

ChibiOS/RT 2.4.3

STM32L1xx HAL Reference Manual

Contents

1 ChibiOS/RT	1
1.1 Copyright	1
1.2 Introduction	1
1.3 Related Documents	1
2 Deprecated List	2
3 Module Index	3
3.1 Modules	3
4 Data Structure Index	4
4.1 Data Structures	4
5 File Index	6
5.1 File List	6
6 Module Documentation	8
6.1 HAL	8
6.1.1 Detailed Description	8
6.1.2 HAL Device Drivers Architecture	8
6.1.2.1 Diagram	9
6.2 Configuration	10
6.2.1 Detailed Description	10
6.2.2 Define Documentation	12
6.2.2.1 HAL_USE_PAL	12
6.2.2.2 HAL_USE_ADC	12
6.2.2.3 HAL_USE_CAN	12
6.2.2.4 HAL_USE_EXT	12
6.2.2.5 HAL_USE_GPT	12
6.2.2.6 HAL_USE_I2C	12
6.2.2.7 HAL_USE_ICU	12
6.2.2.8 HAL_USE_MAC	13
6.2.2.9 HAL_USE_MMC_SPI	13

6.2.2.10 HAL_USE_PWM	13
6.2.2.11 HAL_USE_RTC	13
6.2.2.12 HAL_USE_SDC	13
6.2.2.13 HAL_USE_SERIAL	13
6.2.2.14 HAL_USE_SERIAL_USB	13
6.2.2.15 HAL_USE_SPI	13
6.2.2.16 HAL_USE_UART	13
6.2.2.17 HAL_USE_USB	13
6.2.2.18 ADC_USE_WAIT	13
6.2.2.19 ADC_USE_MUTUAL_EXCLUSION	14
6.2.2.20 CAN_USE_SLEEP_MODE	14
6.2.2.21 I2C_USE_MUTUAL_EXCLUSION	14
6.2.2.22 MAC_USE_EVENTS	14
6.2.2.23 MMC_SECTOR_SIZE	14
6.2.2.24 MMC_NICE_WAITING	14
6.2.2.25 MMC_POLLING_INTERVAL	14
6.2.2.26 MMC_POLLING_DELAY	14
6.2.2.27 MMC_USE_SPI_POLLING	14
6.2.2.28 SDC_INIT_RETRY	14
6.2.2.29 SDC_MMC_SUPPORT	15
6.2.2.30 SDC_NICE_WAITING	15
6.2.2.31 SERIAL_DEFAULT_BITRATE	15
6.2.2.32 SERIAL_BUFFERS_SIZE	15
6.2.2.33 SERIAL_USB_BUFFERS_SIZE	15
6.2.2.34 SPI_USE_WAIT	15
6.2.2.35 SPI_USE_MUTUAL_EXCLUSION	15
6.3 ADC Driver	16
6.3.1 Detailed Description	16
6.3.2 Driver State Machine	16
6.3.3 ADC Operations	16
6.3.3.1 ADC Conversion Groups	16
6.3.3.2 ADC Conversion Modes	17
6.3.3.3 ADC Callbacks	17
6.3.4 Function Documentation	23
6.3.4.1 adcInit	23
6.3.4.2 adcObjectInit	24
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	27
6.3.4.9	adcAcquireBus	27
6.3.4.10	adcReleaseBus	28
6.3.4.11	adcConvert	28
6.3.4.12	CH_IRQ_HANDLER	29
6.3.4.13	adc_lld_init	29
6.3.4.14	adc_lld_start	29
6.3.4.15	adc_lld_stop	29
6.3.4.16	adc_lld_start_conversion	30
6.3.4.17	adc_lld_stop_conversion	30
6.3.4.18	adcSTM32EnableTSVREFE	30
6.3.4.19	adcSTM32DisableTSVREFE	30
6.3.5	Variable Documentation	31
6.3.5.1	ADCD1	31
6.3.6	Define Documentation	31
6.3.6.1	ADC_USE_WAIT	31
6.3.6.2	ADC_USE_MUTUAL_EXCLUSION	31
6.3.6.3	_adc_reset_i	31
6.3.6.4	_adc_reset_s	31
6.3.6.5	_adc_wakeup_isr	32
6.3.6.6	_adc_timeout_isr	32
6.3.6.7	_adc_isr_half_code	33
6.3.6.8	_adc_isr_full_code	33
6.3.6.9	_adc_isr_error_code	34
6.3.6.10	ADC_CR2_EXTSEL_SRC	34
6.3.6.11	ADC_CHANNEL_IN0	34
6.3.6.12	ADC_CHANNEL_IN1	34
6.3.6.13	ADC_CHANNEL_IN2	34
6.3.6.14	ADC_CHANNEL_IN3	34
6.3.6.15	ADC_CHANNEL_IN4	35
6.3.6.16	ADC_CHANNEL_IN5	35
6.3.6.17	ADC_CHANNEL_IN6	35
6.3.6.18	ADC_CHANNEL_IN7	35
6.3.6.19	ADC_CHANNEL_IN8	35
6.3.6.20	ADC_CHANNEL_IN9	35
6.3.6.21	ADC_CHANNEL_IN10	35
6.3.6.22	ADC_CHANNEL_IN11	35
6.3.6.23	ADC_CHANNEL_IN12	35
6.3.6.24	ADC_CHANNEL_IN13	35

6.3.6.25	ADC_CHANNEL_IN14	35
6.3.6.26	ADC_CHANNEL_IN15	35
6.3.6.27	ADC_CHANNEL_SENSOR	36
6.3.6.28	ADC_CHANNEL_VREFINT	36
6.3.6.29	ADC_CHANNEL_IN18	36
6.3.6.30	ADC_CHANNEL_IN19	36
6.3.6.31	ADC_CHANNEL_IN20	36
6.3.6.32	ADC_CHANNEL_IN21	36
6.3.6.33	ADC_CHANNEL_IN22	36
6.3.6.34	ADC_CHANNEL_IN23	36
6.3.6.35	ADC_CHANNEL_IN24	36
6.3.6.36	ADC_CHANNEL_IN25	36
6.3.6.37	ADC_SAMPLE_4	36
6.3.6.38	ADC_SAMPLE_9	36
6.3.6.39	ADC_SAMPLE_16	37
6.3.6.40	ADC_SAMPLE_24	37
6.3.6.41	ADC_SAMPLE_48	37
6.3.6.42	ADC_SAMPLE_96	37
6.3.6.43	ADC_SAMPLE_192	37
6.3.6.44	ADC_SAMPLE_384	37
6.3.6.45	STM32_ADC_USE_ADC1	37
6.3.6.46	STM32_ADC_ADCPRE	37
6.3.6.47	STM32_ADC_ADC1_DMA_PRIORITY	37
6.3.6.48	STM32_ADC_IRQ_PRIORITY	37
6.3.6.49	STM32_ADC_ADC1_DMA_IRQ_PRIORITY	38
6.3.6.50	ADC_SQR1_NUM_CH	38
6.3.6.51	ADC_SQR5_SQ1_N	38
6.3.6.52	ADC_SQR5_SQ2_N	38
6.3.6.53	ADC_SQR5_SQ3_N	38
6.3.6.54	ADC_SQR5_SQ4_N	38
6.3.6.55	ADC_SQR5_SQ5_N	38
6.3.6.56	ADC_SQR5_SQ6_N	38
6.3.6.57	ADC_SQR4_SQ7_N	38
6.3.6.58	ADC_SQR4_SQ8_N	38
6.3.6.59	ADC_SQR4_SQ9_N	38
6.3.6.60	ADC_SQR4_SQ10_N	38
6.3.6.61	ADC_SQR4_SQ11_N	39
6.3.6.62	ADC_SQR4_SQ12_N	39
6.3.6.63	ADC_SQR3_SQ13_N	39
6.3.6.64	ADC_SQR3_SQ14_N	39

6.3.6.65	ADC_SQR3_SQ15_N	39
6.3.6.66	ADC_SQR3_SQ16_N	39
6.3.6.67	ADC_SQR3_SQ17_N	39
6.3.6.68	ADC_SQR3_SQ18_N	39
6.3.6.69	ADC_SQR2_SQ19_N	39
6.3.6.70	ADC_SQR2_SQ20_N	39
6.3.6.71	ADC_SQR2_SQ21_N	39
6.3.6.72	ADC_SQR2_SQ22_N	39
6.3.6.73	ADC_SQR2_SQ23_N	40
6.3.6.74	ADC_SQR2_SQ24_N	40
6.3.6.75	ADC_SQR1_SQ25_N	40
6.3.6.76	ADC_SQR1_SQ26_N	40
6.3.6.77	ADC_SQR1_SQ27_N	40
6.3.6.78	ADC_SMPR3_SMP_AN0	40
6.3.6.79	ADC_SMPR3_SMP_AN1	40
6.3.6.80	ADC_SMPR3_SMP_AN2	40
6.3.6.81	ADC_SMPR3_SMP_AN3	40
6.3.6.82	ADC_SMPR3_SMP_AN4	40
6.3.6.83	ADC_SMPR3_SMP_AN5	40
6.3.6.84	ADC_SMPR3_SMP_AN6	40
6.3.6.85	ADC_SMPR3_SMP_AN7	41
6.3.6.86	ADC_SMPR3_SMP_AN8	41
6.3.6.87	ADC_SMPR3_SMP_AN9	41
6.3.6.88	ADC_SMPR2_SMP_AN10	41
6.3.6.89	ADC_SMPR2_SMP_AN11	41
6.3.6.90	ADC_SMPR2_SMP_AN12	41
6.3.6.91	ADC_SMPR2_SMP_AN13	41
6.3.6.92	ADC_SMPR2_SMP_AN14	41
6.3.6.93	ADC_SMPR2_SMP_AN15	41
6.3.6.94	ADC_SMPR2_SMP_SENSOR	41
6.3.6.95	ADC_SMPR2_SMP_VREF	41
6.3.6.96	ADC_SMPR2_SMP_AN18	41
6.3.6.97	ADC_SMPR2_SMP_AN19	42
6.3.6.98	ADC_SMPR1_SMP_AN20	42
6.3.6.99	ADC_SMPR1_SMP_AN21	42
6.3.6.100	ADC_SMPR1_SMP_AN22	42
6.3.6.101	ADC_SMPR1_SMP_AN23	42
6.3.6.102	ADC_SMPR1_SMP_AN24	42
6.3.6.103	ADC_SMPR1_SMP_AN25	42
6.3.7	Typedef Documentation	42

6.3.7.1	adcsample_t	42
6.3.7.2	adc_channels_num_t	42
6.3.7.3	ADCDriver	42
6.3.7.4	adccallback_t	42
6.3.7.5	adcerrorcallback_t	43
6.3.8	Enumeration Type Documentation	43
6.3.8.1	adcstate_t	43
6.3.8.2	adcerror_t	43
6.4	EXT Driver	43
6.4.1	Detailed Description	43
6.4.2	Driver State Machine	43
6.4.3	EXT Operations.	44
6.4.4	Function Documentation	47
6.4.4.1	extInit	47
6.4.4.2	extObjectInit	47
6.4.4.3	extStart	47
6.4.4.4	extStop	48
6.4.4.5	extChannelEnable	48
6.4.4.6	extChannelDisable	48
6.4.4.7	CH_IRQ_HANDLER	49
6.4.4.8	CH_IRQ_HANDLER	49
6.4.4.9	CH_IRQ_HANDLER	49
6.4.4.10	CH_IRQ_HANDLER	49
6.4.4.11	CH_IRQ_HANDLER	49
6.4.4.12	CH_IRQ_HANDLER	49
6.4.4.13	CH_IRQ_HANDLER	50
6.4.4.14	CH_IRQ_HANDLER	50
6.4.4.15	CH_IRQ_HANDLER	50
6.4.4.16	CH_IRQ_HANDLER	50
6.4.4.17	ext_lld_init	50
6.4.4.18	ext_lld_start	51
6.4.4.19	ext_lld_stop	51
6.4.4.20	ext_lld_channel_enable	51
6.4.4.21	ext_lld_channel_disable	51
6.4.5	Variable Documentation	51
6.4.5.1	EXTD1	51
6.4.6	Define Documentation	52
6.4.6.1	EXT_MAX_CHANNELS	52
6.4.6.2	EXT_CHANNELS_MASK	52
6.4.6.3	EXT_MODE_EXTI	52

6.4.6.4	EXT_MODE_GPIOA	52
6.4.6.5	EXT_MODE_GPIOB	52
6.4.6.6	EXT_MODE_GPIOC	52
6.4.6.7	EXT_MODE_GPIOD	52
6.4.6.8	EXT_MODE_GPIOE	52
6.4.6.9	EXT_MODE_GPIOF	52
6.4.6.10	EXT_MODE_GPIOG	52
6.4.6.11	EXT_MODE_GPIOH	53
6.4.6.12	EXT_MODE_GPIOI	53
6.4.6.13	STM32_EXT EXTI0_IRQ_PRIORITY	53
6.4.6.14	STM32_EXT EXTI1_IRQ_PRIORITY	53
6.4.6.15	STM32_EXT EXTI2_IRQ_PRIORITY	53
6.4.6.16	STM32_EXT EXTI3_IRQ_PRIORITY	53
6.4.6.17	STM32_EXT EXTI4_IRQ_PRIORITY	53
6.4.6.18	STM32_EXT EXTI5_9_IRQ_PRIORITY	53
6.4.6.19	STM32_EXT EXTI10_15_IRQ_PRIORITY	53
6.4.6.20	STM32_EXT EXTI16_IRQ_PRIORITY	53
6.4.6.21	STM32_EXT EXTI17_IRQ_PRIORITY	53
6.4.6.22	STM32_EXT EXTI18_IRQ_PRIORITY	53
6.4.6.23	STM32_EXT EXTI19_IRQ_PRIORITY	54
6.4.6.24	STM32_EXT EXTI20_IRQ_PRIORITY	54
6.4.6.25	STM32_EXT EXTI21_IRQ_PRIORITY	54
6.4.6.26	STM32_EXT EXTI22_IRQ_PRIORITY	54
6.4.7	Typedef Documentation	54
6.4.7.1	expchannel_t	54
6.4.7.2	extcallback_t	54
6.5	GPT Driver	54
6.5.1	Detailed Description	54
6.5.2	Driver State Machine	54
6.5.3	GPT Operations.	55
6.5.4	Function Documentation	58
6.5.4.1	gptInit	58
6.5.4.2	gptObjectInit	58
6.5.4.3	gptStart	58
6.5.4.4	gptStop	59
6.5.4.5	gptStartContinuous	59
6.5.4.6	gptStartContinuous!	60
6.5.4.7	gptStartOneShot	60
6.5.4.8	gptStartOneShot!	61
6.5.4.9	gptStopTimer	61

6.5.4.10	gptStopTimerl	62
6.5.4.11	gptPolledDelay	62
6.5.4.12	gpt_lld_init	63
6.5.4.13	gpt_lld_start	63
6.5.4.14	gpt_lld_stop	63
6.5.4.15	gpt_lld_start_timer	64
6.5.4.16	gpt_lld_stop_timer	64
6.5.4.17	gpt_lld_polled_delay	64
6.5.5	Variable Documentation	64
6.5.5.1	GPTD1	64
6.5.5.2	GPTD2	64
6.5.5.3	GPTD3	65
6.5.5.4	GPTD4	65
6.5.5.5	GPTD5	65
6.5.5.6	GPTD8	65
6.5.6	Define Documentation	65
6.5.6.1	STM32_GPT_USE_TIM1	65
6.5.6.2	STM32_GPT_USE_TIM2	65
6.5.6.3	STM32_GPT_USE_TIM3	66
6.5.6.4	STM32_GPT_USE_TIM4	66
6.5.6.5	STM32_GPT_USE_TIM5	66
6.5.6.6	STM32_GPT_USE_TIM8	66
6.5.6.7	STM32_GPT_TIM1_IRQ_PRIORITY	66
6.5.6.8	STM32_GPT_TIM2_IRQ_PRIORITY	66
6.5.6.9	STM32_GPT_TIM3_IRQ_PRIORITY	66
6.5.6.10	STM32_GPT_TIM4_IRQ_PRIORITY	66
6.5.6.11	STM32_GPT_TIM5_IRQ_PRIORITY	67
6.5.6.12	STM32_GPT_TIM8_IRQ_PRIORITY	67
6.5.7	Typedef Documentation	67
6.5.7.1	GPTDriver	67
6.5.7.2	gptcallback_t	67
6.5.7.3	gptfreq_t	67
6.5.7.4	gptcnt_t	67
6.5.8	Enumeration Type Documentation	67
6.5.8.1	gptstate_t	67
6.6	HAL Driver	67
6.6.1	Detailed Description	67
6.6.2	Function Documentation	76
6.6.2.1	hallInit	76
6.6.2.2	hallsCounterWithin	77

6.6.2.3	halPolledDelay	78
6.6.2.4	hal_lld_init	79
6.6.2.5	stm32_clock_init	79
6.6.3	Define Documentation	79
6.6.3.1	S2RTT	79
6.6.3.2	MS2RTT	80
6.6.3.3	US2RTT	80
6.6.3.4	halGetCounterValue	81
6.6.3.5	halGetCounterFrequency	81
6.6.3.6	HAL_IMPLEMENTS_COUNTERS	81
6.6.3.7	STM32_HSICLK	81
6.6.3.8	STM32_LSICLK	81
6.6.3.9	STM32_VOS_MASK	81
6.6.3.10	STM32_VOS_1P8	81
6.6.3.11	STM32_VOS_1P5	82
6.6.3.12	STM32_VOS_1P2	82
6.6.3.13	STM32_PLS_MASK	82
6.6.3.14	STM32_PLS_LEV0	82
6.6.3.15	STM32_PLS_LEV1	82
6.6.3.16	STM32_PLS_LEV2	82
6.6.3.17	STM32_PLS_LEV3	82
6.6.3.18	STM32_PLS_LEV4	82
6.6.3.19	STM32_PLS_LEV5	82
6.6.3.20	STM32_PLS_LEV6	82
6.6.3.21	STM32_PLS_LEV7	82
6.6.3.22	STM32_RTCPRE_MASK	82
6.6.3.23	STM32_RTCPRE_DIV2	83
6.6.3.24	STM32_RTCPRE_DIV4	83
6.6.3.25	STM32_RTCPRE_DIV8	83
6.6.3.26	STM32_RTCPRE_DIV16	83
6.6.3.27	STM32_SW_MSI	83
6.6.3.28	STM32_SW_HSI	83
6.6.3.29	STM32_SW_HSE	83
6.6.3.30	STM32_SW_PLL	83
6.6.3.31	STM32_HPRE_DIV1	83
6.6.3.32	STM32_HPRE_DIV2	83
6.6.3.33	STM32_HPRE_DIV4	83
6.6.3.34	STM32_HPRE_DIV8	83
6.6.3.35	STM32_HPRE_DIV16	84
6.6.3.36	STM32_HPRE_DIV64	84

6.6.3.37	STM32_HPRE_DIV128	84
6.6.3.38	STM32_HPRE_DIV256	84
6.6.3.39	STM32_HPRE_DIV512	84
6.6.3.40	STM32_PPRE1_DIV1	84
6.6.3.41	STM32_PPRE1_DIV2	84
6.6.3.42	STM32_PPRE1_DIV4	84
6.6.3.43	STM32_PPRE1_DIV8	84
6.6.3.44	STM32_PPRE1_DIV16	84
6.6.3.45	STM32_PPRE2_DIV1	84
6.6.3.46	STM32_PPRE2_DIV2	84
6.6.3.47	STM32_PPRE2_DIV4	85
6.6.3.48	STM32_PPRE2_DIV8	85
6.6.3.49	STM32_PPRE2_DIV16	85
6.6.3.50	STM32_PLLSRC_HSI	85
6.6.3.51	STM32_PLLSRC_HSE	85
6.6.3.52	STM32_MCOSEL_NOCLOCK	85
6.6.3.53	STM32_MCOSEL_SYSCLK	85
6.6.3.54	STM32_MCOSEL_HSI	85
6.6.3.55	STM32_MCOSEL_MSI	85
6.6.3.56	STM32_MCOSEL_HSE	85
6.6.3.57	STM32_MCOSEL_PLL	85
6.6.3.58	STM32_MCOSEL_LSI	85
6.6.3.59	STM32_MCOSEL_LSE	86
6.6.3.60	STM32_MCOPRE_DIV1	86
6.6.3.61	STM32_MCOPRE_DIV2	86
6.6.3.62	STM32_MCOPRE_DIV4	86
6.6.3.63	STM32_MCOPRE_DIV8	86
6.6.3.64	STM32_MCOPRE_DIV16	86
6.6.3.65	STM32_MSIRANGE_MASK	86
6.6.3.66	STM32_MSIRANGE_64K	86
6.6.3.67	STM32_MSIRANGE_128K	86
6.6.3.68	STM32_MSIRANGE_256K	86
6.6.3.69	STM32_MSIRANGE_512K	86
6.6.3.70	STM32_MSIRANGE_1M	86
6.6.3.71	STM32_MSIRANGE_2M	87
6.6.3.72	STM32_MSIRANGE_4M	87
6.6.3.73	STM32_RTCSEL_MASK	87
6.6.3.74	STM32_RTCSEL_NOCLOCK	87
6.6.3.75	STM32_RTCSEL_LSE	87
6.6.3.76	STM32_RTCSEL_LSI	87

6.6.3.77	STM32_RTCSEL_HSEDIV	87
6.6.3.78	WWDG_IRQHandler	87
6.6.3.79	PVD_IRQHandler	87
6.6.3.80	TAMPER_STAMP_IRQHandler	87
6.6.3.81	RTC_WKUP_IRQHandler	87
6.6.3.82	FLASH_IRQHandler	87
6.6.3.83	RCC_IRQHandler	88
6.6.3.84	EXTI0_IRQHandler	88
6.6.3.85	EXTI1_IRQHandler	88
6.6.3.86	EXTI2_IRQHandler	88
6.6.3.87	EXTI3_IRQHandler	88
6.6.3.88	EXTI4_IRQHandler	88
6.6.3.89	DMA1_Ch1_IRQHandler	88
6.6.3.90	DMA1_Ch2_IRQHandler	88
6.6.3.91	DMA1_Ch3_IRQHandler	88
6.6.3.92	DMA1_Ch4_IRQHandler	88
6.6.3.93	DMA1_Ch5_IRQHandler	88
6.6.3.94	DMA1_Ch6_IRQHandler	88
6.6.3.95	DMA1_Ch7_IRQHandler	89
6.6.3.96	ADC1_IRQHandler	89
6.6.3.97	USB_HP_IRQHandler	89
6.6.3.98	USB_LP_IRQHandler	89
6.6.3.99	DAC_IRQHandler	89
6.6.3.100	COMP_IRQHandler	89
6.6.3.101	EXTI9_5_IRQHandler	89
6.6.3.102	TIM9_IRQHandler	89
6.6.3.103	TIM10_IRQHandler	89
6.6.3.104	TIM11_IRQHandler	89
6.6.3.105	LCD_IRQHandler	89
6.6.3.106	TIM2_IRQHandler	89
6.6.3.107	TIM3_IRQHandler	90
6.6.3.108	TIM4_IRQHandler	90
6.6.3.109	I2C1_EV_IRQHandler	90
6.6.3.110	I2C1_ER_IRQHandler	90
6.6.3.111	I2C2_EV_IRQHandler	90
6.6.3.112	I2C2_ER_IRQHandler	90
6.6.3.113	SPI1_IRQHandler	90
6.6.3.114	SPI2_IRQHandler	90
6.6.3.115	USART1_IRQHandler	90
6.6.3.116	USART2_IRQHandler	90

6.6.3.117 USART3_IRQHandler	90
6.6.3.118 EXTI15_10_IRQHandler	90
6.6.3.119 RTC_Alarm_IRQHandler	91
6.6.3.120 USB_FS_WKUP_IRQHandler	91
6.6.3.121 TIM6_IRQHandler	91
6.6.3.122 TIM7_IRQHandler	91
6.6.3.123 STM32_NO_INIT	91
6.6.3.124 STM32_VOS	91
6.6.3.125 STM32_PVD_ENABLE	91
6.6.3.126 STM32_PLS	91
6.6.3.127 STM32_HSI_ENABLED	91
6.6.3.128 STM32_LSI_ENABLED	91
6.6.3.129 STM32_HSE_ENABLED	91
6.6.3.130 STM32_LSE_ENABLED	92
6.6.3.131 STM32_ADC_CLOCK_ENABLED	92
6.6.3.132 STM32_USB_CLOCK_ENABLED	92
6.6.3.133 STM32_MSIRANGE	92
6.6.3.134 STM32_SW	92
6.6.3.135 STM32_PLLSRC	92
6.6.3.136 STM32_PLLMUL_VALUE	92
6.6.3.137 STM32_PLLDIV_VALUE	92
6.6.3.138 STM32_HPRE	93
6.6.3.139 STM32_PPREG1	93
6.6.3.140 STM32_PPREG2	93
6.6.3.141 STM32_MCOSEL	93
6.6.3.142 STM32_MCOPRE	93
6.6.3.143 STM32_RTCSEL	93
6.6.3.144 STM32_RTCPRE	93
6.6.3.145 STM32_HSECLK_MAX	93
6.6.3.146 STM32_SYSCLK_MAX	93
6.6.3.147 STM32_PLLVCO_MAX	93
6.6.3.148 STM32_PLLVCO_MIN	93
6.6.3.149 STM32_PCLK1_MAX	94
6.6.3.150 STM32_PCLK2_MAX	94
6.6.3.151 STM32_0WS_THRESHOLD	94
6.6.3.152 STM32_HSI_AVAILABLE	94
6.6.3.153 STM32_ACTIVATE_PLL	94
6.6.3.154 STM32_PLLMUL	94
6.6.3.155 STM32_PLLDIV	94
6.6.3.156 STM32_PLLCLKIN	94

6.6.3.157 STM32_PLLVCO	94
6.6.3.158 STM32_PLLCLKOUT	94
6.6.3.159 STM32_MSICLK	94
6.6.3.160 STM32_SYSCLK	95
6.6.3.161 STM32_HCLK	95
6.6.3.162 STM32_PCLK1	95
6.6.3.163 STM32_PCLK2	95
6.6.3.164 STM_MCODIVCLK	95
6.6.3.165 STM_MCOCLK	95
6.6.3.166 STM32_HSEDIVCLK	95
6.6.3.167 STM_RTCCLK	95
6.6.3.168 STM32_ADCCLK	95
6.6.3.169 STM32_USBCLK	95
6.6.3.170 STM32_TIMCLK1	95
6.6.3.171 STM32_TIMCLK2	95
6.6.3.172 STM32_FLASHBITS1	96
6.6.3.173 hal_lld_get_counter_value	96
6.6.3.174 hal_lld_get_counter_frequency	96
6.6.4 Typedef Documentation	96
6.6.4.1 halclock_t	96
6.6.4.2 halrtcnt_t	96
6.7 I2C Driver	96
6.7.1 Detailed Description	96
6.7.2 Driver State Machine	97
6.7.3 Function Documentation	100
6.7.3.1 i2cInit	100
6.7.3.2 i2cObjectInit	101
6.7.3.3 i2cStart	101
6.7.3.4 i2cStop	101
6.7.3.5 i2cGetErrors	102
6.7.3.6 i2cMasterTransmitTimeout	102
6.7.3.7 i2cMasterReceiveTimeout	103
6.7.3.8 i2cAcquireBus	104
6.7.3.9 i2cReleaseBus	104
6.7.3.10 CH_IRQ_HANDLER	104
6.7.3.11 CH_IRQ_HANDLER	105
6.7.3.12 CH_IRQ_HANDLER	105
6.7.3.13 CH_IRQ_HANDLER	105
6.7.3.14 CH_IRQ_HANDLER	105
6.7.3.15 CH_IRQ_HANDLER	105

6.7.3.16	i2c_lld_init	105
6.7.3.17	i2c_lld_start	106
6.7.3.18	i2c_lld_stop	106
6.7.3.19	i2c_lld_master_receive_timeout	106
6.7.3.20	i2c_lld_master_transmit_timeout	107
6.7.4	Variable Documentation	108
6.7.4.1	I2CD1	108
6.7.4.2	I2CD2	108
6.7.4.3	I2CD3	108
6.7.5	Define Documentation	108
6.7.5.1	I2CD_NO_ERROR	108
6.7.5.2	I2CD_BUS_ERROR	108
6.7.5.3	I2CD_ARBITRATION_LOST	108
6.7.5.4	I2CD_ACK_FAILURE	108
6.7.5.5	I2CD_OVERRUN	108
6.7.5.6	I2CD_PEC_ERROR	108
6.7.5.7	I2CD_TIMEOUT	108
6.7.5.8	I2CD_SMB_ALERT	108
6.7.5.9	I2C_USE_MUTUAL_EXCLUSION	109
6.7.5.10	i2cMasterTransmit	109
6.7.5.11	i2cMasterReceive	109
6.7.5.12	wakeup_isr	109
6.7.5.13	I2C_CLK_FREQ	109
6.7.5.14	STM32_I2C_USE_I2C1	110
6.7.5.15	STM32_I2C_USE_I2C2	110
6.7.5.16	STM32_I2C_USE_I2C3	110
6.7.5.17	STM32_I2C_I2C1_IRQ_PRIORITY	110
6.7.5.18	STM32_I2C_I2C2_IRQ_PRIORITY	110
6.7.5.19	STM32_I2C_I2C3_IRQ_PRIORITY	110
6.7.5.20	STM32_I2C_I2C1_DMA_PRIORITY	110
6.7.5.21	STM32_I2C_I2C2_DMA_PRIORITY	111
6.7.5.22	STM32_I2C_I2C3_DMA_PRIORITY	111
6.7.5.23	STM32_I2C_DMA_ERROR_HOOK	111
6.7.5.24	STM32_I2C_I2C1_RX_DMA_STREAM	111
6.7.5.25	STM32_I2C_I2C1_TX_DMA_STREAM	111
6.7.5.26	STM32_I2C_I2C2_RX_DMA_STREAM	111
6.7.5.27	STM32_I2C_I2C2_TX_DMA_STREAM	112
6.7.5.28	STM32_I2C_I2C3_RX_DMA_STREAM	112
6.7.5.29	STM32_I2C_I2C3_TX_DMA_STREAM	112
6.7.5.30	STM32_DMA_REQUIRED	112

6.7.5.31	i2c_lld_get_errors	112
6.7.6	Typedef Documentation	112
6.7.6.1	i2caddr_t	112
6.7.6.2	i2cflags_t	112
6.7.6.3	I2CDriver	112
6.7.7	Enumeration Type Documentation	113
6.7.7.1	i2cstate_t	113
6.7.7.2	i2copmode_t	113
6.7.7.3	i2cdutycycle_t	113
6.8	ICU Driver	113
6.8.1	Detailed Description	113
6.8.2	Driver State Machine	113
6.8.3	ICU Operations.	114
6.8.4	Function Documentation	117
6.8.4.1	icuInit	117
6.8.4.2	icuObjectInit	117
6.8.4.3	icuStart	117
6.8.4.4	icuStop	118
6.8.4.5	icuEnable	118
6.8.4.6	icuDisable	119
6.8.4.7	icu_lld_init	119
6.8.4.8	icu_lld_start	120
6.8.4.9	icu_lld_stop	120
6.8.4.10	icu_lld_enable	120
6.8.4.11	icu_lld_disable	120
6.8.5	Variable Documentation	120
6.8.5.1	ICUD1	120
6.8.5.2	ICUD2	121
6.8.5.3	ICUD3	121
6.8.5.4	ICUD4	121
6.8.5.5	ICUD5	121
6.8.5.6	ICUD8	121
6.8.6	Define Documentation	121
6.8.6.1	icuEnablel	121
6.8.6.2	icuDisablel	122
6.8.6.3	icuGetWidthl	122
6.8.6.4	icuGetPeriodl	122
6.8.6.5	_icu_isr_invoke_width_cb	122
6.8.6.6	_icu_isr_invoke_period_cb	123
6.8.6.7	STM32_ICU_USE_TIM1	123

6.8.6.8	STM32_ICU_USE_TIM2	123
6.8.6.9	STM32_ICU_USE_TIM3	123
6.8.6.10	STM32_ICU_USE_TIM4	124
6.8.6.11	STM32_ICU_USE_TIM5	124
6.8.6.12	STM32_ICU_USE_TIM8	124
6.8.6.13	STM32_ICU_TIM1_IRQ_PRIORITY	124
6.8.6.14	STM32_ICU_TIM2_IRQ_PRIORITY	124
6.8.6.15	STM32_ICU_TIM3_IRQ_PRIORITY	124
6.8.6.16	STM32_ICU_TIM4_IRQ_PRIORITY	124
6.8.6.17	STM32_ICU_TIM5_IRQ_PRIORITY	124
6.8.6.18	STM32_ICU_TIM8_IRQ_PRIORITY	124
6.8.6.19	icu_lld_get_width	125
6.8.6.20	icu_lld_get_period	125
6.8.7	Typedef Documentation	125
6.8.7.1	ICUDriver	125
6.8.7.2	icucallback_t	125
6.8.7.3	icufreq_t	125
6.8.7.4	icucnt_t	125
6.8.8	Enumeration Type Documentation	126
6.8.8.1	icustate_t	126
6.8.8.2	icumode_t	126
6.9	MMC over SPI Driver	126
6.9.1	Detailed Description	126
6.9.2	Driver State Machine	126
6.9.3	Function Documentation	128
6.9.3.1	mmcInit	128
6.9.3.2	mmcObjectInit	129
6.9.3.3	mmcStart	129
6.9.3.4	mmcStop	129
6.9.3.5	mmcConnect	130
6.9.3.6	mmcDisconnect	130
6.9.3.7	mmcStartSequentialRead	131
6.9.3.8	mmcSequentialRead	132
6.9.3.9	mmcStopSequentialRead	132
6.9.3.10	mmcStartSequentialWrite	133
6.9.3.11	mmcSequentialWrite	134
6.9.3.12	mmcStopSequentialWrite	134
6.9.4	Define Documentation	135
6.9.4.1	MMC_SECTOR_SIZE	135
6.9.4.2	MMC_NICE_WAITING	135

6.9.4.3	MMC_POLLING_INTERVAL	135
6.9.4.4	MMC_POLLING_DELAY	135
6.9.4.5	mmcGetDriverState	135
6.9.4.6	mmcIsWriteProtected	136
6.9.5	Typedef Documentation	136
6.9.5.1	mmcquery_t	136
6.9.6	Enumeration Type Documentation	136
6.9.6.1	mmcstate_t	136
6.10	PAL Driver	136
6.10.1	Detailed Description	136
6.10.2	Implementation Rules	137
6.10.2.1	Writing on input pads	137
6.10.2.2	Reading from output pads	137
6.10.2.3	Writing unused or unimplemented port bits	137
6.10.2.4	Reading from unused or unimplemented port bits	137
6.10.2.5	Reading or writing on pins associated to other functionalities	137
6.10.3	Function Documentation	141
6.10.3.1	palReadBus	141
6.10.3.2	palWriteBus	142
6.10.3.3	palSetBusMode	142
6.10.3.4	_pal_lld_init	142
6.10.3.5	_pal_lld_setgroupmode	143
6.10.4	Define Documentation	143
6.10.4.1	PAL_MODE_RESET	143
6.10.4.2	PAL_MODE_UNCONNECTED	143
6.10.4.3	PAL_MODE_INPUT	143
6.10.4.4	PAL_MODE_INPUT_PULLUP	143
6.10.4.5	PAL_MODE_INPUT_PULLDOWN	143
6.10.4.6	PAL_MODE_INPUT_ANALOG	143
6.10.4.7	PAL_MODE_OUTPUT_PUSH_PULL	144
6.10.4.8	PAL_MODE_OUTPUT_OPENDRAIN	144
6.10.4.9	PAL_LOW	144
6.10.4.10	PAL_HIGH	144
6.10.4.11	PAL_PORT_BIT	144
6.10.4.12	PAL_GROUP_MASK	144
6.10.4.13	_IOBUS_DATA	144
6.10.4.14	IOBUS_DECL	145
6.10.4.15	pallInit	145
6.10.4.16	palReadPort	145
6.10.4.17	palReadLatch	145

6.10.4.18 palWritePort	146
6.10.4.19 palSetPort	146
6.10.4.20 palClearPort	146
6.10.4.21 palTogglePort	147
6.10.4.22 palReadGroup	147
6.10.4.23 palWriteGroup	147
6.10.4.24 palSetGroupMode	148
6.10.4.25 palReadPad	148
6.10.4.26 palWritePad	149
6.10.4.27 palSetPad	149
6.10.4.28 palClearPad	149
6.10.4.29 palTogglePad	150
6.10.4.30 palSetPadMode	150
6.10.4.31 PAL_MODE_ALTERNATE	150
6.10.4.32 PAL_MODE_RESET	151
6.10.4.33 PAL_MODE_UNCONNECTED	151
6.10.4.34 PAL_MODE_INPUT	151
6.10.4.35 PAL_MODE_INPUT_PULLUP	151
6.10.4.36 PAL_MODE_INPUT_PULLDOWN	151
6.10.4.37 PAL_MODE_INPUT_ANALOG	151
6.10.4.38 PAL_MODE_OUTPUT_PUSH_PULL	151
6.10.4.39 PAL_MODE_OUTPUT_OPENDRAIN	151
6.10.4.40 PAL_IOPORTS_WIDTH	152
6.10.4.41 PAL_WHOLE_PORT	152
6.10.4.42 IOPORT1	152
6.10.4.43 IOPORT2	152
6.10.4.44 IOPORT3	152
6.10.4.45 IOPORT4	152
6.10.4.46 IOPORT5	152
6.10.4.47 IOPORT6	152
6.10.4.48 IOPORT7	152
6.10.4.49 IOPORT8	152
6.10.4.50 IOPORT9	152
6.10.4.51 pal_lld_init	153
6.10.4.52 pal_lld_readport	153
6.10.4.53 pal_lld_readlatch	153
6.10.4.54 pal_lld_writeport	153
6.10.4.55 pal_lld_setport	154
6.10.4.56 pal_lld_clearport	154
6.10.4.57 pal_lld_writegroup	154

6.10.4.58	pal_lld_setgroupmode	154
6.10.4.59	pal_lld_writepad	155
6.10.5	Typedef Documentation	155
6.10.5.1	ioportmask_t	155
6.10.5.2	iomode_t	155
6.10.5.3	ioportid_t	155
6.11	PWM Driver	155
6.11.1	Detailed Description	155
6.11.2	Driver State Machine	156
6.11.3	PWM Operations	156
6.11.4	Function Documentation	159
6.11.4.1	pwmInit	159
6.11.4.2	pwmObjectInit	160
6.11.4.3	pwmStart	160
6.11.4.4	pwmStop	160
6.11.4.5	pwmChangePeriod	161
6.11.4.6	pwmEnableChannel	161
6.11.4.7	pwmDisableChannel	162
6.11.4.8	pwm_lld_init	163
6.11.4.9	pwm_lld_start	163
6.11.4.10	pwm_lld_stop	163
6.11.4.11	pwm_lld_enable_channel	164
6.11.4.12	pwm_lld_disable_channel	164
6.11.5	Variable Documentation	164
6.11.5.1	PWMD1	164
6.11.5.2	PWMD2	165
6.11.5.3	PWMD3	165
6.11.5.4	PWMD4	165
6.11.5.5	PWMD5	165
6.11.5.6	PWMD8	165
6.11.6	Define Documentation	165
6.11.6.1	PWM_OUTPUT_MASK	165
6.11.6.2	PWM_OUTPUT_DISABLED	165
6.11.6.3	PWM_OUTPUT_ACTIVE_HIGH	166
6.11.6.4	PWM_OUTPUT_ACTIVE_LOW	166
6.11.6.5	PWM_FRACTION_TO_WIDTH	166
6.11.6.6	PWM_DEGREES_TO_WIDTH	166
6.11.6.7	PWM_PERCENTAGE_TO_WIDTH	167
6.11.6.8	pwmChangePeriodl	167
6.11.6.9	pwmEnableChannell	168

6.11.6.10	pwmDisableChannel	168
6.11.6.11	PWM_CHANNELS	168
6.11.6.12	PWM_COMPLEMENTARY_OUTPUT_MASK	169
6.11.6.13	PWM_COMPLEMENTARY_OUTPUT_DISABLED	169
6.11.6.14	PWM_COMPLEMENTARY_OUTPUT_ACTIVE_HIGH	169
6.11.6.15	PWM_COMPLEMENTARY_OUTPUT_ACTIVE_LOW	169
6.11.6.16	STM32_PWM_USE_ADVANCED	169
6.11.6.17	STM32_PWM_USE_TIM1	169
6.11.6.18	STM32_PWM_USE_TIM2	170
6.11.6.19	STM32_PWM_USE_TIM3	170
6.11.6.20	STM32_PWM_USE_TIM4	170
6.11.6.21	STM32_PWM_USE_TIM5	170
6.11.6.22	STM32_PWM_USE_TIM8	170
6.11.6.23	STM32_PWM_TIM1_IRQ_PRIORITY	170
6.11.6.24	STM32_PWM_TIM2_IRQ_PRIORITY	170
6.11.6.25	STM32_PWM_TIM3_IRQ_PRIORITY	171
6.11.6.26	STM32_PWM_TIM4_IRQ_PRIORITY	171
6.11.6.27	STM32_PWM_TIM5_IRQ_PRIORITY	171
6.11.6.28	STM32_PWM_TIM8_IRQ_PRIORITY	171
6.11.6.29	pwm_lld_change_period	171
6.11.7	Typedef Documentation	171
6.11.7.1	PWMDriver	171
6.11.7.2	pwmcallback_t	171
6.11.7.3	pwmmode_t	172
6.11.7.4	pwmchannel_t	172
6.11.7.5	pwmcnt_t	172
6.11.8	Enumeration Type Documentation	172
6.11.8.1	pwmstate_t	172
6.12	Serial Driver	172
6.12.1	Detailed Description	172
6.12.2	Driver State Machine	172
6.12.3	Function Documentation	176
6.12.3.1	sdInit	176
6.12.3.2	sdObjectInit	177
6.12.3.3	sdStart	177
6.12.3.4	sdStop	177
6.12.3.5	sdIncomingData	178
6.12.3.6	sdRequestData	178
6.12.3.7	CH_IRQ_HANDLER	179
6.12.3.8	CH_IRQ_HANDLER	179

6.12.3.9 CH_IRQ_HANDLER	179
6.12.3.10 CH_IRQ_HANDLER	179
6.12.3.11 CH_IRQ_HANDLER	179
6.12.3.12 CH_IRQ_HANDLER	179
6.12.3.13 sd_lld_init	180
6.12.3.14 sd_lld_start	180
6.12.3.15 sd_lld_stop	180
6.12.4 Variable Documentation	180
6.12.4.1 SD1	180
6.12.4.2 SD2	181
6.12.4.3 SD3	181
6.12.4.4 SD4	181
6.12.4.5 SD5	181
6.12.4.6 SD6	181
6.12.5 Define Documentation	181
6.12.5.1 SD_PARITY_ERROR	181
6.12.5.2 SD_FRAMING_ERROR	181
6.12.5.3 SD_OVERRUN_ERROR	181
6.12.5.4 SD_NOISE_ERROR	181
6.12.5.5 SD_BREAK_DETECTED	181
6.12.5.6 SERIAL_DEFAULT_BITRATE	181
6.12.5.7 SERIAL_BUFFERS_SIZE	182
6.12.5.8 _serial_driver_methods	182
6.12.5.9 sdPutWouldBlock	182
6.12.5.10 sdGetWouldBlock	182
6.12.5.11 sdPut	183
6.12.5.12 sdPutTimeout	183
6.12.5.13 sdGet	183
6.12.5.14 sdGetTimeout	183
6.12.5.15 sdWrite	184
6.12.5.16 sdWriteTimeout	184
6.12.5.17 sdAsynchronousWrite	184
6.12.5.18 sdRead	185
6.12.5.19 sdReadTimeout	185
6.12.5.20 sdAsynchronousRead	185
6.12.5.21 STM32_SERIAL_USE_USART1	185
6.12.5.22 STM32_SERIAL_USE_USART2	186
6.12.5.23 STM32_SERIAL_USE_USART3	186
6.12.5.24 STM32_SERIAL_USE_UART4	186
6.12.5.25 STM32_SERIAL_USE_UART5	186

6.12.5.26 STM32_SERIAL_USE_USART6	186
6.12.5.27 STM32_SERIAL_USART1_PRIORITY	186
6.12.5.28 STM32_SERIAL_USART2_PRIORITY	186
6.12.5.29 STM32_SERIAL_USART3_PRIORITY	187
6.12.5.30 STM32_SERIAL_UART4_PRIORITY	187
6.12.5.31 STM32_SERIAL_UART5_PRIORITY	187
6.12.5.32 STM32_SERIAL_USART6_PRIORITY	187
6.12.5.33 _serial_driver_data	187
6.12.5.34 USART_CR2_STOP1_BITS	187
6.12.5.35 USART_CR2_STOP0P5_BITS	187
6.12.5.36 USART_CR2_STOP2_BITS	187
6.12.5.37 USART_CR2_STOP1P5_BITS	187
6.12.6 Typedef Documentation	188
6.12.6.1 SerialDriver	188
6.12.7 Enumeration Type Documentation	188
6.12.7.1 sdstate_t	188
6.13 SPI Driver	188
6.13.1 Detailed Description	188
6.13.2 Driver State Machine	188
6.13.3 Function Documentation	192
6.13.3.1 spilinit	192
6.13.3.2 spiObjectInit	192
6.13.3.3 spiStart	192
6.13.3.4 spiStop	193
6.13.3.5 spiSelect	193
6.13.3.6 spiUnselect	194
6.13.3.7 spiStartIgnore	194
6.13.3.8 spiStartExchange	194
6.13.3.9 spiStartSend	195
6.13.3.10 spiStartReceive	195
6.13.3.11 spilignore	196
6.13.3.12 spiExchange	196
6.13.3.13 spiSend	196
6.13.3.14 spiReceive	197
6.13.3.15 spiAcquireBus	197
6.13.3.16 spiReleaseBus	198
6.13.3.17 spi_lld_init	198
6.13.3.18 spi_lld_start	198
6.13.3.19 spi_lld_stop	199
6.13.3.20 spi_lld_select	199

6.13.3.21 spi_lld_unselect	199
6.13.3.22 spi_lld_ignore	200
6.13.3.23 spi_lld_exchange	200
6.13.3.24 spi_lld_send	200
6.13.3.25 spi_lld_receive	201
6.13.3.26 spi_lld_polled_exchange	201
6.13.4 Variable Documentation	202
6.13.4.1 SPID1	202
6.13.4.2 SPID2	202
6.13.4.3 SPID3	202
6.13.5 Define Documentation	202
6.13.5.1 SPI_USE_WAIT	202
6.13.5.2 SPI_USE_MUTUAL_EXCLUSION	202
6.13.5.3 spiSelectl	202
6.13.5.4 spiUnselectl	202
6.13.5.5 spiStartIgnore	203
6.13.5.6 spiStartExchangel	203
6.13.5.7 spiStartSendl	204
6.13.5.8 spiStartReceivel	205
6.13.5.9 spiPolledExchange	205
6.13.5.10 _spi_wait_s	205
6.13.5.11 _spi_wakeup_isr	206
6.13.5.12 _spi_isr_code	206
6.13.5.13 STM32_SPI_USE_SPI1	207
6.13.5.14 STM32_SPI_USE_SPI2	207
6.13.5.15 STM32_SPI_USE_SPI3	207
6.13.5.16 STM32_SPI_SPI1_IRQ_PRIORITY	207
6.13.5.17 STM32_SPI_SPI2_IRQ_PRIORITY	208
6.13.5.18 STM32_SPI_SPI3_IRQ_PRIORITY	208
6.13.5.19 STM32_SPI_SPI1_DMA_PRIORITY	208
6.13.5.20 STM32_SPI_SPI2_DMA_PRIORITY	208
6.13.5.21 STM32_SPI_SPI3_DMA_PRIORITY	208
6.13.5.22 STM32_SPI_DMA_ERROR_HOOK	208
6.13.5.23 STM32_SPI_SPI1_RX_DMA_STREAM	208
6.13.5.24 STM32_SPI_SPI1_TX_DMA_STREAM	208
6.13.5.25 STM32_SPI_SPI2_RX_DMA_STREAM	209
6.13.5.26 STM32_SPI_SPI2_TX_DMA_STREAM	209
6.13.5.27 STM32_SPI_SPI3_RX_DMA_STREAM	209
6.13.5.28 STM32_SPI_SPI3_TX_DMA_STREAM	209
6.13.6 Typedef Documentation	209

6.13.6.1	SPIDriver	209
6.13.6.2	spicallback_t	209
6.13.7	Enumeration Type Documentation	209
6.13.7.1	spistate_t	209
6.14	Time Measurement Driver.	210
6.14.1	Detailed Description	210
6.14.2	Function Documentation	210
6.14.2.1	tmlInit	210
6.14.2.2	tmObjectInit	211
6.14.3	Define Documentation	211
6.14.3.1	tmStartMeasurement	211
6.14.3.2	tmStopMeasurement	211
6.14.4	Typedef Documentation	212
6.14.4.1	TimeMeasurement	212
6.15	UART Driver	212
6.15.1	Detailed Description	212
6.15.2	Driver State Machine	213
6.15.2.1	Transmitter sub State Machine	213
6.15.2.2	Receiver sub State Machine	213
6.15.3	Function Documentation	217
6.15.3.1	uartInit	217
6.15.3.2	uartObjectInit	217
6.15.3.3	uartStart	217
6.15.3.4	uartStop	218
6.15.3.5	uartStartSend	218
6.15.3.6	uartStartSendl	219
6.15.3.7	uartStopSend	219
6.15.3.8	uartStopSendl	220
6.15.3.9	uartStartReceive	221
6.15.3.10	uartStartReceive1	221
6.15.3.11	uartStopReceive	222
6.15.3.12	uartStopReceive1	222
6.15.3.13	CH_IRQ_HANDLER	223
6.15.3.14	CH_IRQ_HANDLER	223
6.15.3.15	CH_IRQ_HANDLER	223
6.15.3.16	uart_lld_init	224
6.15.3.17	uart_lld_start	224
6.15.3.18	uart_lld_stop	224
6.15.3.19	uart_lld_start_send	225
6.15.3.20	uart_lld_stop_send	225

6.15.3.21	uart_lld_start_receive	225
6.15.3.22	uart_lld_stop_receive	226
6.15.4	Variable Documentation	226
6.15.4.1	UARTD1	226
6.15.4.2	UARTD2	226
6.15.4.3	UARTD3	226
6.15.5	Define Documentation	226
6.15.5.1	UART_NO_ERROR	226
6.15.5.2	UART_PARITY_ERROR	226
6.15.5.3	UART_FRAMING_ERROR	227
6.15.5.4	UART_OVERRUN_ERROR	227
6.15.5.5	UART_NOISE_ERROR	227
6.15.5.6	UART_BREAK_DETECTED	227
6.15.5.7	STM32_UART_USE_USART1	227
6.15.5.8	STM32_UART_USE_USART2	227
6.15.5.9	STM32_UART_USE_USART3	227
6.15.5.10	STM32_UART_USART1_IRQ_PRIORITY	227
6.15.5.11	STM32_UART_USART2_IRQ_PRIORITY	227
6.15.5.12	STM32_UART_USART3_IRQ_PRIORITY	228
6.15.5.13	STM32_UART_USART1_DMA_PRIORITY	228
6.15.5.14	STM32_UART_USART2_DMA_PRIORITY	228
6.15.5.15	STM32_UART_USART3_DMA_PRIORITY	228
6.15.5.16	STM32_UART_DMA_ERROR_HOOK	228
6.15.5.17	STM32_UART_USART1_RX_DMA_STREAM	228
6.15.5.18	STM32_UART_USART1_TX_DMA_STREAM	228
6.15.5.19	STM32_UART_USART2_RX_DMA_STREAM	229
6.15.5.20	STM32_UART_USART2_TX_DMA_STREAM	229
6.15.5.21	STM32_UART_USART3_RX_DMA_STREAM	229
6.15.5.22	STM32_UART_USART3_TX_DMA_STREAM	229
6.15.6	Typedef Documentation	229
6.15.6.1	uartflags_t	229
6.15.6.2	UARTDriver	229
6.15.6.3	uartcb_t	229
6.15.6.4	uartccb_t	230
6.15.6.5	uartecb_t	230
6.15.7	Enumeration Type Documentation	230
6.15.7.1	uartstate_t	230
6.15.7.2	uarttxstate_t	230
6.15.7.3	uartrxstate_t	230
6.16	USB Driver	231

6.16.1	Detailed Description	231
6.16.2	Driver State Machine	231
6.16.3	USB Operations	231
6.16.3.1	USB Implementation	231
6.16.3.2	USB Endpoints	232
6.16.3.3	USB Packet Buffers	233
6.16.3.4	USB Callbacks	233
6.16.4	Function Documentation	239
6.16.4.1	usbInit	239
6.16.4.2	usbObjectInit	239
6.16.4.3	usbStart	239
6.16.4.4	usbStop	240
6.16.4.5	usbInitEndpoint1	240
6.16.4.6	usbDisableEndpoints1	241
6.16.4.7	usbStartReceive1	241
6.16.4.8	usbStartTransmit1	242
6.16.4.9	usbStallReceive1	243
6.16.4.10	usbStallTransmit1	243
6.16.4.11	_usb_reset	244
6.16.4.12	_usb_ep0setup	244
6.16.4.13	_usb_ep0in	245
6.16.4.14	_usb_ep0out	246
6.16.4.15	CH_IRQ_HANDLER	247
6.16.4.16	CH_IRQ_HANDLER	247
6.16.4.17	usb_lld_init	248
6.16.4.18	usb_lld_start	248
6.16.4.19	usb_lld_stop	249
6.16.4.20	usb_lld_reset	249
6.16.4.21	usb_lld_set_address	249
6.16.4.22	usb_lld_init_endpoint	250
6.16.4.23	usb_lld_disable_endpoints	250
6.16.4.24	usb_lld_get_status_out	250
6.16.4.25	usb_lld_get_status_in	250
6.16.4.26	usb_lld_read_setup	251
6.16.4.27	usb_lld_read_packet_buffer	251
6.16.4.28	usb_lld_write_packet_buffer	252
6.16.4.29	usb_lld_prepare_receive	252
6.16.4.30	usb_lld_prepare_transmit	252
6.16.4.31	usb_lld_start_out	253
6.16.4.32	usb_lld_start_in	253

6.16.4.33 <code>usb_lld_stall_out</code>	253
6.16.4.34 <code>usb_lld_stall_in</code>	254
6.16.4.35 <code>usb_lld_clear_out</code>	254
6.16.4.36 <code>usb_lld_clear_in</code>	254
6.16.5 Variable Documentation	254
6.16.5.1 <code>USBD1</code>	254
6.16.5.2 <code>in</code>	254
6.16.5.3 <code>out</code>	254
6.16.6 Define Documentation	254
6.16.6.1 <code>USB_DESC_INDEX</code>	255
6.16.6.2 <code>USB_DESC_BYTE</code>	255
6.16.6.3 <code>USB_DESC_WORD</code>	255
6.16.6.4 <code>USB_DESC_BCD</code>	255
6.16.6.5 <code>USB_DESC_DEVICE</code>	255
6.16.6.6 <code>USB_DESC_CONFIGURATION</code>	255
6.16.6.7 <code>USB_DESC_INTERFACE</code>	256
6.16.6.8 <code>USB_DESC_ENDPOINT</code>	256
6.16.6.9 <code>USB_EP_MODE_TYPE</code>	256
6.16.6.10 <code>USB_EP_MODE_TYPE_CTRL</code>	256
6.16.6.11 <code>USB_EP_MODE_TYPE_ISOC</code>	256
6.16.6.12 <code>USB_EP_MODE_TYPE_BULK</code>	256
6.16.6.13 <code>USB_EP_MODE_TYPE_INTR</code>	256
6.16.6.14 <code>USB_EP_MODE_TRANSACTION</code>	256
6.16.6.15 <code>USB_EP_MODE_PACKET</code>	256
6.16.6.16 <code>usbConnectBus</code>	257
6.16.6.17 <code>usbDisconnectBus</code>	257
6.16.6.18 <code>usbGetFrameNumber</code>	257
6.16.6.19 <code>usbGetTransmitStatus</code>	257
6.16.6.20 <code>usbGetReceiveStatus</code>	257
6.16.6.21 <code>usbReadPacketBuffer</code>	258
6.16.6.22 <code>usbWritePacketBuffer</code>	258
6.16.6.23 <code>usbPrepareReceive</code>	259
6.16.6.24 <code>usbPrepareTransmit</code>	259
6.16.6.25 <code>usbGetReceiveTransactionSize</code>	259
6.16.6.26 <code>usbGetReceivePacketSize</code>	260
6.16.6.27 <code>usbSetupTransfer</code>	260
6.16.6.28 <code>usbReadSetup</code>	261
6.16.6.29 <code>_usb_isr_invoke_event_cb</code>	261
6.16.6.30 <code>_usb_isr_invoke_sof_cb</code>	261
6.16.6.31 <code>_usb_isr_invoke_setup_cb</code>	262

6.16.6.32 _usb_isr_invoke_in_cb	262
6.16.6.33 _usb_isr_invoke_out_cb	262
6.16.6.34 USB_ENDPOINTS_NUMBER	263
6.16.6.35 STM32_USB_BASE	263
6.16.6.36 STM32_USBRAM_BASE	263
6.16.6.37 STM32_USB	263
6.16.6.38 STM32_USBRAM	263
6.16.6.39 USB_PMA_SIZE	263
6.16.6.40 EPR_TOGGLE_MASK	263
6.16.6.41 USB_GET_DESCRIPTOR	263
6.16.6.42 USB_ADDR2PTR	263
6.16.6.43 USB_MAX_ENDPOINTS	263
6.16.6.44 USB_SET_ADDRESS_MODE	264
6.16.6.45 STM32_USB_USE_USB1	264
6.16.6.46 STM32_USB_LOW_POWER_ON_SUSPEND	264
6.16.6.47 STM32_USB_USB1_HP_IRQ_PRIORITY	264
6.16.6.48 STM32_USB_USB1_LP_IRQ_PRIORITY	264
6.16.6.49 usb_lld_fetch_word	264
6.16.6.50 usb_lld_get_frame_number	264
6.16.6.51 usb_lld_get_transaction_size	265
6.16.6.52 usb_lld_get_packet_size	265
6.16.7 Typedef Documentation	265
6.16.7.1 USBDriver	265
6.16.7.2 usbep_t	265
6.16.7.3 usbcallback_t	265
6.16.7.4 usbepcallback_t	266
6.16.7.5 usbeventcb_t	266
6.16.7.6 usbreqhandler_t	266
6.16.7.7 usbgetdescriptor_t	266
6.16.8 Enumeration Type Documentation	266
6.16.8.1 usbstate_t	266
6.16.8.2 usbepstatus_t	267
6.16.8.3 usbep0state_t	267
6.16.8.4 usbevent_t	267
6.17 STM32L1xx Drivers	267
6.17.1 Detailed Description	267
6.18 STM32L1xx Initialization Support	268
6.18.1 Supported HW resources	268
6.18.2 STM32L1xx HAL driver implementation features	268
6.19 STM32L1xx ADC Support	268

