

ChibiOS/RT 2.4.3

STM32F4xx 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	5
5.1 File List	5
6 Module Documentation	7
6.1 HAL	7
6.1.1 Detailed Description	7
6.1.2 HAL Device Drivers Architecture	7
6.1.2.1 Diagram	8
6.2 Configuration	9
6.2.1 Detailed Description	9
6.2.2 Define Documentation	11
6.2.2.1 HAL_USE_PAL	11
6.2.2.2 HAL_USE_ADC	11
6.2.2.3 HAL_USE_CAN	11
6.2.2.4 HAL_USE_EXT	11
6.2.2.5 HAL_USE_GPT	11
6.2.2.6 HAL_USE_I2C	12
6.2.2.7 HAL_USE_ICU	12
6.2.2.8 HAL_USE_MAC	12
6.2.2.9 HAL_USE_MMCSPI	12

6.2.2.10 HAL_USE_PWM	12
6.2.2.11 HAL_USE_RTC	12
6.2.2.12 HAL_USE_SDC	12
6.2.2.13 HAL_USE_SERIAL	12
6.2.2.14 HAL_USE_SERIAL_USB	12
6.2.2.15 HAL_USE_SPI	12
6.2.2.16 HAL_USE_UART	12
6.2.2.17 HAL_USE_USB	12
6.2.2.18 ADC_USE_WAIT	13
6.2.2.19 ADC_USE_MUTUAL_EXCLUSION	13
6.2.2.20 CAN_USE_SLEEP_MODE	13
6.2.2.21 I2C_USE_MUTUAL_EXCLUSION	13
6.2.2.22 MAC_USE_EVENTS	13
6.2.2.23 MMC_SECTOR_SIZE	13
6.2.2.24 MMC_NICE_WAITING	13
6.2.2.25 MMC_POLLING_INTERVAL	13
6.2.2.26 MMC_POLLING_DELAY	13
6.2.2.27 MMC_USE_SPI_POLLING	13
6.2.2.28 SDC_INIT_RETRY	14
6.2.2.29 SDC_MMC_SUPPORT	14
6.2.2.30 SDC_NICE_WAITING	14
6.2.2.31 SERIAL_DEFAULT_BITRATE	14
6.2.2.32 SERIAL_BUFFERS_SIZE	14
6.2.2.33 SERIAL_USB_BUFFERS_SIZE	14
6.2.2.34 SPI_USE_WAIT	14
6.2.2.35 SPI_USE_MUTUAL_EXCLUSION	15
6.3 ADC Driver	15
6.3.1 Detailed Description	15
6.3.2 Driver State Machine	15
6.3.3 ADC Operations	15
6.3.3.1 ADC Conversion Groups	15
6.3.3.2 ADC Conversion Modes	16
6.3.3.3 ADC Callbacks	16
6.3.4 Function Documentation	22
6.3.4.1 adcInit	22
6.3.4.2 adcObjectInit	23
6.3.4.3 adcStart	23
6.3.4.4 adcStop	23
6.3.4.5 adcStartConversion	24
6.3.4.6 adcStartConversionl	24

6.3.4.7	adcStopConversion	25
6.3.4.8	adcStopConversionl	26
6.3.4.9	adcAcquireBus	26
6.3.4.10	adcReleaseBus	26
6.3.4.11	adcConvert	27
6.3.4.12	CH_IRQ_HANDLER	27
6.3.4.13	adc_lld_init	28
6.3.4.14	adc_lld_start	28
6.3.4.15	adc_lld_stop	28
6.3.4.16	adc_lld_start_conversion	29
6.3.4.17	adc_lld_stop_conversion	29
6.3.4.18	adcSTM32EnableTSVREFE	29
6.3.4.19	adcSTM32DisableTSVREFE	29
6.3.4.20	adcSTM32EnableVBATE	30
6.3.4.21	adcSTM32DisableVBATE	30
6.3.5	Variable Documentation	30
6.3.5.1	ADCD1	30
6.3.5.2	ADCD2	30
6.3.5.3	ADCD3	30
6.3.6	Define Documentation	30
6.3.6.1	ADC_USE_WAIT	30
6.3.6.2	ADC_USE_MUTUAL_EXCLUSION	30
6.3.6.3	_adc_reset_i	31
6.3.6.4	_adc_reset_s	31
6.3.6.5	_adc_wakeup_isr	31
6.3.6.6	_adc_timeout_isr	32
6.3.6.7	_adc_isr_half_code	32
6.3.6.8	_adc_isr_full_code	33
6.3.6.9	_adc_isr_error_code	33
6.3.6.10	STM32_ADCCLK_MIN	34
6.3.6.11	STM32_ADCCLK_MAX	34
6.3.6.12	ADC_CR2_EXTSEL_SRC	34
6.3.6.13	ADC_CHANNEL_IN0	34
6.3.6.14	ADC_CHANNEL_IN1	34
6.3.6.15	ADC_CHANNEL_IN2	34
6.3.6.16	ADC_CHANNEL_IN3	34
6.3.6.17	ADC_CHANNEL_IN4	34
6.3.6.18	ADC_CHANNEL_IN5	34
6.3.6.19	ADC_CHANNEL_IN6	34
6.3.6.20	ADC_CHANNEL_IN7	34

6.3.6.21	ADC_CHANNEL_IN8	34
6.3.6.22	ADC_CHANNEL_IN9	35
6.3.6.23	ADC_CHANNEL_IN10	35
6.3.6.24	ADC_CHANNEL_IN11	35
6.3.6.25	ADC_CHANNEL_IN12	35
6.3.6.26	ADC_CHANNEL_IN13	35
6.3.6.27	ADC_CHANNEL_IN14	35
6.3.6.28	ADC_CHANNEL_IN15	35
6.3.6.29	ADC_CHANNEL_SENSOR	35
6.3.6.30	ADC_CHANNEL_VREFINT	35
6.3.6.31	ADC_CHANNEL_VBAT	35
6.3.6.32	ADC_SAMPLE_3	36
6.3.6.33	ADC_SAMPLE_15	36
6.3.6.34	ADC_SAMPLE_28	36
6.3.6.35	ADC_SAMPLE_56	36
6.3.6.36	ADC_SAMPLE_84	36
6.3.6.37	ADC_SAMPLE_112	36
6.3.6.38	ADC_SAMPLE_144	36
6.3.6.39	ADC_SAMPLE_480	36
6.3.6.40	STM32_ADC_ADCPRE	36
6.3.6.41	STM32_ADC_USE_ADC1	36
6.3.6.42	STM32_ADC_USE_ADC2	37
6.3.6.43	STM32_ADC_USE_ADC3	37
6.3.6.44	STM32_ADC_ADC1_DMA_STREAM	37
6.3.6.45	STM32_ADC_ADC2_DMA_STREAM	37
6.3.6.46	STM32_ADC_ADC3_DMA_STREAM	37
6.3.6.47	STM32_ADC_ADC1_DMA_PRIORITY	37
6.3.6.48	STM32_ADC_ADC2_DMA_PRIORITY	37
6.3.6.49	STM32_ADC_ADC3_DMA_PRIORITY	37
6.3.6.50	STM32_ADC_IRQ_PRIORITY	37
6.3.6.51	STM32_ADC_ADC1_DMA_IRQ_PRIORITY	38
6.3.6.52	STM32_ADC_ADC2_DMA_IRQ_PRIORITY	38
6.3.6.53	STM32_ADC_ADC3_DMA_IRQ_PRIORITY	38
6.3.6.54	ADC_SQR1_NUM_CH	38
6.3.6.55	ADC_SQR3_SQ1_N	38
6.3.6.56	ADC_SQR3_SQ2_N	38
6.3.6.57	ADC_SQR3_SQ3_N	38
6.3.6.58	ADC_SQR3_SQ4_N	38
6.3.6.59	ADC_SQR3_SQ5_N	38
6.3.6.60	ADC_SQR3_SQ6_N	38

6.3.6.61	ADC_SQR2_SQ7_N	38
6.3.6.62	ADC_SQR2_SQ8_N	38
6.3.6.63	ADC_SQR2_SQ9_N	39
6.3.6.64	ADC_SQR2_SQ10_N	39
6.3.6.65	ADC_SQR2_SQ11_N	39
6.3.6.66	ADC_SQR2_SQ12_N	39
6.3.6.67	ADC_SQR1_SQ13_N	39
6.3.6.68	ADC_SQR1_SQ14_N	39
6.3.6.69	ADC_SQR1_SQ15_N	39
6.3.6.70	ADC_SQR1_SQ16_N	39
6.3.6.71	ADC_SMPR2_SMP_AN0	39
6.3.6.72	ADC_SMPR2_SMP_AN1	39
6.3.6.73	ADC_SMPR2_SMP_AN2	39
6.3.6.74	ADC_SMPR2_SMP_AN3	39
6.3.6.75	ADC_SMPR2_SMP_AN4	40
6.3.6.76	ADC_SMPR2_SMP_AN5	40
6.3.6.77	ADC_SMPR2_SMP_AN6	40
6.3.6.78	ADC_SMPR2_SMP_AN7	40
6.3.6.79	ADC_SMPR2_SMP_AN8	40
6.3.6.80	ADC_SMPR2_SMP_AN9	40
6.3.6.81	ADC_SMPR1_SMP_AN10	40
6.3.6.82	ADC_SMPR1_SMP_AN11	40
6.3.6.83	ADC_SMPR1_SMP_AN12	40
6.3.6.84	ADC_SMPR1_SMP_AN13	40
6.3.6.85	ADC_SMPR1_SMP_AN14	40
6.3.6.86	ADC_SMPR1_SMP_AN15	40
6.3.6.87	ADC_SMPR1_SMP_SENSOR	41
6.3.6.88	ADC_SMPR1_SMP_VREF	41
6.3.6.89	ADC_SMPR1_SMP_VBAT	41
6.3.7	Typedef Documentation	41
6.3.7.1	adcsample_t	41
6.3.7.2	adc_channels_num_t	41
6.3.7.3	ADCDriver	41
6.3.7.4	adccallback_t	41
6.3.7.5	adccallback_t	41
6.3.8	Enumeration Type Documentation	41
6.3.8.1	adcstate_t	41
6.3.8.2	adcerror_t	42
6.4	CAN Driver	42
6.4.1	Detailed Description	42

6.4.2	Driver State Machine	42
6.4.3	Function Documentation	44
6.4.3.1	canInit	44
6.4.3.2	canObjectInit	44
6.4.3.3	canStart	44
6.4.3.4	canStop	45
6.4.3.5	canTransmit	45
6.4.3.6	canReceive	46
6.4.3.7	canGetAndClearFlags	46
6.4.3.8	canSleep	46
6.4.3.9	canWakeup	47
6.4.4	Define Documentation	47
6.4.4.1	CAN_LIMIT_WARNING	47
6.4.4.2	CAN_LIMIT_ERROR	47
6.4.4.3	CAN_BUS_OFF_ERROR	47
6.4.4.4	CAN_FRAMING_ERROR	47
6.4.4.5	CAN_OVERFLOW_ERROR	47
6.4.4.6	CAN_USE_SLEEP_MODE	47
6.4.4.7	canAddFlags!	47
6.4.5	Enumeration Type Documentation	48
6.4.5.1	canstate_t	48
6.5	EXT Driver	48
6.5.1	Detailed Description	48
6.5.2	Driver State Machine	48
6.5.3	EXT Operations.	49
6.5.4	Function Documentation	51
6.5.4.1	extInit	51
6.5.4.2	extObjectInit	52
6.5.4.3	extStart	52
6.5.4.4	extStop	53
6.5.4.5	extChannelEnable	53
6.5.4.6	extChannelDisable	53
6.5.4.7	CH_IRQ_HANDLER	53
6.5.4.8	CH_IRQ_HANDLER	54
6.5.4.9	CH_IRQ_HANDLER	54
6.5.4.10	CH_IRQ_HANDLER	54
6.5.4.11	CH_IRQ_HANDLER	54
6.5.4.12	CH_IRQ_HANDLER	54
6.5.4.13	CH_IRQ_HANDLER	54
6.5.4.14	CH_IRQ_HANDLER	54

6.5.4.15	CH_IRQ_HANDLER	55
6.5.4.16	CH_IRQ_HANDLER	55
6.5.4.17	ext_lld_init	55
6.5.4.18	ext_lld_start	55
6.5.4.19	ext_lld_stop	55
6.5.4.20	ext_lld_channel_enable	56
6.5.4.21	ext_lld_channel_disable	56
6.5.5	Variable Documentation	56
6.5.5.1	EXTD1	56
6.5.6	Define Documentation	56
6.5.6.1	EXT_MAX_CHANNELS	56
6.5.6.2	EXT_CHANNELS_MASK	56
6.5.6.3	EXT_MODE_EXTI	56
6.5.6.4	EXT_MODE_GPIOA	57
6.5.6.5	EXT_MODE_GPIOB	57
6.5.6.6	EXT_MODE_GPIOC	57
6.5.6.7	EXT_MODE_GPIOD	57
6.5.6.8	EXT_MODE_GPIOE	57
6.5.6.9	EXT_MODE_GPIOF	57
6.5.6.10	EXT_MODE_GPIOG	57
6.5.6.11	EXT_MODE_GPIOH	57
6.5.6.12	EXT_MODE_GPIOI	57
6.5.6.13	STM32_EXT EXTI0_IRQ_PRIORITY	57
6.5.6.14	STM32_EXT EXTI1_IRQ_PRIORITY	57
6.5.6.15	STM32_EXT EXTI2_IRQ_PRIORITY	58
6.5.6.16	STM32_EXT EXTI3_IRQ_PRIORITY	58
6.5.6.17	STM32_EXT EXTI4_IRQ_PRIORITY	58
6.5.6.18	STM32_EXT EXTI5_9_IRQ_PRIORITY	58
6.5.6.19	STM32_EXT EXTI10_15_IRQ_PRIORITY	58
6.5.6.20	STM32_EXT EXTI16_IRQ_PRIORITY	58
6.5.6.21	STM32_EXT EXTI17_IRQ_PRIORITY	58
6.5.6.22	STM32_EXT EXTI18_IRQ_PRIORITY	58
6.5.6.23	STM32_EXT EXTI19_IRQ_PRIORITY	58
6.5.6.24	STM32_EXT EXTI20_IRQ_PRIORITY	58
6.5.6.25	STM32_EXT EXTI21_IRQ_PRIORITY	58
6.5.6.26	STM32_EXT EXTI22_IRQ_PRIORITY	58
6.5.7	Typedef Documentation	59
6.5.7.1	expchannel_t	59
6.5.7.2	extcallback_t	59
6.6	GPT Driver	59

6.6.1	Detailed Description	59
6.6.2	Driver State Machine	59
6.6.3	GPT Operations.	60
6.6.4	Function Documentation	63
6.6.4.1	gptInit	63
6.6.4.2	gptObjectInit	63
6.6.4.3	gptStart	63
6.6.4.4	gptStop	64
6.6.4.5	gptStartContinuous	64
6.6.4.6	gptStartContinuousl	65
6.6.4.7	gptStartOneShot	65
6.6.4.8	gptStartOneShotl	66
6.6.4.9	gptStopTimer	66
6.6.4.10	gptStopTimerl	67
6.6.4.11	gptPolledDelay	67
6.6.4.12	gpt_lld_init	68
6.6.4.13	gpt_lld_start	68
6.6.4.14	gpt_lld_stop	68
6.6.4.15	gpt_lld_start_timer	69
6.6.4.16	gpt_lld_stop_timer	69
6.6.4.17	gpt_lld_polled_delay	69
6.6.5	Variable Documentation	69
6.6.5.1	GPTD1	69
6.6.5.2	GPTD2	69
6.6.5.3	GPTD3	70
6.6.5.4	GPTD4	70
6.6.5.5	GPTD5	70
6.6.5.6	GPTD8	70
6.6.6	Define Documentation	70
6.6.6.1	STM32_GPT_USE_TIM1	70
6.6.6.2	STM32_GPT_USE_TIM2	70
6.6.6.3	STM32_GPT_USE_TIM3	71
6.6.6.4	STM32_GPT_USE_TIM4	71
6.6.6.5	STM32_GPT_USE_TIM5	71
6.6.6.6	STM32_GPT_USE_TIM8	71
6.6.6.7	STM32_GPT_TIM1_IRQ_PRIORITY	71
6.6.6.8	STM32_GPT_TIM2_IRQ_PRIORITY	71
6.6.6.9	STM32_GPT_TIM3_IRQ_PRIORITY	71
6.6.6.10	STM32_GPT_TIM4_IRQ_PRIORITY	71
6.6.6.11	STM32_GPT_TIM5_IRQ_PRIORITY	72

6.6.6.12	STM32_GPT_TIM8_IRQ_PRIORITY	72
6.6.7	Typedef Documentation	72
6.6.7.1	GPTDriver	72
6.6.7.2	gptcallback_t	72
6.6.7.3	gptfreq_t	72
6.6.7.4	gptcnt_t	72
6.6.8	Enumeration Type Documentation	72
6.6.8.1	gptstate_t	72
6.7	HAL Driver	72
6.7.1	Detailed Description	72
6.7.2	Function Documentation	83
6.7.2.1	hallInit	83
6.7.2.2	hallsCounterWithin	84
6.7.2.3	halPolledDelay	85
6.7.2.4	hal_lld_init	86
6.7.2.5	stm32_clock_init	86
6.7.3	Define Documentation	86
6.7.3.1	S2RTT	86
6.7.3.2	MS2RTT	87
6.7.3.3	US2RTT	87
6.7.3.4	halGetCounterValue	88
6.7.3.5	halGetCounterFrequency	88
6.7.3.6	HAL_IMPLEMENTS_COUNTERS	88
6.7.3.7	STM32_HSECLK_MAX	88
6.7.3.8	STM32_HSECLK_MIN	88
6.7.3.9	STM32_LSECLK_MAX	88
6.7.3.10	STM32_LSECLK_MIN	88
6.7.3.11	STM32_PLLIN_MAX	89
6.7.3.12	STM32_PLLIN_MIN	89
6.7.3.13	STM32_PLLVCO_MAX	89
6.7.3.14	STM32_PLLVCO_MIN	89
6.7.3.15	STM32_PLLOUT_MAX	89
6.7.3.16	STM32_PLLOUT_MIN	89
6.7.3.17	STM32_PCLK1_MAX	89
6.7.3.18	STM32_PCLK2_MAX	89
6.7.3.19	STM32_SPII2S_MAX	89
6.7.3.20	STM32_HSICLK	89
6.7.3.21	STM32_LSICLK	89
6.7.3.22	STM32_VOS_MASK	89
6.7.3.23	STM32_VOS_LOW	90

6.7.3.24	STM32_VOS_HIGH	90
6.7.3.25	STM32_PLS_MASK	90
6.7.3.26	STM32_PLS_LEV0	90
6.7.3.27	STM32_PLS_LEV1	90
6.7.3.28	STM32_PLS_LEV2	90
6.7.3.29	STM32_PLS_LEV3	90
6.7.3.30	STM32_PLS_LEV4	90
6.7.3.31	STM32_PLS_LEV5	90
6.7.3.32	STM32_PLS_LEV6	90
6.7.3.33	STM32_PLS_LEV7	90
6.7.3.34	STM32_PLLP_MASK	90
6.7.3.35	STM32_PLLP_DIV2	91
6.7.3.36	STM32_PLLP_DIV4	91
6.7.3.37	STM32_PLLP_DIV6	91
6.7.3.38	STM32_PLLP_DIV8	91
6.7.3.39	STM32_PLLSRC_HSI	91
6.7.3.40	STM32_PLLSRC_HSE	91
6.7.3.41	STM32_SW_MASK	91
6.7.3.42	STM32_SW_HSI	91
6.7.3.43	STM32_SW_HSE	91
6.7.3.44	STM32_SW_PLL	91
6.7.3.45	STM32_HPRE_MASK	91
6.7.3.46	STM32_HPRE_DIV1	91
6.7.3.47	STM32_HPRE_DIV2	92
6.7.3.48	STM32_HPRE_DIV4	92
6.7.3.49	STM32_HPRE_DIV8	92
6.7.3.50	STM32_HPRE_DIV16	92
6.7.3.51	STM32_HPRE_DIV64	92
6.7.3.52	STM32_HPRE_DIV128	92
6.7.3.53	STM32_HPRE_DIV256	92
6.7.3.54	STM32_HPRE_DIV512	92
6.7.3.55	STM32_PPREG1_MASK	92
6.7.3.56	STM32_PPREG1_DIV1	92
6.7.3.57	STM32_PPREG1_DIV2	92
6.7.3.58	STM32_PPREG1_DIV4	92
6.7.3.59	STM32_PPREG1_DIV8	93
6.7.3.60	STM32_PPREG1_DIV16	93
6.7.3.61	STM32_PPREG2_MASK	93
6.7.3.62	STM32_PPREG2_DIV1	93
6.7.3.63	STM32_PPREG2_DIV2	93

6.7.3.64 STM32_PPREG1_DIV4	93
6.7.3.65 STM32_PPREG1_DIV8	93
6.7.3.66 STM32_PPREG1_DIV16	93
6.7.3.67 STM32_RTCPRE_MASK	93
6.7.3.68 STM32_MCO1SEL_MASK	93
6.7.3.69 STM32_MCO1SEL_HSI	93
6.7.3.70 STM32_MCO1SEL_LSE	93
6.7.3.71 STM32_MCO1SEL_HSE	94
6.7.3.72 STM32_MCO1SEL_PLL	94
6.7.3.73 STM32_I2SSRC_MASK	94
6.7.3.74 STM32_I2SSRC_PLLI2S	94
6.7.3.75 STM32_I2SSRC_CKIN	94
6.7.3.76 STM32_MCO1PRE_MASK	94
6.7.3.77 STM32_MCO1PRE_DIV1	94
6.7.3.78 STM32_MCO1PRE_DIV2	94
6.7.3.79 STM32_MCO1PRE_DIV3	94
6.7.3.80 STM32_MCO1PRE_DIV4	94
6.7.3.81 STM32_MCO1PRE_DIV5	94
6.7.3.82 STM32_MCO2PRE_MASK	94
6.7.3.83 STM32_MCO2PRE_DIV1	95
6.7.3.84 STM32_MCO2PRE_DIV2	95
6.7.3.85 STM32_MCO2PRE_DIV3	95
6.7.3.86 STM32_MCO2PRE_DIV4	95
6.7.3.87 STM32_MCO2PRE_DIV5	95
6.7.3.88 STM32_MCO2SEL_MASK	95
6.7.3.89 STM32_MCO2SEL_SYSCLK	95
6.7.3.90 STM32_MCO2SEL_PLLI2S	95
6.7.3.91 STM32_MCO2SEL_HSE	95
6.7.3.92 STM32_MCO2SEL_PLL	95
6.7.3.93 STM32_RTC_NOCLOCK	95
6.7.3.94 STM32_RTC_LSE	95
6.7.3.95 STM32_RTC_LSI	96
6.7.3.96 STM32_RTC_HSE	96
6.7.3.97 STM32_PLLI2SN_MASK	96
6.7.3.98 STM32_PLLI2SR_MASK	96
6.7.3.99 STM32_RTCSEL_MASK	96
6.7.3.100 STM32_RTCSEL_NOCLOCK	96
6.7.3.101 STM32_RTCSEL_LSE	96
6.7.3.102 STM32_RTCSEL_LSI	96
6.7.3.103 STM32_RTCSEL_HSEDIV	96

6.7.3.104 WWDG_IRQHandler	96
6.7.3.105 PVD_IRQHandler	96
6.7.3.106 TAMP_STAMP_IRQHandler	96
6.7.3.107 RTC_WKUP_IRQHandler	97
6.7.3.108 FLASH_IRQHandler	97
6.7.3.109 RCC_IRQHandler	97
6.7.3.110 EXTI0_IRQHandler	97
6.7.3.111 EXTI1_IRQHandler	97
6.7.3.112 EXTI2_IRQHandler	97
6.7.3.113 EXTI3_IRQHandler	97
6.7.3.114 EXTI4_IRQHandler	97
6.7.3.115 DMA1_Stream0_IRQHandler	97
6.7.3.116 DMA1_Stream1_IRQHandler	97
6.7.3.117 DMA1_Stream2_IRQHandler	97
6.7.3.118 DMA1_Stream3_IRQHandler	97
6.7.3.119 DMA1_Stream4_IRQHandler	98
6.7.3.120 DMA1_Stream5_IRQHandler	98
6.7.3.121 DMA1_Stream6_IRQHandler	98
6.7.3.122 ADC1_2_3_IRQHandler	98
6.7.3.123 CAN1_TX_IRQHandler	98
6.7.3.124 CAN1_RX0_IRQHandler	98
6.7.3.125 CAN1_RX1_IRQHandler	98
6.7.3.126 CAN1_SCE_IRQHandler	98
6.7.3.127 EXTI9_5_IRQHandler	98
6.7.3.128 TIM1_BRK_IRQHandler	98
6.7.3.129 TIM1_UP_IRQHandler	98
6.7.3.130 TIM1_TRG_COM_IRQHandler	98
6.7.3.131 TIM1_CC_IRQHandler	99
6.7.3.132 TIM2_IRQHandler	99
6.7.3.133 TIM3_IRQHandler	99
6.7.3.134 TIM4_IRQHandler	99
6.7.3.135 I2C1_EV_IRQHandler	99
6.7.3.136 I2C1_ER_IRQHandler	99
6.7.3.137 I2C2_EV_IRQHandler	99
6.7.3.138 I2C2_ER_IRQHandler	99
6.7.3.139 SPI1_IRQHandler	99
6.7.3.140 SPI2_IRQHandler	99
6.7.3.141 USART1_IRQHandler	99
6.7.3.142 USART2_IRQHandler	99
6.7.3.143 USART3_IRQHandler	100

6.7.3.144 EXTI15_10_IRQHandler	100
6.7.3.145 RTC_Alarm_IRQHandler	100
6.7.3.146 OTG_FS_WKUP_IRQHandler	100
6.7.3.147 TIM8_BRK_IRQHandler	100
6.7.3.148 TIM8_UP_IRQHandler	100
6.7.3.149 TIM8_TRG_COM_IRQHandler	100
6.7.3.150 TIM8_CC_IRQHandler	100
6.7.3.151 DMA1_Stream7_IRQHandler	100
6.7.3.152 FSMC_IRQHandler	100
6.7.3.153 SDIO_IRQHandler	100
6.7.3.154 TIM5_IRQHandler	100
6.7.3.155 SPI3_IRQHandler	101
6.7.3.156 UART4_IRQHandler	101
6.7.3.157 UART5_IRQHandler	101
6.7.3.158 TIM6_IRQHandler	101
6.7.3.159 TIM7_IRQHandler	101
6.7.3.160 DMA2_Stream0_IRQHandler	101
6.7.3.161 DMA2_Stream1_IRQHandler	101
6.7.3.162 DMA2_Stream2_IRQHandler	101
6.7.3.163 DMA2_Stream3_IRQHandler	101
6.7.3.164 DMA2_Stream4_IRQHandler	101
6.7.3.165 ETH_IRQHandler	101
6.7.3.166 ETH_WKUP_IRQHandler	101
6.7.3.167 CAN2_TX_IRQHandler	102
6.7.3.168 CAN2_RX0_IRQHandler	102
6.7.3.169 CAN2_RX1_IRQHandler	102
6.7.3.170 CAN2_SCE_IRQHandler	102
6.7.3.171 OTG_FS_IRQHandler	102
6.7.3.172 DMA2_Stream5_IRQHandler	102
6.7.3.173 DMA2_Stream6_IRQHandler	102
6.7.3.174 DMA2_Stream7_IRQHandler	102
6.7.3.175 USART6_IRQHandler	102
6.7.3.176 I2C3_EV_IRQHandler	102
6.7.3.177 I2C3_ER_IRQHandler	102
6.7.3.178 OTG_HS_EP1_OUT_IRQHandler	102
6.7.3.179 OTG_HS_EP1_IN_IRQHandler	103
6.7.3.180 OTG_HS_WKUP_IRQHandler	103
6.7.3.181 OTG_HS_IRQHandler	103
6.7.3.182 DCMI_IRQHandler	103
6.7.3.183 CRYP_IRQHandler	103

6.7.3.184 HASH_RNG_IRQHandler	103
6.7.3.185 FPU_IRQHandler	103
6.7.3.186 STM32_NO_INIT	103
6.7.3.187 STM32_VOS	103
6.7.3.188 STM32_PVD_ENABLE	103
6.7.3.189 STM32_PLS	103
6.7.3.190 STM32_HSI_ENABLED	104
6.7.3.191 STM32_LSI_ENABLED	104
6.7.3.192 STM32_HSE_ENABLED	104
6.7.3.193 STM32_LSE_ENABLED	104
6.7.3.194 STM32_CLOCK48_REQUIRED	104
6.7.3.195 STM32_SW	104
6.7.3.196 STM32_PLLSRC	104
6.7.3.197 STM32_PLLM_VALUE	104
6.7.3.198 STM32_PLLN_VALUE	104
6.7.3.199 STM32_PLLP_VALUE	105
6.7.3.200 STM32_PLLQ_VALUE	105
6.7.3.201 STM32_HPRE	105
6.7.3.202 STM32_PPREG1	105
6.7.3.203 STM32_PPREG2	105
6.7.3.204 STM32_RTCSEL	105
6.7.3.205 STM32_RTCPRE_VALUE	105
6.7.3.206 STM32_MCO1SEL	105
6.7.3.207 STM32_MCO1PRE	106
6.7.3.208 STM32_MCO2SEL	106
6.7.3.209 STM32_MCO2PRE	106
6.7.3.210 STM32_I2SSRC	106
6.7.3.211 STM32_PLLI2SN_VALUE	106
6.7.3.212 STM32_PLLI2SR_VALUE	106
6.7.3.213 STM32_SYSCLK_MAX	106
6.7.3.214 STM32_0WS_THRESHOLD	107
6.7.3.215 STM32_PLLM	107
6.7.3.216 STM32_PLLCLKIN	107
6.7.3.217 STM32_ACTIVATE_PLL	107
6.7.3.218 STM32_PLLN	107
6.7.3.219 STM32_PLLP	107
6.7.3.220 STM32_PLLQ	107
6.7.3.221 STM32_PLLVCO	107
6.7.3.222 STM32_PLLCLKOUT	107
6.7.3.223 STM32_SYSCLK	107

6.7.3.224 STM32_HCLK	107
6.7.3.225 STM32_PCLK1	108
6.7.3.226 STM32_PCLK2	108
6.7.3.227 STM32_ACTIVATE_PLLI2S	108
6.7.3.228 STM32_PLLI2SN	108
6.7.3.229 STM32_PLLI2SR	108
6.7.3.230 STM32_PLLI2SVCO	108
6.7.3.231 STM32_PLLI2SCLKOUT	108
6.7.3.232 STM32_MCO1DIVCLK	108
6.7.3.233 STM32_MCO1CLK	108
6.7.3.234 STM32_MCO2DIVCLK	108
6.7.3.235 STM32_MCO2CLK	108
6.7.3.236 STM32_RTCPRE	108
6.7.3.237 STM32_RTCPRE	109
6.7.3.238 STM32_HSEDIVCLK	109
6.7.3.239 STM32_RTCCLK	109
6.7.3.240 STM32_PLL48CLK	109
6.7.3.241 STM32_TIMCLK1	109
6.7.3.242 STM32_TIMCLK2	109
6.7.3.243 STM32_FLASHBITS	109
6.7.3.244 hal_lld_get_counter_value	109
6.7.3.245 hal_lld_get_counter_frequency	109
6.7.4 Typedef Documentation	110
6.7.4.1 halclock_t	110
6.7.4.2 halrtcnt_t	110
6.8 I2C Driver	110
6.8.1 Detailed Description	110
6.8.2 Driver State Machine	110
6.8.3 Function Documentation	112
6.8.3.1 i2cInit	112
6.8.3.2 i2cObjectInit	112
6.8.3.3 i2cStart	113
6.8.3.4 i2cStop	113
6.8.3.5 i2cGetErrors	113
6.8.3.6 i2cMasterTransmitTimeout	113
6.8.3.7 i2cMasterReceiveTimeout	114
6.8.3.8 i2cAcquireBus	114
6.8.3.9 i2cReleaseBus	115
6.8.4 Define Documentation	115
6.8.4.1 I2CD_NO_ERROR	115

6.8.4.2	I2CD_BUS_ERROR	115
6.8.4.3	I2CD_ARBITRATION_LOST	115
6.8.4.4	I2CD_ACK_FAILURE	115
6.8.4.5	I2CD_OVERRUN	115
6.8.4.6	I2CD_PEC_ERROR	115
6.8.4.7	I2CD_TIMEOUT	116
6.8.4.8	I2CD_SMB_ALERT	116
6.8.4.9	I2C_USE_MUTUAL_EXCLUSION	116
6.8.4.10	i2cMasterTransmit	116
6.8.4.11	i2cMasterReceive	116
6.8.5	Enumeration Type Documentation	116
6.8.5.1	i2cstate_t	116
6.9	ICU Driver	116
6.9.1	Detailed Description	116
6.9.2	Driver State Machine	117
6.9.3	ICU Operations.	117
6.9.4	Function Documentation	120
6.9.4.1	iculinit	120
6.9.4.2	icuObjectInit	120
6.9.4.3	icuStart	121
6.9.4.4	icuStop	121
6.9.4.5	icuEnable	121
6.9.4.6	icuDisable	122
6.9.4.7	icu_lld_init	122
6.9.4.8	icu_lld_start	123
6.9.4.9	icu_lld_stop	123
6.9.4.10	icu_lld_enable	123
6.9.4.11	icu_lld_disable	123
6.9.5	Variable Documentation	124
6.9.5.1	ICUD1	124
6.9.5.2	ICUD2	124
6.9.5.3	ICUD3	124
6.9.5.4	ICUD4	124
6.9.5.5	ICUD5	124
6.9.5.6	ICUD8	124
6.9.6	Define Documentation	125
6.9.6.1	icuEnableI	125
6.9.6.2	icuDisableI	125
6.9.6.3	icuGetWidthI	125
6.9.6.4	icuGetPeriodI	125

6.9.6.5	_icu_isr_invoke_width_cb	126
6.9.6.6	_icu_isr_invoke_period_cb	126
6.9.6.7	STM32_ICU_USE_TIM1	126
6.9.6.8	STM32_ICU_USE_TIM2	126
6.9.6.9	STM32_ICU_USE_TIM3	127
6.9.6.10	STM32_ICU_USE_TIM4	127
6.9.6.11	STM32_ICU_USE_TIM5	127
6.9.6.12	STM32_ICU_USE_TIM8	127
6.9.6.13	STM32_ICU_TIM1_IRQ_PRIORITY	127
6.9.6.14	STM32_ICU_TIM2_IRQ_PRIORITY	127
6.9.6.15	STM32_ICU_TIM3_IRQ_PRIORITY	127
6.9.6.16	STM32_ICU_TIM4_IRQ_PRIORITY	128
6.9.6.17	STM32_ICU_TIM5_IRQ_PRIORITY	128
6.9.6.18	STM32_ICU_TIM8_IRQ_PRIORITY	128
6.9.6.19	icu_lld_get_width	128
6.9.6.20	icu_lld_get_period	128
6.9.7	Typedef Documentation	128
6.9.7.1	ICUDriver	128
6.9.7.2	icucallback_t	128
6.9.7.3	icufreq_t	129
6.9.7.4	icucnt_t	129
6.9.8	Enumeration Type Documentation	129
6.9.8.1	icustate_t	129
6.9.8.2	icumode_t	129
6.10	MAC Driver	129
6.10.1	Detailed Description	129
6.10.2	Function Documentation	130
6.10.2.1	macInit	130
6.10.2.2	macObjectInit	131
6.10.2.3	macStart	131
6.10.2.4	macStop	131
6.10.2.5	macWaitTransmitDescriptor	131
6.10.2.6	macReleaseTransmitDescriptor	132
6.10.2.7	macWaitReceiveDescriptor	132
6.10.2.8	macReleaseReceiveDescriptor	132
6.10.2.9	macPollLinkStatus	133
6.10.3	Define Documentation	133
6.10.3.1	MAC_USE_EVENTS	133
6.10.3.2	macGetReceiveEventSource	133
6.10.3.3	macWriteTransmitDescriptor	133

6.10.3.4	macReadReceiveDescriptor	134
6.10.4	Typedef Documentation	134
6.10.4.1	MACDriver	134
6.10.5	Enumeration Type Documentation	134
6.10.5.1	macstate_t	134
6.11	MMC over SPI Driver	134
6.11.1	Detailed Description	134
6.11.2	Driver State Machine	135
6.11.3	Function Documentation	137
6.11.3.1	mmcInit	137
6.11.3.2	mmcObjectInit	137
6.11.3.3	mmcStart	137
6.11.3.4	mmcStop	137
6.11.3.5	mmcConnect	138
6.11.3.6	mmcDisconnect	139
6.11.3.7	mmcStartSequentialRead	139
6.11.3.8	mmcSequentialRead	140
6.11.3.9	mmcStopSequentialRead	141
6.11.3.10	mmcStartSequentialWrite	141
6.11.3.11	mmcSequentialWrite	142
6.11.3.12	mmcStopSequentialWrite	143
6.11.4	Define Documentation	144
6.11.4.1	MMC_SECTOR_SIZE	144
6.11.4.2	MMC_NICE_WAITING	144
6.11.4.3	MMC_POLLING_INTERVAL	144
6.11.4.4	MMC_POLLING_DELAY	144
6.11.4.5	mmcGetDriverState	144
6.11.4.6	mmclsWriteProtected	144
6.11.5	Typedef Documentation	145
6.11.5.1	mmcquery_t	145
6.11.6	Enumeration Type Documentation	145
6.11.6.1	mmcstate_t	145
6.12	PAL Driver	145
6.12.1	Detailed Description	145
6.12.2	Implementation Rules	145
6.12.2.1	Writing on input pads	145
6.12.2.2	Reading from output pads	146
6.12.2.3	Writing unused or unimplemented port bits	146
6.12.2.4	Reading from unused or unimplemented port bits	146
6.12.2.5	Reading or writing on pins associated to other functionalities	146

6.12.3 Function Documentation	150
6.12.3.1 palReadBus	150
6.12.3.2 palWriteBus	150
6.12.3.3 palSetBusMode	150
6.12.3.4 __pal_lld_init	151
6.12.3.5 __pal_lld_setgroupmode	151
6.12.4 Define Documentation	151
6.12.4.1 PAL_MODE_RESET	151
6.12.4.2 PAL_MODE_UNCONNECTED	152
6.12.4.3 PAL_MODE_INPUT	152
6.12.4.4 PAL_MODE_INPUT_PULLUP	152
6.12.4.5 PAL_MODE_INPUT_PULLDOWN	152
6.12.4.6 PAL_MODE_INPUT_ANALOG	152
6.12.4.7 PAL_MODE_OUTPUT_PUSHPULL	152
6.12.4.8 PAL_MODE_OUTPUT_OPENDRAIN	152
6.12.4.9 PAL_LOW	152
6.12.4.10 PAL_HIGH	152
6.12.4.11 PAL_PORT_BIT	152
6.12.4.12 PAL_GROUP_MASK	153
6.12.4.13 __IOBUS_DATA	153
6.12.4.14 IOBUS_DECL	153
6.12.4.15 pallnit	153
6.12.4.16 palReadPort	154
6.12.4.17 palReadLatch	154
6.12.4.18 palWritePort	154
6.12.4.19 palSetPort	155
6.12.4.20 palClearPort	155
6.12.4.21 palTogglePort	155
6.12.4.22 palReadGroup	156
6.12.4.23 palWriteGroup	156
6.12.4.24 palSetGroupMode	156
6.12.4.25 palReadPad	157
6.12.4.26 palWritePad	157
6.12.4.27 palSetPad	157
6.12.4.28 palClearPad	158
6.12.4.29 palTogglePad	158
6.12.4.30 palSetPadMode	159
6.12.4.31 PAL_MODE_ALTERNATE	159
6.12.4.32 PAL_MODE_RESET	159
6.12.4.33 PAL_MODE_UNCONNECTED	159

6.12.4.34 PAL_MODE_INPUT	159
6.12.4.35 PAL_MODE_INPUT_PULLUP	159
6.12.4.36 PAL_MODE_INPUT_PULLDOWN	160
6.12.4.37 PAL_MODE_INPUT_ANALOG	160
6.12.4.38 PAL_MODE_OUTPUT_PUSHPULL	160
6.12.4.39 PAL_MODE_OUTPUT_OPENDRAIN	160
6.12.4.40 PAL_IOPORTS_WIDTH	160
6.12.4.41 PAL_WHOLE_PORT	160
6.12.4.42 IOPORT1	160
6.12.4.43 IOPORT2	160
6.12.4.44 IOPORT3	160
6.12.4.45 IOPORT4	161
6.12.4.46 IOPORT5	161
6.12.4.47 IOPORT6	161
6.12.4.48 IOPORT7	161
6.12.4.49 IOPORT8	161
6.12.4.50 IOPORT9	161
6.12.4.51 pal_lld_init	161
6.12.4.52 pal_lld_readport	161
6.12.4.53 pal_lld_readlatch	162
6.12.4.54 pal_lld_writeport	162
6.12.4.55 pal_lld_setport	162
6.12.4.56 pal_lld_clearport	162
6.12.4.57 pal_lld_writegroup	163
6.12.4.58 pal_lld_setgroupmode	163
6.12.4.59 pal_lld_writepad	163
6.12.5 Typedef Documentation	163
6.12.5.1 ioportmask_t	164
6.12.5.2 iomode_t	164
6.12.5.3 ioportid_t	164
6.13 PWM Driver	164
6.13.1 Detailed Description	164
6.13.2 Driver State Machine	164
6.13.3 PWM Operations	164
6.13.4 Function Documentation	168
6.13.4.1 pwmInit	168
6.13.4.2 pwmObjectInit	168
6.13.4.3 pwmStart	169
6.13.4.4 pwmStop	169
6.13.4.5 pwmChangePeriod	169

6.13.4.6	pwmEnableChannel	170
6.13.4.7	pwmDisableChannel	171
6.13.4.8	pwm_lld_init	171
6.13.4.9	pwm_lld_start	172
6.13.4.10	pwm_lld_stop	172
6.13.4.11	pwm_lld_enable_channel	172
6.13.4.12	pwm_lld_disable_channel	172
6.13.5	Variable Documentation	173
6.13.5.1	PWMD1	173
6.13.5.2	PWMD2	173
6.13.5.3	PWMD3	173
6.13.5.4	PWMD4	173
6.13.5.5	PWMD5	174
6.13.5.6	PWMD8	174
6.13.6	Define Documentation	174
6.13.6.1	PWM_OUTPUT_MASK	174
6.13.6.2	PWM_OUTPUT_DISABLED	174
6.13.6.3	PWM_OUTPUT_ACTIVE_HIGH	174
6.13.6.4	PWM_OUTPUT_ACTIVE_LOW	174
6.13.6.5	PWM_FRACTION_TO_WIDTH	174
6.13.6.6	PWM_DEGREES_TO_WIDTH	175
6.13.6.7	PWM_PERCENTAGE_TO_WIDTH	175
6.13.6.8	pwmChangePeriodl	175
6.13.6.9	pwmEnableChannell	176
6.13.6.10	pwmDisableChannell	176
6.13.6.11	PWM_CHANNELS	177
6.13.6.12	PWM_COMPLEMENTARY_OUTPUT_MASK	177
6.13.6.13	PWM_COMPLEMENTARY_OUTPUT_DISABLED	177
6.13.6.14	PWM_COMPLEMENTARY_OUTPUT_ACTIVE_HIGH	177
6.13.6.15	PWM_COMPLEMENTARY_OUTPUT_ACTIVE_LOW	178
6.13.6.16	STM32_PWM_USE_ADVANCED	178
6.13.6.17	STM32_PWM_USE_TIM1	178
6.13.6.18	STM32_PWM_USE_TIM2	178
6.13.6.19	STM32_PWM_USE_TIM3	178
6.13.6.20	STM32_PWM_USE_TIM4	178
6.13.6.21	STM32_PWM_USE_TIM5	179
6.13.6.22	STM32_PWM_USE_TIM8	179
6.13.6.23	STM32_PWM_TIM1_IRQ_PRIORITY	179
6.13.6.24	STM32_PWM_TIM2_IRQ_PRIORITY	179
6.13.6.25	STM32_PWM_TIM3_IRQ_PRIORITY	179

6.13.6.26 STM32_PWM_TIM4_IRQ_PRIORITY	179
6.13.6.27 STM32_PWM_TIM5_IRQ_PRIORITY	179
6.13.6.28 STM32_PWM_TIM8_IRQ_PRIORITY	179
6.13.6.29 pwm_lld_change_period	179
6.13.7 TYPEDOC Documentation	180
6.13.7.1 PWMDriver	180
6.13.7.2 pwmcallback_t	180
6.13.7.3 pwmemode_t	180
6.13.7.4 pwmchannel_t	180
6.13.7.5 pwcnt_t	180
6.13.8 ENUMERATION Type Documentation	180
6.13.8.1 pwmstate_t	180
6.14 SERIAL Driver	181
6.14.1 DETAILED Description	181
6.14.2 DRIVER State Machine	181
6.14.3 FUNCTION Documentation	184
6.14.3.1 sdInit	184
6.14.3.2 sdObjectInit	185
6.14.3.3 sdStart	185
6.14.3.4 sdStop	186
6.14.3.5 sdIncomingData	186
6.14.3.6 sdRequestData	187
6.14.3.7 CH_IRQ_HANDLER	187
6.14.3.8 CH_IRQ_HANDLER	187
6.14.3.9 CH_IRQ_HANDLER	187
6.14.3.10 CH_IRQ_HANDLER	188
6.14.3.11 CH_IRQ_HANDLER	188
6.14.3.12 CH_IRQ_HANDLER	188
6.14.3.13 sd_lld_init	188
6.14.3.14 sd_lld_start	188
6.14.3.15 sd_lld_stop	189
6.14.4 VARIABLE Documentation	189
6.14.4.1 SD1	189
6.14.4.2 SD2	189
6.14.4.3 SD3	189
6.14.4.4 SD4	189
6.14.4.5 SD5	189
6.14.4.6 SD6	189
6.14.5 DEFINE Documentation	189
6.14.5.1 SD_PARITY_ERROR	189

6.14.5.2 SD_FRAMING_ERROR	189
6.14.5.3 SD_OVERRUN_ERROR	190
6.14.5.4 SD_NOISE_ERROR	190
6.14.5.5 SD_BREAK_DETECTED	190
6.14.5.6 SERIAL_DEFAULT_BITRATE	190
6.14.5.7 SERIAL_BUFFERS_SIZE	190
6.14.5.8 _serial_driver_methods	190
6.14.5.9 sdPutWouldBlock	190
6.14.5.10 sdGetWouldBlock	191
6.14.5.11 sdPut	191
6.14.5.12 sdPutTimeout	191
6.14.5.13 sdGet	192
6.14.5.14 sdGetTimeout	192
6.14.5.15 sdWrite	192
6.14.5.16 sdWriteTimeout	192
6.14.5.17 sdAsynchronousWrite	193
6.14.5.18 sdRead	193
6.14.5.19 sdReadTimeout	193
6.14.5.20 sdAsynchronousRead	194
6.14.5.21 STM32_SERIAL_USE_USART1	194
6.14.5.22 STM32_SERIAL_USE_USART2	194
6.14.5.23 STM32_SERIAL_USE_USART3	194
6.14.5.24 STM32_SERIAL_USE_UART4	194
6.14.5.25 STM32_SERIAL_USE_UART5	195
6.14.5.26 STM32_SERIAL_USE_USART6	195
6.14.5.27 STM32_SERIAL_USART1_PRIORITY	195
6.14.5.28 STM32_SERIAL_USART2_PRIORITY	195
6.14.5.29 STM32_SERIAL_USART3_PRIORITY	195
6.14.5.30 STM32_SERIAL_UART4_PRIORITY	195
6.14.5.31 STM32_SERIAL_UART5_PRIORITY	195
6.14.5.32 STM32_SERIAL_USART6_PRIORITY	195
6.14.5.33 _serial_driver_data	195
6.14.5.34 USART_CR2_STOP1_BITS	196
6.14.5.35 USART_CR2_STOP0P5_BITS	196
6.14.5.36 USART_CR2_STOP2_BITS	196
6.14.5.37 USART_CR2_STOP1P5_BITS	196
6.14.6 Typedef Documentation	196
6.14.6.1 SerialDriver	196
6.14.7 Enumeration Type Documentation	196
6.14.7.1 sdstate_t	196

6.15 SDC Driver	196
6.15.1 Detailed Description	196
6.15.2 Driver State Machine	197
6.15.3 SDC Operations.	197
6.15.4 Function Documentation	199
6.15.4.1 sdcInit	199
6.15.4.2 sdcObjectInit	199
6.15.4.3 sdcStart	199
6.15.4.4 sdcStop	199
6.15.4.5 sdcConnect	199
6.15.4.6 sdcDisconnect	200
6.15.4.7 sdcRead	200
6.15.4.8 sdcWrite	201
6.15.4.9 _sdc_wait_for_transfer_state	201
6.15.5 Define Documentation	202
6.15.5.1 SDC_BLOCK_SIZE	202
6.15.5.2 SDC_CMD8_PATTERN	202
6.15.5.3 SDC_MODE_CARDTYPE_MASK	202
6.15.5.4 SDC_MODE_CARDTYPE_SDV11	202
6.15.5.5 SDC_MODE_CARDTYPE_SDV20	202
6.15.5.6 SDC_MODE_CARDTYPE_MMC	202
6.15.5.7 SDC_MODE_HIGH_CAPACITY	202
6.15.5.8 SDC_R1_ERROR_MASK	202
6.15.5.9 SDC_INIT_RETRY	202
6.15.5.10 SDC_MMCI_SUPPORT	203
6.15.5.11 SDC_NICE_WAITING	203
6.15.5.12 SDC_R1_ERROR	203
6.15.5.13 SDC_R1_STS	203
6.15.5.14 SDC_R1_IS_CARD_LOCKED	203
6.15.5.15 sdcGetDriverState	203
6.15.5.16 sdclsCardInserted	204
6.15.5.17 sdclsWriteProtected	204
6.15.6 Enumeration Type Documentation	204
6.15.6.1 sdcstate_t	204
6.16 SPI Driver	205
6.16.1 Detailed Description	205
6.16.2 Driver State Machine	205
6.16.3 Function Documentation	208
6.16.3.1 spilnit	208
6.16.3.2 spiObjectInit	209

6.16.3.3	spiStart	209
6.16.3.4	spiStop	209
6.16.3.5	spiSelect	210
6.16.3.6	spiUnselect	210
6.16.3.7	spiStartIgnore	210
6.16.3.8	spiStartExchange	211
6.16.3.9	spiStartSend	211
6.16.3.10	spiStartReceive	212
6.16.3.11	spilgnore	212
6.16.3.12	spiExchange	213
6.16.3.13	spiSend	213
6.16.3.14	spiReceive	213
6.16.3.15	spiAcquireBus	214
6.16.3.16	spiReleaseBus	214
6.16.3.17	spi_lld_init	214
6.16.3.18	spi_lld_start	215
6.16.3.19	spi_lld_stop	215
6.16.3.20	spi_lld_select	216
6.16.3.21	spi_lld_unselect	216
6.16.3.22	spi_lld_ignore	216
6.16.3.23	spi_lld_exchange	217
6.16.3.24	spi_lld_send	217
6.16.3.25	spi_lld_receive	217
6.16.3.26	spi_lld_polled_exchange	218
6.16.4	Variable Documentation	218
6.16.4.1	SPID1	218
6.16.4.2	SPID2	218
6.16.4.3	SPID3	218
6.16.5	Define Documentation	218
6.16.5.1	SPI_USE_WAIT	218
6.16.5.2	SPI_USE_MUTUAL_EXCLUSION	219
6.16.5.3	spiSelectl	219
6.16.5.4	spiUnselectl	219
6.16.5.5	spiStartIgnorel	219
6.16.5.6	spiStartExchangel	220
6.16.5.7	spiStartSendl	220
6.16.5.8	spiStartReceive l	221
6.16.5.9	spiPolledExchange l	222
6.16.5.10	_spi_wait_s	222
6.16.5.11	_spi_wakeup_isr	222

6.16.5.12 _spi_isr_code	223
6.16.5.13 STM32_SPI_USE_SPI1	223
6.16.5.14 STM32_SPI_USE_SPI2	224
6.16.5.15 STM32_SPI_USE_SPI3	224
6.16.5.16 STM32_SPI_SPI1_IRQ_PRIORITY	224
6.16.5.17 STM32_SPI_SPI2_IRQ_PRIORITY	224
6.16.5.18 STM32_SPI_SPI3_IRQ_PRIORITY	224
6.16.5.19 STM32_SPI_SPI1_DMA_PRIORITY	224
6.16.5.20 STM32_SPI_SPI2_DMA_PRIORITY	224
6.16.5.21 STM32_SPI_SPI3_DMA_PRIORITY	225
6.16.5.22 STM32_SPI_DMA_ERROR_HOOK	225
6.16.5.23 STM32_SPI_SPI1_RX_DMA_STREAM	225
6.16.5.24 STM32_SPI_SPI1_TX_DMA_STREAM	225
6.16.5.25 STM32_SPI_SPI2_RX_DMA_STREAM	225
6.16.5.26 STM32_SPI_SPI2_TX_DMA_STREAM	225
6.16.5.27 STM32_SPI_SPI3_RX_DMA_STREAM	225
6.16.5.28 STM32_SPI_SPI3_TX_DMA_STREAM	226
6.16.6 Typedef Documentation	226
6.16.6.1 SPIDriver	226
6.16.6.2 spicallback_t	226
6.16.7 Enumeration Type Documentation	226
6.16.7.1 spistate_t	226
6.17 Time Measurement Driver	226
6.17.1 Detailed Description	226
6.17.2 Function Documentation	227
6.17.2.1 tmInit	227
6.17.2.2 tmObjectInit	227
6.17.3 Define Documentation	227
6.17.3.1 tmStartMeasurement	228
6.17.3.2 tmStopMeasurement	228
6.17.4 Typedef Documentation	228
6.17.4.1 TimeMeasurement	228
6.18 UART Driver	228
6.18.1 Detailed Description	229
6.18.2 Driver State Machine	229
6.18.2.1 Transmitter sub State Machine	229
6.18.2.2 Receiver sub State Machine	230
6.18.3 Function Documentation	233
6.18.3.1 uartInit	233
6.18.3.2 uartObjectInit	234

6.18.3.3	uartStart	234
6.18.3.4	uartStop	234
6.18.3.5	uartStartSend	235
6.18.3.6	uartStartSendl	235
6.18.3.7	uartStopSend	236
6.18.3.8	uartStopSendl	236
6.18.3.9	uartStartReceive	237
6.18.3.10	uartStartReceive1	238
6.18.3.11	uartStopReceive	238
6.18.3.12	uartStopReceive1	239
6.18.3.13	CH_IRQ_HANDLER	240
6.18.3.14	CH_IRQ_HANDLER	240
6.18.3.15	CH_IRQ_HANDLER	240
6.18.3.16	uart_lld_init	240
6.18.3.17	uart_lld_start	240
6.18.3.18	uart_lld_stop	241
6.18.3.19	uart_lld_start_send	241
6.18.3.20	uart_lld_stop_send	242
6.18.3.21	uart_lld_start_receive	242
6.18.3.22	uart_lld_stop_receive	242
6.18.4	Variable Documentation	242
6.18.4.1	UARTD1	243
6.18.4.2	UARTD2	243
6.18.4.3	UARTD3	243
6.18.5	Define Documentation	243
6.18.5.1	UART_NO_ERROR	243
6.18.5.2	UART_PARITY_ERROR	243
6.18.5.3	UART_FRAMING_ERROR	243
6.18.5.4	UART_OVERRUN_ERROR	243
6.18.5.5	UART_NOISE_ERROR	243
6.18.5.6	UART_BREAK_DETECTED	243
6.18.5.7	STM32_UART_USE_USART1	243
6.18.5.8	STM32_UART_USE_USART2	244
6.18.5.9	STM32_UART_USE_USART3	244
6.18.5.10	STM32_UART_USART1_IRQ_PRIORITY	244
6.18.5.11	STM32_UART_USART2_IRQ_PRIORITY	244
6.18.5.12	STM32_UART_USART3_IRQ_PRIORITY	244
6.18.5.13	STM32_UART_USART1_DMA_PRIORITY	244
6.18.5.14	STM32_UART_USART2_DMA_PRIORITY	244
6.18.5.15	STM32_UART_USART3_DMA_PRIORITY	245

