMI	511
8.15.2.2 TIME	511
8.15.2.3 DLC	511
8.15.2.4 RTR	511
8.15.2.5 IDE	512
8.15.2.6 SID	512
8.15.2.7 EID	512
8.15.2.8 data8	512
8.15.2.9 data16	512
8.15.2.10 data32	512
8.16 CANTxFrame Struct Reference	512
8.16.1 Detailed Description	513
8.16.2 Field Documentation	513
8.16.2.1 DLC	513
8.16.2.2 RTR	513
8.16.2.3 IDE	513
8.16.2.4 SID	513
8.16.2.5 EID	513
8.16.2.6 data8	514
8.16.2.7 data16	514
8.16.2.8 data32	514
8.17 cdc_linecoding_t Struct Reference	514
8.17.1 Detailed Description	514
8.18 DACConfig Struct Reference	514
8.18.1 Detailed Description	515
8.19 DACConversionGroup Struct Reference	515
8.19.1 Detailed Description	516
8.19.2 Field Documentation	516
8.19.2.1 num_channels	516
8.19.2.2 end_cb	516
8.19.2.3 error_cb	516
8.20 DACDriver Struct Reference	516
8.20.1 Detailed Description	517
8.20.2 Field Documentation	518
8.20.2.1 state	518
8.20.2.2 grpp	518
8.20.2.3 samples	518

8.20.2.4	depth	518
8.20.2.5	config	518
8.20.2.6	thread	518
8.20.2.7	mutex	518
8.21	EXTChannelConfig Struct Reference	518
8.21.1	Detailed Description	519
8.21.2	Field Documentation	519
8.21.2.1	mode	519
8.21.2.2	cb	519
8.22	EXTConfig Struct Reference	520
8.22.1	Detailed Description	520
8.22.2	Field Documentation	520
8.22.2.1	channels	520
8.23	EXTDriver Struct Reference	521
8.23.1	Detailed Description	521
8.23.2	Field Documentation	521
8.23.2.1	state	521
8.23.2.2	config	522
8.24	FileStream Struct Reference	522
8.24.1	Detailed Description	523
8.24.2	Field Documentation	523
8.24.2.1	vmt	523
8.25	FileStreamVMT Struct Reference	523
8.25.1	Detailed Description	524
8.26	GPTConfig Struct Reference	524
8.26.1	Detailed Description	525
8.26.2	Field Documentation	525
8.26.2.1	frequency	525
8.26.2.2	callback	525
8.27	GPTDriver Struct Reference	526
8.27.1	Detailed Description	526
8.27.2	Field Documentation	526
8.27.2.1	state	526
8.27.2.2	config	526
8.28	I2CConfig Struct Reference	526
8.28.1	Detailed Description	527
8.29	I2CDriver Struct Reference	527
8.29.1	Detailed Description	528
8.29.2	Field Documentation	528
8.29.2.1	state	528

8.29.2.2	config	528
8.29.2.3	errors	528
8.30	I2SConfig Struct Reference	528
8.30.1	Detailed Description	529
8.30.2	Field Documentation	529
8.30.2.1	tx_buffer	529
8.30.2.2	rx_buffer	530
8.30.2.3	size	530
8.30.2.4	end_cb	530
8.31	I2SDriver Struct Reference	530
8.31.1	Detailed Description	531
8.31.2	Field Documentation	531
8.31.2.1	state	531
8.31.2.2	config	531
8.32	ICUConfig Struct Reference	531
8.32.1	Detailed Description	532
8.32.2	Field Documentation	532
8.32.2.1	mode	532
8.32.2.2	frequency	532
8.32.2.3	width_cb	532
8.32.2.4	period_cb	532
8.32.2.5	overflow_cb	532
8.33	ICUDriver Struct Reference	532
8.33.1	Detailed Description	533
8.33.2	Field Documentation	533
8.33.2.1	state	533
8.33.2.2	config	533
8.34	io_buffers_queue Struct Reference	533
8.34.1	Detailed Description	535
8.34.2	Field Documentation	535
8.34.2.1	waiting	535
8.34.2.2	suspended	535
8.34.2.3	bcounter	535
8.34.2.4	bwrptr	535
8.34.2.5	brdptr	535
8.34.2.6	btot	535
8.34.2.7	bsize	535
8.34.2.8	bn	535
8.34.2.9	buffers	535
8.34.2.10	ptr	536

8.34.2.11 top	536
8.34.2.12 notify	536
8.34.2.13 link	536
8.35 io_queue Struct Reference	536
8.35.1 Detailed Description	537
8.35.2 Field Documentation	537
8.35.2.1 q_waiting	537
8.35.2.2 q_counter	537
8.35.2.3 q_buffer	537
8.35.2.4 q_top	537
8.35.2.5 q_wrptr	537
8.35.2.6 q_rdptr	537
8.35.2.7 q_notify	537
8.35.2.8 q_link	538
8.36 IOBus Struct Reference	538
8.36.1 Detailed Description	538
8.36.2 Field Documentation	538
8.36.2.1 portid	538
8.36.2.2 mask	538
8.36.2.3 offset	539
8.37 MACConfig Struct Reference	539
8.37.1 Detailed Description	539
8.37.2 Field Documentation	539
8.37.2.1 mac_address	539
8.38 MACDriver Struct Reference	539
8.38.1 Detailed Description	540
8.38.2 Field Documentation	540
8.38.2.1 state	540
8.38.2.2 config	540
8.38.2.3 tdqueue	541
8.38.2.4 rdqueue	541
8.38.2.5 rdevent	541
8.39 MACReceiveDescriptor Struct Reference	541
8.39.1 Detailed Description	541
8.39.2 Field Documentation	541
8.39.2.1 offset	541
8.39.2.2 size	542
8.40 MACTransmitDescriptor Struct Reference	542
8.40.1 Detailed Description	542
8.40.2 Field Documentation	542

8.40.2.1	offset	542
8.40.2.2	size	542
8.41	MMCCConfig Struct Reference	542
8.41.1	Detailed Description	543
8.41.2	Field Documentation	543
8.41.2.1	spip	543
8.41.2.2	lscfg	544
8.41.2.3	hscfg	544
8.42	MMCDriver Struct Reference	544
8.42.1	Detailed Description	545
8.42.2	Field Documentation	545
8.42.2.1	vmt	545
8.42.2.2	config	546
8.43	MMCDriverVMT Struct Reference	546
8.43.1	Detailed Description	547
8.44	MMCSDBlockDevice Struct Reference	547
8.44.1	Detailed Description	549
8.44.2	Field Documentation	549
8.44.2.1	vmt	549
8.45	MMCSDBlockDeviceVMT Struct Reference	549
8.45.1	Detailed Description	550
8.46	PALConfig Struct Reference	551
8.46.1	Detailed Description	551
8.47	PWMChannelConfig Struct Reference	551
8.47.1	Detailed Description	552
8.47.2	Field Documentation	552
8.47.2.1	mode	552
8.47.2.2	callback	552
8.48	PWMConfig Struct Reference	553
8.48.1	Detailed Description	554
8.48.2	Field Documentation	554
8.48.2.1	frequency	554
8.48.2.2	period	554
8.48.2.3	callback	554
8.48.2.4	channels	554
8.49	PWMDriver Struct Reference	554
8.49.1	Detailed Description	555
8.49.2	Field Documentation	555
8.49.2.1	state	556
8.49.2.2	config	556

8.49.2.3	period	556
8.49.2.4	enabled	556
8.49.2.5	channels	556
8.50	qspi_command_t Struct Reference	556
8.50.1	Detailed Description	556
8.51	QSPIDriver Struct Reference	556
8.51.1	Detailed Description	557
8.51.2	Field Documentation	557
8.51.2.1	end_cb	557
8.52	QSPIDriver Struct Reference	557
8.52.1	Detailed Description	558
8.52.2	Field Documentation	558
8.52.2.1	state	558
8.52.2.2	config	558
8.52.2.3	thread	558
8.52.2.4	mutex	559
8.53	RTCArm Struct Reference	559
8.53.1	Detailed Description	559
8.54	RTCDateTime Struct Reference	559
8.54.1	Detailed Description	560
8.54.2	Field Documentation	560
8.54.2.1	year	560
8.54.2.2	month	560
8.54.2.3	dstflag	560
8.54.2.4	dayofweek	560
8.54.2.5	day	560
8.54.2.6	millisecond	560
8.55	RTCDriver Struct Reference	561
8.55.1	Detailed Description	561
8.55.2	Field Documentation	561
8.55.2.1	vmt	561
8.56	RTCDriverVMT Struct Reference	561
8.56.1	Detailed Description	563
8.57	SDCConfig Struct Reference	563
8.57.1	Detailed Description	564
8.57.2	Field Documentation	564
8.57.2.1	scratchpad	564
8.57.2.2	bus_width	564
8.58	SDCDriver Struct Reference	564
8.58.1	Detailed Description	565

8.58.2	Field Documentation	565
8.58.2.1	vmt	565
8.58.2.2	config	566
8.58.2.3	cardmode	566
8.58.2.4	errors	566
8.58.2.5	rca	566
8.59	SDCDriverVMT Struct Reference	566
8.59.1	Detailed Description	567
8.60	SerialConfig Struct Reference	567
8.60.1	Detailed Description	568
8.60.2	Field Documentation	568
8.60.2.1	speed	568
8.61	SerialDriver Struct Reference	568
8.61.1	Detailed Description	569
8.61.2	Field Documentation	569
8.61.2.1	vmt	569
8.62	SerialDriverVMT Struct Reference	569
8.62.1	Detailed Description	571
8.63	SerialUSBConfig Struct Reference	571
8.63.1	Detailed Description	572
8.63.2	Field Documentation	572
8.63.2.1	usbp	572
8.63.2.2	bulk_in	572
8.63.2.3	bulk_out	572
8.63.2.4	int_in	572
8.64	SerialUSBDriver Struct Reference	572
8.64.1	Detailed Description	574
8.64.2	Field Documentation	574
8.64.2.1	vmt	574
8.65	SerialUSBDriverVMT Struct Reference	574
8.65.1	Detailed Description	576
8.66	SPICConfig Struct Reference	576
8.66.1	Detailed Description	576
8.66.2	Field Documentation	576
8.66.2.1	end_cb	576
8.67	SPIDriver Struct Reference	577
8.67.1	Detailed Description	577
8.67.2	Field Documentation	577
8.67.2.1	state	577
8.67.2.2	config	578

8.67.2.3	thread	578
8.67.2.4	mutex	578
8.68	UARTConfig Struct Reference	578
8.68.1	Detailed Description	579
8.68.2	Field Documentation	579
8.68.2.1	txend1_cb	579
8.68.2.2	txend2_cb	579
8.68.2.3	rxend_cb	579
8.68.2.4	rxchar_cb	579
8.68.2.5	rxerr_cb	579
8.69	UARTDriver Struct Reference	579
8.69.1	Detailed Description	580
8.69.2	Field Documentation	581
8.69.2.1	state	581
8.69.2.2	txstate	581
8.69.2.3	rxstate	581
8.69.2.4	config	581
8.69.2.5	early	581
8.69.2.6	threadrx	581
8.69.2.7	threadtx	581
8.69.2.8	mutex	581
8.70	unpacked_mmc_cid_t Struct Reference	581
8.70.1	Detailed Description	582
8.71	unpacked_mmc_csd_t Struct Reference	582
8.71.1	Detailed Description	582
8.72	unpacked_sdc_cid_t Struct Reference	583
8.72.1	Detailed Description	583
8.73	unpacked_sdc_csd_10_t Struct Reference	583
8.73.1	Detailed Description	584
8.74	unpacked_sdc_csd_20_t Struct Reference	584
8.74.1	Detailed Description	585
8.75	USBConfig Struct Reference	585
8.75.1	Detailed Description	587
8.75.2	Field Documentation	587
8.75.2.1	event_cb	587
8.75.2.2	get_descriptor_cb	587
8.75.2.3	requests_hook_cb	587
8.75.2.4	sof_cb	587
8.76	USBDescriptor Struct Reference	587
8.76.1	Detailed Description	588

8.76.2	Field Documentation	588
8.76.2.1	ud_size	588
8.76.2.2	ud_string	588
8.77	USBDriver Struct Reference	588
8.77.1	Detailed Description	589
8.77.2	Field Documentation	589
8.77.2.1	state	589
8.77.2.2	config	589
8.77.2.3	transmitting	589
8.77.2.4	receiving	589
8.77.2.5	epc	590
8.77.2.6	in_params	590
8.77.2.7	out_params	590
8.77.2.8	ep0state	590
8.77.2.9	ep0next	590
8.77.2.10	ep0n	590
8.77.2.11	ep0endcb	590
8.77.2.12	setup	590
8.77.2.13	status	590
8.77.2.14	address	590
8.77.2.15	configuration	590
8.77.2.16	saved_state	591
8.78	USBEndpointConfig Struct Reference	591
8.78.1	Detailed Description	593
8.78.2	Field Documentation	593
8.78.2.1	ep_mode	593
8.78.2.2	setup_cb	593
8.78.2.3	in_cb	593
8.78.2.4	out_cb	593
8.78.2.5	in_maxsize	594
8.78.2.6	out_maxsize	594
8.78.2.7	in_state	594
8.78.2.8	out_state	594
8.79	USBInEndpointState Struct Reference	594
8.79.1	Detailed Description	595
8.79.2	Field Documentation	595
8.79.2.1	txsize	595
8.79.2.2	txcnt	595
8.79.2.3	txbuf	595
8.79.2.4	thread	595

8.80	USBOutEndpointState Struct Reference	595
8.80.1	Detailed Description	596
8.80.2	Field Documentation	596
8.80.2.1	rxsize	596
8.80.2.2	rxcnt	596
8.80.2.3	rxbuf	596
8.80.2.4	thread	596
8.81	WDGConfig Struct Reference	596
8.81.1	Detailed Description	597
8.82	WDGDriver Struct Reference	597
8.82.1	Detailed Description	597
8.82.2	Field Documentation	597
8.82.2.1	state	597
8.82.2.2	config	598
9	File Documentation	599
9.1	hal.c File Reference	599
9.1.1	Detailed Description	599
9.2	hal.h File Reference	599
9.2.1	Detailed Description	601
9.3	hal_adc.c File Reference	601
9.3.1	Detailed Description	601
9.4	hal_adc.h File Reference	602
9.4.1	Detailed Description	603
9.5	hal_adc_ild.c File Reference	603
9.5.1	Detailed Description	603
9.6	hal_adc_ild.h File Reference	603
9.6.1	Detailed Description	604
9.7	hal_buffers.c File Reference	605
9.7.1	Detailed Description	606
9.8	hal_buffers.h File Reference	606
9.8.1	Detailed Description	608
9.9	hal_can.c File Reference	608
9.9.1	Detailed Description	608
9.10	hal_can.h File Reference	608
9.10.1	Detailed Description	610
9.11	hal_can_ild.c File Reference	610
9.11.1	Detailed Description	610
9.12	hal_can_ild.h File Reference	610
9.12.1	Detailed Description	612

9.13	hal_channels.h File Reference	612
9.13.1	Detailed Description	613
9.14	hal_dac.c File Reference	613
9.14.1	Detailed Description	614
9.15	hal_dac.h File Reference	614
9.15.1	Detailed Description	615
9.16	hal_dac_ild.c File Reference	615
9.16.1	Detailed Description	615
9.17	hal_dac_ild.h File Reference	616
9.17.1	Detailed Description	617
9.18	hal_ext.c File Reference	617
9.18.1	Detailed Description	617
9.19	hal_ext.h File Reference	617
9.19.1	Detailed Description	618
9.20	hal_ext_ild.c File Reference	618
9.20.1	Detailed Description	619
9.21	hal_ext_ild.h File Reference	619
9.21.1	Detailed Description	620
9.22	hal_files.h File Reference	620
9.22.1	Detailed Description	621
9.23	hal_gpt.c File Reference	621
9.23.1	Detailed Description	622
9.24	hal_gpt.h File Reference	622
9.24.1	Detailed Description	623
9.25	hal_gpt_ild.c File Reference	623
9.25.1	Detailed Description	623
9.26	hal_gpt_ild.h File Reference	624
9.26.1	Detailed Description	624
9.27	hal_i2c.c File Reference	625
9.27.1	Detailed Description	625
9.28	hal_i2c.h File Reference	625
9.28.1	Detailed Description	626
9.29	hal_i2c_ild.c File Reference	627
9.29.1	Detailed Description	627
9.30	hal_i2c_ild.h File Reference	627
9.30.1	Detailed Description	628
9.31	hal_i2s.c File Reference	628
9.31.1	Detailed Description	629
9.32	hal_i2s.h File Reference	629
9.32.1	Detailed Description	629

9.33	hal_i2s_lld.c File Reference	630
9.33.1	Detailed Description	630
9.34	hal_i2s_lld.h File Reference	630
9.34.1	Detailed Description	631
9.35	hal_icu.c File Reference	631
9.35.1	Detailed Description	632
9.36	hal_icu.h File Reference	632
9.36.1	Detailed Description	633
9.37	hal_icu_lld.c File Reference	633
9.37.1	Detailed Description	634
9.38	hal_icu_lld.h File Reference	634
9.38.1	Detailed Description	635
9.39	hal_ioblock.h File Reference	635
9.39.1	Detailed Description	636
9.40	hal_lld.c File Reference	636
9.40.1	Detailed Description	636
9.41	hal_lld.h File Reference	636
9.41.1	Detailed Description	636
9.42	hal_mac.c File Reference	636
9.42.1	Detailed Description	637
9.43	hal_mac.h File Reference	637
9.43.1	Detailed Description	638
9.44	hal_mac_lld.c File Reference	638
9.44.1	Detailed Description	639
9.45	hal_mac_lld.h File Reference	639
9.45.1	Detailed Description	640
9.46	hal_mii.h File Reference	640
9.46.1	Detailed Description	642
9.47	hal_mmc_spi.c File Reference	642
9.47.1	Detailed Description	644
9.48	hal_mmc_spi.h File Reference	644
9.48.1	Detailed Description	645
9.49	hal_mmcscd.c File Reference	645
9.49.1	Detailed Description	645
9.50	hal_mmcscd.h File Reference	646
9.50.1	Detailed Description	650
9.51	hal_pal.c File Reference	650
9.51.1	Detailed Description	650
9.52	hal_pal.h File Reference	650
9.52.1	Detailed Description	652

9.53	hal_pal_lld.c File Reference	653
9.53.1	Detailed Description	653
9.54	hal_pal_lld.h File Reference	653
9.54.1	Detailed Description	655
9.55	hal_pwm.c File Reference	655
9.55.1	Detailed Description	655
9.56	hal_pwm.h File Reference	655
9.56.1	Detailed Description	657
9.57	hal_pwm_lld.c File Reference	657
9.57.1	Detailed Description	658
9.58	hal_pwm_lld.h File Reference	658
9.58.1	Detailed Description	659
9.59	hal_qspi.c File Reference	659
9.59.1	Detailed Description	660
9.60	hal_qspi.h File Reference	660
9.60.1	Detailed Description	662
9.61	hal_qspi_lld.c File Reference	662
9.61.1	Detailed Description	663
9.62	hal_qspi_lld.h File Reference	663
9.62.1	Detailed Description	664
9.63	hal_queues.c File Reference	664
9.63.1	Detailed Description	665
9.64	hal_queues.h File Reference	665
9.64.1	Detailed Description	666
9.65	hal_rtc.c File Reference	666
9.65.1	Detailed Description	667
9.66	hal_rtc.h File Reference	667
9.66.1	Detailed Description	668
9.67	hal_rtc_lld.c File Reference	668
9.67.1	Detailed Description	669
9.68	hal_rtc_lld.h File Reference	669
9.68.1	Detailed Description	670
9.69	hal_sdc.c File Reference	670
9.69.1	Detailed Description	671
9.70	hal_sdc.h File Reference	672
9.70.1	Detailed Description	673
9.71	hal_sdc_lld.c File Reference	673
9.71.1	Detailed Description	674
9.72	hal_sdc_lld.h File Reference	674
9.72.1	Detailed Description	675

9.73	hal_serial.c File Reference	675
9.73.1	Detailed Description	676
9.74	hal_serial.h File Reference	676
9.74.1	Detailed Description	678
9.75	hal_serial_ild.c File Reference	678
9.75.1	Detailed Description	678
9.76	hal_serial_ild.h File Reference	678
9.76.1	Detailed Description	679
9.77	hal_serial_usb.c File Reference	679
9.77.1	Detailed Description	680
9.78	hal_serial_usb.h File Reference	680
9.78.1	Detailed Description	681
9.79	hal_spi.c File Reference	681
9.79.1	Detailed Description	682
9.80	hal_spi.h File Reference	682
9.80.1	Detailed Description	684
9.81	hal_spi_ild.c File Reference	684
9.81.1	Detailed Description	684
9.82	hal_spi_ild.h File Reference	684
9.82.1	Detailed Description	685
9.83	hal_st.c File Reference	686
9.83.1	Detailed Description	686
9.84	hal_st.h File Reference	686
9.84.1	Detailed Description	687
9.85	hal_st_ild.c File Reference	687
9.85.1	Detailed Description	687
9.86	hal_st_ild.h File Reference	687
9.86.1	Detailed Description	687
9.87	hal_streams.h File Reference	688
9.87.1	Detailed Description	688
9.88	hal_uart.c File Reference	688
9.88.1	Detailed Description	689
9.89	hal_uart.h File Reference	689
9.89.1	Detailed Description	691
9.90	hal_uart_ild.c File Reference	691
9.90.1	Detailed Description	692
9.91	hal_uart_ild.h File Reference	692
9.91.1	Detailed Description	693
9.92	hal_usb.c File Reference	693
9.92.1	Detailed Description	694

9.93	hal_usb.h File Reference	694
9.93.1	Detailed Description	697
9.94	hal_usb_cdc.h File Reference	697
9.94.1	Detailed Description	698
9.95	hal_usb_llid.c File Reference	698
9.95.1	Detailed Description	699
9.95.2	Variable Documentation	699
9.95.2.1	in	699
9.95.2.2	out	700
9.96	hal_usb_llid.h File Reference	700
9.96.1	Detailed Description	701
9.97	hal_wdg.c File Reference	701
9.97.1	Detailed Description	702
9.98	hal_wdg.h File Reference	702
9.98.1	Detailed Description	702
9.99	halconf.h File Reference	702
9.99.1	Detailed Description	705
Index		707

Chapter 1

ChibiOS/HAL

1.1 Copyright

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

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

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

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

1.2 Introduction

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

1.3 Related Documents

- ChibiOS/HAL General Architecture

Chapter 2

Deprecated List

Global [canReceive](#) (canp, mailbox, crfp, timeout)

Global [canTransmit](#) (canp, mailbox, ctfp, timeout)

Global [sdGetWouldBlock](#) ([SerialDriver](#) *sdp)

Global [sdPutWouldBlock](#) ([SerialDriver](#) *sdp)

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

HAL	203
Configuration	206
Normal Drivers	213
ADC Driver	17
CAN Driver	33
DAC Driver	47
EXT Driver	63
GPT Driver	75
HAL Driver	89
I2C Driver	150
I2S Driver	163
ICU Driver	171
MAC Driver	188
PAL Driver	263
PWM Driver	291
QSPI Driver	311
RTC Driver	327
SDC Driver	337
Serial Driver	365
SPI Driver	391
ST Driver	412
UART Driver	418
USB Driver	444
WDG Driver	484
Complex Drivers	214
MMC over SPI Driver	230
Serial over USB Driver	380
Interfaces	215
Abstract NOR Flash Class	73
Abstract I/O Channel	112
Abstract Files	119
Abstract I/O Block Device	125
Abstract Streams	147
Inner Code	216
I/O Buffers Queues	92
I/O Bytes Queues	132
MMC/SD Block Device	252

Support Code	217
MII/RMII Header	220
USB CDC Header	482
OSAL	218

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

ADCConfig	489
ADCConversionGroup	490
ADCDriver	491
BaseBlockDevice	497
MMCSDBlockDevice	547
MMCDriver	544
BaseBlockDeviceVMT	499
MMCSDBlockDeviceVMT	549
MMCDriverVMT	546
SDCDriverVMT	566
BaseSequentialStream	504
BaseChannel	500
BaseAsynchronousChannel	493
SerialDriver	568
SerialUSBDriver	572
FileStream	522
RTCDriverVMT	561
BaseSequentialStreamVMT	505
BaseChannelVMT	502
BaseAsynchronousChannelVMT	495
SerialDriverVMT	569
SerialUSBDriverVMT	574
FileStreamVMT	523
BlockDeviceInfo	506
CANConfig	507
CANDriver	508
CANRxFrame	510
CANTxFrame	512
cdc_linecoding_t	514
DACConfig	514
DACConversionGroup	515
DACDriver	516
EXTChannelConfig	518
EXTConfig	520
EXTDriver	521
GPTConfig	524

GPTDriver	526
I2CConfig	526
I2CDriver	527
I2SConfig	528
I2SDriver	530
ICUConfig	531
ICUDriver	532
io_buffers_queue	533
io_queue	536
IOBus	538
MACConfig	539
MACDriver	539
MACReceiveDescriptor	541
MACTransmitDescriptor	542
MMCCConfig	542
PALConfig	551
PWMChannelConfig	551
PWMConfig	553
PWMDriver	554
qspi_command_t	556
QSPICConfig	556
QSPIDriver	557
RTCAlarm	559
RTCDateTime	559
RTCDriver	561
SDCConfig	563
SDCDriver	564
SerialConfig	567
SerialUSBConfig	571
SPICConfig	576
SPIDriver	577
UARTConfig	578
UARTDriver	579
unpacked_mmc_cid_t	581
unpacked_mmc_csd_t	582
unpacked_sdc_cid_t	583
unpacked_sdc_csd_10_t	583
unpacked_sdc_csd_20_t	584
USBConfig	585
USBDescriptor	587
USBDriver	588
USBEndpointConfig	591
USBInEndpointState	594
USBOutEndpointState	595
WDGConfig	596
WDGDriver	597

Chapter 5

Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

ADCConfig	Driver configuration structure	489
ADCConversionGroup	Conversion group configuration structure	490
ADCDriver	Structure representing an ADC driver	491
BaseAsynchronousChannel	Base asynchronous channel class	493
BaseAsynchronousChannelVMT	BaseAsynchronousChannel virtual methods table	495
BaseBlockDevice	Base block device class	497
BaseBlockDeviceVMT	BaseBlockDevice virtual methods table	499
BaseChannel	Base channel class	500
BaseChannelVMT	BaseChannel virtual methods table	502
BaseSequentialStream	Base stream class	504
BaseSequentialStreamVMT	BaseSequentialStream virtual methods table	505
BlockDeviceInfo	Block device info	506
CANConfig	Driver configuration structure	507
CANDriver	Structure representing an CAN driver	508
CANRxFrame	CAN received frame	510
CANTxFrame	CAN transmission frame	512
cdc_linecoding_t	Type of Line Coding structure	514
DACConfig	Driver configuration structure	514
DACConversionGroup	DAC Conversion group structure	515

DACDriver	Structure representing a DAC driver	516
EXTChannelConfig	Channel configuration structure	518
EXTConfig	Driver configuration structure	520
EXTDriver	Structure representing an EXT driver	521
FileStream	Base file stream class	522
FileStreamVMT	FileStream virtual methods table	523
GPTConfig	Driver configuration structure	524
GPTDriver	Structure representing a GPT driver	526
I2CConfig	Type of I2C driver configuration structure	526
I2CDriver	Structure representing an I2C driver	527
I2SConfig	Driver configuration structure	528
I2SDriver	Structure representing an I2S driver	530
ICUConfig	Driver configuration structure	531
ICUDriver	Structure representing an ICU driver	532
io_buffers_queue	Structure of a generic buffers queue	533
io_queue	Generic I/O queue structure	536
IOBus	I/O bus descriptor	538
MACConfig	Driver configuration structure	539
MACDriver	Structure representing a MAC driver	539
MACReceiveDescriptor	Structure representing a receive descriptor	541
MACTransmitDescriptor	Structure representing a transmit descriptor	542
MMCCConfig	MMC/SD over SPI driver configuration structure	542
MMCDriver	Structure representing a MMC/SD over SPI driver	544
MMCDriverVMT	MMCDriver virtual methods table	546
MMCSDBlockDevice	MCC/SD block device class	547
MMCSDBlockDeviceVMT	MMCSDBlockDevice virtual methods table	549
PALConfig	Generic I/O ports static initializer	551
PWMChannelConfig	Type of a PWM driver channel configuration structure	551
PWMConfig	Type of a PWM driver configuration structure	553

PWMDriver	Structure representing a PWM driver	554
qspi_command_t	Type of a QSPI command descriptor	556
QSPIDriver	Driver configuration structure	556
QSPIDriver	Structure representing an QSPI driver	557
RTCAAlarm	Type of a structure representing an RTC alarm time stamp	559
RTCDatetime	Type of a structure representing an RTC date/time stamp	559
RTCDriver	Structure representing an RTC driver	561
RTCDriverVMT	RTCDriver virtual methods table	561
SDCConfig	Driver configuration structure	563
SDCDriver	Structure representing an SDC driver	564
SDCDriverVMT	SDCDriver virtual methods table	566
SerialConfig	PLATFORM Serial Driver configuration structure	567
SerialDriver	Full duplex serial driver class	568
SerialDriverVMT	SerialDriver virtual methods table	569
SerialUSBConfig	Serial over USB Driver configuration structure	571
SerialUSBDriver	Full duplex serial driver class	572
SerialUSBDriverVMT	SerialDriver virtual methods table	574
SPIDriver	Driver configuration structure	576
SPIDriver	Structure representing an SPI driver	577
UARTConfig	Driver configuration structure	578
UARTDriver	Structure representing an UART driver	579
unpacked_mmc_cid_t	Unpacked CID register from MMC	581
unpacked_mmc_csd_t	Unpacked CSD register from MMC	582
unpacked_sdc_cid_t	Unpacked CID register from SDC	583
unpacked_sdc_csd_10_t	Unpacked CSD v1.0 register from SDC	583
unpacked_sdc_csd_20_t	Unpacked CSD v2.0 register from SDC	584
USBConfig	Type of an USB driver configuration structure	585
USBDescriptor	Type of an USB descriptor	587
USBDriver	Structure representing an USB driver	588

USBEndpointConfig	
Type of an USB endpoint configuration structure	591
USBInEndpointState	
Type of an IN endpoint state structure	594
USBOutEndpointState	
Type of an OUT endpoint state structure	595
WDGConfig	
Driver configuration structure	596
WDGDriver	
Structure representing an WDG driver	597

Chapter 6

File Index

6.1 File List

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

hal.c	HAL subsystem code	599
hal.h	HAL subsystem header	599
hal_adc.c	ADC Driver code	601
hal_adc.h	ADC Driver macros and structures	602
hal_adc_ild.c	PLATFORM ADC subsystem low level driver source	603
hal_adc_ild.h	PLATFORM ADC subsystem low level driver header	603
hal_buffers.c	I/O Buffers code	605
hal_buffers.h	I/O Buffers macros and structures	606
hal_can.c	CAN Driver code	608
hal_can.h	CAN Driver macros and structures	608
hal_can_ild.c	PLATFORM CAN subsystem low level driver source	610
hal_can_ild.h	PLATFORM CAN subsystem low level driver header	610
hal_channels.h	I/O channels access	612
hal_dac.c	DAC Driver code	613
hal_dac.h	DAC Driver macros and structures	614
hal_dac_ild.c	PLATFORM DAC subsystem low level driver source	615
hal_dac_ild.h	PLATFORM DAC subsystem low level driver header	616
hal_ext.c	EXT Driver code	617
hal_ext.h	EXT Driver macros and structures	617

hal_ext_lld.c	PLATFORM EXT subsystem low level driver source	618
hal_ext_lld.h	PLATFORM EXT subsystem low level driver header	619
hal_files.h	Data files	620
hal_gpt.c	GPT Driver code	621
hal_gpt.h	GPT Driver macros and structures	622
hal_gpt_lld.c	PLATFORM GPT subsystem low level driver source	623
hal_gpt_lld.h	PLATFORM GPT subsystem low level driver header	624
hal_i2c.c	I2C Driver code	625
hal_i2c.h	I2C Driver macros and structures	625
hal_i2c_lld.c	PLATFORM I2C subsystem low level driver source	627
hal_i2c_lld.h	PLATFORM I2C subsystem low level driver header	627
hal_i2s.c	I2S Driver code	628
hal_i2s.h	I2S Driver macros and structures	629
hal_i2s_lld.c	PLATFORM I2S subsystem low level driver source	630
hal_i2s_lld.h	PLATFORM I2S subsystem low level driver header	630
hal_icu.c	ICU Driver code	631
hal_icu.h	ICU Driver macros and structures	632
hal_icu_lld.c	PLATFORM ADC subsystem low level driver source	633
hal_icu_lld.h	PLATFORM ICU subsystem low level driver header	634
hal_ioblock.h	I/O block devices access	635
hal_lld.c	PLATFORM HAL subsystem low level driver source	636
hal_lld.h	PLATFORM HAL subsystem low level driver header	636
hal_mac.c	MAC Driver code	636
hal_mac.h	MAC Driver macros and structures	637
hal_mac_lld.c	PLATFORM MAC subsystem low level driver source	638
hal_mac_lld.h	PLATFORM MAC subsystem low level driver header	639
hal_mii.h	MII macros and structures	640
hal_mmc_spi.c	MMC over SPI driver code	642
hal_mmc_spi.h	MMC over SPI driver header	644

hal_mmcsd.c	MMC/SD cards common code	645
hal_mmcsd.h	MMC/SD cards common header	646
hal_pal.c	I/O Ports Abstraction Layer code	650
hal_pal.h	I/O Ports Abstraction Layer macros, types and structures	650
hal_pal_lld.c	PLATFORM PAL subsystem low level driver source	653
hal_pal_lld.h	PLATFORM PAL subsystem low level driver header	653
hal_pwm.c	PWM Driver code	655
hal_pwm.h	PWM Driver macros and structures	655
hal_pwm_lld.c	PLATFORM PWM subsystem low level driver source	657
hal_pwm_lld.h	PLATFORM PWM subsystem low level driver header	658
hal_qspi.c	QSPI Driver code	659
hal_qspi.h	QSPI Driver macros and structures	660
hal_qspi_lld.c	PLATFORM QSPI subsystem low level driver source	662
hal_qspi_lld.h	PLATFORM QSPI subsystem low level driver header	663
hal_queues.c	I/O Queues code	664
hal_queues.h	I/O Queues macros and structures	665
hal_rtc.c	RTC Driver code	666
hal_rtc.h	RTC Driver macros and structures	667
hal_rtc_lld.c	PLATFORM RTC subsystem low level driver source	668
hal_rtc_lld.h	PLATFORM RTC subsystem low level driver header	669
hal_sdc.c	SDC Driver code	670
hal_sdc.h	SDC Driver macros and structures	672
hal_sdc_lld.c	PLATFORM SDC subsystem low level driver source	673
hal_sdc_lld.h	PLATFORM SDC subsystem low level driver header	674
hal_serial.c	Serial Driver code	675
hal_serial.h	Serial Driver macros and structures	676
hal_serial_lld.c	PLATFORM serial subsystem low level driver source	678
hal_serial_lld.h	PLATFORM serial subsystem low level driver header	678
hal_serial_usb.c	Serial over USB Driver code	679

hal_serial_usb.h	Serial over USB Driver macros and structures	680
hal_spi.c	SPI Driver code	681
hal_spi.h	SPI Driver macros and structures	682
hal_spi_lld.c	PLATFORM SPI subsystem low level driver source	684
hal_spi_lld.h	PLATFORM SPI subsystem low level driver header	684
hal_st.c	ST Driver code	686
hal_st.h	ST Driver macros and structures	686
hal_st_lld.c	PLATFORM ST subsystem low level driver source	687
hal_st_lld.h	PLATFORM ST subsystem low level driver header	687
hal_streams.h	Data streams	688
hal_uart.c	UART Driver code	688
hal_uart.h	UART Driver macros and structures	689
hal_uart_lld.c	PLATFORM UART subsystem low level driver source	691
hal_uart_lld.h	PLATFORM UART subsystem low level driver header	692
hal_usb.c	USB Driver code	693
hal_usb.h	USB Driver macros and structures	694
hal_usb_cdc.h	USB CDC macros and structures	697
hal_usb_lld.c	PLATFORM USB subsystem low level driver source	698
hal_usb_lld.h	PLATFORM USB subsystem low level driver header	700
hal_wdg.c	WDG Driver code	701
hal_wdg.h	WDG Driver macros and structures	702
halconf.h	HAL configuration header	702

Chapter 7

Module Documentation

7.1 ADC Driver

Generic ADC Driver.

7.1.1 Detailed Description

Generic ADC Driver.

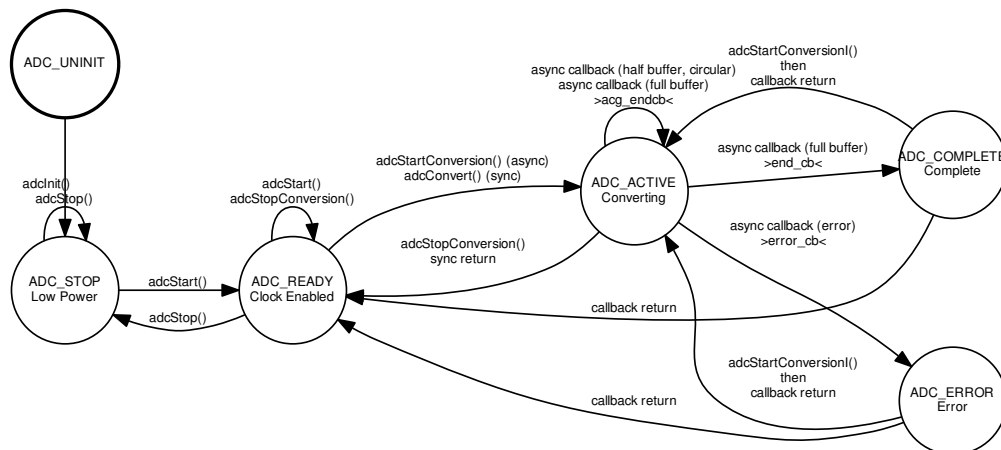
This module implements a generic ADC (Analog to Digital Converter) driver supporting a variety of buffer and conversion modes.

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 $S(CG) \times N$ 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) osalThreadResumeI(&(adcp)->thread, MSG_RESET)`
Resumes a thread waiting for a conversion completion.
- `#define _adc_reset_s(adcp) osalThreadResumeS(&(adcp)->thread, MSG_RESET)`
Resumes a thread waiting for a conversion completion.
- `#define _adc_wakeup_isr(adcp)`
Wakes up the waiting thread.
- `#define _adc_timeout_isr(adcp)`
Wakes up the waiting thread with a timeout message.
- `#define _adc_isr_half_code(adcp)`
Common ISR code, half buffer event.
- `#define _adc_isr_full_code(adcp)`
Common ISR code, full buffer event.
- `#define _adc_isr_error_code(adcp, err)`
Common ISR code, error event.

PLATFORM configuration options

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

Typedefs

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

Enumerations

Variables

- [ADCDriver](#) [ADC1](#)
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) osalThreadResumeI(&(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.

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

Common ISR code, error event.

This code handles the portable part of the ISR code:

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

Note

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

Parameters

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 ADCDriver 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 error.

7.1.6.2 enum `adcerror_t`

Possible ADC failure causes.

Note

Error codes are architecture dependent and should not be 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.

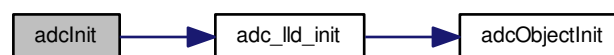
Note

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.

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.

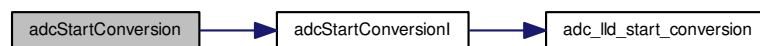
Parameters

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

Starts an ADC conversion.

Starts an asynchronous conversion operation.

Postcondition

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

Note

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

Parameters

in	<i>adcp</i>	pointer to the ADCDriver object
in	<i>grpp</i>	pointer to a ADCCConversionGroup 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	<i>adcp</i>	pointer to the ADCDriver object
in	<i>grpp</i>	pointer to a ADCCConversionGroup 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.

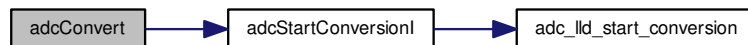
Return values

<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:

**7.1.7.10 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.

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.

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

Function Class:

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

7.1.7.15 void adc_llid_start_conversion (ADCDriver * *adcp*)

Starts an ADC conversion.

Parameters

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

Function Class:

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

7.1.7.16 void adc_llid_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.

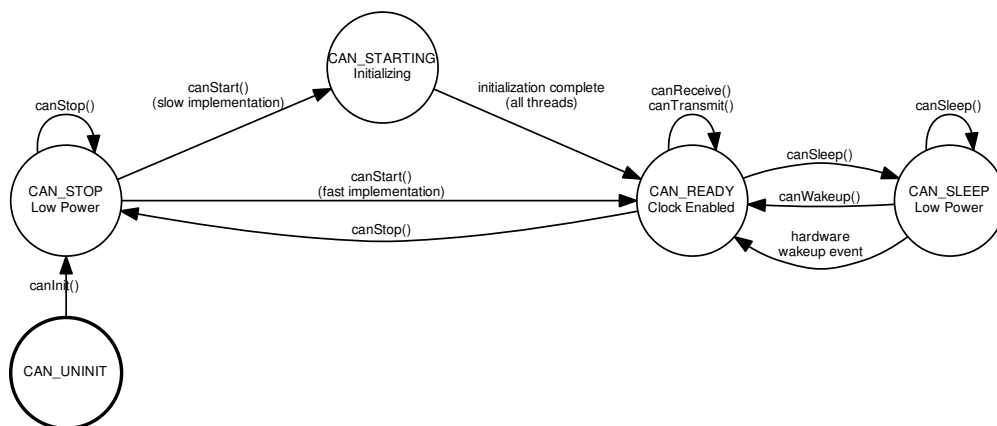
This module implements a generic CAN (Controller Area Network) driver allowing the exchange of information at frame level.

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 1U`

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

CAN configuration options

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

Macro Functions

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

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 *canp)
Initializes the standard part of a CANDriver structure.
- void `canStart` (CANDriver *canp, const CANConfig *config)
Configures and activates the CAN peripheral.
- void `canStop` (CANDriver *canp)
Deactivates the CAN peripheral.
- bool `canTryTransmitI` (CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp)
Can frame transmission attempt.
- bool `canTryReceiveI` (CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp)
Can frame receive attempt.
- msg_t `canTransmitTimeout` (CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp, systime_t timeout)
Can frame transmission.
- msg_t `canReceiveTimeout` (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_llc_init` (void)
Low level CAN driver initialization.
- void `can_llc_start` (CANDriver *canp)
Configures and activates the CAN peripheral.
- void `can_llc_stop` (CANDriver *canp)
Deactivates the CAN peripheral.
- bool `can_llc_is_tx_empty` (CANDriver *canp, canmbx_t mailbox)
Determines whether a frame can be transmitted.
- void `can_llc_transmit` (CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp)
Inserts a frame into the transmit queue.
- bool `can_llc_is_rx_nonempty` (CANDriver *canp, canmbx_t mailbox)
Determines whether a frame has been received.
- void `can_llc_receive` (CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp)
Receives a frame from the input queue.
- void `can_llc_sleep` (CANDriver *canp)
Enters the sleep mode.
- void `can_llc_wakeup` (CANDriver *canp)
Enforces leaving the sleep mode.

Enumerations

Variables

- CANDriver `CAND1`
CAN1 driver identifier.

7.2.3 Macro Definition Documentation

7.2.3.1 `#define CAN_LIMIT_WARNING 1U`

Errors rate warning.

7.2.3.2 `#define CAN_LIMIT_ERROR 2U`

Errors rate error.

7.2.3.3 `#define CAN_BUS_OFF_ERROR 4U`

Bus off condition reached.

7.2.3.4 `#define CAN_FRAMING_ERROR 8U`

Framing error of some kind on the CAN bus.

7.2.3.5 `#define CAN_OVERFLOW_ERROR 16U`

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) (1U << ((mbx) - 1U))`

Converts a mailbox index to a bit mask.

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

Legacy name for `canTransmitTimeout()`.

Deprecated

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

Legacy name for `canReceiveTimeout()`.

Deprecated

7.2.3.11 `#define CAN_TX_MAILBOXES 1`

Number of transmit mailboxes.

7.2.3.12 `#define CAN_RX_MAILBOXES 1`

Number of receive mailboxes.

7.2.3.13 `#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.

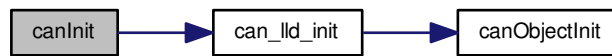
Note

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.

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 <code>NULL</code> .

Function Class:

Normal API, 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 bool canTryTransmitl (CANDriver * canp, canmbx_t mailbox, const CANTxFrame * ctfp)

Can frame transmission attempt.

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

Parameters

in	<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

Returns

The operation result.

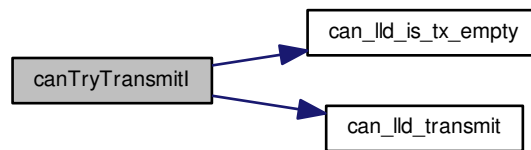
Return values

<i>false</i>	Frame transmitted.
<i>true</i>	Mailbox 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.

Here is the call graph for this function:



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

Can frame receive attempt.

The function tries to fetch a frame from a mailbox.

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

Returns

The operation result.

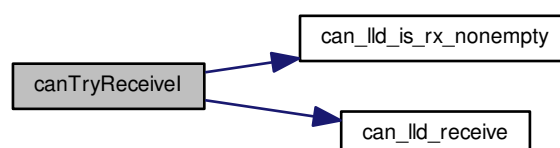
Return values

<i>false</i>	Frame fetched.
<i>true</i>	Mailbox 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:



7.2.6.7 `msg_t canTransmitTimeout (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, <code>CAN_ANY_MAILBOX</code> 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"> <code>TIME_IMMEDIATE</code> immediate timeout. <code>TIME_INFINITE</code> no timeout.

Returns

The operation result.

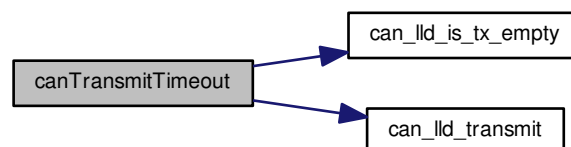
Return values

<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:



7.2.6.8 `msg_t canReceiveTimeout (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, CAN_ANY_MAILBOX 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"> <i>TIME_IMMEDIATE</i> immediate timeout (useful in an event driven scenario where a thread never blocks for I/O). <i>TIME_INFINITE</i> no timeout.

Returns

The operation result.

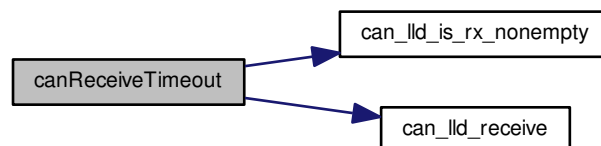
Return values

<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:



7.2.6.9 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:

**7.2.6.10 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:

**7.2.6.11 void can_llid_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:



7.2.6.12 void can_ild_start (CANDriver * canp)

Configures and activates 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.

7.2.6.13 void can_ild_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.

7.2.6.14 bool can_ild_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.

7.2.6.15 void can_ild_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.

7.2.6.16 bool can_ild_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.

7.2.6.17 void can_ild_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.

7.2.6.18 void can_ild_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.

7.2.6.19 void can_ild_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, [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, [dacsample_t](#) *samples, size_t depth)
Starts a DAC conversion.
- void [dacStartConversionI](#) ([DACDriver](#) *dACP, const [DACConversionGroup](#) *grpp, [dacsample_t](#) *samples, size_t depth)
Starts a DAC conversion.
- void [dacStopConversion](#) ([DACDriver](#) *dACP)
Stops an ongoing conversion.
- void [dacStopConversionI](#) ([DACDriver](#) *dACP)
Stops an ongoing conversion.
- msg_t [dacConvert](#) ([DACDriver](#) *dACP, const [DACConversionGroup](#) *grpp, [dacsample_t](#) *samples, size_t depth)
Performs a DAC conversion.
- void [dacAcquireBus](#) ([DACDriver](#) *dACP)
Gains exclusive access to the DAC bus.
- void [dacReleaseBus](#) ([DACDriver](#) *dACP)
Releases exclusive access to the DAC bus.

- void `dac_ll_init` (void)
Low level DAC driver initialization.
- void `dac_ll_start` (DACDriver *dacp)
Configures and activates the DAC peripheral.
- void `dac_ll_stop` (DACDriver *dacp)
Deactivates the DAC peripheral.
- void `dac_ll_put_channel` (DACDriver *dacp, dacchannel_t channel, dacsample_t sample)
Outputs a value directly on a DAC channel.
- void `dac_ll_start_conversion` (DACDriver *dacp)
Starts a DAC conversion.
- void `dac_ll_stop_conversion` (DACDriver *dacp)
Stops an ongoing conversion.

Enumerations

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, 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 <code>buffer</code> 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 be 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.

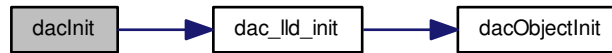
Note

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.

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*, 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.

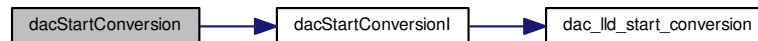
Parameters

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*, 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 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:

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 dacStopConversionI (DACDriver * *dacp*)

Stops an ongoing conversion.

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

Parameters

in	<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.

Here is the call graph for this function:



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

Performs a DAC conversion.

Performs a synchronous conversion operation.

Note

The buffer is organized as a matrix of M*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.

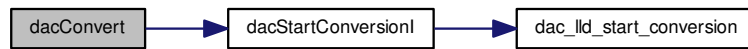
Return values

<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 DACDriver object
----	-------------	-------------------------------------------------

Function Class:

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

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 DACDriver object
----	-------------	-------------------------------------------------

Function Class:

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

7.3.5.13 void dac_llid_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_ild_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_ild_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_ild_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_llid_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 <code>DACDriver</code> object
----	-------------	----------------------------------------------

Function Class:

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

7.3.5.18 void dac_llid_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.

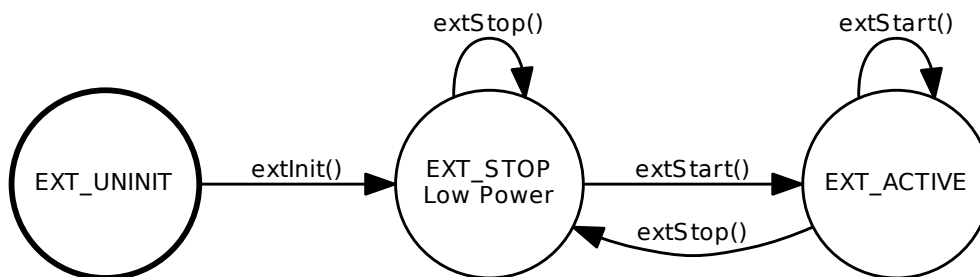
This module implements a generic EXT (EXternal) driver.

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_LOW_LEVEL 5U`
low level 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_llc_init` (void)
Low level EXT driver initialization.
- void `ext_llc_start` (EXTDriver *extp)
Configures and activates the EXT peripheral.
- void `ext_llc_stop` (EXTDriver *extp)
Deactivates the EXT peripheral.
- void `ext_llc_channel_enable` (EXTDriver *extp, expchannel_t channel)
Enables an EXT channel.
- void `ext_llc_channel_disable` (EXTDriver *extp, expchannel_t channel)
Disables an EXT channel.

Enumerations

Variables

- `EXTDriver EXT1`
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_LOW_LEVEL 5U

low level callback.

7.4.4.7 #define EXT_CH_MODE_AUTOSTART 4U

Channel started automatically on driver start.

7.4.4.8 #define extChannelEnable(*extp, channel*) ext_lld_channel_enable(extp, 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:

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 extChannelDisable(*extp, channel*) ext_lld_channel_disable(extp, 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:

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

7.4.4.10 #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.

7.4.4.11 #define EXT_MAX_CHANNELS 20

Available number of EXT channels.

7.4.4.12 #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>EXPDriver</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.

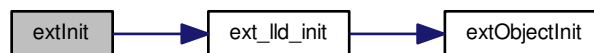
Note

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

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 EXTDriver 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.

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	<i>extp</i>	pointer to the EXTDriver object
in	<i>channel</i>	channel to be changed
in	<i>extcp</i>	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 EXT1

EXT1 driver identifier.

7.5 Abstract NOR Flash Class

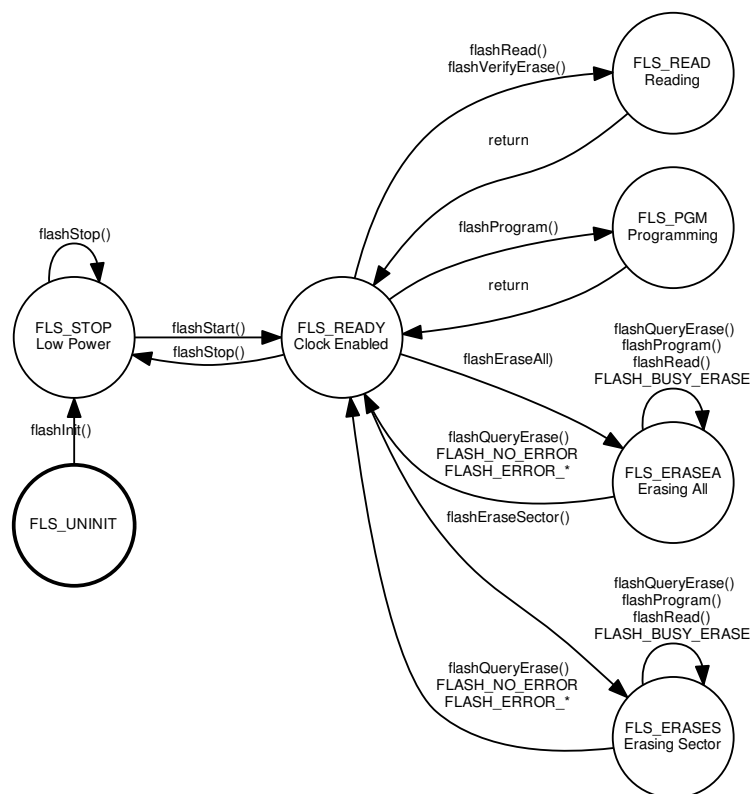
Generic NOR Flash interface.

Generic NOR Flash interface.

This module implements a generic class for NOR Flash devices.

7.5.1 Driver State Machine

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



7.5.2 Flash 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.

7.6 GPT Driver

Generic GPT Driver.

7.6.1 Detailed Description

Generic GPT Driver.

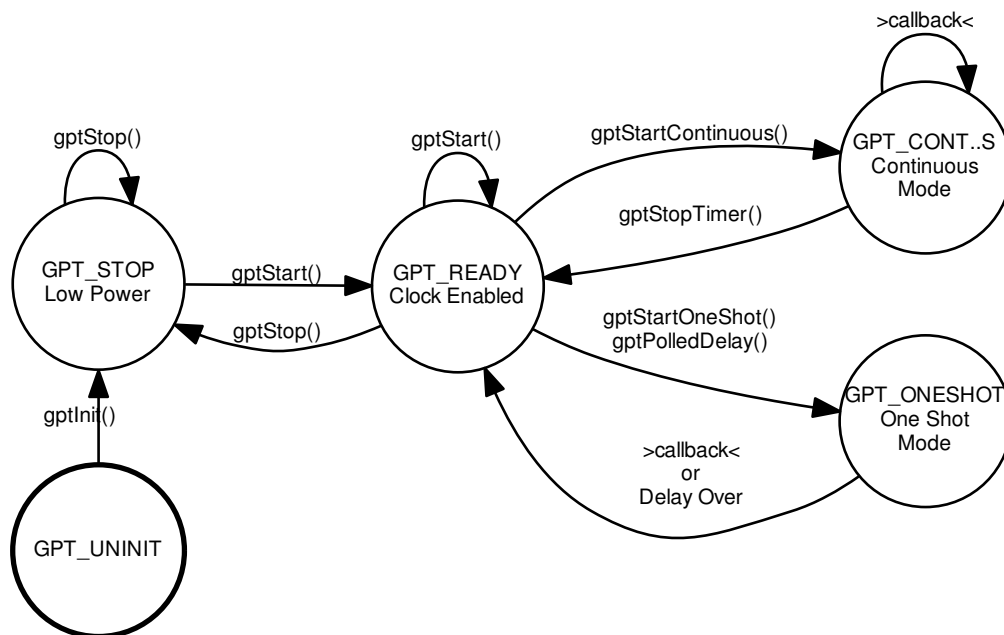
This module implements a generic GPT (General Purpose Timer) driver. The timer can be programmed in order to trigger callbacks after a specified time period or continuously with a specified interval.

Precondition

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

7.6.2 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).



7.6.3 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_ll_get_interval(gptp)`
Returns the interval of GPT peripheral.
- #define `gptGetCounterX(gptp) gpt_ll_get_counter(gptp)`
Returns the counter value of GPT peripheral.
- #define `gpt_ll_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

Variables

- `GPTDriver GPTD1`
GPTD1 driver identifier.

7.6.4 Macro Definition Documentation

7.6.4.1 #define gptChangeInterval(*gtp*, *interval*)

Value:

```
{  
    gpt_lld_change_interval(gtp, 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>gtp</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.

7.6.4.2 #define gptGetIntervalX(*gtp*) gpt_lld_get_interval(gtp)

Returns the interval of GPT peripheral.

Precondition

The GPT unit must be running in continuous mode.

Parameters

in	<i>gtp</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.

7.6.4.3 #define gptGetCounterX(*gtp*) gpt_lld_get_counter(gtp)

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.

7.6.4.4 #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`.

7.6.4.5 #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>gtp</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.

7.6.5 Typedef Documentation

7.6.5.1 typedef struct GPTDriver GPTDriver

Type of a structure representing a GPT driver.

7.6.5.2 typedef void(* gptcallback_t) (GPTDriver *gtp)

GPT notification callback type.

Parameters

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

7.6.5.3 typedef uint32_t gptfreq_t

GPT frequency type.

7.6.5.4 typedef uint16_t gptcnt_t

GPT counter type.

7.6.6 Enumeration Type Documentation

7.6.6.1 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.

7.6.7 Function Documentation

7.6.7.1 void gptlnit (void)

GPT Driver initialization.

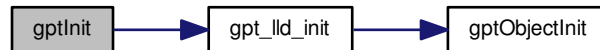
Note

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:

**7.6.7.2 void gptObjectInit (GPTDriver * gtp)**

Initializes the standard part of a `GPTDriver` structure.

Parameters

out	<i>gtp</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.

7.6.7.3 void gptStart (GPTDriver * gtp, const GPTConfig * config)

Configures and activates the GPT peripheral.

Parameters

in	<i>gtp</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:



7.6.7.4 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:



7.6.7.5 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.

7.6.7.6 void gptStartContinuous (GPTDriver * *gtp*, gptcnt_t *interval*)

Starts the timer in continuous mode.

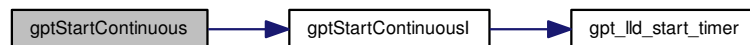
Parameters

in	<i>gtp</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:

**7.6.7.7 void gptStartContinuousI (GPTDriver * *gtp*, gptcnt_t *interval*)**

Starts the timer in continuous mode.

Parameters

in	<i>gtp</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:



7.6.7.8 void gptStartOneShot (GPTDriver * *gptp*, gptcnt_t *interval*)

Starts the timer in one shot mode.

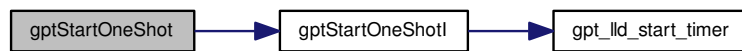
Parameters

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:



7.6.7.9 void gptStartOneShotl (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:



7.6.7.10 void gptStopTimer (GPTDriver * *gptp*)

Stops the timer.

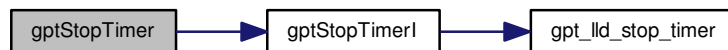
Parameters

in	<i>gtp</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:

7.6.7.11 void gptStopTimerI (GPTDriver * *gtp*)

Stops the timer.

Parameters

in	<i>gtp</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:

7.6.7.12 void gptPolledDelay (GPTDriver * *gtp*, gptcnt_t *interval*)

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

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

Note

The configured callback is not invoked when using this function.

Parameters

in	<i>gtp</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:

**7.6.7.13 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:

**7.6.7.14 void gpt_lld_start (GPTDriver * gtp)**

Configures and activates the GPT peripheral.

Parameters

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

Function Class:

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

7.6.7.15 void gpt_ild_stop (GPTDriver * gtp)

Deactivates the GPT peripheral.

Parameters

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

Function Class:

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

7.6.7.16 void gpt_ild_start_timer (GPTDriver * gtp, gptcnt_t interval)

Starts the timer in continuous mode.

Parameters

in	<i>gtp</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.

7.6.7.17 void gpt_ild_stop_timer (GPTDriver * gtp)

Stops the timer.

Parameters

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

Function Class:

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

7.6.7.18 void gpt_ild_polled_delay (GPTDriver * gtp, gptcnt_t interval)

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

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

Parameters

in	<i>gtp</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.

7.6.8 Variable Documentation**7.6.8.1 GPTDriver GPTD1**

GPTD1 driver identifier.

7.7 HAL Driver

Hardware Abstraction Layer.

7.7.1 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 "5.0.0"`
HAL version string.
- `#define CH_HAL_MAJOR 5`
HAL version major number.
- `#define CH_HAL_MINOR 0`
HAL version minor number.
- `#define CH_HAL_PATCH 0`
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.

7.7.2 Macro Definition Documentation

7.7.2.1 `#define _CHIBIOS_HAL_`

ChibiOS/HAL identification macro.

7.7.2.2 `#define CH_HAL_STABLE 1`

Stable release flag.

7.7.2.3 `#define HAL_VERSION "5.0.0"`

HAL version string.

7.7.2.4 `#define CH_HAL_MAJOR 5`

HAL version major number.

7.7.2.5 `#define CH_HAL_MINOR 0`

HAL version minor number.

7.7.2.6 `#define CH_HAL_PATCH 0`

HAL version patch number.

7.7.3 Function Documentation

7.7.3.1 `void hallnit (void)`

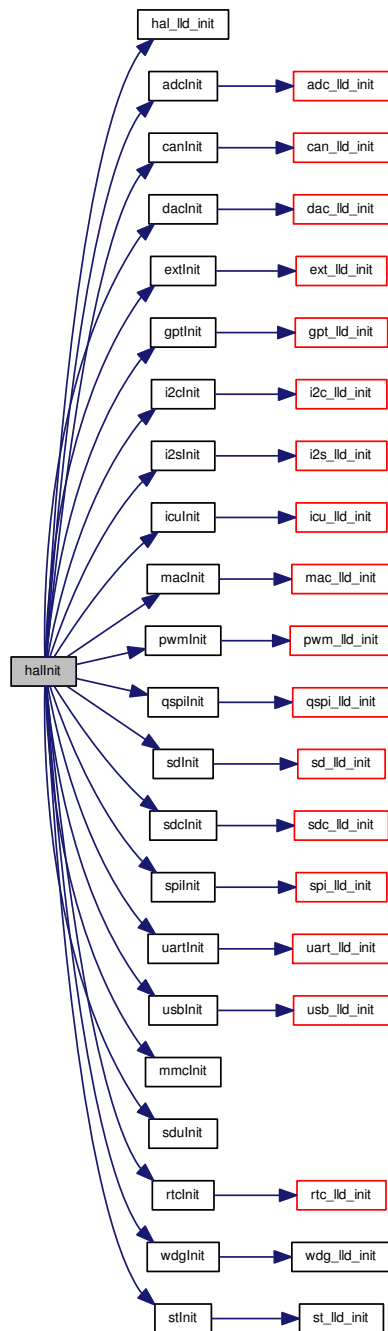
HAL initialization.

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:



7.7.3.2 void hal_ld_init (void)

Low level HAL driver initialization.

Function Class:

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

7.8 I/O Buffers Queues

7.8.1 Detailed Description

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

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

Macros

- `#define BQ_BUFFER_SIZE(n, size) (((size_t)(size) + sizeof (size_t)) * (size_t)(n))`
Computes the size of a buffers queue buffer size.

Macro Functions

- `#define bqSizeX(bqp) ((bqp)->bn)`
Returns the queue's number of buffers.
- `#define bqSpacel(bqp) ((bqp)->bcounter)`
Return the ready buffers number.
- `#define bqGetLinkX(bqp) ((bqp)->link)`
Returns the queue application-defined link.
- `#define bqIsSuspendedX(bqp) ((bqp)->suspended)`
Return the suspended state of the queue.
- `#define bqSuspendl(bqp)`
Puts the queue in suspended state.
- `#define bqResumeX(bqp)`
Resumes normal queue operations.
- `#define ibqlsEmptyl(ibqp) ((bool)(bqSpacel(ibqp) == 0U))`
Evaluates to `TRUE` if the specified input buffers queue is empty.
- `#define ibqlsFullll(ibqp)`
Evaluates to `TRUE` if the specified input buffers queue is full.
- `#define obqlsEmptyl(obqp)`
Evaluates to `true` if the specified output buffers queue is empty.
- `#define obqlsFullll(obqp) ((bool)(bqSpacel(obqp) == 0U))`
Evaluates to `true` if the specified output buffers queue is full.

Typedefs

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

Data Structures

- struct `io_buffers_queue`
Structure of a generic buffers queue.

Functions

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

7.8.2 Macro Definition Documentation

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

Computes the size of a buffers queue buffer size.

Parameters

in	<i>n</i>	number of buffers in the queue
in	<i>size</i>	size of the buffers

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

Returns the queue's number of buffers.

Parameters

in	<i>bqp</i>	pointer to an <code>io_buffers_queue_t</code> structure
----	------------	---------------------------------------------------------

Returns

The number of buffers.

Function Class:

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

7.8.2.3 `#define bqSpacel(bqp) ((bqp)->bcounter)`

Return the ready buffers number.

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

Parameters

in	<i>bqp</i>	pointer to an <code>io_buffers_queue_t</code> structure
----	------------	---------------------------------------------------------

Returns

The number of ready buffers.

Function Class:

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

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

Returns the queue application-defined link.

Parameters

in	<i>bqp</i>	pointer to an <code>io_buffers_queue_t</code> structure
----	------------	---------------------------------------------------------

Returns

The application-defined link.

Function Class:

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

7.8.2.5 #define bqlsSuspendedX(*bqp*) ((bqp)->suspended)

Return the suspended state of the queue.

Parameters

in	<i>bqp</i>	pointer to an <code>io_buffers_queue_t</code> structure
----	------------	---------------------------------------------------------

Returns

The suspended state.

Return values

<i>false</i>	if blocking access to the queue is enabled.
<i>true</i>	if blocking access to the queue is suspended.

Function Class:

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

7.8.2.6 #define bqSuspendI(*bqp*)**Value:**

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

Puts the queue in suspended state.

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

Note

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

Parameters

in	<i>bqp</i>	pointer to an <code>io_buffers_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.

7.8.2.7 #define bqResumeX(*bqp*)**Value:**

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

Resumes normal queue operations.

Parameters

in	<i>bqp</i>	pointer to an <code>io_buffers_queue_t</code> structure
----	------------	---------------------------------------------------------

Function Class:

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

7.8.2.8 #define ibqlsEmpty(*ibqp*) ((bool)(bqSpacel(ibqp) == 0U))

Evaluates to `TRUE` if the specified input buffers queue is empty.

Parameters

in	<i>ibqp</i>	pointer to an <code>input_buffers_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.

7.8.2.9 #define ibqlsFull(*ibqp*)

Value:

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

Evaluates to `TRUE` if the specified input buffers queue is full.

Parameters

in	<i>ibqp</i>	pointer to an <code>input_buffers_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.

7.8.2.10 #define obqlsEmpty(*obqp*)

Value:

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

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

Parameters

in	<i>obqp</i>	pointer to an <code>output_buffers_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.

7.8.2.11 `#define obqIsFull(obqp) ((bool)(bqSpacel(obqp) == 0U))`

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

Parameters

in	<i>obqp</i>	pointer to an <code>output_buffers_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.

7.8.3 Typedef Documentation

7.8.3.1 `typedef struct io_buffers_queue io_buffers_queue_t`

Type of a generic queue of buffers.

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

Double buffer notification callback type.

Parameters

in	<i>iobdp</i>	the buffers queue pointer
----	--------------	---------------------------

7.8.3.3 `typedef io_buffers_queue_t input_buffers_queue_t`

Type of an input buffers queue.

7.8.3.4 `typedef io_buffers_queue_t output_buffers_queue_t`

Type of an output buffers queue.

7.8.4 Function Documentation

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

Initializes an input buffers queue object.

Parameters

out	<i>ibqp</i>	pointer to the input_buffers_queue_t object
in	<i>suspended</i>	initial state of the queue
in	<i>bp</i>	pointer to a memory area allocated for buffers
in	<i>size</i>	buffers size
in	<i>n</i>	number of buffers
in	<i>infy</i>	callback called when a buffer is returned to the queue
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.

7.8.4.2 void ibqResetI (input_buffers_queue_t * *ibqp*)

Resets an input buffers queue.

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

Note

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

Parameters

in	<i>ibqp</i>	pointer to the input_buffers_queue_t object
----	-------------	---------------------------------------------

Function Class:

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

7.8.4.3 uint8_t * ibqGetEmptyBufferI (input_buffers_queue_t * *ibqp*)

Gets the next empty buffer from the queue.

Note

The function always returns the same buffer if called repeatedly.

Parameters

in	<i>ibqp</i>	pointer to the input_buffers_queue_t object
----	-------------	---------------------------------------------

Returns

A pointer to the next buffer to be filled.

Return values

<i>NULL</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.

7.8.4.4 void ibqPostFullBufferI (input_buffers_queue_t * ibqp, size_t size)

Posts a new filled buffer to the queue.

Parameters

in	<i>ibqp</i>	pointer to the <code>input_buffers_queue_t</code> object
in	<i>size</i>	used size of the buffer, cannot be zero

Function Class:

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

7.8.4.5 msg_t ibqGetFullBufferTimeout (input_buffers_queue_t * ibqp, systime_t timeout)

Gets the next filled buffer from the queue.

Note

The function always acquires the same buffer if called repeatedly.

Postcondition

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

Parameters

in	<i>ibqp</i>	pointer to the <code>input_buffers_queue_t</code> object
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

<i>MSG_OK</i>	if a buffer has been acquired.
<i>MSG_TIMEOUT</i>	if the specified time expired.
<i>MSG_RESET</i>	if the queue has been reset or has been put in suspended state.

Function Class:

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

Here is the call graph for this function:



7.8.4.6 msg_t ibqGetFullBufferTimeoutS (input_buffers_queue_t * *ibqp*, systime_t *timeout*)

Gets the next filled buffer from the queue.

Note

The function always acquires the same buffer if called repeatedly.

Postcondition

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

Parameters

in	<i>ibqp</i>	pointer to the <code>input_buffers_queue_t</code> object
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>	if a buffer has been acquired.
<i>MSG_TIMEOUT</i>	if the specified time expired.
<i>MSG_RESET</i>	if the queue has been reset or has been put in suspended state.

Function Class:

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

7.8.4.7 void ibqReleaseEmptyBuffer (input_buffers_queue_t * ibqp)

Releases the buffer back in the queue.

Note

The object callback is called after releasing the buffer.

Parameters

in	<i>ibqp</i>	pointer to the <code>input_buffers_queue_t</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:



7.8.4.8 void ibqReleaseEmptyBufferS (input_buffers_queue_t * ibqp)

Releases the buffer back in the queue.

Note

The object callback is called after releasing the buffer.

Parameters

in	<i>ibqp</i>	pointer to the <code>input_buffers_queue_t</code> object
----	-------------	----------------------------------------------------------

Function Class:

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

7.8.4.9 msg_t ibqGetTimeout (input_buffers_queue_t * ibqp, 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 new buffer arrives in the queue or a timeout occurs.

Parameters

in	<i>ibqp</i>	pointer to the <code>input_buffers_queue_t</code> object
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

A byte value from the queue.

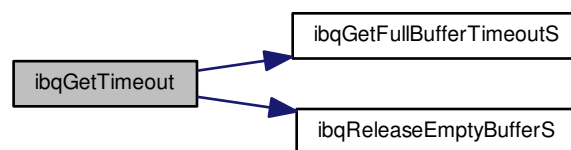
Return values

<i>MSG_TIMEOUT</i>	if the specified time expired.
<i>MSG_RESET</i>	if the queue has been reset or has been put in suspended state.

Function Class:

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

Here is the call graph for this function:

**7.8.4.10 size_t ibqReadTimeout (input_buffers_queue_t * ibqp, 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.

Parameters

in	<i>ibqp</i>	pointer to the <code>input_buffers_queue_t</code> object
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"> • <i>TIME_IMMEDIATE</i> immediate timeout. • <i>TIME_INFINITE</i> no timeout.

Returns

The number of bytes effectively transferred.

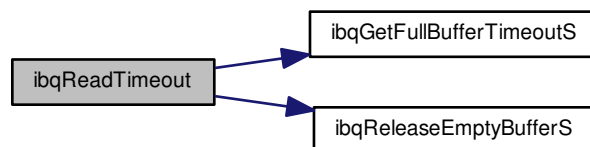
Return values

0	if a timeout occurred.
---	------------------------

Function Class:

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

Here is the call graph for this function:



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

Initializes an output buffers queue object.

Parameters

out	<i>obqp</i>	pointer to the <code>output_buffers_queue_t</code> object
in	<i>suspended</i>	initial state of the queue
in	<i>bp</i>	pointer to a memory area allocated for buffers
in	<i>size</i>	buffers size
in	<i>n</i>	number of buffers
in	<i>onfy</i>	callback called when a buffer is posted in the queue
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.

7.8.4.12 void obqResetI (output_buffers_queue_t * obqp)

Resets an output buffers queue.

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

Note

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

Parameters

in	<i>obqp</i>	pointer to the <code>output_buffers_queue_t</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.

7.8.4.13 uint8_t * obqGetFullBufferI (output_buffers_queue_t * obqp, size_t * sizep)

Gets the next filled buffer from the queue.

Note

The function always returns the same buffer if called repeatedly.

Parameters

in	<i>obqp</i>	pointer to the <code>output_buffers_queue_t</code> object
out	<i>sizep</i>	pointer to the filled buffer size

Returns

A pointer to the filled buffer.

Return values

<i>NULL</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.

7.8.4.14 void obqReleaseEmptyBufferI (output_buffers_queue_t * obqp)

Releases the next filled buffer back in the queue.

Parameters

in	<i>obqp</i>	pointer to the <code>output_buffers_queue_t</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.

7.8.4.15 msg_t obqGetEmptyBufferTimeout (output_buffers_queue_t * obqp, systime_t timeout)

Gets the next empty buffer from the queue.

Note

The function always acquires the same buffer if called repeatedly.

Postcondition

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

Parameters

in	<i>obqp</i>	pointer to the <code>output_buffers_queue_t</code> object
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>	if a buffer has been acquired.
<code>MSG_TIMEOUT</code>	if the specified time expired.
<code>MSG_RESET</code>	if the queue has been reset or has been put in suspended state.

Function Class:

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

Here is the call graph for this function:



7.8.4.16 msg_t obqGetEmptyBufferTimeoutS (output_buffers_queue_t * obqp, systime_t timeout)

Gets the next empty buffer from the queue.

Note

The function always acquires the same buffer if called repeatedly.

Postcondition

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

Parameters

in	<i>obqp</i>	pointer to the <code>output_buffers_queue_t</code> object
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>	if a buffer has been acquired.
<code>MSG_TIMEOUT</code>	if the specified time expired.
<code>MSG_RESET</code>	if the queue has been reset or has been put in suspended state.

Function Class:

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

7.8.4.17 void obqPostFullBuffer (output_buffers_queue_t * obqp, size_t size)

Posts a new filled buffer to the queue.

Note

The object callback is called after releasing the buffer.

Parameters

in	<i>obqp</i>	pointer to the <code>output_buffers_queue_t</code> object
in	<i>size</i>	used size of the buffer, cannot be zero

Function Class:

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

Here is the call graph for this function:



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

Posts a new filled buffer to the queue.

Note

The object callback is called after releasing the buffer.

Parameters

in	<i>obqp</i>	pointer to the <code>output_buffers_queue_t</code> object
in	<i>size</i>	used size of the buffer, cannot be zero

Function Class:

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

7.8.4.19 msg_t `obqPutTimeout (output_buffers_queue_t * obqp, 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 a new buffer is freed in the queue or a timeout occurs.

Parameters

in	<i>obqp</i>	pointer to the <code>output_buffers_queue_t</code> object
in	<i>b</i>	byte value to be transferred

Parameters

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

A byte value from the queue.

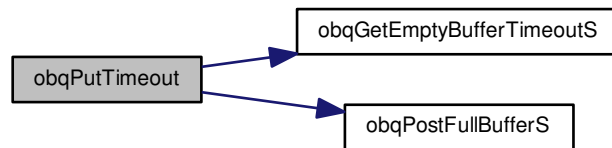
Return values

<i>MSG_TIMEOUT</i>	if the specified time expired.
<i>MSG_RESET</i>	if the queue has been reset or has been put in suspended state.

Function Class:

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

Here is the call graph for this function:



7.8.4.20 `size_t obqWriteTimeout (output_buffers_queue_t * obqp, 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.

Parameters

in	<i>obqp</i>	pointer to the <code>output_buffers_queue_t</code> object
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.

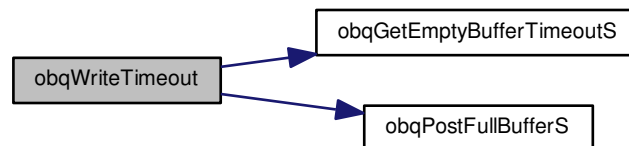
Return values

0	if a timeout occurred.
---	------------------------

Function Class:

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

Here is the call graph for this function:



7.8.4.21 bool obqTryFlushI (output_buffers_queue_t * obqp)

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

Note

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

Parameters

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

Returns

The operation status.

Return values

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

Function Class:

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

7.8.4.22 void obqFlush (output_buffers_queue_t * obqp)

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

Parameters

in	<i>obqp</i>	pointer to the output_buffers_queue_t object
----	-------------	----------------------------------------------

Function Class:

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

Here is the call graph for this function:



7.9 Abstract I/O Channel

7.9.1 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 chnAddFlagsl(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.

7.9.2 Macro Definition Documentation

7.9.2.1 #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.

7.9.2.2 #define _base_channel_data _base_sequential_stream_data

`BaseChannel` specific data.

Note

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

7.9.2.3 #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 <code>BaseChannel</code> or derived class
in	<i>b</i>	the byte value to be written to the channel

Parameters

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.

```
7.9.2.4 #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.

7.9.2.5 #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.

7.9.2.6 #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.

7.9.2.7 #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
----	-----------	-----------------------------------------------------------

Parameters

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.

7.9.2.8 `#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.

7.9.2.9 `#define CHN_NO_ERROR (eventflags_t)0`

No pending conditions.

7.9.2.10 `#define CHN_CONNECTED (eventflags_t)1`

Connection happened.

7.9.2.11 `#define CHN_DISCONNECTED (eventflags_t)2`

Disconnection happened.

7.9.2.12 `#define CHN_INPUT_AVAILABLE (eventflags_t)4`

Data available in the input queue.

7.9.2.13 #define CHN_OUTPUT_EMPTY (eventflags_t)8

Output queue empty.

7.9.2.14 #define CHN_TRANSMISSION_END (eventflags_t)16

Transmission end.

7.9.2.15 #define _base_asynchronous_channel_methods _base_channel_methods \

[BaseAsynchronousChannel](#) specific methods.

7.9.2.16 #define _base_asynchronous_channel_data

Value:

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

[BaseAsynchronousChannel](#) specific data.

7.9.2.17 #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	<i>ip</i>	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.

7.9.2.18 #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	<i>ip</i>	pointer to a BaseAsynchronousChannel or derived class
in	<i>flags</i>	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.

7.10 Abstract Files

7.10.1 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.

7.10.2 Macro Definition Documentation

7.10.2.1 #define FILE_OK STM_OK

No error return code.

7.10.2.2 #define FILE_ERROR STM_TIMEOUT

Error code from the file stream methods.

7.10.2.3 #define FILE_EOF STM_RESET

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

7.10.2.4 #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.

7.10.2.5 #define _file_stream_data _base_sequential_stream_data

[FileStream](#) specific data.

Note

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

7.10.2.6 #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

<code>FILE_ERROR</code>	operation failed.
-------------------------	-------------------

Function Class:

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

7.10.2.7 #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

<code>FILE_ERROR</code>	operation failed.
-------------------------	-------------------

Function Class:

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

7.10.2.8 #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.

7.10.2.9 #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.

7.10.2.10 #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.

7.10.2.11 `#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	<i>ip</i>	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.

7.10.2.12 `#define fileStreamGetSize(ip) ((ip)->vmt->getsize(ip))`

Returns the current file size.

Parameters

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

Returns

The file size.

Return values

<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.

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

Returns the current file pointer position.

Parameters

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

Returns

The current position inside the file.

Return values

<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.

7.10.2.14 #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.

7.10.3 Typedef Documentation

7.10.3.1 typedef uint32_t fileoffset_t

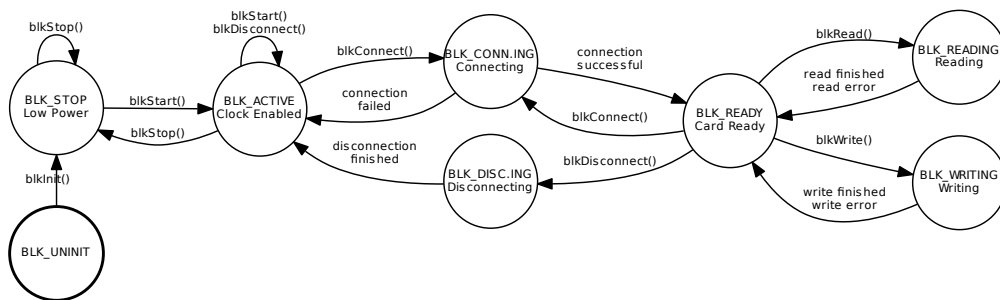
File offset type.

7.11 Abstract I/O Block Device

7.11.1 Detailed Description

7.11.2 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

7.11.3 Macro Definition Documentation

7.11.3.1 #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.

7.11.3.2 #define _base_block_device_data

Value:

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

`BaseBlockDevice` specific data.

7.11.3.3 `#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.

7.11.3.4 `#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.

7.11.3.5 `#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 <code>BaseBlockDevice</code> 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.

7.11.3.6 `#define blkIsWriteProtected(ip) ((ip)->vmt->is_protected(ip))`

Returns the media write protection status.

Parameters

in	<i>ip</i>	pointer to a <code>BaseBlockDevice</code> 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.

7.11.3.7 `#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.

7.11.3.8 `#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.

7.11.3.9 `#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

Parameters

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.

7.11.3.10 `#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.

7.11.3.11 `#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.

7.11.3.12 `#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>bdip</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.

7.11.4 Enumeration Type Documentation**7.11.4.1 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.

7.12 I/O Bytes Queues

7.12.1 Detailed Description

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

There are several kind of queues:

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

Queue functions returned status value

- `#define Q_OK MSG_OK`
Operation successful.
- `#define Q_TIMEOUT MSG_TIMEOUT`
Timeout condition.
- `#define Q_RESET MSG_RESET`
Queue has been reset.
- `#define Q_EMPTY MSG_TIMEOUT`
Queue empty.
- `#define Q_FULL MSG_TIMEOUT`
Queue full,.

Macro Functions

- `#define qSizeX(qp)`
Returns the queue's buffer size.
- `#define qSpacel(qp) ((qp)->q_counter)`
Queue space.
- `#define qGetLink(qp) ((qp)->q_link)`
Returns the queue application-defined link.
- `#define iqGetFullI(iqp) qSpacel(iqp)`
Returns the filled space into an input queue.
- `#define iqGetEmptyI(iqp) (qSizeX(iqp) - qSpacel(iqp))`
Returns the empty space into an input queue.
- `#define iqlsEmptyI(iqp) ((bool)(qSpacel(iqp) == 0U))`
Evaluates to `true` if the specified input queue is empty.
- `#define iqlsFullI(iqp)`
Evaluates to `true` if the specified input queue is full.
- `#define iqGet(iqp) iqGetTimeout(iqp, TIME_INFINITE)`
Input queue read.
- `#define oqGetFullI(oqp) (qSizeX(oqp) - qSpacel(oqp))`
Returns the filled space into an output queue.
- `#define oqGetEmptyI(oqp) qSpacel(oqp)`
Returns the empty space into an output queue.

- `#define oqlsEmptyl(oqp)`
Evaluates to `true` if the specified output queue is empty.
- `#define oqlsFulll(oqp) ((bool)(qSpacel(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 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.

7.12.2 Macro Definition Documentation

7.12.2.1 `#define Q_OK MSG_OK`

Operation successful.

7.12.2.2 #define Q_TIMEOUT MSG_TIMEOUT

Timeout condition.

7.12.2.3 #define Q_RESET MSG_RESET

Queue has been reset.

7.12.2.4 #define Q_EMPTY MSG_TIMEOUT

Queue empty.

7.12.2.5 #define Q_FULL MSG_TIMEOUT

Queue full,.

7.12.2.6 #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	<i>qp</i>	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.

7.12.2.7 #define qSpacel(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.

7.12.2.8 `#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.

7.12.2.9 `#define iqGetFull(iqp) qSpacel(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.

7.12.2.10 `#define iqGetEmptyl(iqp) (qSizeX(iqp) - qSpacel(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.

7.12.2.11 `#define iqlsEmptyl(iqp) ((bool)(qSpacel(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.

7.12.2.12 `#define iqlsFullll(iqp)`

Value:

```
/*lint -save -e9007 [13.5] No side effects, a pointer is passed.*/ \
((bool) (((iqp)->q_wrptr == (iqp)->q_rdptr) && ((iqp)->q_counter != 0U))) \
/*lint -restore*/
```

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.

7.12.2.13 `#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

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

Returns

A byte value from the queue.

Return values

<i>MSG_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.

7.12.2.14 `#define oqGetFull(oqp) (qSizeX(oqp) - qSpace1(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.

7.12.2.15 #define oqGetEmpty(oqp) qSpacel(oqp)

Returns the empty space into an output queue.

Parameters

in	oqp	pointer to an output_queue_t structure
----	-----	----------------------------------------

Returns

The number of empty bytes in the queue.

Return values

0	if the queue is full.
---	-----------------------

Function Class:

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

7.12.2.16 #define oqlsEmpty(oqp)**Value:**

```
/*lint -save -e9007 [13.5] No side effects, a pointer is passed.*/ \
((bool) (((oqp)->q_wrptr == (oqp)->q_rdptr) && ((oqp)->q_counter != 0U))) \
/*lint -restore*/
```

Evaluates to true if the specified output queue is empty.

Parameters

in	oqp	pointer to an output_queue_t structure
----	-----	----------------------------------------

Returns

The queue status.

Return values

<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.

7.12.2.17 `#define oqlsFull(oqp) ((bool)(qSpace1(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.

7.12.2.18 `#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>MSG_OK</i>	if the operation succeeded.
---------------	-----------------------------

Return values

<code>MSG_RESET</code>	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.

7.12.3 Typedef Documentation

7.12.3.1 typedef struct io_queue io_queue_t

Type of a generic I/O queue structure.

7.12.3.2 typedef void(* qnotify_t)(io_queue_t *qp)

Queue notification callback type.

Parameters

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

7.12.3.3 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.

7.12.3.4 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.