6.19.1	Supported HW resources	268
6.19.2	STM32L1xx ADC driver implementation features	269
6.20	STM32L1xx EXT Support	269
6.20.1	Supported HW resources	269
6.20.2	STM32L1xx EXT driver implementation features	269
6.21	STM32L1xx GPT Support	269
6.21.1	Supported HW resources	269
6.21.2	STM32L1xx GPT driver implementation features	269
6.22	STM32L1xx ICU Support	269
6.22.1	Supported HW resources	270
6.22.2	STM32L1xx ICU driver implementation features	270
6.23	STM32L1xx PAL Support	270
6.23.1	Supported HW resources	270
6.23.2	STM32L1xx PAL driver implementation features	270
6.23.3	Supported PAL setup modes	270
6.23.4	Suboptimal behavior	271
6.24	STM32L1xx PWM Support	271
6.24.1	Supported HW resources	271
6.24.2	STM32L1xx PWM driver implementation features	271
6.25	STM32L1xx Serial Support	271
6.25.1	Supported HW resources	271
6.25.2	STM32L1xx Serial driver implementation features	272
6.26	STM32L1xx SPI Support	272
6.26.1	Supported HW resources	272
6.26.2	STM32L1xx SPI driver implementation features	272
6.27	STM32L1xx UART Support	272
6.27.1	Supported HW resources	272
6.27.2	STM32L1xx UART driver implementation features	273
6.28	STM32L1xx USB Support	273
6.28.1	Supported HW resources	273
6.28.2	STM32L1xx USB driver implementation features	273
6.29	STM32L1xx Platform Drivers	273
6.29.1	Detailed Description	273
6.30	STM32L1xx DMA Support	273
6.30.1	Detailed Description	273
6.30.2	Supported HW resources	274
6.30.3	STM32L1xx DMA driver implementation features	274
6.30.4	Function Documentation	277
6.30.4.1	CH_IRQ_HANDLER	277
6.30.4.2	CH_IRQ_HANDLER	277

6.30.4.3	CH_IRQ_HANDLER	277
6.30.4.4	CH_IRQ_HANDLER	277
6.30.4.5	CH_IRQ_HANDLER	277
6.30.4.6	CH_IRQ_HANDLER	278
6.30.4.7	CH_IRQ_HANDLER	278
6.30.4.8	dmalinit	278
6.30.4.9	dmaStreamAllocate	278
6.30.4.10	dmaStreamRelease	279
6.30.5	Variable Documentation	279
6.30.5.1	_stm32_dma_streams	279
6.30.6	Define Documentation	279
6.30.6.1	STM32_DMA1_STREAMS_MASK	280
6.30.6.2	STM32_DMA2_STREAMS_MASK	280
6.30.6.3	STM32_DMA_CCR_RESET_VALUE	280
6.30.6.4	STM32_DMA_STREAMS	280
6.30.6.5	STM32_DMA_ISR_MASK	280
6.30.6.6	STM32_DMA_GETCHANNEL	280
6.30.6.7	STM32_DMA_STREAM_ID	280
6.30.6.8	STM32_DMA_STREAM_ID_MSK	280
6.30.6.9	STM32_DMA_IS_VALID_ID	281
6.30.6.10	STM32_DMA_STREAM	281
6.30.6.11	STM32_DMA_CR_DMEIE	281
6.30.6.12	STM32_DMA_CR_CHSEL_MASK	281
6.30.6.13	STM32_DMA_CR_CHSEL	281
6.30.6.14	dmaStreamSetPeripheral	281
6.30.6.15	dmaStreamSetMemory0	282
6.30.6.16	dmaStreamSetTransactionSize	282
6.30.6.17	dmaStreamGetTransactionSize	283
6.30.6.18	dmaStreamSetMode	283
6.30.6.19	dmaStreamEnable	284
6.30.6.20	dmaStreamDisable	284
6.30.6.21	dmaStreamClearInterrupt	285
6.30.6.22	dmaStartMemcpy	285
6.30.6.23	dmaWaitCompletion	286
6.30.7	Typedef Documentation	286
6.30.7.1	stm32_dmaisr_t	286
6.31	STM32L1xx RCC Support	287
6.31.1	Detailed Description	287
6.31.2	Supported HW resources	287
6.31.3	STM32L1xx RCC driver implementation features	287

6.31.4 Define Documentation	290
6.31.4.1 rccEnableAPB1	290
6.31.4.2 rccDisableAPB1	290
6.31.4.3 rccResetAPB1	290
6.31.4.4 rccEnableAPB2	291
6.31.4.5 rccDisableAPB2	291
6.31.4.6 rccResetAPB2	291
6.31.4.7 rccEnableAHB	292
6.31.4.8 rccDisableAHB	292
6.31.4.9 rccResetAHB	292
6.31.4.10 rccEnableADC1	293
6.31.4.11 rccDisableADC1	293
6.31.4.12 rccResetADC1	293
6.31.4.13 rccEnableDMA1	293
6.31.4.14 rccDisableDMA1	293
6.31.4.15 rccResetDMA1	294
6.31.4.16 rccEnablePWRInterface	294
6.31.4.17 rccDisablePWRInterface	294
6.31.4.18 rccResetPWRInterface	294
6.31.4.19 rccEnableI2C1	294
6.31.4.20 rccDisableI2C1	295
6.31.4.21 rccResetI2C1	295
6.31.4.22 rccEnableI2C2	295
6.31.4.23 rccDisableI2C2	295
6.31.4.24 rccResetI2C2	295
6.31.4.25 rccEnableSPI1	296
6.31.4.26 rccDisableSPI1	296
6.31.4.27 rccResetSPI1	296
6.31.4.28 rccEnableSPI2	296
6.31.4.29 rccDisableSPI2	296
6.31.4.30 rccResetSPI2	297
6.31.4.31 rccEnableTIM2	297
6.31.4.32 rccDisableTIM2	297
6.31.4.33 rccResetTIM2	297
6.31.4.34 rccEnableTIM3	297
6.31.4.35 rccDisableTIM3	298
6.31.4.36 rccResetTIM3	298
6.31.4.37 rccEnableTIM4	298
6.31.4.38 rccDisableTIM4	298
6.31.4.39 rccResetTIM4	298

6.31.4.40 rccEnableUSART1	298
6.31.4.41 rccDisableUSART1	299
6.31.4.42 rccResetUSART1	299
6.31.4.43 rccEnableUSART2	299
6.31.4.44 rccDisableUSART2	299
6.31.4.45 rccResetUSART2	299
6.31.4.46 rccEnableUSART3	300
6.31.4.47 rccDisableUSART3	300
6.31.4.48 rccResetUSART3	300
6.31.4.49 rccEnableUSB	300
6.31.4.50 rccDisableUSB	300
6.31.4.51 rccResetUSB	301
7 Data Structure Documentation	302
7.1 ADCConfig Struct Reference	302
7.1.1 Detailed Description	302
7.2 ADCConversionGroup Struct Reference	302
7.2.1 Detailed Description	302
7.2.2 Field Documentation	304
7.2.2.1 circular	304
7.2.2.2 num_channels	304
7.2.2.3 end_cb	304
7.2.2.4 error_cb	304
7.2.2.5 cr1	304
7.2.2.6 cr2	305
7.2.2.7 smpr1	305
7.2.2.8 smpr2	305
7.2.2.9 smpr3	305
7.2.2.10 sqr1	305
7.2.2.11 sqr2	305
7.2.2.12 sqr3	305
7.2.2.13 sqr4	305
7.2.2.14 sqr5	305
7.3 ADCDriver Struct Reference	306
7.3.1 Detailed Description	306
7.3.2 Field Documentation	308
7.3.2.1 state	308
7.3.2.2 config	308
7.3.2.3 samples	308
7.3.2.4 depth	308

7.3.2.5	grpp	308
7.3.2.6	thread	308
7.3.2.7	mutex	308
7.3.2.8	adc	309
7.3.2.9	dmastp	309
7.3.2.10	dmamode	309
7.4	EXTChannelConfig Struct Reference	309
7.4.1	Detailed Description	309
7.4.2	Field Documentation	310
7.4.2.1	mode	310
7.4.2.2	cb	310
7.5	EXTConfig Struct Reference	310
7.5.1	Detailed Description	310
7.5.2	Field Documentation	311
7.5.2.1	channels	311
7.5.2.2	exti	311
7.6	EXTDriver Struct Reference	312
7.6.1	Detailed Description	312
7.6.2	Field Documentation	312
7.6.2.1	state	312
7.6.2.2	config	313
7.7	GPIO_TypeDef Struct Reference	313
7.7.1	Detailed Description	313
7.8	GPTConfig Struct Reference	313
7.8.1	Detailed Description	313
7.8.2	Field Documentation	315
7.8.2.1	frequency	315
7.8.2.2	callback	315
7.9	GPTDriver Struct Reference	315
7.9.1	Detailed Description	315
7.9.2	Field Documentation	317
7.9.2.1	state	317
7.9.2.2	config	317
7.9.2.3	clock	317
7.9.2.4	tim	317
7.10	I2CConfig Struct Reference	317
7.10.1	Detailed Description	317
7.10.2	Field Documentation	317
7.10.2.1	op_mode	317
7.10.2.2	clock_speed	318

7.10.2.3	duty_cycle	318
7.11	I2CDriver Struct Reference	318
7.11.1	Detailed Description	318
7.11.2	Field Documentation	319
7.11.2.1	state	319
7.11.2.2	config	319
7.11.2.3	errors	319
7.11.2.4	mutex	319
7.11.2.5	thread	319
7.11.2.6	addr	319
7.11.2.7	dmamode	320
7.11.2.8	dmarx	320
7.11.2.9	dmatx	320
7.11.2.10	i2c	320
7.12	ICUConfig Struct Reference	320
7.12.1	Detailed Description	320
7.12.2	Field Documentation	322
7.12.2.1	mode	322
7.12.2.2	frequency	322
7.12.2.3	width_cb	322
7.12.2.4	period_cb	322
7.13	ICUDriver Struct Reference	322
7.13.1	Detailed Description	322
7.13.2	Field Documentation	324
7.13.2.1	state	324
7.13.2.2	config	324
7.13.2.3	clock	324
7.13.2.4	tim	324
7.14	IOBus Struct Reference	324
7.14.1	Detailed Description	324
7.14.2	Field Documentation	325
7.14.2.1	portid	325
7.14.2.2	mask	325
7.14.2.3	offset	326
7.15	MMCConfig Struct Reference	326
7.15.1	Detailed Description	326
7.16	MMCDriver Struct Reference	326
7.16.1	Detailed Description	326
7.16.2	Field Documentation	328
7.16.2.1	state	328

7.16.2.2 config	328
7.16.2.3 spip	328
7.16.2.4 lscfg	328
7.16.2.5 hscfg	328
7.16.2.6 is_protected	328
7.16.2.7 is_inserted	328
7.16.2.8 inserted_event	329
7.16.2.9 removed_event	329
7.16.2.10 vt	329
7.16.2.11 cnt	329
7.17 PALConfig Struct Reference	329
7.17.1 Detailed Description	329
7.17.2 Field Documentation	330
7.17.2.1 PADATA	330
7.17.2.2 PBData	330
7.17.2.3 PCData	331
7.17.2.4 PDData	331
7.18 PWMChannelConfig Struct Reference	331
7.18.1 Detailed Description	331
7.18.2 Field Documentation	333
7.18.2.1 mode	333
7.18.2.2 callback	333
7.19 PWMConfig Struct Reference	333
7.19.1 Detailed Description	333
7.19.2 Field Documentation	335
7.19.2.1 frequency	335
7.19.2.2 period	335
7.19.2.3 callback	335
7.19.2.4 channels	335
7.19.2.5 cr2	335
7.19.2.6 bdtr	336
7.20 PWMDriver Struct Reference	336
7.20.1 Detailed Description	336
7.20.2 Field Documentation	338
7.20.2.1 state	338
7.20.2.2 config	338
7.20.2.3 period	338
7.20.2.4 clock	338
7.20.2.5 tim	338
7.21 SerialConfig Struct Reference	338

7.21.1	Detailed Description	338
7.21.2	Field Documentation	339
7.21.2.1	sc_speed	339
7.21.2.2	sc_cr1	339
7.21.2.3	sc_cr2	339
7.21.2.4	sc_cr3	339
7.22	SerialDriver Struct Reference	339
7.22.1	Detailed Description	339
7.22.2	Field Documentation	340
7.22.2.1	vmt	340
7.23	SerialDriverVMT Struct Reference	340
7.23.1	Detailed Description	340
7.24	SPIConfig Struct Reference	340
7.24.1	Detailed Description	340
7.24.2	Field Documentation	342
7.24.2.1	end_cb	342
7.24.2.2	ssport	342
7.24.2.3	sspad	342
7.24.2.4	cr1	342
7.25	SPIDriver Struct Reference	342
7.25.1	Detailed Description	342
7.25.2	Field Documentation	344
7.25.2.1	state	344
7.25.2.2	config	344
7.25.2.3	thread	344
7.25.2.4	mutex	344
7.25.2.5	spi	344
7.25.2.6	dmarx	344
7.25.2.7	dmatx	344
7.25.2.8	rxdmamode	344
7.25.2.9	txdmamode	345
7.26	stm32_dma_stream_t Struct Reference	345
7.26.1	Detailed Description	345
7.26.2	Field Documentation	345
7.26.2.1	channel	345
7.26.2.2	ifcr	345
7.26.2.3	ishift	345
7.26.2.4	selfindex	345
7.26.2.5	vector	345
7.27	stm32_gpio_setup_t Struct Reference	346

7.27.1	Detailed Description	346
7.27.2	Field Documentation	346
7.27.2.1	moder	346
7.27.2.2	otyper	346
7.27.2.3	ospeedr	346
7.27.2.4	pupdr	346
7.27.2.5	odr	346
7.27.2.6	afrl	346
7.27.2.7	afrh	346
7.28	stm32_tim_t Struct Reference	347
7.28.1	Detailed Description	347
7.29	stm32_usb_descriptor_t Struct Reference	347
7.29.1	Detailed Description	347
7.29.2	Field Documentation	347
7.29.2.1	TXADDR0	347
7.29.2.2	TXCOUNT0	347
7.29.2.3	TXCOUNT1	347
7.29.2.4	RXADDR0	348
7.29.2.5	RXCOUNT0	348
7.29.2.6	RXCOUNT1	348
7.30	stm32_usb_t Struct Reference	348
7.30.1	Detailed Description	348
7.30.2	Field Documentation	348
7.30.2.1	EPR	348
7.31	TimeMeasurement Struct Reference	348
7.31.1	Detailed Description	348
7.31.2	Field Documentation	349
7.31.2.1	start	349
7.31.2.2	stop	349
7.31.2.3	last	349
7.31.2.4	worst	349
7.31.2.5	best	349
7.32	UARTConfig Struct Reference	349
7.32.1	Detailed Description	349
7.32.2	Field Documentation	351
7.32.2.1	txend1_cb	351
7.32.2.2	txend2_cb	351
7.32.2.3	rxend_cb	351
7.32.2.4	rxchar_cb	351
7.32.2.5	rxerr_cb	351

7.32.2.6 speed	351
7.32.2.7 cr1	351
7.32.2.8 cr2	351
7.32.2.9 cr3	351
7.33 UARTDriver Struct Reference	352
7.33.1 Detailed Description	352
7.33.2 Field Documentation	354
7.33.2.1 state	354
7.33.2.2 txstate	354
7.33.2.3 rxstate	354
7.33.2.4 config	354
7.33.2.5 usart	354
7.33.2.6 dmamode	354
7.33.2.7 dmarx	354
7.33.2.8 dmatx	354
7.33.2.9 rdbuf	355
7.34 USBConfig Struct Reference	355
7.34.1 Detailed Description	355
7.34.2 Field Documentation	357
7.34.2.1 event_cb	357
7.34.2.2 get_descriptor_cb	357
7.34.2.3 requests_hook_cb	357
7.34.2.4 sof_cb	357
7.35 USBDescriptor Struct Reference	357
7.35.1 Detailed Description	357
7.35.2 Field Documentation	357
7.35.2.1 ud_size	357
7.35.2.2 ud_string	358
7.36 USBDriver Struct Reference	358
7.36.1 Detailed Description	358
7.36.2 Field Documentation	359
7.36.2.1 state	359
7.36.2.2 config	359
7.36.2.3 param	359
7.36.2.4 transmitting	359
7.36.2.5 receiving	359
7.36.2.6 epc	359
7.36.2.7 ep0state	359
7.36.2.8 ep0next	360
7.36.2.9 ep0n	360

7.36.2.10	ep0endcb	360
7.36.2.11	setup	360
7.36.2.12	status	360
7.36.2.13	address	360
7.36.2.14	configuration	360
7.36.2.15	pmnnext	360
7.37	USBEndpointConfig Struct Reference	360
7.37.1	Detailed Description	360
7.37.2	Field Documentation	362
7.37.2.1	ep_mode	362
7.37.2.2	setup_cb	362
7.37.2.3	in_cb	362
7.37.2.4	out_cb	362
7.37.2.5	in_maxsize	362
7.37.2.6	out_maxsize	363
7.37.2.7	in_state	363
7.37.2.8	out_state	363
7.38	USBInEndpointState Struct Reference	363
7.38.1	Detailed Description	363
7.38.2	Field Documentation	363
7.38.2.1	txbuf	363
7.38.2.2	txsize	363
7.38.2.3	txcnt	363
7.39	USBOutEndpointState Struct Reference	364
7.39.1	Detailed Description	364
7.39.2	Field Documentation	364
7.39.2.1	rwpkts	364
7.39.2.2	rxbuf	364
7.39.2.3	rxsize	364
7.39.2.4	rxcnt	364
8	File Documentation	365
8.1	adc.c File Reference	365
8.1.1	Detailed Description	365
8.2	adc.h File Reference	366
8.2.1	Detailed Description	366
8.3	adc_lld.c File Reference	367
8.3.1	Detailed Description	367
8.4	adc_lld.h File Reference	367
8.4.1	Detailed Description	367

8.5 ext.c File Reference	372
8.5.1 Detailed Description	372
8.6 ext_lld.c File Reference	373
8.6.1 Detailed Description	373
8.7 ext_lld.h File Reference	373
8.7.1 Detailed Description	373
8.8 gpt.c File Reference	375
8.8.1 Detailed Description	375
8.9 gpt.h File Reference	376
8.9.1 Detailed Description	376
8.10 gpt_lld.c File Reference	377
8.10.1 Detailed Description	377
8.11 gpt_lld.h File Reference	378
8.11.1 Detailed Description	378
8.12 hal.c File Reference	379
8.12.1 Detailed Description	379
8.13 hal.h File Reference	379
8.13.1 Detailed Description	379
8.14 hal_lld.c File Reference	380
8.14.1 Detailed Description	380
8.15 hal_lld.h File Reference	380
8.15.1 Detailed Description	380
8.16 halconf.h File Reference	387
8.16.1 Detailed Description	387
8.17 i2c.c File Reference	389
8.17.1 Detailed Description	389
8.18 i2c.h File Reference	389
8.18.1 Detailed Description	389
8.19 i2c_lld.c File Reference	391
8.19.1 Detailed Description	391
8.20 i2c_lld.h File Reference	392
8.20.1 Detailed Description	392
8.21 icu.c File Reference	393
8.21.1 Detailed Description	393
8.22 icu.h File Reference	394
8.22.1 Detailed Description	394
8.23 icu_lld.c File Reference	395
8.23.1 Detailed Description	395
8.24 icu_lld.h File Reference	395
8.24.1 Detailed Description	395

8.25 mmc_spi.c File Reference	397
8.25.1 Detailed Description	397
8.26 mmc_spi.h File Reference	398
8.26.1 Detailed Description	398
8.27 pal.c File Reference	399
8.27.1 Detailed Description	399
8.28 pal.h File Reference	399
8.28.1 Detailed Description	399
8.29 pal_lld.c File Reference	401
8.29.1 Detailed Description	401
8.30 pal_lld.h File Reference	401
8.30.1 Detailed Description	401
8.31 pwm.c File Reference	403
8.31.1 Detailed Description	403
8.32 pwm.h File Reference	404
8.32.1 Detailed Description	404
8.33 pwm_lld.c File Reference	405
8.33.1 Detailed Description	405
8.34 pwm_lld.h File Reference	406
8.34.1 Detailed Description	406
8.35 serial.c File Reference	407
8.35.1 Detailed Description	407
8.36 serial.h File Reference	408
8.36.1 Detailed Description	408
8.37 serial_lld.c File Reference	410
8.37.1 Detailed Description	410
8.38 serial_lld.h File Reference	410
8.38.1 Detailed Description	410
8.39 spi.c File Reference	412
8.39.1 Detailed Description	412
8.40 spi.h File Reference	412
8.40.1 Detailed Description	412
8.41 spi_lld.c File Reference	414
8.41.1 Detailed Description	414
8.42 spi_lld.h File Reference	415
8.42.1 Detailed Description	415
8.43 stm32.h File Reference	416
8.43.1 Detailed Description	416
8.44 stm32_dma.c File Reference	417
8.44.1 Detailed Description	417

8.45	stm32_dma.h File Reference	418
8.45.1	Detailed Description	418
8.46	stm32_rcc.h File Reference	420
8.46.1	Detailed Description	420
8.47	stm32_usb.h File Reference	422
8.47.1	Detailed Description	422
8.48	tm.c File Reference	423
8.48.1	Detailed Description	423
8.49	tm.h File Reference	424
8.49.1	Detailed Description	424
8.50	uart.c File Reference	424
8.50.1	Detailed Description	424
8.51	uart.h File Reference	425
8.51.1	Detailed Description	425
8.52	uart_lld.c File Reference	426
8.52.1	Detailed Description	426
8.53	uart_lld.h File Reference	427
8.53.1	Detailed Description	427
8.54	usb.c File Reference	428
8.54.1	Detailed Description	428
8.55	usb.h File Reference	429
8.55.1	Detailed Description	429
8.56	usb_lld.c File Reference	432
8.56.1	Detailed Description	432
8.56.2	Variable Documentation	433
8.56.2.1	in	433
8.56.2.2	out	433
8.57	usb_lld.h File Reference	434
8.57.1	Detailed Description	434

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 STM32L1xx 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	8
Configuration	10
ADC Driver	16
EXT Driver	43
GPT Driver	54
HAL Driver	67
I2C Driver	96
ICU Driver	113
MMC over SPI Driver	126
PAL Driver	136
PWM Driver	155
Serial Driver	172
SPI Driver	188
Time Measurement Driver.	210
UART Driver	212
USB Driver	231
STM32L1xx Drivers	267
STM32L1xx Initialization Support	268
STM32L1xx ADC Support	268
STM32L1xx EXT Support	269
STM32L1xx GPT Support	269
STM32L1xx ICU Support	269
STM32L1xx PAL Support	270
STM32L1xx PWM Support	271
STM32L1xx Serial Support	271
STM32L1xx SPI Support	272
STM32L1xx UART Support	272
STM32L1xx USB Support	273
STM32L1xx Platform Drivers	273
STM32L1xx DMA Support	273
STM32L1xx RCC Support	287

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

ADCConfig (Driver configuration structure)	302
ADCConversionGroup (Conversion group configuration structure)	302
ADCDriver (Structure representing an ADC driver)	306
EXTChannelConfig (Channel configuration structure)	309
EXTConfig (Driver configuration structure)	310
EXTDriver (Structure representing an EXT driver)	312
GPIO_TypeDef (STM32 GPIO registers block)	313
GPTConfig (Driver configuration structure)	313
GPTDriver (Structure representing a GPT driver)	315
I2CConfig (Driver configuration structure)	317
I2CDriver (Structure representing an I2C driver)	318
ICUConfig (Driver configuration structure)	320
ICUDriver (Structure representing an ICU driver)	322
IOBus (I/O bus descriptor)	324
MMCConfig (Driver configuration structure)	326
MMCDriver (Structure representing a MMC driver)	326
PALConfig (STM32 GPIO static initializer)	329
PWMChannelConfig (PWM driver channel configuration structure)	331
PWMConfig (PWM driver configuration structure)	333
PWMDriver (Structure representing a PWM driver)	336
SerialConfig (STM32 Serial Driver configuration structure)	338
SerialDriver (Full duplex serial driver class)	339
SerialDriverVMT (SerialDriver virtual methods table)	340
SPIConfig (Driver configuration structure)	340
SPIDriver (Structure representing a SPI driver)	342
stm32_dma_stream_t (STM32 DMA stream descriptor structure)	345
stm32_gpio_setup_t (GPIO port setup info)	346
stm32_tim_t (STM32 TIM registers block)	347
stm32_usb_descriptor_t (USB descriptor registers block)	347
stm32_usb_t (USB registers block)	348
TimeMeasurement (Time Measurement structure)	348
UARTConfig (Driver configuration structure)	349
UARTDriver (Structure representing an UART driver)	352
USBConfig (Type of an USB driver configuration structure)	355
USBDescriptor (Type of an USB descriptor)	357
USBDriver (Structure representing an USB driver)	358
USBEndpointConfig (Type of an USB endpoint configuration structure)	360
USBInEndpointState (Type of an endpoint state structure)	363

USBOutEndpointState (Type of an endpoint state structure) 364

Chapter 5

File Index

5.1 File List

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

adc.c (ADC Driver code)	365
adc.h (ADC Driver macros and structures)	366
adc_lld.c (STM32L1xx ADC subsystem low level driver source)	367
adc_lld.h (STM32L1xx ADC subsystem low level driver header)	367
ext.c (EXT Driver code)	372
ext_lld.c (STM32 EXT subsystem low level driver source)	373
ext_lld.h (STM32 EXT subsystem low level driver header)	373
gpt.c (GPT Driver code)	375
gpt.h (GPT Driver macros and structures)	376
gpt_lld.c (STM32 GPT subsystem low level driver source)	377
gpt_lld.h (STM32 GPT subsystem low level driver header)	378
hal.c (HAL subsystem code)	379
hal.h (HAL subsystem header)	379
hal_lld.c (STM32L1xx HAL subsystem low level driver source)	380
hal_lld.h (STM32L1xx HAL subsystem low level driver header)	380
halconf.h (HAL configuration header)	387
i2c.c (I2C Driver code)	389
i2c.h (I2C Driver macros and structures)	389
i2c_lld.c (STM32 I2C subsystem low level driver source)	391
i2c_lld.h (STM32 I2C subsystem low level driver header)	392
icu.c (ICU Driver code)	393
icu.h (ICU Driver macros and structures)	394
icu_lld.c (STM32 ICU subsystem low level driver header)	395
icu_lld.h (STM32 ICU subsystem low level driver header)	395
mmc_spi.c (MMC over SPI driver code)	397
mmc_spi.h (MMC over SPI driver header)	398
pal.c (I/O Ports Abstraction Layer code)	399
pal.h (I/O Ports Abstraction Layer macros, types and structures)	399
pal_lld.c (STM32L1xx/STM32F2xx/STM32F4xx GPIO low level driver code)	401
pal_lld.h (STM32L1xx/STM32F2xx/STM32F4xx GPIO low level driver header)	401
pwm.c (PWM Driver code)	403
pwm.h (PWM Driver macros and structures)	404
pwm_lld.c (STM32 PWM subsystem low level driver header)	405
pwm_lld.h (STM32 PWM subsystem low level driver header)	406
serial.c (Serial Driver code)	407
serial.h (Serial Driver macros and structures)	408
serial_lld.c (STM32 low level serial driver code)	410
serial_lld.h (STM32 low level serial driver header)	410

spi.c (SPI Driver code)	412
spi.h (SPI Driver macros and structures)	412
spi_lld.c (STM32 SPI subsystem low level driver source)	414
spi_lld.h (STM32 SPI subsystem low level driver header)	415
stm32.h (STM32 common header)	416
stm32_dma.c (DMA helper driver code)	417
stm32_dma.h (DMA helper driver header)	418
stm32_rcc.h (RCC helper driver header)	420
stm32_usb.h (STM32 USB registers layout header)	422
tm.c (Time Measurement driver code)	423
tm.h (Time Measurement driver header)	424
uart.c (UART Driver code)	424
uart.h (UART Driver macros and structures)	425
uart_lld.c (STM32 low level UART driver code)	426
uart_lld.h (STM32 low level UART driver header)	427
usb.c (USB Driver code)	428
usb.h (USB Driver macros and structures)	429
usb_lld.c (STM32 USB subsystem low level driver source)	432
usb_lld.h (STM32 USB subsystem low level driver header)	434

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`.

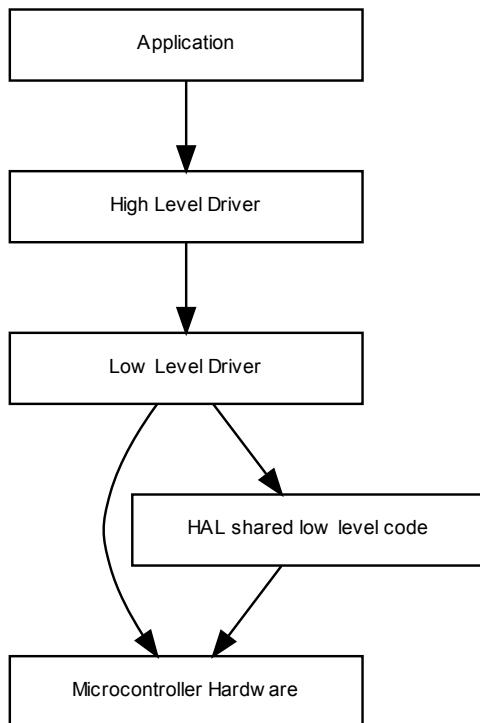
- Low Level Device Driver (**LLD**). This layer contains the platform dependent part of the driver.

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.
- 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.
- Serial Driver

- [Generic Serial 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`
 - [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`

- `#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.

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_MMC_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

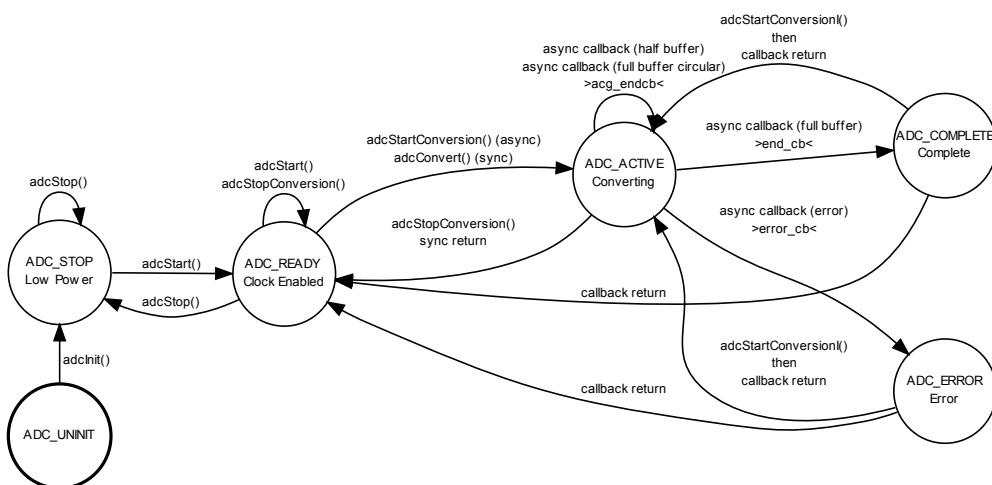
Generic ADC Driver. This module implements a generic ADC (Analog to Digital Converter) driver supporting a variety of buffer and conversion modes.

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 **ADCConfig**
Driver configuration structure.
- struct **ADCDriver**
Structure representing an ADC driver.

Functions

- void **adcInit** (void)
ADC Driver initialization.
- void **adcObjectInit** (**ADCDriver** *adcp)
*Initializes the standard part of a **ADCDriver** structure.*
- void **adcStart** (**ADCDriver** *adcp, const **ADCConfig** *config)
Configures and activates the ADC peripheral.
- void **adcStop** (**ADCDriver** *adcp)
Deactivates the ADC peripheral.
- void **adcStartConversion** (**ADCDriver** *adcp, const **ADCConversionGroup** *grpp, **adcsample_t** *samples, **size_t** depth)
Starts an ADC conversion.
- void **adcStartConversionI** (**ADCDriver** *adcp, const **ADCConversionGroup** *grpp, **adcsample_t** *samples, **size_t** depth)
Starts an ADC conversion.
- void **adcStopConversion** (**ADCDriver** *adcp)
Stops an ongoing conversion.
- void **adcStopConversionI** (**ADCDriver** *adcp)

- `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.
- `CH_IRQ_HANDLER (ADC1_IRQHandler)`

ADC interrupt handler.
- `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.
- `void adcSTM32EnableTSVREFE (void)`

Enables the TSVREFE bit.
- `void adcSTM32DisableTSVREFE (void)`

Disables the TSVREFE bit.

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) << 24)`
Trigger source.

ADC clock divider settings

- `#define ADC_CCR_ADCPRE_DIV1 0`
- `#define ADC_CCR_ADCPRE_DIV2 1`
- `#define ADC_CCR_ADCPRE_DIV4 2`

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.
- `#define ADC_CHANNEL_IN18 18`
External analog input 18.

- `#define ADC_CHANNEL_IN19 19`
External analog input 19.
- `#define ADC_CHANNEL_IN20 20`
External analog input 20.
- `#define ADC_CHANNEL_IN21 21`
External analog input 21.
- `#define ADC_CHANNEL_IN22 22`
External analog input 22.
- `#define ADC_CHANNEL_IN23 23`
External analog input 23.
- `#define ADC_CHANNEL_IN24 24`
External analog input 24.
- `#define ADC_CHANNEL_IN25 25`
External analog input 25.

Sampling rates

- `#define ADC_SAMPLE_4 0`
4 cycles sampling time.
- `#define ADC_SAMPLE_9 1`
9 cycles sampling time.
- `#define ADC_SAMPLE_16 2`
16 cycles sampling time.
- `#define ADC_SAMPLE_24 3`
24 cycles sampling time.
- `#define ADC_SAMPLE_48 4`
48 cycles sampling time.
- `#define ADC_SAMPLE_96 5`
96 cycles sampling time.
- `#define ADC_SAMPLE_192 6`
192 cycles sampling time.
- `#define ADC_SAMPLE_384 7`
384 cycles sampling time.

Configuration options

- `#define STM32_ADC_USE_ADC1 TRUE`
ADC1 driver enable switch.
- `#define STM32_ADC_ADCPRE ADC_CCR_ADCPRE_DIV1`
ADC common clock divider.
- `#define STM32_ADC_ADC1_DMA_PRIORITY 2`
ADC1 DMA priority (0..3|lowest..highest).
- `#define STM32_ADC_IRQ_PRIORITY 5`
ADC interrupt priority level setting.
- `#define STM32_ADC_ADC1_DMA_IRQ_PRIORITY 5`
ADC1 DMA 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_SQR5_SQ1_N(n) ((n) << 0)`
1st channel in seq.
- `#define ADC_SQR5_SQ2_N(n) ((n) << 5)`
2nd channel in seq.
- `#define ADC_SQR5_SQ3_N(n) ((n) << 10)`
3rd channel in seq.
- `#define ADC_SQR5_SQ4_N(n) ((n) << 15)`
4th channel in seq.
- `#define ADC_SQR5_SQ5_N(n) ((n) << 20)`
5th channel in seq.
- `#define ADC_SQR5_SQ6_N(n) ((n) << 25)`
6th channel in seq.
- `#define ADC_SQR4_SQ7_N(n) ((n) << 0)`
7th channel in seq.
- `#define ADC_SQR4_SQ8_N(n) ((n) << 5)`
8th channel in seq.
- `#define ADC_SQR4_SQ9_N(n) ((n) << 10)`
9th channel in seq.
- `#define ADC_SQR4_SQ10_N(n) ((n) << 15)`
10th channel in seq.
- `#define ADC_SQR4_SQ11_N(n) ((n) << 20)`
11th channel in seq.
- `#define ADC_SQR4_SQ12_N(n) ((n) << 25)`
12th channel in seq.
- `#define ADC_SQR3_SQ13_N(n) ((n) << 0)`
13th channel in seq.
- `#define ADC_SQR3_SQ14_N(n) ((n) << 5)`
14th channel in seq.
- `#define ADC_SQR3_SQ15_N(n) ((n) << 10)`
15th channel in seq.
- `#define ADC_SQR3_SQ16_N(n) ((n) << 15)`
16th channel in seq.
- `#define ADC_SQR3_SQ17_N(n) ((n) << 20)`
17th channel in seq.
- `#define ADC_SQR3_SQ18_N(n) ((n) << 25)`
18th channel in seq.
- `#define ADC_SQR2_SQ19_N(n) ((n) << 0)`
19th channel in seq.
- `#define ADC_SQR2_SQ20_N(n) ((n) << 5)`
20th channel in seq.
- `#define ADC_SQR2_SQ21_N(n) ((n) << 10)`
21th channel in seq.
- `#define ADC_SQR2_SQ22_N(n) ((n) << 15)`
22th channel in seq.
- `#define ADC_SQR2_SQ23_N(n) ((n) << 20)`
23th channel in seq.
- `#define ADC_SQR2_SQ24_N(n) ((n) << 25)`

- #define ADC_SQR1_SQ25_N(n) ((n) << 0)
24th channel in seq.
- #define ADC_SQR1_SQ26_N(n) ((n) << 5)
25th channel in seq.
- #define ADC_SQR1_SQ27_N(n) ((n) << 10)
26th channel in seq.
- #define ADC_SQR1_SQ28_N(n) ((n) << 0)
27th channel in seq.

Sampling rate settings helper macros

- #define ADC_SMPR3_SMP_AN0(n) ((n) << 0)
AN0 sampling time.
- #define ADC_SMPR3_SMP_AN1(n) ((n) << 3)
AN1 sampling time.
- #define ADC_SMPR3_SMP_AN2(n) ((n) << 6)
AN2 sampling time.
- #define ADC_SMPR3_SMP_AN3(n) ((n) << 9)
AN3 sampling time.
- #define ADC_SMPR3_SMP_AN4(n) ((n) << 12)
AN4 sampling time.
- #define ADC_SMPR3_SMP_AN5(n) ((n) << 15)
AN5 sampling time.
- #define ADC_SMPR3_SMP_AN6(n) ((n) << 18)
AN6 sampling time.
- #define ADC_SMPR3_SMP_AN7(n) ((n) << 21)
AN7 sampling time.
- #define ADC_SMPR3_SMP_AN8(n) ((n) << 24)
AN8 sampling time.
- #define ADC_SMPR3_SMP_AN9(n) ((n) << 27)
AN9 sampling time.
- #define ADC_SMPR2_SMP_AN10(n) ((n) << 0)
AN10 sampling time.
- #define ADC_SMPR2_SMP_AN11(n) ((n) << 3)
AN11 sampling time.
- #define ADC_SMPR2_SMP_AN12(n) ((n) << 6)
AN12 sampling time.
- #define ADC_SMPR2_SMP_AN13(n) ((n) << 9)
AN13 sampling time.
- #define ADC_SMPR2_SMP_AN14(n) ((n) << 12)
AN14 sampling time.
- #define ADC_SMPR2_SMP_AN15(n) ((n) << 15)
AN15 sampling time.
- #define ADC_SMPR2_SMP_SENSOR(n) ((n) << 18)
Temperature Sensor sampling time.
- #define ADC_SMPR2_SMP_VREF(n) ((n) << 21)
Voltage Reference sampling time.
- #define ADC_SMPR2_SMP_AN18(n) ((n) << 24)
AN18 sampling time.
- #define ADC_SMPR2_SMP_AN19(n) ((n) << 27)
AN19 sampling time.
- #define ADC_SMPR1_SMP_AN20(n) ((n) << 0)

- #define ADC_SMPR1_SMP_AN21(n) ((n) << 3)
AN20 sampling time.
- #define ADC_SMPR1_SMP_AN22(n) ((n) << 6)
AN21 sampling time.
- #define ADC_SMPR1_SMP_AN23(n) ((n) << 9)
AN22 sampling time.
- #define ADC_SMPR1_SMP_AN24(n) ((n) << 12)
AN23 sampling time.
- #define ADC_SMPR1_SMP_AN25(n) ((n) << 15)
AN24 sampling time.
- #define ADC_SMPR1_SMP_AN26(n) ((n) << 18)
AN25 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, ADC_ERR_OVERFLOW = 1 }

Possible ADC failure causes.

6.3.4 Function Documentation

6.3.4.1 void adclinit(void)

ADC Driver initialization.

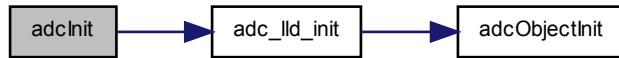
Note

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

Function Class:

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

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

Configures and activates the ADC peripheral.

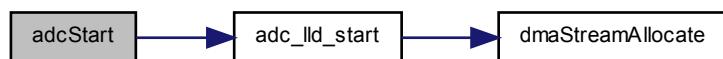
Parameters

in	<i>adcp</i> pointer to the ADCDriver object
in	<i>config</i> pointer to the ADCConfig object. Depending on the implementation the value can be NULL.

Function Class:

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

Here is the call graph for this function:



6.3.4.4 void adcStop (ADCDriver * *adcp*)

Deactivates the ADC peripheral.

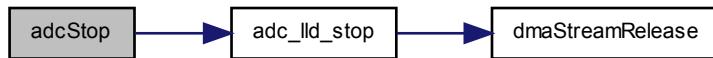
Parameters

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

Function Class:

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

Here is the call graph for this function:



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

in *grpp* pointer to a [ADCConversionGroup](#) object

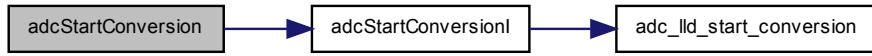
out *samples* pointer to the samples buffer

in *depth* buffer depth (matrix rows number). The buffer depth must be one or an even number.

Function Class:

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

Here is the call graph for this function:



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

Function Class:

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

Function Class:

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

Here is the call graph for this function:



6.3.4.8 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:

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	<i>adcp</i> pointer to the <code>ADCDriver</code> 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

`RDY_OK` Conversion finished.

`RDY_RESET` The conversion has been stopped using `acdStopConversion()` or `acdStopConversionI()`, the result buffer may contain incorrect data.

`RDY_TIMEOUT` 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 CH_IRQ_HANDLER (ADC1_IRQHandler)

ADC interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.3.4.13 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:

**6.3.4.14 void adc_lld_start (ADCDriver * adcp)**

Configures and activates the ADC peripheral.

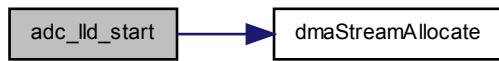
Parameters

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:

**6.3.4.15 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:

**6.3.4.16 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.

6.3.4.17 void adc_lld_stop_conversion (ADCDriver * *adcp*)

Stops an ongoing conversion.

Parameters

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

Function Class:

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

6.3.4.18 void adcSTM32EnableTSVREFE (void)

Enables the TSVREFE bit.

The TSVREFE bit is required in order to sample the internal temperature sensor and internal reference voltage.

Note

This is an STM32-only functionality.

6.3.4.19 void adcSTM32DisableTSVREFE (void)

Disables the TSVREFE bit.

The TSVREFE bit is required in order to sample the internal temperature sensor and internal reference voltage.

Note

This is an STM32-only functionality.

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 ADCDriver 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*) << 24)

Trigger source.

6.3.6.11 #define ADC_CHANNEL_IN0 0

External analog input 0.

6.3.6.12 #define ADC_CHANNEL_IN1 1

External analog input 1.

6.3.6.13 #define ADC_CHANNEL_IN2 2

External analog input 2.

6.3.6.14 #define ADC_CHANNEL_IN3 3

External analog input 3.

6.3.6.15 #define ADC_CHANNEL_IN4 4

External analog input 4.

6.3.6.16 #define ADC_CHANNEL_IN5 5

External analog input 5.

6.3.6.17 #define ADC_CHANNEL_IN6 6

External analog input 6.

6.3.6.18 #define ADC_CHANNEL_IN7 7

External analog input 7.

6.3.6.19 #define ADC_CHANNEL_IN8 8

External analog input 8.

6.3.6.20 #define ADC_CHANNEL_IN9 9

External analog input 9.

6.3.6.21 #define ADC_CHANNEL_IN10 10

External analog input 10.

6.3.6.22 #define ADC_CHANNEL_IN11 11

External analog input 11.

6.3.6.23 #define ADC_CHANNEL_IN12 12

External analog input 12.

6.3.6.24 #define ADC_CHANNEL_IN13 13

External analog input 13.

6.3.6.25 #define ADC_CHANNEL_IN14 14

External analog input 14.

6.3.6.26 #define ADC_CHANNEL_IN15 15

External analog input 15.

6.3.6.27 #define ADC_CHANNEL_SENSOR 16

Internal temperature sensor.

6.3.6.28 #define ADC_CHANNEL_VREFINT 17

Internal reference.

6.3.6.29 #define ADC_CHANNEL_IN18 18

External analog input 18.

6.3.6.30 #define ADC_CHANNEL_IN19 19

External analog input 19.

6.3.6.31 #define ADC_CHANNEL_IN20 20

External analog input 20.

6.3.6.32 #define ADC_CHANNEL_IN21 21

External analog input 21.

6.3.6.33 #define ADC_CHANNEL_IN22 22

External analog input 22.

6.3.6.34 #define ADC_CHANNEL_IN23 23

External analog input 23.

6.3.6.35 #define ADC_CHANNEL_IN24 24

External analog input 24.

6.3.6.36 #define ADC_CHANNEL_IN25 25

External analog input 25.

6.3.6.37 #define ADC_SAMPLE_4 0

4 cycles sampling time.

6.3.6.38 #define ADC_SAMPLE_9 1

9 cycles sampling time.

6.3.6.39 #define ADC_SAMPLE_16 2

16 cycles sampling time.

6.3.6.40 #define ADC_SAMPLE_24 3

24 cycles sampling time.

6.3.6.41 #define ADC_SAMPLE_48 4

48 cycles sampling time.

6.3.6.42 #define ADC_SAMPLE_96 5

96 cycles sampling time.

6.3.6.43 #define ADC_SAMPLE_192 6

192 cycles sampling time.

6.3.6.44 #define ADC_SAMPLE_384 7

384 cycles sampling time.

6.3.6.45 #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.

6.3.6.46 #define STM32_ADC_ADCPRE ADC_CCR_ADCPRE_DIV1

ADC common clock divider.

Note

This setting is influenced by the VDDA voltage and other external conditions, please refer to the STM32L15x datasheet for more info.

See section 6.3.15 "12-bit ADC characteristics".

6.3.6.47 #define STM32_ADC_ADC1_DMA_PRIORITY 2

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

6.3.6.48 #define STM32_ADC_IRQ_PRIORITY 5

ADC interrupt priority level setting.

```
6.3.6.49 #define STM32_ADC_ADC1_DMA_IRQ_PRIORITY 5
```

ADC1 DMA interrupt priority level setting.

```
6.3.6.50 #define ADC_SQR1_NUM_CH( n ) (((n) - 1) << 20)
```

Number of channels in a conversion sequence.

```
6.3.6.51 #define ADC_SQR5_SQ1_N( n ) ((n) << 0)
```

1st channel in seq.

```
6.3.6.52 #define ADC_SQR5_SQ2_N( n ) ((n) << 5)
```

2nd channel in seq.

```
6.3.6.53 #define ADC_SQR5_SQ3_N( n ) ((n) << 10)
```

3rd channel in seq.

```
6.3.6.54 #define ADC_SQR5_SQ4_N( n ) ((n) << 15)
```

4th channel in seq.

```
6.3.6.55 #define ADC_SQR5_SQ5_N( n ) ((n) << 20)
```

5th channel in seq.

```
6.3.6.56 #define ADC_SQR5_SQ6_N( n ) ((n) << 25)
```

6th channel in seq.

```
6.3.6.57 #define ADC_SQR4_SQ7_N( n ) ((n) << 0)
```

7th channel in seq.

```
6.3.6.58 #define ADC_SQR4_SQ8_N( n ) ((n) << 5)
```

8th channel in seq.

```
6.3.6.59 #define ADC_SQR4_SQ9_N( n ) ((n) << 10)
```

9th channel in seq.

```
6.3.6.60 #define ADC_SQR4_SQ10_N( n ) ((n) << 15)
```

10th channel in seq.

```
6.3.6.61 #define ADC_SQR4_SQ11_N( n ) ((n) << 20)
```

11th channel in seq.

```
6.3.6.62 #define ADC_SQR4_SQ12_N( n ) ((n) << 25)
```

12th channel in seq.

```
6.3.6.63 #define ADC_SQR3_SQ13_N( n ) ((n) << 0)
```

13th channel in seq.

```
6.3.6.64 #define ADC_SQR3_SQ14_N( n ) ((n) << 5)
```

14th channel in seq.

```
6.3.6.65 #define ADC_SQR3_SQ15_N( n ) ((n) << 10)
```

15th channel in seq.

```
6.3.6.66 #define ADC_SQR3_SQ16_N( n ) ((n) << 15)
```

16th channel in seq.

```
6.3.6.67 #define ADC_SQR3_SQ17_N( n ) ((n) << 20)
```

17th channel in seq.

```
6.3.6.68 #define ADC_SQR3_SQ18_N( n ) ((n) << 25)
```

18th channel in seq.

```
6.3.6.69 #define ADC_SQR2_SQ19_N( n ) ((n) << 0)
```

19th channel in seq.

```
6.3.6.70 #define ADC_SQR2_SQ20_N( n ) ((n) << 5)
```

20th channel in seq.

```
6.3.6.71 #define ADC_SQR2_SQ21_N( n ) ((n) << 10)
```

21th channel in seq.

```
6.3.6.72 #define ADC_SQR2_SQ22_N( n ) ((n) << 15)
```

22th channel in seq.

6.3.6.73 #define ADC_SQR2_SQ23_N(n) ((n) << 20)

23th channel in seq.

6.3.6.74 #define ADC_SQR2_SQ24_N(n) ((n) << 25)

24th channel in seq.

6.3.6.75 #define ADC_SQR1_SQ25_N(n) ((n) << 0)

25th channel in seq.

6.3.6.76 #define ADC_SQR1_SQ26_N(n) ((n) << 5)

26th channel in seq.

6.3.6.77 #define ADC_SQR1_SQ27_N(n) ((n) << 10)

27th channel in seq.

6.3.6.78 #define ADC_SMPR3_SMP_AN0(n) ((n) << 0)

AN0 sampling time.

6.3.6.79 #define ADC_SMPR3_SMP_AN1(n) ((n) << 3)

AN1 sampling time.

6.3.6.80 #define ADC_SMPR3_SMP_AN2(n) ((n) << 6)

AN2 sampling time.

6.3.6.81 #define ADC_SMPR3_SMP_AN3(n) ((n) << 9)

AN3 sampling time.

6.3.6.82 #define ADC_SMPR3_SMP_AN4(n) ((n) << 12)

AN4 sampling time.

6.3.6.83 #define ADC_SMPR3_SMP_AN5(n) ((n) << 15)

AN5 sampling time.

6.3.6.84 #define ADC_SMPR3_SMP_AN6(n) ((n) << 18)

AN6 sampling time.

```
6.3.6.85 #define ADC_SMPR3_SMP_AN7( n ) ((n) << 21)
```

AN7 sampling time.

```
6.3.6.86 #define ADC_SMPR3_SMP_AN8( n ) ((n) << 24)
```

AN8 sampling time.

```
6.3.6.87 #define ADC_SMPR3_SMP_AN9( n ) ((n) << 27)
```

AN9 sampling time.

```
6.3.6.88 #define ADC_SMPR2_SMP_AN10( n ) ((n) << 0)
```

AN10 sampling time.

```
6.3.6.89 #define ADC_SMPR2_SMP_AN11( n ) ((n) << 3)
```

AN11 sampling time.

```
6.3.6.90 #define ADC_SMPR2_SMP_AN12( n ) ((n) << 6)
```

AN12 sampling time.

```
6.3.6.91 #define ADC_SMPR2_SMP_AN13( n ) ((n) << 9)
```

AN13 sampling time.

```
6.3.6.92 #define ADC_SMPR2_SMP_AN14( n ) ((n) << 12)
```

AN14 sampling time.

```
6.3.6.93 #define ADC_SMPR2_SMP_AN15( n ) ((n) << 15)
```

AN15 sampling time.

```
6.3.6.94 #define ADC_SMPR2_SMP_SENSOR( n ) ((n) << 18)
```

Temperature Sensor sampling time.

```
6.3.6.95 #define ADC_SMPR2_SMP_VREF( n ) ((n) << 21)
```

Voltage Reference sampling time.

```
6.3.6.96 #define ADC_SMPR2_SMP_AN18( n ) ((n) << 24)
```

AN18 sampling time.

```
6.3.6.97 #define ADC_SMPR2_SMP_AN19( n ) ((n) << 27)
```

AN19 sampling time.

```
6.3.6.98 #define ADC_SMPR1_SMP_AN20( n ) ((n) << 0)
```

AN20 sampling time.

```
6.3.6.99 #define ADC_SMPR1_SMP_AN21( n ) ((n) << 3)
```

AN21 sampling time.

```
6.3.6.100 #define ADC_SMPR1_SMP_AN22( n ) ((n) << 6)
```

AN22 sampling time.

```
6.3.6.101 #define ADC_SMPR1_SMP_AN23( n ) ((n) << 9)
```

AN23 sampling time.

```
6.3.6.102 #define ADC_SMPR1_SMP_AN24( n ) ((n) << 12)
```

AN24 sampling time.

```
6.3.6.103 #define ADC_SMPR1_SMP_AN25( n ) ((n) << 15)
```

AN25 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	adcp	pointer to the ADCDriver object triggering the callback
in	buffer	pointer to the most recent samples data
in	n	number of buffer rows available starting from buffer

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

ADC error callback type.

Parameters

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

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.
- `ADC_ERR_OVERFLOW` ADC overflow condition.

6.4 EXT Driver

6.4.1 Detailed Description

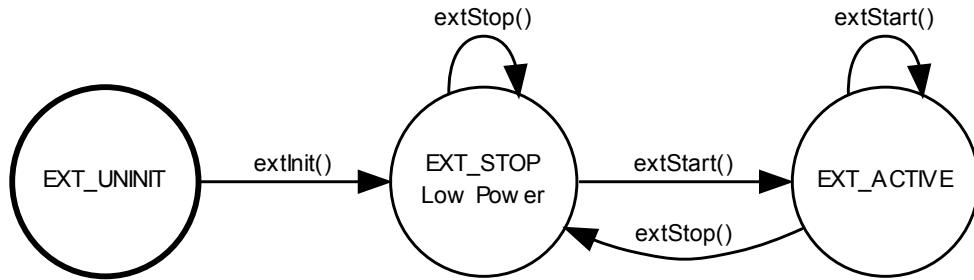
Generic EXT Driver. This module implements a generic EXT (EXTernal) driver.

Precondition

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

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).



6.4.3 EXT Operations.

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

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

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](#))

- **CH_IRQ_HANDLER** (EXTI1_IRQHandler)
 - EXTI[0] interrupt handler.*
- **CH_IRQ_HANDLER** (EXTI2_IRQHandler)
 - EXTI[1] interrupt handler.*
- **CH_IRQ_HANDLER** (EXTI3_IRQHandler)
 - EXTI[2] interrupt handler.*
- **CH_IRQ_HANDLER** (EXTI4_IRQHandler)
 - EXTI[3] interrupt handler.*
- **CH_IRQ_HANDLER** (EXTI9_5_IRQHandler)
 - EXTI[4] interrupt handler.*
 - 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_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.

6.4.4 Function Documentation

6.4.4.1 void extInit(void)

EXT Driver initialization.

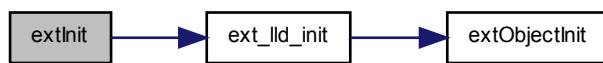
Note

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:



6.4.4.2 void extObjectInit(EXTDriver * extp)

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

Parameters

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

Function Class:

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

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



6.4.4.4 void extStop (EXTDriver * extp)

Deactivates the EXT peripheral.

Parameters

in *extp* pointer to the `EXTDriver` 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:



6.4.4.5 void extChannelEnable (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:

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

6.4.4.6 void extChannelDisable (EXTDriver * extp, expchannel_t channel)

Disables an EXT channel.

Parameters

in *extp* pointer to the `EXTDriver` object
in *channel* channel to be disabled

Function Class:

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

6.4.4.7 CH_IRQ_HANDLER (EXTI0_IRQHandler)

EXTI[0] interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.4.4.8 CH_IRQ_HANDLER (EXTI1_IRQHandler)

EXTI[1] interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.4.4.9 CH_IRQ_HANDLER (EXTI2_IRQHandler)

EXTI[2] interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.4.4.10 CH_IRQ_HANDLER (EXTI3_IRQHandler)

EXTI[3] interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.4.4.11 CH_IRQ_HANDLER (EXTI4_IRQHandler)

EXTI[4] interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.4.4.12 CH_IRQ_HANDLER (EXTI9_5_IRQHandler)

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

Function Class:

Interrupt handler, this function should not be directly invoked.

6.4.4.13 CH_IRQ_HANDLER (EXTI15_10_IRQHandler)

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

Function Class:

Interrupt handler, this function should not be directly invoked.

6.4.4.14 CH_IRQ_HANDLER (PVD_IRQHandler)

EXTI[16] interrupt handler (PVD).

Function Class:

Interrupt handler, this function should not be directly invoked.

6.4.4.15 CH_IRQ_HANDLER (RTCAlarm_IRQHandler)

EXTI[17] interrupt handler (RTC).

Function Class:

Interrupt handler, this function should not be directly invoked.

6.4.4.16 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.

6.4.4.17 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:



6.4.4.18 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.

6.4.4.19 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.

6.4.4.20 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.

6.4.4.21 void ext_lld_channel_disable (EXTDriver * extp, expchannel_t channel)

Disables an EXT channel.

Parameters

in *extp* pointer to the `EXTDriver` object
in *channel* channel to be disabled

Function Class:

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

6.4.5 Variable Documentation

6.4.5.1 EXTDriver EXTD1

EXTD1 driver identifier.

6.4.6 Define Documentation

6.4.6.1 `#define EXT_MAX_CHANNELS STM32 EXTI_NUM_CHANNELS`

Available number of EXT channels.

6.4.6.2 `#define EXT_CHANNELS_MASK ((1 << EXT_MAX_CHANNELS) - 1)`

Mask of the available channels.

6.4.6.3 `#define EXT_MODE_EXTI(m0, m1, m2, m3, m4, m5, m6, m7, m8, m9, m10, m11, m12, m13, m14, m15)`

Value:

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

EXTI-GPIO association macro.

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

6.4.6.4 `#define EXT_MODE_GPIOA 0`

GPIOA identifier.

6.4.6.5 `#define EXT_MODE_GPIOB 1`

GPIOB identifier.

6.4.6.6 `#define EXT_MODE_GPIOC 2`

GPIOC identifier.

6.4.6.7 `#define EXT_MODE_GPIOD 3`

GPIOD identifier.

6.4.6.8 `#define EXT_MODE_GPIOE 4`

GPIOE identifier.

6.4.6.9 `#define EXT_MODE_GPIOF 5`

GPIOF identifier.

6.4.6.10 `#define EXT_MODE_GPIOG 6`

GPIOG identifier.

6.4.6.11 #define EXT_MODE_GPIOH 7

GPIOH identifier.

6.4.6.12 #define EXT_MODE_GPIOI 8

GPIOI identifier.

6.4.6.13 #define STM32_EXT_EXTI0_IRQ_PRIORITY 6

EXTI0 interrupt priority level setting.

6.4.6.14 #define STM32_EXT_EXTI1_IRQ_PRIORITY 6

EXTI1 interrupt priority level setting.

6.4.6.15 #define STM32_EXT_EXTI2_IRQ_PRIORITY 6

EXTI2 interrupt priority level setting.

6.4.6.16 #define STM32_EXT_EXTI3_IRQ_PRIORITY 6

EXTI3 interrupt priority level setting.

