

ChibiOS/RT 2.4.0

STM32F1xx HAL Reference Manual

# Contents

<b>1 ChibiOS/RT</b>	<b>1</b>
1.1 Copyright . . . . .	1
1.2 Introduction . . . . .	1
1.3 Related Documents . . . . .	1
<b>2 Deprecated List</b>	<b>2</b>
<b>3 Module Index</b>	<b>3</b>
3.1 Modules . . . . .	3
<b>4 Data Structure Index</b>	<b>5</b>
4.1 Data Structures . . . . .	5
<b>5 File Index</b>	<b>7</b>
5.1 File List . . . . .	7
<b>6 Module Documentation</b>	<b>9</b>
6.1 HAL . . . . .	9
6.1.1 Detailed Description . . . . .	9
6.1.2 HAL Device Drivers Architecture . . . . .	9
6.1.2.1 Diagram . . . . .	10
6.2 Configuration . . . . .	11
6.2.1 Detailed Description . . . . .	11
6.2.2 Define Documentation . . . . .	13
6.2.2.1 HAL_USE_PAL . . . . .	13
6.2.2.2 HAL_USE_ADC . . . . .	13
6.2.2.3 HAL_USE_CAN . . . . .	13
6.2.2.4 HAL_USE_EXT . . . . .	13
6.2.2.5 HAL_USE_GPT . . . . .	13
6.2.2.6 HAL_USE_I2C . . . . .	14
6.2.2.7 HAL_USE_ICU . . . . .	14
6.2.2.8 HAL_USE_MAC . . . . .	14
6.2.2.9 HAL_USE_MMC_SPI . . . . .	14

6.2.2.10 HAL_USE_PWM . . . . .	14
6.2.2.11 HAL_USE_RTC . . . . .	14
6.2.2.12 HAL_USE_SDC . . . . .	14
6.2.2.13 HAL_USE_SERIAL . . . . .	14
6.2.2.14 HAL_USE_SERIAL_USB . . . . .	14
6.2.2.15 HAL_USE_SPI . . . . .	14
6.2.2.16 HAL_USE_UART . . . . .	14
6.2.2.17 HAL_USE_USB . . . . .	14
6.2.2.18 ADC_USE_WAIT . . . . .	15
6.2.2.19 ADC_USE_MUTUAL_EXCLUSION . . . . .	15
6.2.2.20 CAN_USE_SLEEP_MODE . . . . .	15
6.2.2.21 I2C_USE_MUTUAL_EXCLUSION . . . . .	15
6.2.2.22 MAC_USE_EVENTS . . . . .	15
6.2.2.23 MMC_SECTOR_SIZE . . . . .	15
6.2.2.24 MMC_NICE_WAITING . . . . .	15
6.2.2.25 MMC_POLLING_INTERVAL . . . . .	15
6.2.2.26 MMC_POLLING_DELAY . . . . .	15
6.2.2.27 MMC_USE_SPI_POLLING . . . . .	15
6.2.2.28 SDC_INIT_RETRY . . . . .	16
6.2.2.29 SDC_MMC_SUPPORT . . . . .	16
6.2.2.30 SDC_NICE_WAITING . . . . .	16
6.2.2.31 SERIAL_DEFAULT_BITRATE . . . . .	16
6.2.2.32 SERIAL_BUFFERS_SIZE . . . . .	16
6.2.2.33 SERIAL_USB_BUFFERS_SIZE . . . . .	16
6.2.2.34 SPI_USE_WAIT . . . . .	16
6.2.2.35 SPI_USE_MUTUAL_EXCLUSION . . . . .	17
<b>6.3 ADC Driver . . . . .</b>	<b>17</b>
6.3.1 Detailed Description . . . . .	17
6.3.2 Driver State Machine . . . . .	17
6.3.3 ADC Operations . . . . .	17
6.3.3.1 ADC Conversion Groups . . . . .	17
6.3.3.2 ADC Conversion Modes . . . . .	18
6.3.3.3 ADC Callbacks . . . . .	18
6.3.4 Function Documentation . . . . .	23
6.3.4.1 adcInit . . . . .	23
6.3.4.2 adcObjectInit . . . . .	23
6.3.4.3 adcStart . . . . .	24
6.3.4.4 adcStop . . . . .	24
6.3.4.5 adcStartConversion . . . . .	25
6.3.4.6 adcStartConversionl . . . . .	25

6.3.4.7	adcStopConversion . . . . .	26
6.3.4.8	adcStopConversionl . . . . .	26
6.3.4.9	adcAcquireBus . . . . .	27
6.3.4.10	adcReleaseBus . . . . .	27
6.3.4.11	adcConvert . . . . .	27
6.3.4.12	adc_lld_init . . . . .	28
6.3.4.13	adc_lld_start . . . . .	28
6.3.4.14	adc_lld_stop . . . . .	29
6.3.4.15	adc_lld_start_conversion . . . . .	29
6.3.4.16	adc_lld_stop_conversion . . . . .	30
6.3.5	Variable Documentation . . . . .	30
6.3.5.1	ADCD1 . . . . .	30
6.3.6	Define Documentation . . . . .	30
6.3.6.1	ADC_USE_WAIT . . . . .	30
6.3.6.2	ADC_USE_MUTUAL_EXCLUSION . . . . .	30
6.3.6.3	_adc_reset_i . . . . .	30
6.3.6.4	_adc_reset_s . . . . .	31
6.3.6.5	_adc_wakeup_isr . . . . .	31
6.3.6.6	_adc_timeout_isr . . . . .	31
6.3.6.7	_adc_isr_half_code . . . . .	32
6.3.6.8	_adc_isr_full_code . . . . .	32
6.3.6.9	_adc_isr_error_code . . . . .	33
6.3.6.10	ADC_CR2_EXTSEL_SRC . . . . .	33
6.3.6.11	ADC_CR2_EXTSEL_SWSTART . . . . .	33
6.3.6.12	ADC_CHANNEL_IN0 . . . . .	33
6.3.6.13	ADC_CHANNEL_IN1 . . . . .	33
6.3.6.14	ADC_CHANNEL_IN2 . . . . .	33
6.3.6.15	ADC_CHANNEL_IN3 . . . . .	34
6.3.6.16	ADC_CHANNEL_IN4 . . . . .	34
6.3.6.17	ADC_CHANNEL_IN5 . . . . .	34
6.3.6.18	ADC_CHANNEL_IN6 . . . . .	34
6.3.6.19	ADC_CHANNEL_IN7 . . . . .	34
6.3.6.20	ADC_CHANNEL_IN8 . . . . .	34
6.3.6.21	ADC_CHANNEL_IN9 . . . . .	34
6.3.6.22	ADC_CHANNEL_IN10 . . . . .	34
6.3.6.23	ADC_CHANNEL_IN11 . . . . .	34
6.3.6.24	ADC_CHANNEL_IN12 . . . . .	34
6.3.6.25	ADC_CHANNEL_IN13 . . . . .	34
6.3.6.26	ADC_CHANNEL_IN14 . . . . .	34
6.3.6.27	ADC_CHANNEL_IN15 . . . . .	35

6.3.6.28	ADC_CHANNEL_SENSOR . . . . .	35
6.3.6.29	ADC_CHANNEL_VREFINT . . . . .	35
6.3.6.30	ADC_SAMPLE_1P5 . . . . .	35
6.3.6.31	ADC_SAMPLE_7P5 . . . . .	35
6.3.6.32	ADC_SAMPLE_13P5 . . . . .	35
6.3.6.33	ADC_SAMPLE_28P5 . . . . .	35
6.3.6.34	ADC_SAMPLE_41P5 . . . . .	35
6.3.6.35	ADC_SAMPLE_55P5 . . . . .	35
6.3.6.36	ADC_SAMPLE_71P5 . . . . .	35
6.3.6.37	ADC_SAMPLE_239P5 . . . . .	35
6.3.6.38	STM32_ADC_USE_ADC1 . . . . .	35
6.3.6.39	STM32_ADC_ADC1_DMA_PRIORITY . . . . .	36
6.3.6.40	STM32_ADC_ADC1_IRQ_PRIORITY . . . . .	36
6.3.6.41	ADC_SQR1_NUM_CH . . . . .	36
6.3.6.42	ADC_SQR3_SQ1_N . . . . .	36
6.3.6.43	ADC_SQR3_SQ2_N . . . . .	36
6.3.6.44	ADC_SQR3_SQ3_N . . . . .	36
6.3.6.45	ADC_SQR3_SQ4_N . . . . .	36
6.3.6.46	ADC_SQR3_SQ5_N . . . . .	36
6.3.6.47	ADC_SQR3_SQ6_N . . . . .	36
6.3.6.48	ADC_SQR2_SQ7_N . . . . .	36
6.3.6.49	ADC_SQR2_SQ8_N . . . . .	36
6.3.6.50	ADC_SQR2_SQ9_N . . . . .	37
6.3.6.51	ADC_SQR2_SQ10_N . . . . .	37
6.3.6.52	ADC_SQR2_SQ11_N . . . . .	37
6.3.6.53	ADC_SQR2_SQ12_N . . . . .	37
6.3.6.54	ADC_SQR1_SQ13_N . . . . .	37
6.3.6.55	ADC_SQR1_SQ14_N . . . . .	37
6.3.6.56	ADC_SQR1_SQ15_N . . . . .	37
6.3.6.57	ADC_SQR1_SQ16_N . . . . .	37
6.3.6.58	ADC_SMPR2_SMP_AN0 . . . . .	37
6.3.6.59	ADC_SMPR2_SMP_AN1 . . . . .	37
6.3.6.60	ADC_SMPR2_SMP_AN2 . . . . .	37
6.3.6.61	ADC_SMPR2_SMP_AN3 . . . . .	37
6.3.6.62	ADC_SMPR2_SMP_AN4 . . . . .	38
6.3.6.63	ADC_SMPR2_SMP_AN5 . . . . .	38
6.3.6.64	ADC_SMPR2_SMP_AN6 . . . . .	38
6.3.6.65	ADC_SMPR2_SMP_AN7 . . . . .	38
6.3.6.66	ADC_SMPR2_SMP_AN8 . . . . .	38
6.3.6.67	ADC_SMPR2_SMP_AN9 . . . . .	38

6.3.6.68	ADC_SMPR1_SMP_AN10	38
6.3.6.69	ADC_SMPR1_SMP_AN11	38
6.3.6.70	ADC_SMPR1_SMP_AN12	38
6.3.6.71	ADC_SMPR1_SMP_AN13	38
6.3.6.72	ADC_SMPR1_SMP_AN14	38
6.3.6.73	ADC_SMPR1_SMP_AN15	38
6.3.6.74	ADC_SMPR1_SMP_SENSOR	39
6.3.6.75	ADC_SMPR1_SMP_VREF	39
6.3.7	Typedef Documentation	39
6.3.7.1	adcsample_t	39
6.3.7.2	adc_channels_num_t	39
6.3.7.3	ADCDriver	39
6.3.7.4	adccallback_t	39
6.3.7.5	adcerrorcallback_t	39
6.3.8	Enumeration Type Documentation	39
6.3.8.1	adcstate_t	39
6.3.8.2	adcerror_t	40
6.4	CAN Driver	40
6.4.1	Detailed Description	40
6.4.2	Driver State Machine	40
6.4.3	Function Documentation	43
6.4.3.1	canInit	43
6.4.3.2	canObjectInit	44
6.4.3.3	canStart	44
6.4.3.4	canStop	44
6.4.3.5	canTransmit	45
6.4.3.6	canReceive	46
6.4.3.7	canGetAndClearFlags	47
6.4.3.8	canSleep	47
6.4.3.9	canWakeup	48
6.4.3.10	CH_IRQ_HANDLER	48
6.4.3.11	CH_IRQ_HANDLER	48
6.4.3.12	CH_IRQ_HANDLER	49
6.4.3.13	can_lld_init	49
6.4.3.14	can_lld_start	49
6.4.3.15	can_lld_stop	49
6.4.3.16	can_lld_can_transmit	49
6.4.3.17	can_lld_transmit	50
6.4.3.18	can_lld_can_receive	50
6.4.3.19	can_lld_receive	50

6.4.3.20	can_lld_sleep	51
6.4.3.21	can_lld_wakeup	51
6.4.4	Variable Documentation	51
6.4.4.1	CAND1	51
6.4.5	Define Documentation	51
6.4.5.1	CAN_LIMIT_WARNING	51
6.4.5.2	CAN_LIMIT_ERROR	51
6.4.5.3	CAN_BUS_OFF_ERROR	51
6.4.5.4	CAN_FRAMING_ERROR	51
6.4.5.5	CAN_OVERFLOW_ERROR	52
6.4.5.6	CAN_USE_SLEEP_MODE	52
6.4.5.7	canAddFlags1	52
6.4.5.8	CAN_SUPPORTS_SLEEP	52
6.4.5.9	CAN_MAX_FILTERS	52
6.4.5.10	CAN_BTR_BRP	52
6.4.5.11	CAN_BTR_TS1	52
6.4.5.12	CAN_BTR_TS2	52
6.4.5.13	CAN_BTR_SJW	52
6.4.5.14	CAN_IDE_STD	52
6.4.5.15	CAN_IDE_EXT	53
6.4.5.16	CAN_RTR_DATA	53
6.4.5.17	CAN_RTR_REMOTE	53
6.4.5.18	STM32_CAN_USE_CAN1	53
6.4.5.19	STM32_CAN_CAN1_IRQ_PRIORITY	53
6.4.6	Typedef Documentation	53
6.4.6.1	canstatus_t	53
6.4.7	Enumeration Type Documentation	53
6.4.7.1	canstate_t	53
6.5	EXT Driver	53
6.5.1	Detailed Description	53
6.5.2	Driver State Machine	54
6.5.3	EXT Operations.	54
6.5.4	Function Documentation	57
6.5.4.1	extInit	57
6.5.4.2	extObjectInit	57
6.5.4.3	extStart	58
6.5.4.4	extStop	58
6.5.4.5	extChannelEnable	58
6.5.4.6	extChannelDisable	59
6.5.4.7	CH_IRQ_HANDLER	59

6.5.4.8	CH_IRQ_HANDLER	59
6.5.4.9	CH_IRQ_HANDLER	59
6.5.4.10	CH_IRQ_HANDLER	59
6.5.4.11	CH_IRQ_HANDLER	60
6.5.4.12	CH_IRQ_HANDLER	60
6.5.4.13	CH_IRQ_HANDLER	60
6.5.4.14	CH_IRQ_HANDLER	60
6.5.4.15	CH_IRQ_HANDLER	60
6.5.4.16	CH_IRQ_HANDLER	60
6.5.4.17	ext_lld_init	61
6.5.4.18	ext_lld_start	61
6.5.4.19	ext_lld_stop	61
6.5.4.20	ext_lld_channel_enable	61
6.5.4.21	ext_lld_channel_disable	62
6.5.5	Variable Documentation	62
6.5.5.1	EXTD1	62
6.5.6	Define Documentation	62
6.5.6.1	EXT_MAX_CHANNELS	62
6.5.6.2	EXT_CHANNELS_MASK	62
6.5.6.3	EXT_MODE_EXTI	62
6.5.6.4	EXT_MODE_GPIOA	62
6.5.6.5	EXT_MODE_GPIOB	62
6.5.6.6	EXT_MODE_GPIOC	62
6.5.6.7	EXT_MODE_GPIOD	63
6.5.6.8	EXT_MODE_GPIOE	63
6.5.6.9	EXT_MODE_GPIOF	63
6.5.6.10	EXT_MODE_GPIOG	63
6.5.6.11	EXT_MODE_GPIOH	63
6.5.6.12	EXT_MODE_GPIOI	63
6.5.6.13	STM32_EXT_EXTI0_IRQ_PRIORITY	63
6.5.6.14	STM32_EXT_EXTI1_IRQ_PRIORITY	63
6.5.6.15	STM32_EXT_EXTI2_IRQ_PRIORITY	63
6.5.6.16	STM32_EXT_EXTI3_IRQ_PRIORITY	63
6.5.6.17	STM32_EXT_EXTI4_IRQ_PRIORITY	63
6.5.6.18	STM32_EXT_EXTI5_9_IRQ_PRIORITY	63
6.5.6.19	STM32_EXT_EXTI10_15_IRQ_PRIORITY	64
6.5.6.20	STM32_EXT_EXTI16_IRQ_PRIORITY	64
6.5.6.21	STM32_EXT_EXTI17_IRQ_PRIORITY	64
6.5.6.22	STM32_EXT_EXTI18_IRQ_PRIORITY	64
6.5.6.23	STM32_EXT_EXTI19_IRQ_PRIORITY	64

6.5.6.24	STM32_EXT EXTI20 IRQ_PRIORITY . . . . .	64
6.5.6.25	STM32_EXT EXTI21 IRQ_PRIORITY . . . . .	64
6.5.6.26	STM32_EXT EXTI22 IRQ_PRIORITY . . . . .	64
6.5.7	Typedef Documentation . . . . .	64
6.5.7.1	expchannel_t . . . . .	64
6.5.7.2	extcallback_t . . . . .	64
6.6	GPT Driver . . . . .	64
6.6.1	Detailed Description . . . . .	65
6.6.2	Driver State Machine . . . . .	65
6.6.3	GPT Operations. . . . .	65
6.6.4	Function Documentation . . . . .	68
6.6.4.1	gptInit . . . . .	68
6.6.4.2	gptObjectInit . . . . .	68
6.6.4.3	gptStart . . . . .	68
6.6.4.4	gptStop . . . . .	69
6.6.4.5	gptStartContinuous . . . . .	69
6.6.4.6	gptStartContinuousI . . . . .	70
6.6.4.7	gptStartOneShot . . . . .	70
6.6.4.8	gptStartOneShotI . . . . .	71
6.6.4.9	gptStopTimer . . . . .	71
6.6.4.10	gptStopTimerI . . . . .	72
6.6.4.11	gptPolledDelay . . . . .	72
6.6.4.12	gpt_lld_init . . . . .	73
6.6.4.13	gpt_lld_start . . . . .	73
6.6.4.14	gpt_lld_stop . . . . .	73
6.6.4.15	gpt_lld_start_timer . . . . .	74
6.6.4.16	gpt_lld_stop_timer . . . . .	74
6.6.4.17	gpt_lld_polled_delay . . . . .	74
6.6.5	Variable Documentation . . . . .	74
6.6.5.1	GPTD1 . . . . .	74
6.6.5.2	GPTD2 . . . . .	74
6.6.5.3	GPTD3 . . . . .	75
6.6.5.4	GPTD4 . . . . .	75
6.6.5.5	GPTD5 . . . . .	75
6.6.5.6	GPTD8 . . . . .	75
6.6.6	Define Documentation . . . . .	75
6.6.6.1	STM32_GPT_USE_TIM1 . . . . .	75
6.6.6.2	STM32_GPT_USE_TIM2 . . . . .	75
6.6.6.3	STM32_GPT_USE_TIM3 . . . . .	76
6.6.6.4	STM32_GPT_USE_TIM4 . . . . .	76

6.6.6.5	STM32_GPT_USE_TIM5 . . . . .	76
6.6.6.6	STM32_GPT_USE_TIM8 . . . . .	76
6.6.6.7	STM32_GPT_TIM1_IRQ_PRIORITY . . . . .	76
6.6.6.8	STM32_GPT_TIM2_IRQ_PRIORITY . . . . .	76
6.6.6.9	STM32_GPT_TIM3_IRQ_PRIORITY . . . . .	76
6.6.6.10	STM32_GPT_TIM4_IRQ_PRIORITY . . . . .	76
6.6.6.11	STM32_GPT_TIM5_IRQ_PRIORITY . . . . .	77
6.6.6.12	STM32_GPT_TIM8_IRQ_PRIORITY . . . . .	77
6.6.7	Typedef Documentation . . . . .	77
6.6.7.1	GPTDriver . . . . .	77
6.6.7.2	gptcallback_t . . . . .	77
6.6.7.3	gptfreq_t . . . . .	77
6.6.7.4	gptcnt_t . . . . .	77
6.6.8	Enumeration Type Documentation . . . . .	77
6.6.8.1	gptstate_t . . . . .	77
6.7	HAL Driver . . . . .	77
6.7.1	Detailed Description . . . . .	77
6.7.2	Function Documentation . . . . .	80
6.7.2.1	hallinit . . . . .	80
6.7.2.2	hal_lld_init . . . . .	81
6.7.2.3	stm32_clock_init . . . . .	82
6.7.3	Define Documentation . . . . .	82
6.7.3.1	S2RTT . . . . .	82
6.7.3.2	MS2RTT . . . . .	82
6.7.3.3	US2RTT . . . . .	83
6.7.3.4	halGetCounterValue . . . . .	83
6.7.3.5	halGetCounterFrequency . . . . .	83
6.7.3.6	hallsCounterWithin . . . . .	84
6.7.3.7	halPolledDelay . . . . .	85
6.7.3.8	HAL_IMPLEMENTS_COUNTERS . . . . .	85
6.7.3.9	STM32_HSICLK . . . . .	85
6.7.3.10	STM32_LSIGCLK . . . . .	85
6.7.3.11	STM32_PLS_MASK . . . . .	85
6.7.3.12	STM32_PLS_LEV0 . . . . .	85
6.7.3.13	STM32_PLS_LEV1 . . . . .	85
6.7.3.14	STM32_PLS_LEV2 . . . . .	85
6.7.3.15	STM32_PLS_LEV3 . . . . .	86
6.7.3.16	STM32_PLS_LEV4 . . . . .	86
6.7.3.17	STM32_PLS_LEV5 . . . . .	86
6.7.3.18	STM32_PLS_LEV6 . . . . .	86

6.7.3.19	STM32_PLS_LEV7	86
6.7.3.20	STM32_NO_INIT	86
6.7.3.21	STM32_PVD_ENABLE	86
6.7.3.22	STM32_PLS	86
6.7.3.23	STM32_HSI_ENABLED	86
6.7.3.24	STM32_LSI_ENABLED	86
6.7.3.25	STM32_HSE_ENABLED	86
6.7.3.26	STM32_LSE_ENABLED	86
6.7.3.27	hal_lld_get_counter_value	87
6.7.3.28	hal_lld_get_counter_frequency	87
6.7.4	Typedef Documentation	87
6.7.4.1	halclock_t	87
6.7.4.2	halrtcnt_t	87
6.8	I2C Driver	87
6.8.1	Detailed Description	87
6.8.2	Driver State Machine	88
6.8.3	Function Documentation	91
6.8.3.1	i2cInit	91
6.8.3.2	i2cObjectInit	92
6.8.3.3	i2cStart	92
6.8.3.4	i2cStop	92
6.8.3.5	i2cGetErrors	93
6.8.3.6	i2cMasterTransmitTimeout	93
6.8.3.7	i2cMasterReceiveTimeout	94
6.8.3.8	i2cAcquireBus	95
6.8.3.9	i2cReleaseBus	95
6.8.3.10	CH_IRQ_HANDLER	95
6.8.3.11	CH_IRQ_HANDLER	96
6.8.3.12	CH_IRQ_HANDLER	96
6.8.3.13	CH_IRQ_HANDLER	96
6.8.3.14	CH_IRQ_HANDLER	96
6.8.3.15	CH_IRQ_HANDLER	96
6.8.3.16	i2c_lld_init	96
6.8.3.17	i2c_lld_start	97
6.8.3.18	i2c_lld_stop	97
6.8.3.19	i2c_lld_master_receive_timeout	97
6.8.3.20	i2c_lld_master_transmit_timeout	98
6.8.4	Variable Documentation	99
6.8.4.1	I2CD1	99
6.8.4.2	I2CD2	99

6.8.4.3	I2CD3 . . . . .	99
6.8.5	Define Documentation . . . . .	99
6.8.5.1	I2CD_NO_ERROR . . . . .	99
6.8.5.2	I2CD_BUS_ERROR . . . . .	99
6.8.5.3	I2CD_ARBITRATION_LOST . . . . .	99
6.8.5.4	I2CD_ACK_FAILURE . . . . .	99
6.8.5.5	I2CD_OVERRUN . . . . .	99
6.8.5.6	I2CD_PEC_ERROR . . . . .	99
6.8.5.7	I2CD_TIMEOUT . . . . .	99
6.8.5.8	I2CD_SMB_ALERT . . . . .	99
6.8.5.9	I2C_USE_MUTUAL_EXCLUSION . . . . .	100
6.8.5.10	i2cMasterTransmit . . . . .	100
6.8.5.11	i2cMasterReceive . . . . .	100
6.8.5.12	wakeup_isr . . . . .	100
6.8.5.13	I2C_CLK_FREQ . . . . .	100
6.8.5.14	STM32_I2C_USE_I2C1 . . . . .	101
6.8.5.15	STM32_I2C_USE_I2C2 . . . . .	101
6.8.5.16	STM32_I2C_USE_I2C3 . . . . .	101
6.8.5.17	STM32_I2C_I2C1_IRQ_PRIORITY . . . . .	101
6.8.5.18	STM32_I2C_I2C2_IRQ_PRIORITY . . . . .	101
6.8.5.19	STM32_I2C_I2C3_IRQ_PRIORITY . . . . .	101
6.8.5.20	STM32_I2C_I2C1_DMA_PRIORITY . . . . .	101
6.8.5.21	STM32_I2C_I2C2_DMA_PRIORITY . . . . .	102
6.8.5.22	STM32_I2C_I2C3_DMA_PRIORITY . . . . .	102
6.8.5.23	STM32_I2C_DMA_ERROR_HOOK . . . . .	102
6.8.5.24	STM32_I2C_I2C1_RX_DMA_STREAM . . . . .	102
6.8.5.25	STM32_I2C_I2C1_TX_DMA_STREAM . . . . .	102
6.8.5.26	STM32_I2C_I2C2_RX_DMA_STREAM . . . . .	102
6.8.5.27	STM32_I2C_I2C2_TX_DMA_STREAM . . . . .	103
6.8.5.28	STM32_I2C_I2C3_RX_DMA_STREAM . . . . .	103
6.8.5.29	STM32_I2C_I2C3_TX_DMA_STREAM . . . . .	103
6.8.5.30	STM32_DMA_REQUIRED . . . . .	103
6.8.5.31	i2c_lld_get_errors . . . . .	103
6.8.6	Typedef Documentation . . . . .	103
6.8.6.1	i2caddr_t . . . . .	103
6.8.6.2	i2cflags_t . . . . .	103
6.8.6.3	I2CDriver . . . . .	103
6.8.7	Enumeration Type Documentation . . . . .	104
6.8.7.1	i2cstate_t . . . . .	104
6.8.7.2	i2copmode_t . . . . .	104

6.8.7.3	i2cdutycycle_t . . . . .	104
6.9	ICU Driver . . . . .	104
6.9.1	Detailed Description . . . . .	104
6.9.2	Driver State Machine . . . . .	104
6.9.3	ICU Operations. . . . .	105
6.9.4	Function Documentation . . . . .	108
6.9.4.1	icuInit . . . . .	108
6.9.4.2	icuObjectInit . . . . .	108
6.9.4.3	icuStart . . . . .	108
6.9.4.4	icuStop . . . . .	109
6.9.4.5	icuEnable . . . . .	109
6.9.4.6	icuDisable . . . . .	110
6.9.4.7	icu_lld_init . . . . .	110
6.9.4.8	icu_lld_start . . . . .	111
6.9.4.9	icu_lld_stop . . . . .	111
6.9.4.10	icu_lld_enable . . . . .	111
6.9.4.11	icu_lld_disable . . . . .	111
6.9.5	Variable Documentation . . . . .	111
6.9.5.1	ICUD1 . . . . .	111
6.9.5.2	ICUD2 . . . . .	112
6.9.5.3	ICUD3 . . . . .	112
6.9.5.4	ICUD4 . . . . .	112
6.9.5.5	ICUD5 . . . . .	112
6.9.5.6	ICUD8 . . . . .	112
6.9.6	Define Documentation . . . . .	112
6.9.6.1	icuEnableI . . . . .	112
6.9.6.2	icuDisableI . . . . .	113
6.9.6.3	icuGetWidthI . . . . .	113
6.9.6.4	icuGetPeriodI . . . . .	113
6.9.6.5	_icu_isr_invoke_width_cb . . . . .	113
6.9.6.6	_icu_isr_invoke_period_cb . . . . .	114
6.9.6.7	STM32_ICU_USE_TIM1 . . . . .	114
6.9.6.8	STM32_ICU_USE_TIM2 . . . . .	114
6.9.6.9	STM32_ICU_USE_TIM3 . . . . .	114
6.9.6.10	STM32_ICU_USE_TIM4 . . . . .	115
6.9.6.11	STM32_ICU_USE_TIM5 . . . . .	115
6.9.6.12	STM32_ICU_USE_TIM8 . . . . .	115
6.9.6.13	STM32_ICU_TIM1_IRQ_PRIORITY . . . . .	115
6.9.6.14	STM32_ICU_TIM2_IRQ_PRIORITY . . . . .	115
6.9.6.15	STM32_ICU_TIM3_IRQ_PRIORITY . . . . .	115

6.9.6.16	STM32_ICU_TIM4_IRQ_PRIORITY . . . . .	115
6.9.6.17	STM32_ICU_TIM5_IRQ_PRIORITY . . . . .	115
6.9.6.18	STM32_ICU_TIM8_IRQ_PRIORITY . . . . .	115
6.9.6.19	icu_lld_get_width . . . . .	116
6.9.6.20	icu_lld_get_period . . . . .	116
6.9.7	Typedef Documentation . . . . .	116
6.9.7.1	ICUDriver . . . . .	116
6.9.7.2	icucallback_t . . . . .	116
6.9.7.3	icufreq_t . . . . .	116
6.9.7.4	icucnt_t . . . . .	116
6.9.8	Enumeration Type Documentation . . . . .	117
6.9.8.1	icustate_t . . . . .	117
6.9.8.2	icumode_t . . . . .	117
6.10	MMC over SPI Driver . . . . .	117
6.10.1	Detailed Description . . . . .	117
6.10.2	Driver State Machine . . . . .	117
6.10.3	Function Documentation . . . . .	119
6.10.3.1	mmcInit . . . . .	119
6.10.3.2	mmcObjectInit . . . . .	120
6.10.3.3	mmcStart . . . . .	120
6.10.3.4	mmcStop . . . . .	120
6.10.3.5	mmcConnect . . . . .	121
6.10.3.6	mmcDisconnect . . . . .	121
6.10.3.7	mmcStartSequentialRead . . . . .	122
6.10.3.8	mmcSequentialRead . . . . .	123
6.10.3.9	mmcStopSequentialRead . . . . .	123
6.10.3.10	mmcStartSequentialWrite . . . . .	124
6.10.3.11	mmcSequentialWrite . . . . .	125
6.10.3.12	mmcStopSequentialWrite . . . . .	125
6.10.4	Define Documentation . . . . .	126
6.10.4.1	MMC_SECTOR_SIZE . . . . .	126
6.10.4.2	MMC_NICE_WAITING . . . . .	126
6.10.4.3	MMC_POLLING_INTERVAL . . . . .	126
6.10.4.4	MMC_POLLING_DELAY . . . . .	126
6.10.4.5	mmcGetDriverState . . . . .	126
6.10.4.6	mmclsWriteProtected . . . . .	127
6.10.5	Typedef Documentation . . . . .	127
6.10.5.1	mmcquery_t . . . . .	127
6.10.6	Enumeration Type Documentation . . . . .	127
6.10.6.1	mmcstate_t . . . . .	127

6.11 PAL Driver . . . . .	127
6.11.1 Detailed Description . . . . .	127
6.11.2 Implementation Rules . . . . .	128
6.11.2.1 Writing on input pads . . . . .	128
6.11.2.2 Reading from output pads . . . . .	128
6.11.2.3 Writing unused or unimplemented port bits . . . . .	128
6.11.2.4 Reading from unused or unimplemented port bits . . . . .	128
6.11.2.5 Reading or writing on pins associated to other functionalities . . . . .	128
6.11.3 Function Documentation . . . . .	132
6.11.3.1 palReadBus . . . . .	132
6.11.3.2 palWriteBus . . . . .	132
6.11.3.3 palSetBusMode . . . . .	132
6.11.3.4 _pal_lld_init . . . . .	133
6.11.3.5 _pal_lld_setgroupmode . . . . .	133
6.11.4 Define Documentation . . . . .	133
6.11.4.1 PAL_MODE_RESET . . . . .	133
6.11.4.2 PAL_MODE_UNCONNECTED . . . . .	133
6.11.4.3 PAL_MODE_INPUT . . . . .	133
6.11.4.4 PAL_MODE_INPUT_PULLUP . . . . .	134
6.11.4.5 PAL_MODE_INPUT_PULLDOWN . . . . .	134
6.11.4.6 PAL_MODE_INPUT_ANALOG . . . . .	134
6.11.4.7 PAL_MODE_OUTPUT_PUSH_PULL . . . . .	134
6.11.4.8 PAL_MODE_OUTPUT_OPENDRAIN . . . . .	134
6.11.4.9 PAL_LOW . . . . .	134
6.11.4.10 PAL_HIGH . . . . .	134
6.11.4.11 PAL_PORT_BIT . . . . .	134
6.11.4.12 PAL_GROUP_MASK . . . . .	134
6.11.4.13 _IOBUS_DATA . . . . .	135
6.11.4.14 IOBUS_DECL . . . . .	135
6.11.4.15 pallinit . . . . .	135
6.11.4.16 palReadPort . . . . .	135
6.11.4.17 palReadLatch . . . . .	136
6.11.4.18 palWritePort . . . . .	136
6.11.4.19 palSetPort . . . . .	136
6.11.4.20 palClearPort . . . . .	137
6.11.4.21 palTogglePort . . . . .	137
6.11.4.22 palReadGroup . . . . .	137
6.11.4.23 palWriteGroup . . . . .	138
6.11.4.24 palSetGroupMode . . . . .	138
6.11.4.25 palReadPad . . . . .	138

6.11.4.26 palWritePad . . . . .	139
6.11.4.27 palSetPad . . . . .	139
6.11.4.28 palClearPad . . . . .	140
6.11.4.29 palTogglePad . . . . .	140
6.11.4.30 palSetPadMode . . . . .	140
6.11.4.31 PAL_MODE_STM32_ALTERNATE_PUSHPULL . . . . .	141
6.11.4.32 PAL_MODE_STM32_ALTERNATE_OPENDRAIN . . . . .	141
6.11.4.33 PAL_IOPORTS_WIDTH . . . . .	141
6.11.4.34 PAL_WHOLE_PORT . . . . .	141
6.11.4.35 IOPORT1 . . . . .	141
6.11.4.36 IOPORT2 . . . . .	141
6.11.4.37 IOPORT3 . . . . .	141
6.11.4.38 IOPORT4 . . . . .	141
6.11.4.39 IOPORT5 . . . . .	141
6.11.4.40 IOPORT6 . . . . .	141
6.11.4.41 IOPORT7 . . . . .	141
6.11.4.42 pal_lld_init . . . . .	142
6.11.4.43 pal_lld_readport . . . . .	142
6.11.4.44 pal_lld_readdlatch . . . . .	142
6.11.4.45 pal_lld_writeport . . . . .	142
6.11.4.46 pal_lld_setport . . . . .	143
6.11.4.47 pal_lld_clearport . . . . .	143
6.11.4.48 pal_lld_writegroup . . . . .	143
6.11.4.49 pal_lld_setgroupmode . . . . .	144
6.11.4.50 pal_lld_writepad . . . . .	144
6.11.5 Typedef Documentation . . . . .	144
6.11.5.1 ioportmask_t . . . . .	145
6.11.5.2 iomode_t . . . . .	145
6.11.5.3 ioportid_t . . . . .	145
6.12 PWM Driver . . . . .	145
6.12.1 Detailed Description . . . . .	145
6.12.2 Driver State Machine . . . . .	145
6.12.3 PWM Operations . . . . .	145
6.12.4 Function Documentation . . . . .	149
6.12.4.1 pwmlInit . . . . .	149
6.12.4.2 pwmObjectInit . . . . .	149
6.12.4.3 pwmStart . . . . .	150
6.12.4.4 pwmStop . . . . .	150
6.12.4.5 pwmChangePeriod . . . . .	150
6.12.4.6 pwmEnableChannel . . . . .	151

6.12.4.7	pwmDisableChannel . . . . .	152
6.12.4.8	pwm_lld_init . . . . .	152
6.12.4.9	pwm_lld_start . . . . .	153
6.12.4.10	pwm_lld_stop . . . . .	153
6.12.4.11	pwm_lld_enable_channel . . . . .	153
6.12.4.12	pwm_lld_disable_channel . . . . .	153
6.12.5	Variable Documentation . . . . .	154
6.12.5.1	PWMD1 . . . . .	154
6.12.5.2	PWMD2 . . . . .	154
6.12.5.3	PWMD3 . . . . .	154
6.12.5.4	PWMD4 . . . . .	154
6.12.5.5	PWMD5 . . . . .	155
6.12.5.6	PWMD8 . . . . .	155
6.12.6	Define Documentation . . . . .	155
6.12.6.1	PWM_OUTPUT_MASK . . . . .	155
6.12.6.2	PWM_OUTPUT_DISABLED . . . . .	155
6.12.6.3	PWM_OUTPUT_ACTIVE_HIGH . . . . .	155
6.12.6.4	PWM_OUTPUT_ACTIVE_LOW . . . . .	155
6.12.6.5	PWM_FRACTION_TO_WIDTH . . . . .	155
6.12.6.6	PWM_DEGREES_TO_WIDTH . . . . .	156
6.12.6.7	PWM_PERCENTAGE_TO_WIDTH . . . . .	156
6.12.6.8	pwmChangePeriodl . . . . .	156
6.12.6.9	pwmEnableChannell . . . . .	157
6.12.6.10	pwmDisableChannell . . . . .	157
6.12.6.11	PWM_CHANNELS . . . . .	158
6.12.6.12	PWM_COMPLEMENTARY_OUTPUT_MASK . . . . .	158
6.12.6.13	PWM_COMPLEMENTARY_OUTPUT_DISABLED . . . . .	158
6.12.6.14	PWM_COMPLEMENTARY_OUTPUT_ACTIVE_HIGH . . . . .	158
6.12.6.15	PWM_COMPLEMENTARY_OUTPUT_ACTIVE_LOW . . . . .	159
6.12.6.16	STM32_PWM_USE_ADVANCED . . . . .	159
6.12.6.17	STM32_PWM_USE_TIM1 . . . . .	159
6.12.6.18	STM32_PWM_USE_TIM2 . . . . .	159
6.12.6.19	STM32_PWM_USE_TIM3 . . . . .	159
6.12.6.20	STM32_PWM_USE_TIM4 . . . . .	159
6.12.6.21	STM32_PWM_USE_TIM5 . . . . .	160
6.12.6.22	STM32_PWM_USE_TIM8 . . . . .	160
6.12.6.23	STM32_PWM_TIM1_IRQ_PRIORITY . . . . .	160
6.12.6.24	STM32_PWM_TIM2_IRQ_PRIORITY . . . . .	160
6.12.6.25	STM32_PWM_TIM3_IRQ_PRIORITY . . . . .	160
6.12.6.26	STM32_PWM_TIM4_IRQ_PRIORITY . . . . .	160

6.12.6.27 STM32_PWM_TIM5_IRQ_PRIORITY . . . . .	160
6.12.6.28 STM32_PWM_TIM8_IRQ_PRIORITY . . . . .	160
6.12.6.29 pwm_lld_change_period . . . . .	160
6.12.7 Typedef Documentation . . . . .	161
6.12.7.1 PWMDriver . . . . .	161
6.12.7.2 pwmcallback_t . . . . .	161
6.12.7.3 pwmmode_t . . . . .	161
6.12.7.4 pwmchannel_t . . . . .	161
6.12.7.5 pwmcnt_t . . . . .	161
6.12.8 Enumeration Type Documentation . . . . .	161
6.12.8.1 pwmstate_t . . . . .	161
6.13 RTC Driver . . . . .	162
6.13.1 Detailed Description . . . . .	162
6.13.2 Function Documentation . . . . .	164
6.13.2.1 rtcInit . . . . .	164
6.13.2.2 rtcSetTime . . . . .	164
6.13.2.3 rtcGetTime . . . . .	164
6.13.2.4 rtcSetAlarm . . . . .	165
6.13.2.5 rtcGetAlarm . . . . .	165
6.13.2.6 rtcSetCallback . . . . .	165
6.13.2.7 CH_IRQ_HANDLER . . . . .	165
6.13.2.8 rtc_lld_init . . . . .	165
6.13.2.9 rtc_lld_set_time . . . . .	166
6.13.2.10 rtc_lld_get_time . . . . .	166
6.13.2.11 rtc_lld_set_alarm . . . . .	166
6.13.2.12 rtc_lld_get_alarm . . . . .	166
6.13.2.13 rtc_lld_set_callback . . . . .	167
6.13.3 Variable Documentation . . . . .	167
6.13.3.1 RTCD1 . . . . .	167
6.13.4 Define Documentation . . . . .	167
6.13.4.1 rtcSetTimel . . . . .	167
6.13.4.2 rtcGetTimel . . . . .	167
6.13.4.3 rtcSetAlarml . . . . .	168
6.13.4.4 rtcGetAlarml . . . . .	168
6.13.4.5 rtcSetCallbackl . . . . .	168
6.13.4.6 rtc_lld_apb1_sync . . . . .	168
6.13.4.7 rtc_lld_wait_write . . . . .	169
6.13.4.8 rtc_lld_acquire . . . . .	169
6.13.4.9 rtc_lld_release . . . . .	169
6.13.4.10 RTC_SUPPORTS_CALLBACKS . . . . .	169

6.13.4.11 RTC_ALARMS . . . . .	169
6.13.5 Typedef Documentation . . . . .	169
6.13.5.1 RTCDriver . . . . .	169
6.13.5.2 RTCTime . . . . .	169
6.13.5.3 RTCAlarm . . . . .	170
6.13.5.4 RTCCallbackConfig . . . . .	170
6.13.5.5 rtcalarm_t . . . . .	170
6.13.5.6 rtccb_t . . . . .	170
6.13.6 Enumeration Type Documentation . . . . .	170
6.13.6.1 rtcevent_t . . . . .	170
6.14 SDC Driver . . . . .	170
6.14.1 Detailed Description . . . . .	170
6.14.2 Driver State Machine . . . . .	170
6.14.3 SDC Operations. . . . .	171
6.14.4 Function Documentation . . . . .	174
6.14.4.1 sdcInit . . . . .	174
6.14.4.2 sdcObjectInit . . . . .	174
6.14.4.3 sdcStart . . . . .	174
6.14.4.4 sdcStop . . . . .	175
6.14.4.5 sdcConnect . . . . .	175
6.14.4.6 sdcDisconnect . . . . .	176
6.14.4.7 sdcRead . . . . .	177
6.14.4.8 sdcWrite . . . . .	177
6.14.4.9 _sdc_wait_for_transfer_state . . . . .	178
6.14.4.10 CH_IRQ_HANDLER . . . . .	179
6.14.4.11 sdc_lld_init . . . . .	179
6.14.4.12 sdc_lld_start . . . . .	179
6.14.4.13 sdc_lld_stop . . . . .	180
6.14.4.14 sdc_lld_start_clk . . . . .	180
6.14.4.15 sdc_lld_set_data_clk . . . . .	180
6.14.4.16 sdc_lld_stop_clk . . . . .	181
6.14.4.17 sdc_lld_set_bus_mode . . . . .	181
6.14.4.18 sdc_lld_send_cmd_none . . . . .	181
6.14.4.19 sdc_lld_send_cmd_short . . . . .	181
6.14.4.20 sdc_lld_send_cmd_short_crc . . . . .	182
6.14.4.21 sdc_lld_send_cmd_long_crc . . . . .	182
6.14.4.22 sdc_lld_read . . . . .	183
6.14.4.23 sdc_lld_write . . . . .	183
6.14.5 Variable Documentation . . . . .	183
6.14.5.1 SDCD1 . . . . .	183

6.14.6 Define Documentation . . . . .	183
6.14.6.1 SDC_BLOCK_SIZE . . . . .	183
6.14.6.2 SDC_CMD8_PATTERN . . . . .	184
6.14.6.3 SDC_MODE_CARDTYPE_MASK . . . . .	184
6.14.6.4 SDC_MODE_CARDTYPE_SDV11 . . . . .	184
6.14.6.5 SDC_MODE_CARDTYPE_SDV20 . . . . .	184
6.14.6.6 SDC_MODE_CARDTYPE_MMC . . . . .	184
6.14.6.7 SDC_MODE_HIGH_CAPACITY . . . . .	184
6.14.6.8 SDC_R1_ERROR_MASK . . . . .	184
6.14.6.9 SDC_INIT_RETRY . . . . .	184
6.14.6.10 SDC_MMC_SUPPORT . . . . .	184
6.14.6.11 SDC_NICE_WAITING . . . . .	184
6.14.6.12 SDC_R1_ERROR . . . . .	185
6.14.6.13 SDC_R1_STS . . . . .	185
6.14.6.14 SDC_R1_IS_CARD_LOCKED . . . . .	185
6.14.6.15 sdcGetDriverState . . . . .	185
6.14.6.16 sdclsCardInserted . . . . .	185
6.14.6.17 sdclsWriteProtected . . . . .	186
6.14.6.18 STM32_SDC_DATATIMEOUT . . . . .	186
6.14.6.19 STM32_SDC_SDIO_DMA_PRIORITY . . . . .	186
6.14.6.20 STM32_SDC_SDIO_IRQ_PRIORITY . . . . .	186
6.14.6.21 STM32_SDC_UNALIGNED_SUPPORT . . . . .	186
6.14.7 Typedef Documentation . . . . .	186
6.14.7.1 sdcmode_t . . . . .	186
6.14.7.2 SDCDriver . . . . .	187
6.14.8 Enumeration Type Documentation . . . . .	187
6.14.8.1 sdclstate_t . . . . .	187
6.14.8.2 sdcbusmode_t . . . . .	187
6.15 Serial Driver . . . . .	187
6.15.1 Detailed Description . . . . .	187
6.15.2 Driver State Machine . . . . .	187
6.15.3 Function Documentation . . . . .	191
6.15.3.1 sdInit . . . . .	191
6.15.3.2 sdObjectInit . . . . .	192
6.15.3.3 sdStart . . . . .	192
6.15.3.4 sdStop . . . . .	192
6.15.3.5 sdIncomingData . . . . .	193
6.15.3.6 sdRequestData . . . . .	193
6.15.3.7 CH_IRQ_HANDLER . . . . .	194
6.15.3.8 CH_IRQ_HANDLER . . . . .	194

6.15.3.9 CH_IRQ_HANDLER . . . . .	194
6.15.3.10 CH_IRQ_HANDLER . . . . .	194
6.15.3.11 CH_IRQ_HANDLER . . . . .	194
6.15.3.12 CH_IRQ_HANDLER . . . . .	194
6.15.3.13 sd_lld_init . . . . .	195
6.15.3.14 sd_lld_start . . . . .	195
6.15.3.15 sd_lld_stop . . . . .	195
6.15.4 Variable Documentation . . . . .	195
6.15.4.1 SD1 . . . . .	195
6.15.4.2 SD2 . . . . .	196
6.15.4.3 SD3 . . . . .	196
6.15.4.4 SD4 . . . . .	196
6.15.4.5 SD5 . . . . .	196
6.15.4.6 SD6 . . . . .	196
6.15.5 Define Documentation . . . . .	196
6.15.5.1 SD_PARITY_ERROR . . . . .	196
6.15.5.2 SD_FRAMING_ERROR . . . . .	196
6.15.5.3 SD_OVERRUN_ERROR . . . . .	196
6.15.5.4 SD_NOISE_ERROR . . . . .	196
6.15.5.5 SD_BREAK_DETECTED . . . . .	196
6.15.5.6 SERIAL_DEFAULT_BITRATE . . . . .	196
6.15.5.7 SERIAL_BUFFERS_SIZE . . . . .	197
6.15.5.8 _serial_driver_methods . . . . .	197
6.15.5.9 sdPutWouldBlock . . . . .	197
6.15.5.10 sdGetWouldBlock . . . . .	197
6.15.5.11 sdPut . . . . .	198
6.15.5.12 sdPutTimeout . . . . .	198
6.15.5.13 sdGet . . . . .	198
6.15.5.14 sdGetTimeout . . . . .	198
6.15.5.15 sdWrite . . . . .	199
6.15.5.16 sdWriteTimeout . . . . .	199
6.15.5.17 sdAsynchronousWrite . . . . .	199
6.15.5.18 sdRead . . . . .	200
6.15.5.19 sdReadTimeout . . . . .	200
6.15.5.20 sdAsynchronousRead . . . . .	200
6.15.5.21 STM32_SERIAL_USE_USART1 . . . . .	200
6.15.5.22 STM32_SERIAL_USE_USART2 . . . . .	201
6.15.5.23 STM32_SERIAL_USE_USART3 . . . . .	201
6.15.5.24 STM32_SERIAL_USE_UART4 . . . . .	201
6.15.5.25 STM32_SERIAL_USE_UART5 . . . . .	201

6.15.5.26 STM32_SERIAL_USE_USART6 . . . . .	201
6.15.5.27 STM32_SERIAL_USART1_PRIORITY . . . . .	201
6.15.5.28 STM32_SERIAL_USART2_PRIORITY . . . . .	201
6.15.5.29 STM32_SERIAL_USART3_PRIORITY . . . . .	202
6.15.5.30 STM32_SERIAL_UART4_PRIORITY . . . . .	202
6.15.5.31 STM32_SERIAL_UART5_PRIORITY . . . . .	202
6.15.5.32 STM32_SERIAL_USART6_PRIORITY . . . . .	202
6.15.5.33 _serial_driver_data . . . . .	202
6.15.5.34 USART_CR2_STOP1_BITS . . . . .	202
6.15.5.35 USART_CR2_STOP0P5_BITS . . . . .	202
6.15.5.36 USART_CR2_STOP2_BITS . . . . .	202
6.15.5.37 USART_CR2_STOP1P5_BITS . . . . .	202
6.15.6 Typedef Documentation . . . . .	203
6.15.6.1 SerialDriver . . . . .	203
6.15.7 Enumeration Type Documentation . . . . .	203
6.15.7.1 sdstate_t . . . . .	203
6.16 Serial over USB Driver . . . . .	203
6.16.1 Detailed Description . . . . .	203
6.16.2 Driver State Machine . . . . .	203
6.16.3 Function Documentation . . . . .	205
6.16.3.1 sduInit . . . . .	205
6.16.3.2 sduObjectInit . . . . .	205
6.16.3.3 sduStart . . . . .	205
6.16.3.4 sduStop . . . . .	205
6.16.3.5 sduRequestsHook . . . . .	206
6.16.3.6 sduDataTransmitted . . . . .	206
6.16.3.7 sduDataReceived . . . . .	207
6.16.3.8 sduInterruptTransmitted . . . . .	207
6.16.4 Define Documentation . . . . .	207
6.16.4.1 SERIAL_USB_BUFFERS_SIZE . . . . .	207
6.16.4.2 _serial_usb_driver_data . . . . .	208
6.16.4.3 _serial_usb_driver_methods . . . . .	208
6.16.5 Typedef Documentation . . . . .	208
6.16.5.1 SerialUSBDriver . . . . .	208
6.16.6 Enumeration Type Documentation . . . . .	208
6.16.6.1 sdustate_t . . . . .	208
6.17 SPI Driver . . . . .	208
6.17.1 Detailed Description . . . . .	208
6.17.2 Driver State Machine . . . . .	209
6.17.3 Function Documentation . . . . .	212

6.17.3.1	spilinit . . . . .	212
6.17.3.2	spiObjectInit . . . . .	213
6.17.3.3	spiStart . . . . .	213
6.17.3.4	spiStop . . . . .	213
6.17.3.5	spiSelect . . . . .	214
6.17.3.6	spiUnselect . . . . .	214
6.17.3.7	spiStartIgnore . . . . .	214
6.17.3.8	spiStartExchange . . . . .	215
6.17.3.9	spiStartSend . . . . .	215
6.17.3.10	spiStartReceive . . . . .	216
6.17.3.11	spilgnore . . . . .	216
6.17.3.12	spiExchange . . . . .	216
6.17.3.13	spiSend . . . . .	217
6.17.3.14	spiReceive . . . . .	217
6.17.3.15	spiAcquireBus . . . . .	218
6.17.3.16	spiReleaseBus . . . . .	218
6.17.3.17	spi_lld_init . . . . .	218
6.17.3.18	spi_lld_start . . . . .	219
6.17.3.19	spi_lld_stop . . . . .	219
6.17.3.20	spi_lld_select . . . . .	219
6.17.3.21	spi_lld_unselect . . . . .	220
6.17.3.22	spi_lld_ignore . . . . .	220
6.17.3.23	spi_lld_exchange . . . . .	220
6.17.3.24	spi_lld_send . . . . .	221
6.17.3.25	spi_lld_receive . . . . .	221
6.17.3.26	spi_lld_polled_exchange . . . . .	221
6.17.4	Variable Documentation . . . . .	222
6.17.4.1	SPID1 . . . . .	222
6.17.4.2	SPID2 . . . . .	222
6.17.4.3	SPID3 . . . . .	222
6.17.5	Define Documentation . . . . .	222
6.17.5.1	SPI_USE_WAIT . . . . .	222
6.17.5.2	SPI_USE_MUTUAL_EXCLUSION . . . . .	222
6.17.5.3	spiSelectl . . . . .	222
6.17.5.4	spiUnselectl . . . . .	223
6.17.5.5	spiStartIgnorel . . . . .	223
6.17.5.6	spiStartExchangel . . . . .	223
6.17.5.7	spiStartSendl . . . . .	224
6.17.5.8	spiStartReceivel . . . . .	225
6.17.5.9	spiPolledExchange . . . . .	225

6.17.5.10 _spi_wait_s . . . . .	226
6.17.5.11 _spi_wakeup_isr . . . . .	226
6.17.5.12 _spi_isr_code . . . . .	226
6.17.5.13 STM32_SPI_USE_SPI1 . . . . .	227
6.17.5.14 STM32_SPI_USE_SPI2 . . . . .	227
6.17.5.15 STM32_SPI_USE_SPI3 . . . . .	227
6.17.5.16 STM32_SPI_SPI1_IRQ_PRIORITY . . . . .	228
6.17.5.17 STM32_SPI_SPI2_IRQ_PRIORITY . . . . .	228
6.17.5.18 STM32_SPI_SPI3_IRQ_PRIORITY . . . . .	228
6.17.5.19 STM32_SPI_SPI1_DMA_PRIORITY . . . . .	228
6.17.5.20 STM32_SPI_SPI2_DMA_PRIORITY . . . . .	228
6.17.5.21 STM32_SPI_SPI3_DMA_PRIORITY . . . . .	228
6.17.5.22 STM32_SPI_DMA_ERROR_HOOK . . . . .	228
6.17.5.23 STM32_SPI_SPI1_RX_DMA_STREAM . . . . .	228
6.17.5.24 STM32_SPI_SPI1_TX_DMA_STREAM . . . . .	229
6.17.5.25 STM32_SPI_SPI2_RX_DMA_STREAM . . . . .	229
6.17.5.26 STM32_SPI_SPI2_TX_DMA_STREAM . . . . .	229
6.17.5.27 STM32_SPI_SPI3_RX_DMA_STREAM . . . . .	229
6.17.5.28 STM32_SPI_SPI3_TX_DMA_STREAM . . . . .	229
6.17.6 Typedef Documentation . . . . .	229
6.17.6.1 SPIDriver . . . . .	229
6.17.6.2 spicallback_t . . . . .	229
6.17.7 Enumeration Type Documentation . . . . .	230
6.17.7.1 spistate_t . . . . .	230
6.18 Time Measurement Driver. . . . .	230
6.18.1 Detailed Description . . . . .	230
6.18.2 Function Documentation . . . . .	231
6.18.2.1 tmInit . . . . .	231
6.18.2.2 tmObjectInit . . . . .	231
6.18.3 Define Documentation . . . . .	231
6.18.3.1 tmStartMeasurement . . . . .	231
6.18.3.2 tmStopMeasurement . . . . .	232
6.18.4 Typedef Documentation . . . . .	232
6.18.4.1 TimeMeasurement . . . . .	232
6.19 UART Driver . . . . .	232
6.19.1 Detailed Description . . . . .	232
6.19.2 Driver State Machine . . . . .	233
6.19.2.1 Transmitter sub State Machine . . . . .	233
6.19.2.2 Receiver sub State Machine . . . . .	234
6.19.3 Function Documentation . . . . .	237

6.19.3.1	uartInit	237
6.19.3.2	uartObjectInit	237
6.19.3.3	uartStart	237
6.19.3.4	uartStop	238
6.19.3.5	uartStartSend	238
6.19.3.6	uartStartSendl	239
6.19.3.7	uartStopSend	239
6.19.3.8	uartStopSendl	240
6.19.3.9	uartStartReceive	241
6.19.3.10	uartStartReceive1	241
6.19.3.11	uartStopReceive	242
6.19.3.12	uartStopReceive1	242
6.19.3.13	CH_IRQ_HANDLER	243
6.19.3.14	CH_IRQ_HANDLER	243
6.19.3.15	CH_IRQ_HANDLER	243
6.19.3.16	uart_lld_init	244
6.19.3.17	uart_lld_start	244
6.19.3.18	uart_lld_stop	244
6.19.3.19	uart_lld_start_send	245
6.19.3.20	uart_lld_stop_send	245
6.19.3.21	uart_lld_start_receive	245
6.19.3.22	uart_lld_stop_receive	246
6.19.4	Variable Documentation	246
6.19.4.1	UARTD1	246
6.19.4.2	UARTD2	246
6.19.4.3	UARTD3	246
6.19.5	Define Documentation	246
6.19.5.1	UART_NO_ERROR	246
6.19.5.2	UART_PARITY_ERROR	246
6.19.5.3	UART_FRAMING_ERROR	247
6.19.5.4	UART_OVERRUN_ERROR	247
6.19.5.5	UART_NOISE_ERROR	247
6.19.5.6	UART_BREAK_DETECTED	247
6.19.5.7	STM32_UART_USE_USART1	247
6.19.5.8	STM32_UART_USE_USART2	247
6.19.5.9	STM32_UART_USE_USART3	247
6.19.5.10	STM32_UART_USART1_IRQ_PRIORITY	247
6.19.5.11	STM32_UART_USART2_IRQ_PRIORITY	247
6.19.5.12	STM32_UART_USART3_IRQ_PRIORITY	248
6.19.5.13	STM32_UART_USART1_DMA_PRIORITY	248

6.19.5.14 STM32_UART_USART2_DMA_PRIORITY . . . . .	248
6.19.5.15 STM32_UART_USART3_DMA_PRIORITY . . . . .	248
6.19.5.16 STM32_UART_DMA_ERROR_HOOK . . . . .	248
6.19.5.17 STM32_UART_USART1_RX_DMA_STREAM . . . . .	248
6.19.5.18 STM32_UART_USART1_TX_DMA_STREAM . . . . .	248
6.19.5.19 STM32_UART_USART2_RX_DMA_STREAM . . . . .	249
6.19.5.20 STM32_UART_USART2_TX_DMA_STREAM . . . . .	249
6.19.5.21 STM32_UART_USART3_RX_DMA_STREAM . . . . .	249
6.19.5.22 STM32_UART_USART3_TX_DMA_STREAM . . . . .	249
6.19.6 Typedef Documentation . . . . .	249
6.19.6.1 uartflags_t . . . . .	249
6.19.6.2 UARTDriver . . . . .	249
6.19.6.3 uartcb_t . . . . .	249
6.19.6.4 uartccb_t . . . . .	250
6.19.6.5 uarteccb_t . . . . .	250
6.19.7 Enumeration Type Documentation . . . . .	250
6.19.7.1 uartstate_t . . . . .	250
6.19.7.2 uarttxstate_t . . . . .	250
6.19.7.3 uartrxstate_t . . . . .	250
6.20 USB Driver . . . . .	251
6.20.1 Detailed Description . . . . .	251
6.20.2 Driver State Machine . . . . .	251
6.20.3 USB Operations . . . . .	251
6.20.3.1 USB Implementation . . . . .	251
6.20.3.2 USB Endpoints . . . . .	252
6.20.3.3 USB Packet Buffers . . . . .	253
6.20.3.4 USB Callbacks . . . . .	253
6.20.4 Function Documentation . . . . .	259
6.20.4.1 usblInit . . . . .	259
6.20.4.2 usbObjectInit . . . . .	259
6.20.4.3 usbStart . . . . .	259
6.20.4.4 usbStop . . . . .	260
6.20.4.5 usblInitEndpointl . . . . .	260
6.20.4.6 usbDisableEndpointsl . . . . .	261
6.20.4.7 usbStartReceive1 . . . . .	261
6.20.4.8 usbStartTransmitl . . . . .	262
6.20.4.9 usbStallReceive1 . . . . .	263
6.20.4.10 usbStallTransmitl . . . . .	263
6.20.4.11 _usb_reset . . . . .	264
6.20.4.12 _usb_ep0setup . . . . .	264

6.20.4.13 _usb_ep0in . . . . .	265
6.20.4.14 _usb_ep0out . . . . .	266
6.20.4.15 CH_IRQ_HANDLER . . . . .	267
6.20.4.16 CH_IRQ_HANDLER . . . . .	267
6.20.4.17 usb_lld_init . . . . .	268
6.20.4.18 usb_lld_start . . . . .	268
6.20.4.19 usb_lld_stop . . . . .	269
6.20.4.20 usb_lld_reset . . . . .	269
6.20.4.21 usb_lld_set_address . . . . .	269
6.20.4.22 usb_lld_init_endpoint . . . . .	270
6.20.4.23 usb_lld_disable_endpoints . . . . .	270
6.20.4.24 usb_lld_get_status_out . . . . .	270
6.20.4.25 usb_lld_get_status_in . . . . .	270
6.20.4.26 usb_lld_read_setup . . . . .	271
6.20.4.27 usb_lld_read_packet_buffer . . . . .	271
6.20.4.28 usb_lld_write_packet_buffer . . . . .	272
6.20.4.29 usb_lld_prepare_receive . . . . .	272
6.20.4.30 usb_lld_prepare_transmit . . . . .	272
6.20.4.31 usb_lld_start_out . . . . .	273
6.20.4.32 usb_lld_start_in . . . . .	273
6.20.4.33 usb_lld_stall_out . . . . .	273
6.20.4.34 usb_lld_stall_in . . . . .	274
6.20.4.35 usb_lld_clear_out . . . . .	274
6.20.4.36 usb_lld_clear_in . . . . .	274
6.20.5 Variable Documentation . . . . .	274
6.20.5.1 USBD1 . . . . .	274
6.20.5.2 in . . . . .	274
6.20.5.3 out . . . . .	274
6.20.6 Define Documentation . . . . .	274
6.20.6.1 USB_DESC_INDEX . . . . .	275
6.20.6.2 USB_DESC_BYTE . . . . .	275
6.20.6.3 USB_DESC_WORD . . . . .	275
6.20.6.4 USB_DESC_BCD . . . . .	275
6.20.6.5 USB_DESC_DEVICE . . . . .	275
6.20.6.6 USB_DESC_CONFIGURATION . . . . .	275
6.20.6.7 USB_DESC_INTERFACE . . . . .	276
6.20.6.8 USB_DESC_ENDPOINT . . . . .	276
6.20.6.9 USB_EP_MODE_TYPE . . . . .	276
6.20.6.10 USB_EP_MODE_TYPE_CTRL . . . . .	276
6.20.6.11 USB_EP_MODE_TYPE_ISOC . . . . .	276

6.20.6.12 USB_EP_MODE_TYPE_BULK . . . . .	276
6.20.6.13 USB_EP_MODE_TYPE_INTR . . . . .	276
6.20.6.14 USB_EP_MODE_TRANSACTION . . . . .	276
6.20.6.15 USB_EP_MODE_PACKET . . . . .	276
6.20.6.16 usbConnectBus . . . . .	277
6.20.6.17 usbDisconnectBus . . . . .	277
6.20.6.18 usbGetFrameNumber . . . . .	277
6.20.6.19 usbGetTransmitStatus() . . . . .	277
6.20.6.20 usbGetReceiveStatus() . . . . .	277
6.20.6.21 usbReadPacketBuffer . . . . .	278
6.20.6.22 usbWritePacketBuffer . . . . .	278
6.20.6.23 usbPrepareReceive . . . . .	279
6.20.6.24 usbPrepareTransmit . . . . .	279
6.20.6.25 usbGetReceiveTransactionSize() . . . . .	279
6.20.6.26 usbGetReceivePacketSize() . . . . .	280
6.20.6.27 usbSetupTransfer . . . . .	280
6.20.6.28 usbReadSetup . . . . .	281
6.20.6.29 _usb_isr_invoke_event_cb . . . . .	281
6.20.6.30 _usb_isr_invoke_sof_cb . . . . .	281
6.20.6.31 _usb_isr_invoke_setup_cb . . . . .	282
6.20.6.32 _usb_isr_invoke_in_cb . . . . .	282
6.20.6.33 _usb_isr_invoke_out_cb . . . . .	282
6.20.6.34 USB_ENDPOINTS_NUMBER . . . . .	283
6.20.6.35 STM32_USB_BASE . . . . .	283
6.20.6.36 STM32_USBRAM_BASE . . . . .	283
6.20.6.37 STM32_USB . . . . .	283
6.20.6.38 STM32_USBRAM . . . . .	283
6.20.6.39 USB_PMA_SIZE . . . . .	283
6.20.6.40 EPR_TOGGLE_MASK . . . . .	283
6.20.6.41 USB_GET_DESCRIPTOR . . . . .	283
6.20.6.42 USB_ADDR2PTR . . . . .	283
6.20.6.43 USB_MAX_ENDPOINTS . . . . .	283
6.20.6.44 USB_SET_ADDRESS_MODE . . . . .	284
6.20.6.45 STM32_USB_USE_USB1 . . . . .	284
6.20.6.46 STM32_USB_LOW_POWER_ON_SUSPEND . . . . .	284
6.20.6.47 STM32_USB_USB1_HP_IRQ_PRIORITY . . . . .	284
6.20.6.48 STM32_USB_USB1_LP_IRQ_PRIORITY . . . . .	284
6.20.6.49 usb_ll_fetch_word . . . . .	284
6.20.6.50 usb_ll_get_frame_number . . . . .	284
6.20.6.51 usb_ll_get_transaction_size . . . . .	285

6.20.6.52 <code>usb_lld_get_packet_size</code>	285
6.20.7 <a href="#">Typedef Documentation</a>	285
6.20.7.1 <code>USBDriver</code>	285
6.20.7.2 <code>usbep_t</code>	285
6.20.7.3 <code>usbcallback_t</code>	285
6.20.7.4 <code>usbepcallback_t</code>	286
6.20.7.5 <code>usbeventcb_t</code>	286
6.20.7.6 <code>usbreqhandler_t</code>	286
6.20.7.7 <code>usbgetdescriptor_t</code>	286
6.20.8 <a href="#">Enumeration Type Documentation</a>	286
6.20.8.1 <code>usbstate_t</code>	286
6.20.8.2 <code>usbepstatus_t</code>	287
6.20.8.3 <code>usbep0state_t</code>	287
6.20.8.4 <code>usbevent_t</code>	287
6.21 STM32F100 HAL Support	287
6.21.1 <a href="#">Detailed Description</a>	287
6.21.2 <a href="#">Define Documentation</a>	295
6.21.2.1 <code>STM32_SYSCLK_MAX</code>	295
6.21.2.2 <code>STM32_HSECLK_MAX</code>	295
6.21.2.3 <code>STM32_HSECLK_MIN</code>	295
6.21.2.4 <code>STM32_LSECLK_MAX</code>	295
6.21.2.5 <code>STM32_LSECLK_MIN</code>	295
6.21.2.6 <code>STM32_PLLIN_MAX</code>	295
6.21.2.7 <code>STM32_PLLIN_MIN</code>	295
6.21.2.8 <code>STM32_PLLOUT_MAX</code>	295
6.21.2.9 <code>STM32_PLLOUT_MIN</code>	295
6.21.2.10 <code>STM32_PCLK1_MAX</code>	295
6.21.2.11 <code>STM32_PCLK2_MAX</code>	295
6.21.2.12 <code>STM32_ADCCLK_MAX</code>	295
6.21.2.13 <code>STM32_SW_HSI</code>	296
6.21.2.14 <code>STM32_SW_HSE</code>	296
6.21.2.15 <code>STM32_SW_PLL</code>	296
6.21.2.16 <code>STM32_HPRE_DIV1</code>	296
6.21.2.17 <code>STM32_HPRE_DIV2</code>	296
6.21.2.18 <code>STM32_HPRE_DIV4</code>	296
6.21.2.19 <code>STM32_HPRE_DIV8</code>	296
6.21.2.20 <code>STM32_HPRE_DIV16</code>	296
6.21.2.21 <code>STM32_HPRE_DIV64</code>	296
6.21.2.22 <code>STM32_HPRE_DIV128</code>	296
6.21.2.23 <code>STM32_HPRE_DIV256</code>	296

6.21.2.24 STM32_HPRE_DIV512 . . . . .	296
6.21.2.25 STM32_PPREG1_DIV1 . . . . .	297
6.21.2.26 STM32_PPREG1_DIV2 . . . . .	297
6.21.2.27 STM32_PPREG1_DIV4 . . . . .	297
6.21.2.28 STM32_PPREG1_DIV8 . . . . .	297
6.21.2.29 STM32_PPREG1_DIV16 . . . . .	297
6.21.2.30 STM32_PPREG2_DIV1 . . . . .	297
6.21.2.31 STM32_PPREG2_DIV2 . . . . .	297
6.21.2.32 STM32_PPREG2_DIV4 . . . . .	297
6.21.2.33 STM32_PPREG2_DIV8 . . . . .	297
6.21.2.34 STM32_PPREG2_DIV16 . . . . .	297
6.21.2.35 STM32_ADCPRE_DIV2 . . . . .	297
6.21.2.36 STM32_ADCPRE_DIV4 . . . . .	297
6.21.2.37 STM32_ADCPRE_DIV6 . . . . .	298
6.21.2.38 STM32_ADCPRE_DIV8 . . . . .	298
6.21.2.39 STM32_PLLSRC_HSI . . . . .	298
6.21.2.40 STM32_PLLSRC_HSE . . . . .	298
6.21.2.41 STM32_PLLXTPRE_DIV1 . . . . .	298
6.21.2.42 STM32_PLLXTPRE_DIV2 . . . . .	298
6.21.2.43 STM32_MCOSEL_NOCLOCK . . . . .	298
6.21.2.44 STM32_MCOSEL_SYSCLK . . . . .	298
6.21.2.45 STM32_MCOSEL_HSI . . . . .	298
6.21.2.46 STM32_MCOSEL_HSE . . . . .	298
6.21.2.47 STM32_MCOSEL_PLLDIV2 . . . . .	298
6.21.2.48 STM32_RTCSEL_MASK . . . . .	298
6.21.2.49 STM32_RTCSEL_NOCLOCK . . . . .	299
6.21.2.50 STM32_RTCSEL_LSE . . . . .	299
6.21.2.51 STM32_RTCSEL_LSI . . . . .	299
6.21.2.52 STM32_RTCSEL_HSEDIV . . . . .	299
6.21.2.53 WWDG_IRQHandler . . . . .	299
6.21.2.54 PVD_IRQHandler . . . . .	299
6.21.2.55 TAMPER_IRQHandler . . . . .	299
6.21.2.56 RTC_IRQHandler . . . . .	299
6.21.2.57 FLASH_IRQHandler . . . . .	299
6.21.2.58 RCC_IRQHandler . . . . .	299
6.21.2.59 EXTI0_IRQHandler . . . . .	299
6.21.2.60 EXTI1_IRQHandler . . . . .	299
6.21.2.61 EXTI2_IRQHandler . . . . .	300
6.21.2.62 EXTI3_IRQHandler . . . . .	300
6.21.2.63 EXTI4_IRQHandler . . . . .	300

6.21.2.64 DMA1_Ch1_IRQHandler . . . . .	300
6.21.2.65 DMA1_Ch2_IRQHandler . . . . .	300
6.21.2.66 DMA1_Ch3_IRQHandler . . . . .	300
6.21.2.67 DMA1_Ch4_IRQHandler . . . . .	300
6.21.2.68 DMA1_Ch5_IRQHandler . . . . .	300
6.21.2.69 DMA1_Ch6_IRQHandler . . . . .	300
6.21.2.70 DMA1_Ch7_IRQHandler . . . . .	300
6.21.2.71 ADC1_2_IRQHandler . . . . .	300
6.21.2.72 EXTI9_5_IRQHandler . . . . .	300
6.21.2.73 TIM1_BRK_IRQHandler . . . . .	301
6.21.2.74 TIM1_UP_IRQHandler . . . . .	301
6.21.2.75 TIM1_TRG_COM_IRQHandler . . . . .	301
6.21.2.76 TIM1_CC_IRQHandler . . . . .	301
6.21.2.77 TIM2_IRQHandler . . . . .	301
6.21.2.78 TIM3_IRQHandler . . . . .	301
6.21.2.79 TIM4_IRQHandler . . . . .	301
6.21.2.80 I2C1_EV_IRQHandler . . . . .	301
6.21.2.81 I2C1_ER_IRQHandler . . . . .	301
6.21.2.82 I2C2_EV_IRQHandler . . . . .	301
6.21.2.83 I2C2_ER_IRQHandler . . . . .	301
6.21.2.84 SPI1_IRQHandler . . . . .	301
6.21.2.85 SPI2_IRQHandler . . . . .	302
6.21.2.86 USART1_IRQHandler . . . . .	302
6.21.2.87 USART2_IRQHandler . . . . .	302
6.21.2.88 USART3_IRQHandler . . . . .	302
6.21.2.89 EXTI15_10_IRQHandler . . . . .	302
6.21.2.90 RTC_Alarm_IRQHandler . . . . .	302
6.21.2.91 CEC_IRQHandler . . . . .	302
6.21.2.92 TIM12_IRQHandler . . . . .	302
6.21.2.93 TIM13_IRQHandler . . . . .	302
6.21.2.94 TIM14_IRQHandler . . . . .	302
6.21.2.95 STM32_SW . . . . .	302
6.21.2.96 STM32_PLLSRC . . . . .	303
6.21.2.97 STM32_PLLXTPRE . . . . .	303
6.21.2.98 STM32_PLLMUL_VALUE . . . . .	303
6.21.2.99 STM32_HPRE . . . . .	303
6.21.2.100STM32_PPREG1 . . . . .	303
6.21.2.101STM32_PPREG2 . . . . .	303
6.21.2.102STM32_ADCPRE . . . . .	303
6.21.2.103STM32_MCOSEL . . . . .	303

6.21.2.104	STM32_RTCSEL	304
6.21.2.105	STM32_ACTIVATE_PLL	304
6.21.2.106	STM32_PLLMUL	304
6.21.2.107	STM32_PLLCLKIN	304
6.21.2.108	STM32_PLLCLKOUT	304
6.21.2.109	STM32_SYSCLK	304
6.21.2.110	STM32_HCLK	304
6.21.2.111	STM32_PCLK1	304
6.21.2.112	STM32_PCLK2	304
6.21.2.113	STM32_RTCCLK	304
6.21.2.114	STM32_ADCCLK	304
6.21.2.115	STM32_TIMCLK1	304
6.21.2.116	STM32_TIMCLK2	305
6.21.2.117	STM32_FLASHBITS	305
6.22	STM32F103 HAL Support	305
6.22.1	Detailed Description	305
6.22.2	Define Documentation	316
6.22.2.1	STM32_SYSCLK_MAX	316
6.22.2.2	STM32_HSECLK_MAX	316
6.22.2.3	STM32_HSECLK_MIN	316
6.22.2.4	STM32_LSECLK_MAX	317
6.22.2.5	STM32_LSECLK_MIN	317
6.22.2.6	STM32_PLLIN_MAX	317
6.22.2.7	STM32_PLLIN_MIN	317
6.22.2.8	STM32_PLLOUT_MAX	317
6.22.2.9	STM32_PLLOUT_MIN	317
6.22.2.10	STM32_PCLK1_MAX	317
6.22.2.11	STM32_PCLK2_MAX	317
6.22.2.12	STM32_ADCCLK_MAX	317
6.22.2.13	STM32_SW_HSI	317
6.22.2.14	STM32_SW_HSE	317
6.22.2.15	STM32_SW_PLL	317
6.22.2.16	STM32_HPRE_DIV1	318
6.22.2.17	STM32_HPRE_DIV2	318
6.22.2.18	STM32_HPRE_DIV4	318
6.22.2.19	STM32_HPRE_DIV8	318
6.22.2.20	STM32_HPRE_DIV16	318
6.22.2.21	STM32_HPRE_DIV64	318
6.22.2.22	STM32_HPRE_DIV128	318
6.22.2.23	STM32_HPRE_DIV256	318

6.22.2.24 STM32_HPRE_DIV512 . . . . .	318
6.22.2.25 STM32_PPREG1_DIV1 . . . . .	318
6.22.2.26 STM32_PPREG1_DIV2 . . . . .	318
6.22.2.27 STM32_PPREG1_DIV4 . . . . .	318
6.22.2.28 STM32_PPREG1_DIV8 . . . . .	319
6.22.2.29 STM32_PPREG1_DIV16 . . . . .	319
6.22.2.30 STM32_PPREG2_DIV1 . . . . .	319
6.22.2.31 STM32_PPREG2_DIV2 . . . . .	319
6.22.2.32 STM32_PPREG2_DIV4 . . . . .	319
6.22.2.33 STM32_PPREG2_DIV8 . . . . .	319
6.22.2.34 STM32_PPREG2_DIV16 . . . . .	319
6.22.2.35 STM32_ADCPRE_DIV2 . . . . .	319
6.22.2.36 STM32_ADCPRE_DIV4 . . . . .	319
6.22.2.37 STM32_ADCPRE_DIV6 . . . . .	319
6.22.2.38 STM32_ADCPRE_DIV8 . . . . .	319
6.22.2.39 STM32_PLLSRC_HSI . . . . .	319
6.22.2.40 STM32_PLLSRC_HSE . . . . .	320
6.22.2.41 STM32_PLLXTPRE_DIV1 . . . . .	320
6.22.2.42 STM32_PLLXTPRE_DIV2 . . . . .	320
6.22.2.43 STM32_USBPRE_DIV1P5 . . . . .	320
6.22.2.44 STM32_USBPRE_DIV1 . . . . .	320
6.22.2.45 STM32_MCOSEL_NOCLOCK . . . . .	320
6.22.2.46 STM32_MCOSEL_SYSCLK . . . . .	320
6.22.2.47 STM32_MCOSEL_HSI . . . . .	320
6.22.2.48 STM32_MCOSEL_HSE . . . . .	320
6.22.2.49 STM32_MCOSEL_PLLDIV2 . . . . .	320
6.22.2.50 STM32_RTCSEL_MASK . . . . .	320
6.22.2.51 STM32_RTCSEL_NOCLOCK . . . . .	320
6.22.2.52 STM32_RTCSEL_LSE . . . . .	321
6.22.2.53 STM32_RTCSEL_LSI . . . . .	321
6.22.2.54 STM32_RTCSEL_HSEDIV . . . . .	321
6.22.2.55 WWDG_IRQHandler . . . . .	321
6.22.2.56 PVD_IRQHandler . . . . .	321
6.22.2.57 TAMPER_IRQHandler . . . . .	321
6.22.2.58 RTC_IRQHandler . . . . .	321
6.22.2.59 FLASH_IRQHandler . . . . .	321
6.22.2.60 RCC_IRQHandler . . . . .	321
6.22.2.61 EXTI0_IRQHandler . . . . .	321
6.22.2.62 EXTI1_IRQHandler . . . . .	321
6.22.2.63 EXTI2_IRQHandler . . . . .	321

6.22.2.64 EXTI3_IRQHandler . . . . .	322
6.22.2.65 EXTI4_IRQHandler . . . . .	322
6.22.2.66 DMA1_Ch1_IRQHandler . . . . .	322
6.22.2.67 DMA1_Ch2_IRQHandler . . . . .	322
6.22.2.68 DMA1_Ch3_IRQHandler . . . . .	322
6.22.2.69 DMA1_Ch4_IRQHandler . . . . .	322
6.22.2.70 DMA1_Ch5_IRQHandler . . . . .	322
6.22.2.71 DMA1_Ch6_IRQHandler . . . . .	322
6.22.2.72 DMA1_Ch7_IRQHandler . . . . .	322
6.22.2.73 ADC1_2_IRQHandler . . . . .	322
6.22.2.74 CAN1_TX_IRQHandler . . . . .	322
6.22.2.75 USB_HP_IRQHandler . . . . .	322
6.22.2.76 CAN1_RX0_IRQHandler . . . . .	323
6.22.2.77 USB_LP_IRQHandler . . . . .	323
6.22.2.78 CAN1_RX1_IRQHandler . . . . .	323
6.22.2.79 CAN1_SCE_IRQHandler . . . . .	323
6.22.2.80 EXTI9_5_IRQHandler . . . . .	323
6.22.2.81 TIM1_BRK_IRQHandler . . . . .	323
6.22.2.82 TIM1_UP_IRQHandler . . . . .	323
6.22.2.83 TIM1_TRG_COM_IRQHandler . . . . .	323
6.22.2.84 TIM1_CC_IRQHandler . . . . .	323
6.22.2.85 TIM2_IRQHandler . . . . .	323
6.22.2.86 TIM3_IRQHandler . . . . .	323
6.22.2.87 TIM4_IRQHandler . . . . .	323
6.22.2.88 I2C1_EV_IRQHandler . . . . .	324
6.22.2.89 I2C1_ER_IRQHandler . . . . .	324
6.22.2.90 I2C2_EV_IRQHandler . . . . .	324
6.22.2.91 I2C2_ER_IRQHandler . . . . .	324
6.22.2.92 SPI1_IRQHandler . . . . .	324
6.22.2.93 SPI2_IRQHandler . . . . .	324
6.22.2.94 USART1_IRQHandler . . . . .	324
6.22.2.95 USART2_IRQHandler . . . . .	324
6.22.2.96 USART3_IRQHandler . . . . .	324
6.22.2.97 EXTI15_10_IRQHandler . . . . .	324
6.22.2.98 RTC_Alarm_IRQHandler . . . . .	324
6.22.2.99 USB_FS_WKUP_IRQHandler . . . . .	324
6.22.2.100TIM8_BRK_IRQHandler . . . . .	325
6.22.2.101TIM8_UP_IRQHandler . . . . .	325
6.22.2.102TIM8_TRG_COM_IRQHandler . . . . .	325
6.22.2.103TIM8_CC_IRQHandler . . . . .	325

6.22.2.104ADC3_IRQHandler . . . . .	325
6.22.2.105FSMC_IRQHandler . . . . .	325
6.22.2.106SDIO_IRQHandler . . . . .	325
6.22.2.107TIM5_IRQHandler . . . . .	325
6.22.2.108SPI3_IRQHandler . . . . .	325
6.22.2.109UART4_IRQHandler . . . . .	325
6.22.2.110UART5_IRQHandler . . . . .	325
6.22.2.111TIM6_IRQHandler . . . . .	325
6.22.2.112TIM7_IRQHandler . . . . .	326
6.22.2.113DMA2_Ch1_IRQHandler . . . . .	326
6.22.2.114DMA2_Ch2_IRQHandler . . . . .	326
6.22.2.115DMA2_Ch3_IRQHandler . . . . .	326
6.22.2.116DMA2_Ch4_5_IRQHandler . . . . .	326
6.22.2.117STM32_SW . . . . .	326
6.22.2.118STM32_PLLSRC . . . . .	326
6.22.2.119STM32_PLLXTPRE . . . . .	326
6.22.2.120STM32_PLLMUL_VALUE . . . . .	327
6.22.2.121STM32_HPRE . . . . .	327
6.22.2.122STM32_PPREG . . . . .	327
6.22.2.123STM32_PPREG2 . . . . .	327
6.22.2.124STM32_ADCPRE . . . . .	327
6.22.2.125STM32_USB_CLOCK_REQUIRED . . . . .	327
6.22.2.126STM32_USBPRE . . . . .	327
6.22.2.127STM32_MCOSEL . . . . .	327
6.22.2.128STM32_RTCSEL . . . . .	327
6.22.2.129STM32_ACTIVATE_PLL . . . . .	327
6.22.2.130STM32_PLLMUL . . . . .	328
6.22.2.131STM32_PLLCLKIN . . . . .	328
6.22.2.132STM32_PLLCLKOUT . . . . .	328
6.22.2.133STM32_SYSCLK . . . . .	328
6.22.2.134STM32_HCLK . . . . .	328
6.22.2.135STM32_PCLK1 . . . . .	328
6.22.2.136STM32_PCLK2 . . . . .	328
6.22.2.137STM32_RTCCLK . . . . .	328
6.22.2.138STM32_ADCCLK . . . . .	328
6.22.2.139STM32_USBCLK . . . . .	328
6.22.2.140STM32_TIMCLK1 . . . . .	328
6.22.2.141STM32_TIMCLK2 . . . . .	328
6.22.2.142STM32_FLASHBITS . . . . .	329
6.23 STM32F105/F107 HAL Support . . . . .	329

6.23.1	Detailed Description	329
6.23.2	Define Documentation	336
6.23.2.1	STM32_SYSCLK_MAX	336
6.23.2.2	STM32_HSECLK_MAX	336
6.23.2.3	STM32_HSECLK_MIN	336
6.23.2.4	STM32_LSECLK_MAX	336
6.23.2.5	STM32_LSECLK_MIN	336
6.23.2.6	STM32_PLL1IN_MAX	336
6.23.2.7	STM32_PLL1IN_MIN	336
6.23.2.8	STM32_PLL23IN_MAX	336
6.23.2.9	STM32_PLL23IN_MIN	336
6.23.2.10	STM32_PLL1VCO_MAX	336
6.23.2.11	STM32_PLL1VCO_MIN	337
6.23.2.12	STM32_PLL23VCO_MAX	337
6.23.2.13	STM32_PLL23VCO_MIN	337
6.23.2.14	STM32_PCLK1_MAX	337
6.23.2.15	STM32_PCLK2_MAX	337
6.23.2.16	STM32_ADCCLK_MAX	337
6.23.2.17	STM32_SPII2S_MAX	337
6.23.2.18	STM32_SW_HSI	337
6.23.2.19	STM32_SW_HSE	337
6.23.2.20	STM32_SW_PLL	337
6.23.2.21	STM32_HPRE_DIV1	337
6.23.2.22	STM32_HPRE_DIV2	337
6.23.2.23	STM32_HPRE_DIV4	338
6.23.2.24	STM32_HPRE_DIV8	338
6.23.2.25	STM32_HPRE_DIV16	338
6.23.2.26	STM32_HPRE_DIV64	338
6.23.2.27	STM32_HPRE_DIV128	338
6.23.2.28	STM32_HPRE_DIV256	338
6.23.2.29	STM32_HPRE_DIV512	338
6.23.2.30	STM32_PPREG1_DIV1	338
6.23.2.31	STM32_PPREG1_DIV2	338
6.23.2.32	STM32_PPREG1_DIV4	338
6.23.2.33	STM32_PPREG1_DIV8	338
6.23.2.34	STM32_PPREG1_DIV16	338
6.23.2.35	STM32_PPREG2_DIV1	339
6.23.2.36	STM32_PPREG2_DIV2	339
6.23.2.37	STM32_PPREG2_DIV4	339
6.23.2.38	STM32_PPREG2_DIV8	339

6.23.2.39 STM32_PPREG2_DIV16 . . . . .	339
6.23.2.40 STM32_ADCPRE_DIV2 . . . . .	339
6.23.2.41 STM32_ADCPRE_DIV4 . . . . .	339
6.23.2.42 STM32_ADCPRE_DIV6 . . . . .	339
6.23.2.43 STM32_ADCPRE_DIV8 . . . . .	339
6.23.2.44 STM32_PLLSRC_HSI . . . . .	339
6.23.2.45 STM32_PLLSRC_PREDIV1 . . . . .	339
6.23.2.46 STM32_OTGFSPRE_DIV2 . . . . .	339
6.23.2.47 STM32_OTGFSPRE_DIV3 . . . . .	340
6.23.2.48 STM32_MCOSEL_NOCLOCK . . . . .	340
6.23.2.49 STM32_MCOSEL_SYSCLK . . . . .	340
6.23.2.50 STM32_MCOSEL_HSI . . . . .	340
6.23.2.51 STM32_MCOSEL_HSE . . . . .	340
6.23.2.52 STM32_MCOSEL_PLLDIV2 . . . . .	340
6.23.2.53 STM32_MCOSEL_PLL2 . . . . .	340
6.23.2.54 STM32_MCOSEL_PLL3DIV2 . . . . .	340
6.23.2.55 STM32_MCOSEL_XT1 . . . . .	340
6.23.2.56 STM32_MCOSEL_PLL3 . . . . .	340
6.23.2.57 STM32_RTCSEL_MASK . . . . .	340
6.23.2.58 STM32_RTCSEL_NOCLOCK . . . . .	340
6.23.2.59 STM32_RTCSEL_LSE . . . . .	341
6.23.2.60 STM32_RTCSEL_LSI . . . . .	341
6.23.2.61 STM32_RTCSEL_HSEDIV . . . . .	341
6.23.2.62 STM32_PREDIV1SRC_HSE . . . . .	341
6.23.2.63 STM32_PREDIV1SRC_PLL2 . . . . .	341
6.23.2.64 WWDG_IRQHandler . . . . .	341
6.23.2.65 PVD_IRQHandler . . . . .	341
6.23.2.66 TAMPER_IRQHandler . . . . .	341
6.23.2.67 RTC_IRQHandler . . . . .	341
6.23.2.68 FLASH_IRQHandler . . . . .	341
6.23.2.69 RCC_IRQHandler . . . . .	341
6.23.2.70 EXTI0_IRQHandler . . . . .	341
6.23.2.71 EXTI1_IRQHandler . . . . .	342
6.23.2.72 EXTI2_IRQHandler . . . . .	342
6.23.2.73 EXTI3_IRQHandler . . . . .	342
6.23.2.74 EXTI4_IRQHandler . . . . .	342
6.23.2.75 DMA1_Ch1_IRQHandler . . . . .	342
6.23.2.76 DMA1_Ch2_IRQHandler . . . . .	342
6.23.2.77 DMA1_Ch3_IRQHandler . . . . .	342
6.23.2.78 DMA1_Ch4_IRQHandler . . . . .	342

6.23.2.79 DMA1_Ch5_IRQHandler . . . . .	342
6.23.2.80 DMA1_Ch6_IRQHandler . . . . .	342
6.23.2.81 DMA1_Ch7_IRQHandler . . . . .	342
6.23.2.82 ADC1_2_IRQHandler . . . . .	342
6.23.2.83 CAN1_TX_IRQHandler . . . . .	343
6.23.2.84 CAN1_RX0_IRQHandler . . . . .	343
6.23.2.85 CAN1_RX1_IRQHandler . . . . .	343
6.23.2.86 CAN1_SCE_IRQHandler . . . . .	343
6.23.2.87 EXTI9_5_IRQHandler . . . . .	343
6.23.2.88 TIM1_BRK_IRQHandler . . . . .	343
6.23.2.89 TIM1_UP_IRQHandler . . . . .	343
6.23.2.90 TIM1_TRG_COM_IRQHandler . . . . .	343
6.23.2.91 TIM1_CC_IRQHandler . . . . .	343
6.23.2.92 TIM2_IRQHandler . . . . .	343
6.23.2.93 TIM3_IRQHandler . . . . .	343
6.23.2.94 TIM4_IRQHandler . . . . .	343
6.23.2.95 I2C1_EV_IRQHandler . . . . .	344
6.23.2.96 I2C1_ER_IRQHandler . . . . .	344
6.23.2.97 I2C2_EV_IRQHandler . . . . .	344
6.23.2.98 I2C2_ER_IRQHandler . . . . .	344
6.23.2.99 SPI1_IRQHandler . . . . .	344
6.23.2.100SPI2_IRQHandler . . . . .	344
6.23.2.101USART1_IRQHandler . . . . .	344
6.23.2.102USART2_IRQHandler . . . . .	344
6.23.2.103USART3_IRQHandler . . . . .	344
6.23.2.104EXTI15_10_IRQHandler . . . . .	344
6.23.2.105RTC_Alarm_IRQHandler . . . . .	344
6.23.2.106OTG_FS_WKUP_IRQHandler . . . . .	344
6.23.2.107TIM5_IRQHandler . . . . .	345
6.23.2.108SPI3_IRQHandler . . . . .	345
6.23.2.109UART4_IRQHandler . . . . .	345
6.23.2.110UART5_IRQHandler . . . . .	345
6.23.2.111TIM6_IRQHandler . . . . .	345
6.23.2.112TIM7_IRQHandler . . . . .	345
6.23.2.113DMA2_Ch1_IRQHandler . . . . .	345
6.23.2.114DMA2_Ch2_IRQHandler . . . . .	345
6.23.2.115DMA2_Ch3_IRQHandler . . . . .	345
6.23.2.116DMA2_Ch4_IRQHandler . . . . .	345
6.23.2.117DMA2_Ch5_IRQHandler . . . . .	345
6.23.2.118ETH_IRQHandler . . . . .	345

6.23.2.119ETH_WKUP_IRQHandler . . . . .	346
6.23.2.120CAN2_TX_IRQHandler . . . . .	346
6.23.2.121CAN2_RX0_IRQHandler . . . . .	346
6.23.2.122CAN2_RX1_IRQHandler . . . . .	346
6.23.2.123CAN2_SCE_IRQHandler . . . . .	346
6.23.2.124OTG_FS_IRQHandler . . . . .	346
6.23.2.125STM32_SW . . . . .	346
6.23.2.126STM32_PLLSRC . . . . .	346
6.23.2.127STM32_PREDIV1SRC . . . . .	346
6.23.2.128STM32_PREDIV1_VALUE . . . . .	347
6.23.2.129STM32_PLLMUL_VALUE . . . . .	347
6.23.2.130STM32_PREDIV2_VALUE . . . . .	347
6.23.2.131STM32_PLL2MUL_VALUE . . . . .	347
6.23.2.132STM32_PLL3MUL_VALUE . . . . .	347
6.23.2.133STM32_HPRE . . . . .	347
6.23.2.134STM32_PPREG . . . . .	347
6.23.2.135STM32_PPREG2 . . . . .	348
6.23.2.136STM32_ADCPRE . . . . .	348
6.23.2.137STM32_OTG_CLOCK_REQUIRED . . . . .	348
6.23.2.138STM32_OTGFSPREG . . . . .	348
6.23.2.139STM32_I2S_CLOCK_REQUIRED . . . . .	348
6.23.2.140STM32_MCOSEL . . . . .	348
6.23.2.141STM32_RTCSEL . . . . .	348
6.23.2.142STM32_ACTIVATE_PLL1 . . . . .	348
6.23.2.143STM32_ACTIVATE_PLL2 . . . . .	348
6.23.2.144STM32_ACTIVATE_PLL3 . . . . .	348
6.23.2.145STM32_PREDIV1 . . . . .	348
6.23.2.146STM32_PREDIV2 . . . . .	348
6.23.2.147STM32_PLLMUL . . . . .	349
6.23.2.148STM32_PLL2MUL . . . . .	349
6.23.2.149STM32_PLL3MUL . . . . .	349
6.23.2.150STM32_PLL2CLKIN . . . . .	349
6.23.2.151STM32_PLL2CLKOUT . . . . .	349
6.23.2.152STM32_PLL2VCO . . . . .	349
6.23.2.153STM32_PLL3CLKIN . . . . .	349
6.23.2.154STM32_PLL3CLKOUT . . . . .	349
6.23.2.155STM32_PLL3VCO . . . . .	349
6.23.2.156STM32_PREDIV1CLK . . . . .	349
6.23.2.157STM32_PLLCLKIN . . . . .	349
6.23.2.158STM32_PLLCLKOUT . . . . .	349

6.23.2.159STM32_PLLVCO . . . . .	350
6.23.2.160STM32_SYSCLK . . . . .	350
6.23.2.161STM32_HCLK . . . . .	350
6.23.2.162STM32_PCLK1 . . . . .	350
6.23.2.163STM32_PCLK2 . . . . .	350
6.23.2.164STM32_RTCCLK . . . . .	350
6.23.2.165STM32_ADCCLK . . . . .	350
6.23.2.166STM32_OTGFSCLK . . . . .	350
6.23.2.167STM32_TIMCLK1 . . . . .	350
6.23.2.168STM32_TIMCLK2 . . . . .	350
6.23.2.169STM32_FLASHBITS . . . . .	350
6.24 STM32F1xx Drivers . . . . .	350
6.24.1 Detailed Description . . . . .	351
6.25 STM32F1xx Initialization Support . . . . .	351
6.25.1 Supported HW resources . . . . .	351
6.25.2 STM32F1xx HAL driver implementation features . . . . .	351
6.26 STM32F1xx ADC Support . . . . .	352
6.26.1 Supported HW resources . . . . .	352
6.26.2 STM32F1xx ADC driver implementation features . . . . .	352
6.27 STM32F1xx CAN Support . . . . .	352
6.27.1 Supported HW resources . . . . .	352
6.27.2 STM32F1xx CAN driver implementation features . . . . .	352
6.28 STM32F1xx EXT Support . . . . .	352
6.28.1 Supported HW resources . . . . .	352
6.28.2 STM32F1xx EXT driver implementation features . . . . .	353
6.29 STM32F1xx GPT Support . . . . .	353
6.29.1 Supported HW resources . . . . .	353
6.29.2 STM32F1xx GPT driver implementation features . . . . .	353
6.30 STM32F1xx I2C Support . . . . .	353
6.30.1 Supported HW resources . . . . .	353
6.30.2 STM32F1xx I2C driver implementation features . . . . .	353
6.31 STM32F1xx ICU Support . . . . .	353
6.31.1 Supported HW resources . . . . .	354
6.31.2 STM32F1xx ICU driver implementation features . . . . .	354
6.32 STM32F1xx MAC Support . . . . .	354
6.32.1 Supported HW resources . . . . .	354
6.33 STM32F1xx PAL Support . . . . .	354
6.33.1 Supported HW resources . . . . .	354
6.33.2 STM32F1xx PAL driver implementation features . . . . .	354
6.33.3 Supported PAL setup modes . . . . .	355

6.33.4 Suboptimal behavior . . . . .	355
6.34 STM32F1xx PWM Support . . . . .	355
6.34.1 Supported HW resources . . . . .	355
6.34.2 STM32F1xx PWM driver implementation features . . . . .	356
6.35 STM32F1xx RTC Support . . . . .	356
6.35.1 Supported HW resources . . . . .	356
6.36 STM32F1xx SDC Support . . . . .	356
6.36.1 Supported HW resources . . . . .	356
6.36.2 STM32F1xx SDC driver implementation features . . . . .	356
6.37 STM32F1xx Serial Support . . . . .	356
6.37.1 Supported HW resources . . . . .	356
6.37.2 STM32F1xx Serial driver implementation features . . . . .	357
6.38 STM32F1xx SPI Support . . . . .	357
6.38.1 Supported HW resources . . . . .	357
6.38.2 STM32F1xx SPI driver implementation features . . . . .	357
6.39 STM32F1xx UART Support . . . . .	357
6.39.1 Supported HW resources . . . . .	357
6.39.2 STM32F1xx UART driver implementation features . . . . .	358
6.40 STM32F1xx USB Support . . . . .	358
6.40.1 Supported HW resources . . . . .	358
6.40.2 STM32F1xx USB driver implementation features . . . . .	358
6.41 STM32F1xx Platform Drivers . . . . .	358
6.41.1 Detailed Description . . . . .	358
6.42 STM32F1xx DMA Support . . . . .	358
6.42.1 Detailed Description . . . . .	358
6.42.2 Supported HW resources . . . . .	359
6.42.3 STM32F1xx DMA driver implementation features . . . . .	359
6.42.4 Function Documentation . . . . .	362
6.42.4.1 CH_IRQ_HANDLER . . . . .	362
6.42.4.2 CH_IRQ_HANDLER . . . . .	362
6.42.4.3 CH_IRQ_HANDLER . . . . .	362
6.42.4.4 CH_IRQ_HANDLER . . . . .	363
6.42.4.5 CH_IRQ_HANDLER . . . . .	363
6.42.4.6 CH_IRQ_HANDLER . . . . .	363
6.42.4.7 CH_IRQ_HANDLER . . . . .	363
6.42.4.8 CH_IRQ_HANDLER . . . . .	363
6.42.4.9 CH_IRQ_HANDLER . . . . .	363
6.42.4.10 CH_IRQ_HANDLER . . . . .	363
6.42.4.11 CH_IRQ_HANDLER . . . . .	364
6.42.4.12 CH_IRQ_HANDLER . . . . .	364

6.42.4.13	dmaInit . . . . .	364
6.42.4.14	dmaStreamAllocate . . . . .	364
6.42.4.15	dmaStreamRelease . . . . .	365
6.42.5	Variable Documentation . . . . .	365
6.42.5.1	_stm32_dma_streams . . . . .	365
6.42.6	Define Documentation . . . . .	366
6.42.6.1	STM32_DMA1_STREAMS_MASK . . . . .	366
6.42.6.2	STM32_DMA2_STREAMS_MASK . . . . .	366
6.42.6.3	STM32_DMA_CCR_RESET_VALUE . . . . .	366
6.42.6.4	STM32_DMA_STREAMS . . . . .	366
6.42.6.5	STM32_DMA_ISR_MASK . . . . .	366
6.42.6.6	STM32_DMA_GETCHANNEL . . . . .	366
6.42.6.7	STM32_DMA_STREAM_ID_MSK . . . . .	366
6.42.6.8	STM32_DMA_IS_VALID_ID . . . . .	367
6.42.6.9	STM32_DMA_STREAM_ID . . . . .	367
6.42.6.10	STM32_DMA_STREAM . . . . .	367
6.42.6.11	STM32_DMA_CR_DMEIE . . . . .	367
6.42.6.12	STM32_DMA_CR_CHSEL_MASK . . . . .	367
6.42.6.13	STM32_DMA_CR_CHSEL . . . . .	367
6.42.6.14	dmaStreamSetPeripheral . . . . .	368
6.42.6.15	dmaStreamSetMemory0 . . . . .	368
6.42.6.16	dmaStreamSetTransactionSize . . . . .	369
6.42.6.17	dmaStreamGetTransactionSize . . . . .	369
6.42.6.18	dmaStreamSetMode . . . . .	369
6.42.6.19	dmaStreamEnable . . . . .	370
6.42.6.20	dmaStreamDisable . . . . .	370
6.42.6.21	dmaStreamClearInterrupt . . . . .	371
6.42.6.22	dmaStartMemcpy . . . . .	371
6.42.6.23	dmaWaitCompletion . . . . .	372
6.42.7	Typedef Documentation . . . . .	372
6.42.7.1	stm32_dmaisr_t . . . . .	373
6.43	STM32F1xx RCC Support . . . . .	373
6.43.1	Detailed Description . . . . .	373
6.43.2	Supported HW resources . . . . .	373
6.43.3	STM32F1xx RCC driver implementation features . . . . .	373
6.43.4	Define Documentation . . . . .	377
6.43.4.1	rccEnableAPB1 . . . . .	377
6.43.4.2	rccDisableAPB1 . . . . .	378
6.43.4.3	rccResetAPB1 . . . . .	378
6.43.4.4	rccEnableAPB2 . . . . .	378

6.43.4.5 rccDisableAPB2 . . . . .	379
6.43.4.6 rccResetAPB2 . . . . .	379
6.43.4.7 rccEnableAHB . . . . .	380
6.43.4.8 rccDisableAHB . . . . .	380
6.43.4.9 rccResetAHB . . . . .	380
6.43.4.10 rccEnableADC1 . . . . .	381
6.43.4.11 rccDisableADC1 . . . . .	381
6.43.4.12 rccResetADC1 . . . . .	381
6.43.4.13 rccEnableBKPIInterface . . . . .	381
6.43.4.14 rccDisableBKPIInterface . . . . .	382
6.43.4.15 rccResetBKPIInterface . . . . .	382
6.43.4.16 rccResetBKP . . . . .	382
6.43.4.17 rccEnablePWRInterface . . . . .	382
6.43.4.18 rccDisablePWRInterface . . . . .	382
6.43.4.19 rccResetPWRInterface . . . . .	383
6.43.4.20 rccEnableCAN1 . . . . .	383
6.43.4.21 rccDisableCAN1 . . . . .	383
6.43.4.22 rccResetCAN1 . . . . .	383
6.43.4.23 rccEnableDMA1 . . . . .	384
6.43.4.24 rccDisableDMA1 . . . . .	384
6.43.4.25 rccResetDMA1 . . . . .	384
6.43.4.26 rccEnableDMA2 . . . . .	384
6.43.4.27 rccDisableDMA2 . . . . .	385
6.43.4.28 rccResetDMA2 . . . . .	385
6.43.4.29 rccEnableETH . . . . .	385
6.43.4.30 rccDisableETH . . . . .	385
6.43.4.31 rccResetETH . . . . .	386
6.43.4.32 rccEnableI2C1 . . . . .	386
6.43.4.33 rccDisableI2C1 . . . . .	386
6.43.4.34 rccResetI2C1 . . . . .	386
6.43.4.35 rccEnableI2C2 . . . . .	387
6.43.4.36 rccDisableI2C2 . . . . .	387
6.43.4.37 rccResetI2C2 . . . . .	387
6.43.4.38 rccEnableSDIO . . . . .	387
6.43.4.39 rccDisableSDIO . . . . .	388
6.43.4.40 rccResetSDIO . . . . .	388
6.43.4.41 rccEnableSPI1 . . . . .	388
6.43.4.42 rccDisableSPI1 . . . . .	388
6.43.4.43 rccResetSPI1 . . . . .	389
6.43.4.44 rccEnableSPI2 . . . . .	389

6.43.4.45 rccDisableSPI2 . . . . .	389
6.43.4.46 rccResetSPI2 . . . . .	389
6.43.4.47 rccEnableSPI3 . . . . .	389
6.43.4.48 rccDisableSPI3 . . . . .	390
6.43.4.49 rccResetSPI3 . . . . .	390
6.43.4.50 rccEnableTIM1 . . . . .	390
6.43.4.51 rccDisableTIM1 . . . . .	390
6.43.4.52 rccResetTIM1 . . . . .	391
6.43.4.53 rccEnableTIM2 . . . . .	391
6.43.4.54 rccDisableTIM2 . . . . .	391
6.43.4.55 rccResetTIM2 . . . . .	391
6.43.4.56 rccEnableTIM3 . . . . .	391
6.43.4.57 rccDisableTIM3 . . . . .	392
6.43.4.58 rccResetTIM3 . . . . .	392
6.43.4.59 rccEnableTIM4 . . . . .	392
6.43.4.60 rccDisableTIM4 . . . . .	392
6.43.4.61 rccResetTIM4 . . . . .	393
6.43.4.62 rccEnableTIM5 . . . . .	393
6.43.4.63 rccDisableTIM5 . . . . .	393
6.43.4.64 rccResetTIM5 . . . . .	393
6.43.4.65 rccEnableTIM8 . . . . .	394
6.43.4.66 rccDisableTIM8 . . . . .	394
6.43.4.67 rccResetTIM8 . . . . .	394
6.43.4.68 rccEnableUSART1 . . . . .	394
6.43.4.69 rccDisableUSART1 . . . . .	395
6.43.4.70 rccResetUSART1 . . . . .	395
6.43.4.71 rccEnableUSART2 . . . . .	395
6.43.4.72 rccDisableUSART2 . . . . .	395
6.43.4.73 rccResetUSART2 . . . . .	396
6.43.4.74 rccEnableUSART3 . . . . .	396
6.43.4.75 rccDisableUSART3 . . . . .	396
6.43.4.76 rccResetUSART3 . . . . .	396
6.43.4.77 rccEnableUART4 . . . . .	396
6.43.4.78 rccDisableUART4 . . . . .	397
6.43.4.79 rccResetUART4 . . . . .	397
6.43.4.80 rccEnableUART5 . . . . .	397
6.43.4.81 rccDisableUART5 . . . . .	397
6.43.4.82 rccResetUART5 . . . . .	398
6.43.4.83 rccEnableUSB . . . . .	398
6.43.4.84 rccDisableUSB . . . . .	398

6.43.4.85 rccResetUSB . . . . .	398
<b>7 Data Structure Documentation</b>	<b>399</b>
7.1 ADCConfig Struct Reference . . . . .	399
7.1.1 Detailed Description . . . . .	399
7.2 ADCConversionGroup Struct Reference . . . . .	399
7.2.1 Detailed Description . . . . .	399
7.2.2 Field Documentation . . . . .	401
7.2.2.1 circular . . . . .	401
7.2.2.2 num_channels . . . . .	401
7.2.2.3 end_cb . . . . .	401
7.2.2.4 error_cb . . . . .	401
7.2.2.5 cr1 . . . . .	401
7.2.2.6 cr2 . . . . .	401
7.2.2.7 smpr1 . . . . .	401
7.2.2.8 smpr2 . . . . .	402
7.2.2.9 sqr1 . . . . .	402
7.2.2.10 sqr2 . . . . .	402
7.2.2.11 sqr3 . . . . .	402
7.3 ADCDriver Struct Reference . . . . .	402
7.3.1 Detailed Description . . . . .	402
7.3.2 Field Documentation . . . . .	404
7.3.2.1 state . . . . .	404
7.3.2.2 config . . . . .	404
7.3.2.3 samples . . . . .	404
7.3.2.4 depth . . . . .	404
7.3.2.5 grpp . . . . .	404
7.3.2.6 thread . . . . .	404
7.3.2.7 mutex . . . . .	404
7.3.2.8 adc . . . . .	405
7.3.2.9 dmastp . . . . .	405
7.3.2.10 dmamode . . . . .	405
7.4 CANConfig Struct Reference . . . . .	405
7.4.1 Detailed Description . . . . .	405
7.4.2 Field Documentation . . . . .	406
7.4.2.1 mcr . . . . .	406
7.4.2.2 btr . . . . .	406
7.4.2.3 num . . . . .	406
7.4.2.4 filters . . . . .	406
7.5 CANDriver Struct Reference . . . . .	406

7.5.1	Detailed Description	406
7.5.2	Field Documentation	408
7.5.2.1	state	408
7.5.2.2	config	408
7.5.2.3	txsem	408
7.5.2.4	rxsem	408
7.5.2.5	rxfull_event	408
7.5.2.6	txempty_event	408
7.5.2.7	error_event	409
7.5.2.8	status	409
7.5.2.9	sleep_event	409
7.5.2.10	wakeup_event	409
7.5.2.11	can	409
7.6	CANFilter Struct Reference	409
7.6.1	Detailed Description	409
7.6.2	Field Documentation	409
7.6.2.1	mode	409
7.6.2.2	scale	410
7.6.2.3	assignment	410
7.6.2.4	register1	410
7.6.2.5	register2	410
7.7	CANRxFrame Struct Reference	410
7.7.1	Detailed Description	410
7.7.2	Field Documentation	410
7.7.2.1	FMI	410
7.7.2.2	TIME	410
7.7.2.3	DLC	411
7.7.2.4	RTR	411
7.7.2.5	IDE	411
7.7.2.6	SID	411
7.7.2.7	EID	411
7.7.2.8	data8	411
7.7.2.9	data16	411
7.7.2.10	data32	411
7.8	CANTxFrame Struct Reference	411
7.8.1	Detailed Description	411
7.8.2	Field Documentation	411
7.8.2.1	DLC	411
7.8.2.2	RTR	412
7.8.2.3	IDE	412

7.8.2.4	SID . . . . .	412
7.8.2.5	EID . . . . .	412
7.8.2.6	data8 . . . . .	412
7.8.2.7	data16 . . . . .	412
7.8.2.8	data32 . . . . .	412
7.9	EXTChannelConfig Struct Reference . . . . .	412
7.9.1	Detailed Description . . . . .	412
7.9.2	Field Documentation . . . . .	413
7.9.2.1	mode . . . . .	413
7.9.2.2	cb . . . . .	413
7.10	EXTConfig Struct Reference . . . . .	414
7.10.1	Detailed Description . . . . .	414
7.10.2	Field Documentation . . . . .	414
7.10.2.1	channels . . . . .	415
7.10.2.2	exti . . . . .	415
7.11	EXTDriver Struct Reference . . . . .	415
7.11.1	Detailed Description . . . . .	415
7.11.2	Field Documentation . . . . .	416
7.11.2.1	state . . . . .	416
7.11.2.2	config . . . . .	416
7.12	GPTConfig Struct Reference . . . . .	416
7.12.1	Detailed Description . . . . .	416
7.12.2	Field Documentation . . . . .	418
7.12.2.1	frequency . . . . .	418
7.12.2.2	callback . . . . .	418
7.13	GPTDriver Struct Reference . . . . .	418
7.13.1	Detailed Description . . . . .	418
7.13.2	Field Documentation . . . . .	420
7.13.2.1	state . . . . .	420
7.13.2.2	config . . . . .	420
7.13.2.3	clock . . . . .	420
7.13.2.4	tim . . . . .	420
7.14	I2CConfig Struct Reference . . . . .	420
7.14.1	Detailed Description . . . . .	420
7.14.2	Field Documentation . . . . .	420
7.14.2.1	op_mode . . . . .	420
7.14.2.2	clock_speed . . . . .	421
7.14.2.3	duty_cycle . . . . .	421
7.15	I2CDriver Struct Reference . . . . .	421
7.15.1	Detailed Description . . . . .	421

7.15.2 Field Documentation . . . . .	422
7.15.2.1 state . . . . .	422
7.15.2.2 config . . . . .	422
7.15.2.3 errors . . . . .	422
7.15.2.4 mutex . . . . .	422
7.15.2.5 thread . . . . .	422
7.15.2.6 addr . . . . .	422
7.15.2.7 dmemode . . . . .	423
7.15.2.8 dmarx . . . . .	423
7.15.2.9 dmatx . . . . .	423
7.15.2.10 i2c . . . . .	423
7.16 ICUConfig Struct Reference . . . . .	423
7.16.1 Detailed Description . . . . .	423
7.16.2 Field Documentation . . . . .	425
7.16.2.1 mode . . . . .	425
7.16.2.2 frequency . . . . .	425
7.16.2.3 width_cb . . . . .	425
7.16.2.4 period_cb . . . . .	425
7.17 ICUDriver Struct Reference . . . . .	425
7.17.1 Detailed Description . . . . .	425
7.17.2 Field Documentation . . . . .	427
7.17.2.1 state . . . . .	427
7.17.2.2 config . . . . .	427
7.17.2.3 clock . . . . .	427
7.17.2.4 tim . . . . .	427
7.18 IOBus Struct Reference . . . . .	427
7.18.1 Detailed Description . . . . .	427
7.18.2 Field Documentation . . . . .	428
7.18.2.1 portid . . . . .	428
7.18.2.2 mask . . . . .	428
7.18.2.3 offset . . . . .	428
7.19 MMCCConfig Struct Reference . . . . .	428
7.19.1 Detailed Description . . . . .	428
7.20 MMCDriver Struct Reference . . . . .	428
7.20.1 Detailed Description . . . . .	428
7.20.2 Field Documentation . . . . .	430
7.20.2.1 state . . . . .	430
7.20.2.2 config . . . . .	430
7.20.2.3 spip . . . . .	430
7.20.2.4 lscfg . . . . .	430

7.20.2.5	hscfg	430
7.20.2.6	is_protected	430
7.20.2.7	is_inserted	430
7.20.2.8	inserted_event	431
7.20.2.9	removed_event	431
7.20.2.10	vt	431
7.20.2.11	cnt	431
7.21	PALConfig Struct Reference	431
7.21.1	Detailed Description	431
7.21.2	Field Documentation	433
7.21.2.1	PAData	433
7.21.2.2	PBData	433
7.21.2.3	PCData	433
7.21.2.4	PDData	433
7.21.2.5	PEData	433
7.21.2.6	PFData	433
7.21.2.7	PGData	433
7.22	PWMChannelConfig Struct Reference	433
7.22.1	Detailed Description	433
7.22.2	Field Documentation	435
7.22.2.1	mode	435
7.22.2.2	callback	435
7.23	PWMConfig Struct Reference	435
7.23.1	Detailed Description	435
7.23.2	Field Documentation	437
7.23.2.1	frequency	437
7.23.2.2	period	437
7.23.2.3	callback	437
7.23.2.4	channels	437
7.23.2.5	cr2	437
7.23.2.6	bdtr	438
7.24	PWMDriver Struct Reference	438
7.24.1	Detailed Description	438
7.24.2	Field Documentation	440
7.24.2.1	state	440
7.24.2.2	config	440
7.24.2.3	period	440
7.24.2.4	clock	440
7.24.2.5	tim	440
7.25	RTCAlarm Struct Reference	440

7.25.1	Detailed Description	440
7.25.2	Field Documentation	440
7.25.2.1	tv_sec	440
7.26	RTCCallbackConfig Struct Reference	441
7.26.1	Detailed Description	441
7.26.2	Field Documentation	441
7.26.2.1	callback	441
7.27	RTCDriver Struct Reference	441
7.27.1	Detailed Description	441
7.27.2	Field Documentation	442
7.27.2.1	callback	442
7.28	RTCTime Struct Reference	442
7.28.1	Detailed Description	442
7.28.2	Field Documentation	442
7.28.2.1	tv_sec	442
7.28.2.2	tv_msec	442
7.29	SDCConfig Struct Reference	443
7.29.1	Detailed Description	443
7.30	SDCDriver Struct Reference	443
7.30.1	Detailed Description	443
7.30.2	Field Documentation	444
7.30.2.1	state	444
7.30.2.2	config	444
7.30.2.3	cardmode	444
7.30.2.4	cid	444
7.30.2.5	csd	444
7.30.2.6	rca	444
7.30.2.7	thread	444
7.31	SerialConfig Struct Reference	444
7.31.1	Detailed Description	444
7.31.2	Field Documentation	445
7.31.2.1	sc_speed	445
7.31.2.2	sc_cr1	445
7.31.2.3	sc_cr2	445
7.31.2.4	sc_cr3	445
7.32	SerialDriver Struct Reference	445
7.32.1	Detailed Description	445
7.32.2	Field Documentation	446
7.32.2.1	vmt	446
7.33	SerialDriverVMT Struct Reference	446

7.33.1	Detailed Description	446
7.34	SerialUSBConfig Struct Reference	446
7.34.1	Detailed Description	446
7.34.2	Field Documentation	448
7.34.2.1	usbp	448
7.34.2.2	usb_config	448
7.35	SerialUSBDriver Struct Reference	448
7.35.1	Detailed Description	448
7.35.2	Field Documentation	448
7.35.2.1	vmt	449
7.36	SerialUSBDriverVMT Struct Reference	449
7.36.1	Detailed Description	449
7.37	SPIConfig Struct Reference	449
7.37.1	Detailed Description	449
7.37.2	Field Documentation	450
7.37.2.1	end_cb	450
7.37.2.2	ssport	450
7.37.2.3	sspad	450
7.37.2.4	cr1	450
7.38	SPIDriver Struct Reference	450
7.38.1	Detailed Description	450
7.38.2	Field Documentation	452
7.38.2.1	state	452
7.38.2.2	config	452
7.38.2.3	thread	452
7.38.2.4	mutex	452
7.38.2.5	spi	452
7.38.2.6	dmarx	452
7.38.2.7	dmatx	452
7.38.2.8	rxdmamode	452
7.38.2.9	txdmamode	452
7.39	stm32_dma_stream_t Struct Reference	453
7.39.1	Detailed Description	453
7.39.2	Field Documentation	453
7.39.2.1	channel	453
7.39.2.2	ifcr	453
7.39.2.3	ishift	453
7.39.2.4	selfindex	453
7.39.2.5	vector	453
7.40	stm32_gpio_setup_t Struct Reference	453

7.40.1	Detailed Description	453
7.40.2	Field Documentation	454
7.40.2.1	odr	454
7.40.2.2	crl	454
7.40.2.3	crh	454
7.41	stm32_tim_t Struct Reference	454
7.41.1	Detailed Description	454
7.42	stm32_usb_descriptor_t Struct Reference	454
7.42.1	Detailed Description	454
7.42.2	Field Documentation	455
7.42.2.1	TXADDR0	455
7.42.2.2	TXCOUNT0	455
7.42.2.3	TXCOUNT1	455
7.42.2.4	RXADDR0	455
7.42.2.5	RXCOUNT0	455
7.42.2.6	RXCOUNT1	455
7.43	stm32_usb_t Struct Reference	455
7.43.1	Detailed Description	455
7.43.2	Field Documentation	455
7.43.2.1	EPR	455
7.44	TimeMeasurement Struct Reference	456
7.44.1	Detailed Description	456
7.44.2	Field Documentation	456
7.44.2.1	start	456
7.44.2.2	stop	456
7.44.2.3	last	456
7.44.2.4	worst	456
7.44.2.5	best	456
7.45	UARTConfig Struct Reference	456
7.45.1	Detailed Description	456
7.45.2	Field Documentation	458
7.45.2.1	txend1_cb	458
7.45.2.2	txend2_cb	458
7.45.2.3	rxend_cb	458
7.45.2.4	rxchar_cb	458
7.45.2.5	rxerr_cb	458
7.45.2.6	speed	458
7.45.2.7	cr1	458
7.45.2.8	cr2	458
7.45.2.9	cr3	458

7.46 UARTDriver Struct Reference . . . . .	459
7.46.1 Detailed Description . . . . .	459
7.46.2 Field Documentation . . . . .	461
7.46.2.1 state . . . . .	461
7.46.2.2 txstate . . . . .	461
7.46.2.3 rxstate . . . . .	461
7.46.2.4 config . . . . .	461
7.46.2.5 usart . . . . .	461
7.46.2.6 dmamode . . . . .	461
7.46.2.7 dmarx . . . . .	461
7.46.2.8 dmatx . . . . .	461
7.46.2.9 rdbuf . . . . .	462
7.47 USBConfig Struct Reference . . . . .	462
7.47.1 Detailed Description . . . . .	462
7.47.2 Field Documentation . . . . .	464
7.47.2.1 event_cb . . . . .	464
7.47.2.2 get_descriptor_cb . . . . .	464
7.47.2.3 requests_hook_cb . . . . .	464
7.47.2.4 sof_cb . . . . .	464
7.48 USBDescriptor Struct Reference . . . . .	464
7.48.1 Detailed Description . . . . .	464
7.48.2 Field Documentation . . . . .	464
7.48.2.1 ud_size . . . . .	464
7.48.2.2 ud_string . . . . .	465
7.49 USBDriver Struct Reference . . . . .	465
7.49.1 Detailed Description . . . . .	465
7.49.2 Field Documentation . . . . .	466
7.49.2.1 state . . . . .	466
7.49.2.2 config . . . . .	466
7.49.2.3 param . . . . .	466
7.49.2.4 transmitting . . . . .	466
7.49.2.5 receiving . . . . .	466
7.49.2.6 epc . . . . .	466
7.49.2.7 ep0state . . . . .	466
7.49.2.8 ep0next . . . . .	467
7.49.2.9 ep0n . . . . .	467
7.49.2.10 ep0endcb . . . . .	467
7.49.2.11 setup . . . . .	467
7.49.2.12 status . . . . .	467
7.49.2.13 address . . . . .	467

7.49.2.14 configuration . . . . .	467
7.49.2.15 pmnext . . . . .	467
7.50 USBEndpointConfig Struct Reference . . . . .	467
7.50.1 Detailed Description . . . . .	467
7.50.2 Field Documentation . . . . .	469
7.50.2.1 ep_mode . . . . .	469
7.50.2.2 setup_cb . . . . .	469
7.50.2.3 in_cb . . . . .	469
7.50.2.4 out_cb . . . . .	469
7.50.2.5 in_maxsize . . . . .	469
7.50.2.6 out_maxsize . . . . .	470
7.50.2.7 in_state . . . . .	470
7.50.2.8 out_state . . . . .	470
7.51 USBIInEndpointState Struct Reference . . . . .	470
7.51.1 Detailed Description . . . . .	470
7.51.2 Field Documentation . . . . .	470
7.51.2.1 txbuf . . . . .	470
7.51.2.2 txsize . . . . .	470
7.51.2.3 txcnt . . . . .	470
7.52 USBOutEndpointState Struct Reference . . . . .	471
7.52.1 Detailed Description . . . . .	471
7.52.2 Field Documentation . . . . .	471
7.52.2.1 rxpkts . . . . .	471
7.52.2.2 rxbuf . . . . .	471
7.52.2.3 rxsize . . . . .	471
7.52.2.4 rxcnt . . . . .	471
<b>8 File Documentation</b> . . . . .	<b>472</b>
8.1 adc.c File Reference . . . . .	472
8.1.1 Detailed Description . . . . .	472
8.2 adc.h File Reference . . . . .	473
8.2.1 Detailed Description . . . . .	473
8.3 adc_lld.c File Reference . . . . .	474
8.3.1 Detailed Description . . . . .	474
8.4 adc_lld.h File Reference . . . . .	474
8.4.1 Detailed Description . . . . .	474
8.5 can.c File Reference . . . . .	478
8.5.1 Detailed Description . . . . .	478
8.6 can.h File Reference . . . . .	478
8.6.1 Detailed Description . . . . .	478

8.7	can_lld.c File Reference . . . . .	479
8.7.1	Detailed Description . . . . .	479
8.8	can_lld.h File Reference . . . . .	480
8.8.1	Detailed Description . . . . .	480
8.9	ext.c File Reference . . . . .	481
8.9.1	Detailed Description . . . . .	481
8.10	ext_lld.c File Reference . . . . .	482
8.10.1	Detailed Description . . . . .	482
8.11	ext_lld.h File Reference . . . . .	483
8.11.1	Detailed Description . . . . .	483
8.12	gpt.c File Reference . . . . .	485
8.12.1	Detailed Description . . . . .	485
8.13	gpt.h File Reference . . . . .	485
8.13.1	Detailed Description . . . . .	485
8.14	gpt_lld.c File Reference . . . . .	486
8.14.1	Detailed Description . . . . .	486
8.15	gpt_lld.h File Reference . . . . .	487
8.15.1	Detailed Description . . . . .	487
8.16	hal.c File Reference . . . . .	488
8.16.1	Detailed Description . . . . .	488
8.17	hal.h File Reference . . . . .	488
8.17.1	Detailed Description . . . . .	488
8.18	hal_lld.c File Reference . . . . .	489
8.18.1	Detailed Description . . . . .	490
8.19	hal_lld.h File Reference . . . . .	490
8.19.1	Detailed Description . . . . .	490
8.20	hal_lld_f100.h File Reference . . . . .	491
8.20.1	Detailed Description . . . . .	491
8.21	hal_lld_f103.h File Reference . . . . .	498
8.21.1	Detailed Description . . . . .	498
8.22	hal_lld_f105_f107.h File Reference . . . . .	507
8.22.1	Detailed Description . . . . .	507
8.23	halconf.h File Reference . . . . .	513
8.23.1	Detailed Description . . . . .	513
8.24	i2c.c File Reference . . . . .	515
8.24.1	Detailed Description . . . . .	515
8.25	i2c.h File Reference . . . . .	516
8.25.1	Detailed Description . . . . .	516
8.26	i2c_lld.c File Reference . . . . .	517
8.26.1	Detailed Description . . . . .	517

8.27 i2c_lld.h File Reference . . . . .	518
8.27.1 Detailed Description . . . . .	518
8.28 icu.c File Reference . . . . .	520
8.28.1 Detailed Description . . . . .	520
8.29 icu.h File Reference . . . . .	520
8.29.1 Detailed Description . . . . .	520
8.30 icu_lld.c File Reference . . . . .	521
8.30.1 Detailed Description . . . . .	521
8.31 icu_lld.h File Reference . . . . .	522
8.31.1 Detailed Description . . . . .	522
8.32 mmc_spi.c File Reference . . . . .	523
8.32.1 Detailed Description . . . . .	523
8.33 mmc_spi.h File Reference . . . . .	524
8.33.1 Detailed Description . . . . .	524
8.34 pal.c File Reference . . . . .	525
8.34.1 Detailed Description . . . . .	525
8.35 pal.h File Reference . . . . .	526
8.35.1 Detailed Description . . . . .	526
8.36 pal_lld.c File Reference . . . . .	527
8.36.1 Detailed Description . . . . .	527
8.37 pal_lld.h File Reference . . . . .	528
8.37.1 Detailed Description . . . . .	528
8.38 pwm.c File Reference . . . . .	529
8.38.1 Detailed Description . . . . .	529
8.39 pwm.h File Reference . . . . .	530
8.39.1 Detailed Description . . . . .	530
8.40 pwm_lld.c File Reference . . . . .	531
8.40.1 Detailed Description . . . . .	531
8.41 pwm_lld.h File Reference . . . . .	532
8.41.1 Detailed Description . . . . .	532
8.42 rtc.c File Reference . . . . .	533
8.42.1 Detailed Description . . . . .	533
8.43 rtc.h File Reference . . . . .	534
8.43.1 Detailed Description . . . . .	534
8.44 rtc_lld.c File Reference . . . . .	534
8.44.1 Detailed Description . . . . .	534
8.45 rtc_lld.h File Reference . . . . .	535
8.45.1 Detailed Description . . . . .	535
8.46 sdc.c File Reference . . . . .	536
8.46.1 Detailed Description . . . . .	536

8.47 sdc.h File Reference . . . . .	537
8.47.1 Detailed Description . . . . .	537
8.48 sdc_lld.c File Reference . . . . .	539
8.48.1 Detailed Description . . . . .	539
8.49 sdc_lld.h File Reference . . . . .	539
8.49.1 Detailed Description . . . . .	539
8.50 serial.c File Reference . . . . .	541
8.50.1 Detailed Description . . . . .	541
8.51 serial.h File Reference . . . . .	541
8.51.1 Detailed Description . . . . .	541
8.52 serial_lld.c File Reference . . . . .	543
8.52.1 Detailed Description . . . . .	543
8.53 serial_lld.h File Reference . . . . .	544
8.53.1 Detailed Description . . . . .	544
8.54 serial_usb.c File Reference . . . . .	545
8.54.1 Detailed Description . . . . .	545
8.55 serial_usb.h File Reference . . . . .	546
8.55.1 Detailed Description . . . . .	546
8.56 spi.c File Reference . . . . .	547
8.56.1 Detailed Description . . . . .	547
8.57 spi.h File Reference . . . . .	548
8.57.1 Detailed Description . . . . .	548
8.58 spi_lld.c File Reference . . . . .	549
8.58.1 Detailed Description . . . . .	549
8.59 spi_lld.h File Reference . . . . .	550
8.59.1 Detailed Description . . . . .	550
8.60 stm32.h File Reference . . . . .	551
8.60.1 Detailed Description . . . . .	551
8.61 stm32_dma.c File Reference . . . . .	552
8.61.1 Detailed Description . . . . .	552
8.62 stm32_dma.h File Reference . . . . .	553
8.62.1 Detailed Description . . . . .	553
8.63 stm32_rcc.h File Reference . . . . .	556
8.63.1 Detailed Description . . . . .	556
8.64 stm32_usb.h File Reference . . . . .	559
8.64.1 Detailed Description . . . . .	559
8.65 tm.c File Reference . . . . .	560
8.65.1 Detailed Description . . . . .	560
8.66 tm.h File Reference . . . . .	560
8.66.1 Detailed Description . . . . .	560

8.67 uart.c File Reference . . . . .	561
8.67.1 Detailed Description . . . . .	561
8.68 uart.h File Reference . . . . .	562
8.68.1 Detailed Description . . . . .	562
8.69 uart_lld.c File Reference . . . . .	563
8.69.1 Detailed Description . . . . .	563
8.70 uart_lld.h File Reference . . . . .	564
8.70.1 Detailed Description . . . . .	564
8.71 usb.c File Reference . . . . .	565
8.71.1 Detailed Description . . . . .	565
8.72 usb.h File Reference . . . . .	566
8.72.1 Detailed Description . . . . .	566
8.73 usb_lld.c File Reference . . . . .	569
8.73.1 Detailed Description . . . . .	569
8.73.2 Variable Documentation . . . . .	570
8.73.2.1 in . . . . .	570
8.73.2.2 out . . . . .	570
8.74 usb_lld.h File Reference . . . . .	570
8.74.1 Detailed Description . . . . .	570

# **Chapter 1**

## **ChibiOS/RT**

### **1.1 Copyright**

Copyright (C) 2006..2011 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/RT portable HAL API and the implemented STM32F1xx drivers.

### **1.3 Related Documents**

- ChibiOS/RT General Architecture
- ChibiOS/RT Cortex-Mx/GCC Kernel Reference Manual
- ChibiOS/RT Cortex-Mx/IAR Kernel Reference Manual
- ChibiOS/RT Cortex-Mx/RVCT Kernel Reference Manual

## Chapter 2

### Deprecated List

Global `sdGetWouldBlock(sdp)`

Global `sdPutWouldBlock(sdp)`

# Chapter 3

## Module Index

### 3.1 Modules

Here is a list of all modules:

HAL . . . . .	9
Configuration . . . . .	11
ADC Driver . . . . .	17
CAN Driver . . . . .	40
EXT Driver . . . . .	53
GPT Driver . . . . .	64
HAL Driver . . . . .	77
STM32F100 HAL Support . . . . .	287
STM32F103 HAL Support . . . . .	305
STM32F105/F107 HAL Support . . . . .	329
I2C Driver . . . . .	87
ICU Driver . . . . .	104
MMC over SPI Driver . . . . .	117
PAL Driver . . . . .	127
PWM Driver . . . . .	145
RTC Driver . . . . .	162
SDC Driver . . . . .	170
Serial Driver . . . . .	187
Serial over USB Driver . . . . .	203
SPI Driver . . . . .	208
Time Measurement Driver. . . . .	230
UART Driver . . . . .	232
USB Driver . . . . .	251
STM32F1xx Drivers . . . . .	350
STM32F1xx Initialization Support . . . . .	351
STM32F1xx ADC Support . . . . .	352
STM32F1xx CAN Support . . . . .	352
STM32F1xx EXT Support . . . . .	352
STM32F1xx GPT Support . . . . .	353
STM32F1xx I2C Support . . . . .	353
STM32F1xx ICU Support . . . . .	353
STM32F1xx MAC Support . . . . .	354
STM32F1xx PAL Support . . . . .	354
STM32F1xx PWM Support . . . . .	355
STM32F1xx RTC Support . . . . .	356
STM32F1xx SDC Support . . . . .	356
STM32F1xx Serial Support . . . . .	356
STM32F1xx SPI Support . . . . .	357

STM32F1xx UART Support . . . . .	357
STM32F1xx USB Support . . . . .	358
STM32F1xx Platform Drivers . . . . .	358
STM32F1xx DMA Support . . . . .	358
STM32F1xx RCC Support . . . . .	373

## Chapter 4

# Data Structure Index

### 4.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">ADCConfig</a> (Driver configuration structure ) . . . . .	399
<a href="#">ADCConversionGroup</a> (Conversion group configuration structure ) . . . . .	399
<a href="#">ADCDriver</a> (Structure representing an ADC driver ) . . . . .	402
<a href="#">CANConfig</a> (Driver configuration structure ) . . . . .	405
<a href="#">CANDriver</a> (Structure representing an CAN driver ) . . . . .	406
<a href="#">CANFilter</a> (CAN filter ) . . . . .	409
<a href="#">CANRxFrame</a> (CAN received frame ) . . . . .	410
<a href="#">CANTxFrame</a> (CAN transmission frame ) . . . . .	411
<a href="#">EXTChannelConfig</a> (Channel configuration structure ) . . . . .	412
<a href="#">EXTConfig</a> (Driver configuration structure ) . . . . .	414
<a href="#">EXTDriver</a> (Structure representing an EXT driver ) . . . . .	415
<a href="#">GPTConfig</a> (Driver configuration structure ) . . . . .	416
<a href="#">GPTDriver</a> (Structure representing a GPT driver ) . . . . .	418
<a href="#">I2CConfig</a> (Driver configuration structure ) . . . . .	420
<a href="#">I2CDriver</a> (Structure representing an I2C driver ) . . . . .	421
<a href="#">ICUConfig</a> (Driver configuration structure ) . . . . .	423
<a href="#">ICUDriver</a> (Structure representing an ICU driver ) . . . . .	425
<a href="#">IOBus</a> (I/O bus descriptor ) . . . . .	427
<a href="#">MMCConfig</a> (Driver configuration structure ) . . . . .	428
<a href="#">MMCDriver</a> (Structure representing a MMC driver ) . . . . .	428
<a href="#">PALConfig</a> (STM32 GPIO static initializer ) . . . . .	431
<a href="#">PWMChannelConfig</a> (PWM driver channel configuration structure ) . . . . .	433
<a href="#">PWMConfig</a> (PWM driver configuration structure ) . . . . .	435
<a href="#">PWMDriver</a> (Structure representing a PWM driver ) . . . . .	438
<a href="#">RTCAlarm</a> (Structure representing an RTC alarm time stamp ) . . . . .	440
<a href="#">RTCCallbackConfig</a> (Structure representing an RTC callbacks config ) . . . . .	441
<a href="#">RTCDriver</a> (Structure representing an RTC driver ) . . . . .	441
<a href="#">RTCTime</a> (Structure representing an RTC time stamp ) . . . . .	442
<a href="#">SDCConfig</a> (Driver configuration structure ) . . . . .	443
<a href="#">SDCDriver</a> (Structure representing an SDC driver ) . . . . .	443
<a href="#">SerialConfig</a> (STM32 Serial Driver configuration structure ) . . . . .	444
<a href="#">SerialDriver</a> (Full duplex serial driver class ) . . . . .	445
<a href="#">SerialDriverVMT</a> ( <a href="#">SerialDriver</a> virtual methods table ) . . . . .	446
<a href="#">SerialUSBConfig</a> (Serial over USB Driver configuration structure ) . . . . .	446
<a href="#">SerialUSBDriver</a> (Full duplex serial driver class ) . . . . .	448
<a href="#">SerialUSBDriverVMT</a> ( <a href="#">SerialDriver</a> virtual methods table ) . . . . .	449
<a href="#">SPIConfig</a> (Driver configuration structure ) . . . . .	449
<a href="#">SPIDriver</a> (Structure representing a SPI driver ) . . . . .	450

<a href="#">stm32_dma_stream_t</a> (STM32 DMA stream descriptor structure ) . . . . .	453
<a href="#">stm32_gpio_setup_t</a> (GPIO port setup info ) . . . . .	453
<a href="#">stm32_tim_t</a> (STM32 TIM registers block ) . . . . .	454
<a href="#">stm32_usb_descriptor_t</a> (USB descriptor registers block ) . . . . .	454
<a href="#">stm32_usb_t</a> (USB registers block ) . . . . .	455
<a href="#">TimeMeasurement</a> (Time Measurement structure ) . . . . .	456
<a href="#">UARTConfig</a> (Driver configuration structure ) . . . . .	456
<a href="#">UARTDriver</a> (Structure representing an UART driver ) . . . . .	459
<a href="#">USBCConfig</a> (Type of an USB driver configuration structure ) . . . . .	462
<a href="#">USBDescriptor</a> (Type of an USB descriptor ) . . . . .	464
<a href="#">USBDriver</a> (Structure representing an USB driver ) . . . . .	465
<a href="#">USBEndpointConfig</a> (Type of an USB endpoint configuration structure ) . . . . .	467
<a href="#">USBInEndpointState</a> (Type of an endpoint state structure ) . . . . .	470
<a href="#">USBOutEndpointState</a> (Type of an endpoint state structure ) . . . . .	471

# Chapter 5

## File Index

### 5.1 File List

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

<code>adc.c</code> (ADC Driver code ) . . . . .	472
<code>adc.h</code> (ADC Driver macros and structures ) . . . . .	473
<code>adc_lld.c</code> (STM32F1xx ADC subsystem low level driver source ) . . . . .	474
<code>adc_lld.h</code> (STM32F1xx ADC subsystem low level driver header ) . . . . .	474
<code>can.c</code> (CAN Driver code ) . . . . .	478
<code>can.h</code> (CAN Driver macros and structures ) . . . . .	478
<code>can_lld.c</code> (STM32 CAN subsystem low level driver source ) . . . . .	479
<code>can_lld.h</code> (STM32 CAN subsystem low level driver header ) . . . . .	480
<code>ext.c</code> (EXT Driver code ) . . . . .	481
<code>ext_lld.c</code> (STM32 EXT subsystem low level driver source ) . . . . .	482
<code>ext_lld.h</code> (STM32 EXT subsystem low level driver header ) . . . . .	483
<code>gpt.c</code> (GPT Driver code ) . . . . .	485
<code>gpt.h</code> (GPT Driver macros and structures ) . . . . .	485
<code>gpt_lld.c</code> (STM32 GPT subsystem low level driver source ) . . . . .	486
<code>gpt_lld.h</code> (STM32 GPT subsystem low level driver header ) . . . . .	487
<code>hal.c</code> (HAL subsystem code ) . . . . .	488
<code>hal.h</code> (HAL subsystem header ) . . . . .	488
<code>hal_lld.c</code> (STM32F1xx HAL subsystem low level driver source ) . . . . .	489
<code>hal_lld.h</code> (STM32F1xx HAL subsystem low level driver header ) . . . . .	490
<code>hal_lld_f100.h</code> (STM32F100 Value Line HAL subsystem low level driver header ) . . . . .	491
<code>hal_lld_f103.h</code> (STM32F103 Performance Line HAL subsystem low level driver header ) . . . . .	498
<code>hal_lld_f105_f107.h</code> (STM32F10x Connectivity Line HAL subsystem low level driver header ) . . . . .	507
<code>halconf.h</code> (HAL configuration header ) . . . . .	513
<code>i2c.c</code> (I2C Driver code ) . . . . .	515
<code>i2c.h</code> (I2C Driver macros and structures ) . . . . .	516
<code>i2c_lld.c</code> (STM32 I2C subsystem low level driver source ) . . . . .	517
<code>i2c_lld.h</code> (STM32 I2C subsystem low level driver header ) . . . . .	518
<code>icu.c</code> (ICU Driver code ) . . . . .	520
<code>icu.h</code> (ICU Driver macros and structures ) . . . . .	520
<code>icu_lld.c</code> (STM32 ICU subsystem low level driver header ) . . . . .	521
<code>icu_lld.h</code> (STM32 ICU subsystem low level driver header ) . . . . .	522
<code>mmc_spi.c</code> (MMC over SPI driver code ) . . . . .	523
<code>mmc_spi.h</code> (MMC over SPI driver header ) . . . . .	524
<code>pal.c</code> (I/O Ports Abstraction Layer code ) . . . . .	525
<code>pal.h</code> (I/O Ports Abstraction Layer macros, types and structures ) . . . . .	526
<code>pal_lld.c</code> (STM32F1xx GPIO low level driver code ) . . . . .	527
<code>pal_lld.h</code> (STM32F1xx GPIO low level driver header ) . . . . .	528
<code>pwm.c</code> (PWM Driver code ) . . . . .	529

<a href="#">pwm.h</a> (PWM Driver macros and structures ) . . . . .	530
<a href="#">pwm_lld.c</a> (STM32 PWM subsystem low level driver header ) . . . . .	531
<a href="#">pwm_lld.h</a> (STM32 PWM subsystem low level driver header ) . . . . .	532
<a href="#">rtc.c</a> (RTC Driver code ) . . . . .	533
<a href="#">rtc.h</a> (RTC Driver macros and structures ) . . . . .	534
<a href="#">rtc_lld.c</a> (STM32 RTC subsystem low level driver header ) . . . . .	534
<a href="#">rtc_lld.h</a> (STM32F1xx RTC subsystem low level driver header ) . . . . .	535
<a href="#">sdc.c</a> (SDC Driver code ) . . . . .	536
<a href="#">sdc.h</a> (SDC Driver macros and structures ) . . . . .	537
<a href="#">sdc_lld.c</a> (STM32 SDC subsystem low level driver source ) . . . . .	539
<a href="#">sdc_lld.h</a> (STM32 SDC subsystem low level driver header ) . . . . .	539
<a href="#">serial.c</a> (Serial Driver code ) . . . . .	541
<a href="#">serial.h</a> (Serial Driver macros and structures ) . . . . .	541
<a href="#">serial_lld.c</a> (STM32 low level serial driver code ) . . . . .	543
<a href="#">serial_lld.h</a> (STM32 low level serial driver header ) . . . . .	544
<a href="#">serial_usb.c</a> (Serial over USB Driver code ) . . . . .	545
<a href="#">serial_usb.h</a> (Serial over USB Driver macros and structures ) . . . . .	546
<a href="#">spi.c</a> (SPI Driver code ) . . . . .	547
<a href="#">spi.h</a> (SPI Driver macros and structures ) . . . . .	548
<a href="#">spi_lld.c</a> (STM32 SPI subsystem low level driver source ) . . . . .	549
<a href="#">spi_lld.h</a> (STM32 SPI subsystem low level driver header ) . . . . .	550
<a href="#">stm32.h</a> (STM32 common header ) . . . . .	551
<a href="#">stm32_dma.c</a> (DMA helper driver code ) . . . . .	552
<a href="#">stm32_dma.h</a> (DMA helper driver header ) . . . . .	553
<a href="#">stm32_rcc.h</a> (RCC helper driver header ) . . . . .	556
<a href="#">stm32_usb.h</a> (STM32 USB registers layout header ) . . . . .	559
<a href="#">tm.c</a> (Time Measurement driver code ) . . . . .	560
<a href="#">tm.h</a> (Time Measurement driver header ) . . . . .	560
<a href="#">uart.c</a> (UART Driver code ) . . . . .	561
<a href="#">uart.h</a> (UART Driver macros and structures ) . . . . .	562
<a href="#">uart_lld.c</a> (STM32 low level UART driver code ) . . . . .	563
<a href="#">uart_lld.h</a> (STM32 low level UART driver header ) . . . . .	564
<a href="#">usb.c</a> (USB Driver code ) . . . . .	565
<a href="#">usb.h</a> (USB Driver macros and structures ) . . . . .	566
<a href="#">usb_lld.c</a> (STM32 USB subsystem low level driver source ) . . . . .	569
<a href="#">usb_lld.h</a> (STM32 USB subsystem low level driver header ) . . . . .	570

# Chapter 6

## Module Documentation

### 6.1 HAL

#### 6.1.1 Detailed Description

Hardware Abstraction Layer. Under ChibiOS/RT the set of the various device driver interfaces is called the HAL subsystem: Hardware Abstraction Layer. The HAL is the abstract interface between ChibiOS/RT application and hardware.