7.12.4 Function Documentation

7.12.4.1 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
-----	------------	----------------------------------------------------

Parameters

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 NULL.
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.

7.12.4.2 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 MSG_RESET.

Note

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

Parameters

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

Function Class:

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

7.12.4.3 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 input_queue_t structure
in	<i>b</i>	the byte value to be written in the queue

Returns

The operation status.

Return values

<i>MSG_OK</i>	if the operation has been completed with success.
<i>MSG_TIMEOUT</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.

7.12.4.4 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 after removing a character from the queue.

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

<code>MSG_TIMEOUT</code>	if the specified time expired.
<code>MSG_RESET</code>	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.

7.12.4.5 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 after removing each character from the queue.

Parameters

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

Parameters

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"> • <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.

7.12.4.6 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 output_queue_t 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 NULL.
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.

7.12.4.7 void oqResetI (output_queue_t * oqp)

Resets an output queue.

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

Note

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

Parameters

in	<i>oqp</i>	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.

7.12.4.8 `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 putting the character into 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
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>	if the operation succeeded.
<i>MSG_TIMEOUT</i>	if the specified time expired.
<i>MSG_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.

7.12.4.9 `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

<code>MSG_TIMEOUT</code>	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.

7.12.4.10 `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 putting each character into the queue.

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"> • <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.

7.13 Abstract Streams

7.13.1 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.

7.13.2 Macro Definition Documentation

7.13.2.1 `#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.

7.13.2.2 #define _base_sequential_stream_data

[BaseSequentialStream](#) specific data.

Note

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

7.13.2.3 #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	<i>ip</i>	pointer to a BaseSequentialStream 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.

Function Class:

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

7.13.2.4 #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	<i>ip</i>	pointer to a BaseSequentialStream 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.

Function Class:

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

7.13.2.5 #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.

7.13.2.6 #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.

7.14 I2C Driver

Generic I2C Driver.

7.14.1 Detailed Description

Generic I2C Driver.

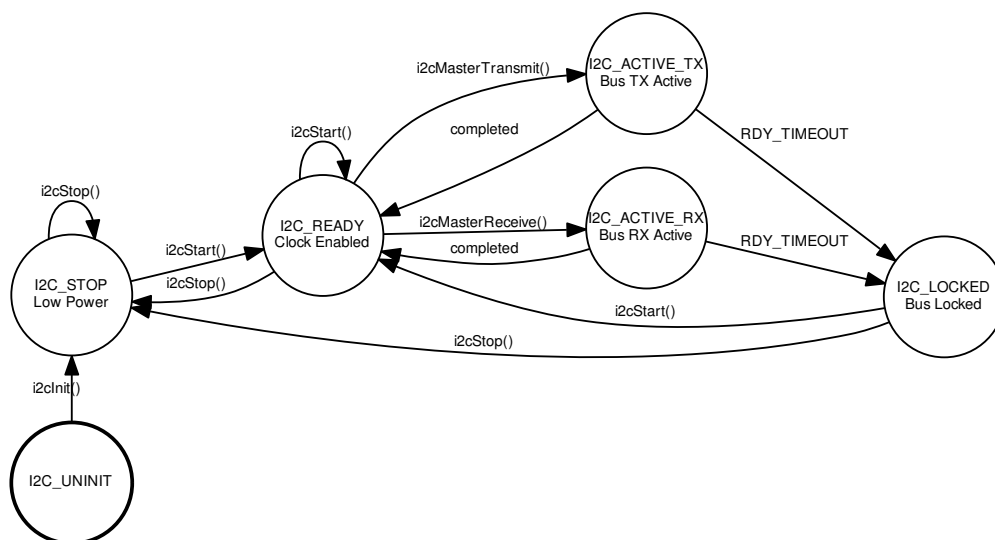
This module implements a generic I2C (Inter-Integrated Circuit) driver.

Precondition

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

7.14.2 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).



The driver is not thread safe for performance reasons, if you need to access the I2C bus from multiple threads then use the `i2cAcquireBus()` and `i2cReleaseBus()` APIs in order to gain exclusive access.

Macros

- `#define I2C_USE_MUTUAL_EXCLUSION TRUE`
Enables the mutual exclusion APIs on the I2C bus.
- `#define _i2c_wakeup_isr(i2cp)`
Wakes up the waiting thread notifying no errors.
- `#define _i2c_wakeup_error_isr(i2cp)`

- *Wakes up the waiting thread notifying errors.*
- #define `i2cMasterTransmit`(i2cp, addr, txbuf, txbytes, rxbuf, rxbytes)
Wrap i2cMasterTransmitTimeout function with TIME_INFINITE timeout.
- #define `i2cMasterReceive`(i2cp, addr, rxbuf, rxbytes) (`i2cMasterReceiveTimeout`(i2cp, addr, rxbuf, rxbytes, TIME_INFINITE))
Wrap i2cMasterReceiveTimeout function with TIME_INFINITE timeout.
- #define `i2c_Ild_get_errors`(i2cp) ((i2cp)->errors)
Get errors from I2C driver.

I2C bus error conditions

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

PLATFORM configuration options

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

Typedefs

- typedef uint16_t `i2caddr_t`
Type representing an I2C address.
- typedef uint32_t `i2cflags_t`
Type of I2C Driver condition flags.
- typedef struct `I2CDriver` `I2CDriver`
Type of a structure representing an I2C driver.

Data Structures

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

Functions

- void `i2cInit` (void)
I2C Driver initialization.
- void `i2cObjectInit` (`I2CDriver` *i2cp)
Initializes the standard part of a `I2CDriver` structure.
- void `i2cStart` (`I2CDriver` *i2cp, const `I2CConfig` *config)
Configures and activates the I2C peripheral.
- void `i2cStop` (`I2CDriver` *i2cp)
Deactivates the I2C peripheral.
- `i2cflags_t` `i2cGetErrors` (`I2CDriver` *i2cp)
Returns the errors mask associated to the previous operation.
- `msg_t` `i2cMasterTransmitTimeout` (`I2CDriver` *i2cp, `i2caddr_t` addr, const `uint8_t` *txbuf, `size_t` txbytes, `uint8_t` *rxbuf, `size_t` rxbytes, `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.
- void `i2c_lld_init` (void)
Low level I2C driver initialization.
- void `i2c_lld_start` (`I2CDriver` *i2cp)
Configures and activates the I2C peripheral.
- void `i2c_lld_stop` (`I2CDriver` *i2cp)
Deactivates the I2C peripheral.
- `msg_t` `i2c_lld_master_receive_timeout` (`I2CDriver` *i2cp, `i2caddr_t` addr, `uint8_t` *rxbuf, `size_t` rxbytes, `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` *rxbuf, `size_t` rxbytes, `systime_t` timeout)
Transmits data via the I2C bus as master.

Enumerations

Variables

- `I2CDriver` `I2CD1`
I2C1 driver identifier.

7.14.3 Macro Definition Documentation

7.14.3.1 `#define I2C_NO_ERROR 0x00`

No error.

7.14.3.2 `#define I2C_BUS_ERROR 0x01`

Bus Error.

7.14.3.3 `#define I2C_ARBITRATION_LOST 0x02`

Arbitration Lost.

7.14.3.4 `#define I2C_ACK_FAILURE 0x04`

Acknowledge Failure.

7.14.3.5 `#define I2C_OVERRUN 0x08`

Overrun/Underrun.

7.14.3.6 `#define I2C_PEC_ERROR 0x10`

PEC Error in reception.

7.14.3.7 `#define I2C_TIMEOUT 0x20`

Hardware timeout.

7.14.3.8 `#define I2C_SMB_ALERT 0x40`

SMBus Alert.

7.14.3.9 `#define I2C_USE_MUTUAL_EXCLUSION TRUE`

Enables the mutual exclusion APIs on the I2C bus.

7.14.3.10 `#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	<i>i2cp</i>	pointer to the I2CDriver object
----	-------------	-------------------------------------------------

Function Class:

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

7.14.3.11 `#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.

7.14.3.12 `#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.

7.14.3.13 `#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.

7.14.3.14 `#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`.

7.14.3.15 `#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.

7.14.4 Typedef Documentation**7.14.4.1 typedef uint16_t i2caddr_t**

Type representing an I2C address.

7.14.4.2 typedef uint32_t i2cflags_t

Type of I2C Driver condition flags.

7.14.4.3 typedef struct I2CDriver I2CDriver

Type of a structure representing an I2C driver.

7.14.5 Enumeration Type Documentation**7.14.5.1 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.

7.14.6 Function Documentation**7.14.6.1 void i2cInit (void)**

I2C Driver initialization.

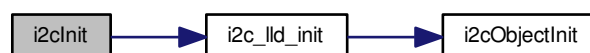
Note

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

Function Class:

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

Here is the call graph for this function:



7.14.6.2 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.

7.14.6.3 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:



7.14.6.4 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:



7.14.6.5 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.

7.14.6.6 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.

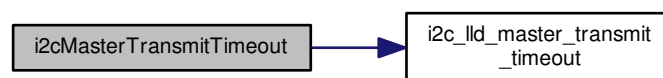
Return values

<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:



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

Receives data from the I2C bus.

Parameters

in	<i>i2cp</i>	pointer to the I2CDriver object
in	<i>addr</i>	slave device address (7 bits) without R/W bit
out	<i>rxbuf</i>	pointer to 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.

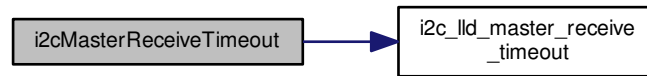
Return values

<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.

Function Class:

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

Here is the call graph for this function:



7.14.6.8 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	<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.

7.14.6.9 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.

7.14.6.10 void i2c_llid_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:



7.14.6.11 void i2c_llid_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.

7.14.6.12 void i2c_llid_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.

7.14.6.13 msg_t i2c_llid_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

Parameters

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.

7.14.6.14 `msg_t i2c_ild_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>	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.

7.14.7 Variable Documentation**7.14.7.1 I2CDriver I2CD1**

I2C1 driver identifier.

7.15 I2S Driver

Generic I2S Driver.

7.15.1 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`.

7.15.2 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_llc_init` (void)
Low level I2S driver initialization.
- void `i2s_llc_start` (I2SDriver *i2sp)
Configures and activates the I2S peripheral.
- void `i2s_llc_stop` (I2SDriver *i2sp)
Deactivates the I2S peripheral.
- void `i2s_llc_start_exchange` (I2SDriver *i2sp)
Starts a I2S data exchange.
- void `i2s_llc_stop_exchange` (I2SDriver *i2sp)
Stops the ongoing data exchange.

Enumerations

Variables

- `I2SDriver I2SD1`
I2S2 driver identifier.

7.15.3 Macro Definition Documentation

7.15.3.1 #define i2sStartExchange(i2sp)

Value:

```
{
    i2s_llc_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.

7.15.3.2 #define i2sStopExchange(i2sp)

Value:

```

{
    i2s_ll_d_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.

7.15.3.3 #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.

7.15.3.4 #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.

7.15.3.5 #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`.

7.15.4 Typedef Documentation

7.15.4.1 typedef struct I2SDriver I2SDriver

Type of a structure representing an I2S driver.

7.15.4.2 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

7.15.5 Enumeration Type Documentation

7.15.5.1 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.

7.15.6 Function Documentation

7.15.6.1 void i2sInit (void)

I2S Driver initialization.

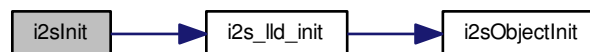
Note

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:



7.15.6.2 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.

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

Configures and activates the I2S peripheral.

Parameters

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



7.15.6.4 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:



7.15.6.5 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.

7.15.6.6 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	<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.

7.15.6.7 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:

**7.15.6.8 void i2s_lld_start (I2SDriver * i2sp)**

Configures and activates the I2S peripheral.

Parameters

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

Function Class:

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

7.15.6.9 void i2s_llid_stop (I2SDriver * i2sp)

Deactivates the I2S peripheral.

Parameters

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

Function Class:

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

7.15.6.10 void i2s_llid_start_exchange (I2SDriver * i2sp)

Starts a I2S data exchange.

Parameters

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

Function Class:

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

7.15.6.11 void i2s_llid_stop_exchange (I2SDriver * i2sp)

Stops the ongoing data exchange.

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

Parameters

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

Function Class:

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

7.15.7 Variable Documentation**7.15.7.1 I2SDriver I2SD1**

I2S2 driver identifier.

7.16 ICU Driver

Generic ICU Driver.

7.16.1 Detailed Description

Generic ICU Driver.

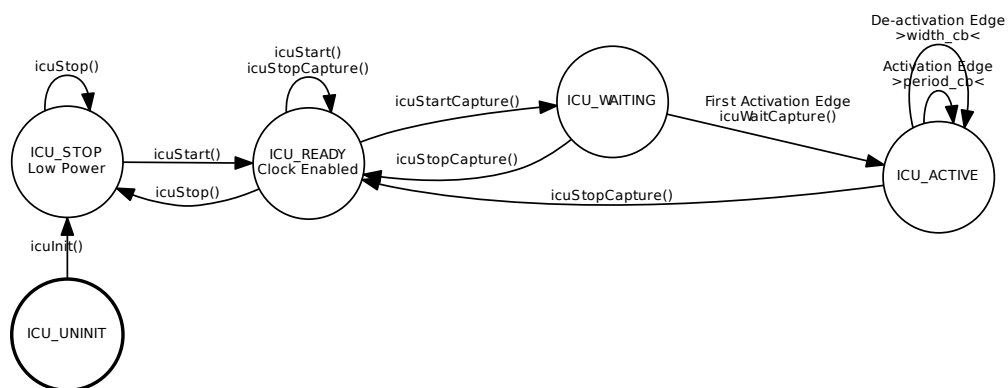
This module implements a generic ICU (Input Capture Unit) driver. The purpose of the driver is to measure period and duty cycle of an input digital signal (PWM input).

Precondition

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

7.16.2 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).



7.16.3 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 icuStartCaptureI(icup)`
Starts the input capture.
- `#define icuStopCaptureI(icup)`
Stops the input capture.
- `#define icuEnableNotificationsI(icup) icu_lld_enable_notifications(icup)`
Enables notifications.
- `#define icuDisableNotificationsI(icup) icu_lld_disable_notifications(icup)`
Disables notifications.
- `#define icuAreNotificationsEnabledX(icup) icu_lld_are_notifications_enabled(icup)`
Check on notifications status.
- `#define icuGetWidthX(icup) icu_lld_get_width(icup)`
Returns the width of the latest pulse.
- `#define icuGetPeriodX(icup) icu_lld_get_period(icup)`
Returns the width of the latest cycle.

Low level driver helper macros

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

PLATFORM configuration options

- `#define PLATFORM_ICU_USE_ICU1 FALSE`
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 [iculnit](#) (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

Variables

- `ICUDriver ICUD1`
ICUD1 driver identifier.

7.16.4 Macro Definition Documentation

7.16.4.1 #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.

7.16.4.2 #define icuStopCapture(icup)

Value:

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

Stops 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.

7.16.4.3 #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 <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.

7.16.4.4 #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 <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.

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

Check on notifications status.

Parameters

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

7.16.4.6 #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.

7.16.4.7 #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.

7.16.4.8 #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.

7.16.4.9 #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.

7.16.4.10 #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.

7.16.4.11 `#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`.

7.16.4.12 `#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.

7.16.4.13 `#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.

7.16.4.14 `#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.

7.16.5 Typedef Documentation

7.16.5.1 `typedef struct ICUDriver ICUDriver`

Type of a structure representing an ICU driver.

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

ICU notification callback type.

Parameters

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

7.16.5.3 `typedef uint32_t icufreq_t`

ICU frequency type.

7.16.5.4 `typedef uint32_t icucnt_t`

ICU counter type.

7.16.6 Enumeration Type Documentation

7.16.6.1 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.

7.16.6.2 enum icumode_t

ICU driver mode.

Enumerator

ICU_INPUT_ACTIVE_HIGH Trigger on rising edge.

ICU_INPUT_ACTIVE_LOW Trigger on falling edge.

7.16.7 Function Documentation

7.16.7.1 void icuInit (void)

ICU Driver initialization.

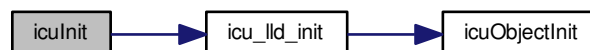
Note

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

Function Class:

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

Here is the call graph for this function:



7.16.7.2 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.

7.16.7.3 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:

**7.16.7.4 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:



7.16.7.5 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.

7.16.7.6 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.
<i>true</i>	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:

**7.16.7.7 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.

7.16.7.8 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 ICUDriver object
----	-------------	-------------------------------------------------

Function Class:

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

7.16.7.9 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.

7.16.7.10 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:



7.16.7.11 void icu_lld_start (ICUDriver * icup)

Configures and activates the ICU peripheral.

Parameters

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

Function Class:

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

7.16.7.12 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.

7.16.7.13 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.

7.16.7.14 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.

7.16.7.15 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.

7.16.7.16 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.

7.16.7.17 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.

7.16.8 Variable Documentation

7.16.8.1 ICUDriver ICUD1

ICUD1 driver identifier.

Note

The driver ICUD1 allocates the complex timer TIM1 when enabled.

7.17 MAC Driver

Generic MAC Driver.

7.17.1 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

Variables

- `MACDriver ETHD1`
MAC1 driver identifier.

7.17.2 Macro Definition Documentation

7.17.2.1 `#define MAC_USE_ZERO_COPY FALSE`

Enables an event sources for incoming packets.

7.17.2.2 `#define MAC_USE_EVENTS TRUE`

Enables an event sources for incoming packets.

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

Returns the received frames event source.

Parameters

in	<i>macp</i>	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.

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

Writes to a transmit descriptor's stream.

Parameters

in	<i>tdp</i>	pointer to a <code>MACTransmitDescriptor</code> 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:

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

7.17.2.5 `#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.

7.17.2.6 `#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.

7.17.2.7 `#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

in	<i>rdp</i>	pointer to a MACReceiveDescriptor 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

<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.

7.17.2.8 `#define MAC_SUPPORTS_ZERO_COPY TRUE`

This implementation supports the zero-copy mode API.

7.17.2.9 `#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`.

7.17.3 Typedef Documentation

7.17.3.1 `typedef struct MACDriver MACDriver`

Type of a structure representing a MAC driver.

7.17.4 Enumeration Type Documentation

7.17.4.1 `enum macstate_t`

Driver state machine possible states.

Enumerator

MAC_UNINIT Not initialized.

MAC_STOP Stopped.

MAC_ACTIVE Active.

7.17.5 Function Documentation

7.17.5.1 void macInit (void)

MAC Driver initialization.

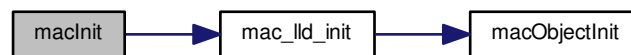
Note

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:



7.17.5.2 void macObjectInit (MACDriver * macp)

Initialize the standard part of a `MACDriver` structure.

Parameters

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

Function Class:

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

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

Configures and activates the MAC peripheral.

Parameters

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



7.17.5.4 void macStop (MACDriver * macp)

Deactivates the MAC peripheral.

Parameters

in	macp	pointer to the MACDriver object
----	------	-------------------------------------------------

Function Class:

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

Here is the call graph for this function:



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

Returns

The operation status.

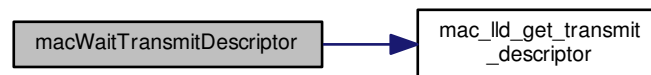
Return values

<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:



7.17.5.6 void macReleaseTransmitDescriptor (**MACTransmitDescriptor** * *tdp*)

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

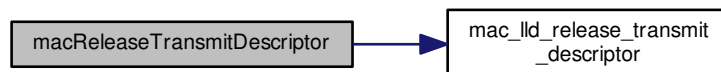
Parameters

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:



7.17.5.7 msg_t macWaitReceiveDescriptor (**MACDriver** * *macp*, **MACReceiveDescriptor** * *rdp*, **sys_time_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.

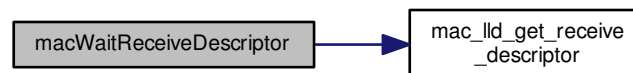
Return values

<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:



7.17.5.8 void macReleaseReceiveDescriptor ([MACReceiveDescriptor](#) * *rdp*)

Releases a receive descriptor.

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

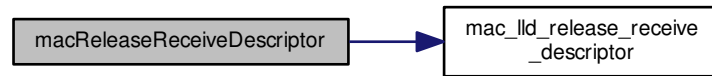
Parameters

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

Function Class:

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

Here is the call graph for this function:



7.17.5.9 bool macPollLinkStatus (MACDriver * macp)

Updates and returns the link status.

Parameters

in	macp	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:



7.17.5.10 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:



7.17.5.11 void mac_lld_start (MACDriver * macp)

Configures and activates the MAC peripheral.

Parameters

in	macp	pointer to the MACDriver object
----	------	-------------------------------------------------

Function Class:

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

7.17.5.12 void mac_lld_stop (MACDriver * macp)

Deactivates the MAC peripheral.

Parameters

in	macp	pointer to the MACDriver object
----	------	-------------------------------------------------

Function Class:

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

7.17.5.13 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	macp	pointer to the MACDriver object
out	tdp	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.

7.17.5.14 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

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

Function Class:

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

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

Returns a receive descriptor.

Parameters

in	<i>macp</i>	pointer to the MACDriver object
out	<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.

7.17.5.16 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

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

Function Class:

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

7.17.5.17 bool mac_llc_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.

7.17.5.18 size_t mac_llc_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.

7.17.5.19 size_t mac_llc_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.

7.17.5.20 `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 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 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

<i>NULL</i>	if the buffer chain has been entirely scanned.
-------------	------------------------------------------------

Function Class:

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

7.17.5.21 `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 MACReceiveDescriptor 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

<i>NULL</i>	if the buffer chain has been entirely scanned.
-------------	------------------------------------------------

Function Class:

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

7.17.6 Variable Documentation**7.17.6.1 MACDriver ETHD1**

MAC1 driver identifier.

7.18 HAL

Hardware Abstraction Layer.

7.18.1 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.

7.18.2 HAL Device Drivers Architecture

The HAL contains several kind of modules:

- Normal Device Drivers
- Complex Device Drivers
- Interfaces
- Inner Code

7.18.3 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:

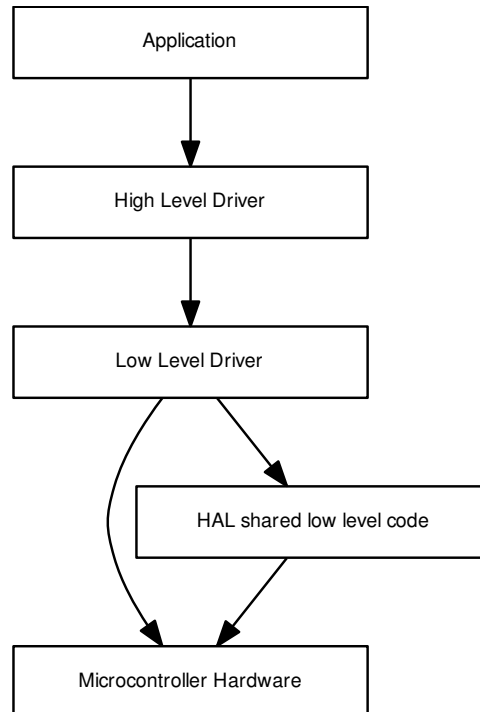
- `<driver>.c`, the HLD implementation file. This file must be included in the Makefile in order to use the driver.
- `<driver>.h`, the HLD header file. This file is implicitly included by the HAL header file [hal.h](#).

- 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.

7.18.3.1 Diagram



7.18.4 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.

7.18.5 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.

7.18.6 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.

- [Support Code](#)

HAL Support Code.

- [OSAL](#)

Operating System Abstraction Layer.

7.19 Configuration

HAL Configuration.

7.19.1 Detailed Description

HAL Configuration.

The file `halconf.h` contains the high level settings for all the drivers supported by the HAL. The low level, platform dependent, settings are contained in the `mcuconf.h` file instead and are describe in the various platforms reference manuals.

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_QSPI TRUE`
Enables the QSPI 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.
- #define `HAL_USE_WDG` `TRUE`
Enables the WDG 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_MMC_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.
- #define `SERIAL_USB_BUFFERS_NUMBER` 2
Serial over USB number of buffers.

SPI driver related setting

- #define `SPI_USE_WAIT` TRUE
Enables synchronous APIs.
- #define `SPI_USE_MUTUAL_EXCLUSION` TRUE
Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.

UART driver related setting

- #define `UART_USE_WAIT` TRUE
Enables synchronous APIs.
- #define `UART_USE_MUTUAL_EXCLUSION` TRUE
Enables the `uartAcquireBus()` and `uartReleaseBus()` APIs.

USB driver related setting

- #define `USB_USE_WAIT` TRUE
Enables synchronous APIs.

7.19.2 Macro Definition Documentation

7.19.2.1 #define HAL_USE_PAL TRUE

Enables the PAL subsystem.

7.19.2.2 #define HAL_USE_ADC TRUE

Enables the ADC subsystem.

7.19.2.3 #define HAL_USE_CAN TRUE

Enables the CAN subsystem.

7.19.2.4 #define HAL_USE_DAC FALSE

Enables the DAC subsystem.

7.19.2.5 #define HAL_USE_EXT TRUE

Enables the EXT subsystem.

7.19.2.6 `#define HAL_USE_GPT TRUE`

Enables the GPT subsystem.

7.19.2.7 `#define HAL_USE_I2C TRUE`

Enables the I2C subsystem.

7.19.2.8 `#define HAL_USE_I2S TRUE`

Enables the I2S subsystem.

7.19.2.9 `#define HAL_USE_ICU TRUE`

Enables the ICU subsystem.

7.19.2.10 `#define HAL_USE_MAC TRUE`

Enables the MAC subsystem.

7.19.2.11 `#define HAL_USE_MMC_SPI TRUE`

Enables the MMC_SPI subsystem.

7.19.2.12 `#define HAL_USE_PWM TRUE`

Enables the PWM subsystem.

7.19.2.13 `#define HAL_USE_QSPI TRUE`

Enables the QSPI subsystem.

7.19.2.14 `#define HAL_USE_RTC TRUE`

Enables the RTC subsystem.

7.19.2.15 `#define HAL_USE_SDC TRUE`

Enables the SDC subsystem.

7.19.2.16 `#define HAL_USE_SERIAL TRUE`

Enables the SERIAL subsystem.

7.19.2.17 `#define HAL_USE_SERIAL_USB TRUE`

Enables the SERIAL over USB subsystem.

7.19.2.18 #define HAL_USE_SPI TRUE

Enables the SPI subsystem.

7.19.2.19 #define HAL_USE_UART TRUE

Enables the UART subsystem.

7.19.2.20 #define HAL_USE_USB TRUE

Enables the USB subsystem.

7.19.2.21 #define HAL_USE_WDG TRUE

Enables the WDG subsystem.

7.19.2.22 #define ADC_USE_WAIT TRUE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.19.2.23 #define ADC_USE_MUTUAL_EXCLUSION TRUE

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

Note

Disabling this option saves both code and data space.

7.19.2.24 #define CAN_USE_SLEEP_MODE TRUE

Sleep mode related APIs inclusion switch.

7.19.2.25 #define I2C_USE_MUTUAL_EXCLUSION TRUE

Enables the mutual exclusion APIs on the I2C bus.

7.19.2.26 #define MAC_USE_ZERO_COPY TRUE

Enables an event sources for incoming packets.

7.19.2.27 #define MAC_USE_EVENTS TRUE

Enables an event sources for incoming packets.

7.19.2.28 #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.

7.19.2.29 #define SDC_INIT_RETRY 100

Number of initialization attempts before rejecting the card.

Note

Attempts are performed at 10mS intervals.

7.19.2.30 #define SDC_MMC_SUPPORT TRUE

Include support for MMC cards.

Note

MMC support is not yet implemented so this option must be kept at `FALSE`.

7.19.2.31 #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.

7.19.2.32 #define SERIAL_DEFAULT_BITRATE 38400

Default bit rate.

Configuration parameter, this is the baud rate selected for the default configuration.

7.19.2.33 #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.

7.19.2.34 #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.

7.19.2.35 `#define SERIAL_USB_BUFFERS_NUMBER 2`

Serial over USB number of buffers.

Note

The default is 2 buffers.

7.19.2.36 `#define SPI_USE_WAIT TRUE`

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.19.2.37 `#define SPI_USE_MUTUAL_EXCLUSION TRUE`

Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.

Note

Disabling this option saves both code and data space.

7.19.2.38 `#define UART_USE_WAIT TRUE`

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.19.2.39 `#define UART_USE_MUTUAL_EXCLUSION TRUE`

Enables the `uartAcquireBus()` and `uartReleaseBus()` APIs.

Note

Disabling this option saves both code and data space.

7.19.2.40 `#define USB_USE_WAIT TRUE`

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.20 Normal Drivers

HAL Normal Drivers.

7.20.1 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.
- [QSPI Driver](#)
Generic QSPI Driver.
- [RTC Driver](#)
Generic RTC Driver.
- [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.
- [WDG Driver](#)
Generic WDG Driver.

7.21 Complex Drivers

HAL Complex Drivers.

7.21.1 Detailed Description

HAL Complex Drivers.

Modules

- [MMC over SPI Driver](#)
Generic MMC driver.
- [Serial over USB Driver](#)
Serial over USB Driver.

7.22 Interfaces

HAL Interfaces.

7.22.1 Detailed Description

HAL Interfaces.

Modules

- [Abstract NOR Flash Class](#)
Generic NOR Flash interface.
- [Abstract I/O Channel](#)
- [Abstract Files](#)
- [Abstract I/O Block Device](#)
- [Abstract Streams](#)

7.23 Inner Code

HAL Inner Code.

7.23.1 Detailed Description

HAL Inner Code.

Modules

- [I/O Buffers Queues](#)
- [I/O Bytes Queues](#)
- [MMC/SD Block Device](#)

7.24 Support Code

HAL Support Code.

7.24.1 Detailed Description

HAL Support Code.

Modules

- [MII/RMII Header](#)
MII/RMII Support Header.
- [USB CDC Header](#)
USB CDC Support Header.

7.25 OSAL

Operating System Abstraction Layer.

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 thin 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.

7.26 MII/RMII Header

IIII/RMII Support Header.

7.26.1 Detailed Description

IIII/RMII Support Header.

This header contains definitions and types related to MII/RMII.

Generic MII registers

- #define MII_BMCR 0x00
- #define MII_BMSR 0x01
- #define MII_PHYSID1 0x02
- #define MII_PHYSID2 0x03
- #define MII_ADVERTISE 0x04
- #define MII_LPA 0x05
- #define MII_EXPANSION 0x06
- #define MII_ANNPTR 0x07
- #define MII_CTRL1000 0x09
- #define MII_STAT1000 0x0a
- #define MII_ESTATUS 0x0f
- #define MII_PHYSTS 0x10
- #define MII_MICR 0x11
- #define MII_DCOUNTER 0x12
- #define MII_FCSCOUNTER 0x13
- #define MII_NWAYTEST 0x14
- #define MII_RERRCOUNTER 0x15
- #define MII_SREVISION 0x16
- #define MII_RESV1 0x17
- #define MII_LBRRERROR 0x18
- #define MII_PHYADDR 0x19
- #define MII_RESV2 0x1a
- #define MII_TPISTATUS 0x1b
- #define MII_NCONFIG 0x1c

Basic mode control register

- #define BMCR_RESV 0x007f
- #define BMCR_CTST 0x0080
- #define BMCR_FULLDPLX 0x0100
- #define BMCR_ANRESTART 0x0200
- #define BMCR_ISOLATE 0x0400
- #define BMCR_PDOWN 0x0800
- #define BMCR_ANENABLE 0x1000
- #define BMCR_SPEED100 0x2000
- #define BMCR_LOOPBACK 0x4000
- #define BMCR_RESET 0x8000

Basic mode status register

- #define `BMSR_ERCAP` 0x0001
- #define `BMSR_JCD` 0x0002
- #define `BMSR_LSTATUS` 0x0004
- #define `BMSR_ANEGCAPABLE` 0x0008
- #define `BMSR_RFAULT` 0x0010
- #define `BMSR_ANEGCOMPLETE` 0x0020
- #define `BMSR_MFPRESUPPCAP` 0x0040
- #define `BMSR_RESV` 0x0780
- #define `BMSR_10HALF` 0x0800
- #define `BMSR_10FULL` 0x1000
- #define `BMSR_100HALF` 0x2000
- #define `BMSR_100FULL` 0x4000
- #define `BMSR_100BASE4` 0x8000

Advertisement control register

- #define `ADVERTISE_SLCT` 0x001f
- #define `ADVERTISE_CSMA` 0x0001
- #define `ADVERTISE_10HALF` 0x0020
- #define `ADVERTISE_10FULL` 0x0040
- #define `ADVERTISE_100HALF` 0x0080
- #define `ADVERTISE_100FULL` 0x0100
- #define `ADVERTISE_100BASE4` 0x0200
- #define `ADVERTISE_PAUSE_CAP` 0x0400
- #define `ADVERTISE_PAUSE_ASYM` 0x0800
- #define `ADVERTISE_RESV` 0x1000
- #define `ADVERTISE_RFAULT` 0x2000
- #define `ADVERTISE_LPACK` 0x4000
- #define `ADVERTISE_NPAGE` 0x8000
- #define `ADVERTISE_FULL`
- #define `ADVERTISE_ALL`

Link partner ability register

- #define `LPA_SLCT` 0x001f
- #define `LPA_10HALF` 0x0020
- #define `LPA_10FULL` 0x0040
- #define `LPA_100HALF` 0x0080
- #define `LPA_100FULL` 0x0100
- #define `LPA_100BASE4` 0x0200
- #define `LPA_PAUSE_CAP` 0x0400
- #define `LPA_PAUSE_ASYM` 0x0800
- #define `LPA_RESV` 0x1000
- #define `LPA_RFAULT` 0x2000
- #define `LPA_LPACK` 0x4000
- #define `LPA_NPAGE` 0x8000
- #define `LPA_DUPLEX` (`LPA_10FULL` | `LPA_100FULL`)
- #define `LPA_100` (`LPA_100FULL` | `LPA_100HALF` | `LPA_100BASE4`)

Expansion register for auto-negotiation

- #define `EXPANSION_NWAY` 0x0001
- #define `EXPANSION_LCWP` 0x0002
- #define `EXPANSION_ENABLENPAGE` 0x0004
- #define `EXPANSION_NPCAPABLE` 0x0008
- #define `EXPANSION_MFAULTS` 0x0010
- #define `EXPANSION_RESV` 0xffe0

N-way test register

- #define `NWAYTEST_RESV1` 0x00ff
- #define `NWAYTEST_LOOPBACK` 0x0100
- #define `NWAYTEST_RESV2` 0xfe00

PHY identifiers

- #define `MII_DM9161_ID` 0x0181b8a0
- #define `MII_AM79C875_ID` 0x00225540
- #define `MII_KS8721_ID` 0x00221610
- #define `MII_STE101P_ID` 0x00061C50
- #define `MII_DP83848I_ID` 0x20005C90
- #define `MII_LAN8710A_ID` 0x0007C0F1
- #define `MII_LAN8720_ID` 0x0007C0F0
- #define `MII_LAN8742A_ID` 0x0007C130

7.26.2 Macro Definition Documentation

7.26.2.1 #define `MII_BMCR` 0x00

Basic mode control register.

7.26.2.2 #define `MII_BMSR` 0x01

Basic mode status register.

7.26.2.3 #define `MII_PHYSID1` 0x02

PHYS ID 1.

7.26.2.4 #define `MII_PHYSID2` 0x03

PHYS ID 2.

7.26.2.5 #define `MII_ADVERTISE` 0x04

Advertisement control reg.

7.26.2.6 #define `MII_LPA` 0x05

Link partner ability reg.

7.26.2.7 `#define MII_EXPANSION 0x06`

Expansion register.

7.26.2.8 `#define MII_ANNPTR 0x07`

1000BASE-T control.

7.26.2.9 `#define MII_CTRL1000 0x09`

1000BASE-T control.

7.26.2.10 `#define MII_STAT1000 0x0a`

1000BASE-T status.

7.26.2.11 `#define MII_ESTATUS 0x0f`

Extended Status.

7.26.2.12 `#define MII_PHYSTS 0x10`

PHY Status register.

7.26.2.13 `#define MII_MICR 0x11`

MII Interrupt ctrl register.

7.26.2.14 `#define MII_DCOUNTER 0x12`

Disconnect counter.

7.26.2.15 `#define MII_FCSCOUNTER 0x13`

False carrier counter.

7.26.2.16 `#define MII_NWAYTEST 0x14`

N-way auto-neg test reg.

7.26.2.17 `#define MII_RERRCOUNTER 0x15`

Receive error counter.

7.26.2.18 `#define MII_SREVISION 0x16`

Silicon revision.

7.26.2.19 `#define MII_RESV1 0x17`

Reserved.

7.26.2.20 `#define MII_LBRERROR 0x18`

Lpback, rx, bypass error.

7.26.2.21 `#define MII_PHYADDR 0x19`

PHY address.

7.26.2.22 `#define MII_RESV2 0x1a`

Reserved.

7.26.2.23 `#define MII_TPISTATUS 0x1b`

TPI status for 10Mbps.

7.26.2.24 `#define MII_NCONFIG 0x1c`

Network interface config.

7.26.2.25 `#define BMCR_RESV 0x007f`

Unused.

7.26.2.26 `#define BMCR_CTST 0x0080`

Collision test.

7.26.2.27 `#define BMCR_FULLDPLX 0x0100`

Full duplex.

7.26.2.28 `#define BMCR_ANRESTART 0x0200`

Auto negotiation restart.

7.26.2.29 `#define BMCR_ISOLATE 0x0400`

Disconnect DP83840 from MII.

7.26.2.30 `#define BMCR_PDOWN 0x0800`

Powerdown.

7.26.2.31 `#define BMCR_ANENABLE 0x1000`

Enable auto negotiation.

7.26.2.32 `#define BMCR_SPEED100 0x2000`

Select 100Mbps.

7.26.2.33 `#define BMCR_LOOPBACK 0x4000`

TXD loopback bit.

7.26.2.34 `#define BMCR_RESET 0x8000`

Reset.

7.26.2.35 `#define BMSR_ERCAP 0x0001`

Ext-reg capability.

7.26.2.36 `#define BMSR_JCD 0x0002`

Jabber detected.

7.26.2.37 `#define BMSR_LSTATUS 0x0004`

Link status.

7.26.2.38 `#define BMSR_ANEGCAPABLE 0x0008`

Able to do auto-negotiation.

7.26.2.39 `#define BMSR_RFAULT 0x0010`

Remote fault detected.

7.26.2.40 `#define BMSR_ANEGCOMPLETE 0x0020`

Auto-negotiation complete.

7.26.2.41 `#define BMSR_MFPRESUPPCAP 0x0040`

Able to suppress preamble.

7.26.2.42 `#define BMSR_RESV 0x0780`

Unused.

7.26.2.43 `#define BMSR_10HALF 0x0800`

Can do 10mbps, half-duplex.

7.26.2.44 `#define BMSR_10FULL 0x1000`

Can do 10mbps, full-duplex.

7.26.2.45 `#define BMSR_100HALF 0x2000`

Can do 100mbps, half-duplex.

7.26.2.46 `#define BMSR_100FULL 0x4000`

Can do 100mbps, full-duplex.

7.26.2.47 `#define BMSR_100BASE4 0x8000`

Can do 100mbps, 4k packets.

7.26.2.48 `#define ADVERTISE_SLCT 0x001f`

Selector bits.

7.26.2.49 `#define ADVERTISE_CSMA 0x0001`

Only selector supported.

7.26.2.50 `#define ADVERTISE_10HALF 0x0020`

Try for 10mbps half-duplex.

7.26.2.51 `#define ADVERTISE_10FULL 0x0040`

Try for 10mbps full-duplex.

7.26.2.52 `#define ADVERTISE_100HALF 0x0080`

Try for 100mbps half-duplex.

7.26.2.53 `#define ADVERTISE_100FULL 0x0100`

Try for 100mbps full-duplex.

7.26.2.54 `#define ADVERTISE_100BASE4 0x0200`

Try for 100mbps 4k packets.

7.26.2.55 `#define ADVERTISE_PAUSE_CAP 0x0400`

Try for pause.

7.26.2.56 `#define ADVERTISE_PAUSE_ASYM 0x0800`

Try for asymmetric pause.

7.26.2.57 `#define ADVERTISE_RESV 0x1000`

Unused.

7.26.2.58 `#define ADVERTISE_RFAULT 0x2000`

Say we can detect faults.

7.26.2.59 `#define ADVERTISE_LPACK 0x4000`

Ack link partners response.

7.26.2.60 `#define ADVERTISE_NPAGE 0x8000`

Next page bit.

7.26.2.61 `#define LPA_SLCT 0x001f`

Same as advertise selector.

7.26.2.62 `#define LPA_10HALF 0x0020`

Can do 10mbps half-duplex.

7.26.2.63 `#define LPA_10FULL 0x0040`

Can do 10mbps full-duplex.

7.26.2.64 `#define LPA_100HALF 0x0080`

Can do 100mbps half-duplex.

7.26.2.65 `#define LPA_100FULL 0x0100`

Can do 100mbps full-duplex.

7.26.2.66 `#define LPA_100BASE4 0x0200`

Can do 100mbps 4k packets.

7.26.2.67 `#define LPA_PAUSE_CAP 0x0400`

Can pause.

7.26.2.68 `#define LPA_PAUSE_ASYM 0x0800`

Can pause asymmetrically.

7.26.2.69 `#define LPA_RESV 0x1000`

Unused.

7.26.2.70 `#define LPA_RFAULT 0x2000`

Link partner faulted.

7.26.2.71 `#define LPA_LPACK 0x4000`

Link partner acked us.

7.26.2.72 `#define LPA_NPAGE 0x8000`

Next page bit.

7.26.2.73 `#define EXPANSION_NWAY 0x0001`

Can do N-way auto-nego.

7.26.2.74 `#define EXPANSION_LCWP 0x0002`

Got new RX page code word.

7.26.2.75 `#define EXPANSION_ENABLENPAGE 0x0004`

This enables npage words.

7.26.2.76 `#define EXPANSION_NPCAPABLE 0x0008`

Link partner supports npage.

7.26.2.77 `#define EXPANSION_MFAULTS 0x0010`

Multiple faults detected.

7.26.2.78 `#define EXPANSION_RESV 0xffe0`

Unused.

7.26.2.79 `#define NWAYTEST_RESV1 0x00ff`

Unused.

7.26.2.80 `#define NWAYTEST_LOOPBACK 0x0100`

Enable loopback for N-way.

7.26.2.81 `#define NWAYTEST_RESV2 0xfe00`

Unused.

7.27 MMC over SPI Driver

Generic MMC driver.

7.27.1 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`.

7.27.2 Driver State Machine

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

7.27.3 Driver Operations

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

Macros

- `#define _mmc_driver_methods _mmc_sd_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 `MMConfig`
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 **recvr1** (MMCDriver *mmcp)
Receives a single byte response.
- static uint8_t **recvr3** (MMCDriver *mmcp, uint8_t *buffer)
Receives a three byte response.
- static uint8_t **send_command_R1** (MMCDriver *mmcp, uint8_t cmd, uint32_t arg)
Sends a command 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 **mmclnit** (void)
MMC over SPI driver initialization.
- void **mmcObjectInit** (MMCDriver *mmcp)
Initializes an instance.
- void **mmcStart** (MMCDriver *mmcp, const MMCCfg *config)
Configures and activates the MMC peripheral.
- void **mmcStop** (MMCDriver *mmcp)
Disables the MMC peripheral.
- bool **mmcConnect** (MMCDriver *mmcp)
Performs the initialization procedure on the inserted card.
- bool **mmcDisconnect** (MMCDriver *mmcp)
Brings the driver in a state safe for card removal.
- bool **mmcStartSequentialRead** (MMCDriver *mmcp, uint32_t startblk)
Starts a sequential read.
- bool **mmcSequentialRead** (MMCDriver *mmcp, uint8_t *buffer)
Reads a block within a sequential read operation.
- bool **mmcStopSequentialRead** (MMCDriver *mmcp)
Stops a sequential read gracefully.
- bool **mmcStartSequentialWrite** (MMCDriver *mmcp, uint32_t startblk)
Starts a sequential write.
- 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 $x^7 + x^3 + 1$).

7.27.4 Macro Definition Documentation

7.27.4.1 `#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.

7.27.4.2 `#define _mmc_driver_methods _mmc_sd_block_device_methods`

[MMCDriver](#) specific methods.

7.27.4.3 `#define mmcIsCardInserted(mmcp) mmc_lld_is_card_inserted(mmcp)`

Returns the card insertion status.

Note

This macro wraps a low level function named `sdc_lld_is_card_inserted()`, this function must be provided by the application because it is not part of the SDC driver.

Parameters

in	<code>mmcp</code>	pointer to the MMCDriver object
----	-------------------	-------------------------------------------------

Returns

The card state.

Return values

<code>FALSE</code>	card not inserted.
<code>TRUE</code>	card inserted.

Function Class:

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

7.27.4.4 `#define mmcIsWriteProtected(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.

7.27.5 Function Documentation

7.27.5.1 `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

7.27.5.2 `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:



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

Sends a command header.

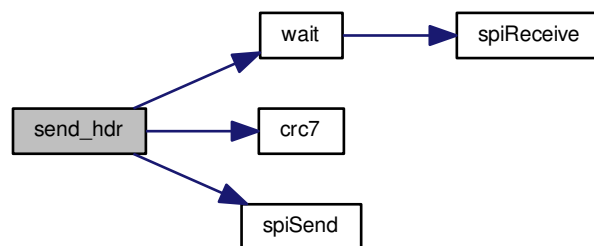
Parameters

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:



7.27.5.4 static uint8_t recvr1 (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

<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:

**7.27.5.5 static uint8_t recvr3 (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.

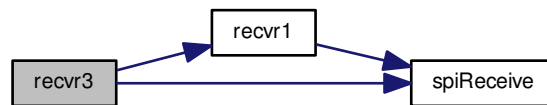
Return values

<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:



7.27.5.6 `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 MMCDriver object
in	<i>cmd</i>	the command id
in	<i>arg</i>	the command argument

Returns

The response as an `uint8_t` value.

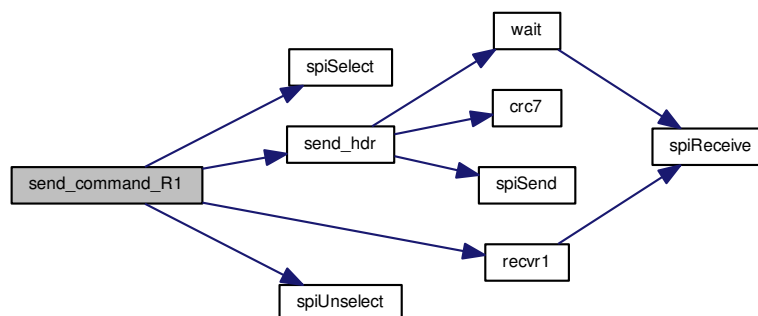
Return values

<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:



7.27.5.7 `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 <code>uint8_t</code> buffer

Returns

The first byte of the response (R1) as an `uint8_t` value.

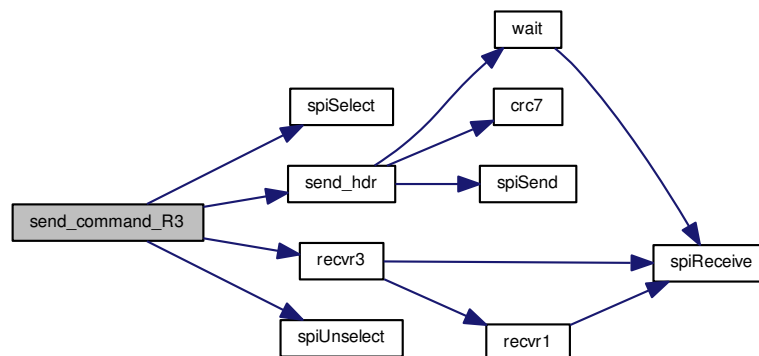
Return values

<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:



7.27.5.8 `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.

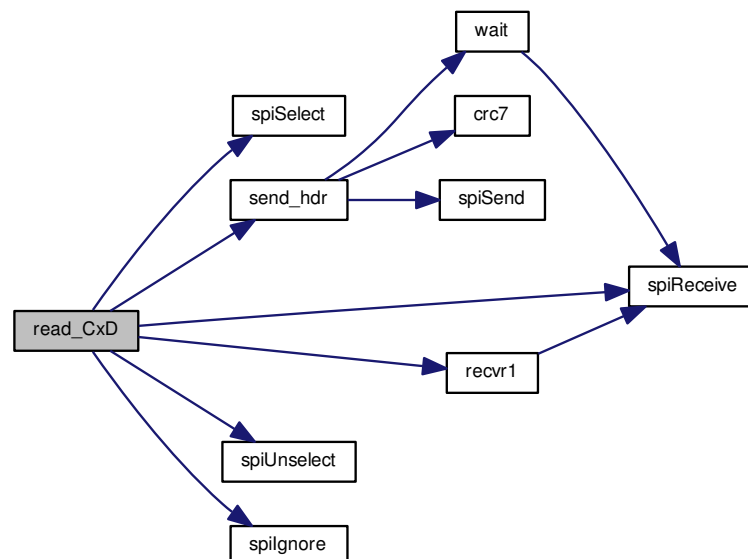
Return values

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

Function Class:

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

Here is the call graph for this function:

**7.27.5.9 static void sync (MMCDriver * mmcp) [static]**

Waits that the card reaches an idle state.

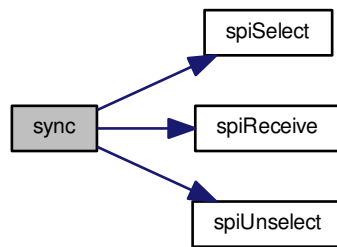
Parameters

in	<i>mmcp</i>	pointer to the <code>MMCDriver</code> object
----	-------------	----------------------------------------------

Function Class:

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

Here is the call graph for this function:



7.27.5.10 void mmclnit (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.

7.27.5.11 void mmcObjectInit (MMCDriver * mmcp)

Initializes an instance.

Parameters

out	<i>mmcp</i>	pointer to the <code>MMCDriver</code> object
-----	-------------	----------------------------------------------

Function Class:

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

7.27.5.12 void mmcStart (MMCDriver * mmcp, const MMCCConfig * config)

Configures and activates the MMC peripheral.

Parameters

in	<i>mmcp</i>	pointer to the <code>MMCDriver</code> object
in	<i>config</i>	pointer to the <code>MMCCConfig</code> object.

Function Class:

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

7.27.5.13 void mmcStop (MMCDriver * mmcp)

Disables the MMC peripheral.

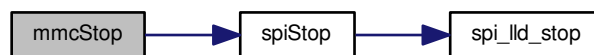
Parameters

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:

**7.27.5.14 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

in	<i>mmcp</i>	pointer to the MMCDriver object
----	-------------	-------------------------------------------------

Returns

The operation status.

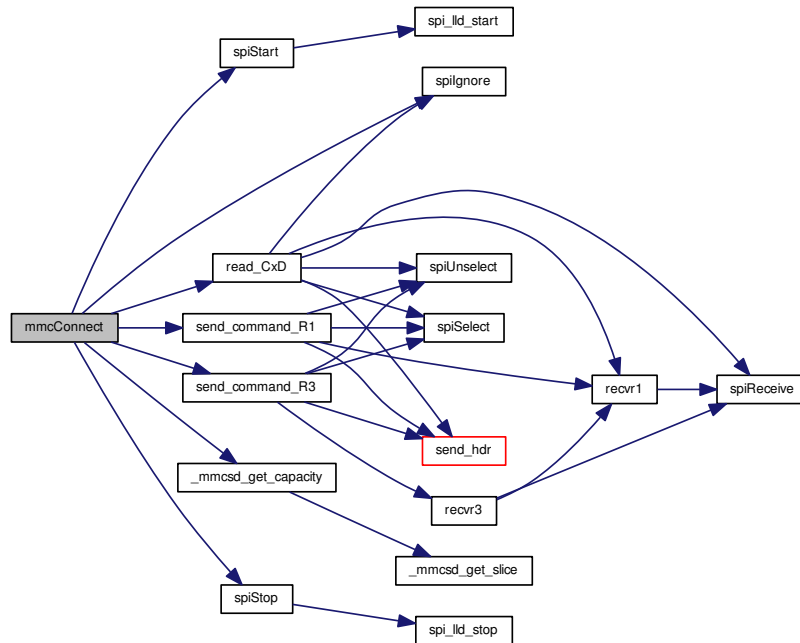
Return values

<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:



7.27.5.15 bool mmcDisconnect (MMCDriver * mmcp)

Brings the driver in a state safe for card removal.

Parameters

in	mmcp	pointer to the MMCDriver object
----	------	-------------------------------------------------

Returns

The operation status.

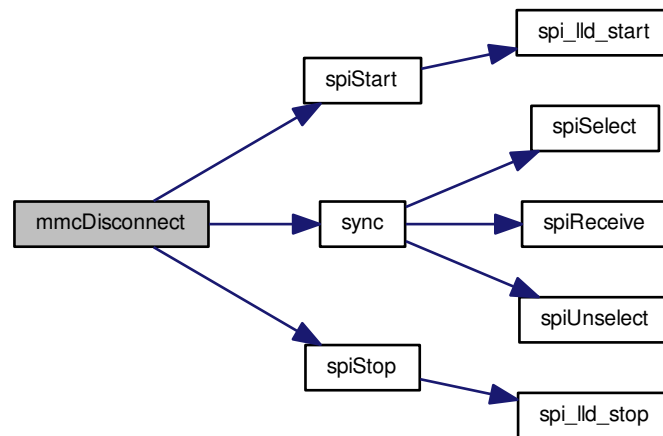
Return values

<i>HAL_SUCCESS</i>	the operation succeeded and the driver is now in the <code>MMC_INSERTED</code> 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:



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

Starts a sequential read.

Parameters

in	<i>mmcp</i>	pointer to the <code>MMCDriver</code> object
in	<i>startblk</i>	first block to read

Returns

The operation status.

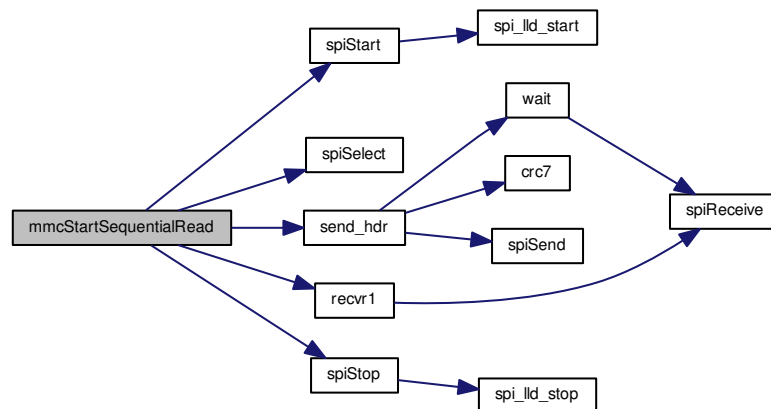
Return values

<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:



7.27.5.17 bool mmcSequentialRead (MMCDriver * mmcp, uint8_t * buffer)

Reads a block within a sequential read operation.

Parameters

in	<i>mmcp</i>	pointer to the MMCDriver object
out	<i>buffer</i>	pointer to the read buffer

Returns

The operation status.

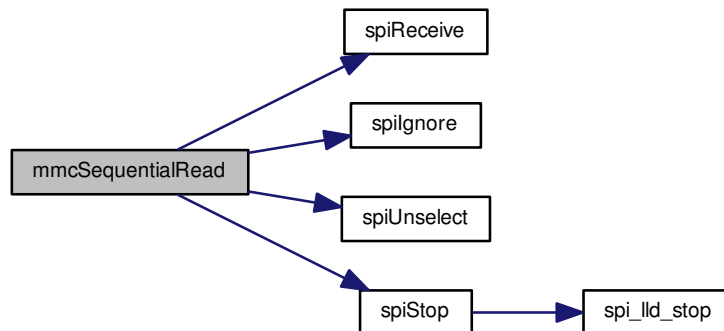
Return values

<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:



7.27.5.18 `bool mmcStopSequentialRead (MMCDriver * mmcp)`

Stops a sequential read gracefully.

Parameters

in	<code>mmcp</code>	pointer to the MMCDriver object
----	-------------------	-------------------------------------------------

Returns

The operation status.

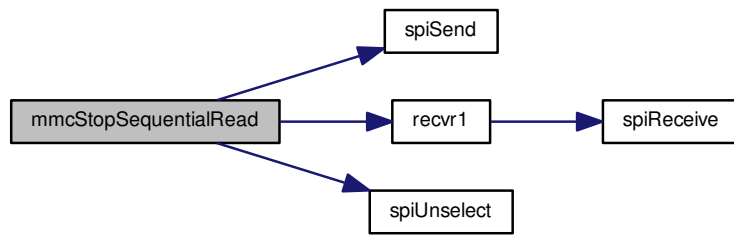
Return values

<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:



7.27.5.19 bool mmcStartSequentialWrite (MMCDriver * mmcp, uint32_t startblk)

Starts a sequential write.

Parameters

in	<i>mmcp</i>	pointer to the <code>MMCDriver</code> object
in	<i>startblk</i>	first block to write

Returns

The operation status.

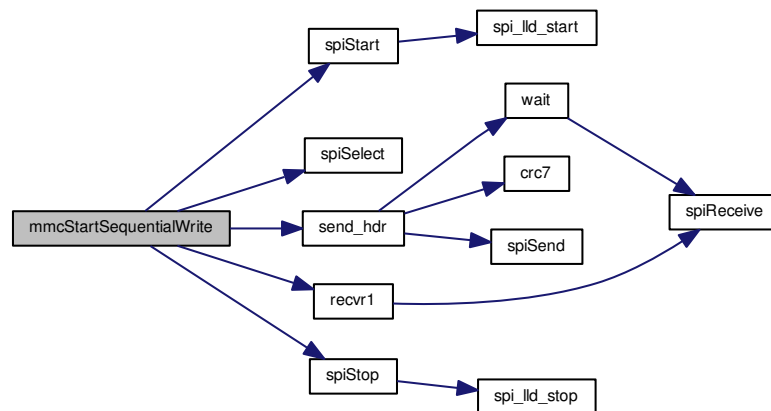
Return values

<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:



7.27.5.20 `bool mmcSequentialWrite (MMCDriver * mmcp, const uint8_t * buffer)`

Writes a block within a sequential write operation.

Parameters

in	<i>mmcp</i>	pointer to the MMCDriver object
out	<i>buffer</i>	pointer to the write buffer

Returns

The operation status.

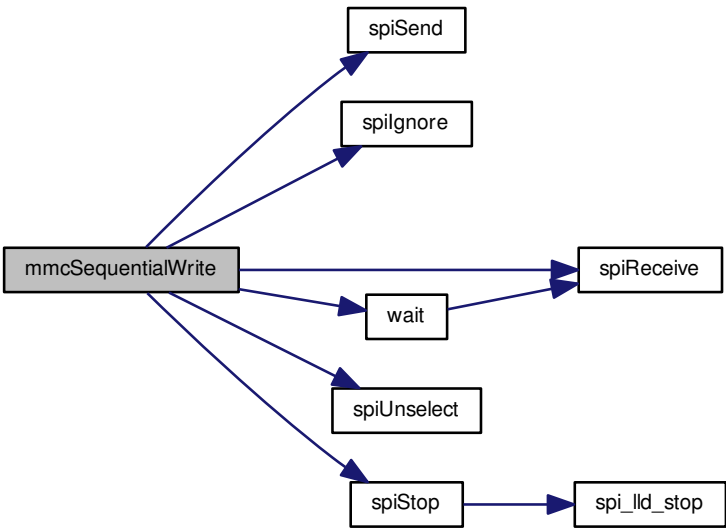
Return values

<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:



7.27.5.21 bool mmcStopSequentialWrite (MMCDriver * mmcp)

Stops a sequential write gracefully.

Parameters

in	<i>mmcp</i>	pointer to the <code>MMCDriver</code> object
----	-------------	----------------------------------------------

Returns

The operation status.

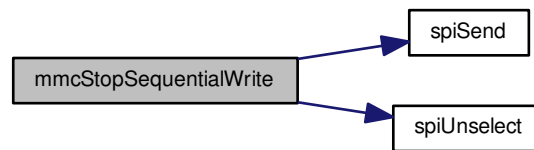
Return values

<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:



7.27.5.22 bool mmcSync (MMCDriver * mmcp)

Waits for card idle condition.

Parameters

in	<i>mmcp</i>	pointer to the MMCDriver object
----	-------------	-------------------------------------------------

Returns

The operation status.

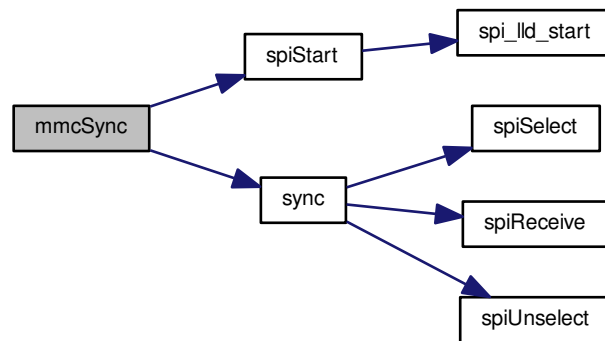
Return values

<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:



7.27.5.23 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

<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.

7.27.5.24 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.

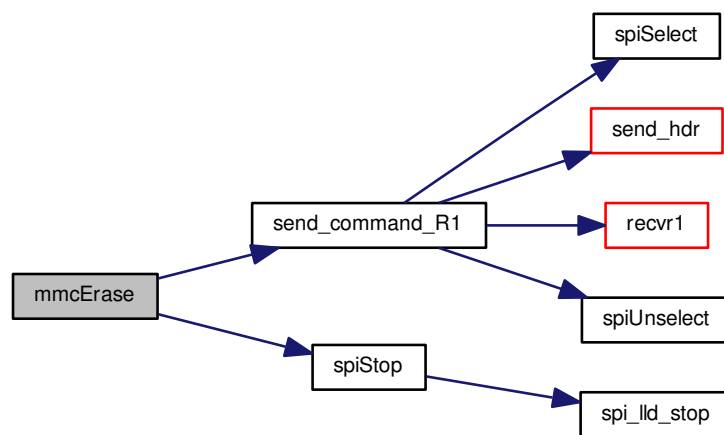
Return values

<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:

**7.27.6 Variable Documentation****7.27.6.1** `const struct MMCDriverVMT mmc_vmt` [static]**Initial value:**

```

= {
    (bool (*)(void *))mmc_llid_is_card_inserted,
    (bool (*)(void *))mmc_llid_is_write_protected,
    (bool (*)(void *))mmcConnect,
    (bool (*)(void *))mmcDisconnect,
    mmc_read,
    mmc_write,
    (bool (*)(void *))mmcSync,
    (bool (*)(void *, BlockDeviceInfo *))mmcGetInfo
}

```

Virtual methods table.

7.27.6.2 `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, 0x0b, 0x02, 0x3d, 0x34, 0x2f, 0x26,
    0x51, 0x58, 0x43, 0x4a, 0x75, 0x7c, 0x67, 0x6e, 0x32, 0x3b, 0x20, 0x29,
    0x16, 0x1f, 0x04, 0x0d, 0x7a, 0x73, 0x68, 0x61, 0x5e, 0x57, 0x4c, 0x45,
    0x2b, 0x22, 0x39, 0x30, 0x0f, 0x06, 0x1d, 0x14, 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, 0x0a,
    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, 0x0e, 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 $x^7 + x^3 + 1$).

7.28 MMC/SD Block Device

7.28.1 Detailed Description

This module implements a common ancestor for all device drivers accessing MMC or SD cards. This interface inherits the state machine and the interface from the [Abstract I/O Block Device](#) module.

Macros

- `#define MMCSD_BLOCK_SIZE 512U`
Fixed block size for MMC/SD block devices.
- `#define MMCSD_R1_ERROR_MASK 0xFDFFE008U`
Mask of error bits in R1 responses.
- `#define MMCSD_CMD8_PATTERN 0x000001AAU`
Fixed pattern for CMD8.
- `#define _mmcscd_block_device_methods _base_block_device_methods`
MMCSDBlockDevice specific methods.
- `#define _mmcscd_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_BLK_LEN_SLICE** 83U,80U
- #define **MMCSD_CSD_MMC_READ_BLK_PARTIAL_SLICE** 79U,79U
- #define **MMCSD_CSD_MMC_WRITE_BLK_MISALIGN_SLICE** 78U,78U
- #define **MMCSD_CSD_MMC_READ_BLK_MISALIGN_SLICE** 77U,77U
- #define **MMCSD_CSD_MMC_DSR_IMP_SLICE** 76U,76U
- #define **MMCSD_CSD_MMC_C_SIZE_SLICE** 73U,62U
- #define **MMCSD_CSD_MMC_VDD_R_CURR_MIN_SLICE** 61U,59U
- #define **MMCSD_CSD_MMC_VDD_R_CURR_MAX_SLICE** 58U,56U
- #define **MMCSD_CSD_MMC_VDD_W_CURR_MIN_SLICE** 55U,53U
- #define **MMCSD_CSD_MMC_VDD_W_CURR_MAX_SLICE** 52U,50U
- #define **MMCSD_CSD_MMC_C_SIZE_MULT_SLICE** 49U,47U
- #define **MMCSD_CSD_MMC_ERASE_GRP_SIZE_SLICE** 46U,42U
- #define **MMCSD_CSD_MMC_ERASE_GRP_MULT_SLICE** 41U,37U
- #define **MMCSD_CSD_MMC_WP_GRP_SIZE_SLICE** 36U,32U
- #define **MMCSD_CSD_MMC_WP_GRP_ENABLE_SLICE** 31U,31U
- #define **MMCSD_CSD_MMC_DEFAULT_ECC_SLICE** 30U,29U
- #define **MMCSD_CSD_MMC_R2W_FACTOR_SLICE** 28U,26U
- #define **MMCSD_CSD_MMC_WRITE_BLK_LEN_SLICE** 25U,22U
- #define **MMCSD_CSD_MMC_WRITE_BLK_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_BLK_PARTIAL_SLICE** 21U,21U
- #define **MMCSD_CSD_20_WRITE_BLK_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 `MMCSID_CID_SDC_CRC_SLICE` 7U,1U
Slice position of values in CID register.
- #define `MMCSID_CID_SDC_MDT_M_SLICE` 11U,8U
- #define `MMCSID_CID_SDC_MDT_Y_SLICE` 19U,12U
- #define `MMCSID_CID_SDC_PSN_SLICE` 55U,24U
- #define `MMCSID_CID_SDC_PRV_M_SLICE` 59U,56U
- #define `MMCSID_CID_SDC_PRV_N_SLICE` 63U,60U
- #define `MMCSID_CID_SDC_PNM0_SLICE` 71U,64U
- #define `MMCSID_CID_SDC_PNM1_SLICE` 79U,72U
- #define `MMCSID_CID_SDC_PNM2_SLICE` 87U,80U
- #define `MMCSID_CID_SDC_PNM3_SLICE` 95U,88U
- #define `MMCSID_CID_SDC_PNM4_SLICE` 103U,96U
- #define `MMCSID_CID_SDC_OID_SLICE` 119U,104U
- #define `MMCSID_CID_SDC_MID_SLICE` 127U,120U
- #define `MMCSID_CID_MMC_CRC_SLICE` 7U,1U
- #define `MMCSID_CID_MMC_MDT_Y_SLICE` 11U,8U
- #define `MMCSID_CID_MMC_MDT_M_SLICE` 15U,12U
- #define `MMCSID_CID_MMC_PSN_SLICE` 47U,16U
- #define `MMCSID_CID_MMC_PRV_M_SLICE` 51U,48U
- #define `MMCSID_CID_MMC_PRV_N_SLICE` 55U,52U
- #define `MMCSID_CID_MMC_PNM0_SLICE` 63U,56U
- #define `MMCSID_CID_MMC_PNM1_SLICE` 71U,64U
- #define `MMCSID_CID_MMC_PNM2_SLICE` 79U,72U
- #define `MMCSID_CID_MMC_PNM3_SLICE` 87U,80U
- #define `MMCSID_CID_MMC_PNM4_SLICE` 95U,88U
- #define `MMCSID_CID_MMC_PNM5_SLICE` 103U,96U
- #define `MMCSID_CID_MMC_OID_SLICE` 119U,104U
- #define `MMCSID_CID_MMC_MID_SLICE` 127U,120U

R1 response utilities

- #define `MMCSID_R1_ERROR(r1)` (((r1) & `MMCSID_R1_ERROR_MASK`) != 0U)
Evaluates to `TRUE` if the R1 response contains error flags.
- #define `MMCSID_R1_STS(r1)` (((r1) >> 9U) & 15U)
Returns the status field of an R1 response.
- #define `MMCSID_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 `MMCSIDBlockDeviceVMT`
MMCSIDBlockDevice virtual methods table.
- struct `MMCSIDBlockDevice`
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 CID register from MMC.
- struct `unpacked_sdc_csd_20_t`
Unpacked CSD v1.0 register from SDC.
- struct `unpacked_mmc_csd_t`
Unpacked CSD v2.0 register from SDC.
- struct `unpacked_mmc_csd_t`
Unpacked CSD register from MMC.

Functions

- `uint32_t _mmcscd_get_slice` (const `uint32_t` *data, `uint32_t` end, `uint32_t` start)
Gets a bit field from a words array.
- `uint32_t _mmcscd_get_capacity` (const `uint32_t` *csd)
Extract card capacity from a CSD.
- `uint32_t _mmcscd_get_capacity_ext` (const `uint8_t` *ext_csd)
Extract MMC card capacity from EXT_CSD.
- void `_mmcscd_unpack_sdc_cid` (const `MMCSDBlockDevice` *sdcp, `unpacked_sdc_cid_t` *cidsdc)
Unpacks SDC CID array in structure.
- void `_mmcscd_unpack_mmc_cid` (const `MMCSDBlockDevice` *sdcp, `unpacked_mmc_cid_t` *cidmmc)
Unpacks MMC CID array in structure.
- void `_mmcscd_unpack_csd_mmc` (const `MMCSDBlockDevice` *sdcp, `unpacked_mmc_csd_t` *csdmmc)
Unpacks MMC CSD array in structure.
- void `_mmcscd_unpack_csd_v10` (const `MMCSDBlockDevice` *sdcp, `unpacked_sdc_csd_10_t` *csd10)
Unpacks SDC CSD v1.0 array in structure.
- void `_mmcscd_unpack_csd_v20` (const `MMCSDBlockDevice` *sdcp, `unpacked_sdc_csd_20_t` *csd20)
Unpacks SDC CSD v2.0 array in structure.

7.28.2 Macro Definition Documentation

7.28.2.1 #define MMCSD_BLOCK_SIZE 512U

Fixed block size for MMC/SD block devices.

7.28.2.2 #define MMCSD_R1_ERROR_MASK 0xFDFFE008U

Mask of error bits in R1 responses.

7.28.2.3 #define MMCSD_CMD8_PATTERN 0x000001AAU

Fixed pattern for CMD8.

7.28.2.4 #define MMCSD_CSD_MMC_CSD_STRUCTURE_SLICE 127U,126U

Slice position of values in CSD register.

7.28.2.5 #define MMCSD_CID_SDC_CRC_SLICE 7U,1U

Slice position of values in CID register.

7.28.2.6 `#define _mmcscd_block_device_methods _base_block_device_methods`

`MMCSDBlockDevice` specific methods.

7.28.2.7 `#define _mmcscd_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.

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

Evaluates to `TRUE` if the R1 response contains error flags.

Parameters

in	<i>r1</i>	the r1 response
----	-----------	-----------------

7.28.2.9 `#define MMCSD_R1_STS(r1) (((r1) >> 9U) & 15U)`

Returns the status field of an R1 response.

Parameters

in	<i>r1</i>	the r1 response
----	-----------	-----------------

7.28.2.10 `#define MMCSD_R1_IS_CARD_LOCKED(r1) (((r1) >> 21U) & 1U) != 0U)`

Evaluates to `TRUE` if the R1 response indicates a locked card.

Parameters

in	<i>r1</i>	the r1 response
----	-----------	-----------------

7.28.2.11 `#define mmcscdGetCardCapacity(ip) ((ip)->capacity)`

Returns the card capacity in blocks.

Parameters

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

Returns

The card capacity.

Function Class:

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

7.28.3 Function Documentation**7.28.3.1 `uint32_t _mmc_sd_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.

7.28.3.2 `uint32_t _mmc_sd_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:



7.28.3.3 `uint32_t _mmcscd_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.

7.28.3.4 `void _mmcscd_unpack_sdc_cid (const MMCSDBlockDevice * sdcp, unpacked_sdc_cid_t * cidsc)`

Unpacks SDC CID array in structure.

Parameters

in	<i>sdcp</i>	pointer to the MMCSDBlockDevice object
out	<i>cidsc</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:



7.28.3.5 void _mmcscd_unpack_mmc_cid (const MMCSDBlockDevice * *sdc*p, unpacked_mmc_cid_t * *cidmmc*)

Unpacks MMC CID array in structure.

Parameters

in	<i>sdc</i> p	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:



7.28.3.6 void _mmcscd_unpack_csd_mmc (const MMCSDBlockDevice * *sdc*p, unpacked_mmc_csd_t * *csdmmc*)

Unpacks MMC CSD array in structure.

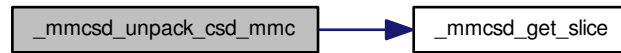
Parameters

in	<i>sdc</i> p	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:



7.28.3.7 void `_mmcscd_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 <code>MMCSDBlockDevice</code> object
out	<i>csd10</i>	pointer to the <code>unpacked_sdc_csd_10_t</code> object

Function Class:

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

Here is the call graph for this function:



7.28.3.8 void `_mmcscd_unpack_csd_v20` (const `MMCSDBlockDevice` * *sdcp*, `unpacked_sdc_csd_20_t` * *csd20*)

Unpacks SDC CSD v2.0 array in structure.

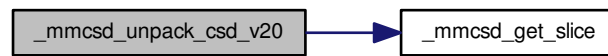
Parameters

in	<i>sdcp</i>	pointer to the <code>MMCSDBlockDevice</code> object
out	<i>csd20</i>	pointer to the <code>unpacked_sdc_csd_20_t</code> object

Function Class:

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

Here is the call graph for this function:



7.29 PAL Driver

I/O Ports Abstraction Layer.

7.29.1 Detailed Description

I/O Ports Abstraction Layer.

This module defines an abstract interface for digital I/O ports. Note that most I/O ports functions are just macros. The macros have default software implementations that can be redefined in a PAL Low Level Driver if the target hardware supports special features like, for example, atomic bit set/reset/masking. Please refer to the ports specific documentation for details.

The [PAL Driver](#) driver has the advantage to make the access to the I/O ports platform independent and still be optimized for the specific architectures.

Note that the PAL Low Level Driver may also offer non standard macro and functions in order to support specific features but, of course, the use of such interfaces would not be portable. Such interfaces shall be marked with the architecture name inside the function names.

Precondition

In order to use the PAL driver the `HAL_USE_PAL` option must be enabled in [halconf.h](#).

7.29.2 Implementation Rules

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

7.29.2.1 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.

7.29.2.2 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.

7.29.2.3 Writing unused or unimplemented port bits

The behavior is not specified.

7.29.2.4 Reading from unused or unimplemented port bits

The behavior is not specified.

7.29.2.5 Reading or writing on pins associated to other functionalities

The behavior is not specified.

Macros

- `#define PAL_PORT_BIT(n) ((iportmask_t)(1U << (n)))`
Port bit helper macro.
- `#define PAL_GROUP_MASK(width) ((iportmask_t)(1U << (width)) - 1U)`
Bits group mask helper.
- `#define _IOBUS_DATA(name, port, width, offset) {port, PAL_GROUP_MASK(width), offset}`
Data part of a static I/O bus initializer.
- `#define IOBUS_DECL(name, port, width, offset) IOBus name = _IOBUS_DATA(name, port, width, offset)`
Static I/O bus initializer.
- `#define IOPORT1 0`
First I/O port identifier.
- `#define pal_lld_init(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.

PAL event modes

- #define `PAL_EVENT_MODE_EDGES_MASK` 3U
Mask of edges field.
- #define `PAL_EVENT_MODE_DISABLED` 0U
Channel disabled.
- #define `PAL_EVENT_MODE_RISING_EDGE` 1U
Rising edge callback.
- #define `PAL_EVENT_MODE_FALLING_EDGE` 2U
Falling edge callback.
- #define `PAL_EVENT_MODE_BOTH_EDGES` 3U
Both edges callback.

Macro Functions

- #define `palInit(config) pal_Ild_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 logic state.
- #define `palWritePad(port, pad, bit)`

Writes a logic state on an output pad.
- #define `palSetPad(port, pad)` `palSetPort(port, PAL_PORT_BIT(pad))`

Sets a pad logic state to PAL_HIGH.
- #define `palClearPad(port, pad)` `palClearPort(port, PAL_PORT_BIT(pad))`

Clears a pad logic state to PAL_LOW.
- #define `palTogglePad(port, pad)` `palTogglePort(port, PAL_PORT_BIT(pad))`

Toggles a pad logic state.
- #define `palSetPadMode(port, pad, mode)` `palSetGroupMode(port, PAL_PORT_BIT(pad), 0U, mode)`

Pad mode setup.
- #define `palPadEnableEventl(port, pad, mode, callback)`

Pad event enable.
- #define `palPadDisableEventl(port, pad)`

Pad event disable.
- #define `palReadLine(line)` `palReadPad(PAL_PORT(line), PAL_PAD(line))`

Reads an input line logic state.
- #define `palWriteLine(line, bit)` `palWritePad(PAL_PORT(line), PAL_PAD(line), bit)`

Writes a logic state on an output line.
- #define `palSetLine(line)` `palSetPad(PAL_PORT(line), PAL_PAD(line))`

Sets a line logic state to PAL_HIGH.
- #define `palClearLine(line)` `palClearPad(PAL_PORT(line), PAL_PAD(line))`

Clears a line logic state to PAL_LOW.
- #define `palToggleLine(line)` `palTogglePad(PAL_PORT(line), PAL_PAD(line))`

Toggles a line logic state.
- #define `palSetLineMode(line, mode)` `palSetPadMode(PAL_PORT(line), PAL_PAD(line), mode)`

Line mode setup.
- #define `palLineEnableEventl(line, mode, callback)` `palPadEnableEventl(PAL_PORT(line), PAL_PAD(line), mode, callback)`

Line event enable.
- #define `palLineDisableEventl(line)` `palPadDisableEventl(PAL_PORT(line), PAL_PAD(line))`

Line event disable.

Port related definitions

- #define `PAL_IOPORTS_WIDTH` 16U

Width, in bits, of an I/O port.
- #define `PAL_WHOLE_PORT` `((ioportmask_t)0xFFFFU)`

Whole port mask.

Line handling macros

- `#define PAL_LINE(port, pad) (((ioline_t)((uint32_t)(port)) | ((uint32_t)(pad))))`
Forms a line identifier.
- `#define PAL_PORT(line) ((stm32_gpio_t *)(((uint32_t)(line)) & 0xFFFFFFFF0U))`
Decodes a port identifier from a line identifier.
- `#define PAL_PAD(line) ((uint32_t)((uint32_t)(line) & 0x0000000FU))`
Decodes a pad identifier from a line identifier.
- `#define PAL_NOLINE 0U`
Value identifying an invalid line.

Typedefs

- `typedef void(* palcallback_t) (void)`
Type of a PAL event callback.
- `typedef uint32_t ioportmask_t`
Digital I/O port sized unsigned type.
- `typedef uint32_t iomode_t`
Digital I/O modes.
- `typedef uint32_t ioline_t`
Type of an I/O line.
- `typedef uint32_t ioportid_t`
Port Identifier.

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.

7.29.3 Macro Definition Documentation

7.29.3.1 `#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.

7.29.3.2 `#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_PUSHPULL` for example.

7.29.3.3 `#define PAL_MODE_INPUT 2U`

Regular input high-Z pad.

7.29.3.4 `#define PAL_MODE_INPUT_PULLUP 3U`

Input pad with weak pull up resistor.

7.29.3.5 `#define PAL_MODE_INPUT_PULLDOWN 4U`

Input pad with weak pull down resistor.

7.29.3.6 `#define PAL_MODE_INPUT_ANALOG 5U`

Analog input mode.

7.29.3.7 `#define PAL_MODE_OUTPUT_PUSHPULL 6U`

Push-pull output pad.

7.29.3.8 `#define PAL_MODE_OUTPUT_OPENDRAIN 7U`

Open-drain output pad.

7.29.3.9 `#define PAL_LOW 0U`

Logical low state.

7.29.3.10 `#define PAL_HIGH 1U`

Logical high state.

7.29.3.11 `#define PAL_EVENT_MODE_EDGES_MASK 3U`

Mask of edges field.

7.29.3.12 `#define PAL_EVENT_MODE_DISABLED 0U`

Channel disabled.

7.29.3.13 `#define PAL_EVENT_MODE_RISING_EDGE 1U`

Rising edge callback.

7.29.3.14 `#define PAL_EVENT_MODE_FALLING_EDGE 2U`

Falling edge callback.

7.29.3.15 `#define PAL_EVENT_MODE_BOTH_EDGES 3U`

Both edges callback.

7.29.3.16 `#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.

7.29.3.17 `#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.

7.29.3.18 `#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

7.29.3.19 `#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

7.29.3.20 `#define palInit(config) pal_Ild_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.

7.29.3.21 `#define palReadPort(port) ((void)(port), 0U)`

Reads the physical I/O port states.

Note

The function can be called from any context.

Parameters

in	<i>port</i>	port identifier
----	-------------	-----------------

Returns

The port logic states.

Function Class:

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

7.29.3.22 `#define palReadLatch(port) ((void)(port), 0U)`

Reads the output latch.

The purpose of this function is to read back the latched output value.

Note

The function can be called from any context.

Parameters

in	<i>port</i>	port identifier
----	-------------	-----------------

Returns

The latched logic states.

Function Class:

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

7.29.3.23 `#define palWritePort(port, bits) ((void)(port), (void)(bits))`

Writes a bits mask on a I/O port.

Note

The function can be called from any context.

Parameters

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.

7.29.3.24 `#define palSetPort(port, bits) palWritePort(port, palReadLatch(port) | (bits))`

Sets a bits mask on a I/O port.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.
The function can be called from any context.

Parameters

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.

7.29.3.25 `#define palClearPort(port, bits) palWritePort(port, palReadLatch(port) & ~(bits))`

Clears a bits mask on a I/O port.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.
The function can be called from any context.

Parameters

in	<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.

7.29.3.26 `#define palTogglePort(port, bits) palWritePort(port, palReadLatch(port) ^ (bits))`

Toggles a bits mask on a I/O port.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.
The function can be called from any context.

Parameters

in	<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.

7.29.3.27 `#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 logic AND is performed on the input data
in	<i>offset</i>	group bit offset within the port

Returns

The group logic states.

Function Class:

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

7.29.3.28 #define palWriteGroup(port, mask, offset, bits)**Value:**

```
palWritePort(port, (palReadLatch(port) & ~((mask) << (offset))) | \
                (((bits) & (mask)) << (offset)))
```

Writes a group of bits.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`. The function can be called from any context.

Parameters

in	<i>port</i>	port identifier
in	<i>mask</i>	group mask, a logic 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.

7.29.3.29 #define palSetGroupMode(port, mask, offset, mode)

Pads group mode setup.

This function programs a pads group belonging to the same port with the specified mode.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`. Programming an unknown or unsupported mode is silently ignored. The function can be called from any context.

Parameters

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.

7.29.3.30 `#define palReadPad(port, pad) ((palReadPort(port) >> (pad)) & 1U)`

Reads an input pad logic state.

Note

The function can be called from any context.

Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port

Returns

The logic state.

Return values

<i>PAL_LOW</i>	low logic state.
<i>PAL_HIGH</i>	high logic state.

Function Class:

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

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

Value:

```
palWritePort(port, (palReadLatch(port) & ~PAL_PORT_BIT(pad)) |
    \
    (((bit) & 1U) << pad))
```

Writes a logic state on an output pad.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.
The function can be called from any context.

Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port
in	<i>bit</i>	logic value, the value must be <code>PAL_LOW</code> or <code>PAL_HIGH</code>

Function Class:

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

7.29.3.32 `#define palSetPad(port, pad) palSetPort(port, PAL_PORT_BIT(pad))`

Sets a pad logic state to `PAL_HIGH`.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.
The function can be called from any context.

Parameters

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.

7.29.3.33 `#define palClearPad(port, pad) palClearPort(port, PAL_PORT_BIT(pad))`

Clears a pad logic state to `PAL_LOW`.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.
The function can be called from any context.

Parameters

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.

7.29.3.34 `#define palTogglePad(port, pad) palTogglePort(port, PAL_PORT_BIT(pad))`

Toggles a pad logic state.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.
The function can be called from any context.

Parameters

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.

7.29.3.35 `#define palSetPadMode(port, pad, mode) palSetGroupMode(port, PAL_PORT_BIT(pad), 0U, mode)`

Pad mode setup.

This function programs a pad with the specified mode.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.
Programming an unknown or unsupported mode is silently ignored.
The function can be called from any context.

Parameters

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.

7.29.3.36 `#define palPadEnableEvent(port, pad, mode, callback)`

Pad event enable.

This function programs an event callback in the specified mode.

Note

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 event mode
in	<i>callback</i>	event callback function

Function Class:

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

handlers.

7.29.3.37 #define palPadDisableEventl(*port*, *pad*)

Pad event disable.

This function disables previously programmed event callbacks.

Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port

Function Class:

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

7.29.3.38 #define palReadLine(*line*) palReadPad(PAL_PORT(line), PAL_PAD(line))

Reads an input line logic state.

Note

The function can be called from any context.

Parameters

in	<i>line</i>	line identifier
----	-------------	-----------------

Returns

The logic state.

Return values

<i>PAL_LOW</i>	low logic state.
<i>PAL_HIGH</i>	high logic state.

Function Class:

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

7.29.3.39 #define palWriteLine(*line*, *bit*) palWritePad(PAL_PORT(line), PAL_PAD(line), bit)

Writes a logic state on an output line.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.
The function can be called from any context.

Parameters

in	<i>line</i>	line identifier
in	<i>bit</i>	logic value, the value must be <code>PAL_LOW</code> or <code>PAL_HIGH</code>

Function Class:

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

7.29.3.40 `#define palSetLine(line) palSetPad(PAL_PORT(line), PAL_PAD(line))`

Sets a line logic state to `PAL_HIGH`.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.
The function can be called from any context.

Parameters

in	<i>line</i>	line identifier
----	-------------	-----------------

Function Class:

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

7.29.3.41 `#define palClearLine(line) palClearPad(PAL_PORT(line), PAL_PAD(line))`

Clears a line logic state to `PAL_LOW`.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.
The function can be called from any context.

Parameters

in	<i>line</i>	line identifier
----	-------------	-----------------

Function Class:

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

7.29.3.42 `#define palToggleLine(line) palTogglePad(PAL_PORT(line), PAL_PAD(line))`

Toggles a line logic state.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.
The function can be called from any context.

Parameters

in	<i>line</i>	line identifier
----	-------------	-----------------

Function Class:

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

7.29.3.43 `#define palSetLineMode(line, mode) palSetPadMode(PAL_PORT(line), PAL_PAD(line), mode)`

Line mode setup.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.
The function can be called from any context.