6.4.6.17 #define STM32_EXT_EXTI4_IRQ_PRIORITY 6

EXTI4 interrupt priority level setting.

6.4.6.18 #define STM32_EXT_EXTI5_9_IRQ_PRIORITY 6

EXTI9..5 interrupt priority level setting.

6.4.6.19 #define STM32_EXT_EXTI10_15_IRQ_PRIORITY 6

EXTI15..10 interrupt priority level setting.

6.4.6.20 #define STM32_EXT_EXTI16_IRQ_PRIORITY 6

EXTI16 interrupt priority level setting.

6.4.6.21 #define STM32_EXT_EXTI17_IRQ_PRIORITY 6

EXTI17 interrupt priority level setting.

6.4.6.22 #define STM32_EXT_EXTI18_IRQ_PRIORITY 6

EXTI18 interrupt priority level setting.

6.4.6.23 `#define STM32_EXT EXTI19_IRQ_PRIORITY 6`

EXTI19 interrupt priority level setting.

6.4.6.24 `#define STM32_EXT EXTI20_IRQ_PRIORITY 6`

EXTI20 interrupt priority level setting.

6.4.6.25 `#define STM32_EXT EXTI21_IRQ_PRIORITY 6`

EXTI21 interrupt priority level setting.

6.4.6.26 `#define STM32_EXT EXTI22_IRQ_PRIORITY 6`

EXTI22 interrupt priority level setting.

6.4.7 Typedef Documentation

6.4.7.1 `typedef uint32_t expchannel_t`

EXT channel identifier.

6.4.7.2 `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

6.5 GPT Driver

6.5.1 Detailed Description

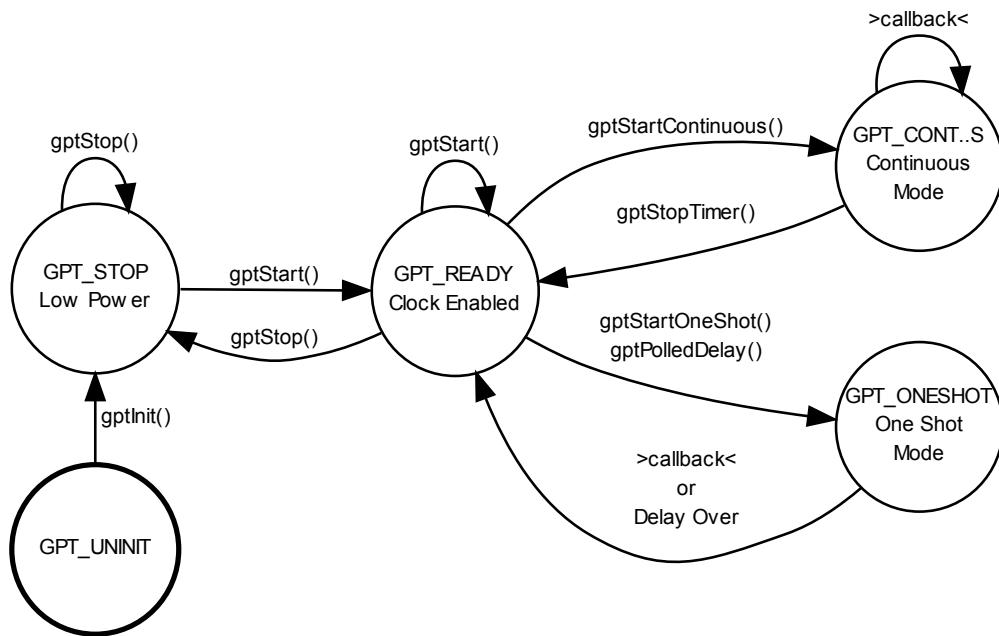
Generic GPT Driver. This module implements a generic GPT (General Purpose Timer) driver. The timer can be programmed in order to trigger callbacks after a specified time period or continuously with a specified interval.

Precondition

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

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 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.

6.5.4 Function Documentation

6.5.4.1 void gptInit (void)

GPT Driver initialization.

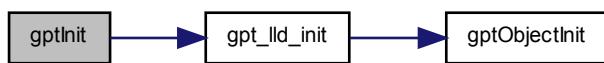
Note

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:



6.5.4.2 void gptObjectInit (GPTDriver * gptp)

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

Parameters

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

Function Class:

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

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

Configures and activates the GPT peripheral.

Parameters

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

Function Class:

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

Here is the call graph for this function:



6.5.4.4 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:



6.5.4.5 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:



6.5.4.6 void gptStartContinuous(GPTDriver * gptp, gptcnt_t interval)

Starts the timer in continuous mode.

Parameters

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

Function Class:

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:



6.5.4.7 void gptStartOneShot(GPTDriver * gptp, gptcnt_t interval)

Starts the timer in one shot mode.

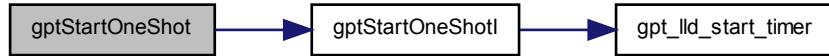
Parameters

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

Function Class:

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

Here is the call graph for this function:



6.5.4.8 void gptStartOneShotl (GPTDriver * *gptp*, gptcnt_t *interval*)

Starts the timer in one shot mode.

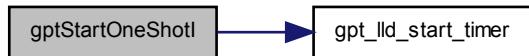
Parameters

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

Function Class:

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

Here is the call graph for this function:



6.5.4.9 void gptStopTimer (GPTDriver * *gptp*)

Stops the timer.

Parameters

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

Function Class:

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

Here is the call graph for this function:



6.5.4.10 void gptStopTimerI (GPTDriver * *gptp*)

Stops the timer.

Parameters

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

Function Class:

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

Here is the call graph for this function:



6.5.4.11 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 GPTDriver object
in	<i>interval</i> time interval in ticks

Function Class:

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

Here is the call graph for this function:



6.5.4.12 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:



6.5.4.13 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.

6.5.4.14 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.

6.5.4.15 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.

6.5.4.16 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.

6.5.4.17 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.

6.5.5 Variable Documentation

6.5.5.1 GPTDriver GPTD1

GPTD1 driver identifier.

Note

The driver GPTD1 allocates the complex timer TIM1 when enabled.

6.5.5.2 GPTDriver GPTD2

GPTD2 driver identifier.

Note

The driver GPTD2 allocates the timer TIM2 when enabled.

6.5.5.3 GPTDriver GPTD3

GPTD3 driver identifier.

Note

The driver GPTD3 allocates the timer TIM3 when enabled.

6.5.5.4 GPTDriver GPTD4

GPTD4 driver identifier.

Note

The driver GPTD4 allocates the timer TIM4 when enabled.

6.5.5.5 GPTDriver GPTD5

GPTD5 driver identifier.

Note

The driver GPTD5 allocates the timer TIM5 when enabled.

6.5.5.6 GPTDriver GPTD8

GPTD8 driver identifier.

Note

The driver GPTD8 allocates the timer TIM8 when enabled.

6.5.6 Define Documentation

6.5.6.1 #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.

6.5.6.2 #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.

6.5.6.3 #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.

6.5.6.4 #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.

6.5.6.5 #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.

6.5.6.6 #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.

6.5.6.7 #define STM32_GPT_TIM1_IRQ_PRIORITY 7

GPTD1 interrupt priority level setting.

6.5.6.8 #define STM32_GPT_TIM2_IRQ_PRIORITY 7

GPTD2 interrupt priority level setting.

6.5.6.9 #define STM32_GPT_TIM3_IRQ_PRIORITY 7

GPTD3 interrupt priority level setting.

6.5.6.10 #define STM32_GPT_TIM4_IRQ_PRIORITY 7

GPTD4 interrupt priority level setting.

```
6.5.6.11 #define STM32_GPT_TIM5_IRQ_PRIORITY 7
```

GPTD5 interrupt priority level setting.

```
6.5.6.12 #define STM32_GPT_TIM8_IRQ_PRIORITY 7
```

GPTD5 interrupt priority level setting.

6.5.7 Typedef Documentation

```
6.5.7.1 typedef struct GPTDriver GPTDriver
```

Type of a structure representing a GPT driver.

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

GPT notification callback type.

Parameters

in *gptp* pointer to a `GPTDriver` object

```
6.5.7.3 typedef uint32_t gptfreq_t
```

GPT frequency type.

```
6.5.7.4 typedef uint16_t gptcnt_t
```

GPT counter type.

6.5.8 Enumeration Type Documentation

```
6.5.8.1 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.

6.6 HAL Driver

6.6.1 Detailed Description

Hardware Abstraction Layer. The HAL (Hardware Abstraction Layer) driver performs the system initialization and includes the platform support code shared by the other drivers. This driver does contain any API function except

for a general initialization function `halInit()` that must be invoked before any HAL service can be used, usually the HAL initialization should be performed immediately before the kernel initialization.

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

Data Structures

- struct `stm32_tim_t`
STM32 TIM registers block.

Functions

- void `halInit(void)`
HAL initialization.
- bool_t `hallsCounterWithin(halrcnt_t start, halrcnt_t end)`
Realtime window test.
- void `halPolledDelay(halrcnt_t ticks)`
Polled delay.
- void `hal_lld_init(void)`
Low level HAL driver initialization.
- void `stm32_clock_init(void)`
STM32L1xx voltage, 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.

Platform identification

- #define `PLATFORM_NAME` "STM32L1 Ultra Low Power Medium Density"

Internal clock sources

- #define `STM32_HSICLK` 16000000
- #define `STM32_LSICLK` 38000

PWR_CR register bits definitions

- #define STM32_VOS_MASK (3 << 11)
- #define STM32_VOS_1P8 (1 << 11)
- #define STM32_VOS_1P5 (2 << 11)
- #define STM32_VOS_1P2 (3 << 11)
- #define STM32_PLS_MASK (7 << 5)
- #define STM32_PLSLEV0 (0 << 5)
- #define STM32_PLSLEV1 (1 << 5)
- #define STM32_PLSLEV2 (2 << 5)
- #define STM32_PLSLEV3 (3 << 5)
- #define STM32_PLSLEV4 (4 << 5)
- #define STM32_PLSLEV5 (5 << 5)
- #define STM32_PLSLEV6 (6 << 5)
- #define STM32_PLSLEV7 (7 << 5)

RCC_CR register bits definitions

- #define STM32_RTCPRE_MASK (3 << 29)
- #define STM32_RTCPRE_DIV2 (0 << 29)
- #define STM32_RTCPRE_DIV4 (1 << 29)
- #define STM32_RTCPRE_DIV8 (2 << 29)
- #define STM32_RTCPRE_DIV16 (3 << 29)

RCC_CFGR register bits definitions

- #define STM32_SW_MSI (0 << 0)
- #define STM32_SW_HSI (1 << 0)
- #define STM32_SW_HSE (2 << 0)
- #define STM32_SW_PLL (3 << 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_PLLSRC_HSI (0 << 16)
- #define STM32_PLLSRC_HSE (1 << 16)
- #define STM32_MCOSEL_NOCLOCK (0 << 24)
- #define STM32_MCOSEL_SYSCLK (1 << 24)
- #define STM32_MCOSEL_HSI (2 << 24)

- #define STM32_MCOSEL_MSI (3 << 24)
- #define STM32_MCOSEL_HSE (4 << 24)
- #define STM32_MCOSEL_PLL (5 << 24)
- #define STM32_MCOSEL_LSI (6 << 24)
- #define STM32_MCOSEL_LSE (7 << 24)
- #define STM32_MCOPRE_DIV1 (0 << 28)
- #define STM32_MCOPRE_DIV2 (1 << 28)
- #define STM32_MCOPRE_DIV4 (2 << 28)
- #define STM32_MCOPRE_DIV8 (3 << 28)
- #define STM32_MCOPRE_DIV16 (4 << 28)

RCC_ICSCR register bits definitions

- #define STM32_MSIRANGE_MASK (7 << 13)
- #define STM32_MSIRANGE_64K (0 << 13)
- #define STM32_MSIRANGE_128K (1 << 13)
- #define STM32_MSIRANGE_256K (2 << 13)
- #define STM32_MSIRANGE_512K (3 << 13)
- #define STM32_MSIRANGE_1M (4 << 13)
- #define STM32_MSIRANGE_2M (5 << 13)
- #define STM32_MSIRANGE_4M (6 << 13)

RCC_CSR register bits definitions

- #define STM32_RTCSEL_MASK (3 << 16)
- #define STM32_RTCSEL_NOCLOCK (0 << 16)
- #define STM32_RTCSEL_LSE (1 << 16)
- #define STM32_RTCSEL_LSI (2 << 16)
- #define STM32_RTCSEL_HSEDIV (3 << 16)

STM32L1xx 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_CAN_MAX_FILTERS 0
- #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 23
- #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 TRUE
- #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_RTC_HAS_SUBSECONDS FALSE
- #define STM32_RTC_IS_CALENDAR 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 FALSE
- #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 TRUE
- #define STM32_HAS_TIM10 TRUE
- #define STM32_HAS_TIM11 TRUE
- #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

IRQ VECTOR names

- #define WWDG_IRQHandler Vector40
- #define PVD_IRQHandler Vector44
- #define TAMPER_STAMP_IRQHandler Vector48
- #define RTC_WKUP_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_IRQHandler Vector88
- #define USB_HP_IRQHandler Vector8C
- #define USB_LP_IRQHandler Vector90
- #define DAC_IRQHandler Vector94

- #define COMP_IRQHandler Vector98
- #define EXTI9_5_IRQHandler Vector9C
- #define TIM9_IRQHandler VectorA0
- #define TIM10_IRQHandler VectorA4
- #define TIM11_IRQHandler VectorA8
- #define LCD_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 TIM6_IRQHandler VectorEC
- #define TIM7_IRQHandler VectorF0

Configuration options

- #define STM32_NO_INIT FALSE
Disables the PWR/RCC initialization in the HAL.
- #define STM32_VOS STM32_VOS_1P8
Core voltage selection.
- #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 TRUE
Enables or disables the LSI clock source.
- #define STM32_HSE_ENABLED FALSE
Enables or disables the HSE clock source.
- #define STM32_LSE_ENABLED FALSE
Enables or disables the LSE clock source.
- #define STM32_ADC_CLOCK_ENABLED TRUE
ADC clock setting.
- #define STM32_USB_CLOCK_ENABLED TRUE
USB clock setting.
- #define STM32_MSIRANGE STM32_MSIRANGE_2M
MSI frequency setting.
- #define STM32_SW STM32_SW_PLL
Main clock source selection.
- #define STM32_PLLSRC STM32_PLLSRC_HSI
Clock source for the PLL.

- `#define STM32_PLLMUL_VALUE 6`
PLL multiplier value.
- `#define STM32_PLLDIV_VALUE 3`
PLL divider 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_MCOSEL STM32_MCOSEL_NOCLOCK`
MCO clock source.
- `#define STM32_MCOPRE STM32_MCOPRE_DIV1`
MCO divider setting.
- `#define STM32_RTCSEL STM32_RTCSEL_LSE`
RTC/LCD clock source.
- `#define STM32_RTCPRE STM32_RTCPRE_DIV2`
HSE divider toward RTC setting.

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 HAL_IMPLEMENT_COUNTERS TRUE`
Defines the support for realtime counters in the HAL.
- `#define STM32_HSECLK_MAX 32000000`
Maximum HSE clock frequency at current voltage setting.
- `#define STM32_SYSCLK_MAX 32000000`
Maximum SYCLK clock frequency at current voltage setting.
- `#define STM32_PLLVCO_MAX 96000000`
Maximum VCO clock frequency at current voltage setting.
- `#define STM32_PLLVCO_MIN 6000000`
Minimum VCO clock frequency at current voltage setting.
- `#define STM32_PCLK1_MAX 32000000`
Maximum APB1 clock frequency.
- `#define STM32_PCLK2_MAX 32000000`

- `#define STM32_0WS_THRESHOLD 16000000`
Maximum APB2 clock frequency.
- `#define STM32_HSI_AVAILABLE TRUE`
HSI availability at current voltage settings.
- `#define STM32_ACTIVATE_PLL TRUE`
PLL activation flag.
- `#define STM32_PLLMUL (0 << 18)`
PLLMUL field.
- `#define STM32_PLLDIV (1 << 22)`
PLLDIV field.
- `#define STM32_PLLCLKIN STM32_HSECLK`
PLL input clock frequency.
- `#define STM32_PLLVCO (STM32_PLLCLKIN * STM32_PLLMUL_VALUE)`
PLL VCO frequency.
- `#define STM32_PLLCLKOUT (STM32_PLLVCO / STM32_PLLDIV_VALUE)`
PLL output clock frequency.
- `#define STM32_MSICLK 2100000`
MSI frequency.
- `#define STM32_SYSCLK 2100000`
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 STM_MCODIVCLK 0`
MCO divider clock.
- `#define STM_MCOCLK STM_MCODIVCLK`
MCO output pin clock.
- `#define STM32_HSEDIVCLK (STM32_HSECLK / 2)`
HSE divider toward RTC clock.
- `#define STM_RTCCLK 0`
RTC/LCD clock.
- `#define STM32_ADCCLK STM32_HSICLK`
ADC frequency.
- `#define STM32_USBCLK (STM32_PLLVCO / 2)`
USB frequency.
- `#define STM32_TIMCLK1 (STM32_PCLK1 * 1)`
Timers 2, 3, 4, 6, 7 clock.
- `#define STM32_TIMCLK2 (STM32_PCLK2 * 1)`
Timers 9, 10, 11 clock.
- `#define STM32_FLASHBITS1 0x00000000`
Flash settings.
- `#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.

6.6.2 Function Documentation

6.6.2.1 void hallinit(void)

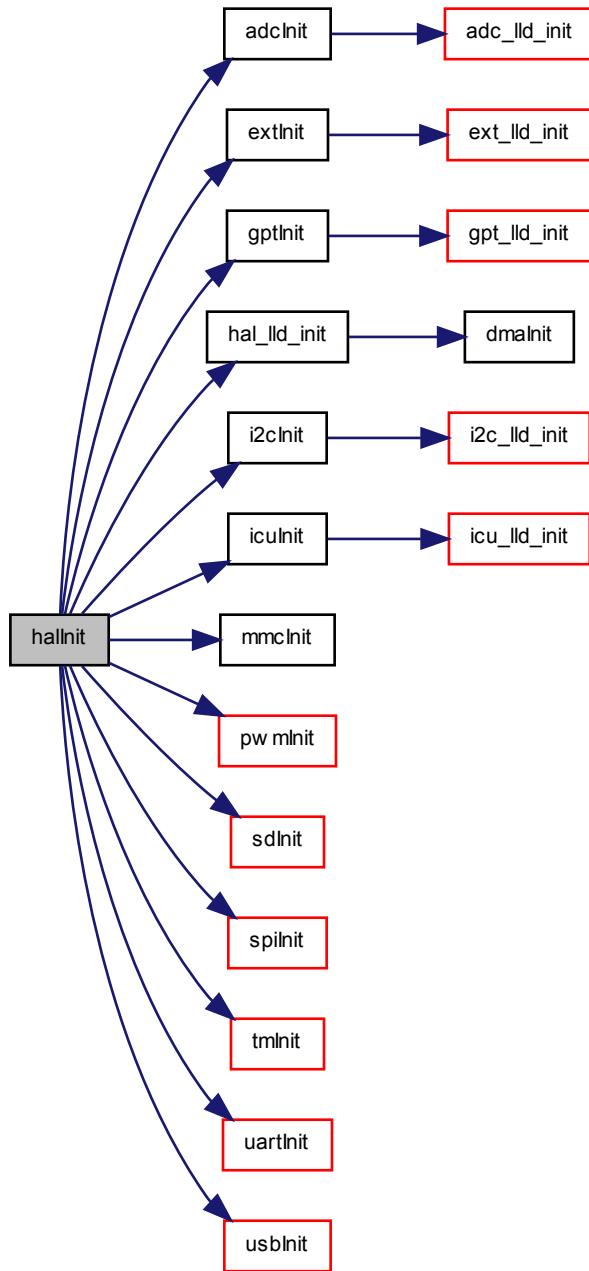
HAL initialization.

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:



6.6.2.2 bool_t hallsCounterWithin (**halrtcnt_t start, halrtcnt_t 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.
 This is an optional service that could not be implemented in all HAL implementations.
 This function can be called from any context.

Example 1

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

```
halrcnt_t start = halGetCounterValue();
halrcnt_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.

```
halrcnt_t start = halGetCounterValue();
halrcnt_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

<i>TRUE</i>	current time within the specified time window.
<i>FALSE</i>	current time not within the specified time window.

Function Class:

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

6.6.2.3 void halPolledDelay (*halrcnt_t ticks*)

Polled delay.

Note

The real delays is always few cycles in excess of the specified value.
 This is an optional service that could not be implemented in all HAL implementations.
 This function can be called from any context.

Parameters

in	<i>ticks</i>	number of ticks
----	--------------	-----------------

Function Class:

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

Here is the call graph for this function:



6.6.2.4 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:



6.6.2.5 void stm32_clock_init(void)

STM32L1xx voltage, 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. Clocks and internal voltage initialization.

6.6.3 Define Documentation

6.6.3.1 #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.

6.6.3.2 #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.

6.6.3.3 #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.

6.6.3.4 #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.
This function can be called from any context.

Returns

The value of the system free running counter of type halrtcnt_t.

Function Class:

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

6.6.3.5 #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.
This function can be called from any context.

Returns

The realtime counter frequency of type halclock_t.

Function Class:

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

6.6.3.6 #define HAL_IMPLEMENT_COUNTERS TRUE

Defines the support for realtime counters in the HAL.

6.6.3.7 #define STM32_HSICLK 16000000

High speed internal clock.

6.6.3.8 #define STM32_LSIGCLK 38000

Low speed internal clock.

6.6.3.9 #define STM32_VOS_MASK (3 << 11)

Core voltage mask.

6.6.3.10 #define STM32_VOS_1P8 (1 << 11)

Core voltage 1.8 Volts.

6.6.3.11 #define STM32_VOS_1P5 (2 << 11)

Core voltage 1.5 Volts.

6.6.3.12 #define STM32_VOS_1P2 (3 << 11)

Core voltage 1.2 Volts.

6.6.3.13 #define STM32_PLS_MASK (7 << 5)

PLS bits mask.

6.6.3.14 #define STM32_PLS_LEV0 (0 << 5)

PVD level 0.

6.6.3.15 #define STM32_PLS_LEV1 (1 << 5)

PVD level 1.

6.6.3.16 #define STM32_PLS_LEV2 (2 << 5)

PVD level 2.

6.6.3.17 #define STM32_PLS_LEV3 (3 << 5)

PVD level 3.

6.6.3.18 #define STM32_PLS_LEV4 (4 << 5)

PVD level 4.

6.6.3.19 #define STM32_PLS_LEV5 (5 << 5)

PVD level 5.

6.6.3.20 #define STM32_PLS_LEV6 (6 << 5)

PVD level 6.

6.6.3.21 #define STM32_PLS_LEV7 (7 << 5)

PVD level 7.

6.6.3.22 #define STM32_RTCPRE_MASK (3 << 29)

RTCPRE mask.

6.6.3.23 `#define STM32_RTCPRE_DIV2 (0 << 29)`

HSE divided by 2.

6.6.3.24 `#define STM32_RTCPRE_DIV4 (1 << 29)`

HSE divided by 4.

6.6.3.25 `#define STM32_RTCPRE_DIV8 (2 << 29)`

HSE divided by 2.

6.6.3.26 `#define STM32_RTCPRE_DIV16 (3 << 29)`

HSE divided by 16.

6.6.3.27 `#define STM32_SW_MSI (0 << 0)`

SYSCLK source is MSI.

6.6.3.28 `#define STM32_SW_HSI (1 << 0)`

SYSCLK source is HSI.

6.6.3.29 `#define STM32_SW_HSE (2 << 0)`

SYSCLK source is HSE.

6.6.3.30 `#define STM32_SW_PLL (3 << 0)`

SYSCLK source is PLL.

6.6.3.31 `#define STM32_HPRE_DIV1 (0 << 4)`

SYSCLK divided by 1.

6.6.3.32 `#define STM32_HPRE_DIV2 (8 << 4)`

SYSCLK divided by 2.

6.6.3.33 `#define STM32_HPRE_DIV4 (9 << 4)`

SYSCLK divided by 4.

6.6.3.34 `#define STM32_HPRE_DIV8 (10 << 4)`

SYSCLK divided by 8.

6.6.3.35 `#define STM32_HPRE_DIV16 (11 << 4)`

SYSCLK divided by 16.

6.6.3.36 `#define STM32_HPRE_DIV64 (12 << 4)`

SYSCLK divided by 64.

6.6.3.37 `#define STM32_HPRE_DIV128 (13 << 4)`

SYSCLK divided by 128.

6.6.3.38 `#define STM32_HPRE_DIV256 (14 << 4)`

SYSCLK divided by 256.

6.6.3.39 `#define STM32_HPRE_DIV512 (15 << 4)`

SYSCLK divided by 512.

6.6.3.40 `#define STM32_PPRE1_DIV1 (0 << 8)`

HCLK divided by 1.

6.6.3.41 `#define STM32_PPRE1_DIV2 (4 << 8)`

HCLK divided by 2.

6.6.3.42 `#define STM32_PPRE1_DIV4 (5 << 8)`

HCLK divided by 4.

6.6.3.43 `#define STM32_PPRE1_DIV8 (6 << 8)`

HCLK divided by 8.

6.6.3.44 `#define STM32_PPRE1_DIV16 (7 << 8)`

HCLK divided by 16.

6.6.3.45 `#define STM32_PPRE2_DIV1 (0 << 11)`

HCLK divided by 1.

6.6.3.46 `#define STM32_PPRE2_DIV2 (4 << 11)`

HCLK divided by 2.

6.6.3.47 #define STM32_PPREG2_DIV4 (5 << 11)

HCLK divided by 4.

6.6.3.48 #define STM32_PPREG2_DIV8 (6 << 11)

HCLK divided by 8.

6.6.3.49 #define STM32_PPREG2_DIV16 (7 << 11)

HCLK divided by 16.

6.6.3.50 #define STM32_PLLSRC_HSI (0 << 16)

PLL clock source is HSI.

6.6.3.51 #define STM32_PLLSRC_HSE (1 << 16)

PLL clock source is HSE.

6.6.3.52 #define STM32_MCOSEL_NOCLOCK (0 << 24)

No clock on MCO pin.

6.6.3.53 #define STM32_MCOSEL_SYSCLK (1 << 24)

SYSCLK on MCO pin.

6.6.3.54 #define STM32_MCOSEL_HSI (2 << 24)

HSI clock on MCO pin.

6.6.3.55 #define STM32_MCOSEL_MSI (3 << 24)

MSI clock on MCO pin.

6.6.3.56 #define STM32_MCOSEL_HSE (4 << 24)

HSE clock on MCO pin.

6.6.3.57 #define STM32_MCOSEL_PLL (5 << 24)

PLL clock on MCO pin.

6.6.3.58 #define STM32_MCOSEL_LSI (6 << 24)

LSI clock on MCO pin.

6.6.3.59 #define STM32_MCOSEL_LSE (7 << 24)

LSE clock on MCO pin.

6.6.3.60 #define STM32_MCOPRE_DIV1 (0 << 28)

MCO divided by 1.

6.6.3.61 #define STM32_MCOPRE_DIV2 (1 << 28)

MCO divided by 1.

6.6.3.62 #define STM32_MCOPRE_DIV4 (2 << 28)

MCO divided by 1.

6.6.3.63 #define STM32_MCOPRE_DIV8 (3 << 28)

MCO divided by 1.

6.6.3.64 #define STM32_MCOPRE_DIV16 (4 << 28)

MCO divided by 1.

6.6.3.65 #define STM32_MSIRANGE_MASK (7 << 13)

MSIRANGE field mask.

6.6.3.66 #define STM32_MSIRANGE_64K (0 << 13)

64kHz nominal.

6.6.3.67 #define STM32_MSIRANGE_128K (1 << 13)

128kHz nominal.

6.6.3.68 #define STM32_MSIRANGE_256K (2 << 13)

256kHz nominal.

6.6.3.69 #define STM32_MSIRANGE_512K (3 << 13)

512kHz nominal.

6.6.3.70 #define STM32_MSIRANGE_1M (4 << 13)

1MHz nominal.

6.6.3.71 #define STM32_MSIRANGE_2M (5 << 13)

2MHz nominal.

6.6.3.72 #define STM32_MSIRANGE_4M (6 << 13)

4MHz nominal

6.6.3.73 #define STM32_RTCSEL_MASK (3 << 16)

RTC source mask.

6.6.3.74 #define STM32_RTCSEL_NOCLOCK (0 << 16)

No RTC source.

6.6.3.75 #define STM32_RTCSEL_LSE (1 << 16)

RTC source is LSE.

6.6.3.76 #define STM32_RTCSEL_LSI (2 << 16)

RTC source is LSI.

6.6.3.77 #define STM32_RTCSEL_HSEDIV (3 << 16)

RTC source is HSE divided.

6.6.3.78 #define WWDG_IRQHandler Vector40

Window Watchdog.

6.6.3.79 #define PVD_IRQHandler Vector44

PVD through EXTI Line detect.

6.6.3.80 #define TAMPER_STAMP_IRQHandler Vector48

Tamper and Time Stamp through EXTI.

6.6.3.81 #define RTC_WKUP_IRQHandler Vector4C

RTC Wakeup Timer through EXTI.

6.6.3.82 #define FLASH_IRQHandler Vector50

Flash.

6.6.3.83 #define RCC_IRQHandler Vector54

RCC.

6.6.3.84 #define EXTI0_IRQHandler Vector58

EXTI Line 0.

6.6.3.85 #define EXTI1_IRQHandler Vector5C

EXTI Line 1.

6.6.3.86 #define EXTI2_IRQHandler Vector60

EXTI Line 2.

6.6.3.87 #define EXTI3_IRQHandler Vector64

EXTI Line 3.

6.6.3.88 #define EXTI4_IRQHandler Vector68

EXTI Line 4.

6.6.3.89 #define DMA1_Ch1_IRQHandler Vector6C

DMA1 Channel 1.

6.6.3.90 #define DMA1_Ch2_IRQHandler Vector70

DMA1 Channel 2.

6.6.3.91 #define DMA1_Ch3_IRQHandler Vector74

DMA1 Channel 3.

6.6.3.92 #define DMA1_Ch4_IRQHandler Vector78

DMA1 Channel 4.

6.6.3.93 #define DMA1_Ch5_IRQHandler Vector7C

DMA1 Channel 5.

6.6.3.94 #define DMA1_Ch6_IRQHandler Vector80

DMA1 Channel 6.

6.6.3.95 #define DMA1_Ch7_IRQHandler Vector84

DMA1 Channel 7.

6.6.3.96 #define ADC1_IRQHandler Vector88

ADC1.

6.6.3.97 #define USB_HP_IRQHandler Vector8C

USB High Priority.

6.6.3.98 #define USB_LP_IRQHandler Vector90

USB Low Priority.

6.6.3.99 #define DAC_IRQHandler Vector94

DAC.

6.6.3.100 #define COMP_IRQHandler Vector98

Comparator through EXTI.

6.6.3.101 #define EXTI9_5_IRQHandler Vector9C

EXTI Line 9..5.

6.6.3.102 #define TIM9_IRQHandler VectorA0

TIM9.

6.6.3.103 #define TIM10_IRQHandler VectorA4

TIM10.

6.6.3.104 #define TIM11_IRQHandler VectorA8

TIM11.

6.6.3.105 #define LCD_IRQHandler VectorAC

LCD.

6.6.3.106 #define TIM2_IRQHandler VectorB0

TIM2.

6.6.3.107 #define TIM3_IRQHandler VectorB4

TIM3.

6.6.3.108 #define TIM4_IRQHandler VectorB8

TIM4.

6.6.3.109 #define I2C1_EV_IRQHandler VectorBC

I2C1 Event.

6.6.3.110 #define I2C1_ER_IRQHandler VectorC0

I2C1 Error.

6.6.3.111 #define I2C2_EV_IRQHandler VectorC4

I2C2 Event.

6.6.3.112 #define I2C2_ER_IRQHandler VectorC8

I2C2 Error.

6.6.3.113 #define SPI1_IRQHandler VectorCC

SPI1.

6.6.3.114 #define SPI2_IRQHandler VectorD0

SPI2.

6.6.3.115 #define USART1_IRQHandler VectorD4

USART1.

6.6.3.116 #define USART2_IRQHandler VectorD8

USART2.

6.6.3.117 #define USART3_IRQHandler VectorDC

USART3.

6.6.3.118 #define EXTI15_10_IRQHandler VectorE0

EXTI Line 15..10.

6.6.3.119 #define RTC_Alarm_IRQHandler VectorE4

RTC Alarm through EXTI.

6.6.3.120 #define USB_FS_WKUP_IRQHandler VectorE8

USB Wakeup from suspend.

6.6.3.121 #define TIM6_IRQHandler VectorEC

TIM6.

6.6.3.122 #define TIM7_IRQHandler VectorF0

TIM7.

6.6.3.123 #define STM32_NO_INIT FALSE

Disables the PWR/RCC initialization in the HAL.

6.6.3.124 #define STM32_VOS STM32_VOS_1P8

Core voltage selection.

Note

This setting affects all the performance and clock related settings, the maximum performance is only obtainable selecting the maximum voltage.

6.6.3.125 #define STM32_PVD_ENABLE FALSE

Enables or disables the programmable voltage detector.

6.6.3.126 #define STM32_PLS STM32_PLS_LEV0

Sets voltage level for programmable voltage detector.

6.6.3.127 #define STM32_HSI_ENABLED TRUE

Enables or disables the HSI clock source.

6.6.3.128 #define STM32_LSI_ENABLED TRUE

Enables or disables the LSI clock source.

6.6.3.129 #define STM32_HSE_ENABLED FALSE

Enables or disables the HSE clock source.

6.6.3.130 #define STM32_LSE_ENABLED FALSE

Enables or disables the LSE clock source.

6.6.3.131 #define STM32_ADC_CLOCK_ENABLED TRUE

ADC clock setting.

6.6.3.132 #define STM32_USB_CLOCK_ENABLED TRUE

USB clock setting.

6.6.3.133 #define STM32_MSIRANGE STM32_MSIRANGE_2M

MSI frequency setting.

6.6.3.134 #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 32MHz system clock from the internal 16MHz HSI clock.

6.6.3.135 #define STM32_PLLSRC STM32_PLLSRC_HSI

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 32MHz system clock from the internal 16MHz HSI clock.

6.6.3.136 #define STM32_PLLMUL_VALUE 6

PLL multiplier value.

Note

The allowed values are 3, 4, 6, 8, 12, 16, 32, 48.

The default value is calculated for a 32MHz system clock from the internal 16MHz HSI clock.

6.6.3.137 #define STM32_PLLDIV_VALUE 3

PLL divider value.

Note

The allowed values are 2, 3, 4.

The default value is calculated for a 32MHz system clock from the internal 16MHz HSI clock.

6.6.3.138 #define STM32_HPRE STM32_HPRE_DIV1

AHB prescaler value.

Note

The default value is calculated for a 32MHz system clock from the internal 16MHz HSI clock.

6.6.3.139 #define STM32_PPRE1 STM32_PPRE1_DIV1

APB1 prescaler value.

6.6.3.140 #define STM32_PPRE2 STM32_PPRE2_DIV1

APB2 prescaler value.

6.6.3.141 #define STM32_MCOSEL STM32_MCOSEL_NOCLOCK

MCO clock source.

6.6.3.142 #define STM32_MCOPRE STM32_MCOPRE_DIV1

MCO divider setting.

6.6.3.143 #define STM32_RTCSEL STM32_RTCSEL_LSE

RTC/LCD clock source.

6.6.3.144 #define STM32_RTCPRE STM32_RTCPRE_DIV2

HSE divider toward RTC setting.

6.6.3.145 #define STM32_HSECLK_MAX 32000000

Maximum HSE clock frequency at current voltage setting.

6.6.3.146 #define STM32_SYSCLK_MAX 32000000

Maximum SYSCLK clock frequency at current voltage setting.

6.6.3.147 #define STM32_PLLVCO_MAX 96000000

Maximum VCO clock frequency at current voltage setting.

6.6.3.148 #define STM32_PLLVCO_MIN 6000000

Minimum VCO clock frequency at current voltage setting.

6.6.3.149 #define STM32_PCLK1_MAX 32000000

Maximum APB1 clock frequency.

6.6.3.150 #define STM32_PCLK2_MAX 32000000

Maximum APB2 clock frequency.

6.6.3.151 #define STM32_0WS_THRESHOLD 16000000

Maximum frequency not requiring a wait state for flash accesses.

6.6.3.152 #define STM32_HSI_AVAILABLE TRUE

HSI availability at current voltage settings.

6.6.3.153 #define STM32_ACTIVATE_PLL TRUE

PLL activation flag.

6.6.3.154 #define STM32_PLLMUL (0 << 18)

PLLMUL field.

6.6.3.155 #define STM32_PLLDIV (1 << 22)

PLLDIV field.

6.6.3.156 #define STM32_PLLCLKIN STM32_HSECLK

PLL input clock frequency.

6.6.3.157 #define STM32_PLLVCO (STM32_PLLCLKIN * STM32_PLLMUL_VALUE)

PLL VCO frequency.

6.6.3.158 #define STM32_PLLCLKOUT (STM32_PLLVCO / STM32_PLLDIV_VALUE)

PLL output clock frequency.

6.6.3.159 #define STM32_MSICLK 2100000

MSI frequency.

Note

Values are taken from the STM8Lxx datasheet.

6.6.3.160 #define STM32_SYSCLK 2100000

System clock source.

6.6.3.161 #define STM32_HCLK (STM32_SYSCLK / 1)

AHB frequency.

6.6.3.162 #define STM32_PCLK1 (STM32_HCLK / 1)

APB1 frequency.

6.6.3.163 #define STM32_PCLK2 (STM32_HCLK / 1)

APB2 frequency.

6.6.3.164 #define STM_MCODIVCLK 0

MCO divider clock.

6.6.3.165 #define STM_MCOCLK STM_MCODIVCLK

MCO output pin clock.

6.6.3.166 #define STM32_HSEDIVCLK (STM32_HSECLK / 2)

HSE divider toward RTC clock.

6.6.3.167 #define STM_RTCCLK 0

RTC/LCD clock.

6.6.3.168 #define STM32_ADCCLK STM32_HSICLK

ADC frequency.

6.6.3.169 #define STM32_USBCLK (STM32_PLLVCO / 2)

USB frequency.

6.6.3.170 #define STM32_TIMCLK1 (STM32_PCLK1 * 1)

Timers 2, 3, 4, 6, 7 clock.

6.6.3.171 #define STM32_TIMCLK2 (STM32_PCLK2 * 1)

Timers 9, 10, 11 clock.

```
6.6.3.172 #define STM32_FLASHBITS1 0x00000000
```

Flash settings.

```
6.6.3.173 #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 halrtcnt_t.

Function Class:

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

```
6.6.3.174 #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.

6.6.4 Typedef Documentation

```
6.6.4.1 typedef uint32_t halclock_t
```

Type representing a system clock frequency.

```
6.6.4.2 typedef uint32_t halrtcnt_t
```

Type of the realtime free counter value.

6.7 I2C Driver

6.7.1 Detailed Description

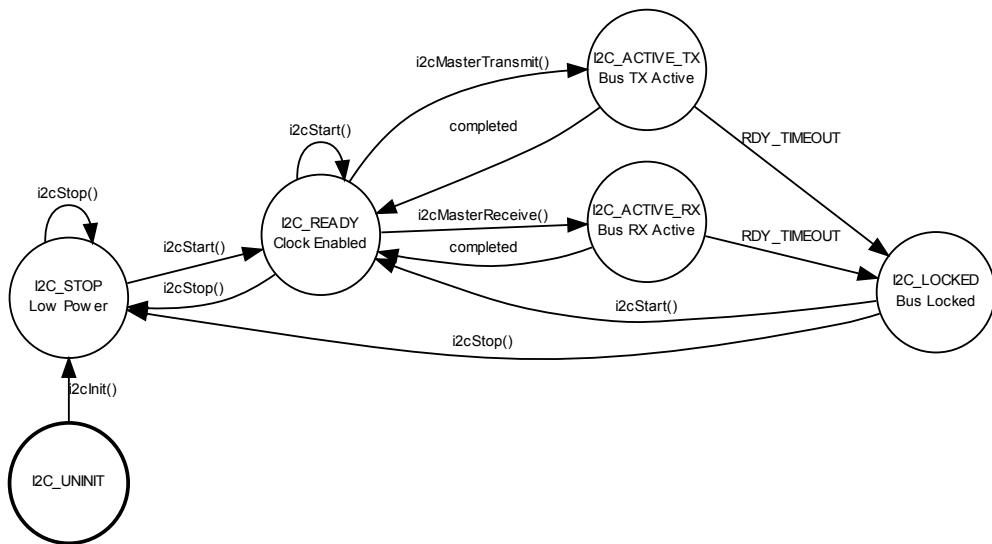
Generic I2C Driver. This module implements a generic I2C (Inter-Integrated Circuit) driver.

Precondition

In order to use the I2C driver the HAL_USE_I2C option must be enabled in [halconf.h](#).

6.7.2 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.

6.7.3 Function Documentation

6.7.3.1 void i2cInit(void)

I2C Driver initialization.

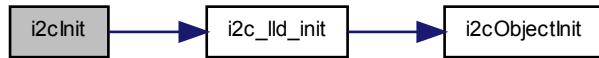
Note

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:



6.7.3.2 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.

6.7.3.3 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:



6.7.3.4 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:

**6.7.3.5 i2cflags_t i2cGetErrors (I2CDriver * i2cp)**

Returns the errors mask associated to the previous operation.

Parameters

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

Returns

The errors mask.

Function Class:

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

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

Returns

The operation status.

Return values

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:



6.7.3.7 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 I2CDriver 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"> • <i>TIME_INFINITE</i> no timeout.

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:



6.7.3.8 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.

6.7.3.9 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.

6.7.3.10 CH_IRQ_HANDLER (I2C1_EV_IRQHandler)

I2C1 event interrupt handler.

Function Class:

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

6.7.3.11 CH_IRQ_HANDLER (I2C1_ER_IRQHandler)

I2C1 error interrupt handler.

6.7.3.12 CH_IRQ_HANDLER (I2C2_EV_IRQHandler)

I2C2 event interrupt handler.

Function Class:

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

6.7.3.13 CH_IRQ_HANDLER (I2C2_ER_IRQHandler)

I2C2 error interrupt handler.

Function Class:

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

6.7.3.14 CH_IRQ_HANDLER (I2C3_EV_IRQHandler)

I2C3 event interrupt handler.

Function Class:

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

6.7.3.15 CH_IRQ_HANDLER (I2C3_ER_IRQHandler)

I2C3 error interrupt handler.

Function Class:

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

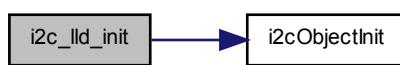
6.7.3.16 void i2c_lld_init (void)

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:



6.7.3.17 void i2c_lld_start (I2CDriver * i2cp)

Configures and activates the I2C peripheral.

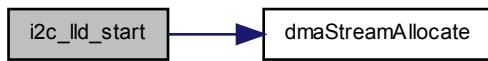
Parameters

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:

**6.7.3.18 void i2c_lld_stop (I2CDriver * i2cp)**

Deactivates the I2C peripheral.

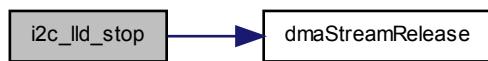
Parameters

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:

**6.7.3.19 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. After a timeout the driver must be stopped and restarted because the bus is in an uncertain state.

Function Class:

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

6.7.3.20 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. After a timeout the driver must be stopped and restarted because the bus is in an uncertain state.

Function Class:

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

6.7.4 Variable Documentation

6.7.4.1 I2CDriver I2CD1

I2C1 driver identifier.

6.7.4.2 I2CDriver I2CD2

I2C2 driver identifier.

6.7.4.3 I2CDriver I2CD3

I2C3 driver identifier.

6.7.5 Define Documentation

6.7.5.1 #define I2CD_NO_ERROR 0x00

No error.

6.7.5.2 #define I2CD_BUS_ERROR 0x01

Bus Error.

6.7.5.3 #define I2CD_ARBITRATION_LOST 0x02

Arbitration Lost (master mode).

6.7.5.4 #define I2CD_ACK_FAILURE 0x04

Acknowledge Failure.

6.7.5.5 #define I2CD_OVERRUN 0x08

Overrun/Underrun.

6.7.5.6 #define I2CD_PEC_ERROR 0x10

PEC Error in reception.

6.7.5.7 #define I2CD_TIMEOUT 0x20

Hardware timeout.

6.7.5.8 #define I2CD_SMB_ALERT 0x40

SMBus Alert.

6.7.5.9 #define I2C_USE_MUTUAL_EXCLUSION TRUE

Enables the mutual exclusion APIs on the I2C bus.

6.7.5.10 #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.

6.7.5.11 #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.

6.7.5.12 #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 I2CDriver object
in	<i>msg</i>	wakeup message

Function Class:

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

6.7.5.13 #define I2C_CLK_FREQ ((STM32_PCLK1) / 1000000)

Peripheral clock frequency.

6.7.5.14 #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.

6.7.5.15 #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.

6.7.5.16 #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.

6.7.5.17 #define STM32_I2C_I2C1_IRQ_PRIORITY 10

I2C1 interrupt priority level setting.

6.7.5.18 #define STM32_I2C_I2C2_IRQ_PRIORITY 10

I2C2 interrupt priority level setting.

6.7.5.19 #define STM32_I2C_I2C3_IRQ_PRIORITY 10

I2C3 interrupt priority level setting.

6.7.5.20 #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.

```
6.7.5.21 #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.

```
6.7.5.22 #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.

```
6.7.5.23 #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.

```
6.7.5.24 #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.

```
6.7.5.25 #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.

```
6.7.5.26 #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.

```
6.7.5.27 #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.

```
6.7.5.28 #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.

```
6.7.5.29 #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.

```
6.7.5.30 #define STM32_DMA_REQUIRED
```

error checks

```
6.7.5.31 #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.

6.7.6 Typedef Documentation

```
6.7.6.1 typedef uint16_t i2caddr_t
```

Type representing I2C address.

```
6.7.6.2 typedef uint32_t i2cflags_t
```

I2C Driver condition flags type.

```
6.7.6.3 typedef struct I2CDriver I2CDriver
```

Type of a structure representing an I2C driver.

6.7.7 Enumeration Type Documentation

6.7.7.1 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.

6.7.7.2 enum i2copmode_t

Supported modes for the I2C bus.

6.7.7.3 enum i2cdutycycle_t

Supported duty cycle modes for the I2C bus.

6.8 ICU Driver

6.8.1 Detailed Description

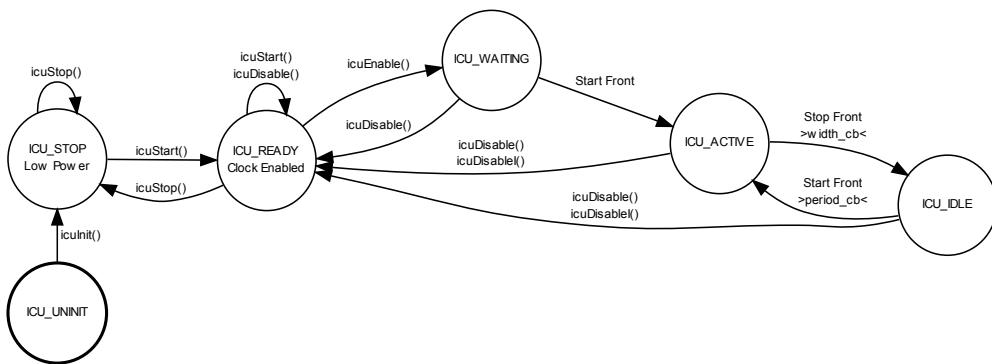
Generic ICU Driver. This module implements a generic ICU (Input Capture Unit) driver.

Precondition

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

6.8.2 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).



6.8.3 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.

6.8.4 Function Documentation

6.8.4.1 void icuInit(void)

ICU Driver initialization.

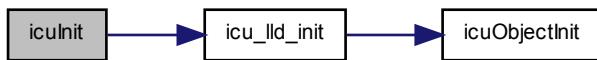
Note

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:



6.8.4.2 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.

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

Configures and activates the ICU peripheral.

Parameters

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

Function Class:

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

Here is the call graph for this function:



6.8.4.4 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:



6.8.4.5 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:



6.8.4.6 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:



6.8.4.7 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:



6.8.4.8 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.

6.8.4.9 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.

6.8.4.10 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.

6.8.4.11 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.

6.8.5 Variable Documentation

6.8.5.1 ICUDriver ICUD1

ICUD1 driver identifier.

Note

The driver ICUD1 allocates the complex timer TIM1 when enabled.

6.8.5.2 ICUDriver ICUD2

ICUD2 driver identifier.

Note

The driver ICUD1 allocates the timer TIM2 when enabled.

6.8.5.3 ICUDriver ICUD3

ICUD3 driver identifier.

Note

The driver ICUD1 allocates the timer TIM3 when enabled.

6.8.5.4 ICUDriver ICUD4

ICUD4 driver identifier.

Note

The driver ICUD4 allocates the timer TIM4 when enabled.

6.8.5.5 ICUDriver ICUD5

ICUD5 driver identifier.

Note

The driver ICUD5 allocates the timer TIM5 when enabled.

6.8.5.6 ICUDriver ICUD8

ICUD8 driver identifier.

Note

The driver ICUD8 allocates the timer TIM8 when enabled.

6.8.6 Define Documentation

6.8.6.1 #define icuEnable(*icup*) icu_ll_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.

6.8.6.2 #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.

6.8.6.3 #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.

6.8.6.4 #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.

6.8.6.5 #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.

6.8.6.6 #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.

6.8.6.7 #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.

6.8.6.8 #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.

6.8.6.9 #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.

6.8.6.10 #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.

6.8.6.11 #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.

6.8.6.12 #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.

6.8.6.13 #define STM32_ICU_TIM1_IRQ_PRIORITY 7

ICUD1 interrupt priority level setting.

6.8.6.14 #define STM32_ICU_TIM2_IRQ_PRIORITY 7

ICUD2 interrupt priority level setting.

6.8.6.15 #define STM32_ICU_TIM3_IRQ_PRIORITY 7

ICUD3 interrupt priority level setting.

6.8.6.16 #define STM32_ICU_TIM4_IRQ_PRIORITY 7

ICUD4 interrupt priority level setting.

6.8.6.17 #define STM32_ICU_TIM5_IRQ_PRIORITY 7

ICUD5 interrupt priority level setting.

6.8.6.18 #define STM32_ICU_TIM8_IRQ_PRIORITY 7

ICUD8 interrupt priority level setting.

```
6.8.6.19 #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.

```
6.8.6.20 #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.

6.8.7 Typedef Documentation

```
6.8.7.1 typedef struct ICUDriver ICUDriver
```

Type of a structure representing an ICU driver.

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

ICU notification callback type.

Parameters

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

```
6.8.7.3 typedef uint32_t icufreq_t
```

ICU frequency type.

```
6.8.7.4 typedef uint16_t icucnt_t
```

ICU counter type.

6.8.8 Enumeration Type Documentation

6.8.8.1 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.

6.8.8.2 enum icumode_t

ICU driver mode.

Enumerator:

ICU_INPUT_ACTIVE_HIGH Trigger on rising edge.

ICU_INPUT_ACTIVE_LOW Trigger on falling edge.

6.9 MMC over SPI Driver

6.9.1 Detailed Description

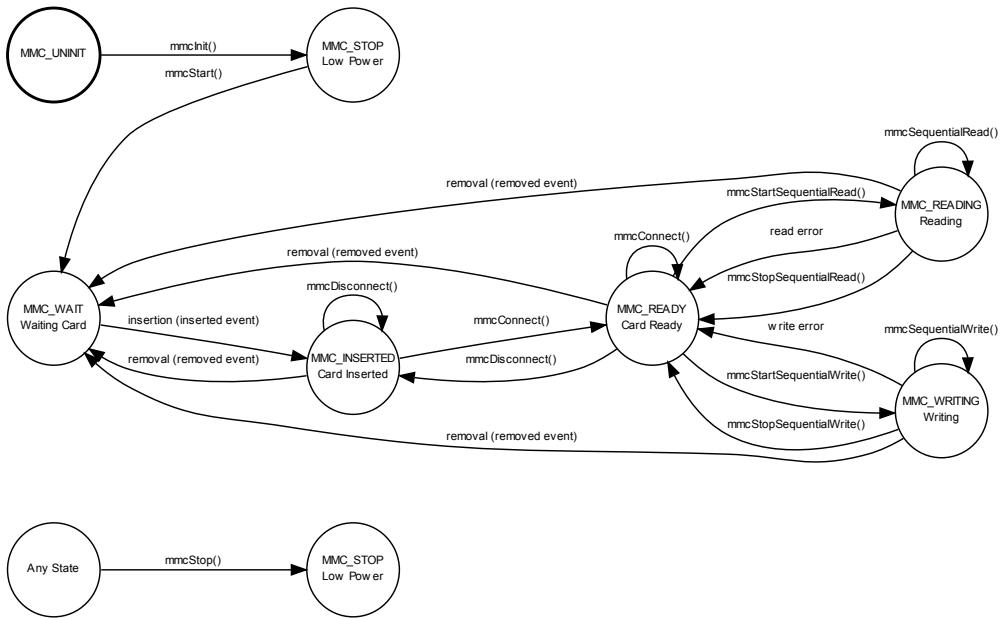
Generic MMC driver. This module implements a portable MMC/SD driver that uses a SPI driver as physical layer. Hot plugging and removal are supported through kernel events.

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`.

6.9.2 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).



Data Structures

- struct **MMCCfg**
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 **MMCCfg** *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)`
Stops a sequential read gracefully.
- `bool_t mmcSequentialWrite (MMCDriver *mmcp, const uint8_t *buffer)`
Starts a sequential write.
- `bool_t mmcStopSequentialWrite (MMCDriver *mmcp)`
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.

6.9.3 Function Documentation

6.9.3.1 void mmcInit (void)

MMC over SPI driver initialization.

Note

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

Function Class:

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

6.9.3.2 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 MMCDriver 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.

6.9.3.3 void mmcStart (**MMCDriver** * *mmcp*, const **MMCConfig** * *config*)

Configures and activates the MMC peripheral.

Parameters

in	<i>mmcp</i>	pointer to the MMCDriver object
in	<i>config</i>	pointer to the MMCConfig object. Must be NULL.

Function Class:

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

6.9.3.4 void mmcStop (**MMCDriver** * *mmcp*)

Disables the MMC peripheral.

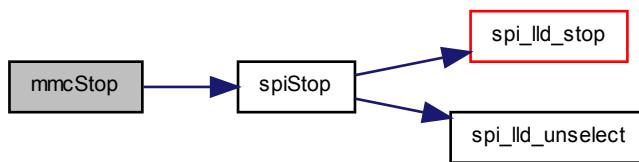
Parameters

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

Function Class:

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

Here is the call graph for this function:



6.9.3.5 `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

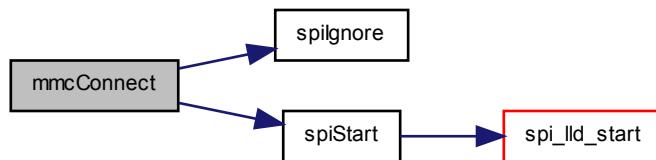
`FALSE` the operation succeeded and the driver is now in the `MMC_READY` state.

`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:



6.9.3.6 `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

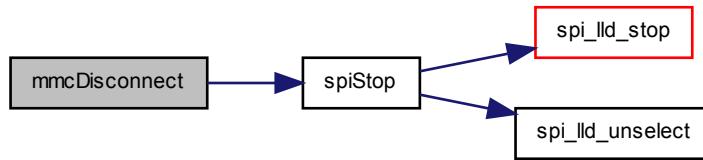
`FALSE` the operation succeeded and the driver is now in the `MMC_INSERTED` state.

`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:

**6.9.3.7 bool_t mmcStartSequentialRead (MMCDriver * mmcp, uint32_t startblk)**

Starts a sequential read.

Parameters

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

Returns

The operation status.

Return values

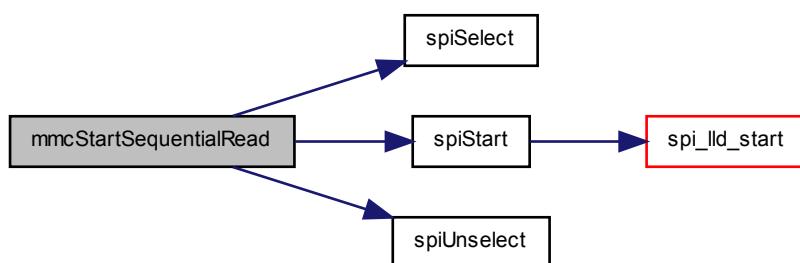
FALSE the operation succeeded.

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:



6.9.3.8 `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

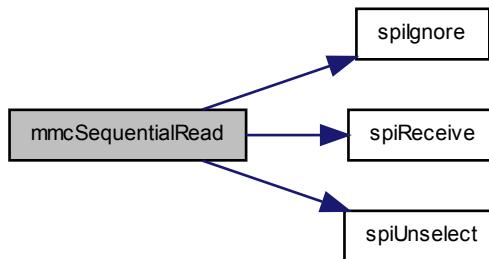
FALSE the operation succeeded.

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:



6.9.3.9 `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

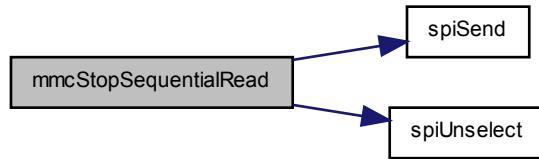
FALSE the operation succeeded.

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:



6.9.3.10 `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.

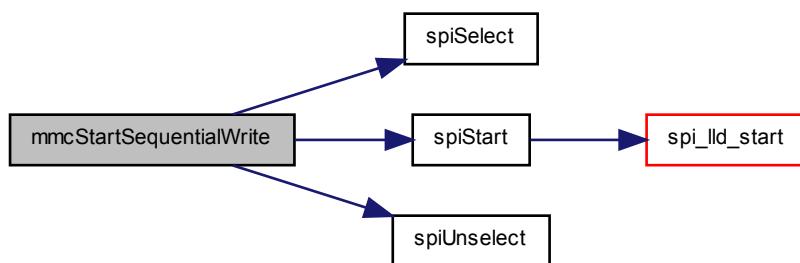
Return values

<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:



6.9.3.11 `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

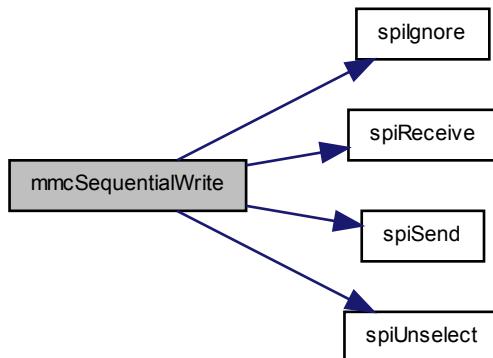
FALSE the operation succeeded.

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:



6.9.3.12 `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

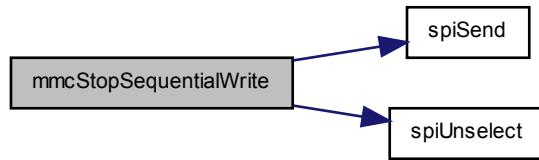
FALSE the operation succeeded.

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:



6.9.4 Define Documentation

6.9.4.1 #define MMC_SECTOR_SIZE 512

Block size for MMC transfers.

6.9.4.2 #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.

6.9.4.3 #define MMC_POLLING_INTERVAL 10

Number of positive insertion queries before generating the insertion event.

6.9.4.4 #define MMC_POLLING_DELAY 10

Interval, in milliseconds, between insertion queries.

6.9.4.5 #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.

6.9.4.6 #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.

6.9.5 Typedef Documentation

6.9.5.1 `typedef bool_t(* mmcquery_t)(void)`

Function used to query some hardware status bits.

Returns

The status.

6.9.6 Enumeration Type Documentation

6.9.6.1 `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.

6.10 PAL Driver

6.10.1 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`.

6.10.2 Implementation Rules

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

6.10.2.1 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.

6.10.2.2 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.

6.10.2.3 Writing unused or unimplemented port bits

The behavior is not specified.

6.10.2.4 Reading from unused or unimplemented port bits

The behavior is not specified.

6.10.2.5 Reading or writing on pins associated to other functionalities

The behavior is not specified.

Data Structures

- struct `IOBus`
I/O bus descriptor.
- struct `GPIO_TypeDef`
STM32 GPIO registers block.
- 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_STM32_MODE_MASK (3 << 0)`
- `#define PAL_STM32_MODE_INPUT (0 << 0)`
- `#define PAL_STM32_MODE_OUTPUT (1 << 0)`
- `#define PAL_STM32_MODE_ALTERNATE (2 << 0)`
- `#define PAL_STM32_MODE_ANALOG (3 << 0)`
- `#define PAL_STM32_OTYPE_MASK (1 << 2)`
- `#define PAL_STM32_OTYPE_PUSH_PULL (0 << 2)`
- `#define PAL_STM32_OTYPE_OPENDRAIN (1 << 2)`
- `#define PAL_STM32_OSPEED_MASK (3 << 3)`
- `#define PAL_STM32_OSPEED_LOWEST (0 << 3)`
- `#define PAL_STM32_OSPEED_MID1 (1 << 3)`
- `#define PAL_STM32_OSPEED_MID2 (2 << 3)`
- `#define PAL_STM32_OSPEED_HIGHEST (3 << 3)`
- `#define PAL_STM32_PUDR_MASK (3 << 5)`
- `#define PAL_STM32_PUDR_FLOATING (0 << 5)`

- #define **PAL_STM32_PUDR_PULLUP** (1 << 5)
- #define **PAL_STM32_PUDR_PULLDOWN** (2 << 5)
- #define **PAL_STM32_ALTERNATE_MASK** (15 << 7)
- #define **PAL_STM32_ALTERNATE**(n) ((n) << 7)
- #define **PAL_MODE_ALTERNATE**(n)

Alternate function.

Standard I/O mode flags

- #define **PAL_MODE_RESET** PAL_STM32_MODE_INPUT
This mode is implemented as input.
- #define **PAL_MODE_UNCONNECTED** PAL_STM32_MODE_OUTPUT
This mode is implemented as output.
- #define **PAL_MODE_INPUT** PAL_STM32_MODE_INPUT
Regular input high-Z pad.
- #define **PAL_MODE_INPUT_PULLUP**
Input pad with weak pull up resistor.
- #define **PAL_MODE_INPUT_PULLDOWN**
Input pad with weak pull down resistor.
- #define **PAL_MODE_INPUT_ANALOG** PAL_STM32_MODE_ANALOG
Analog input mode.
- #define **PAL_MODE_OUTPUT_PUSH_PULL**
Push-pull output pad.
- #define **PAL_MODE_OUTPUT_OPENDRAIN**
Open-drain output pad.

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

- `#define IOPORT7 GPIOG`
GPIO port G identifier.
- `#define IOPORT8 GPIOH`
GPIO port H identifier.
- `#define IOPORT9 GPIOI`
GPIO port I 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_readlatch(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.H.set = (uint16_t)(bits))`
Sets a bits mask on a I/O port.
- `#define pal_lld_clearport(port, bits) ((port)->BSRR.H.clear = (uint16_t)(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.

6.10.3 Function Documentation

6.10.3.1 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 `bus` the I/O bus, pointer to a `IOBus` 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.

6.10.3.2 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.

6.10.3.3 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.

6.10.3.4 void _pal_lld_init (const PALConfig * config)

STM32 I/O ports configuration.

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

Parameters

in	<code>config</code>	the STM32 ports configuration
----	---------------------	-------------------------------

Function Class:

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

6.10.3.5 void _pal_lid_setgroupmode (ioportid_t port, ioportmask_t mask, iomode_t mode)

Pads mode setup.

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

Note

PAL_MODE_UNCONNECTED is implemented as push pull at minimum speed.

Parameters

in	<i>port</i>	the port identifier
in	<i>mask</i>	the group mask
in	<i>mode</i>	the mode

Function Class:

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

6.10.4 Define Documentation**6.10.4.1 #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.

6.10.4.2 #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_PUSH_PULL as example.

6.10.4.3 #define PAL_MODE_INPUT 2

Regular input high-Z pad.

6.10.4.4 #define PAL_MODE_INPUT_PULLUP 3

Input pad with weak pull up resistor.

6.10.4.5 #define PAL_MODE_INPUT_PULLDOWN 4

Input pad with weak pull down resistor.

6.10.4.6 #define PAL_MODE_INPUT_ANALOG 5

Analog input mode.

6.10.4.7 #define PAL_MODE_OUTPUT_PUSH_PULL 6

Push-pull output pad.

6.10.4.8 #define PAL_MODE_OUTPUT_OPENDRAIN 7

Open-drain output pad.

6.10.4.9 #define PAL_LOW 0

Logical low state.

6.10.4.10 #define PAL_HIGH 1

Logical high state.

6.10.4.11 #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.

6.10.4.12 #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.

6.10.4.13 #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 IOBus variable
in	<i>port</i>	I/O port descriptor
in	<i>width</i>	bus width in bits
in	<i>offset</i>	bus bit offset within the port

```
6.10.4.14 #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 IOBus variable
in	<i>port</i>	I/O port descriptor
in	<i>width</i>	bus width in bits
in	<i>offset</i>	bus bit offset within the port

```
6.10.4.15 #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.