#### 6.1.2 HAL Device Drivers Architecture

A device driver is usually 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`.

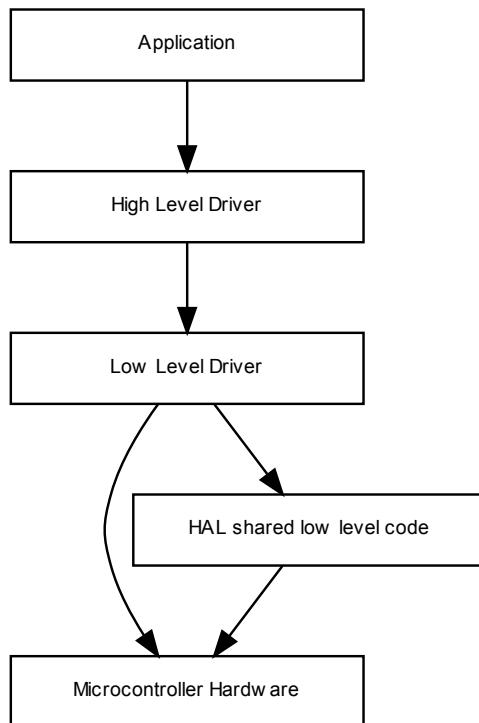
- Low Level Device Driver (**LLD**). This layer contains the platform dependent part of the driver.

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.

The LLD may be not present in those drivers that do not access the hardware directly but through other device drivers, as example the `MMC_SPI` driver uses the `SPI` and `PAL` drivers in order to implement its functionalities.

## 6.1.2.1 Diagram



## Modules

- Configuration
  - HAL Driver Configuration.*
- ADC Driver
  - Generic ADC Driver.*
- CAN Driver
  - Generic CAN Driver.*
- EXT Driver
  - Generic EXT Driver.*
- GPT Driver
  - Generic GPT Driver.*
- HAL Driver
  - Hardware Abstraction Layer.*
- I2C Driver
  - Generic I2C Driver.*
- ICU Driver
  - Generic ICU Driver.*
- MMC over SPI Driver
  - Generic MMC driver.*
- PAL Driver
  - I/O Ports Abstraction Layer.*
- PWM Driver

- **Generic PWM Driver.**
- **RTC Driver**
  - Real Time Clock Abstraction Layer.*
- **SDC Driver**
  - Generic SD Card Driver.*
- **Serial Driver**
  - Generic Serial Driver.*
- **Serial over USB Driver**
  - Serial over USB Driver.*
- **SPI Driver**
  - Generic SPI Driver.*
- **Time Measurement Driver.**
  - Time Measurement unit.*
- **UART Driver**
  - Generic UART Driver.*
- **USB Driver**
  - Generic USB Driver.*

## 6.2 Configuration

### 6.2.1 Detailed Description

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

#### Drivers enable switches

- `#define HAL_USE_TM TRUE`
  - Enables the TM subsystem.*
- `#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_EXT FALSE`
  - Enables the EXT subsystem.*
- `#define HAL_USE_GPT FALSE`
  - Enables the GPT subsystem.*
- `#define HAL_USE_I2C FALSE`
  - Enables the I2C subsystem.*
- `#define HAL_USE_ICU FALSE`
  - Enables the ICU subsystem.*
- `#define HAL_USE_MAC TRUE`
  - Enables the MAC subsystem.*
- `#define HAL_USE_MMC_SPI TRUE`
  - Enables the MMC\_SPI subsystem.*
- `#define HAL_USE_PWM TRUE`
  - Enables the PWM subsystem.*
- `#define HAL_USE_RTC FALSE`

- `#define HAL_USE_SDC FALSE`  
*Enables the SDC subsystem.*
- `#define HAL_USE_SERIAL TRUE`  
*Enables the SERIAL subsystem.*
- `#define HAL_USE_SERIAL_USB TRUE`  
*Enables the SERIAL over USB subsystem.*
- `#define HAL_USE_SPI TRUE`  
*Enables the SPI subsystem.*
- `#define HAL_USE_UART TRUE`  
*Enables the UART subsystem.*
- `#define HAL_USE_USB TRUE`  
*Enables the USB subsystem.*

### ADC driver related setting

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

### CAN driver related setting

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

### I2C driver related setting

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

### MAC driver related setting

- `#define MAC_USE_EVENTS TRUE`  
*Enables an event sources for incoming packets.*

### MMC\_SPI driver related setting

- `#define MMC_SECTOR_SIZE 512`  
*Block size for MMC transfers.*
- `#define MMC_NICE_WAITING TRUE`  
*Delays insertions.*
- `#define MMC_POLLING_INTERVAL 10`  
*Number of positive insertion queries before generating the insertion event.*
- `#define MMC_POLLING_DELAY 10`  
*Interval, in milliseconds, between insertion queries.*
- `#define MMC_USE_SPI_POLLING TRUE`  
*Uses the SPI polled API for small data transfers.*

## SDC driver related setting

- `#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.*

## 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 64`  
*Serial over USB buffers size.*

## SPI driver related setting

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

### 6.2.2 Define Documentation

#### 6.2.2.1 `#define HAL_USE_PAL TRUE`

Enables the PAL subsystem.

#### 6.2.2.2 `#define HAL_USE_ADC TRUE`

Enables the ADC subsystem.

#### 6.2.2.3 `#define HAL_USE_CAN TRUE`

Enables the CAN subsystem.

#### 6.2.2.4 `#define HAL_USE_EXT FALSE`

Enables the EXT subsystem.

#### 6.2.2.5 `#define HAL_USE_GPT FALSE`

Enables the GPT subsystem.

6.2.2.6 #define HAL\_USE\_I2C FALSE

Enables the I2C subsystem.

6.2.2.7 #define HAL\_USE\_ICU FALSE

Enables the ICU subsystem.

6.2.2.8 #define HAL\_USE\_MAC TRUE

Enables the MAC subsystem.

6.2.2.9 #define HAL\_USE\_MMC\_SPI TRUE

Enables the MMC\_SPI subsystem.

6.2.2.10 #define HAL\_USE\_PWM TRUE

Enables the PWM subsystem.

6.2.2.11 #define HAL\_USE\_RTC FALSE

Enables the RTC subsystem.

6.2.2.12 #define HAL\_USE\_SDC FALSE

Enables the SDC subsystem.

6.2.2.13 #define HAL\_USE\_SERIAL TRUE

Enables the SERIAL subsystem.

6.2.2.14 #define HAL\_USE\_SERIAL\_USB TRUE

Enables the SERIAL over USB subsystem.

6.2.2.15 #define HAL\_USE\_SPI TRUE

Enables the SPI subsystem.

6.2.2.16 #define HAL\_USE\_UART TRUE

Enables the UART subsystem.

6.2.2.17 #define HAL\_USE\_USB TRUE

Enables the USB subsystem.

**6.2.2.18 #define ADC\_USE\_WAIT TRUE**

Enables synchronous APIs.

**Note**

Disabling this option saves both code and data space.

**6.2.2.19 #define ADC\_USE\_MUTUAL\_EXCLUSION TRUE**

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

**Note**

Disabling this option saves both code and data space.

**6.2.2.20 #define CAN\_USE\_SLEEP\_MODE TRUE**

Sleep mode related APIs inclusion switch.

**6.2.2.21 #define I2C\_USE\_MUTUAL\_EXCLUSION TRUE**

Enables the mutual exclusion APIs on the I2C bus.

**6.2.2.22 #define MAC\_USE\_EVENTS TRUE**

Enables an event sources for incoming packets.

**6.2.2.23 #define MMC\_SECTOR\_SIZE 512**

Block size for MMC transfers.

**6.2.2.24 #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.

**6.2.2.25 #define MMC\_POLLING\_INTERVAL 10**

Number of positive insertion queries before generating the insertion event.

**6.2.2.26 #define MMC\_POLLING\_DELAY 10**

Interval, in milliseconds, between insertion queries.

**6.2.2.27 #define MMC\_USE\_SPI\_POLLING TRUE**

Uses the SPI polled API for small data transfers.

Polled transfers usually improve performance because it saves two context switches and interrupt servicing. Note that this option has no effect on large transfers which are always performed using DMAs/IRQs.

**6.2.2.28 #define SDC\_INIT\_RETRY 100**

Number of initialization attempts before rejecting the card.

**Note**

Attempts are performed at 10mS intervals.

**6.2.2.29 #define SDC\_MMIC\_SUPPORT FALSE**

Include support for MMC cards.

**Note**

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

**6.2.2.30 #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.

**6.2.2.31 #define SERIAL\_DEFAULT\_BITRATE 38400**

Default bit rate.

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

**6.2.2.32 #define SERIAL\_BUFFERS\_SIZE 16**

Serial buffers size.

Configuration parameter, you can change the depth of the queue buffers depending on the requirements of your application.

**Note**

The default is 64 bytes for both the transmission and receive buffers.

**6.2.2.33 #define SERIAL\_USB\_BUFFERS\_SIZE 64**

Serial over USB buffers size.

Configuration parameter, the buffer size must be a multiple of the USB data endpoint maximum packet size.

**Note**

The default is 64 bytes for both the transmission and receive buffers.

**6.2.2.34 #define SPI\_USE\_WAIT TRUE**

Enables synchronous APIs.

**Note**

Disabling this option saves both code and data space.

### 6.2.2.35 #define SPI\_USE\_MUTUAL\_EXCLUSION TRUE

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

#### Note

Disabling this option saves both code and data space.

## 6.3 ADC Driver

### 6.3.1 Detailed Description

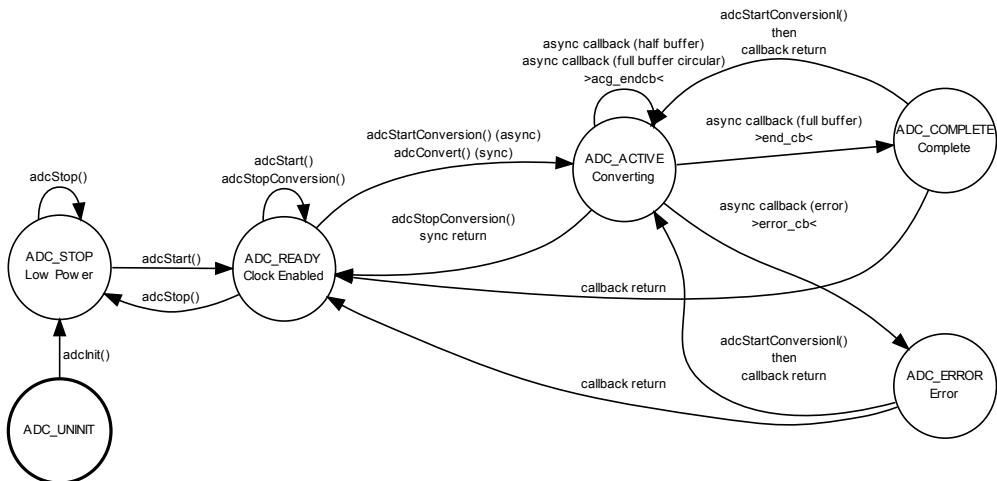
Generic ADC Driver. This module implements a generic ADC (Analog to Digital Converter) driver supporting a variety of buffer and conversion modes.

#### Precondition

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

### 6.3.2 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).



### 6.3.3 ADC Operations

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

#### 6.3.3.1 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.

### 6.3.3.2 ADC Conversion Modes

The driver supports several conversion modes:

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

### 6.3.3.3 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 linear and circular modes 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.

## Data Structures

- struct `ADCConversionGroup`  
*Conversion group configuration structure.*
- struct `ADCCConfig`  
*Driver configuration structure.*
- struct `ADCDriver`  
*Structure representing an ADC driver.*

## 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 `ADCConversionGroup` \*grpp, `adcsample_t` \*samples, `size_t` depth)

- `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.*
- `void adcAcquireBus (ADCDriver *adcp)`

*Gains exclusive access to the ADC peripheral.*
- `void adcReleaseBus (ADCDriver *adcp)`

*Releases exclusive access to the ADC peripheral.*
- `msg_t adcConvert (ADCDriver *adcp, const ADCConversionGroup *grpp, adcsample_t *samples, size_t depth)`

*Performs an ADC conversion.*
- `void adc_lld_init (void)`

*Low level ADC driver initialization.*
- `void adc_lld_start (ADCDriver *adcp)`

*Configures and activates the ADC peripheral.*
- `void adc_lld_stop (ADCDriver *adcp)`

*Deactivates the ADC peripheral.*
- `void adc_lld_start_conversion (ADCDriver *adcp)`

*Starts an ADC conversion.*
- `void adc_lld_stop_conversion (ADCDriver *adcp)`

*Stops an ongoing conversion.*

## Variables

- `ADCDriver ADCD1`

*ADC1 driver identifier.*

## 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)`

*Resumes a thread waiting for a conversion completion.*
- `#define _adc_reset_s(adcp)`

*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.*

### Triggers selection

- `#define ADC_CR2_EXTSEL_SRC(n) ((n) << 17)`  
*Trigger source.*
- `#define ADC_CR2_EXTSEL_SWSTART (7 << 17)`  
*Software trigger.*

### Available analog channels

- `#define ADC_CHANNEL_IN0 0`  
*External analog input 0.*
- `#define ADC_CHANNEL_IN1 1`  
*External analog input 1.*
- `#define ADC_CHANNEL_IN2 2`  
*External analog input 2.*
- `#define ADC_CHANNEL_IN3 3`  
*External analog input 3.*
- `#define ADC_CHANNEL_IN4 4`  
*External analog input 4.*
- `#define ADC_CHANNEL_IN5 5`  
*External analog input 5.*
- `#define ADC_CHANNEL_IN6 6`  
*External analog input 6.*
- `#define ADC_CHANNEL_IN7 7`  
*External analog input 7.*
- `#define ADC_CHANNEL_IN8 8`  
*External analog input 8.*
- `#define ADC_CHANNEL_IN9 9`  
*External analog input 9.*
- `#define ADC_CHANNEL_IN10 10`  
*External analog input 10.*
- `#define ADC_CHANNEL_IN11 11`  
*External analog input 11.*
- `#define ADC_CHANNEL_IN12 12`  
*External analog input 12.*
- `#define ADC_CHANNEL_IN13 13`  
*External analog input 13.*
- `#define ADC_CHANNEL_IN14 14`  
*External analog input 14.*
- `#define ADC_CHANNEL_IN15 15`  
*External analog input 15.*
- `#define ADC_CHANNEL_SENSOR 16`  
*Internal temperature sensor.*
- `#define ADC_CHANNEL_VREFINT 17`  
*Internal reference.*

## Sampling rates

- #define **ADC\_SAMPLE\_1P5** 0  
    *1.5 cycles sampling time.*
- #define **ADC\_SAMPLE\_7P5** 1  
    *7.5 cycles sampling time.*
- #define **ADC\_SAMPLE\_13P5** 2  
    *13.5 cycles sampling time.*
- #define **ADC\_SAMPLE\_28P5** 3  
    *28.5 cycles sampling time.*
- #define **ADC\_SAMPLE\_41P5** 4  
    *41.5 cycles sampling time.*
- #define **ADC\_SAMPLE\_55P5** 5  
    *55.5 cycles sampling time.*
- #define **ADC\_SAMPLE\_71P5** 6  
    *71.5 cycles sampling time.*
- #define **ADC\_SAMPLE\_239P5** 7  
    *239.5 cycles sampling time.*

## Configuration options

- #define **STM32\_ADC\_USE\_ADC1** TRUE  
    *ADC1 driver enable switch.*
- #define **STM32\_ADC\_ADC1\_DMA\_PRIORITY** 2  
    *ADC1 DMA priority (0..3|lowest..highest).*
- #define **STM32\_ADC\_ADC1\_IRQ\_PRIORITY** 5  
    *ADC1 interrupt priority level setting.*

## Sequences building helper macros

- #define **ADC\_SQR1\_NUM\_CH**(n) (((n) - 1) << 20)  
    *Number of channels in a conversion sequence.*
- #define **ADC\_SQR3\_SQ1\_N**(n) ((n) << 0)  
    *1st channel in seq.*
- #define **ADC\_SQR3\_SQ2\_N**(n) ((n) << 5)  
    *2nd channel in seq.*
- #define **ADC\_SQR3\_SQ3\_N**(n) ((n) << 10)  
    *3rd channel in seq.*
- #define **ADC\_SQR3\_SQ4\_N**(n) ((n) << 15)  
    *4th channel in seq.*
- #define **ADC\_SQR3\_SQ5\_N**(n) ((n) << 20)  
    *5th channel in seq.*
- #define **ADC\_SQR3\_SQ6\_N**(n) ((n) << 25)  
    *6th channel in seq.*
- #define **ADC\_SQR2\_SQ7\_N**(n) ((n) << 0)  
    *7th channel in seq.*
- #define **ADC\_SQR2\_SQ8\_N**(n) ((n) << 5)  
    *8th channel in seq.*
- #define **ADC\_SQR2\_SQ9\_N**(n) ((n) << 10)  
    *9th channel in seq.*
- #define **ADC\_SQR2\_SQ10\_N**(n) ((n) << 15)

- #define ADC\_SQR2\_SQ11\_N(n) ((n) << 20)
  - 10th channel in seq.*
- #define ADC\_SQR2\_SQ12\_N(n) ((n) << 25)
  - 11th channel in seq.*
- #define ADC\_SQR1\_SQ13\_N(n) ((n) << 0)
  - 12th channel in seq.*
- #define ADC\_SQR1\_SQ14\_N(n) ((n) << 5)
  - 13th channel in seq.*
- #define ADC\_SQR1\_SQ15\_N(n) ((n) << 10)
  - 14th channel in seq.*
- #define ADC\_SQR1\_SQ16\_N(n) ((n) << 15)
  - 15th channel in seq.*
- #define ADC\_SQR1\_SQ17\_N(n) ((n) << 16)
  - 16th channel in seq.*

### Sampling rate settings helper macros

- #define ADC\_SMPR2\_SMP\_AN0(n) ((n) << 0)
  - AN0 sampling time.*
- #define ADC\_SMPR2\_SMP\_AN1(n) ((n) << 3)
  - AN1 sampling time.*
- #define ADC\_SMPR2\_SMP\_AN2(n) ((n) << 6)
  - AN2 sampling time.*
- #define ADC\_SMPR2\_SMP\_AN3(n) ((n) << 9)
  - AN3 sampling time.*
- #define ADC\_SMPR2\_SMP\_AN4(n) ((n) << 12)
  - AN4 sampling time.*
- #define ADC\_SMPR2\_SMP\_AN5(n) ((n) << 15)
  - AN5 sampling time.*
- #define ADC\_SMPR2\_SMP\_AN6(n) ((n) << 18)
  - AN6 sampling time.*
- #define ADC\_SMPR2\_SMP\_AN7(n) ((n) << 21)
  - AN7 sampling time.*
- #define ADC\_SMPR2\_SMP\_AN8(n) ((n) << 24)
  - AN8 sampling time.*
- #define ADC\_SMPR2\_SMP\_AN9(n) ((n) << 27)
  - AN9 sampling time.*
- #define ADC\_SMPR1\_SMP\_AN10(n) ((n) << 0)
  - AN10 sampling time.*
- #define ADC\_SMPR1\_SMP\_AN11(n) ((n) << 3)
  - AN11 sampling time.*
- #define ADC\_SMPR1\_SMP\_AN12(n) ((n) << 6)
  - AN12 sampling time.*
- #define ADC\_SMPR1\_SMP\_AN13(n) ((n) << 9)
  - AN13 sampling time.*
- #define ADC\_SMPR1\_SMP\_AN14(n) ((n) << 12)
  - AN14 sampling time.*
- #define ADC\_SMPR1\_SMP\_AN15(n) ((n) << 15)
  - AN15 sampling time.*
- #define ADC\_SMPR1\_SMP\_SENSOR(n) ((n) << 18)
  - Temperature Sensor sampling time.*
- #define ADC\_SMPR1\_SMP\_VREF(n) ((n) << 21)
  - Voltage Reference sampling time.*

## Typedefs

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

## Enumerations

- `enum adcstate_t {`  
`ADC_UNINIT = 0, ADC_STOP = 1, ADC_READY = 2, ADC_ACTIVE = 3,`  
`ADC_COMPLETE = 4, ADC_ERROR = 5 }`  
*Driver state machine possible states.*
- `enum adcerror_t { ADC_ERR_DMAFAILURE = 0 }`  
*Possible ADC failure causes.*

### 6.3.4 Function Documentation

#### 6.3.4.1 void adclnit ( void )

ADC Driver initialization.

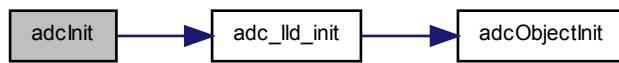
##### Note

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:



#### 6.3.4.2 void adcObjectInit ( ADCDriver \* adcp )

Initializes the standard part of a `ADCDriver` structure.

##### Parameters

out                    *adcp* pointer to the [ADCDriver](#) object

#### Function Class:

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

##### 6.3.4.3 void adcStart ( [ADCDriver](#) \* *adcp*, [ADCConfig](#) \* *config* )

Configures and activates the ADC peripheral.

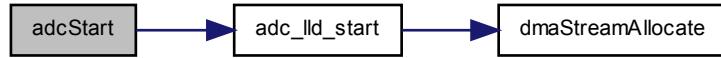
#### Parameters

in	<i>adcp</i> pointer to the <a href="#">ADCDriver</a> object
in	<i>config</i> pointer to the <a href="#">ADCConfig</a> 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:



##### 6.3.4.4 void adcStop ( [ADCDriver](#) \* *adcp* )

Deactivates the ADC peripheral.

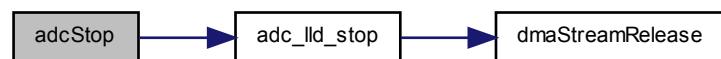
#### Parameters

in	<i>adcp</i> pointer to the <a href="#">ADCDriver</a> 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:



```
6.3.4.5 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 <a href="#">ADCDriver</a> object
in	<i>grpp</i>	pointer to a <a href="#">ADCConversionGroup</a> 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:



```
6.3.4.6 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 <a href="#">ADCDriver</a> object
in	<i>grpp</i>	pointer to a <a href="#">ADCConversionGroup</a> 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:



#### 6.3.4.7 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.

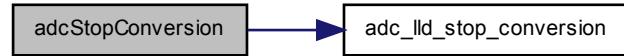
##### Parameters

in *adcp* pointer to the `ADCDriver` object

##### Function Class:

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

Here is the call graph for this function:



#### 6.3.4.8 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.

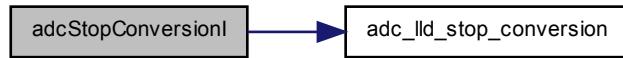
##### Parameters

in *adcp* pointer to the `ADCDriver` object

##### Function Class:

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

Here is the call graph for this function:



#### 6.3.4.9 void adcAcquireBus ( ADCDriver \* *adcp* )

Gains exclusive access to the ADC peripheral.

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

##### Precondition

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

##### Parameters

in *adcp* pointer to the `ADCDriver` object

##### Function Class:

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

#### 6.3.4.10 void adcReleaseBus ( ADCDriver \* *adcp* )

Releases exclusive access to the ADC peripheral.

##### Precondition

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

##### Parameters

in *adcp* pointer to the `ADCDriver` object

##### Function Class:

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

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

**Returns**

The operation result.

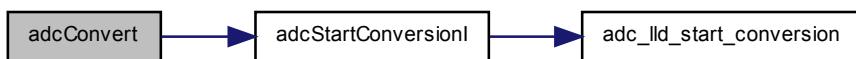
**Return values**

<i>RDY_OK</i>	Conversion finished.
<i>RDY_RESET</i>	The conversion has been stopped using <code>acdStopConversion()</code> or <code>acdStopConversionI()</code> , the result buffer may contain incorrect data.
<i>RDY_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:

**6.3.4.12 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:

**6.3.4.13 void adc\_lld\_start( ADCDriver \* adcp )**

Configures and activates the ADC peripheral.

**Parameters**

in *adcp* pointer to the [ADCDriver](#) object

**Function Class:**

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

Here is the call graph for this function:

**6.3.4.14 void adc\_lld\_stop ( ADCDriver \* *adcp* )**

Deactivates the ADC peripheral.

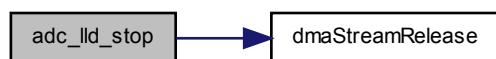
**Parameters**

in *adcp* pointer to the [ADCDriver](#) object

**Function Class:**

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

Here is the call graph for this function:

**6.3.4.15 void adc\_lld\_start\_conversion ( ADCDriver \* *adcp* )**

Starts an ADC conversion.

**Parameters**

in *adcp* pointer to the [ADCDriver](#) object

**Function Class:**

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

#### 6.3.4.16 void adc\_ll\_stop\_conversion ( ADCDriver \* *adcp* )

Stops an ongoing conversion.

##### Parameters

in *adcp* pointer to the [ADCDriver](#) object

##### Function Class:

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

## 6.3.5 Variable Documentation

### 6.3.5.1 ADCDriver ADCD1

ADC1 driver identifier.

## 6.3.6 Define Documentation

### 6.3.6.1 #define ADC\_USE\_WAIT TRUE

Enables synchronous APIs.

##### Note

Disabling this option saves both code and data space.

### 6.3.6.2 #define ADC\_USE\_MUTUAL\_EXCLUSION TRUE

Enables the [adcAcquireBus\(\)](#) and [adcReleaseBus\(\)](#) APIs.

##### Note

Disabling this option saves both code and data space.

### 6.3.6.3 #define \_adc\_reset\_i( *adcp* )

##### Value:

```
{
    if ((adcp)->thread != NULL) {
        Thread *tp = (adcp)->thread;
        (adcp)->thread = NULL;
        tp->p_u.rdymsg = RDY_RESET;
        chSchReadyI(tp);
    }
}
```

Resumes a thread waiting for a conversion completion.

##### Parameters

in *adcp* pointer to the [ADCDriver](#) object

##### Function Class:

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

6.3.6.4 #define \_adc\_reset\_s( *adcp* )**Value:**

```
{
    if ((adcp)->thread != NULL) {
        Thread *tp = (adcp)->thread;
        (adcp)->thread = NULL;
        chSchWakeupS(tp, RDY_RESET);
    }
}
```

Resumes a thread waiting for a conversion completion.

**Parameters**

in *adcp* pointer to the [ADCDriver](#) object

**Function Class:**

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

6.3.6.5 #define \_adc\_wakeup\_isr( *adcp* )**Value:**

```
{
    chSysLockFromIsr();
    if ((adcp)->thread != NULL) {
        Thread *tp;
        tp = (adcp)->thread;
        (adcp)->thread = NULL;
        tp->p_u.rdymsg = RDY_OK;
        chSchReadyI(tp);
    }
    chSysUnlockFromIsr();
}
```

Wakes up the waiting thread.

**Parameters**

in *adcp* pointer to the [ADCDriver](#) object

**Function Class:**

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

6.3.6.6 #define \_adc\_timeout\_isr( *adcp* )**Value:**

```
{
    chSysLockFromIsr();
    if ((adcp)->thread != NULL) {
        Thread *tp;
        tp = (adcp)->thread;
        (adcp)->thread = NULL;
        tp->p_u.rdymsg = RDY_TIMEOUT;
        chSchReadyI(tp);
    }
    chSysUnlockFromIsr();
}
```

Wakes up the waiting thread with a timeout message.

#### Parameters

in *adcp* pointer to the [ADCDriver](#) object

#### Function Class:

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

#### 6.3.6.7 #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 *adcp* pointer to the [ADCDriver](#) object

#### Function Class:

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

#### 6.3.6.8 #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 *adcp* pointer to the [ADCDriver](#) object

#### Function Class:

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

6.3.6.9 #define \_adc\_isr\_error\_code( *adcp*, *err* )

**Value:**

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

Common ISR code, error event.

This code handles the portable part of the ISR code:

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

**Note**

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

**Parameters**

in	<i>adcp</i>	pointer to the <a href="#">ADCDriver</a> object
in	<i>err</i>	platform dependent error code

**Function Class:**

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

6.3.6.10 #define ADC\_CR2\_EXTSEL\_SRC( *n* ) ((*n*) << 17)

Trigger source.

6.3.6.11 #define ADC\_CR2\_EXTSEL\_SWSTART (7 << 17)

Software trigger.

6.3.6.12 #define ADC\_CHANNEL\_IN0 0

External analog input 0.

6.3.6.13 #define ADC\_CHANNEL\_IN1 1

External analog input 1.

6.3.6.14 #define ADC\_CHANNEL\_IN2 2

External analog input 2.

6.3.6.15 #define ADC\_CHANNEL\_IN3 3

External analog input 3.

6.3.6.16 #define ADC\_CHANNEL\_IN4 4

External analog input 4.

6.3.6.17 #define ADC\_CHANNEL\_IN5 5

External analog input 5.

6.3.6.18 #define ADC\_CHANNEL\_IN6 6

External analog input 6.

6.3.6.19 #define ADC\_CHANNEL\_IN7 7

External analog input 7.

6.3.6.20 #define ADC\_CHANNEL\_IN8 8

External analog input 8.

6.3.6.21 #define ADC\_CHANNEL\_IN9 9

External analog input 9.

6.3.6.22 #define ADC\_CHANNEL\_IN10 10

External analog input 10.

6.3.6.23 #define ADC\_CHANNEL\_IN11 11

External analog input 11.

6.3.6.24 #define ADC\_CHANNEL\_IN12 12

External analog input 12.

6.3.6.25 #define ADC\_CHANNEL\_IN13 13

External analog input 13.

6.3.6.26 #define ADC\_CHANNEL\_IN14 14

External analog input 14.

6.3.6.27 #define ADC\_CHANNEL\_IN15 15

External analog input 15.

6.3.6.28 #define ADC\_CHANNEL\_SENSOR 16

Internal temperature sensor.

6.3.6.29 #define ADC\_CHANNEL\_VREFINT 17

Internal reference.

6.3.6.30 #define ADC\_SAMPLE\_1P5 0

1.5 cycles sampling time.

6.3.6.31 #define ADC\_SAMPLE\_7P5 1

7.5 cycles sampling time.

6.3.6.32 #define ADC\_SAMPLE\_13P5 2

13.5 cycles sampling time.

6.3.6.33 #define ADC\_SAMPLE\_28P5 3

28.5 cycles sampling time.

6.3.6.34 #define ADC\_SAMPLE\_41P5 4

41.5 cycles sampling time.

6.3.6.35 #define ADC\_SAMPLE\_55P5 5

55.5 cycles sampling time.

6.3.6.36 #define ADC\_SAMPLE\_71P5 6

71.5 cycles sampling time.

6.3.6.37 #define ADC\_SAMPLE\_239P5 7

239.5 cycles sampling time.

6.3.6.38 #define STM32\_ADC\_USE\_ADC1 TRUE

ADC1 driver enable switch.

If set to TRUE the support for ADC1 is included.

**Note**

The default is TRUE.

6.3.6.39 #define STM32\_ADC\_ADC1\_DMA\_PRIORITY 2

ADC1 DMA priority (0..3|lowest..highest).

6.3.6.40 #define STM32\_ADC\_ADC1\_IRQ\_PRIORITY 5

ADC1 interrupt priority level setting.

6.3.6.41 #define ADC\_SQR1\_NUM\_CH( n ) (((n) - 1) << 20)

Number of channels in a conversion sequence.

6.3.6.42 #define ADC\_SQR3\_SQ1\_N( n ) ((n) << 0)

1st channel in seq.

6.3.6.43 #define ADC\_SQR3\_SQ2\_N( n ) ((n) << 5)

2nd channel in seq.

6.3.6.44 #define ADC\_SQR3\_SQ3\_N( n ) ((n) << 10)

3rd channel in seq.

6.3.6.45 #define ADC\_SQR3\_SQ4\_N( n ) ((n) << 15)

4th channel in seq.

6.3.6.46 #define ADC\_SQR3\_SQ5\_N( n ) ((n) << 20)

5th channel in seq.

6.3.6.47 #define ADC\_SQR3\_SQ6\_N( n ) ((n) << 25)

6th channel in seq.

6.3.6.48 #define ADC\_SQR2\_SQ7\_N( n ) ((n) << 0)

7th channel in seq.

6.3.6.49 #define ADC\_SQR2\_SQ8\_N( n ) ((n) << 5)

8th channel in seq.

6.3.6.50 #define ADC\_SQR2\_SQ9\_N( n ) ((n) << 10)

9th channel in seq.

6.3.6.51 #define ADC\_SQR2\_SQ10\_N( n ) ((n) << 15)

10th channel in seq.

6.3.6.52 #define ADC\_SQR2\_SQ11\_N( n ) ((n) << 20)

11th channel in seq.

6.3.6.53 #define ADC\_SQR2\_SQ12\_N( n ) ((n) << 25)

12th channel in seq.

6.3.6.54 #define ADC\_SQR1\_SQ13\_N( n ) ((n) << 0)

13th channel in seq.

6.3.6.55 #define ADC\_SQR1\_SQ14\_N( n ) ((n) << 5)

14th channel in seq.

6.3.6.56 #define ADC\_SQR1\_SQ15\_N( n ) ((n) << 10)

15th channel in seq.

6.3.6.57 #define ADC\_SQR1\_SQ16\_N( n ) ((n) << 15)

16th channel in seq.

6.3.6.58 #define ADC\_SMPR2\_SMP\_AN0( n ) ((n) << 0)

AN0 sampling time.

6.3.6.59 #define ADC\_SMPR2\_SMP\_AN1( n ) ((n) << 3)

AN1 sampling time.

6.3.6.60 #define ADC\_SMPR2\_SMP\_AN2( n ) ((n) << 6)

AN2 sampling time.

6.3.6.61 #define ADC\_SMPR2\_SMP\_AN3( n ) ((n) << 9)

AN3 sampling time.

```
6.3.6.62 #define ADC_SMPR2_SMP_AN4( n ) ((n) << 12)
```

AN4 sampling time.

```
6.3.6.63 #define ADC_SMPR2_SMP_AN5( n ) ((n) << 15)
```

AN5 sampling time.

```
6.3.6.64 #define ADC_SMPR2_SMP_AN6( n ) ((n) << 18)
```

AN6 sampling time.

```
6.3.6.65 #define ADC_SMPR2_SMP_AN7( n ) ((n) << 21)
```

AN7 sampling time.

```
6.3.6.66 #define ADC_SMPR2_SMP_AN8( n ) ((n) << 24)
```

AN8 sampling time.

```
6.3.6.67 #define ADC_SMPR2_SMP_AN9( n ) ((n) << 27)
```

AN9 sampling time.

```
6.3.6.68 #define ADC_SMPR1_SMP_AN10( n ) ((n) << 0)
```

AN10 sampling time.

```
6.3.6.69 #define ADC_SMPR1_SMP_AN11( n ) ((n) << 3)
```

AN11 sampling time.

```
6.3.6.70 #define ADC_SMPR1_SMP_AN12( n ) ((n) << 6)
```

AN12 sampling time.

```
6.3.6.71 #define ADC_SMPR1_SMP_AN13( n ) ((n) << 9)
```

AN13 sampling time.

```
6.3.6.72 #define ADC_SMPR1_SMP_AN14( n ) ((n) << 12)
```

AN14 sampling time.

```
6.3.6.73 #define ADC_SMPR1_SMP_AN15( n ) ((n) << 15)
```

AN15 sampling time.

```
6.3.6.74 #define ADC_SMPR1_SMP_SENSOR( n ) ((n) << 18)
```

Temperature Sensor sampling time.

```
6.3.6.75 #define ADC_SMPR1_SMP_VREF( n ) ((n) << 21)
```

Voltage Reference sampling time.

### 6.3.7 Typedef Documentation

```
6.3.7.1 typedef uint16_t adcsample_t
```

ADC sample data type.

```
6.3.7.2 typedef uint16_t adc_channels_num_t
```

Channels number in a conversion group.

```
6.3.7.3 typedef struct ADCDriver ADCDriver
```

Type of a structure representing an ADC driver.

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

ADC notification callback type.

#### Parameters

in	<i>adcp</i>	pointer to the <a href="#">ADCDriver</a> 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>

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

ADC error callback type.

#### Parameters

in	<i>adcp</i>	pointer to the <a href="#">ADCDriver</a> object triggering the callback
----	-------------	---

### 6.3.8 Enumeration Type Documentation

```
6.3.8.1 enum adcstate_t
```

Driver state machine possible states.

#### Enumerator:

**ADC\_UNINIT** Not initialized.

**ADC\_STOP** Stopped.

**ADC\_READY** Ready.

**ADC\_ACTIVE** Converting.

**ADC\_COMPLETE** Conversion complete.

**ADC\_ERROR** Conversion complete.

### 6.3.8.2 enum adcerror\_t

Possible ADC failure causes.

#### Note

Error codes are architecture dependent and should not relied upon.

#### Enumerator:

**ADC\_ERR\_DMAFAILURE** DMA operations failure.

## 6.4 CAN Driver

### 6.4.1 Detailed Description

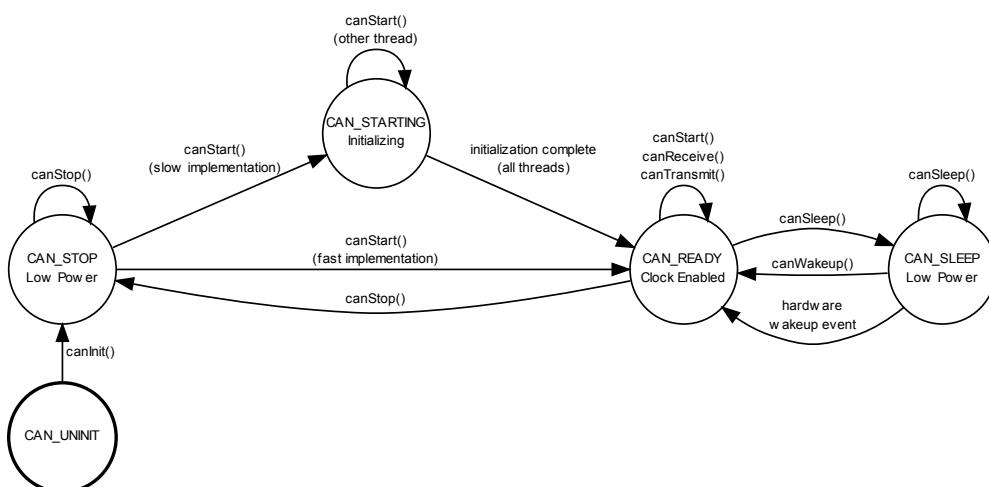
Generic CAN Driver. This module implements a generic CAN (Controller Area Network) driver allowing the exchange of information at frame level.

#### Precondition

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

### 6.4.2 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).



## Data Structures

- struct **CANTxFrame**  
*CAN transmission frame.*
- struct **CANRxFrame**  
*CAN received frame.*
- struct **CANFilter**  
*CAN filter.*
- 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.*
- msg\_t **canTransmit** (**CANDriver** \*canp, const **CANTxFrame** \*ctfp, systime\_t timeout)  
*Can frame transmission.*
- msg\_t **canReceive** (**CANDriver** \*canp, **CANRxFrame** \*crfp, systime\_t timeout)  
*Can frame receive.*
- canstatus\_t **canGetAndClearFlags** (**CANDriver** \*canp)  
*Returns the current status mask and clears it.*
- void **canSleep** (**CANDriver** \*canp)  
*Enters the sleep mode.*
- void **canWakeup** (**CANDriver** \*canp)  
*Enforces leaving the sleep mode.*
- **CH\_IRQ\_HANDLER** (**CAN1\_TX\_IRQHandler**)  
*CAN1 TX interrupt handler.*
- **CH\_IRQ\_HANDLER** (**CAN1\_RX1\_IRQHandler**)  
*CAN1 RX1 interrupt handler.*
- **CH\_IRQ\_HANDLER** (**CAN1\_SCE\_IRQHandler**)  
*CAN1 SCE interrupt handler.*
- void **can\_lld\_init** (void)  
*Low level CAN driver initialization.*
- void **can\_lld\_start** (**CANDriver** \*canp)  
*Configures and activates the CAN peripheral.*
- void **can\_lld\_stop** (**CANDriver** \*canp)  
*Deactivates the CAN peripheral.*
- bool\_t **can\_lld\_can\_transmit** (**CANDriver** \*canp)  
*Determines whether a frame can be transmitted.*
- void **can\_lld\_transmit** (**CANDriver** \*canp, const **CANTxFrame** \*ctfp)  
*Inserts a frame into the transmit queue.*
- bool\_t **can\_lld\_can\_receive** (**CANDriver** \*canp)  
*Determines whether a frame has been received.*

- void `can_lld_receive (CANDriver *canp, CANRxFrame *crfp)`  
*Receives a frame from the input queue.*
- void `can_lld_sleep (CANDriver *canp)`  
*Enters the sleep mode.*
- void `can_lld_wakeup (CANDriver *canp)`  
*Enforces leaving the sleep mode.*

## Variables

- CANDriver CAND1  
*ADC1 driver identifier.*

## CAN status flags

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

## CAN configuration options

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

## Macro Functions

- #define `canAddFlagsI(canp, mask) ((canp)->status |= (mask))`  
*Adds some flags to the CAN status mask.*

## Configuration options

- #define `STM32_CAN_USE_CAN1` TRUE  
*CAN1 driver enable switch.*
- #define `STM32_CAN_CAN1_IRQ_PRIORITY` 11  
*CAN1 interrupt priority level setting.*

## Defines

- #define `CAN_SUPPORTS_SLEEP` TRUE  
*This switch defines whether the driver implementation supports a low power switch mode with automatic an wakeup feature.*
- #define `CAN_MAX_FILTERS` 28  
*Minimum number of CAN filters.*
- #define `CAN_BTR_BR(n)` (n)

- #define **CAN\_BTR\_TS1**(n) ((n) << 16)  
*TS1 field macro.*
- #define **CAN\_BTR\_TS2**(n) ((n) << 20)  
*TS2 field macro.*
- #define **CAN\_BTR\_SJW**(n) ((n) << 24)  
*SJW field macro.*
- #define **CAN\_IDE\_STD** 0  
*Standard id.*
- #define **CAN\_IDE\_EXT** 1  
*Extended id.*
- #define **CAN\_RTR\_DATA** 0  
*Data frame.*
- #define **CAN\_RTR\_REMOTE** 1  
*Remote frame.*

## Typedefs

- typedef uint32\_t **canstatus\_t**  
*CAN status flags.*

## Enumerations

- enum **canstate\_t** {
   
**CAN\_UNINIT** = 0, **CAN\_STOP** = 1, **CAN\_STARTING** = 2, **CAN\_READY** = 3,  
**CAN\_SLEEP** = 4
 }
   
*Driver state machine possible states.*

## 6.4.3 Function Documentation

### 6.4.3.1 void canInit( void )

CAN Driver initialization.

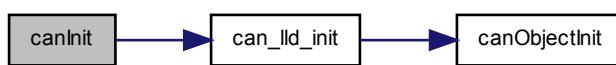
#### Note

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:



**6.4.3.2 void canObjectInit ( CANDriver \* *canp* )**

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

**Parameters**

out                    *canp* pointer to the [CANDriver](#) object

**Function Class:**

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

**6.4.3.3 void canStart ( CANDriver \* *canp*, const CANConfig \* *config* )**

Configures and activates the CAN peripheral.

**Note**

Activating the CAN bus can be a slow operation this this function is not atomic, it waits internally for the initialization to complete.

**Parameters**

in                    *canp* pointer to the [CANDriver](#) object

in                    *config* pointer to the [CANConfig](#) object. Depending on the implementation the value can be NULL.

**Function Class:**

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

Here is the call graph for this function:

**6.4.3.4 void canStop ( CANDriver \* *canp* )**

Deactivates the CAN peripheral.

**Parameters**

in                    *canp* pointer to the [CANDriver](#) object

**Function Class:**

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

Here is the call graph for this function:



#### 6.4.3.5 msg\_t canTransmit ( CANDriver \* *canp*, 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 <a href="#">CANDriver</a> object
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"><li>• <i>TIME_IMMEDIATE</i> immediate timeout.</li><li>• <i>TIME_INFINITE</i> no timeout.</li></ul>

##### Returns

The operation result.

##### Return values

*RDY\_OK* the frame has been queued for transmission.

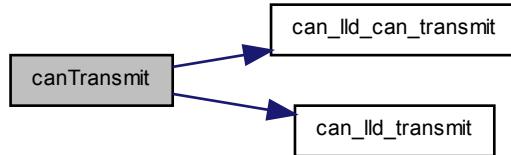
*RDY\_TIMEOUT* The operation has timed out.

*RDY\_RESET* The driver has been stopped while waiting.

##### Function Class:

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

Here is the call graph for this function:



#### 6.4.3.6 msg\_t canReceive ( CANDriver \* *canp*, 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 <a href="#">CANDriver</a> object
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"> <li>• <i>TIME_IMMEDIATE</i> immediate timeout (useful in an event driven scenario where a thread never blocks for I/O).</li> <li>• <i>TIME_INFINITE</i> no timeout.</li> </ul>

##### Returns

The operation result.

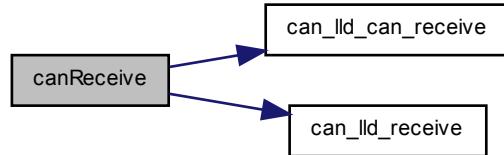
##### Return values

<i>RDY_OK</i>	a frame has been received and placed in the buffer.
<i>RDY_TIMEOUT</i>	The operation has timed out.
<i>RDY_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:



#### 6.4.3.7 `canstatus_t canGetAndClearFlags ( CANDriver * canp )`

Returns the current status mask and clears it.

##### Parameters

in *canp* pointer to the `CANDriver` object

##### Returns

The status flags mask.

##### Function Class:

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

#### 6.4.3.8 `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.

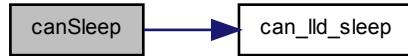
##### Parameters

in *canp* pointer to the `CANDriver` object

##### Function Class:

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

Here is the call graph for this function:



#### 6.4.3.9 void canWakeup ( [CANDriver](#) \* *canp* )

Enforces leaving the sleep mode.

##### Note

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

##### Parameters

in                    *canp*    pointer to the [CANDriver](#) object

Here is the call graph for this function:



#### 6.4.3.10 CH\_IRQ\_HANDLER ( [CAN1\\_TX\\_IRQHandler](#) )

CAN1 TX interrupt handler.

##### Function Class:

Interrupt handler, this function should not be directly invoked.

#### 6.4.3.11 CH\_IRQ\_HANDLER ( [CAN1\\_RX1\\_IRQHandler](#) )

CAN1 RX1 interrupt handler.

##### Function Class:

Interrupt handler, this function should not be directly invoked.

**6.4.3.12 CH\_IRQ\_HANDLER ( CAN1\_SCE\_IRQHandler )**

CAN1 SCE interrupt handler.

**Function Class:**

Interrupt handler, this function should not be directly invoked.

**6.4.3.13 void can\_lld\_init ( void )**

Low level CAN driver initialization.

**Function Class:**

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

Here is the call graph for this function:

**6.4.3.14 void can\_lld\_start ( CANDriver \* canp )**

Configures and activates the CAN peripheral.

**Parameters**

in                    *canp*    pointer to the [CANDriver](#) object

**Function Class:**

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

**6.4.3.15 void can\_lld\_stop ( CANDriver \* canp )**

Deactivates the CAN peripheral.

**Parameters**

in                    *canp*    pointer to the [CANDriver](#) object

**Function Class:**

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

**6.4.3.16 bool\_t can\_lld\_can\_transmit ( CANDriver \* canp )**

Determines whether a frame can be transmitted.

**Parameters**

in                    *canp* pointer to the `CANDriver` object

**Returns**

The queue space availability.

**Return values**

*FALSE* no space in the transmit queue.

*TRUE* transmit slot available.

**Function Class:**

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

**6.4.3.17 void can\_lld\_transmit ( `CANDriver` \* *canp*, `const CANTxFrame` \* *ctfp* )**

Inserts a frame into the transmit queue.

**Parameters**

in                    *canp* pointer to the `CANDriver` object

in                    *ctfp* pointer to the CAN frame to be transmitted

**Function Class:**

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

**6.4.3.18 bool\_t can\_lld\_can\_receive ( `CANDriver` \* *canp* )**

Determines whether a frame has been received.

**Parameters**

in                    *canp* pointer to the `CANDriver` object

**Returns**

The queue space availability.

**Return values**

*FALSE* no space in the transmit queue.

*TRUE* transmit slot available.

**Function Class:**

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

**6.4.3.19 void can\_lld\_receive ( `CANDriver` \* *canp*, `CANRxFrame` \* *crfp* )**

Receives a frame from the input queue.

**Parameters**

in                    *canp* pointer to the `CANDriver` object

out                  *crfp* pointer to the buffer where the CAN frame is copied

**Function Class:**

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

**6.4.3.20 void can\_lld\_sleep ( CANDriver \* *canp* )**

Enters the sleep mode.

**Parameters**

in *canp* pointer to the [CANDriver](#) object

**Function Class:**

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

**6.4.3.21 void can\_lld\_wakeup ( CANDriver \* *canp* )**

Enforces leaving the sleep mode.

**Parameters**

in *canp* pointer to the [CANDriver](#) object

**Function Class:**

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

## 6.4.4 Variable Documentation

### 6.4.4.1 CANDriver CAND1

ADC1 driver identifier.

## 6.4.5 Define Documentation

### 6.4.5.1 #define CAN\_LIMIT\_WARNING 1

Errors rate warning.

### 6.4.5.2 #define CAN\_LIMIT\_ERROR 2

Errors rate error.

### 6.4.5.3 #define CAN\_BUS\_OFF\_ERROR 4

Bus off condition reached.

### 6.4.5.4 #define CAN\_FRAMING\_ERROR 8

Framing error of some kind on the CAN bus.

**6.4.5.5 #define CAN\_OVERFLOW\_ERROR 16**

Overflow in receive queue.

**6.4.5.6 #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.

**6.4.5.7 #define canAddFlags( canp, mask ) ((canp)->status |= (mask))**

Adds some flags to the CAN status mask.

**Parameters**

in	<code>canp</code>	pointer to the <code>CANDriver</code> object
in	<code>mask</code>	flags to be added to the status 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.

**6.4.5.8 #define CAN\_SUPPORTS\_SLEEP TRUE**

This switch defines whether the driver implementation supports a low power switch mode with automatic an wakeup feature.

**6.4.5.9 #define CAN\_MAX\_FILTERS 28**

Minimum number of CAN filters.

**6.4.5.10 #define CAN\_BTR\_BR( n )(n)**

BRP field macro.

**6.4.5.11 #define CAN\_BTR\_TS1( n )(n)<< 16**

TS1 field macro.

**6.4.5.12 #define CAN\_BTR\_TS2( n )(n)<< 20**

TS2 field macro.

**6.4.5.13 #define CAN\_BTR\_SJW( n )(n)<< 24**

SJW field macro.

**6.4.5.14 #define CAN\_IDE\_STD 0**

Standard id.

6.4.5.15 #define CAN\_IDE\_EXT 1

Extended id.

6.4.5.16 #define CAN\_RTR\_DATA 0

Data frame.

6.4.5.17 #define CAN\_RTR\_REMOTE 1

Remote frame.

6.4.5.18 #define STM32\_CAN\_USE\_CAN1 TRUE

CAN1 driver enable switch.

If set to TRUE the support for ADC1 is included.

#### Note

The default is TRUE.

6.4.5.19 #define STM32\_CAN\_CAN1\_IRQ\_PRIORITY 11

CAN1 interrupt priority level setting.

## 6.4.6 Typedef Documentation

6.4.6.1 typedef uint32\_t canstatus\_t

CAN status flags.

## 6.4.7 Enumeration Type Documentation

6.4.7.1 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.

## 6.5 EXT Driver

### 6.5.1 Detailed Description

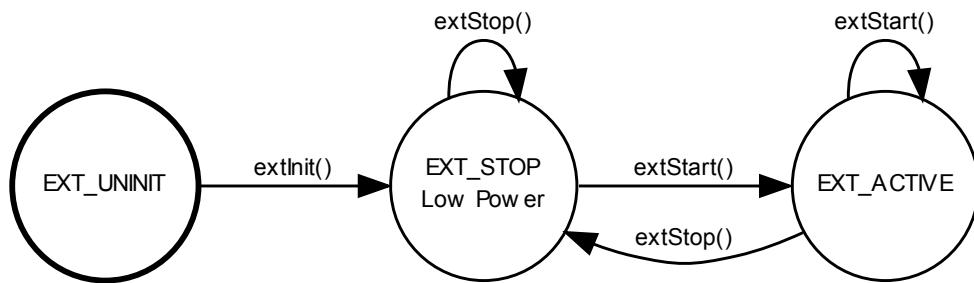
Generic EXT Driver. This module implements a generic EXT (EXternal) driver.

### Precondition

In order to use the EXT driver the `HAL_USE_EXT` option must be enabled in `halconf.h`.

### 6.5.2 Driver State Machine

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



### 6.5.3 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.

### 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.*
  - `CH_IRQ_HANDLER (EXTI0_IRQHandler)`  
*EXTI[0] interrupt handler.*
  - `CH_IRQ_HANDLER (EXTI1_IRQHandler)`  
*EXTI[1] interrupt handler.*
  - `CH_IRQ_HANDLER (EXTI2_IRQHandler)`  
*EXTI[2] interrupt handler.*
  - `CH_IRQ_HANDLER (EXTI3_IRQHandler)`  
*EXTI[3] interrupt handler.*
  - `CH_IRQ_HANDLER (EXTI4_IRQHandler)`  
*EXTI[4] interrupt handler.*
  - `CH_IRQ_HANDLER (EXTI9_5_IRQHandler)`  
*EXTI[5]...EXTI[9] interrupt handler.*
  - `CH_IRQ_HANDLER (EXTI15_10_IRQHandler)`  
*EXTI[10]...EXTI[15] interrupt handler.*
  - `CH_IRQ_HANDLER (PVD_IRQHandler)`  
*EXTI[16] interrupt handler (PVD).*
  - `CH_IRQ_HANDLER (RTCAlarm_IRQHandler)`  
*EXTI[17] interrupt handler (RTC).*
  - `CH_IRQ_HANDLER (USB_FS_WKUP_IRQHandler)`  
*EXTI[18] interrupt handler (USB\_FS\_WKUP).*
  - 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 EXTD1`  
*EXTD1 driver identifier.*

## EXTI configuration helpers

- #define `EXT_MODE_EXTI`(m0, m1, m2, m3, m4, m5, m6, m7, m8, m9, m10, m11, m12, m13, m14, m15)  
*EXTI-GPIO association macro.*
- #define `EXT_MODE_GPIOA` 0  
*GPIOA identifier.*
- #define `EXT_MODE_GPIOB` 1  
*GPIOB identifier.*
- #define `EXT_MODE_GPIOC` 2  
*GPIOC identifier.*
- #define `EXT_MODE_GPIOD` 3  
*GPIOD identifier.*
- #define `EXT_MODE_GPIOE` 4  
*GPIOE identifier.*
- #define `EXT_MODE_GPIOF` 5  
*GPIOF identifier.*
- #define `EXT_MODE_GPIOG` 6  
*GPIOG identifier.*
- #define `EXT_MODE_GPIOH` 7  
*GPIOH identifier.*
- #define `EXT_MODE_GPIOI` 8  
*GPIOI identifier.*

## Configuration options

- #define `STM32_EXT_EXTI0_IRQ_PRIORITY` 6  
*EXTI0 interrupt priority level setting.*
- #define `STM32_EXT_EXTI1_IRQ_PRIORITY` 6  
*EXTI1 interrupt priority level setting.*
- #define `STM32_EXT_EXTI2_IRQ_PRIORITY` 6  
*EXTI2 interrupt priority level setting.*
- #define `STM32_EXT_EXTI3_IRQ_PRIORITY` 6  
*EXTI3 interrupt priority level setting.*
- #define `STM32_EXT_EXTI4_IRQ_PRIORITY` 6  
*EXTI4 interrupt priority level setting.*
- #define `STM32_EXT_EXTI5_9_IRQ_PRIORITY` 6  
*EXTI9..5 interrupt priority level setting.*
- #define `STM32_EXT_EXTI10_15_IRQ_PRIORITY` 6  
*EXTI15..10 interrupt priority level setting.*
- #define `STM32_EXT_EXTI16_IRQ_PRIORITY` 6  
*EXTI16 interrupt priority level setting.*
- #define `STM32_EXT_EXTI17_IRQ_PRIORITY` 6  
*EXTI17 interrupt priority level setting.*
- #define `STM32_EXT_EXTI18_IRQ_PRIORITY` 6  
*EXTI18 interrupt priority level setting.*
- #define `STM32_EXT_EXTI19_IRQ_PRIORITY` 6  
*EXTI19 interrupt priority level setting.*
- #define `STM32_EXT_EXTI20_IRQ_PRIORITY` 6  
*EXTI20 interrupt priority level setting.*
- #define `STM32_EXT_EXTI21_IRQ_PRIORITY` 6  
*EXTI21 interrupt priority level setting.*
- #define `STM32_EXT_EXTI22_IRQ_PRIORITY` 6  
*EXTI22 interrupt priority level setting.*

## Defines

- `#define EXT_MAX_CHANNELS STM32 EXTI_NUM_CHANNELS`  
*Available number of EXT channels.*
- `#define EXT_CHANNELS_MASK ((1 << EXT_MAX_CHANNELS) - 1)`  
*Mask of the available channels.*

## 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.*

### 6.5.4 Function Documentation

#### 6.5.4.1 void extInit( void )

EXT Driver initialization.

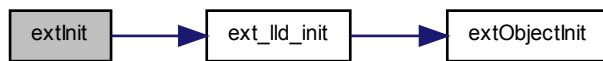
##### Note

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:



#### 6.5.4.2 void extObjectInit( EXTDriver \* extp )

Initializes the standard part of a `EXTDriver` structure.

##### Parameters

`out extp` pointer to the `EXTDriver` object

##### Function Class:

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

**6.5.4.3 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:

**6.5.4.4 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:

**6.5.4.5 void extChannelEnable ( EXTDriver \* *extp*, expchannel\_t *channel* )**

Enables an EXT channel.

**Parameters**

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

**Function Class:**

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

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

Disables an EXT channel.

**Parameters**

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

**Function Class:**

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

**6.5.4.7 CH\_IRQ\_HANDLER ( `EXTI0_IRQHandler` )**

EXTI[0] interrupt handler.

**Function Class:**

Interrupt handler, this function should not be directly invoked.

**6.5.4.8 CH\_IRQ\_HANDLER ( `EXTI1_IRQHandler` )**

EXTI[1] interrupt handler.

**Function Class:**

Interrupt handler, this function should not be directly invoked.

**6.5.4.9 CH\_IRQ\_HANDLER ( `EXTI2_IRQHandler` )**

EXTI[2] interrupt handler.

**Function Class:**

Interrupt handler, this function should not be directly invoked.

**6.5.4.10 CH\_IRQ\_HANDLER ( `EXTI3_IRQHandler` )**

EXTI[3] interrupt handler.

**Function Class:**

Interrupt handler, this function should not be directly invoked.

**6.5.4.11 CH\_IRQ\_HANDLER ( EXTI4\_IRQHandler )**

EXTI[4] interrupt handler.

**Function Class:**

Interrupt handler, this function should not be directly invoked.

**6.5.4.12 CH\_IRQ\_HANDLER ( EXTI9\_5\_IRQHandler )**

EXTI[5]...EXTI[9] interrupt handler.

**Function Class:**

Interrupt handler, this function should not be directly invoked.

**6.5.4.13 CH\_IRQ\_HANDLER ( EXTI15\_10\_IRQHandler )**

EXTI[10]...EXTI[15] interrupt handler.

**Function Class:**

Interrupt handler, this function should not be directly invoked.

**6.5.4.14 CH\_IRQ\_HANDLER ( PVD\_IRQHandler )**

EXTI[16] interrupt handler (PVD).

**Function Class:**

Interrupt handler, this function should not be directly invoked.

**6.5.4.15 CH\_IRQ\_HANDLER ( RTCAlarm\_IRQHandler )**

EXTI[17] interrupt handler (RTC).

**Function Class:**

Interrupt handler, this function should not be directly invoked.

**6.5.4.16 CH\_IRQ\_HANDLER ( USB\_FS\_WKUP\_IRQHandler )**

EXTI[18] interrupt handler (USB\_FS\_WKUP).

**Function Class:**

Interrupt handler, this function should not be directly invoked.

**6.5.4.17 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:

**6.5.4.18 void ext\_lld\_start ( EXTDriver \* extp )**

Configures and activates the EXT peripheral.

**Parameters**

in                    *extp* pointer to the [EXTDriver](#) object

**Function Class:**

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

**6.5.4.19 void ext\_lld\_stop ( EXTDriver \* extp )**

Deactivates the EXT peripheral.

**Parameters**

in                    *extp* pointer to the [EXTDriver](#) object

**Function Class:**

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

**6.5.4.20 void ext\_lld\_channel\_enable ( EXTDriver \* extp, expchannel\_t channel )**

Enables an EXT channel.

**Parameters**

in                    *extp* pointer to the [EXTDriver](#) object  
in                    *channel* channel to be enabled

**Function Class:**

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

#### 6.5.4.21 void ext\_lld\_channel\_disable ( EXTDriver \* extp, expchannel\_t channel )

Disables an EXT channel.

##### Parameters

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

##### Function Class:

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

## 6.5.5 Variable Documentation

### 6.5.5.1 EXTDriver EXTD1

EXTD1 driver identifier.

## 6.5.6 Define Documentation

### 6.5.6.1 #define EXT\_MAX\_CHANNELS STM32\_EXTI\_NUM\_CHANNELS

Available number of EXT channels.

### 6.5.6.2 #define EXT\_CHANNELS\_MASK ((1 << EXT\_MAX\_CHANNELS) - 1)

Mask of the available channels.

### 6.5.6.3 #define EXT\_MODE\_EXTI( m0, m1, m2, m3, m4, m5, m6, m7, m8, m9, m10, m11, m12, m13, m14, m15 )

##### Value:

```
{
  ((m0) << 0) | ((m1) << 4) | ((m2) << 8) | ((m3) << 12),
  ((m4) << 0) | ((m5) << 4) | ((m6) << 8) | ((m7) << 12),
  ((m8) << 0) | ((m9) << 4) | ((m10) << 8) | ((m11) << 12),
  ((m12) << 0) | ((m13) << 4) | ((m14) << 8) | ((m15) << 12)
}
```

EXTI-GPIO association macro.

Helper macro to associate a GPIO to each of the Mx EXTI inputs.

### 6.5.6.4 #define EXT\_MODE\_GPIOA 0

GPIOA identifier.

### 6.5.6.5 #define EXT\_MODE\_GPIOB 1

GPIOB identifier.

### 6.5.6.6 #define EXT\_MODE\_GPIOC 2

GPIOC identifier.

6.5.6.7 #define EXT\_MODE\_GPIOD 3

GPIOD identifier.

6.5.6.8 #define EXT\_MODE\_GPIOE 4

GPIOE identifier.

6.5.6.9 #define EXT\_MODE\_GPIOF 5

GPIOF identifier.

6.5.6.10 #define EXT\_MODE\_GPIOG 6

GPIOG identifier.

6.5.6.11 #define EXT\_MODE\_GPIOH 7

GPIOH identifier.

6.5.6.12 #define EXT\_MODE\_GPIOI 8

GPIOI identifier.

6.5.6.13 #define STM32\_EXT\_EXTI0\_IRQ\_PRIORITY 6

EXTI0 interrupt priority level setting.

6.5.6.14 #define STM32\_EXT\_EXTI1\_IRQ\_PRIORITY 6

EXTI1 interrupt priority level setting.

6.5.6.15 #define STM32\_EXT\_EXTI2\_IRQ\_PRIORITY 6

EXTI2 interrupt priority level setting.

6.5.6.16 #define STM32\_EXT\_EXTI3\_IRQ\_PRIORITY 6

EXTI3 interrupt priority level setting.

6.5.6.17 #define STM32\_EXT\_EXTI4\_IRQ\_PRIORITY 6

EXTI4 interrupt priority level setting.

6.5.6.18 #define STM32\_EXT\_EXTI5\_9\_IRQ\_PRIORITY 6

EXTI9..5 interrupt priority level setting.

6.5.6.19 #define STM32\_EXT EXTI10\_IRQ\_PRIORITY 6

EXTI10 interrupt priority level setting.

6.5.6.20 #define STM32\_EXT EXTI16\_IRQ\_PRIORITY 6

EXTI16 interrupt priority level setting.

6.5.6.21 #define STM32\_EXT EXTI17\_IRQ\_PRIORITY 6

EXTI17 interrupt priority level setting.

6.5.6.22 #define STM32\_EXT EXTI18\_IRQ\_PRIORITY 6

EXTI18 interrupt priority level setting.

6.5.6.23 #define STM32\_EXT EXTI19\_IRQ\_PRIORITY 6

EXTI19 interrupt priority level setting.

6.5.6.24 #define STM32\_EXT EXTI20\_IRQ\_PRIORITY 6

EXTI20 interrupt priority level setting.

6.5.6.25 #define STM32\_EXT EXTI21\_IRQ\_PRIORITY 6

EXTI21 interrupt priority level setting.

6.5.6.26 #define STM32\_EXT EXTI22\_IRQ\_PRIORITY 6

EXTI22 interrupt priority level setting.

## 6.5.7 Typedef Documentation

6.5.7.1 **typedef uint32\_t expchannel\_t**

EXT channel identifier.

6.5.7.2 **typedef void(\* extcallback\_t)(EXTDriver \*extp, expchannel\_t channel)**

Type of an EXT generic notification callback.

### Parameters

in *extp* pointer to the EXPDriver object triggering the callback

## 6.6 GPT Driver

### 6.6.1 Detailed Description

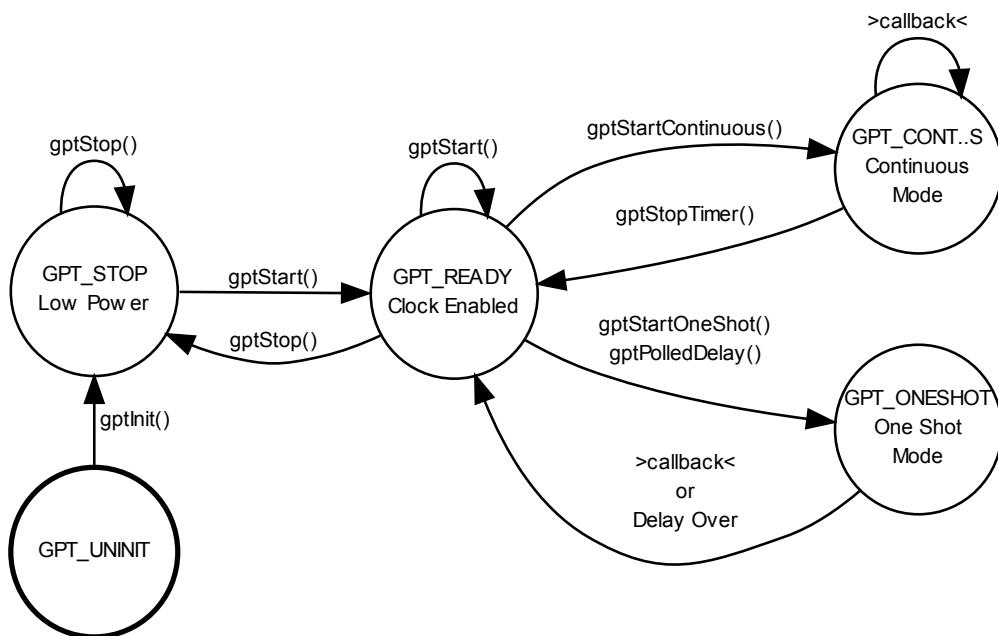
Generic GPT Driver. This module implements a generic GPT (General Purpose Timer) driver. The timer can be programmed in order to trigger callbacks after a specified time period or continuously with a specified interval.

#### Precondition

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

### 6.6.2 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).



### 6.6.3 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.

## 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 [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.*

## Variables

- `GPTDriver GPTD1`  
`GPTD1 driver identifier.`
- `GPTDriver GPTD2`  
`GPTD2 driver identifier.`
- `GPTDriver GPTD3`  
`GPTD3 driver identifier.`
- `GPTDriver GPTD4`  
`GPTD4 driver identifier.`
- `GPTDriver GPTD5`  
`GPTD5 driver identifier.`
- `GPTDriver GPTD8`  
`GPTD8 driver identifier.`

## Configuration options

- `#define STM32_GPT_USE_TIM1 TRUE`  
`GPTD1 driver enable switch.`
- `#define STM32_GPT_USE_TIM2 TRUE`  
`GPTD2 driver enable switch.`
- `#define STM32_GPT_USE_TIM3 TRUE`  
`GPTD3 driver enable switch.`
- `#define STM32_GPT_USE_TIM4 TRUE`  
`GPTD4 driver enable switch.`
- `#define STM32_GPT_USE_TIM5 TRUE`  
`GPTD5 driver enable switch.`
- `#define STM32_GPT_USE_TIM8 TRUE`  
`GPTD8 driver enable switch.`
- `#define STM32_GPT_TIM1_IRQ_PRIORITY 7`  
`GPTD1 interrupt priority level setting.`
- `#define STM32_GPT_TIM2_IRQ_PRIORITY 7`  
`GPTD2 interrupt priority level setting.`
- `#define STM32_GPT_TIM3_IRQ_PRIORITY 7`  
`GPTD3 interrupt priority level setting.`
- `#define STM32_GPT_TIM4_IRQ_PRIORITY 7`  
`GPTD4 interrupt priority level setting.`
- `#define STM32_GPT_TIM5_IRQ_PRIORITY 7`  
`GPTD5 interrupt priority level setting.`
- `#define STM32_GPT_TIM8_IRQ_PRIORITY 7`  
`GPTD5 interrupt priority level setting.`

## 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.`

## Enumerations

```
• enum gptstate_t {
    GPT_UNINIT = 0, GPT_STOP = 1, GPT_READY = 2, GPT_CONTINUOUS = 3,
    GPT_ONESHOT = 4 }
```

*Driver state machine possible states.*

### 6.6.4 Function Documentation

#### 6.6.4.1 void gptInit ( void )

GPT Driver initialization.

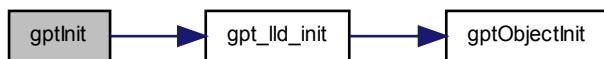
##### Note

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

##### Function Class:

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

Here is the call graph for this function:



#### 6.6.4.2 void gptObjectInit ( GPTDriver \* gptp )

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

##### Parameters

out	<i>gptp</i> pointer to the <a href="#">GPTDriver</a> object
-----	---

##### Function Class:

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

#### 6.6.4.3 void gptStart ( GPTDriver \* gptp, const GPTConfig \* config )

Configures and activates the GPT peripheral.

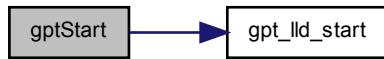
##### Parameters

in	<i>gptp</i> pointer to the <a href="#">GPTDriver</a> object
in	<i>config</i> pointer to the <a href="#">GPTConfig</a> 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:

**6.6.4.4 void gptStop ( GPTDriver \* gptp )**

Deactivates the GPT peripheral.

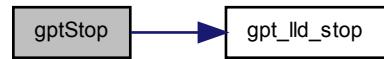
**Parameters**

in                    *gptp*    pointer to the [GPTDriver](#) object

**Function Class:**

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

Here is the call graph for this function:

**6.6.4.5 void gptStartContinuous ( GPTDriver \* gptp, gptcnt\_t interval )**

Starts the timer in continuous mode.

**Parameters**

in                    *gptp*    pointer to the [GPTDriver](#) object  
in                    *interval*   period in ticks

**Function Class:**

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

Here is the call graph for this function:



#### 6.6.4.6 void gptStartContinuous( GPTDriver \* gptp, gptcnt\_t interval )

Starts the timer in continuous mode.

##### Parameters

in	<i>gptp</i>	pointer to the <a href="#">GPTDriver</a> 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:



#### 6.6.4.7 void gptStartOneShot( GPTDriver \* gptp, gptcnt\_t interval )

Starts the timer in one shot mode.

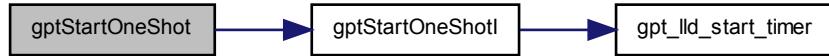
##### Parameters

in	<i>gptp</i>	pointer to the <a href="#">GPTDriver</a> 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:



#### 6.6.4.8 void gptStartOneShotl ( GPTDriver \* *gptp*, gptcnt\_t *interval* )

Starts the timer in one shot mode.

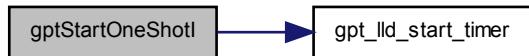
##### Parameters

in	<i>gptp</i>	pointer to the <a href="#">GPTDriver</a> 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:



#### 6.6.4.9 void gptStopTimer ( GPTDriver \* *gptp* )

Stops the timer.

##### Parameters

in	<i>gptp</i>	pointer to the <a href="#">GPTDriver</a> 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:



#### 6.6.4.10 void gptStopTimerI ( GPTDriver \* *gptp* )

Stops the timer.

##### Parameters

in	<i>gptp</i> pointer to the <a href="#">GPTDriver</a> 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:



#### 6.6.4.11 void gptPolledDelay ( GPTDriver \* *gptp*, gptcnt\_t *interval* )

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

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

##### Note

The configured callback is not invoked when using this function.

##### Parameters

in	<i>gptp</i> pointer to the <a href="#">GPTDriver</a> 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:



#### 6.6.4.12 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:



#### 6.6.4.13 void gpt\_lld\_start ( GPTDriver \* gptp )

Configures and activates the GPT peripheral.

##### Parameters

in                    *gptp*    pointer to the [GPTDriver](#) object

##### Function Class:

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

#### 6.6.4.14 void gpt\_lld\_stop ( GPTDriver \* gptp )

Deactivates the GPT peripheral.

##### Parameters

in                    *gptp*    pointer to the [GPTDriver](#) object

##### Function Class:

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

**6.6.4.15 void gpt\_lld\_start\_timer ( *GPTDriver* \* *gptp*, *gptcnt\_t* *interval* )**

Starts the timer in continuous mode.

**Parameters**

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

**Function Class:**

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

**6.6.4.16 void gpt\_lld\_stop\_timer ( *GPTDriver* \* *gptp* )**

Stops the timer.

**Parameters**

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

**Function Class:**

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

**6.6.4.17 void gpt\_lld\_polled\_delay ( *GPTDriver* \* *gptp*, *gptcnt\_t* *interval* )**

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

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

**Parameters**

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

**Function Class:**

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

**6.6.5 Variable Documentation****6.6.5.1 GPTDriver GPTD1**

GPTD1 driver identifier.

**Note**

The driver GPTD1 allocates the complex timer TIM1 when enabled.

**6.6.5.2 GPTDriver GPTD2**

GPTD2 driver identifier.

**Note**

The driver GPTD2 allocates the timer TIM2 when enabled.

#### 6.6.5.3 GPTDriver GPTD3

GPTD3 driver identifier.

##### Note

The driver GPTD3 allocates the timer TIM3 when enabled.

#### 6.6.5.4 GPTDriver GPTD4

GPTD4 driver identifier.

##### Note

The driver GPTD4 allocates the timer TIM4 when enabled.

#### 6.6.5.5 GPTDriver GPTD5

GPTD5 driver identifier.

##### Note

The driver GPTD5 allocates the timer TIM5 when enabled.

#### 6.6.5.6 GPTDriver GPTD8

GPTD8 driver identifier.

##### Note

The driver GPTD8 allocates the timer TIM8 when enabled.

### 6.6.6 Define Documentation

#### 6.6.6.1 #define STM32\_GPT\_USE\_TIM1 TRUE

GPTD1 driver enable switch.

If set to TRUE the support for GPTD1 is included.

##### Note

The default is TRUE.

#### 6.6.6.2 #define STM32\_GPT\_USE\_TIM2 TRUE

GPTD2 driver enable switch.

If set to TRUE the support for GPTD2 is included.

##### Note

The default is TRUE.

**6.6.6.3 #define STM32\_GPT\_USE\_TIM3 TRUE**

GPTD3 driver enable switch.

If set to TRUE the support for GPTD3 is included.

**Note**

The default is TRUE.

**6.6.6.4 #define STM32\_GPT\_USE\_TIM4 TRUE**

GPTD4 driver enable switch.

If set to TRUE the support for GPTD4 is included.

**Note**

The default is TRUE.

**6.6.6.5 #define STM32\_GPT\_USE\_TIM5 TRUE**

GPTD5 driver enable switch.

If set to TRUE the support for GPTD5 is included.

**Note**

The default is TRUE.

**6.6.6.6 #define STM32\_GPT\_USE\_TIM8 TRUE**

GPTD8 driver enable switch.

If set to TRUE the support for GPTD8 is included.

**Note**

The default is TRUE.

**6.6.6.7 #define STM32\_GPT\_TIM1\_IRQ\_PRIORITY 7**

GPTD1 interrupt priority level setting.

**6.6.6.8 #define STM32\_GPT\_TIM2\_IRQ\_PRIORITY 7**

GPTD2 interrupt priority level setting.

**6.6.6.9 #define STM32\_GPT\_TIM3\_IRQ\_PRIORITY 7**

GPTD3 interrupt priority level setting.

**6.6.6.10 #define STM32\_GPT\_TIM4\_IRQ\_PRIORITY 7**

GPTD4 interrupt priority level setting.

6.6.6.11 `#define STM32_GPT_TIM5_IRQ_PRIORITY 7`

GPTD5 interrupt priority level setting.

6.6.6.12 `#define STM32_GPT_TIM8_IRQ_PRIORITY 7`

GPTD5 interrupt priority level setting.

## 6.6.7 Typedef Documentation

6.6.7.1 `typedef struct GPTDriver GPTDriver`

Type of a structure representing a GPT driver.

6.6.7.2 `typedef void(* gptcallback_t)(GPTDriver *gpt)`

GPT notification callback type.

### Parameters

in *gpt* pointer to a `GPTDriver` object

6.6.7.3 `typedef uint32_t gptfreq_t`

GPT frequency type.

6.6.7.4 `typedef uint16_t gptcnt_t`

GPT counter type.

## 6.6.8 Enumeration Type Documentation

6.6.8.1 `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.

## 6.7 HAL Driver

### 6.7.1 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.

## Data Structures

- struct `stm32_tim_t`  
*STM32 TIM registers block.*

## Modules

- STM32F100 HAL Support
- STM32F103 HAL Support
- STM32F105/F107 HAL Support

## Functions

- void `halInit(void)`  
*HAL initialization.*
- void `hal_lld_init(void)`  
*Low level HAL driver initialization.*
- void `stm32_clock_init(void)`  
*STM32 clocks and PLL initialization.*

## Time conversion utilities for the realtime counter

- #define `S2RTT(sec)` (`halGetCounterFrequency() * (sec)`)  
*Seconds to realtime ticks.*
- #define `MS2RTT(msec)` (((`halGetCounterFrequency()` + 999UL) / 1000UL) \* (msec))  
*Milliseconds to realtime ticks.*
- #define `US2RTT(usec)`  
*Microseconds to realtime ticks.*

## Macro Functions

- #define `halGetCounterValue()` `hal_lld_get_counter_value()`  
*Returns the current value of the system free running counter.*
- #define `halGetCounterFrequency()` `hal_lld_get_counter_frequency()`  
*Realtime counter frequency.*
- #define `hallsCounterWithin(start, end)`  
*Realtime window test.*

## Internal clock sources

- #define `STM32_HSICLK` 8000000
- #define `STM32_LSICLK` 40000

### PWR\_CR register bits definitions

- #define STM32\_PLS\_MASK (7 << 5)
- #define STM32\_PLS\_LEV0 (0 << 5)
- #define STM32\_PLS\_LEV1 (1 << 5)
- #define STM32\_PLS\_LEV2 (2 << 5)
- #define STM32\_PLS\_LEV3 (3 << 5)
- #define STM32\_PLS\_LEV4 (4 << 5)
- #define STM32\_PLS\_LEV5 (5 << 5)
- #define STM32\_PLS\_LEV6 (6 << 5)
- #define STM32\_PLS\_LEV7 (7 << 5)

### Configuration options

- #define STM32\_NO\_INIT FALSE  
*Disables the PWR/RCC initialization in the HAL.*
- #define STM32\_PVD\_ENABLE FALSE  
*Enables or disables the programmable voltage detector.*
- #define STM32\_PLS STM32\_PLS\_LEV0  
*Sets voltage level for programmable voltage detector.*
- #define STM32\_HSI\_ENABLED TRUE  
*Enables or disables the HSI clock source.*
- #define STM32\_LSI\_ENABLED FALSE  
*Enables or disables the LSI clock source.*
- #define STM32\_HSE\_ENABLED TRUE  
*Enables or disables the HSE clock source.*
- #define STM32\_LSE\_ENABLED FALSE  
*Enables or disables the LSE clock source.*

### Platform identification

- #define PLATFORM\_NAME "STM32"

### TIM units references

- #define STM32\_TIM1 ((stm32\_tim\_t \*)TIM1\_BASE)
- #define STM32\_TIM2 ((stm32\_tim\_t \*)TIM2\_BASE)
- #define STM32\_TIM3 ((stm32\_tim\_t \*)TIM3\_BASE)
- #define STM32\_TIM4 ((stm32\_tim\_t \*)TIM4\_BASE)
- #define STM32\_TIM5 ((stm32\_tim\_t \*)TIM5\_BASE)
- #define STM32\_TIM6 ((stm32\_tim\_t \*)TIM6\_BASE)
- #define STM32\_TIM7 ((stm32\_tim\_t \*)TIM7\_BASE)
- #define STM32\_TIM8 ((stm32\_tim\_t \*)TIM8\_BASE)
- #define STM32\_TIM9 ((stm32\_tim\_t \*)TIM9\_BASE)
- #define STM32\_TIM10 ((stm32\_tim\_t \*)TIM10\_BASE)
- #define STM32\_TIM11 ((stm32\_tim\_t \*)TIM11\_BASE)
- #define STM32\_TIM12 ((stm32\_tim\_t \*)TIM12\_BASE)
- #define STM32\_TIM13 ((stm32\_tim\_t \*)TIM13\_BASE)
- #define STM32\_TIM14 ((stm32\_tim\_t \*)TIM14\_BASE)

## Defines

- `#define halPolledDelay(ticks)`  
*Polled delay.*
- `#define HAL_IMPLEMENTES_COUNTERS TRUE`  
*Defines the support for realtime counters in the HAL.*
- `#define hal_lld_get_counter_value() DWT_CYCCNT`  
*Returns the current value of the system free running counter.*
- `#define hal_lld_get_counter_frequency() STM32_HCLK`  
*Realtime counter frequency.*

## Typedefs

- `typedef uint32_t halclock_t`  
*Type representing a system clock frequency.*
- `typedef uint32_t halrtcnt_t`  
*Type of the realtime free counter value.*

## 6.7.2 Function Documentation

### 6.7.2.1 void halInit( void )

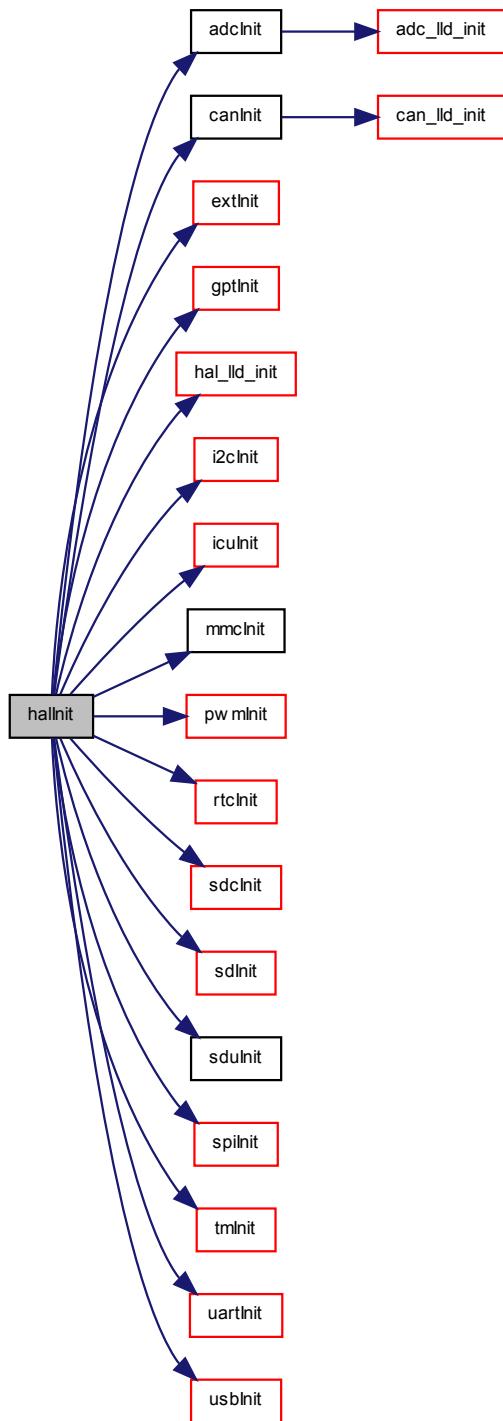
HAL initialization.

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

#### Function Class:

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

Here is the call graph for this function:



### 6.7.2.2 void hal\_lld\_init( void )

Low level HAL driver initialization.

**Function Class:**

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

Here is the call graph for this function:

**6.7.2.3 void stm32\_clock\_init( void )**

STM32 clocks and PLL initialization.

**Note**

All the involved constants come from the file `board.h`.  
This function should be invoked just after the system reset.

**Function Class:**

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

**6.7.3 Define Documentation****6.7.3.1 #define S2RTT( sec ) (halGetCounterFrequency() \* (sec))**

Seconds to realtime ticks.

Converts from seconds to realtime ticks number.

**Note**

The result is rounded upward to the next tick boundary.

**Parameters**

in                   sec   number of seconds

**Returns**

The number of ticks.

**Function Class:**

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

**6.7.3.2 #define MS2RTT( msec ) (((halGetCounterFrequency() + 999UL) / 1000UL) \* (msec))**

Milliseconds to realtime ticks.

Converts from milliseconds to realtime ticks number.

**Note**

The result is rounded upward to the next tick boundary.

**Parameters**

in *msec* number of milliseconds

**Returns**

The number of ticks.

**Function Class:**

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

**6.7.3.3 #define US2RTT( *usec* )****Value:**

```
((halGetCounterFrequency() + 999999UL) / 1000000UL) * \
(usec)
```

Microseconds to realtime ticks.

Converts from microseconds to realtime ticks number.

**Note**

The result is rounded upward to the next tick boundary.

**Parameters**

in *usec* number of microseconds

**Returns**

The number of ticks.

**Function Class:**

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

**6.7.3.4 #define halGetCounterValue( ) hal\_lld\_get\_counter\_value()**

Returns the current value of the system free running counter.

**Note**

This is an optional service that could not be implemented in all HAL implementations.

**Returns**

The value of the system free running counter of type halrtcnt\_t.

**Function Class:**

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

**6.7.3.5 #define halGetCounterFrequency( ) hal\_lld\_get\_counter\_frequency()**

Realtime counter frequency.

**Note**

This is an optional service that could not be implemented in all HAL implementations.

**Returns**

The realtime counter frequency of type halclock\_t.

**Function Class:**

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

**6.7.3.6 #define hallsCounterWithin( start, end )****Value:**

```
(end > start ? (halGetCounterValue() >= start) &&
           (halGetCounterValue() < end) :
           (halGetCounterValue() >= start) ||
           (halGetCounterValue() < end)) \\\
```

Realtime window test.

This function verifies if the current realtime counter value lies within the specified range or not. The test takes care of the realtime counter wrapping to zero on overflow.

**Note**

When start==end then the function returns always true because the whole time range is specified.

**Example 1**

Example of a guarded loop using the realtime counter. The loop implements a timeout after one second.

```
halrtcnt_t start = halGetCounterValue();
halrtcnt_t timeout = start + S2RTT(1);
while (my_condition) {
    if (!halIsCounterWithin(start, timeout))
        return TIMEOUT;
    // Do something.
}
// Continue.
```

**Example 2**

Example of a loop that lasts exactly 50 microseconds.

```
halrtcnt_t start = halGetCounterValue();
halrtcnt_t timeout = start + US2RTT(50);
while (halIsCounterWithin(start, timeout)) {
    // Do something.
}
// Continue.
```

**Parameters**

in	<i>start</i> the start of the time window (inclusive)
in	<i>end</i> the end of the time window (non inclusive)

**Return values**

*TRUE* current time within the specified time window.

*FALSE* current time not within the specified time window.

**Function Class:**

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

**6.7.3.7 #define halPolledDelay( ticks )****Value:**

```
{                                     \
    halrtcnt_t start = halGetCounterValue();           \
    halrtcnt_t timeout = start + (ticks);             \
    while (halIsCounterWithin(start, timeout))         \
    ;                                                 \
}
```

Polled delay.

**Note**

The real delays is always few cycles in excess of the specified value.

**Parameters**

in                   *ticks*    number of ticks

**Function Class:**

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

**6.7.3.8 #define HAL\_IMPLEMENT\_COUNTERS TRUE**

Defines the support for realtime counters in the HAL.

**6.7.3.9 #define STM32\_HSICLK 8000000**

High speed internal clock.

**6.7.3.10 #define STM32\_LSIGCLK 40000**

Low speed internal clock.

**6.7.3.11 #define STM32\_PLS\_MASK (7 << 5)**

PLS bits mask.

**6.7.3.12 #define STM32\_PLSLEV0 (0 << 5)**

PVD level 0.

**6.7.3.13 #define STM32\_PLSLEV1 (1 << 5)**

PVD level 1.

**6.7.3.14 #define STM32\_PLSLEV2 (2 << 5)**

PVD level 2.

6.7.3.15 `#define STM32_PLS_lev3 (3 << 5)`

PVD level 3.

6.7.3.16 `#define STM32_PLS_lev4 (4 << 5)`

PVD level 4.

6.7.3.17 `#define STM32_PLS_lev5 (5 << 5)`

PVD level 5.

6.7.3.18 `#define STM32_PLS_lev6 (6 << 5)`

PVD level 6.

6.7.3.19 `#define STM32_PLS_lev7 (7 << 5)`

PVD level 7.

6.7.3.20 `#define STM32_NO_INIT FALSE`

Disables the PWR/RCC initialization in the HAL.

6.7.3.21 `#define STM32_PVD_ENABLE FALSE`

Enables or disables the programmable voltage detector.

6.7.3.22 `#define STM32_PLS STM32_PLS_lev0`

Sets voltage level for programmable voltage detector.

6.7.3.23 `#define STM32_HSI_ENABLED TRUE`

Enables or disables the HSI clock source.

6.7.3.24 `#define STM32_LSI_ENABLED FALSE`

Enables or disables the LSI clock source.

6.7.3.25 `#define STM32_HSE_ENABLED TRUE`

Enables or disables the HSE clock source.

6.7.3.26 `#define STM32_LSE_ENABLED FALSE`

Enables or disables the LSE clock source.

6.7.3.27 #define hal\_lld\_get\_counter\_value( ) DWT\_CYCCNT

Returns the current value of the system free running counter.

**Note**

This service is implemented by returning the content of the DWT\_CYCCNT register.

**Returns**

The value of the system free running counter of type halrcnt\_t.

**Function Class:**

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

6.7.3.28 #define hal\_lld\_get\_counter\_frequency( ) STM32\_HCLK

Realtime counter frequency.

**Note**

The DWT\_CYCCNT register is incremented directly by the system clock so this function returns STM32\_HCLK.

**Returns**

The realtime counter frequency of type halclock\_t.

**Function Class:**

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

## 6.7.4 Typedef Documentation

6.7.4.1 `typedef uint32_t halclock_t`

Type representing a system clock frequency.

6.7.4.2 `typedef uint32_t halrcnt_t`

Type of the realtime free counter value.

## 6.8 I2C Driver

### 6.8.1 Detailed Description

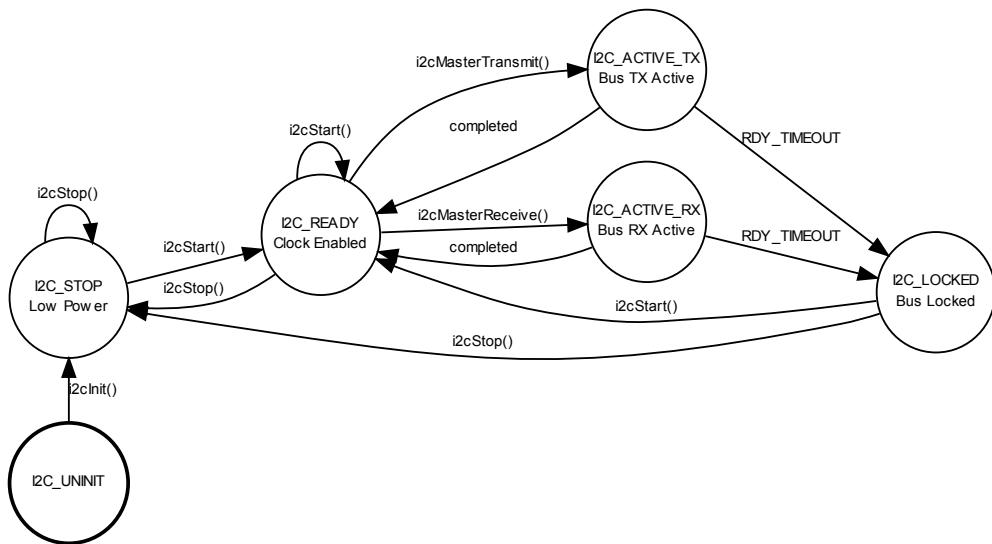
Generic I2C Driver. This module implements a generic I2C (Inter-Integrated Circuit) driver.

**Precondition**

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

### 6.8.2 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.

### Data Structures

- struct [I2CConfig](#)  
*Driver configuration structure.*
- struct [I2CDriver](#)  
*Structure representing an I2C driver.*

### Functions

- void [i2cInit](#) (void)  
*I2C Driver initialization.*
- void [i2cObjectInit](#) ([I2CDriver](#) \*i2cp)  
*Initializes the standard part of a [I2CDriver](#) structure.*
- void [i2cStart](#) ([I2CDriver](#) \*i2cp, const [I2CConfig](#) \*config)  
*Configures and activates the I2C peripheral.*
- void [i2cStop](#) ([I2CDriver](#) \*i2cp)  
*Deactivates the I2C peripheral.*
- [i2cflags\\_t](#) [i2cGetErrors](#) ([I2CDriver](#) \*i2cp)  
*Returns the errors mask associated to the previous operation.*
- msg\_t [i2cMasterTimeout](#) ([I2CDriver](#) \*i2cp, [i2caddr\\_t](#) addr, const uint8\_t \*txbuf, size\_t txbytes, uint8\_t \*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.*
- `CH_IRQ_HANDLER` (I2C1\_EV\_IRQHandler)
  - I2C1 event interrupt handler.*
- `CH_IRQ_HANDLER` (I2C1\_ER\_IRQHandler)
  - I2C1 error interrupt handler.*
- `CH_IRQ_HANDLER` (I2C2\_EV\_IRQHandler)
  - I2C2 event interrupt handler.*
- `CH_IRQ_HANDLER` (I2C2\_ER\_IRQHandler)
  - I2C2 error interrupt handler.*
- `CH_IRQ_HANDLER` (I2C3\_EV\_IRQHandler)
  - I2C3 event interrupt handler.*
- `CH_IRQ_HANDLER` (I2C3\_ER\_IRQHandler)
  - I2C3 error interrupt handler.*
- 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.*
- I2CDriver I2CD2
  - I2C2 driver identifier.*
- I2CDriver I2CD3
  - I2C3 driver identifier.*

## I2C bus error conditions

- #define `I2CD_NO_ERROR` 0x00
  - No error.*
- #define `I2CD_BUS_ERROR` 0x01
  - Bus Error.*
- #define `I2CD_ARBITRATION_LOST` 0x02
  - Arbitration Lost (master mode).*
- #define `I2CD_ACK_FAILURE` 0x04

- `#define I2CD_OVERRUN 0x08`  
*Overrun/Underrun.*
- `#define I2CD_PEC_ERROR 0x10`  
*PEC Error in reception.*
- `#define I2CD_TIMEOUT 0x20`  
*Hardware timeout.*
- `#define I2CD_SMB_ALERT 0x40`  
*SMBus Alert.*

## Configuration options

- `#define STM32_I2C_USE_I2C1 FALSE`  
*I2C1 driver enable switch.*
- `#define STM32_I2C_USE_I2C2 FALSE`  
*I2C2 driver enable switch.*
- `#define STM32_I2C_USE_I2C3 FALSE`  
*I2C3 driver enable switch.*
- `#define STM32_I2C_I2C1_IRQ_PRIORITY 10`  
*I2C1 interrupt priority level setting.*
- `#define STM32_I2C_I2C2_IRQ_PRIORITY 10`  
*I2C2 interrupt priority level setting.*
- `#define STM32_I2C_I2C3_IRQ_PRIORITY 10`  
*I2C3 interrupt priority level setting.*
- `#define STM32_I2C_I2C1_DMA_PRIORITY 1`  
*I2C1 DMA priority (0..3|lowest..highest).*
- `#define STM32_I2C_I2C2_DMA_PRIORITY 1`  
*I2C2 DMA priority (0..3|lowest..highest).*
- `#define STM32_I2C_I2C3_DMA_PRIORITY 1`  
*I2C3 DMA priority (0..3|lowest..highest).*
- `#define STM32_I2C_DMA_ERROR_HOOK(i2cp) chSysHalt()`  
*I2C DMA error hook.*
- `#define STM32_I2C_I2C1_RX_DMA_STREAM STM32_DMA_STREAM_ID(1, 0)`  
*DMA stream used for I2C1 RX operations.*
- `#define STM32_I2C_I2C1_TX_DMA_STREAM STM32_DMA_STREAM_ID(1, 6)`  
*DMA stream used for I2C1 TX operations.*
- `#define STM32_I2C_I2C2_RX_DMA_STREAM STM32_DMA_STREAM_ID(1, 2)`  
*DMA stream used for I2C2 RX operations.*
- `#define STM32_I2C_I2C2_TX_DMA_STREAM STM32_DMA_STREAM_ID(1, 7)`  
*DMA stream used for I2C2 TX operations.*
- `#define STM32_I2C_I2C3_RX_DMA_STREAM STM32_DMA_STREAM_ID(1, 2)`  
*DMA stream used for I2C3 RX operations.*
- `#define STM32_I2C_I2C3_TX_DMA_STREAM STM32_DMA_STREAM_ID(1, 4)`  
*DMA stream used for I2C3 TX operations.*

## Defines

- `#define I2C_USE_MUTUAL_EXCLUSION TRUE`  
*Enables the mutual exclusion APIs on the I2C bus.*
- `#define i2cMasterTransmit(i2cp, addr, txbuf, txbytes, rxbuf, rxbytes)`  
*Wrap i2cMasterTransmit function with TIME\_INFINITE timeout.*
- `#define i2cMasterReceive(i2cp, addr, rxbuf, rxbytes) (i2cMasterReceiveTimeout(i2cp, addr, rxbuf, rxbytes, TIME_INFINITE))`  
*Wrap i2cMasterReceive function with TIME\_INFINITE timeout.*
- `#define wakeup_isr(i2cp, msg)`  
*Wakes up the waiting thread.*
- `#define I2C_CLK_FREQ ((STM32_PCLK1) / 1000000)`  
*Peripheral clock frequency.*
- `#define STM32_DMA_REQUIRED`  
*error checks*
- `#define i2c_lld_get_errors(i2cp) ((i2cp)->errors)`  
*Get errors from I2C driver.*

## Typedefs

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

## Enumerations

- `enum i2cstate_t {`  
 `I2C_UNINIT = 0, I2C_STOP = 1, I2C_READY = 2, I2C_ACTIVE_TX = 3,`  
 `I2C_ACTIVE_RX = 4 }`  
*Driver state machine possible states.*
- `enum i2copmode_t`  
*Supported modes for the I2C bus.*
- `enum i2cdutycycle_t`  
*Supported duty cycle modes for the I2C bus.*

### 6.8.3 Function Documentation

#### 6.8.3.1 void i2cInit( void )

I2C Driver initialization.

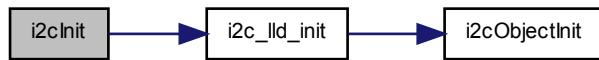
#### Note

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:



#### 6.8.3.2 void i2cObjectInit ( I2CDriver \* *i2cp* )

Initializes the standard part of a `I2CDriver` structure.

##### Parameters

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

##### Function Class:

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

#### 6.8.3.3 void i2cStart ( I2CDriver \* *i2cp*, const I2CConfig \* *config* )

Configures and activates the I2C peripheral.

##### Parameters

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



#### 6.8.3.4 void i2cStop ( I2CDriver \* *i2cp* )

Deactivates the I2C peripheral.

##### Parameters

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

**Function Class:**

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

Here is the call graph for this function:

**6.8.3.5 i2cflags\_t i2cGetErrors ( I2CDriver \* i2cp )**

Returns the errors mask associated to the previous operation.

**Parameters**

in	<i>i2cp</i> pointer to the <a href="#">I2CDriver</a> 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.

**6.8.3.6 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 <a href="#">I2CDriver</a> 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"> <li>• <i>TIME_INFINITE</i> no timeout.</li> </ul>

**Returns**

The operation status.

**Return values**

*RDY\_OK* if the function succeeded.  
*RDY\_RESET* if one or more I2C errors occurred, the errors can be retrieved using [i2cGetErrors\(\)](#).  
*RDY\_TIMEOUT* if a timeout occurred before operation end.

**Function Class:**

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

Here is the call graph for this function:



### 6.8.3.7 msg\_t i2cMasterReceiveTimeout ( I2CDriver \* i2cp, i2caddr\_t addr, uint8\_t \* rdbuf, size\_t rxbytes, systime\_t timeout )

Receives data from the I2C bus.

**Parameters**

in	<i>i2cp</i>	pointer to the <a href="#">I2CDriver</a> object
in	<i>addr</i>	slave device address (7 bits) without R/W bit
out	<i>rdbuf</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"> <li>• <i>TIME_INFINITE</i> no timeout.</li> </ul>

**Returns**

The operation status.

**Return values**

*RDY\_OK* if the function succeeded.  
*RDY\_RESET* if one or more I2C errors occurred, the errors can be retrieved using [i2cGetErrors\(\)](#).  
*RDY\_TIMEOUT* if a timeout occurred before operation end.

**Function Class:**

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

Here is the call graph for this function:



#### 6.8.3.8 void i2cAcquireBus ( I2CDriver \* *i2cp* )

Gains exclusive access to the I2C 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 `I2C_USE_MUTUAL_EXCLUSION` must be enabled.

##### Parameters

in *i2cp* pointer to the `I2CDriver` object

##### Function Class:

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

#### 6.8.3.9 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 *i2cp* pointer to the `I2CDriver` object

##### Function Class:

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

#### 6.8.3.10 CH\_IRQ\_HANDLER ( I2C1\_EV\_IRQHandler )

I2C1 event interrupt handler.

##### Function Class:

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

**6.8.3.11 CH\_IRQ\_HANDLER ( I2C1\_ER\_IRQHandler )**

I2C1 error interrupt handler.

**6.8.3.12 CH\_IRQ\_HANDLER ( I2C2\_EV\_IRQHandler )**

I2C2 event interrupt handler.

**Function Class:**

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

**6.8.3.13 CH\_IRQ\_HANDLER ( I2C2\_ER\_IRQHandler )**

I2C2 error interrupt handler.

**Function Class:**

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

**6.8.3.14 CH\_IRQ\_HANDLER ( I2C3\_EV\_IRQHandler )**

I2C3 event interrupt handler.

**Function Class:**

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

**6.8.3.15 CH\_IRQ\_HANDLER ( I2C3\_ER\_IRQHandler )**

I2C3 error interrupt handler.

**Function Class:**

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

**6.8.3.16 void i2c\_lld\_init ( void )**

Low level I2C driver initialization.

**Function Class:**

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

Here is the call graph for this function:



**6.8.3.17 void i2c\_lld\_start ( I2CDriver \* i2cp )**

Configures and activates the I2C peripheral.

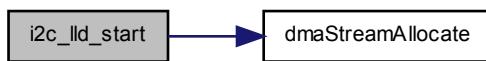
**Parameters**

in                   *i2cp*   pointer to the [I2CDriver](#) object

**Function Class:**

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

Here is the call graph for this function:

**6.8.3.18 void i2c\_lld\_stop ( I2CDriver \* i2cp )**

Deactivates the I2C peripheral.

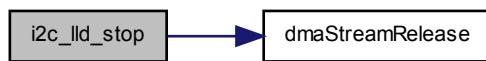
**Parameters**

in                   *i2cp*   pointer to the [I2CDriver](#) object

**Function Class:**

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

Here is the call graph for this function:

**6.8.3.19 msg\_t i2c\_lld\_master\_receive\_timeout ( I2CDriver \* i2cp, i2caddr\_t addr, uint8\_t \* rdbuf, size\_t rxbytes, systime\_t timeout )**

Receives data via the I2C bus as master.

Number of receiving bytes must be more than 1 because of stm32 hardware restrictions.

**Parameters**

in	<i>i2cp</i>	pointer to the <code>I2CDriver</code> object
in	<i>addr</i>	slave device address
out	<i>rxbuf</i>	pointer to the receive buffer
in	<i>rxbytes</i>	number of bytes to be received
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed:
• <i>TIME_INFINITE</i> no timeout.		

**Returns**

The operation status.

**Return values**

<i>RDY_OK</i>	if the function succeeded.
<i>RDY_RESET</i>	if one or more I2C errors occurred, the errors can be retrieved using <code>i2cGetErrors()</code> .
<i>RDY_TIMEOUT</i>	if a timeout occurred before operation end. <b>After a timeout the driver must be stopped and restarted because the bus is in an uncertain state.</b>

**Function Class:**

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

**6.8.3.20 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.

Number of receiving bytes must be 0 or more than 1 because of stm32 hardware restrictions.

**Parameters**

in	<i>i2cp</i>	pointer to the <code>I2CDriver</code> 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:
• <i>TIME_INFINITE</i> no timeout.		

**Returns**

The operation status.

**Return values**

<i>RDY_OK</i>	if the function succeeded.
<i>RDY_RESET</i>	if one or more I2C errors occurred, the errors can be retrieved using <code>i2cGetErrors()</code> .
<i>RDY_TIMEOUT</i>	if a timeout occurred before operation end. <b>After a timeout the driver must be stopped and restarted because the bus is in an uncertain state.</b>

**Function Class:**

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

## 6.8.4 Variable Documentation

### 6.8.4.1 I2CDriver I2CD1

I2C1 driver identifier.

### 6.8.4.2 I2CDriver I2CD2

I2C2 driver identifier.

### 6.8.4.3 I2CDriver I2CD3

I2C3 driver identifier.

## 6.8.5 Define Documentation

### 6.8.5.1 #define I2CD\_NO\_ERROR 0x00

No error.

### 6.8.5.2 #define I2CD\_BUS\_ERROR 0x01

Bus Error.

### 6.8.5.3 #define I2CD\_ARBITRATION\_LOST 0x02

Arbitration Lost (master mode).

### 6.8.5.4 #define I2CD\_ACK\_FAILURE 0x04

Acknowledge Failure.

### 6.8.5.5 #define I2CD\_OVERRUN 0x08

Overrun/Underrun.

### 6.8.5.6 #define I2CD\_PEC\_ERROR 0x10

PEC Error in reception.

### 6.8.5.7 #define I2CD\_TIMEOUT 0x20

Hardware timeout.

### 6.8.5.8 #define I2CD\_SMB\_ALERT 0x40

SMBus Alert.

#### 6.8.5.9 #define I2C\_USE\_MUTUAL\_EXCLUSION TRUE

Enables the mutual exclusion APIs on the I2C bus.

#### 6.8.5.10 #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.

#### 6.8.5.11 #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.

#### 6.8.5.12 #define wakeup\_isr( *i2cp*, *msg* )

##### Value:

```
{
    chSysLockFromIsr();
    if ((i2cp)->thread != NULL) {
        Thread *tp = (i2cp)->thread;
        (i2cp)->thread = NULL;
        tp->p_u.rdymsg = (msg);
        chSchReadyI(tp);
    }
    chSysUnlockFromIsr();
}
```

Wakes up the waiting thread.

##### Parameters

in	<i>i2cp</i>	pointer to the <a href="#">I2CDriver</a> object
in	<i>msg</i>	wakeup message

##### Function Class:

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

#### 6.8.5.13 #define I2C\_CLK\_FREQ ((STM32\_PCLK1) / 1000000)

Peripheral clock frequency.

**6.8.5.14 #define STM32\_I2C\_USE\_I2C1 FALSE**

I2C1 driver enable switch.

If set to TRUE the support for I2C1 is included.

**Note**

The default is FALSE.

**6.8.5.15 #define STM32\_I2C\_USE\_I2C2 FALSE**

I2C2 driver enable switch.

If set to TRUE the support for I2C2 is included.

**Note**

The default is FALSE.

**6.8.5.16 #define STM32\_I2C\_USE\_I2C3 FALSE**

I2C3 driver enable switch.

If set to TRUE the support for I2C3 is included.

**Note**

The default is FALSE.

**6.8.5.17 #define STM32\_I2C\_I2C1\_IRQ\_PRIORITY 10**

I2C1 interrupt priority level setting.

**6.8.5.18 #define STM32\_I2C\_I2C2\_IRQ\_PRIORITY 10**

I2C2 interrupt priority level setting.

**6.8.5.19 #define STM32\_I2C\_I2C3\_IRQ\_PRIORITY 10**

I2C3 interrupt priority level setting.

**6.8.5.20 #define STM32\_I2C\_I2C1\_DMA\_PRIORITY 1**

I2C1 DMA priority (0..3|lowest..highest).

**Note**

The priority level is used for both the TX and RX DMA streams but because of the streams ordering the RX stream has always priority over the TX stream.

6.8.5.21 #define STM32\_I2C\_I2C2\_DMA\_PRIORITY 1

I2C2 DMA priority (0..3|lowest..highest).

**Note**

The priority level is used for both the TX and RX DMA streams but because of the streams ordering the RX stream has always priority over the TX stream.

6.8.5.22 #define STM32\_I2C\_I2C3\_DMA\_PRIORITY 1

I2C3 DMA priority (0..3|lowest..highest).

**Note**

The priority level is used for both the TX and RX DMA streams but because of the streams ordering the RX stream has always priority over the TX stream.

6.8.5.23 #define STM32\_I2C\_DMA\_ERROR\_HOOK( *i2cp* ) chSysHalt()

I2C DMA error hook.

**Note**

The default action for DMA errors is a system halt because DMA error can only happen because programming errors.

6.8.5.24 #define STM32\_I2C\_I2C1\_RX\_DMA\_STREAM STM32\_DMA\_STREAM\_ID(1, 0)

DMA stream used for I2C1 RX operations.

**Note**

This option is only available on platforms with enhanced DMA.

6.8.5.25 #define STM32\_I2C\_I2C1\_TX\_DMA\_STREAM STM32\_DMA\_STREAM\_ID(1, 6)

DMA stream used for I2C1 TX operations.

**Note**

This option is only available on platforms with enhanced DMA.

6.8.5.26 #define STM32\_I2C\_I2C2\_RX\_DMA\_STREAM STM32\_DMA\_STREAM\_ID(1, 2)

DMA stream used for I2C2 RX operations.

**Note**

This option is only available on platforms with enhanced DMA.

```
6.8.5.27 #define STM32_I2C_I2C2_TX_DMA_STREAM STM32_DMA_STREAM_ID(1, 7)
```

DMA stream used for I2C2 TX operations.

**Note**

This option is only available on platforms with enhanced DMA.

```
6.8.5.28 #define STM32_I2C_I2C3_RX_DMA_STREAM STM32_DMA_STREAM_ID(1, 2)
```

DMA stream used for I2C3 RX operations.

**Note**

This option is only available on platforms with enhanced DMA.

```
6.8.5.29 #define STM32_I2C_I2C3_TX_DMA_STREAM STM32_DMA_STREAM_ID(1, 4)
```

DMA stream used for I2C3 TX operations.

**Note**

This option is only available on platforms with enhanced DMA.

```
6.8.5.30 #define STM32_DMA_REQUIRED
```

error checks

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

Get errors from I2C driver.

**Parameters**

in *i2cp* pointer to the [I2CDriver](#) object

**Function Class:**

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

## 6.8.6 Typedef Documentation

```
6.8.6.1 typedef uint16_t i2caddr_t
```

Type representing I2C address.

```
6.8.6.2 typedef uint32_t i2cflags_t
```

I2C Driver condition flags type.

```
6.8.6.3 typedef struct I2CDriver I2CDriver
```

Type of a structure representing an I2C driver.

### 6.8.7 Enumeration Type Documentation

#### 6.8.7.1 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.

#### 6.8.7.2 enum i2copmode\_t

Supported modes for the I2C bus.

#### 6.8.7.3 enum i2cdutycycle\_t

Supported duty cycle modes for the I2C bus.

## 6.9 ICU Driver

### 6.9.1 Detailed Description

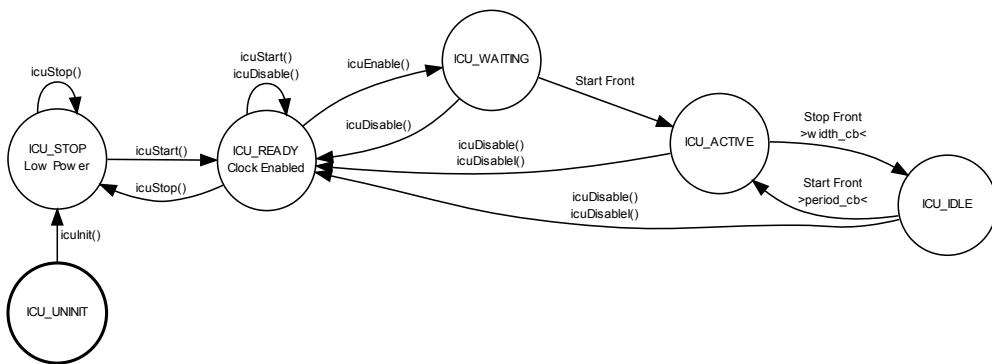
Generic ICU Driver. This module implements a generic ICU (Input Capture Unit) driver.

#### Precondition

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

### 6.9.2 Driver State Machine

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



### 6.9.3 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.

After the activation the ICU unit can be in one of the following states at any time:

- **ICU\_WAITING**, waiting the first start signal.
- **ICU\_ACTIVE**, after a start signal.
- **ICU\_IDLE**, after a stop signal.

Callbacks are invoked when start or stop signals occur.

## Data Structures

- struct **ICUConfig**  
*Driver configuration structure.*
- struct **ICUDriver**  
*Structure representing an ICU driver.*

## Functions

- void **icuInit** (void)  
*ICU Driver initialization.*
- void **icuObjectInit** (**ICUDriver** \*icup)

- **void icuStart (ICUDriver \*icup, const ICUConfig \*config)**  
*Configures and activates the ICU peripheral.*
- **void icuStop (ICUDriver \*icup)**  
*Deactivates the ICU peripheral.*
- **void icuEnable (ICUDriver \*icup)**  
*Enables the input capture.*
- **void icuDisable (ICUDriver \*icup)**  
*Disables the input capture.*
- **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\_enable (ICUDriver \*icup)**  
*Enables the input capture.*
- **void icu\_lld\_disable (ICUDriver \*icup)**  
*Disables the input capture.*

## Variables

- **ICUDriver ICUD1**  
*ICUD1 driver identifier.*
- **ICUDriver ICUD2**  
*ICUD2 driver identifier.*
- **ICUDriver ICUD3**  
*ICUD3 driver identifier.*
- **ICUDriver ICUD4**  
*ICUD4 driver identifier.*
- **ICUDriver ICUD5**  
*ICUD5 driver identifier.*
- **ICUDriver ICUD8**  
*ICUD8 driver identifier.*

## Macro Functions

- **#define icuEnable(icup) icu\_lld\_enable(icup)**  
*Enables the input capture.*
- **#define icuDisable(icup) icu\_lld\_disable(icup)**  
*Disables the input capture.*
- **#define icuGetWidth(icup) icu\_lld\_get\_width(icup)**  
*Returns the width of the latest pulse.*
- **#define icuGetPeriod(icup) icu\_lld\_get\_period(icup)**  
*Returns the width of the latest cycle.*

## Low Level driver helper macros

- **#define \_icu\_isr\_invoke\_width\_cb(icup)**  
*Common ISR code, ICU width event.*
- **#define \_icu\_isr\_invoke\_period\_cb(icup)**  
*Common ISR code, ICU period event.*

## Configuration options

- `#define STM32_ICU_USE_TIM1 TRUE`  
*ICUD1 driver enable switch.*
- `#define STM32_ICU_USE_TIM2 TRUE`  
*ICUD2 driver enable switch.*
- `#define STM32_ICU_USE_TIM3 TRUE`  
*ICUD3 driver enable switch.*
- `#define STM32_ICU_USE_TIM4 TRUE`  
*ICUD4 driver enable switch.*
- `#define STM32_ICU_USE_TIM5 TRUE`  
*ICUD5 driver enable switch.*
- `#define STM32_ICU_USE_TIM8 TRUE`  
*ICUD8 driver enable switch.*
- `#define STM32_ICU_TIM1_IRQ_PRIORITY 7`  
*ICUD1 interrupt priority level setting.*
- `#define STM32_ICU_TIM2_IRQ_PRIORITY 7`  
*ICUD2 interrupt priority level setting.*
- `#define STM32_ICU_TIM3_IRQ_PRIORITY 7`  
*ICUD3 interrupt priority level setting.*
- `#define STM32_ICU_TIM4_IRQ_PRIORITY 7`  
*ICUD4 interrupt priority level setting.*
- `#define STM32_ICU_TIM5_IRQ_PRIORITY 7`  
*ICUD5 interrupt priority level setting.*
- `#define STM32_ICU_TIM8_IRQ_PRIORITY 7`  
*ICUD8 interrupt priority level setting.*

## Defines

- `#define icu_lld_get_width(icup) ((icup)->tim->CCR[1] + 1)`  
*Returns the width of the latest pulse.*
- `#define icu_lld_get_period(icup) ((icup)->tim->CCR[0] + 1)`  
*Returns the width of the latest cycle.*

## 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 uint16_t icucont_t`  
*ICU counter type.*

## Enumerations

- `enum icustate_t {`  
 `ICU_UNINIT = 0, ICU_STOP = 1, ICU_READY = 2, ICU_WAITING = 3,`  
 `ICU_ACTIVE = 4, ICU_IDLE = 5 }`  
*Driver state machine possible states.*
- `enum icumode_t { ICU_INPUT_ACTIVE_HIGH = 0, ICU_INPUT_ACTIVE_LOW = 1 }`  
*ICU driver mode.*

## 6.9.4 Function Documentation

### 6.9.4.1 void icuInit( void )

ICU Driver initialization.

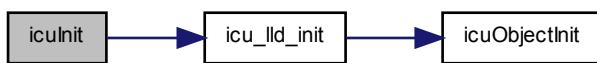
#### Note

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:



### 6.9.4.2 void icuObjectInit( ICUDriver \* icup )

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

#### Parameters

out                    *icup*    pointer to the [ICUDriver](#) object

#### Function Class:

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

### 6.9.4.3 void icuStart( ICUDriver \* icup, const ICUConfig \* config )

Configures and activates the ICU peripheral.

#### Parameters

in	<i>icup</i> pointer to the <a href="#">ICUDriver</a> object
in	<i>config</i> pointer to the <a href="#">ICUConfig</a> 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:



#### 6.9.4.4 void icuStop ( ICUDriver \* *icup* )

Deactivates the ICU peripheral.

##### Parameters

in *icup* pointer to the [ICUDriver](#) object

##### Function Class:

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

Here is the call graph for this function:



#### 6.9.4.5 void icuEnable ( ICUDriver \* *icup* )

Enables the input capture.

##### Parameters

in *icup* pointer to the [ICUDriver](#) object

##### Function Class:

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

Here is the call graph for this function:



#### 6.9.4.6 void icuDisable ( ICUDriver \* *icup* )

Disables the input capture.

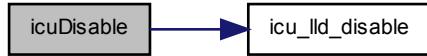
##### Parameters

in *icup* pointer to the [ICUDriver](#) object

##### Function Class:

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

Here is the call graph for this function:



#### 6.9.4.7 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:



**6.9.4.8 void icu\_lld\_start ( ICUDriver \* *icup* )**

Configures and activates the ICU peripheral.

**Parameters**

in *icup* pointer to the [ICUDriver](#) object

**Function Class:**

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

**6.9.4.9 void icu\_lld\_stop ( ICUDriver \* *icup* )**

Deactivates the ICU peripheral.

**Parameters**

in *icup* pointer to the [ICUDriver](#) object

**Function Class:**

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

**6.9.4.10 void icu\_lld\_enable ( ICUDriver \* *icup* )**

Enables the input capture.

**Parameters**

in *icup* pointer to the [ICUDriver](#) object

**Function Class:**

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

**6.9.4.11 void icu\_lld\_disable ( ICUDriver \* *icup* )**

Disables the input capture.

**Parameters**

in *icup* pointer to the [ICUDriver](#) object

**Function Class:**

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

## 6.9.5 Variable Documentation

### 6.9.5.1 ICUDriver ICUD1

ICUD1 driver identifier.

**Note**

The driver ICUD1 allocates the complex timer TIM1 when enabled.

### 6.9.5.2 ICUDriver ICUD2

ICUD2 driver identifier.

#### Note

The driver ICUD1 allocates the timer TIM2 when enabled.

### 6.9.5.3 ICUDriver ICUD3

ICUD3 driver identifier.

#### Note

The driver ICUD1 allocates the timer TIM3 when enabled.

### 6.9.5.4 ICUDriver ICUD4

ICUD4 driver identifier.

#### Note

The driver ICUD4 allocates the timer TIM4 when enabled.

### 6.9.5.5 ICUDriver ICUD5

ICUD5 driver identifier.

#### Note

The driver ICUD5 allocates the timer TIM5 when enabled.

### 6.9.5.6 ICUDriver ICUD8

ICUD8 driver identifier.

#### Note

The driver ICUD8 allocates the timer TIM8 when enabled.

## 6.9.6 Define Documentation

### 6.9.6.1 #define icuEnable( *icup* ) icu\_llid\_enable(*icup*)

Enables the input capture.

#### Parameters

in                   *icup*   pointer to the [ICUDriver](#) object

#### Function Class:

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

**6.9.6.2 #define icuDisable( *icup* ) icu\_lld\_disable(*icup*)**

Disables the input capture.

**Parameters**

in *icup* pointer to the [ICUDriver](#) object

**Function Class:**

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

**6.9.6.3 #define icuGetWidth( *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.

**Parameters**

in *icup* pointer to the [ICUDriver](#) object

**Returns**

The number of 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.

**6.9.6.4 #define icuGetPeriod( *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.

**Parameters**

in *icup* pointer to the [ICUDriver](#) object

**Returns**

The number of 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.

**6.9.6.5 #define \_icu\_isr\_invoke\_width\_cb( *icup* )**

**Value:**

```
{
    (icup)->state = ICU_IDLE;
    (icup)->config->width_cb(icup);
}
```

Common ISR code, ICU width event.

**Parameters**

in *icup* pointer to the [ICUDriver](#) object

**Function Class:**

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

**6.9.6.6 #define \_icu\_isr\_invoke\_period\_cb( *icup* )****Value:**

```
{           \
    icustate_t previous_state = (icup)->state;           \
    (icup)->state = ICU_ACTIVE;           \
    if (previous_state != ICU_WAITING)           \
        (icup)->config->period_cb(icup);           \
}
```

Common ISR code, ICU period event.

**Parameters**

in *icup* pointer to the [ICUDriver](#) object

**Function Class:**

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

**6.9.6.7 #define STM32\_ICU\_USE\_TIM1 TRUE**

ICUD1 driver enable switch.

If set to TRUE the support for ICUD1 is included.

**Note**

The default is TRUE.

**6.9.6.8 #define STM32\_ICU\_USE\_TIM2 TRUE**

ICUD2 driver enable switch.

If set to TRUE the support for ICUD2 is included.

**Note**

The default is TRUE.

**6.9.6.9 #define STM32\_ICU\_USE\_TIM3 TRUE**

ICUD3 driver enable switch.

If set to TRUE the support for ICUD3 is included.

**Note**

The default is TRUE.

**6.9.6.10 #define STM32\_ICU\_USE\_TIM4 TRUE**

ICUD4 driver enable switch.

If set to TRUE the support for ICUD4 is included.

**Note**

The default is TRUE.

**6.9.6.11 #define STM32\_ICU\_USE\_TIM5 TRUE**

ICUD5 driver enable switch.

If set to TRUE the support for ICUD5 is included.

**Note**

The default is TRUE.

**6.9.6.12 #define STM32\_ICU\_USE\_TIM8 TRUE**

ICUD8 driver enable switch.

If set to TRUE the support for ICUD8 is included.

**Note**

The default is TRUE.

**6.9.6.13 #define STM32\_ICU\_TIM1\_IRQ\_PRIORITY 7**

ICUD1 interrupt priority level setting.

**6.9.6.14 #define STM32\_ICU\_TIM2\_IRQ\_PRIORITY 7**

ICUD2 interrupt priority level setting.

**6.9.6.15 #define STM32\_ICU\_TIM3\_IRQ\_PRIORITY 7**

ICUD3 interrupt priority level setting.

**6.9.6.16 #define STM32\_ICU\_TIM4\_IRQ\_PRIORITY 7**

ICUD4 interrupt priority level setting.

**6.9.6.17 #define STM32\_ICU\_TIM5\_IRQ\_PRIORITY 7**

ICUD5 interrupt priority level setting.

**6.9.6.18 #define STM32\_ICU\_TIM8\_IRQ\_PRIORITY 7**

ICUD8 interrupt priority level setting.

```
6.9.6.19 #define icu_lld_get_width( icup ) ((icup)->tim->CCR[1] + 1)
```

Returns the width of the latest pulse.

The pulse width is defined as number of ticks between the start edge and the stop edge.

#### Parameters

in *icup* pointer to the [ICUDriver](#) object

#### Returns

The number of ticks.

#### Function Class:

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

```
6.9.6.20 #define icu_lld_get_period( icup ) ((icup)->tim->CCR[0] + 1)
```

Returns the width of the latest cycle.

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

#### Parameters

in *icup* pointer to the [ICUDriver](#) object

#### Returns

The number of ticks.

#### Function Class:

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

## 6.9.7 Typedef Documentation

```
6.9.7.1 typedef struct ICUDriver ICUDriver
```

Type of a structure representing an ICU driver.

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

ICU notification callback type.

#### Parameters

in *icup* pointer to a [ICUDriver](#) object

```
6.9.7.3 typedef uint32_t icufreq_t
```

ICU frequency type.

```
6.9.7.4 typedef uint16_t icucnt_t
```

ICU counter type.

### 6.9.8 Enumeration Type Documentation

#### 6.9.8.1 enum icustate\_t

Driver state machine possible states.

##### Enumerator:

***ICU\_UNINIT*** Not initialized.

***ICU\_STOP*** Stopped.

***ICU\_READY*** Ready.

***ICU\_WAITING*** Waiting first edge.

***ICU\_ACTIVE*** Active cycle phase.

***ICU\_IDLE*** Idle cycle phase.

#### 6.9.8.2 enum icumode\_t

ICU driver mode.

##### Enumerator:

***ICU\_INPUT\_ACTIVE\_HIGH*** Trigger on rising edge.

***ICU\_INPUT\_ACTIVE\_LOW*** Trigger on falling edge.

## 6.10 MMC over SPI Driver

### 6.10.1 Detailed Description

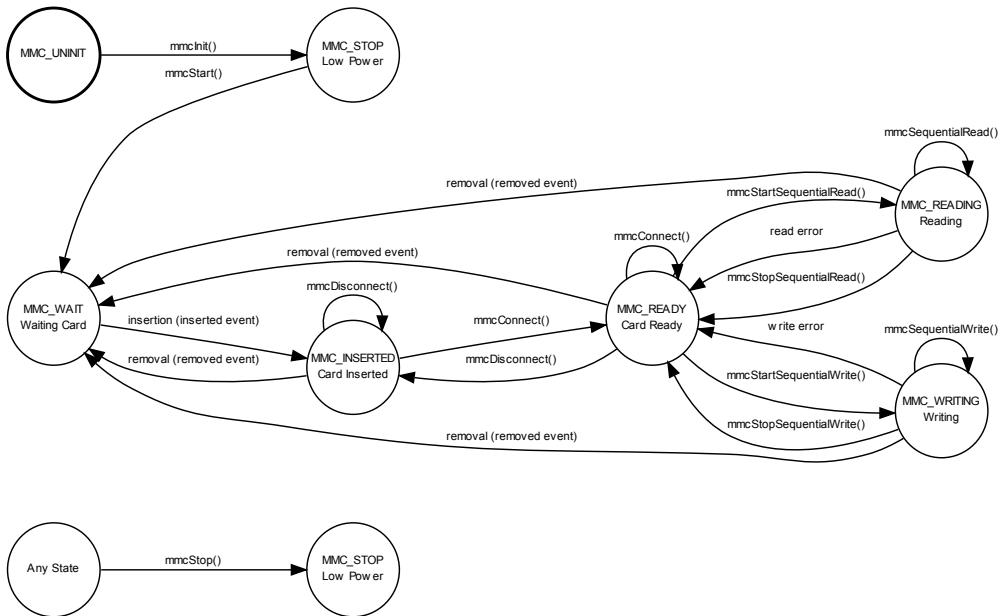
Generic MMC driver. This module implements a portable MMC/SD driver that uses a SPI driver as physical layer. Hot plugging and removal are supported through kernel events.

#### 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`.

### 6.10.2 Driver State Machine

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



## Data Structures

- struct **MMConfig**  
*Driver configuration structure.*
- struct **MMCDriver**  
*Structure representing a MMC driver.*

## Functions

- void **mmcInit** (void)  
*MMC over SPI driver initialization.*
- void **mmcObjectInit** (**MMCDriver** \*mmcp, **SPIDriver** \*spip, const **SPIConfig** \*lscfg, const **SPIConfig** \*hscfg, **mmcquery\_t** is\_protected, **mmcquery\_t** is\_inserted)  
*Initializes an instance.*
- void **mmcStart** (**MMCDriver** \*mmcp, const **MMConfig** \*config)  
*Configures and activates the MMC peripheral.*
- void **mmcStop** (**MMCDriver** \*mmcp)  
*Disables the MMC peripheral.*
- **bool\_t** **mmcConnect** (**MMCDriver** \*mmcp)  
*Performs the initialization procedure on the inserted card.*
- **bool\_t** **mmcDisconnect** (**MMCDriver** \*mmcp)  
*Brings the driver in a state safe for card removal.*
- **bool\_t** **mmcStartSequentialRead** (**MMCDriver** \*mmcp, **uint32\_t** startblk)  
*Starts a sequential read.*
- **bool\_t** **mmcSequentialRead** (**MMCDriver** \*mmcp, **uint8\_t** \*buffer)  
*Reads a block within a sequential read operation.*
- **bool\_t** **mmcStopSequentialRead** (**MMCDriver** \*mmcp)

- `bool_t mmcStartSequentialWrite (MMCDriver *mmcp, uint32_t startblk)`  
*Starts a sequential write.*
- `bool_t mmcSequentialWrite (MMCDriver *mmcp, const uint8_t *buffer)`  
*Writes a block within a sequential write operation.*
- `bool_t mmcStopSequentialWrite (MMCDriver *mmcp)`  
*Stops a sequential write gracefully.*

## MMC\_SPI configuration options

- `#define MMC_SECTOR_SIZE 512`  
*Block size for MMC transfers.*
- `#define MMC_NICE_WAITING TRUE`  
*Delays insertions.*
- `#define MMC_POLLING_INTERVAL 10`  
*Number of positive insertion queries before generating the insertion event.*
- `#define MMC_POLLING_DELAY 10`  
*Interval, in milliseconds, between insertion queries.*

## Macro Functions

- `#define mmcGetDriverState(mmc) ((mmc)->state)`  
*Returns the driver state.*
- `#define mmclsWriteProtected(mmc) ((mmc)->is_protected())`  
*Returns the write protect status.*

## Typedefs

- `typedef bool_t(* mmcquery_t )(void)`  
*Function used to query some hardware status bits.*

## Enumerations

- `enum mmcstate_t {`  
`MMC_UNINIT = 0, MMC_STOP = 1, MMC_WAIT = 2, MMC_INSERTED = 3,`  
`MMC_READY = 4, MMC_READING = 5, MMC_WRITING = 6 }`  
*Driver state machine possible states.*

### 6.10.3 Function Documentation

#### 6.10.3.1 void mmclInit ( 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.

```
6.10.3.2 void mmcObjectInit ( MMCDriver * mmcp, SPIDriver * spip, const SPIConfig * lscfg, const SPIConfig * hscfg, mmcquery_t is_protected, mmcquery_t is_inserted )
```

Initializes an instance.

#### Parameters

out	<i>mmcp</i>	pointer to the <a href="#">MMCDriver</a> object
in	<i>spip</i>	pointer to the SPI driver to be used as interface
in	<i>lscfg</i>	low speed configuration for the SPI driver
in	<i>hscfg</i>	high speed configuration for the SPI driver
in	<i>is_protected</i>	function that returns the card write protection setting
in	<i>is_inserted</i>	function that returns the card insertion sensor status

#### Function Class:

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

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

Configures and activates the MMC peripheral.

#### Parameters

in	<i>mmcp</i>	pointer to the <a href="#">MMCDriver</a> object
in	<i>config</i>	pointer to the <a href="#">MMCConfig</a> object. Must be NULL.

#### Function Class:

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

```
6.10.3.4 void mmcStop ( MMCDriver * mmcp )
```

Disables the MMC peripheral.

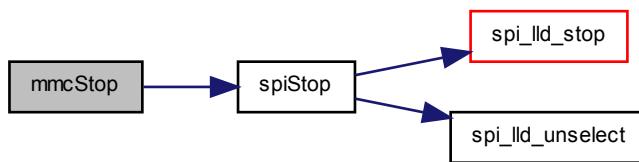
#### Parameters

in	<i>mmcp</i>	pointer to the <a href="#">MMCDriver</a> 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:



### 6.10.3.5 `bool_t mmcConnect( MMCDriver * mmcp )`

Performs the initialization procedure on the inserted card.

This function should be invoked when a card is inserted and brings the driver in the `MMC_READY` state where it is possible to perform read and write operations.

#### Note

It is possible to invoke this function from the insertion event handler.

#### Parameters

in `mmcp` pointer to the `MMCDriver` object

#### Returns

The operation status.

#### Return values

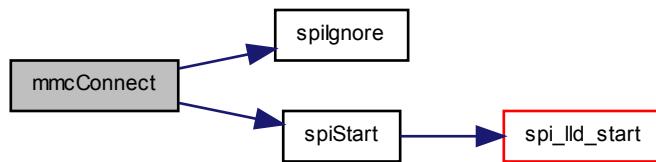
`FALSE` the operation succeeded and the driver is now in the `MMC_READY` state.

`TRUE` 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:



### 6.10.3.6 `bool_t 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

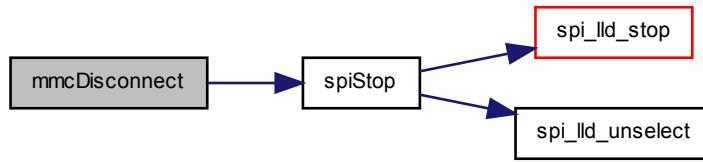
`FALSE` the operation succeeded and the driver is now in the `MMC_INSERTED` state.

`TRUE` 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:



### 6.10.3.7 `bool_t mmcStartSequentialRead( MMCDriver * mmcp, uint32_t startblk )`

Starts a sequential read.

**Parameters**

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

**Returns**

The operation status.

**Return values**

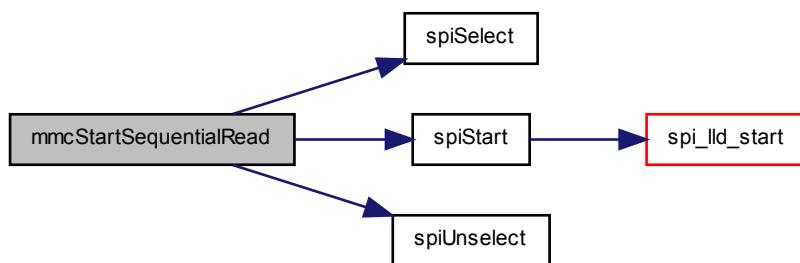
`FALSE` the operation succeeded.

`TRUE` 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:



### 6.10.3.8 `bool_t mmcSequentialRead ( MMCDriver * mmcp, uint8_t * buffer )`

Reads a block within a sequential read operation.

#### Parameters

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

#### Returns

The operation status.

#### Return values

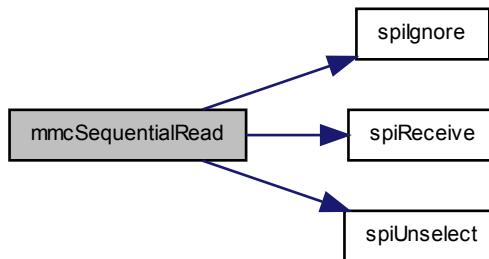
*FALSE* the operation succeeded.

*TRUE* 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:



### 6.10.3.9 `bool_t mmcStopSequentialRead ( MMCDriver * mmcp )`

Stops a sequential read gracefully.

#### Parameters

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

#### Returns

The operation status.

#### Return values

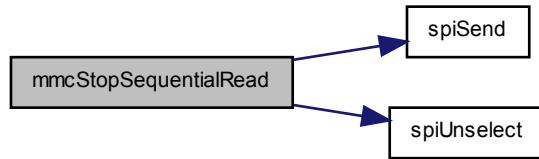
*FALSE* the operation succeeded.

*TRUE* 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:



#### 6.10.3.10 `bool_t mmcStartSequentialWrite ( MMCDriver * mmcp, uint32_t startblk )`

Starts a sequential write.

##### Parameters

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

##### Returns

The operation status.

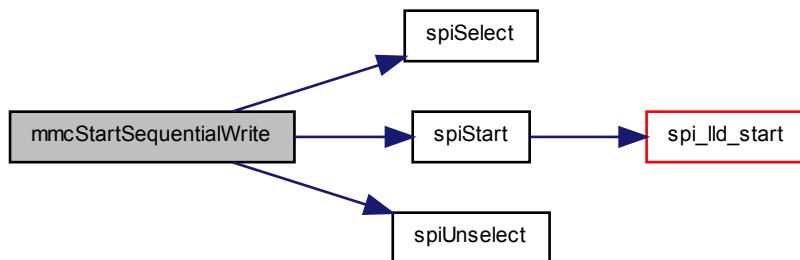
##### Return values

<code>FALSE</code>	the operation succeeded.
<code>TRUE</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:



### 6.10.3.11 `bool_t mmcSequentialWrite ( MMCDriver * mmcp, const uint8_t * buffer )`

Writes a block within a sequential write operation.

#### Parameters

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

#### Returns

The operation status.

#### Return values

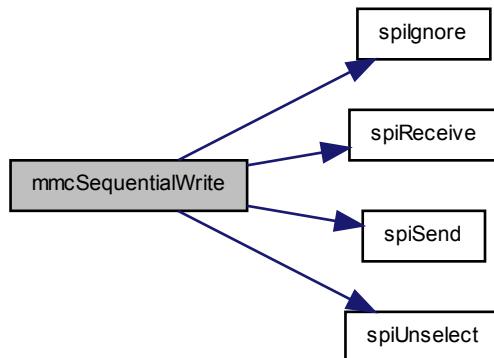
*FALSE* the operation succeeded.

*TRUE* 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:



### 6.10.3.12 `bool_t 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

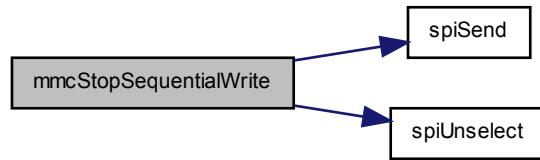
*FALSE* the operation succeeded.

*TRUE* 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:



## 6.10.4 Define Documentation

### 6.10.4.1 #define MMC\_SECTOR\_SIZE 512

Block size for MMC transfers.

### 6.10.4.2 #define MMC\_NICE\_WAITING TRUE

Delays insertions.

If enabled this option inserts delays into the MMC waiting routines releasing some extra CPU time for the threads with lower priority, this may slow down the driver a bit however. This option is recommended also if the SPI driver does not use a DMA channel and heavily loads the CPU.

### 6.10.4.3 #define MMC\_POLLING\_INTERVAL 10

Number of positive insertion queries before generating the insertion event.

### 6.10.4.4 #define MMC\_POLLING\_DELAY 10

Interval, in milliseconds, between insertion queries.

### 6.10.4.5 #define mmcGetDriverState( mmcp ) ((mmcp)->state)

Returns the driver state.

#### Parameters

in *mmcp* pointer to the [MMCDriver](#) object

#### Returns

The driver state.

**Function Class:**

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

6.10.4.6 #define mmcIsWriteProtected( *mmcp* ) ((*mmcp*)>is\_protected())

Returns the write protect status.

#### Parameters

in *mmcp* pointer to the [MMCDriver](#) object

#### Returns

The card state.

#### Return values

*FALSE* card not inserted.

*TRUE* card inserted.

#### Function Class:

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

## 6.10.5 Typedef Documentation

### 6.10.5.1 `typedef bool_t(* mmcquery_t)(void)`

Function used to query some hardware status bits.

#### Returns

The status.

## 6.10.6 Enumeration Type Documentation

### 6.10.6.1 `enum mmcstate_t`

Driver state machine possible states.

#### Enumerator:

**MMC\_UNINIT** Not initialized.

**MMC\_STOP** Stopped.

**MMC\_WAIT** Waiting card.

**MMC\_INSERTED** Card inserted.

**MMC\_READY** Card ready.

**MMC\_READING** Reading.

**MMC\_WRITING** Writing.

## 6.11 PAL Driver

### 6.11.1 Detailed Description

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

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

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

### Precondition

In order to use the PAL driver the `HAL_USE_PAL` option must be enabled in `halconf.h`.

## 6.11.2 Implementation Rules

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

### 6.11.2.1 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.

### 6.11.2.2 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.

### 6.11.2.3 Writing unused or unimplemented port bits

The behavior is not specified.

### 6.11.2.4 Reading from unused or unimplemented port bits

The behavior is not specified.

### 6.11.2.5 Reading or writing on pins associated to other functionalities

The behavior is not specified.

## Data Structures

- struct `IOBus`  
*I/O bus descriptor.*
- struct `stm32_gpio_setup_t`  
*GPIO port setup info.*
- struct `PALConfig`  
*STM32 GPIO 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.*

## Pads mode constants

- `#define PAL_MODE_RESET 0`  
*After reset state.*
- `#define PAL_MODE_UNCONNECTED 1`  
*Safe state for **unconnected** pads.*
- `#define PAL_MODE_INPUT 2`  
*Regular input high-Z pad.*
- `#define PAL_MODE_INPUT_PULLUP 3`  
*Input pad with weak pull up resistor.*
- `#define PAL_MODE_INPUT_PULLDOWN 4`  
*Input pad with weak pull down resistor.*
- `#define PAL_MODE_INPUT_ANALOG 5`  
*Analog input mode.*
- `#define PAL_MODE_OUTPUT_PUSH_PULL 6`  
*Push-pull output pad.*
- `#define PAL_MODE_OUTPUT_OPENDRAIN 7`  
*Open-drain output pad.*

## Logic level constants

- `#define PAL_LOW 0`  
*Logical low state.*
- `#define PAL_HIGH 1`  
*Logical high state.*

## Macro Functions

- `#define pallInit(config) pal_lld_init(config)`  
*PAL subsystem initialization.*
- `#define palReadPort(port) ((void)(port), 0)`  
*Reads the physical I/O port states.*
- `#define palReadLatch(port) ((void)(port), 0)`  
*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)) & 1)`  
*Reads an input pad logical state.*
- `#define palWritePad(port, pad, bit)`  
*Writes a logical state on an output pad.*
- `#define palSetPad(port, pad) palSetPort(port, PAL_PORT_BIT(pad))`  
*Sets a pad logical state to PAL\_HIGH.*
- `#define palClearPad(port, pad) palClearPort(port, PAL_PORT_BIT(pad))`  
*Clears a pad logical state to PAL\_LOW.*
- `#define palTogglePad(port, pad) palTogglePort(port, PAL_PORT_BIT(pad))`  
*Toggles a pad logical state.*
- `#define palSetPadMode(port, pad, mode) palSetGroupMode(port, PAL_PORT_BIT(pad), 0, mode)`  
*Pad mode setup.*

## STM32-specific I/O mode flags

- `#define PAL_MODE_STM32_ALTERNATE_PUSH_PULL 16`  
*STM32 specific alternate push-pull output mode.*
- `#define PAL_MODE_STM32_ALTERNATE_OPENDRAIN 17`  
*STM32 specific alternate open-drain output mode.*

## Defines

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

- Static I/O bus initializer.*
- **#define PAL\_IOPORTS\_WIDTH 16**  
*Width, in bits, of an I/O port.*
  - **#define PAL\_WHOLE\_PORT ((ioportmask\_t)0xFFFF)**  
*Whole port mask.*
  - **#define IOPORT1 GPIOA**  
*GPIO port A identifier.*
  - **#define IOPORT2 GPIOB**  
*GPIO port B identifier.*
  - **#define IOPORT3 GPIOC**  
*GPIO port C identifier.*
  - **#define IOPORT4 GPIOD**  
*GPIO port D identifier.*
  - **#define IOPORT5 GPIOE**  
*GPIO port E identifier.*
  - **#define IOPORT6 GPIOF**  
*GPIO port F identifier.*
  - **#define IOPORT7 GPIOG**  
*GPIO port G identifier.*
  - **#define pal\_lld\_init(config) \_pal\_lld\_init(config)**  
*GPIO ports subsystem initialization.*
  - **#define pal\_lld\_readport(port) ((port)->IDR)**  
*Reads an I/O port.*
  - **#define pal\_lld\_readdlatch(port) ((port)->ODR)**  
*Reads the output latch.*
  - **#define pal\_lld\_writeport(port, bits) ((port)->ODR = (bits))**  
*Writes on a I/O port.*
  - **#define pal\_lld\_setport(port, bits) ((port)->BSRR = (bits))**  
*Sets a bits mask on a I/O port.*
  - **#define pal\_lld\_clearport(port, bits) ((port)->BRR = (bits))**  
*Clears a bits mask on a I/O port.*
  - **#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\_writepad(port, pad, bit) pal\_lld\_writegroup(port, 1, pad, bit)**  
*Writes a logical state on an output pad.*

## Typedefs

- **typedef uint32\_t ioportmask\_t**  
*Digital I/O port sized unsigned type.*
- **typedef uint32\_t iomode\_t**  
*Digital I/O modes.*
- **typedef GPIO\_TypeDef \* ioportid\_t**  
*Port Identifier.*

### 6.11.3 Function Documentation

#### 6.11.3.1 `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 `chSysLock()` and `chSysUnlock()`.

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.