Parameters

in	<i>line</i>	line identifier
in	<i>mode</i>	pad mode

Function Class:

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

7.29.3.44 `#define palLineEnableEventl(line, mode, callback) palPadEnableEventl(PAL_PORT(line), PAL_PAD(line), mode, callback)`

Line event enable.

Parameters

in	<i>line</i>	line identifier
in	<i>mode</i>	line event mode
in	<i>callback</i>	event callback function

Function Class:

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

7.29.3.45 `#define palLineDisableEventI(line) palPadDisableEventI(PAL_PORT(line), PAL_PAD(line))`

Line event disable.

Parameters

in	<i>line</i>	line identifier
----	-------------	-----------------

Function Class:

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

7.29.3.46 `#define PAL_IOPORTS_WIDTH 16U`

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

7.29.3.47 `#define PAL_WHOLE_PORT ((ioportmask_t)0xFFFFU)`

Whole port mask.

This macro specifies all the valid bits into a port.

7.29.3.48 `#define PAL_LINE(port, pad) ((ioline_t)((uint32_t)(port) | ((uint32_t)(pad)))`

Forms a line identifier.

A port/pad pair are encoded into an `ioline_t` type. The encoding of this type is platform-dependent.

7.29.3.49 `#define PAL_PORT(line) ((stm32_gpio_t *)(((uint32_t)(line)) & 0xFFFFFFF0U))`

Decodes a port identifier from a line identifier.

7.29.3.50 `#define PAL_PAD(line) ((uint32_t)((uint32_t)(line) & 0x000000FU))`

Decodes a pad identifier from a line identifier.

7.29.3.51 `#define PAL_NOLINE 0U`

Value identifying an invalid line.

7.29.3.52 `#define IOPORT1 0`

First I/O port identifier.

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

7.29.3.53 `#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.

7.29.3.54 #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.

7.29.3.55 #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.

7.29.3.56 #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.

7.29.3.57 #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.

7.29.3.58 #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.

7.29.3.59 #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.

7.29.3.60 #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.

7.29.3.61 #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.

7.29.3.62 #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.

7.29.3.63 #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.

7.29.3.64 #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 <i>PAL_LOW</i> or <i>PAL_HIGH</i>

Function Class:

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

7.29.3.65 #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.

7.29.3.66 #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.

7.29.3.67 #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.

7.29.3.68 `#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.

7.29.4 Typedef Documentation

7.29.4.1 `typedef void(* palcallback_t) (void)`

Type of a PAL event callback.

7.29.4.2 `typedef uint32_t ioportmask_t`

Digital I/O port sized unsigned type.

7.29.4.3 `typedef uint32_t iomode_t`

Digital I/O modes.

7.29.4.4 `typedef uint32_t ioline_t`

Type of an I/O line.

7.29.4.5 `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.

7.29.5 Function Documentation

7.29.5.1 `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.

7.29.5.2 `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

in	<i>bus</i>	the I/O bus, pointer to a IOBus structure
in	<i>bits</i>	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.

7.29.5.3 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.

7.29.5.4 void _pal_lld_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.

7.29.5.5 void _pal_lld_setgroupmode (ioportid_t *port*, ioportmask_t *mask*, iomode_t *mode*)

Pads mode setup.

This function programs a pads group belonging to the same port with the specified mode.

Parameters

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.

7.30 PWM Driver

Generic PWM Driver.

7.30.1 Detailed Description

Generic PWM Driver.

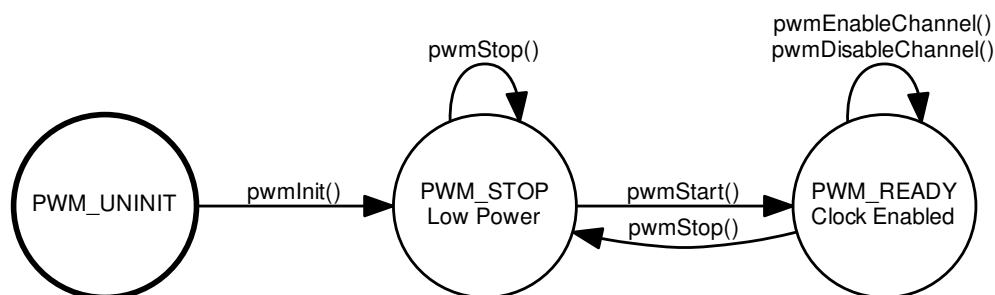
This module implements a generic PWM (Pulse Width Modulation) driver.

Precondition

In order to use the PWM driver the `HAL_USE_PWM` option must be enabled in `halconf.h`.

7.30.2 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).



7.30.3 PWM Operations.

This driver abstracts a generic PWM timer composed of:

- A clock prescaler.
- A main up counter.
- A comparator register that resets the main counter to zero when the limit is reached. An optional callback can be generated when this happens.
- An array of `PWM_CHANNELS` PWM channels, each channel has an output, a comparator and is able to invoke an optional callback when a comparator match with the main counter happens.

A PWM channel output can be in two different states:

- **IDLE**, when the channel is disabled or after a match occurred.
- **ACTIVE**, when the channel is enabled and a match didn't occur yet in the current PWM cycle.

Note that the two states can be associated to both logical zero or one in the `PWMChannelConfig` structure.

Macros

- #define `PWM_CHANNELS` 4
Number of PWM channels per PWM driver.
- #define `pwm_lld_change_period`(pwmp, period)
Changes the period the PWM peripheral.

PWM output mode macros

- #define `PWM_OUTPUT_MASK` 0x0FU
Standard output modes mask.
- #define `PWM_OUTPUT_DISABLED` 0x00U
Output not driven, callback only.
- #define `PWM_OUTPUT_ACTIVE_HIGH` 0x01U
Positive PWM logic, active is logic level one.
- #define `PWM_OUTPUT_ACTIVE_LOW` 0x02U
Inverse PWM logic, active is logic level zero.

PWM duty cycle conversion

- #define `PWM_FRACTION_TO_WIDTH`(pwmp, denominator, numerator)
Converts from fraction to pulse width.
- #define `PWM_DEGREES_TO_WIDTH`(pwmp, degrees) `PWM_FRACTION_TO_WIDTH`(pwmp, 36000, degrees)
Converts from degrees to pulse width.
- #define `PWM_PERCENTAGE_TO_WIDTH`(pwmp, percentage) `PWM_FRACTION_TO_WIDTH`(pwmp, 10000, percentage)
Converts from percentage to pulse width.

Macro Functions

- #define `pwmChangePeriodI`(pwmp, value)
Changes the period the PWM peripheral.
- #define `pwmEnableChannelI`(pwmp, channel, width)
Enables a PWM channel.
- #define `pwmDisableChannelI`(pwmp, channel)
Disables a PWM channel.
- #define `pwmIsChannelEnabledI`(pwmp, channel) (((pwmp)->enabled & ((`pwmchnmsk_t`)1U << (`pwmchnmsk_`
`_t`)(channel)))) != 0U
Returns a PWM channel status.
- #define `pwmEnablePeriodicNotificationI`(pwmp) `pwm_lld_enable_periodic_notification`(pwmp)
Enables the periodic activation edge notification.
- #define `pwmDisablePeriodicNotificationI`(pwmp) `pwm_lld_disable_periodic_notification`(pwmp)
Disables the periodic activation edge notification.
- #define `pwmEnableChannelNotificationI`(pwmp, channel) `pwm_lld_enable_channel_notification`(pwmp, chan-
nel)
Enables a channel de-activation edge notification.
- #define `pwmDisableChannelNotificationI`(pwmp, channel) `pwm_lld_disable_channel_notification`(pwmp,
channel)
Disables a channel de-activation edge notification.

PLATFORM configuration options

- #define `PLATFORM_PWM_USE_PWM1` FALSE
PWMD1 driver enable switch.

Typedefs

- typedef struct `PWMDriver` `PWMDriver`
Type of a structure representing a PWM driver.
- typedef void(* `pwmcallback_t`) (`PWMDriver` *pwmp)
Type of a PWM notification callback.
- typedef uint32_t `pwmmode_t`
Type of a PWM mode.
- typedef uint8_t `pwmchannel_t`
Type of a PWM channel.
- typedef uint32_t `pwmchnmsk_t`
Type of a channels mask.
- typedef uint32_t `pwmcnt_t`
Type of a PWM counter.

Data Structures

- struct `PWMChannelConfig`
Type of a PWM driver channel configuration structure.
- struct `PWMConfig`
Type of a PWM driver configuration structure.
- struct `PWMDriver`
Structure representing a PWM driver.

Functions

- void `pwmInit` (void)
PWM Driver initialization.
- void `pwmObjectInit` (`PWMDriver` *pwmp)
Initializes the standard part of a `PWMDriver` structure.
- void `pwmStart` (`PWMDriver` *pwmp, const `PWMConfig` *config)
Configures and activates the PWM peripheral.
- void `pwmStop` (`PWMDriver` *pwmp)
Deactivates the PWM peripheral.
- void `pwmChangePeriod` (`PWMDriver` *pwmp, `pwmcnt_t` period)
Changes the period the PWM peripheral.
- void `pwmEnableChannel` (`PWMDriver` *pwmp, `pwmchannel_t` channel, `pwmcnt_t` width)
Enables a PWM channel.
- void `pwmDisableChannel` (`PWMDriver` *pwmp, `pwmchannel_t` channel)
Disables a PWM channel and its notification.
- void `pwmEnablePeriodicNotification` (`PWMDriver` *pwmp)
Enables the periodic activation edge notification.
- void `pwmDisablePeriodicNotification` (`PWMDriver` *pwmp)
Disables the periodic activation edge notification.
- void `pwmEnableChannelNotification` (`PWMDriver` *pwmp, `pwmchannel_t` channel)

- Enables a channel de-activation edge notification.*

 - void `pwmDisableChannelNotification` (`PWMDriver *pwmp`, `pwmchannel_t channel`)

Disables a channel de-activation edge notification.
- void `pwm_lld_init` (void)

Low level PWM driver initialization.
- void `pwm_lld_start` (`PWMDriver *pwmp`)

Configures and activates the PWM peripheral.
- void `pwm_lld_stop` (`PWMDriver *pwmp`)

Deactivates the PWM peripheral.
- void `pwm_lld_enable_channel` (`PWMDriver *pwmp`, `pwmchannel_t channel`, `pwmcnt_t width`)

Enables a PWM channel.
- void `pwm_lld_disable_channel` (`PWMDriver *pwmp`, `pwmchannel_t channel`)

Disables a PWM channel and its notification.
- void `pwm_lld_enable_periodic_notification` (`PWMDriver *pwmp`)

Enables the periodic activation edge notification.
- void `pwm_lld_disable_periodic_notification` (`PWMDriver *pwmp`)

Disables the periodic activation edge notification.
- void `pwm_lld_enable_channel_notification` (`PWMDriver *pwmp`, `pwmchannel_t channel`)

Enables a channel de-activation edge notification.
- void `pwm_lld_disable_channel_notification` (`PWMDriver *pwmp`, `pwmchannel_t channel`)

Disables a channel de-activation edge notification.

Enumerations

Variables

- `PWMDriver PWMD1`
PWMD1 driver identifier.

7.30.4 Macro Definition Documentation

7.30.4.1 `#define PWM_OUTPUT_MASK 0x0FU`

Standard output modes mask.

7.30.4.2 `#define PWM_OUTPUT_DISABLED 0x00U`

Output not driven, callback only.

7.30.4.3 `#define PWM_OUTPUT_ACTIVE_HIGH 0x01U`

Positive PWM logic, active is logic level one.

7.30.4.4 `#define PWM_OUTPUT_ACTIVE_LOW 0x02U`

Inverse PWM logic, active is logic level zero.

7.30.4.5 `#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 PWMDriver 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.

7.30.4.6 `#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 PWMDriver 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.

7.30.4.7 `#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.

7.30.4.8 `#define pwmChangePeriod(pwmp, value)`

Value:

```
{
    (pwmp)->period = (value);
    pwm_llc_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.

7.30.4.9 #define pwmEnableChannel(pwmp, channel, width)**Value:**

```
do {
    (pwmp)->enabled |= ((pwmchnmsk_t)1U << (pwmchnmsk_t)(channel));
    pwm_llc_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.

7.30.4.10 #define pwmDisableChannel(pwmp, channel)**Value:**

```
do {
    (pwmp)->enabled &= ~((pwmchnmsk_t)1U << (pwmchnmsk_t)(channel));
    pwm_llc_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 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.

```
7.30.4.11 #define pwmlsChannelEnabled( 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.

```
7.30.4.12 #define pwmEnablePeriodicNotification( pwmp ) pwm_ild_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.

7.30.4.13 `#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.

7.30.4.14 `#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.

7.30.4.15 `#define pwmDisableChannelNotification(pwmp, channel) pwm_ild_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 <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.

7.30.4.16 `#define PWM_CHANNELS 4`

Number of PWM channels per PWM driver.

7.30.4.17 `#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`.

7.30.4.18 `#define pwm_ild_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.

7.30.5 Typedef Documentation**7.30.5.1 typedef struct PWMDriver PWMDriver**

Type of a structure representing a PWM driver.

7.30.5.2 typedef void(* pwmcallback_t) (PWMDriver *pwmp)

Type of a PWM notification callback.

Parameters

in	<i>pwmp</i>	pointer to a PWMDriver object
----	-------------	-----------------------------------------------

7.30.5.3 typedef uint32_t pwmmode_t

Type of a PWM mode.

7.30.5.4 typedef uint8_t pwmchannel_t

Type of a PWM channel.

7.30.5.5 typedef uint32_t pwmchnmsk_t

Type of a channels mask.

7.30.5.6 typedef uint32_t pwmcnt_t

Type of a PWM counter.

7.30.6 Enumeration Type Documentation

7.30.6.1 enum pwmstate_t

Driver state machine possible states.

Enumerator

PWM_UNINIT Not initialized.

PWM_STOP Stopped.

PWM_READY Ready.

7.30.7 Function Documentation

7.30.7.1 void pwmInit (void)

PWM Driver initialization.

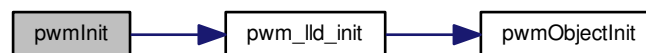
Note

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

Function Class:

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

Here is the call graph for this function:



7.30.7.2 void pwmObjectInit (PWMDriver * pwmp)

Initializes the standard part of a `PWMDriver` structure.

Parameters

out	<i>pwmp</i>	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.

7.30.7.3 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

in	<i>pwmp</i>	pointer to a PWMDriver object
in	<i>config</i>	pointer to a PWMConfig object

Function Class:

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

Here is the call graph for this function:

**7.30.7.4 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:

**7.30.7.5 void pwmChangePeriod (PWMDriver * pwmp, pwmcnt_t period)**

Changes the period the PWM peripheral.

This function changes the period of a PWM unit that has already been activated using [pwmStart\(\)](#).

Precondition

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

Postcondition

The PWM unit period is changed to the new value.

Note

If a period is specified that is shorter than the pulse width programmed in one of the channels then the behavior is not guaranteed.

Parameters

in	<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.

7.30.7.6 `void pwmEnableChannel (PWMDriver * pwmp, pwmchannel_t channel, pwmcnt_t width)`

Enables a PWM channel.

Precondition

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

Postcondition

The channel is active using the specified configuration.

Note

Depending on the hardware implementation this function has effect starting on the next cycle (recommended implementation) or immediately (fallback implementation).

Parameters

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:

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

7.30.7.7 `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 <code>PWMDriver</code> 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.

7.30.7.8 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 <code>PWMDriver</code> object
----	-------------	--------------------------------------------

Function Class:

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

7.30.7.9 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.

7.30.7.10 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.

7.30.7.11 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	<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.

7.30.7.12 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:



7.30.7.13 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	<i>pwmp</i>	pointer to a PWMDriver object
----	-------------	-----------------------------------------------

Function Class:

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

7.30.7.14 void pwm_lld_stop (PWMDriver * pwmp)

Deactivates the PWM peripheral.

Parameters

in	<i>pwmp</i>	pointer to a PWMDriver object
----	-------------	-----------------------------------------------

Function Class:

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

7.30.7.15 void pwm_lld_enable_channel (PWMDriver * pwmp, pwmchannel_t channel, pwmcnt_t width)

Enables a PWM channel.

Precondition

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

Postcondition

The channel is active using the specified configuration.

Note

The function has effect at the next cycle start.
Channel notification is not enabled.

Parameters

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:

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

7.30.7.16 `void pwm_lld_disable_channel (PWMDriver * pwmp, pwmchannel_t channel)`

Disables a PWM channel and its notification.

Precondition

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

Postcondition

The channel is disabled and its output line returned to the idle state.

Note

The function has effect at the next cycle start.

Parameters

in	<i>pwmp</i>	pointer to a <code>PWMDriver</code> object
in	<i>channel</i>	PWM channel identifier (0...channels-1)

Function Class:

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

7.30.7.17 `void pwm_lld_enable_periodic_notification (PWMDriver * pwmp)`

Enables the periodic activation edge notification.

Precondition

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

Note

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

Parameters

in	<i>pwmp</i>	pointer to a <code>PWMDriver</code> object
----	-------------	--------------------------------------------

Function Class:

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

7.30.7.18 void pwm_ild_disable_periodic_notification (PWMDriver * pwmp)

Disables the periodic activation edge notification.

Precondition

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

Note

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

Parameters

in	<i>pwmp</i>	pointer to a <code>PWMDriver</code> object
----	-------------	--------------------------------------------

Function Class:

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

7.30.7.19 void pwm_ild_enable_channel_notification (PWMDriver * pwmp, pwmchannel_t channel)

Enables a channel de-activation edge notification.

Precondition

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

The channel must have been activated using `pwmEnableChannel()`.

Note

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

Parameters

in	<i>pwmp</i>	pointer to a <code>PWMDriver</code> object
in	<i>channel</i>	PWM channel identifier (0...channels-1)

Function Class:

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

7.30.7.20 `void pwm_lld_disable_channel_notification (PWMDriver * pwp, 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>pwp</i>	pointer to a <code>PWMDriver</code> object
in	<i>channel</i>	PWM channel identifier (0...channels-1)

Function Class:

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

7.30.8 Variable Documentation**7.30.8.1 PWMDriver PWMD1**

PWMD1 driver identifier.

Note

The driver PWMD1 allocates the complex timer TIM1 when enabled.

7.31 QSPI Driver

Generic QSPI Driver.

7.31.1 Detailed Description

Generic QSPI Driver.

This module defines an abstract interface for a Quad SPI communication interface.

Precondition

In order to use the QSPI driver the `HAL_USE_QSPI` option must be enabled in `halconf.h`.

Transfer options

- `#define QSPI_CFG_CMD_MASK` (0xFFLU << 0LU)
- `#define QSPI_CFG_CMD(n)` ((n) << 0LU)
- `#define QSPI_CFG_CMD_MODE_MASK` (3LU << 8LU)
- `#define QSPI_CFG_CMD_MODE_NONE` (0LU << 8LU)
- `#define QSPI_CFG_CMD_MODE_ONE_LINE` (1LU << 8LU)
- `#define QSPI_CFG_CMD_MODE_TWO_LINES` (2LU << 8LU)
- `#define QSPI_CFG_CMD_MODE_FOUR_LINES` (3LU << 8LU)
- `#define QSPI_CFG_ADDR_MODE_MASK` (3LU << 10LU)
- `#define QSPI_CFG_ADDR_MODE_NONE` (0LU << 10LU)
- `#define QSPI_CFG_ADDR_MODE_ONE_LINE` (1LU << 10LU)
- `#define QSPI_CFG_ADDR_MODE_TWO_LINES` (2LU << 10LU)
- `#define QSPI_CFG_ADDR_MODE_FOUR_LINES` (3LU << 10LU)
- `#define QSPI_CFG_ADDR_SIZE_MASK` (3LU << 12LU)
- `#define QSPI_CFG_ADDR_SIZE_8` (0LU << 12LU)
- `#define QSPI_CFG_ADDR_SIZE_16` (1LU << 12LU)
- `#define QSPI_CFG_ADDR_SIZE_24` (2LU << 12LU)
- `#define QSPI_CFG_ADDR_SIZE_32` (3LU << 12LU)
- `#define QSPI_CFG_ALT_MODE_MASK` (3LU << 14LU)
- `#define QSPI_CFG_ALT_MODE_NONE` (0LU << 14LU)
- `#define QSPI_CFG_ALT_MODE_ONE_LINE` (1LU << 14LU)
- `#define QSPI_CFG_ALT_MODE_TWO_LINES` (2LU << 14LU)
- `#define QSPI_CFG_ALT_MODE_FOUR_LINES` (3LU << 14LU)
- `#define QSPI_CFG_ALT_SIZE_MASK` (3LU << 16LU)
- `#define QSPI_CFG_ALT_SIZE_8` (0LU << 16LU)
- `#define QSPI_CFG_ALT_SIZE_16` (1LU << 16LU)
- `#define QSPI_CFG_ALT_SIZE_24` (2LU << 16LU)
- `#define QSPI_CFG_ALT_SIZE_32` (3LU << 16LU)
- `#define QSPI_CFG_DUMMY_CYCLES_MASK` (0x1FLU << 18LU)
- `#define QSPI_CFG_DUMMY_CYCLES(n)` ((n) << 18LU)
- `#define QSPI_CFG_DATA_MODE_MASK` (3LU << 24LU)
- `#define QSPI_CFG_DATA_MODE_NONE` (0LU << 24LU)
- `#define QSPI_CFG_DATA_MODE_ONE_LINE` (1LU << 24LU)
- `#define QSPI_CFG_DATA_MODE_TWO_LINES` (2LU << 24LU)
- `#define QSPI_CFG_DATA_MODE_FOUR_LINES` (3LU << 24LU)
- `#define QSPI_CFG_SIOO` (1LU << 28LU)
- `#define QSPI_CFG_DDRM` (1LU << 31LU)

QSPI configuration options

- #define `QSPI_USE_WAIT` TRUE
Enables synchronous APIs.
- #define `QSPI_USE_MUTUAL_EXCLUSION` TRUE
Enables the `qspiAcquireBus()` and `qspiReleaseBus()` APIs.

Macro Functions

- #define `qspiStartCommandl`(qspi, cmdp)
Sends a command without data phase.
- #define `qspiStartSendl`(qspi, cmdp, n, txbuf)
Sends data over the QSPI bus.
- #define `qspiStartReceiveI`(qspi, cmdp, n, rxbuf)
Receives data from the QSPI bus.
- #define `qspiMapFlashl`(qspi, cmdp, addrp) `qspi_llc_map_flash`(qspi, cmdp, addrp)
Maps in memory space a QSPI flash device.
- #define `qspiUnmapFlashl`(qspi) `qspi_llc_unmap_flash`(qspi)
Maps in memory space a QSPI flash device.

Low level driver helper macros

- #define `_qspi_wakeup_isr`(qspi)
Wakes up the waiting thread.
- #define `_qspi_isr_code`(qspi)
Common ISR code.

QSPI capabilities

- #define `QSPI_SUPPORTS_MEMMAP` TRUE

Configuration options

- #define `PLATFORM_QSPI_USE_QSPI1` FALSE
QSPID1 driver enable switch.

Typedefs

- typedef struct `QSPIDriver` `QSPIDriver`
Type of a structure representing an QSPI driver.
- typedef void(* `qspicallback_t`) (`QSPIDriver` *qspi)
Type of a QSPI notification callback.

Data Structures

- struct `qspi_command_t`
Type of a QSPI command descriptor.
- struct `QSPIDriver`
Driver configuration structure.
- struct `QSPIDriver`
Structure representing an QSPI driver.

Functions

- void **qsplInit** (void)
QSPI Driver initialization.
- void **qsplObjectInit** (QSPIDriver *qspip)
*Initializes the standard part of a *QSPIDriver* structure.*
- void **qsplStart** (QSPIDriver *qspip, const QSPIConfig *config)
Configures and activates the QSPI peripheral.
- void **qsplStop** (QSPIDriver *qspip)
Deactivates the QSPI peripheral.
- void **qsplStartCommand** (QSPIDriver *qspip, const qspi_command_t *cmdp)
Sends a command without data phase.
- void **qsplStartSend** (QSPIDriver *qspip, const qspi_command_t *cmdp, size_t n, const uint8_t *txbuf)
Sends a command with data over the QSPI bus.
- void **qsplStartReceive** (QSPIDriver *qspip, const qspi_command_t *cmdp, size_t n, uint8_t *rxbuf)
Sends a command then receives data over the QSPI bus.
- void **qsplCommand** (QSPIDriver *qspip, const qspi_command_t *cmdp)
Sends a command without data phase.
- void **qsplSend** (QSPIDriver *qspip, const qspi_command_t *cmdp, size_t n, const uint8_t *txbuf)
Sends a command with data over the QSPI bus.
- void **qsplReceive** (QSPIDriver *qspip, const qspi_command_t *cmdp, size_t n, uint8_t *rxbuf)
Sends a command then receives data over the QSPI bus.
- void **qsplMapFlash** (QSPIDriver *qspip, const qspi_command_t *cmdp, uint8_t **addrp)
Maps in memory space a QSPI flash device.
- void **qsplUnmapFlash** (QSPIDriver *qspip)
Maps in memory space a QSPI flash device.
- void **qsplAcquireBus** (QSPIDriver *qspip)
Gains exclusive access to the QSPI bus.
- void **qsplReleaseBus** (QSPIDriver *qspip)
Releases exclusive access to the QSPI bus.
- void **qspl_lld_init** (void)
Low level QSPI driver initialization.
- void **qspl_lld_start** (QSPIDriver *qspip)
Configures and activates the QSPI peripheral.
- void **qspl_lld_stop** (QSPIDriver *qspip)
Deactivates the QSPI peripheral.
- void **qspl_lld_command** (QSPIDriver *qspip, const qspi_command_t *cmdp)
Sends a command without data phase.
- void **qspl_lld_send** (QSPIDriver *qspip, const qspi_command_t *cmdp, size_t n, const uint8_t *txbuf)
Sends a command with data over the QSPI bus.
- void **qspl_lld_receive** (QSPIDriver *qspip, const qspi_command_t *cmdp, size_t n, uint8_t *rxbuf)
Sends a command then receives data over the QSPI bus.
- void **qspl_lld_map_flash** (QSPIDriver *qspip, const qspi_command_t *cmdp, uint8_t **addrp)
Maps in memory space a QSPI flash device.
- void **qspl_lld_unmap_flash** (QSPIDriver *qspip)
Maps in memory space a QSPI flash device.

Enumerations

Variables

- **QSPIDriver QSPID1**
QSPID1 driver identifier.

7.31.2 Macro Definition Documentation

7.31.2.1 `#define QSPI_USE_WAIT TRUE`

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.31.2.2 `#define QSPI_USE_MUTUAL_EXCLUSION TRUE`

Enables the `qspiAcquireBus()` and `qspiReleaseBus()` APIs.

Note

Disabling this option saves both code and data space.

7.31.2.3 `#define qspiStartCommand(qspip, cmdp)`

Value:

```
{
    osalDbgAssert(((cmdp)->cfg & QSPI_CFG_DATA_MODE_MASK) ==
                  QSPI_CFG_DATA_MODE_NONE,
                  "data mode specified");
    (qspip)->state = QSPI_ACTIVE;
    qspi_llc_command(qspip, cmdp);
}
```

Sends a command without data phase.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

in	<i>qspip</i>	pointer to the <code>QSPIDriver</code> object
in	<i>cmdp</i>	pointer to the command descriptor

Function Class:

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

7.31.2.4 `#define qspiStartSend(qspip, cmdp, n, txbuf)`

Value:

```
{
    osalDbgAssert(((cmdp)->cfg & QSPI_CFG_DATA_MODE_MASK) !=
                  QSPI_CFG_DATA_MODE_NONE,
                  "data mode required");
    (qspip)->state = QSPI_ACTIVE;
}
```



```

    qspi_ll_d_send(qspip, cmdp, n, txbuf);
}

```

Sends data over the QSPI bus.

This asynchronous function starts a transmit operation.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object
in	<i>cmdp</i>	pointer to the command descriptor
in	<i>n</i>	number of bytes to send or zero if no data phase
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.

7.31.2.5 #define qspiStartReceive(*qspip, cmdp, n, rxbuf*)

Value:

```

{
    osalDbgAssert(((cmdp)->cfg & QSPI_CFG_DATA_MODE_MASK) !=
        QSPI_CFG_DATA_MODE_NONE,
        "data mode required");
    (qspip)->state = QSPI_ACTIVE;
    qspi_ll_d_receive(qspip, cmdp, n, rxbuf);
}

```

Receives data from the QSPI bus.

This asynchronous function starts a receive operation.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object
in	<i>cmdp</i>	pointer to the command descriptor
in	<i>n</i>	number of bytes to receive or zero if no data phase
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.

7.31.2.6 #define qspiMapFlash(*qspip*, *cmdp*, *addrp*) qspi_ild_map_flash(*qspip*, *cmdp*, *addrp*)

Maps in memory space a QSPI flash device.

Precondition

The memory flash device must be initialized appropriately before mapping it in memory space.

Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object
in	<i>cmdp</i>	pointer to the command descriptor
out	<i>addrp</i>	pointer to the memory start address of the mapped flash or <code>NULL</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.

7.31.2.7 #define qspiUnmapFlash(*qspip*) qspi_ild_unmap_flash(*qspip*)

Maps in memory space a QSPI flash device.

Postcondition

The memory flash device must be re-initialized for normal commands exchange.

Parameters

in	<i>qspip</i>	pointer to the QSPIDriver 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.

7.31.2.8 #define _qspi_wakeup_isr(*qspip*)

Value:

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

Wakes up the waiting thread.

Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object
----	--------------	--------------------------------------------------

Function Class:

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

7.31.2.9 #define _qspi_isr_code(qspip)**Value:**

```
{
    if ((qspip)->config->end_cb) {
        (qspip)->state = QSPI_COMPLETE;
        (qspip)->config->end_cb(qspip);
        if ((qspip)->state == QSPI_COMPLETE)
            (qspip)->state = QSPI_READY;
    }
    else
        (qspip)->state = QSPI_READY;
    _qspi_wakeup_isr(qspip);
}
```

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>qspip</i>	pointer to the <code>QSPIDriver</code> object
----	--------------	-----------------------------------------------

Function Class:

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

7.31.2.10 #define PLATFORM_QSPI_USE_QSPID1 FALSE

QSPID1 driver enable switch.

If set to `TRUE` the support for QSPID1 is included.

Note

The default is `FALSE`.

7.31.3 Typedef Documentation**7.31.3.1 typedef struct QSPIDriver QSPIDriver**

Type of a structure representing an QSPI driver.

7.31.3.2 typedef void(* qspicallback_t) (QSPIDriver *qspip)

Type of a QSPI notification callback.

Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object triggering the callback
----	--------------	--------------------------------------------------------------------------

7.31.4 Enumeration Type Documentation

7.31.4.1 enum qspistate_t

Driver state machine possible states.

Enumerator

- QSPI_UNINIT** Not initialized.
- QSPI_STOP** Stopped.
- QSPI_READY** Ready.
- QSPI_ACTIVE** Exchanging data.
- QSPI_COMPLETE** Asynchronous operation complete.
- QSPI_MEMMAP** In memory mapped mode.

7.31.5 Function Documentation

7.31.5.1 void qspilnit (void)

QSPI Driver initialization.

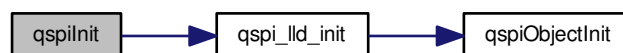
Note

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:



7.31.5.2 void qspiObjectInit (QSPIDriver * qspip)

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

Parameters

out	<i>qspip</i>	pointer to the QSPIDriver object
-----	--------------	--------------------------------------------------

Function Class:

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

7.31.5.3 void qspiStart (QSPIDriver * *qspip*, const QSPICConfig * *config*)

Configures and activates the QSPI peripheral.

Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object
in	<i>config</i>	pointer to the QSPICConfig 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:

**7.31.5.4 void qspiStop (QSPIDriver * *qspip*)**

Deactivates the QSPI peripheral.

Note

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

Parameters

in	<i>qspip</i>	pointer to the QSPIDriver 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:



7.31.5.5 void qspiStartCommand (QSPIDriver * *qspip*, const qspi_command_t * *cmdp*)

Sends a command without data phase.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object
in	<i>cmdp</i>	pointer to the command descriptor

Function Class:

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

7.31.5.6 void qspiStartSend (QSPIDriver * *qspip*, const qspi_command_t * *cmdp*, size_t *n*, const uint8_t * *txbuf*)

Sends a command with data over the QSPI bus.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object
in	<i>cmdp</i>	pointer to the command descriptor
in	<i>n</i>	number of bytes 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.

7.31.5.7 void qspiStartReceive (QSPIDriver * *qspip*, const qspi_command_t * *cmdp*, size_t *n*, uint8_t * *rxbuf*)

Sends a command then receives data over the QSPI bus.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object
in	<i>cmdp</i>	pointer to the command descriptor
in	<i>n</i>	number of bytes to send
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.

7.31.5.8 void `qspiCommand (QSPIDriver * qspip, const qspi_command_t * cmdp)`

Sends a command without data phase.

Precondition

In order to use this function the option `QSPI_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>qspip</i>	pointer to the QSPIDriver object
in	<i>cmdp</i>	pointer to the command descriptor

Function Class:

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

7.31.5.9 void `qspiSend (QSPIDriver * qspip, const qspi_command_t * cmdp, size_t n, const uint8_t * txbuf)`

Sends a command with data over the QSPI bus.

Precondition

In order to use this function the option `QSPI_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>qspip</i>	pointer to the QSPIDriver object
in	<i>cmdp</i>	pointer to the command descriptor
in	<i>n</i>	number of bytes 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.

7.31.5.10 void qspiReceive (QSPIDriver * *qspip*, const qspi_command_t * *cmdp*, size_t *n*, uint8_t * *rxbuf*)

Sends a command then receives data over the QSPI bus.

Precondition

In order to use this function the option QSPI_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>qspip</i>	pointer to the QSPIDriver object
in	<i>cmdp</i>	pointer to the command descriptor
in	<i>n</i>	number of bytes to send
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.

7.31.5.11 void qspiMapFlash (QSPIDriver * *qspip*, const qspi_command_t * *cmdp*, uint8_t ** *addrp*)

Maps in memory space a QSPI flash device.

Precondition

The memory flash device must be initialized appropriately before mapping it in memory space.

Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object
in	<i>cmdp</i>	pointer to the command descriptor
out	<i>addrp</i>	pointer to the memory start address of the mapped flash or NULL

Function Class:

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

7.31.5.12 void qspiUnmapFlash (QSPIDriver * *qspip*)

Maps in memory space a QSPI flash device.

Postcondition

The memory flash device must be re-initialized for normal commands exchange.

Parameters

in	<i>qspi</i>	pointer to the QSPIDriver object
----	-------------	--------------------------------------------------

Function Class:

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

7.31.5.13 void qspiAcquireBus (QSPIDriver * *qspi*)

Gains exclusive access to the QSPI bus.

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

Precondition

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

Parameters

in	<i>qspi</i>	pointer to the QSPIDriver object
----	-------------	--------------------------------------------------

Function Class:

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

7.31.5.14 void qspiReleaseBus (QSPIDriver * *qspi*)

Releases exclusive access to the QSPI bus.

Precondition

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

Parameters

in	<i>qspi</i>	pointer to the QSPIDriver object
----	-------------	--------------------------------------------------

Function Class:

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

7.31.5.15 void qspi_lld_init (void)

Low level QSPI driver initialization.

Function Class:

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

Here is the call graph for this function:



7.31.5.16 void qspi_llc_start (QSPIDriver * *qspi*)

Configures and activates the QSPI peripheral.

Parameters

in	<i>qspi</i>	pointer to the QSPIDriver object
----	-------------	--------------------------------------------------

Function Class:

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

7.31.5.17 void qspi_llc_stop (QSPIDriver * *qspi*)

Deactivates the QSPI peripheral.

Parameters

in	<i>qspi</i>	pointer to the QSPIDriver object
----	-------------	--------------------------------------------------

Function Class:

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

7.31.5.18 void qspi_llc_command (QSPIDriver * *qspi*, const qspi_command_t * *cmdp*)

Sends a command without data phase.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

in	<i>qspi</i>	pointer to the QSPIDriver object
in	<i>cmd</i>	pointer to the command descriptor

Function Class:

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

7.31.5.19 `void qspi_llc_send (QSPIDriver * qspip, const qspi_command_t * cmdp, size_t n, const uint8_t * txbuf)`

Sends a command with data over the QSPI bus.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object
in	<i>cmd</i>	pointer to the command descriptor
in	<i>n</i>	number of bytes to send
in	<i>txbuf</i>	the pointer to the transmit buffer

Function Class:

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

7.31.5.20 `void qspi_llc_receive (QSPIDriver * qspip, const qspi_command_t * cmdp, size_t n, uint8_t * rxbuf)`

Sends a command then receives data over the QSPI bus.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object
in	<i>cmd</i>	pointer to the command descriptor
in	<i>n</i>	number of bytes to send
out	<i>rxbuf</i>	the pointer to the receive buffer

Function Class:

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

7.31.5.21 `void qspi_llc_map_flash (QSPIDriver * qspip, const qspi_command_t * cmdp, uint8_t ** addrp)`

Maps in memory space a QSPI flash device.

Precondition

The memory flash device must be initialized appropriately before mapping it in memory space.

Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object
in	<i>cmdp</i>	pointer to the command descriptor
out	<i>addrp</i>	pointer to the memory start address of the mapped flash or NULL

Function Class:

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

7.31.5.22 void qspi_lld_unmap_flash (QSPIDriver * *qspip*)

Maps in memory space a QSPI flash device.

Postcondition

The memory flash device must be re-initialized for normal commands exchange.

Parameters

in	<i>qspip</i>	pointer to the QSPIDriver object
----	--------------	--------------------------------------------------

Function Class:

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

7.31.6 Variable Documentation**7.31.6.1 QSPIDriver QSPID1**

QSPID1 driver identifier.

7.32 RTC Driver

Generic RTC Driver.

7.32.1 Detailed Description

Generic RTC Driver.

This module defines an abstract interface for a Real Time Clock Peripheral.

Precondition

In order to use the RTC driver the `HAL_USE_RTC` option must be enabled in `halconf.h`.

Macros

- `#define RTC_BASE_YEAR 1980U`
Base year of the calendar.
- `#define _rtc_driver_methods _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 in the driver.
- `#define RTC_ALARMS 2`
Number of alarms available.
- `#define RTC_HAS_STORAGE FALSE`
Presence of a local persistent storage.

PLATFORM configuration options

- #define `PLATFORM_RTC_USE_RTC1` FALSE
RTCD1 driver enable switch.

Typedefs

- typedef struct `RTCDriver` `RTCDriver`
Type of a structure representing an RTC driver.
- typedef 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 `RTCAAlarm`
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 `RTCAAlarm` *alarmspec)
Set alarm time.
- void `rtcGetAlarm` (`RTCDriver` *rtcp, `rtcalarm_t` alarm, `RTCAAlarm` *alarmspec)
Get current alarm.
- void `rtcSetCallback` (`RTCDriver` *rtcp, `rtccb_t` callback)
Enables or disables RTC callbacks.
- void `rtcConvertDateTimeToStructTm` (const `RTCDateTime` *timespec, struct tm *timp, uint32_t *tv_msec)
Convert `RTCDateTime` to broken-down time structure.
- void `rtcConvertStructTmToDateTime` (const struct tm *timp, uint32_t tv_msec, `RTCDateTime` *timespec)
Convert broken-down time structure to `RTCDateTime`.
- uint32_t `rtcConvertDateTimeToFAT` (const `RTCDateTime` *timespec)
Get current time in format suitable for usage in FAT file system.
- void `rtc_lld_init` (void)
RTC driver identifier.
- void `rtc_lld_set_time` (`RTCDriver` *rtcp, const `RTCDateTime` *timespec)

Set current time.

- void `rtc_lld_get_time` (`RTCDriver` *rtcp, `RTCDateTime` *timespec)

Get current time.

- void `rtc_lld_set_alarm` (`RTCDriver` *rtcp, `rtcalarm_t` alarm, const `RTCAAlarm` *alarmspec)

Set alarm time.

- void `rtc_lld_get_alarm` (`RTCDriver` *rtcp, `rtcalarm_t` alarm, `RTCAAlarm` *alarmspec)

Get alarm time.

Enumerations

7.32.2 Macro Definition Documentation

7.32.2.1 #define RTC_BASE_YEAR 1980U

Base year of the calendar.

7.32.2.2 #define RTC_SUPPORTS_CALLBACKS TRUE

Callback support in the driver.

7.32.2.3 #define RTC_ALARMS 2

Number of alarms available.

7.32.2.4 #define RTC_HAS_STORAGE FALSE

Presence of a local persistent storage.

7.32.2.5 #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`.

7.32.2.6 #define _rtc_driver_methods _file_stream_methods

`FileStream` specific methods.

7.32.3 Typedef Documentation

7.32.3.1 typedef struct RTCDriver RTCDriver

Type of a structure representing an RTC driver.

7.32.3.2 typedef uint32_t rtcalarm_t

Type of an RTC alarm number.

7.32.3.3 typedef void(* rtccb_t) (RTCDriver *rtcp, rtcevent_t event)

Type of a generic RTC callback.

7.32.4 Enumeration Type Documentation

7.32.4.1 enum rtcevent_t

Type of an RTC event.

7.32.5 Function Documentation

7.32.5.1 void rtcInit (void)

RTC Driver initialization.

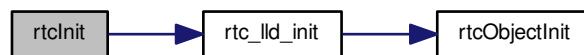
Note

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:



7.32.5.2 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	rtcp	pointer to RTC driver structure
-----	------	---------------------------------

Function Class:

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

7.32.5.3 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:

**7.32.5.4 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:



7.32.5.5 void rtcSetAlarm (RTCDriver * *rtcp*, rtcalarm_t *alarm*, const RTCAAlarm * *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 RTCAAlarm structure or NULL

Function Class:

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

Here is the call graph for this function:



7.32.5.6 void rtcGetAlarm (RTCDriver * *rtcp*, rtcalarm_t *alarm*, RTCAAlarm * *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:

7.32.5.7 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.

7.32.5.8 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 <code>NULL</code>

Function Class:

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

7.32.5.9 `void rtcConvertStructTmToDateTime (const struct tm * timep, uint32_t tv_msec, RTCDatetime * timespec)`

Convert broken-down time structure to [RTCDatetime](#).

Parameters

in	<i>timep</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.

7.32.5.10 `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.

7.32.5.11 `void rtc_lld_init (void)`

RTC driver identifier.

Enable access to registers.

Function Class:

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

Here is the call graph for this function:



7.32.5.12 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.

7.32.5.13 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.

7.32.5.14 void rtc_lld_set_alarm (RTCDriver * *rtcp*, rtcalarm_t *alarm*, const RTCAAlarm * *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 RTCArm structure.

Function Class:

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

7.32.5.15 void rtc_lld_get_alarm (RTCDriver * *rtcp*, rtcalarm_t *alarm*, RTCArm * *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 RTCArm structure

Function Class:

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

7.33 SDC Driver

Generic SD Card Driver.

7.33.1 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`.

7.33.2 Driver State Machine

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

7.33.3 Driver Operations

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

Macros

- `#define _sdc_driver_methods _mmc_sdc_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.
- #define `SDC_INIT_OCR_V20` 0x50FF8000U
OCR initialization constant for V20 cards.
- #define `SDC_INIT_OCR` 0x80100000U
OCR initialization constant for non-V20 cards.

Macro Functions

- #define `sdclsCardInserted(sdc)` (`sdc_llid_is_card_inserted(sdc)`)
Returns the card insertion status.
- #define `sdclsWriteProtected(sdc)` (`sdc_llid_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 `sdclflags_t`
SDC Driver condition flags type.
- typedef struct `SDCDriver` `SDCDriver`
Type of a structure representing an SDC driver.

Data Structures

- struct `SDCConfig`
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 `sd_cmd6_construct` (`sd_switch_t` mode, `sd_switch_function_t` function, uint32_t value)

Constructs CMD6 argument for SDC.
- static uint16_t `sd_cmd6_extract_info` (`sd_switch_function_t` function, const uint8_t *buf)

Extracts information from CMD6 answer.
- static bool `sd_cmd6_check_status` (`sd_switch_function_t` function, const uint8_t *buf)

Checks status after switching using CMD6.
- static bool `sd_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 `sd_set_bus_width` (`SDCDriver` *sdcp)

Sets bus width for SDC.
- static bool `mmc_set_bus_width` (`SDCDriver` *sdcp)

Sets bus width for MMC.
- bool `_sd_wait_for_transfer_state` (`SDCDriver` *sdcp)

Wait for the card to complete pending operations.
- void `sdclnit` (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 `sdRead` (`SDCDriver` *sdcp, uint32_t startblk, uint8_t *buf, uint32_t n)

Reads one or more blocks.
- bool `sdWrite` (`SDCDriver` *sdcp, uint32_t startblk, const uint8_t *buf, uint32_t n)

Writes one or more blocks.
- `sdclflags_t` `sdGetAndClearErrors` (`SDCDriver` *sdcp)

Returns the errors mask associated to the previous operation.
- bool `sdSync` (`SDCDriver` *sdcp)

Waits for card idle condition.
- bool `sdGetInfo` (`SDCDriver` *sdcp, `BlockDeviceInfo` *bdip)

Returns the media info.
- bool `sdErase` (`SDCDriver` *sdcp, uint32_t startblk, uint32_t endblk)

Erases the supplied blocks.
- void `sd_llc_init` (void)

Low level SDC driver initialization.
- void `sd_llc_start` (`SDCDriver` *sdcp)

Configures and activates the SDC peripheral.
- void `sd_llc_stop` (`SDCDriver` *sdcp)

Deactivates the SDC peripheral.

- void `sd_c_lld_start_clk` (`SDCDriver *sdcp`)
Starts the SDIO clock and sets it to init mode (400kHz or less).
- void `sd_c_lld_set_data_clk` (`SDCDriver *sdcp`, `sdcbusclk_t clk`)
Sets the SDIO clock to data mode (25MHz or less).
- void `sd_c_lld_stop_clk` (`SDCDriver *sdcp`)
Stops the SDIO clock.
- void `sd_c_lld_set_bus_mode` (`SDCDriver *sdcp`, `sdcbusmode_t mode`)
Switches the bus to 4 bits mode.
- void `sd_c_lld_send_cmd_none` (`SDCDriver *sdcp`, `uint8_t cmd`, `uint32_t arg`)
Sends an SDIO command with no response expected.
- bool `sd_c_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 `sd_c_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 `sd_c_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 `sd_c_lld_read` (`SDCDriver *sdcp`, `uint32_t startblk`, `uint8_t *buf`, `uint32_t n`)
Reads one or more blocks.
- bool `sd_c_lld_write` (`SDCDriver *sdcp`, `uint32_t startblk`, `const uint8_t *buf`, `uint32_t n`)
Writes one or more blocks.
- bool `sd_c_lld_sync` (`SDCDriver *sdcp`)
Waits for card idle condition.

Enumerations

Variables

- static const struct `SDCDriverVMT sd_c_vmt`
Virtual methods table.
- `SDCDriver SDCD1`
SDCD1 driver identifier.

7.33.4 Macro Definition Documentation

7.33.4.1 #define SDC_INIT_RETRY 100

Number of initialization attempts before rejecting the card.

Note

Attempts are performed at 10mS intervals.

7.33.4.2 #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.

7.33.4.3 #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.

7.33.4.4 #define SDC_INIT_OCR_V20 0x50FF8000U

OCR initialization constant for V20 cards.

7.33.4.5 #define SDC_INIT_OCR 0x80100000U

OCR initialization constant for non-V20 cards.

7.33.4.6 #define sdclsCardInserted(*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 SDCDriver 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.

7.33.4.7 #define sdclsWriteProtected(*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	<i>sdcp</i>	pointer to the SDCDriver 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.

7.33.4.8 #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`.

7.33.4.9 #define _sdcd_driver_methods _mmc_sdc_block_device_methods

[SDCDriver](#) specific methods.

7.33.5 Typedef Documentation**7.33.5.1 typedef uint32_t sdcmode_t**

Type of card flags.

7.33.5.2 typedef uint32_t sdcflags_t

SDC Driver condition flags type.

7.33.5.3 typedef struct SDCDriver SDCDriver

Type of a structure representing an SDC driver.

7.33.6 Enumeration Type Documentation**7.33.6.1 enum mmc_switch_t**

MMC switch mode.

7.33.6.2 enum `sd_switch_t`

SDC switch mode.

7.33.6.3 enum `sd_switch_function_t`

SDC switch function.

7.33.6.4 enum `sdcbusmode_t`

Type of SDIO bus mode.

7.33.6.5 enum `sdcbusclk_t`

Max supported clock.

7.33.7 Function Documentation

7.33.7.1 static bool `mode_detect (SDCDriver * sdc)` [static]

Detects card mode.

Parameters

in	<i>sdc</i>	pointer to the SDCDriver object
----	------------	-------------------------------------------------

Returns

The operation status.

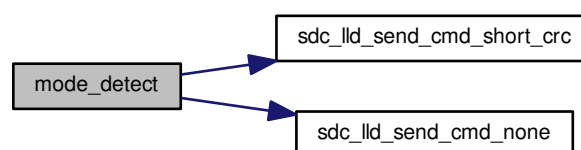
Return values

<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:



7.33.7.2 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:



7.33.7.3 static bool sdc_init (**SDCDriver** * *sdcp*) [static]

Init procedure for SDC.

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
----	-------------	-------------------------------------------------

Returns

The operation status.

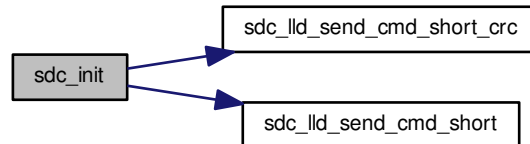
Return values

<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:



7.33.7.4 `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.

7.33.7.5 `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.

7.33.7.6 `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.

7.33.7.7 `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.

7.33.7.8 `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 SDCDriver object
out	<i>clk</i>	pointer to clock enum

Returns

The operation status.

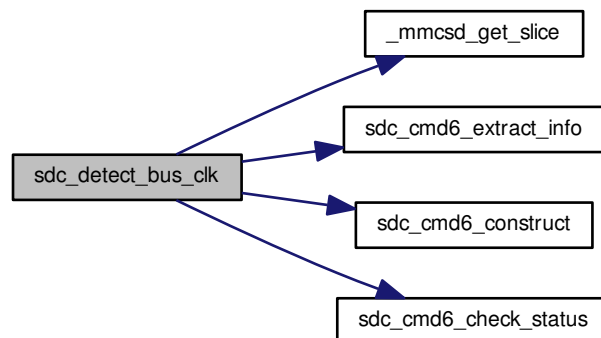
Return values

<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:



7.33.7.9 `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 SDCDriver object
out	<i>clk</i>	pointer to clock enum

Returns

The operation status.

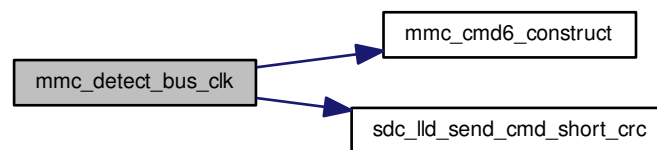
Return values

<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:



7.33.7.10 `static bool detect_bus_clk (SDCDriver * sdcp, sdcbusclk_t * clk)` [static]

Reads supported bus clock and switch card to appropriate mode.

Parameters

in	<i>sdc</i> p	pointer to the SDCDriver object
out	<i>clk</i>	pointer to clock enum

Returns

The operation status.

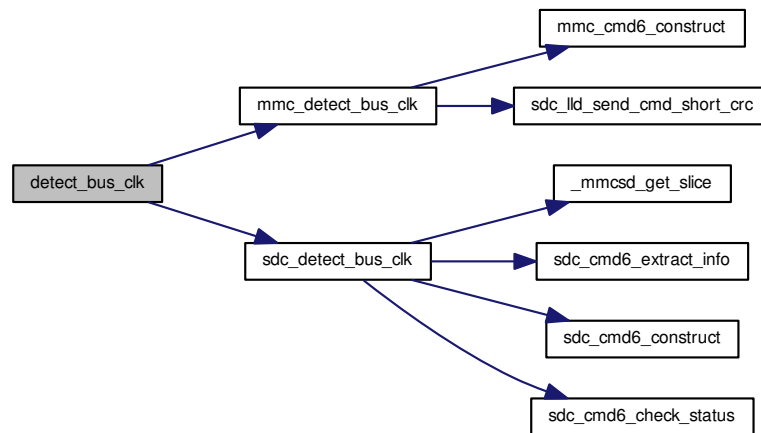
Return values

<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:



7.33.7.11 static bool sdc_set_bus_width (SDCDriver * *sdcp*) [static]

Sets bus width for SDC.

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
----	-------------	-------------------------------------------------

Returns

The operation status.

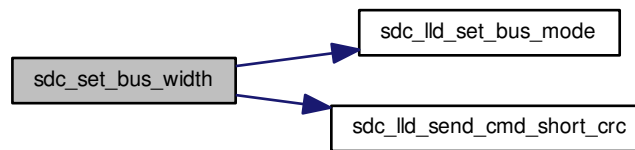
Return values

<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:



7.33.7.12 `static bool mmc_set_bus_width (SDCDriver * sdc) [static]`

Sets bus width for MMC.

Parameters

in	<i>sdc</i>	pointer to the SDCDriver object
----	------------	-------------------------------------------------

Returns

The operation status.

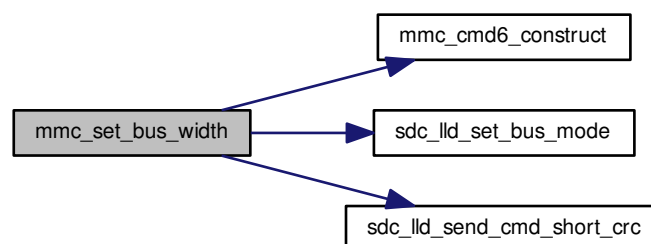
Return values

<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:



7.33.7.13 `bool _sdc_wait_for_transfer_state (SDCDriver * sdcp)`

Wait for the card to complete pending operations.

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
----	-------------	-------------------------------------------------

Returns

The operation status.

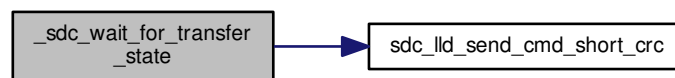
Return values

<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:



7.33.7.14 `void sdclnit (void)`

SDC Driver initialization.

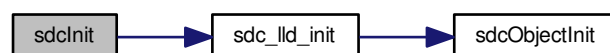
Note

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:



7.33.7.15 void sdcObjectInit (SDCDriver * *sdcp*)

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

Parameters

out	<i>sdcp</i>	pointer to the SDCDriver object
-----	-------------	-------------------------------------------------

Function Class:

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

7.33.7.16 void sdcStart (SDCDriver * *sdcp*, 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 SDCCConfig 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:

**7.33.7.17 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:



7.33.7.18 bool sdcConnect (SDCDriver * *sdc*)

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>sdc</i>	pointer to the SDCDriver object
----	------------	-------------------------------------------------

Returns

The operation status.

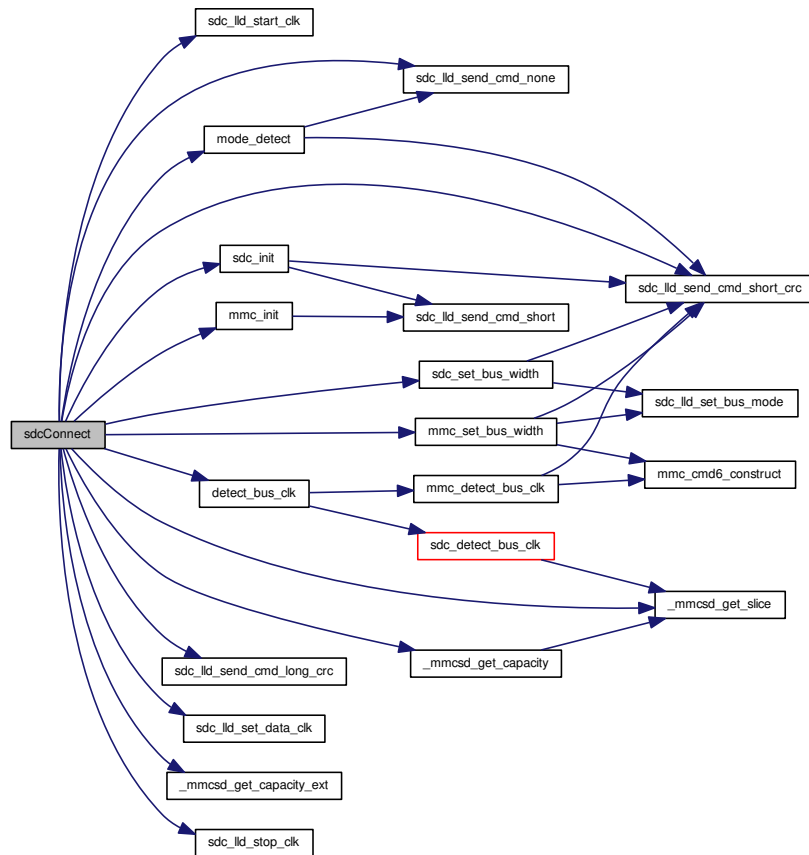
Return values

<code>HAL_SUCCESS</code>	operation succeeded.
<code>HAL_FAILED</code>	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:



7.33.7.19 bool sdcDisconnect (SDCDriver * sdc)

Brings the driver in a state safe for card removal.

Parameters

in	<i>sdc</i>	pointer to the SDCDriver object
----	------------	-------------------------------------------------

Returns

The operation status.

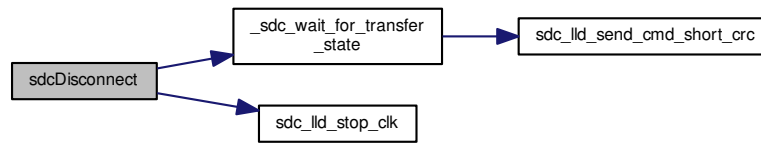
Return values

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

Function Class:

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

Here is the call graph for this function:



7.33.7.20 `bool sdcRead (SDCDriver * sdc, 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>sdc</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:

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

Here is the call graph for this function:



7.33.7.21 `bool sdcWrite (SDCDriver * sdcp, uint32_t startblk, const uint8_t * buf, uint32_t n)`

Writes one or more blocks.

Precondition

The driver must be in the `BLK_READY` state after a successful `sdcConnect()` invocation.

Parameters

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

<code>HAL_SUCCESS</code>	operation succeeded.
<code>HAL_FAILED</code>	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:



7.33.7.22 `sdcflags_t sdcGetAndClearErrors (SDCDriver * sdcp)`

Returns the errors mask associated to the previous operation.

Parameters

in	<i>sdcp</i>	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.

7.33.7.23 bool sdcSync (SDCDriver * *sdcp*)

Waits for card idle condition.

Parameters

in	<i>sdcp</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.

Here is the call graph for this function:

**7.33.7.24 bool sdcGetInfo (SDCDriver * *sdcp*, BlockDeviceInfo * *bdip*)**

Returns the media info.

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
out	<i>bdip</i>	pointer to a BlockDeviceInfo structure

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.

7.33.7.25 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.

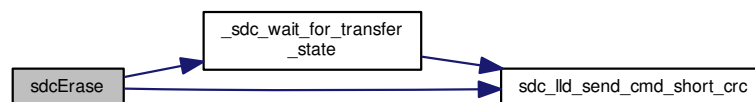
Return values

<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:

**7.33.7.26 void sdc_ll_init (void)**

Low level SDC driver initialization.

Function Class:

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

Here is the call graph for this function:

**7.33.7.27 void sdc_lld_start (SDCDriver * *sdcp*)**

Configures and activates the SDC peripheral.

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
----	-------------	-------------------------------------------------

Function Class:

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

7.33.7.28 void sdc_lld_stop (SDCDriver * *sdcp*)

Deactivates the SDC peripheral.

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
----	-------------	-------------------------------------------------

Function Class:

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

7.33.7.29 void sdc_lld_start_clk (SDCDriver * *sdcp*)

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.

7.33.7.30 void `sd_c_ild_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.

7.33.7.31 void `sd_c_ild_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.

7.33.7.32 void `sd_c_ild_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.

7.33.7.33 void `sd_c_ild_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.

7.33.7.34 `bool sdc_ild_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.

7.33.7.35 `bool sdc_ild_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.

7.33.7.36 `bool sdc_1ld_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 SDCDriver 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

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

Function Class:

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

7.33.7.37 `bool sdc_1ld_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 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:

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

7.33.7.38 `bool sdc_1ld_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 SDCDriver 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.

7.33.7.39 `bool sdc_1ld_sync (SDCDriver * sdcp)`

Waits for card idle condition.

Parameters

in	<i>sdcp</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.

7.33.8 Variable Documentation

7.33.8.1 const struct **SDCDriverVMT** **sdv_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.

7.33.8.2 **SDCDriver** **SDCD1**

SDCD1 driver identifier.

7.34 Serial Driver

Generic Serial Driver.

7.34.1 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.

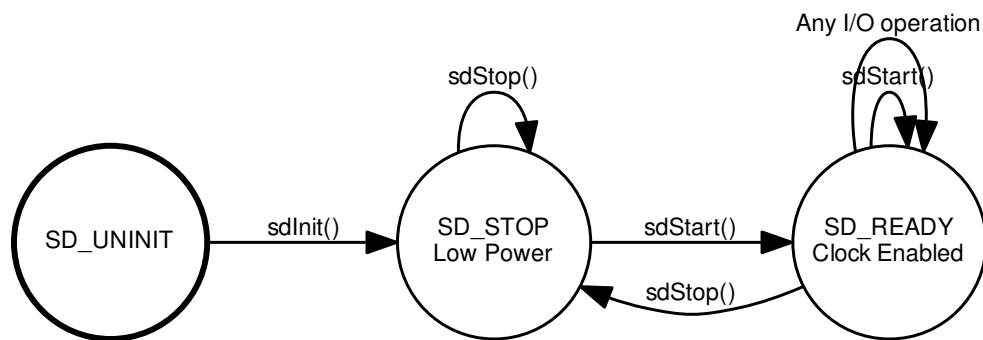
The module also contains functions that make the implementation of the interrupt service routines much easier.

Precondition

In order to use the SERIAL driver the `HAL_USE_SERIAL` option must be enabled in [halconf.h](#).

7.34.2 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).



Macros

- `#define _serial_driver_methods_base_asynchronous_channel_methods`
SerialDriver specific methods.
- `#define _serial_driver_data`
SerialDriver specific data.

Serial status flags

- `#define SD_PARITY_ERROR (eventflags_t)32`

- *Parity.*
- #define `SD_FRAMING_ERROR` (eventflags_t)64
- *Framing.*
- #define `SD_OVERRUN_ERROR` (eventflags_t)128
- *Overflow.*
- #define `SD_NOISE_ERROR` (eventflags_t)256
- *Line noise.*
- #define `SD_BREAK_DETECTED` (eventflags_t)512
- *LIN Break.*
- #define `SD_QUEUE_FULL_ERROR` (eventflags_t)1024
- *Queue full.*

Serial configuration options

- #define `SERIAL_DEFAULT_BITRATE` 38400
- *Default bit rate.*
- #define `SERIAL_BUFFERS_SIZE` 16
- *Serial buffers size.*

Macro Functions

- #define `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 [sdIncomingDataI](#) ([SerialDriver](#) *sdp, uint8_t b)
Handles incoming data.
- msg_t [sdRequestDataI](#) ([SerialDriver](#) *sdp)
Handles outgoing data.
- bool [sdPutWouldBlock](#) ([SerialDriver](#) *sdp)
Direct output check on a [SerialDriver](#).
- bool [sdGetWouldBlock](#) ([SerialDriver](#) *sdp)
Direct input check on a [SerialDriver](#).
- 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

Variables

- [SerialDriver](#) SD1
USART1 serial driver identifier.
- static const [SerialConfig](#) default_config
Driver default configuration.

7.34.3 Macro Definition Documentation

7.34.3.1 #define SD_PARITY_ERROR (eventflags_t)32

Parity.

7.34.3.2 `#define SD_FRAMING_ERROR (eventflags_t)64`

Framing.

7.34.3.3 `#define SD_OVERRUN_ERROR (eventflags_t)128`

Overflow.

7.34.3.4 `#define SD_NOISE_ERROR (eventflags_t)256`

Line noise.

7.34.3.5 `#define SD_BREAK_DETECTED (eventflags_t)512`

LIN Break.

7.34.3.6 `#define SD_QUEUE_FULL_ERROR (eventflags_t)1024`

Queue full.

7.34.3.7 `#define SERIAL_DEFAULT_BITRATE 38400`

Default bit rate.

Configuration parameter, this is the baud rate selected for the default configuration.

7.34.3.8 `#define SERIAL_BUFFERS_SIZE 16`

Serial buffers size.

Configuration parameter, you can change the depth of the queue buffers depending on the requirements of your application.

Note

The default is 16 bytes for both the transmission and receive buffers.
This is a global setting and it can be overridden by low level driver specific settings.

7.34.3.9 `#define _serial_driver_methods_base_asynchronous_channel_methods`

`SerialDriver` specific methods.

7.34.3.10 `#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.

7.34.3.11 `#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.

7.34.3.12 `#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.

7.34.3.13 `#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.

7.34.3.14 `#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.

7.34.3.15 `#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.

7.34.3.16 `#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.

7.34.3.17 `#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.

7.34.3.18 `#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.

7.34.3.19 `#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.

7.34.3.20 `#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`.

7.34.3.21 #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.

7.34.4 Typedef Documentation

7.34.4.1 typedef struct SerialDriver SerialDriver

Structure representing a serial driver.

7.34.5 Enumeration Type Documentation

7.34.5.1 enum sdstate_t

Driver state machine possible states.

Enumerator

SD_UNINIT Not initialized.

SD_STOP Stopped.

SD_READY Ready.

7.34.6 Function Documentation

7.34.6.1 void sdInit (void)

Serial Driver initialization.

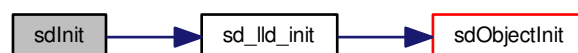
Note

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

Function Class:

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

Here is the call graph for this function:



7.34.6.2 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.

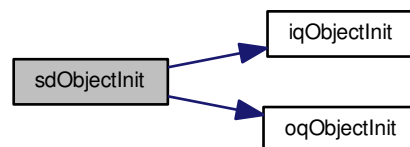
Parameters

out	<i>sdp</i>	pointer to a SerialDriver 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 <code>NULL</code> .
in	<i>onotify</i>	pointer to a callback function that is invoked when some data is written in the Queue. The value can be <code>NULL</code> .

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:



7.34.6.3 void sdStart (SerialDriver * sdp, const SerialConfig * config)

Configures and starts the driver.

Parameters

in	<i>sdp</i>	pointer to a SerialDriver 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:

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

Here is the call graph for this function:



7.34.6.4 void sdStop (**SerialDriver** * *sdp*)

Stops the driver.

Any thread waiting on the driver's queues will be awakened with the message `MSG_RESET`.

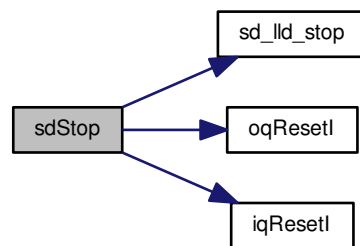
Parameters

in	<i>sdp</i>	pointer to a SerialDriver object
----	------------	--------------------------------------------------

Function Class:

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

Here is the call graph for this function:



7.34.6.5 void sdIncomingData (**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 SerialDriver 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:

**7.34.6.6 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 SerialDriver structure
----	------------	-----------------------------------------------------

Returns

The byte value read from the driver's output queue.

Return values

<i>MSG_TIMEOUT</i>	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:



7.34.6.7 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	sdp	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.

7.34.6.8 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.

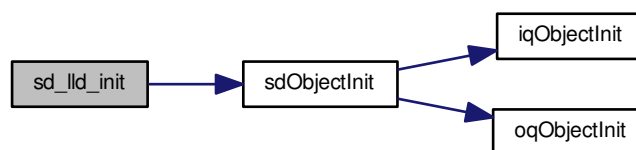
7.34.6.9 void sd_lld_init (void)

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:

**7.34.6.10 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 SerialDriver 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.

7.34.6.11 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 SerialDriver object
----	------------	--------------------------------------------------

Function Class:

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

7.34.7 Variable Documentation**7.34.7.1 SerialDriver SD1**

USART1 serial driver identifier.

7.34.7.2 const SerialConfig default_config [static]**Initial value:**

```
= {  
    38400  
}
```

Driver default configuration.

7.35 Serial over USB Driver

Serial over USB Driver.

7.35.1 Detailed Description

Serial over USB Driver.

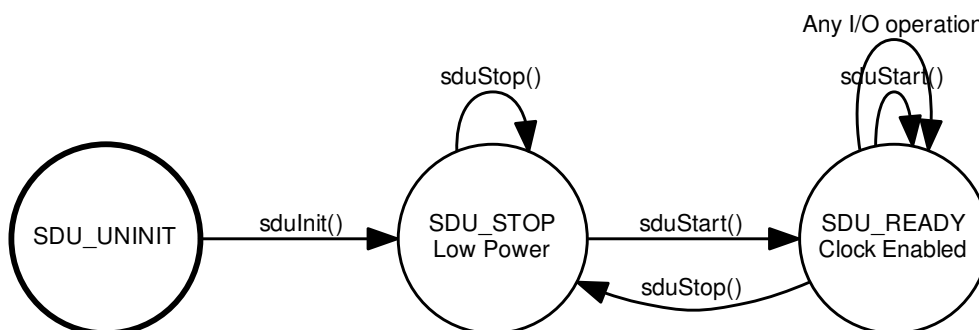
This module implements an USB Communication Device Class (CDC) as a normal serial communication port accessible from the device application.

Precondition

In order to use the USB over Serial driver the `HAL_USE_SERIAL_USB` option must be enabled in `halconf.h`.

7.35.2 Driver State Machine

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



Macros

- `#define _serial_usb_driver_data`
SerialDriver specific data.
- `#define _serial_usb_driver_methods_base_asynchronous_channel_methods`
SerialUSBDriver specific methods.

SERIAL_USB configuration options

- `#define SERIAL_USB_BUFFERS_SIZE 256`
Serial over USB buffers size.

- `#define SERIAL_USB_BUFFERS_NUMBER 2`
Serial over USB number of buffers.

Typedefs

- typedef struct `SerialUSBDriver` `SerialUSBDriver`
Structure representing a serial over USB driver.

Data Structures

- struct `SerialUSBConfig`
Serial over USB Driver configuration structure.
- struct `SerialUSBDriverVMT`
`SerialDriver` virtual methods table.
- struct `SerialUSBDriver`
Full duplex serial driver class.

Functions

- static void `ibnotify` (`io_buffers_queue_t` *bqp)
Notification of empty buffer released into the input buffers queue.
- static void `obnotify` (`io_buffers_queue_t` *bqp)
Notification of filled buffer inserted into the output buffers queue.
- void `sduInit` (void)
Serial Driver initialization.
- void `sduObjectInit` (`SerialUSBDriver` *sdup)
Initializes a generic full duplex driver object.
- void `sduStart` (`SerialUSBDriver` *sdup, const `SerialUSBConfig` *config)
Configures and starts the driver.
- void `sduStop` (`SerialUSBDriver` *sdup)
Stops the driver.
- void `sduSuspendHookI` (`SerialUSBDriver` *sdup)
USB device suspend handler.
- void `sduWakeupHookI` (`SerialUSBDriver` *sdup)
USB device wakeup handler.
- void `sduConfigureHookI` (`SerialUSBDriver` *sdup)
USB device configured handler.
- bool `sduRequestsHook` (`USBDriver` *usbp)
Default requests hook.
- void `sduSOFHookI` (`SerialUSBDriver` *sdup)
SOF handler.
- void `sduDataTransmitted` (`USBDriver` *usbp, `usbep_t` ep)
Default data transmitted callback.
- void `sduDataReceived` (`USBDriver` *usbp, `usbep_t` ep)
Default data received callback.
- void `sduInterruptTransmitted` (`USBDriver` *usbp, `usbep_t` ep)
Default data received callback.

Enumerations

7.35.3 Macro Definition Documentation

7.35.3.1 #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.

7.35.3.2 #define SERIAL_USB_BUFFERS_NUMBER 2

Serial over USB number of buffers.

Note

The default is 2 buffers.

7.35.3.3 #define _serial_usb_driver_data

Value:

```
_base_asynchronous_channel_data
/* Driver state.*/
sdustate_t          state;
/* Input buffers queue.*/
input_buffers_queue_t  ibqueue;
/* Output queue.*/
output_buffers_queue_t obqueue;
/* Input buffer.*/
uint8_t             ib[BQ_BUFFER_SIZE(
    SERIAL_USB_BUFFERS_NUMBER, \
                                SERIAL_USB_BUFFERS_SIZE)];
/* Output buffer.*/
uint8_t             ob[BQ_BUFFER_SIZE(
    SERIAL_USB_BUFFERS_NUMBER, \
                                SERIAL_USB_BUFFERS_SIZE)];
/* End of the mandatory fields.*/
/* Current configuration data.*/
const SerialUSBConfig *config;
```

`SerialDriver` specific data.

7.35.3.4 #define _serial_usb_driver_methods _base_asynchronous_channel_methods

`SerialUSBDriver` specific methods.

7.35.4 Typedef Documentation

7.35.4.1 typedef struct SerialUSBDriver SerialUSBDriver

Structure representing a serial over USB driver.

7.35.5 Enumeration Type Documentation

7.35.5.1 enum sdustate_t

Driver state machine possible states.

Enumerator

SDU_UNINIT Not initialized.

SDU_STOP Stopped.

SDU_READY Ready.

7.35.6 Function Documentation

7.35.6.1 static void ibnotify (io_buffers_queue_t * bqp) [static]

Notification of empty buffer released into the input buffers queue.

Parameters

in	<i>bqp</i>	the buffers queue pointer.
----	------------	----------------------------

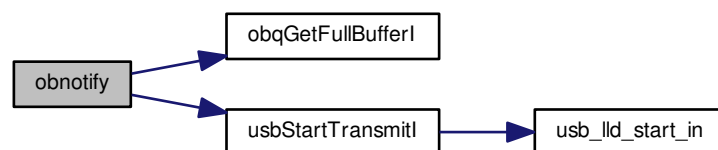
7.35.6.2 static void obnotify (io_buffers_queue_t * bqp) [static]

Notification of filled buffer inserted into the output buffers queue.

Parameters

in	<i>bqp</i>	the buffers queue pointer.
----	------------	----------------------------

Here is the call graph for this function:



7.35.6.3 void sdulnit (void)

Serial Driver initialization.

Note

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

Function Class:

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

7.35.6.4 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.

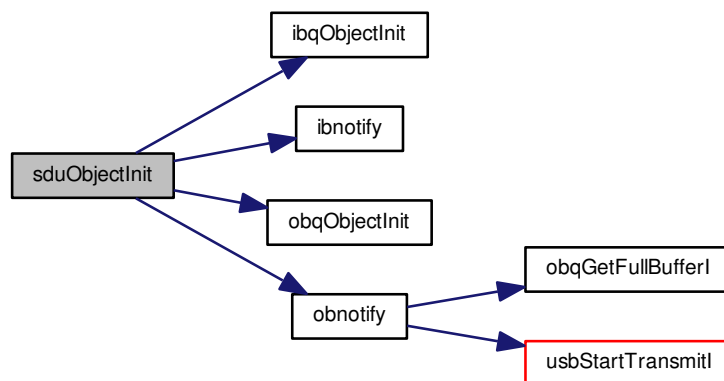
Parameters

out	<i>sdup</i>	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:

**7.35.6.5 void sduStart (SerialUSBDriver * sdup, const SerialUSBConfig * config)**

Configures and starts the driver.

Parameters

in	<i>sdup</i>	pointer to a <code>SerialUSBDriver</code> 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.

7.35.6.6 void sduStop (SerialUSBDriver * sdup)

Stops the driver.

Any thread waiting on the driver's queues will be awakened with the message MSG_RESET.

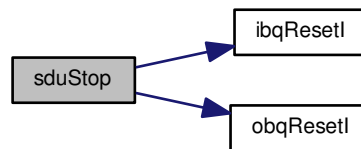
Parameters

in	<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:

**7.35.6.7 void sduSuspendHookI (SerialUSBDriver * sdup)**

USB device suspend handler.

Generates a CHN_DISCONNECT event and puts queues in non-blocking mode, this way the application cannot get stuck in the middle of an I/O operations.

Note

If this function is not called from an ISR then an explicit call to `osalOsRescheduleS()` is necessary afterward.

Parameters

in	<i>sdup</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.

7.35.6.8 void sduWakeupHookl (SerialUSBDriver * sdup)

USB device wakeup handler.

Generates a CHN_CONNECT event and resumes normal queues operations.

Note

If this function is not called from an ISR then an explicit call to `osalOsRescheduleS()` is necessary afterward.

Parameters

in	<i>sdup</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.

7.35.6.9 void sduConfigureHookl (SerialUSBDriver * sdup)

USB device configured handler.

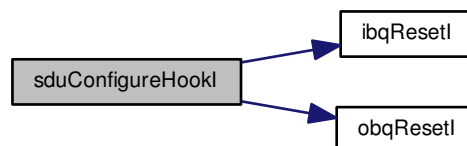
Parameters

in	<i>sdup</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:



7.35.6.10 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.

7.35.6.11 void sduSOFHookI (SerialUSBDriver * *sdu*)

SOF handler.

The SOF interrupt is used for automatic flushing of incomplete buffers pending in the output queue.

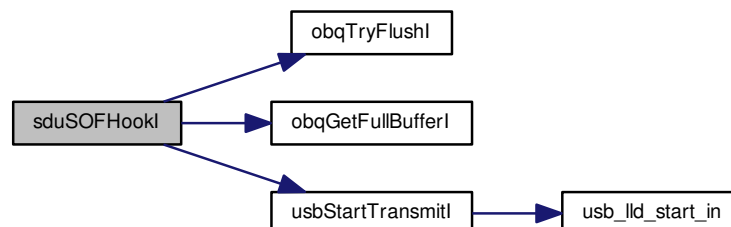
Parameters

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:



7.35.6.12 void sduDataTransmitted (USBDriver * *usbp*, *usbep_t ep*)

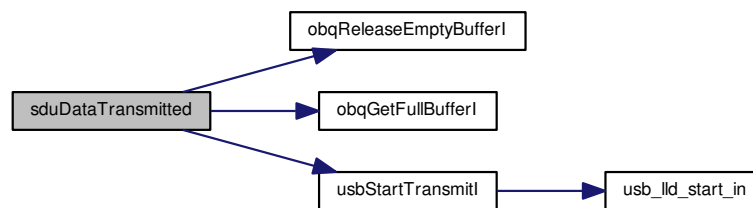
Default data transmitted callback.

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>	IN endpoint number

Here is the call graph for this function:



7.35.6.13 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>	OUT endpoint number

Here is the call graph for this function:



7.35.6.14 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

7.36 SPI Driver

Generic SPI Driver.

7.36.1 Detailed Description

Generic SPI Driver.

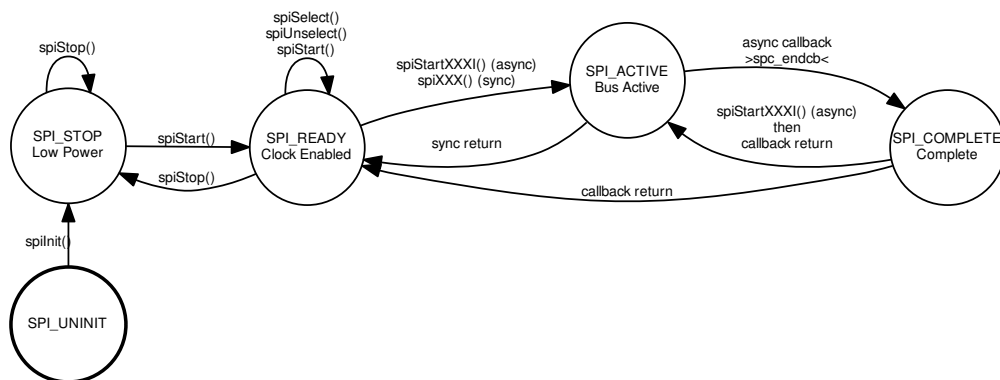
This module implements a generic SPI (Serial Peripheral Interface) driver allowing bidirectional and monodirectional transfers, complex atomic transactions are supported as well.

Precondition

In order to use the SPI driver the `HAL_USE_SPI` option must be enabled in `halconf.h`.

7.36.2 Driver State Machine

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



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(spi)`

- 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 spiStartReceiveI(spip, n, rxbuf)`
Receives data from the SPI bus.
- `#define spiPolledExchange(spip, frame) spi_llc_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.

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 spilnit (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 `spiIgnore (SPIDriver *spip, size_t n)`
Ignores data on the SPI bus.
- void `spiExchange (SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)`
Exchanges data on the SPI bus.
- void `spiSend (SPIDriver *spip, size_t n, const void *txbuf)`
Sends data over the SPI bus.
- void `spiReceive (SPIDriver *spip, size_t n, void *rxbuf)`
Receives data from the SPI bus.
- void `spiAcquireBus (SPIDriver *spip)`
Gains exclusive access to the SPI bus.
- void `spiReleaseBus (SPIDriver *spip)`
Releases exclusive access to the SPI bus.
- void `spi_llc_init (void)`
Low level SPI driver initialization.
- void `spi_llc_start (SPIDriver *spip)`
Configures and activates the SPI peripheral.
- void `spi_llc_stop (SPIDriver *spip)`
Deactivates the SPI peripheral.
- void `spi_llc_select (SPIDriver *spip)`
Asserts the slave select signal and prepares for transfers.
- void `spi_llc_unselect (SPIDriver *spip)`
Deasserts the slave select signal.
- void `spi_llc_ignore (SPIDriver *spip, size_t n)`
Ignores data on the SPI bus.
- void `spi_llc_exchange (SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)`
Exchanges data on the SPI bus.
- void `spi_llc_send (SPIDriver *spip, size_t n, const void *txbuf)`
Sends data over the SPI bus.
- void `spi_llc_receive (SPIDriver *spip, size_t n, void *rxbuf)`
Receives data from the SPI bus.
- uint16_t `spi_llc_polled_exchange (SPIDriver *spip, uint16_t frame)`
Exchanges one frame using a polled wait.

Enumerations

Variables

- `SPIDriver SPID1`
SPI1 driver identifier.

7.36.3 Macro Definition Documentation

7.36.3.1 #define SPI_USE_WAIT TRUE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.36.3.2 #define SPI_USE_MUTUAL_EXCLUSION TRUE

Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.

Note

Disabling this option saves both code and data space.

7.36.3.3 #define spiSelectl(*spip*)

Value:

```
{
    spi_llc_select(spip);
}
```

Asserts the slave select signal and prepares for transfers.

Parameters

in	<i>spip</i>	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.

7.36.3.4 #define spiUnselectl(*spip*)

Value:

```
{
    spi_llc_unselect(spip);
}
```

Deasserts the slave select signal.

The previously selected peripheral is unselected.

Parameters

in	<i>spip</i>	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.

7.36.3.5 #define spiStartIgnore(*spip*, *n*)**Value:**

```
{
    (spip)->state = SPI_ACTIVE;
    spi_llc_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 <code>SPIDriver</code> 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.

7.36.3.6 #define spiStartExchange(*spip*, *n*, *txbuf*, *rxbuf*)**Value:**

```
{
    (spip)->state = SPI_ACTIVE;
    spi_llc_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.

7.36.3.7 #define spiStartSendI(spip, n, txbuf)**Value:**

```
{
    (spip)->state = SPI_ACTIVE;
    spi_llc_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.