6.18.5.16 STM32_UART_DMA_ERROR_HOOK	245
6.18.5.17 STM32_UART_USART1_RX_DMA_STREAM	245
6.18.5.18 STM32_UART_USART1_TX_DMA_STREAM	245
6.18.5.19 STM32_UART_USART2_RX_DMA_STREAM	245
6.18.5.20 STM32_UART_USART2_TX_DMA_STREAM	245
6.18.5.21 STM32_UART_USART3_RX_DMA_STREAM	246
6.18.5.22 STM32_UART_USART3_TX_DMA_STREAM	246
6.18.6 Typedef Documentation	246
6.18.6.1 uartflags_t	246
6.18.6.2 UARTDriver	246
6.18.6.3 uartcb_t	246
6.18.6.4 uartccb_t	246
6.18.6.5 uarteccb_t	246
6.18.7 Enumeration Type Documentation	247
6.18.7.1 uartstate_t	247
6.18.7.2 uarttxstate_t	247
6.18.7.3 uartrxstate_t	247
6.19 STM32F4xx Drivers	247
6.19.1 Detailed Description	247
6.20 STM32F4xx Initialization Support	248
6.20.1 Supported HW resources	248
6.20.2 STM32F4xx HAL driver implementation features	248
6.21 STM32F4xx ADC Support	248
6.21.1 Supported HW resources	248
6.21.2 STM32F4xx ADC driver implementation features	248
6.22 STM32F4xx CAN Support	249
6.22.1 Supported HW resources	249
6.22.2 STM32F4xx CAN driver implementation features	249
6.23 STM32F4xx EXT Support	249
6.23.1 Supported HW resources	249
6.23.2 STM32F4xx EXT driver implementation features	249
6.24 STM32F4xx GPT Support	249
6.24.1 Supported HW resources	249
6.24.2 STM32F4xx GPT driver implementation features	250
6.25 STM32F4xx ICU Support	250
6.25.1 Supported HW resources	250
6.25.2 STM32F4xx ICU driver implementation features	250
6.26 STM32F4xx MAC Support	250
6.26.1 Supported HW resources	250
6.26.2 STM32F4xx MAC driver implementation features	250

6.27 STM32F4xx PAL Support	250
6.27.1 Supported HW resources	251
6.27.2 STM32F4xx PAL driver implementation features	251
6.27.3 Supported PAL setup modes	251
6.27.4 Suboptimal behavior	251
6.28 STM32F4xx PWM Support	252
6.28.1 Supported HW resources	252
6.28.2 STM32F4xx PWM driver implementation features	252
6.29 STM32F4xx SDC Support	252
6.29.1 Supported HW resources	252
6.29.2 STM32F4xx SDC driver implementation features	252
6.30 STM32F4xx Serial Support	252
6.30.1 Supported HW resources	253
6.30.2 STM32F4xx Serial driver implementation features	253
6.31 STM32F4xx SPI Support	253
6.31.1 Supported HW resources	253
6.31.2 STM32F4xx SPI driver implementation features	253
6.32 STM32F4xx UART Support	254
6.32.1 Supported HW resources	254
6.32.2 STM32F4xx UART driver implementation features	254
6.33 STM32F4xx Platform Drivers	254
6.33.1 Detailed Description	254
6.34 STM32F4xx DMA Support	254
6.34.1 Detailed Description	254
6.34.2 Supported HW resources	255
6.34.3 STM32F4xx DMA driver implementation features	255
6.34.4 Function Documentation	259
6.34.4.1 CH_IRQ_HANDLER	259
6.34.4.2 CH_IRQ_HANDLER	259
6.34.4.3 CH_IRQ_HANDLER	259
6.34.4.4 CH_IRQ_HANDLER	259
6.34.4.5 CH_IRQ_HANDLER	259
6.34.4.6 CH_IRQ_HANDLER	259
6.34.4.7 CH_IRQ_HANDLER	260
6.34.4.8 CH_IRQ_HANDLER	260
6.34.4.9 CH_IRQ_HANDLER	260
6.34.4.10 CH_IRQ_HANDLER	260
6.34.4.11 CH_IRQ_HANDLER	260
6.34.4.12 CH_IRQ_HANDLER	260
6.34.4.13 CH_IRQ_HANDLER	260

6.34.4.14 CH_IRQ_HANDLER	261
6.34.4.15 CH_IRQ_HANDLER	261
6.34.4.16 CH_IRQ_HANDLER	261
6.34.4.17 dmalInit	261
6.34.4.18 dmaStreamAllocate	261
6.34.4.19 dmaStreamRelease	262
6.34.5 Variable Documentation	262
6.34.5.1 _stm32_dma_streams	262
6.34.6 Define Documentation	263
6.34.6.1 STM32_DMA1_STREAMS_MASK	263
6.34.6.2 STM32_DMA2_STREAMS_MASK	263
6.34.6.3 STM32_DMA_CR_RESET_VALUE	263
6.34.6.4 STM32_DMA_FCR_RESET_VALUE	263
6.34.6.5 STM32_DMA_STREAMS	263
6.34.6.6 STM32_DMA_ISR_MASK	263
6.34.6.7 STM32_DMA_GETCHANNEL	263
6.34.6.8 STM32_DMA_STREAM_ID	264
6.34.6.9 STM32_DMA_STREAM_ID_MSK	264
6.34.6.10 STM32_DMA_IS_VALID_ID	264
6.34.6.11 STM32_DMA_STREAM	264
6.34.6.12 dmaStreamSetPeripheral	265
6.34.6.13 dmaStreamSetMemory0	265
6.34.6.14 dmaStreamSetMemory1	266
6.34.6.15 dmaStreamSetTransactionSize	266
6.34.6.16 dmaStreamGetTransactionSize	266
6.34.6.17 dmaStreamSetMode	267
6.34.6.18 dmaStreamSetFIFO	267
6.34.6.19 dmaStreamEnable	268
6.34.6.20 dmaStreamDisable	268
6.34.6.21 dmaStreamClearInterrupt	269
6.34.6.22 dmaStartMemcpy	269
6.34.6.23 dmaWaitCompletion	270
6.34.7 Typedef Documentation	270
6.34.7.1 stm32_dmaisr_t	270
6.35 STM32F4xx RCC Support	271
6.35.1 Detailed Description	271
6.35.2 Supported HW resources	271
6.35.3 STM32F4xx RCC driver implementation features	271
6.35.4 Define Documentation	276
6.35.4.1 rccEnableAPB1	276

6.35.4.2 rccDisableAPB1	276
6.35.4.3 rccResetAPB1	276
6.35.4.4 rccEnableAPB2	277
6.35.4.5 rccDisableAPB2	277
6.35.4.6 rccResetAPB2	277
6.35.4.7 rccEnableAHB1	278
6.35.4.8 rccDisableAHB1	278
6.35.4.9 rccResetAHB1	278
6.35.4.10 rccEnableAHB2	279
6.35.4.11 rccDisableAHB2	279
6.35.4.12 rccResetAHB2	279
6.35.4.13 rccEnableAHB3	280
6.35.4.14 rccDisableAHB3	280
6.35.4.15 rccResetAHB3	280
6.35.4.16 rccEnableADC1	281
6.35.4.17 rccDisableADC1	281
6.35.4.18 rccResetADC1	281
6.35.4.19 rccEnableADC2	281
6.35.4.20 rccDisableADC2	281
6.35.4.21 rccResetADC2	282
6.35.4.22 rccEnableADC3	282
6.35.4.23 rccDisableADC3	282
6.35.4.24 rccResetADC3	282
6.35.4.25 rccEnableDMA1	282
6.35.4.26 rccDisableDMA1	283
6.35.4.27 rccResetDMA1	283
6.35.4.28 rccEnableDMA2	283
6.35.4.29 rccDisableDMA2	283
6.35.4.30 rccResetDMA2	283
6.35.4.31 rccEnablePWRInterface	284
6.35.4.32 rccDisablePWRInterface	284
6.35.4.33 rccResetPWRInterface	284
6.35.4.34 rccEnableCAN1	284
6.35.4.35 rccDisableCAN1	284
6.35.4.36 rccResetCAN1	285
6.35.4.37 rccEnableCAN2	285
6.35.4.38 rccDisableCAN2	285
6.35.4.39 rccResetCAN2	285
6.35.4.40 rccEnableETH	285
6.35.4.41 rccDisableETH	286

6.35.4.42 rccResetETH	286
6.35.4.43 rccEnableI2C1	286
6.35.4.44 rccDisableI2C1	286
6.35.4.45 rccResetI2C1	287
6.35.4.46 rccEnableI2C2	287
6.35.4.47 rccDisableI2C2	287
6.35.4.48 rccResetI2C2	287
6.35.4.49 rccEnableI2C3	287
6.35.4.50 rccDisableI2C3	287
6.35.4.51 rccResetI2C3	288
6.35.4.52 rccEnableOTG_FS	288
6.35.4.53 rccDisableOTG_FS	288
6.35.4.54 rccResetOTG_FS	288
6.35.4.55 rccEnableSDIO	288
6.35.4.56 rccDisableSDIO	289
6.35.4.57 rccResetSDIO	289
6.35.4.58 rccEnableSPI1	289
6.35.4.59 rccDisableSPI1	289
6.35.4.60 rccResetSPI1	290
6.35.4.61 rccEnableSPI2	290
6.35.4.62 rccDisableSPI2	290
6.35.4.63 rccResetSPI2	290
6.35.4.64 rccEnableSPI3	290
6.35.4.65 rccDisableSPI3	291
6.35.4.66 rccResetSPI3	291
6.35.4.67 rccEnableTIM1	291
6.35.4.68 rccDisableTIM1	291
6.35.4.69 rccResetTIM1	291
6.35.4.70 rccEnableTIM2	292
6.35.4.71 rccDisableTIM2	292
6.35.4.72 rccResetTIM2	292
6.35.4.73 rccEnableTIM3	292
6.35.4.74 rccDisableTIM3	292
6.35.4.75 rccResetTIM3	293
6.35.4.76 rccEnableTIM4	293
6.35.4.77 rccDisableTIM4	293
6.35.4.78 rccResetTIM4	293
6.35.4.79 rccEnableTIM5	293
6.35.4.80 rccDisableTIM5	294
6.35.4.81 rccResetTIM5	294

6.35.4.82 rccEnableTIM8	294
6.35.4.83 rccDisableTIM8	294
6.35.4.84 rccResetTIM8	295
6.35.4.85 rccEnableUSART1	295
6.35.4.86 rccDisableUSART1	295
6.35.4.87 rccResetUSART1	295
6.35.4.88 rccEnableUSART2	295
6.35.4.89 rccDisableUSART2	296
6.35.4.90 rccResetUSART2	296
6.35.4.91 rccEnableUSART3	296
6.35.4.92 rccDisableUSART3	296
6.35.4.93 rccResetUSART3	296
6.35.4.94 rccEnableUSART6	296
6.35.4.95 rccDisableUSART6	297
6.35.4.96 rccEnableUART4	297
6.35.4.97 rccDisableUART4	297
6.35.4.98 rccResetUART4	297
6.35.4.99 rccEnableUART5	298
6.35.4.100 rccDisableUART5	298
6.35.4.101 rccResetUART5	298
6.35.4.102 rccResetUSART6	298
7 Data Structure Documentation	299
7.1 ADCConfig Struct Reference	299
7.1.1 Detailed Description	299
7.2 ADCConversionGroup Struct Reference	299
7.2.1 Detailed Description	299
7.2.2 Field Documentation	301
7.2.2.1 circular	301
7.2.2.2 num_channels	301
7.2.2.3 end_cb	301
7.2.2.4 error_cb	301
7.2.2.5 cr1	301
7.2.2.6 cr2	301
7.2.2.7 smpr1	301
7.2.2.8 smpr2	302
7.2.2.9 sqr1	302
7.2.2.10 sqr2	302
7.2.2.11 sqr3	302
7.3 ADCDriver Struct Reference	302

7.3.1	Detailed Description	302
7.3.2	Field Documentation	304
7.3.2.1	state	304
7.3.2.2	config	304
7.3.2.3	samples	304
7.3.2.4	depth	304
7.3.2.5	grpp	304
7.3.2.6	thread	304
7.3.2.7	mutex	304
7.3.2.8	adc	305
7.3.2.9	dmastp	305
7.3.2.10	dmamode	305
7.4	EXTChannelConfig Struct Reference	305
7.4.1	Detailed Description	305
7.4.2	Field Documentation	306
7.4.2.1	mode	306
7.4.2.2	cb	306
7.5	EXTConfig Struct Reference	306
7.5.1	Detailed Description	306
7.5.2	Field Documentation	307
7.5.2.1	channels	307
7.5.2.2	exti	307
7.6	EXTDriver Struct Reference	308
7.6.1	Detailed Description	308
7.6.2	Field Documentation	308
7.6.2.1	state	308
7.6.2.2	config	309
7.7	GPIO_TypeDef Struct Reference	309
7.7.1	Detailed Description	309
7.8	GPTConfig Struct Reference	309
7.8.1	Detailed Description	309
7.8.2	Field Documentation	311
7.8.2.1	frequency	311
7.8.2.2	callback	311
7.9	GPTDriver Struct Reference	311
7.9.1	Detailed Description	311
7.9.2	Field Documentation	313
7.9.2.1	state	313
7.9.2.2	config	313
7.9.2.3	clock	313

7.9.2.4	tim	313
7.10	ICUConfig Struct Reference	313
7.10.1	Detailed Description	313
7.10.2	Field Documentation	315
7.10.2.1	mode	315
7.10.2.2	frequency	315
7.10.2.3	width_cb	315
7.10.2.4	period_cb	315
7.11	ICUDriver Struct Reference	315
7.11.1	Detailed Description	315
7.11.2	Field Documentation	317
7.11.2.1	state	317
7.11.2.2	config	317
7.11.2.3	clock	317
7.11.2.4	tim	317
7.12	IOBus Struct Reference	317
7.12.1	Detailed Description	317
7.12.2	Field Documentation	318
7.12.2.1	portid	318
7.12.2.2	mask	318
7.12.2.3	offset	319
7.13	MMCConfig Struct Reference	319
7.13.1	Detailed Description	319
7.14	MMCDriver Struct Reference	319
7.14.1	Detailed Description	319
7.14.2	Field Documentation	321
7.14.2.1	state	321
7.14.2.2	config	321
7.14.2.3	spip	321
7.14.2.4	lscfg	321
7.14.2.5	hscfg	321
7.14.2.6	is_protected	321
7.14.2.7	is_inserted	321
7.14.2.8	inserted_event	322
7.14.2.9	removed_event	322
7.14.2.10	vt	322
7.14.2.11	cnt	322
7.15	PALConfig Struct Reference	322
7.15.1	Detailed Description	322
7.15.2	Field Documentation	323

7.15.2.1	PAData	323
7.15.2.2	PBData	323
7.15.2.3	PCData	324
7.15.2.4	PDData	324
7.16	PWMChannelConfig Struct Reference	324
7.16.1	Detailed Description	324
7.16.2	Field Documentation	326
7.16.2.1	mode	326
7.16.2.2	callback	326
7.17	PWMConfig Struct Reference	326
7.17.1	Detailed Description	326
7.17.2	Field Documentation	328
7.17.2.1	frequency	328
7.17.2.2	period	328
7.17.2.3	callback	328
7.17.2.4	channels	328
7.17.2.5	cr2	328
7.17.2.6	bdtr	329
7.18	PWMDriver Struct Reference	329
7.18.1	Detailed Description	329
7.18.2	Field Documentation	331
7.18.2.1	state	331
7.18.2.2	config	331
7.18.2.3	period	331
7.18.2.4	clock	331
7.18.2.5	tim	331
7.19	SerialConfig Struct Reference	331
7.19.1	Detailed Description	331
7.19.2	Field Documentation	332
7.19.2.1	sc_speed	332
7.19.2.2	sc_cr1	332
7.19.2.3	sc_cr2	332
7.19.2.4	sc_cr3	332
7.20	SerialDriver Struct Reference	332
7.20.1	Detailed Description	332
7.20.2	Field Documentation	333
7.20.2.1	vmt	333
7.21	SerialDriverVMT Struct Reference	333
7.21.1	Detailed Description	333
7.22	SPIConfig Struct Reference	333

7.22.1	Detailed Description	333
7.22.2	Field Documentation	335
7.22.2.1	end_cb	335
7.22.2.2	ssport	335
7.22.2.3	sspad	335
7.22.2.4	cr1	335
7.23	SPIDriver Struct Reference	335
7.23.1	Detailed Description	335
7.23.2	Field Documentation	337
7.23.2.1	state	337
7.23.2.2	config	337
7.23.2.3	thread	337
7.23.2.4	mutex	337
7.23.2.5	spi	337
7.23.2.6	dmarx	337
7.23.2.7	dmatx	337
7.23.2.8	rxdmamode	337
7.23.2.9	txdmamode	338
7.24	stm32_dma_stream_t Struct Reference	338
7.24.1	Detailed Description	338
7.24.2	Field Documentation	338
7.24.2.1	stream	338
7.24.2.2	ifcr	338
7.24.2.3	ishift	338
7.24.2.4	selfindex	338
7.24.2.5	vector	338
7.25	stm32_gpio_setup_t Struct Reference	339
7.25.1	Detailed Description	339
7.25.2	Field Documentation	339
7.25.2.1	moder	339
7.25.2.2	otyper	339
7.25.2.3	ospeedr	339
7.25.2.4	pupdr	339
7.25.2.5	odr	339
7.25.2.6	afrl	339
7.25.2.7	afrh	339
7.26	stm32_tim_t Struct Reference	340
7.26.1	Detailed Description	340
7.27	TimeMeasurement Struct Reference	340
7.27.1	Detailed Description	340

7.27.2 Field Documentation	340
7.27.2.1 start	340
7.27.2.2 stop	340
7.27.2.3 last	340
7.27.2.4 worst	340
7.27.2.5 best	341
7.28 UARTConfig Struct Reference	341
7.28.1 Detailed Description	341
7.28.2 Field Documentation	342
7.28.2.1 txend1_cb	342
7.28.2.2 txend2_cb	342
7.28.2.3 rxend_cb	342
7.28.2.4 rxchar_cb	342
7.28.2.5 rxerr_cb	342
7.28.2.6 speed	342
7.28.2.7 cr1	342
7.28.2.8 cr2	343
7.28.2.9 cr3	343
7.29 UARTDriver Struct Reference	343
7.29.1 Detailed Description	343
7.29.2 Field Documentation	345
7.29.2.1 state	345
7.29.2.2 txstate	345
7.29.2.3 rxstate	345
7.29.2.4 config	345
7.29.2.5 usart	345
7.29.2.6 dmamode	345
7.29.2.7 dmarx	345
7.29.2.8 dmatx	345
7.29.2.9 rdbuf	346
8 File Documentation	347
8.1 adc.c File Reference	347
8.1.1 Detailed Description	347
8.2 adc.h File Reference	348
8.2.1 Detailed Description	348
8.3 adc_lld.c File Reference	349
8.3.1 Detailed Description	349
8.4 adc_lld.h File Reference	350
8.4.1 Detailed Description	350

8.5	can.c File Reference	354
8.5.1	Detailed Description	354
8.6	can.h File Reference	355
8.6.1	Detailed Description	355
8.7	ext.c File Reference	356
8.7.1	Detailed Description	356
8.8	ext_lld.c File Reference	356
8.8.1	Detailed Description	356
8.9	ext_lld.h File Reference	357
8.9.1	Detailed Description	357
8.10	gpt.c File Reference	359
8.10.1	Detailed Description	359
8.11	gpt.h File Reference	359
8.11.1	Detailed Description	359
8.12	gpt_lld.c File Reference	360
8.12.1	Detailed Description	360
8.13	gpt_lld.h File Reference	361
8.13.1	Detailed Description	361
8.14	hal.c File Reference	362
8.14.1	Detailed Description	362
8.15	hal.h File Reference	363
8.15.1	Detailed Description	363
8.16	hal_lld.c File Reference	364
8.16.1	Detailed Description	364
8.17	hal_lld.h File Reference	364
8.17.1	Detailed Description	364
8.18	halconf.h File Reference	372
8.18.1	Detailed Description	372
8.19	i2c.c File Reference	374
8.19.1	Detailed Description	374
8.20	i2c.h File Reference	375
8.20.1	Detailed Description	375
8.21	icu.c File Reference	376
8.21.1	Detailed Description	376
8.22	icu.h File Reference	376
8.22.1	Detailed Description	376
8.23	icu_lld.c File Reference	378
8.23.1	Detailed Description	378
8.24	icu_lld.h File Reference	378
8.24.1	Detailed Description	378

8.25 mac.c File Reference	380
8.25.1 Detailed Description	380
8.26 mac.h File Reference	380
8.26.1 Detailed Description	380
8.27 mmc_spi.c File Reference	381
8.27.1 Detailed Description	381
8.28 mmc_spi.h File Reference	382
8.28.1 Detailed Description	382
8.29 pal.c File Reference	383
8.29.1 Detailed Description	383
8.30 pal.h File Reference	384
8.30.1 Detailed Description	384
8.31 pal_Ild.c File Reference	385
8.31.1 Detailed Description	385
8.32 pal_Ild.h File Reference	386
8.32.1 Detailed Description	386
8.33 pwm.c File Reference	388
8.33.1 Detailed Description	388
8.34 pwm.h File Reference	388
8.34.1 Detailed Description	388
8.35 pwm_Ild.c File Reference	390
8.35.1 Detailed Description	390
8.36 pwm_Ild.h File Reference	390
8.36.1 Detailed Description	390
8.37 sdc.c File Reference	392
8.37.1 Detailed Description	392
8.38 sdc.h File Reference	393
8.38.1 Detailed Description	393
8.39 serial.c File Reference	394
8.39.1 Detailed Description	394
8.40 serial.h File Reference	394
8.40.1 Detailed Description	394
8.41 serial_Ild.c File Reference	396
8.41.1 Detailed Description	396
8.42 serial_Ild.h File Reference	397
8.42.1 Detailed Description	397
8.43 spi.c File Reference	398
8.43.1 Detailed Description	398
8.44 spi.h File Reference	399
8.44.1 Detailed Description	399

8.45 spi_lld.c File Reference	400
8.45.1 Detailed Description	400
8.46 spi_lld.h File Reference	401
8.46.1 Detailed Description	401
8.47 stm32.h File Reference	403
8.47.1 Detailed Description	403
8.48 stm32_dma.c File Reference	403
8.48.1 Detailed Description	403
8.49 stm32_dma.h File Reference	405
8.49.1 Detailed Description	405
8.50 stm32_rcc.h File Reference	407
8.50.1 Detailed Description	407
8.51 tm.c File Reference	412
8.51.1 Detailed Description	412
8.52 tm.h File Reference	412
8.52.1 Detailed Description	412
8.53 uart.c File Reference	413
8.53.1 Detailed Description	413
8.54 uart.h File Reference	413
8.54.1 Detailed Description	413
8.55 uart_lld.c File Reference	415
8.55.1 Detailed Description	415
8.56 uart_lld.h File Reference	415
8.56.1 Detailed Description	415

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 STM32F4xx 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	7
Configuration	9
ADC Driver	15
CAN Driver	42
EXT Driver	48
GPT Driver	59
HAL Driver	72
I2C Driver	110
ICU Driver	116
MAC Driver	129
MMC over SPI Driver	134
PAL Driver	145
PWM Driver	164
Serial Driver	181
SDC Driver	196
SPI Driver	205
Time Measurement Driver.	226
UART Driver	228
STM32F4xx Drivers	247
STM32F4xx Initialization Support	248
STM32F4xx ADC Support	248
STM32F4xx CAN Support	249
STM32F4xx EXT Support	249
STM32F4xx GPT Support	249
STM32F4xx ICU Support	250
STM32F4xx MAC Support	250
STM32F4xx PAL Support	250
STM32F4xx PWM Support	252
STM32F4xx SDC Support	252
STM32F4xx Serial Support	252
STM32F4xx SPI Support	253
STM32F4xx UART Support	254
STM32F4xx Platform Drivers	254
STM32F4xx DMA Support	254
STM32F4xx RCC Support	271

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

ADCConfig (Driver configuration structure)	299
ADCConversionGroup (Conversion group configuration structure)	299
ADCDriver (Structure representing an ADC driver)	302
EXTChannelConfig (Channel configuration structure)	305
EXTConfig (Driver configuration structure)	306
EXTDriver (Structure representing an EXT driver)	308
GPIO_TypeDef (STM32 GPIO registers block)	309
GPTConfig (Driver configuration structure)	309
GPTDriver (Structure representing a GPT driver)	311
ICUConfig (Driver configuration structure)	313
ICUDriver (Structure representing an ICU driver)	315
IOBus (I/O bus descriptor)	317
MMCConfig (Driver configuration structure)	319
MMCDriver (Structure representing a MMC driver)	319
PALConfig (STM32 GPIO static initializer)	322
PWMDriver (Structure representing a PWM driver)	324
PWMDriver (Structure representing a PWM driver)	326
SerialConfig (STM32 Serial Driver configuration structure)	329
SerialDriver (Full duplex serial driver class)	331
SerialDriverVMT (SerialDriver virtual methods table)	332
SPIConfig (Driver configuration structure)	333
SPIDriver (Structure representing a SPI driver)	335
stm32_dma_stream_t (STM32 DMA stream descriptor structure)	338
stm32_gpio_setup_t (GPIO port setup info)	339
stm32_tim_t (STM32 TIM registers block)	340
TimeMeasurement (Time Measurement structure)	340
UARTConfig (Driver configuration structure)	341
UARTDriver (Structure representing an UART driver)	343

Chapter 5

File Index

5.1 File List

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

adc.c (ADC Driver code)	347
adc.h (ADC Driver macros and structures)	348
adc_lld.c (STM32F4xx ADC subsystem low level driver source)	349
adc_lld.h (STM32F4xx ADC subsystem low level driver header)	350
can.c (CAN Driver code)	354
can.h (CAN Driver macros and structures)	355
ext.c (EXT Driver code)	356
ext_lld.c (STM32 EXT subsystem low level driver source)	356
ext_lld.h (STM32 EXT subsystem low level driver header)	357
gpt.c (GPT Driver code)	359
gpt.h (GPT Driver macros and structures)	359
gpt_lld.c (STM32 GPT subsystem low level driver source)	360
gpt_lld.h (STM32 GPT subsystem low level driver header)	361
hal.c (HAL subsystem code)	362
hal.h (HAL subsystem header)	363
hal_lld.c (STM32F4xx HAL subsystem low level driver source)	364
hal_lld.h (STM32F4xx HAL subsystem low level driver header)	364
halconf.h (HAL configuration header)	372
i2c.c (I2C Driver code)	374
i2c.h (I2C Driver macros and structures)	375
icu.c (ICU Driver code)	376
icu.h (ICU Driver macros and structures)	376
icu_lld.c (STM32 ICU subsystem low level driver header)	378
icu_lld.h (STM32 ICU subsystem low level driver header)	378
mac.c (MAC Driver code)	380
mac.h (MAC Driver macros and structures)	380
mmc_spi.c (MMC over SPI driver code)	381
mmc_spi.h (MMC over SPI driver header)	382
pal.c (I/O Ports Abstraction Layer code)	383
pal.h (I/O Ports Abstraction Layer macros, types and structures)	384
pal_lld.c (STM32L1xx/STM32F2xx/STM32F4xx GPIO low level driver code)	385
pal_lld.h (STM32L1xx/STM32F2xx/STM32F4xx GPIO low level driver header)	386
pwm.c (PWM Driver code)	388
pwm.h (PWM Driver macros and structures)	388
pwm_lld.c (STM32 PWM subsystem low level driver header)	390
pwm_lld.h (STM32 PWM subsystem low level driver header)	390
sdc.c (SDC Driver code)	392
sdc.h (SDC Driver macros and structures)	393

serial.c (Serial Driver code)	394
serial.h (Serial Driver macros and structures)	394
serial_lld.c (STM32 low level serial driver code)	396
serial_lld.h (STM32 low level serial driver header)	397
spi.c (SPI Driver code)	398
spi.h (SPI Driver macros and structures)	399
spi_lld.c (STM32 SPI subsystem low level driver source)	400
spi_lld.h (STM32 SPI subsystem low level driver header)	401
stm32.h (STM32 common header)	403
stm32_dma.c (Enhanced DMA helper driver code)	403
stm32_dma.h (Enhanced-DMA helper driver header)	405
stm32_rcc.h (RCC helper driver header)	407
tm.c (Time Measurement driver code)	412
tm.h (Time Measurement driver header)	412
uart.c (UART Driver code)	413
uart.h (UART Driver macros and structures)	413
uart_lld.c (STM32 low level UART driver code)	415
uart_lld.h (STM32 low level UART driver header)	415

Chapter 6

Module Documentation

6.1 HAL

6.1.1 Detailed Description

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

6.1.2 HAL Device Drivers Architecture

A device driver is usually split in two layers:

- High Level Device Driver (**HLD**). This layer contains the definitions of the driver's APIs and the platform independent part of the driver.

An HLD is composed by two files:

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

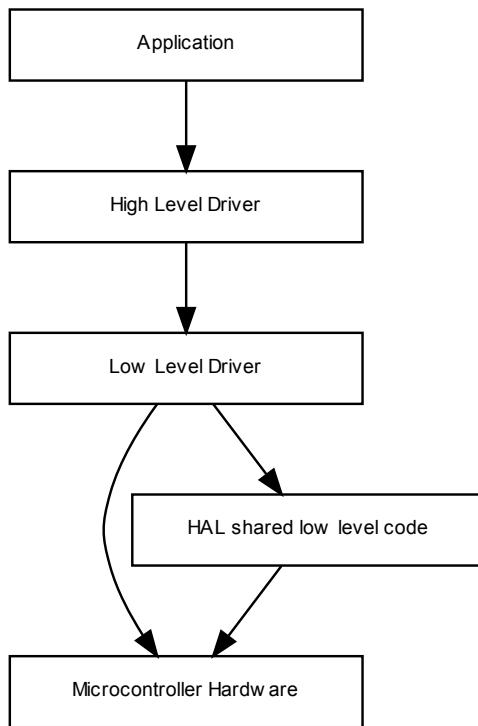
- Low Level Device Driver (**LLD**). This layer contains the platform dependent part of the driver.

A LLD is composed by two files:

- `<driver>_lld.c`, the LLD implementation file. This file must be included in the Makefile in order to use the driver.
- `<driver>_lld.h`, the LLD header file. This file is implicitly included by the HLD header file.

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

6.1.2.1 Diagram



Modules

- Configuration
HAL Driver Configuration.
- ADC Driver
Generic ADC Driver.
- CAN Driver
Generic CAN Driver.
- EXT Driver
Generic EXT Driver.
- GPT Driver
Generic GPT Driver.
- HAL Driver
Hardware Abstraction Layer.
- I2C Driver
Generic I2C Driver.
- ICU Driver
Generic ICU Driver.
- MAC Driver
Generic MAC 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.
 - [SDC Driver](#)
Generic SD Card Driver.
 - [SPI Driver](#)
Generic SPI Driver.
 - [Time Measurement Driver.](#)
Time Measurement unit.
 - [UART Driver](#)
Generic UART 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`

- `#define HAL_USE_SERIAL_USB TRUE`
Enables the SERIAL over USB subsystem.
- `#define HAL_USE_SPI TRUE`
Enables the SPI subsystem.
- `#define HAL_USE_UART TRUE`
Enables the UART subsystem.
- `#define HAL_USE_USB TRUE`
Enables the USB subsystem.

ADC driver related setting

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

CAN driver related setting

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

I2C driver related setting

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

MAC driver related setting

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

MMC_SPI driver related setting

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

SDC driver related setting

- `#define SDC_INIT_RETRY 100`
Number of initialization attempts before rejecting the card.
- `#define SDC_MMC_SUPPORT FALSE`
Include support for MMC cards.
- `#define SDC_NICE_WAITING TRUE`
Delays insertions.

SERIAL driver related setting

- `#define SERIAL_DEFAULT_BITRATE 38400`
Default bit rate.
- `#define SERIAL_BUFFERS_SIZE 16`
Serial buffers size.

SERIAL_USB driver related setting

- `#define SERIAL_USB_BUFFERS_SIZE 64`
Serial over USB buffers size.

SPI driver related setting

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

6.2.2 Define Documentation

6.2.2.1 `#define HAL_USE_PAL TRUE`

Enables the PAL subsystem.

6.2.2.2 `#define HAL_USE_ADC TRUE`

Enables the ADC subsystem.

6.2.2.3 `#define HAL_USE_CAN TRUE`

Enables the CAN subsystem.

6.2.2.4 `#define HAL_USE_EXT FALSE`

Enables the EXT subsystem.

6.2.2.5 `#define HAL_USE_GPT FALSE`

Enables the GPT subsystem.

6.2.2.6 #define HAL_USE_I2C FALSE

Enables the I2C subsystem.

6.2.2.7 #define HAL_USE_ICU FALSE

Enables the ICU subsystem.

6.2.2.8 #define HAL_USE_MAC TRUE

Enables the MAC subsystem.

6.2.2.9 #define HAL_USE_MMC_SPI TRUE

Enables the MMC_SPI subsystem.

6.2.2.10 #define HAL_USE_PWM TRUE

Enables the PWM subsystem.

6.2.2.11 #define HAL_USE_RTC FALSE

Enables the RTC subsystem.

6.2.2.12 #define HAL_USE_SDC FALSE

Enables the SDC subsystem.

6.2.2.13 #define HAL_USE_SERIAL TRUE

Enables the SERIAL subsystem.

6.2.2.14 #define HAL_USE_SERIAL_USB TRUE

Enables the SERIAL over USB subsystem.

6.2.2.15 #define HAL_USE_SPI TRUE

Enables the SPI subsystem.

6.2.2.16 #define HAL_USE_UART TRUE

Enables the UART subsystem.

6.2.2.17 #define HAL_USE_USB TRUE

Enables the USB subsystem.

6.2.2.18 #define ADC_USE_WAIT TRUE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

6.2.2.19 #define ADC_USE_MUTUAL_EXCLUSION TRUE

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

Note

Disabling this option saves both code and data space.

6.2.2.20 #define CAN_USE_SLEEP_MODE TRUE

Sleep mode related APIs inclusion switch.

6.2.2.21 #define I2C_USE_MUTUAL_EXCLUSION TRUE

Enables the mutual exclusion APIs on the I2C bus.

6.2.2.22 #define MAC_USE_EVENTS TRUE

Enables an event sources for incoming packets.

6.2.2.23 #define MMC_SECTOR_SIZE 512

Block size for MMC transfers.

6.2.2.24 #define MMC_NICE_WAITING TRUE

Delays insertions.

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

6.2.2.25 #define MMC_POLLING_INTERVAL 10

Number of positive insertion queries before generating the insertion event.

6.2.2.26 #define MMC_POLLING_DELAY 10

Interval, in milliseconds, between insertion queries.

6.2.2.27 #define MMC_USE_SPI_POLLING TRUE

Uses the SPI polled API for small data transfers.

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

6.2.2.28 #define SDC_INIT_RETRY 100

Number of initialization attempts before rejecting the card.

Note

Attempts are performed at 10mS intervals.

6.2.2.29 #define SDC_MMIC_SUPPORT FALSE

Include support for MMC cards.

Note

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

6.2.2.30 #define SDC_NICE_WAITING TRUE

Delays insertions.

If enabled this options inserts delays into the MMC waiting routines releasing some extra CPU time for the threads with lower priority, this may slow down the driver a bit however.

6.2.2.31 #define SERIAL_DEFAULT_BITRATE 38400

Default bit rate.

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

6.2.2.32 #define SERIAL_BUFFERS_SIZE 16

Serial buffers size.

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

Note

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

6.2.2.33 #define SERIAL_USB_BUFFERS_SIZE 64

Serial over USB buffers size.

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

Note

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

6.2.2.34 #define SPI_USE_WAIT TRUE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

6.2.2.35 #define SPI_USE_MUTUAL_EXCLUSION TRUE

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

Note

Disabling this option saves both code and data space.

6.3 ADC Driver

6.3.1 Detailed Description

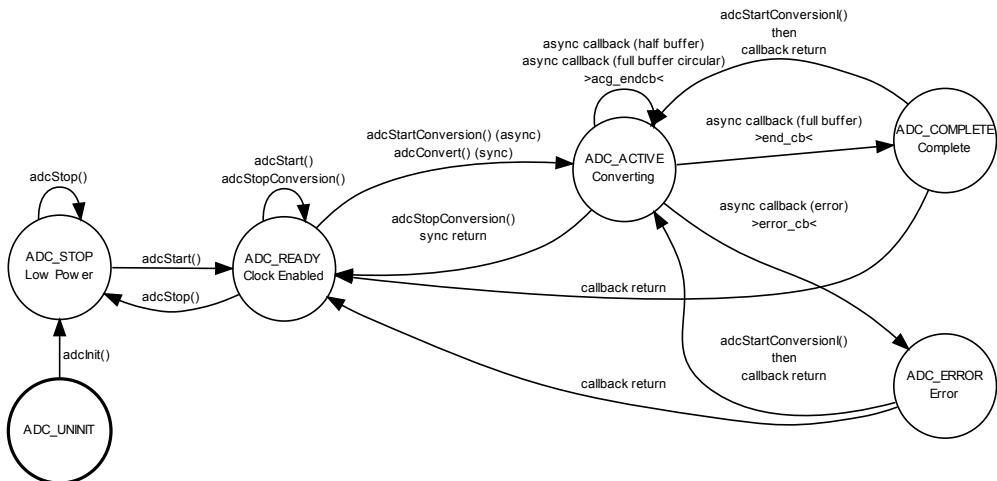
Generic ADC Driver. This module implements a generic ADC (Analog to Digital Converter) driver supporting a variety of buffer and conversion modes.

Precondition

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

6.3.2 Driver State Machine

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



6.3.3 ADC Operations

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

6.3.3.1 ADC Conversion Groups

The `ADCConversionGroup` is the objects that specifies a physical conversion operation. This structure contains some standard fields and several implementation-dependent fields.

The standard fields define the CG mode, the number of channels belonging to the CG and the optional callbacks.

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

6.3.3.2 ADC Conversion Modes

The driver supports several conversion modes:

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

6.3.3.3 ADC Callbacks

The driver is able to invoke callbacks during the conversion process. A callback is invoked when the operation has been completed or, in circular mode, when the buffer has been filled and the operation is restarted. In linear and circular modes a callback is also invoked when the buffer is half filled.

The "half filled" and "filled" callbacks in circular mode allow to implement "streaming processing" of the sampled data, while the driver is busy filling one half of the buffer the application can process the other half, this allows for continuous interleaved operations.

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

Data Structures

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

Functions

- void `adclnit` (void)
ADC Driver initialization.
- void `adcObjectInit` (`ADCDriver` *adcp)
Initializes the standard part of a `ADCDriver` structure.
- void `adcStart` (`ADCDriver` *adcp, const `ADCCConfig` *config)
Configures and activates the ADC peripheral.
- void `adcStop` (`ADCDriver` *adcp)
Deactivates the ADC peripheral.
- void `adcStartConversion` (`ADCDriver` *adcp, const `ADCConversionGroup` *grpp, `adcsample_t` *samples, `size_t` depth)

- 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.
- 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_2_3_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.
- void `adcSTM32EnableVBATE` (void)
 - Enables the VBATE bit.
- void `adcSTM32DisableVBATE` (void)
 - Disables the VBATE bit.

Variables

- `ADCDriver ADCD1`
 - *ADC1 driver identifier.*
- `ADCDriver ADCD2`
 - *ADC2 driver identifier.*
- `ADCDriver ADCD3`
 - *ADC3 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.

Absolute Maximum Ratings

- `#define STM32_ADCCLK_MIN 600000`
Minimum ADC clock frequency.
- `#define STM32_ADCCLK_MAX 36000000`
Maximum ADC clock frequency.

Triggers selection

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

ADC clock divider settings

- `#define ADC_CCR_ADCPRE_DIV2 0`
- `#define ADC_CCR_ADCPRE_DIV4 1`
- `#define ADC_CCR_ADCPRE_DIV6 2`
- `#define ADC_CCR_ADCPRE_DIV8 3`

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`

- #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_VBAT 18
VBAT.

Sampling rates

- #define ADC_SAMPLE_3 0
3 cycles sampling time.
- #define ADC_SAMPLE_15 1
15 cycles sampling time.
- #define ADC_SAMPLE_28 2
28 cycles sampling time.
- #define ADC_SAMPLE_56 3
56 cycles sampling time.
- #define ADC_SAMPLE_84 4
84 cycles sampling time.
- #define ADC_SAMPLE_112 5
112 cycles sampling time.
- #define ADC_SAMPLE_144 6
144 cycles sampling time.
- #define ADC_SAMPLE_480 7
480 cycles sampling time.

Configuration options

- `#define STM32_ADC_ADCPRE ADC_CCR_ADCPRE_DIV2`
ADC common clock divider.
- `#define STM32_ADC_USE_ADC1 TRUE`
ADC1 driver enable switch.
- `#define STM32_ADC_USE_ADC2 TRUE`
ADC2 driver enable switch.
- `#define STM32_ADC_USE_ADC3 TRUE`
ADC3 driver enable switch.
- `#define STM32_ADC_ADC1_DMA_STREAM STM32_DMA_STREAM_ID(2, 4)`
DMA stream used for ADC1 operations.
- `#define STM32_ADC_ADC2_DMA_STREAM STM32_DMA_STREAM_ID(2, 2)`
DMA stream used for ADC2 operations.
- `#define STM32_ADC_ADC3_DMA_STREAM STM32_DMA_STREAM_ID(2, 1)`
DMA stream used for ADC3 operations.
- `#define STM32_ADC_ADC1_DMA_PRIORITY 2`
ADC1 DMA priority (0..3|lowest..highest).
- `#define STM32_ADC_ADC2_DMA_PRIORITY 2`
ADC2 DMA priority (0..3|lowest..highest).
- `#define STM32_ADC_ADC3_DMA_PRIORITY 2`
ADC3 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.
- `#define STM32_ADC_ADC2_DMA_IRQ_PRIORITY 5`
ADC2 DMA interrupt priority level setting.
- `#define STM32_ADC_ADC3_DMA_IRQ_PRIORITY 5`
ADC3 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_SQR3_SQ1_N(n) ((n) << 0)`
1st channel in seq.
- `#define ADC_SQR3_SQ2_N(n) ((n) << 5)`
2nd channel in seq.
- `#define ADC_SQR3_SQ3_N(n) ((n) << 10)`
3rd channel in seq.
- `#define ADC_SQR3_SQ4_N(n) ((n) << 15)`
4th channel in seq.
- `#define ADC_SQR3_SQ5_N(n) ((n) << 20)`
5th channel in seq.
- `#define ADC_SQR3_SQ6_N(n) ((n) << 25)`
6th channel in seq.
- `#define ADC_SQR2_SQ7_N(n) ((n) << 0)`
7th channel in seq.
- `#define ADC_SQR2_SQ8_N(n) ((n) << 5)`
8th channel in seq.

- #define `ADC_SQR2_SQ9_N(n) ((n) << 10)`
9th channel in seq.
- #define `ADC_SQR2_SQ10_N(n) ((n) << 15)`
10th channel in seq.
- #define `ADC_SQR2_SQ11_N(n) ((n) << 20)`
11th channel in seq.
- #define `ADC_SQR2_SQ12_N(n) ((n) << 25)`
12th channel in seq.
- #define `ADC_SQR1_SQ13_N(n) ((n) << 0)`
13th channel in seq.
- #define `ADC_SQR1_SQ14_N(n) ((n) << 5)`
14th channel in seq.
- #define `ADC_SQR1_SQ15_N(n) ((n) << 10)`
15th channel in seq.
- #define `ADC_SQR1_SQ16_N(n) ((n) << 15)`
16th channel in seq.

Sampling rate settings helper macros

- #define `ADC_SMPR2_SMP_AN0(n) ((n) << 0)`
AN0 sampling time.
- #define `ADC_SMPR2_SMP_AN1(n) ((n) << 3)`
AN1 sampling time.
- #define `ADC_SMPR2_SMP_AN2(n) ((n) << 6)`
AN2 sampling time.
- #define `ADC_SMPR2_SMP_AN3(n) ((n) << 9)`
AN3 sampling time.
- #define `ADC_SMPR2_SMP_AN4(n) ((n) << 12)`
AN4 sampling time.
- #define `ADC_SMPR2_SMP_AN5(n) ((n) << 15)`
AN5 sampling time.
- #define `ADC_SMPR2_SMP_AN6(n) ((n) << 18)`
AN6 sampling time.
- #define `ADC_SMPR2_SMP_AN7(n) ((n) << 21)`
AN7 sampling time.
- #define `ADC_SMPR2_SMP_AN8(n) ((n) << 24)`
AN8 sampling time.
- #define `ADC_SMPR2_SMP_AN9(n) ((n) << 27)`
AN9 sampling time.
- #define `ADC_SMPR1_SMP_AN10(n) ((n) << 0)`
AN10 sampling time.
- #define `ADC_SMPR1_SMP_AN11(n) ((n) << 3)`
AN11 sampling time.
- #define `ADC_SMPR1_SMP_AN12(n) ((n) << 6)`
AN12 sampling time.
- #define `ADC_SMPR1_SMP_AN13(n) ((n) << 9)`
AN13 sampling time.
- #define `ADC_SMPR1_SMP_AN14(n) ((n) << 12)`
AN14 sampling time.
- #define `ADC_SMPR1_SMP_AN15(n) ((n) << 15)`
AN15 sampling time.

- `#define ADC_SMPR1_SMP_SENSOR(n) ((n) << 18)`
Temperature Sensor sampling time.
- `#define ADC_SMPR1_SMP_VREF(n) ((n) << 21)`
Voltage Reference sampling time.
- `#define ADC_SMPR1_SMP_VBAT(n) ((n) << 24)`
VBAT 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 adcInit(void)

ADC Driver initialization.

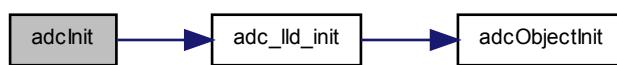
Note

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

Function Class:

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

Here is the call graph for this function:



6.3.4.2 void adcObjectInit (ADCDriver * *adcp*)

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

Parameters

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

Function Class:

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

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

Configures and activates the ADC peripheral.

Parameters

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

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

Function Class:

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

Here is the call graph for this function:

**6.3.4.4 void adcStop (ADCDriver * *adcp*)**

Deactivates the ADC peripheral.

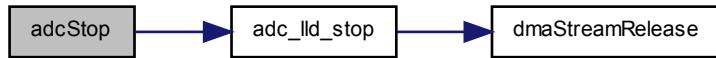
Parameters

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.

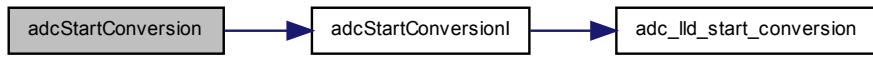
Parameters

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

Function Class:

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

Function Class:

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

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

Performs an ADC conversion.

Performs a synchronous conversion operation.

Note

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

Parameters

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

Returns

The operation result.

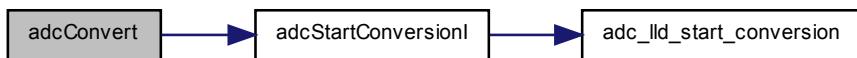
Return values

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

Function Class:

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

Here is the call graph for this function:



```
6.3.4.12 CH_IRQ_HANDLER( ADC1_2_3_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.

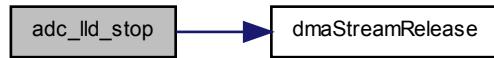
Parameters

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.

6.3.4.20 void adcSTM32EnableVBATE (void)

Enables the VBAT bit.

The VBAT bit is required in order to sample the VBAT channel.

Note

This is an STM32-only functionality.

6.3.4.21 void adcSTM32DisableVBATE (void)

Disables the VBAT bit.

The VBAT bit is required in order to sample the VBAT channel.

Note

This is an STM32-only functionality.

6.3.5 Variable Documentation

6.3.5.1 ADCDriver ADCD1

ADC1 driver identifier.

6.3.5.2 ADCDriver ADCD2

ADC2 driver identifier.

6.3.5.3 ADCDriver ADCD3

ADC3 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	<i>adcp</i> 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 STM32_ADCCLK_MIN 600000

Minimum ADC clock frequency.

6.3.6.11 #define STM32_ADCCLK_MAX 36000000

Maximum ADC clock frequency.

6.3.6.12 #define ADC_CR2_EXTSEL_SRC(n) ((n) << 24)

Trigger source.

6.3.6.13 #define ADC_CHANNEL_IN0 0

External analog input 0.

6.3.6.14 #define ADC_CHANNEL_IN1 1

External analog input 1.

6.3.6.15 #define ADC_CHANNEL_IN2 2

External analog input 2.

6.3.6.16 #define ADC_CHANNEL_IN3 3

External analog input 3.

6.3.6.17 #define ADC_CHANNEL_IN4 4

External analog input 4.

6.3.6.18 #define ADC_CHANNEL_IN5 5

External analog input 5.

6.3.6.19 #define ADC_CHANNEL_IN6 6

External analog input 6.

6.3.6.20 #define ADC_CHANNEL_IN7 7

External analog input 7.

6.3.6.21 #define ADC_CHANNEL_IN8 8

External analog input 8.

6.3.6.22 #define ADC_CHANNEL_IN9 9

External analog input 9.

6.3.6.23 #define ADC_CHANNEL_IN10 10

External analog input 10.

6.3.6.24 #define ADC_CHANNEL_IN11 11

External analog input 11.

6.3.6.25 #define ADC_CHANNEL_IN12 12

External analog input 12.

6.3.6.26 #define ADC_CHANNEL_IN13 13

External analog input 13.

6.3.6.27 #define ADC_CHANNEL_IN14 14

External analog input 14.

6.3.6.28 #define ADC_CHANNEL_IN15 15

External analog input 15.

6.3.6.29 #define ADC_CHANNEL_SENSOR 16

Internal temperature sensor.

Note

Available onADC1 only.

6.3.6.30 #define ADC_CHANNEL_VREFINT 17

Internal reference.

Note

Available onADC1 only.

6.3.6.31 #define ADC_CHANNEL_VBAT 18

VBAT.

Note

Available onADC1 only.

6.3.6.32 #define ADC_SAMPLE_3 0

3 cycles sampling time.

6.3.6.33 #define ADC_SAMPLE_15 1

15 cycles sampling time.

6.3.6.34 #define ADC_SAMPLE_28 2

28 cycles sampling time.

6.3.6.35 #define ADC_SAMPLE_56 3

56 cycles sampling time.

6.3.6.36 #define ADC_SAMPLE_84 4

84 cycles sampling time.

6.3.6.37 #define ADC_SAMPLE_112 5

112 cycles sampling time.

6.3.6.38 #define ADC_SAMPLE_144 6

144 cycles sampling time.

6.3.6.39 #define ADC_SAMPLE_480 7

480 cycles sampling time.

6.3.6.40 #define STM32_ADC_ADCPRE ADC_CCR_ADCPRE_DIV2

ADC common clock divider.

Note

This setting is influenced by the VDDA voltage and other external conditions, please refer to the STM32F4xx datasheet for more info.

See section 5.3.20 "12-bit ADC characteristics".

6.3.6.41 #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.

6.3.6.42 #define STM32_ADC_USE_ADC2 TRUE

ADC2 driver enable switch.

If set to TRUE the support for ADC2 is included.

Note

The default is TRUE.

6.3.6.43 #define STM32_ADC_USE_ADC3 TRUE

ADC3 driver enable switch.

If set to TRUE the support for ADC3 is included.

Note

The default is TRUE.

6.3.6.44 #define STM32_ADC_ADC1_DMA_STREAM STM32_DMA_STREAM_ID(2, 4)

DMA stream used for ADC1 operations.

6.3.6.45 #define STM32_ADC_ADC2_DMA_STREAM STM32_DMA_STREAM_ID(2, 2)

DMA stream used for ADC2 operations.

6.3.6.46 #define STM32_ADC_ADC3_DMA_STREAM STM32_DMA_STREAM_ID(2, 1)

DMA stream used for ADC3 operations.

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_ADC2_DMA_PRIORITY 2

ADC2 DMA priority (0..3|lowest..highest).

6.3.6.49 #define STM32_ADC_ADC3_DMA_PRIORITY 2

ADC3 DMA priority (0..3|lowest..highest).

6.3.6.50 #define STM32_ADC_IRQ_PRIORITY 5

ADC interrupt priority level setting.

Note

This setting is shared among ADC1, ADC2 and ADC3 because all ADCs share the same vector.

```
6.3.6.51 #define STM32_ADC_ADC1_DMA_IRQ_PRIORITY 5
```

ADC1 DMA interrupt priority level setting.

```
6.3.6.52 #define STM32_ADC_ADC2_DMA_IRQ_PRIORITY 5
```

ADC2 DMA interrupt priority level setting.

```
6.3.6.53 #define STM32_ADC_ADC3_DMA_IRQ_PRIORITY 5
```

ADC3 DMA interrupt priority level setting.

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

Number of channels in a conversion sequence.

```
6.3.6.55 #define ADC_SQR3_SQ1_N( n ) ((n) << 0)
```

1st channel in seq.

```
6.3.6.56 #define ADC_SQR3_SQ2_N( n ) ((n) << 5)
```

2nd channel in seq.

```
6.3.6.57 #define ADC_SQR3_SQ3_N( n ) ((n) << 10)
```

3rd channel in seq.

```
6.3.6.58 #define ADC_SQR3_SQ4_N( n ) ((n) << 15)
```

4th channel in seq.

```
6.3.6.59 #define ADC_SQR3_SQ5_N( n ) ((n) << 20)
```

5th channel in seq.

```
6.3.6.60 #define ADC_SQR3_SQ6_N( n ) ((n) << 25)
```

6th channel in seq.

```
6.3.6.61 #define ADC_SQR2_SQ7_N( n ) ((n) << 0)
```

7th channel in seq.

```
6.3.6.62 #define ADC_SQR2_SQ8_N( n ) ((n) << 5)
```

8th channel in seq.

6.3.6.63 #define ADC_SQR2_SQ9_N(n) ((n) << 10)

9th channel in seq.

6.3.6.64 #define ADC_SQR2_SQ10_N(n) ((n) << 15)

10th channel in seq.

6.3.6.65 #define ADC_SQR2_SQ11_N(n) ((n) << 20)

11th channel in seq.

6.3.6.66 #define ADC_SQR2_SQ12_N(n) ((n) << 25)

12th channel in seq.

6.3.6.67 #define ADC_SQR1_SQ13_N(n) ((n) << 0)

13th channel in seq.

6.3.6.68 #define ADC_SQR1_SQ14_N(n) ((n) << 5)

14th channel in seq.

6.3.6.69 #define ADC_SQR1_SQ15_N(n) ((n) << 10)

15th channel in seq.

6.3.6.70 #define ADC_SQR1_SQ16_N(n) ((n) << 15)

16th channel in seq.

6.3.6.71 #define ADC_SMPR2_SMP_AN0(n) ((n) << 0)

AN0 sampling time.

6.3.6.72 #define ADC_SMPR2_SMP_AN1(n) ((n) << 3)

AN1 sampling time.

6.3.6.73 #define ADC_SMPR2_SMP_AN2(n) ((n) << 6)

AN2 sampling time.

6.3.6.74 #define ADC_SMPR2_SMP_AN3(n) ((n) << 9)

AN3 sampling time.

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

AN4 sampling time.

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

AN5 sampling time.

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

AN6 sampling time.

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

AN7 sampling time.

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

AN8 sampling time.

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

AN9 sampling time.

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

AN10 sampling time.

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

AN11 sampling time.

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

AN12 sampling time.

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

AN13 sampling time.

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

AN14 sampling time.

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

AN15 sampling time.

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

Temperature Sensor sampling time.

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

Voltage Reference sampling time.