##### Parameters

in	<code>bus</code> the I/O bus, pointer to a <code>IOBus</code> structure
----	---

##### Returns

The bus logical states.

##### Function Class:

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

#### 6.11.3.2 `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 `chSysLock()` and `chSysUnlock()`.

The default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

##### Parameters

in	<code>bus</code> the I/O bus, pointer to a <code>IOBus</code> structure
in	<code>bits</code> the bits to be written on the I/O bus. Values exceeding the bus width are masked so most significant bits are lost.

##### Function Class:

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

#### 6.11.3.3 `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 `chSysLock()` and `chSysUnlock()`.

The default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

##### Parameters

in	<code>bus</code> the I/O bus, pointer to a <code>IOBus</code> structure
in	<code>mode</code> the mode

**Function Class:**

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

**6.11.3.4 void \_pal\_lld\_init ( const PALConfig \* config )**

STM32 I/O ports configuration.

Ports A-D(E, F, G) clocks enabled, AFIO clock enabled.

**Parameters**

in                   *config* the STM32 ports configuration

**Function Class:**

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

**6.11.3.5 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.

**Note**

PAL\_MODE\_UNCONNECTED is implemented as push pull output at 2MHz.

Writing on pads programmed as pull-up or pull-down has the side effect to modify the resistor setting because the output latched data is used for the resistor selection.

**Parameters**

in                   *port* the port identifier  
in                   *mask* the group mask  
in                   *mode* the mode

**Function Class:**

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

**6.11.4 Define Documentation****6.11.4.1 #define PAL\_MODE\_RESET 0**

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.

**6.11.4.2 #define PAL\_MODE\_UNCONNECTED 1**

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\_PUSHULL as example.

**6.11.4.3 #define PAL\_MODE\_INPUT 2**

Regular input high-Z pad.

6.11.4.4 #define PAL\_MODE\_INPUT\_PULLUP 3

Input pad with weak pull up resistor.

6.11.4.5 #define PAL\_MODE\_INPUT\_PULLDOWN 4

Input pad with weak pull down resistor.

6.11.4.6 #define PAL\_MODE\_INPUT\_ANALOG 5

Analog input mode.

6.11.4.7 #define PAL\_MODE\_OUTPUT\_PUSH\_PULL 6

Push-pull output pad.

6.11.4.8 #define PAL\_MODE\_OUTPUT\_OPENDRAIN 7

Open-drain output pad.

6.11.4.9 #define PAL\_LOW 0

Logical low state.

6.11.4.10 #define PAL\_HIGH 1

Logical high state.

6.11.4.11 #define PAL\_PORT\_BIT( *n* ) ((ioportmask\_t)(1 << (*n*)))

Port bit helper macro.

This macro calculates the mask of a bit within a port.

#### Parameters

in *n* bit position within the port

#### Returns

The bit mask.

6.11.4.12 #define PAL\_GROUP\_MASK( *width* ) ((ioportmask\_t)(1 << (*width*)) - 1)

Bits group mask helper.

This macro calculates the mask of a bits group.

#### Parameters

in *width* group width

#### Returns

The group mask.

---

```
6.11.4.13 #define _IOBUS_DATA( name, port, width, offset ) {port, PAL_GROUP_MASK(width), offset}
```

Data part of a static I/O bus initializer.

This macro should be used when statically initializing an I/O bus that is part of a bigger structure.

#### Parameters

in	<i>name</i>	name of the <b>IOBus</b> 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

---

```
6.11.4.14 #define IOBUS_DECL( name, port, width, offset ) IOBus name = _IOBUS_DATA(name, port, width, offset)
```

Static I/O bus initializer.

#### Parameters

in	<i>name</i>	name of the <b>IOBus</b> 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

---

```
6.11.4.15 #define palInit( config ) pal_lld_init(config)
```

PAL subsystem initialization.

#### Note

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

#### Parameters

in	<i>config</i>	pointer to an architecture specific configuration structure. This structure is defined in the low level driver header.
----	---------------	--

#### Function Class:

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

---

```
6.11.4.16 #define palReadPort( port ) ((void)(port), 0)
```

Reads the physical I/O port states.

#### Note

The default implementation always return zero and computes the parameter eventual side effects.

#### Parameters

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

#### Returns

The port logical states.

**Function Class:**

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

6.11.4.17 #define palReadLatch( *port* ) ((void)(*port*), 0)

Reads the output latch.

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

**Note**

The default implementation always return zero and computes the parameter eventual side effects.

**Parameters**

in                   *port* port identifier

**Returns**

The latched logical states.

**Function Class:**

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

6.11.4.18 #define palWritePort( *port*, *bits* ) ((void)(*port*), (void)(*bits*))

Writes a bits mask on a I/O port.

**Note**

The default implementation does nothing except computing the parameters eventual side effects.

**Parameters**

in                   *port* port identifier

in                   *bits* bits to be written on the specified port

**Function Class:**

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

6.11.4.19 #define palSetPort( *port*, *bits* ) palWritePort(*port*, palReadLatch(*port*) | (*bits*))

Sets a bits mask on a I/O port.

**Note**

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `chSysLock()` and `chSysUnlock()`.

The default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

**Parameters**

in                   *port* port identifier

in                   *bits* bits to be ORed on the specified port

**Function Class:**

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

**6.11.4.20 #define palClearPort( *port*, *bits* ) palWritePort(*port*, palReadLatch(*port*) & ~(*bits*))**

Clears a bits mask on a I/O port.

**Note**

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `chSysLock()` and `chSysUnlock()`.

The default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

**Parameters**

in	<i>port</i>	port identifier
in	<i>bits</i>	bits to be cleared on the specified port

**Function Class:**

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

**6.11.4.21 #define palTogglePort( *port*, *bits* ) palWritePort(*port*, palReadLatch(*port*) ^ (*bits*))**

Toggles a bits mask on a I/O port.

**Note**

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `chSysLock()` and `chSysUnlock()`.

The default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

**Parameters**

in	<i>port</i>	port identifier
in	<i>bits</i>	bits to be XORed on the specified port

**Function Class:**

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

**6.11.4.22 #define palReadGroup( *port*, *mask*, *offset* ) ((palReadPort(*port*) >> (*offset*)) & (*mask*))**

Reads a group of bits.

**Parameters**

in	<i>port</i>	port identifier
in	<i>mask</i>	group mask, a logical AND is performed on the input data
in	<i>offset</i>	group bit offset within the port

**Returns**

The group logical states.

**Function Class:**

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

6.11.4.23 #define palWriteGroup( *port*, *mask*, *offset*, *bits* )

**Value:**

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

Writes a group of bits.

**Parameters**

in	<i>port</i>	port identifier
in	<i>mask</i>	group mask, a logical AND is performed on the output data
in	<i>offset</i>	group bit offset within the port
in	<i>bits</i>	bits to be written. Values exceeding the group width are masked.

**Function Class:**

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

6.11.4.24 #define palSetGroupMode( *port*, *mask*, *offset*, *mode* )

Pads group mode setup.

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

**Note**

Programming an unknown or unsupported mode is silently ignored.

**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:**

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

6.11.4.25 #define palReadPad( *port*, *pad* ) ((*palReadPort*(*port*) >> (*pad*)) & 1)

Reads an input pad logical state.

**Note**

The default implementation not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

The default implementation internally uses the [palReadPort\(\)](#).

**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:**

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

6.11.4.26 #define **palWritePad( port, pad, bit )**

**Value:**

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

Writes a logical state on an output pad.

**Note**

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `chSysLock()` and `chSysUnlock()`.

The default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

The default implementation internally uses the `palReadLatch()` and `palWritePort()`.

**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 <code>PAL_LOW</code> or <code>PAL_HIGH</code>

**Function Class:**

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

6.11.4.27 #define **palSetPad( port, pad ) palSetPort(port, PAL\_PORT\_BIT(pad))**

Sets a pad logical state to `PAL_HIGH`.

**Note**

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `chSysLock()` and `chSysUnlock()`.

The default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

The default implementation internally uses the `palSetPort()`.

**Parameters**

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

**Function Class:**

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

6.11.4.28 #define palClearPad( *port*, *pad* ) palClearPort(*port*, PAL\_PORT\_BIT(*pad*))

Clears a pad logical state to PAL\_LOW.

#### Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between chSysLock() and chSysUnlock().

The default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

The default implementation internally uses the [palClearPort\(\)](#).

#### Parameters

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

#### Function Class:

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

6.11.4.29 #define palTogglePad( *port*, *pad* ) palTogglePort(*port*, PAL\_PORT\_BIT(*pad*))

Toggles a pad logical state.

#### Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between chSysLock() and chSysUnlock().

The default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

The default implementation internally uses the [palTogglePort\(\)](#).

#### Parameters

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

#### Function Class:

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

6.11.4.30 #define palSetPadMode( *port*, *pad*, *mode* ) palSetGroupMode(*port*, PAL\_PORT\_BIT(*pad*), 0, *mode*)

Pad mode setup.

This function programs a pad with the specified mode.

#### Note

The default implementation not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

Programming an unknown or unsupported mode is silently ignored.

#### Parameters

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

**Function Class:**

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

6.11.4.31 #define PAL\_MODE STM32\_ALTERNATE\_PUSHPULL 16

STM32 specific alternate push-pull output mode.

6.11.4.32 #define PAL\_MODE STM32\_ALTERNATE\_OPENDRAIN 17

STM32 specific alternate open-drain output mode.

6.11.4.33 #define PAL\_IOPORTS\_WIDTH 16

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

6.11.4.34 #define PAL\_WHOLE\_PORT ((ioportmask\_t)0xFFFF)

Whole port mask.

This macro specifies all the valid bits into a port.

6.11.4.35 #define IOPORT1 GPIOA

GPIO port A identifier.

6.11.4.36 #define IOPORT2 GPIOB

GPIO port B identifier.

6.11.4.37 #define IOPORT3 GPIOC

GPIO port C identifier.

6.11.4.38 #define IOPORT4 GPIOD

GPIO port D identifier.

6.11.4.39 #define IOPORT5 GPIOE

GPIO port E identifier.

6.11.4.40 #define IOPORT6 GPIOF

GPIO port F identifier.

6.11.4.41 #define IOPORT7 GPIOG

GPIO port G identifier.

```
6.11.4.42 #define pal_lld_init( config ) _pal_lld_init(config)
```

GPIO ports subsystem initialization.

**Function Class:**

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

```
6.11.4.43 #define pal_lld_readport( port ) ((port)->IDR)
```

Reads an I/O port.

This function is implemented by reading the GPIO IDR register, the implementation has no side effects.

**Note**

This function is not meant to be invoked directly by the application code.

**Parameters**

in                   *port*    port identifier

**Returns**

The port bits.

**Function Class:**

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

```
6.11.4.44 #define pal_lld_readlatch( port ) ((port)->ODR)
```

Reads the output latch.

This function is implemented by reading the GPIO ODR register, the implementation has no side effects.

**Note**

This function is not meant to be invoked directly by the application code.

**Parameters**

in                   *port*    port identifier

**Returns**

The latched logical states.

**Function Class:**

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

```
6.11.4.45 #define pal_lld_writeport( port, bits ) ((port)->ODR = (bits))
```

Writes on a I/O port.

This function is implemented by writing the GPIO ODR register, the implementation has no side effects.

**Note**

Writing on pads programmed as pull-up or pull-down has the side effect to modify the resistor setting because the output latched data is used for the resistor selection.

**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.

6.11.4.46 #define pal\_lld\_setport( *port*, *bits* ) ((*port*)>BSRR = (*bits*))

Sets a bits mask on a I/O port.

This function is implemented by writing the GPIO BSRR register, the implementation has no side effects.

**Note**

Writing on pads programmed as pull-up or pull-down has the side effect to modify the resistor setting because the output latched data is used for the resistor selection.

**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.

6.11.4.47 #define pal\_lld\_clearport( *port*, *bits* ) ((*port*)>BRR = (*bits*))

Clears a bits mask on a I/O port.

This function is implemented by writing the GPIO BRR register, the implementation has no side effects.

**Note**

Writing on pads programmed as pull-up or pull-down has the side effect to modify the resistor setting because the output latched data is used for the resistor selection.

**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.

6.11.4.48 #define pal\_lld\_writegroup( *port*, *mask*, *offset*, *bits* )

**Value:**

```
((port)>BSRR = ((~(bits) & (mask)) << (16 + (offset))) | \
((bits) & (mask)) << (offset))) \
```

Writes a group of bits.

This function is implemented by writing the GPIO BSRR register, the implementation has no side effects.

**Note**

Writing on pads programmed as pull-up or pull-down has the side effect to modify the resistor setting because the output latched data is used for the resistor selection.

**Parameters**

in	<i>port</i>	port identifier
in	<i>mask</i>	group mask
in	<i>offset</i>	the 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.

```
6.11.4.49 #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**

Writing on pads programmed as pull-up or pull-down has the side effect to modify the resistor setting because the output latched data is used for the resistor selection.

**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.

```
6.11.4.50 #define pal_lld_writepad( port, pad, bit ) pal_lld_writegroup(port, 1, pad, bit)
```

Writes a logical state on an output pad.

**Note**

Writing on pads programmed as pull-up or pull-down has the side effect to modify the resistor setting because the output latched data is used for the resistor selection.

**Parameters**

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

**Function Class:**

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

## 6.11.5 Typedef Documentation

### 6.11.5.1 `typedef uint32_t ioportmask_t`

Digital I/O port sized unsigned type.

### 6.11.5.2 `typedef uint32_t iomode_t`

Digital I/O modes.

### 6.11.5.3 `typedef GPIO_TypeDef* 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.

## 6.12 PWM Driver

### 6.12.1 Detailed Description

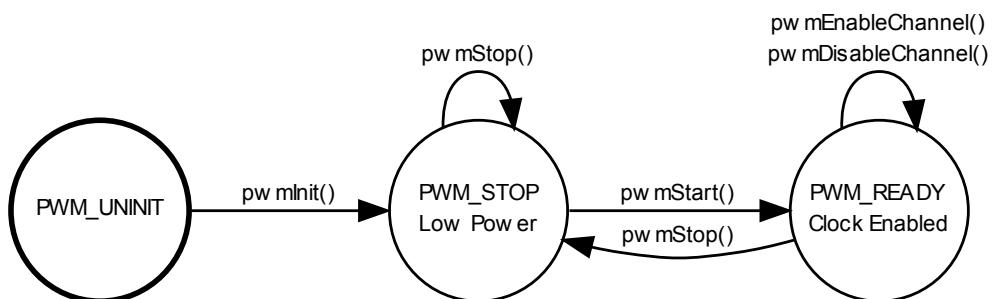
Generic PWM Driver. This module implements a generic PWM (Pulse Width Modulation) driver.

#### Precondition

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

### 6.12.2 Driver State Machine

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



### 6.12.3 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.

## Data Structures

- struct `PWMChannelConfig`  
*PWM driver channel configuration structure.*
- struct `PWMConfig`  
*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.*
- 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.*

## Variables

- **PWMDriver PWMD1**  
*PWMD1 driver identifier.*
- **PWMDriver PWMD2**  
*PWMD2 driver identifier.*
- **PWMDriver PWMD3**  
*PWMD3 driver identifier.*
- **PWMDriver PWMD4**  
*PWMD4 driver identifier.*
- **PWMDriver PWMD5**  
*PWMD5 driver identifier.*
- **PWMDriver PWMD8**  
*PWMD8 driver identifier.*

## PWM output mode macros

- **#define PWM\_OUTPUT\_MASK 0x0F**  
*Standard output modes mask.*
- **#define PWM\_OUTPUT\_DISABLED 0x00**  
*Output not driven, callback only.*
- **#define PWM\_OUTPUT\_ACTIVE\_HIGH 0x01**  
*Positive PWM logic, active is logic level one.*
- **#define PWM\_OUTPUT\_ACTIVE\_LOW 0x02**  
*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, period)**  
*Changes the period the PWM peripheral.*
- **#define pwmEnableChannelI(pwmp, channel, width) pwm\_lld\_enable\_channel(pwmp, channel, width)**  
*Enables a PWM channel.*
- **#define pwmDisableChannelI(pwmp, channel) pwm\_lld\_disable\_channel(pwmp, channel)**  
*Disables a PWM channel.*

## Configuration options

- `#define STM32_PWM_USE_ADVANCED TRUE`  
*If advanced timer features switch.*
- `#define STM32_PWM_USE_TIM1 TRUE`  
*PWMD1 driver enable switch.*
- `#define STM32_PWM_USE_TIM2 TRUE`  
*PWMD2 driver enable switch.*
- `#define STM32_PWM_USE_TIM3 TRUE`  
*PWMD3 driver enable switch.*
- `#define STM32_PWM_USE_TIM4 TRUE`  
*PWMD4 driver enable switch.*
- `#define STM32_PWM_USE_TIM5 TRUE`  
*PWMD5 driver enable switch.*
- `#define STM32_PWM_USE_TIM8 TRUE`  
*PWMD8 driver enable switch.*
- `#define STM32_PWM_TIM1_IRQ_PRIORITY 7`  
*PWMD1 interrupt priority level setting.*
- `#define STM32_PWM_TIM2_IRQ_PRIORITY 7`  
*PWMD2 interrupt priority level setting.*
- `#define STM32_PWM_TIM3_IRQ_PRIORITY 7`  
*PWMD3 interrupt priority level setting.*
- `#define STM32_PWM_TIM4_IRQ_PRIORITY 7`  
*PWMD4 interrupt priority level setting.*
- `#define STM32_PWM_TIM5_IRQ_PRIORITY 7`  
*PWMD5 interrupt priority level setting.*
- `#define STM32_PWM_TIM8_IRQ_PRIORITY 7`  
*PWMD8 interrupt priority level setting.*

## Defines

- `#define PWM_CHANNELS 4`  
*Number of PWM channels per PWM driver.*
- `#define PWM_COMPLEMENTARY_OUTPUT_MASK 0xF0`  
*Complementary output modes mask.*
- `#define PWM_COMPLEMENTARY_OUTPUT_DISABLED 0x00`  
*Complementary output not driven.*
- `#define PWM_COMPLEMENTARY_OUTPUT_ACTIVE_HIGH 0x10`  
*Complementary output, active is logic level one.*
- `#define PWM_COMPLEMENTARY_OUTPUT_ACTIVE_LOW 0x20`  
*Complementary output, active is logic level zero.*
- `#define pwm_lld_change_period(pwmp, period) ((pwmp)->tim->ARR = (uint16_t)((period) - 1))`  
*Changes the period the PWM peripheral.*

## Typedefs

- **typedef struct PWMDriver PWMDriver**  
*Type of a structure representing a PWM driver.*
- **typedef void(\* pwmcallback\_t)(PWMDriver \*pwmp)**  
*PWM notification callback type.*
- **typedef uint32\_t pwmmode\_t**  
*PWM mode type.*
- **typedef uint8\_t pwmchannel\_t**  
*PWM channel type.*
- **typedef uint16\_t pwcnt\_t**  
*PWM counter type.*

## Enumerations

- **enum pwmstate\_t { PWM\_UNINIT = 0, PWM\_STOP = 1, PWM\_READY = 2 }**  
*Driver state machine possible states.*

### 6.12.4 Function Documentation

#### 6.12.4.1 void pwmlInit ( void )

PWM Driver initialization.

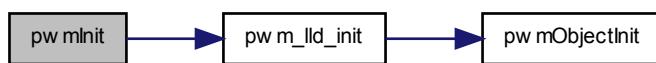
##### Note

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

##### Function Class:

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

Here is the call graph for this function:



#### 6.12.4.2 void pwmObjectInit ( PWMDriver \* pwmp )

Initializes the standard part of a **PWMDriver** structure.

##### Parameters

out **pwmp** pointer to a **PWMDriver** object

##### Function Class:

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

#### 6.12.4.3 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.

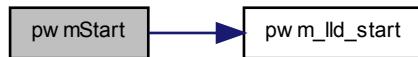
##### Parameters

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

##### Function Class:

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

Here is the call graph for this function:



#### 6.12.4.4 void pwmStop ( **PWMDriver** \* *pwmp* )

Deactivates the PWM peripheral.

##### 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.

Here is the call graph for this function:



#### 6.12.4.5 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	<code>pwmp</code>	pointer to a <code>PWMDriver</code> object
in	<code>period</code>	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.

### 6.12.4.6 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	<code>pwmp</code>	pointer to a <code>PWMDriver</code> object
in	<code>channel</code>	PWM channel identifier (0... <code>PWM_CHANNELS-1</code> )
in	<code>width</code>	PWM pulse width as clock pulses number

#### Function Class:

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

Here is the call graph for this function:



#### 6.12.4.7 void pwmDisableChannel ( PWMDriver \* *pwmp*, pwmchannel\_t *channel* )

Disables a PWM channel.

##### Precondition

The PWM unit must have been activated using [pwmStart \(\)](#).

##### Postcondition

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

##### Note

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

##### Parameters

in	<i>pwmp</i>	pointer to a <a href="#">PWMDriver</a> object
in	<i>channel</i>	PWM channel identifier (0...PWM_CHANNELS-1)

##### Function Class:

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

Here is the call graph for this function:



#### 6.12.4.8 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:



**6.12.4.9 void pwm\_lld\_start ( PWMDriver \* pwmp )**

Configures and activates the PWM peripheral.

**Note**

Starting a driver that is already in the `PWM_READY` state disables all the active channels.

**Parameters**

in *pwmp* pointer to a `PWMDriver` object

**Function Class:**

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

**6.12.4.10 void pwm\_lld\_stop ( PWMDriver \* pwmp )**

Deactivates the PWM peripheral.

**Parameters**

in *pwmp* pointer to a `PWMDriver` object

**Function Class:**

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

**6.12.4.11 void pwm\_lld\_enable\_channel ( PWMDriver \* pwmp, pwmchannel\_t channel, pwcnt\_t width )**

Enables a PWM channel.

**Precondition**

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

**Postcondition**

The channel is active using the specified configuration.

**Note**

The function has effect at the next cycle start.

**Parameters**

in	<i>pwmp</i>	pointer to a <code>PWMDriver</code> object
in	<i>channel</i>	PWM channel identifier (0... <code>PWM_CHANNELS-1</code> )
in	<i>width</i>	PWM pulse width as clock pulses number

**Function Class:**

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

**6.12.4.12 void pwm\_lld\_disable\_channel ( PWMDriver \* pwmp, pwmchannel\_t channel )**

Disables a PWM channel.

**Precondition**

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

**Postcondition**

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

**Note**

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...PWM_CHANNELS-1)

**Function Class:**

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

## 6.12.5 Variable Documentation

### 6.12.5.1 PWMDriver PWMD1

PWMD1 driver identifier.

**Note**

The driver PWMD1 allocates the complex timer TIM1 when enabled.

### 6.12.5.2 PWMDriver PWMD2

PWMD2 driver identifier.

**Note**

The driver PWMD2 allocates the timer TIM2 when enabled.

### 6.12.5.3 PWMDriver PWMD3

PWMD3 driver identifier.

**Note**

The driver PWMD3 allocates the timer TIM3 when enabled.

### 6.12.5.4 PWMDriver PWMD4

PWMD4 driver identifier.

**Note**

The driver PWMD4 allocates the timer TIM4 when enabled.

### 6.12.5.5 PWMDriver PWMD5

PWMD5 driver identifier.

#### Note

The driver PWMD5 allocates the timer TIM5 when enabled.

### 6.12.5.6 PWMDriver PWMD8

PWMD8 driver identifier.

#### Note

The driver PWMD5 allocates the timer TIM5 when enabled.

## 6.12.6 Define Documentation

### 6.12.6.1 #define PWM\_OUTPUT\_MASK 0x0F

Standard output modes mask.

### 6.12.6.2 #define PWM\_OUTPUT\_DISABLED 0x00

Output not driven, callback only.

### 6.12.6.3 #define PWM\_OUTPUT\_ACTIVE\_HIGH 0x01

Positive PWM logic, active is logic level one.

### 6.12.6.4 #define PWM\_OUTPUT\_ACTIVE\_LOW 0x02

Inverse PWM logic, active is logic level zero.

### 6.12.6.5 #define PWM\_FRACTION\_TO\_WIDTH( *pwmp*, *denominator*, *numerator* )

#### Value:

```
((uint16_t) (((uint32_t) (pwmp)->period) *  
           (uint32_t) (numerator)) / (uint32_t) (denominator))) \
```

Converts from fraction to pulse width.

#### Note

Be careful with rounding errors, this is integer math not magic. You can specify tenths of thousandth but make sure you have the proper hardware resolution by carefully choosing the clock source and prescaler settings, see `PWM_COMPUTE_PSC`.

#### Parameters

in	<i>pwmp</i>	pointer to a <code>PWMDriver</code> object
in	<i>denominator</i>	denominator of the fraction
in	<i>numerator</i>	numerator of the fraction

**Returns**

The pulse width to be passed to `pwmEnableChannel()`.

**Function Class:**

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

6.12.6.6 #define PWM\_DEGREES\_TO\_WIDTH( *pwmp*, *degrees* ) PWM\_FRACTION\_TO\_WIDTH(*pwmp*, 36000, *degrees*)

Converts from degrees to pulse width.

**Note**

Be careful with rounding errors, this is integer math not magic. You can specify hundredths of degrees but make sure you have the proper hardware resolution by carefully choosing the clock source and prescaler settings, see `PWM_COMPUTE_PSC`.

**Parameters**

in	<i>pwmp</i>	pointer to a <code>PWMDriver</code> object
in	<i>degrees</i>	degrees as an integer between 0 and 36000

**Returns**

The pulse width to be passed to `pwmEnableChannel()`.

**Function Class:**

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

6.12.6.7 #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 <code>PWMDriver</code> 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.

6.12.6.8 #define pwmChangePeriod( *pwmp*, *period* )

**Value:**

{ \ }

```
(pwmp)->period = (period);
pwm_lld_change_period(pwmp, period);
}
```

Changes the period the PWM peripheral.

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

#### Precondition

The PWM unit must have been activated using [pwmStart \(\)](#).

#### Postcondition

The PWM unit period is changed to the new value.

#### Note

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 <a href="#">PWMDriver</a> object
in	<i>period</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.

#### 6.12.6.9 #define pwmEnableChannell( *pwmp*, *channel*, *width* ) pwm\_lld\_enable\_channel(*pwmp*, *channel*, *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 <a href="#">PWMDriver</a> object
in	<i>channel</i>	PWM channel identifier (0...PWM_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.

#### 6.12.6.10 #define pwmDisableChannell( *pwmp*, *channel* ) pwm\_lld\_disable\_channel(*pwmp*, *channel*)

Disables a PWM channel.

**Precondition**

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

**Postcondition**

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

**Note**

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

**Parameters**

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

**6.12.6.11 #define PWM\_CHANNELS 4**

Number of PWM channels per PWM driver.

**6.12.6.12 #define PWM\_COMPLEMENTARY\_OUTPUT\_MASK 0xF0**

Complementary output modes mask.

**Note**

This is an STM32-specific setting.

**6.12.6.13 #define PWM\_COMPLEMENTARY\_OUTPUT\_DISABLED 0x00**

Complementary output not driven.

**Note**

This is an STM32-specific setting.

**6.12.6.14 #define PWM\_COMPLEMENTARY\_OUTPUT\_ACTIVE\_HIGH 0x10**

Complementary output, active is logic level one.

**Note**

This is an STM32-specific setting.

This setting is only available if the configuration option `STM32_PWM_USE_ADVANCED` is set to TRUE and only for advanced timers TIM1 and TIM8.

6.12.6.15 #define PWM\_COMPLEMENTARY\_OUTPUT\_ACTIVE\_LOW 0x20

Complementary output, active is logic level zero.

**Note**

This is an STM32-specific setting.

This setting is only available if the configuration option STM32\_PWM\_USE\_ADVANCED is set to TRUE and only for advanced timers TIM1 and TIM8.

6.12.6.16 #define STM32\_PWM\_USE\_ADVANCED TRUE

If advanced timer features switch.

If set to TRUE the advanced features for TIM1 and TIM8 are enabled.

**Note**

The default is TRUE.

6.12.6.17 #define STM32\_PWM\_USE\_TIM1 TRUE

PWMD1 driver enable switch.

If set to TRUE the support for PWMD1 is included.

**Note**

The default is TRUE.

6.12.6.18 #define STM32\_PWM\_USE\_TIM2 TRUE

PWMD2 driver enable switch.

If set to TRUE the support for PWMD2 is included.

**Note**

The default is TRUE.

6.12.6.19 #define STM32\_PWM\_USE\_TIM3 TRUE

PWMD3 driver enable switch.

If set to TRUE the support for PWMD3 is included.

**Note**

The default is TRUE.

6.12.6.20 #define STM32\_PWM\_USE\_TIM4 TRUE

PWMD4 driver enable switch.

If set to TRUE the support for PWMD4 is included.

**Note**

The default is TRUE.

6.12.6.21 #define STM32\_PWM\_USE\_TIM5 TRUE

PWMD5 driver enable switch.

If set to TRUE the support for PWMD5 is included.

#### Note

The default is TRUE.

6.12.6.22 #define STM32\_PWM\_USE\_TIM8 TRUE

PWMD8 driver enable switch.

If set to TRUE the support for PWMD8 is included.

#### Note

The default is TRUE.

6.12.6.23 #define STM32\_PWM\_TIM1\_IRQ\_PRIORITY 7

PWMD1 interrupt priority level setting.

6.12.6.24 #define STM32\_PWM\_TIM2\_IRQ\_PRIORITY 7

PWMD2 interrupt priority level setting.

6.12.6.25 #define STM32\_PWM\_TIM3\_IRQ\_PRIORITY 7

PWMD3 interrupt priority level setting.

6.12.6.26 #define STM32\_PWM\_TIM4\_IRQ\_PRIORITY 7

PWMD4 interrupt priority level setting.

6.12.6.27 #define STM32\_PWM\_TIM5\_IRQ\_PRIORITY 7

PWMD5 interrupt priority level setting.

6.12.6.28 #define STM32\_PWM\_TIM8\_IRQ\_PRIORITY 7

PWMD8 interrupt priority level setting.

6.12.6.29 #define pwm\_lld\_change\_period( *pwmp*, *period* ) ((*pwmp*)>tim->ARR = (uint16\_t)((*period*) - 1))

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 <code>PWMDriver</code> object
in	<i>period</i>	new cycle time in ticks

**Function Class:**

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

### 6.12.7 Typedef Documentation

#### 6.12.7.1 `typedef struct PWMDriver PWMDriver`

Type of a structure representing a PWM driver.

#### 6.12.7.2 `typedef void(* pwmcallback_t)(PWMDriver *pwmp)`

PWM notification callback type.

**Parameters**

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

#### 6.12.7.3 `typedef uint32_t pwmmode_t`

PWM mode type.

#### 6.12.7.4 `typedef uint8_t pwmchannel_t`

PWM channel type.

#### 6.12.7.5 `typedef uint16_t pwmcnt_t`

PWM counter type.

### 6.12.8 Enumeration Type Documentation

#### 6.12.8.1 `enum pwmstate_t`

Driver state machine possible states.

**Enumerator:**

**`PWM_UNINIT`** Not initialized.

**`PWM_STOP`** Stopped.

**`PWM_READY`** Ready.

## 6.13 RTC Driver

### 6.13.1 Detailed Description

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

#### Precondition

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

#### Data Structures

- struct `RTCCallbackConfig`  
*Structure representing an RTC callbacks config.*
- struct `RTCTime`  
*Structure representing an RTC time stamp.*
- struct `RTCAlarm`  
*Structure representing an RTC alarm time stamp.*
- struct `RTCDriver`  
*Structure representing an RTC driver.*

#### Functions

- void `rtcInit` (void)  
*RTC Driver initialization.*
- void `rtcSetTime` (RTCDriver \*rtcp, const RTCTime \*timespec)  
*Set current time.*
- void `rtcGetTime` (RTCDriver \*rtcp, RTCTime \*timespec)  
*Get current time.*
- void `rtcSetAlarm` (RTCDriver \*rtcp, rtcalarm\_t alarm, const RTCAlarm \*alarmspec)  
*Set alarm time.*
- void `rtcGetAlarm` (RTCDriver \*rtcp, rtcalarm\_t alarm, RTCAlarm \*alarmspec)  
*Get current alarm.*
- void `rtcSetCallback` (RTCDriver \*rtcp, rtccb\_t callback)  
*Enables or disables RTC callbacks.*
- `CH_IRQ_HANDLER` (RTC\_IRQHandler)  
*RTC interrupt handler.*
- void `rtc_lld_init` (void)  
*Enable access to registers and initialize RTC if BKP domain was previously reseted.*
- void `rtc_lld_set_time` (RTCDriver \*rtcp, const RTCTime \*timespec)  
*Set current time.*
- void `rtc_lld_get_time` (RTCDriver \*rtcp, RTCTime \*timespec)  
*Get current time.*
- void `rtc_lld_set_alarm` (RTCDriver \*rtcp, rtcalarm\_t alarm, const RTCAlarm \*alarmspec)  
*Set alarm time.*
- void `rtc_lld_get_alarm` (RTCDriver \*rtcp, rtcalarm\_t alarm, RTCAlarm \*alarmspec)  
*Get current alarm.*
- void `rtc_lld_set_callback` (RTCDriver \*rtcp, rtccb\_t callback)  
*Enables or disables RTC callbacks.*

## Variables

- **RTCDriver RTCD1**  
*RTC driver identifier.*

## Configuration options

- #define **STM32\_RTC\_IRQ\_PRIORITY** 15

## Defines

- #define **rtcSetTimel**(rtcp, timespec) rtc\_lld\_set\_time(rtcp, timespec)  
*Set current time.*
- #define **rtcGetTimel**(rtcp, timespec) rtc\_lld\_get\_time(rtcp, timespec)  
*Get current time.*
- #define **rtcSetAlarml**(rtcp, alarm, alarmspec) rtc\_lld\_set\_alarm(rtcp, alarm, alarmspec)  
*Set alarm time.*
- #define **rtcGetAlarml**(rtcp, alarm, alarmspec) rtc\_lld\_get\_alarm(rtcp, alarm, alarmspec)  
*Get current alarm.*
- #define **rtcSetCallbackl**(rtcp, callback) rtc\_lld\_set\_callback(rtcp, callback)  
*Enables or disables RTC callbacks.*
- #define **rtc\_lld\_apb1\_sync()** {while (((RTC->CRL & RTC\_CRL\_RSF) == 0);}  
*Wait for synchronization of RTC registers with APB1 bus.*
- #define **rtc\_lld\_wait\_write()** {while (((RTC->CRL & RTC\_CRL\_RTOFF) == 0);}  
*Wait for previous write operation complete.*
- #define **rtc\_lld\_acquire()** {rtc\_lld\_wait\_write(); RTC->CRL |= RTC\_CRL\_CNF;}  
*Acquires write access to RTC registers.*
- #define **rtc\_lld\_release()** {RTC->CRL &= ~RTC\_CRL\_CNF;}  
*Releases write access to RTC registers.*
- #define **RTC\_SUPPORTS\_CALLBACKS** TRUE  
*This RTC implementation supports callbacks.*
- #define **RTC\_ALARMS** 1  
*One alarm comparator available.*

## Typedefs

- typedef struct **RTCDriver RTCDriver**  
*Type of a structure representing an RTC driver.*
- typedef struct **RTCTime RTCTime**  
*Type of a structure representing an RTC time stamp.*
- typedef struct **RTCAlarm RTCAlarm**  
*Type of a structure representing an RTC alarm time stamp.*
- typedef struct **RTCCallbackConfig RTCCallbackConfig**  
*Type of a structure representing an RTC callbacks config.*
- typedef uint32\_t **rtcalarm\_t**  
*Type of an RTC alarm.*
- typedef void(\* **rtccb\_t**)(RTCDriver \*rtcp, rtcevent\_t event)  
*Type of a generic RTC callback.*

## Enumerations

- enum `rtcevent_t` { , `RTC_EVENT_ALARM` = 1, `RTC_EVENT_OVERFLOW` = 2 }
- Type of an RTC event.*

### 6.13.2 Function Documentation

#### 6.13.2.1 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:



#### 6.13.2.2 void rtcSetTime( RTCDriver \* rtcp, const RTCTime \* timespec )

Set current time.

##### Parameters

in	<code>rtcp</code>	pointer to RTC driver structure
in	<code>timespec</code>	pointer to a <code>RTCTime</code> structure

##### Function Class:

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

#### 6.13.2.3 void rtcGetTime( RTCDriver \* rtcp, RTCTime \* timespec )

Get current time.

##### Parameters

in	<code>rtcp</code>	pointer to RTC driver structure
out	<code>timespec</code>	pointer to a <code>RTCTime</code> structure

##### Function Class:

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

**6.13.2.4 void rtcSetAlarm ( RTCDriver \* *rtcp*, rtcalarm\_t *alarm*, const RTCAlarm \* *alarmspec* )**

Set alarm time.

**Parameters**

in	<i>rtcp</i>	pointer to RTC driver structure
in	<i>alarm</i>	alarm identifier
in	<i>alarmspec</i>	pointer to a <a href="#">RTCAlarm</a> structure or NULL

**Function Class:**

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

**6.13.2.5 void rtcGetAlarm ( RTCDriver \* *rtcp*, rtcalarm\_t *alarm*, RTCAlarm \* *alarmspec* )**

Get current alarm.

**Note**

If an alarm has not been set then the returned alarm specification is not meaningful.

**Parameters**

in	<i>rtcp</i>	pointer to RTC driver structure
in	<i>alarm</i>	alarm identifier
out	<i>alarmspec</i>	pointer to a <a href="#">RTCAlarm</a> structure

**Function Class:**

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

**6.13.2.6 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.

**Parameters**

in	<i>rtcp</i>	pointer to RTC driver structure
in	<i>callback</i>	callback function pointer or <code>NULL</code>

**Function Class:**

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

**6.13.2.7 CH\_IRQ\_HANDLER ( RTC\_IRQHandler )**

RTC interrupt handler.

**Function Class:**

Interrupt handler, this function should not be directly invoked.

**6.13.2.8 void rtc\_lld\_init ( void )**

Enable access to registers and initialize RTC if BKP domain was previously reseted.

**Note**

: Cold start time of LSE oscillator on STM32 platform takes about 3 seconds.

**Function Class:**

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

6.13.2.9 void rtc\_lld\_set\_time ( **RTCDriver** \* *rtcp*, const **RTCTime** \* *timespec* )

Set current time.

**Note**

Fractional part will be silently ignored. There is no possibility to change it on STM32F1xx platform.

**Parameters**

in	<i>rtcp</i>	pointer to RTC driver structure
in	<i>timespec</i>	pointer to a <b>RTCTime</b> structure

**Function Class:**

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

6.13.2.10 void rtc\_lld\_get\_time ( **RTCDriver** \* *rtcp*, **RTCTime** \* *timespec* )

Get current time.

**Parameters**

in	<i>rtcp</i>	pointer to RTC driver structure
in	<i>timespec</i>	pointer to a <b>RTCTime</b> structure

**Function Class:**

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

6.13.2.11 void rtc\_lld\_set\_alarm ( **RTCDriver** \* *rtcp*, **rtcalarm\_t** *alarm*, const **RTCAlarm** \* *alarmspec* )

Set alarm time.

**Note**

Default value after BKP domain reset is 0xFFFFFFFF

**Parameters**

in	<i>rtcp</i>	pointer to RTC driver structure
in	<i>alarm</i>	alarm identifier
in	<i>alarmspec</i>	pointer to a <b>RTCAlarm</b> structure

**Function Class:**

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

6.13.2.12 void rtc\_lld\_get\_alarm ( **RTCDriver** \* *rtcp*, **rtcalarm\_t** *alarm*, **RTCAlarm** \* *alarmspec* )

Get current alarm.

**Note**

If an alarm has not been set then the returned alarm specification is not meaningful.  
Default value after BKP domain reset is 0xFFFFFFFF.

**Parameters**

in	<i>rtcp</i>	pointer to RTC driver structure
in	<i>alarm</i>	alarm identifier
out	<i>alarmspec</i>	pointer to a <a href="#">RTCAAlarm</a> structure

**Function Class:**

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

**6.13.2.13 void rtc\_lld\_set\_callback ( RTCDriver \* *rtcp*, rtccb\_t *callback* )**

Enables or disables RTC callbacks.

This function enables or disables callbacks, use a `NULL` pointer in order to disable a callback.

**Parameters**

in	<i>rtcp</i>	pointer to RTC driver structure
in	<i>callback</i>	callback function pointer or <code>NULL</code>

**Function Class:**

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

**6.13.3 Variable Documentation****6.13.3.1 RTCDriver RTCD1**

RTC driver identifier.

**6.13.4 Define Documentation****6.13.4.1 #define rtcSetTime( *rtcp*, *timespec* ) rtc\_lld\_set\_time(*rtcp*,*timespec*)**

Set current time.

**Parameters**

in	<i>rtcp</i>	pointer to RTC driver structure
in	<i>timespec</i>	pointer to a <a href="#">RTCTime</a> 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.

**6.13.4.2 #define rtcGetTime( *rtcp*, *timespec* ) rtc\_lld\_get\_time(*rtcp*,*timespec*)**

Get current time.

**Parameters**

in	<i>rtcp</i>	pointer to RTC driver structure
out	<i>timespec</i>	pointer to a <a href="#">RTCTime</a> 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.

6.13.4.3 #define rtcSetAlarmI( *rtcp*, *alarm*, *alarmspec* ) rtc\_lld\_set\_alarm(*rtcp*, *alarm*, *alarmspec*)

Set alarm time.

**Parameters**

in	<i>rtcp</i>	pointer to RTC driver structure
in	<i>alarm</i>	alarm identifier
in	<i>alarmspec</i>	pointer to a <a href="#">RTCAalarm</a> structure or NULL

**Function Class:**

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

6.13.4.4 #define rtcGetAlarmI( *rtcp*, *alarm*, *alarmspec* ) rtc\_lld\_get\_alarm(*rtcp*, *alarm*, *alarmspec*)

Get current alarm.

**Note**

If an alarm has not been set then the returned alarm specification is not meaningful.

**Parameters**

in	<i>rtcp</i>	pointer to RTC driver structure
in	<i>alarm</i>	alarm identifier
out	<i>alarmspec</i>	pointer to a <a href="#">RTCAalarm</a> 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.

6.13.4.5 #define rtcSetCallbackI( *rtcp*, *callback* ) rtc\_lld\_set\_callback(*rtcp*, *callback*)

Enables or disables RTC callbacks.

This function enables or disables the callback, use a `NULL` pointer in order to disable it.

**Parameters**

in	<i>rtcp</i>	pointer to RTC driver structure
in	<i>callback</i>	callback function pointer 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.

6.13.4.6 #define rtc\_lld\_apb1\_sync( ) {while ((RTC->CRL & RTC\_CRL\_RSF) == 0);}

Wait for synchronization of RTC registers with APB1 bus.

This function must be invoked before trying to read RTC registers in the backup domain: DIV, CNT, ALR. CR registers can always be read.

**Function Class:**

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

```
6.13.4.7 #define rtc_lld_wait_write( ) {while ((RTC->CRL & RTC_CRL_RTOFF) == 0);}
```

Wait for previous write operation complete.

This function must be invoked before writing to any RTC registers

**Function Class:**

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

```
6.13.4.8 #define rtc_lld_acquire( ) {rtc_lld_wait_write(); RTC->CRL |= RTC_CRL_CNF;}
```

Acquires write access to RTC registers.

Before writing to the backup domain RTC registers the previous write operation must be completed. Use this function before writing to PRL, CNT, ALR registers.

**Function Class:**

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

```
6.13.4.9 #define rtc_lld_release( ) {RTC->CRL &= ~RTC_CRL_CNF;}
```

Releases write access to RTC registers.

**Function Class:**

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

```
6.13.4.10 #define RTC_SUPPORTS_CALLBACKS TRUE
```

This RTC implementation supports callbacks.

```
6.13.4.11 #define RTC_ALARMS 1
```

One alarm comparator available.

## 6.13.5 Typedef Documentation

```
6.13.5.1 typedef struct RTCDriver RTCDriver
```

Type of a structure representing an RTC driver.

```
6.13.5.2 typedef struct RTCTime RTCTime
```

Type of a structure representing an RTC time stamp.

**6.13.5.3 `typedef struct RTCAlarm RTCAlarm`**

Type of a structure representing an RTC alarm time stamp.

**6.13.5.4 `typedef struct RTCCallbackConfig RTCCallbackConfig`**

Type of a structure representing an RTC callbacks config.

**6.13.5.5 `typedef uint32_t rtcalarm_t`**

Type of an RTC alarm.

Meaningful on platforms with more than 1 alarm comparator.

**6.13.5.6 `typedef void(* rtccb_t)(RTCDriver *rtcp, rtcevent_t event)`**

Type of a generic RTC callback.

**6.13.6 Enumeration Type Documentation****6.13.6.1 `enum rtcevent_t`**

Type of an RTC event.

**Enumerator:**

***RTC\_EVENT\_ALARM*** Triggered every second.

***RTC\_EVENT\_OVERFLOW*** Triggered on alarm.

**6.14 SDC Driver****6.14.1 Detailed Description**

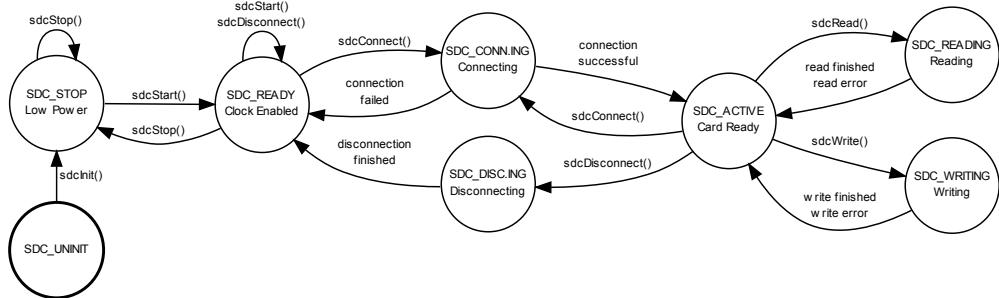
Generic SD Card Driver. This module implements a generic SDC (Secure Digital Card) driver.

**Precondition**

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

**6.14.2 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).



### 6.14.3 SDC Operations.

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

#### Data Structures

- struct **SDCConfig**  
*Driver configuration structure.*
- struct **SDCDriver**  
*Structure representing an SDC driver.*

#### Functions

- void **sdcInit** (void)  
*SDC Driver initialization.*
- void **sdcObjectInit** (**SDCDriver** \*sdcp)  
*Initializes the standard part of a **SDCDriver** structure.*
- void **sdcStart** (**SDCDriver** \*sdcp, const **SDCConfig** \*config)  
*Configures and activates the SDC peripheral.*
- void **sdcStop** (**SDCDriver** \*sdcp)  
*Deactivates the SDC peripheral.*
- bool\_t **sdcConnect** (**SDCDriver** \*sdcp)  
*Performs the initialization procedure on the inserted card.*
- bool\_t **sdcDisconnect** (**SDCDriver** \*sdcp)  
*Brings the driver in a state safe for card removal.*
- bool\_t **sdcRead** (**SDCDriver** \*sdcp, uint32\_t startblk, uint8\_t \*buf, uint32\_t n)  
*Reads one or more blocks.*
- bool\_t **sdcWrite** (**SDCDriver** \*sdcp, uint32\_t startblk, const uint8\_t \*buf, uint32\_t n)  
*Writes one or more blocks.*
- bool\_t **\_sdc\_wait\_for\_transfer\_state** (**SDCDriver** \*sdcp)  
*Wait for the card to complete pending operations.*
- **CH\_IRQ\_HANDLER** (**SDIO IRQHandler**)  
*SDIO IRQ handler.*
- void **sdc\_lld\_init** (void)  
*Low level SDC driver initialization.*

- void `sdc_ll_start (SDCDriver *sdcp)`  
*Configures and activates the SDC peripheral.*
- void `sdc_ll_stop (SDCDriver *sdcp)`  
*Deactivates the SDC peripheral.*
- void `sdc_ll_start_clk (SDCDriver *sdcp)`  
*Starts the SDIO clock and sets it to init mode (400KHz or less).*
- void `sdc_ll_set_data_clk (SDCDriver *sdcp)`  
*Sets the SDIO clock to data mode (25MHz or less).*
- void `sdc_ll_stop_clk (SDCDriver *sdcp)`  
*Stops the SDIO clock.*
- void `sdc_ll_set_bus_mode (SDCDriver *sdcp, sdcbusmode_t mode)`  
*Switches the bus to 4 bits mode.*
- void `sdc_ll_send_cmd_none (SDCDriver *sdcp, uint8_t cmd, uint32_t arg)`  
*Sends an SDIO command with no response expected.*
- bool\_t `sdc_ll_send_cmd_short (SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp)`  
*Sends an SDIO command with a short response expected.*
- bool\_t `sdc_ll_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\_t `sdc_ll_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\_t `sdc_ll_read (SDCDriver *sdcp, uint32_t startblk, uint8_t *buf, uint32_t n)`  
*Reads one or more blocks.*
- bool\_t `sdc_ll_write (SDCDriver *sdcp, uint32_t startblk, const uint8_t *buf, uint32_t n)`  
*Writes one or more blocks.*

## Variables

- SDCDriver `SDCD1`  
*SDCD1 driver identifier.*

## SD card types

- #define `SDC_MODE_CARDTYPE_MASK` 0xF  
*Card type mask.*
- #define `SDC_MODE_CARDTYPE_SDV11` 0  
*Card is SD V1.1.*
- #define `SDC_MODE_CARDTYPE_SDV20` 1  
*Card is SD V2.0.*
- #define `SDC_MODE_CARDTYPE_MMC` 2  
*Card is MMC.*
- #define `SDC_MODE_HIGH_CAPACITY` 0x10  
*High cap.card.*

## SDC configuration options

- #define `SDC_INIT_RETRY` 100  
*Number of initialization attempts before rejecting the card.*
- #define `SDC_MMCSUPPORT` FALSE  
*Include support for MMC cards.*
- #define `SDC_NICE_WAITING` TRUE  
*Delays insertions.*

## R1 response utilities

- `#define SDC_R1_ERROR(r1) (((r1) & SDC_R1_ERROR_MASK) != 0)`  
*Evaluates to TRUE if the R1 response contains error flags.*
- `#define SDC_R1_STS(r1) (((r1) >> 9) & 15)`  
*Returns the status field of an R1 response.*
- `#define SDC_R1_IS_CARD_LOCKED(r1) (((r1) >> 21) & 1)`  
*Evaluates to TRUE if the R1 response indicates a locked card.*

## Macro Functions

- `#define sdcGetDriverState(sdcp) ((sdcp)->state)`  
*Returns the driver state.*
- `#define sdclsCardInserted(sdcp) (sdc_ll_is_card_inserted(sdcp))`  
*Returns the card insertion status.*
- `#define sdclsWriteProtected(sdcp) (sdc_ll_is_write_protected(sdcp))`  
*Returns the write protect status.*

## Configuration options

- `#define STM32_SDC_DATETIMEOUT 0x000FFFFF`  
*SDIO data timeout in SDIO clock cycles.*
- `#define STM32_SDC_SDIO_DMA_PRIORITY 3`  
*SDIO DMA priority (0..3|lowest..highest).*
- `#define STM32_SDC_SDIO_IRQ_PRIORITY 9`  
*SDIO interrupt priority level setting.*
- `#define STM32_SDC_UNALIGNED_SUPPORT TRUE`  
*SDIO support for unaligned transfers.*

## Defines

- `#define SDC_BLOCK_SIZE 512`
- `#define SDC_CMD8_PATTERN 0x000001AA`  
*Fixed pattern for CMD8.*
- `#define SDC_R1_ERROR_MASK 0xFDFFFE008`  
*Mask of error bits in R1 responses.*

## Typedefs

- `typedef uint32_t sdcmode_t`  
*Type of card flags.*
- `typedef struct SDCDriver SDCDriver`  
*Type of a structure representing an SDC driver.*

## Enumerations

- `enum sdcstate_t {`  
 `SDC_UNINIT = 0, SDC_STOP = 1, SDC_READY = 2, SDC_CONNECTING = 3,`  
 `SDC_DISCONNECTING = 4, SDC_ACTIVE = 5, SDC_READING = 6, SDC_WRITING = 7 }`  
*Driver state machine possible states.*
- `enum sdcbusmode_t`  
*Type of SDIO bus mode.*

## 6.14.4 Function Documentation

### 6.14.4.1 void sdclInit ( void )

SDC Driver initialization.

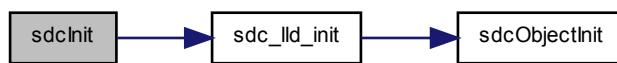
#### Note

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:



### 6.14.4.2 void sdcObjectInit ( SDCDriver \* sdc )

Initializes the standard part of a `SDCDriver` structure.

#### Parameters

out	<code>sdc</code> pointer to the <code>SDCDriver</code> object
-----	---

#### Function Class:

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

### 6.14.4.3 void sdcStart ( SDCDriver \* sdc, const SDCConfig \* config )

Configures and activates the SDC peripheral.

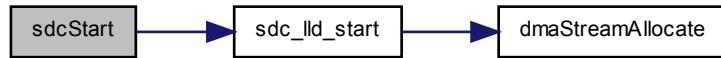
#### Parameters

in	<code>sdc</code> pointer to the <code>SDCDriver</code> object
in	<code>config</code> pointer to the <code>SDCConfig</code> object, can be <code>NULL</code> 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:



#### 6.14.4.4 void sdcStop ( **SDCDriver** \* *sdcP* )

Deactivates the SDC peripheral.

##### Parameters

in *sdcP* pointer to the **SDCDriver** object

##### Function Class:

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

Here is the call graph for this function:



#### 6.14.4.5 bool\_t sdcConnect ( **SDCDriver** \* *sdcP* )

Performs the initialization procedure on the inserted card.

This function should be invoked when a card is inserted and brings the driver in the **SDC\_ACTIVE** state where it is possible to perform read and write operations.

##### Parameters

in *sdcP* pointer to the **SDCDriver** object

##### Returns

The operation status.

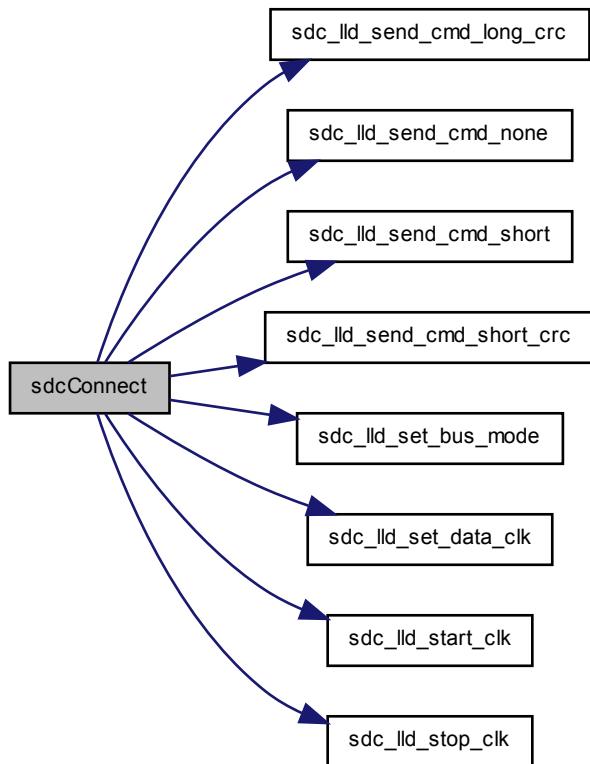
##### Return values

<i>FALSE</i>	operation succeeded, the driver is now in the <b>SDC_ACTIVE</b> state.
<i>TRUE</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:



#### 6.14.4.6 `bool_t sdcDisconnect( SDCDriver *sdcp )`

Brings the driver in a state safe for card removal.

**Parameters**

`in` `sdcp` pointer to the `SDCDriver` object

**Returns**

The operation status.

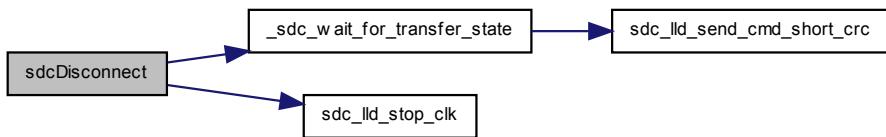
**Return values**

<code>FALSE</code>	the operation succeeded and the driver is now in the <code>SDC_READY</code> state.
<code>TRUE</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:



#### 6.14.4.7 `bool_t sdcRead ( SDCDriver * sdc, uint32_t startblk, uint8_t * buf, uint32_t n )`

Reads one or more blocks.

##### Precondition

The driver must be in the `SDC_ACTIVE` state after a successful `sdcConnect()` invocation.

##### Parameters

in	<code>sdc</code>	pointer to the <code>SDCDriver</code> object
in	<code>startblk</code>	first block to read
out	<code>buf</code>	pointer to the read buffer
in	<code>n</code>	number of blocks to read

##### Returns

The operation status.

##### Return values

<code>FALSE</code>	operation succeeded, the requested blocks have been read.
<code>TRUE</code>	operation failed, the state of the buffer is uncertain.

##### Function Class:

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

Here is the call graph for this function:



#### 6.14.4.8 `bool_t sdcWrite ( SDCDriver * sdc, uint32_t startblk, const uint8_t * buf, uint32_t n )`

Writes one or more blocks.

**Precondition**

The driver must be in the SDC\_ACTIVE state after a successful [sdcConnect\(\)](#) invocation.

**Parameters**

in	<i>sdcp</i>	pointer to the <a href="#">SDCDriver</a> 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.

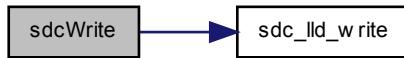
**Return values**

<i>FALSE</i>	operation succeeded, the requested blocks have been written.
<i>TRUE</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:

**6.14.4.9 bool\_t \_sdc\_wait\_for\_transfer\_state ( [SDCDriver](#) \* *sdcp* )**

Wait for the card to complete pending operations.

**Parameters**

in	<i>sdcp</i>	pointer to the <a href="#">SDCDriver</a> object
----	-------------	---

**Returns**

The operation status.

**Return values**

<i>FALSE</i>	the card is now in transfer state.
<i>TRUE</i>	an error occurred while waiting or the card is in an unexpected state.

**Function Class:**

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

Here is the call graph for this function:



#### 6.14.4.10 CH IRQ\_HANDLER ( SDIO IRQHandler )

SDIO IRQ handler.

##### Function Class:

Interrupt handler, this function should not be directly invoked.

#### 6.14.4.11 void sdc\_lld\_init ( void )

Low level SDC driver initialization.

##### Function Class:

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

Here is the call graph for this function:



#### 6.14.4.12 void sdc\_lld\_start ( SDCDriver \* sdcp )

Configures and activates the SDC peripheral.

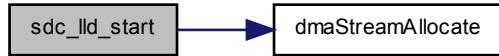
##### Parameters

in	<i>sdcp</i>	pointer to the <code>SDCDriver</code> object, must be NULL, this driver does not require any configuration
----	-------------	--

##### Function Class:

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

Here is the call graph for this function:



#### 6.14.4.13 void sdc\_lld\_stop ( SDCDriver \* sdc )

Deactivates the SDC peripheral.

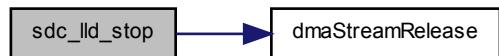
##### Parameters

in                    *sdc* pointer to the [SDCDriver](#) object

##### Function Class:

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

Here is the call graph for this function:



#### 6.14.4.14 void sdc\_lld\_start\_clk ( SDCDriver \* sdc )

Starts the SDIO clock and sets it to init mode (400KHz or less).

##### Parameters

in                    *sdc* pointer to the [SDCDriver](#) object

##### Function Class:

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

#### 6.14.4.15 void sdc\_lld\_set\_data\_clk ( SDCDriver \* sdc )

Sets the SDIO clock to data mode (25MHz or less).

##### Parameters

in                    *sdc* pointer to the [SDCDriver](#) object

**Function Class:**

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

**6.14.4.16 void sdc\_lld\_stop\_clk ( *SDCDriver* \* *sdcp* )**

Stops the SDIO clock.

**Parameters**

in *sdcp* pointer to the *SDCDriver* object

**Function Class:**

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

**6.14.4.17 void sdc\_lld\_set\_bus\_mode ( *SDCDriver* \* *sdcp*, *sdcbusmode\_t mode* )**

Switches the bus to 4 bits mode.

**Parameters**

in *sdcp* pointer to the *SDCDriver* object

in *mode* bus mode

**Function Class:**

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

**6.14.4.18 void sdc\_lld\_send\_cmd\_none ( *SDCDriver* \* *sdcp*, *uint8\_t cmd*, *uint32\_t arg* )**

Sends an SDIO command with no response expected.

**Parameters**

in *sdcp* pointer to the *SDCDriver* object

in *cmd* card command

in *arg* command argument

**Function Class:**

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

**6.14.4.19 bool\_t sdc\_lld\_send\_cmd\_short ( *SDCDriver* \* *sdcp*, *uint8\_t cmd*, *uint32\_t arg*, *uint32\_t \* resp* )**

Sends an SDIO command with a short response expected.

**Note**

The CRC is not verified.

**Parameters**

in *sdcp* pointer to the *SDCDriver* object

in *cmd* card command

in *arg* command argument

out *resp* pointer to the response buffer (one word)

**Returns**

The operation status.

**Return values**

<i>FALSE</i>	the operation succeeded.
<i>TRUE</i>	the operation failed because timeout, CRC check or other errors.

**Function Class:**

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

6.14.4.20 `bool_t sdc_lld_send_cmd_short_crc ( SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp )`

Sends an SDIO command with a short response expected and CRC.

**Parameters**

in	<i>sdcp</i>	pointer to the <code>SDCDriver</code> object
in	<i>cmd</i>	card command
in	<i>arg</i>	command argument
out	<i>resp</i>	pointer to the response buffer (one word)

**Returns**

The operation status.

**Return values**

<i>FALSE</i>	the operation succeeded.
<i>TRUE</i>	the operation failed because timeout, CRC check or other errors.

**Function Class:**

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

6.14.4.21 `bool_t sdc_lld_send_cmd_long_crc ( SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp )`

Sends an SDIO command with a long response expected and CRC.

**Parameters**

in	<i>sdcp</i>	pointer to the <code>SDCDriver</code> object
in	<i>cmd</i>	card command
in	<i>arg</i>	command argument
out	<i>resp</i>	pointer to the response buffer (four words)

**Returns**

The operation status.

**Return values**

<i>FALSE</i>	the operation succeeded.
<i>TRUE</i>	the operation failed because timeout, CRC check or other errors.

**Function Class:**

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

6.14.4.22 `bool_t sdc_lld_read ( SDCDriver * sdc, uint32_t startblk, uint8_t * buf, uint32_t n )`

Reads one or more blocks.

#### 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

*FALSE* operation succeeded, the requested blocks have been read.

*TRUE* operation failed, the state of the buffer is uncertain.

#### Function Class:

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

6.14.4.23 `bool_t sdc_lld_write ( SDCDriver * sdc, uint32_t startblk, const uint8_t * buf, uint32_t n )`

Writes one or more blocks.

#### Parameters

in	<i>sdc</i>	pointer to the <code>SDCDriver</code> object
in	<i>startblk</i>	first block to write
out	<i>buf</i>	pointer to the write buffer
in	<i>n</i>	number of blocks to write

#### Returns

The operation status.

#### Return values

*FALSE* operation succeeded, the requested blocks have been written.

*TRUE* operation failed.

#### Function Class:

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

## 6.14.5 Variable Documentation

### 6.14.5.1 SDCDriver SD\_CD1

SD\_CD1 driver identifier.

## 6.14.6 Define Documentation

### 6.14.6.1 #define SDC\_BLOCK\_SIZE 512

Fixed block size.

6.14.6.2 #define SDC\_CMD8\_PATTERN 0x000001AA

Fixed pattern for CMD8.

6.14.6.3 #define SDC\_MODE\_CARDTYPE\_MASK 0xF

Card type mask.

6.14.6.4 #define SDC\_MODE\_CARDTYPE\_SDV11 0

Card is SD V1.1.

6.14.6.5 #define SDC\_MODE\_CARDTYPE\_SDV20 1

Card is SD V2.0.

6.14.6.6 #define SDC\_MODE\_CARDTYPE\_MMC 2

Card is MMC.

6.14.6.7 #define SDC\_MODE\_HIGH\_CAPACITY 0x10

High cap.card.

6.14.6.8 #define SDC\_R1\_ERROR\_MASK 0xFDFFFE008

Mask of error bits in R1 responses.

6.14.6.9 #define SDC\_INIT\_RETRY 100

Number of initialization attempts before rejecting the card.

#### Note

Attempts are performed at 10mS intervals.

6.14.6.10 #define SDC\_MMCSUPPORT FALSE

Include support for MMC cards.

#### Note

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

6.14.6.11 #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.

6.14.6.12 #define SDC\_R1\_ERROR( *r1* ) (((*r1*) & SDC\_R1\_ERROR\_MASK) != 0)

Evaluates to TRUE if the R1 response contains error flags.

**Parameters**

in *r1* the r1 response

6.14.6.13 #define SDC\_R1\_STS( *r1* ) (((*r1*) >> 9) & 15)

Returns the status field of an R1 response.

**Parameters**

in *r1* the r1 response

6.14.6.14 #define SDC\_R1\_IS\_CARD\_LOCKED( *r1* ) (((*r1*) >> 21) & 1)

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

**Parameters**

in *r1* the r1 response

6.14.6.15 #define sdcGetDriverState( *sdcp* ) ((*sdcp*)->state)

Returns the driver state.

**Parameters**

in *sdcp* pointer to the [SDCDriver](#) object

**Returns**

The driver state.

**Function Class:**

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

6.14.6.16 #define sdclIsCardInserted( *sdcp* ) (sdcl\_lld\_is\_card\_inserted(*sdcp*))

Returns the card insertion status.

**Note**

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

**Parameters**

in *sdcp* pointer to the [SDCDriver](#) object

**Returns**

The card state.

**Return values**

*FALSE* card not inserted.  
*TRUE* card inserted.

**Function Class:**

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

6.14.6.17 #define sdclIsWriteProtected( *sdcp* )(sdc\_lld\_is\_write\_protected(*sdcp*))

Returns the write protect status.

**Note**

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

**Parameters**

in *sdcp* pointer to the [SDCDriver](#) object

**Returns**

The card state.

**Return values**

*FALSE* card not inserted.  
*TRUE* card inserted.

**Function Class:**

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

6.14.6.18 #define STM32\_SDC\_DATATIMEOUT 0x000FFFFF

SDIO data timeout in SDIO clock cycles.

6.14.6.19 #define STM32\_SDC\_SDIO\_DMA\_PRIORITY 3

SDIO DMA priority (0..3|lowest..highest).

6.14.6.20 #define STM32\_SDC\_SDIO\_IRQ\_PRIORITY 9

SDIO interrupt priority level setting.

6.14.6.21 #define STM32\_SDC\_UNALIGNED\_SUPPORT TRUE

SDIO support for unaligned transfers.

## 6.14.7 Typedef Documentation

6.14.7.1 `typedef uint32_t sdcmode_t`

Type of card flags.

### 6.14.7.2 `typedef struct SDCDriver SDCDriver`

Type of a structure representing an SDC driver.

## 6.14.8 Enumeration Type Documentation

### 6.14.8.1 `enum sdcstate_t`

Driver state machine possible states.

#### Enumerator:

***SDC\_UNINIT*** Not initialized.

***SDC\_STOP*** Stopped.

***SDC\_READY*** Ready.

***SDC\_CONNECTING*** Card connection in progress.

***SDC\_DISCONNECTING*** Card disconnection in progress.

***SDC\_ACTIVE*** Cart initialized.

***SDC\_READING*** Read operation in progress.

***SDC\_WRITING*** Write operation in progress.

### 6.14.8.2 `enum sdcbusmode_t`

Type of SDIO bus mode.

## 6.15 Serial Driver

### 6.15.1 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.

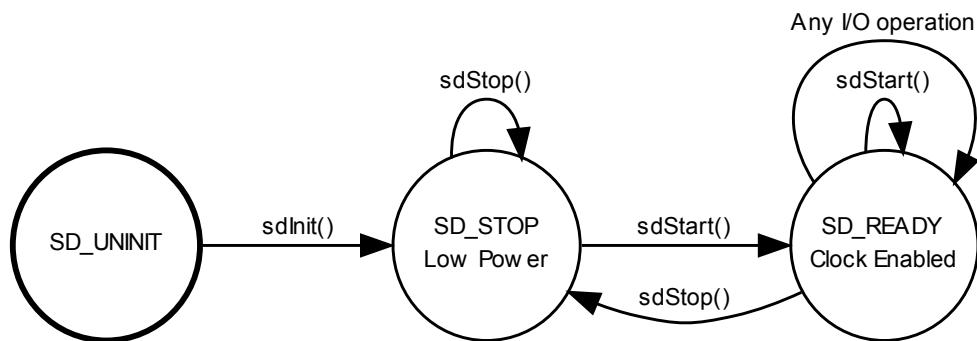
The module also contains functions that make the implementation of the interrupt service routines much easier.

#### Precondition

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

### 6.15.2 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).



## Data Structures

- struct [SerialDriverVMT](#)  
*SerialDriver virtual methods table.*
- struct [SerialDriver](#)  
*Full duplex serial driver class.*
- struct [SerialConfig](#)  
*STM32 Serial Driver configuration structure.*

## Functions

- void [sdInit](#) (void)  
*Serial Driver initialization.*
- void [sdObjectInit](#) (SerialDriver \*sdp, qnotify\_t inotify, qnotify\_t onotify)  
*Initializes a generic full duplex driver object.*
- void [sdStart](#) (SerialDriver \*sdp, const SerialConfig \*config)  
*Configures and starts the driver.*
- void [sdStop](#) (SerialDriver \*sdp)  
*Stops the driver.*
- void [sdIncomingData](#) (SerialDriver \*sdp, uint8\_t b)  
*Handles incoming data.*
- msg\_t [sdRequestData](#) (SerialDriver \*sdp)  
*Handles outgoing data.*
- [CH\\_IRQ\\_HANDLER](#) (USART1\_IRQHandler)  
*USART1 interrupt handler.*
- [CH\\_IRQ\\_HANDLER](#) (USART2\_IRQHandler)  
*USART2 interrupt handler.*
- [CH\\_IRQ\\_HANDLER](#) (USART3\_IRQHandler)  
*USART3 interrupt handler.*
- [CH\\_IRQ\\_HANDLER](#) (UART4\_IRQHandler)  
*UART4 interrupt handler.*
- [CH\\_IRQ\\_HANDLER](#) (UART5\_IRQHandler)

- **CH\_IRQ\_HANDLER** (USART6\_IRQHandler)  
*USART1 interrupt handler.*
- 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.*
- **SerialDriver SD2**  
*USART2 serial driver identifier.*
- **SerialDriver SD3**  
*USART3 serial driver identifier.*
- **SerialDriver SD4**  
*UART4 serial driver identifier.*
- **SerialDriver SD5**  
*UART5 serial driver identifier.*
- **SerialDriver SD6**  
*USART6 serial driver identifier.*

## Serial status flags

- #define **SD\_PARITY\_ERROR** 32  
*Parity error happened.*
- #define **SD\_FRAMING\_ERROR** 64  
*Framing error happened.*
- #define **SD\_OVERRUN\_ERROR** 128  
*Overflow happened.*
- #define **SD\_NOISE\_ERROR** 256  
*Noise on the line.*
- #define **SD\_BREAK\_DETECTED** 512  
*Break detected.*

## Serial configuration options

- #define **SERIAL\_DEFAULT\_BITRATE** 38400  
*Default bit rate.*
- #define **SERIAL\_BUFFERS\_SIZE** 16  
*Serial buffers size.*

## Macro Functions

- `#define sdPutWouldBlock(sdp) chOQIsFull(&(sdp)->oqueue)`  
*Direct output check on a `SerialDriver`.*
- `#define sdGetWouldBlock(sdp) chIQIsEmpty(&(sdp)->iqueue)`  
*Direct input check on a `SerialDriver`.*
- `#define sdPut(sdp, b) chOQPut(&(sdp)->oqueue, b)`  
*Direct write to a `SerialDriver`.*
- `#define sdPutTimeout(sdp, b, t) chOQPutTimeout(&(sdp)->oqueue, b, t)`  
*Direct write to a `SerialDriver` with timeout specification.*
- `#define sdGet(sdp) chIQGet(&(sdp)->iqueue)`  
*Direct read from a `SerialDriver`.*
- `#define sdGetTimeout(sdp, t) chIQGetTimeout(&(sdp)->iqueue, t)`  
*Direct read from a `SerialDriver` with timeout specification.*
- `#define sdWrite(sdp, b, n) chOQWriteTimeout(&(sdp)->oqueue, b, n, TIME_INFINITE)`  
*Direct blocking write to a `SerialDriver`.*
- `#define sdWriteTimeout(sdp, b, n, t) chOQWriteTimeout(&(sdp)->oqueue, b, n, t)`  
*Direct blocking write to a `SerialDriver` with timeout specification.*
- `#define sdAsynchronousWrite(sdp, b, n) chOQWriteTimeout(&(sdp)->oqueue, b, n, TIME_IMMEDIATE)`  
*Direct non-blocking write to a `SerialDriver`.*
- `#define sdRead(sdp, b, n) chIQReadTimeout(&(sdp)->iqueue, b, n, TIME_INFINITE)`  
*Direct blocking read from a `SerialDriver`.*
- `#define sdReadTimeout(sdp, b, n, t) chIQReadTimeout(&(sdp)->iqueue, b, n, t)`  
*Direct blocking read from a `SerialDriver` with timeout specification.*
- `#define sdAsynchronousRead(sdp, b, n) chIQReadTimeout(&(sdp)->iqueue, b, n, TIME_IMMEDIATE)`  
*Direct non-blocking read from a `SerialDriver`.*

## Configuration options

- `#define STM32_SERIAL_USE_USART1 TRUE`  
*USART1 driver enable switch.*
- `#define STM32_SERIAL_USE_USART2 TRUE`  
*USART2 driver enable switch.*
- `#define STM32_SERIAL_USE_USART3 TRUE`  
*USART3 driver enable switch.*
- `#define STM32_SERIAL_USE_UART4 TRUE`  
*UART4 driver enable switch.*
- `#define STM32_SERIAL_USE_UART5 TRUE`  
*UART5 driver enable switch.*
- `#define STM32_SERIAL_USE_USART6 TRUE`  
*USART6 driver enable switch.*
- `#define STM32_SERIAL_USART1_PRIORITY 12`  
*USART1 interrupt priority level setting.*
- `#define STM32_SERIAL_USART2_PRIORITY 12`  
*USART2 interrupt priority level setting.*
- `#define STM32_SERIAL_USART3_PRIORITY 12`  
*USART3 interrupt priority level setting.*
- `#define STM32_SERIAL_UART4_PRIORITY 12`  
*UART4 interrupt priority level setting.*
- `#define STM32_SERIAL_UART5_PRIORITY 12`  
*UART5 interrupt priority level setting.*
- `#define STM32_SERIAL_USART6_PRIORITY 12`  
*USART6 interrupt priority level setting.*

## Defines

- `#define _serial_driver_methods _base_asynchronous_channel_methods`  
`SerialDriver specific methods.`
- `#define _serial_driver_data`  
`SerialDriver specific data.`
- `#define USART_CR2_STOP1_BITS (0 << 12)`  
`CR2 1 stop bit value.`
- `#define USART_CR2_STOP0P5_BITS (1 << 12)`  
`CR2 0.5 stop bit value.`
- `#define USART_CR2_STOP2_BITS (2 << 12)`  
`CR2 2 stop bit value.`
- `#define USART_CR2_STOP1P5_BITS (3 << 12)`  
`CR2 1.5 stop bit value.`

## Typedefs

- `typedef struct SerialDriver SerialDriver`  
`Structure representing a serial driver.`

## Enumerations

- `enum sdstate_t { SD_UNINIT = 0, SD_STOP = 1, SD_READY = 2 }`  
`Driver state machine possible states.`

### 6.15.3 Function Documentation

#### 6.15.3.1 void sdInit( void )

Serial Driver initialization.

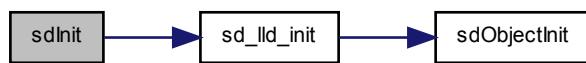
##### Note

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:



### 6.15.3.2 void sdObjectInit ( *SerialDriver* \* *sdp*, *qnotify\_t* *inotify*, *qnotify\_t* *onotify* )

Initializes a generic full duplex driver object.

The HW dependent part of the initialization has to be performed outside, usually in the hardware initialization code.

#### Parameters

out	<i>sdp</i>	pointer to a <i>SerialDriver</i> structure
in	<i>inotify</i>	pointer to a callback function that is invoked when some data is read from the Queue. The value can be NULL.
in	<i>onotify</i>	pointer to a callback function that is invoked when some data is written in the Queue. The value can be NULL.

#### Function Class:

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

### 6.15.3.3 void sdStart ( *SerialDriver* \* *sdp*, *const SerialConfig* \* *config* )

Configures and starts the driver.

#### Parameters

in	<i>sdp</i>	pointer to a <i>SerialDriver</i> object
in	<i>config</i>	the architecture-dependent serial driver configuration. If this parameter is set to NULL then a default configuration is used.

#### Function Class:

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

Here is the call graph for this function:



### 6.15.3.4 void sdStop ( *SerialDriver* \* *sdp* )

Stops the driver.

Any thread waiting on the driver's queues will be awakened with the message Q\_RESET.

#### Parameters

in	<i>sdp</i>	pointer to a <i>SerialDriver</i> 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:



#### 6.15.3.5 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 <i>SerialDriver</i> structure
in	<i>b</i>	the byte to be written in the driver's Input Queue

##### Function Class:

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

#### 6.15.3.6 msg\_t sdRequestData ( *SerialDriver* \* *sdp* )

Handles outgoing data.

Must be called from the output interrupt service routine in order to get the next byte to be transmitted.

##### Note

In order to gain some performance it is suggested to not use this function directly but copy this code directly into the interrupt service routine.

##### Parameters

in	<i>sdp</i>	pointer to a <i>SerialDriver</i> structure
----	------------	--

##### Returns

The byte value read from the driver's output queue.

##### Return values

*Q\_EMPTY* 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.

**6.15.3.7 CH\_IRQ\_HANDLER ( USART1\_IRQHandler )**

USART1 interrupt handler.

**Function Class:**

Interrupt handler, this function should not be directly invoked.

**6.15.3.8 CH\_IRQ\_HANDLER ( USART2\_IRQHandler )**

USART2 interrupt handler.

**Function Class:**

Interrupt handler, this function should not be directly invoked.

**6.15.3.9 CH\_IRQ\_HANDLER ( USART3\_IRQHandler )**

USART3 interrupt handler.

**Function Class:**

Interrupt handler, this function should not be directly invoked.

**6.15.3.10 CH\_IRQ\_HANDLER ( UART4\_IRQHandler )**

UART4 interrupt handler.

**Function Class:**

Interrupt handler, this function should not be directly invoked.

**6.15.3.11 CH\_IRQ\_HANDLER ( UART5\_IRQHandler )**

UART5 interrupt handler.

**Function Class:**

Interrupt handler, this function should not be directly invoked.

**6.15.3.12 CH\_IRQ\_HANDLER ( USART6\_IRQHandler )**

USART1 interrupt handler.

**Function Class:**

Interrupt handler, this function should not be directly invoked.

**6.15.3.13 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:

**6.15.3.14 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 <a href="#">SerialDriver</a> 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.

**6.15.3.15 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 <a href="#">SerialDriver</a> object
----	------------	--

**Function Class:**

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

**6.15.4 Variable Documentation****6.15.4.1 SerialDriver SD1**

USART1 serial driver identifier.

#### 6.15.4.2 SerialDriver SD2

USART2 serial driver identifier.

#### 6.15.4.3 SerialDriver SD3

USART3 serial driver identifier.

#### 6.15.4.4 SerialDriver SD4

UART4 serial driver identifier.

#### 6.15.4.5 SerialDriver SD5

UART5 serial driver identifier.

#### 6.15.4.6 SerialDriver SD6

USART6 serial driver identifier.

### 6.15.5 Define Documentation

#### 6.15.5.1 #define SD\_PARITY\_ERROR 32

Parity error happened.

#### 6.15.5.2 #define SD\_FRAMING\_ERROR 64

Framing error happened.

#### 6.15.5.3 #define SD\_OVERRUN\_ERROR 128

Overflow happened.

#### 6.15.5.4 #define SD\_NOISE\_ERROR 256

Noise on the line.

#### 6.15.5.5 #define SD\_BREAK\_DETECTED 512

Break detected.

#### 6.15.5.6 #define SERIAL\_DEFAULT\_BITRATE 38400

Default bit rate.

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

**6.15.5.7 #define SERIAL\_BUFFERS\_SIZE 16**

Serial buffers size.

Configuration parameter, you can change the depth of the queue buffers depending on the requirements of your application.

**Note**

The default is 16 bytes for both the transmission and receive buffers.

**6.15.5.8 #define \_serial\_driver\_methods \_base\_asynchronous\_channel\_methods**

[SerialDriver](#) specific methods.

**6.15.5.9 #define sdPutWouldBlock( sdp ) chOQIsFull(&(sdp)->oqueue)**

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.

**See also**

[chIOPutWouldBlock\(\)](#)

**Deprecated****Function Class:**

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

**6.15.5.10 #define sdGetWouldBlock( sdp ) chIQIsEmpty(&(sdp)->iqueue)**

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.

**See also**

[chIOWGetWouldBlock\(\)](#)

**Deprecated****Function Class:**

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

6.15.5.11 #define sdPut( *sdp*, *b* ) chOQPut(&(*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

[chIOPut\(\)](#)

#### Function Class:

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

6.15.5.12 #define sdPutTimeout( *sdp*, *b*, *t* ) chOQPutTimeout(&(*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

[chIOPutTimeout\(\)](#)

#### Function Class:

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

6.15.5.13 #define sdGet( *sdp* ) chIQGet(&(*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

[chIOWGet\(\)](#)

#### Function Class:

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

6.15.5.14 #define sdGetTimeout( *sdp*, *t* ) chIQGetTimeout(&(*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**

`chIOGetTimeout()`

**Function Class:**

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

**6.15.5.15 #define sdWrite( sdp, b, n ) chOQWriteTimeout(&(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**

`chIOWriteTimeout()`

**Function Class:**

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

**6.15.5.16 #define sdWriteTimeout( sdp, b, n, t ) chOQWriteTimeout(&(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**

`chIOWriteTimeout()`

**Function Class:**

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

**6.15.5.17 #define sdAsynchronousWrite( sdp, b, n ) chOQWriteTimeout(&(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**

`chIOWriteTimeout()`

**Function Class:**

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

```
6.15.5.18 #define sdRead( sdp, b, n ) chIQReadTimeout(&(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

[chIOReadTimeout\(\)](#)

#### Function Class:

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

```
6.15.5.19 #define sdReadTimeout( sdp, b, n, t ) chIQReadTimeout(&(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

[chIOReadTimeout\(\)](#)

#### Function Class:

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

```
6.15.5.20 #define sdAsynchronousRead( sdp, b, n ) chIQReadTimeout(&(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

[chIOReadTimeout\(\)](#)

#### Function Class:

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

```
6.15.5.21 #define STM32_SERIAL_USE_USART1 TRUE
```

USART1 driver enable switch.

If set to TRUE the support for USART1 is included.

#### Note

The default is TRUE.

6.15.5.22 #define STM32\_SERIAL\_USE\_USART2 TRUE

USART2 driver enable switch.

If set to TRUE the support for USART2 is included.

**Note**

The default is TRUE.

6.15.5.23 #define STM32\_SERIAL\_USE\_USART3 TRUE

USART3 driver enable switch.

If set to TRUE the support for USART3 is included.

**Note**

The default is TRUE.

6.15.5.24 #define STM32\_SERIAL\_USE\_UART4 TRUE

UART4 driver enable switch.

If set to TRUE the support for UART4 is included.

**Note**

The default is TRUE.

6.15.5.25 #define STM32\_SERIAL\_USE\_UART5 TRUE

UART5 driver enable switch.

If set to TRUE the support for UART5 is included.

**Note**

The default is TRUE.

6.15.5.26 #define STM32\_SERIAL\_USE\_USART6 TRUE

USART6 driver enable switch.

If set to TRUE the support for USART6 is included.

**Note**

The default is TRUE.

6.15.5.27 #define STM32\_SERIAL\_USART1\_PRIORITY 12

USART1 interrupt priority level setting.

6.15.5.28 #define STM32\_SERIAL\_USART2\_PRIORITY 12)

USART2 interrupt priority level setting.

6.15.5.29 #define STM32\_SERIAL\_USART3\_PRIORITY 12

USART3 interrupt priority level setting.

6.15.5.30 #define STM32\_SERIAL\_UART4\_PRIORITY 12

UART4 interrupt priority level setting.

6.15.5.31 #define STM32\_SERIAL\_UART5\_PRIORITY 12

UART5 interrupt priority level setting.

6.15.5.32 #define STM32\_SERIAL\_USART6\_PRIORITY 12

USART6 interrupt priority level setting.

6.15.5.33 #define \_serial\_driver\_data

**Value:**

```
_base_asynchronous_channel_data
/* Driver state.*/
sdstate_t           state;
/* Input queue.*/
InputQueue          iqueue;
/* Output queue.*/
OutputQueue         oqueue;
/* Input circular buffer.*/
uint8_t             ib[SERIAL_BUFFERS_SIZE];
/* Output circular buffer.*/
uint8_t             ob[SERIAL_BUFFERS_SIZE];
/* End of the mandatory fields.*/
/* Pointer to the USART registers block.*/
USART_TypeDef      *usart;
```

[SerialDriver](#) specific data.

6.15.5.34 #define USART\_CR2\_STOP1\_BITS (0 << 12)

CR2 1 stop bit value.

6.15.5.35 #define USART\_CR2\_STOP0P5\_BITS (1 << 12)

CR2 0.5 stop bit value.

6.15.5.36 #define USART\_CR2\_STOP2\_BITS (2 << 12)

CR2 2 stop bit value.

6.15.5.37 #define USART\_CR2\_STOP1P5\_BITS (3 << 12)

CR2 1.5 stop bit value.

### 6.15.6 Typedef Documentation

#### 6.15.6.1 `typedef struct SerialDriver SerialDriver`

Structure representing a serial driver.

### 6.15.7 Enumeration Type Documentation

#### 6.15.7.1 `enum sdstate_t`

Driver state machine possible states.

#### Enumerator:

**`SD_UNINIT`** Not initialized.

**`SD_STOP`** Stopped.

**`SD_READY`** Ready.

## 6.16 Serial over USB Driver

### 6.16.1 Detailed Description

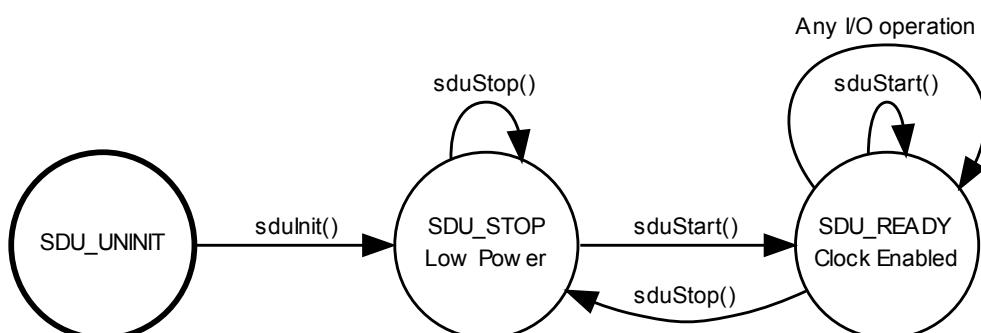
Serial over USB Driver. This module implements an USB Communication Device Class (CDC) as a normal serial communication port accessible from the device application.

#### Precondition

In order to use the USB over Serial driver the `HAL_USE_SERIAL_USB` option must be enabled in `halconf.h`.

### 6.16.2 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).



## Data Structures

- struct `SerialUSBConfig`  
*Serial over USB Driver configuration structure.*
- struct `SerialUSBDriverVMT`  
*SerialDriver virtual methods table.*
- struct `SerialUSBDriver`  
*Full duplex serial driver class.*

## 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.*
- `bool_t` `sduRequestsHook` (`USBDriver` \*`usbp`)  
*Default requests hook.*
- void `sduDataTransmitted` (`USBDriver` \*`usbp`, `usbep_t` `ep`)  
*Default data transmitted callback.*
- void `sduDataReceived` (`USBDriver` \*`usbp`, `usbep_t` `ep`)  
*Default data received callback.*
- void `sduInterruptTransmitted` (`USBDriver` \*`usbp`, `usbep_t` `ep`)  
*Default data received callback.*

## SERIAL\_USB configuration options

- `#define SERIAL_USB_BUFFERS_SIZE 64`  
*Serial over USB buffers size.*

## Defines

- `#define _serial_usb_driver_data`  
*SerialDriver specific data.*
- `#define _serial_usb_driver_methods` `_base_asynchronous_channel_methods`  
*SerialUSBDriver specific methods.*

## Typedefs

- `typedef struct SerialUSBDriver SerialUSBDriver`  
*Structure representing a serial over USB driver.*

## Enumerations

- enum `sdustate_t` { `SDU_UNINIT` = 0, `SDU_STOP` = 1, `SDU_READY` = 2 }  
*Driver state machine possible states.*

### 6.16.3 Function Documentation

#### 6.16.3.1 void sduInit ( void )

Serial Driver initialization.

##### Note

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

##### Function Class:

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

#### 6.16.3.2 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	<code>sdup</code> pointer to a <code>SerialUSBDriver</code> structure
-----	---

##### Function Class:

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

#### 6.16.3.3 void sduStart ( `SerialUSBDriver *sdup, const SerialUSBConfig *config` )

Configures and starts the driver.

##### Parameters

in	<code>sdup</code> pointer to a <code>SerialUSBDriver</code> object
in	<code>config</code> the serial over USB driver configuration

##### Function Class:

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

Here is the call graph for this function:



#### 6.16.3.4 void sduStop ( `SerialUSBDriver *sdup` )

Stops the driver.

Any thread waiting on the driver's queues will be awakened with the message `Q_RESET`.

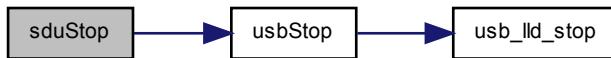
**Parameters**

in *sduP* pointer to a `SerialUSBDriver` object

**Function Class:**

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

Here is the call graph for this function:

**6.16.3.5 bool\_t sduRequestsHook ( USBDriver \* *usbp* )**

Default requests hook.

Applications wanting to use the Serial over USB driver can use this function as requests hook in the USB configuration. The following requests are emulated:

- `CDC_GET_LINE_CODING`.
- `CDC_SET_LINE_CODING`.
- `CDC_SET_CONTROL_LINE_STATE`.

**Parameters**

in *usbp* pointer to the `USBDriver` object

**Returns**

The hook status.

**Return values**

`TRUE` Message handled internally.

`FALSE` Message not handled.

**6.16.3.6 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 <code>USBDriver</code> object
in	<i>ep</i>	endpoint number

Here is the call graph for this function:



#### 6.16.3.7 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 <b>USBDriver</b> object
in	<i>ep</i>	endpoint number

Here is the call graph for this function:



#### 6.16.3.8 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 <b>USBDriver</b> object
in	<i>ep</i>	endpoint number

## 6.16.4 Define Documentation

### 6.16.4.1 #define SERIAL\_USB\_BUFFERS\_SIZE 64

Serial over USB buffers size.

Configuration parameter, the buffer size must be a multiple of the USB data endpoint maximum packet size.

**Note**

The default is 64 bytes for both the transmission and receive buffers.

**6.16.4.2 #define \_serial\_usb\_driver\_data****Value:**

```
_base_asynchronous_channel_data
/* Driver state.*/
sdustate_t           state;
/* Input queue.*/
InputQueue            iqueue;
/* Output queue.*/
OutputQueue           oqueue;
/* Input buffer.*/
uint8_t                ib[SERIAL_USB_BUFFERS_SIZE];
/* Output buffer.*/
uint8_t                ob[SERIAL_USB_BUFFERS_SIZE];
/* End of the mandatory fields.*/
/* Current configuration data.*/
const SerialUSBConfig *config;
```

[SerialDriver](#) specific data.

**6.16.4.3 #define \_serial\_usb\_driver\_methods \_base\_asynchronous\_channel\_methods**

[SerialUSBDriver](#) specific methods.

**6.16.5 Typedef Documentation****6.16.5.1 typedef struct SerialUSBDriver SerialUSBDriver**

Structure representing a serial over USB driver.

**6.16.6 Enumeration Type Documentation****6.16.6.1 enum sdustate\_t**

Driver state machine possible states.

**Enumerator:**

**SDU\_UNINIT** Not initialized.

**SDU\_STOP** Stopped.

**SDU\_READY** Ready.

**6.17 SPI Driver****6.17.1 Detailed Description**

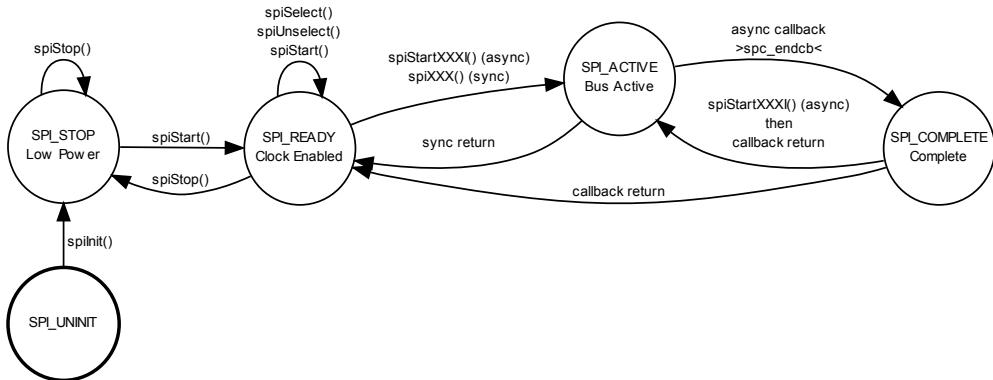
Generic SPI Driver. This module implements a generic SPI (Serial Peripheral Interface) driver allowing bidirectional and monodirectional transfers, complex atomic transactions are supported as well.

**Precondition**

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

### 6.17.2 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.

## Data Structures

- struct `SPICfg`  
*Driver configuration structure.*
- struct `SPIDrv`  
*Structure representing a SPI driver.*

## Functions

- void `spiInit (void)`  
*SPI Driver initialization.*
- void `spiObjectInit (SPIDrv *spip)`  
*Initializes the standard part of a `SPIDrv` structure.*
- void `spiStart (SPIDrv *spip, const SPICfg *config)`  
*Configures and activates the SPI peripheral.*
- void `spiStop (SPIDrv *spip)`  
*Deactivates the SPI peripheral.*
- void `spiSelect (SPIDrv *spip)`  
*Asserts the slave select signal and prepares for transfers.*
- void `spiUnselect (SPIDrv *spip)`  
*Deasserts the slave select signal.*
- void `spiStartIgnore (SPIDrv *spip, size_t n)`  
*Ignores data on the SPI bus.*
- void `spiStartExchange (SPIDrv *spip, size_t n, const void *txbuf, void *rxbuf)`  
*Exchanges data on the SPI bus.*

- void **spiStartSend** (SPIDriver \*spip, size\_t n, const void \*txbuf)  
*Sends data over the SPI bus.*
- void **spiStartReceive** (SPIDriver \*spip, size\_t n, void \*rxbuf)  
*Receives data from the SPI bus.*
- void **spilgnore** (SPIDriver \*spip, size\_t n)  
*Ignores data on the SPI bus.*
- void **spiExchange** (SPIDriver \*spip, size\_t n, const void \*txbuf, void \*rxbuf)  
*Exchanges data on the SPI bus.*
- void **spiSend** (SPIDriver \*spip, size\_t n, const void \*txbuf)  
*Sends data over the SPI bus.*
- void **spiReceive** (SPIDriver \*spip, size\_t n, void \*rxbuf)  
*Receives data from the SPI bus.*
- void **spiAcquireBus** (SPIDriver \*spip)  
*Gains exclusive access to the SPI bus.*
- void **spiReleaseBus** (SPIDriver \*spip)  
*Releases exclusive access to the SPI bus.*
- 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.*
- SPIDriver **SPID2**  
*SPI2 driver identifier.*
- SPIDriver **SPID3**  
*SPI3 driver identifier.*

## SPI configuration options

- #define **SPI\_USE\_WAIT** TRUE  
*Enables synchronous APIs.*
- #define **SPI\_USE\_MUTUAL\_EXCLUSION** TRUE  
*Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.*

## Macro Functions

- `#define spiSelectI(spip)`  
*Asserts the slave select signal and prepares for transfers.*
- `#define spiUnselectI(spip)`  
*Deasserts the slave select signal.*
- `#define spiStartIgnoreI(spip, n)`  
*Ignores data on the SPI bus.*
- `#define spiStartExchangeI(spip, n, txbuf, rxbuf)`  
*Exchanges data on the SPI bus.*
- `#define spiStartSendI(spip, n, txbuf)`  
*Sends data over the SPI bus.*
- `#define spiStartReceiveI(spip, n, rxbuf)`  
*Receives data from the SPI bus.*
- `#define spiPolledExchangeI(spip, frame) spi_lld_polled_exchange(spip, frame)`  
*Exchanges one frame using a polled wait.*

## Low Level driver helper macros

- `#define _spi_wait_s(spip)`  
*Waits for operation completion.*
- `#define _spi_wakeup_isr(spip)`  
*Wakes up the waiting thread.*
- `#define _spi_isr_code(spip)`  
*Common ISR code.*

## Configuration options

- `#define STM32_SPI_USE_SPI1 TRUE`  
*SPI1 driver enable switch.*
- `#define STM32_SPI_USE_SPI2 TRUE`  
*SPI2 driver enable switch.*
- `#define STM32_SPI_USE_SPI3 FALSE`  
*SPI3 driver enable switch.*
- `#define STM32_SPI_SPI1_IRQ_PRIORITY 10`  
*SPI1 interrupt priority level setting.*
- `#define STM32_SPI_SPI2_IRQ_PRIORITY 10`  
*SPI2 interrupt priority level setting.*
- `#define STM32_SPI_SPI3_IRQ_PRIORITY 10`  
*SPI3 interrupt priority level setting.*
- `#define STM32_SPI_SPI1_DMA_PRIORITY 1`  
*SPI1 DMA priority (0..3|lowest..highest).*
- `#define STM32_SPI_SPI2_DMA_PRIORITY 1`  
*SPI2 DMA priority (0..3|lowest..highest).*
- `#define STM32_SPI_SPI3_DMA_PRIORITY 1`  
*SPI3 DMA priority (0..3|lowest..highest).*
- `#define STM32_SPI_DMA_ERROR_HOOK(spip) chSysHalt()`  
*SPI DMA error hook.*
- `#define STM32_SPI_SPI1_RX_DMA_STREAM STM32_DMA_STREAM_ID(2, 0)`  
*DMA stream used for SPI1 RX operations.*
- `#define STM32_SPI_SPI1_TX_DMA_STREAM STM32_DMA_STREAM_ID(2, 3)`

- `#define STM32_SPI_SPI2_RX_DMA_STREAM STM32_DMA_STREAM_ID(1, 3)`  
*DMA stream used for SPI2 RX operations.*
- `#define STM32_SPI_SPI2_TX_DMA_STREAM STM32_DMA_STREAM_ID(1, 4)`  
*DMA stream used for SPI2 TX operations.*
- `#define STM32_SPI_SPI3_RX_DMA_STREAM STM32_DMA_STREAM_ID(1, 0)`  
*DMA stream used for SPI3 RX operations.*
- `#define STM32_SPI_SPI3_TX_DMA_STREAM STM32_DMA_STREAM_ID(1, 7)`  
*DMA stream used for SPI3 TX operations.*

## Typedefs

- `typedef struct SPIDriver SPIDriver`  
*Type of a structure representing an SPI driver.*
- `typedef void(* spicallback_t )(SPIDriver *spip)`  
*SPI notification callback type.*

## Enumerations

- `enum spistate_t {`  
`SPI_UNINIT = 0, SPI_STOP = 1, SPI_READY = 2, SPI_ACTIVE = 3,`  
`SPI_COMPLETE = 4 }`  
*Driver state machine possible states.*

### 6.17.3 Function Documentation

#### 6.17.3.1 void spilinit ( void )

SPI Driver initialization.

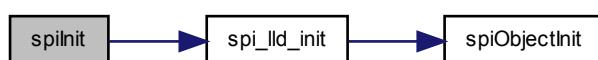
#### Note

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

#### Function Class:

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

Here is the call graph for this function:



### 6.17.3.2 void spiObjectInit ( **SPIDriver** \* *spip* )

Initializes the standard part of a **SPIDriver** structure.

#### Parameters

out	<i>spip</i> pointer to the <b>SPIDriver</b> object
-----	--

#### Function Class:

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

### 6.17.3.3 void spiStart ( **SPIDriver** \* *spip*, const **SPIConfig** \* *config* )

Configures and activates the SPI peripheral.

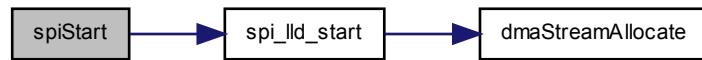
#### Parameters

in	<i>spip</i> pointer to the <b>SPIDriver</b> object
in	<i>config</i> pointer to the <b>SPIConfig</b> 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:



### 6.17.3.4 void spiStop ( **SPIDriver** \* *spip* )

Deactivates the SPI peripheral.

#### Note

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

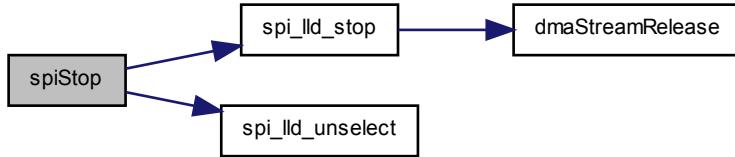
#### Parameters

in	<i>spip</i> pointer to the <b>SPIDriver</b> 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:



#### 6.17.3.5 void spiSelect ( *SPIDriver* \* *spip* )

Asserts the slave select signal and prepares for transfers.

##### Parameters

in *spip* pointer to the *SPIDriver* object

##### Function Class:

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

#### 6.17.3.6 void spiUnselect ( *SPIDriver* \* *spip* )

Deasserts the slave select signal.

The previously selected peripheral is unselected.

##### Parameters

in *spip* pointer to the *SPIDriver* object

##### Function Class:

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

#### 6.17.3.7 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 *spip* pointer to the *SPIDriver* object

in *n* number of words to be ignored

**Function Class:**

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

**6.17.3.8 void spiStartExchange ( *SPIDriver \* spip*, *size\_t n*, *const void \* txbuf*, *void \* rdbuf* )**

Exchanges data on the SPI bus.

This asynchronous function starts a simultaneous transmit/receive operation.

**Precondition**

A slave must have been selected using [spiSelect\(\)](#) or [spiSelectI\(\)](#).

**Postcondition**

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

**Note**

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

**Parameters**

in	<i>spip</i>	pointer to the <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>rdbuf</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.

**6.17.3.9 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 <code>SPIDriver</code> object
in	<i>n</i>	number of words to send
in	<i>txbuf</i>	the pointer to the transmit buffer

**Function Class:**

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

### 6.17.3.10 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 <b>SPIDriver</b> 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.

### 6.17.3.11 void spilgnore ( **SPIDriver** \* *spip*, **size\_t** *n* )

Ignores data on the SPI bus.

This synchronous function performs the transmission of a series of idle words on the SPI bus and ignores the received data.

#### Precondition

In order to use this function the option **SPI\_USE\_WAIT** must be enabled.

In order to use this function the driver must have been configured without callbacks (*end\_cb* = NULL).

#### Parameters

in	<i>spip</i>	pointer to the <b>SPIDriver</b> 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.

### 6.17.3.12 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 <code>SPIDriver</code> object
in	<i>n</i>	number of words to be exchanged
in	<i>txbuf</i>	the pointer to the transmit buffer
out	<i>rxbuf</i>	the pointer to the receive buffer

**Function Class:**

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

**6.17.3.13 void spiSend ( `SPIDriver * spip, size_t n, const void * txbuf` )**

Sends data over the SPI bus.

This synchronous function performs a transmit operation.

**Precondition**

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

In order to use this function the driver must have been configured without callbacks (`end_cb = NULL`).

**Note**

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

**Parameters**

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>n</i>	number of words to send
in	<i>txbuf</i>	the pointer to the transmit buffer

**Function Class:**

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

**6.17.3.14 void spiReceive ( `SPIDriver * spip, size_t n, void * rdbuf` )**

Receives data from the SPI bus.

This synchronous function performs a receive operation.

**Precondition**

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

In order to use this function the driver must have been configured without callbacks (`end_cb = NULL`).

**Note**

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

**Parameters**

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>n</i>	number of words to receive
out	<i>rdbuf</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.

**6.17.3.15 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 *spip* pointer to the [SPIDriver](#) object

**Function Class:**

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

**6.17.3.16 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 *spip* pointer to the [SPIDriver](#) object

**Function Class:**

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

**6.17.3.17 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:



**6.17.3.18 void spi\_lld\_start ( SPIDriver \* spip )**

Configures and activates the SPI peripheral.

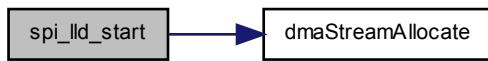
**Parameters**

in                    *spip* pointer to the [SPIDriver](#) object

**Function Class:**

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

Here is the call graph for this function:

**6.17.3.19 void spi\_lld\_stop ( SPIDriver \* spip )**

Deactivates the SPI peripheral.

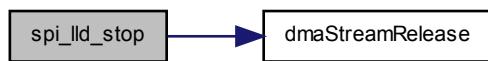
**Parameters**

in                    *spip* pointer to the [SPIDriver](#) object

**Function Class:**

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

Here is the call graph for this function:

**6.17.3.20 void spi\_lld\_select ( SPIDriver \* spip )**

Asserts the slave select signal and prepares for transfers.

**Parameters**

in                    *spip* pointer to the [SPIDriver](#) object

**Function Class:**

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

**6.17.3.21 void spi\_lld\_unselect ( **SPIDriver** \* *spip* )**

Deasserts the slave select signal.

The previously selected peripheral is unselected.

**Parameters**

in *spip* pointer to the **SPIDriver** object

**Function Class:**

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

**6.17.3.22 void spi\_lld\_ignore ( **SPIDriver** \* *spip*, size\_t *n* )**

Ignores data on the SPI bus.

This asynchronous function starts the transmission of a series of idle words on the SPI bus and ignores the received data.

**Postcondition**

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

**Parameters**

in *spip* pointer to the **SPIDriver** object

in *n* number of words to be ignored

**Function Class:**

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

**6.17.3.23 void spi\_lld\_exchange ( **SPIDriver** \* *spip*, size\_t *n*, const void \* *txbuf*, void \* *rxbuf* )**

Exchanges data on the SPI bus.

This asynchronous function starts a simultaneous transmit/receive operation.

**Postcondition**

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

**Note**

The buffers are organized as uint8\_t arrays for data sizes below or equal to 8 bits else it is organized as uint16\_t arrays.

**Parameters**

in *spip* pointer to the **SPIDriver** object

in *n* number of words to be exchanged

in *txbuf* the pointer to the transmit buffer

out *rxbuf* the pointer to the receive buffer

**Function Class:**

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

6.17.3.24 void spi\_lld\_send ( **SPIDriver** \* *spip*, **size\_t** *n*, const void \* *txbuf* )

Sends data over the SPI bus.

This asynchronous function starts a transmit operation.

**Postcondition**

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

**Note**

The buffers are organized as uint8\_t arrays for data sizes below or equal to 8 bits else it is organized as uint16\_t arrays.

**Parameters**

in	<i>spip</i>	pointer to the <b>SPIDriver</b> 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.

6.17.3.25 void spi\_lld\_receive ( **SPIDriver** \* *spip*, **size\_t** *n*, void \* *rxbuf* )

Receives data from the SPI bus.

This asynchronous function starts a receive operation.

**Postcondition**

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

**Note**

The buffers are organized as uint8\_t arrays for data sizes below or equal to 8 bits else it is organized as uint16\_t arrays.

**Parameters**

in	<i>spip</i>	pointer to the <b>SPIDriver</b> 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.

6.17.3.26 uint16\_t spi\_lld\_polled\_exchange ( **SPIDriver** \* *spip*, **uint16\_t** *frame* )

Exchanges one frame using a polled wait.

This synchronous function exchanges one frame using a polled synchronization method. This function is useful when exchanging small amount of data on high speed channels, usually in this situation is much more efficient just wait for completion using polling than suspending the thread waiting for an interrupt.

**Parameters**

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>frame</i>	the data frame to send over the SPI bus

**Returns**

The received data frame from the SPI bus.

## 6.17.4 Variable Documentation

### 6.17.4.1 `SPIDriver SPID1`

SPI1 driver identifier.

### 6.17.4.2 `SPIDriver SPID2`

SPI2 driver identifier.

### 6.17.4.3 `SPIDriver SPID3`

SPI3 driver identifier.

## 6.17.5 Define Documentation

### 6.17.5.1 `#define SPI_USE_WAIT TRUE`

Enables synchronous APIs.

**Note**

Disabling this option saves both code and data space.

### 6.17.5.2 `#define SPI_USE_MUTUAL_EXCLUSION TRUE`

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

**Note**

Disabling this option saves both code and data space.

### 6.17.5.3 `#define spiSelect( spip )`

**Value:**

```
{                                     \
    spi_lld_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.

6.17.5.4 #define spiUnselect( *spip* )**Value:**

```
{
    spi_lld_unselect(spip);
}
```

Deasserts the slave select signal.

The previously selected peripheral is unselected.

**Parameters**

in	<i>spip</i> pointer to the <a href="#">SPIDriver</a> 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.

6.17.5.5 #define spiStartIgnore( *spip*, *n* )**Value:**

```
{
    (spip)->state = SPI_ACTIVE;
    spi_lld_ignore(spip, n);
}
```

Ignores data on the SPI bus.

This asynchronous function starts the transmission of a series of idle words on the SPI bus and ignores the received data.

**Precondition**

A slave must have been selected using [spiSelect\(\)](#) or [spiSelectI\(\)](#).

**Postcondition**

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

**Parameters**

in	<i>spip</i> pointer to the <a href="#">SPIDriver</a> 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.

6.17.5.6 #define spiStartExchange( *spip*, *n*, *txbuf*, *rxbuf* )**Value:**

```
{
    (spip)->state = SPI_ACTIVE;
    spi_lld_exchange(spip, n, txbuf, rdbuf);
}
```

Exchanges data on the SPI bus.

This asynchronous function starts a simultaneous transmit/receive operation.

#### Precondition

A slave must have been selected using `spiSelect()` or `spiSelectI()`.

#### Postcondition

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

#### Note

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

#### Parameters

in	<i>spip</i>	pointer to the <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>rdbuf</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.

#### 6.17.5.7 #define spiStartSend( spip, n, txbuf )

#### Value:

```
{
    (spip)->state = SPI_ACTIVE;
    spi_lld_send(spip, n, txbuf);
}
```

Sends data over the SPI bus.

This asynchronous function starts a transmit operation.

#### Precondition

A slave must have been selected using `spiSelect()` or `spiSelectI()`.

#### Postcondition

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

#### Note

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

#### Parameters

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>n</i>	number of words to send
in	<i>txbuf</i>	the pointer to the transmit buffer

**Function Class:**

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

6.17.5.8 #define spiStartReceive( *spip*, *n*, *rxbuf* )**Value:**

```
{
    (spip)->state = SPI_ACTIVE;
    spi_lld_receive(spip, n, rxbuf);
}
```

Receives data from the SPI bus.

This asynchronous function starts a receive operation.

**Precondition**

A slave must have been selected using [spiSelect\(\)](#) or [spiSelectI\(\)](#).

**Postcondition**

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

**Note**

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

**Parameters**

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>n</i>	number of words to receive
out	<i>rxbuf</i>	the pointer to the receive buffer

**Function Class:**

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

6.17.5.9 #define spiPolledExchange( *spip*, *frame* ) spi\_lld\_polled\_exchange(*spip*, *frame*)

Exchanges one frame using a polled wait.

This synchronous function exchanges one frame using a polled synchronization method. This function is useful when exchanging small amount of data on high speed channels, usually in this situation is much more efficient just wait for completion using polling than suspending the thread waiting for an interrupt.

**Note**

This API is implemented as a macro in order to minimize latency.

**Parameters**

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>frame</i>	the data frame to send over the SPI bus

**Returns**

The received data frame from the SPI bus.

6.17.5.10 #define \_spi\_wait\_s( *spip* )**Value:**

```
{
    chDbgAssert((spip)->thread == NULL, \
                "_spi_wait()", "#1", "already waiting"); \
    (spip)->thread = chThdSelf(); \
    chSchGoSleepS(THD_STATE_SUSPENDED); \
}
```

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 SPI driver using this function.

**Parameters**

in *spip* pointer to the [SPIDriver](#) object

**Function Class:**

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

6.17.5.11 #define \_spi\_wakeup\_isr( *spip* )**Value:**

```
{
    if ((spip)->thread != NULL) { \
        Thread *tp = (spip)->thread; \
        (spip)->thread = NULL; \
        chSysLockFromIsr(); \
        chSchReadyI(tp); \
        chSysUnlockFromIsr(); \
    } \
}
```

Wakes up the waiting thread.

**Parameters**

in *spip* pointer to the [SPIDriver](#) object

**Function Class:**

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

6.17.5.12 #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 *spip* pointer to the [SPIDriver](#) object

#### Function Class:

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

#### 6.17.5.13 #define STM32\_SPI\_USE\_SPI1 TRUE

SPI1 driver enable switch.

If set to TRUE the support for SPI1 is included.

#### Note

The default is TRUE.

#### 6.17.5.14 #define STM32\_SPI\_USE\_SPI2 TRUE

SPI2 driver enable switch.

If set to TRUE the support for SPI2 is included.

#### Note

The default is TRUE.

#### 6.17.5.15 #define STM32\_SPI\_USE\_SPI3 FALSE

SPI3 driver enable switch.

If set to TRUE the support for SPI3 is included.

#### Note

The default is TRUE.

6.17.5.16 #define STM32\_SPI\_SPI1\_IRQ\_PRIORITY 10

SPI1 interrupt priority level setting.

6.17.5.17 #define STM32\_SPI\_SPI2\_IRQ\_PRIORITY 10

SPI2 interrupt priority level setting.

6.17.5.18 #define STM32\_SPI\_SPI3\_IRQ\_PRIORITY 10

SPI3 interrupt priority level setting.

6.17.5.19 #define STM32\_SPI\_SPI1\_DMA\_PRIORITY 1

SPI1 DMA priority (0..3|lowest..highest).

**Note**

The priority level is used for both the TX and RX DMA streams but because of the streams ordering the RX stream has always priority over the TX stream.

6.17.5.20 #define STM32\_SPI\_SPI2\_DMA\_PRIORITY 1

SPI2 DMA priority (0..3|lowest..highest).

**Note**

The priority level is used for both the TX and RX DMA streams but because of the streams ordering the RX stream has always priority over the TX stream.

6.17.5.21 #define STM32\_SPI\_SPI3\_DMA\_PRIORITY 1

SPI3 DMA priority (0..3|lowest..highest).

**Note**

The priority level is used for both the TX and RX DMA streams but because of the streams ordering the RX stream has always priority over the TX stream.

6.17.5.22 #define STM32\_SPI\_DMA\_ERROR\_HOOK( *spip* ) chSysHalt()

SPI DMA error hook.

6.17.5.23 #define STM32\_SPI\_SPI1\_RX\_DMA\_STREAM STM32\_DMA\_STREAM\_ID(2, 0)

DMA stream used for SPI1 RX operations.

**Note**

This option is only available on platforms with enhanced DMA.

```
6.17.5.24 #define STM32_SPI_SPI1_TX_DMA_STREAM STM32_DMA_STREAM_ID(2, 3)
```

DMA stream used for SPI1 TX operations.

**Note**

This option is only available on platforms with enhanced DMA.

```
6.17.5.25 #define STM32_SPI_SPI2_RX_DMA_STREAM STM32_DMA_STREAM_ID(1, 3)
```

DMA stream used for SPI2 RX operations.

**Note**

This option is only available on platforms with enhanced DMA.

```
6.17.5.26 #define STM32_SPI_SPI2_TX_DMA_STREAM STM32_DMA_STREAM_ID(1, 4)
```

DMA stream used for SPI2 TX operations.

**Note**

This option is only available on platforms with enhanced DMA.

```
6.17.5.27 #define STM32_SPI_SPI3_RX_DMA_STREAM STM32_DMA_STREAM_ID(1, 0)
```

DMA stream used for SPI3 RX operations.

**Note**

This option is only available on platforms with enhanced DMA.

```
6.17.5.28 #define STM32_SPI_SPI3_TX_DMA_STREAM STM32_DMA_STREAM_ID(1, 7)
```

DMA stream used for SPI3 TX operations.

**Note**

This option is only available on platforms with enhanced DMA.

## 6.17.6 Typedef Documentation

```
6.17.6.1 typedef struct SPIDriver SPIDriver
```

Type of a structure representing an SPI driver.

```
6.17.6.2 typedef void(* spicallback_t)(SPIDriver *spip)
```

SPI notification callback type.

**Parameters**

in                   `spip` pointer to the `SPIDriver` object triggering the callback

### 6.17.7 Enumeration Type Documentation

#### 6.17.7.1 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.

## 6.18 Time Measurement Driver.

### 6.18.1 Detailed Description

Time Measurement unit. This module implements a time measurement mechanism able to monitor a portion of code and store the best/worst/last measurement. The measurement is performed using the realtime counter mechanism abstracted in the HAL driver.

### Data Structures

- struct [TimeMeasurement](#)  
*Time Measurement structure.*

### Functions

- void [tmInit](#) (void)  
*Initializes the Time Measurement unit.*
- void [tmObjectInit](#) ([TimeMeasurement](#) \*tmp)  
*Initializes a [TimeMeasurement](#) object.*

### Defines

- #define [tmStartMeasurement](#)(tmp) (tmp)->start(tmp)  
*Starts a measurement.*
- #define [tmStopMeasurement](#)(tmp) (tmp)->stop(tmp)  
*Stops a measurement.*

### Typedefs

- typedef struct [TimeMeasurement](#) [TimeMeasurement](#)  
*Type of a Time Measurement object.*

## 6.18.2 Function Documentation

### 6.18.2.1 void tmInit( void )

Initializes the Time Measurement unit.

#### 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:



### 6.18.2.2 void tmObjectInit( TimeMeasurement \* tmp )

Initializes a [TimeMeasurement](#) object.

#### Parameters

out                    *tmp*    pointer to a [TimeMeasurement](#) structure

#### Function Class:

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

## 6.18.3 Define Documentation

### 6.18.3.1 #define tmStartMeasurement( tmp ) (tmp)->start(tmp)

Starts a measurement.

#### Precondition

The [TimeMeasurement](#) must be initialized.

#### Note

This function can be invoked in any context.

#### Parameters

in,out                *tmp*    pointer to a [TimeMeasurement](#) structure

#### Function Class:

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

6.18.3.2 #define tmStopMeasurement( *tmp* ) (*tmp*)->stop(*tmp*)

Stops a measurement.

#### Precondition

The [TimeMeasurement](#) must be initialized.

#### Note

This function can be invoked in any context.

#### Parameters

in, out                   *tmp*   pointer to a [TimeMeasurement](#) structure

#### Function Class:

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

## 6.18.4 Typedef Documentation

### 6.18.4.1 typedef struct TimeMeasurement TimeMeasurement

Type of a Time Measurement object.

#### Note

Start/stop of measurements is performed through the function pointers in order to avoid inlining of those functions which could compromise measurement accuracy.

The maximum measurable time period depends on the implementation of the realtime counter in the HAL driver.

The measurement is not 100% cycle-accurate, it can be in excess of few cycles depending on the compiler and target architecture.

Interrupts can affect measurement if the measurement is performed with interrupts enabled.

## 6.19 UART Driver

### 6.19.1 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.

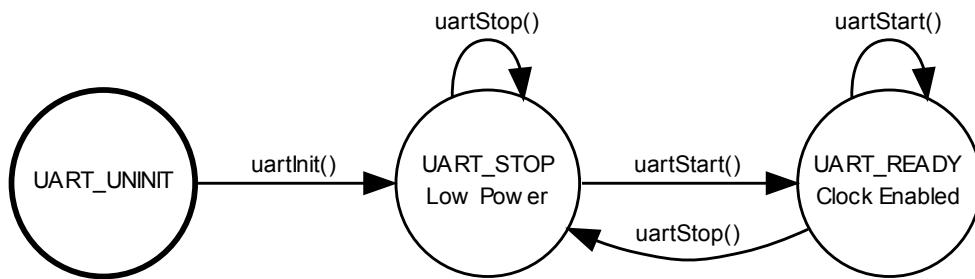
If your application requires a synchronous buffered driver then the [Serial Driver](#) should be used instead.

### Precondition

In order to use the UART driver the `HAL_USE_UART` option must be enabled in `halconf.h`.

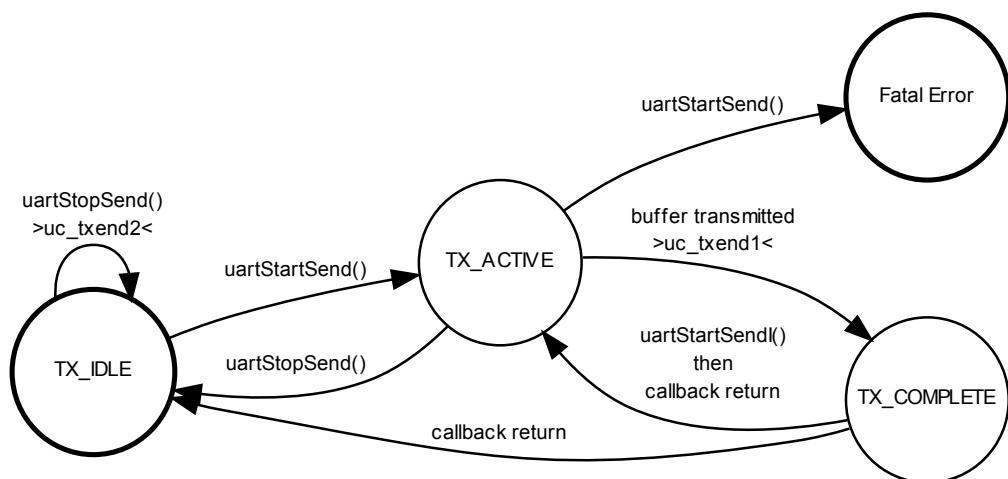
#### 6.19.2 Driver State Machine

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).



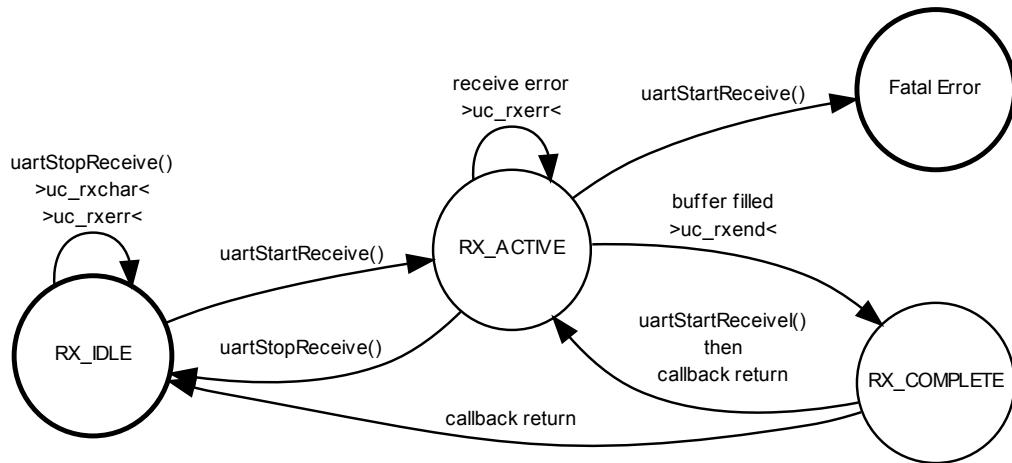
##### 6.19.2.1 Transmitter sub State Machine

The follow diagram describes the transmitter state machine, this diagram is valid while the driver is in the `UART_READY` state. This state machine is automatically reset to the `TX_IDLE` state each time the driver enters the `UART_READY` state.



### 6.19.2.2 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.



## Data Structures

- struct `UARTConfig`  
*Driver configuration structure.*
- struct `UARTDriver`  
*Structure representing an UART driver.*

## Functions

- void `uartInit` (void)  
*UART Driver initialization.*
- void `uartObjectInit` (`UARTDriver` \*uartp)  
*Initializes the standard part of a `UARTDriver` structure.*
- void `uartStart` (`UARTDriver` \*uartp, const `UARTConfig` \*config)  
*Configures and activates the UART peripheral.*
- void `uartStop` (`UARTDriver` \*uartp)  
*Deactivates the UART peripheral.*
- void `uartStartSend` (`UARTDriver` \*uartp, size\_t n, const void \*txbuf)  
*Starts a transmission on the UART peripheral.*
- void `uartStartSendl` (`UARTDriver` \*uartp, size\_t n, const void \*txbuf)  
*Starts a transmission on the UART peripheral.*
- size\_t `uartStopSend` (`UARTDriver` \*uartp)  
*Stops any ongoing transmission.*
- size\_t `uartStopSendl` (`UARTDriver` \*uartp)  
*Stops any ongoing transmission.*
- void `uartStartReceive` (`UARTDriver` \*uartp, size\_t n, void \*rxbuf)

- Starts a receive operation on the UART peripheral.
- void `uartStartReceive (UARTDriver *uartp, size_t n, void *rdbuf)`
  - Starts a receive operation on the UART peripheral.
- size\_t `uartStopReceive (UARTDriver *uartp)`
  - Stops any ongoing receive operation.
- size\_t `uartStopReceive1 (UARTDriver *uartp)`
  - Stops any ongoing receive operation.
- `CH_IRQ_HANDLER (USART1_IRQHandler)`
  - USART1 IRQ handler.
- `CH_IRQ_HANDLER (USART2_IRQHandler)`
  - USART2 IRQ handler.
- `CH_IRQ_HANDLER (USART3_IRQHandler)`
  - USART3 IRQ handler.
- void `uart_lld_init (void)`
  - Low level UART driver initialization.
- void `uart_lld_start (UARTDriver *uartp)`
  - Configures and activates the UART peripheral.
- void `uart_lld_stop (UARTDriver *uartp)`
  - Deactivates the UART peripheral.
- void `uart_lld_start_send (UARTDriver *uartp, size_t n, const void *txbuf)`
  - Starts a transmission on the UART peripheral.
- size\_t `uart_lld_stop_send (UARTDriver *uartp)`
  - Stops any ongoing transmission.
- void `uart_lld_start_receive (UARTDriver *uartp, size_t n, void *rdbuf)`
  - Starts a receive operation on the UART peripheral.
- size\_t `uart_lld_stop_receive (UARTDriver *uartp)`
  - Stops any ongoing receive operation.

## Variables

- `UARTDriver UARTD1`
  - USART1 UART driver identifier.
- `UARTDriver UARTD2`
  - USART2 UART driver identifier.
- `UARTDriver UARTD3`
  - USART3 UART driver identifier.

## 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.

## Configuration options

- `#define STM32_UART_USE_USART1 TRUE`  
*UART driver on USART1 enable switch.*
- `#define STM32_UART_USE_USART2 TRUE`  
*UART driver on USART2 enable switch.*
- `#define STM32_UART_USE_USART3 TRUE`  
*UART driver on USART3 enable switch.*
- `#define STM32_UART_USART1_IRQ_PRIORITY 12`  
*USART1 interrupt priority level setting.*
- `#define STM32_UART_USART2_IRQ_PRIORITY 12`  
*USART2 interrupt priority level setting.*
- `#define STM32_UART_USART3_IRQ_PRIORITY 12`  
*USART3 interrupt priority level setting.*
- `#define STM32_UART_USART1_DMA_PRIORITY 0`  
*USART1 DMA priority (0..3|lowest..highest).*
- `#define STM32_UART_USART2_DMA_PRIORITY 0`  
*USART2 DMA priority (0..3|lowest..highest).*
- `#define STM32_UART_USART3_DMA_PRIORITY 0`  
*USART3 DMA priority (0..3|lowest..highest).*
- `#define STM32_UART_DMA_ERROR_HOOK(uartp) chSysHalt()`  
*USART1 DMA error hook.*
- `#define STM32_UART_USART1_RX_DMA_STREAM STM32_DMA_STREAM_ID(2, 5)`  
*DMA stream used for USART1 RX operations.*
- `#define STM32_UART_USART1_TX_DMA_STREAM STM32_DMA_STREAM_ID(2, 7)`  
*DMA stream used for USART1 TX operations.*
- `#define STM32_UART_USART2_RX_DMA_STREAM STM32_DMA_STREAM_ID(1, 5)`  
*DMA stream used for USART2 RX operations.*
- `#define STM32_UART_USART2_TX_DMA_STREAM STM32_DMA_STREAM_ID(1, 6)`  
*DMA stream used for USART2 TX operations.*
- `#define STM32_UART_USART3_RX_DMA_STREAM STM32_DMA_STREAM_ID(1, 1)`  
*DMA stream used for USART3 RX operations.*
- `#define STM32_UART_USART3_TX_DMA_STREAM STM32_DMA_STREAM_ID(1, 3)`  
*DMA stream used for USART3 TX operations.*

## Typedefs

- `typedef uint32_t uartflags_t`  
*UART driver condition flags type.*
- `typedef struct UARDriver UARDriver`  
*Structure representing an UART driver.*
- `typedef void(* uartcb_t )(UARDriver *uartp)`  
*Generic UART notification callback type.*
- `typedef void(* uartccb_t )(UARDriver *uartp, uint16_t c)`  
*Character received UART notification callback type.*
- `typedef void(* uarteccb_t )(UARDriver *uartp, uartflags_t e)`  
*Receive error UART notification callback type.*

## Enumerations

- enum `uartstate_t` { `UART_UNINIT` = 0, `UART_STOP` = 1, `UART_READY` = 2 }

*Driver state machine possible states.*

- enum `uartxstate_t` { `UART_TX_IDLE` = 0, `UART_TX_ACTIVE` = 1, `UART_TX_COMPLETE` = 2 }

*Transmitter state machine states.*

- enum `uartrxstate_t` { `UART_RX_IDLE` = 0, `UART_RX_ACTIVE` = 1, `UART_RX_COMPLETE` = 2 }

*Receiver state machine states.*

## 6.19.3 Function Documentation

### 6.19.3.1 void uartInit ( void )

UART Driver initialization.

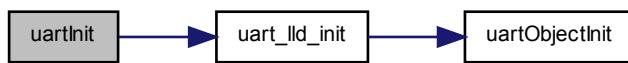
#### Note

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:



### 6.19.3.2 void uartObjectInit ( `UARTDriver` \* `uartp` )

Initializes the standard part of a `UARTDriver` structure.

#### Parameters

<code>out</code>	<code>uartp</code> pointer to the <code>UARTDriver</code> object
------------------	--

#### Function Class:

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

### 6.19.3.3 void uartStart ( `UARTDriver` \* `uartp`, const `UARTConfig` \* `config` )

Configures and activates the UART peripheral.

#### Parameters

<code>in</code>	<code>uartp</code> pointer to the <code>UARTDriver</code> object
<code>in</code>	<code>config</code> pointer to the <code>UARTConfig</code> object

**Function Class:**

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

Here is the call graph for this function:

**6.19.3.4 void uartStop ( **UARTDriver** \* *uartp* )**

Deactivates the UART peripheral.

**Parameters**

in	<i>uartp</i> pointer to the <b>UARTDriver</b> 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:

**6.19.3.5 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 <b>UARTDriver</b> 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:



### 6.19.3.6 void uartStartSendl ( **UARTDriver** \* *uartp*, **size\_t** *n*, const void \* *txbuf* )

Starts a transmission on the UART peripheral.

**Note**

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

This function has to be invoked from a lock zone.

**Parameters**

in	<i>uartp</i>	pointer to the <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:



### 6.19.3.7 **size\_t** uartStopSend ( **UARTDriver** \* *uartp* )

Stops any ongoing transmission.

**Note**

Stopping a transmission also suppresses the transmission callbacks.

**Parameters**

in *uartp* pointer to the [UARTDriver](#) object

**Returns**

The number of data frames not transmitted by the stopped transmit operation.

**Return values**

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:

**6.19.3.8 size\_t uartStopSend( [UARTDriver](#) \* *uartp* )**

Stops any ongoing transmission.

**Note**

Stopping a transmission also suppresses the transmission callbacks.  
This function has to be invoked from a lock zone.

**Parameters**

in *uartp* pointer to the [UARTDriver](#) object

**Returns**

The number of data frames not transmitted by the stopped transmit operation.

**Return values**

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:



#### 6.19.3.9 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 <b>UARTDriver</b> object
in	<i>n</i>	number of data frames to send
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:



#### 6.19.3.10 void uartStartReceive1 ( **UARTDriver** \* *uartp*, **size\_t** *n*, **void** \* *rxbuf* )

Starts a receive operation on the UART peripheral.

#### Note

The buffers are organized as **uint8\_t** arrays for data sizes below or equal to 8 bits else it is organized as **uint16\_t** arrays.

This function has to be invoked from a lock zone.

#### Parameters

in	<i>uartp</i>	pointer to the <b>UARTDriver</b> object
in	<i>n</i>	number of data frames to send
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:

**6.19.3.11 size\_t uartStopReceive ( **UARTDriver** \* *uartp* )**

Stops any ongoing receive operation.

**Note**

Stopping a receive operation also suppresses the receive callbacks.

**Parameters**

in *uartp* pointer to the **UARTDriver** object

**Returns**

The number of data frames not received by the stopped receive operation.

**Return values**

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:

**6.19.3.12 size\_t uartStopReceive1 ( **UARTDriver** \* *uartp* )**

Stops any ongoing receive operation.

**Note**

Stopping a receive operation also suppresses the receive callbacks.  
This function has to be invoked from a lock zone.

**Parameters**

in *uartp* pointer to the [UARTDriver](#) object

**Returns**

The number of data frames not received by the stopped receive operation.

**Return values**

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:



### 6.19.3.13 CH\_IRQ\_HANDLER ( USART1\_IRQHandler )

USART1 IRQ handler.

**Function Class:**

Interrupt handler, this function should not be directly invoked.

### 6.19.3.14 CH\_IRQ\_HANDLER ( USART2\_IRQHandler )

USART2 IRQ handler.

**Function Class:**

Interrupt handler, this function should not be directly invoked.

### 6.19.3.15 CH\_IRQ\_HANDLER ( USART3\_IRQHandler )

USART3 IRQ handler.

**Function Class:**

Interrupt handler, this function should not be directly invoked.

**6.19.3.16 void uart\_lld\_init( void )**

Low level UART driver initialization.

**Function Class:**

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

Here is the call graph for this function:

**6.19.3.17 void uart\_lld\_start( UARTDriver \* uartp )**

Configures and activates the UART peripheral.

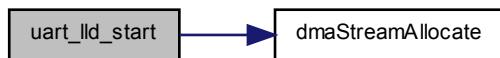
**Parameters**

in                    *uartp*    pointer to the [UARTDriver](#) object

**Function Class:**

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

Here is the call graph for this function:

**6.19.3.18 void uart\_lld\_stop( UARTDriver \* uartp )**

Deactivates the UART peripheral.

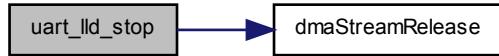
**Parameters**

in                    *uartp*    pointer to the [UARTDriver](#) object

**Function Class:**

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

Here is the call graph for this function:



### 6.19.3.19 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 <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:

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

### 6.19.3.20 **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 <code>UARTDriver</code> 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.

### 6.19.3.21 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 <code>UARTDriver</code> 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.

### 6.19.3.22 `size_t uart_ll_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 <code>UARTDriver</code> 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.

## 6.19.4 Variable Documentation

### 6.19.4.1 `UARTDriver UARTD1`

USART1 UART driver identifier.

### 6.19.4.2 `UARTDriver UARTD2`

USART2 UART driver identifier.

### 6.19.4.3 `UARTDriver UARTD3`

USART3 UART driver identifier.

## 6.19.5 Define Documentation

### 6.19.5.1 `#define UART_NO_ERROR 0`

No pending conditions.

### 6.19.5.2 `#define UART_PARITY_ERROR 4`

Parity error happened.

6.19.5.3 #define UART\_FRAMING\_ERROR 8

Framing error happened.

6.19.5.4 #define UART\_OVERRUN\_ERROR 16

Overflow happened.

6.19.5.5 #define UART\_NOISE\_ERROR 32

Noise on the line.

6.19.5.6 #define UART\_BREAK\_DETECTED 64

Break detected.

6.19.5.7 #define STM32\_UART\_USE\_USART1 TRUE

UART driver on USART1 enable switch.

If set to TRUE the support for USART1 is included.

#### Note

The default is FALSE.

6.19.5.8 #define STM32\_UART\_USE\_USART2 TRUE

UART driver on USART2 enable switch.

If set to TRUE the support for USART2 is included.

#### Note

The default is FALSE.

6.19.5.9 #define STM32\_UART\_USE\_USART3 TRUE

UART driver on USART3 enable switch.

If set to TRUE the support for USART3 is included.

#### Note

The default is FALSE.

6.19.5.10 #define STM32\_UART\_USART1\_IRQ\_PRIORITY 12

USART1 interrupt priority level setting.

6.19.5.11 #define STM32\_UART\_USART2\_IRQ\_PRIORITY 12

USART2 interrupt priority level setting.

```
6.19.5.12 #define STM32_UART_USART3_IRQ_PRIORITY 12
```

USART3 interrupt priority level setting.

```
6.19.5.13 #define STM32_UART_USART1_DMA_PRIORITY 0
```

USART1 DMA priority (0..3|lowest..highest).

**Note**

The priority level is used for both the TX and RX DMA channels but because of the channels ordering the RX channel has always priority over the TX channel.

```
6.19.5.14 #define STM32_UART_USART2_DMA_PRIORITY 0
```

USART2 DMA priority (0..3|lowest..highest).

**Note**

The priority level is used for both the TX and RX DMA channels but because of the channels ordering the RX channel has always priority over the TX channel.

```
6.19.5.15 #define STM32_UART_USART3_DMA_PRIORITY 0
```

USART3 DMA priority (0..3|lowest..highest).

**Note**

The priority level is used for both the TX and RX DMA channels but because of the channels ordering the RX channel has always priority over the TX channel.

```
6.19.5.16 #define STM32_UART_DMA_ERROR_HOOK( uartp ) chSysHalt()
```

USART1 DMA error hook.

**Note**

The default action for DMA errors is a system halt because DMA error can only happen because programming errors.

```
6.19.5.17 #define STM32_UART_USART1_RX_DMA_STREAM STM32_DMA_STREAM_ID(2, 5)
```

DMA stream used for USART1 RX operations.

**Note**

This option is only available on platforms with enhanced DMA.

```
6.19.5.18 #define STM32_UART_USART1_TX_DMA_STREAM STM32_DMA_STREAM_ID(2, 7)
```

DMA stream used for USART1 TX operations.

**Note**

This option is only available on platforms with enhanced DMA.

```
6.19.5.19 #define STM32_UART_USART2_RX_DMA_STREAM STM32_DMA_STREAM_ID(1, 5)
```

DMA stream used for USART2 RX operations.

**Note**

This option is only available on platforms with enhanced DMA.

```
6.19.5.20 #define STM32_UART_USART2_TX_DMA_STREAM STM32_DMA_STREAM_ID(1, 6)
```

DMA stream used for USART2 TX operations.

**Note**

This option is only available on platforms with enhanced DMA.

```
6.19.5.21 #define STM32_UART_USART3_RX_DMA_STREAM STM32_DMA_STREAM_ID(1, 1)
```

DMA stream used for USART3 RX operations.

**Note**

This option is only available on platforms with enhanced DMA.

```
6.19.5.22 #define STM32_UART_USART3_TX_DMA_STREAM STM32_DMA_STREAM_ID(1, 3)
```

DMA stream used for USART3 TX operations.

**Note**

This option is only available on platforms with enhanced DMA.

## 6.19.6 Typedef Documentation

```
6.19.6.1 typedef uint32_t uartflags_t
```

UART driver condition flags type.

```
6.19.6.2 typedef struct UARTDriver UARTDriver
```

Structure representing an UART driver.