7.36.3.8 #define spiStartReceive(*spip*, *n*, *rxbuf*)**Value:**

```
{
    (spip)->state = SPI_ACTIVE;
    spi_llc_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.

7.36.3.9 #define spiPolledExchange(*spip*, *frame*) spi_llc_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 SPIDriver object
in	<i>frame</i>	the data frame to send over the SPI bus

Returns

The received data frame from the SPI bus.

7.36.3.10 #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.

7.36.3.11 #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.

7.36.3.12 `#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`.

7.36.4 Typedef Documentation

7.36.4.1 `typedef struct SPIDriver SPIDriver`

Type of a structure representing an SPI driver.

7.36.4.2 `typedef void(* spicallback_t)(SPIDriver *spip)`

SPI notification callback type.

Parameters

in	<i>spip</i>	pointer to the SPIDriver object triggering the callback
----	-------------	-------------------------------------------------------------------------

7.36.5 Enumeration Type Documentation

7.36.5.1 `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.

7.36.6 Function Documentation

7.36.6.1 `void spilnit (void)`

SPI Driver initialization.

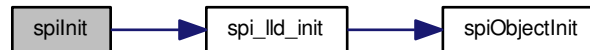
Note

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:

**7.36.6.2 void spiObjectInit (SPIDriver * *spip*)**

Initializes the standard part of a `SPIDriver` structure.

Parameters

out	<i>spip</i>	pointer to the <code>SPIDriver</code> object
-----	-------------	----------------------------------------------

Function Class:

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

7.36.6.3 void spiStart (SPIDriver * *spip*, const SPICConfig * *config*)

Configures and activates the SPI peripheral.

Parameters

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>config</i>	pointer to the <code>SPICConfig</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:



7.36.6.4 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 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:



7.36.6.5 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.

7.36.6.6 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.

7.36.6.7 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 SPIDriver 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.

7.36.6.8 void spiStartExchange (SPIDriver * *spip*, size_t *n*, const void * *txbuf*, void * *rxbuf*)

Exchanges data on the SPI bus.

This asynchronous function starts a simultaneous transmit/receive operation.

Precondition

A slave must have been selected using [spiSelect\(\)](#) or [spiSelectI\(\)](#).

Postcondition

At the end of the operation the configured callback is invoked.

Note

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

Parameters

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.

7.36.6.9 `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.

7.36.6.10 `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.

7.36.6.11 void spIgnore (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 <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.

7.36.6.12 void spiExchange (SPIDriver * spip, size_t n, const void * txbuf, void * rxbuf)

Exchanges data on the SPI bus.

This synchronous function performs a simultaneous transmit/receive operation.

Precondition

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

In order to use this function the driver must have been configured without callbacks (`end_cb = NULL`).

Note

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

Parameters

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.

7.36.6.13 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 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.

7.36.6.14 void spiReceive ([SPIDriver](#) * *spip*, size_t *n*, void * *rxbuf*)

Receives data from the SPI bus.

This synchronous function performs a receive operation.

Precondition

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

In order to use this function the driver must have been configured without callbacks (`end_cb = NULL`).

Note

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

Parameters

in	<i>spip</i>	pointer to the SPIDriver 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.

7.36.6.15 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.

7.36.6.16 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.

7.36.6.17 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:



7.36.6.18 void spi_llc_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.

7.36.6.19 void spi_llc_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.

7.36.6.20 void spi_llc_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.

7.36.6.21 void spi_llid_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.

7.36.6.22 void spi_llid_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.

7.36.6.23 void spi_llid_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.

7.36.6.24 void spi_ll_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.

7.36.6.25 void spi_ll_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>rxbuf</i>	the pointer to the receive buffer

Function Class:

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

7.36.6.26 `uint16_t spi_llc_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.

7.36.7 Variable Documentation

7.36.7.1 SPIDriver SPID1

SPI1 driver identifier.

7.37 ST Driver

Generic System Tick Driver.

7.37.1 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 (sys_time_t abstime)`
Starts the alarm.
- `void stStopAlarm (void)`
Stops the alarm interrupt.
- `void stSetAlarm (sys_time_t abstime)`
Sets the alarm time.
- `sys_time_t stGetAlarm (void)`
Returns the current alarm time.
- `void st_lld_init (void)`
Low level ST driver initialization.
- `static sys_time_t st_lld_get_counter (void)`
Returns the time counter value.
- `static void st_lld_start_alarm (sys_time_t abstime)`
Starts the alarm.
- `static void st_lld_stop_alarm (void)`
Stops the alarm interrupt.
- `static void st_lld_set_alarm (sys_time_t abstime)`
Sets the alarm time.
- `static sys_time_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.

7.37.2 Macro Definition Documentation

7.37.2.1 `#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.

7.37.2.2 #define stlsAlarmActive() 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.

7.37.3 Function Documentation**7.37.3.1 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:



7.37.3.2 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:



7.37.3.3 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:



7.37.3.4 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:



7.37.3.5 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:



7.37.3.6 void st_lld_init (void)

Low level ST driver initialization.

Function Class:

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

7.37.3.7 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.

7.37.3.8 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	<i>abstime</i>	the time to be set for the first alarm
----	----------------	----------------------------------------

Function Class:

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

7.37.3.9 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.

7.37.3.10 static void st_lld_set_alarm (systime_t *abstime*) [inline],[static]

Sets the alarm time.

Parameters

in	<i>abstime</i>	the time to be set for the next alarm
----	----------------	---------------------------------------

Function Class:

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

7.37.3.11 `static systime_t st_ild_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.

7.37.3.12 `static bool st_ild_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.

7.38 UART Driver

Generic UART Driver.

7.38.1 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.

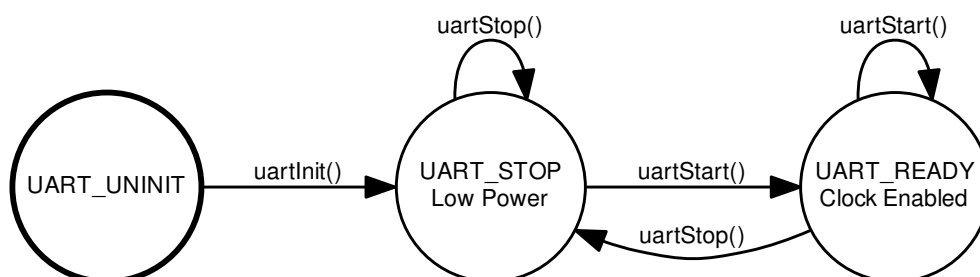
If your application requires a synchronous buffered driver then the [Serial Driver](#) should be used instead.

Precondition

In order to use the UART driver the `HAL_USE_UART` option must be enabled in `halconf.h`.

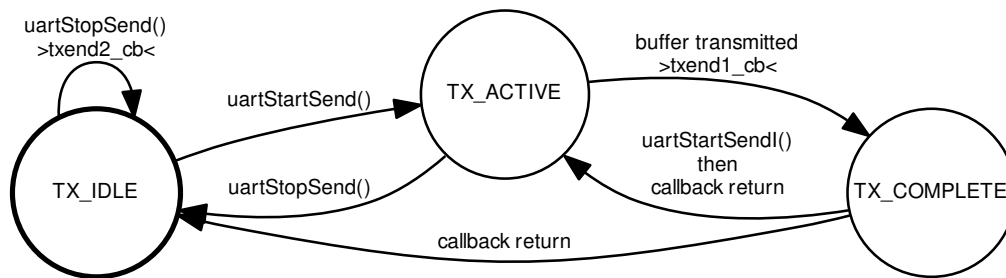
7.38.2 Driver State Machine

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



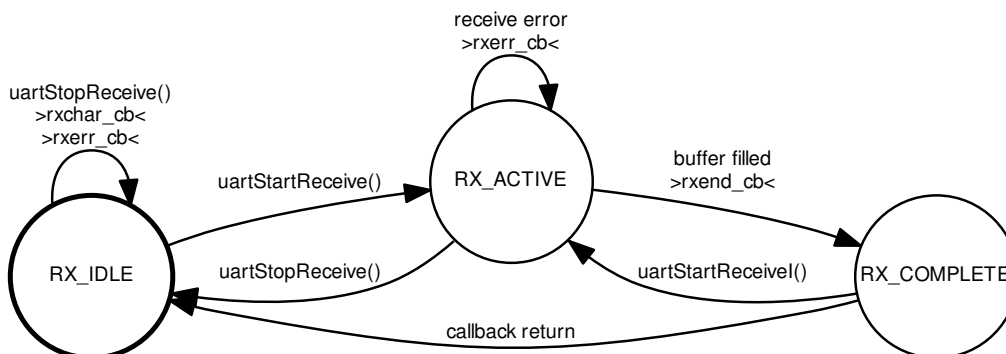
7.38.2.1 Transmitter sub State Machine

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.



7.38.2.2 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.*

UART configuration options

- #define `UART_USE_WAIT` FALSE
- Enables synchronous APIs.*
 - #define `UART_USE_MUTUAL_EXCLUSION` FALSE
 - Enables the `uartAcquireBus()` and `uartReleaseBus()` APIs.*

Low level driver helper macros

- #define `_uart_wakeup_tx1_isr(uartp)`
- Wakes up the waiting thread in case of early TX complete.*
 - #define `_uart_wakeup_tx2_isr(uartp)`
 - Wakes up the waiting thread in case of late TX complete.*
 - #define `_uart_wakeup_rx_complete_isr(uartp)`
 - Wakes up the waiting thread in case of RX complete.*
 - #define `_uart_wakeup_rx_error_isr(uartp)`
 - Wakes up the waiting thread in case of RX error.*
 - #define `_uart_wakeup_rx_timeout_isr(uartp)`
 - Wakes up the waiting thread in case of RX timeout.*
 - #define `_uart_tx1_isr_code(uartp)`
 - Common ISR code for early TX.*
 - #define `_uart_tx2_isr_code(uartp)`
 - Common ISR code for late TX.*
 - #define `_uart_rx_complete_isr_code(uartp)`
 - Common ISR code for RX complete.*
 - #define `_uart_rx_error_isr_code(uartp, errors)`
 - Common ISR code for RX error.*
 - #define `_uart_rx_idle_code(uartp)`
 - Common ISR code for RX on idle.*
 - #define `_uart_timeout_isr_code(uartp)`
 - Timeout ISR code for receiver.*

PLATFORM configuration options

- #define `PLATFORM_UART_USE_UART1` FALSE
- UART driver enable switch.*

Typedefs

- typedef uint32_t [uartflags_t](#)
UART driver condition flags type.
- typedef struct [UARTDriver](#) [UARTDriver](#)
Type of structure representing an UART driver.
- typedef void(* [uartcb_t](#)) ([UARTDriver](#) *uartp)
Generic UART notification callback type.
- typedef void(* [uartccb_t](#)) ([UARTDriver](#) *uartp, uint16_t c)
Character received UART notification callback type.
- typedef void(* [uartecb_t](#)) ([UARTDriver](#) *uartp, [uartflags_t](#) e)
Receive error UART notification callback type.

Data Structures

- struct [UARTConfig](#)
Driver configuration structure.
- struct [UARTDriver](#)
Structure representing an UART driver.

Functions

- void [uartInit](#) (void)
UART Driver initialization.
- void [uartObjectInit](#) ([UARTDriver](#) *uartp)
Initializes the standard part of a [UARTDriver](#) structure.
- void [uartStart](#) ([UARTDriver](#) *uartp, const [UARTConfig](#) *config)
Configures and activates the UART peripheral.
- void [uartStop](#) ([UARTDriver](#) *uartp)
Deactivates the UART peripheral.
- void [uartStartSend](#) ([UARTDriver](#) *uartp, size_t n, const void *txbuf)
Starts a transmission on the UART peripheral.
- void [uartStartSendI](#) ([UARTDriver](#) *uartp, size_t n, const void *txbuf)
Starts a transmission on the UART peripheral.
- size_t [uartStopSend](#) ([UARTDriver](#) *uartp)
Stops any ongoing transmission.
- size_t [uartStopSendI](#) ([UARTDriver](#) *uartp)
Stops any ongoing transmission.
- void [uartStartReceive](#) ([UARTDriver](#) *uartp, size_t n, void *rxbuf)
Starts a receive operation on the UART peripheral.
- void [uartStartReceiveI](#) ([UARTDriver](#) *uartp, size_t n, void *rxbuf)
Starts a receive operation on the UART peripheral.
- size_t [uartStopReceive](#) ([UARTDriver](#) *uartp)
Stops any ongoing receive operation.
- size_t [uartStopReceiveI](#) ([UARTDriver](#) *uartp)
Stops any ongoing receive operation.
- msg_t [uartSendTimeout](#) ([UARTDriver](#) *uartp, size_t *np, const void *txbuf, systime_t timeout)
Performs a transmission on the UART peripheral.
- msg_t [uartSendFullTimeout](#) ([UARTDriver](#) *uartp, size_t *np, const void *txbuf, systime_t timeout)
Performs a transmission on the UART peripheral.

- `msg_t uartReceiveTimeout (UARTDriver *uartp, size_t *np, void *rxbuf, systime_t timeout)`
Performs a receive operation on the UART peripheral.
- `void uartAcquireBus (UARTDriver *uartp)`
Gains exclusive access to the UART bus.
- `void uartReleaseBus (UARTDriver *uartp)`
Releases exclusive access to the UART bus.
- `void uart_lld_init (void)`
Low level UART driver initialization.
- `void uart_lld_start (UARTDriver *uartp)`
Configures and activates the UART peripheral.
- `void uart_lld_stop (UARTDriver *uartp)`
Deactivates the UART peripheral.
- `void uart_lld_start_send (UARTDriver *uartp, size_t n, const void *txbuf)`
Starts a transmission on the UART peripheral.
- `size_t uart_lld_stop_send (UARTDriver *uartp)`
Stops any ongoing transmission.
- `void uart_lld_start_receive (UARTDriver *uartp, size_t n, void *rxbuf)`
Starts a receive operation on the UART peripheral.
- `size_t uart_lld_stop_receive (UARTDriver *uartp)`
Stops any ongoing receive operation.

Enumerations

Variables

- `UARTDriver UARTD1`
UART1 driver identifier.

7.38.3 Macro Definition Documentation

7.38.3.1 `#define UART_NO_ERROR 0`

No pending conditions.

7.38.3.2 `#define UART_PARITY_ERROR 4`

Parity error happened.

7.38.3.3 `#define UART_FRAMING_ERROR 8`

Framing error happened.

7.38.3.4 `#define UART_OVERRUN_ERROR 16`

Overflow happened.

7.38.3.5 `#define UART_NOISE_ERROR 32`

Noise on the line.

7.38.3.6 `#define UART_BREAK_DETECTED 64`

Break detected.

7.38.3.7 `#define UART_USE_WAIT FALSE`

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.38.3.8 `#define UART_USE_MUTUAL_EXCLUSION FALSE`

Enables the `uartAcquireBus()` and `uartReleaseBus()` APIs.

Note

Disabling this option saves both code and data space.

7.38.3.9 `#define _uart_wakeup_tx1_isr(uartp)`**Value:**

```
{
    if ((uartp)->early == true) {
        osalSysLockFromISR();
        osalThreadResumeI (&(uartp)->threadtx, MSG_OK);
        osalSysUnlockFromISR();
    }
}
```

Wakes up the waiting thread in case of early TX complete.

Parameters

| | | |
|----|--------------------|-----------------------------------------------|
| in | <code>uartp</code> | pointer to the <code>UARTDriver</code> object |
|----|--------------------|-----------------------------------------------|

Function Class:

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

7.38.3.10 `#define _uart_wakeup_tx2_isr(uartp)`**Value:**

```
{
    if ((uartp)->early == false) {
        osalSysLockFromISR();
        osalThreadResumeI (&(uartp)->threadtx, MSG_OK);
        osalSysUnlockFromISR();
    }
}
```

Wakes up the waiting thread in case of late TX complete.

Parameters

| | | |
|----|--------------|--------------------------------------------------|
| in | <i>uartp</i> | pointer to the UARTDriver object |
|----|--------------|--------------------------------------------------|

Function Class:

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

7.38.3.11 #define _uart_wakeup_rx_complete_isr(*uartp*)**Value:**

```
{
    osalSysLockFromISR();
    osalThreadResumeI(&(uartp)->threadrx, MSG_OK);
    osalSysUnlockFromISR();
}
```

Wakes up the waiting thread in case of RX complete.

Parameters

| | | |
|----|--------------|--------------------------------------------------|
| in | <i>uartp</i> | pointer to the UARTDriver object |
|----|--------------|--------------------------------------------------|

Function Class:

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

7.38.3.12 #define _uart_wakeup_rx_error_isr(*uartp*)**Value:**

```
{
    osalSysLockFromISR();
    osalThreadResumeI(&(uartp)->threadrx, MSG_RESET);
    osalSysUnlockFromISR();
}
```

Wakes up the waiting thread in case of RX error.

Parameters

| | | |
|----|--------------|--------------------------------------------------|
| in | <i>uartp</i> | pointer to the UARTDriver object |
|----|--------------|--------------------------------------------------|

Function Class:

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

7.38.3.13 #define _uart_wakeup_rx_timeout_isr(*uartp*)**Value:**

```
{
    osalSysLockFromISR();
}
```

```
osalThreadResumeI(&(uartp)->threadrx, MSG_TIMEOUT);
osalSysUnlockFromISR();
}
```

Wakes up the waiting thread in case of RX timeout.

Parameters

| | | |
|----|--------------|--------------------------------------------------|
| in | <i>uartp</i> | pointer to the UARTDriver object |
|----|--------------|--------------------------------------------------|

Function Class:

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

7.38.3.14 #define _uart_tx1_isr_code(*uartp*)

Value:

```
{
    (uartp)->txstate = UART_TX_COMPLETE;
    if ((uartp)->config->txendl_cb != NULL) {
        (uartp)->config->txendl_cb(uartp);
    }
    if ((uartp)->txstate == UART_TX_COMPLETE) {
        (uartp)->txstate = UART_TX_IDLE;
    }
    _uart_wakeup_tx1_isr(uartp);
}
```

Common ISR code for early TX.

This code handles the portable part of the ISR code:

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

Note

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

Parameters

| | | |
|----|--------------|--------------------------------------------------|
| in | <i>uartp</i> | pointer to the UARTDriver object |
|----|--------------|--------------------------------------------------|

Function Class:

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

7.38.3.15 #define _uart_tx2_isr_code(*uartp*)

Value:

```
{
    if ((uartp)->config->txend2_cb != NULL) {
        (uartp)->config->txend2_cb(uartp);
    }
}
```

```

    }
    _uart_wakeup_tx2_isr(uartp);
}

```

Common ISR code for late TX.

This code handles the portable part of the ISR code:

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

Note

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

Parameters

| | | |
|----|--------------|--------------------------------------------------|
| in | <i>uartp</i> | pointer to the UARTDriver object |
|----|--------------|--------------------------------------------------|

Function Class:

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

7.38.3.16 #define _uart_rx_complete_isr_code(*uartp*)

Value:

```

{
    (uartp)->rxstate = UART_RX_COMPLETE;
    if ((uartp)->config->rxend_cb != NULL) {
        (uartp)->config->rxend_cb(uartp);
    }
    if ((uartp)->rxstate == UART_RX_COMPLETE) {
        (uartp)->rxstate = UART_RX_IDLE;
        uart_enter_rx_idle_loop(uartp);
    }
    _uart_wakeup_rx_complete_isr(uartp);
}

```

Common ISR code for RX complete.

This code handles the portable part of the ISR code:

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

Note

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

Parameters

| | | |
|----|--------------|--------------------------------------------------|
| in | <i>uartp</i> | pointer to the UARTDriver object |
|----|--------------|--------------------------------------------------|

Function Class:

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

7.38.3.17 #define _uart_rx_error_isr_code(uartp, errors)**Value:**

```
{
    if ((uartp)->config->rxerr_cb != NULL) {
        (uartp)->config->rxerr_cb(uartp, errors);
    }
    _uart_wakeup_rx_error_isr(uartp);
}
```

Common ISR code for RX error.

This code handles the portable part of the ISR code:

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

Note

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

Parameters

| | | |
|----|---------------|--------------------------------------------------|
| in | <i>uartp</i> | pointer to the UARTDriver object |
| in | <i>errors</i> | mask of errors to be reported |

Function Class:

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

7.38.3.18 #define _uart_rx_idle_code(uartp)**Value:**

```
{
    if ((uartp)->config->rxchar_cb != NULL)
        (uartp)->config->rxchar_cb(uartp, (uartp)->rxbuf);
}
```

Common ISR code for RX on idle.

This code handles the portable part of the ISR code:

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

Note

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

Parameters

| | | |
|----|--------------|--------------------------------------------------|
| in | <i>uartp</i> | pointer to the UARTDriver object |
|----|--------------|--------------------------------------------------|

Function Class:

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

7.38.3.19 #define _uart_timeout_isr_code(*uartp*)**Value:**

```
{
    if ((uartp)->config->timeout_cb != NULL) {
        (uartp)->config->timeout_cb(uartp);
    }
    _uart_wakeup_rx_timeout_isr(uartp);
}
```

Timeout ISR code for receiver.

This code handles the portable part of the ISR code:

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

Note

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

Parameters

| | | |
|----|--------------|--------------------------------------------------|
| in | <i>uartp</i> | pointer to the UARTDriver object |
|----|--------------|--------------------------------------------------|

Function Class:

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

7.38.3.20 #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`.

7.38.4 Typedef Documentation**7.38.4.1 typedef uint32_t uartflags_t**

UART driver condition flags type.

7.38.4.2 typedef struct UARTDriver UARTDriver

Type of structure representing an UART driver.

7.38.4.3 typedef void(* uartcb_t) (UARTDriver *uartp)

Generic UART notification callback type.

Parameters

| | | |
|----|--------------|--------------------------------------------------|
| in | <i>uartp</i> | pointer to the UARTDriver object |
|----|--------------|--------------------------------------------------|

7.38.4.4 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 |

7.38.4.5 typedef void(* uarteceb_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 |

7.38.5 Enumeration Type Documentation

7.38.5.1 enum uartstate_t

Driver state machine possible states.

Enumerator

UART_UNINIT Not initialized.

UART_STOP Stopped.

UART_READY Ready.

7.38.5.2 enum uarttxstate_t

Transmitter state machine states.

Enumerator

UART_TX_IDLE Not transmitting.

UART_TX_ACTIVE Transmitting.

UART_TX_COMPLETE Buffer complete.

7.38.5.3 enum uartrxstate_t

Receiver state machine states.

Enumerator

UART_RX_IDLE Not receiving.

UART_RX_ACTIVE Receiving.

UART_RX_COMPLETE Buffer complete.

7.38.6 Function Documentation

7.38.6.1 void uartInit (void)

UART Driver initialization.

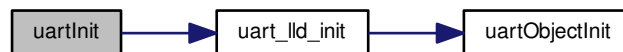
Note

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:



7.38.6.2 void uartObjectInit (UARTDriver * uartp)

Initializes the standard part of a `UARTDriver` structure.

Parameters

| | | |
|-----|--------------|-----------------------------------------------|
| out | <i>uartp</i> | 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.

7.38.6.3 void uartStart (UARTDriver * *uartp*, const UARTConfig * *config*)

Configures and activates the UART peripheral.

Parameters

| | | |
|----|---------------|--------------------------------------------------|
| in | <i>uartp</i> | pointer to the UARTDriver object |
| in | <i>config</i> | pointer to the UARTConfig object |

Function Class:

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

Here is the call graph for this function:

**7.38.6.4 void uartStop (UARTDriver * *uartp*)**

Deactivates the UART peripheral.

Parameters

| | | |
|----|--------------|--------------------------------------------------|
| in | <i>uartp</i> | pointer to the UARTDriver object |
|----|--------------|--------------------------------------------------|

Function Class:

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

Here is the call graph for this function:

**7.38.6.5 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 <code>UARTDriver</code> 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:



7.38.6.6 `void uartStartSendI (UARTDriver * uartp, size_t n, const void * txbuf)`

Starts a transmission on the UART peripheral.

Note

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

This function has to be invoked from a lock zone.

Parameters

| | | |
|----|--------------|-----------------------------------------------|
| in | <i>uartp</i> | pointer to the <code>UARTDriver</code> 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:



7.38.6.7 `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:



7.38.6.8 `size_t uartStopSendI (UARTDriver * uartp)`

Stops any ongoing transmission.

Note

Stopping a transmission also suppresses the transmission callbacks.
This function has to be invoked from a lock zone.

Parameters

| | | |
|----|--------------|--------------------------------------------------|
| in | <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:

**7.38.6.9 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:



7.38.6.10 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.

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 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.

Here is the call graph for this function:



7.38.6.11 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 <code>UARTDriver</code> 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:

**7.38.6.12 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 <code>UARTDriver</code> 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:



7.38.6.13 msg_t uartSendTimeout (UARTDriver * uartp, size_t * np, const void * txbuf, systime_t timeout)

Performs a transmission on the UART peripheral.

Note

The function returns when the specified number of frames have been sent to the UART or on timeout. The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays. This function implements a software timeout, it does not use any underlying HW timeout mechanism.

Parameters

| | | |
|---------|----------------|--------------------------------------------------------------------------------------|
| in | <i>uartp</i> | pointer to the UARTDriver object |
| in, out | <i>np</i> | number of data frames to transmit, on exit the number of frames actually transmitted |
| in | <i>txbuf</i> | the pointer to the transmit buffer |
| in | <i>timeout</i> | operation timeout |

Returns

The operation status.

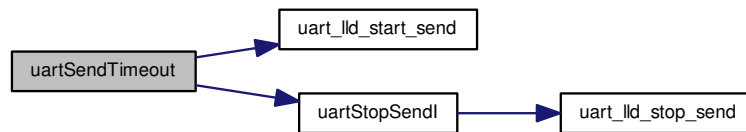
Return values

| | |
|--------------------|------------------------------------------|
| <i>MSG_OK</i> | if the operation completed successfully. |
| <i>MSG_TIMEOUT</i> | if the operation timed out. |

Function Class:

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

Here is the call graph for this function:



7.38.6.14 `msg_t uartSendFullTimeout (UARTDriver * uartp, size_t * np, const void * txbuf, systime_t timeout)`

Performs a transmission on the UART peripheral.

Note

The function returns when the specified number of frames have been physically transmitted or on timeout. The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays. This function implements a software timeout, it does not use any underlying HW timeout mechanism.

Parameters

| | | |
|---------|----------------|--------------------------------------------------------------------------------------|
| in | <i>uartp</i> | pointer to the UARTDriver object |
| in, out | <i>np</i> | number of data frames to transmit, on exit the number of frames actually transmitted |
| in | <i>txbuf</i> | the pointer to the transmit buffer |
| in | <i>timeout</i> | operation timeout |

Returns

The operation status.

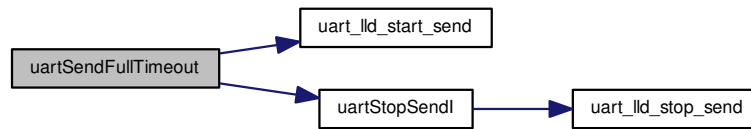
Return values

| | |
|--------------------|------------------------------------------|
| <i>MSG_OK</i> | if the operation completed successfully. |
| <i>MSG_TIMEOUT</i> | if the operation timed out. |

Function Class:

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

Here is the call graph for this function:



7.38.6.15 `msg_t uartReceiveTimeout (UARTDriver * uartp, size_t * np, void * rxbuf, systime_t timeout)`

Performs a receive operation on the UART peripheral.

Note

The function returns when the specified number of frames have been received or on error/timeout. The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays. This function implements a software timeout, it does not use any underlying HW timeout mechanism.

Parameters

| | | |
|---------|----------------|----------------------------------------------------------------------------------|
| in | <i>uartp</i> | pointer to the UARTDriver object |
| in, out | <i>np</i> | number of data frames to receive, on exit the number of frames actually received |
| in | <i>rxbuf</i> | the pointer to the receive buffer |
| in | <i>timeout</i> | operation timeout |

Returns

The operation status.

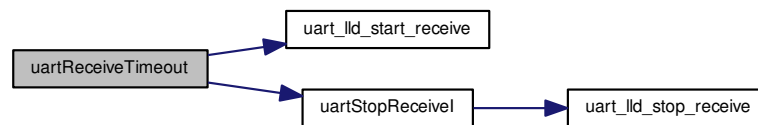
Return values

| | |
|--------------------|------------------------------------------|
| <i>MSG_OK</i> | if the operation completed successfully. |
| <i>MSG_TIMEOUT</i> | if the operation timed out. |
| <i>MSG_RESET</i> | in case of a receive error. |

Function Class:

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

Here is the call graph for this function:



7.38.6.16 void uartAcquireBus (UARTDriver * *uartp*)

Gains exclusive access to the UART bus.

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

Precondition

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

Parameters

| | | |
|----|--------------|-----------------------------------------------|
| in | <i>uartp</i> | 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.

7.38.6.17 void uartReleaseBus (UARTDriver * *uartp*)

Releases exclusive access to the UART bus.

Precondition

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

Parameters

| | | |
|----|--------------|-----------------------------------------------|
| in | <i>uartp</i> | 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.

7.38.6.18 void uart_ll_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:

**7.38.6.19 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.

7.38.6.20 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.

7.38.6.21 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.

7.38.6.22 `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.

7.38.6.23 `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.

7.38.6.24 `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.

7.38.7 Variable Documentation**7.38.7.1 UARTDriver UARTD1**

UART1 driver identifier.

7.39 USB Driver

Generic USB Driver.

7.39.1 Detailed Description

Generic USB Driver.

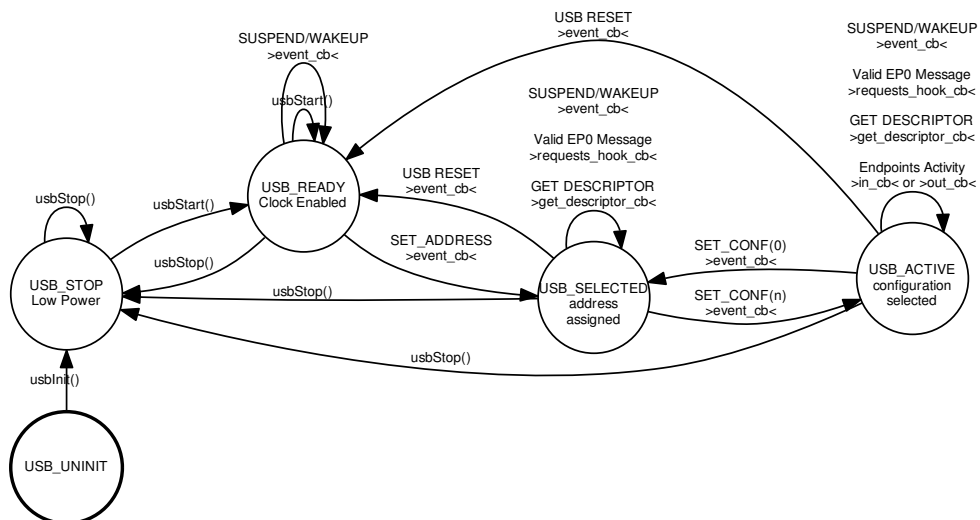
This module implements a generic USB (Universal Serial Bus) driver supporting device-mode operations.

Precondition

In order to use the USB driver the `HAL_USE_USB` option must be enabled in `halconf.h`.

7.39.2 Driver State Machine

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



7.39.3 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.

7.39.3.1 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

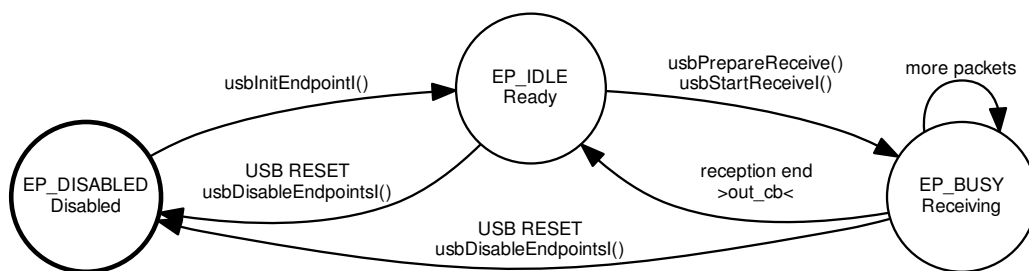
required by the USB device class to be implemented and of the handling of the specific messages sent over the endpoint zero. Standard messages are handled internally to the driver. The application can use hooks in order to handle custom messages or override the handling of the default handling of standard messages.

7.39.3.2 USB Endpoints

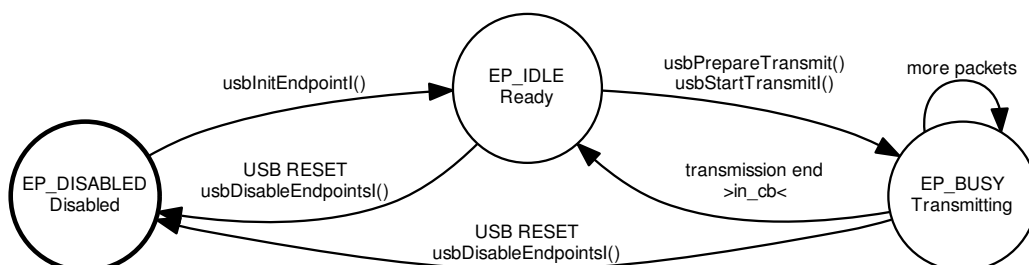
USB endpoints are the objects that the application uses to exchange data with the host. There are two kind of endpoints:

- **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:



7.39.3.3 USB Callbacks

The USB driver uses callbacks in order to interact with the application. There are several kinds of callbacks to be handled:

- Driver events callback. As example errors, suspend event, reset event etc.
- Messages Hook callback. This hook allows the application to implement handling of custom messages or to override the default handling of standard messages on endpoint zero.
- Descriptor Requested callback. When the driver endpoint zero handler receives a GET_DESCRIPTOR message and needs to send a descriptor to the host it queries the application using this callback.
- Start of Frame callback. This callback is invoked each time a SOF packet is received.
- Endpoint callbacks. Each endpoint informs the application about I/O conditions using those callbacks.

Macros

- `#define USB_USE_WAIT FALSE`
Enables synchronous APIs.
- `#define USB_MAX_ENDPOINTS 4`
Maximum endpoint address.
- `#define USB_EP0_STATUS_STAGE USB_EP0_STATUS_STAGE_SW`
Status stage handling method.
- `#define USB_SET_ADDRESS_MODE USB_LATE_SET_ADDRESS`
The address can be changed immediately upon packet reception.
- `#define USB_SET_ADDRESS_ACK_HANDLING USB_SET_ADDRESS_ACK_SW`
Method for set address acknowledge.
- `#define usb_lld_get_frame_number(usbp) 0`
Returns the current frame number.
- `#define usb_lld_get_transaction_size(usbp, ep) ((usbp)->epc[ep]->out_state->rxcnt)`
Returns the exact size of a receive transaction.
- `#define usb_lld_connect_bus(usbp)`
Connects the USB device.
- `#define usb_lld_disconnect_bus(usbp)`
Disconnect the USB device.

Helper macros for USB descriptors

- `#define USB_DESC_INDEX(i) ((uint8_t)(i))`
Helper macro for index values into descriptor strings.
- `#define USB_DESC_BYTE(b) ((uint8_t)(b))`
Helper macro for byte values into descriptor strings.
- `#define USB_DESC_WORD(w)`
Helper macro for word values into descriptor strings.
- `#define USB_DESC_BCD(bcd)`
Helper macro for BCD values into descriptor strings.
- `#define USB_DESC_DEVICE_SIZE 18U`
- `#define USB_DESC_DEVICE(bcdUSB, bDeviceClass, bDeviceSubClass, bDeviceProtocol, bMaxPacketSize, idVendor, idProduct, bcdDevice, iManufacturer, iProduct, iSerialNumber, bNumConfigurations)`
Device Descriptor helper macro.
- `#define USB_DESC_CONFIGURATION_SIZE 9U`

Configuration Descriptor size.

- #define `USB_DESC_CONFIGURATION`(wTotalLength, bNumInterfaces, bConfigurationValue, iConfiguration, bmAttributes, bMaxPower)

Configuration Descriptor helper macro.

- #define `USB_DESC_INTERFACE_SIZE` 9U

Interface Descriptor size.

- #define `USB_DESC_INTERFACE`(bInterfaceNumber, bAlternateSetting, bNumEndpoints, bInterfaceClass, bInterfaceSubClass, bInterfaceProtocol, iInterface)

Interface Descriptor helper macro.

- #define `USB_DESC_INTERFACE_ASSOCIATION_SIZE` 8U

Interface Association Descriptor size.

- #define `USB_DESC_INTERFACE_ASSOCIATION`(bFirstInterface, bInterfaceCount, bFunctionClass, bFunctionSubClass, bFunctionProtocol, iInterface)

Interface Association Descriptor helper macro.

- #define `USB_DESC_ENDPOINT_SIZE` 7U

Endpoint Descriptor size.

- #define `USB_DESC_ENDPOINT`(bEndpointAddress, bmAttributes, wMaxPacketSize, bInterval)

Endpoint Descriptor helper macro.

Endpoint types and settings

- #define `USB_EP_MODE_TYPE` 0x0003U
- #define `USB_EP_MODE_TYPE_CTRL` 0x0000U
- #define `USB_EP_MODE_TYPE_ISOC` 0x0001U
- #define `USB_EP_MODE_TYPE_BULK` 0x0002U
- #define `USB_EP_MODE_TYPE_INTR` 0x0003U

Macro Functions

- #define `usbGetDriverStateI`(usbp) ((usbp)->state)

Returns the driver state.

- #define `usbConnectBus`(usbp) `usb_lld_connect_bus`(usbp)

Connects the USB device.

- #define `usbDisconnectBus`(usbp) `usb_lld_disconnect_bus`(usbp)

Disconnect the USB device.

- #define `usbGetFrameNumberX`(usbp) `usb_lld_get_frame_number`(usbp)

Returns the current frame number.

- #define `usbGetTransmitStatusI`(usbp, ep) (((usbp)->transmitting & (uint16_t)((unsigned)1U << (unsigned)(ep))) != 0U)

Returns the status of an IN endpoint.

- #define `usbGetReceiveStatusI`(usbp, ep) (((usbp)->receiving & (uint16_t)((unsigned)1U << (unsigned)(ep))) != 0U)

Returns the status of an OUT endpoint.

- #define `usbGetReceiveTransactionSizeX`(usbp, ep) `usb_lld_get_transaction_size`(usbp, ep)

Returns the exact size of a receive transaction.

- #define `usbSetupTransfer`(usbp, buf, n, endcb)

Request transfer setup.

- #define `usbReadSetup`(usbp, ep, buf) `usb_lld_read_setup`(usbp, ep, buf)

Reads a setup packet from the dedicated packet buffer.

Low level driver helper macros

- `#define _usb_isr_invoke_event_cb(usbp, evt)`
Common ISR code, usb event callback.
- `#define _usb_isr_invoke_sof_cb(usbp)`
Common ISR code, SOF callback.
- `#define _usb_isr_invoke_setup_cb(usbp, ep)`
Common ISR code, setup packet callback.
- `#define _usb_isr_invoke_in_cb(usbp, ep)`
Common ISR code, IN endpoint callback.
- `#define _usb_isr_invoke_out_cb(usbp, ep)`
Common ISR code, OUT endpoint event.

PLATFORM configuration options

- `#define PLATFORM_USB_USE_USB1 FALSE`
USB driver enable switch.

Typedefs

- `typedef struct USBDriver USBDriver`
Type of a structure representing an USB driver.
- `typedef uint8_t usbep_t`
Type of an endpoint identifier.
- `typedef void(* usbcallback_t) (USBDriver *usbp)`
Type of an USB generic notification callback.
- `typedef void(* usbepcallback_t) (USBDriver *usbp, usbep_t ep)`
Type of an USB endpoint callback.
- `typedef void(* usbeventcb_t) (USBDriver *usbp, usbevent_t event)`
Type of an USB event notification callback.
- `typedef bool(* usbreqhandler_t) (USBDriver *usbp)`
Type of a requests handler callback.
- `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 `usbInitEndpointI` (`USBDriver *usbp`, `usbep_t ep`, const `USBEndpointConfig *epcp`)
Enables an endpoint.
- void `usbDisableEndpointsI` (`USBDriver *usbp`)
Disables all the active endpoints.
- void `usbStartReceiveI` (`USBDriver *usbp`, `usbep_t ep`, `uint8_t *buf`, `size_t n`)
Starts a receive transaction on an OUT endpoint.
- void `usbStartTransmitI` (`USBDriver *usbp`, `usbep_t ep`, const `uint8_t *buf`, `size_t n`)
Starts a transmit transaction on an IN endpoint.
- `msg_t` `usbReceive` (`USBDriver *usbp`, `usbep_t ep`, `uint8_t *buf`, `size_t n`)
Performs a receive transaction on an OUT endpoint.
- `msg_t` `usbTransmit` (`USBDriver *usbp`, `usbep_t ep`, const `uint8_t *buf`, `size_t n`)
Performs a transmit transaction on an IN endpoint.
- bool `usbStallReceiveI` (`USBDriver *usbp`, `usbep_t ep`)
Stalls an OUT endpoint.
- bool `usbStallTransmitI` (`USBDriver *usbp`, `usbep_t ep`)
Stalls an IN endpoint.
- void `_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

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.*

7.39.4 Macro Definition Documentation

7.39.4.1 #define USB_DESC_INDEX(i) ((uint8_t)(i))

Helper macro for index values into descriptor strings.

7.39.4.2 #define USB_DESC_BYTE(b) ((uint8_t)(b))

Helper macro for byte values into descriptor strings.

7.39.4.3 #define USB_DESC_WORD(w)

Value:

```
(uint8_t)((w) & 255U),  
(uint8_t)((w) >> 8) & 255U) \
```

Helper macro for word values into descriptor strings.

7.39.4.4 #define USB_DESC_BCD(bcd)

Value:

```
(uint8_t)((bcd) & 255U),  
(uint8_t)((bcd) >> 8) & 255) \
```

Helper macro for BCD values into descriptor strings.

7.39.4.5 #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.

7.39.4.6 #define USB_DESC_CONFIGURATION_SIZE 9U

Configuration Descriptor size.

7.39.4.7 **#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.

7.39.4.8 **#define** USB_DESC_INTERFACE_SIZE 9U

Interface Descriptor size.

7.39.4.9 **#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.

7.39.4.10 **#define** USB_DESC_INTERFACE_ASSOCIATION_SIZE 8U

Interface Association Descriptor size.

7.39.4.11 **#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.

7.39.4.12 **#define USB_DESC_ENDPOINT_SIZE 7U**

Endpoint Descriptor size.

7.39.4.13 **#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.

7.39.4.14 **#define USB_EP_MODE_TYPE 0x0003U**

Endpoint type mask.

7.39.4.15 **#define USB_EP_MODE_TYPE_CTRL 0x0000U**

Control endpoint.

7.39.4.16 **#define USB_EP_MODE_TYPE_ISOC 0x0001U**

Isochronous endpoint.

7.39.4.17 **#define USB_EP_MODE_TYPE_BULK 0x0002U**

Bulk endpoint.

7.39.4.18 **#define USB_EP_MODE_TYPE_INTR 0x0003U**

Interrupt endpoint.

7.39.4.19 `#define USB_USE_WAIT FALSE`

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.39.4.20 `#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.

7.39.4.21 `#define usbConnectBus(usbp) usb_lld_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.

7.39.4.22 `#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.

7.39.4.23 `#define usbGetFrameNumberX(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:

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

```
7.39.4.24 #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.

```
7.39.4.25 #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.

7.39.4.26 `#define usbGetReceiveTransactionSizeX(usbp, ep) usb_lld_get_transaction_size(usbp, ep)`

Returns the exact size of a receive transaction.

The received size can be different from the size specified in `usbStartReceiveI()` because the last packet could have a size different from the expected one.

Parameters

| | | |
|----|-------------|----------------------------------------------|
| in | <i>usbp</i> | pointer to the <code>USBDriver</code> object |
| in | <i>ep</i> | endpoint number |

Returns

Received data size.

Function Class:

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

7.39.4.27 `#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 <code>USBDriver</code> 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 <code>NULL</code> |

Function Class:

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

7.39.4.28 `#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.

7.39.4.29 `#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.

7.39.4.30 `#define _usb_isr_invoke_sof_cb(usbp)`

Value:

```
{
    if ((usbp->config->sof_cb) != NULL) {
        (usbp->config->sof_cb)(usbp);
    }
}
```

Common ISR code, SOF callback.

Parameters

| | | |
|----|-------------|-------------------------------------------------|
| in | <i>usbp</i> | pointer to the USBDriver object |
|----|-------------|-------------------------------------------------|

Function Class:

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

7.39.4.31 #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.

7.39.4.32 #define _usb_isr_invoke_in_cb(usbp, ep)**Value:**

```
{
    (usbp)->transmitting &= ~(1 << (ep));
    if ((usbp)->epc[ep]->in_cb != NULL) {
        (usbp)->epc[ep]->in_cb(usbp, ep);
    }
    osalSysLockFromISR();
    osalThreadResumeI(&(usbp)->epc[ep]->in_state->thread, MSG_OK);
    osalSysUnlockFromISR();
}
```

Common ISR code, IN endpoint callback.

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.

7.39.4.33 #define _usb_isr_invoke_out_cb(usbp, ep)**Value:**


```

{
    (usbp)->receiving &= ~(1 << (ep)); \
    if ((usbp)->epc[ep]->out_cb != NULL) { \
        (usbp)->epc[ep]->out_cb(usbp, ep); \
    } \
    osalSysLockFromISR(); \
    osalThreadResumeI (&(usbp)->epc[ep]->out_state->thread, \
        usbGetReceiveTransactionSizeX(usbp, ep)); \
    osalSysUnlockFromISR(); \
}

```

Common ISR code, OUT endpoint event.

Parameters

| | | |
|----|-------------|-------------------------------------------------|
| 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.

7.39.4.34 #define USB_MAX_ENDPOINTS 4

Maximum endpoint address.

7.39.4.35 #define USB_EP0_STATUS_STAGE USB_EP0_STATUS_STAGE_SW

Status stage handling method.

7.39.4.36 #define USB_SET_ADDRESS_MODE USB_LATE_SET_ADDRESS

The address can be changed immediately upon packet reception.

7.39.4.37 #define USB_SET_ADDRESS_ACK_HANDLING USB_SET_ADDRESS_ACK_SW

Method for set address acknowledge.

7.39.4.38 #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`.

7.39.4.39 #define usb_lld_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.

7.39.4.40 `#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 | <i>usbp</i> | pointer to the <code>USBDriver</code> object |
| in | <i>ep</i> | endpoint number |

Returns

Received data size.

Function Class:

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

7.39.4.41 `#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.

7.39.4.42 `#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.

7.39.5 Typedef Documentation

7.39.5.1 `typedef struct USBDriver USBDriver`

Type of a structure representing an USB driver.

7.39.5.2 typedef uint8_t usbep_t

Type of an endpoint identifier.

7.39.5.3 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 |
|----|-------------|-------------------------------------------------------------------------|

7.39.5.4 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 |

7.39.5.5 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 |

7.39.5.6 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. |

7.39.5.7 `typedef const USBDescriptor*(* usbgetdescriptor_t) (USBDriver *usb, uint8_t dtype, uint8_t dindex, uint16_t lang)`

Type of an USB descriptor-retrieving callback.

7.39.6 Enumeration Type Documentation

7.39.6.1 `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.

7.39.6.2 `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.

7.39.6.3 `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.

7.39.6.4 `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_UNCONFIGURED Configuration removed.
USB_EVENT_SUSPEND Entering suspend mode.
USB_EVENT_WAKEUP Leaving suspend mode.
USB_EVENT_STALLED Endpoint 0 error, stalled.

7.39.7 Function Documentation

7.39.7.1 `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:



7.39.7.2 `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 |
|----|-------------|-------------------------------------------------|

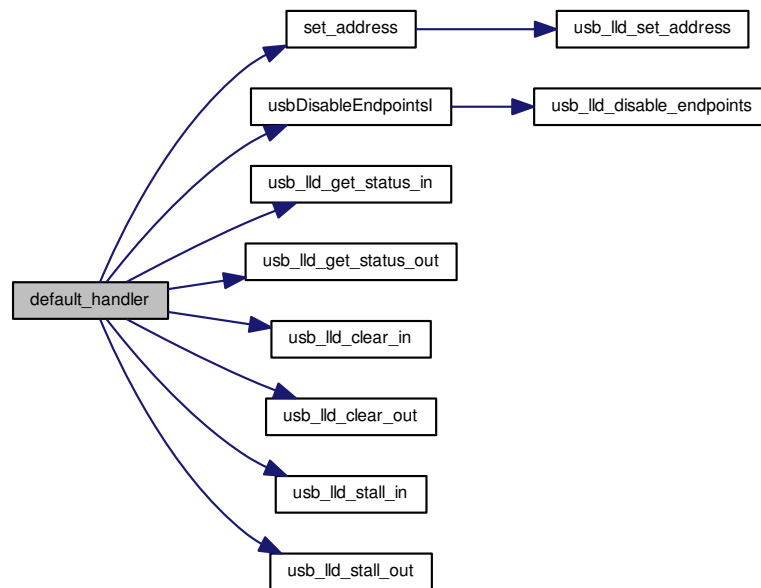
Returns

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:



7.39.7.3 void usbInit (void)

USB Driver initialization.

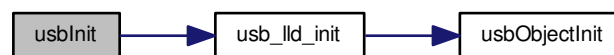
Note

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

Function Class:

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

Here is the call graph for this function:



7.39.7.4 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.

7.39.7.5 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:

**7.39.7.6 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:



7.39.7.7 void usbInitEndpointI (USBDriver * *usbp*, usbp_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:



7.39.7.8 void usbDisableEndpointsI (USBDriver * *usbp*)

Disables all the active endpoints.

This function disables all the active endpoints except the endpoint zero.

Note

This function must be invoked in response of a SET_CONFIGURATION message with configuration number zero.

Parameters

| | | |
|----|-------------|-------------------------------------------------|
| in | <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:



7.39.7.9 void usbStartReceive (USBDriver * *usbp*, *usbep_t* *ep*, *uint8_t* * *buf*, *size_t* *n*)

Starts a receive transaction on an OUT endpoint.

Note

This function is meant to be called from ISR context outside critical zones because there is a potentially slow operation inside.

Parameters

| | | |
|-----|-------------|----------------------------------------------------------------------------------------------------|
| in | <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. It is recommended a multiple of the packet size because the excess is discarded. |

Function Class:

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

Here is the call graph for this function:



7.39.7.10 void usbStartTransmitl (USBDriver * *usbp*, usbep_t *ep*, const uint8_t * *buf*, size_t *n*)

Starts a transmit transaction on an IN endpoint.

Note

This function is meant to be called from ISR context outside critical zones because there is a potentially slow operation inside.

Parameters

| | | |
|----|-------------|--------------------------------------------------|
| 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:

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:



7.39.7.11 msg_t usbReceive (USBDriver * *usbp*, usbep_t *ep*, uint8_t * *buf*, size_t *n*)

Performs a receive transaction on an OUT endpoint.

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. It is recommended a multiple of the packet size because the excess is discarded. |

Returns

The received effective data size, it can be less than the amount specified.

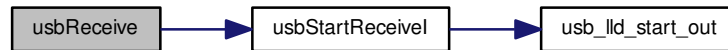
Return values

| | |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>MSG_RESET</i> | driver not in <code>USB_ACTIVE</code> state or the operation has been aborted by an USB reset or a transition to the <code>USB_SUSPENDED</code> state. |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|

Function Class:

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

Here is the call graph for this function:



7.39.7.12 `msg_t usbTransmit (USBDriver * usbp, usbep_t ep, const uint8_t * buf, size_t n)`

Performs a transmit transaction on an IN endpoint.

Parameters

| | | |
|----|-------------|--------------------------------------------------|
| in | <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 |

Returns

The operation status.

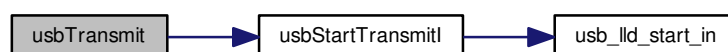
Return values

| | |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>MSG_OK</i> | operation performed successfully. |
| <i>MSG_RESET</i> | driver not in <code>USB_ACTIVE</code> state or the operation has been aborted by an USB reset or a transition to the <code>USB_SUSPENDED</code> state. |

Function Class:

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

Here is the call graph for this function:



7.39.7.13 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:



7.39.7.14 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:

**7.39.7.15 void _usb_reset (USBDriver * usbp)**

USB reset routine.

This function must be invoked when an USB bus reset condition is detected.

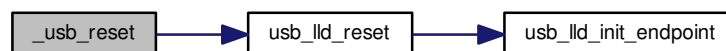
Parameters

| | | |
|----|-------------|-------------------------------------------------|
| 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:

**7.39.7.16 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.

7.39.7.17 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.

7.39.7.18 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.

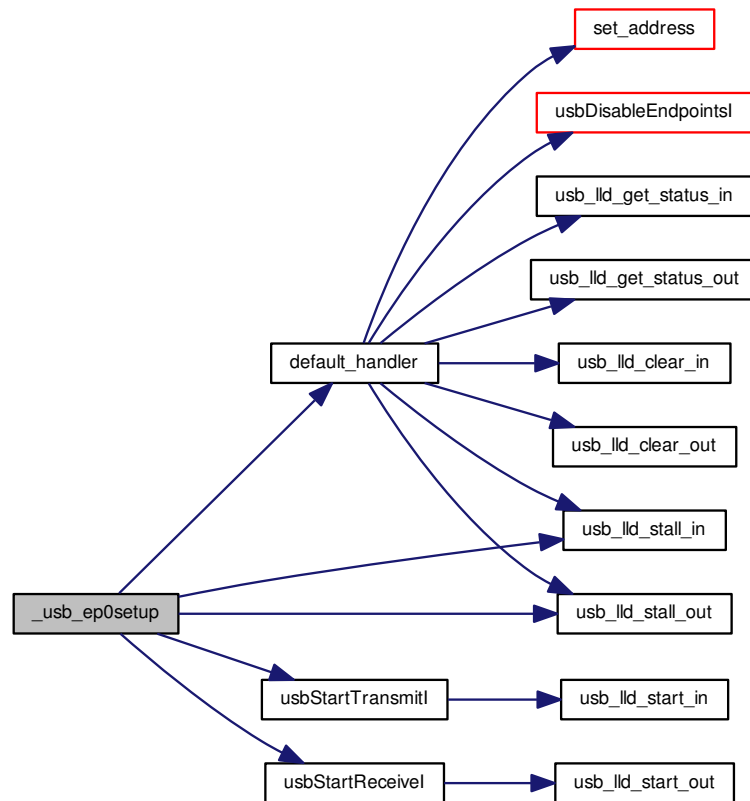
Parameters

| | | |
|----|-------------|-------------------------------------------------|
| 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:



7.39.7.19 `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.

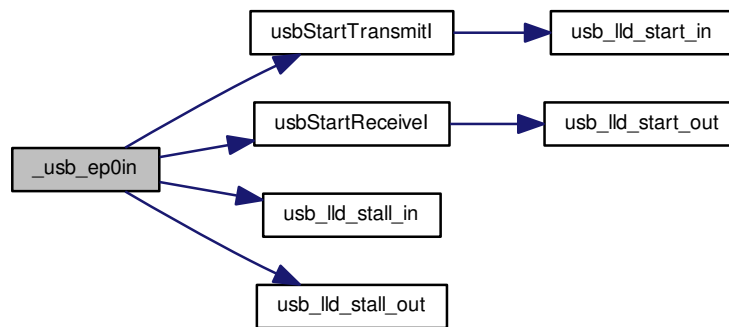
Parameters

| | | |
|----|-------------|-------------------------------------------------|
| 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:



7.39.7.20 `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.

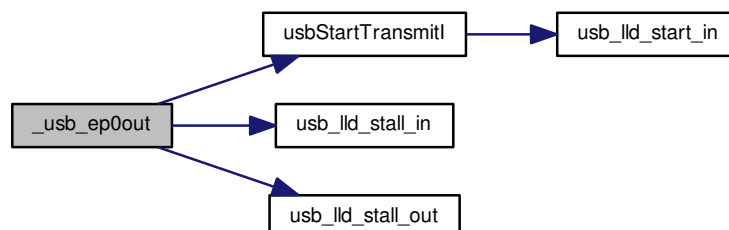
Parameters

| | | |
|----|-------------|-------------------------------------------------|
| 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:



7.39.7.21 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:

**7.39.7.22 void usb_lld_start (USBDriver * usbp)**

Configures and activates 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.

7.39.7.23 void usb_lld_stop (USBDriver * usbp)

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.

7.39.7.24 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:



7.39.7.25 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.

7.39.7.26 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.

7.39.7.27 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.

7.39.7.28 `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.

7.39.7.29 `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.

7.39.7.30 `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.

7.39.7.31 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.

7.39.7.32 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.

7.39.7.33 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.

7.39.7.34 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.

7.39.7.35 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.

7.39.7.36 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.

7.39.7.37 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.

7.39.7.38 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 USBDriver object |
| in | <i>ep</i> | endpoint number |

Function Class:

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

7.39.8 Variable Documentation

7.39.8.1 USBDriver USBD1

USB1 driver identifier.

7.39.8.2 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.

7.39.8.3 USBInEndpointState { ... } in

IN EP0 state.

7.39.8.4 USBOutEndpointState { ... } out

OUT EP0 state.

7.39.8.5 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.

7.40 USB CDC Header

USB CDC Support Header.

7.40.1 Detailed Description

USB CDC Support Header.

This header contains definitions and types related to USB CDC.

CDC specific messages.

- `#define CDC_SEND_ENCAPSULATED_COMMAND 0x00U`
- `#define CDC_GET_ENCAPSULATED_RESPONSE 0x01U`
- `#define CDC_SET_COMM_FEATURE 0x02U`
- `#define CDC_GET_COMM_FEATURE 0x03U`
- `#define CDC_CLEAR_COMM_FEATURE 0x04U`
- `#define CDC_SET_AUX_LINE_STATE 0x10U`
- `#define CDC_SET_HOOK_STATE 0x11U`
- `#define CDC_PULSE_SETUP 0x12U`
- `#define CDC_SEND_PULSE 0x13U`
- `#define CDC_SET_PULSE_TIME 0x14U`
- `#define CDC_RING_AUX_JACK 0x15U`
- `#define CDC_SET_LINE_CODING 0x20U`
- `#define CDC_GET_LINE_CODING 0x21U`
- `#define CDC_SET_CONTROL_LINE_STATE 0x22U`
- `#define CDC_SEND_BREAK 0x23U`
- `#define CDC_SET_RINGER_PARMS 0x30U`
- `#define CDC_GET_RINGER_PARMS 0x31U`
- `#define CDC_SET_OPERATION_PARMS 0x32U`
- `#define CDC_GET_OPERATION_PARMS 0x33U`

CDC classes

- `#define CDC_COMMUNICATION_INTERFACE_CLASS 0x02U`
- `#define CDC_DATA_INTERFACE_CLASS 0x0AU`

CDC subclasses

- `#define CDC_ABSTRACT_CONTROL_MODEL 0x02U`

CDC descriptors

- `#define CDC_CS_INTERFACE 0x24U`

CDC subdescriptors

- `#define CDC_HEADER 0x00U`
- `#define CDC_CALL_MANAGEMENT 0x01U`
- `#define CDC_ABSTRACT_CONTROL_MANAGEMENT 0x02U`
- `#define CDC_UNION 0x06U`

Line Control bit definitions.

- `#define LC_STOP_1 0U`
- `#define LC_STOP_1P5 1U`
- `#define LC_STOP_2 2U`
- `#define LC_PARITY_NONE 0U`
- `#define LC_PARITY_ODD 1U`
- `#define LC_PARITY_EVEN 2U`
- `#define LC_PARITY_MARK 3U`
- `#define LC_PARITY_SPACE 4U`

Data Structures

- `struct cdc_linecoding_t`
Type of Line Coding structure.

7.41 WDG Driver

Generic WDG Driver.

7.41.1 Detailed Description

Generic WDG Driver.

This module defines an abstract interface for a watchdog timer.

Precondition

In order to use the WDG driver the `HAL_USE_WDG` option must be enabled in `halconf.h`.

Macros

- `#define wdgReset(wdgp) wdg_lld_reset(wdgp)`
Resets WDG's counter.

Configuration options

- `#define PLATFORM_WDG_USE_WDG1 FALSE`
WDG1 driver enable switch.

Typedefs

- `typedef struct WDGDriver WDGDriver`
Type of a structure representing an WDG driver.

Data Structures

- `struct WDGConfig`
Driver configuration structure.
- `struct WDGDriver`
Structure representing an WDG driver.

Functions

- `void wdgInit (void)`
WDG Driver initialization.
- `void wdgStart (WDGDriver *wdgp, const WDGConfig *config)`
Configures and activates the WDG peripheral.
- `void wdgStop (WDGDriver *wdgp)`
Deactivates the WDG peripheral.
- `void wdgReset (WDGDriver *wdgp)`
Resets WDG's counter.
- `void wdg_lld_init (void)`
Low level WDG driver initialization.
- `void wdg_lld_start (WDGDriver *wdgp)`
Configures and activates the WDG peripheral.
- `void wdg_lld_stop (WDGDriver *wdgp)`

Deactivates the WDG peripheral.

- void `wdg_lld_reset` (`WDGDriver` *wdgp)

Reloads WDG's counter.

Enumerations

7.41.2 Macro Definition Documentation

7.41.2.1 `#define wdgResetl(wdgp) wdg_lld_reset(wdgp)`

Resets WDG's counter.

Parameters

| | | |
|----|------|----------------------------------------------|
| in | wdgp | pointer to the <code>WDGDriver</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.

7.41.2.2 `#define PLATFORM_WDG_USE_WDG1 FALSE`

WDG1 driver enable switch.

Note

The default is `FALSE`.

7.41.3 Typedef Documentation

7.41.3.1 `typedef struct WDGDriver WDGDriver`

Type of a structure representing an WDG driver.

7.41.4 Enumeration Type Documentation

7.41.4.1 `enum wdgstate_t`

Driver state machine possible states.

Enumerator

`WDG_UNINIT` Not initialized.

`WDG_STOP` Stopped.

`WDG_READY` Ready.

7.41.5 Function Documentation

7.41.5.1 `void wdgInit(void)`

WDG Driver initialization.

Note

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

Function Class:

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

Here is the call graph for this function:

**7.41.5.2 void wdgStart (WDGDriver * wdgp, const WDGConfig * config)**

Configures and activates the WDG peripheral.

Parameters

| | | |
|----|---------------|-------------------------------------------------|
| in | <i>wdgp</i> | pointer to the WDGDriver object |
| in | <i>config</i> | pointer to the WDGConfig object |

Function Class:

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

Here is the call graph for this function:

**7.41.5.3 void wdgStop (WDGDriver * wdgp)**

Deactivates the WDG peripheral.

Parameters

| | | |
|----|-------------|-------------------------------------------------|
| in | <i>wdgp</i> | pointer to the WDGDriver object |
|----|-------------|-------------------------------------------------|

Function Class:

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

Here is the call graph for this function:



7.41.5.4 void wdgReset (WDGDriver * wdgp)

Resets WDG's counter.

Parameters

| | | |
|----|------|-------------------------------------------------|
| in | wdgp | pointer to the WDGDriver object |
|----|------|-------------------------------------------------|

Function Class:

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

7.41.5.5 void wdg_llid_init (void)

Low level WDG driver initialization.

Function Class:

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

7.41.5.6 void wdg_llid_start (WDGDriver * wdgp)

Configures and activates the WDG peripheral.

Parameters

| | | |
|----|------|-------------------------------------------------|
| in | wdgp | pointer to the WDGDriver object |
|----|------|-------------------------------------------------|

Function Class:

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

7.41.5.7 void wdg_llid_stop (WDGDriver * wdgp)

Deactivates the WDG peripheral.

Parameters

| | | |
|----|-------------|-------------------------------------------------|
| in | <i>wdgp</i> | pointer to the WDGDriver object |
|----|-------------|-------------------------------------------------|

Function Class:

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

7.41.5.8 void `wdg_lld_reset` ([WDGDriver](#) * *wdgp*)

Reloads WDG's counter.

Parameters

| | | |
|----|-------------|-------------------------------------------------|
| in | <i>wdgp</i> | pointer to the WDGDriver object |
|----|-------------|-------------------------------------------------|

Function Class:

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

Chapter 8

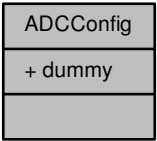
Data Structure Documentation

8.1 ADCConfig Struct Reference

Driver configuration structure.

```
#include <hal_adc_ll.h>
```

Collaboration diagram for ADCConfig:



8.1.1 Detailed Description

Driver configuration structure.

Note

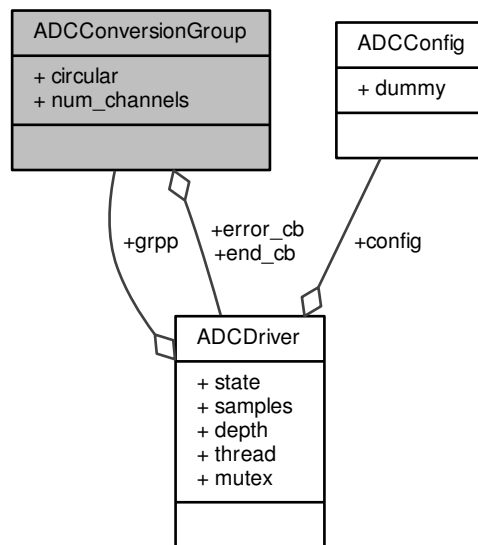
It could be empty on some architectures.

8.2 ADCConversionGroup Struct Reference

Conversion group configuration structure.

```
#include <hal_adc_llld.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 ADCCConversionGroup::circular`

Enables the circular buffer mode for the group.

8.2.2.2 `adc_channels_num_t ADCCConversionGroup::num_channels`

Number of the analog channels belonging to the conversion group.

8.2.2.3 `adccallback_t ADCCConversionGroup::end_cb`

Callback function associated to the group or `NULL`.

8.2.2.4 `adcerrorcallback_t ADCCConversionGroup::error_cb`

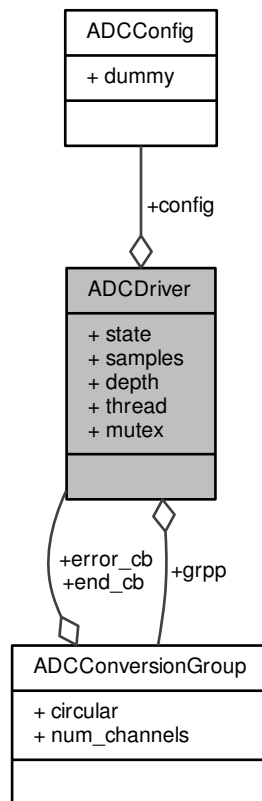
Error callback or `NULL`.

8.3 ADCDriver Struct Reference

Structure representing an ADC driver.

```
#include <hal_adc_llid.h>
```

Collaboration diagram for ADCDriver:



Data Fields

- `adcstate_t state`
Driver state.
- `const ADCCConfig * config`
Current configuration data.
- `adcsample_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

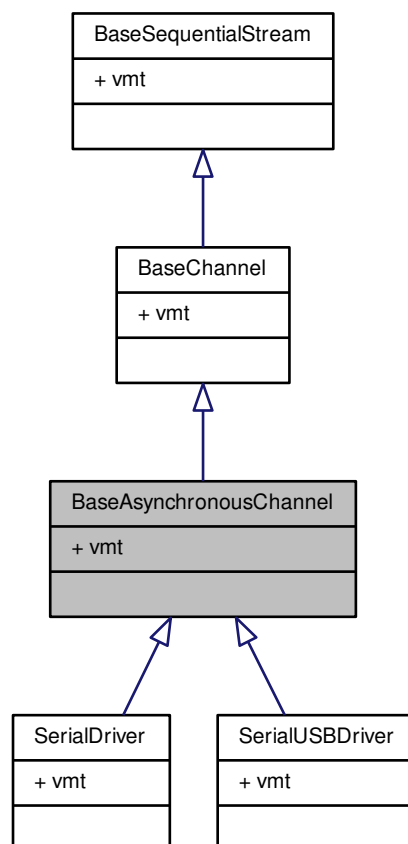
Mutex protecting the peripheral.

8.4 BaseAsynchronousChannel Struct Reference

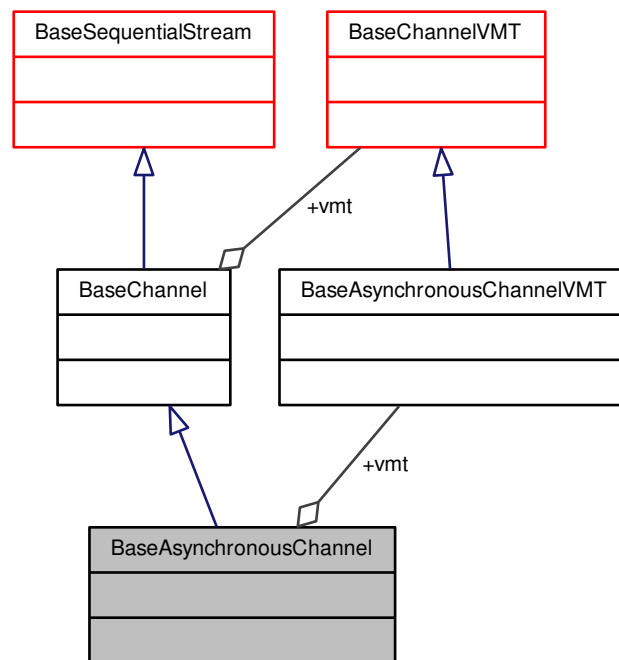
Base asynchronous channel class.

```
#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

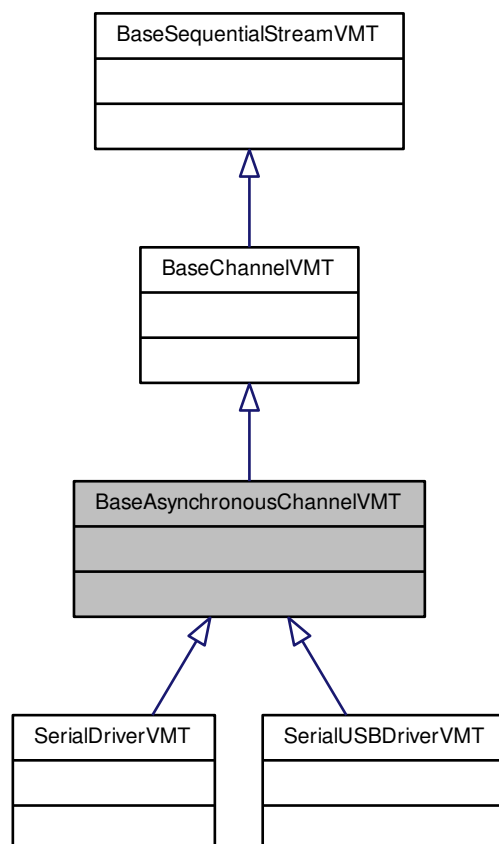
Virtual Methods Table.

8.5 BaseAsynchronousChannelVMT Struct Reference

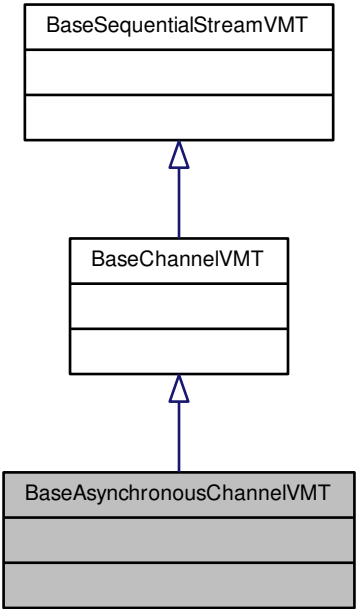
[BaseAsynchronousChannel](#) virtual methods table.

```
#include <hal_channels.h>
```

Inheritance diagram for BaseAsynchronousChannelVMT:



Collaboration diagram for BaseAsynchronousChannelVMT:



8.5.1 Detailed Description

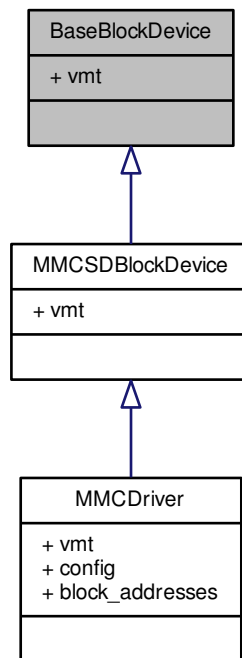
BaseAsynchronousChannel virtual methods table.

8.6 BaseBlockDevice Struct Reference

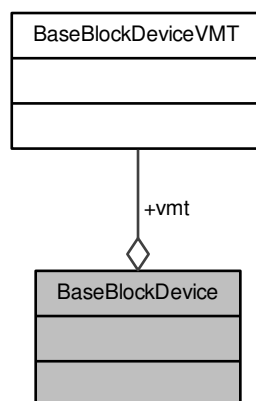
Base block device class.

```
#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`

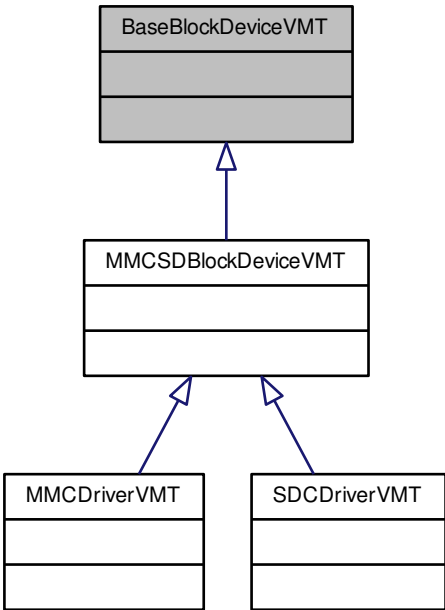
Virtual Methods Table.

8.7 BaseBlockDeviceVMT Struct Reference

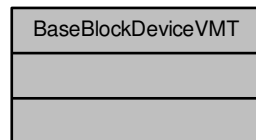
`BaseBlockDevice` virtual methods table.

```
#include <hal_ioblock.h>
```

Inheritance diagram for BaseBlockDeviceVMT:



Collaboration diagram for BaseBlockDeviceVMT:



8.7.1 Detailed Description

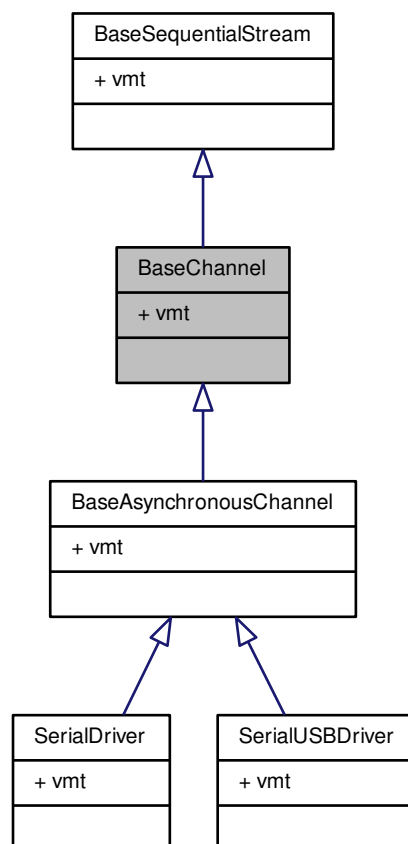
[BaseBlockDevice](#) virtual methods table.

8.8 BaseChannel Struct Reference

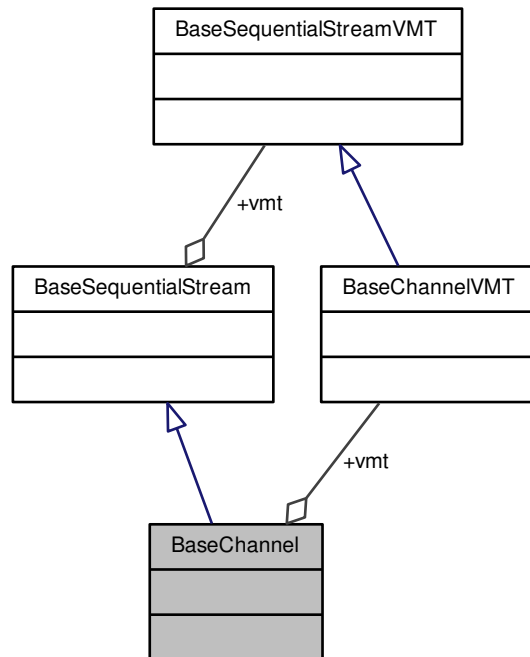
Base channel class.

```
#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**

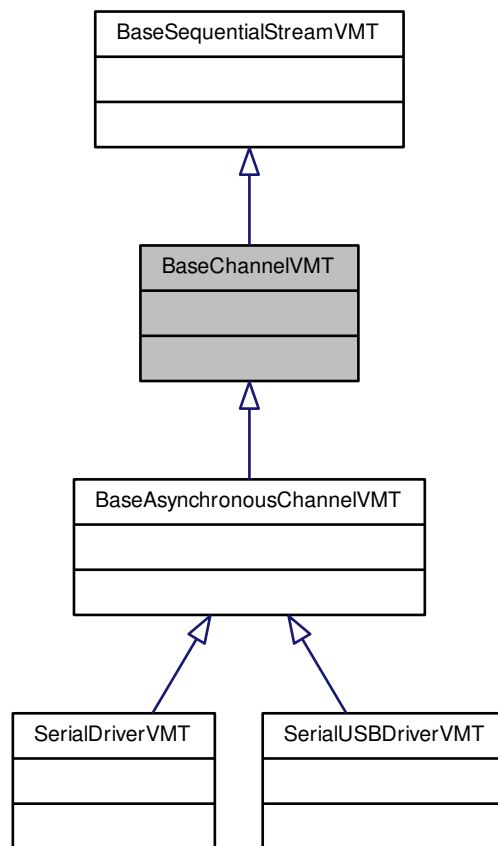
Virtual Methods Table.

8.9 BaseChannelVMT Struct Reference

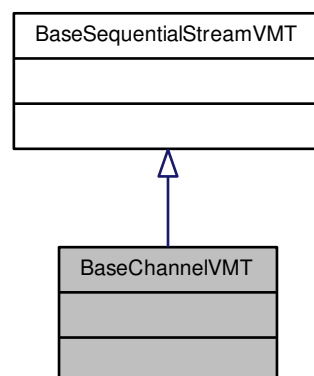
[BaseChannel](#) virtual methods table.

```
#include <hal_channels.h>
```

Inheritance diagram for BaseChannelVMT:



Collaboration diagram for BaseChannelVMT:



8.9.1 Detailed Description

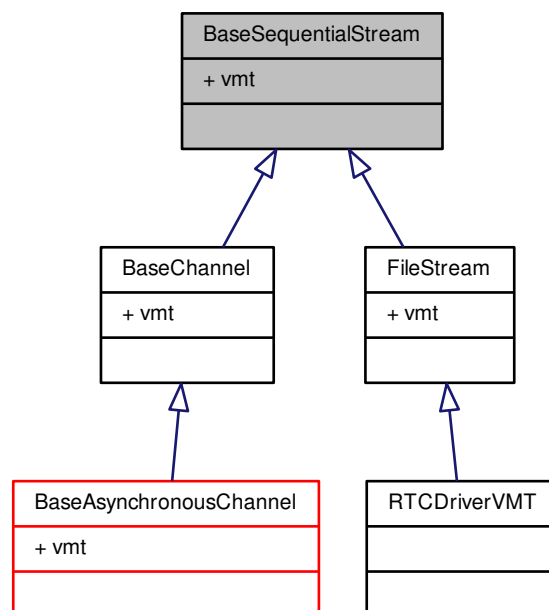
[BaseChannel](#) virtual methods table.

8.10 BaseSequentialStream Struct Reference

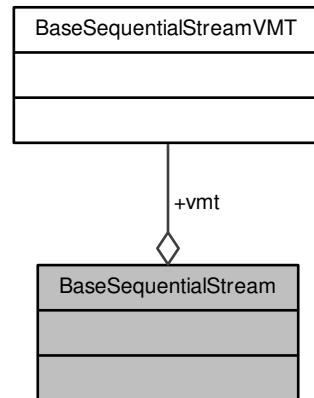
Base stream class.

```
#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

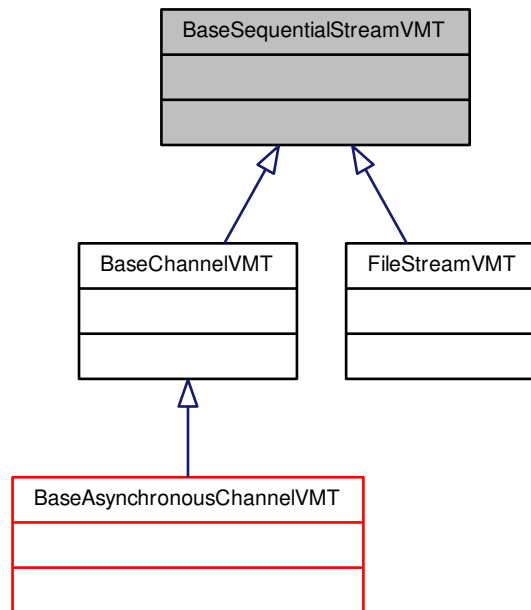
Virtual Methods Table.

8.11 BaseSequentialStreamVMT Struct Reference

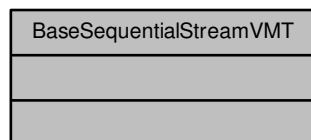
[BaseSequentialStream](#) virtual methods table.

```
#include <hal_streams.h>
```

Inheritance diagram for BaseSequentialStreamVMT:



Collaboration diagram for BaseSequentialStreamVMT:



8.11.1 Detailed Description

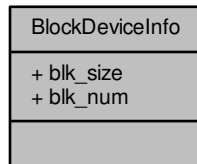
[BaseSequentialStream](#) virtual methods table.

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

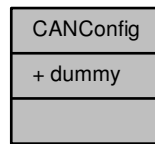
Total number of blocks.

8.13 CANConfig Struct Reference

Driver configuration structure.

```
#include <hal_can_lld.h>
```

Collaboration diagram for CANConfig:



8.13.1 Detailed Description

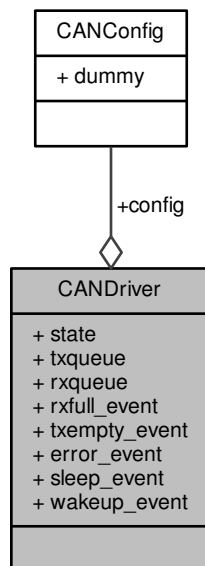
Driver configuration structure.

8.14 CANDriver Struct Reference

Structure representing an CAN driver.

```
#include <hal_can_llid.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

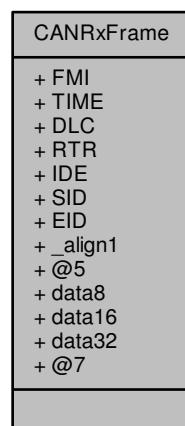
Exiting sleep state event.

8.15 CANRxFrame Struct Reference

CAN received frame.

```
#include <hal_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]`

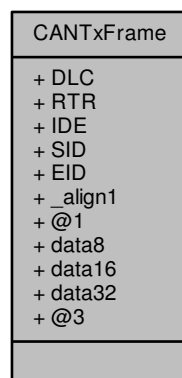
Frame data.

8.16 CANTxFrame Struct Reference

CAN transmission frame.

```
#include <hal_can_lld.h>
```

Collaboration diagram for CANTxFrame:



Data Fields

- uint8_t **DLC**:4
Data length.
- uint8_t **RTR**:1
Frame type.
- uint8_t **IDE**:1
Identifier type.
- uint32_t **SID**:11
Standard identifier.
- uint32_t **EID**:29
Extended identifier.
- uint8_t **data8** [8]
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]

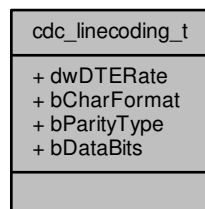
Frame data.

8.17 cdc_linecoding_t Struct Reference

Type of Line Coding structure.

```
#include <hal_usb_cdc.h>
```

Collaboration diagram for cdc_linecoding_t:



8.17.1 Detailed Description

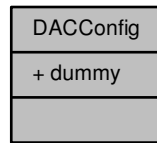
Type of Line Coding structure.

8.18 DACConfig Struct Reference

Driver configuration structure.

```
#include <hal_dac_llid.h>
```


Collaboration diagram for DACConfig:



8.18.1 Detailed Description

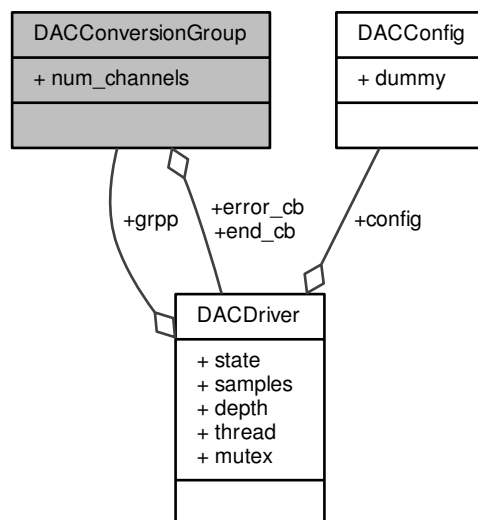
Driver configuration structure.

8.19 DACConversionGroup Struct Reference

DAC Conversion group structure.

```
#include <hal_dac_llid.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`

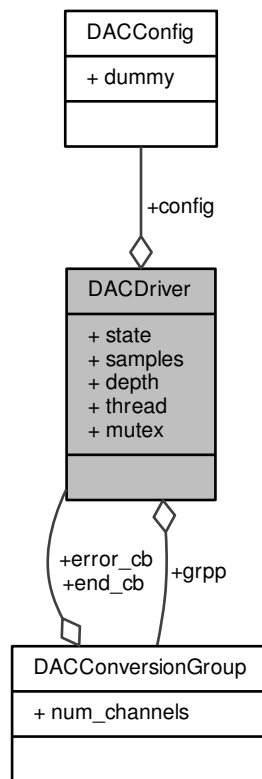
Error handling callback or NULL.

8.20 DACDriver Struct Reference

Structure representing a DAC driver.

```
#include <hal_dac_lld.h>
```

Collaboration diagram for DACDriver:



Data Fields

- `dacstate_t state`
Driver state.
- `const DACConversionGroup * grpp`
Conversion group.
- `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 `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`

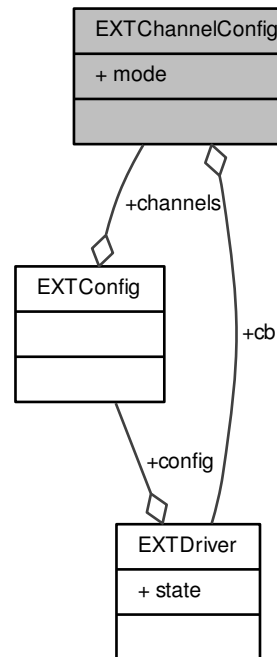
Mutex protecting the bus.

8.21 EXTChannelConfig Struct Reference

Channel configuration structure.

```
#include <hal_ext_llld.h>
```

Collaboration diagram for EXTChannelConfig:



Data Fields

- `uint32_t mode`
Channel mode.
- `extcallback_t cb`
Channel callback.

8.21.1 Detailed Description

Channel configuration structure.

8.21.2 Field Documentation

8.21.2.1 `uint32_t EXTChannelConfig::mode`

Channel mode.

8.21.2.2 `extcallback_t EXTChannelConfig::cb`

Channel callback.

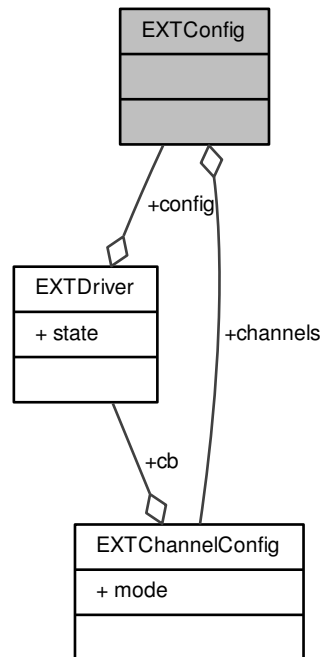
In the STM32 implementation a `NULL` callback pointer is valid and configures the channel as an event sources instead of an interrupt source.

8.22 EXTConfig Struct Reference

Driver configuration structure.

```
#include <hal_ext_lld.h>
```

Collaboration diagram for EXTConfig:



Data Fields

- [EXTChannelConfig channels \[EXT_MAX_CHANNELS\]](#)
Channel configurations.

8.22.1 Detailed Description

Driver configuration structure.

Note

It could be empty on some architectures.

8.22.2 Field Documentation

8.22.2.1 EXTChannelConfig EXTConfig::channels[EXT_MAX_CHANNELS]

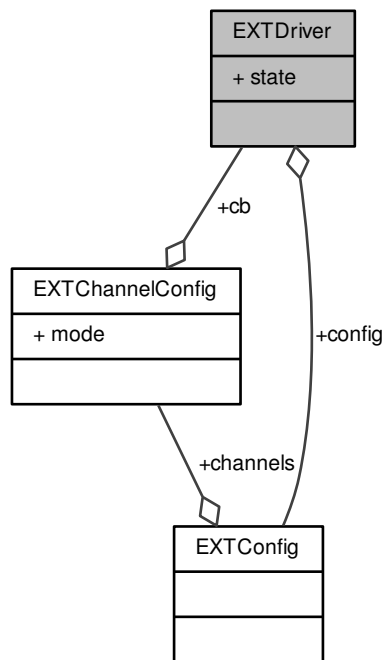
Channel configurations.

8.23 EXTDriver Struct Reference

Structure representing an EXT driver.

```
#include <hal_ext_lld.h>
```

Collaboration diagram for EXTDriver:



Data Fields

- [extstate_t state](#)
Driver state.
- `const EXTConfig * config`
Current configuration data.

8.23.1 Detailed Description

Structure representing an EXT driver.

8.23.2 Field Documentation

8.23.2.1 `extstate_t EXTDriver::state`

Driver state.

8.23.2.2 `const EXTConfig* EXTDriver::config`

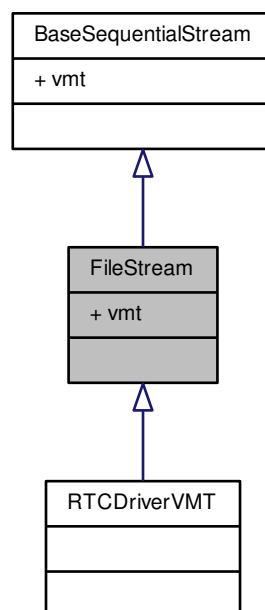
Current configuration data.

8.24 FileStream Struct Reference

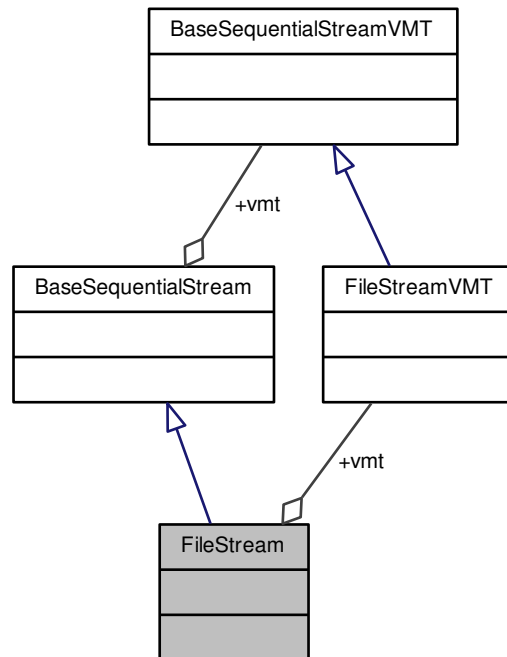
Base file stream class.