```
6.3.6.89 #define ADC_SMPR1_SMP_VBAT( n ) ((n) << 24)
```

VBAT 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	adcp	pointer to the ADCDriver object triggering the callback
in	err	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 CAN Driver

6.4.1 Detailed Description

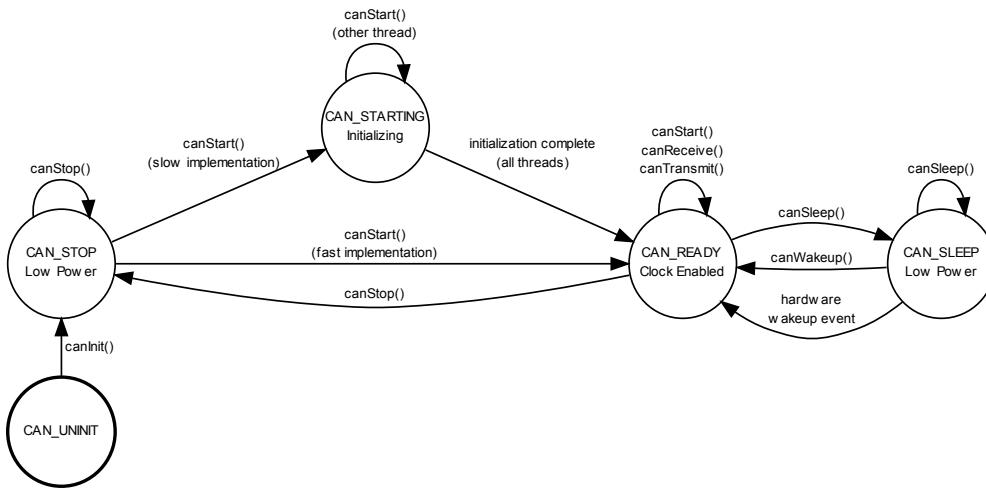
Generic CAN Driver. This module implements a generic CAN (Controller Area Network) driver allowing the exchange of information at frame level.

Precondition

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

6.4.2 Driver State Machine

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



Functions

- void **canInit** (void)

CAN Driver initialization.
- void **canObjectInit** (CANDriver *canp)

Initializes the standard part of a CANDriver structure.
- void **canStart** (CANDriver *canp, const CANConfig *config)

Configures and activates the CAN peripheral.
- void **canStop** (CANDriver *canp)

Deactivates the CAN peripheral.
- msg_t **canTransmit** (CANDriver *canp, const CANTxFrame *ctfp, systime_t timeout)

Can frame transmission.
- msg_t **canReceive** (CANDriver *canp, CANRxFrame *crfp, systime_t timeout)

Can frame receive.
- canstatus_t **canGetAndClearFlags** (CANDriver *canp)

Returns the current status mask and clears it.
- void **canSleep** (CANDriver *canp)

Enters the sleep mode.
- void **canWakeup** (CANDriver *canp)

Forces leaving the sleep mode.

CAN status flags

- #define **CAN_LIMIT_WARNING** 1

Errors rate warning.
- #define **CAN_LIMIT_ERROR** 2

Errors rate error.
- #define **CAN_BUS_OFF_ERROR** 4

Bus off condition reached.
- #define **CAN_FRAMING_ERROR** 8

- `#define CAN_OVERFLOW_ERROR 16`
Overflow in receive queue.

CAN configuration options

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

Macro Functions

- `#define canAddFlags1(canp, mask) ((canp)->status |= (mask))`
Adds some flags to the CAN status mask.

Enumerations

- enum `canstate_t` {

`CAN_UNINIT = 0, CAN_STOP = 1, CAN_STARTING = 2, CAN_READY = 3,`

`CAN_SLEEP = 4 }`

Driver state machine possible states.

6.4.3 Function Documentation

6.4.3.1 void canInit(void)

CAN Driver initialization.

Note

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

Function Class:

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

6.4.3.2 void canObjectInit(CANDriver * canp)

Initializes the standard part of a CANDriver structure.

Parameters

`out canp` pointer to the CANDriver object

Function Class:

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

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

Configures and activates the CAN peripheral.

Note

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

initialization to complete.

Parameters

in	<i>canp</i>	pointer to the CANDriver object
in	<i>config</i>	pointer to the CANConfig object. Depending on the implementation the value can be NULL.

Function Class:

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

6.4.3.4 void canStop (CANDriver * *canp*)

Deactivates the CAN peripheral.

Parameters

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

Function Class:

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

6.4.3.5 msg_t canTransmit (CANDriver * *canp*, const CANTxFrame * *ctfp*, systime_t *timeout*)

Can frame transmission.

The specified frame is queued for transmission, if the hardware queue is full then the invoking thread is queued.

Note

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

Parameters

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

Returns

The operation result.

Return values

RDY_OK the frame has been queued for transmission.

RDY_TIMEOUT The operation has timed out.

RDY_RESET The driver has been stopped while waiting.

Function Class:

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

6.4.3.6 `msg_t canReceive (CANDriver * canp, CANRxFrame * crfp, systime_t timeout)`

Can frame receive.

The function waits until a frame is received.

Note

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

Parameters

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

Returns

The operation result.

Return values

`RDY_OK` a frame has been received and placed in the buffer.

`RDY_TIMEOUT` The operation has timed out.

`RDY_RESET` The driver has been stopped while waiting.

Function Class:

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

6.4.3.7 `canstatus_t canGetAndClearFlags (CANDriver * canp)`

Returns the current status mask and clears it.

Parameters

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

Returns

The status flags mask.

Function Class:

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

6.4.3.8 `void canSleep (CANDriver * canp)`

Enters the sleep mode.

This function puts the CAN driver in sleep mode and broadcasts the `sleep_event` event source.

Precondition

In order to use this function the option `CAN_USE_SLEEP_MODE` must be enabled and the `CAN_SUPPORTS_SLEEP` mode must be supported by the low level driver.

Parameters

in *canp* pointer to the CANDriver object

Function Class:

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

6.4.3.9 void canWakeup (CANDriver * *canp*)

Enforces leaving the sleep mode.

Note

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

Parameters

in *canp* pointer to the CANDriver object

6.4.4 Define Documentation**6.4.4.1 #define CAN_LIMIT_WARNING 1**

Errors rate warning.

6.4.4.2 #define CAN_LIMIT_ERROR 2

Errors rate error.

6.4.4.3 #define CAN_BUS_OFF_ERROR 4

Bus off condition reached.

6.4.4.4 #define CAN_FRAMING_ERROR 8

Framing error of some kind on the CAN bus.

6.4.4.5 #define CAN_OVERFLOW_ERROR 16

Overflow in receive queue.

6.4.4.6 #define CAN_USE_SLEEP_MODE TRUE

Sleep mode related APIs inclusion switch.

This option can only be enabled if the CAN implementation supports the sleep mode, see the macro CAN_SUPPORTS_SLEEP exported by the underlying implementation.

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

Adds some flags to the CAN status mask.

Parameters

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

Function Class:

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

6.4.5 Enumeration Type Documentation**6.4.5.1 enum canstate_t**

Driver state machine possible states.

Enumerator:

CAN_UNINIT Not initialized.

CAN_STOP Stopped.

CAN_STARTING Starting.

CAN_READY Ready.

CAN_SLEEP Sleep state.

6.5 EXT Driver**6.5.1 Detailed Description**

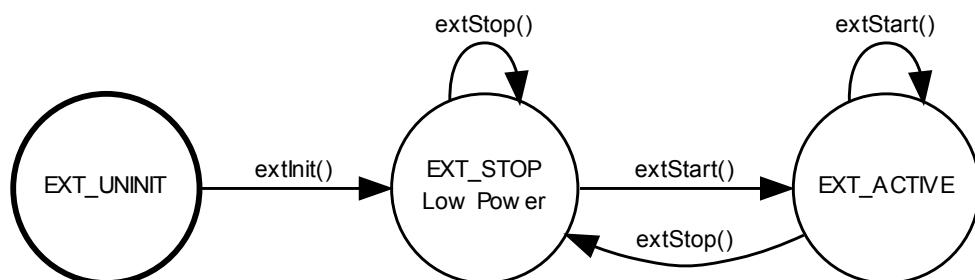
Generic EXT Driver. This module implements a generic EXT (EXTernal) driver.

Precondition

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

6.5.2 Driver State Machine

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



6.5.3 EXT Operations.

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

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

Data Structures

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

Functions

- void **extInit** (void)
EXT Driver initialization.
- void **extObjectInit** (EXTDriver *extp)
Initializes the standard part of a [EXTDriver](#) structure.
- void **extStart** (EXTDriver *extp, const EXTConfig *config)
Configures and activates the EXT peripheral.
- void **extStop** (EXTDriver *extp)
Deactivates the EXT peripheral.
- void **extChannelEnable** (EXTDriver *extp, expchannel_t channel)
Enables an EXT channel.
- void **extChannelDisable** (EXTDriver *extp, expchannel_t channel)
Disables an EXT channel.
- **CH_IRQ_HANDLER** (EXTI0_IRQHandler)
EXTI[0] interrupt handler.
- **CH_IRQ_HANDLER** (EXTI1_IRQHandler)
EXTI[1] interrupt handler.
- **CH_IRQ_HANDLER** (EXTI2_IRQHandler)
EXTI[2] interrupt handler.
- **CH_IRQ_HANDLER** (EXTI3_IRQHandler)
EXTI[3] interrupt handler.
- **CH_IRQ_HANDLER** (EXTI4_IRQHandler)
EXTI[4] interrupt handler.
- **CH_IRQ_HANDLER** (EXTI9_5_IRQHandler)
EXTI[5]...EXTI[9] interrupt handler.
- **CH_IRQ_HANDLER** (EXTI15_10_IRQHandler)
EXTI[10]...EXTI[15] interrupt handler.
- **CH_IRQ_HANDLER** (PVD_IRQHandler)
EXTI[16] interrupt handler (PVD).
- **CH_IRQ_HANDLER** (RTCAlarm_IRQHandler)

- **CH_IRQ_HANDLER** (USB_FS_WKUP_IRQHandler)

EXTI[17] interrupt handler (RTC).
- **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.

6.5.4 Function Documentation

6.5.4.1 void extInit(void)

EXT Driver initialization.

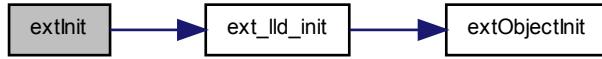
Note

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

Function Class:

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

Here is the call graph for this function:



6.5.4.2 void extObjectInit (EXTDriver * extp)

Initializes the standard part of a `EXTDriver` structure.

Parameters

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

Function Class:

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

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

Configures and activates the EXT peripheral.

Postcondition

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

Parameters

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

Function Class:

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

Here is the call graph for this function:



6.5.4.4 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:

**6.5.4.5 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.

6.5.4.6 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.

6.5.4.7 CH_IRQ_HANDLER (EXTI0_IRQHandler)

EXTI[0] interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.5.4.8 CH_IRQ_HANDLER (EXTI1_IRQHandler)

EXTI[1] interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.5.4.9 CH_IRQ_HANDLER (EXTI2_IRQHandler)

EXTI[2] interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.5.4.10 CH_IRQ_HANDLER (EXTI3_IRQHandler)

EXTI[3] interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.5.4.11 CH_IRQ_HANDLER (EXTI4_IRQHandler)

EXTI[4] interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.5.4.12 CH_IRQ_HANDLER (EXTI9_5_IRQHandler)

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

Function Class:

Interrupt handler, this function should not be directly invoked.

6.5.4.13 CH_IRQ_HANDLER (EXTI15_10_IRQHandler)

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

Function Class:

Interrupt handler, this function should not be directly invoked.

6.5.4.14 CH_IRQ_HANDLER (PVD_IRQHandler)

EXTI[16] interrupt handler (PVD).

Function Class:

Interrupt handler, this function should not be directly invoked.

6.5.4.15 CH_IRQ_HANDLER (RTCAlarm_IRQHandler)

EXTI[17] interrupt handler (RTC).

Function Class:

Interrupt handler, this function should not be directly invoked.

6.5.4.16 CH_IRQ_HANDLER (USB_FS_WKUP_IRQHandler)

EXTI[18] interrupt handler (USB_FS_WKUP).

Function Class:

Interrupt handler, this function should not be directly invoked.

6.5.4.17 void ext_lld_init (void)

Low level EXT driver initialization.

Function Class:

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

Here is the call graph for this function:

**6.5.4.18 void ext_lld_start (EXTDriver * extp)**

Configures and activates the EXT peripheral.

Parameters

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

Function Class:

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

6.5.4.19 void ext_lld_stop (EXTDriver * extp)

Deactivates the EXT peripheral.

Parameters

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

Function Class:

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

6.5.4.20 void ext_lld_channel_enable (EXTDriver * extp, expchannel_t channel)

Enables an EXT channel.

Parameters

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

Function Class:

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

6.5.4.21 void ext_lld_channel_disable (EXTDriver * extp, expchannel_t channel)

Disables an EXT channel.

Parameters

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

Function Class:

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

6.5.5 Variable Documentation**6.5.5.1 EXTDriver EXTD1**

EXTD1 driver identifier.

6.5.6 Define Documentation**6.5.6.1 #define EXT_MAX_CHANNELS STM32 EXTI_NUM_CHANNELS**

Available number of EXT channels.

6.5.6.2 #define EXT_CHANNELS_MASK ((1 << EXT_MAX_CHANNELS) - 1)

Mask of the available channels.

6.5.6.3 #define EXT_MODE_EXTI(m0, m1, m2, m3, m4, m5, m6, m7, m8, m9, m10, m11, m12, m13, m14, m15)**Value:**

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

EXTI-GPIO association macro.

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

6.5.6.4 #define EXT_MODE_GPIOA 0

GPIOA identifier.

6.5.6.5 #define EXT_MODE_GPIOB 1

GPIOB identifier.

6.5.6.6 #define EXT_MODE_GPIOC 2

GPIOC identifier.

6.5.6.7 #define EXT_MODE_GPIOD 3

GPIOD identifier.

6.5.6.8 #define EXT_MODE_GPIOE 4

GPIOE identifier.

6.5.6.9 #define EXT_MODE_GPIOF 5

GPIOF identifier.

6.5.6.10 #define EXT_MODE_GPIOG 6

GPIOG identifier.

6.5.6.11 #define EXT_MODE_GPIOH 7

GPIOH identifier.

6.5.6.12 #define EXT_MODE_GPIOI 8

GPIOI identifier.

6.5.6.13 #define STM32_EXT_EXTI0_IRQ_PRIORITY 6

EXTI0 interrupt priority level setting.

6.5.6.14 #define STM32_EXT_EXTI1_IRQ_PRIORITY 6

EXTI1 interrupt priority level setting.

6.5.6.15 #define STM32_EXT EXTI2_IRQ_PRIORITY 6

EXTI2 interrupt priority level setting.

6.5.6.16 #define STM32_EXT EXTI3_IRQ_PRIORITY 6

EXTI3 interrupt priority level setting.

6.5.6.17 #define STM32_EXT EXTI4_IRQ_PRIORITY 6

EXTI4 interrupt priority level setting.

6.5.6.18 #define STM32_EXT EXTI5_9_IRQ_PRIORITY 6

EXTI9..5 interrupt priority level setting.

6.5.6.19 #define STM32_EXT EXTI10_15_IRQ_PRIORITY 6

EXTI15..10 interrupt priority level setting.

6.5.6.20 #define STM32_EXT EXTI16_IRQ_PRIORITY 6

EXTI16 interrupt priority level setting.

6.5.6.21 #define STM32_EXT EXTI17_IRQ_PRIORITY 6

EXTI17 interrupt priority level setting.

6.5.6.22 #define STM32_EXT EXTI18_IRQ_PRIORITY 6

EXTI18 interrupt priority level setting.

6.5.6.23 #define STM32_EXT EXTI19_IRQ_PRIORITY 6

EXTI19 interrupt priority level setting.

6.5.6.24 #define STM32_EXT EXTI20_IRQ_PRIORITY 6

EXTI20 interrupt priority level setting.

6.5.6.25 #define STM32_EXT EXTI21_IRQ_PRIORITY 6

EXTI21 interrupt priority level setting.

6.5.6.26 #define STM32_EXT EXTI22_IRQ_PRIORITY 6

EXTI22 interrupt priority level setting.

6.5.7 Typedef Documentation

6.5.7.1 `typedef uint32_t expchannel_t`

EXT channel identifier.

6.5.7.2 `typedef void(* extcallback_t)(EXTDriver *extp, expchannel_t channel)`

Type of an EXT generic notification callback.

Parameters

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

6.6 GPT Driver

6.6.1 Detailed Description

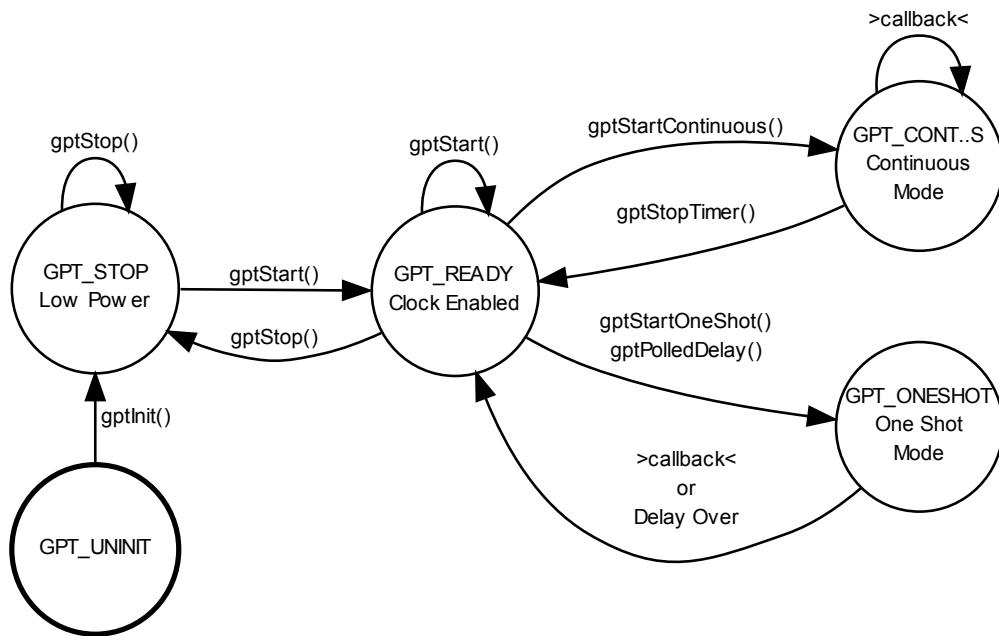
Generic GPT Driver. This module implements a generic GPT (General Purpose Timer) driver. The timer can be programmed in order to trigger callbacks after a specified time period or continuously with a specified interval.

Precondition

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

6.6.2 Driver State Machine

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



6.6.3 GPT Operations.

This driver abstracts a generic timer composed of:

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

The timer can operate in three different modes:

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

Data Structures

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

Functions

- void `gptInit` (void)

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

Initializes the standard part of a `GPTDriver` structure.
- void `gptStart` (`GPTDriver` *`gptp`, const `GPTConfig` *`config`)

Configures and activates the GPT peripheral.
- void `gptStop` (`GPTDriver` *`gptp`)

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

Starts the timer in continuous mode.
- void `gptStartContinuousl` (`GPTDriver` *`gptp`, `gptcnt_t` `interval`)

Starts the timer in continuous mode.
- void `gptStartOneShot` (`GPTDriver` *`gptp`, `gptcnt_t` `interval`)

Starts the timer in one shot mode.
- void `gptStartOneShotl` (`GPTDriver` *`gptp`, `gptcnt_t` `interval`)

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

Stops the timer.
- void `gptStopTimerl` (`GPTDriver` *`gptp`)

Stops the timer.
- void `gptPolledDelay` (`GPTDriver` *`gptp`, `gptcnt_t` `interval`)

Starts the timer in one shot mode and waits for completion.
- void `gpt_lld_init` (void)

Low level GPT driver initialization.
- void `gpt_lld_start` (`GPTDriver` *`gptp`)

Configures and activates the GPT peripheral.
- void `gpt_lld_stop` (`GPTDriver` *`gptp`)

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

Starts the timer in continuous mode.
- void `gpt_lld_stop_timer` (`GPTDriver` *`gptp`)

Stops the timer.
- void `gpt_lld_polled_delay` (`GPTDriver` *`gptp`, `gptcnt_t` `interval`)

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

Variables

- `GPTDriver GPTD1`

GPTD1 driver identifier.
- `GPTDriver GPTD2`

GPTD2 driver identifier.
- `GPTDriver GPTD3`

GPTD3 driver identifier.
- `GPTDriver GPTD4`

GPTD4 driver identifier.
- `GPTDriver GPTD5`

GPTD5 driver identifier.
- `GPTDriver GPTD8`

GPTD8 driver identifier.

Configuration options

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

Typedefs

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

Enumerations

- `enum gptstate_t {`
 `GPT_UNINIT = 0, GPT_STOP = 1, GPT_READY = 2, GPT_CONTINUOUS = 3,`
 `GPT_ONESHOT = 4 }`
Driver state machine possible states.

6.6.4 Function Documentation

6.6.4.1 void gptInit (void)

GPT Driver initialization.

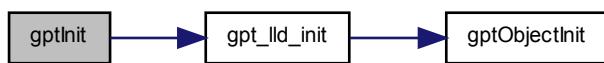
Note

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

Function Class:

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

Here is the call graph for this function:



6.6.4.2 void gptObjectInit (GPTDriver * gptp)

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

Parameters

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

Function Class:

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

6.6.4.3 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:



6.6.4.4 void gptStop (GPTDriver * *gptp*)

Deactivates the GPT peripheral.

Parameters

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

Function Class:

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

Here is the call graph for this function:



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

Starts the timer in continuous mode.

Parameters

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

Function Class:

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

Here is the call graph for this function:



6.6.4.6 void gptStartContinuous(GPTDriver * gptp, gptcnt_t interval)

Starts the timer in continuous mode.

Parameters

in	<i>gptp</i>	pointer to the 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:



6.6.4.7 void gptStartOneShot(GPTDriver * gptp, gptcnt_t interval)

Starts the timer in one shot mode.

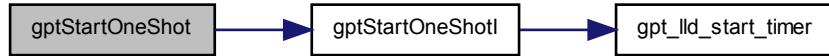
Parameters

in	<i>gptp</i>	pointer to the 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:



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

Starts the timer in one shot mode.

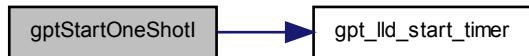
Parameters

in	<i>gptp</i>	pointer to the 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:



6.6.4.9 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:



6.6.4.10 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:



6.6.4.11 void gptPolledDelay (GPTDriver * *gptp*, gptcnt_t *interval*)

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

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

Note

The configured callback is not invoked when using this function.

Parameters

in	<i>gptp</i> pointer to the 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:



6.6.4.12 void gpt_lld_init (void)

Low level GPT driver initialization.

Function Class:

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

Here is the call graph for this function:



6.6.4.13 void gpt_lld_start (GPTDriver * gptp)

Configures and activates the GPT peripheral.

Parameters

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

Function Class:

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

6.6.4.14 void gpt_lld_stop (GPTDriver * gptp)

Deactivates the GPT peripheral.

Parameters

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

Function Class:

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

6.6.4.15 void gpt_lld_start_timer (*GPTDriver* * *gptp*, *gptcnt_t* *interval*)

Starts the timer in continuous mode.

Parameters

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

Function Class:

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

6.6.4.16 void gpt_lld_stop_timer (*GPTDriver* * *gptp*)

Stops the timer.

Parameters

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

Function Class:

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

6.6.4.17 void gpt_lld_polled_delay (*GPTDriver* * *gptp*, *gptcnt_t* *interval*)

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

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

Parameters

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

Function Class:

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

6.6.5 Variable Documentation

6.6.5.1 GPTDriver GPTD1

GPTD1 driver identifier.

Note

The driver GPTD1 allocates the complex timer TIM1 when enabled.

6.6.5.2 GPTDriver GPTD2

GPTD2 driver identifier.

Note

The driver GPTD2 allocates the timer TIM2 when enabled.

6.6.5.3 GPTDriver GPTD3

GPTD3 driver identifier.

Note

The driver GPTD3 allocates the timer TIM3 when enabled.

6.6.5.4 GPTDriver GPTD4

GPTD4 driver identifier.

Note

The driver GPTD4 allocates the timer TIM4 when enabled.

6.6.5.5 GPTDriver GPTD5

GPTD5 driver identifier.

Note

The driver GPTD5 allocates the timer TIM5 when enabled.

6.6.5.6 GPTDriver GPTD8

GPTD8 driver identifier.

Note

The driver GPTD8 allocates the timer TIM8 when enabled.

6.6.6 Define Documentation

6.6.6.1 #define STM32_GPT_USE_TIM1 TRUE

GPTD1 driver enable switch.

If set to TRUE the support for GPTD1 is included.

Note

The default is TRUE.

6.6.6.2 #define STM32_GPT_USE_TIM2 TRUE

GPTD2 driver enable switch.

If set to TRUE the support for GPTD2 is included.

Note

The default is TRUE.

6.6.6.3 #define STM32_GPT_USE_TIM3 TRUE

GPTD3 driver enable switch.

If set to TRUE the support for GPTD3 is included.

Note

The default is TRUE.

6.6.6.4 #define STM32_GPT_USE_TIM4 TRUE

GPTD4 driver enable switch.

If set to TRUE the support for GPTD4 is included.

Note

The default is TRUE.

6.6.6.5 #define STM32_GPT_USE_TIM5 TRUE

GPTD5 driver enable switch.

If set to TRUE the support for GPTD5 is included.

Note

The default is TRUE.

6.6.6.6 #define STM32_GPT_USE_TIM8 TRUE

GPTD8 driver enable switch.

If set to TRUE the support for GPTD8 is included.

Note

The default is TRUE.

6.6.6.7 #define STM32_GPT_TIM1_IRQ_PRIORITY 7

GPTD1 interrupt priority level setting.

6.6.6.8 #define STM32_GPT_TIM2_IRQ_PRIORITY 7

GPTD2 interrupt priority level setting.

6.6.6.9 #define STM32_GPT_TIM3_IRQ_PRIORITY 7

GPTD3 interrupt priority level setting.

6.6.6.10 #define STM32_GPT_TIM4_IRQ_PRIORITY 7

GPTD4 interrupt priority level setting.

6.6.6.11 `#define STM32_GPT_TIM5_IRQ_PRIORITY 7`

GPTD5 interrupt priority level setting.

6.6.6.12 `#define STM32_GPT_TIM8_IRQ_PRIORITY 7`

GPTD5 interrupt priority level setting.

6.6.7 Typedef Documentation

6.6.7.1 `typedef struct GPTDriver GPTDriver`

Type of a structure representing a GPT driver.

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

GPT notification callback type.

Parameters

in *gptp* pointer to a `GPTDriver` object

6.6.7.3 `typedef uint32_t gptfreq_t`

GPT frequency type.

6.6.7.4 `typedef uint16_t gptcnt_t`

GPT counter type.

6.6.8 Enumeration Type Documentation

6.6.8.1 `enum gptstate_t`

Driver state machine possible states.

Enumerator:

`GPT_UNINIT` Not initialized.

`GPT_STOP` Stopped.

`GPT_READY` Ready.

`GPT_CONTINUOUS` Active in continuous mode.

`GPT_ONESHOT` Active in one shot mode.

6.7 HAL Driver

6.7.1 Detailed Description

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

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

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

Data Structures

- struct `stm32_tim_t`
STM32 TIM registers block.

Functions

- void `halInit(void)`
HAL initialization.
- bool_t `halIsCounterWithin(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)`
STM32F2xx 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` "STM32F4 High Performance & DSP"

Absolute Maximum Ratings

- #define `STM32_HSECLK_MAX` 26000000
Maximum HSE clock frequency.
- #define `STM32_HSECLK_MIN` 1000000
Minimum HSE clock frequency.
- #define `STM32_LSECLK_MAX` 1000000

- #define STM32_LSECLK_MIN 32768
Minimum LSE clock frequency.
- #define STM32_PLLIN_MAX 2000000
Maximum PLLs input clock frequency.
- #define STM32_PLLIN_MIN 950000
Minimum PLLs input clock frequency.
- #define STM32_PLLVCO_MAX 432000000
Maximum PLLs VCO clock frequency.
- #define STM32_PLLVCO_MIN 192000000
Maximum PLLs VCO clock frequency.
- #define STM32_PLLOUT_MAX 168000000
Maximum PLL output clock frequency.
- #define STM32_PLLOUT_MIN 24000000
Minimum PLL output clock frequency.
- #define STM32_PCLK1_MAX 42000000
Maximum APB1 clock frequency.
- #define STM32_PCLK2_MAX 84000000
Maximum APB2 clock frequency.
- #define STM32_SPII2S_MAX 37500000
Maximum SPI/I2S clock frequency.

Internal clock sources

- #define STM32_HSICLK 16000000
- #define STM32_LSICLK 32000

PWR_CR register bits definitions

- #define STM32_VOS_MASK (1 << 14)
- #define STM32_VOS_LOW (0 << 14)
- #define STM32_VOS_HIGH (1 << 14)
- #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_PLLCFGR register bits definitions

- #define STM32_PLLP_MASK (3 << 16)
- #define STM32_PLLP_DIV2 (0 << 16)
- #define STM32_PLLP_DIV4 (1 << 16)
- #define STM32_PLLP_DIV6 (2 << 16)
- #define STM32_PLLP_DIV8 (3 << 16)
- #define STM32_PLLSRC_HSI (0 << 22)
- #define STM32_PLLSRC_HSE (1 << 22)

RCC_CFGR register bits definitions

- #define STM32_SW_MASK (3 << 0)
- #define STM32_SW_HSI (0 << 0)
- #define STM32_SW_HSE (1 << 0)
- #define STM32_SW_PLL (2 << 0)
- #define STM32_HPRE_MASK (15 << 4)
- #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_MASK (7 << 10)
- #define STM32_PPREG1_DIV1 (0 << 10)
- #define STM32_PPREG1_DIV2 (4 << 10)
- #define STM32_PPREG1_DIV4 (5 << 10)
- #define STM32_PPREG1_DIV8 (6 << 10)
- #define STM32_PPREG1_DIV16 (7 << 10)
- #define STM32_PPREG2_MASK (7 << 13)
- #define STM32_PPREG2_DIV1 (0 << 13)
- #define STM32_PPREG2_DIV2 (4 << 13)
- #define STM32_PPREG2_DIV4 (5 << 13)
- #define STM32_PPREG2_DIV8 (6 << 13)
- #define STM32_PPREG2_DIV16 (7 << 13)
- #define STM32_RTCPRE_MASK (31 << 16)
- #define STM32_MCO1SEL_MASK (3 << 21)
- #define STM32_MCO1SEL_HSI (0 << 21)
- #define STM32_MCO1SEL_LSE (1 << 21)
- #define STM32_MCO1SEL_HSE (2 << 21)
- #define STM32_MCO1SEL_PLL (3 << 21)
- #define STM32_I2SSRC_MASK (1 << 23)
- #define STM32_I2SSRC_PLLI2S (0 << 23)
- #define STM32_I2SSRC_CKIN (1 << 23)
- #define STM32_MCO1PRE_MASK (7 << 24)
- #define STM32_MCO1PRE_DIV1 (0 << 24)
- #define STM32_MCO1PRE_DIV2 (4 << 24)
- #define STM32_MCO1PRE_DIV3 (5 << 24)
- #define STM32_MCO1PRE_DIV4 (6 << 24)
- #define STM32_MCO1PRE_DIV5 (7 << 24)
- #define STM32_MCO2PRE_MASK (7 << 27)
- #define STM32_MCO2PRE_DIV1 (0 << 27)
- #define STM32_MCO2PRE_DIV2 (4 << 27)
- #define STM32_MCO2PRE_DIV3 (5 << 27)
- #define STM32_MCO2PRE_DIV4 (6 << 27)
- #define STM32_MCO2PRE_DIV5 (7 << 27)
- #define STM32_MCO2SEL_MASK (3U << 30)
- #define STM32_MCO2SEL_SYSCLK (0U << 30)
- #define STM32_MCO2SEL_PLLI2S (1U << 30)
- #define STM32_MCO2SEL_HSE (2U << 30)
- #define STM32_MCO2SEL_PLL (3U << 30)
- #define STM32_RTC_NOCLOCK (0 << 8)
- #define STM32_RTC_LSE (1 << 8)
- #define STM32_RTC_LSI (2 << 8)
- #define STM32_RTC_HSE (3 << 8)

RCC_PLLI2SCFGR register bits definitions

- #define STM32_PLLI2SN_MASK (511 << 6)
- #define STM32_PLLI2SR_MASK (7 << 28)

RCC_BDCR register bits definitions

- #define STM32_RTCSEL_MASK (3 << 8)
- #define STM32_RTCSEL_NOCLOCK (0 << 8)
- #define STM32_RTCSEL_LSE (1 << 8)
- #define STM32_RTCSEL_LSI (2 << 8)
- #define STM32_RTCSEL_HSEDIV (3 << 8)

STM32F4xx capabilities

- #define STM32_HAS_ADC1 TRUE
- #define STM32_ADC1_DMA_MSK
- #define STM32_ADC1_DMA_CHN 0x00000000
- #define STM32_HAS_ADC2 TRUE
- #define STM32_ADC2_DMA_MSK
- #define STM32_ADC2_DMA_CHN 0x00001100
- #define STM32_HAS_ADC3 TRUE
- #define STM32_ADC3_DMA_MSK
- #define STM32_ADC3_DMA_CHN 0x00000022
- #define STM32_HAS_CAN1 TRUE
- #define STM32_HAS_CAN2 TRUE
- #define STM32_CAN_MAX_FILTERS 28
- #define STM32_HAS_DAC TRUE
- #define STM32_ADVANCED_DMA TRUE
- #define STM32_HAS_DMA1 TRUE
- #define STM32_HAS_DMA2 TRUE
- #define STM32_HAS_ETH TRUE
- #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 TRUE
- #define STM32_HAS_GPIOG TRUE
- #define STM32_HAS_GPIOH TRUE
- #define STM32_HAS_GPIOI TRUE
- #define STM32_HAS_I2C1 TRUE
- #define STM32_I2C1_RX_DMA_MSK
- #define STM32_I2C1_RX_DMA_CHN 0x00100001
- #define STM32_I2C1_TX_DMA_MSK
- #define STM32_I2C1_TX_DMA_CHN 0x11000000
- #define STM32_HAS_I2C2 TRUE
- #define STM32_I2C2_RX_DMA_MSK
- #define STM32_I2C2_RX_DMA_CHN 0x00007700
- #define STM32_I2C2_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 7))
- #define STM32_I2C2_TX_DMA_CHN 0x70000000
- #define STM32_HAS_I2C3 TRUE
- #define STM32_I2C3_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 2))

- #define STM32_I2C3_RX_DMA_CHN 0x00000300
- #define STM32_I2C3_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 4))
- #define STM32_I2C3_TX_DMA_CHN 0x00030000
- #define STM32_HAS_RTC TRUE
- #define STM32_RTC_HAS_SUBSECONDS TRUE
- #define STM32_RTC_IS_CALENDAR TRUE
- #define STM32_HAS_SDIO TRUE
- #define STM32_SDC_SDIO_DMA_MSK
- #define STM32_SDC_SDIO_DMA_CHN 0x04004000
- #define STM32_HAS_SPI1 TRUE
- #define STM32_SPI1_RX_DMA_MSK
- #define STM32_SPI1_RX_DMA_CHN 0x00000303
- #define STM32_SPI1_TX_DMA_MSK
- #define STM32_SPI1_TX_DMA_CHN 0x00303000
- #define STM32_HAS_SPI2 TRUE
- #define STM32_SPI2_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 3))
- #define STM32_SPI2_RX_DMA_CHN 0x00000000
- #define STM32_SPI2_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 4))
- #define STM32_SPI2_TX_DMA_CHN 0x00000000
- #define STM32_HAS_SPI3 TRUE
- #define STM32_SPI3_RX_DMA_MSK
- #define STM32_SPI3_RX_DMA_CHN 0x00000000
- #define STM32_SPI3_TX_DMA_MSK
- #define STM32_SPI3_TX_DMA_CHN 0x00000000
- #define STM32_HAS_TIM1 TRUE
- #define STM32_HAS_TIM2 TRUE
- #define STM32_HAS_TIM3 TRUE
- #define STM32_HAS_TIM4 TRUE
- #define STM32_HAS_TIM5 TRUE
- #define STM32_HAS_TIM6 TRUE
- #define STM32_HAS_TIM7 TRUE
- #define STM32_HAS_TIM8 TRUE
- #define STM32_HAS_TIM9 TRUE
- #define STM32_HAS_TIM10 TRUE
- #define STM32_HAS_TIM11 TRUE
- #define STM32_HAS_TIM12 TRUE
- #define STM32_HAS_TIM13 TRUE
- #define STM32_HAS_TIM14 TRUE
- #define STM32_HAS_TIM15 FALSE
- #define STM32_HAS_TIM16 FALSE
- #define STM32_HAS_TIM17 FALSE
- #define STM32_HAS_USART1 TRUE
- #define STM32_USART1_RX_DMA_MSK
- #define STM32_USART1_RX_DMA_CHN 0x00400400
- #define STM32_USART1_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(2, 7))
- #define STM32_USART1_TX_DMA_CHN 0x40000000
- #define STM32_HAS_USART2 TRUE
- #define STM32_USART2_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 5))
- #define STM32_USART2_RX_DMA_CHN 0x00400000
- #define STM32_USART2_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 6))
- #define STM32_USART2_TX_DMA_CHN 0x04000000
- #define STM32_HAS_USART3 TRUE
- #define STM32_USART3_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 1))
- #define STM32_USART3_RX_DMA_CHN 0x00000040
- #define STM32_USART3_TX_DMA_MSK

- #define STM32_USART3_TX_DMA_CHN 0x00074000
- #define STM32_HAS_UART4 TRUE
- #define STM32_UART4_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 2))
- #define STM32_UART4_RX_DMA_CHN 0x00000400
- #define STM32_UART4_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 4))
- #define STM32_UART4_TX_DMA_CHN 0x00040000
- #define STM32_HAS_UART5 TRUE
- #define STM32_UART5_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 0))
- #define STM32_UART5_RX_DMA_CHN 0x00000004
- #define STM32_UART5_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 7))
- #define STM32_UART5_TX_DMA_CHN 0x40000000
- #define STM32_HAS_USART6 TRUE
- #define STM32_USART6_RX_DMA_MSK
- #define STM32_USART6_RX_DMA_CHN 0x00000550
- #define STM32_USART6_TX_DMA_MSK
- #define STM32_USART6_TX_DMA_CHN 0x55000000
- #define STM32_HAS_USB FALSE
- #define STM32_HAS_OTG1 TRUE
- #define STM32_HAS_OTG2 TRUE

IRQ VECTOR names

- #define WWDG_IRQHandler Vector40
- #define PVD_IRQHandler Vector44
- #define TAMP_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_Stream0_IRQHandler Vector6C
- #define DMA1_Stream1_IRQHandler Vector70
- #define DMA1_Stream2_IRQHandler Vector74
- #define DMA1_Stream3_IRQHandler Vector78
- #define DMA1_Stream4_IRQHandler Vector7C
- #define DMA1_Stream5_IRQHandler Vector80
- #define DMA1_Stream6_IRQHandler Vector84
- #define ADC1_2_3_IRQHandler Vector88
- #define CAN1_TX_IRQHandler Vector8C
- #define CAN1_RX0_IRQHandler Vector90
- #define CAN1_RX1_IRQHandler Vector94
- #define CAN1_SCE_IRQHandler Vector98
- #define EXTI9_5_IRQHandler Vector9C
- #define TIM1_BRK_IRQHandler VectorA0
- #define TIM1_UP_IRQHandler VectorA4
- #define TIM1_TRG_COM_IRQHandler VectorA8
- #define TIM1_CC_IRQHandler VectorAC
- #define TIM2_IRQHandler VectorB0
- #define TIM3_IRQHandler VectorB4
- #define TIM4_IRQHandler VectorB8
- #define I2C1_EV_IRQHandler VectorBC

- #define I2C1_ER_IRQHandler VectorC0
- #define I2C2_EV_IRQHandler VectorC4
- #define I2C2_ER_IRQHandler VectorC8
- #define SPI1_IRQHandler VectorCC
- #define SPI2_IRQHandler VectorD0
- #define USART1_IRQHandler VectorD4
- #define USART2_IRQHandler VectorD8
- #define USART3_IRQHandler VectorDC
- #define EXTI15_10_IRQHandler VectorE0
- #define RTC_Alarm_IRQHandler VectorE4
- #define OTG_FS_WKUP_IRQHandler VectorE8
- #define TIM8_BRK_IRQHandler VectorEC
- #define TIM8_UP_IRQHandler VectorF0
- #define TIM8_TRG_COM_IRQHandler VectorF4
- #define TIM8_CC_IRQHandler VectorF8
- #define DMA1_Stream7_IRQHandler VectorFC
- #define FSMC_IRQHandler Vector100
- #define SDIO_IRQHandler Vector104
- #define TIM5_IRQHandler Vector108
- #define SPI3_IRQHandler Vector10C
- #define UART4_IRQHandler Vector110
- #define UART5_IRQHandler Vector114
- #define TIM6_IRQHandler Vector118
- #define TIM7_IRQHandler Vector11C
- #define DMA2_Stream0_IRQHandler Vector120
- #define DMA2_Stream1_IRQHandler Vector124
- #define DMA2_Stream2_IRQHandler Vector128
- #define DMA2_Stream3_IRQHandler Vector12C
- #define DMA2_Stream4_IRQHandler Vector130
- #define ETH_IRQHandler Vector134
- #define ETH_WKUP_IRQHandler Vector138
- #define CAN2_TX_IRQHandler Vector13C
- #define CAN2_RX0_IRQHandler Vector140
- #define CAN2_RX1_IRQHandler Vector144
- #define CAN2_SCE_IRQHandler Vector148
- #define OTG_FS_IRQHandler Vector14C
- #define DMA2_Stream5_IRQHandler Vector150
- #define DMA2_Stream6_IRQHandler Vector154
- #define DMA2_Stream7_IRQHandler Vector158
- #define USART6_IRQHandler Vector15C
- #define I2C3_EV_IRQHandler Vector160
- #define I2C3_ER_IRQHandler Vector164
- #define OTG_HS_EP1_OUT_IRQHandler Vector168
- #define OTG_HS_EP1_IN_IRQHandler Vector16C
- #define OTG_HS_WKUP_IRQHandler Vector170
- #define OTG_HS_IRQHandler Vector174
- #define DCMI_IRQHandler Vector178
- #define CRYP_IRQHandler Vector17C
- #define HASH_RNG_IRQHandler Vector180
- #define FPU_IRQHandler Vector184

Configuration options

- `#define STM32_NO_INIT FALSE`
Disables the PWR/RCC initialization in the HAL.
- `#define STM32_VOS STM32_VOS_HIGH`
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 FALSE`
Enables or disables the LSI clock source.
- `#define STM32_HSE_ENABLED TRUE`
Enables or disables the HSE clock source.
- `#define STM32_LSE_ENABLED FALSE`
Enables or disables the LSE clock source.
- `#define STM32_CLOCK48_REQUIRED TRUE`
USB/SDIO clock setting.
- `#define STM32_SW STM32_SW_PLL`
Main clock source selection.
- `#define STM32_PLLSRC STM32_PLLSRC_HSE`
Clock source for the PLLs.
- `#define STM32_PLLM_VALUE 8`
PLL M divider value.
- `#define STM32_PLLN_VALUE 336`
PLL N multiplier value.
- `#define STM32_PLLP_VALUE 2`
PLL P divider value.
- `#define STM32_PLLQ_VALUE 7`
PLL Q multiplier value.
- `#define STM32_HPRE STM32_HPRE_DIV1`
AHB prescaler value.
- `#define STM32_PPREG1 STM32_PPREG1_DIV4`
APB1 prescaler value.
- `#define STM32_PPREG2 STM32_PPREG2_DIV2`
APB2 prescaler value.
- `#define STM32_RTCSEL STM32_RTCSEL_LSE`
RTC clock source.
- `#define STM32_RTCPRE_VALUE 8`
RTC HSE prescaler value.
- `#define STM32_MCO1SEL STM32_MCO1SEL_HSI`
MC01 clock source value.
- `#define STM32_MCO1PRE STM32_MCO1PRE_DIV1`
MC01 prescaler value.
- `#define STM32_MCO2SEL STM32_MCO2SEL_SYSCLK`
MC02 clock source value.
- `#define STM32_MCO2PRE STM32_MCO2PRE_DIV5`
MC02 prescaler value.
- `#define STM32_I2SSRC STM32_I2SSRC_CKIN`

- `#define STM32_PLLI2SN_VALUE 192`
PLL_{I2SN} multiplier value.
- `#define STM32_PLLI2SR_VALUE 5`
PLL_{I2SR} multiplier value.

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_IMPLEMENTS_COUNTERS TRUE`
Defines the support for realtime counters in the HAL.
- `#define STM32_SYSCLK_MAX 168000000`
Maximum SYSCLK.
- `#define STM32_0WS_THRESHOLD 30000000`
Maximum frequency thresholds and wait states for flash access.
- `#define STM32_PLLM (STM32_PLLM_VALUE << 0)`
STM32_PLLM field.
- `#define STM32_PLLCLKIN (STM32_HSECLK / STM32_PLLM_VALUE)`
PLLs input clock frequency.
- `#define STM32_ACTIVATE_PLL TRUE`
PLL activation flag.
- `#define STM32_PLLN (STM32_PLLN_VALUE << 6)`
STM32_PLLN field.
- `#define STM32_PLLP (0 << 16)`
STM32_PLLP field.
- `#define STM32_PLLQ (STM32_PLLQ_VALUE << 24)`
STM32_PLLQ field.
- `#define STM32_PLLVCO (STM32_PLLCLKIN * STM32_PLLN_VALUE)`
PLL VCO frequency.
- `#define STM32_PLLCLKOUT (STM32_PLLVCO / STM32_PLLP_VALUE)`
PLL output clock frequency.
- `#define STM32_SYSCLK STM32_HSICLK`
System clock source.
- `#define STM32_HCLK (STM32_SYSCLK / 1)`
AHB frequency.

- `#define STM32_PCLK1 (STM32_HCLK / 1)`
APB1 frequency.
- `#define STM32_PCLK2 (STM32_HCLK / 1)`
APB2 frequency.
- `#define STM32_ACTIVATE_PLLI2S TRUE`
PLL activation flag.
- `#define STM32_PLLI2SN (STM32_PLLI2SN_VALUE << 6)`
STM32_PLLI2SN field.
- `#define STM32_PLLI2SR (STM32_PLLI2SR_VALUE << 28)`
STM32_PLLI2SR field.
- `#define STM32_PLLI2SVCO (STM32_PLLCLKIN * STM32_PLLI2SN_VALUE)`
PLL VCO frequency.
- `#define STM32_PLLI2SCLKOUT (STM32_PLLI2SVCO / STM32_PLLI2SR)`
PLL I2S output clock frequency.
- `#define STM32_MCO1DIVCLK STM32_HSICLK`
MCO1 divider clock.
- `#define STM32_MCO1CLK STM32_MCO1DIVCLK`
MCO1 output pin clock.
- `#define STM32_MCO2DIVCLK STM32_HSECLK`
MCO2 divider clock.
- `#define STM32_MCO2CLK STM32_MCO2DIVCLK`
MCO2 output pin clock.
- `#define STM32_RTCPRE (STM32_RTCPRE_VALUE << 16)`
RTC HSE divider setting.
- `#define STM32_RTCPRE (STM32_RTCPRE_VALUE << 16)`
RTC HSE divider setting.
- `#define STM32_HSEDIVCLK (STM32_HSECLK / STM32_RTCPRE_VALUE)`
HSE divider toward RTC clock.
- `#define STM32_RTCCLK 0`
RTC clock.
- `#define STM32_PLL48CLK (STM32_PLLVCO / STM32_PLLQ_VALUE)`
48MHz frequency.
- `#define STM32_TIMCLK1 (STM32_PCLK1 * 1)`
Timers 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13, 14 clock.
- `#define STM32_TIMCLK2 (STM32_PCLK2 * 1)`
Timers 1, 8 clock.
- `#define STM32_FLASHBITS 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.

6.7.2 Function Documentation

6.7.2.1 void halInit(void)

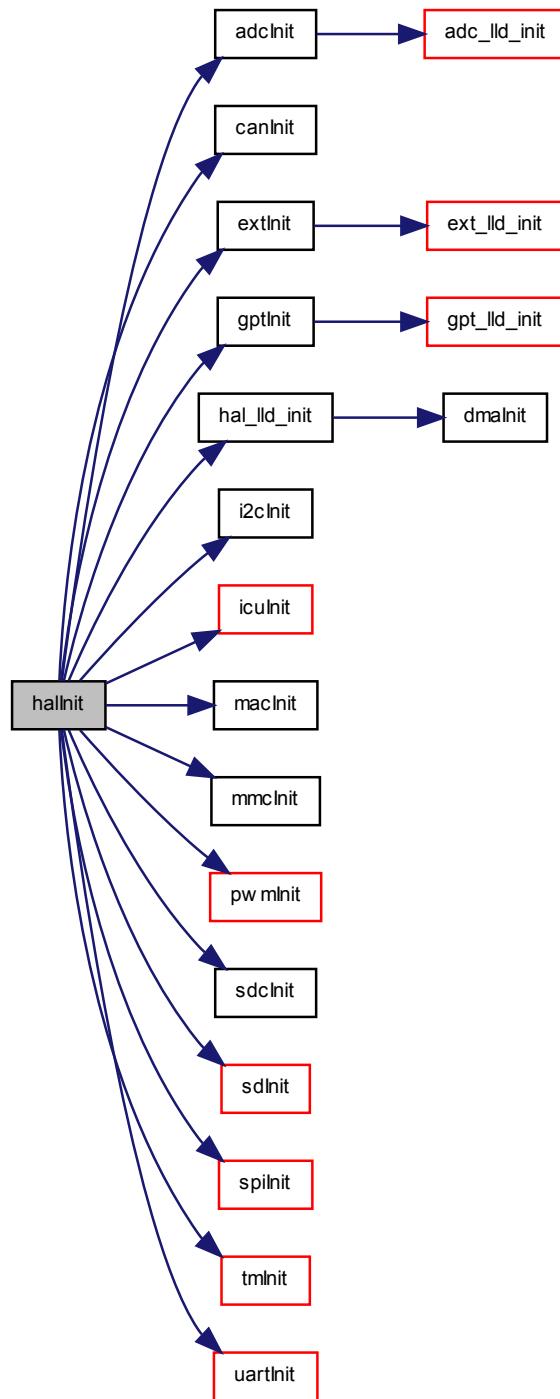
HAL initialization.

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

Function Class:

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

Here is the call graph for this function:



6.7.2.2 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.

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

Example 2

Example of a loop that lasts exactly 50 microseconds.

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

Parameters

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

Return values

<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.

6.7.2.3 void halPolledDelay (*halrtcnt_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:



6.7.2.4 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:



6.7.2.5 void stm32_clock_init(void)

STM32F2xx clocks and PLL initialization.

Note

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

Function Class:

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

6.7.3 Define Documentation

6.7.3.1 #define S2RTT(sec) (halGetCounterFrequency() * (sec))

Seconds to realtime ticks.

Converts from seconds to realtime ticks number.

Note

The result is rounded upward to the next tick boundary.

Parameters

in sec number of seconds

Returns

The number of ticks.

Function Class:

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

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

Milliseconds to realtime ticks.

Converts from milliseconds to realtime ticks number.

Note

The result is rounded upward to the next tick boundary.

Parameters

in msec number of milliseconds

Returns

The number of ticks.

Function Class:

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

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

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

Microseconds to realtime ticks.

Converts from microseconds to realtime ticks number.

Note

The result is rounded upward to the next tick boundary.

Parameters

in usec number of microseconds

Returns

The number of ticks.

Function Class:

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

6.7.3.4 #define halGetCounterValue() hal_lld_get_counter_value()

Returns the current value of the system free running counter.

Note

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

6.7.3.5 #define halGetCounterFrequency() hal_lld_get_counter_frequency()

Realtime counter frequency.

Note

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

6.7.3.6 #define HAL_IMPLEMENTERS_COUNTERS TRUE

Defines the support for realtime counters in the HAL.

6.7.3.7 #define STM32_HSECLK_MAX 26000000

Maximum HSE clock frequency.

6.7.3.8 #define STM32_HSECLK_MIN 1000000

Minimum HSE clock frequency.

6.7.3.9 #define STM32_LSECLK_MAX 1000000

Maximum LSE clock frequency.

6.7.3.10 #define STM32_LSECLK_MIN 32768

Minimum LSE clock frequency.

6.7.3.11 #define STM32_PLLIN_MAX 2000000

Maximum PLLs input clock frequency.

6.7.3.12 #define STM32_PLLIN_MIN 950000

Minimum PLLs input clock frequency.

6.7.3.13 #define STM32_PLLVCO_MAX 432000000

Maximum PLLs VCO clock frequency.

6.7.3.14 #define STM32_PLLVCO_MIN 192000000

Maximum PLLs VCO clock frequency.

6.7.3.15 #define STM32_PLLOUT_MAX 168000000

Maximum PLL output clock frequency.

6.7.3.16 #define STM32_PLLOUT_MIN 24000000

Minimum PLL output clock frequency.

6.7.3.17 #define STM32_PCLK1_MAX 42000000

Maximum APB1 clock frequency.

6.7.3.18 #define STM32_PCLK2_MAX 84000000

Maximum APB2 clock frequency.

6.7.3.19 #define STM32_SPII2S_MAX 37500000

Maximum SPI/I2S clock frequency.

6.7.3.20 #define STM32_HSICLK 16000000

High speed internal clock.

6.7.3.21 #define STM32_LSIGCLK 32000

Low speed internal clock.

6.7.3.22 #define STM32_VOS_MASK (1 << 14)

Core voltage mask.

6.7.3.23 #define STM32_VOS_LOW (0 << 14)

Core voltage set to low.

6.7.3.24 #define STM32_VOS_HIGH (1 << 14)

Core voltage set to high.

6.7.3.25 #define STM32_PLS_MASK (7 << 5)

PLS bits mask.

6.7.3.26 #define STM32_PLSLEV0 (0 << 5)

PVD level 0.

6.7.3.27 #define STM32_PLSLEV1 (1 << 5)

PVD level 1.

6.7.3.28 #define STM32_PLSLEV2 (2 << 5)

PVD level 2.

6.7.3.29 #define STM32_PLSLEV3 (3 << 5)

PVD level 3.

6.7.3.30 #define STM32_PLSLEV4 (4 << 5)

PVD level 4.

6.7.3.31 #define STM32_PLSLEV5 (5 << 5)

PVD level 5.

6.7.3.32 #define STM32_PLSLEV6 (6 << 5)

PVD level 6.

6.7.3.33 #define STM32_PLSLEV7 (7 << 5)

PVD level 7.

6.7.3.34 #define STM32_PLLP_MASK (3 << 16)

PLLp mask.

6.7.3.35 #define STM32_PLLP_DIV2 (0 << 16)

PLL clock divided by 2.

6.7.3.36 #define STM32_PLLP_DIV4 (1 << 16)

PLL clock divided by 4.

6.7.3.37 #define STM32_PLLP_DIV6 (2 << 16)

PLL clock divided by 6.

6.7.3.38 #define STM32_PLLP_DIV8 (3 << 16)

PLL clock divided by 8.

6.7.3.39 #define STM32_PLLSRC_HSI (0 << 22)

PLL clock source is HSI.

6.7.3.40 #define STM32_PLLSRC_HSE (1 << 22)

PLL clock source is HSE.

6.7.3.41 #define STM32_SW_MASK (3 << 0)

SW mask.

6.7.3.42 #define STM32_SW_HSI (0 << 0)

SYSCLK source is HSI.

6.7.3.43 #define STM32_SW_HSE (1 << 0)

SYSCLK source is HSE.

6.7.3.44 #define STM32_SW_PLL (2 << 0)

SYSCLK source is PLL.

6.7.3.45 #define STM32_HPRE_MASK (15 << 4)

HPRE mask.

6.7.3.46 #define STM32_HPRE_DIV1 (0 << 4)

SYSCLK divided by 1.

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

SYSCLK divided by 2.

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

SYSCLK divided by 4.

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

SYSCLK divided by 8.

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

SYSCLK divided by 16.

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

SYSCLK divided by 64.

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

SYSCLK divided by 128.

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

SYSCLK divided by 256.

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

SYSCLK divided by 512.

6.7.3.55 `#define STM32_PPREG1_MASK (7 << 10)`

PPREG1 mask.

6.7.3.56 `#define STM32_PPREG1_DIV1 (0 << 10)`

HCLK divided by 1.

6.7.3.57 `#define STM32_PPREG1_DIV2 (4 << 10)`

HCLK divided by 2.

6.7.3.58 `#define STM32_PPREG1_DIV4 (5 << 10)`

HCLK divided by 4.

6.7.3.59 #define STM32_PPREG1_DIV8 (6 << 10)

HCLK divided by 8.

6.7.3.60 #define STM32_PPREG1_DIV16 (7 << 10)

HCLK divided by 16.

6.7.3.61 #define STM32_PPREG2_MASK (7 << 13)

PPREG2 mask.

6.7.3.62 #define STM32_PPREG2_DIV1 (0 << 13)

HCLK divided by 1.

6.7.3.63 #define STM32_PPREG2_DIV2 (4 << 13)

HCLK divided by 2.

6.7.3.64 #define STM32_PPREG2_DIV4 (5 << 13)

HCLK divided by 4.

6.7.3.65 #define STM32_PPREG2_DIV8 (6 << 13)

HCLK divided by 8.

6.7.3.66 #define STM32_PPREG2_DIV16 (7 << 13)

HCLK divided by 16.

6.7.3.67 #define STM32_RTCPRE_MASK (31 << 16)

RTCPRE mask.

6.7.3.68 #define STM32_MCO1SEL_MASK (3 << 21)

MCO1 mask.

6.7.3.69 #define STM32_MCO1SEL_HSI (0 << 21)

HSI clock on MCO1 pin.

6.7.3.70 #define STM32_MCO1SEL_LSE (1 << 21)

LSE clock on MCO1 pin.

6.7.3.71 #define STM32_MCO1SEL_HSE (2 << 21)

HSE clock on MCO1 pin.

6.7.3.72 #define STM32_MCO1SEL_PLL (3 << 21)

PLL clock on MCO1 pin.

6.7.3.73 #define STM32_I2SSRC_MASK (1 << 23)

I2CSRC mask.

6.7.3.74 #define STM32_I2SSRC_PLLI2S (0 << 23)

I2SSRC is PLLI2S.

6.7.3.75 #define STM32_I2SSRC_CKIN (1 << 23)

I2S_CKIN is PLLI2S.

6.7.3.76 #define STM32_MCO1PRE_MASK (7 << 24)

MCO1PRE mask.

6.7.3.77 #define STM32_MCO1PRE_DIV1 (0 << 24)

MCO1 divided by 1.

6.7.3.78 #define STM32_MCO1PRE_DIV2 (4 << 24)

MCO1 divided by 2.

6.7.3.79 #define STM32_MCO1PRE_DIV3 (5 << 24)

MCO1 divided by 3.

6.7.3.80 #define STM32_MCO1PRE_DIV4 (6 << 24)

MCO1 divided by 4.

6.7.3.81 #define STM32_MCO1PRE_DIV5 (7 << 24)

MCO1 divided by 5.

6.7.3.82 #define STM32_MCO2PRE_MASK (7 << 27)

MCO2PRE mask.

6.7.3.83 #define STM32_MCO2PRE_DIV1 (0 << 27)

MCO2 divided by 1.

6.7.3.84 #define STM32_MCO2PRE_DIV2 (4 << 27)

MCO2 divided by 2.

6.7.3.85 #define STM32_MCO2PRE_DIV3 (5 << 27)

MCO2 divided by 3.

6.7.3.86 #define STM32_MCO2PRE_DIV4 (6 << 27)

MCO2 divided by 4.

6.7.3.87 #define STM32_MCO2PRE_DIV5 (7 << 27)

MCO2 divided by 5.

6.7.3.88 #define STM32_MCO2SEL_MASK (3U << 30)

MCO2 mask.

6.7.3.89 #define STM32_MCO2SEL_SYSCLK (0U << 30)

SYSCLK clock on MCO2 pin.

6.7.3.90 #define STM32_MCO2SEL_PLLI2S (1U << 30)

PLLI2S clock on MCO2 pin.

6.7.3.91 #define STM32_MCO2SEL_HSE (2U << 30)

HSE clock on MCO2 pin.

6.7.3.92 #define STM32_MCO2SEL_PLL (3U << 30)

PLL clock on MCO2 pin.

6.7.3.93 #define STM32_RTC_NOCLOCK (0 << 8)

No clock.

6.7.3.94 #define STM32_RTC_LSE (1 << 8)

LSE used as RTC clock.

6.7.3.95 #define STM32_RTC_LSI (2 << 8)

LSI used as RTC clock.

6.7.3.96 #define STM32_RTC_HSE (3 << 8)

HSE divided by programmable prescaler used as RTC clock

6.7.3.97 #define STM32_PLLI2SN_MASK (511 << 6)

PLLl2SN mask.

6.7.3.98 #define STM32_PLLI2SR_MASK (7 << 28)

PLLl2SR mask.

6.7.3.99 #define STM32_RTCSEL_MASK (3 << 8)

RTC source mask.

6.7.3.100 #define STM32_RTCSEL_NOCLOCK (0 << 8)

No RTC source.

6.7.3.101 #define STM32_RTCSEL_LSE (1 << 8)

RTC source is LSE.

6.7.3.102 #define STM32_RTCSEL_LSI (2 << 8)

RTC source is LSI.

6.7.3.103 #define STM32_RTCSEL_HSEDIV (3 << 8)

RTC source is HSE divided.

6.7.3.104 #define WWDG_IRQHandler Vector40

Window Watchdog.

6.7.3.105 #define PVD_IRQHandler Vector44

PVD through EXTI Line detect.

6.7.3.106 #define TAMP_STAMP_IRQHandler Vector48

Tamper and TimeStamp through EXTI Line.

6.7.3.107 #define RTC_WKUP_IRQHandler Vector4C

RTC wakeup EXTI Line.

6.7.3.108 #define FLASH_IRQHandler Vector50

Flash.

6.7.3.109 #define RCC_IRQHandler Vector54

RCC.

6.7.3.110 #define EXTI0_IRQHandler Vector58

EXTI Line 0.

6.7.3.111 #define EXTI1_IRQHandler Vector5C

EXTI Line 1.

6.7.3.112 #define EXTI2_IRQHandler Vector60

EXTI Line 2.

6.7.3.113 #define EXTI3_IRQHandler Vector64

EXTI Line 3.

6.7.3.114 #define EXTI4_IRQHandler Vector68

EXTI Line 4.

6.7.3.115 #define DMA1_Stream0_IRQHandler Vector6C

DMA1 Stream 0.

6.7.3.116 #define DMA1_Stream1_IRQHandler Vector70

DMA1 Stream 1.

6.7.3.117 #define DMA1_Stream2_IRQHandler Vector74

DMA1 Stream 2.

6.7.3.118 #define DMA1_Stream3_IRQHandler Vector78

DMA1 Stream 3.

6.7.3.119 #define DMA1_Stream4_IRQHandler Vector7C

DMA1 Stream 4.

6.7.3.120 #define DMA1_Stream5_IRQHandler Vector80

DMA1 Stream 5.

6.7.3.121 #define DMA1_Stream6_IRQHandler Vector84

DMA1 Stream 6.

6.7.3.122 #define ADC1_2_3_IRQHandler Vector88

ADC1, ADC2 and ADC3.

6.7.3.123 #define CAN1_TX_IRQHandler Vector8C

CAN1 TX.

6.7.3.124 #define CAN1_RX0_IRQHandler Vector90

CAN1 RX0.

6.7.3.125 #define CAN1_RX1_IRQHandler Vector94

CAN1 RX1.

6.7.3.126 #define CAN1_SCE_IRQHandler Vector98

CAN1 SCE.

6.7.3.127 #define EXTI9_5_IRQHandler Vector9C

EXTI Line 9..5.

6.7.3.128 #define TIM1_BRK_IRQHandler VectorA0

TIM1 Break.

6.7.3.129 #define TIM1_UP_IRQHandler VectorA4

TIM1 Update.

6.7.3.130 #define TIM1_TRG_COM_IRQHandler VectorA8

TIM1 Trigger and Commutation.

6.7.3.131 #define TIM1_CC_IRQHandler VectorAC

TIM1 Capture Compare.

6.7.3.132 #define TIM2_IRQHandler VectorB0

TIM2.

6.7.3.133 #define TIM3_IRQHandler VectorB4

TIM3.

6.7.3.134 #define TIM4_IRQHandler VectorB8

TIM4.

6.7.3.135 #define I2C1_EV_IRQHandler VectorBC

I2C1 Event.

6.7.3.136 #define I2C1_ER_IRQHandler VectorC0

I2C1 Error.

6.7.3.137 #define I2C2_EV_IRQHandler VectorC4

I2C2 Event.

6.7.3.138 #define I2C2_ER_IRQHandler VectorC8

I2C1 Error.

6.7.3.139 #define SPI1_IRQHandler VectorCC

SPI1.

6.7.3.140 #define SPI2_IRQHandler VectorD0

SPI2.

6.7.3.141 #define USART1_IRQHandler VectorD4

USART1.

6.7.3.142 #define USART2_IRQHandler VectorD8

USART2.

6.7.3.143 #define USART3_IRQHandler VectorDC

USART3.

6.7.3.144 #define EXTI15_10_IRQHandler VectorE0

EXTI Line 15..10.

6.7.3.145 #define RTC_Alarm_IRQHandler VectorE4

RTC alarms (A and B) through EXTI line.

6.7.3.146 #define OTG_FS_WKUP_IRQHandler VectorE8

USB OTG FS Wakeup through EXTI line.

6.7.3.147 #define TIM8_BRK_IRQHandler VectorEC

TIM8 Break.

6.7.3.148 #define TIM8_UP_IRQHandler VectorF0

TIM8 Update.

6.7.3.149 #define TIM8_TRG_COM_IRQHandler VectorF4

TIM8 Trigger and Commutation.

6.7.3.150 #define TIM8_CC_IRQHandler VectorF8

TIM8 Capture Compare.

6.7.3.151 #define DMA1_Stream7_IRQHandler VectorFC

DMA1 Stream 7.

6.7.3.152 #define FSMC_IRQHandler Vector100

FSMC.

6.7.3.153 #define SDIO_IRQHandler Vector104

SDIO.

6.7.3.154 #define TIM5_IRQHandler Vector108

TIM5.

6.7.3.155 #define SPI3_IRQHandler Vector10C

SPI3.

6.7.3.156 #define UART4_IRQHandler Vector110

UART4.

6.7.3.157 #define UART5_IRQHandler Vector114

UART5.

6.7.3.158 #define TIM6_IRQHandler Vector118

TIM6.

6.7.3.159 #define TIM7_IRQHandler Vector11C

TIM7.

6.7.3.160 #define DMA2_Stream0_IRQHandler Vector120

DMA2 Stream0.

6.7.3.161 #define DMA2_Stream1_IRQHandler Vector124

DMA2 Stream1.

6.7.3.162 #define DMA2_Stream2_IRQHandler Vector128

DMA2 Stream2.

6.7.3.163 #define DMA2_Stream3_IRQHandler Vector12C

DMA2 Stream3.

6.7.3.164 #define DMA2_Stream4_IRQHandler Vector130

DMA2 Stream4.

6.7.3.165 #define ETH_IRQHandler Vector134

Ethernet.

6.7.3.166 #define ETH_WKUP_IRQHandler Vector138

Ethernet Wakeup through EXTI line.

6.7.3.167 #define CAN2_TX_IRQHandler Vector13C

CAN2 TX.

6.7.3.168 #define CAN2_RX0_IRQHandler Vector140

CAN2 RX0.

6.7.3.169 #define CAN2_RX1_IRQHandler Vector144

CAN2 RX1.

6.7.3.170 #define CAN2_SCE_IRQHandler Vector148

CAN2 SCE.

6.7.3.171 #define OTG_FS_IRQHandler Vector14C

USB OTG FS.

6.7.3.172 #define DMA2_Stream5_IRQHandler Vector150

DMA2 Stream5.

6.7.3.173 #define DMA2_Stream6_IRQHandler Vector154

DMA2 Stream6.

6.7.3.174 #define DMA2_Stream7_IRQHandler Vector158

DMA2 Stream7.

6.7.3.175 #define USART6_IRQHandler Vector15C

USART6.

6.7.3.176 #define I2C3_EV_IRQHandler Vector160

I2C3 Event.

6.7.3.177 #define I2C3_ER_IRQHandler Vector164

I2C3 Error.

6.7.3.178 #define OTG_HS_EP1_OUT_IRQHandler Vector168

USB OTG HS End Point 1 Out.

6.7.3.179 #define OTG_HS_EP1_IN_IRQHandler Vector16C

USB OTG HS End Point 1 In.

6.7.3.180 #define OTG_HS_WKUP_IRQHandler Vector170

USB OTG HS Wakeup through EXTI line.

6.7.3.181 #define OTG_HS_IRQHandler Vector174

USB OTG HS.

6.7.3.182 #define DCMI_IRQHandler Vector178

DCMI.

6.7.3.183 #define CRYP_IRQHandler Vector17C

CRYP.

6.7.3.184 #define HASH_RNG_IRQHandler Vector180

Hash and Rng.

6.7.3.185 #define FPU_IRQHandler Vector184

Floating Point Unit.

6.7.3.186 #define STM32_NO_INIT FALSE

Disables the PWR/RCC initialization in the HAL.

6.7.3.187 #define STM32_VOS STM32_VOS_HIGH

Core voltage selection.

Note

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

6.7.3.188 #define STM32_PVD_ENABLE FALSE

Enables or disables the programmable voltage detector.

6.7.3.189 #define STM32_PLS STM32_PLS_LEV0

Sets voltage level for programmable voltage detector.

6.7.3.190 #define STM32_HSI_ENABLED TRUE

Enables or disables the HSI clock source.

6.7.3.191 #define STM32_LSI_ENABLED FALSE

Enables or disables the LSI clock source.

6.7.3.192 #define STM32_HSE_ENABLED TRUE

Enables or disables the HSE clock source.

6.7.3.193 #define STM32_LSE_ENABLED FALSE

Enables or disables the LSE clock source.

6.7.3.194 #define STM32_CLOCK48_REQUIRED TRUE

USB/SDIO clock setting.

6.7.3.195 #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 168MHz system clock from an external 8MHz HSE clock.

6.7.3.196 #define STM32_PLLSRC STM32_PLLSRC_HSE

Clock source for the PLLs.

Note

This setting has only effect if the PLL is selected as the system clock source.

The default value is calculated for a 168MHz system clock from an external 8MHz HSE clock.

6.7.3.197 #define STM32_PLLM_VALUE 8

PLLM divider value.

Note

The allowed values are 2..63.

The default value is calculated for a 168MHz system clock from an external 8MHz HSE clock.

6.7.3.198 #define STM32_PLLN_VALUE 336

PLLN multiplier value.

Note

The allowed values are 192..432.

The default value is calculated for a 168MHz system clock from an external 8MHz HSE clock.

6.7.3.199 #define STM32_PLLP_VALUE 2

PLLP divider value.

Note

The allowed values are 2, 4, 6, 8.

The default value is calculated for a 168MHz system clock from an external 8MHz HSE clock.

6.7.3.200 #define STM32_PLLQ_VALUE 7

PLLQ multiplier value.

Note

The allowed values are 2..15.

The default value is calculated for a 168MHz system clock from an external 8MHz HSE clock.

6.7.3.201 #define STM32_HPRE STM32_HPRE_DIV1

AHB prescaler value.

Note

The default value is calculated for a 168MHz system clock from an external 8MHz HSE clock.

6.7.3.202 #define STM32_PPRE1 STM32_PPRE1_DIV4

APB1 prescaler value.

6.7.3.203 #define STM32_PPRE2 STM32_PPRE2_DIV2

APB2 prescaler value.

6.7.3.204 #define STM32_RTCSEL STM32_RTCSEL_LSE

RTC clock source.

6.7.3.205 #define STM32_RTCPRE_VALUE 8

RTC HSE prescaler value.

6.7.3.206 #define STM32_MCO1SEL STM32_MCO1SEL_HSI

MC01 clock source value.

Note

The default value outputs HSI clock on MC01 pin.