```
6.19.6.3 typedef void(* uartcb_t)(UARTDriver *uartp)
```

Generic UART notification callback type.

**Parameters**

in                    *uartp* pointer to the [UARTDriver](#) object

6.19.6.4 `typedef void(* uartccb_t)(UARTDriver *uartp, uint16_t c)`

Character received UART notification callback type.

#### Parameters

in	<i>uartp</i>	pointer to the <code>UARTDriver</code> object
in	<i>c</i>	received character

6.19.6.5 `typedef void(* uarteccb_t)(UARTDriver *uartp, uartflags_t e)`

Receive error UART notification callback type.

#### Parameters

in	<i>uartp</i>	pointer to the <code>UARTDriver</code> object
in	<i>e</i>	receive error mask

## 6.19.7 Enumeration Type Documentation

6.19.7.1 `enum uartstate_t`

Driver state machine possible states.

#### Enumerator:

`UART_UNINIT` Not initialized.

`UART_STOP` Stopped.

`UART_READY` Ready.

6.19.7.2 `enum uarttxstate_t`

Transmitter state machine states.

#### Enumerator:

`UART_TX_IDLE` Not transmitting.

`UART_TX_ACTIVE` Transmitting.

`UART_TX_COMPLETE` Buffer complete.

6.19.7.3 `enum uartrxstate_t`

Receiver state machine states.

#### Enumerator:

`UART_RX_IDLE` Not receiving.

`UART_RX_ACTIVE` Receiving.

`UART_RX_COMPLETE` Buffer complete.

## 6.20 USB Driver

### 6.20.1 Detailed Description

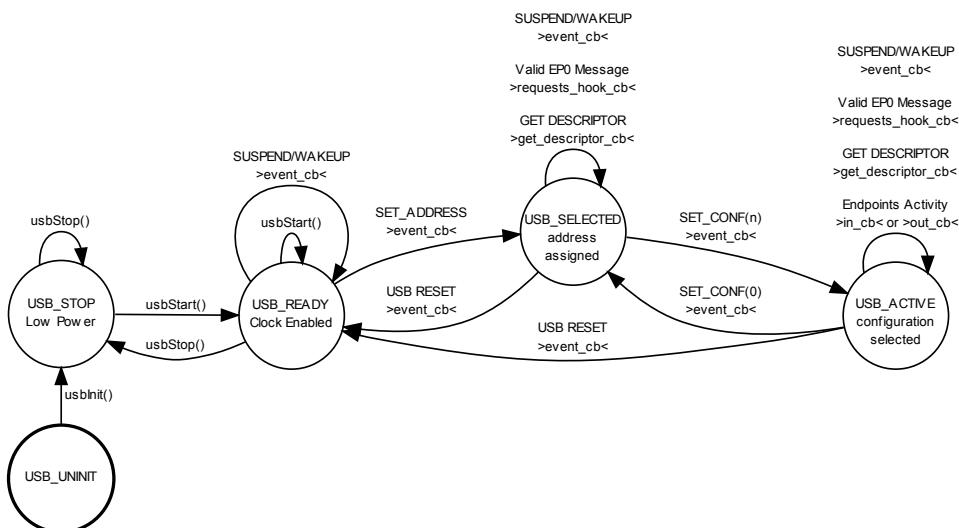
Generic USB Driver. This module implements a generic USB (Universal Serial Bus) driver supporting device-mode operations.

#### Precondition

In order to use the USB driver the `HAL_USE_USB` option must be enabled in `halconf.h`.

### 6.20.2 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).



### 6.20.3 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.

#### 6.20.3.1 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.

### 6.20.3.2 USB Endpoints

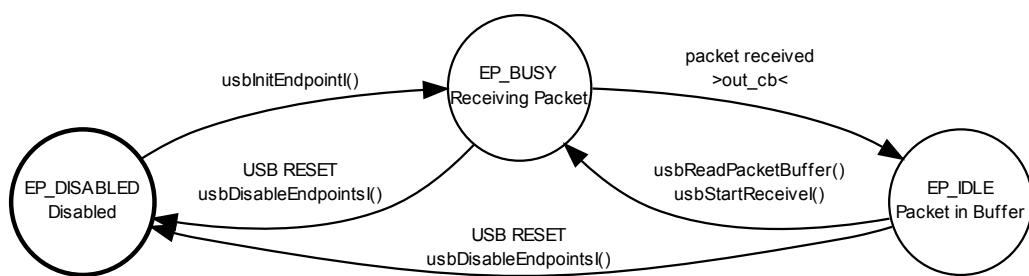
USB endpoints are the objects that the application uses to exchange data with the host. There are two kind of endpoints:

- **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.

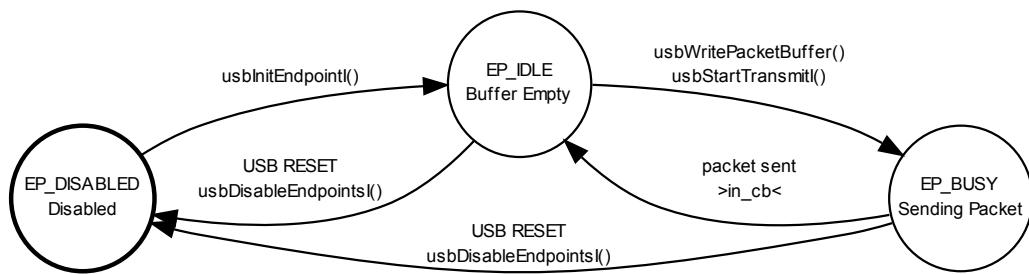
In ChibiOS/RT the endpoints can be configured in two distinct ways:

- **Packet Mode.** In this mode the driver invokes a callback each time a packet has been received or transmitted. This mode is especially suited for those applications handling continuous streams of data.

States diagram for OUT endpoints in packet mode:

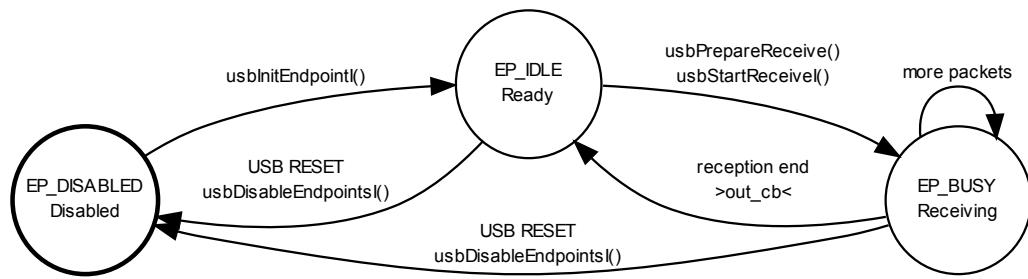


States diagram for IN endpoints in packet mode:

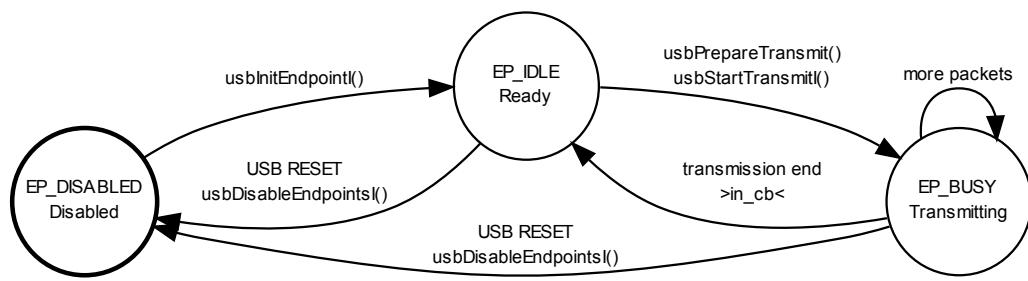


- **Transaction Mode.** In this mode the driver invokes a callback only after a large, potentially multi-packet, transfer has been completed, a callback is invoked only at the end of the transfer.

States diagram for OUT endpoints in transaction mode:



States diagram for IN endpoints in transaction mode:



### 6.20.3.3 USB Packet Buffers

An important difference between packet and transaction modes is that there is a dedicated endpoint buffer in packet mode while in transaction mode the application has to specify its own buffer for duration of the whole transfer.

Packet buffers cannot be accessed directly by the application because those could not be necessarily memory mapped, a buffer could be a FIFO or some other kind of memory accessible in a special way depending on the underlying hardware architecture, the functions `usbReadPacketI()` and `usbWritePacketI()` allow to access packet buffers in an abstract way.

### 6.20.3.4 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.

## Data Structures

- struct **USBDescriptor**  
*Type of an USB descriptor.*
- struct **stm32\_usb\_t**  
*USB registers block.*
- struct **stm32\_usb\_descriptor\_t**  
*USB descriptor registers block.*
- struct **USBInEndpointState**  
*Type of an endpoint state structure.*
- struct **USBOutEndpointState**  
*Type of an 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

- void **usbInit** (void)  
*USB Driver initialization.*
- void **usbObjectInit** (USBDriver \*usbp)  
*Initializes the standard part of a `USBDriver` structure.*
- void **usbStart** (USBDriver \*usbp, const USBConfig \*config)  
*Configures and activates the USB peripheral.*
- void **usbStop** (USBDriver \*usbp)  
*Deactivates the USB peripheral.*
- void **usbInitEndpoint** (USBDriver \*usbp, usbep\_t ep, const USBEndpointConfig \*epcp)  
*Enables an endpoint.*
- void **usbDisableEndpoints** (USBDriver \*usbp)  
*Disables all the active endpoints.*
- bool\_t **usbStartReceive** (USBDriver \*usbp, usbep\_t ep)  
*Starts a receive transaction on an OUT endpoint.*
- bool\_t **usbStartTransmit** (USBDriver \*usbp, usbep\_t ep)  
*Starts a transmit transaction on an IN endpoint.*
- bool\_t **usbStallReceive** (USBDriver \*usbp, usbep\_t ep)  
*Stalls an OUT endpoint.*
- bool\_t **usbStallTransmit** (USBDriver \*usbp, usbep\_t ep)  
*Stalls an IN endpoint.*
- void **\_usb\_reset** (USBDriver \*usbp)  
*USB reset 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.*
- **CH\_IRQ\_HANDLER** (Vector8C)  
*USB high priority interrupt handler.*

- **CH\_IRQ\_HANDLER** (Vector90)  
*USB low priority interrupt handler.*
- 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.*
- size\_t **usb\_lld\_read\_packet\_buffer** (USBDriver \*usbp, usbep\_t ep, uint8\_t \*buf, size\_t n)  
*Reads from a dedicated packet buffer.*
- void **usb\_lld\_write\_packet\_buffer** (USBDriver \*usbp, usbep\_t ep, const uint8\_t \*buf, size\_t n)  
*Writes to a dedicated packet buffer.*
- void **usb\_lld\_prepare\_receive** (USBDriver \*usbp, usbep\_t ep, uint8\_t \*buf, size\_t n)  
*Prepares for a receive operation.*
- void **usb\_lld\_prepare\_transmit** (USBDriver \*usbp, usbep\_t ep, const uint8\_t \*buf, size\_t n)  
*Prepares for a transmit operation.*
- void **usb\_lld\_start\_out** (USBDriver \*usbp, usbep\_t ep)  
*Starts a receive operation on an OUT endpoint.*
- void **usb\_lld\_start\_in** (USBDriver \*usbp, usbep\_t ep)  
*Starts a transmit operation on an IN endpoint.*
- void **usb\_lld\_stall\_out** (USBDriver \*usbp, usbep\_t ep)  
*Brings an OUT endpoint in the stalled state.*
- void **usb\_lld\_stall\_in** (USBDriver \*usbp, usbep\_t ep)  
*Brings an IN endpoint in the stalled state.*
- void **usb\_lld\_clear\_out** (USBDriver \*usbp, usbep\_t ep)  
*Brings an OUT endpoint in the active state.*
- void **usb\_lld\_clear\_in** (USBDriver \*usbp, usbep\_t ep)  
*Brings an IN endpoint in the active state.*

## Variables

- **USBDriver USBD1**  
*USB1 driver identifier.*

## 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(bcdUSB, bDeviceClass, bDeviceSubClass, bDeviceProtocol, bMaxPacketSize, idVendor, idProduct, bcdDevice, iManufacturer, iProduct, iSerialNumber, bNumConfigurations)`  
*Device Descriptor helper macro.*
- `#define USB_DESC_CONFIGURATION(wTotalLength, bNumInterfaces, bConfigurationValue, iConfiguration, bmAttributes, bMaxPower)`  
*Configuration Descriptor helper macro.*
- `#define USB_DESC_INTERFACE(blInterfaceNumber, bAlternateSetting, bNumEndpoints, blInterfaceClass, blInterfaceSubClass, blInterfaceProtocol, ilInterface)`  
*Interface Descriptor helper macro.*
- `#define USB_DESC_ENDPOINT(bEndpointAddress, bmAttributes, wMaxPacketSize, blInterval)`  
*Endpoint Descriptor helper macro.*

## Endpoint types and settings

- `#define USB_EP_MODE_TYPE 0x0003`
- `#define USB_EP_MODE_TYPE_CTRL 0x0000`
- `#define USB_EP_MODE_TYPE_ISOC 0x0001`
- `#define USB_EP_MODE_TYPE_BULK 0x0002`
- `#define USB_EP_MODE_TYPE_INTR 0x0003`
- `#define USB_EP_MODE_TRANSACTION 0x0000`
- `#define USB_EP_MODE_PACKET 0x0010`

## Macro Functions

- `#define usbConnectBus(usbp) usb_lld_connect_bus(usbp)`  
*Connects the USB device.*
- `#define usbDisconnectBus(usbp) usb_lld_disconnect_bus(usbp)`  
*Disconnect the USB device.*
- `#define usbGetFrameNumber(usbp) usb_lld_get_frame_number(usbp)`  
*Returns the current frame number.*
- `#define usbGetTransmitStatusl(usbp, ep) ((usbp)->transmitting & (1 << (ep)))`  
*Returns the status of an IN endpoint.*
- `#define usbGetReceiveStatusl(usbp, ep) ((usbp)->receiving & (1 << (ep)))`  
*Returns the status of an OUT endpoint.*
- `#define usbReadPacketBuffer(usbp, ep, buf, n) usb_lld_read_packet_buffer(usbp, ep, buf, n)`  
*Reads from a dedicated packet buffer.*
- `#define usbWritePacketBuffer(usbp, ep, buf, n) usb_lld_write_packet_buffer(usbp, ep, buf, n)`  
*Writes to a dedicated packet buffer.*
- `#define usbPrepareReceive(usbp, ep, buf, n) usb_lld_prepare_receive(usbp, ep, buf, n)`  
*Prepares for a receive transaction on an OUT endpoint.*
- `#define usbPrepareTransmit(usbp, ep, buf, n) usb_lld_prepare_transmit(usbp, ep, buf, n)`  
*Prepares for a transmit transaction on an IN endpoint.*

- `#define usbGetReceiveTransactionSize(usb, ep) usb_ll_get_transaction_size(usb, ep)`  
*Returns the exact size of a receive transaction.*
- `#define usbGetReceivePacketSize(usb, ep) usb_ll_get_packet_size(usb, ep)`  
*Returns the exact size of a received packet.*
- `#define usbSetupTransfer(usb, buf, n, endcb)`  
*Request transfer setup.*
- `#define usbReadSetup(usb, ep, buf) usb_ll_read_setup(usb, ep, buf)`  
*Reads a setup packet from the dedicated packet buffer.*

## Low Level driver helper macros

- `#define _usb_isr_invoke_event_cb(usb, evt)`  
*Common ISR code, usb event callback.*
- `#define _usb_isr_invoke_sof_cb(usb)`  
*Common ISR code, SOF callback.*
- `#define _usb_isr_invoke_setup_cb(usb, ep)`  
*Common ISR code, setup packet callback.*
- `#define _usb_isr_invoke_in_cb(usb, ep)`  
*Common ISR code, IN endpoint callback.*
- `#define _usb_isr_invoke_out_cb(usb, ep)`  
*Common ISR code, OUT endpoint event.*

## Register aliases

- `#define RXADDR1 TXADDR0`
- `#define TXADDR1 RXADDR0`

## Defines

- `#define USB_ENDOPOINTS_NUMBER 7`  
*Number of the available endpoints.*
- `#define STM32_USB_BASE (APB1PERIPH_BASE + 0x5C00)`  
*USB registers block numeric address.*
- `#define STM32_USBRAM_BASE (APB1PERIPH_BASE + 0x6000)`  
*USB RAM numeric address.*
- `#define STM32_USB ((stm32_usb_t *)STM32_USB_BASE)`  
*Pointer to the USB registers block.*
- `#define STM32_USBRAM ((uint32_t *)STM32_USBRAM_BASE)`  
*Pointer to the USB RAM.*
- `#define USB_PMA_SIZE 512`  
*Size of the dedicated packet memory.*
- `#define EPR_TOGGLE_MASK`  
*Mask of all the toggling bits in the EPR register.*
- `#define USB_GET_DESCRIPTOR(ep)`  
*Returns an endpoint descriptor pointer.*
- `#define USB_ADDR2PTR(addr) ((uint32_t *)((addr) * 2 + STM32_USBRAM_BASE))`  
*Converts from a PMA address to a physical address.*
- `#define USB_MAX_ENDPOINTS USB_ENDOPOINTS_NUMBER`  
*Maximum endpoint address.*
- `#define USB_SET_ADDRESS_MODE USB_LATE_SET_ADDRESS`

- This device requires the address change after the status packet.*
- `#define STM32_USB_USE_USB1 TRUE`  
*USB1 driver enable switch.*
  - `#define STM32_USB_LOW_POWER_ON_SUSPEND FALSE`  
*Enables the USB device low power mode on suspend.*
  - `#define STM32_USB_USB1_HP_IRQ_PRIORITY 6`  
*USB1 interrupt priority level setting.*
  - `#define STM32_USB_USB1_LP_IRQ_PRIORITY 14`  
*USB1 interrupt priority level setting.*
  - `#define usb_lld_fetch_word(p) (*(uint16_t *)(p))`  
*Fetches a 16 bits word value from an USB message.*
  - `#define usb_lld_get_frame_number(usbp) (STM32_USB->FNR & FNR_FN_MASK)`  
*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_get_packet_size(usbp, ep) ((size_t)USB_GET_DESCRIPTOR(ep)->RXCOUNT & RXCOUNT_-COUNT_MASK)`  
*Returns the exact size of a received packet.*

## 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_t(* usbreqhandler_t )(USBDriver *usbp)`  
*Type of a requests handler callback.*
- `typedef const USBDescriptor *(* usbgetdescriptor_t )(USBDriver *usbp, uint8_t dtype, uint8_t dindex, uint16_-t lang)`  
*Type of an USB descriptor-retrieving callback.*

## Enumerations

- `enum usbstate_t {`  
`USB_UNINIT = 0, USB_STOP = 1, USB_READY = 2, USB_SELECTED = 3,`  
`USB_ACTIVE = 4 }`  
*Type of a driver state machine possible states.*
- `enum usbepstatus_t { EP_STATUS_DISABLED = 0, EP_STATUS_STALLED = 1, EP_STATUS_ACTIVE = 2 }`  
*Type of an endpoint status.*
- `enum usbep0state_t {`  
`USB_EP0_WAITING_SETUP, USB_EP0_TX, USB_EP0_WAITING_STS, USB_EP0_RX,`  
`USB_EP0_SENDING_STS, USB_EP0_ERROR }`  
*Type of an endpoint zero state machine states.*

- enum `usbevent_t` {
 `USB_EVENT_RESET` = 0, `USB_EVENT_ADDRESS` = 1, `USB_EVENT_CONFIGURED` = 2, `USB_EVENT_SUSPEND` = 3,
 `USB_EVENT_WAKEUP` = 4, `USB_EVENT_STALLED` = 5
 }

*Type of an enumeration of the possible USB events.*

#### 6.20.4 Function Documentation

##### 6.20.4.1 void usblInit ( void )

USB Driver initialization.

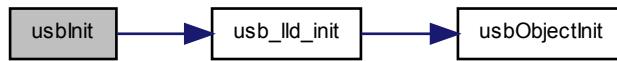
###### Note

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

###### Function Class:

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

Here is the call graph for this function:



##### 6.20.4.2 void usbObjectInit ( USBDriver \* usbp )

Initializes the standard part of a `USBDriver` structure.

###### Parameters

out	<code>usbp</code> pointer to the <code>USBDriver</code> object
-----	--

###### Function Class:

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

##### 6.20.4.3 void usbStart ( USBDriver \* usbp, const USBConfig \* config )

Configures and activates the USB peripheral.

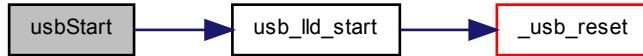
###### Parameters

in	<code>usbp</code> pointer to the <code>USBDriver</code> object
in	<code>config</code> pointer to the <code>USBConfig</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:



#### 6.20.4.4 void usbStop ( **USBDriver** \* *usbp* )

Deactivates the USB peripheral.

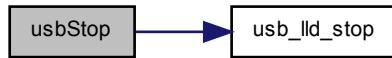
##### Parameters

in	<i>usbp</i> pointer to the <b>USBDriver</b> 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:



#### 6.20.4.5 void usblInitEndpointI ( **USBDriver** \* *usbp*, **usbep\_t** *ep*, const **USBEndpointConfig** \* *epcp* )

Enables an endpoint.

This function enables an endpoint, both IN and/or OUT directions depending on the configuration structure.

##### Note

This function must be invoked in response of a SET\_CONFIGURATION or SET\_INTERFACE message.

##### Parameters

in	<i>usbp</i> pointer to the <b>USBDriver</b> 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:



#### 6.20.4.6 void usbDisableEndpointsl ( **USBDriver** \* *usbp* )

Disables all the active endpoints.

This function disables all the active endpoints except the endpoint zero.

##### Note

This function must be invoked in response of a SET\_CONFIGURATION message with configuration number zero.

##### Parameters

in	<i>usbp</i> pointer to the <a href="#">USBDriver</a> 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:



#### 6.20.4.7 bool\_t usbStartReceive1 ( **USBDriver** \* *usbp*, **usbep\_t** *ep* )

Starts a receive transaction on an OUT endpoint.

##### Postcondition

The endpoint callback is invoked when the transfer has been completed.

##### Parameters

in	<i>usbp</i> pointer to the <a href="#">USBDriver</a> object
in	<i>ep</i> endpoint number

**Returns**

The operation status.

**Return values**

<i>FALSE</i>	Operation started successfully.
<i>TRUE</i>	Endpoint busy, operation not started.

**Function Class:**

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

Here is the call graph for this function:

**6.20.4.8 bool\_t usbStartTransmit( USBDriver \* *usbp*, usbep\_t *ep* )**

Starts a transmit transaction on an IN endpoint.

**Postcondition**

The endpoint callback is invoked when the transfer has been completed.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
in	<i>ep</i>	endpoint number

**Returns**

The operation status.

**Return values**

<i>FALSE</i>	Operation started successfully.
<i>TRUE</i>	Endpoint busy, operation not started.

**Function Class:**

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

Here is the call graph for this function:



#### 6.20.4.9 `bool_t usbStallReceive( USBDriver * usbp, usbep_t ep )`

Stalls an OUT endpoint.

##### Parameters

in	<code>usbp</code>	pointer to the <code>USBDriver</code> object
in	<code>ep</code>	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:



#### 6.20.4.10 `bool_t usbStallTransmit( USBDriver * usbp, usbep_t ep )`

Stalls an IN endpoint.

##### Parameters

in	<code>usbp</code>	pointer to the <code>USBDriver</code> object
in	<code>ep</code>	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:

**6.20.4.11 void \_usb\_reset( USBDriver \* usbp )**

USB reset routine.

This function must be invoked when an USB bus reset condition is detected.

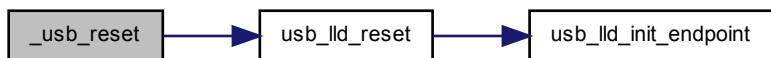
**Parameters**

in                   *usbp*   pointer to the [USBDriver](#) object

**Function Class:**

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

Here is the call graph for this function:

**6.20.4.12 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.

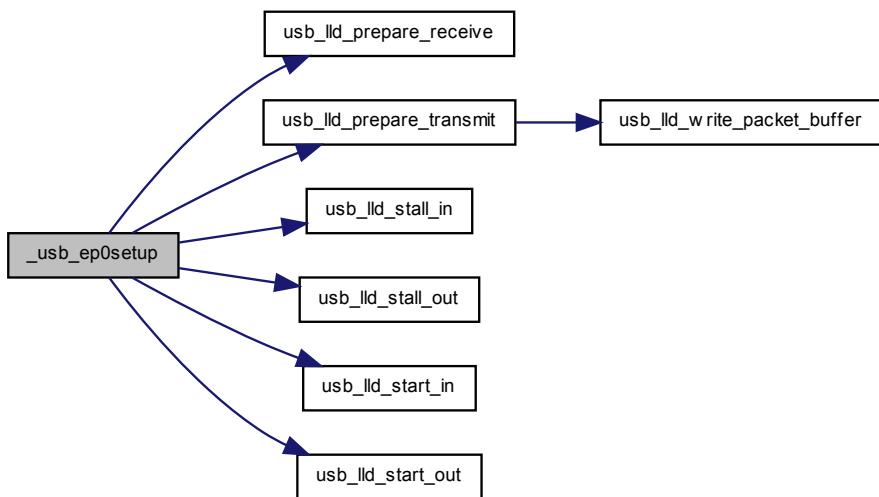
**Parameters**

in	<i>usbp</i> pointer to the <code>USBDriver</code> 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:

**6.20.4.13 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.

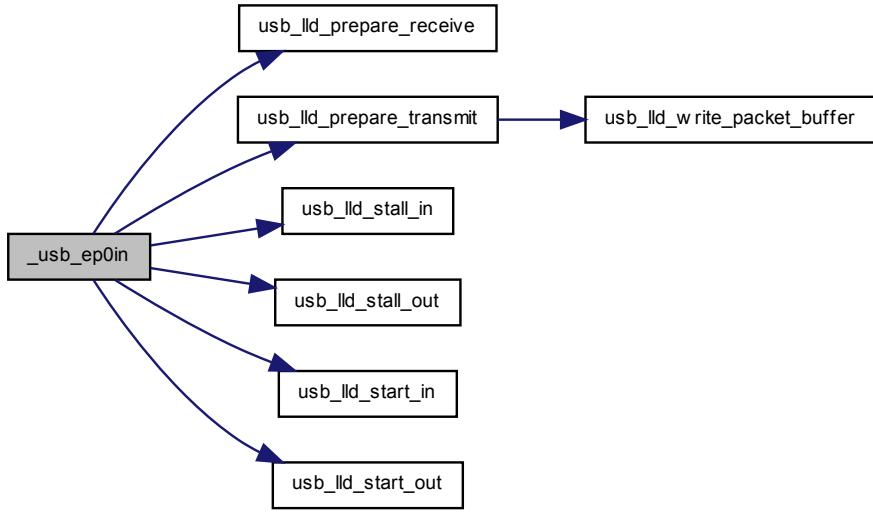
**Parameters**

in	<i>usbp</i> pointer to the <code>USBDriver</code> 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:



#### 6.20.4.14 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.

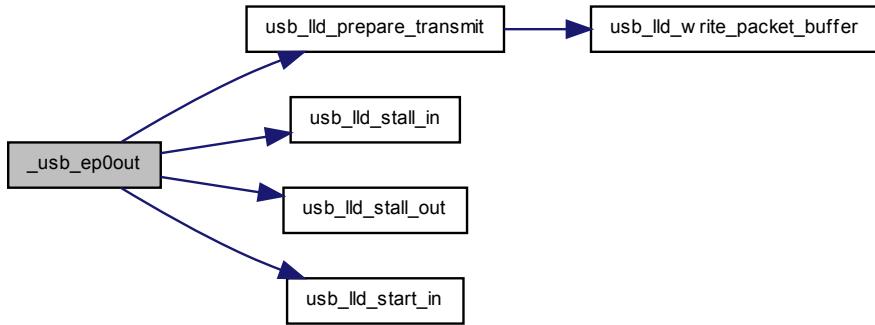
##### Parameters

in	<code>usbp</code>	pointer to the <code>USBDriver</code> object
in	<code>ep</code>	endpoint number, always zero

##### Function Class:

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

Here is the call graph for this function:



#### 6.20.4.15 CH\_IRQ\_HANDLER ( Vector8C )

USB high priority interrupt handler.

##### Function Class:

Interrupt handler, this function should not be directly invoked.

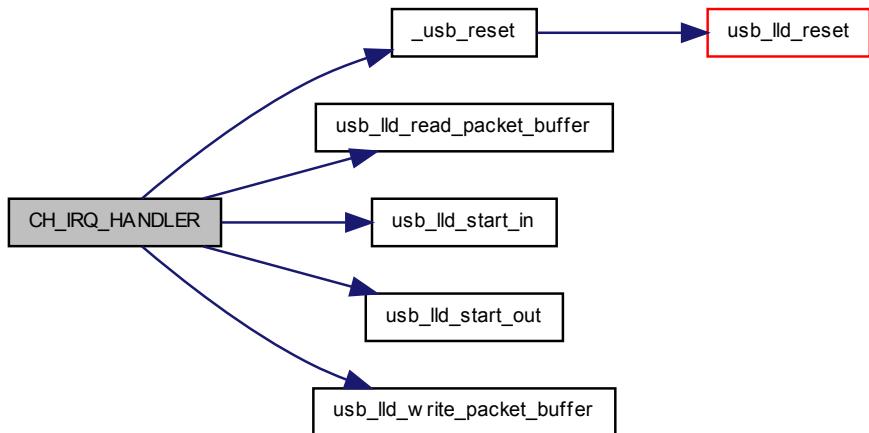
#### 6.20.4.16 CH\_IRQ\_HANDLER ( Vector90 )

USB low priority interrupt handler.

##### Function Class:

Interrupt handler, this function should not be directly invoked.

Here is the call graph for this function:



#### 6.20.4.17 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:



#### 6.20.4.18 void usb\_lld\_start( USBDriver \* usbp )

Configures and activates the USB peripheral.

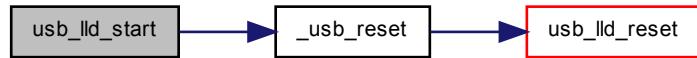
##### Parameters

in *usbp* pointer to the `USBDriver` object

##### Function Class:

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

Here is the call graph for this function:



#### 6.20.4.19 void usb\_lld\_stop ( **USBDriver** \* *usbp* )

Deactivates the USB peripheral.

##### Parameters

in *usbp* pointer to the **USBDriver** object

##### Function Class:

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

#### 6.20.4.20 void usb\_lld\_reset ( **USBDriver** \* *usbp* )

USB low level reset routine.

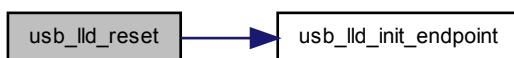
##### Parameters

in *usbp* pointer to the **USBDriver** object

##### Function Class:

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

Here is the call graph for this function:



#### 6.20.4.21 void usb\_lld\_set\_address ( **USBDriver** \* *usbp* )

Sets the USB address.

##### Parameters

in *usbp* pointer to the **USBDriver** object

**Function Class:**

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

6.20.4.22 void usb\_lld\_init\_endpoint ( **USBDriver** \* *usbp*, **usbep\_t** *ep* )

Enables an endpoint.

**Parameters**

in	<i>usbp</i>	pointer to the <b>USBDriver</b> object
in	<i>ep</i>	endpoint number

**Function Class:**

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

6.20.4.23 void usb\_lld\_disable\_endpoints ( **USBDriver** \* *usbp* )

Disables all the active endpoints except the endpoint zero.

**Parameters**

in	<i>usbp</i>	pointer to the <b>USBDriver</b> object
----	-------------	--

**Function Class:**

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

6.20.4.24 **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 <b>USBDriver</b> object
in	<i>ep</i>	endpoint number

**Returns**

The endpoint status.

**Return values**

*EP\_STATUS\_DISABLED* The endpoint is not active.

*EP\_STATUS\_STALLED* The endpoint is stalled.

*EP\_STATUS\_ACTIVE* The endpoint is active.

**Function Class:**

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

6.20.4.25 **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 <code>USBDriver</code> 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.

**6.20.4.26 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 <code>USBDriver</code> 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.

**6.20.4.27 size\_t usb\_lld\_read\_packet\_buffer ( `USBDriver` \* *usbp*, `usbep_t` *ep*, `uint8_t` \* *buf*, `size_t` *n* )**

Reads from a dedicated packet buffer.

**Precondition**

In order to use this function the endpoint must have been initialized in packet mode.

**Note**

This function can be invoked both in thread and IRQ context.

**Parameters**

in	<i>usbp</i>	pointer to the <code>USBDriver</code> object
in	<i>ep</i>	endpoint number
out	<i>buf</i>	buffer where to copy the packet data
in	<i>n</i>	maximum number of bytes to copy. This value must not exceed the maximum packet size for this endpoint.

**Returns**

The received packet size regardless the specified *n* parameter.

**Return values**

0 Zero size packet received.

**Function Class:**

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

6.20.4.28 void usb\_lld\_write\_packet\_buffer ( **USBDriver** \* *usbp*, **usbep\_t** *ep*, const **uint8\_t** \* *buf*, **size\_t** *n* )

Writes to a dedicated packet buffer.

**Precondition**

In order to use this function the endpoint must have been initialized in packet mode.

**Note**

This function can be invoked both in thread and IRQ context.

**Parameters**

in	<i>usbp</i>	pointer to the <b>USBDriver</b> object
in	<i>ep</i>	endpoint number
in	<i>buf</i>	buffer where to fetch the packet data
in	<i>n</i>	maximum number of bytes to copy. This value must not exceed the maximum packet size for this endpoint.

**Function Class:**

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

6.20.4.29 void usb\_lld\_prepare\_receive ( **USBDriver** \* *usbp*, **usbep\_t** *ep*, **uint8\_t** \* *buf*, **size\_t** *n* )

Prepares for a receive operation.

**Parameters**

in	<i>usbp</i>	pointer to the <b>USBDriver</b> object
in	<i>ep</i>	endpoint number
out	<i>buf</i>	buffer where to copy the received data
in	<i>n</i>	maximum number of bytes to copy

**Function Class:**

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

6.20.4.30 void usb\_lld\_prepare\_transmit ( **USBDriver** \* *usbp*, **usbep\_t** *ep*, const **uint8\_t** \* *buf*, **size\_t** *n* )

Prepares for a transmit operation.

**Parameters**

in	<i>usbp</i>	pointer to the <b>USBDriver</b> object
in	<i>ep</i>	endpoint number
in	<i>buf</i>	buffer where to fetch the data to be transmitted
in	<i>n</i>	maximum number of bytes to copy

**Function Class:**

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

Here is the call graph for this function:

**6.20.4.31 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 <a href="#">USBDriver</a> object
in	<i>ep</i>	endpoint number

**Function Class:**

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

**6.20.4.32 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 <a href="#">USBDriver</a> object
in	<i>ep</i>	endpoint number

**Function Class:**

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

**6.20.4.33 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 <a href="#">USBDriver</a> object
in	<i>ep</i>	endpoint number

**Function Class:**

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

**6.20.4.34 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 <b>USBDriver</b> object
in	<i>ep</i>	endpoint number

**Function Class:**

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

**6.20.4.35 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 <b>USBDriver</b> object
in	<i>ep</i>	endpoint number

**Function Class:**

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

**6.20.4.36 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 <b>USBDriver</b> object
in	<i>ep</i>	endpoint number

**Function Class:**

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

**6.20.5 Variable Documentation****6.20.5.1 USBDriver USBD1**

USB1 driver identifier.

**6.20.5.2 USBInEndpointState { ... } in**

IN EP0 state.

**6.20.5.3 USBOutEndpointState { ... } out**

OUT EP0 state.

**6.20.6 Define Documentation**

6.20.6.1 #define USB\_DESC\_INDEX( *i* ) ((uint8\_t)(*i*))

Helper macro for index values into descriptor strings.

6.20.6.2 #define USB\_DESC\_BYTE( *b* ) ((uint8\_t)(*b*))

Helper macro for byte values into descriptor strings.

6.20.6.3 #define USB\_DESC\_WORD( *w* )**Value:**

```
(uint8_t) ((w) & 255),  
          \  
(uint8_t) (((w) >> 8) & 255)
```

Helper macro for word values into descriptor strings.

6.20.6.4 #define USB\_DESC\_BCD( *bcd* )**Value:**

```
(uint8_t) ((bcd) & 255),  
          \  
(uint8_t) (((bcd) >> 8) & 255)
```

Helper macro for BCD values into descriptor strings.

6.20.6.5 #define USB\_DESC\_DEVICE( *bcdUSB*, *bDeviceClass*, *bDeviceSubClass*, *bDeviceProtocol*, *bMaxPacketSize*,  
*idVendor*, *idProduct*, *bcdDevice*, *iManufacturer*, *iProduct*, *iSerialNumber*, *bNumConfigurations* )**Value:**

```
USB_DESC_BYTE (18),  
          \  
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.

6.20.6.6 #define USB\_DESC\_CONFIGURATION( *wTotalLength*, *bNumInterfaces*, *bConfigurationValue*, *iConfiguration*,  
*bmAttributes*, *bMaxPower* )**Value:**

```
USB_DESC_BYTE (9),  
          \  
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.

```
6.20.6.7 #define USB_DESC_INTERFACE( bInterfaceNumber, bAlternateSetting, bNumEndpoints, bInterfaceClass,  
bInterfaceSubClass, bInterfaceProtocol, iInterface )
```

**Value:**

```
USB_DESC_BYTE(9),  
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.

```
6.20.6.8 #define USB_DESC_ENDPOINT( bEndpointAddress, bmAttributes, wMaxPacketSize, bInterval )
```

**Value:**

```
USB_DESC_BYTE(7),  
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.

```
6.20.6.9 #define USB_EP_MODE_TYPE 0x0003
```

Endpoint type mask.

```
6.20.6.10 #define USB_EP_MODE_TYPE_CTRL 0x0000
```

Control endpoint.

```
6.20.6.11 #define USB_EP_MODE_TYPE_ISOC 0x0001
```

Isochronous endpoint.

```
6.20.6.12 #define USB_EP_MODE_TYPE_BULK 0x0002
```

Bulk endpoint.

```
6.20.6.13 #define USB_EP_MODE_TYPE_INTR 0x0003
```

Interrupt endpoint.

```
6.20.6.14 #define USB_EP_MODE_TRANSACTION 0x0000
```

Transaction mode.

```
6.20.6.15 #define USB_EP_MODE_PACKET 0x0010
```

Packet mode enabled.

```
6.20.6.16 #define usbConnectBus( usbp ) usb_lld_connect_bus(usbp)
```

Connects the USB device.

```
6.20.6.17 #define usbDisconnectBus( usbp ) usb_lld_disconnect_bus(usbp)
```

Disconnect the USB device.

```
6.20.6.18 #define usbGetFrameNumber( usbp ) usb_lld_get_frame_number(usbp)
```

Returns the current frame number.

#### Parameters

in *usbp* pointer to the [USBDriver](#) object

#### Returns

The current frame number.

#### Function Class:

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

```
6.20.6.19 #define usbGetTransmitStatus( usbp, ep ) ((usbp)>transmitting & (1 << (ep)))
```

Returns the status of an IN endpoint.

#### Parameters

in *usbp* pointer to the [USBDriver](#) object  
in *ep* endpoint number

#### Returns

The operation status.

#### Return values

*FALSE* Endpoint ready.  
*TRUE* Endpoint transmitting.

#### Function Class:

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

```
6.20.6.20 #define usbGetReceiveStatus( usbp, ep ) ((usbp)>receiving & (1 << (ep)))
```

Returns the status of an OUT endpoint.

#### Parameters

in *usbp* pointer to the [USBDriver](#) object  
in *ep* 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.

6.20.6.21 #define `usbReadPacketBuffer( usbp, ep, buf, n ) usb_lld_read_packet_buffer(usbp, ep, buf, n)`

Reads from a dedicated packet buffer.

**Precondition**

In order to use this function the endpoint must have been initialized in packet mode.

**Note**

This function can be invoked both in thread and IRQ context.

**Parameters**

in	<i>usbp</i>	pointer to the <code>USBDriver</code> object
in	<i>ep</i>	endpoint number
out	<i>buf</i>	buffer where to copy the packet data
in	<i>n</i>	maximum number of bytes to copy. This value must not exceed the maximum packet size for this endpoint.

**Returns**

The received packet size regardless the specified *n* parameter.

**Return values**

<i>0</i>	Zero size packet received.
----------	----------------------------

**Function Class:**

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

6.20.6.22 #define `usbWritePacketBuffer( usbp, ep, buf, n ) usb_lld_write_packet_buffer(usbp, ep, buf, n)`

Writes to a dedicated packet buffer.

**Precondition**

In order to use this function the endpoint must have been initialized in packet mode.

**Note**

This function can be invoked both in thread and IRQ context.

**Parameters**

in	<i>usbp</i>	pointer to the <code>USBDriver</code> object
in	<i>ep</i>	endpoint number
in	<i>buf</i>	buffer where to fetch the packet data
in	<i>n</i>	maximum number of bytes to copy. This value must not exceed the maximum packet size for this endpoint.

**Function Class:**

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

**6.20.6.23 #define usbPrepareReceive( *usbp*, *ep*, *buf*, *n* ) usb\_lld\_prepare\_receive(*usbp*, *ep*, *buf*, *n*)**

Prepares for a receive transaction on an OUT endpoint.

**Precondition**

In order to use this function the endpoint must have been initialized in transaction mode.

**Postcondition**

The endpoint is ready for [usbStartReceiveI\(\)](#).

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
in	<i>ep</i>	endpoint number
out	<i>buf</i>	buffer where to copy the received data
in	<i>n</i>	maximum number of bytes to copy

**Function Class:**

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

**6.20.6.24 #define usbPrepareTransmit( *usbp*, *ep*, *buf*, *n* ) usb\_lld\_prepare\_transmit(*usbp*, *ep*, *buf*, *n*)**

Prepares for a transmit transaction on an IN endpoint.

**Precondition**

In order to use this function the endpoint must have been initialized in transaction mode.

**Postcondition**

The endpoint is ready for [usbStartTransmitI\(\)](#).

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
in	<i>ep</i>	endpoint number
in	<i>buf</i>	buffer where to fetch the data to be transmitted
in	<i>n</i>	maximum number of bytes to copy

**Function Class:**

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

**6.20.6.25 #define usbGetReceiveTransactionSize( *usbp*, *ep* ) usb\_lld\_get\_transaction\_size(*usbp*, *ep*)**

Returns the exact size of a receive transaction.

The received size can be different from the size specified in [usbStartReceiveI\(\)](#) because the last packet could have a size different from the expected one.

**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:**

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

6.20.6.26 #define `usbGetReceivePacketSize( usbp, ep )` `usb_lld_get_packet_size(usbp, ep)`

Returns the exact size of a received packet.

**Precondition**

The OUT endpoint must have been configured in packet 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:**

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

6.20.6.27 #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 NULL

**Function Class:**

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

---

```
6.20.6.28 #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 <code>USBDriver</code> 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.

---

```
6.20.6.29 #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 <code>USBDriver</code> object
in	<i>evt</i>	USB event code

#### Function Class:

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

---

```
6.20.6.30 #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 <code>USBDriver</code> object
----	-------------	--

#### Function Class:

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

6.20.6.31 #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 <a href="#">USBDriver</a> object
in	<i>ep</i>	endpoint number

**Function Class:**

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

6.20.6.32 #define \_usb\_isr\_invoke\_in\_cb( *usbp*, *ep* )

**Value:**

```
{
    (usbp)->transmitting &= ~(1 << (ep));
    (usbp)->epc[ep]->in_cb(usbp, ep);
}
```

Common ISR code, IN endpoint callback.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
in	<i>ep</i>	endpoint number

**Function Class:**

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

6.20.6.33 #define \_usb\_isr\_invoke\_out\_cb( *usbp*, *ep* )

**Value:**

```
{
    (usbp)->receiving &= ~(1 << (ep));
    (usbp)->epc[ep]->out_cb(usbp, ep);
}
```

Common ISR code, OUT endpoint event.

**Parameters**

in	<i>usbp</i>	pointer to the <a href="#">USBDriver</a> object
in	<i>ep</i>	endpoint number

**Function Class:**

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

## 6.20.6.34 #define USB\_ENDOPOINTS\_NUMBER 7

Number of the available endpoints.

This value does not include the endpoint 0 which is always present.

## 6.20.6.35 #define STM32\_USB\_BASE (APB1PERIPH\_BASE + 0x5C00)

USB registers block numeric address.

## 6.20.6.36 #define STM32\_USBRAM\_BASE (APB1PERIPH\_BASE + 0x6000)

USB RAM numeric address.

## 6.20.6.37 #define STM32\_USB ((stm32\_usb\_t \*)STM32\_USB\_BASE)

Pointer to the USB registers block.

## 6.20.6.38 #define STM32\_USBRAM ((uint32\_t \*)STM32\_USBRAM\_BASE)

Pointer to the USB RAM.

## 6.20.6.39 #define USB\_PMA\_SIZE 512

Size of the dedicated packet memory.

## 6.20.6.40 #define EPR\_TOGGLE\_MASK

**Value:**

```
(EPR_STAT_TX_MASK | EPR_DTOG_TX | \
     \ \
     EPR_STAT_RX_MASK | EPR_DTOG_RX | \
     EPR_SETUP)
```

Mask of all the toggling bits in the EPR register.

## 6.20.6.41 #define USB\_GET\_DESCRIPTOR( ep )

**Value:**

```
((stm32_usb_descriptor_t *)((uint32_t)STM32_USBRAM_BASE + \
     (uint32_t)STM32_USB->BTABLE * 2 + \
     (uint32_t)(ep) * \
     sizeof(stm32_usb_descriptor_t))) \ \
     \ \
     \
```

Returns an endpoint descriptor pointer.

## 6.20.6.42 #define USB\_ADDR2PTR( addr ) ((uint32\_t \*)((addr) \* 2 + STM32\_USBRAM\_BASE))

Converts from a PMA address to a physical address.

## 6.20.6.43 #define USB\_MAX\_ENDPOINTS USB\_ENDOPOINTS\_NUMBER

Maximum endpoint address.

6.20.6.44 #define USB\_SET\_ADDRESS\_MODE USB\_LATE\_SET\_ADDRESS

This device requires the address change after the status packet.

6.20.6.45 #define STM32\_USB\_USE\_USB1 TRUE

USB1 driver enable switch.

If set to TRUE the support for USB1 is included.

#### Note

The default is TRUE.

6.20.6.46 #define STM32\_USB\_LOW\_POWER\_ON\_SUSPEND FALSE

Enables the USB device low power mode on suspend.

6.20.6.47 #define STM32\_USB\_USB1\_HP\_IRQ\_PRIORITY 6

USB1 interrupt priority level setting.

6.20.6.48 #define STM32\_USB\_USB1\_LP\_IRQ\_PRIORITY 14

USB1 interrupt priority level setting.

6.20.6.49 #define usb\_lld\_fetch\_word( *p* ) (\*(uint16\_t \*)(*p*))

Fetches a 16 bits word value from an USB message.

#### Parameters

in *p* pointer to the 16 bits word

#### Function Class:

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

6.20.6.50 #define usb\_lld\_get\_frame\_number( *usbp* ) (STM32\_USB->FNR & FNR\_FN\_MASK)

Returns the current frame number.

#### Parameters

in *usbp* pointer to the [USBDriver](#) object

#### Returns

The current frame number.

#### Function Class:

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

```
6.20.6.51 #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.

```
6.20.6.52 #define usb_lld_get_packet_size( usbp, ep ) ((size_t)USB_GET_DESCRIPTOR(ep)->RXCOUNT & RXCOUNT_COUNT_MASK)
```

Returns the exact size of a received packet.

#### Precondition

The OUT endpoint must have been configured in packet 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.

## 6.20.7 Typedef Documentation

```
6.20.7.1 typedef struct USBDriver USBDriver
```

Type of a structure representing an USB driver.

```
6.20.7.2 typedef uint8_t usbep_t
```

Type of an endpoint identifier.

```
6.20.7.3 typedef void(* usbcallback_t)(USBDriver *usbp)
```

Type of an USB generic notification callback.

**Parameters**

in *usbp* pointer to the `USBDriver` object triggering the callback

**6.20.7.4 `typedef void(* usbepcallback_t)(USBDriver *usbp, usbep_t ep)`**

Type of an USB endpoint callback.

**Parameters**

in	<i>usbp</i> pointer to the <code>USBDriver</code> object triggering the callback
in	<i>ep</i> endpoint number

**6.20.7.5 `typedef void(* usbeventcb_t)(USBDriver *usbp, usbevent_t event)`**

Type of an USB event notification callback.

**Parameters**

in	<i>usbp</i> pointer to the <code>USBDriver</code> object triggering the callback
in	<i>event</i> event type

**6.20.7.6 `typedef bool_t(* 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 <code>USBDriver</code> object triggering the callback
----	--

**Returns**

The request handling exit code.

**Return values**

*FALSE* Request not recognized by the handler.

*TRUE* Request handled.

**6.20.7.7 `typedef const USBDescriptor*(* usbgetdescriptor_t)(USBDriver *usbp, uint8_t dtype, uint8_t dindex, uint16_t lang)`**

Type of an USB descriptor-retrieving callback.

**6.20.8 Enumeration Type Documentation****6.20.8.1 `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.

#### 6.20.8.2 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.

#### 6.20.8.3 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\_STS** Waiting status.

**USB\_EP0\_RX** Receiving.

**USB\_EP0\_SENDING\_STS** Sending status.

**USB\_EP0\_ERROR** Error, EP0 stalled.

#### 6.20.8.4 enum usbevent\_t

Type of an enumeration of the possible USB events.

**Enumerator:**

**USB\_EVENT\_RESET** Driver has been reset by host.

**USB\_EVENT\_ADDRESS** Address assigned.

**USB\_EVENT\_CONFIGURED** Configuration selected.

**USB\_EVENT\_SUSPEND** Entering suspend mode.

**USB\_EVENT\_WAKEUP** Leaving suspend mode.

**USB\_EVENT\_STALLED** Endpoint 0 error, stalled.

## 6.21 STM32F100 HAL Support

### 6.21.1 Detailed Description

HAL support for STM32 Value Line LD, MD and HD sub-families.

#### Platform identification

- #define **PLATFORM\_NAME** "STM32F1 Value Line"

## Absolute Maximum Ratings

- #define **STM32\_SYSCLK\_MAX** 24000000  
*Maximum system clock frequency.*
- #define **STM32\_HSECLK\_MAX** 24000000  
*Maximum HSE clock frequency.*
- #define **STM32\_HSECLK\_MIN** 1000000  
*Minimum HSE clock frequency.*
- #define **STM32\_LSECLK\_MAX** 1000000  
*Maximum LSE clock frequency.*
- #define **STM32\_LSECLK\_MIN** 32768  
*Minimum LSE clock frequency.*
- #define **STM32\_PLLIN\_MAX** 24000000  
*Maximum PLLs input clock frequency.*
- #define **STM32\_PLLIN\_MIN** 1000000  
*Maximum PLLs input clock frequency.*
- #define **STM32\_PLLOUT\_MAX** 24000000  
*Maximum PLL output clock frequency.*
- #define **STM32\_PLLOUT\_MIN** 16000000  
*Maximum PLL output clock frequency.*
- #define **STM32\_PCLK1\_MAX** 24000000  
*Maximum APB1 clock frequency.*
- #define **STM32\_PCLK2\_MAX** 24000000  
*Maximum APB2 clock frequency.*
- #define **STM32\_ADCCLK\_MAX** 12000000  
*Maximum ADC clock frequency.*

## RCC\_CFGR register bits definitions

- #define **STM32\_SW\_HSI** (0 << 0)
- #define **STM32\_SW\_HSE** (1 << 0)
- #define **STM32\_SW\_PLL** (2 << 0)
- #define **STM32\_HPRE\_DIV1** (0 << 4)
- #define **STM32\_HPRE\_DIV2** (8 << 4)
- #define **STM32\_HPRE\_DIV4** (9 << 4)
- #define **STM32\_HPRE\_DIV8** (10 << 4)
- #define **STM32\_HPRE\_DIV16** (11 << 4)
- #define **STM32\_HPRE\_DIV64** (12 << 4)
- #define **STM32\_HPRE\_DIV128** (13 << 4)
- #define **STM32\_HPRE\_DIV256** (14 << 4)
- #define **STM32\_HPRE\_DIV512** (15 << 4)
- #define **STM32\_PPREG1\_DIV1** (0 << 8)
- #define **STM32\_PPREG1\_DIV2** (4 << 8)
- #define **STM32\_PPREG1\_DIV4** (5 << 8)
- #define **STM32\_PPREG1\_DIV8** (6 << 8)
- #define **STM32\_PPREG1\_DIV16** (7 << 8)
- #define **STM32\_PPREG2\_DIV1** (0 << 11)
- #define **STM32\_PPREG2\_DIV2** (4 << 11)
- #define **STM32\_PPREG2\_DIV4** (5 << 11)
- #define **STM32\_PPREG2\_DIV8** (6 << 11)
- #define **STM32\_PPREG2\_DIV16** (7 << 11)
- #define **STM32\_ADCPRE\_DIV2** (0 << 14)
- #define **STM32\_ADCPRE\_DIV4** (1 << 14)

- #define STM32\_ADCPRE\_DIV6 (2 << 14)
- #define STM32\_ADCPRE\_DIV8 (3 << 14)
- #define STM32\_PLLSRC\_HSI (0 << 16)
- #define STM32\_PLLSRC\_HSE (1 << 16)
- #define STM32\_PLLXTPRE\_DIV1 (0 << 17)
- #define STM32\_PLLXTPRE\_DIV2 (1 << 17)
- #define STM32\_MCOSEL\_NOCLOCK (0 << 24)
- #define STM32\_MCOSEL\_SYSCLK (4 << 24)
- #define STM32\_MCOSEL\_HSI (5 << 24)
- #define STM32\_MCOSEL\_HSE (6 << 24)
- #define STM32\_MCOSEL\_PLLDIV2 (7 << 24)
- #define STM32\_RTCSEL\_MASK (3 << 8)
- #define STM32\_RTCSEL\_NOCLOCK (0 << 8)
- #define STM32\_RTCSEL\_LSE (1 << 8)
- #define STM32\_RTCSEL\_LSI (2 << 8)
- #define STM32\_RTCSEL\_HSEDIV (3 << 8)

### STM32F100 LD capabilities

- #define STM32\_RTCSEL\_HAS\_SUBSECONDS TRUE
- #define STM32\_USART2\_RX\_DMA\_MSK (STM32\_DMA\_STREAM\_ID\_MSK(1, 6))
- #define STM32\_USART2\_RX\_DMA\_CHN 0x00000000
- #define STM32\_USART2\_TX\_DMA\_MSK (STM32\_DMA\_STREAM\_ID\_MSK(1, 7))
- #define STM32\_USART2\_TX\_DMA\_CHN 0x00000000

### STM32F100 MD capabilities

- #define STM32\_HAS\_ADC1 TRUE
- #define STM32\_HAS\_ADC1 TRUE
- #define STM32\_HAS\_ADC2 FALSE
- #define STM32\_HAS\_ADC2 FALSE
- #define STM32\_HAS\_ADC3 FALSE
- #define STM32\_HAS\_ADC3 FALSE
- #define STM32\_HAS\_CAN1 FALSE
- #define STM32\_HAS\_CAN1 FALSE
- #define STM32\_HAS\_CAN2 FALSE
- #define STM32\_HAS\_CAN2 FALSE
- #define STM32\_HAS\_DAC TRUE
- #define STM32\_HAS\_DAC TRUE
- #define STM32\_ADVANCED\_DMA FALSE
- #define STM32\_ADVANCED\_DMA FALSE
- #define STM32\_HAS\_DMA1 TRUE
- #define STM32\_HAS\_DMA1 TRUE
- #define STM32\_HAS\_DMA2 FALSE
- #define STM32\_HAS\_DMA2 FALSE
- #define STM32\_HAS\_ETH FALSE
- #define STM32\_HAS\_ETH FALSE
- #define STM32\_EXTI\_NUM\_CHANNELS 18
- #define STM32\_EXTI\_NUM\_CHANNELS 19
- #define STM32\_HAS\_GPIOA TRUE
- #define STM32\_HAS\_GPIOA TRUE
- #define STM32\_HAS\_GPIOB TRUE
- #define STM32\_HAS\_GPIOB TRUE
- #define STM32\_HAS\_GPIOC TRUE

- #define **STM32\_HAS\_GPIOC** TRUE
- #define **STM32\_HAS\_GPIOD** TRUE
- #define **STM32\_HAS\_GPIOD** TRUE
- #define **STM32\_HAS\_GPIOE** TRUE
- #define **STM32\_HAS\_GPIOE** TRUE
- #define **STM32\_HAS\_GPIOF** FALSE
- #define **STM32\_HAS\_GPIOF** FALSE
- #define **STM32\_HAS\_GPIOG** FALSE
- #define **STM32\_HAS\_GPIOG** FALSE
- #define **STM32\_HAS\_GPIOH** FALSE
- #define **STM32\_HAS\_GPIOH** FALSE
- #define **STM32\_HAS\_GPIOI** FALSE
- #define **STM32\_HAS\_GPIOI** FALSE
- #define **STM32\_HAS\_I2C1** TRUE
- #define **STM32\_HAS\_I2C1** TRUE
- #define **STM32\_I2C1\_RX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 7))
- #define **STM32\_I2C1\_RX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 7))
- #define **STM32\_I2C1\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_I2C1\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_I2C1\_TX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 6))
- #define **STM32\_I2C1\_TX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 6))
- #define **STM32\_I2C1\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_I2C1\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_I2C2** FALSE
- #define **STM32\_HAS\_I2C2** TRUE
- #define **STM32\_I2C2\_RX\_DMA\_MSK** 0
- #define **STM32\_I2C2\_RX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 5))
- #define **STM32\_I2C2\_RX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 5))
- #define **STM32\_I2C2\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_I2C2\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_I2C2\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_I2C2\_TX\_DMA\_MSK** 0
- #define **STM32\_I2C2\_TX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 4))
- #define **STM32\_I2C2\_TX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 4))
- #define **STM32\_I2C2\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_I2C2\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_I2C2\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_I2C3** FALSE
- #define **STM32\_HAS\_I2C3** FALSE
- #define **STM32\_SPI3\_RX\_DMA\_MSK** 0
- #define **STM32\_SPI3\_RX\_DMA\_MSK** 0
- #define **STM32\_SPI3\_RX\_DMA\_MSK** 0
- #define **STM32\_SPI3\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_SPI3\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_SPI3\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_SPI3\_TX\_DMA\_MSK** 0
- #define **STM32\_SPI3\_TX\_DMA\_MSK** 0
- #define **STM32\_SPI3\_TX\_DMA\_MSK** 0
- #define **STM32\_SPI3\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_SPI3\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_SPI3\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_RTC** TRUE
- #define **STM32\_HAS\_RTC** TRUE
- #define **STM32\_HAS\_SDIO** FALSE
- #define **STM32\_HAS\_SDIO** FALSE

- #define **STM32\_HAS\_SPI1** TRUE
- #define **STM32\_HAS\_SPI1** TRUE
- #define **STM32\_SPI1\_RX\_DMA\_MSK** STM32\_DMA\_STREAM\_ID\_MSK(1, 2)
- #define **STM32\_SPI1\_RX\_DMA\_MSK** STM32\_DMA\_STREAM\_ID\_MSK(1, 2)
- #define **STM32\_SPI1\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_SPI1\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_SPI1\_TX\_DMA\_MSK** STM32\_DMA\_STREAM\_ID\_MSK(1, 3)
- #define **STM32\_SPI1\_TX\_DMA\_MSK** STM32\_DMA\_STREAM\_ID\_MSK(1, 3)
- #define **STM32\_SPI1\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_SPI1\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_SPI2** FALSE
- #define **STM32\_HAS\_SPI2** TRUE
- #define **STM32\_SPI2\_RX\_DMA\_MSK** 0
- #define **STM32\_SPI2\_RX\_DMA\_MSK** STM32\_DMA\_STREAM\_ID\_MSK(1, 4)
- #define **STM32\_SPI2\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_SPI2\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_SPI2\_TX\_DMA\_MSK** 0
- #define **STM32\_SPI2\_TX\_DMA\_MSK** STM32\_DMA\_STREAM\_ID\_MSK(1, 5)
- #define **STM32\_SPI2\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_SPI2\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_SPI3** FALSE
- #define **STM32\_HAS\_SPI3** FALSE
- #define **STM32\_HAS\_TIM1** TRUE
- #define **STM32\_HAS\_TIM1** TRUE
- #define **STM32\_HAS\_TIM2** TRUE
- #define **STM32\_HAS\_TIM2** TRUE
- #define **STM32\_HAS\_TIM3** TRUE
- #define **STM32\_HAS\_TIM3** TRUE
- #define **STM32\_HAS\_TIM4** FALSE
- #define **STM32\_HAS\_TIM4** TRUE
- #define **STM32\_HAS\_TIM5** FALSE
- #define **STM32\_HAS\_TIM5** FALSE
- #define **STM32\_HAS\_TIM6** TRUE
- #define **STM32\_HAS\_TIM6** TRUE
- #define **STM32\_HAS\_TIM7** TRUE
- #define **STM32\_HAS\_TIM7** TRUE
- #define **STM32\_HAS\_TIM8** FALSE
- #define **STM32\_HAS\_TIM8** FALSE
- #define **STM32\_HAS\_TIM9** FALSE
- #define **STM32\_HAS\_TIM9** FALSE
- #define **STM32\_HAS\_TIM10** FALSE
- #define **STM32\_HAS\_TIM10** FALSE
- #define **STM32\_HAS\_TIM11** FALSE
- #define **STM32\_HAS\_TIM11** FALSE
- #define **STM32\_HAS\_TIM12** FALSE
- #define **STM32\_HAS\_TIM12** FALSE
- #define **STM32\_HAS\_TIM13** FALSE
- #define **STM32\_HAS\_TIM13** FALSE
- #define **STM32\_HAS\_TIM14** FALSE
- #define **STM32\_HAS\_TIM14** FALSE
- #define **STM32\_HAS\_TIM15** TRUE
- #define **STM32\_HAS\_TIM15** TRUE
- #define **STM32\_HAS\_TIM16** TRUE
- #define **STM32\_HAS\_TIM16** TRUE
- #define **STM32\_HAS\_TIM17** TRUE

- #define STM32\_HAS\_TIM17 TRUE
- #define STM32\_HAS\_USART1 TRUE
- #define STM32\_HAS\_USART1 TRUE
- #define STM32\_USART1\_RX\_DMA\_MSK (STM32\_DMA\_STREAM\_ID\_MSK(1, 5))
- #define STM32\_USART1\_RX\_DMA\_MSK (STM32\_DMA\_STREAM\_ID\_MSK(1, 5))
- #define STM32\_USART1\_RX\_DMA\_CHN 0x00000000
- #define STM32\_USART1\_RX\_DMA\_CHN 0x00000000
- #define STM32\_USART1\_TX\_DMA\_MSK (STM32\_DMA\_STREAM\_ID\_MSK(1, 4))
- #define STM32\_USART1\_TX\_DMA\_MSK (STM32\_DMA\_STREAM\_ID\_MSK(1, 4))
- #define STM32\_USART1\_TX\_DMA\_CHN 0x00000000
- #define STM32\_USART1\_TX\_DMA\_CHN 0x00000000
- #define STM32\_HAS\_USART2 TRUE
- #define STM32\_HAS\_USART2 TRUE
- #define STM32\_HAS\_USART3 FALSE
- #define STM32\_HAS\_USART3 TRUE
- #define STM32\_USART3\_RX\_DMA\_MSK 0
- #define STM32\_USART3\_RX\_DMA\_MSK (STM32\_DMA\_STREAM\_ID\_MSK(1, 3))
- #define STM32\_USART3\_RX\_DMA\_CHN 0x00000000
- #define STM32\_USART3\_RX\_DMA\_CHN 0x00000000
- #define STM32\_USART3\_TX\_DMA\_MSK 0
- #define STM32\_USART3\_TX\_DMA\_MSK (STM32\_DMA\_STREAM\_ID\_MSK(1, 2))
- #define STM32\_USART3\_TX\_DMA\_CHN 0x00000000
- #define STM32\_USART3\_TX\_DMA\_CHN 0x00000000
- #define STM32\_HAS\_UART4 FALSE
- #define STM32\_HAS\_UART4 FALSE
- #define STM32\_UART4\_RX\_DMA\_MSK 0
- #define STM32\_UART4\_RX\_DMA\_MSK 0
- #define STM32\_UART4\_RX\_DMA\_CHN 0x00000000
- #define STM32\_UART4\_RX\_DMA\_CHN 0x00000000
- #define STM32\_UART4\_TX\_DMA\_MSK 0
- #define STM32\_UART4\_TX\_DMA\_MSK 0
- #define STM32\_UART4\_TX\_DMA\_CHN 0x00000000
- #define STM32\_UART4\_TX\_DMA\_CHN 0x00000000
- #define STM32\_HAS\_UART5 FALSE
- #define STM32\_HAS\_UART5 FALSE
- #define STM32\_UART5\_RX\_DMA\_MSK 0
- #define STM32\_UART5\_RX\_DMA\_MSK 0
- #define STM32\_UART5\_RX\_DMA\_CHN 0x00000000
- #define STM32\_UART5\_RX\_DMA\_CHN 0x00000000
- #define STM32\_UART5\_TX\_DMA\_MSK 0
- #define STM32\_UART5\_TX\_DMA\_MSK 0
- #define STM32\_UART5\_TX\_DMA\_CHN 0x00000000
- #define STM32\_UART5\_TX\_DMA\_CHN 0x00000000
- #define STM32\_HAS\_USART6 FALSE
- #define STM32\_HAS\_USART6 FALSE
- #define STM32\_USART6\_RX\_DMA\_MSK 0
- #define STM32\_USART6\_RX\_DMA\_MSK 0
- #define STM32\_USART6\_RX\_DMA\_CHN 0x00000000
- #define STM32\_USART6\_RX\_DMA\_CHN 0x00000000
- #define STM32\_USART6\_TX\_DMA\_MSK 0
- #define STM32\_USART6\_TX\_DMA\_MSK 0
- #define STM32\_USART6\_TX\_DMA\_CHN 0x00000000
- #define STM32\_USART6\_TX\_DMA\_CHN 0x00000000
- #define STM32\_HAS\_USB FALSE
- #define STM32\_HAS\_USB FALSE

- #define **STM32\_HAS\_OTG1** FALSE
- #define **STM32\_HAS\_OTG1** FALSE
- #define **STM32\_HAS\_OTG2** FALSE
- #define **STM32\_HAS\_OTG2** FALSE
- #define **STM32\_I2C3\_RX\_DMA\_MSK** 0
- #define **STM32\_I2C3\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_I2C3\_TX\_DMA\_MSK** 0
- #define **STM32\_I2C3\_TX\_DMA\_CHN** 0x00000000

## IRQ VECTOR names

- #define **WWDG\_IRQHandler** Vector40
- #define **PVD\_IRQHandler** Vector44
- #define **TAMPER\_IRQHandler** Vector48
- #define **RTC\_IRQHandler** Vector4C
- #define **FLASH\_IRQHandler** Vector50
- #define **RCC\_IRQHandler** Vector54
- #define **EXTI0\_IRQHandler** Vector58
- #define **EXTI1\_IRQHandler** Vector5C
- #define **EXTI2\_IRQHandler** Vector60
- #define **EXTI3\_IRQHandler** Vector64
- #define **EXTI4\_IRQHandler** Vector68
- #define **DMA1\_Ch1\_IRQHandler** Vector6C
- #define **DMA1\_Ch2\_IRQHandler** Vector70
- #define **DMA1\_Ch3\_IRQHandler** Vector74
- #define **DMA1\_Ch4\_IRQHandler** Vector78
- #define **DMA1\_Ch5\_IRQHandler** Vector7C
- #define **DMA1\_Ch6\_IRQHandler** Vector80
- #define **DMA1\_Ch7\_IRQHandler** Vector84
- #define **ADC1\_2\_IRQHandler** Vector88
- #define **EXTI9\_5\_IRQHandler** Vector9C
- #define **TIM1\_BRK\_IRQHandler** VectorA0
- #define **TIM1\_UP\_IRQHandler** VectorA4
- #define **TIM1\_TRG\_COM\_IRQHandler** VectorA8
- #define **TIM1\_CC\_IRQHandler** VectorAC
- #define **TIM2\_IRQHandler** VectorB0
- #define **TIM3\_IRQHandler** VectorB4
- #define **TIM4\_IRQHandler** VectorB8
- #define **I2C1\_EV\_IRQHandler** VectorBC
- #define **I2C1\_ER\_IRQHandler** VectorC0
- #define **I2C2\_EV\_IRQHandler** VectorC4
- #define **I2C2\_ER\_IRQHandler** VectorC8
- #define **SPI1\_IRQHandler** VectorCC
- #define **SPI2\_IRQHandler** VectorD0
- #define **USART1\_IRQHandler** VectorD4
- #define **USART2\_IRQHandler** VectorD8
- #define **USART3\_IRQHandler** VectorDC
- #define **EXTI15\_10\_IRQHandler** VectorE0
- #define **RTC\_Alarm\_IRQHandler** VectorE4
- #define **CEC\_IRQHandler** VectorE8
- #define **TIM12\_IRQHandler** VectorEC
- #define **TIM13\_IRQHandler** VectorF0
- #define **TIM14\_IRQHandler** VectorF4

## Configuration options

- `#define STM32_SW STM32_SW_PLL`  
*Main clock source selection.*
- `#define STM32_PLLSRC STM32_PLLSRC_HSE`  
*Clock source for the PLL.*
- `#define STM32_PLLXTPRE STM32_PLLXTPRE_DIV1`  
*Crystal PLL pre-divider.*
- `#define STM32_PLLMUL_VALUE 3`  
*PLL multiplier value.*
- `#define STM32_HPRE STM32_HPRE_DIV1`  
*AHB prescaler value.*
- `#define STM32_PPREG1 STM32_PPREG1_DIV1`  
*APB1 prescaler value.*
- `#define STM32_PPREG2 STM32_PPREG2_DIV1`  
*APB2 prescaler value.*
- `#define STM32_ADCPRE STM32_ADCPRE_DIV2`  
*ADC prescaler value.*
- `#define STM32_MCOSEL STM32_MCOSEL_NOCLOCK`  
*MCO pin setting.*
- `#define STM32_RTCSEL STM32_RTCSEL_LSI`  
*Clock source selecting. LSI by default.*

## Defines

- `#define STM32_ACTIVATE_PLL TRUE`  
*PLL activation flag.*
- `#define STM32_PLLMUL ((STM32_PLLMUL_VALUE - 2) << 18)`  
*PLLMUL field.*
- `#define STM32_PLLCLKIN (STM32_HSECLK / 1)`  
*PLL input clock frequency.*
- `#define STM32_PLLCLKOUT (STM32_PLLCLKIN * STM32_PLLMUL_VALUE)`  
*PLL output clock frequency.*
- `#define STM32_SYSCLK STM32_PLLCLKOUT`  
*System clock source.*
- `#define STM32_HCLK (STM32_SYSCLK / 1)`  
*AHB frequency.*
- `#define STM32_PCLK1 (STM32_HCLK / 1)`  
*APB1 frequency.*
- `#define STM32_PCLK2 (STM32_HCLK / 1)`  
*APB2 frequency.*
- `#define STM32_RTCCLK STM32_LSECLK`  
*RTC clock.*
- `#define STM32_ADCCLK (STM32_PCLK2 / 2)`  
*ADC frequency.*
- `#define STM32_TIMCLK1 (STM32_PCLK1 * 1)`  
*Timers 2, 3, 4, 5, 6, 7, 12, 13, 14 clock.*
- `#define STM32_TIMCLK2 (STM32_PCLK2 * 1)`  
*Timers 1, 8, 9, 10, 11 clock.*
- `#define STM32_FLASHBITS 0x00000010`  
*Flash settings.*

## 6.21.2 Define Documentation

6.21.2.1 `#define STM32_SYSCLK_MAX 24000000`

Maximum system clock frequency.

6.21.2.2 `#define STM32_HSECLK_MAX 24000000`

Maximum HSE clock frequency.

6.21.2.3 `#define STM32_HSECLK_MIN 1000000`

Minimum HSE clock frequency.

6.21.2.4 `#define STM32_LSECLK_MAX 1000000`

Maximum LSE clock frequency.

6.21.2.5 `#define STM32_LSECLK_MIN 32768`

Minimum LSE clock frequency.

6.21.2.6 `#define STM32_PLLIN_MAX 24000000`

Maximum PLLs input clock frequency.

6.21.2.7 `#define STM32_PLLIN_MIN 1000000`

Maximum PLLs input clock frequency.

6.21.2.8 `#define STM32_PLLOUT_MAX 24000000`

Maximum PLL output clock frequency.

6.21.2.9 `#define STM32_PLLOUT_MIN 16000000`

Maximum PLL output clock frequency.

6.21.2.10 `#define STM32_PCLK1_MAX 24000000`

Maximum APB1 clock frequency.

6.21.2.11 `#define STM32_PCLK2_MAX 24000000`

Maximum APB2 clock frequency.

6.21.2.12 `#define STM32_ADCCLK_MAX 12000000`

Maximum ADC clock frequency.

6.21.2.13 `#define STM32_SW_HSI (0 << 0)`

SYSCLK source is HSI.

6.21.2.14 `#define STM32_SW_HSE (1 << 0)`

SYSCLK source is HSE.

6.21.2.15 `#define STM32_SW_PLL (2 << 0)`

SYSCLK source is PLL.

6.21.2.16 `#define STM32_HPRE_DIV1 (0 << 4)`

SYSCLK divided by 1.

6.21.2.17 `#define STM32_HPRE_DIV2 (8 << 4)`

SYSCLK divided by 2.

6.21.2.18 `#define STM32_HPRE_DIV4 (9 << 4)`

SYSCLK divided by 4.

6.21.2.19 `#define STM32_HPRE_DIV8 (10 << 4)`

SYSCLK divided by 8.

6.21.2.20 `#define STM32_HPRE_DIV16 (11 << 4)`

SYSCLK divided by 16.

6.21.2.21 `#define STM32_HPRE_DIV64 (12 << 4)`

SYSCLK divided by 64.

6.21.2.22 `#define STM32_HPRE_DIV128 (13 << 4)`

SYSCLK divided by 128.

6.21.2.23 `#define STM32_HPRE_DIV256 (14 << 4)`

SYSCLK divided by 256.

6.21.2.24 `#define STM32_HPRE_DIV512 (15 << 4)`

SYSCLK divided by 512.

6.21.2.25 `#define STM32_PPREG1_DIV1 (0 << 8)`

HCLK divided by 1.

6.21.2.26 `#define STM32_PPREG1_DIV2 (4 << 8)`

HCLK divided by 2.

6.21.2.27 `#define STM32_PPREG1_DIV4 (5 << 8)`

HCLK divided by 4.

6.21.2.28 `#define STM32_PPREG1_DIV8 (6 << 8)`

HCLK divided by 8.

6.21.2.29 `#define STM32_PPREG1_DIV16 (7 << 8)`

HCLK divided by 16.

6.21.2.30 `#define STM32_PPREG2_DIV1 (0 << 11)`

HCLK divided by 1.

6.21.2.31 `#define STM32_PPREG2_DIV2 (4 << 11)`

HCLK divided by 2.

6.21.2.32 `#define STM32_PPREG2_DIV4 (5 << 11)`

HCLK divided by 4.

6.21.2.33 `#define STM32_PPREG2_DIV8 (6 << 11)`

HCLK divided by 8.

6.21.2.34 `#define STM32_PPREG2_DIV16 (7 << 11)`

HCLK divided by 16.

6.21.2.35 `#define STM32_ADCPRE_DIV2 (0 << 14)`

HCLK divided by 2.

6.21.2.36 `#define STM32_ADCPRE_DIV4 (1 << 14)`

HCLK divided by 4.

6.21.2.37 `#define STM32_ADCPRE_DIV6 (2 << 14)`

HCLK divided by 6.

6.21.2.38 `#define STM32_ADCPRE_DIV8 (3 << 14)`

HCLK divided by 8.

6.21.2.39 `#define STM32_PLLSRC_HSI (0 << 16)`

PLL clock source is HSI.

6.21.2.40 `#define STM32_PLLSRC_HSE (1 << 16)`

PLL clock source is HSE.

6.21.2.41 `#define STM32_PLLXTPRE_DIV1 (0 << 17)`

HSE divided by 1.

6.21.2.42 `#define STM32_PLLXTPRE_DIV2 (1 << 17)`

HSE divided by 2.

6.21.2.43 `#define STM32_MCOSEL_NOCLOCK (0 << 24)`

No clock on MCO pin.

6.21.2.44 `#define STM32_MCOSEL_SYSCLK (4 << 24)`

SYSCLK on MCO pin.

6.21.2.45 `#define STM32_MCOSEL_HSI (5 << 24)`

HSI clock on MCO pin.

6.21.2.46 `#define STM32_MCOSEL_HSE (6 << 24)`

HSE clock on MCO pin.

6.21.2.47 `#define STM32_MCOSEL_PLLDIV2 (7 << 24)`

PLL/2 clock on MCO pin.

6.21.2.48 `#define STM32_RTCSEL_MASK (3 << 8)`

RTC clock source mask.

6.21.2.49 #define STM32\_RTCSEL\_NOCLOCK (0 << 8)

No clock.

6.21.2.50 #define STM32\_RTCSEL\_LSE (1 << 8)

LSE used as RTC clock.

6.21.2.51 #define STM32\_RTCSEL\_LSI (2 << 8)

LSI used as RTC clock.

6.21.2.52 #define STM32\_RTCSEL\_HSEDIV (3 << 8)

HSE divided by 128 used as RTC clock.

6.21.2.53 #define WWDG\_IRQHandler Vector40

Window Watchdog.

6.21.2.54 #define PVD\_IRQHandler Vector44

PVD through EXTI Line detect.

6.21.2.55 #define TAMPER\_IRQHandler Vector48

Tamper.

6.21.2.56 #define RTC\_IRQHandler Vector4C

RTC.

6.21.2.57 #define FLASH\_IRQHandler Vector50

Flash.

6.21.2.58 #define RCC\_IRQHandler Vector54

RCC.

6.21.2.59 #define EXTI0\_IRQHandler Vector58

EXTI Line 0.

6.21.2.60 #define EXTI1\_IRQHandler Vector5C

EXTI Line 1.

6.21.2.61 #define EXTI2\_IRQHandler Vector60

EXTI Line 2.

6.21.2.62 #define EXTI3\_IRQHandler Vector64

EXTI Line 3.

6.21.2.63 #define EXTI4\_IRQHandler Vector68

EXTI Line 4.

6.21.2.64 #define DMA1\_Ch1\_IRQHandler Vector6C

DMA1 Channel 1.

6.21.2.65 #define DMA1\_Ch2\_IRQHandler Vector70

DMA1 Channel 2.

6.21.2.66 #define DMA1\_Ch3\_IRQHandler Vector74

DMA1 Channel 3.

6.21.2.67 #define DMA1\_Ch4\_IRQHandler Vector78

DMA1 Channel 4.

6.21.2.68 #define DMA1\_Ch5\_IRQHandler Vector7C

DMA1 Channel 5.

6.21.2.69 #define DMA1\_Ch6\_IRQHandler Vector80

DMA1 Channel 6.

6.21.2.70 #define DMA1\_Ch7\_IRQHandler Vector84

DMA1 Channel 7.

6.21.2.71 #define ADC1\_2\_IRQHandler Vector88

ADC1\_2.

6.21.2.72 #define EXTI9\_5\_IRQHandler Vector9C

EXTI Line 9..5.

6.21.2.73 #define TIM1\_BRK\_IRQHandler VectorA0

TIM1 Break.

6.21.2.74 #define TIM1\_UP\_IRQHandler VectorA4

TIM1 Update.

6.21.2.75 #define TIM1\_TRG\_COM\_IRQHandler VectorA8

TIM1 Trigger and Commutation.

6.21.2.76 #define TIM1\_CC\_IRQHandler VectorAC

TIM1 Capture Compare.

6.21.2.77 #define TIM2\_IRQHandler VectorB0

TIM2.

6.21.2.78 #define TIM3\_IRQHandler VectorB4

TIM3.

6.21.2.79 #define TIM4\_IRQHandler VectorB8

TIM4.

6.21.2.80 #define I2C1\_EV\_IRQHandler VectorBC

I2C1 Event.

6.21.2.81 #define I2C1\_ER\_IRQHandler VectorC0

I2C1 Error.

6.21.2.82 #define I2C2\_EV\_IRQHandler VectorC4

I2C2 Event.

6.21.2.83 #define I2C2\_ER\_IRQHandler VectorC8

I2C2 Error.

6.21.2.84 #define SPI1\_IRQHandler VectorCC

SPI1.

6.21.2.85 #define SPI2\_IRQHandler VectorD0

SPI2.

6.21.2.86 #define USART1\_IRQHandler VectorD4

USART1.

6.21.2.87 #define USART2\_IRQHandler VectorD8

USART2.

6.21.2.88 #define USART3\_IRQHandler VectorDC

USART3.

6.21.2.89 #define EXTI15\_10\_IRQHandler VectorE0

EXTI Line 15..10.

6.21.2.90 #define RTC\_Alarm\_IRQHandler VectorE4

RTC Alarm through EXTI.

6.21.2.91 #define CEC\_IRQHandler VectorE8

CEC.

6.21.2.92 #define TIM12\_IRQHandler VectorEC

TIM12.

6.21.2.93 #define TIM13\_IRQHandler VectorF0

TIM13.

6.21.2.94 #define TIM14\_IRQHandler VectorF4

TIM14.

6.21.2.95 #define STM32\_SW STM32\_SW\_PLL

Main clock source selection.

#### Note

If the selected clock source is not the PLL then the PLL is not initialized and started.

The default value is calculated for a 72MHz system clock from a 8MHz crystal using the PLL.

```
6.21.2.96 #define STM32_PLLSRC STM32_PLLSRC_HSE
```

Clock source for the PLL.

**Note**

This setting has only effect if the PLL is selected as the system clock source.

The default value is calculated for a 72MHz system clock from a 8MHz crystal using the PLL.

```
6.21.2.97 #define STM32_PLLXTPRE STM32_PLLXTPRE_DIV1
```

Crystal PLL pre-divider.

**Note**

This setting has only effect if the PLL is selected as the system clock source.

The default value is calculated for a 72MHz system clock from a 8MHz crystal using the PLL.

```
6.21.2.98 #define STM32_PLLMUL_VALUE 3
```

PLL multiplier value.

**Note**

The allowed range is 2...16.

The default value is calculated for a 24MHz system clock from a 8MHz crystal using the PLL.

```
6.21.2.99 #define STM32_HPRE STM32_HPRE_DIV1
```

AHB prescaler value.

**Note**

The default value is calculated for a 24MHz system clock from a 8MHz crystal using the PLL.

```
6.21.2.100 #define STM32_PPRE1 STM32_PPRE1_DIV1
```

APB1 prescaler value.

```
6.21.2.101 #define STM32_PPRE2 STM32_PPRE2_DIV1
```

APB2 prescaler value.

```
6.21.2.102 #define STM32_ADCPRE STM32_ADCPRE_DIV2
```

ADC prescaler value.

```
6.21.2.103 #define STM32_MCOSEL STM32_MCOSEL_NOCLOCK
```

MCO pin setting.

6.21.2.104 #define STM32\_RTCSEL STM32\_RTCSEL\_LSI

Clock source selecting. LSI by default.

6.21.2.105 #define STM32\_ACTIVATE\_PLL TRUE

PLL activation flag.

6.21.2.106 #define STM32\_PLLMUL ((STM32\_PLLMUL\_VALUE - 2) << 18)

PLLMUL field.

6.21.2.107 #define STM32\_PLLCLKIN (STM32\_HSECLK / 1)

PLL input clock frequency.

6.21.2.108 #define STM32\_PLLCLKOUT (STM32\_PLLCLKIN \* STM32\_PLLMUL\_VALUE)

PLL output clock frequency.

6.21.2.109 #define STM32\_SYSCLK STM32\_PLLCLKOUT

System clock source.

6.21.2.110 #define STM32\_HCLK (STM32\_SYSCLK / 1)

AHB frequency.

6.21.2.111 #define STM32\_PCLK1 (STM32\_HCLK / 1)

APB1 frequency.

6.21.2.112 #define STM32\_PCLK2 (STM32\_HCLK / 1)

APB2 frequency.

6.21.2.113 #define STM32\_RTCCLK STM32\_LSECLK

RTC clock.

6.21.2.114 #define STM32\_ADCCLK (STM32\_PCLK2 / 2)

ADC frequency.

6.21.2.115 #define STM32\_TIMCLK1 (STM32\_PCLK1 \* 1)

Timers 2, 3, 4, 5, 6, 7, 12, 13, 14 clock.

6.21.2.116 #define STM32\_TIMCLK2 (STM32\_PCLK2 \* 1)

Timers 1, 8, 9, 10, 11 clock.

6.21.2.117 #define STM32\_FLASHBITS 0x00000010

Flash settings.

## 6.22 STM32F103 HAL Support

### 6.22.1 Detailed Description

HAL support for STM32 Performance Line LD, MD and HD sub-families.

#### Platform identification

- #define PLATFORM\_NAME "STM32F1 Performance Line"

#### Absolute Maximum Ratings

- #define STM32\_SYSCLK\_MAX 72000000  
*Maximum system clock frequency.*
- #define STM32\_HSECLK\_MAX 25000000  
*Maximum HSE clock frequency.*
- #define STM32\_HSECLK\_MIN 1000000  
*Minimum HSE clock frequency.*
- #define STM32\_LSECLK\_MAX 1000000  
*Maximum LSE clock frequency.*
- #define STM32\_LSECLK\_MIN 32768  
*Minimum LSE clock frequency.*
- #define STM32\_PLLIN\_MAX 25000000  
*Maximum PLLs input clock frequency.*
- #define STM32\_PLLIN\_MIN 1000000  
*Maximum PLLs input clock frequency.*
- #define STM32\_PLLOUT\_MAX 72000000  
*Maximum PLL output clock frequency.*
- #define STM32\_PLLOUT\_MIN 16000000  
*Maximum PLL output clock frequency.*
- #define STM32\_PCLK1\_MAX 36000000  
*Maximum APB1 clock frequency.*
- #define STM32\_PCLK2\_MAX 72000000  
*Maximum APB2 clock frequency.*
- #define STM32\_ADCCLK\_MAX 14000000  
*Maximum ADC clock frequency.*

### RCC\_CFGR register bits definitions

- #define STM32\_SW\_HSI (0 << 0)
- #define STM32\_SW\_HSE (1 << 0)
- #define STM32\_SW\_PLL (2 << 0)
- #define STM32\_HPRE\_DIV1 (0 << 4)
- #define STM32\_HPRE\_DIV2 (8 << 4)
- #define STM32\_HPRE\_DIV4 (9 << 4)
- #define STM32\_HPRE\_DIV8 (10 << 4)
- #define STM32\_HPRE\_DIV16 (11 << 4)
- #define STM32\_HPRE\_DIV64 (12 << 4)
- #define STM32\_HPRE\_DIV128 (13 << 4)
- #define STM32\_HPRE\_DIV256 (14 << 4)
- #define STM32\_HPRE\_DIV512 (15 << 4)
- #define STM32\_PPREG1\_DIV1 (0 << 8)
- #define STM32\_PPREG1\_DIV2 (4 << 8)
- #define STM32\_PPREG1\_DIV4 (5 << 8)
- #define STM32\_PPREG1\_DIV8 (6 << 8)
- #define STM32\_PPREG1\_DIV16 (7 << 8)
- #define STM32\_PPREG2\_DIV1 (0 << 11)
- #define STM32\_PPREG2\_DIV2 (4 << 11)
- #define STM32\_PPREG2\_DIV4 (5 << 11)
- #define STM32\_PPREG2\_DIV8 (6 << 11)
- #define STM32\_PPREG2\_DIV16 (7 << 11)
- #define STM32\_ADCPRE\_DIV2 (0 << 14)
- #define STM32\_ADCPRE\_DIV4 (1 << 14)
- #define STM32\_ADCPRE\_DIV6 (2 << 14)
- #define STM32\_ADCPRE\_DIV8 (3 << 14)
- #define STM32\_PLLSRC\_HSI (0 << 16)
- #define STM32\_PLLSRC\_HSE (1 << 16)
- #define STM32\_PLLXTPRE\_DIV1 (0 << 17)
- #define STM32\_PLLXTPRE\_DIV2 (1 << 17)
- #define STM32\_USBPRE\_DIV1P5 (0 << 22)
- #define STM32\_USBPRE\_DIV1 (1 << 22)
- #define STM32\_MCOSEL\_NOCLOCK (0 << 24)
- #define STM32\_MCOSEL\_SYSCLK (4 << 24)
- #define STM32\_MCOSEL\_HSI (5 << 24)
- #define STM32\_MCOSEL\_HSE (6 << 24)
- #define STM32\_MCOSEL\_PLLDIV2 (7 << 24)
- #define STM32\_RTCSEL\_MASK (3 << 8)
- #define STM32\_RTCSEL\_NOCLOCK (0 << 8)
- #define STM32\_RTCSEL\_LSE (1 << 8)
- #define STM32\_RTCSEL\_LSI (2 << 8)
- #define STM32\_RTCSEL\_HSEDIV (3 << 8)

### STM32F103 XL capabilities

- #define STM32\_HAS\_ADC1 TRUE
- #define STM32\_HAS\_ADC1 TRUE
- #define STM32\_HAS\_ADC1 TRUE
- #define STM32\_HAS\_ADC1 TRUE
- #define STM32\_HAS\_ADC2 TRUE
- #define STM32\_HAS\_ADC2 TRUE
- #define STM32\_HAS\_ADC2 TRUE
- #define STM32\_HAS\_ADC2 TRUE

- #define **STM32\_HAS\_ADC3** FALSE
- #define **STM32\_HAS\_ADC3** FALSE
- #define **STM32\_HAS\_ADC3** TRUE
- #define **STM32\_HAS\_ADC3** TRUE
- #define **STM32\_HAS\_CAN1** TRUE
- #define **STM32\_HAS\_CAN2** FALSE
- #define **STM32\_HAS\_CAN2** FALSE
- #define **STM32\_HAS\_CAN2** FALSE
- #define **STM32\_HAS\_CAN2** FALSE
- #define **STM32\_HAS\_DAC** FALSE
- #define **STM32\_HAS\_DAC** FALSE
- #define **STM32\_HAS\_DAC** TRUE
- #define **STM32\_HAS\_DAC** TRUE
- #define **STM32\_ADVANCED\_DMA** FALSE
- #define **STM32\_ADVANCED\_DMA** FALSE
- #define **STM32\_ADVANCED\_DMA** FALSE
- #define **STM32\_ADVANCED\_DMA** FALSE
- #define **STM32\_HAS\_DMA1** TRUE
- #define **STM32\_HAS\_DMA1** TRUE
- #define **STM32\_HAS\_DMA1** TRUE
- #define **STM32\_HAS\_DMA1** TRUE
- #define **STM32\_HAS\_DMA2** FALSE
- #define **STM32\_HAS\_DMA2** FALSE
- #define **STM32\_HAS\_DMA2** TRUE
- #define **STM32\_HAS\_DMA2** TRUE
- #define **STM32\_HAS\_ETH** FALSE
- #define **STM32\_HAS\_ETH** FALSE
- #define **STM32\_HAS\_ETH** FALSE
- #define **STM32\_HAS\_ETH** FALSE
- #define **STM32\_EXTI\_NUM\_CHANNELS** 19
- #define **STM32\_EXTI\_NUM\_CHANNELS** 19
- #define **STM32\_EXTI\_NUM\_CHANNELS** 19
- #define **STM32\_EXTI\_NUM\_CHANNELS** 19
- #define **STM32\_HAS\_GPIOA** TRUE
- #define **STM32\_HAS\_GPIOA** TRUE
- #define **STM32\_HAS\_GPIOA** TRUE
- #define **STM32\_HAS\_GPIOA** TRUE
- #define **STM32\_HAS\_GPIOB** TRUE
- #define **STM32\_HAS\_GPIOB** TRUE
- #define **STM32\_HAS\_GPIOB** TRUE
- #define **STM32\_HAS\_GPIOB** TRUE
- #define **STM32\_HAS\_GPIOC** TRUE
- #define **STM32\_HAS\_GPIOC** TRUE
- #define **STM32\_HAS\_GPIOC** TRUE
- #define **STM32\_HAS\_GPIOC** TRUE
- #define **STM32\_HAS\_GPIOD** TRUE
- #define **STM32\_HAS\_GPIOD** TRUE
- #define **STM32\_HAS\_GPIOD** TRUE
- #define **STM32\_HAS\_GPIOD** TRUE
- #define **STM32\_HAS\_GPIOE** FALSE
- #define **STM32\_HAS\_GPIOE** TRUE
- #define **STM32\_HAS\_GPIOE** TRUE

- #define **STM32\_HAS\_GPIOE** TRUE
- #define **STM32\_HAS\_GPIOF** FALSE
- #define **STM32\_HAS\_GPIOF** FALSE
- #define **STM32\_HAS\_GPIOF** TRUE
- #define **STM32\_HAS\_GPIOF** TRUE
- #define **STM32\_HAS\_GPIOG** FALSE
- #define **STM32\_HAS\_GPIOG** FALSE
- #define **STM32\_HAS\_GPIOG** TRUE
- #define **STM32\_HAS\_GPIOG** TRUE
- #define **STM32\_HAS\_GPIOH** FALSE
- #define **STM32\_HAS\_GPIOH** FALSE
- #define **STM32\_HAS\_GPIOH** FALSE
- #define **STM32\_HAS\_GPIOH** FALSE
- #define **STM32\_HAS\_GPIOI** FALSE
- #define **STM32\_HAS\_GPIOI** FALSE
- #define **STM32\_HAS\_GPIOI** FALSE
- #define **STM32\_HAS\_I2C1** TRUE
- #define **STM32\_HAS\_I2C1** TRUE
- #define **STM32\_HAS\_I2C1** TRUE
- #define **STM32\_HAS\_I2C1** TRUE
- #define **STM32\_I2C1\_RX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 7))
- #define **STM32\_I2C1\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_I2C1\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_I2C1\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_I2C1\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_I2C1\_TX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 6))
- #define **STM32\_I2C1\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_I2C1\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_I2C1\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_I2C1\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_I2C2** FALSE
- #define **STM32\_HAS\_I2C2** TRUE
- #define **STM32\_HAS\_I2C2** TRUE
- #define **STM32\_HAS\_I2C2** TRUE
- #define **STM32\_I2C2\_RX\_DMA\_MSK** 0
- #define **STM32\_I2C2\_RX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 5))
- #define **STM32\_I2C2\_RX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 5))
- #define **STM32\_I2C2\_RX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 5))
- #define **STM32\_I2C2\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_I2C2\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_I2C2\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_I2C2\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_I2C2\_TX\_DMA\_MSK** 0
- #define **STM32\_I2C2\_TX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 4))
- #define **STM32\_I2C2\_TX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 4))
- #define **STM32\_I2C2\_TX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 4))
- #define **STM32\_I2C2\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_I2C2\_TX\_DMA\_CHN** 0x00000000



- #define **STM32\_SPI1\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_SPI1\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_SPI1\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_SPI2** FALSE
- #define **STM32\_HAS\_SPI2** TRUE
- #define **STM32\_HAS\_SPI2** TRUE
- #define **STM32\_HAS\_SPI2** TRUE
- #define **STM32\_SPI2\_RX\_DMA\_MSK** 0
- #define **STM32\_SPI2\_RX\_DMA\_MSK** STM32\_DMA\_STREAM\_ID\_MSK(1, 4)
- #define **STM32\_SPI2\_RX\_DMA\_MSK** STM32\_DMA\_STREAM\_ID\_MSK(1, 4)
- #define **STM32\_SPI2\_RX\_DMA\_MSK** STM32\_DMA\_STREAM\_ID\_MSK(1, 4)
- #define **STM32\_SPI2\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_SPI2\_TX\_DMA\_MSK** 0
- #define **STM32\_SPI2\_TX\_DMA\_MSK** STM32\_DMA\_STREAM\_ID\_MSK(1, 5)
- #define **STM32\_SPI2\_TX\_DMA\_MSK** STM32\_DMA\_STREAM\_ID\_MSK(1, 5)
- #define **STM32\_SPI2\_TX\_DMA\_MSK** STM32\_DMA\_STREAM\_ID\_MSK(1, 5)
- #define **STM32\_SPI2\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_SPI2\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_SPI2\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_SPI2\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_SPI3** FALSE
- #define **STM32\_HAS\_SPI3** FALSE
- #define **STM32\_HAS\_SPI3** TRUE
- #define **STM32\_HAS\_SPI3** TRUE
- #define **STM32\_HAS\_TIM1** TRUE
- #define **STM32\_HAS\_TIM1** TRUE
- #define **STM32\_HAS\_TIM1** TRUE
- #define **STM32\_HAS\_TIM1** TRUE
- #define **STM32\_HAS\_TIM2** TRUE
- #define **STM32\_HAS\_TIM2** TRUE
- #define **STM32\_HAS\_TIM2** TRUE
- #define **STM32\_HAS\_TIM2** TRUE
- #define **STM32\_HAS\_TIM3** TRUE
- #define **STM32\_HAS\_TIM4** FALSE
- #define **STM32\_HAS\_TIM4** TRUE
- #define **STM32\_HAS\_TIM4** TRUE
- #define **STM32\_HAS\_TIM4** TRUE
- #define **STM32\_HAS\_TIM5** FALSE
- #define **STM32\_HAS\_TIM5** FALSE
- #define **STM32\_HAS\_TIM5** TRUE
- #define **STM32\_HAS\_TIM5** TRUE
- #define **STM32\_HAS\_TIM6** FALSE
- #define **STM32\_HAS\_TIM6** FALSE
- #define **STM32\_HAS\_TIM6** TRUE
- #define **STM32\_HAS\_TIM6** TRUE
- #define **STM32\_HAS\_TIM7** FALSE
- #define **STM32\_HAS\_TIM7** FALSE
- #define **STM32\_HAS\_TIM7** TRUE
- #define **STM32\_HAS\_TIM7** TRUE

- #define **STM32\_HAS\_TIM8** FALSE
- #define **STM32\_HAS\_TIM8** FALSE
- #define **STM32\_HAS\_TIM8** TRUE
- #define **STM32\_HAS\_TIM8** TRUE
- #define **STM32\_HAS\_TIM9** FALSE
- #define **STM32\_HAS\_TIM9** FALSE
- #define **STM32\_HAS\_TIM9** TRUE
- #define **STM32\_HAS\_TIM9** FALSE
- #define **STM32\_HAS\_TIM10** FALSE
- #define **STM32\_HAS\_TIM10** FALSE
- #define **STM32\_HAS\_TIM10** TRUE
- #define **STM32\_HAS\_TIM10** FALSE
- #define **STM32\_HAS\_TIM11** FALSE
- #define **STM32\_HAS\_TIM11** FALSE
- #define **STM32\_HAS\_TIM11** TRUE
- #define **STM32\_HAS\_TIM11** FALSE
- #define **STM32\_HAS\_TIM12** FALSE
- #define **STM32\_HAS\_TIM12** FALSE
- #define **STM32\_HAS\_TIM12** TRUE
- #define **STM32\_HAS\_TIM12** FALSE
- #define **STM32\_HAS\_TIM13** FALSE
- #define **STM32\_HAS\_TIM13** FALSE
- #define **STM32\_HAS\_TIM13** TRUE
- #define **STM32\_HAS\_TIM13** FALSE
- #define **STM32\_HAS\_TIM14** FALSE
- #define **STM32\_HAS\_TIM14** FALSE
- #define **STM32\_HAS\_TIM14** TRUE
- #define **STM32\_HAS\_TIM14** FALSE
- #define **STM32\_HAS\_TIM15** FALSE
- #define **STM32\_HAS\_TIM15** FALSE
- #define **STM32\_HAS\_TIM15** FALSE
- #define **STM32\_HAS\_TIM15** FALSE
- #define **STM32\_HAS\_TIM16** FALSE
- #define **STM32\_HAS\_TIM16** FALSE
- #define **STM32\_HAS\_TIM17** FALSE
- #define **STM32\_HAS\_TIM17** FALSE
- #define **STM32\_HAS\_USART1** TRUE
- #define **STM32\_HAS\_USART1** TRUE
- #define **STM32\_HAS\_USART1** TRUE
- #define **STM32\_HAS\_USART1** TRUE
- #define **STM32\_USART1\_RX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 5))
- #define **STM32\_USART1\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART1\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART1\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART1\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART1\_TX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 4))
- #define **STM32\_USART1\_TX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 4))
- #define **STM32\_USART1\_TX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 4))

- #define **STM32\_USART1\_TX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 4))
- #define **STM32\_USART1\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART1\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART1\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART1\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_USART2** TRUE
- #define **STM32\_HAS\_USART2** TRUE
- #define **STM32\_HAS\_USART2** TRUE
- #define **STM32\_HAS\_USART2** TRUE
- #define **STM32\_USART2\_RX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 6))
- #define **STM32\_USART2\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART2\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART2\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART2\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART2\_TX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 7))
- #define **STM32\_USART2\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART2\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART2\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART2\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART3\_RX\_DMA\_MSK** 0
- #define **STM32\_USART3\_RX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 3))
- #define **STM32\_USART3\_RX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 3))
- #define **STM32\_USART3\_RX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 3))
- #define **STM32\_USART3\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART3\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART3\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART3\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART3\_TX\_DMA\_MSK** 0
- #define **STM32\_USART3\_TX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 2))
- #define **STM32\_USART3\_TX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 2))
- #define **STM32\_USART3\_TX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 2))
- #define **STM32\_USART3\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART3\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART3\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART3\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_UART4** FALSE
- #define **STM32\_HAS\_UART4** FALSE
- #define **STM32\_HAS\_UART4** TRUE
- #define **STM32\_HAS\_UART4** TRUE
- #define **STM32\_UART4\_RX\_DMA\_MSK** 0
- #define **STM32\_UART4\_RX\_DMA\_MSK** 0
- #define **STM32\_UART4\_RX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(2, 3))
- #define **STM32\_UART4\_RX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(2, 3))
- #define **STM32\_UART4\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_UART4\_RX\_DMA\_CHN** 0x00000000

- #define **STM32\_UART4\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_UART4\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_UART4\_TX\_DMA\_MSK** 0
- #define **STM32\_UART4\_TX\_DMA\_MSK** 0
- #define **STM32\_UART4\_TX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(2, 5))
- #define **STM32\_UART4\_TX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(2, 5))
- #define **STM32\_UART4\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_UART4\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_UART4\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_UART4\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_UART5** FALSE
- #define **STM32\_HAS\_UART5** FALSE
- #define **STM32\_HAS\_UART5** TRUE
- #define **STM32\_HAS\_UART5** TRUE
- #define **STM32\_UART5\_RX\_DMA\_MSK** 0
- #define **STM32\_UART5\_RX\_DMA\_MSK** 0
- #define **STM32\_UART5\_RX\_DMA\_MSK** 0
- #define **STM32\_UART5\_RX\_DMA\_MSK** 0
- #define **STM32\_UART5\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_UART5\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_UART5\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_UART5\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_UART5\_TX\_DMA\_MSK** 0
- #define **STM32\_UART5\_TX\_DMA\_MSK** 0
- #define **STM32\_UART5\_TX\_DMA\_MSK** 0
- #define **STM32\_UART5\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_UART5\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_UART5\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_UART5\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_USART6** FALSE
- #define **STM32\_HAS\_USART6** FALSE
- #define **STM32\_HAS\_USART6** FALSE
- #define **STM32\_HAS\_USART6** FALSE
- #define **STM32\_USART6\_RX\_DMA\_MSK** 0
- #define **STM32\_USART6\_RX\_DMA\_MSK** 0
- #define **STM32\_USART6\_RX\_DMA\_MSK** 0
- #define **STM32\_USART6\_RX\_DMA\_MSK** 0
- #define **STM32\_USART6\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART6\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART6\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART6\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART6\_TX\_DMA\_MSK** 0
- #define **STM32\_USART6\_TX\_DMA\_MSK** 0
- #define **STM32\_USART6\_TX\_DMA\_MSK** 0
- #define **STM32\_USART6\_TX\_DMA\_MSK** 0
- #define **STM32\_USART6\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART6\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART6\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART6\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_USB** FALSE
- #define **STM32\_HAS\_USB** TRUE
- #define **STM32\_HAS\_USB** TRUE
- #define **STM32\_HAS\_USB** TRUE
- #define **STM32\_HAS\_OTG1** FALSE

- #define STM32\_HAS\_OTG1 FALSE
- #define STM32\_HAS\_OTG1 FALSE
- #define STM32\_HAS\_OTG1 FALSE
- #define STM32\_HAS\_OTG2 FALSE
- #define STM32\_HAS\_OTG2 FALSE
- #define STM32\_HAS\_OTG2 FALSE
- #define STM32\_I2C3\_RX\_DMA\_MSK 0
- #define STM32\_I2C3\_RX\_DMA\_MSK 0
- #define STM32\_I2C3\_RX\_DMA\_MSK 0
- #define STM32\_I2C3\_RX\_DMA\_CHN 0x00000000
- #define STM32\_I2C3\_RX\_DMA\_CHN 0x00000000
- #define STM32\_I2C3\_RX\_DMA\_CHN 0x00000000
- #define STM32\_I2C3\_TX\_DMA\_MSK 0
- #define STM32\_I2C3\_TX\_DMA\_MSK 0
- #define STM32\_I2C3\_TX\_DMA\_MSK 0
- #define STM32\_I2C3\_TX\_DMA\_CHN 0x00000000
- #define STM32\_I2C3\_TX\_DMA\_CHN 0x00000000
- #define STM32\_I2C3\_TX\_DMA\_CHN 0x00000000

## IRQ VECTOR names

- #define WWDG\_IRQHandler Vector40
- #define PVD\_IRQHandler Vector44
- #define TAMPER\_IRQHandler Vector48
- #define RTC\_IRQHandler Vector4C
- #define FLASH\_IRQHandler Vector50
- #define RCC\_IRQHandler Vector54
- #define EXTI0\_IRQHandler Vector58
- #define EXTI1\_IRQHandler Vector5C
- #define EXTI2\_IRQHandler Vector60
- #define EXTI3\_IRQHandler Vector64
- #define EXTI4\_IRQHandler Vector68
- #define DMA1\_Ch1\_IRQHandler Vector6C
- #define DMA1\_Ch2\_IRQHandler Vector70
- #define DMA1\_Ch3\_IRQHandler Vector74
- #define DMA1\_Ch4\_IRQHandler Vector78
- #define DMA1\_Ch5\_IRQHandler Vector7C
- #define DMA1\_Ch6\_IRQHandler Vector80
- #define DMA1\_Ch7\_IRQHandler Vector84
- #define ADC1\_2\_IRQHandler Vector88
- #define CAN1\_TX\_IRQHandler Vector8C
- #define USB\_HP\_IRQHandler Vector8C
- #define CAN1\_RX0\_IRQHandler Vector90
- #define USB\_LP\_IRQHandler Vector90
- #define CAN1\_RX1\_IRQHandler Vector94
- #define CAN1\_SCE\_IRQHandler Vector98
- #define EXTI9\_5\_IRQHandler Vector9C
- #define TIM1\_BRK\_IRQHandler VectorA0
- #define TIM1\_UP\_IRQHandler VectorA4
- #define TIM1\_TRG\_COM\_IRQHandler VectorA8
- #define TIM1\_CC\_IRQHandler VectorAC
- #define TIM2\_IRQHandler VectorB0
- #define TIM3\_IRQHandler VectorB4

- #define TIM4\_IRQHandler VectorB8
- #define I2C1\_EV\_IRQHandler VectorBC
- #define I2C1\_ER\_IRQHandler VectorC0
- #define I2C2\_EV\_IRQHandler VectorC4
- #define I2C2\_ER\_IRQHandler VectorC8
- #define SPI1\_IRQHandler VectorCC
- #define SPI2\_IRQHandler VectorD0
- #define USART1\_IRQHandler VectorD4
- #define USART2\_IRQHandler VectorD8
- #define USART3\_IRQHandler VectorDC
- #define EXTI15\_10\_IRQHandler VectorE0
- #define RTC\_Alarm\_IRQHandler VectorE4
- #define USB\_FS\_WKUP\_IRQHandler VectorE8
- #define TIM8\_BRK\_IRQHandler VectorEC
- #define TIM8\_UP\_IRQHandler VectorF0
- #define TIM8\_TRG\_COM\_IRQHandler VectorF4
- #define TIM8\_CC\_IRQHandler VectorF8
- #define ADC3\_IRQHandler VectorFC
- #define FSMC\_IRQHandler Vector100
- #define SDIO\_IRQHandler Vector104
- #define TIM5\_IRQHandler Vector108
- #define SPI3\_IRQHandler Vector10C
- #define UART4\_IRQHandler Vector110
- #define UART5\_IRQHandler Vector114
- #define TIM6\_IRQHandler Vector118
- #define TIM7\_IRQHandler Vector11C
- #define DMA2\_Ch1\_IRQHandler Vector120
- #define DMA2\_Ch2\_IRQHandler Vector124
- #define DMA2\_Ch3\_IRQHandler Vector128
- #define DMA2\_Ch4\_5\_IRQHandler Vector12C

## Configuration options

- #define STM32\_SW STM32\_SW\_PLL  
*Main clock source selection.*
- #define STM32\_PLLSRC STM32\_PLLSRC\_HSE  
*Clock source for the PLL.*
- #define STM32\_PLLXTPRE STM32\_PLLXTPRE\_DIV1  
*Crystal PLL pre-divider.*
- #define STM32\_PLLMUL\_VALUE 9  
*PLL multiplier value.*
- #define STM32\_HPRE STM32\_HPRE\_DIV1  
*AHB prescaler value.*
- #define STM32\_PPREG1 STM32\_PPREG1\_DIV2  
*APB1 prescaler value.*
- #define STM32\_PPREG2 STM32\_PPREG2\_DIV2  
*APB2 prescaler value.*
- #define STM32\_ADCPRE STM32\_ADCPRE\_DIV4  
*ADC prescaler value.*
- #define STM32\_USB\_CLOCK\_REQUIRED TRUE  
*USB clock setting.*
- #define STM32\_USBPREG STM32\_USBPREG\_DIV1P5  
*USB prescaler initialization.*

- `#define STM32_MCOSEL STM32_MCOSEL_NOCLOCK`  
*MCO pin setting.*
- `#define STM32_RTCSEL STM32_RTCSEL_LSI`  
*Clock source selecting. LSI by default.*

## Defines

- `#define STM32_ACTIVATE_PLL TRUE`  
*PLL activation flag.*
- `#define STM32_PLLMUL ((STM32_PLLMUL_VALUE - 2) << 18)`  
*PLLMUL field.*
- `#define STM32_PLLCLKIN (STM32_HSECLK / 1)`  
*PLL input clock frequency.*
- `#define STM32_PLLCLKOUT (STM32_PLLCLKIN * STM32_PLLMUL_VALUE)`  
*PLL output clock frequency.*
- `#define STM32_SYSCLK STM32_PLLCLKOUT`  
*System clock source.*
- `#define STM32_HCLK (STM32_SYSCLK / 1)`  
*AHB frequency.*
- `#define STM32_PCLK1 (STM32_HCLK / 1)`  
*APB1 frequency.*
- `#define STM32_PCLK2 (STM32_HCLK / 1)`  
*APB2 frequency.*
- `#define STM32_RTCCLK STM32_LSECLK`  
*RTC clock.*
- `#define STM32_ADCCLK (STM32_PCLK2 / 2)`  
*ADC frequency.*
- `#define STM32_USBCLK ((STM32_PLLCLKOUT * 2) / 3)`  
*USB frequency.*
- `#define STM32_TIMCLK1 (STM32_PCLK1 * 1)`  
*Timers 2, 3, 4, 5, 6, 7, 12, 13, 14 clock.*
- `#define STM32_TIMCLK2 (STM32_PCLK2 * 1)`  
*Timers 1, 8, 9, 10, 11 clock.*
- `#define STM32_FLASHBITS 0x00000010`  
*Flash settings.*

### 6.22.2 Define Documentation

#### 6.22.2.1 `#define STM32_SYSCLK_MAX 72000000`

Maximum system clock frequency.

#### 6.22.2.2 `#define STM32_HSECLK_MAX 25000000`

Maximum HSE clock frequency.

#### 6.22.2.3 `#define STM32_HSECLK_MIN 1000000`

Minimum HSE clock frequency.

6.22.2.4 `#define STM32_LSECLK_MAX 1000000`

Maximum LSE clock frequency.

6.22.2.5 `#define STM32_LSECLK_MIN 32768`

Minimum LSE clock frequency.

6.22.2.6 `#define STM32_PLLIN_MAX 25000000`

Maximum PLLs input clock frequency.

6.22.2.7 `#define STM32_PLLIN_MIN 1000000`

Maximum PLLs input clock frequency.

6.22.2.8 `#define STM32_PLLOUT_MAX 72000000`

Maximum PLL output clock frequency.

6.22.2.9 `#define STM32_PLLOUT_MIN 16000000`

Maximum PLL output clock frequency.

6.22.2.10 `#define STM32_PCLK1_MAX 36000000`

Maximum APB1 clock frequency.

6.22.2.11 `#define STM32_PCLK2_MAX 72000000`

Maximum APB2 clock frequency.

6.22.2.12 `#define STM32_ADCCLK_MAX 14000000`

Maximum ADC clock frequency.

6.22.2.13 `#define STM32_SW_HSI (0 << 0)`

SYSCLK source is HSI.

6.22.2.14 `#define STM32_SW_HSE (1 << 0)`

SYSCLK source is HSE.

6.22.2.15 `#define STM32_SW_PLL (2 << 0)`

SYSCLK source is PLL.

6.22.2.16 `#define STM32_HPRE_DIV1 (0 << 4)`

SYSCLK divided by 1.

6.22.2.17 `#define STM32_HPRE_DIV2 (8 << 4)`

SYSCLK divided by 2.

6.22.2.18 `#define STM32_HPRE_DIV4 (9 << 4)`

SYSCLK divided by 4.

6.22.2.19 `#define STM32_HPRE_DIV8 (10 << 4)`

SYSCLK divided by 8.

6.22.2.20 `#define STM32_HPRE_DIV16 (11 << 4)`

SYSCLK divided by 16.

6.22.2.21 `#define STM32_HPRE_DIV64 (12 << 4)`

SYSCLK divided by 64.

6.22.2.22 `#define STM32_HPRE_DIV128 (13 << 4)`

SYSCLK divided by 128.

6.22.2.23 `#define STM32_HPRE_DIV256 (14 << 4)`

SYSCLK divided by 256.

6.22.2.24 `#define STM32_HPRE_DIV512 (15 << 4)`

SYSCLK divided by 512.

6.22.2.25 `#define STM32_PPRE1_DIV1 (0 << 8)`

HCLK divided by 1.

6.22.2.26 `#define STM32_PPRE1_DIV2 (4 << 8)`

HCLK divided by 2.

6.22.2.27 `#define STM32_PPRE1_DIV4 (5 << 8)`

HCLK divided by 4.

6.22.2.28 #define STM32\_PPREG1\_DIV8 (6 << 8)

HCLK divided by 8.

6.22.2.29 #define STM32\_PPREG1\_DIV16 (7 << 8)

HCLK divided by 16.

6.22.2.30 #define STM32\_PPREG2\_DIV1 (0 << 11)

HCLK divided by 1.

6.22.2.31 #define STM32\_PPREG2\_DIV2 (4 << 11)

HCLK divided by 2.

6.22.2.32 #define STM32\_PPREG2\_DIV4 (5 << 11)

HCLK divided by 4.

6.22.2.33 #define STM32\_PPREG2\_DIV8 (6 << 11)

HCLK divided by 8.

6.22.2.34 #define STM32\_PPREG2\_DIV16 (7 << 11)

HCLK divided by 16.

6.22.2.35 #define STM32\_ADCPRE\_DIV2 (0 << 14)

HCLK divided by 2.

6.22.2.36 #define STM32\_ADCPRE\_DIV4 (1 << 14)

HCLK divided by 4.

6.22.2.37 #define STM32\_ADCPRE\_DIV6 (2 << 14)

HCLK divided by 6.

6.22.2.38 #define STM32\_ADCPRE\_DIV8 (3 << 14)

HCLK divided by 8.

6.22.2.39 #define STM32\_PLLSRC\_HSI (0 << 16)

PLL clock source is HSI.

6.22.2.40 `#define STM32_PLLSRC_HSE (1 << 16)`

PLL clock source is HSE.

6.22.2.41 `#define STM32_PLLXTPRE_DIV1 (0 << 17)`

HSE divided by 1.

6.22.2.42 `#define STM32_PLLXTPRE_DIV2 (1 << 17)`

HSE divided by 2.

6.22.2.43 `#define STM32_USBPRE_DIV1P5 (0 << 22)`

PLLOUT divided by 1.5.

6.22.2.44 `#define STM32_USBPRE_DIV1 (1 << 22)`

PLLOUT divided by 1.

6.22.2.45 `#define STM32_MCOSEL_NOCLOCK (0 << 24)`

No clock on MCO pin.

6.22.2.46 `#define STM32_MCOSEL_SYSCLK (4 << 24)`

SYSCLK on MCO pin.

6.22.2.47 `#define STM32_MCOSEL_HSI (5 << 24)`

HSI clock on MCO pin.

6.22.2.48 `#define STM32_MCOSEL_HSE (6 << 24)`

HSE clock on MCO pin.

6.22.2.49 `#define STM32_MCOSEL_PLLDIV2 (7 << 24)`

PLL/2 clock on MCO pin.

6.22.2.50 `#define STM32_RTCSEL_MASK (3 << 8)`

RTC clock source mask.

6.22.2.51 `#define STM32_RTCSEL_NOCLOCK (0 << 8)`

No clock.

6.22.2.52 #define STM32\_RTCSEL\_LSE (1 << 8)

LSE used as RTC clock.

6.22.2.53 #define STM32\_RTCSEL\_LSI (2 << 8)

LSI used as RTC clock.

6.22.2.54 #define STM32\_RTCSEL\_HSEDIV (3 << 8)

HSE divided by 128 used as RTC clock.

6.22.2.55 #define WWDG\_IRQHandler Vector40

Window Watchdog.

6.22.2.56 #define PVD\_IRQHandler Vector44

PVD through EXTI Line detect.

6.22.2.57 #define TAMPER\_IRQHandler Vector48

Tamper.

6.22.2.58 #define RTC\_IRQHandler Vector4C

RTC.

6.22.2.59 #define FLASH\_IRQHandler Vector50

Flash.

6.22.2.60 #define RCC\_IRQHandler Vector54

RCC.

6.22.2.61 #define EXTI0\_IRQHandler Vector58

EXTI Line 0.

6.22.2.62 #define EXTI1\_IRQHandler Vector5C

EXTI Line 1.

6.22.2.63 #define EXTI2\_IRQHandler Vector60

EXTI Line 2.

6.22.2.64 #define EXTI3\_IRQHandler Vector64

EXTI Line 3.

6.22.2.65 #define EXTI4\_IRQHandler Vector68

EXTI Line 4.

6.22.2.66 #define DMA1\_Ch1\_IRQHandler Vector6C

DMA1 Channel 1.

6.22.2.67 #define DMA1\_Ch2\_IRQHandler Vector70

DMA1 Channel 2.

6.22.2.68 #define DMA1\_Ch3\_IRQHandler Vector74

DMA1 Channel 3.

6.22.2.69 #define DMA1\_Ch4\_IRQHandler Vector78

DMA1 Channel 4.

6.22.2.70 #define DMA1\_Ch5\_IRQHandler Vector7C

DMA1 Channel 5.

6.22.2.71 #define DMA1\_Ch6\_IRQHandler Vector80

DMA1 Channel 6.

6.22.2.72 #define DMA1\_Ch7\_IRQHandler Vector84

DMA1 Channel 7.

6.22.2.73 #define ADC1\_2\_IRQHandler Vector88

ADC1\_2.

6.22.2.74 #define CAN1\_TX\_IRQHandler Vector8C

CAN1 TX.

6.22.2.75 #define USB\_HP\_IRQHandler Vector8C

USB High Priority, CAN1 TX.

6.22.2.76 #define CAN1\_RX0\_IRQHandler Vector90

CAN1 RX0.

6.22.2.77 #define USB\_LP\_IRQHandler Vector90

USB Low Priority, CAN1 RX0.

6.22.2.78 #define CAN1\_RX1\_IRQHandler Vector94

CAN1 RX1.

6.22.2.79 #define CAN1\_SCE\_IRQHandler Vector98

CAN1 SCE.

6.22.2.80 #define EXTI9\_5\_IRQHandler Vector9C

EXTI Line 9..5.

6.22.2.81 #define TIM1\_BRK\_IRQHandler VectorA0

TIM1 Break.

6.22.2.82 #define TIM1\_UP\_IRQHandler VectorA4

TIM1 Update.

6.22.2.83 #define TIM1\_TRG\_COM\_IRQHandler VectorA8

TIM1 Trigger and Commutation.

6.22.2.84 #define TIM1\_CC\_IRQHandler VectorAC

TIM1 Capture Compare.

6.22.2.85 #define TIM2\_IRQHandler VectorB0

TIM2.

6.22.2.86 #define TIM3\_IRQHandler VectorB4

TIM3.

6.22.2.87 #define TIM4\_IRQHandler VectorB8

TIM4.

6.22.2.88 #define I2C1\_EV\_IRQHandler VectorBC

I2C1 Event.

6.22.2.89 #define I2C1\_ER\_IRQHandler VectorC0

I2C1 Error.

6.22.2.90 #define I2C2\_EV\_IRQHandler VectorC4

I2C2 Event.

6.22.2.91 #define I2C2\_ER\_IRQHandler VectorC8

I2C2 Error.

6.22.2.92 #define SPI1\_IRQHandler VectorCC

SPI1.

6.22.2.93 #define SPI2\_IRQHandler VectorD0

SPI2.

6.22.2.94 #define USART1\_IRQHandler VectorD4

USART1.

6.22.2.95 #define USART2\_IRQHandler VectorD8

USART2.

6.22.2.96 #define USART3\_IRQHandler VectorDC

USART3.

6.22.2.97 #define EXTI15\_10\_IRQHandler VectorE0

EXTI Line 15..10.

6.22.2.98 #define RTC\_Alarm\_IRQHandler VectorE4

RTC Alarm through EXTI.

6.22.2.99 #define USB\_FS\_WKUP\_IRQHandler VectorE8

USB Wakeup from suspend.

6.22.2.100 #define TIM8\_BRK\_IRQHandler VectorEC

TIM8 Break.

6.22.2.101 #define TIM8\_UP\_IRQHandler VectorF0

TIM8 Update.

6.22.2.102 #define TIM8\_TRG\_COM\_IRQHandler VectorF4

TIM8 Trigger and Commutation.

6.22.2.103 #define TIM8\_CC\_IRQHandler VectorF8

TIM8 Capture Compare.

6.22.2.104 #define ADC3\_IRQHandler VectorFC

ADC3.

6.22.2.105 #define FSMC\_IRQHandler Vector100

FSMC.

6.22.2.106 #define SDIO\_IRQHandler Vector104

SDIO.

6.22.2.107 #define TIM5\_IRQHandler Vector108

TIM5.

6.22.2.108 #define SPI3\_IRQHandler Vector10C

SPI3.

6.22.2.109 #define UART4\_IRQHandler Vector110

UART4.

6.22.2.110 #define UART5\_IRQHandler Vector114

UART5.

6.22.2.111 #define TIM6\_IRQHandler Vector118

TIM6.

6.22.2.112 #define TIM7\_IRQHandler Vector11C

TIM7.

6.22.2.113 #define DMA2\_Ch1\_IRQHandler Vector120

DMA2 Channel1.

6.22.2.114 #define DMA2\_Ch2\_IRQHandler Vector124

DMA2 Channel2.

6.22.2.115 #define DMA2\_Ch3\_IRQHandler Vector128

DMA2 Channel3.

6.22.2.116 #define DMA2\_Ch4\_5\_IRQHandler Vector12C

DMA2 Channel4 & Channel5.

6.22.2.117 #define STM32\_SW STM32\_SW\_PLL

Main clock source selection.

#### Note

If the selected clock source is not the PLL then the PLL is not initialized and started.

The default value is calculated for a 72MHz system clock from a 8MHz crystal using the PLL.

6.22.2.118 #define STM32\_PLLSRC STM32\_PLLSRC\_HSE

Clock source for the PLL.

#### Note

This setting has only effect if the PLL is selected as the system clock source.

The default value is calculated for a 72MHz system clock from a 8MHz crystal using the PLL.

6.22.2.119 #define STM32\_PLLXTPRE STM32\_PLLXTPRE\_DIV1

Crystal PLL pre-divider.

#### Note

This setting has only effect if the PLL is selected as the system clock source.

The default value is calculated for a 72MHz system clock from a 8MHz crystal using the PLL.

6.22.2.120 #define STM32\_PLLMUL\_VALUE 9

PLL multiplier value.

**Note**

The allowed range is 2...16.

The default value is calculated for a 72MHz system clock from a 8MHz crystal using the PLL.

6.22.2.121 #define STM32\_HPRE STM32\_HPRE\_DIV1

AHB prescaler value.

**Note**

The default value is calculated for a 72MHz system clock from a 8MHz crystal using the PLL.

6.22.2.122 #define STM32\_PPREG STM32\_PPREG\_DIV2

APB1 prescaler value.

6.22.2.123 #define STM32\_PPREG2 STM32\_PPREG2\_DIV2

APB2 prescaler value.

6.22.2.124 #define STM32\_ADCPRE STM32\_ADCPRE\_DIV4

ADC prescaler value.

6.22.2.125 #define STM32\_USB\_CLOCK\_REQUIRED TRUE

USB clock setting.

6.22.2.126 #define STM32\_USBPREG STM32\_USBPREG\_DIV1P5

USB prescaler initialization.

6.22.2.127 #define STM32\_MCOSEL STM32\_MCOSEL\_NOCLOCK

MCO pin setting.

6.22.2.128 #define STM32\_RTCSEL STM32\_RTCSEL\_LSI

Clock source selecting. LSI by default.

6.22.2.129 #define STM32\_ACTIVATE\_PLL TRUE

PLL activation flag.

6.22.2.130 `#define STM32_PLLMUL ((STM32_PLLMUL_VALUE - 2) << 18)`

PLLMUL field.

6.22.2.131 `#define STM32_PLLCLKIN (STM32_HSECLK / 1)`

PLL input clock frequency.

6.22.2.132 `#define STM32_PLLCLKOUT (STM32_PLLCLKIN * STM32_PLLMUL_VALUE)`

PLL output clock frequency.

6.22.2.133 `#define STM32_SYSCLK STM32_PLLCLKOUT`

System clock source.

6.22.2.134 `#define STM32_HCLK (STM32_SYSCLK / 1)`

AHB frequency.

6.22.2.135 `#define STM32_PCLK1 (STM32_HCLK / 1)`

APB1 frequency.

6.22.2.136 `#define STM32_PCLK2 (STM32_HCLK / 1)`

APB2 frequency.

6.22.2.137 `#define STM32_RTCCLK STM32_LSECLK`

RTC clock.

6.22.2.138 `#define STM32_ADCCLK (STM32_PCLK2 / 2)`

ADC frequency.

6.22.2.139 `#define STM32_USBCLK ((STM32_PLLCLKOUT * 2) / 3)`

USB frequency.

6.22.2.140 `#define STM32_TIMCLK1 (STM32_PCLK1 * 1)`

Timers 2, 3, 4, 5, 6, 7, 12, 13, 14 clock.

6.22.2.141 `#define STM32_TIMCLK2 (STM32_PCLK2 * 1)`

Timers 1, 8, 9, 10, 11 clock.

6.22.2.142 #define STM32\_FLASHBITS 0x00000010

Flash settings.

## 6.23 STM32F105/F107 HAL Support

### 6.23.1 Detailed Description

HAL support for STM32 Connectivity Line sub-family.

#### Platform identification

- #define **PLATFORM\_NAME** "STM32F1 Connectivity Line"

#### Absolute Maximum Ratings

- #define **STM32\_SYSCLK\_MAX** 72000000  
*Maximum system clock frequency.*
- #define **STM32\_HSECLK\_MAX** 50000000  
*Maximum HSE clock frequency.*
- #define **STM32\_HSECLK\_MIN** 1000000  
*Minimum HSE clock frequency.*
- #define **STM32\_LSECLK\_MAX** 1000000  
*Maximum LSE clock frequency.*
- #define **STM32\_LSECLK\_MIN** 32768  
*Minimum LSE clock frequency.*
- #define **STM32\_PLL1IN\_MAX** 12000000  
*Maximum PLLs input clock frequency.*
- #define **STM32\_PLL1IN\_MIN** 3000000  
*Maximum PLL1 input clock frequency.*
- #define **STM32\_PLL23IN\_MAX** 5000000  
*Maximum PLL1 input clock frequency.*
- #define **STM32\_PLL23IN\_MIN** 3000000  
*Maximum PLL2 and PLL3 input clock frequency.*
- #define **STM32\_PLL1VCO\_MAX** 144000000  
*Maximum PLL1 VCO clock frequency.*
- #define **STM32\_PLL1VCO\_MIN** 360000000  
*Maximum PLL1 VCO clock frequency.*
- #define **STM32\_PLL23VCO\_MAX** 148000000  
*Maximum PLL2 and PLL3 VCO clock frequency.*
- #define **STM32\_PLL23VCO\_MIN** 80000000  
*Maximum PLL2 and PLL3 VCO clock frequency.*
- #define **STM32\_PCLK1\_MAX** 36000000  
*Maximum APB1 clock frequency.*
- #define **STM32\_PCLK2\_MAX** 72000000  
*Maximum APB2 clock frequency.*
- #define **STM32\_ADCCLK\_MAX** 14000000  
*Maximum ADC clock frequency.*
- #define **STM32\_SPII2S\_MAX** 18000000  
*Maximum SPI/I2S clock frequency.*

### RCC\_CFGR register bits definitions

- #define STM32\_SW\_HSI (0 << 0)
- #define STM32\_SW\_HSE (1 << 0)
- #define STM32\_SW\_PLL (2 << 0)
- #define STM32\_HPRE\_DIV1 (0 << 4)
- #define STM32\_HPRE\_DIV2 (8 << 4)
- #define STM32\_HPRE\_DIV4 (9 << 4)
- #define STM32\_HPRE\_DIV8 (10 << 4)
- #define STM32\_HPRE\_DIV16 (11 << 4)
- #define STM32\_HPRE\_DIV64 (12 << 4)
- #define STM32\_HPRE\_DIV128 (13 << 4)
- #define STM32\_HPRE\_DIV256 (14 << 4)
- #define STM32\_HPRE\_DIV512 (15 << 4)
- #define STM32\_PPREG1\_DIV1 (0 << 8)
- #define STM32\_PPREG1\_DIV2 (4 << 8)
- #define STM32\_PPREG1\_DIV4 (5 << 8)
- #define STM32\_PPREG1\_DIV8 (6 << 8)
- #define STM32\_PPREG1\_DIV16 (7 << 8)
- #define STM32\_PPREG2\_DIV1 (0 << 11)
- #define STM32\_PPREG2\_DIV2 (4 << 11)
- #define STM32\_PPREG2\_DIV4 (5 << 11)
- #define STM32\_PPREG2\_DIV8 (6 << 11)
- #define STM32\_PPREG2\_DIV16 (7 << 11)
- #define STM32\_ADCPRE\_DIV2 (0 << 14)
- #define STM32\_ADCPRE\_DIV4 (1 << 14)
- #define STM32\_ADCPRE\_DIV6 (2 << 14)
- #define STM32\_ADCPRE\_DIV8 (3 << 14)
- #define STM32\_PLLSRC\_HSI (0 << 16)
- #define STM32\_PLLSRC\_PREDIV1 (1 << 16)
- #define STM32\_OTGFSPRE\_DIV2 (1 << 22)
- #define STM32\_OTGFSPRE\_DIV3 (0 << 22)
- #define STM32\_MCOSEL\_NOCLOCK (0 << 24)
- #define STM32\_MCOSEL\_SYSCLK (4 << 24)
- #define STM32\_MCOSEL\_HSI (5 << 24)
- #define STM32\_MCOSEL\_HSE (6 << 24)
- #define STM32\_MCOSEL\_PLLDIV2 (7 << 24)
- #define STM32\_MCOSEL\_PLL2 (8 << 24)
- #define STM32\_MCOSEL\_PLL3DIV2 (9 << 24)
- #define STM32\_MCOSEL\_XT1 (10 << 24)
- #define STM32\_MCOSEL\_PLL3 (11 << 24)
- #define STM32\_RTCSEL\_MASK (3 << 8)
- #define STM32\_RTCSEL\_NOCLOCK (0 << 8)
- #define STM32\_RTCSEL\_LSE (1 << 8)
- #define STM32\_RTCSEL\_LSI (2 << 8)
- #define STM32\_RTCSEL\_HSEDIV (3 << 8)

### RCC\_CFGR2 register bits definitions

- #define STM32\_PREDIV1SRC\_HSE (0 << 16)
- #define STM32\_PREDIV1SRC\_PLL2 (1 << 16)

## STM32F105/F107 CL capabilities

- #define **STM32\_HAS\_ADC1** TRUE
- #define **STM32\_HAS\_ADC2** TRUE
- #define **STM32\_HAS\_ADC3** FALSE
- #define **STM32\_HAS\_CAN1** TRUE
- #define **STM32\_HAS\_CAN2** TRUE
- #define **STM32\_HAS\_DAC** TRUE
- #define **STM32\_ADVANCED\_DMA** FALSE
- #define **STM32\_HAS\_DMA1** TRUE
- #define **STM32\_HAS\_DMA2** TRUE
- #define **STM32\_HAS\_ETH** TRUE
- #define **STM32\_EXTI\_NUM\_CHANNELS** 20
- #define **STM32\_HAS\_GPIOA** TRUE
- #define **STM32\_HAS\_GPIOB** TRUE
- #define **STM32\_HAS\_GPIOC** TRUE
- #define **STM32\_HAS\_GPIOD** TRUE
- #define **STM32\_HAS\_GPIOE** TRUE
- #define **STM32\_HAS\_GPIOF** FALSE
- #define **STM32\_HAS\_GPIOG** FALSE
- #define **STM32\_HAS\_GPIOH** FALSE
- #define **STM32\_HAS\_GPIOI** FALSE
- #define **STM32\_HAS\_I2C1** TRUE
- #define **STM32\_I2C1\_RX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 7))
- #define **STM32\_I2C1\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_I2C1\_TX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 6))
- #define **STM32\_I2C1\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_I2C2** TRUE
- #define **STM32\_I2C2\_RX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 5))
- #define **STM32\_I2C2\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_I2C2\_TX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 4))
- #define **STM32\_I2C2\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_I2C3** FALSE
- #define **STM32\_I2C3\_RX\_DMA\_MSK** 0
- #define **STM32\_I2C3\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_I2C3\_TX\_DMA\_MSK** 0
- #define **STM32\_I2C3\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_RTC** TRUE
- #define **STM32\_RTCSEL\_HAS\_SUBSECONDS** TRUE
- #define **STM32\_HAS\_SDIO** FALSE
- #define **STM32\_HAS\_SPI1** TRUE
- #define **STM32\_SPI1\_RX\_DMA\_MSK** STM32\_DMA\_STREAM\_ID\_MSK(1, 2)
- #define **STM32\_SPI1\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_SPI1\_TX\_DMA\_MSK** STM32\_DMA\_STREAM\_ID\_MSK(1, 3)
- #define **STM32\_SPI1\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_SPI2** TRUE
- #define **STM32\_SPI2\_RX\_DMA\_MSK** STM32\_DMA\_STREAM\_ID\_MSK(1, 4)
- #define **STM32\_SPI2\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_SPI2\_TX\_DMA\_MSK** STM32\_DMA\_STREAM\_ID\_MSK(1, 5)
- #define **STM32\_SPI2\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_SPI3** TRUE
- #define **STM32\_SPI3\_RX\_DMA\_MSK** STM32\_DMA\_STREAM\_ID\_MSK(2, 1)
- #define **STM32\_SPI3\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_SPI3\_TX\_DMA\_MSK** STM32\_DMA\_STREAM\_ID\_MSK(2, 2)
- #define **STM32\_SPI3\_TX\_DMA\_CHN** 0x00000000

- #define **STM32\_HAS\_TIM1** TRUE
- #define **STM32\_HAS\_TIM2** TRUE
- #define **STM32\_HAS\_TIM3** TRUE
- #define **STM32\_HAS\_TIM4** TRUE
- #define **STM32\_HAS\_TIM5** TRUE
- #define **STM32\_HAS\_TIM6** TRUE
- #define **STM32\_HAS\_TIM7** TRUE
- #define **STM32\_HAS\_TIM8** FALSE
- #define **STM32\_HAS\_TIM9** FALSE
- #define **STM32\_HAS\_TIM10** FALSE
- #define **STM32\_HAS\_TIM11** FALSE
- #define **STM32\_HAS\_TIM12** FALSE
- #define **STM32\_HAS\_TIM13** FALSE
- #define **STM32\_HAS\_TIM14** FALSE
- #define **STM32\_HAS\_TIM15** FALSE
- #define **STM32\_HAS\_TIM16** FALSE
- #define **STM32\_HAS\_TIM17** FALSE
- #define **STM32\_HAS\_USART1** TRUE
- #define **STM32\_USART1\_RX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 5))
- #define **STM32\_USART1\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART1\_TX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 4))
- #define **STM32\_USART1\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_USART2** TRUE
- #define **STM32\_USART2\_RX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 6))
- #define **STM32\_USART2\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART2\_TX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 7))
- #define **STM32\_USART2\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_USART3** TRUE
- #define **STM32\_USART3\_RX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 3))
- #define **STM32\_USART3\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART3\_TX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 2))
- #define **STM32\_USART3\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_UART4** TRUE
- #define **STM32\_UART4\_RX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(2, 3))
- #define **STM32\_UART4\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_UART4\_TX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(2, 5))
- #define **STM32\_UART4\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_UART5** TRUE
- #define **STM32\_UART5\_RX\_DMA\_MSK** 0
- #define **STM32\_UART5\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_UART5\_TX\_DMA\_MSK** 0
- #define **STM32\_UART5\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_USART6** FALSE
- #define **STM32\_USART6\_RX\_DMA\_MSK** 0
- #define **STM32\_USART6\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART6\_TX\_DMA\_MSK** 0
- #define **STM32\_USART6\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_USB** FALSE
- #define **STM32\_HAS\_OTG1** TRUE
- #define **STM32\_HAS\_OTG2** FALSE

## IRQ VECTOR names

- #define WWDG\_IRQHandler Vector40
- #define PVD\_IRQHandler Vector44
- #define TAMPER\_IRQHandler Vector48
- #define RTC\_IRQHandler Vector4C
- #define FLASH\_IRQHandler Vector50
- #define RCC\_IRQHandler Vector54
- #define EXTI0\_IRQHandler Vector58
- #define EXTI1\_IRQHandler Vector5C
- #define EXTI2\_IRQHandler Vector60
- #define EXTI3\_IRQHandler Vector64
- #define EXTI4\_IRQHandler Vector68
- #define DMA1\_Ch1\_IRQHandler Vector6C
- #define DMA1\_Ch2\_IRQHandler Vector70
- #define DMA1\_Ch3\_IRQHandler Vector74
- #define DMA1\_Ch4\_IRQHandler Vector78
- #define DMA1\_Ch5\_IRQHandler Vector7C
- #define DMA1\_Ch6\_IRQHandler Vector80
- #define DMA1\_Ch7\_IRQHandler Vector84
- #define ADC1\_2\_IRQHandler Vector88
- #define CAN1\_TX\_IRQHandler Vector8C
- #define CAN1\_RX0\_IRQHandler Vector90
- #define CAN1\_RX1\_IRQHandler Vector94
- #define CAN1\_SCE\_IRQHandler Vector98
- #define EXTI9\_5\_IRQHandler Vector9C
- #define TIM1\_BRK\_IRQHandler VectorA0
- #define TIM1\_UP\_IRQHandler VectorA4
- #define TIM1\_TRG\_COM\_IRQHandler VectorA8
- #define TIM1\_CC\_IRQHandler VectorAC
- #define TIM2\_IRQHandler VectorB0
- #define TIM3\_IRQHandler VectorB4
- #define TIM4\_IRQHandler VectorB8
- #define I2C1\_EV\_IRQHandler VectorBC
- #define I2C1\_ER\_IRQHandler VectorC0
- #define I2C2\_EV\_IRQHandler VectorC4
- #define I2C2\_ER\_IRQHandler VectorC8
- #define SPI1\_IRQHandler VectorCC
- #define SPI2\_IRQHandler VectorD0
- #define USART1\_IRQHandler VectorD4
- #define USART2\_IRQHandler VectorD8
- #define USART3\_IRQHandler VectorDC
- #define EXTI15\_10\_IRQHandler VectorE0
- #define RTC\_Alarm\_IRQHandler VectorE4
- #define OTG\_FS\_WKUP\_IRQHandler VectorE8
- #define TIM5\_IRQHandler Vector108
- #define SPI3\_IRQHandler Vector10C
- #define UART4\_IRQHandler Vector110
- #define UART5\_IRQHandler Vector114
- #define TIM6\_IRQHandler Vector118
- #define TIM7\_IRQHandler Vector11C
- #define DMA2\_Ch1\_IRQHandler Vector120
- #define DMA2\_Ch2\_IRQHandler Vector124
- #define DMA2\_Ch3\_IRQHandler Vector128
- #define DMA2\_Ch4\_IRQHandler Vector12C

- #define DMA2\_Ch5\_IRQHandler Vector130
- #define ETH\_IRQHandler Vector134
- #define ETH\_WKUP\_IRQHandler Vector138
- #define CAN2\_TX\_IRQHandler Vector13C
- #define CAN2\_RX0\_IRQHandler Vector140
- #define CAN2\_RX1\_IRQHandler Vector144
- #define CAN2\_SCE\_IRQHandler Vector148
- #define OTG\_FS\_IRQHandler Vector14C

## Configuration options

- #define STM32\_SW STM32\_SW\_PLL  
*Main clock source selection.*
- #define STM32\_PLLSRC STM32\_PLLSRC\_PREDIV1  
*Clock source for the PLL.*
- #define STM32\_PREDIV1SRC STM32\_PREDIV1SRC\_HSE  
*PREDIV1 clock source.*
- #define STM32\_PREDIV1\_VALUE 5  
*PREDIV1 division factor.*
- #define STM32\_PLLMUL\_VALUE 9  
*PLL multiplier value.*
- #define STM32\_PREDIV2\_VALUE 5  
*PREDIV2 division factor.*
- #define STM32\_PLL2MUL\_VALUE 8  
*PLL2 multiplier value.*
- #define STM32\_PLL3MUL\_VALUE 10  
*PLL3 multiplier value.*
- #define STM32\_HPRE STM32\_HPRE\_DIV1  
*AHB prescaler value.*
- #define STM32\_PPREG1 STM32\_PPREG1\_DIV2  
*APB1 prescaler value.*
- #define STM32\_PPREG2 STM32\_PPREG2\_DIV2  
*APB2 prescaler value.*
- #define STM32\_ADCPRE STM32\_ADCPRE\_DIV4  
*ADC prescaler value.*
- #define STM32\_OTG\_CLOCK\_REQUIRED TRUE  
*USB clock setting.*
- #define STM32\_OTGFSPRE STM32\_OTGFSPRE\_DIV3  
*OTG prescaler initialization.*
- #define STM32\_I2S\_CLOCK\_REQUIRED FALSE  
*Dedicated I2S clock setting.*
- #define STM32\_MCOSEL STM32\_MCOSEL\_NOCLOCK  
*MCO pin setting.*
- #define STM32\_RTCSEL STM32\_RTCSEL\_HSEDIV  
*Clock source selecting. LSI by default.*

## Defines

- `#define STM32_ACTIVATE_PLL1 TRUE`  
*PLL1 activation flag.*
- `#define STM32_ACTIVATE_PLL2 TRUE`  
*PLL2 activation flag.*
- `#define STM32_ACTIVATE_PLL3 TRUE`  
*PLL3 activation flag.*
- `#define STM32_PREDIV1 ((STM32_PREDIV1_VALUE - 1) << 0)`  
*PREDIV1 field.*
- `#define STM32_PREDIV2 ((STM32_PREDIV2_VALUE - 1) << 4)`  
*PREDIV2 field.*
- `#define STM32_PLLMUL ((STM32_PLLMUL_VALUE - 2) << 18)`  
*PLLMUL field.*
- `#define STM32_PLL2MUL ((STM32_PLL2MUL_VALUE - 2) << 8)`  
*PLL2MUL field.*
- `#define STM32_PLL3MUL ((STM32_PLL3MUL_VALUE - 2) << 12)`  
*PLL3MUL field.*
- `#define STM32_PLL2CLKIN (STM32_HSECLK / STM32_PREDIV2_VALUE)`  
*PLL2 input frequency.*
- `#define STM32_PLL2CLKOUT (STM32_PLL2CLKIN * STM32_PLL2MUL_VALUE)`  
*PLL2 output clock frequency.*
- `#define STM32_PLL2VCO (STM32_PLL2CLKOUT * 2)`  
*PLL2 VCO clock frequency.*
- `#define STM32_PLL3CLKIN (STM32_HSECLK / STM32_PREDIV2_VALUE)`  
*PLL3 input frequency.*
- `#define STM32_PLL3CLKOUT (STM32_PLL3CLKIN * STM32_PLL3MUL_VALUE)`  
*PLL3 output clock frequency.*
- `#define STM32_PLL3VCO (STM32_PLL3CLKOUT * 2)`  
*PLL3 VCO clock frequency.*
- `#define STM32_PREDIV1CLK STM32_HSECLK`  
*PREDIV1 input frequency.*
- `#define STM32_PLLCLKIN (STM32_PREDIV1CLK / STM32_PREDIV1_VALUE)`  
*PLL input clock frequency.*
- `#define STM32_PLLCLKOUT (STM32_PLLCLKIN * STM32_PLLMUL_VALUE)`  
*PLL output clock frequency.*
- `#define STM32_PLLVCO (STM32_PLLCLKOUT * 2)`  
*PLL VCO clock frequency.*
- `#define STM32_SYSCLK STM32_PLLCLKOUT`  
*System clock source.*
- `#define STM32_HCLK (STM32_SYSCLK / 1)`  
*AHB frequency.*
- `#define STM32_PCLK1 (STM32_HCLK / 1)`  
*APB1 frequency.*
- `#define STM32_PCLK2 (STM32_HCLK / 1)`  
*APB2 frequency.*
- `#define STM32_RTCCLK STM32_LSECLK`  
*RTC clock.*
- `#define STM32_ADCCLK (STM32_PCLK2 / 2)`  
*ADC frequency.*
- `#define STM32_OTGFSCLK (STM32_PLLVCO / 3)`

- OTG frequency.*
- `#define STM32_TIMCLK1 (STM32_PCLK1 * 1)`  
*Timers 2, 3, 4, 5, 6, 7 clock.*
  - `#define STM32_TIMCLK2 (STM32_PCLK2 * 1)`  
*Timers 1, 8 clock.*
  - `#define STM32_FLASHBITS 0x00000010`  
*Flash settings.*

## 6.23.2 Define Documentation

6.23.2.1 `#define STM32_SYSCLK_MAX 72000000`

Maximum system clock frequency.

6.23.2.2 `#define STM32_HSECLK_MAX 50000000`

Maximum HSE clock frequency.

6.23.2.3 `#define STM32_HSECLK_MIN 1000000`

Minimum HSE clock frequency.

6.23.2.4 `#define STM32_LSECLK_MAX 1000000`

Maximum LSE clock frequency.

6.23.2.5 `#define STM32_LSECLK_MIN 32768`

Minimum LSE clock frequency.

6.23.2.6 `#define STM32_PLL1IN_MAX 12000000`

Maximum PLLs input clock frequency.

6.23.2.7 `#define STM32_PLL1IN_MIN 3000000`

Maximum PLL1 input clock frequency.

6.23.2.8 `#define STM32_PLL23IN_MAX 5000000`

Maximum PLL1 input clock frequency.

6.23.2.9 `#define STM32_PLL23IN_MIN 3000000`

Maximum PLL2 and PLL3 input clock frequency.

6.23.2.10 `#define STM32_PLL1VCO_MAX 144000000`

Maximum PLL1 VCO clock frequency.

6.23.2.11 #define STM32\_PLL1VCO\_MIN 36000000

Maximum PLL1 VCO clock frequency.

6.23.2.12 #define STM32\_PLL23VCO\_MAX 148000000

Maximum PLL2 and PLL3 VCO clock frequency.

6.23.2.13 #define STM32\_PLL23VCO\_MIN 80000000

Maximum PLL2 and PLL3 VCO clock frequency.

6.23.2.14 #define STM32\_PCLK1\_MAX 36000000

Maximum APB1 clock frequency.

6.23.2.15 #define STM32\_PCLK2\_MAX 72000000

Maximum APB2 clock frequency.

6.23.2.16 #define STM32\_ADCCLK\_MAX 14000000

Maximum ADC clock frequency.

6.23.2.17 #define STM32\_SPII2S\_MAX 18000000

Maximum SPI/I2S clock frequency.

6.23.2.18 #define STM32\_SW\_HSI (0 << 0)

SYSCLK source is HSI.

6.23.2.19 #define STM32\_SW\_HSE (1 << 0)

SYSCLK source is HSE.

6.23.2.20 #define STM32\_SW\_PLL (2 << 0)

SYSCLK source is PLL.

6.23.2.21 #define STM32\_HPRE\_DIV1 (0 << 4)

SYSCLK divided by 1.

6.23.2.22 #define STM32\_HPRE\_DIV2 (8 << 4)

SYSCLK divided by 2.

6.23.2.23 `#define STM32_HPRE_DIV4 (9 << 4)`

SYSCLK divided by 4.

6.23.2.24 `#define STM32_HPRE_DIV8 (10 << 4)`

SYSCLK divided by 8.

6.23.2.25 `#define STM32_HPRE_DIV16 (11 << 4)`

SYSCLK divided by 16.

6.23.2.26 `#define STM32_HPRE_DIV64 (12 << 4)`

SYSCLK divided by 64.

6.23.2.27 `#define STM32_HPRE_DIV128 (13 << 4)`

SYSCLK divided by 128.

6.23.2.28 `#define STM32_HPRE_DIV256 (14 << 4)`

SYSCLK divided by 256.

6.23.2.29 `#define STM32_HPRE_DIV512 (15 << 4)`

SYSCLK divided by 512.

6.23.2.30 `#define STM32_PPRE1_DIV1 (0 << 8)`

HCLK divided by 1.

6.23.2.31 `#define STM32_PPRE1_DIV2 (4 << 8)`

HCLK divided by 2.

6.23.2.32 `#define STM32_PPRE1_DIV4 (5 << 8)`

HCLK divided by 4.

6.23.2.33 `#define STM32_PPRE1_DIV8 (6 << 8)`

HCLK divided by 8.

6.23.2.34 `#define STM32_PPRE1_DIV16 (7 << 8)`

HCLK divided by 16.

6.23.2.35 `#define STM32_PPREG2_DIV1 (0 << 11)`

HCLK divided by 1.

6.23.2.36 `#define STM32_PPREG2_DIV2 (4 << 11)`

HCLK divided by 2.

6.23.2.37 `#define STM32_PPREG2_DIV4 (5 << 11)`

HCLK divided by 4.

6.23.2.38 `#define STM32_PPREG2_DIV8 (6 << 11)`

HCLK divided by 8.

6.23.2.39 `#define STM32_PPREG2_DIV16 (7 << 11)`

HCLK divided by 16.

6.23.2.40 `#define STM32_ADCPRE_DIV2 (0 << 14)`

HCLK divided by 2.

6.23.2.41 `#define STM32_ADCPRE_DIV4 (1 << 14)`

HCLK divided by 4.

6.23.2.42 `#define STM32_ADCPRE_DIV6 (2 << 14)`

HCLK divided by 6.

6.23.2.43 `#define STM32_ADCPRE_DIV8 (3 << 14)`

HCLK divided by 8.

6.23.2.44 `#define STM32_PLLSRC_HSI (0 << 16)`

PLL clock source is HSI.

6.23.2.45 `#define STM32_PLLSRC_PREDIV1 (1 << 16)`

PLL clock source is PREDIV1.

6.23.2.46 `#define STM32_OTGFSPRE_DIV2 (1 << 22)`

HCLK\*2 divided by 2.

6.23.2.47 `#define STM32_OTGFSPRE_DIV3 (0 << 22)`

HCLK\*2 divided by 3.

6.23.2.48 `#define STM32_MCOSEL_NOCLOCK (0 << 24)`

No clock on MCO pin.

6.23.2.49 `#define STM32_MCOSEL_SYSCLK (4 << 24)`

SYSCLK on MCO pin.

6.23.2.50 `#define STM32_MCOSEL_HSI (5 << 24)`

HSI clock on MCO pin.

6.23.2.51 `#define STM32_MCOSEL_HSE (6 << 24)`

HSE clock on MCO pin.

6.23.2.52 `#define STM32_MCOSEL_PLLDIV2 (7 << 24)`

PLL/2 clock on MCO pin.

6.23.2.53 `#define STM32_MCOSEL_PLL2 (8 << 24)`

PLL2 clock on MCO pin.

6.23.2.54 `#define STM32_MCOSEL_PLL3DIV2 (9 << 24)`

PLL3/2 clock on MCO pin.

6.23.2.55 `#define STM32_MCOSEL_XT1 (10 << 24)`

XT1 clock on MCO pin.

6.23.2.56 `#define STM32_MCOSEL_PLL3 (11 << 24)`

PLL3 clock on MCO pin.

6.23.2.57 `#define STM32_RTCSEL_MASK (3 << 8)`

RTC clock source mask.

6.23.2.58 `#define STM32_RTCSEL_NOCLOCK (0 << 8)`

No clock.

6.23.2.59 #define STM32\_RTCSEL\_LSE (1 << 8)

LSE used as RTC clock.

6.23.2.60 #define STM32\_RTCSEL\_LSI (2 << 8)

LSI used as RTC clock.

6.23.2.61 #define STM32\_RTCSEL\_HSEDIV (3 << 8)

HSE divided by 128 used as RTC clock.

6.23.2.62 #define STM32\_PREDIV1SRC\_HSE (0 << 16)

PREDIV1 source is HSE.

6.23.2.63 #define STM32\_PREDIV1SRC\_PLL2 (1 << 16)

PREDIV1 source is PLL2.

6.23.2.64 #define WWDG\_IRQHandler Vector40

Window Watchdog.

6.23.2.65 #define PVD\_IRQHandler Vector44

PVD through EXTI Line detect.

6.23.2.66 #define TAMPER\_IRQHandler Vector48

Tamper.

6.23.2.67 #define RTC\_IRQHandler Vector4C

RTC.

6.23.2.68 #define FLASH\_IRQHandler Vector50

Flash.

6.23.2.69 #define RCC\_IRQHandler Vector54

RCC.

6.23.2.70 #define EXTI0\_IRQHandler Vector58

EXTI Line 0.

6.23.2.71 #define EXTI1\_IRQHandler Vector5C

EXTI Line 1.

6.23.2.72 #define EXTI2\_IRQHandler Vector60

EXTI Line 2.

6.23.2.73 #define EXTI3\_IRQHandler Vector64

EXTI Line 3.

6.23.2.74 #define EXTI4\_IRQHandler Vector68

EXTI Line 4.

6.23.2.75 #define DMA1\_Ch1\_IRQHandler Vector6C

DMA1 Channel 1.

6.23.2.76 #define DMA1\_Ch2\_IRQHandler Vector70

DMA1 Channel 2.

6.23.2.77 #define DMA1\_Ch3\_IRQHandler Vector74

DMA1 Channel 3.

6.23.2.78 #define DMA1\_Ch4\_IRQHandler Vector78

DMA1 Channel 4.

6.23.2.79 #define DMA1\_Ch5\_IRQHandler Vector7C

DMA1 Channel 5.

6.23.2.80 #define DMA1\_Ch6\_IRQHandler Vector80

DMA1 Channel 6.

6.23.2.81 #define DMA1\_Ch7\_IRQHandler Vector84

DMA1 Channel 7.

6.23.2.82 #define ADC1\_2\_IRQHandler Vector88

ADC1 and ADC2.

6.23.2.83 #define CAN1\_TX\_IRQHandler Vector8C

CAN1 TX.

6.23.2.84 #define CAN1\_RX0\_IRQHandler Vector90

CAN1 RX0.

6.23.2.85 #define CAN1\_RX1\_IRQHandler Vector94

CAN1 RX1.

6.23.2.86 #define CAN1\_SCE\_IRQHandler Vector98

CAN1 SCE.

6.23.2.87 #define EXTI9\_5\_IRQHandler Vector9C

EXTI Line 9..5.

6.23.2.88 #define TIM1\_BRK\_IRQHandler VectorA0

TIM1 Break.

6.23.2.89 #define TIM1\_UP\_IRQHandler VectorA4

TIM1 Update.

6.23.2.90 #define TIM1\_TRG\_COM\_IRQHandler VectorA8

TIM1 Trigger and Commutation.

6.23.2.91 #define TIM1\_CC\_IRQHandler VectorAC

TIM1 Capture Compare.

6.23.2.92 #define TIM2\_IRQHandler VectorB0

TIM2.

6.23.2.93 #define TIM3\_IRQHandler VectorB4

TIM3.

6.23.2.94 #define TIM4\_IRQHandler VectorB8

TIM4.

6.23.2.95 #define I2C1\_EV\_IRQHandler VectorBC

I2C1 Event.

6.23.2.96 #define I2C1\_ER\_IRQHandler VectorC0

I2C1 Error.

6.23.2.97 #define I2C2\_EV\_IRQHandler VectorC4

I2C2 Event.

6.23.2.98 #define I2C2\_ER\_IRQHandler VectorC8

I2C1 Error.

6.23.2.99 #define SPI1\_IRQHandler VectorCC

SPI1.

6.23.2.100 #define SPI2\_IRQHandler VectorD0

SPI2.

6.23.2.101 #define USART1\_IRQHandler VectorD4

USART1.

6.23.2.102 #define USART2\_IRQHandler VectorD8

USART2.

6.23.2.103 #define USART3\_IRQHandler VectorDC

USART3.

6.23.2.104 #define EXTI15\_10\_IRQHandler VectorE0

EXTI Line 15..10.

6.23.2.105 #define RTC\_Alarm\_IRQHandler VectorE4

RTC alarm through EXTI line.

6.23.2.106 #define OTG\_FS\_WKUP\_IRQHandler VectorE8

USB OTG FS Wakeup through EXTI line.

6.23.2.107 #define TIM5\_IRQHandler Vector108

TIM5.

6.23.2.108 #define SPI3\_IRQHandler Vector10C

SPI3.

6.23.2.109 #define UART4\_IRQHandler Vector110

UART4.

6.23.2.110 #define UART5\_IRQHandler Vector114

UART5.

6.23.2.111 #define TIM6\_IRQHandler Vector118

TIM6.

6.23.2.112 #define TIM7\_IRQHandler Vector11C

TIM7.

6.23.2.113 #define DMA2\_Ch1\_IRQHandler Vector120

DMA2 Channel1.

6.23.2.114 #define DMA2\_Ch2\_IRQHandler Vector124

DMA2 Channel2.

6.23.2.115 #define DMA2\_Ch3\_IRQHandler Vector128

DMA2 Channel3.

6.23.2.116 #define DMA2\_Ch4\_IRQHandler Vector12C

DMA2 Channel4.

6.23.2.117 #define DMA2\_Ch5\_IRQHandler Vector130

DMA2 Channel5.

6.23.2.118 #define ETH\_IRQHandler Vector134

Ethernet.

6.23.2.119 #define ETH\_WKUP\_IRQHandler Vector138

Ethernet Wakeup through EXTI line.

6.23.2.120 #define CAN2\_TX\_IRQHandler Vector13C

CAN2 TX.

6.23.2.121 #define CAN2\_RX0\_IRQHandler Vector140

CAN2 RX0.

6.23.2.122 #define CAN2\_RX1\_IRQHandler Vector144

CAN2 RX1.

6.23.2.123 #define CAN2\_SCE\_IRQHandler Vector148

CAN2 SCE.

6.23.2.124 #define OTG\_FS\_IRQHandler Vector14C

USB OTG FS.

6.23.2.125 #define STM32\_SW STM32\_SW\_PLL

Main clock source selection.

#### Note

The default value is calculated for a 72MHz system clock from a 25MHz crystal using both PLL and PLL2.

6.23.2.126 #define STM32\_PLLSRC STM32\_PLLSRC\_PREDIV1

Clock source for the PLL.

#### Note

The default value is calculated for a 72MHz system clock from a 25MHz crystal using both PLL and PLL2.

6.23.2.127 #define STM32\_PREDIV1SRC STM32\_PREDIV1SRC\_HSE

PREDIV1 clock source.

#### Note

The default value is calculated for a 72MHz system clock from a 25MHz crystal using both PLL and PLL2.

6.23.2.128 #define STM32\_PREDIV1\_VALUE 5

PREDIV1 division factor.

**Note**

The allowed range is 1...16.

The default value is calculated for a 72MHz system clock from a 25MHz crystal using both PLL and PLL2.

6.23.2.129 #define STM32\_PLLMUL\_VALUE 9

PLL multiplier value.

**Note**

The allowed range is 4...9.

The default value is calculated for a 72MHz system clock from a 25MHz crystal using both PLL and PLL2.

6.23.2.130 #define STM32\_PREDIV2\_VALUE 5

PREDIV2 division factor.

**Note**

The allowed range is 1...16.

The default value is calculated for a 72MHz system clock from a 25MHz crystal using both PLL and PLL2.

6.23.2.131 #define STM32\_PLL2MUL\_VALUE 8

PLL2 multiplier value.

**Note**

The default value is calculated for a 72MHz system clock from a 25MHz crystal using both PLL and PLL2.

6.23.2.132 #define STM32\_PLL3MUL\_VALUE 10

PLL3 multiplier value.

**Note**

The default value is calculated for a 50MHz clock from a 25MHz crystal.

6.23.2.133 #define STM32\_HPRE STM32\_HPRE\_DIV1

AHB prescaler value.

**Note**

The default value is calculated for a 72MHz system clock from a 25MHz crystal using both PLL and PLL2.

6.23.2.134 #define STM32\_PPREG STM32\_PPREG\_DIV2

APB1 prescaler value.

6.23.2.135 #define STM32\_PPREG STM32\_PPREG\_DIV2

APB2 prescaler value.

6.23.2.136 #define STM32\_ADCPRE STM32\_ADCPRE\_DIV4

ADC prescaler value.

6.23.2.137 #define STM32\_OTG\_CLOCK\_REQUIRED TRUE

USB clock setting.

6.23.2.138 #define STM32\_OTGFSPRE STM32\_OTGFSPRE\_DIV3

OTG prescaler initialization.

6.23.2.139 #define STM32\_I2S\_CLOCK\_REQUIRED FALSE

Dedicated I2S clock setting.

6.23.2.140 #define STM32\_MCOSEL STM32\_MCOSEL\_NOCLOCK

MCO pin setting.

6.23.2.141 #define STM32\_RTCSEL STM32\_RTCSEL\_HSEDIV

Clock source selecting. LSI by default.

6.23.2.142 #define STM32\_ACTIVATE\_PLL1 TRUE

PLL1 activation flag.

6.23.2.143 #define STM32\_ACTIVATE\_PLL2 TRUE

PLL2 activation flag.

6.23.2.144 #define STM32\_ACTIVATE\_PLL3 TRUE

PLL3 activation flag.

6.23.2.145 #define STM32\_PREDIV1 ((STM32\_PREDIV1\_VALUE - 1) << 0)

PREDIV1 field.

6.23.2.146 #define STM32\_PREDIV2 ((STM32\_PREDIV2\_VALUE - 1) << 4)

PREDIV2 field.

6.23.2.147 #define STM32\_PLLMUL ((STM32\_PLLMUL\_VALUE - 2) << 18)

PLLMUL field.

6.23.2.148 #define STM32\_PLL2MUL ((STM32\_PLL2MUL\_VALUE - 2) << 8)

PLL2MUL field.

6.23.2.149 #define STM32\_PLL3MUL ((STM32\_PLL3MUL\_VALUE - 2) << 12)

PLL3MUL field.

6.23.2.150 #define STM32\_PLL2CLKIN (STM32\_HSECLK / STM32\_PREDIV2\_VALUE)

PLL2 input frequency.

6.23.2.151 #define STM32\_PLL2CLKOUT (STM32\_PLL2CLKIN \* STM32\_PLL2MUL\_VALUE)

PLL2 output clock frequency.

6.23.2.152 #define STM32\_PLL2VCO (STM32\_PLL2CLKOUT \* 2)

PLL2 VCO clock frequency.

6.23.2.153 #define STM32\_PLL3CLKIN (STM32\_HSECLK / STM32\_PREDIV2\_VALUE)

PLL3 input frequency.

6.23.2.154 #define STM32\_PLL3CLKOUT (STM32\_PLL3CLKIN \* STM32\_PLL3MUL\_VALUE)

PLL3 output clock frequency.

6.23.2.155 #define STM32\_PLL3VCO (STM32\_PLL3CLKOUT \* 2)

PLL3 VCO clock frequency.

6.23.2.156 #define STM32\_PREDIV1CLK STM32\_HSECLK

PREDIV1 input frequency.

6.23.2.157 #define STM32\_PLLCLKIN (STM32\_PREDIV1CLK / STM32\_PREDIV1\_VALUE)

PLL input clock frequency.

6.23.2.158 #define STM32\_PLLCLKOUT (STM32\_PLLCLKIN \* STM32\_PLLMUL\_VALUE)

PLL output clock frequency.

6.23.2.159 #define STM32\_PLLVCO (STM32\_PLLCLKOUT \* 2)

PLL VCO clock frequency.

6.23.2.160 #define STM32\_SYSCLK STM32\_PLLCLKOUT

System clock source.

6.23.2.161 #define STM32\_HCLK (STM32\_SYSCLK / 1)

AHB frequency.

6.23.2.162 #define STM32\_PCLK1 (STM32\_HCLK / 1)

APB1 frequency.

6.23.2.163 #define STM32\_PCLK2 (STM32\_HCLK / 1)

APB2 frequency.

6.23.2.164 #define STM32\_RTCCLK STM32\_LSECLK

RTC clock.

6.23.2.165 #define STM32\_ADCCLK (STM32\_PCLK2 / 2)

ADC frequency.

6.23.2.166 #define STM32\_OTGFSCLK (STM32\_PLLVCO / 3)

OTG frequency.

6.23.2.167 #define STM32\_TIMCLK1 (STM32\_PCLK1 \* 1)

Timers 2, 3, 4, 5, 6, 7 clock.

6.23.2.168 #define STM32\_TIMCLK2 (STM32\_PCLK2 \* 1)

Timers 1, 8 clock.

6.23.2.169 #define STM32\_FLASHBITS 0x00000010

Flash settings.

## 6.24 STM32F1xx Drivers

### 6.24.1 Detailed Description

This section describes all the supported drivers on the STM32F1xx platform and the implementation details of the single drivers.

#### Modules

- [STM32F1xx Initialization Support](#)
- [STM32F1xx ADC Support](#)
- [STM32F1xx CAN Support](#)
- [STM32F1xx EXT Support](#)
- [STM32F1xx GPT Support](#)
- [STM32F1xx I2C Support](#)
- [STM32F1xx ICU Support](#)
- [STM32F1xx MAC Support](#)
- [STM32F1xx PAL Support](#)
- [STM32F1xx PWM Support](#)
- [STM32F1xx RTC Support](#)
- [STM32F1xx SDC Support](#)
- [STM32F1xx Serial Support](#)
- [STM32F1xx SPI Support](#)
- [STM32F1xx UART Support](#)
- [STM32F1xx USB Support](#)
- [STM32F1xx Platform Drivers](#)

## 6.25 STM32F1xx Initialization Support

The STM32F1xx HAL support is responsible for system initialization.

### 6.25.1 Supported HW resources

- PLL1.
- PLL2 (where present).
- RCC.
- Flash.

### 6.25.2 STM32F1xx HAL driver implementation features

- PLLs startup and stabilization.
- Clock tree initialization.
- Clock source selection.
- Flash wait states initialization based on the selected clock options.
- SYSTICK initialization based on current clock and kernel required rate.
- DMA support initialization.

## 6.26 STM32F1xx ADC Support

The STM32F1xx ADC driver supports the ADC peripherals using DMA channels for maximum performance.

### 6.26.1 Supported HW resources

- ADC1.
- DMA1.

### 6.26.2 STM32F1xx ADC driver implementation features

- Clock stop for reduced power usage when the driver is in stop state.
- Streaming conversion using DMA for maximum performance.
- Programmable ADC interrupt priority level.
- Programmable DMA bus priority for each DMA channel.
- Programmable DMA interrupt priority for each DMA channel.
- DMA errors detection.

## 6.27 STM32F1xx CAN Support

The STM32F1xx CAN driver uses the CAN peripherals.

### 6.27.1 Supported HW resources

- bxCAN1.

### 6.27.2 STM32F1xx CAN driver implementation features

- Clock stop for reduced power usage when the driver is in stop state.
- Support for bxCAN sleep mode.
- Programmable bxCAN interrupts priority level.

## 6.28 STM32F1xx EXT Support

The STM32F1xx EXT driver uses the EXTI peripheral.

### 6.28.1 Supported HW resources

- EXTI.

### 6.28.2 STM32F1xx EXT driver implementation features

- Each EXTI channel can be independently enabled and programmed.
- Programmable EXTI interrupts priority level.
- Capability to work as event sources (WFE) rather than interrupt sources.

## 6.29 STM32F1xx GPT Support

The STM32F1xx GPT driver uses the TIMx peripherals.

### 6.29.1 Supported HW resources

- TIM1.
- TIM2.
- TIM3.
- TIM4.
- TIM5.

### 6.29.2 STM32F1xx GPT driver implementation features

- Each timer can be independently enabled and programmed. Unused peripherals are left in low power mode.
- Programmable TIMx interrupts priority level.

## 6.30 STM32F1xx I2C Support

The STM32F1xx I2C driver uses the I2Cx peripherals.

### 6.30.1 Supported HW resources

- I2C1.
- I2C2.

### 6.30.2 STM32F1xx I2C driver implementation features

- Each I2C port can be independently enabled and programmed. Unused peripherals are left in low power mode.
- Programmable I2Cx interrupts priority level.

## 6.31 STM32F1xx ICU Support

The STM32F1xx ICU driver uses the TIMx peripherals.

### 6.31.1 Supported HW resources

- TIM1.
- TIM2.
- TIM3.
- TIM4.
- TIM5.

### 6.31.2 STM32F1xx ICU driver implementation features

- Each timer can be independently enabled and programmed. Unused peripherals are left in low power mode.
- Programmable TIMx interrupts priority level.

## 6.32 STM32F1xx MAC Support

The STM32 MAC driver supports the ETH peripheral.

### 6.32.1 Supported HW resources

- ETH.

## 6.33 STM32F1xx PAL Support

The STM32F1xx PAL driver uses the GPIO peripherals.

### 6.33.1 Supported HW resources

- AFIO.
- GPIOA.
- GPIOB.
- GPIOC.
- GPIOD.
- GPIOE (where present).
- GPIOF (where present).
- GPIOG (where present).

### 6.33.2 STM32F1xx PAL driver implementation features

The PAL driver implementation fully supports the following hardware capabilities:

- 16 bits wide ports.
- Atomic set/reset functions.

- Atomic set+reset function (atomic bus operations).
- Output latched regardless of the pad setting.
- Direct read of input pads regardless of the pad setting.

### 6.33.3 Supported PAL setup modes

The STM32F1xx PAL driver supports the following I/O modes:

- PAL\_MODE\_RESET.
- PAL\_MODE\_UNCONNECTED.
- PAL\_MODE\_INPUT.
- PAL\_MODE\_INPUT\_PULLUP.
- PAL\_MODE\_INPUT\_PULLDOWN.
- PAL\_MODE\_INPUT\_ANALOG.
- PAL\_MODE\_OUTPUT\_PUSH\_PULL.
- PAL\_MODE\_OUTPUT\_OPENDRAIN.
- PAL\_MODE\_STM32F1xx\_ALTERNATE\_PUSH\_PULL (non standard).
- PAL\_MODE\_STM32F1xx\_ALTERNATE\_OPENDRAIN (non standard).

Any attempt to setup an invalid mode is ignored.

### 6.33.4 Suboptimal behavior

The STM32F1xx GPIO is less than optimal in several areas, the limitations should be taken in account while using the PAL driver:

- Pad/port toggling operations are not atomic.
- Pad/group mode setup is not atomic.
- Writing on pads/groups/ports programmed as input with pull-up/down resistor can change the resistor setting because the output latch is used for resistor selection.

## 6.34 STM32F1xx PWM Support

The STM32F1xx PWM driver uses the TIMx peripherals.

### 6.34.1 Supported HW resources

- TIM1.
- TIM2.
- TIM3.
- TIM4.
- TIM5.

### 6.34.2 STM32F1xx PWM driver implementation features

- Each timer can be independently enabled and programmed. Unused peripherals are left in low power mode.
- Four independent PWM channels per timer.
- Programmable TIMx interrupts priority level.

## 6.35 STM32F1xx RTC Support

The STM32F1xx RTC driver uses the RTC peripheral.

### 6.35.1 Supported HW resources

- RTC.

## 6.36 STM32F1xx SDC Support

The STM32F1xx SDC driver uses the SDIO peripheral.

### 6.36.1 Supported HW resources

- SDIO.
- DMA2.

### 6.36.2 STM32F1xx SDC driver implementation features

- Clock stop for reduced power usage when the driver is in stop state.
- Programmable interrupt priority.
- DMA is used for receiving and transmitting.
- Programmable DMA bus priority for each DMA channel.

## 6.37 STM32F1xx Serial Support

The STM32F1xx Serial driver uses the USART/UART peripherals in a buffered, interrupt driven, implementation.

### 6.37.1 Supported HW resources

The serial driver can support any of the following hardware resources:

- USART1.
- USART2.
- USART3 (where present).
- UART4 (where present).
- UART5 (where present).

### 6.37.2 STM32F1xx Serial driver implementation features

- Clock stop for reduced power usage when the driver is in stop state.
- Each UART/USART can be independently enabled and programmed. Unused peripherals are left in low power mode.
- Fully interrupt driven.
- Programmable priority levels for each UART/USART.

## 6.38 STM32F1xx SPI Support

The SPI driver supports the STM32F1xx SPI peripherals using DMA channels for maximum performance.

### 6.38.1 Supported HW resources

- SPI1.
- SPI2.
- SPI3 (where present).
- DMA1.
- DMA2 (where present).

### 6.38.2 STM32F1xx SPI driver implementation features

- Clock stop for reduced power usage when the driver is in stop state.
- Each SPI can be independently enabled and programmed. Unused peripherals are left in low power mode.
- Programmable interrupt priority levels for each SPI.
- DMA is used for receiving and transmitting.
- Programmable DMA bus priority for each DMA channel.
- Programmable DMA interrupt priority for each DMA channel.
- Programmable DMA error hook.

## 6.39 STM32F1xx USART Support

The USART driver supports the STM32F1xx USART peripherals using DMA channels for maximum performance.

### 6.39.1 Supported HW resources

The USART driver can support any of the following hardware resources:

- USART1.
- USART2.
- USART3 (where present).
- USART4 (where present).
- DMA1.
- DMA2 (where present).

### 6.39.2 STM32F1xx UART driver implementation features

- Clock stop for reduced power usage when the driver is in stop state.
- Each UART/USART can be independently enabled and programmed. Unused peripherals are left in low power mode.
- Programmable interrupt priority levels for each UART/USART.
- DMA is used for receiving and transmitting.
- Programmable DMA bus priority for each DMA channel.
- Programmable DMA interrupt priority for each DMA channel.
- Programmable DMA error hook.

## 6.40 STM32F1xx USB Support

The USB driver supports the STM32F1xx USB peripheral.

### 6.40.1 Supported HW resources

The USB driver can support any of the following hardware resources:

- USB.

### 6.40.2 STM32F1xx USB driver implementation features

- Clock stop for reduced power usage when the driver is in stop state.
- Programmable interrupt priority levels.
- Each endpoint programmable in Control, Bulk and Interrupt modes.

## 6.41 STM32F1xx Platform Drivers

### 6.41.1 Detailed Description

Platform support drivers. Platform drivers do not implement HAL standard driver templates, their role is to support platform specific functionalities.

#### Modules

- [STM32F1xx DMA Support](#)
- [STM32F1xx RCC Support](#)

## 6.42 STM32F1xx DMA Support

### 6.42.1 Detailed Description

This DMA helper driver is used by the other drivers in order to access the shared DMA resources in a consistent way.

### 6.42.2 Supported HW resources

The DMA driver can support any of the following hardware resources:

- DMA1.
- DMA2 (where present).

### 6.42.3 STM32F1xx DMA driver implementation features

- Exports helper functions/macros to the other drivers that share the DMA resource.
- Automatic DMA clock stop when not in use by any driver.
- DMA streams and interrupt vectors sharing among multiple drivers.

DMA sharing helper driver. In the STM32 the DMA streams are a shared resource, this driver allows to allocate and free DMA streams at runtime in order to allow all the other device drivers to coordinate the access to the resource.

#### Note

The DMA ISR handlers are all declared into this module because sharing, the various device drivers can associate a callback to IRSs when allocating streams.

### Data Structures

- struct `stm32_dma_stream_t`  
*STM32 DMA stream descriptor structure.*

### Functions

- `CH_IRQ_HANDLER (DMA1_Ch1_IRQHandler)`  
*DMA1 stream 1 shared interrupt handler.*
- `CH_IRQ_HANDLER (DMA1_Ch2_IRQHandler)`  
*DMA1 stream 2 shared interrupt handler.*
- `CH_IRQ_HANDLER (DMA1_Ch3_IRQHandler)`  
*DMA1 stream 3 shared interrupt handler.*
- `CH_IRQ_HANDLER (DMA1_Ch4_IRQHandler)`  
*DMA1 stream 4 shared interrupt handler.*
- `CH_IRQ_HANDLER (DMA1_Ch5_IRQHandler)`  
*DMA1 stream 5 shared interrupt handler.*
- `CH_IRQ_HANDLER (DMA1_Ch6_IRQHandler)`  
*DMA1 stream 6 shared interrupt handler.*
- `CH_IRQ_HANDLER (DMA1_Ch7_IRQHandler)`  
*DMA1 stream 7 shared interrupt handler.*
- `CH_IRQ_HANDLER (DMA2_Ch1_IRQHandler)`  
*DMA2 stream 1 shared interrupt handler.*
- `CH_IRQ_HANDLER (DMA2_Ch2_IRQHandler)`  
*DMA2 stream 2 shared interrupt handler.*
- `CH_IRQ_HANDLER (DMA2_Ch3_IRQHandler)`  
*DMA2 stream 3 shared interrupt handler.*
- `CH_IRQ_HANDLER (DMA2_Ch4_IRQHandler)`  
*DMA2 stream 4 shared interrupt handler.*
- `CH_IRQ_HANDLER (DMA2_Ch5_IRQHandler)`

- DMA2 stream 5 shared interrupt handler.
- void **dmalnit** (void)
  - STM32 DMA helper initialization.*
- bool\_t **dmaStreamAllocate** (const **stm32\_dma\_stream\_t** \*dmastp, uint32\_t priority, **stm32\_dmaisr\_t** func, void \*param)
  - Allocates a DMA stream.*
- void **dmaStreamRelease** (const **stm32\_dma\_stream\_t** \*dmastp)
  - Releases a DMA stream.*

## Variables

- const **stm32\_dma\_stream\_t \_stm32\_dma\_streams** [STM32\_DMA\_STREAMS]
  - DMA streams descriptors.*

## DMA streams identifiers

- #define **STM32\_DMA\_STREAM\_ID**(dma, stream) (((dma) - 1) \* 7) + ((stream) - 1)
  - Returns an unique numeric identifier for a DMA stream.*
- #define **STM32\_DMA\_STREAM**(id) (&\_stm32\_dma\_streams[id])
  - Returns a pointer to a **stm32\_dma\_stream\_t** structure.*
- #define **STM32\_DMA1\_STREAM1** STM32\_DMA\_STREAM(0)
- #define **STM32\_DMA1\_STREAM2** STM32\_DMA\_STREAM(1)
- #define **STM32\_DMA1\_STREAM3** STM32\_DMA\_STREAM(2)
- #define **STM32\_DMA1\_STREAM4** STM32\_DMA\_STREAM(3)
- #define **STM32\_DMA1\_STREAM5** STM32\_DMA\_STREAM(4)
- #define **STM32\_DMA1\_STREAM6** STM32\_DMA\_STREAM(5)
- #define **STM32\_DMA1\_STREAM7** STM32\_DMA\_STREAM(6)
- #define **STM32\_DMA2\_STREAM1** STM32\_DMA\_STREAM(7)
- #define **STM32\_DMA2\_STREAM2** STM32\_DMA\_STREAM(8)
- #define **STM32\_DMA2\_STREAM3** STM32\_DMA\_STREAM(9)
- #define **STM32\_DMA2\_STREAM4** STM32\_DMA\_STREAM(10)
- #define **STM32\_DMA2\_STREAM5** STM32\_DMA\_STREAM(11)

## CR register constants common to all DMA types

- #define **STM32\_DMA\_CR\_EN** DMA\_CCR1\_EN
- #define **STM32\_DMA\_CR\_TEIE** DMA\_CCR1\_TEIE
- #define **STM32\_DMA\_CR\_HTIE** DMA\_CCR1\_HTIE
- #define **STM32\_DMA\_CR\_TCIE** DMA\_CCR1\_TCIE
- #define **STM32\_DMA\_CR\_DIR\_MASK** (DMA\_CCR1\_DIR | DMA\_CCR1\_MEM2MEM)
- #define **STM32\_DMA\_CR\_DIR\_P2M** 0
- #define **STM32\_DMA\_CR\_DIR\_M2P** DMA\_CCR1\_DIR
- #define **STM32\_DMA\_CR\_DIR\_M2M** DMA\_CCR1\_MEM2MEM
- #define **STM32\_DMA\_CR\_CIRC** DMA\_CCR1\_CIRC
- #define **STM32\_DMA\_CR\_PINC** DMA\_CCR1\_PINC
- #define **STM32\_DMA\_CR\_MINC** DMA\_CCR1\_MINC
- #define **STM32\_DMA\_CR\_PSIZE\_MASK** DMA\_CCR1\_PSIZE
- #define **STM32\_DMA\_CR\_PSIZE\_BYTE** 0
- #define **STM32\_DMA\_CR\_PSIZE\_HWORD** DMA\_CCR1\_PSIZE\_0
- #define **STM32\_DMA\_CR\_PSIZE\_WORD** DMA\_CCR1\_PSIZE\_1
- #define **STM32\_DMA\_CR\_MSIZE\_MASK** DMA\_CCR1\_MSIZE
- #define **STM32\_DMA\_CR\_MSIZE\_BYTE** 0

- #define **STM32\_DMA\_CR\_MSIZEx\_HWWORD** DMA\_CCR1\_MSIZE\_0
- #define **STM32\_DMA\_CR\_MSIZEx\_WORD** DMA\_CCR1\_MSIZE\_1
- #define **STM32\_DMA\_CR\_SIZE\_MASK**
- #define **STM32\_DMA\_CR\_PL\_MASK** DMA\_CCR1\_PL
- #define **STM32\_DMA\_CR\_PL(n)** ((n) << 12)

### CR register constants only found in enhanced DMA

- #define **STM32\_DMA\_CR\_DMEIE** 0  
*Ignored by normal DMA.*
- #define **STM32\_DMA\_CR\_CHSEL\_MASK** 0  
*Ignored by normal DMA.*
- #define **STM32\_DMA\_CR\_CHSEL(n)** 0  
*Ignored by normal DMA.*

### Status flags passed to the ISR callbacks

- #define **STM32\_DMA\_ISR\_FEIF** 0
- #define **STM32\_DMA\_ISR\_DMEIF** 0
- #define **STM32\_DMA\_ISR\_TEIF** DMA\_ISR\_TEIF1
- #define **STM32\_DMA\_ISR\_HTIF** DMA\_ISR\_HTIF1
- #define **STM32\_DMA\_ISR\_TCIF** DMA\_ISR\_TCIF1

### Macro Functions

- #define **dmaStreamSetPeripheral**(dmastp, addr)  
*Associates a peripheral data register to a DMA stream.*
- #define **dmaStreamSetMemory0**(dmastp, addr)  
*Associates a memory destination to a DMA stream.*
- #define **dmaStreamSetTransactionSize**(dmastp, size)  
*Sets the number of transfers to be performed.*
- #define **dmaStreamGetTransactionSize**(dmastp) ((size\_t)((dmastp)->channel->CNDTR))  
*Returns the number of transfers to be performed.*
- #define **dmaStreamSetMode**(dmastp, mode)  
*Programs the stream mode settings.*
- #define **dmaStreamEnable**(dmastp)  
*DMA stream enable.*
- #define **dmaStreamDisable**(dmastp)  
*DMA stream disable.*
- #define **dmaStreamClearInterrupt**(dmastp)  
*DMA stream interrupt sources clear.*
- #define **dmaStartMemCopy**(dmastp, mode, src, dst, n)  
*Starts a memory to memory operation using the specified stream.*
- #define **dmaWaitCompletion**(dmastp)  
*Polled wait for DMA transfer end.*

## Defines

- `#define STM32_DMA1_STREAMS_MASK 0x0000007F`  
*Mask of the DMA1 streams in dma\_streams\_mask.*
- `#define STM32_DMA2_STREAMS_MASK 0x00000F80`  
*Mask of the DMA2 streams in dma\_streams\_mask.*
- `#define STM32_DMA_CCR_RESET_VALUE 0x00000000`  
*Post-reset value of the stream CCR register.*
- `#define STM32_DMA_STREAMS 12`  
*Total number of DMA streams.*
- `#define STM32_DMA_ISR_MASK 0x0F`  
*Mask of the ISR bits passed to the DMA callback functions.*
- `#define STM32_DMA_GETCHANNEL(n, c) 0`  
*Returns the channel associated to the specified stream.*
- `#define STM32_DMA_STREAM_ID_MSK(dma, stream) (1 << STM32_DMA_STREAM_ID(dma, stream))`  
*Returns a DMA stream identifier mask.*
- `#define STM32_DMA_IS_VALID_ID(id, mask) (((1 << (id)) & (mask)))`  
*Checks if a DMA stream unique identifier belongs to a mask.*

## Typedefs

- `typedef void(* stm32_dmaisr_t )(void *p, uint32_t flags)`  
*STM32 DMA ISR function type.*

### 6.42.4 Function Documentation

#### 6.42.4.1 CH\_IRQ\_HANDLER ( DMA1\_Ch1\_IRQHandler )

DMA1 stream 1 shared interrupt handler.

##### Function Class:

Interrupt handler, this function should not be directly invoked.

#### 6.42.4.2 CH\_IRQ\_HANDLER ( DMA1\_Ch2\_IRQHandler )

DMA1 stream 2 shared interrupt handler.

##### Function Class:

Interrupt handler, this function should not be directly invoked.

#### 6.42.4.3 CH\_IRQ\_HANDLER ( DMA1\_Ch3\_IRQHandler )

DMA1 stream 3 shared interrupt handler.

##### Function Class:

Interrupt handler, this function should not be directly invoked.

**6.42.4.4 CH\_IRQ\_HANDLER ( DMA1\_Ch4\_IRQHandler )**

DMA1 stream 4 shared interrupt handler.

**Function Class:**

Interrupt handler, this function should not be directly invoked.

**6.42.4.5 CH\_IRQ\_HANDLER ( DMA1\_Ch5\_IRQHandler )**

DMA1 stream 5 shared interrupt handler.

**Function Class:**

Interrupt handler, this function should not be directly invoked.

**6.42.4.6 CH\_IRQ\_HANDLER ( DMA1\_Ch6\_IRQHandler )**

DMA1 stream 6 shared interrupt handler.

**Function Class:**

Interrupt handler, this function should not be directly invoked.

**6.42.4.7 CH\_IRQ\_HANDLER ( DMA1\_Ch7\_IRQHandler )**

DMA1 stream 7 shared interrupt handler.

**Function Class:**

Interrupt handler, this function should not be directly invoked.

**6.42.4.8 CH\_IRQ\_HANDLER ( DMA2\_Ch1\_IRQHandler )**

DMA2 stream 1 shared interrupt handler.

**Function Class:**

Interrupt handler, this function should not be directly invoked.

**6.42.4.9 CH\_IRQ\_HANDLER ( DMA2\_Ch2\_IRQHandler )**

DMA2 stream 2 shared interrupt handler.

**Function Class:**

Interrupt handler, this function should not be directly invoked.

**6.42.4.10 CH\_IRQ\_HANDLER ( DMA2\_Ch3\_IRQHandler )**

DMA2 stream 3 shared interrupt handler.

**Function Class:**

Interrupt handler, this function should not be directly invoked.

#### 6.42.4.11 CH\_IRQ\_HANDLER ( DMA2\_Ch4\_IRQHandler )

DMA2 stream 4 shared interrupt handler.

##### Function Class:

Interrupt handler, this function should not be directly invoked.

#### 6.42.4.12 CH\_IRQ\_HANDLER ( DMA2\_Ch5\_IRQHandler )

DMA2 stream 5 shared interrupt handler.

##### Function Class:

Interrupt handler, this function should not be directly invoked.

#### 6.42.4.13 void dmalnit ( void )

STM32 DMA helper initialization.

##### Function Class:

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

#### 6.42.4.14 bool\_t dmaStreamAllocate ( const stm32\_dma\_stream\_t \* *dmastp*, uint32\_t *priority*, stm32\_dmaisr\_t *func*, void \* *param* )

Allocates a DMA stream.

The stream is allocated and, if required, the DMA clock enabled. The function also enables the IRQ vector associated to the stream and initializes its priority.

##### Precondition

The stream must not be already in use or an error is returned.

##### Postcondition

The stream is allocated and the default ISR handler redirected to the specified function.

The stream ISR vector is enabled and its priority configured.

The stream must be freed using [dmaStreamRelease\(\)](#) before it can be reused with another peripheral.

The stream is in its post-reset state.

##### Note

This function can be invoked in both ISR or thread context.

##### Parameters

in	<i>dmastp</i>	pointer to a <a href="#">stm32_dma_stream_t</a> structure
in	<i>priority</i>	IRQ priority mask for the DMA stream
in	<i>func</i>	handling function pointer, can be NULL
in	<i>param</i>	a parameter to be passed to the handling function

##### Returns

The operation status.

**Return values**

<i>FALSE</i>	no error, stream taken.
<i>TRUE</i>	error, stream already taken.

**Function Class:**

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

**6.42.4.15 void dmaStreamRelease ( const [stm32\\_dma\\_stream\\_t](#) \* *dmastp* )**

Releases a DMA stream.

The stream is freed and, if required, the DMA clock disabled. Trying to release a unallocated stream is an illegal operation and is trapped if assertions are enabled.

**Precondition**

The stream must have been allocated using [dmaStreamAllocate\(\)](#).

**Postcondition**

The stream is again available.

**Note**

This function can be invoked in both ISR or thread context.

**Parameters**

in                    *dmastp*    pointer to a [stm32\\_dma\\_stream\\_t](#) structure

**Function Class:**

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

**6.42.5 Variable Documentation****6.42.5.1 const [stm32\\_dma\\_stream\\_t](#) \_stm32\_dma\_streams[STM32\_DMA\_STREAMS]****Initial value:**