```
#include <hal_files.h>
```

Inheritance diagram for FileStream:



Collaboration diagram for FileStream:



Data Fields

- const struct [FileStreamVMT](#) * **vmt**

Virtual Methods Table.

8.24.1 Detailed Description

Base file stream class.

This class represents a generic file data stream.

8.24.2 Field Documentation

8.24.2.1 const struct [FileStreamVMT](#)* **FileStream::vmt**

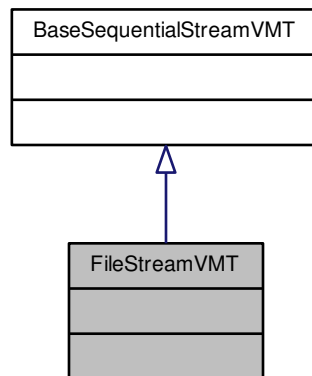
Virtual Methods Table.

8.25 FileStreamVMT Struct Reference

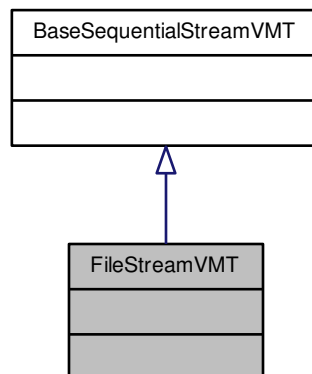
[FileStream](#) virtual methods table.

```
#include <hal_files.h>
```

Inheritance diagram for FileStreamVMT:



Collaboration diagram for FileStreamVMT:



8.25.1 Detailed Description

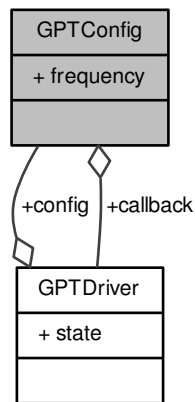
`FileStream` virtual methods table.

8.26 GPTConfig Struct Reference

Driver configuration structure.

```
#include <hal_gpt_lld.h>
```

Collaboration diagram for GPTConfig:



Data Fields

- [gptfreq_t frequency](#)
Timer clock in Hz.
- [gptcallback_t callback](#)
Timer callback pointer.

8.26.1 Detailed Description

Driver configuration structure.

Note

It could be empty on some architectures.

8.26.2 Field Documentation

8.26.2.1 `gptfreq_t GPTConfig::frequency`

Timer clock in Hz.

Note

The low level can use assertions in order to catch invalid frequency specifications.

8.26.2.2 `gptcallback_t GPTConfig::callback`

Timer callback pointer.

Note

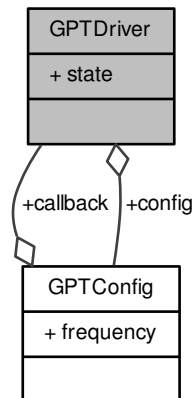
This callback is invoked on GPT counter events.

8.27 GPTDriver Struct Reference

Structure representing a GPT driver.

```
#include <hal_gpt_lld.h>
```

Collaboration diagram for GPTDriver:



Data Fields

- [gptstate_t state](#)
Driver state.
- `const GPTConfig * config`
Current configuration data.

8.27.1 Detailed Description

Structure representing a GPT driver.

8.27.2 Field Documentation

8.27.2.1 `gptstate_t GPTDriver::state`

Driver state.

8.27.2.2 `const GPTConfig* GPTDriver::config`

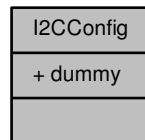
Current configuration data.

8.28 I2CConfig Struct Reference

Type of I2C driver configuration structure.

```
#include <hal_i2c_llid.h>
```

Collaboration diagram for I2CConfig:



8.28.1 Detailed Description

Type of I2C driver configuration structure.

Note

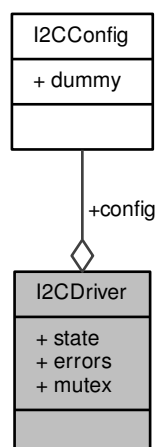
Implementations may extend this structure to contain more, architecture dependent, fields.

8.29 I2CDriver Struct Reference

Structure representing an I2C driver.

```
#include <hal_i2c_llid.h>
```

Collaboration diagram for I2CDriver:



Data Fields

- [i2cstate_t state](#)

Driver state.

- `const I2CConfig * config`

Current configuration data.

- [i2cflags_t errors](#)

Error flags.

8.29.1 Detailed Description

Structure representing an I2C driver.

8.29.2 Field Documentation

8.29.2.1 `i2cstate_t I2CDriver::state`

Driver state.

8.29.2.2 `const I2CConfig* I2CDriver::config`

Current configuration data.

8.29.2.3 `i2cflags_t I2CDriver::errors`

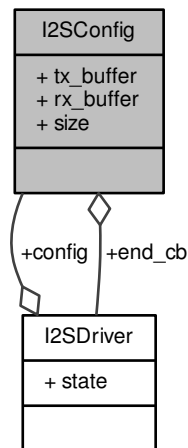
Error flags.

8.30 I2SConfig Struct Reference

Driver configuration structure.

```
#include <hal_i2s_lld.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.

8.30.1 Detailed Description

Driver configuration structure.

Note

It could be empty on some architectures.

8.30.2 Field Documentation

8.30.2.1 `const void* I2SConfig::tx_buffer`

Transmission buffer pointer.

Note

Can be `NULL` if TX is not required.

8.30.2.2 void* I2SConfig::rx_buffer

Receive buffer pointer.

Note

Can be `NULL` if RX is not required.

8.30.2.3 size_t I2SConfig::size

TX and RX buffers size as number of samples.

8.30.2.4 i2scallback_t I2SConfig::end_cb

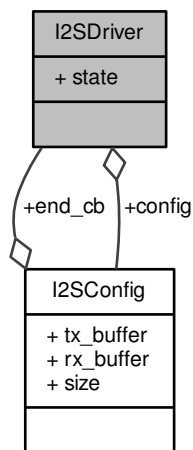
Callback function called during streaming.

8.31 I2SDriver Struct Reference

Structure representing an I2S driver.

```
#include <hal_i2s_lld.h>
```

Collaboration diagram for I2SDriver:



Data Fields

- [i2sstate_t state](#)
Driver state.
- [const I2SConfig * config](#)
Current configuration data.

8.31.1 Detailed Description

Structure representing an I2S driver.

8.31.2 Field Documentation

8.31.2.1 `i2sstate_t` `I2SDriver::state`

Driver state.

8.31.2.2 `const I2SConfig*` `I2SDriver::config`

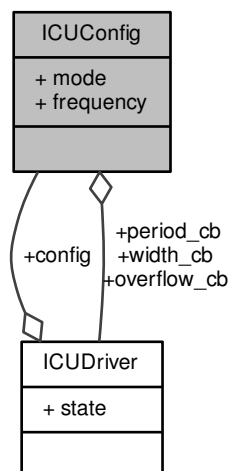
Current configuration data.

8.32 ICUConfig Struct Reference

Driver configuration structure.

```
#include <hal_icu_1ld.h>
```

Collaboration diagram for ICUConfig:



Data Fields

- [icumode_t](#) `mode`
Driver mode.
- [icufreq_t](#) `frequency`
Timer clock in Hz.
- [icucallback_t](#) `width_cb`
Callback for pulse width measurement.
- [icucallback_t](#) `period_cb`

Callback for cycle period measurement.

- `icucallback_t overflow_cb`

Callback for timer overflow.

8.32.1 Detailed Description

Driver configuration structure.

Note

It could be empty on some architectures.

8.32.2 Field Documentation

8.32.2.1 `icumode_t ICUConfig::mode`

Driver mode.

8.32.2.2 `icufreq_t ICUConfig::frequency`

Timer clock in Hz.

Note

The low level can use assertions in order to catch invalid frequency specifications.

8.32.2.3 `icucallback_t ICUConfig::width_cb`

Callback for pulse width measurement.

8.32.2.4 `icucallback_t ICUConfig::period_cb`

Callback for cycle period measurement.

8.32.2.5 `icucallback_t ICUConfig::overflow_cb`

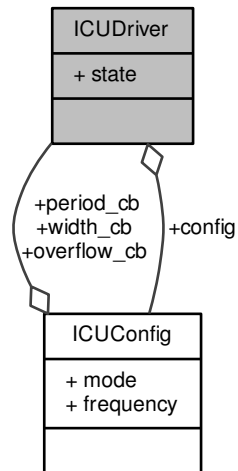
Callback for timer overflow.

8.33 ICUDriver Struct Reference

Structure representing an ICU driver.

```
#include <hal_icu_llld.h>
```

Collaboration diagram for ICUDriver:



Data Fields

- [icustate_t state](#)
Driver state.
- [const ICUConfig * config](#)
Current configuration data.

8.33.1 Detailed Description

Structure representing an ICU driver.

8.33.2 Field Documentation

8.33.2.1 `icustate_t ICUDriver::state`

Driver state.

8.33.2.2 `const ICUConfig* ICUDriver::config`

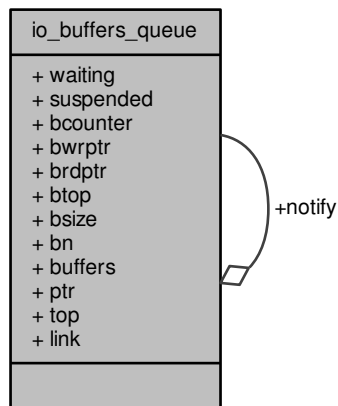
Current configuration data.

8.34 io_buffers_queue Struct Reference

Structure of a generic buffers queue.

```
#include <hal_buffers.h>
```

Collaboration diagram for io_buffers_queue:



Data Fields

- threads_queue_t [waiting](#)
Queue of waiting threads.
- bool [suspended](#)
Queue suspended state flag.
- volatile size_t [bcounter](#)
Active buffers counter.
- uint8_t * [bwrptr](#)
Buffer write pointer.
- uint8_t * [brdptr](#)
Buffer read pointer.
- uint8_t * [btop](#)
Pointer to the buffers boundary.
- size_t [bsize](#)
Size of buffers.
- size_t [bn](#)
Number of buffers.
- uint8_t * [buffers](#)
Queue of buffer objects.
- uint8_t * [ptr](#)
Pointer for R/W sequential access.
- uint8_t * [top](#)
Boundary for R/W sequential access.
- [bqnotify_t](#) [notify](#)
Data notification callback.
- void * [link](#)
Application defined field.

8.34.1 Detailed Description

Structure of a generic buffers queue.

8.34.2 Field Documentation

8.34.2.1 `threads_queue_t io_buffers_queue::waiting`

Queue of waiting threads.

8.34.2.2 `bool io_buffers_queue::suspended`

Queue suspended state flag.

8.34.2.3 `volatile size_t io_buffers_queue::bcounter`

Active buffers counter.

8.34.2.4 `uint8_t* io_buffers_queue::bwrptr`

Buffer write pointer.

8.34.2.5 `uint8_t* io_buffers_queue::brdptr`

Buffer read pointer.

8.34.2.6 `uint8_t* io_buffers_queue::btop`

Pointer to the buffers boundary.

8.34.2.7 `size_t io_buffers_queue::bsize`

Size of buffers.

Note

The buffer size must be not lower than `sizeof(size_t) + 2` because the first bytes are used to store the used size of the buffer.

8.34.2.8 `size_t io_buffers_queue::bn`

Number of buffers.

8.34.2.9 `uint8_t* io_buffers_queue::buffers`

Queue of buffer objects.

8.34.2.10 `uint8_t* io_buffers_queue::ptr`

Pointer for R/W sequential access.

Note

It is `NULL` if a new buffer must be fetched from the queue.

8.34.2.11 `uint8_t* io_buffers_queue::top`

Boundary for R/W sequential access.

8.34.2.12 `bqnotify_t io_buffers_queue::notify`

Data notification callback.

8.34.2.13 `void* io_buffers_queue::link`

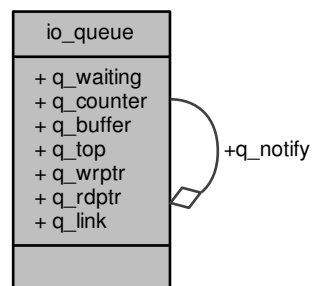
Application defined field.

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`
Queue of waiting threads.
- `volatile size_t q_counter`
Resources counter.
- `uint8_t * q_buffer`
Pointer to the queue buffer.

- `uint8_t * q_top`
Pointer to the first location after the buffer.
- `uint8_t * q_wrptr`
Write pointer.
- `uint8_t * q_rdptr`
Read pointer.
- `qnotify_t q_notify`
Data notification callback.
- `void * q_link`
Application defined field.

8.35.1 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`

Queue of waiting threads.

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_wrptr`

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

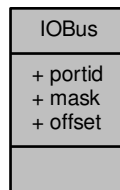
Application defined field.

8.36 IOBus Struct Reference

I/O bus descriptor.

```
#include <hal_pal.h>
```

Collaboration diagram for IOBus:



Data Fields

- [ioportid_t portid](#)
Port identifier.
- [ioportmask_t mask](#)
Bus mask aligned to port bit 0.
- [uint_fast8_t offset](#)
Offset, within the port, of the least significant bit of the bus.

8.36.1 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 ioportid_t IOBus::portid

Port identifier.

8.36.2.2 ioportmask_t IOBus::mask

Bus mask aligned to port bit 0.

Note

The bus mask implicitly define the bus width. A logic AND is performed on the bus data.

8.36.2.3 uint_fast8_t IOBus::offset

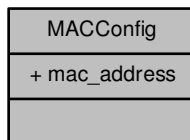
Offset, within the port, of the least significant bit of the bus.

8.37 MACConfig Struct Reference

Driver configuration structure.

```
#include <hal_mac_llld.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

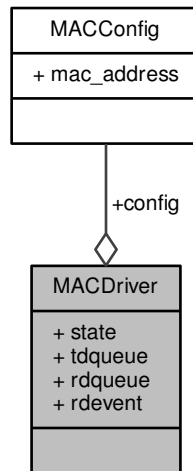
MAC address.

8.38 MACDriver Struct Reference

Structure representing a MAC driver.

```
#include <hal_mac_llld.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

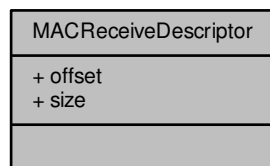
Receive event.

8.39 MACReceiveDescriptor Struct Reference

Structure representing a receive descriptor.

```
#include <hal_mac_llid.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`

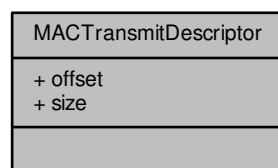
Available data size.

8.40 MACTransmitDescriptor Struct Reference

Structure representing a transmit descriptor.

```
#include <hal_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`

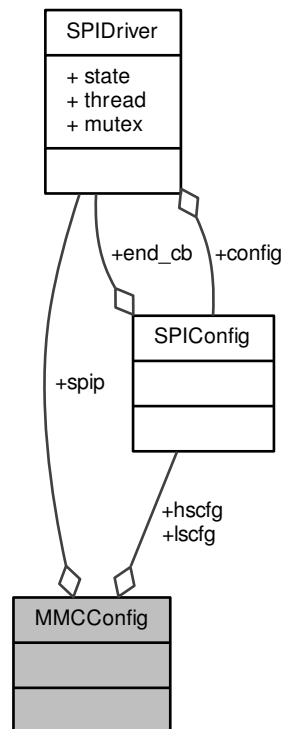
Available space size.

8.41 MMCConfig Struct Reference

MMC/SD over SPI driver configuration structure.

```
#include <hal_mmc_spi.h>
```

Collaboration diagram for MMCCConfig:



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* MMCCConfig::spip`

SPI driver associated to this MMC driver.

8.41.2.2 `const SPIConfig* MMCCConfig::lscfg`

SPI low speed configuration used during initialization.

8.41.2.3 `const SPIConfig* MMCCConfig::hscfg`

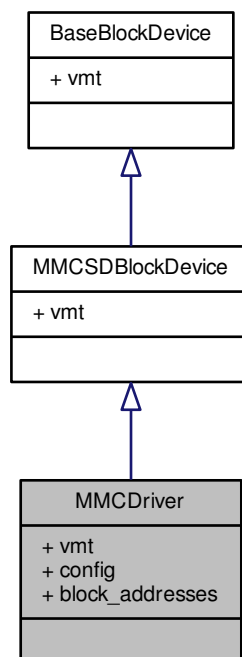
SPI high speed configuration used during transfers.

8.42 MMCDriver Struct Reference

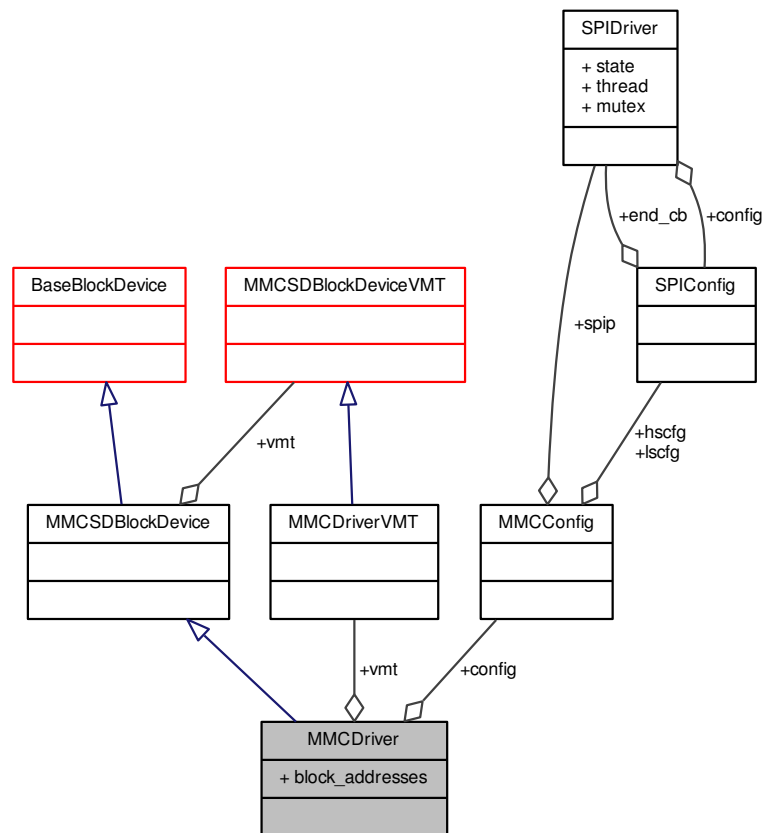
Structure representing a MMC/SD over SPI driver.

```
#include <hal_mmc_spi.h>
```

Inheritance diagram for MMCDriver:



Collaboration diagram for MMCDriver:



Data Fields

- const struct [MMCDriverVMT](#) * `vmt`
Virtual Methods Table.
- `_mmcsd_block_device_data` const [MMCCfg](#) * `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 `_mmc_sd_block_device_data` const `MMCCConfig*` `MMCDriver::config`

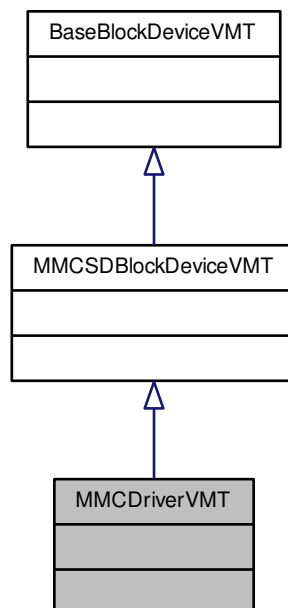
Current configuration data.

8.43 `MMCDriverVMT` Struct Reference

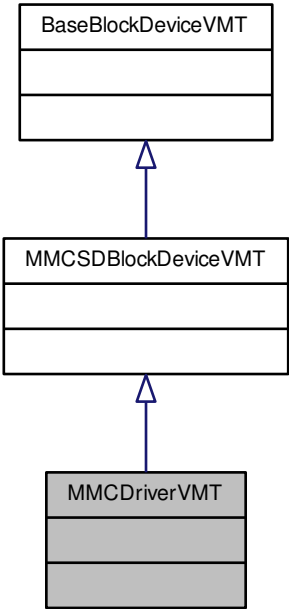
`MMCDriver` virtual methods table.

```
#include <hal_mmc_spi.h>
```

Inheritance diagram for `MMCDriverVMT`:



Collaboration diagram for MMCDriverVMT:



8.43.1 Detailed Description

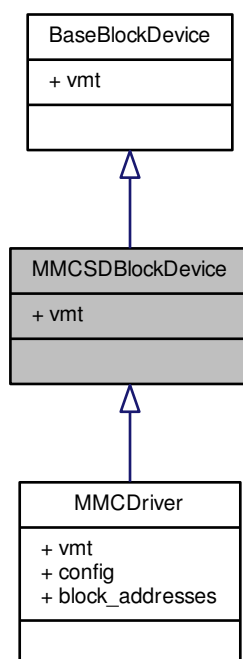
MMCDriver virtual methods table.

8.44 MMCSDBlockDevice Struct Reference

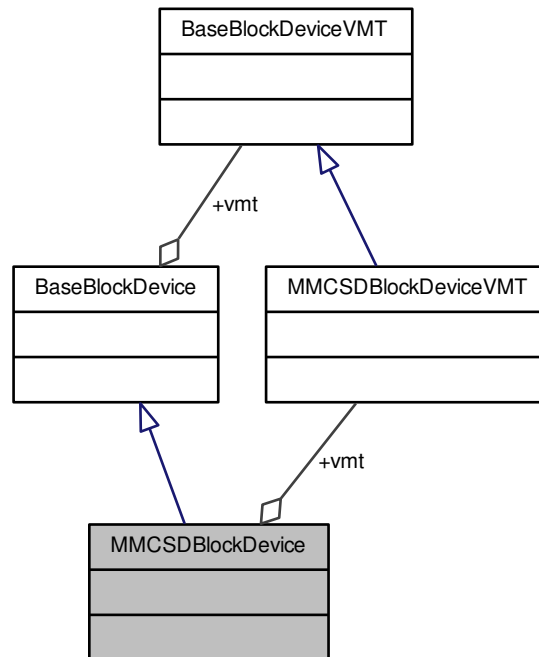
MCC/SD block device class.

```
#include <hal_mmc.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*

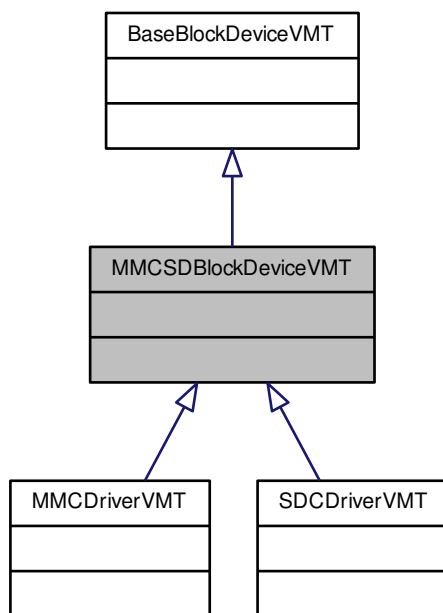
Virtual Methods Table.

8.45 MMCSDBlockDeviceVMT Struct Reference

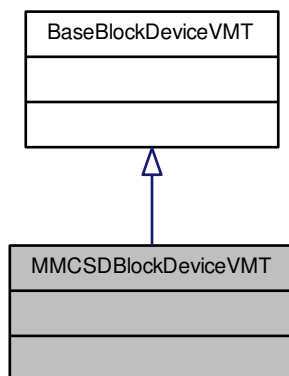
[MMCSDBlockDevice](#) virtual methods table.

```
#include <hal_mmcsd.h>
```

Inheritance diagram for MMCSDBlockDeviceVMT:



Collaboration diagram for MMCSDBlockDeviceVMT:



8.45.1 Detailed Description

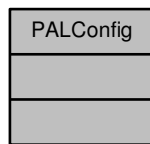
[MMCSDBlockDevice](#) virtual methods table.

8.46 PALConfig Struct Reference

Generic I/O ports static initializer.

```
#include <hal_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

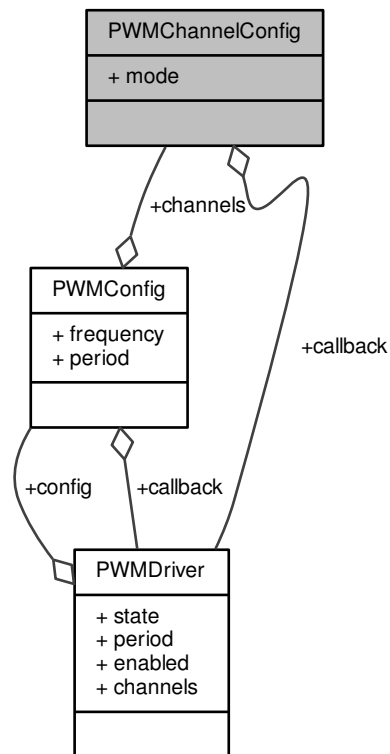
Implementations may extend this structure to contain more, architecture dependent, fields.

8.47 PWMChannelConfig Struct Reference

Type of a PWM driver channel configuration structure.

```
#include <hal_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

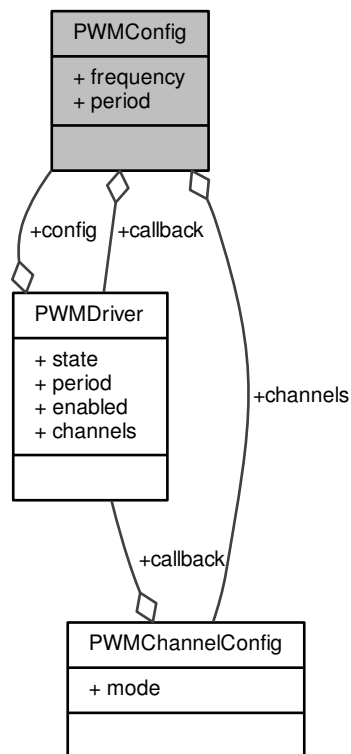
This callback is invoked on the channel compare event. If set to `NULL` then the callback is disabled.

8.48 PWMConfig Struct Reference

Type of a PWM driver configuration structure.

```
#include <hal_pwm_lld.h>
```

Collaboration diagram for PWMConfig:



Data Fields

- `uint32_t frequency`
Timer clock in Hz.
- `pwmcnt_t period`
PWM period in ticks.
- `pwmcallback_t callback`
Periodic callback pointer.
- `PWMChannelConfig channels [PWM_CHANNELS]`
Channels configurations.

8.48.1 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]`

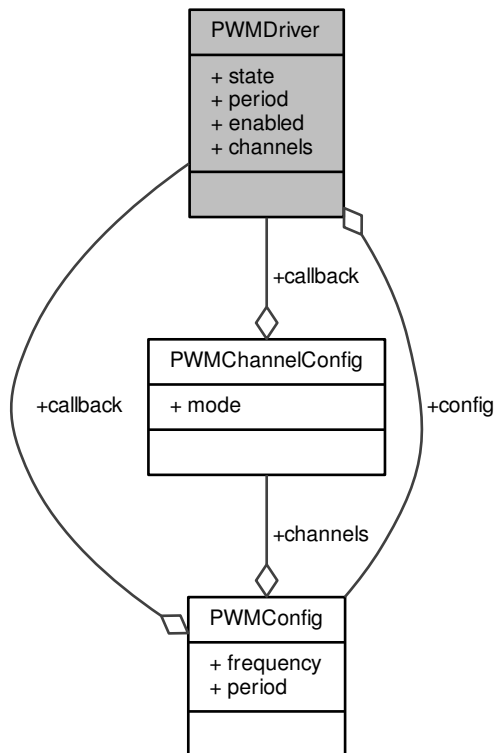
Channels configurations.

8.49 PWMDriver Struct Reference

Structure representing a PWM driver.

```
#include <hal_pwm_lld.h>
```


Collaboration diagram for PWMDriver:



Data Fields

- [pwmstate_t state](#)
Driver state.
- `const PWMConfig * config`
Current driver configuration data.
- [pwmcnt_t period](#)
Current PWM period in ticks.
- [pwmchnmsk_t enabled](#)
Mask of the enabled channels.
- [pwmchannel_t channels](#)
Number of channels in this instance.

8.49.1 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`

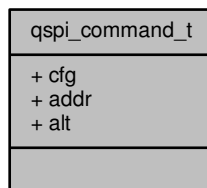
Number of channels in this instance.

8.50 `qspi_command_t` Struct Reference

Type of a QSPI command descriptor.

```
#include <hal_qspi.h>
```

Collaboration diagram for `qspi_command_t`:



8.50.1 Detailed Description

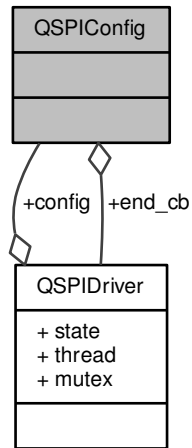
Type of a QSPI command descriptor.

8.51 `QSPIConfig` Struct Reference

Driver configuration structure.

```
#include <hal_qspi_lld.h>
```

Collaboration diagram for QSPIDriver:



Data Fields

- [qspicallback_t end_cb](#)

Operation complete callback or NULL.

8.51.1 Detailed Description

Driver configuration structure.

8.51.2 Field Documentation

8.51.2.1 qspicallback_t QSPIDriverConfig::end_cb

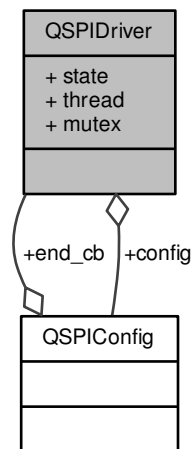
Operation complete callback or NULL.

8.52 QSPIDriver Struct Reference

Structure representing an QSPI driver.

```
#include <hal_qspi_llid.h>
```

Collaboration diagram for QSPIDriver:



Data Fields

- [qspistate_t](#) `state`
Driver state.
- `const` [QSPICongig](#) * `config`
Current configuration data.
- `thread_reference_t` [thread](#)
Waiting thread.
- `mutex_t` [mutex](#)
Mutex protecting the peripheral.

8.52.1 Detailed Description

Structure representing an QSPI driver.

8.52.2 Field Documentation

8.52.2.1 `qspistate_t` `QSPIDriver::state`

Driver state.

8.52.2.2 `const` `QSPICongig*` `QSPIDriver::config`

Current configuration data.

8.52.2.3 `thread_reference_t` `QSPIDriver::thread`

Waiting thread.

8.52.2.4 mutex_t QSPIDriver::mutex

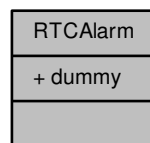
Mutex protecting the peripheral.

8.53 RTCArm Struct Reference

Type of a structure representing an RTC alarm time stamp.

```
#include <hal_rtc_lld.h>
```

Collaboration diagram for RTCArm:



8.53.1 Detailed Description

Type of a structure representing an RTC alarm time stamp.

8.54 RTCDatetime Struct Reference

Type of a structure representing an RTC date/time stamp.

```
#include <hal_rtc.h>
```

Collaboration diagram for RTCDatetime:



Data Fields

- uint32_t [year](#): 8
Years since 1980.
- uint32_t [month](#): 4
Months 1..12.
- uint32_t [dstflag](#): 1
DST correction flag.
- uint32_t [dayofweek](#): 3
Day of week 1..7.
- uint32_t [day](#): 5
Day of the month 1..31.
- uint32_t [millisecond](#): 27
Milliseconds since midnight.

8.54.1 Detailed Description

Type of a structure representing an RTC date/time stamp.

8.54.2 Field Documentation

8.54.2.1 uint32_t RTCDateTime::year

Years since 1980.

8.54.2.2 uint32_t RTCDateTime::month

Months 1..12.

8.54.2.3 uint32_t RTCDateTime::dstflag

DST correction flag.

8.54.2.4 uint32_t RTCDateTime::dayofweek

Day of week 1..7.

8.54.2.5 uint32_t RTCDateTime::day

Day of the month 1..31.

8.54.2.6 uint32_t RTCDateTime::millisecond

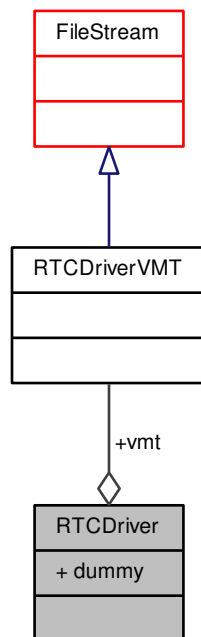
Milliseconds since midnight.

8.55 RTCDriver Struct Reference

Structure representing an RTC driver.

```
#include <hal_rtc_lld.h>
```

Collaboration diagram for RTCDriver:



Data Fields

- const struct [RTCDriverVMT](#) * [vmt](#)
Virtual Methods Table.

8.55.1 Detailed Description

Structure representing an RTC driver.

8.55.2 Field Documentation

8.55.2.1 const struct [RTCDriverVMT](#)* [RTCDriver::vmt](#)

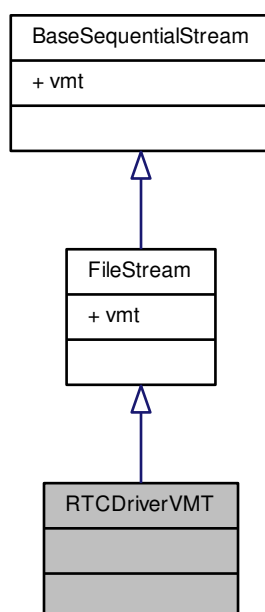
Virtual Methods Table.

8.56 RTCDriverVMT Struct Reference

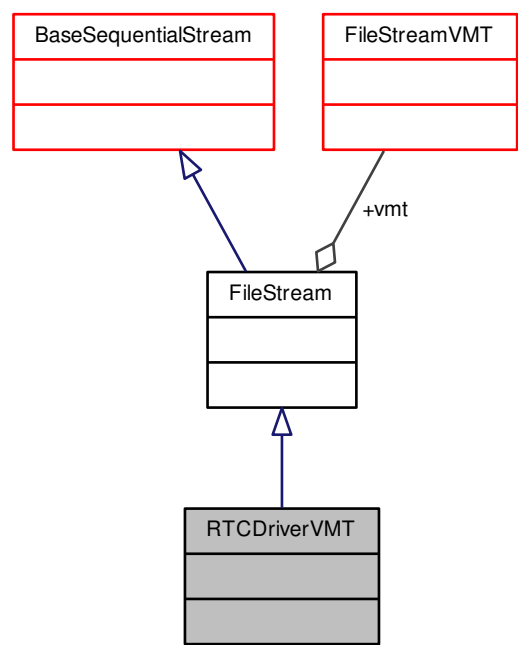
[RTCDriver](#) virtual methods table.

```
#include <hal_rtc_lld.h>
```

Inheritance diagram for RTCDriverVMT:



Collaboration diagram for RTCDriverVMT:



Additional Inherited Members

8.56.1 Detailed Description

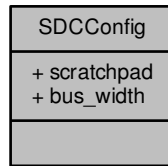
`RTCDriver` virtual methods table.

8.57 SDCConfig Struct Reference

Driver configuration structure.

```
#include <hal_sdc_lld.h>
```

Collaboration diagram for SDCCConfig:



Data Fields

- `uint8_t * scratchpad`
Working area for memory consuming operations.
- `sdcbusmode_t bus_width`
Bus width.

8.57.1 Detailed Description

Driver configuration structure.

Note

It could be empty on some architectures.

8.57.2 Field Documentation

8.57.2.1 `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.

8.57.2.2 `sdcbusmode_t SDCCConfig::bus_width`

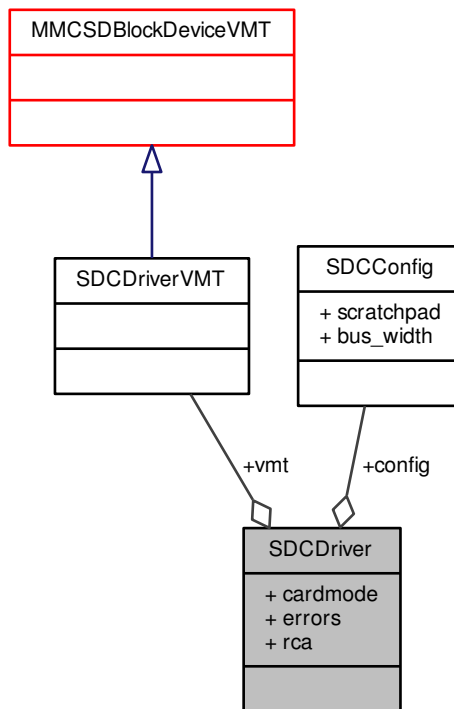
Bus width.

8.58 SDCCDriver Struct Reference

Structure representing an SDC driver.

```
#include <hal_sdc_llid.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.
- `sdclflags_t errors`
Errors flags.
- `uint32_t rca`
Card RCA.

8.58.1 Detailed Description

Structure representing an SDC driver.

8.58.2 Field Documentation

8.58.2.1 `const struct SDCDriverVMT * SDCDriver::vmt`

Virtual Methods Table.

8.58.2.2 `_mmc_sd_block_device_data` const `SDCConfig*` `SDCDriver::config`

Current configuration data.

8.58.2.3 `sdcmode_t` `SDCDriver::cardmode`

Various flags regarding the mounted card.

8.58.2.4 `sdcflags_t` `SDCDriver::errors`

Errors flags.

8.58.2.5 `uint32_t` `SDCDriver::rca`

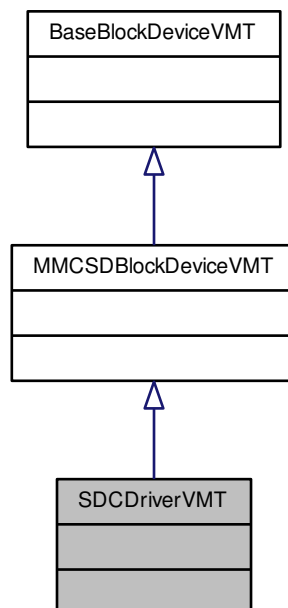
Card RCA.

8.59 SDCDriverVMT Struct Reference

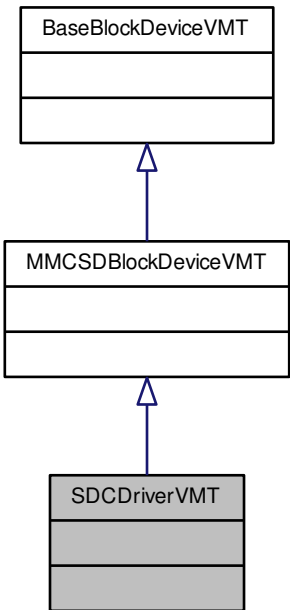
`SDCDriver` virtual methods table.

```
#include <hal_sdc_lld.h>
```

Inheritance diagram for `SDCDriverVMT`:



Collaboration diagram for SDCDriverVMT:



8.59.1 Detailed Description

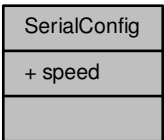
[SDCDriver](#) virtual methods table.

8.60 SerialConfig Struct Reference

PLATFORM Serial Driver configuration structure.

```
#include <hal_serial_llid.h>
```

Collaboration diagram for SerialConfig:



Data Fields

- uint32_t [speed](#)
Bit rate.

8.60.1 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.

8.60.2 Field Documentation

8.60.2.1 uint32_t SerialConfig::speed

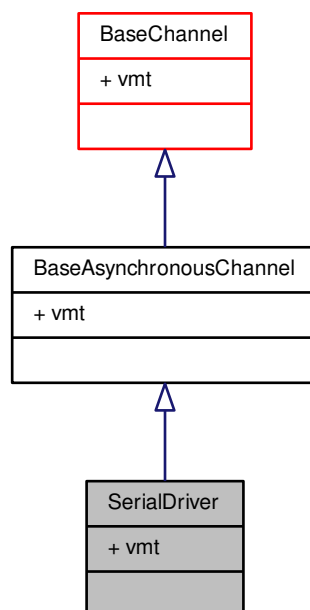
Bit rate.

8.61 SerialDriver Struct Reference

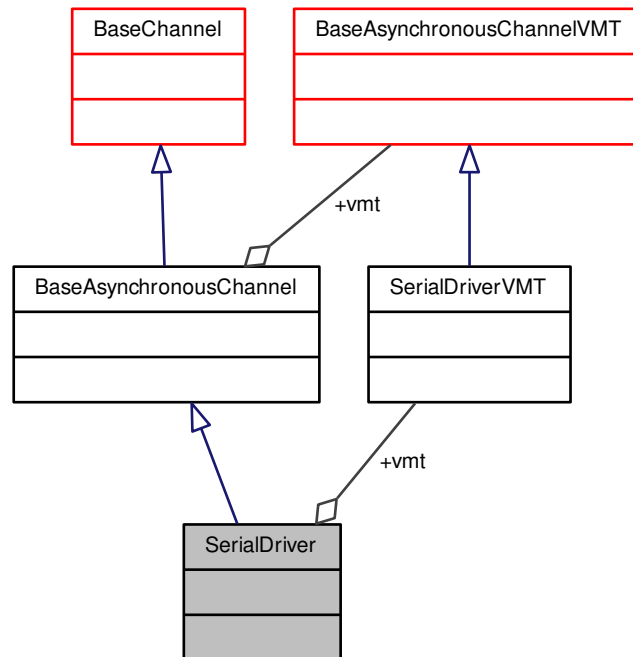
Full duplex serial driver class.

```
#include <hal_serial.h>
```

Inheritance diagram for SerialDriver:



Collaboration diagram for SerialDriver:



Data Fields

- const struct [SerialDriverVMT](#) * **vmt**

Virtual Methods Table.

8.61.1 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 [SerialDriverVMT](#)* **SerialDriver::vmt**

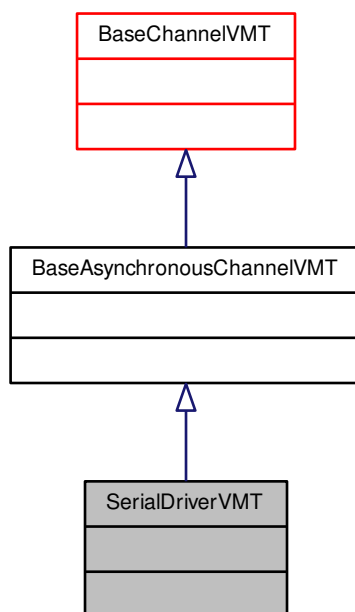
Virtual Methods Table.

8.62 SerialDriverVMT Struct Reference

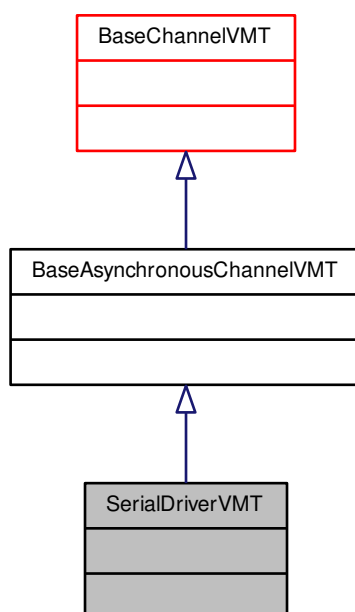
[SerialDriver](#) virtual methods table.

```
#include <hal_serial.h>
```

Inheritance diagram for SerialDriverVMT:



Collaboration diagram for SerialDriverVMT:



8.62.1 Detailed Description

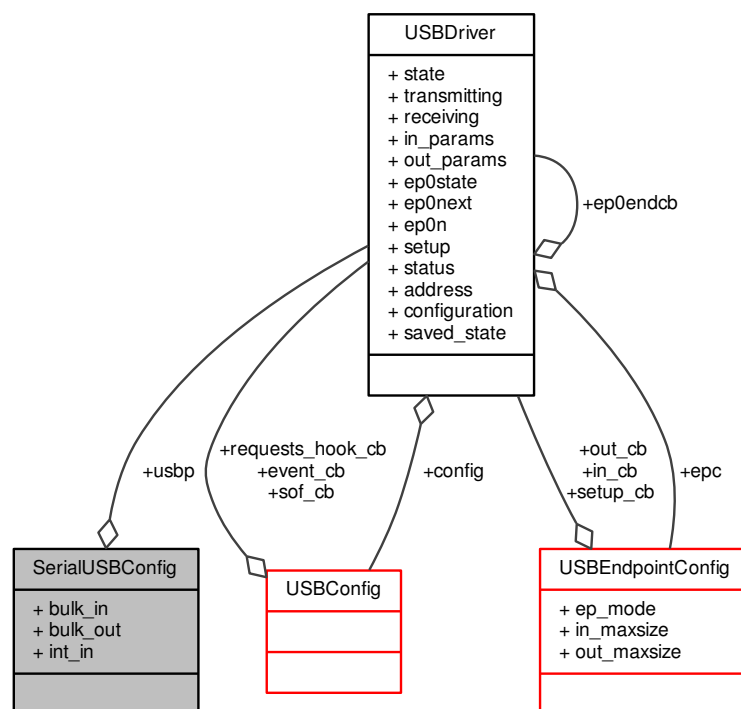
[SerialDriver](#) virtual methods table.

8.63 SerialUSBConfig Struct Reference

Serial over USB Driver configuration structure.

```
#include <hal_serial_usb.h>
```

Collaboration diagram for SerialUSBConfig:



Data Fields

- [USBDriver](#) * [usbp](#)
USB driver to use.
- [usbep_t](#) [bulk_in](#)
Bulk IN endpoint used for outgoing data transfer.
- [usbep_t](#) [bulk_out](#)
Bulk OUT endpoint used for incoming data transfer.
- [usbep_t](#) [int_in](#)
Interrupt IN endpoint used for notifications.

8.63.1 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.

8.63.2 Field Documentation

8.63.2.1 USBDriver* SerialUSBConfig::usbp

USB driver to use.

8.63.2.2 usbep_t SerialUSBConfig::bulk_in

Bulk IN endpoint used for outgoing data transfer.

8.63.2.3 usbep_t SerialUSBConfig::bulk_out

Bulk OUT endpoint used for incoming data transfer.

8.63.2.4 usbep_t SerialUSBConfig::int_in

Interrupt IN endpoint used for notifications.

Note

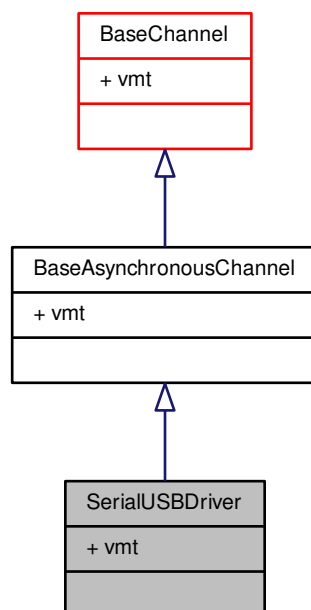
If set to zero then the INT endpoint is assumed to be not present, USB descriptors must be changed accordingly.

8.64 SerialUSBDriver Struct Reference

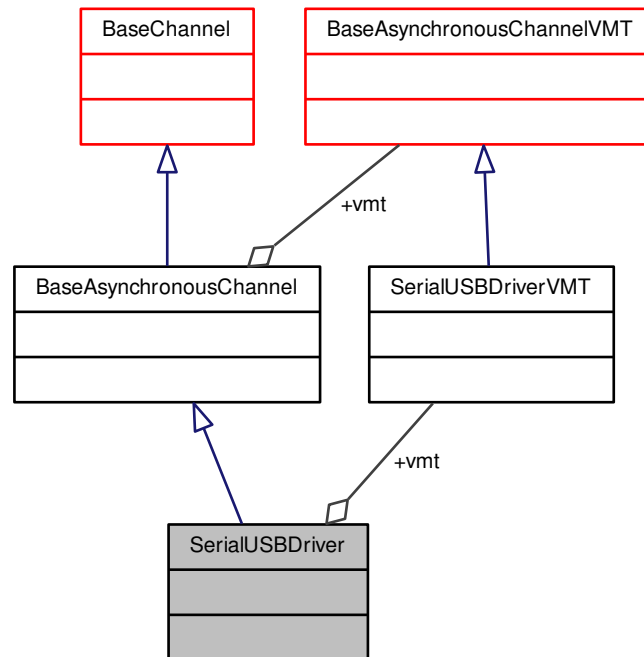
Full duplex serial driver class.

```
#include <hal_serial_usb.h>
```

Inheritance diagram for SerialUSBDriver:



Collaboration diagram for SerialUSBDriver:



Data Fields

- const struct [SerialUSBDriverVMT](#) * **vmt**

Virtual Methods Table.

8.64.1 Detailed Description

Full duplex serial driver class.

This class extends [BaseAsynchronousChannel](#) by adding physical I/O queues.

8.64.2 Field Documentation

8.64.2.1 const struct [SerialUSBDriverVMT](#)* [SerialUSBDriver::vmt](#)

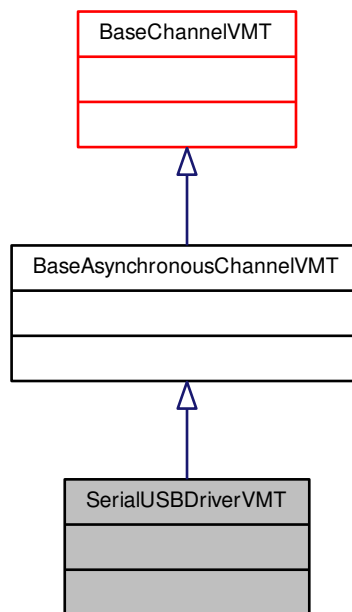
Virtual Methods Table.

8.65 SerialUSBDriverVMT Struct Reference

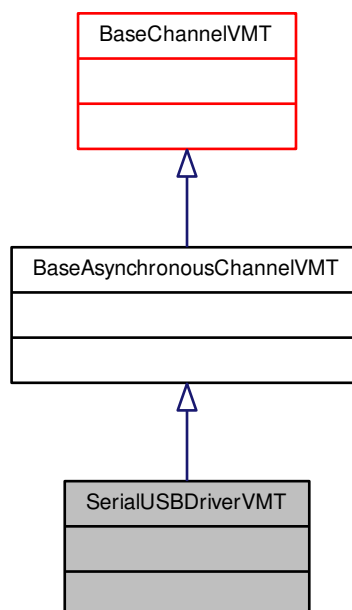
[SerialDriver](#) virtual methods table.

```
#include <hal_serial_usb.h>
```

Inheritance diagram for SerialUSBDriverVMT:



Collaboration diagram for SerialUSBDriverVMT:



8.65.1 Detailed Description

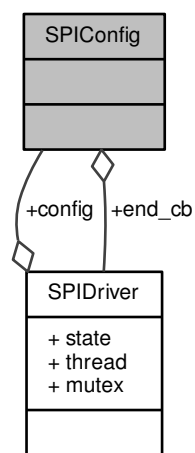
[SerialDriver](#) virtual methods table.

8.66 SPIConfig Struct Reference

Driver configuration structure.

```
#include <hal_spi_lld.h>
```

Collaboration diagram for SPIConfig:



Data Fields

- [spicallback_t end_cb](#)

Operation complete callback or `NULL`.

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 spicallback_t SPIConfig::end_cb

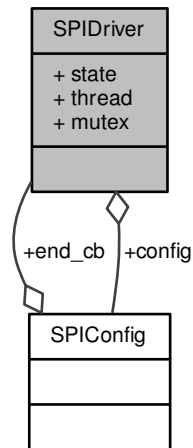
Operation complete callback or `NULL`.

8.67 SPIDriver Struct Reference

Structure representing an SPI driver.

```
#include <hal_spi_lld.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.

8.67.1 Detailed Description

Structure representing an SPI driver.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.

8.67.2 Field Documentation

8.67.2.1 spistate_t SPIDriver::state

Driver state.

8.67.2.2 `const SPIDriver::config`

Current configuration data.

8.67.2.3 `thread_reference_t SPIDriver::thread`

Waiting thread.

8.67.2.4 `mutex_t SPIDriver::mutex`

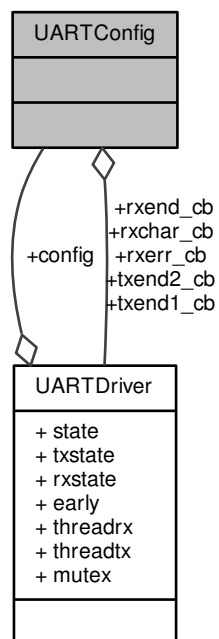
Mutex protecting the peripheral.

8.68 UARTConfig Struct Reference

Driver configuration structure.

```
#include <hal_uart_llid.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.

8.68.1 Detailed Description

Driver configuration structure.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.

8.68.2 Field Documentation

8.68.2.1 `uartcb_t UARTConfig::txend1_cb`

End of transmission buffer callback.

8.68.2.2 `uartcb_t UARTConfig::txend2_cb`

Physical end of transmission callback.

8.68.2.3 `uartcb_t UARTConfig::rxend_cb`

Receive buffer filled callback.

8.68.2.4 `uartccb_t UARTConfig::rxchar_cb`

Character received while out if the `UART_RECEIVE` state.

8.68.2.5 `uartecb_t UARTConfig::rxerr_cb`

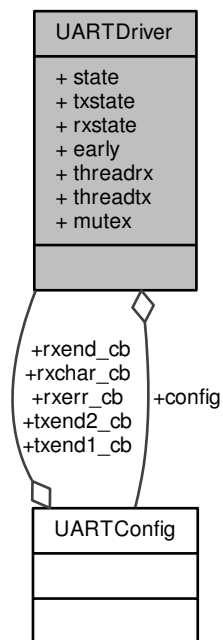
Receive error callback.

8.69 UARTDriver Struct Reference

Structure representing an UART driver.

```
#include <hal_uart_lld.h>
```

Collaboration diagram for UARTDriver:



Data Fields

- `uartstate_t` `state`
Driver state.
- `uarttxstate_t` `txstate`
Transmitter state.
- `uartrxstate_t` `rxstate`
Receiver state.
- `const UARTConfig *` `config`
Current configuration data.
- `bool` `early`
Synchronization flag for transmit operations.
- `thread_reference_t` `threadrx`
Waiting thread on RX.
- `thread_reference_t` `threadtx`
Waiting thread on TX.
- `mutex_t` `mutex`
Mutex protecting the peripheral.

8.69.1 Detailed Description

Structure representing an UART driver.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.

8.69.2 Field Documentation**8.69.2.1 uartstate_t UARTDriver::state**

Driver state.

8.69.2.2 uarttxstate_t UARTDriver::txstate

Transmitter state.

8.69.2.3 uartrxstate_t UARTDriver::rxstate

Receiver state.

8.69.2.4 const UARTConfig* UARTDriver::config

Current configuration data.

8.69.2.5 bool UARTDriver::early

Synchronization flag for transmit operations.

8.69.2.6 thread_reference_t UARTDriver::threadrx

Waiting thread on RX.

8.69.2.7 thread_reference_t UARTDriver::threadtx

Waiting thread on TX.

8.69.2.8 mutex_t UARTDriver::mutex

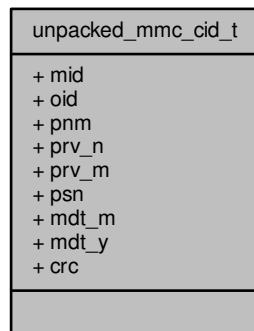
Mutex protecting the peripheral.

8.70 unpacked_mmc_cid_t Struct Reference

Unpacked CID register from MMC.

```
#include <hal_mmcsd.h>
```

Collaboration diagram for `unpacked_mmc_cid_t`:



8.70.1 Detailed Description

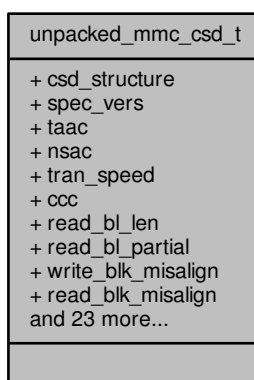
Unpacked CID register from MMC.

8.71 `unpacked_mmc_csd_t` Struct Reference

Unpacked CSD register from MMC.

```
#include <hal_mmc_csd.h>
```

Collaboration diagram for `unpacked_mmc_csd_t`:



8.71.1 Detailed Description

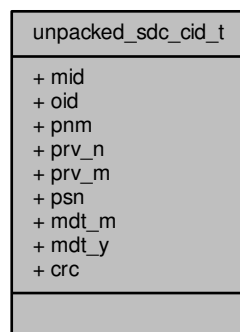
Unpacked CSD register from MMC.

8.72 unpacked_sdc_cid_t Struct Reference

Unpacked CID register from SDC.

```
#include <hal_mmcscsd.h>
```

Collaboration diagram for unpacked_sdc_cid_t:



8.72.1 Detailed Description

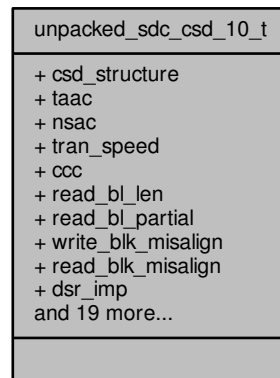
Unpacked CID register from SDC.

8.73 unpacked_sdc_csd_10_t Struct Reference

Unpacked CSD v1.0 register from SDC.

```
#include <hal_mmcscsd.h>
```

Collaboration diagram for `unpacked_sdc_csd_10_t`:



8.73.1 Detailed Description

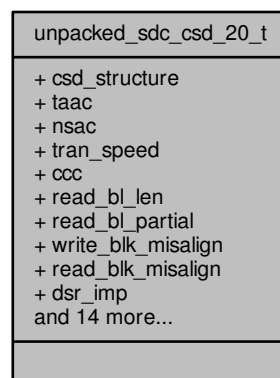
Unpacked CSD v1.0 register from SDC.

8.74 `unpacked_sdc_csd_20_t` Struct Reference

Unpacked CSD v2.0 register from SDC.

```
#include <hal_mmcscd.h>
```

Collaboration diagram for `unpacked_sdc_csd_20_t`:



8.74.1 Detailed Description

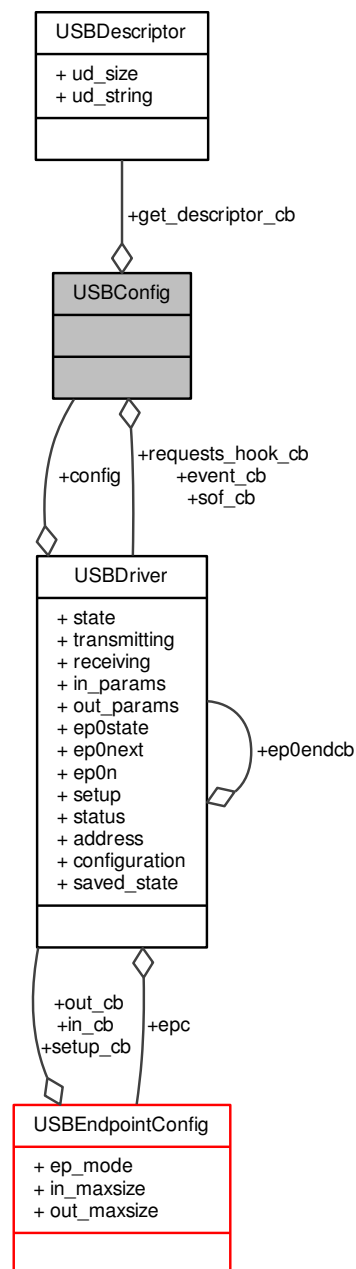
Unpacked CSD v2.0 register from SDC.

8.75 USBConfig Struct Reference

Type of an USB driver configuration structure.

```
#include <hal_usb_lld.h>
```

Collaboration diagram for USBConfig:



Data Fields

- [usbeventcb_t event_cb](#)
USB events callback.
- [usbgetdescriptor_t get_descriptor_cb](#)
Device GET_DESCRIPTOR request callback.
- [usbreqhandler_t requests_hook_cb](#)

Requests hook callback.

- [usbcallback_t sof_cb](#)

Start Of Frame callback.

8.75.1 Detailed Description

Type of an USB driver configuration structure.

8.75.2 Field Documentation

8.75.2.1 `usbeventcb_t USBConfig::event_cb`

USB events callback.

This callback is invoked when an USB driver event is registered.

8.75.2.2 `usbgetdescriptor_t USBConfig::get_descriptor_cb`

Device GET_DESCRIPTOR request callback.

Note

This callback is mandatory and cannot be set to `NULL`.

8.75.2.3 `usbreqhandler_t USBConfig::requests_hook_cb`

Requests hook callback.

This hook allows to be notified of standard requests or to handle non standard requests.

8.75.2.4 `usbcallback_t USBConfig::sof_cb`

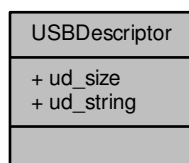
Start Of Frame callback.

8.76 USBDescriptor Struct Reference

Type of an USB descriptor.

```
#include <hal_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.

8.76.1 Detailed Description

Type of an USB descriptor.

8.76.2 Field Documentation

8.76.2.1 `size_t USBDescriptor::ud_size`

Descriptor size in unicode characters.

8.76.2.2 `const uint8_t* USBDescriptor::ud_string`

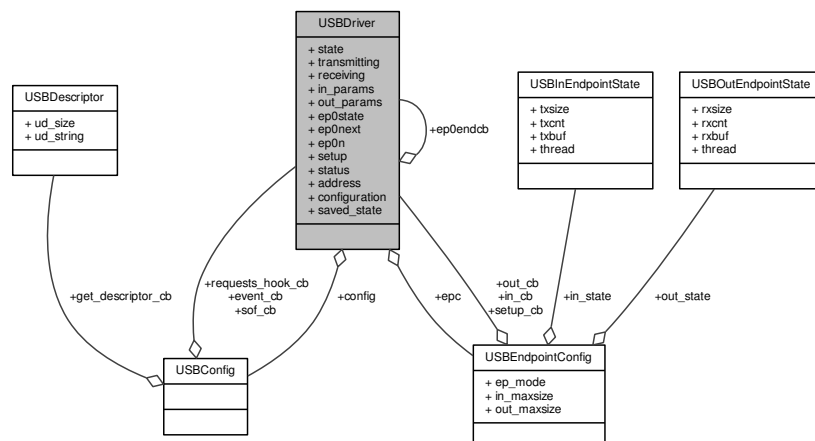
Pointer to the descriptor.

8.77 USBDriver Struct Reference

Structure representing an USB driver.

```
#include <hal_usb_ll.h>
```

Collaboration diagram for USBDriver:



Data Fields

- `usbstate_t state`
Driver state.
- `const USBConfig * config`

- Current configuration data.*
- `uint16_t transmitting`
 - Bit map of the transmitting IN endpoints.*
- `uint16_t receiving`
 - Bit map of the receiving OUT endpoints.*
- `const USBEndpointConfig * epc [USB_MAX_ENDPOINTS+1]`
 - Active endpoints configurations.*
- `void * in_params [USB_MAX_ENDPOINTS]`
 - Fields available to user, it can be used to associate an application-defined handler to an IN endpoint.*
- `void * out_params [USB_MAX_ENDPOINTS]`
 - Fields available to user, it can be used to associate an application-defined handler to an OUT endpoint.*
- `usbep0state_t ep0state`
 - Endpoint 0 state.*
- `uint8_t * ep0next`
 - Next position in the buffer to be transferred through endpoint 0.*
- `size_t ep0n`
 - Number of bytes yet to be transferred through endpoint 0.*
- `usbcallback_t ep0endcb`
 - Endpoint 0 end transaction callback.*
- `uint8_t setup [8]`
 - Setup packet buffer.*
- `uint16_t status`
 - Current USB device status.*
- `uint8_t address`
 - Assigned USB address.*
- `uint8_t configuration`
 - Current USB device configuration.*
- `usbstate_t saved_state`
 - State of the driver when a suspend happened.*

8.77.1 Detailed Description

Structure representing an USB driver.

8.77.2 Field Documentation

8.77.2.1 `usbstate_t USBDriver::state`

Driver state.

8.77.2.2 `const USBConfig* USBDriver::config`

Current configuration data.

8.77.2.3 `uint16_t USBDriver::transmitting`

Bit map of the transmitting IN endpoints.

8.77.2.4 `uint16_t USBDriver::receiving`

Bit map of the receiving OUT endpoints.

8.77.2.5 const USBEndpointConfig* USBDriver::epc[USB_MAX_ENDPOINTS+1]

Active endpoints configurations.

8.77.2.6 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.

8.77.2.7 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.

8.77.2.8 usbep0state_t USBDriver::ep0state

Endpoint 0 state.

8.77.2.9 uint8_t* USBDriver::ep0next

Next position in the buffer to be transferred through endpoint 0.

8.77.2.10 size_t USBDriver::ep0n

Number of bytes yet to be transferred through endpoint 0.

8.77.2.11 usbcallback_t USBDriver::ep0endcb

Endpoint 0 end transaction callback.

8.77.2.12 uint8_t USBDriver::setup[8]

Setup packet buffer.

8.77.2.13 uint16_t USBDriver::status

Current USB device status.

8.77.2.14 uint8_t USBDriver::address

Assigned USB address.

8.77.2.15 uint8_t USBDriver::configuration

Current USB device configuration.

8.77.2.16 usbstate_t USBDriver::saved_state

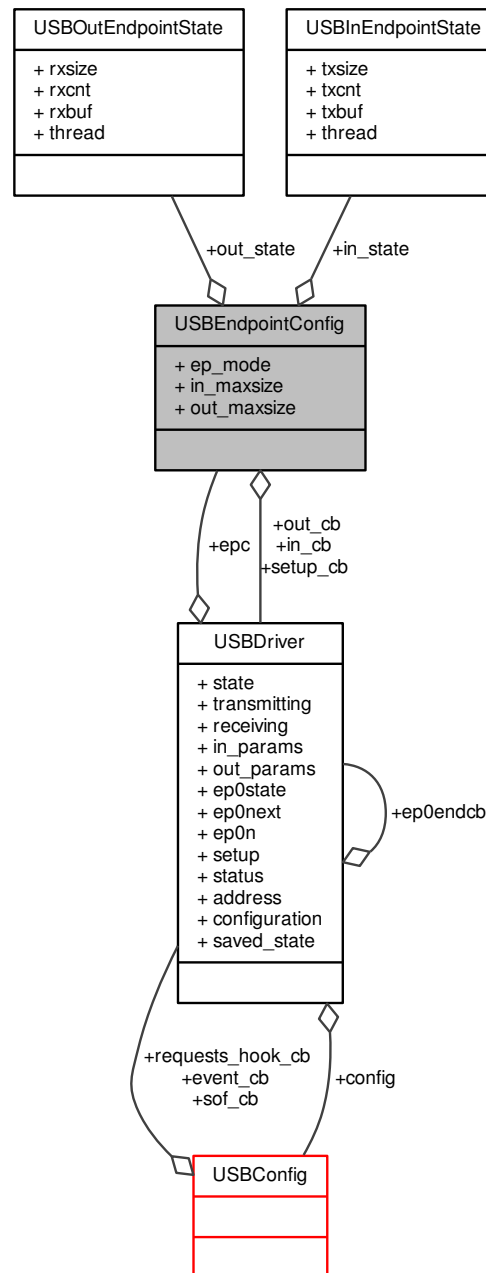
State of the driver when a suspend happened.

8.78 USBEndpointConfig Struct Reference

Type of an USB endpoint configuration structure.

```
#include <hal_usb_lld.h>
```

Collaboration diagram for USBEndpointConfig:



Data Fields

- `uint32_t ep_mode`
Type and mode of the endpoint.
- `usbepcallback_t setup_cb`
Setup packet notification callback.
- `usbepcallback_t in_cb`

- IN endpoint notification callback.*
- [usbepcallback_t out_cb](#)
OUT endpoint notification callback.
- [uint16_t in_maxsize](#)
IN endpoint maximum packet size.
- [uint16_t out_maxsize](#)
OUT endpoint maximum packet size.
- [USBInEndpointState * in_state](#)
USBEndpointState associated to the IN endpoint.
- [USBOutEndpointState * out_state](#)
USBEndpointState associated to the OUT endpoint.

8.78.1 Detailed Description

Type of an USB endpoint configuration structure.

Note

Platform specific restrictions may apply to endpoints.

8.78.2 Field Documentation

8.78.2.1 [uint32_t USBEndpointConfig::ep_mode](#)

Type and mode of the endpoint.

8.78.2.2 [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.

8.78.2.3 [usbepcallback_t USBEndpointConfig::in_cb](#)

IN endpoint notification callback.

This field must be set to `NULL` if the IN endpoint is not used.

8.78.2.4 [usbepcallback_t USBEndpointConfig::out_cb](#)

OUT endpoint notification callback.

This field must be set to `NULL` if the OUT endpoint is not used.

8.78.2.5 `uint16_t USBEndpointConfig::in_maxsize`

IN endpoint maximum packet size.

This field must be set to zero if the IN endpoint is not used.

8.78.2.6 `uint16_t USBEndpointConfig::out_maxsize`

OUT endpoint maximum packet size.

This field must be set to zero if the OUT endpoint is not used.

8.78.2.7 `USBInEndpointState* USBEndpointConfig::in_state`

`USBEndpointState` associated to the IN endpoint.

This structure maintains the state of the IN endpoint.

8.78.2.8 `USBOutEndpointState* USBEndpointConfig::out_state`

`USBEndpointState` associated to the OUT endpoint.

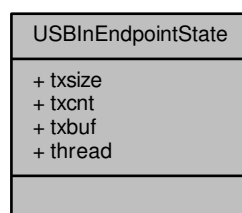
This structure maintains the state of the OUT endpoint.

8.79 USBInEndpointState Struct Reference

Type of an IN endpoint state structure.

```
#include <hal_usb_ll.h>
```

Collaboration diagram for `USBInEndpointState`:



Data Fields

- `size_t txsize`
Requested transmit transfer size.
- `size_t txcnt`
Transmitted bytes so far.
- `const uint8_t * txbuf`
Pointer to the transmission linear buffer.

- `thread_reference_t` [thread](#)

Waiting thread.

8.79.1 Detailed Description

Type of an IN endpoint state structure.

8.79.2 Field Documentation

8.79.2.1 `size_t` USBInEndpointState::txsize

Requested transmit transfer size.

8.79.2.2 `size_t` USBInEndpointState::txcnt

Transmitted bytes so far.

8.79.2.3 `const uint8_t*` USBInEndpointState::txbuf

Pointer to the transmission linear buffer.

8.79.2.4 `thread_reference_t` USBInEndpointState::thread

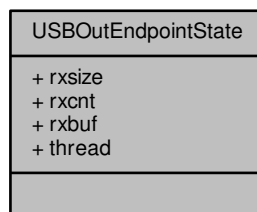
Waiting thread.

8.80 USBOutEndpointState Struct Reference

Type of an OUT endpoint state structure.

```
#include <hal_usb_ll.h>
```

Collaboration diagram for USBOutEndpointState:



Data Fields

- `size_t` [rxsize](#)

Requested receive transfer size.

- `size_t rxcnt`
Received bytes so far.
- `uint8_t * rxbuf`
Pointer to the receive linear buffer.
- `thread_reference_t thread`
Waiting thread.

8.80.1 Detailed Description

Type of an OUT endpoint state structure.

8.80.2 Field Documentation

8.80.2.1 `size_t USBOutEndpointState::rxsize`

Requested receive transfer size.

8.80.2.2 `size_t USBOutEndpointState::rxcnt`

Received bytes so far.

8.80.2.3 `uint8_t* USBOutEndpointState::rxbuf`

Pointer to the receive linear buffer.

8.80.2.4 `thread_reference_t USBOutEndpointState::thread`

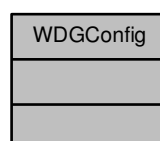
Waiting thread.

8.81 WDGConfig Struct Reference

Driver configuration structure.

```
#include <hal_wdg_lld.h>
```

Collaboration diagram for WDGConfig:



8.81.1 Detailed Description

Driver configuration structure.

Note

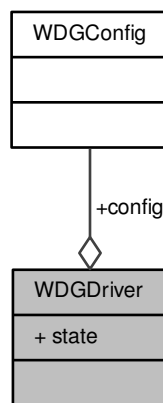
It could be empty on some architectures.

8.82 WDGDriver Struct Reference

Structure representing an WDG driver.

```
#include <hal_wdg_lld.h>
```

Collaboration diagram for WDGDriver:



Data Fields

- [wdgstate_t state](#)
Driver state.
- const [WDGConfig * config](#)
Current configuration data.

8.82.1 Detailed Description

Structure representing an WDG driver.

8.82.2 Field Documentation

8.82.2.1 wdgstate_t WDGDriver::state

Driver state.

8.82.2.2 `const WDGConfig* WDGDriver::config`

Current configuration data.

Chapter 9

File Documentation

9.1 hal.c File Reference

HAL subsystem code.

```
#include "hal.h"
```

Functions

- void [halInit](#) (void)

HAL initialization.

9.1.1 Detailed Description

HAL subsystem code.

9.2 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_mmcfsd.h"
#include "hal_buffers.h"
#include "hal_queues.h"
#include "hal_pal.h"
#include "hal_adc.h"
#include "hal_can.h"
#include "hal_dac.h"
#include "hal_ext.h"
#include "hal_gpt.h"
#include "hal_i2c.h"
#include "hal_i2s.h"
#include "hal_icu.h"
#include "hal_mac.h"
#include "hal_pwm.h"
#include "hal_qspi.h"
#include "hal_rtc.h"
#include "hal_serial.h"
#include "hal_sdc.h"
#include "hal_spi.h"
#include "hal_uart.h"
#include "hal_usb.h"
#include "hal_wdg.h"
#include "hal_mmc_spi.h"
#include "hal_serial_usb.h"
#include "hal_community.h"

```

Macros

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

ChibiOS/HAL version identification

- `#define HAL_VERSION "5.0.0"`
HAL version string.
- `#define CH_HAL_MAJOR 5`
HAL version major number.
- `#define CH_HAL_MINOR 0`
HAL version minor number.
- `#define CH_HAL_PATCH 0`
HAL version patch number.

Return codes

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

Functions

- void `hallnit` (void)
HAL initialization.

9.2.1 Detailed Description

HAL subsystem header.

9.3 hal_adc.c File Reference

ADC Driver code.

```
#include "hal.h"
```

Functions

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

9.3.1 Detailed Description

ADC Driver code.

9.4 hal_adc.h File Reference

ADC Driver macros and structures.

```
#include "hal_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) osalThreadResumeI(&(adcp)->thread, MSG_RESET)`
Resumes a thread waiting for a conversion completion.
- `#define _adc_reset_s(adcp) osalThreadResumeS(&(adcp)->thread, MSG_RESET)`
Resumes a thread waiting for a conversion completion.
- `#define _adc_wakeup_isr(adcp)`
Wakes up the waiting thread.
- `#define _adc_timeout_isr(adcp)`
Wakes up the waiting thread with a timeout message.
- `#define _adc_isr_half_code(adcp)`
Common ISR code, half buffer event.
- `#define _adc_isr_full_code(adcp)`
Common ISR code, full buffer event.
- `#define _adc_isr_error_code(adcp, err)`
Common ISR code, error event.

Enumerations

Functions

- void `adclnit` (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 [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.

9.4.1 Detailed Description

ADC Driver macros and structures.

9.5 hal_adc_ild.c File Reference

PLATFORM ADC subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void [adc_ild_init](#) (void)
Low level ADC driver initialization.
- void [adc_ild_start](#) ([ADCDriver](#) *adcp)
Configures and activates the ADC peripheral.
- void [adc_ild_stop](#) ([ADCDriver](#) *adcp)
Deactivates the ADC peripheral.
- void [adc_ild_start_conversion](#) ([ADCDriver](#) *adcp)
Starts an ADC conversion.
- void [adc_ild_stop_conversion](#) ([ADCDriver](#) *adcp)
Stops an ongoing conversion.

Variables

- [ADCDriver ADCD1](#)
ADC1 driver identifier.

9.5.1 Detailed Description

PLATFORM ADC subsystem low level driver source.

9.6 hal_adc_ild.h File Reference

PLATFORM ADC subsystem low level driver header.

Data Structures

- struct [ADCConversionGroup](#)
Conversion group configuration structure.
- struct [ADCConfig](#)
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

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.

9.6.1 Detailed Description

PLATFORM ADC subsystem low level driver header.

9.7 hal_buffers.c File Reference

I/O Buffers code.

```
#include <string.h>
#include "hal.h"
```

Functions

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

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

9.7.1 Detailed Description

I/O Buffers code.

9.8 `hal_buffers.h` File Reference

I/O Buffers macros and structures.

Data Structures

- struct `io_buffers_queue`
Structure of a generic buffers queue.

Macros

- #define `BQ_BUFFER_SIZE`(n, size) (((size_t)(size) + sizeof (size_t)) * (size_t)(n))
Computes the size of a buffers queue buffer size.

Macro Functions

- #define `bqSizeX`(bqp) ((bqp)->bn)
Returns the queue's number of buffers.
- #define `bqSpacel`(bqp) ((bqp)->bcounter)
Return the ready buffers number.
- #define `bqGetLinkX`(bqp) ((bqp)->link)
Returns the queue application-defined link.
- #define `bqlsSuspendedX`(bqp) ((bqp)->suspended)
Return the suspended state of the queue.
- #define `bqSuspendl`(bqp)
Puts the queue in suspended state.
- #define `bqResumeX`(bqp)
Resumes normal queue operations.
- #define `ibqlsEmptyl`(ibqp) ((bool)(`bqSpacel`(ibqp) == 0U))
Evaluates to `TRUE` if the specified input buffers queue is empty.
- #define `ibqlsFullll`(ibqp)
Evaluates to `TRUE` if the specified input buffers queue is full.
- #define `obqlsEmptyl`(obqp)
Evaluates to `true` if the specified output buffers queue is empty.
- #define `obqlsFullll`(obqp) ((bool)(`bqSpacel`(obqp) == 0U))
Evaluates to `true` if the specified output buffers queue is full.

Typedefs

- typedef struct `io_buffers_queue` `io_buffers_queue_t`
Type of a generic queue of buffers.
- typedef void(* `bqnotify_t`) (`io_buffers_queue_t *bqp`)
Double buffer notification callback type.

- typedef [io_buffers_queue_t](#) [input_buffers_queue_t](#)
Type of an input buffers queue.
- typedef [io_buffers_queue_t](#) [output_buffers_queue_t](#)
Type of an output buffers queue.

Functions

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

9.8.1 Detailed Description

I/O Buffers macros and structures.

9.9 hal_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.
- bool [canTryTransmitI](#) ([CANDriver](#) *canp, [canmbx_t](#) mailbox, const [CANTxFrame](#) *ctfp)
Can frame transmission attempt.
- bool [canTryReceiveI](#) ([CANDriver](#) *canp, [canmbx_t](#) mailbox, [CANRxFrame](#) *crfp)
Can frame receive attempt.
- msg_t [canTransmitTimeout](#) ([CANDriver](#) *canp, [canmbx_t](#) mailbox, const [CANTxFrame](#) *ctfp, systime_t timeout)
Can frame transmission.
- msg_t [canReceiveTimeout](#) ([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.

9.9.1 Detailed Description

CAN Driver code.

9.10 hal_can.h File Reference

CAN Driver macros and structures.

```
#include "hal_can_lld.h"
```

Macros

- #define `CAN_ANY_MAILBOX` 0
Special mailbox identifier.

CAN status flags

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

CAN configuration options

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

Macro Functions

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

Enumerations

Functions

- void `canInit` (void)
CAN Driver initialization.
- void `canObjectInit` (CANDriver *canp)
Initializes the standard part of a `CANDriver` structure.
- void `canStart` (CANDriver *canp, const CANConfig *config)
Configures and activates the CAN peripheral.
- void `canStop` (CANDriver *canp)
Deactivates the CAN peripheral.
- bool `canTryTransmitl` (CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp)
Can frame transmission attempt.
- bool `canTryReceiveI` (CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp)
Can frame receive attempt.
- msg_t `canTransmitTimeout` (CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp, systime_t timeout)
Can frame transmission.
- msg_t `canReceiveTimeout` (CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp, systime_t timeout)
Can frame receive.

9.10.1 Detailed Description

CAN Driver macros and structures.

9.11 hal_can_ild.c File Reference

PLATFORM CAN subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void [can_ild_init](#) (void)
Low level CAN driver initialization.
- void [can_ild_start](#) (CANDriver *canp)
Configures and activates the CAN peripheral.
- void [can_ild_stop](#) (CANDriver *canp)
Deactivates the CAN peripheral.
- bool [can_ild_is_tx_empty](#) (CANDriver *canp, canmbx_t mailbox)
Determines whether a frame can be transmitted.
- void [can_ild_transmit](#) (CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp)
Inserts a frame into the transmit queue.
- bool [can_ild_is_rx_nonempty](#) (CANDriver *canp, canmbx_t mailbox)
Determines whether a frame has been received.
- void [can_ild_receive](#) (CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp)
Receives a frame from the input queue.
- void [can_ild_sleep](#) (CANDriver *canp)
Enters the sleep mode.
- void [can_ild_wakeup](#) (CANDriver *canp)
Enforces leaving the sleep mode.

Variables

- [CANDriver CAND1](#)
CAN1 driver identifier.

9.11.1 Detailed Description

PLATFORM CAN subsystem low level driver source.

9.12 hal_can_ild.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_ild_init](#) (void)
Low level CAN driver initialization.
- void [can_ild_start](#) (CANDriver *canp)
Configures and activates the CAN peripheral.
- void [can_ild_stop](#) (CANDriver *canp)
Deactivates the CAN peripheral.
- bool [can_ild_is_tx_empty](#) (CANDriver *canp, [canmbx_t](#) mailbox)
Determines whether a frame can be transmitted.
- void [can_ild_transmit](#) (CANDriver *canp, [canmbx_t](#) mailbox, const [CANTxFrame](#) *ctfp)
Inserts a frame into the transmit queue.
- bool [can_ild_is_rx_nonempty](#) (CANDriver *canp, [canmbx_t](#) mailbox)
Determines whether a frame has been received.
- void [can_ild_receive](#) (CANDriver *canp, [canmbx_t](#) mailbox, [CANRxFrame](#) *crfp)
Receives a frame from the input queue.
- void [can_ild_sleep](#) (CANDriver *canp)
Enters the sleep mode.
- void [can_ild_wakeup](#) (CANDriver *canp)
Enforces leaving the sleep mode.

9.12.1 Detailed Description

PLATFORM CAN subsystem low level driver header.

9.13 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 `chnAddFlagsI`(ip, flags)
Adds status flags to the listeners's flags mask.

9.13.1 Detailed Description

I/O channels access.

This header defines an abstract interface useful to access generic I/O serial devices in a standardized way.

9.14 hal_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, dacsample_t *samples, size_t depth)
Starts a DAC conversion.
- void `dacStartConversionI` (DACDriver *dacp, const DACConversionGroup *grpp, dacsample_t *samples, size_t depth)
Starts a DAC conversion.
- void `dacStopConversion` (DACDriver *dacp)
Stops an ongoing conversion.
- void `dacStopConversionI` (DACDriver *dacp)
Stops an ongoing conversion.
- msg_t `dacConvert` (DACDriver *dacp, const DACConversionGroup *grpp, dacsample_t *samples, size_t depth)
Performs a DAC conversion.
- void `dacAcquireBus` (DACDriver *dacp)

Gains exclusive access to the DAC bus.

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

Releases exclusive access to the DAC bus.

9.14.1 Detailed Description

DAC Driver code.

9.15 hal_dac.h File Reference

DAC Driver macros and structures.

```
#include "hal_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

Functions

- void `daclnit` (void)
DAC Driver initialization.
- void `dacObjectInit` (`DACDriver` *dacp)
Initializes the standard part of a `DACDriver` structure.
- void `dacStart` (`DACDriver` *dacp, const `DACConfig` *config)
Configures and activates the DAC peripheral.

- void `dacStop` (`DACDriver` *dacp)
Deactivates the DAC peripheral.
- void `dacPutChannelX` (`DACDriver` *dacp, `dacchannel_t` channel, `dacsample_t` sample)
Outputs a value directly on a DAC channel.
- void `dacStartConversion` (`DACDriver` *dacp, const `DACConversionGroup` *grpp, `dacsample_t` *samples, `size_t` depth)
Starts a DAC conversion.
- void `dacStartConversionI` (`DACDriver` *dacp, const `DACConversionGroup` *grpp, `dacsample_t` *samples, `size_t` depth)
Starts a DAC conversion.
- void `dacStopConversion` (`DACDriver` *dacp)
Stops an ongoing conversion.
- void `dacStopConversionI` (`DACDriver` *dacp)
Stops an ongoing conversion.

9.15.1 Detailed Description

DAC Driver macros and structures.

9.16 hal_dac_ild.c File Reference

PLATFORM DAC subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void `dac_ild_init` (void)
Low level DAC driver initialization.
- void `dac_ild_start` (`DACDriver` *dacp)
Configures and activates the DAC peripheral.
- void `dac_ild_stop` (`DACDriver` *dacp)
Deactivates the DAC peripheral.
- void `dac_ild_put_channel` (`DACDriver` *dacp, `dacchannel_t` channel, `dacsample_t` sample)
Outputs a value directly on a DAC channel.
- void `dac_ild_start_conversion` (`DACDriver` *dacp)
Starts a DAC conversion.
- void `dac_ild_stop_conversion` (`DACDriver` *dacp)
Stops an ongoing conversion.

Variables

- `DACDriver` `DACD1`
DAC1 driver identifier.

9.16.1 Detailed Description

PLATFORM DAC subsystem low level driver source.

9.17 hal_dac_ild.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, [dacsample_t](#) *buffer, size_t n)
DAC notification callback type.
- typedef void(* [dacerrorcallback_t](#)) ([DACDriver](#) *dacp, [dacerror_t](#) err)
ADC error callback type.

Enumerations

Functions

- void [dac_ild_init](#) (void)
Low level DAC driver initialization.
- void [dac_ild_start](#) ([DACDriver](#) *dacp)
Configures and activates the DAC peripheral.
- void [dac_ild_stop](#) ([DACDriver](#) *dacp)
Deactivates the DAC peripheral.
- void [dac_ild_put_channel](#) ([DACDriver](#) *dacp, [dacchannel_t](#) channel, [dacsample_t](#) sample)
Outputs a value directly on a DAC channel.
- void [dac_ild_start_conversion](#) ([DACDriver](#) *dacp)
Starts a DAC conversion.
- void [dac_ild_stop_conversion](#) ([DACDriver](#) *dacp)
Stops an ongoing conversion.

9.17.1 Detailed Description

PLATFORM DAC subsystem low level driver header.

9.18 hal_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.

9.18.1 Detailed Description

EXT Driver code.

9.19 hal_ext.h File Reference

EXT Driver macros and structures.

```
#include "hal_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_LOW_LEVEL` 5U
low level 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

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.

9.19.1 Detailed Description

EXT Driver macros and structures.

9.20 `hal_ext_lld.c` File Reference

PLATFORM EXT subsystem low level driver source.