```
6.7.3.207 #define STM32_MCO1PRE STM32_MCO1PRE_DIV1
```

MC01 prescaler value.

Note

The default value outputs HSI clock on MC01 pin.

```
6.7.3.208 #define STM32_MCO2SEL STM32_MCO2SEL_SYSCLK
```

MC02 clock source value.

Note

The default value outputs SYSCLK / 5 on MC02 pin.

```
6.7.3.209 #define STM32_MCO2PRE STM32_MCO2PRE_DIV5
```

MC02 prescaler value.

Note

The default value outputs SYSCLK / 5 on MC02 pin.

```
6.7.3.210 #define STM32_I2SSRC STM32_I2SSRC_CKIN
```

I2S clock source.

```
6.7.3.211 #define STM32_PLLI2SN_VALUE 192
```

PLL2SN multiplier value.

Note

The allowed values are 192..432.

```
6.7.3.212 #define STM32_PLLI2SR_VALUE 5
```

PLL2SR multiplier value.

Note

The allowed values are 2..7.

```
6.7.3.213 #define STM32_SYSCLK_MAX 168000000
```

Maximum SYSCLK.

Note

It is a function of the core voltage setting.

```
6.7.3.214 #define STM32_OWS_THRESHOLD 30000000
```

Maximum frequency thresholds and wait states for flash access.

Note

The values are valid for 2.7V to 3.6V supply range.

```
6.7.3.215 #define STM32_PLLM (STM32_PLLM_VALUE << 0)
```

STM32_PLLM field.

```
6.7.3.216 #define STM32_PLLCLKIN (STM32_HSECLK / STM32_PLLM_VALUE)
```

PLLs input clock frequency.

```
6.7.3.217 #define STM32_ACTIVATE_PLL TRUE
```

PLL activation flag.

```
6.7.3.218 #define STM32_PLLN (STM32_PLLN_VALUE << 6)
```

STM32_PLLN field.

```
6.7.3.219 #define STM32_PLLP (0 << 16)
```

STM32_PLLP field.

```
6.7.3.220 #define STM32_PLLQ (STM32_PLLQ_VALUE << 24)
```

STM32_PLLQ field.

```
6.7.3.221 #define STM32_PLLVCO (STM32_PLLCLKIN * STM32_PLLN_VALUE)
```

PLL VCO frequency.

```
6.7.3.222 #define STM32_PLLCLKOUT (STM32_PLLVCO / STM32_PLLP_VALUE)
```

PLL output clock frequency.

```
6.7.3.223 #define STM32_SYSCLK STM32_HSICLK
```

System clock source.

```
6.7.3.224 #define STM32_HCLK (STM32_SYSCLK / 1)
```

AHB frequency.

6.7.3.225 #define STM32_PCLK1 (STM32_HCLK / 1)

APB1 frequency.

6.7.3.226 #define STM32_PCLK2 (STM32_HCLK / 1)

APB2 frequency.

6.7.3.227 #define STM32_ACTIVATE_PLLI2S TRUE

PLL activation flag.

6.7.3.228 #define STM32_PLLI2SN (STM32_PLLI2SN_VALUE << 6)

STM32_PLLI2SN field.

6.7.3.229 #define STM32_PLLI2SR (STM32_PLLI2SR_VALUE << 28)

STM32_PLLI2SR field.

6.7.3.230 #define STM32_PLLI2SVCO (STM32_PLLCLKIN * STM32_PLLI2SN_VALUE)

PLL VCO frequency.

6.7.3.231 #define STM32_PLLI2SCLKOUT (STM32_PLLI2SVCO / STM32_PLLI2SR)

PLLI2S output clock frequency.

6.7.3.232 #define STM32_MCO1DIVCLK STM32_HSICLK

MCO1 divider clock.

6.7.3.233 #define STM32_MCO1CLK STM32_MCO1DIVCLK

MCO1 output pin clock.

6.7.3.234 #define STM32_MCO2DIVCLK STM32_HSECLK

MCO2 divider clock.

6.7.3.235 #define STM32_MCO2CLK STM32_MCO2DIVCLK

MCO2 output pin clock.

6.7.3.236 #define STM32_RTCPRE (STM32_RTCPRE_VALUE << 16)

RTC HSE divider setting.

6.7.3.237 #define STM32_RTCPRE (STM32_RTCPRE_VALUE << 16)

RTC HSE divider setting.

6.7.3.238 #define STM32_HSEDIVCLK (STM32_HSECLK / STM32_RTCPRE_VALUE)

HSE divider toward RTC clock.

6.7.3.239 #define STM32_RTCCLK 0

RTC clock.

6.7.3.240 #define STM32_PLL48CLK (STM32_PLLVCO / STM32_PLLQ_VALUE)

48MHz frequency.

6.7.3.241 #define STM32_TIMCLK1 (STM32_PCLK1 * 1)

Timers 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13, 14 clock.

6.7.3.242 #define STM32_TIMCLK2 (STM32_PCLK2 * 1)

Timers 1, 8 clock.

6.7.3.243 #define STM32_FLASHBITS 0x00000000

Flash settings.

6.7.3.244 #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.

6.7.3.245 #define hal_lld_get_counter_frequency() STM32_HCLK

Realtime counter frequency.

Note

The DWT_CYCCNT register is incremented directly by the system clock so this function returns STM32_HCLK.

Returns

The realtime counter frequency of type `halclock_t`.

Function Class:

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

6.7.4 Typedef Documentation

6.7.4.1 `typedef uint32_t halclock_t`

Type representing a system clock frequency.

6.7.4.2 `typedef uint32_t halrcnt_t`

Type of the realtime free counter value.

6.8 I2C Driver

6.8.1 Detailed Description

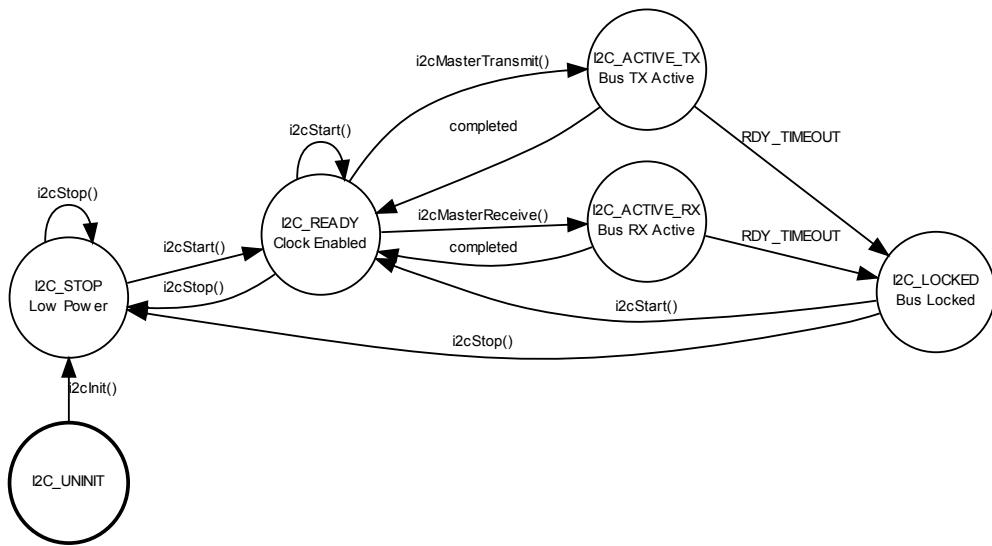
Generic I2C Driver. This module implements a generic I2C (Inter-Integrated Circuit) driver.

Precondition

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

6.8.2 Driver State Machine

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



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

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.

I2C bus error conditions

- #define [I2CD_NO_ERROR](#) 0x00

- `#define I2CD_BUS_ERROR 0x01`
No error.
- `#define I2CD_ARBITRATION_LOST 0x02`
Bus Error.
- `#define I2CD_ACK_FAILURE 0x04`
Arbitration Lost (master mode).
- `#define I2CD_OVERRUN 0x08`
Acknowledge Failure.
- `#define I2CD_PEC_ERROR 0x10`
Overrun/Underrun.
- `#define I2CD_TIMEOUT 0x20`
PEC Error in reception.
- `#define I2CD_SMB_ALERT 0x40`
Hardware timeout.
- `#define I2CD_SMB_ALERT 0x40`
SMBus Alert.

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.

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.*

6.8.3 Function Documentation

6.8.3.1 void i2cInit (void)

I2C Driver initialization.

Note

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

Function Class:

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

6.8.3.2 void i2cObjectInit (I2CDriver * i2cp)

Initializes the standard part of a `I2CDriver` structure.

Parameters

out *i2cp* pointer to the I2CDriver object

Function Class:

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

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

Configures and activates the I2C peripheral.

Parameters

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

Function Class:

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

6.8.3.4 void i2cStop (I2CDriver * *i2cp*)

Deactivates the I2C peripheral.

Parameters

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

Function Class:

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

6.8.3.5 i2cflags_t i2cGetErrors (I2CDriver * *i2cp*)

Returns the errors mask associated to the previous operation.

Parameters

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

6.8.3.6 msg_t i2cMasterTransmitTimeout (I2CDriver * *i2cp*, i2caddr_t *addr*, const uint8_t * *txbuf*, size_t *txbytes*, uint8_t * *rxbuf*, size_t *rxbytes*, systime_t *timeout*)

Sends data via the I2C bus.

Function designed to realize "read-through-write" transfer paradigm. If you want transmit data without any further read, than set **rxbytes** field to 0.

Parameters

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

Returns

The operation status.

Return values

<i>RDY_OK</i>	if the function succeeded.
<i>RDY_RESET</i>	if one or more I2C errors occurred, the errors can be retrieved using i2cGetErrors() .
<i>RDY_TIMEOUT</i>	if a timeout occurred before operation end.

Function Class:

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

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

Receives data from the I2C bus.

Parameters

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

Returns

The operation status.

Return values

<i>RDY_OK</i>	if the function succeeded.
<i>RDY_RESET</i>	if one or more I2C errors occurred, the errors can be retrieved using i2cGetErrors() .
<i>RDY_TIMEOUT</i>	if a timeout occurred before operation end.

Function Class:

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

6.8.3.8 void i2cAcquireBus (I2CDriver * *i2cp*)

Gains exclusive access to the I2C bus.

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

Precondition

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

Parameters

in *i2cp* pointer to the `I2CDriver` object

Function Class:

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

6.8.3.9 void i2cReleaseBus (`I2CDriver` * *i2cp*)

Releases exclusive access to the I2C bus.

Precondition

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

Parameters

in *i2cp* pointer to the `I2CDriver` object

Function Class:

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

6.8.4 Define Documentation

6.8.4.1 #define I2CD_NO_ERROR 0x00

No error.

6.8.4.2 #define I2CD_BUS_ERROR 0x01

Bus Error.

6.8.4.3 #define I2CD_ARBITRATION_LOST 0x02

Arbitration Lost (master mode).

6.8.4.4 #define I2CD_ACK_FAILURE 0x04

Acknowledge Failure.

6.8.4.5 #define I2CD_OVERRUN 0x08

Overrun/Underrun.

6.8.4.6 #define I2CD_PEC_ERROR 0x10

PEC Error in reception.

6.8.4.7 #define I2CD_TIMEOUT 0x20

Hardware timeout.

6.8.4.8 #define I2CD_SMB_ALERT 0x40

SMBus Alert.

6.8.4.9 #define I2C_USE_MUTUAL_EXCLUSION TRUE

Enables the mutual exclusion APIs on the I2C bus.

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

Value:

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

Wrap i2cMasterTransmitTimeout function with TIME_INFINITE timeout.

Function Class:

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

6.8.4.11 #define i2cMasterReceive(*i2cp*, *addr*, *rxbuf*, *rxbytes*) (i2cMasterReceiveTimeout(*i2cp*, *addr*, *rxbuf*, *rxbytes*,
TIME_INFINITE))

Wrap i2cMasterReceiveTimeout function with TIME_INFINITE timeout.

Function Class:

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

6.8.5 Enumeration Type Documentation

6.8.5.1 enum i2cstate_t

Driver state machine possible states.

Enumerator:

I2C_UNINIT Not initialized.

I2C_STOP Stopped.

I2C_READY Ready.

I2C_ACTIVE_TX Transmitting.

I2C_ACTIVE_RX Receiving.

6.9 ICU Driver

6.9.1 Detailed Description

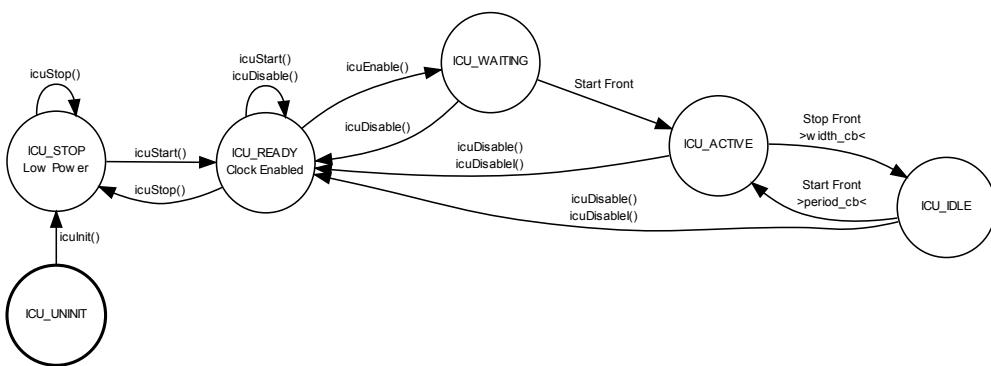
Generic ICU Driver. This module implements a generic ICU (Input Capture Unit) driver.

Precondition

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

6.9.2 Driver State Machine

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



6.9.3 ICU Operations.

This driver abstracts a generic Input Capture Unit composed of:

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

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

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

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

- **ICU_WAITING**, waiting the first start signal.
- **ICU_ACTIVE**, after a start signal.
- **ICU_IDLE**, after a stop signal.

Callbacks are invoked when start or stop signals occur.

Data Structures

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

Functions

- void **icuInit** (void)
ICU Driver initialization.
- void **icuObjectInit** (ICUDriver *icup)
*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.
- 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 icucnt_t**
ICU counter type.

Enumerations

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

6.9.4 Function Documentation

6.9.4.1 void icuInit(void)

ICU Driver initialization.

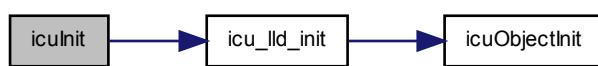
Note

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

Function Class:

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

Here is the call graph for this function:



6.9.4.2 void icuObjectInit(ICUDriver * icup)

Initializes the standard part of a **ICUDriver** structure.

Parameters

out **icup** pointer to the **ICUDriver** object

Function Class:

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

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

Configures and activates the ICU peripheral.

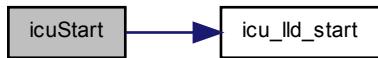
Parameters

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

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

Deactivates the ICU peripheral.

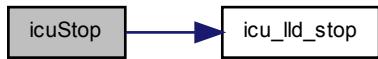
Parameters

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

Function Class:

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

Here is the call graph for this function:

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

Enables the input capture.

Parameters

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

Function Class:

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

Here is the call graph for this function:

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

Disables the input capture.

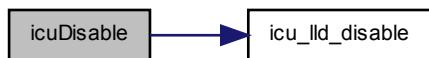
Parameters

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

Function Class:

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

Here is the call graph for this function:

**6.9.4.7 void icu_lld_init (void)**

Low level ICU driver initialization.

Function Class:

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

Here is the call graph for this function:



6.9.4.8 void icu_lld_start (ICUDriver * *icup*)

Configures and activates the ICU peripheral.

Parameters

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

Function Class:

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

6.9.4.9 void icu_lld_stop (ICUDriver * *icup*)

Deactivates the ICU peripheral.

Parameters

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

Function Class:

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

6.9.4.10 void icu_lld_enable (ICUDriver * *icup*)

Enables the input capture.

Parameters

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

Function Class:

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

6.9.4.11 void icu_lld_disable (ICUDriver * *icup*)

Disables the input capture.

Parameters

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

Function Class:

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

6.9.5 Variable Documentation

6.9.5.1 ICUDriver ICUD1

ICUD1 driver identifier.

Note

The driver ICUD1 allocates the complex timer TIM1 when enabled.

6.9.5.2 ICUDriver ICUD2

ICUD2 driver identifier.

Note

The driver ICUD1 allocates the timer TIM2 when enabled.

6.9.5.3 ICUDriver ICUD3

ICUD3 driver identifier.

Note

The driver ICUD1 allocates the timer TIM3 when enabled.

6.9.5.4 ICUDriver ICUD4

ICUD4 driver identifier.

Note

The driver ICUD4 allocates the timer TIM4 when enabled.

6.9.5.5 ICUDriver ICUD5

ICUD5 driver identifier.

Note

The driver ICUD5 allocates the timer TIM5 when enabled.

6.9.5.6 ICUDriver ICUD8

ICUD8 driver identifier.

Note

The driver ICUD8 allocates the timer TIM8 when enabled.

6.9.6 Define Documentation

6.9.6.1 #define icuEnable(*icup*) icu_lld_enable(*icup*)

Enables the input capture.

Parameters

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

Function Class:

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

6.9.6.2 #define icuDisable(*icup*) icu_lld_disable(*icup*)

Disables the input capture.

Parameters

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

Function Class:

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

6.9.6.3 #define icuGetWidth(*icup*) icu_lld_get_width(*icup*)

Returns the width of the latest pulse.

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

Parameters

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

Returns

The number of ticks.

Function Class:

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

6.9.6.4 #define icuGetPeriod(*icup*) icu_lld_get_period(*icup*)

Returns the width of the latest cycle.

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

Parameters

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

Returns

The number of ticks.

Function Class:

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

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

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

Common ISR code, ICU width event.

Parameters

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

Function Class:

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

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

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

Common ISR code, ICU period event.

Parameters

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

Function Class:

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

6.9.6.7 #define STM32_ICU_USE_TIM1 TRUE

ICUD1 driver enable switch.

If set to TRUE the support for ICUD1 is included.

Note

The default is TRUE.

6.9.6.8 #define STM32_ICU_USE_TIM2 TRUE

ICUD2 driver enable switch.

If set to TRUE the support for ICUD2 is included.

Note

The default is TRUE.

6.9.6.9 #define STM32_ICU_USE_TIM3 TRUE

ICUD3 driver enable switch.

If set to TRUE the support for ICUD3 is included.

Note

The default is TRUE.

6.9.6.10 #define STM32_ICU_USE_TIM4 TRUE

ICUD4 driver enable switch.

If set to TRUE the support for ICUD4 is included.

Note

The default is TRUE.

6.9.6.11 #define STM32_ICU_USE_TIM5 TRUE

ICUD5 driver enable switch.

If set to TRUE the support for ICUD5 is included.

Note

The default is TRUE.

6.9.6.12 #define STM32_ICU_USE_TIM8 TRUE

ICUD8 driver enable switch.

If set to TRUE the support for ICUD8 is included.

Note

The default is TRUE.

6.9.6.13 #define STM32_ICU_TIM1_IRQ_PRIORITY 7

ICUD1 interrupt priority level setting.

6.9.6.14 #define STM32_ICU_TIM2_IRQ_PRIORITY 7

ICUD2 interrupt priority level setting.

6.9.6.15 #define STM32_ICU_TIM3_IRQ_PRIORITY 7

ICUD3 interrupt priority level setting.

6.9.6.16 #define STM32_ICU_TIM4_IRQ_PRIORITY 7

ICUD4 interrupt priority level setting.

6.9.6.17 #define STM32_ICU_TIM5_IRQ_PRIORITY 7

ICUD5 interrupt priority level setting.

6.9.6.18 #define STM32_ICU_TIM8_IRQ_PRIORITY 7

ICUD8 interrupt priority level setting.

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

Returns the width of the latest pulse.

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

Parameters

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

Returns

The number of ticks.

Function Class:

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

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

Returns the width of the latest cycle.

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

Parameters

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

Returns

The number of ticks.

Function Class:

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

6.9.7 Typedef Documentation

6.9.7.1 **typedef struct ICUDriver ICUDriver**

Type of a structure representing an ICU driver.

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

ICU notification callback type.

Parameters

in *icup* pointer to a `ICUDriver` object

6.9.7.3 `typedef uint32_t icufreq_t`

ICU frequency type.

6.9.7.4 `typedef uint16_t icucnt_t`

ICU counter type.

6.9.8 Enumeration Type Documentation**6.9.8.1 `enum icustate_t`**

Driver state machine possible states.

Enumerator:

ICU_UNINIT Not initialized.

ICU_STOP Stopped.

ICU_READY Ready.

ICU_WAITING Waiting first edge.

ICU_ACTIVE Active cycle phase.

ICU_IDLE Idle cycle phase.

6.9.8.2 `enum icumode_t`

ICU driver mode.

Enumerator:

ICU_INPUT_ACTIVE_HIGH Trigger on rising edge.

ICU_INPUT_ACTIVE_LOW Trigger on falling edge.

6.10 MAC Driver**6.10.1 Detailed Description**

Generic MAC driver. This module implements a generic MAC (Media Access Control) driver for Ethernet controllers.

Precondition

In order to use the MAC driver the `HAL_USE_MAC` option must be enabled in `halconf.h`.

Functions

- void `macInit` (void)
MAC Driver initialization.
- void `macObjectInit` (`MACDriver` **macp*)

- `void macStart (MACDriver *macp, const MACConfig *config)`
Configures and activates the MAC peripheral.
- `void macStop (MACDriver *macp)`
Deactivates the MAC peripheral.
- `msg_t macWaitTransmitDescriptor (MACDriver *macp, MACTransmitDescriptor *tdp, systime_t time)`
Allocates a transmission descriptor.
- `void macReleaseTransmitDescriptor (MACTransmitDescriptor *tdp)`
Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.
- `msg_t macWaitReceiveDescriptor (MACDriver *macp, MACReceiveDescriptor *rdp, systime_t time)`
Waits for a received frame.
- `void macReleaseReceiveDescriptor (MACReceiveDescriptor *rdp)`
Releases a receive descriptor.
- `bool_t macPollLinkStatus (MACDriver *macp)`
Updates and returns the link status.

MAC configuration options

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

Macro Functions

- `#define macGetReceiveEventSource(macp) (&(macp)->rdevent)`
Returns the received frames event source.
- `#define macWriteTransmitDescriptor(tdp, buf, size) mac_lld_write_transmit_descriptor(tdp, buf, size)`
Writes to a transmit descriptor's stream.
- `#define macReadReceiveDescriptor(rdp, buf, size) mac_lld_read_receive_descriptor(rdp, buf, size)`
Reads from a receive descriptor's stream.

Typedefs

- `typedef struct MACDriver MACDriver`
Type of a structure representing a MAC driver.

Enumerations

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

6.10.2 Function Documentation

6.10.2.1 void macInit (void)

MAC Driver initialization.

Note

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

Function Class:

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

6.10.2.2 void macObjectInit (MACDriver * *macp*)

Initialize the standard part of a MACDriver structure.

Parameters

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

Function Class:

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

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

Configures and activates the MAC peripheral.

Parameters

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

Function Class:

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

6.10.2.4 void macStop (MACDriver * *macp*)

Deactivates the MAC peripheral.

Parameters

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

Function Class:

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

6.10.2.5 msg_t macWaitTransmitDescriptor (MACDriver * *macp*, MACTransmitDescriptor * *tdp*, systime_t *time*)

Allocates a transmission descriptor.

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

Parameters

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

Returns

The operation status.

Return values

RDY_OK the descriptor was obtained.
RDY_TIMEOUT the operation timed out, descriptor not initialized.

Function Class:

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

6.10.2.6 void macReleaseTransmitDescriptor (MACTransmitDescriptor * *tdp*)

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

Parameters

in *tdp* the pointer to the `MACTransmitDescriptor` structure

Function Class:

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

6.10.2.7 msg_t macWaitReceiveDescriptor (MACDriver * *macp*, MACReceiveDescriptor * *rdp*, systime_t *time*)

Waits for a received frame.

Stops until a frame is received and buffered. If a frame is not immediately available then the invoking thread is queued until one is received.

Parameters

in *macp* pointer to the `MACDriver` object

out *rdp* pointer to a `MACReceiveDescriptor` structure

in *time* the number of ticks before the operation timeouts, the following special values are allowed:

- *TIME_IMMEDIATE* immediate timeout.
- *TIME_INFINITE* no timeout.

Returns

The operation status.

Return values

RDY_OK the descriptor was obtained.
RDY_TIMEOUT the operation timed out, descriptor not initialized.

Function Class:

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

6.10.2.8 void macReleaseReceiveDescriptor (MACReceiveDescriptor * *rdp*)

Releases a receive descriptor.

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

Parameters

in *rdp* the pointer to the `MACReceiveDescriptor` structure

Function Class:

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

6.10.2.9 bool_t macPollLinkStatus (MACDriver * *macp*)

Updates and returns the link status.

Parameters

in *macp* pointer to the MACDriver object

Returns

The link status.

Return values

TRUE if the link is active.

FALSE if the link is down.

Function Class:

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

6.10.3 Define Documentation**6.10.3.1 #define MAC_USE_EVENTS TRUE**

Enables an event sources for incoming packets.

6.10.3.2 #define macGetReceiveEventSource(*macp*) (&(*macp*)>rdevent)

Returns the received frames event source.

Parameters

in *macp* pointer to the MACDriver object

Returns

The pointer to the EventSource structure.

Function Class:

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

6.10.3.3 #define macWriteTransmitDescriptor(*tdp*, *buf*, *size*) mac_lld_write_transmit_descriptor(*tdp*, *buf*, *size*)

Writes to a transmit descriptor's stream.

Parameters

in *tdp* pointer to a MACTransmitDescriptor structure
in *buf* pointer to the buffer containing the data to be written
in *size* number of bytes to be written

Returns

The number of bytes written into the descriptor's stream, this value can be less than the amount specified in

the parameter `size` if the maximum frame size is reached.

Function Class:

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

6.10.3.4 #define macReadReceiveDescriptor(*rdp*, *buf*, *size*) mac_lld_read_descriptor(*rdp*, *buf*, *size*)

Reads from a receive descriptor's stream.

Parameters

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

Returns

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

Function Class:

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

6.10.4 Typedef Documentation

6.10.4.1 typedef struct MACDriver MACDriver

Type of a structure representing a MAC driver.

6.10.5 Enumeration Type Documentation

6.10.5.1 enum macstate_t

Driver state machine possible states.

Enumerator:

MAC_UNINIT Not initialized.

MAC_STOP Stopped.

MAC_ACTIVE Active.

6.11 MMC over SPI Driver

6.11.1 Detailed Description

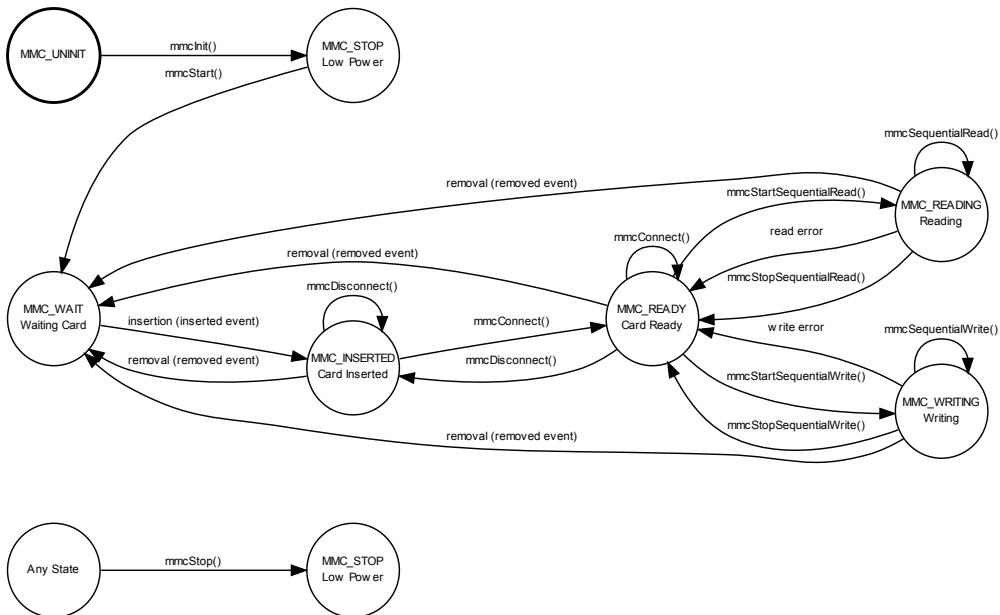
Generic MMC driver. This module implements a portable MMC/SD driver that uses a SPI driver as physical layer. Hot plugging and removal are supported through kernel events.

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`.

6.11.2 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).



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)`
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.

MMC_SPI configuration options

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

Macro Functions

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

Typedefs

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

Enumerations

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

6.11.3 Function Documentation

6.11.3.1 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.

6.11.3.2 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 <code>MMCDriver</code> 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.

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

Configures and activates the MMC peripheral.

Parameters

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

Function Class:

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

6.11.3.4 void mmcStop (MMCDriver * *mmcp*)

Disables the MMC peripheral.

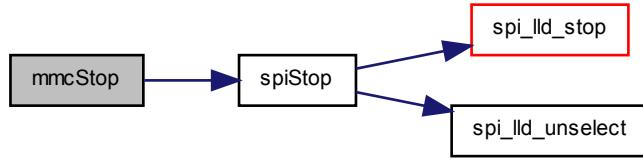
Parameters

in	<i>mmcp</i>	pointer to the <code>MMCDriver</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:



6.11.3.5 `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.

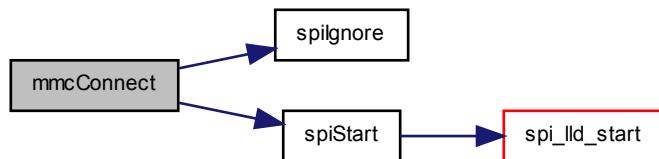
Return values

<code>FALSE</code>	the operation succeeded and the driver is now in the <code>MMC_READY</code> state.
<code>TRUE</code>	the operation failed.

Function Class:

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

Here is the call graph for this function:



6.11.3.6 `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.

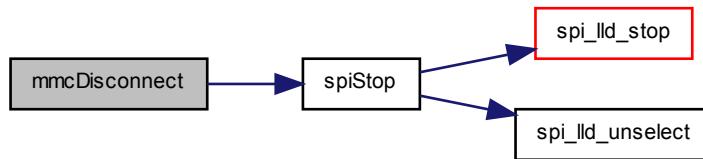
Return values

<i>FALSE</i>	the operation succeeded and the driver is now in the <code>MMC_INSERTED</code> state.
<i>TRUE</i>	the operation failed.

Function Class:

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

Here is the call graph for this function:



6.11.3.7 `bool_t mmcStartSequentialRead (MMCDriver * mmcp, uint32_t startblk)`

Starts a sequential read.

Parameters

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

Returns

The operation status.

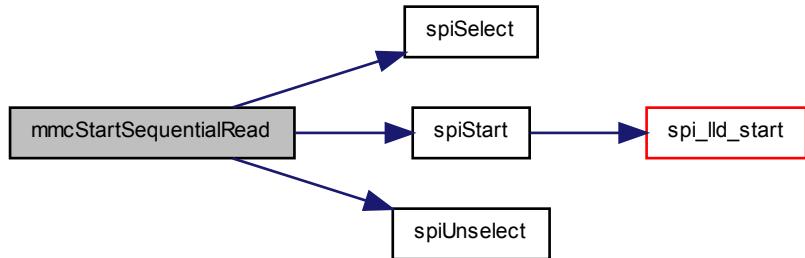
Return values

<i>FALSE</i>	the operation succeeded.
<i>TRUE</i>	the operation failed.

Function Class:

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

Here is the call graph for this function:



6.11.3.8 `bool_t mmcSequentialRead (MMCDriver * mmcp, uint8_t * buffer)`

Reads a block within a sequential read operation.

Parameters

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

Returns

The operation status.

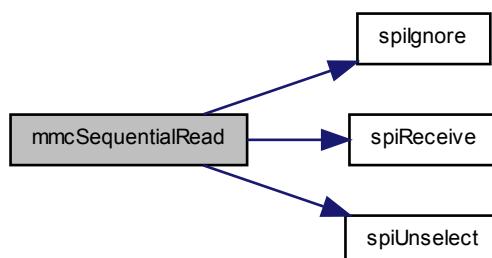
Return values

<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:



6.11.3.9 `bool_t mmcStopSequentialRead(MMCDriver * mmcp)`

Stops a sequential read gracefully.

Parameters

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

Returns

The operation status.

Return values

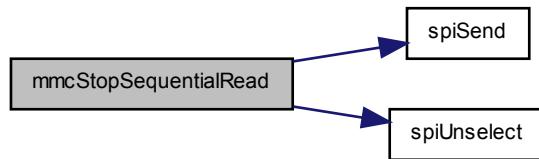
FALSE the operation succeeded.

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:



6.11.3.10 `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

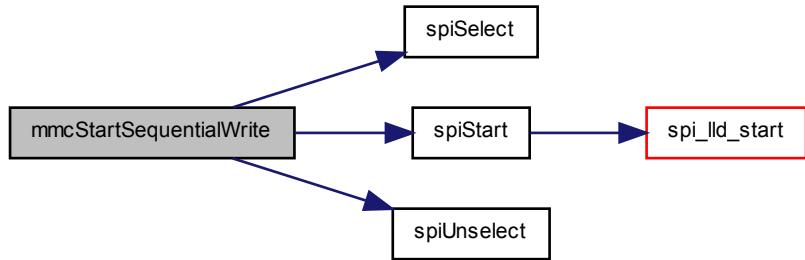
FALSE the operation succeeded.

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:



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

Writes a block within a sequential write operation.

Parameters

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

Returns

The operation status.

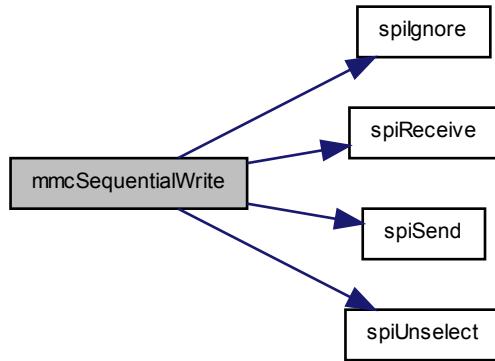
Return values

<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:



6.11.3.12 `bool_t mmcStopSequentialWrite (MMCDriver * mmcp)`

Stops a sequential write gracefully.

Parameters

`in mmcp` pointer to the `MMCDriver` object

Returns

The operation status.

Return values

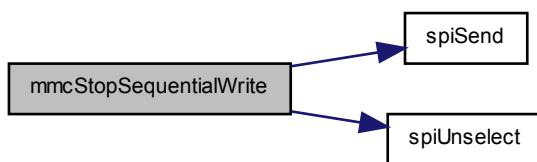
`FALSE` the operation succeeded.

`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:



6.11.4 Define Documentation

6.11.4.1 `#define MMC_SECTOR_SIZE 512`

Block size for MMC transfers.

6.11.4.2 `#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.

6.11.4.3 `#define MMC_POLLING_INTERVAL 10`

Number of positive insertion queries before generating the insertion event.

6.11.4.4 `#define MMC_POLLING_DELAY 10`

Interval, in milliseconds, between insertion queries.

6.11.4.5 `#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.

6.11.4.6 `#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.

6.11.5 Typedef Documentation

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

Function used to query some hardware status bits.

Returns

The status.

6.11.6 Enumeration Type Documentation

6.11.6.1 `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.

6.12 PAL Driver

6.12.1 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`.

6.12.2 Implementation Rules

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

6.12.2.1 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.

6.12.2.2 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.

6.12.2.3 Writing unused or unimplemented port bits

The behavior is not specified.

6.12.2.4 Reading from unused or unimplemented port bits

The behavior is not specified.

6.12.2.5 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_PUSHPULL 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)`

- `#define palSetPad(port, pad) palSetPort(port, PAL_PORT_BIT(pad))`
Writes a logical state on an output pad.
- `#define palClearPad(port, pad) palClearPort(port, PAL_PORT_BIT(pad))`
Sets a pad logical state to PAL_HIGH.
- `#define palTogglePad(port, pad) palTogglePort(port, PAL_PORT_BIT(pad))`
Clears a pad logical state to PAL_LOW.
- `#define palSetPadMode(port, pad, mode) palSetGroupMode(port, PAL_PORT_BIT(pad), 0, mode)`
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`
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_readdlatch(port) ((port)->ODR)`
Reads the output latch.
- `#define pal_lld_writeport(port, bits) ((port)->ODR = (bits))`
Writes on a I/O port.
- `#define pal_lld_setport(port, bits) ((port)->BSRR.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.

6.12.3 Function Documentation

6.12.3.1 `ioportmask_t palReadBus (IOBus * bus)`

Read from an I/O bus.

Note

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

The function internally uses the `palReadGroup()` macro. The use of this function is preferred when you value code size, readability and error checking over speed.

Parameters

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

Returns

The bus logical states.

Function Class:

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

6.12.3.2 `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.

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

Function Class:

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

6.12.3.4 void _pal_lld_init(const PALConfig * config)

STM32 I/O ports configuration.

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

Parameters

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

Function Class:

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

6.12.3.5 void _pal_lld_setgroupmode(ioportid_t port, ioportmask_t mask, iomode_t mode)

Pads mode setup.

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

Note

`PAL_MODE_UNCONNECTED` is implemented as push pull 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.

6.12.4 Define Documentation**6.12.4.1 #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.

6.12.4.2 #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.

6.12.4.3 #define PAL_MODE_INPUT 2

Regular input high-Z pad.

6.12.4.4 #define PAL_MODE_INPUT_PULLUP 3

Input pad with weak pull up resistor.

6.12.4.5 #define PAL_MODE_INPUT_PULLDOWN 4

Input pad with weak pull down resistor.

6.12.4.6 #define PAL_MODE_INPUT_ANALOG 5

Analog input mode.

6.12.4.7 #define PAL_MODE_OUTPUT_PUSH_PULL 6

Push-pull output pad.

6.12.4.8 #define PAL_MODE_OUTPUT_OPENDRAIN 7

Open-drain output pad.

6.12.4.9 #define PAL_LOW 0

Logical low state.

6.12.4.10 #define PAL_HIGH 1

Logical high state.

6.12.4.11 #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.

```
6.12.4.12 #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	<i>width</i>	group width
----	--------------	-------------

Returns

The group mask.

```
6.12.4.13 #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

```
6.12.4.14 #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

```
6.12.4.15 #define pallInit( config ) pal_lld_init(config)
```

PAL subsystem initialization.

Note

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

Parameters

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

Function Class:

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

6.12.4.16 #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 *port* 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.

6.12.4.17 #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.

6.12.4.18 #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.

6.12.4.19 #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	<i>port</i>	port identifier
in	<i>bits</i>	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.

6.12.4.20 #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.

6.12.4.21 #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.