```
{
{DMA1_Channel1, &DMA1->IFCR, 0, 0, DMA1_Channel1_IRQHandler},
{DMA1_Channel2, &DMA1->IFCR, 4, 1, DMA1_Channel2_IRQHandler},
{DMA1_Channel3, &DMA1->IFCR, 8, 2, DMA1_Channel3_IRQHandler},
{DMA1_Channel4, &DMA1->IFCR, 12, 3, DMA1_Channel4_IRQHandler},
{DMA1_Channel5, &DMA1->IFCR, 16, 4, DMA1_Channel5_IRQHandler},
{DMA1_Channel6, &DMA1->IFCR, 20, 5, DMA1_Channel6_IRQHandler},
{DMA1_Channel7, &DMA1->IFCR, 24, 6, DMA1_Channel7_IRQHandler},

{DMA2_Channel1, &DMA2->IFCR, 0, 7, DMA2_Channel1_IRQHandler},
{DMA2_Channel2, &DMA2->IFCR, 4, 8, DMA2_Channel2_IRQHandler},
{DMA2_Channel3, &DMA2->IFCR, 8, 9, DMA2_Channel3_IRQHandler},

{DMA2_Channel4, &DMA2->IFCR, 12, 10, DMA2_Channel4_IRQHandler},
{DMA2_Channel5, &DMA2->IFCR, 16, 11, DMA2_Channel5_IRQHandler},

}
}
```

DMA streams descriptors.

This table keeps the association between an unique stream identifier and the involved physical registers.

#### Note

Don't use this array directly, use the appropriate wrapper macros instead: STM32\_DMA1\_STREAM1, STM32\_DMA1\_STREAM2 etc.

### 6.42.6 Define Documentation

**6.42.6.1 #define STM32\_DMA1\_STREAMS\_MASK 0x0000007F**

Mask of the DMA1 streams in `dma_streams_mask`.

**6.42.6.2 #define STM32\_DMA2\_STREAMS\_MASK 0x00000F80**

Mask of the DMA2 streams in `dma_streams_mask`.

**6.42.6.3 #define STM32\_DMA\_CCR\_RESET\_VALUE 0x00000000**

Post-reset value of the stream CCR register.

**6.42.6.4 #define STM32\_DMA\_STREAMS 12**

Total number of DMA streams.

#### Note

This is the total number of streams among all the DMA units.

**6.42.6.5 #define STM32\_DMA\_ISR\_MASK 0x0F**

Mask of the ISR bits passed to the DMA callback functions.

**6.42.6.6 #define STM32\_DMA\_GETCHANNEL( n, c ) 0**

Returns the channel associated to the specified stream.

#### Parameters

in	<i>n</i>	the stream number (0...STM32_DMA_STREAMS-1)
in	<i>c</i>	a stream/channel association word, one channel per nibble, not associated channels must be set to 0xF

#### Returns

Always zero, in this platform there is no dynamic association between streams and channels.

**6.42.6.7 #define STM32\_DMA\_STREAM\_ID\_MSK( dma, stream )(1 << STM32\_DMA\_STREAM\_ID(dma, stream))**

Returns a DMA stream identifier mask.

#### Parameters

in	<i>dma</i>	the DMA unit number
in	<i>stream</i>	the stream number

**Returns**

A DMA stream identifier mask.

6.42.6.8 #define STM32\_DMA\_IS\_VALID\_ID( *id*, *mask* ) (((1 << (*id*)) & (*mask*)))

Checks if a DMA stream unique identifier belongs to a mask.

**Parameters**

in	<i>id</i>	the stream numeric identifier
in	<i>mask</i>	the stream numeric identifiers mask

**Return values**

*The* check result.

*FALSE* *id* does not belong to the mask.

*TRUE* *id* belongs to the mask.

6.42.6.9 #define STM32\_DMA\_STREAM\_ID( *dma*, *stream* ) (((*dma*) - 1) \* 7) + ((*stream*) - 1))

Returns an unique numeric identifier for a DMA stream.

**Parameters**

in	<i>dma</i>	the DMA unit number
in	<i>stream</i>	the stream number

**Returns**

An unique numeric stream identifier.

6.42.6.10 #define STM32\_DMA\_STREAM( *id* ) (&\_stm32\_dma\_streams[*id*])

Returns a pointer to a [stm32\\_dma\\_stream\\_t](#) structure.

**Parameters**

in	<i>id</i>	the stream numeric identifier
----	-----------	-------------------------------

**Returns**

A pointer to the [stm32\\_dma\\_stream\\_t](#) constant structure associated to the DMA stream.

6.42.6.11 #define STM32\_DMA\_CR\_DMEIE 0

Ignored by normal DMA.

6.42.6.12 #define STM32\_DMA\_CR\_CHSEL\_MASK 0

Ignored by normal DMA.

6.42.6.13 #define STM32\_DMA\_CR\_CHSEL( *n* ) 0

Ignored by normal DMA.

6.42.6.14 #define dmaStreamSetPeripheral( *dmastp*, *addr* )

**Value:**

```
{
    (dmastp)->channel->CPAR  = (uint32_t) (addr);
}
```

Associates a peripheral data register to a DMA stream.

**Note**

This function can be invoked in both ISR or thread context.

**Precondition**

The stream must have been allocated using [dmaStreamAllocate\(\)](#).

**Postcondition**

After use the stream can be released using [dmaStreamRelease\(\)](#).

**Parameters**

in	<i>dmastp</i>	pointer to a <a href="#">stm32_dma_stream_t</a> structure
in	<i>addr</i>	value to be written in the CPAR register

**Function Class:**

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

6.42.6.15 #define dmaStreamSetMemory0( *dmastp*, *addr* )

**Value:**

```
{
    (dmastp)->channel->CMAR  = (uint32_t) (addr);
}
```

Associates a memory destination to a DMA stream.

**Note**

This function can be invoked in both ISR or thread context.

**Precondition**

The stream must have been allocated using [dmaStreamAllocate\(\)](#).

**Postcondition**

After use the stream can be released using [dmaStreamRelease\(\)](#).

**Parameters**

in	<i>dmastp</i>	pointer to a <a href="#">stm32_dma_stream_t</a> structure
in	<i>addr</i>	value to be written in the CMAR register

**Function Class:**

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

6.42.6.16 #define dmaStreamSetTransactionSize( *dmastp*, *size* )

**Value:**

```
{
    (dmastp)->channel->CNDTR  = (uint32_t)(size);
}
```

Sets the number of transfers to be performed.

**Note**

This function can be invoked in both ISR or thread context.

**Precondition**

The stream must have been allocated using [dmaStreamAllocate\(\)](#).

**Postcondition**

After use the stream can be released using [dmaStreamRelease\(\)](#).

**Parameters**

in	<i>dmastp</i>	pointer to a <a href="#">stm32_dma_stream_t</a> structure
in	<i>size</i>	value to be written in the CNDTR register

**Function Class:**

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

6.42.6.17 #define dmaStreamGetTransactionSize( *dmastp* ) ((size\_t)((dmastp)->channel->CNDTR))

Returns the number of transfers to be performed.

**Note**

This function can be invoked in both ISR or thread context.

**Precondition**

The stream must have been allocated using [dmaStreamAllocate\(\)](#).

**Postcondition**

After use the stream can be released using [dmaStreamRelease\(\)](#).

**Parameters**

in	<i>dmastp</i>	pointer to a <a href="#">stm32_dma_stream_t</a> structure
----	---------------	---

**Returns**

The number of transfers to be performed.

**Function Class:**

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

6.42.6.18 #define dmaStreamSetMode( *dmastp*, *mode* )

**Value:**

```
{
    (dmastp)->channel->CCR  = (uint32_t)(mode);
}
```

Programs the stream mode settings.

#### Note

This function can be invoked in both ISR or thread context.

#### Precondition

The stream must have been allocated using `dmaStreamAllocate()`.

#### Postcondition

After use the stream can be released using `dmaStreamRelease()`.

#### Parameters

in	<code>dmastp</code>	pointer to a <code>stm32_dma_stream_t</code> structure
in	<code>mode</code>	value to be written in the CCR register

#### Function Class:

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

6.42.6.19 #define `dmaStreamEnable( dmastp )`

#### Value:

```
{
    (dmastp)->channel->CCR |= STM32_DMA_CR_EN;
}
```

DMA stream enable.

#### Note

This function can be invoked in both ISR or thread context.

#### Precondition

The stream must have been allocated using `dmaStreamAllocate()`.

#### Postcondition

After use the stream can be released using `dmaStreamRelease()`.

#### Parameters

in	<code>dmastp</code>	pointer to a <code>stm32_dma_stream_t</code> structure
----	---------------------	--

#### Function Class:

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

6.42.6.20 #define `dmaStreamDisable( dmastp )`

#### Value:

```
{
```

```
(dmastp)->channel->CCR &= ~STM32_DMA_CR_EN;
dmaStreamClearInterrupt(dmastp);
}
```

DMA stream disable.

The function disables the specified stream and then clears any pending interrupt.

#### Note

This function can be invoked in both ISR or thread context.

#### Precondition

The stream must have been allocated using [dmaStreamAllocate\(\)](#).

#### Postcondition

After use the stream can be released using [dmaStreamRelease\(\)](#).

#### Parameters

in                    *dmastp*    pointer to a [stm32\\_dma\\_stream\\_t](#) structure

#### Function Class:

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

**6.42.6.21 #define dmaStreamClearInterrupt( *dmastp* )**

#### Value:

```
{
  * (dmastp)->ifcr = STM32_DMA_ISR_MASK << (dmastp)->ishift;
}
```

DMA stream interrupt sources clear.

#### Note

This function can be invoked in both ISR or thread context.

#### Precondition

The stream must have been allocated using [dmaStreamAllocate\(\)](#).

#### Postcondition

After use the stream can be released using [dmaStreamRelease\(\)](#).

#### Parameters

in                    *dmastp*    pointer to a [stm32\\_dma\\_stream\\_t](#) structure

#### Function Class:

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

**6.42.6.22 #define dmaStartMemCopy( *dmastp*, *mode*, *src*, *dst*, *n* )**

#### Value:

```
{                    \
```

```

dmaStreamSetPeripheral(dmastp, src);
dmaStreamSetMemory0(dmastp, dst);
dmaStreamSetTransactionSize(dmastp, n);
dmaStreamSetMode(dmastp, (mode) |
                  STM32_DMA_CR_MINC | STM32_DMA_CR_PINC | \
                  STM32_DMA_CR_DIR_M2M | STM32_DMA_CR_EN); \
}

```

Starts a memory to memory operation using the specified stream.

#### Note

The default transfer data mode is "byte to byte" but it can be changed by specifying extra options in the `mode` parameter.

#### Precondition

The stream must have been allocated using `dmaStreamAllocate()`.

#### Postcondition

After use the stream can be released using `dmaStreamRelease()`.

#### Parameters

in	<code>dmastp</code>	pointer to a <code>stm32_dma_stream_t</code> structure
in	<code>mode</code>	value to be written in the CCR register, this value is implicitly ORed with:
		<ul style="list-style-type: none"> <li>• <code>STM32_DMA_CR_MINC</code></li> <li>• <code>STM32_DMA_CR_PINC</code></li> <li>• <code>STM32_DMA_CR_DIR_M2M</code></li> <li>• <code>STM32_DMA_CR_EN</code></li> </ul>
in	<code>src</code>	source address
in	<code>dst</code>	destination address
in	<code>n</code>	number of data units to copy

#### 6.42.6.23 #define dmaWaitCompletion( `dmastp` )

#### Value:

```

while ((dmastp)->channel->CNDTR > 0)
;
dmaStreamDisable(dmastp);
}

```

Polled wait for DMA transfer end.

#### Precondition

The stream must have been allocated using `dmaStreamAllocate()`.

#### Postcondition

After use the stream can be released using `dmaStreamRelease()`.

#### Parameters

in	<code>dmastp</code>	pointer to a <code>stm32_dma_stream_t</code> structure
----	---------------------	--

## 6.42.7 Typedef Documentation

6.42.7.1 `typedef void(* stm32_dmaisr_t)(void *p, uint32_t flags)`

STM32 DMA ISR function type.

#### Parameters

in	<i>p</i>	parameter for the registered function
in	<i>flags</i>	pre-shifted content of the ISR register, the bits are aligned to bit zero

## 6.43 STM32F1xx RCC Support

### 6.43.1 Detailed Description

This RCC helper driver is used by the other drivers in order to access the shared RCC resources in a consistent way.

### 6.43.2 Supported HW resources

- RCC.

### 6.43.3 STM32F1xx RCC driver implementation features

- Peripherals reset.
- Peripherals clock enable.
- Peripherals clock disable.

### Generic RCC operations

- `#define rccEnableAPB1(mask, lp)`  
*Enables the clock of one or more peripheral on the APB1 bus.*
- `#define rccDisableAPB1(mask, lp)`  
*Disables the clock of one or more peripheral on the APB1 bus.*
- `#define rccResetAPB1(mask)`  
*Resets one or more peripheral on the APB1 bus.*
- `#define rccEnableAPB2(mask, lp)`  
*Enables the clock of one or more peripheral on the APB2 bus.*
- `#define rccDisableAPB2(mask, lp)`  
*Disables the clock of one or more peripheral on the APB2 bus.*
- `#define rccResetAPB2(mask)`  
*Resets one or more peripheral on the APB2 bus.*
- `#define rccEnableAHB(mask, lp)`  
*Enables the clock of one or more peripheral on the AHB bus.*
- `#define rccDisableAHB(mask, lp)`  
*Disables the clock of one or more peripheral on the AHB bus.*
- `#define rccResetAHB(mask)`  
*Resets one or more peripheral on the AHB bus.*

## ADC peripherals specific RCC operations

- `#define rccEnableADC1(lp) rccEnableAPB2(RCC_APB2ENR_ADC1EN, lp)`  
*Enables the ADC1 peripheral clock.*
- `#define rccDisableADC1(lp) rccDisableAPB2(RCC_APB2ENR_ADC1EN, lp)`  
*Disables the ADC1 peripheral clock.*
- `#define rccResetADC1() rccResetAPB2(RCC_APB2RSTR_ADC1RST)`  
*Resets the ADC1 peripheral.*

## Backup domain interface specific RCC operations

- `#define rccEnableBKPIInterface(lp) rccEnableAPB1((RCC_APB1ENR_BKPEN | RCC_APB1ENR_PWREN), lp)`  
*Enables the BKP interface clock.*
- `#define rccDisableBKPIInterface(lp) rccDisableAPB1((RCC_APB1ENR_BKPEN | RCC_APB1ENR_PWREN), lp)`  
*Disables BKP interface clock.*
- `#define rccResetBKPIInterface() rccResetAPB1(RCC_APB1ENR_BKPRST)`  
*Resets the Backup Domain interface.*
- `#define rccResetBKP() (RCC->BDCR |= RCC_BDCR_BDRST)`  
*Resets the entire Backup Domain.*

## PWR interface specific RCC operations

- `#define rccEnablePWRInterface(lp) rccEnableAPB1(RCC_APB1ENR_PWREN, lp)`  
*Enables the PWR interface clock.*
- `#define rccDisablePWRInterface(lp) rccDisableAPB1(RCC_APB1ENR_BKPEN, lp)`  
*Disables PWR interface clock.*
- `#define rccResetPWRInterface() rccResetAPB1(RCC_APB1ENR_BKPRST)`  
*Resets the PWR interface.*

## CAN peripherals specific RCC operations

- `#define rccEnableCAN1(lp) rccEnableAPB1(RCC_APB1ENR_CAN1EN, lp)`  
*Enables the CAN1 peripheral clock.*
- `#define rccDisableCAN1(lp) rccDisableAPB1(RCC_APB1ENR_CAN1EN, lp)`  
*Disables the CAN1 peripheral clock.*
- `#define rccResetCAN1() rccResetAPB1(RCC_APB1RSTR_CAN1RST)`  
*Resets the CAN1 peripheral.*

## DMA peripherals specific RCC operations

- `#define rccEnableDMA1(lp) rccEnableAHB(RCC_AHBENR_DMA1EN, lp)`  
*Enables the DMA1 peripheral clock.*
- `#define rccDisableDMA1(lp) rccDisableAHB(RCC_AHBENR_DMA1EN, lp)`  
*Disables the DMA1 peripheral clock.*
- `#define rccResetDMA1()`  
*Resets the DMA1 peripheral.*
- `#define rccEnableDMA2(lp) rccEnableAHB(RCC_AHBENR_DMA2EN, lp)`  
*Enables the DMA2 peripheral clock.*

- `#define rccDisableDMA2(lp) rccDisableAHB(RCC_AHBENR_DMA2EN, lp)`  
*Disables the DMA2 peripheral clock.*
- `#define rccResetDMA2()`  
*Resets the DMA1 peripheral.*

## ETH peripheral specific RCC operations

- `#define rccEnableETH(lp)`  
*Enables the ETH peripheral clock.*
- `#define rccDisableETH(lp)`  
*Disables the ETH peripheral clock.*
- `#define rccResetETH() rccResetAHB(RCC_AHBRSTR_ETHMACRST)`  
*Resets the ETH peripheral.*

## I2C peripherals specific RCC operations

- `#define rccEnableI2C1(lp) rccEnableAPB1(RCC_APB1ENR_I2C1EN, lp)`  
*Enables the I2C1 peripheral clock.*
- `#define rccDisableI2C1(lp) rccDisableAPB1(RCC_APB1ENR_I2C1EN, lp)`  
*Disables the I2C1 peripheral clock.*
- `#define rccResetI2C1() rccResetAPB1(RCC_APB1RSTR_I2C1RST)`  
*Resets the I2C1 peripheral.*
- `#define rccEnableI2C2(lp) rccEnableAPB1(RCC_APB1ENR_I2C2EN, lp)`  
*Enables the I2C2 peripheral clock.*
- `#define rccDisableI2C2(lp) rccDisableAPB1(RCC_APB1ENR_I2C2EN, lp)`  
*Disables the I2C2 peripheral clock.*
- `#define rccResetI2C2() rccResetAPB1(RCC_APB1RSTR_I2C2RST)`  
*Resets the I2C2 peripheral.*

## SDIO peripheral specific RCC operations

- `#define rccEnableSDIO(lp) rccEnableAHB(RCC_AHBENR_SDIOEN, lp)`  
*Enables the SDIO peripheral clock.*
- `#define rccDisableSDIO(lp) rccDisableAHB(RCC_AHBENR_SDIOEN, lp)`  
*Disables the SDIO peripheral clock.*
- `#define rccResetSDIO()`  
*Resets the SDIO peripheral.*

## SPI peripherals specific RCC operations

- `#define rccEnableSPI1(lp) rccEnableAPB2(RCC_APB2ENR_SPI1EN, lp)`  
*Enables the SPI1 peripheral clock.*
- `#define rccDisableSPI1(lp) rccDisableAPB2(RCC_APB2ENR_SPI1EN, lp)`  
*Disables the SPI1 peripheral clock.*
- `#define rccResetSPI1() rccResetAPB2(RCC_APB2RSTR_SPI1RST)`  
*Resets the SPI1 peripheral.*
- `#define rccEnableSPI2(lp) rccEnableAPB1(RCC_APB1ENR_SPI2EN, lp)`  
*Enables the SPI2 peripheral clock.*
- `#define rccDisableSPI2(lp) rccDisableAPB1(RCC_APB1ENR_SPI2EN, lp)`  
*Disables the SPI2 peripheral clock.*

- `#define rccResetSPI2() rccResetAPB1(RCC_APB1RSTR_SPI2RST)`  
*Resets the SPI2 peripheral.*
- `#define rccEnableSPI3(lp) rccEnableAPB1(RCC_APB1ENR_SPI3EN, lp)`  
*Enables the SPI3 peripheral clock.*
- `#define rccDisableSPI3(lp) rccDisableAPB1(RCC_APB1ENR_SPI3EN, lp)`  
*Disables the SPI3 peripheral clock.*
- `#define rccResetSPI3() rccResetAPB1(RCC_APB1RSTR_SPI3RST)`  
*Resets the SPI3 peripheral.*

## TIM peripherals specific RCC operations

- `#define rccEnableTIM1(lp) rccEnableAPB2(RCC_APB2ENR_TIM1EN, lp)`  
*Enables the TIM1 peripheral clock.*
- `#define rccDisableTIM1(lp) rccDisableAPB2(RCC_APB2ENR_TIM1EN, lp)`  
*Disables the TIM1 peripheral clock.*
- `#define rccResetTIM1() rccResetAPB2(RCC_APB2RSTR_TIM1RST)`  
*Resets the TIM1 peripheral.*
- `#define rccEnableTIM2(lp) rccEnableAPB1(RCC_APB1ENR_TIM2EN, lp)`  
*Enables the TIM2 peripheral clock.*
- `#define rccDisableTIM2(lp) rccDisableAPB1(RCC_APB1ENR_TIM2EN, lp)`  
*Disables the TIM2 peripheral clock.*
- `#define rccResetTIM2() rccResetAPB1(RCC_APB1RSTR_TIM2RST)`  
*Resets the TIM2 peripheral.*
- `#define rccEnableTIM3(lp) rccEnableAPB1(RCC_APB1ENR_TIM3EN, lp)`  
*Enables the TIM3 peripheral clock.*
- `#define rccDisableTIM3(lp) rccDisableAPB1(RCC_APB1ENR_TIM3EN, lp)`  
*Disables the TIM3 peripheral clock.*
- `#define rccResetTIM3() rccResetAPB1(RCC_APB1RSTR_TIM3RST)`  
*Resets the TIM3 peripheral.*
- `#define rccEnableTIM4(lp) rccEnableAPB1(RCC_APB1ENR_TIM4EN, lp)`  
*Enables the TIM4 peripheral clock.*
- `#define rccDisableTIM4(lp) rccDisableAPB1(RCC_APB1ENR_TIM4EN, lp)`  
*Disables the TIM4 peripheral clock.*
- `#define rccResetTIM4() rccResetAPB1(RCC_APB1RSTR_TIM4RST)`  
*Resets the TIM4 peripheral.*
- `#define rccEnableTIM5(lp) rccEnableAPB1(RCC_APB1ENR_TIM5EN, lp)`  
*Enables the TIM5 peripheral clock.*
- `#define rccDisableTIM5(lp) rccDisableAPB1(RCC_APB1ENR_TIM5EN, lp)`  
*Disables the TIM5 peripheral clock.*
- `#define rccResetTIM5() rccResetAPB1(RCC_APB1RSTR_TIM5RST)`  
*Resets the TIM5 peripheral.*
- `#define rccEnableTIM8(lp) rccEnableAPB2(RCC_APB2ENR_TIM8EN, lp)`  
*Enables the TIM8 peripheral clock.*
- `#define rccDisableTIM8(lp) rccDisableAPB2(RCC_APB2ENR_TIM8EN, lp)`  
*Disables the TIM8 peripheral clock.*
- `#define rccResetTIM8() rccResetAPB2(RCC_APB2RSTR_TIM8RST)`  
*Resets the TIM8 peripheral.*

## USART/UART peripherals specific RCC operations

- `#define rccEnableUSART1(lp) rccEnableAPB2(RCC_APB2ENR_USART1EN, lp)`  
*Enables the USART1 peripheral clock.*
- `#define rccDisableUSART1(lp) rccDisableAPB2(RCC_APB2ENR_USART1EN, lp)`  
*Disables the USART1 peripheral clock.*
- `#define rccResetUSART1() rccResetAPB2(RCC_APB2RSTR_USART1RST)`  
*Resets the USART1 peripheral.*
- `#define rccEnableUSART2(lp) rccEnableAPB1(RCC_APB1ENR_USART2EN, lp)`  
*Enables the USART2 peripheral clock.*
- `#define rccDisableUSART2(lp) rccDisableAPB1(RCC_APB1ENR_USART2EN, lp)`  
*Disables the USART2 peripheral clock.*
- `#define rccResetUSART2() rccResetAPB1(RCC_APB1RSTR_USART2RST)`  
*Resets the USART2 peripheral.*
- `#define rccEnableUSART3(lp) rccEnableAPB1(RCC_APB1ENR_USART3EN, lp)`  
*Enables the USART3 peripheral clock.*
- `#define rccDisableUSART3(lp) rccDisableAPB1(RCC_APB1ENR_USART3EN, lp)`  
*Disables the USART3 peripheral clock.*
- `#define rccResetUSART3() rccResetAPB1(RCC_APB1RSTR_USART3RST)`  
*Resets the USART3 peripheral.*
- `#define rccEnableUART4(lp) rccEnableAPB1(RCC_APB1ENR_UART4EN, lp)`  
*Enables the UART4 peripheral clock.*
- `#define rccDisableUART4(lp) rccDisableAPB1(RCC_APB1ENR_UART4EN, lp)`  
*Disables the UART4 peripheral clock.*
- `#define rccResetUART4() rccResetAPB1(RCC_APB1RSTR_UART4RST)`  
*Resets the UART4 peripheral.*
- `#define rccEnableUART5(lp) rccEnableAPB1(RCC_APB1ENR_UART5EN, lp)`  
*Enables the UART5 peripheral clock.*
- `#define rccDisableUART5(lp) rccDisableAPB1(RCC_APB1ENR_UART5EN, lp)`  
*Disables the UART5 peripheral clock.*
- `#define rccResetUART5() rccResetAPB1(RCC_APB1RSTR_UART5RST)`  
*Resets the UART5 peripheral.*

## USB peripheral specific RCC operations

- `#define rccEnableUSB(lp) rccEnableAPB1(RCC_APB1ENR_USBEN, lp)`  
*Enables the USB peripheral clock.*
- `#define rccDisableUSB(lp) rccDisableAPB1(RCC_APB1ENR_USBEN, lp)`  
*Disables the USB peripheral clock.*
- `#define rccResetUSB() rccResetAPB1(RCC_APB1RSTR_USBRST)`  
*Resets the USB peripheral.*

### 6.43.4 Define Documentation

#### 6.43.4.1 `#define rccEnableAPB1( mask, lp )`

**Value:**

```
{
    RCC->APB1ENR |= (mask);
}
```

Enables the clock of one or more peripheral on the APB1 bus.

#### Note

The *lp* parameter is ignored in this family.

#### Parameters

in	<i>mask</i>	APB1 peripherals mask
in	<i>lp</i>	low power enable flag

#### Function Class:

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

#### 6.43.4.2 #define rccDisableAPB1( *mask*, *lp* )

#### Value:

```
{
    RCC->APB1ENR &= ~(mask);
}
```

Disables the clock of one or more peripheral on the APB1 bus.

#### Note

The *lp* parameter is ignored in this family.

#### Parameters

in	<i>mask</i>	APB1 peripherals mask
in	<i>lp</i>	low power enable flag

#### Function Class:

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

#### 6.43.4.3 #define rccResetAPB1( *mask* )

#### Value:

```
{
    RCC->APB1RSTR |= (mask);
    RCC->APB1RSTR = 0;
}
```

Resets one or more peripheral on the APB1 bus.

#### Parameters

in	<i>mask</i>	APB1 peripherals mask
----	-------------	-----------------------

#### Function Class:

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

#### 6.43.4.4 #define rccEnableAPB2( *mask*, *lp* )

#### Value:

```
{
  RCC->APB2ENR |= (mask);
}
```

Enables the clock of one or more peripheral on the APB2 bus.

#### Note

The *lp* parameter is ignored in this family.

#### Parameters

in	<i>mask</i>	APB2 peripherals mask
in	<i>lp</i>	low power enable flag

#### Function Class:

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

#### 6.43.4.5 #define rccDisableAPB2( *mask*, *lp* )

#### Value:

```
{
  RCC->APB2ENR &= ~ (mask);
}
```

Disables the clock of one or more peripheral on the APB2 bus.

#### Note

The *lp* parameter is ignored in this family.

#### Parameters

in	<i>mask</i>	APB2 peripherals mask
in	<i>lp</i>	low power enable flag

#### Function Class:

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

#### 6.43.4.6 #define rccResetAPB2( *mask* )

#### Value:

```
{
  RCC->APB2RSTR |= (mask);
  RCC->APB2RSTR = 0;
}
```

Resets one or more peripheral on the APB2 bus.

#### Parameters

in	<i>mask</i>	APB2 peripherals mask
----	-------------	-----------------------

#### Function Class:

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

6.43.4.7 #define rccEnableAHB( *mask*, *lp* )**Value:**

```
{
    RCC->AHBENR |= (mask);
}
```

Enables the clock of one or more peripheral on the AHB bus.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in	<i>mask</i>	AHB peripherals mask
in	<i>lp</i>	low power enable flag

**Function Class:**

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

6.43.4.8 #define rccDisableAHB( *mask*, *lp* )**Value:**

```
{
    RCC->AHBENR &= ~ (mask);
}
```

Disables the clock of one or more peripheral on the AHB bus.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in	<i>mask</i>	AHB peripherals mask
in	<i>lp</i>	low power enable flag

**Function Class:**

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

6.43.4.9 #define rccResetAHB( *mask* )**Value:**

```
{
    RCC->AHBRSTR |= (mask);
    RCC->AHBRSTR = 0;
}
```

Resets one or more peripheral on the AHB bus.

**Parameters**

in	<i>mask</i>	AHB peripherals mask
----	-------------	----------------------

**Function Class:**

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

6.43.4.10 #define rccEnableADC1( *lp* ) rccEnableAPB2(RCC\_APB2ENR\_ADC1EN, *lp*)

Enables the ADC1 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                   *lp* low power enable flag

**Function Class:**

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

6.43.4.11 #define rccDisableADC1( *lp* ) rccDisableAPB2(RCC\_APB2ENR\_ADC1EN, *lp*)

Disables the ADC1 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                   *lp* low power enable flag

**Function Class:**

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

6.43.4.12 #define rccResetADC1( ) rccResetAPB2(RCC\_APB2RSTR\_ADC1RST)

Resets the ADC1 peripheral.

**Function Class:**

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

6.43.4.13 #define rccEnableBKPIface( *lp* ) rccEnableAPB1((RCC\_APB1ENR\_BKPEN | RCC\_APB1ENR\_PWREN), *lp*)

Enables the BKP interface clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                   *lp* low power enable flag

**Function Class:**

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

6.43.4.14 #define rccDisableBKPInterface( *lp* ) rccDisableAPB1((RCC\_APB1ENR\_BKPEN | RCC\_APB1ENR\_PWREN), *lp*)

Disables BKP interface clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                   *lp* low power enable flag

**Function Class:**

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

6.43.4.15 #define rccResetBKPInterface( ) rccResetAPB1(RCC\_APB1ENR\_BKPRST)

Resets the Backup Domain interface.

**Function Class:**

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

6.43.4.16 #define rccResetBKP( )(RCC->BDCR |= RCC\_BDCR\_BDRST)

Resets the entire Backup Domain.

**Function Class:**

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

6.43.4.17 #define rccEnablePWRInterface( *lp* ) rccEnableAPB1(RCC\_APB1ENR\_PWREN, *lp*)

Enables the PWR interface clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                   *lp* low power enable flag

**Function Class:**

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

6.43.4.18 #define rccDisablePWRInterface( *lp* ) rccDisableAPB1(RCC\_APB1ENR\_BKPEN, *lp*)

Disables PWR interface clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                   *lp* low power enable flag

**Function Class:**

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

6.43.4.19 #define rccResetPWRInterface( ) rccResetAPB1(RCC\_APB1ENR\_BKPRST)

Resets the PWR interface.

**Function Class:**

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

6.43.4.20 #define rccEnableCAN1( *lp* ) rccEnableAPB1(RCC\_APB1ENR\_CAN1EN, *lp*)

Enables the CAN1 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                   *lp* low power enable flag

**Function Class:**

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

6.43.4.21 #define rccDisableCAN1( *lp* ) rccDisableAPB1(RCC\_APB1ENR\_CAN1EN, *lp*)

Disables the CAN1 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                   *lp* low power enable flag

**Function Class:**

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

6.43.4.22 #define rccResetCAN1( ) rccResetAPB1(RCC\_APB1RSTR\_CAN1RST)

Resets the CAN1 peripheral.

**Function Class:**

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

6.43.4.23 #define rccEnableDMA1( *lp* ) rccEnableAHB(RCC\_AHBENR\_DMA1EN, *lp*)

Enables the DMA1 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in *lp* low power enable flag

**Function Class:**

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

6.43.4.24 #define rccDisableDMA1( *lp* ) rccDisableAHB(RCC\_AHBENR\_DMA1EN, *lp*)

Disables the DMA1 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in *lp* low power enable flag

**Function Class:**

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

6.43.4.25 #define rccResetDMA1( )

Resets the DMA1 peripheral.

**Note**

Not supported in this family, does nothing.

**Function Class:**

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

6.43.4.26 #define rccEnableDMA2( *lp* ) rccEnableAHB(RCC\_AHBENR\_DMA2EN, *lp*)

Enables the DMA2 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in *lp* low power enable flag

**Function Class:**

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

---

```
6.43.4.27 #define rccDisableDMA2( /p ) rccDisableAHB(RCC_AHBENR_DMA2EN, lp)
```

Disables the DMA2 peripheral clock.

#### Note

The *lp* parameter is ignored in this family.

#### Parameters

in	<i>lp</i> low power enable flag
----	---------------------------------

#### Function Class:

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

---

```
6.43.4.28 #define rccResetDMA2( )
```

Resets the DMA1 peripheral.

#### Note

Not supported in this family, does nothing.

#### Function Class:

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

---

```
6.43.4.29 #define rccEnableETH( /p )
```

#### Value:

```
rccEnableAHB (RCC_AHBENR_ETHMACEN | \
                  RCC_AHBENR_ETHMACTXEN | \
                  RCC_AHBENR_ETHMACRXEN, /p)
```

Enables the ETH peripheral clock.

#### Note

The *lp* parameter is ignored in this family.

#### Parameters

in	<i>lp</i> low power enable flag
----	---------------------------------

#### Function Class:

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

---

```
6.43.4.30 #define rccDisableETH( /p )
```

#### Value:

```
rccDisableAHB (RCC_AHBENR_ETHMACEN | \
                  RCC_AHBENR_ETHMACTXEN | \
                  RCC_AHBENR_ETHMACRXEN, /p)
```

Disables the ETH peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                   *lp* low power enable flag

**Function Class:**

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

**6.43.4.31 #define rccResetETH( ) rccResetAHB(RCC\_AHBRSTR\_ETHMACRST)**

Resets the ETH peripheral.

**Function Class:**

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

**6.43.4.32 #define rccEnableI2C1( *lp* ) rccEnableAPB1(RCC\_APB1ENR\_I2C1EN, *lp*)**

Enables the I2C1 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                   *lp* low power enable flag

**Function Class:**

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

**6.43.4.33 #define rccDisableI2C1( *lp* ) rccDisableAPB1(RCC\_APB1ENR\_I2C1EN, *lp*)**

Disables the I2C1 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                   *lp* low power enable flag

**Function Class:**

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

**6.43.4.34 #define rccResetI2C1( ) rccResetAPB1(RCC\_APB1RSTR\_I2C1RST)**

Resets the I2C1 peripheral.

**Function Class:**

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

6.43.4.35 #define rccEnableI2C2( *lp* ) rccEnableAPB1(RCC\_APB1ENR\_I2C2EN, *lp*)

Enables the I2C2 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in *lp* low power enable flag

**Function Class:**

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

6.43.4.36 #define rccDisableI2C2( *lp* ) rccDisableAPB1(RCC\_APB1ENR\_I2C2EN, *lp*)

Disables the I2C2 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in *lp* low power enable flag

**Function Class:**

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

6.43.4.37 #define rccResetI2C2( ) rccResetAPB1(RCC\_APB1RSTR\_I2C2RST)

Resets the I2C2 peripheral.

**Function Class:**

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

6.43.4.38 #define rccEnableSDIO( *lp* ) rccEnableAHB(RCC\_AHBENR\_SDIOEN, *lp*)

Enables the SDIO peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in *lp* low power enable flag

**Function Class:**

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

6.43.4.39 #define rccDisableSDIO( *lp* ) rccDisableAHB(RCC\_AHBENR\_SDIOEN, *lp*)

Disables the SDIO peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in *lp* low power enable flag

**Function Class:**

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

6.43.4.40 #define rccResetSDIO( )

Resets the SDIO peripheral.

**Note**

Not supported in this family, does nothing.

**Function Class:**

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

6.43.4.41 #define rccEnableSPI1( *lp* ) rccEnableAPB2(RCC\_APB2ENR\_SPI1EN, *lp*)

Enables the SPI1 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in *lp* low power enable flag

**Function Class:**

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

6.43.4.42 #define rccDisableSPI1( *lp* ) rccDisableAPB2(RCC\_APB2ENR\_SPI1EN, *lp*)

Disables the SPI1 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in *lp* low power enable flag

**Function Class:**

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

```
6.43.4.43 #define rccResetSPI1( ) rccResetAPB2(RCC_APB2RSTR_SPI1RST)
```

Resets the SPI1 peripheral.

**Function Class:**

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

```
6.43.4.44 #define rccEnableSPI2( lp ) rccEnableAPB1(RCC_APB1ENR_SPI2EN, lp)
```

Enables the SPI2 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                   *lp* low power enable flag

**Function Class:**

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

```
6.43.4.45 #define rccDisableSPI2( lp ) rccDisableAPB1(RCC_APB1ENR_SPI2EN, lp)
```

Disables the SPI2 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                   *lp* low power enable flag

**Function Class:**

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

```
6.43.4.46 #define rccResetSPI2( ) rccResetAPB1(RCC_APB1RSTR_SPI2RST)
```

Resets the SPI2 peripheral.

**Function Class:**

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

```
6.43.4.47 #define rccEnableSPI3( lp ) rccEnableAPB1(RCC_APB1ENR_SPI3EN, lp)
```

Enables the SPI3 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                    *lp* low power enable flag

**Function Class:**

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

6.43.4.48 #define rccDisableSPI3( *lp* ) rccDisableAPB1(RCC\_APB1ENR\_SPI3EN, *lp*)

Disables the SPI3 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                    *lp* low power enable flag

**Function Class:**

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

6.43.4.49 #define rccResetSPI3( ) rccResetAPB1(RCC\_APB1RSTR\_SPI3RST)

Resets the SPI3 peripheral.

**Function Class:**

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

6.43.4.50 #define rccEnableTIM1( *lp* ) rccEnableAPB2(RCC\_APB2ENR\_TIM1EN, *lp*)

Enables the TIM1 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                    *lp* low power enable flag

**Function Class:**

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

6.43.4.51 #define rccDisableTIM1( *lp* ) rccDisableAPB2(RCC\_APB2ENR\_TIM1EN, *lp*)

Disables the TIM1 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                    *lp* low power enable flag

**Function Class:**

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

6.43.4.52 #define rccResetTIM1( ) rccResetAPB2(RCC\_APB2RSTR\_TIM1RST)

Resets the TIM1 peripheral.

**Function Class:**

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

6.43.4.53 #define rccEnableTIM2( *lp* ) rccEnableAPB1(RCC\_APB1ENR\_TIM2EN, *lp*)

Enables the TIM2 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                   *lp* low power enable flag

**Function Class:**

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

6.43.4.54 #define rccDisableTIM2( *lp* ) rccDisableAPB1(RCC\_APB1ENR\_TIM2EN, *lp*)

Disables the TIM2 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                   *lp* low power enable flag

**Function Class:**

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

6.43.4.55 #define rccResetTIM2( ) rccResetAPB1(RCC\_APB1RSTR\_TIM2RST)

Resets the TIM2 peripheral.

**Function Class:**

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

6.43.4.56 #define rccEnableTIM3( *lp* ) rccEnableAPB1(RCC\_APB1ENR\_TIM3EN, *lp*)

Enables the TIM3 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                   *lp* low power enable flag

**Function Class:**

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

6.43.4.57 #define rccDisableTIM3( *lp* ) rccDisableAPB1(RCC\_APB1ENR\_TIM3EN, *lp*)

Disables the TIM3 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                   *lp* low power enable flag

**Function Class:**

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

6.43.4.58 #define rccResetTIM3( ) rccResetAPB1(RCC\_APB1RSTR\_TIM3RST)

Resets the TIM3 peripheral.

**Function Class:**

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

6.43.4.59 #define rccEnableTIM4( *lp* ) rccEnableAPB1(RCC\_APB1ENR\_TIM4EN, *lp*)

Enables the TIM4 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                   *lp* low power enable flag

**Function Class:**

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

6.43.4.60 #define rccDisableTIM4( *lp* ) rccDisableAPB1(RCC\_APB1ENR\_TIM4EN, *lp*)

Disables the TIM4 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                    *lp* low power enable flag

**Function Class:**

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

6.43.4.61 #define rccResetTIM4( ) rccResetAPB1(RCC\_APB1RSTR\_TIM4RST)

Resets the TIM4 peripheral.

**Function Class:**

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

6.43.4.62 #define rccEnableTIM5( *lp* ) rccEnableAPB1(RCC\_APB1ENR\_TIM5EN, *lp*)

Enables the TIM5 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                    *lp* low power enable flag

**Function Class:**

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

6.43.4.63 #define rccDisableTIM5( *lp* ) rccDisableAPB1(RCC\_APB1ENR\_TIM5EN, *lp*)

Disables the TIM5 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                    *lp* low power enable flag

**Function Class:**

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

6.43.4.64 #define rccResetTIM5( ) rccResetAPB1(RCC\_APB1RSTR\_TIM5RST)

Resets the TIM5 peripheral.

**Function Class:**

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

6.43.4.65 #define rccEnableTIM8( *lp* ) rccEnableAPB2(RCC\_APB2ENR\_TIM8EN, *lp*)

Enables the TIM8 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in *lp* low power enable flag

**Function Class:**

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

6.43.4.66 #define rccDisableTIM8( *lp* ) rccDisableAPB2(RCC\_APB2ENR\_TIM8EN, *lp*)

Disables the TIM8 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in *lp* low power enable flag

**Function Class:**

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

6.43.4.67 #define rccResetTIM8( ) rccResetAPB2(RCC\_APB2RSTR\_TIM8RST)

Resets the TIM8 peripheral.

**Function Class:**

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

6.43.4.68 #define rccEnableUSART1( *lp* ) rccEnableAPB2(RCC\_APB2ENR\_USART1EN, *lp*)

Enables the USART1 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in *lp* low power enable flag

**Function Class:**

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

```
6.43.4.69 #define rccDisableUSART1( lp ) rccDisableAPB2(RCC_APB2ENR_USART1EN, lp)
```

Disables the USART1 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in *lp* low power enable flag

**Function Class:**

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

```
6.43.4.70 #define rccResetUSART1( ) rccResetAPB2(RCC_APB2RSTR_USART1RST)
```

Resets the USART1 peripheral.

**Function Class:**

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

```
6.43.4.71 #define rccEnableUSART2( lp ) rccEnableAPB1(RCC_APB1ENR_USART2EN, lp)
```

Enables the USART2 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in *lp* low power enable flag

**Function Class:**

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

```
6.43.4.72 #define rccDisableUSART2( lp ) rccDisableAPB1(RCC_APB1ENR_USART2EN, lp)
```

Disables the USART2 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in *lp* low power enable flag

**Function Class:**

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

```
6.43.4.73 #define rccResetUSART2( ) rccResetAPB1(RCC_APB1RSTR_USART2RST)
```

Resets the USART2 peripheral.

**Function Class:**

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

```
6.43.4.74 #define rccEnableUSART3( /p ) rccEnableAPB1(RCC_APB1ENR_USART3EN, lp)
```

Enables the USART3 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                   *lp* low power enable flag

**Function Class:**

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

```
6.43.4.75 #define rccDisableUSART3( /p ) rccDisableAPB1(RCC_APB1ENR_USART3EN, lp)
```

Disables the USART3 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                   *lp* low power enable flag

**Function Class:**

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

```
6.43.4.76 #define rccResetUSART3( ) rccResetAPB1(RCC_APB1RSTR_USART3RST)
```

Resets the USART3 peripheral.

**Function Class:**

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

```
6.43.4.77 #define rccEnableUART4( /p ) rccEnableAPB1(RCC_APB1ENR_UART4EN, lp)
```

Enables the UART4 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                    *lp* low power enable flag

**Function Class:**

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

6.43.4.78 #define rccDisableUART4( *lp* ) rccDisableAPB1(RCC\_APB1ENR\_UART4EN, *lp*)

Disables the UART4 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                    *lp* low power enable flag

**Function Class:**

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

6.43.4.79 #define rccResetUART4( ) rccResetAPB1(RCC\_APB1RSTR\_UART4RST)

Resets the UART4 peripheral.

**Function Class:**

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

6.43.4.80 #define rccEnableUART5( *lp* ) rccEnableAPB1(RCC\_APB1ENR\_UART5EN, *lp*)

Enables the UART5 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                    *lp* low power enable flag

**Function Class:**

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

6.43.4.81 #define rccDisableUART5( *lp* ) rccDisableAPB1(RCC\_APB1ENR\_UART5EN, *lp*)

Disables the UART5 peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                    *lp* low power enable flag

**Function Class:**

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

6.43.4.82 #define rccResetUART5( ) rccResetAPB1(RCC\_APB1RSTR\_UART5RST)

Resets the UART5 peripheral.

**Function Class:**

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

6.43.4.83 #define rccEnableUSB( *lp* ) rccEnableAPB1(RCC\_APB1ENR\_USBEN, *lp*)

Enables the USB peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                   *lp* low power enable flag

**Function Class:**

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

6.43.4.84 #define rccDisableUSB( *lp* ) rccDisableAPB1(RCC\_APB1ENR\_USBEN, *lp*)

Disables the USB peripheral clock.

**Note**

The *lp* parameter is ignored in this family.

**Parameters**

in                   *lp* low power enable flag

**Function Class:**

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

6.43.4.85 #define rccResetUSB( ) rccResetAPB1(RCC\_APB1RSTR\_USBRST)

Resets the USB peripheral.

**Function Class:**

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

## Chapter 7

# Data Structure Documentation

## 7.1 ADCConfig Struct Reference

### 7.1.1 Detailed Description

Driver configuration structure.

#### Note

It could be empty on some architectures.

```
#include <adc_ll.h>
```

## 7.2 ADCConversionGroup Struct Reference

### 7.2.1 Detailed Description

Conversion group configuration structure.

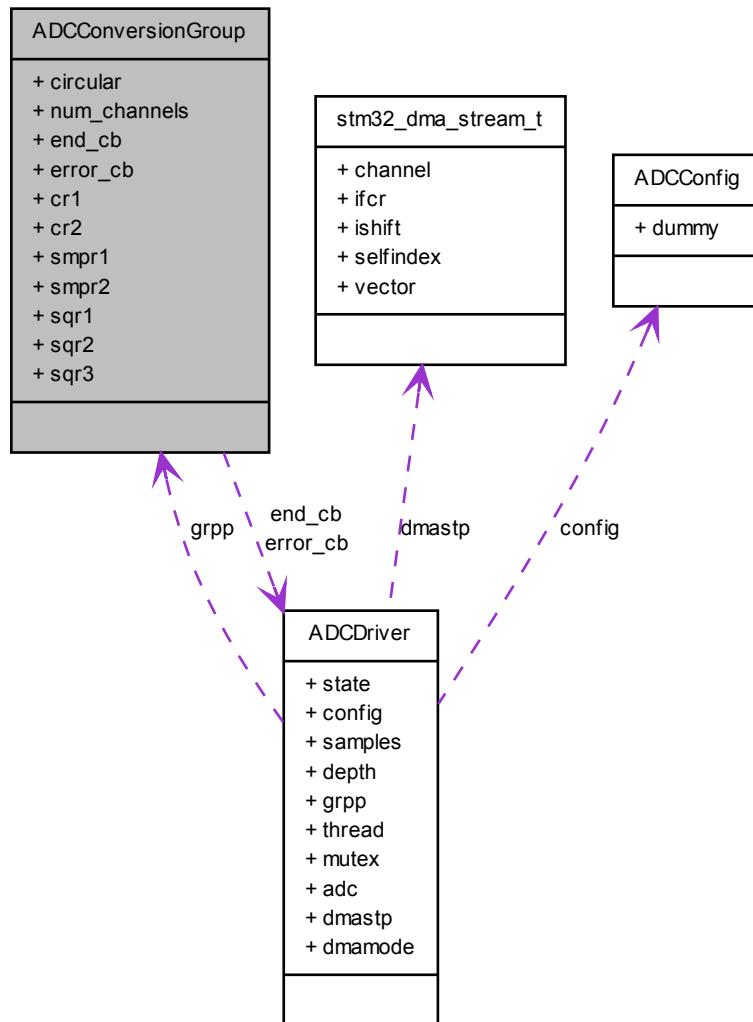
This implementation-dependent structure describes a conversion operation.

#### Note

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

```
#include <adc_ll.h>
```

Collaboration diagram for ADCConversionGroup:



## Data Fields

- `bool_t 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`.*
- `uint32_t cr1`  
*ADC CR1 register initialization data.*
- `uint32_t cr2`

- `uint32_t smpr1`  
*ADC SMPR1 register initialization data.*
- `uint32_t smpr2`  
*ADC SMPR2 register initialization data.*
- `uint32_t sqr1`  
*ADC SQR1 register initialization data.*
- `uint32_t sqr2`  
*ADC SQR2 register initialization data.*
- `uint32_t sqr3`  
*ADC SQR3 register initialization data.*

## 7.2.2 Field Documentation

### 7.2.2.1 `bool_t ADCConversionGroup::circular`

Enables the circular buffer mode for the group.

### 7.2.2.2 `adc_channels_num_t ADCConversionGroup::num_channels`

Number of the analog channels belonging to the conversion group.

### 7.2.2.3 `adccallback_t ADCConversionGroup::end_cb`

Callback function associated to the group or NULL.

### 7.2.2.4 `adcerrorcallback_t ADCConversionGroup::error_cb`

Error callback or NULL.

### 7.2.2.5 `uint32_t ADCConversionGroup::cr1`

ADC CR1 register initialization data.

#### Note

All the required bits must be defined into this field except `ADC_CR1_SCAN` that is enforced inside the driver.

### 7.2.2.6 `uint32_t ADCConversionGroup::cr2`

ADC CR2 register initialization data.

#### Note

All the required bits must be defined into this field except `ADC_CR2_DMA`, `ADC_CR2_CONT` and `ADC_CR2_ADON` that are enforced inside the driver.

### 7.2.2.7 `uint32_t ADCConversionGroup::smpr1`

ADC SMPR1 register initialization data.

In this field must be specified the sample times for channels 10...17.

**7.2.2.8 uint32\_t ADCConversionGroup::smpr2**

ADC SMPR2 register initialization data.

In this field must be specified the sample times for channels 0...9.

**7.2.2.9 uint32\_t ADCConversionGroup::sqr1**

ADC SQR1 register initialization data.

Conversion group sequence 13...16 + sequence length.

**7.2.2.10 uint32\_t ADCConversionGroup::sqr2**

ADC SQR2 register initialization data.

Conversion group sequence 7...12.

**7.2.2.11 uint32\_t ADCConversionGroup::sqr3**

ADC SQR3 register initialization data.

Conversion group sequence 1...6.

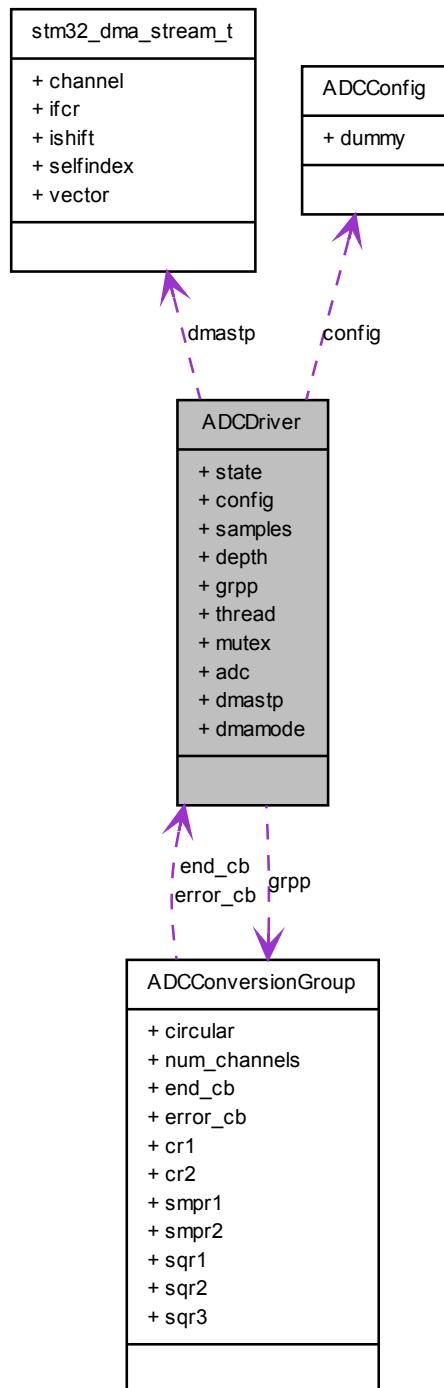
## 7.3 ADCDriver Struct Reference

### 7.3.1 Detailed Description

Structure representing an ADC driver.

```
#include <adc_lld.h>
```

Collaboration diagram for ADCDriver:



## Data Fields

- `adcstate_t state`

*Driver state.*

- const [ADCConfig](#) \* **config**  
*Current configuration data.*
- [adcsample\\_t](#) \* **samples**  
*Current samples buffer pointer or NULL.*
- [size\\_t](#) **depth**  
*Current samples buffer depth or 0.*
- const [ADCConversionGroup](#) \* **grpp**  
*Current conversion group pointer or NULL.*
- Thread \* **thread**  
*Waiting thread.*
- Mutex **mutex**  
*Mutex protecting the peripheral.*
- ADC\_TypeDef \* **adc**  
*Pointer to the ADCx registers block.*
- const [stm32\\_dma\\_stream\\_t](#) \* **dmastp**  
*Pointer to associated SMA channel.*
- uint32\_t **dmamode**  
*DMA mode bit mask.*

### 7.3.2 Field Documentation

#### 7.3.2.1 [adcstate\\_t](#) ADCDriver::state

Driver state.

#### 7.3.2.2 const [ADCConfig](#)\* ADCDriver::config

Current configuration data.

#### 7.3.2.3 [adcsample\\_t](#)\* ADCDriver::samples

Current samples buffer pointer or NULL.

#### 7.3.2.4 [size\\_t](#) ADCDriver::depth

Current samples buffer depth or 0.

#### 7.3.2.5 const [ADCConversionGroup](#)\* ADCDriver::grpp

Current conversion group pointer or NULL.

#### 7.3.2.6 Thread\* ADCDriver::thread

Waiting thread.

#### 7.3.2.7 Mutex ADCDriver::mutex

Mutex protecting the peripheral.

7.3.2.8 `ADC_TypeDef* ADCDriver::adc`

Pointer to the ADCx registers block.

7.3.2.9 `const stm32_dma_stream_t* ADCDriver::dmastp`

Pointer to associated SMA channel.

7.3.2.10 `uint32_t ADCDriver::dmamode`

DMA mode bit mask.

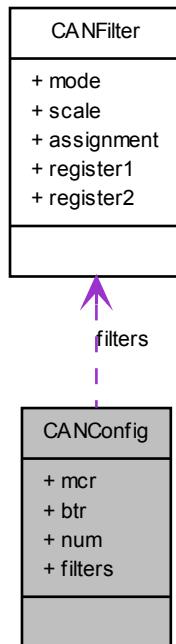
## 7.4 CANConfig Struct Reference

### 7.4.1 Detailed Description

Driver configuration structure.

```
#include <can_ll.h>
```

Collaboration diagram for CANConfig:



### Data Fields

- `uint32_t mcr`

*CAN MCR register initialization data.*

- `uint32_t btr`  
*CAN BTR register initialization data.*
- `uint32_t num`  
*Number of elements into the filters array.*
- `const CANFilter * filters`  
*Pointer to an array of `CANFilter` structures.*

## 7.4.2 Field Documentation

### 7.4.2.1 `uint32_t CANConfig::mcr`

CAN MCR register initialization data.

#### Note

Some bits in this register are enforced by the driver regardless their status in this field.

### 7.4.2.2 `uint32_t CANConfig::btr`

CAN BTR register initialization data.

#### Note

Some bits in this register are enforced by the driver regardless their status in this field.

### 7.4.2.3 `uint32_t CANConfig::num`

Number of elements into the filters array.

#### Note

By setting this field to zero a default filter is enabled that allows all frames, this should be adequate for simple applications.

### 7.4.2.4 `const CANFilter* CANConfig::filters`

Pointer to an array of `CANFilter` structures.

#### Note

This field can be set to `NULL` if the field `num` is set to zero.

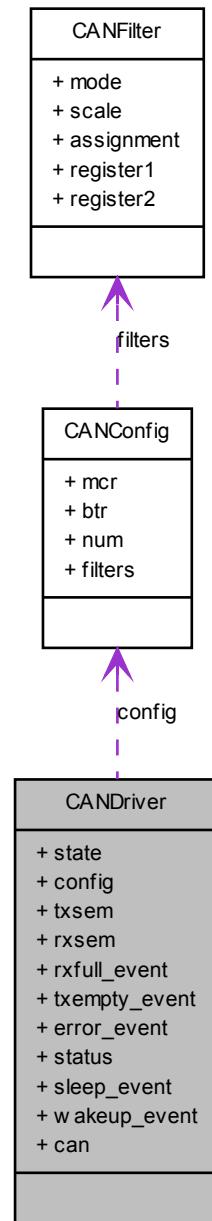
## 7.5 CANDriver Struct Reference

### 7.5.1 Detailed Description

Structure representing an CAN driver.

```
#include <can_lld.h>
```

Collaboration diagram for CANDriver:



## Data Fields

- `canstate_t state`  
*Driver state.*
- `const CANConfig * config`  
*Current configuration data.*
- Semaphore `txsem`  
*Transmission queue semaphore.*

- Semaphore [rxsem](#)  
*Receive queue semaphore.*
- EventSource [rxfull\\_event](#)  
*One or more frames become available.*
- EventSource [txempty\\_event](#)  
*One or more transmission slots become available.*
- EventSource [error\\_event](#)  
*A CAN bus error happened.*
- [canstatus\\_t status](#)  
*Error flags set when an error event is broadcasted.*
- EventSource [sleep\\_event](#)  
*Entering sleep state event.*
- EventSource [wakeup\\_event](#)  
*Exiting sleep state event.*
- CAN\_TypeDef \* [can](#)  
*Pointer to the CAN registers.*

## 7.5.2 Field Documentation

### 7.5.2.1 [canstate\\_t CANDriver::state](#)

Driver state.

### 7.5.2.2 [const CANConfig\\* CANDriver::config](#)

Current configuration data.

### 7.5.2.3 [Semaphore CANDriver::txsem](#)

Transmission queue semaphore.

### 7.5.2.4 [Semaphore CANDriver::rxsem](#)

Receive queue semaphore.

### 7.5.2.5 [EventSource 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.

### 7.5.2.6 [EventSource CANDriver::txempty\\_event](#)

One or more transmission slots become available.

### 7.5.2.7 EventSource CANDriver::error\_event

A CAN bus error happened.

### 7.5.2.8 canstatus\_t CANDriver::status

Error flags set when an error event is broadcasted.

### 7.5.2.9 EventSource CANDriver::sleep\_event

Entering sleep state event.

### 7.5.2.10 EventSource CANDriver::wakeup\_event

Exiting sleep state event.

### 7.5.2.11 CAN\_TypeDef\* CANDriver::can

Pointer to the CAN registers.

## 7.6 CANFilter Struct Reference

### 7.6.1 Detailed Description

CAN filter.

#### Note

Refer to the STM32 reference manual for info about filters.

```
#include <can_ll.h>
```

#### Data Fields

- uint32\_t **mode**:1  
*Filter mode.*
- uint32\_t **scale**:1  
*Filter scale.*
- uint32\_t **assignment**:1  
*Filter mode.*
- uint32\_t **register1**  
*Filter register 1 (identifier).*
- uint32\_t **register2**  
*Filter register 2 (mask/identifier depending on mode=0/1).*

### 7.6.2 Field Documentation

#### 7.6.2.1 uint32\_t CANFilter::mode

Filter mode.

**Note**

This bit represent the CAN\_FM1R register bit associated to this filter (0=mask mode, 1=list mode).

**7.6.2.2 uint32\_t CANFilter::scale**

Filter scale.

**Note**

This bit represent the CAN\_FS1R register bit associated to this filter (0=16 bits mode, 1=32 bits mode).

**7.6.2.3 uint32\_t CANFilter::assignment**

Filter mode.

**Note**

This bit represent the CAN\_FFA1R register bit associated to this filter, must be set to zero in this version of the driver.

**7.6.2.4 uint32\_t CANFilter::register1**

Filter register 1 (identifier).

**7.6.2.5 uint32\_t CANFilter::register2**

Filter register 2 (mask/identifier depending on mode=0/1).

## 7.7 CANRxFrame Struct Reference

### 7.7.1 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.

```
#include <can_lld.h>
```

### 7.7.2 Field Documentation

**7.7.2.1 uint8\_t CANRxFrame::FMI**

Filter id.

**7.7.2.2 uint16\_t CANRxFrame::TIME**

Time stamp.

**7.7.2.3 uint8\_t CANRxFrame::DLC**

Data length.

**7.7.2.4 uint8\_t CANRxFrame::RTR**

Frame type.

**7.7.2.5 uint8\_t CANRxFrame::IDE**

Identifier type.

**7.7.2.6 uint32\_t CANRxFrame::SID**

Standard identifier.

**7.7.2.7 uint32\_t CANRxFrame::EID**

Extended identifier.

**7.7.2.8 uint8\_t CANRxFrame::data8[8]**

Frame data.

**7.7.2.9 uint16\_t CANRxFrame::data16[4]**

Frame data.

**7.7.2.10 uint32\_t CANRxFrame::data32[2]**

Frame data.

## 7.8 CANTxFrame Struct Reference

### 7.8.1 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.

```
#include <can_ll.h>
```

### 7.8.2 Field Documentation

**7.8.2.1 uint8\_t CANTxFrame::DLC**

Data length.

**7.8.2.2 uint8\_t CANTxFrame::RTR**

Frame type.

**7.8.2.3 uint8\_t CANTxFrame::IDE**

Identifier type.

**7.8.2.4 uint32\_t CANTxFrame::SID**

Standard identifier.

**7.8.2.5 uint32\_t CANTxFrame::EID**

Extended identifier.

**7.8.2.6 uint8\_t CANTxFrame::data8[8]**

Frame data.

**7.8.2.7 uint16\_t CANTxFrame::data16[4]**

Frame data.

**7.8.2.8 uint32\_t CANTxFrame::data32[2]**

Frame data.

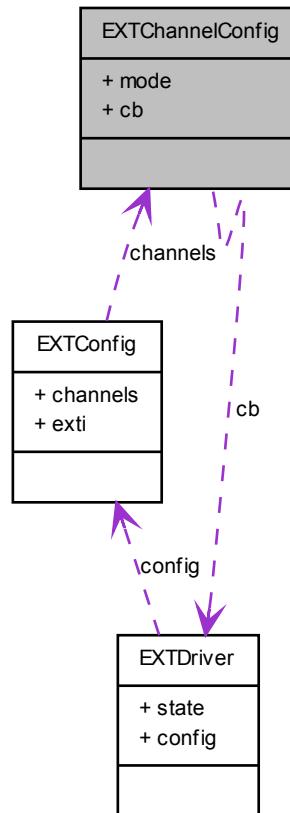
## 7.9 EXTChannelConfig Struct Reference

### 7.9.1 Detailed Description

Channel configuration structure.

```
#include <ext_lld.h>
```

Collaboration diagram for EXTChannelConfig:



## Data Fields

- `uint32_t mode`  
*Channel mode.*
- `extcallback_t cb`  
*Channel callback.*

### 7.9.2 Field Documentation

#### 7.9.2.1 `uint32_t EXTChannelConfig::mode`

Channel mode.

#### 7.9.2.2 `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.

## 7.10 EXTConfig Struct Reference

### 7.10.1 Detailed Description

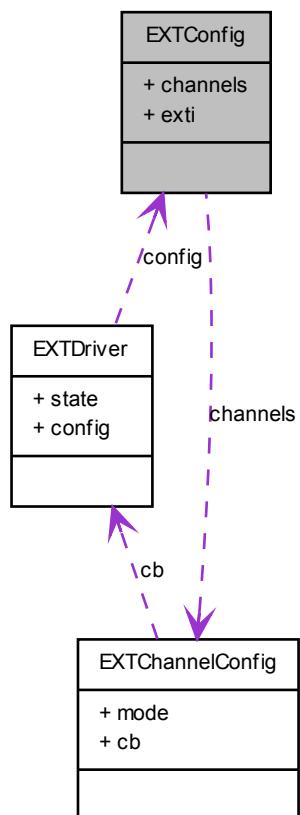
Driver configuration structure.

#### Note

It could be empty on some architectures.

```
#include <ext_lld.h>
```

Collaboration diagram for EXTConfig:



### Data Fields

- **EXTChannelConfig channels [EXT\_MAX\_CHANNELS]**  
*Channel configurations.*
- **uint16\_t exti [4]**  
*Initialization values for EXTICRx registers.*

### 7.10.2 Field Documentation

## 7.10.2.1 EXTChannelConfig EXTConfig::channels[EXT\_MAX\_CHANNELS]

Channel configurations.

## 7.10.2.2 uint16\_t EXTConfig::exti[4]

Initialization values for EXTICRx registers.

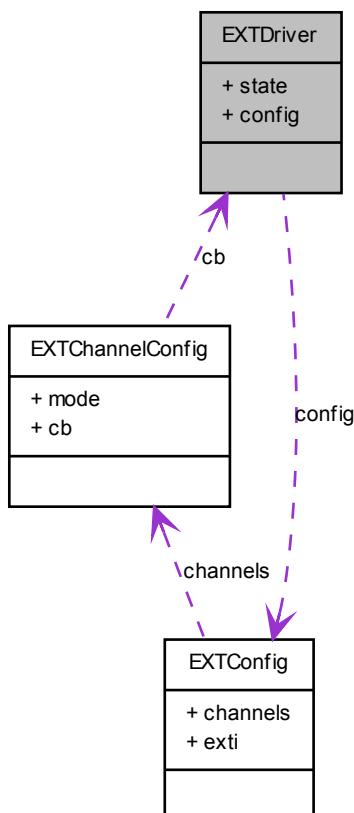
## 7.11 EXTDriver Struct Reference

### 7.11.1 Detailed Description

Structure representing an EXT driver.

```
#include <ext_lld.h>
```

Collaboration diagram for EXTDriver:



### Data Fields

- extstate\_t **state**

*Driver state.*

- const EXTConfig \* config  
*Current configuration data.*

## 7.11.2 Field Documentation

### 7.11.2.1 extstate\_t EXTDriver::state

Driver state.

### 7.11.2.2 const EXTConfig\* EXTDriver::config

Current configuration data.

## 7.12 GPTConfig Struct Reference

### 7.12.1 Detailed Description

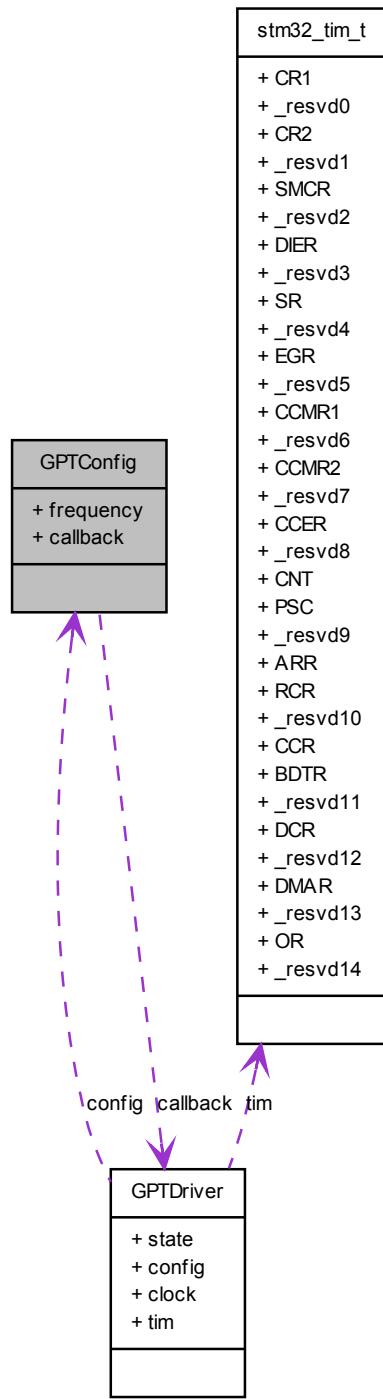
Driver configuration structure.

#### Note

It could be empty on some architectures.

```
#include <gpt_lld.h>
```

Collaboration diagram for GPTConfig:



## Data Fields

- `gptfreq_t frequency`

*Timer clock in Hz.*

- `gptcallback_t callback`

*Timer callback pointer.*

## 7.12.2 Field Documentation

### 7.12.2.1 `gptfreq_t GPTConfig::frequency`

Timer clock in Hz.

#### Note

The low level can use assertions in order to catch invalid frequency specifications.

### 7.12.2.2 `gptcallback_t GPTConfig::callback`

Timer callback pointer.

#### Note

This callback is invoked on GPT counter events.

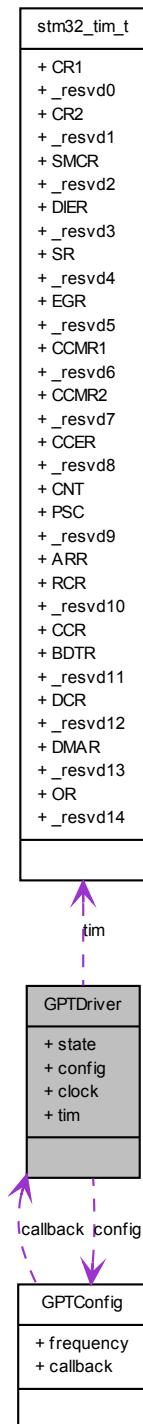
## 7.13 GPTDriver Struct Reference

### 7.13.1 Detailed Description

Structure representing a GPT driver.

```
#include <gpt_lld.h>
```

Collaboration diagram for GPTDriver:



## Data Fields

- [gptstate\\_t state](#)

*Driver state.*

- const [GPTConfig](#) \* **config**  
*Current configuration data.*
- uint32\_t **clock**  
*Timer base clock.*
- [stm32\\_tim\\_t](#) \* **tim**  
*Pointer to the TIMx registers block.*

## 7.13.2 Field Documentation

### 7.13.2.1 [gptstate\\_t](#) GPTDriver::state

Driver state.

### 7.13.2.2 const [GPTConfig](#)\* GPTDriver::config

Current configuration data.

### 7.13.2.3 uint32\_t GPTDriver::clock

Timer base clock.

### 7.13.2.4 [stm32\\_tim\\_t](#)\* GPTDriver::tim

Pointer to the TIMx registers block.

## 7.14 I2CConfig Struct Reference

### 7.14.1 Detailed Description

Driver configuration structure.

```
#include <i2c_llld.h>
```

### Data Fields

- [i2copmode\\_t](#) **op\_mode**  
*Specifies the I2C mode.*
- uint32\_t **clock\_speed**  
*Specifies the clock frequency.*
- [i2cdutycycle\\_t](#) **duty\_cycle**  
*Specifies the I2C fast mode duty cycle.*

## 7.14.2 Field Documentation

### 7.14.2.1 [i2copmode\\_t](#) I2CConfig::op\_mode

Specifies the I2C mode.

7.14.2.2 `uint32_t I2CConfig::clock_speed`

Specifies the clock frequency.

**Note**

Must be set to a value lower than 400kHz.

7.14.2.3 `i2cdutycycle_t I2CConfig::duty_cycle`

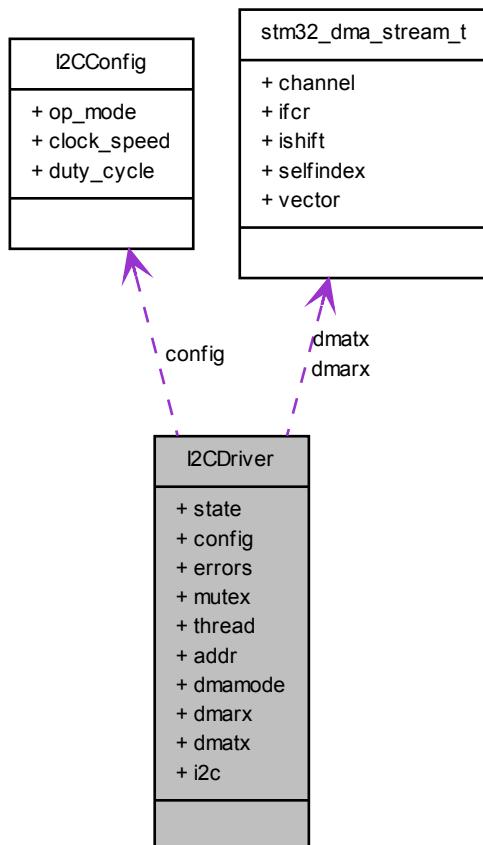
Specifies the I2C fast mode duty cycle.

**7.15 I2CDriver Struct Reference****7.15.1 Detailed Description**

Structure representing an I2C driver.

```
#include <i2c_llld.h>
```

Collaboration diagram for I2CDriver:



## Data Fields

- **i2cstate\_t state**  
*Driver state.*
- **const I2CConfig \* config**  
*Current configuration data.*
- **i2cflags\_t errors**  
*Error flags.*
- **Mutex mutex**  
*Mutex protecting the bus.*
- **Thread \* thread**  
*Thread waiting for I/O completion.*
- **i2caddr\_t addr**  
*Current slave address without R/W bit.*
- **uint32\_t dmemode**  
*DMA mode bit mask.*
- **const stm32\_dma\_stream\_t \* dmarx**  
*Receive DMA channel.*
- **const stm32\_dma\_stream\_t \* dmatx**  
*Transmit DMA channel.*
- **I2C\_TypeDef \* i2c**  
*Pointer to the I2Cx registers block.*

### 7.15.2 Field Documentation

#### 7.15.2.1 i2cstate\_t I2CDriver::state

Driver state.

#### 7.15.2.2 const I2CConfig\* I2CDriver::config

Current configuration data.

#### 7.15.2.3 i2cflags\_t I2CDriver::errors

Error flags.

#### 7.15.2.4 Mutex I2CDriver::mutex

Mutex protecting the bus.

#### 7.15.2.5 Thread\* I2CDriver::thread

Thread waiting for I/O completion.

#### 7.15.2.6 i2caddr\_t I2CDriver::addr

Current slave address without R/W bit.

7.15.2.7 `uint32_t I2CDriver::dmemode`

DMA mode bit mask.

7.15.2.8 `const stm32_dma_stream_t* I2CDriver::dmาร์ก`

Receive DMA channel.

7.15.2.9 `const stm32_dma_stream_t* I2CDriver::dmاترخ`

Transmit DMA channel.

7.15.2.10 `I2C_TypeDef* I2CDriver::i2c`

Pointer to the I2Cx registers block.

## 7.16 ICUConfig Struct Reference

### 7.16.1 Detailed Description

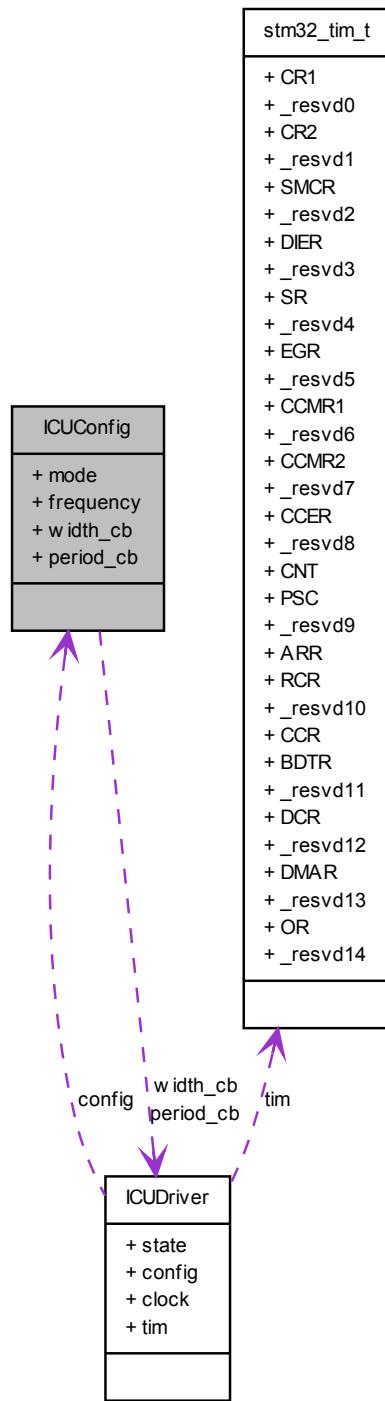
Driver configuration structure.

#### Note

It could be empty on some architectures.

```
#include <icu_ll.h>
```

Collaboration diagram for ICUConfig:



## Data Fields

- `icemode_t mode`

*Driver mode.*

- **icufreq\_t frequency**  
*Timer clock in Hz.*
- **icucallback\_t width\_cb**  
*Callback for pulse width measurement.*
- **icucallback\_t period\_cb**  
*Callback for cycle period measurement.*

## 7.16.2 Field Documentation

### 7.16.2.1 icumode\_t ICUConfig::mode

Driver mode.

### 7.16.2.2 icufreq\_t ICUConfig::frequency

Timer clock in Hz.

#### Note

The low level can use assertions in order to catch invalid frequency specifications.

### 7.16.2.3 icucallback\_t ICUConfig::width\_cb

Callback for pulse width measurement.

### 7.16.2.4 icucallback\_t ICUConfig::period\_cb

Callback for cycle period measurement.

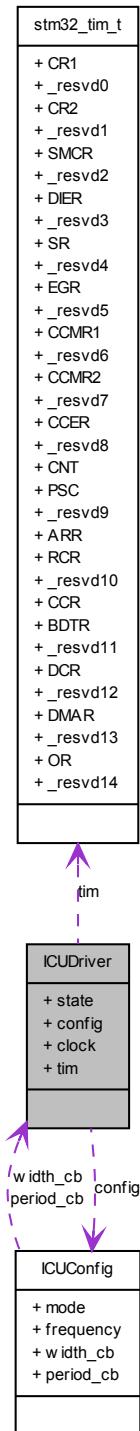
## 7.17 ICUDriver Struct Reference

### 7.17.1 Detailed Description

Structure representing an ICU driver.

```
#include <icu_lld.h>
```

Collaboration diagram for ICUDriver:



## Data Fields

- `icustate_t state`

*Driver state.*

- const [ICUConfig](#) \* config  
*Current configuration data.*
- uint32\_t [clock](#)  
*Timer base clock.*
- [stm32\\_tim\\_t](#) \* [tim](#)  
*Pointer to the TIMx registers block.*

## 7.17.2 Field Documentation

### 7.17.2.1 [icustate\\_t](#) ICUDriver::state

Driver state.

### 7.17.2.2 const [ICUConfig](#)\* ICUDriver::config

Current configuration data.

### 7.17.2.3 uint32\_t ICUDriver::clock

Timer base clock.

### 7.17.2.4 [stm32\\_tim\\_t](#)\* ICUDriver::tim

Pointer to the TIMx registers block.

## 7.18 IOBus Struct Reference

### 7.18.1 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.

```
#include <pal.h>
```

#### 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.*

## 7.18.2 Field Documentation

### 7.18.2.1 ioportid\_t IOBus::portid

Port identifier.

### 7.18.2.2 ioportmask\_t IOBus::mask

Bus mask aligned to port bit 0.

#### Note

The bus mask implicitly define the bus width. A logical AND is performed on the bus data.

### 7.18.2.3 uint\_fast8\_t IOBus::offset

Offset, within the port, of the least significant bit of the bus.

## 7.19 MMCConfig Struct Reference

### 7.19.1 Detailed Description

Driver configuration structure.

#### Note

Not required in the current implementation.

```
#include <mmc_spi.h>
```

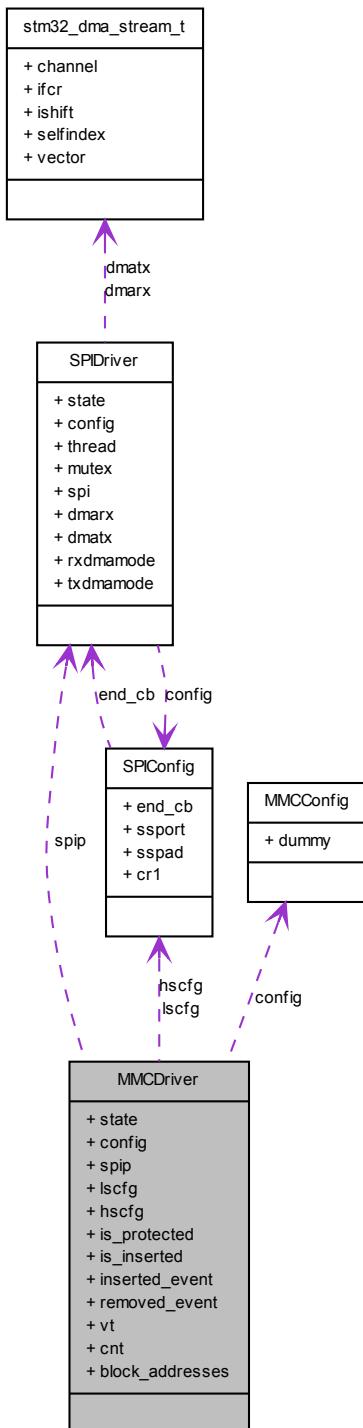
## 7.20 MMCDriver Struct Reference

### 7.20.1 Detailed Description

Structure representing a MMC driver.

```
#include <mmc_spi.h>
```

Collaboration diagram for MMCDriver:



## Data Fields

- [mmcstate\\_t state](#)

*Driver state.*

- const [MMCConfig](#) \* config  
*Current configuration data.*
- [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.*
- [mmcquery\\_t](#) is\_protected  
*Write protect status query function.*
- [mmcquery\\_t](#) is\_inserted  
*Insertion status query function.*
- EventSource inserted\_event  
*Card insertion event source.*
- EventSource removed\_event  
*Card removal event source.*
- VirtualTimer vt  
*MMC insertion polling timer.*
- uint\_fast8\_t cnt  
*Insertion counter.*

## 7.20.2 Field Documentation

### 7.20.2.1 mmcstate\_t MMCDriver::state

Driver state.

### 7.20.2.2 const MMCConfig\* MMCDriver::config

Current configuration data.

### 7.20.2.3 SPIDriver\* MMCDriver::spip

SPI driver associated to this MMC driver.

### 7.20.2.4 const SPIConfig\* MMCDriver::lscfg

SPI low speed configuration used during initialization.

### 7.20.2.5 const SPIConfig\* MMCDriver::hscfg

SPI high speed configuration used during transfers.

### 7.20.2.6 mmcquery\_t MMCDriver::is\_protected

Write protect status query function.

### 7.20.2.7 mmcquery\_t MMCDriver::is\_inserted

Insertion status query function.

### 7.20.2.8 EventSource MMCDriver::inserted\_event

Card insertion event source.

### 7.20.2.9 EventSource MMCDriver::removed\_event

Card removal event source.

### 7.20.2.10 VirtualTimer MMCDriver::vt

MMC insertion polling timer.

### 7.20.2.11 uint\_fast8\_t MMCDriver::cnt

Insertion counter.

## 7.21 PALConfig Struct Reference

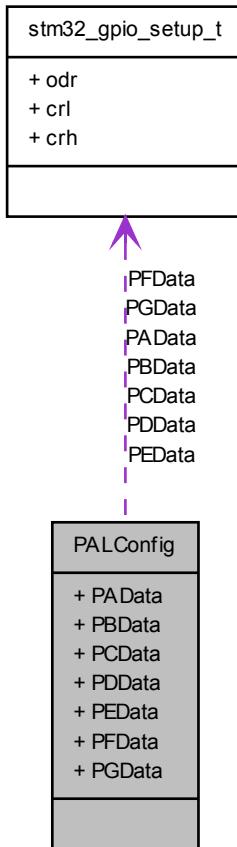
### 7.21.1 Detailed Description

STM32 GPIO 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.

```
#include <pal_lld.h>
```

Collaboration diagram for PALConfig:



## Data Fields

- [`stm32\_gpio\_setup\_t` PAData](#)  
*Port A setup data.*
- [`stm32\_gpio\_setup\_t` PBData](#)  
*Port B setup data.*
- [`stm32\_gpio\_setup\_t` PCData](#)  
*Port C setup data.*
- [`stm32\_gpio\_setup\_t` PDData](#)  
*Port D setup data.*
- [`stm32\_gpio\_setup\_t` PEData](#)  
*Port E setup data.*
- [`stm32\_gpio\_setup\_t` PFData](#)  
*Port F setup data.*
- [`stm32\_gpio\_setup\_t` PGData](#)  
*Port G setup data.*

## 7.21.2 Field Documentation

### 7.21.2.1 `stm32_gpio_setup_t PALConfig::PAData`

Port A setup data.

### 7.21.2.2 `stm32_gpio_setup_t PALConfig::PBData`

Port B setup data.

### 7.21.2.3 `stm32_gpio_setup_t PALConfig::PCData`

Port C setup data.

### 7.21.2.4 `stm32_gpio_setup_t PALConfig::PDData`

Port D setup data.

### 7.21.2.5 `stm32_gpio_setup_t PALConfig::PEData`

Port E setup data.

### 7.21.2.6 `stm32_gpio_setup_t PALConfig::PFData`

Port F setup data.

### 7.21.2.7 `stm32_gpio_setup_t PALConfig::PGData`

Port G setup data.

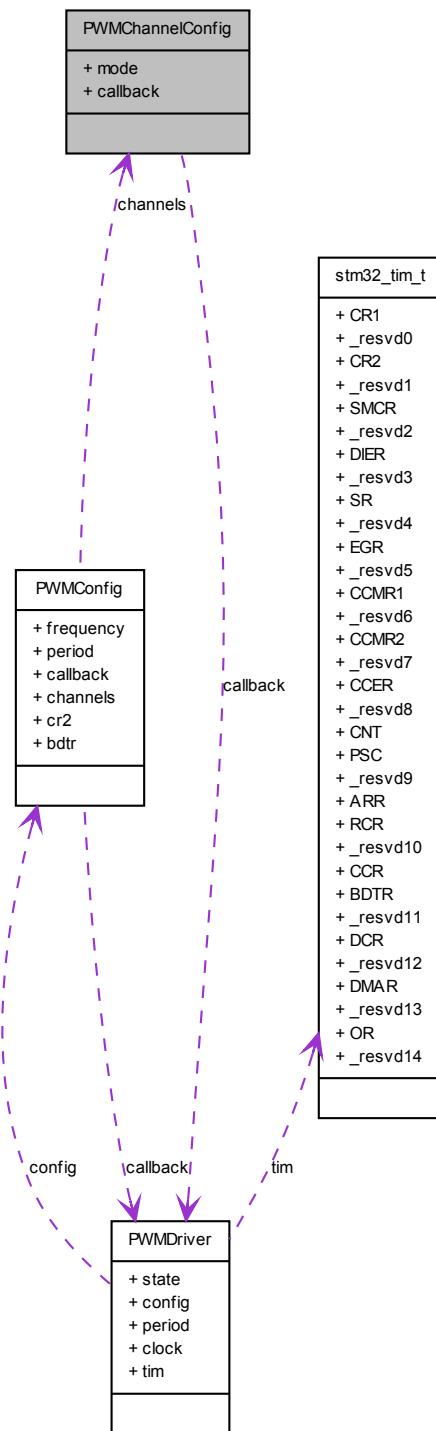
## 7.22 PWMChannelConfig Struct Reference

### 7.22.1 Detailed Description

PWM driver channel configuration structure.

```
#include <pwm_lld.h>
```

Collaboration diagram for PWMChannelConfig:



## Data Fields

- `pwmmode_t mode`

*Channel active logic level.*

- `pwmcallback_t callback`  
*Channel callback pointer.*

## 7.22.2 Field Documentation

### 7.22.2.1 pwmmode\_t PWMChannelConfig::mode

Channel active logic level.

### 7.22.2.2 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.

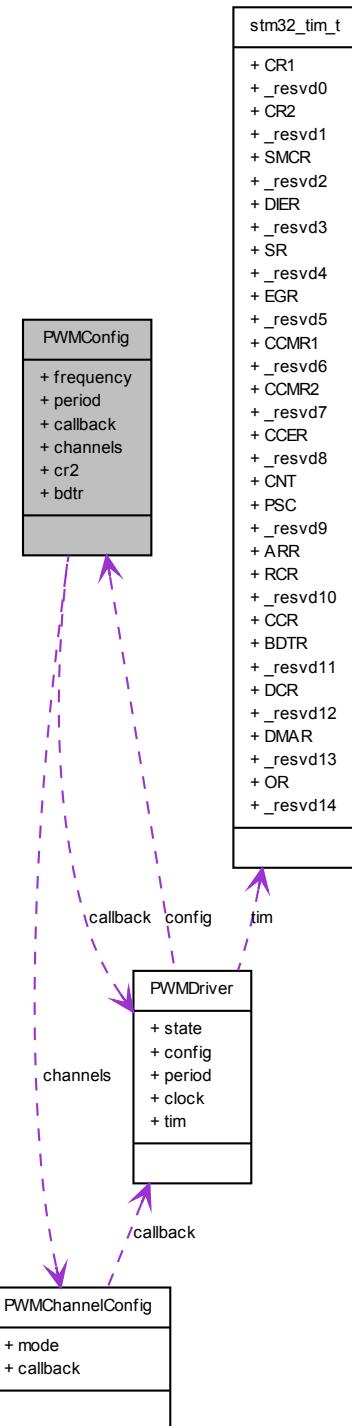
## 7.23 PWMConfig Struct Reference

### 7.23.1 Detailed Description

PWM driver configuration structure.

```
#include <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.*
- `uint16_t cr2`  
*TIM CR2 register initialization data.*
- `uint16_t bdtr`  
*TIM BDTR (break & dead-time) register initialization data.*

## 7.23.2 Field Documentation

### 7.23.2.1 uint32\_t PWMConfig::frequency

Timer clock in Hz.

#### Note

The low level can use assertions in order to catch invalid frequency specifications.

### 7.23.2.2 pwmcnt\_t PWMConfig::period

PWM period in ticks.

#### Note

The low level can use assertions in order to catch invalid period specifications.

### 7.23.2.3 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.

### 7.23.2.4 PWMChannelConfig PWMConfig::channels[PWM\_CHANNELS]

Channels configurations.

### 7.23.2.5 uint16\_t PWMConfig::cr2

TIM CR2 register initialization data.

#### Note

The value of this field should normally be equal to zero.

### 7.23.2.6 uint16\_t PWMConfig::bdtr

TIM BDTR (break & dead-time) register initialization data.

#### Note

The value of this field should normally be equal to zero.

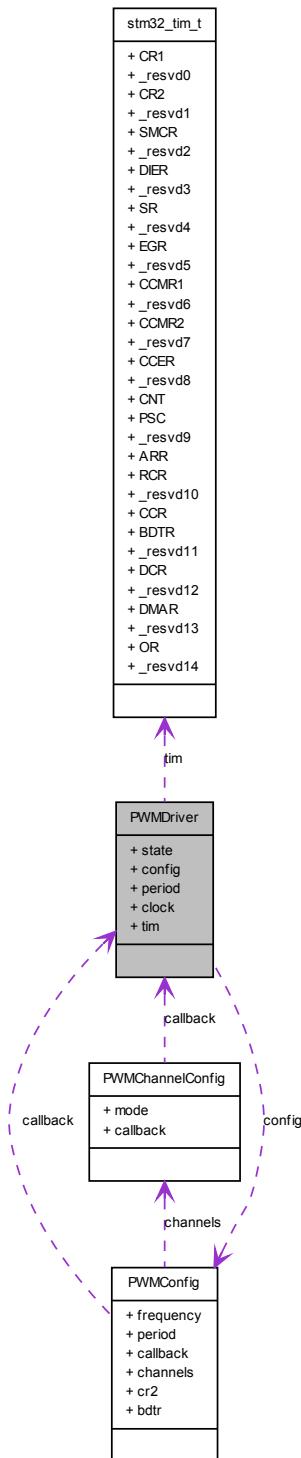
## 7.24 PWMDriver Struct Reference

### 7.24.1 Detailed Description

Structure representing a PWM driver.

```
#include <pwm_llld.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.*
- [uint32\\_t](#) clock  
*Timer base clock.*
- [stm32\\_tim\\_t](#) \* tim  
*Pointer to the TIMx registers block.*

## 7.24.2 Field Documentation

### 7.24.2.1 [pwmstate\\_t](#) PWMDriver::state

Driver state.

### 7.24.2.2 const [PWMConfig](#)\* PWMDriver::config

Current driver configuration data.

### 7.24.2.3 [pwmcnt\\_t](#) PWMDriver::period

Current PWM period in ticks.

### 7.24.2.4 [uint32\\_t](#) PWMDriver::clock

Timer base clock.

### 7.24.2.5 [stm32\\_tim\\_t](#)\* PWMDriver::tim

Pointer to the TIMx registers block.

## 7.25 RTCAlarm Struct Reference

### 7.25.1 Detailed Description

Structure representing an RTC alarm time stamp.

```
#include <rtc_ll.h>
```

### Data Fields

- [uint32\\_t](#) tv\_sec  
*Seconds since UNIX epoch.*

## 7.25.2 Field Documentation

### 7.25.2.1 [uint32\\_t](#) RTCAlarm::tv\_sec

Seconds since UNIX epoch.

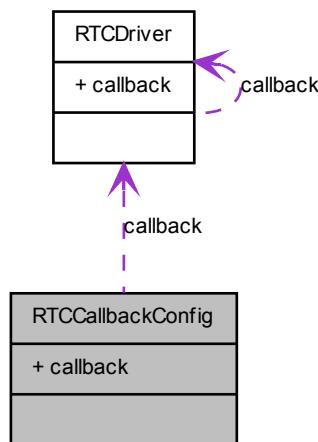
## 7.26 RTCCallbackConfig Struct Reference

### 7.26.1 Detailed Description

Structure representing an RTC callbacks config.

```
#include <rtc_lld.h>
```

Collaboration diagram for RTCCallbackConfig:



### Data Fields

- `rtccb_t callback`

*Generic RTC callback pointer.*

### 7.26.2 Field Documentation

#### 7.26.2.1 `rtccb_t RTCCallbackConfig::callback`

Generic RTC callback pointer.

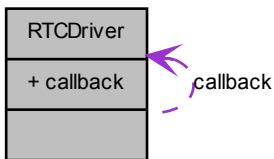
## 7.27 RTCDriver Struct Reference

### 7.27.1 Detailed Description

Structure representing an RTC driver.

```
#include <rtc_lld.h>
```

Collaboration diagram for RTCDriver:



## Data Fields

- `rtccb_t callback`

*Callback pointer.*

### 7.27.2 Field Documentation

#### 7.27.2.1 `rtccb_t RTCDriver::callback`

Callback pointer.

## 7.28 RTCTime Struct Reference

### 7.28.1 Detailed Description

Structure representing an RTC time stamp.

```
#include <rtc_lld.h>
```

## Data Fields

- `uint32_t tv_sec`

*Seconds since UNIX epoch.*

- `uint32_t tv_nsec`

*Fractional part.*

### 7.28.2 Field Documentation

#### 7.28.2.1 `uint32_t RTCTime::tv_sec`

Seconds since UNIX epoch.

#### 7.28.2.2 `uint32_t RTCTime::tv_nsec`

Fractional part.

## 7.29 SDCCConfig Struct Reference

### 7.29.1 Detailed Description

Driver configuration structure.

#### Note

It could be empty on some architectures.

```
#include <sdc_lld.h>
```

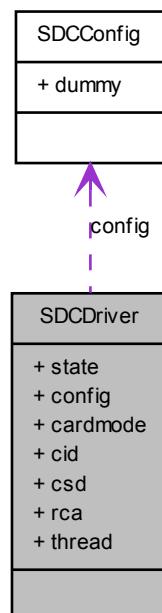
## 7.30 SDCDriver Struct Reference

### 7.30.1 Detailed Description

Structure representing an SDC driver.

```
#include <sdc_lld.h>
```

Collaboration diagram for SDCDriver:



### Data Fields

- `sdcstate_t state`  
*Driver state.*
- `const SDCCConfig * config`  
*Current configuration data.*

- `sdcmode_t cardmode`  
*Various flags regarding the mounted card.*
- `uint32_t cid [4]`  
*Card CID.*
- `uint32_t csd [4]`  
*Card CSD.*
- `uint32_t rca`  
*Card RCA.*
- `Thread * thread`  
*Thread waiting for I/O completion IRQ.*

## 7.30.2 Field Documentation

### 7.30.2.1 `sdcstate_t SDCDriver::state`

Driver state.

### 7.30.2.2 `const SDCCConfig* SDCDriver::config`

Current configuration data.

### 7.30.2.3 `sdcmode_t SDCDriver::cardmode`

Various flags regarding the mounted card.

### 7.30.2.4 `uint32_t SDCDriver::cid[4]`

Card CID.

### 7.30.2.5 `uint32_t SDCDriver::csd[4]`

Card CSD.

### 7.30.2.6 `uint32_t SDCDriver::rca`

Card RCA.

### 7.30.2.7 `Thread* SDCDriver::thread`

Thread waiting for I/O completion IRQ.

## 7.31 SerialConfig Struct Reference

### 7.31.1 Detailed Description

STM32 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.

```
#include <serial_lld.h>
```

**Data Fields**

- `uint32_t sc_speed`  
*Bit rate.*
- `uint16_t sc_cr1`  
*Initialization value for the CR1 register.*
- `uint16_t sc_cr2`  
*Initialization value for the CR2 register.*
- `uint16_t sc_cr3`  
*Initialization value for the CR3 register.*

### 7.31.2 Field Documentation

#### 7.31.2.1 `uint32_t SerialConfig::sc_speed`

Bit rate.

#### 7.31.2.2 `uint16_t SerialConfig::sc_cr1`

Initialization value for the CR1 register.

#### 7.31.2.3 `uint16_t SerialConfig::sc_cr2`

Initialization value for the CR2 register.

#### 7.31.2.4 `uint16_t SerialConfig::sc_cr3`

Initialization value for the CR3 register.

## 7.32 SerialDriver Struct Reference

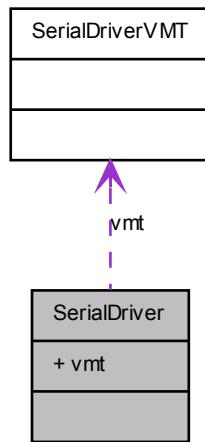
### 7.32.1 Detailed Description

Full duplex serial driver class.

This class extends `BaseAsynchronousChannel` by adding physical I/O queues.

```
#include <serial.h>
```

Collaboration diagram for SerialDriver:



## Data Fields

- struct [SerialDriverVMT](#) \* [vmt](#)  
*Virtual Methods Table.*

### 7.32.2 Field Documentation

#### 7.32.2.1 struct [SerialDriverVMT](#)\* [SerialDriver::vmt](#)

Virtual Methods Table.

## 7.33 SerialDriverVMT Struct Reference

### 7.33.1 Detailed Description

[SerialDriver](#) virtual methods table.

```
#include <serial.h>
```

## 7.34 SerialUSBConfig Struct Reference

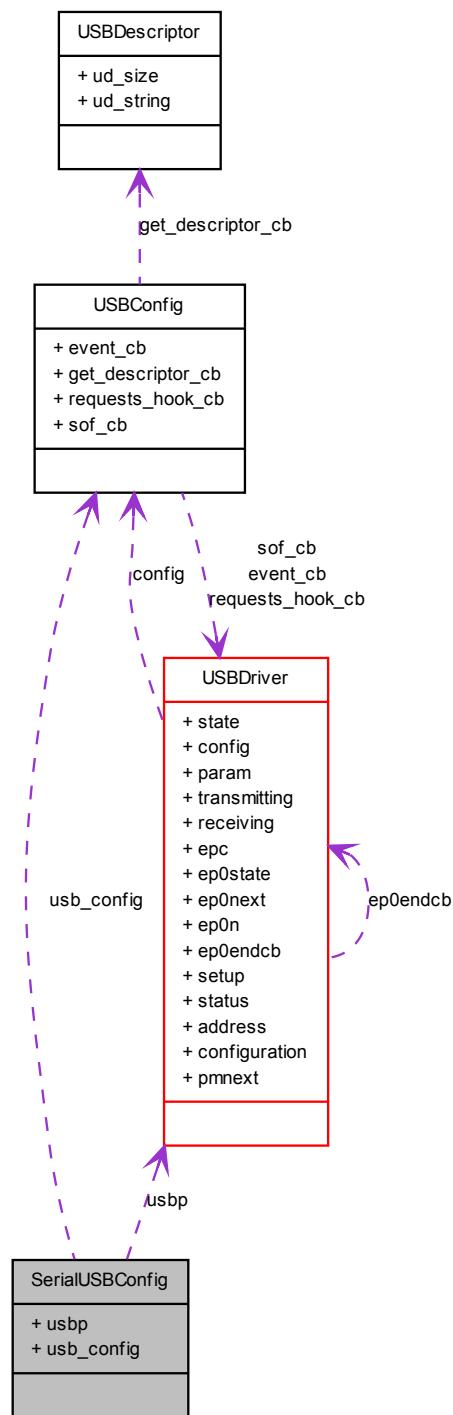
### 7.34.1 Detailed Description

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.

```
#include <serial_usb.h>
```

Collaboration diagram for SerialUSBConfig:



## Data Fields

- `USBDriver * usbp`

*USB driver to use.*

- **USBConfig usb\_config**  
*USB driver configuration structure.*

### 7.34.2 Field Documentation

#### 7.34.2.1 **USBDriver\* SerialUSBCConfig::usbp**

USB driver to use.

#### 7.34.2.2 **USBConfig SerialUSBCConfig::usb\_config**

USB driver configuration structure.

## 7.35 SerialUSBDriver Struct Reference

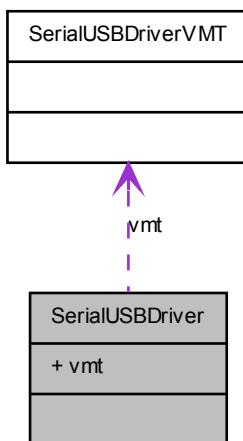
### 7.35.1 Detailed Description

Full duplex serial driver class.

This class extends `BaseAsynchronousChannel` by adding physical I/O queues.

```
#include <serial_usb.h>
```

Collaboration diagram for `SerialUSBDriver`:



### Data Fields

- struct [SerialUSBDriverVMT](#) \* **vmt**  
*Virtual Methods Table.*

### 7.35.2 Field Documentation

### 7.35.2.1 struct SerialUSBDriverVMT\* SerialUSBDriver::vmt

Virtual Methods Table.

## 7.36 SerialUSBDriverVMT Struct Reference

### 7.36.1 Detailed Description

[SerialDriver](#) virtual methods table.

```
#include <serial_usb.h>
```

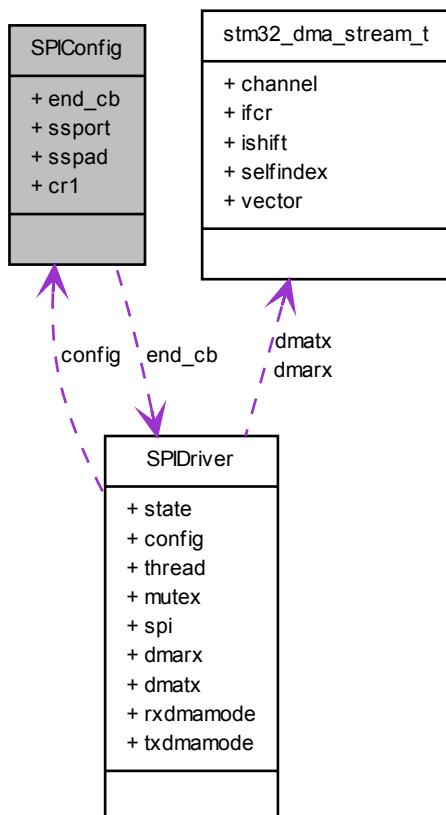
## 7.37 SPIConfig Struct Reference

### 7.37.1 Detailed Description

Driver configuration structure.