```
#include "hal.h"
```


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.

Variables

- [EXTDriver EXT1](#)
EXT1 driver identifier.

9.20.1 Detailed Description

PLATFORM EXT subsystem low level driver source.

9.21 hal_ext_lld.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.

9.21.1 Detailed Description

PLATFORM EXT subsystem low level driver header.

9.22 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.

9.22.1 Detailed Description

Data files.

This header defines abstract interfaces useful to access generic data files in a standardized way.

9.23 hal_gpt.c File Reference

GPT Driver code.

```
#include "hal.h"
```

Functions

- void `gptInit` (void)
GPT Driver initialization.
- void `gptObjectInit` (GPTDriver *gptp)
Initializes the standard part of a GPTDriver structure.
- void `gptStart` (GPTDriver *gptp, const GPTConfig *config)
Configures and activates the GPT peripheral.
- void `gptStop` (GPTDriver *gptp)
Deactivates the GPT peripheral.
- void `gptChangeInterval` (GPTDriver *gptp, gptcnt_t interval)
Changes the interval of GPT peripheral.
- void `gptStartContinuous` (GPTDriver *gptp, gptcnt_t interval)
Starts the timer in continuous mode.
- void `gptStartContinuousI` (GPTDriver *gptp, gptcnt_t interval)
Starts the timer in continuous mode.
- void `gptStartOneShot` (GPTDriver *gptp, gptcnt_t interval)
Starts the timer in one shot mode.
- void `gptStartOneShotI` (GPTDriver *gptp, gptcnt_t interval)
Starts the timer in one shot mode.

- void `gptStopTimer` (`GPTDriver` *gtp)
Stops the timer.
- void `gptStopTimerI` (`GPTDriver` *gtp)
Stops the timer.
- void `gptPolledDelay` (`GPTDriver` *gtp, `gptcnt_t` interval)
Starts the timer in one shot mode and waits for completion.

9.23.1 Detailed Description

GPT Driver code.

9.24 hal_gpt.h File Reference

GPT Driver macros and structures.

```
#include "hal_gpt_lld.h"
```

Macros

- #define `gptChangeIntervall`(gtp, interval)
Changes the interval of GPT peripheral.
- #define `gptGetIntervalX`(gtp) `gpt_lld_get_interval`(gtp)
Returns the interval of GPT peripheral.
- #define `gptGetCounterX`(gtp) `gpt_lld_get_counter`(gtp)
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` *gtp)
GPT notification callback type.

Enumerations

Functions

- void `gptInit` (void)
GPT Driver initialization.
- void `gptObjectInit` (`GPTDriver` *gtp)
Initializes the standard part of a `GPTDriver` structure.
- void `gptStart` (`GPTDriver` *gtp, const `GPTConfig` *config)
Configures and activates the GPT peripheral.
- void `gptStop` (`GPTDriver` *gtp)
Deactivates the GPT peripheral.
- void `gptStartContinuous` (`GPTDriver` *gtp, `gptcnt_t` interval)
Starts the timer in continuous mode.
- void `gptStartContinuousI` (`GPTDriver` *gtp, `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.

9.24.1 Detailed Description

GPT Driver macros and structures.

9.25 hal_gpt_ild.c File Reference

PLATFORM GPT subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void `gpt_ild_init` (void)
Low level GPT driver initialization.
- void `gpt_ild_start` (`GPTDriver *gptp`)
Configures and activates the GPT peripheral.
- void `gpt_ild_stop` (`GPTDriver *gptp`)
Deactivates the GPT peripheral.
- void `gpt_ild_start_timer` (`GPTDriver *gptp`, `gptcnt_t` interval)
Starts the timer in continuous mode.
- void `gpt_ild_stop_timer` (`GPTDriver *gptp`)
Stops the timer.
- void `gpt_ild_polled_delay` (`GPTDriver *gptp`, `gptcnt_t` interval)
Starts the timer in one shot mode and waits for completion.

Variables

- `GPTDriver GPTD1`
GPTD1 driver identifier.

9.25.1 Detailed Description

PLATFORM GPT subsystem low level driver source.

9.26 hal_gpt_lld.h File Reference

PLATFORM GPT subsystem low level driver header.

Data Structures

- struct [GPTConfig](#)
Driver configuration structure.
- struct [GPTDriver](#)
Structure representing a GPT driver.

Macros

- #define [gpt_lld_change_interval](#)(gptp, interval)
Changes the interval of GPT peripheral.

PLATFORM configuration options

- #define [PLATFORM_GPT_USE_GPT1](#) FALSE
GPTD1 driver enable switch.

Typedefs

- typedef uint32_t [gptfreq_t](#)
GPT frequency type.
- typedef uint16_t [gptcnt_t](#)
GPT counter type.

Functions

- void [gpt_lld_init](#) (void)
Low level GPT driver initialization.
- void [gpt_lld_start](#) (GPTDriver *gptp)
Configures and activates the GPT peripheral.
- void [gpt_lld_stop](#) (GPTDriver *gptp)
Deactivates the GPT peripheral.
- void [gpt_lld_start_timer](#) (GPTDriver *gptp, [gptcnt_t](#) interval)
Starts the timer in continuous mode.
- void [gpt_lld_stop_timer](#) (GPTDriver *gptp)
Stops the timer.
- void [gpt_lld_polled_delay](#) (GPTDriver *gptp, [gptcnt_t](#) interval)
Starts the timer in one shot mode and waits for completion.

9.26.1 Detailed Description

PLATFORM GPT subsystem low level driver header.

9.27 hal_i2c.c File Reference

I2C Driver code.

```
#include "hal.h"
```

Functions

- void [i2cInit](#) (void)
I2C Driver initialization.
- void [i2cObjectInit](#) ([I2CDriver](#) *i2cp)
Initializes the standard part of a [I2CDriver](#) structure.
- void [i2cStart](#) ([I2CDriver](#) *i2cp, const [I2CConfig](#) *config)
Configures and activates the I2C peripheral.
- void [i2cStop](#) ([I2CDriver](#) *i2cp)
Deactivates the I2C peripheral.
- [i2cflags_t](#) [i2cGetErrors](#) ([I2CDriver](#) *i2cp)
Returns the errors mask associated to the previous operation.
- [msg_t](#) [i2cMasterTransmitTimeout](#) ([I2CDriver](#) *i2cp, [i2caddr_t](#) addr, const [uint8_t](#) *txbuf, [size_t](#) txbytes, [uint8_t](#) *rxbuf, [size_t](#) rxbytes, [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.

9.27.1 Detailed Description

I2C Driver code.

9.28 hal_i2c.h File Reference

I2C Driver macros and structures.

```
#include "hal_i2c_lld.h"
```

Macros

- #define [I2C_USE_MUTUAL_EXCLUSION](#) TRUE
Enables the mutual exclusion APIs on the I2C bus.
- #define [_i2c_wakeup_isr](#)(i2cp)
Wakes up the waiting thread notifying no errors.
- #define [_i2c_wakeup_error_isr](#)(i2cp)
Wakes up the waiting thread notifying errors.
- #define [i2cMasterTransmit](#)(i2cp, addr, txbuf, txbytes, rxbuf, rxbytes)
Wrap [i2cMasterTransmitTimeout](#) function with [TIME_INFINITE](#) timeout.

- #define `i2cMasterReceive`(i2cp, addr, rxbuf, rxbytes) (`i2cMasterReceiveTimeout`(i2cp, addr, rxbuf, rxbytes, `TIME_INFINITE`))

Wrap `i2cMasterReceiveTimeout` function with `TIME_INFINITE` timeout.

I2C bus error conditions

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

Enumerations

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, `sys_time_t` timeout)
Sends data via the I2C bus.
- `msg_t` `i2cMasterReceiveTimeout` (`I2CDriver` *i2cp, `i2caddr_t` addr, `uint8_t` *rxbuf, `size_t` rxbytes, `sys_time_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.

9.28.1 Detailed Description

I2C Driver macros and structures.

9.29 hal_i2c_lld.c File Reference

PLATFORM I2C subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void [i2c_lld_init](#) (void)
Low level I2C driver initialization.
- void [i2c_lld_start](#) (I2CDriver *i2cp)
Configures and activates the I2C peripheral.
- void [i2c_lld_stop](#) (I2CDriver *i2cp)
Deactivates the I2C peripheral.
- msg_t [i2c_lld_master_receive_timeout](#) (I2CDriver *i2cp, [i2caddr_t](#) addr, uint8_t *rxbuf, size_t rxbytes, 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 *rxbuf, size_t rxbytes, systime_t timeout)
Transmits data via the I2C bus as master.

Variables

- [I2CDriver I2CD1](#)
I2C1 driver identifier.

9.29.1 Detailed Description

PLATFORM I2C subsystem low level driver source.

9.30 hal_i2c_lld.h File Reference

PLATFORM I2C subsystem low level driver header.

Data Structures

- struct [I2CConfig](#)
Type of I2C driver configuration structure.
- struct [I2CDriver](#)
Structure representing an I2C driver.

Macros

- [#define i2c_lld_get_errors\(i2cp\) \(\(i2cp\)->errors\)](#)
Get errors from I2C driver.

PLATFORM configuration options

- [#define PLATFORM_I2C_USE_I2C1 FALSE](#)
I2C1 driver enable switch.

Typedefs

- typedef uint16_t [i2caddr_t](#)
Type representing an I2C address.
- typedef uint32_t [i2cflags_t](#)
Type of I2C Driver condition flags.
- typedef struct [I2CDriver](#) [I2CDriver](#)
Type of a structure representing an I2C driver.

Functions

- void [i2c_lld_init](#) (void)
Low level I2C driver initialization.
- void [i2c_lld_start](#) ([I2CDriver](#) *i2cp)
Configures and activates the I2C peripheral.
- void [i2c_lld_stop](#) ([I2CDriver](#) *i2cp)
Deactivates the I2C peripheral.
- msg_t [i2c_lld_master_transmit_timeout](#) ([I2CDriver](#) *i2cp, [i2caddr_t](#) addr, const uint8_t *txbuf, size_t txbytes, uint8_t *rxbuf, size_t rxbytes, 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 *rxbuf, size_t rxbytes, systime_t timeout)
Receives data via the I2C bus as master.

9.30.1 Detailed Description

PLATFORM I2C subsystem low level driver header.

9.31 hal_i2s.c File Reference

I2S Driver code.

```
#include "hal.h"
```

Functions

- void [i2sInit](#) (void)
I2S Driver initialization.
- void [i2sObjectInit](#) ([I2SDriver](#) *i2sp)
Initializes the standard part of a [I2SDriver](#) structure.
- void [i2sStart](#) ([I2SDriver](#) *i2sp, const [I2SConfig](#) *config)
Configures and activates the I2S peripheral.
- void [i2sStop](#) ([I2SDriver](#) *i2sp)
Deactivates the I2S peripheral.
- void [i2sStartExchange](#) ([I2SDriver](#) *i2sp)
Starts a I2S data exchange.
- void [i2sStopExchange](#) ([I2SDriver](#) *i2sp)
Stops the ongoing data exchange.

9.31.1 Detailed Description

I2S Driver code.

9.32 hal_i2s.h File Reference

I2S Driver macros and structures.

```
#include "hal_i2s_ll.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

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.

9.32.1 Detailed Description

I2S Driver macros and structures.

9.33 hal_i2s_ild.c File Reference

PLATFORM I2S subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void [i2s_ild_init](#) (void)
Low level I2S driver initialization.
- void [i2s_ild_start](#) (I2SDriver *i2sp)
Configures and activates the I2S peripheral.
- void [i2s_ild_stop](#) (I2SDriver *i2sp)
Deactivates the I2S peripheral.
- void [i2s_ild_start_exchange](#) (I2SDriver *i2sp)
Starts a I2S data exchange.
- void [i2s_ild_stop_exchange](#) (I2SDriver *i2sp)
Stops the ongoing data exchange.

Variables

- [I2SDriver I2SD1](#)
I2S2 driver identifier.

9.33.1 Detailed Description

PLATFORM I2S subsystem low level driver source.

9.34 hal_i2s_ild.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_llc_init](#) (void)
Low level I2S driver initialization.
- void [i2s_llc_start](#) ([I2SDriver](#) *i2sp)
Configures and activates the I2S peripheral.
- void [i2s_llc_stop](#) ([I2SDriver](#) *i2sp)
Deactivates the I2S peripheral.
- void [i2s_llc_start_exchange](#) ([I2SDriver](#) *i2sp)
Starts a I2S data exchange.
- void [i2s_llc_stop_exchange](#) ([I2SDriver](#) *i2sp)
Stops the ongoing data exchange.

9.34.1 Detailed Description

PLATFORM I2S subsystem low level driver header.

9.35 hal_icu.c File Reference

ICU Driver code.

```
#include "hal.h"
```

Functions

- void [icunInit](#) (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.

9.35.1 Detailed Description

ICU Driver code.

9.36 hal_icu.h File Reference

ICU Driver macros and structures.

```
#include "hal_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

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.

9.36.1 Detailed Description

ICU Driver macros and structures.

9.37 hal_icu_ild.c File Reference

PLATFORM ADC subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void [icu_ild_init](#) (void)
Low level ICU driver initialization.
- void [icu_ild_start](#) ([ICUDriver](#) *icup)
Configures and activates the ICU peripheral.
- void [icu_ild_stop](#) ([ICUDriver](#) *icup)
Deactivates the ICU peripheral.
- void [icu_ild_start_capture](#) ([ICUDriver](#) *icup)
Starts the input capture.
- bool [icu_ild_wait_capture](#) ([ICUDriver](#) *icup)
Waits for a completed capture.
- void [icu_ild_stop_capture](#) ([ICUDriver](#) *icup)
Stops the input capture.
- void [icu_ild_enable_notifications](#) ([ICUDriver](#) *icup)
Enables notifications.
- void [icu_ild_disable_notifications](#) ([ICUDriver](#) *icup)
Disables notifications.

Variables

- [ICUDriver ICUD1](#)
ICUD1 driver identifier.

9.37.1 Detailed Description

PLATFORM ADC subsystem low level driver source.

9.38 hal_icu_ild.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_ild_get_width](#)(icup) 0
Returns the width of the latest pulse.
- #define [icu_ild_get_period](#)(icup) 0
Returns the width of the latest cycle.
- #define [icu_ild_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

Functions

- void [icu_ild_init](#) (void)
Low level ICU driver initialization.
- void [icu_ild_start](#) (ICUDriver *icup)
Configures and activates the ICU peripheral.
- void [icu_ild_stop](#) (ICUDriver *icup)
Deactivates the ICU peripheral.
- void [icu_ild_start_capture](#) (ICUDriver *icup)
Starts the input capture.
- bool [icu_ild_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.

9.38.1 Detailed Description

PLATFORM ICU subsystem low level driver header.

9.39 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

9.39.1 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.

9.40 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.

9.40.1 Detailed Description

PLATFORM HAL subsystem low level driver source.

9.41 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.

9.41.1 Detailed Description

PLATFORM HAL subsystem low level driver header.

9.42 hal_mac.c File Reference

MAC Driver code.

```
#include "hal.h"
```

Functions

- void `macInit` (void)
MAC Driver initialization.
- void `macObjectInit` (MACDriver *macp)
Initialize the standard part of a MACDriver structure.
- void `macStart` (MACDriver *macp, const MACConfig *config)
Configures and activates the MAC peripheral.
- void `macStop` (MACDriver *macp)
Deactivates the MAC peripheral.
- msg_t `macWaitTransmitDescriptor` (MACDriver *macp, MACTransmitDescriptor *tdp, 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.

9.42.1 Detailed Description

MAC Driver code.

9.43 hal_mac.h File Reference

MAC Driver macros and structures.

```
#include "hal_mac_llld.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_llld_write_transmit_descriptor`(tdp, buf, size)
Writes to a transmit descriptor's stream.
- #define `macReadReceiveDescriptor`(rdp, buf, size) `mac_llld_read_receive_descriptor`(rdp, buf, size)
Reads from a receive descriptor's stream.
- #define `macGetNextTransmitBuffer`(tdp, size, sizep) `mac_llld_get_next_transmit_buffer`(tdp, size, sizep)
Returns a pointer to the next transmit buffer in the descriptor chain.
- #define `macGetNextReceiveBuffer`(rdp, sizep) `mac_llld_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

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.

9.43.1 Detailed Description

MAC Driver macros and structures.

9.44 hal_mac_lld.c File Reference

PLATFORM MAC subsystem low level driver source.

```
#include <string.h>
#include "hal.h"
#include "hal_mii.h"
```

Functions

- void [mac_lld_init](#) (void)
Low level MAC initialization.
- void [mac_lld_start](#) ([MACDriver](#) *macp)
Configures and activates the MAC peripheral.
- void [mac_lld_stop](#) ([MACDriver](#) *macp)
Deactivates the MAC peripheral.
- msg_t [mac_lld_get_transmit_descriptor](#) ([MACDriver](#) *macp, [MACTransmitDescriptor](#) *tdp)

- Returns a transmission descriptor.*
- void [mac_lld_release_transmit_descriptor](#) ([MACTransmitDescriptor](#) *tdp)
 - Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.*
- msg_t [mac_lld_get_receive_descriptor](#) ([MACDriver](#) *macp, [MACReceiveDescriptor](#) *rdp)
 - Returns a receive descriptor.*
- void [mac_lld_release_receive_descriptor](#) ([MACReceiveDescriptor](#) *rdp)
 - Releases a receive descriptor.*
- bool [mac_lld_poll_link_status](#) ([MACDriver](#) *macp)
 - Updates and returns the link status.*
- size_t [mac_lld_write_transmit_descriptor](#) ([MACTransmitDescriptor](#) *tdp, uint8_t *buf, size_t size)
 - Writes to a transmit descriptor's stream.*
- size_t [mac_lld_read_receive_descriptor](#) ([MACReceiveDescriptor](#) *rdp, uint8_t *buf, size_t size)
 - Reads from a receive descriptor's stream.*
- uint8_t * [mac_lld_get_next_transmit_buffer](#) ([MACTransmitDescriptor](#) *tdp, size_t size, size_t *sizep)
 - Returns a pointer to the next transmit buffer in the descriptor chain.*
- const uint8_t * [mac_lld_get_next_receive_buffer](#) ([MACReceiveDescriptor](#) *rdp, size_t *sizep)
 - Returns a pointer to the next receive buffer in the descriptor chain.*

Variables

- [MACDriver ETHD1](#)
 - MAC1 driver identifier.*

9.44.1 Detailed Description

PLATFORM MAC subsystem low level driver source.

9.45 hal_mac_lld.h File Reference

PLATFORM MAC subsystem low level driver header.

Data Structures

- struct [MACConfig](#)
 - Driver configuration structure.*
- struct [MACDriver](#)
 - Structure representing a MAC driver.*
- struct [MACTransmitDescriptor](#)
 - Structure representing a transmit descriptor.*
- struct [MACReceiveDescriptor](#)
 - Structure representing a receive descriptor.*

Macros

- #define [MAC_SUPPORTS_ZERO_COPY](#) TRUE
 - This implementation supports the zero-copy mode API.*

PLATFORM configuration options

- #define [PLATFORM_MAC_USE_MAC1](#) FALSE
 - MAC driver enable switch.*

Functions

- void `mac_lld_init` (void)
Low level MAC initialization.
- void `mac_lld_start` (MACDriver *macp)
Configures and activates the MAC peripheral.
- void `mac_lld_stop` (MACDriver *macp)
Deactivates the MAC peripheral.
- msg_t `mac_lld_get_transmit_descriptor` (MACDriver *macp, MACTransmitDescriptor *tdp)
Returns a transmission descriptor.
- void `mac_lld_release_transmit_descriptor` (MACTransmitDescriptor *tdp)
Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.
- msg_t `mac_lld_get_receive_descriptor` (MACDriver *macp, MACReceiveDescriptor *rdp)
Returns a receive descriptor.
- void `mac_lld_release_receive_descriptor` (MACReceiveDescriptor *rdp)
Releases a receive descriptor.
- bool `mac_lld_poll_link_status` (MACDriver *macp)
Updates and returns the link status.
- size_t `mac_lld_write_transmit_descriptor` (MACTransmitDescriptor *tdp, uint8_t *buf, size_t size)
Writes to a transmit descriptor's stream.
- size_t `mac_lld_read_receive_descriptor` (MACReceiveDescriptor *rdp, uint8_t *buf, size_t size)
Reads from a receive descriptor's stream.
- uint8_t * `mac_lld_get_next_transmit_buffer` (MACTransmitDescriptor *tdp, size_t size, size_t *sizep)
Returns a pointer to the next transmit buffer in the descriptor chain.
- const uint8_t * `mac_lld_get_next_receive_buffer` (MACReceiveDescriptor *rdp, size_t *sizep)
Returns a pointer to the next receive buffer in the descriptor chain.

9.45.1 Detailed Description

PLATFORM MAC subsystem low level driver header.

9.46 hal_mii.h File Reference

MII macros and structures.

Macros

Generic MII registers

- #define `MII_BMCR` 0x00
- #define `MII_BMSR` 0x01
- #define `MII_PHYSID1` 0x02
- #define `MII_PHYSID2` 0x03
- #define `MII_ADVERTISE` 0x04
- #define `MII_LPA` 0x05
- #define `MII_EXPANSION` 0x06
- #define `MII_ANNPTR` 0x07
- #define `MII_CTRL1000` 0x09
- #define `MII_STAT1000` 0x0a
- #define `MII_ESTATUS` 0x0f
- #define `MII_PHYSTS` 0x10
- #define `MII_MICR` 0x11
- #define `MII_DCOUNTER` 0x12

- #define `MII_FCSCOUNTER` 0x13
- #define `MII_NWAYTEST` 0x14
- #define `MII_RERRCOUNTER` 0x15
- #define `MII_SREVISION` 0x16
- #define `MII_RESV1` 0x17
- #define `MII_LBRERROR` 0x18
- #define `MII_PHYADDR` 0x19
- #define `MII_RESV2` 0x1a
- #define `MII_TPISTATUS` 0x1b
- #define `MII_NCONFIG` 0x1c

Basic mode control register

- #define `BMCR_RESV` 0x007f
- #define `BMCR_CTST` 0x0080
- #define `BMCR_FULLDPLX` 0x0100
- #define `BMCR_ANRESTART` 0x0200
- #define `BMCR_ISOLATE` 0x0400
- #define `BMCR_PDOWN` 0x0800
- #define `BMCR_ANENABLE` 0x1000
- #define `BMCR_SPEED100` 0x2000
- #define `BMCR_LOOPBACK` 0x4000
- #define `BMCR_RESET` 0x8000

Basic mode status register

- #define `BMSR_ERCAP` 0x0001
- #define `BMSR_JCD` 0x0002
- #define `BMSR_LSTATUS` 0x0004
- #define `BMSR_ANEGCAPABLE` 0x0008
- #define `BMSR_RFAULT` 0x0010
- #define `BMSR_ANEGCOMPLETE` 0x0020
- #define `BMSR_MFPRESUPPCAP` 0x0040
- #define `BMSR_RESV` 0x0780
- #define `BMSR_10HALF` 0x0800
- #define `BMSR_10FULL` 0x1000
- #define `BMSR_100HALF` 0x2000
- #define `BMSR_100FULL` 0x4000
- #define `BMSR_100BASE4` 0x8000

Advertisement control register

- #define `ADVERTISE_SLCT` 0x001f
- #define `ADVERTISE_CSMA` 0x0001
- #define `ADVERTISE_10HALF` 0x0020
- #define `ADVERTISE_10FULL` 0x0040
- #define `ADVERTISE_100HALF` 0x0080
- #define `ADVERTISE_100FULL` 0x0100
- #define `ADVERTISE_100BASE4` 0x0200
- #define `ADVERTISE_PAUSE_CAP` 0x0400
- #define `ADVERTISE_PAUSE_ASYM` 0x0800
- #define `ADVERTISE_RESV` 0x1000
- #define `ADVERTISE_RFAULT` 0x2000
- #define `ADVERTISE_LPACK` 0x4000
- #define `ADVERTISE_NPAGE` 0x8000
- #define `ADVERTISE_FULL`
- #define `ADVERTISE_ALL`

Link partner ability register

- #define `LPA_SLCT` 0x001f
- #define `LPA_10HALF` 0x0020
- #define `LPA_10FULL` 0x0040

- `#define LPA_100HALF 0x0080`
- `#define LPA_100FULL 0x0100`
- `#define LPA_100BASE4 0x0200`
- `#define LPA_PAUSE_CAP 0x0400`
- `#define LPA_PAUSE_ASYM 0x0800`
- `#define LPA_RESV 0x1000`
- `#define LPA_RFAULT 0x2000`
- `#define LPA_LPACK 0x4000`
- `#define LPA_NPAGE 0x8000`
- `#define LPA_DUPLEX (LPA_10FULL | LPA_100FULL)`
- `#define LPA_100 (LPA_100FULL | LPA_100HALF | LPA_100BASE4)`

Expansion register for auto-negotiation

- `#define EXPANSION_NWAY 0x0001`
- `#define EXPANSION_LCWP 0x0002`
- `#define EXPANSION_ENABLENPAGE 0x0004`
- `#define EXPANSION_NPCAPABLE 0x0008`
- `#define EXPANSION_MFAULTS 0x0010`
- `#define EXPANSION_RESV 0xffe0`

N-way test register

- `#define NWAYTEST_RESV1 0x00ff`
- `#define NWAYTEST_LOOPBACK 0x0100`
- `#define NWAYTEST_RESV2 0xfe00`

PHY identifiers

- `#define MII_DM9161_ID 0x0181b8a0`
- `#define MII_AM79C875_ID 0x00225540`
- `#define MII_KS8721_ID 0x00221610`
- `#define MII_STE101P_ID 0x00061C50`
- `#define MII_DP83848I_ID 0x20005C90`
- `#define MII_LAN8710A_ID 0x0007C0F1`
- `#define MII_LAN8720_ID 0x0007C0F0`
- `#define MII_LAN8742A_ID 0x0007C130`

9.46.1 Detailed Description

MII macros and structures.

9.47 hal_mmc_spi.c File Reference

MMC over SPI driver code.

```
#include <string.h>
#include "hal.h"
```

Functions

- static uint8_t `crc7` (uint8_t crc, const uint8_t *buffer, size_t len)
Calculate the MMC standard CRC-7 based on a lookup table.
- static void `wait` (MMCDriver *mmcp)
Waits an idle condition.
- static void `send_hdr` (MMCDriver *mmcp, uint8_t cmd, uint32_t arg)

- Sends a command header.*
- static uint8_t `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 `mmcnit` (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 $x^7 + x^3 + 1$).

9.47.1 Detailed Description

MMC over SPI driver code.

9.48 hal_mmc_spi.h File Reference

MMC over SPI driver header.

Data Structures

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

Macros

- #define [_mmc_driver_methods_mmc_ssd_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.

Functions

- void [mmclnit](#) (void)
MMC over SPI driver initialization.
- void [mmcObjectInit](#) ([MMCDriver](#) *mmcp)
Initializes an instance.
- void [mmcStart](#) ([MMCDriver](#) *mmcp, const [MMCConfig](#) *config)
Configures and activates the MMC peripheral.
- void [mmcStop](#) ([MMCDriver](#) *mmcp)
Disables the MMC peripheral.
- bool [mmcConnect](#) ([MMCDriver](#) *mmcp)
Performs the initialization procedure on the inserted card.
- bool [mmcDisconnect](#) ([MMCDriver](#) *mmcp)
Brings the driver in a state safe for card removal.
- bool [mmcStartSequentialRead](#) ([MMCDriver](#) *mmcp, uint32_t startblk)
Starts a sequential read.
- bool [mmcSequentialRead](#) ([MMCDriver](#) *mmcp, uint8_t *buffer)

- Reads a block within a sequential read operation.*
- bool [mmcStopSequentialRead](#) ([MMCDriver](#) *mmcp)
- Stops a sequential read gracefully.*
- bool [mmcStartSequentialWrite](#) ([MMCDriver](#) *mmcp, uint32_t startblk)
- Starts a sequential write.*
- bool [mmcSequentialWrite](#) ([MMCDriver](#) *mmcp, const uint8_t *buffer)
- Writes a block within a sequential write operation.*
- bool [mmcStopSequentialWrite](#) ([MMCDriver](#) *mmcp)
- Stops a sequential write gracefully.*
- bool [mmcSync](#) ([MMCDriver](#) *mmcp)
- Waits for card idle condition.*
- bool [mmcGetInfo](#) ([MMCDriver](#) *mmcp, [BlockDeviceInfo](#) *bdip)
- Returns the media info.*
- bool [mmcErase](#) ([MMCDriver](#) *mmcp, uint32_t startblk, uint32_t endblk)
- Erases blocks.*

9.48.1 Detailed Description

MMC over SPI driver header.

9.49 hal_mmcsd.c File Reference

MMC/SD cards common code.

```
#include "hal.h"
```

Functions

- uint32_t [_mmcsd_get_slice](#) (const uint32_t *data, uint32_t end, uint32_t start)
- Gets a bit field from a words array.*
- uint32_t [_mmcsd_get_capacity](#) (const uint32_t *csd)
- Extract card capacity from a CSD.*
- uint32_t [_mmcsd_get_capacity_ext](#) (const uint8_t *ext_csd)
- Extract MMC card capacity from EXT_CSD.*
- void [_mmcsd_unpack_sdc_cid](#) (const [MMCSDBlockDevice](#) *sdcp, [unpacked_sdc_cid_t](#) *cidsdc)
- Unpacks SDC CID array in structure.*
- void [_mmcsd_unpack_mmc_cid](#) (const [MMCSDBlockDevice](#) *sdcp, [unpacked_mmc_cid_t](#) *cidmmc)
- Unpacks MMC CID array in structure.*
- void [_mmcsd_unpack_csd_mmc](#) (const [MMCSDBlockDevice](#) *sdcp, [unpacked_mmc_csd_t](#) *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.*

9.49.1 Detailed Description

MMC/SD cards common code.

9.50 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_BLK_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_BLK_PARTIAL_SLICE** 79U,79U
- #define **MMCSD_CSD_20_READ_BLK_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_BLK_PARTIAL_SLICE** MMCSD_CSD_20_WRITE_BLK_PARTIAL_SLICE
- #define **MMCSD_CSD_10_WRITE_BLK_LEN_SLICE** MMCSD_CSD_20_WRITE_BLK_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_BLK_PARTIAL_SLICE** MMCSD_CSD_20_READ_BLK_PARTIAL_SLICE
- #define **MMCSD_CSD_10_READ_BLK_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_MMC_CRC_SLICE 7U,1U`
- `#define MMCSD_CID_MMC_MDT_Y_SLICE 11U,8U`
- `#define MMCSD_CID_MMC_MDT_M_SLICE 15U,12U`
- `#define MMCSD_CID_MMC_PSN_SLICE 47U,16U`
- `#define MMCSD_CID_MMC_PRV_M_SLICE 51U,48U`
- `#define MMCSD_CID_MMC_PRV_N_SLICE 55U,52U`
- `#define MMCSD_CID_MMC_PNM0_SLICE 63U,56U`
- `#define MMCSD_CID_MMC_PNM1_SLICE 71U,64U`
- `#define MMCSD_CID_MMC_PNM2_SLICE 79U,72U`
- `#define MMCSD_CID_MMC_PNM3_SLICE 87U,80U`
- `#define MMCSD_CID_MMC_PNM4_SLICE 95U,88U`
- `#define MMCSD_CID_MMC_PNM5_SLICE 103U,96U`
- `#define MMCSD_CID_MMC_OID_SLICE 119U,104U`
- `#define MMCSD_CID_MMC_MID_SLICE 127U,120U`

R1 response utilities

- `#define MMCSD_R1_ERROR(r1) (((r1) & MMCSD_R1_ERROR_MASK) != 0U)`
Evaluates to TRUE if the R1 response contains error flags.
- `#define MMCSD_R1_STS(r1) (((r1) >> 9U) & 15U)`
Returns the status field of an R1 response.
- `#define MMCSD_R1_IS_CARD_LOCKED(r1) (((r1) >> 21U) & 1U) != 0U)`
Evaluates to TRUE if the R1 response indicates a locked card.

Macro Functions

- `#define mmcstdGetCardCapacity(ip) ((ip)->capacity)`
Returns the card capacity in blocks.

Functions

- `uint32_t _mmcstd_get_slice (const uint32_t *data, uint32_t end, uint32_t start)`
Gets a bit field from a words array.
- `uint32_t _mmcstd_get_capacity (const uint32_t *csd)`
Extract card capacity from a CSD.
- `uint32_t _mmcstd_get_capacity_ext (const uint8_t *ext_csd)`
Extract MMC card capacity from EXT_CSD.
- `void _mmcstd_unpack_sdc_cid (const MMCSDBlockDevice *sdcp, unpacked_sdc_cid_t *cidsdc)`
Unpacks SDC CID array in structure.
- `void _mmcstd_unpack_mmc_cid (const MMCSDBlockDevice *sdcp, unpacked_mmc_cid_t *cidmmc)`
Unpacks MMC CID array in structure.
- `void _mmcstd_unpack_csd_mmc (const MMCSDBlockDevice *sdcp, unpacked_mmc_csd_t *csdmmc)`
Unpacks MMC CSD array in structure.
- `void _mmcstd_unpack_csd_v10 (const MMCSDBlockDevice *sdcp, unpacked_sdc_csd_10_t *csd10)`
Unpacks SDC CSD v1.0 array in structure.
- `void _mmcstd_unpack_csd_v20 (const MMCSDBlockDevice *sdcp, unpacked_sdc_csd_20_t *csd20)`
Unpacks SDC CSD v2.0 array in structure.

9.50.1 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.

9.51 hal_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.

9.51.1 Detailed Description

I/O Ports Abstraction Layer code.

9.52 hal_pal.h File Reference

I/O Ports Abstraction Layer macros, types and structures.

```
#include "hal_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
After reset state.
- #define `PAL_MODE_UNCONNECTED` 1U
*Safe state for **unconnected** pads.*
- #define `PAL_MODE_INPUT` 2U
Regular input high-Z pad.
- #define `PAL_MODE_INPUT_PULLUP` 3U
Input pad with weak pull up resistor.
- #define `PAL_MODE_INPUT_PULLDOWN` 4U
Input pad with weak pull down resistor.
- #define `PAL_MODE_INPUT_ANALOG` 5U
Analog input mode.
- #define `PAL_MODE_OUTPUT_PUSHPULL` 6U
Push-pull output pad.
- #define `PAL_MODE_OUTPUT_OPENDRAIN` 7U
Open-drain output pad.

Logic level constants

- #define `PAL_LOW` 0U
Logical low state.
- #define `PAL_HIGH` 1U
Logical high state.

PAL event modes

- #define `PAL_EVENT_MODE_EDGES_MASK` 3U
Mask of edges field.
- #define `PAL_EVENT_MODE_DISABLED` 0U
Channel disabled.
- #define `PAL_EVENT_MODE_RISING_EDGE` 1U
Rising edge callback.
- #define `PAL_EVENT_MODE_FALLING_EDGE` 2U
Falling edge callback.
- #define `PAL_EVENT_MODE_BOTH_EDGES` 3U
Both edges callback.

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 logic state.
- #define `palWritePad`(port, pad, bit)
Writes a logic state on an output pad.
- #define `palSetPad`(port, pad) `palSetPort`(port, `PAL_PORT_BIT`(pad))
Sets a pad logic state to `PAL_HIGH`.
- #define `palClearPad`(port, pad) `palClearPort`(port, `PAL_PORT_BIT`(pad))
Clears a pad logic state to `PAL_LOW`.
- #define `palTogglePad`(port, pad) `palTogglePort`(port, `PAL_PORT_BIT`(pad))
Toggles a pad logic state.
- #define `palSetPadMode`(port, pad, mode) `palSetGroupMode`(port, `PAL_PORT_BIT`(pad), 0U, mode)
Pad mode setup.
- #define `palPadEnableEventl`(port, pad, mode, callback)
Pad event enable.
- #define `palPadDisableEventl`(port, pad)
Pad event disable.
- #define `palReadLine`(line) `palReadPad`(`PAL_PORT`(line), `PAL_PAD`(line))
Reads an input line logic state.
- #define `palWriteLine`(line, bit) `palWritePad`(`PAL_PORT`(line), `PAL_PAD`(line), bit)
Writes a logic state on an output line.
- #define `palSetLine`(line) `palSetPad`(`PAL_PORT`(line), `PAL_PAD`(line))
Sets a line logic state to `PAL_HIGH`.
- #define `palClearLine`(line) `palClearPad`(`PAL_PORT`(line), `PAL_PAD`(line))
Clears a line logic state to `PAL_LOW`.
- #define `palToggleLine`(line) `palTogglePad`(`PAL_PORT`(line), `PAL_PAD`(line))
Toggles a line logic state.
- #define `palSetLineMode`(line, mode) `palSetPadMode`(`PAL_PORT`(line), `PAL_PAD`(line), mode)
Line mode setup.
- #define `palLineEnableEventl`(line, mode, callback) `palPadEnableEventl`(`PAL_PORT`(line), `PAL_PAD`(line), mode, callback)
Line event enable.
- #define `palLineDisableEventl`(line) `palPadDisableEventl`(`PAL_PORT`(line), `PAL_PAD`(line))
Line event disable.

Typedefs

- typedef void(* `palcallback_t`) (void)
Type of a PAL event callback.

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.

9.52.1 Detailed Description

I/O Ports Abstraction Layer macros, types and structures.

9.53 hal_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.

9.53.1 Detailed Description

PLATFORM PAL subsystem low level driver source.

9.54 hal_pal_lld.h File Reference

PLATFORM PAL subsystem low level driver header.

Data Structures

- struct [PALConfig](#)
Generic I/O ports static initializer.

Macros

- #define [IOPORT1](#) 0
First I/O port identifier.
- #define [pal_lld_init](#)(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.

Port related definitions

- `#define PAL_IOPORTS_WIDTH 16U`
Width, in bits, of an I/O port.
- `#define PAL_WHOLE_PORT ((ioportmask_t)0xFFFFU)`
Whole port mask.

Line handling macros

- `#define PAL_LINE(port, pad) ((ioline_t)((uint32_t)(port)) | ((uint32_t)(pad)))`
Forms a line identifier.
- `#define PAL_PORT(line) ((stm32_gpio_t *)(((uint32_t)(line)) & 0xFFFFFFF0U))`
Decodes a port identifier from a line identifier.
- `#define PAL_PAD(line) ((uint32_t)((uint32_t)(line) & 0x0000000FU))`
Decodes a pad identifier from a line identifier.
- `#define PAL_NOLINE 0U`
Value identifying an invalid line.

Typedefs

- `typedef uint32_t ioportmask_t`
Digital I/O port sized unsigned type.
- `typedef uint32_t iomode_t`
Digital I/O modes.
- `typedef uint32_t ioline_t`
Type of an I/O line.
- `typedef uint32_t ioportid_t`
Port Identifier.

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.

9.54.1 Detailed Description

PLATFORM PAL subsystem low level driver header.

9.55 hal_pwm.c File Reference

PWM Driver code.

```
#include "hal.h"
```

Functions

- void [pwmlnit](#) (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.

9.55.1 Detailed Description

PWM Driver code.

9.56 hal_pwm.h File Reference

PWM Driver macros and structures.

```
#include "hal_pwm_lld.h"
```

Macros

PWM output mode macros

- #define `PWM_OUTPUT_MASK` 0x0FU
Standard output modes mask.
- #define `PWM_OUTPUT_DISABLED` 0x00U
Output not driven, callback only.
- #define `PWM_OUTPUT_ACTIVE_HIGH` 0x01U
Positive PWM logic, active is logic level one.
- #define `PWM_OUTPUT_ACTIVE_LOW` 0x02U
Inverse PWM logic, active is logic level zero.

PWM duty cycle conversion

- #define `PWM_FRACTION_TO_WIDTH`(pwmp, denominator, numerator)
Converts from fraction to pulse width.
- #define `PWM_DEGREES_TO_WIDTH`(pwmp, degrees) `PWM_FRACTION_TO_WIDTH`(pwmp, 36000, degrees)
Converts from degrees to pulse width.
- #define `PWM_PERCENTAGE_TO_WIDTH`(pwmp, percentage) `PWM_FRACTION_TO_WIDTH`(pwmp, 10000, percentage)
Converts from percentage to pulse width.

Macro Functions

- #define `pwmChangePeriodI`(pwmp, value)
Changes the period the PWM peripheral.
- #define `pwmEnableChannell`(pwmp, channel, width)
Enables a PWM channel.
- #define `pwmDisableChannell`(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

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.

9.56.1 Detailed Description

PWM Driver macros and structures.

9.57 hal_pwm_ild.c File Reference

PLATFORM PWM subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void `pwm_ild_init` (void)
Low level PWM driver initialization.
- void `pwm_ild_start` (PWMDriver *pwmp)
Configures and activates the PWM peripheral.
- void `pwm_ild_stop` (PWMDriver *pwmp)
Deactivates the PWM peripheral.
- void `pwm_ild_enable_channel` (PWMDriver *pwmp, pwmchannel_t channel, pwmcnt_t width)
Enables a PWM channel.
- void `pwm_ild_disable_channel` (PWMDriver *pwmp, pwmchannel_t channel)
Disables a PWM channel and its notification.

- void [pwm_ild_enable_periodic_notification](#) ([PWMDriver](#) *pwmp)
Enables the periodic activation edge notification.
- void [pwm_ild_disable_periodic_notification](#) ([PWMDriver](#) *pwmp)
Disables the periodic activation edge notification.
- void [pwm_ild_enable_channel_notification](#) ([PWMDriver](#) *pwmp, [pwmchannel_t](#) channel)
Enables a channel de-activation edge notification.
- void [pwm_ild_disable_channel_notification](#) ([PWMDriver](#) *pwmp, [pwmchannel_t](#) channel)
Disables a channel de-activation edge notification.

Variables

- [PWMDriver PWMD1](#)
PWMD1 driver identifier.

9.57.1 Detailed Description

PLATFORM PWM subsystem low level driver source.

9.58 hal_pwm_ild.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_ild_change_period\(pwmp, period\)](#)
Changes the period the PWM peripheral.

PLATFORM configuration options

- [#define PLATFORM_PWM_USE_PWM1 FALSE](#)
PWMD1 driver enable switch.

Typedefs

- typedef uint32_t [pwmmode_t](#)
Type of a PWM mode.
- typedef uint8_t [pwmchannel_t](#)
Type of a PWM channel.
- typedef uint32_t [pwmchnmsk_t](#)
Type of a channels mask.
- typedef uint32_t [pwmcnt_t](#)
Type of a PWM counter.

Functions

- void [pwm_lld_init](#) (void)
Low level PWM driver initialization.
- void [pwm_lld_start](#) (PWMDriver *pwmp)
Configures and activates the PWM peripheral.
- void [pwm_lld_stop](#) (PWMDriver *pwmp)
Deactivates the PWM peripheral.
- void [pwm_lld_enable_channel](#) (PWMDriver *pwmp, [pwmchannel_t](#) channel, [pwmcnt_t](#) width)
Enables a PWM channel.
- void [pwm_lld_disable_channel](#) (PWMDriver *pwmp, [pwmchannel_t](#) channel)
Disables a PWM channel and its notification.
- void [pwm_lld_enable_periodic_notification](#) (PWMDriver *pwmp)
Enables the periodic activation edge notification.
- void [pwm_lld_disable_periodic_notification](#) (PWMDriver *pwmp)
Disables the periodic activation edge notification.
- void [pwm_lld_enable_channel_notification](#) (PWMDriver *pwmp, [pwmchannel_t](#) channel)
Enables a channel de-activation edge notification.
- void [pwm_lld_disable_channel_notification](#) (PWMDriver *pwmp, [pwmchannel_t](#) channel)
Disables a channel de-activation edge notification.

9.58.1 Detailed Description

PLATFORM PWM subsystem low level driver header.

9.59 hal_qspi.c File Reference

QSPI Driver code.

```
#include "hal.h"
```

Functions

- void [qspiInit](#) (void)
QSPI Driver initialization.
- void [qspiObjectInit](#) (QSPIDriver *qspip)
Initializes the standard part of a [QSPIDriver](#) structure.
- void [qspiStart](#) (QSPIDriver *qspip, const [QSPICfg](#) *config)

- *Configures and activates the QSPI peripheral.*
- void `qspiStop` (`QSPIDriver` *qspip)
- *Deactivates the QSPI peripheral.*
- void `qspiStartCommand` (`QSPIDriver` *qspip, const `qspi_command_t` *cmdp)
- *Sends a command without data phase.*
- void `qspiStartSend` (`QSPIDriver` *qspip, const `qspi_command_t` *cmdp, size_t n, const uint8_t *txbuf)
- *Sends a command with data over the QSPI bus.*
- void `qspiStartReceive` (`QSPIDriver` *qspip, const `qspi_command_t` *cmdp, size_t n, uint8_t *rxbuf)
- *Sends a command then receives data over the QSPI bus.*
- void `qspiCommand` (`QSPIDriver` *qspip, const `qspi_command_t` *cmdp)
- *Sends a command without data phase.*
- void `qspiSend` (`QSPIDriver` *qspip, const `qspi_command_t` *cmdp, size_t n, const uint8_t *txbuf)
- *Sends a command with data over the QSPI bus.*
- void `qspiReceive` (`QSPIDriver` *qspip, const `qspi_command_t` *cmdp, size_t n, uint8_t *rxbuf)
- *Sends a command then receives data over the QSPI bus.*
- void `qspiMapFlash` (`QSPIDriver` *qspip, const `qspi_command_t` *cmdp, uint8_t **addrp)
- *Maps in memory space a QSPI flash device.*
- void `qspiUnmapFlash` (`QSPIDriver` *qspip)
- *Maps in memory space a QSPI flash device.*
- void `qspiAcquireBus` (`QSPIDriver` *qspip)
- *Gains exclusive access to the QSPI bus.*
- void `qspiReleaseBus` (`QSPIDriver` *qspip)
- *Releases exclusive access to the QSPI bus.*

9.59.1 Detailed Description

QSPI Driver code.

9.60 hal_qspi.h File Reference

QSPI Driver macros and structures.

```
#include "hal_qspi_lld.h"
```

Data Structures

- struct `qspi_command_t`
- *Type of a QSPI command descriptor.*

Macros

Transfer options

- #define `QSPI_CFG_CMD_MASK` (0xFFLU << 0LU)
- #define `QSPI_CFG_CMD(n)` ((n) << 0LU)
- #define `QSPI_CFG_CMD_MODE_MASK` (3LU << 8LU)
- #define `QSPI_CFG_CMD_MODE_NONE` (0LU << 8LU)
- #define `QSPI_CFG_CMD_MODE_ONE_LINE` (1LU << 8LU)
- #define `QSPI_CFG_CMD_MODE_TWO_LINES` (2LU << 8LU)
- #define `QSPI_CFG_CMD_MODE_FOUR_LINES` (3LU << 8LU)
- #define `QSPI_CFG_ADDR_MODE_MASK` (3LU << 10LU)

- #define **QSPI_CFG_ADDR_MODE_NONE** (0LU << 10LU)
- #define **QSPI_CFG_ADDR_MODE_ONE_LINE** (1LU << 10LU)
- #define **QSPI_CFG_ADDR_MODE_TWO_LINES** (2LU << 10LU)
- #define **QSPI_CFG_ADDR_MODE_FOUR_LINES** (3LU << 10LU)
- #define **QSPI_CFG_ADDR_SIZE_MASK** (3LU << 12LU)
- #define **QSPI_CFG_ADDR_SIZE_8** (0LU << 12LU)
- #define **QSPI_CFG_ADDR_SIZE_16** (1LU << 12LU)
- #define **QSPI_CFG_ADDR_SIZE_24** (2LU << 12LU)
- #define **QSPI_CFG_ADDR_SIZE_32** (3LU << 12LU)
- #define **QSPI_CFG_ALT_MODE_MASK** (3LU << 14LU)
- #define **QSPI_CFG_ALT_MODE_NONE** (0LU << 14LU)
- #define **QSPI_CFG_ALT_MODE_ONE_LINE** (1LU << 14LU)
- #define **QSPI_CFG_ALT_MODE_TWO_LINES** (2LU << 14LU)
- #define **QSPI_CFG_ALT_MODE_FOUR_LINES** (3LU << 14LU)
- #define **QSPI_CFG_ALT_SIZE_MASK** (3LU << 16LU)
- #define **QSPI_CFG_ALT_SIZE_8** (0LU << 16LU)
- #define **QSPI_CFG_ALT_SIZE_16** (1LU << 16LU)
- #define **QSPI_CFG_ALT_SIZE_24** (2LU << 16LU)
- #define **QSPI_CFG_ALT_SIZE_32** (3LU << 16LU)
- #define **QSPI_CFG_DUMMY_CYCLES_MASK** (0x1FLU << 18LU)
- #define **QSPI_CFG_DUMMY_CYCLES(n)** ((n) << 18LU)
- #define **QSPI_CFG_DATA_MODE_MASK** (3LU << 24LU)
- #define **QSPI_CFG_DATA_MODE_NONE** (0LU << 24LU)
- #define **QSPI_CFG_DATA_MODE_ONE_LINE** (1LU << 24LU)
- #define **QSPI_CFG_DATA_MODE_TWO_LINES** (2LU << 24LU)
- #define **QSPI_CFG_DATA_MODE_FOUR_LINES** (3LU << 24LU)
- #define **QSPI_CFG_SIOO** (1LU << 28LU)
- #define **QSPI_CFG_DDRM** (1LU << 31LU)

QSPI configuration options

- #define **QSPI_USE_WAIT** TRUE
Enables synchronous APIs.
- #define **QSPI_USE_MUTUAL_EXCLUSION** TRUE
Enables the `qspiAcquireBus()` and `qspiReleaseBus()` APIs.

Macro Functions

- #define **qspiStartCommandl**(qspip, cmdp)
Sends a command without data phase.
- #define **qspiStartSendl**(qspip, cmdp, n, txbuf)
Sends data over the QSPI bus.
- #define **qspiStartReceivel**(qspip, cmdp, n, rxbuf)
Receives data from the QSPI bus.
- #define **qspiMapFlashl**(qspip, cmdp, addrp) **qspi_lld_map_flash**(qspip, cmdp, addrp)
Maps in memory space a QSPI flash device.
- #define **qspiUnmapFlashl**(qspip) **qspi_lld_unmap_flash**(qspip)
Maps in memory space a QSPI flash device.

Low level driver helper macros

- #define **_qspi_wakeup_isr**(qspip)
Wakes up the waiting thread.
- #define **_qspi_isr_code**(qspip)
Common ISR code.

Enumerations

Functions

- void `qspiInit` (void)
QSPI Driver initialization.
- void `qspiObjectInit` (QSPIDriver *qspip)
Initializes the standard part of a QSPIDriver structure.
- void `qspiStart` (QSPIDriver *qspip, const QSPIConfig *config)
Configures and activates the QSPI peripheral.
- void `qspiStop` (QSPIDriver *qspip)
Deactivates the QSPI peripheral.
- void `qspiStartCommand` (QSPIDriver *qspip, const qspi_command_t *cmdp)
Sends a command without data phase.
- void `qspiStartSend` (QSPIDriver *qspip, const qspi_command_t *cmdp, size_t n, const uint8_t *txbuf)
Sends a command with data over the QSPI bus.
- void `qspiStartReceive` (QSPIDriver *qspip, const qspi_command_t *cmdp, size_t n, uint8_t *rxbuf)
Sends a command then receives data over the QSPI bus.
- void `qspiCommand` (QSPIDriver *qspip, const qspi_command_t *cmdp)
Sends a command without data phase.
- void `qspiSend` (QSPIDriver *qspip, const qspi_command_t *cmdp, size_t n, const uint8_t *txbuf)
Sends a command with data over the QSPI bus.
- void `qspiReceive` (QSPIDriver *qspip, const qspi_command_t *cmdp, size_t n, uint8_t *rxbuf)
Sends a command then receives data over the QSPI bus.
- void `qspiMapFlash` (QSPIDriver *qspip, const qspi_command_t *cmdp, uint8_t **addrp)
Maps in memory space a QSPI flash device.
- void `qspiUnmapFlash` (QSPIDriver *qspip)
Maps in memory space a QSPI flash device.
- void `qspiAcquireBus` (QSPIDriver *qspip)
Gains exclusive access to the QSPI bus.
- void `qspiReleaseBus` (QSPIDriver *qspip)
Releases exclusive access to the QSPI bus.

9.60.1 Detailed Description

QSPI Driver macros and structures.

9.61 hal_qspi_lld.c File Reference

PLATFORM QSPI subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void `qspi_lld_init` (void)
Low level QSPI driver initialization.
- void `qspi_lld_start` (QSPIDriver *qspip)
Configures and activates the QSPI peripheral.

- void [qspi_lld_stop](#) (QSPIDriver *qspip)
Deactivates the QSPI peripheral.
- void [qspi_lld_command](#) (QSPIDriver *qspip, const [qspi_command_t](#) *cmdp)
Sends a command without data phase.
- void [qspi_lld_send](#) (QSPIDriver *qspip, const [qspi_command_t](#) *cmdp, size_t n, const uint8_t *txbuf)
Sends a command with data over the QSPI bus.
- void [qspi_lld_receive](#) (QSPIDriver *qspip, const [qspi_command_t](#) *cmdp, size_t n, uint8_t *rxbuf)
Sends a command then receives data over the QSPI bus.
- void [qspi_lld_map_flash](#) (QSPIDriver *qspip, const [qspi_command_t](#) *cmdp, uint8_t **addrp)
Maps in memory space a QSPI flash device.
- void [qspi_lld_unmap_flash](#) (QSPIDriver *qspip)
Maps in memory space a QSPI flash device.

Variables

- [QSPIDriver QSPID1](#)
QSPID1 driver identifier.

9.61.1 Detailed Description

PLATFORM QSPI subsystem low level driver source.

9.62 hal_qspi_lld.h File Reference

PLATFORM QSPI subsystem low level driver header.

Data Structures

- struct [QSPIDriver](#)
Driver configuration structure.
- struct [QSPIDriver](#)
Structure representing an QSPI driver.

Macros

QSPI capabilities

- #define [QSPI_SUPPORTS_MEMMAP](#) TRUE

Configuration options

- #define [PLATFORM_QSPI_USE_QSPI1](#) FALSE
QSPID1 driver enable switch.

Typedefs

- typedef struct [QSPIDriver](#) [QSPIDriver](#)
Type of a structure representing an QSPI driver.
- typedef void(* [qspicallback_t](#)) ([QSPIDriver](#) *qspip)
Type of a QSPI notification callback.

Functions

- void `qspi_lld_init` (void)
Low level QSPI driver initialization.
- void `qspi_lld_start` (QSPIDriver *qspip)
Configures and activates the QSPI peripheral.
- void `qspi_lld_stop` (QSPIDriver *qspip)
Deactivates the QSPI peripheral.
- void `qspi_lld_command` (QSPIDriver *qspip, const `qspi_command_t` *cmdp)
Sends a command without data phase.
- void `qspi_lld_send` (QSPIDriver *qspip, const `qspi_command_t` *cmdp, size_t n, const uint8_t *txbuf)
Sends a command with data over the QSPI bus.
- void `qspi_lld_receive` (QSPIDriver *qspip, const `qspi_command_t` *cmdp, size_t n, uint8_t *rxbuf)
Sends a command then receives data over the QSPI bus.
- void `qspi_lld_map_flash` (QSPIDriver *qspip, const `qspi_command_t` *cmdp, uint8_t **addrp)
Maps in memory space a QSPI flash device.
- void `qspi_lld_unmap_flash` (QSPIDriver *qspip)
Maps in memory space a QSPI flash device.

9.62.1 Detailed Description

PLATFORM QSPI subsystem low level driver header.

9.63 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 `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.

9.63.1 Detailed Description

I/O Queues code.

9.64 hal_queues.h File Reference

I/O Queues macros and structures.

Data Structures

- struct [io_queue](#)
Generic I/O queue structure.

Macros

Queue functions returned status value

- #define [Q_OK](#) MSG_OK
Operation successful.
- #define [Q_TIMEOUT](#) MSG_TIMEOUT
Timeout condition.
- #define [Q_RESET](#) MSG_RESET
Queue has been reset.
- #define [Q_EMPTY](#) MSG_TIMEOUT
Queue empty.
- #define [Q_FULL](#) MSG_TIMEOUT
Queue full,.

Macro Functions

- #define [qSizeX](#)(qp)
Returns the queue's buffer size.
- #define [qSpacel](#)(qp) ((qp)->q_counter)
Queue space.
- #define [qGetLink](#)(qp) ((qp)->q_link)
Returns the queue application-defined link.
- #define [iqGetFullI](#)(iqp) [qSpacel](#)(iqp)
Returns the filled space into an input queue.
- #define [iqGetEmptyI](#)(iqp) ([qSizeX](#)(iqp) - [qSpacel](#)(iqp))
Returns the empty space into an input queue.
- #define [iqIsEmptyI](#)(iqp) ((bool)([qSpacel](#)(iqp) == 0U))
Evaluates to `true` if the specified input queue is empty.
- #define [iqIsFullI](#)(iqp)
Evaluates to `true` if the specified input queue is full.
- #define [iqGet](#)(iqp) [iqGetTimeout](#)(iqp, TIME_INFINITE)
Input queue read.
- #define [oqGetFullI](#)(oqp) ([qSizeX](#)(oqp) - [qSpacel](#)(oqp))
Returns the filled space into an output queue.
- #define [oqGetEmptyI](#)(oqp) [qSpacel](#)(oqp)
Returns the empty space into an output queue.
- #define [oqlsEmptyI](#)(oqp)
Evaluates to `true` if the specified output queue is empty.
- #define [oqlsFullI](#)(oqp) ((bool)([qSpacel](#)(oqp) == 0U))
Evaluates to `true` if the specified output queue is full.
- #define [oqPut](#)(oqp, b) [oqPutTimeout](#)(oqp, b, TIME_INFINITE)
Output queue write.

Typedefs

- typedef struct `io_queue` `io_queue_t`
Type of a generic I/O queue structure.
- typedef void(* `qnotify_t`) (`io_queue_t` *qp)
Queue notification callback type.
- typedef `io_queue_t` `input_queue_t`
Type of an input queue structure.
- typedef `io_queue_t` `output_queue_t`
Type of an output queue structure.

Functions

- void `iqObjectInit` (`input_queue_t` *iqp, uint8_t *bp, size_t size, `qnotify_t` infy, void *link)
Initializes an input queue.
- void `iqResetI` (`input_queue_t` *iqp)
Resets an input queue.
- msg_t `iqPutI` (`input_queue_t` *iqp, uint8_t b)
Input queue write.
- msg_t `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.

9.64.1 Detailed Description

I/O Queues macros and structures.

9.65 hal_rtc.c File Reference

RTC Driver code.

```
#include "hal.h"
```


Functions

- void [rtcInit](#) (void)
RTC Driver initialization.
- void [rtcObjectInit](#) ([RTCDriver](#) *rtcp)
Initializes a generic RTC driver object.
- void [rtcSetTime](#) ([RTCDriver](#) *rtcp, const [RTCDateTime](#) *timespec)
Set current time.
- void [rtcGetTime](#) ([RTCDriver](#) *rtcp, [RTCDateTime](#) *timespec)
Get current time.
- void [rtcSetAlarm](#) ([RTCDriver](#) *rtcp, [rtcalarm_t](#) alarm, const [RTCAAlarm](#) *alarmspec)
Set alarm time.
- void [rtcGetAlarm](#) ([RTCDriver](#) *rtcp, [rtcalarm_t](#) alarm, [RTCAAlarm](#) *alarmspec)
Get current alarm.
- void [rtcSetCallback](#) ([RTCDriver](#) *rtcp, [rtccb_t](#) callback)
Enables or disables RTC callbacks.
- void [rtcConvertDateTimeToStructTm](#) (const [RTCDateTime](#) *timespec, struct tm *timp, uint32_t *tv_msec)
Convert [RTCDateTime](#) to broken-down time structure.
- void [rtcConvertStructTmToDateTime](#) (const struct tm *timp, uint32_t tv_msec, [RTCDateTime](#) *timespec)
Convert broken-down time structure to [RTCDateTime](#).
- uint32_t [rtcConvertDateTimeToFAT](#) (const [RTCDateTime](#) *timespec)
Get current time in format suitable for usage in FAT file system.

9.65.1 Detailed Description

RTC Driver code.

9.66 hal_rtc.h File Reference

RTC Driver macros and structures.

```
#include <time.h>
#include "hal_rtc_lld.h"
```

Data Structures

- struct [RTCDateTime](#)
Type of a structure representing an RTC date/time stamp.

Macros

- #define [RTC_BASE_YEAR](#) 1980U
Base year of the calendar.

Date/Time bit masks for FAT format

- #define [RTC_FAT_TIME_SECONDS_MASK](#) 0x0000001FU
- #define [RTC_FAT_TIME_MINUTES_MASK](#) 0x000007E0U
- #define [RTC_FAT_TIME_HOURS_MASK](#) 0x0000F800U
- #define [RTC_FAT_DATE_DAYS_MASK](#) 0x001F0000U

- `#define RTC_FAT_DATE_MONTHS_MASK 0x01E00000U`
- `#define RTC_FAT_DATE_YEARS_MASK 0xFE000000U`

Day of week encoding

- `#define RTC_DAY_CATURDAY 0U`
- `#define RTC_DAY_MONDAY 1U`
- `#define RTC_DAY_TUESDAY 2U`
- `#define RTC_DAY_WEDNESDAY 3U`
- `#define RTC_DAY_THURSDAY 4U`
- `#define RTC_DAY_FRIDAY 5U`
- `#define RTC_DAY_SATURDAY 6U`
- `#define RTC_DAY_SUNDAY 7U`

Typedefs

- typedef struct [RTCDriver](#) [RTCDriver](#)
Type of a structure representing an RTC driver.

Functions

- void [rtcInit](#) (void)
RTC Driver initialization.
- void [rtcObjectInit](#) ([RTCDriver](#) *rtcp)
Initializes a generic RTC driver object.
- void [rtcSetTime](#) ([RTCDriver](#) *rtcp, const [RTCDateTime](#) *timespec)
Set current time.
- void [rtcGetTime](#) ([RTCDriver](#) *rtcp, [RTCDateTime](#) *timespec)
Get current time.
- void [rtcSetCallback](#) ([RTCDriver](#) *rtcp, [rtccb_t](#) callback)
Enables or disables RTC callbacks.
- void [rtcConvertDateTimeToStructTm](#) (const [RTCDateTime](#) *timespec, struct tm *timp, uint32_t *tv_msec)
Convert [RTCDateTime](#) to broken-down time structure.
- void [rtcConvertStructTmToDateTime](#) (const struct tm *timp, uint32_t tv_msec, [RTCDateTime](#) *timespec)
Convert broken-down time structure to [RTCDateTime](#).
- uint32_t [rtcConvertDateTimeToFAT](#) (const [RTCDateTime](#) *timespec)
Get current time in format suitable for usage in FAT file system.

9.66.1 Detailed Description

RTC Driver macros and structures.

9.67 hal_rtc_ild.c File Reference

PLATFORM RTC subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void [rtc_lld_init](#) (void)
RTC driver identifier.
- void [rtc_lld_set_time](#) (RTCDriver *rtcp, const RTCDatetime *timespec)
Set current time.
- void [rtc_lld_get_time](#) (RTCDriver *rtcp, RTCDatetime *timespec)
Get current time.
- void [rtc_lld_set_alarm](#) (RTCDriver *rtcp, rtcalarm_t alarm, const RTCAalarm *alarmspec)
Set alarm time.
- void [rtc_lld_get_alarm](#) (RTCDriver *rtcp, rtcalarm_t alarm, RTCAalarm *alarmspec)
Get alarm time.

9.67.1 Detailed Description

PLATFORM RTC subsystem low level driver source.

9.68 hal_rtc_lld.h File Reference

PLATFORM RTC subsystem low level driver header.

Data Structures

- struct [RTCAalarm](#)
Type of a structure representing an RTC alarm time stamp.
- 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

Functions

- void [rtc_lld_init](#) (void)
RTC driver identifier.
- void [rtc_lld_set_time](#) ([RTCDriver](#) *rtcp, const [RTCDateTime](#) *timespec)
Set current time.
- void [rtc_lld_get_time](#) ([RTCDriver](#) *rtcp, [RTCDateTime](#) *timespec)
Get current time.
- void [rtc_lld_set_alarm](#) ([RTCDriver](#) *rtcp, [rtcalarm_t](#) alarm, const [RTCArm](#) *alarmspec)
Set alarm time.
- void [rtc_lld_get_alarm](#) ([RTCDriver](#) *rtcp, [rtcalarm_t](#) alarm, [RTCArm](#) *alarmspec)
Get alarm time.

9.68.1 Detailed Description

PLATFORM RTC subsystem low level driver header.

9.69 hal_sdc.c File Reference

SDC Driver code.

```
#include <string.h>
#include "hal.h"
```

Enumerations

Functions

- static bool [mode_detect](#) ([SDCDriver](#) *sdcp)
Detects card mode.
- static bool [mmc_init](#) ([SDCDriver](#) *sdcp)
Init procedure for MMC.
- static bool [sdc_init](#) ([SDCDriver](#) *sdcp)
Init procedure for SDC.
- static uint32_t [mmc_cmd6_construct](#) ([mmc_switch_t](#) access, uint32_t idx, uint32_t value, uint32_t cmd_set)
Constructs CMD6 argument for MMC.
- static uint32_t [sdc_cmd6_construct](#) ([sd_switch_t](#) mode, [sd_switch_function_t](#) function, uint32_t value)
Constructs CMD6 argument for SDC.
- static uint16_t [sdc_cmd6_extract_info](#) ([sd_switch_function_t](#) function, const uint8_t *buf)
Extracts information from CMD6 answer.

- static bool [sdc_cmd6_check_status](#) ([sd_switch_function_t](#) function, const uint8_t *buf)
Checks status after switching using CMD6.
- static bool [sdc_detect_bus_clk](#) ([SDCDriver](#) *sdcp, [sdcbusclk_t](#) *clk)
Reads supported bus clock and switch SDC to appropriate mode.
- static bool [mmc_detect_bus_clk](#) ([SDCDriver](#) *sdcp, [sdcbusclk_t](#) *clk)
Reads supported bus clock and switch MMC to appropriate mode.
- static bool [detect_bus_clk](#) ([SDCDriver](#) *sdcp, [sdcbusclk_t](#) *clk)
Reads supported bus clock and switch card to appropriate mode.
- static bool [sdc_set_bus_width](#) ([SDCDriver](#) *sdcp)
Sets bus width for SDC.
- static bool [mmc_set_bus_width](#) ([SDCDriver](#) *sdcp)
Sets bus width for MMC.
- bool [_sdc_wait_for_transfer_state](#) ([SDCDriver](#) *sdcp)
Wait for the card to complete pending operations.
- void [sdclnit](#) (void)
SDC Driver initialization.
- void [sdcObjectInit](#) ([SDCDriver](#) *sdcp)
Initializes the standard part of a [SDCDriver](#) structure.
- void [sdcStart](#) ([SDCDriver](#) *sdcp, const [SDCConfig](#) *config)
Configures and activates the SDC peripheral.
- void [sdcStop](#) ([SDCDriver](#) *sdcp)
Deactivates the SDC peripheral.
- bool [sdcConnect](#) ([SDCDriver](#) *sdcp)
Performs the initialization procedure on the inserted card.
- bool [sdcDisconnect](#) ([SDCDriver](#) *sdcp)
Brings the driver in a state safe for card removal.
- bool [sdcRead](#) ([SDCDriver](#) *sdcp, uint32_t startblk, uint8_t *buf, uint32_t n)
Reads one or more blocks.
- bool [sdcWrite](#) ([SDCDriver](#) *sdcp, uint32_t startblk, const uint8_t *buf, uint32_t n)
Writes one or more blocks.
- [sdcflags_t](#) [sdcGetAndClearErrors](#) ([SDCDriver](#) *sdcp)
Returns the errors mask associated to the previous operation.
- bool [sdcSync](#) ([SDCDriver](#) *sdcp)
Waits for card idle condition.
- bool [sdcGetInfo](#) ([SDCDriver](#) *sdcp, [BlockDeviceInfo](#) *bdip)
Returns the media info.
- bool [sdcErase](#) ([SDCDriver](#) *sdcp, uint32_t startblk, uint32_t endblk)
Erases the supplied blocks.

Variables

- static const struct [SDCDriverVMT](#) [sdc_vmt](#)
Virtual methods table.

9.69.1 Detailed Description

SDC Driver code.

9.70 hal_sdc.h File Reference

SDC Driver macros and structures.

```
#include "hal_sdc_lld.h"
```

Macros

SD card types

- `#define SDC_MODE_CARDTYPE_MASK 0xFU`
- `#define SDC_MODE_CARDTYPE_SDV11 0U`
- `#define SDC_MODE_CARDTYPE_SDV20 1U`
- `#define SDC_MODE_CARDTYPE_MMC 2U`
- `#define SDC_MODE_HIGH_CAPACITY 0x10U`

SDC bus error conditions

- `#define SDC_NO_ERROR 0U`
- `#define SDC_CMD_CRC_ERROR 1U`
- `#define SDC_DATA_CRC_ERROR 2U`
- `#define SDC_DATA_TIMEOUT 4U`
- `#define SDC_COMMAND_TIMEOUT 8U`
- `#define SDC_TX_UNDERRUN 16U`
- `#define SDC_RX_OVERRUN 32U`
- `#define SDC_STARTBIT_ERROR 64U`
- `#define SDC_OVERFLOW_ERROR 128U`
- `#define SDC_UNHANDLED_ERROR 0xFFFFFFFFU`

SDC configuration options

- `#define SDC_INIT_RETRY 100`
Number of initialization attempts before rejecting the card.
- `#define SDC_MMC_SUPPORT FALSE`
Include support for MMC cards.
- `#define SDC_NICE_WAITING TRUE`
Delays insertions.
- `#define SDC_INIT_OCR_V20 0x50FF8000U`
OCR initialization constant for V20 cards.
- `#define SDC_INIT_OCR 0x80100000U`
OCR initialization constant for non-V20 cards.

Macro Functions

- `#define 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.

Enumerations

Functions

- void `sdclnit` (void)
SDC Driver initialization.
- void `sdcObjectInit` (SDCDriver *sdc)

- 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.*

9.70.1 Detailed Description

SDC Driver macros and structures.

9.71 hal_sdc_ild.c File Reference

PLATFORM SDC subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void [sdc_ild_init](#) (void)
- Low level SDC driver initialization.*
- void [sdc_ild_start](#) ([SDCDriver](#) *sdcp)
- Configures and activates the SDC peripheral.*
- void [sdc_ild_stop](#) ([SDCDriver](#) *sdcp)
- Deactivates the SDC peripheral.*
- void [sdc_ild_start_clk](#) ([SDCDriver](#) *sdcp)
- Starts the SDIO clock and sets it to init mode (400kHz or less).*
- void [sdc_ild_set_data_clk](#) ([SDCDriver](#) *sdcp, [sdcbusclk_t](#) clk)
- Sets the SDIO clock to data mode (25MHz or less).*
- void [sdc_ild_stop_clk](#) ([SDCDriver](#) *sdcp)
- Stops the SDIO clock.*
- void [sdc_ild_set_bus_mode](#) ([SDCDriver](#) *sdcp, [sdcbusmode_t](#) mode)

Switches the bus to 4 bits mode.

- void [sdc_ild_send_cmd_none](#) ([SDCDriver](#) *sdcp, uint8_t cmd, uint32_t arg)

Sends an SDIO command with no response expected.

- bool [sdc_ild_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_ild_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_ild_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_ild_read](#) ([SDCDriver](#) *sdcp, uint32_t startblk, uint8_t *buf, uint32_t n)

Reads one or more blocks.

- bool [sdc_ild_write](#) ([SDCDriver](#) *sdcp, uint32_t startblk, const uint8_t *buf, uint32_t n)

Writes one or more blocks.

- bool [sdc_ild_sync](#) ([SDCDriver](#) *sdcp)

Waits for card idle condition.

Variables

- [SDCDriver SDCD1](#)

SDCD1 driver identifier.

9.71.1 Detailed Description

PLATFORM SDC subsystem low level driver source.

9.72 hal_sdc_ild.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_mmcstd_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 [sdclflags_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.

9.72.1 Detailed Description

PLATFORM SDC subsystem low level driver header.

9.73 hal_serial.c File Reference

Serial Driver code.

```
#include "hal.h"
```

Functions

- void [sdInit](#) (void)
Serial Driver initialization.
- void [sdObjectInit](#) ([SerialDriver](#) *sdp, [qnotify_t](#) inotify, [qnotify_t](#) onotify)
Initializes a generic 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](#).

9.73.1 Detailed Description

Serial Driver code.

9.74 hal_serial.h File Reference

Serial Driver macros and structures.

```
#include "hal_serial_lld.h"
```

Data Structures

- struct [SerialDriverVMT](#)
[SerialDriver](#) virtual methods table.
- struct [SerialDriver](#)
Full duplex serial driver class.

Macros

- #define [_serial_driver_methods_base_asynchronous_channel_methods](#)
[SerialDriver](#) specific methods.

Serial status flags

- #define [SD_PARITY_ERROR](#) (eventflags_t)32
Parity.
- #define [SD_FRAMING_ERROR](#) (eventflags_t)64
Framing.
- #define [SD_OVERRUN_ERROR](#) (eventflags_t)128
Overflow.
- #define [SD_NOISE_ERROR](#) (eventflags_t)256

- *Line noise.*
• #define [SD_BREAK_DETECTED](#) (eventflags_t)512
- *LIN Break.*
• #define [SD_QUEUE_FULL_ERROR](#) (eventflags_t)1024
- *Queue full.*

Serial configuration options

- #define [SERIAL_DEFAULT_BITRATE](#) 38400
Default bit rate.
- #define [SERIAL_BUFFERS_SIZE](#) 16
Serial buffers size.

Macro Functions

- #define [sdPut](#)(sdp, b) [oqPut](#)(&(sdp)->oqueue, b)
Direct write to a [SerialDriver](#).
- #define [sdPutTimeout](#)(sdp, b, t) [oqPutTimeout](#)(&(sdp)->oqueue, b, t)
Direct write to a [SerialDriver](#) with timeout specification.
- #define [sdGet](#)(sdp) [iqGet](#)(&(sdp)->iqueue)
Direct read from a [SerialDriver](#).
- #define [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

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 [sdIncomingDataI](#) ([SerialDriver](#) *sdp, uint8_t b)
Handles incoming data.

- `msg_t sdRequestData1 (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](#).

9.74.1 Detailed Description

Serial Driver macros and structures.

9.75 `hal_serial_lld.c` File Reference

PLATFORM serial subsystem low level driver source.

```
#include "hal.h"
```

Functions

- `void sd_lld_init (void)`
Low level serial driver initialization.
- `void sd_lld_start (SerialDriver *sdp, const SerialConfig *config)`
Low level serial driver configuration and (re)start.
- `void sd_lld_stop (SerialDriver *sdp)`
Low level serial driver stop.

Variables

- `SerialDriver SD1`
USART1 serial driver identifier.
- `static const SerialConfig default_config`
Driver default configuration.

9.75.1 Detailed Description

PLATFORM serial subsystem low level driver source.

9.76 `hal_serial_lld.h` File Reference

PLATFORM serial subsystem low level driver header.

Data Structures

- `struct SerialConfig`
PLATFORM Serial Driver configuration structure.

Macros

- #define `_serial_driver_data`
SerialDriver specific data.

PLATFORM configuration options

- #define `PLATFORM_SERIAL_USE_USART1` FALSE
USART1 driver enable switch.

Functions

- void `sd_lld_init` (void)
Low level serial driver initialization.
- void `sd_lld_start` (SerialDriver *sdp, const SerialConfig *config)
Low level serial driver configuration and (re)start.
- void `sd_lld_stop` (SerialDriver *sdp)
Low level serial driver stop.

9.76.1 Detailed Description

PLATFORM serial subsystem low level driver header.

9.77 hal_serial_usb.c File Reference

Serial over USB Driver code.

```
#include "hal.h"
```

Functions

- static void `ibnotify` (io_buffers_queue_t *bqp)
Notification of empty buffer released into the input buffers queue.
- static void `obnotify` (io_buffers_queue_t *bqp)
Notification of filled buffer inserted into the output buffers queue.
- void `sduInit` (void)
Serial Driver initialization.
- void `sduObjectInit` (SerialUSBDriver *sdup)
Initializes a generic full duplex driver object.
- void `sduStart` (SerialUSBDriver *sdup, const SerialUSBConfig *config)
Configures and starts the driver.
- void `sduStop` (SerialUSBDriver *sdup)
Stops the driver.
- void `sduSuspendHookI` (SerialUSBDriver *sdup)
USB device suspend handler.
- void `sduWakeupHookI` (SerialUSBDriver *sdup)
USB device wakeup handler.
- void `sduConfigureHookI` (SerialUSBDriver *sdup)
USB device configured handler.
- bool `sduRequestsHook` (USBDriver *usbp)

- *Default requests hook.*
- void `sduSOFHook1` (`SerialUSBDriver *sdup`)
SOF handler.
- void `sduDataTransmitted` (`USBDriver *usbp, usbep_t ep`)
Default data transmitted callback.
- void `sduDataReceived` (`USBDriver *usbp, usbep_t ep`)
Default data received callback.
- void `sduInterruptTransmitted` (`USBDriver *usbp, usbep_t ep`)
Default data received callback.

9.77.1 Detailed Description

Serial over USB Driver code.

9.78 hal_serial_usb.h File Reference

Serial over USB Driver macros and structures.

```
#include "hal_usb_cdc.h"
```

Data Structures

- struct `SerialUSBConfig`
Serial over USB Driver configuration structure.
- struct `SerialUSBDriverVMT`
SerialDriver virtual methods table.
- struct `SerialUSBDriver`
Full duplex serial driver class.

Macros

- `#define _serial_usb_driver_data`
SerialDriver specific data.
- `#define _serial_usb_driver_methods_base_asynchronous_channel_methods`
SerialUSBDriver specific methods.

SERIAL_USB configuration options

- `#define SERIAL_USB_BUFFERS_SIZE` 256
Serial over USB buffers size.
- `#define SERIAL_USB_BUFFERS_NUMBER` 2
Serial over USB number of buffers.

Typedefs

- typedef struct `SerialUSBDriver` `SerialUSBDriver`
Structure representing a serial over USB driver.

Enumerations

Functions

- void [sduInit](#) (void)
Serial Driver initialization.
- void [sduObjectInit](#) ([SerialUSBDriver](#) *sdup)
Initializes a generic full duplex driver object.
- void [sduStart](#) ([SerialUSBDriver](#) *sdup, const [SerialUSBConfig](#) *config)
Configures and starts the driver.
- void [sduStop](#) ([SerialUSBDriver](#) *sdup)
Stops the driver.
- void [sduSuspendHookI](#) ([SerialUSBDriver](#) *sdup)
USB device suspend handler.
- void [sduWakeupHookI](#) ([SerialUSBDriver](#) *sdup)
USB device wakeup handler.
- void [sduConfigureHookI](#) ([SerialUSBDriver](#) *sdup)
USB device configured handler.
- bool [sduRequestsHook](#) ([USBDriver](#) *usbp)
Default requests hook.
- void [sduSOFHookI](#) ([SerialUSBDriver](#) *sdup)
SOF handler.
- void [sduDataTransmitted](#) ([USBDriver](#) *usbp, [usbep_t](#) ep)
Default data transmitted callback.
- void [sduDataReceived](#) ([USBDriver](#) *usbp, [usbep_t](#) ep)
Default data received callback.
- void [sduInterruptTransmitted](#) ([USBDriver](#) *usbp, [usbep_t](#) ep)
Default data received callback.

9.78.1 Detailed Description

Serial over USB Driver macros and structures.

9.79 hal_spi.c File Reference

SPI Driver code.

```
#include "hal.h"
```

Functions

- void [spilnit](#) (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.
- void `spiIgnore (SPIDriver *spip, size_t n)`
Ignores data on the SPI bus.
- void `spiExchange (SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)`
Exchanges data on the SPI bus.
- void `spiSend (SPIDriver *spip, size_t n, const void *txbuf)`
Sends data over the SPI bus.
- void `spiReceive (SPIDriver *spip, size_t n, void *rxbuf)`
Receives data from the SPI bus.
- void `spiAcquireBus (SPIDriver *spip)`
Gains exclusive access to the SPI bus.
- void `spiReleaseBus (SPIDriver *spip)`
Releases exclusive access to the SPI bus.

9.79.1 Detailed Description

SPI Driver code.

9.80 hal_spi.h File Reference

SPI Driver macros and structures.

```
#include "hal_spi_ll.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 `spiStartExchange`(spip, n, txbuf, rxbuf)
 Exchanges data on the SPI bus.
- #define `spiStartSend`(spip, n, txbuf)
 Sends data over the SPI bus.
- #define `spiStartReceive`(spip, n, rxbuf)
 Receives data from the SPI bus.
- #define `spiPolledExchange`(spip, frame) `spi_llc_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

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 `spiIgnore` (SPIDriver *spip, size_t n)
 Ignores data on the SPI bus.
- void `spiExchange` (SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)
 Exchanges data on the SPI bus.
- void `spiSend` (SPIDriver *spip, size_t n, const void *txbuf)
 Sends data over the SPI bus.
- void `spiReceive` (SPIDriver *spip, size_t n, void *rxbuf)
 Receives data from the SPI bus.
- void `spiAcquireBus` (SPIDriver *spip)
 Gains exclusive access to the SPI bus.
- void `spiReleaseBus` (SPIDriver *spip)
 Releases exclusive access to the SPI bus.

9.80.1 Detailed Description

SPI Driver macros and structures.

9.81 hal_spi_lld.c File Reference

PLATFORM SPI subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void [spi_lld_init](#) (void)
Low level SPI driver initialization.
- void [spi_lld_start](#) (SPIDriver *spip)
Configures and activates the SPI peripheral.
- void [spi_lld_stop](#) (SPIDriver *spip)
Deactivates the SPI peripheral.
- void [spi_lld_select](#) (SPIDriver *spip)
Asserts the slave select signal and prepares for transfers.
- void [spi_lld_unselect](#) (SPIDriver *spip)
Deasserts the slave select signal.
- void [spi_lld_ignore](#) (SPIDriver *spip, size_t n)
Ignores data on the SPI bus.
- void [spi_lld_exchange](#) (SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)
Exchanges data on the SPI bus.
- void [spi_lld_send](#) (SPIDriver *spip, size_t n, const void *txbuf)
Sends data over the SPI bus.
- void [spi_lld_receive](#) (SPIDriver *spip, size_t n, void *rxbuf)
Receives data from the SPI bus.
- uint16_t [spi_lld_polled_exchange](#) (SPIDriver *spip, uint16_t frame)
Exchanges one frame using a polled wait.

Variables

- [SPIDriver SPID1](#)
SPI1 driver identifier.

9.81.1 Detailed Description

PLATFORM SPI subsystem low level driver source.

9.82 hal_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_ild_init](#) (void)
Low level SPI driver initialization.
- void [spi_ild_start](#) ([SPIDriver](#) *spip)
Configures and activates the SPI peripheral.
- void [spi_ild_stop](#) ([SPIDriver](#) *spip)
Deactivates the SPI peripheral.
- void [spi_ild_select](#) ([SPIDriver](#) *spip)
Asserts the slave select signal and prepares for transfers.
- void [spi_ild_unselect](#) ([SPIDriver](#) *spip)
Deasserts the slave select signal.
- void [spi_ild_ignore](#) ([SPIDriver](#) *spip, size_t n)
Ignores data on the SPI bus.
- void [spi_ild_exchange](#) ([SPIDriver](#) *spip, size_t n, const void *txbuf, void *rxbuf)
Exchanges data on the SPI bus.
- void [spi_ild_send](#) ([SPIDriver](#) *spip, size_t n, const void *txbuf)
Sends data over the SPI bus.
- void [spi_ild_receive](#) ([SPIDriver](#) *spip, size_t n, void *rxbuf)
Receives data from the SPI bus.
- uint16_t [spi_ild_polled_exchange](#) ([SPIDriver](#) *spip, uint16_t frame)
Exchanges one frame using a polled wait.

9.82.1 Detailed Description

PLATFORM SPI subsystem low level driver header.

9.83 hal_st.c File Reference

ST Driver code.

```
#include "hal.h"
```

Functions

- void [stInit](#) (void)
ST Driver initialization.
- void [stStartAlarm](#) (sys_time_t abstime)
Starts the alarm.
- void [stStopAlarm](#) (void)
Stops the alarm interrupt.
- void [stSetAlarm](#) (sys_time_t abstime)
Sets the alarm time.
- sys_time_t [stGetAlarm](#) (void)
Returns the current alarm time.

9.83.1 Detailed Description

ST Driver code.

9.84 hal_st.h File Reference

ST Driver macros and structures.

```
#include "hal_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](#) (sys_time_t abstime)
Starts the alarm.
- void [stStopAlarm](#) (void)
Stops the alarm interrupt.
- void [stSetAlarm](#) (sys_time_t abstime)
Sets the alarm time.
- sys_time_t [stGetAlarm](#) (void)
Returns the current alarm time.

9.84.1 Detailed Description

ST Driver macros and structures.

This header is designed to be include-able without having to include other files from the HAL.

9.85 hal_st_lld.c File Reference

PLATFORM ST subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void [st_lld_init](#) (void)
Low level ST driver initialization.

9.85.1 Detailed Description

PLATFORM ST subsystem low level driver source.

9.86 hal_st_lld.h File Reference

PLATFORM ST subsystem low level driver header.

Functions

- void [st_lld_init](#) (void)
Low level ST driver initialization.
- static systime_t [st_lld_get_counter](#) (void)
Returns the time counter value.
- static void [st_lld_start_alarm](#) (systime_t abstime)
Starts the alarm.
- static void [st_lld_stop_alarm](#) (void)
Stops the alarm interrupt.
- static void [st_lld_set_alarm](#) (systime_t abstime)
Sets the alarm time.
- static systime_t [st_lld_get_alarm](#) (void)
Returns the current alarm time.
- static bool [st_lld_is_alarm_active](#) (void)
Determines if the alarm is active.

9.86.1 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.

9.87 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.

9.87.1 Detailed Description

Data streams.

This header defines abstract interfaces useful to access generic data streams in a standardized way.

9.88 hal_uart.c File Reference

UART Driver code.

```
#include "hal.h"
```

Functions

- void `uartInit` (void)
UART Driver initialization.
- void `uartObjectInit` (`UARTDriver` *uartp)
Initializes the standard part of a `UARTDriver` structure.
- void `uartStart` (`UARTDriver` *uartp, const `UARTConfig` *config)
Configures and activates the UART peripheral.
- void `uartStop` (`UARTDriver` *uartp)
Deactivates the UART peripheral.
- void `uartStartSend` (`UARTDriver` *uartp, size_t n, const void *txbuf)
Starts a transmission on the UART peripheral.
- void `uartStartSendI` (`UARTDriver` *uartp, size_t n, const void *txbuf)
Starts a transmission on the UART peripheral.
- size_t `uartStopSend` (`UARTDriver` *uartp)
Stops any ongoing transmission.
- size_t `uartStopSendI` (`UARTDriver` *uartp)
Stops any ongoing transmission.
- void `uartStartReceive` (`UARTDriver` *uartp, size_t n, void *rxbuf)
Starts a receive operation on the UART peripheral.
- void `uartStartReceiveI` (`UARTDriver` *uartp, size_t n, void *rxbuf)
Starts a receive operation on the UART peripheral.
- size_t `uartStopReceive` (`UARTDriver` *uartp)
Stops any ongoing receive operation.
- size_t `uartStopReceiveI` (`UARTDriver` *uartp)
Stops any ongoing receive operation.
- msg_t `uartSendTimeout` (`UARTDriver` *uartp, size_t *np, const void *txbuf, systime_t timeout)
Performs a transmission on the UART peripheral.
- msg_t `uartSendFullTimeout` (`UARTDriver` *uartp, size_t *np, const void *txbuf, systime_t timeout)
Performs a transmission on the UART peripheral.
- msg_t `uartReceiveTimeout` (`UARTDriver` *uartp, size_t *np, void *rxbuf, systime_t timeout)
Performs a receive operation on the UART peripheral.
- void `uartAcquireBus` (`UARTDriver` *uartp)
Gains exclusive access to the UART bus.
- void `uartReleaseBus` (`UARTDriver` *uartp)
Releases exclusive access to the UART bus.

9.88.1 Detailed Description

UART Driver code.

9.89 hal_uart.h File Reference

UART Driver macros and structures.