6.12.4.22 #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.

6.12.4.23 #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.

6.12.4.24 #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.

6.12.4.25 #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.

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

Function Class:

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

6.12.4.27 #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.

6.12.4.28 #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.

6.12.4.29 #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.

```
6.12.4.30 #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.

```
6.12.4.31 #define PAL_MODE_ALTERNATE( n )
```

Value:

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

Alternate function.

Parameters

in	<i>n</i>	alternate function selector
----	----------	-----------------------------

```
6.12.4.32 #define PAL_MODE_RESET PAL_STM32_MODE_INPUT
```

This mode is implemented as input.

```
6.12.4.33 #define PAL_MODE_UNCONNECTED PAL_STM32_MODE_OUTPUT
```

This mode is implemented as output.

```
6.12.4.34 #define PAL_MODE_INPUT PAL_STM32_MODE_INPUT
```

Regular input high-Z pad.

```
6.12.4.35 #define PAL_MODE_INPUT_PULLUP
```

Value:

```
(PAL_STM32_MODE_INPUT |           \
             \          \
             PAL_STM32_PUDR_PULLUP)
```

Input pad with weak pull up resistor.

6.12.4.36 #define PAL_MODE_INPUT_PULLDOWN

Value:

```
(PAL_STM32_MODE_INPUT | \  
    PAL_STM32_PUDR_PULLDOWN)
```

Input pad with weak pull down resistor.

6.12.4.37 #define PAL_MODE_INPUT_ANALOG PAL_STM32_MODE_ANALOG

Analog input mode.

6.12.4.38 #define PAL_MODE_OUTPUT_PUSH_PULL

Value:

```
(PAL_STM32_MODE_OUTPUT | \  
    PAL_STM32_OTYPE_PUSH_PULL)
```

Push-pull output pad.

6.12.4.39 #define PAL_MODE_OUTPUT_OPENDRAIN

Value:

```
(PAL_STM32_MODE_OUTPUT | \  
    PAL_STM32_OTYPE_OPENDRAIN)
```

Open-drain output pad.

6.12.4.40 #define PAL_IOPORTS_WIDTH 16

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

6.12.4.41 #define PAL_WHOLE_PORT ((ioportmask_t)0xFFFF)

Whole port mask.

This macro specifies all the valid bits into a port.

6.12.4.42 #define IOPORT1 GPIOA

GPIO port A identifier.

6.12.4.43 #define IOPORT2 GPIOB

GPIO port B identifier.

6.12.4.44 #define IOPORT3 GPIOC

GPIO port C identifier.

6.12.4.45 #define IOPORT4 GPIOD

GPIO port D identifier.

6.12.4.46 #define IOPORT5 GPIOE

GPIO port E identifier.

6.12.4.47 #define IOPORT6 GPIOF

GPIO port F identifier.

6.12.4.48 #define IOPORT7 GPIOG

GPIO port G identifier.

6.12.4.49 #define IOPORT8 GPIOH

GPIO port H identifier.

6.12.4.50 #define IOPORT9 GPIOI

GPIO port I identifier.

6.12.4.51 #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.

6.12.4.52 #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.

6.12.4.53 #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	<i>port</i>	port identifier
----	-------------	-----------------

Returns

The latched logical states.

Function Class:

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

6.12.4.54 #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	<i>port</i>	port identifier
in	<i>bits</i>	bits to be written on the specified port

Function Class:

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

6.12.4.55 #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.

6.12.4.56 #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.

6.12.4.57 #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.

6.12.4.58 #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.

6.12.4.59 #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.

6.12.5 Typedef Documentation

6.12.5.1 `typedef uint32_t ioportmask_t`

Digital I/O port sized unsigned type.

6.12.5.2 `typedef uint32_t iomode_t`

Digital I/O modes.

6.12.5.3 `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.

6.13 PWM Driver

6.13.1 Detailed Description

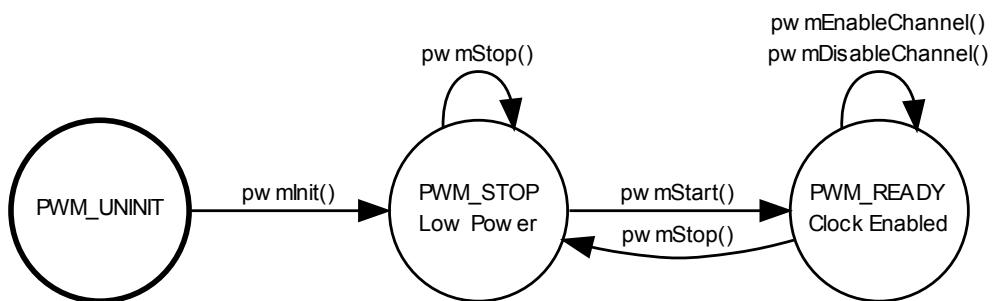
Generic PWM Driver. This module implements a generic PWM (Pulse Width Modulation) driver.

Precondition

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

6.13.2 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).



6.13.3 PWM Operations.

This driver abstracts a generic PWM timer composed of:

- A clock prescaler.

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

A PWM channel output can be in two different states:

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

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

Data Structures

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

Functions

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

Variables

- **PWMDriver PWMD1**
PWMD1 driver identifier.
- **PWMDriver PWMD2**
PWMD2 driver identifier.
- **PWMDriver PWMD3**
PWMD3 driver identifier.
- **PWMDriver PWMD4**
PWMD4 driver identifier.
- **PWMDriver PWMD5**
PWMD5 driver identifier.
- **PWMDriver PWMD8**
PWMD8 driver identifier.

PWM output mode macros

- **#define PWM_OUTPUT_MASK 0x0F**
Standard output modes mask.
- **#define PWM_OUTPUT_DISABLED 0x00**
Output not driven, callback only.
- **#define PWM_OUTPUT_ACTIVE_HIGH 0x01**
Positive PWM logic, active is logic level one.
- **#define PWM_OUTPUT_ACTIVE_LOW 0x02**
Inverse PWM logic, active is logic level zero.

PWM duty cycle conversion

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

Macro Functions

- **#define pwmChangePeriodI(pwmp, period)**
Changes the period the PWM peripheral.
- **#define pwmEnableChannelI(pwmp, channel, width) pwm_lld_enable_channel(pwmp, channel, width)**
Enables a PWM channel.
- **#define pwmDisableChannelI(pwmp, channel) pwm_lld_disable_channel(pwmp, channel)**
Disables a PWM channel.

Configuration options

- `#define STM32_PWM_USE_ADVANCED TRUE`
If advanced timer features switch.
- `#define STM32_PWM_USE_TIM1 TRUE`
PWMD1 driver enable switch.
- `#define STM32_PWM_USE_TIM2 TRUE`
PWMD2 driver enable switch.
- `#define STM32_PWM_USE_TIM3 TRUE`
PWMD3 driver enable switch.
- `#define STM32_PWM_USE_TIM4 TRUE`
PWMD4 driver enable switch.
- `#define STM32_PWM_USE_TIM5 TRUE`
PWMD5 driver enable switch.
- `#define STM32_PWM_USE_TIM8 TRUE`
PWMD8 driver enable switch.
- `#define STM32_PWM_TIM1_IRQ_PRIORITY 7`
PWMD1 interrupt priority level setting.
- `#define STM32_PWM_TIM2_IRQ_PRIORITY 7`
PWMD2 interrupt priority level setting.
- `#define STM32_PWM_TIM3_IRQ_PRIORITY 7`
PWMD3 interrupt priority level setting.
- `#define STM32_PWM_TIM4_IRQ_PRIORITY 7`
PWMD4 interrupt priority level setting.
- `#define STM32_PWM_TIM5_IRQ_PRIORITY 7`
PWMD5 interrupt priority level setting.
- `#define STM32_PWM_TIM8_IRQ_PRIORITY 7`
PWMD8 interrupt priority level setting.

Defines

- `#define PWM_CHANNELS 4`
Number of PWM channels per PWM driver.
- `#define PWM_COMPLEMENTARY_OUTPUT_MASK 0xF0`
Complementary output modes mask.
- `#define PWM_COMPLEMENTARY_OUTPUT_DISABLED 0x00`
Complementary output not driven.
- `#define PWM_COMPLEMENTARY_OUTPUT_ACTIVE_HIGH 0x10`
Complementary output, active is logic level one.
- `#define PWM_COMPLEMENTARY_OUTPUT_ACTIVE_LOW 0x20`
Complementary output, active is logic level zero.
- `#define pwm_lld_change_period(pwmp, period) ((pwmp)->tim->ARR = (uint16_t)((period) - 1))`
Changes the period the PWM peripheral.

Typedefs

- **typedef struct PWMDriver PWMDriver**
Type of a structure representing a PWM driver.
- **typedef void(* pwmcallback_t)(PWMDriver *pwmp)**
PWM notification callback type.
- **typedef uint32_t pwmmode_t**
PWM mode type.
- **typedef uint8_t pwmchannel_t**
PWM channel type.
- **typedef uint16_t pwcnt_t**
PWM counter type.

Enumerations

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

6.13.4 Function Documentation

6.13.4.1 void pwmlInit (void)

PWM Driver initialization.

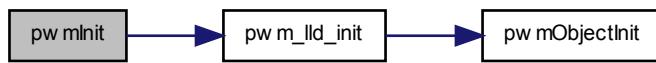
Note

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

Function Class:

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

Here is the call graph for this function:



6.13.4.2 void pwmObjectInit (PWMDriver * pwmp)

Initializes the standard part of a **PWMDriver** structure.

Parameters

out *pwmp* pointer to a **PWMDriver** object

Function Class:

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

6.13.4.3 void pwmStart (PWMDriver * *pwmp*, const PWMConfig * *config*)

Configures and activates the PWM peripheral.

Note

Starting a driver that is already in the `PWM_READY` state disables all the active channels.

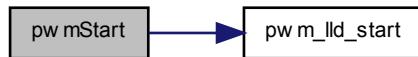
Parameters

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

Function Class:

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

Here is the call graph for this function:

**6.13.4.4 void pwmStop (PWMDriver * *pwmp*)**

Deactivates the PWM peripheral.

Parameters

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

Function Class:

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

Here is the call graph for this function:

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

Function Class:

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

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

Enables a PWM channel.

Precondition

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

Postcondition

The channel is active using the specified configuration.

Note

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

Parameters

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

Function Class:

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

Here is the call graph for this function:



6.13.4.7 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:



6.13.4.8 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:



6.13.4.9 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.

6.13.4.10 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.

6.13.4.11 void pwm_lld_enable_channel (PWMDriver * pwmp, pwmchannel_t channel, pwcnt_t width)

Enables a PWM channel.

Precondition

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

Postcondition

The channel is active using the specified configuration.

Note

The function has effect at the next cycle start.

Parameters

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

Function Class:

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

6.13.4.12 void pwm_lld_disable_channel (PWMDriver * pwmp, pwmchannel_t channel)

Disables a PWM channel.

Precondition

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

Postcondition

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

Note

The function has effect at the next cycle start.

Parameters

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

Function Class:

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

6.13.5 Variable Documentation

6.13.5.1 PWMDriver PWMD1

PWMD1 driver identifier.

Note

The driver PWMD1 allocates the complex timer TIM1 when enabled.

6.13.5.2 PWMDriver PWMD2

PWMD2 driver identifier.

Note

The driver PWMD2 allocates the timer TIM2 when enabled.

6.13.5.3 PWMDriver PWMD3

PWMD3 driver identifier.

Note

The driver PWMD3 allocates the timer TIM3 when enabled.

6.13.5.4 PWMDriver PWMD4

PWMD4 driver identifier.

Note

The driver PWMD4 allocates the timer TIM4 when enabled.

6.13.5.5 PWMDriver PWMD5

PWMD5 driver identifier.

Note

The driver PWMD5 allocates the timer TIM5 when enabled.

6.13.5.6 PWMDriver PWMD8

PWMD8 driver identifier.

Note

The driver PWMD5 allocates the timer TIM5 when enabled.

6.13.6 Define Documentation

6.13.6.1 #define PWM_OUTPUT_MASK 0x0F

Standard output modes mask.

6.13.6.2 #define PWM_OUTPUT_DISABLED 0x00

Output not driven, callback only.

6.13.6.3 #define PWM_OUTPUT_ACTIVE_HIGH 0x01

Positive PWM logic, active is logic level one.

6.13.6.4 #define PWM_OUTPUT_ACTIVE_LOW 0x02

Inverse PWM logic, active is logic level zero.

6.13.6.5 #define PWM_FRACTION_TO_WIDTH(*pwmp*, *denominator*, *numerator*)

Value:

```
((uint16_t) (((uint32_t) (pwmp)->period) *
             (uint32_t) (numerator)) / (uint32_t) (denominator))) \
```

Converts from fraction to pulse width.

Note

Be careful with rounding errors, this is integer math not magic. You can specify tenths of thousandth but make sure you have the proper hardware resolution by carefully choosing the clock source and prescaler settings, see `PWM_COMPUTE_PSC`.

Parameters

in	<i>pwmp</i>	pointer to a <code>PWMDriver</code> object
in	<i>denominator</i>	denominator of the fraction
in	<i>numerator</i>	numerator of the fraction

Returns

The pulse width to be passed to `pwmEnableChannel()`.

Function Class:

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

6.13.6.6 #define PWM_DEGREES_TO_WIDTH(*pwmp*, *degrees*) PWM_FRACTION_TO_WIDTH(*pwmp*, 36000, *degrees*)

Converts from degrees to pulse width.

Note

Be careful with rounding errors, this is integer math not magic. You can specify hundredths of degrees but make sure you have the proper hardware resolution by carefully choosing the clock source and prescaler settings, see `PWM_COMPUTE_PSC`.

Parameters

in	<i>pwmp</i>	pointer to a <code>PWMDriver</code> object
in	<i>degrees</i>	degrees as an integer between 0 and 36000

Returns

The pulse width to be passed to `pwmEnableChannel()`.

Function Class:

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

6.13.6.7 #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.

6.13.6.8 #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 PWMDriver 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.

6.13.6.9 #define pwmEnableChannell(*pwmp*, *channel*, *width*) pwm_lld_enable_channel(*pwmp*, *channel*, *width*)

Enables a PWM channel.

Precondition

The PWM unit must have been activated using [pwmStart \(\)](#).

Postcondition

The channel is active using the specified configuration.

Note

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

Parameters

in	<i>pwmp</i>	pointer to a 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.

6.13.6.10 #define pwmDisableChannell(*pwmp*, *channel*) pwm_lld_disable_channel(*pwmp*, *channel*)

Disables a PWM channel.

Precondition

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

Postcondition

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

Note

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

Parameters

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

Function Class:

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

6.13.6.11 #define PWM_CHANNELS 4

Number of PWM channels per PWM driver.

6.13.6.12 #define PWM_COMPLEMENTARY_OUTPUT_MASK 0xF0

Complementary output modes mask.

Note

This is an STM32-specific setting.

6.13.6.13 #define PWM_COMPLEMENTARY_OUTPUT_DISABLED 0x00

Complementary output not driven.

Note

This is an STM32-specific setting.

6.13.6.14 #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.

6.13.6.15 #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.

6.13.6.16 #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.

6.13.6.17 #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.

6.13.6.18 #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.

6.13.6.19 #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.

6.13.6.20 #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.

```
6.13.6.21 #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.

```
6.13.6.22 #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.

```
6.13.6.23 #define STM32_PWM_TIM1_IRQ_PRIORITY 7
```

PWMD1 interrupt priority level setting.

```
6.13.6.24 #define STM32_PWM_TIM2_IRQ_PRIORITY 7
```

PWMD2 interrupt priority level setting.

```
6.13.6.25 #define STM32_PWM_TIM3_IRQ_PRIORITY 7
```

PWMD3 interrupt priority level setting.

```
6.13.6.26 #define STM32_PWM_TIM4_IRQ_PRIORITY 7
```

PWMD4 interrupt priority level setting.

```
6.13.6.27 #define STM32_PWM_TIM5_IRQ_PRIORITY 7
```

PWMD5 interrupt priority level setting.

```
6.13.6.28 #define STM32_PWM_TIM8_IRQ_PRIORITY 7
```

PWMD8 interrupt priority level setting.

```
6.13.6.29 #define pwm_lld_change_period( pwmp, period ) ((pwmp)->tim->ARR = (uint16_t)((period) - 1))
```

Changes the period the PWM peripheral.

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

Precondition

The PWM unit must have been activated using [pwmStart \(\)](#).

Postcondition

The PWM unit period is changed to the new value.

Note

The function has effect at the next cycle start.

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

Parameters

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

Function Class:

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

6.13.7 Typedef Documentation

6.13.7.1 `typedef struct PWMDriver PWMDriver`

Type of a structure representing a PWM driver.

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

PWM notification callback type.

Parameters

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

6.13.7.3 `typedef uint32_t pwmmode_t`

PWM mode type.

6.13.7.4 `typedef uint8_t pwmchannel_t`

PWM channel type.

6.13.7.5 `typedef uint16_t pwcnt_t`

PWM counter type.

6.13.8 Enumeration Type Documentation

6.13.8.1 `enum pwmstate_t`

Driver state machine possible states.

Enumerator:

PWM_UNINIT Not initialized.

PWM_STOP Stopped.

PWM_READY Ready.

6.14 Serial Driver

6.14.1 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.

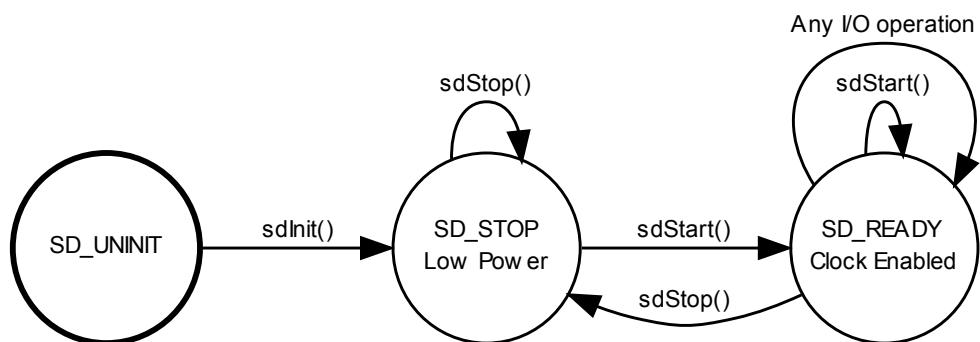
The module also contains functions that make the implementation of the interrupt service routines much easier.

Precondition

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

6.14.2 Driver State Machine

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



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)

- **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)**
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.

Serial status flags

- **#define SD_PARITY_ERROR 32**
Parity error happened.
- **#define SD_FRAMING_ERROR 64**
Framing error happened.
- **#define SD_OVERRUN_ERROR 128**

- `#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.

6.14.3 Function Documentation

6.14.3.1 void sdlInit(void)

Serial Driver initialization.

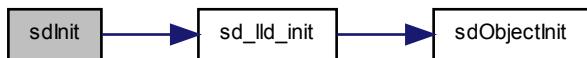
Note

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

Function Class:

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

Here is the call graph for this function:

**6.14.3.2 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.

6.14.3.3 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:



6.14.3.4 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 <code>SerialDriver</code> object
----	--

Function Class:

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

Here is the call graph for this function:



6.14.3.5 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 <code>SerialDriver</code> 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.

6.14.3.6 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 *sdp* pointer to a [SerialDriver](#) 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.

6.14.3.7 CH_IRQ_HANDLER (USART1_IRQHandler)

USART1 interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.14.3.8 CH_IRQ_HANDLER (USART2_IRQHandler)

USART2 interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.14.3.9 CH_IRQ_HANDLER (USART3_IRQHandler)

USART3 interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.14.3.10 CH_IRQ_HANDLER (UART4_IRQHandler)

UART4 interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.14.3.11 CH_IRQ_HANDLER (UART5_IRQHandler)

UART5 interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.14.3.12 CH_IRQ_HANDLER (USART6_IRQHandler)

USART1 interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.14.3.13 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:

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

Function Class:

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

6.14.3.15 void sd_ll_stop (SerialDriver * sdp)

Low level serial driver stop.

De-initializes the USART, stops the associated clock, resets the interrupt vector.

Parameters

in *sdp* pointer to a `SerialDriver` object

Function Class:

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

6.14.4 Variable Documentation**6.14.4.1 SerialDriver SD1**

USART1 serial driver identifier.

6.14.4.2 SerialDriver SD2

USART2 serial driver identifier.

6.14.4.3 SerialDriver SD3

USART3 serial driver identifier.

6.14.4.4 SerialDriver SD4

UART4 serial driver identifier.

6.14.4.5 SerialDriver SD5

UART5 serial driver identifier.

6.14.4.6 SerialDriver SD6

USART6 serial driver identifier.

6.14.5 Define Documentation**6.14.5.1 #define SD_PARITY_ERROR 32**

Parity error happened.

6.14.5.2 #define SD_FRAMING_ERROR 64

Framing error happened.

6.14.5.3 #define SD_OVERRUN_ERROR 128

Overflow happened.

6.14.5.4 #define SD_NOISE_ERROR 256

Noise on the line.

6.14.5.5 #define SD_BREAK_DETECTED 512

Break detected.

6.14.5.6 #define SERIAL_DEFAULT_BITRATE 38400

Default bit rate.

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

6.14.5.7 #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.

6.14.5.8 #define _serial_driver_methods _base_asynchronous_channel_methods

[SerialDriver](#) specific methods.

6.14.5.9 #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.

```
6.14.5.10 #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

[chIOGetWouldBlock\(\)](#)

Deprecated

Function Class:

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

```
6.14.5.11 #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.

```
6.14.5.12 #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.

6.14.5.13 #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

[chIOGet\(\)](#)

Function Class:

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

6.14.5.14 #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.

6.14.5.15 #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.

6.14.5.16 #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.

6.14.5.17 `#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.

6.14.5.18 `#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

`chIORReadTimeout()`

Function Class:

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

6.14.5.19 `#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

`chIORReadTimeout()`

Function Class:

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

```
6.14.5.20 #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.

```
6.14.5.21 #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.

```
6.14.5.22 #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.

```
6.14.5.23 #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.

```
6.14.5.24 #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.

6.14.5.25 #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.

6.14.5.26 #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.

6.14.5.27 #define STM32_SERIAL_USART1_PRIORITY 12

USART1 interrupt priority level setting.

6.14.5.28 #define STM32_SERIAL_USART2_PRIORITY 12

USART2 interrupt priority level setting.

6.14.5.29 #define STM32_SERIAL_USART3_PRIORITY 12

USART3 interrupt priority level setting.

6.14.5.30 #define STM32_SERIAL_UART4_PRIORITY 12

UART4 interrupt priority level setting.

6.14.5.31 #define STM32_SERIAL_UART5_PRIORITY 12

UART5 interrupt priority level setting.

6.14.5.32 #define STM32_SERIAL_USART6_PRIORITY 12

USART6 interrupt priority level setting.

6.14.5.33 #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.

6.14.5.34 #define USART_CR2_STOP1_BITS (0 << 12)

CR2 1 stop bit value.

6.14.5.35 #define USART_CR2_STOP0P5_BITS (1 << 12)

CR2 0.5 stop bit value.

6.14.5.36 #define USART_CR2_STOP2_BITS (2 << 12)

CR2 2 stop bit value.

6.14.5.37 #define USART_CR2_STOP1P5_BITS (3 << 12)

CR2 1.5 stop bit value.

6.14.6 Typedef Documentation

6.14.6.1 typedef struct SerialDriver SerialDriver

Structure representing a serial driver.

6.14.7 Enumeration Type Documentation

6.14.7.1 enum sdstate_t

Driver state machine possible states.

Enumerator:

SD_UNINIT Not initialized.

SD_STOP Stopped.

SD_READY Ready.

6.15 SDC Driver

6.15.1 Detailed Description

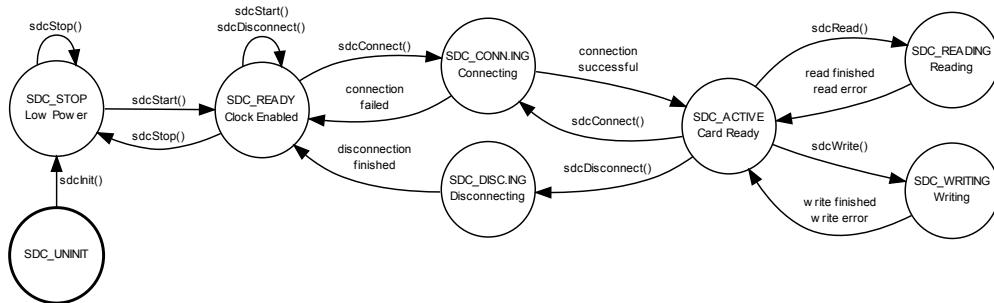
Generic SD Card Driver. This module implements a generic SDC (Secure Digital Card) driver.

Precondition

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

6.15.2 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).



6.15.3 SDC Operations.

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

Functions

- void **sdcInit** (void)
SDC Driver initialization.
- void **sdcObjectInit** (SDCDriver *sdcp)
Initializes the standard part of a SD CDriver structure.
- void **sdcStart** (SDCDriver *sdcp, const SDCCConfig *config)
Configures and activates the SDC peripheral.
- void **sdcStop** (SDCDriver *sdcp)
Deactivates the SDC peripheral.
- bool_t **sdcConnect** (SDCDriver *sdcp)
Performs the initialization procedure on the inserted card.
- bool_t **sdcDisconnect** (SDCDriver *sdcp)
Brings the driver in a state safe for card removal.
- bool_t **sdcRead** (SDCDriver *sdcp, uint32_t startblk, uint8_t *buf, uint32_t n)
Reads one or more blocks.
- bool_t **sdcWrite** (SDCDriver *sdcp, uint32_t startblk, const uint8_t *buf, uint32_t n)
Writes one or more blocks.
- bool_t **_sdc_wait_for_transfer_state** (SDCDriver *sdcp)
Wait for the card to complete pending operations.

SD cart types

- #define **SDC_MODE_CARDTYPE_MASK** 0xF
Card type mask.

- #define **SDC_MODE_CARDTYPE_SDV11** 0
Card is SD V1.1.
- #define **SDC_MODE_CARDTYPE_SDV20** 1
Card is SD V2.0.
- #define **SDC_MODE_CARDTYPE_MMC** 2
Card is MMC.
- #define **SDC_MODE_HIGH_CAPACITY** 0x10
High cap.card.

SDC configuration options

- #define **SDC_INIT_RETRY** 100
Number of initialization attempts before rejecting the card.
- #define **SDC_MMCSUPPORT** FALSE
Include support for MMC cards.
- #define **SDC_NICE_WAITING** TRUE
Delays insertions.

R1 response utilities

- #define **SDC_R1_ERROR**(r1) (((r1) & SDC_R1_ERROR_MASK) != 0)
Evaluates to TRUE if the R1 response contains error flags.
- #define **SDC_R1_STS**(r1) (((r1) >> 9) & 15)
Returns the status field of an R1 response.
- #define **SDC_R1_IS_CARD_LOCKED**(r1) (((r1) >> 21) & 1)
Evaluates to TRUE if the R1 response indicates a locked card.

Macro Functions

- #define **sdcGetDriverState**(sdcp) ((sdcp)->state)
Returns the driver state.
- #define **sdclsCardInserted**(sdcp) (sdc_lld_is_card_inserted(sdcp))
Returns the card insertion status.
- #define **sdclsWriteProtected**(sdcp) (sdc_lld_is_write_protected(sdcp))
Returns the write protect status.

Defines

- #define **SDC_BLOCK_SIZE** 512
- #define **SDC_CMD8_PATTERN** 0x000001AA
Fixed pattern for CMD8.
- #define **SDC_R1_ERROR_MASK** 0xFDFF008
Mask of error bits in R1 responses.

Enumerations

- enum **sdcstate_t** {

 SDC_UNINIT = 0, **SDC_STOP** = 1, **SDC_READY** = 2, **SDC_CONNECTING** = 3,

 SDC_DISCONNECTING = 4, **SDC_ACTIVE** = 5, **SDC_READING** = 6, **SDC_WRITING** = 7
 }
- Driver state machine possible states.*

6.15.4 Function Documentation

6.15.4.1 void sdclInit (void)

SDC Driver initialization.

Note

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

Function Class:

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

6.15.4.2 void sdcObjectInit (SDCDriver * *sdcp*)

Initializes the standard part of a `SDCDriver` structure.

Parameters

out *sdcp* pointer to the `SDCDriver` object

Function Class:

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

6.15.4.3 void sdcStart (SDCDriver * *sdcp*, const SDCCConfig * *config*)

Configures and activates the SDC peripheral.

Parameters

in *sdcp* pointer to the `SDCDriver` object

in *config* pointer to the `SDCCConfig` object, can be `NULL` if the driver supports a default configuration or requires no configuration

Function Class:

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

6.15.4.4 void sdcStop (SDCDriver * *sdcp*)

Deactivates the SDC peripheral.

Parameters

in *sdcp* pointer to the `SDCDriver` object

Function Class:

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

6.15.4.5 bool_t sdcConnect (SDCDriver * *sdcp*)

Performs the initialization procedure on the inserted card.

This function should be invoked when a card is inserted and brings the driver in the `SDC_ACTIVE` state where it

is possible to perform read and write operations.

Parameters

in *sdcp* pointer to the SDCDriver object

Returns

The operation status.

Return values

FALSE operation succeeded, the driver is now in the SDC_ACTIVE state.
TRUE operation failed.

Function Class:

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

6.15.4.6 bool_t sdcDisconnect (SDCDriver * *sdcp*)

Brings the driver in a state safe for card removal.

Parameters

in *sdcp* pointer to the SDCDriver object

Returns

The operation status.

Return values

FALSE the operation succeeded and the driver is now in the SDC_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:

**6.15.4.7 bool_t sdcRead (SDCDriver * *sdcp*, uint32_t *startblk*, uint8_t * *buf*, uint32_t *n*)**

Reads one or more blocks.

Precondition

The driver must be in the SDC_ACTIVE state after a successful [sdcConnect\(\)](#) invocation.

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
in	<i>startblk</i>	first block to read
out	<i>buf</i>	pointer to the read buffer
in	<i>n</i>	number of blocks to read

Returns

The operation status.

Return values

<i>FALSE</i>	operation succeeded, the requested blocks have been read.
<i>TRUE</i>	operation failed, the state of the buffer is uncertain.

Function Class:

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

6.15.4.8 bool_t sdcWrite (SDCDriver * *sdcp*, uint32_t *startblk*, const uint8_t * *buf*, uint32_t *n*)

Writes one or more blocks.

Precondition

The driver must be in the SDC_ACTIVE state after a successful [sdcConnect\(\)](#) invocation.

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
in	<i>startblk</i>	first block to write
out	<i>buf</i>	pointer to the write buffer
in	<i>n</i>	number of blocks to write

Returns

The operation status.

Return values

<i>FALSE</i>	operation succeeded, the requested blocks have been written.
<i>TRUE</i>	operation failed.

Function Class:

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

6.15.4.9 bool_t _sdc_wait_for_transfer_state (SDCDriver * *sdcp*)

Wait for the card to complete pending operations.

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
----	-------------	---------------------------------

Returns

The operation status.

Return values

<i>FALSE</i>	the card is now in transfer state.
--------------	------------------------------------

TRUE an error occurred while waiting or the card is in an unexpected state.

Function Class:

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

6.15.5 Define Documentation

6.15.5.1 #define SDC_BLOCK_SIZE 512

Fixed block size.

6.15.5.2 #define SDC_CMD8_PATTERN 0x000001AA

Fixed pattern for CMD8.

6.15.5.3 #define SDC_MODE_CARDTYPE_MASK 0xF

Card type mask.

6.15.5.4 #define SDC_MODE_CARDTYPE_SDV11 0

Card is SD V1.1.

6.15.5.5 #define SDC_MODE_CARDTYPE_SDV20 1

Card is SD V2.0.

6.15.5.6 #define SDC_MODE_CARDTYPE_MMC 2

Card is MMC.

6.15.5.7 #define SDC_MODE_HIGH_CAPACITY 0x10

High cap.card.

6.15.5.8 #define SDC_R1_ERROR_MASK 0xFDFFFE008

Mask of error bits in R1 responses.

6.15.5.9 #define SDC_INIT_RETRY 100

Number of initialization attempts before rejecting the card.

Note

Attempts are performed at 10mS intervals.

6.15.5.10 #define SDC_MMIC_SUPPORT FALSE

Include support for MMC cards.

Note

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

6.15.5.11 #define SDC_NICE_WAITING TRUE

Delays insertions.

If enabled this options inserts delays into the MMC waiting routines releasing some extra CPU time for the threads with lower priority, this may slow down the driver a bit however.

6.15.5.12 #define SDC_R1_ERROR(*r1*) (((*r1*) & SDC_R1_ERROR_MASK) != 0)

Evaluates to TRUE if the R1 response contains error flags.

Parameters

in *r1* the r1 response

6.15.5.13 #define SDC_R1_STS(*r1*) (((*r1*) >> 9) & 15)

Returns the status field of an R1 response.

Parameters

in *r1* the r1 response

6.15.5.14 #define SDC_R1_IS_CARD_LOCKED(*r1*) (((*r1*) >> 21) & 1)

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

Parameters

in *r1* the r1 response

6.15.5.15 #define sdcGetDriverState(*sdcp*) ((*sdcp*)->state)

Returns the driver state.

Parameters

in *sdcp* pointer to the SDCDriver object

Returns

The driver state.

Function Class:

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

6.15.5.16 #define sdclsCardInserted(*sdcp*) (sdc_lld_is_card_inserted(*sdcp*))

Returns the card insertion status.

Note

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

Parameters

in *sdcp* pointer to the `SDCDriver` object

Returns

The card state.

Return values

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

Function Class:

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

6.15.5.17 #define sdclsWriteProtected(*sdcp*) (sdc_lld_is_write_protected(*sdcp*))

Returns the write protect status.

Note

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

Parameters

in *sdcp* pointer to the `SDCDriver` object

Returns

The card state.

Return values

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

Function Class:

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

6.15.6 Enumeration Type Documentation

6.15.6.1 enum `sdcstate_t`

Driver state machine possible states.

Enumerator:

<i>SDC_UNINIT</i>	Not initialized.
<i>SDC_STOP</i>	Stopped.

- SDC_READY** Ready.
- SDC_CONNECTING** Card connection in progress.
- SDC_DISCONNECTING** Card disconnection in progress.
- SDC_ACTIVE** Card initialized.
- SDC_READING** Read operation in progress.
- SDC_WRITING** Write operation in progress.

6.16 SPI Driver

6.16.1 Detailed Description

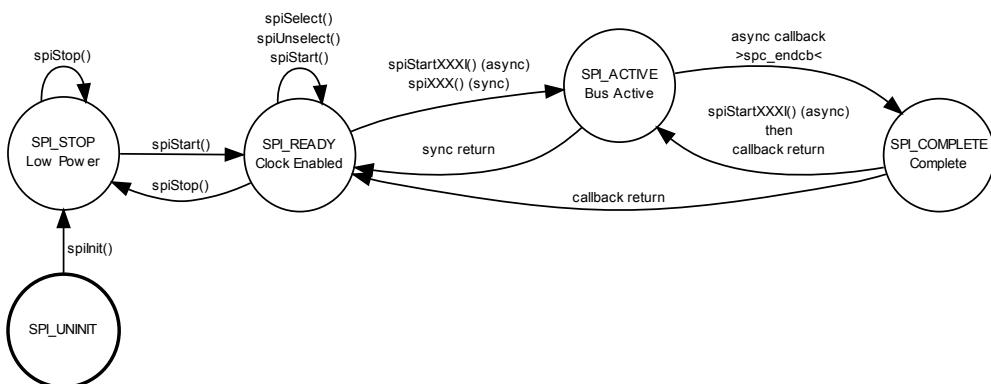
Generic SPI Driver. This module implements a generic SPI (Serial Peripheral Interface) driver allowing bidirectional and monodirectional transfers, complex atomic transactions are supported as well.

Precondition

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

6.16.2 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).



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

Functions

- void **spiInit** (void)

SPI Driver initialization.
- void **spiObjectInit** (SPIDriver *spip)

Initializes the standard part of a `SPIDriver` structure.
- void **spiStart** (SPIDriver *spip, const SPIConfig *config)

Configures and activates the SPI peripheral.
- void **spiStop** (SPIDriver *spip)

Deactivates the SPI peripheral.
- void **spiSelect** (SPIDriver *spip)

Asserts the slave select signal and prepares for transfers.
- void **spiUnselect** (SPIDriver *spip)

Deasserts the slave select signal.
- void **spiStartIgnore** (SPIDriver *spip, size_t n)

Ignores data on the SPI bus.
- void **spiStartExchange** (SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)

Exchanges data on the SPI bus.
- void **spiStartSend** (SPIDriver *spip, size_t n, const void *txbuf)

Sends data over the SPI bus.
- void **spiStartReceive** (SPIDriver *spip, size_t n, void *rxbuf)

Receives data from the SPI bus.
- void **spiIgnore** (SPIDriver *spip, size_t n)

Ignores data on the SPI bus.
- void **spiExchange** (SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)

Exchanges data on the SPI bus.
- void **spiSend** (SPIDriver *spip, size_t n, const void *txbuf)

Sends data over the SPI bus.
- void **spiReceive** (SPIDriver *spip, size_t n, void *rxbuf)

Receives data from the SPI bus.
- void **spiAcquireBus** (SPIDriver *spip)

Gains exclusive access to the SPI bus.
- void **spiReleaseBus** (SPIDriver *spip)

Releases exclusive access to the SPI bus.
- void **spi_lld_init** (void)

Low level SPI driver initialization.
- void **spi_lld_start** (SPIDriver *spip)

Configures and activates the SPI peripheral.
- void **spi_lld_stop** (SPIDriver *spip)

Deactivates the SPI peripheral.
- void **spi_lld_select** (SPIDriver *spip)

Asserts the slave select signal and prepares for transfers.
- void **spi_lld_unselect** (SPIDriver *spip)

Deasserts the slave select signal.
- void **spi_lld_ignore** (SPIDriver *spip, size_t n)

Ignores data on the SPI bus.
- void **spi_lld_exchange** (SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)

Exchanges data on the SPI bus.
- void **spi_lld_send** (SPIDriver *spip, size_t n, const void *txbuf)

Sends data over the SPI bus.
- void **spi_lld_receive** (SPIDriver *spip, size_t n, void *rxbuf)

Receives data from the SPI bus.
- uint16_t **spi_lld_polled_exchange** (SPIDriver *spip, uint16_t frame)

Exchanges one frame using a polled wait.

Variables

- **SPIDriver SPID1**
SPI1 driver identifier.
- **SPIDriver SPID2**
SPI2 driver identifier.
- **SPIDriver SPID3**
SPI3 driver identifier.

SPI configuration options

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

Macro Functions

- **#define 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.

6.16.3 Function Documentation

6.16.3.1 void spilinit (void)

SPI Driver initialization.

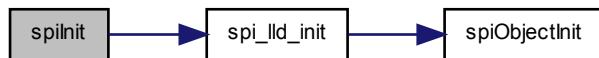
Note

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

Function Class:

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

Here is the call graph for this function:

**6.16.3.2 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.

6.16.3.3 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:

**6.16.3.4 void spiStop (**SPIDriver** * *spip*)**

Deactivates the SPI peripheral.

Note

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

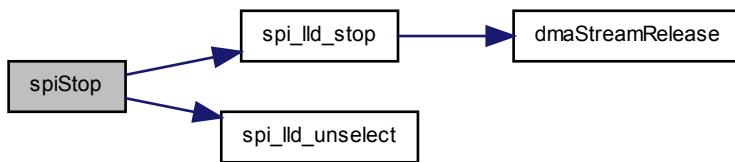
Parameters

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:

**6.16.3.5 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.

6.16.3.6 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.

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

Function Class:

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

6.16.3.8 void spiStartExchange (`SPIDriver * spip`, `size_t n`, `const void * txbuf`, `void * rdbuf`)

Exchanges data on the SPI bus.

This asynchronous function starts a simultaneous transmit/receive operation.

Precondition

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

Postcondition

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

Note

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

Parameters

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

Function Class:

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

6.16.3.9 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.

6.16.3.10 void spiStartReceive (`SPIDriver * spip, size_t n, void * rdbuf`)

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>rdbuf</i>	the pointer to the receive buffer

Function Class:

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

6.16.3.11 void spilgnore (`SPIDriver * spip, size_t n`)

Ignores data on the SPI bus.

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

Precondition

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

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

Parameters

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

Function Class:

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

6.16.3.12 void spiExchange (*SPIDriver* * *spip*, *size_t* *n*, const void * *txbuf*, void * *rxbuf*)

Exchanges data on the SPI bus.

This synchronous function performs a simultaneous transmit/receive operation.

Precondition

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

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

Note

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

Parameters

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

6.16.3.13 void spiSend (*SPIDriver* * *spip*, *size_t* *n*, const void * *txbuf*)

Sends data over the SPI bus.

This synchronous function performs a transmit operation.

Precondition

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

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

Note

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

Parameters

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

6.16.3.14 void spiReceive (*SPIDriver* * *spip*, *size_t* *n*, void * *rxbuf*)

Receives data from the SPI bus.

This synchronous function performs a receive operation.

Precondition

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

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

Note

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

Parameters

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

Function Class:

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

6.16.3.15 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	<i>spip</i>	pointer to the <code>SPIDriver</code> object
----	-------------	--

Function Class:

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

6.16.3.16 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	<i>spip</i>	pointer to the <code>SPIDriver</code> object
----	-------------	--

Function Class:

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

6.16.3.17 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:

**6.16.3.18 void spi_lld_start (SPIDriver * *spip*)**

Configures and activates the SPI peripheral.

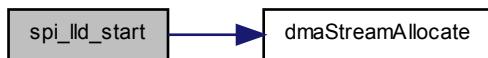
Parameters

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:

**6.16.3.19 void spi_lld_stop (SPIDriver * *spip*)**

Deactivates the SPI peripheral.

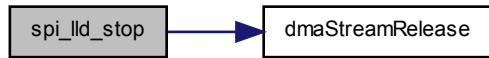
Parameters

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:



6.16.3.20 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.

6.16.3.21 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.

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

Function Class:

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

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

Function Class:

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

6.16.3.24 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 SPIDriver object
in	<i>n</i>	number of words to send
in	<i>txbuf</i>	the pointer to the transmit buffer

Function Class:

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

6.16.3.25 void spi_lld_receive (**SPIDriver * *spip*, **size_t** *n*, **void** * *rxbuf*)**

Receives data from the SPI bus.

This asynchronous function starts a receive operation.

Postcondition

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

Note

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

Parameters

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

6.16.3.26 `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.

6.16.4 Variable Documentation**6.16.4.1 SPIDriver SPID1**

SPI1 driver identifier.

6.16.4.2 SPIDriver SPID2

SPI2 driver identifier.

6.16.4.3 SPIDriver SPID3

SPI3 driver identifier.

6.16.5 Define Documentation**6.16.5.1 `#define SPI_USE_WAIT TRUE`**

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

6.16.5.2 #define SPI_USE_MUTUAL_EXCLUSION TRUE

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

Note

Disabling this option saves both code and data space.

6.16.5.3 #define spiSelect(*spip*)**Value:**

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

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.

6.16.5.4 #define spiUnselect(*spip*)**Value:**

```
{
    spi_lld_unselect(spip);
}
```

Deasserts the slave select signal.

The previously selected peripheral is unselected.

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.

6.16.5.5 #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 <code>SPI_driver</code> object
in	<i>n</i>	number of words to be ignored

Function Class:

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

6.16.5.6 #define spiStartExchange(*spip*, *n*, *txbuf*, *rxbuf*)**Value:**

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

Exchanges data on the SPI bus.

This asynchronous function starts a simultaneous transmit/receive operation.

Precondition

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

Postcondition

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

Note

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

Parameters

in	<i>spip</i>	pointer to the <code>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:

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

6.16.5.7 #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.

6.16.5.8 #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	<code>spip</code>	pointer to the <code>SPIDriver</code> object
in	<code>n</code>	number of words to receive
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.

6.16.5.9 #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 SPIDriver object
in	<i>frame</i>	the data frame to send over the SPI bus

Returns

The received data frame from the SPI bus.

6.16.5.10 #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	<i>spip</i>	pointer to the SPIDriver object
----	-------------	---

Function Class:

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

6.16.5.11 #define _spi_wakeup_isr(*spip*)**Value:**

```
{
    if ((spip)->thread != NULL) {
        Thread *tp = (spip)->thread;
        (spip)->thread = NULL;
        chSysLockFromIsr();
        chSchReadyI(tp);
        chSysUnlockFromIsr();
    }
}
```

Wakes up the waiting thread.

Parameters

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

Function Class:

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

6.16.5.12 #define _spi_isr_code(*spip*)

Value:

```
{
    if ((spip)->config->end_cb) {
        (spip)->state = SPI_COMPLETE;
        (spip)->config->end_cb(spip);
        if ((spip)->state == SPI_COMPLETE)
            (spip)->state = SPI_READY;
    }
    else
        (spip)->state = SPI_READY;
    _spi_wakeup_isr(spip);
}
```

Common ISR code.

This code handles the portable part of the ISR code:

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

Note

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

Parameters

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

Function Class:

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

6.16.5.13 #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.

6.16.5.14 #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.

6.16.5.15 #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.

6.16.5.16 #define STM32_SPI_SPI1_IRQ_PRIORITY 10

SPI1 interrupt priority level setting.

6.16.5.17 #define STM32_SPI_SPI2_IRQ_PRIORITY 10

SPI2 interrupt priority level setting.

6.16.5.18 #define STM32_SPI_SPI3_IRQ_PRIORITY 10

SPI3 interrupt priority level setting.

6.16.5.19 #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.

6.16.5.20 #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.

```
6.16.5.21 #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.

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

SPI DMA error hook.

```
6.16.5.23 #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.

```
6.16.5.24 #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.

```
6.16.5.25 #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.

```
6.16.5.26 #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.

```
6.16.5.27 #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.

```
6.16.5.28 #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.

6.16.6 Typedef Documentation

6.16.6.1 `typedef struct SPIDriver SPIDriver`

Type of a structure representing an SPI driver.

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

SPI notification callback type.

Parameters

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

6.16.7 Enumeration Type Documentation

6.16.7.1 `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.

6.17 Time Measurement Driver.

6.17.1 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.

6.17.2 Function Documentation

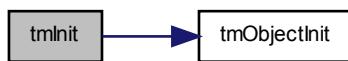
6.17.2.1 void `tmlInit (void)`

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:



6.17.2.2 void `tmObjectInit (TimeMeasurement * tmp)`

Initializes a `TimeMeasurement` object.

Parameters

out `tmp` pointer to a `TimeMeasurement` structure

Function Class:

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

6.17.3 Define Documentation

6.17.3.1 #define tmStartMeasurement(*tmp*) (*tmp*)>start(*tmp*)

Starts a measurement.

Precondition

The [TimeMeasurement](#) must be initialized.

Note

This function can be invoked in any context.

Parameters

in, out *tmp* pointer to a [TimeMeasurement](#) structure

Function Class:

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

6.17.3.2 #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.

6.17.4 Typedef Documentation**6.17.4.1 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.

6.18 UART Driver

6.18.1 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.

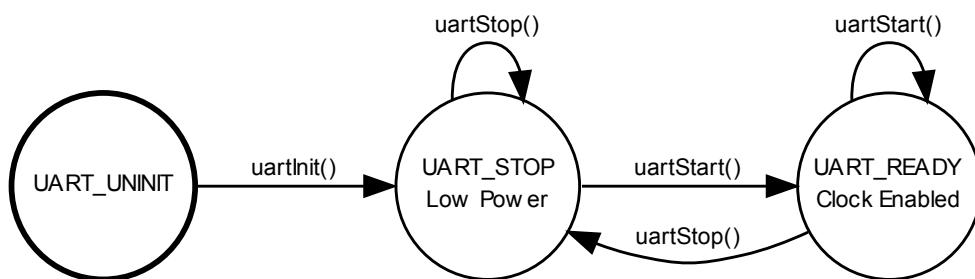
If your application requires a synchronous buffered driver then the [Serial Driver](#) should be used instead.

Precondition

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

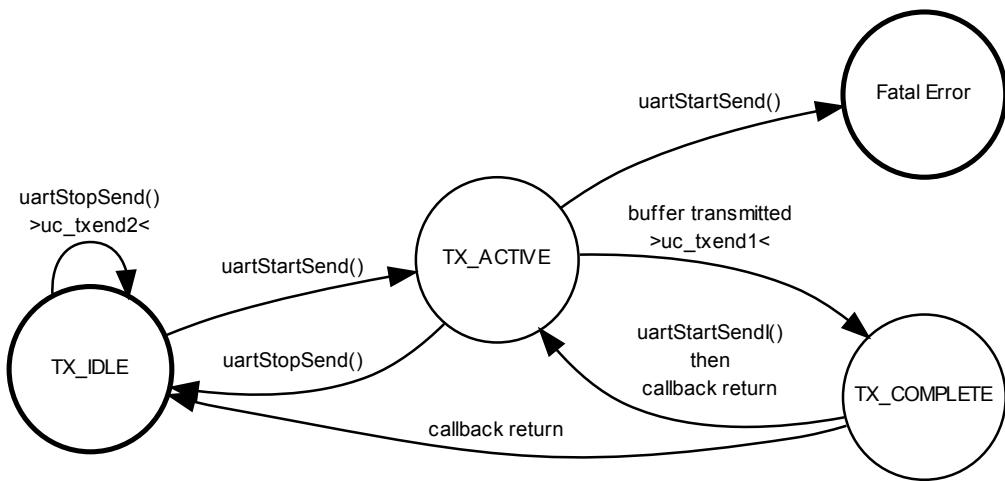
6.18.2 Driver State Machine

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).



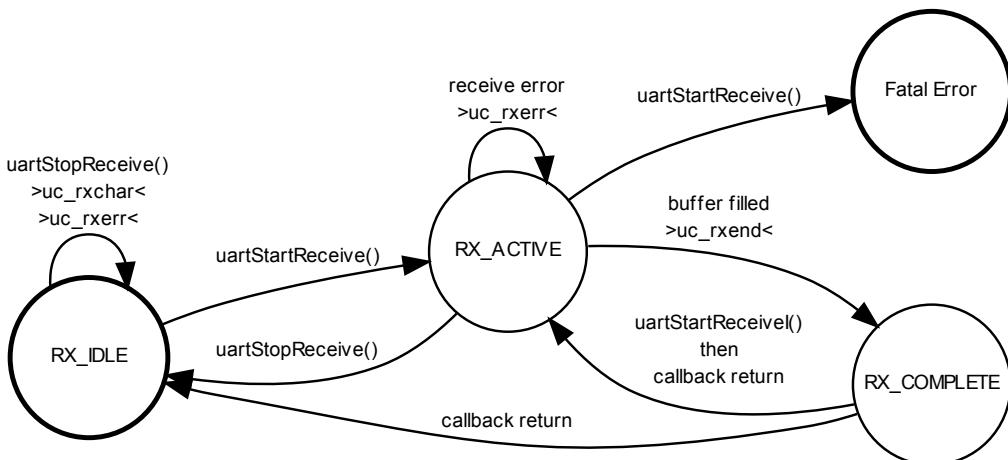
6.18.2.1 Transmitter sub State Machine

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.



6.18.2.2 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 `uartStartReceive1` (`UARTDriver` *uartp, size_t n, void *rxbuf)
Starts a receive operation on the UART peripheral.
- size_t `uartStopReceive` (`UARTDriver` *uartp)
Stops any ongoing receive operation.
- size_t `uartStopReceive1` (`UARTDriver` *uartp)
Stops any ongoing receive operation.
- `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)
Starts a receive operation on the UART peripheral.
- size_t `uart_lld_stop_receive` (`UARTDriver` *uartp)
Stops any ongoing receive operation.

Variables

- `UARTDriver UARD1`
USART1 UART driver identifier.
- `UARTDriver UARD2`
USART2 UART driver identifier.
- `UARTDriver UARD3`
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)`

- `#define STM32_UART_USART2_RX_DMA_STREAM STM32_DMA_STREAM_ID(1, 6)`
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(* uar tcb_t)(UARTDriver *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.

6.18.3 Function Documentation

6.18.3.1 void uartInit (void)

UART Driver initialization.

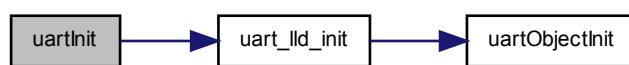
Note

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:



6.18.3.2 void uartObjectInit (**UARTDriver * *uartp*)**

Initializes the standard part of a **UARTDriver** structure.

Parameters

out *uartp* pointer to the **UARTDriver** object

Function Class:

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

6.18.3.3 void uartStart (**UARTDriver * *uartp*, const **UARTConfig** * *config*)**

Configures and activates the UART peripheral.

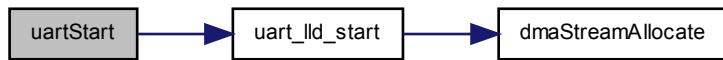
Parameters

in	<i>uartp</i> pointer to the UARTDriver object
in	<i>config</i> pointer to the UARTConfig object

Function Class:

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

Here is the call graph for this function:

**6.18.3.4 void uartStop (**UARTDriver** * *uartp*)**

Deactivates the UART peripheral.

Parameters

in *uartp* pointer to the **UARTDriver** object

Function Class:

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

Here is the call graph for this function:



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

Function Class:

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

Here is the call graph for this function:



6.18.3.6 void uartStartSendl (**UARTDriver** * *uartp*, **size_t** *n*, const void * *txbuf*)

Starts a transmission on the UART peripheral.

Note

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

This function has to be invoked from a lock zone.

Parameters

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

Function Class:

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

Here is the call graph for this function:

**6.18.3.7 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:

**6.18.3.8 size_t uartStopSendl (**UARTDriver** * *uartp*)**

Stops any ongoing transmission.

Note

Stopping a transmission also suppresses the transmission callbacks.
This function has to be invoked from a lock zone.

Parameters

in *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:

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

in *n* number of data frames to send

in *rxbuf* 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:



6.18.3.10 void uartStartReceive(**UARTDriver** * *uartp*, **size_t** *n*, **void** * *rxbuf*)

Starts a receive operation on the UART peripheral.

Note

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

This function has to be invoked from a lock zone.

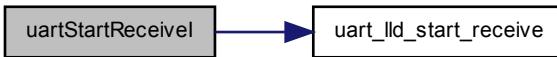
Parameters

in	<i>uartp</i>	pointer to the UARTDriver object
in	<i>n</i>	number of data frames to 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:



6.18.3.11 **size_t** uartStopReceive(**UARTDriver** * *uartp*)

Stops any ongoing receive operation.

Note

Stopping a receive operation also suppresses the receive callbacks.

Parameters

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

Returns

The number of data frames not received by the stopped receive operation.

Return values

0 There was no receive operation in progress.

Function Class:

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

Here is the call graph for this function:

**6.18.3.12 size_t uartStopReceive([UARTDriver](#) * *uartp*)**

Stops any ongoing receive operation.

Note

Stopping a receive operation also suppresses the receive callbacks.
This function has to be invoked from a lock zone.

Parameters

in *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:



6.18.3.13 CH_IRQ_HANDLER (USART1_IRQHandler)

USART1 IRQ handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.18.3.14 CH_IRQ_HANDLER (USART2_IRQHandler)

USART2 IRQ handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.18.3.15 CH_IRQ_HANDLER (USART3_IRQHandler)

USART3 IRQ handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.18.3.16 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:

**6.18.3.17 void uart_lld_start (UARTDriver * uartp)**

Configures and activates the UART peripheral.

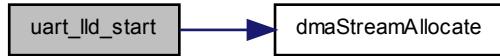
Parameters

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:



6.18.3.18 void uart_lld_stop (**UARTDriver** * *uartp*)

Deactivates the UART peripheral.