```
6.10.4.16 #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.

```
6.10.4.17 #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.

6.10.4.18 #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.

6.10.4.19 #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.

6.10.4.20 #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.

```
6.10.4.21 #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.

```
6.10.4.22 #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.

```
6.10.4.23 #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.

6.10.4.24 #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.

6.10.4.25 #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.

6.10.4.26 #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 PAL_LOW or PAL_HIGH

Function Class:

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

6.10.4.27 #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.

6.10.4.28 #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.

6.10.4.29 #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.

6.10.4.30 #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.

6.10.4.31 #define PAL_MODE_ALTERNATE(*n*)

Value:

```
(PAL_STM32_MODE_ALTERNATE |      \
           \      \
           PAL_STM32_ALTERNATE(n))
```

Alternate function.

Parameters

in *n* alternate function selector

6.10.4.32 #define PAL_MODE_RESET PAL_STM32_MODE_INPUT

This mode is implemented as input.

6.10.4.33 #define PAL_MODE_UNCONNECTED PAL_STM32_MODE_OUTPUT

This mode is implemented as output.

6.10.4.34 #define PAL_MODE_INPUT PAL_STM32_MODE_INPUT

Regular input high-Z pad.

6.10.4.35 #define PAL_MODE_INPUT_PULLUP

Value:

(PAL_STM32_MODE_INPUT | \
PAL_STM32_PUDR_PULLUP)

Input pad with weak pull up resistor.

6.10.4.36 #define PAL_MODE_INPUT_PULLDOWN

Value:

(PAL_STM32_MODE_INPUT | \
PAL_STM32_PUDR_PULLDOWN)

Input pad with weak pull down resistor.

6.10.4.37 #define PAL_MODE_INPUT_ANALOG PAL_STM32_MODE_ANALOG

Analog input mode.

6.10.4.38 #define PAL_MODE_OUTPUT_PUSH_PULL

Value:

(PAL_STM32_MODE_OUTPUT | \
PAL_STM32_OTYPE_PUSH_PULL)

Push-pull output pad.

6.10.4.39 #define PAL_MODE_OUTPUT_OPENDRAIN

Value:

(PAL_STM32_MODE_OUTPUT | \
PAL_STM32_OTYPE_OPENDRAIN)

Open-drain output pad.

6.10.4.40 `#define PAL_IOPORTS_WIDTH 16`

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

6.10.4.41 `#define PAL_WHOLE_PORT ((ioportmask_t)0xFFFF)`

Whole port mask.

This macro specifies all the valid bits into a port.

6.10.4.42 `#define IOPORT1 GPIOA`

GPIO port A identifier.

6.10.4.43 `#define IOPORT2 GPIOB`

GPIO port B identifier.

6.10.4.44 `#define IOPORT3 GPIOC`

GPIO port C identifier.

6.10.4.45 `#define IOPORT4 GPIOD`

GPIO port D identifier.

6.10.4.46 `#define IOPORT5 GPIOE`

GPIO port E identifier.

6.10.4.47 `#define IOPORT6 GPIOF`

GPIO port F identifier.

6.10.4.48 `#define IOPORT7 GPIOG`

GPIO port G identifier.

6.10.4.49 `#define IOPORT8 GPIOH`

GPIO port H identifier.

6.10.4.50 `#define IOPORT9 GPIOI`

GPIO port I identifier.

```
6.10.4.51 #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.

```
6.10.4.52 #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.

```
6.10.4.53 #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.

```
6.10.4.54 #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.

Parameters

in *port* port identifier

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

Function Class:

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

```
6.10.4.55 #define pal_lld_setport( port, bits ) ((port)->BSRR.H.set = (uint16_t)(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.

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.

```
6.10.4.56 #define pal_lld_clearport( port, bits ) ((port)->BSRR.H.clear = (uint16_t)(bits))
```

Clears a bits mask on a I/O port.

This function is implemented by writing the GPIO BSRR register, the implementation has no side effects.

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.

```
6.10.4.57 #define pal_lld_writegroup( port, mask, offset, bits )
```

Value:

```
((port)->BSRR.W = (((~(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.

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.

```
6.10.4.58 #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.

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.

6.10.4.59 #define pal_lld_writepad(*port*, *pad*, *bit*) pal_lld_writegroup(*port*, 1, *pad*, *bit*)

Writes a logical state on an output pad.

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.

6.10.5 Typedef Documentation

6.10.5.1 `typedef uint32_t ioportmask_t`

Digital I/O port sized unsigned type.

6.10.5.2 `typedef uint32_t iomode_t`

Digital I/O modes.

6.10.5.3 `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.

6.11 PWM Driver

6.11.1 Detailed Description

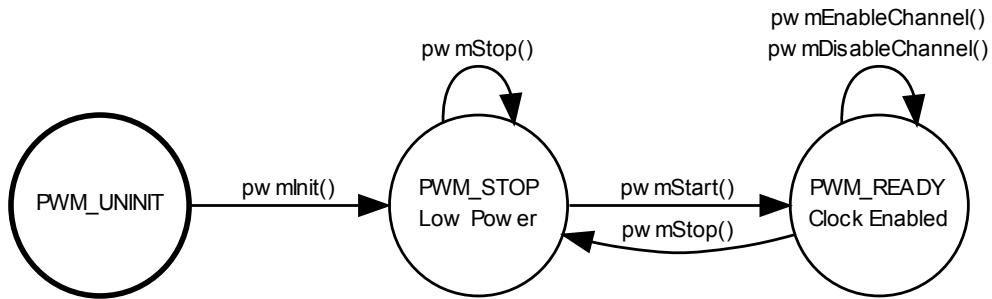
Generic PWM Driver. This module implements a generic PWM (Pulse Width Modulation) driver.

Precondition

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

6.11.2 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).



6.11.3 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 pwmChangePeriod(l)(pwmp, period)`
Changes the period the PWM peripheral.
- `#define pwmEnableChannel(l)(pwmp, channel, width) pwm_ll_enable_channel(pwmp, channel, width)`
Enables a PWM channel.
- `#define pwmDisableChannel(l)(pwmp, channel) pwm_ll_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 pwemode_t`
PWM mode type.
- `typedef uint8_t pwmchannel_t`
PWM channel type.
- `typedef uint16_t pwmcnt_t`
PWM counter type.

Enumerations

- `enum pwmstate_t { PWM_UNINIT = 0, PWM_STOP = 1, PWM_READY = 2 }`
Driver state machine possible states.

6.11.4 Function Documentation

6.11.4.1 void pwminit(void)

PWM Driver initialization.

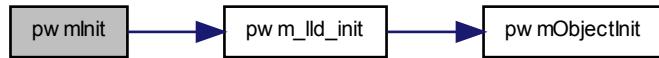
Note

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

Function Class:

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

Here is the call graph for this function:



6.11.4.2 void pwmObjectInit (PWMDriver * *pwmp*)

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

Parameters

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

Function Class:

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

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

Function Class:

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

Here is the call graph for this function:



6.11.4.4 void pwmStop (PWMDriver * *pwmp*)

Deactivates the PWM peripheral.

Parameters

in *pwmp* pointer to a `PWMDriver` object

Function Class:

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

Here is the call graph for this function:

**6.11.4.5 void pwmChangePeriod (PWMDriver * *pwmp*, pwcnt_t *period*)**

Changes the period the PWM peripheral.

This function changes the period of a PWM unit that has already been activated using `pwmStart ()`.

Precondition

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

Postcondition

The PWM unit period is changed to the new value.

Note

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

Parameters

in	<i>pwmp</i>	pointer to a <code>PWMDriver</code> object
in	<i>period</i>	new cycle time in ticks

Function Class:

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

6.11.4.6 void pwmEnableChannel (PWMDriver * *pwmp*, pwmchannel_t *channel*, pwcnt_t *width*)

Enables a PWM channel.

Precondition

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

Postcondition

The channel is active using the specified configuration.

Note

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

Parameters

in	<i>pwmp</i>	pointer to a PWMDriver object
in	<i>channel</i>	PWM channel identifier (0...PWM_CHANNELS-1)
in	<i>width</i>	PWM pulse width as clock pulses number

Function Class:

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

Here is the call graph for this function:

**6.11.4.7 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 PWMDriver 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:



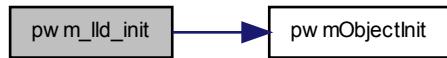
6.11.4.8 void pwm_lld_init (void)

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:



6.11.4.9 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.

6.11.4.10 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.

6.11.4.11 void pwm_lld_enable_channel (PWMDriver * *pwmp*, pwmchannel_t *channel*, pwcmt_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 PWMDriver 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:

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

6.11.4.12 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 PWMDriver object
in	<i>channel</i>	PWM channel identifier (0...PWM_CHANNELS-1)

Function Class:

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

6.11.5 Variable Documentation

6.11.5.1 PWMDriver PWMD1

PWMD1 driver identifier.

Note

The driver PWMD1 allocates the complex timer TIM1 when enabled.

6.11.5.2 PWMDriver PWMD2

PWMD2 driver identifier.

Note

The driver PWMD2 allocates the timer TIM2 when enabled.

6.11.5.3 PWMDriver PWMD3

PWMD3 driver identifier.

Note

The driver PWMD3 allocates the timer TIM3 when enabled.

6.11.5.4 PWMDriver PWMD4

PWMD4 driver identifier.

Note

The driver PWMD4 allocates the timer TIM4 when enabled.

6.11.5.5 PWMDriver PWMD5

PWMD5 driver identifier.

Note

The driver PWMD5 allocates the timer TIM5 when enabled.

6.11.5.6 PWMDriver PWMD8

PWMD8 driver identifier.

Note

The driver PWMD5 allocates the timer TIM5 when enabled.

6.11.6 Define Documentation**6.11.6.1 #define PWM_OUTPUT_MASK 0x0F**

Standard output modes mask.

6.11.6.2 #define PWM_OUTPUT_DISABLED 0x00

Output not driven, callback only.

6.11.6.3 #define PWM_OUTPUT_ACTIVE_HIGH 0x01

Positive PWM logic, active is logic level one.

6.11.6.4 #define PWM_OUTPUT_ACTIVE_LOW 0x02

Inverse PWM logic, active is logic level zero.

6.11.6.5 #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 PWMDriver object
in	<i>denominator</i>	denominator of the fraction
in	<i>numerator</i>	numerator of the fraction

Returns

The pulse width to be passed to [pwmEnableChannel\(\)](#).

Function Class:

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

6.11.6.6 #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 PWMDriver object
in	<i>degrees</i>	degrees as an integer between 0 and 36000

Returns

The pulse width to be passed to [pwmEnableChannel\(\)](#).

Function Class:

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

```
6.11.6.7 #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.

```
6.11.6.8 #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 <code>PWMDriver</code> 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.

```
6.11.6.9 #define pwmEnableChannel( 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 PWMDriver 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.

```
6.11.6.10 #define pwmDisableChannel( 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 PWMDriver object
in	<i>channel</i>	PWM channel identifier (0...PWM_CHANNELS-1)

Function Class:

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

```
6.11.6.11 #define PWM_CHANNELS 4
```

Number of PWM channels per PWM driver.

6.11.6.12 #define PWM_COMPLEMENTARY_OUTPUT_MASK 0xF0

Complementary output modes mask.

Note

This is an STM32-specific setting.

6.11.6.13 #define PWM_COMPLEMENTARY_OUTPUT_DISABLED 0x00

Complementary output not driven.

Note

This is an STM32-specific setting.

6.11.6.14 #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.

6.11.6.15 #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.

6.11.6.16 #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.

6.11.6.17 #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.

6.11.6.18 #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.

6.11.6.19 #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.

6.11.6.20 #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.

6.11.6.21 #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.

6.11.6.22 #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.

6.11.6.23 #define STM32_PWM_TIM1_IRQ_PRIORITY 7

PWMD1 interrupt priority level setting.

6.11.6.24 #define STM32_PWM_TIM2_IRQ_PRIORITY 7

PWMD2 interrupt priority level setting.

```
6.11.6.25 #define STM32_PWM_TIM3_IRQ_PRIORITY 7
```

PWMD3 interrupt priority level setting.

```
6.11.6.26 #define STM32_PWM_TIM4_IRQ_PRIORITY 7
```

PWMD4 interrupt priority level setting.

```
6.11.6.27 #define STM32_PWM_TIM5_IRQ_PRIORITY 7
```

PWMD5 interrupt priority level setting.

```
6.11.6.28 #define STM32_PWM_TIM8_IRQ_PRIORITY 7
```

PWMD8 interrupt priority level setting.

```
6.11.6.29 #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 PWMDriver object
in	<i>period</i>	new cycle time in ticks

Function Class:

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

6.11.7 Typedef Documentation

```
6.11.7.1 typedef struct PWMDriver PWMDriver
```

Type of a structure representing a PWM driver.

```
6.11.7.2 typedef void(* pwmcallback_t)(PWMDriver *pwmp)
```

PWM notification callback type.

Parameters

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

6.11.7.3 `typedef uint32_t pwmmode_t`

PWM mode type.

6.11.7.4 `typedef uint8_t pwmchannel_t`

PWM channel type.

6.11.7.5 `typedef uint16_t pwcnt_t`

PWM counter type.

6.11.8 Enumeration Type Documentation

6.11.8.1 `enum pwmstate_t`

Driver state machine possible states.

Enumerator:

PWM_UNINIT Not initialized.

PWM_STOP Stopped.

PWM_READY Ready.

6.12 Serial Driver

6.12.1 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.

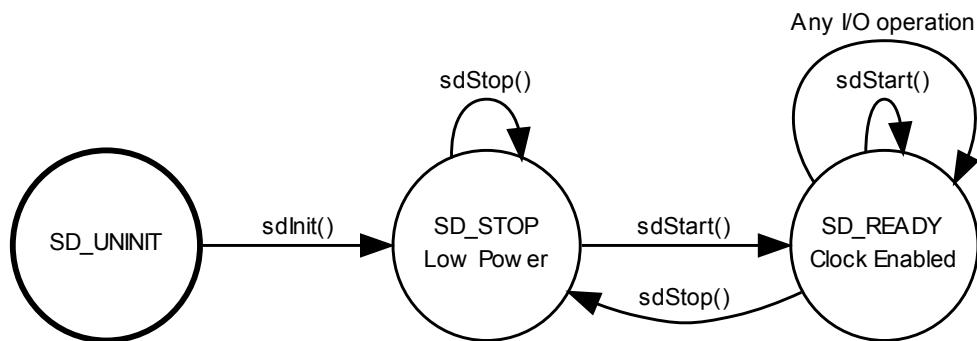
The module also contains functions that make the implementation of the interrupt service routines much easier.

Precondition

In order to use the SERIAL driver the `HAL_USE_SERIAL` option must be enabled in `halconf.h`.

6.12.2 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).



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)
USART5 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**
USART4 serial driver identifier.
- **SerialDriver SD5**
USART5 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.`

6.12.3 Function Documentation

6.12.3.1 void sdInit(void)

Serial Driver initialization.

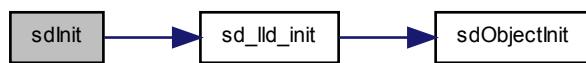
Note

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:



6.12.3.2 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.

6.12.3.3 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:



6.12.3.4 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:



6.12.3.5 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.

6.12.3.6 *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.

6.12.3.7 CH_IRQ_HANDLER (USART1_IRQHandler)

USART1 interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.12.3.8 CH_IRQ_HANDLER (USART2_IRQHandler)

USART2 interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.12.3.9 CH_IRQ_HANDLER (USART3_IRQHandler)

USART3 interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.12.3.10 CH_IRQ_HANDLER (UART4_IRQHandler)

UART4 interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.12.3.11 CH_IRQ_HANDLER (UART5_IRQHandler)

UART5 interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.12.3.12 CH_IRQ_HANDLER (USART6_IRQHandler)

USART1 interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.12.3.13 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:

**6.12.3.14 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 SerialDriver object
in	<i>config</i>	the architecture-dependent serial driver configuration. If this parameter is set to <code>NULL</code> then a default configuration is used.

Function Class:

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

6.12.3.15 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 SerialDriver object
----	------------	--

Function Class:

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

6.12.4 Variable Documentation**6.12.4.1 SerialDriver SD1**

USART1 serial driver identifier.

6.12.4.2 SerialDriver SD2

USART2 serial driver identifier.

6.12.4.3 SerialDriver SD3

USART3 serial driver identifier.

6.12.4.4 SerialDriver SD4

UART4 serial driver identifier.

6.12.4.5 SerialDriver SD5

UART5 serial driver identifier.

6.12.4.6 SerialDriver SD6

USART6 serial driver identifier.

6.12.5 Define Documentation**6.12.5.1 #define SD_PARITY_ERROR 32**

Parity error happened.

6.12.5.2 #define SD_FRAMING_ERROR 64

Framing error happened.

6.12.5.3 #define SD_OVERRUN_ERROR 128

Overflow happened.

6.12.5.4 #define SD_NOISE_ERROR 256

Noise on the line.

6.12.5.5 #define SD_BREAK_DETECTED 512

Break detected.

6.12.5.6 #define SERIAL_DEFAULT_BITRATE 38400

Default bit rate.

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

6.12.5.7 #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.

6.12.5.8 #define _serial_driver_methods _base_asynchronous_channel_methods

[SerialDriver](#) specific methods.

6.12.5.9 #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.

6.12.5.10 #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.

```
6.12.5.11 #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.

```
6.12.5.12 #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.

```
6.12.5.13 #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.

```
6.12.5.14 #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.

6.12.5.15 #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.

6.12.5.16 #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.

6.12.5.17 #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.

```
6.12.5.18 #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.

```
6.12.5.19 #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.

```
6.12.5.20 #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.

```
6.12.5.21 #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.

6.12.5.22 #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.

6.12.5.23 #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.

6.12.5.24 #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.

6.12.5.25 #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.

6.12.5.26 #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.

6.12.5.27 #define STM32_SERIAL_USART1_PRIORITY 12

USART1 interrupt priority level setting.

6.12.5.28 #define STM32_SERIAL_USART2_PRIORITY 12

USART2 interrupt priority level setting.

6.12.5.29 #define STM32_SERIAL_USART3_PRIORITY 12

USART3 interrupt priority level setting.

6.12.5.30 #define STM32_SERIAL_UART4_PRIORITY 12

UART4 interrupt priority level setting.

6.12.5.31 #define STM32_SERIAL_UART5_PRIORITY 12

UART5 interrupt priority level setting.

6.12.5.32 #define STM32_SERIAL_USART6_PRIORITY 12

USART6 interrupt priority level setting.

6.12.5.33 #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.

6.12.5.34 #define USART_CR2_STOP1_BITS (0 << 12)

CR2 1 stop bit value.

6.12.5.35 #define USART_CR2_STOP0P5_BITS (1 << 12)

CR2 0.5 stop bit value.

6.12.5.36 #define USART_CR2_STOP2_BITS (2 << 12)

CR2 2 stop bit value.

6.12.5.37 #define USART_CR2_STOP1P5_BITS (3 << 12)

CR2 1.5 stop bit value.

6.12.6 Typedef Documentation

6.12.6.1 `typedef struct SerialDriver SerialDriver`

Structure representing a serial driver.

6.12.7 Enumeration Type Documentation

6.12.7.1 `enum sdstate_t`

Driver state machine possible states.

Enumerator:

`SD_UNINIT` Not initialized.

`SD_STOP` Stopped.

`SD_READY` Ready.

6.13 SPI Driver

6.13.1 Detailed Description

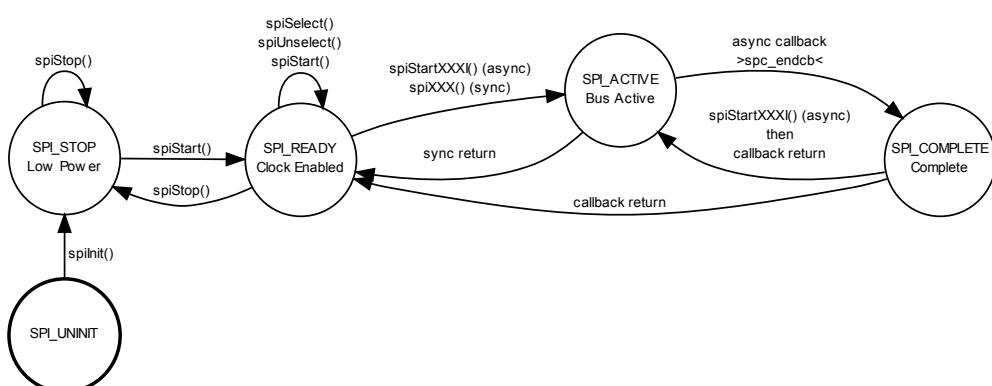
Generic SPI Driver. This module implements a generic SPI (Serial Peripheral Interface) driver allowing bidirectional and monodirectional transfers, complex atomic transactions are supported as well.

Precondition

In order to use the SPI driver the `HAL_USE_SPI` option must be enabled in `halconf.h`.

6.13.2 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 `SPIConfig`
Driver configuration structure.
- struct `SPIDriver`
Structure representing a SPI driver.

Functions

- void `spilInit (void)`
SPI Driver initialization.
- void `spiObjectInit (SPIDriver *spip)`
Initializes the standard part of a `SPIDriver` structure.
- void `spiStart (SPIDriver *spip, const SPIConfig *config)`
Configures and activates the SPI peripheral.
- void `spiStop (SPIDriver *spip)`
Deactivates the SPI peripheral.
- void `spiSelect (SPIDriver *spip)`
Asserts the slave select signal and prepares for transfers.
- void `spiUnselect (SPIDriver *spip)`
Deasserts the slave select signal.
- void `spiStartIgnore (SPIDriver *spip, size_t n)`
Ignores data on the SPI bus.
- void `spiStartExchange (SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)`
Exchanges data on the SPI bus.
- void `spiStartSend (SPIDriver *spip, size_t n, const void *txbuf)`
Sends data over the SPI bus.
- void `spiStartReceive (SPIDriver *spip, size_t n, void *rxbuf)`
Receives data from the SPI bus.
- void `spilgnore (SPIDriver *spip, size_t n)`
Ignores data on the SPI bus.
- void `spiExchange (SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)`
Exchanges data on the SPI bus.
- void `spiSend (SPIDriver *spip, size_t n, const void *txbuf)`
Sends data over the SPI bus.
- void `spiReceive (SPIDriver *spip, size_t n, void *rxbuf)`
Receives data from the SPI bus.
- void `spiAcquireBus (SPIDriver *spip)`
Gains exclusive access to the SPI bus.
- void `spiReleaseBus (SPIDriver *spip)`
Releases exclusive access to the SPI bus.
- 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)`

- **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 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(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)`
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.

Enumerations

```
• enum spistate_t {
    SPI_UNINIT = 0, SPI_STOP = 1, SPI_READY = 2, SPI_ACTIVE = 3,
    SPI_COMPLETE = 4 }
```

Driver state machine possible states.

6.13.3 Function Documentation

6.13.3.1 void spiInit(void)

SPI Driver initialization.

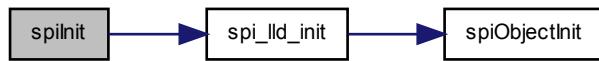
Note

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

Function Class:

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

Here is the call graph for this function:



6.13.3.2 void spiObjectInit(SPIDriver * spip)

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

Parameters

out	<i>spip</i> pointer to the SPIDriver object
-----	---

Function Class:

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

6.13.3.3 void spiStart(SPIDriver * spip, const SPIConfig * config)

Configures and activates the SPI peripheral.

Parameters

in	<i>spip</i> pointer to the SPIDriver object
in	<i>config</i> pointer to the SPIConfig object

Function Class:

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

Here is the call graph for this function:

**6.13.3.4 void spiStop ([SPIDriver](#) * *spip*)**

Deactivates the SPI peripheral.

Note

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

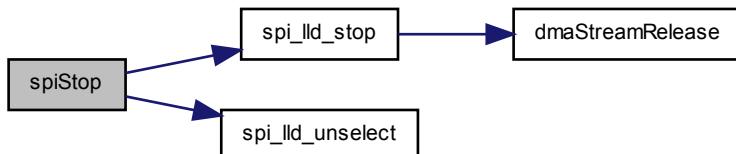
Parameters

in *spip* pointer to the [SPIDriver](#) object

Function Class:

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

Here is the call graph for this function:

**6.13.3.5 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.

6.13.3.6 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.

6.13.3.7 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.

6.13.3.8 void spiStartExchange (*SPIDriver* * *spip*, *size_t* *n*, const void * *txbuf*, void * *rxbuf*)

Exchanges data on the SPI bus.

This asynchronous function starts a simultaneous transmit/receive operation.

Precondition

A slave must have been selected using *spiSelect()* or *spiSelectI()*.

Postcondition

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

Note

The buffers are organized as *uint8_t* arrays for data sizes below or equal to 8 bits else it is organized as *uint16_t* arrays.

Parameters

in	<i>spip</i>	pointer to the <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.

6.13.3.9 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.

6.13.3.10 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 <code>SPIDriver</code> object
in	<i>n</i>	number of words to receive
out	<i>rxbuf</i>	the pointer to the receive buffer

Function Class:

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

6.13.3.11 void spiIgnore (*SPI_driver* * *spip*, *size_t* *n*)

Ignores data on the SPI bus.

This synchronous function performs the transmission of a series of idle words on the SPI bus and ignores the received data.

Precondition

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

In order to use this function the driver must have been configured without callbacks (`end_cb = NULL`).

Parameters

in	<i>spip</i>	pointer to the <code>SPI_driver</code> object
in	<i>n</i>	number of words to be ignored

Function Class:

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

6.13.3.12 void spiExchange (*SPI_driver* * *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>SPI_driver</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.

6.13.3.13 void spiSend (*SPI_driver* * *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	<code>spip</code>	pointer to the <code>SPIDriver</code> object
in	<code>n</code>	number of words to send
in	<code>txbuf</code>	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.

6.13.3.14 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	<code>spip</code>	pointer to the <code>SPIDriver</code> object
in	<code>n</code>	number of words to receive
out	<code>rdbuf</code>	the pointer to the receive buffer

Function Class:

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

6.13.3.15 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	<code>spip</code>	pointer to the <code>SPIDriver</code> object
----	-------------------	--

Function Class:

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

6.13.3.16 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.

6.13.3.17 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:

**6.13.3.18 void spi_lld_start (**SPIDriver** * *spip*)**

Configures and activates the SPI peripheral.

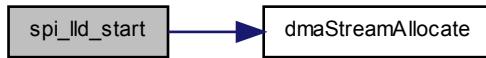
Parameters

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:



6.13.3.19 void spi_lld_stop (**SPIDriver** * *spip*)

Deactivates the SPI peripheral.

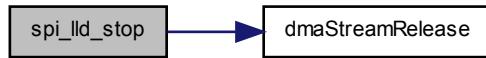
Parameters

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:



6.13.3.20 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.

6.13.3.21 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.

6.13.3.22 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.

6.13.3.23 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.

6.13.3.24 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 <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:

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

6.13.3.25 void spi_lld_receive (`SPIDriver * spip, size_t n, void * rdbuf`)

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 <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:

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

6.13.3.26 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.

6.13.4 Variable Documentation

6.13.4.1 **SPIDriver SPID1**

SPI1 driver identifier.

6.13.4.2 **SPIDriver SPID2**

SPI2 driver identifier.

6.13.4.3 **SPIDriver SPID3**

SPI3 driver identifier.

6.13.5 Define Documentation

6.13.5.1 `#define SPI_USE_WAIT TRUE`

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

6.13.5.2 `#define SPI_USE_MUTUAL_EXCLUSION TRUE`

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

Note

Disabling this option saves both code and data space.

6.13.5.3 `#define spiSelect(spip)`

Value:

```
{\n    spi_lld_select(spip);\n}
```

Asserts the slave select signal and prepares for transfers.

Parameters

in `spip` pointer to the `SPIDriver` object

Function Class:

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

6.13.5.4 `#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 SPIDriver object
----	---

Function Class:

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

6.13.5.5 #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 SPIDriver object
in	<i>n</i> number of words to be ignored

Function Class:

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

6.13.5.6 #define spiStartExchange(*spip*, *n*, *txbuf*, *rxbuf*)

Value:

```
{
    (spip)->state = SPI_ACTIVE;
    spi_lld_exchange(spip, n, txbuf, rxbuf);
}
```

Exchanges data on the SPI bus.

This asynchronous function starts a simultaneous transmit/receive operation.

Precondition

A slave must have been selected using `spiSelect()` or `spiSelectI()`.

Postcondition

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

Note

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

Parameters

in	<code>spip</code>	pointer to the <code>SPIDriver</code> object
in	<code>n</code>	number of words to be exchanged
in	<code>txbuf</code>	the pointer to the transmit buffer
out	<code>rxbuf</code>	the pointer to the receive buffer

Function Class:

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

6.13.5.7 #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	<code>spip</code>	pointer to the <code>SPIDriver</code> object
in	<code>n</code>	number of words to send
in	<code>txbuf</code>	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.

6.13.5.8 #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.

6.13.5.9 #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.

6.13.5.10 #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 [SPI_driver](#) object

Function Class:

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

6.13.5.11 #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 [SPI_driver](#) object

Function Class:

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

6.13.5.12 #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(spi); \}
```

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.

6.13.5.13 #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.

6.13.5.14 #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.

6.13.5.15 #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.

6.13.5.16 #define STM32_SPI_SPI1_IRQ_PRIORITY 10

SPI1 interrupt priority level setting.

```
6.13.5.17 #define STM32_SPI_SPI2_IRQ_PRIORITY 10
```

SPI2 interrupt priority level setting.

```
6.13.5.18 #define STM32_SPI_SPI3_IRQ_PRIORITY 10
```

SPI3 interrupt priority level setting.

```
6.13.5.19 #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.

```
6.13.5.20 #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.

```
6.13.5.21 #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.

```
6.13.5.22 #define STM32_SPI_DMA_ERROR_HOOK( spip ) chSysHalt()
```

SPI DMA error hook.

```
6.13.5.23 #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.

```
6.13.5.24 #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.

```
6.13.5.25 #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.

```
6.13.5.26 #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.

```
6.13.5.27 #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.

```
6.13.5.28 #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.

6.13.6 Typedef Documentation

```
6.13.6.1 typedef struct SPIDriver SPIDriver
```

Type of a structure representing an SPI driver.

```
6.13.6.2 typedef void(* spicallback_t)(SPIDriver *spip)
```

SPI notification callback type.

Parameters

in `spip` pointer to the `SPIDriver` object triggering the callback

6.13.7 Enumeration Type Documentation

```
6.13.7.1 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.

6.14 Time Measurement Driver.

6.14.1 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.

6.14.2 Function Documentation

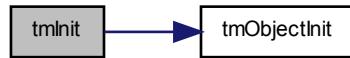
6.14.2.1 void tmInit (void)

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:



6.14.2.2 void tmObjectInit (TimeMeasurement * tmp)

Initializes a [TimeMeasurement](#) object.

Parameters

out	<i>tmp</i> pointer to a TimeMeasurement structure
-----	---

Function Class:

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

6.14.3 Define Documentation

6.14.3.1 #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	<i>tmp</i> pointer to a TimeMeasurement structure
--------	---

Function Class:

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

6.14.3.2 #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.

6.14.4 Typedef Documentation

6.14.4.1 [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.

6.15 UART Driver

6.15.1 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.

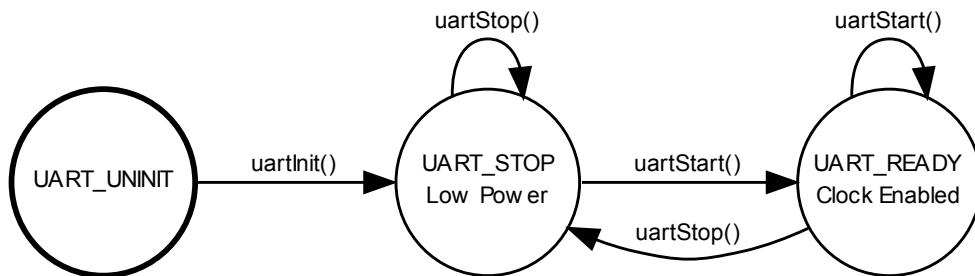
If your application requires a synchronous buffered driver then the [Serial Driver](#) should be used instead.

Precondition

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

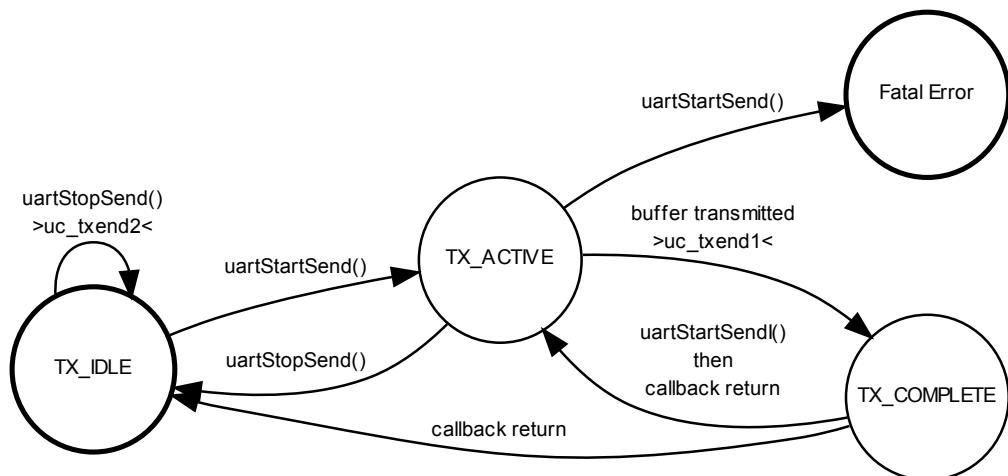
6.15.2 Driver State Machine

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).



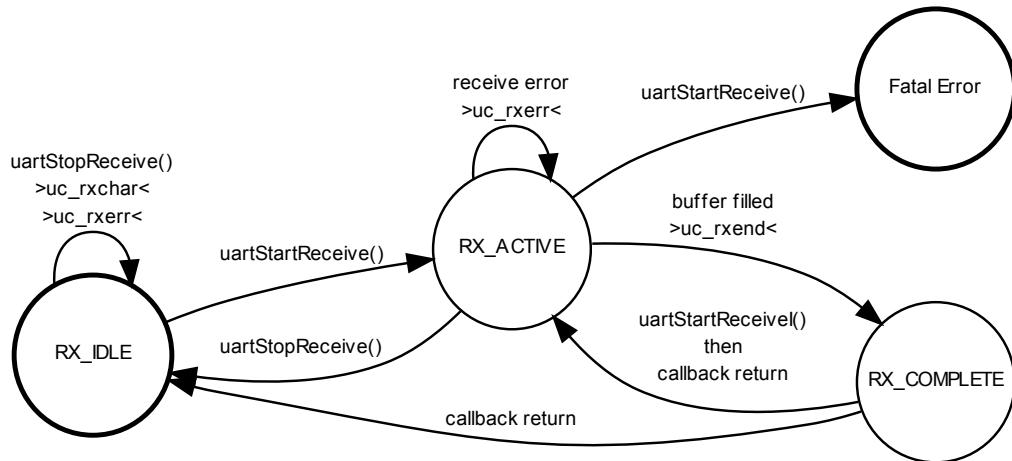
6.15.2.1 Transmitter sub State Machine

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.



6.15.2.2 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 [uartStartReceivev](#) ([UARTDriver](#) *uartp, size_t n, void *rxbuf)
Starts a receive operation on the UART peripheral.
- size_t [uartStopReceive](#) ([UARTDriver](#) *uartp)
Stops any ongoing receive operation.
- size_t [uartStopReceivev](#) ([UARTDriver](#) *uartp)

- **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.

6.15.3 Function Documentation

6.15.3.1 void uartInit (void)

UART Driver initialization.

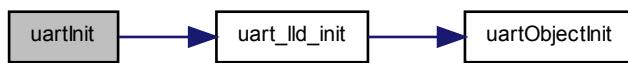
Note

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:



6.15.3.2 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.

6.15.3.3 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:

**6.15.3.4 void uartStop (**UARTDriver** * *uartp*)**

Deactivates the UART peripheral.

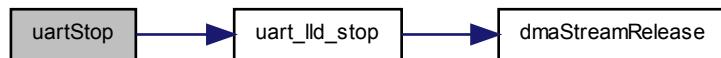
Parameters

in	<i>uartp</i> pointer to the UARTDriver object
----	--

Function Class:

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

Here is the call graph for this function:

**6.15.3.5 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 UARTDriver object
in	<i>n</i> number of data frames to send
in	<i>txbuf</i> the pointer to the transmit buffer

Function Class:

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

Here is the call graph for this function:



6.15.3.6 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 UARTDriver object
in	<i>n</i>	number of data frames to send
in	<i>txbuf</i>	the pointer to the transmit buffer

Function Class:

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

Here is the call graph for this function:



6.15.3.7 **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:

**6.15.3.8 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:



6.15.3.9 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 UARTDriver 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:



6.15.3.10 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 UARTDriver 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:



6.15.3.11 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:



6.15.3.12 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:

**6.15.3.13 CH_IRQ_HANDLER (USART1_IRQHandler)**

USART1 IRQ handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.15.3.14 CH_IRQ_HANDLER (USART2_IRQHandler)

USART2 IRQ handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.15.3.15 CH_IRQ_HANDLER (USART3_IRQHandler)

USART3 IRQ handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.15.3.16 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:

**6.15.3.17 void uart_lld_start(UARTDriver * uartp)**

Configures and activates the UART peripheral.

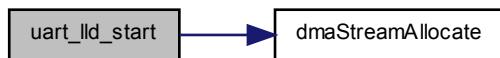
Parameters

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:

**6.15.3.18 void uart_lld_stop(UARTDriver * uartp)**

Deactivates the UART peripheral.

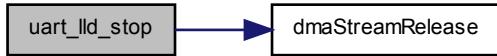
Parameters

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:



6.15.3.19 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.

6.15.3.20 **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.

6.15.3.21 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.

6.15.3.22 `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.

6.15.4 Variable Documentation

6.15.4.1 `UARTDriver UARTD1`

USART1 UART driver identifier.

6.15.4.2 `UARTDriver UARTD2`

USART2 UART driver identifier.

6.15.4.3 `UARTDriver UARTD3`

USART3 UART driver identifier.

6.15.5 Define Documentation

6.15.5.1 `#define UART_NO_ERROR 0`

No pending conditions.

6.15.5.2 `#define UART_PARITY_ERROR 4`

Parity error happened.

6.15.5.3 #define UART_FRAMING_ERROR 8

Framing error happened.

6.15.5.4 #define UART_OVERRUN_ERROR 16

Overflow happened.

6.15.5.5 #define UART_NOISE_ERROR 32

Noise on the line.

6.15.5.6 #define UART_BREAK_DETECTED 64

Break detected.

6.15.5.7 #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.

6.15.5.8 #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.

6.15.5.9 #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.

6.15.5.10 #define STM32_UART_USART1_IRQ_PRIORITY 12

USART1 interrupt priority level setting.

6.15.5.11 #define STM32_UART_USART2_IRQ_PRIORITY 12

USART2 interrupt priority level setting.

```
6.15.5.12 #define STM32_UART_USART3_IRQ_PRIORITY 12
```

USART3 interrupt priority level setting.

```
6.15.5.13 #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.

```
6.15.5.14 #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.

```
6.15.5.15 #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.

```
6.15.5.16 #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.

```
6.15.5.17 #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.

```
6.15.5.18 #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.

```
6.15.5.19 #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.

```
6.15.5.20 #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.

```
6.15.5.21 #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.

```
6.15.5.22 #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.

6.15.6 Typedef Documentation

```
6.15.6.1 typedef uint32_t uartflags_t
```

UART driver condition flags type.

```
6.15.6.2 typedef struct UARTDriver UARTDriver
```

Structure representing an UART driver.

```
6.15.6.3 typedef void(* uartcb_t)(UARTDriver *uartp)
```

Generic UART notification callback type.

Parameters

in *uartp* pointer to the [UARTDriver](#) object

6.15.6.4 `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

6.15.6.5 `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

6.15.7 Enumeration Type Documentation**6.15.7.1 `enum uartstate_t`**

Driver state machine possible states.

Enumerator:

`UART_UNINIT` Not initialized.

`UART_STOP` Stopped.

`UART_READY` Ready.

6.15.7.2 `enum uarttxstate_t`

Transmitter state machine states.

Enumerator:

`UART_TX_IDLE` Not transmitting.

`UART_TX_ACTIVE` Transmitting.

`UART_TX_COMPLETE` Buffer complete.

6.15.7.3 `enum uartrxstate_t`

Receiver state machine states.

Enumerator:

`UART_RX_IDLE` Not receiving.

`UART_RX_ACTIVE` Receiving.

`UART_RX_COMPLETE` Buffer complete.

6.16 USB Driver

6.16.1 Detailed Description

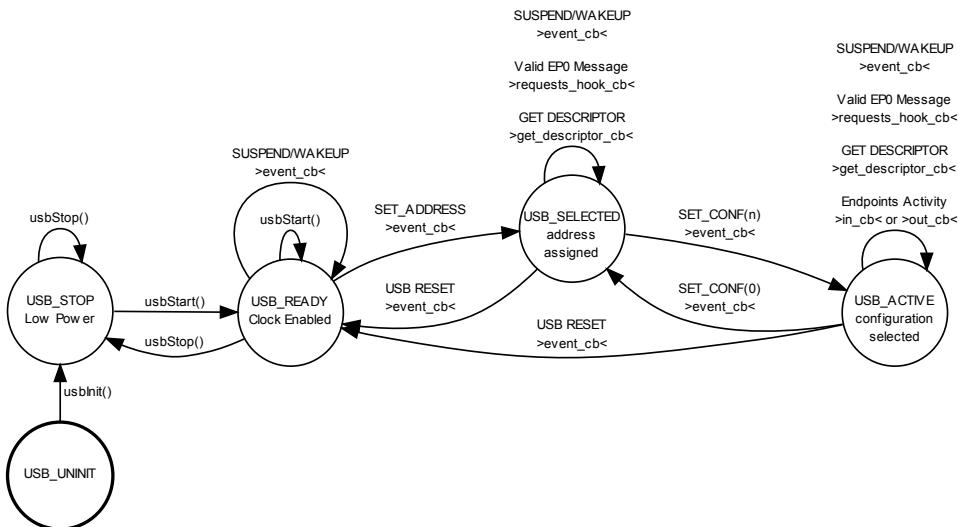
Generic USB Driver. This module implements a generic USB (Universal Serial Bus) driver supporting device-mode operations.

Precondition

In order to use the USB driver the `HAL_USE_USB` option must be enabled in `halconf.h`.

6.16.2 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).



6.16.3 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.

6.16.3.1 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.

6.16.3.2 USB Endpoints

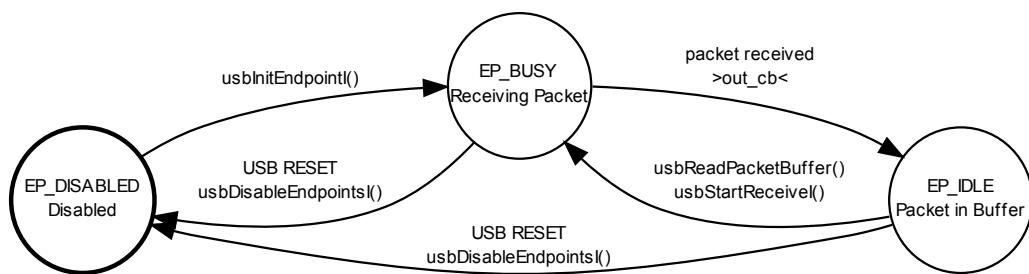
USB endpoints are the objects that the application uses to exchange data with the host. There are two kind of endpoints:

- **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.

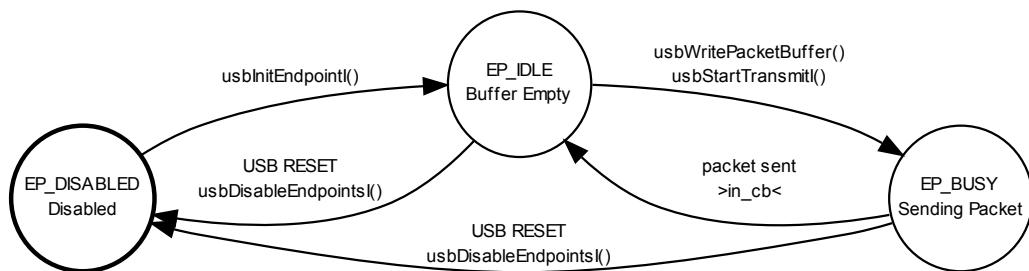
In ChibiOS/RT the endpoints can be configured in two distinct ways:

- **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:

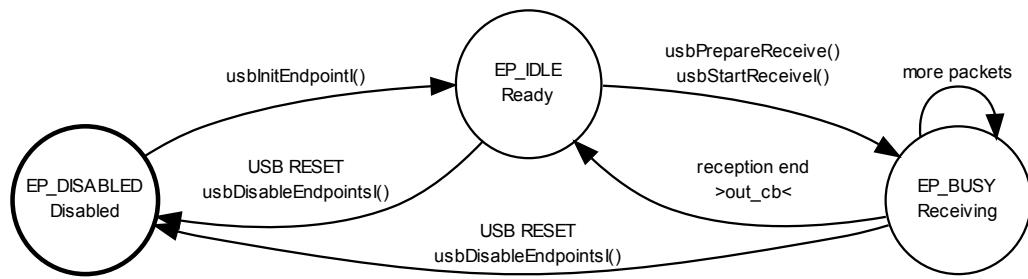


States diagram for IN endpoints in packet mode:

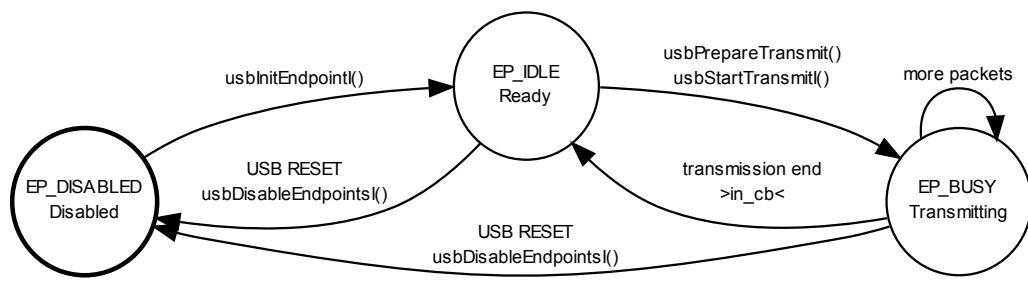


- **Transaction Mode.** In this mode the driver invokes a callback only after a large, potentially multi-packet, transfer has been completed, a callback is invoked only at the end of the transfer.

States diagram for OUT endpoints in transaction mode:



States diagram for IN endpoints in transaction mode:



6.16.3.3 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.

6.16.3.4 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.

6.16.4 Function Documentation

6.16.4.1 void usblInit (void)

USB Driver initialization.

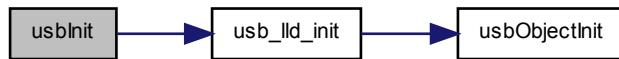
Note

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:



6.16.4.2 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.

6.16.4.3 void usbStart (USBDriver * usbp, const USBConfig * config)

Configures and activates the USB peripheral.

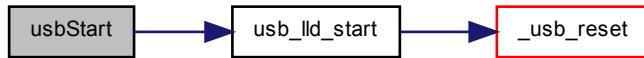
Parameters

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:



6.16.4.4 void usbStop (**USBDriver** * *usbp*)

Deactivates the USB peripheral.

Parameters

in	<i>usbp</i> pointer to the USBDriver object
----	--

Function Class:

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

Here is the call graph for this function:



6.16.4.5 void usblInitEndpointl (**USBDriver** * *usbp*, **usbep_t** *ep*, const **USBEndpointConfig** * *epcp*)

Enables an endpoint.

This function enables an endpoint, both IN and/or OUT directions depending on the configuration structure.

Note

This function must be invoked in response of a SET_CONFIGURATION or SET_INTERFACE message.

Parameters

in	<i>usbp</i> pointer to the USBDriver object
in	<i>ep</i> endpoint number
in	<i>epcp</i> the endpoint configuration

Function Class:

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

Here is the call graph for this function:



6.16.4.6 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 USBDriver object
----	---

Function Class:

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

Here is the call graph for this function:



6.16.4.7 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 USBDriver object
in	<i>ep</i> endpoint number

Returns

The operation status.

Return values

<i>FALSE</i>	Operation started successfully.
<i>TRUE</i>	Endpoint busy, operation not started.

Function Class:

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

Here is the call graph for this function:

**6.16.4.8 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 USBDriver object
in	<i>ep</i>	endpoint number

Returns

The operation status.

Return values

<i>FALSE</i>	Operation started successfully.
<i>TRUE</i>	Endpoint busy, operation not started.

Function Class:

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

Here is the call graph for this function:



6.16.4.9 `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:



6.16.4.10 `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:

**6.16.4.11 void _usb_reset(USBDriver * usbp)**

USB reset routine.

This function must be invoked when an USB bus reset condition is detected.

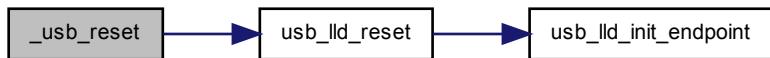
Parameters

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:

**6.16.4.12 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.

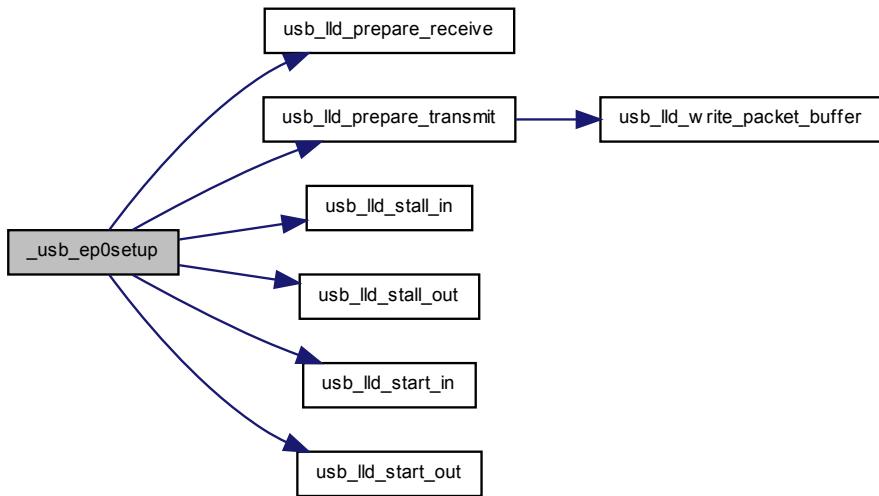
Parameters

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:

**6.16.4.13 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.

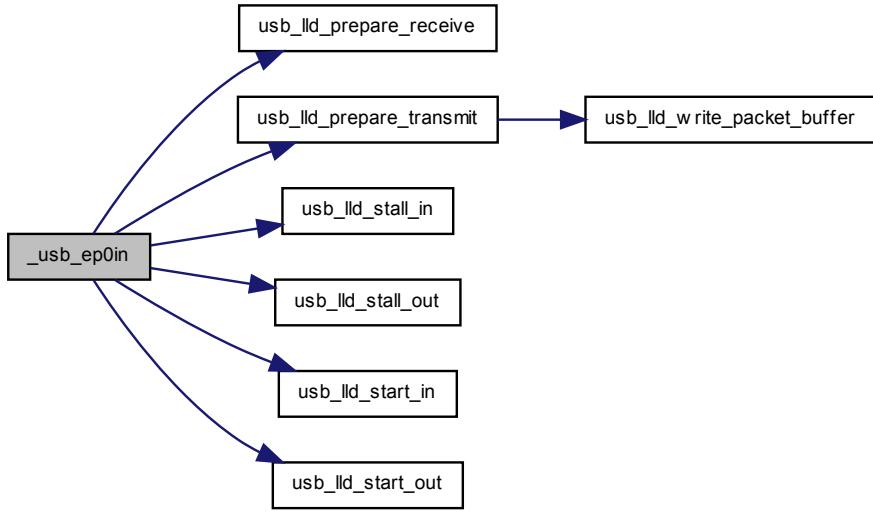
Parameters

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:



6.16.4.14 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.

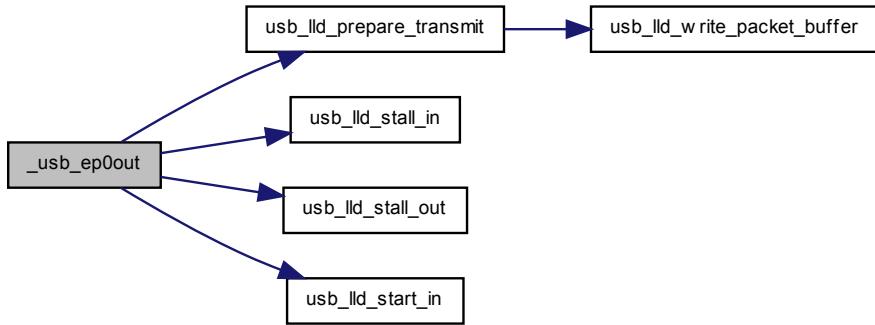
Parameters

in	usbp	pointer to the USBDriver object
in	ep	endpoint number, always zero

Function Class:

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

Here is the call graph for this function:



6.16.4.15 CH_IRQ_HANDLER (Vector8C)

USB high priority interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

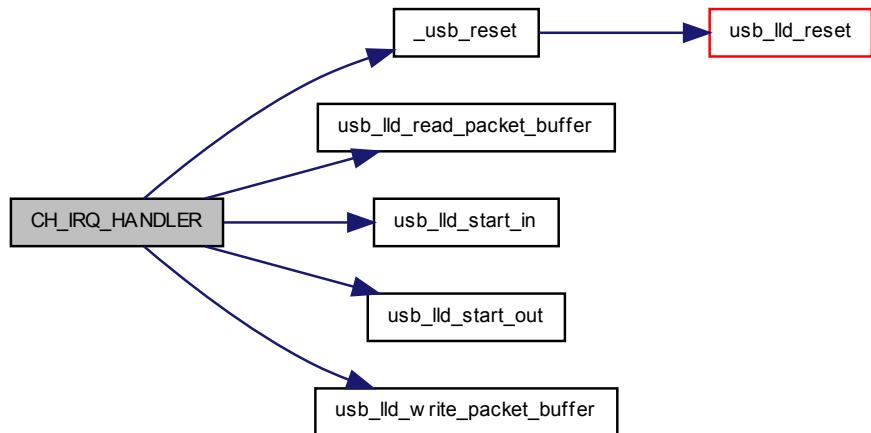
6.16.4.16 CH_IRQ_HANDLER (Vector90)

USB low priority interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

Here is the call graph for this function:



6.16.4.17 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:



6.16.4.18 void usb_lld_start(USBDriver * usbp)

Configures and activates the USB peripheral.

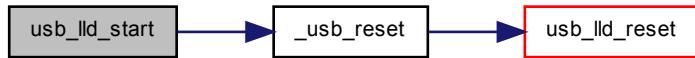
Parameters

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:



6.16.4.19 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.

6.16.4.20 void usb_lld_reset (**USBDriver** * *usbp*)

USB low level reset routine.

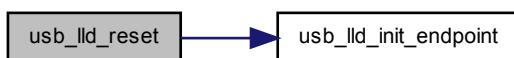
Parameters

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:



6.16.4.21 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.

6.16.4.22 void usb_lld_init_endpoint (**USBDriver** * *usbp*, **usbep_t** *ep*)

Enables an endpoint.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

6.16.4.23 void usb_lld_disable_endpoints (**USBDriver** * *usbp*)

Disables all the active endpoints except the endpoint zero.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
----	-------------	--

Function Class:

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

6.16.4.24 **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 USBDriver 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.

6.16.4.25 **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.

6.16.4.26 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.

6.16.4.27 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.

6.16.4.28 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 USBDriver 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.

6.16.4.29 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 USBDriver 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.

6.16.4.30 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 USBDriver object
in	<i>ep</i>	endpoint number
in	<i>buf</i>	buffer where to fetch the data to be transmitted
in	<i>n</i>	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:

**6.16.4.31 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 USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

6.16.4.32 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 USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

6.16.4.33 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 USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

6.16.4.34 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 USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

6.16.4.35 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 USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

6.16.4.36 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 USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

6.16.5 Variable Documentation

6.16.5.1 **USBDriver USBD1**

USB1 driver identifier.

6.16.5.2 **USBInEndpointState { ... } in**

IN EP0 state.

6.16.5.3 **USBOutEndpointState { ... } out**

OUT EP0 state.

6.16.6 Define Documentation

6.16.6.1 #define USB_DESC_INDEX(*i*) ((uint8_t)(*i*))

Helper macro for index values into descriptor strings.

6.16.6.2 #define USB_DESC_BYTE(*b*) ((uint8_t)(*b*))

Helper macro for byte values into descriptor strings.

6.16.6.3 #define USB_DESC_WORD(*w*)

Value:

```
(uint8_t) ((w) & 255),  
          \  
(uint8_t) (((w) >> 8) & 255)
```

Helper macro for word values into descriptor strings.

6.16.6.4 #define USB_DESC_BCD(*bcd*)

Value:

```
(uint8_t) ((bcd) & 255),  
          \  
(uint8_t) (((bcd) >> 8) & 255)
```

Helper macro for BCD values into descriptor strings.

**6.16.6.5 #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.

**6.16.6.6 #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.