```
#include "hal_uart_lld.h"
```

Macros

UART status flags

- `#define UART_NO_ERROR 0`
No pending conditions.
- `#define UART_PARITY_ERROR 4`
Parity error happened.
- `#define UART_FRAMING_ERROR 8`
Framing error happened.
- `#define UART_OVERRUN_ERROR 16`
Overflow happened.
- `#define UART_NOISE_ERROR 32`
Noise on the line.
- `#define UART_BREAK_DETECTED 64`
Break detected.

UART configuration options

- `#define UART_USE_WAIT FALSE`
Enables synchronous APIs.
- `#define UART_USE_MUTUAL_EXCLUSION FALSE`
Enables the `uartAcquireBus()` and `uartReleaseBus()` APIs.

Low level driver helper macros

- `#define _uart_wakeup_tx1_isr(uartp)`
Wakes up the waiting thread in case of early TX complete.
- `#define _uart_wakeup_tx2_isr(uartp)`
Wakes up the waiting thread in case of late TX complete.
- `#define _uart_wakeup_rx_complete_isr(uartp)`
Wakes up the waiting thread in case of RX complete.
- `#define _uart_wakeup_rx_error_isr(uartp)`
Wakes up the waiting thread in case of RX error.
- `#define _uart_wakeup_rx_timeout_isr(uartp)`
Wakes up the waiting thread in case of RX timeout.
- `#define _uart_tx1_isr_code(uartp)`
Common ISR code for early TX.
- `#define _uart_tx2_isr_code(uartp)`
Common ISR code for late TX.
- `#define _uart_rx_complete_isr_code(uartp)`
Common ISR code for RX complete.
- `#define _uart_rx_error_isr_code(uartp, errors)`
Common ISR code for RX error.
- `#define _uart_rx_idle_code(uartp)`
Common ISR code for RX on idle.
- `#define _uart_timeout_isr_code(uartp)`
Timeout ISR code for receiver.

Enumerations

Functions

- void `uartInit` (void)
UART Driver initialization.
- void `uartObjectInit` (UARTDriver *uartp)
Initializes the standard part of a `UARTDriver` structure.

- void `uartStart` (`UARTDriver` *uartp, const `UARTConfig` *config)
Configures and activates the UART peripheral.
- void `uartStop` (`UARTDriver` *uartp)
Deactivates the UART peripheral.
- void `uartStartSend` (`UARTDriver` *uartp, size_t n, const void *txbuf)
Starts a transmission on the UART peripheral.
- void `uartStartSendI` (`UARTDriver` *uartp, size_t n, const void *txbuf)
Starts a transmission on the UART peripheral.
- size_t `uartStopSend` (`UARTDriver` *uartp)
Stops any ongoing transmission.
- size_t `uartStopSendI` (`UARTDriver` *uartp)
Stops any ongoing transmission.
- void `uartStartReceive` (`UARTDriver` *uartp, size_t n, void *rxbuf)
Starts a receive operation on the UART peripheral.
- void `uartStartReceiveI` (`UARTDriver` *uartp, size_t n, void *rxbuf)
Starts a receive operation on the UART peripheral.
- size_t `uartStopReceive` (`UARTDriver` *uartp)
Stops any ongoing receive operation.
- size_t `uartStopReceiveI` (`UARTDriver` *uartp)
Stops any ongoing receive operation.
- msg_t `uartSendTimeout` (`UARTDriver` *uartp, size_t *np, const void *txbuf, systime_t timeout)
Performs a transmission on the UART peripheral.
- msg_t `uartSendFullTimeout` (`UARTDriver` *uartp, size_t *np, const void *txbuf, systime_t timeout)
Performs a transmission on the UART peripheral.
- msg_t `uartReceiveTimeout` (`UARTDriver` *uartp, size_t *np, void *rxbuf, systime_t timeout)
Performs a receive operation on the UART peripheral.
- void `uartAcquireBus` (`UARTDriver` *uartp)
Gains exclusive access to the UART bus.
- void `uartReleaseBus` (`UARTDriver` *uartp)
Releases exclusive access to the UART bus.

9.89.1 Detailed Description

UART Driver macros and structures.

9.90 hal_uart_lld.c File Reference

PLATFORM UART subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void `uart_lld_init` (void)
Low level UART driver initialization.
- void `uart_lld_start` (`UARTDriver` *uartp)
Configures and activates the UART peripheral.
- void `uart_lld_stop` (`UARTDriver` *uartp)
Deactivates the UART peripheral.

- void [uart_lld_start_send](#) ([UARTDriver](#) *uartp, size_t n, const void *txbuf)
Starts a transmission on the UART peripheral.
- size_t [uart_lld_stop_send](#) ([UARTDriver](#) *uartp)
Stops any ongoing transmission.
- void [uart_lld_start_receive](#) ([UARTDriver](#) *uartp, size_t n, void *rxbuf)
Starts a receive operation on the UART peripheral.
- size_t [uart_lld_stop_receive](#) ([UARTDriver](#) *uartp)
Stops any ongoing receive operation.

Variables

- [UARTDriver](#) [UARTD1](#)
UART1 driver identifier.

9.90.1 Detailed Description

PLATFORM UART subsystem low level driver source.

9.91 [hal_uart_lld.h](#) File Reference

PLATFORM UART subsystem low level driver header.

Data Structures

- struct [UARTConfig](#)
Driver configuration structure.
- struct [UARTDriver](#)
Structure representing an UART driver.

Macros

PLATFORM configuration options

- #define [PLATFORM_UART_USE_UART1](#) FALSE
UART driver enable switch.

Typedefs

- typedef uint32_t [uartflags_t](#)
UART driver condition flags type.
- typedef struct [UARTDriver](#) [UARTDriver](#)
Type of structure representing an UART driver.
- typedef void(* [uartcb_t](#)) ([UARTDriver](#) *uartp)
Generic UART notification callback type.
- typedef void(* [uartccb_t](#)) ([UARTDriver](#) *uartp, uint16_t c)
Character received UART notification callback type.
- typedef void(* [uartecb_t](#)) ([UARTDriver](#) *uartp, [uartflags_t](#) e)
Receive error UART notification callback type.

Functions

- void `uart_lld_init` (void)
Low level UART driver initialization.
- void `uart_lld_start` (UARTDriver *uartp)
Configures and activates the UART peripheral.
- void `uart_lld_stop` (UARTDriver *uartp)
Deactivates the UART peripheral.
- void `uart_lld_start_send` (UARTDriver *uartp, size_t n, const void *txbuf)
Starts a transmission on the UART peripheral.
- size_t `uart_lld_stop_send` (UARTDriver *uartp)
Stops any ongoing transmission.
- void `uart_lld_start_receive` (UARTDriver *uartp, size_t n, void *rxbuf)
Starts a receive operation on the UART peripheral.
- size_t `uart_lld_stop_receive` (UARTDriver *uartp)
Stops any ongoing receive operation.

9.91.1 Detailed Description

PLATFORM UART subsystem low level driver header.

9.92 hal_usb.c File Reference

USB Driver code.

```
#include <string.h>
#include "hal.h"
```

Functions

- static void `set_address` (USBDriver *usbp)
SET ADDRESS transaction callback.
- static bool `default_handler` (USBDriver *usbp)
Standard requests handler.
- void `usbInit` (void)
USB Driver initialization.
- void `usbObjectInit` (USBDriver *usbp)
Initializes the standard part of a USBDriver structure.
- void `usbStart` (USBDriver *usbp, const USBConfig *config)
Configures and activates the USB peripheral.
- void `usbStop` (USBDriver *usbp)
Deactivates the USB peripheral.
- void `usbInitEndpointI` (USBDriver *usbp, usbep_t ep, const USBEndpointConfig *epcp)
Enables an endpoint.
- void `usbDisableEndpointsI` (USBDriver *usbp)
Disables all the active endpoints.
- void `usbStartReceiveI` (USBDriver *usbp, usbep_t ep, uint8_t *buf, size_t n)
Starts a receive transaction on an OUT endpoint.
- void `usbStartTransmitI` (USBDriver *usbp, usbep_t ep, const uint8_t *buf, size_t n)

- Starts a transmit transaction on an IN endpoint.*

 - `msg_t usbReceive (USBDriver *usbp, usbep_t ep, uint8_t *buf, size_t n)`

Performs a receive transaction on an OUT endpoint.

 - `msg_t usbTransmit (USBDriver *usbp, usbep_t ep, const uint8_t *buf, size_t n)`

Performs a transmit transaction on an IN endpoint.

 - `bool usbStallReceive (USBDriver *usbp, usbep_t ep)`

Stalls an OUT endpoint.

 - `bool usbStallTransmit (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.

9.92.1 Detailed Description

USB Driver code.

9.93 hal_usb.h File Reference

USB Driver macros and structures.

```
#include "hal_usb_lld.h"
```

Data Structures

- struct `USBDescriptor`
- Type of an USB descriptor.*

Macros

- `#define USB_USE_WAIT FALSE`
- Enables synchronous APIs.*

Helper macros for USB descriptors

- `#define USB_DESC_INDEX(i) ((uint8_t)(i))`
- Helper macro for index values into descriptor strings.*
- `#define USB_DESC_BYTE(b) ((uint8_t)(b))`
- Helper macro for byte values into descriptor strings.*
- `#define USB_DESC_WORD(w)`
- Helper macro for word values into descriptor strings.*

- `#define USB_DESC_BCD(bcd)`
Helper macro for BCD values into descriptor strings.
- `#define USB_DESC_DEVICE_SIZE 18U`
- `#define USB_DESC_DEVICE(bcdUSB, bDeviceClass, bDeviceSubClass, bDeviceProtocol, bMaxPacketSize, idVendor, idProduct, bcdDevice, iManufacturer, iProduct, iSerialNumber, bNumConfigurations)`
Device Descriptor helper macro.
- `#define USB_DESC_CONFIGURATION_SIZE 9U`
Configuration Descriptor size.
- `#define USB_DESC_CONFIGURATION(wTotalLength, bNumInterfaces, bConfigurationValue, iConfiguration, bmAttributes, bMaxPower)`
Configuration Descriptor helper macro.
- `#define USB_DESC_INTERFACE_SIZE 9U`
Interface Descriptor size.
- `#define USB_DESC_INTERFACE(bInterfaceNumber, bAlternateSetting, bNumEndpoints, bInterfaceClass, bInterfaceSubClass, bInterfaceProtocol, iInterface)`
Interface Descriptor helper macro.
- `#define USB_DESC_INTERFACE_ASSOCIATION_SIZE 8U`
Interface Association Descriptor size.
- `#define USB_DESC_INTERFACE_ASSOCIATION(bFirstInterface, bInterfaceCount, bFunctionClass, bFunctionSubClass, bFunctionProtocol, iInterface)`
Interface Association Descriptor helper macro.
- `#define USB_DESC_ENDPOINT_SIZE 7U`
Endpoint Descriptor size.
- `#define USB_DESC_ENDPOINT(bEndpointAddress, bmAttributes, wMaxPacketSize, bInterval)`
Endpoint Descriptor helper macro.

Endpoint types and settings

- `#define USB_EP_MODE_TYPE 0x0003U`
- `#define USB_EP_MODE_TYPE_CTRL 0x0000U`
- `#define USB_EP_MODE_TYPE_ISOC 0x0001U`
- `#define USB_EP_MODE_TYPE_BULK 0x0002U`
- `#define USB_EP_MODE_TYPE_INTR 0x0003U`

Macro Functions

- `#define usbGetDriverStatel(usbp) ((usbp)->state)`
Returns the driver state.
- `#define usbConnectBus(usbp) usb_ll_connect_bus(usbp)`
Connects the USB device.
- `#define usbDisconnectBus(usbp) usb_ll_disconnect_bus(usbp)`
Disconnect the USB device.
- `#define usbGetFrameNumberX(usbp) usb_ll_get_frame_number(usbp)`
Returns the current frame number.
- `#define usbGetTransmitStatusl(usbp, ep) (((usbp)->transmitting & (uint16_t)((unsigned)1U << (unsigned)(ep))) != 0U)`
Returns the status of an IN endpoint.
- `#define usbGetReceiveStatusl(usbp, ep) (((usbp)->receiving & (uint16_t)((unsigned)1U << (unsigned)(ep))) != 0U)`
Returns the status of an OUT endpoint.
- `#define usbGetReceiveTransactionSizeX(usbp, ep) usb_ll_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_ll_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

Functions

- void `usbInit` (void)
USB Driver initialization.
- void `usbObjectInit` (`USBDriver` *usbp)
Initializes the standard part of a `USBDriver` structure.
- void `usbStart` (`USBDriver` *usbp, const `USBConfig` *config)
Configures and activates the USB peripheral.
- void `usbStop` (`USBDriver` *usbp)
Deactivates the USB peripheral.
- void `usbInitEndpointI` (`USBDriver` *usbp, `usbep_t` ep, const `USBEndpointConfig` *epcp)
Enables an endpoint.
- void `usbDisableEndpointsI` (`USBDriver` *usbp)
Disables all the active endpoints.
- void `usbStartReceiveI` (`USBDriver` *usbp, `usbep_t` ep, uint8_t *buf, size_t n)
Starts a receive transaction on an OUT endpoint.
- void `usbStartTransmitI` (`USBDriver` *usbp, `usbep_t` ep, const uint8_t *buf, size_t n)
Starts a transmit transaction on an IN endpoint.
- msg_t `usbReceive` (`USBDriver` *usbp, `usbep_t` ep, uint8_t *buf, size_t n)
Performs a receive transaction on an OUT endpoint.
- msg_t `usbTransmit` (`USBDriver` *usbp, `usbep_t` ep, const uint8_t *buf, size_t n)

- Performs a transmit transaction on an IN endpoint.*
- bool [usbStallReceive1](#) (USBDriver *usbp, [usbep_t](#) ep)
Stalls an OUT endpoint.
- bool [usbStallTransmit1](#) (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.

9.93.1 Detailed Description

USB Driver macros and structures.

9.94 hal_usb_cdc.h File Reference

USB CDC macros and structures.

Data Structures

- struct [cdc_linecoding_t](#)
Type of Line Coding structure.

Macros

CDC specific messages.

- #define **CDC_SEND_ENCAPSULATED_COMMAND** 0x00U
- #define **CDC_GET_ENCAPSULATED_RESPONSE** 0x01U
- #define **CDC_SET_COMM_FEATURE** 0x02U
- #define **CDC_GET_COMM_FEATURE** 0x03U
- #define **CDC_CLEAR_COMM_FEATURE** 0x04U
- #define **CDC_SET_AUX_LINE_STATE** 0x10U
- #define **CDC_SET_HOOK_STATE** 0x11U
- #define **CDC_PULSE_SETUP** 0x12U
- #define **CDC_SEND_PULSE** 0x13U
- #define **CDC_SET_PULSE_TIME** 0x14U
- #define **CDC_RING_AUX_JACK** 0x15U
- #define **CDC_SET_LINE_CODING** 0x20U
- #define **CDC_GET_LINE_CODING** 0x21U
- #define **CDC_SET_CONTROL_LINE_STATE** 0x22U
- #define **CDC_SEND_BREAK** 0x23U
- #define **CDC_SET_RINGER_PARMS** 0x30U
- #define **CDC_GET_RINGER_PARMS** 0x31U
- #define **CDC_SET_OPERATION_PARMS** 0x32U

- `#define CDC_GET_OPERATION_PARMS 0x33U`

CDC classes

- `#define CDC_COMMUNICATION_INTERFACE_CLASS 0x02U`
- `#define CDC_DATA_INTERFACE_CLASS 0x0AU`

CDC subclasses

- `#define CDC_ABSTRACT_CONTROL_MODEL 0x02U`

CDC descriptors

- `#define CDC_CS_INTERFACE 0x24U`

CDC subdescriptors

- `#define CDC_HEADER 0x00U`
- `#define CDC_CALL_MANAGEMENT 0x01U`
- `#define CDC_ABSTRACT_CONTROL_MANAGEMENT 0x02U`
- `#define CDC_UNION 0x06U`

Line Control bit definitions.

- `#define LC_STOP_1 0U`
- `#define LC_STOP_1P5 1U`
- `#define LC_STOP_2 2U`
- `#define LC_PARITY_NONE 0U`
- `#define LC_PARITY_ODD 1U`
- `#define LC_PARITY_EVEN 2U`
- `#define LC_PARITY_MARK 3U`
- `#define LC_PARITY_SPACE 4U`

9.94.1 Detailed Description

USB CDC macros and structures.

9.95 hal_usb_lld.c File Reference

PLATFORM USB subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void `usb_lld_init` (void)
Low level USB driver initialization.
- void `usb_lld_start` (USBDriver *usbp)
Configures and activates the USB peripheral.
- void `usb_lld_stop` (USBDriver *usbp)
Deactivates the USB peripheral.
- void `usb_lld_reset` (USBDriver *usbp)
USB low level reset routine.
- void `usb_lld_set_address` (USBDriver *usbp)

- Sets the USB address.*
- void `usb_ild_init_endpoint` (USBDriver *usbp, usbep_t ep)
- Enables an endpoint.*
- void `usb_ild_disable_endpoints` (USBDriver *usbp)
- Disables all the active endpoints except the endpoint zero.*
- `usbepstatus_t` `usb_ild_get_status_out` (USBDriver *usbp, usbep_t ep)
- Returns the status of an OUT endpoint.*
- `usbepstatus_t` `usb_ild_get_status_in` (USBDriver *usbp, usbep_t ep)
- Returns the status of an IN endpoint.*
- void `usb_ild_read_setup` (USBDriver *usbp, usbep_t ep, uint8_t *buf)
- Reads a setup packet from the dedicated packet buffer.*
- void `usb_ild_prepare_receive` (USBDriver *usbp, usbep_t ep)
- Prepares for a receive operation.*
- void `usb_ild_prepare_transmit` (USBDriver *usbp, usbep_t ep)
- Prepares for a transmit operation.*
- void `usb_ild_start_out` (USBDriver *usbp, usbep_t ep)
- Starts a receive operation on an OUT endpoint.*
- void `usb_ild_start_in` (USBDriver *usbp, usbep_t ep)
- Starts a transmit operation on an IN endpoint.*
- void `usb_ild_stall_out` (USBDriver *usbp, usbep_t ep)
- Brings an OUT endpoint in the stalled state.*
- void `usb_ild_stall_in` (USBDriver *usbp, usbep_t ep)
- Brings an IN endpoint in the stalled state.*
- void `usb_ild_clear_out` (USBDriver *usbp, usbep_t ep)
- Brings an OUT endpoint in the active state.*
- void `usb_ild_clear_in` (USBDriver *usbp, usbep_t ep)
- Brings an IN endpoint in the active state.*

Variables

- USBDriver `USB1`
- 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.*

9.95.1 Detailed Description

PLATFORM USB subsystem low level driver source.

9.95.2 Variable Documentation

9.95.2.1 USBInEndpointState in

IN EP0 state.

9.95.2.2 USBOutEndpointState out

OUT EP0 state.

9.96 hal_usb_lld.h File Reference

PLATFORM USB subsystem low level driver header.

Data Structures

- struct [USBInEndpointState](#)
Type of an IN endpoint state structure.
- struct [USBOutEndpointState](#)
Type of an OUT endpoint state structure.
- struct [USBEndpointConfig](#)
Type of an USB endpoint configuration structure.
- struct [USBConfig](#)
Type of an USB driver configuration structure.
- struct [USBDriver](#)
Structure representing an USB driver.

Macros

- #define [USB_MAX_ENDPOINTS](#) 4
Maximum endpoint address.
- #define [USB_EP0_STATUS_STAGE](#) USB_EP0_STATUS_STAGE_SW
Status stage handling method.
- #define [USB_SET_ADDRESS_MODE](#) USB_LATE_SET_ADDRESS
The address can be changed immediately upon packet reception.
- #define [USB_SET_ADDRESS_ACK_HANDLING](#) USB_SET_ADDRESS_ACK_SW
Method for set address acknowledge.
- #define [usb_lld_get_frame_number](#)(usbp) 0
Returns the current frame number.
- #define [usb_lld_get_transaction_size](#)(usbp, ep) ((usbp)->epc[ep]->out_state->rxcnt)
Returns the exact size of a receive transaction.
- #define [usb_lld_connect_bus](#)(usbp)
Connects the USB device.
- #define [usb_lld_disconnect_bus](#)(usbp)
Disconnect the USB device.

PLATFORM configuration options

- #define [PLATFORM_USB_USE_USB1](#) FALSE
USB driver enable switch.

Functions

- void `usb_lld_init` (void)
Low level USB driver initialization.
- void `usb_lld_start` (USBDriver *usbp)
Configures and activates the USB peripheral.
- void `usb_lld_stop` (USBDriver *usbp)
Deactivates the USB peripheral.
- void `usb_lld_reset` (USBDriver *usbp)
USB low level reset routine.
- void `usb_lld_set_address` (USBDriver *usbp)
Sets the USB address.
- void `usb_lld_init_endpoint` (USBDriver *usbp, usbep_t ep)
Enables an endpoint.
- void `usb_lld_disable_endpoints` (USBDriver *usbp)
Disables all the active endpoints except the endpoint zero.
- `usbepstatus_t` `usb_lld_get_status_in` (USBDriver *usbp, usbep_t ep)
Returns the status of an IN endpoint.
- `usbepstatus_t` `usb_lld_get_status_out` (USBDriver *usbp, usbep_t ep)
Returns the status of an OUT endpoint.
- void `usb_lld_read_setup` (USBDriver *usbp, usbep_t ep, uint8_t *buf)
Reads a setup packet from the dedicated packet buffer.
- void `usb_lld_prepare_receive` (USBDriver *usbp, usbep_t ep)
Prepares for a receive operation.
- void `usb_lld_prepare_transmit` (USBDriver *usbp, usbep_t ep)
Prepares for a transmit operation.
- void `usb_lld_start_out` (USBDriver *usbp, usbep_t ep)
Starts a receive operation on an OUT endpoint.
- void `usb_lld_start_in` (USBDriver *usbp, usbep_t ep)
Starts a transmit operation on an IN endpoint.
- void `usb_lld_stall_out` (USBDriver *usbp, usbep_t ep)
Brings an OUT endpoint in the stalled state.
- void `usb_lld_stall_in` (USBDriver *usbp, usbep_t ep)
Brings an IN endpoint in the stalled state.
- void `usb_lld_clear_out` (USBDriver *usbp, usbep_t ep)
Brings an OUT endpoint in the active state.
- void `usb_lld_clear_in` (USBDriver *usbp, usbep_t ep)
Brings an IN endpoint in the active state.

9.96.1 Detailed Description

PLATFORM USB subsystem low level driver header.

9.97 hal_wdg.c File Reference

WDG Driver code.

```
#include "hal.h"
```

Functions

- void `wdgInit` (void)
WDG Driver initialization.
- void `wdgStart` (`WDGDriver` *wdgp, const `WDGConfig` *config)
Configures and activates the WDG peripheral.
- void `wdgStop` (`WDGDriver` *wdgp)
Deactivates the WDG peripheral.
- void `wdgReset` (`WDGDriver` *wdgp)
Resets WDG's counter.

9.97.1 Detailed Description

WDG Driver code.

9.98 hal_wdg.h File Reference

WDG Driver macros and structures.