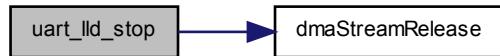
Parameters

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:



6.18.3.19 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 *uartp* pointer to the **UARTDriver** object
in *n* number of data frames to send
in *txbuf* the pointer to the transmit buffer

Function Class:

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

6.18.3.20 size_t uart_lld_stop_send (**UARTDriver * *uartp*)**

Stops any ongoing transmission.

Note

Stopping a transmission also suppresses the transmission callbacks.

Parameters

in *uartp* pointer to the **UARTDriver** object

Returns

The number of data frames not transmitted by the stopped transmit operation.

Function Class:

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

6.18.3.21 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 *uartp* pointer to the **UARTDriver** object

in *n* number of data frames to send

out *rxbuf* the pointer to the receive buffer

Function Class:

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

6.18.3.22 **size_t uart_lld_stop_receive (**UARTDriver** * *uartp*)**

Stops any ongoing receive operation.

Note

Stopping a receive operation also suppresses the receive callbacks.

Parameters

in *uartp* pointer to the **UARTDriver** object

Returns

The number of data frames not received by the stopped receive operation.

Function Class:

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

6.18.4 Variable Documentation

6.18.4.1 **UARTDriver UARTD1**

USART1 UART driver identifier.

6.18.4.2 **UARTDriver UARTD2**

USART2 UART driver identifier.

6.18.4.3 **UARTDriver UARTD3**

USART3 UART driver identifier.

6.18.5 Define Documentation

6.18.5.1 `#define UART_NO_ERROR 0`

No pending conditions.

6.18.5.2 `#define UART_PARITY_ERROR 4`

Parity error happened.

6.18.5.3 `#define UART_FRAMING_ERROR 8`

Framing error happened.

6.18.5.4 `#define UART_OVERRUN_ERROR 16`

Overflow happened.

6.18.5.5 `#define UART_NOISE_ERROR 32`

Noise on the line.

6.18.5.6 `#define UART_BREAK_DETECTED 64`

Break detected.

6.18.5.7 `#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`.

6.18.5.8 #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.

6.18.5.9 #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.

6.18.5.10 #define STM32_UART_USART1_IRQ_PRIORITY 12

USART1 interrupt priority level setting.

6.18.5.11 #define STM32_UART_USART2_IRQ_PRIORITY 12

USART2 interrupt priority level setting.

6.18.5.12 #define STM32_UART_USART3_IRQ_PRIORITY 12

USART3 interrupt priority level setting.

6.18.5.13 #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.

6.18.5.14 #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.

6.18.5.15 #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.

6.18.5.16 #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.

6.18.5.17 #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.

6.18.5.18 #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.

6.18.5.19 #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.

6.18.5.20 #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.

6.18.5.21 #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.

6.18.5.22 #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.

6.18.6 Typedef Documentation

6.18.6.1 **typedef uint32_t uartflags_t**

UART driver condition flags type.

6.18.6.2 **typedef struct UARTDriver UARTDriver**

Structure representing an UART driver.

6.18.6.3 **typedef void(* uartcb_t)(UARTDriver *uartp)**

Generic UART notification callback type.

Parameters

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

6.18.6.4 **typedef void(* uartccb_t)(UARTDriver *uartp, uint16_t c)**

Character received UART notification callback type.

Parameters

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

6.18.6.5 **typedef void(* uarteccb_t)(UARTDriver *uartp, uartflags_t e)**

Receive error UART notification callback type.

Parameters

in *uartp* pointer to the [UARTDriver](#) object
in *e* receive error mask

6.18.7 Enumeration Type Documentation

6.18.7.1 enum uartstate_t

Driver state machine possible states.

Enumerator:

UART_UNINIT Not initialized.

UART_STOP Stopped.

UART_READY Ready.

6.18.7.2 enum uartxstate_t

Transmitter state machine states.

Enumerator:

UART_TX_IDLE Not transmitting.

UART_TX_ACTIVE Transmitting.

UART_TX_COMPLETE Buffer complete.

6.18.7.3 enum uartrxstate_t

Receiver state machine states.

Enumerator:

UART_RX_IDLE Not receiving.

UART_RX_ACTIVE Receiving.

UART_RX_COMPLETE Buffer complete.

6.19 STM32F4xx Drivers

6.19.1 Detailed Description

This section describes all the supported drivers on the STM32F4xx platform and the implementation details of the single drivers.

Modules

- [STM32F4xx Initialization Support](#)
- [STM32F4xx ADC Support](#)
- [STM32F4xx CAN Support](#)
- [STM32F4xx EXT Support](#)
- [STM32F4xx GPT Support](#)
- [STM32F4xx ICU Support](#)
- [STM32F4xx MAC Support](#)
- [STM32F4xx PAL Support](#)
- [STM32F4xx PWM Support](#)
- [STM32F4xx SDC Support](#)
- [STM32F4xx Serial Support](#)
- [STM32F4xx SPI Support](#)
- [STM32F4xx UART Support](#)
- [STM32F4xx Platform Drivers](#)

6.20 STM32F4xx Initialization Support

The STM32F4xx HAL support is responsible for system initialization.

6.20.1 Supported HW resources

- PLL1.
- PLL2.
- RCC.
- Flash.

6.20.2 STM32F4xx 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.

6.21 STM32F4xx ADC Support

The STM32F4xx ADC driver supports the ADC peripherals using DMA channels for maximum performance.

6.21.1 Supported HW resources

- ADC1.
- ADC2.
- ADC3.
- DMA2.

6.21.2 STM32F4xx 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.

6.22 STM32F4xx CAN Support

The STM32F4xx CAN driver uses the CAN peripherals.

6.22.1 Supported HW resources

- bxCAN1.

6.22.2 STM32F4xx CAN driver implementation features

- Clock stop for reduced power usage when the driver is in stop state.
- Support for bxCAN sleep mode.
- Programmable bxCAN interrupts priority level.

6.23 STM32F4xx EXT Support

The STM32F4xx EXT driver uses the EXTI peripheral.

6.23.1 Supported HW resources

- EXTI.

6.23.2 STM32F4xx 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.

6.24 STM32F4xx GPT Support

The STM32F4xx GPT driver uses the TIMx peripherals.

6.24.1 Supported HW resources

- TIM1.
- TIM2.
- TIM3.
- TIM4.
- TIM5.
- TIM8.

6.24.2 STM32F4xx 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.

6.25 STM32F4xx ICU Support

The STM32F4xx ICU driver uses the TIMx peripherals.

6.25.1 Supported HW resources

- TIM1.
- TIM2.
- TIM3.
- TIM4.
- TIM5.
- TIM8.

6.25.2 STM32F4xx 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.

6.26 STM32F4xx MAC Support

The STM32F4xx MAC driver supports the ETH peripheral.

6.26.1 Supported HW resources

- ETH.
- PHY (external).

6.26.2 STM32F4xx MAC driver implementation features

- Dedicated DMA operations.
- Support for checksum off-loading.

6.27 STM32F4xx PAL Support

The STM32F4xx PAL driver uses the GPIO peripherals.

6.27.1 Supported HW resources

- GPIOA.
- GPIOB.
- GPIOC.
- GPIOD.
- GPIOE.
- GPIOF.
- GPIOG.
- GPIOH.
- GPIOI.

6.27.2 STM32F4xx 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.

6.27.3 Supported PAL setup modes

The STM32F4xx 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.

6.27.4 Suboptimal behavior

The STM32F4xx 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.

6.28 STM32F4xx PWM Support

The STM32F4xx PWM driver uses the TIMx peripherals.

6.28.1 Supported HW resources

- TIM1.
- TIM2.
- TIM3.
- TIM4.
- TIM5.
- TIM8.

6.28.2 STM32F4xx 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.

6.29 STM32F4xx SDC Support

The STM32F4xx SDC driver uses the SDIO peripheral.

6.29.1 Supported HW resources

- SDIO.
- DMA2.

6.29.2 STM32F4xx SDC driver implementation features

- Clock stop for reduced power usage when the driver is in stop state.
- Programmable interrupt priority.
- DMA is used for receiving and transmitting.
- Programmable DMA bus priority for each DMA channel.

6.30 STM32F4xx Serial Support

The STM32F4xx Serial driver uses the USART/UART peripherals in a buffered, interrupt driven, implementation.

6.30.1 Supported HW resources

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

- USART1.
- USART2.
- USART3.
- UART4.
- UART5.
- USART6.

6.30.2 STM32F4xx 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.

6.31 STM32F4xx SPI Support

The SPI driver supports the STM32F4xx SPI peripherals using DMA channels for maximum performance.

6.31.1 Supported HW resources

- SPI1.
- SPI2.
- SPI3.
- DMA1.
- DMA2.

6.31.2 STM32F4xx 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.

6.32 STM32F4xx UART Support

The UART driver supports the STM32F4xx USART peripherals using DMA channels for maximum performance.

6.32.1 Supported HW resources

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

- USART1.
- USART2.
- USART3.
- DMA1.
- DMA2.

6.32.2 STM32F4xx 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.

6.33 STM32F4xx Platform Drivers

6.33.1 Detailed Description

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

Modules

- [STM32F4xx DMA Support](#)
- [STM32F4xx RCC Support](#)

6.34 STM32F4xx DMA Support

6.34.1 Detailed Description

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

6.34.2 Supported HW resources

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

- DMA1.
- DMA2.

6.34.3 STM32F4xx 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_Stream0_IRQHandler)`
DMA1 stream 0 shared interrupt handler.
- `CH_IRQ_HANDLER (DMA1_Stream1_IRQHandler)`
DMA1 stream 1 shared interrupt handler.
- `CH_IRQ_HANDLER (DMA1_Stream2_IRQHandler)`
DMA1 stream 2 shared interrupt handler.
- `CH_IRQ_HANDLER (DMA1_Stream3_IRQHandler)`
DMA1 stream 3 shared interrupt handler.
- `CH_IRQ_HANDLER (DMA1_Stream4_IRQHandler)`
DMA1 stream 4 shared interrupt handler.
- `CH_IRQ_HANDLER (DMA1_Stream5_IRQHandler)`
DMA1 stream 5 shared interrupt handler.
- `CH_IRQ_HANDLER (DMA1_Stream6_IRQHandler)`
DMA1 stream 6 shared interrupt handler.
- `CH_IRQ_HANDLER (DMA1_Stream7_IRQHandler)`
DMA1 stream 7 shared interrupt handler.
- `CH_IRQ_HANDLER (DMA2_Stream0_IRQHandler)`
DMA2 stream 0 shared interrupt handler.
- `CH_IRQ_HANDLER (DMA2_Stream1_IRQHandler)`
DMA2 stream 1 shared interrupt handler.
- `CH_IRQ_HANDLER (DMA2_Stream2_IRQHandler)`
DMA2 stream 2 shared interrupt handler.
- `CH_IRQ_HANDLER (DMA2_Stream3_IRQHandler)`

- **CH_IRQ_HANDLER** (DMA2_Stream4_IRQHandler)
DMA2 stream 4 shared interrupt handler.
- **CH_IRQ_HANDLER** (DMA2_Stream5_IRQHandler)
DMA2 stream 5 shared interrupt handler.
- **CH_IRQ_HANDLER** (DMA2_Stream6_IRQHandler)
DMA2 stream 6 shared interrupt handler.
- **CH_IRQ_HANDLER** (DMA2_Stream7_IRQHandler)
DMA2 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) (&_stm32_dma_streams[id])
*Returns a pointer to a **stm32_dma_stream_t** structure.*
- #define **STM32_DMA1_STREAM0** STM32_DMA_STREAM(0)
- #define **STM32_DMA1_STREAM1** STM32_DMA_STREAM(1)
- #define **STM32_DMA1_STREAM2** STM32_DMA_STREAM(2)
- #define **STM32_DMA1_STREAM3** STM32_DMA_STREAM(3)
- #define **STM32_DMA1_STREAM4** STM32_DMA_STREAM(4)
- #define **STM32_DMA1_STREAM5** STM32_DMA_STREAM(5)
- #define **STM32_DMA1_STREAM6** STM32_DMA_STREAM(6)
- #define **STM32_DMA1_STREAM7** STM32_DMA_STREAM(7)
- #define **STM32_DMA2_STREAM0** STM32_DMA_STREAM(8)
- #define **STM32_DMA2_STREAM1** STM32_DMA_STREAM(9)
- #define **STM32_DMA2_STREAM2** STM32_DMA_STREAM(10)
- #define **STM32_DMA2_STREAM3** STM32_DMA_STREAM(11)
- #define **STM32_DMA2_STREAM4** STM32_DMA_STREAM(12)
- #define **STM32_DMA2_STREAM5** STM32_DMA_STREAM(13)
- #define **STM32_DMA2_STREAM6** STM32_DMA_STREAM(14)
- #define **STM32_DMA2_STREAM7** STM32_DMA_STREAM(15)

CR register constants common to all DMA types

- #define **STM32_DMA_CR_EN** DMA_SxCR_EN
- #define **STM32_DMA_CR_TEIE** DMA_SxCR_TEIE
- #define **STM32_DMA_CR_HTIE** DMA_SxCR_HTIE
- #define **STM32_DMA_CR_TCIE** DMA_SxCR_TCIE
- #define **STM32_DMA_CR_DIR_MASK** DMA_SxCR_DIR
- #define **STM32_DMA_CR_DIR_P2M** 0
- #define **STM32_DMA_CR_DIR_M2P** DMA_SxCR_DIR_0

- #define **STM32_DMA_CR_DIR_M2M** DMA_SxCR_DIR_1
- #define **STM32_DMA_CR_CIRC** DMA_SxCR_CIRC
- #define **STM32_DMA_CR_PINC** DMA_SxCR_PINC
- #define **STM32_DMA_CR_MINC** DMA_SxCR_MINC
- #define **STM32_DMA_CR_PSIZE_MASK** DMA_SxCR_PSIZE
- #define **STM32_DMA_CR_PSIZE_BYTE** 0
- #define **STM32_DMA_CR_PSIZE_HWORD** DMA_SxCR_PSIZE_0
- #define **STM32_DMA_CR_PSIZE_WORD** DMA_SxCR_PSIZE_1
- #define **STM32_DMA_CR_MSIZEMASK** DMA_SxCR_MSIZE
- #define **STM32_DMA_CR_MSIZEBYTE** 0
- #define **STM32_DMA_CR_MSIZEHWORD** DMA_SxCR_MSIZE_0
- #define **STM32_DMA_CR_MSIZEWORD** DMA_SxCR_MSIZE_1
- #define **STM32_DMA_CR_SIZE_MASK**
- #define **STM32_DMA_CR_PL_MASK** DMA_SxCR_PL
- #define **STM32_DMA_CR_PL(n)** ((n) << 16)

CR register constants only found in STM32F2xx/STM32F4xx

- #define **STM32_DMA_CR_DMEIE** DMA_SxCR_DMEIE
- #define **STM32_DMA_CR_PFCTRL** DMA_SxCR_PFCTRL
- #define **STM32_DMA_CR_PINCOS** DMA_SxCR_PINCOS
- #define **STM32_DMA_CR_DBM** DMA_SxCR_DBM
- #define **STM32_DMA_CR_CT** DMA_SxCR_CT
- #define **STM32_DMA_CR_PBURST_MASK** DMA_SxCR_PBURST
- #define **STM32_DMA_CR_PBURST_SINGLE** 0
- #define **STM32_DMA_CR_PBURST_INCR4** DMA_SxCR_PBURST_0
- #define **STM32_DMA_CR_PBURST_INCR8** DMA_SxCR_PBURST_1
- #define **STM32_DMA_CR_PBURST_INCR16** (DMA_SxCR_PBURST_0 | DMA_SxCR_PBURST_1)
- #define **STM32_DMA_CR_MBURST_MASK** DMA_SxCR_MBURST
- #define **STM32_DMA_CR_MBURST_SINGLE** 0
- #define **STM32_DMA_CR_MBURST_INCR4** DMA_SxCR_MBURST_0
- #define **STM32_DMA_CR_MBURST_INCR8** DMA_SxCR_MBURST_1
- #define **STM32_DMA_CR_MBURST_INCR16** (DMA_SxCR_MBURST_0 | DMA_SxCR_MBURST_1)
- #define **STM32_DMA_CR_CHSEL_MASK** DMA_SxCR_CHSEL
- #define **STM32_DMA_CR_CHSEL(n)** ((n) << 25)

FCR register constants only found in STM32F2xx/STM32F4xx

- #define **STM32_DMA_FCR_FEIE** DMA_SxFCR_FEIE
- #define **STM32_DMA_FCR_FS_MASK** DMA_SxFCR_FS
- #define **STM32_DMA_FCR_DMDIS** DMA_SxFCR_DMDIS
- #define **STM32_DMA_FCR_FTH_MASK** DMA_SxFCR_FTH
- #define **STM32_DMA_FCR_FTH_1Q** 0
- #define **STM32_DMA_FCR_FTH_HALF** DMA_SxFCR_FTH_0
- #define **STM32_DMA_FCR_FTH_3Q** DMA_SxFCR_FTH_1
- #define **STM32_DMA_FCR_FTH_FULL** (DMA_SxFCR_FTH_0 | DMA_SxFCR_FTH_1)

Status flags passed to the ISR callbacks

- #define **STM32_DMA_ISR_FEIF** DMA_LISR_FEIFO
- #define **STM32_DMA_ISR_DMEIF** DMA_LISR_DMEIF0
- #define **STM32_DMA_ISR_TEIF** DMA_LISR_TEIFO
- #define **STM32_DMA_ISR_HTIF** DMA_LISR_HTIFO
- #define **STM32_DMA_ISR_TCIF** DMA_LISR_TCIFO

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 dmaStreamSetMemory1(dmastp, addr)`
Associates an alternate memory destination to a DMA stream.
- `#define dmaStreamSetTransactionSize(dmastp, size)`
Sets the number of transfers to be performed.
- `#define dmaStreamGetTransactionSize(dmastp) ((size_t)((dmastp)->stream->NDTR))`
Returns the number of transfers to be performed.
- `#define dmaStreamSetMode(dmastp, mode)`
Programs the stream mode settings.
- `#define dmaStreamSetFIFO(dmastp, mode)`
Programs the stream FIFO settings.
- `#define dmaStreamEnable(dmastp)`
DMA stream enable.
- `#define dmaStreamDisable(dmastp)`
DMA stream disable.
- `#define dmaStreamClearInterrupt(dmastp)`
DMA stream interrupt sources clear.
- `#define dmaStartMemcpy(dmastp, mode, src, dst, n)`
Starts a memory to memory operation using the specified stream.
- `#define dmaWaitCompletion(dmastp)`
Polling wait for DMA transfer end.

Defines

- `#define STM32_DMA1_STREAMS_MASK 0x000000FF`
Mask of the DMA1 streams in dma_streams_mask.
- `#define STM32_DMA2_STREAMS_MASK 0x0000FF00`
Mask of the DMA2 streams in dma_streams_mask.
- `#define STM32_DMA_CR_RESET_VALUE 0x00000000`
Post-reset value of the stream CR register.
- `#define STM32_DMA_FCR_RESET_VALUE 0x00000021`
Post-reset value of the stream FCR register.
- `#define STM32_DMA_STREAMS 16`
Total number of DMA streams.
- `#define STM32_DMA_ISR_MASK 0x3D`
Mask of the ISR bits passed to the DMA callback functions.
- `#define STM32_DMA_GETCHANNEL(id, c) (((c) >> (((id) & 7) * 4)) & 7)`
Returns the channel associated to the specified stream.
- `#define STM32_DMA_STREAM_ID(dma, stream) (((dma) - 1) * 8) + (stream))`
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.

Typedefs

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

6.34.4 Function Documentation

6.34.4.1 CH_IRQ_HANDLER (DMA1_Stream0_IRQHandler)

DMA1 stream 0 shared interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.34.4.2 CH_IRQ_HANDLER (DMA1_Stream1_IRQHandler)

DMA1 stream 1 shared interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.34.4.3 CH_IRQ_HANDLER (DMA1_Stream2_IRQHandler)

DMA1 stream 2 shared interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.34.4.4 CH_IRQ_HANDLER (DMA1_Stream3_IRQHandler)

DMA1 stream 3 shared interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.34.4.5 CH_IRQ_HANDLER (DMA1_Stream4_IRQHandler)

DMA1 stream 4 shared interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.34.4.6 CH_IRQ_HANDLER (DMA1_Stream5_IRQHandler)

DMA1 stream 5 shared interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.34.4.7 CH_IRQ_HANDLER (DMA1_Stream6_IRQHandler)

DMA1 stream 6 shared interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.34.4.8 CH_IRQ_HANDLER (DMA1_Stream7_IRQHandler)

DMA1 stream 7 shared interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.34.4.9 CH_IRQ_HANDLER (DMA2_Stream0_IRQHandler)

DMA2 stream 0 shared interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.34.4.10 CH_IRQ_HANDLER (DMA2_Stream1_IRQHandler)

DMA2 stream 1 shared interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.34.4.11 CH_IRQ_HANDLER (DMA2_Stream2_IRQHandler)

DMA2 stream 2 shared interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.34.4.12 CH_IRQ_HANDLER (DMA2_Stream3_IRQHandler)

DMA2 stream 3 shared interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.34.4.13 CH_IRQ_HANDLER (DMA2_Stream4_IRQHandler)

DMA2 stream 4 shared interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.34.4.14 CH_IRQ_HANDLER (DMA2_Stream5_IRQHandler)

DMA2 stream 5 shared interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.34.4.15 CH_IRQ_HANDLER (DMA2_Stream6_IRQHandler)

DMA2 stream 6 shared interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.34.4.16 CH_IRQ_HANDLER (DMA2_Stream7_IRQHandler)

DMA2 stream 7 shared interrupt handler.

Function Class:

Interrupt handler, this function should not be directly invoked.

6.34.4.17 void dmalnit (void)

STM32 DMA helper initialization.

Function Class:

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

6.34.4.18 bool_t dmaStreamAllocate (const stm32_dma_stream_t * dmastp, uint32_t priority, stm32_dmaisr_t func, void * param)

Allocates a DMA stream.

The stream is allocated and, if required, the DMA clock enabled. The function also enables the IRQ vector associated to the stream and initializes its priority.

Precondition

The stream must not be already in use or an error is returned.

Postcondition

The stream is allocated and the default ISR handler redirected to the specified function.

The stream ISR vector is enabled and its priority configured.

The stream must be freed using [dmaStreamRelease \(\)](#) before it can be reused with another peripheral.

The stream is in its post-reset state.

Note

This function can be invoked in both ISR or thread context.

Parameters

in	<i>dmastp</i>	pointer to a <code>stm32_dma_stream_t</code> 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

FALSE no error, stream taken.

TRUE error, stream already taken.

Function Class:

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

6.34.4.19 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	<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.

6.34.5 Variable Documentation**6.34.5.1 const `stm32_dma_stream_t` _`stm32_dma_streams`[STM32_DMA_STREAMS]****Initial value:**

```
{
{DMA1_Stream0, &DMA1->LIFCR, 0, 0, DMA1_Stream0_IRQn},
{DMA1_Stream1, &DMA1->LIFCR, 6, 1, DMA1_Stream1_IRQn},
{DMA1_Stream2, &DMA1->LIFCR, 16, 2, DMA1_Stream2_IRQn},
{DMA1_Stream3, &DMA1->LIFCR, 22, 3, DMA1_Stream3_IRQn},
{DMA1_Stream4, &DMA1->HIFCR, 0, 4, DMA1_Stream4_IRQn},
{DMA1_Stream5, &DMA1->HIFCR, 6, 5, DMA1_Stream5_IRQn},
{DMA1_Stream6, &DMA1->HIFCR, 16, 6, DMA1_Stream6_IRQn},
{DMA1_Stream7, &DMA1->HIFCR, 22, 7, DMA1_Stream7_IRQn},
{DMA2_Stream0, &DMA2->LIFCR, 0, 8, DMA2_Stream0_IRQn},
{DMA2_Stream1, &DMA2->LIFCR, 6, 9, DMA2_Stream1_IRQn},
```

```
{DMA2_Stream2, &DMA2->LIFCR, 16, 10, DMA2_Stream2_IRQn},  
{DMA2_Stream3, &DMA2->LIFCR, 22, 11, DMA2_Stream3_IRQn},  
{DMA2_Stream4, &DMA2->HIFCR, 0, 12, DMA2_Stream4_IRQn},  
{DMA2_Stream5, &DMA2->HIFCR, 6, 13, DMA2_Stream5_IRQn},  
{DMA2_Stream6, &DMA2->HIFCR, 16, 14, DMA2_Stream6_IRQn},  
{DMA2_Stream7, &DMA2->HIFCR, 22, 15, DMA2_Stream7_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_STREAM0, STM32_DMA1_STREAM1 etc.

6.34.6 Define Documentation

6.34.6.1 #define STM32_DMA1_STREAMS_MASK 0x000000FF

Mask of the DMA1 streams in `dma_streams_mask`.

6.34.6.2 #define STM32_DMA2_STREAMS_MASK 0x0000FF00

Mask of the DMA2 streams in `dma_streams_mask`.

6.34.6.3 #define STM32_DMA_CR_RESET_VALUE 0x00000000

Post-reset value of the stream CR register.

6.34.6.4 #define STM32_DMA_FCR_RESET_VALUE 0x00000021

Post-reset value of the stream FCR register.

6.34.6.5 #define STM32_DMA_STREAMS 16

Total number of DMA streams.

Note

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

6.34.6.6 #define STM32_DMA_ISR_MASK 0x3D

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

6.34.6.7 #define STM32_DMA_GETCHANNEL(id, c) (((c) >> (((id) & 7) * 4)) & 7)

Returns the channel associated to the specified stream.

Parameters

in	<i>id</i>	the unique numeric stream identifier
in	<i>c</i>	a stream/channel association word, one channel per nibble

Returns

Returns the channel associated to the stream.

6.34.6.8 #define STM32_DMA_STREAM_ID(*dma*, *stream*) (((*dma*) - 1) * 8) + (*stream*)

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.

6.34.6.9 #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.

6.34.6.10 #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.

6.34.6.11 #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.

6.34.6.12 #define dmaStreamSetPeripheral(*dmastp*, *addr*)

Value:

```
{
    (dmastp)->stream->PAR  = (uint32_t) (addr);
}
```

Associates a peripheral data register to a DMA stream.

Note

This function can be invoked in both ISR or thread context.

Precondition

The stream must have been allocated using [dmaStreamAllocate\(\)](#).

Postcondition

After use the stream can be released using [dmaStreamRelease\(\)](#).

Parameters

in	<i>dmastp</i>	pointer to a stm32_dma_stream_t structure
in	<i>addr</i>	value to be written in the PAR register

Function Class:

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

6.34.6.13 #define dmaStreamSetMemory0(*dmastp*, *addr*)

Value:

```
{
    (dmastp)->stream->M0AR  = (uint32_t) (addr);
}
```

Associates a memory destination to a DMA stream.

Note

This function can be invoked in both ISR or thread context.

Precondition

The stream must have been allocated using [dmaStreamAllocate\(\)](#).

Postcondition

After use the stream can be released using [dmaStreamRelease\(\)](#).

Parameters

in	<i>dmastp</i>	pointer to a stm32_dma_stream_t structure
in	<i>addr</i>	value to be written in the M0AR register

Function Class:

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

6.34.6.14 #define dmaStreamSetMemory1(*dmastp*, *addr*)**Value:**

```
{
    (dmastp)->stream->M1AR  = (uint32_t) (addr);
}
```

Associates an alternate memory destination to a DMA stream.

Note

This function can be invoked in both ISR or thread context.

Parameters

in	<i>dmastp</i>	pointer to a stm32_dma_stream_t structure
in	<i>addr</i>	value to be written in the M1AR register

Function Class:

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

6.34.6.15 #define dmaStreamSetTransactionSize(*dmastp*, *size*)**Value:**

```
{
    (dmastp)->stream->NDTR  = (uint32_t) (size);
}
```

Sets the number of transfers to be performed.

Note

This function can be invoked in both ISR or thread context.

Precondition

The stream must have been allocated using [dmaStreamAllocate\(\)](#).

Postcondition

After use the stream can be released using [dmaStreamRelease\(\)](#).

Parameters

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

Function Class:

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

6.34.6.16 #define dmaStreamGetTransactionSize(*dmastp*) ((size_t)((dmastp)->stream->NDTR))

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.

6.34.6.17 #define dmaStreamSetMode(*dmastp*, *mode*)**Value:**

```
{
    (dmastp)->stream->CR  = (uint32_t) (mode);
}
```

Programs the stream mode settings.

Note

This function can be invoked in both ISR or thread context.

Precondition

The stream must have been allocated using `dmaStreamAllocate()`.

Postcondition

After use the stream can be released using `dmaStreamRelease()`.

Parameters

in	<code>dmastp</code> pointer to a <code>stm32_dma_stream_t</code> structure
in	<code>mode</code> value to be written in the CR register

Function Class:

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

6.34.6.18 #define dmaStreamSetFIFO(*dmastp*, *mode*)**Value:**

```
{
    (dmastp)->stream->FCR = (uint32_t) (mode);
}
```

Programs the stream FIFO 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 FCR register

Function Class:

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

6.34.6.19 #define `dmaStreamEnable(dmastp)`

Value:

```
{
  (dmastp)->stream->CR |= 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.

6.34.6.20 #define `dmaStreamDisable(dmastp)`

Value:

```
{
  (dmastp)->stream->CR &= ~STM32_DMA_CR_EN;
  while (((dmastp)->stream->CR & STM32_DMA_CR_EN) != 0)
    ;
  dmaStreamClearInterrupt(dmastp);
}
```

DMA stream disable.

The function disables the specified stream, waits for the disable operation to complete 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.

6.34.6.21 #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.

6.34.6.22 #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

6.34.6.23 `#define dmaWaitCompletion(dmastp)`**Value:**

```
{
    while ((dmastp)->stream->NDTR > 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
----	---------------------	--

6.34.7 Typedef Documentation

6.34.7.1 `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 xISR register, the bits are aligned to bit zero

6.35 STM32F4xx RCC Support

6.35.1 Detailed Description

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

6.35.2 Supported HW resources

- RCC.

6.35.3 STM32F4xx 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 rccEnableAHB1(mask, lp)`
Enables the clock of one or more peripheral on the AHB1 bus.
- `#define rccDisableAHB1(mask, lp)`
Disables the clock of one or more peripheral on the AHB1 bus.
- `#define rccResetAHB1(mask)`
Resets one or more peripheral on the AHB1 bus.
- `#define rccEnableAHB2(mask, lp)`
Enables the clock of one or more peripheral on the AHB2 bus.
- `#define rccDisableAHB2(mask, lp)`
Disables the clock of one or more peripheral on the AHB2 bus.
- `#define rccResetAHB2(mask)`
Resets one or more peripheral on the AHB2 bus.
- `#define rccEnableAHB3(mask, lp)`
Enables the clock of one or more peripheral on the AHB3 (FSMC) bus.
- `#define rccDisableAHB3(mask, lp)`
Disables the clock of one or more peripheral on the AHB3 (FSMC) bus.
- `#define rccResetAHB3(mask)`
Resets one or more peripheral on the AHB3 (FSMC) 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.
- `#define rccEnableADC2(lp) rccEnableAPB2(RCC_APB2ENR_ADC2EN, lp)`
Enables the ADC2 peripheral clock.
- `#define rccDisableADC2(lp) rccDisableAPB2(RCC_APB2ENR_ADC2EN, lp)`
Disables the ADC2 peripheral clock.
- `#define rccResetADC2() rccResetAPB2(RCC_APB2RSTR_ADC2RST)`
Resets the ADC2 peripheral.
- `#define rccEnableADC3(lp) rccEnableAPB2(RCC_APB2ENR_ADC3EN, lp)`
Enables the ADC3 peripheral clock.
- `#define rccDisableADC3(lp) rccDisableAPB2(RCC_APB2ENR_ADC3EN, lp)`
Disables the ADC3 peripheral clock.
- `#define rccResetADC3() rccResetAPB2(RCC_APB2RSTR_ADC3RST)`
Resets the ADC3 peripheral.

DMA peripheral specific RCC operations

- `#define rccEnableDMA1(lp) rccEnableAHB1(RCC_AHB1ENR_DMA1EN, lp)`
Enables the DMA1 peripheral clock.
- `#define rccDisableDMA1(lp) rccDisableAHB1(RCC_AHB1ENR_DMA1EN, lp)`
Disables the DMA1 peripheral clock.
- `#define rccResetDMA1() rccResetAHB1(RCC_AHB1RSTR_DMA1RST)`
Resets the DMA1 peripheral.
- `#define rccEnableDMA2(lp) rccEnableAHB1(RCC_AHB1ENR_DMA2EN, lp)`
Enables the DMA2 peripheral clock.
- `#define rccDisableDMA2(lp) rccDisableAHB1(RCC_AHB1ENR_DMA2EN, lp)`
Disables the DMA2 peripheral clock.
- `#define rccResetDMA2() rccResetAHB1(RCC_AHB1RSTR_DMA2RST)`
Resets the DMA2 peripheral.

PWR interface specific RCC operations

- `#define rccEnablePWRIface(lp) rccEnableAPB1(RCC_APB1ENR_PWREN, lp)`
Enables the PWR interface clock.
- `#define rccDisablePWRIface(lp) rccDisableAPB1(RCC_APB1ENR_PWREN, lp)`
Disables PWR interface clock.
- `#define rccResetPWRIface() rccResetAPB1(RCC_APB1RSTR_PWRRST)`
Resets the PWR interface.

CAN peripherals specific RCC operations

- #define `rccEnableCAN1(lp)` rccEnableAPB1(RCC_APB1ENR_CAN1EN, lp)
Enables the CAN1 peripheral clock.
- #define `rccDisableCAN1(lp)` rccDisableAPB1(RCC_APB1ENR_CAN1EN, lp)
Disables the CAN1 peripheral clock.
- #define `rccResetCAN1()` rccResetAPB1(RCC_APB1RSTR_CAN1RST)
Resets the CAN1 peripheral.
- #define `rccEnableCAN2(lp)` rccEnableAPB1(RCC_APB1ENR_CAN2EN, lp)
Enables the CAN2 peripheral clock.
- #define `rccDisableCAN2(lp)` rccDisableAPB1(RCC_APB1ENR_CAN2EN, lp)
Disables the CAN2 peripheral clock.
- #define `rccResetCAN2()` rccResetAPB1(RCC_APB1RSTR_CAN2RST)
Resets the CAN2 peripheral.

ETH peripheral specific RCC operations

- #define `rccEnableETH(lp)`
Enables the ETH peripheral clock.
- #define `rccDisableETH(lp)`
Disables the ETH peripheral clock.
- #define `rccResetETH()` rccResetAHB1(RCC_AHB1RSTR_ETHMACRST)
Resets the ETH peripheral.

I2C peripherals specific RCC operations

- #define `rccEnableI2C1(lp)` rccEnableAPB1(RCC_APB1ENR_I2C1EN, lp)
Enables the I2C1 peripheral clock.
- #define `rccDisableI2C1(lp)` rccDisableAPB1(RCC_APB1ENR_I2C1EN, lp)
Disables the I2C1 peripheral clock.
- #define `rccResetI2C1()` rccResetAPB1(RCC_APB1RSTR_I2C1RST)
Resets the I2C1 peripheral.
- #define `rccEnableI2C2(lp)` rccEnableAPB1(RCC_APB1ENR_I2C2EN, lp)
Enables the I2C2 peripheral clock.
- #define `rccDisableI2C2(lp)` rccDisableAPB1(RCC_APB1ENR_I2C2EN, lp)
Disables the I2C2 peripheral clock.
- #define `rccResetI2C2()` rccResetAPB1(RCC_APB1RSTR_I2C2RST)
Resets the I2C2 peripheral.
- #define `rccEnableI2C3(lp)` rccEnableAPB1(RCC_APB1ENR_I2C3EN, lp)
Enables the I2C3 peripheral clock.
- #define `rccDisableI2C3(lp)` rccDisableAPB1(RCC_APB1ENR_I2C3EN, lp)
Disables the I2C3 peripheral clock.
- #define `rccResetI2C3()` rccResetAPB1(RCC_APB1RSTR_I2C3RST)
Resets the I2C3 peripheral.

OTG peripherals specific RCC operations

- #define `rccEnableOTG_FS(lp)` rccEnableAHB2(RCC_AHB2LPENR_OTGFSLPEN, lp)
Enables the OTG_FS peripheral clock.
- #define `rccDisableOTG_FS(lp)` rccDisableAHB2(RCC_AHB2LPENR_OTGFSLPEN, lp)
Disables the OTG_FS peripheral clock.
- #define `rccResetOTG_FS()` rccResetAHB2(RCC_AHB2RSTR_OTGFSRST)
Resets the OTG_FS peripheral.

SDIO peripheral specific RCC operations

- `#define rccEnableSDIO(lp) rccEnableAPB2(RCC_APB2ENR_SDIOEN, lp)`
Enables the SDIO peripheral clock.
- `#define rccDisableSDIO(lp) rccDisableAPB2(RCC_APB2ENR_SDIOEN, lp)`
Disables the SDIO peripheral clock.
- `#define rccResetSDIO() rccResetAPB2(RCC_APB2RSTR_SDIORST)`
Resets the SDIO peripheral.

SPI peripherals specific RCC operations

- `#define rccEnableSPI1(lp) rccEnableAPB2(RCC_APB2ENR_SPI1EN, lp)`
Enables the SPI1 peripheral clock.
- `#define rccDisableSPI1(lp) rccDisableAPB2(RCC_APB2ENR_SPI1EN, lp)`
Disables the SPI1 peripheral clock.
- `#define rccResetSPI1() rccResetAPB2(RCC_APB2RSTR_SPI1RST)`
Resets the SPI1 peripheral.
- `#define rccEnableSPI2(lp) rccEnableAPB1(RCC_APB1ENR_SPI2EN, lp)`
Enables the SPI2 peripheral clock.
- `#define rccDisableSPI2(lp) rccDisableAPB1(RCC_APB1ENR_SPI2EN, lp)`
Disables the SPI2 peripheral clock.
- `#define rccResetSPI2() rccResetAPB1(RCC_APB1RSTR_SPI2RST)`
Resets the SPI2 peripheral.
- `#define rccEnableSPI3(lp) rccEnableAPB1(RCC_APB1ENR_SPI3EN, lp)`
Enables the SPI3 peripheral clock.
- `#define rccDisableSPI3(lp) rccDisableAPB1(RCC_APB1ENR_SPI3EN, lp)`
Disables the SPI3 peripheral clock.
- `#define rccResetSPI3() rccResetAPB1(RCC_APB1RSTR_SPI3RST)`
Resets the SPI3 peripheral.

TIM peripherals specific RCC operations

- `#define rccEnableTIM1(lp) rccEnableAPB2(RCC_APB2ENR_TIM1EN, lp)`
Enables the TIM1 peripheral clock.
- `#define rccDisableTIM1(lp) rccDisableAPB2(RCC_APB2ENR_TIM1EN, lp)`
Disables the TIM1 peripheral clock.
- `#define rccResetTIM1() rccResetAPB2(RCC_APB2RSTR_TIM1RST)`
Resets the TIM1 peripheral.
- `#define rccEnableTIM2(lp) rccEnableAPB1(RCC_APB1ENR_TIM2EN, lp)`
Enables the TIM2 peripheral clock.
- `#define rccDisableTIM2(lp) rccDisableAPB1(RCC_APB1ENR_TIM2EN, lp)`
Disables the TIM2 peripheral clock.
- `#define rccResetTIM2() rccResetAPB1(RCC_APB1RSTR_TIM2RST)`
Resets the TIM2 peripheral.
- `#define rccEnableTIM3(lp) rccEnableAPB1(RCC_APB1ENR_TIM3EN, lp)`
Enables the TIM3 peripheral clock.
- `#define rccDisableTIM3(lp) rccDisableAPB1(RCC_APB1ENR_TIM3EN, lp)`
Disables the TIM3 peripheral clock.
- `#define rccResetTIM3() rccResetAPB1(RCC_APB1RSTR_TIM3RST)`
Resets the TIM3 peripheral.
- `#define rccEnableTIM4(lp) rccEnableAPB1(RCC_APB1ENR_TIM4EN, lp)`

- `#define rccDisableTIM4(lp) rccDisableAPB1(RCC_APB1ENR_TIM4EN, lp)`
Disables the TIM4 peripheral clock.
- `#define rccResetTIM4() rccResetAPB1(RCC_APB1RSTR_TIM4RST)`
Resets the TIM4 peripheral.
- `#define rccEnableTIM5(lp) rccEnableAPB1(RCC_APB1ENR_TIM5EN, lp)`
Enables the TIM5 peripheral clock.
- `#define rccDisableTIM5(lp) rccDisableAPB1(RCC_APB1ENR_TIM5EN, lp)`
Disables the TIM5 peripheral clock.
- `#define rccResetTIM5() rccResetAPB1(RCC_APB1RSTR_TIM5RST)`
Resets the TIM5 peripheral.
- `#define rccEnableTIM8(lp) rccEnableAPB2(RCC_APB2ENR_TIM8EN, lp)`
Enables the TIM8 peripheral clock.
- `#define rccDisableTIM8(lp) rccDisableAPB2(RCC_APB2ENR_TIM8EN, lp)`
Disables the TIM8 peripheral clock.
- `#define rccResetTIM8() rccResetAPB2(RCC_APB2RSTR_TIM8RST)`
Resets the TIM8 peripheral.

USART/UART peripherals specific RCC operations

- `#define rccEnableUSART1(lp) rccEnableAPB2(RCC_APB2ENR_USART1EN, lp)`
Enables the USART1 peripheral clock.
- `#define rccDisableUSART1(lp) rccDisableAPB2(RCC_APB2ENR_USART1EN, lp)`
Disables the USART1 peripheral clock.
- `#define rccResetUSART1() rccResetAPB2(RCC_APB2RSTR_USART1RST)`
Resets the USART1 peripheral.
- `#define rccEnableUSART2(lp) rccEnableAPB1(RCC_APB1ENR_USART2EN, lp)`
Enables the USART2 peripheral clock.
- `#define rccDisableUSART2(lp) rccDisableAPB1(RCC_APB1ENR_USART2EN, lp)`
Disables the USART2 peripheral clock.
- `#define rccResetUSART2() rccResetAPB1(RCC_APB1RSTR_USART2RST)`
Resets the USART2 peripheral.
- `#define rccEnableUSART3(lp) rccEnableAPB1(RCC_APB1ENR_USART3EN, lp)`
Enables the USART3 peripheral clock.
- `#define rccDisableUSART3(lp) rccDisableAPB1(RCC_APB1ENR_USART3EN, lp)`
Disables the USART3 peripheral clock.
- `#define rccResetUSART3() rccResetAPB1(RCC_APB1RSTR_USART3RST)`
Resets the USART3 peripheral.
- `#define rccEnableUSART6(lp) rccEnableAPB2(RCC_APB2ENR_USART6EN, lp)`
Enables the USART6 peripheral clock.
- `#define rccDisableUSART6(lp) rccDisableAPB2(RCC_APB2ENR_USART6EN, lp)`
Disables the USART6 peripheral clock.
- `#define rccEnableUART4(lp) rccEnableAPB1(RCC_APB1ENR_UART4EN, lp)`
Enables the UART4 peripheral clock.
- `#define rccDisableUART4(lp) rccDisableAPB1(RCC_APB1ENR_UART4EN, lp)`
Disables the UART4 peripheral clock.
- `#define rccResetUART4() rccResetAPB1(RCC_APB1RSTR_UART4RST)`
Resets the UART4 peripheral.
- `#define rccEnableUART5(lp) rccEnableAPB1(RCC_APB1ENR_UART5EN, lp)`
Enables the UART5 peripheral clock.
- `#define rccDisableUART5(lp) rccDisableAPB1(RCC_APB1ENR_UART5EN, lp)`
Disables the UART5 peripheral clock.

- `#define rccResetUART5() rccResetAPB1(RCC_APB1RSTR_UART5RST)`
Disables the UART5 peripheral clock.
- `#define rccResetUSART6() rccResetAPB2(RCC_APB2RSTR_USART6RST)`
Resets the USART6 peripheral.

6.35.4 Define Documentation

6.35.4.1 #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.

6.35.4.2 #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.

6.35.4.3 #define rccResetAPB1(*mask*)

Value:

```
{
    RCC->APB1RSTR |= (mask);
    RCC->APB1RSTR = 0;
}
```

Resets one or more peripheral on the APB1 bus.

Parameters

in *mask* APB1 peripherals mask

Function Class:

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

6.35.4.4 #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 *mask* APB2 peripherals mask

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.

6.35.4.5 #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 *mask* APB2 peripherals mask

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.

6.35.4.6 #define rccResetAPB2(*mask*)**Value:**

```
{
    RCC->APB2RSTR |= (mask);
    RCC->APB2RSTR = 0;
}
```

Resets one or more peripheral on the APB2 bus.

Parameters

in *mask* APB2 peripherals mask

Function Class:

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

6.35.4.7 #define rccEnableAHB1(*mask*, *lp*)**Value:**

```
{
    RCC->AHB1ENR |= (mask);
    if (lp)
        RCC->AHB1LPENR |= (mask);
}
```

Enables the clock of one or more peripheral on the AHB1 bus.

Parameters

in *mask* AHB1 peripherals mask

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.

6.35.4.8 #define rccDisableAHB1(*mask*, *lp*)**Value:**

```
{
    RCC->AHB1ENR &= ~ (mask);
    if (lp)
        RCC->AHB1LPENR &= ~ (mask);
}
```

Disables the clock of one or more peripheral on the AHB1 bus.

Parameters

in *mask* AHB1 peripherals mask

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.

6.35.4.9 #define rccResetAHB1(*mask*)**Value:**

```
{
    RCC->AHB1RSTR |= (mask);
    RCC->AHB1RSTR = 0;
}
```

Resets one or more peripheral on the AHB1 bus.

Parameters

in *mask* AHB1 peripherals mask

Function Class:

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

6.35.4.10 #define rccEnableAHB2(*mask*, *lp*)**Value:**

```
{
    RCC->AHB2ENR |= (mask);
    if (lp)
        RCC->AHB2LPENR |= (mask);
}
```

Enables the clock of one or more peripheral on the AHB2 bus.

Parameters

in *mask* AHB2 peripherals mask

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.

6.35.4.11 #define rccDisableAHB2(*mask*, *lp*)**Value:**

```
{
    RCC->AHB2ENR &= ~ (mask);
    if (lp)
        RCC->AHB2LPENR &= ~ (mask);
}
```

Disables the clock of one or more peripheral on the AHB2 bus.

Parameters

in *mask* AHB2 peripherals mask

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.

6.35.4.12 #define rccResetAHB2(*mask*)**Value:**

```
{
    RCC->AHB2RSTR |= (mask);
    RCC->AHB2RSTR = 0;
}
```

Resets one or more peripheral on the AHB2 bus.

Parameters

in *mask* AHB2 peripherals mask

Function Class:

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

6.35.4.13 #define rccEnableAHB3(*mask*, *lp*)**Value:**

```
{
    RCC->AHB3ENR |= (mask);
    if (lp)
        RCC->AHB3LPENR |= (mask);
}
```

Enables the clock of one or more peripheral on the AHB3 (FSMC) bus.

Parameters

in *mask* AHB3 peripherals mask

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.

6.35.4.14 #define rccDisableAHB3(*mask*, *lp*)**Value:**

```
{
    RCC->AHB3ENR &= ~ (mask);
    if (lp)
        RCC->AHB3LPENR &= ~ (mask);
}
```

Disables the clock of one or more peripheral on the AHB3 (FSMC) bus.

Parameters

in *mask* AHB3 peripherals mask

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.

6.35.4.15 #define rccResetAHB3(*mask*)**Value:**

```
{
    RCC->AHB3RSTR |= (mask);
    RCC->AHB3RSTR = 0;
}
```

Resets one or more peripheral on the AHB3 (FSMC) bus.

Parameters

in *mask* AHB3 peripherals mask

Function Class:

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

6.35.4.16 #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.

6.35.4.17 #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.

6.35.4.18 #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.

6.35.4.19 #define rccEnableADC2(*lp*) rccEnableAPB2(RCC_APB2ENR_ADC2EN, *lp*)

Enables the ADC2 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.

6.35.4.20 #define rccDisableADC2(*lp*) rccDisableAPB2(RCC_APB2ENR_ADC2EN, *lp*)

Disables the ADC2 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.

6.35.4.21 #define rccResetADC2() rccResetAPB2(RCC_APB2RSTR_ADC2RST)

Resets the ADC2 peripheral.

Function Class:

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

6.35.4.22 #define rccEnableADC3(*lp*) rccEnableAPB2(RCC_APB2ENR_ADC3EN, *lp*)

Enables the ADC3 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.

6.35.4.23 #define rccDisableADC3(*lp*) rccDisableAPB2(RCC_APB2ENR_ADC3EN, *lp*)

Disables the ADC3 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.

6.35.4.24 #define rccResetADC3() rccResetAPB2(RCC_APB2RSTR_ADC3RST)

Resets the ADC3 peripheral.

Function Class:

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

6.35.4.25 #define rccEnableDMA1(*lp*) rccEnableAHB1(RCC_AHB1ENR_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.

6.35.4.26 #define rccDisableDMA1(*lp*) rccDisableAHB1(RCC_AHB1ENR_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.

6.35.4.27 #define rccResetDMA1() rccResetAHB1(RCC_AHB1RSTR_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.

6.35.4.28 #define rccEnableDMA2(*lp*) rccEnableAHB1(RCC_AHB1ENR_DMA2EN, *lp*)

Enables the DMA2 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.

6.35.4.29 #define rccDisableDMA2(*lp*) rccDisableAHB1(RCC_AHB1ENR_DMA2EN, *lp*)

Disables the DMA2 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.

6.35.4.30 #define rccResetDMA2() rccResetAHB1(RCC_AHB1RSTR_DMA2RST)

Resets the DMA2 peripheral.

Function Class:

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

6.35.4.31 #define rccEnablePWRInterface(*lp*) rccEnableAPB1(RCC_APB1ENR_PWREN, *lp*)

Enables the PWR interface 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.

6.35.4.32 #define rccDisablePWRInterface(*lp*) rccDisableAPB1(RCC_APB1ENR_PWREN, *lp*)

Disables PWR interface 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.

6.35.4.33 #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.

6.35.4.34 #define rccEnableCAN1(*lp*) rccEnableAPB1(RCC_APB1ENR_CAN1EN, *lp*)

Enables the CAN1 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.

6.35.4.35 #define rccDisableCAN1(*lp*) rccDisableAPB1(RCC_APB1ENR_CAN1EN, *lp*)

Disables the CAN1 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.

6.35.4.36 #define rccResetCAN1() rccResetAPB1(RCC_APB1RSTR_CAN1RST)

Resets the CAN1 peripheral.

Function Class:

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

6.35.4.37 #define rccEnableCAN2(*lp*) rccEnableAPB1(RCC_APB1ENR_CAN2EN, *lp*)

Enables the CAN2 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.

6.35.4.38 #define rccDisableCAN2(*lp*) rccDisableAPB1(RCC_APB1ENR_CAN2EN, *lp*)

Disables the CAN2 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.

6.35.4.39 #define rccResetCAN2() rccResetAPB1(RCC_APB1RSTR_CAN2RST)

Resets the CAN2 peripheral.

Function Class:

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

6.35.4.40 #define rccEnableETH(*lp*)

Value:

```
rccEnableAHB1 (RCC_AHB1ENR_ETHMACEN | \  
RCC_AHB1ENR_ETHMACTXEN | \  
RCC_AHB1ENR_ETHMACRXEN, lp)
```

Enables the ETH peripheral clock.

Note

The *lp* parameter is ignored in this family.

Parameters

in *lp* low power enable flag

Function Class:

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

6.35.4.41 #define rccDisableETH(*lp*)**Value:**

```
rccDisableAHB1(RCC_AHB1ENR_ETHMACEN |           \
                  RCC_AHB1ENR_ETHMACTXEN |           \
                  RCC_AHB1ENR_ETHMACRXEN, lp)
```

Disables the ETH peripheral clock.

Note

The *lp* parameter is ignored in this family.

Parameters

in	<i>lp</i> low power enable flag
----	---------------------------------

Function Class:

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

6.35.4.42 #define rccResetETH() rccResetAHB1(RCC_AHB1RSTR_ETHMACRST)

Resets the ETH peripheral.

Function Class:

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

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

Enables the I2C1 peripheral clock.

Parameters

in	<i>lp</i> low power enable flag
----	---------------------------------

Function Class:

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

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

Disables the I2C1 peripheral clock.

Parameters

in	<i>lp</i> low power enable flag
----	---------------------------------

Function Class:

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

6.35.4.45 #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.

6.35.4.46 #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.

6.35.4.47 #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.

6.35.4.48 #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.

6.35.4.49 #define rccEnableI2C3(*lp*) rccEnableAPB1(RCC_APB1ENR_I2C3EN, *lp*)

Enables the I2C3 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.

6.35.4.50 #define rccDisableI2C3(*lp*) rccDisableAPB1(RCC_APB1ENR_I2C3EN, *lp*)

Disables the I2C3 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.

6.35.4.51 #define rccResetI2C3() rccResetAPB1(RCC_APB1RSTR_I2C3RST)

Resets the I2C3 peripheral.

Function Class:

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

6.35.4.52 #define rccEnableOTG_FS(*lp*) rccEnableAHB2(RCC_AHB2LPENR_OTGFSLPEN, *lp*)

Enables the OTG_FS 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.

6.35.4.53 #define rccDisableOTG_FS(*lp*) rccEnableAHB2(RCC_AHB2LPENR_OTGFSLPEN, *lp*)

Disables the OTG_FS 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.

6.35.4.54 #define rccResetOTG_FS() rccResetAHB2(RCC_AHB2RSTR_OTGFSRST)

Resets the OTG_FS peripheral.

Function Class:

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

6.35.4.55 #define rccEnableSDIO(*lp*) rccEnableAPB2(RCC_APB2ENR_SDIOEN, *lp*)

Enables the SDIO peripheral clock.

Note

The *lp* parameter is ignored in this family.

Parameters

in *lp* low power enable flag

Function Class:

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

6.35.4.56 #define rccDisableSDIO(*lp*) rccDisableAPB2(RCC_APB2ENR_SDIOEN, *lp*)

Disables the SDIO peripheral clock.

Note

The *lp* parameter is ignored in this family.

Parameters

in *lp* low power enable flag

Function Class:

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

6.35.4.57 #define rccResetSDIO() rccResetAPB2(RCC_APB2RSTR_SDIORST)

Resets the SDIO peripheral.

Note

Not supported in this family, does nothing.

Function Class:

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

6.35.4.58 #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.

6.35.4.59 #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.

6.35.4.60 #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.

6.35.4.61 #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.

6.35.4.62 #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.

6.35.4.63 #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.

6.35.4.64 #define rccEnableSPI3(*lp*) rccEnableAPB1(RCC_APB1ENR_SPI3EN, *lp*)

Enables the SPI3 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.

```
6.35.4.65 #define rccDisableSPI3( lp ) rccDisableAPB1(RCC_APB1ENR_SPI3EN, lp)
```

Disables the SPI3 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.