```
#include <spi_llld.h>
```

Collaboration diagram for SPIConfig:



## Data Fields

- **spicallback\_t end\_cb**  
*Operation complete callback or NULL.*
- **ioportid\_t ssport**  
*The chip select line port.*
- **uint16\_t sspad**  
*The chip select line pad number.*
- **uint16\_t cr1**  
*SPI initialization data.*

## 7.37.2 Field Documentation

### 7.37.2.1 spicallback\_t SPIConfig::end\_cb

Operation complete callback or NULL.

### 7.37.2.2 ioportid\_t SPIConfig::ssport

The chip select line port.

### 7.37.2.3 uint16\_t SPIConfig::sspad

The chip select line pad number.

### 7.37.2.4 uint16\_t SPIConfig::cr1

SPI initialization data.

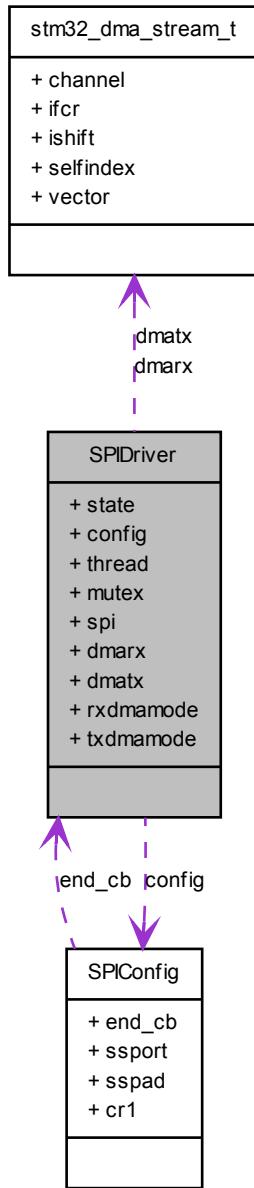
## 7.38 SPIDriver Struct Reference

### 7.38.1 Detailed Description

Structure representing a SPI driver.

```
#include <spi_llld.h>
```

Collaboration diagram for SPIDriver:



## Data Fields

- `spistate_t state`  
*Driver state.*
- `const SPIConfig * config`  
*Current configuration data.*
- `Thread * thread`  
*Waiting thread.*
- `Mutex mutex`

- **SPI\_TypeDef \* spi**  
*Pointer to the SPIx registers block.*
- **const stm32\_dma\_stream\_t \* dmarx**  
*Receive DMA stream.*
- **const stm32\_dma\_stream\_t \* dmatx**  
*Transmit DMA stream.*
- **uint32\_t rxdmamode**  
*RX DMA mode bit mask.*
- **uint32\_t txdmamode**  
*TX DMA mode bit mask.*

## 7.38.2 Field Documentation

### 7.38.2.1 spistate\_t SPIDriver::state

Driver state.

### 7.38.2.2 const SPIConfig\* SPIDriver::config

Current configuration data.

### 7.38.2.3 Thread\* SPIDriver::thread

Waiting thread.

### 7.38.2.4 Mutex SPIDriver::mutex

Mutex protecting the bus.

### 7.38.2.5 SPI\_TypeDef\* SPIDriver::spi

Pointer to the SPIx registers block.

### 7.38.2.6 const stm32\_dma\_stream\_t\* SPIDriver::dmarx

Receive DMA stream.

### 7.38.2.7 const stm32\_dma\_stream\_t\* SPIDriver::dmatx

Transmit DMA stream.

### 7.38.2.8 uint32\_t SPIDriver::rxdmamode

RX DMA mode bit mask.

### 7.38.2.9 uint32\_t SPIDriver::txdmamode

TX DMA mode bit mask.

## 7.39 `stm32_dma_stream_t` Struct Reference

### 7.39.1 Detailed Description

STM32 DMA stream descriptor structure.

```
#include <stm32_dma.h>
```

### Data Fields

- `DMA_Channel_TypeDef * channel`  
*Associated DMA channel.*
- `volatile uint32_t * ifcr`  
*Associated IFCR reg.*
- `uint8_t ishift`  
*Bits offset in xIFCR register.*
- `uint8_t selfindex`  
*Index to self in array.*
- `uint8_t vector`  
*Associated IRQ vector.*

### 7.39.2 Field Documentation

#### 7.39.2.1 `DMA_Channel_TypeDef* stm32_dma_stream_t::channel`

Associated DMA channel.

#### 7.39.2.2 `volatile uint32_t* stm32_dma_stream_t::ifcr`

Associated IFCR reg.

#### 7.39.2.3 `uint8_t stm32_dma_stream_t::ishift`

Bits offset in xIFCR register.

#### 7.39.2.4 `uint8_t stm32_dma_stream_t::selfindex`

Index to self in array.

#### 7.39.2.5 `uint8_t stm32_dma_stream_t::vector`

Associated IRQ vector.

## 7.40 `stm32_gpio_setup_t` Struct Reference

### 7.40.1 Detailed Description

GPIO port setup info.

```
#include <pal_lld.h>
```

## Data Fields

- `uint32_t odr`
- `uint32_t crl`
- `uint32_t crh`

### 7.40.2 Field Documentation

#### 7.40.2.1 `uint32_t stm32_gpio_setup_t::odr`

Initial value for ODR register.

#### 7.40.2.2 `uint32_t stm32_gpio_setup_t::crl`

Initial value for CRL register.

#### 7.40.2.3 `uint32_t stm32_gpio_setup_t::crh`

Initial value for CRH register.

## 7.41 `stm32_tim_t` Struct Reference

### 7.41.1 Detailed Description

STM32 TIM registers block.

#### Note

Redefined from the ST headers because the non uniform declaration of the CCR registers among the various sub-families.

```
#include <stm32.h>
```

## 7.42 `stm32_usb_descriptor_t` Struct Reference

### 7.42.1 Detailed Description

USB descriptor registers block.

```
#include <stm32_usb.h>
```

## Data Fields

- `volatile uint32_t TXADDR0`  
*TX buffer offset register.*
- `volatile uint16_t TXCOUNT0`  
*TX counter register 0.*
- `volatile uint16_t TXCOUNT1`  
*TX counter register 1.*
- `volatile uint32_t RXADDR0`  
*RX buffer offset register.*

- volatile uint16\_t RXCOUNT0  
*RX counter register 0.*
- volatile uint16\_t RXCOUNT1  
*RX counter register 1.*

## 7.42.2 Field Documentation

### 7.42.2.1 volatile uint32\_t stm32\_usb\_descriptor\_t::TXADDR0

TX buffer offset register.

### 7.42.2.2 volatile uint16\_t stm32\_usb\_descriptor\_t::TXCOUNT0

TX counter register 0.

### 7.42.2.3 volatile uint16\_t stm32\_usb\_descriptor\_t::TXCOUNT1

TX counter register 1.

### 7.42.2.4 volatile uint32\_t stm32\_usb\_descriptor\_t::RXADDR0

RX buffer offset register.

### 7.42.2.5 volatile uint16\_t stm32\_usb\_descriptor\_t::RXCOUNT0

RX counter register 0.

### 7.42.2.6 volatile uint16\_t stm32\_usb\_descriptor\_t::RXCOUNT1

RX counter register 1.

## 7.43 stm32\_usb\_t Struct Reference

### 7.43.1 Detailed Description

USB registers block.

```
#include <stm32_usb.h>
```

### Data Fields

- volatile uint32\_t EPR [USB\_ENDPOINTS\_NUMBER+1]  
*Endpoint registers.*

## 7.43.2 Field Documentation

### 7.43.2.1 volatile uint32\_t stm32\_usb\_t::EPR[USB\_ENDPOINTS\_NUMBER+1]

Endpoint registers.

## 7.44 TimeMeasurement Struct Reference

### 7.44.1 Detailed Description

Time Measurement structure.

```
#include <tm.h>
```

#### Data Fields

- `void(* start )(TimeMeasurement *tmp)`  
*Starts a measurement.*
- `void(* stop )(TimeMeasurement *tmp)`  
*Stops a measurement.*
- `halrcnt_t last`  
*Last measurement.*
- `halrcnt_t worst`  
*Worst measurement.*
- `halrcnt_t best`  
*Best measurement.*

### 7.44.2 Field Documentation

#### 7.44.2.1 `void(* TimeMeasurement::start)(TimeMeasurement *tmp)`

Starts a measurement.

#### 7.44.2.2 `void(* TimeMeasurement::stop)(TimeMeasurement *tmp)`

Stops a measurement.

#### 7.44.2.3 `halrcnt_t TimeMeasurement::last`

Last measurement.

#### 7.44.2.4 `halrcnt_t TimeMeasurement::worst`

Worst measurement.

#### 7.44.2.5 `halrcnt_t TimeMeasurement::best`

Best measurement.

## 7.45 UARTConfig Struct Reference

### 7.45.1 Detailed Description

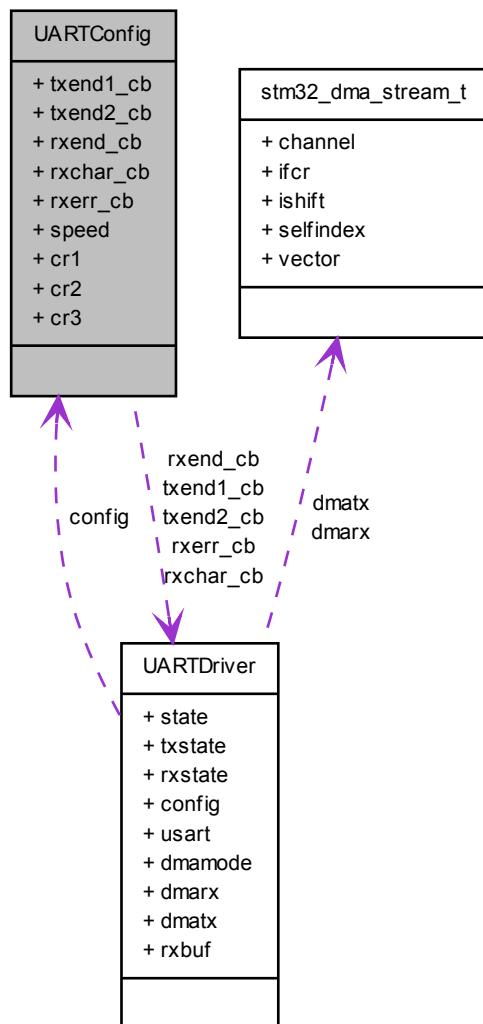
Driver configuration structure.

**Note**

It could be empty on some architectures.

```
#include <uart_lld.h>
```

Collaboration diagram for UARTConfig:

**Data Fields**

- **uartcb\_t txend1\_cb**  
*End of transmission buffer callback.*
- **uartcb\_t txend2\_cb**  
*Physical end of transmission callback.*
- **uartcb\_t rxend\_cb**  
*Receive buffer filled callback.*
- **uartccb\_t rxchar\_cb**

*Character received while out if the UART\_RECEIVE state.*

- `uartcb_t rxerr_cb`

*Receive error callback.*

- `uint32_t speed`

*Bit rate.*

- `uint16_t cr1`

*Initialization value for the CR1 register.*

- `uint16_t cr2`

*Initialization value for the CR2 register.*

- `uint16_t cr3`

*Initialization value for the CR3 register.*

## 7.45.2 Field Documentation

### 7.45.2.1 `uartcb_t` `UARTConfig::txend1_cb`

End of transmission buffer callback.

### 7.45.2.2 `uartcb_t` `UARTConfig::txend2_cb`

Physical end of transmission callback.

### 7.45.2.3 `uartcb_t` `UARTConfig::rxend_cb`

Receive buffer filled callback.

### 7.45.2.4 `uartccb_t` `UARTConfig::rxchar_cb`

Character received while out if the UART\_RECEIVE state.

### 7.45.2.5 `uartcb_t` `UARTConfig::rxerr_cb`

Receive error callback.

### 7.45.2.6 `uint32_t` `UARTConfig::speed`

Bit rate.

### 7.45.2.7 `uint16_t` `UARTConfig::cr1`

Initialization value for the CR1 register.

### 7.45.2.8 `uint16_t` `UARTConfig::cr2`

Initialization value for the CR2 register.

### 7.45.2.9 `uint16_t` `UARTConfig::cr3`

Initialization value for the CR3 register.

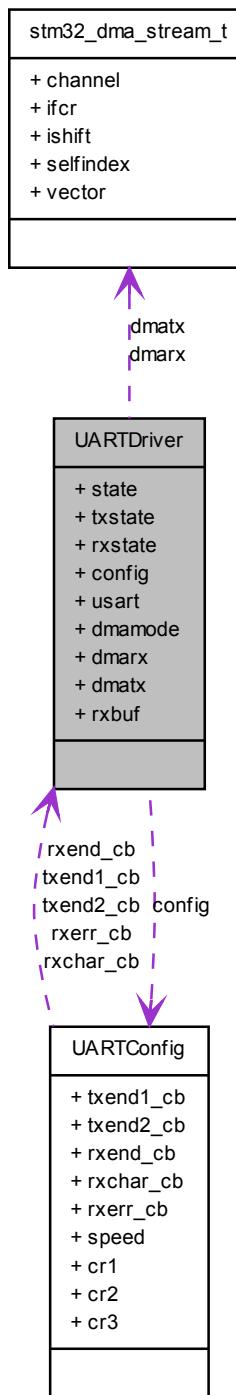
## 7.46 UARTDriver Struct Reference

### 7.46.1 Detailed Description

Structure representing an UART driver.

```
#include <uart_llld.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.*
- `USART_TypeDef * usart`  
*Pointer to the USART registers block.*
- `uint32_t dmamode`  
*DMA mode bit mask.*
- `const stm32_dma_stream_t * dmarx`  
*Receive DMA channel.*
- `const stm32_dma_stream_t * dmatx`  
*Transmit DMA channel.*
- `volatile uint16_t rdbuf`  
*Default receive buffer while into `UART_RX_IDLE` state.*

## 7.46.2 Field Documentation

### 7.46.2.1 `uartstate_t` `UARTDriver::state`

Driver state.

### 7.46.2.2 `uarttxstate_t` `UARTDriver::txstate`

Transmitter state.

### 7.46.2.3 `uartrxstate_t` `UARTDriver::rxstate`

Receiver state.

### 7.46.2.4 `const UARTConfig*` `UARTDriver::config`

Current configuration data.

### 7.46.2.5 `USART_TypeDef*` `UARTDriver::usart`

Pointer to the USART registers block.

### 7.46.2.6 `uint32_t` `UARTDriver::dmamode`

DMA mode bit mask.

### 7.46.2.7 `const stm32_dma_stream_t*` `UARTDriver::dmarx`

Receive DMA channel.

### 7.46.2.8 `const stm32_dma_stream_t*` `UARTDriver::dmatx`

Transmit DMA channel.

### 7.46.2.9 volatile uint16\_t UARTDriver::rdbuf

Default receive buffer while into `UART_RX_IDLE` state.

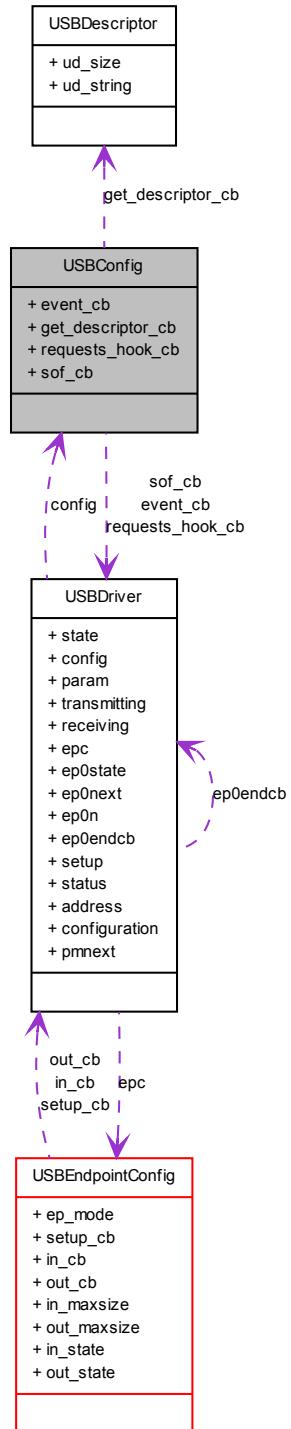
## 7.47 USBConfig Struct Reference

### 7.47.1 Detailed Description

Type of an USB driver configuration structure.

```
#include <usb_lld.h>
```

Collaboration diagram for USBConfig:



## Data Fields

- [usbeventcb\\_t event\\_cb](#)

*USB events callback.*

- `usbgetdescriptor_t get_descriptor_cb`  
*Device GET\_DESCRIPTOR request callback.*
- `usbreqhandler_t requests_hook_cb`  
*Requests hook callback.*
- `usbcallback_t sof_cb`  
*Start Of Frame callback.*

## 7.47.2 Field Documentation

### 7.47.2.1 `usbeventcb_t USBConfig::event_cb`

USB events callback.

This callback is invoked when an USB driver event is registered.

### 7.47.2.2 `usbgetdescriptor_t USBConfig::get_descriptor_cb`

Device GET\_DESCRIPTOR request callback.

#### Note

This callback is mandatory and cannot be set to NULL.

### 7.47.2.3 `usbreqhandler_t USBConfig::requests_hook_cb`

Requests hook callback.

This hook allows to be notified of standard requests or to handle non standard requests.

### 7.47.2.4 `usbcallback_t USBConfig::sof_cb`

Start Of Frame callback.

## 7.48 USBDescriptor Struct Reference

### 7.48.1 Detailed Description

Type of an USB descriptor.

```
#include <usb.h>
```

#### Data Fields

- `size_t ud_size`  
*Descriptor size in unicode characters.*
- `const uint8_t * ud_string`  
*Pointer to the descriptor.*

## 7.48.2 Field Documentation

### 7.48.2.1 `size_t USBDescriptor::ud_size`

Descriptor size in unicode characters.

### 7.48.2.2 const uint8\_t\* USBDescriptor::ud\_string

Pointer to the descriptor.

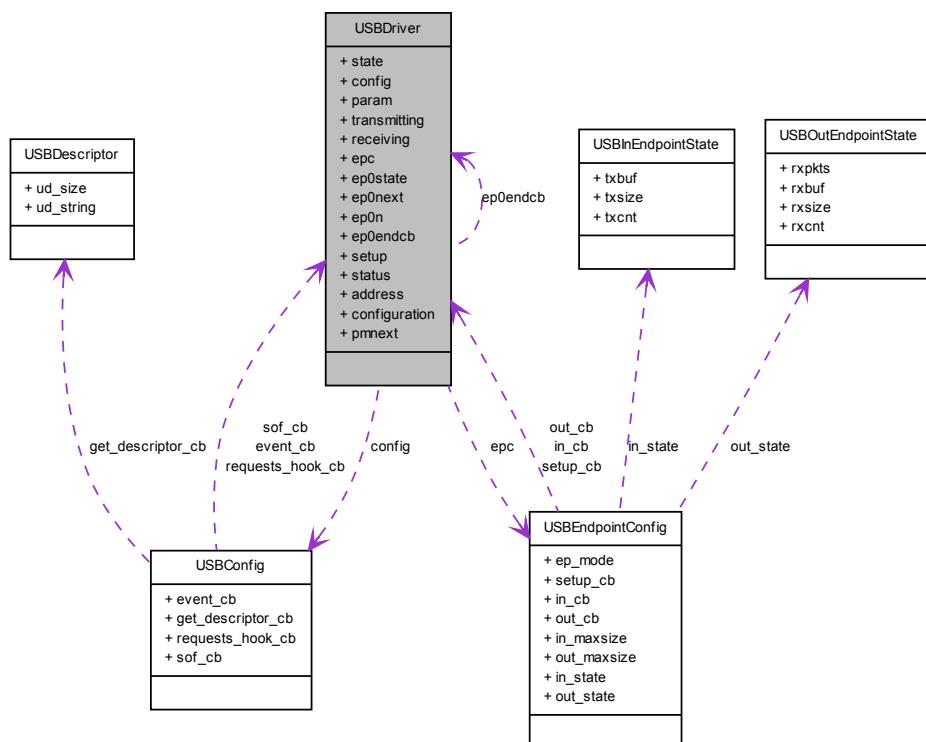
## 7.49 USBDriver Struct Reference

### 7.49.1 Detailed Description

Structure representing an USB driver.

```
#include <usb_ll.h>
```

Collaboration diagram for USBDriver:



## Data Fields

- **usbstate\_t state**  
*Driver state.*
- **const USBConfig \* config**  
*Current configuration data.*
- **void \* param**  
*Field available to user, it can be used to associate an application-defined handler to the USB driver.*
- **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.*
- [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.*
- uint32\_t **pmnext**  
*Pointer to the next address in the packet memory.*

## 7.49.2 Field Documentation

### 7.49.2.1 [usbstate\\_t](#) **USBDriver::state**

Driver state.

### 7.49.2.2 const [USBConfig](#)\* **USBDriver::config**

Current configuration data.

### 7.49.2.3 void\* **USBDriver::param**

Field available to user, it can be used to associate an application-defined handler to the USB driver.

### 7.49.2.4 uint16\_t **USBDriver::transmitting**

Bit map of the transmitting IN endpoints.

### 7.49.2.5 uint16\_t **USBDriver::receiving**

Bit map of the receiving OUT endpoints.

### 7.49.2.6 const [USBEndpointConfig](#)\* **USBDriver::epc[USB\_MAX\_ENDPOINTS+1]**

Active endpoints configurations.

### 7.49.2.7 [usbep0state\\_t](#) **USBDriver::ep0state**

Endpoint 0 state.

**7.49.2.8 uint8\_t\* USBDriver::ep0next**

Next position in the buffer to be transferred through endpoint 0.

**7.49.2.9 size\_t USBDriver::ep0n**

Number of bytes yet to be transferred through endpoint 0.

**7.49.2.10 usbcallback\_t USBDriver::ep0endcb**

Endpoint 0 end transaction callback.

**7.49.2.11 uint8\_t USBDriver::setup[8]**

Setup packet buffer.

**7.49.2.12 uint16\_t USBDriver::status**

Current USB device status.

**7.49.2.13 uint8\_t USBDriver::address**

Assigned USB address.

**7.49.2.14 uint8\_t USBDriver::configuration**

Current USB device configuration.

**7.49.2.15 uint32\_t USBDriver::pmnnext**

Pointer to the next address in the packet memory.

## 7.50 USBEndpointConfig Struct Reference

### 7.50.1 Detailed Description

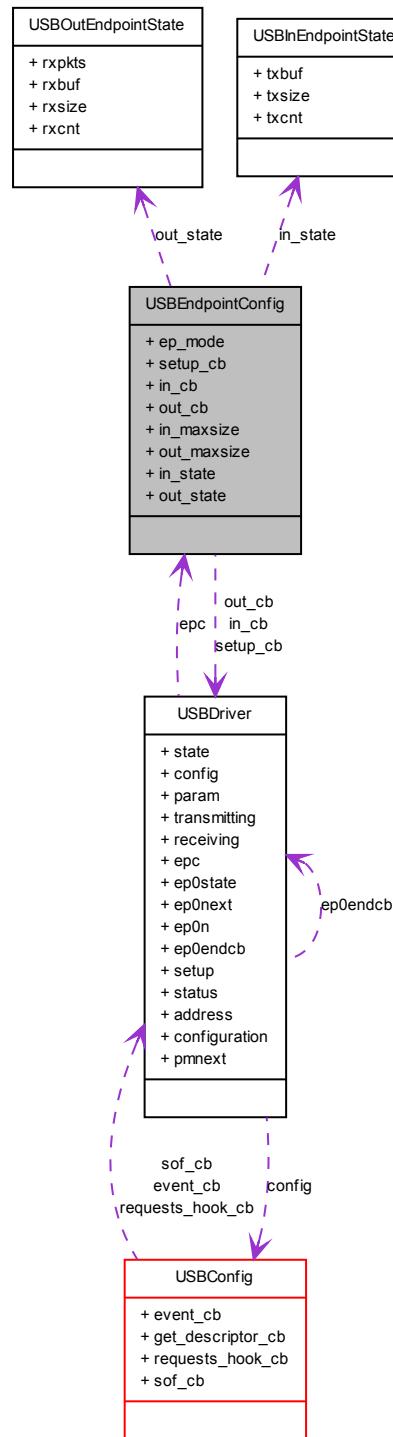
Type of an USB endpoint configuration structure.

**Note**

Platform specific restrictions may apply to endpoints.

```
#include <usb_llld.h>
```

Collaboration diagram for USBEndpointConfig:



## Data Fields

- `uint32_t ep_mode`

*Type and mode of the endpoint.*

- **usbepcallback\_t setup\_cb**  
*Setup packet notification callback.*
- **usbepcallback\_t in\_cb**  
*IN endpoint notification callback.*
- **usbepcallback\_t out\_cb**  
*OUT endpoint notification callback.*
- **uint16\_t in\_maxsize**  
*IN endpoint maximum packet size.*
- **uint16\_t out\_maxsize**  
*OUT endpoint maximum packet size.*
- **USBInEndpointState \* in\_state**  
*USBInEndpointState associated to the IN endpoint.*
- **USBOutEndpointState \* out\_state**  
*USBOutEndpointState associated to the OUT endpoint.*

## 7.50.2 Field Documentation

### 7.50.2.1 uint32\_t USBEndpointConfig::ep\_mode

Type and mode of the endpoint.

### 7.50.2.2 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.

### 7.50.2.3 usbepcallback\_t USBEndpointConfig::in\_cb

IN endpoint notification callback.

This field must be set to `NULL` if the IN endpoint is not used.

### 7.50.2.4 usbepcallback\_t USBEndpointConfig::out\_cb

OUT endpoint notification callback.

This field must be set to `NULL` if the OUT endpoint is not used.

### 7.50.2.5 uint16\_t USBEndpointConfig::in\_maxsize

IN endpoint maximum packet size.

This field must be set to zero if the IN endpoint is not used.

### 7.50.2.6 `uint16_t USBEndpointConfig::out_maxsize`

OUT endpoint maximum packet size.

This field must be set to zero if the OUT endpoint is not used.

### 7.50.2.7 `USBInEndpointState* USBEndpointConfig::in_state`

USBEndpointState associated to the IN endpoint.

This structure maintains the state of the IN endpoint when the endpoint is not in packet mode. Endpoints configured in packet mode must set this field to NULL.

### 7.50.2.8 `USBOutEndpointState* USBEndpointConfig::out_state`

USBEndpointState associated to the OUT endpoint.

This structure maintains the state of the OUT endpoint when the endpoint is not in packet mode. Endpoints configured in packet mode must set this field to NULL.

## 7.51 USBInEndpointState Struct Reference

### 7.51.1 Detailed Description

Type of an endpoint state structure.

```
#include <usb_lld.h>
```

### Data Fields

- `const uint8_t * txbuf`  
*Pointer to the transmission buffer.*
- `size_t txsize`  
*Requested transmit transfer size.*
- `size_t txcnt`  
*Transmitted bytes so far.*

### 7.51.2 Field Documentation

#### 7.51.2.1 `const uint8_t* USBInEndpointState::txbuf`

Pointer to the transmission buffer.

#### 7.51.2.2 `size_t USBInEndpointState::txsize`

Requested transmit transfer size.

#### 7.51.2.3 `size_t USBInEndpointState::txcnt`

Transmitted bytes so far.

## 7.52 USBOutEndpointState Struct Reference

### 7.52.1 Detailed Description

Type of an endpoint state structure.

```
#include <usb_lld.h>
```

### Data Fields

- `uint16_t rxpkts`  
*Number of packets to receive.*
- `uint8_t * rxbuf`  
*Pointer to the receive buffer.*
- `size_t rxsize`  
*Requested receive transfer size.*
- `size_t rxcnt`  
*Received bytes so far.*

### 7.52.2 Field Documentation

#### 7.52.2.1 `uint16_t USBOutEndpointState::rxpkts`

Number of packets to receive.

#### 7.52.2.2 `uint8_t* USBOutEndpointState::rxbuf`

Pointer to the receive buffer.

#### 7.52.2.3 `size_t USBOutEndpointState::rxsize`

Requested receive transfer size.

#### 7.52.2.4 `size_t USBOutEndpointState::rxcnt`

Received bytes so far.

# Chapter 8

## File Documentation

### 8.1 adc.c File Reference

#### 8.1.1 Detailed Description

```
ADC Driver code. #include "ch.h"  
#include "hal.h"
```

#### Functions

- void `adclInit` (void)  
*ADC Driver initialization.*
- void `adcObjectInit` (`ADCDriver` \*adcp)  
*Initializes the standard part of a `ADCDriver` structure.*
- void `adcStart` (`ADCDriver` \*adcp, const `ADCConfig` \*config)  
*Configures and activates the ADC peripheral.*
- void `adcStop` (`ADCDriver` \*adcp)  
*Deactivates the ADC peripheral.*
- void `adcStartConversion` (`ADCDriver` \*adcp, const `ADCConversionGroup` \*grpp, `adcsample_t` \*samples, `size_t` depth)  
*Starts an ADC conversion.*
- void `adcStartConversionI` (`ADCDriver` \*adcp, const `ADCConversionGroup` \*grpp, `adcsample_t` \*samples, `size_t` depth)  
*Starts an ADC conversion.*
- void `adcStopConversion` (`ADCDriver` \*adcp)  
*Stops an ongoing conversion.*
- void `adcStopConversionI` (`ADCDriver` \*adcp)  
*Stops an ongoing conversion.*
- msg\_t `adcConvert` (`ADCDriver` \*adcp, const `ADCConversionGroup` \*grpp, `adcsample_t` \*samples, `size_t` depth)  
*Performs an ADC conversion.*
- void `adcAcquireBus` (`ADCDriver` \*adcp)  
*Gains exclusive access to the ADC peripheral.*
- void `adcReleaseBus` (`ADCDriver` \*adcp)  
*Releases exclusive access to the ADC peripheral.*

## 8.2 adc.h File Reference

### 8.2.1 Detailed Description

ADC Driver macros and structures. #include "adc\_lld.h"

#### Functions

- void **adclinit** (void)  
*ADC Driver initialization.*
- void **adcObjectInit** (ADCDriver \*adcp)  
*Initializes the standard part of a `ADCDriver` structure.*
- void **adcStart** (ADCDriver \*adcp, const ADCConfig \*config)  
*Configures and activates the ADC peripheral.*
- void **adcStop** (ADCDriver \*adcp)  
*Deactivates the ADC peripheral.*
- void **adcStartConversion** (ADCDriver \*adcp, const ADCConversionGroup \*grpp, adcsample\_t \*samples, size\_t depth)  
*Starts an ADC conversion.*
- void **adcStartConversionI** (ADCDriver \*adcp, const ADCConversionGroup \*grpp, adcsample\_t \*samples, size\_t depth)  
*Starts an ADC conversion.*
- void **adcStopConversion** (ADCDriver \*adcp)  
*Stops an ongoing conversion.*
- void **adcStopConversionI** (ADCDriver \*adcp)  
*Stops an ongoing conversion.*
- void **adcAcquireBus** (ADCDriver \*adcp)  
*Gains exclusive access to the ADC peripheral.*
- void **adcReleaseBus** (ADCDriver \*adcp)  
*Releases exclusive access to the ADC peripheral.*

#### Defines

##### 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)  
*Resumes a thread waiting for a conversion completion.*
- #define **\_adc\_reset\_s**(adcp)  
*Resumes a thread waiting for a conversion completion.*
- #define **\_adc\_wakeup\_isr**(adcp)  
*Wakes up the waiting thread.*
- #define **\_adc\_timeout\_isr**(adcp)  
*Wakes up the waiting thread with a timeout message.*
- #define **\_adc\_isr\_half\_code**(adcp)  
*Common ISR code, half buffer event.*
- #define **\_adc\_isr\_full\_code**(adcp)  
*Common ISR code, full buffer event.*
- #define **\_adc\_isr\_error\_code**(adcp, err)  
*Common ISR code, error event.*

## Enumerations

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

## 8.3 adc\_lld.c File Reference

### 8.3.1 Detailed Description

STM32F1xx ADC subsystem low level driver source.

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

## Functions

- void `adc_lld_init` (void)  
*Low level ADC driver initialization.*
- void `adc_lld_start` (ADCDriver \*adcp)  
*Configures and activates the ADC peripheral.*
- void `adc_lld_stop` (ADCDriver \*adcp)  
*Deactivates the ADC peripheral.*
- void `adc_lld_start_conversion` (ADCDriver \*adcp)  
*Starts an ADC conversion.*
- void `adc_lld_stop_conversion` (ADCDriver \*adcp)  
*Stops an ongoing conversion.*

## Variables

- ADCDriver `ADCD1`  
*ADC1 driver identifier.*

## 8.4 adc\_lld.h File Reference

### 8.4.1 Detailed Description

STM32F1xx ADC subsystem low level driver header.

## Data Structures

- struct `ADCCConversionGroup`  
*Conversion group configuration structure.*
- struct `ADCCConfig`  
*Driver configuration structure.*
- struct `ADCDriver`  
*Structure representing an ADC driver.*

## Functions

- 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.*

## Defines

### Triggers selection

- #define `ADC_CR2_EXTSEL_SRC(n)` ((n) << 17)  
*Trigger source.*
- #define `ADC_CR2_EXTSEL_SWSTART` (7 << 17)  
*Software trigger.*

### Available analog channels

- #define `ADC_CHANNEL_IN0` 0  
*External analog input 0.*
- #define `ADC_CHANNEL_IN1` 1  
*External analog input 1.*
- #define `ADC_CHANNEL_IN2` 2  
*External analog input 2.*
- #define `ADC_CHANNEL_IN3` 3  
*External analog input 3.*
- #define `ADC_CHANNEL_IN4` 4  
*External analog input 4.*
- #define `ADC_CHANNEL_IN5` 5  
*External analog input 5.*
- #define `ADC_CHANNEL_IN6` 6  
*External analog input 6.*
- #define `ADC_CHANNEL_IN7` 7  
*External analog input 7.*
- #define `ADC_CHANNEL_IN8` 8  
*External analog input 8.*
- #define `ADC_CHANNEL_IN9` 9  
*External analog input 9.*
- #define `ADC_CHANNEL_IN10` 10  
*External analog input 10.*
- #define `ADC_CHANNEL_IN11` 11  
*External analog input 11.*
- #define `ADC_CHANNEL_IN12` 12  
*External analog input 12.*
- #define `ADC_CHANNEL_IN13` 13  
*External analog input 13.*
- #define `ADC_CHANNEL_IN14` 14  
*External analog input 14.*
- #define `ADC_CHANNEL_IN15` 15  
*External analog input 15.*
- #define `ADC_CHANNEL_SENSOR` 16  
*Internal temperature sensor.*
- #define `ADC_CHANNEL_VREFINT` 17  
*Internal reference.*

### Sampling rates

- #define `ADC_SAMPLE_1P5` 0  
    *1.5 cycles sampling time.*
- #define `ADC_SAMPLE_7P5` 1  
    *7.5 cycles sampling time.*
- #define `ADC_SAMPLE_13P5` 2  
    *13.5 cycles sampling time.*
- #define `ADC_SAMPLE_28P5` 3  
    *28.5 cycles sampling time.*
- #define `ADC_SAMPLE_41P5` 4  
    *41.5 cycles sampling time.*
- #define `ADC_SAMPLE_55P5` 5  
    *55.5 cycles sampling time.*
- #define `ADC_SAMPLE_71P5` 6  
    *71.5 cycles sampling time.*
- #define `ADC_SAMPLE_239P5` 7  
    *239.5 cycles sampling time.*

### Configuration options

- #define `STM32_ADC_USE_ADC1` TRUE  
    *ADC1 driver enable switch.*
- #define `STM32_ADC_ADC1_DMA_PRIORITY` 2  
    *ADC1 DMA priority (0..3) lowest..highest).*
- #define `STM32_ADC_ADC1_IRQ_PRIORITY` 5  
    *ADC1 interrupt priority level setting.*

### Sequences building helper macros

- #define `ADC_SQR1_NUM_CH`(n) (((n) - 1) << 20)  
    *Number of channels in a conversion sequence.*
- #define `ADC_SQR3_SQ1_N`(n) ((n) << 0)  
    *1st channel in seq.*
- #define `ADC_SQR3_SQ2_N`(n) ((n) << 5)  
    *2nd channel in seq.*
- #define `ADC_SQR3_SQ3_N`(n) ((n) << 10)  
    *3rd channel in seq.*
- #define `ADC_SQR3_SQ4_N`(n) ((n) << 15)  
    *4th channel in seq.*
- #define `ADC_SQR3_SQ5_N`(n) ((n) << 20)  
    *5th channel in seq.*
- #define `ADC_SQR3_SQ6_N`(n) ((n) << 25)  
    *6th channel in seq.*
- #define `ADC_SQR2_SQ7_N`(n) ((n) << 0)  
    *7th channel in seq.*
- #define `ADC_SQR2_SQ8_N`(n) ((n) << 5)  
    *8th channel in seq.*
- #define `ADC_SQR2_SQ9_N`(n) ((n) << 10)  
    *9th channel in seq.*
- #define `ADC_SQR2_SQ10_N`(n) ((n) << 15)  
    *10th channel in seq.*
- #define `ADC_SQR2_SQ11_N`(n) ((n) << 20)  
    *11th channel in seq.*
- #define `ADC_SQR2_SQ12_N`(n) ((n) << 25)  
    *12th channel in seq.*
- #define `ADC_SQR1_SQ13_N`(n) ((n) << 0)  
    *13th channel in seq.*
- #define `ADC_SQR1_SQ14_N`(n) ((n) << 5)  
    *14th channel in seq.*
- #define `ADC_SQR1_SQ15_N`(n) ((n) << 10)  
    *15th channel in seq.*
- #define `ADC_SQR1_SQ16_N`(n) ((n) << 15)  
    *16th channel in seq.*

### Sampling rate settings helper macros

- `#define ADC_SMPR2_SMP_AN0(n) ((n) << 0)`  
*AN0 sampling time.*
- `#define ADC_SMPR2_SMP_AN1(n) ((n) << 3)`  
*AN1 sampling time.*
- `#define ADC_SMPR2_SMP_AN2(n) ((n) << 6)`  
*AN2 sampling time.*
- `#define ADC_SMPR2_SMP_AN3(n) ((n) << 9)`  
*AN3 sampling time.*
- `#define ADC_SMPR2_SMP_AN4(n) ((n) << 12)`  
*AN4 sampling time.*
- `#define ADC_SMPR2_SMP_AN5(n) ((n) << 15)`  
*AN5 sampling time.*
- `#define ADC_SMPR2_SMP_AN6(n) ((n) << 18)`  
*AN6 sampling time.*
- `#define ADC_SMPR2_SMP_AN7(n) ((n) << 21)`  
*AN7 sampling time.*
- `#define ADC_SMPR2_SMP_AN8(n) ((n) << 24)`  
*AN8 sampling time.*
- `#define ADC_SMPR2_SMP_AN9(n) ((n) << 27)`  
*AN9 sampling time.*
- `#define ADC_SMPR1_SMP_AN10(n) ((n) << 0)`  
*AN10 sampling time.*
- `#define ADC_SMPR1_SMP_AN11(n) ((n) << 3)`  
*AN11 sampling time.*
- `#define ADC_SMPR1_SMP_AN12(n) ((n) << 6)`  
*AN12 sampling time.*
- `#define ADC_SMPR1_SMP_AN13(n) ((n) << 9)`  
*AN13 sampling time.*
- `#define ADC_SMPR1_SMP_AN14(n) ((n) << 12)`  
*AN14 sampling time.*
- `#define ADC_SMPR1_SMP_AN15(n) ((n) << 15)`  
*AN15 sampling time.*
- `#define ADC_SMPR1_SMP_SENSOR(n) ((n) << 18)`  
*Temperature Sensor sampling time.*
- `#define ADC_SMPR1_SMP_VREF(n) ((n) << 21)`  
*Voltage Reference sampling time.*

### Typedefs

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

### Enumerations

- `enum adcerror_t { ADC_ERR_DMAFAILURE = 0 }`  
*Possible ADC failure causes.*

## 8.5 can.c File Reference

### 8.5.1 Detailed Description

```
CAN Driver code. #include "ch.h"
#include "hal.h"
```

### Functions

- void **canInit** (void)
 

*CAN Driver initialization.*
- void **canObjectInit** (**CANDriver** \*canp)
 

*Initializes the standard part of a **CANDriver** structure.*
- void **canStart** (**CANDriver** \*canp, const **CANConfig** \*config)
 

*Configures and activates the CAN peripheral.*
- void **canStop** (**CANDriver** \*canp)
 

*Deactivates the CAN peripheral.*
- msg\_t **canTransmit** (**CANDriver** \*canp, const **CANTxFrame** \*ctfp, systime\_t timeout)
 

*Can frame transmission.*
- msg\_t **canReceive** (**CANDriver** \*canp, **CANRxFrame** \*crfp, systime\_t timeout)
 

*Can frame receive.*
- canstatus\_t **canGetAndClearFlags** (**CANDriver** \*canp)
 

*Returns the current status mask and clears it.*
- void **canSleep** (**CANDriver** \*canp)
 

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

*Enforces leaving the sleep mode.*

## 8.6 can.h File Reference

### 8.6.1 Detailed Description

CAN Driver macros and structures. #include "can\_lld.h"

### Functions

- void **canInit** (void)
 

*CAN Driver initialization.*
- void **canObjectInit** (**CANDriver** \*canp)
 

*Initializes the standard part of a **CANDriver** structure.*
- void **canStart** (**CANDriver** \*canp, const **CANConfig** \*config)
 

*Configures and activates the CAN peripheral.*
- void **canStop** (**CANDriver** \*canp)
 

*Deactivates the CAN peripheral.*
- msg\_t **canTransmit** (**CANDriver** \*canp, const **CANTxFrame** \*ctfp, systime\_t timeout)
 

*Can frame transmission.*
- msg\_t **canReceive** (**CANDriver** \*canp, **CANRxFrame** \*crfp, systime\_t timeout)
 

*Can frame receive.*
- canstatus\_t **canGetAndClearFlags** (**CANDriver** \*canp)
 

*Returns the current status mask and clears it.*

## Defines

### CAN status flags

- #define CAN\_LIMIT\_WARNING 1  
*Errors rate warning.*
- #define CAN\_LIMIT\_ERROR 2  
*Errors rate error.*
- #define CAN\_BUS\_OFF\_ERROR 4  
*Bus off condition reached.*
- #define CAN\_FRAMING\_ERROR 8  
*Framing error of some kind on the CAN bus.*
- #define CAN\_OVERFLOW\_ERROR 16  
*Overflow in receive queue.*

### CAN configuration options

- #define CAN\_USE\_SLEEP\_MODE TRUE  
*Sleep mode related APIs inclusion switch.*

### Macro Functions

- #define canAddFlagsI(camp, mask) ((camp)->status |= (mask))  
*Adds some flags to the CAN status mask.*

## Enumerations

- enum canstate\_t {
   
CAN\_UNINIT = 0, CAN\_STOP = 1, CAN\_STARTING = 2, CAN\_READY = 3,
   
CAN\_SLEEP = 4
 }
- Driver state machine possible states.*

## 8.7 can\_lld.c File Reference

### 8.7.1 Detailed Description

STM32 CAN subsystem low level driver source.

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

## Functions

- CH\_IRQ\_HANDLER (CAN1\_TX\_IRQHandler)  
*CAN1 TX interrupt handler.*
- CH\_IRQ\_HANDLER (CAN1\_RX1\_IRQHandler)  
*CAN1 RX1 interrupt handler.*
- CH\_IRQ\_HANDLER (CAN1\_SCE\_IRQHandler)  
*CAN1 SCE interrupt handler.*
- void can\_lld\_init (void)  
*Low level CAN driver initialization.*
- void can\_lld\_start (CANDriver \*camp)  
*Configures and activates the CAN peripheral.*
- void can\_lld\_stop (CANDriver \*camp)  
*Deactivates the CAN peripheral.*

- `bool_t can_lld_can_transmit (CANDriver *canp)`  
*Determines whether a frame can be transmitted.*
- `void can_lld_transmit (CANDriver *canp, const CANTxFrame *ctfp)`  
*Inserts a frame into the transmit queue.*
- `bool_t can_lld_can_receive (CANDriver *canp)`  
*Determines whether a frame has been received.*
- `void can_lld_receive (CANDriver *canp, CANRxFrame *crfp)`  
*Receives a frame from the input queue.*
- `void can_lld_sleep (CANDriver *canp)`  
*Enters the sleep mode.*
- `void can_lld_wakeup (CANDriver *canp)`  
*Enforces leaving the sleep mode.*

## Variables

- `CANDriver CAND1`  
*ADC1 driver identifier.*

## 8.8 can\_lld.h File Reference

### 8.8.1 Detailed Description

STM32 CAN subsystem low level driver header.

### Data Structures

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

### Functions

- `void can_lld_init (void)`  
*Low level CAN driver initialization.*
- `void can_lld_start (CANDriver *canp)`  
*Configures and activates the CAN peripheral.*
- `void can_lld_stop (CANDriver *canp)`  
*Deactivates the CAN peripheral.*
- `bool_t can_lld_can_transmit (CANDriver *canp)`  
*Determines whether a frame can be transmitted.*
- `void can_lld_transmit (CANDriver *canp, const CANTxFrame *ctfp)`  
*Inserts a frame into the transmit queue.*

- `bool_t can_lld_can_receive (CANDriver *canp)`  
*Determines whether a frame has been received.*
- `void can_lld_receive (CANDriver *canp, CANRxFrame *crfp)`  
*Receives a frame from the input queue.*

## Defines

- `#define CAN_SUPPORTS_SLEEP TRUE`  
*This switch defines whether the driver implementation supports a low power switch mode with automatic an wakeup feature.*
- `#define CAN_MAX_FILTERS 28`  
*Minimum number of CAN filters.*
- `#define CAN_BTR_BRP(n) (n)`  
*BRP field macro.*
- `#define CAN_BTR_TS1(n) ((n) << 16)`  
*TS1 field macro.*
- `#define CAN_BTR_TS2(n) ((n) << 20)`  
*TS2 field macro.*
- `#define CAN_BTR_SJW(n) ((n) << 24)`  
*SJW field macro.*
- `#define CAN_IDE_STD 0`  
*Standard id.*
- `#define CAN_IDE_EXT 1`  
*Extended id.*
- `#define CAN_RTR_DATA 0`  
*Data frame.*
- `#define CAN_RTR_REMOTE 1`  
*Remote frame.*

## Configuration options

- `#define STM32_CAN_USE_CAN1 TRUE`  
*CAN1 driver enable switch.*
- `#define STM32_CAN_CAN1_IRQ_PRIORITY 11`  
*CAN1 interrupt priority level setting.*

## Typedefs

- `typedef uint32_t canstatus_t`  
*CAN status flags.*

## 8.9 ext.c File Reference

### 8.9.1 Detailed Description

```
EXT Driver code. #include "ch.h"
#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.*

## 8.10 ext\_lld.c File Reference

### 8.10.1 Detailed Description

```
STM32 EXT subsystem low level driver source. #include "ch.h"
#include "hal.h"
```

## Functions

- `CH_IRQ_HANDLER` (EXTI0\_IRQHandler)  
*EXTI[0] interrupt handler.*
- `CH_IRQ_HANDLER` (EXTI1\_IRQHandler)  
*EXTI[1] interrupt handler.*
- `CH_IRQ_HANDLER` (EXTI2\_IRQHandler)  
*EXTI[2] interrupt handler.*
- `CH_IRQ_HANDLER` (EXTI3\_IRQHandler)  
*EXTI[3] interrupt handler.*
- `CH_IRQ_HANDLER` (EXTI4\_IRQHandler)  
*EXTI[4] interrupt handler.*
- `CH_IRQ_HANDLER` (EXTI9\_5\_IRQHandler)  
*EXTI[5]...EXTI[9] interrupt handler.*
- `CH_IRQ_HANDLER` (EXTI15\_10\_IRQHandler)  
*EXTI[10]...EXTI[15] interrupt handler.*
- `CH_IRQ_HANDLER` (PVD\_IRQHandler)  
*EXTI[16] interrupt handler (PVD).*
- `CH_IRQ_HANDLER` (RTCAlarm\_IRQHandler)  
*EXTI[17] interrupt handler (RTC).*
- `CH_IRQ_HANDLER` (USB\_FS\_WKUP\_IRQHandler)  
*EXTI[18] interrupt handler (USB\_FS\_WKUP).*
- 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_ll_channel_enable (EXTDriver *extp, expchannel_t channel)`  
*Enables an EXT channel.*
- void `ext_ll_channel_disable (EXTDriver *extp, expchannel_t channel)`  
*Disables an EXT channel.*

## Variables

- `EXTDriver EXTD1`  
*EXTD1 driver identifier.*

## 8.11 ext\_ll.h File Reference

### 8.11.1 Detailed Description

STM32 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.*

## Functions

- void `ext_ll_init (void)`  
*Low level EXT driver initialization.*
- void `ext_ll_start (EXTDriver *extp)`  
*Configures and activates the EXT peripheral.*
- void `ext_ll_stop (EXTDriver *extp)`  
*Deactivates the EXT peripheral.*
- void `ext_ll_channel_enable (EXTDriver *extp, expchannel_t channel)`  
*Enables an EXT channel.*
- void `ext_ll_channel_disable (EXTDriver *extp, expchannel_t channel)`  
*Disables an EXT channel.*

## Defines

- `#define EXT_MAX_CHANNELS STM32 EXTI_NUM_CHANNELS`  
*Available number of EXT channels.*
- `#define EXT_CHANNELS_MASK ((1 << EXT_MAX_CHANNELS) - 1)`  
*Mask of the available channels.*

### EXTI configuration helpers

- `#define EXT_MODE_EXTI(m0, m1, m2, m3, m4, m5, m6, m7, m8, m9, m10, m11, m12, m13, m14, m15)`  
*EXTI-GPIO association macro.*
- `#define EXT_MODE_GPIOA 0`  
*GPIOA identifier.*
- `#define EXT_MODE_GPIOB 1`  
*GPIOB identifier.*
- `#define EXT_MODE_GPIOC 2`  
*GPIOC identifier.*
- `#define EXT_MODE_GPIOD 3`  
*GPIOD identifier.*
- `#define EXT_MODE_GPIOE 4`  
*GPIOE identifier.*
- `#define EXT_MODE_GPIOF 5`  
*GPIOF identifier.*
- `#define EXT_MODE_GPIOG 6`  
*GPIOG identifier.*
- `#define EXT_MODE_GPIOH 7`  
*GPIOH identifier.*
- `#define EXT_MODE_GPIOI 8`  
*GPIOI identifier.*

### Configuration options

- `#define STM32_EXT_EXTI0_IRQ_PRIORITY 6`  
*EXTI0 interrupt priority level setting.*
- `#define STM32_EXT_EXTI1_IRQ_PRIORITY 6`  
*EXTI1 interrupt priority level setting.*
- `#define STM32_EXT_EXTI2_IRQ_PRIORITY 6`  
*EXTI2 interrupt priority level setting.*
- `#define STM32_EXT_EXTI3_IRQ_PRIORITY 6`  
*EXTI3 interrupt priority level setting.*
- `#define STM32_EXT_EXTI4_IRQ_PRIORITY 6`  
*EXTI4 interrupt priority level setting.*
- `#define STM32_EXT_EXTI5_9_IRQ_PRIORITY 6`  
*EXTI9..5 interrupt priority level setting.*
- `#define STM32_EXT_EXTI10_15_IRQ_PRIORITY 6`  
*EXTI15..10 interrupt priority level setting.*
- `#define STM32_EXT_EXTI16_IRQ_PRIORITY 6`  
*EXTI16 interrupt priority level setting.*
- `#define STM32_EXT_EXTI17_IRQ_PRIORITY 6`  
*EXTI17 interrupt priority level setting.*
- `#define STM32_EXT_EXTI18_IRQ_PRIORITY 6`  
*EXTI18 interrupt priority level setting.*
- `#define STM32_EXT_EXTI19_IRQ_PRIORITY 6`  
*EXTI19 interrupt priority level setting.*
- `#define STM32_EXT_EXTI20_IRQ_PRIORITY 6`  
*EXTI20 interrupt priority level setting.*
- `#define STM32_EXT_EXTI21_IRQ_PRIORITY 6`  
*EXTI21 interrupt priority level setting.*
- `#define STM32_EXT_EXTI22_IRQ_PRIORITY 6`  
*EXTI22 interrupt priority level setting.*

### 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.*

## 8.12 gpt.c File Reference

### 8.12.1 Detailed Description

GPT Driver code.

```
#include "ch.h"
#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 **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.*

## 8.13 gpt.h File Reference

### 8.13.1 Detailed Description

GPT Driver macros and structures.

```
#include "gpt_lld.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 **gptStartContinuous** (GPTDriver \*gptp, gptcnt\_t interval)

- void `gptStartContinuousl` (GPTDriver \*gptp, gptcnt\_t interval)
 

*Starts the timer in continuous mode.*
- void `gptStartOneShotl` (GPTDriver \*gptp, gptcnt\_t interval)
 

*Starts the timer in one shot mode.*
- 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.*

## Typedefs

- typedef struct GPTDriver GPTDriver
 

*Type of a structure representing a GPT driver.*
- typedef void(\* gptcallback\_t )(GPTDriver \*gptp)
 

*GPT notification callback type.*

## Enumerations

- enum gptstate\_t {
 

`GPT_UNINIT` = 0, `GPT_STOP` = 1, `GPT_READY` = 2, `GPT_CONTINUOUS` = 3,  
`GPT_ONESHOT` = 4 }

*Driver state machine possible states.*

## 8.14 gpt\_lld.c File Reference

### 8.14.1 Detailed Description

```
STM32 GPT subsystem low level driver source. #include "ch.h"
#include "hal.h"
```

## Functions

- void `gpt_lld_init` (void)
 

*Low level GPT driver initialization.*
- void `gpt_lld_start` (GPTDriver \*gptp)
 

*Configures and activates the GPT peripheral.*
- void `gpt_lld_stop` (GPTDriver \*gptp)
 

*Deactivates the GPT peripheral.*
- void `gpt_lld_start_timer` (GPTDriver \*gptp, gptcnt\_t interval)
 

*Starts the timer in continuous mode.*
- void `gpt_lld_stop_timer` (GPTDriver \*gptp)
 

*Stops the timer.*
- void `gpt_lld_polled_delay` (GPTDriver \*gptp, gptcnt\_t interval)
 

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

## Variables

- **GPTDriver GPTD1**  
*GPTD1 driver identifier.*
- **GPTDriver GPTD2**  
*GPTD2 driver identifier.*
- **GPTDriver GPTD3**  
*GPTD3 driver identifier.*
- **GPTDriver GPTD4**  
*GPTD4 driver identifier.*
- **GPTDriver GPTD5**  
*GPTD5 driver identifier.*
- **GPTDriver GPTD8**  
*GPTD8 driver identifier.*

## 8.15 gpt\_ll.h File Reference

### 8.15.1 Detailed Description

STM32 GPT subsystem low level driver header.

## Data Structures

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

## Functions

- void **gpt\_ll\_init** (void)  
*Low level GPT driver initialization.*
- void **gpt\_ll\_start** (GPTDriver \*gptp)  
*Configures and activates the GPT peripheral.*
- void **gpt\_ll\_stop** (GPTDriver \*gptp)  
*Deactivates the GPT peripheral.*
- void **gpt\_ll\_start\_timer** (GPTDriver \*gptp, gptcnt\_t interval)  
*Starts the timer in continuous mode.*
- void **gpt\_ll\_stop\_timer** (GPTDriver \*gptp)  
*Stops the timer.*
- void **gpt\_ll\_polled\_delay** (GPTDriver \*gptp, gptcnt\_t interval)  
*Starts the timer in one shot mode and waits for completion.*

## Defines

### Configuration options

- #define **STM32\_GPT\_USE\_TIM1** TRUE  
*GPTD1 driver enable switch.*
- #define **STM32\_GPT\_USE\_TIM2** TRUE

```

GPTD2 driver enable switch.
• #define STM32_GPT_USE_TIM3 TRUE
GPTD3 driver enable switch.
• #define STM32_GPT_USE_TIM4 TRUE
GPTD4 driver enable switch.
• #define STM32_GPT_USE_TIM5 TRUE
GPTD5 driver enable switch.
• #define STM32_GPT_USE_TIM8 TRUE
GPTD8 driver enable switch.
• #define STM32_GPT_TIM1_IRQ_PRIORITY 7
GPTD1 interrupt priority level setting.
• #define STM32_GPT_TIM2_IRQ_PRIORITY 7
GPTD2 interrupt priority level setting.
• #define STM32_GPT_TIM3_IRQ_PRIORITY 7
GPTD3 interrupt priority level setting.
• #define STM32_GPT_TIM4_IRQ_PRIORITY 7
GPTD4 interrupt priority level setting.
• #define STM32_GPT_TIM5_IRQ_PRIORITY 7
GPTD5 interrupt priority level setting.
• #define STM32_GPT_TIM8_IRQ_PRIORITY 7
GPTD5 interrupt priority level setting.

```

## Typedefs

- typedef uint32\_t gptfreq\_t
 *GPT frequency type.*
- typedef uint16\_t gptcnt\_t
 *GPT counter type.*

## 8.16 hal.c File Reference

### 8.16.1 Detailed Description

HAL subsystem code.

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

## Functions

- void halInit(void)
 *HAL initialization.*

## 8.17 hal.h File Reference

### 8.17.1 Detailed Description

HAL subsystem header.

```
#include "board.h"
#include "halconf.h"
#include "hal_lld.h"
#include "tm.h"
#include "pal.h"
```

```
#include "adc.h"
#include "can.h"
#include "ext.h"
#include "gpt.h"
#include "i2c.h"
#include "icu.h"
#include "mac.h"
#include "pwm.h"
#include "rtc.h"
#include "serial.h"
#include "sdc.h"
#include "spi.h"
#include "uart.h"
#include "usb.h"
#include "mmc_spi.h"
#include "serial_usb.h"
```

## Functions

- void **halInit** (void)

*HAL initialization.*

## Defines

- #define **halPolledDelay**(ticks)  
*Polled delay.*

### Time conversion utilities for the realtime counter

- #define **S2RTT**(sec) (halGetCounterFrequency() \* (sec))  
*Seconds to realtime ticks.*
- #define **MS2RTT**(msec) (((halGetCounterFrequency() + 999UL) / 1000UL) \* (msec))  
*Milliseconds to realtime ticks.*
- #define **US2RTT**(usec)  
*Microseconds to realtime ticks.*

### Macro Functions

- #define **halGetCounterValue**() hal\_lld\_get\_counter\_value()  
*Returns the current value of the system free running counter.*
- #define **halGetCounterFrequency**() hal\_lld\_get\_counter\_frequency()  
*Realtime counter frequency.*
- #define **halIsCounterWithin**(start, end)  
*Realtime window test.*

## 8.18 hal\_lld.c File Reference

### 8.18.1 Detailed Description

STM32F1xx HAL subsystem low level driver source.

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

### Functions

- void [hal\\_ll\\_init](#) (void)  
*Low level HAL driver initialization.*
- void [stm32\\_clock\\_init](#) (void)  
*STM32 clocks and PLL initialization.*

## 8.19 hal\_ll.h File Reference

### 8.19.1 Detailed Description

STM32F1xx HAL subsystem low level driver header.

#### Precondition

This module requires the following macros to be defined in the `board.h` file:

- STM32\_LSECLK.
- STM32\_HSECLK.

One of the following macros must also be defined:

- STM32F10X\_LD\_VL for Value Line Low Density devices.
- STM32F10X\_MD\_VL for Value Line Medium Density devices.
- STM32F10X\_LD for Performance Low Density devices.
- STM32F10X\_MD for Performance Medium Density devices.
- STM32F10X\_HD for Performance High Density devices.
- STM32F10X\_XL for Performance eXtra Density devices.
- STM32F10X\_CL for Connectivity Line devices.

```
#include "stm32.h"
#include "stm32_dma.h"
#include "stm32_rcc.h"
```

### Functions

- void [hal\\_ll\\_init](#) (void)  
*Low level HAL driver initialization.*
- void [stm32\\_clock\\_init](#) (void)  
*STM32 clocks and PLL initialization.*

### Defines

- #define [HAL\\_IMPLEMENTES\\_COUNTERS](#) TRUE  
*Defines the support for realtime counters in the HAL.*
- #define [hal\\_ll\\_get\\_counter\\_value](#)() DWT\_CYCCNT  
*Returns the current value of the system free running counter.*
- #define [hal\\_ll\\_get\\_counter\\_frequency](#)() STM32\_HCLK  
*Realtime counter frequency.*

### Internal clock sources

- #define STM32\_HSICLK 8000000
- #define STM32\_LSICLK 40000

### PWR\_CR register bits definitions

- #define STM32\_PLS\_MASK (7 << 5)
- #define STM32\_PLS\_LEV0 (0 << 5)
- #define STM32\_PLS\_LEV1 (1 << 5)
- #define STM32\_PLS\_LEV2 (2 << 5)
- #define STM32\_PLS\_LEV3 (3 << 5)
- #define STM32\_PLS\_LEV4 (4 << 5)
- #define STM32\_PLS\_LEV5 (5 << 5)
- #define STM32\_PLS\_LEV6 (6 << 5)
- #define STM32\_PLS\_LEV7 (7 << 5)

### Configuration options

- #define STM32\_NO\_INIT FALSE  
*Disables the PWR/RCC initialization in the HAL.*
- #define STM32\_PVD\_ENABLE FALSE  
*Enables or disables the programmable voltage detector.*
- #define STM32\_PLS STM32\_PLS\_LEV0  
*Sets voltage level for programmable voltage detector.*
- #define STM32\_HSI\_ENABLED TRUE  
*Enables or disables the HSI clock source.*
- #define STM32\_LSI\_ENABLED FALSE  
*Enables or disables the LSI clock source.*
- #define STM32\_HSE\_ENABLED TRUE  
*Enables or disables the HSE clock source.*
- #define STM32\_LSE\_ENABLED FALSE  
*Enables or disables the LSE clock source.*

### Platform identification

- #define PLATFORM\_NAME "STM32"

## Typedefs

- typedef uint32\_t halclock\_t  
*Type representing a system clock frequency.*
- typedef uint32\_t halrtcnt\_t  
*Type of the realtime free counter value.*

## 8.20 hal\_lld\_f100.h File Reference

### 8.20.1 Detailed Description

STM32F100 Value Line HAL subsystem low level driver header.

### Defines

- #define STM32\_ACTIVATE\_PLL TRUE  
*PLL activation flag.*
- #define STM32\_PLLMUL ((STM32\_PLLMUL\_VALUE - 2) << 18)

- PLLmul field.*
- #define **STM32\_PLLCLKIN** (STM32\_HSECLK / 1)  
*PLL input clock frequency.*
- #define **STM32\_PLLCLKOUT** (STM32\_PLLCLKIN \* STM32\_PLLMUL\_VALUE)  
*PLL output clock frequency.*
- #define **STM32\_SYSCLK** STM32\_PLLCLKOUT  
*System clock source.*
- #define **STM32\_HCLK** (STM32\_SYSCLK / 1)  
*AHB frequency.*
- #define **STM32\_PCLK1** (STM32\_HCLK / 1)  
*APB1 frequency.*
- #define **STM32\_PCLK2** (STM32\_HCLK / 1)  
*APB2 frequency.*
- #define **STM32\_RTCCLK** STM32\_LSECLK  
*RTC clock.*
- #define **STM32\_ADCCLK** (STM32\_PCLK2 / 2)  
*ADC frequency.*
- #define **STM32\_TIMCLK1** (STM32\_PCLK1 \* 1)  
*Timers 2, 3, 4, 5, 6, 7, 12, 13, 14 clock.*
- #define **STM32\_TIMCLK2** (STM32\_PCLK2 \* 1)  
*Timers 1, 8, 9, 10, 11 clock.*
- #define **STM32\_FLASHBITS** 0x00000010  
*Flash settings.*

#### Platform identification

- #define **PLATFORM\_NAME** "STM32F1 Value Line"

#### Absolute Maximum Ratings

- #define **STM32\_SYSCLK\_MAX** 24000000  
*Maximum system clock frequency.*
- #define **STM32\_HSECLK\_MAX** 24000000  
*Maximum HSE clock frequency.*
- #define **STM32\_HSECLK\_MIN** 1000000  
*Minimum HSE clock frequency.*
- #define **STM32\_LSECLK\_MAX** 1000000  
*Maximum LSE clock frequency.*
- #define **STM32\_LSECLK\_MIN** 32768  
*Minimum LSE clock frequency.*
- #define **STM32\_PLLIN\_MAX** 24000000  
*Maximum PLLs input clock frequency.*
- #define **STM32\_PLLIN\_MIN** 1000000  
*Maximum PLLs input clock frequency.*
- #define **STM32\_PLLOUT\_MAX** 24000000  
*Maximum PLL output clock frequency.*
- #define **STM32\_PLLOUT\_MIN** 16000000  
*Maximum PLL output clock frequency.*
- #define **STM32\_PCLK1\_MAX** 24000000  
*Maximum APB1 clock frequency.*
- #define **STM32\_PCLK2\_MAX** 24000000  
*Maximum APB2 clock frequency.*
- #define **STM32\_ADCCLK\_MAX** 12000000  
*Maximum ADC clock frequency.*

### RCC\_CFGR register bits definitions

- #define STM32\_SW\_HSI (0 << 0)
- #define STM32\_SW\_HSE (1 << 0)
- #define STM32\_SW\_PLL (2 << 0)
- #define STM32\_HPRE\_DIV1 (0 << 4)
- #define STM32\_HPRE\_DIV2 (8 << 4)
- #define STM32\_HPRE\_DIV4 (9 << 4)
- #define STM32\_HPRE\_DIV8 (10 << 4)
- #define STM32\_HPRE\_DIV16 (11 << 4)
- #define STM32\_HPRE\_DIV64 (12 << 4)
- #define STM32\_HPRE\_DIV128 (13 << 4)
- #define STM32\_HPRE\_DIV256 (14 << 4)
- #define STM32\_HPRE\_DIV512 (15 << 4)
- #define STM32\_PPRE1\_DIV1 (0 << 8)
- #define STM32\_PPRE1\_DIV2 (4 << 8)
- #define STM32\_PPRE1\_DIV4 (5 << 8)
- #define STM32\_PPRE1\_DIV8 (6 << 8)
- #define STM32\_PPRE1\_DIV16 (7 << 8)
- #define STM32\_PPRE2\_DIV1 (0 << 11)
- #define STM32\_PPRE2\_DIV2 (4 << 11)
- #define STM32\_PPRE2\_DIV4 (5 << 11)
- #define STM32\_PPRE2\_DIV8 (6 << 11)
- #define STM32\_PPRE2\_DIV16 (7 << 11)
- #define STM32\_ADCPRE\_DIV2 (0 << 14)
- #define STM32\_ADCPRE\_DIV4 (1 << 14)
- #define STM32\_ADCPRE\_DIV6 (2 << 14)
- #define STM32\_ADCPRE\_DIV8 (3 << 14)
- #define STM32\_PLLSRC\_HSI (0 << 16)
- #define STM32\_PLLSRC\_HSE (1 << 16)
- #define STM32\_PLLXTPRE\_DIV1 (0 << 17)
- #define STM32\_PLLXTPRE\_DIV2 (1 << 17)
- #define STM32\_MCOSEL\_NOCLOCK (0 << 24)
- #define STM32\_MCOSEL\_SYSCLK (4 << 24)
- #define STM32\_MCOSEL\_HSI (5 << 24)
- #define STM32\_MCOSEL\_HSE (6 << 24)
- #define STM32\_MCOSEL\_PLLDIV2 (7 << 24)
- #define STM32\_RTCSEL\_MASK (3 << 8)
- #define STM32\_RTCSEL\_NOCLOCK (0 << 8)
- #define STM32\_RTCSEL\_LSE (1 << 8)
- #define STM32\_RTCSEL\_LSI (2 << 8)
- #define STM32\_RTCSEL\_HSEDIV (3 << 8)

### STM32F100 MD capabilities

- #define STM32\_HAS\_ADC1 TRUE
- #define STM32\_HAS\_ADC2 FALSE
- #define STM32\_HAS\_ADC3 FALSE
- #define STM32\_HAS\_CAN1 FALSE
- #define STM32\_HAS\_CAN2 FALSE
- #define STM32\_HAS\_DAC TRUE
- #define STM32\_ADVANCED\_DMA FALSE
- #define STM32\_HAS\_DMA1 TRUE
- #define STM32\_HAS\_DMA2 FALSE
- #define STM32\_HAS\_ETH FALSE
- #define STM32\_EXTI\_NUM\_CHANNELS 18
- #define STM32\_HAS\_GPIOA TRUE
- #define STM32\_HAS\_GPIOB TRUE
- #define STM32\_HAS\_GPIOC TRUE
- #define STM32\_HAS\_GPIOD TRUE
- #define STM32\_HAS\_GPIOE TRUE
- #define STM32\_HAS\_GPIOF FALSE
- #define STM32\_HAS\_GPIOG FALSE
- #define STM32\_HAS\_GPIOH FALSE
- #define STM32\_HAS\_GPIOI FALSE

```
• #define STM32_HAS_I2C1 TRUE
• #define STM32_I2C1_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 7))
• #define STM32_I2C1_RX_DMA_CHN 0x00000000
• #define STM32_I2C1_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 6))
• #define STM32_I2C1_TX_DMA_CHN 0x00000000
• #define STM32_HAS_I2C2 FALSE
• #define STM32_I2C2_RX_DMA_MSK 0
• #define STM32_I2C2_RX_DMA_CHN 0x00000000
• #define STM32_I2C2_TX_DMA_MSK 0
• #define STM32_I2C2_TX_DMA_CHN 0x00000000
• #define STM32_HAS_I2C3 FALSE
• #define STM32_SPI3_RX_DMA_MSK 0
• #define STM32_SPI3_RX_DMA_CHN 0x00000000
• #define STM32_SPI3_TX_DMA_MSK 0
• #define STM32_SPI3_TX_DMA_CHN 0x00000000
• #define STM32_HAS_RTC TRUE
• #define STM32_HAS_SDIO FALSE
• #define STM32_HAS_SPI1 TRUE
• #define STM32_SPI1_RX_DMA_MSK STM32_DMA_STREAM_ID_MSK(1, 2)
• #define STM32_SPI1_RX_DMA_CHN 0x00000000
• #define STM32_SPI1_TX_DMA_MSK STM32_DMA_STREAM_ID_MSK(1, 3)
• #define STM32_SPI1_TX_DMA_CHN 0x00000000
• #define STM32_HAS_SPI2 FALSE
• #define STM32_SPI2_RX_DMA_MSK 0
• #define STM32_SPI2_RX_DMA_CHN 0x00000000
• #define STM32_SPI2_TX_DMA_MSK 0
• #define STM32_SPI2_TX_DMA_CHN 0x00000000
• #define STM32_HAS_SPI3 FALSE
• #define STM32_SPI3_RX_DMA_MSK 0
• #define STM32_SPI3_RX_DMA_CHN 0x00000000
• #define STM32_SPI3_TX_DMA_MSK 0
• #define STM32_SPI3_TX_DMA_CHN 0x00000000
• #define STM32_HAS_TIM1 TRUE
• #define STM32_HAS_TIM2 TRUE
• #define STM32_HAS_TIM3 TRUE
• #define STM32_HAS_TIM4 FALSE
• #define STM32_HAS_TIM5 FALSE
• #define STM32_HAS_TIM6 TRUE
• #define STM32_HAS_TIM7 TRUE
• #define STM32_HAS_TIM8 FALSE
• #define STM32_HAS_TIM9 FALSE
• #define STM32_HAS_TIM10 FALSE
• #define STM32_HAS_TIM11 FALSE
• #define STM32_HAS_TIM12 FALSE
• #define STM32_HAS_TIM13 FALSE
• #define STM32_HAS_TIM14 FALSE
• #define STM32_HAS_TIM15 TRUE
• #define STM32_HAS_TIM16 TRUE
• #define STM32_HAS_TIM17 TRUE
• #define STM32_HAS_USART1 TRUE
• #define STM32_USART1_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 5))
• #define STM32_USART1_RX_DMA_CHN 0x00000000
• #define STM32_USART1_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 4))
• #define STM32_USART1_TX_DMA_CHN 0x00000000
• #define STM32_HAS_USART2 TRUE
• #define STM32_HAS_USART3 FALSE
• #define STM32_USART3_RX_DMA_MSK 0
• #define STM32_USART3_RX_DMA_CHN 0x00000000
• #define STM32_USART3_TX_DMA_MSK 0
• #define STM32_USART3_TX_DMA_CHN 0x00000000
• #define STM32_HAS_UART4 FALSE
• #define STM32_UART4_RX_DMA_MSK 0
• #define STM32_UART4_RX_DMA_CHN 0x00000000
• #define STM32_UART4_TX_DMA_MSK 0
• #define STM32_UART4_TX_DMA_CHN 0x00000000
```

```
• #define STM32_HAS_UART5 FALSE
• #define STM32_UART5_RX_DMA_MSK 0
• #define STM32_UART5_RX_DMA_CHN 0x00000000
• #define STM32_UART5_TX_DMA_MSK 0
• #define STM32_UART5_TX_DMA_CHN 0x00000000
• #define STM32_HAS_USART6 FALSE
• #define STM32_USART6_RX_DMA_MSK 0
• #define STM32_USART6_RX_DMA_CHN 0x00000000
• #define STM32_USART6_TX_DMA_MSK 0
• #define STM32_USART6_TX_DMA_CHN 0x00000000
• #define STM32_HAS_USB FALSE
• #define STM32_HAS_OTG1 FALSE
• #define STM32_HAS_OTG2 FALSE
• #define STM32_HAS_ADC1 TRUE
• #define STM32_HAS_ADC2 FALSE
• #define STM32_HAS_ADC3 FALSE
• #define STM32_HAS_CAN1 FALSE
• #define STM32_HAS_CAN2 FALSE
• #define STM32_HAS_DAC TRUE
• #define STM32_ADVANCED_DMA FALSE
• #define STM32_HAS_DMA1 TRUE
• #define STM32_HAS_DMA2 FALSE
• #define STM32_HAS_ETH FALSE
• #define STM32_EXTI_NUM_CHANNELS 19
• #define STM32_HAS_GPIOA TRUE
• #define STM32_HAS_GPIOB TRUE
• #define STM32_HAS_GPIOC TRUE
• #define STM32_HAS_GPIOD TRUE
• #define STM32_HAS_GPIOE TRUE
• #define STM32_HAS_GPIOF FALSE
• #define STM32_HAS_GPIOG FALSE
• #define STM32_HAS_GPIOH FALSE
• #define STM32_HAS_GPIOI FALSE
• #define STM32_HAS_I2C1 TRUE
• #define STM32_I2C1_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 7))
• #define STM32_I2C1_RX_DMA_CHN 0x00000000
• #define STM32_I2C1_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 6))
• #define STM32_I2C1_TX_DMA_CHN 0x00000000
• #define STM32_HAS_I2C2 TRUE
• #define STM32_I2C2_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 5))
• #define STM32_I2C2_RX_DMA_CHN 0x00000000
• #define STM32_I2C2_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 4))
• #define STM32_I2C2_TX_DMA_CHN 0x00000000
• #define STM32_HAS_I2C3 FALSE
• #define STM32_I2C3_RX_DMA_MSK 0
• #define STM32_I2C3_RX_DMA_CHN 0x00000000
• #define STM32_I2C3_TX_DMA_MSK 0
• #define STM32_I2C3_TX_DMA_CHN 0x00000000
• #define STM32_HAS_RTC TRUE
• #define STM32_HAS_SDIO FALSE
• #define STM32_HAS_SPI1 TRUE
• #define STM32_SPI1_RX_DMA_MSK STM32_DMA_STREAM_ID_MSK(1, 2)
• #define STM32_SPI1_RX_DMA_CHN 0x00000000
• #define STM32_SPI1_TX_DMA_MSK STM32_DMA_STREAM_ID_MSK(1, 3)
• #define STM32_SPI1_TX_DMA_CHN 0x00000000
• #define STM32_HAS_SPI2 TRUE
• #define STM32_SPI2_RX_DMA_MSK STM32_DMA_STREAM_ID_MSK(1, 4)
• #define STM32_SPI2_RX_DMA_CHN 0x00000000
• #define STM32_SPI2_TX_DMA_MSK STM32_DMA_STREAM_ID_MSK(1, 5)
• #define STM32_SPI2_TX_DMA_CHN 0x00000000
• #define STM32_HAS_SPI3 FALSE
• #define STM32_SPI3_RX_DMA_MSK 0
• #define STM32_SPI3_RX_DMA_CHN 0x00000000
• #define STM32_SPI3_TX_DMA_MSK 0
• #define STM32_SPI3_TX_DMA_CHN 0x00000000
```

- #define **STM32\_HAS\_TIM1** TRUE
- #define **STM32\_HAS\_TIM2** TRUE
- #define **STM32\_HAS\_TIM3** TRUE
- #define **STM32\_HAS\_TIM4** TRUE
- #define **STM32\_HAS\_TIM5** FALSE
- #define **STM32\_HAS\_TIM6** TRUE
- #define **STM32\_HAS\_TIM7** TRUE
- #define **STM32\_HAS\_TIM8** FALSE
- #define **STM32\_HAS\_TIM9** FALSE
- #define **STM32\_HAS\_TIM10** FALSE
- #define **STM32\_HAS\_TIM11** FALSE
- #define **STM32\_HAS\_TIM12** FALSE
- #define **STM32\_HAS\_TIM13** FALSE
- #define **STM32\_HAS\_TIM14** FALSE
- #define **STM32\_HAS\_TIM15** TRUE
- #define **STM32\_HAS\_TIM16** TRUE
- #define **STM32\_HAS\_TIM17** TRUE
- #define **STM32\_HAS\_USART1** TRUE
- #define **STM32\_USART1\_RX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 5))
- #define **STM32\_USART1\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART1\_TX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 4))
- #define **STM32\_USART1\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_USART2** TRUE
- #define **STM32\_I2C2\_RX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 5))
- #define **STM32\_I2C2\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_I2C2\_TX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 4))
- #define **STM32\_I2C2\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_USART3** TRUE
- #define **STM32\_USART3\_RX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 3))
- #define **STM32\_USART3\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART3\_TX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 2))
- #define **STM32\_USART3\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_UART4** FALSE
- #define **STM32\_UART4\_RX\_DMA\_MSK** 0
- #define **STM32\_UART4\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_UART4\_TX\_DMA\_MSK** 0
- #define **STM32\_UART4\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_UART5** FALSE
- #define **STM32\_UART5\_RX\_DMA\_MSK** 0
- #define **STM32\_UART5\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_UART5\_TX\_DMA\_MSK** 0
- #define **STM32\_UART5\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_USART6** FALSE
- #define **STM32\_USART6\_RX\_DMA\_MSK** 0
- #define **STM32\_USART6\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART6\_TX\_DMA\_MSK** 0
- #define **STM32\_USART6\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_USB** FALSE
- #define **STM32\_HAS\_OTG1** FALSE
- #define **STM32\_HAS\_OTG2** FALSE

#### STM32F100 LD capabilities

- #define **STM32\_RTCSEL\_HAS\_SUBSECONDS** TRUE
- #define **STM32\_USART2\_RX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 6))
- #define **STM32\_USART2\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART2\_TX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 7))
- #define **STM32\_USART2\_TX\_DMA\_CHN** 0x00000000

## IRQ VECTOR names

- #define `WWDG_IRQHandler` Vector40
- #define `PVD_IRQHandler` Vector44
- #define `TAMPER_IRQHandler` Vector48
- #define `RTC_IRQHandler` Vector4C
- #define `FLASH_IRQHandler` Vector50
- #define `RCC_IRQHandler` Vector54
- #define `EXTI0_IRQHandler` Vector58
- #define `EXTI1_IRQHandler` Vector5C
- #define `EXTI2_IRQHandler` Vector60
- #define `EXTI3_IRQHandler` Vector64
- #define `EXTI4_IRQHandler` Vector68
- #define `DMA1_Ch1_IRQHandler` Vector6C
- #define `DMA1_Ch2_IRQHandler` Vector70
- #define `DMA1_Ch3_IRQHandler` Vector74
- #define `DMA1_Ch4_IRQHandler` Vector78
- #define `DMA1_Ch5_IRQHandler` Vector7C
- #define `DMA1_Ch6_IRQHandler` Vector80
- #define `DMA1_Ch7_IRQHandler` Vector84
- #define `ADC1_2_IRQHandler` Vector88
- #define `EXTI9_5_IRQHandler` Vector9C
- #define `TIM1_BRK_IRQHandler` VectorA0
- #define `TIM1_UP_IRQHandler` VectorA4
- #define `TIM1_TRG_COM_IRQHandler` VectorA8
- #define `TIM1_CC_IRQHandler` VectorAC
- #define `TIM2_IRQHandler` VectorB0
- #define `TIM3_IRQHandler` VectorB4
- #define `TIM4_IRQHandler` VectorB8
- #define `I2C1_EV_IRQHandler` VectorBC
- #define `I2C1_ER_IRQHandler` VectorC0
- #define `I2C2_EV_IRQHandler` VectorC4
- #define `I2C2_ER_IRQHandler` VectorC8
- #define `SPI1_IRQHandler` VectorCC
- #define `SPI2_IRQHandler` VectorD0
- #define `USART1_IRQHandler` VectorD4
- #define `USART2_IRQHandler` VectorD8
- #define `USART3_IRQHandler` VectorDC
- #define `EXTI15_10_IRQHandler` VectorE0
- #define `RTC_Alarm_IRQHandler` VectorE4
- #define `CEC_IRQHandler` VectorE8
- #define `TIM12_IRQHandler` VectorEC
- #define `TIM13_IRQHandler` VectorF0
- #define `TIM14_IRQHandler` VectorF4

## Configuration options

- #define `STM32_SW` STM32\_SW\_PLL  
*Main clock source selection.*
- #define `STM32_PLLSRC` STM32\_PLLSRC\_HSE  
*Clock source for the PLL.*
- #define `STM32_PLLXTPRE` STM32\_PLLXTPRE\_DIV1  
*Crystal PLL pre-divider.*
- #define `STM32_PLLMUL_VALUE` 3  
*PLL multiplier value.*
- #define `STM32_HPRE` STM32\_HPRE\_DIV1  
*AHB prescaler value.*
- #define `STM32_PPREG1` STM32\_PPREG1\_DIV1  
*APB1 prescaler value.*
- #define `STM32_PPREG2` STM32\_PPREG2\_DIV1  
*APB2 prescaler value.*
- #define `STM32_ADCPRE` STM32\_ADCPRE\_DIV2  
*ADC prescaler value.*
- #define `STM32_MCOSEL` STM32\_MCOSEL\_NOCLOCK  
*MCO pin setting.*
- #define `STM32_RTCSEL` STM32\_RTCSEL\_LSI  
*Clock source selecting. LSI by default.*

## 8.21 hal\_lld\_f103.h File Reference

### 8.21.1 Detailed Description

STM32F103 Performance Line HAL subsystem low level driver header.

#### Defines

- `#define STM32_ACTIVATE_PLL TRUE`  
*PLL activation flag.*
- `#define STM32_PLLMUL ((STM32_PLLMUL_VALUE - 2) << 18)`  
*PLLMUL field.*
- `#define STM32_PLLCLKIN (STM32_HSECLK / 1)`  
*PLL input clock frequency.*
- `#define STM32_PLLCLKOUT (STM32_PLLCLKIN * STM32_PLLMUL_VALUE)`  
*PLL output clock frequency.*
- `#define STM32_SYSCLK STM32_PLLCLKOUT`  
*System clock source.*
- `#define STM32_HCLK (STM32_SYSCLK / 1)`  
*AHB frequency.*
- `#define STM32_PCLK1 (STM32_HCLK / 1)`  
*APB1 frequency.*
- `#define STM32_PCLK2 (STM32_HCLK / 1)`  
*APB2 frequency.*
- `#define STM32_RTCCLK STM32_LSECLK`  
*RTC clock.*
- `#define STM32_ADCCLK (STM32_PCLK2 / 2)`  
*ADC frequency.*
- `#define STM32_USBCLK ((STM32_PLLCLKOUT * 2) / 3)`  
*USB frequency.*
- `#define STM32_TIMCLK1 (STM32_PCLK1 * 1)`  
*Timers 2, 3, 4, 5, 6, 7, 12, 13, 14 clock.*
- `#define STM32_TIMCLK2 (STM32_PCLK2 * 1)`  
*Timers 1, 8, 9, 10, 11 clock.*
- `#define STM32_FLASHBITS 0x00000010`  
*Flash settings.*

#### Platform identification

- `#define PLATFORM_NAME "STM32F1 Performance Line"`

#### Absolute Maximum Ratings

- `#define STM32_SYSCLK_MAX 72000000`  
*Maximum system clock frequency.*
- `#define STM32_HSECLK_MAX 25000000`  
*Maximum HSE clock frequency.*
- `#define STM32_HSECLK_MIN 1000000`  
*Minimum HSE clock frequency.*
- `#define STM32_LSECLK_MAX 1000000`  
*Maximum LSE clock frequency.*
- `#define STM32_LSECLK_MIN 32768`  
*Minimum LSE clock frequency.*
- `#define STM32_PLLIN_MAX 25000000`

- #define STM32\_PLLIN\_MIN 1000000
 

*Maximum PLLs input clock frequency.*
- #define STM32\_PLLOUT\_MAX 72000000
 

*Maximum PLLs output clock frequency.*
- #define STM32\_PLLOUT\_MIN 16000000
 

*Maximum PLL output clock frequency.*
- #define STM32\_PCLK1\_MAX 36000000
 

*Maximum APB1 clock frequency.*
- #define STM32\_PCLK2\_MAX 72000000
 

*Maximum APB2 clock frequency.*
- #define STM32\_ADCCLK\_MAX 14000000
 

*Maximum ADC clock frequency.*

#### RCC\_CFGR register bits definitions

- #define STM32\_SW\_HSI (0 << 0)
- #define STM32\_SW\_HSE (1 << 0)
- #define STM32\_SW\_PLL (2 << 0)
- #define STM32\_HPRE\_DIV1 (0 << 4)
- #define STM32\_HPRE\_DIV2 (8 << 4)
- #define STM32\_HPRE\_DIV4 (9 << 4)
- #define STM32\_HPRE\_DIV8 (10 << 4)
- #define STM32\_HPRE\_DIV16 (11 << 4)
- #define STM32\_HPRE\_DIV64 (12 << 4)
- #define STM32\_HPRE\_DIV128 (13 << 4)
- #define STM32\_HPRE\_DIV256 (14 << 4)
- #define STM32\_HPRE\_DIV512 (15 << 4)
- #define STM32\_PPREG1\_DIV1 (0 << 8)
- #define STM32\_PPREG1\_DIV2 (4 << 8)
- #define STM32\_PPREG1\_DIV4 (5 << 8)
- #define STM32\_PPREG1\_DIV8 (6 << 8)
- #define STM32\_PPREG1\_DIV16 (7 << 8)
- #define STM32\_PPREG2\_DIV1 (0 << 11)
- #define STM32\_PPREG2\_DIV2 (4 << 11)
- #define STM32\_PPREG2\_DIV4 (5 << 11)
- #define STM32\_PPREG2\_DIV8 (6 << 11)
- #define STM32\_PPREG2\_DIV16 (7 << 11)
- #define STM32\_ADCPRE\_DIV2 (0 << 14)
- #define STM32\_ADCPRE\_DIV4 (1 << 14)
- #define STM32\_ADCPRE\_DIV6 (2 << 14)
- #define STM32\_ADCPRE\_DIV8 (3 << 14)
- #define STM32\_PLLSRC\_HSI (0 << 16)
- #define STM32\_PLLSRC\_HSE (1 << 16)
- #define STM32\_PLLXTPRE\_DIV1 (0 << 17)
- #define STM32\_PLLXTPRE\_DIV2 (1 << 17)
- #define STM32\_USBPRE\_DIV1P5 (0 << 22)
- #define STM32\_USBPRE\_DIV1 (1 << 22)
- #define STM32\_MCOSEL\_NOCLOCK (0 << 24)
- #define STM32\_MCOSEL\_SYSCLK (4 << 24)
- #define STM32\_MCOSEL\_HSI (5 << 24)
- #define STM32\_MCOSEL\_HSE (6 << 24)
- #define STM32\_MCOSEL\_PLLDIV2 (7 << 24)
- #define STM32\_RTCSEL\_MASK (3 << 8)
- #define STM32\_RTCSEL\_NOCLOCK (0 << 8)
- #define STM32\_RTCSEL\_LSE (1 << 8)
- #define STM32\_RTCSEL\_LSI (2 << 8)
- #define STM32\_RTCSEL\_HSEDIV (3 << 8)

## STM32F103 XL capabilities

- #define **STM32\_HAS\_ADC1** TRUE
- #define **STM32\_HAS\_ADC2** TRUE
- #define **STM32\_HAS\_ADC3** FALSE
- #define **STM32\_HAS\_CAN1** TRUE
- #define **STM32\_HAS\_CAN2** FALSE
- #define **STM32\_HAS\_DAC** FALSE
- #define **STM32\_ADVANCED\_DMA** FALSE
- #define **STM32\_HAS\_DMA1** TRUE
- #define **STM32\_HAS\_DMA2** FALSE
- #define **STM32\_HAS\_ETH** FALSE
- #define **STM32\_EXTI\_NUM\_CHANNELS** 19
- #define **STM32\_HAS\_GPIOA** TRUE
- #define **STM32\_HAS\_GPIOB** TRUE
- #define **STM32\_HAS\_GPIOC** TRUE
- #define **STM32\_HAS\_GPIOD** TRUE
- #define **STM32\_HAS\_GPIOE** FALSE
- #define **STM32\_HAS\_GPIOF** FALSE
- #define **STM32\_HAS\_GPIOG** FALSE
- #define **STM32\_HAS\_GPIOH** FALSE
- #define **STM32\_HAS\_GPIOI** FALSE
- #define **STM32\_HAS\_I2C1** TRUE
- #define **STM32\_I2C1\_RX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 7))
- #define **STM32\_I2C1\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_I2C1\_TX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 6))
- #define **STM32\_I2C1\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_I2C2** FALSE
- #define **STM32\_I2C2\_RX\_DMA\_MSK** 0
- #define **STM32\_I2C2\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_I2C2\_TX\_DMA\_MSK** 0
- #define **STM32\_I2C2\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_I2C3** FALSE
- #define **STM32\_SPI3\_RX\_DMA\_MSK** 0
- #define **STM32\_SPI3\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_SPI3\_TX\_DMA\_MSK** 0
- #define **STM32\_SPI3\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_RTC** TRUE
- #define **STM32\_RTCSEL\_HAS\_SUBSECONDS** TRUE
- #define **STM32\_HAS\_SDIO** FALSE
- #define **STM32\_HAS\_SPI1** TRUE
- #define **STM32\_SPI1\_RX\_DMA\_MSK** STM32\_DMA\_STREAM\_ID\_MSK(1, 2)
- #define **STM32\_SPI1\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_SPI1\_TX\_DMA\_MSK** STM32\_DMA\_STREAM\_ID\_MSK(1, 3)
- #define **STM32\_SPI1\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_SPI2** FALSE
- #define **STM32\_SPI2\_RX\_DMA\_MSK** 0
- #define **STM32\_SPI2\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_SPI2\_TX\_DMA\_MSK** 0
- #define **STM32\_SPI2\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_SPI3** FALSE
- #define **STM32\_SPI3\_RX\_DMA\_MSK** 0
- #define **STM32\_SPI3\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_SPI3\_TX\_DMA\_MSK** 0
- #define **STM32\_SPI3\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_TIM1** TRUE
- #define **STM32\_HAS\_TIM2** TRUE
- #define **STM32\_HAS\_TIM3** TRUE
- #define **STM32\_HAS\_TIM4** FALSE
- #define **STM32\_HAS\_TIM5** FALSE
- #define **STM32\_HAS\_TIM6** FALSE
- #define **STM32\_HAS\_TIM7** FALSE
- #define **STM32\_HAS\_TIM8** FALSE
- #define **STM32\_HAS\_TIM9** FALSE
- #define **STM32\_HAS\_TIM10** FALSE

```
• #define STM32_HAS_TIM11 FALSE
• #define STM32_HAS_TIM12 FALSE
• #define STM32_HAS_TIM13 FALSE
• #define STM32_HAS_TIM14 FALSE
• #define STM32_HAS_TIM15 FALSE
• #define STM32_HAS_TIM16 FALSE
• #define STM32_HAS_TIM17 FALSE
• #define STM32_HAS_USART1 TRUE
• #define STM32_USART1_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 5))
• #define STM32_USART1_RX_DMA_CHN 0x00000000
• #define STM32_USART1_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 4))
• #define STM32_USART1_TX_DMA_CHN 0x00000000
• #define STM32_HAS_USART2 TRUE
• #define STM32_USART2_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 6))
• #define STM32_USART2_RX_DMA_CHN 0x00000000
• #define STM32_USART2_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 7))
• #define STM32_USART2_TX_DMA_CHN 0x00000000
• #define STM32_HAS_USART3 FALSE
• #define STM32_USART3_RX_DMA_MSK 0
• #define STM32_USART3_RX_DMA_CHN 0x00000000
• #define STM32_USART3_TX_DMA_MSK 0
• #define STM32_USART3_TX_DMA_CHN 0x00000000
• #define STM32_HAS_UART4 FALSE
• #define STM32_UART4_RX_DMA_MSK 0
• #define STM32_UART4_RX_DMA_CHN 0x00000000
• #define STM32_UART4_TX_DMA_MSK 0
• #define STM32_UART4_TX_DMA_CHN 0x00000000
• #define STM32_HAS_UART5 FALSE
• #define STM32_UART5_RX_DMA_MSK 0
• #define STM32_UART5_RX_DMA_CHN 0x00000000
• #define STM32_UART5_TX_DMA_MSK 0
• #define STM32_UART5_TX_DMA_CHN 0x00000000
• #define STM32_HAS_USART6 FALSE
• #define STM32_USART6_RX_DMA_MSK 0
• #define STM32_USART6_RX_DMA_CHN 0x00000000
• #define STM32_USART6_TX_DMA_MSK 0
• #define STM32_USART6_TX_DMA_CHN 0x00000000
• #define STM32_HAS_USB FALSE
• #define STM32_HAS_OTG1 FALSE
• #define STM32_HAS_OTG2 FALSE
• #define STM32_HAS_ADC1 TRUE
• #define STM32_HAS_ADC2 TRUE
• #define STM32_HAS_ADC3 FALSE
• #define STM32_HAS_CAN1 TRUE
• #define STM32_HAS_CAN2 FALSE
• #define STM32_HAS_DAC FALSE
• #define STM32_ADVANCED_DMA FALSE
• #define STM32_HAS_DMA1 TRUE
• #define STM32_HAS_DMA2 FALSE
• #define STM32_HAS_ETH FALSE
• #define STM32_EXTI_NUM_CHANNELS 19
• #define STM32_HAS_GPIOA TRUE
• #define STM32_HAS_GPIOB TRUE
• #define STM32_HAS_GPIOC TRUE
• #define STM32_HAS_GPIOD TRUE
• #define STM32_HAS_GPIOE TRUE
• #define STM32_HAS_GPIOF FALSE
• #define STM32_HAS_GPIOG FALSE
• #define STM32_HAS_GPIOH FALSE
• #define STM32_HAS_GPIOI FALSE
• #define STM32_HAS_I2C1 TRUE
• #define STM32_I2C1_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 7))
• #define STM32_I2C1_RX_DMA_CHN 0x00000000
• #define STM32_I2C1_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 6))
• #define STM32_I2C1_TX_DMA_CHN 0x00000000
```

```
• #define STM32_HAS_I2C2 TRUE
• #define STM32_I2C2_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 5))
• #define STM32_I2C2_RX_DMA_CHN 0x00000000
• #define STM32_I2C2_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 4))
• #define STM32_I2C2_TX_DMA_CHN 0x00000000
• #define STM32_HAS_I2C3 FALSE
• #define STM32_I2C3_RX_DMA_MSK 0
• #define STM32_I2C3_RX_DMA_CHN 0x00000000
• #define STM32_I2C3_TX_DMA_MSK 0
• #define STM32_I2C3_TX_DMA_CHN 0x00000000
• #define STM32_HAS_RTC TRUE
• #define STM32_RTCSEL_HAS_SUBSECONDS TRUE
• #define STM32_HAS_SDIO FALSE
• #define STM32_HAS_SPI1 TRUE
• #define STM32_SPI1_RX_DMA_MSK STM32_DMA_STREAM_ID_MSK(1, 2)
• #define STM32_SPI1_RX_DMA_CHN 0x00000000
• #define STM32_SPI1_TX_DMA_MSK STM32_DMA_STREAM_ID_MSK(1, 3)
• #define STM32_SPI1_TX_DMA_CHN 0x00000000
• #define STM32_HAS_SPI2 TRUE
• #define STM32_SPI2_RX_DMA_MSK STM32_DMA_STREAM_ID_MSK(1, 4)
• #define STM32_SPI2_RX_DMA_CHN 0x00000000
• #define STM32_SPI2_TX_DMA_MSK STM32_DMA_STREAM_ID_MSK(1, 5)
• #define STM32_SPI2_TX_DMA_CHN 0x00000000
• #define STM32_HAS_SPI3 FALSE
• #define STM32_SPI3_RX_DMA_MSK 0
• #define STM32_SPI3_RX_DMA_CHN 0x00000000
• #define STM32_SPI3_TX_DMA_MSK 0
• #define STM32_SPI3_TX_DMA_CHN 0x00000000
• #define STM32_HAS_TIM1 TRUE
• #define STM32_HAS_TIM2 TRUE
• #define STM32_HAS_TIM3 TRUE
• #define STM32_HAS_TIM4 TRUE
• #define STM32_HAS_TIM5 FALSE
• #define STM32_HAS_TIM6 FALSE
• #define STM32_HAS_TIM7 FALSE
• #define STM32_HAS_TIM8 FALSE
• #define STM32_HAS_TIM9 FALSE
• #define STM32_HAS_TIM10 FALSE
• #define STM32_HAS_TIM11 FALSE
• #define STM32_HAS_TIM12 FALSE
• #define STM32_HAS_TIM13 FALSE
• #define STM32_HAS_TIM14 FALSE
• #define STM32_HAS_TIM15 FALSE
• #define STM32_HAS_TIM16 FALSE
• #define STM32_HAS_TIM17 FALSE
• #define STM32_HAS_USART1 TRUE
• #define STM32_USART1_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 5))
• #define STM32_USART1_RX_DMA_CHN 0x00000000
• #define STM32_USART1_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 4))
• #define STM32_USART1_TX_DMA_CHN 0x00000000
• #define STM32_HAS_USART2 TRUE
• #define STM32_USART2_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 6))
• #define STM32_USART2_RX_DMA_CHN 0x00000000
• #define STM32_USART2_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 7))
• #define STM32_USART2_TX_DMA_CHN 0x00000000
• #define STM32_HAS_USART3 TRUE
• #define STM32_USART3_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 3))
• #define STM32_USART3_RX_DMA_CHN 0x00000000
• #define STM32_USART3_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 2))
• #define STM32_USART3_TX_DMA_CHN 0x00000000
• #define STM32_HAS_UART4 FALSE
• #define STM32_UART4_RX_DMA_MSK 0
• #define STM32_UART4_RX_DMA_CHN 0x00000000
• #define STM32_UART4_TX_DMA_MSK 0
• #define STM32_UART4_TX_DMA_CHN 0x00000000
```

```
• #define STM32_HAS_UART5 FALSE
• #define STM32_UART5_RX_DMA_MSK 0
• #define STM32_UART5_RX_DMA_CHN 0x00000000
• #define STM32_UART5_TX_DMA_MSK 0
• #define STM32_UART5_TX_DMA_CHN 0x00000000
• #define STM32_HAS_USART6 FALSE
• #define STM32_USART6_RX_DMA_MSK 0
• #define STM32_USART6_RX_DMA_CHN 0x00000000
• #define STM32_USART6_TX_DMA_MSK 0
• #define STM32_USART6_TX_DMA_CHN 0x00000000
• #define STM32_HAS_USB TRUE
• #define STM32_HAS_OTG1 FALSE
• #define STM32_HAS_OTG2 FALSE
• #define STM32_HAS_ADC1 TRUE
• #define STM32_HAS_ADC2 TRUE
• #define STM32_HAS_ADC3 TRUE
• #define STM32_HAS_CAN1 TRUE
• #define STM32_HAS_CAN2 FALSE
• #define STM32_HAS_DAC TRUE
• #define STM32_ADVANCED_DMA FALSE
• #define STM32_HAS_DMA1 TRUE
• #define STM32_HAS_DMA2 TRUE
• #define STM32_HAS_ETH FALSE
• #define STM32_EXTI_NUM_CHANNELS 19
• #define STM32_HAS_GPIOA TRUE
• #define STM32_HAS_GPIOB TRUE
• #define STM32_HAS_GPIOC TRUE
• #define STM32_HAS_GPIOD TRUE
• #define STM32_HAS_GPIOE TRUE
• #define STM32_HAS_GPIOF TRUE
• #define STM32_HAS_GPIOG TRUE
• #define STM32_HAS_GPIOH FALSE
• #define STM32_HAS_GPIOI FALSE
• #define STM32_HAS_I2C1 TRUE
• #define STM32_I2C1_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 7))
• #define STM32_I2C1_RX_DMA_CHN 0x00000000
• #define STM32_I2C1_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 6))
• #define STM32_I2C1_TX_DMA_CHN 0x00000000
• #define STM32_HAS_I2C2 TRUE
• #define STM32_I2C2_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 5))
• #define STM32_I2C2_RX_DMA_CHN 0x00000000
• #define STM32_I2C2_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 4))
• #define STM32_I2C2_TX_DMA_CHN 0x00000000
• #define STM32_HAS_I2C3 FALSE
• #define STM32_I2C3_RX_DMA_MSK 0
• #define STM32_I2C3_RX_DMA_CHN 0x00000000
• #define STM32_I2C3_TX_DMA_MSK 0
• #define STM32_I2C3_TX_DMA_CHN 0x00000000
• #define STM32_HAS_RTC TRUE
• #define STM32_RTCSEL_HAS_SUBSECONDS TRUE
• #define STM32_HAS_SDIO TRUE
• #define STM32_HAS_SPI1 TRUE
• #define STM32_SPI1_RX_DMA_MSK STM32_DMA_STREAM_ID_MSK(1, 2)
• #define STM32_SPI1_RX_DMA_CHN 0x00000000
• #define STM32_SPI1_TX_DMA_MSK STM32_DMA_STREAM_ID_MSK(1, 3)
• #define STM32_SPI1_TX_DMA_CHN 0x00000000
• #define STM32_HAS_SPI2 TRUE
• #define STM32_SPI2_RX_DMA_MSK STM32_DMA_STREAM_ID_MSK(1, 4)
• #define STM32_SPI2_RX_DMA_CHN 0x00000000
• #define STM32_SPI2_TX_DMA_MSK STM32_DMA_STREAM_ID_MSK(1, 5)
• #define STM32_SPI2_TX_DMA_CHN 0x00000000
• #define STM32_HAS_SPI3 TRUE
• #define STM32_SPI3_RX_DMA_MSK STM32_DMA_STREAM_ID_MSK(2, 1)
• #define STM32_SPI3_RX_DMA_CHN 0x00000000
• #define STM32_SPI3_TX_DMA_MSK STM32_DMA_STREAM_ID_MSK(2, 2)
```

```
• #define STM32_SPI3_TX_DMA_CHN 0x00000000
• #define STM32_HAS_TIM1 TRUE
• #define STM32_HAS_TIM2 TRUE
• #define STM32_HAS_TIM3 TRUE
• #define STM32_HAS_TIM4 TRUE
• #define STM32_HAS_TIM5 TRUE
• #define STM32_HAS_TIM6 TRUE
• #define STM32_HAS_TIM7 TRUE
• #define STM32_HAS_TIM8 TRUE
• #define STM32_HAS_TIM9 TRUE
• #define STM32_HAS_TIM10 TRUE
• #define STM32_HAS_TIM11 TRUE
• #define STM32_HAS_TIM12 TRUE
• #define STM32_HAS_TIM13 TRUE
• #define STM32_HAS_TIM14 TRUE
• #define STM32_HAS_TIM15 FALSE
• #define STM32_HAS_TIM16 FALSE
• #define STM32_HAS_TIM17 FALSE
• #define STM32_HAS_USART1 TRUE
• #define STM32_USART1_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 5))
• #define STM32_USART1_RX_DMA_CHN 0x00000000
• #define STM32_USART1_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 4))
• #define STM32_USART1_TX_DMA_CHN 0x00000000
• #define STM32_HAS_USART2 TRUE
• #define STM32_USART2_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 6))
• #define STM32_USART2_RX_DMA_CHN 0x00000000
• #define STM32_USART2_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 7))
• #define STM32_USART2_TX_DMA_CHN 0x00000000
• #define STM32_HAS_USART3 TRUE
• #define STM32_USART3_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 3))
• #define STM32_USART3_RX_DMA_CHN 0x00000000
• #define STM32_USART3_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 2))
• #define STM32_USART3_TX_DMA_CHN 0x00000000
• #define STM32_HAS_UART4 TRUE
• #define STM32_UART4_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(2, 3))
• #define STM32_UART4_RX_DMA_CHN 0x00000000
• #define STM32_UART4_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(2, 5))
• #define STM32_UART4_TX_DMA_CHN 0x00000000
• #define STM32_HAS_UART5 TRUE
• #define STM32_UART5_RX_DMA_MSK 0
• #define STM32_UART5_RX_DMA_CHN 0x00000000
• #define STM32_UART5_TX_DMA_MSK 0
• #define STM32_UART5_TX_DMA_CHN 0x00000000
• #define STM32_HAS_USART6 FALSE
• #define STM32_USART6_RX_DMA_MSK 0
• #define STM32_USART6_RX_DMA_CHN 0x00000000
• #define STM32_USART6_TX_DMA_MSK 0
• #define STM32_USART6_TX_DMA_CHN 0x00000000
• #define STM32_HAS_USB TRUE
• #define STM32_HAS_OTG1 FALSE
• #define STM32_HAS_OTG2 FALSE
• #define STM32_HAS_ADC1 TRUE
• #define STM32_HAS_ADC2 TRUE
• #define STM32_HAS_ADC3 TRUE
• #define STM32_HAS_CAN1 TRUE
• #define STM32_HAS_CAN2 FALSE
• #define STM32_HAS_DAC TRUE
• #define STM32_ADVANCED_DMA FALSE
• #define STM32_HAS_DMA1 TRUE
• #define STM32_HAS_DMA2 TRUE
• #define STM32_HAS_ETH FALSE
• #define STM32_EXTI_NUM_CHANNELS 19
• #define STM32_HAS_GPIOA TRUE
• #define STM32_HAS_GPIOB TRUE
• #define STM32_HAS_GPIOC TRUE
```

```
• #define STM32_HAS_GPIOD TRUE
• #define STM32_HAS_GPIOE TRUE
• #define STM32_HAS_GPIOF TRUE
• #define STM32_HAS_GPIOG TRUE
• #define STM32_HAS_GPIOH FALSE
• #define STM32_HAS_GPIOI FALSE
• #define STM32_HAS_I2C1 TRUE
• #define STM32_I2C1_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 7))
• #define STM32_I2C1_RX_DMA_CHN 0x00000000
• #define STM32_I2C1_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 6))
• #define STM32_I2C1_TX_DMA_CHN 0x00000000
• #define STM32_HAS_I2C2 TRUE
• #define STM32_I2C2_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 5))
• #define STM32_I2C2_RX_DMA_CHN 0x00000000
• #define STM32_I2C2_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 4))
• #define STM32_I2C2_TX_DMA_CHN 0x00000000
• #define STM32_HAS_I2C3 FALSE
• #define STM32_I2C3_RX_DMA_MSK 0
• #define STM32_I2C3_RX_DMA_CHN 0x00000000
• #define STM32_I2C3_TX_DMA_MSK 0
• #define STM32_I2C3_TX_DMA_CHN 0x00000000
• #define STM32_HAS_RTC TRUE
• #define STM32_RTCSEL_HAS_SUBSECONDS TRUE
• #define STM32_HAS_SDIO TRUE
• #define STM32_HAS_SPI1 TRUE
• #define STM32_SPI1_RX_DMA_MSK STM32_DMA_STREAM_ID_MSK(1, 2)
• #define STM32_SPI1_RX_DMA_CHN 0x00000000
• #define STM32_SPI1_TX_DMA_MSK STM32_DMA_STREAM_ID_MSK(1, 3)
• #define STM32_SPI1_TX_DMA_CHN 0x00000000
• #define STM32_HAS_SPI2 TRUE
• #define STM32_SPI2_RX_DMA_MSK STM32_DMA_STREAM_ID_MSK(1, 4)
• #define STM32_SPI2_RX_DMA_CHN 0x00000000
• #define STM32_SPI2_TX_DMA_MSK STM32_DMA_STREAM_ID_MSK(1, 5)
• #define STM32_SPI2_TX_DMA_CHN 0x00000000
• #define STM32_HAS_SPI3 TRUE
• #define STM32_SPI3_RX_DMA_MSK STM32_DMA_STREAM_ID_MSK(2, 1)
• #define STM32_SPI3_RX_DMA_CHN 0x00000000
• #define STM32_SPI3_TX_DMA_MSK STM32_DMA_STREAM_ID_MSK(2, 2)
• #define STM32_SPI3_TX_DMA_CHN 0x00000000
• #define STM32_HAS_TIM1 TRUE
• #define STM32_HAS_TIM2 TRUE
• #define STM32_HAS_TIM3 TRUE
• #define STM32_HAS_TIM4 TRUE
• #define STM32_HAS_TIM5 TRUE
• #define STM32_HAS_TIM6 TRUE
• #define STM32_HAS_TIM7 TRUE
• #define STM32_HAS_TIM8 TRUE
• #define STM32_HAS_TIM9 FALSE
• #define STM32_HAS_TIM10 FALSE
• #define STM32_HAS_TIM11 FALSE
• #define STM32_HAS_TIM12 FALSE
• #define STM32_HAS_TIM13 FALSE
• #define STM32_HAS_TIM14 FALSE
• #define STM32_HAS_TIM15 FALSE
• #define STM32_HAS_TIM16 FALSE
• #define STM32_HAS_TIM17 FALSE
• #define STM32_HAS_USART1 TRUE
• #define STM32_USART1_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 5))
• #define STM32_USART1_RX_DMA_CHN 0x00000000
• #define STM32_USART1_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 4))
• #define STM32_USART1_TX_DMA_CHN 0x00000000
• #define STM32_HAS_USART2 TRUE
• #define STM32_USART2_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 6))
• #define STM32_USART2_RX_DMA_CHN 0x00000000
• #define STM32_USART2_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 7))
```

- #define **STM32\_USART2\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_USART3** TRUE
- #define **STM32\_USART3\_RX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 3))
- #define **STM32\_USART3\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART3\_TX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(1, 2))
- #define **STM32\_USART3\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_UART4** TRUE
- #define **STM32\_UART4\_RX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(2, 3))
- #define **STM32\_UART4\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_UART4\_TX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(2, 5))
- #define **STM32\_UART4\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_UART5** TRUE
- #define **STM32\_UART5\_RX\_DMA\_MSK** 0
- #define **STM32\_UART5\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_UART5\_TX\_DMA\_MSK** 0
- #define **STM32\_UART5\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_USART6** FALSE
- #define **STM32\_USART6\_RX\_DMA\_MSK** 0
- #define **STM32\_USART6\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART6\_TX\_DMA\_MSK** 0
- #define **STM32\_USART6\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_USB** TRUE
- #define **STM32\_HAS\_OTG1** FALSE
- #define **STM32\_HAS\_OTG2** FALSE

#### IRQ VECTOR names

- #define **WWDG\_IRQHandler** Vector40
- #define **PVD\_IRQHandler** Vector44
- #define **TAMPER\_IRQHandler** Vector48
- #define **RTC\_IRQHandler** Vector4C
- #define **FLASH\_IRQHandler** Vector50
- #define **RCC\_IRQHandler** Vector54
- #define **EXTI0\_IRQHandler** Vector58
- #define **EXTI1\_IRQHandler** Vector5C
- #define **EXTI2\_IRQHandler** Vector60
- #define **EXTI3\_IRQHandler** Vector64
- #define **EXTI4\_IRQHandler** Vector68
- #define **DMA1\_Ch1\_IRQHandler** Vector6C
- #define **DMA1\_Ch2\_IRQHandler** Vector70
- #define **DMA1\_Ch3\_IRQHandler** Vector74
- #define **DMA1\_Ch4\_IRQHandler** Vector78
- #define **DMA1\_Ch5\_IRQHandler** Vector7C
- #define **DMA1\_Ch6\_IRQHandler** Vector80
- #define **DMA1\_Ch7\_IRQHandler** Vector84
- #define **ADC1\_2\_IRQHandler** Vector88
- #define **CAN1\_TX\_IRQHandler** Vector8C
- #define **USB\_HP\_IRQHandler** Vector8C
- #define **CAN1\_RX0\_IRQHandler** Vector90
- #define **USB\_LP\_IRQHandler** Vector90
- #define **CAN1\_RX1\_IRQHandler** Vector94
- #define **CAN1\_SCE\_IRQHandler** Vector98
- #define **EXTI9\_5\_IRQHandler** Vector9C
- #define **TIM1\_BRK\_IRQHandler** VectorA0
- #define **TIM1\_UP\_IRQHandler** VectorA4
- #define **TIM1\_TRG\_COM\_IRQHandler** VectorA8
- #define **TIM1\_CC\_IRQHandler** VectorAC
- #define **TIM2\_IRQHandler** VectorB0
- #define **TIM3\_IRQHandler** VectorB4
- #define **TIM4\_IRQHandler** VectorB8
- #define **I2C1\_EV\_IRQHandler** VectorBC
- #define **I2C1\_ER\_IRQHandler** VectorC0
- #define **I2C2\_EV\_IRQHandler** VectorC4
- #define **I2C2\_ER\_IRQHandler** VectorC8
- #define **SPI1\_IRQHandler** VectorCC

- #define SPI2\_IRQHandler VectorD0
- #define USART1\_IRQHandler VectorD4
- #define USART2\_IRQHandler VectorD8
- #define USART3\_IRQHandler VectorDC
- #define EXTI15\_10\_IRQHandler VectorE0
- #define RTC\_Alarm\_IRQHandler VectorE4
- #define USB\_FS\_WKUP\_IRQHandler VectorE8
- #define TIM8\_BRK\_IRQHandler VectorEC
- #define TIM8\_UP\_IRQHandler VectorF0
- #define TIM8\_TRG\_COM\_IRQHandler VectorF4
- #define TIM8\_CC\_IRQHandler VectorF8
- #define ADC3\_IRQHandler VectorFC
- #define FSMC\_IRQHandler Vector100
- #define SDIO\_IRQHandler Vector104
- #define TIM5\_IRQHandler Vector108
- #define SPI3\_IRQHandler Vector10C
- #define UART4\_IRQHandler Vector110
- #define UART5\_IRQHandler Vector114
- #define TIM6\_IRQHandler Vector118
- #define TIM7\_IRQHandler Vector11C
- #define DMA2\_Ch1\_IRQHandler Vector120
- #define DMA2\_Ch2\_IRQHandler Vector124
- #define DMA2\_Ch3\_IRQHandler Vector128
- #define DMA2\_Ch4\_5\_IRQHandler Vector12C

#### Configuration options

- #define STM32\_SW STM32\_SW\_PLL  
*Main clock source selection.*
- #define STM32\_PLLSRC STM32\_PLLSRC\_HSE  
*Clock source for the PLL.*
- #define STM32\_PLLXTPRE STM32\_PLLXTPRE\_DIV1  
*Crystal PLL pre-divider.*
- #define STM32\_PLLMUL\_VALUE 9  
*PLL multiplier value.*
- #define STM32\_HPRE STM32\_HPRE\_DIV1  
*AHB prescaler value.*
- #define STM32\_PPREG1 STM32\_PPREG1\_DIV2  
*APB1 prescaler value.*
- #define STM32\_PPREG2 STM32\_PPREG2\_DIV2  
*APB2 prescaler value.*
- #define STM32\_ADCPRE STM32\_ADCPRE\_DIV4  
*ADC prescaler value.*
- #define STM32\_USB\_CLOCK\_REQUIRED TRUE  
*USB clock setting.*
- #define STM32\_USBPRE STM32\_USBPRE\_DIV1P5  
*USB prescaler initialization.*
- #define STM32\_MCOSEL STM32\_MCOSEL\_NOCLOCK  
*MCO pin setting.*
- #define STM32\_RTCSEL STM32\_RTCSEL\_LSI  
*Clock source selecting. LSI by default.*

## 8.22 hal\_ll\_f105\_f107.h File Reference

### 8.22.1 Detailed Description

STM32F10x Connectivity Line HAL subsystem low level driver header.

## Defines

- #define STM32\_ACTIVATE\_PLL1 TRUE  
*PLL1 activation flag.*
- #define STM32\_ACTIVATE\_PLL2 TRUE  
*PLL2 activation flag.*
- #define STM32\_ACTIVATE\_PLL3 TRUE  
*PLL3 activation flag.*
- #define STM32\_PREDIV1 ((STM32\_PREDIV1\_VALUE - 1) << 0)  
*PREDIV1 field.*
- #define STM32\_PREDIV2 ((STM32\_PREDIV2\_VALUE - 1) << 4)  
*PREDIV2 field.*
- #define STM32\_PLLMUL ((STM32\_PLLMUL\_VALUE - 2) << 18)  
*PLLMUL field.*
- #define STM32\_PLL2MUL ((STM32\_PLL2MUL\_VALUE - 2) << 8)  
*PLL2MUL field.*
- #define STM32\_PLL3MUL ((STM32\_PLL3MUL\_VALUE - 2) << 12)  
*PLL3MUL field.*
- #define STM32\_PLL2CLKIN (STM32\_HSECLK / STM32\_PREDIV2\_VALUE)  
*PLL2 input frequency.*
- #define STM32\_PLL2CLKOUT (STM32\_PLL2CLKIN \* STM32\_PLL2MUL\_VALUE)  
*PLL2 output clock frequency.*
- #define STM32\_PLL2VCO (STM32\_PLL2CLKOUT \* 2)  
*PLL2 VCO clock frequency.*
- #define STM32\_PLL3CLKIN (STM32\_HSECLK / STM32\_PREDIV2\_VALUE)  
*PLL3 input frequency.*
- #define STM32\_PLL3CLKOUT (STM32\_PLL3CLKIN \* STM32\_PLL3MUL\_VALUE)  
*PLL3 output clock frequency.*
- #define STM32\_PLL3VCO (STM32\_PLL3CLKOUT \* 2)  
*PLL3 VCO clock frequency.*
- #define STM32\_PREDIV1CLK STM32\_HSECLK  
*PREDIV1 input frequency.*
- #define STM32\_PLLCLKIN (STM32\_PREDIV1CLK / STM32\_PREDIV1\_VALUE)  
*PLL input clock frequency.*
- #define STM32\_PLLCLKOUT (STM32\_PLLCLKIN \* STM32\_PLLMUL\_VALUE)  
*PLL output clock frequency.*
- #define STM32\_PLLVCO (STM32\_PLLCLKOUT \* 2)  
*PLL VCO clock frequency.*
- #define STM32\_SYSCLK STM32\_PLLCLKOUT  
*System clock source.*
- #define STM32\_HCLK (STM32\_SYSCLK / 1)  
*AHB frequency.*
- #define STM32\_PCLK1 (STM32\_HCLK / 1)  
*APB1 frequency.*
- #define STM32\_PCLK2 (STM32\_HCLK / 1)  
*APB2 frequency.*
- #define STM32\_RTCCLK STM32\_LSECLK  
*RTC clock.*
- #define STM32\_ADCCLK (STM32\_PCLK2 / 2)  
*ADC frequency.*
- #define STM32\_OTGFSCLK (STM32\_PLLVCO / 3)

- #define **STM32\_TIMCLK1** (STM32\_PCLK1 \* 1)  
*Timers 2, 3, 4, 5, 6, 7 clock.*
- #define **STM32\_TIMCLK2** (STM32\_PCLK2 \* 1)  
*Timers 1, 8 clock.*
- #define **STM32\_FLASHBITS** 0x00000010  
*Flash settings.*

#### Platform identification

- #define **PLATFORM\_NAME** "STM32F1 Connectivity Line"

#### Absolute Maximum Ratings

- #define **STM32\_SYSCLK\_MAX** 72000000  
*Maximum system clock frequency.*
- #define **STM32\_HSECLK\_MAX** 50000000  
*Maximum HSE clock frequency.*
- #define **STM32\_HSECLK\_MIN** 1000000  
*Minimum HSE clock frequency.*
- #define **STM32\_LSECLK\_MAX** 1000000  
*Maximum LSE clock frequency.*
- #define **STM32\_LSECLK\_MIN** 32768  
*Minimum LSE clock frequency.*
- #define **STM32\_PLL1IN\_MAX** 12000000  
*Maximum PLLs input clock frequency.*
- #define **STM32\_PLL1IN\_MIN** 3000000  
*Maximum PLL1 input clock frequency.*
- #define **STM32\_PLL23IN\_MAX** 5000000  
*Maximum PLL1 input clock frequency.*
- #define **STM32\_PLL23IN\_MIN** 3000000  
*Maximum PLL2 and PLL3 input clock frequency.*
- #define **STM32\_PLL1VCO\_MAX** 144000000  
*Maximum PLL1 VCO clock frequency.*
- #define **STM32\_PLL1VCO\_MIN** 36000000  
*Maximum PLL1 VCO clock frequency.*
- #define **STM32\_PLL23VCO\_MAX** 148000000  
*Maximum PLL2 and PLL3 VCO clock frequency.*
- #define **STM32\_PLL23VCO\_MIN** 80000000  
*Maximum PLL2 and PLL3 VCO clock frequency.*
- #define **STM32\_PCLK1\_MAX** 36000000  
*Maximum APB1 clock frequency.*
- #define **STM32\_PCLK2\_MAX** 72000000  
*Maximum APB2 clock frequency.*
- #define **STM32\_ADCCLK\_MAX** 14000000  
*Maximum ADC clock frequency.*
- #define **STM32\_SPII2S\_MAX** 18000000  
*Maximum SPI/I2S clock frequency.*

#### RCC\_CFGR register bits definitions

- #define **STM32\_SW\_HSI** (0 << 0)
- #define **STM32\_SW\_HSE** (1 << 0)
- #define **STM32\_SW\_PLL** (2 << 0)
- #define **STM32\_HPRE\_DIV1** (0 << 4)
- #define **STM32\_HPRE\_DIV2** (8 << 4)
- #define **STM32\_HPRE\_DIV4** (9 << 4)
- #define **STM32\_HPRE\_DIV8** (10 << 4)
- #define **STM32\_HPRE\_DIV16** (11 << 4)

- #define STM32\_HPRE\_DIV64 (12 << 4)
- #define STM32\_HPRE\_DIV128 (13 << 4)
- #define STM32\_HPRE\_DIV256 (14 << 4)
- #define STM32\_HPRE\_DIV512 (15 << 4)
- #define STM32\_PPREG1\_DIV1 (0 << 8)
- #define STM32\_PPREG1\_DIV2 (4 << 8)
- #define STM32\_PPREG1\_DIV4 (5 << 8)
- #define STM32\_PPREG1\_DIV8 (6 << 8)
- #define STM32\_PPREG1\_DIV16 (7 << 8)
- #define STM32\_PPREG2\_DIV1 (0 << 11)
- #define STM32\_PPREG2\_DIV2 (4 << 11)
- #define STM32\_PPREG2\_DIV4 (5 << 11)
- #define STM32\_PPREG2\_DIV8 (6 << 11)
- #define STM32\_PPREG2\_DIV16 (7 << 11)
- #define STM32\_ADCPRE\_DIV2 (0 << 14)
- #define STM32\_ADCPRE\_DIV4 (1 << 14)
- #define STM32\_ADCPRE\_DIV6 (2 << 14)
- #define STM32\_ADCPRE\_DIV8 (3 << 14)
- #define STM32\_PLLSRC\_HSI (0 << 16)
- #define STM32\_PLLSRC\_PREDIV1 (1 << 16)
- #define STM32\_OTGFSPRE\_DIV2 (1 << 22)
- #define STM32\_OTGFSPRE\_DIV3 (0 << 22)
- #define STM32\_MCOSEL\_NOCLOCK (0 << 24)
- #define STM32\_MCOSEL\_SYSCLK (4 << 24)
- #define STM32\_MCOSEL\_HSI (5 << 24)
- #define STM32\_MCOSEL\_HSE (6 << 24)
- #define STM32\_MCOSEL\_PLLDIV2 (7 << 24)
- #define STM32\_MCOSEL\_PLL2 (8 << 24)
- #define STM32\_MCOSEL\_PLL3DIV2 (9 << 24)
- #define STM32\_MCOSEL\_XT1 (10 << 24)
- #define STM32\_MCOSEL\_PLL3 (11 << 24)
- #define STM32\_RTCSEL\_MASK (3 << 8)
- #define STM32\_RTCSEL\_NOCLOCK (0 << 8)
- #define STM32\_RTCSEL\_LSE (1 << 8)
- #define STM32\_RTCSEL\_LSI (2 << 8)
- #define STM32\_RTCSEL\_HSEDIV (3 << 8)

#### RCC\_CFGR2 register bits definitions

- #define STM32\_PREDIV1SRC\_HSE (0 << 16)
- #define STM32\_PREDIV1SRC\_PLL2 (1 << 16)

#### STM32F105/F107 CL capabilities

- #define STM32\_HAS\_ADC1 TRUE
- #define STM32\_HAS\_ADC2 TRUE
- #define STM32\_HAS\_ADC3 FALSE
- #define STM32\_HAS\_CAN1 TRUE
- #define STM32\_HAS\_CAN2 TRUE
- #define STM32\_HAS\_DAC TRUE
- #define STM32\_ADVANCED\_DMA FALSE
- #define STM32\_HAS\_DMA1 TRUE
- #define STM32\_HAS\_DMA2 TRUE
- #define STM32\_HAS\_ETH TRUE
- #define STM32\_EXTI\_NUM\_CHANNELS 20
- #define STM32\_HAS\_GPIOA TRUE
- #define STM32\_HAS\_GPIOB TRUE
- #define STM32\_HAS\_GPIOC TRUE
- #define STM32\_HAS\_GPIOD TRUE
- #define STM32\_HAS\_GPIOE TRUE
- #define STM32\_HAS\_GPIOF FALSE
- #define STM32\_HAS\_GPIOG FALSE
- #define STM32\_HAS\_GPIOH FALSE
- #define STM32\_HAS\_GPIOI FALSE

```
• #define STM32_HAS_I2C1 TRUE
• #define STM32_I2C1_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 7))
• #define STM32_I2C1_RX_DMA_CHN 0x00000000
• #define STM32_I2C1_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 6))
• #define STM32_I2C1_TX_DMA_CHN 0x00000000
• #define STM32_HAS_I2C2 TRUE
• #define STM32_I2C2_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 5))
• #define STM32_I2C2_RX_DMA_CHN 0x00000000
• #define STM32_I2C2_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 4))
• #define STM32_I2C2_TX_DMA_CHN 0x00000000
• #define STM32_HAS_I2C3 FALSE
• #define STM32_I2C3_RX_DMA_MSK 0
• #define STM32_I2C3_RX_DMA_CHN 0x00000000
• #define STM32_I2C3_TX_DMA_MSK 0
• #define STM32_I2C3_TX_DMA_CHN 0x00000000
• #define STM32_HAS_RTC TRUE
• #define STM32_RTCSEL_HAS_SUBSECONDS TRUE
• #define STM32_HAS_SDIO FALSE
• #define STM32_HAS_SPI1 TRUE
• #define STM32_SPI1_RX_DMA_MSK STM32_DMA_STREAM_ID_MSK(1, 2)
• #define STM32_SPI1_RX_DMA_CHN 0x00000000
• #define STM32_SPI1_TX_DMA_MSK STM32_DMA_STREAM_ID_MSK(1, 3)
• #define STM32_SPI1_TX_DMA_CHN 0x00000000
• #define STM32_HAS_SPI2 TRUE
• #define STM32_SPI2_RX_DMA_MSK STM32_DMA_STREAM_ID_MSK(1, 4)
• #define STM32_SPI2_RX_DMA_CHN 0x00000000
• #define STM32_SPI2_TX_DMA_MSK STM32_DMA_STREAM_ID_MSK(1, 5)
• #define STM32_SPI2_TX_DMA_CHN 0x00000000
• #define STM32_HAS_SPI3 TRUE
• #define STM32_SPI3_RX_DMA_MSK STM32_DMA_STREAM_ID_MSK(2, 1)
• #define STM32_SPI3_RX_DMA_CHN 0x00000000
• #define STM32_SPI3_TX_DMA_MSK STM32_DMA_STREAM_ID_MSK(2, 2)
• #define STM32_SPI3_TX_DMA_CHN 0x00000000
• #define STM32_HAS_TIM1 TRUE
• #define STM32_HAS_TIM2 TRUE
• #define STM32_HAS_TIM3 TRUE
• #define STM32_HAS_TIM4 TRUE
• #define STM32_HAS_TIM5 TRUE
• #define STM32_HAS_TIM6 TRUE
• #define STM32_HAS_TIM7 TRUE
• #define STM32_HAS_TIM8 FALSE
• #define STM32_HAS_TIM9 FALSE
• #define STM32_HAS_TIM10 FALSE
• #define STM32_HAS_TIM11 FALSE
• #define STM32_HAS_TIM12 FALSE
• #define STM32_HAS_TIM13 FALSE
• #define STM32_HAS_TIM14 FALSE
• #define STM32_HAS_TIM15 FALSE
• #define STM32_HAS_TIM16 FALSE
• #define STM32_HAS_TIM17 FALSE
• #define STM32_HAS_USART1 TRUE
• #define STM32_USART1_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 5))
• #define STM32_USART1_RX_DMA_CHN 0x00000000
• #define STM32_USART1_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 4))
• #define STM32_USART1_TX_DMA_CHN 0x00000000
• #define STM32_HAS_USART2 TRUE
• #define STM32_USART2_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 6))
• #define STM32_USART2_RX_DMA_CHN 0x00000000
• #define STM32_USART2_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 7))
• #define STM32_USART2_TX_DMA_CHN 0x00000000
• #define STM32_HAS_USART3 TRUE
• #define STM32_USART3_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 3))
• #define STM32_USART3_RX_DMA_CHN 0x00000000
• #define STM32_USART3_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 2))
• #define STM32_USART3_TX_DMA_CHN 0x00000000
```

- #define **STM32\_HAS\_UART4** TRUE
- #define **STM32\_UART4\_RX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(2, 3))
- #define **STM32\_UART4\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_UART4\_TX\_DMA\_MSK** (STM32\_DMA\_STREAM\_ID\_MSK(2, 5))
- #define **STM32\_UART4\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_UART5** TRUE
- #define **STM32\_UART5\_RX\_DMA\_MSK** 0
- #define **STM32\_UART5\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_UART5\_TX\_DMA\_MSK** 0
- #define **STM32\_UART5\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_USART6** FALSE
- #define **STM32\_USART6\_RX\_DMA\_MSK** 0
- #define **STM32\_USART6\_RX\_DMA\_CHN** 0x00000000
- #define **STM32\_USART6\_TX\_DMA\_MSK** 0
- #define **STM32\_USART6\_TX\_DMA\_CHN** 0x00000000
- #define **STM32\_HAS\_USB** FALSE
- #define **STM32\_HAS\_OTG1** TRUE
- #define **STM32\_HAS\_OTG2** FALSE

#### IRQ VECTOR names

- #define **WWDG\_IRQHandler** Vector40
- #define **PVD\_IRQHandler** Vector44
- #define **TAMPER\_IRQHandler** Vector48
- #define **RTC\_IRQHandler** Vector4C
- #define **FLASH\_IRQHandler** Vector50
- #define **RCC\_IRQHandler** Vector54
- #define **EXTI0\_IRQHandler** Vector58
- #define **EXTI1\_IRQHandler** Vector5C
- #define **EXTI2\_IRQHandler** Vector60
- #define **EXTI3\_IRQHandler** Vector64
- #define **EXTI4\_IRQHandler** Vector68
- #define **DMA1\_Ch1\_IRQHandler** Vector6C
- #define **DMA1\_Ch2\_IRQHandler** Vector70
- #define **DMA1\_Ch3\_IRQHandler** Vector74
- #define **DMA1\_Ch4\_IRQHandler** Vector78
- #define **DMA1\_Ch5\_IRQHandler** Vector7C
- #define **DMA1\_Ch6\_IRQHandler** Vector80
- #define **DMA1\_Ch7\_IRQHandler** Vector84
- #define **ADC1\_2\_IRQHandler** Vector88
- #define **CAN1\_TX\_IRQHandler** Vector8C
- #define **CAN1\_RX0\_IRQHandler** Vector90
- #define **CAN1\_RX1\_IRQHandler** Vector94
- #define **CAN1\_SCE\_IRQHandler** Vector98
- #define **EXTI9\_5\_IRQHandler** Vector9C
- #define **TIM1\_BRK\_IRQHandler** VectorA0
- #define **TIM1\_UP\_IRQHandler** VectorA4
- #define **TIM1\_TRG\_COM\_IRQHandler** VectorA8
- #define **TIM1\_CC\_IRQHandler** VectorAC
- #define **TIM2\_IRQHandler** VectorB0
- #define **TIM3\_IRQHandler** VectorB4
- #define **TIM4\_IRQHandler** VectorB8
- #define **I2C1\_EV\_IRQHandler** VectorBC
- #define **I2C1\_ER\_IRQHandler** VectorC0
- #define **I2C2\_EV\_IRQHandler** VectorC4
- #define **I2C2\_ER\_IRQHandler** VectorC8
- #define **SPI1\_IRQHandler** VectorCC
- #define **SPI2\_IRQHandler** VectorD0
- #define **USART1\_IRQHandler** VectorD4
- #define **USART2\_IRQHandler** VectorD8
- #define **USART3\_IRQHandler** VectorDC
- #define **EXTI15\_10\_IRQHandler** VectorE0
- #define **RTC\_Alarm\_IRQHandler** VectorE4
- #define **OTG\_FS\_WKUP\_IRQHandler** VectorE8
- #define **TIM5\_IRQHandler** Vector108

- #define SPI3\_IRQHandler Vector10C
- #define UART4\_IRQHandler Vector110
- #define UART5\_IRQHandler Vector114
- #define TIM6\_IRQHandler Vector118
- #define TIM7\_IRQHandler Vector11C
- #define DMA2\_Ch1\_IRQHandler Vector120
- #define DMA2\_Ch2\_IRQHandler Vector124
- #define DMA2\_Ch3\_IRQHandler Vector128
- #define DMA2\_Ch4\_IRQHandler Vector12C
- #define DMA2\_Ch5\_IRQHandler Vector130
- #define ETH\_IRQHandler Vector134
- #define ETH\_WKUP\_IRQHandler Vector138
- #define CAN2\_TX\_IRQHandler Vector13C
- #define CAN2\_RX0\_IRQHandler Vector140
- #define CAN2\_RX1\_IRQHandler Vector144
- #define CAN2\_SCE\_IRQHandler Vector148
- #define OTG\_FS\_IRQHandler Vector14C

#### Configuration options

- #define STM32\_SW STM32\_SW\_PLL  
*Main clock source selection.*
- #define STM32\_PLLSRC STM32\_PLLSRC\_PREDIV1  
*Clock source for the PLL.*
- #define STM32\_PREDIV1SRC STM32\_PREDIV1SRC\_HSE  
*PREDIV1 clock source.*
- #define STM32\_PREDIV1\_VALUE 5  
*PREDIV1 division factor.*
- #define STM32\_PLLMUL\_VALUE 9  
*PLL multiplier value.*
- #define STM32\_PREDIV2\_VALUE 5  
*PREDIV2 division factor.*
- #define STM32\_PLL2MUL\_VALUE 8  
*PLL2 multiplier value.*
- #define STM32\_PLL3MUL\_VALUE 10  
*PLL3 multiplier value.*
- #define STM32\_HPRE STM32\_HPRE\_DIV1  
*AHB prescaler value.*
- #define STM32\_PPREG1 STM32\_PPREG1\_DIV2  
*APB1 prescaler value.*
- #define STM32\_PPREG2 STM32\_PPREG2\_DIV2  
*APB2 prescaler value.*
- #define STM32\_ADCPRE STM32\_ADCPRE\_DIV4  
*ADC prescaler value.*
- #define STM32\_OTG\_CLOCK\_REQUIRED TRUE  
*USB clock setting.*
- #define STM32\_OTGFSPRE STM32\_OTGFSPRE\_DIV3  
*OTG prescaler initialization.*
- #define STM32\_I2S\_CLOCK\_REQUIRED FALSE  
*Dedicated I2S clock setting.*
- #define STM32\_MCOSEL STM32\_MCOSEL\_NOCLOCK  
*MCO pin setting.*
- #define STM32\_RTCSEL STM32\_RTCSEL\_HSEDIV  
*Clock source selecting. LSI by default.*

## 8.23 halconf.h File Reference

### 8.23.1 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. #include "mcuconf.h"

## Defines

### Drivers enable switches

- #define **HAL\_USE\_TM** TRUE  
*Enables the TM subsystem.*
- #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\_EXT** FALSE  
*Enables the EXT subsystem.*
- #define **HAL\_USE\_GPT** FALSE  
*Enables the GPT subsystem.*
- #define **HAL\_USE\_I2C** FALSE  
*Enables the I2C subsystem.*
- #define **HAL\_USE\_ICU** FALSE  
*Enables the ICU subsystem.*
- #define **HAL\_USE\_MAC** TRUE  
*Enables the MAC subsystem.*
- #define **HAL\_USE\_MMC\_SPI** TRUE  
*Enables the MMC\_SPI subsystem.*
- #define **HAL\_USE\_PWM** TRUE  
*Enables the PWM subsystem.*
- #define **HAL\_USE\_RTC** FALSE  
*Enables the RTC subsystem.*
- #define **HAL\_USE\_SDC** FALSE  
*Enables the SDC subsystem.*
- #define **HAL\_USE\_SERIAL** TRUE  
*Enables the SERIAL subsystem.*
- #define **HAL\_USE\_SERIAL\_USB** TRUE  
*Enables the SERIAL over USB subsystem.*
- #define **HAL\_USE\_SPI** TRUE  
*Enables the SPI subsystem.*
- #define **HAL\_USE\_UART** TRUE  
*Enables the UART subsystem.*
- #define **HAL\_USE\_USB** TRUE  
*Enables the USB subsystem.*

### ADC driver related setting

- #define **ADC\_USE\_WAIT** TRUE  
*Enables synchronous APIs.*
- #define **ADC\_USE\_MUTUAL\_EXCLUSION** TRUE  
*Enables the `adcAcquireBus()` and `adcReleaseBus()` APIs.*

### CAN driver related setting

- #define **CAN\_USE\_SLEEP\_MODE** TRUE  
*Sleep mode related APIs inclusion switch.*

### I2C driver related setting

- #define **I2C\_USE\_MUTUAL\_EXCLUSION** TRUE  
*Enables the mutual exclusion APIs on the I2C bus.*

### MAC driver related setting

- #define **MAC\_USE\_EVENTS** TRUE  
*Enables an event sources for incoming packets.*

#### MMC\_SPI driver related setting

- `#define MMC_SECTOR_SIZE 512`  
*Block size for MMC transfers.*
- `#define MMC_NICE_WAITING TRUE`  
*Delays insertions.*
- `#define MMC_POLLING_INTERVAL 10`  
*Number of positive insertion queries before generating the insertion event.*
- `#define MMC_POLLING_DELAY 10`  
*Interval, in milliseconds, between insertion queries.*
- `#define MMC_USE_SPI_POLLING TRUE`  
*Uses the SPI polled API for small data transfers.*

#### SDC driver related setting

- `#define SDC_INIT_RETRY 100`  
*Number of initialization attempts before rejecting the card.*
- `#define SDC_MMIC_SUPPORT FALSE`  
*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 64`  
*Serial over USB buffers size.*

#### SPI driver related setting

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

## 8.24 i2c.c File Reference

### 8.24.1 Detailed Description

```
I2C Driver code. #include "ch.h"
#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 *rdbuf, size_t rxbytes, systime_t timeout)`  
*Sends data via the I2C bus.*
- `msg_t i2cMasterReceiveTimeout (I2CDriver *i2cp, i2caddr_t addr, uint8_t *rdbuf, size_t rxbytes, systime_t timeout)`  
*Receives data from the I2C bus.*
- void `i2cAcquireBus (I2CDriver *i2cp)`  
*Gains exclusive access to the I2C bus.*
- void `i2cReleaseBus (I2CDriver *i2cp)`  
*Releases exclusive access to the I2C bus.*

## 8.25 i2c.h File Reference

### 8.25.1 Detailed Description

I2C Driver macros and structures. #include "i2c\_llld.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 *rdbuf, size_t rxbytes, systime_t timeout)`  
*Sends data via the I2C bus.*
- `msg_t i2cMasterReceiveTimeout (I2CDriver *i2cp, i2caddr_t addr, uint8_t *rdbuf, size_t rxbytes, systime_t timeout)`  
*Receives data from the I2C bus.*

#### Defines

- `#define I2C_USE_MUTUAL_EXCLUSION TRUE`  
*Enables the mutual exclusion APIs on the I2C bus.*
- `#define i2cMasterTransmit(i2cp, addr, txbuf, txbytes, rdbuf, rxbytes)`  
*Wrap `i2cMasterTransmit` function with `TIME_INFINITE` timeout.*
- `#define i2cMasterReceive(i2cp, addr, rdbuf, rxbytes) (i2cMasterReceiveTimeout(i2cp, addr, rdbuf, rxbytes, TIME_INFINITE))`  
*Wrap `i2cMasterReceive` function with `TIME_INFINITE` timeout.*

### I2C bus error conditions

- `#define I2CD_NO_ERROR 0x00`  
*No error.*
- `#define I2CD_BUS_ERROR 0x01`  
*Bus Error.*
- `#define I2CD_ARBITRATION_LOST 0x02`  
*Arbitration Lost (master mode).*
- `#define I2CD_ACK_FAILURE 0x04`  
*Acknowledge Failure.*
- `#define I2CD_OVERRUN 0x08`  
*Overrun/Underrun.*
- `#define I2CD_PEC_ERROR 0x10`  
*PEC Error in reception.*
- `#define I2CD_TIMEOUT 0x20`  
*Hardware timeout.*
- `#define I2CD_SMB_ALERT 0x40`  
*SMBus Alert.*

### Enumerations

- enum `i2cstate_t` {
   
`I2C_UNINIT = 0, I2C_STOP = 1, I2C_READY = 2, I2C_ACTIVE_TX = 3,`
  
`I2C_ACTIVE_RX = 4`
}
   
*Driver state machine possible states.*

## 8.26 i2c\_lld.c File Reference

### 8.26.1 Detailed Description

STM32 I2C subsystem low level driver source.

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

### Functions

- `CH_IRQ_HANDLER (I2C1_EV_IRQHandler)`  
*I2C1 event interrupt handler.*
- `CH_IRQ_HANDLER (I2C1_ER_IRQHandler)`  
*I2C1 error interrupt handler.*
- `CH_IRQ_HANDLER (I2C2_EV_IRQHandler)`  
*I2C2 event interrupt handler.*
- `CH_IRQ_HANDLER (I2C2_ER_IRQHandler)`  
*I2C2 error interrupt handler.*
- `CH_IRQ_HANDLER (I2C3_EV_IRQHandler)`  
*I2C3 event interrupt handler.*
- `CH_IRQ_HANDLER (I2C3_ER_IRQHandler)`  
*I2C3 error interrupt handler.*
- 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)`

- `msg_t i2c_ll_master_receive_timeout (I2CDriver *i2cp, i2caddr_t addr, uint8_t *rxbuff, size_t rxbytes, systime_t timeout)`

*Deactivates the I2C peripheral.*
- `msg_t i2c_ll_master_transmit_timeout (I2CDriver *i2cp, i2caddr_t addr, const uint8_t *txbuff, size_t txbytes, uint8_t *rxbuff, size_t rxbytes, systime_t timeout)`

*Receives data via the I2C bus as master.*
- `msg_t i2c_ll_master_transmit_timeout (I2CDriver *i2cp, i2caddr_t addr, const uint8_t *txbuff, size_t txbytes, uint8_t *rxbuff, size_t rxbytes, systime_t timeout)`

*Transmits data via the I2C bus as master.*

## Variables

- `I2CDriver I2CD1`  
*I2C1 driver identifier.*
- `I2CDriver I2CD2`  
*I2C2 driver identifier.*
- `I2CDriver I2CD3`  
*I2C3 driver identifier.*

## Defines

- `#define wakeup_isr(i2cp, msg)`  
*Wakes up the waiting thread.*

## 8.27 i2c\_ll.h File Reference

### 8.27.1 Detailed Description

STM32 I2C subsystem low level driver header.

## Data Structures

- struct `I2CConfig`  
*Driver configuration structure.*
- struct `I2CDriver`  
*Structure representing an I2C driver.*

## Functions

- `void i2c_ll_init (void)`  
*Low level I2C driver initialization.*
- `void i2c_ll_start (I2CDriver *i2cp)`  
*Configures and activates the I2C peripheral.*
- `void i2c_ll_stop (I2CDriver *i2cp)`  
*Deactivates the I2C peripheral.*
- `msg_t i2c_ll_master_transmit_timeout (I2CDriver *i2cp, i2caddr_t addr, const uint8_t *txbuff, size_t txbytes, uint8_t *rxbuff, size_t rxbytes, systime_t timeout)`  
*Transmits data via the I2C bus as master.*
- `msg_t i2c_ll_master_receive_timeout (I2CDriver *i2cp, i2caddr_t addr, uint8_t *rxbuff, size_t rxbytes, systime_t timeout)`  
*Receives data via the I2C bus as master.*

## Defines

- `#define I2C_CLK_FREQ ((STM32_PCLK1) / 1000000)`  
*Peripheral clock frequency.*
- `#define STM32_DMA_REQUIRED`  
*error checks*
- `#define i2c_ll_get_errors(i2cp) ((i2cp)->errors)`  
*Get errors from I2C driver.*

## Configuration options

- `#define STM32_I2C_USE_I2C1 FALSE`  
*I2C1 driver enable switch.*
- `#define STM32_I2C_USE_I2C2 FALSE`  
*I2C2 driver enable switch.*
- `#define STM32_I2C_USE_I2C3 FALSE`  
*I2C3 driver enable switch.*
- `#define STM32_I2C_I2C1_IRQ_PRIORITY 10`  
*I2C1 interrupt priority level setting.*
- `#define STM32_I2C_I2C2_IRQ_PRIORITY 10`  
*I2C2 interrupt priority level setting.*
- `#define STM32_I2C_I2C3_IRQ_PRIORITY 10`  
*I2C3 interrupt priority level setting.*
- `#define STM32_I2C_I2C1_DMA_PRIORITY 1`  
*I2C1 DMA priority (0..3|lowest..highest).*
- `#define STM32_I2C_I2C2_DMA_PRIORITY 1`  
*I2C2 DMA priority (0..3|lowest..highest).*
- `#define STM32_I2C_I2C3_DMA_PRIORITY 1`  
*I2C3 DMA priority (0..3|lowest..highest).*
- `#define STM32_I2C_DMA_ERROR_HOOK(i2cp) chSysHalt()`  
*I2C DMA error hook.*
- `#define STM32_I2C_I2C1_RX_DMA_STREAM STM32_DMA_STREAM_ID(1, 0)`  
*DMA stream used for I2C1 RX operations.*
- `#define STM32_I2C_I2C1_TX_DMA_STREAM STM32_DMA_STREAM_ID(1, 6)`  
*DMA stream used for I2C1 TX operations.*
- `#define STM32_I2C_I2C2_RX_DMA_STREAM STM32_DMA_STREAM_ID(1, 2)`  
*DMA stream used for I2C2 RX operations.*
- `#define STM32_I2C_I2C2_TX_DMA_STREAM STM32_DMA_STREAM_ID(1, 7)`  
*DMA stream used for I2C2 TX operations.*
- `#define STM32_I2C_I2C3_RX_DMA_STREAM STM32_DMA_STREAM_ID(1, 2)`  
*DMA stream used for I2C3 RX operations.*
- `#define STM32_I2C_I2C3_TX_DMA_STREAM STM32_DMA_STREAM_ID(1, 4)`  
*DMA stream used for I2C3 TX operations.*

## Typedefs

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

## Enumerations

- enum `i2copmode_t`  
*Supported modes for the I2C bus.*
- enum `i2cdutycycle_t`  
*Supported duty cycle modes for the I2C bus.*

## 8.28 icu.c File Reference

### 8.28.1 Detailed Description

ICU Driver code.