```
#include "hal_wdg_lld.h"
```

Macros

- #define `wdgReset`(wdgp) `wdg_lld_reset`(wdgp)
Resets WDG's counter.

Enumerations

Functions

- void `wdgInit` (void)
WDG Driver initialization.
- void `wdgStart` (`WDGDriver` *wdgp, const `WDGConfig` *config)
Configures and activates the WDG peripheral.
- void `wdgStop` (`WDGDriver` *wdgp)
Deactivates the WDG peripheral.
- void `wdgReset` (`WDGDriver` *wdgp)
Resets WDG's counter.

9.98.1 Detailed Description

WDG Driver macros and structures.

9.99 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_QSPI` TRUE
Enables the QSPI 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.
- #define `HAL_USE_WDG` TRUE
Enables the WDG 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_MMC_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.
- #define `SERIAL_USB_BUFFERS_NUMBER` 2
Serial over USB number of buffers.

SPI driver related setting

- #define `SPI_USE_WAIT` TRUE
Enables synchronous APIs.
- #define `SPI_USE_MUTUAL_EXCLUSION` TRUE
Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.

UART driver related setting

- #define `UART_USE_WAIT` TRUE
Enables synchronous APIs.
- #define `UART_USE_MUTUAL_EXCLUSION` TRUE
Enables the `uartAcquireBus()` and `uartReleaseBus()` APIs.

USB driver related setting

- #define `USB_USE_WAIT` TRUE
Enables synchronous APIs.

9.99.1 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.

Index

- [_CHIBIOS_HAL_](#)
 - [HAL Driver, 90](#)
- [_IOBUS_DATA](#)
 - [PAL Driver, 269](#)
- [_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, 21](#)
- [_base_asynchronous_channel_data](#)
 - [Abstract I/O Channel, 117](#)
- [_base_asynchronous_channel_methods](#)
 - [Abstract I/O Channel, 117](#)
- [_base_block_device_data](#)
 - [Abstract I/O Block Device, 126](#)
- [_base_block_device_methods](#)
 - [Abstract I/O Block Device, 126](#)
- [_base_channel_data](#)
 - [Abstract I/O Channel, 113](#)
- [_base_channel_methods](#)
 - [Abstract I/O Channel, 113](#)
- [_base_sequential_stream_data](#)
 - [Abstract Streams, 148](#)
- [_base_sequential_stream_methods](#)
 - [Abstract Streams, 147](#)
- [_dac_isr_error_code](#)
 - [DAC Driver, 52](#)
- [_dac_isr_full_code](#)
 - [DAC Driver, 51](#)
- [_dac_isr_half_code](#)
 - [DAC Driver, 51](#)
- [_dac_reset_i](#)
 - [DAC Driver, 50](#)
- [_dac_reset_s](#)
 - [DAC Driver, 50](#)
- [_dac_timeout_isr](#)
 - [DAC Driver, 50](#)
- [_dac_wait_s](#)
 - [DAC Driver, 49](#)
- [_dac_wakeup_isr](#)
 - [DAC Driver, 50](#)
- [_file_stream_data](#)
 - [Abstract Files, 120](#)
- [_file_stream_methods](#)
 - [Abstract Files, 120](#)
- [_i2c_wakeup_error_isr](#)
 - [I2C Driver, 153](#)
- [_i2c_wakeup_isr](#)
 - [I2C Driver, 153](#)
- [_i2s_isr_full_code](#)
 - [I2S Driver, 165](#)
- [_i2s_isr_half_code](#)
 - [I2S Driver, 165](#)
- [_icu_isr_invoke_overflow_cb](#)
 - [ICU Driver, 177](#)
- [_icu_isr_invoke_period_cb](#)
 - [ICU Driver, 177](#)
- [_icu_isr_invoke_width_cb](#)
 - [ICU Driver, 177](#)
- [_mmc_driver_methods](#)
 - [MMC over SPI Driver, 232](#)
- [_mmc_sd_block_device_data](#)
 - [MMC/SD Block Device, 257](#)
- [_mmc_sd_block_device_methods](#)
 - [MMC/SD Block Device, 256](#)
- [_mmc_sd_get_capacity](#)
 - [MMC/SD Block Device, 258](#)
- [_mmc_sd_get_capacity_ext](#)
 - [MMC/SD Block Device, 259](#)
- [_mmc_sd_get_slice](#)
 - [MMC/SD Block Device, 258](#)
- [_mmc_sd_unpack_csd_mmc](#)
 - [MMC/SD Block Device, 260](#)
- [_mmc_sd_unpack_csd_v10](#)
 - [MMC/SD Block Device, 261](#)
- [_mmc_sd_unpack_csd_v20](#)
 - [MMC/SD Block Device, 261](#)
- [_mmc_sd_unpack_mmc_cid](#)
 - [MMC/SD Block Device, 260](#)
- [_mmc_sd_unpack_sdc_cid](#)
 - [MMC/SD Block Device, 259](#)
- [_pal_ild_init](#)
 - [PAL Driver, 289](#)
- [_pal_ild_setgroupmode](#)
 - [PAL Driver, 289](#)
- [_qspi_isr_code](#)
 - [QSPI Driver, 317](#)
- [_qspi_wakeup_isr](#)
 - [QSPI Driver, 316](#)

- `_rtc_driver_methods`
 - RTC Driver, [329](#)
- `_sdc_driver_methods`
 - SDC Driver, [342](#)
- `_sdc_wait_for_transfer_state`
 - SDC Driver, [350](#)
- `_serial_driver_data`
 - Serial Driver, [371](#)
- `_serial_driver_methods`
 - Serial Driver, [368](#)
- `_serial_usb_driver_data`
 - Serial over USB Driver, [382](#)
- `_serial_usb_driver_methods`
 - Serial over USB Driver, [382](#)
- `_spi_isr_code`
 - SPI Driver, [398](#)
- `_spi_wakeup_isr`
 - SPI Driver, [398](#)
- `_uart_rx_complete_isr_code`
 - UART Driver, [426](#)
- `_uart_rx_error_isr_code`
 - UART Driver, [427](#)
- `_uart_rx_idle_code`
 - UART Driver, [427](#)
- `_uart_timeout_isr_code`
 - UART Driver, [428](#)
- `_uart_tx1_isr_code`
 - UART Driver, [425](#)
- `_uart_tx2_isr_code`
 - UART Driver, [425](#)
- `_uart_wakeup_rx_complete_isr`
 - UART Driver, [424](#)
- `_uart_wakeup_rx_error_isr`
 - UART Driver, [424](#)
- `_uart_wakeup_rx_timeout_isr`
 - UART Driver, [424](#)
- `_uart_wakeup_tx1_isr`
 - UART Driver, [423](#)
- `_uart_wakeup_tx2_isr`
 - UART Driver, [423](#)
- `_usb_ep0in`
 - USB Driver, [473](#)
- `_usb_ep0out`
 - USB Driver, [474](#)
- `_usb_ep0setup`
 - USB Driver, [472](#)
- `_usb_isr_invoke_event_cb`
 - USB Driver, [457](#)
- `_usb_isr_invoke_in_cb`
 - USB Driver, [458](#)
- `_usb_isr_invoke_out_cb`
 - USB Driver, [458](#)
- `_usb_isr_invoke_setup_cb`
 - USB Driver, [458](#)
- `_usb_isr_invoke_sof_cb`
 - USB Driver, [457](#)
- `_usb_reset`
 - USB Driver, [471](#)
- `_usb_suspend`
 - USB Driver, [471](#)
- `_usb_wakeup`
 - USB Driver, [472](#)
- 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`, [21](#)
 - 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, [32](#)
 - ADCDriver, [24](#)
 - `adc_channels_num_t`, [24](#)
 - `adc_ild_init`, [31](#)
 - `adc_ild_start`, [31](#)
 - `adc_ild_start_conversion`, [32](#)
 - `adc_ild_stop`, [31](#)
 - `adc_ild_stop_conversion`, [32](#)
 - `adcAcquireBus`, [30](#)
 - `adcConvert`, [29](#)
 - `adclnit`, [25](#)
 - `adcObjectInit`, [25](#)
 - `adcReleaseBus`, [30](#)
 - `adcStart`, [26](#)
 - `adcStartConversion`, [27](#)
 - `adcStartConversionI`, [27](#)
 - `adcStop`, [26](#)
 - `adcStopConversion`, [28](#)
 - `adcStopConversionI`, [29](#)
 - `adccallback_t`, [24](#)
 - `adcerror_t`, [25](#)
 - `adcerrorcallback_t`, [24](#)
 - `adcsample_t`, [24](#)
 - `adcstate_t`, [24](#)
 - 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, [210](#)
- ADC_USE_WAIT
 - ADC Driver, [21](#)
 - Configuration, [210](#)
- ADCCfg, [489](#)
- ADCConversionGroup, [490](#)
 - circular, [491](#)
 - end_cb, [491](#)
 - error_cb, [491](#)
 - num_channels, [491](#)
- ADCD1
 - ADC Driver, [32](#)
- ADCDriver, [491](#)
 - ADC Driver, [24](#)
 - config, [493](#)
 - depth, [493](#)
 - grpp, [493](#)
 - mutex, [493](#)
 - samples, [493](#)
 - state, [493](#)
 - thread, [493](#)
- ADVERTISE_100BASE4
 - MII/RMII Header, [226](#)
- ADVERTISE_100FULL
 - MII/RMII Header, [226](#)
- ADVERTISE_100HALF
 - MII/RMII Header, [226](#)
- ADVERTISE_10FULL
 - MII/RMII Header, [226](#)
- ADVERTISE_10HALF
 - MII/RMII Header, [226](#)
- ADVERTISE_CSMA
 - MII/RMII Header, [226](#)
- ADVERTISE_LPACK
 - MII/RMII Header, [227](#)
- ADVERTISE_NPAGE
 - MII/RMII Header, [227](#)
- ADVERTISE_PAUSE_ASYM
 - MII/RMII Header, [227](#)
- ADVERTISE_PAUSE_CAP
 - MII/RMII Header, [226](#)
- ADVERTISE_RESV
 - MII/RMII Header, [227](#)
- ADVERTISE_RFAULT
 - MII/RMII Header, [227](#)
- ADVERTISE_SLCT
 - MII/RMII Header, [226](#)
- Abstract Files, [119](#)
 - _file_stream_data, [120](#)
 - _file_stream_methods, [120](#)
 - FILE_EOF, [120](#)
 - FILE_ERROR, [120](#)
 - FILE_OK, [120](#)
 - fileStreamClose, [122](#)
 - fileStreamGet, [122](#)
 - fileStreamGetError, [123](#)
 - fileStreamGetPosition, [123](#)
 - fileStreamGetSize, [123](#)
 - fileStreamPut, [121](#)
 - fileStreamRead, [121](#)
 - fileStreamSeek, [124](#)
 - fileStreamWrite, [120](#)
 - fileoffset_t, [124](#)
- Abstract I/O Block Device, [125](#)
 - _base_block_device_data, [126](#)
 - _base_block_device_methods, [126](#)
 - BLK_ACTIVE, [131](#)
 - BLK_CONNECTING, [131](#)
 - BLK_DISCONNECTING, [131](#)
 - BLK_READING, [131](#)
 - BLK_READY, [131](#)
 - BLK_STOP, [131](#)
 - BLK_SYNCING, [131](#)
 - BLK_UNINIT, [131](#)
 - BLK_WRITING, [131](#)
 - blkConnect, [128](#)
 - blkDisconnect, [129](#)
 - blkGetDriverState, [126](#)
 - blkGetInfo, [131](#)
 - blkIsInserted, [127](#)
 - blkIsTransferring, [127](#)
 - blkIsWriteProtected, [128](#)
 - blkRead, [129](#)
 - blkSync, [130](#)
 - blkWrite, [130](#)
 - blkstate_t, [131](#)
- Abstract I/O Channel, [112](#)
 - _base_asynchronous_channel_data, [117](#)
 - _base_asynchronous_channel_methods, [117](#)
 - _base_channel_data, [113](#)
 - _base_channel_methods, [113](#)
 - CHN_CONNECTED, [116](#)
 - CHN_DISCONNECTED, [116](#)
 - CHN_INPUT_AVAILABLE, [116](#)
 - CHN_NO_ERROR, [116](#)
 - CHN_OUTPUT_EMPTY, [116](#)
 - CHN_TRANSMISSION_END, [117](#)
 - chnAddFlagsI, [117](#)
 - chnGetEventSource, [117](#)
 - chnGetTimeout, [114](#)
 - chnPutTimeout, [113](#)
 - chnRead, [115](#)
 - chnReadTimeout, [116](#)
 - chnWrite, [114](#)
 - chnWriteTimeout, [115](#)
- Abstract NOR Flash Class, [73](#)

- Abstract Streams, [147](#)
 - [_base_sequential_stream_data](#), [148](#)
 - [_base_sequential_stream_methods](#), [147](#)
 - [streamGet](#), [149](#)
 - [streamPut](#), [148](#)
 - [streamRead](#), [148](#)
 - [streamWrite](#), [148](#)
- [adc_channels_num_t](#)
 - ADC Driver, [24](#)
- [adc_lld_init](#)
 - ADC Driver, [31](#)
- [adc_lld_start](#)
 - ADC Driver, [31](#)
- [adc_lld_start_conversion](#)
 - ADC Driver, [32](#)
- [adc_lld_stop](#)
 - ADC Driver, [31](#)
- [adc_lld_stop_conversion](#)
 - ADC Driver, [32](#)
- [adcAcquireBus](#)
 - ADC Driver, [30](#)
- [adcConvert](#)
 - ADC Driver, [29](#)
- [adclnit](#)
 - ADC Driver, [25](#)
- [adcObjectInit](#)
 - ADC Driver, [25](#)
- [adcReleaseBus](#)
 - ADC Driver, [30](#)
- [adcStart](#)
 - ADC Driver, [26](#)
- [adcStartConversion](#)
 - ADC Driver, [27](#)
- [adcStartConversionI](#)
 - ADC Driver, [27](#)
- [adcStop](#)
 - ADC Driver, [26](#)
- [adcStopConversion](#)
 - ADC Driver, [28](#)
- [adcStopConversionI](#)
 - ADC Driver, [29](#)
- [adccallback_t](#)
 - ADC Driver, [24](#)
- [adcerror_t](#)
 - ADC Driver, [25](#)
- [adcerrorcallback_t](#)
 - ADC Driver, [24](#)
- [adcsample_t](#)
 - ADC Driver, [24](#)
- [adcstate_t](#)
 - ADC Driver, [24](#)
- [address](#)
 - USB Driver, [590](#)
- BLK_ACTIVE
 - Abstract I/O Block Device, [131](#)
- BLK_CONNECTING
 - Abstract I/O Block Device, [131](#)
- BLK_DISCONNECTING
 - Abstract I/O Block Device, [131](#)
- BLK_READING
 - Abstract I/O Block Device, [131](#)
- BLK_READY
 - Abstract I/O Block Device, [131](#)
- BLK_STOP
 - Abstract I/O Block Device, [131](#)
- BLK_SYNCING
 - Abstract I/O Block Device, [131](#)
- BLK_UNINIT
 - Abstract I/O Block Device, [131](#)
- BLK_WRITING
 - Abstract I/O Block Device, [131](#)
- BMCR_ANENABLE
 - MII/RMII Header, [224](#)
- BMCR_ANRESTART
 - MII/RMII Header, [224](#)
- BMCR_CTST
 - MII/RMII Header, [224](#)
- BMCR_FULLDPLX
 - MII/RMII Header, [224](#)
- BMCR_ISOLATE
 - MII/RMII Header, [224](#)
- BMCR_LOOPBACK
 - MII/RMII Header, [225](#)
- BMCR_PDOWN
 - MII/RMII Header, [224](#)
- BMCR_RESET
 - MII/RMII Header, [225](#)
- BMCR_RESV
 - MII/RMII Header, [224](#)
- BMCR_SPEED100
 - MII/RMII Header, [225](#)
- BMSR_100BASE4
 - MII/RMII Header, [226](#)
- BMSR_100FULL
 - MII/RMII Header, [226](#)
- BMSR_100HALF
 - MII/RMII Header, [226](#)
- BMSR_10FULL
 - MII/RMII Header, [226](#)
- BMSR_10HALF
 - MII/RMII Header, [225](#)
- BMSR_ANEGCAPABLE
 - MII/RMII Header, [225](#)
- BMSR_ANEGCOMPLETE
 - MII/RMII Header, [225](#)
- BMSR_ERCAP
 - MII/RMII Header, [225](#)
- BMSR_JCD
 - MII/RMII Header, [225](#)
- BMSR_LSTATUS
 - MII/RMII Header, [225](#)
- BMSR_MFPRESUPPCAP
 - MII/RMII Header, [225](#)
- BMSR_RESV
 - MII/RMII Header, [225](#)
- BMSR_RFAULT

- MII/RMII Header, 225
- BQ_BUFFER_SIZE
 - I/O Buffers Queues, 94
- BaseAsynchronousChannel, 493
 - vmt, 495
- BaseAsynchronousChannelVMT, 495
- BaseBlockDevice, 497
 - vmt, 499
- BaseBlockDeviceVMT, 499
- BaseChannel, 500
 - vmt, 502
- BaseChannelVMT, 502
- BaseSequentialStream, 504
 - vmt, 505
- BaseSequentialStreamVMT, 505
- bcounter
 - io_buffers_queue, 535
- blk_num
 - BlockDeviceInfo, 507
- blk_size
 - BlockDeviceInfo, 507
- blkConnect
 - Abstract I/O Block Device, 128
- blkDisconnect
 - Abstract I/O Block Device, 129
- blkGetDriverState
 - Abstract I/O Block Device, 126
- blkGetInfo
 - Abstract I/O Block Device, 131
- blkIsInserted
 - Abstract I/O Block Device, 127
- blkIsTransferring
 - Abstract I/O Block Device, 127
- blkIsWriteProtected
 - Abstract I/O Block Device, 128
- blkRead
 - Abstract I/O Block Device, 129
- blkSync
 - Abstract I/O Block Device, 130
- blkWrite
 - Abstract I/O Block Device, 130
- blkstate_t
 - Abstract I/O Block Device, 131
- BlockDeviceInfo, 506
 - blk_num, 507
 - blk_size, 507
- bn
 - io_buffers_queue, 535
- bqGetLinkX
 - I/O Buffers Queues, 94
- bqIsSuspendedX
 - I/O Buffers Queues, 95
- bqResumeX
 - I/O Buffers Queues, 96
- bqSizeX
 - I/O Buffers Queues, 94
- bqSpacel
 - I/O Buffers Queues, 94
- bqSuspendI
 - I/O Buffers Queues, 95
- bqnotify_t
 - I/O Buffers Queues, 98
- brdptr
 - io_buffers_queue, 535
- bsize
 - io_buffers_queue, 535
- btop
 - io_buffers_queue, 535
- buffers
 - io_buffers_queue, 535
- bulk_in
 - SerialUSBConfig, 572
- bulk_out
 - SerialUSBConfig, 572
- bus_width
 - SDCConfig, 564
- bwrptr
 - io_buffers_queue, 535
- CAN Driver, 33
 - CAN_ANY_MAILBOX, 36
 - CAN_BUS_OFF_ERROR, 36
 - CAN_FRAMING_ERROR, 36
 - CAN_LIMIT_ERROR, 36
 - CAN_LIMIT_WARNING, 36
 - CAN_MAILBOX_TO_MASK, 36
 - CAN_OVERFLOW_ERROR, 36
 - CAN_READY, 37
 - CAN_RX_MAILBOXES, 37
 - CAN_SLEEP, 37
 - CAN_STARTING, 37
 - CAN_STOP, 37
 - CAN_TX_MAILBOXES, 36
 - CAN_UNINIT, 37
 - CAN_USE_SLEEP_MODE, 36
 - CAND1, 46
 - can_llid_init, 43
 - can_llid_is_rx_nonempty, 45
 - can_llid_is_tx_empty, 44
 - can_llid_receive, 45
 - can_llid_sleep, 46
 - can_llid_start, 44
 - can_llid_stop, 44
 - can_llid_transmit, 45
 - can_llid_wakeup, 46
 - canInit, 37
 - canObjectInit, 38
 - canReceive, 36
 - canReceiveTimeout, 41
 - canSleep, 42
 - canStart, 38
 - canStop, 39
 - canTransmit, 36
 - canTransmitTimeout, 41
 - canTryReceiveI, 40
 - canTryTransmitI, 39
 - canWakeup, 43

- canmbx_t, [37](#)
- canstate_t, [37](#)
- PLATFORM_CAN_USE_CAN1, [37](#)
- CAN_ANY_MAILBOX
 - CAN Driver, [36](#)
- CAN_BUS_OFF_ERROR
 - CAN Driver, [36](#)
- CAN_FRAMING_ERROR
 - CAN Driver, [36](#)
- CAN_LIMIT_ERROR
 - CAN Driver, [36](#)
- CAN_LIMIT_WARNING
 - CAN Driver, [36](#)
- CAN_MAILBOX_TO_MASK
 - CAN Driver, [36](#)
- CAN_OVERFLOW_ERROR
 - CAN Driver, [36](#)
- CAN_READY
 - CAN Driver, [37](#)
- CAN_RX_MAILBOXES
 - CAN Driver, [37](#)
- CAN_SLEEP
 - CAN Driver, [37](#)
- CAN_STARTING
 - CAN Driver, [37](#)
- CAN_STOP
 - CAN Driver, [37](#)
- CAN_TX_MAILBOXES
 - CAN Driver, [36](#)
- CAN_UNINIT
 - CAN Driver, [37](#)
- CAN_USE_SLEEP_MODE
 - CAN Driver, [36](#)
 - Configuration, [210](#)
- CANConfig, [507](#)
- CAND1
 - CAN Driver, [46](#)
- CANDriver, [508](#)
 - config, [509](#)
 - error_event, [510](#)
 - rxfull_event, [509](#)
 - rxqueue, [509](#)
 - sleep_event, [510](#)
 - state, [509](#)
 - txempty_event, [509](#)
 - txqueue, [509](#)
 - wakeup_event, [510](#)
- CANRxFrame, [510](#)
 - DLC, [511](#)
 - data16, [512](#)
 - data32, [512](#)
 - data8, [512](#)
 - EID, [512](#)
 - FMI, [511](#)
 - IDE, [511](#)
 - RTR, [511](#)
 - SID, [512](#)
 - TIME, [511](#)
- CANTxFrame, [512](#)
 - DLC, [513](#)
 - data16, [514](#)
 - data32, [514](#)
 - data8, [513](#)
 - EID, [513](#)
 - IDE, [513](#)
 - RTR, [513](#)
 - SID, [513](#)
- CH_HAL_MAJOR
 - HAL Driver, [90](#)
- CH_HAL_MINOR
 - HAL Driver, [90](#)
- CH_HAL_PATCH
 - HAL Driver, [90](#)
- CH_HAL_STABLE
 - HAL Driver, [90](#)
- CHN_CONNECTED
 - Abstract I/O Channel, [116](#)
- CHN_DISCONNECTED
 - Abstract I/O Channel, [116](#)
- CHN_INPUT_AVAILABLE
 - Abstract I/O Channel, [116](#)
- CHN_NO_ERROR
 - Abstract I/O Channel, [116](#)
- CHN_OUTPUT_EMPTY
 - Abstract I/O Channel, [116](#)
- CHN_TRANSMISSION_END
 - Abstract I/O Channel, [117](#)
- callback
 - GPTConfig, [525](#)
 - PWMChannelConfig, [552](#)
 - PWMConfig, [554](#)
- can_lld_init
 - CAN Driver, [43](#)
- can_lld_is_rx_nonempty
 - CAN Driver, [45](#)
- can_lld_is_tx_empty
 - CAN Driver, [44](#)
- can_lld_receive
 - CAN Driver, [45](#)
- can_lld_sleep
 - CAN Driver, [46](#)
- can_lld_start
 - CAN Driver, [44](#)
- can_lld_stop
 - CAN Driver, [44](#)
- can_lld_transmit
 - CAN Driver, [45](#)
- can_lld_wakeup
 - CAN Driver, [46](#)
- canInit
 - CAN Driver, [37](#)
- canObjectInit
 - CAN Driver, [38](#)
- canReceive
 - CAN Driver, [36](#)
- canReceiveTimeout

- CAN Driver, [41](#)
- canSleep
 - CAN Driver, [42](#)
- canStart
 - CAN Driver, [38](#)
- canStop
 - CAN Driver, [39](#)
- canTransmit
 - CAN Driver, [36](#)
- canTransmitTimeout
 - CAN Driver, [41](#)
- canTryReceive
 - CAN Driver, [40](#)
- canTryTransmit
 - CAN Driver, [39](#)
- canWakeup
 - CAN Driver, [43](#)
- canmbx_t
 - CAN Driver, [37](#)
- canstate_t
 - CAN Driver, [37](#)
- cardmode
 - SDCDriver, [566](#)
- cb
 - EXTChannelConfig, [519](#)
- cdc_linecoding_t, [514](#)
- channels
 - EXTConfig, [520](#)
 - PWMConfig, [554](#)
 - PWMDriver, [556](#)
- chnAddFlags
 - Abstract I/O Channel, [117](#)
- chnGetEventSource
 - Abstract I/O Channel, [117](#)
- chnGetTimeout
 - Abstract I/O Channel, [114](#)
- chnPutTimeout
 - Abstract I/O Channel, [113](#)
- chnRead
 - Abstract I/O Channel, [115](#)
- chnReadTimeout
 - Abstract I/O Channel, [116](#)
- chnWrite
 - Abstract I/O Channel, [114](#)
- chnWriteTimeout
 - Abstract I/O Channel, [115](#)
- circular
 - ADCConversionGroup, [491](#)
- Complex Drivers, [214](#)
- config
 - ADCDriver, [493](#)
 - CANDriver, [509](#)
 - DACDriver, [518](#)
 - EXTDriver, [521](#)
 - GPTDriver, [526](#)
 - I2CDriver, [528](#)
 - I2SDriver, [531](#)
 - ICUDriver, [533](#)
 - MACDriver, [540](#)
 - MMCDriver, [545](#)
 - PWMDriver, [556](#)
 - QSPIDriver, [558](#)
 - SDCDriver, [565](#)
 - SPIDriver, [577](#)
 - UARTDriver, [581](#)
 - USBDriver, [589](#)
 - WDGDriver, [597](#)
- Configuration, [206](#)
 - ADC_USE_MUTUAL_EXCLUSION, [210](#)
 - ADC_USE_WAIT, [210](#)
 - CAN_USE_SLEEP_MODE, [210](#)
 - HAL_USE_ADC, [208](#)
 - HAL_USE_CAN, [208](#)
 - HAL_USE_DAC, [208](#)
 - HAL_USE_EXT, [208](#)
 - HAL_USE_GPT, [208](#)
 - HAL_USE_I2C, [209](#)
 - HAL_USE_I2S, [209](#)
 - HAL_USE_ICU, [209](#)
 - HAL_USE_MAC, [209](#)
 - HAL_USE_MMC_SPI, [209](#)
 - HAL_USE_PAL, [208](#)
 - HAL_USE_PWM, [209](#)
 - HAL_USE_QSPI, [209](#)
 - HAL_USE_RTC, [209](#)
 - HAL_USE_SDC, [209](#)
 - HAL_USE_SERIAL_USB, [209](#)
 - HAL_USE_SERIAL, [209](#)
 - HAL_USE_SPI, [209](#)
 - HAL_USE_UART, [210](#)
 - HAL_USE_USB, [210](#)
 - HAL_USE_WDG, [210](#)
 - I2C_USE_MUTUAL_EXCLUSION, [210](#)
 - MAC_USE_EVENTS, [210](#)
 - MAC_USE_ZERO_COPY, [210](#)
 - MMC_NICE_WAITING, [210](#)
 - SDC_INIT_RETRY, [211](#)
 - SDC_MMC_SUPPORT, [211](#)
 - SDC_NICE_WAITING, [211](#)
 - SERIAL_BUFFERS_SIZE, [211](#)
 - SERIAL_DEFAULT_BITRATE, [211](#)
 - SERIAL_USB_BUFFERS_NUMBER, [211](#)
 - SERIAL_USB_BUFFERS_SIZE, [211](#)
 - SPI_USE_MUTUAL_EXCLUSION, [212](#)
 - SPI_USE_WAIT, [212](#)
 - UART_USE_MUTUAL_EXCLUSION, [212](#)
 - UART_USE_WAIT, [212](#)
 - USB_USE_WAIT, [212](#)
- configuration
 - USBDriver, [590](#)
- crc7
 - MMC over SPI Driver, [233](#)
- crc7_lookup_table
 - MMC over SPI Driver, [250](#)
- DAC Driver, [47](#)
 - _dac_isr_error_code, [52](#)

- [_dac_isr_full_code](#), 51
- [_dac_isr_half_code](#), 51
- [_dac_reset_i](#), 50
- [_dac_reset_s](#), 50
- [_dac_timeout_isr](#), 50
- [_dac_wait_s](#), 49
- [_dac_wakeup_isr](#), 50
- [DAC_ACTIVE](#), 54
- [DAC_COMPLETE](#), 54
- [DAC_ERR_DMAFAILURE](#), 54
- [DAC_ERR_UNDERFLOW](#), 54
- [DAC_ERROR](#), 54
- [DAC_MAX_CHANNELS](#), 53
- [DAC_READY](#), 54
- [DAC_STOP](#), 54
- [DAC_UNINIT](#), 54
- [DAC_USE_MUTUAL_EXCLUSION](#), 49
- [DAC_USE_WAIT](#), 49
- [DACD1](#), 62
- [DACDriver](#), 53
- [dac_ild_init](#), 60
- [dac_ild_put_channel](#), 61
- [dac_ild_start](#), 61
- [dac_ild_start_conversion](#), 61
- [dac_ild_stop](#), 61
- [dac_ild_stop_conversion](#), 62
- [dacAcquireBus](#), 60
- [dacConvert](#), 59
- [dacInit](#), 54
- [dacObjectInit](#), 55
- [dacPutChannelX](#), 56
- [dacReleaseBus](#), 60
- [dacStart](#), 55
- [dacStartConversion](#), 56
- [dacStartConversionI](#), 57
- [dacStop](#), 55
- [dacStopConversion](#), 58
- [dacStopConversionI](#), 58
- [daccallback_t](#), 53
- [dacchannel_t](#), 53
- [dacerror_t](#), 54
- [dacerrorcallback_t](#), 53
- [dacsamples_t](#), 53
- [dacstate_t](#), 54
- [PLATFORM_DAC_USE_DAC1](#), 53
- [DAC_ACTIVE](#)
 - [DAC Driver](#), 54
- [DAC_COMPLETE](#)
 - [DAC Driver](#), 54
- [DAC_ERR_DMAFAILURE](#)
 - [DAC Driver](#), 54
- [DAC_ERR_UNDERFLOW](#)
 - [DAC Driver](#), 54
- [DAC_ERROR](#)
 - [DAC Driver](#), 54
- [DAC_MAX_CHANNELS](#)
 - [DAC Driver](#), 53
- [DAC_READY](#)
 - [DAC Driver](#), 54
- [DAC_STOP](#)
 - [DAC Driver](#), 54
- [DAC_UNINIT](#)
 - [DAC Driver](#), 54
- [DAC_USE_MUTUAL_EXCLUSION](#)
 - [DAC Driver](#), 49
- [DAC_USE_WAIT](#)
 - [DAC Driver](#), 49
- [DACConfig](#), 514
- [DACConversionGroup](#), 515
 - [end_cb](#), 516
 - [error_cb](#), 516
 - [num_channels](#), 516
- [DACD1](#)
 - [DAC Driver](#), 62
- [DACDriver](#), 516
 - [config](#), 518
 - [DAC Driver](#), 53
 - [depth](#), 518
 - [grpp](#), 518
 - [mutex](#), 518
 - [samples](#), 518
 - [state](#), 518
 - [thread](#), 518
- [DLC](#)
 - [CANRxFrame](#), 511
 - [CANTxFrame](#), 513
- [dac_ild_init](#)
 - [DAC Driver](#), 60
- [dac_ild_put_channel](#)
 - [DAC Driver](#), 61
- [dac_ild_start](#)
 - [DAC Driver](#), 61
- [dac_ild_start_conversion](#)
 - [DAC Driver](#), 61
- [dac_ild_stop](#)
 - [DAC Driver](#), 61
- [dac_ild_stop_conversion](#)
 - [DAC Driver](#), 62
- [dacAcquireBus](#)
 - [DAC Driver](#), 60
- [dacConvert](#)
 - [DAC Driver](#), 59
- [dacInit](#)
 - [DAC Driver](#), 54
- [dacObjectInit](#)
 - [DAC Driver](#), 55
- [dacPutChannelX](#)
 - [DAC Driver](#), 56
- [dacReleaseBus](#)
 - [DAC Driver](#), 60
- [dacStart](#)
 - [DAC Driver](#), 55
- [dacStartConversion](#)
 - [DAC Driver](#), 56
- [dacStartConversionI](#)
 - [DAC Driver](#), 57

- dacStop
 - DAC Driver, [55](#)
- dacStopConversion
 - DAC Driver, [58](#)
- dacStopConversionI
 - DAC Driver, [58](#)
- daccallback_t
 - DAC Driver, [53](#)
- dacchannel_t
 - DAC Driver, [53](#)
- dacerror_t
 - DAC Driver, [54](#)
- dacerrorcallback_t
 - DAC Driver, [53](#)
- dacsample_t
 - DAC Driver, [53](#)
- dacstate_t
 - DAC Driver, [54](#)
- data16
 - CANRxFrame, [512](#)
 - CANTxFrame, [514](#)
- data32
 - CANRxFrame, [512](#)
 - CANTxFrame, [514](#)
- data8
 - CANRxFrame, [512](#)
 - CANTxFrame, [513](#)
- day
 - RTCDateTime, [560](#)
- dayofweek
 - RTCDateTime, [560](#)
- default_config
 - Serial Driver, [379](#)
- default_handler
 - USB Driver, [463](#)
- depth
 - ADCDriver, [493](#)
 - DACDriver, [518](#)
- detect_bus_clk
 - SDC Driver, [348](#)
- dstflag
 - RTCDateTime, [560](#)
- EID
 - CANRxFrame, [512](#)
 - CANTxFrame, [513](#)
- EP_STATUS_ACTIVE
 - USB Driver, [462](#)
- EP_STATUS_DISABLED
 - USB Driver, [462](#)
- EP_STATUS_STALLED
 - USB Driver, [462](#)
- ETHD1
 - MAC Driver, [202](#)
- EXPANSION_ENABLENPAGE
 - MII/RMII Header, [228](#)
- EXPANSION_LCWP
 - MII/RMII Header, [228](#)
- EXPANSION_MFAULTS
 - MII/RMII Header, [228](#)
- EXPANSION_NPCAPABLE
 - MII/RMII Header, [228](#)
- EXPANSION_NWAY
 - MII/RMII Header, [228](#)
- EXPANSION_RESV
 - MII/RMII Header, [228](#)
- EXT Driver, [63](#)
 - EXT_ACTIVE, [68](#)
 - EXT_CH_MODE_AUTOSTART, [66](#)
 - EXT_CH_MODE_BOTH_EDGES, [65](#)
 - EXT_CH_MODE_DISABLED, [65](#)
 - EXT_CH_MODE_EDGES_MASK, [65](#)
 - EXT_CH_MODE_FALLING_EDGE, [65](#)
 - EXT_CH_MODE_LOW_LEVEL, [66](#)
 - EXT_CH_MODE_RISING_EDGE, [65](#)
 - EXT_MAX_CHANNELS, [67](#)
 - EXT_STOP, [68](#)
 - EXT_UNINIT, [68](#)
 - EXTD1, [72](#)
 - EXTDriver, [67](#)
 - expchannel_t, [67](#)
 - ext_llc_channel_disable, [72](#)
 - ext_llc_channel_enable, [72](#)
 - ext_llc_init, [71](#)
 - ext_llc_start, [71](#)
 - ext_llc_stop, [71](#)
 - extChannelDisable, [70](#)
 - extChannelDisableI, [66](#)
 - extChannelEnable, [69](#)
 - extChannelEnableI, [66](#)
 - extInit, [68](#)
 - extObjectInit, [68](#)
 - extSetChannelMode, [66](#)
 - extSetChannelModel, [70](#)
 - extStart, [68](#)
 - extStop, [69](#)
 - extcallback_t, [67](#)
 - extstate_t, [67](#)
 - PLATFORM_EXT_USE_EXT1, [67](#)
- EXT_ACTIVE
 - EXT Driver, [68](#)
- EXT_CH_MODE_AUTOSTART
 - EXT Driver, [66](#)
- EXT_CH_MODE_BOTH_EDGES
 - EXT Driver, [65](#)
- EXT_CH_MODE_DISABLED
 - EXT Driver, [65](#)
- EXT_CH_MODE_EDGES_MASK
 - EXT Driver, [65](#)
- EXT_CH_MODE_FALLING_EDGE
 - EXT Driver, [65](#)
- EXT_CH_MODE_LOW_LEVEL
 - EXT Driver, [66](#)
- EXT_CH_MODE_RISING_EDGE
 - EXT Driver, [65](#)
- EXT_MAX_CHANNELS
 - EXT Driver, [67](#)

- EXT_STOP
 - EXT Driver, 68
- EXT_UNINIT
 - EXT Driver, 68
- EXTChannelConfig, 518
 - cb, 519
 - mode, 519
- EXTConfig, 520
 - channels, 520
- EXTD1
 - EXT Driver, 72
- EXTDriver, 521
 - config, 521
 - EXT Driver, 67
 - state, 521
- early
 - UARTDriver, 581
- enabled
 - PWMDriver, 556
- end_cb
 - ADCCConversionGroup, 491
 - DACConversionGroup, 516
 - I2SConfig, 530
 - QSPICongfig, 557
 - SPICongfig, 576
- ep0_state
 - USB Driver, 480
- ep0config
 - USB Driver, 480
- ep0endcb
 - USBDriver, 590
- ep0n
 - USBDriver, 590
- ep0next
 - USBDriver, 590
- ep0state
 - USBDriver, 590
- ep_mode
 - USBEndpointConfig, 593
- epc
 - USBDriver, 589
- error_cb
 - ADCCConversionGroup, 491
 - DACConversionGroup, 516
- error_event
 - CANDriver, 510
- errors
 - I2CDriver, 528
 - SDCDriver, 566
- event_cb
 - USBConfig, 587
- expchannel_t
 - EXT Driver, 67
- ext_llid_channel_disable
 - EXT Driver, 72
- ext_llid_channel_enable
 - EXT Driver, 72
- ext_llid_init
 - EXT Driver, 71
- ext_llid_start
 - EXT Driver, 71
- ext_llid_stop
 - EXT Driver, 71
- extChannelDisable
 - EXT Driver, 70
- extChannelDisableI
 - EXT Driver, 66
- extChannelEnable
 - EXT Driver, 69
- extChannelEnableI
 - EXT Driver, 66
- extInit
 - EXT Driver, 68
- extObjectInit
 - EXT Driver, 68
- extSetChannelMode
 - EXT Driver, 66
- extSetChannelModel
 - EXT Driver, 70
- extStart
 - EXT Driver, 68
- extStop
 - EXT Driver, 69
- extcallback_t
 - EXT Driver, 67
- extstate_t
 - EXT Driver, 67
- FILE_EOF
 - Abstract Files, 120
- FILE_ERROR
 - Abstract Files, 120
- FILE_OK
 - Abstract Files, 120
- FMI
 - CANRxFrame, 511
- FileStream, 522
 - vmt, 523
- fileStreamClose
 - Abstract Files, 122
- fileStreamGet
 - Abstract Files, 122
- fileStreamGetError
 - Abstract Files, 123
- fileStreamGetPosition
 - Abstract Files, 123
- fileStreamGetSize
 - Abstract Files, 123
- fileStreamPut
 - Abstract Files, 121
- fileStreamRead
 - Abstract Files, 121
- fileStreamSeek
 - Abstract Files, 124
- FileStreamVMT, 523
- fileStreamWrite
 - Abstract Files, 120

- fileoffset_t
 - Abstract Files, 124
- frequency
 - GPTConfig, 525
 - ICUConfig, 532
 - PWMConfig, 554
- GPT Driver, 75
 - GPT_CONTINUOUS, 80
 - GPT_ONESHOT, 80
 - GPT_READY, 80
 - GPT_STOP, 80
 - GPT_UNINIT, 80
 - GPTD1, 88
 - GPTDriver, 80
 - gpt_lld_change_interval, 79
 - gpt_lld_init, 86
 - gpt_lld_polled_delay, 87
 - gpt_lld_start, 86
 - gpt_lld_start_timer, 87
 - gpt_lld_stop, 86
 - gpt_lld_stop_timer, 87
 - gptChangeInterval, 82
 - gptChangeIntervall, 78
 - gptGetCounterX, 78
 - gptGetIntervalX, 78
 - gptInit, 80
 - gptObjectInit, 81
 - gptPolledDelay, 85
 - gptStart, 81
 - gptStartContinuous, 83
 - gptStartContinuousI, 83
 - gptStartOneShot, 83
 - gptStartOneShotI, 84
 - gptStop, 82
 - gptStopTimer, 84
 - gptStopTimerI, 85
 - gptcallback_t, 80
 - gptcnt_t, 80
 - gptfreq_t, 80
 - gptstate_t, 80
 - PLATFORM_GPT_USE_GPT1, 79
- GPT_CONTINUOUS
 - GPT Driver, 80
- GPT_ONESHOT
 - GPT Driver, 80
- GPT_READY
 - GPT Driver, 80
- GPT_STOP
 - GPT Driver, 80
- GPT_UNINIT
 - GPT Driver, 80
- GPTConfig, 524
 - callback, 525
 - frequency, 525
- GPTD1
 - GPT Driver, 88
- GPTDriver, 526
 - config, 526
 - GPT Driver, 80
 - state, 526
- get_descriptor_cb
 - USBConfig, 587
- gpt_lld_change_interval
 - GPT Driver, 79
- gpt_lld_init
 - GPT Driver, 86
- gpt_lld_polled_delay
 - GPT Driver, 87
- gpt_lld_start
 - GPT Driver, 86
- gpt_lld_start_timer
 - GPT Driver, 87
- gpt_lld_stop
 - GPT Driver, 86
- gpt_lld_stop_timer
 - GPT Driver, 87
- gptChangeInterval
 - GPT Driver, 82
- gptChangeIntervall
 - GPT Driver, 78
- gptGetCounterX
 - GPT Driver, 78
- gptGetIntervalX
 - GPT Driver, 78
- gptInit
 - GPT Driver, 80
- gptObjectInit
 - GPT Driver, 81
- gptPolledDelay
 - GPT Driver, 85
- gptStart
 - GPT Driver, 81
- gptStartContinuous
 - GPT Driver, 83
- gptStartContinuousI
 - GPT Driver, 83
- gptStartOneShot
 - GPT Driver, 83
- gptStartOneShotI
 - GPT Driver, 84
- gptStop
 - GPT Driver, 82
- gptStopTimer
 - GPT Driver, 84
- gptStopTimerI
 - GPT Driver, 85
- gptcallback_t
 - GPT Driver, 80
- gptcnt_t
 - GPT Driver, 80
- gptfreq_t
 - GPT Driver, 80
- gptstate_t
 - GPT Driver, 80
- grpp
 - ADCDriver, 493

- DACDriver, 518
- HAL Driver, 89
 - _CHIBIOS_HAL_, 90
 - CH_HAL_MAJOR, 90
 - CH_HAL_MINOR, 90
 - CH_HAL_PATCH, 90
 - CH_HAL_STABLE, 90
 - HAL_VERSION, 90
 - hal_ild_init, 91
 - halInit, 90
- HAL_USE_ADC
 - Configuration, 208
- HAL_USE_CAN
 - Configuration, 208
- HAL_USE_DAC
 - Configuration, 208
- HAL_USE_EXT
 - Configuration, 208
- HAL_USE_GPT
 - Configuration, 208
- HAL_USE_I2C
 - Configuration, 209
- HAL_USE_I2S
 - Configuration, 209
- HAL_USE_ICU
 - Configuration, 209
- HAL_USE_MAC
 - Configuration, 209
- HAL_USE_MMC_SPI
 - Configuration, 209
- HAL_USE_PAL
 - Configuration, 208
- HAL_USE_PWM
 - Configuration, 209
- HAL_USE_QSPI
 - Configuration, 209
- HAL_USE_RTC
 - Configuration, 209
- HAL_USE_SDC
 - Configuration, 209
- HAL_USE_SERIAL_USB
 - Configuration, 209
- HAL_USE_SERIAL
 - Configuration, 209
- HAL_USE_SPI
 - Configuration, 209
- HAL_USE_UART
 - Configuration, 210
- HAL_USE_USB
 - Configuration, 210
- HAL_USE_WDG
 - Configuration, 210
- HAL_VERSION
 - HAL Driver, 90
- HAL, 203
- hal.c, 599
- hal.h, 599
- hal_adc.c, 601
- hal_adc.h, 602
- hal_adc_ild.c, 603
- hal_adc_ild.h, 603
- hal_buffers.c, 605
- hal_buffers.h, 606
- hal_can.c, 608
- hal_can.h, 608
- hal_can_ild.c, 610
- hal_can_ild.h, 610
- hal_channels.h, 612
- hal_dac.c, 613
- hal_dac.h, 614
- hal_dac_ild.c, 615
- hal_dac_ild.h, 616
- hal_ext.c, 617
- hal_ext.h, 617
- hal_ext_ild.c, 618
- hal_ext_ild.h, 619
- hal_files.h, 620
- hal_gpt.c, 621
- hal_gpt.h, 622
- hal_gpt_ild.c, 623
- hal_gpt_ild.h, 624
- hal_i2c.c, 625
- hal_i2c.h, 625
- hal_i2c_ild.c, 627
- hal_i2c_ild.h, 627
- hal_i2s.c, 628
- hal_i2s.h, 629
- hal_i2s_ild.c, 630
- hal_i2s_ild.h, 630
- hal_icu.c, 631
- hal_icu.h, 632
- hal_icu_ild.c, 633
- hal_icu_ild.h, 634
- hal_ioblock.h, 635
- hal_ild.c, 636
- hal_ild.h, 636
- hal_ild_init
 - HAL Driver, 91
- hal_mac.c, 636
- hal_mac.h, 637
- hal_mac_ild.c, 638
- hal_mac_ild.h, 639
- hal_mii.h, 640
- hal_mmc_spi.c, 642
- hal_mmc_spi.h, 644
- hal_mmcsd.c, 645
- hal_mmcsd.h, 646
- hal_pal.c, 650
- hal_pal.h, 650
- hal_pal_ild.c, 653
- hal_pal_ild.h, 653
- hal_pwm.c, 655
- hal_pwm.h, 655
- hal_pwm_ild.c, 657
- hal_pwm_ild.h, 658
- hal_qspi.c, 659

- hal_qspi.h, 660
- hal_qspi_llc.c, 662
- hal_qspi_llc.h, 663
- hal_queues.c, 664
- hal_queues.h, 665
- hal_rtc.c, 666
- hal_rtc.h, 667
- hal_rtc_llc.c, 668
- hal_rtc_llc.h, 669
- hal_sdc.c, 670
- hal_sdc.h, 672
- hal_sdc_llc.c, 673
- hal_sdc_llc.h, 674
- hal_serial.c, 675
- hal_serial.h, 676
- hal_serial_llc.c, 678
- hal_serial_llc.h, 678
- hal_serial_usb.c, 679
- hal_serial_usb.h, 680
- hal_spi.c, 681
- hal_spi.h, 682
- hal_spi_llc.c, 684
- hal_spi_llc.h, 684
- hal_st.c, 686
- hal_st.h, 686
- hal_st_llc.c, 687
- hal_st_llc.h, 687
- hal_streams.h, 688
- hal_uart.c, 688
- hal_uart.h, 689
- hal_uart_llc.c, 691
- hal_uart_llc.h, 692
- hal_usb.c, 693
- hal_usb.h, 694
- hal_usb_cdc.h, 697
- hal_usb_llc.c, 698
 - in, 699
 - out, 699
- hal_usb_llc.h, 700
- hal_wdg.c, 701
- hal_wdg.h, 702
- halInit
 - HAL Driver, 90
- halconf.h, 702
- hscfg
 - MMCCConfig, 544
- I/O Buffers Queues, 92
 - BQ_BUFFER_SIZE, 94
 - bqGetLinkX, 94
 - bqIsSuspendedX, 95
 - bqResumeX, 96
 - bqSizeX, 94
 - bqSpacel, 94
 - bqSuspendI, 95
 - bqnotify_t, 98
 - ibqGetEmptyBufferI, 99
 - ibqGetFullBufferTimeout, 100
 - ibqGetFullBufferTimeoutS, 101
 - ibqGetTimeout, 103
 - ibqIsEmptyI, 96
 - ibqIsFullI, 96
 - ibqObjectInit, 99
 - ibqPostFullBufferI, 100
 - ibqReadTimeout, 103
 - ibqReleaseEmptyBuffer, 102
 - ibqReleaseEmptyBufferS, 102
 - ibqResetI, 99
 - input_buffers_queue_t, 98
 - io_buffers_queue_t, 98
 - obqFlush, 111
 - obqGetEmptyBufferTimeout, 106
 - obqGetEmptyBufferTimeoutS, 107
 - obqGetFullBufferI, 105
 - obqIsEmptyI, 97
 - obqIsFullI, 98
 - obqObjectInit, 104
 - obqPostFullBuffer, 107
 - obqPostFullBufferS, 108
 - obqPutTimeout, 108
 - obqReleaseEmptyBufferI, 105
 - obqResetI, 105
 - obqTryFlushI, 110
 - obqWriteTimeout, 109
 - output_buffers_queue_t, 98
- I/O Bytes Queues, 132
 - input_queue_t, 141
 - io_queue_t, 141
 - iqGet, 138
 - iqGetEmptyI, 135
 - iqGetFullI, 135
 - iqGetTimeout, 143
 - iqIsEmptyI, 136
 - iqIsFullI, 136
 - iqObjectInit, 141
 - iqPutI, 142
 - iqReadTimeout, 143
 - iqResetI, 142
 - oqGetEmptyI, 139
 - oqGetFullI, 138
 - oqGetI, 145
 - oqIsEmptyI, 139
 - oqIsFullI, 140
 - oqObjectInit, 144
 - oqPut, 140
 - oqPutTimeout, 145
 - oqResetI, 144
 - oqWriteTimeout, 146
 - output_queue_t, 141
 - Q_EMPTY, 134
 - Q_FULL, 134
 - Q_OK, 133
 - Q_RESET, 134
 - Q_TIMEOUT, 133
 - qGetLink, 135
 - qSizeX, 134
 - qSpacel, 134

- qnotify_t, 141
- I2C Driver, 150
 - _i2c_wakeup_error_isr, 153
 - _i2c_wakeup_isr, 153
 - I2C_ACK_FAILURE, 153
 - I2C_ACTIVE_RX, 155
 - I2C_ACTIVE_TX, 155
 - I2C_ARBITRATION_LOST, 152
 - I2C_BUS_ERROR, 152
 - I2C_NO_ERROR, 152
 - I2C_OVERRUN, 153
 - I2C_PEC_ERROR, 153
 - I2C_READY, 155
 - I2C_SMB_ALERT, 153
 - I2C_STOP, 155
 - I2C_TIMEOUT, 153
 - I2C_UNINIT, 155
 - I2C_USE_MUTUAL_EXCLUSION, 153
 - I2CD1, 162
 - I2CDriver, 155
 - i2c_llc_get_errors, 154
 - i2c_llc_init, 159
 - i2c_llc_master_receive_timeout, 160
 - i2c_llc_master_transmit_timeout, 161
 - i2c_llc_start, 160
 - i2c_llc_stop, 160
 - i2cAcquireBus, 159
 - i2cGetErrors, 157
 - i2cInit, 155
 - i2cMasterReceive, 154
 - i2cMasterReceiveTimeout, 158
 - i2cMasterTransmit, 154
 - i2cMasterTransmitTimeout, 157
 - i2cObjectInit, 156
 - i2cReleaseBus, 159
 - i2cStart, 156
 - i2cStop, 156
 - i2caddr_t, 155
 - i2cflags_t, 155
 - i2cstate_t, 155
 - PLATFORM_I2C_USE_I2C1, 154
- I2C_ACK_FAILURE
 - I2C Driver, 153
- I2C_ACTIVE_RX
 - I2C Driver, 155
- I2C_ACTIVE_TX
 - I2C Driver, 155
- I2C_ARBITRATION_LOST
 - I2C Driver, 152
- I2C_BUS_ERROR
 - I2C Driver, 152
- I2C_NO_ERROR
 - I2C Driver, 152
- I2C_OVERRUN
 - I2C Driver, 153
- I2C_PEC_ERROR
 - I2C Driver, 153
- I2C_READY
 - I2C Driver, 155
- I2C_SMB_ALERT
 - I2C Driver, 153
- I2C_STOP
 - I2C Driver, 155
- I2C_TIMEOUT
 - I2C Driver, 153
- I2C_UNINIT
 - I2C Driver, 155
- I2C_USE_MUTUAL_EXCLUSION
 - Configuration, 210
 - I2C Driver, 153
- I2CConfig, 526
- I2CD1
 - I2C Driver, 162
- I2CDriver, 527
 - config, 528
 - errors, 528
 - I2C Driver, 155
 - state, 528
- I2S Driver, 163
 - _i2s_isr_full_code, 165
 - _i2s_isr_half_code, 165
 - I2S_ACTIVE, 167
 - I2S_COMPLETE, 167
 - I2S_READY, 167
 - I2S_STOP, 167
 - I2S_UNINIT, 167
 - I2SD1, 170
 - I2SDriver, 166
 - i2s_llc_init, 169
 - i2s_llc_start, 169
 - i2s_llc_start_exchange, 170
 - i2s_llc_stop, 170
 - i2s_llc_stop_exchange, 170
 - i2sInit, 167
 - i2sObjectInit, 167
 - i2sStart, 167
 - i2sStartExchange, 168
 - i2sStartExchangeI, 164
 - i2sStop, 168
 - i2sStopExchange, 169
 - i2sStopExchangeI, 164
 - i2scallback_t, 166
 - i2sstate_t, 167
 - PLATFORM_I2S_USE_I2S1, 166
- I2S_ACTIVE
 - I2S Driver, 167
- I2S_COMPLETE
 - I2S Driver, 167
- I2S_READY
 - I2S Driver, 167
- I2S_STOP
 - I2S Driver, 167
- I2S_UNINIT
 - I2S Driver, 167
- I2SConfig, 528
 - end_cb, 530

- rx_buffer, [529](#)
- size, [530](#)
- tx_buffer, [529](#)
- I2SD1
 - I2S Driver, [170](#)
- I2SDriver, [530](#)
 - config, [531](#)
 - I2S Driver, [166](#)
 - state, [531](#)
- i2c_llid_get_errors
 - I2C Driver, [154](#)
- i2c_llid_init
 - I2C Driver, [159](#)
- i2c_llid_master_receive_timeout
 - I2C Driver, [160](#)
- i2c_llid_master_transmit_timeout
 - I2C Driver, [161](#)
- i2c_llid_start
 - I2C Driver, [160](#)
- i2c_llid_stop
 - I2C Driver, [160](#)
- i2cAcquireBus
 - I2C Driver, [159](#)
- i2cGetErrors
 - I2C Driver, [157](#)
- i2cInit
 - I2C Driver, [155](#)
- i2cMasterReceive
 - I2C Driver, [154](#)
- i2cMasterReceiveTimeout
 - I2C Driver, [158](#)
- i2cMasterTransmit
 - I2C Driver, [154](#)
- i2cMasterTransmitTimeout
 - I2C Driver, [157](#)
- i2cObjectInit
 - I2C Driver, [156](#)
- i2cReleaseBus
 - I2C Driver, [159](#)
- i2cStart
 - I2C Driver, [156](#)
- i2cStop
 - I2C Driver, [156](#)
- i2caddr_t
 - I2C Driver, [155](#)
- i2cflags_t
 - I2C Driver, [155](#)
- i2cstate_t
 - I2C Driver, [155](#)
- i2s_llid_init
 - I2S Driver, [169](#)
- i2s_llid_start
 - I2S Driver, [169](#)
- i2s_llid_start_exchange
 - I2S Driver, [170](#)
- i2s_llid_stop
 - I2S Driver, [170](#)
- i2s_llid_stop_exchange
 - I2S Driver, [170](#)
- i2sInit
 - I2S Driver, [167](#)
- i2sObjectInit
 - I2S Driver, [167](#)
- i2sStart
 - I2S Driver, [167](#)
- i2sStartExchange
 - I2S Driver, [168](#)
- i2sStartExchangel
 - I2S Driver, [164](#)
- i2sStop
 - I2S Driver, [168](#)
- i2sStopExchange
 - I2S Driver, [169](#)
- i2sStopExchangel
 - I2S Driver, [164](#)
- i2scallback_t
 - I2S Driver, [166](#)
- i2sstate_t
 - I2S Driver, [167](#)
- ICU Driver, [171](#)
 - _icu_isr_invoke_overflow_cb, [177](#)
 - _icu_isr_invoke_period_cb, [177](#)
 - _icu_isr_invoke_width_cb, [177](#)
 - ICU_ACTIVE, [180](#)
 - ICU_INPUT_ACTIVE_HIGH, [180](#)
 - ICU_INPUT_ACTIVE_LOW, [180](#)
 - ICU_READY, [180](#)
 - ICU_STOP, [180](#)
 - ICU_UNINIT, [180](#)
 - ICU_WAITING, [180](#)
 - ICUD1, [187](#)
 - ICUDriver, [179](#)
 - icu_llid_are_notifications_enabled, [179](#)
 - icu_llid_disable_notifications, [186](#)
 - icu_llid_enable_notifications, [186](#)
 - icu_llid_get_period, [178](#)
 - icu_llid_get_width, [178](#)
 - icu_llid_init, [184](#)
 - icu_llid_start, [184](#)
 - icu_llid_start_capture, [185](#)
 - icu_llid_stop, [184](#)
 - icu_llid_stop_capture, [185](#)
 - icu_llid_wait_capture, [185](#)
 - icuAreNotificationsEnabledX, [175](#)
 - icuDisableNotifications, [184](#)
 - icuDisableNotificationsI, [175](#)
 - icuEnableNotifications, [183](#)
 - icuEnableNotificationsI, [174](#)
 - icuGetPeriodX, [176](#)
 - icuGetWidthX, [176](#)
 - iculnit, [180](#)
 - icuObjectInit, [180](#)
 - icuStart, [181](#)
 - icuStartCapture, [182](#)
 - icuStartCaptureI, [174](#)
 - icuStop, [181](#)

- icuStopCapture, [183](#)
- icuStopCaptureI, [174](#)
- icuWaitCapture, [182](#)
- icucallback_t, [179](#)
- icucnt_t, [179](#)
- icufreq_t, [179](#)
- icumode_t, [180](#)
- icustate_t, [180](#)
- PLATFORM_ICU_USE_ICU1, [178](#)
- ICU_ACTIVE
 - ICU Driver, [180](#)
- ICU_INPUT_ACTIVE_HIGH
 - ICU Driver, [180](#)
- ICU_INPUT_ACTIVE_LOW
 - ICU Driver, [180](#)
- ICU_READY
 - ICU Driver, [180](#)
- ICU_STOP
 - ICU Driver, [180](#)
- ICU_UNINIT
 - ICU Driver, [180](#)
- ICU_WAITING
 - ICU Driver, [180](#)
- ICUConfig, [531](#)
 - frequency, [532](#)
 - mode, [532](#)
 - overflow_cb, [532](#)
 - period_cb, [532](#)
 - width_cb, [532](#)
- ICUD1
 - ICU Driver, [187](#)
- ICUDriver, [532](#)
 - config, [533](#)
 - ICU Driver, [179](#)
 - state, [533](#)
- IDE
 - CANRxFrame, [511](#)
 - CANTxFrame, [513](#)
- IOBUS_DECL
 - PAL Driver, [269](#)
- IOBus, [538](#)
 - mask, [538](#)
 - offset, [539](#)
 - portid, [538](#)
- IOPORT1
 - PAL Driver, [280](#)
- ibnotify
 - Serial over USB Driver, [383](#)
- ibqGetEmptyBufferI
 - I/O Buffers Queues, [99](#)
- ibqGetFullBufferTimeout
 - I/O Buffers Queues, [100](#)
- ibqGetFullBufferTimeoutS
 - I/O Buffers Queues, [101](#)
- ibqGetTimeout
 - I/O Buffers Queues, [103](#)
- ibqlsEmptyI
 - I/O Buffers Queues, [96](#)
- ibqlsFullI
 - I/O Buffers Queues, [96](#)
- ibqObjectInit
 - I/O Buffers Queues, [99](#)
- ibqPostFullBufferI
 - I/O Buffers Queues, [100](#)
- ibqReadTimeout
 - I/O Buffers Queues, [103](#)
- ibqReleaseEmptyBuffer
 - I/O Buffers Queues, [102](#)
- ibqReleaseEmptyBufferS
 - I/O Buffers Queues, [102](#)
- ibqResetI
 - I/O Buffers Queues, [99](#)
- icu_ild_are_notifications_enabled
 - ICU Driver, [179](#)
- icu_ild_disable_notifications
 - ICU Driver, [186](#)
- icu_ild_enable_notifications
 - ICU Driver, [186](#)
- icu_ild_get_period
 - ICU Driver, [178](#)
- icu_ild_get_width
 - ICU Driver, [178](#)
- icu_ild_init
 - ICU Driver, [184](#)
- icu_ild_start
 - ICU Driver, [184](#)
- icu_ild_start_capture
 - ICU Driver, [185](#)
- icu_ild_stop
 - ICU Driver, [184](#)
- icu_ild_stop_capture
 - ICU Driver, [185](#)
- icu_ild_wait_capture
 - ICU Driver, [185](#)
- icuAreNotificationsEnabledX
 - ICU Driver, [175](#)
- icuDisableNotifications
 - ICU Driver, [184](#)
- icuDisableNotificationsI
 - ICU Driver, [175](#)
- icuEnableNotifications
 - ICU Driver, [183](#)
- icuEnableNotificationsI
 - ICU Driver, [174](#)
- icuGetPeriodX
 - ICU Driver, [176](#)
- icuGetWidthX
 - ICU Driver, [176](#)
- icuInit
 - ICU Driver, [180](#)
- icuObjectInit
 - ICU Driver, [180](#)
- icuStart
 - ICU Driver, [181](#)
- icuStartCapture
 - ICU Driver, [182](#)

- icuStartCaptureI
 - ICU Driver, [174](#)
- icuStop
 - ICU Driver, [181](#)
- icuStopCapture
 - ICU Driver, [183](#)
- icuStopCaptureI
 - ICU Driver, [174](#)
- icuWaitCapture
 - ICU Driver, [182](#)
- icucallback_t
 - ICU Driver, [179](#)
- icucnt_t
 - ICU Driver, [179](#)
- icufreq_t
 - ICU Driver, [179](#)
- icumode_t
 - ICU Driver, [180](#)
- icustate_t
 - ICU Driver, [180](#)
- in
 - hal_usb_lld.c, [699](#)
 - USB Driver, [480](#)
- in_cb
 - USBEndpointConfig, [593](#)
- in_maxsize
 - USBEndpointConfig, [593](#)
- in_params
 - USBDriver, [590](#)
- in_state
 - USBEndpointConfig, [594](#)
- Inner Code, [216](#)
- input_buffers_queue_t
 - I/O Buffers Queues, [98](#)
- input_queue_t
 - I/O Bytes Queues, [141](#)
- int_in
 - SerialUSBConfig, [572](#)
- Interfaces, [215](#)
- io_buffers_queue, [533](#)
 - bcounter, [535](#)
 - bn, [535](#)
 - brdptr, [535](#)
 - bsize, [535](#)
 - btcp, [535](#)
 - buffers, [535](#)
 - bwrptr, [535](#)
 - link, [536](#)
 - notify, [536](#)
 - ptr, [535](#)
 - suspended, [535](#)
 - top, [536](#)
 - waiting, [535](#)
- io_buffers_queue_t
 - I/O Buffers Queues, [98](#)
- io_queue, [536](#)
 - q_buffer, [537](#)
 - q_counter, [537](#)
 - q_link, [537](#)
 - q_notify, [537](#)
 - q_rdprr, [537](#)
 - q_top, [537](#)
 - q_waiting, [537](#)
 - q_wrprr, [537](#)
- io_queue_t
 - I/O Bytes Queues, [141](#)
- ioline_t
 - PAL Driver, [288](#)
- iomode_t
 - PAL Driver, [288](#)
- ioportid_t
 - PAL Driver, [288](#)
- ioportmask_t
 - PAL Driver, [287](#)
- iqGet
 - I/O Bytes Queues, [138](#)
- iqGetEmptyI
 - I/O Bytes Queues, [135](#)
- iqGetFullI
 - I/O Bytes Queues, [135](#)
- iqGetTimeout
 - I/O Bytes Queues, [143](#)
- iqIsEmptyI
 - I/O Bytes Queues, [136](#)
- iqIsFullI
 - I/O Bytes Queues, [136](#)
- iqObjectInit
 - I/O Bytes Queues, [141](#)
- iqPutI
 - I/O Bytes Queues, [142](#)
- iqReadTimeout
 - I/O Bytes Queues, [143](#)
- iqResetI
 - I/O Bytes Queues, [142](#)
- LPA_100BASE4
 - MII/RMII Header, [227](#)
- LPA_100FULL
 - MII/RMII Header, [227](#)
- LPA_100HALF
 - MII/RMII Header, [227](#)
- LPA_10FULL
 - MII/RMII Header, [227](#)
- LPA_10HALF
 - MII/RMII Header, [227](#)
- LPA_LPACK
 - MII/RMII Header, [228](#)
- LPA_NPAGE
 - MII/RMII Header, [228](#)
- LPA_PAUSE_ASYM
 - MII/RMII Header, [228](#)
- LPA_PAUSE_CAP
 - MII/RMII Header, [227](#)
- LPA_RESV
 - MII/RMII Header, [228](#)
- LPA_RFAULT
 - MII/RMII Header, [228](#)

- LPA_SLCT
 - MII/RMII Header, 227
- link
 - io_buffers_queue, 536
- lscfg
 - MMCCConfig, 543
- MAC Driver, 188
 - ETHD1, 202
 - MAC_ACTIVE, 193
 - MAC_STOP, 193
 - MAC_SUPPORTS_ZERO_COPY, 192
 - MAC_UNINIT, 193
 - MAC_USE_EVENTS, 190
 - MAC_USE_ZERO_COPY, 190
 - MACDriver, 192
 - mac_lld_get_next_receive_buffer, 201
 - mac_lld_get_next_transmit_buffer, 201
 - mac_lld_get_receive_descriptor, 199
 - mac_lld_get_transmit_descriptor, 198
 - mac_lld_init, 197
 - mac_lld_poll_link_status, 200
 - mac_lld_read_receive_descriptor, 200
 - mac_lld_release_receive_descriptor, 199
 - mac_lld_release_transmit_descriptor, 199
 - mac_lld_start, 198
 - mac_lld_stop, 198
 - mac_lld_write_transmit_descriptor, 200
 - macGetNextReceiveBuffer, 191
 - macGetNextTransmitBuffer, 191
 - macGetReceiveEventSource, 190
 - macInit, 193
 - macObjectInit, 193
 - macPollLinkStatus, 197
 - macReadReceiveDescriptor, 191
 - macReleaseReceiveDescriptor, 196
 - macReleaseTransmitDescriptor, 195
 - macStart, 193
 - macStop, 194
 - macWaitReceiveDescriptor, 195
 - macWaitTransmitDescriptor, 194
 - macWriteTransmitDescriptor, 190
 - macstate_t, 192
 - PLATFORM_MAC_USE_MAC1, 192
- MAC_ACTIVE
 - MAC Driver, 193
- MAC_STOP
 - MAC Driver, 193
- MAC_SUPPORTS_ZERO_COPY
 - MAC Driver, 192
- MAC_UNINIT
 - MAC Driver, 193
- MAC_USE_EVENTS
 - Configuration, 210
 - MAC Driver, 190
- MAC_USE_ZERO_COPY
 - Configuration, 210
 - MAC Driver, 190
- MACConfig, 539
 - mac_address, 539
- MACDriver, 539
 - config, 540
 - MAC Driver, 192
 - rdevent, 541
 - rdqueue, 541
 - state, 540
 - tdqueue, 540
- MACReceiveDescriptor, 541
 - offset, 541
 - size, 541
- MACTransmitDescriptor, 542
 - offset, 542
 - size, 542
- MII/RMII Header, 220
 - ADVERTISE_100BASE4, 226
 - ADVERTISE_100FULL, 226
 - ADVERTISE_100HALF, 226
 - ADVERTISE_10FULL, 226
 - ADVERTISE_10HALF, 226
 - ADVERTISE_CSMA, 226
 - ADVERTISE_LPACK, 227
 - ADVERTISE_NPAGE, 227
 - ADVERTISE_PAUSE_ASYM, 227
 - ADVERTISE_PAUSE_CAP, 226
 - ADVERTISE_RESV, 227
 - ADVERTISE_RFAULT, 227
 - ADVERTISE_SLCT, 226
 - BMCR_ANENABLE, 224
 - BMCR_ANRESTART, 224
 - BMCR_CTST, 224
 - BMCR_FULLDPLX, 224
 - BMCR_ISOLATE, 224
 - BMCR_LOOPBACK, 225
 - BMCR_PDOWN, 224
 - BMCR_RESET, 225
 - BMCR_RESV, 224
 - BMCR_SPEED100, 225
 - BMSR_100BASE4, 226
 - BMSR_100FULL, 226
 - BMSR_100HALF, 226
 - BMSR_10FULL, 226
 - BMSR_10HALF, 225
 - BMSR_ANEGCAPABLE, 225
 - BMSR_ANEGCOMPLETE, 225
 - BMSR_ERCAP, 225
 - BMSR_JCD, 225
 - BMSR_LSTATUS, 225
 - BMSR_MFPRESUPPCAP, 225
 - BMSR_RESV, 225
 - BMSR_RFAULT, 225
 - EXPANSION_ENABLENPAGE, 228
 - EXPANSION_LCWP, 228
 - EXPANSION_MFAULTS, 228
 - EXPANSION_NPCAPABLE, 228
 - EXPANSION_NWAY, 228
 - EXPANSION_RESV, 228
 - LPA_100BASE4, 227

- LPA_100FULL, [227](#)
- LPA_100HALF, [227](#)
- LPA_10FULL, [227](#)
- LPA_10HALF, [227](#)
- LPA_LPACK, [228](#)
- LPA_NPAGE, [228](#)
- LPA_PAUSE_ASYM, [228](#)
- LPA_PAUSE_CAP, [227](#)
- LPA_RESV, [228](#)
- LPA_RFAULT, [228](#)
- LPA_SLCT, [227](#)
- MII_ADVERTISE, [222](#)
- MII_ANNPTR, [223](#)
- MII_BMCR, [222](#)
- MII_BMSR, [222](#)
- MII_CTRL1000, [223](#)
- MII_DCOUNTER, [223](#)
- MII_ESTATUS, [223](#)
- MII_EXPANSION, [222](#)
- MII_FCSCOUNTER, [223](#)
- MII_LBRERROR, [224](#)
- MII_LPA, [222](#)
- MII_MICR, [223](#)
- MII_NCONFIG, [224](#)
- MII_NWAYTEST, [223](#)
- MII_PHYADDR, [224](#)
- MII_PHYSID1, [222](#)
- MII_PHYSID2, [222](#)
- MII_PHYSTS, [223](#)
- MII_RERRCOUNTER, [223](#)
- MII_RESV1, [223](#)
- MII_RESV2, [224](#)
- MII_SREVISION, [223](#)
- MII_STAT1000, [223](#)
- MII_TPISTATUS, [224](#)
- NWAYTEST_LOOPBACK, [229](#)
- NWAYTEST_RESV1, [228](#)
- NWAYTEST_RESV2, [229](#)
- MII_ADVERTISE
 - MII/RMII Header, [222](#)
- MII_ANNPTR
 - MII/RMII Header, [223](#)
- MII_BMCR
 - MII/RMII Header, [222](#)
- MII_BMSR
 - MII/RMII Header, [222](#)
- MII_CTRL1000
 - MII/RMII Header, [223](#)
- MII_DCOUNTER
 - MII/RMII Header, [223](#)
- MII_ESTATUS
 - MII/RMII Header, [223](#)
- MII_EXPANSION
 - MII/RMII Header, [222](#)
- MII_FCSCOUNTER
 - MII/RMII Header, [223](#)
- MII_LBRERROR
 - MII/RMII Header, [224](#)
- MII_LPA
 - MII/RMII Header, [222](#)
- MII_MICR
 - MII/RMII Header, [223](#)
- MII_NCONFIG
 - MII/RMII Header, [224](#)
- MII_NWAYTEST
 - MII/RMII Header, [223](#)
- MII_PHYADDR
 - MII/RMII Header, [224](#)
- MII_PHYSID1
 - MII/RMII Header, [222](#)
- MII_PHYSID2
 - MII/RMII Header, [222](#)
- MII_PHYSTS
 - MII/RMII Header, [223](#)
- MII_RERRCOUNTER
 - MII/RMII Header, [223](#)
- MII_RESV1
 - MII/RMII Header, [223](#)
- MII_RESV2
 - MII/RMII Header, [224](#)
- MII_SREVISION
 - MII/RMII Header, [223](#)
- MII_STAT1000
 - MII/RMII Header, [223](#)
- MII_TPISTATUS
 - MII/RMII Header, [224](#)
- MMC over SPI Driver, [230](#)
 - _mmc_driver_methods, [232](#)
 - crc7, [233](#)
 - crc7_lookup_table, [250](#)
 - MMC_NICE_WAITING, [232](#)
 - mmc_vmt, [250](#)
 - mmcConnect, [240](#)
 - mmcDisconnect, [241](#)
 - mmcErase, [249](#)
 - mmcGetInfo, [249](#)
 - mmcInit, [239](#)
 - mmclsCardInserted, [232](#)
 - mmclsWriteProtected, [232](#)
 - mmcObjectInit, [239](#)
 - mmcSequentialRead, [243](#)
 - mmcSequentialWrite, [246](#)
 - mmcStart, [239](#)
 - mmcStartSequentialRead, [242](#)
 - mmcStartSequentialWrite, [245](#)
 - mmcStop, [240](#)
 - mmcStopSequentialRead, [244](#)
 - mmcStopSequentialWrite, [247](#)
 - mmcSync, [248](#)
 - read_CxD, [237](#)
 - recvr1, [234](#)
 - recvr3, [235](#)
 - send_command_R1, [236](#)
 - send_command_R3, [236](#)
 - send_hdr, [234](#)
 - sync, [238](#)

- wait, [233](#)
- MMC/SD Block Device, [252](#)
 - _mmcscd_block_device_data, [257](#)
 - _mmcscd_block_device_methods, [256](#)
 - _mmcscd_get_capacity, [258](#)
 - _mmcscd_get_capacity_ext, [259](#)
 - _mmcscd_get_slice, [258](#)
 - _mmcscd_unpack_csd_mmc, [260](#)
 - _mmcscd_unpack_csd_v10, [261](#)
 - _mmcscd_unpack_csd_v20, [261](#)
 - _mmcscd_unpack_mmc_cid, [260](#)
 - _mmcscd_unpack_sdc_cid, [259](#)
 - MMCSDBLOCK_SIZE, [256](#)
 - MMCSDCID_SDC_CRC_SLICE, [256](#)
 - MMCSDCMD8_PATTERN, [256](#)
 - MMCSDCSD_MMC_CSD_STRUCTURE_SLICE, [256](#)
 - MMCSDR1_ERROR_MASK, [256](#)
 - MMCSDR1_ERROR, [257](#)
 - MMCSDR1_IS_CARD_LOCKED, [257](#)
 - MMCSDR1_STS, [257](#)
 - mmcscdGetCardCapacity, [257](#)
- MMC_NICE_WAITING
 - Configuration, [210](#)
 - MMC over SPI Driver, [232](#)
- MMCCConfig, [542](#)
 - hscfg, [544](#)
 - lscfg, [543](#)
 - spip, [543](#)
- MMCDriver, [544](#)
 - config, [545](#)
 - vmt, [545](#)
- MMCDriverVMT, [546](#)
- MMCSDBLOCK_SIZE
 - MMC/SD Block Device, [256](#)
- MMCSDCID_SDC_CRC_SLICE
 - MMC/SD Block Device, [256](#)
- MMCSDCMD8_PATTERN
 - MMC/SD Block Device, [256](#)
- MMCSDCSD_MMC_CSD_STRUCTURE_SLICE
 - MMC/SD Block Device, [256](#)
- MMCSDR1_ERROR_MASK
 - MMC/SD Block Device, [256](#)
- MMCSDR1_ERROR
 - MMC/SD Block Device, [257](#)
- MMCSDR1_IS_CARD_LOCKED
 - MMC/SD Block Device, [257](#)
- MMCSDR1_STS
 - MMC/SD Block Device, [257](#)
- MMCSDBlockDevice, [547](#)
 - vmt, [549](#)
- MMCSDBlockDeviceVMT, [549](#)
- mac_address
 - MACConfig, [539](#)
- mac_llid_get_next_receive_buffer
 - MAC Driver, [201](#)
- mac_llid_get_next_transmit_buffer
 - MAC Driver, [201](#)
- mac_llid_get_receive_descriptor
 - MAC Driver, [199](#)
- mac_llid_get_transmit_descriptor
 - MAC Driver, [198](#)
- mac_llid_init
 - MAC Driver, [197](#)
- mac_llid_poll_link_status
 - MAC Driver, [200](#)
- mac_llid_read_receive_descriptor
 - MAC Driver, [200](#)
- mac_llid_release_receive_descriptor
 - MAC Driver, [199](#)
- mac_llid_release_transmit_descriptor
 - MAC Driver, [199](#)
- mac_llid_start
 - MAC Driver, [198](#)
- mac_llid_stop
 - MAC Driver, [198](#)
- mac_llid_write_transmit_descriptor
 - MAC Driver, [200](#)
- macGetNextReceiveBuffer
 - MAC Driver, [191](#)
- macGetNextTransmitBuffer
 - MAC Driver, [191](#)
- macGetReceiveEventSource
 - MAC Driver, [190](#)
- macInit
 - MAC Driver, [193](#)
- macObjectInit
 - MAC Driver, [193](#)
- macPollLinkStatus
 - MAC Driver, [197](#)
- macReadReceiveDescriptor
 - MAC Driver, [191](#)
- macReleaseReceiveDescriptor
 - MAC Driver, [196](#)
- macReleaseTransmitDescriptor
 - MAC Driver, [195](#)
- macStart
 - MAC Driver, [193](#)
- macStop
 - MAC Driver, [194](#)
- macWaitReceiveDescriptor
 - MAC Driver, [195](#)
- macWaitTransmitDescriptor
 - MAC Driver, [194](#)
- macWriteTransmitDescriptor
 - MAC Driver, [190](#)
- macstate_t
 - MAC Driver, [192](#)
- mask
 - IOBus, [538](#)
- millisecond
 - RTCDateTime, [560](#)
- mmc_cmd6_construct
 - SDC Driver, [345](#)
- mmc_detect_bus_clk
 - SDC Driver, [347](#)

- mmc_init
 - SDC Driver, [344](#)
- mmc_set_bus_width
 - SDC Driver, [350](#)
- mmc_switch_t
 - SDC Driver, [342](#)
- mmc_vmt
 - MMC over SPI Driver, [250](#)
- mmcConnect
 - MMC over SPI Driver, [240](#)
- mmcDisconnect
 - MMC over SPI Driver, [241](#)
- mmcErase
 - MMC over SPI Driver, [249](#)
- mmcGetInfo
 - MMC over SPI Driver, [249](#)
- mmcInit
 - MMC over SPI Driver, [239](#)
- mmclsCardInserted
 - MMC over SPI Driver, [232](#)
- mmclsWriteProtected
 - MMC over SPI Driver, [232](#)
- mmcObjectInit
 - MMC over SPI Driver, [239](#)
- mmcSequentialRead
 - MMC over SPI Driver, [243](#)
- mmcSequentialWrite
 - MMC over SPI Driver, [246](#)
- mmcStart
 - MMC over SPI Driver, [239](#)
- mmcStartSequentialRead
 - MMC over SPI Driver, [242](#)
- mmcStartSequentialWrite
 - MMC over SPI Driver, [245](#)
- mmcStop
 - MMC over SPI Driver, [240](#)
- mmcStopSequentialRead
 - MMC over SPI Driver, [244](#)
- mmcStopSequentialWrite
 - MMC over SPI Driver, [247](#)
- mmcSync
 - MMC over SPI Driver, [248](#)
- mmcscdGetCardCapacity
 - MMC/SD Block Device, [257](#)
- mode
 - EXTChannelConfig, [519](#)
 - ICUConfig, [532](#)
 - PWMChannelConfig, [552](#)
- mode_detect
 - SDC Driver, [343](#)
- month
 - RTCDatetime, [560](#)
- mutex
 - ADCDriver, [493](#)
 - DACDriver, [518](#)
 - QSPIDriver, [558](#)
 - SPIDriver, [578](#)
 - UARTDriver, [581](#)
- NWAYTEST_LOOPBACK
 - MII/RMII Header, [229](#)
- NWAYTEST_RESV1
 - MII/RMII Header, [228](#)
- NWAYTEST_RESV2
 - MII/RMII Header, [229](#)
- Normal Drivers, [213](#)
- notify
 - io_buffers_queue, [536](#)
- num_channels
 - ADCCConversionGroup, [491](#)
 - DACCConversionGroup, [516](#)
- OSAL, [218](#)
- obnotify
 - Serial over USB Driver, [383](#)
- obqFlush
 - I/O Buffers Queues, [111](#)
- obqGetEmptyBufferTimeout
 - I/O Buffers Queues, [106](#)
- obqGetEmptyBufferTimeoutS
 - I/O Buffers Queues, [107](#)
- obqGetFullBufferI
 - I/O Buffers Queues, [105](#)
- obqIsEmptyI
 - I/O Buffers Queues, [97](#)
- obqIsFullI
 - I/O Buffers Queues, [98](#)
- obqObjectInit
 - I/O Buffers Queues, [104](#)
- obqPostFullBuffer
 - I/O Buffers Queues, [107](#)
- obqPostFullBufferS
 - I/O Buffers Queues, [108](#)
- obqPutTimeout
 - I/O Buffers Queues, [108](#)
- obqReleaseEmptyBufferI
 - I/O Buffers Queues, [105](#)
- obqResetI
 - I/O Buffers Queues, [105](#)
- obqTryFlushI
 - I/O Buffers Queues, [110](#)
- obqWriteTimeout
 - I/O Buffers Queues, [109](#)
- offset
 - IOBus, [539](#)
 - MACReceiveDescriptor, [541](#)
 - MACTransmitDescriptor, [542](#)
- oqGetEmptyI
 - I/O Bytes Queues, [139](#)
- oqGetFullI
 - I/O Bytes Queues, [138](#)
- oqGetI
 - I/O Bytes Queues, [145](#)
- oqlsEmptyI
 - I/O Bytes Queues, [139](#)
- oqlsFullI
 - I/O Bytes Queues, [140](#)
- oqObjectInit

- I/O Bytes Queues, [144](#)
- oqPut
 - I/O Bytes Queues, [140](#)
- oqPutTimeout
 - I/O Bytes Queues, [145](#)
- oqResetI
 - I/O Bytes Queues, [144](#)
- oqWriteTimeout
 - I/O Bytes Queues, [146](#)
- out
 - hal_usb_llid.c, [699](#)
 - USB Driver, [480](#)
- out_cb
 - USBEndpointConfig, [593](#)
- out_maxsize
 - USBEndpointConfig, [594](#)
- out_params
 - USBDriver, [590](#)
- out_state
 - USBEndpointConfig, [594](#)
- output_buffers_queue_t
 - I/O Buffers Queues, [98](#)
- output_queue_t
 - I/O Bytes Queues, [141](#)
- overflow_cb
 - ICUConfig, [532](#)
- PAL Driver, [263](#)
 - _IOBUS_DATA, [269](#)
 - _pal_llid_init, [289](#)
 - _pal_llid_setgroupmode, [289](#)
 - IOBUS_DECL, [269](#)
 - IOPORT1, [280](#)
 - ioline_t, [288](#)
 - iomode_t, [288](#)
 - ioportid_t, [288](#)
 - ioportmask_t, [287](#)
 - PAL_EVENT_MODE_BOTH_EDGES, [269](#)
 - PAL_EVENT_MODE_DISABLED, [268](#)
 - PAL_EVENT_MODE_EDGES_MASK, [268](#)
 - PAL_EVENT_MODE_FALLING_EDGE, [268](#)
 - PAL_EVENT_MODE_RISING_EDGE, [268](#)
 - PAL_GROUP_MASK, [269](#)
 - PAL_HIGH, [268](#)
 - PAL_IOPORTS_WIDTH, [280](#)
 - PAL_LINE, [280](#)
 - PAL_LOW, [268](#)
 - PAL_MODE_INPUT_ANALOG, [268](#)
 - PAL_MODE_INPUT_PULLDOWN, [268](#)
 - PAL_MODE_INPUT_PULLUP, [268](#)
 - PAL_MODE_INPUT, [268](#)
 - PAL_MODE_OUTPUT_OPENDRAIN, [268](#)
 - PAL_MODE_OUTPUT_PUSHPULL, [268](#)
 - PAL_MODE_RESET, [267](#)
 - PAL_MODE_UNCONNECTED, [267](#)
 - PAL_NOLINE, [280](#)
 - PAL_PAD, [280](#)
 - PAL_PORT_BIT, [269](#)
 - PAL_PORT, [280](#)
 - PAL_WHOLE_PORT, [280](#)
 - pal_llid_clearpad, [286](#)
 - pal_llid_clearport, [282](#)
 - pal_llid_init, [280](#)
 - pal_llid_readgroup, [283](#)
 - pal_llid_readlatch, [281](#)
 - pal_llid_readpad, [284](#)
 - pal_llid_readport, [281](#)
 - pal_llid_setgroupmode, [284](#)
 - pal_llid_setpad, [285](#)
 - pal_llid_setpadmode, [287](#)
 - pal_llid_setport, [282](#)
 - pal_llid_togglepad, [286](#)
 - pal_llid_toggleport, [283](#)
 - pal_llid_writegroup, [283](#)
 - pal_llid_writepad, [285](#)
 - pal_llid_writeport, [281](#)
 - palClearLine, [278](#)
 - palClearPad, [275](#)
 - palClearPort, [271](#)
 - palInit, [270](#)
 - palLineDisableEventI, [279](#)
 - palLineEnableEventI, [279](#)
 - palPadDisableEventI, [277](#)
 - palPadEnableEventI, [276](#)
 - palReadBus, [288](#)
 - palReadGroup, [272](#)
 - palReadLatch, [270](#)
 - palReadLine, [277](#)
 - palReadPad, [274](#)
 - palReadPort, [270](#)
 - palSetBusMode, [289](#)
 - palSetGroupMode, [273](#)
 - palSetLine, [278](#)
 - palSetLineMode, [279](#)
 - palSetPad, [275](#)
 - palSetPadMode, [276](#)
 - palSetPort, [271](#)
 - palToggleLine, [278](#)
 - palTogglePad, [275](#)
 - palTogglePort, [272](#)
 - palWriteBus, [288](#)
 - palWriteGroup, [273](#)
 - palWriteLine, [277](#)
 - palWritePad, [274](#)
 - palWritePort, [271](#)
 - palcallback_t, [287](#)
- PAL_EVENT_MODE_BOTH_EDGES
 - PAL Driver, [269](#)
- PAL_EVENT_MODE_DISABLED
 - PAL Driver, [268](#)
- PAL_EVENT_MODE_EDGES_MASK
 - PAL Driver, [268](#)
- PAL_EVENT_MODE_FALLING_EDGE
 - PAL Driver, [268](#)
- PAL_EVENT_MODE_RISING_EDGE
 - PAL Driver, [268](#)
- PAL_GROUP_MASK

- PAL Driver, [269](#)
- PAL_HIGH
 - PAL Driver, [268](#)
- PAL_IOPORTS_WIDTH
 - PAL Driver, [280](#)
- PAL_LINE
 - PAL Driver, [280](#)
- PAL_LOW
 - PAL Driver, [268](#)
- PAL_MODE_INPUT_ANALOG
 - PAL Driver, [268](#)
- PAL_MODE_INPUT_PULLDOWN
 - PAL Driver, [268](#)
- PAL_MODE_INPUT_PULLUP
 - PAL Driver, [268](#)
- PAL_MODE_INPUT
 - PAL Driver, [268](#)
- PAL_MODE_OUTPUT_OPENDRAIN
 - PAL Driver, [268](#)
- PAL_MODE_OUTPUT_PUSHPULL
 - PAL Driver, [268](#)
- PAL_MODE_RESET
 - PAL Driver, [267](#)
- PAL_MODE_UNCONNECTED
 - PAL Driver, [267](#)
- PAL_NOLINE
 - PAL Driver, [280](#)
- PAL_PAD
 - PAL Driver, [280](#)
- PAL_PORT_BIT
 - PAL Driver, [269](#)
- PAL_PORT
 - PAL Driver, [280](#)
- PAL_WHOLE_PORT
 - PAL Driver, [280](#)
- PALConfig, [551](#)
- PLATFORM_ADC_USE_ADC1
 - ADC Driver, [24](#)
- PLATFORM_CAN_USE_CAN1
 - CAN Driver, [37](#)
- PLATFORM_DAC_USE_DAC1
 - DAC Driver, [53](#)
- PLATFORM_EXT_USE_EXT1
 - EXT Driver, [67](#)
- PLATFORM_GPT_USE_GPT1
 - GPT Driver, [79](#)
- PLATFORM_I2C_USE_I2C1
 - I2C Driver, [154](#)
- PLATFORM_I2S_USE_I2S1
 - I2S Driver, [166](#)
- PLATFORM_ICU_USE_ICU1
 - ICU Driver, [178](#)
- PLATFORM_MAC_USE_MAC1
 - MAC Driver, [192](#)
- PLATFORM_PWM_USE_PWM1
 - PWM Driver, [300](#)
- PLATFORM_QSPI_USE_QSPI1
 - QSPI Driver, [317](#)
- PLATFORM_RTC_USE_RTC1
 - RTC Driver, [329](#)
- PLATFORM_SDC_USE_SDC1
 - SDC Driver, [342](#)
- PLATFORM_SERIAL_USE_USART1
 - Serial Driver, [371](#)
- PLATFORM_SPI_USE_SPI1
 - SPI Driver, [399](#)
- PLATFORM_UART_USE_UART1
 - UART Driver, [428](#)
- PLATFORM_USB_USE_USB1
 - USB Driver, [459](#)
- PLATFORM_WDG_USE_WDG1
 - WDG Driver, [485](#)
- PWM Driver, [291](#)
 - PLATFORM_PWM_USE_PWM1, [300](#)
 - PWM_CHANNELS, [300](#)
 - PWM_DEGREES_TO_WIDTH, [295](#)
 - PWM_FRACTION_TO_WIDTH, [294](#)
 - PWM_OUTPUT_ACTIVE_HIGH, [294](#)
 - PWM_OUTPUT_ACTIVE_LOW, [294](#)
 - PWM_OUTPUT_DISABLED, [294](#)
 - PWM_OUTPUT_MASK, [294](#)
 - PWM_PERCENTAGE_TO_WIDTH, [295](#)
 - PWM_READY, [302](#)
 - PWM_STOP, [302](#)
 - PWM_UNINIT, [302](#)
 - PWMD1, [310](#)
 - PWMDriver, [301](#)
 - pwm_lld_change_period, [300](#)
 - pwm_lld_disable_channel, [308](#)
 - pwm_lld_disable_channel_notification, [310](#)
 - pwm_lld_disable_periodic_notification, [309](#)
 - pwm_lld_enable_channel, [307](#)
 - pwm_lld_enable_channel_notification, [309](#)
 - pwm_lld_enable_periodic_notification, [308](#)
 - pwm_lld_init, [306](#)
 - pwm_lld_start, [307](#)
 - pwm_lld_stop, [307](#)
 - pwmChangePeriod, [303](#)
 - pwmChangePeriodI, [296](#)
 - pwmDisableChannel, [304](#)
 - pwmDisableChannelNotification, [306](#)
 - pwmDisableChannelNotificationI, [300](#)
 - pwmDisableChannell, [297](#)
 - pwmDisablePeriodicNotification, [305](#)
 - pwmDisablePeriodicNotificationI, [299](#)
 - pwmEnableChannel, [304](#)
 - pwmEnableChannelNotification, [306](#)
 - pwmEnableChannelNotificationI, [299](#)
 - pwmEnableChannell, [297](#)
 - pwmEnablePeriodicNotification, [305](#)
 - pwmEnablePeriodicNotificationI, [298](#)
 - pwmInit, [302](#)
 - pwmIsChannelEnabledI, [298](#)
 - pwmObjectInit, [302](#)
 - pwmStart, [302](#)
 - pwmStop, [303](#)

- pwmcallback_t, 301
- pwmchannel_t, 301
- pwmchnmsk_t, 301
- pwmcnt_t, 301
- pwmmode_t, 301
- pwmstate_t, 302
- PWM_CHANNELS
 - PWM Driver, 300
- PWM_DEGREES_TO_WIDTH
 - PWM Driver, 295
- PWM_FRACTION_TO_WIDTH
 - PWM Driver, 294
- PWM_OUTPUT_ACTIVE_HIGH
 - PWM Driver, 294
- PWM_OUTPUT_ACTIVE_LOW
 - PWM Driver, 294
- PWM_OUTPUT_DISABLED
 - PWM Driver, 294
- PWM_OUTPUT_MASK
 - PWM Driver, 294
- PWM_PERCENTAGE_TO_WIDTH
 - PWM Driver, 295
- PWM_READY
 - PWM Driver, 302
- PWM_STOP
 - PWM Driver, 302
- PWM_UNINIT
 - PWM Driver, 302
- PWMChannelConfig, 551
 - callback, 552
 - mode, 552
- PWMConfig, 553
 - callback, 554
 - channels, 554
 - frequency, 554
 - period, 554
- PWMD1
 - PWM Driver, 310
- PWMDriver, 554
 - channels, 556
 - config, 556
 - enabled, 556
 - PWM Driver, 301
 - period, 556
 - state, 555
- pal_ild_clearpad
 - PAL Driver, 286
- pal_ild_clearport
 - PAL Driver, 282
- pal_ild_init
 - PAL Driver, 280
- pal_ild_readgroup
 - PAL Driver, 283
- pal_ild_readlatch
 - PAL Driver, 281
- pal_ild_readpad
 - PAL Driver, 284
- pal_ild_readport
 - PAL Driver, 281
- pal_ild_setgroupmode
 - PAL Driver, 284
- pal_ild_setpad
 - PAL Driver, 285
- pal_ild_setpadmode
 - PAL Driver, 287
- pal_ild_setport
 - PAL Driver, 282
- pal_ild_togglepad
 - PAL Driver, 286
- pal_ild_toggleport
 - PAL Driver, 283
- pal_ild_writegroup
 - PAL Driver, 283
- pal_ild_writepad
 - PAL Driver, 285
- pal_ild_writeport
 - PAL Driver, 281
- palClearLine
 - PAL Driver, 278
- palClearPad
 - PAL Driver, 275
- palClearPort
 - PAL Driver, 271
- palInit
 - PAL Driver, 270
- palLineDisableEventI
 - PAL Driver, 279
- palLineEnableEventI
 - PAL Driver, 279
- palPadDisableEventI
 - PAL Driver, 277
- palPadEnableEventI
 - PAL Driver, 276
- palReadBus
 - PAL Driver, 288
- palReadGroup
 - PAL Driver, 272
- palReadLatch
 - PAL Driver, 270
- palReadLine
 - PAL Driver, 277
- palReadPad
 - PAL Driver, 274
- palReadPort
 - PAL Driver, 270
- palSetBusMode
 - PAL Driver, 289
- palSetGroupMode
 - PAL Driver, 273
- palSetLine
 - PAL Driver, 278
- palSetLineMode
 - PAL Driver, 279
- palSetPad
 - PAL Driver, 275
- palSetPadMode
 - PAL Driver, 275

- PAL Driver, [276](#)
- palSetPort
 - PAL Driver, [271](#)
- palToggleLine
 - PAL Driver, [278](#)
- palTogglePad
 - PAL Driver, [275](#)
- palTogglePort
 - PAL Driver, [272](#)
- palWriteBus
 - PAL Driver, [288](#)
- palWriteGroup
 - PAL Driver, [273](#)
- palWriteLine
 - PAL Driver, [277](#)
- palWritePad
 - PAL Driver, [274](#)
- palWritePort
 - PAL Driver, [271](#)
- palcallback_t
 - PAL Driver, [287](#)
- period
 - PWMConfig, [554](#)
 - PWMDriver, [556](#)
- period_cb
 - ICUConfig, [532](#)
- portid
 - IOBus, [538](#)
- ptr
 - io_buffers_queue, [535](#)
- pwm_ild_change_period
 - PWM Driver, [300](#)
- pwm_ild_disable_channel
 - PWM Driver, [308](#)
- pwm_ild_disable_channel_notification
 - PWM Driver, [310](#)
- pwm_ild_disable_periodic_notification
 - PWM Driver, [309](#)
- pwm_ild_enable_channel
 - PWM Driver, [307](#)
- pwm_ild_enable_channel_notification
 - PWM Driver, [309](#)
- pwm_ild_enable_periodic_notification
 - PWM Driver, [308](#)
- pwm_ild_init
 - PWM Driver, [306](#)
- pwm_ild_start
 - PWM Driver, [307](#)
- pwm_ild_stop
 - PWM Driver, [307](#)
- pwmChangePeriod
 - PWM Driver, [303](#)
- pwmChangePeriodI
 - PWM Driver, [296](#)
- pwmDisableChannel
 - PWM Driver, [304](#)
- pwmDisableChannelNotification
 - PWM Driver, [306](#)
- pwmDisableChannelNotificationI
 - PWM Driver, [300](#)
- pwmDisableChannelI
 - PWM Driver, [297](#)
- pwmDisablePeriodicNotification
 - PWM Driver, [305](#)
- pwmDisablePeriodicNotificationI
 - PWM Driver, [299](#)
- pwmEnableChannel
 - PWM Driver, [304](#)
- pwmEnableChannelNotification
 - PWM Driver, [306](#)
- pwmEnableChannelNotificationI
 - PWM Driver, [299](#)
- pwmEnableChannelI
 - PWM Driver, [297](#)
- pwmEnablePeriodicNotification
 - PWM Driver, [305](#)
- pwmEnablePeriodicNotificationI
 - PWM Driver, [298](#)
- pwmInit
 - PWM Driver, [302](#)
- pwmIsChannelEnabledI
 - PWM Driver, [298](#)
- pwmObjectInit
 - PWM Driver, [302](#)
- pwmStart
 - PWM Driver, [302](#)
- pwmStop
 - PWM Driver, [303](#)
- pwmcallback_t
 - PWM Driver, [301](#)
- pwmchannel_t
 - PWM Driver, [301](#)
- pwmchnmsk_t
 - PWM Driver, [301](#)
- pwmcnt_t
 - PWM Driver, [301](#)
- pwmmode_t
 - PWM Driver, [301](#)
- pwmstate_t
 - PWM Driver, [302](#)
- Q_EMPTY
 - I/O Bytes Queues, [134](#)
- Q_FULL
 - I/O Bytes Queues, [134](#)
- Q_OK
 - I/O Bytes Queues, [133](#)
- Q_RESET
 - I/O Bytes Queues, [134](#)
- Q_TIMEOUT
 - I/O Bytes Queues, [133](#)
- q_buffer
 - io_queue, [537](#)
- q_counter
 - io_queue, [537](#)
- q_link
 - io_queue, [537](#)

- q_notify
 - io_queue, 537
- q_rdprr
 - io_queue, 537
- q_top
 - io_queue, 537
- q_waiting
 - io_queue, 537
- q_wrptr
 - io_queue, 537
- qGetLink
 - I/O Bytes Queues, 135
- QSPI Driver, 311
 - _qspi_isr_code, 317
 - _qspi_wakeup_isr, 316
 - PLATFORM_QSPI_USE_QSPI1, 317
 - QSPI_ACTIVE, 318
 - QSPI_COMPLETE, 318
 - QSPI_MEMMAP, 318
 - QSPI_READY, 318
 - QSPI_STOP, 318
 - QSPI_UNINIT, 318
 - QSPI_USE_MUTUAL_EXCLUSION, 314
 - QSPI_USE_WAIT, 314
 - QSPID1, 326
 - QSPIDriver, 317
 - qspi_llc_command, 324
 - qspi_llc_init, 323
 - qspi_llc_map_flash, 325
 - qspi_llc_receive, 325
 - qspi_llc_send, 325
 - qspi_llc_start, 324
 - qspi_llc_stop, 324
 - qspi_llc_unmap_flash, 326
 - qspiAcquireBus, 323
 - qspiCommand, 321
 - qspiInit, 318
 - qspiMapFlash, 322
 - qspiMapFlashI, 315
 - qspiObjectInit, 318
 - qspiReceive, 322
 - qspiReleaseBus, 323
 - qspiSend, 321
 - qspiStart, 319
 - qspiStartCommand, 320
 - qspiStartCommandI, 314
 - qspiStartReceive, 320
 - qspiStartReceiveI, 315
 - qspiStartSend, 320
 - qspiStartSendI, 314
 - qspiStop, 319
 - qspiUnmapFlash, 322
 - qspiUnmapFlashI, 316
 - qspicallback_t, 317
 - qspistate_t, 318
- QSPI_ACTIVE
 - QSPI Driver, 318
- QSPI_COMPLETE
 - QSPI Driver, 318
- QSPI_MEMMAP
 - QSPI Driver, 318
- QSPI_READY
 - QSPI Driver, 318
- QSPI_STOP
 - QSPI Driver, 318
- QSPI_UNINIT
 - QSPI Driver, 318
- QSPI_USE_MUTUAL_EXCLUSION
 - QSPI Driver, 314
- QSPI_USE_WAIT
 - QSPI Driver, 314
- QSPID1
 - QSPI Driver, 326
- QSPIDriver, 557
 - config, 558
 - mutex, 558
 - QSPI Driver, 317
 - state, 558
 - thread, 558
- qSizeX
 - I/O Bytes Queues, 134
- qSpacel
 - I/O Bytes Queues, 134
- qnotify_t
 - I/O Bytes Queues, 141
- qspi_command_t, 556
- qspi_llc_command
 - QSPI Driver, 324
- qspi_llc_init
 - QSPI Driver, 323
- qspi_llc_map_flash
 - QSPI Driver, 325
- qspi_llc_receive
 - QSPI Driver, 325
- qspi_llc_send
 - QSPI Driver, 325
- qspi_llc_start
 - QSPI Driver, 324
- qspi_llc_stop
 - QSPI Driver, 324
- qspi_llc_unmap_flash
 - QSPI Driver, 326
- qspiAcquireBus
 - QSPI Driver, 323
- qspiCommand
 - QSPI Driver, 321
- qspiInit
 - QSPI Driver, 318
- qspiMapFlash
 - QSPI Driver, 322
- qspiMapFlashI
 - QSPI Driver, 315
- qspiObjectInit
 - QSPI Driver, 318

- qspiReceive
 - QSPI Driver, [322](#)
- qspiReleaseBus
 - QSPI Driver, [323](#)
- qspiSend
 - QSPI Driver, [321](#)
- qspiStart
 - QSPI Driver, [319](#)
- qspiStartCommand
 - QSPI Driver, [320](#)
- qspiStartCommandI
 - QSPI Driver, [314](#)
- qspiStartReceive
 - QSPI Driver, [320](#)
- qspiStartReceiveI
 - QSPI Driver, [315](#)
- qspiStartSend
 - QSPI Driver, [320](#)
- qspiStartSendI
 - QSPI Driver, [314](#)
- qspiStop
 - QSPI Driver, [319](#)
- qspiUnmapFlash
 - QSPI Driver, [322](#)
- qspiUnmapFlashI
 - QSPI Driver, [316](#)
- qspicallback_t
 - QSPI Driver, [317](#)
- qspistate_t
 - QSPI Driver, [318](#)
- RTC Driver, [327](#)
 - _rtc_driver_methods, [329](#)
 - PLATFORM_RTC_USE_RTC1, [329](#)
 - RTC_ALARMS, [329](#)
 - RTC_BASE_YEAR, [329](#)
 - RTC_HAS_STORAGE, [329](#)
 - RTC_SUPPORTS_CALLBACKS, [329](#)
 - RTCDriver, [329](#)
 - rtc_lld_get_alarm, [336](#)
 - rtc_lld_get_time, [335](#)
 - rtc_lld_init, [334](#)
 - rtc_lld_set_alarm, [335](#)
 - rtc_lld_set_time, [335](#)
 - rtcConvertDateTimeToFAT, [334](#)
 - rtcConvertDateTimeToStructTm, [333](#)
 - rtcConvertStructTmToDateTime, [334](#)
 - rtcGetAlarm, [332](#)
 - rtcGetTime, [331](#)
 - rtcInit, [330](#)
 - rtcObjectInit, [330](#)
 - rtcSetAlarm, [332](#)
 - rtcSetCallback, [333](#)
 - rtcSetTime, [330](#)
 - rtcalarm_t, [329](#)
 - rtccb_t, [329](#)
 - rtcevent_t, [330](#)
- RTC_ALARMS
 - RTC Driver, [329](#)
- RTC_BASE_YEAR
 - RTC Driver, [329](#)
- RTC_HAS_STORAGE
 - RTC Driver, [329](#)
- RTC_SUPPORTS_CALLBACKS
 - RTC Driver, [329](#)
- RTCAlarm, [559](#)
- RTCDateTime, [559](#)
 - day, [560](#)
 - dayofweek, [560](#)
 - dstflag, [560](#)
 - millisecond, [560](#)
 - month, [560](#)
 - year, [560](#)
- RTCDriver, [561](#)
 - RTC Driver, [329](#)
 - vmt, [561](#)
- RTCDriverVMT, [561](#)
- RTR
 - CANRxFrame, [511](#)
 - CANTxFrame, [513](#)
- rca
 - SDCDriver, [566](#)
- rdevent
 - MACDriver, [541](#)
- rdqueue
 - MACDriver, [541](#)
- read_CxD
 - MMC over SPI Driver, [237](#)
- receiving
 - USBDriver, [589](#)
- recvr1
 - MMC over SPI Driver, [234](#)
- recvr3
 - MMC over SPI Driver, [235](#)
- requests_hook_cb
 - USBConfig, [587](#)
- rtc_lld_get_alarm
 - RTC Driver, [336](#)
- rtc_lld_get_time
 - RTC Driver, [335](#)
- rtc_lld_init
 - RTC Driver, [334](#)
- rtc_lld_set_alarm
 - RTC Driver, [335](#)
- rtc_lld_set_time
 - RTC Driver, [335](#)
- rtcConvertDateTimeToFAT
 - RTC Driver, [334](#)
- rtcConvertDateTimeToStructTm
 - RTC Driver, [333](#)
- rtcConvertStructTmToDateTime
 - RTC Driver, [334](#)
- rtcGetAlarm
 - RTC Driver, [332](#)
- rtcGetTime
 - RTC Driver, [331](#)
- rtcInit

- RTC Driver, [330](#)
- rtcObjectInit
 - RTC Driver, [330](#)
- rtcSetAlarm
 - RTC Driver, [332](#)
- rtcSetCallback
 - RTC Driver, [333](#)
- rtcSetTime
 - RTC Driver, [330](#)
- rtcalarm_t
 - RTC Driver, [329](#)
- rtccb_t
 - RTC Driver, [329](#)
- rtcevent_t
 - RTC Driver, [330](#)
- rx_buffer
 - I2SConfig, [529](#)
- rxbuf
 - USBOutEndpointState, [596](#)
- rxchar_cb
 - UARTConfig, [579](#)
- rxcnt
 - USBOutEndpointState, [596](#)
- rxend_cb
 - UARTConfig, [579](#)
- rxerr_cb
 - UARTConfig, [579](#)
- rxfull_event
 - CANDriver, [509](#)
- rxqueue
 - CANDriver, [509](#)
- rxsize
 - USBOutEndpointState, [596](#)
- rxstate
 - UARTDriver, [581](#)
- SD1
 - Serial Driver, [379](#)
- SD_BREAK_DETECTED
 - Serial Driver, [368](#)
- SD_FRAMING_ERROR
 - Serial Driver, [367](#)
- SD_NOISE_ERROR
 - Serial Driver, [368](#)
- SD_OVERRUN_ERROR
 - Serial Driver, [368](#)
- SD_PARITY_ERROR
 - Serial Driver, [367](#)
- SD_QUEUE_FULL_ERROR
 - Serial Driver, [368](#)
- SD_READY
 - Serial Driver, [372](#)
- SD_STOP
 - Serial Driver, [372](#)
- SD_UNINIT
 - Serial Driver, [372](#)
- SDC Driver, [337](#)
 - _sdc_driver_methods, [342](#)
 - _sdc_wait_for_transfer_state, [350](#)
 - detect_bus_clk, [348](#)
 - mmc_cmd6_construct, [345](#)
 - mmc_detect_bus_clk, [347](#)
 - mmc_init, [344](#)
 - mmc_set_bus_width, [350](#)
 - mmc_switch_t, [342](#)
 - mode_detect, [343](#)
 - PLATFORM_SDC_USE_SDC1, [342](#)
 - SDC_INIT_OCR_V20, [341](#)
 - SDC_INIT_OCR, [341](#)
 - SDC_INIT_RETRY, [340](#)
 - SDC_MMC_SUPPORT, [340](#)
 - SDC_NICE_WAITING, [340](#)
 - SDCD1, [364](#)
 - SDCDriver, [342](#)
 - sd_switch_function_t, [343](#)
 - sd_switch_t, [342](#)
 - sdccmd6_check_status, [346](#)
 - sdccmd6_construct, [345](#)
 - sdccmd6_extract_info, [346](#)
 - sdccdetect_bus_clk, [346](#)
 - sdccinit, [344](#)
 - sdcclld_init, [358](#)
 - sdcclld_read, [362](#)
 - sdcclld_send_cmd_long_crc, [362](#)
 - sdcclld_send_cmd_none, [360](#)
 - sdcclld_send_cmd_short, [361](#)
 - sdcclld_send_cmd_short_crc, [361](#)
 - sdcclld_set_bus_mode, [360](#)
 - sdcclld_set_data_clk, [359](#)
 - sdcclld_start, [359](#)
 - sdcclld_start_clk, [359](#)
 - sdcclld_stop, [359](#)
 - sdcclld_stop_clk, [360](#)
 - sdcclld_sync, [363](#)
 - sdcclld_write, [363](#)
 - sdccset_bus_width, [349](#)
 - sdccvmt, [363](#)
 - sdccConnect, [353](#)
 - sdccDisconnect, [354](#)
 - sdccErase, [358](#)
 - sdccGetAndClearErrors, [356](#)
 - sdccGetInfo, [357](#)
 - sdccInit, [351](#)
 - sdccIsCardInserted, [341](#)
 - sdccIsWriteProtected, [341](#)
 - sdccObjectInit, [352](#)
 - sdccRead, [355](#)
 - sdccStart, [352](#)
 - sdccStop, [352](#)
 - sdccSync, [357](#)
 - sdccWrite, [355](#)
 - sdccbusclk_t, [343](#)
 - sdccbusmode_t, [343](#)
 - sdccflags_t, [342](#)
 - sdccmode_t, [342](#)
- SDC_INIT_OCR_V20
 - SDC Driver, [341](#)

- SDC_INIT_OCR
 - SDC Driver, [341](#)
- SDC_INIT_RETRY
 - Configuration, [211](#)
 - SDC Driver, [340](#)
- SDC_MMC_SUPPORT
 - Configuration, [211](#)
 - SDC Driver, [340](#)
- SDC_NICE_WAITING
 - Configuration, [211](#)
 - SDC Driver, [340](#)
- SDCConfig, [563](#)
 - bus_width, [564](#)
 - scratchpad, [564](#)
- SDCD1
 - SDC Driver, [364](#)
- SDCDriver, [564](#)
 - cardmode, [566](#)
 - config, [565](#)
 - errors, [566](#)
 - rca, [566](#)
 - SDC Driver, [342](#)
 - vmt, [565](#)
- SDCDriverVMT, [566](#)
- SDU_READY
 - Serial over USB Driver, [383](#)
- SDU_STOP
 - Serial over USB Driver, [383](#)
- SDU_UNINIT
 - Serial over USB Driver, [383](#)
- SERIAL_BUFFERS_SIZE
 - Configuration, [211](#)
 - Serial Driver, [368](#)
- SERIAL_DEFAULT_BITRATE
 - Configuration, [211](#)
 - Serial Driver, [368](#)
- SERIAL_USB_BUFFERS_NUMBER
 - Configuration, [211](#)
 - Serial over USB Driver, [382](#)
- SERIAL_USB_BUFFERS_SIZE
 - Configuration, [211](#)
 - Serial over USB Driver, [382](#)
- SID
 - CANRxFrame, [512](#)
 - CANTxFrame, [513](#)
- SPI Driver, [391](#)
 - _spi_isr_code, [398](#)
 - _spi_wakeup_isr, [398](#)
 - PLATFORM_SPI_USE_SPI1, [399](#)
 - SPI_ACTIVE, [399](#)
 - SPI_COMPLETE, [399](#)
 - SPI_READY, [399](#)
 - SPI_STOP, [399](#)
 - SPI_UNINIT, [399](#)
 - SPI_USE_MUTUAL_EXCLUSION, [394](#)
 - SPI_USE_WAIT, [394](#)
 - SPID1, [411](#)
 - SPIDriver, [399](#)
 - spi_llc_exchange, [409](#)
 - spi_llc_ignore, [409](#)
 - spi_llc_init, [407](#)
 - spi_llc_polled_exchange, [410](#)
 - spi_llc_receive, [410](#)
 - spi_llc_select, [408](#)
 - spi_llc_send, [410](#)
 - spi_llc_start, [408](#)
 - spi_llc_stop, [408](#)
 - spi_llc_unselect, [408](#)
 - spiAcquireBus, [407](#)
 - spiExchange, [405](#)
 - spiIgnore, [405](#)
 - spiInit, [399](#)
 - spiObjectInit, [400](#)
 - spiPolledExchange, [397](#)
 - spiReceive, [406](#)
 - spiReleaseBus, [407](#)
 - spiSelect, [401](#)
 - spiSelectI, [394](#)
 - spiSend, [406](#)
 - spiStart, [400](#)
 - spiStartExchange, [402](#)
 - spiStartExchangeI, [395](#)
 - spiStartIgnore, [402](#)
 - spiStartIgnoreI, [395](#)
 - spiStartReceive, [404](#)
 - spiStartReceiveI, [397](#)
 - spiStartSend, [404](#)
 - spiStartSendI, [396](#)
 - spiStop, [401](#)
 - spiUnselect, [401](#)
 - spiUnselectI, [394](#)
 - spicallback_t, [399](#)
 - spistate_t, [399](#)
- SPI_ACTIVE
 - SPI Driver, [399](#)
- SPI_COMPLETE
 - SPI Driver, [399](#)
- SPI_READY
 - SPI Driver, [399](#)
- SPI_STOP
 - SPI Driver, [399](#)
- SPI_UNINIT
 - SPI Driver, [399](#)
- SPI_USE_MUTUAL_EXCLUSION
 - Configuration, [212](#)
 - SPI Driver, [394](#)
- SPI_USE_WAIT
 - Configuration, [212](#)
 - SPI Driver, [394](#)
- SPIConfig, [576](#)
 - end_cb, [576](#)
- SPID1
 - SPI Driver, [411](#)
- SPIDriver, [577](#)
 - config, [577](#)
 - mutex, [578](#)

- SPI Driver, 399
- state, 577
- thread, 578
- ST Driver, 412
 - st_lld_get_alarm, 417
 - st_lld_get_counter, 416
 - st_lld_init, 415
 - st_lld_is_alarm_active, 417
 - st_lld_set_alarm, 416
 - st_lld_start_alarm, 416
 - st_lld_stop_alarm, 416
 - stGetAlarm, 415
 - stGetCounter, 412
 - stInit, 413
 - stIsAlarmActive, 413
 - stSetAlarm, 414
 - stStartAlarm, 413
 - stStopAlarm, 414
- samples
 - ADCDriver, 493
 - DACDriver, 518
- saved_state
 - USBDriver, 590
- scratchpad
 - SDCConfig, 564
- sd_lld_init
 - Serial Driver, 378
- sd_lld_start
 - Serial Driver, 378
- sd_lld_stop
 - Serial Driver, 379
- sd_switch_function_t
 - SDC Driver, 343
- sd_switch_t
 - SDC Driver, 342
- sdAsynchronousRead
 - Serial Driver, 371
- sdAsynchronousWrite
 - Serial Driver, 370
- sdGet
 - Serial Driver, 369
- sdGetTimeout
 - Serial Driver, 369
- sdGetWouldBlock
 - Serial Driver, 377
- sdIncomingData
 - Serial Driver, 374
- sdInit
 - Serial Driver, 372
- sdObjectInit
 - Serial Driver, 373
- sdPut
 - Serial Driver, 368
- sdPutTimeout
 - Serial Driver, 369
- sdPutWouldBlock
 - Serial Driver, 377
- sdRead
 - Serial Driver, 370
- sdReadTimeout
 - Serial Driver, 371
- sdRequestData
 - Serial Driver, 376
- sdStart
 - Serial Driver, 373
- sdStop
 - Serial Driver, 374
- sdWrite
 - Serial Driver, 369
- sdWriteTimeout
 - Serial Driver, 370
- sd_cmd6_check_status
 - SDC Driver, 346
- sd_cmd6_construct
 - SDC Driver, 345
- sd_cmd6_extract_info
 - SDC Driver, 346
- sd_detect_bus_clk
 - SDC Driver, 346
- sd_init
 - SDC Driver, 344
- sd_lld_init
 - SDC Driver, 358
- sd_lld_read
 - SDC Driver, 362
- sd_lld_send_cmd_long_crc
 - SDC Driver, 362
- sd_lld_send_cmd_none
 - SDC Driver, 360
- sd_lld_send_cmd_short
 - SDC Driver, 361
- sd_lld_send_cmd_short_crc
 - SDC Driver, 361
- sd_lld_set_bus_mode
 - SDC Driver, 360
- sd_lld_set_data_clk
 - SDC Driver, 359
- sd_lld_start
 - SDC Driver, 359
- sd_lld_start_clk
 - SDC Driver, 359
- sd_lld_stop
 - SDC Driver, 359
- sd_lld_stop_clk
 - SDC Driver, 360
- sd_lld_sync
 - SDC Driver, 363
- sd_lld_write
 - SDC Driver, 363
- sd_set_bus_width
 - SDC Driver, 349
- sd_vmt
 - SDC Driver, 363
- sdConnect
 - SDC Driver, 353
- sdDisconnect

- SDC Driver, [354](#)
- sdErase
 - SDC Driver, [358](#)
- sdGetAndClearErrors
 - SDC Driver, [356](#)
- sdGetInfo
 - SDC Driver, [357](#)
- sdInit
 - SDC Driver, [351](#)
- sdIsCardInserted
 - SDC Driver, [341](#)
- sdIsWriteProtected
 - SDC Driver, [341](#)
- sdObjectInit
 - SDC Driver, [352](#)
- sdRead
 - SDC Driver, [355](#)
- sdStart
 - SDC Driver, [352](#)
- sdStop
 - SDC Driver, [352](#)
- sdSync
 - SDC Driver, [357](#)
- sdWrite
 - SDC Driver, [355](#)
- sdBusclk_t
 - SDC Driver, [343](#)
- sdBusmode_t
 - SDC Driver, [343](#)
- sdFlags_t
 - SDC Driver, [342](#)
- sdMode_t
 - SDC Driver, [342](#)
- sdState_t
 - Serial Driver, [372](#)
- sduConfigureHookI
 - Serial over USB Driver, [386](#)
- sduDataReceived
 - Serial over USB Driver, [388](#)
- sduDataTransmitted
 - Serial over USB Driver, [387](#)
- sduInit
 - Serial over USB Driver, [383](#)
- sduInterruptTransmitted
 - Serial over USB Driver, [388](#)
- sduObjectInit
 - Serial over USB Driver, [384](#)
- sduRequestsHook
 - Serial over USB Driver, [386](#)
- sduSOFHookI
 - Serial over USB Driver, [387](#)
- sduStart
 - Serial over USB Driver, [384](#)
- sduStop
 - Serial over USB Driver, [385](#)
- sduSuspendHookI
 - Serial over USB Driver, [385](#)
- sduWakeupHookI
 - Serial over USB Driver, [385](#)
- sdustate_t
 - Serial over USB Driver, [383](#)
- send_command_R1
 - MMC over SPI Driver, [236](#)
- send_command_R3
 - MMC over SPI Driver, [236](#)
- send_hdr
 - MMC over SPI Driver, [234](#)
- Serial Driver, [365](#)
 - _serial_driver_data, [371](#)
 - _serial_driver_methods, [368](#)
 - default_config, [379](#)
 - PLATFORM_SERIAL_USE_USART1, [371](#)
 - SD1, [379](#)
 - SD_BREAK_DETECTED, [368](#)
 - SD_FRAMING_ERROR, [367](#)
 - SD_NOISE_ERROR, [368](#)
 - SD_OVERRUN_ERROR, [368](#)
 - SD_PARITY_ERROR, [367](#)
 - SD_QUEUE_FULL_ERROR, [368](#)
 - SD_READY, [372](#)
 - SD_STOP, [372](#)
 - SD_UNINIT, [372](#)
 - SERIAL_BUFFERS_SIZE, [368](#)
 - SERIAL_DEFAULT_BITRATE, [368](#)
 - sd_ild_init, [378](#)
 - sd_ild_start, [378](#)
 - sd_ild_stop, [379](#)
 - sdAsynchronousRead, [371](#)
 - sdAsynchronousWrite, [370](#)
 - sdGet, [369](#)
 - sdGetTimeout, [369](#)
 - sdGetWouldBlock, [377](#)
 - sdIncomingDataI, [374](#)
 - sdInit, [372](#)
 - sdObjectInit, [373](#)
 - sdPut, [368](#)
 - sdPutTimeout, [369](#)
 - sdPutWouldBlock, [377](#)
 - sdRead, [370](#)
 - sdReadTimeout, [371](#)
 - sdRequestDataI, [376](#)
 - sdStart, [373](#)
 - sdStop, [374](#)
 - sdWrite, [369](#)
 - sdWriteTimeout, [370](#)
 - sdstate_t, [372](#)
 - SerialDriver, [372](#)
- Serial over USB Driver, [380](#)
 - _serial_usb_driver_data, [382](#)
 - _serial_usb_driver_methods, [382](#)
 - ibnotify, [383](#)
 - obnotify, [383](#)
 - SDU_READY, [383](#)
 - SDU_STOP, [383](#)
 - SDU_UNINIT, [383](#)
 - SERIAL_USB_BUFFERS_NUMBER, [382](#)

- SERIAL_USB_BUFFERS_SIZE, 382
- sduConfigureHookI, 386
- sduDataReceived, 388
- sduDataTransmitted, 387
- sduInit, 383
- sduInterruptTransmitted, 388
- sduObjectInit, 384
- sduRequestsHook, 386
- sduSOFHookI, 387
- sduStart, 384
- sduStop, 385
- sduSuspendHookI, 385
- sduWakeupHookI, 385
- sdustate_t, 383
- SerialUSBDriver, 382
- SerialConfig, 567
 - speed, 568
- SerialDriver, 568
 - Serial Driver, 372
 - vmt, 569
- SerialDriverVMT, 569
- SerialUSBConfig, 571
 - bulk_in, 572
 - bulk_out, 572
 - int_in, 572
 - usbp, 572
- SerialUSBDriver, 572
 - Serial over USB Driver, 382
 - vmt, 574
- SerialUSBDriverVMT, 574
- set_address
 - USB Driver, 463
- setup
 - USBDriver, 590
- setup_cb
 - USBEndpointConfig, 593
- size
 - I2SConfig, 530
 - MACReceiveDescriptor, 541
 - MACTransmitDescriptor, 542
- sleep_event
 - CANDriver, 510
- sof_cb
 - USBConfig, 587
- speed
 - SerialConfig, 568
- spi_llc_exchange
 - SPI Driver, 409
- spi_llc_ignore
 - SPI Driver, 409
- spi_llc_init
 - SPI Driver, 407
- spi_llc_polled_exchange
 - SPI Driver, 410
- spi_llc_receive
 - SPI Driver, 410
- spi_llc_select
 - SPI Driver, 408
- spi_llc_send
 - SPI Driver, 410
- spi_llc_start
 - SPI Driver, 408
- spi_llc_stop
 - SPI Driver, 408
- spi_llc_unselect
 - SPI Driver, 408
- spiAcquireBus
 - SPI Driver, 407
- spiExchange
 - SPI Driver, 405
- spiIgnore
 - SPI Driver, 405
- spiInit
 - SPI Driver, 399
- spiObjectInit
 - SPI Driver, 400
- spiPolledExchange
 - SPI Driver, 397
- spiReceive
 - SPI Driver, 406
- spiReleaseBus
 - SPI Driver, 407
- spiSelect
 - SPI Driver, 401
- spiSelectI
 - SPI Driver, 394
- spiSend
 - SPI Driver, 406
- spiStart
 - SPI Driver, 400
- spiStartExchange
 - SPI Driver, 402
- spiStartExchangeI
 - SPI Driver, 395
- spiStartIgnore
 - SPI Driver, 402
- spiStartIgnoreI
 - SPI Driver, 395
- spiStartReceive
 - SPI Driver, 404
- spiStartReceiveI
 - SPI Driver, 397
- spiStartSend
 - SPI Driver, 404
- spiStartSendI
 - SPI Driver, 396
- spiStop
 - SPI Driver, 401
- spiUnselect
 - SPI Driver, 401
- spiUnselectI
 - SPI Driver, 394
- spicallback_t
 - SPI Driver, 399
- spip
 - MMCCConfig, 543

- spistate_t
 - SPI Driver, [399](#)
- st_llid_get_alarm
 - ST Driver, [417](#)
- st_llid_get_counter
 - ST Driver, [416](#)
- st_llid_init
 - ST Driver, [415](#)
- st_llid_is_alarm_active
 - ST Driver, [417](#)
- st_llid_set_alarm
 - ST Driver, [416](#)
- st_llid_start_alarm
 - ST Driver, [416](#)
- st_llid_stop_alarm
 - ST Driver, [416](#)
- stGetAlarm
 - ST Driver, [415](#)
- stGetCounter
 - ST Driver, [412](#)
- stInit
 - ST Driver, [413](#)
- stIsAlarmActive
 - ST Driver, [413](#)
- stSetAlarm
 - ST Driver, [414](#)
- stStartAlarm
 - ST Driver, [413](#)
- stStopAlarm
 - ST Driver, [414](#)
- state
 - ADCDriver, [493](#)
 - CANDriver, [509](#)
 - DACDriver, [518](#)
 - EXTDriver, [521](#)
 - GPTDriver, [526](#)
 - I2CDriver, [528](#)
 - I2SDriver, [531](#)
 - ICUDriver, [533](#)
 - MACDriver, [540](#)
 - PWMDriver, [555](#)
 - QSPIDriver, [558](#)
 - SPIDriver, [577](#)
 - UARTDriver, [581](#)
 - USBDriver, [589](#)
 - WDGDriver, [597](#)
- status
 - USBDriver, [590](#)
- streamGet
 - Abstract Streams, [149](#)
- streamPut
 - Abstract Streams, [148](#)
- streamRead
 - Abstract Streams, [148](#)
- streamWrite
 - Abstract Streams, [148](#)
- Support Code, [217](#)
- suspended
 - io_buffers_queue, [535](#)
- sync
 - MMC over SPI Driver, [238](#)
- TIME
 - CANRxFrame, [511](#)
- tdqueue
 - MACDriver, [540](#)
- thread
 - ADCDriver, [493](#)
 - DACDriver, [518](#)
 - QSPIDriver, [558](#)
 - SPIDriver, [578](#)
 - USBInEndpointState, [595](#)
 - USBOutEndpointState, [596](#)
- threadrx
 - UARTDriver, [581](#)
- threadtx
 - UARTDriver, [581](#)
- top
 - io_buffers_queue, [536](#)
- transmitting
 - USBDriver, [589](#)
- tx_buffer
 - I2SConfig, [529](#)
- txbuf
 - USBInEndpointState, [595](#)
- txcnt
 - USBInEndpointState, [595](#)
- txempty_event
 - CANDriver, [509](#)
- txend1_cb
 - UARTConfig, [579](#)
- txend2_cb
 - UARTConfig, [579](#)
- txqueue
 - CANDriver, [509](#)
- txsize
 - USBInEndpointState, [595](#)
- txstate
 - UARTDriver, [581](#)
- UART Driver, [418](#)
 - _uart_rx_complete_isr_code, [426](#)
 - _uart_rx_error_isr_code, [427](#)
 - _uart_rx_idle_code, [427](#)
 - _uart_timeout_isr_code, [428](#)
 - _uart_tx1_isr_code, [425](#)
 - _uart_tx2_isr_code, [425](#)
 - _uart_wakeup_rx_complete_isr, [424](#)
 - _uart_wakeup_rx_error_isr, [424](#)
 - _uart_wakeup_rx_timeout_isr, [424](#)
 - _uart_wakeup_tx1_isr, [423](#)
 - _uart_wakeup_tx2_isr, [423](#)
 - PLATFORM_UART_USE_UART1, [428](#)
 - UART_BREAK_DETECTED, [422](#)
 - UART_FRAMING_ERROR, [422](#)
 - UART_NO_ERROR, [422](#)
 - UART_NOISE_ERROR, [422](#)

- UART_OVERRUN_ERROR, [422](#)
- UART_PARITY_ERROR, [422](#)
- UART_READY, [429](#)
- UART_RX_ACTIVE, [430](#)
- UART_RX_COMPLETE, [430](#)
- UART_RX_IDLE, [430](#)
- UART_STOP, [429](#)
- UART_TX_ACTIVE, [429](#)
- UART_TX_COMPLETE, [430](#)
- UART_TX_IDLE, [429](#)
- UART_UNINIT, [429](#)
- UART_USE_MUTUAL_EXCLUSION, [423](#)
- UART_USE_WAIT, [423](#)
- UARTD1, [443](#)
- UARTDriver, [428](#)
- uart_lld_init, [440](#)
- uart_lld_start, [441](#)
- uart_lld_start_receive, [442](#)
- uart_lld_start_send, [441](#)
- uart_lld_stop, [441](#)
- uart_lld_stop_receive, [442](#)
- uart_lld_stop_send, [442](#)
- uartAcquireBus, [440](#)
- uartInit, [430](#)
- uartObjectInit, [430](#)
- uartReceiveTimeout, [439](#)
- uartReleaseBus, [440](#)
- uartSendFullTimeout, [438](#)
- uartSendTimeout, [437](#)
- uartStart, [430](#)
- uartStartReceive, [434](#)
- uartStartReceiveI, [435](#)
- uartStartSend, [431](#)
- uartStartSendI, [432](#)
- uartStop, [431](#)
- uartStopReceive, [435](#)
- uartStopReceiveI, [436](#)
- uartStopSend, [433](#)
- uartStopSendI, [433](#)
- uartcb_t, [429](#)
- uartccb_t, [429](#)
- uartecb_t, [429](#)
- uartflags_t, [428](#)
- uartrxstate_t, [430](#)
- uartstate_t, [429](#)
- uarttxstate_t, [429](#)
- UART_BREAK_DETECTED
 - UART Driver, [422](#)
- UART_FRAMING_ERROR
 - UART Driver, [422](#)
- UART_NO_ERROR
 - UART Driver, [422](#)
- UART_NOISE_ERROR
 - UART Driver, [422](#)
- UART_OVERRUN_ERROR
 - UART Driver, [422](#)
- UART_PARITY_ERROR
 - UART Driver, [422](#)
- UART_READY
 - UART Driver, [429](#)
- UART_RX_ACTIVE
 - UART Driver, [430](#)
- UART_RX_COMPLETE
 - UART Driver, [430](#)
- UART_RX_IDLE
 - UART Driver, [430](#)
- UART_STOP
 - UART Driver, [429](#)
- UART_TX_ACTIVE
 - UART Driver, [429](#)
- UART_TX_COMPLETE
 - UART Driver, [430](#)
- UART_TX_IDLE
 - UART Driver, [429](#)
- UART_UNINIT
 - UART Driver, [429](#)
- UART_USE_MUTUAL_EXCLUSION
 - Configuration, [212](#)
 - UART Driver, [423](#)
- UART_USE_WAIT
 - Configuration, [212](#)
 - UART Driver, [423](#)
- UARTConfig, [578](#)
 - rxchar_cb, [579](#)
 - rxend_cb, [579](#)
 - rxerr_cb, [579](#)
 - txend1_cb, [579](#)
 - txend2_cb, [579](#)
- UARTD1
 - UART Driver, [443](#)
- UARTDriver, [579](#)
 - config, [581](#)
 - early, [581](#)
 - mutex, [581](#)
 - rxstate, [581](#)
 - state, [581](#)
 - threadrx, [581](#)
 - threadtx, [581](#)
 - txstate, [581](#)
 - UART Driver, [428](#)
- USB CDC Header, [482](#)
- USB Driver, [444](#)
 - _usb_ep0in, [473](#)
 - _usb_ep0out, [474](#)
 - _usb_ep0setup, [472](#)
 - _usb_isr_invoke_event_cb, [457](#)
 - _usb_isr_invoke_in_cb, [458](#)
 - _usb_isr_invoke_out_cb, [458](#)
 - _usb_isr_invoke_setup_cb, [458](#)
 - _usb_isr_invoke_sof_cb, [457](#)
 - _usb_reset, [471](#)
 - _usb_suspend, [471](#)
 - _usb_wakeup, [472](#)
 - default_handler, [463](#)
 - EP_STATUS_ACTIVE, [462](#)
 - EP_STATUS_DISABLED, [462](#)

- EP_STATUS_STALLED, [462](#)
- ep0_state, [480](#)
- ep0config, [480](#)
- in, [480](#)
- out, [480](#)
- PLATFORM_USB_USE_USB1, [459](#)
- set_address, [463](#)
- USB_ACTIVE, [462](#)
- USB_DESC_BCD, [451](#)
- USB_DESC_BYTE, [450](#)
- USB_DESC_CONFIGURATION_SIZE, [451](#)
- USB_DESC_CONFIGURATION, [451](#)
- USB_DESC_DEVICE, [451](#)
- USB_DESC_ENDPOINT_SIZE, [453](#)
- USB_DESC_ENDPOINT, [453](#)
- USB_DESC_INDEX, [450](#)
- USB_DESC_INTERFACE_ASSOCIATION_SIZE, [452](#)
- USB_DESC_INTERFACE_ASSOCIATION, [452](#)
- USB_DESC_INTERFACE_SIZE, [452](#)
- USB_DESC_INTERFACE, [452](#)
- USB_DESC_WORD, [451](#)
- USB_EP0_ERROR, [462](#)
- USB_EP0_RX, [462](#)
- USB_EP0_SENDING_STS, [462](#)
- USB_EP0_STATUS_STAGE, [459](#)
- USB_EP0_TX, [462](#)
- USB_EP0_WAITING_SETUP, [462](#)
- USB_EP0_WAITING_STS, [462](#)
- USB_EP0_WAITING_TX0, [462](#)
- USB_EP_MODE_TYPE_BULK, [453](#)
- USB_EP_MODE_TYPE_CTRL, [453](#)
- USB_EP_MODE_TYPE_INTR, [453](#)
- USB_EP_MODE_TYPE_ISOC, [453](#)
- USB_EP_MODE_TYPE, [453](#)
- USB_EVENT_ADDRESS, [462](#)
- USB_EVENT_CONFIGURED, [463](#)
- USB_EVENT_RESET, [462](#)
- USB_EVENT_STALLED, [463](#)
- USB_EVENT_SUSPEND, [463](#)
- USB_EVENT_UNCONFIGURED, [463](#)
- USB_EVENT_WAKEUP, [463](#)
- USB_MAX_ENDPOINTS, [459](#)
- USB_READY, [462](#)
- USB_SELECTED, [462](#)
- USB_SET_ADDRESS_ACK_HANDLING, [459](#)
- USB_SET_ADDRESS_MODE, [459](#)
- USB_STOP, [462](#)
- USB_SUSPENDED, [462](#)
- USB_UNINIT, [462](#)
- USB_USE_WAIT, [453](#)
- USB1, [480](#)
- USBDriver, [460](#)
- usb_llc_clear_in, [480](#)
- usb_llc_clear_out, [479](#)
- usb_llc_connect_bus, [460](#)
- usb_llc_disable_endpoints, [476](#)
- usb_llc_disconnect_bus, [460](#)
- usb_llc_get_frame_number, [459](#)
- usb_llc_get_status_in, [477](#)
- usb_llc_get_status_out, [477](#)
- usb_llc_get_transaction_size, [460](#)
- usb_llc_init, [474](#)
- usb_llc_init_endpoint, [476](#)
- usb_llc_prepare_receive, [478](#)
- usb_llc_prepare_transmit, [478](#)
- usb_llc_read_setup, [477](#)
- usb_llc_reset, [475](#)
- usb_llc_set_address, [476](#)
- usb_llc_stall_in, [479](#)
- usb_llc_stall_out, [479](#)
- usb_llc_start, [475](#)
- usb_llc_start_in, [479](#)
- usb_llc_start_out, [478](#)
- usb_llc_stop, [475](#)
- usbConnectBus, [454](#)
- usbDisableEndpointsI, [466](#)
- usbDisconnectBus, [454](#)
- usbGetDriverStateI, [454](#)
- usbGetFrameNumberX, [454](#)
- usbGetReceiveStatusI, [455](#)
- usbGetReceiveTransactionSizeX, [456](#)
- usbGetTransmitStatusI, [455](#)
- usbInit, [464](#)
- usbInitEndpointI, [466](#)
- usbObjectInit, [464](#)
- usbReadSetup, [456](#)
- usbReceive, [468](#)
- usbSetupTransfer, [456](#)
- usbStallReceiveI, [469](#)
- usbStallTransmitI, [470](#)
- usbStart, [465](#)
- usbStartReceiveI, [467](#)
- usbStartTransmitI, [467](#)
- usbStop, [465](#)
- usbTransmit, [469](#)
- usbcallback_t, [461](#)
- usbep0state_t, [462](#)
- usbep_t, [460](#)
- usbepcallback_t, [461](#)
- usbepstatus_t, [462](#)
- usbevent_t, [462](#)
- usbeventcb_t, [461](#)
- usbgetdescriptor_t, [461](#)
- usbreqhandler_t, [461](#)
- usbstate_t, [462](#)
- USB_ACTIVE
 - USB Driver, [462](#)
- USB_DESC_BCD
 - USB Driver, [451](#)
- USB_DESC_BYTE
 - USB Driver, [450](#)
- USB_DESC_CONFIGURATION_SIZE
 - USB Driver, [451](#)
- USB_DESC_CONFIGURATION
 - USB Driver, [451](#)

- USB_DESC_DEVICE
 - USB Driver, [451](#)
- USB_DESC_ENDPOINT_SIZE
 - USB Driver, [453](#)
- USB_DESC_ENDPOINT
 - USB Driver, [453](#)
- USB_DESC_INDEX
 - USB Driver, [450](#)
- USB_DESC_INTERFACE_ASSOCIATION_SIZE
 - USB Driver, [452](#)
- USB_DESC_INTERFACE_ASSOCIATION
 - USB Driver, [452](#)
- USB_DESC_INTERFACE_SIZE
 - USB Driver, [452](#)
- USB_DESC_INTERFACE
 - USB Driver, [452](#)
- USB_DESC_WORD
 - USB Driver, [451](#)
- USB_EP0_ERROR
 - USB Driver, [462](#)
- USB_EP0_RX
 - USB Driver, [462](#)
- USB_EP0_SENDING_STS
 - USB Driver, [462](#)
- USB_EP0_STATUS_STAGE
 - USB Driver, [459](#)
- USB_EP0_TX
 - USB Driver, [462](#)
- USB_EP0_WAITING_SETUP
 - USB Driver, [462](#)
- USB_EP0_WAITING_STS
 - USB Driver, [462](#)
- USB_EP0_WAITING_TX0
 - USB Driver, [462](#)
- USB_EP_MODE_TYPE_BULK
 - USB Driver, [453](#)
- USB_EP_MODE_TYPE_CTRL
 - USB Driver, [453](#)
- USB_EP_MODE_TYPE_INTR
 - USB Driver, [453](#)
- USB_EP_MODE_TYPE_ISOC
 - USB Driver, [453](#)
- USB_EP_MODE_TYPE
 - USB Driver, [453](#)
- USB_EVENT_ADDRESS
 - USB Driver, [462](#)
- USB_EVENT_CONFIGURED
 - USB Driver, [463](#)
- USB_EVENT_RESET
 - USB Driver, [462](#)
- USB_EVENT_STALLED
 - USB Driver, [463](#)
- USB_EVENT_SUSPEND
 - USB Driver, [463](#)
- USB_EVENT_UNCONFIGURED
 - USB Driver, [463](#)
- USB_EVENT_WAKEUP
 - USB Driver, [463](#)
- USB_MAX_ENDPOINTS
 - USB Driver, [459](#)
- USB_READY
 - USB Driver, [462](#)
- USB_SELECTED
 - USB Driver, [462](#)
- USB_SET_ADDRESS_ACK_HANDLING
 - USB Driver, [459](#)
- USB_SET_ADDRESS_MODE
 - USB Driver, [459](#)
- USB_STOP
 - USB Driver, [462](#)
- USB_SUSPENDED
 - USB Driver, [462](#)
- USB_UNINIT
 - USB Driver, [462](#)
- USB_USE_WAIT
 - Configuration, [212](#)
 - USB Driver, [453](#)
- USBConfig, [585](#)
 - event_cb, [587](#)
 - get_descriptor_cb, [587](#)
 - requests_hook_cb, [587](#)
 - sof_cb, [587](#)
- USBD1
 - USB Driver, [480](#)
- USBDescriptor, [587](#)
 - ud_size, [588](#)
 - ud_string, [588](#)
- USBDriver, [588](#)
 - address, [590](#)
 - config, [589](#)
 - configuration, [590](#)
 - ep0endcb, [590](#)
 - ep0n, [590](#)
 - ep0next, [590](#)
 - ep0state, [590](#)
 - epc, [589](#)
 - in_params, [590](#)
 - out_params, [590](#)
 - receiving, [589](#)
 - saved_state, [590](#)
 - setup, [590](#)
 - state, [589](#)
 - status, [590](#)
 - transmitting, [589](#)
 - USB Driver, [460](#)
- USBEndpointConfig, [591](#)
 - ep_mode, [593](#)
 - in_cb, [593](#)
 - in_maxsize, [593](#)
 - in_state, [594](#)
 - out_cb, [593](#)
 - out_maxsize, [594](#)
 - out_state, [594](#)
 - setup_cb, [593](#)
- USBInEndpointState, [594](#)
 - thread, [595](#)

- txbuf, [595](#)
- txcnt, [595](#)
- txsize, [595](#)
- USBOutEndpointState, [595](#)
 - rxbuf, [596](#)
 - rxcnt, [596](#)
 - rxsize, [596](#)
 - thread, [596](#)
- uart_lld_init
 - UART Driver, [440](#)
- uart_lld_start
 - UART Driver, [441](#)
- uart_lld_start_receive
 - UART Driver, [442](#)
- uart_lld_start_send
 - UART Driver, [441](#)
- uart_lld_stop
 - UART Driver, [441](#)
- uart_lld_stop_receive
 - UART Driver, [442](#)
- uart_lld_stop_send
 - UART Driver, [442](#)
- uartAcquireBus
 - UART Driver, [440](#)
- uartInit
 - UART Driver, [430](#)
- uartObjectInit
 - UART Driver, [430](#)
- uartReceiveTimeout
 - UART Driver, [439](#)
- uartReleaseBus
 - UART Driver, [440](#)
- uartSendFullTimeout
 - UART Driver, [438](#)
- uartSendTimeout
 - UART Driver, [437](#)
- uartStart
 - UART Driver, [430](#)
- uartStartReceive
 - UART Driver, [434](#)
- uartStartReceive1
 - UART Driver, [435](#)
- uartStartSend
 - UART Driver, [431](#)
- uartStartSend1
 - UART Driver, [432](#)
- uartStop
 - UART Driver, [431](#)
- uartStopReceive
 - UART Driver, [435](#)
- uartStopReceive1
 - UART Driver, [436](#)
- uartStopSend
 - UART Driver, [433](#)
- uartStopSend1
 - UART Driver, [433](#)
- uartcb_t
 - UART Driver, [429](#)
- uartccb_t
 - UART Driver, [429](#)
- uartecb_t
 - UART Driver, [429](#)
- uartflags_t
 - UART Driver, [428](#)
- uartrxstate_t
 - UART Driver, [430](#)
- uartstate_t
 - UART Driver, [429](#)
- uarttxstate_t
 - UART Driver, [429](#)
- ud_size
 - USBDescriptor, [588](#)
- ud_string
 - USBDescriptor, [588](#)
- unpacked_mmc_cid_t, [581](#)
- unpacked_mmc_csd_t, [582](#)
- unpacked_sdc_cid_t, [583](#)
- unpacked_sdc_csd_10_t, [583](#)
- unpacked_sdc_csd_20_t, [584](#)
- usb_lld_clear_in
 - USB Driver, [480](#)
- usb_lld_clear_out
 - USB Driver, [479](#)
- usb_lld_connect_bus
 - USB Driver, [460](#)
- usb_lld_disable_endpoints
 - USB Driver, [476](#)
- usb_lld_disconnect_bus
 - USB Driver, [460](#)
- usb_lld_get_frame_number
 - USB Driver, [459](#)
- usb_lld_get_status_in
 - USB Driver, [477](#)
- usb_lld_get_status_out
 - USB Driver, [477](#)
- usb_lld_get_transaction_size
 - USB Driver, [460](#)
- usb_lld_init
 - USB Driver, [474](#)
- usb_lld_init_endpoint
 - USB Driver, [476](#)
- usb_lld_prepare_receive
 - USB Driver, [478](#)
- usb_lld_prepare_transmit
 - USB Driver, [478](#)
- usb_lld_read_setup
 - USB Driver, [477](#)
- usb_lld_reset
 - USB Driver, [475](#)
- usb_lld_set_address
 - USB Driver, [476](#)
- usb_lld_stall_in
 - USB Driver, [479](#)
- usb_lld_stall_out
 - USB Driver, [479](#)
- usb_lld_start

- USB Driver, [475](#)
- usb_lld_start_in
 - USB Driver, [479](#)
- usb_lld_start_out
 - USB Driver, [478](#)
- usb_lld_stop
 - USB Driver, [475](#)
- usbConnectBus
 - USB Driver, [454](#)
- usbDisableEndpointsI
 - USB Driver, [466](#)
- usbDisconnectBus
 - USB Driver, [454](#)
- usbGetDriverStateI
 - USB Driver, [454](#)
- usbGetFrameNumberX
 - USB Driver, [454](#)
- usbGetReceiveStatusI
 - USB Driver, [455](#)
- usbGetReceiveTransactionSizeX
 - USB Driver, [456](#)
- usbGetTransmitStatusI
 - USB Driver, [455](#)
- usbInit
 - USB Driver, [464](#)
- usbInitEndpointI
 - USB Driver, [466](#)
- usbObjectInit
 - USB Driver, [464](#)
- usbReadSetup
 - USB Driver, [456](#)
- usbReceive
 - USB Driver, [468](#)
- usbSetupTransfer
 - USB Driver, [456](#)
- usbStallReceiveI
 - USB Driver, [469](#)
- usbStallTransmitI
 - USB Driver, [470](#)
- usbStart
 - USB Driver, [465](#)
- usbStartReceiveI
 - USB Driver, [467](#)
- usbStartTransmitI
 - USB Driver, [467](#)
- usbStop
 - USB Driver, [465](#)
- usbTransmit
 - USB Driver, [469](#)
- usbcallback_t
 - USB Driver, [461](#)
- usbep0state_t
 - USB Driver, [462](#)
- usbep_t
 - USB Driver, [460](#)
- usbepcallback_t
 - USB Driver, [461](#)
- usbepstatus_t

- USB Driver, [462](#)
- usbevent_t
 - USB Driver, [462](#)
- usbeventcb_t
 - USB Driver, [461](#)
- usbgetdescriptor_t
 - USB Driver, [461](#)
- usbp
 - SerialUSBConfig, [572](#)
- usbreqhandler_t
 - USB Driver, [461](#)
- usbstate_t
 - USB Driver, [462](#)
- vmt
 - BaseAsynchronousChannel, [495](#)
 - BaseBlockDevice, [499](#)
 - BaseChannel, [502](#)
 - BaseSequentialStream, [505](#)
 - FileStream, [523](#)
 - MMCDriver, [545](#)
 - MMCSDBlockDevice, [549](#)
 - RTCDriver, [561](#)
 - SDCDriver, [565](#)
 - SerialDriver, [569](#)
 - SerialUSBDriver, [574](#)
- WDG Driver, [484](#)
 - PLATFORM_WDG_USE_WDG1, [485](#)
 - WDG_READY, [485](#)
 - WDG_STOP, [485](#)
 - WDG_UNINIT, [485](#)
 - WDGDriver, [485](#)
 - wdg_lld_init, [487](#)
 - wdg_lld_reset, [488](#)
 - wdg_lld_start, [487](#)
 - wdg_lld_stop, [487](#)
 - wdgInit, [485](#)
 - wdgReset, [487](#)
 - wdgResetI, [485](#)
 - wdgStart, [486](#)
 - wdgStop, [486](#)
 - wdgstate_t, [485](#)
- WDG_READY
 - WDG Driver, [485](#)
- WDG_STOP
 - WDG Driver, [485](#)
- WDG_UNINIT
 - WDG Driver, [485](#)
- WDGConfig, [596](#)
- WDGDriver, [597](#)
 - config, [597](#)
 - state, [597](#)
 - WDG Driver, [485](#)
- wait
 - MMC over SPI Driver, [233](#)
- waiting
 - io_buffers_queue, [535](#)
- wakeup_event

- CANDriver, [510](#)
- wdg_llid_init
 - WDG Driver, [487](#)
- wdg_llid_reset
 - WDG Driver, [488](#)
- wdg_llid_start
 - WDG Driver, [487](#)
- wdg_llid_stop
 - WDG Driver, [487](#)
- wdgInit
 - WDG Driver, [485](#)
- wdgReset
 - WDG Driver, [487](#)
- wdgResetI
 - WDG Driver, [485](#)
- wdgStart
 - WDG Driver, [486](#)
- wdgStop
 - WDG Driver, [486](#)
- wdgstate_t
 - WDG Driver, [485](#)
- width_cb
 - ICUConfig, [532](#)
- year
 - RTCDateTime, [560](#)