```
6.35.4.66 #define rccResetSPI3( ) rccResetAPB1(RCC_APB1RSTR_SPI3RST)
```

Resets the SPI3 peripheral.

Function Class:

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

```
6.35.4.67 #define rccEnableTIM1( lp ) rccEnableAPB2(RCC_APB2ENR_TIM1EN, lp)
```

Enables the TIM1 peripheral clock.

Note

The *lp* parameter is ignored in this family.

Parameters

in *lp* low power enable flag

Function Class:

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

```
6.35.4.68 #define rccDisableTIM1( lp ) rccDisableAPB2(RCC_APB2ENR_TIM1EN, lp)
```

Disables the TIM1 peripheral clock.

Note

The *lp* parameter is ignored in this family.

Parameters

in *lp* low power enable flag

Function Class:

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

```
6.35.4.69 #define rccResetTIM1( ) rccResetAPB2(RCC_APB2RSTR_TIM1RST)
```

Resets the TIM1 peripheral.

Function Class:

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

6.35.4.70 #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.

6.35.4.71 #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.

6.35.4.72 #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.

6.35.4.73 #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.

6.35.4.74 #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.

6.35.4.75 #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.

6.35.4.76 #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.

6.35.4.77 #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.

6.35.4.78 #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.

6.35.4.79 #define rccEnableTIM5(*lp*) rccEnableAPB1(RCC_APB1ENR_TIM5EN, *lp*)

Enables the TIM5 peripheral clock.

Note

The *lp* parameter is ignored in this family.

Parameters

in *lp* low power enable flag

Function Class:

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

6.35.4.80 #define rccDisableTIM5(*lp*) rccDisableAPB1(RCC_APB1ENR_TIM5EN, *lp*)

Disables the TIM5 peripheral clock.

Note

The *lp* parameter is ignored in this family.

Parameters

in *lp* low power enable flag

Function Class:

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

6.35.4.81 #define rccResetTIM5() rccResetAPB1(RCC_APB1RSTR_TIM5RST)

Resets the TIM5 peripheral.

Function Class:

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

6.35.4.82 #define rccEnableTIM8(*lp*) rccEnableAPB2(RCC_APB2ENR_TIM8EN, *lp*)

Enables the TIM8 peripheral clock.

Note

The *lp* parameter is ignored in this family.

Parameters

in *lp* low power enable flag

Function Class:

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

6.35.4.83 #define rccDisableTIM8(*lp*) rccDisableAPB2(RCC_APB2ENR_TIM8EN, *lp*)

Disables the TIM8 peripheral clock.

Note

The *lp* parameter is ignored in this family.

Parameters

in *lp* low power enable flag

Function Class:

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

6.35.4.84 #define rccResetTIM8() rccResetAPB2(RCC_APB2RSTR_TIM8RST)

Resets the TIM8 peripheral.

Function Class:

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

6.35.4.85 #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.

6.35.4.86 #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.

6.35.4.87 #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.

6.35.4.88 #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.

```
6.35.4.89 #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.

```
6.35.4.90 #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.

```
6.35.4.91 #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.

```
6.35.4.92 #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.

```
6.35.4.93 #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.

```
6.35.4.94 #define rccEnableUSART6( lp ) rccEnableAPB2(RCC_APB2ENR_USART6EN, lp)
```

Enables the USART6 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.

6.35.4.95 #define rccDisableUSART6(*lp*) rccDisableAPB2(RCC_APB2ENR_USART6EN, *lp*)

Disables the USART6 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.

6.35.4.96 #define rccEnableUART4(*lp*) rccEnableAPB1(RCC_APB1ENR_UART4EN, *lp*)

Enables the UART4 peripheral clock.

Note

The *lp* parameter is ignored in this family.

Parameters

in *lp* low power enable flag

Function Class:

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

6.35.4.97 #define rccDisableUART4(*lp*) rccDisableAPB1(RCC_APB1ENR_UART4EN, *lp*)

Disables the UART4 peripheral clock.

Note

The *lp* parameter is ignored in this family.

Parameters

in *lp* low power enable flag

Function Class:

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

6.35.4.98 #define rccResetUART4() rccResetAPB1(RCC_APB1RSTR_UART4RST)

Resets the UART4 peripheral.

Function Class:

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

6.35.4.99 #define rccEnableUART5(*lp*) rccEnableAPB1(RCC_APB1ENR_UART5EN, *lp*)

Enables the UART5 peripheral clock.

Note

The *lp* parameter is ignored in this family.

Parameters

in *lp* low power enable flag

Function Class:

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

6.35.4.100 #define rccDisableUART5(*lp*) rccDisableAPB1(RCC_APB1ENR_UART5EN, *lp*)

Disables the UART5 peripheral clock.

Note

The *lp* parameter is ignored in this family.

Parameters

in *lp* low power enable flag

Function Class:

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

6.35.4.101 #define rccResetUART5() rccResetAPB1(RCC_APB1RSTR_UART5RST)

Resets the UART5 peripheral.

Function Class:

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

6.35.4.102 #define rccResetUSART6() rccResetAPB2(RCC_APB2RSTR_USART6RST)

Resets the USART6 peripheral.

Function Class:

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

Chapter 7

Data Structure Documentation

7.1 ADCConfig Struct Reference

7.1.1 Detailed Description

Driver configuration structure.

Note

It could be empty on some architectures.

```
#include <adc_ll.h>
```

7.2 ADCConversionGroup Struct Reference

7.2.1 Detailed Description

Conversion group configuration structure.

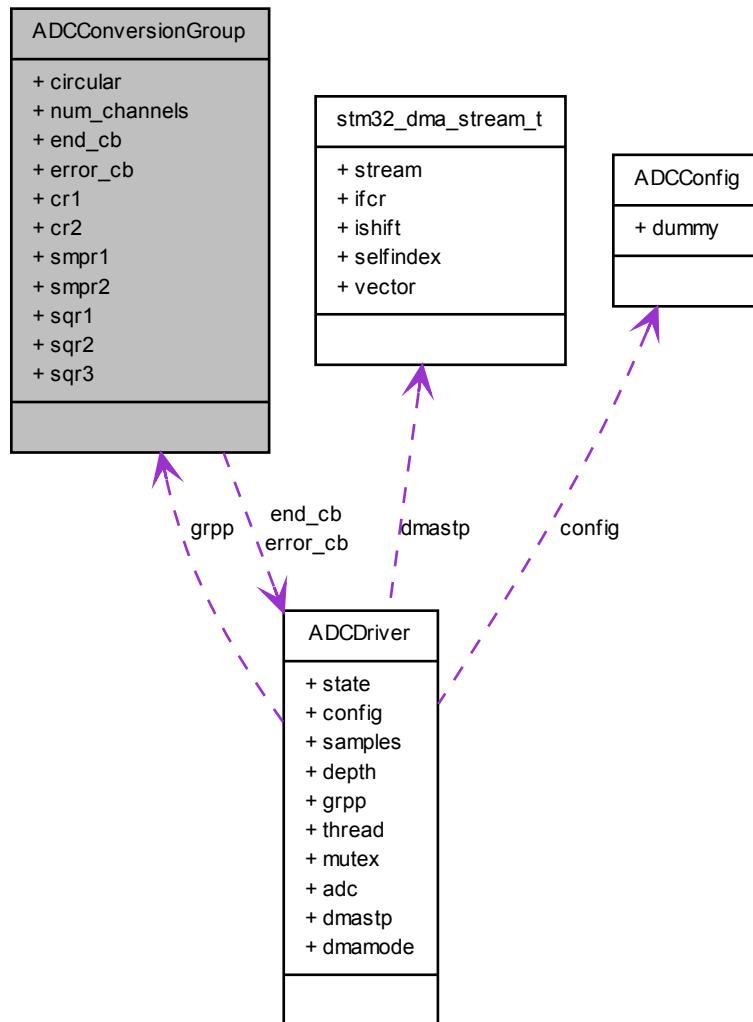
This implementation-dependent structure describes a conversion operation.

Note

The use of this configuration structure requires knowledge of STM32 ADC cell registers interface, please refer to the STM32 reference manual for details.

```
#include <adc_ll.h>
```

Collaboration diagram for ADCConversionGroup:



Data Fields

- `bool_t circular`
Enables the circular buffer mode for the group.
- `adc_channels_num_t num_channels`
Number of the analog channels belonging to the conversion group.
- `adccallback_t end_cb`
Callback function associated to the group or `NULL`.
- `adcerrorcallback_t error_cb`
Error callback or `NULL`.
- `uint32_t cr1`
ADC CR1 register initialization data.
- `uint32_t cr2`

- `uint32_t smpr1`
ADC SMPR1 register initialization data.
- `uint32_t smpr2`
ADC SMPR2 register initialization data.
- `uint32_t sqr1`
ADC SQR1 register initialization data.
- `uint32_t sqr2`
ADC SQR2 register initialization data.
- `uint32_t sqr3`
ADC SQR3 register initialization data.

7.2.2 Field Documentation

7.2.2.1 `bool_t ADCConversionGroup::circular`

Enables the circular buffer mode for the group.

7.2.2.2 `adc_channels_num_t ADCConversionGroup::num_channels`

Number of the analog channels belonging to the conversion group.

7.2.2.3 `adccallback_t ADCConversionGroup::end_cb`

Callback function associated to the group or NULL.

7.2.2.4 `adcerrorcallback_t ADCConversionGroup::error_cb`

Error callback or NULL.

7.2.2.5 `uint32_t ADCConversionGroup::cr1`

ADC CR1 register initialization data.

Note

All the required bits must be defined into this field except `ADC_CR1_SCAN` that is enforced inside the driver.

7.2.2.6 `uint32_t ADCConversionGroup::cr2`

ADC CR2 register initialization data.

Note

All the required bits must be defined into this field except `ADC_CR2_DMA`, `ADC_CR2_CONT` and `ADC_CR2_ADON` that are enforced inside the driver.

7.2.2.7 `uint32_t ADCConversionGroup::smpr1`

ADC SMPR1 register initialization data.

In this field must be specified the sample times for channels 10...18.

7.2.2.8 uint32_t ADCConversionGroup::smpr2

ADC SMPR2 register initialization data.

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

7.2.2.9 uint32_t ADCConversionGroup::sqr1

ADC SQR1 register initialization data.

Conversion group sequence 13...16 + sequence length.

7.2.2.10 uint32_t ADCConversionGroup::sqr2

ADC SQR2 register initialization data.

Conversion group sequence 7...12.

7.2.2.11 uint32_t ADCConversionGroup::sqr3

ADC SQR3 register initialization data.

Conversion group sequence 1...6.

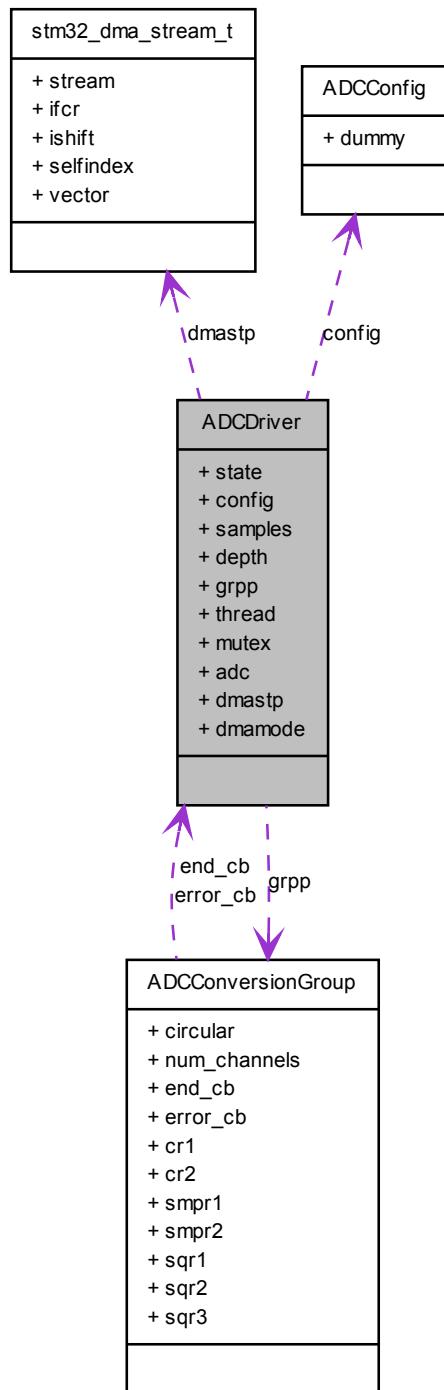
7.3 ADCDriver Struct Reference

7.3.1 Detailed Description

Structure representing an ADC driver.

```
#include <adc_lld.h>
```

Collaboration diagram for ADCDriver:



Data Fields

- `adcstate_t state`

Driver state.

- const [ADCConfig](#) * **config**
Current configuration data.
- [adcsample_t](#) * **samples**
Current samples buffer pointer or NULL.
- [size_t](#) **depth**
Current samples buffer depth or 0.
- const [ADCConversionGroup](#) * **grpp**
Current conversion group pointer or NULL.
- Thread * **thread**
Waiting thread.
- Mutex **mutex**
Mutex protecting the peripheral.
- ADC_TypeDef * **adc**
Pointer to the ADCx registers block.
- const [stm32_dma_stream_t](#) * **dmastp**
Pointer to associated SMA channel.
- uint32_t **dmamode**
DMA mode bit mask.

7.3.2 Field Documentation

7.3.2.1 [adcstate_t](#) ADCDriver::state

Driver state.

7.3.2.2 const [ADCConfig](#)* ADCDriver::config

Current configuration data.

7.3.2.3 [adcsample_t](#)* ADCDriver::samples

Current samples buffer pointer or NULL.

7.3.2.4 [size_t](#) ADCDriver::depth

Current samples buffer depth or 0.

7.3.2.5 const [ADCConversionGroup](#)* ADCDriver::grpp

Current conversion group pointer or NULL.

7.3.2.6 Thread* ADCDriver::thread

Waiting thread.

7.3.2.7 Mutex ADCDriver::mutex

Mutex protecting the peripheral.

7.3.2.8 ADC_TypeDef* ADCDriver::adc

Pointer to the ADCx registers block.

7.3.2.9 const stm32_dma_stream_t* ADCDriver::dmastp

Pointer to associated SMA channel.

7.3.2.10 uint32_t ADCDriver::dmamode

DMA mode bit mask.

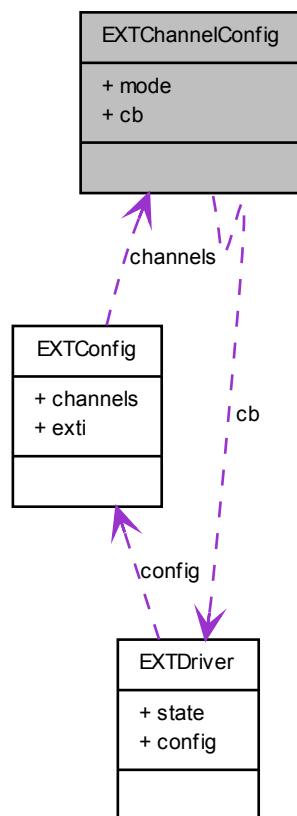
7.4 EXTChannelConfig Struct Reference

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

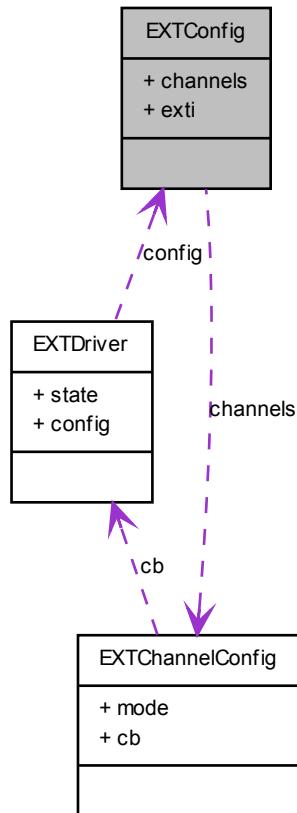
Driver configuration structure.

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.

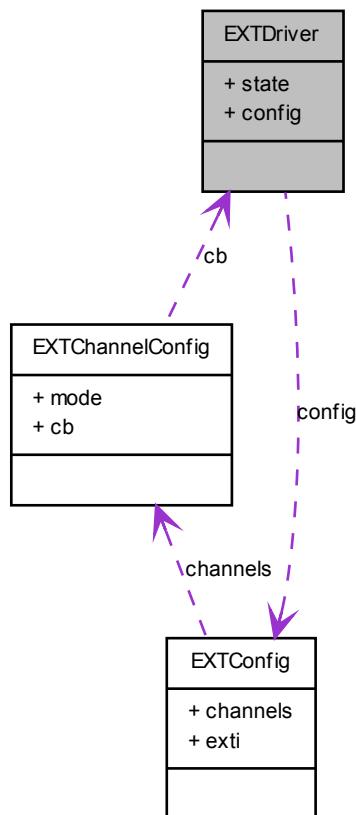
7.6 EXTDriver Struct Reference

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

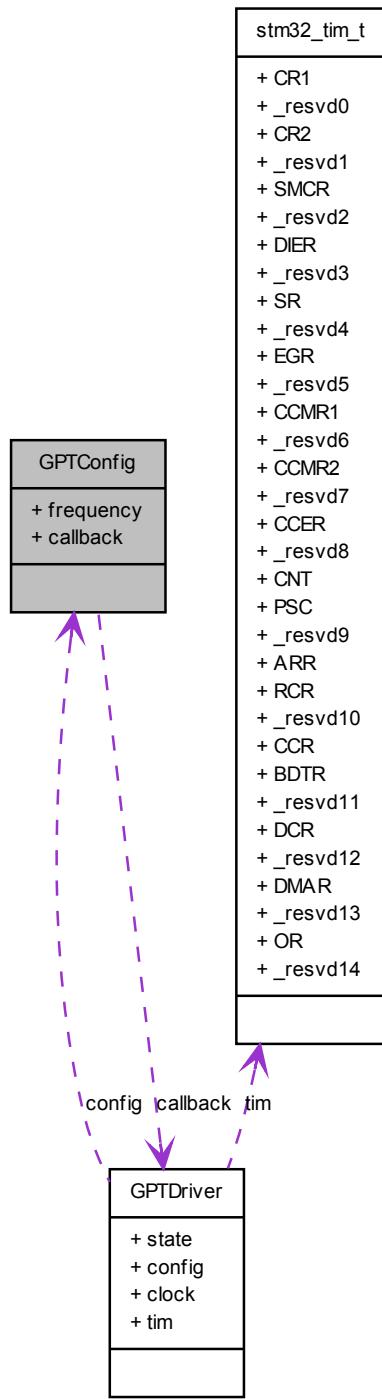
Driver configuration structure.

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.

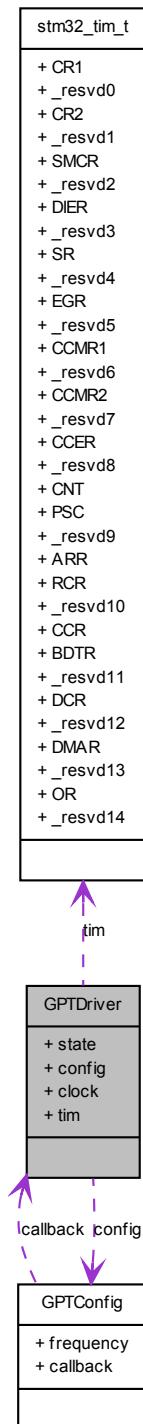
7.9 GPTDriver Struct Reference

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 ICUConfig Struct Reference

7.10.1 Detailed Description

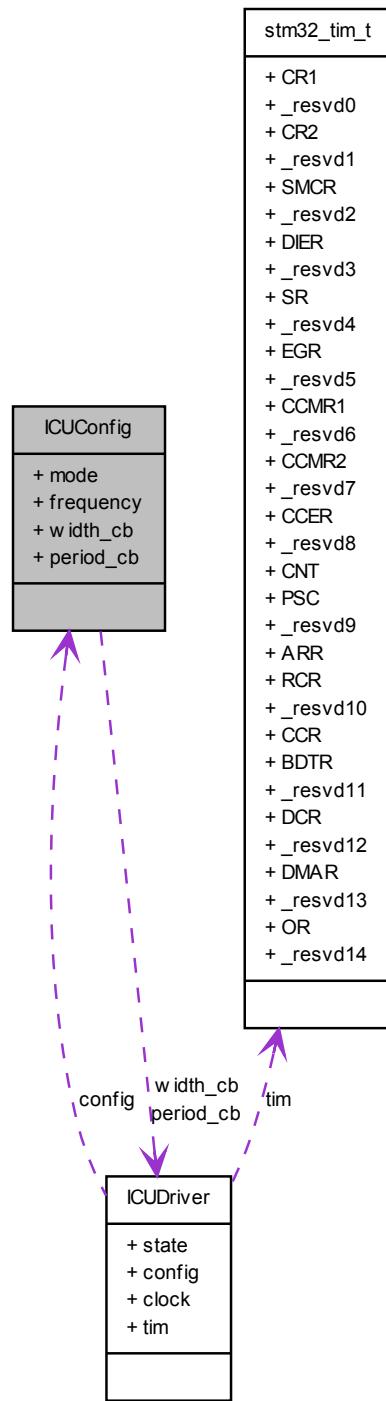
Driver configuration structure.

Note

It could be empty on some architectures.

```
#include <icu_lld.h>
```

Collaboration diagram for ICUConfig:



Data Fields

- `icemode_t mode`

Driver mode.

- **icufreq_t frequency**
Timer clock in Hz.
- **icucallback_t width_cb**
Callback for pulse width measurement.
- **icucallback_t period_cb**
Callback for cycle period measurement.

7.10.2 Field Documentation

7.10.2.1 icumode_t ICUConfig::mode

Driver mode.

7.10.2.2 icufreq_t ICUConfig::frequency

Timer clock in Hz.

Note

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

7.10.2.3 icucallback_t ICUConfig::width_cb

Callback for pulse width measurement.

7.10.2.4 icucallback_t ICUConfig::period_cb

Callback for cycle period measurement.

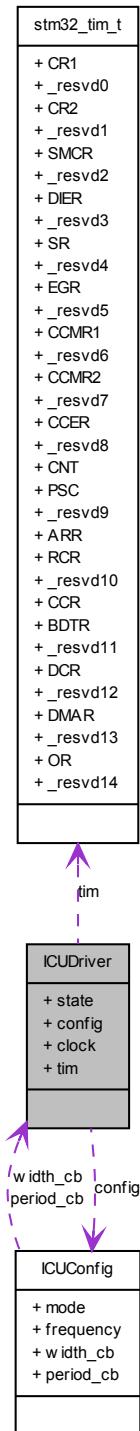
7.11 ICUDriver Struct Reference

7.11.1 Detailed Description

Structure representing an ICU driver.

```
#include <icu_llld.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.

7.11.2 Field Documentation

7.11.2.1 [icustate_t](#) ICUDriver::state

Driver state.

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

Current configuration data.

7.11.2.3 uint32_t ICUDriver::clock

Timer base clock.

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

Pointer to the TIMx registers block.

7.12 IOBus Struct Reference

7.12.1 Detailed Description

I/O bus descriptor.

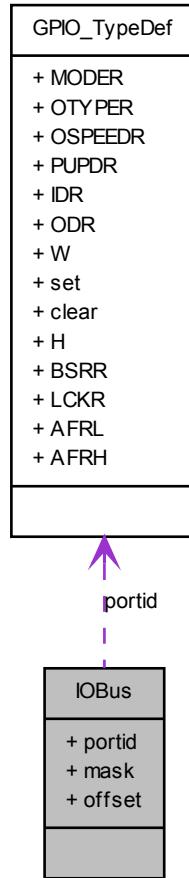
This structure describes a group of contiguous digital I/O lines that have to be handled as bus.

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.

7.12.2 Field Documentation

7.12.2.1 ioprtid_t IOBus::portid

Port identifier.

7.12.2.2 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.

7.12.2.3 uint_fast8_t IOBus::offset

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

7.13 MMCConfig Struct Reference

7.13.1 Detailed Description

Driver configuration structure.

Note

Not required in the current implementation.

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

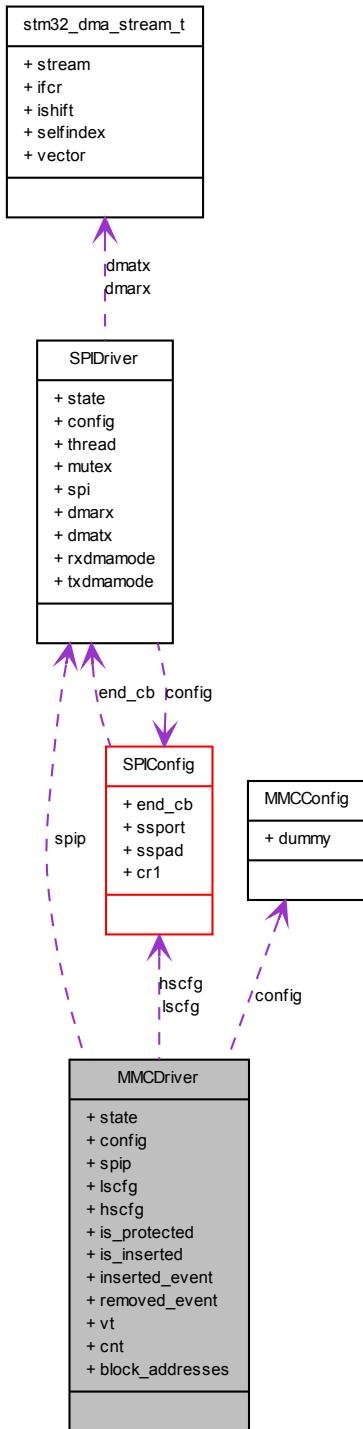
7.14 MMCDriver Struct Reference

7.14.1 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.

7.14.2 Field Documentation

7.14.2.1 mmcstate_t MMCDriver::state

Driver state.

7.14.2.2 const MMCConfig* MMCDriver::config

Current configuration data.

7.14.2.3 SPIDriver* MMCDriver::spip

SPI driver associated to this MMC driver.

7.14.2.4 const SPIConfig* MMCDriver::lscfg

SPI low speed configuration used during initialization.

7.14.2.5 const SPIConfig* MMCDriver::hscfg

SPI high speed configuration used during transfers.

7.14.2.6 mmcquery_t MMCDriver::is_protected

Write protect status query function.

7.14.2.7 mmcquery_t MMCDriver::is_inserted

Insertion status query function.

7.14.2.8 EventSource MMCDriver::inserted_event

Card insertion event source.

7.14.2.9 EventSource MMCDriver::removed_event

Card removal event source.

7.14.2.10 VirtualTimer MMCDriver::vt

MMC insertion polling timer.

7.14.2.11 uint_fast8_t MMCDriver::cnt

Insertion counter.

7.15 PALConfig Struct Reference

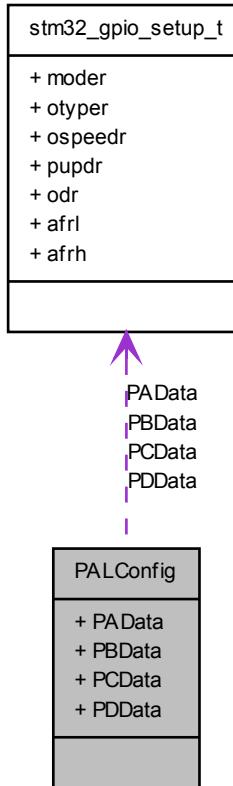
7.15.1 Detailed Description

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.

7.15.2 Field Documentation

7.15.2.1 `stm32_gpio_setup_t PALConfig::PAData`

Port A setup data.

7.15.2.2 `stm32_gpio_setup_t PALConfig::PBData`

Port B setup data.

7.15.2.3 `stm32_gpio_setup_t` PALConfig::PCData

Port C setup data.

7.15.2.4 `stm32_gpio_setup_t` PALConfig::PDData

Port D setup data.

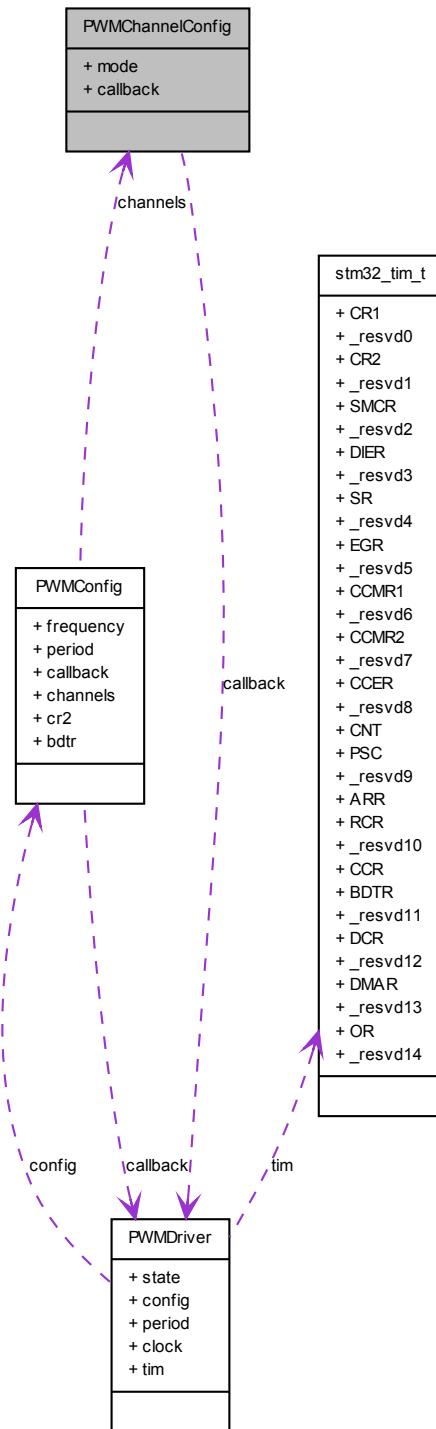
7.16 PWMChannelConfig Struct Reference

7.16.1 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.

7.16.2 Field Documentation

7.16.2.1 pwmmode_t PWMChannelConfig::mode

Channel active logic level.

7.16.2.2 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.

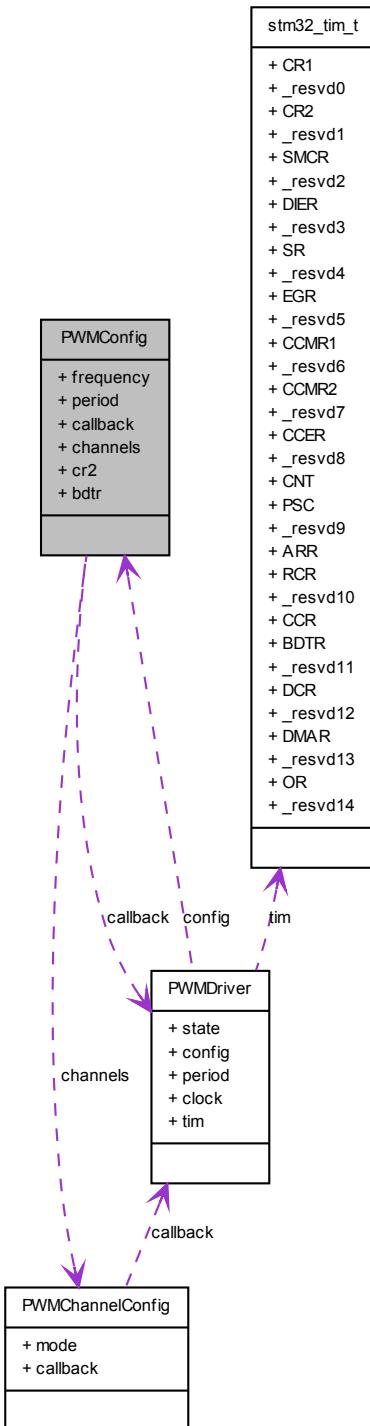
7.17 PWMConfig Struct Reference

7.17.1 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.

7.17.2 Field Documentation

7.17.2.1 `uint32_t PWMConfig::frequency`

Timer clock in Hz.

Note

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

7.17.2.2 `pwmcnt_t PWMConfig::period`

PWM period in ticks.

Note

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

7.17.2.3 `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.

7.17.2.4 `PWMChannelConfig PWMConfig::channels[PWM_CHANNELS]`

Channels configurations.

7.17.2.5 `uint16_t PWMConfig::cr2`

TIM CR2 register initialization data.

Note

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

7.17.2.6 uint16_t PWMConfig::bdtr

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

Note

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

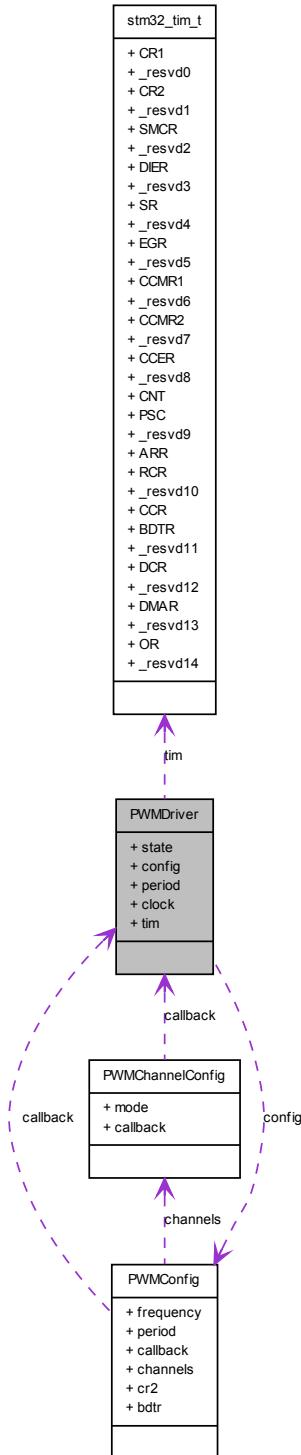
7.18 PWMDriver Struct Reference

7.18.1 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.

7.18.2 Field Documentation

7.18.2.1 [pwmstate_t](#) PWMDriver::state

Driver state.

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

Current driver configuration data.

7.18.2.3 [pwmcnt_t](#) PWMDriver::period

Current PWM period in ticks.

7.18.2.4 [uint32_t](#) PWMDriver::clock

Timer base clock.

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

Pointer to the TIMx registers block.

7.19 SerialConfig Struct Reference

7.19.1 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.

7.19.2 Field Documentation

7.19.2.1 `uint32_t SerialConfig::sc_speed`

Bit rate.

7.19.2.2 `uint16_t SerialConfig::sc_cr1`

Initialization value for the CR1 register.

7.19.2.3 `uint16_t SerialConfig::sc_cr2`

Initialization value for the CR2 register.

7.19.2.4 `uint16_t SerialConfig::sc_cr3`

Initialization value for the CR3 register.

7.20 SerialDriver Struct Reference

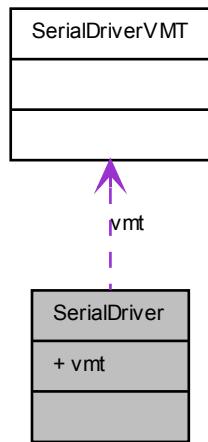
7.20.1 Detailed Description

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.

7.20.2 Field Documentation

7.20.2.1 struct `SerialDriverVMT`* `SerialDriver::vmt`

Virtual Methods Table.

7.21 SerialDriverVMT Struct Reference

7.21.1 Detailed Description

`SerialDriver` virtual methods table.

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

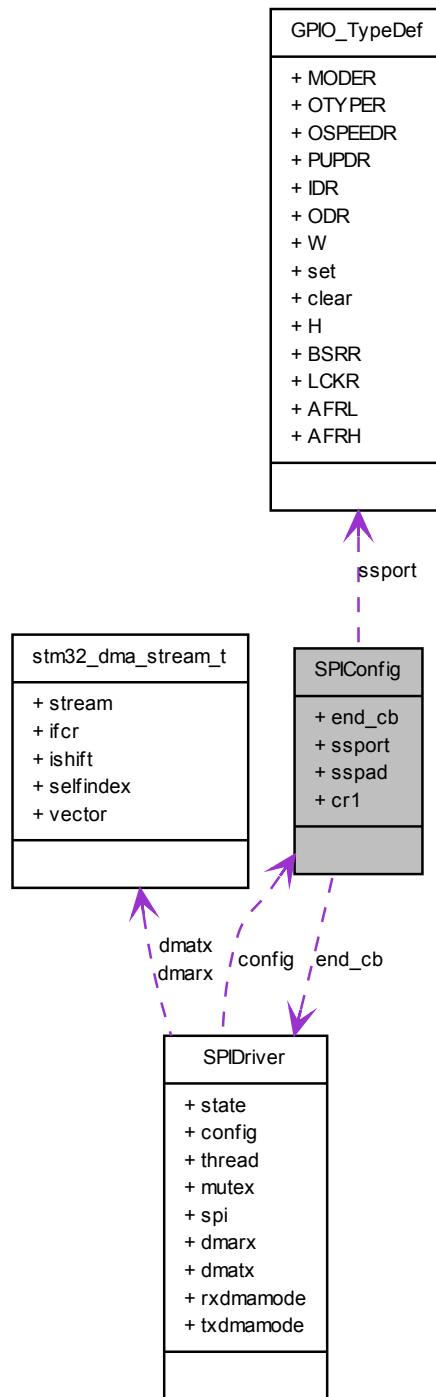
7.22 SPIConfig Struct Reference

7.22.1 Detailed Description

Driver configuration structure.

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

Collaboration diagram for SPIConfig:



Data Fields

- `spicallback_t end_cb`

Operation complete callback or NULL.

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

7.22.2 Field Documentation

7.22.2.1 spicallback_t SPIConfig::end_cb

Operation complete callback or NULL.

7.22.2.2 ioprtid_t SPIConfig::ssport

The chip select line port.

7.22.2.3 uint16_t SPIConfig::sspad

The chip select line pad number.

7.22.2.4 uint16_t SPIConfig::cr1

SPI initialization data.

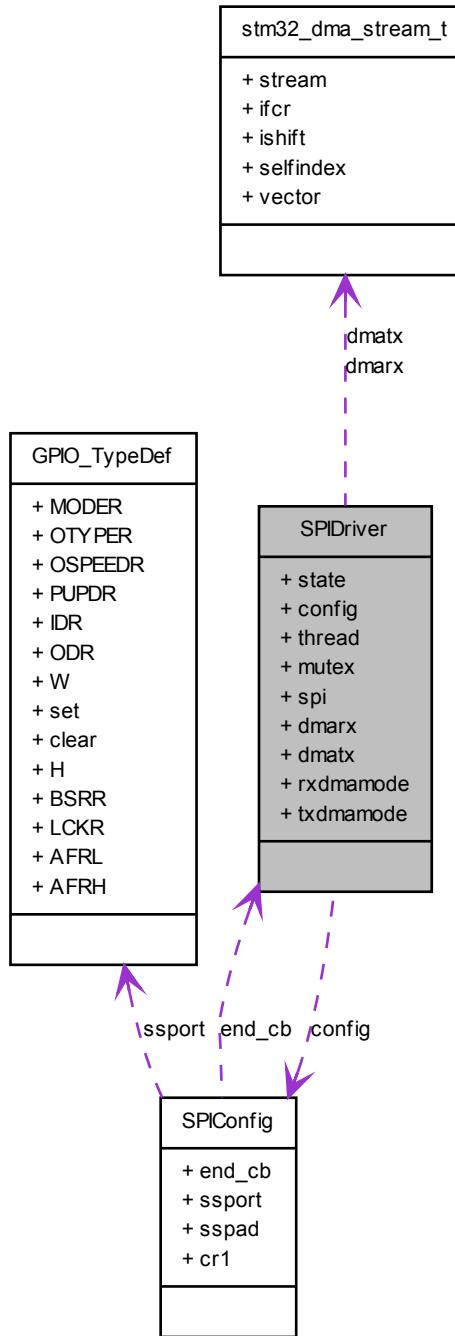
7.23 SPIDriver Struct Reference

7.23.1 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.

7.23.2 Field Documentation

7.23.2.1 spistate_t SPIDriver::state

Driver state.

7.23.2.2 const SPIConfig* SPIDriver::config

Current configuration data.

7.23.2.3 Thread* SPIDriver::thread

Waiting thread.

7.23.2.4 Mutex SPIDriver::mutex

Mutex protecting the bus.

7.23.2.5 SPI_TypeDef* SPIDriver::spi

Pointer to the SPIx registers block.

7.23.2.6 const stm32_dma_stream_t* SPIDriver::dmarx

Receive DMA stream.

7.23.2.7 const stm32_dma_stream_t* SPIDriver::dmatx

Transmit DMA stream.

7.23.2.8 uint32_t SPIDriver::rxdmamode

RX DMA mode bit mask.

7.23.2.9 uint32_t SPIDriver::txdmamemode

TX DMA mode bit mask.

7.24 stm32_dma_stream_t Struct Reference

7.24.1 Detailed Description

STM32 DMA stream descriptor structure.

```
#include <stm32_dma.h>
```

Data Fields

- DMA_Stream_TypeDef * **stream**
Associated DMA stream.
- 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.

7.24.2 Field Documentation

7.24.2.1 DMA_Stream_TypeDef* **stm32_dma_stream_t::stream**

Associated DMA stream.

7.24.2.2 volatile uint32_t* **stm32_dma_stream_t::ifcr**

Associated IFCR reg.

7.24.2.3 uint8_t **stm32_dma_stream_t::ishift**

Bits offset in xIFCR register.

7.24.2.4 uint8_t **stm32_dma_stream_t::selfindex**

Index to self in array.

7.24.2.5 uint8_t **stm32_dma_stream_t::vector**

Associated IRQ vector.

7.25 `stm32_gpio_setup_t` Struct Reference

7.25.1 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`

7.25.2 Field Documentation

7.25.2.1 `uint32_t stm32_gpio_setup_t::moder`

Initial value for MODER register.

7.25.2.2 `uint32_t stm32_gpio_setup_t::otyper`

Initial value for OTYPER register.

7.25.2.3 `uint32_t stm32_gpio_setup_t::ospeedr`

Initial value for OSPEEDR register.

7.25.2.4 `uint32_t stm32_gpio_setup_t::pupdr`

Initial value for PUPDR register.

7.25.2.5 `uint32_t stm32_gpio_setup_t::odr`

Initial value for ODR register.

7.25.2.6 `uint32_t stm32_gpio_setup_t::afrl`

Initial value for AFRL register.

7.25.2.7 `uint32_t stm32_gpio_setup_t::afrh`

Initial value for AFRH register.

7.26 `stm32_tim_t` Struct Reference

7.26.1 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>
```

7.27 TimeMeasurement Struct Reference

7.27.1 Detailed Description

Time Measurement structure.

```
#include <tm.h>
```

Data Fields

- `void(* start)(TimeMeasurement *tmp)`
Starts a measurement.
- `void(* stop)(TimeMeasurement *tmp)`
Stops a measurement.
- `halrtcnt_t last`
Last measurement.
- `halrtcnt_t worst`
Worst measurement.
- `halrtcnt_t best`
Best measurement.

7.27.2 Field Documentation

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

Starts a measurement.

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

Stops a measurement.

7.27.2.3 `halrtcnt_t TimeMeasurement::last`

Last measurement.

7.27.2.4 `halrtcnt_t TimeMeasurement::worst`

Worst measurement.

7.27.2.5 halrtcnt_t TimeMeasurement::best

Best measurement.

7.28 UARTConfig Struct Reference

7.28.1 Detailed Description

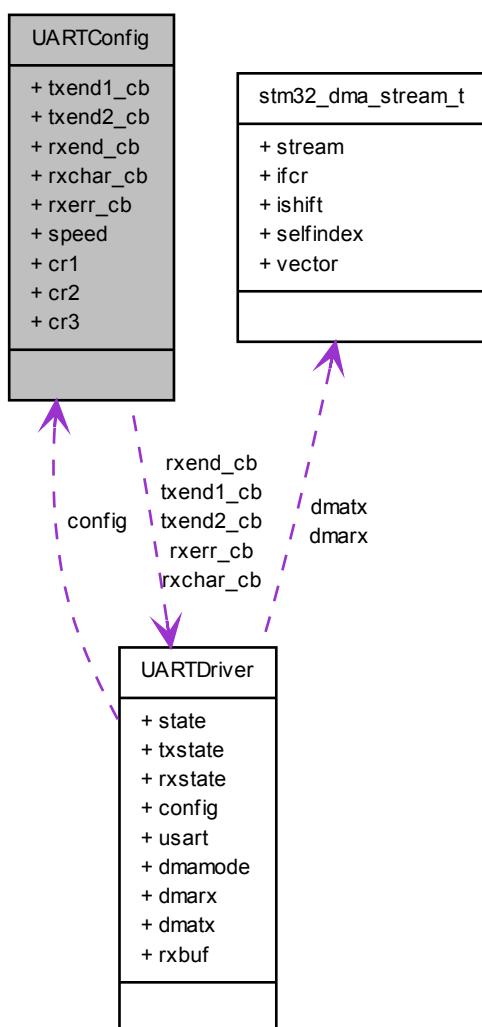
Driver configuration structure.

Note

It could be empty on some architectures.

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

Collaboration diagram for UARTConfig:



Data Fields

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

7.28.2 Field Documentation

7.28.2.1 `uartcb_t` `UARTConfig::txend1_cb`

End of transmission buffer callback.

7.28.2.2 `uartcb_t` `UARTConfig::txend2_cb`

Physical end of transmission callback.

7.28.2.3 `uartcb_t` `UARTConfig::rxend_cb`

Receive buffer filled callback.

7.28.2.4 `uartccb_t` `UARTConfig::rxchar_cb`

Character received while out if the `UART_RECEIVE` state.

7.28.2.5 `uartecb_t` `UARTConfig::rxerr_cb`

Receive error callback.

7.28.2.6 `uint32_t` `UARTConfig::speed`

Bit rate.

7.28.2.7 `uint16_t` `UARTConfig::cr1`

Initialization value for the CR1 register.

7.28.2.8 uint16_t UARTConfig::cr2

Initialization value for the CR2 register.

7.28.2.9 uint16_t UARTConfig::cr3

Initialization value for the CR3 register.

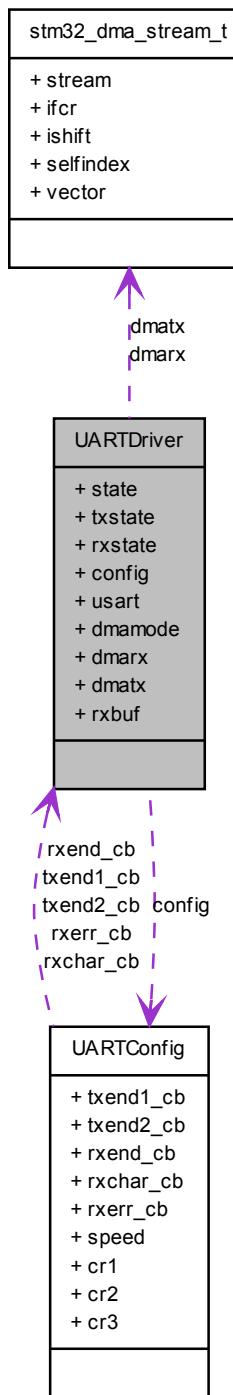
7.29 UARTDriver Struct Reference

7.29.1 Detailed Description

Structure representing an UART driver.

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

Collaboration diagram for UARTDriver:



Data Fields

- `uartstate_t state`
Driver state.

- **uartxstate_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.

7.29.2 Field Documentation

7.29.2.1 **uartstate_t** `UARTDriver::state`

Driver state.

7.29.2.2 **uartxstate_t** `UARTDriver::txstate`

Transmitter state.

7.29.2.3 **uartrxstate_t** `UARTDriver::rxstate`

Receiver state.

7.29.2.4 **const UARTConfig*** `UARTDriver::config`

Current configuration data.

7.29.2.5 **USART_TypeDef*** `UARTDriver::usart`

Pointer to the USART registers block.

7.29.2.6 **uint32_t** `UARTDriver::dmamode`

DMA mode bit mask.

7.29.2.7 **const stm32_dma_stream_t*** `UARTDriver::dmarx`

Receive DMA channel.

7.29.2.8 **const stm32_dma_stream_t*** `UARTDriver::dmatx`

Transmit DMA channel.

7.29.2.9 volatile uint16_t UARTDriver::rdbuf

Default receive buffer while into `UART_RX_IDLE` state.

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

STM32F4xx ADC subsystem low level driver source.

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

Functions

- `CH_IRQ_HANDLER (ADC1_2_3_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.
- `void adcSTM32EnableVBATE (void)`
Enables the VBATE bit.
- `void adcSTM32DisableVBATE (void)`
Disables the VBATE bit.

Variables

- `ADCDriver ADCD1`
ADC1 driver identifier.
- `ADCDriver ADCD2`
ADC2 driver identifier.
- `ADCDriver ADCD3`
ADC3 driver identifier.

8.4 adc_ll.h File Reference

8.4.1 Detailed Description

STM32F4xx 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.
- void [adcSTM32EnableVBATE](#) (void)
Enables the VBATE bit.
- void [adcSTM32DisableVBATE](#) (void)
Disables the VBATE bit.

Defines

Absolute Maximum Ratings

- #define [STM32_ADCCLK_MIN](#) 600000
Minimum ADC clock frequency.
- #define [STM32_ADCCLK_MAX](#) 36000000
Maximum ADC clock frequency.

Triggers selection

- #define [ADC_CR2_EXTSEL_SRC\(n\)](#) ((n) << 24)
Trigger source.

ADC clock divider settings

- #define **ADC_CCR_ADCPRE_DIV2** 0
- #define **ADC_CCR_ADCPRE_DIV4** 1
- #define **ADC_CCR_ADCPRE_DIV6** 2
- #define **ADC_CCR_ADCPRE_DIV8** 3

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_VBAT** 18
VBAT.

Sampling rates

- #define **ADC_SAMPLE_3** 0
3 cycles sampling time.
- #define **ADC_SAMPLE_15** 1
15 cycles sampling time.
- #define **ADC_SAMPLE_28** 2
28 cycles sampling time.
- #define **ADC_SAMPLE_56** 3
56 cycles sampling time.
- #define **ADC_SAMPLE_84** 4
84 cycles sampling time.

- #define ADC_SAMPLE_112 5
112 cycles sampling time.
- #define ADC_SAMPLE_144 6
144 cycles sampling time.
- #define ADC_SAMPLE_480 7
480 cycles sampling time.

Configuration options

- #define STM32_ADC_ADCPRE ADC_CCR_ADCPRE_DIV2
ADC common clock divider.
- #define STM32_ADC_USE_ADC1 TRUE
ADC1 driver enable switch.
- #define STM32_ADC_USE_ADC2 TRUE
ADC2 driver enable switch.
- #define STM32_ADC_USE_ADC3 TRUE
ADC3 driver enable switch.
- #define STM32_ADC_ADC1_DMA_STREAM STM32_DMA_STREAM_ID(2, 4)
DMA stream used for ADC1 operations.
- #define STM32_ADC_ADC2_DMA_STREAM STM32_DMA_STREAM_ID(2, 2)
DMA stream used for ADC2 operations.
- #define STM32_ADC_ADC3_DMA_STREAM STM32_DMA_STREAM_ID(2, 1)
DMA stream used for ADC3 operations.
- #define STM32_ADC_ADC1_DMA_PRIORITY 2
ADC1 DMA priority (0..3|lowest..highest).
- #define STM32_ADC_ADC2_DMA_PRIORITY 2
ADC2 DMA priority (0..3|lowest..highest).
- #define STM32_ADC_ADC3_DMA_PRIORITY 2
ADC3 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.
- #define STM32_ADC_ADC2_DMA_IRQ_PRIORITY 5
ADC2 DMA interrupt priority level setting.
- #define STM32_ADC_ADC3_DMA_IRQ_PRIORITY 5
ADC3 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_SQR3_SQ1_N(n) ((n) << 0)
1st channel in seq.
- #define ADC_SQR3_SQ2_N(n) ((n) << 5)
2nd channel in seq.
- #define ADC_SQR3_SQ3_N(n) ((n) << 10)
3rd channel in seq.
- #define ADC_SQR3_SQ4_N(n) ((n) << 15)
4th channel in seq.
- #define ADC_SQR3_SQ5_N(n) ((n) << 20)
5th channel in seq.
- #define ADC_SQR3_SQ6_N(n) ((n) << 25)
6th channel in seq.
- #define ADC_SQR2_SQ7_N(n) ((n) << 0)
7th channel in seq.
- #define ADC_SQR2_SQ8_N(n) ((n) << 5)
8th channel in seq.
- #define ADC_SQR2_SQ9_N(n) ((n) << 10)

- #define `ADC_SQR2_SQ10_N(n) ((n) << 15)`
9th channel in seq.
- #define `ADC_SQR2_SQ11_N(n) ((n) << 20)`
10th channel in seq.
- #define `ADC_SQR2_SQ12_N(n) ((n) << 25)`
11th channel in seq.
- #define `ADC_SQR1_SQ13_N(n) ((n) << 0)`
12th channel in seq.
- #define `ADC_SQR1_SQ14_N(n) ((n) << 5)`
13th channel in seq.
- #define `ADC_SQR1_SQ15_N(n) ((n) << 10)`
14th channel in seq.
- #define `ADC_SQR1_SQ16_N(n) ((n) << 15)`
15th channel in seq.
- #define `ADC_SQR1_SQ17_N(n) ((n) << 20)`
16th channel in seq.

Sampling rate settings helper macros

- #define `ADC_SMPR2_SMP_AN0(n) ((n) << 0)`
AN0 sampling time.
- #define `ADC_SMPR2_SMP_AN1(n) ((n) << 3)`
AN1 sampling time.
- #define `ADC_SMPR2_SMP_AN2(n) ((n) << 6)`
AN2 sampling time.
- #define `ADC_SMPR2_SMP_AN3(n) ((n) << 9)`
AN3 sampling time.
- #define `ADC_SMPR2_SMP_AN4(n) ((n) << 12)`
AN4 sampling time.
- #define `ADC_SMPR2_SMP_AN5(n) ((n) << 15)`
AN5 sampling time.
- #define `ADC_SMPR2_SMP_AN6(n) ((n) << 18)`
AN6 sampling time.
- #define `ADC_SMPR2_SMP_AN7(n) ((n) << 21)`
AN7 sampling time.
- #define `ADC_SMPR2_SMP_AN8(n) ((n) << 24)`
AN8 sampling time.
- #define `ADC_SMPR2_SMP_AN9(n) ((n) << 27)`
AN9 sampling time.
- #define `ADC_SMPR1_SMP_AN10(n) ((n) << 0)`
AN10 sampling time.
- #define `ADC_SMPR1_SMP_AN11(n) ((n) << 3)`
AN11 sampling time.
- #define `ADC_SMPR1_SMP_AN12(n) ((n) << 6)`
AN12 sampling time.
- #define `ADC_SMPR1_SMP_AN13(n) ((n) << 9)`
AN13 sampling time.
- #define `ADC_SMPR1_SMP_AN14(n) ((n) << 12)`
AN14 sampling time.
- #define `ADC_SMPR1_SMP_AN15(n) ((n) << 15)`
AN15 sampling time.
- #define `ADC_SMPR1_SMP_SENSOR(n) ((n) << 18)`
Temperature Sensor sampling time.
- #define `ADC_SMPR1_SMP_VREF(n) ((n) << 21)`
Voltage Reference sampling time.
- #define `ADC_SMPR1_SMP_VBAT(n) ((n) << 24)`
VBAT 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 can.c File Reference

8.5.1 Detailed Description

CAN Driver code. #include "ch.h"
 #include "hal.h"

Functions

- `void canInit (void)`
CAN Driver initialization.
- `void canObjectInit (CANDriver *canp)`
Initializes the standard part of a CANDriver structure.
- `void canStart (CANDriver *canp, const CANConfig *config)`
Configures and activates the CAN peripheral.
- `void canStop (CANDriver *canp)`
Deactivates the CAN peripheral.
- `msg_t canTransmit (CANDriver *canp, const CANTxFrame *ctfp, systime_t timeout)`
Can frame transmission.
- `msg_t canReceive (CANDriver *canp, CANRxFrame *crfp, systime_t timeout)`
Can frame receive.
- `canstatus_t canGetAndClearFlags (CANDriver *canp)`
Returns the current status mask and clears it.
- `void canSleep (CANDriver *canp)`
Enters the sleep mode.
- `void canWakeup (CANDriver *canp)`
Enforces leaving the sleep mode.

8.6 can.h File Reference

8.6.1 Detailed Description

CAN Driver macros and structures. #include "can_lld.h"

Functions

- void **canInit** (void)
CAN Driver initialization.
- void **canObjectInit** (CANDriver *canp)
Initializes the standard part of a CANDriver structure.
- void **canStart** (CANDriver *canp, const CANConfig *config)
Configures and activates the CAN peripheral.
- void **canStop** (CANDriver *canp)
Deactivates the CAN peripheral.
- msg_t **canTransmit** (CANDriver *canp, const CANTxFrame *ctfp, systime_t timeout)
Can frame transmission.
- msg_t **canReceive** (CANDriver *canp, CANRxFrame *crfp, systime_t timeout)
Can frame receive.
- canstatus_t **canGetAndClearFlags** (CANDriver *canp)
Returns the current status mask and clears it.