```
6.16.6.7 #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.

```
6.16.6.8 #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.

```
6.16.6.9 #define USB_EP_MODE_TYPE 0x0003
```

Endpoint type mask.

```
6.16.6.10 #define USB_EP_MODE_TYPE_CTRL 0x0000
```

Control endpoint.

```
6.16.6.11 #define USB_EP_MODE_TYPE_ISOC 0x0001
```

Isochronous endpoint.

```
6.16.6.12 #define USB_EP_MODE_TYPE_BULK 0x0002
```

Bulk endpoint.

```
6.16.6.13 #define USB_EP_MODE_TYPE_INTR 0x0003
```

Interrupt endpoint.

```
6.16.6.14 #define USB_EP_MODE_TRANSACTION 0x0000
```

Transaction mode.

```
6.16.6.15 #define USB_EP_MODE_PACKET 0x0010
```

Packet mode enabled.

```
6.16.6.16 #define usbConnectBus( usbp ) usb_lld_connect_bus(usbp)
```

Connects the USB device.

```
6.16.6.17 #define usbDisconnectBus( usbp ) usb_lld_disconnect_bus(usbp)
```

Disconnect the USB device.

```
6.16.6.18 #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.

```
6.16.6.19 #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.

```
6.16.6.20 #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.

6.16.6.21 #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

<i>in</i>	<i>usbp</i>	pointer to the USBDriver object
<i>in</i>	<i>ep</i>	endpoint number
<i>out</i>	<i>buf</i>	buffer where to copy the packet data
<i>in</i>	<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:

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

6.16.6.22 #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

<i>in</i>	<i>usbp</i>	pointer to the USBDriver object
<i>in</i>	<i>ep</i>	endpoint number
<i>in</i>	<i>buf</i>	buffer where to fetch the packet data
<i>in</i>	<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.

6.16.6.23 #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 USBDriver 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.

6.16.6.24 #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 USBDriver object
in	<i>ep</i>	endpoint number
in	<i>buf</i>	buffer where to fetch the data to be transmitted
in	<i>n</i>	maximum number of bytes to copy

Function Class:

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

6.16.6.25 #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.

6.16.6.26 #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.

6.16.6.27 #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.

```
6.16.6.28 #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.

```
6.16.6.29 #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.

```
6.16.6.30 #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.

6.16.6.31 #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 USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

6.16.6.32 #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 USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

6.16.6.33 #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 USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

6.16.6.34 #define USB_ENDPOINTS_NUMBER 7

Number of the available endpoints.

This value does not include the endpoint 0 which is always present.

6.16.6.35 #define STM32_USB_BASE (APB1PERIPH_BASE + 0x5C00)

USB registers block numeric address.

6.16.6.36 #define STM32_USBRAM_BASE (APB1PERIPH_BASE + 0x6000)

USB RAM numeric address.

6.16.6.37 #define STM32_USB ((stm32_usb_t *)STM32_USB_BASE)

Pointer to the USB registers block.

6.16.6.38 #define STM32_USBRAM ((uint32_t *)STM32_USBRAM_BASE)

Pointer to the USB RAM.

6.16.6.39 #define USB_PMA_SIZE 512

Size of the dedicated packet memory.

6.16.6.40 #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.

6.16.6.41 #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.

6.16.6.42 #define USB_ADDR2PTR(addr) ((uint32_t *)((addr) * 2 + STM32_USBRAM_BASE))

Converts from a PMA address to a physical address.

6.16.6.43 #define USB_MAX_ENDPOINTS USB_ENDPOINTS_NUMBER

Maximum endpoint address.

```
6.16.6.44 #define USB_SET_ADDRESS_MODE USB_LATE_SET_ADDRESS
```

This device requires the address change after the status packet.

```
6.16.6.45 #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.

```
6.16.6.46 #define STM32_USB_LOW_POWER_ON_SUSPEND FALSE
```

Enables the USB device low power mode on suspend.

```
6.16.6.47 #define STM32_USB_USB1_HP_IRQ_PRIORITY 6
```

USB1 interrupt priority level setting.

```
6.16.6.48 #define STM32_USB_USB1_LP_IRQ_PRIORITY 14
```

USB1 interrupt priority level setting.

```
6.16.6.49 #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.

```
6.16.6.50 #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.

```
6.16.6.51 #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.

```
6.16.6.52 #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.

6.16.7 Typedef Documentation

```
6.16.7.1 typedef struct USBDriver USBDriver
```

Type of a structure representing an USB driver.

```
6.16.7.2 typedef uint8_t usbep_t
```

Type of an endpoint identifier.

```
6.16.7.3 typedef void(* usbcallback_t)(USBDriver *usbp)
```

Type of an USB generic notification callback.

Parameters

in *usbp* pointer to the `USBDriver` object triggering the callback

6.16.7.4 `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

6.16.7.5 `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

6.16.7.6 `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.

6.16.7.7 `typedef const USBDescriptor*(* usbgetdescriptor_t)(USBDriver *usbp, uint8_t dtype, uint8_t dindex, uint16_t lang)`

Type of an USB descriptor-retrieving callback.

6.16.8 Enumeration Type Documentation**6.16.8.1 `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.

6.16.8.2 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.

6.16.8.3 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.

6.16.8.4 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.

6.17 STM32L1xx Drivers

6.17.1 Detailed Description

This section describes all the supported drivers on the STM32L1xx platform and the implementation details of the single drivers.

Modules

- STM32L1xx Initialization Support
- STM32L1xx ADC Support
- STM32L1xx EXT Support
- STM32L1xx GPT Support
- STM32L1xx ICU Support
- STM32L1xx PAL Support
- STM32L1xx PWM Support
- STM32L1xx Serial Support
- STM32L1xx SPI Support
- STM32L1xx UART Support
- STM32L1xx USB Support
- STM32L1xx Platform Drivers

6.18 STM32L1xx Initialization Support

The STM32L1xx HAL support is responsible for system initialization.

6.18.1 Supported HW resources

- PLL1.
- RCC.
- Flash.

6.18.2 STM32L1xx HAL driver implementation features

- PLL 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.

6.19 STM32L1xx ADC Support

The STM32L1xx ADC driver supports the ADC peripherals using DMA channels for maximum performance.

6.19.1 Supported HW resources

- ADC1.
- DMA1.

6.19.2 STM32L1xx 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 and ADC errors detection.

6.20 STM32L1xx EXT Support

The STM32L1xx EXT driver uses the EXTI peripheral.

6.20.1 Supported HW resources

- EXTI.

6.20.2 STM32L1xx 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.

6.21 STM32L1xx GPT Support

The STM32L1xx GPT driver uses the TIMx peripherals.

6.21.1 Supported HW resources

- TIM2.
- TIM3.
- TIM4.

6.21.2 STM32L1xx 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.

6.22 STM32L1xx ICU Support

The STM32L1xx ICU driver uses the TIMx peripherals.

6.22.1 Supported HW resources

- TIM2.
- TIM3.
- TIM4.

6.22.2 STM32L1xx 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.

6.23 STM32L1xx PAL Support

The STM32L1xx PAL driver uses the GPIO peripherals.

6.23.1 Supported HW resources

- GPIOA.
- GPIOB.
- GPIOC.
- GPIOD.
- GPIOE.
- GPIOH.

6.23.2 STM32L1xx 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.

6.23.3 Supported PAL setup modes

The STM32L1xx 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_ALTERNATE (non standard).

Any attempt to setup an invalid mode is ignored.

6.23.4 Suboptimal behavior

The STM32L1xx 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.

6.24 STM32L1xx PWM Support

The STM32L1xx PWM driver uses the TIMx peripherals.

6.24.1 Supported HW resources

- TIM1.
- TIM2.
- TIM3.
- TIM4.

6.24.2 STM32L1xx 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.

6.25 STM32L1xx Serial Support

The STM32L1xx Serial driver uses the USART/UART peripherals in a buffered, interrupt driven, implementation.

6.25.1 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).

6.25.2 STM32L1xx 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.

6.26 STM32L1xx SPI Support

The SPI driver supports the STM32L1xx SPI peripherals using DMA channels for maximum performance.

6.26.1 Supported HW resources

- SPI1.
- SPI2.
- SPI3 (where present).
- DMA1.
- DMA2 (where present).

6.26.2 STM32L1xx 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.

6.27 STM32L1xx UART Support

The UART driver supports the STM32L1xx USART peripherals using DMA channels for maximum performance.

6.27.1 Supported HW resources

The UART driver can support any of the following hardware resources:

- USART1.
- USART2.
- USART3 (where present).
- DMA1.

6.27.2 STM32L1xx 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.

6.28 STM32L1xx USB Support

The USB driver supports the STM32L1xx USB peripheral.

6.28.1 Supported HW resources

The USB driver can support any of the following hardware resources:

- USB.

6.28.2 STM32L1xx 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.

6.29 STM32L1xx Platform Drivers

6.29.1 Detailed Description

Platform support drivers. Platform drivers do not implement HAL standard driver templates, their role is to support platform specific functionalities.

Modules

- [STM32L1xx DMA Support](#)
- [STM32L1xx RCC Support](#)

6.30 STM32L1xx DMA Support

6.30.1 Detailed Description

This DMA helper driver is used by the other drivers in order to access the shared DMA resources in a consistent way.

6.30.2 Supported HW resources

The DMA driver can support any of the following hardware resources:

- DMA1.

6.30.3 STM32L1xx 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 ISRs 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.
- 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.

DMA streams identifiers

- `#define STM32_DMA_STREAM_ID(dma, stream) ((stream) - 1)`
Returns an unique numeric identifier for a DMA 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.
- `#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)`

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_BYT 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_BYT 0`
- `#define STM32_DMA_CR_MSIZE_HWORD DMA_CCR1_MSIZE_0`
- `#define STM32_DMA_CR_MSIZE_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 7`
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.

Typedefs

- typedef void(* **stm32_dmaisr_t**)(void *p, uint32_t flags)

STM32 DMA ISR function type.

6.30.4 Function Documentation

6.30.4.1 CH_IRQ_HANDLER (DMA1_Ch1_IRQHandler)

DMA1 stream 1 shared interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.30.4.2 CH_IRQ_HANDLER (DMA1_Ch2_IRQHandler)

DMA1 stream 2 shared interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.30.4.3 CH_IRQ_HANDLER (DMA1_Ch3_IRQHandler)

DMA1 stream 3 shared interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.30.4.4 CH_IRQ_HANDLER (DMA1_Ch4_IRQHandler)

DMA1 stream 4 shared interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.30.4.5 CH_IRQ_HANDLER (DMA1_Ch5_IRQHandler)

DMA1 stream 5 shared interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.30.4.6 CH_IRQ_HANDLER (DMA1_Ch6_IRQHandler)

DMA1 stream 6 shared interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.30.4.7 CH_IRQ_HANDLER (DMA1_Ch7_IRQHandler)

DMA1 stream 7 shared interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.30.4.8 void dmalinit (void)

STM32 DMA helper initialization.

Function Class:

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

6.30.4.9 bool_t dmaStreamAllocate (const stm32_dma_stream_t * *dmaslp*, 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>dmaslp</i>	pointer to a stm32_dma_stream_t 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.

6.30.4.10 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.

6.30.5 Variable Documentation**6.30.5.1 const [stm32_dma_stream_t](#) _stm32_dma_streams[STM32_DMA_STREAMS]****Initial value:**

```
{
{DMA1_Channel1, &DMA1->IFCR, 0, 0, DMA1_Channel1 IRQn},
{DMA1_Channel2, &DMA1->IFCR, 4, 1, DMA1_Channel2 IRQn},
{DMA1_Channel3, &DMA1->IFCR, 8, 2, DMA1_Channel3 IRQn},
{DMA1_Channel4, &DMA1->IFCR, 12, 3, DMA1_Channel4 IRQn},
{DMA1_Channel5, &DMA1->IFCR, 16, 4, DMA1_Channel5 IRQn},
{DMA1_Channel6, &DMA1->IFCR, 20, 5, DMA1_Channel6 IRQn},
{DMA1_Channel7, &DMA1->IFCR, 24, 6, DMA1_Channel7 IRQn}
}
```

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.

6.30.6 Define Documentation

6.30.6.1 #define STM32_DMA1_STREAMS_MASK 0x0000007F

Mask of the DMA1 streams in `dma_streams_mask`.

6.30.6.2 #define STM32_DMA2_STREAMS_MASK 0x00000F80

Mask of the DMA2 streams in `dma_streams_mask`.

6.30.6.3 #define STM32_DMA_CCR_RESET_VALUE 0x00000000

Post-reset value of the stream CCR register.

6.30.6.4 #define STM32_DMA_STREAMS 7

Total number of DMA streams.

Note

This is the total number of streams among all the DMA units.

6.30.6.5 #define STM32_DMA_ISR_MASK 0x0F

Mask of the ISR bits passed to the DMA callback functions.

6.30.6.6 #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.

6.30.6.7 #define STM32_DMA_STREAM_ID(*dma*, *stream*) ((*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.

6.30.6.8 #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.

6.30.6.9 #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

<i>The</i>	check result.
<i>FALSE</i>	<i>id</i> does not belong to the mask.
<i>TRUE</i>	<i>id</i> belongs to the mask.

6.30.6.10 #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.

6.30.6.11 #define STM32_DMA_CR_DMEIE 0

Ignored by normal DMA.

6.30.6.12 #define STM32_DMA_CR_CHSEL_MASK 0

Ignored by normal DMA.

6.30.6.13 #define STM32_DMA_CR_CHSEL(*n*) 0

Ignored by normal DMA.

6.30.6.14 #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	<code>dmastp</code>	pointer to a <code>stm32_dma_stream_t</code> structure
in	<code>addr</code>	value to be written in the CPAR register

Function Class:

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

6.30.6.15 #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	<code>dmastp</code>	pointer to a <code>stm32_dma_stream_t</code> structure
in	<code>addr</code>	value to be written in the CMAR register

Function Class:

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

6.30.6.16 #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	<code>dmastp</code>	pointer to a <code>stm32_dma_stream_t</code> structure
in	<code>size</code>	value to be written in the CNDTR register

Function Class:

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

6.30.6.17 #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	<code>dmastp</code>	pointer to a <code>stm32_dma_stream_t</code> structure
----	---------------------	--

Returns

The number of transfers to be performed.

Function Class:

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

6.30.6.18 #define `dmaStreamSetMode(dmastp, mode)`

Value:

```
{
    \n
    (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	<i>dmastp</i>	pointer to a <code>stm32_dma_stream_t</code> structure
in	<i>mode</i>	value to be written in the CCR register

Function Class:

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

6.30.6.19 #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	<i>dmastp</i>	pointer to a <code>stm32_dma_stream_t</code> structure
----	---------------	--

Function Class:

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

6.30.6.20 #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.

6.30.6.21 #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.

6.30.6.22 #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"> • <code>STM32_DMA_CR_MINC</code> • <code>STM32_DMA_CR_PINC</code> • <code>STM32_DMA_CR_DIR_M2M</code> • <code>STM32_DMA_CR_EN</code>
in	<code>src</code>	source address
in	<code>dst</code>	destination address
in	<code>n</code>	number of data units to copy

6.30.6.23 #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
----	---------------------	--

6.30.7 Typedef Documentation

6.30.7.1 `typedef void(* stm32_dmaISR_t)(void *p, uint32_t flags)`

STM32 DMA ISR function type.

Parameters

in	<code>p</code>	parameter for the registered function
in	<code>flags</code>	pre-shifted content of the ISR register, the bits are aligned to bit zero

6.31 STM32L1xx RCC Support

6.31.1 Detailed Description

This RCC helper driver is used by the other drivers in order to access the shared RCC resources in a consistent way.

6.31.2 Supported HW resources

- RCC.

6.31.3 STM32L1xx 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.

DMA peripheral 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() rccResetAHB(RCC_AHBRSTR_DMA1RST)`
Resets the DMA1 peripheral.

PWR interface specific RCC operations

- `#define rccEnablePWRInterface(lp) rccEnableAPB1(RCC_APB1ENR_PWREN, lp)`
Enables the PWR interface clock.
- `#define rccDisablePWRInterface(lp) rccDisableAPB1(RCC_APB1ENR_PWREN, lp)`
Disables PWR interface clock.
- `#define rccResetPWRInterface() rccResetAPB1(RCC_APB1RSTR_PWR_RST)`
Resets the PWR interface.

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.

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.

TIM peripherals specific RCC operations

- `#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.

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.

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.

6.31.4 Define Documentation

6.31.4.1 #define rccEnableAPB1(*mask*, *lp*)

Value:

```
{
    RCC->APB1ENR |= (mask);
    if (lp)
        RCC->APB1LPENR |= (mask);
}
```

Enables the clock of one or more peripheral on the APB1 bus.

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.

6.31.4.2 #define rccDisableAPB1(*mask*, *lp*)

Value:

```
{
    RCC->APB1ENR &= ~ (mask);
    if (lp)
        RCC->APB1LPENR &= ~ (mask);
}
```

Disables the clock of one or more peripheral on the APB1 bus.

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.

6.31.4.3 #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.

6.31.4.4 #define rccEnableAPB2(*mask*, *lp*)**Value:**

```
{
    RCC->APB2ENR |= (mask);
    if (lp)
        RCC->APB2LPENR |= (mask);
}
```

Enables the clock of one or more peripheral on the APB2 bus.

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.

6.31.4.5 #define rccDisableAPB2(*mask*, *lp*)**Value:**

```
{
    RCC->APB2ENR &= ~ (mask);
    if (lp)
        RCC->APB2LPENR &= ~ (mask);
}
```

Disables the clock of one or more peripheral on the APB2 bus.

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.

6.31.4.6 #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.

6.31.4.7 #define rccEnableAHB(*mask*, *lp*)**Value:**

```
{
    RCC->AHBENR |= (mask);
    if (lp)
        RCC->AHBLPENR |= (mask);
}
```

Enables the clock of one or more peripheral on the AHB bus.

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.

6.31.4.8 #define rccDisableAHB(*mask*, *lp*)**Value:**

```
{
    RCC->AHBENR &= ~ (mask);
    if (lp)
        RCC->AHBLPENR &= ~ (mask);
}
```

Disables the clock of one or more peripheral on the AHB bus.

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.

6.31.4.9 #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.

6.31.4.10 #define rccEnableADC1(*lp*) rccEnableAPB2(RCC_APB2ENR_ADC1EN, *lp*)

Enables the ADC1 peripheral clock.

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.

6.31.4.11 #define rccDisableADC1(*lp*) rccDisableAPB2(RCC_APB2ENR_ADC1EN, *lp*)

Disables the ADC1 peripheral clock.

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.

6.31.4.12 #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.

6.31.4.13 #define rccEnableDMA1(*lp*) rccEnableAHB(RCC_AHBENR_DMA1EN, *lp*)

Enables the DMA1 peripheral clock.

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.

6.31.4.14 #define rccDisableDMA1(*lp*) rccDisableAHB(RCC_AHBENR_DMA1EN, *lp*)

Disables the DMA1 peripheral clock.

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.

6.31.4.15 #define rccResetDMA1() rccResetAHB(RCC_AHBRSTR_DMA1RST)

Resets the DMA1 peripheral.

Function Class:

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

6.31.4.16 #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.

6.31.4.17 #define rccDisablePWRInterface(*lp*) rccDisableAPB1(RCC_APB1ENR_PWREN, *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.

6.31.4.18 #define rccResetPWRInterface() rccResetAPB1(RCC_APB1RSTR_PWRRST)

Resets the PWR interface.

Function Class:

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

6.31.4.19 #define rccEnableI2C1(*lp*) rccEnableAPB1(RCC_APB1ENR_I2C1EN, *lp*)

Enables the I2C1 peripheral clock.

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.

6.31.4.20 #define rccDisableI2C1(*lp*) rccDisableAPB1(RCC_APB1ENR_I2C1EN, *lp*)

Disables the I2C1 peripheral clock.

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.

6.31.4.21 #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.

6.31.4.22 #define rccEnableI2C2(*lp*) rccEnableAPB1(RCC_APB1ENR_I2C2EN, *lp*)

Enables the I2C2 peripheral clock.

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.

6.31.4.23 #define rccDisableI2C2(*lp*) rccDisableAPB1(RCC_APB1ENR_I2C2EN, *lp*)

Disables the I2C2 peripheral clock.

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.

6.31.4.24 #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.

6.31.4.25 #define rccEnableSPI1(*lp*) rccEnableAPB2(RCC_APB2ENR_SPI1EN, *lp*)

Enables the SPI1 peripheral clock.

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.

6.31.4.26 #define rccDisableSPI1(*lp*) rccDisableAPB2(RCC_APB2ENR_SPI1EN, *lp*)

Disables the SPI1 peripheral clock.

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.

6.31.4.27 #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.

6.31.4.28 #define rccEnableSPI2(*lp*) rccEnableAPB1(RCC_APB1ENR_SPI2EN, *lp*)

Enables the SPI2 peripheral clock.

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.

6.31.4.29 #define rccDisableSPI2(*lp*) rccDisableAPB1(RCC_APB1ENR_SPI2EN, *lp*)

Disables the SPI2 peripheral clock.

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.

6.31.4.30 #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.

6.31.4.31 #define rccEnableTIM2(*lp*) rccEnableAPB1(RCC_APB1ENR_TIM2EN, *lp*)

Enables the TIM2 peripheral clock.

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.

6.31.4.32 #define rccDisableTIM2(*lp*) rccDisableAPB1(RCC_APB1ENR_TIM2EN, *lp*)

Disables the TIM2 peripheral clock.

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.

6.31.4.33 #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.

6.31.4.34 #define rccEnableTIM3(*lp*) rccEnableAPB1(RCC_APB1ENR_TIM3EN, *lp*)

Enables the TIM3 peripheral clock.

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.

6.31.4.35 #define rccDisableTIM3(*lp*) rccDisableAPB1(RCC_APB1ENR_TIM3EN, *lp*)

Disables the TIM3 peripheral clock.

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.

6.31.4.36 #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.

6.31.4.37 #define rccEnableTIM4(*lp*) rccEnableAPB1(RCC_APB1ENR_TIM4EN, *lp*)

Enables the TIM4 peripheral clock.

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.

6.31.4.38 #define rccDisableTIM4(*lp*) rccDisableAPB1(RCC_APB1ENR_TIM4EN, *lp*)

Disables the TIM4 peripheral clock.

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.

6.31.4.39 #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.

6.31.4.40 #define rccEnableUSART1(*lp*) rccEnableAPB2(RCC_APB2ENR_USART1EN, *lp*)

Enables the USART1 peripheral clock.

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.

6.31.4.41 #define rccDisableUSART1(*lp*) rccDisableAPB2(RCC_APB2ENR_USART1EN, *lp*)

Disables the USART1 peripheral clock.

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.

6.31.4.42 #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.

6.31.4.43 #define rccEnableUSART2(*lp*) rccEnableAPB1(RCC_APB1ENR_USART2EN, *lp*)

Enables the USART2 peripheral clock.

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.

6.31.4.44 #define rccDisableUSART2(*lp*) rccDisableAPB1(RCC_APB1ENR_USART2EN, *lp*)

Disables the USART2 peripheral clock.

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.

6.31.4.45 #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.

6.31.4.46 #define rccEnableUSART3(*lp*) rccEnableAPB1(RCC_APB1ENR_USART3EN, *lp*)

Enables the USART3 peripheral clock.

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.

6.31.4.47 #define rccDisableUSART3(*lp*) rccDisableAPB1(RCC_APB1ENR_USART3EN, *lp*)

Disables the USART3 peripheral clock.

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.

6.31.4.48 #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.

6.31.4.49 #define rccEnableUSB(*lp*) rccEnableAPB1(RCC_APB1ENR_USBEN, *lp*)

Enables the USB peripheral clock.

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.

6.31.4.50 #define rccDisableUSB(*lp*) rccDisableAPB1(RCC_APB1ENR_USBEN, *lp*)

Disables the USB peripheral clock.

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.

6.31.4.51 #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.

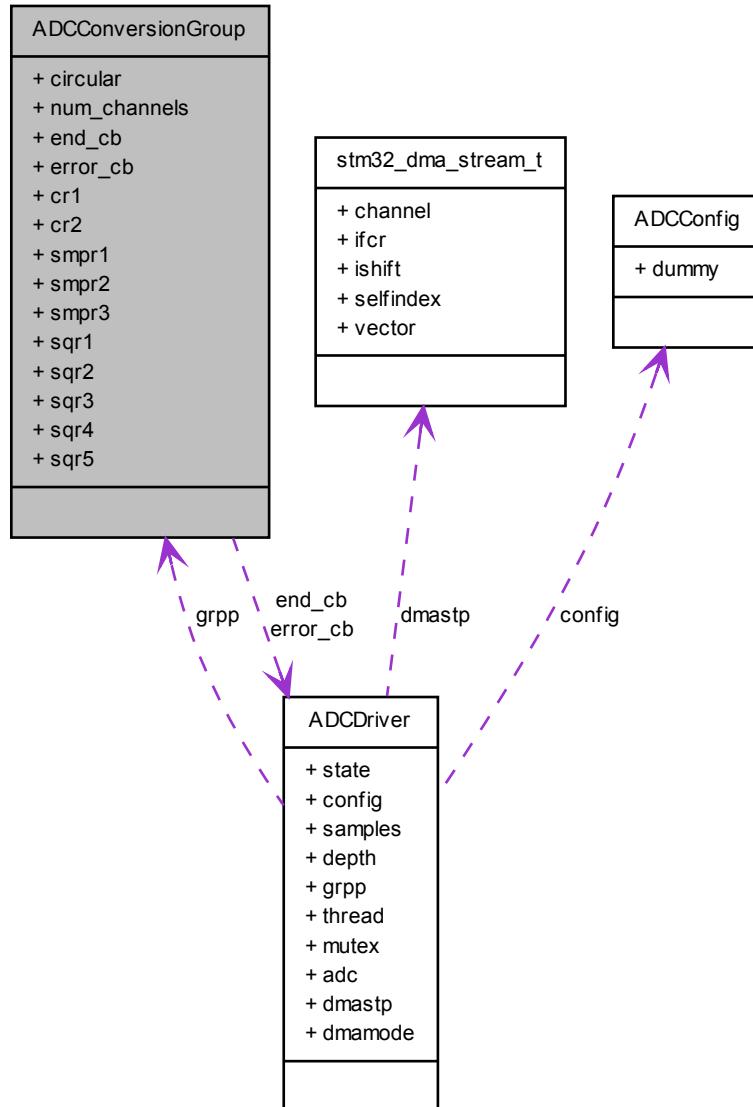
This implementation-dependent structure describes a conversion operation.

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`

- `uint32_t cr2`
ADC CR2 register initialization data.
- `uint32_t smpr1`
ADC SMPR1 register initialization data.
- `uint32_t smpr2`
ADC SMPR2 register initialization data.
- `uint32_t smpr3`
ADC SMPR3 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.
- `uint32_t sqr4`
ADC SQR4 register initialization data.
- `uint32_t sqr5`
ADC SQR5 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 20...25.

7.2.2.8 uint32_t ADCConversionGroup::smpr2

ADC SMPR2 register initialization data.

In this field must be specified the sample times for channels 10...19.

7.2.2.9 uint32_t ADCConversionGroup::smpr3

ADC SMPR3 register initialization data.

In this field must be specified the sample times for channels 0...9.

7.2.2.10 uint32_t ADCConversionGroup::sqr1

ADC SQR1 register initialization data.

Conversion group sequence 25...27 + sequence length.

7.2.2.11 uint32_t ADCConversionGroup::sqr2

ADC SQR2 register initialization data.

Conversion group sequence 19...24.

7.2.2.12 uint32_t ADCConversionGroup::sqr3

ADC SQR3 register initialization data.

Conversion group sequence 13...18.

7.2.2.13 uint32_t ADCConversionGroup::sqr4

ADC SQR3 register initialization data.

Conversion group sequence 7...12.

7.2.2.14 uint32_t ADCConversionGroup::sqr5

ADC SQR3 register initialization data.

Conversion group sequence 1...6.

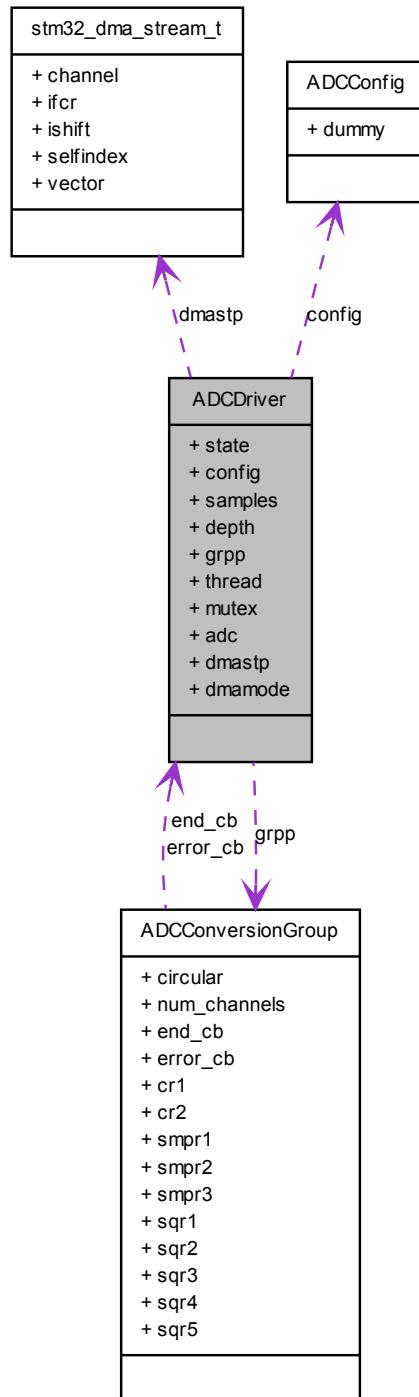
7.3 ADCDriver Struct Reference

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.

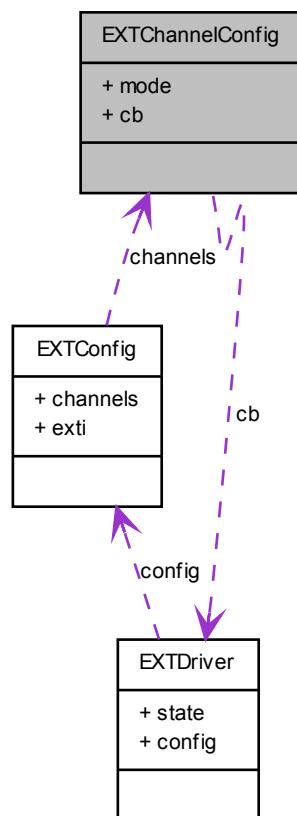
7.4 EXTChannelConfig Struct Reference

7.4.1 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.

7.4.2 Field Documentation

7.4.2.1 `uint32_t EXTChannelConfig::mode`

Channel mode.

7.4.2.2 `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.

7.5 EXTConfig Struct Reference

7.5.1 Detailed Description

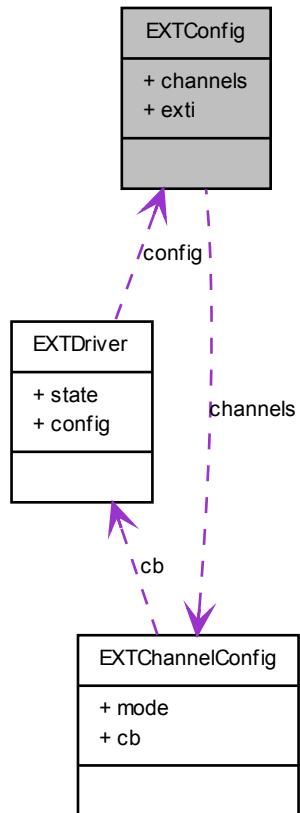
Driver configuration structure.

Note

It could be empty on some architectures.

```
#include <ext_llld.h>
```

Collaboration diagram for EXTConfig:



Data Fields

- `EXTChannelConfig channels [EXT_MAX_CHANNELS]`

Channel configurations.

- `uint16_t exti [4]`

Initialization values for EXTICRx registers.

7.5.2 Field Documentation

7.5.2.1 `EXTChannelConfig EXTConfig::channels[EXT_MAX_CHANNELS]`

Channel configurations.

7.5.2.2 `uint16_t EXTConfig::exti[4]`

Initialization values for EXTICRx registers.

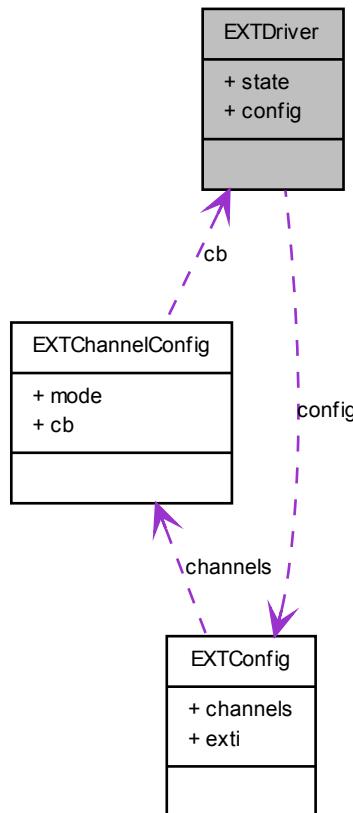
7.6 EXTDriver Struct Reference

7.6.1 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.

7.6.2 Field Documentation

7.6.2.1 extstate_t EXTDriver::state

Driver state.

7.6.2.2 const EXTConfig* EXTDriver::config

Current configuration data.

7.7 GPIO_TypeDef Struct Reference

7.7.1 Detailed Description

STM32 GPIO registers block.

```
#include <pal_lld.h>
```

7.8 GPTConfig Struct Reference

7.8.1 Detailed Description

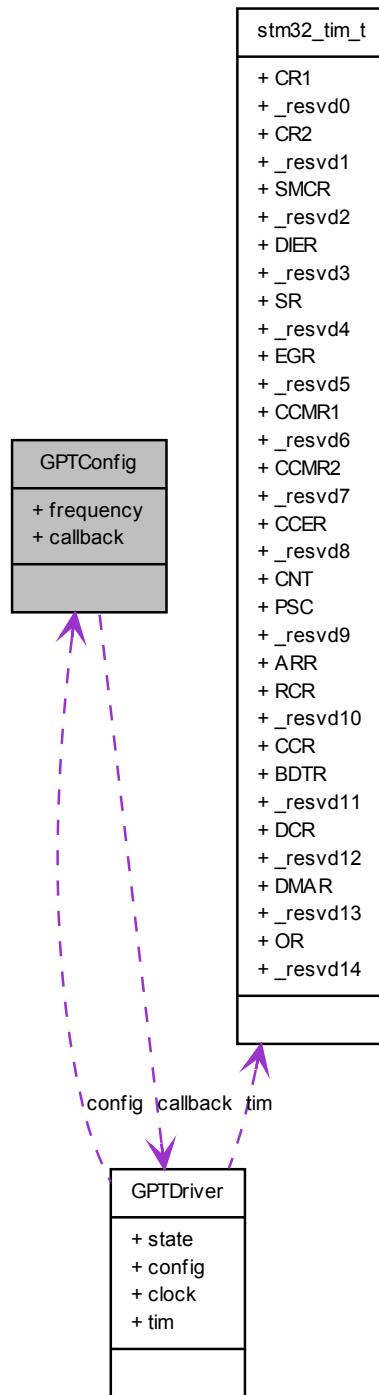
Driver configuration structure.

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.

7.8.2 Field Documentation

7.8.2.1 `gptfreq_t GPTConfig::frequency`

Timer clock in Hz.

Note

The low level can use assertions in order to catch invalid frequency specifications.

7.8.2.2 `gptcallback_t GPTConfig::callback`

Timer callback pointer.

Note

This callback is invoked on GPT counter events.

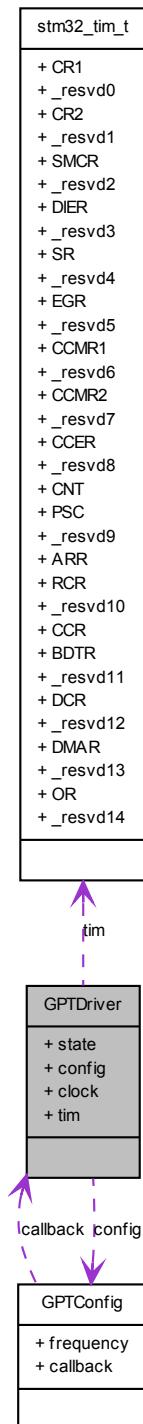
7.9 GPTDriver Struct Reference

7.9.1 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.

7.9.2 Field Documentation

7.9.2.1 [gptstate_t](#) GPTDriver::state

Driver state.

7.9.2.2 const [GPTConfig](#)* GPTDriver::config

Current configuration data.

7.9.2.3 uint32_t GPTDriver::clock

Timer base clock.

7.9.2.4 [stm32_tim_t](#)* GPTDriver::tim

Pointer to the TIMx registers block.

7.10 I2CConfig Struct Reference

7.10.1 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.

7.10.2 Field Documentation

7.10.2.1 [i2copmode_t](#) I2CConfig::op_mode

Specifies the I2C mode.

7.10.2.2 `uint32_t I2CConfig::clock_speed`

Specifies the clock frequency.

Note

Must be set to a value lower than 400kHz.

7.10.2.3 `i2cdutycycle_t I2CConfig::duty_cycle`

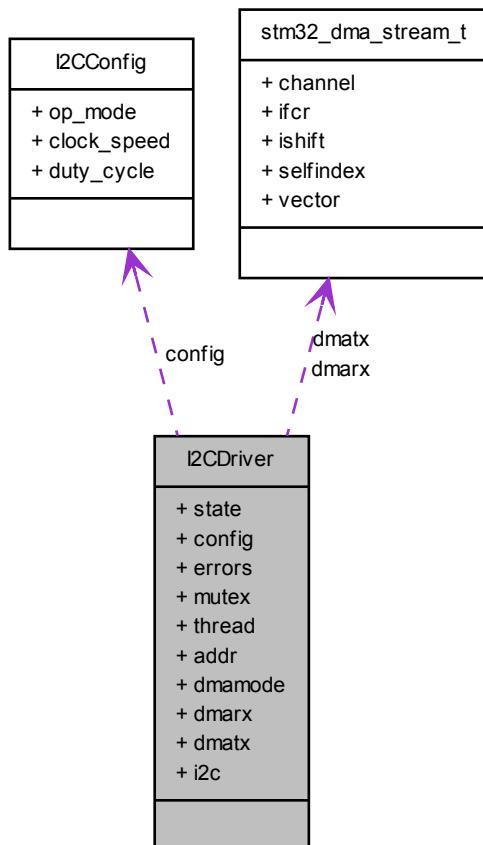
Specifies the I2C fast mode duty cycle.

7.11 I2CDriver Struct Reference**7.11.1 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.

7.11.2 Field Documentation

7.11.2.1 i2cstate_t I2CDriver::state

Driver state.

7.11.2.2 const I2CConfig* I2CDriver::config

Current configuration data.

7.11.2.3 i2cflags_t I2CDriver::errors

Error flags.

7.11.2.4 Mutex I2CDriver::mutex

Mutex protecting the bus.

7.11.2.5 Thread* I2CDriver::thread

Thread waiting for I/O completion.

7.11.2.6 i2caddr_t I2CDriver::addr

Current slave address without R/W bit.

7.11.2.7 `uint32_t I2CDriver::dmamode`

DMA mode bit mask.

7.11.2.8 `const stm32_dma_stream_t* I2CDriver::dmarx`

Receive DMA channel.

7.11.2.9 `const stm32_dma_stream_t* I2CDriver::dmatx`

Transmit DMA channel.

7.11.2.10 `I2C_TypeDef* I2CDriver::i2c`

Pointer to the I2Cx registers block.

7.12 ICUConfig Struct Reference

7.12.1 Detailed Description

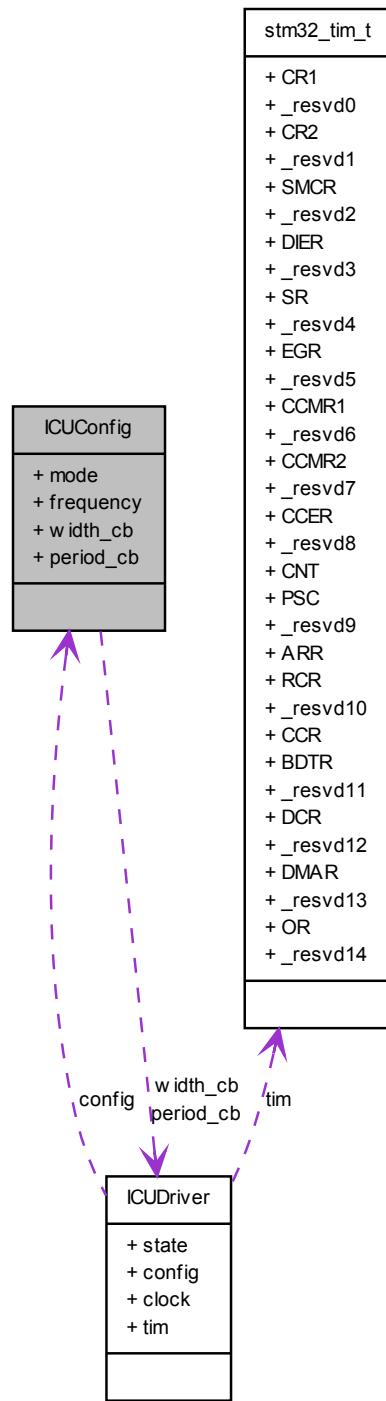
Driver configuration structure.

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.

7.12.2 Field Documentation

7.12.2.1 icumode_t ICUConfig::mode

Driver mode.

7.12.2.2 icufreq_t ICUConfig::frequency

Timer clock in Hz.

Note

The low level can use assertions in order to catch invalid frequency specifications.

7.12.2.3 icucallback_t ICUConfig::width_cb

Callback for pulse width measurement.

7.12.2.4 icucallback_t ICUConfig::period_cb

Callback for cycle period measurement.

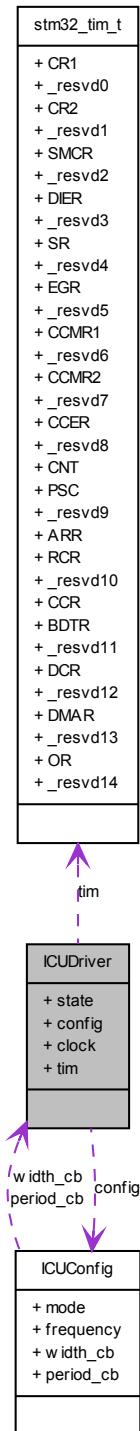
7.13 ICUDriver Struct Reference

7.13.1 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.

7.13.2 Field Documentation

7.13.2.1 [icustate_t](#) ICUDriver::state

Driver state.

7.13.2.2 const [ICUConfig](#)* ICUDriver::config

Current configuration data.

7.13.2.3 uint32_t ICUDriver::clock

Timer base clock.

7.13.2.4 [stm32_tim_t](#)* ICUDriver::tim

Pointer to the TIMx registers block.

7.14 IOBus Struct Reference

7.14.1 Detailed Description

I/O bus descriptor.

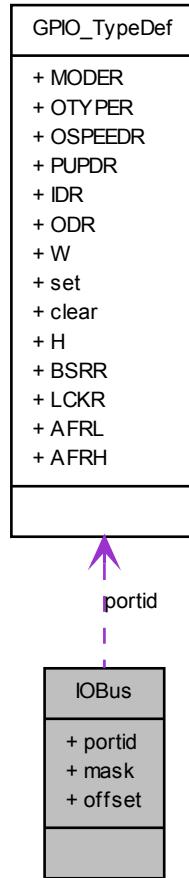
This structure describes a group of contiguous digital I/O lines that have to be handled as bus.

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>
```

Collaboration diagram for IOBus:



Data Fields

- **ioprtid_t portid**
Port identifier.
- **ioprtmask_t mask**
Bus mask aligned to port bit 0.
- **uint_fast8_t offset**
Offset, within the port, of the least significant bit of the bus.

7.14.2 Field Documentation

7.14.2.1 ioprtid_t IOBus::portid

Port identifier.

7.14.2.2 ioprtmask_t IOBus::mask

Bus mask aligned to port bit 0.

Note

The bus mask implicitly define the bus width. A logical AND is performed on the bus data.

7.14.2.3 uint_fast8_t IOBus::offset

Offset, within the port, of the least significant bit of the bus.

7.15 MMCConfig Struct Reference

7.15.1 Detailed Description

Driver configuration structure.

Note

Not required in the current implementation.

```
#include <mmc_spi.h>
```

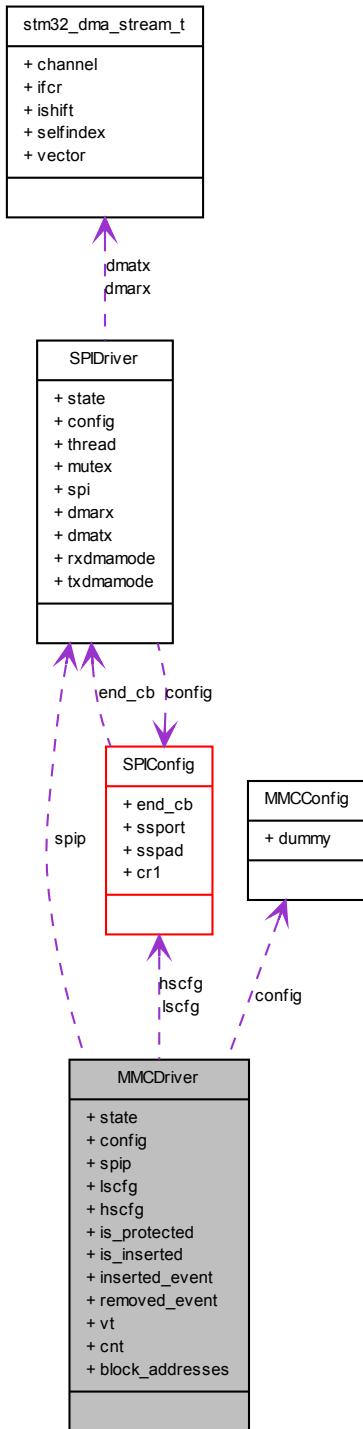
7.16 MMCDriver Struct Reference

7.16.1 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.

7.16.2 Field Documentation

7.16.2.1 mmcstate_t MMCDriver::state

Driver state.

7.16.2.2 const MMCConfig* MMCDriver::config

Current configuration data.

7.16.2.3 SPIDriver* MMCDriver::spip

SPI driver associated to this MMC driver.

7.16.2.4 const SPIConfig* MMCDriver::lscfg

SPI low speed configuration used during initialization.

7.16.2.5 const SPIConfig* MMCDriver::hscfg

SPI high speed configuration used during transfers.

7.16.2.6 mmcquery_t MMCDriver::is_protected

Write protect status query function.

7.16.2.7 mmcquery_t MMCDriver::is_inserted

Insertion status query function.

7.16.2.8 EventSource MMCDriver::inserted_event

Card insertion event source.

7.16.2.9 EventSource MMCDriver::removed_event

Card removal event source.

7.16.2.10 VirtualTimer MMCDriver::vt

MMC insertion polling timer.

7.16.2.11 uint_fast8_t MMCDriver::cnt

Insertion counter.

7.17 PALConfig Struct Reference

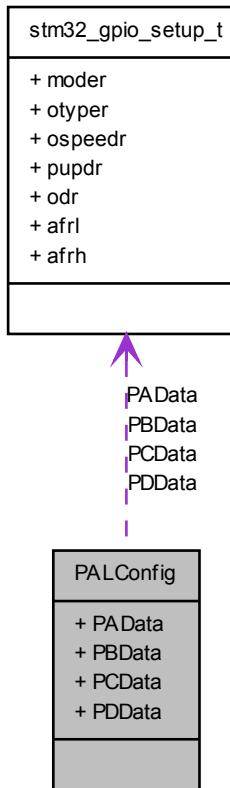
7.17.1 Detailed Description

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.

7.17.2 Field Documentation

7.17.2.1 `stm32_gpio_setup_t PALConfig::PADATA`

Port A setup data.

7.17.2.2 `stm32_gpio_setup_t PALConfig::PBData`

Port B setup data.

7.17.2.3 `stm32_gpio_setup_t` PALConfig::PCData

Port C setup data.

7.17.2.4 `stm32_gpio_setup_t` PALConfig::PDData

Port D setup data.

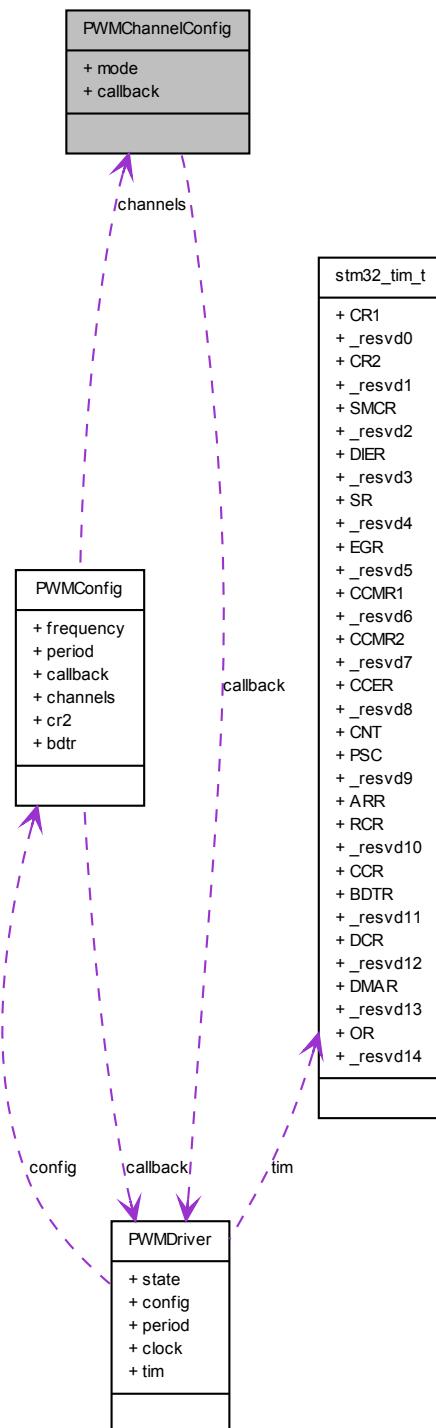
7.18 PWMChannelConfig Struct Reference

7.18.1 Detailed Description

PWM driver channel configuration structure.

```
#include <pwm_ll.h>
```

Collaboration diagram for PWMChannelConfig:



Data Fields

- `pwmmode_t mode`

Channel active logic level.

- `pwmcallback_t callback`
Channel callback pointer.

7.18.2 Field Documentation

7.18.2.1 pwmmode_t PWMChannelConfig::mode

Channel active logic level.

7.18.2.2 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.

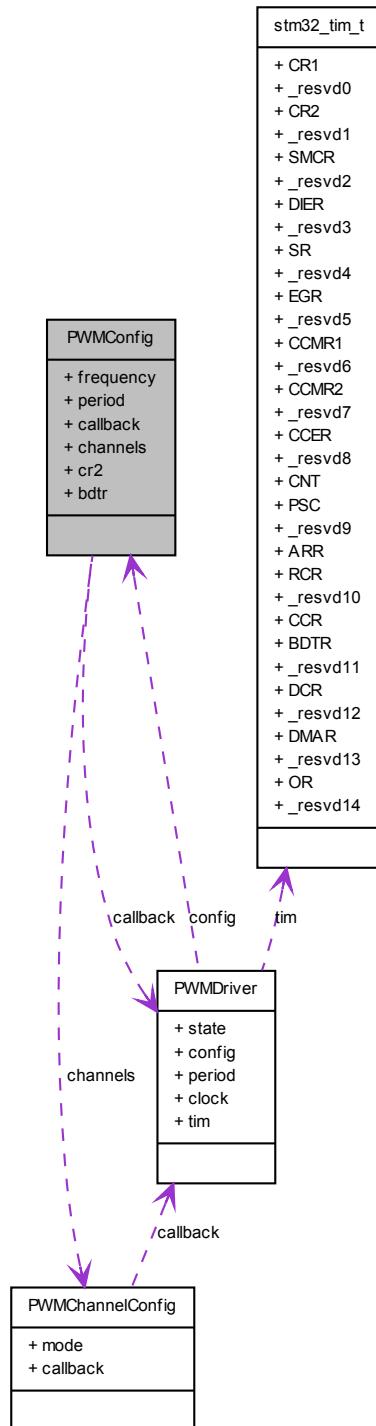
7.19 PWMConfig Struct Reference

7.19.1 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.

7.19.2 Field Documentation

7.19.2.1 uint32_t PWMConfig::frequency

Timer clock in Hz.

Note

The low level can use assertions in order to catch invalid frequency specifications.

7.19.2.2 pwmcnt_t PWMConfig::period

PWM period in ticks.

Note

The low level can use assertions in order to catch invalid period specifications.

7.19.2.3 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.

7.19.2.4 PWMChannelConfig PWMConfig::channels[PWM_CHANNELS]

Channels configurations.

7.19.2.5 uint16_t PWMConfig::cr2

TIM CR2 register initialization data.

Note

The value of this field should normally be equal to zero.

7.19.2.6 uint16_t PWMConfig::bdtr

TIM BDTR (break & dead-time) register initialization data.

Note

The value of this field should normally be equal to zero.

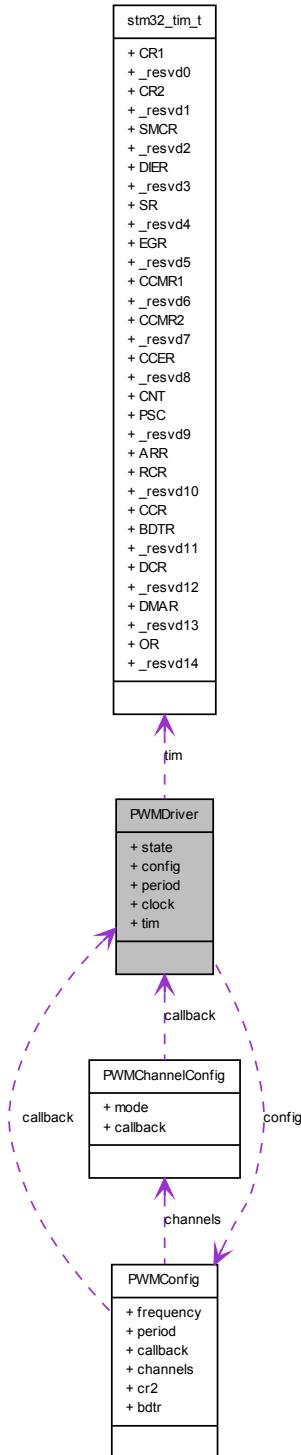
7.20 PWMDriver Struct Reference

7.20.1 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.

7.20.2 Field Documentation

7.20.2.1 [pwmstate_t](#) PWMDriver::state

Driver state.

7.20.2.2 const [PWMConfig](#)* PWMDriver::config

Current driver configuration data.

7.20.2.3 [pwmcnt_t](#) PWMDriver::period

Current PWM period in ticks.

7.20.2.4 [uint32_t](#) PWMDriver::clock

Timer base clock.

7.20.2.5 [stm32_tim_t](#)* PWMDriver::tim

Pointer to the TIMx registers block.

7.21 SerialConfig Struct Reference

7.21.1 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.

7.21.2 Field Documentation

7.21.2.1 `uint32_t SerialConfig::sc_speed`

Bit rate.

7.21.2.2 `uint16_t SerialConfig::sc_cr1`

Initialization value for the CR1 register.

7.21.2.3 `uint16_t SerialConfig::sc_cr2`

Initialization value for the CR2 register.