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

## Functions

- void `icuInit` (void)  
*ICU Driver initialization.*
- void `icuObjectInit` (`ICUDriver` \*`icup`)  
*Initializes the standard part of a `ICUDriver` structure.*
- void `icuStart` (`ICUDriver` \*`icup`, const `ICUConfig` \*`config`)  
*Configures and activates the ICU peripheral.*
- void `icuStop` (`ICUDriver` \*`icup`)  
*Deactivates the ICU peripheral.*
- void `icuEnable` (`ICUDriver` \*`icup`)  
*Enables the input capture.*
- void `icuDisable` (`ICUDriver` \*`icup`)  
*Disables the input capture.*

## 8.29 icu.h File Reference

### 8.29.1 Detailed Description

ICU Driver macros and structures.

```
#include "icu_lld.h"
```

## Functions

- void `icuInit` (void)  
*ICU Driver initialization.*
- void `icuObjectInit` (`ICUDriver` \*`icup`)  
*Initializes the standard part of a `ICUDriver` structure.*
- void `icuStart` (`ICUDriver` \*`icup`, const `ICUConfig` \*`config`)  
*Configures and activates the ICU peripheral.*
- void `icuStop` (`ICUDriver` \*`icup`)  
*Deactivates the ICU peripheral.*
- void `icuEnable` (`ICUDriver` \*`icup`)  
*Enables the input capture.*
- void `icuDisable` (`ICUDriver` \*`icup`)  
*Disables the input capture.*

## Defines

### Macro Functions

- `#define icuEnableI(icup) icu_lld_enable(icup)`  
*Enables the input capture.*
- `#define icuDisableI(icup) icu_lld_disable(icup)`  
*Disables the input capture.*
- `#define icuGetWidthI(icup) icu_lld_get_width(icup)`  
*Returns the width of the latest pulse.*
- `#define icuGetPeriodI(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.*

## Typedefs

- `typedef struct ICUDriver ICUDriver`  
*Type of a structure representing an ICU driver.*
- `typedef void(* icucallback_t )(ICUDriver *icup)`  
*ICU notification callback type.*

## Enumerations

- `enum icustate_t {`  
`ICU_UNINIT = 0, ICU_STOP = 1, ICU_READY = 2, ICU_WAITING = 3,`  
`ICU_ACTIVE = 4, ICU_IDLE = 5 }`  
*Driver state machine possible states.*

## 8.30 icu\_lld.c File Reference

### 8.30.1 Detailed Description

STM32 ICU subsystem low level driver header.

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

## Functions

- `void icu_lld_init (void)`  
*Low level ICU driver initialization.*
- `void icu_lld_start (ICUDriver *icup)`  
*Configures and activates the ICU peripheral.*
- `void icu_lld_stop (ICUDriver *icup)`  
*Deactivates the ICU peripheral.*
- `void icu_lld_enable (ICUDriver *icup)`  
*Enables the input capture.*
- `void icu_lld_disable (ICUDriver *icup)`  
*Disables the input capture.*

## Variables

- **ICUDriver ICUD1**  
*ICUD1 driver identifier.*
- **ICUDriver ICUD2**  
*ICUD2 driver identifier.*
- **ICUDriver ICUD3**  
*ICUD3 driver identifier.*
- **ICUDriver ICUD4**  
*ICUD4 driver identifier.*
- **ICUDriver ICUD5**  
*ICUD5 driver identifier.*
- **ICUDriver ICUD8**  
*ICUD8 driver identifier.*

## 8.31 icu\_lld.h File Reference

### 8.31.1 Detailed Description

STM32 ICU subsystem low level driver header.

## Data Structures

- struct **ICUConfig**  
*Driver configuration structure.*
- struct **ICUDriver**  
*Structure representing an ICU driver.*

## Functions

- void **icu\_lld\_init** (void)  
*Low level ICU driver initialization.*
- void **icu\_lld\_start** (ICUDriver \*icup)  
*Configures and activates the ICU peripheral.*
- void **icu\_lld\_stop** (ICUDriver \*icup)  
*Deactivates the ICU peripheral.*
- void **icu\_lld\_enable** (ICUDriver \*icup)  
*Enables the input capture.*
- void **icu\_lld\_disable** (ICUDriver \*icup)  
*Disables the input capture.*

## Defines

- #define **icu\_lld\_get\_width**(icup) ((icup)->tim->CCR[1] + 1)  
*Returns the width of the latest pulse.*
- #define **icu\_lld\_get\_period**(icup) ((icup)->tim->CCR[0] + 1)  
*Returns the width of the latest cycle.*

### Configuration options

- `#define STM32_ICU_USE_TIM1 TRUE`  
*ICUD1 driver enable switch.*
- `#define STM32_ICU_USE_TIM2 TRUE`  
*ICUD2 driver enable switch.*
- `#define STM32_ICU_USE_TIM3 TRUE`  
*ICUD3 driver enable switch.*
- `#define STM32_ICU_USE_TIM4 TRUE`  
*ICUD4 driver enable switch.*
- `#define STM32_ICU_USE_TIM5 TRUE`  
*ICUD5 driver enable switch.*
- `#define STM32_ICU_USE_TIM8 TRUE`  
*ICUD8 driver enable switch.*
- `#define STM32_ICU_TIM1_IRQ_PRIORITY 7`  
*ICUD1 interrupt priority level setting.*
- `#define STM32_ICU_TIM2_IRQ_PRIORITY 7`  
*ICUD2 interrupt priority level setting.*
- `#define STM32_ICU_TIM3_IRQ_PRIORITY 7`  
*ICUD3 interrupt priority level setting.*
- `#define STM32_ICU_TIM4_IRQ_PRIORITY 7`  
*ICUD4 interrupt priority level setting.*
- `#define STM32_ICU_TIM5_IRQ_PRIORITY 7`  
*ICUD5 interrupt priority level setting.*
- `#define STM32_ICU_TIM8_IRQ_PRIORITY 7`  
*ICUD8 interrupt priority level setting.*

### Typedefs

- `typedef uint32_t icufreq_t`  
*ICU frequency type.*
- `typedef uint16_t icucnt_t`  
*ICU counter type.*

### Enumerations

- `enum icumode_t { ICU_INPUT_ACTIVE_HIGH = 0, ICU_INPUT_ACTIVE_LOW = 1 }`  
*ICU driver mode.*

## 8.32 mmc\_spi.c File Reference

### 8.32.1 Detailed Description

MMC over SPI driver code.

```
#include <string.h>
#include "ch.h"
#include "hal.h"
```

### Functions

- `void mmcInit (void)`  
*MMC over SPI driver initialization.*
- `void mmcObjectInit (MMCDriver *mmcp, SPIDriver *spip, const SPIConfig *lscfg, const SPIConfig *hscfg, mmcquery_t is_protected, mmcquery_t is_inserted)`

- **void mmcStart (MMCDriver \*mmcp, const MMCCConfig \*config)**  
*Configures and activates the MMC peripheral.*
- **void mmcStop (MMCDriver \*mmcp)**  
*Disables the MMC peripheral.*
- **bool\_t mmcConnect (MMCDriver \*mmcp)**  
*Performs the initialization procedure on the inserted card.*
- **bool\_t mmcDisconnect (MMCDriver \*mmcp)**  
*Brings the driver in a state safe for card removal.*
- **bool\_t mmcStartSequentialRead (MMCDriver \*mmcp, uint32\_t startblk)**  
*Starts a sequential read.*
- **bool\_t mmcSequentialRead (MMCDriver \*mmcp, uint8\_t \*buffer)**  
*Reads a block within a sequential read operation.*
- **bool\_t mmcStopSequentialRead (MMCDriver \*mmcp)**  
*Stops a sequential read gracefully.*
- **bool\_t mmcStartSequentialWrite (MMCDriver \*mmcp, uint32\_t startblk)**  
*Starts a sequential write.*
- **bool\_t mmcSequentialWrite (MMCDriver \*mmcp, const uint8\_t \*buffer)**  
*Writes a block within a sequential write operation.*
- **bool\_t mmcStopSequentialWrite (MMCDriver \*mmcp)**  
*Stops a sequential write gracefully.*

## 8.33 mmc\_spi.h File Reference

### 8.33.1 Detailed Description

MMC over SPI driver header.

#### Data Structures

- **struct MMCCConfig**  
*Driver configuration structure.*
- **struct MMCDriver**  
*Structure representing a MMC driver.*

#### Functions

- **void mmcInit (void)**  
*MMC over SPI driver initialization.*
- **void mmcObjectInit (MMCDriver \*mmcp, SPIDriver \*spip, const SPIConfig \*lscfg, const SPIConfig \*hscfg, mmcquery\_t is\_protected, mmcquery\_t is\_inserted)**  
*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\_t mmcConnect (MMCDriver \*mmcp)**  
*Performs the initialization procedure on the inserted card.*
- **bool\_t mmcDisconnect (MMCDriver \*mmcp)**  
*Brings the driver in a state safe for card removal.*

- `bool_t mmcStartSequentialRead (MMCDriver *mmcp, uint32_t startblk)`  
*Starts a sequential read.*
- `bool_t mmcSequentialRead (MMCDriver *mmcp, uint8_t *buffer)`  
*Reads a block within a sequential read operation.*
- `bool_t mmcStopSequentialRead (MMCDriver *mmcp)`  
*Stops a sequential read gracefully.*
- `bool_t mmcStartSequentialWrite (MMCDriver *mmcp, uint32_t startblk)`  
*Starts a sequential write.*
- `bool_t mmcSequentialWrite (MMCDriver *mmcp, const uint8_t *buffer)`  
*Writes a block within a sequential write operation.*
- `bool_t mmcStopSequentialWrite (MMCDriver *mmcp)`  
*Stops a sequential write gracefully.*

## Defines

### MMC\_SPI configuration options

- `#define MMC_SECTOR_SIZE 512`  
*Block size for MMC transfers.*
- `#define MMC_NICE_WAITING TRUE`  
*Delays insertions.*
- `#define MMC_POLLING_INTERVAL 10`  
*Number of positive insertion queries before generating the insertion event.*
- `#define MMC_POLLING_DELAY 10`  
*Interval, in milliseconds, between insertion queries.*

## Macro Functions

- `#define mmcGetDriverState(mmcp) ((mmcp)->state)`  
*Returns the driver state.*
- `#define mmclsWriteProtected(mmcp) ((mmcp)->is_protected())`  
*Returns the write protect status.*

## Typedefs

- `typedef bool_t(* mmcquery_t )(void)`  
*Function used to query some hardware status bits.*

## Enumerations

- `enum mmcstate_t {`  
`MMC_UNINIT = 0, MMC_STOP = 1, MMC_WAIT = 2, MMC_INSERTED = 3,`  
`MMC_READY = 4, MMC_READING = 5, MMC_WRITING = 6 }`  
*Driver state machine possible states.*

## 8.34 pal.c File Reference

### 8.34.1 Detailed Description

```
I/O Ports Abstraction Layer code. #include "ch.h"
#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.*

## 8.35 pal.h File Reference

### 8.35.1 Detailed Description

I/O Ports Abstraction Layer macros, types and structures. #include "pal\_lld.h"

## Data Structures

- struct `IOBus`  
*I/O bus descriptor.*

## 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.*

## Defines

- `#define PAL_PORT_BIT(n) ((ioportmask_t)(1 << (n)))`  
*Port bit helper macro.*
- `#define PAL_GROUP_MASK(width) ((ioportmask_t)(1 << (width)) - 1)`  
*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 0`  
*After reset state.*
- `#define PAL_MODE_UNCONNECTED 1`  
*Safe state for **unconnected** pads.*
- `#define PAL_MODE_INPUT 2`  
*Regular input high-Z pad.*
- `#define PAL_MODE_INPUT_PULLUP 3`  
*Input pad with weak pull up resistor.*
- `#define PAL_MODE_INPUT_PULLDOWN 4`

- `#define PAL_MODE_INPUT_ANALOG 5`  
*Analog input mode.*
- `#define PAL_MODE_OUTPUT_PUSH_PULL 6`  
*Push-pull output pad.*
- `#define PAL_MODE_OUTPUT_OPENDRAIN 7`  
*Open-drain output pad.*

#### Logic level constants

- `#define PAL_LOW 0`  
*Logical low state.*
- `#define PAL_HIGH 1`  
*Logical high state.*

#### Macro Functions

- `#define palInit(config) pal_lld_init(config)`  
*PAL subsystem initialization.*
- `#define palReadPort(port) ((void)(port), 0)`  
*Reads the physical I/O port states.*
- `#define palReadLatch(port) ((void)(port), 0)`  
*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)) & 1)`  
*Reads an input pad logical state.*
- `#define palWritePad(port, pad, bit)`  
*Writes a logical state on an output pad.*
- `#define palSetPad(port, pad) palSetPort(port, PAL_PORT_BIT(pad))`  
*Sets a pad logical state to PAL\_HIGH.*
- `#define palClearPad(port, pad) palClearPort(port, PAL_PORT_BIT(pad))`  
*Clears a pad logical state to PAL\_LOW.*
- `#define palTogglePad(port, pad) palTogglePort(port, PAL_PORT_BIT(pad))`  
*Toggles a pad logical state.*
- `#define palSetPadMode(port, pad, mode) palSetGroupMode(port, PAL_PORT_BIT(pad), 0, mode)`  
*Pad mode setup.*

## 8.36 pal\_lld.c File Reference

### 8.36.1 Detailed Description

```
STM32F1xx GPIO low level driver code. #include "ch.h"
#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.*

## 8.37 pal\_lld.h File Reference

### 8.37.1 Detailed Description

STM32F1xx GPIO low level driver header.

## Data Structures

- struct `stm32_gpio_setup_t`  
*GPIO port setup info.*
- struct `PALConfig`  
*STM32 GPIO static initializer.*

## 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.*

## Defines

- `#define PAL_IOPORTS_WIDTH 16`  
*Width, in bits, of an I/O port.*
- `#define PAL_WHOLE_PORT ((ioportmask_t)0xFFFF)`  
*Whole port mask.*
- `#define IOPORT1 GPIOA`  
*GPIO port A identifier.*
- `#define IOPORT2 GPIOB`  
*GPIO port B identifier.*
- `#define IOPORT3 GPIOC`  
*GPIO port C identifier.*
- `#define IOPORT4 GPIOD`  
*GPIO port D identifier.*
- `#define IOPORT5 GPIOE`  
*GPIO port E identifier.*
- `#define IOPORT6 GPIOF`  
*GPIO port F identifier.*
- `#define IOPORT7 GPIOG`  
*GPIO port G identifier.*
- `#define pal_lld_init(config) _pal_lld_init(config)`  
*GPIO ports subsystem initialization.*
- `#define pal_lld_readport(port) ((port)->IDR)`

- `#define pal_lld_readlatch(port) ((port)->ODR)`  
*Reads an I/O port.*
- `#define pal_lld_writeport(port, bits) ((port)->ODR = (bits))`  
*Reads the output latch.*
- `#define pal_lld_setport(port, bits) ((port)->BSRR = (bits))`  
*Writes on a I/O port.*
- `#define pal_lld_clearport(port, bits) ((port)->BRR = (bits))`  
*Sets a bits mask on a I/O port.*
- `#define pal_lld_writegroup(port, mask, offset, bits)`  
*Clears a bits mask on a I/O port.*
- `#define pal_lld_setgroupmode(port, mask, offset, mode) _pal_lld_setgroupmode(port, mask << offset, mode)`  
*Writes a group of bits.*
- `#define pal_lld_writepad(port, pad, bit) pal_lld_writegroup(port, 1, pad, bit)`  
*Pads group mode setup.*
- `#define pal_lld_writepad(port, pad, bit) pal_lld_writegroup(port, 1, pad, bit)`  
*Writes a logical state on an output pad.*

#### STM32-specific I/O mode flags

- `#define PAL_MODE_STM32_ALTERNATE_PUSHPULL 16`  
*STM32 specific alternate push-pull output mode.*
- `#define PAL_MODE_STM32_ALTERNATE_OPENDRAIN 17`  
*STM32 specific alternate open-drain output mode.*

### Typedefs

- `typedef uint32_t ioportmask_t`  
*Digital I/O port sized unsigned type.*
- `typedef uint32_t iomode_t`  
*Digital I/O modes.*
- `typedef GPIO_TypeDef * ioportid_t`  
*Port Identifier.*

## 8.38 pwm.c File Reference

### 8.38.1 Detailed Description

```
PWM Driver code. #include "ch.h"
#include "hal.h"
```

### Functions

- `void pwmInit (void)`  
*PWM Driver initialization.*
- `void pwmObjectInit (PWMDriver *pwmp)`  
*Initializes the standard part of a `PWMDriver` structure.*
- `void pwmStart (PWMDriver *pwmp, const PWMConfig *config)`  
*Configures and activates the PWM peripheral.*
- `void pwmStop (PWMDriver *pwmp)`  
*Deactivates the PWM peripheral.*

- void `pwmChangePeriod (PWMDriver *pwmp, pwcnt_t period)`  
*Changes the period the PWM peripheral.*
- void `pwmEnableChannel (PWMDriver *pwmp, pwmchannel_t channel, pwcnt_t width)`  
*Enables a PWM channel.*
- void `pwmDisableChannel (PWMDriver *pwmp, pwmchannel_t channel)`  
*Disables a PWM channel.*

## 8.39 pwm.h File Reference

### 8.39.1 Detailed Description

PWM Driver macros and structures. #include "pwm\_ll.h"

#### Functions

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

#### Defines

##### PWM output mode macros

- #define `PWM_OUTPUT_MASK` 0x0F  
*Standard output modes mask.*
- #define `PWM_OUTPUT_DISABLED` 0x00  
*Output not driven, callback only.*
- #define `PWM_OUTPUT_ACTIVE_HIGH` 0x01  
*Positive PWM logic, active is logic level one.*
- #define `PWM_OUTPUT_ACTIVE_LOW` 0x02  
*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, period)`  
*Changes the period the PWM peripheral.*
- `#define pwmEnableChannelI(pwmp, channel, width) pwm_lld_enable_channel(pwmp, channel, width)`  
*Enables a PWM channel.*
- `#define pwmDisableChannelI(pwmp, channel) pwm_lld_disable_channel(pwmp, channel)`  
*Disables a PWM channel.*

### Typedefs

- `typedef struct PWMDriver PWMDriver`  
*Type of a structure representing a PWM driver.*
- `typedef void(* pwmcallback_t )(PWMDriver *pwmp)`  
*PWM notification callback type.*

### Enumerations

- `enum pwmstate_t { PWM_UNINIT = 0, PWM_STOP = 1, PWM_READY = 2 }`  
*Driver state machine possible states.*

## 8.40 pwm\_lld.c File Reference

### 8.40.1 Detailed Description

STM32 PWM subsystem low level driver header. `#include "ch.h"`  
`#include "hal.h"`

### Functions

- `void pwm_lld_init (void)`  
*Low level PWM driver initialization.*
- `void pwm_lld_start (PWMDriver *pwmp)`  
*Configures and activates the PWM peripheral.*
- `void pwm_lld_stop (PWMDriver *pwmp)`  
*Deactivates the PWM peripheral.*
- `void pwm_lld_enable_channel (PWMDriver *pwmp, pwmchannel_t channel, pwcnt_t width)`  
*Enables a PWM channel.*
- `void pwm_lld_disable_channel (PWMDriver *pwmp, pwmchannel_t channel)`  
*Disables a PWM channel.*

### Variables

- `PWMDriver PWMD1`  
*PWMD1 driver identifier.*
- `PWMDriver PWMD2`  
*PWMD2 driver identifier.*
- `PWMDriver PWMD3`  
*PWMD3 driver identifier.*
- `PWMDriver PWMD4`

- **PWMD4 driver identifier.**
- **PWMDriver PWMD5**
  - PWMD5 driver identifier.*
- **PWMDriver PWMD8**
  - PWMD8 driver identifier.*

## 8.41 pwm\_ll.h File Reference

### 8.41.1 Detailed Description

STM32 PWM subsystem low level driver header.

#### Data Structures

- struct **PWMChannelConfig**
  - PWM driver channel configuration structure.*
- struct **PWMConfig**
  - PWM driver configuration structure.*
- struct **PWMDriver**
  - Structure representing a PWM driver.*

#### Functions

- void **pwm\_ll\_init (void)**
  - Low level PWM driver initialization.*
- void **pwm\_ll\_start (PWMDriver \*pwmp)**
  - Configures and activates the PWM peripheral.*
- void **pwm\_ll\_stop (PWMDriver \*pwmp)**
  - Deactivates the PWM peripheral.*
- void **pwm\_ll\_enable\_channel (PWMDriver \*pwmp, pwmchannel\_t channel, pwmcnt\_t width)**
  - Enables a PWM channel.*
- void **pwm\_ll\_disable\_channel (PWMDriver \*pwmp, pwmchannel\_t channel)**
  - Disables a PWM channel.*

#### Defines

- #define **PWM\_CHANNELS** 4
  - Number of PWM channels per PWM driver.*
- #define **PWM\_COMPLEMENTARY\_OUTPUT\_MASK** 0xF0
  - Complementary output modes mask.*
- #define **PWM\_COMPLEMENTARY\_OUTPUT\_DISABLED** 0x00
  - Complementary output not driven.*
- #define **PWM\_COMPLEMENTARY\_OUTPUT\_ACTIVE\_HIGH** 0x10
  - Complementary output, active is logic level one.*
- #define **PWM\_COMPLEMENTARY\_OUTPUT\_ACTIVE\_LOW** 0x20
  - Complementary output, active is logic level zero.*
- #define **pwm\_ll\_change\_period(pwmp, period)** ((pwmp)->tim->ARR = (uint16\_t)((period) - 1))
  - Changes the period the PWM peripheral.*

### Configuration options

- `#define STM32_PWM_USE_ADVANCED TRUE`  
*If advanced timer features switch.*
- `#define STM32_PWM_USE_TIM1 TRUE`  
*PWMD1 driver enable switch.*
- `#define STM32_PWM_USE_TIM2 TRUE`  
*PWMD2 driver enable switch.*
- `#define STM32_PWM_USE_TIM3 TRUE`  
*PWMD3 driver enable switch.*
- `#define STM32_PWM_USE_TIM4 TRUE`  
*PWMD4 driver enable switch.*
- `#define STM32_PWM_USE_TIM5 TRUE`  
*PWMD5 driver enable switch.*
- `#define STM32_PWM_USE_TIM8 TRUE`  
*PWMD8 driver enable switch.*
- `#define STM32_PWM_TIM1_IRQ_PRIORITY 7`  
*PWMD1 interrupt priority level setting.*
- `#define STM32_PWM_TIM2_IRQ_PRIORITY 7`  
*PWMD2 interrupt priority level setting.*
- `#define STM32_PWM_TIM3_IRQ_PRIORITY 7`  
*PWMD3 interrupt priority level setting.*
- `#define STM32_PWM_TIM4_IRQ_PRIORITY 7`  
*PWMD4 interrupt priority level setting.*
- `#define STM32_PWM_TIM5_IRQ_PRIORITY 7`  
*PWMD5 interrupt priority level setting.*
- `#define STM32_PWM_TIM8_IRQ_PRIORITY 7`  
*PWMD8 interrupt priority level setting.*

### Typedefs

- `typedef uint32_t pwmmode_t`  
*PWM mode type.*
- `typedef uint8_t pwmchannel_t`  
*PWM channel type.*
- `typedef uint16_t pwmcnt_t`  
*PWM counter type.*

## 8.42 rtc.c File Reference

### 8.42.1 Detailed Description

```
RTC Driver code. #include "ch.h"
#include "hal.h"
```

### Functions

- `void rtcInit (void)`  
*RTC Driver initialization.*
- `void rtcSetTime (RTCDriver *rtcp, const RTCTime *timespec)`  
*Set current time.*
- `void rtcGetTime (RTCDriver *rtcp, RTCTime *timespec)`  
*Get current time.*

- void `rtcSetAlarm` (`RTCDriver` \*`rtcp`, `rtcalarm_t` `alarm`, const `RTCAlarm` \*`alarmspec`)  
*Set alarm time.*
- void `rtcGetAlarm` (`RTCDriver` \*`rtcp`, `rtcalarm_t` `alarm`, `RTCAlarm` \*`alarmspec`)  
*Get current alarm.*
- void `rtcSetCallback` (`RTCDriver` \*`rtcp`, `rtccb_t` `callback`)  
*Enables or disables RTC callbacks.*

## 8.43 rtc.h File Reference

### 8.43.1 Detailed Description

RTC Driver macros and structures. #include "rtc\_lld.h"

#### Functions

- void `rtcInit` (void)  
*RTC Driver initialization.*
- void `rtcSetTime` (`RTCDriver` \*`rtcp`, const `RTCTime` \*`timespec`)  
*Set current time.*
- void `rtcGetTime` (`RTCDriver` \*`rtcp`, `RTCTime` \*`timespec`)  
*Get current time.*

#### Defines

- #define `rtcSetTimel`(`rtcp`, `timespec`) `rtc_lld_set_time`(`rtcp`, `timespec`)  
*Set current time.*
- #define `rtcGetTimel`(`rtcp`, `timespec`) `rtc_lld_get_time`(`rtcp`, `timespec`)  
*Get current time.*
- #define `rtcSetAlarms`(`rtcp`, `alarm`, `alarmspec`) `rtc_lld_set_alarm`(`rtcp`, `alarm`, `alarmspec`)  
*Set alarm time.*
- #define `rtcGetAlarms`(`rtcp`, `alarm`, `alarmspec`) `rtc_lld_get_alarm`(`rtcp`, `alarm`, `alarmspec`)  
*Get current alarm.*
- #define `rtcSetCallback`(`rtcp`, `callback`) `rtc_lld_set_callback`(`rtcp`, `callback`)  
*Enables or disables RTC callbacks.*

#### Typedefs

- typedef struct `RTCDriver` `RTCDriver`  
*Type of a structure representing an RTC driver.*
- typedef struct `RTCTime` `RTCTime`  
*Type of a structure representing an RTC time stamp.*

## 8.44 rtc\_lld.c File Reference

### 8.44.1 Detailed Description

STM32 RTC subsystem low level driver header. #include "ch.h"  
  #include "hal.h"

## Functions

- **CH\_IRQ\_HANDLER** (RTC\_IRQHandler)

*RTC interrupt handler.*

- **void rtc\_ll\_init (void)**

*Enable access to registers and initialize RTC if BKP domain was previously reseted.*

- **void rtc\_ll\_set\_time (RTCDriver \*rtcp, const RTCTime \*timespec)**

*Set current time.*

- **void rtc\_ll\_get\_time (RTCDriver \*rtcp, RTCTime \*timespec)**

*Get current time.*

- **void rtc\_ll\_set\_alarm (RTCDriver \*rtcp, rtcalarm\_t alarm, const RTCAlarm \*alarmspec)**

*Set alarm time.*

- **void rtc\_ll\_get\_alarm (RTCDriver \*rtcp, rtcalarm\_t alarm, RTCAlarm \*alarmspec)**

*Get current alarm.*

- **void rtc\_ll\_set\_callback (RTCDriver \*rtcp, rtccb\_t callback)**

*Enables or disables RTC callbacks.*

## Variables

- **RTCDriver RTCD1**

*RTC driver identifier.*

## Defines

- **#define rtc\_ll\_apb1\_sync()** {while (((RTC->CRL & RTC\_CRL\_RSF) == 0);}

*Wait for synchronization of RTC registers with APB1 bus.*

- **#define rtc\_ll\_wait\_write()** {while (((RTC->CRL & RTC\_CRL\_RTOFF) == 0);}

*Wait for previous write operation complete.*

- **#define rtc\_ll\_acquire()** {rtc\_ll\_wait\_write(); RTC->CRL |= RTC\_CRL\_CNF;}

*Acquires write access to RTC registers.*

- **#define rtc\_ll\_release()** {RTC->CRL &= ~RTC\_CRL\_CNF;}

*Releases write access to RTC registers.*

## 8.45 rtc\_ll.h File Reference

### 8.45.1 Detailed Description

STM32F1xx RTC subsystem low level driver header.

## Data Structures

- struct **RTCCallbackConfig**  
*Structure representing an RTC callbacks config.*
- struct **RTCTime**  
*Structure representing an RTC time stamp.*
- struct **RTCAlarm**  
*Structure representing an RTC alarm time stamp.*
- struct **RTCDriver**  
*Structure representing an RTC driver.*

## Functions

- void `rtc_lld_init` (void)  
*Enable access to registers and initialize RTC if BKP domain was previously reseted.*
- void `rtc_lld_set_time` (`RTCDriver` \*`rtcp`, const `RTCTime` \*`timespec`)  
*Set current time.*
- void `rtc_lld_get_time` (`RTCDriver` \*`rtcp`, `RTCTime` \*`timespec`)  
*Get current time.*
- void `rtc_lld_set_alarm` (`RTCDriver` \*`rtcp`, `rtcalarm_t` `alarm`, const `RTCAlarm` \*`alarmspec`)  
*Set alarm time.*
- void `rtc_lld_get_alarm` (`RTCDriver` \*`rtcp`, `rtcalarm_t` `alarm`, `RTCAlarm` \*`alarmspec`)  
*Get current alarm.*
- void `rtc_lld_set_callback` (`RTCDriver` \*`rtcp`, `rtccb_t` `callback`)  
*Enables or disables RTC callbacks.*

## Defines

- `#define RTC_SUPPORTS_CALLBACKS TRUE`  
*This RTC implementation supports callbacks.*
- `#define RTC_ALARMS 1`  
*One alarm comparator available.*

### Configuration options

- `#define STM32_RTC_IRQ_PRIORITY 15`

## Typedefs

- `typedef struct RTCAlarm RTCAlarm`  
*Type of a structure representing an RTC alarm time stamp.*
- `typedef struct RTCCallbackConfig RTCCallbackConfig`  
*Type of a structure representing an RTC callbacks config.*
- `typedef uint32_t rtcalarm_t`  
*Type of an RTC alarm.*
- `typedef void(* rtccb_t)(RTCDriver *rtcp, rtcevent_t event)`  
*Type of a generic RTC callback.*

## Enumerations

- `enum rtcevent_t { RTC_EVENT_ALARM = 1, RTC_EVENT_OVERFLOW = 2 }`  
*Type of an RTC event.*

## 8.46 sdc.c File Reference

### 8.46.1 Detailed Description

```
SDC Driver code. #include "ch.h"
#include "hal.h"
```

## Functions

- **bool\_t \_sdc\_wait\_for\_transfer\_state (SDCDriver \*sdcp)**  
*Wait for the card to complete pending operations.*
- **void sdclinit (void)**  
*SDC Driver initialization.*
- **void sdcObjectInit (SDCDriver \*sdcp)**  
*Initializes the standard part of a `SDCDriver` structure.*
- **void sdcStart (SDCDriver \*sdcp, const SDCCConfig \*config)**  
*Configures and activates the SDC peripheral.*
- **void sdcStop (SDCDriver \*sdcp)**  
*Deactivates the SDC peripheral.*
- **bool\_t sdcConnect (SDCDriver \*sdcp)**  
*Performs the initialization procedure on the inserted card.*
- **bool\_t sdcDisconnect (SDCDriver \*sdcp)**  
*Brings the driver in a state safe for card removal.*
- **bool\_t sdcRead (SDCDriver \*sdcp, uint32\_t startblk, uint8\_t \*buf, uint32\_t n)**  
*Reads one or more blocks.*
- **bool\_t sdcWrite (SDCDriver \*sdcp, uint32\_t startblk, const uint8\_t \*buf, uint32\_t n)**  
*Writes one or more blocks.*

## 8.47 sdc.h File Reference

### 8.47.1 Detailed Description

SDC Driver macros and structures. #include "sdc\_llld.h"

## Functions

- **void sdclinit (void)**  
*SDC Driver initialization.*
- **void sdcObjectInit (SDCDriver \*sdcp)**  
*Initializes the standard part of a `SDCDriver` structure.*
- **void sdcStart (SDCDriver \*sdcp, const SDCCConfig \*config)**  
*Configures and activates the SDC peripheral.*
- **void sdcStop (SDCDriver \*sdcp)**  
*Deactivates the SDC peripheral.*
- **bool\_t sdcConnect (SDCDriver \*sdcp)**  
*Performs the initialization procedure on the inserted card.*
- **bool\_t sdcDisconnect (SDCDriver \*sdcp)**  
*Brings the driver in a state safe for card removal.*
- **bool\_t sdcRead (SDCDriver \*sdcp, uint32\_t startblk, uint8\_t \*buf, uint32\_t n)**  
*Reads one or more blocks.*
- **bool\_t sdcWrite (SDCDriver \*sdcp, uint32\_t startblk, const uint8\_t \*buf, uint32\_t n)**  
*Writes one or more blocks.*
- **bool\_t \_sdc\_wait\_for\_transfer\_state (SDCDriver \*sdcp)**  
*Wait for the card to complete pending operations.*

## Defines

- #define `SDC_BLOCK_SIZE` 512
- #define `SDC_CMD8_PATTERN` 0x000001AA  
*Fixed pattern for CMD8.*
- #define `SDC_R1_ERROR_MASK` 0xFDFFFE008  
*Mask of error bits in R1 responses.*

## SD card types

- #define `SDC_MODE_CARDTYPE_MASK` 0xF  
*Card type mask.*
- #define `SDC_MODE_CARDTYPE_SDV11` 0  
*Card is SD V1.1.*
- #define `SDC_MODE_CARDTYPE_SDV20` 1  
*Card is SD V2.0.*
- #define `SDC_MODE_CARDTYPE_MMC` 2  
*Card is MMC.*
- #define `SDC_MODE_HIGH_CAPACITY` 0x10  
*High cap.card.*

## 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.*

## R1 response utilities

- #define `SDC_R1_ERROR(r1)` (((r1) & `SDC_R1_ERROR_MASK`) != 0)  
*Evaluates to TRUE if the R1 response contains error flags.*
- #define `SDC_R1_STS(r1)` (((r1) >> 9) & 15)  
*Returns the status field of an R1 response.*
- #define `SDC_R1_IS_CARD_LOCKED(r1)` (((r1) >> 21) & 1)  
*Evaluates to TRUE if the R1 response indicates a locked card.*

## Macro Functions

- #define `sdcGetDriverState(sdc)` ((sdc)->state)  
*Returns the driver state.*
- #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

- enum `sdcstate_t` {
   
    `SDC_UNINIT` = 0, `SDC_STOP` = 1, `SDC_READY` = 2, `SDC_CONNECTING` = 3,
   
    `SDC_DISCONNECTING` = 4, `SDC_ACTIVE` = 5, `SDC_READING` = 6, `SDC_WRITING` = 7 }
   
*Driver state machine possible states.*

## 8.48 sdc\_lld.c File Reference

### 8.48.1 Detailed Description

STM32 SDC subsystem low level driver source.

```
#include <string.h>
#include "ch.h"
#include "hal.h"
```

### Functions

- **`CH_IRQ_HANDLER (SDIO_IRQHandler)`**  
*SDIO IRQ handler.*
- **`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)`**  
*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_t 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_t 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_t 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_t sdc_lld_read (SDCDriver *sdcp, uint32_t startblk, uint8_t *buf, uint32_t n)`**  
*Reads one or more blocks.*
- **`bool_t sdc_lld_write (SDCDriver *sdcp, uint32_t startblk, const uint8_t *buf, uint32_t n)`**  
*Writes one or more blocks.*

### Variables

- **`SDCDriver SDCD1`**  
*SDCD1 driver identifier.*

## 8.49 sdc\_lld.h File Reference

### 8.49.1 Detailed Description

STM32 SDC subsystem low level driver header.

## Data Structures

- struct **SDCConfig**  
*Driver configuration structure.*
- struct **SDCDriver**  
*Structure representing an SDC driver.*

## Functions

- void **sdc\_ll\_init** (void)  
*Low level SDC driver initialization.*
- void **sdc\_ll\_start** (**SDCDriver** \*sdcp)  
*Configures and activates the SDC peripheral.*
- void **sdc\_ll\_stop** (**SDCDriver** \*sdcp)  
*Deactivates the SDC peripheral.*
- void **sdc\_ll\_start\_clk** (**SDCDriver** \*sdcp)  
*Starts the SDIO clock and sets it to init mode (400KHz or less).*
- void **sdc\_ll\_set\_data\_clk** (**SDCDriver** \*sdcp)  
*Sets the SDIO clock to data mode (25MHz or less).*
- void **sdc\_ll\_stop\_clk** (**SDCDriver** \*sdcp)  
*Stops the SDIO clock.*
- void **sdc\_ll\_set\_bus\_mode** (**SDCDriver** \*sdcp, **sdcbusmode\_t** mode)  
*Switches the bus to 4 bits mode.*
- void **sdc\_ll\_send\_cmd\_none** (**SDCDriver** \*sdcp, uint8\_t cmd, uint32\_t arg)  
*Sends an SDIO command with no response expected.*
- bool\_t **sdc\_ll\_send\_cmd\_short** (**SDCDriver** \*sdcp, uint8\_t cmd, uint32\_t arg, uint32\_t \*resp)  
*Sends an SDIO command with a short response expected.*
- bool\_t **sdc\_ll\_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\_t **sdc\_ll\_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\_t **sdc\_ll\_read** (**SDCDriver** \*sdcp, uint32\_t startblk, uint8\_t \*buf, uint32\_t n)  
*Reads one or more blocks.*
- bool\_t **sdc\_ll\_write** (**SDCDriver** \*sdcp, uint32\_t startblk, const uint8\_t \*buf, uint32\_t n)  
*Writes one or more blocks.*

## Defines

### Configuration options

- #define **STM32\_SDC\_DATATIMEOUT** 0x000FFFFF  
*SDIO data timeout in SDIO clock cycles.*
- #define **STM32\_SDC\_SDIO\_DMA\_PRIORITY** 3  
*SDIO DMA priority (0..3|lowest..highest).*
- #define **STM32\_SDC\_SDIO\_IRQ\_PRIORITY** 9  
*SDIO interrupt priority level setting.*
- #define **STM32\_SDC\_UNALIGNED\_SUPPORT** TRUE  
*SDIO support for unaligned transfers.*

## Typedefs

- **typedef uint32\_t sdcmode\_t**  
*Type of card flags.*
- **typedef struct SDCDriver SDCDriver**  
*Type of a structure representing an SDC driver.*

## Enumerations

- **enum sdcbusmode\_t**  
*Type of SDIO bus mode.*

## 8.50 serial.c File Reference

### 8.50.1 Detailed Description

Serial Driver code. #include "ch.h"  
#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.*

## 8.51 serial.h File Reference

### 8.51.1 Detailed Description

Serial Driver macros and structures. #include "serial\_lld.h"

## Data Structures

- **struct SerialDriverVMT**  
*SerialDriver virtual methods table.*
- **struct SerialDriver**  
*Full duplex serial driver class.*

## 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.*

## Defines

- #define **\_serial\_driver\_methods** \_base\_asynchronous\_channel\_methods  
*SerialDriver specific methods.*

### Serial status flags

- #define **SD\_PARITY\_ERROR** 32  
*Parity error happened.*
- #define **SD\_FRAMING\_ERROR** 64  
*Framing error happened.*
- #define **SD\_OVERRUN\_ERROR** 128  
*Overflow happened.*
- #define **SD\_NOISE\_ERROR** 256  
*Noise on the line.*
- #define **SD\_BREAK\_DETECTED** 512  
*Break detected.*

### Serial configuration options

- #define **SERIAL\_DEFAULT\_BITRATE** 38400  
*Default bit rate.*
- #define **SERIAL\_BUFFERS\_SIZE** 16  
*Serial buffers size.*

## Macro Functions

- #define **sdPutWouldBlock**(sdp) chOQIsFull(&(sdp)->oqueue)  
*Direct output check on a **SerialDriver**.*
- #define **sdGetWouldBlock**(sdp) chIQIsEmpty(&(sdp)->iqueue)  
*Direct input check on a **SerialDriver**.*
- #define **sdPut**(sdp, b) chOQPut(&(sdp)->oqueue, b)  
*Direct write to a **SerialDriver**.*
- #define **sdPutTimeout**(sdp, b, t) chOQPutTimeout(&(sdp)->oqueue, b, t)  
*Direct write to a **SerialDriver** with timeout specification.*
- #define **sdGet**(sdp) chIQGet(&(sdp)->iqueue)  
*Direct read from a **SerialDriver**.*
- #define **sdGetTimeout**(sdp, t) chIQGetTimeout(&(sdp)->iqueue, t)  
*Direct read from a **SerialDriver** with timeout specification.*
- #define **sdWrite**(sdp, b, n) chOQWriteTimeout(&(sdp)->oqueue, b, n, TIME\_INFINITE)  
*Direct blocking write to a **SerialDriver**.*
- #define **sdWriteTimeout**(sdp, b, n, t) chOQWriteTimeout(&(sdp)->oqueue, b, n, t)

- `#define sdAsynchronousWrite(sdp, b, n) chOQWriteTimeout(&(sdp)->oqueue, b, n, TIME_IMMEDIATE)`  
*Direct blocking write to a [SerialDriver](#) with timeout specification.*
- `#define sdRead(sdp, b, n) chIQReadTimeout(&(sdp)->iqueue, b, n, TIME_INFINITE)`  
*Direct non-blocking write to a [SerialDriver](#).*
- `#define sdReadTimeout(sdp, b, n, t) chIQReadTimeout(&(sdp)->iqueue, b, n, t)`  
*Direct blocking read from a [SerialDriver](#) with timeout specification.*
- `#define sdAsynchronousRead(sdp, b, n) chIQReadTimeout(&(sdp)->iqueue, b, n, TIME_IMMEDIATE)`  
*Direct non-blocking read from a [SerialDriver](#).*

## Typedefs

- `typedef struct SerialDriver SerialDriver`

*Structure representing a serial driver.*

## Enumerations

- `enum sdstate_t { SD_UNINIT = 0, SD_STOP = 1, SD_READY = 2 }`

*Driver state machine possible states.*

## 8.52 serial\_Ild.c File Reference

### 8.52.1 Detailed Description

STM32 low level serial driver code.

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

## Functions

- `CH_IRQ_HANDLER (USART1_IRQHandler)`  
*USART1 interrupt handler.*
- `CH_IRQ_HANDLER (USART2_IRQHandler)`  
*USART2 interrupt handler.*
- `CH_IRQ_HANDLER (USART3_IRQHandler)`  
*USART3 interrupt handler.*
- `CH_IRQ_HANDLER (UART4_IRQHandler)`  
*UART4 interrupt handler.*
- `CH_IRQ_HANDLER (UART5_IRQHandler)`  
*UART5 interrupt handler.*
- `CH_IRQ_HANDLER (USART6_IRQHandler)`  
*USART1 interrupt handler.*
- `void sd_Ild_init (void)`  
*Low level serial driver initialization.*
- `void sd_Ild_start (SerialDriver *sdp, const SerialConfig *config)`  
*Low level serial driver configuration and (re)start.*
- `void sd_Ild_stop (SerialDriver *sdp)`  
*Low level serial driver stop.*

## Variables

- **SerialDriver SD1**  
*USART1 serial driver identifier.*
- **SerialDriver SD2**  
*USART2 serial driver identifier.*
- **SerialDriver SD3**  
*USART3 serial driver identifier.*
- **SerialDriver SD4**  
*USART4 serial driver identifier.*
- **SerialDriver SD5**  
*USART5 serial driver identifier.*
- **SerialDriver SD6**  
*USART6 serial driver identifier.*

## 8.53 serial\_lld.h File Reference

### 8.53.1 Detailed Description

STM32 low level serial driver header.

## Data Structures

- struct **SerialConfig**  
*STM32 Serial Driver configuration structure.*

## Functions

- void **sd\_lld\_init** (void)  
*Low level serial driver initialization.*
- void **sd\_lld\_start** (**SerialDriver** \*sdp, const **SerialConfig** \*config)  
*Low level serial driver configuration and (re)start.*
- void **sd\_lld\_stop** (**SerialDriver** \*sdp)  
*Low level serial driver stop.*

## Defines

- #define **\_serial\_driver\_data**  
*SerialDriver specific data.*
- #define **USART\_CR2\_STOP1\_BITS** (0 << 12)  
*CR2 1 stop bit value.*
- #define **USART\_CR2\_STOP0P5\_BITS** (1 << 12)  
*CR2 0.5 stop bit value.*
- #define **USART\_CR2\_STOP2\_BITS** (2 << 12)  
*CR2 2 stop bit value.*
- #define **USART\_CR2\_STOP1P5\_BITS** (3 << 12)  
*CR2 1.5 stop bit value.*

### Configuration options

- `#define STM32_SERIAL_USE_USART1 TRUE`  
*USART1 driver enable switch.*
- `#define STM32_SERIAL_USE_USART2 TRUE`  
*USART2 driver enable switch.*
- `#define STM32_SERIAL_USE_USART3 TRUE`  
*USART3 driver enable switch.*
- `#define STM32_SERIAL_USE_UART4 TRUE`  
*UART4 driver enable switch.*
- `#define STM32_SERIAL_USE_UART5 TRUE`  
*UART5 driver enable switch.*
- `#define STM32_SERIAL_USE_USART6 TRUE`  
*USART6 driver enable switch.*
- `#define STM32_SERIAL_USART1_PRIORITY 12`  
*USART1 interrupt priority level setting.*
- `#define STM32_SERIAL_USART2_PRIORITY 12`  
*USART2 interrupt priority level setting.*
- `#define STM32_SERIAL_USART3_PRIORITY 12`  
*USART3 interrupt priority level setting.*
- `#define STM32_SERIAL_UART4_PRIORITY 12`  
*UART4 interrupt priority level setting.*
- `#define STM32_SERIAL_UART5_PRIORITY 12`  
*UART5 interrupt priority level setting.*
- `#define STM32_SERIAL_USART6_PRIORITY 12`  
*USART6 interrupt priority level setting.*

## 8.54 serial\_usb.c File Reference

### 8.54.1 Detailed Description

```
Serial over USB Driver code. #include "ch.h"
#include "hal.h"
#include "usb_cdc.h"
```

### 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.*
- `bool_t sduRequestsHook (USBDriver *usbp)`  
*Default requests hook.*
- `void sduDataTransmitted (USBDriver *usbp, usbep_t ep)`  
*Default data transmitted callback.*
- `void sduDataReceived (USBDriver *usbp, usbep_t ep)`  
*Default data received callback.*
- `void sduInterruptTransmitted (USBDriver *usbp, usbep_t ep)`  
*Default data received callback.*

## 8.55 serial\_usb.h File Reference

### 8.55.1 Detailed Description

Serial over USB Driver macros and structures.

#### Data Structures

- struct [SerialUSBConfig](#)  
*Serial over USB Driver configuration structure.*
- struct [SerialUSBDriverVMT](#)  
*SerialDriver virtual methods table.*
- struct [SerialUSBDriver](#)  
*Full duplex serial driver class.*

#### Functions

- void [sdulInit](#) (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.*
- bool\_t [sduRequestsHook](#) ([USBDriver](#) \*usbp)  
*Default requests hook.*
- void [sduDataTransmitted](#) ([USBDriver](#) \*usbp, [usbep\\_t](#) ep)  
*Default data transmitted callback.*
- void [sduDataReceived](#) ([USBDriver](#) \*usbp, [usbep\\_t](#) ep)  
*Default data received callback.*
- void [sdulInterruptTransmitted](#) ([USBDriver](#) \*usbp, [usbep\\_t](#) ep)  
*Default data received callback.*

#### Defines

- #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](#) 64  
*Serial over USB buffers size.*

#### Typedefs

- typedef struct [SerialUSBDriver](#) [SerialUSBDriver](#)  
*Structure representing a serial over USB driver.*

## Enumerations

- enum **sdustate\_t** { **SDU\_UNINIT** = 0, **SDU\_STOP** = 1, **SDU\_READY** = 2 }
- Driver state machine possible states.*

## 8.56 spi.c File Reference

### 8.56.1 Detailed Description

```
SPI Driver code. #include "ch.h"
#include "hal.h"
```

## Functions

- void **spilinit** (void)
 

*SPI Driver initialization.*
- void **spiObjectInit** (**SPIDriver** \*spip)
 

*Initializes the standard part of a **SPIDriver** structure.*
- void **spiStart** (**SPIDriver** \*spip, const **SPIConfig** \*config)
 

*Configures and activates the SPI peripheral.*
- void **spiStop** (**SPIDriver** \*spip)
 

*Deactivates the SPI peripheral.*
- void **spiSelect** (**SPIDriver** \*spip)
 

*Asserts the slave select signal and prepares for transfers.*
- void **spiUnselect** (**SPIDriver** \*spip)
 

*Deasserts the slave select signal.*
- void **spiStartIgnore** (**SPIDriver** \*spip, size\_t n)
 

*Ignores data on the SPI bus.*
- void **spiStartExchange** (**SPIDriver** \*spip, size\_t n, const void \*txbuf, void \*rxbuf)
 

*Exchanges data on the SPI bus.*
- void **spiStartSend** (**SPIDriver** \*spip, size\_t n, const void \*txbuf)
 

*Sends data over the SPI bus.*
- void **spiStartReceive** (**SPIDriver** \*spip, size\_t n, void \*rxbuf)
 

*Receives data from the SPI bus.*
- void **spignore** (**SPIDriver** \*spip, size\_t n)
 

*Ignores data on the SPI bus.*
- void **spiExchange** (**SPIDriver** \*spip, size\_t n, const void \*txbuf, void \*rxbuf)
 

*Exchanges data on the SPI bus.*
- void **spiSend** (**SPIDriver** \*spip, size\_t n, const void \*txbuf)
 

*Sends data over the SPI bus.*
- void **spiReceive** (**SPIDriver** \*spip, size\_t n, void \*rxbuf)
 

*Receives data from the SPI bus.*
- void **spiAcquireBus** (**SPIDriver** \*spip)
 

*Gains exclusive access to the SPI bus.*
- void **spiReleaseBus** (**SPIDriver** \*spip)
 

*Releases exclusive access to the SPI bus.*

## 8.57 spi.h File Reference

### 8.57.1 Detailed Description

SPI Driver macros and structures. #include "spi\_lld.h"

#### Functions

- void **spiInit** (void)
 

*SPI Driver initialization.*
- void **spiObjectInit** (SPIDriver \*spip)
 

*Initializes the standard part of a SPIDriver structure.*
- void **spiStart** (SPIDriver \*spip, const SPICConfig \*config)
 

*Configures and activates the SPI peripheral.*
- void **spiStop** (SPIDriver \*spip)
 

*Deactivates the SPI peripheral.*
- void **spiSelect** (SPIDriver \*spip)
 

*Asserts the slave select signal and prepares for transfers.*
- void **spiUnselect** (SPIDriver \*spip)
 

*Deasserts the slave select signal.*
- void **spiStartIgnore** (SPIDriver \*spip, size\_t n)
 

*Ignores data on the SPI bus.*
- void **spiStartExchange** (SPIDriver \*spip, size\_t n, const void \*txbuf, void \*rxbuf)
 

*Exchanges data on the SPI bus.*
- void **spiStartSend** (SPIDriver \*spip, size\_t n, const void \*txbuf)
 

*Sends data over the SPI bus.*
- void **spiStartReceive** (SPIDriver \*spip, size\_t n, void \*rxbuf)
 

*Receives data from the SPI bus.*

#### Defines

##### SPI configuration options

- #define **SPI\_USE\_WAIT** TRUE
 

*Enables synchronous APIs.*
- #define **SPI\_USE\_MUTUAL\_EXCLUSION** TRUE
 

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

##### Macro Functions

- #define **spiSelectl**(spip)
 

*Asserts the slave select signal and prepares for transfers.*
- #define **spiUnselectl**(spip)
 

*Deasserts the slave select signal.*
- #define **spiStartIgnorel**(spip, n)
 

*Ignores data on the SPI bus.*
- #define **spiStartExchangel**(spip, n, txbuf, rxbuf)
 

*Exchanges data on the SPI bus.*
- #define **spiStartSendl**(spip, n, txbuf)
 

*Sends data over the SPI bus.*
- #define **spiStartReceivel**(spip, n, rxbuf)
 

*Receives data from the SPI bus.*
- #define **spiPolledExchange**(spip, frame) spi\_lld\_polled\_exchange(spip, frame)
 

*Exchanges one frame using a polled wait.*

### Low Level driver helper macros

- `#define _spi_wait_s(spir)`  
*Waits for operation completion.*
- `#define _spi_wakeup_isr(spir)`  
*Wakes up the waiting thread.*
- `#define _spi_isr_code(spir)`  
*Common ISR code.*

### Enumerations

- enum `spistate_t` {
   
`SPI_UNINIT = 0, SPI_STOP = 1, SPI_READY = 2, SPI_ACTIVE = 3,`
  
`SPI_COMPLETE = 4 }`
  
*Driver state machine possible states.*

## 8.58 spi\_lld.c File Reference

### 8.58.1 Detailed Description

STM32 SPI subsystem low level driver source. #include "ch.h"  
 #include "hal.h"

### Functions

- void `spi_lld_init (void)`  
*Low level SPI driver initialization.*
- void `spi_lld_start (SPIDriver *spir)`  
*Configures and activates the SPI peripheral.*
- void `spi_lld_stop (SPIDriver *spir)`  
*Deactivates the SPI peripheral.*
- void `spi_lld_select (SPIDriver *spir)`  
*Asserts the slave select signal and prepares for transfers.*
- void `spi_lld_unselect (SPIDriver *spir)`  
*Deasserts the slave select signal.*
- void `spi_lld_ignore (SPIDriver *spir, size_t n)`  
*Ignores data on the SPI bus.*
- void `spi_lld_exchange (SPIDriver *spir, size_t n, const void *txbuf, void *rxbuf)`  
*Exchanges data on the SPI bus.*
- void `spi_lld_send (SPIDriver *spir, size_t n, const void *txbuf)`  
*Sends data over the SPI bus.*
- void `spi_lld_receive (SPIDriver *spir, size_t n, void *rxbuf)`  
*Receives data from the SPI bus.*
- uint16\_t `spi_lld_polled_exchange (SPIDriver *spir, uint16_t frame)`  
*Exchanges one frame using a polled wait.*

## Variables

- **SPIDriver SPID1**  
*SPI1 driver identifier.*
- **SPIDriver SPID2**  
*SPI2 driver identifier.*
- **SPIDriver SPID3**  
*SPI3 driver identifier.*

## 8.59 spi\_lld.h File Reference

### 8.59.1 Detailed Description

STM32 SPI subsystem low level driver header.

## Data Structures

- struct **SPIConfig**  
*Driver configuration structure.*
- struct **SPIDriver**  
*Structure representing a SPI driver.*

## Functions

- void **spi\_lld\_init** (void)  
*Low level SPI driver initialization.*
- void **spi\_lld\_start** (SPIDriver \*spip)  
*Configures and activates the SPI peripheral.*
- void **spi\_lld\_stop** (SPIDriver \*spip)  
*Deactivates the SPI peripheral.*
- void **spi\_lld\_select** (SPIDriver \*spip)  
*Asserts the slave select signal and prepares for transfers.*
- void **spi\_lld\_unselect** (SPIDriver \*spip)  
*Deasserts the slave select signal.*
- void **spi\_lld\_ignore** (SPIDriver \*spip, size\_t n)  
*Ignores data on the SPI bus.*
- void **spi\_lld\_exchange** (SPIDriver \*spip, size\_t n, const void \*txbuf, void \*rxbuf)  
*Exchanges data on the SPI bus.*
- void **spi\_lld\_send** (SPIDriver \*spip, size\_t n, const void \*txbuf)  
*Sends data over the SPI bus.*
- void **spi\_lld\_receive** (SPIDriver \*spip, size\_t n, void \*rxbuf)  
*Receives data from the SPI bus.*
- uint16\_t **spi\_lld\_polled\_exchange** (SPIDriver \*spip, uint16\_t frame)  
*Exchanges one frame using a polled wait.*

## Defines

### Configuration options

- `#define STM32_SPI_USE_SPI1 TRUE`  
*SPI1 driver enable switch.*
- `#define STM32_SPI_USE_SPI2 TRUE`  
*SPI2 driver enable switch.*
- `#define STM32_SPI_USE_SPI3 FALSE`  
*SPI3 driver enable switch.*
- `#define STM32_SPI_SPI1_IRQ_PRIORITY 10`  
*SPI1 interrupt priority level setting.*
- `#define STM32_SPI_SPI2_IRQ_PRIORITY 10`  
*SPI2 interrupt priority level setting.*
- `#define STM32_SPI_SPI3_IRQ_PRIORITY 10`  
*SPI3 interrupt priority level setting.*
- `#define STM32_SPI_SPI1_DMA_PRIORITY 1`  
*SPI1 DMA priority (0..3|lowest..highest).*
- `#define STM32_SPI_SPI2_DMA_PRIORITY 1`  
*SPI2 DMA priority (0..3|lowest..highest).*
- `#define STM32_SPI_SPI3_DMA_PRIORITY 1`  
*SPI3 DMA priority (0..3|lowest..highest).*
- `#define STM32_SPI_DMA_ERROR_HOOK(spip) chSysHalt()`  
*SPI DMA error hook.*
- `#define STM32_SPI_SPI1_RX_DMA_STREAM STM32_DMA_STREAM_ID(2, 0)`  
*DMA stream used for SPI1 RX operations.*
- `#define STM32_SPI_SPI1_TX_DMA_STREAM STM32_DMA_STREAM_ID(2, 3)`  
*DMA stream used for SPI1 TX operations.*
- `#define STM32_SPI_SPI2_RX_DMA_STREAM STM32_DMA_STREAM_ID(1, 3)`  
*DMA stream used for SPI2 RX operations.*
- `#define STM32_SPI_SPI2_TX_DMA_STREAM STM32_DMA_STREAM_ID(1, 4)`  
*DMA stream used for SPI2 TX operations.*
- `#define STM32_SPI_SPI3_RX_DMA_STREAM STM32_DMA_STREAM_ID(1, 0)`  
*DMA stream used for SPI3 RX operations.*
- `#define STM32_SPI_SPI3_TX_DMA_STREAM STM32_DMA_STREAM_ID(1, 7)`  
*DMA stream used for SPI3 TX operations.*

## Typedefs

- `typedef struct SPIDriver SPIDriver`  
*Type of a structure representing an SPI driver.*
- `typedef void(* spicallback_t )(SPIDriver *spip)`  
*SPI notification callback type.*

## 8.60 stm32.h File Reference

### 8.60.1 Detailed Description

STM32 common header.

#### Precondition

One of the following macros must be defined before including this header, the macro selects the inclusion of the appropriate vendor header:

- STM32F10X\_LD\_VL for Value Line Low Density devices.
- STM32F10X\_MD\_VL for Value Line Medium Density devices.

- STM32F10X\_LD for Performance Low Density devices.
- STM32F10X\_MD for Performance Medium Density devices.
- STM32F10X\_HD for Performance High Density devices.
- STM32F10X\_XL for Performance eXtra Density devices.
- STM32F10X\_CL for Connectivity Line devices.
- STM32F2XX for High-performance STM32 F-2 devices.
- STM32F4XX for High-performance STM32 F-4 devices.
- STM32L1XX\_MD for Ultra Low Power Medium-density devices.

```
#include "stm32f10x.h"
#include "stm32f2xx.h"
#include "stm32f4xx.h"
#include "stm32l1xx.h"
```

## Data Structures

- struct **stm32\_tim\_t**  
*STM32 TIM registers block.*

## Defines

### TIM units references

- #define **STM32\_TIM1** ((**stm32\_tim\_t** \*)TIM1\_BASE)
- #define **STM32\_TIM2** ((**stm32\_tim\_t** \*)TIM2\_BASE)
- #define **STM32\_TIM3** ((**stm32\_tim\_t** \*)TIM3\_BASE)
- #define **STM32\_TIM4** ((**stm32\_tim\_t** \*)TIM4\_BASE)
- #define **STM32\_TIM5** ((**stm32\_tim\_t** \*)TIM5\_BASE)
- #define **STM32\_TIM6** ((**stm32\_tim\_t** \*)TIM6\_BASE)
- #define **STM32\_TIM7** ((**stm32\_tim\_t** \*)TIM7\_BASE)
- #define **STM32\_TIM8** ((**stm32\_tim\_t** \*)TIM8\_BASE)
- #define **STM32\_TIM9** ((**stm32\_tim\_t** \*)TIM9\_BASE)
- #define **STM32\_TIM10** ((**stm32\_tim\_t** \*)TIM10\_BASE)
- #define **STM32\_TIM11** ((**stm32\_tim\_t** \*)TIM11\_BASE)
- #define **STM32\_TIM12** ((**stm32\_tim\_t** \*)TIM12\_BASE)
- #define **STM32\_TIM13** ((**stm32\_tim\_t** \*)TIM13\_BASE)
- #define **STM32\_TIM14** ((**stm32\_tim\_t** \*)TIM14\_BASE)

## 8.61 stm32\_dma.c File Reference

### 8.61.1 Detailed Description

DMA helper driver code.

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

## Functions

- **CH\_IRQ\_HANDLER** (DMA1\_Ch1\_IRQHandler)  
*DMA1 stream 1 shared interrupt handler.*
- **CH\_IRQ\_HANDLER** (DMA1\_Ch2\_IRQHandler)  
*DMA1 stream 2 shared interrupt handler.*

- **CH\_IRQ\_HANDLER** (DMA1\_Ch3\_IRQHandler)  
*DMA1 stream 3 shared interrupt handler.*
- **CH\_IRQ\_HANDLER** (DMA1\_Ch4\_IRQHandler)  
*DMA1 stream 4 shared interrupt handler.*
- **CH\_IRQ\_HANDLER** (DMA1\_Ch5\_IRQHandler)  
*DMA1 stream 5 shared interrupt handler.*
- **CH\_IRQ\_HANDLER** (DMA1\_Ch6\_IRQHandler)  
*DMA1 stream 6 shared interrupt handler.*
- **CH\_IRQ\_HANDLER** (DMA1\_Ch7\_IRQHandler)  
*DMA1 stream 7 shared interrupt handler.*
- **CH\_IRQ\_HANDLER** (DMA2\_Ch1\_IRQHandler)  
*DMA2 stream 1 shared interrupt handler.*
- **CH\_IRQ\_HANDLER** (DMA2\_Ch2\_IRQHandler)  
*DMA2 stream 2 shared interrupt handler.*
- **CH\_IRQ\_HANDLER** (DMA2\_Ch3\_IRQHandler)  
*DMA2 stream 3 shared interrupt handler.*
- **CH\_IRQ\_HANDLER** (DMA2\_Ch4\_IRQHandler)  
*DMA2 stream 4 shared interrupt handler.*
- **CH\_IRQ\_HANDLER** (DMA2\_Ch5\_IRQHandler)  
*DMA2 stream 5 shared interrupt handler.*
- void **dmalinit** (void)  
*STM32 DMA helper initialization.*
- bool\_t **dmaStreamAllocate** (const **stm32\_dma\_stream\_t** \*dmastp, uint32\_t priority, **stm32\_dmaisr\_t** func, void \*param)  
*Allocates a DMA stream.*
- void **dmaStreamRelease** (const **stm32\_dma\_stream\_t** \*dmastp)  
*Releases a DMA stream.*

## Variables

- const **stm32\_dma\_stream\_t \_stm32\_dma\_streams** [STM32\_DMA\_STREAMS]  
*DMA streams descriptors.*

## Defines

- #define **STM32\_DMA1\_STREAMS\_MASK** 0x0000007F  
*Mask of the DMA1 streams in dma\_streams\_mask.*
- #define **STM32\_DMA2\_STREAMS\_MASK** 0x00000F80  
*Mask of the DMA2 streams in dma\_streams\_mask.*
- #define **STM32\_DMA\_CCR\_RESET\_VALUE** 0x00000000  
*Post-reset value of the stream CCR register.*

## 8.62 stm32\_dma.h File Reference

### 8.62.1 Detailed Description

DMA helper driver header.

#### Note

This file requires definitions from the ST header file stm32f10x.h.

This driver uses the new naming convention used for the STM32F2xx so the "DMA channels" are referred as "DMA streams".

## Data Structures

- struct `stm32_dma_stream_t`  
*STM32 DMA stream descriptor structure.*

## Functions

- void `dmalinit` (void)  
*STM32 DMA helper initialization.*
- bool\_t `dmaStreamAllocate` (const `stm32_dma_stream_t` \*dmastp, uint32\_t priority, `stm32_dmaisr_t` func, void \*param)  
*Allocates a DMA stream.*
- void `dmaStreamRelease` (const `stm32_dma_stream_t` \*dmastp)  
*Releases a DMA stream.*

## Defines

- #define `STM32_DMA_STREAMS` 12  
*Total number of DMA streams.*
- #define `STM32_DMA_ISR_MASK` 0x0F  
*Mask of the ISR bits passed to the DMA callback functions.*
- #define `STM32_DMA_GETCHANNEL`(n, c) 0  
*Returns the channel associated to the specified stream.*
- #define `STM32_DMA_STREAM_ID_MSK`(dma, stream) ( $1 \ll\ll \text{STM32\_STREAM\_ID}(\text{dma}, \text{stream})$ )  
*Returns a DMA stream identifier mask.*
- #define `STM32_DMA_IS_VALID_ID`(id, mask) (((1  $\ll\ll$  (id)) & (mask)))  
*Checks if a DMA stream unique identifier belongs to a mask.*

### DMA streams identifiers

- #define `STM32_DMA_STREAM_ID`(dma, stream) (((dma) - 1) \* 7) + ((stream) - 1))  
*Returns an unique numeric identifier for a DMA stream.*
- #define `STM32_DMA_STREAM`(id) (&`_stm32_dma_streams`[id])  
*Returns a pointer to a `stm32_dma_stream_t` structure.*
- #define `STM32_DMA1_STREAM1` `STM32_DMA_STREAM(0)`
- #define `STM32_DMA1_STREAM2` `STM32_DMA_STREAM(1)`
- #define `STM32_DMA1_STREAM3` `STM32_DMA_STREAM(2)`
- #define `STM32_DMA1_STREAM4` `STM32_DMA_STREAM(3)`
- #define `STM32_DMA1_STREAM5` `STM32_DMA_STREAM(4)`
- #define `STM32_DMA1_STREAM6` `STM32_DMA_STREAM(5)`
- #define `STM32_DMA1_STREAM7` `STM32_DMA_STREAM(6)`
- #define `STM32_DMA2_STREAM1` `STM32_DMA_STREAM(7)`
- #define `STM32_DMA2_STREAM2` `STM32_DMA_STREAM(8)`
- #define `STM32_DMA2_STREAM3` `STM32_DMA_STREAM(9)`
- #define `STM32_DMA2_STREAM4` `STM32_DMA_STREAM(10)`
- #define `STM32_DMA2_STREAM5` `STM32_DMA_STREAM(11)`

### CR register constants common to all DMA types

- #define `STM32_DMA_CR_EN` `DMA_CCR1_EN`
- #define `STM32_DMA_CR_TEIE` `DMA_CCR1_TEIE`
- #define `STM32_DMA_CR_HTIE` `DMA_CCR1_HTIE`
- #define `STM32_DMA_CR_TCIE` `DMA_CCR1_TCIE`
- #define `STM32_DMA_CR_DIR_MASK` (`DMA_CCR1_DIR` | `DMA_CCR1_MEM2MEM`)
- #define `STM32_DMA_CR_DIR_P2M` 0
- #define `STM32_DMA_CR_DIR_M2P` `DMA_CCR1_DIR`
- #define `STM32_DMA_CR_DIR_M2M` `DMA_CCR1_MEM2MEM`

- #define **STM32\_DMA\_CR\_CIRC** DMA\_CCR1\_CIRC
- #define **STM32\_DMA\_CR\_PINC** DMA\_CCR1\_PINC
- #define **STM32\_DMA\_CR\_MINC** DMA\_CCR1\_MINC
- #define **STM32\_DMA\_CR\_PSIZE\_MASK** DMA\_CCR1\_PSIZE
- #define **STM32\_DMA\_CR\_PSIZE\_BYTEn** BYTE 0
- #define **STM32\_DMA\_CR\_PSIZE\_HWWORD** DMA\_CCR1\_PSIZE\_0
- #define **STM32\_DMA\_CR\_PSIZE\_WORD** DMA\_CCR1\_PSIZE\_1
- #define **STM32\_DMA\_CR\_MSIZEMASK** DMA\_CCR1\_MSIZE
- #define **STM32\_DMA\_CR\_MSIZE\_BYTEn** BYTE 0
- #define **STM32\_DMA\_CR\_MSIZEMASK\_HWWORD** DMA\_CCR1\_MSIZE\_0
- #define **STM32\_DMA\_CR\_MSIZEMASK\_WORD** DMA\_CCR1\_MSIZE\_1
- #define **STM32\_DMA\_CR\_SIZE\_MASK**
- #define **STM32\_DMA\_CR\_PLMASK** DMA\_CCR1\_PL
- #define **STM32\_DMA\_CR\_PL(n)** ((n) << 12)

CR register constants only found in enhanced DMA

- #define **STM32\_DMA\_CR\_DMEIE** 0  
*Ignored by normal DMA.*
- #define **STM32\_DMA\_CR\_CHSEL\_MASK** 0  
*Ignored by normal DMA.*
- #define **STM32\_DMA\_CR\_CHSEL(n)** 0  
*Ignored by normal DMA.*

Status flags passed to the ISR callbacks

- #define **STM32\_DMA\_ISR\_FEIF** 0
- #define **STM32\_DMA\_ISR\_DMEIF** 0
- #define **STM32\_DMA\_ISR\_TEIF** DMA\_ISR\_TEIF1
- #define **STM32\_DMA\_ISR\_HTIF** DMA\_ISR\_HTIF1
- #define **STM32\_DMA\_ISR\_TCIF** DMA\_ISR\_TCIF1

### Macro Functions

- #define **dmaStreamSetPeripheral**(dmastp, addr)  
*Associates a peripheral data register to a DMA stream.*
- #define **dmaStreamSetMemory0**(dmastp, addr)  
*Associates a memory destination to a DMA stream.*
- #define **dmaStreamSetTransactionSize**(dmastp, size)  
*Sets the number of transfers to be performed.*
- #define **dmaStreamGetTransactionSize**(dmastp) ((size\_t)((dmastp)->channel->CNDTR))  
*Returns the number of transfers to be performed.*
- #define **dmaStreamSetMode**(dmastp, mode)  
*Programs the stream mode settings.*
- #define **dmaStreamEnable**(dmastp)  
*DMA stream enable.*
- #define **dmaStreamDisable**(dmastp)  
*DMA stream disable.*
- #define **dmaStreamClearInterrupt**(dmastp)  
*DMA stream interrupt sources clear.*
- #define **dmaStartMemCopy**(dmastp, mode, src, dst, n)  
*Starts a memory to memory operation using the specified stream.*
- #define **dmaWaitCompletion**(dmastp)  
*Polled wait for DMA transfer end.*

### Typedefs

- typedef void(\* **stm32\_dmaisr\_t** )(void \*p, uint32\_t flags)  
*STM32 DMA ISR function type.*

## 8.63 stm32\_rcc.h File Reference

### 8.63.1 Detailed Description

RCC helper driver header.

#### Note

This file requires definitions from the ST header file `stm32f10x.h`.

#### Defines

##### Generic RCC operations

- `#define rccEnableAPB1(mask, lp)`  
*Enables the clock of one or more peripheral on the APB1 bus.*
- `#define rccDisableAPB1(mask, lp)`  
*Disables the clock of one or more peripheral on the APB1 bus.*
- `#define rccResetAPB1(mask)`  
*Resets one or more peripheral on the APB1 bus.*
- `#define rccEnableAPB2(mask, lp)`  
*Enables the clock of one or more peripheral on the APB2 bus.*
- `#define rccDisableAPB2(mask, lp)`  
*Disables the clock of one or more peripheral on the APB2 bus.*
- `#define rccResetAPB2(mask)`  
*Resets one or more peripheral on the APB2 bus.*
- `#define rccEnableAHB(mask, lp)`  
*Enables the clock of one or more peripheral on the AHB bus.*
- `#define rccDisableAHB(mask, lp)`  
*Disables the clock of one or more peripheral on the AHB bus.*
- `#define rccResetAHB(mask)`  
*Resets one or more peripheral on the AHB bus.*

##### ADC peripherals specific RCC operations

- `#define rccEnableADC1(lp) rccEnableAPB2(RCC_APB2ENR_ADC1EN, lp)`  
*Enables the ADC1 peripheral clock.*
- `#define rccDisableADC1(lp) rccDisableAPB2(RCC_APB2ENR_ADC1EN, lp)`  
*Disables the ADC1 peripheral clock.*
- `#define rccResetADC1() rccResetAPB2(RCC_APB2RSTR_ADC1RST)`  
*Resets the ADC1 peripheral.*

##### Backup domain interface specific RCC operations

- `#define rccEnableBKPIInterface(lp) rccEnableAPB1((RCC_APB1ENR_BKPEN | RCC_APB1ENR_PWREN), lp)`  
*Enables the BKP interface clock.*
- `#define rccDisableBKPIInterface(lp) rccDisableAPB1((RCC_APB1ENR_BKPEN | RCC_APB1ENR_PWREN), lp)`  
*Disables BKP interface clock.*
- `#define rccResetBKPIInterface() rccResetAPB1(RCC_APB1ENR_BKPRST)`  
*Resets the Backup Domain interface.*
- `#define rccResetBKP() (RCC->BDCR |= RCC_BDCR_BDRST)`  
*Resets the entire Backup Domain.*

##### PWR interface specific RCC operations

- `#define rccEnablePWRInterface(lp) rccEnableAPB1(RCC_APB1ENR_PWREN, lp)`  
*Enables the PWR interface clock.*
- `#define rccDisablePWRInterface(lp) rccDisableAPB1(RCC_APB1ENR_BKPEN, lp)`  
*Disables PWR interface clock.*
- `#define rccResetPWRInterface() rccResetAPB1(RCC_APB1ENR_BKPRST)`  
*Resets the PWR interface.*

### CAN peripherals specific RCC operations

- `#define rccEnableCAN1(lp) rccEnableAPB1(RCC_APB1ENR_CAN1EN, lp)`  
*Enables the CAN1 peripheral clock.*
- `#define rccDisableCAN1(lp) rccDisableAPB1(RCC_APB1ENR_CAN1EN, lp)`  
*Disables the CAN1 peripheral clock.*
- `#define rccResetCAN1() rccResetAPB1(RCC_APB1RSTR_CAN1RST)`  
*Resets the CAN1 peripheral.*

### DMA peripherals specific RCC operations

- `#define rccEnableDMA1(lp) rccEnableAHB(RCC_AHBENR_DMA1EN, lp)`  
*Enables the DMA1 peripheral clock.*
- `#define rccDisableDMA1(lp) rccDisableAHB(RCC_AHBENR_DMA1EN, lp)`  
*Disables the DMA1 peripheral clock.*
- `#define rccResetDMA1()`  
*Resets the DMA1 peripheral.*
- `#define rccEnableDMA2(lp) rccEnableAHB(RCC_AHBENR_DMA2EN, lp)`  
*Enables the DMA2 peripheral clock.*
- `#define rccDisableDMA2(lp) rccDisableAHB(RCC_AHBENR_DMA2EN, lp)`  
*Disables the DMA2 peripheral clock.*
- `#define rccResetDMA2()`  
*Resets the DMA1 peripheral.*

### ETH peripheral specific RCC operations

- `#define rccEnableETH(lp)`  
*Enables the ETH peripheral clock.*
- `#define rccDisableETH(lp)`  
*Disables the ETH peripheral clock.*
- `#define rccResetETH() rccResetAHB(RCC_AHBRSTR_ETHMACRST)`  
*Resets the ETH peripheral.*

### I2C peripherals specific RCC operations

- `#define rccEnableI2C1(lp) rccEnableAPB1(RCC_APB1ENR_I2C1EN, lp)`  
*Enables the I2C1 peripheral clock.*
- `#define rccDisableI2C1(lp) rccDisableAPB1(RCC_APB1ENR_I2C1EN, lp)`  
*Disables the I2C1 peripheral clock.*
- `#define rccResetI2C1() rccResetAPB1(RCC_APB1RSTR_I2C1RST)`  
*Resets the I2C1 peripheral.*
- `#define rccEnableI2C2(lp) rccEnableAPB1(RCC_APB1ENR_I2C2EN, lp)`  
*Enables the I2C2 peripheral clock.*
- `#define rccDisableI2C2(lp) rccDisableAPB1(RCC_APB1ENR_I2C2EN, lp)`  
*Disables the I2C2 peripheral clock.*
- `#define rccResetI2C2() rccResetAPB1(RCC_APB1RSTR_I2C2RST)`  
*Resets the I2C2 peripheral.*

### SDIO peripheral specific RCC operations

- `#define rccEnableSDIO(lp) rccEnableAHB(RCC_AHBENR_SDIOEN, lp)`  
*Enables the SDIO peripheral clock.*
- `#define rccDisableSDIO(lp) rccDisableAHB(RCC_AHBENR_SDIOEN, lp)`  
*Disables the SDIO peripheral clock.*
- `#define rccResetSDIO()`  
*Resets the SDIO peripheral.*

### SPI peripherals specific RCC operations

- `#define rccEnableSPI1(lp) rccEnableAPB2(RCC_APB2ENR_SPI1EN, lp)`  
*Enables the SPI1 peripheral clock.*
- `#define rccDisableSPI1(lp) rccDisableAPB2(RCC_APB2ENR_SPI1EN, lp)`  
*Disables the SPI1 peripheral clock.*
- `#define rccResetSPI1() rccResetAPB2(RCC_APB2RSTR_SPI1RST)`  
*Resets the SPI1 peripheral.*
- `#define rccEnableSPI2(lp) rccEnableAPB1(RCC_APB1ENR_SPI2EN, lp)`  
*Enables the SPI2 peripheral clock.*
- `#define rccDisableSPI2(lp) rccDisableAPB1(RCC_APB1ENR_SPI2EN, lp)`  
*Disables the SPI2 peripheral clock.*
- `#define rccResetSPI2() rccResetAPB1(RCC_APB1RSTR_SPI2RST)`  
*Resets the SPI2 peripheral.*
- `#define rccEnableSPI3(lp) rccEnableAPB1(RCC_APB1ENR_SPI3EN, lp)`  
*Enables the SPI3 peripheral clock.*
- `#define rccDisableSPI3(lp) rccDisableAPB1(RCC_APB1ENR_SPI3EN, lp)`  
*Disables the SPI3 peripheral clock.*
- `#define rccResetSPI3() rccResetAPB1(RCC_APB1RSTR_SPI3RST)`  
*Resets the SPI3 peripheral.*

### TIM peripherals specific RCC operations

- `#define rccEnableTIM1(lp) rccEnableAPB2(RCC_APB2ENR_TIM1EN, lp)`  
*Enables the TIM1 peripheral clock.*
- `#define rccDisableTIM1(lp) rccDisableAPB2(RCC_APB2ENR_TIM1EN, lp)`  
*Disables the TIM1 peripheral clock.*
- `#define rccResetTIM1() rccResetAPB2(RCC_APB2RSTR_TIM1RST)`  
*Resets the TIM1 peripheral.*
- `#define rccEnableTIM2(lp) rccEnableAPB1(RCC_APB1ENR_TIM2EN, lp)`  
*Enables the TIM2 peripheral clock.*
- `#define rccDisableTIM2(lp) rccDisableAPB1(RCC_APB1ENR_TIM2EN, lp)`  
*Disables the TIM2 peripheral clock.*
- `#define rccResetTIM2() rccResetAPB1(RCC_APB1RSTR_TIM2RST)`  
*Resets the TIM2 peripheral.*
- `#define rccEnableTIM3(lp) rccEnableAPB1(RCC_APB1ENR_TIM3EN, lp)`  
*Enables the TIM3 peripheral clock.*
- `#define rccDisableTIM3(lp) rccDisableAPB1(RCC_APB1ENR_TIM3EN, lp)`  
*Disables the TIM3 peripheral clock.*
- `#define rccResetTIM3() rccResetAPB1(RCC_APB1RSTR_TIM3RST)`  
*Resets the TIM3 peripheral.*
- `#define rccEnableTIM4(lp) rccEnableAPB1(RCC_APB1ENR_TIM4EN, lp)`  
*Enables the TIM4 peripheral clock.*
- `#define rccDisableTIM4(lp) rccDisableAPB1(RCC_APB1ENR_TIM4EN, lp)`  
*Disables the TIM4 peripheral clock.*
- `#define rccResetTIM4() rccResetAPB1(RCC_APB1RSTR_TIM4RST)`  
*Resets the TIM4 peripheral.*
- `#define rccEnableTIM5(lp) rccEnableAPB1(RCC_APB1ENR_TIM5EN, lp)`  
*Enables the TIM5 peripheral clock.*
- `#define rccDisableTIM5(lp) rccDisableAPB1(RCC_APB1ENR_TIM5EN, lp)`  
*Disables the TIM5 peripheral clock.*
- `#define rccResetTIM5() rccResetAPB1(RCC_APB1RSTR_TIM5RST)`  
*Resets the TIM5 peripheral.*
- `#define rccEnableTIM8(lp) rccEnableAPB2(RCC_APB2ENR_TIM8EN, lp)`  
*Enables the TIM8 peripheral clock.*
- `#define rccDisableTIM8(lp) rccDisableAPB2(RCC_APB2ENR_TIM8EN, lp)`  
*Disables the TIM8 peripheral clock.*
- `#define rccResetTIM8() rccResetAPB2(RCC_APB2RSTR_TIM8RST)`  
*Resets the TIM8 peripheral.*

### USART/UART peripherals specific RCC operations

- `#define rccEnableUSART1(lp) rccEnableAPB2(RCC_APB2ENR_USART1EN, lp)`  
*Enables the USART1 peripheral clock.*
- `#define rccDisableUSART1(lp) rccDisableAPB2(RCC_APB2ENR_USART1EN, lp)`  
*Disables the USART1 peripheral clock.*
- `#define rccResetUSART1() rccResetAPB2(RCC_APB2RSTR_USART1RST)`  
*Resets the USART1 peripheral.*
- `#define rccEnableUSART2(lp) rccEnableAPB1(RCC_APB1ENR_USART2EN, lp)`  
*Enables the USART2 peripheral clock.*
- `#define rccDisableUSART2(lp) rccDisableAPB1(RCC_APB1ENR_USART2EN, lp)`  
*Disables the USART2 peripheral clock.*
- `#define rccResetUSART2() rccResetAPB1(RCC_APB1RSTR_USART2RST)`  
*Resets the USART2 peripheral.*
- `#define rccEnableUSART3(lp) rccEnableAPB1(RCC_APB1ENR_USART3EN, lp)`  
*Enables the USART3 peripheral clock.*
- `#define rccDisableUSART3(lp) rccDisableAPB1(RCC_APB1ENR_USART3EN, lp)`  
*Disables the USART3 peripheral clock.*
- `#define rccResetUSART3() rccResetAPB1(RCC_APB1RSTR_USART3RST)`  
*Resets the USART3 peripheral.*
- `#define rccEnableUART4(lp) rccEnableAPB1(RCC_APB1ENR_UART4EN, lp)`  
*Enables the UART4 peripheral clock.*
- `#define rccDisableUART4(lp) rccDisableAPB1(RCC_APB1ENR_UART4EN, lp)`  
*Disables the UART4 peripheral clock.*
- `#define rccResetUART4() rccResetAPB1(RCC_APB1RSTR_UART4RST)`  
*Resets the UART4 peripheral.*
- `#define rccEnableUART5(lp) rccEnableAPB1(RCC_APB1ENR_UART5EN, lp)`  
*Enables the UART5 peripheral clock.*
- `#define rccDisableUART5(lp) rccDisableAPB1(RCC_APB1ENR_UART5EN, lp)`  
*Disables the UART5 peripheral clock.*
- `#define rccResetUART5() rccResetAPB1(RCC_APB1RSTR_UART5RST)`  
*Resets the UART5 peripheral.*

### USB peripheral specific RCC operations

- `#define rccEnableUSB(lp) rccEnableAPB1(RCC_APB1ENR_USBEN, lp)`  
*Enables the USB peripheral clock.*
- `#define rccDisableUSB(lp) rccDisableAPB1(RCC_APB1ENR_USBEN, lp)`  
*Disables the USB peripheral clock.*
- `#define rccResetUSB() rccResetAPB1(RCC_APB1RSTR_USBRST)`  
*Resets the USB peripheral.*

## 8.64 stm32\_usb.h File Reference

### 8.64.1 Detailed Description

STM32 USB registers layout header.

#### Note

This file requires definitions from the ST STM32 header files `stm32f10x.h` or `stm32l1xx.h`.

### Data Structures

- struct `stm32_usb_t`  
*USB registers block.*
- struct `stm32_usb_descriptor_t`  
*USB descriptor registers block.*

## Defines

- `#define USB_ENDOPOINTS_NUMBER 7`  
*Number of the available endpoints.*
- `#define STM32_USB_BASE (APB1PERIPH_BASE + 0x5C00)`  
*USB registers block numeric address.*
- `#define STM32_USBRAM_BASE (APB1PERIPH_BASE + 0x6000)`  
*USB RAM numeric address.*
- `#define STM32_USB ((stm32_usb_t *)STM32_USB_BASE)`  
*Pointer to the USB registers block.*
- `#define STM32_USBRAM ((uint32_t *)STM32_USBRAM_BASE)`  
*Pointer to the USB RAM.*
- `#define USB_PMA_SIZE 512`  
*Size of the dedicated packet memory.*
- `#define EPR_TOGGLE_MASK`  
*Mask of all the toggling bits in the EPR register.*
- `#define USB_GET_DESCRIPTOR(ep)`  
*Returns an endpoint descriptor pointer.*
- `#define USB_ADDR2PTR(addr) ((uint32_t *)((addr) * 2 + STM32_USBRAM_BASE))`  
*Converts from a PMA address to a physical address.*

## Register aliases

- `#define RXADDR1 TXADDR0`
- `#define TXADDR1 RXADDR0`

## 8.65 tm.c File Reference

### 8.65.1 Detailed Description

Time Measurement driver code.

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

## Functions

- `void tmInit (void)`  
*Initializes the Time Measurement unit.*
- `void tmObjectInit (TimeMeasurement *tmp)`  
*Initializes a `TimeMeasurement` object.*

## 8.66 tm.h File Reference

### 8.66.1 Detailed Description

Time Measurement driver header.

## Data Structures

- `struct TimeMeasurement`  
*Time Measurement structure.*

## Functions

- void **tmInit** (void)  
*Initializes the Time Measurement unit.*
- void **tmObjectInit** (**TimeMeasurement** \*tmp)  
*Initializes a **TimeMeasurement** object.*

## Defines

- #define **tmStartMeasurement**(tmp) (tmp)->start(tmp)  
*Starts a measurement.*
- #define **tmStopMeasurement**(tmp) (tmp)->stop(tmp)  
*Stops a measurement.*

## Typedefs

- **typedef struct TimeMeasurement TimeMeasurement**  
*Type of a Time Measurement object.*

## 8.67 uart.c File Reference

### 8.67.1 Detailed Description

UART Driver code.

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

## Functions

- void **uartInit** (void)  
*UART Driver initialization.*
- void **uartObjectInit** (**UARTDriver** \*uartp)  
*Initializes the standard part of a **UARTDriver** structure.*
- void **uartStart** (**UARTDriver** \*uartp, const **UARTConfig** \*config)  
*Configures and activates the UART peripheral.*
- void **uartStop** (**UARTDriver** \*uartp)  
*Deactivates the UART peripheral.*
- void **uartStartSend** (**UARTDriver** \*uartp, size\_t n, const void \*txbuf)  
*Starts a transmission on the UART peripheral.*
- void **uartStartSendl** (**UARTDriver** \*uartp, size\_t n, const void \*txbuf)  
*Starts a transmission on the UART peripheral.*
- size\_t **uartStopSend** (**UARTDriver** \*uartp)  
*Stops any ongoing transmission.*
- size\_t **uartStopSendl** (**UARTDriver** \*uartp)  
*Stops any ongoing transmission.*
- void **uartStartReceive** (**UARTDriver** \*uartp, size\_t n, void \*rxbuf)  
*Starts a receive operation on the UART peripheral.*
- void **uartStartReceivel** (**UARTDriver** \*uartp, size\_t n, void \*rxbuf)  
*Starts a receive operation on the UART peripheral.*
- size\_t **uartStopReceive** (**UARTDriver** \*uartp)

- size\_t **uartStopReceive1** (UARTDriver \*uartp)
 

*Stops any ongoing receive operation.*

## 8.68 uart.h File Reference

### 8.68.1 Detailed Description

UART Driver macros and structures. #include "uart\_llld.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 **uartStartSend1** (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 **uartStopSend1** (UARTDriver \*uartp)
 

*Stops any ongoing transmission.*
- void **uartStartReceive** (UARTDriver \*uartp, size\_t n, void \*rdbuf)
 

*Starts a receive operation on the UART peripheral.*
- void **uartStartReceive1** (UARTDriver \*uartp, size\_t n, void \*rdbuf)
 

*Starts a receive operation on the UART peripheral.*
- size\_t **uartStopReceive** (UARTDriver \*uartp)
 

*Stops any ongoing receive operation.*
- size\_t **uartStopReceive1** (UARTDriver \*uartp)
 

*Stops any ongoing receive operation.*

#### Defines

##### UART status flags

- #define **UART\_NO\_ERROR** 0
 

*No pending conditions.*
- #define **UART\_PARITY\_ERROR** 4
 

*Parity error happened.*
- #define **UART\_FRAMING\_ERROR** 8
 

*Framing error happened.*
- #define **UART\_OVERRUN\_ERROR** 16
 

*Overflow happened.*
- #define **UART\_NOISE\_ERROR** 32
 

*Noise on the line.*
- #define **UART\_BREAK\_DETECTED** 64
 

*Break detected.*

## Enumerations

- enum `uartstate_t` { `UART_UNINIT` = 0, `UART_STOP` = 1, `UART_READY` = 2 }

*Driver state machine possible states.*

- enum `uartxstate_t` { `UART_TX_IDLE` = 0, `UART_TX_ACTIVE` = 1, `UART_TX_COMPLETE` = 2 }

*Transmitter state machine states.*

- enum `uartrxstate_t` { `UART_RX_IDLE` = 0, `UART_RX_ACTIVE` = 1, `UART_RX_COMPLETE` = 2 }

*Receiver state machine states.*

## 8.69 uart\_lld.c File Reference

### 8.69.1 Detailed Description

STM32 low level UART driver code.

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

## Functions

- `CH_IRQ_HANDLER (USART1_IRQHandler)`  
*USART1 IRQ handler.*
- `CH_IRQ_HANDLER (USART2_IRQHandler)`  
*USART2 IRQ handler.*
- `CH_IRQ_HANDLER (USART3_IRQHandler)`  
*USART3 IRQ handler.*
- void `uart_lld_init (void)`  
*Low level UART driver initialization.*
- void `uart_lld_start (UARTDriver *uartp)`  
*Configures and activates the UART peripheral.*
- void `uart_lld_stop (UARTDriver *uartp)`  
*Deactivates the UART peripheral.*
- void `uart_lld_start_send (UARTDriver *uartp, size_t n, const void *txbuf)`  
*Starts a transmission on the UART peripheral.*
- size\_t `uart_lld_stop_send (UARTDriver *uartp)`  
*Stops any ongoing transmission.*
- void `uart_lld_start_receive (UARTDriver *uartp, size_t n, void *rdbuf)`  
*Starts a receive operation on the UART peripheral.*
- size\_t `uart_lld_stop_receive (UARTDriver *uartp)`  
*Stops any ongoing receive operation.*

## Variables

- `UARTDriver UARTD1`  
*USART1 UART driver identifier.*
- `UARTDriver UARTD2`  
*USART2 UART driver identifier.*
- `UARTDriver UARTD3`  
*USART3 UART driver identifier.*

## 8.70 uart\_ll.h File Reference

### 8.70.1 Detailed Description

STM32 low level UART driver header.

#### Data Structures

- struct [UARTConfig](#)  
*Driver configuration structure.*
- struct [UARTDriver](#)  
*Structure representing an UART driver.*

#### Functions

- void [uart\\_ll\\_init](#) (void)  
*Low level UART driver initialization.*
- void [uart\\_ll\\_start](#) ([UARTDriver](#) \*uartp)  
*Configures and activates the UART peripheral.*
- void [uart\\_ll\\_stop](#) ([UARTDriver](#) \*uartp)  
*Deactivates the UART peripheral.*
- void [uart\\_ll\\_start\\_send](#) ([UARTDriver](#) \*uartp, size\_t n, const void \*txbuf)  
*Starts a transmission on the UART peripheral.*
- size\_t [uart\\_ll\\_stop\\_send](#) ([UARTDriver](#) \*uartp)  
*Stops any ongoing transmission.*
- void [uart\\_ll\\_start\\_receive](#) ([UARTDriver](#) \*uartp, size\_t n, void \*rdbuf)  
*Starts a receive operation on the UART peripheral.*
- size\_t [uart\\_ll\\_stop\\_receive](#) ([UARTDriver](#) \*uartp)  
*Stops any ongoing receive operation.*

#### Defines

##### Configuration options

- #define [STM32\\_UART\\_USE\\_USART1](#) TRUE  
*UART driver on USART1 enable switch.*
- #define [STM32\\_UART\\_USE\\_USART2](#) TRUE  
*UART driver on USART2 enable switch.*
- #define [STM32\\_UART\\_USE\\_USART3](#) TRUE  
*UART driver on USART3 enable switch.*
- #define [STM32\\_UART\\_USART1\\_IRQ\\_PRIORITY](#) 12  
*USART1 interrupt priority level setting.*
- #define [STM32\\_UART\\_USART2\\_IRQ\\_PRIORITY](#) 12  
*USART2 interrupt priority level setting.*
- #define [STM32\\_UART\\_USART3\\_IRQ\\_PRIORITY](#) 12  
*USART3 interrupt priority level setting.*
- #define [STM32\\_UART\\_USART1\\_DMA\\_PRIORITY](#) 0  
*USART1 DMA priority (0..3|lowest..highest).*
- #define [STM32\\_UART\\_USART2\\_DMA\\_PRIORITY](#) 0  
*USART2 DMA priority (0..3|lowest..highest).*
- #define [STM32\\_UART\\_USART3\\_DMA\\_PRIORITY](#) 0  
*USART3 DMA priority (0..3|lowest..highest).*
- #define [STM32\\_UART\\_DMA\\_ERROR\\_HOOK](#)(uartp) chSysHalt()

- `#define STM32_UART_USART1_RX_DMA_STREAM STM32_DMA_STREAM_ID(2, 5)`  
*DMA stream used for USART1 RX operations.*
- `#define STM32_UART_USART1_TX_DMA_STREAM STM32_DMA_STREAM_ID(2, 7)`  
*DMA stream used for USART1 TX operations.*
- `#define STM32_UART_USART2_RX_DMA_STREAM STM32_DMA_STREAM_ID(1, 5)`  
*DMA stream used for USART2 RX operations.*
- `#define STM32_UART_USART2_TX_DMA_STREAM STM32_DMA_STREAM_ID(1, 6)`  
*DMA stream used for USART2 TX operations.*
- `#define STM32_UART_USART3_RX_DMA_STREAM STM32_DMA_STREAM_ID(1, 1)`  
*DMA stream used for USART3 RX operations.*
- `#define STM32_UART_USART3_TX_DMA_STREAM STM32_DMA_STREAM_ID(1, 3)`  
*DMA stream used for USART3 TX operations.*

## Typedefs

- `typedef uint32_t uartflags_t`  
*UART driver condition flags type.*
- `typedef struct UARTDriver UARTDriver`  
*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(* uarceb_t )(UARTDriver *uartp, uartflags_t e)`  
*Receive error UART notification callback type.*

## 8.71 usb.c File Reference

### 8.71.1 Detailed Description

```
USB Driver code. #include <string.h>
#include "ch.h"
#include "hal.h"
#include "usb.h"
```

## Functions

- `void usbInit (void)`  
*USB Driver initialization.*
- `void usbObjectInit (USBDriver *usbp)`  
*Initializes the standard part of a `USBDriver` structure.*
- `void usbStart (USBDriver *usbp, const USBConfig *config)`  
*Configures and activates the USB peripheral.*
- `void usbStop (USBDriver *usbp)`  
*Deactivates the USB peripheral.*
- `void usbInitEndpoint (USBDriver *usbp, usbep_t ep, const USBEndpointConfig *epcp)`  
*Enables an endpoint.*
- `void usbDisableEndpoints (USBDriver *usbp)`  
*Disables all the active endpoints.*

- `bool_t usbStartReceive1 (USBDriver *usbp, usbep_t ep)`  
*Starts a receive transaction on an OUT endpoint.*
- `bool_t usbStartTransmit1 (USBDriver *usbp, usbep_t ep)`  
*Starts a transmit transaction on an IN endpoint.*
- `bool_t usbStallReceive1 (USBDriver *usbp, usbep_t ep)`  
*Stalls an OUT endpoint.*
- `bool_t usbStallTransmit1 (USBDriver *usbp, usbep_t ep)`  
*Stalls an IN endpoint.*
- `void _usb_reset (USBDriver *usbp)`  
*USB reset 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.*

## 8.72 usb.h File Reference

### 8.72.1 Detailed Description

USB Driver macros and structures. #include "usb\_lld.h"

#### Data Structures

- struct `USBDescriptor`  
*Type of an USB descriptor.*

#### 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 usbInitEndpoint1 (USBDriver *usbp, usbep_t ep, const USBEndpointConfig *epcp)`  
*Enables an endpoint.*
- `void usbDisableEndpoints1 (USBDriver *usbp)`  
*Disables all the active endpoints.*
- `bool_t usbStartReceive1 (USBDriver *usbp, usbep_t ep)`  
*Starts a receive transaction on an OUT endpoint.*
- `bool_t usbStartTransmit1 (USBDriver *usbp, usbep_t ep)`  
*Starts a transmit transaction on an IN endpoint.*
- `bool_t usbStallReceive1 (USBDriver *usbp, usbep_t ep)`  
*Stalls an OUT endpoint.*
- `bool_t usbStallTransmit1 (USBDriver *usbp, usbep_t ep)`

- `void _usb_reset (USBDriver *usbp)`  
*USB reset 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.*

## Defines

### 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(bcdUSB, bDeviceClass, bDeviceSubClass,bDeviceProtocol, bMaxPacketSize, idVendor,idProduct, bcdDevice, iManufacturer,iProduct, iSerialNumber, bNumConfigurations)`  
*Device Descriptor helper macro.*
- `#define USB_DESC_CONFIGURATION(wTotalLength, bNumInterfaces,bConfigurationValue, iConfiguration,bmAttributes, bMaxPower)`  
*Configuration Descriptor helper macro.*
- `#define USB_DESC_INTERFACE(blInterfaceNumber, bAlternateSetting,bNumEndpoints, blInterfaceClass,blInterfaceSubClass, blInterfaceProtocol,ilInterface)`  
*Interface Descriptor helper macro.*
- `#define USB_DESC_ENDPOINT(bEndpointAddress, bmAttributes, wMaxPacketSize,bInterval)`  
*Endpoint Descriptor helper macro.*

### Endpoint types and settings

- `#define USB_EP_MODE_TYPE 0x0003`
- `#define USB_EP_MODE_TYPE_CTRL 0x0000`
- `#define USB_EP_MODE_TYPE_ISOC 0x0001`
- `#define USB_EP_MODE_TYPE_BULK 0x0002`
- `#define USB_EP_MODE_TYPE_INTR 0x0003`
- `#define USB_EP_MODE_TRANSACTION 0x0000`
- `#define USB_EP_MODE_PACKET 0x0010`

### Macro Functions

- `#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 usbGetFrameNumber(usbp) usb_ll_get_frame_number(usbp)`  
*Returns the current frame number.*
- `#define usbGetTransmitStatusl(usbp, ep) ((usbp)->transmitting & (1 << (ep)))`  
*Returns the status of an IN endpoint.*
- `#define usbGetReceiveStatusl(usbp, ep) ((usbp)->receiving & (1 << (ep)))`  
*Returns the status of an OUT endpoint.*
- `#define usbReadPacketBuffer(usbp, ep, buf, n) usb_ll_read_packet_buffer(usbp, ep, buf, n)`  
*Reads from a dedicated packet buffer.*

- `#define usbWritePacketBuffer(usbp, ep, buf, n) usb_lld_write_packet_buffer(usbp, ep, buf, n)`  
*Writes to a dedicated packet buffer.*
- `#define usbPrepareReceive(usbp, ep, buf, n) usb_lld_prepare_receive(usbp, ep, buf, n)`  
*Prepares for a receive transaction on an OUT endpoint.*
- `#define usbPrepareTransmit(usbp, ep, buf, n) usb_lld_prepare_transmit(usbp, ep, buf, n)`  
*Prepares for a transmit transaction on an IN endpoint.*
- `#define usbGetReceiveTransactionSize(usbp, ep) usb_lld_get_transaction_size(usbp, ep)`  
*Returns the exact size of a receive transaction.*
- `#define usbGetReceivePacketSize(usbp, ep) usb_lld_get_packet_size(usbp, ep)`  
*Returns the exact size of a received packet.*
- `#define usbSetupTransfer(usbp, buf, n, endcb)`  
*Request transfer setup.*
- `#define usbReadSetup(usbp, ep, buf) usb_lld_read_setup(usbp, ep, buf)`  
*Reads a setup packet from the dedicated packet buffer.*

#### Low Level driver helper macros

- `#define _usb_isr_invoke_event_cb(usbp, evt)`  
*Common ISR code, usb event callback.*
- `#define _usb_isr_invoke_sof_cb(usbp)`  
*Common ISR code, SOF callback.*
- `#define _usb_isr_invoke_setup_cb(usbp, ep)`  
*Common ISR code, setup packet callback.*
- `#define _usb_isr_invoke_in_cb(usbp, ep)`  
*Common ISR code, IN endpoint callback.*
- `#define _usb_isr_invoke_out_cb(usbp, ep)`  
*Common ISR code, OUT endpoint event.*

#### Typedefs

- `typedef struct USBDriver USBDriver`  
*Type of a structure representing an USB driver.*
- `typedef uint8_t usbep_t`  
*Type of an endpoint identifier.*
- `typedef void(* usbcallback_t )(USBDriver *usbp)`  
*Type of an USB generic notification callback.*
- `typedef void(* usbecallback_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_t(* usbreqhandler_t )(USBDriver *usbp)`  
*Type of a requests handler callback.*
- `typedef const USBDescriptor *(* usbgetdescriptor_t )(USBDriver *usbp, uint8_t dtype, uint8_t dindex, uint16_t lang)`  
*Type of an USB descriptor-retrieving callback.*

#### Enumerations

- `enum usbstate_t {`  
`USB_UNINIT = 0, USB_STOP = 1, USB_READY = 2, USB_SELECTED = 3,`  
`USB_ACTIVE = 4 }`  
*Type of a driver state machine possible states.*
- `enum usbepstatus_t { EP_STATUS_DISABLED = 0, EP_STATUS_STALLED = 1, EP_STATUS_ACTIVE = 2 }`

- *Type of an endpoint status.*
- enum `usbep0state_t` {
 `USB_EP0_WAITING_SETUP, USB_EP0_TX, USB_EP0_WAITING_STS, USB_EP0_RX,`  
`USB_EP0_SENDING_STS, USB_EP0_ERROR }`
  - *Type of an endpoint zero state machine states.*
  - enum `usbevent_t` {
 `USB_EVENT_RESET = 0, USB_EVENT_ADDRESS = 1, USB_EVENT_CONFIGURED = 2, USB_EVENT_SUSPEND = 3,`  
`USB_EVENT_WAKEUP = 4, USB_EVENT_STALLED = 5 }`
    - *Type of an enumeration of the possible USB events.*

## 8.73 usb\_lld.c File Reference

### 8.73.1 Detailed Description

```
STM32 USB subsystem low level driver source. #include <string.h>
#include "ch.h"
#include "hal.h"
```

### Functions

- `CH_IRQ_HANDLER` (Vector8C)
  - *USB high priority interrupt handler.*
- `CH_IRQ_HANDLER` (Vector90)
  - *USB low priority interrupt handler.*
- 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.*
- size\_t `usb_lld_read_packet_buffer` (USBDriver \*usbp, `usbep_t` ep, uint8\_t \*buf, size\_t n)
  - *Reads from a dedicated packet buffer.*
- void `usb_lld_write_packet_buffer` (USBDriver \*usbp, `usbep_t` ep, const uint8\_t \*buf, size\_t n)
  - *Writes to a dedicated packet buffer.*

- void `usb_lld_prepare_receive (USBDriver *usbp, usbep_t ep, uint8_t *buf, size_t n)`  
*Prepares for a receive operation.*
- void `usb_lld_prepare_transmit (USBDriver *usbp, usbep_t ep, const uint8_t *buf, size_t n)`  
*Prepares for a transmit operation.*
- void `usb_lld_start_out (USBDriver *usbp, usbep_t ep)`  
*Starts a receive operation on an OUT endpoint.*
- void `usb_lld_start_in (USBDriver *usbp, usbep_t ep)`  
*Starts a transmit operation on an IN endpoint.*
- void `usb_lld_stall_out (USBDriver *usbp, usbep_t ep)`  
*Brings an OUT endpoint in the stalled state.*
- void `usb_lld_stall_in (USBDriver *usbp, usbep_t ep)`  
*Brings an IN endpoint in the stalled state.*
- void `usb_lld_clear_out (USBDriver *usbp, usbep_t ep)`  
*Brings an OUT endpoint in the active state.*
- void `usb_lld_clear_in (USBDriver *usbp, usbep_t ep)`  
*Brings an IN endpoint in the active state.*

## Variables

- `USBDriver USBD1`  
*USB1 driver identifier.*

### 8.73.2 Variable Documentation

#### 8.73.2.1 USBInEndpointState in

IN EP0 state.

#### 8.73.2.2 USBOutEndpointState out

OUT EP0 state.

## 8.74 usb\_ll.h File Reference

### 8.74.1 Detailed Description

STM32 USB subsystem low level driver header. #include "stm32\_usb.h"

## Data Structures

- struct `USBInEndpointState`  
*Type of an endpoint state structure.*
- struct `USBOutEndpointState`  
*Type of an 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

- void `usb_ll_init` (void)  
*Low level USB driver initialization.*
- void `usb_ll_start` (USBDriver \*usbp)  
*Configures and activates the USB peripheral.*
- void `usb_ll_stop` (USBDriver \*usbp)  
*Deactivates the USB peripheral.*
- void `usb_ll_reset` (USBDriver \*usbp)  
*USB low level reset routine.*
- void `usb_ll_set_address` (USBDriver \*usbp)  
*Sets the USB address.*
- void `usb_ll_init_endpoint` (USBDriver \*usbp, usbep\_t ep)  
*Enables an endpoint.*
- void `usb_ll_disable_endpoints` (USBDriver \*usbp)  
*Disables all the active endpoints except the endpoint zero.*
- `usbepstatus_t usb_ll_get_status_in` (USBDriver \*usbp, usbep\_t ep)  
*Returns the status of an IN endpoint.*
- `usbepstatus_t usb_ll_get_status_out` (USBDriver \*usbp, usbep\_t ep)  
*Returns the status of an OUT endpoint.*
- void `usb_ll_read_setup` (USBDriver \*usbp, usbep\_t ep, uint8\_t \*buf)  
*Reads a setup packet from the dedicated packet buffer.*
- size\_t `usb_ll_read_packet_buffer` (USBDriver \*usbp, usbep\_t ep, uint8\_t \*buf, size\_t n)  
*Reads from a dedicated packet buffer.*
- void `usb_ll_write_packet_buffer` (USBDriver \*usbp, usbep\_t ep, const uint8\_t \*buf, size\_t n)  
*Writes to a dedicated packet buffer.*
- void `usb_ll_prepare_receive` (USBDriver \*usbp, usbep\_t ep, uint8\_t \*buf, size\_t n)  
*Prepares for a receive operation.*
- void `usb_ll_prepare_transmit` (USBDriver \*usbp, usbep\_t ep, const uint8\_t \*buf, size\_t n)  
*Prepares for a transmit operation.*
- void `usb_ll_start_out` (USBDriver \*usbp, usbep\_t ep)  
*Starts a receive operation on an OUT endpoint.*
- void `usb_ll_start_in` (USBDriver \*usbp, usbep\_t ep)  
*Starts a transmit operation on an IN endpoint.*
- void `usb_ll_stall_out` (USBDriver \*usbp, usbep\_t ep)  
*Brings an OUT endpoint in the stalled state.*
- void `usb_ll_stall_in` (USBDriver \*usbp, usbep\_t ep)  
*Brings an IN endpoint in the stalled state.*
- void `usb_ll_clear_out` (USBDriver \*usbp, usbep\_t ep)  
*Brings an OUT endpoint in the active state.*
- void `usb_ll_clear_in` (USBDriver \*usbp, usbep\_t ep)  
*Brings an IN endpoint in the active state.*

## Defines

- #define `USB_MAX_ENDPOINTS` USB\_ENDPOINTS\_NUMBER  
*Maximum endpoint address.*
- #define `USB_SET_ADDRESS_MODE` USB\_LATE\_SET\_ADDRESS  
*This device requires the address change after the status packet.*
- #define `STM32_USB_USE_USB1` TRUE  
*USB1 driver enable switch.*

- `#define STM32_USB_LOW_POWER_ON_SUSPEND FALSE`  
*Enables the USB device low power mode on suspend.*
- `#define STM32_USB_USB1_HP_IRQ_PRIORITY 6`  
*USB1 interrupt priority level setting.*
- `#define STM32_USB_USB1_LP_IRQ_PRIORITY 14`  
*USB1 interrupt priority level setting.*
- `#define usb_ll_fetch_word(p) (*(uint16_t *)(p))`  
*Fetches a 16 bits word value from an USB message.*
- `#define usb_ll_get_frame_number(usbp) (STM32_USB->FNR & FNR_FN_MASK)`  
*Returns the current frame number.*
- `#define usb_ll_get_transaction_size(usbp, ep) ((usbp)->epc[ep]->out_state->rxcnt)`  
*Returns the exact size of a receive transaction.*
- `#define usb_ll_get_packet_size(usbp, ep) ((size_t)USB_GET_DESCRIPTOR(ep)->RXCOUNT & RXCOUNT_COUNT_MASK)`  
*Returns the exact size of a received packet.*

# Index

\_IOBUS\_DATA  
    PAL Driver, 135

\_adc\_isr\_error\_code  
    ADC Driver, 32

\_adc\_isr\_full\_code  
    ADC Driver, 32

\_adc\_isr\_half\_code  
    ADC Driver, 32

\_adc\_isr\_invoke\_event\_cb  
    USB Driver, 281

\_adc\_reset\_i  
    ADC Driver, 30

\_adc\_reset\_s  
    ADC Driver, 31

\_adc\_timeout\_isr  
    ADC Driver, 31

\_adc\_wakeup\_isr  
    ADC Driver, 31

\_icu\_isr\_invoke\_period\_cb  
    ICU Driver, 114

\_icu\_isr\_invoke\_width\_cb  
    ICU Driver, 113

\_pal\_lld\_init  
    PAL Driver, 133

\_pal\_lld\_setgroupmode  
    PAL Driver, 133

\_sdc\_wait\_for\_transfer\_state  
    SDC Driver, 178

\_serial\_driver\_data  
    Serial Driver, 202

\_serial\_driver\_methods  
    Serial Driver, 197

\_serial\_usb\_driver\_data  
    Serial over USB Driver, 208

\_serial\_usb\_driver\_methods  
    Serial over USB Driver, 208

\_spi\_isr\_code  
    SPI Driver, 226

\_spi\_wait\_s  
    SPI Driver, 225

\_spi\_wakeup\_isr  
    SPI Driver, 226

\_stm32\_dma\_streams  
    STM32F1xx DMA Support, 365

\_usb\_ep0in  
    USB Driver, 265

\_usb\_ep0out  
    USB Driver, 266

\_usb\_ep0setup  
    USB Driver, 264

\_usb\_isr\_invoke\_in\_cb  
    USB Driver, 282

\_usb\_isr\_invoke\_out\_cb  
    USB Driver, 282

\_usb\_isr\_invoke\_setup\_cb  
    USB Driver, 282

\_usb\_isr\_invoke\_sof\_cb  
    USB Driver, 281

\_usb\_reset  
    USB Driver, 264

adc  
    ADCDriver, 404

ADC Driver, 17

    \_adc\_isr\_error\_code, 32

    \_adc\_isr\_full\_code, 32

    \_adc\_isr\_half\_code, 32

    adc\_reset\_i, 30

    adc\_reset\_s, 31

    adc\_timeout\_isr, 31

    adc\_wakeup\_isr, 31

    ADC\_ACTIVE, 39

    ADC\_COMPLETE, 39

    ADC\_ERR\_DMAFAILURE, 40

    ADC\_ERROR, 40

    ADC\_READY, 39

    ADC\_STOP, 39

    ADC\_UNINIT, 39

    ADC\_CHANNEL\_IN0, 33

    ADC\_CHANNEL\_IN1, 33

    ADC\_CHANNEL\_IN10, 34

    ADC\_CHANNEL\_IN11, 34

    ADC\_CHANNEL\_IN12, 34

    ADC\_CHANNEL\_IN13, 34

    ADC\_CHANNEL\_IN14, 34

    ADC\_CHANNEL\_IN15, 34

    ADC\_CHANNEL\_IN2, 33

    ADC\_CHANNEL\_IN3, 33

    ADC\_CHANNEL\_IN4, 34

    ADC\_CHANNEL\_IN5, 34

    ADC\_CHANNEL\_IN6, 34

    ADC\_CHANNEL\_IN7, 34

    ADC\_CHANNEL\_IN8, 34

    ADC\_CHANNEL\_IN9, 34

    ADC\_CHANNEL\_SENSOR, 35

    ADC\_CHANNEL\_VREFINT, 35

    adc\_channels\_num\_t, 39

    ADC\_CR2\_EXTSEL\_SRC, 33

    ADC\_CR2\_EXTSEL\_SWSTART, 33

    adc\_lld\_init, 28

adc\_lld\_start, 28  
adc\_lld\_start\_conversion, 29  
adc\_lld\_stop, 29  
adc\_lld\_stop\_conversion, 29  
ADC\_SAMPLE\_13P5, 35  
ADC\_SAMPLE\_1P5, 35  
ADC\_SAMPLE\_239P5, 35  
ADC\_SAMPLE\_28P5, 35  
ADC\_SAMPLE\_41P5, 35  
ADC\_SAMPLE\_55P5, 35  
ADC\_SAMPLE\_71P5, 35  
ADC\_SAMPLE\_7P5, 35  
ADC\_SMPR1\_SMP\_AN10, 38  
ADC\_SMPR1\_SMP\_AN11, 38  
ADC\_SMPR1\_SMP\_AN12, 38  
ADC\_SMPR1\_SMP\_AN13, 38  
ADC\_SMPR1\_SMP\_AN14, 38  
ADC\_SMPR1\_SMP\_AN15, 38  
ADC\_SMPR1\_SMP\_SENSOR, 38  
ADC\_SMPR1\_SMP\_VREF, 39  
ADC\_SMPR2\_SMP\_AN0, 37  
ADC\_SMPR2\_SMP\_AN1, 37  
ADC\_SMPR2\_SMP\_AN2, 37  
ADC\_SMPR2\_SMP\_AN3, 37  
ADC\_SMPR2\_SMP\_AN4, 37  
ADC\_SMPR2\_SMP\_AN5, 38  
ADC\_SMPR2\_SMP\_AN6, 38  
ADC\_SMPR2\_SMP\_AN7, 38  
ADC\_SMPR2\_SMP\_AN8, 38  
ADC\_SMPR2\_SMP\_AN9, 38  
ADC\_SQR1\_NUM\_CH, 36  
ADC\_SQR1\_SQ13\_N, 37  
ADC\_SQR1\_SQ14\_N, 37  
ADC\_SQR1\_SQ15\_N, 37  
ADC\_SQR1\_SQ16\_N, 37  
ADC\_SQR2\_SQ10\_N, 37  
ADC\_SQR2\_SQ11\_N, 37  
ADC\_SQR2\_SQ12\_N, 37  
ADC\_SQR2\_SQ7\_N, 36  
ADC\_SQR2\_SQ8\_N, 36  
ADC\_SQR2\_SQ9\_N, 36  
ADC\_SQR3\_SQ1\_N, 36  
ADC\_SQR3\_SQ2\_N, 36  
ADC\_SQR3\_SQ3\_N, 36  
ADC\_SQR3\_SQ4\_N, 36  
ADC\_SQR3\_SQ5\_N, 36  
ADC\_SQR3\_SQ6\_N, 36  
ADC\_USE\_MUTUAL\_EXCLUSION, 30  
ADC\_USE\_WAIT, 30  
adcAcquireBus, 27  
adccallback\_t, 39  
adcConvert, 27  
ADCD1, 30  
ADCDriver, 39  
adcerror\_t, 40  
adcerrorcallback\_t, 39  
adcInit, 23  
adcObjectInit, 23  
adcReleaseBus, 27  
adcsample\_t, 39  
adcStart, 24  
adcStartConversion, 24  
adcStartConversionl, 25  
adcstate\_t, 39  
adcStop, 24  
adcStopConversion, 26  
adcStopConversionl, 26  
STM32\_ADC\_ADC1\_DMA\_PRIORITY, 36  
STM32\_ADC\_ADC1\_IRQ\_PRIORITY, 36  
STM32\_ADC\_USE\_ADC1, 35  
adc.c, 472  
adc.h, 473  
ADC1\_2\_IRQHandler  
    STM32F100 HAL Support, 300  
    STM32F103 HAL Support, 322  
    STM32F105/F107 HAL Support, 342  
ADC3\_IRQHandler  
    STM32F103 HAL Support, 325  
ADC\_ACTIVE  
    ADC Driver, 39  
ADC\_COMPLETE  
    ADC Driver, 39  
ADC\_ERR\_DMAFAILURE  
    ADC Driver, 40  
ADC\_ERROR  
    ADC Driver, 40  
ADC\_READY  
    ADC Driver, 39  
ADC\_STOP  
    ADC Driver, 39  
ADC\_UNINIT  
    ADC Driver, 39  
ADC\_CHANNEL\_IN0  
    ADC Driver, 33  
ADC\_CHANNEL\_IN1  
    ADC Driver, 33  
ADC\_CHANNEL\_IN10  
    ADC Driver, 34  
ADC\_CHANNEL\_IN11  
    ADC Driver, 34  
ADC\_CHANNEL\_IN12  
    ADC Driver, 34  
ADC\_CHANNEL\_IN13  
    ADC Driver, 34  
ADC\_CHANNEL\_IN14  
    ADC Driver, 34  
ADC\_CHANNEL\_IN15  
    ADC Driver, 34  
ADC\_CHANNEL\_IN2  
    ADC Driver, 33  
ADC\_CHANNEL\_IN3  
    ADC Driver, 33  
ADC\_CHANNEL\_IN4  
    ADC Driver, 34  
ADC\_CHANNEL\_IN5  
    ADC Driver, 34  
ADC\_CHANNEL\_IN6  
    ADC Driver, 34

ADC\_CHANNEL\_IN7  
    ADC Driver, 34  
ADC\_CHANNEL\_IN8  
    ADC Driver, 34  
ADC\_CHANNEL\_IN9  
    ADC Driver, 34  
ADC\_CHANNEL\_SENSOR  
    ADC Driver, 35  
ADC\_CHANNEL\_VREFINT  
    ADC Driver, 35  
adc\_channels\_num\_t  
    ADC Driver, 39  
ADC\_CR2\_EXTSEL\_SRC  
    ADC Driver, 33  
ADC\_CR2\_EXTSEL\_SWSTART  
    ADC Driver, 33  
adc\_lld.c, 474  
adc\_lld.h, 474  
adc\_lld\_init  
    ADC Driver, 28  
adc\_lld\_start  
    ADC Driver, 28  
adc\_lld\_start\_conversion  
    ADC Driver, 29  
adc\_lld\_stop  
    ADC Driver, 29  
adc\_lld\_stop\_conversion  
    ADC Driver, 29  
ADC\_SAMPLE\_13P5  
    ADC Driver, 35  
ADC\_SAMPLE\_1P5  
    ADC Driver, 35  
ADC\_SAMPLE\_239P5  
    ADC Driver, 35  
ADC\_SAMPLE\_28P5  
    ADC Driver, 35  
ADC\_SAMPLE\_41P5  
    ADC Driver, 35  
ADC\_SAMPLE\_55P5  
    ADC Driver, 35  
ADC\_SAMPLE\_71P5  
    ADC Driver, 35  
ADC\_SAMPLE\_7P5  
    ADC Driver, 35  
ADC\_SAMPLE\_71P5  
    ADC Driver, 35  
ADC\_SMPR1\_SMP\_AN10  
    ADC Driver, 38  
ADC\_SMPR1\_SMP\_AN11  
    ADC Driver, 38  
ADC\_SMPR1\_SMP\_AN12  
    ADC Driver, 38  
ADC\_SMPR1\_SMP\_AN13  
    ADC Driver, 38  
ADC\_SMPR1\_SMP\_AN14  
    ADC Driver, 38  
ADC\_SMPR1\_SMP\_AN15  
    ADC Driver, 38  
ADC\_SMPR1\_SMP\_SENSOR  
    ADC Driver, 38  
ADC\_SMPR1\_SMP\_VREF  
    ADC Driver, 39  
ADC\_SMPR2\_SMP\_AN0  
    ADC Driver, 37  
ADC\_SMPR2\_SMP\_AN1  
    ADC Driver, 37  
ADC\_SMPR2\_SMP\_AN2  
    ADC Driver, 37  
ADC\_SMPR2\_SMP\_AN3  
    ADC Driver, 37  
ADC\_SMPR2\_SMP\_AN4  
    ADC Driver, 37  
ADC\_SMPR2\_SMP\_AN5  
    ADC Driver, 38  
ADC\_SMPR2\_SMP\_AN6  
    ADC Driver, 38  
ADC\_SMPR2\_SMP\_AN7  
    ADC Driver, 38  
ADC\_SMPR2\_SMP\_AN8  
    ADC Driver, 38  
ADC\_SMPR2\_SMP\_AN9  
    ADC Driver, 38  
ADC\_SQR1\_NUM\_CH  
    ADC Driver, 36  
ADC\_SQR1\_SQ13\_N  
    ADC Driver, 37  
ADC\_SQR1\_SQ14\_N  
    ADC Driver, 37  
ADC\_SQR1\_SQ15\_N  
    ADC Driver, 37  
ADC\_SQR1\_SQ16\_N  
    ADC Driver, 37  
ADC\_SQR2\_SQ10\_N  
    ADC Driver, 37  
ADC\_SQR2\_SQ11\_N  
    ADC Driver, 37  
ADC\_SQR2\_SQ12\_N  
    ADC Driver, 37  
ADC\_SQR2\_SQ7\_N  
    ADC Driver, 36  
ADC\_SQR2\_SQ9\_N  
    ADC Driver, 36  
ADC\_SQR3\_SQ1\_N  
    ADC Driver, 36  
ADC\_SQR3\_SQ2\_N  
    ADC Driver, 36  
ADC\_SQR3\_SQ8\_N  
    ADC Driver, 36  
ADC\_SQR2\_SQ3\_N  
    ADC Driver, 36  
ADC\_SQR3\_SQ9\_N  
    ADC Driver, 36  
ADC\_SQR3\_SQ4\_N  
    ADC Driver, 36  
ADC\_SQR3\_SQ5\_N  
    ADC Driver, 36  
ADC\_SQR3\_SQ6\_N  
    ADC Driver, 36  
ADC\_USE\_MUTUAL\_EXCLUSION  
    ADC Driver, 30  
    Configuration, 15  
ADC\_USE\_WAIT

ADC Driver, 30  
Configuration, 14  
adcAcquireBus  
    ADC Driver, 27  
adccallback\_t  
    ADC Driver, 39  
ADCCfg, 399  
ADCCConversionGroup, 399  
    circular, 401  
    cr1, 401  
    cr2, 401  
    end\_cb, 401  
    error\_cb, 401  
    num\_channels, 401  
    smpr1, 401  
    smpr2, 401  
    sqr1, 402  
    sqr2, 402  
    sqr3, 402  
adcConvert  
    ADC Driver, 27  
ADCD1  
    ADC Driver, 30  
ADCDriver, 402  
    adc, 404  
    ADC Driver, 39  
    config, 404  
    depth, 404  
    dmamode, 405  
    dmastp, 405  
    grpp, 404  
    mutex, 404  
    samples, 404  
    state, 404  
    thread, 404  
adcerror\_t  
    ADC Driver, 40  
adcerrorcallback\_t  
    ADC Driver, 39  
adclinit  
    ADC Driver, 23  
adcObjectInit  
    ADC Driver, 23  
adcReleaseBus  
    ADC Driver, 27  
adcsample\_t  
    ADC Driver, 39  
adcStart  
    ADC Driver, 24  
adcStartConversion  
    ADC Driver, 24  
adcStartConversionl  
    ADC Driver, 25  
adcstate\_t  
    ADC Driver, 39  
adcStop  
    ADC Driver, 24  
adcStopConversion  
    ADC Driver, 26  
adcStopConversionl  
    ADC Driver, 26  
    addr  
        I2CDriver, 422  
    address  
        USBDriver, 467  
    assignment  
        CANFilter, 410  
    bdtr  
        PWMConfig, 437  
    best  
        TimeMeasurement, 456  
    btr  
        CANConfig, 406  
    callback  
        GPTConfig, 418  
        PWMChannelConfig, 435  
        PWMConfig, 437  
        RTCCallbackConfig, 441  
        RTCDriver, 442  
    can  
        CANDriver, 409  
    CAN Driver, 40  
        CAN\_READY, 53  
        CAN\_SLEEP, 53  
        CAN\_STARTING, 53  
        CAN\_STOP, 53  
        CAN\_UNINIT, 53  
        CAN\_BTR\_BRP, 52  
        CAN\_BTR\_SJW, 52  
        CAN\_BTR\_TS1, 52  
        CAN\_BTR\_TS2, 52  
        CAN\_BUS\_OFF\_ERROR, 51  
        CAN\_FRAMING\_ERROR, 51  
        CAN\_IDE\_EXT, 52  
        CAN\_IDE\_STD, 52  
        CAN\_LIMIT\_ERROR, 51  
        CAN\_LIMIT\_WARNING, 51  
        can\_lld\_can\_receive, 50  
        can\_lld\_can\_transmit, 49  
        can\_lld\_init, 49  
        can\_lld\_receive, 50  
        can\_lld\_sleep, 51  
        can\_lld\_start, 49  
        can\_lld\_stop, 49  
        can\_lld\_transmit, 50  
        can\_lld\_wakeup, 51  
        CAN\_MAX\_FILTERS, 52  
        CAN\_OVERFLOW\_ERROR, 51  
        CAN\_RTR\_DATA, 53  
        CAN\_RTR\_REMOTE, 53  
        CAN\_SUPPORTS\_SLEEP, 52  
        CAN\_USE\_SLEEP\_MODE, 52  
        canAddFlagsl, 52  
        CAND1, 51  
        canGetAndClearFlags, 47  
        canInit, 43

canObjectInit, 43  
canReceive, 46  
canSleep, 47  
canStart, 44  
canstate\_t, 53  
canstatus\_t, 53  
canStop, 44  
canTransmit, 45  
canWakeUp, 48  
CH\_IRQ\_HANDLER, 48  
STM32\_CAN\_CAN1 IRQ\_PRIORITY, 53  
STM32\_CAN\_USE\_CAN1, 53  
can.c, 478  
can.h, 478  
CAN1\_RX0\_IRQHandler  
    STM32F103 HAL Support, 322  
    STM32F105/F107 HAL Support, 343  
CAN1\_RX1\_IRQHandler  
    STM32F103 HAL Support, 323  
    STM32F105/F107 HAL Support, 343  
CAN1\_SCE\_IRQHandler  
    STM32F103 HAL Support, 323  
    STM32F105/F107 HAL Support, 343  
CAN1\_TX\_IRQHandler  
    STM32F103 HAL Support, 322  
    STM32F105/F107 HAL Support, 342  
CAN2\_RX0\_IRQHandler  
    STM32F105/F107 HAL Support, 346  
CAN2\_RX1\_IRQHandler  
    STM32F105/F107 HAL Support, 346  
CAN2\_SCE\_IRQHandler  
    STM32F105/F107 HAL Support, 346  
CAN2\_TX\_IRQHandler  
    STM32F105/F107 HAL Support, 346  
CAN\_READY  
    CAN Driver, 53  
CAN\_SLEEP  
    CAN Driver, 53  
CAN\_STARTING  
    CAN Driver, 53  
CAN\_STOP  
    CAN Driver, 53  
CAN\_UNINIT  
    CAN Driver, 53  
CAN\_BTR\_BRP  
    CAN Driver, 52  
CAN\_BTR\_SJW  
    CAN Driver, 52  
CAN\_BTR\_TS1  
    CAN Driver, 52  
CAN\_BTR\_TS2  
    CAN Driver, 52  
CAN\_BUS\_OFF\_ERROR  
    CAN Driver, 51  
CAN\_FRAMING\_ERROR  
    CAN Driver, 51  
CAN\_IDE\_EXT  
    CAN Driver, 52  
CAN\_IDE\_STD  
    CAN Driver, 52  
CAN Driver, 52  
CAN\_LIMIT\_ERROR  
    CAN Driver, 51  
CAN\_LIMIT\_WARNING  
    CAN Driver, 51  
can\_lld\_c, 479  
can\_lld\_h, 480  
can\_lld\_can\_receive  
    CAN Driver, 50  
can\_lld\_can\_transmit  
    CAN Driver, 49  
can\_lld\_init  
    CAN Driver, 49  
can\_lld\_receive  
    CAN Driver, 50  
can\_lld\_sleep  
    CAN Driver, 51  
can\_lld\_start  
    CAN Driver, 49  
can\_lld\_stop  
    CAN Driver, 49  
can\_lld\_transmit  
    CAN Driver, 50  
can\_lld\_wakeup  
    CAN Driver, 51  
CAN\_MAX\_FILTERS  
    CAN Driver, 52  
CAN\_OVERFLOW\_ERROR  
    CAN Driver, 51  
CAN\_RTR\_DATA  
    CAN Driver, 53  
CAN\_RTR\_REMOTE  
    CAN Driver, 53  
CAN\_SUPPORTS\_SLEEP  
    CAN Driver, 52  
CAN\_USE\_SLEEP\_MODE  
    CAN Driver, 52  
    Configuration, 15  
canAddFlags()  
    CAN Driver, 52  
CANConfig, 405  
    btr, 406  
    filters, 406  
    mcr, 406  
    num, 406  
CAND1  
    CAN Driver, 51  
CANDriver, 406  
    can, 409  
    config, 408  
    error\_event, 408  
    rxfull\_event, 408  
    rxsem, 408  
    sleep\_event, 409  
    state, 408  
    status, 409  
    txempty\_event, 408  
    txsem, 408  
    wakeup\_event, 409

CANFilter, 409  
  assignment, 410  
  mode, 409  
  register1, 410  
  register2, 410  
  scale, 410  
canGetAndClearFlags  
  CAN Driver, 47  
canInit  
  CAN Driver, 43  
canObjectInit  
  CAN Driver, 43  
canReceive  
  CAN Driver, 46  
CANTxFrame, 410  
  data16, 411  
  data32, 411  
  data8, 411  
  DLC, 410  
  EID, 411  
  FMI, 410  
  IDE, 411  
  RTR, 411  
  SID, 411  
  TIME, 410  
canSleep  
  CAN Driver, 47  
canStart  
  CAN Driver, 44  
canstate\_t  
  CAN Driver, 53  
canstatus\_t  
  CAN Driver, 53  
canStop  
  CAN Driver, 44  
canTransmit  
  CAN Driver, 45  
CANTxFrame, 411  
  data16, 412  
  data32, 412  
  data8, 412  
  DLC, 411  
  EID, 412  
  IDE, 412  
  RTR, 411  
  SID, 412  
canWakeup  
  CAN Driver, 48  
cardmode  
  SDCDriver, 444  
cb  
  EXTChannelConfig, 413  
CEC\_IRQHandler  
  STM32F100 HAL Support, 302  
CH\_IRQ\_HANDLER  
  CAN Driver, 48  
  EXT Driver, 59, 60  
  I2C Driver, 95, 96  
  RTC Driver, 165  
  SDC Driver, 179  
  Serial Driver, 194  
  STM32F1xx DMA Support, 362–364  
  UART Driver, 243  
  USB Driver, 267  
channel  
  stm32\_dma\_stream\_t, 453  
channels  
  EXTConfig, 414  
  PWMConfig, 437  
cid  
  SDCDriver, 444  
circular  
  ADCConversionGroup, 401  
clock  
  GPTDriver, 420  
  ICUDriver, 427  
  PWMDriver, 440  
clock\_speed  
  I2CConfig, 420  
cnt  
  MMCDriver, 431  
config  
  ADCDriver, 404  
  CANDriver, 408  
  EXTDriver, 416  
  GPTDriver, 420  
  I2CDriver, 422  
  ICUDriver, 427  
  MMCDriver, 430  
  PWMDriver, 440  
  SDCDriver, 444  
  SPIDriver, 452  
  UARTDriver, 461  
  USBDriver, 466  
Configuration, 11  
  ADC\_USE\_MUTUAL\_EXCLUSION, 15  
  ADC\_USE\_WAIT, 14  
  CAN\_USE\_SLEEP\_MODE, 15  
  HAL\_USE\_ADC, 13  
  HAL\_USE\_CAN, 13  
  HAL\_USE\_EXT, 13  
  HAL\_USE\_GPT, 13  
  HAL\_USE\_I2C, 13  
  HAL\_USE\_ICU, 14  
  HAL\_USE\_MAC, 14  
  HAL\_USE\_MMCSPI, 14  
  HAL\_USE\_PAL, 13  
  HAL\_USE\_PWM, 14  
  HAL\_USE\_RTC, 14  
  HAL\_USE\_SDC, 14  
  HAL\_USE\_SERIAL, 14  
  HAL\_USE\_SERIALUSB, 14  
  HAL\_USE\_SPI, 14  
  HAL\_USE\_UART, 14  
  HAL\_USE\_USB, 14  
  I2C\_USE\_MUTUAL\_EXCLUSION, 15  
  MAC\_USE\_EVENTS, 15  
  MMC\_NICE\_WAITING, 15

MMC\_POLLING\_DELAY, 15  
MMC\_POLLING\_INTERVAL, 15  
MMC\_SECTOR\_SIZE, 15  
MMC\_USE\_SPI\_POLLING, 15  
SDC\_INIT\_RETRY, 15  
SDC\_MMC\_SUPPORT, 16  
SDC\_NICE\_WAITING, 16  
SERIAL\_BUFFERS\_SIZE, 16  
SERIAL\_DEFAULT\_BITRATE, 16  
SERIAL\_USB\_BUFFERS\_SIZE, 16  
SPI\_USE\_MUTUAL\_EXCLUSION, 16  
SPI\_USE\_WAIT, 16  
configuration  
    USBDriver, 467  
cr1  
    ADCConversionGroup, 401  
    SPIConfig, 450  
    UARTConfig, 458  
cr2  
    ADCConversionGroup, 401  
    PWMConfig, 437  
    UARTConfig, 458  
cr3  
    UARTConfig, 458  
crh  
    stm32\_gpio\_setup\_t, 454  
crl  
    stm32\_gpio\_setup\_t, 454  
csd  
    SDCDriver, 444  
  
data16  
    CANRxFrame, 411  
    CANTxFrame, 412  
data32  
    CANRxFrame, 411  
    CANTxFrame, 412  
data8  
    CANRxFrame, 411  
    CANTxFrame, 412  
depth  
    ADCDriver, 404  
DLC  
    CANRxFrame, 410  
    CANTxFrame, 411  
DMA1\_Ch1\_IRQHandler  
    STM32F100 HAL Support, 300  
    STM32F103 HAL Support, 322  
    STM32F105/F107 HAL Support, 342  
DMA1\_Ch2\_IRQHandler  
    STM32F100 HAL Support, 300  
    STM32F103 HAL Support, 322  
    STM32F105/F107 HAL Support, 342  
DMA1\_Ch3\_IRQHandler  
    STM32F100 HAL Support, 300  
    STM32F103 HAL Support, 322  
    STM32F105/F107 HAL Support, 342  
DMA1\_Ch4\_IRQHandler  
    STM32F100 HAL Support, 300  
    STM32F103 HAL Support, 322  
    STM32F105/F107 HAL Support, 342  
STM32F103 HAL Support, 322  
STM32F105/F107 HAL Support, 342  
DMA1\_Ch5\_IRQHandler  
    STM32F100 HAL Support, 300  
    STM32F103 HAL Support, 322  
    STM32F105/F107 HAL Support, 342  
DMA1\_Ch6\_IRQHandler  
    STM32F100 HAL Support, 300  
    STM32F103 HAL Support, 322  
    STM32F105/F107 HAL Support, 342  
DMA1\_Ch7\_IRQHandler  
    STM32F100 HAL Support, 300  
    STM32F103 HAL Support, 322  
    STM32F105/F107 HAL Support, 342  
DMA2\_Ch1\_IRQHandler  
    STM32F103 HAL Support, 326  
    STM32F105/F107 HAL Support, 345  
DMA2\_Ch2\_IRQHandler  
    STM32F103 HAL Support, 326  
    STM32F105/F107 HAL Support, 345  
DMA2\_Ch3\_IRQHandler  
    STM32F103 HAL Support, 326  
    STM32F105/F107 HAL Support, 345  
DMA2\_Ch4\_5\_IRQHandler  
    STM32F103 HAL Support, 326  
DMA2\_Ch4\_IRQHandler  
    STM32F105/F107 HAL Support, 345  
DMA2\_Ch5\_IRQHandler  
    STM32F105/F107 HAL Support, 345  
dmaInit  
    STM32F1xx DMA Support, 364  
dmamode  
    ADCDriver, 405  
    I2CDriver, 422  
    UARTDriver, 461  
dmarx  
    I2CDriver, 423  
    SPIDriver, 452  
    UARTDriver, 461  
dmaStartMemcpy  
    STM32F1xx DMA Support, 371  
dmastp  
    ADCDriver, 405  
dmaStreamAllocate  
    STM32F1xx DMA Support, 364  
dmaStreamClearInterrupt  
    STM32F1xx DMA Support, 371  
dmaStreamDisable  
    STM32F1xx DMA Support, 370  
dmaStreamEnable  
    STM32F1xx DMA Support, 370  
dmaStreamGetSize  
    STM32F1xx DMA Support, 369  
dmaStreamRelease  
    STM32F1xx DMA Support, 365  
dmaStreamSetMemory0  
    STM32F1xx DMA Support, 368  
dmaStreamSetMode  
    STM32F1xx DMA Support, 369

dmaStreamSetPeripheral  
    STM32F1xx DMA Support, 367

dmaStreamSetTransactionSize  
    STM32F1xx DMA Support, 368

dmatx  
    I2CDriver, 423  
    SPIDriver, 452  
    UARTDriver, 461

dmaWaitCompletion  
    STM32F1xx DMA Support, 372

duty\_cycle  
    I2CConfig, 421

EID  
    CANRxFrame, 411  
    CANTxFrame, 412

end\_cb  
    ADCConversionGroup, 401  
    SPIConfig, 450

ep0endcb  
    USBDriver, 467

ep0n  
    USBDriver, 467

ep0next  
    USBDriver, 466

ep0state  
    USBDriver, 466

EP\_STATUS\_ACTIVE  
    USB Driver, 287

EP\_STATUS\_DISABLED  
    USB Driver, 287

EP\_STATUS\_STALLED  
    USB Driver, 287

ep\_mode  
    USBEndpointConfig, 469

epc  
    USBDriver, 466

EPR  
    stm32\_usb\_t, 455

EPR\_TOGGLE\_MASK  
    USB Driver, 283

error\_cb  
    ADCConversionGroup, 401

error\_event  
    CANDriver, 408

errors  
    I2CDriver, 422

ETH\_IRQHandler  
    STM32F105/F107 HAL Support, 345

ETH\_WKUP\_IRQHandler  
    STM32F105/F107 HAL Support, 345

event\_cb  
    USBCConfig, 464

expchannel\_t  
    EXT Driver, 64

EXT Driver, 53  
    CH\_IRQ\_HANDLER, 59, 60  
    expchannel\_t, 64  
    EXT\_CHANNELS\_MASK, 62

ext\_lld\_channel\_disable, 61  
ext\_lld\_channel\_enable, 61  
ext\_lld\_init, 60  
ext\_lld\_start, 61  
ext\_lld\_stop, 61  
EXT\_MAX\_CHANNELS, 62  
EXT\_MODE\_EXTI, 62  
EXT\_MODE\_GPIOA, 62  
EXT\_MODE\_GPIOB, 62  
EXT\_MODE\_GPIOC, 62  
EXT\_MODE\_GPIOD, 62  
EXT\_MODE\_GPIOE, 63  
EXT\_MODE\_GPIOF, 63  
EXT\_MODE\_GPIOG, 63  
EXT\_MODE\_GPIOH, 63  
EXT\_MODE\_GPIOI, 63  
extcallback\_t, 64  
extChannelDisable, 59  
extChannelEnable, 58  
EXTD1, 62  
extInit, 57  
extObjectInit, 57  
extStart, 57  
extStop, 58  
STM32\_EXT\_EXTI0\_IRQ\_PRIORITY, 63  
STM32\_EXT\_EXTI10\_15\_IRQ\_PRIORITY, 63  
STM32\_EXT\_EXTI16\_IRQ\_PRIORITY, 64  
STM32\_EXT\_EXTI17\_IRQ\_PRIORITY, 64  
STM32\_EXT\_EXTI18\_IRQ\_PRIORITY, 64  
STM32\_EXT\_EXTI19\_IRQ\_PRIORITY, 64  
STM32\_EXT\_EXTI1\_IRQ\_PRIORITY, 63  
STM32\_EXT\_EXTI20\_IRQ\_PRIORITY, 64  
STM32\_EXT\_EXTI21\_IRQ\_PRIORITY, 64  
STM32\_EXT\_EXTI22\_IRQ\_PRIORITY, 64  
STM32\_EXT\_EXTI2\_IRQ\_PRIORITY, 63  
STM32\_EXT\_EXTI3\_IRQ\_PRIORITY, 63  
STM32\_EXT\_EXTI4\_IRQ\_PRIORITY, 63  
STM32\_EXT\_EXTI5\_9\_IRQ\_PRIORITY, 63

ext.c, 481

EXT\_CHANNELS\_MASK  
    EXT Driver, 62

ext\_lld.c, 482

ext\_lld.h, 483

ext\_lld\_channel\_disable  
    EXT Driver, 61

ext\_lld\_channel\_enable  
    EXT Driver, 61

ext\_lld\_init  
    EXT Driver, 60

ext\_lld\_start  
    EXT Driver, 61

ext\_lld\_stop  
    EXT Driver, 61

EXT\_MAX\_CHANNELS  
    EXT Driver, 62

EXT\_MODE\_EXTI  
    EXT Driver, 62

EXT\_MODE\_GPIOA  
    EXT Driver, 62

EXT\_MODE\_GPIOB  
  EXT Driver, 62  
EXT\_MODE\_GPIOC  
  EXT Driver, 62  
EXT\_MODE\_GPIOD  
  EXT Driver, 62  
EXT\_MODE\_GPIOE  
  EXT Driver, 63  
EXT\_MODE\_GPIOF  
  EXT Driver, 63  
EXT\_MODE\_GPIOG  
  EXT Driver, 63  
EXT\_MODE\_GPIOH  
  EXT Driver, 63  
EXT\_MODE\_GPIOI  
  EXT Driver, 63  
extcallback\_t  
  EXT Driver, 64  
EXTChannelConfig, 412  
  cb, 413  
  mode, 413  
extChannelDisable  
  EXT Driver, 59  
extChannelEnable  
  EXT Driver, 58  
EXTConfig, 414  
  channels, 414  
  exti, 415  
EXTD1  
  EXT Driver, 62  
EXTDriver, 415  
  config, 416  
  state, 416  
exti  
  EXTConfig, 415  
EXTI0\_IRQHandler  
  STM32F100 HAL Support, 299  
  STM32F103 HAL Support, 321  
  STM32F105/F107 HAL Support, 341  
EXTI15\_10\_IRQHandler  
  STM32F100 HAL Support, 302  
  STM32F103 HAL Support, 324  
  STM32F105/F107 HAL Support, 344  
EXTI1\_IRQHandler  
  STM32F100 HAL Support, 299  
  STM32F103 HAL Support, 321  
  STM32F105/F107 HAL Support, 341  
EXTI2\_IRQHandler  
  STM32F100 HAL Support, 299  
  STM32F103 HAL Support, 321  
  STM32F105/F107 HAL Support, 342  
EXTI3\_IRQHandler  
  STM32F100 HAL Support, 300  
  STM32F103 HAL Support, 321  
  STM32F105/F107 HAL Support, 342  
EXTI4\_IRQHandler  
  STM32F100 HAL Support, 300  
  STM32F103 HAL Support, 322  
  STM32F105/F107 HAL Support, 342  
EXTI9\_5\_IRQHandler  
  STM32F100 HAL Support, 300  
  STM32F103 HAL Support, 323  
  STM32F105/F107 HAL Support, 343  
extInit  
  EXT Driver, 57  
extObjectInit  
  EXT Driver, 57  
extStart  
  EXT Driver, 57  
extStop  
  EXT Driver, 58  
filters  
  CANConfig, 406  
FLASH\_IRQHandler  
  STM32F100 HAL Support, 299  
  STM32F103 HAL Support, 321  
  STM32F105/F107 HAL Support, 341  
FMI  
  CANRxFrame, 410  
frequency  
  GPTConfig, 418  
  ICUConfig, 425  
  PWMConfig, 437  
FSMC\_IRQHandler  
  STM32F103 HAL Support, 325  
get\_descriptor\_cb  
  USBConfig, 464  
GPT Driver, 64  
  GPT\_CONTINUOUS, 77  
  GPT\_ONESHOT, 77  
  GPT\_READY, 77  
  GPT\_STOP, 77  
  GPT\_UNINIT, 77  
  gpt\_lld\_init, 73  
  gpt\_lld\_polled\_delay, 74  
  gpt\_lld\_start, 73  
  gpt\_lld\_start\_timer, 73  
  gpt\_lld\_stop, 73  
  gpt\_lld\_stop\_timer, 74  
  gptcallback\_t, 77  
  gptcnt\_t, 77  
  GPTD1, 74  
  GPTD2, 74  
  GPTD3, 74  
  GPTD4, 75  
  GPTD5, 75  
  GPTD8, 75  
  GPTDriver, 77  
  gptfreq\_t, 77  
  gptInit, 68  
  gptObjectInit, 68  
  gptPolledDelay, 72  
  gptStart, 68  
  gptStartContinuous, 69  
  gptStartContinuousl, 70  
  gptStartOneShot, 70

gptStartOneShotI, 71  
gptstate\_t, 77  
gptStop, 69  
gptStopTimer, 71  
gptStopTimerI, 72  
STM32\_GPT\_TIM1\_IRQ\_PRIORITY, 76  
STM32\_GPT\_TIM2\_IRQ\_PRIORITY, 76  
STM32\_GPT\_TIM3\_IRQ\_PRIORITY, 76  
STM32\_GPT\_TIM4\_IRQ\_PRIORITY, 76  
STM32\_GPT\_TIM5\_IRQ\_PRIORITY, 76  
STM32\_GPT\_TIM8\_IRQ\_PRIORITY, 77  
STM32\_GPT\_USE\_TIM1, 75  
STM32\_GPT\_USE\_TIM2, 75  
STM32\_GPT\_USE\_TIM3, 75  
STM32\_GPT\_USE\_TIM4, 76  
STM32\_GPT\_USE\_TIM5, 76  
STM32\_GPT\_USE\_TIM8, 76  
gpt.c, 485  
gpt.h, 485  
GPT\_CONTINUOUS  
    GPT Driver, 77  
GPT\_ONESHOT  
    GPT Driver, 77  
GPT\_READY  
    GPT Driver, 77  
GPT\_STOP  
    GPT Driver, 77  
GPT\_UNINIT  
    GPT Driver, 77  
gpt\_lld.c, 486  
gpt\_lld.h, 487  
gpt\_lld\_init  
    GPT Driver, 73  
gpt\_lld\_polled\_delay  
    GPT Driver, 74  
gpt\_lld\_start  
    GPT Driver, 73  
gpt\_lld\_start\_timer  
    GPT Driver, 73  
gpt\_lld\_stop  
    GPT Driver, 73  
gpt\_lld\_stop\_timer  
    GPT Driver, 74  
gptcallback\_t  
    GPT Driver, 77  
gptcnt\_t  
    GPT Driver, 77  
GPTConfig, 416  
    callback, 418  
    frequency, 418  
GPTD1  
    GPT Driver, 74  
GPTD2  
    GPT Driver, 74  
GPTD3  
    GPT Driver, 74  
GPTD4  
    GPT Driver, 75  
GPTD5  
    GPT Driver, 75  
GPT Driver, 75  
GPTD8  
    GPT Driver, 75  
GPTDriver, 418  
    clock, 420  
    config, 420  
    GPT Driver, 77  
    state, 420  
    tim, 420  
gptfreq\_t  
    GPT Driver, 77  
gptInit  
    GPT Driver, 68  
gptObjectInit  
    GPT Driver, 68  
gptPolledDelay  
    GPT Driver, 72  
gptStart  
    GPT Driver, 68  
gptStartContinuous  
    GPT Driver, 69  
gptStartContinuousI  
    GPT Driver, 70  
gptStartOneShot  
    GPT Driver, 70  
gptStartOneShotI  
    GPT Driver, 71  
gptstate\_t  
    GPT Driver, 77  
gptStop  
    GPT Driver, 69  
gptStopTimer  
    GPT Driver, 71  
gptStopTimerI  
    GPT Driver, 72  
grpp  
    ADCDriver, 404  
HAL, 9  
HAL Driver, 77  
    HAL\_IMPLEMENTS\_COUNTERS, 85  
    hal\_lld\_get\_counter\_frequency, 87  
    hal\_lld\_get\_counter\_value, 86  
    hal\_lld\_init, 81  
    halclock\_t, 87  
    halGetCounterFrequency, 83  
    halGetCounterValue, 83  
    hallInit, 80  
    hallsCounterWithin, 84  
    halPolledDelay, 85  
    halrtcnt\_t, 87  
    MS2RTT, 82  
    S2RTT, 82  
    stm32\_clock\_init, 82  
    STM32\_HSE\_ENABLED, 86  
    STM32\_HSI\_ENABLED, 86  
    STM32\_HSICLK, 85  
    STM32\_LSE\_ENABLED, 86  
    STM32\_LSI\_ENABLED, 86