7.21.2.4 `uint16_t SerialConfig::sc_cr3`

Initialization value for the CR3 register.

7.22 SerialDriver Struct Reference

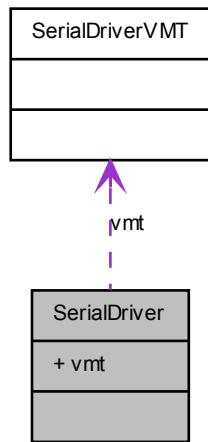
7.22.1 Detailed Description

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.

7.22.2 Field Documentation

7.22.2.1 struct SerialDriverVMT* SerialDriver::vmt

Virtual Methods Table.

7.23 SerialDriverVMT Struct Reference

7.23.1 Detailed Description

`SerialDriver` virtual methods table.

```
#include <serial.h>
```

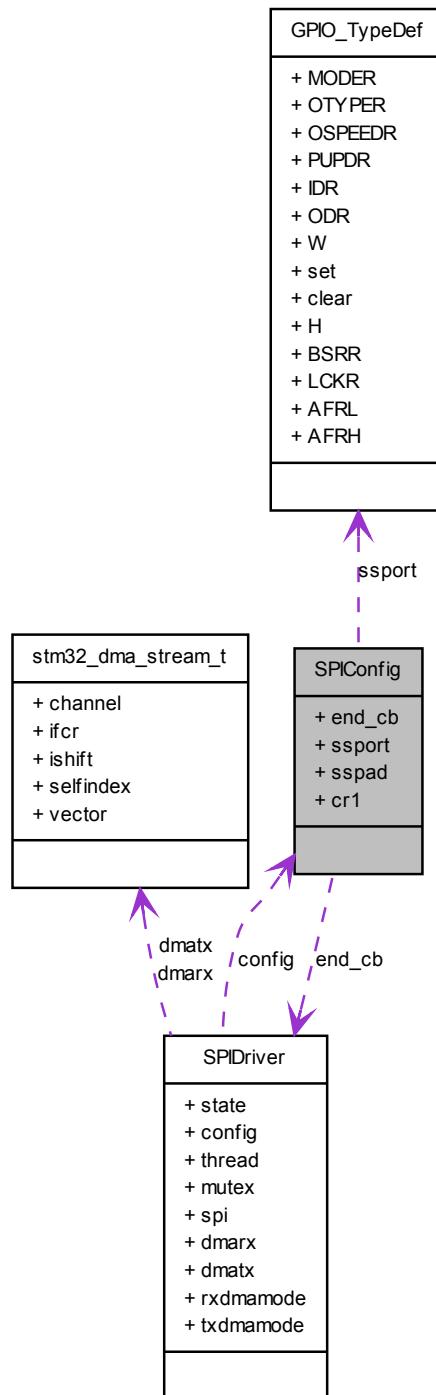
7.24 SPIConfig Struct Reference

7.24.1 Detailed Description

Driver configuration structure.

```
#include <spi_lld.h>
```

Collaboration diagram for SPIConfig:



Data Fields

- `spicallback_t end_cb`

Operation complete callback or NULL.

- **iportid_t ssport**
The chip select line port.
- **uint16_t sspad**
The chip select line pad number.
- **uint16_t cr1**
SPI initialization data.

7.24.2 Field Documentation

7.24.2.1 spicallback_t SPIConfig::end_cb

Operation complete callback or NULL.

7.24.2.2 iportid_t SPIConfig::ssport

The chip select line port.

7.24.2.3 uint16_t SPIConfig::sspad

The chip select line pad number.

7.24.2.4 uint16_t SPIConfig::cr1

SPI initialization data.

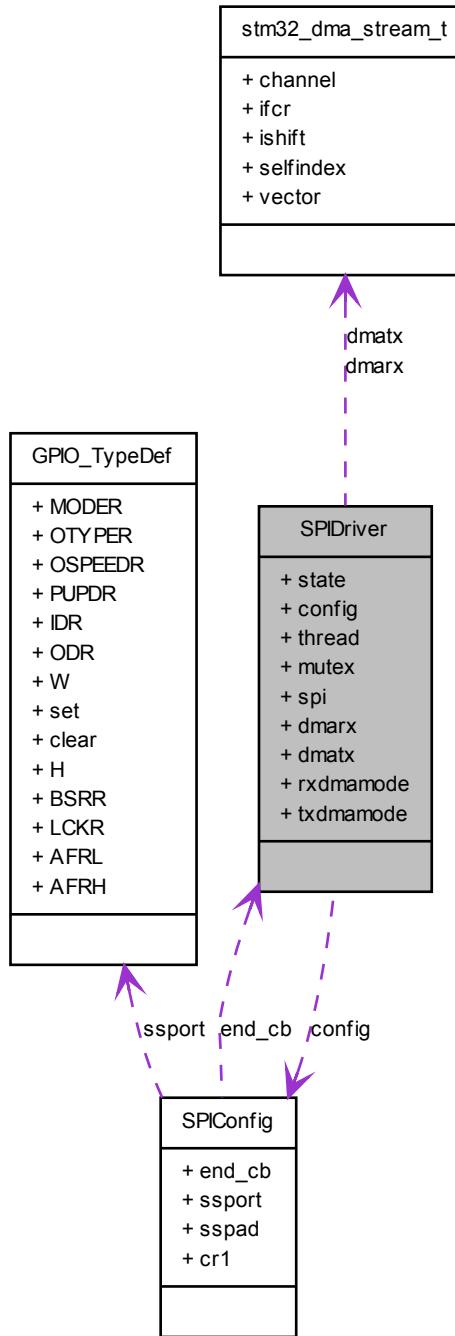
7.25 SPIDriver Struct Reference

7.25.1 Detailed Description

Structure representing a SPI driver.

```
#include <spi_llld.h>
```

Collaboration diagram for SPIDriver:



Data Fields

- **spistate_t state**
Driver state.
- const **SPICConfig * config**

- **Thread * thread**
Waiting thread.
- **Mutex mutex**
Mutex protecting the bus.
- **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.

7.25.2 Field Documentation

7.25.2.1 spistate_t SPIDriver::state

Driver state.

7.25.2.2 const SPIConfig* SPIDriver::config

Current configuration data.

7.25.2.3 Thread* SPIDriver::thread

Waiting thread.

7.25.2.4 Mutex SPIDriver::mutex

Mutex protecting the bus.

7.25.2.5 SPI_TypeDef* SPIDriver::spi

Pointer to the SPIx registers block.

7.25.2.6 const stm32_dma_stream_t* SPIDriver::dmarx

Receive DMA stream.

7.25.2.7 const stm32_dma_stream_t* SPIDriver::dmatx

Transmit DMA stream.

7.25.2.8 uint32_t SPIDriver::rxdmamode

RX DMA mode bit mask.

7.25.2.9 uint32_t SPIDriver::txdmamemode

TX DMA mode bit mask.

7.26 stm32_dma_stream_t Struct Reference

7.26.1 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.

7.26.2 Field Documentation

7.26.2.1 DMA_Channel_TypeDef* **stm32_dma_stream_t::channel**

Associated DMA channel.

7.26.2.2 volatile uint32_t* **stm32_dma_stream_t::ifcr**

Associated IFCR reg.

7.26.2.3 uint8_t **stm32_dma_stream_t::ishift**

Bits offset in xIFCR register.

7.26.2.4 uint8_t **stm32_dma_stream_t::selfindex**

Index to self in array.

7.26.2.5 uint8_t **stm32_dma_stream_t::vector**

Associated IRQ vector.

7.27 `stm32_gpio_setup_t` Struct Reference

7.27.1 Detailed Description

GPIO port setup info.

```
#include <pal_lld.h>
```

Data Fields

- `uint32_t moder`
- `uint32_t otyper`
- `uint32_t ospeedr`
- `uint32_t pupdr`
- `uint32_t odr`
- `uint32_t afrl`
- `uint32_t afrh`

7.27.2 Field Documentation

7.27.2.1 `uint32_t stm32_gpio_setup_t::moder`

Initial value for MODER register.

7.27.2.2 `uint32_t stm32_gpio_setup_t::otyper`

Initial value for OTYPER register.

7.27.2.3 `uint32_t stm32_gpio_setup_t::ospeedr`

Initial value for OSPEEDR register.

7.27.2.4 `uint32_t stm32_gpio_setup_t::pupdr`

Initial value for PUPDR register.

7.27.2.5 `uint32_t stm32_gpio_setup_t::odr`

Initial value for ODR register.

7.27.2.6 `uint32_t stm32_gpio_setup_t::afrl`

Initial value for AFRL register.

7.27.2.7 `uint32_t stm32_gpio_setup_t::afrh`

Initial value for AFRH register.

7.28 `stm32_tim_t` Struct Reference

7.28.1 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>
```

7.29 `stm32_usb_descriptor_t` Struct Reference

7.29.1 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.

7.29.2 Field Documentation

7.29.2.1 `volatile uint32_t stm32_usb_descriptor_t::TXADDR0`

TX buffer offset register.

7.29.2.2 `volatile uint16_t stm32_usb_descriptor_t::TXCOUNT0`

TX counter register 0.

7.29.2.3 `volatile uint16_t stm32_usb_descriptor_t::TXCOUNT1`

TX counter register 1.

7.29.2.4 volatile uint32_t `stm32_usb_descriptor_t`::`RXADDR0`

RX buffer offset register.

7.29.2.5 volatile uint16_t `stm32_usb_descriptor_t`::`RXCOUNT0`

RX counter register 0.

7.29.2.6 volatile uint16_t `stm32_usb_descriptor_t`::`RXCOUNT1`

RX counter register 1.

7.30 `stm32_usb_t` Struct Reference

7.30.1 Detailed Description

USB registers block.

```
#include <stm32_usb.h>
```

Data Fields

- volatile uint32_t `EPR` [USB_ENDPOINTS_NUMBER+1]
Endpoint registers.

7.30.2 Field Documentation

7.30.2.1 volatile uint32_t `stm32_usb_t`::`EPR[USB_ENDPOINTS_NUMBER+1]`

Endpoint registers.

7.31 TimeMeasurement Struct Reference

7.31.1 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.

7.31.2 Field Documentation

7.31.2.1 `void(* TimeMeasurement::start)(TimeMeasurement *tmp)`

Starts a measurement.

7.31.2.2 `void(* TimeMeasurement::stop)(TimeMeasurement *tmp)`

Stops a measurement.

7.31.2.3 `halrtcnt_t TimeMeasurement::last`

Last measurement.

7.31.2.4 `halrtcnt_t TimeMeasurement::worst`

Worst measurement.

7.31.2.5 `halrtcnt_t TimeMeasurement::best`

Best measurement.

7.32 UARTConfig Struct Reference

7.32.1 Detailed Description

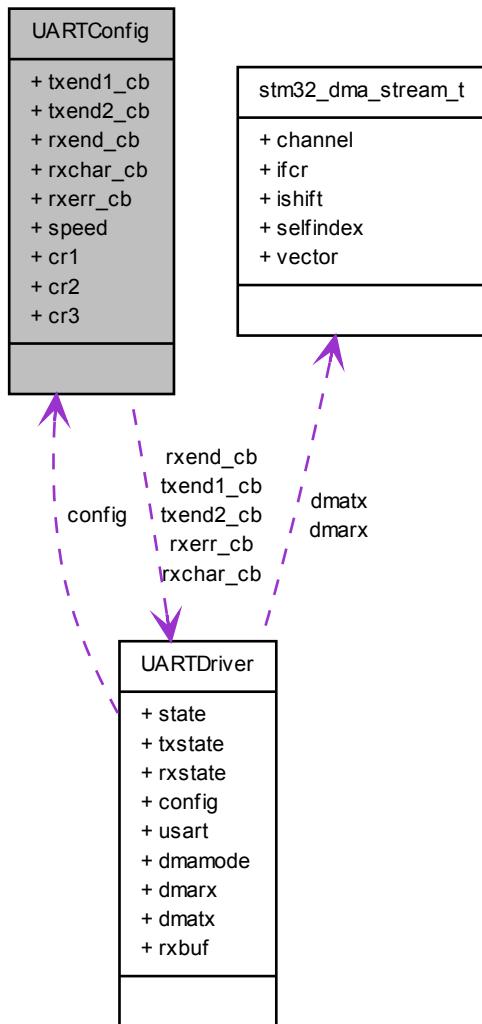
Driver configuration structure.

Note

It could be empty on some architectures.

```
#include <uart_llld.h>
```

Collaboration diagram for UARTConfig:



Data Fields

- **uartcb_t txend1_cb**
End of transmission buffer callback.
- **uartcb_t txend2_cb**
Physical end of transmission callback.
- **uartcb_t rxend_cb**
Receive buffer filled callback.
- **uartccb_t rxchar_cb**
Character received while out if the `UART_RECEIVE` state.
- **uartecb_t rxerr_cb**
Receive error callback.
- **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.

7.32.2 Field Documentation

7.32.2.1 `uartcb_t` `UARTConfig::txend1_cb`

End of transmission buffer callback.

7.32.2.2 `uartcb_t` `UARTConfig::txend2_cb`

Physical end of transmission callback.

7.32.2.3 `uartcb_t` `UARTConfig::rxend_cb`

Receive buffer filled callback.

7.32.2.4 `uartccb_t` `UARTConfig::rxchar_cb`

Character received while out if the `UART_RECEIVE` state.

7.32.2.5 `uartccb_t` `UARTConfig::rxerr_cb`

Receive error callback.

7.32.2.6 `uint32_t` `UARTConfig::speed`

Bit rate.

7.32.2.7 `uint16_t` `UARTConfig::cr1`

Initialization value for the CR1 register.

7.32.2.8 `uint16_t` `UARTConfig::cr2`

Initialization value for the CR2 register.

7.32.2.9 `uint16_t` `UARTConfig::cr3`

Initialization value for the CR3 register.

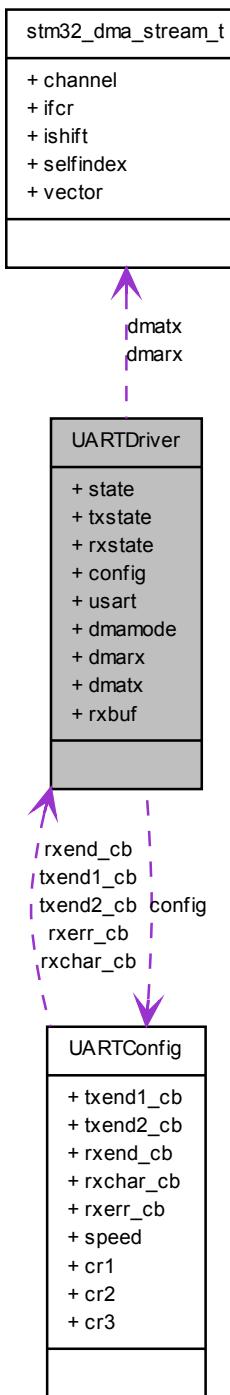
7.33 UARTDriver Struct Reference

7.33.1 Detailed Description

Structure representing an UART driver.

```
#include <uart_lld.h>
```

Collaboration diagram for UARTDriver:



Data Fields

- `uartstate_t state`

Driver state.

- `uarttxstate_t txstate`
Transmitter state.
- `uartrxstate_t rxstate`
Receiver state.
- `const UARTConfig * config`
Current configuration data.
- `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.

7.33.2 Field Documentation

7.33.2.1 `uartstate_t` `UARTDriver::state`

Driver state.

7.33.2.2 `uarttxstate_t` `UARTDriver::txstate`

Transmitter state.

7.33.2.3 `uartrxstate_t` `UARTDriver::rxstate`

Receiver state.

7.33.2.4 `const UARTConfig*` `UARTDriver::config`

Current configuration data.

7.33.2.5 `USART_TypeDef*` `UARTDriver::usart`

Pointer to the USART registers block.

7.33.2.6 `uint32_t` `UARTDriver::dmamode`

DMA mode bit mask.

7.33.2.7 `const stm32_dma_stream_t*` `UARTDriver::dmarx`

Receive DMA channel.

7.33.2.8 `const stm32_dma_stream_t*` `UARTDriver::dmatx`

Transmit DMA channel.

7.33.2.9 volatile uint16_t UARTDriver::rdbuf

Default receive buffer while into `UART_RX_IDLE` state.

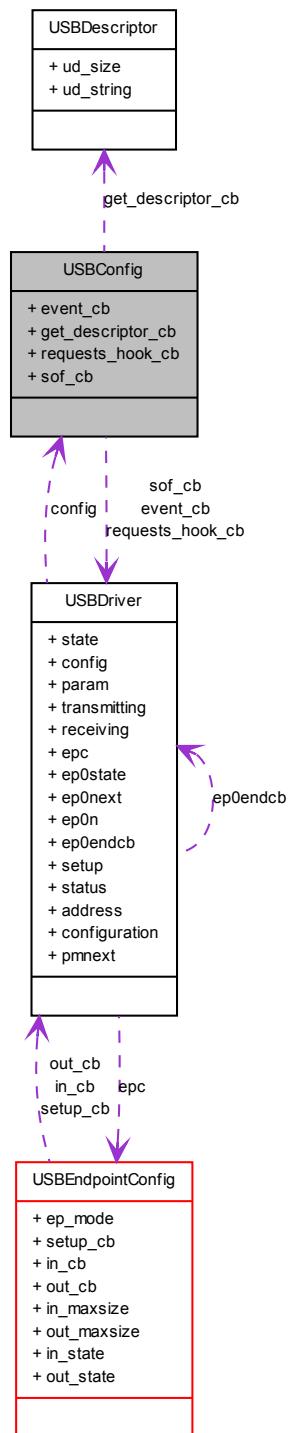
7.34 USBConfig Struct Reference

7.34.1 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.

7.34.2 Field Documentation

7.34.2.1 `usbeventcb_t USBConfig::event_cb`

USB events callback.

This callback is invoked when an USB driver event is registered.

7.34.2.2 `usbgetdescriptor_t USBConfig::get_descriptor_cb`

Device GET_DESCRIPTOR request callback.

Note

This callback is mandatory and cannot be set to NULL.

7.34.2.3 `usbreqhandler_t USBConfig::requests_hook_cb`

Requests hook callback.

This hook allows to be notified of standard requests or to handle non standard requests.

7.34.2.4 `usbcallback_t USBConfig::sof_cb`

Start Of Frame callback.

7.35 USBDescriptor Struct Reference

7.35.1 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.

7.35.2 Field Documentation

7.35.2.1 `size_t USBDescriptor::ud_size`

Descriptor size in unicode characters.

7.35.2.2 const uint8_t* USBDescriptor::ud_string

Pointer to the descriptor.

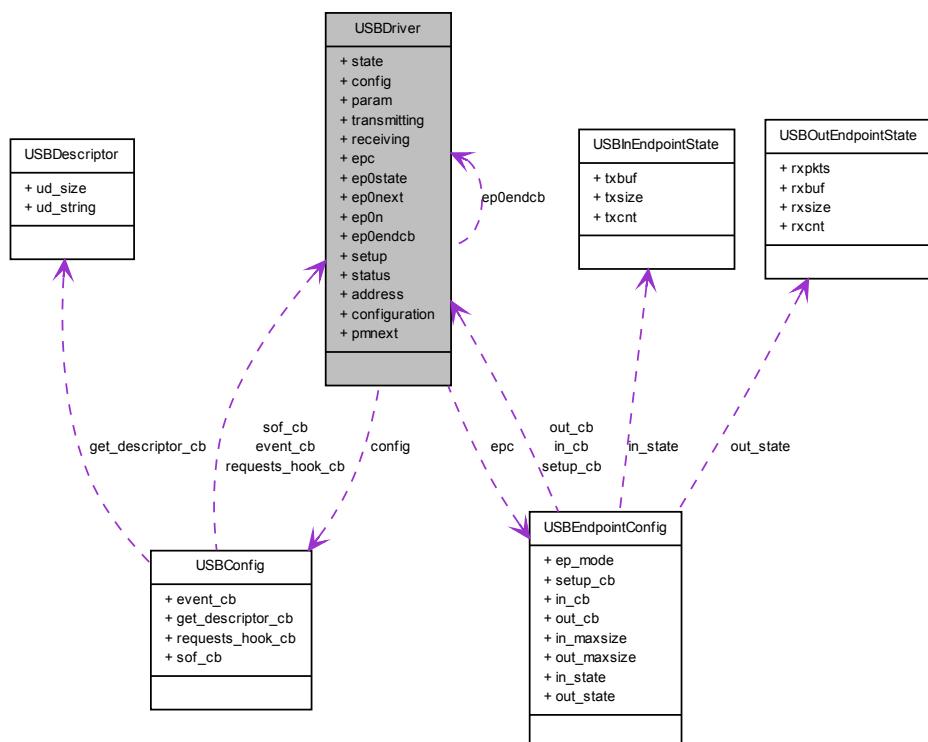
7.36 USBDriver Struct Reference

7.36.1 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.

7.36.2 Field Documentation

7.36.2.1 [usbstate_t](#) [USBDriver::state](#)

Driver state.

7.36.2.2 const [USBConfig](#)* [USBDriver::config](#)

Current configuration data.

7.36.2.3 void* [USBDriver::param](#)

Field available to user, it can be used to associate an application-defined handler to the USB driver.

7.36.2.4 uint16_t [USBDriver::transmitting](#)

Bit map of the transmitting IN endpoints.

7.36.2.5 uint16_t [USBDriver::receiving](#)

Bit map of the receiving OUT endpoints.

7.36.2.6 const [USBEndpointConfig](#)* [USBDriver::epc](#)[USB_MAX_ENDPOINTS+1]

Active endpoints configurations.

7.36.2.7 [usbep0state_t](#) [USBDriver::ep0state](#)

Endpoint 0 state.

7.36.2.8 uint8_t* USBDriver::ep0next

Next position in the buffer to be transferred through endpoint 0.

7.36.2.9 size_t USBDriver::ep0n

Number of bytes yet to be transferred through endpoint 0.

7.36.2.10 usbcallback_t USBDriver::ep0endcb

Endpoint 0 end transaction callback.

7.36.2.11 uint8_t USBDriver::setup[8]

Setup packet buffer.

7.36.2.12 uint16_t USBDriver::status

Current USB device status.

7.36.2.13 uint8_t USBDriver::address

Assigned USB address.

7.36.2.14 uint8_t USBDriver::configuration

Current USB device configuration.

7.36.2.15 uint32_t USBDriver::pmnnext

Pointer to the next address in the packet memory.

7.37 USBEndpointConfig Struct Reference

7.37.1 Detailed Description

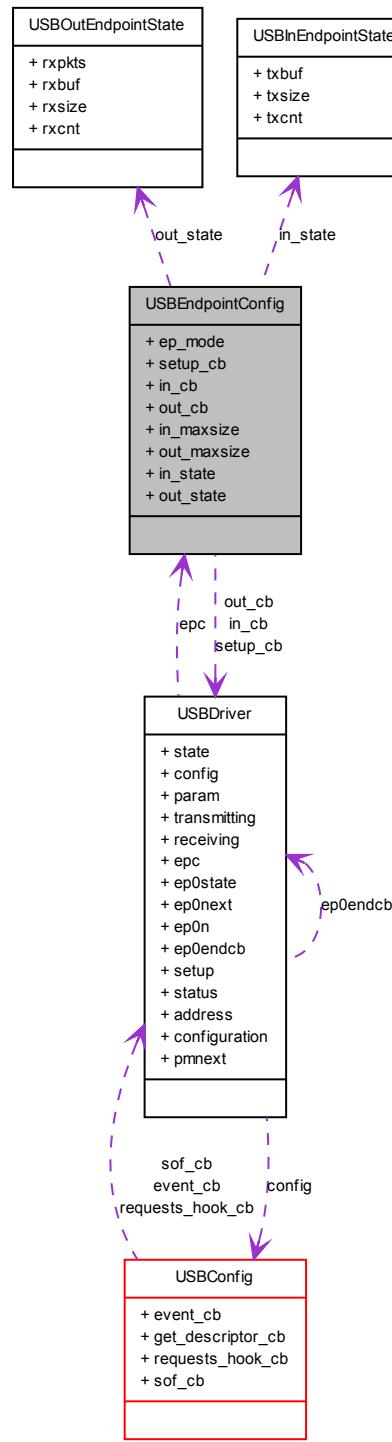
Type of an USB endpoint configuration structure.

Note

Platform specific restrictions may apply to endpoints.

```
#include <usb_lld.h>
```

Collaboration diagram for USBEndpointConfig:



Data Fields

- `uint32_t ep_mode`

Type and mode of the endpoint.

- `usbepcallback_t setup_cb`
Setup packet notification callback.
- `usbepcallback_t in_cb`
IN endpoint notification callback.
- `usbepcallback_t out_cb`
OUT endpoint notification callback.
- `uint16_t in_maxsize`
IN endpoint maximum packet size.
- `uint16_t out_maxsize`
OUT endpoint maximum packet size.
- `USBInEndpointState * in_state`
USBInEndpointState associated to the IN endpoint.
- `USBOutEndpointState * out_state`
USBOutEndpointState associated to the OUT endpoint.

7.37.2 Field Documentation

7.37.2.1 `uint32_t USBEndpointConfig::ep_mode`

Type and mode of the endpoint.

7.37.2.2 `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.

7.37.2.3 `usbepcallback_t USBEndpointConfig::in_cb`

IN endpoint notification callback.

This field must be set to `NULL` if the IN endpoint is not used.

7.37.2.4 `usbepcallback_t USBEndpointConfig::out_cb`

OUT endpoint notification callback.

This field must be set to `NULL` if the OUT endpoint is not used.

7.37.2.5 `uint16_t USBEndpointConfig::in_maxsize`

IN endpoint maximum packet size.

This field must be set to zero if the IN endpoint is not used.

7.37.2.6 `uint16_t USBEndpointConfig::out_maxsize`

OUT endpoint maximum packet size.

This field must be set to zero if the OUT endpoint is not used.

7.37.2.7 `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.

7.37.2.8 `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.

7.38 USBInEndpointState Struct Reference

7.38.1 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.

7.38.2 Field Documentation

7.38.2.1 `const uint8_t* USBInEndpointState::txbuf`

Pointer to the transmission buffer.

7.38.2.2 `size_t USBInEndpointState::txsize`

Requested transmit transfer size.

7.38.2.3 `size_t USBInEndpointState::txcnt`

Transmitted bytes so far.

7.39 USBOutEndpointState Struct Reference

7.39.1 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.

7.39.2 Field Documentation

7.39.2.1 `uint16_t USBOutEndpointState::rxpkts`

Number of packets to receive.

7.39.2.2 `uint8_t* USBOutEndpointState::rxbuf`

Pointer to the receive buffer.

7.39.2.3 `size_t USBOutEndpointState::rxsize`

Requested receive transfer size.

7.39.2.4 `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 `adcStartConversionl` (`ADCDriver` *adcp, const `ADCConversionGroup` *grpp, `adcsample_t` *samples, `size_t` depth)
Starts an ADC conversion.
- void `adcStopConversion` (`ADCDriver` *adcp)
Stops an ongoing conversion.
- void `adcStopConversionl` (`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

STM32L1xx ADC subsystem low level driver source.

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

Functions

- `CH_IRQ_HANDLER (ADC1_IRQHandler)`
ADC interrupt handler.
- `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.
- `void adcSTM32EnableTSVREFE (void)`
Enables the TSVREFE bit.
- `void adcSTM32DisableTSVREFE (void)`
Disables the TSVREFE bit.

Variables

- `ADCDriver ADCD1`
ADC1 driver identifier.

8.4 adc_lld.h File Reference

8.4.1 Detailed Description

STM32L1xx ADC subsystem low level driver header.

Data Structures

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

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.
- void **adcSTM32EnableTSVREFE** (void)
Enables the TSVREFE bit.
- void **adcSTM32DisableTSVREFE** (void)
Disables the TSVREFE bit.

Defines

Triggers selection

- #define **ADC_CR2_EXTSEL_SRC(n)** ((n) << 24)
Trigger source.

ADC clock divider settings

- #define **ADC_CCR_ADCPRE_DIV1** 0
- #define **ADC_CCR_ADCPRE_DIV2** 1
- #define **ADC_CCR_ADCPRE_DIV4** 2

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.
- #define `ADC_CHANNEL_IN18` 18
External analog input 18.
- #define `ADC_CHANNEL_IN19` 19
External analog input 19.
- #define `ADC_CHANNEL_IN20` 20
External analog input 20.
- #define `ADC_CHANNEL_IN21` 21
External analog input 21.
- #define `ADC_CHANNEL_IN22` 22
External analog input 22.
- #define `ADC_CHANNEL_IN23` 23
External analog input 23.
- #define `ADC_CHANNEL_IN24` 24
External analog input 24.
- #define `ADC_CHANNEL_IN25` 25
External analog input 25.

Sampling rates

- #define `ADC_SAMPLE_4` 0
4 cycles sampling time.
- #define `ADC_SAMPLE_9` 1
9 cycles sampling time.
- #define `ADC_SAMPLE_16` 2
16 cycles sampling time.
- #define `ADC_SAMPLE_24` 3
24 cycles sampling time.
- #define `ADC_SAMPLE_48` 4
48 cycles sampling time.
- #define `ADC_SAMPLE_96` 5
96 cycles sampling time.
- #define `ADC_SAMPLE_192` 6
192 cycles sampling time.
- #define `ADC_SAMPLE_384` 7
384 cycles sampling time.

Configuration options

- `#define STM32_ADC_USE_ADC1 TRUE`
ADC1 driver enable switch.
- `#define STM32_ADC_ADCPRE ADC_CCR_ADCPRE_DIV1`
ADC common clock divider.
- `#define STM32_ADC_ADC1_DMA_PRIORITY 2`
ADC1 DMA priority (0..3|lowest..highest).
- `#define STM32_ADC_IRQ_PRIORITY 5`
ADC interrupt priority level setting.
- `#define STM32_ADC_ADC1_DMA_IRQ_PRIORITY 5`
ADC1 DMA 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_SQR5_SQ1_N(n) ((n) << 0)`
1st channel in seq.
- `#define ADC_SQR5_SQ2_N(n) ((n) << 5)`
2nd channel in seq.
- `#define ADC_SQR5_SQ3_N(n) ((n) << 10)`
3rd channel in seq.
- `#define ADC_SQR5_SQ4_N(n) ((n) << 15)`
4th channel in seq.
- `#define ADC_SQR5_SQ5_N(n) ((n) << 20)`
5th channel in seq.
- `#define ADC_SQR5_SQ6_N(n) ((n) << 25)`
6th channel in seq.
- `#define ADC_SQR4_SQ7_N(n) ((n) << 0)`
7th channel in seq.
- `#define ADC_SQR4_SQ8_N(n) ((n) << 5)`
8th channel in seq.
- `#define ADC_SQR4_SQ9_N(n) ((n) << 10)`
9th channel in seq.
- `#define ADC_SQR4_SQ10_N(n) ((n) << 15)`
10th channel in seq.
- `#define ADC_SQR4_SQ11_N(n) ((n) << 20)`
11th channel in seq.
- `#define ADC_SQR4_SQ12_N(n) ((n) << 25)`
12th channel in seq.
- `#define ADC_SQR3_SQ13_N(n) ((n) << 0)`
13th channel in seq.
- `#define ADC_SQR3_SQ14_N(n) ((n) << 5)`
14th channel in seq.
- `#define ADC_SQR3_SQ15_N(n) ((n) << 10)`
15th channel in seq.
- `#define ADC_SQR3_SQ16_N(n) ((n) << 15)`
16th channel in seq.
- `#define ADC_SQR3_SQ17_N(n) ((n) << 20)`
17th channel in seq.
- `#define ADC_SQR3_SQ18_N(n) ((n) << 25)`
18th channel in seq.
- `#define ADC_SQR2_SQ19_N(n) ((n) << 0)`
19th channel in seq.
- `#define ADC_SQR2_SQ20_N(n) ((n) << 5)`
20th channel in seq.
- `#define ADC_SQR2_SQ21_N(n) ((n) << 10)`
21th channel in seq.
- `#define ADC_SQR2_SQ22_N(n) ((n) << 15)`

- #define ADC_SQR2_SQ23_N(n) ((n) << 20)
 22th channel in seq.
- #define ADC_SQR2_SQ24_N(n) ((n) << 25)
 23th channel in seq.
- #define ADC_SQR1_SQ25_N(n) ((n) << 0)
 24th channel in seq.
- #define ADC_SQR1_SQ26_N(n) ((n) << 5)
 25th channel in seq.
- #define ADC_SQR1_SQ27_N(n) ((n) << 10)
 26th channel in seq.
- #define ADC_SQR1_SQ28_N(n) ((n) << 6)
 27th channel in seq.

Sampling rate settings helper macros

- #define ADC_SMPR3_SMP_AN0(n) ((n) << 0)
 AN0 sampling time.
- #define ADC_SMPR3_SMP_AN1(n) ((n) << 3)
 AN1 sampling time.
- #define ADC_SMPR3_SMP_AN2(n) ((n) << 6)
 AN2 sampling time.
- #define ADC_SMPR3_SMP_AN3(n) ((n) << 9)
 AN3 sampling time.
- #define ADC_SMPR3_SMP_AN4(n) ((n) << 12)
 AN4 sampling time.
- #define ADC_SMPR3_SMP_AN5(n) ((n) << 15)
 AN5 sampling time.
- #define ADC_SMPR3_SMP_AN6(n) ((n) << 18)
 AN6 sampling time.
- #define ADC_SMPR3_SMP_AN7(n) ((n) << 21)
 AN7 sampling time.
- #define ADC_SMPR3_SMP_AN8(n) ((n) << 24)
 AN8 sampling time.
- #define ADC_SMPR3_SMP_AN9(n) ((n) << 27)
 AN9 sampling time.
- #define ADC_SMPR2_SMP_AN10(n) ((n) << 0)
 AN10 sampling time.
- #define ADC_SMPR2_SMP_AN11(n) ((n) << 3)
 AN11 sampling time.
- #define ADC_SMPR2_SMP_AN12(n) ((n) << 6)
 AN12 sampling time.
- #define ADC_SMPR2_SMP_AN13(n) ((n) << 9)
 AN13 sampling time.
- #define ADC_SMPR2_SMP_AN14(n) ((n) << 12)
 AN14 sampling time.
- #define ADC_SMPR2_SMP_AN15(n) ((n) << 15)
 AN15 sampling time.
- #define ADC_SMPR2_SMP_SENSOR(n) ((n) << 18)
 Temperature Sensor sampling time.
- #define ADC_SMPR2_SMP_VREF(n) ((n) << 21)
 Voltage Reference sampling time.
- #define ADC_SMPR2_SMP_AN18(n) ((n) << 24)
 AN18 sampling time.
- #define ADC_SMPR2_SMP_AN19(n) ((n) << 27)
 AN19 sampling time.
- #define ADC_SMPR1_SMP_AN20(n) ((n) << 0)
 AN20 sampling time.
- #define ADC_SMPR1_SMP_AN21(n) ((n) << 3)
 AN21 sampling time.
- #define ADC_SMPR1_SMP_AN22(n) ((n) << 6)

- `#define ADC_SMPR1_SMP_AN23(n) ((n) << 9)`
AN23 sampling time.
- `#define ADC_SMPR1_SMP_AN24(n) ((n) << 12)`
AN24 sampling time.
- `#define ADC_SMPR1_SMP_AN25(n) ((n) << 15)`
AN25 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, ADC_ERR_OVERFLOW = 1 }`
Possible ADC failure causes.

8.5 ext.c File Reference

8.5.1 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.

8.6 ext_lld.c File Reference

8.6.1 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_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.

8.7 ext_lld.h File Reference

8.7.1 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.

8.8 gpt.c File Reference

8.8.1 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)`

- void **gptStartContinuous** (GPTDriver *gptp, gptcnt_t interval)
Deactivates the GPT peripheral.
- 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.

8.9 gpt.h File Reference

8.9.1 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)
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.

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.

8.10 gpt_lld.c File Reference

8.10.1 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.

8.11 gpt_ll.h File Reference

8.11.1 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.

8.12 hal.c File Reference

8.12.1 Detailed Description

HAL subsystem code.

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

Functions

- `void hallInit (void)`
HAL initialization.
- `bool_t hallsCounterWithin (halrcnt_t start, halrcnt_t end)`
Realtime window test.
- `void halPolledDelay (halrcnt_t ticks)`
Polled delay.

8.13 hal.h File Reference

8.13.1 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

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.

8.14 hal_lld.c File Reference

8.14.1 Detailed Description

STM32L1xx HAL subsystem low level driver source.

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

Functions

- void **hal_lld_init** (void)
Low level HAL driver initialization.
- void **stm32_clock_init** (void)
STM32L1xx voltage, clocks and PLL initialization.

8.15 hal_lld.h File Reference

8.15.1 Detailed Description

STM32L1xx 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:

- STM32L1XX_MD for Ultra Low Power Medium-density devices.

```
#include "stm32.h"
#include "stm32_dma.h"
#include "stm32_rcc.h"
```

Functions

- void **hal_lld_init** (void)
Low level HAL driver initialization.
- void **stm32_clock_init** (void)
STM32L1xx voltage, clocks and PLL initialization.

Defines

- #define **HAL_IMPLEMENTS_COUNTERS** TRUE
Defines the support for realtime counters in the HAL.
- #define **STM32_HSECLK_MAX** 32000000
Maximum HSE clock frequency at current voltage setting.
- #define **STM32_SYSCLK_MAX** 32000000
Maximum SYSCLOCK clock frequency at current voltage setting.
- #define **STM32_PLLVCO_MAX** 96000000
Maximum VCO clock frequency at current voltage setting.
- #define **STM32_PLLVCO_MIN** 6000000
Minimum VCO clock frequency at current voltage setting.
- #define **STM32_PCLK1_MAX** 32000000
Maximum APB1 clock frequency.
- #define **STM32_PCLK2_MAX** 32000000
Maximum APB2 clock frequency.
- #define **STM32_0WS_THRESHOLD** 16000000
Maximum frequency not requiring a wait state for flash accesses.
- #define **STM32_HSI_AVAILABLE** TRUE
HSI availability at current voltage settings.
- #define **STM32_ACTIVATE_PLL** TRUE
PLL activation flag.
- #define **STM32_PLLMUL** (0 << 18)
PLLMUL field.
- #define **STM32_PLLDIV** (1 << 22)
PLLDIV field.
- #define **STM32_PLLCLKIN** STM32_HSECLK
PLL input clock frequency.
- #define **STM32_PLLVCO** (STM32_PLLCLKIN * STM32_PLLMUL_VALUE)
PLL VCO frequency.
- #define **STM32_PLLCLKOUT** (STM32_PLLVCO / STM32_PLLDIV_VALUE)
PLL output clock frequency.
- #define **STM32_MSICLK** 2100000
MSI frequency.

- `#define STM32_SYSCLK 2100000`
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 STM_MCODIVCLK 0`
MCO divider clock.
- `#define STM_MCOCLK STM_MCODIVCLK`
MCO output pin clock.
- `#define STM32_HSEDIVCLK (STM32_HSECLK / 2)`
HSE divider toward RTC clock.
- `#define STM_RTCCLK 0`
RTC/LCD clock.
- `#define STM32_ADCCLK STM32_HSICLK`
ADC frequency.
- `#define STM32_USBCLK (STM32_PLLVCO / 2)`
USB frequency.
- `#define STM32_TIMCLK1 (STM32_PCLK1 * 1)`
Timers 2, 3, 4, 6, 7 clock.
- `#define STM32_TIMCLK2 (STM32_PCLK2 * 1)`
Timers 9, 10, 11 clock.
- `#define STM32_FLASHBITS1 0x00000000`
Flash settings.
- `#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.

Platform identification

- `#define PLATFORM_NAME "STM32L1 Ultra Low Power Medium Density"`

Internal clock sources

- `#define STM32_HSICLK 16000000`
- `#define STM32_LSICLK 38000`

PWR_CR register bits definitions

- `#define STM32_VOS_MASK (3 << 11)`
- `#define STM32_VOS_1P8 (1 << 11)`
- `#define STM32_VOS_1P5 (2 << 11)`
- `#define STM32_VOS_1P2 (3 << 11)`
- `#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)`

RCC_CR register bits definitions

- #define `STM32_RTCPRE_MASK` (3 << 29)
- #define `STM32_RTCPRE_DIV2` (0 << 29)
- #define `STM32_RTCPRE_DIV4` (1 << 29)
- #define `STM32_RTCPRE_DIV8` (2 << 29)
- #define `STM32_RTCPRE_DIV16` (3 << 29)

RCC_CFGR register bits definitions

- #define `STM32_SW_MSI` (0 << 0)
- #define `STM32_SW_HSI` (1 << 0)
- #define `STM32_SW_HSE` (2 << 0)
- #define `STM32_SW_PLL` (3 << 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_PLLSRC_HSI` (0 << 16)
- #define `STM32_PLLSRC_HSE` (1 << 16)
- #define `STM32_MCOSEL_NOCLOCK` (0 << 24)
- #define `STM32_MCOSEL_SYSCLK` (1 << 24)
- #define `STM32_MCOSEL_HSI` (2 << 24)
- #define `STM32_MCOSEL_MSI` (3 << 24)
- #define `STM32_MCOSEL_HSE` (4 << 24)
- #define `STM32_MCOSEL_PLL` (5 << 24)
- #define `STM32_MCOSEL_LSI` (6 << 24)
- #define `STM32_MCOSEL_LSE` (7 << 24)
- #define `STM32_MCOPRE_DIV1` (0 << 28)
- #define `STM32_MCOPRE_DIV2` (1 << 28)
- #define `STM32_MCOPRE_DIV4` (2 << 28)
- #define `STM32_MCOPRE_DIV8` (3 << 28)
- #define `STM32_MCOPRE_DIV16` (4 << 28)

RCC_ICSCR register bits definitions

- #define `STM32_MSIRANGE_MASK` (7 << 13)
- #define `STM32_MSIRANGE_64K` (0 << 13)
- #define `STM32_MSIRANGE_128K` (1 << 13)
- #define `STM32_MSIRANGE_256K` (2 << 13)
- #define `STM32_MSIRANGE_512K` (3 << 13)
- #define `STM32_MSIRANGE_1M` (4 << 13)
- #define `STM32_MSIRANGE_2M` (5 << 13)
- #define `STM32_MSIRANGE_4M` (6 << 13)

RCC_CSR register bits definitions

- #define `STM32_RTCSEL_MASK` (3 << 16)
- #define `STM32_RTCSEL_NOCLOCK` (0 << 16)
- #define `STM32_RTCSEL_LSE` (1 << 16)
- #define `STM32_RTCSEL_LSI` (2 << 16)
- #define `STM32_RTCSEL_HSEDIV` (3 << 16)

STM32L1xx 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_CAN_MAX_FILTERS** 0
- #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** 23
- #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** TRUE
- #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_RTC_HAS_SUBSECONDS** FALSE
- #define **STM32_RTC_IS_CALENDAR** 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** FALSE
- #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** TRUE
- #define **STM32_HAS_TIM10** TRUE
- #define **STM32_HAS_TIM11** TRUE
- #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

IRQ VECTOR names

- #define **WWDG_IRQHandler** Vector40
- #define **PVD_IRQHandler** Vector44
- #define **TAMPER_STAMP_IRQHandler** Vector48
- #define **RTC_WKUP_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_IRQHandler** Vector88
- #define **USB_HP_IRQHandler** Vector8C

- #define [USB_LP_IRQHandler](#) Vector90
- #define [DAC_IRQHandler](#) Vector94
- #define [COMP_IRQHandler](#) Vector98
- #define [EXTI9_5_IRQHandler](#) Vector9C
- #define [TIM9_IRQHandler](#) VectorA0
- #define [TIM10_IRQHandler](#) VectorA4
- #define [TIM11_IRQHandler](#) VectorA8
- #define [LCD_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 [TIM6_IRQHandler](#) VectorEC
- #define [TIM7_IRQHandler](#) VectorF0

Configuration options

- #define [STM32_NO_INIT](#) FALSE
Disables the PWR/RCC initialization in the HAL.
- #define [STM32_VOS](#) STM32_VOS_1P8
Core voltage selection.
- #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](#) TRUE
Enables or disables the LSI clock source.
- #define [STM32_HSE_ENABLED](#) FALSE
Enables or disables the HSE clock source.
- #define [STM32_LSE_ENABLED](#) FALSE
Enables or disables the LSE clock source.
- #define [STM32_ADC_CLOCK_ENABLED](#) TRUE
ADC clock setting.
- #define [STM32_USB_CLOCK_ENABLED](#) TRUE
USB clock setting.
- #define [STM32_MSIRANGE](#) STM32_MSIRANGE_2M
MSI frequency setting.
- #define [STM32_SW](#) STM32_SW_PLL
Main clock source selection.
- #define [STM32_PLLSRC](#) STM32_PLLSRC_HSI
Clock source for the PLL.
- #define [STM32_PLLMUL_VALUE](#) 6
PLL multiplier value.
- #define [STM32_PLLDIV_VALUE](#) 3
PLL divider value.
- #define [STM32_HPRE](#) STM32_HPRE_DIV1
AHB prescaler value.
- #define [STM32_PPREG1](#) STM32_PPREG1_DIV1

- `#define STM32_PPREG STM32_PPREG_DIV1`
APB1 prescaler value.
- `#define STM32_PPREF STM32_PPREF_DIV1`
APB2 prescaler value.
- `#define STM32_MCOSEL STM32_MCOSEL_NOCLOCK`
MCO clock source.
- `#define STM32_MCOPRE STM32_MCOPRE_DIV1`
MCO divider setting.
- `#define STM32_RTCSEL STM32_RTCSEL_LSE`
RTC/LCD clock source.
- `#define STM32_RTCPRE STM32_RTCPRE_DIV2`
HSE divider toward RTC setting.

Typedefs

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

8.16 halconf.h File Reference

8.16.1 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`
- `#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_MMCSPI TRUE`
Enables the MMCSPI 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_MMCSUPPORT** 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.

8.17 i2c.c File Reference

8.17.1 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` *`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.

8.18 i2c.h File Reference

8.18.1 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 *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.

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 `i2cMasterTransmitTimeout` function with `TIME_INFINITE` timeout.
- `#define i2cMasterReceive(i2cp, addr, rxbuf, rxbytes) (i2cMasterReceiveTimeout(i2cp, addr, rxbuf, rxbytes, TIME_INFINITE))`
Wrap `i2cMasterReceiveTimeout` function with `TIME_INFINITE` timeout.

I2C bus error conditions

- `#define 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.

8.19 i2c_lld.c File Reference

8.19.1 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)

Deactivates the I2C peripheral.
- msg_t **i2c_lld_master_receive_timeout** (I2CDriver *i2cp, i2caddr_t addr, uint8_t *rdbuf, size_t rxbytes, systime_t timeout)

Receives data via the I2C bus as master.
- msg_t **i2c_lld_master_transmit_timeout** (I2CDriver *i2cp, i2caddr_t addr, const uint8_t *txbuf, size_t txbytes, uint8_t *rdbuf, size_t rxbytes, systime_t timeout)

Transmits data via the I2C bus as master.

Variables

- I2CDriver **I2CD1**

I2C1 driver identifier.
- I2CDriver **I2CD2**

I2C2 driver identifier.
- I2CDriver **I2CD3**

I2C3 driver identifier.

Defines

- #define **wakeup_isr**(i2cp, msg)

Wakes up the waiting thread.

8.20 i2c_ll.h File Reference

8.20.1 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` *`txbuf`, `size_t` `txbytes`, `uint8_t` *`rxbuf`, `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` *`rxbuf`, `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.

8.21 icu.c File Reference

8.21.1 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)`

- void **icuEnable** (ICUDriver *icup)
Deactivates the ICU peripheral.
- void **icuDisable** (ICUDriver *icup)
Enables the input capture.
- void **icuStop** (ICUDriver *icup)
Disables the input capture.

8.22 icu.h File Reference

8.22.1 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.

8.23 icu_lld.c File Reference

8.23.1 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.

8.24 icu_lld.h File Reference

8.24.1 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_ll_init** (void)
Low level ICU driver initialization.
- void **icu_ll_start** (ICUDriver *icup)
Configures and activates the ICU peripheral.
- void **icu_ll_stop** (ICUDriver *icup)
Deactivates the ICU peripheral.
- void **icu_ll_enable** (ICUDriver *icup)
Enables the input capture.
- void **icu_ll_disable** (ICUDriver *icup)
Disables the input capture.

Defines

- #define **icu_ll_get_width**(icup) ((icup)->tim->CCR[1] + 1)
Returns the width of the latest pulse.
- #define **icu_ll_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.

8.25 mmc_spi.c File Reference

8.25.1 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)`
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.

8.26 mmc_spi.h File Reference

8.26.1 Detailed Description

MMC over SPI driver header.

Data Structures

- struct [MMCConfig](#)
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 [MMCConfig](#) *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 mmcIsWriteProtected(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.

8.27 pal.c File Reference

8.27.1 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.

8.28 pal.h File Reference

8.28.1 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`
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))`

- `#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.

8.29 pal_lld.c File Reference

8.29.1 Detailed Description

STM32L1xx/STM32F2xx/STM32F4xx 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.

8.30 pal_lld.h File Reference

8.30.1 Detailed Description

STM32L1xx/STM32F2xx/STM32F4xx GPIO low level driver header.

Data Structures

- struct `GPIO_TypeDef`
STM32 GPIO registers block.
- 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 IOPORT8 GPIOH`
GPIO port H identifier.
- `#define IOPORT9 GPIOI`
GPIO port I 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_readlatch(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.H.set = (uint16_t)(bits))`
Sets a bits mask on a I/O port.
- `#define pal_lld_clearport(port, bits) ((port)->BSRR.H.clear = (uint16_t)(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.

STM32-specific I/O mode flags

- #define **PAL_STM32_MODE_MASK** (3 << 0)
- #define **PAL_STM32_MODE_INPUT** (0 << 0)
- #define **PAL_STM32_MODE_OUTPUT** (1 << 0)
- #define **PAL_STM32_MODE_ALTERNATE** (2 << 0)
- #define **PAL_STM32_MODE_ANALOG** (3 << 0)
- #define **PAL_STM32_OTYPE_MASK** (1 << 2)
- #define **PAL_STM32_OTYPE_PUSH_PULL** (0 << 2)
- #define **PAL_STM32_OTYPE_OPENDRAIN** (1 << 2)
- #define **PAL_STM32_OSPEED_MASK** (3 << 3)
- #define **PAL_STM32_OSPEED_LOWEST** (0 << 3)
- #define **PAL_STM32_OSPEED_MID1** (1 << 3)
- #define **PAL_STM32_OSPEED_MID2** (2 << 3)
- #define **PAL_STM32_OSPEED_HIGHEST** (3 << 3)
- #define **PAL_STM32_PUDR_MASK** (3 << 5)
- #define **PAL_STM32_PUDR_FLOATING** (0 << 5)
- #define **PAL_STM32_PUDR_PULLUP** (1 << 5)
- #define **PAL_STM32_PUDR_PULLDOWN** (2 << 5)
- #define **PAL_STM32_ALTERNATE_MASK** (15 << 7)
- #define **PAL_STM32_ALTERNATE(n)** ((n) << 7)
- #define **PAL_MODE_ALTERNATE(n)**

Alternate function.

Standard I/O mode flags

- #define **PAL_MODE_RESET** PAL_STM32_MODE_INPUT
This mode is implemented as input.
- #define **PAL_MODE_UNCONNECTED** PAL_STM32_MODE_OUTPUT
This mode is implemented as output.
- #define **PAL_MODE_INPUT** PAL_STM32_MODE_INPUT
Regular input high-Z pad.
- #define **PAL_MODE_INPUT_PULLUP**
Input pad with weak pull up resistor.
- #define **PAL_MODE_INPUT_PULLDOWN**
Input pad with weak pull down resistor.
- #define **PAL_MODE_INPUT_ANALOG** PAL_STM32_MODE_ANALOG
Analog input mode.
- #define **PAL_MODE_OUTPUT_PUSH_PULL**
Push-pull output pad.
- #define **PAL_MODE_OUTPUT_OPENDRAIN**
Open-drain 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.

8.31 pwm.c File Reference

8.31.1 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, **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.

8.32 pwm.h File Reference

8.32.1 Detailed Description

PWM Driver macros and structures. #include "pwm_lld.h"

Functions

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

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.

8.33 pwm_lld.c File Reference

8.33.1 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, pwcmt_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.

8.34 pwm_lld.h File Reference

8.34.1 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_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.

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

- `#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.

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 pwemode_t`
PWM mode type.
- `typedef uint8_t pwmchannel_t`
PWM channel type.
- `typedef uint16_t pwmcnt_t`
PWM counter type.

8.35 serial.c File Reference

8.35.1 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.

8.36 serial.h File Reference

8.36.1 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)`
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`.

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.

8.37 serial_lld.c File Reference

8.37.1 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_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.

8.38 serial_lld.h File Reference

8.38.1 Detailed Description

STM32 low level serial driver header.

Data Structures

- struct `SerialConfig`
STM32 Serial Driver configuration structure.

Functions

- void `sd_ll_init` (void)
Low level serial driver initialization.
- void `sd_ll_start` (`SerialDriver` *`sdp`, const `SerialConfig` *`config`)
Low level serial driver configuration and (re)start.
- void `sd_ll_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.

8.39 spi.c File Reference

8.39.1 Detailed Description

```
SPI Driver code. #include "ch.h"  
#include "hal.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 SPIConfig *config)
Configures and activates the SPI peripheral.
- void **spiStop** (SPIDriver *spip)
Deactivates the SPI peripheral.
- void **spiSelect** (SPIDriver *spip)
Asserts the slave select signal and prepares for transfers.
- void **spiUnselect** (SPIDriver *spip)
Deasserts the slave select signal.
- void **spiStartIgnore** (SPIDriver *spip, size_t n)
Ignores data on the SPI bus.
- void **spiStartExchange** (SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)
Exchanges data on the SPI bus.
- void **spiStartSend** (SPIDriver *spip, size_t n, const void *txbuf)
Sends data over the SPI bus.
- void **spiStartReceive** (SPIDriver *spip, size_t n, void *rxbuf)
Receives data from the SPI bus.
- void **spiIgnore** (SPIDriver *spip, size_t n)
Ignores data on the SPI bus.
- void **spiExchange** (SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)
Exchanges data on the SPI bus.
- void **spiSend** (SPIDriver *spip, size_t n, const void *txbuf)
Sends data over the SPI bus.
- void **spiReceive** (SPIDriver *spip, size_t n, void *rxbuf)
Receives data from the SPI bus.
- void **spiAcquireBus** (SPIDriver *spip)
Gains exclusive access to the SPI bus.
- void **spiReleaseBus** (SPIDriver *spip)
Releases exclusive access to the SPI bus.

8.40 spi.h File Reference

8.40.1 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 `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.

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`(spip)

Waits for operation completion.
- #define `_spi_wakeup_isr`(spip)

Wakes up the waiting thread.
- #define `_spi_isr_code`(spip)

Common ISR code.

Enumerations

- enum `spistate_t` {
 `SPI_UNINIT` = 0, `SPI_STOP` = 1, `SPI_READY` = 2, `SPI_ACTIVE` = 3,
 `SPI_COMPLETE` = 4 }
Driver state machine possible states.

8.41 spi_lld.c File Reference

8.41.1 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 *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.

8.42 spi_lld.h File Reference

8.42.1 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.