STM32\_LSICLK, 85  
STM32\_NO\_INIT, 86  
STM32\_PLS, 86  
STM32\_PLS\_LEV0, 85  
STM32\_PLS\_LEV1, 85  
STM32\_PLS\_LEV2, 85  
STM32\_PLS\_LEV3, 85  
STM32\_PLS\_LEV4, 86  
STM32\_PLS\_LEV5, 86  
STM32\_PLS\_LEV6, 86  
STM32\_PLS\_LEV7, 86  
STM32\_PLS\_MASK, 85  
STM32\_PVD\_ENABLE, 86  
US2RTT, 83  
  
hal.c, 488  
hal.h, 488  
HAL\_IMPLEMENTS\_COUNTERS  
    HAL Driver, 85  
hal\_lld.c, 489  
hal\_lld.h, 490  
hal\_lld\_f100.h, 491  
hal\_lld\_f103.h, 498  
hal\_lld\_f105\_f107.h, 507  
hal\_lld\_get\_counter\_frequency  
    HAL Driver, 87  
hal\_lld\_get\_counter\_value  
    HAL Driver, 86  
hal\_lld\_init  
    HAL Driver, 81  
HAL\_USE\_ADC  
    Configuration, 13  
HAL\_USE\_CAN  
    Configuration, 13  
HAL\_USE\_EXT  
    Configuration, 13  
HAL\_USE\_GPT  
    Configuration, 13  
HAL\_USE\_I2C  
    Configuration, 13  
HAL\_USE\_ICU  
    Configuration, 14  
HAL\_USE\_MAC  
    Configuration, 14  
HAL\_USE\_MMC\_SPI  
    Configuration, 14  
HAL\_USE\_PAL  
    Configuration, 13  
HAL\_USE\_PWM  
    Configuration, 14  
HAL\_USE\_RTC  
    Configuration, 14  
HAL\_USE\_SDC  
    Configuration, 14  
HAL\_USE\_SERIAL  
    Configuration, 14  
HAL\_USE\_SERIAL\_USB  
    Configuration, 14  
HAL\_USE\_SPI  
    Configuration, 14  
  
HAL\_USE\_UART  
    Configuration, 14  
HAL\_USE\_USB  
    Configuration, 14  
halclock\_t  
    HAL Driver, 87  
halconf.h, 513  
halGetCounterFrequency  
    HAL Driver, 83  
halGetCounterValue  
    HAL Driver, 83  
hallinit  
    HAL Driver, 80  
hallsCounterWithin  
    HAL Driver, 84  
halPolledDelay  
    HAL Driver, 85  
halrtcnt\_t  
    HAL Driver, 87  
hscfg  
    MMCDriver, 430  
  
i2c  
    I2CDriver, 423  
I2C Driver, 87  
    CH\_IRQ\_HANDLER, 95, 96  
    I2C\_ACTIVE\_RX, 104  
    I2C\_ACTIVE\_TX, 104  
    I2C\_READY, 104  
    I2C\_STOP, 104  
    I2C\_UNINIT, 104  
    I2C\_CLK\_FREQ, 100  
    i2c\_lld\_get\_errors, 103  
    i2c\_lld\_init, 96  
    i2c\_lld\_master\_receive\_timeout, 97  
    i2c\_lld\_master\_transmit\_timeout, 98  
    i2c\_lld\_start, 96  
    i2c\_lld\_stop, 97  
    I2C\_USE\_MUTUAL\_EXCLUSION, 99  
    i2cAcquireBus, 95  
    i2caddr\_t, 103  
    I2CD1, 99  
    I2CD2, 99  
    I2CD3, 99  
    I2CD\_ACK\_FAILURE, 99  
    I2CD\_ARBITRATION\_LOST, 99  
    I2CD\_BUS\_ERROR, 99  
    I2CD\_NO\_ERROR, 99  
    I2CD\_OVERRUN, 99  
    I2CD\_PEC\_ERROR, 99  
    I2CD\_SMB\_ALERT, 99  
    I2CD\_TIMEOUT, 99  
    I2CDriver, 103  
    i2cdutycycle\_t, 104  
    i2cflags\_t, 103  
    i2cGetErrors, 93  
    i2cInit, 91  
    i2cMasterReceive, 100  
    i2cMasterReceiveTimeout, 94

i2cMasterTransmit, 100  
i2cMasterTransmitTimeout, 93  
i2cObjectInit, 92  
i2copmode\_t, 104  
i2cReleaseBus, 95  
i2cStart, 92  
i2cstate\_t, 104  
i2cStop, 92  
STM32\_DMA\_REQUIRED, 103  
STM32\_I2C\_DMA\_ERROR\_HOOK, 102  
STM32\_I2C\_I2C1\_DMA\_PRIORITY, 101  
STM32\_I2C\_I2C1\_IRQ\_PRIORITY, 101  
STM32\_I2C\_I2C1\_RX\_DMA\_STREAM, 102  
STM32\_I2C\_I2C1\_TX\_DMA\_STREAM, 102  
STM32\_I2C\_I2C2\_DMA\_PRIORITY, 101  
STM32\_I2C\_I2C2\_IRQ\_PRIORITY, 101  
STM32\_I2C\_I2C2\_RX\_DMA\_STREAM, 102  
STM32\_I2C\_I2C2\_TX\_DMA\_STREAM, 102  
STM32\_I2C\_I2C3\_DMA\_PRIORITY, 102  
STM32\_I2C\_I2C3\_IRQ\_PRIORITY, 101  
STM32\_I2C\_I2C3\_RX\_DMA\_STREAM, 103  
STM32\_I2C\_I2C3\_TX\_DMA\_STREAM, 103  
STM32\_I2C\_USE\_I2C1, 100  
STM32\_I2C\_USE\_I2C2, 101  
STM32\_I2C\_USE\_I2C3, 101  
wakeup\_isr, 100  
i2c.c, 515  
i2c.h, 516  
I2C1\_ER\_IRQHandler  
    STM32F100 HAL Support, 301  
    STM32F103 HAL Support, 324  
    STM32F105/F107 HAL Support, 344  
I2C1\_EV\_IRQHandler  
    STM32F100 HAL Support, 301  
    STM32F103 HAL Support, 323  
    STM32F105/F107 HAL Support, 343  
I2C2\_ER\_IRQHandler  
    STM32F100 HAL Support, 301  
    STM32F103 HAL Support, 324  
    STM32F105/F107 HAL Support, 344  
I2C2\_EV\_IRQHandler  
    STM32F100 HAL Support, 301  
    STM32F103 HAL Support, 324  
    STM32F105/F107 HAL Support, 344  
I2C\_ACTIVE\_RX  
    I2C Driver, 104  
I2C\_ACTIVE\_TX  
    I2C Driver, 104  
I2C\_READY  
    I2C Driver, 104  
I2C\_STOP  
    I2C Driver, 104  
I2C\_UNINIT  
    I2C Driver, 104  
I2C\_CLK\_FREQ  
    I2C Driver, 100  
i2c\_lld.c, 517  
i2c\_lld.h, 518  
i2c\_lld\_get\_errors  
    I2C Driver, 103  
i2c\_lld\_init  
    I2C Driver, 96  
i2c\_lld\_master\_receive\_timeout  
    I2C Driver, 97  
i2c\_lld\_master\_transmit\_timeout  
    I2C Driver, 98  
i2c\_lld\_start  
    I2C Driver, 96  
i2c\_lld\_stop  
    I2C Driver, 97  
I2C\_USE\_MUTUAL\_EXCLUSION  
    Configuration, 15  
    I2C Driver, 99  
i2cAcquireBus  
    I2C Driver, 95  
i2caddr\_t  
    I2C Driver, 103  
I2CConfig, 420  
    clock\_speed, 420  
    duty\_cycle, 421  
    op\_mode, 420  
I2CD1  
    I2C Driver, 99  
I2CD2  
    I2C Driver, 99  
I2CD3  
    I2C Driver, 99  
I2CD\_ACK\_FAILURE  
    I2C Driver, 99  
I2CD\_ARBITRATION\_LOST  
    I2C Driver, 99  
I2CD\_BUS\_ERROR  
    I2C Driver, 99  
I2CD\_NO\_ERROR  
    I2C Driver, 99  
I2CD\_OVERRUN  
    I2C Driver, 99  
I2CD\_PEC\_ERROR  
    I2C Driver, 99  
I2CD\_SMB\_ALERT  
    I2C Driver, 99  
I2CD\_TIMEOUT  
    I2C Driver, 99  
I2CDriver, 421  
    addr, 422  
    config, 422  
    dmemode, 422  
    dmarx, 423  
    dmatx, 423  
    errors, 422  
    i2c, 423  
    I2C Driver, 103  
    mutex, 422  
    state, 422  
    thread, 422  
i2cdutycycle\_t  
    I2C Driver, 104  
i2cflags\_t

I2C Driver, 103  
i2cGetErrors  
    I2C Driver, 93  
i2cInit  
    I2C Driver, 91  
i2cMasterReceive  
    I2C Driver, 100  
i2cMasterReceiveTimeout  
    I2C Driver, 94  
i2cMasterTransmit  
    I2C Driver, 100  
i2cMasterTransmitTimeout  
    I2C Driver, 93  
i2cObjectInit  
    I2C Driver, 92  
i2copcode\_t  
    I2C Driver, 104  
i2cReleaseBus  
    I2C Driver, 95  
i2cStart  
    I2C Driver, 92  
i2cstate\_t  
    I2C Driver, 104  
i2cStop  
    I2C Driver, 92  
ICU Driver, 104  
    \_icu\_isr\_invoke\_period\_cb, 114  
    \_icu\_isr\_invoke\_width\_cb, 113  
    ICU\_ACTIVE, 117  
    ICU\_IDLE, 117  
    ICU\_INPUT\_ACTIVE\_HIGH, 117  
    ICU\_INPUT\_ACTIVE\_LOW, 117  
    ICU\_READY, 117  
    ICU\_STOP, 117  
    ICU\_UNINIT, 117  
    ICU\_WAITING, 117  
    icu\_lld\_disable, 111  
    icu\_lld\_enable, 111  
    icu\_lld\_get\_period, 116  
    icu\_lld\_get\_width, 115  
    icu\_lld\_init, 110  
    icu\_lld\_start, 110  
    icu\_lld\_stop, 111  
    icucallback\_t, 116  
    icucnt\_t, 116  
    ICUD1, 111  
    ICUD2, 111  
    ICUD3, 112  
    ICUD4, 112  
    ICUD5, 112  
    ICUD8, 112  
    icuDisable, 110  
    icuDisableI, 112  
    ICUDriver, 116  
    icuEnable, 109  
    icuEnableI, 112  
    icufreq\_t, 116  
    icuGetPeriodI, 113  
    icuGetWidthI, 113  
    icuInit, 108  
    icumode\_t, 117  
    icuObjectInit, 108  
    icuStart, 108  
    icustate\_t, 117  
    icuStop, 109  
        STM32\_ICU\_TIM1\_IRQ\_PRIORITY, 115  
        STM32\_ICU\_TIM2\_IRQ\_PRIORITY, 115  
        STM32\_ICU\_TIM3\_IRQ\_PRIORITY, 115  
        STM32\_ICU\_TIM4\_IRQ\_PRIORITY, 115  
        STM32\_ICU\_TIM5\_IRQ\_PRIORITY, 115  
        STM32\_ICU\_TIM8\_IRQ\_PRIORITY, 115  
        STM32\_ICU\_USE\_TIM1, 114  
        STM32\_ICU\_USE\_TIM2, 114  
        STM32\_ICU\_USE\_TIM3, 114  
        STM32\_ICU\_USE\_TIM4, 114  
        STM32\_ICU\_USE\_TIM5, 115  
        STM32\_ICU\_USE\_TIM8, 115  
icu.c, 520  
icu.h, 520  
ICU\_ACTIVE  
    ICU Driver, 117  
ICU\_IDLE  
    ICU Driver, 117  
ICU\_INPUT\_ACTIVE\_HIGH  
    ICU Driver, 117  
ICU\_INPUT\_ACTIVE\_LOW  
    ICU Driver, 117  
ICU\_READY  
    ICU Driver, 117  
ICU\_STOP  
    ICU Driver, 117  
ICU\_UNINIT  
    ICU Driver, 117  
ICU\_WAITING  
    ICU Driver, 117  
icu\_lld.c, 521  
icu\_lld.h, 522  
icu\_lld\_disable  
    ICU Driver, 111  
icu\_lld\_enable  
    ICU Driver, 111  
icu\_lld\_get\_period  
    ICU Driver, 116  
icu\_lld\_get\_width  
    ICU Driver, 115  
icu\_lld\_init  
    ICU Driver, 110  
icu\_lld\_start  
    ICU Driver, 110  
icu\_lld\_stop  
    ICU Driver, 111  
icucallback\_t  
    ICU Driver, 116  
icucnt\_t  
    ICU Driver, 116  
ICUConfig, 423  
    frequency, 425  
    mode, 425

period\_cb, 425  
width\_cb, 425  
ICUD1  
    ICU Driver, 111  
ICUD2  
    ICU Driver, 111  
ICUD3  
    ICU Driver, 112  
ICUD4  
    ICU Driver, 112  
ICUD5  
    ICU Driver, 112  
ICUD8  
    ICU Driver, 112  
icuDisable  
    ICU Driver, 110  
icuDisable1  
    ICU Driver, 112  
ICUDriver, 425  
    clock, 427  
    config, 427  
    ICU Driver, 116  
    state, 427  
    tim, 427  
icuEnable  
    ICU Driver, 109  
icuEnable1  
    ICU Driver, 112  
icufreq\_t  
    ICU Driver, 116  
icuGetPeriod1  
    ICU Driver, 113  
icuGetWidth1  
    ICU Driver, 113  
iculInit  
    ICU Driver, 108  
icumode\_t  
    ICU Driver, 117  
icuObjectInit  
    ICU Driver, 108  
icuStart  
    ICU Driver, 108  
icustate\_t  
    ICU Driver, 117  
icuStop  
    ICU Driver, 109  
IDE  
    CANRxFrame, 411  
    CANTxFrame, 412  
ifcr  
    stm32\_dma\_stream\_t, 453  
in  
    USB Driver, 274  
    usb\_lld.c, 570  
in\_cb  
    USBEndpointConfig, 469  
in\_maxsize  
    USBEndpointConfig, 469  
in\_state

    USBEndpointConfig, 470  
    MMCDriver, 430  
    IOBus, 427  
        mask, 428  
        offset, 428  
        portid, 428  
    IOBUS\_DECL  
        PAL Driver, 135  
    iomode\_t  
        PAL Driver, 145  
    IOPORT1  
        PAL Driver, 141  
    IOPORT2  
        PAL Driver, 141  
    IOPORT3  
        PAL Driver, 141  
    IOPORT4  
        PAL Driver, 141  
    IOPORT5  
        PAL Driver, 141  
    IOPORT6  
        PAL Driver, 141  
    IOPORT7  
        PAL Driver, 141  
    ioportid\_t  
        PAL Driver, 145  
    ioportmask\_t  
        PAL Driver, 144  
    is\_inserted  
        MMCDriver, 430  
    is\_protected  
        MMCDriver, 430  
    ishift  
        stm32\_dma\_stream\_t, 453  
    last  
        TimeMeasurement, 456  
    lscfg  
        MMCDriver, 430  
    MAC\_USE\_EVENTS  
        Configuration, 15  
    mask  
        IOBus, 428  
    mcr  
        CANConfig, 406  
    MMC over SPI Driver, 117  
        MMC\_INSERTED, 127  
        MMC\_READING, 127  
        MMC\_READY, 127  
        MMC\_STOP, 127  
        MMC\_UNINIT, 127  
        MMC\_WAIT, 127  
        MMC\_WRITING, 127  
        MMC\_NICE\_WAITING, 126  
        MMC\_POLLING\_DELAY, 126  
        MMC\_POLLING\_INTERVAL, 126  
        MMC\_SECTOR\_SIZE, 126

mmcConnect, 120  
mmcDisconnect, 121  
mmcGetDriverState, 126  
mmcInit, 119  
mmclsWriteProtected, 126  
mmcObjectInit, 119  
mmcquery\_t, 127  
mmcSequentialRead, 122  
mmcSequentialWrite, 124  
mmcStart, 120  
mmcStartSequentialRead, 122  
mmcStartSequentialWrite, 124  
mmcstate\_t, 127  
mmcStop, 120  
mmcStopSequentialRead, 123  
mmcStopSequentialWrite, 125  
**MMC\_INSERTED**  
    MMC over SPI Driver, 127  
**MMC\_READING**  
    MMC over SPI Driver, 127  
**MMC\_READY**  
    MMC over SPI Driver, 127  
**MMC\_STOP**  
    MMC over SPI Driver, 127  
**MMC\_UNINIT**  
    MMC over SPI Driver, 127  
**MMC\_WAIT**  
    MMC over SPI Driver, 127  
**MMC\_WRITING**  
    MMC over SPI Driver, 127  
**MMC\_NICE\_WAITING**  
    Configuration, 15  
    MMC over SPI Driver, 126  
**MMC\_POLLING\_DELAY**  
    Configuration, 15  
    MMC over SPI Driver, 126  
**MMC\_POLLING\_INTERVAL**  
    Configuration, 15  
    MMC over SPI Driver, 126  
**MMC\_SECTOR\_SIZE**  
    Configuration, 15  
    MMC over SPI Driver, 126  
mmc\_spi.c, 523  
mmc\_spi.h, 524  
**MMC\_USE\_SPI\_POLLING**  
    Configuration, 15  
**MMCConfig**, 428  
mmcConnect  
    MMC over SPI Driver, 120  
mmcDisconnect  
    MMC over SPI Driver, 121  
**MMCDriver**, 428  
    cnt, 431  
    config, 430  
    hscfg, 430  
    inserted\_event, 430  
    is\_inserted, 430  
    is\_protected, 430  
    lscfg, 430  
    removed\_event, 431  
    spip, 430  
    state, 430  
    vt, 431  
mmcGetDriverState  
    MMC over SPI Driver, 126  
mmcInit  
    MMC over SPI Driver, 119  
mmclsWriteProtected  
    MMC over SPI Driver, 126  
mmcObjectInit  
    MMC over SPI Driver, 119  
mmcquery\_t  
    MMC over SPI Driver, 127  
mmcSequentialRead  
    MMC over SPI Driver, 122  
mmcSequentialWrite  
    MMC over SPI Driver, 124  
mmcStart  
    MMC over SPI Driver, 120  
mmcStartSequentialRead  
    MMC over SPI Driver, 122  
mmcStartSequentialWrite  
    MMC over SPI Driver, 124  
mmcstate\_t  
    MMC over SPI Driver, 127  
mmcStop  
    MMC over SPI Driver, 120  
mmcStopSequentialRead  
    MMC over SPI Driver, 123  
mmcStopSequentialWrite  
    MMC over SPI Driver, 125  
mode  
    CANFilter, 409  
    EXTChannelConfig, 413  
    ICUConfig, 425  
    PWMChannelConfig, 435  
**MS2RTT**  
    HAL Driver, 82  
mutex  
    ADCDriver, 404  
    I2CDriver, 422  
    SPIDriver, 452  
num  
    CANConfig, 406  
num\_channels  
    ADCCConversionGroup, 401  
odr  
    stm32\_gpio\_setup\_t, 454  
offset  
    IOBus, 428  
op\_mode  
    I2CConfig, 420  
**OTG\_FS\_IRQHandler**  
    STM32F105/F107 HAL Support, 346  
**OTG\_FS\_WKUP\_IRQHandler**  
    STM32F105/F107 HAL Support, 344

out  
  USB Driver, 274  
  usb\_lld.c, 570

out\_cb  
  USBEndpointConfig, 469

out\_maxsize  
  USBEndpointConfig, 469

out\_state  
  USBEndpointConfig, 470

PAData  
  PALConfig, 433

PAL Driver, 127  
  \_IOBUS\_DATA, 135  
  \_pal\_lld\_init, 133  
  \_pal\_lld\_setgroupmode, 133

IOBUS\_DECL, 135

iomode\_t, 145

IOPORT1, 141

IOPORT2, 141

IOPORT3, 141

IOPORT4, 141

IOPORT5, 141

IOPORT6, 141

IOPORT7, 141

ioportid\_t, 145

ioportmask\_t, 144

PAL\_GROUP\_MASK, 134

PAL\_HIGH, 134

PAL\_IOPORTS\_WIDTH, 141

pal\_lld\_clearport, 143

pal\_lld\_init, 141

pal\_lld\_readlatch, 142

pal\_lld\_readport, 142

pal\_lld\_setgroupmode, 144

pal\_lld\_setport, 143

pal\_lld\_writegroup, 143

pal\_lld\_writepad, 144

pal\_lld\_writeport, 142

PAL\_LOW, 134

PAL\_MODE\_INPUT, 133

PAL\_MODE\_INPUT\_ANALOG, 134

PAL\_MODE\_INPUT\_PULLDOWN, 134

PAL\_MODE\_INPUT\_PULLUP, 133

PAL\_MODE\_OUTPUT\_OPENDRAIN, 134

PAL\_MODE\_OUTPUT\_PUSH\_PULL, 134

PAL\_MODE\_RESET, 133

PAL\_MODE\_STM32\_ALTERNATE\_OPENDRAIN, 141

PAL\_MODE\_STM32\_ALTERNATE\_PUSH\_PULL, 141

PAL\_MODE\_UNCONNECTED, 133

PAL\_PORT\_BIT, 134

PAL\_WHOLE\_PORT, 141

palClearPad, 139

palClearPort, 137

pallInit, 135

palReadBus, 132

palReadGroup, 137

palReadLatch, 136

palReadPad, 138

palReadPort, 135

palSetBusMode, 132

palSetGroupMode, 138

palSetPad, 139

palSetPadMode, 140

palSetPort, 136

palTogglePad, 140

palTogglePort, 137

palWriteBus, 132

palWriteGroup, 138

palWritePad, 139

palWritePort, 136

pal.c, 525

pal.h, 526

PAL\_GROUP\_MASK  
  PAL Driver, 134

PAL\_HIGH  
  PAL Driver, 134

PAL\_IOPORTS\_WIDTH  
  PAL Driver, 141

pal\_lld.c, 527

pal\_lld.h, 528

pal\_lld\_clearport  
  PAL Driver, 143

pal\_lld\_init  
  PAL Driver, 141

pal\_lld\_readlatch  
  PAL Driver, 142

pal\_lld\_readport  
  PAL Driver, 142

pal\_lld\_setgroupmode  
  PAL Driver, 144

pal\_lld\_setport  
  PAL Driver, 143

pal\_lld\_writegroup  
  PAL Driver, 143

pal\_lld\_writepad  
  PAL Driver, 144

pal\_lld\_writeport  
  PAL Driver, 142

PAL\_LOW  
  PAL Driver, 134

PAL\_MODE\_INPUT  
  PAL Driver, 133

PAL\_MODE\_INPUT\_ANALOG  
  PAL Driver, 134

PAL\_MODE\_INPUT\_PULLDOWN  
  PAL Driver, 134

PAL\_MODE\_OUTPUT\_OPENDRAIN  
  PAL Driver, 134

PAL\_MODE\_OUTPUT\_PUSH\_PULL  
  PAL Driver, 134

PAL\_MODE\_RESET  
  PAL Driver, 133

PAL\_MODE\_STM32\_ALTERNATE\_OPENDRAIN  
  PAL Driver, 141

PAL\_MODE\_STM32\_ALTERNATE\_PUSH\_PULL  
  PAL Driver, 138

PAL Driver, 141  
PAL\_MODE\_UNCONNECTED  
    PAL Driver, 133  
PAL\_PORT\_BIT  
    PAL Driver, 134  
PAL\_WHOLE\_PORT  
    PAL Driver, 141  
palClearPad  
    PAL Driver, 139  
palClearPort  
    PAL Driver, 137  
PALConfig, 431  
    PAData, 433  
    PBData, 433  
    PCData, 433  
    PDData, 433  
    PEData, 433  
    PFData, 433  
    PGData, 433  
palInit  
    PAL Driver, 135  
palReadBus  
    PAL Driver, 132  
palReadGroup  
    PAL Driver, 137  
palReadLatch  
    PAL Driver, 136  
palReadPad  
    PAL Driver, 138  
palReadPort  
    PAL Driver, 135  
palSetBusMode  
    PAL Driver, 132  
palSetGroupMode  
    PAL Driver, 138  
palSetPad  
    PAL Driver, 139  
palSetPadMode  
    PAL Driver, 140  
palSetPort  
    PAL Driver, 136  
palTogglePad  
    PAL Driver, 140  
palTogglePort  
    PAL Driver, 137  
palWriteBus  
    PAL Driver, 132  
palWriteGroup  
    PAL Driver, 138  
palWritePad  
    PAL Driver, 139  
palWritePort  
    PAL Driver, 136  
param  
    USBDriver, 466  
PBData  
    PALConfig, 433  
PCData  
    PALConfig, 433  
PDData  
    PALConfig, 433  
PEData  
    PALConfig, 433  
period  
    PWMConfig, 437  
    PWMDriver, 440  
period\_cb  
    ICUConfig, 425  
PFData  
    PALConfig, 433  
PGData  
    PALConfig, 433  
pmnext  
    USBDriver, 467  
portid  
    IOBus, 428  
PVD\_IRQHandler  
    STM32F100 HAL Support, 299  
    STM32F103 HAL Support, 321  
    STM32F105/F107 HAL Support, 341  
PWM Driver, 145  
    PWM\_READY, 161  
    PWM\_STOP, 161  
    PWM\_UNINIT, 161  
    PWM\_CHANNELS, 158  
    PWM\_COMPLEMENTARY\_OUTPUT\_ACTIVE\_HIGH,  
        158  
    PWM\_COMPLEMENTARY\_OUTPUT\_ACTIVE\_LOW,  
        158  
    PWM\_COMPLEMENTARY\_OUTPUT\_DISABLED, 158  
    PWM\_COMPLEMENTARY\_OUTPUT\_MASK, 158  
    PWM\_DEGREES\_TO\_WIDTH, 156  
    PWM\_FRACTION\_TO\_WIDTH, 155  
    pwm\_lld\_change\_period, 160  
    pwm\_lld\_disable\_channel, 153  
    pwm\_lld\_enable\_channel, 153  
    pwm\_lld\_init, 152  
    pwm\_lld\_start, 152  
    pwm\_lld\_stop, 153  
    PWM\_OUTPUT\_ACTIVE\_HIGH, 155  
    PWM\_OUTPUT\_ACTIVE\_LOW, 155  
    PWM\_OUTPUT\_DISABLED, 155  
    PWM\_OUTPUT\_MASK, 155  
    PWM\_PERCENTAGE\_TO\_WIDTH, 156  
    pwmcallback\_t, 161  
    pwmChangePeriod, 150  
    pwmChangePeriodl, 156  
    pwmchannel\_t, 161  
    pwmcnt\_t, 161  
    PWMD1, 154  
    PWMD2, 154  
    PWMD3, 154  
    PWMD4, 154  
    PWMD5, 154  
    PWMD8, 155  
    pwmDisableChannel, 152  
    pwmDisableChannell, 157  
    PWMDriver, 161

pwmEnableChannel, 151  
pwmEnableChannell, 157  
pwmInit, 149  
pwmmode\_t, 161  
 pwmObjectInit, 149  
 pwmStart, 150  
 pwmstate\_t, 161  
 pwmStop, 150  
 STM32\_PWM\_TIM1\_IRQ\_PRIORITY, 160  
 STM32\_PWM\_TIM2\_IRQ\_PRIORITY, 160  
 STM32\_PWM\_TIM3\_IRQ\_PRIORITY, 160  
 STM32\_PWM\_TIM4\_IRQ\_PRIORITY, 160  
 STM32\_PWM\_TIM5\_IRQ\_PRIORITY, 160  
 STM32\_PWM\_TIM8\_IRQ\_PRIORITY, 160  
 STM32\_PWM\_USE\_ADVANCED, 159  
 STM32\_PWM\_USE\_TIM1, 159  
 STM32\_PWM\_USE\_TIM2, 159  
 STM32\_PWM\_USE\_TIM3, 159  
 STM32\_PWM\_USE\_TIM4, 159  
 STM32\_PWM\_USE\_TIM5, 159  
 STM32\_PWM\_USE\_TIM8, 160  
pwm.c, 529  
pwm.h, 530  
PWM\_READY  
    PWM Driver, 161  
PWM\_STOP  
    PWM Driver, 161  
PWM\_UNINIT  
    PWM Driver, 161  
PWM\_CHANNELS  
    PWM Driver, 158  
PWM\_COMPLEMENTARY\_OUTPUT\_ACTIVE\_HIGH  
    PWM Driver, 158  
PWM\_COMPLEMENTARY\_OUTPUT\_ACTIVE\_LOW  
    PWM Driver, 158  
PWM\_COMPLEMENTARY\_OUTPUT\_DISABLED  
    PWM Driver, 158  
PWM\_COMPLEMENTARY\_OUTPUT\_MASK  
    PWM Driver, 158  
PWM\_DEGREES\_TO\_WIDTH  
    PWM Driver, 156  
PWM\_FRACTION\_TO\_WIDTH  
    PWM Driver, 155  
pwm\_lld.c, 531  
pwm\_lld.h, 532  
pwm\_lld\_change\_period  
    PWM Driver, 160  
pwm\_lld\_disable\_channel  
    PWM Driver, 153  
pwm\_lld\_enable\_channel  
    PWM Driver, 153  
pwm\_lld\_init  
    PWM Driver, 152  
pwm\_lld\_start  
    PWM Driver, 152  
pwm\_lld\_stop  
    PWM Driver, 153  
PWM\_OUTPUT\_ACTIVE\_HIGH  
    PWM Driver, 155  
PWM\_OUTPUT\_ACTIVE\_LOW  
    PWM Driver, 155  
PWM\_OUTPUT\_DISABLED  
    PWM Driver, 155  
PWM\_OUTPUT\_MASK  
    PWM Driver, 155  
PWM\_PERCENTAGE\_TO\_WIDTH  
    PWM Driver, 156  
pwmcallback\_t  
    PWM Driver, 161  
pwmChangePeriod  
    PWM Driver, 150  
pwmChangePeriodl  
    PWM Driver, 156  
pwmchannel\_t  
    PWM Driver, 161  
PWMChannelConfig, 433  
    callback, 435  
    mode, 435  
pwmcnt\_t  
    PWM Driver, 161  
PWMConfig, 435  
    bdtr, 437  
    callback, 437  
    channels, 437  
    cr2, 437  
    frequency, 437  
    period, 437  
PWMD1  
    PWM Driver, 154  
PWMD2  
    PWM Driver, 154  
PWMD3  
    PWM Driver, 154  
PWMD4  
    PWM Driver, 154  
PWMD5  
    PWM Driver, 154  
PWMD8  
    PWM Driver, 155  
pwmDisableChannel  
    PWM Driver, 152  
pwmDisableChannell  
    PWM Driver, 157  
PWMDriver, 438  
    clock, 440  
    config, 440  
    period, 440  
    PWM Driver, 161  
    state, 440  
    tim, 440  
pwmEnableChannel  
    PWM Driver, 151  
pwmEnableChannell  
    PWM Driver, 157  
pwmInit  
    PWM Driver, 149  
pwmmode\_t  
    PWM Driver, 161

pwmObjectInit  
    PWM Driver, 149

pwmStart  
    PWM Driver, 150

pwmstate\_t  
    PWM Driver, 161

pwmStop  
    PWM Driver, 150

rca  
    SDCDriver, 444

RCC\_IRQHandler  
    STM32F100 HAL Support, 299  
    STM32F103 HAL Support, 321  
    STM32F105/F107 HAL Support, 341

rccDisableADC1  
    STM32F1xx RCC Support, 381

rccDisableAHB  
    STM32F1xx RCC Support, 380

rccDisableAPB1  
    STM32F1xx RCC Support, 378

rccDisableAPB2  
    STM32F1xx RCC Support, 379

rccDisableBKPIInterface  
    STM32F1xx RCC Support, 382

rccDisableCAN1  
    STM32F1xx RCC Support, 383

rccDisableDMA1  
    STM32F1xx RCC Support, 384

rccDisableDMA2  
    STM32F1xx RCC Support, 384

rccDisableETH  
    STM32F1xx RCC Support, 385

rccDisableI2C1  
    STM32F1xx RCC Support, 386

rccDisableI2C2  
    STM32F1xx RCC Support, 387

rccDisablePWRInterface  
    STM32F1xx RCC Support, 382

rccDisableSDIO  
    STM32F1xx RCC Support, 387

rccDisableSPI1  
    STM32F1xx RCC Support, 388

rccDisableSPI2  
    STM32F1xx RCC Support, 389

rccDisableSPI3  
    STM32F1xx RCC Support, 390

rccDisableTIM1  
    STM32F1xx RCC Support, 390

rccDisableTIM2  
    STM32F1xx RCC Support, 391

rccDisableTIM3  
    STM32F1xx RCC Support, 392

rccDisableTIM4  
    STM32F1xx RCC Support, 392

rccDisableTIM5  
    STM32F1xx RCC Support, 393

rccDisableTIM8  
    STM32F1xx RCC Support, 394

rccDisableUART4  
    STM32F1xx RCC Support, 397

rccDisableUART5  
    STM32F1xx RCC Support, 397

rccDisableUSART1  
    STM32F1xx RCC Support, 394

rccDisableUSART2  
    STM32F1xx RCC Support, 395

rccDisableUSART3  
    STM32F1xx RCC Support, 396

rccDisableUSB  
    STM32F1xx RCC Support, 398

rccEnableADC1  
    STM32F1xx RCC Support, 381

rccEnableAHB  
    STM32F1xx RCC Support, 379

rccEnableAPB1  
    STM32F1xx RCC Support, 377

rccEnableAPB2  
    STM32F1xx RCC Support, 378

rccEnableBKPIInterface  
    STM32F1xx RCC Support, 381

rccEnableCAN1  
    STM32F1xx RCC Support, 383

rccEnableDMA1  
    STM32F1xx RCC Support, 384

rccEnableDMA2  
    STM32F1xx RCC Support, 384

rccEnableETH  
    STM32F1xx RCC Support, 385

rccEnableI2C1  
    STM32F1xx RCC Support, 386

rccEnableI2C2  
    STM32F1xx RCC Support, 387

rccEnablePWRInterface  
    STM32F1xx RCC Support, 382

rccEnableSDIO  
    STM32F1xx RCC Support, 387

rccEnableSPI1  
    STM32F1xx RCC Support, 388

rccEnableSPI2  
    STM32F1xx RCC Support, 389

rccEnableSPI3  
    STM32F1xx RCC Support, 389

rccEnableTIM1  
    STM32F1xx RCC Support, 390

rccEnableTIM2  
    STM32F1xx RCC Support, 391

rccEnableTIM3  
    STM32F1xx RCC Support, 391

rccEnableTIM4  
    STM32F1xx RCC Support, 392

rccEnableTIM5  
    STM32F1xx RCC Support, 393

rccEnableTIM8  
    STM32F1xx RCC Support, 393

rccEnableUART4  
    STM32F1xx RCC Support, 396

rccEnableUART5  
    STM32F1xx RCC Support, 396

STM32F1xx RCC Support, 397  
rccEnableUSART1  
    STM32F1xx RCC Support, 394  
rccEnableUSART2  
    STM32F1xx RCC Support, 395  
rccEnableUSART3  
    STM32F1xx RCC Support, 396  
rccEnableUSB  
    STM32F1xx RCC Support, 398  
rccResetADC1  
    STM32F1xx RCC Support, 381  
rccResetAHB  
    STM32F1xx RCC Support, 380  
rccResetAPB1  
    STM32F1xx RCC Support, 378  
rccResetAPB2  
    STM32F1xx RCC Support, 379  
rccResetBKP  
    STM32F1xx RCC Support, 382  
rccResetBKPIInterface  
    STM32F1xx RCC Support, 382  
rccResetCAN1  
    STM32F1xx RCC Support, 383  
rccResetDMA1  
    STM32F1xx RCC Support, 384  
rccResetDMA2  
    STM32F1xx RCC Support, 385  
rccResetETH  
    STM32F1xx RCC Support, 386  
rccResetI2C1  
    STM32F1xx RCC Support, 386  
rccResetI2C2  
    STM32F1xx RCC Support, 387  
rccResetPWRInterface  
    STM32F1xx RCC Support, 383  
rccResetSDIO  
    STM32F1xx RCC Support, 388  
rccResetSPI1  
    STM32F1xx RCC Support, 388  
rccResetSPI2  
    STM32F1xx RCC Support, 389  
rccResetSPI3  
    STM32F1xx RCC Support, 390  
rccResetTIM1  
    STM32F1xx RCC Support, 391  
rccResetTIM2  
    STM32F1xx RCC Support, 391  
rccResetTIM3  
    STM32F1xx RCC Support, 392  
rccResetTIM4  
    STM32F1xx RCC Support, 393  
rccResetTIM5  
    STM32F1xx RCC Support, 393  
rccResetTIM8  
    STM32F1xx RCC Support, 394  
rccResetUART4  
    STM32F1xx RCC Support, 397  
rccResetUART5  
    STM32F1xx RCC Support, 398  
rccResetUSART1  
    STM32F1xx RCC Support, 395  
rccResetUSART2  
    STM32F1xx RCC Support, 395  
rccResetUSART3  
    STM32F1xx RCC Support, 396  
rccResetUSB  
    STM32F1xx RCC Support, 398  
receiving  
    USBDriver, 466  
register1  
    CANFilter, 410  
register2  
    CANFilter, 410  
removed\_event  
    MMCDriver, 431  
requests\_hook\_cb  
    USBCConfig, 464  
RTC Driver, 162  
    CH\_IRQ\_HANDLER, 165  
    RTC\_EVENT\_ALARM, 170  
    RTC\_EVENT\_OVERFLOW, 170  
    RTC\_ALARMS, 169  
    rtc\_lld\_acquire, 169  
    rtc\_lld\_apb1\_sync, 168  
    rtc\_lld\_get\_alarm, 166  
    rtc\_lld\_get\_time, 166  
    rtc\_lld\_init, 165  
    rtc\_lld\_release, 169  
    rtc\_lld\_set\_alarm, 166  
    rtc\_lld\_set\_callback, 167  
    rtc\_lld\_set\_time, 166  
    rtc\_lld\_wait\_write, 169  
    RTC\_SUPPORTS\_CALLBACKS, 169  
    RTCAlarm, 169  
    rtcalarm\_t, 170  
    RTCCallbackConfig, 170  
    rtccb\_t, 170  
    RTCD1, 167  
    RTCDriver, 169  
    rtcevent\_t, 170  
    rtcGetAlarm, 165  
    rtcGetAlarml, 168  
    rtcGetTime, 164  
    rtcGetTimel, 167  
    rtchnit, 164  
    rtcSetAlarm, 164  
    rtcSetAlarml, 168  
    rtcSetCallback, 165  
    rtcSetCallbackl, 168  
    rtcSetTime, 164  
    rtcSetTimel, 167  
    RTCTime, 169  
    rtc.c, 533  
    rtc.h, 534  
    RTC\_EVENT\_ALARM  
        RTC Driver, 170  
    RTC\_EVENT\_OVERFLOW  
        RTC Driver, 170

RTC\_Alarm\_IRQHandler  
    STM32F100 HAL Support, 302  
    STM32F103 HAL Support, 324  
    STM32F105/F107 HAL Support, 344

RTC\_ALARMS  
    RTC Driver, 169

RTC\_IRQHandler  
    STM32F100 HAL Support, 299  
    STM32F103 HAL Support, 321  
    STM32F105/F107 HAL Support, 341

rtc\_lld.c, 534

rtc\_lld.h, 535

rtc\_lld\_acquire  
    RTC Driver, 169

rtc\_lld\_apb1\_sync  
    RTC Driver, 168

rtc\_lld\_get\_alarm  
    RTC Driver, 166

rtc\_lld\_get\_time  
    RTC Driver, 166

rtc\_lld\_init  
    RTC Driver, 165

rtc\_lld\_release  
    RTC Driver, 169

rtc\_lld\_set\_alarm  
    RTC Driver, 166

rtc\_lld\_set\_callback  
    RTC Driver, 167

rtc\_lld\_set\_time  
    RTC Driver, 166

rtc\_lld\_wait\_write  
    RTC Driver, 169

RTC\_SUPPORTS\_CALLBACKS  
    RTC Driver, 169

RTCAlarm, 440  
    RTC Driver, 169  
    tv\_sec, 440

rtcalarm\_t  
    RTC Driver, 170

RTCCallbackConfig, 441  
    callback, 441  
    RTC Driver, 170

rtccb\_t  
    RTC Driver, 170

RTCD1  
    RTC Driver, 167

RTCDriver, 441  
    callback, 442  
    RTC Driver, 169

rtcevent\_t  
    RTC Driver, 170

rtcGetAlarm  
    RTC Driver, 165

rtcGetAlarms  
    RTC Driver, 168

rtcGetTime  
    RTC Driver, 164

rtcGetTimel  
    RTC Driver, 167

rtcInit  
    RTC Driver, 164

rtcSetAlarm  
    RTC Driver, 164

rtcSetAlarms  
    RTC Driver, 168

rtcSetCallback  
    RTC Driver, 165

rtcSetCallback1  
    RTC Driver, 168

rtcSetTime  
    RTC Driver, 164

rtcSetTimel  
    RTC Driver, 167

RTCTime, 442  
    RTC Driver, 169  
    tv\_msec, 442  
    tv\_sec, 442

RTR  
    CANRxFrame, 411  
    CANTxFrame, 411

RXADDR0  
    stm32\_usb\_descriptor\_t, 455

rxbuf  
    UARTDriver, 461  
    USBOutEndpointState, 471

rxchar\_cb  
    UARTConfig, 458

rxcnt  
    USBOutEndpointState, 471

RXCOUNT0  
    stm32\_usb\_descriptor\_t, 455

RXCOUNT1  
    stm32\_usb\_descriptor\_t, 455

rxdmamode  
    SPIDriver, 452

rxend\_cb  
    UARTConfig, 458

rxerr\_cb  
    UARTConfig, 458

rxfull\_event  
    CANDriver, 408

rxpkts  
    USBOutEndpointState, 471

rxsem  
    CANDriver, 408

rysize  
    USBOutEndpointState, 471

rxstate  
    UARTDriver, 461

S2RTT  
    HAL Driver, 82

samples  
    ADCDriver, 404

sc\_cr1  
    SerialConfig, 445

sc\_cr2  
    SerialConfig, 445

sc\_cr3  
    SerialConfig, 445  
sc\_speed  
    SerialConfig, 445  
scale  
    CANFilter, 410  
SD1  
    Serial Driver, 195  
SD2  
    Serial Driver, 195  
SD3  
    Serial Driver, 196  
SD4  
    Serial Driver, 196  
SD5  
    Serial Driver, 196  
SD6  
    Serial Driver, 196  
SD\_READY  
    Serial Driver, 203  
SD\_STOP  
    Serial Driver, 203  
SD\_UNINIT  
    Serial Driver, 203  
SD\_BREAK\_DETECTED  
    Serial Driver, 196  
SD\_FRAMING\_ERROR  
    Serial Driver, 196  
sd\_lld\_init  
    Serial Driver, 194  
sd\_lld\_start  
    Serial Driver, 195  
sd\_lld\_stop  
    Serial Driver, 195  
SD\_NOISE\_ERROR  
    Serial Driver, 196  
SD\_OVERRUN\_ERROR  
    Serial Driver, 196  
SD\_PARITY\_ERROR  
    Serial Driver, 196  
sdAsynchronousRead  
    Serial Driver, 200  
sdAsynchronousWrite  
    Serial Driver, 199  
SDC Driver, 170  
    \_sdc\_wait\_for\_transfer\_state, 178  
    CH\_IRQ\_HANDLER, 179  
    SDC\_ACTIVE, 187  
    SDC\_CONNECTING, 187  
    SDC\_DISCONNECTING, 187  
    SDC\_READING, 187  
    SDC\_READY, 187  
    SDC\_STOP, 187  
    SDC\_UNINIT, 187  
    SDC\_WRITING, 187  
    SDC\_BLOCK\_SIZE, 183  
    SDC\_CMD8\_PATTERN, 183  
    SDC\_INIT\_RETRY, 184  
    sdc\_lld\_init, 179  
        sdc\_lld\_read, 182  
        sdc\_lld\_send\_cmd\_long\_crc, 182  
        sdc\_lld\_send\_cmd\_none, 181  
        sdc\_lld\_send\_cmd\_short, 181  
        sdc\_lld\_send\_cmd\_short\_crc, 182  
        sdc\_lld\_set\_bus\_mode, 181  
        sdc\_lld\_set\_data\_clk, 180  
        sdc\_lld\_start, 179  
        sdc\_lld\_start\_clk, 180  
        sdc\_lld\_stop, 180  
        sdc\_lld\_stop\_clk, 181  
        sdc\_lld\_write, 183  
    SDC\_MODE\_MMC\_SUPPORT, 184  
    SDC\_MODE\_CARDTYPE\_MASK, 184  
    SDC\_MODE\_CARDTYPE\_MMC, 184  
    SDC\_MODE\_CARDTYPE\_SDV11, 184  
    SDC\_MODE\_CARDTYPE\_SDV20, 184  
    SDC\_MODE\_HIGH\_CAPACITY, 184  
    SDC\_NICE\_WAITING, 184  
    SDC\_R1\_ERROR, 184  
    SDC\_R1\_ERROR\_MASK, 184  
    SDC\_R1\_IS\_CARD\_LOCKED, 185  
    SDC\_R1\_STS, 185  
    sdcbusmode\_t, 187  
    sdcConnect, 175  
    SDCD1, 183  
    sdcDisconnect, 176  
    SDCDriver, 186  
    sdcGetDriverState, 185  
    sdclInit, 174  
    sdclsCardInserted, 185  
    sdclsWriteProtected, 186  
    sdemode\_t, 186  
    sdcObjectInit, 174  
    sdcRead, 177  
    sdcStart, 174  
    sdystate\_t, 187  
    sdcStop, 175  
    sdcWrite, 177  
    STM32\_SDC\_DATATIMEOUT, 186  
    STM32\_SDC\_SDIO\_DMA\_PRIORITY, 186  
    STM32\_SDC\_SDIO\_IRQ\_PRIORITY, 186  
    STM32\_SDC\_UNALIGNED\_SUPPORT, 186  
    sdc.c, 536  
    sdc.h, 537  
    SDC\_ACTIVE  
        SDC Driver, 187  
    SDC\_CONNECTING  
        SDC Driver, 187  
    SDC\_DISCONNECTING  
        SDC Driver, 187  
    SDC\_READING  
        SDC Driver, 187  
    SDC\_READY  
        SDC Driver, 187  
    SDC\_STOP  
        SDC Driver, 187  
    SDC\_UNINIT  
        SDC Driver, 187

SDC\_WRITING  
    SDC Driver, 187  
SDC\_BLOCK\_SIZE  
    SDC Driver, 183  
SDC\_CMD8\_PATTERN  
    SDC Driver, 183  
SDC\_INIT\_RETRY  
    Configuration, 15  
    SDC Driver, 184  
sdc\_lld.c, 539  
sdc\_lld.h, 539  
sdc\_lld\_init  
    SDC Driver, 179  
sdc\_lld\_read  
    SDC Driver, 182  
sdc\_lld\_send\_cmd\_long\_crc  
    SDC Driver, 182  
sdc\_lld\_send\_cmd\_none  
    SDC Driver, 181  
sdc\_lld\_send\_cmd\_short  
    SDC Driver, 181  
sdc\_lld\_send\_cmd\_short\_crc  
    SDC Driver, 182  
sdc\_lld\_set\_bus\_mode  
    SDC Driver, 181  
sdc\_lld\_set\_data\_clk  
    SDC Driver, 180  
sdc\_lld\_start  
    SDC Driver, 179  
sdc\_lld\_start\_clk  
    SDC Driver, 180  
sdc\_lld\_stop  
    SDC Driver, 180  
sdc\_lld\_write  
    SDC Driver, 183  
SDC\_MMC\_SUPPORT  
    Configuration, 16  
    SDC Driver, 184  
SDC\_MODE\_CARDTYPE\_MASK  
    SDC Driver, 184  
SDC\_MODE\_CARDTYPE\_MMC  
    SDC Driver, 184  
SDC\_MODE\_CARDTYPE\_SDV11  
    SDC Driver, 184  
SDC\_MODE\_CARDTYPE\_SDV20  
    SDC Driver, 184  
SDC\_MODE\_HIGH\_CAPACITY  
    SDC Driver, 184  
SDC\_NICE\_WAITING  
    Configuration, 16  
    SDC Driver, 184  
SDC\_R1\_ERROR  
    SDC Driver, 184  
SDC\_R1\_ERROR\_MASK  
    SDC Driver, 184  
SDC\_R1\_IS\_CARD\_LOCKED  
    SDC Driver, 185  
SDC\_R1\_STS  
    SDC Driver, 185  
sdcbusmode\_t  
    SDC Driver, 187  
SDCConfig, 443  
sdcConnect  
    SDC Driver, 175  
SDCD1  
    SDC Driver, 183  
sdcDisconnect  
    SDC Driver, 176  
SDCDriver, 443  
    cardmode, 444  
    cid, 444  
    config, 444  
    csd, 444  
    rca, 444  
    SDC Driver, 186  
    state, 444  
    thread, 444  
sdcGetDriverState  
    SDC Driver, 185  
sdclInit  
    SDC Driver, 174  
sdclsCardInserted  
    SDC Driver, 185  
sdclsWriteProtected  
    SDC Driver, 186  
sdemode\_t  
    SDC Driver, 186  
sdcObjectInit  
    SDC Driver, 174  
sdcRead  
    SDC Driver, 177  
sdcStart  
    SDC Driver, 174  
sdcstate\_t  
    SDC Driver, 187  
sdcStop  
    SDC Driver, 175  
sdcWrite  
    SDC Driver, 177  
sdGet  
    Serial Driver, 198  
sdGetTimeout  
    Serial Driver, 198  
sdGetWouldBlock  
    Serial Driver, 197  
sdIncomingData  
    Serial Driver, 193  
sdInit  
    Serial Driver, 191  
SDIO\_IRQHandler  
    STM32F103 HAL Support, 325  
sdObjectInit  
    Serial Driver, 191  
sdPut  
    Serial Driver, 197  
sdPutTimeout

Serial Driver, 198  
sdPutWouldBlock  
    Serial Driver, 197  
sdRead  
    Serial Driver, 199  
sdReadTimeout  
    Serial Driver, 200  
sdRequestData  
    Serial Driver, 193  
sdStart  
    Serial Driver, 192  
sdstate\_t  
    Serial Driver, 203  
sdStop  
    Serial Driver, 192  
SDU\_READY  
    Serial over USB Driver, 208  
SDU\_STOP  
    Serial over USB Driver, 208  
SDU\_UNINIT  
    Serial over USB Driver, 208  
sduDataReceived  
    Serial over USB Driver, 207  
sduDataTransmitted  
    Serial over USB Driver, 206  
sduInit  
    Serial over USB Driver, 205  
sduInterruptTransmitted  
    Serial over USB Driver, 207  
sduObjectInit  
    Serial over USB Driver, 205  
sduRequestsHook  
    Serial over USB Driver, 206  
sduStart  
    Serial over USB Driver, 205  
sdustate\_t  
    Serial over USB Driver, 208  
sduStop  
    Serial over USB Driver, 205  
sdWrite  
    Serial Driver, 199  
sdWriteTimeout  
    Serial Driver, 199  
selfindex  
    stm32\_dma\_stream\_t, 453  
Serial Driver, 187  
    \_serial\_driver\_data, 202  
    \_serial\_driver\_methods, 197  
CH\_IRQ\_HANDLER, 194  
SD1, 195  
SD2, 195  
SD3, 196  
SD4, 196  
SD5, 196  
SD6, 196  
SD\_READY, 203  
SD\_STOP, 203  
SD\_UNINIT, 203  
SD\_BREAK\_DETECTED, 196  
SD\_FRAMING\_ERROR, 196  
sd\_lld\_init, 194  
sd\_lld\_start, 195  
sd\_lld\_stop, 195  
SD\_NOISE\_ERROR, 196  
SD\_OVERRUN\_ERROR, 196  
SD\_PARITY\_ERROR, 196  
sdAsynchronousRead, 200  
sdAsynchronousWrite, 199  
sdGet, 198  
sdGetTimeout, 198  
sdGetWouldBlock, 197  
sdIncomingData, 193  
sdInit, 191  
sdObjectInit, 191  
sdPut, 197  
sdPutTimeout, 198  
sdPutWouldBlock, 197  
sdRead, 199  
sdReadTimeout, 200  
sdRequestData, 193  
sdStart, 192  
sdstate\_t, 203  
sdStop, 192  
sdWrite, 199  
sdWriteTimeout, 199  
SERIAL\_BUFFERS\_SIZE, 196  
SERIAL\_DEFAULT\_BITRATE, 196  
SerialDriver, 203  
STM32\_SERIAL\_UART4\_PRIORITY, 202  
STM32\_SERIAL\_UART5\_PRIORITY, 202  
STM32\_SERIAL\_USART1\_PRIORITY, 201  
STM32\_SERIAL\_USART2\_PRIORITY, 201  
STM32\_SERIAL\_USART3\_PRIORITY, 201  
STM32\_SERIAL\_USART6\_PRIORITY, 202  
STM32\_SERIAL\_USE\_UART4, 201  
STM32\_SERIAL\_USE\_UART5, 201  
STM32\_SERIAL\_USE\_USART1, 200  
STM32\_SERIAL\_USE\_USART2, 200  
STM32\_SERIAL\_USE\_USART3, 201  
STM32\_SERIAL\_USE\_USART6, 201  
USART\_CR2\_STOP0P5\_BITS, 202  
USART\_CR2\_STOP1\_BITS, 202  
USART\_CR2\_STOP1P5\_BITS, 202  
USART\_CR2\_STOP2\_BITS, 202  
Serial over USB Driver, 203  
    \_serial\_usb\_driver\_data, 208  
    \_serial\_usb\_driver\_methods, 208  
SDU\_READY, 208  
SDU\_STOP, 208  
SDU\_UNINIT, 208  
sduDataReceived, 207  
sduDataTransmitted, 206  
sduInit, 205  
sduInterruptTransmitted, 207  
sduObjectInit, 205  
sduRequestsHook, 206  
sduStart, 205  
sdustate\_t, 208

sduStop, 205  
SERIAL\_USB\_BUFFERS\_SIZE, 207  
SerialUSBDriver, 208  
serial.c, 541  
serial.h, 541  
SERIAL\_BUFFERS\_SIZE  
    Configuration, 16  
    Serial Driver, 196  
SERIAL\_DEFAULT\_BITRATE  
    Configuration, 16  
    Serial Driver, 196  
serial\_lld.c, 543  
serial\_lld.h, 544  
serial\_usb.c, 545  
serial\_usb.h, 546  
SERIAL\_USB\_BUFFERS\_SIZE  
    Configuration, 16  
    Serial over USB Driver, 207  
SerialConfig, 444  
    sc\_cr1, 445  
    sc\_cr2, 445  
    sc\_cr3, 445  
    sc\_speed, 445  
SerialDriver, 445  
    Serial Driver, 203  
    vmt, 446  
SerialDriverVMT, 446  
SerialUSBConfig, 446  
    usb\_config, 448  
    usbp, 448  
SerialUSBDriver, 448  
    Serial over USB Driver, 208  
    vmt, 448  
SerialUSBDriverVMT, 449  
setup  
    USBDriver, 467  
setup\_cb  
    USBEndpointConfig, 469  
SID  
    CANRxFrame, 411  
    CANTxFrame, 412  
sleep\_event  
    CANDriver, 409  
smpr1  
    ADCConversionGroup, 401  
smpr2  
    ADCConversionGroup, 401  
sof\_cb  
    USBConfig, 464  
speed  
    UARTConfig, 458  
spi  
    SPIDriver, 452  
SPI Driver, 208  
    \_spi\_isr\_code, 226  
    \_spi\_wait\_s, 225  
    \_spi\_wakeup\_isr, 226  
    SPI\_ACTIVE, 230  
    SPI\_COMPLETE, 230  
        SPI\_READY, 230  
        SPI\_STOP, 230  
        SPI\_UNINIT, 230  
        spi\_lld\_exchange, 220  
        spi\_lld\_ignore, 220  
        spi\_lld\_init, 218  
        spi\_lld\_polled\_exchange, 221  
        spi\_lld\_receive, 221  
        spi\_lld\_select, 219  
        spi\_lld\_send, 221  
        spi\_lld\_start, 218  
        spi\_lld\_stop, 219  
        spi\_lld\_unselect, 220  
        SPI\_USE\_MUTUAL\_EXCLUSION, 222  
        SPI\_USE\_WAIT, 222  
        spiAcquireBus, 218  
        spicallback\_t, 229  
        SPID1, 222  
        SPID2, 222  
        SPID3, 222  
        SPIDriver, 229  
        spiExchange, 216  
        spilgnore, 216  
        spilInit, 212  
        spiObjectInit, 212  
        spiPolledExchange, 225  
        spiReceive, 217  
        spiReleaseBus, 218  
        spiSelect, 214  
        spiSelectl, 222  
        spiSend, 217  
        spiStart, 213  
        spiStartExchange, 215  
        spiStartExchangel, 223  
        spiStartIgnore, 214  
        spiStartIgnrel, 223  
        spiStartReceive, 215  
        spiStartReceivel, 225  
        spiStartSend, 215  
        spiStartSendl, 224  
        spistate\_t, 230  
        spiStop, 213  
        spiUnselect, 214  
        spiUnselectl, 223  
        STM32\_SPI\_DMA\_ERROR\_HOOK, 228  
        STM32\_SPI\_SPI1\_DMA\_PRIORITY, 228  
        STM32\_SPI\_SPI1\_IRQ\_PRIORITY, 227  
        STM32\_SPI\_SPI1\_RX\_DMA\_STREAM, 228  
        STM32\_SPI\_SPI1\_TX\_DMA\_STREAM, 228  
        STM32\_SPI\_SPI2\_DMA\_PRIORITY, 228  
        STM32\_SPI\_SPI2\_IRQ\_PRIORITY, 228  
        STM32\_SPI\_SPI2\_RX\_DMA\_STREAM, 229  
        STM32\_SPI\_SPI2\_TX\_DMA\_STREAM, 229  
        STM32\_SPI\_SPI3\_DMA\_PRIORITY, 228  
        STM32\_SPI\_SPI3\_IRQ\_PRIORITY, 228  
        STM32\_SPI\_SPI3\_RX\_DMA\_STREAM, 229  
        STM32\_SPI\_SPI3\_TX\_DMA\_STREAM, 229  
        STM32\_SPI\_USE\_SPI1, 227  
        STM32\_SPI\_USE\_SPI2, 227

STM32\_SPI\_USE\_SPI3, 227  
spi.c, 547  
spi.h, 548  
SPI1\_IRQHandler  
    STM32F100 HAL Support, 301  
    STM32F103 HAL Support, 324  
    STM32F105/F107 HAL Support, 344  
SPI2\_IRQHandler  
    STM32F100 HAL Support, 301  
    STM32F103 HAL Support, 324  
    STM32F105/F107 HAL Support, 344  
SPI3\_IRQHandler  
    STM32F103 HAL Support, 325  
    STM32F105/F107 HAL Support, 345  
SPI\_ACTIVE  
    SPI Driver, 230  
SPI\_COMPLETE  
    SPI Driver, 230  
SPI\_READY  
    SPI Driver, 230  
SPI\_STOP  
    SPI Driver, 230  
SPI\_UNINIT  
    SPI Driver, 230  
spi\_lld.c, 549  
spi\_lld.h, 550  
spi\_lld\_exchange  
    SPI Driver, 220  
spi\_lld\_ignore  
    SPI Driver, 220  
spi\_lld\_init  
    SPI Driver, 218  
spi\_lld\_polled\_exchange  
    SPI Driver, 221  
spi\_lld\_receive  
    SPI Driver, 221  
spi\_lld\_select  
    SPI Driver, 219  
spi\_lld\_send  
    SPI Driver, 221  
spi\_lld\_start  
    SPI Driver, 218  
spi\_lld\_stop  
    SPI Driver, 219  
spi\_lld\_unselect  
    SPI Driver, 220  
SPI\_USE\_MUTUAL\_EXCLUSION  
    Configuration, 16  
    SPI Driver, 222  
SPI\_USE\_WAIT  
    Configuration, 16  
    SPI Driver, 222  
spiAcquireBus  
    SPI Driver, 218  
spicallback\_t  
    SPI Driver, 229  
SPIConfig, 449  
    cr1, 450  
    end\_cb, 450  
            sspad, 450  
            ssport, 450  
SPID1  
    SPI Driver, 222  
SPID2  
    SPI Driver, 222  
SPID3  
    SPI Driver, 222  
SPIDriver, 450  
    config, 452  
    dmarx, 452  
    dmatx, 452  
    mutex, 452  
    rxdmamode, 452  
    spi, 452  
    SPI Driver, 229  
    state, 452  
    thread, 452  
    txdmamode, 452  
spiExchange  
    SPI Driver, 216  
spilgnore  
    SPI Driver, 216  
spilnit  
    SPI Driver, 212  
spiObjectInit  
    SPI Driver, 212  
spip  
    MMCDriver, 430  
spiPolledExchange  
    SPI Driver, 225  
spiReceive  
    SPI Driver, 217  
spiReleaseBus  
    SPI Driver, 218  
spiSelect  
    SPI Driver, 214  
spiSelectl  
    SPI Driver, 222  
spiSend  
    SPI Driver, 217  
spiStart  
    SPI Driver, 213  
spiStartExchange  
    SPI Driver, 215  
spiStartExchangel  
    SPI Driver, 223  
spiStartIgnore  
    SPI Driver, 214  
spiStartIgnorel  
    SPI Driver, 223  
spiStartReceive  
    SPI Driver, 215  
spiStartReceive1  
    SPI Driver, 225  
spiStartSend  
    SPI Driver, 215  
spiStartSend1  
    SPI Driver, 224

spistate\_t  
    SPI Driver, 230

spiStop  
    SPI Driver, 213

spiUnselect  
    SPI Driver, 214

spiUnselect!  
    SPI Driver, 223

sqr1  
    ADCConversionGroup, 402

sqr2  
    ADCConversionGroup, 402

sqr3  
    ADCConversionGroup, 402

sspad  
    SPIConfig, 450

ssport  
    SPIConfig, 450

start  
    TimeMeasurement, 456

state  
    ADCDriver, 404  
    CANDriver, 408  
    EXTDriver, 416  
    GPTDriver, 420  
    I2CDriver, 422  
    ICUDriver, 427  
    MMCDriver, 430  
    PWMDriver, 440  
    SDCDriver, 444  
    SPIDriver, 452  
    UARTDriver, 461  
    USBDriver, 466

status  
    CANDriver, 409  
    USBDriver, 467

stm32.h, 551

STM32\_ACTIVATE\_PLL  
    STM32F100 HAL Support, 304  
    STM32F103 HAL Support, 327

STM32\_ACTIVATE\_PLL1  
    STM32F105/F107 HAL Support, 348

STM32\_ACTIVATE\_PLL2  
    STM32F105/F107 HAL Support, 348

STM32\_ACTIVATE\_PLL3  
    STM32F105/F107 HAL Support, 348

STM32\_ADC\_ADC1\_DMA\_PRIORITY  
    ADC Driver, 36

STM32\_ADC\_ADC1\_IRQ\_PRIORITY  
    ADC Driver, 36

STM32\_ADC\_USE\_ADC1  
    ADC Driver, 35

STM32\_ADCCLK  
    STM32F100 HAL Support, 304  
    STM32F103 HAL Support, 328  
    STM32F105/F107 HAL Support, 350

STM32\_ADCCLK\_MAX  
    STM32F100 HAL Support, 295  
    STM32F103 HAL Support, 317

STM32\_ADCPRE  
    STM32F100 HAL Support, 303  
    STM32F103 HAL Support, 327  
    STM32F105/F107 HAL Support, 348

STM32\_ADCPRE\_DIV2  
    STM32F100 HAL Support, 297  
    STM32F103 HAL Support, 319  
    STM32F105/F107 HAL Support, 339

STM32\_ADCPRE\_DIV4  
    STM32F100 HAL Support, 297  
    STM32F103 HAL Support, 319  
    STM32F105/F107 HAL Support, 339

STM32\_ADCPRE\_DIV6  
    STM32F100 HAL Support, 297  
    STM32F103 HAL Support, 319  
    STM32F105/F107 HAL Support, 339

STM32\_ADCPRE\_DIV8  
    STM32F100 HAL Support, 298  
    STM32F103 HAL Support, 319  
    STM32F105/F107 HAL Support, 339

STM32\_CAN\_CAN1\_IRQ\_PRIORITY  
    CAN Driver, 53

STM32\_CAN\_USE\_CAN1  
    CAN Driver, 53

stm32\_clock\_init  
    HAL Driver, 82

stm32\_dma.c, 552

stm32\_dma.h, 553

STM32\_DMA1\_STREAMS\_MASK  
    STM32F1xx DMA Support, 366

STM32\_DMA2\_STREAMS\_MASK  
    STM32F1xx DMA Support, 366

STM32\_DMA\_CCR\_RESET\_VALUE  
    STM32F1xx DMA Support, 366

STM32\_DMA\_CR\_CHSEL  
    STM32F1xx DMA Support, 367

STM32\_DMA\_CR\_CHSEL\_MASK  
    STM32F1xx DMA Support, 367

STM32\_DMA\_CR\_DMEIE  
    STM32F1xx DMA Support, 367

STM32\_DMA\_GETCHANNEL  
    STM32F1xx DMA Support, 366

STM32\_DMA\_IS\_VALID\_ID  
    STM32F1xx DMA Support, 367

STM32\_DMA\_ISR\_MASK  
    STM32F1xx DMA Support, 366

STM32\_DMA\_REQUIRED  
    I2C Driver, 103

STM32\_DMA\_STREAM  
    STM32F1xx DMA Support, 367

STM32\_DMA\_STREAM\_ID  
    STM32F1xx DMA Support, 367

STM32\_DMA\_STREAM\_ID\_MSK  
    STM32F1xx DMA Support, 366

stm32\_dma\_stream\_t, 453  
    channel, 453  
    ifcr, 453  
    ishift, 453



STM32F103 HAL Support, [316](#)  
STM32F105/F107 HAL Support, [336](#)

STM32\_HSI\_ENABLED  
    HAL Driver, [86](#)

STM32\_HSICLK  
    HAL Driver, [85](#)

STM32\_I2C\_DMA\_ERROR\_HOOK  
    I2C Driver, [102](#)

STM32\_I2C\_I2C1\_DMA\_PRIORITY  
    I2C Driver, [101](#)

STM32\_I2C\_I2C1\_IRQ\_PRIORITY  
    I2C Driver, [101](#)

STM32\_I2C\_I2C1\_RX\_DMA\_STREAM  
    I2C Driver, [102](#)

STM32\_I2C\_I2C1\_TX\_DMA\_STREAM  
    I2C Driver, [102](#)

STM32\_I2C\_I2C2\_DMA\_PRIORITY  
    I2C Driver, [101](#)

STM32\_I2C\_I2C2\_IRQ\_PRIORITY  
    I2C Driver, [101](#)

STM32\_I2C\_I2C2\_RX\_DMA\_STREAM  
    I2C Driver, [102](#)

STM32\_I2C\_I2C2\_TX\_DMA\_STREAM  
    I2C Driver, [102](#)

STM32\_I2C\_I2C3\_DMA\_PRIORITY  
    I2C Driver, [102](#)

STM32\_I2C\_I2C3\_IRQ\_PRIORITY  
    I2C Driver, [101](#)

STM32\_I2C\_I2C3\_RX\_DMA\_STREAM  
    I2C Driver, [103](#)

STM32\_I2C\_I2C3\_TX\_DMA\_STREAM  
    I2C Driver, [103](#)

STM32\_I2C\_USE\_I2C1  
    I2C Driver, [100](#)

STM32\_I2C\_USE\_I2C2  
    I2C Driver, [101](#)

STM32\_I2C\_USE\_I2C3  
    I2C Driver, [101](#)

STM32\_I2S\_CLOCK\_REQUIRED  
    STM32F105/F107 HAL Support, [348](#)

STM32\_ICU\_TIM1\_IRQ\_PRIORITY  
    ICU Driver, [115](#)

STM32\_ICU\_TIM2\_IRQ\_PRIORITY  
    ICU Driver, [115](#)

STM32\_ICU\_TIM3\_IRQ\_PRIORITY  
    ICU Driver, [115](#)

STM32\_ICU\_TIM4\_IRQ\_PRIORITY  
    ICU Driver, [115](#)

STM32\_ICU\_TIM5\_IRQ\_PRIORITY  
    ICU Driver, [115](#)

STM32\_ICU\_TIM8\_IRQ\_PRIORITY  
    ICU Driver, [115](#)

STM32\_ICU\_USE\_TIM1  
    ICU Driver, [114](#)

STM32\_ICU\_USE\_TIM2  
    ICU Driver, [114](#)

STM32\_ICU\_USE\_TIM3  
    ICU Driver, [114](#)

STM32\_ICU\_USE\_TIM4  
    ICU Driver, [114](#)

ICU Driver, [114](#)

STM32\_ICU\_USE\_TIM5  
    ICU Driver, [115](#)

STM32\_ICU\_USE\_TIM8  
    ICU Driver, [115](#)

STM32\_LSE\_ENABLED  
    HAL Driver, [86](#)

STM32\_LSECLK\_MAX  
    STM32F100 HAL Support, [295](#)  
    STM32F103 HAL Support, [316](#)  
    STM32F105/F107 HAL Support, [336](#)

STM32\_LSECLK\_MIN  
    STM32F100 HAL Support, [295](#)  
    STM32F103 HAL Support, [317](#)  
    STM32F105/F107 HAL Support, [336](#)

STM32\_LSI\_ENABLED  
    HAL Driver, [86](#)

STM32\_LSICLK  
    HAL Driver, [85](#)

STM32\_MCOSEL  
    STM32F100 HAL Support, [303](#)  
    STM32F103 HAL Support, [327](#)  
    STM32F105/F107 HAL Support, [348](#)

STM32\_MCOSEL\_HSE  
    STM32F100 HAL Support, [298](#)  
    STM32F103 HAL Support, [320](#)  
    STM32F105/F107 HAL Support, [340](#)

STM32\_MCOSEL\_HSI  
    STM32F100 HAL Support, [298](#)  
    STM32F103 HAL Support, [320](#)  
    STM32F105/F107 HAL Support, [340](#)

STM32\_MCOSEL\_NO\_CLOCK  
    STM32F100 HAL Support, [298](#)  
    STM32F103 HAL Support, [320](#)  
    STM32F105/F107 HAL Support, [340](#)

STM32\_MCOSEL\_PLL2  
    STM32F105/F107 HAL Support, [340](#)

STM32\_MCOSEL\_PLL3  
    STM32F105/F107 HAL Support, [340](#)

STM32\_MCOSEL\_PLLDIV2  
    STM32F105/F107 HAL Support, [340](#)

STM32\_MCOSEL\_PLLDIV2  
    STM32F100 HAL Support, [298](#)  
    STM32F103 HAL Support, [320](#)  
    STM32F105/F107 HAL Support, [340](#)

STM32\_MCOSEL\_SYSCLK  
    STM32F100 HAL Support, [298](#)  
    STM32F103 HAL Support, [320](#)  
    STM32F105/F107 HAL Support, [340](#)

STM32\_MCOSEL\_XT1  
    STM32F105/F107 HAL Support, [340](#)

STM32\_NO\_INIT  
    HAL Driver, [86](#)

STM32\_OTG\_CLOCK\_REQUIRED  
    STM32F105/F107 HAL Support, [348](#)

STM32\_OTGFSCLK  
    STM32F105/F107 HAL Support, [350](#)

STM32\_OTGFSPRE  
    STM32F105/F107 HAL Support, [348](#)

STM32\_OTGFSPRE\_DIV2  
    STM32F105/F107 HAL Support, 339

STM32\_OTGFSPRE\_DIV3  
    STM32F105/F107 HAL Support, 339

STM32\_PCLK1  
    STM32F100 HAL Support, 304  
    STM32F103 HAL Support, 328  
    STM32F105/F107 HAL Support, 350

STM32\_PCLK1\_MAX  
    STM32F100 HAL Support, 295  
    STM32F103 HAL Support, 317  
    STM32F105/F107 HAL Support, 337

STM32\_PCLK2  
    STM32F100 HAL Support, 304  
    STM32F103 HAL Support, 328  
    STM32F105/F107 HAL Support, 350

STM32\_PCLK2\_MAX  
    STM32F100 HAL Support, 295  
    STM32F103 HAL Support, 317  
    STM32F105/F107 HAL Support, 337

STM32\_PLL1IN\_MAX  
    STM32F105/F107 HAL Support, 336

STM32\_PLL1IN\_MIN  
    STM32F105/F107 HAL Support, 336

STM32\_PLL1VCO\_MAX  
    STM32F105/F107 HAL Support, 336

STM32\_PLL1VCO\_MIN  
    STM32F105/F107 HAL Support, 336

STM32\_PLL23IN\_MAX  
    STM32F105/F107 HAL Support, 336

STM32\_PLL23IN\_MIN  
    STM32F105/F107 HAL Support, 336

STM32\_PLL23VCO\_MAX  
    STM32F105/F107 HAL Support, 337

STM32\_PLL23VCO\_MIN  
    STM32F105/F107 HAL Support, 337

STM32\_PLL2CLKIN  
    STM32F105/F107 HAL Support, 349

STM32\_PLL2CLKOUT  
    STM32F105/F107 HAL Support, 349

STM32\_PLL2MUL  
    STM32F105/F107 HAL Support, 349

STM32\_PLL2MUL\_VALUE  
    STM32F105/F107 HAL Support, 347

STM32\_PLL2VCO  
    STM32F105/F107 HAL Support, 349

STM32\_PLL3CLKIN  
    STM32F105/F107 HAL Support, 349

STM32\_PLL3CLKOUT  
    STM32F105/F107 HAL Support, 349

STM32\_PLL3MUL  
    STM32F105/F107 HAL Support, 349

STM32\_PLL3MUL\_VALUE  
    STM32F105/F107 HAL Support, 347

STM32\_PLL3VCO  
    STM32F105/F107 HAL Support, 349

STM32\_PLLCLKIN  
    STM32F100 HAL Support, 304  
    STM32F103 HAL Support, 328

STM32F105/F107 HAL Support, 349

STM32\_PLLCLKOUT  
    STM32F100 HAL Support, 304

STM32\_PLLIN\_MAX  
    STM32F100 HAL Support, 295  
    STM32F103 HAL Support, 317

STM32\_PLLIN\_MIN  
    STM32F100 HAL Support, 295  
    STM32F103 HAL Support, 317

STM32\_PLLMUL  
    STM32F100 HAL Support, 304  
    STM32F103 HAL Support, 327  
    STM32F105/F107 HAL Support, 348

STM32\_PLLMUL\_VALUE  
    STM32F100 HAL Support, 303  
    STM32F103 HAL Support, 326  
    STM32F105/F107 HAL Support, 347

STM32\_PLLOUT\_MAX  
    STM32F100 HAL Support, 295  
    STM32F103 HAL Support, 317

STM32\_PLLOUT\_MIN  
    STM32F100 HAL Support, 295  
    STM32F103 HAL Support, 317

STM32\_PLLSRC  
    STM32F100 HAL Support, 302  
    STM32F103 HAL Support, 326  
    STM32F105/F107 HAL Support, 346

STM32\_PLLSRC\_HSE  
    STM32F100 HAL Support, 298  
    STM32F103 HAL Support, 319

STM32\_PLLSRC\_HSI  
    STM32F100 HAL Support, 298  
    STM32F103 HAL Support, 319  
    STM32F105/F107 HAL Support, 339

STM32\_PLLSRC\_PREDIV1  
    STM32F105/F107 HAL Support, 339

STM32\_PLLVCO  
    STM32F105/F107 HAL Support, 349

STM32\_PLLXTPRE  
    STM32F100 HAL Support, 303  
    STM32F103 HAL Support, 326

STM32\_PLLXTPRE\_DIV1  
    STM32F100 HAL Support, 298  
    STM32F103 HAL Support, 320

STM32\_PLLXTPRE\_DIV2  
    STM32F100 HAL Support, 298  
    STM32F103 HAL Support, 320

STM32\_PLS  
    HAL Driver, 86

STM32\_PLS\_LEV0  
    HAL Driver, 85

STM32\_PLS\_LEV1  
    HAL Driver, 85

STM32\_PLS\_LEV2  
    HAL Driver, 85

STM32\_PLS\_LEV3  
    HAL Driver, 85

STM32\_PLS\_LEV4  
    HAL Driver, 86  
STM32\_PLS\_LEV5  
    HAL Driver, 86  
STM32\_PLS\_LEV6  
    HAL Driver, 86  
STM32\_PLS\_LEV7  
    HAL Driver, 86  
STM32\_PLS\_MASK  
    HAL Driver, 85  
STM32\_PPREG1  
    STM32F100 HAL Support, 303  
    STM32F103 HAL Support, 327  
    STM32F105/F107 HAL Support, 347  
STM32\_PPREG1\_DIV1  
    STM32F100 HAL Support, 296  
    STM32F103 HAL Support, 318  
    STM32F105/F107 HAL Support, 338  
STM32\_PPREG1\_DIV16  
    STM32F100 HAL Support, 297  
    STM32F103 HAL Support, 319  
    STM32F105/F107 HAL Support, 338  
STM32\_PPREG1\_DIV2  
    STM32F100 HAL Support, 297  
    STM32F103 HAL Support, 318  
    STM32F105/F107 HAL Support, 338  
STM32\_PPREG1\_DIV4  
    STM32F100 HAL Support, 297  
    STM32F103 HAL Support, 318  
    STM32F105/F107 HAL Support, 338  
STM32\_PPREG1\_DIV8  
    STM32F100 HAL Support, 297  
    STM32F103 HAL Support, 318  
    STM32F105/F107 HAL Support, 338  
STM32\_PPREG2  
    STM32F100 HAL Support, 303  
    STM32F103 HAL Support, 327  
    STM32F105/F107 HAL Support, 347  
STM32\_PPREG2\_DIV1  
    STM32F100 HAL Support, 297  
    STM32F103 HAL Support, 319  
    STM32F105/F107 HAL Support, 338  
STM32\_PPREG2\_DIV16  
    STM32F100 HAL Support, 297  
    STM32F103 HAL Support, 319  
    STM32F105/F107 HAL Support, 339  
STM32\_PPREG2\_DIV2  
    STM32F100 HAL Support, 297  
    STM32F103 HAL Support, 319  
    STM32F105/F107 HAL Support, 339  
STM32\_PPREG2\_DIV4  
    STM32F100 HAL Support, 297  
    STM32F103 HAL Support, 319  
    STM32F105/F107 HAL Support, 339  
STM32\_PPREG2\_DIV8  
    STM32F100 HAL Support, 297  
    STM32F103 HAL Support, 319  
    STM32F105/F107 HAL Support, 339  
STM32\_PREDIV1  
    STM32F105/F107 HAL Support, 348  
STM32\_PREDIV1\_VALUE  
    STM32F105/F107 HAL Support, 346  
STM32\_PREDIV1CLK  
    STM32F105/F107 HAL Support, 349  
STM32\_PREDIV1SRC  
    STM32F105/F107 HAL Support, 346  
STM32\_PREDIV1SRC\_HSE  
    STM32F105/F107 HAL Support, 341  
STM32\_PREDIV1SRC\_PLL2  
    STM32F105/F107 HAL Support, 341  
STM32\_PREDIV2  
    STM32F105/F107 HAL Support, 348  
STM32\_PREDIV2\_VALUE  
    STM32F105/F107 HAL Support, 347  
STM32\_PVD\_ENABLE  
    HAL Driver, 86  
STM32\_PWM\_TIM1\_IRQ\_PRIORITY  
    PWM Driver, 160  
STM32\_PWM\_TIM2\_IRQ\_PRIORITY  
    PWM Driver, 160  
STM32\_PWM\_TIM3\_IRQ\_PRIORITY  
    PWM Driver, 160  
STM32\_PWM\_TIM4\_IRQ\_PRIORITY  
    PWM Driver, 160  
STM32\_PWM\_TIM5\_IRQ\_PRIORITY  
    PWM Driver, 160  
STM32\_PWM\_TIM8\_IRQ\_PRIORITY  
    PWM Driver, 160  
STM32\_PWM\_USE\_ADVANCED  
    PWM Driver, 159  
STM32\_PWM\_USE\_TIM1  
    PWM Driver, 159  
STM32\_PWM\_USE\_TIM2  
    PWM Driver, 159  
STM32\_PWM\_USE\_TIM3  
    PWM Driver, 159  
STM32\_PWM\_USE\_TIM4  
    PWM Driver, 159  
STM32\_PWM\_USE\_TIM5  
    PWM Driver, 159  
STM32\_PWM\_USE\_TIM8  
    PWM Driver, 160  
stm32\_rcc.h, 556  
STM32\_RTCCLK  
    STM32F100 HAL Support, 304  
    STM32F103 HAL Support, 328  
    STM32F105/F107 HAL Support, 350  
STM32\_RTCSEL  
    STM32F100 HAL Support, 303  
    STM32F103 HAL Support, 327  
    STM32F105/F107 HAL Support, 348  
STM32\_RTCSEL\_HSEDIV  
    STM32F100 HAL Support, 299  
    STM32F103 HAL Support, 321  
    STM32F105/F107 HAL Support, 341  
STM32\_RTCSEL\_LSE  
    STM32F100 HAL Support, 299  
    STM32F103 HAL Support, 320

- STM32F105/F107 HAL Support, [340](#)
- STM32\_RTCSEL\_LSI
  - STM32F100 HAL Support, [299](#)
  - STM32F103 HAL Support, [321](#)
  - STM32F105/F107 HAL Support, [341](#)
- STM32\_RTCSEL\_MASK
  - STM32F100 HAL Support, [298](#)
  - STM32F103 HAL Support, [320](#)
  - STM32F105/F107 HAL Support, [340](#)
- STM32\_RTCSEL\_NOCLOCK
  - STM32F100 HAL Support, [298](#)
  - STM32F103 HAL Support, [320](#)
  - STM32F105/F107 HAL Support, [340](#)
- STM32\_SDC\_DATATIMEOUT
  - SDC Driver, [186](#)
- STM32\_SDC\_SDIO\_DMA\_PRIORITY
  - SDC Driver, [186](#)
- STM32\_SDC\_SDIO\_IRQ\_PRIORITY
  - SDC Driver, [186](#)
- STM32\_SDC\_UNALIGNED\_SUPPORT
  - SDC Driver, [186](#)
- STM32\_SERIAL\_UART4\_PRIORITY
  - Serial Driver, [202](#)
- STM32\_SERIAL\_UART5\_PRIORITY
  - Serial Driver, [202](#)
- STM32\_SERIAL\_USART1\_PRIORITY
  - Serial Driver, [201](#)
- STM32\_SERIAL\_USART2\_PRIORITY
  - Serial Driver, [201](#)
- STM32\_SERIAL\_USART3\_PRIORITY
  - Serial Driver, [201](#)
- STM32\_SERIAL\_USART6\_PRIORITY
  - Serial Driver, [202](#)
- STM32\_SERIAL\_USE\_UART4
  - Serial Driver, [201](#)
- STM32\_SERIAL\_USE\_UART5
  - Serial Driver, [201](#)
- STM32\_SERIAL\_USE\_USART1
  - Serial Driver, [200](#)
- STM32\_SERIAL\_USE\_USART2
  - Serial Driver, [200](#)
- STM32\_SERIAL\_USE\_USART3
  - Serial Driver, [201](#)
- STM32\_SERIAL\_USE\_USART6
  - Serial Driver, [201](#)
- STM32\_SPI\_DMA\_ERROR\_HOOK
  - SPI Driver, [228](#)
- STM32\_SPI\_SPI1\_DMA\_PRIORITY
  - SPI Driver, [228](#)
- STM32\_SPI\_SPI1\_IRQ\_PRIORITY
  - SPI Driver, [227](#)
- STM32\_SPI\_SPI1\_RX\_DMA\_STREAM
  - SPI Driver, [228](#)
- STM32\_SPI\_SPI1\_TX\_DMA\_STREAM
  - SPI Driver, [228](#)
- STM32\_SPI\_SPI2\_DMA\_PRIORITY
  - SPI Driver, [228](#)
- STM32\_SPI\_SPI2\_IRQ\_PRIORITY
  - SPI Driver, [228](#)
- STM32\_SPI\_SPI2\_RX\_DMA\_STREAM
  - SPI Driver, [229](#)
- STM32\_SPI\_SPI2\_TX\_DMA\_STREAM
  - SPI Driver, [229](#)
- STM32\_SPI\_SPI3\_DMA\_PRIORITY
  - SPI Driver, [228](#)
- STM32\_SPI\_SPI3\_IRQ\_PRIORITY
  - SPI Driver, [228](#)
- STM32\_SPI\_SPI3\_RX\_DMA\_STREAM
  - SPI Driver, [229](#)
- STM32\_SPI\_SPI3\_TX\_DMA\_STREAM
  - SPI Driver, [229](#)
- STM32\_SPI\_USE\_SPI1
  - SPI Driver, [227](#)
- STM32\_SPI\_USE\_SPI2
  - SPI Driver, [227](#)
- STM32\_SPI\_USE\_SPI3
  - SPI Driver, [227](#)
- STM32\_SPII2S\_MAX
  - STM32F105/F107 HAL Support, [337](#)
- STM32\_SW
  - STM32F100 HAL Support, [302](#)
  - STM32F103 HAL Support, [326](#)
  - STM32F105/F107 HAL Support, [346](#)
- STM32\_SW\_HSE
  - STM32F100 HAL Support, [296](#)
  - STM32F103 HAL Support, [317](#)
  - STM32F105/F107 HAL Support, [337](#)
- STM32\_SW\_HSI
  - STM32F100 HAL Support, [295](#)
  - STM32F103 HAL Support, [317](#)
  - STM32F105/F107 HAL Support, [337](#)
- STM32\_SW\_PLL
  - STM32F100 HAL Support, [296](#)
  - STM32F103 HAL Support, [317](#)
  - STM32F105/F107 HAL Support, [337](#)
- STM32\_SYSCLK
  - STM32F100 HAL Support, [304](#)
  - STM32F103 HAL Support, [328](#)
  - STM32F105/F107 HAL Support, [350](#)
- STM32\_SYSCLK\_MAX
  - STM32F100 HAL Support, [295](#)
  - STM32F103 HAL Support, [316](#)
  - STM32F105/F107 HAL Support, [336](#)
- stm32\_tim\_t, [454](#)
- STM32\_TIMCLK1
  - STM32F100 HAL Support, [304](#)
  - STM32F103 HAL Support, [328](#)
  - STM32F105/F107 HAL Support, [350](#)
- STM32\_TIMCLK2
  - STM32F100 HAL Support, [304](#)
  - STM32F103 HAL Support, [328](#)
  - STM32F105/F107 HAL Support, [350](#)
- STM32\_UART\_DMA\_ERROR\_HOOK
  - UART Driver, [248](#)
- STM32\_UART\_USART1\_DMA\_PRIORITY
  - UART Driver, [248](#)
- STM32\_UART\_USART1\_IRQ\_PRIORITY
  - UART Driver, [247](#)

STM32\_UART\_USART1\_RX\_DMA\_STREAM  
    UART Driver, 248  
STM32\_UART\_USART1\_TX\_DMA\_STREAM  
    UART Driver, 248  
STM32\_UART\_USART2\_DMA\_PRIORITY  
    UART Driver, 248  
STM32\_UART\_USART2\_IRQ\_PRIORITY  
    UART Driver, 247  
STM32\_UART\_USART2\_RX\_DMA\_STREAM  
    UART Driver, 248  
STM32\_UART\_USART2\_TX\_DMA\_STREAM  
    UART Driver, 249  
STM32\_UART\_USART3\_DMA\_PRIORITY  
    UART Driver, 248  
STM32\_UART\_USART3\_IRQ\_PRIORITY  
    UART Driver, 247  
STM32\_UART\_USART3\_RX\_DMA\_STREAM  
    UART Driver, 249  
STM32\_UART\_USART3\_TX\_DMA\_STREAM  
    UART Driver, 249  
STM32\_UART\_USE\_USART1  
    UART Driver, 247  
STM32\_UART\_USE\_USART2  
    UART Driver, 247  
STM32\_UART\_USE\_USART3  
    UART Driver, 247  
STM32\_USB  
    USB Driver, 283  
stm32\_usb.h, 559  
STM32\_USB\_BASE  
    USB Driver, 283  
STM32\_USB\_CLOCK\_REQUIRED  
    STM32F103 HAL Support, 327  
stm32\_usb\_descriptor\_t, 454  
    RXADDR0, 455  
    RXCOUNT0, 455  
    RXCOUNT1, 455  
    TXADDR0, 455  
    TXCOUNT0, 455  
    TXCOUNT1, 455  
STM32\_USB\_LOW\_POWER\_ON\_SUSPEND  
    USB Driver, 284  
stm32\_usb\_t, 455  
    EPR, 455  
STM32\_USB\_USB1\_HP\_IRQ\_PRIORITY  
    USB Driver, 284  
STM32\_USB\_USB1\_LP\_IRQ\_PRIORITY  
    USB Driver, 284  
STM32\_USB\_USE\_USB1  
    USB Driver, 284  
STM32\_USBCLK  
    STM32F103 HAL Support, 328  
STM32\_USBPRE  
    STM32F103 HAL Support, 327  
STM32\_USBPRE\_DIV1  
    STM32F103 HAL Support, 320  
STM32\_USBPRE\_DIV1P5  
    STM32F103 HAL Support, 320  
STM32\_USBRAM  
    USB Driver, 283  
STM32\_USBRAM\_BASE  
    USB Driver, 283  
STM32F100 HAL Support, 287  
    ADC1\_2\_IRQHandler, 300  
    CEC\_IRQHandler, 302  
    DMA1\_Ch1\_IRQHandler, 300  
    DMA1\_Ch2\_IRQHandler, 300  
    DMA1\_Ch3\_IRQHandler, 300  
    DMA1\_Ch4\_IRQHandler, 300  
    DMA1\_Ch5\_IRQHandler, 300  
    DMA1\_Ch6\_IRQHandler, 300  
    DMA1\_Ch7\_IRQHandler, 300  
    EXTI0\_IRQHandler, 299  
    EXTI15\_10\_IRQHandler, 302  
    EXTI1\_IRQHandler, 299  
    EXTI2\_IRQHandler, 299  
    EXTI3\_IRQHandler, 300  
    EXTI4\_IRQHandler, 300  
    EXTI9\_5\_IRQHandler, 300  
    FLASH\_IRQHandler, 299  
    I2C1\_ER\_IRQHandler, 301  
    I2C1\_EV\_IRQHandler, 301  
    I2C2\_ER\_IRQHandler, 301  
    I2C2\_EV\_IRQHandler, 301  
    PVD\_IRQHandler, 299  
    RCC\_IRQHandler, 299  
    RTC\_Alarm\_IRQHandler, 302  
    RTC\_IRQHandler, 299  
    SPI1\_IRQHandler, 301  
    SPI2\_IRQHandler, 301  
    STM32\_ACTIVATE\_PLL, 304  
    STM32\_ADCCLK, 304  
    STM32\_ADCCLK\_MAX, 295  
    STM32\_ADCPRE, 303  
    STM32\_ADCPRE\_DIV2, 297  
    STM32\_ADCPRE\_DIV4, 297  
    STM32\_ADCPRE\_DIV6, 297  
    STM32\_ADCPRE\_DIV8, 298  
    STM32\_FLASHBITS, 305  
    STM32\_HCLK, 304  
    STM32\_HPRE, 303  
    STM32\_HPRE\_DIV1, 296  
    STM32\_HPRE\_DIV128, 296  
    STM32\_HPRE\_DIV16, 296  
    STM32\_HPRE\_DIV2, 296  
    STM32\_HPRE\_DIV256, 296  
    STM32\_HPRE\_DIV4, 296  
    STM32\_HPRE\_DIV512, 296  
    STM32\_HPRE\_DIV64, 296  
    STM32\_HPRE\_DIV8, 296  
    STM32\_HSECLK\_MAX, 295  
    STM32\_HSECLK\_MIN, 295  
    STM32\_LSECLK\_MAX, 295  
    STM32\_LSECLK\_MIN, 295  
    STM32\_MCOSEL, 303  
    STM32\_MCOSEL\_HSE, 298  
    STM32\_MCOSEL\_HSI, 298  
    STM32\_MCOSEL\_NOCLOCK, 298

STM32\_MCOSEL\_PLLDIV2, 298  
STM32\_MCOSEL\_SYSCLK, 298  
STM32\_PCLK1, 304  
STM32\_PCLK1\_MAX, 295  
STM32\_PCLK2, 304  
STM32\_PCLK2\_MAX, 295  
STM32\_PLLCLKIN, 304  
STM32\_PLLCLKOUT, 304  
STM32\_PLLIN\_MAX, 295  
STM32\_PLLIN\_MIN, 295  
STM32\_PLLMUL, 304  
STM32\_PLLMUL\_VALUE, 303  
STM32\_PLLOUT\_MAX, 295  
STM32\_PLLOUT\_MIN, 295  
STM32\_PLLSRC, 302  
STM32\_PLLSRC\_HSE, 298  
STM32\_PLLSRC\_HSI, 298  
STM32\_PLLXTPRE, 303  
STM32\_PLLXTPRE\_DIV1, 298  
STM32\_PLLXTPRE\_DIV2, 298  
STM32\_PPREG1, 303  
STM32\_PPREG1\_DIV1, 296  
STM32\_PPREG1\_DIV16, 297  
STM32\_PPREG1\_DIV2, 297  
STM32\_PPREG1\_DIV4, 297  
STM32\_PPREG1\_DIV8, 297  
STM32\_PPREG2, 303  
STM32\_PPREG2\_DIV1, 297  
STM32\_PPREG2\_DIV16, 297  
STM32\_PPREG2\_DIV2, 297  
STM32\_PPREG2\_DIV4, 297  
STM32\_PPREG2\_DIV8, 297  
STM32\_RTCCLK, 304  
STM32\_RTCSEL, 303  
STM32\_RTCSEL\_HSEDIV, 299  
STM32\_RTCSEL\_LSE, 299  
STM32\_RTCSEL\_LSI, 299  
STM32\_RTCSEL\_MASK, 298  
STM32\_RTCSEL\_NOCLOCK, 298  
STM32\_SW, 302  
STM32\_SW\_HSE, 296  
STM32\_SW\_HSI, 295  
STM32\_SW\_PLL, 296  
STM32\_SYSCLK, 304  
STM32\_SYSCLK\_MAX, 295  
STM32\_TIMCLK1, 304  
STM32\_TIMCLK2, 304  
TAMPER\_IRQHandler, 299  
TIM12\_IRQHandler, 302  
TIM13\_IRQHandler, 302  
TIM14\_IRQHandler, 302  
TIM1\_BRK\_IRQHandler, 300  
TIM1\_CC\_IRQHandler, 301  
TIM1\_TRG\_COM\_IRQHandler, 301  
TIM1\_UP\_IRQHandler, 301  
TIM2\_IRQHandler, 301  
TIM3\_IRQHandler, 301  
TIM4\_IRQHandler, 301  
USART1\_IRQHandler, 302  
USART2\_IRQHandler, 302  
USART3\_IRQHandler, 302  
WWDG\_IRQHandler, 299  
STM32F103 HAL Support, 305  
ADC1\_2\_IRQHandler, 322  
ADC3\_IRQHandler, 325  
CAN1\_RX0\_IRQHandler, 322  
CAN1\_RX1\_IRQHandler, 323  
CAN1\_SCE\_IRQHandler, 323  
CAN1\_TX\_IRQHandler, 322  
DMA1\_Ch1\_IRQHandler, 322  
DMA1\_Ch2\_IRQHandler, 322  
DMA1\_Ch3\_IRQHandler, 322  
DMA1\_Ch4\_IRQHandler, 322  
DMA1\_Ch5\_IRQHandler, 322  
DMA1\_Ch6\_IRQHandler, 322  
DMA1\_Ch7\_IRQHandler, 322  
DMA2\_Ch1\_IRQHandler, 326  
DMA2\_Ch2\_IRQHandler, 326  
DMA2\_Ch3\_IRQHandler, 326  
DMA2\_Ch4\_5\_IRQHandler, 326  
EXTI0\_IRQHandler, 321  
EXTI15\_10\_IRQHandler, 324  
EXTI1\_IRQHandler, 321  
EXTI2\_IRQHandler, 321  
EXTI3\_IRQHandler, 321  
EXTI4\_IRQHandler, 322  
EXTI9\_5\_IRQHandler, 323  
FLASH\_IRQHandler, 321  
FSMC\_IRQHandler, 325  
I2C1\_ER\_IRQHandler, 324  
I2C1\_EV\_IRQHandler, 323  
I2C2\_ER\_IRQHandler, 324  
I2C2\_EV\_IRQHandler, 324  
PVD\_IRQHandler, 321  
RCC\_IRQHandler, 321  
RTC\_Alarm\_IRQHandler, 324  
RTC\_IRQHandler, 321  
SDIO\_IRQHandler, 325  
SPI1\_IRQHandler, 324  
SPI2\_IRQHandler, 324  
SPI3\_IRQHandler, 325  
STM32\_ACTIVATE\_PLL, 327  
STM32\_ADCCLK, 328  
STM32\_ADCCLK\_MAX, 317  
STM32\_ADCPRE, 327  
STM32\_ADCPRE\_DIV2, 319  
STM32\_ADCPRE\_DIV4, 319  
STM32\_ADCPRE\_DIV6, 319  
STM32\_ADCPRE\_DIV8, 319  
STM32\_FLASHBITS, 328  
STM32\_HCLK, 328  
STM32\_HPRE, 327  
STM32\_HPRE\_DIV1, 317  
STM32\_HPRE\_DIV128, 318  
STM32\_HPRE\_DIV16, 318  
STM32\_HPRE\_DIV2, 318  
STM32\_HPRE\_DIV256, 318  
STM32\_HPRE\_DIV4, 318

STM32\_HPRE\_DIV512, 318  
STM32\_HPRE\_DIV64, 318  
STM32\_HPRE\_DIV8, 318  
STM32\_HSECLK\_MAX, 316  
STM32\_HSECLK\_MIN, 316  
STM32\_LSECLK\_MAX, 316  
STM32\_LSECLK\_MIN, 317  
STM32\_MCOSEL, 327  
STM32\_MCOSEL\_HSE, 320  
STM32\_MCOSEL\_HSI, 320  
STM32\_MCOSEL\_NOCLOCK, 320  
STM32\_MCOSEL\_PLLDIV2, 320  
STM32\_MCOSEL\_SYSCLK, 320  
STM32\_PCLK1, 328  
STM32\_PCLK1\_MAX, 317  
STM32\_PCLK2, 328  
STM32\_PCLK2\_MAX, 317  
STM32\_PLLCLKIN, 328  
STM32\_PLLCLKOUT, 328  
STM32\_PLLIN\_MAX, 317  
STM32\_PLLIN\_MIN, 317  
STM32\_PLLMUL, 327  
STM32\_PLLMUL\_VALUE, 326  
STM32\_PLLOUT\_MAX, 317  
STM32\_PLLOUT\_MIN, 317  
STM32\_PLLSRC, 326  
STM32\_PLLSRC\_HSE, 319  
STM32\_PLLSRC\_HSI, 319  
STM32\_PLLXTPRE, 326  
STM32\_PLLXTPRE\_DIV1, 320  
STM32\_PLLXTPRE\_DIV2, 320  
STM32\_PPREG, 327  
STM32\_PPREG1\_DIV1, 318  
STM32\_PPREG1\_DIV16, 319  
STM32\_PPREG1\_DIV2, 318  
STM32\_PPREG1\_DIV4, 318  
STM32\_PPREG1\_DIV8, 318  
STM32\_PPREG2, 327  
STM32\_PPREG2\_DIV1, 319  
STM32\_PPREG2\_DIV16, 319  
STM32\_PPREG2\_DIV2, 319  
STM32\_PPREG2\_DIV4, 319  
STM32\_PPREG2\_DIV8, 319  
STM32\_RTCCLK, 328  
STM32\_RTCSEL, 327  
STM32\_RTCSEL\_HSEDIV, 321  
STM32\_RTCSEL\_LSE, 320  
STM32\_RTCSEL\_LSI, 321  
STM32\_RTCSEL\_MASK, 320  
STM32\_RTCSEL\_NOCLOCK, 320  
STM32\_SW, 326  
STM32\_SW\_HSE, 317  
STM32\_SW\_HSI, 317  
STM32\_SW\_PLL, 317  
STM32\_SYSCLK, 328  
STM32\_SYSCLK\_MAX, 316  
STM32\_TIMCLK1, 328  
STM32\_TIMCLK2, 328  
STM32\_USB\_CLOCK\_REQUIRED, 327  
STM32\_USBCLK, 328  
STM32\_USBPRE, 327  
STM32\_USBPRE\_DIV1, 320  
STM32\_USBPRE\_DIV1P5, 320  
TAMPER\_IRQHandler, 321  
TIM1\_BRK\_IRQHandler, 323  
TIM1\_CC\_IRQHandler, 323  
TIM1\_TRG\_COM\_IRQHandler, 323  
TIM1\_UP\_IRQHandler, 323  
TIM2\_IRQHandler, 323  
TIM3\_IRQHandler, 323  
TIM4\_IRQHandler, 323  
TIM5\_IRQHandler, 325  
TIM6\_IRQHandler, 325  
TIM7\_IRQHandler, 325  
TIM8\_BRK\_IRQHandler, 324  
TIM8\_CC\_IRQHandler, 325  
TIM8\_TRG\_COM\_IRQHandler, 325  
TIM8\_UP\_IRQHandler, 325  
UART4\_IRQHandler, 325  
UART5\_IRQHandler, 325  
USART1\_IRQHandler, 324  
USART2\_IRQHandler, 324  
USART3\_IRQHandler, 324  
USB\_FS\_WKUP\_IRQHandler, 324  
USB\_HP\_IRQHandler, 322  
USB\_LP\_IRQHandler, 323  
WWDG\_IRQHandler, 321  
STM32F105/F107 HAL Support, 329  
ADC1\_2\_IRQHandler, 342  
CAN1\_RX0\_IRQHandler, 343  
CAN1\_RX1\_IRQHandler, 343  
CAN1\_SCE\_IRQHandler, 343  
CAN1\_TX\_IRQHandler, 342  
CAN2\_RX0\_IRQHandler, 346  
CAN2\_RX1\_IRQHandler, 346  
CAN2\_SCE\_IRQHandler, 346  
CAN2\_TX\_IRQHandler, 346  
DMA1\_Ch1\_IRQHandler, 342  
DMA1\_Ch2\_IRQHandler, 342  
DMA1\_Ch3\_IRQHandler, 342  
DMA1\_Ch4\_IRQHandler, 342  
DMA1\_Ch5\_IRQHandler, 342  
DMA1\_Ch6\_IRQHandler, 342  
DMA1\_Ch7\_IRQHandler, 342  
DMA2\_Ch1\_IRQHandler, 345  
DMA2\_Ch2\_IRQHandler, 345  
DMA2\_Ch3\_IRQHandler, 345  
DMA2\_Ch4\_IRQHandler, 345  
DMA2\_Ch5\_IRQHandler, 345  
ETH\_IRQHandler, 345  
ETH\_WKUP\_IRQHandler, 345  
EXTI0\_IRQHandler, 341  
EXTI15\_10\_IRQHandler, 344  
EXTI1\_IRQHandler, 341  
EXTI2\_IRQHandler, 342  
EXTI3\_IRQHandler, 342  
EXTI4\_IRQHandler, 342  
EXTI9\_5\_IRQHandler, 343

FLASH\_IRQHandler, 341  
I2C1\_ER\_IRQHandler, 344  
I2C1\_EV\_IRQHandler, 343  
I2C2\_ER\_IRQHandler, 344  
I2C2\_EV\_IRQHandler, 344  
OTG\_FS\_IRQHandler, 346  
OTG\_FS\_WKUP\_IRQHandler, 344  
PVD\_IRQHandler, 341  
RCC\_IRQHandler, 341  
RTC\_Alarm\_IRQHandler, 344  
RTC\_IRQHandler, 341  
SPI1\_IRQHandler, 344  
SPI2\_IRQHandler, 344  
SPI3\_IRQHandler, 345  
STM32\_ACTIVATE\_PLL1, 348  
STM32\_ACTIVATE\_PLL2, 348  
STM32\_ACTIVATE\_PLL3, 348  
STM32\_ADCCLK, 350  
STM32\_ADCCLK\_MAX, 337  
STM32\_ADCPRE, 348  
STM32\_ADCPRE\_DIV2, 339  
STM32\_ADCPRE\_DIV4, 339  
STM32\_ADCPRE\_DIV6, 339  
STM32\_ADCPRE\_DIV8, 339  
STM32\_FLASHBITS, 350  
STM32\_HCLK, 350  
STM32\_HPRE, 347  
STM32\_HPRE\_DIV1, 337  
STM32\_HPRE\_DIV128, 338  
STM32\_HPRE\_DIV16, 338  
STM32\_HPRE\_DIV2, 337  
STM32\_HPRE\_DIV256, 338  
STM32\_HPRE\_DIV4, 337  
STM32\_HPRE\_DIV512, 338  
STM32\_HPRE\_DIV64, 338  
STM32\_HPRE\_DIV8, 338  
STM32\_HSECLK\_MAX, 336  
STM32\_HSECLK\_MIN, 336  
STM32\_I2S\_CLOCK\_REQUIRED, 348  
STM32\_LSECLK\_MAX, 336  
STM32\_LSECLK\_MIN, 336  
STM32\_MCOSEL, 348  
STM32\_MCOSEL\_HSE, 340  
STM32\_MCOSEL\_HSI, 340  
STM32\_MCOSEL\_NOCLOCK, 340  
STM32\_MCOSEL\_PLL2, 340  
STM32\_MCOSEL\_PLL3, 340  
STM32\_MCOSEL\_PLL3DIV2, 340  
STM32\_MCOSEL\_PLLDIV2, 340  
STM32\_MCOSEL\_SYSCLK, 340  
STM32\_MCOSEL\_XT1, 340  
STM32\_OTG\_CLOCK\_REQUIRED, 348  
STM32\_OTGFSCLK, 350  
STM32\_OTGFSPRE, 348  
STM32\_OTGFSPRE\_DIV2, 339  
STM32\_OTGFSPRE\_DIV3, 339  
STM32\_PCLK1, 350  
STM32\_PCLK1\_MAX, 337  
STM32\_PCLK2, 350  
STM32\_PCLK2\_MAX, 337  
STM32\_PLL1IN\_MAX, 336  
STM32\_PLL1IN\_MIN, 336  
STM32\_PLL1VCO\_MAX, 336  
STM32\_PLL1VCO\_MIN, 336  
STM32\_PLL23IN\_MAX, 336  
STM32\_PLL23IN\_MIN, 336  
STM32\_PLL23VCO\_MAX, 337  
STM32\_PLL23VCO\_MIN, 337  
STM32\_PLL2CLKIN, 349  
STM32\_PLL2CLKOUT, 349  
STM32\_PLL2MUL, 349  
STM32\_PLL2MUL\_VALUE, 347  
STM32\_PLL2VCO, 349  
STM32\_PLL3CLKIN, 349  
STM32\_PLL3CLKOUT, 349  
STM32\_PLL3MUL, 349  
STM32\_PLL3MUL\_VALUE, 347  
STM32\_PLL3VCO, 349  
STM32\_PLLCLKIN, 349  
STM32\_PLLCLKOUT, 349  
STM32\_PLLMUL, 348  
STM32\_PLLMUL\_VALUE, 347  
STM32\_PLLSRC, 346  
STM32\_PLLSRC\_HSI, 339  
STM32\_PLLSRC\_PREDIV1, 339  
STM32\_PLLVCO, 349  
STM32\_PPREG1, 347  
STM32\_PPREG1\_DIV1, 338  
STM32\_PPREG1\_DIV16, 338  
STM32\_PPREG1\_DIV2, 338  
STM32\_PPREG1\_DIV4, 338  
STM32\_PPREG1\_DIV8, 338  
STM32\_PPREG2, 347  
STM32\_PPREG2\_DIV1, 338  
STM32\_PPREG2\_DIV16, 339  
STM32\_PPREG2\_DIV2, 339  
STM32\_PPREG2\_DIV4, 339  
STM32\_PPREG2\_DIV8, 339  
STM32\_PREDIV1, 348  
STM32\_PREDIV1\_VALUE, 346  
STM32\_PREDIV1CLK, 349  
STM32\_PREDIV1SRC, 346  
STM32\_PREDIV1SRC\_HSE, 341  
STM32\_PREDIV1SRC\_PLL2, 341  
STM32\_PREDIV2, 348  
STM32\_PREDIV2\_VALUE, 347  
STM32\_RTCCLK, 350  
STM32\_RTCSEL, 348  
STM32\_RTCSEL\_HSEDIV, 341  
STM32\_RTCSEL\_LSE, 340  
STM32\_RTCSEL\_LSI, 341  
STM32\_RTCSEL\_MASK, 340  
STM32\_RTCSEL\_NOCLOCK, 340  
STM32\_SPII2S\_MAX, 337  
STM32\_SW, 346  
STM32\_SW\_HSE, 337  
STM32\_SW\_HSI, 337  
STM32\_SW\_PLL, 337

STM32\_SYSCLK, 350  
STM32\_SYSCLK\_MAX, 336  
STM32\_TIMCLK1, 350  
STM32\_TIMCLK2, 350  
TAMPER\_IRQHandler, 341  
TIM1\_BRK\_IRQHandler, 343  
TIM1\_CC\_IRQHandler, 343  
TIM1\_TRG\_COM\_IRQHandler, 343  
TIM1\_UP\_IRQHandler, 343  
TIM2\_IRQHandler, 343  
TIM3\_IRQHandler, 343  
TIM4\_IRQHandler, 343  
TIM5\_IRQHandler, 344  
TIM6\_IRQHandler, 345  
TIM7\_IRQHandler, 345  
UART4\_IRQHandler, 345  
UART5\_IRQHandler, 345  
USART1\_IRQHandler, 344  
USART2\_IRQHandler, 344  
USART3\_IRQHandler, 344  
WWDG\_IRQHandler, 341  
STM32F1xx ADC Support, 352  
STM32F1xx CAN Support, 352  
STM32F1xx DMA Support, 358  
  \_stm32\_dma\_streams, 365  
  CH IRQ\_HANDLER, 362–364  
  dmaInit, 364  
  dmaStartMemcpy, 371  
  dmaStreamAllocate, 364  
  dmaStreamClearInterrupt, 371  
  dmaStreamDisable, 370  
  dmaStreamEnable, 370  
  dmaStreamGetTransactionSize, 369  
  dmaStreamRelease, 365  
  dmaStreamSetMemory0, 368  
  dmaStreamSetMode, 369  
  dmaStreamSetPeripheral, 367  
  dmaStreamSetTransactionSize, 368  
  dmaWaitCompletion, 372  
  STM32\_DMA1\_STREAMS\_MASK, 366  
  STM32\_DMA2\_STREAMS\_MASK, 366  
  STM32\_DMA\_CCR\_RESET\_VALUE, 366  
  STM32\_DMA\_CR\_CHSEL, 367  
  STM32\_DMA\_CR\_CHSEL\_MASK, 367  
  STM32\_DMA\_CR\_DMEIE, 367  
  STM32\_DMA\_GETCHANNEL, 366  
  STM32\_DMA\_IS\_VALID\_ID, 367  
  STM32\_DMA\_ISR\_MASK, 366  
  STM32\_DMA\_STREAM, 367  
  STM32\_DMA\_STREAM\_ID, 367  
  STM32\_DMA\_STREAM\_ID\_MSK, 366  
  STM32\_DMA\_STREAMS, 366  
  stm32\_dmaisr\_t, 372  
STM32F1xx Drivers, 350  
STM32F1xx EXT Support, 352  
STM32F1xx GPT Support, 353  
STM32F1xx I2C Support, 353  
STM32F1xx ICU Support, 353  
STM32F1xx Initialization Support, 351  
STM32F1xx MAC Support, 354  
STM32F1xx PAL Support, 354  
STM32F1xx Platform Drivers, 358  
STM32F1xx PWM Support, 355  
STM32F1xx RCC Support, 373  
  rccDisableADC1, 381  
  rccDisableAHB, 380  
  rccDisableAPB1, 378  
  rccDisableAPB2, 379  
  rccDisableBKPInterface, 382  
  rccDisableCAN1, 383  
  rccDisableDMA1, 384  
  rccDisableDMA2, 384  
  rccDisableETH, 385  
  rccDisableI2C1, 386  
  rccDisableI2C2, 387  
  rccDisablePWRInterface, 382  
  rccDisableSDIO, 387  
  rccDisableSPI1, 388  
  rccDisableSPI2, 389  
  rccDisableSPI3, 390  
  rccDisableTIM1, 390  
  rccDisableTIM2, 391  
  rccDisableTIM3, 392  
  rccDisableTIM4, 392  
  rccDisableTIM5, 393  
  rccDisableTIM8, 394  
  rccDisableUART4, 397  
  rccDisableUART5, 397  
  rccDisableUSART1, 394  
  rccDisableUSART2, 395  
  rccDisableUSART3, 396  
  rccDisableUSB, 398  
  rccEnableADC1, 381  
  rccEnableAHB, 379  
  rccEnableAPB1, 377  
  rccEnableAPB2, 378  
  rccEnableBKPInterface, 381  
  rccEnableCAN1, 383  
  rccEnableDMA1, 384  
  rccEnableDMA2, 384  
  rccEnableETH, 385  
  rccEnableI2C1, 386  
  rccEnableI2C2, 387  
  rccEnablePWRInterface, 382  
  rccEnableSDIO, 387  
  rccEnableSPI1, 388  
  rccEnableSPI2, 389  
  rccEnableSPI3, 389  
  rccEnableTIM1, 390  
  rccEnableTIM2, 391  
  rccEnableTIM3, 391  
  rccEnableTIM4, 392  
  rccEnableTIM5, 393  
  rccEnableTIM8, 393  
  rccEnableUART4, 396  
  rccEnableUART5, 397  
  rccEnableUSART1, 394  
  rccEnableUSART2, 395

rccEnableUSART3, 396  
rccEnableUSB, 398  
rccResetADC1, 381  
rccResetAHB, 380  
rccResetAPB1, 378  
rccResetAPB2, 379  
rccResetBKP, 382  
rccResetBKPIInterface, 382  
rccResetCAN1, 383  
rccResetDMA1, 384  
rccResetDMA2, 385  
rccResetETH, 386  
rccResetI2C1, 386  
rccResetI2C2, 387  
rccResetPWRIInterface, 383  
rccResetSDIO, 388  
rccResetSPI1, 388  
rccResetSPI2, 389  
rccResetSPI3, 390  
rccResetTIM1, 391  
rccResetTIM2, 391  
rccResetTIM3, 392  
rccResetTIM4, 393  
rccResetTIM5, 393  
rccResetTIM8, 394  
rccResetUART4, 397  
rccResetUART5, 398  
rccResetUSART1, 395  
rccResetUSART2, 395  
rccResetUSART3, 396  
rccResetUSB, 398  
STM32F1xx RTC Support, 356  
STM32F1xx SDC Support, 356  
STM32F1xx Serial Support, 356  
STM32F1xx SPI Support, 357  
STM32F1xx UART Support, 357  
STM32F1xx USB Support, 358  
stop  
    TimeMeasurement, 456  
TAMPER\_IRQHandler  
    STM32F100 HAL Support, 299  
    STM32F103 HAL Support, 321  
    STM32F105/F107 HAL Support, 341  
thread  
    ADCDriver, 404  
    I2CDriver, 422  
    SDCDriver, 444  
    SPIDriver, 452  
tim  
    GPTDriver, 420  
    ICUDriver, 427  
    PWMDriver, 440  
TIM12\_IRQHandler  
    STM32F100 HAL Support, 302  
TIM13\_IRQHandler  
    STM32F100 HAL Support, 302  
TIM14\_IRQHandler  
    STM32F100 HAL Support, 302  
                TIM1\_BRK\_IRQHandler  
                    STM32F100 HAL Support, 300  
                    STM32F103 HAL Support, 323  
                    STM32F105/F107 HAL Support, 343  
                TIM1\_CC\_IRQHandler  
                    STM32F100 HAL Support, 301  
                    STM32F103 HAL Support, 323  
                    STM32F105/F107 HAL Support, 343  
                TIM1\_TRG\_COM\_IRQHandler  
                    STM32F100 HAL Support, 301  
                    STM32F103 HAL Support, 323  
                    STM32F105/F107 HAL Support, 343  
                TIM1\_UP\_IRQHandler  
                    STM32F100 HAL Support, 301  
                    STM32F103 HAL Support, 323  
                    STM32F105/F107 HAL Support, 343  
                TIM2\_IRQHandler  
                    STM32F100 HAL Support, 301  
                    STM32F103 HAL Support, 323  
                    STM32F105/F107 HAL Support, 343  
                TIM3\_IRQHandler  
                    STM32F100 HAL Support, 301  
                    STM32F103 HAL Support, 323  
                    STM32F105/F107 HAL Support, 343  
                TIM4\_IRQHandler  
                    STM32F100 HAL Support, 301  
                    STM32F103 HAL Support, 323  
                    STM32F105/F107 HAL Support, 343  
                TIM5\_IRQHandler  
                    STM32F103 HAL Support, 325  
                    STM32F105/F107 HAL Support, 344  
                TIM6\_IRQHandler  
                    STM32F103 HAL Support, 325  
                    STM32F105/F107 HAL Support, 345  
                TIM7\_IRQHandler  
                    STM32F103 HAL Support, 325  
                    STM32F105/F107 HAL Support, 345  
                TIM8\_BRK\_IRQHandler  
                    STM32F103 HAL Support, 324  
                TIM8\_CC\_IRQHandler  
                    STM32F103 HAL Support, 325  
                TIM8\_TRG\_COM\_IRQHandler  
                    STM32F103 HAL Support, 325  
                TIM8\_UP\_IRQHandler  
                    STM32F103 HAL Support, 325  
TIME  
    CANRxFrame, 410  
Time Measurement Driver., 230  
    TimeMeasurement, 232  
        tmlInit, 231  
        tmObjectInit, 231  
        tmStartMeasurement, 231  
        tmStopMeasurement, 231  
    TimeMeasurement, 456  
        best, 456  
        last, 456  
        start, 456  
        stop, 456  
    Time Measurement Driver., 232

worst, 456  
tm.c, 560  
tm.h, 560  
tmlInit  
    Time Measurement Driver., 231  
tmObjectInit  
    Time Measurement Driver., 231  
tmStartMeasurement  
    Time Measurement Driver., 231  
tmStopMeasurement  
    Time Measurement Driver., 231  
transmitting  
    USBDriver, 466  
tv\_msec  
    RTCTime, 442  
tv\_sec  
    RTCAlarm, 440  
    RTCTime, 442  
TXADDR0  
    stm32\_usb\_descriptor\_t, 455  
txbuf  
    USBInEndpointState, 470  
txcnt  
    USBInEndpointState, 470  
TXCOUNT0  
    stm32\_usb\_descriptor\_t, 455  
TXCOUNT1  
    stm32\_usb\_descriptor\_t, 455  
txdmamode  
    SPIDriver, 452  
txempty\_event  
    CANDriver, 408  
txend1\_cb  
    UARTConfig, 458  
txend2\_cb  
    UARTConfig, 458  
txsem  
    CANDriver, 408  
txsize  
    USBInEndpointState, 470  
txstate  
    UARTDriver, 461  
  
UART Driver, 232  
    CH\_IRQ\_HANDLER, 243  
    STM32\_UART\_DMA\_ERROR\_HOOK, 248  
    STM32\_UART\_USART1\_DMA\_PRIORITY, 248  
    STM32\_UART\_USART1\_IRQ\_PRIORITY, 247  
    STM32\_UART\_USART1\_RX\_DMA\_STREAM, 248  
    STM32\_UART\_USART1\_TX\_DMA\_STREAM, 248  
    STM32\_UART\_USART2\_DMA\_PRIORITY, 248  
    STM32\_UART\_USART2\_IRQ\_PRIORITY, 247  
    STM32\_UART\_USART2\_RX\_DMA\_STREAM, 248  
    STM32\_UART\_USART2\_TX\_DMA\_STREAM, 249  
    STM32\_UART\_USART3\_DMA\_PRIORITY, 248  
    STM32\_UART\_USART3\_IRQ\_PRIORITY, 247  
    STM32\_UART\_USART3\_RX\_DMA\_STREAM, 249  
    STM32\_UART\_USART3\_TX\_DMA\_STREAM, 249  
    STM32\_UART\_USE\_USART1, 247  
    STM32\_UART\_USE\_USART2, 247  
    STM32\_UART\_USE\_USART3, 247  
    UART\_READY, 250  
    UART\_RX\_ACTIVE, 250  
    UART\_RX\_COMPLETE, 250  
    UART\_RX\_IDLE, 250  
    UART\_STOP, 250  
    UART\_TX\_ACTIVE, 250  
    UART\_TX\_COMPLETE, 250  
    UART\_TX\_IDLE, 250  
    UART\_UNINIT, 250  
    UART\_BREAK\_DETECTED, 247  
    UART\_FRAMING\_ERROR, 246  
    uart\_lld\_init, 243  
    uart\_lld\_start, 244  
    uart\_lld\_start\_receive, 245  
    uart\_lld\_start\_send, 245  
    uart\_lld\_stop, 244  
    uart\_lld\_stop\_receive, 246  
    uart\_lld\_stop\_send, 245  
    UART\_NO\_ERROR, 246  
    UART\_NOISE\_ERROR, 247  
    UART\_OVERRUN\_ERROR, 247  
    UART\_PARITY\_ERROR, 246  
    uartccb\_t, 249  
    uartccb\_t, 249  
    UARTD1, 246  
    UARTD2, 246  
    UARTD3, 246  
    UARTDriver, 249  
    uartecb\_t, 250  
    uartflags\_t, 249  
    uartInit, 237  
    uartObjectInit, 237  
    uartrxstate\_t, 250  
    uartStart, 237  
    uartStartReceive, 241  
    uartStartReceive1, 241  
    uartStartSend, 238  
    uartStartSend1, 239  
    uartstate\_t, 250  
    uartStop, 238  
    uartStopReceive, 242  
    uartStopReceive1, 242  
    uartStopSend, 239  
    uartStopSend1, 240  
    uarttxstate\_t, 250  
    uart.c, 561  
    uart.h, 562  
    UART4\_IRQHandler  
        STM32F103 HAL Support, 325  
        STM32F105/F107 HAL Support, 345  
    UART5\_IRQHandler  
        STM32F103 HAL Support, 325  
        STM32F105/F107 HAL Support, 345  
    UART\_READY  
        UART Driver, 250  
    UART\_RX\_ACTIVE  
        UART Driver, 250

UART\_RX\_COMPLETE  
    UART Driver, 250

UART\_RX\_IDLE  
    UART Driver, 250

UART\_STOP  
    UART Driver, 250

UART\_TX\_ACTIVE  
    UART Driver, 250

UART\_TX\_COMPLETE  
    UART Driver, 250

UART\_TX\_IDLE  
    UART Driver, 250

UART\_UNINIT  
    UART Driver, 250

UART\_BREAK\_DETECTED  
    UART Driver, 247

UART\_FRAMING\_ERROR  
    UART Driver, 246

uart\_lld.c, 563

uart\_lld.h, 564

uart\_lld\_init  
    UART Driver, 243

uart\_lld\_start  
    UART Driver, 244

uart\_lld\_start\_receive  
    UART Driver, 245

uart\_lld\_start\_send  
    UART Driver, 245

uart\_lld\_stop  
    UART Driver, 244

uart\_lld\_stop\_receive  
    UART Driver, 246

uart\_lld\_stop\_send  
    UART Driver, 245

UART\_NO\_ERROR  
    UART Driver, 246

UART\_NOISE\_ERROR  
    UART Driver, 247

UART\_OVERRUN\_ERROR  
    UART Driver, 247

UART\_PARITY\_ERROR  
    UART Driver, 246

uartcb\_t  
    UART Driver, 249

uartccb\_t  
    UART Driver, 249

UARTConfig, 456

    cr1, 458

    cr2, 458

    cr3, 458

    rxchar\_cb, 458

    rxend\_cb, 458

    rxerr\_cb, 458

    speed, 458

    txend1\_cb, 458

    txend2\_cb, 458

UARTD1  
    UART Driver, 246

UARTD2

    UART Driver, 246

    UART Driver, 246

    UART Driver, 246

    config, 461

    dmemode, 461

    dmarx, 461

    dmatx, 461

    rdbuf, 461

    rxstate, 461

    state, 461

    txstate, 461

    UART Driver, 249

    usart, 461

    uartecb\_t

    UART Driver, 250

    uartflags\_t

    UART Driver, 249

    uartInit

        UART Driver, 237

    uartObjectInit

        UART Driver, 237

    uartrxstate\_t

        UART Driver, 250

    uartStart

        UART Driver, 237

    uartStartReceive

        UART Driver, 241

    uartStartReceive1

        UART Driver, 241

    uartStartSend

        UART Driver, 238

    uartStartSend1

        UART Driver, 239

    uartstate\_t

        UART Driver, 250

    uartStop

        UART Driver, 238

    uartStopReceive

        UART Driver, 242

    uartStopReceive1

        UART Driver, 242

    uartStopSend

        UART Driver, 239

    uartStopSend1

        UART Driver, 240

    uarttxstate\_t

        UART Driver, 250

    ud\_size

        USBDescriptor, 464

    ud\_string

        USBDescriptor, 464

    US2RTT

        HAL Driver, 83

    usart

        UARTDriver, 461

    USART1\_IRQHandler

        STM32F100 HAL Support, 302

        STM32F103 HAL Support, 324

STM32F105/F107 HAL Support, 344  
USART2\_IRQHandler  
  STM32F100 HAL Support, 302  
  STM32F103 HAL Support, 324  
  STM32F105/F107 HAL Support, 344  
USART3\_IRQHandler  
  STM32F100 HAL Support, 302  
  STM32F103 HAL Support, 324  
  STM32F105/F107 HAL Support, 344  
USART\_CR2\_STOP0P5\_BITS  
  Serial Driver, 202  
USART\_CR2\_STOP1\_BITS  
  Serial Driver, 202  
USART\_CR2\_STOP1P5\_BITS  
  Serial Driver, 202  
USART\_CR2\_STOP2\_BITS  
  Serial Driver, 202  
USB Driver, 251  
  \_usb\_ep0in, 265  
  \_usb\_ep0out, 266  
  \_usb\_ep0setup, 264  
  \_usb\_isr\_invoke\_event\_cb, 281  
  \_usb\_isr\_invoke\_in\_cb, 282  
  \_usb\_isr\_invoke\_out\_cb, 282  
  \_usb\_isr\_invoke\_setup\_cb, 282  
  \_usb\_isr\_invoke\_sof\_cb, 281  
  \_usb\_reset, 264  
  CH\_IRQ\_HANDLER, 267  
  EP\_STATUS\_ACTIVE, 287  
  EP\_STATUS\_DISABLED, 287  
  EP\_STATUS\_STALLED, 287  
  EPR\_TOGGLE\_MASK, 283  
  in, 274  
  out, 274  
  STM32\_USB, 283  
  STM32\_USB\_BASE, 283  
  STM32\_USB\_LOW\_POWER\_ON\_SUSPEND, 284  
  STM32\_USB\_USB1\_HP\_IRQ\_PRIORITY, 284  
  STM32\_USB\_USB1\_LP\_IRQ\_PRIORITY, 284  
  STM32\_USB\_USE\_USB1, 284  
  STM32\_USBRAM, 283  
  STM32\_USBRAM\_BASE, 283  
  USB\_ACTIVE, 287  
  USB\_EP0\_ERROR, 287  
  USB\_EP0\_RX, 287  
  USB\_EP0\_SENDING\_STS, 287  
  USB\_EP0\_TX, 287  
  USB\_EP0\_WAITING\_SETUP, 287  
  USB\_EP0\_WAITING\_STS, 287  
  USB\_EVENT\_ADDRESS, 287  
  USB\_EVENT\_CONFIGURED, 287  
  USB\_EVENT\_RESET, 287  
  USB\_EVENT\_STALLED, 287  
  USB\_EVENT\_SUSPEND, 287  
  USB\_EVENT\_WAKEUP, 287  
  USB\_READY, 286  
  USB\_SELECTED, 287  
  USB\_STOP, 286  
  USB\_UNINIT, 286  
  USB\_ADDR2PTR, 283  
  USB\_DESC\_BCD, 275  
  USB\_DESC\_BYTEx, 275  
  USB\_DESC\_CONFIGURATION, 275  
  USB\_DESC\_DEVICE, 275  
  USB\_DESC\_ENDPOINT, 276  
  USB\_DESC\_INDEX, 274  
  USB\_DESC\_INTERFACE, 275  
  USB\_DESC\_WORD, 275  
  USB\_ENDPOINTS\_NUMBER, 282  
  USB\_EP\_MODE\_PACKET, 276  
  USB\_EP\_MODE\_TRANSACTION, 276  
  USB\_EP\_MODE\_TYPE, 276  
  USB\_EP\_MODE\_TYPE\_BULK, 276  
  USB\_EP\_MODE\_TYPE\_CTRL, 276  
  USB\_EP\_MODE\_TYPE\_INTR, 276  
  USB\_EP\_MODE\_TYPE\_ISOC, 276  
  USB\_GET\_DESCRIPTOR, 283  
  usb\_lld\_clear\_in, 274  
  usb\_lld\_clear\_out, 274  
  usb\_lld\_disable\_endpoints, 270  
  usb\_lld\_fetch\_word, 284  
  usb\_lld\_get\_frame\_number, 284  
  usb\_lld\_get\_packet\_size, 285  
  usb\_lld\_get\_status\_in, 270  
  usb\_lld\_get\_status\_out, 270  
  usb\_lld\_get\_transaction\_size, 284  
  usb\_lld\_init, 268  
  usb\_lld\_init\_endpoint, 270  
  usb\_lld\_prepare\_receive, 272  
  usb\_lld\_prepare\_transmit, 272  
  usb\_lld\_read\_packet\_buffer, 271  
  usb\_lld\_read\_setup, 271  
  usb\_lld\_reset, 269  
  usb\_lld\_set\_address, 269  
  usb\_lld\_stall\_in, 273  
  usb\_lld\_stall\_out, 273  
  usb\_lld\_start, 268  
  usb\_lld\_start\_in, 273  
  usb\_lld\_start\_out, 273  
  usb\_lld\_stop, 269  
  usb\_lld\_write\_packet\_buffer, 272  
  USB\_MAX\_ENDPOINTS, 283  
  USB\_PMA\_SIZE, 283  
  USB\_SET\_ADDRESS\_MODE, 283  
  uscallback\_t, 285  
  usbConnectBus, 276  
  USBD1, 274  
  usbDisableEndpoints, 261  
  usbDisconnectBus, 277  
  USBDriver, 285  
  usbep0state\_t, 287  
  usbep\_t, 285  
  usbepcallback\_t, 286  
  usbepstatus\_t, 287  
  usbevent\_t, 287  
  usbeventcb\_t, 286  
  usbgetdescriptor\_t, 286  
  usbGetFrameNumber, 277

usbGetReceivePacketSize, 280  
usbGetReceiveStatusI, 277  
usbGetReceiveTransactionSizeI, 279  
usbGetTransmitStatusI, 277  
usbInit, 259  
usbInitEndpointI, 260  
usbObjectInit, 259  
usbPrepareReceive, 279  
usbPrepareTransmit, 279  
usbReadPacketBuffer, 278  
usbReadSetup, 280  
usbreqhandler\_t, 286  
usbSetupTransfer, 280  
usbStallReceiveI, 263  
usbStallTransmitI, 263  
usbStart, 259  
usbStartReceiveI, 261  
usbStartTransmitI, 262  
usbstate\_t, 286  
usbStop, 260  
usbWritePacketBuffer, 278  
  
usb.c, 565  
usb.h, 566  
USB\_ACTIVE  
    USB Driver, 287  
USB\_EP0\_ERROR  
    USB Driver, 287  
USB\_EP0\_RX  
    USB Driver, 287  
USB\_EP0\_SENDING\_STS  
    USB Driver, 287  
USB\_EP0\_TX  
    USB Driver, 287  
USB\_EP0\_WAITING\_SETUP  
    USB Driver, 287  
USB\_EP0\_WAITING\_STS  
    USB Driver, 287  
USB\_EVENT\_ADDRESS  
    USB Driver, 287  
USB\_EVENT\_CONFIGURED  
    USB Driver, 287  
USB\_EVENT\_RESET  
    USB Driver, 287  
USB\_EVENT\_STALLED  
    USB Driver, 287  
USB\_EVENT\_SUSPEND  
    USB Driver, 287  
USB\_EVENT\_WAKEUP  
    USB Driver, 287  
USB\_READY  
    USB Driver, 286  
USB\_SELECTED  
    USB Driver, 287  
USB\_STOP  
    USB Driver, 286  
USB\_UNINIT  
    USB Driver, 286  
USB\_ADDR2PTR  
    USB Driver, 283  
  
usb\_config  
    SerialUSBConfig, 448  
USB\_DESC\_BCD  
    USB Driver, 275  
USB\_DESC\_BYTE  
    USB Driver, 275  
USB\_DESC\_CONFIGURATION  
    USB Driver, 275  
USB\_DESC\_DEVICE  
    USB Driver, 275  
USB\_DESC\_ENDPOINT  
    USB Driver, 276  
USB\_DESC\_INDEX  
    USB Driver, 274  
USB\_DESC\_INTERFACE  
    USB Driver, 275  
USB\_DESC\_WORD  
    USB Driver, 275  
USB\_ENDPOINTS\_NUMBER  
    USB Driver, 282  
USB\_EP\_MODE\_PACKET  
    USB Driver, 276  
USB\_EP\_MODE\_TRANSACTION  
    USB Driver, 276  
USB\_EP\_MODE\_TYPE  
    USB Driver, 276  
USB\_EP\_MODE\_TYPE\_BULK  
    USB Driver, 276  
USB\_EP\_MODE\_TYPE\_CTRL  
    USB Driver, 276  
USB\_EP\_MODE\_TYPE\_INTR  
    USB Driver, 276  
USB\_EP\_MODE\_TYPE\_ISOC  
    USB Driver, 276  
USB\_FS\_WKUP\_IRQHandler  
    STM32F103 HAL Support, 324  
USB\_GET\_DESCRIPTOR  
    USB Driver, 283  
USB\_HP\_IRQHandler  
    STM32F103 HAL Support, 322  
usb\_lld.c, 569  
    in, 570  
    out, 570  
usb\_lld.h, 570  
usb\_lld\_clear\_in  
    USB Driver, 274  
usb\_lld\_clear\_out  
    USB Driver, 274  
usb\_lld\_disable\_endpoints  
    USB Driver, 270  
usb\_lld\_fetch\_word  
    USB Driver, 284  
usb\_lld\_get\_frame\_number  
    USB Driver, 284  
usb\_lld\_get\_packet\_size  
    USB Driver, 285  
usb\_lld\_get\_status\_in  
    USB Driver, 270  
usb\_lld\_get\_status\_out

USB Driver, 270  
usb\_lld\_get\_transaction\_size  
    USB Driver, 284  
usb\_lld\_init  
    USB Driver, 268  
usb\_lld\_init\_endpoint  
    USB Driver, 270  
usb\_lld\_prepare\_receive  
    USB Driver, 272  
usb\_lld\_prepare\_transmit  
    USB Driver, 272  
usb\_lld\_read\_packet\_buffer  
    USB Driver, 271  
usb\_lld\_read\_setup  
    USB Driver, 271  
usb\_lld\_reset  
    USB Driver, 269  
usb\_lld\_set\_address  
    USB Driver, 269  
usb\_lld\_stall\_in  
    USB Driver, 273  
usb\_lld\_stall\_out  
    USB Driver, 273  
usb\_lld\_start  
    USB Driver, 268  
usb\_lld\_start\_in  
    USB Driver, 273  
usb\_lld\_start\_out  
    USB Driver, 273  
usb\_lld\_stop  
    USB Driver, 269  
usb\_lld\_write\_packet\_buffer  
    USB Driver, 272  
USB\_LP\_IRQHandler  
    STM32F103 HAL Support, 323  
USB\_MAX\_ENDPOINTS  
    USB Driver, 283  
USB\_PMA\_SIZE  
    USB Driver, 283  
USB\_SET\_ADDRESS\_MODE  
    USB Driver, 283  
usbcallback\_t  
    USB Driver, 285  
USBConfig, 462  
    event\_cb, 464  
    get\_descriptor\_cb, 464  
    requests\_hook\_cb, 464  
    sof\_cb, 464  
usbConnectBus  
    USB Driver, 276  
USBD1  
    USB Driver, 274  
USBDescriptor, 464  
    ud\_size, 464  
    ud\_string, 464  
usbDisableEndpoints  
    USB Driver, 261  
usbDisconnectBus  
    USB Driver, 277  
USBDriver, 465  
    address, 467  
    config, 466  
    configuration, 467  
    ep0endcb, 467  
    ep0n, 467  
    ep0next, 466  
    ep0state, 466  
    epc, 466  
    param, 466  
    pmnext, 467  
    receiving, 466  
    setup, 467  
    state, 466  
    status, 467  
    transmitting, 466  
    USB Driver, 285  
USBEndpointConfig, 467  
    ep\_mode, 469  
    in\_cb, 469  
    in\_maxsize, 469  
    in\_state, 470  
    out\_cb, 469  
    out\_maxsize, 469  
    out\_state, 470  
    setup\_cb, 469  
usbepr0state\_t  
    USB Driver, 287  
usbept\_t  
    USB Driver, 285  
usbepcallback\_t  
    USB Driver, 286  
usbepstatus\_t  
    USB Driver, 287  
usbevent\_t  
    USB Driver, 287  
usbeventcb\_t  
    USB Driver, 286  
usbgetdescriptor\_t  
    USB Driver, 286  
usbGetFrameNumber  
    USB Driver, 277  
usbGetReceivePacketSize  
    USB Driver, 280  
usbGetReceiveStatus  
    USB Driver, 277  
usbGetReceiveTransactionSize  
    USB Driver, 279  
usbGetTransmitStatus  
    USB Driver, 277  
USBInEndpointState, 470  
    txbuf, 470  
    txcnt, 470  
    txsize, 470  
usbInit  
    USB Driver, 259  
usbInitEndpoint  
    USB Driver, 260  
usbObjectInit

USB Driver, 259  
USBOutEndpointState, 471  
  rdbuf, 471  
  rxcnt, 471  
  rxpkts, 471  
  rxsize, 471  
usbp  
  SerialUSBConfig, 448  
usbPrepareReceive  
  USB Driver, 279  
usbPrepareTransmit  
  USB Driver, 279  
usbReadPacketBuffer  
  USB Driver, 278  
usbReadSetup  
  USB Driver, 280  
usbreqhandler\_t  
  USB Driver, 286  
usbSetupTransfer  
  USB Driver, 280  
usbStallReceive1  
  USB Driver, 263  
usbStallTransmit1  
  USB Driver, 263  
usbStart  
  USB Driver, 259  
usbStartReceive1  
  USB Driver, 261  
usbStartTransmit1  
  USB Driver, 262  
usbstate\_t  
  USB Driver, 286  
usbStop  
  USB Driver, 260  
usbWritePacketBuffer  
  USB Driver, 278

vector  
  stm32\_dma\_stream\_t, 453

vmt  
  SerialDriver, 446  
  SerialUSBDriver, 448

vt  
  MMCDriver, 431

wakeup\_event  
  CANDriver, 409

wakeup\_isr  
  I2C Driver, 100

width\_cb  
  ICUConfig, 425

worst  
  TimeMeasurement, 456

WWDG\_IRQHandler  
  STM32F100 HAL Support, 299  
  STM32F103 HAL Support, 321  
  STM32F105/F107 HAL Support, 341