8.43 stm32.h File Reference

8.43.1 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"
```

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)

8.44 stm32_dma.c File Reference

8.44.1 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.
- 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 0x00000007F`
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 0x000000000`
Post-reset value of the stream CCR register.

8.45 stm32_dma.h File Reference

8.45.1 Detailed Description

DMA helper driver header.

Note

This file requires definitions from the ST header file `stm32l1xx.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 7`
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.

DMA streams identifiers

- `#define STM32_DMA_STREAM_ID(dma, stream) ((stream) - 1)`
Returns an unique numeric identifier for a DMA 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.
- `#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)`

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_BYT 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_MSIZE_MASK DMA_CCR1_MSIZE`
- `#define STM32_DMA_CR_MSIZE_BYT 0`
- `#define STM32_DMA_CR_MSIZE_HWWORD DMA_CCR1_MSIZE_0`
- `#define STM32_DMA_CR_MSIZE_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.

Typedefs

- `typedef void(* stm32_dmaisr_t)(void *p, uint32_t flags)`
STM32 DMA ISR function type.

8.46 stm32_rcc.h File Reference

8.46.1 Detailed Description

RCC helper driver header.

Note

This file requires definitions from the ST header file `stm3211xx.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.

DMA peripheral 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() rccResetAHB(RCC_AHBRSTR_DMA1RST)`
Resets the DMA1 peripheral.

PWR interface specific RCC operations

- `#define rccEnablePWRInterface(lp) rccEnableAPB1(RCC_APB1ENR_PWREN, lp)`
Enables the PWR interface clock.
- `#define rccDisablePWRInterface(lp) rccDisableAPB1(RCC_APB1ENR_PWREN, lp)`
Disables PWR interface clock.
- `#define rccResetPWRInterface() rccResetAPB1(RCC_APB1RSTR_PWRRST)`
Resets the PWR interface.

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.

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.

TIM peripherals specific RCC operations

- `#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.

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.

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.

8.47 stm32_usb.h File Reference

8.47.1 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_ENDPOINTS_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`

8.48 tm.c File Reference

8.48.1 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.

8.49 tm.h File Reference

8.49.1 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.

8.50 uart.c File Reference

8.50.1 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)

- `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 uartStartReceivevl (UARTDriver *uartp, size_t n, void *rxbuf)`

Starts a receive operation on the UART peripheral.
- `size_t uartStopReceive (UARTDriver *uartp)`

Stops any ongoing receive operation.
- `size_t uartStopReceivevl (UARTDriver *uartp)`

Stops any ongoing receive operation.

8.51 uart.h File Reference

8.51.1 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 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 uartStartReceivevl (UARTDriver *uartp, size_t n, void *rxbuf)`

Starts a receive operation on the UART peripheral.
- `size_t uartStopReceive (UARTDriver *uartp)`

Stops any ongoing receive operation.
- `size_t uartStopReceivevl (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.

8.52 uart_lld.c File Reference

8.52.1 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 *rxbuf)

- `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.

8.53 uart_lld.h File Reference

8.53.1 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_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.

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()`
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 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(* uarteccb_t)(UARTDriver *uartp, uartflags_t e)`
Receive error UART notification callback type.

8.54 usb.c File Reference

8.54.1 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 **usbInitEndpointl** (**USBDriver** *usbp, **usbep_t** ep, const **USBEndpointConfig** *epcp)
Enables an endpoint.
- void **usbDisableEndpointsl** (**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 **usbStartTransmitl** (**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 **usbStallTransmitl** (**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.

8.55 usb.h File Reference

8.55.1 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 `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.

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`(*bInterfaceNumber*, *bAlternateSetting*, *bNumEndpoints*, *bInterfaceClass*, *bInterfaceSubClass*, *bInterfaceProtocol*, *iInterface*)

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_lld_connect_bus(usbp)`
Connects the USB device.
- #define `usbDisconnectBus(usbp)` `usb_lld_disconnect_bus(usbp)`
Disconnects 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 `usbGetReceiveTransactionSizel(usbp, ep)` `usb_lld_get_transaction_size(usbp, ep)`
Returns the exact size of a receive transaction.
- #define `usbGetReceivePacketSizel(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(*)(USBDriver *usbp)` `usbcallback_t`

- **typedef void(* usbepcallback_t)(USBDriver *usbp, usbep_t ep)**
Type of an USB generic notification callback.
- **typedef void(* usbeventcb_t)(USBDriver *usbp, usbevent_t event)**
Type of an USB endpoint 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.

8.56 usb_lld.c File Reference

8.56.1 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)**

- void `usb_lld_reset (USBDriver *usbp)`
Deactivates the USB peripheral.
- 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.

8.56.2 Variable Documentation

8.56.2.1 USBInEndpointState in

IN EP0 state.

8.56.2.2 USBOutEndpointState out

OUT EP0 state.

8.57 usb_ll.h File Reference

8.57.1 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)

- void `usb_ll_start_in (USBDriver *usbp, usbep_t ep)`
Starts a receive operation on an OUT endpoint.
- void `usb_ll_stall_out (USBDriver *usbp, usbep_t ep)`
Starts a transmit operation on an IN endpoint.
- void `usb_ll_stall_in (USBDriver *usbp, usbep_t ep)`
Brings an OUT 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, 144

_adc_isr_error_code
 ADC Driver, 33

_adc_isr_full_code
 ADC Driver, 33

_adc_isr_half_code
 ADC Driver, 33

_adc_reset_i
 ADC Driver, 31

_adc_reset_s
 ADC Driver, 31

_adc_timeout_isr
 ADC Driver, 32

_adc_wakeup_isr
 ADC Driver, 32

_icu_isr_invoke_period_cb
 ICU Driver, 123

_icu_isr_invoke_width_cb
 ICU Driver, 122

_pal_lld_init
 PAL Driver, 142

_pal_lld_setgroupmode
 PAL Driver, 143

_serial_driver_data
 Serial Driver, 187

_serial_driver_methods
 Serial Driver, 182

_spi_isr_code
 SPI Driver, 206

_spi_wait_s
 SPI Driver, 205

_spi_wakeup_isr
 SPI Driver, 206

_stm32_dma_streams
 STM32L1xx DMA Support, 279

_usb_ep0in
 USB Driver, 245

_usb_ep0out
 USB Driver, 246

_usb_ep0setup
 USB Driver, 244

_usb_isr_invoke_event_cb
 USB Driver, 261

_usb_isr_invoke_in_cb
 USB Driver, 262

_usb_isr_invoke_out_cb
 USB Driver, 262

_usb_isr_invoke_setup_cb
 USB Driver, 262

_usb_isr_invoke_sof_cb
 USB Driver, 261

_usb_reset
 USB Driver, 244

adc
 ADCDriver, 308

ADC Driver, 16

 _adc_isr_error_code, 33

 _adc_isr_full_code, 33

 _adc_isr_half_code, 33

 _adc_reset_i, 31

 _adc_reset_s, 31

 _adc_timeout_isr, 32

 _adc_wakeup_isr, 32

 ADC_ACTIVE, 43

 ADC_COMPLETE, 43

 ADC_ERR_DMAFAILURE, 43

 ADC_ERR_OVERFLOW, 43

 ADC_ERROR, 43

 ADC_READY, 43

 ADC_STOP, 43

 ADC_UNINIT, 43

 ADC_CHANNEL_IN0, 34

 ADC_CHANNEL_IN1, 34

 ADC_CHANNEL_IN10, 35

 ADC_CHANNEL_IN11, 35

 ADC_CHANNEL_IN12, 35

 ADC_CHANNEL_IN13, 35

 ADC_CHANNEL_IN14, 35

 ADC_CHANNEL_IN15, 35

 ADC_CHANNEL_IN18, 36

 ADC_CHANNEL_IN19, 36

 ADC_CHANNEL_IN2, 34

 ADC_CHANNEL_IN20, 36

 ADC_CHANNEL_IN21, 36

 ADC_CHANNEL_IN22, 36

 ADC_CHANNEL_IN23, 36

 ADC_CHANNEL_IN24, 36

 ADC_CHANNEL_IN25, 36

 ADC_CHANNEL_IN3, 34

 ADC_CHANNEL_IN4, 34

 ADC_CHANNEL_IN5, 35

 ADC_CHANNEL_IN6, 35

 ADC_CHANNEL_IN7, 35

 ADC_CHANNEL_IN8, 35

 ADC_CHANNEL_IN9, 35

 ADC_CHANNEL_SENSOR, 35

 ADC_CHANNEL_VREFINT, 36

adc_channels_num_t, 42

ADC_CR2_EXTSEL_SRC, 34
adc_lld_init, 29
adc_lld_start, 29
adc_lld_start_conversion, 30
adc_lld_stop, 29
adc_lld_stop_conversion, 30
ADC_SAMPLE_16, 36
ADC_SAMPLE_192, 37
ADC_SAMPLE_24, 37
ADC_SAMPLE_384, 37
ADC_SAMPLE_4, 36
ADC_SAMPLE_48, 37
ADC_SAMPLE_9, 36
ADC_SAMPLE_96, 37
ADC_SMPR1_SMP_AN20, 42
ADC_SMPR1_SMP_AN21, 42
ADC_SMPR1_SMP_AN22, 42
ADC_SMPR1_SMP_AN23, 42
ADC_SMPR1_SMP_AN24, 42
ADC_SMPR1_SMP_AN25, 42
ADC_SMPR2_SMP_AN10, 41
ADC_SMPR2_SMP_AN11, 41
ADC_SMPR2_SMP_AN12, 41
ADC_SMPR2_SMP_AN13, 41
ADC_SMPR2_SMP_AN14, 41
ADC_SMPR2_SMP_AN15, 41
ADC_SMPR2_SMP_AN18, 41
ADC_SMPR2_SMP_AN19, 41
ADC_SMPR2_SMP_SENSOR, 41
ADC_SMPR2_SMP_VREF, 41
ADC_SMPR3_SMP_AN0, 40
ADC_SMPR3_SMP_AN1, 40
ADC_SMPR3_SMP_AN2, 40
ADC_SMPR3_SMP_AN3, 40
ADC_SMPR3_SMP_AN4, 40
ADC_SMPR3_SMP_AN5, 40
ADC_SMPR3_SMP_AN6, 40
ADC_SMPR3_SMP_AN7, 40
ADC_SMPR3_SMP_AN8, 41
ADC_SMPR3_SMP_AN9, 41
ADC_SQR1_NUM_CH, 38
ADC_SQR1_SQ25_N, 40
ADC_SQR1_SQ26_N, 40
ADC_SQR1_SQ27_N, 40
ADC_SQR2_SQ19_N, 39
ADC_SQR2_SQ20_N, 39
ADC_SQR2_SQ21_N, 39
ADC_SQR2_SQ22_N, 39
ADC_SQR2_SQ23_N, 39
ADC_SQR2_SQ24_N, 40
ADC_SQR3_SQ13_N, 39
ADC_SQR3_SQ14_N, 39
ADC_SQR3_SQ15_N, 39
ADC_SQR3_SQ16_N, 39
ADC_SQR3_SQ17_N, 39
ADC_SQR3_SQ18_N, 39
ADC_SQR4_SQ10_N, 38
ADC_SQR4_SQ11_N, 38
ADC_SQR4_SQ12_N, 39
ADC_SQR4_SQ7_N, 38
ADC_SQR4_SQ8_N, 38
ADC_SQR4_SQ9_N, 38
ADC_SQR5_SQ1_N, 38
ADC_SQR5_SQ2_N, 38
ADC_SQR5_SQ3_N, 38
ADC_SQR5_SQ4_N, 38
ADC_SQR5_SQ5_N, 38
ADC_SQR5_SQ6_N, 38
ADC_USE_MUTUAL_EXCLUSION, 31
ADC_USE_WAIT, 31
adcAcquireBus, 27
adccallback_t, 42
adcConvert, 28
ADCD1, 31
ADCDriver, 42
adcerror_t, 43
adcerrorcallback_t, 43
adclInit, 23
adcObjectInit, 24
adcReleaseBus, 28
adcsample_t, 42
adcStart, 24
adcStartConversion, 25
adcStartConversionl, 25
adcstate_t, 43
adcSTM32DisableTSVREFE, 30
adcSTM32EnableTSVREFE, 30
adcStop, 24
adcStopConversion, 26
adcStopConversionl, 27
CH_IRQ_HANDLER, 28
STM32_ADC_ADC1_DMA_IRQ_PRIORITY, 37
STM32_ADC_ADC1_DMA_PRIORITY, 37
STM32_ADC_ADCPRE, 37
STM32_ADC_IRQ_PRIORITY, 37
STM32_ADC_USE_ADC1, 37
adc.c, 365
adc.h, 366
ADC1_IRQHandler
 HAL Driver, 89
ADC_ACTIVE
 ADC Driver, 43
ADC_COMPLETE
 ADC Driver, 43
ADC_ERR_DMAFAILURE
 ADC Driver, 43
ADC_ERR_OVERFLOW
 ADC Driver, 43
ADC_ERROR
 ADC Driver, 43
ADC_READY
 ADC Driver, 43
ADC_STOP
 ADC Driver, 43
ADC_UNINIT
 ADC Driver, 43
ADC_CHANNEL_IN0
 ADC Driver, 34

ADC_CHANNEL_IN1
 ADC Driver, 34
ADC_CHANNEL_IN10
 ADC Driver, 35
ADC_CHANNEL_IN11
 ADC Driver, 35
ADC_CHANNEL_IN12
 ADC Driver, 35
ADC_CHANNEL_IN13
 ADC Driver, 35
ADC_CHANNEL_IN14
 ADC Driver, 35
ADC_CHANNEL_IN15
 ADC Driver, 35
ADC_CHANNEL_IN18
 ADC Driver, 36
ADC_CHANNEL_IN19
 ADC Driver, 36
ADC_CHANNEL_IN2
 ADC Driver, 34
ADC_CHANNEL_IN20
 ADC Driver, 36
ADC_CHANNEL_IN21
 ADC Driver, 36
ADC_CHANNEL_IN22
 ADC Driver, 36
ADC_CHANNEL_IN23
 ADC Driver, 36
ADC_CHANNEL_IN24
 ADC Driver, 36
ADC_CHANNEL_IN25
 ADC Driver, 36
ADC_CHANNEL_IN3
 ADC Driver, 34
ADC_CHANNEL_IN4
 ADC Driver, 34
ADC_CHANNEL_IN5
 ADC Driver, 35
ADC_CHANNEL_IN6
 ADC Driver, 35
ADC_CHANNEL_IN7
 ADC Driver, 35
ADC_CHANNEL_IN8
 ADC Driver, 35
ADC_CHANNEL_IN9
 ADC Driver, 35
ADC_CHANNEL_SENSOR
 ADC Driver, 35
ADC_CHANNEL_VREFINT
 ADC Driver, 36
adc_channels_num_t
 ADC Driver, 42
ADC_CR2_EXTSEL_SRC
 ADC Driver, 34
adc_lld.c, 367
adc_lld.h, 367
adc_lld_init
 ADC Driver, 29
adc_lld_start

ADC Driver, 29
adc_lld_start_conversion
 ADC Driver, 30
adc_lld_stop
 ADC Driver, 29
adc_lld_stop_conversion
 ADC Driver, 30
ADC_SAMPLE_16
 ADC Driver, 36
ADC_SAMPLE_192
 ADC Driver, 37
ADC_SAMPLE_24
 ADC Driver, 37
ADC_SAMPLE_384
 ADC Driver, 37
ADC_SAMPLE_4
 ADC Driver, 36
ADC_SAMPLE_48
 ADC Driver, 37
ADC_SAMPLE_9
 ADC Driver, 36
ADC_SAMPLE_96
 ADC Driver, 37
ADC_SMPR1_SMP_AN20
 ADC Driver, 42
ADC_SMPR1_SMP_AN21
 ADC Driver, 42
ADC_SMPR1_SMP_AN22
 ADC Driver, 42
ADC_SMPR1_SMP_AN23
 ADC Driver, 42
ADC_SMPR1_SMP_AN24
 ADC Driver, 42
ADC_SMPR1_SMP_AN25
 ADC Driver, 42
ADC_SMPR2_SMP_AN10
 ADC Driver, 41
ADC_SMPR2_SMP_AN11
 ADC Driver, 41
ADC_SMPR2_SMP_AN12
 ADC Driver, 41
ADC_SMPR2_SMP_AN13
 ADC Driver, 41
ADC_SMPR2_SMP_AN14
 ADC Driver, 41
ADC_SMPR2_SMP_AN15
 ADC Driver, 41
ADC_SMPR2_SMP_AN18
 ADC Driver, 41
ADC_SMPR2_SMP_AN19
 ADC Driver, 41
ADC_SMPR2_SMP_SENSOR
 ADC Driver, 41
ADC_SMPR2_SMP_VREF
 ADC Driver, 41
ADC_SMPR3_SMP_AN0
 ADC Driver, 40
ADC_SMPR3_SMP_AN1
 ADC Driver, 40

ADC_SMPR3_SMP_AN2
 ADC Driver, 40
ADC_SMPR3_SMP_AN3
 ADC Driver, 40
ADC_SMPR3_SMP_AN4
 ADC Driver, 40
ADC_SMPR3_SMP_AN5
 ADC Driver, 40
ADC_SMPR3_SMP_AN6
 ADC Driver, 40
ADC_SMPR3_SMP_AN7
 ADC Driver, 40
ADC_SMPR3_SMP_AN8
 ADC Driver, 41
ADC_SMPR3_SMP_AN9
 ADC Driver, 41
ADC_SQR1_NUM_CH
 ADC Driver, 38
ADC_SQR1_SQ25_N
 ADC Driver, 40
ADC_SQR1_SQ26_N
 ADC Driver, 40
ADC_SQR1_SQ27_N
 ADC Driver, 40
ADC_SQR2_SQ19_N
 ADC Driver, 39
ADC_SQR2_SQ20_N
 ADC Driver, 39
ADC_SQR2_SQ21_N
 ADC Driver, 39
ADC_SQR2_SQ22_N
 ADC Driver, 39
ADC_SQR2_SQ23_N
 ADC Driver, 39
ADC_SQR2_SQ24_N
 ADC Driver, 40
ADC_SQR3_SQ13_N
 ADC Driver, 39
ADC_SQR3_SQ14_N
 ADC Driver, 39
ADC_SQR3_SQ15_N
 ADC Driver, 39
ADC_SQR3_SQ16_N
 ADC Driver, 39
ADC_SQR3_SQ17_N
 ADC Driver, 39
ADC_SQR3_SQ18_N
 ADC Driver, 39
ADC_SQR4_SQ10_N
 ADC Driver, 38
ADC_SQR4_SQ11_N
 ADC Driver, 38
ADC_SQR4_SQ12_N
 ADC Driver, 39
ADC_SQR4_SQ7_N
 ADC Driver, 38
ADC_SQR4_SQ8_N
 ADC Driver, 38
ADC_SQR4_SQ9_N
 ADC Driver, 38
ADC Driver, 38
ADC_SQR5_SQ1_N
 ADC Driver, 38
ADC_SQR5_SQ2_N
 ADC Driver, 38
ADC_SQR5_SQ3_N
 ADC Driver, 38
ADC_SQR5_SQ4_N
 ADC Driver, 38
ADC_SQR5_SQ5_N
 ADC Driver, 38
ADC_SQR5_SQ6_N
 ADC Driver, 38
ADC_USE_MUTUAL_EXCLUSION
 ADC Driver, 31
 Configuration, 13
ADC_USE_WAIT
 ADC Driver, 31
 Configuration, 13
adcAcquireBus
 ADC Driver, 27
adccallback_t
 ADC Driver, 42
ADCConfig, 302
ADCCConversionGroup, 302
 circular, 304
 cr1, 304
 cr2, 304
 end_cb, 304
 error_cb, 304
 num_channels, 304
 smpr1, 305
 smpr2, 305
 smpr3, 305
 sqr1, 305
 sqr2, 305
 sqr3, 305
 sqr4, 305
 sqr5, 305
adcConvert
 ADC Driver, 28
ADCD1
 ADC Driver, 31
ADCDriver, 306
 adc, 308
 ADC Driver, 42
 config, 308
 depth, 308
 dmamode, 309
 dmastp, 309
 grpp, 308
 mutex, 308
 samples, 308
 state, 308
 thread, 308
adcerror_t
 ADC Driver, 43
adcerrorcallback_t
 ADC Driver, 43

adclInit
 ADC Driver, 23
adcObjectInit
 ADC Driver, 24
adcReleaseBus
 ADC Driver, 28
adcsample_t
 ADC Driver, 42
adcStart
 ADC Driver, 24
adcStartConversion
 ADC Driver, 25
adcStartConversionl
 ADC Driver, 25
adcstate_t
 ADC Driver, 43
adcSTM32DisableTSVREFE
 ADC Driver, 30
adcSTM32EnableTSVREFE
 ADC Driver, 30
adcStop
 ADC Driver, 24
adcStopConversion
 ADC Driver, 26
adcStopConversionl
 ADC Driver, 27
addr
 I2CDriver, 319
address
 USBDriver, 360
afrh
 stm32_gpio_setup_t, 346
afrl
 stm32_gpio_setup_t, 346
bdtr
 PWMConfig, 335
best
 TimeMeasurement, 349
callback
 GPTConfig, 315
 PWMChannelConfig, 333
 PWMConfig, 335
CAN_USE_SLEEP_MODE
 Configuration, 14
cb
 EXTChannelConfig, 310
CH_IRQ_HANDLER
 ADC Driver, 28
 EXT Driver, 49, 50
 I2C Driver, 104, 105
 Serial Driver, 179
 STM32L1xx DMA Support, 277, 278
 UART Driver, 223
 USB Driver, 247
channel
 stm32_dma_stream_t, 345
channels

EXTConfig, 311
PWMConfig, 335
circular
 ADCConversionGroup, 304
clock
 GPTDriver, 317
 ICUDriver, 324
 PWMDriver, 338
clock_speed
 I2CConfig, 317
cnt
 MMCDriver, 329
COMP_IRQHandler
 HAL Driver, 89
config
 ACDDriver, 308
 EXTDriver, 312
 GPTDriver, 317
 I2CDriver, 319
 ICUDriver, 324
 MMCDriver, 328
 PWMDriver, 338
 SPIDriver, 344
 UARTDriver, 354
 USBDriver, 359
Configuration, 10
 ADC_USE_MUTUAL_EXCLUSION, 13
 ADC_USE_WAIT, 13
 CAN_USE_SLEEP_MODE, 14
 HAL_USE_ADC, 12
 HAL_USE_CAN, 12
 HAL_USE_EXT, 12
 HAL_USE_GPT, 12
 HAL_USE_I2C, 12
 HAL_USE_ICU, 12
 HAL_USE_MAC, 12
 HAL_USE_MMCSPI, 13
 HAL_USE_PAL, 12
 HAL_USE_PWM, 13
 HAL_USE_RTC, 13
 HAL_USE_SDC, 13
 HAL_USE_SERIAL, 13
 HAL_USE_SERIALUSB, 13
 HAL_USE_SPI, 13
 HAL_USE_UART, 13
 HAL_USE_USB, 13
 I2C_USE_MUTUAL_EXCLUSION, 14
 MAC_USE_EVENTS, 14
 MMC_NICE_WAITING, 14
 MMC_POLLING_DELAY, 14
 MMC_POLLING_INTERVAL, 14
 MMC_SECTOR_SIZE, 14
 MMC_USE_SPI_POLLING, 14
 SDC_INIT_RETRY, 14
 SDC_MMCSUPPORT, 14
 SDC_NICE_WAITING, 15
 SERIAL_BUFFERS_SIZE, 15
 SERIAL_DEFAULT_BITRATE, 15
 SERIAL_USB_BUFFERS_SIZE, 15

SPI_USE_MUTUAL_EXCLUSION, 15
SPI_USE_WAIT, 15
configuration
 USBDriver, 360
cr1
 ADCConversionGroup, 304
 SPIConfig, 342
 UARTConfig, 351
cr2
 ADCConversionGroup, 304
 PWMConfig, 335
 UARTConfig, 351
cr3
 UARTConfig, 351
DAC_IRQHandler
 HAL Driver, 89
depth
 ADCDriver, 308
DMA1_Ch1_IRQHandler
 HAL Driver, 88
DMA1_Ch2_IRQHandler
 HAL Driver, 88
DMA1_Ch3_IRQHandler
 HAL Driver, 88
DMA1_Ch4_IRQHandler
 HAL Driver, 88
DMA1_Ch5_IRQHandler
 HAL Driver, 88
DMA1_Ch6_IRQHandler
 HAL Driver, 88
DMA1_Ch7_IRQHandler
 HAL Driver, 88
dmalinit
 STM32L1xx DMA Support, 278
dmamode
 ADCDriver, 309
 I2CDriver, 319
 UARTDriver, 354
dmarx
 I2CDriver, 320
 SPIDriver, 344
 UARTDriver, 354
dmaStartMemcpy
 STM32L1xx DMA Support, 285
dmastp
 ADCDriver, 309
dmaStreamAllocate
 STM32L1xx DMA Support, 278
dmaStreamClearInterrupt
 STM32L1xx DMA Support, 285
dmaStreamDisable
 STM32L1xx DMA Support, 284
dmaStreamEnable
 STM32L1xx DMA Support, 284
dmaStreamGetTransactionSize
 STM32L1xx DMA Support, 283
dmaStreamRelease
 STM32L1xx DMA Support, 279
dmaStreamSetMemory0
 STM32L1xx DMA Support, 282
dmaStreamSetMode
 STM32L1xx DMA Support, 283
dmaStreamSetPeripheral
 STM32L1xx DMA Support, 281
dmaStreamSetTransactionSize
 STM32L1xx DMA Support, 282
dmatx
 I2CDriver, 320
 SPIDriver, 344
 UARTDriver, 354
dmaWaitCompletion
 STM32L1xx DMA Support, 286
duty_cycle
 I2CConfig, 318
end_cb
 ADCConversionGroup, 304
 SPIConfig, 342
ep0endcb
 USBDriver, 360
ep0n
 USBDriver, 360
ep0next
 USBDriver, 359
ep0state
 USBDriver, 359
EP_STATUS_ACTIVE
 USB Driver, 267
EP_STATUS_DISABLED
 USB Driver, 267
EP_STATUS_STALLED
 USB Driver, 267
ep_mode
 USBEndpointConfig, 362
epc
 USBDriver, 359
EPR
 stm32_usb_t, 348
EPR_TOGGLE_MASK
 USB Driver, 263
error_cb
 ADCConversionGroup, 304
errors
 I2CDriver, 319
event_cb
 USBConfig, 357
expchannel_t
 EXT Driver, 54
EXT Driver, 43
 CH_IRQ_HANDLER, 49, 50
 expchannel_t, 54
 EXT_CHANNELS_MASK, 52
 ext_lld_channel_disable, 51
 ext_lld_channel_enable, 51
 ext_lld_init, 50
 ext_lld_start, 50
 ext_lld_stop, 51

EXT_MAX_CHANNELS, 52
EXT_MODE_EXTI, 52
EXT_MODE_GPIOA, 52
EXT_MODE_GPIOB, 52
EXT_MODE_GPIOC, 52
EXT_MODE_GPIOD, 52
EXT_MODE_GPIOE, 52
EXT_MODE_GPIOF, 52
EXT_MODE_GPIOG, 52
EXT_MODE_GPIOH, 52
EXT_MODE_GPIOI, 53
extcallback_t, 54
extChannelDisable, 48
extChannelEnable, 48
EXTD1, 51
extInit, 47
extObjectInit, 47
extStart, 47
extStop, 48
STM32_EXT EXTI0 IRQ_PRIORITY, 53
STM32_EXT EXTI10_15 IRQ_PRIORITY, 53
STM32_EXT EXTI16 IRQ_PRIORITY, 53
STM32_EXT EXTI17 IRQ_PRIORITY, 53
STM32_EXT EXTI18 IRQ_PRIORITY, 53
STM32_EXT EXTI19 IRQ_PRIORITY, 53
STM32_EXT EXTI1 IRQ_PRIORITY, 53
STM32_EXT EXTI20 IRQ_PRIORITY, 54
STM32_EXT EXTI21 IRQ_PRIORITY, 54
STM32_EXT EXTI22 IRQ_PRIORITY, 54
STM32_EXT EXTI2 IRQ_PRIORITY, 53
STM32_EXT EXTI3 IRQ_PRIORITY, 53
STM32_EXT EXTI4 IRQ_PRIORITY, 53
STM32_EXT EXTI5_9 IRQ_PRIORITY, 53
ext.c, 372
EXT_CHANNELS_MASK
 EXT Driver, 52
ext_lld.c, 373
ext_lld.h, 373
ext_lld_channel_disable
 EXT Driver, 51
ext_lld_channel_enable
 EXT Driver, 51
ext_lld_init
 EXT Driver, 50
ext_lld_start
 EXT Driver, 50
ext_lld_stop
 EXT Driver, 51
EXT_MAX_CHANNELS
 EXT Driver, 52
EXT_MODE_EXTI
 EXT Driver, 52
EXT_MODE_GPIOA
 EXT Driver, 52
EXT_MODE_GPIOB
 EXT Driver, 52
EXT_MODE_GPIOC
 EXT Driver, 52
EXT_MODE_GPIOD
 EXT Driver, 52
EXT_MODE_GPIOE
 EXT Driver, 52
EXT_MODE_GPIOF
 EXT Driver, 52
EXT_MODE_GPIOG
 EXT Driver, 52
EXT_MODE_GPIOH
 EXT Driver, 52
EXT_MODE_GPIOI
 EXT Driver, 53
extcallback_t
 EXT Driver, 54
EXTChannelConfig, 309
 cb, 310
 mode, 310
extChannelDisable
 EXT Driver, 48
extChannelEnable
 EXT Driver, 48
EXTConfig, 310
 channels, 311
 exti, 311
EXTD1
 EXT Driver, 51
EXTDriver, 312
 config, 312
 state, 312
exti
 EXTConfig, 311
EXTI0_IRQHandler
 HAL Driver, 88
EXTI15_10_IRQHandler
 HAL Driver, 90
EXTI1_IRQHandler
 HAL Driver, 88
EXTI2_IRQHandler
 HAL Driver, 88
EXTI3_IRQHandler
 HAL Driver, 88
EXTI4_IRQHandler
 HAL Driver, 88
EXTI9_5_IRQHandler
 HAL Driver, 89
extInit
 EXT Driver, 47
extObjectInit
 EXT Driver, 47
extStart
 EXT Driver, 47
extStop
 EXT Driver, 48
FLASH_IRQHandler
 HAL Driver, 87
frequency
 GPTConfig, 315
 ICUConfig, 322
 PWMConfig, 335

get_descriptor_cb
 USBConfig, 357

GPIO_TypeDef, 313

GPT Driver, 54
 GPT_CONTINUOUS, 67
 GPT_ONESHOT, 67
 GPT_READY, 67
 GPT_STOP, 67
 GPT_UNINIT, 67
 gpt_lld_init, 63
 gpt_lld_polled_delay, 64
 gpt_lld_start, 63
 gpt_lld_start_timer, 63
 gpt_lld_stop, 63
 gpt_lld_stop_timer, 64
 gptcallback_t, 67
 gptcnt_t, 67
 GPTD1, 64
 GPTD2, 64
 GPTD3, 64
 GPTD4, 65
 GPTD5, 65
 GPTD8, 65
 GPTDriver, 67
 gptfreq_t, 67
 gptInit, 58
 gptObjectInit, 58
 gptPolledDelay, 62
 gptStart, 58
 gptStartContinuous, 59
 gptStartContinuousl, 60
 gptStartOneShot, 60
 gptStartOneShotl, 61
 gptstate_t, 67
 gptStop, 59
 gptStopTimer, 61
 gptStopTimerl, 62
 STM32_GPT_TIM1_IRQ_PRIORITY, 66
 STM32_GPT_TIM2_IRQ_PRIORITY, 66
 STM32_GPT_TIM3_IRQ_PRIORITY, 66
 STM32_GPT_TIM4_IRQ_PRIORITY, 66
 STM32_GPT_TIM5_IRQ_PRIORITY, 66
 STM32_GPT_TIM8_IRQ_PRIORITY, 67
 STM32_GPT_USE_TIM1, 65
 STM32_GPT_USE_TIM2, 65
 STM32_GPT_USE_TIM3, 65
 STM32_GPT_USE_TIM4, 66
 STM32_GPT_USE_TIM5, 66
 STM32_GPT_USE_TIM8, 66

gpt.c, 375

gpt.h, 376

GPT_CONTINUOUS
 GPT Driver, 67

GPT_ONESHOT
 GPT Driver, 67

GPT_READY
 GPT Driver, 67

GPT_STOP
 GPT Driver, 67

GPT_UNINIT
 GPT Driver, 67

gpt_lld.c, 377

gpt_lld.h, 378

gpt_lld_init
 GPT Driver, 63

gpt_lld_polled_delay
 GPT Driver, 64

gpt_lld_start
 GPT Driver, 63

gpt_lld_start_timer
 GPT Driver, 63

gpt_lld_stop
 GPT Driver, 63

gpt_lld_stop_timer
 GPT Driver, 64

gptcallback_t
 GPT Driver, 67

gptcnt_t
 GPT Driver, 67

GPTConfig, 313
 callback, 315
 frequency, 315

GPTD1
 GPT Driver, 64

GPTD2
 GPT Driver, 64

GPTD3
 GPT Driver, 64

GPTD4
 GPT Driver, 65

GPTD5
 GPT Driver, 65

GPTD8
 GPT Driver, 65

GPTDriver, 315
 clock, 317
 config, 317
 GPT Driver, 67
 state, 317
 tim, 317

gptfreq_t
 GPT Driver, 67

gptInit
 GPT Driver, 58

gptObjectInit
 GPT Driver, 58

gptPolledDelay
 GPT Driver, 62

gptStart
 GPT Driver, 58

gptStartContinuous
 GPT Driver, 59

gptStartContinuousl
 GPT Driver, 60

gptStartOneShot
 GPT Driver, 60

gptStartOneShotl
 GPT Driver, 61

gptstate_t
 GPT Driver, 67

gptStop
 GPT Driver, 59

gptStopTimer
 GPT Driver, 61

gptStopTimerl
 GPT Driver, 62

grpp
 ADCDriver, 308

HAL, 8

HAL Driver, 67

- ADC1_IRQHandler, 89
- COMP_IRQHandler, 89
- DAC_IRQHandler, 89
- DMA1_Ch1_IRQHandler, 88
- DMA1_Ch2_IRQHandler, 88
- DMA1_Ch3_IRQHandler, 88
- DMA1_Ch4_IRQHandler, 88
- DMA1_Ch5_IRQHandler, 88
- DMA1_Ch6_IRQHandler, 88
- DMA1_Ch7_IRQHandler, 88
- EXTI0_IRQHandler, 88
- EXTI15_10_IRQHandler, 90
- EXTI1_IRQHandler, 88
- EXTI2_IRQHandler, 88
- EXTI3_IRQHandler, 88
- EXTI4_IRQHandler, 88
- EXTI9_5_IRQHandler, 89
- FLASH_IRQHandler, 87
- HAL_IMPLEMENTS_COUNTERS, 81
- hal_lld_get_counter_frequency, 96
- hal_lld_get_counter_value, 96
- hal_lld_init, 79
- halclock_t, 96
- halGetCounterFrequency, 81
- halGetCounterValue, 80
- hallInit, 76
- hallsCounterWithin, 77
- halPolledDelay, 78
- halrtcnt_t, 96
- I2C1_ER_IRQHandler, 90
- I2C1_EV_IRQHandler, 90
- I2C2_ER_IRQHandler, 90
- I2C2_EV_IRQHandler, 90
- LCD_IRQHandler, 89
- MS2RTT, 80
- PVD_IRQHandler, 87
- RCC_IRQHandler, 87
- RTC_Alarm_IRQHandler, 90
- RTC_WKUP_IRQHandler, 87
- S2RTT, 79
- SPI1_IRQHandler, 90
- SPI2_IRQHandler, 90
- STM32_0WS_THRESHOLD, 94
- STM32_ACTIVATE_PLL, 94
- STM32_ADC_CLOCK_ENABLED, 92
- STM32_ADCCLK, 95
- stm32_clock_init, 79
- STM32_FLASHBITS1, 95
- STM32_HCLK, 95
- STM32_HPRE, 92
- STM32_HPRE_DIV1, 83
- STM32_HPRE_DIV128, 84
- STM32_HPRE_DIV16, 83
- STM32_HPRE_DIV2, 83
- STM32_HPRE_DIV256, 84
- STM32_HPRE_DIV4, 83
- STM32_HPRE_DIV512, 84
- STM32_HPRE_DIV64, 84
- STM32_HPRE_DIV8, 83
- STM32_HSE_ENABLED, 91
- STM32_HSECLK_MAX, 93
- STM32_HSEDIVCLK, 95
- STM32_HSI_AVAILABLE, 94
- STM32_HSI_ENABLED, 91
- STM32_HSICLK, 81
- STM32_LSE_ENABLED, 91
- STM32_LSI_ENABLED, 91
- STM32_LSICLK, 81
- STM32_MCOPRE, 93
- STM32_MCOPRE_DIV1, 86
- STM32_MCOPRE_DIV16, 86
- STM32_MCOPRE_DIV2, 86
- STM32_MCOPRE_DIV4, 86
- STM32_MCOPRE_DIV8, 86
- STM32_MCOSEL, 93
- STM32_MCOSEL_HSE, 85
- STM32_MCOSEL_HSI, 85
- STM32_MCOSEL_LSE, 85
- STM32_MCOSEL_LSI, 85
- STM32_MCOSEL_MS1, 85
- STM32_MCOSEL_NO CLOCK, 85
- STM32_MCOSEL_PLL, 85
- STM32_MCOSEL_SYSCLK, 85
- STM32_MSICLK, 94
- STM32_MSIRANGE, 92
- STM32_MSIRANGE_128K, 86
- STM32_MSIRANGE_1M, 86
- STM32_MSIRANGE_256K, 86
- STM32_MSIRANGE_2M, 86
- STM32_MSIRANGE_4M, 87
- STM32_MSIRANGE_512K, 86
- STM32_MSIRANGE_64K, 86
- STM32_MSIRANGE_MASK, 86
- STM32_NO_INIT, 91
- STM32_PCLK1, 95
- STM32_PCLK1_MAX, 93
- STM32_PCLK2, 95
- STM32_PCLK2_MAX, 94
- STM32_PLLCLKIN, 94
- STM32_PLLCLKOUT, 94
- STM32_PLLDIV, 94
- STM32_PLLDIV_VALUE, 92
- STM32_PLLMUL, 94
- STM32_PLLMUL_VALUE, 92
- STM32_PLLSRC, 92

STM32_PLLSRC_HSE, 85
STM32_PLLSRC_HSI, 85
STM32_PLLVCO, 94
STM32_PLLVCO_MAX, 93
STM32_PLLVCO_MIN, 93
STM32_PLS, 91
STM32_PLS_LEV0, 82
STM32_PLS_LEV1, 82
STM32_PLS_LEV2, 82
STM32_PLS_LEV3, 82
STM32_PLS_LEV4, 82
STM32_PLS_LEV5, 82
STM32_PLS_LEV6, 82
STM32_PLS_LEV7, 82
STM32_PLS_MASK, 82
STM32_PPREG1, 93
STM32_PPREG1_DIV1, 84
STM32_PPREG1_DIV16, 84
STM32_PPREG1_DIV2, 84
STM32_PPREG1_DIV4, 84
STM32_PPREG1_DIV8, 84
STM32_PPREG2, 93
STM32_PPREG2_DIV1, 84
STM32_PPREG2_DIV16, 85
STM32_PPREG2_DIV2, 84
STM32_PPREG2_DIV4, 84
STM32_PPREG2_DIV8, 85
STM32_PVD_ENABLE, 91
STM32_RTCPRE, 93
STM32_RTCPRE_DIV16, 83
STM32_RTCPRE_DIV2, 82
STM32_RTCPRE_DIV4, 83
STM32_RTCPRE_DIV8, 83
STM32_RTCPRE_MASK, 82
STM32_RTCSEL, 93
STM32_RTCSEL_HSEDIV, 87
STM32_RTCSEL_LSE, 87
STM32_RTCSEL_LSI, 87
STM32_RTCSEL_MASK, 87
STM32_RTCSEL_NOCLOCK, 87
STM32_SW, 92
STM32_SW_HSE, 83
STM32_SW_HSI, 83
STM32_SW_MSI, 83
STM32_SW_PLL, 83
STM32_SYSCLK, 94
STM32_SYSCLK_MAX, 93
STM32_TIMCLK1, 95
STM32_TIMCLK2, 95
STM32_USB_CLOCK_ENABLED, 92
STM32_USBCLK, 95
STM32_VOS, 91
STM32_VOS_1P2, 82
STM32_VOS_1P5, 81
STM32_VOS_1P8, 81
STM32_VOS_MASK, 81
STM_MCOCLK, 95
STM_MCODIVCLK, 95
STM_RTCCLK, 95
TAMPER_STAMP_IRQHandler, 87
TIM10_IRQHandler, 89
TIM11_IRQHandler, 89
TIM2_IRQHandler, 89
TIM3_IRQHandler, 89
TIM4_IRQHandler, 90
TIM6_IRQHandler, 91
TIM7_IRQHandler, 91
TIM9_IRQHandler, 89
US2RTT, 80
USART1_IRQHandler, 90
USART2_IRQHandler, 90
USART3_IRQHandler, 90
USB_FS_WKUP_IRQHandler, 91
USB_HP_IRQHandler, 89
USB_LP_IRQHandler, 89
WWDG_IRQHandler, 87
hal.c, 379
hal.h, 379
HAL_IMPLEMENTANTS_COUNTERS
 HAL Driver, 81
hal_lld.c, 380
hal_lld.h, 380
hal_lld_get_counter_frequency
 HAL Driver, 96
hal_lld_get_counter_value
 HAL Driver, 96
hal_lld_init
 HAL Driver, 79
HAL_USE_ADC
 Configuration, 12
HAL_USE_CAN
 Configuration, 12
HAL_USE_EXT
 Configuration, 12
HAL_USE_GPT
 Configuration, 12
HAL_USE_I2C
 Configuration, 12
HAL_USE_ICU
 Configuration, 12
HAL_USE_MAC
 Configuration, 12
HAL_USE_MMC_SPI
 Configuration, 13
HAL_USE_PAL
 Configuration, 12
HAL_USE_PWM
 Configuration, 13
HAL_USE_RTC
 Configuration, 13
HAL_USE_SDC
 Configuration, 13
HAL_USE_SERIAL
 Configuration, 13
HAL_USE_SERIAL_USB
 Configuration, 13
HAL_USE_SPI
 Configuration, 13

HAL_USE_UART
 Configuration, 13
HAL_USE_USB
 Configuration, 13
halclock_t
 HAL Driver, 96
halconf.h, 387
halGetCounterFrequency
 HAL Driver, 81
halGetCounterValue
 HAL Driver, 80
hallInit
 HAL Driver, 76
hallsCounterWithin
 HAL Driver, 77
halPolledDelay
 HAL Driver, 78
halrtcnt_t
 HAL Driver, 96
hscfg
 MMCDriver, 328

i2c
 I2CDriver, 320
I2C Driver, 96
 CH_IRQ_HANDLER, 104, 105
 I2C_ACTIVE_RX, 113
 I2C_ACTIVE_TX, 113
 I2C_READY, 113
 I2C_STOP, 113
 I2C_UNINIT, 113
 I2C_CLK_FREQ, 109
 i2c_lld_get_errors, 112
 i2c_lld_init, 105
 i2c_lld_master_receive_timeout, 106
 i2c_lld_master_transmit_timeout, 107
 i2c_lld_start, 105
 i2c_lld_stop, 106
 I2C_USE_MUTUAL_EXCLUSION, 108
 i2cAcquireBus, 104
 i2caddr_t, 112
 I2CD1, 108
 I2CD2, 108
 I2CD3, 108
 I2CD_ACK_FAILURE, 108
 I2CD_ARBITRATION_LOST, 108
 I2CD_BUS_ERROR, 108
 I2CD_NO_ERROR, 108
 I2CD_OVERRUN, 108
 I2CD_PEC_ERROR, 108
 I2CD_SMB_ALERT, 108
 I2CD_TIMEOUT, 108
 I2CDriver, 112
 i2cdutycycle_t, 113
 i2cflags_t, 112
 i2cGetErrors, 102
 i2cInit, 100
 i2cMasterReceive, 109
 i2cMasterReceiveTimeout, 103
 i2cMasterTransmit, 109
 i2cMasterTransmitTimeout, 102
 i2cObjectInit, 101
 i2copmode_t, 113
 i2cReleaseBus, 104
 i2cStart, 101
 i2cstate_t, 113
 i2cStop, 101
 STM32_DMA_REQUIRED, 112
 STM32_I2C_DMA_ERROR_HOOK, 111
 STM32_I2C_I2C1_DMA_PRIORITY, 110
 STM32_I2C_I2C1_IRQ_PRIORITY, 110
 STM32_I2C_I2C1_RX_DMA_STREAM, 111
 STM32_I2C_I2C1_TX_DMA_STREAM, 111
 STM32_I2C_I2C2_DMA_PRIORITY, 110
 STM32_I2C_I2C2_IRQ_PRIORITY, 110
 STM32_I2C_I2C2_RX_DMA_STREAM, 111
 STM32_I2C_I2C2_TX_DMA_STREAM, 111
 STM32_I2C_I2C3_DMA_PRIORITY, 111
 STM32_I2C_I2C3_IRQ_PRIORITY, 110
 STM32_I2C_I2C3_RX_DMA_STREAM, 112
 STM32_I2C_I2C3_TX_DMA_STREAM, 112
 STM32_I2C_USE_I2C1, 109
 STM32_I2C_USE_I2C2, 110
 STM32_I2C_USE_I2C3, 110
 wakeup_isr, 109

 i2c.c, 389
 i2c.h, 389
 I2C1_ER_IRQHandler
 HAL Driver, 90
 I2C1_EV_IRQHandler
 HAL Driver, 90
 I2C2_ER_IRQHandler
 HAL Driver, 90
 I2C2_EV_IRQHandler
 HAL Driver, 90
 I2C_ACTIVE_RX
 I2C Driver, 113
 I2C_ACTIVE_TX
 I2C Driver, 113
 I2C_READY
 I2C Driver, 113
 I2C_STOP
 I2C Driver, 113
 I2C_UNINIT
 I2C Driver, 113
 I2C_CLK_FREQ
 I2C Driver, 109
 i2c_lld.c, 391
 i2c_lld.h, 392
 i2c_lld_get_errors
 I2C Driver, 112
 i2c_lld_init
 I2C Driver, 105
 i2c_lld_master_receive_timeout
 I2C Driver, 106
 i2c_lld_master_transmit_timeout
 I2C Driver, 107
 i2c_lld_start

I2C Driver, 105
i2c_lld_stop
 I2C Driver, 106
I2C_USE_MUTUAL_EXCLUSION
 Configuration, 14
 I2C Driver, 108
i2cAcquireBus
 I2C Driver, 104
i2caddr_t
 I2C Driver, 112
I2CConfig, 317
 clock_speed, 317
 duty_cycle, 318
 op_mode, 317
I2CD1
 I2C Driver, 108
I2CD2
 I2C Driver, 108
I2CD3
 I2C Driver, 108
I2CD_ACK_FAILURE
 I2C Driver, 108
I2CD_ARBITRATION_LOST
 I2C Driver, 108
I2CD_BUS_ERROR
 I2C Driver, 108
I2CD_NO_ERROR
 I2C Driver, 108
I2CD_OVERRUN
 I2C Driver, 108
I2CD_PEC_ERROR
 I2C Driver, 108
I2CD_SMB_ALERT
 I2C Driver, 108
I2CD_TIMEOUT
 I2C Driver, 108
I2CDriver, 318
 addr, 319
 config, 319
 dmemode, 319
 dmarx, 320
 dmatx, 320
 errors, 319
 i2c, 320
 I2C Driver, 112
 mutex, 319
 state, 319
 thread, 319
i2cdutycycle_t
 I2C Driver, 113
i2cflags_t
 I2C Driver, 112
i2cGetErrors
 I2C Driver, 102
i2cInit
 I2C Driver, 100
i2cMasterReceive
 I2C Driver, 109
i2cMasterReceiveTimeout

I2C Driver, 103
i2cMasterTransmit
 I2C Driver, 109
i2cMasterTransmitTimeout
 I2C Driver, 102
i2cObjectInit
 I2C Driver, 101
i2copmode_t
 I2C Driver, 113
i2cReleaseBus
 I2C Driver, 104
i2cStart
 I2C Driver, 101
i2cstate_t
 I2C Driver, 113
i2cStop
 I2C Driver, 101
ICU Driver, 113
 _icu_isr_invoke_period_cb, 123
 _icu_isr_invoke_width_cb, 122
 ICU_ACTIVE, 126
 ICU_IDLE, 126
 ICU_INPUT_ACTIVE_HIGH, 126
 ICU_INPUT_ACTIVE_LOW, 126
 ICU_READY, 126
 ICU_STOP, 126
 ICU_UNINIT, 126
 ICU_WAITING, 126
 icu_lld_disable, 120
 icu_lld_enable, 120
 icu_lld_get_period, 125
 icu_lld_get_width, 124
 icu_lld_init, 119
 icu_lld_start, 119
 icu_lld_stop, 120
 icucallback_t, 125
 icucnt_t, 125
 ICUD1, 120
 ICUD2, 120
 ICUD3, 121
 ICUD4, 121
 ICUD5, 121
 ICUD8, 121
 icuDisable, 119
 icuDisablel, 121
 ICUDriver, 125
 icuEnable, 118
 icuEnablel, 121
 icufreq_t, 125
 icuGetPeriodl, 122
 icuGetWidthl, 122
 icuInit, 117
 icumode_t, 126
 icuObjectInit, 117
 icuStart, 117
 icustate_t, 126
 icuStop, 118
 STM32_ICU_TIM1_IRQ_PRIORITY, 124
 STM32_ICU_TIM2_IRQ_PRIORITY, 124

STM32_ICU_TIM3_IRQ_PRIORITY, 124
STM32_ICU_TIM4_IRQ_PRIORITY, 124
STM32_ICU_TIM5_IRQ_PRIORITY, 124
STM32_ICU_TIM8_IRQ_PRIORITY, 124
STM32_ICU_USE_TIM1, 123
STM32_ICU_USE_TIM2, 123
STM32_ICU_USE_TIM3, 123
STM32_ICU_USE_TIM4, 123
STM32_ICU_USE_TIM5, 124
STM32_ICU_USE_TIM8, 124

icu.c, 393
icu.h, 394
ICU_ACTIVE
 ICU Driver, 126
ICU_IDLE
 ICU Driver, 126
ICU_INPUT_ACTIVE_HIGH
 ICU Driver, 126
ICU_INPUT_ACTIVE_LOW
 ICU Driver, 126
ICU_READY
 ICU Driver, 126
ICU_STOP
 ICU Driver, 126
ICU_UNINIT
 ICU Driver, 126
ICU_WAITING
 ICU Driver, 126
icu_lld.c, 395
icu_lld.h, 395
icu_lld_disable
 ICU Driver, 120
icu_lld_enable
 ICU Driver, 120
icu_lld_get_period
 ICU Driver, 125
icu_lld_get_width
 ICU Driver, 124
icu_lld_init
 ICU Driver, 119
icu_lld_start
 ICU Driver, 119
icu_lld_stop
 ICU Driver, 120
icucallback_t
 ICU Driver, 125
icucnt_t
 ICU Driver, 125
ICUConfig, 320
 frequency, 322
 mode, 322
 period_cb, 322
 width_cb, 322
ICUD1
 ICU Driver, 120
ICUD2
 ICU Driver, 120
ICUD3
 ICU Driver, 121

ICUD4
 ICU Driver, 121
ICUD5
 ICU Driver, 121
ICUD8
 ICU Driver, 121
icuDisable
 ICU Driver, 119
icuDisableI
 ICU Driver, 121
ICUDriver, 322
 clock, 324
 config, 324
 ICU Driver, 125
 state, 324
 tim, 324
icuEnable
 ICU Driver, 118
icuEnableI
 ICU Driver, 121
icufreq_t
 ICU Driver, 125
icuGetPeriodI
 ICU Driver, 122
icuGetWidthI
 ICU Driver, 122
icuInit
 ICU Driver, 117
icumode_t
 ICU Driver, 126
icuObjectInit
 ICU Driver, 117
icuStart
 ICU Driver, 117
icustate_t
 ICU Driver, 126
icuStop
 ICU Driver, 118
ifcr
 stm32_dma_stream_t, 345
in
 USB Driver, 254
 usb_lld.c, 433
in_cb
 USBEndpointConfig, 362
in_maxsize
 USBEndpointConfig, 362
in_state
 USBEndpointConfig, 363
inserted_event
 MMCDriver, 328
IOBus, 324
 mask, 325
 offset, 326
 portid, 325
IOBUS_DECL
 PAL Driver, 144
iomode_t
 PAL Driver, 155

IOPORT1
 PAL Driver, 152
IOPORT2
 PAL Driver, 152
IOPORT3
 PAL Driver, 152
IOPORT4
 PAL Driver, 152
IOPORT5
 PAL Driver, 152
IOPORT6
 PAL Driver, 152
IOPORT7
 PAL Driver, 152
IOPORT8
 PAL Driver, 152
IOPORT9
 PAL Driver, 152
ioportid_t
 PAL Driver, 155
ioportmask_t
 PAL Driver, 155
is_inserted
 MMCDriver, 328
is_protected
 MMCDriver, 328
ishift
 stm32_dma_stream_t, 345
last
 TimeMeasurement, 349
LCD_IRQHandler
 HAL Driver, 89
lscfg
 MMCDriver, 328
MAC_USE_EVENTS
 Configuration, 14
mask
 IOBus, 325
MMC over SPI Driver, 126
 MMC_INSERTED, 136
 MMC_READING, 136
 MMC_READY, 136
 MMC_STOP, 136
 MMC_UNINIT, 136
 MMC_WAIT, 136
 MMC_WRITING, 136
 MMC_NICE_WAITING, 135
 MMC_POLLING_DELAY, 135
 MMC_POLLING_INTERVAL, 135
 MMC_SECTOR_SIZE, 135
mmcConnect, 129
mmcDisconnect, 130
mmcGetDriverState, 135
mmcInit, 128
mmclsWriteProtected, 135
mmcObjectInit, 128
mmcquery_t, 136
mmcSequentialRead, 131
mmcSequentialWrite, 133
mmcStart, 129
mmcStartSequentialRead, 131
mmcStartSequentialWrite, 133
mmcstate_t, 136
mmcStop, 129
mmcStopSequentialRead, 132
mmcStopSequentialWrite, 134
MMC_INSERTED
 MMC over SPI Driver, 136
MMC_READING
 MMC over SPI Driver, 136
MMC_READY
 MMC over SPI Driver, 136
MMC_STOP
 MMC over SPI Driver, 136
MMC_UNINIT
 MMC over SPI Driver, 136
MMC_WAIT
 MMC over SPI Driver, 136
MMC_WRITING
 MMC over SPI Driver, 136
MMC_NICE_WAITING
 Configuration, 14
 MMC over SPI Driver, 135
MMC_POLLING_DELAY
 Configuration, 14
 MMC over SPI Driver, 135
MMC_POLLING_INTERVAL
 Configuration, 14
 MMC over SPI Driver, 135
MMC_SECTOR_SIZE
 Configuration, 14
 MMC over SPI Driver, 135
mmc_spi.c, 397
mmc_spi.h, 398
MMC_USE_SPI_POLLING
 Configuration, 14
MMCConfig, 326
mmcConnect
 MMC over SPI Driver, 129
mmcDisconnect
 MMC over SPI Driver, 130
MMCDriver, 326
 cnt, 329
 config, 328
 hscfg, 328
 inserted_event, 328
 is_inserted, 328
 is_protected, 328
 lscfg, 328
 removed_event, 329
 spip, 328
 state, 328
 vt, 329
mmcGetDriverState
 MMC over SPI Driver, 135
mmcInit

MMC over SPI Driver, 128
mmclsWriteProtected
 MMC over SPI Driver, 135
mmcObjectInit
 MMC over SPI Driver, 128
mmcquery_t
 MMC over SPI Driver, 136
mmcSequentialRead
 MMC over SPI Driver, 131
mmcSequentialWrite
 MMC over SPI Driver, 133
mmcStart
 MMC over SPI Driver, 129
mmcStartSequentialRead
 MMC over SPI Driver, 131
mmcStartSequentialWrite
 MMC over SPI Driver, 133
mmcstate_t
 MMC over SPI Driver, 136
mmcStop
 MMC over SPI Driver, 129
mmcStopSequentialRead
 MMC over SPI Driver, 132
mmcStopSequentialWrite
 MMC over SPI Driver, 134
mode
 EXTChannelConfig, 310
 ICUConfig, 322
 PWMChannelConfig, 333
moder
 stm32_gpio_setup_t, 346
MS2RTT
 HAL Driver, 80
mutex
 ADCDriver, 308
 I2CDriver, 319
 SPIDriver, 344
num_channels
 ADCConversionGroup, 304
odr
 stm32_gpio_setup_t, 346
offset
 IOBus, 326
op_mode
 I2CConfig, 317
ospeedr
 stm32_gpio_setup_t, 346
otyper
 stm32_gpio_setup_t, 346
out
 USB Driver, 254
 usb_llc.c, 433
out_cb
 USBEndpointConfig, 362
out_maxsize
 USBEndpointConfig, 362
out_state
 USBEndpointConfig, 363
PAData
 PALConfig, 330
PAL Driver, 136
 _IOBUS_DATA, 144
 _pal_llc_init, 142
 _pal_llc_setgroupmode, 143
 IOBUS_DECL, 144
 iomode_t, 155
 IOPORT1, 152
 IOPORT2, 152
 IOPORT3, 152
 IOPORT4, 152
 IOPORT5, 152
 IOPORT6, 152
 IOPORT7, 152
 IOPORT8, 152
 IOPORT9, 152
 iportid_t, 155
 iportmask_t, 155
 PAL_GROUP_MASK, 144
 PAL_HIGH, 144
 PAL_IOPORTS_WIDTH, 151
 pal_llc_clearport, 154
 pal_llc_init, 152
 pal_llc_readlatch, 153
 pal_llc_readport, 153
 pal_llc_setgroupmode, 154
 pal_llc_setport, 154
 pal_llc_writegroup, 154
 pal_llc_writepad, 155
 pal_llc_writeport, 153
 PAL_LOW, 144
 PAL_MODE_ALTERNATE, 150
 PAL_MODE_INPUT, 143, 151
 PAL_MODE_INPUT_ANALOG, 143, 151
 PAL_MODE_INPUT_PULLDOWN, 143, 151
 PAL_MODE_INPUT_PULLUP, 143, 151
 PAL_MODE_OUTPUT_OPENDRAIN, 144, 151
 PAL_MODE_OUTPUT_PUSH_PULL, 143, 151
 PAL_MODE_RESET, 143, 151
 PAL_MODE_UNCONNECTED, 143, 151
 PAL_PORT_BIT, 144
 PAL_WHOLE_PORT, 152
 palClearPad, 149
 palClearPort, 146
 pallInit, 145
 palReadBus, 141
 palReadGroup, 147
 palReadLatch, 145
 palReadPad, 148
 palReadPort, 145
 palSetBusMode, 142
 palSetGroupMode, 148
 palSetPad, 149
 palSetPadMode, 150
 palSetPort, 146
 palTogglePad, 150

palTogglePort, 147
palWriteBus, 142
palWriteGroup, 147
palWritePad, 148
palWritePort, 146
pal.c, 399
pal.h, 399
PAL_GROUP_MASK
 PAL Driver, 144
PAL_HIGH
 PAL Driver, 144
PAL_IOPORTS_WIDTH
 PAL Driver, 151
pal_lld.c, 401
pal_lld.h, 401
pal_lld_clearport
 PAL Driver, 154
pal_lld_init
 PAL Driver, 152
pal_lld_readlatch
 PAL Driver, 153
pal_lld_readport
 PAL Driver, 153
pal_lld_setgroupmode
 PAL Driver, 154
pal_lld_setport
 PAL Driver, 154
pal_lld_writegroup
 PAL Driver, 154
pal_lld_writepad
 PAL Driver, 155
pal_lld_writeport
 PAL Driver, 153
PAL_LOW
 PAL Driver, 144
PAL_MODE_ALTERNATE
 PAL Driver, 150
PAL_MODE_INPUT
 PAL Driver, 143, 151
PAL_MODE_INPUT_ANALOG
 PAL Driver, 143, 151
PAL_MODE_INPUT_PULLDOWN
 PAL Driver, 143, 151
PAL_MODE_INPUT_PULLUP
 PAL Driver, 143, 151
PAL_MODE_OUTPUT_OPENDRAIN
 PAL Driver, 144, 151
PAL_MODE_OUTPUT_PUSH_PULL
 PAL Driver, 143, 151
PAL_MODE_RESET
 PAL Driver, 143, 151
PAL_MODE_UNCONNECTED
 PAL Driver, 143, 151
PAL_PORT_BIT
 PAL Driver, 144
PAL_WHOLE_PORT
 PAL Driver, 152
palClearPad
 PAL Driver, 149
palClearPort
 PAL Driver, 146
PALConfig, 329
 PADATA, 330
 PBData, 330
 PCData, 330
 PDData, 331
palInit
 PAL Driver, 145
palReadBus
 PAL Driver, 141
palReadGroup
 PAL Driver, 147
palReadLatch
 PAL Driver, 145
palReadPad
 PAL Driver, 148
palReadPort
 PAL Driver, 145
palSetBusMode
 PAL Driver, 142
palSetGroupMode
 PAL Driver, 148
palSetPad
 PAL Driver, 149
palSetPadMode
 PAL Driver, 150
palSetPort
 PAL Driver, 146
palTogglePad
 PAL Driver, 150
palTogglePort
 PAL Driver, 147
palWriteBus
 PAL Driver, 142
palWriteGroup
 PAL Driver, 147
palWritePad
 PAL Driver, 148
palWritePort
 PAL Driver, 146
param
 USBDriver, 359
PBData
 PALConfig, 330
PCData
 PALConfig, 330
PDData
 PALConfig, 331
period
 PWMConfig, 335
 PWMDriver, 338
period_cb
 ICUConfig, 322
pmnext
 USBDriver, 360
portid
 IOBus, 325
pupdr

stm32_gpio_setup_t, 346
PVD_IRQHandler
 HAL Driver, 87
PWM Driver, 155
 PWM_READY, 172
 PWM_STOP, 172
 PWM_UNINIT, 172
 PWM_CHANNELS, 168
 PWM_COMPLEMENTARY_OUTPUT_ACTIVE_HIGH, 169
 PWM_COMPLEMENTARY_OUTPUT_ACTIVE_LOW, 169
 PWM_COMPLEMENTARY_OUTPUT_DISABLED, 169
 PWM_COMPLEMENTARY_OUTPUT_MASK, 168
 PWM_DEGREES_TO_WIDTH, 166
 PWM_FRACTION_TO_WIDTH, 166
pwm_lld_change_period, 171
pwm_lld_disable_channel, 164
pwm_lld_enable_channel, 164
pwm_lld_init, 163
pwm_lld_start, 163
pwm_lld_stop, 163
 PWM_OUTPUT_ACTIVE_HIGH, 165
 PWM_OUTPUT_ACTIVE_LOW, 166
 PWM_OUTPUT_DISABLED, 165
 PWM_OUTPUT_MASK, 165
 PWM_PERCENTAGE_TO_WIDTH, 166
pwmcallback_t, 171
pwmChangePeriod, 161
pwmChangePeriodl, 167
pwmchannel_t, 172
pwmcnt_t, 172
PWMD1, 164
PWMD2, 165
PWMD3, 165
PWMD4, 165
PWMD5, 165
PWMD8, 165
pwmDisableChannel, 162
pwmDisableChannell, 168
PWMDriver, 171
pwmEnableChannel, 161
pwmEnableChannell, 167
pwmInit, 159
pwemode_t, 172
pwmObjectInit, 160
pwmStart, 160
pwmstate_t, 172
pwmStop, 160
 STM32_PWM_TIM1_IRQ_PRIORITY, 170
 STM32_PWM_TIM2_IRQ_PRIORITY, 170
 STM32_PWM_TIM3_IRQ_PRIORITY, 170
 STM32_PWM_TIM4_IRQ_PRIORITY, 171
 STM32_PWM_TIM5_IRQ_PRIORITY, 171
 STM32_PWM_TIM8_IRQ_PRIORITY, 171
 STM32_PWM_USE_ADVANCED, 169
 STM32_PWM_USE_TIM1, 169
 STM32_PWM_USE_TIM2, 169
 STM32_PWM_USE_TIM3, 170
 STM32_PWM_USE_TIM4, 170
 STM32_PWM_USE_TIM5, 170
 STM32_PWM_USE_TIM8, 170
pwm.c, 403
pwm.h, 404
 PWM_READY
 PWM Driver, 172
 PWM_STOP
 PWM_UNINIT
 PWM_CHANNELS
 PWM_COMPLEMENTARY_OUTPUT_ACTIVE_HIGH
 PWM Driver, 168
 PWM_COMPLEMENTARY_OUTPUT_ACTIVE_LOW
 PWM Driver, 169
 PWM_COMPLEMENTARY_OUTPUT_DISABLED
 PWM Driver, 169
 PWM_COMPLEMENTARY_OUTPUT_MASK
 PWM Driver, 168
 PWM_DEGREES_TO_WIDTH
 PWM Driver, 166
 PWM_FRACTION_TO_WIDTH
 PWM Driver, 166
pwm_lld.c, 405
pwm_lld.h, 406
pwm_lld_change_period
 PWM Driver, 171
pwm_lld_disable_channel
 PWM Driver, 164
pwm_lld_enable_channel
 PWM Driver, 164
pwm_lld_init
 PWM Driver, 163
pwm_lld_start
 PWM Driver, 163
pwm_lld_stop
 PWM Driver, 163
 PWM_OUTPUT_ACTIVE_HIGH
 PWM Driver, 165
 PWM_OUTPUT_ACTIVE_LOW
 PWM Driver, 166
 PWM_OUTPUT_DISABLED
 PWM Driver, 165
 PWM_OUTPUT_MASK
 PWM Driver, 165
 PWM_PERCENTAGE_TO_WIDTH
 PWM Driver, 166
pwmcallback_t
 PWM Driver, 171
pwmChangePeriod
 PWM Driver, 161
pwmChangePeriodl
 PWM Driver, 167
pwmchannel_t
 PWM Driver, 172
PWMChannelConfig, 331
 callback, 333

mode, 333
pwmcnt_t
 PWM Driver, 172
PWMConfig, 333
 bdtr, 335
 callback, 335
 channels, 335
 cr2, 335
 frequency, 335
 period, 335
PWMD1
 PWM Driver, 164
PWMD2
 PWM Driver, 165
PWMD3
 PWM Driver, 165
PWMD4
 PWM Driver, 165
PWMD5
 PWM Driver, 165
PWMD8
 PWM Driver, 165
pwmDisableChannel
 PWM Driver, 162
pwmDisableChannell
 PWM Driver, 168
PWMDriver, 336
 clock, 338
 config, 338
 period, 338
 PWM Driver, 171
 state, 338
 tim, 338
pwmEnableChannel
 PWM Driver, 161
pwmEnableChannell
 PWM Driver, 167
pwmInit
 PWM Driver, 159
pwemode_t
 PWM Driver, 172
pwmObjectInit
 PWM Driver, 160
pwmStart
 PWM Driver, 160
pwmstate_t
 PWM Driver, 172
pwmStop
 PWM Driver, 160

RCC_IRQHandler
 HAL Driver, 87
rccDisableADC1
 STM32L1xx RCC Support, 293
rccDisableAHB
 STM32L1xx RCC Support, 292
rccDisableAPB1
 STM32L1xx RCC Support, 290
rccDisableAPB2
 STM32L1xx RCC Support, 296
rccDisableDMA1
 STM32L1xx RCC Support, 293
rccDisableI2C1
 STM32L1xx RCC Support, 295
rccDisableI2C2
 STM32L1xx RCC Support, 295
rccDisablePWRInterface
 STM32L1xx RCC Support, 294
rccDisableSPI1
 STM32L1xx RCC Support, 296
rccDisableSPI2
 STM32L1xx RCC Support, 296
rccDisableTIM2
 STM32L1xx RCC Support, 297
rccDisableTIM3
 STM32L1xx RCC Support, 297
rccDisableTIM4
 STM32L1xx RCC Support, 298
rccDisableUSART1
 STM32L1xx RCC Support, 299
rccDisableUSART2
 STM32L1xx RCC Support, 299
rccDisableUSART3
 STM32L1xx RCC Support, 300
rccDisableUSB
 STM32L1xx RCC Support, 300
rccEnableADC1
 STM32L1xx RCC Support, 293
rccEnableAHB
 STM32L1xx RCC Support, 292
rccEnableAPB1
 STM32L1xx RCC Support, 290
rccEnableAPB2
 STM32L1xx RCC Support, 291
rccEnableDMA1
 STM32L1xx RCC Support, 293
rccEnableI2C1
 STM32L1xx RCC Support, 294
rccEnableI2C2
 STM32L1xx RCC Support, 295
rccEnablePWRInterface
 STM32L1xx RCC Support, 294
rccEnableSPI1
 STM32L1xx RCC Support, 296
rccEnableSPI2
 STM32L1xx RCC Support, 296
rccEnableTIM2
 STM32L1xx RCC Support, 297
rccEnableTIM3
 STM32L1xx RCC Support, 297
rccEnableTIM4
 STM32L1xx RCC Support, 298
rccEnableUSART1
 STM32L1xx RCC Support, 298
rccEnableUSART2
 STM32L1xx RCC Support, 299
rccEnableUSART3
 STM32L1xx RCC Support, 300

rccEnableUSB
 STM32L1xx RCC Support, 300

rccResetADC1
 STM32L1xx RCC Support, 293

rccResetAHB
 STM32L1xx RCC Support, 292

rccResetAPB1
 STM32L1xx RCC Support, 290

rccResetAPB2
 STM32L1xx RCC Support, 291

rccResetDMA1
 STM32L1xx RCC Support, 294

rccResetI2C1
 STM32L1xx RCC Support, 295

rccResetI2C2
 STM32L1xx RCC Support, 295

rccResetPWRInterface
 STM32L1xx RCC Support, 294

rccResetSPI1
 STM32L1xx RCC Support, 296

rccResetSPI2
 STM32L1xx RCC Support, 297

rccResetTIM2
 STM32L1xx RCC Support, 297

rccResetTIM3
 STM32L1xx RCC Support, 298

rccResetTIM4
 STM32L1xx RCC Support, 298

rccResetUSART1
 STM32L1xx RCC Support, 299

rccResetUSART2
 STM32L1xx RCC Support, 299

rccResetUSART3
 STM32L1xx RCC Support, 300

rccResetUSB
 STM32L1xx RCC Support, 301

receiving
 USBDriver, 359

removed_event
 MMCDriver, 329

requests_hook_cb
 USBConfig, 357

RTC_Alarm_IRQHandler
 HAL Driver, 90

RTC_WKUP_IRQHandler
 HAL Driver, 87

RXADDR0
 stm32_usb_descriptor_t, 347

rxbuf
 UARTDriver, 354
 USBOutEndpointState, 364

rxchar_cb
 UARTConfig, 351

rxcnt
 USBOutEndpointState, 364

RXCOUNT0
 stm32_usb_descriptor_t, 348

RXCOUNT1
 stm32_usb_descriptor_t, 348

rxdmamode
 SPIDriver, 344

rxend_cb
 UARTConfig, 351

rxerr_cb
 UARTConfig, 351

rxpkts
 USBOutEndpointState, 364

rxsize
 USBOutEndpointState, 364

rxstate
 UARTDriver, 354

S2RTT
 HAL Driver, 79

samples
 ADCDriver, 308

sc_cr1
 SerialConfig, 339

sc_cr2
 SerialConfig, 339

sc_cr3
 SerialConfig, 339

sc_speed
 SerialConfig, 339

SD1
 Serial Driver, 180

SD2
 Serial Driver, 180

SD3
 Serial Driver, 181

SD4
 Serial Driver, 181

SD5
 Serial Driver, 181

SD6
 Serial Driver, 181

SD_READY
 Serial Driver, 188

SD_STOP
 Serial Driver, 188

SD_UNINIT
 Serial Driver, 188

SD_BREAK_DETECTED
 Serial Driver, 181

SD_FRAMING_ERROR
 Serial Driver, 181

sd_lld_init
 Serial Driver, 179

sd_lld_start
 Serial Driver, 180

sd_lld_stop
 Serial Driver, 180

SD_NOISE_ERROR
 Serial Driver, 181

SD_OVERRUN_ERROR
 Serial Driver, 181

SD_PARITY_ERROR
 Serial Driver, 181

sdAsynchronousRead
 Serial Driver, 185
sdAsynchronousWrite
 Serial Driver, 184
SDC_INIT_RETRY
 Configuration, 14
SDC_MMC_SUPPORT
 Configuration, 14
SDC_NICE_WAITING
 Configuration, 15
sdGet
 Serial Driver, 183
sdGetTimeout
 Serial Driver, 183
sdGetWouldBlock
 Serial Driver, 182
sdIncomingDataL
 Serial Driver, 178
sdInit
 Serial Driver, 176
sdObjectInit
 Serial Driver, 176
sdPut
 Serial Driver, 182
sdPutTimeout
 Serial Driver, 183
sdPutWouldBlock
 Serial Driver, 182
sdRead
 Serial Driver, 184
sdReadTimeout
 Serial Driver, 185
sdRequestDataL
 Serial Driver, 178
sdStart
 Serial Driver, 177
sdstate_t
 Serial Driver, 188
sdStop
 Serial Driver, 177
sdWrite
 Serial Driver, 184
sdWriteTimeout
 Serial Driver, 184
selfindex
 stm32_dma_stream_t, 345
Serial Driver, 172
 _serial_driver_data, 187
 _serial_driver_methods, 182
 CH_IRQ_HANDLER, 179
 SD1, 180
 SD2, 180
 SD3, 181
 SD4, 181
 SD5, 181
 SD6, 181
 SD_READY, 188
 SD_STOP, 188
 SD_UNINIT, 188
 SD_BREAK_DETECTED, 181
 SD_FRAMING_ERROR, 181
 sd_lld_init, 179
 sd_lld_start, 180
 sd_lld_stop, 180
 SD_NOISE_ERROR, 181
 SD_OVERRUN_ERROR, 181
 SD_PARITY_ERROR, 181
 sdAsynchronousRead, 185
 sdAsynchronousWrite, 184
 sdGet, 183
 sdGetTimeout, 183
 sdGetWouldBlock, 182
 sdIncomingDataL, 178
 sdInit, 176
 sdObjectInit, 176
 sdPut, 182
 sdPutTimeout, 183
 sdPutWouldBlock, 182
 sdRead, 184
 sdReadTimeout, 185
 sdRequestDataL, 178
 sdStart, 177
 sdstate_t, 188
 sdStop, 177
 sdWrite, 184
 sdWriteTimeout, 184
 SERIAL_BUFFERS_SIZE, 181
 SERIAL_DEFAULT_BITRATE, 181
 SerialDriver, 188
 STM32_SERIAL_UART4_PRIORITY, 187
 STM32_SERIAL_UART5_PRIORITY, 187
 STM32_SERIAL_USART1_PRIORITY, 186
 STM32_SERIAL_USART2_PRIORITY, 186
 STM32_SERIAL_USART3_PRIORITY, 186
 STM32_SERIAL_USART6_PRIORITY, 187
 STM32_SERIAL_USE_UART4, 186
 STM32_SERIAL_USE_UART5, 186
 STM32_SERIAL_USE_USART1, 185
 STM32_SERIAL_USE_USART2, 185
 STM32_SERIAL_USE_USART3, 186
 STM32_SERIAL_USE_USART6, 186
 USART_CR2_STOP0P5_BITS, 187
 USART_CR2_STOP1_BITS, 187
 USART_CR2_STOP1P5_BITS, 187
 USART_CR2_STOP2_BITS, 187
serial.c, 407
serial.h, 408
SERIAL_BUFFERS_SIZE
 Configuration, 15
 Serial Driver, 181
SERIAL_DEFAULT_BITRATE
 Configuration, 15
 Serial Driver, 181
serial_lld.c, 410
serial_lld.h, 410
SERIAL_USB_BUFFERS_SIZE
 Configuration, 15
SerialConfig, 338

sc_cr1, 339
sc_cr2, 339
sc_cr3, 339
sc_speed, 339
SerialDriver, 339
 Serial Driver, 188
 vmt, 340
SerialDriverVMT, 340
setup
 USBDriver, 360
setup_cb
 USBEndpointConfig, 362
smpr1
 ADCConversionGroup, 305
smpr2
 ADCConversionGroup, 305
smpr3
 ADCConversionGroup, 305
sof_cb
 USBConfig, 357
speed
 UARTConfig, 351
spi
 SPIDriver, 344
SPI Driver, 188
 _spi_isr_code, 206
 _spi_wait_s, 205
 _spi_wakeup_isr, 206
 SPI_ACTIVE, 210
 SPI_COMPLETE, 210
 SPI_READY, 210
 SPI_STOP, 210
 SPI_UNINIT, 209
 spi_lld_exchange, 200
 spi_lld_ignore, 200
 spi_lld_init, 198
 spi_lld_polled_exchange, 201
 spi_lld_receive, 201
 spi_lld_select, 199
 spi_lld_send, 200
 spi_lld_start, 198
 spi_lld_stop, 199
 spi_lld_unselect, 199
 SPI_USE_MUTUAL_EXCLUSION, 202
 SPI_USE_WAIT, 202
 spiAcquireBus, 197
 spicallback_t, 209
 SPID1, 202
 SPID2, 202
 SPID3, 202
 SPIDriver, 209
 spiExchange, 196
 spilgnore, 196
 spilinit, 192
 spiObjectInit, 192
 spiPolledExchange, 205
 spiReceive, 197
 spiReleaseBus, 197
 spiSelect, 193
 spiSelectl, 202
 spiSend, 196
 spiStart, 192
 spiStartExchange, 194
 spiStartExchangel, 203
 spiStartIgnore, 194
 spiStartIgnorel, 203
 spiStartReceive, 195
 spiStartReceiveL, 204
 spiStartSend, 195
 spiStartSendL, 204
 spistate_t, 209
 spiStop, 193
 spiUnselect, 194
 spiUnselectl, 202
 STM32_SPI_DMA_ERROR_HOOK, 208
 STM32_SPI_SPI1_DMA_PRIORITY, 208
 STM32_SPI_SPI1_IRQ_PRIORITY, 207
 STM32_SPI_SPI1_RX_DMA_STREAM, 208
 STM32_SPI_SPI1_TX_DMA_STREAM, 208
 STM32_SPI_SPI2_DMA_PRIORITY, 208
 STM32_SPI_SPI2_IRQ_PRIORITY, 207
 STM32_SPI_SPI2_RX_DMA_STREAM, 208
 STM32_SPI_SPI2_TX_DMA_STREAM, 209
 STM32_SPI_SPI3_DMA_PRIORITY, 208
 STM32_SPI_SPI3_IRQ_PRIORITY, 208
 STM32_SPI_SPI3_RX_DMA_STREAM, 209
 STM32_SPI_SPI3_TX_DMA_STREAM, 209
 STM32_SPI_USE_SPI1, 207
 STM32_SPI_USE_SPI2, 207
 STM32_SPI_USE_SPI3, 207
 spi.c, 412
 spi.h, 412
 SPI1_IRQHandler
 HAL Driver, 90
 SPI2_IRQHandler
 HAL Driver, 90
 SPI_ACTIVE
 SPI Driver, 210
 SPI_COMPLETE
 SPI Driver, 210
 SPI_READY
 SPI Driver, 210
 SPI_STOP
 SPI Driver, 210
 SPI_UNINIT
 SPI Driver, 209
 spi_lld.c, 414
 spi_lld.h, 415
 spi_lld_exchange
 SPI Driver, 200
 spi_lld_ignore
 SPI Driver, 200
 spi_lld_init
 SPI Driver, 198
 spi_lld_polled_exchange
 SPI Driver, 201
 spi_lld_receive
 SPI Driver, 201

spi_lld_select
 SPI Driver, 199
spi_lld_send
 SPI Driver, 200
spi_lld_start
 SPI Driver, 198
spi_lld_stop
 SPI Driver, 199
spi_lld_unselect
 SPI Driver, 199
SPI_USE_MUTUAL_EXCLUSION
 Configuration, 15
 SPI Driver, 202
SPI_USE_WAIT
 Configuration, 15
 SPI Driver, 202
spiAcquireBus
 SPI Driver, 197
spicallback_t
 SPI Driver, 209
SPIConfig, 340
 cr1, 342
 end_cb, 342
 sspad, 342
 ssport, 342
SPID1
 SPI Driver, 202
SPID2
 SPI Driver, 202
SPID3
 SPI Driver, 202
SPIDriver, 342
 config, 344
 dmars, 344
 dmatx, 344
 mutex, 344
 rxdmamode, 344
 spi, 344
 SPI Driver, 209
 state, 344
 thread, 344
 txdmamode, 344
spiExchange
 SPI Driver, 196
spilgnore
 SPI Driver, 196
spilinit
 SPI Driver, 192
spiObjectInit
 SPI Driver, 192
spip
 MMCDriver, 328
spiPolledExchange
 SPI Driver, 205
spiReceive
 SPI Driver, 197
spiReleaseBus
 SPI Driver, 197
spiSelect
 SPI Driver, 193
spiSelectl
 SPI Driver, 202
spiSend
 SPI Driver, 196
spiStart
 SPI Driver, 192
spiStartExchange
 SPI Driver, 194
spiStartExchangel
 SPI Driver, 203
spiStartIgnore
 SPI Driver, 194
spiStartIgnoreL
 SPI Driver, 203
spiStartReceive
 SPI Driver, 195
spiStartReceiveL
 SPI Driver, 204
spiStartSend
 SPI Driver, 195
spiStartSendL
 SPI Driver, 204
spistate_t
 SPI Driver, 209
spiStop
 SPI Driver, 193
spiUnselect
 SPI Driver, 194
spiUnselectl
 SPI Driver, 202
sqr1
 ADCConversionGroup, 305
sqr2
 ADCConversionGroup, 305
sqr3
 ADCConversionGroup, 305
sqr4
 ADCConversionGroup, 305
sqr5
 ADCConversionGroup, 305
sspad
 SPIConfig, 342
ssport
 SPIConfig, 342
start
 TimeMeasurement, 349
state
 ADCDriver, 308
 EXTDriver, 312
 GPTDriver, 317
 I2CDriver, 319
 ICUDriver, 324
 MMCDriver, 328
 PWMDriver, 338
 SPIDriver, 344
 UARTDriver, 354
 USBDriver, 359
status

USBDriver, 360
stm32.h, 416
STM32_0WS_THRESHOLD
 HAL Driver, 94
STM32_ACTIVATE_PLL
 HAL Driver, 94
STM32_ADC_ADC1_DMA_IRQ_PRIORITY
 ADC Driver, 37
STM32_ADC_ADC1_DMA_PRIORITY
 ADC Driver, 37
STM32_ADC_ADCPRE
 ADC Driver, 37
STM32_ADC_CLOCK_ENABLED
 HAL Driver, 92
STM32_ADC_IRQ_PRIORITY
 ADC Driver, 37
STM32_ADC_USE_ADC1
 ADC Driver, 37
STM32_ADCCLK
 HAL Driver, 95
stm32_clock_init
 HAL Driver, 79
stm32_dma.c, 417
stm32_dma.h, 418
STM32_DMA1_STREAMS_MASK
 STM32L1xx DMA Support, 279
STM32_DMA2_STREAMS_MASK
 STM32L1xx DMA Support, 280
STM32_DMA_CCR_RESET_VALUE
 STM32L1xx DMA Support, 280
STM32_DMA_CR_CHSEL
 STM32L1xx DMA Support, 281
STM32_DMA_CR_CHSEL_MASK
 STM32L1xx DMA Support, 281
STM32_DMA_CR_DMEIE
 STM32L1xx DMA Support, 281
STM32_DMA_GETCHANNEL
 STM32L1xx DMA Support, 280
STM32_DMA_IS_VALID_ID
 STM32L1xx DMA Support, 281
STM32_DMA_ISR_MASK
 STM32L1xx DMA Support, 280
STM32_DMA_REQUIRED
 I2C Driver, 112
STM32_DMA_STREAM
 STM32L1xx DMA Support, 281
STM32_DMA_STREAM_ID
 STM32L1xx DMA Support, 280
STM32_DMA_STREAM_ID_MSK
 STM32L1xx DMA Support, 280
stm32_dma_stream_t, 345
 channel, 345
 ifcr, 345
 ishift, 345
 selfindex, 345
 vector, 345
STM32_DMA_STREAMS
 STM32L1xx DMA Support, 280
stm32_dmaisr_t
 STM32L1xx DMA Support, 286
STM32_EXT_EXTI0_IRQ_PRIORITY
 EXT Driver, 53
STM32_EXT_EXTI10_15_IRQ_PRIORITY
 EXT Driver, 53
STM32_EXT_EXTI16_IRQ_PRIORITY
 EXT Driver, 53
STM32_EXT_EXTI17_IRQ_PRIORITY
 EXT Driver, 53
STM32_EXT_EXTI18_IRQ_PRIORITY
 EXT Driver, 53
STM32_EXT_EXTI19_IRQ_PRIORITY
 EXT Driver, 53
STM32_EXT_EXTI1_IRQ_PRIORITY
 EXT Driver, 53
STM32_EXT_EXTI20_IRQ_PRIORITY
 EXT Driver, 54
STM32_EXT_EXTI21_IRQ_PRIORITY
 EXT Driver, 54
STM32_EXT_EXTI22_IRQ_PRIORITY
 EXT Driver, 54
STM32_EXT_EXTI2_IRQ_PRIORITY
 EXT Driver, 53
STM32_EXT_EXTI3_IRQ_PRIORITY
 EXT Driver, 53
STM32_EXT_EXTI4_IRQ_PRIORITY
 EXT Driver, 53
STM32_EXT_EXTI5_9_IRQ_PRIORITY
 EXT Driver, 53
STM32_FLASHBITS1
 HAL Driver, 95
stm32_gpio_setup_t, 346
 afrh, 346
 afrl, 346
 moder, 346
 odr, 346
 ospeedr, 346
 otyper, 346
 pupdr, 346
STM32_GPT_TIM1_IRQ_PRIORITY
 GPT Driver, 66
STM32_GPT_TIM2_IRQ_PRIORITY
 GPT Driver, 66
STM32_GPT_TIM3_IRQ_PRIORITY
 GPT Driver, 66
STM32_GPT_TIM4_IRQ_PRIORITY
 GPT Driver, 66
STM32_GPT_TIM5_IRQ_PRIORITY
 GPT Driver, 66
STM32_GPT_TIM8_IRQ_PRIORITY
 GPT Driver, 67
STM32_GPT_USE_TIM1
 GPT Driver, 65
STM32_GPT_USE_TIM2
 GPT Driver, 65
STM32_GPT_USE_TIM3
 GPT Driver, 65
STM32_GPT_USE_TIM4
 GPT Driver, 66

STM32_GPT_USE_TIM5
 GPT Driver, 66
STM32_GPT_USE_TIM8
 GPT Driver, 66
STM32_HCLK
 HAL Driver, 95
STM32_HPRE
 HAL Driver, 92
STM32_HPRE_DIV1
 HAL Driver, 83
STM32_HPRE_DIV128
 HAL Driver, 84
STM32_HPRE_DIV16
 HAL Driver, 83
STM32_HPRE_DIV2
 HAL Driver, 83
STM32_HPRE_DIV256
 HAL Driver, 84
STM32_HPRE_DIV4
 HAL Driver, 83
STM32_HPRE_DIV512
 HAL Driver, 84
STM32_HPRE_DIV64
 HAL Driver, 84
STM32_HPRE_DIV8
 HAL Driver, 83
STM32_HSE_ENABLED
 HAL Driver, 91
STM32_HSECLK_MAX
 HAL Driver, 93
STM32_HSEDIVCLK
 HAL Driver, 95
STM32_HSI_AVAILABLE
 HAL Driver, 94
STM32_HSI_ENABLED
 HAL Driver, 91
STM32_HSICLK
 HAL Driver, 81
STM32_I2C_DMA_ERROR_HOOK
 I2C Driver, 111
STM32_I2C_I2C1_DMA_PRIORITY
 I2C Driver, 110
STM32_I2C_I2C1_IRQ_PRIORITY
 I2C Driver, 110
STM32_I2C_I2C1_RX_DMA_STREAM
 I2C Driver, 111
STM32_I2C_I2C1_TX_DMA_STREAM
 I2C Driver, 111
STM32_I2C_I2C2_DMA_PRIORITY
 I2C Driver, 110
STM32_I2C_I2C2_IRQ_PRIORITY
 I2C Driver, 110
STM32_I2C_I2C2_RX_DMA_STREAM
 I2C Driver, 111
STM32_I2C_I2C2_TX_DMA_STREAM
 I2C Driver, 111
STM32_I2C_I2C3_DMA_PRIORITY
 I2C Driver, 111
STM32_I2C_I2C3_IRQ_PRIORITY
 I2C Driver, 110
I2C Driver, 110
STM32_I2C_I2C3_RX_DMA_STREAM
 I2C Driver, 112
STM32_I2C_I2C3_TX_DMA_STREAM
 I2C Driver, 112
STM32_I2C_USE_I2C1
 I2C Driver, 109
STM32_I2C_USE_I2C2
 I2C Driver, 110
STM32_I2C_USE_I2C3
 I2C Driver, 110
STM32_ICU_TIM1_IRQ_PRIORITY
 ICU Driver, 124
STM32_ICU_TIM2_IRQ_PRIORITY
 ICU Driver, 124
STM32_ICU_TIM3_IRQ_PRIORITY
 ICU Driver, 124
STM32_ICU_TIM4_IRQ_PRIORITY
 ICU Driver, 124
STM32_ICU_TIM5_IRQ_PRIORITY
 ICU Driver, 124
STM32_ICU_TIM8_IRQ_PRIORITY
 ICU Driver, 124
STM32_ICU_USE_TIM1
 ICU Driver, 123
STM32_ICU_USE_TIM2
 ICU Driver, 123
STM32_ICU_USE_TIM3
 ICU Driver, 123
STM32_ICU_USE_TIM4
 ICU Driver, 123
STM32_ICU_USE_TIM5
 ICU Driver, 124
STM32_ICU_USE_TIM8
 ICU Driver, 124
STM32_LSE_ENABLED
 HAL Driver, 91
STM32_LSI_ENABLED
 HAL Driver, 91
STM32_LSICLK
 HAL Driver, 81
STM32_MCOPRE
 HAL Driver, 93
STM32_MCOPRE_DIV1
 HAL Driver, 86
STM32_MCOPRE_DIV16
 HAL Driver, 86
STM32_MCOPRE_DIV2
 HAL Driver, 86
STM32_MCOPRE_DIV4
 HAL Driver, 86
STM32_MCOPRE_DIV8
 HAL Driver, 86
STM32_MCOSEL
 HAL Driver, 93
STM32_MCOSEL_HSE
 HAL Driver, 85
STM32_MCOSEL_HSI
 HAL Driver, 85

STM32_MCOSEL_LSE
 HAL Driver, 85
STM32_MCOSEL_LSI
 HAL Driver, 85
STM32_MCOSEL_MSI
 HAL Driver, 85
STM32_MCOSEL_NOCLOCK
 HAL Driver, 85
STM32_MCOSEL_PLL
 HAL Driver, 85
STM32_MCOSEL_SYSCLK
 HAL Driver, 85
STM32_MSICLK
 HAL Driver, 94
STM32_MSIRANGE
 HAL Driver, 92
STM32_MSIRANGE_128K
 HAL Driver, 86
STM32_MSIRANGE_1M
 HAL Driver, 86
STM32_MSIRANGE_256K
 HAL Driver, 86
STM32_MSIRANGE_2M
 HAL Driver, 86
STM32_MSIRANGE_4M
 HAL Driver, 87
STM32_MSIRANGE_512K
 HAL Driver, 86
STM32_MSIRANGE_64K
 HAL Driver, 86
STM32_MSIRANGE_MASK
 HAL Driver, 86
STM32_NO_INIT
 HAL Driver, 91
STM32_PCLK1
 HAL Driver, 95
STM32_PCLK1_MAX
 HAL Driver, 93
STM32_PCLK2
 HAL Driver, 95
STM32_PCLK2_MAX
 HAL Driver, 94
STM32_PLLCLKIN
 HAL Driver, 94
STM32_PLLCLKOUT
 HAL Driver, 94
STM32_PLLDIV
 HAL Driver, 94
STM32_PLLDIV_VALUE
 HAL Driver, 92
STM32_PLLMUL
 HAL Driver, 94
STM32_PLLMUL_VALUE
 HAL Driver, 92
STM32_PLLSRC
 HAL Driver, 92
STM32_PLLSRC_HSE
 HAL Driver, 85
STM32_PLLSRC_HSI
 HAL Driver, 85
STM32_PLLVCO
 HAL Driver, 94
STM32_PLLVCO_MAX
 HAL Driver, 93
STM32_PLLVCO_MIN
 HAL Driver, 93
STM32_PLS
 HAL Driver, 91
STM32_PLS_LEV0
 HAL Driver, 82
STM32_PLS_LEV1
 HAL Driver, 82
STM32_PLS_LEV2
 HAL Driver, 82
STM32_PLS_LEV3
 HAL Driver, 82
STM32_PLS_LEV4
 HAL Driver, 82
STM32_PLS_LEV5
 HAL Driver, 82
STM32_PLS_LEV6
 HAL Driver, 82
STM32_PLS_LEV7
 HAL Driver, 82
STM32_PLS_MASK
 HAL Driver, 82
STM32_PPREG1
 HAL Driver, 93
STM32_PPREG1_DIV1
 HAL Driver, 84
STM32_PPREG1_DIV16
 HAL Driver, 84
STM32_PPREG1_DIV2
 HAL Driver, 84
STM32_PPREG1_DIV4
 HAL Driver, 84
STM32_PPREG1_DIV8
 HAL Driver, 84
STM32_PPREG2
 HAL Driver, 93
STM32_PPREG2_DIV1
 HAL Driver, 84
STM32_PPREG2_DIV16
 HAL Driver, 85
STM32_PPREG2_DIV2
 HAL Driver, 84
STM32_PPREG2_DIV4
 HAL Driver, 84
STM32_PPREG2_DIV8
 HAL Driver, 85
STM32_PVD_ENABLE
 HAL Driver, 91
STM32_PWM_TIM1_IRQ_PRIORITY
 PWM Driver, 170
STM32_PWM_TIM2_IRQ_PRIORITY
 PWM Driver, 170
STM32_PWM_TIM3_IRQ_PRIORITY
 PWM Driver, 170

STM32_PWM_TIM4_IRQ_PRIORITY
 PWM Driver, 171

STM32_PWM_TIM5_IRQ_PRIORITY
 PWM Driver, 171

STM32_PWM_TIM8_IRQ_PRIORITY
 PWM Driver, 171

STM32_PWM_USE_ADVANCED
 PWM Driver, 169

STM32_PWM_USE_TIM1
 PWM Driver, 169

STM32_PWM_USE_TIM2
 PWM Driver, 169

STM32_PWM_USE_TIM3
 PWM Driver, 170

STM32_PWM_USE_TIM4
 PWM Driver, 170

STM32_PWM_USE_TIM5
 PWM Driver, 170

STM32_PWM_USE_TIM8
 PWM Driver, 170

stm32_rcc.h, 420

STM32_RTCPRE
 HAL Driver, 93

STM32_RTCPRE_DIV16
 HAL Driver, 83

STM32_RTCPRE_DIV2
 HAL Driver, 82

STM32_RTCPRE_DIV4
 HAL Driver, 83

STM32_RTCPRE_DIV8
 HAL Driver, 83

STM32_RTCPRE_MASK
 HAL Driver, 82

STM32_RTCSEL
 HAL Driver, 93

STM32_RTCSEL_HSEDIV
 HAL Driver, 87

STM32_RTCSEL_LSE
 HAL Driver, 87

STM32_RTCSEL_LSI
 HAL Driver, 87

STM32_RTCSEL_NOCLOCK
 HAL Driver, 87

STM32_SERIAL_UART4_PRIORITY
 Serial Driver, 187

STM32_SERIAL_UART5_PRIORITY
 Serial Driver, 187

STM32_SERIAL_USART1_PRIORITY
 Serial Driver, 186

STM32_SERIAL_USART2_PRIORITY
 Serial Driver, 186

STM32_SERIAL_USART3_PRIORITY
 Serial Driver, 186

STM32_SERIAL_USART6_PRIORITY
 Serial Driver, 187

STM32_SERIAL_USE_UART4
 Serial Driver, 186

STM32_SERIAL_USE_UART5
 Serial Driver, 186

STM32_SERIAL_USE_USART1
 Serial Driver, 185

STM32_SERIAL_USE_USART2
 Serial Driver, 185

STM32_SERIAL_USE_USART3
 Serial Driver, 186

STM32_SERIAL_USE_USART6
 Serial Driver, 186

STM32_SPI_DMA_ERROR_HOOK
 SPI Driver, 208

STM32_SPI_SPI1_DMA_PRIORITY
 SPI Driver, 208

STM32_SPI_SPI1_IRQ_PRIORITY
 SPI Driver, 207

STM32_SPI_SPI1_RX_DMA_STREAM
 SPI Driver, 208

STM32_SPI_SPI1_TX_DMA_STREAM
 SPI Driver, 208

STM32_SPI_SPI2_DMA_PRIORITY
 SPI Driver, 208

STM32_SPI_SPI2_IRQ_PRIORITY
 SPI Driver, 207

STM32_SPI_SPI2_RX_DMA_STREAM
 SPI Driver, 208

STM32_SPI_SPI2_TX_DMA_STREAM
 SPI Driver, 209

STM32_SPI_SPI3_DMA_PRIORITY
 SPI Driver, 208

STM32_SPI_SPI3_IRQ_PRIORITY
 SPI Driver, 208

STM32_SPI_SPI3_RX_DMA_STREAM
 SPI Driver, 209

STM32_SPI_SPI3_TX_DMA_STREAM
 SPI Driver, 209

STM32_SPI_USE_SPI1
 SPI Driver, 207

STM32_SPI_USE_SPI2
 SPI Driver, 207

STM32_SW
 HAL Driver, 92

STM32_SW_HSE
 HAL Driver, 83

STM32_SW_HSI
 HAL Driver, 83

STM32_SW_MSI
 HAL Driver, 83

STM32_SW_PLL
 HAL Driver, 83

STM32_SYSCLK
 HAL Driver, 94

STM32_SYSCLK_MAX
 HAL Driver, 93

stm32_tim_t, 347

STM32_TIMCLK1
 HAL Driver, 95

STM32_TIMCLK2
 HAL Driver, 95

STM32_UART_DMA_ERROR_HOOK
 UART Driver, 228

STM32_UART_USART1_DMA_PRIORITY
 UART Driver, 228

STM32_UART_USART1_IRQ_PRIORITY
 UART Driver, 227

STM32_UART_USART1_RX_DMA_STREAM
 UART Driver, 228

STM32_UART_USART1_TX_DMA_STREAM
 UART Driver, 228

STM32_UART_USART2_DMA_PRIORITY
 UART Driver, 228

STM32_UART_USART2_IRQ_PRIORITY
 UART Driver, 227

STM32_UART_USART2_RX_DMA_STREAM
 UART Driver, 228

STM32_UART_USART2_TX_DMA_STREAM
 UART Driver, 229

STM32_UART_USART3_DMA_PRIORITY
 UART Driver, 228

STM32_UART_USART3_IRQ_PRIORITY
 UART Driver, 227

STM32_UART_USART3_RX_DMA_STREAM
 UART Driver, 229

STM32_UART_USART3_TX_DMA_STREAM
 UART Driver, 229

STM32_UART_USE_USART1
 UART Driver, 227

STM32_UART_USE_USART2
 UART Driver, 227

STM32_UART_USE_USART3
 UART Driver, 227

STM32_USB
 USB Driver, 263

stm32_usb.h, 422

STM32_USB_BASE
 USB Driver, 263

STM32_USB_CLOCK_ENABLED
 HAL Driver, 92

stm32_usb_descriptor_t, 347

- RXADDR0, 347
- RXCOUNT0, 348
- RXCOUNT1, 348
- TXADDR0, 347
- TXCOUNT0, 347
- TXCOUNT1, 347

STM32_USB_LOW_POWER_ON_SUSPEND
 USB Driver, 264

stm32_usb_t, 348

- EPR, 348

STM32_USB_USB1_HP_IRQ_PRIORITY
 USB Driver, 264

STM32_USB_USB1_LP_IRQ_PRIORITY
 USB Driver, 264

STM32_USB_USE_USB1
 USB Driver, 264

STM32_USBCLK
 HAL Driver, 95

STM32_USBRAM
 USB Driver, 263

STM32_USBRAM_BASE
 USB Driver, 263

STM32_VOS
 HAL Driver, 91

STM32_VOS_1P2
 HAL Driver, 82

STM32_VOS_1P5
 HAL Driver, 81

STM32_VOS_1P8
 HAL Driver, 81

STM32_VOS_MASK
 HAL Driver, 81

STM32L1xx ADC Support, 268

STM32L1xx DMA Support, 273

- _stm32_dma_streams, 279
- CH_IRQ_HANDLER, 277, 278
- dmalinit, 278
- dmaStartMemcpy, 285
- dmaStreamAllocate, 278
- dmaStreamClearInterrupt, 285
- dmaStreamDisable, 284
- dmaStreamEnable, 284
- dmaStreamGetTransactionSize, 283
- dmaStreamRelease, 279
- dmaStreamSetMemory0, 282
- dmaStreamSetMode, 283
- dmaStreamSetPeripheral, 281
- dmaStreamSetTransactionSize, 282
- dmaWaitCompletion, 286
- STM32_DMA1_STREAMS_MASK, 279
- STM32_DMA2_STREAMS_MASK, 280
- STM32_DMA_CCR_RESET_VALUE, 280
- STM32_DMA_CR_CHSEL, 281
- STM32_DMA_CR_CHSEL_MASK, 281
- STM32_DMA_CR_DMEIE, 281
- STM32_DMA_GETCHANNEL, 280
- STM32_DMA_IS_VALID_ID, 281
- STM32_DMA_ISR_MASK, 280
- STM32_DMA_STREAM, 281
- STM32_DMA_STREAM_ID, 280
- STM32_DMA_STREAM_ID_MSK, 280
- STM32_DMA_STREAMS, 280
- stm32_dmaisr_t, 286

STM32L1xx Drivers, 267

STM32L1xx EXT Support, 269

STM32L1xx GPT Support, 269

STM32L1xx ICU Support, 269

STM32L1xx Initialization Support, 268

STM32L1xx PAL Support, 270

STM32L1xx Platform Drivers, 273

STM32L1xx PWM Support, 271

STM32L1xx RCC Support, 287

- rccDisableADC1, 293
- rccDisableAHB, 292
- rccDisableAPB1, 290
- rccDisableAPB2, 291

rccDisableDMA1, 293
rccDisableI2C1, 295
rccDisableI2C2, 295
rccDisablePWRInterface, 294
rccDisableSPI1, 296
rccDisableSPI2, 296
rccDisableTIM2, 297
rccDisableTIM3, 297
rccDisableTIM4, 298
rccDisableUSART1, 299
rccDisableUSART2, 299
rccDisableUSART3, 300
rccDisableUSB, 300
rccEnableADC1, 293
rccEnableAHB, 292
rccEnableAPB1, 290
rccEnableAPB2, 291
rccEnableDMA1, 293
rccEnableI2C1, 294
rccEnableI2C2, 295
rccEnablePWRInterface, 294
rccEnableSPI1, 296
rccEnableSPI2, 296
rccEnableTIM2, 297
rccEnableTIM3, 297
rccEnableTIM4, 298
rccEnableUSART1, 298
rccEnableUSART2, 299
rccEnableUSART3, 300
rccEnableUSB, 300
rccResetADC1, 293
rccResetAHB, 292
rccResetAPB1, 290
rccResetAPB2, 291
rccResetDMA1, 294
rccResetI2C1, 295
rccResetI2C2, 295
rccResetPWRInterface, 294
rccResetSPI1, 296
rccResetSPI2, 297
rccResetTIM2, 297
rccResetTIM3, 298
rccResetTIM4, 298
rccResetUSART1, 299
rccResetUSART2, 299
rccResetUSART3, 300
rccResetUSB, 301
STM32L1xx Serial Support, 271
STM32L1xx SPI Support, 272
STM32L1xx UART Support, 272
STM32L1xx USB Support, 273
STM_MCOCLK
 HAL Driver, 95
STM_MCODIVCLK
 HAL Driver, 95
STM_RTCCLK
 HAL Driver, 95
stop
 TimeMeasurement, 349
TAMPER_STAMP_IRQHandler
 HAL Driver, 87
thread
 ADCDriver, 308
 I2CDriver, 319
 SPIDriver, 344
tim
 GPTDriver, 317
 ICUDriver, 324
 PWMDriver, 338
 TIM10_IRQHandler
 HAL Driver, 89
 TIM11_IRQHandler
 HAL Driver, 89
 TIM2_IRQHandler
 HAL Driver, 89
 TIM3_IRQHandler
 HAL Driver, 89
 TIM4_IRQHandler
 HAL Driver, 90
 TIM6_IRQHandler
 HAL Driver, 91
 TIM7_IRQHandler
 HAL Driver, 91
 TIM9_IRQHandler
 HAL Driver, 89
Time Measurement Driver., 210
 TimeMeasurement, 212
 tmlInit, 210
 tmObjectInit, 211
 tmStartMeasurement, 211
 tmStopMeasurement, 211
TimeMeasurement, 348
 best, 349
 last, 349
 start, 349
 stop, 349
 Time Measurement Driver., 212
 worst, 349
tm.c, 423
tm.h, 424
tmlInit
 Time Measurement Driver., 210
tmObjectInit
 Time Measurement Driver., 211
tmStartMeasurement
 Time Measurement Driver., 211
tmStopMeasurement
 Time Measurement Driver., 211
transmitting
 USBDriver, 359
TXADDR0
 stm32_usb_descriptor_t, 347
txbuf
 USBInEndpointState, 363
txcnt
 USBInEndpointState, 363
TXCOUNT0
 stm32_usb_descriptor_t, 347

TXCOUNT1
 stm32_usb_descriptor_t, 347

txdmamode
 SPIDriver, 344

txend1_cb
 UARTConfig, 351

txend2_cb
 UARTConfig, 351

txsize
 USBInEndpointState, 363

txstate
 UARTDriver, 354

UART Driver, 212
 CH_IRQ_HANDLER, 223
 STM32_UART_DMA_ERROR_HOOK, 228
 STM32_UART_USART1_DMA_PRIORITY, 228
 STM32_UART_USART1_IRQ_PRIORITY, 227
 STM32_UART_USART1_RX_DMA_STREAM, 228
 STM32_UART_USART1_TX_DMA_STREAM, 228
 STM32_UART_USART2_DMA_PRIORITY, 228
 STM32_UART_USART2_IRQ_PRIORITY, 227
 STM32_UART_USART2_RX_DMA_STREAM, 228
 STM32_UART_USART2_TX_DMA_STREAM, 229
 STM32_UART_USART3_DMA_PRIORITY, 228
 STM32_UART_USART3_IRQ_PRIORITY, 227
 STM32_UART_USART3_RX_DMA_STREAM, 229
 STM32_UART_USART3_TX_DMA_STREAM, 229
 STM32_UART_USE_USART1, 227
 STM32_UART_USE_USART2, 227
 STM32_UART_USE_USART3, 227
 UART_READY, 230
 UART_RX_ACTIVE, 230
 UART_RX_COMPLETE, 230
 UART_RX_IDLE, 230
 UART_STOP, 230
 UART_TX_ACTIVE, 230
 UART_TX_COMPLETE, 230
 UART_TX_IDLE, 230
 UART_UNINIT, 230
 UART_BREAK_DETECTED, 227
 UART_FRAMING_ERROR, 226
 uart_lld_init, 223
 uart_lld_start, 224
 uart_lld_start_receive, 225
 uart_lld_start_send, 225
 uart_lld_stop, 224
 uart_lld_stop_receive, 226
 uart_lld_stop_send, 225
 UART_NO_ERROR, 226
 UART_NOISE_ERROR, 227
 UART_OVERRUN_ERROR, 227
 UART_PARITY_ERROR, 226
 uartccb_t, 229
 UARTD1, 226
 UARTD2, 226
 UARTD3, 226
 UARTDriver, 229

 uartecb_t, 230
 uartflags_t, 229
 uartInit, 217
 uartObjectInit, 217
 uartrxstate_t, 230
 uartStart, 217
 uartStartReceive, 221
 uartStartReceivel, 221
 uartStartSend, 218
 uartStartSendl, 219
 uartstate_t, 230
 uartStop, 218
 uartStopReceive, 222
 uartStopReceivel, 222
 uartStopSend, 219
 uartStopSendl, 220
 uarttxstate_t, 230

 uart.c, 424
 uart.h, 425

 UART_READY
 UART Driver, 230

 UART_RX_ACTIVE
 UART Driver, 230

 UART_RX_COMPLETE
 UART Driver, 230

 UART_RX_IDLE
 UART Driver, 230

 UART_STOP
 UART Driver, 230

 UART_TX_ACTIVE
 UART Driver, 230

 UART_TX_COMPLETE
 UART Driver, 230

 UART_TX_IDLE
 UART Driver, 230

 UART_UNINIT
 UART Driver, 230

 UART_BREAK_DETECTED
 UART Driver, 227

 UART_FRAMING_ERROR
 UART Driver, 226

 uart_lld.c, 426
 uart_lld.h, 427

 uart_lld_init
 UART Driver, 223

 uart_lld_start
 UART Driver, 224

 uart_lld_start_receive
 UART Driver, 225

 uart_lld_start_send
 UART Driver, 225

 uart_lld_stop
 UART Driver, 224

 uart_lld_stop_receive
 UART Driver, 226

 uart_lld_stop_send
 UART Driver, 225

 UART_NO_ERROR
 UART Driver, 226

UART_NOISE_ERROR
 UART Driver, 227

UART_OVERRUN_ERROR
 UART Driver, 227

UART_PARITY_ERROR
 UART Driver, 226

uartcb_t
 UART Driver, 229

uartccb_t
 UART Driver, 229

UARTConfig, 349

 cr1, 351

 cr2, 351

 cr3, 351

 rxchar_cb, 351

 rxend_cb, 351

 rxerr_cb, 351

 speed, 351

 txend1_cb, 351

 txend2_cb, 351

UARTD1
 UART Driver, 226

UARTD2
 UART Driver, 226

UARTD3
 UART Driver, 226

UARTDriver, 352

 config, 354

 dmemode, 354

 dmarx, 354

 dmatx, 354

 rdbuf, 354

 rxstate, 354

 state, 354

 txstate, 354

 UART Driver, 229

 uart, 354

uartecb_t
 UART Driver, 230

uartflags_t
 UART Driver, 229

uartInit
 UART Driver, 217

uartObjectInit
 UART Driver, 217

uartrxstate_t
 UART Driver, 230

uartStart
 UART Driver, 217

uartStartReceive
 UART Driver, 221

uartStartReceive1
 UART Driver, 221

uartStartSend
 UART Driver, 218

uartStartSend1
 UART Driver, 219

uartstate_t
 UART Driver, 230

uartStop
 UART Driver, 218

uartStopReceive
 UART Driver, 222

uartStopReceive1
 UART Driver, 222

uartStopSend
 UART Driver, 219

uartStopSend1
 UART Driver, 220

uarttxstate_t
 UART Driver, 230

ud_size
 USBDescriptor, 357

ud_string
 USBDescriptor, 357

US2RTT
 HAL Driver, 80

usart
 UARTDriver, 354

USART1_IRQHandler
 HAL Driver, 90

USART2_IRQHandler
 HAL Driver, 90

USART3_IRQHandler
 HAL Driver, 90

USART_CR2_STOP0P5_BITS
 Serial Driver, 187

USART_CR2_STOP1_BITS
 Serial Driver, 187

USART_CR2_STOP1P5_BITS
 Serial Driver, 187

USART_CR2_STOP2_BITS
 Serial Driver, 187

USB Driver, 231

 _usb_ep0in, 245

 _usb_ep0out, 246

 _usb_ep0setup, 244

 _usb_isr_invoke_event_cb, 261

 _usb_isr_invoke_in_cb, 262

 _usb_isr_invoke_out_cb, 262

 _usb_isr_invoke_setup_cb, 262

 _usb_isr_invoke_sof_cb, 261

 _usb_reset, 244

 CH_IRQ_HANDLER, 247

 EP_STATUS_ACTIVE, 267

 EP_STATUS_DISABLED, 267

 EP_STATUS_STALLED, 267

 EPR_TOGGLE_MASK, 263

 in, 254

 out, 254

 STM32_USB, 263

 STM32_USB_BASE, 263

 STM32_USB_LOW_POWER_ON_SUSPEND, 264

 STM32_USB_USB1_HP_IRQ_PRIORITY, 264

 STM32_USB_USB1_LP_IRQ_PRIORITY, 264

 STM32_USB_USE_USB1, 264

 STM32_USBRAM, 263

 STM32_USBRAM_BASE, 263

USB_ACTIVE, 267
USB_EP0_ERROR, 267
USB_EP0_RX, 267
USB_EP0_SENDING_STS, 267
USB_EP0_TX, 267
USB_EP0_WAITING_SETUP, 267
USB_EP0_WAITING_STS, 267
USB_EVENT_ADDRESS, 267
USB_EVENT_CONFIGURED, 267
USB_EVENT_RESET, 267
USB_EVENT_STALLED, 267
USB_EVENT_SUSPEND, 267
USB_EVENT_WAKEUP, 267
USB_READY, 266
USB_SELECTED, 267
USB_STOP, 266
USB_UNINIT, 266
USB_ADDR2PTR, 263
USB_DESC_BCD, 255
USB_DESC_BYTE, 255
USB_DESC_CONFIGURATION, 255
USB_DESC_DEVICE, 255
USB_DESC_ENDPOINT, 256
USB_DESC_INDEX, 254
USB_DESC_INTERFACE, 255
USB_DESC_WORD, 255
USB_ENDPOINTS_NUMBER, 262
USB_EP_MODE_PACKET, 256
USB_EP_MODE_TRANSACTION, 256
USB_EP_MODE_TYPE, 256
USB_EP_MODE_TYPE_BULK, 256
USB_EP_MODE_TYPE_CTRL, 256
USB_EP_MODE_TYPE_INTR, 256
USB_EP_MODE_TYPE_ISOC, 256
USB_GET_DESCRIPTOR, 263
usb_lld_clear_in, 254
usb_lld_clear_out, 254
usb_lld_disable_endpoints, 250
usb_lld_fetch_word, 264
usb_lld_get_frame_number, 264
usb_lld_get_packet_size, 265
usb_lld_get_status_in, 250
usb_lld_get_status_out, 250
usb_lld_get_transaction_size, 264
usb_lld_init, 248
usb_lld_init_endpoint, 250
usb_lld_prepare_receive, 252
usb_lld_prepare_transmit, 252
usb_lld_read_packet_buffer, 251
usb_lld_read_setup, 251
usb_lld_reset, 249
usb_lld_set_address, 249
usb_lld_stall_in, 253
usb_lld_stall_out, 253
usb_lld_start, 248
usb_lld_start_in, 253
usb_lld_start_out, 253
usb_lld_stop, 249
usb_lld_write_packet_buffer, 252
USB_MAX_ENDPOINTS, 263
USB_PMA_SIZE, 263
USB_SET_ADDRESS_MODE, 263
usbcallback_t, 265
usbConnectBus, 256
USBD1, 254
usbDisableEndpointsl, 241
usbDisconnectBus, 257
USBDriver, 265
usbep0state_t, 267
usbep_t, 265
usbepcallback_t, 266
usbepstatus_t, 267
usbevent_t, 267
usbeventcb_t, 266
usbgetdescriptor_t, 266
usbGetFrameNumber, 257
usbGetReceivePacketSizel, 260
usbGetReceiveStatusl, 257
usbGetReceiveTransactionSizel, 259
usbGetTransmitStatusl, 257
usbInit, 239
usbInitEndpointl, 240
usbObjectInit, 239
usbPrepareReceive, 259
usbPrepareTransmit, 259
usbReadPacketBuffer, 258
usbReadSetup, 260
usbreqhandler_t, 266
usbSetupTransfer, 260
usbStallReceivev, 243
usbStallTransmitl, 243
usbStart, 239
usbStartReceivev, 241
usbStartTransmitl, 242
usbstate_t, 266
usbStop, 240
usbWritePacketBuffer, 258
usb.c, 428
usb.h, 429
USB_ACTIVE
 USB Driver, 267
USB_EP0_ERROR
 USB Driver, 267
USB_EP0_RX
 USB Driver, 267
USB_EP0_SENDING_STS
 USB Driver, 267
USB_EP0_TX
 USB Driver, 267
USB_EP0_WAITING_SETUP
 USB Driver, 267
USB_EP0_WAITING_STS
 USB Driver, 267
USB_EVENT_ADDRESS
 USB Driver, 267
USB_EVENT_CONFIGURED
 USB Driver, 267
USB_EVENT_RESET

USB Driver, 267
USB_EVENT_STALLED
 USB Driver, 267
USB_EVENT_SUSPEND
 USB Driver, 267
USB_EVENT_WAKEUP
 USB Driver, 267
USB_READY
 USB Driver, 266
USB_SELECTED
 USB Driver, 267
USB_STOP
 USB Driver, 266
USB_UNINIT
 USB Driver, 266
USB_ADDR2PTR
 USB Driver, 263
USB_DESC_BCD
 USB Driver, 255
USB_DESC_BYTE
 USB Driver, 255
USB_DESC_CONFIGURATION
 USB Driver, 255
USB_DESC_DEVICE
 USB Driver, 255
USB_DESC_ENDPOINT
 USB Driver, 256
USB_DESC_INDEX
 USB Driver, 254
USB_DESC_INTERFACE
 USB Driver, 255
USB_DESC_WORD
 USB Driver, 255
USB_ENDPOINTS_NUMBER
 USB Driver, 262
USB_EP_MODE_PACKET
 USB Driver, 256
USB_EP_MODE_TRANSACTION
 USB Driver, 256
USB_EP_MODE_TYPE
 USB Driver, 256
USB_EP_MODE_TYPE_BULK
 USB Driver, 256
USB_EP_MODE_TYPE_CTRL
 USB Driver, 256
USB_EP_MODE_TYPE_INTR
 USB Driver, 256
USB_EP_MODE_TYPE_ISOC
 USB Driver, 256
USB_FS_WKUP_IRQHandler
 HAL Driver, 91
USB_GET_DESCRIPTOR
 USB Driver, 263
USB_HP_IRQHandler
 HAL Driver, 89
usb_lld.c, 432
 in, 433
 out, 433
usb_lld.h, 434
usb_lld_clear_in
 USB Driver, 254
usb_lld_clear_out
 USB Driver, 254
usb_lld_disable_endpoints
 USB Driver, 250
usb_lld_fetch_word
 USB Driver, 264
usb_lld_get_frame_number
 USB Driver, 264
usb_lld_get_packet_size
 USB Driver, 265
usb_lld_get_status_in
 USB Driver, 250
usb_lld_get_status_out
 USB Driver, 250
usb_lld_get_transaction_size
 USB Driver, 264
usb_lld_init
 USB Driver, 248
usb_lld_init_endpoint
 USB Driver, 250
usb_lld_prepare_receive
 USB Driver, 252
usb_lld_prepare_transmit
 USB Driver, 252
usb_lld_read_packet_buffer
 USB Driver, 251
usb_lld_read_setup
 USB Driver, 251
usb_lld_reset
 USB Driver, 249
usb_lld_set_address
 USB Driver, 249
usb_lld_stall_in
 USB Driver, 253
usb_lld_stall_out
 USB Driver, 253
usb_lld_start
 USB Driver, 248
usb_lld_start_in
 USB Driver, 253
usb_lld_start_out
 USB Driver, 253
usb_lld_stop
 USB Driver, 249
usb_lld_write_packet_buffer
 USB Driver, 252
USB_LP_IRQHandler
 HAL Driver, 89
USB_MAX_ENDPOINTS
 USB Driver, 263
USB_PMA_SIZE
 USB Driver, 263
USB_SET_ADDRESS_MODE
 USB Driver, 263
uscallback_t
 USB Driver, 265
USBConfig, 355

event_cb, 357
get_descriptor_cb, 357
requests_hook_cb, 357
sof_cb, 357
usbConnectBus
 USB Driver, 256
USBD1
 USB Driver, 254
USBDescriptor, 357
 ud_size, 357
 ud_string, 357
usbDisableEndpointsI
 USB Driver, 241
usbDisconnectBus
 USB Driver, 257
USBDriver, 358
 address, 360
 config, 359
 configuration, 360
 ep0endcb, 360
 ep0n, 360
 ep0next, 359
 ep0state, 359
 epc, 359
 param, 359
 pmnnext, 360
 receiving, 359
 setup, 360
 state, 359
 status, 360
 transmitting, 359
 USB Driver, 265
USBEndpointConfig, 360
 ep_mode, 362
 in_cb, 362
 in_maxsize, 362
 in_state, 363
 out_cb, 362
 out_maxsize, 362
 out_state, 363
 setup_cb, 362
usbep0state_t
 USB Driver, 267
usbep_t
 USB Driver, 265
usbepcallback_t
 USB Driver, 266
usbepstatus_t
 USB Driver, 267
usbevent_t
 USB Driver, 267
usbeventcb_t
 USB Driver, 266
usbgetdescriptor_t
 USB Driver, 266
usbGetFrameNumber
 USB Driver, 257
usbGetReceivePacketSizeI
 USB Driver, 260
usbGetReceiveStatusI
 USB Driver, 257
usbGetReceiveTransactionSizeI
 USB Driver, 259
usbGetTransmitStatusI
 USB Driver, 257
USBInEndpointState, 363
 txbuf, 363
 txcnt, 363
 txsize, 363
usbInit
 USB Driver, 239
usbInitEndpointI
 USB Driver, 240
usbObjectInit
 USB Driver, 239
USBOutEndpointState, 364
 rdbuf, 364
 rcnt, 364
 rpkts, 364
 rszize, 364
usbPrepareReceive
 USB Driver, 259
usbPrepareTransmit
 USB Driver, 259
usbReadPacketBuffer
 USB Driver, 258
usbReadSetup
 USB Driver, 260
usbreqhandler_t
 USB Driver, 266
usbSetupTransfer
 USB Driver, 260
usbStallReceiveI
 USB Driver, 243
usbStallTransmitI
 USB Driver, 243
usbStart
 USB Driver, 239
usbStartReceiveI
 USB Driver, 241
usbStartTransmitI
 USB Driver, 242
usbstate_t
 USB Driver, 266
usbStop
 USB Driver, 240
usbWritePacketBuffer
 USB Driver, 258
vector
 stm32_dma_stream_t, 345
vmt
 SerialDriver, 340
vt
 MMCDriver, 329
wakeup_isr
 I2C Driver, 109

width_cb
 ICUConfig, [322](#)
worst
 TimeMeasurement, [349](#)
WWDG_IRQHandler
 HAL Driver, [87](#)