Defines

CAN status flags

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

CAN configuration options

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

Macro Functions

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

Enumerations

- enum **canstate_t** {
CAN_UNINIT = 0, **CAN_STOP** = 1, **CAN_STARTING** = 2, **CAN_READY** = 3,
CAN_SLEEP = 4 }
Driver state machine possible states.

8.7 ext.c File Reference

8.7.1 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.

8.8 ext_lld.c File Reference

8.8.1 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_ll_init** (void)
Low level EXT driver initialization.
- void **ext_ll_start** (EXTDriver *extp)
Configures and activates the EXT peripheral.
- void **ext_ll_stop** (EXTDriver *extp)
Deactivates the EXT peripheral.
- void **ext_ll_channel_enable** (EXTDriver *extp, expchannel_t channel)
Enables an EXT channel.
- void **ext_ll_channel_disable** (EXTDriver *extp, expchannel_t channel)
Disables an EXT channel.

Variables

- **EXTDriver EXTD1**
EXTD1 driver identifier.

8.9 ext_ll.h File Reference

8.9.1 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.

8.10 gpt.c File Reference

8.10.1 Detailed Description

GPT Driver code.

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

Functions

- `void gptInit(void)`
GPT Driver initialization.
- `void gptObjectInit(GPTDriver *gptp)`
Initializes the standard part of a GPTDriver structure.
- `void gptStart(GPTDriver *gptp, const GPTConfig *config)`
Configures and activates the GPT peripheral.
- `void gptStop(GPTDriver *gptp)`
Deactivates the GPT peripheral.
- `void gptStartContinuous(GPTDriver *gptp, gptcnt_t interval)`
Starts the timer in continuous mode.
- `void gptStartContinuousl(GPTDriver *gptp, gptcnt_t interval)`
Starts the timer in continuous mode.
- `void gptStartOneShot(GPTDriver *gptp, gptcnt_t interval)`
Starts the timer in one shot mode.
- `void gptStartOneShotl(GPTDriver *gptp, gptcnt_t interval)`
Starts the timer in one shot mode.
- `void gptStopTimer(GPTDriver *gptp)`
Stops the timer.
- `void gptStopTimerl(GPTDriver *gptp)`
Stops the timer.
- `void gptPolledDelay(GPTDriver *gptp, gptcnt_t interval)`
Starts the timer in one shot mode and waits for completion.

8.11 gpt.h File Reference

8.11.1 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.

8.12 gpt_ll.c File Reference

8.12.1 Detailed Description

```
STM32 GPT subsystem low level driver source. #include "ch.h"
#include "hal.h"
```

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.

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.

8.13 gpt_ll.h File Reference

8.13.1 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.

8.14 hal.c File Reference

8.14.1 Detailed Description

```
HAL subsystem code. #include "ch.h"
#include "hal.h"
```

Functions

- void **halInit** (void)
HAL initialization.
- bool_t **hallsCounterWithin** (halrtcnt_t start, halrtcnt_t end)
Realtime window test.
- void **halPolledDelay** (halrtcnt_t ticks)
Polled delay.

8.15 hal.h File Reference

8.15.1 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))

- `#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.

8.16 hal_lld.c File Reference

8.16.1 Detailed Description

STM32F4xx 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)`
STM32F2xx clocks and PLL initialization.

8.17 hal_lld.h File Reference

8.17.1 Detailed Description

STM32F4xx 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.
- STM32_VDD (as hundredths of Volt).

One of the following macros must also be defined:

- STM32F4XX for High-performance STM32 F-4 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)`
STM32F2xx clocks and PLL initialization.

Defines

- `#define HAL_IMPLEMENTS_COUNTERS TRUE`
Defines the support for realtime counters in the HAL.
- `#define STM32_SYSCLK_MAX 168000000`
Maximum SYSCLK.
- `#define STM32_0WS_THRESHOLD 30000000`
Maximum frequency thresholds and wait states for flash access.
- `#define STM32_PLLM (STM32_PLLM_VALUE << 0)`
STM32_PLLM field.
- `#define STM32_PLLCLKIN (STM32_HSECLK / STM32_PLLM_VALUE)`
PLLs input clock frequency.
- `#define STM32_ACTIVATE_PLL TRUE`
PLL activation flag.
- `#define STM32_PLLN (STM32_PLLN_VALUE << 6)`
STM32_PLLN field.
- `#define STM32_PLLP (0 << 16)`
STM32_PLLP field.
- `#define STM32_PLLQ (STM32_PLLQ_VALUE << 24)`
STM32_PLLQ field.
- `#define STM32_PLLVCO (STM32_PLLCLKIN * STM32_PLLN_VALUE)`
PLL VCO frequency.
- `#define STM32_PLLCLKOUT (STM32_PLLVCO / STM32_PLLP_VALUE)`
PLL output clock frequency.
- `#define STM32_SYSCLK STM32_HSICLK`
System clock source.
- `#define STM32_HCLK (STM32_SYSCLK / 1)`
AHB frequency.
- `#define STM32_PCLK1 (STM32_HCLK / 1)`
APB1 frequency.
- `#define STM32_PCLK2 (STM32_HCLK / 1)`
APB2 frequency.
- `#define STM32_ACTIVATE_PLLI2S TRUE`
PLL activation flag.
- `#define STM32_PLLI2SN (STM32_PLLI2SN_VALUE << 6)`
STM32_PLLI2SN field.
- `#define STM32_PLLI2SR (STM32_PLLI2SR_VALUE << 28)`
STM32_PLLI2SR field.
- `#define STM32_PLLI2SVCO (STM32_PLLCLKIN * STM32_PLLI2SN_VALUE)`
PLL VCO frequency.
- `#define STM32_PLLI2SCLKOUT (STM32_PLLI2SVCO / STM32_PLLI2SR)`
PLL I2S output clock frequency.
- `#define STM32_MCO1DIVCLK STM32_HSICLK`
MCO1 divider clock.
- `#define STM32_MCO1CLK STM32_MCO1DIVCLK`
MCO1 output pin clock.
- `#define STM32_MCO2DIVCLK STM32_HSECLK`
MCO2 divider clock.
- `#define STM32_MCO2CLK STM32_MCO2DIVCLK`
MCO2 output pin clock.
- `#define STM32_RTCPRE (STM32_RTCPRE_VALUE << 16)`

- **#define STM32_HSEDIVCLK** (STM32_HSECLK / STM32_RTCPRE_VALUE)

HSE divider toward RTC clock.
- **#define STM32_RTCCLK** 0

RTC clock.
- **#define STM32_RTCPRE** (STM32_RTCPRE_VALUE << 16)

RTC HSE divider setting.
- **#define STM32_PLL48CLK** (STM32_PLLVCO / STM32_PLLQ_VALUE)

48MHz frequency.
- **#define STM32_TIMCLK1** (STM32_PCLK1 * 1)

Timers 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13, 14 clock.
- **#define STM32_TIMCLK2** (STM32_PCLK2 * 1)

Timers 1, 8 clock.
- **#define STM32_FLASHBITS** 0x00000000

Flash settings.
- **#define hal_ll_get_counter_value()** DWT_CYCCNT

Returns the current value of the system free running counter.
- **#define hal_ll_get_counter_frequency()** STM32_HCLK

Realtime counter frequency.

Platform identification

- **#define PLATFORM_NAME** "STM32F4 High Performance & DSP"

Absolute Maximum Ratings

- **#define STM32_HSECLK_MAX** 26000000

Maximum HSE clock frequency.
- **#define STM32_HSECLK_MIN** 1000000

Minimum HSE clock frequency.
- **#define STM32_LSECLK_MAX** 1000000

Maximum LSE clock frequency.
- **#define STM32_LSECLK_MIN** 32768

Minimum LSE clock frequency.
- **#define STM32_PLLIN_MAX** 2000000

Maximum PLLs input clock frequency.
- **#define STM32_PLLIN_MIN** 950000

Minimum PLLs input clock frequency.
- **#define STM32_PLLVCO_MAX** 432000000

Maximum PLLs VCO clock frequency.
- **#define STM32_PLLVCO_MIN** 192000000

Maximum PLLs VCO clock frequency.
- **#define STM32_PLLOUT_MAX** 168000000

Maximum PLL output clock frequency.
- **#define STM32_PLLOUT_MIN** 240000000

Minimum PLL output clock frequency.
- **#define STM32_PCLK1_MAX** 42000000

Maximum APB1 clock frequency.
- **#define STM32_PCLK2_MAX** 84000000

Maximum APB2 clock frequency.
- **#define STM32_SPII2S_MAX** 37500000

Maximum SPI/I2S clock frequency.

Internal clock sources

- **#define STM32_HSICLK** 16000000
- **#define STM32_LSICLK** 32000

PWR_CR register bits definitions

- #define STM32_VOS_MASK (1 << 14)
- #define STM32_VOS_LOW (0 << 14)
- #define STM32_VOS_HIGH (1 << 14)
- #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_PLLCFGR register bits definitions

- #define STM32_PLLP_MASK (3 << 16)
- #define STM32_PLLP_DIV2 (0 << 16)
- #define STM32_PLLP_DIV4 (1 << 16)
- #define STM32_PLLP_DIV6 (2 << 16)
- #define STM32_PLLP_DIV8 (3 << 16)
- #define STM32_PLLSRC_HSI (0 << 22)
- #define STM32_PLLSRC_HSE (1 << 22)

RCC_CFGR register bits definitions

- #define STM32_SW_MASK (3 << 0)
- #define STM32_SW_HSI (0 << 0)
- #define STM32_SW_HSE (1 << 0)
- #define STM32_SW_PLL (2 << 0)
- #define STM32_HPRE_MASK (15 << 4)
- #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_MASK (7 << 10)
- #define STM32_PPREG1_DIV1 (0 << 10)
- #define STM32_PPREG1_DIV2 (4 << 10)
- #define STM32_PPREG1_DIV4 (5 << 10)
- #define STM32_PPREG1_DIV8 (6 << 10)
- #define STM32_PPREG1_DIV16 (7 << 10)
- #define STM32_PPREG2_MASK (7 << 13)
- #define STM32_PPREG2_DIV1 (0 << 13)
- #define STM32_PPREG2_DIV2 (4 << 13)
- #define STM32_PPREG2_DIV4 (5 << 13)
- #define STM32_PPREG2_DIV8 (6 << 13)
- #define STM32_PPREG2_DIV16 (7 << 13)
- #define STM32_RTCPRE_MASK (31 << 16)
- #define STM32_MCO1SEL_MASK (3 << 21)
- #define STM32_MCO1SEL_HSI (0 << 21)
- #define STM32_MCO1SEL_LSE (1 << 21)
- #define STM32_MCO1SEL_HSE (2 << 21)
- #define STM32_MCO1SEL_PLL (3 << 21)
- #define STM32_I2SSRC_MASK (1 << 23)
- #define STM32_I2SSRC_PLLI2S (0 << 23)
- #define STM32_I2SSRC_CKIN (1 << 23)
- #define STM32_MCO1PRE_MASK (7 << 24)
- #define STM32_MCO1PRE_DIV1 (0 << 24)
- #define STM32_MCO1PRE_DIV2 (4 << 24)

- #define STM32_MCO1PRE_DIV3 (5 << 24)
- #define STM32_MCO1PRE_DIV4 (6 << 24)
- #define STM32_MCO1PRE_DIV5 (7 << 24)
- #define STM32_MCO2PRE_MASK (7 << 27)
- #define STM32_MCO2PRE_DIV1 (0 << 27)
- #define STM32_MCO2PRE_DIV2 (4 << 27)
- #define STM32_MCO2PRE_DIV3 (5 << 27)
- #define STM32_MCO2PRE_DIV4 (6 << 27)
- #define STM32_MCO2PRE_DIV5 (7 << 27)
- #define STM32_MCO2SEL_MASK (3U << 30)
- #define STM32_MCO2SEL_SYSCLK (0U << 30)
- #define STM32_MCO2SEL_PLLI2S (1U << 30)
- #define STM32_MCO2SEL_HSE (2U << 30)
- #define STM32_MCO2SEL_PLL (3U << 30)
- #define STM32_RTC_NOCLOCK (0 << 8)
- #define STM32_RTC_LSE (1 << 8)
- #define STM32_RTC_LSI (2 << 8)
- #define STM32_RTC_HSE (3 << 8)

RCC_PLLI2SCFGR register bits definitions

- #define STM32_PLLI2SN_MASK (511 << 6)
- #define STM32_PLLI2SR_MASK (7 << 28)

RCC_BDCR register bits definitions

- #define STM32_RTCSEL_MASK (3 << 8)
- #define STM32_RTCSEL_NOCLOCK (0 << 8)
- #define STM32_RTCSEL_LSE (1 << 8)
- #define STM32_RTCSEL_LSI (2 << 8)
- #define STM32_RTCSEL_HSEDIV (3 << 8)

STM32F4xx capabilities

- #define STM32_HAS_ADC1 TRUE
- #define STM32_ADC1_DMA_MSK
- #define STM32_ADC1_DMA_CHN 0x00000000
- #define STM32_HAS_ADC2 TRUE
- #define STM32_ADC2_DMA_MSK
- #define STM32_ADC2_DMA_CHN 0x00001100
- #define STM32_HAS_ADC3 TRUE
- #define STM32_ADC3_DMA_MSK
- #define STM32_ADC3_DMA_CHN 0x00000022
- #define STM32_HAS_CAN1 TRUE
- #define STM32_HAS_CAN2 TRUE
- #define STM32_CAN_MAX_FILTERS 28
- #define STM32_HAS_DAC TRUE
- #define STM32_ADVANCED_DMA TRUE
- #define STM32_HAS_DMA1 TRUE
- #define STM32_HAS_DMA2 TRUE
- #define STM32_HAS_ETH TRUE
- #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 TRUE
- #define STM32_HAS_GPIOG TRUE
- #define STM32_HAS_GPIOH TRUE
- #define STM32_HAS_GPIOI TRUE
- #define STM32_HAS_I2C1 TRUE
- #define STM32_I2C1_RX_DMA_MSK
- #define STM32_I2C1_RX_DMA_CHN 0x00100001

```
• #define STM32_I2C1_TX_DMA_MSK
• #define STM32_I2C1_TX_DMA_CHN 0x11000000
• #define STM32_HAS_I2C2 TRUE
• #define STM32_I2C2_RX_DMA_MSK
• #define STM32_I2C2_RX_DMA_CHN 0x00007700
• #define STM32_I2C2_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 7))
• #define STM32_I2C2_TX_DMA_CHN 0x70000000
• #define STM32_HAS_I2C3 TRUE
• #define STM32_I2C3_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 2))
• #define STM32_I2C3_RX_DMA_CHN 0x00000300
• #define STM32_I2C3_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 4))
• #define STM32_I2C3_TX_DMA_CHN 0x00030000
• #define STM32_HAS_RTC TRUE
• #define STM32_RTC_HAS_SUBSECONDS TRUE
• #define STM32_RTC_IS_CALENDAR TRUE
• #define STM32_HAS_SDIO TRUE
• #define STM32_SDC_SDIO_DMA_MSK
• #define STM32_SDC_SDIO_DMA_CHN 0x04004000
• #define STM32_HAS_SPI1 TRUE
• #define STM32_SPI1_RX_DMA_MSK
• #define STM32_SPI1_RX_DMA_CHN 0x00000303
• #define STM32_SPI1_TX_DMA_MSK
• #define STM32_SPI1_TX_DMA_CHN 0x00303000
• #define STM32_HAS_SPI2 TRUE
• #define STM32_SPI2_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 3))
• #define STM32_SPI2_RX_DMA_CHN 0x00000000
• #define STM32_SPI2_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 4))
• #define STM32_SPI2_TX_DMA_CHN 0x00000000
• #define STM32_HAS_SPI3 TRUE
• #define STM32_SPI3_RX_DMA_MSK
• #define STM32_SPI3_RX_DMA_CHN 0x00000000
• #define STM32_SPI3_TX_DMA_MSK
• #define STM32_SPI3_TX_DMA_CHN 0x00000000
• #define STM32_HAS_TIM1 TRUE
• #define STM32_HAS_TIM2 TRUE
• #define STM32_HAS_TIM3 TRUE
• #define STM32_HAS_TIM4 TRUE
• #define STM32_HAS_TIM5 TRUE
• #define STM32_HAS_TIM6 TRUE
• #define STM32_HAS_TIM7 TRUE
• #define STM32_HAS_TIM8 TRUE
• #define STM32_HAS_TIM9 TRUE
• #define STM32_HAS_TIM10 TRUE
• #define STM32_HAS_TIM11 TRUE
• #define STM32_HAS_TIM12 TRUE
• #define STM32_HAS_TIM13 TRUE
• #define STM32_HAS_TIM14 TRUE
• #define STM32_HAS_TIM15 FALSE
• #define STM32_HAS_TIM16 FALSE
• #define STM32_HAS_TIM17 FALSE
• #define STM32_HAS_USART1 TRUE
• #define STM32_USART1_RX_DMA_MSK
• #define STM32_USART1_RX_DMA_CHN 0x00400400
• #define STM32_USART1_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(2, 7))
• #define STM32_USART1_TX_DMA_CHN 0x40000000
• #define STM32_HAS_USART2 TRUE
• #define STM32_USART2_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 5))
• #define STM32_USART2_RX_DMA_CHN 0x00400000
• #define STM32_USART2_TX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 6))
• #define STM32_USART2_TX_DMA_CHN 0x04000000
• #define STM32_HAS_USART3 TRUE
• #define STM32_USART3_RX_DMA_MSK (STM32_DMA_STREAM_ID_MSK(1, 1))
• #define STM32_USART3_RX_DMA_CHN 0x00000040
• #define STM32_USART3_TX_DMA_MSK
• #define STM32_USART3_TX_DMA_CHN 0x00074000
```

- #define **STM32_HAS_UART4** TRUE
- #define **STM32_UART4_RX_DMA_MSK** (STM32_DMA_STREAM_ID_MSK(1, 2))
- #define **STM32_UART4_RX_DMA_CHN** 0x00000400
- #define **STM32_UART4_TX_DMA_MSK** (STM32_DMA_STREAM_ID_MSK(1, 4))
- #define **STM32_UART4_TX_DMA_CHN** 0x00040000
- #define **STM32_HAS_UART5** TRUE
- #define **STM32_UART5_RX_DMA_MSK** (STM32_DMA_STREAM_ID_MSK(1, 0))
- #define **STM32_UART5_RX_DMA_CHN** 0x00000004
- #define **STM32_UART5_TX_DMA_MSK** (STM32_DMA_STREAM_ID_MSK(1, 7))
- #define **STM32_UART5_TX_DMA_CHN** 0x40000000
- #define **STM32_HAS_USART6** TRUE
- #define **STM32_USART6_RX_DMA_MSK**
- #define **STM32_USART6_RX_DMA_CHN** 0x00000550
- #define **STM32_USART6_TX_DMA_MSK**
- #define **STM32_USART6_TX_DMA_CHN** 0x55000000
- #define **STM32_HAS_USB** FALSE
- #define **STM32_HAS_OTG1** TRUE
- #define **STM32_HAS_OTG2** TRUE

IRQ VECTOR names

- #define **WWDG_IRQHandler** Vector40
- #define **PVD_IRQHandler** Vector44
- #define **TAMP_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_Stream0_IRQHandler** Vector6C
- #define **DMA1_Stream1_IRQHandler** Vector70
- #define **DMA1_Stream2_IRQHandler** Vector74
- #define **DMA1_Stream3_IRQHandler** Vector78
- #define **DMA1_Stream4_IRQHandler** Vector7C
- #define **DMA1_Stream5_IRQHandler** Vector80
- #define **DMA1_Stream6_IRQHandler** Vector84
- #define **ADC1_2_3_IRQHandler** Vector88
- #define **CAN1_TX_IRQHandler** Vector8C
- #define **CAN1_RX0_IRQHandler** Vector90
- #define **CAN1_RX1_IRQHandler** Vector94
- #define **CAN1_SCE_IRQHandler** Vector98
- #define **EXTI9_5_IRQHandler** Vector9C
- #define **TIM1_BRK_IRQHandler** VectorA0
- #define **TIM1_UP_IRQHandler** VectorA4
- #define **TIM1_TRG_COM_IRQHandler** VectorA8
- #define **TIM1_CC_IRQHandler** VectorAC
- #define **TIM2_IRQHandler** VectorB0
- #define **TIM3_IRQHandler** VectorB4
- #define **TIM4_IRQHandler** VectorB8
- #define **I2C1_EV_IRQHandler** VectorBC
- #define **I2C1_ER_IRQHandler** VectorC0
- #define **I2C2_EV_IRQHandler** VectorC4
- #define **I2C2_ER_IRQHandler** VectorC8
- #define **SPI1_IRQHandler** VectorCC
- #define **SPI2_IRQHandler** VectorD0
- #define **USART1_IRQHandler** VectorD4
- #define **USART2_IRQHandler** VectorD8
- #define **USART3_IRQHandler** VectorDC
- #define **EXTI15_10_IRQHandler** VectorE0
- #define **RTC_Alarm_IRQHandler** VectorE4
- #define **OTG_FS_WKUP_IRQHandler** VectorE8
- #define **TIM8_BRK_IRQHandler** VectorEC

- #define `TIM8_UP_IRQHandler` VectorF0
- #define `TIM8_TRG_COM_IRQHandler` VectorF4
- #define `TIM8_CC_IRQHandler` VectorF8
- #define `DMA1_Stream7_IRQHandler` VectorFC
- #define `FSMC_IRQHandler` Vector100
- #define `SDIO_IRQHandler` Vector104
- #define `TIM5_IRQHandler` Vector108
- #define `SPI3_IRQHandler` Vector10C
- #define `UART4_IRQHandler` Vector110
- #define `UART5_IRQHandler` Vector114
- #define `TIM6_IRQHandler` Vector118
- #define `TIM7_IRQHandler` Vector11C
- #define `DMA2_Stream0_IRQHandler` Vector120
- #define `DMA2_Stream1_IRQHandler` Vector124
- #define `DMA2_Stream2_IRQHandler` Vector128
- #define `DMA2_Stream3_IRQHandler` Vector12C
- #define `DMA2_Stream4_IRQHandler` Vector130
- #define `ETH_IRQHandler` Vector134
- #define `ETH_WKUP_IRQHandler` Vector138
- #define `CAN2_TX_IRQHandler` Vector13C
- #define `CAN2_RX0_IRQHandler` Vector140
- #define `CAN2_RX1_IRQHandler` Vector144
- #define `CAN2_SCE_IRQHandler` Vector148
- #define `OTG_FS_IRQHandler` Vector14C
- #define `DMA2_Stream5_IRQHandler` Vector150
- #define `DMA2_Stream6_IRQHandler` Vector154
- #define `DMA2_Stream7_IRQHandler` Vector158
- #define `USART6_IRQHandler` Vector15C
- #define `I2C3_EV_IRQHandler` Vector160
- #define `I2C3_ER_IRQHandler` Vector164
- #define `OTG_HS_EP1_OUT_IRQHandler` Vector168
- #define `OTG_HS_EP1_IN_IRQHandler` Vector16C
- #define `OTG_HS_WKUP_IRQHandler` Vector170
- #define `OTG_HS_IRQHandler` Vector174
- #define `DCMI_IRQHandler` Vector178
- #define `CRYP_IRQHandler` Vector17C
- #define `HASH_RNG_IRQHandler` Vector180
- #define `FPU_IRQHandler` Vector184

Configuration options

- #define `STM32_NO_INIT` FALSE

Disables the PWR/RCC initialization in the HAL.
- #define `STM32_VOS` STM32_VOS_HIGH

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` FALSE

Enables or disables the LSI clock source.
- #define `STM32_HSE_ENABLED` TRUE

Enables or disables the HSE clock source.
- #define `STM32_LSE_ENABLED` FALSE

Enables or disables the LSE clock source.
- #define `STM32_CLOCK48_REQUIRED` TRUE

USB/SDIO clock setting.
- #define `STM32_SW` STM32_SW_PLL

Main clock source selection.
- #define `STM32_PLLSRC` STM32_PLLSRC_HSE

Clock source for the PLLs.

- `#define STM32_PLLM_VALUE 8`
PLLM divider value.
- `#define STM32_PLLN_VALUE 336`
PLLN multiplier value.
- `#define STM32_PLLP_VALUE 2`
PLLP divider value.
- `#define STM32_PLLQ_VALUE 7`
PLLQ multiplier value.
- `#define STM32_HPRE STM32_HPRE_DIV1`
AHB prescaler value.
- `#define STM32_PPREG1 STM32_PPREG1_DIV4`
APB1 prescaler value.
- `#define STM32_PPREG2 STM32_PPREG2_DIV2`
APB2 prescaler value.
- `#define STM32_RTCSEL STM32_RTCSEL_LSE`
RTC clock source.
- `#define STM32_RTCPRE_VALUE 8`
RTC HSE prescaler value.
- `#define STM32_MCO1SEL STM32_MCO1SEL_HSI`
MC01 clock source value.
- `#define STM32_MCO1PRE STM32_MCO1PRE_DIV1`
MC01 prescaler value.
- `#define STM32_MCO2SEL STM32_MCO2SEL_SYSCLK`
MC02 clock source value.
- `#define STM32_MCO2PRE STM32_MCO2PRE_DIV5`
MC02 prescaler value.
- `#define STM32_I2SSRC STM32_I2SSRC_CKIN`
I2S clock source.
- `#define STM32_PLLI2SN_VALUE 192`
PLL_{I2S}N multiplier value.
- `#define STM32_PLLI2SR_VALUE 5`
PLL_{I2S}R multiplier value.

Typedefs

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

8.18 halconf.h File Reference

8.18.1 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_MMC_SPI` TRUE
Enables the MMC_SPI subsystem.
- #define `HAL_USE_PWM` TRUE
Enables the PWM subsystem.
- #define `HAL_USE_RTC` FALSE
Enables the RTC subsystem.
- #define `HAL_USE_SDC` FALSE
Enables the SDC subsystem.
- #define `HAL_USE_SERIAL` TRUE
Enables the SERIAL subsystem.
- #define `HAL_USE_SERIAL_USB` TRUE
Enables the SERIAL over USB subsystem.
- #define `HAL_USE_SPI` TRUE
Enables the SPI subsystem.
- #define `HAL_USE_UART` TRUE
Enables the UART subsystem.
- #define `HAL_USE_USB` TRUE
Enables the USB subsystem.

ADC driver related setting

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

CAN driver related setting

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

I2C driver related setting

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

MAC driver related setting

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

MMC_SPI driver related setting

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

SDC driver related setting

- `#define SDC_INIT_RETRY 100`
Number of initialization attempts before rejecting the card.
- `#define SDC_MMIC_SUPPORT FALSE`
Include support for MMC cards.
- `#define SDC_NICE_WAITING TRUE`
Delays insertions.

SERIAL driver related setting

- `#define SERIAL_DEFAULT_BITRATE 38400`
Default bit rate.
- `#define SERIAL_BUFFERS_SIZE 16`
Serial buffers size.

SERIAL_USB driver related setting

- `#define SERIAL_USB_BUFFERS_SIZE 64`
Serial over USB buffers size.

SPI driver related setting

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

8.19 i2c.c File Reference

8.19.1 Detailed Description

```
I2C Driver code. #include "ch.h"
#include "hal.h"
```

Functions

- `void i2cInit (void)`
I2C Driver initialization.
- `void i2cObjectInit (I2CDriver *i2cp)`
Initializes the standard part of a `I2CDriver` structure.
- `void i2cStart (I2CDriver *i2cp, const I2CConfig *config)`
Configures and activates the I2C peripheral.

- void **i2cStop** (I2CDriver *i2cp)
Deactivates the I2C peripheral.
- i2cflags_t **i2cGetErrors** (I2CDriver *i2cp)
Returns the errors mask associated to the previous operation.
- msg_t **i2cMasterTransmitTimeout** (I2CDriver *i2cp, i2caddr_t addr, const uint8_t *txbuf, size_t txbytes, uint8_t *rdbuf, size_t rxbytes, systime_t timeout)
Sends data via the I2C bus.
- msg_t **i2cMasterReceiveTimeout** (I2CDriver *i2cp, i2caddr_t addr, uint8_t *rdbuf, size_t rxbytes, systime_t timeout)
Receives data from the I2C bus.
- void **i2cAcquireBus** (I2CDriver *i2cp)
Gains exclusive access to the I2C bus.
- void **i2cReleaseBus** (I2CDriver *i2cp)
Releases exclusive access to the I2C bus.

8.20 i2c.h File Reference

8.20.1 Detailed Description

I2C Driver macros and structures. #include "i2c_llld.h"

Functions

- void **i2cInit** (void)
I2C Driver initialization.
- void **i2cObjectInit** (I2CDriver *i2cp)
Initializes the standard part of a I2CDriver structure.
- void **i2cStart** (I2CDriver *i2cp, const I2CConfig *config)
Configures and activates the I2C peripheral.
- void **i2cStop** (I2CDriver *i2cp)
Deactivates the I2C peripheral.
- i2cflags_t **i2cGetErrors** (I2CDriver *i2cp)
Returns the errors mask associated to the previous operation.
- msg_t **i2cMasterTransmitTimeout** (I2CDriver *i2cp, i2caddr_t addr, const uint8_t *txbuf, size_t txbytes, uint8_t *rdbuf, size_t rxbytes, systime_t timeout)
Sends data via the I2C bus.
- msg_t **i2cMasterReceiveTimeout** (I2CDriver *i2cp, i2caddr_t addr, uint8_t *rdbuf, size_t rxbytes, systime_t timeout)
Receives data from the I2C bus.

Defines

- #define **I2C_USE_MUTUAL_EXCLUSION** TRUE
Enables the mutual exclusion APIs on the I2C bus.
- #define **i2cMasterTransmit**(i2cp, addr, txbuf, txbytes, rdbuf, rxbytes)
Wrap i2cMasterTransmit function with TIME_INFINITE timeout.
- #define **i2cMasterReceive**(i2cp, addr, rdbuf, rxbytes) (i2cMasterReceiveTimeout(i2cp, addr, rdbuf, rxbytes, TIME_INFINITE))
Wrap i2cMasterReceive function with TIME_INFINITE timeout.

I2C bus error conditions

- #define **I2CD_NO_ERROR** 0x00
No error.
- #define **I2CD_BUS_ERROR** 0x01
Bus Error.
- #define **I2CD_ARBITRATION_LOST** 0x02
Arbitration Lost (master mode).
- #define **I2CD_ACK_FAILURE** 0x04
Acknowledge Failure.
- #define **I2CD_OVERRUN** 0x08
Overrun/Underrun.
- #define **I2CD_PEC_ERROR** 0x10
PEC Error in reception.
- #define **I2CD_TIMEOUT** 0x20
Hardware timeout.
- #define **I2CD_SMB_ALERT** 0x40
SMBus Alert.

Enumerations

- enum **i2cstate_t** {

 I2C_UNINIT = 0, **I2C_STOP** = 1, **I2C_READY** = 2, **I2C_ACTIVE_TX** = 3,

 I2C_ACTIVE_RX = 4
 }

Driver state machine possible states.

8.21 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)
Deactivates the ICU peripheral.
- void **icuEnable** (ICUDriver *icup)
Enables the input capture.
- void **icuDisable** (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_lld_init` (void)
Low level ICU driver initialization.
- void `icu_lld_start` (ICUDriver *icup)
Configures and activates the ICU peripheral.
- void `icu_lld_stop` (ICUDriver *icup)
Deactivates the ICU peripheral.
- void `icu_lld_enable` (ICUDriver *icup)
Enables the input capture.
- void `icu_lld_disable` (ICUDriver *icup)
Disables the input capture.

Defines

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

Configuration options

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

Typedefs

- typedef uint32_t `icufreq_t`
ICU frequency type.
- typedef uint16_t `icucnt_t`
ICU counter type.

Enumerations

- enum `icemode_t` { `ICU_INPUT_ACTIVE_HIGH` = 0, `ICU_INPUT_ACTIVE_LOW` = 1 }
- ICU driver mode.*

8.25 mac.c File Reference

8.25.1 Detailed Description

MAC Driver code.

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

Functions

- void `macInit` (void)
MAC Driver initialization.
- void `macObjectInit` (`MACDriver` *`macp`)
Initialize the standard part of a MACDriver structure.
- void `macStart` (`MACDriver` *`macp`, const `MACConfig` *`config`)
Configures and activates the MAC peripheral.
- void `macStop` (`MACDriver` *`macp`)
Deactivates the MAC peripheral.
- `msg_t` `macWaitTransmitDescriptor` (`MACDriver` *`macp`, `MACTransmitDescriptor` *`tdp`, `systime_t` `time`)
Allocates a transmission descriptor.
- void `macReleaseTransmitDescriptor` (`MACTransmitDescriptor` *`tdp`)
Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.
- `msg_t` `macWaitReceiveDescriptor` (`MACDriver` *`macp`, `MACReceiveDescriptor` *`rdp`, `systime_t` `time`)
Waits for a received frame.
- void `macReleaseReceiveDescriptor` (`MACReceiveDescriptor` *`rdp`)
Releases a receive descriptor.
- `bool_t` `macPollLinkStatus` (`MACDriver` *`macp`)
Updates and returns the link status.

8.26 mac.h File Reference

8.26.1 Detailed Description

MAC Driver macros and structures.

```
#include "mac_lld.h"
```

Functions

- void `macInit` (void)
MAC Driver initialization.
- void `macObjectInit` (`MACDriver` *`macp`)
Initialize the standard part of a MACDriver structure.
- void `macStart` (`MACDriver` *`macp`, const `MACConfig` *`config`)
Configures and activates the MAC peripheral.
- void `macStop` (`MACDriver` *`macp`)
Deactivates the MAC peripheral.

- msg_t **macWaitTransmitDescriptor** (MACDriver *macp, MACTransmitDescriptor *tdp, systime_t time)

Allocates a transmission descriptor.
- void **macReleaseTransmitDescriptor** (MACTransmitDescriptor *tdp)

Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.
- msg_t **macWaitReceiveDescriptor** (MACDriver *macp, MACReceiveDescriptor *rdp, systime_t time)

Waits for a received frame.
- void **macReleaseReceiveDescriptor** (MACReceiveDescriptor *rdp)

Releases a receive descriptor.
- bool_t **macPollLinkStatus** (MACDriver *macp)

Updates and returns the link status.

Defines

MAC configuration options

- #define **MAC_USE_EVENTS** TRUE

Enables an event sources for incoming packets.

Macro Functions

- #define **macGetReceiveEventSource**(macp) (&(macp)->rdevent)

Returns the received frames event source.
- #define **macWriteTransmitDescriptor**(tdp, buf, size) mac_lld_write_transmit_descriptor(tdp, buf, size)

Writes to a transmit descriptor's stream.
- #define **macReadReceiveDescriptor**(rdp, buf, size) mac_lld_read_receive_descriptor(rdp, buf, size)

Reads from a receive descriptor's stream.

Typedefs

- typedef struct **MACDriver** MACDriver

Type of a structure representing a MAC driver.

Enumerations

- enum **macstate_t** { **MAC_UNINIT** = 0, **MAC_STOP** = 1, **MAC_ACTIVE** = 2 }

Driver state machine possible states.

8.27 mmc_spi.c File Reference

8.27.1 Detailed Description

```
MMC over SPI driver code. #include <string.h>
#include "ch.h"
#include "hal.h"
```

Functions

- void **mmcInit** (void)

MMC over SPI driver initialization.
- void **mmcObjectInit** (MMCDriver *mmcp, SPIDriver *spip, const SPIConfig *lscfg, const SPIConfig *hscfg, mmcquery_t is_protected, mmcquery_t is_inserted)

- `void mmcStart (MMCDriver *mmcp, const MMCCConfig *config)`
Configures and activates the MMC peripheral.
- `void mmcStop (MMCDriver *mmcp)`
Disables the MMC peripheral.
- `bool_t mmcConnect (MMCDriver *mmcp)`
Performs the initialization procedure on the inserted card.
- `bool_t mmcDisconnect (MMCDriver *mmcp)`
Brings the driver in a state safe for card removal.
- `bool_t mmcStartSequentialRead (MMCDriver *mmcp, uint32_t startblk)`
Starts a sequential read.
- `bool_t mmcSequentialRead (MMCDriver *mmcp, uint8_t *buffer)`
Reads a block within a sequential read operation.
- `bool_t mmcStopSequentialRead (MMCDriver *mmcp)`
Stops a sequential read gracefully.
- `bool_t mmcStartSequentialWrite (MMCDriver *mmcp, uint32_t startblk)`
Starts a sequential write.
- `bool_t mmcSequentialWrite (MMCDriver *mmcp, const uint8_t *buffer)`
Writes a block within a sequential write operation.
- `bool_t mmcStopSequentialWrite (MMCDriver *mmcp)`
Stops a sequential write gracefully.

8.28 mmc_spi.h File Reference

8.28.1 Detailed Description

MMC over SPI driver header.

Data Structures

- struct `MMCCConfig`
Driver configuration structure.
- struct `MMCDriver`
Structure representing a MMC driver.

Functions

- `void mmcInit (void)`
MMC over SPI driver initialization.
- `void mmcObjectInit (MMCDriver *mmcp, SPIDriver *spip, const SPIConfig *lscfg, const SPIConfig *hscfg, mmcquery_t is_protected, mmcquery_t is_inserted)`
Initializes an instance.
- `void mmcStart (MMCDriver *mmcp, const MMCCConfig *config)`
Configures and activates the MMC peripheral.
- `void mmcStop (MMCDriver *mmcp)`
Disables the MMC peripheral.
- `bool_t mmcConnect (MMCDriver *mmcp)`
Performs the initialization procedure on the inserted card.
- `bool_t mmcDisconnect (MMCDriver *mmcp)`
Brings the driver in a state safe for card removal.

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

Defines

MMC_SPI configuration options

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

Macro Functions

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

Typedefs

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

Enumerations

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

8.29 pal.c File Reference

8.29.1 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.

8.30 pal.h File Reference

8.30.1 Detailed Description

I/O Ports Abstraction Layer macros, types and structures. #include "pal_lld.h"

Data Structures

- struct `IOBus`
I/O bus descriptor.

Functions

- `ioportmask_t palReadBus (IOBus *bus)`
Read from an I/O bus.
- `void palWriteBus (IOBus *bus, ioportmask_t bits)`
Write to an I/O bus.
- `void palSetBusMode (IOBus *bus, iomode_t mode)`
Programs a bus with the specified mode.

Defines

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

Pads mode constants

- `#define PAL_MODE_RESET 0`
After reset state.
- `#define PAL_MODE_UNCONNECTED 1`
*Safe state for **unconnected** pads.*
- `#define PAL_MODE_INPUT 2`
Regular input high-Z pad.
- `#define PAL_MODE_INPUT_PULLUP 3`
Input pad with weak pull up resistor.
- `#define PAL_MODE_INPUT_PULLDOWN 4`

- `#define PAL_MODE_INPUT_ANALOG 5`
Analog input mode.
- `#define PAL_MODE_OUTPUT_PUSH_PULL 6`
Push-pull output pad.
- `#define PAL_MODE_OUTPUT_OPENDRAIN 7`
Open-drain output pad.

Logic level constants

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

Macro Functions

- `#define palInit(config) pal_lld_init(config)`
PAL subsystem initialization.
- `#define palReadPort(port) ((void)(port), 0)`
Reads the physical I/O port states.
- `#define palReadLatch(port) ((void)(port), 0)`
Reads the output latch.
- `#define palWritePort(port, bits) ((void)(port), (void)(bits))`
Writes a bits mask on a I/O port.
- `#define palSetPort(port, bits) palWritePort(port, palReadLatch(port) | (bits))`
Sets a bits mask on a I/O port.
- `#define palClearPort(port, bits) palWritePort(port, palReadLatch(port) & ~ (bits))`
Clears a bits mask on a I/O port.
- `#define palTogglePort(port, bits) palWritePort(port, palReadLatch(port) ^ (bits))`
Toggles a bits mask on a I/O port.
- `#define palReadGroup(port, mask, offset) ((palReadPort(port) >> (offset)) & (mask))`
Reads a group of bits.
- `#define palWriteGroup(port, mask, offset, bits)`
Writes a group of bits.
- `#define palSetGroupMode(port, mask, offset, mode)`
Pads group mode setup.
- `#define palReadPad(port, pad) ((palReadPort(port) >> (pad)) & 1)`
Reads an input pad logical state.
- `#define palWritePad(port, pad, bit)`
Writes a logical state on an output pad.
- `#define palSetPad(port, pad) palSetPort(port, PAL_PORT_BIT(pad))`
Sets a pad logical state to PAL_HIGH.
- `#define palClearPad(port, pad) palClearPort(port, PAL_PORT_BIT(pad))`
Clears a pad logical state to PAL_LOW.
- `#define palTogglePad(port, pad) palTogglePort(port, PAL_PORT_BIT(pad))`
Toggles a pad logical state.
- `#define palSetPadMode(port, pad, mode) palSetGroupMode(port, PAL_PORT_BIT(pad), 0, mode)`
Pad mode setup.

8.31 pal_lld.c File Reference

8.31.1 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.

8.32 pal_lld.h File Reference

8.32.1 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`

- `#define IOPORT9 GPIOI`
GPIO port H 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`

- `#define PAL_MODE_INPUT_ANALOG PAL_STM32_MODE_ANALOG`
Analog input mode.
- `#define PAL_MODE_OUTPUT_PUSHPULL`
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.

8.33 pwm.c File Reference

8.33.1 Detailed Description

PWM Driver code.

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

Functions

- `void pwmlinit (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.

8.34 pwm.h File Reference

8.34.1 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.*

8.35 pwm_lld.c File Reference

8.35.1 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`, `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.

8.36 pwm_lld.h File Reference

8.36.1 Detailed Description

STM32 PWM subsystem low level driver header.

Data Structures

- struct **PWMChannelConfig**
PWM driver channel configuration structure.
- struct **PWMConfig**
PWM driver configuration structure.
- struct **PWMDriver**
Structure representing a PWM driver.

Functions

- void **pwm_ll_init** (void)
Low level PWM driver initialization.
- void **pwm_ll_start** (**PWMDriver** *pwmp)
Configures and activates the PWM peripheral.
- void **pwm_ll_stop** (**PWMDriver** *pwmp)
Deactivates the PWM peripheral.
- void **pwm_ll_enable_channel** (**PWMDriver** *pwmp, **pwmchannel_t** channel, **pwmcnt_t** width)
Enables a PWM channel.
- void **pwm_ll_disable_channel** (**PWMDriver** *pwmp, **pwmchannel_t** channel)
Disables a PWM channel.

Defines

- #define **PWM_CHANNELS** 4
Number of PWM channels per PWM driver.
- #define **PWM_COMPLEMENTARY_OUTPUT_MASK** 0xF0
Complementary output modes mask.
- #define **PWM_COMPLEMENTARY_OUTPUT_DISABLED** 0x00
Complementary output not driven.
- #define **PWM_COMPLEMENTARY_OUTPUT_ACTIVE_HIGH** 0x10
Complementary output, active is logic level one.
- #define **PWM_COMPLEMENTARY_OUTPUT_ACTIVE_LOW** 0x20
Complementary output, active is logic level zero.
- #define **pwm_ll_change_period**(pwmp, period) ((pwmp)->tim->ARR = (uint16_t)((period) - 1))
Changes the period the PWM peripheral.

Configuration options

- #define **STM32_PWM_USE_ADVANCED** TRUE
If advanced timer features switch.
- #define **STM32_PWM_USE_TIM1** TRUE
PWMD1 driver enable switch.
- #define **STM32_PWM_USE_TIM2** TRUE
PWMD2 driver enable switch.
- #define **STM32_PWM_USE_TIM3** TRUE
PWMD3 driver enable switch.
- #define **STM32_PWM_USE_TIM4** TRUE
PWMD4 driver enable switch.
- #define **STM32_PWM_USE_TIM5** TRUE
PWMD5 driver enable switch.
- #define **STM32_PWM_USE_TIM8** TRUE
PWMD8 driver enable switch.

- `#define STM32_PWM_TIM1_IRQ_PRIORITY 7`
PWMD1 interrupt priority level setting.
- `#define STM32_PWM_TIM2_IRQ_PRIORITY 7`
PWMD2 interrupt priority level setting.
- `#define STM32_PWM_TIM3_IRQ_PRIORITY 7`
PWMD3 interrupt priority level setting.
- `#define STM32_PWM_TIM4_IRQ_PRIORITY 7`
PWMD4 interrupt priority level setting.
- `#define STM32_PWM_TIM5_IRQ_PRIORITY 7`
PWMD5 interrupt priority level setting.
- `#define STM32_PWM_TIM8_IRQ_PRIORITY 7`
PWMD8 interrupt priority level setting.

Typedefs

- `typedef uint32_t pwmmode_t`
PWM mode type.
- `typedef uint8_t pwmchannel_t`
PWM channel type.
- `typedef uint16_t pwmcnt_t`
PWM counter type.

8.37 sdc.c File Reference

8.37.1 Detailed Description

SDC Driver code.

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

Functions

- `bool_t _sdc_wait_for_transfer_state (SDCDriver *sdcp)`
Wait for the card to complete pending operations.
- `void sdclinit (void)`
SDC Driver initialization.
- `void sdcObjectInit (SDCDriver *sdcp)`
Initializes the standard part of a SDCDriver structure.
- `void sdcStart (SDCDriver *sdcp, const SDCCConfig *config)`
Configures and activates the SDC peripheral.
- `void sdcStop (SDCDriver *sdcp)`
Deactivates the SDC peripheral.
- `bool_t sdcConnect (SDCDriver *sdcp)`
Performs the initialization procedure on the inserted card.
- `bool_t sdcDisconnect (SDCDriver *sdcp)`
Brings the driver in a state safe for card removal.
- `bool_t sdcRead (SDCDriver *sdcp, uint32_t startblk, uint8_t *buf, uint32_t n)`
Reads one or more blocks.
- `bool_t sdcWrite (SDCDriver *sdcp, uint32_t startblk, const uint8_t *buf, uint32_t n)`
Writes one or more blocks.

8.38 sdc.h File Reference

8.38.1 Detailed Description

SDC Driver macros and structures. #include "sdc_lld.h"

Functions

- void **sdcInit** (void)
SDC Driver initialization.
- void **sdcObjectInit** (SDCDriver *sdcp)
Initializes the standard part of a SDCDriver structure.
- void **sdcStart** (SDCDriver *sdcp, const SDCCConfig *config)
Configures and activates the SDC peripheral.
- void **sdcStop** (SDCDriver *sdcp)
Deactivates the SDC peripheral.
- bool_t **sdcConnect** (SDCDriver *sdcp)
Performs the initialization procedure on the inserted card.
- bool_t **sdcDisconnect** (SDCDriver *sdcp)
Brings the driver in a state safe for card removal.
- bool_t **sdcRead** (SDCDriver *sdcp, uint32_t startblk, uint8_t *buf, uint32_t n)
Reads one or more blocks.
- bool_t **sdcWrite** (SDCDriver *sdcp, uint32_t startblk, const uint8_t *buf, uint32_t n)
Writes one or more blocks.
- bool_t **_sdc_wait_for_transfer_state** (SDCDriver *sdcp)
Wait for the card to complete pending operations.

Defines

- #define **SDC_BLOCK_SIZE** 512
- #define **SDC_CMD8_PATTERN** 0x0000001AA
Fixed pattern for CMD8.
- #define **SDC_R1_ERROR_MASK** 0xFDFFFE008
Mask of error bits in R1 responses.

SD card types

- #define **SDC_MODE_CARDTYPE_MASK** 0xF
Card type mask.
- #define **SDC_MODE_CARDTYPE_SDV11** 0
Card is SD V1.1.
- #define **SDC_MODE_CARDTYPE_SDV20** 1
Card is SD V2.0.
- #define **SDC_MODE_CARDTYPE_MMC** 2
Card is MMC.
- #define **SDC_MODE_HIGH_CAPACITY** 0x10
High cap.card.

SDC configuration options

- #define **SDC_INIT_RETRY** 100
Number of initialization attempts before rejecting the card.
- #define **SDC_MMC_SUPPORT** FALSE
Include support for MMC cards.
- #define **SDC_NICE_WAITING** TRUE
Delays insertions.

R1 response utilities

- `#define SDC_R1_ERROR(r1) (((r1) & SDC_R1_ERROR_MASK) != 0)`
Evaluates to TRUE if the R1 response contains error flags.
- `#define SDC_R1_STS(r1) (((r1) >> 9) & 15)`
Returns the status field of an R1 response.
- `#define SDC_R1_IS_CARD_LOCKED(r1) (((r1) >> 21) & 1)`
Evaluates to TRUE if the R1 response indicates a locked card.

Macro Functions

- `#define sdcGetDriverState(sdc) ((sdc)->state)`
Returns the driver state.
- `#define sdclsCardInserted(sdc) (sdc_lld_is_card_inserted(sdc))`
Returns the card insertion status.
- `#define sdclsWriteProtected(sdc) (sdc_lld_is_write_protected(sdc))`
Returns the write protect status.

Enumerations

- enum `sdcstate_t` {

`SDC_UNINIT = 0, SDC_STOP = 1, SDC_READY = 2, SDC_CONNECTING = 3,`

`SDC_DISCONNECTING = 4, SDC_ACTIVE = 5, SDC_READING = 6, SDC_WRITING = 7 }`

Driver state machine possible states.

8.39 serial.c File Reference

8.39.1 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.

8.40 serial.h File Reference

8.40.1 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.

8.41 serial_lld.c File Reference

8.41.1 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)`

- **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.

8.42 serial_lld.h File Reference

8.42.1 Detailed Description

STM32 low level serial driver header.

Data Structures

- **struct SerialConfig**
STM32 Serial Driver configuration structure.

Functions

- **void sd_lld_init (void)**
Low level serial driver initialization.
- **void sd_lld_start (SerialDriver *sdp, const SerialConfig *config)**
Low level serial driver configuration and (re)start.
- **void sd_lld_stop (SerialDriver *sdp)**
Low level serial driver stop.

Defines

- **#define _serial_driver_data**
SerialDriver specific data.
- **#define USART_CR2_STOP1_BITS (0 << 12)**
CR2 1 stop bit value.
- **#define USART_CR2_STOP0P5_BITS (1 << 12)**
CR2 0.5 stop bit value.

- `#define USART_CR2_STOP2_BITS (2 << 12)`
CR2 2 stop bit value.
- `#define USART_CR2_STOP1P5_BITS (3 << 12)`
CR2 1.5 stop bit value.

Configuration options

- `#define STM32_SERIAL_USE_USART1 TRUE`
USART1 driver enable switch.
- `#define STM32_SERIAL_USE_USART2 TRUE`
USART2 driver enable switch.
- `#define STM32_SERIAL_USE_USART3 TRUE`
USART3 driver enable switch.
- `#define STM32_SERIAL_USE_UART4 TRUE`
UART4 driver enable switch.
- `#define STM32_SERIAL_USE_UART5 TRUE`
UART5 driver enable switch.
- `#define STM32_SERIAL_USE_USART6 TRUE`
USART6 driver enable switch.
- `#define STM32_SERIAL_USART1_PRIORITY 12`
USART1 interrupt priority level setting.
- `#define STM32_SERIAL_USART2_PRIORITY 12`
USART2 interrupt priority level setting.
- `#define STM32_SERIAL_USART3_PRIORITY 12`
USART3 interrupt priority level setting.
- `#define STM32_SERIAL_UART4_PRIORITY 12`
UART4 interrupt priority level setting.
- `#define STM32_SERIAL_UART5_PRIORITY 12`
UART5 interrupt priority level setting.
- `#define STM32_SERIAL_USART6_PRIORITY 12`
USART6 interrupt priority level setting.

8.43 spi.c File Reference

8.43.1 Detailed Description

SPI Driver code.

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

Functions

- `void spilinit (void)`
SPI Driver initialization.
- `void spiObjectInit (SPIDriver *spip)`
Initializes the standard part of a `SPIDriver` structure.
- `void spiStart (SPIDriver *spip, const SPIConfig *config)`
Configures and activates the SPI peripheral.
- `void spiStop (SPIDriver *spip)`
Deactivates the SPI peripheral.
- `void spiSelect (SPIDriver *spip)`
Asserts the slave select signal and prepares for transfers.
- `void spiUnselect (SPIDriver *spip)`
Deasserts the slave select signal.

- void **spiStartIgnore** (**SPIDriver** *spip, **size_t** n)
Ignores data on the SPI bus.
- void **spiStartExchange** (**SPIDriver** *spip, **size_t** n, const void *txbuf, void *rxbuf)
Exchanges data on the SPI bus.
- void **spiStartSend** (**SPIDriver** *spip, **size_t** n, const void *txbuf)
Sends data over the SPI bus.
- void **spiStartReceive** (**SPIDriver** *spip, **size_t** n, void *rxbuf)
Receives data from the SPI bus.
- void **spignore** (**SPIDriver** *spip, **size_t** n)
Ignores data on the SPI bus.
- void **spiExchange** (**SPIDriver** *spip, **size_t** n, const void *txbuf, void *rxbuf)
Exchanges data on the SPI bus.
- void **spiSend** (**SPIDriver** *spip, **size_t** n, const void *txbuf)
Sends data over the SPI bus.
- void **spiReceive** (**SPIDriver** *spip, **size_t** n, void *rxbuf)
Receives data from the SPI bus.
- void **spiAcquireBus** (**SPIDriver** *spip)
Gains exclusive access to the SPI bus.
- void **spiReleaseBus** (**SPIDriver** *spip)
Releases exclusive access to the SPI bus.

8.44 spi.h File Reference

8.44.1 Detailed Description

SPI Driver macros and structures. #include "spi_llld.h"

Functions

- void **spinit** (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 `spiStartIgnore`(spip, n)
Ignores data on the SPI bus.
- #define `spiStartExchange`(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.

8.45 spi_lld.c File Reference

8.45.1 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)

- 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.

8.46 spi_lld.h File Reference

8.46.1 Detailed Description

STM32 SPI subsystem low level driver header.

Data Structures

- struct **SPICConfig**

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)

- **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 *rdbuf)**
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 *rdbuf)**
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.

8.47 stm32.h File Reference

8.47.1 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)

8.48 stm32_dma.c File Reference

8.48.1 Detailed Description

Enhanced DMA helper driver code.

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

Functions

- **CH_IRQ_HANDLER** (DMA1_Stream0_IRQHandler)
DMA1 stream 0 shared interrupt handler.
- **CH_IRQ_HANDLER** (DMA1_Stream1_IRQHandler)
DMA1 stream 1 shared interrupt handler.
- **CH_IRQ_HANDLER** (DMA1_Stream2_IRQHandler)
DMA1 stream 2 shared interrupt handler.
- **CH_IRQ_HANDLER** (DMA1_Stream3_IRQHandler)
DMA1 stream 3 shared interrupt handler.
- **CH_IRQ_HANDLER** (DMA1_Stream4_IRQHandler)
DMA1 stream 4 shared interrupt handler.
- **CH_IRQ_HANDLER** (DMA1_Stream5_IRQHandler)
DMA1 stream 5 shared interrupt handler.
- **CH_IRQ_HANDLER** (DMA1_Stream6_IRQHandler)
DMA1 stream 6 shared interrupt handler.
- **CH_IRQ_HANDLER** (DMA1_Stream7_IRQHandler)
DMA1 stream 7 shared interrupt handler.
- **CH_IRQ_HANDLER** (DMA2_Stream0_IRQHandler)
DMA2 stream 0 shared interrupt handler.
- **CH_IRQ_HANDLER** (DMA2_Stream1_IRQHandler)
DMA2 stream 1 shared interrupt handler.
- **CH_IRQ_HANDLER** (DMA2_Stream2_IRQHandler)
DMA2 stream 2 shared interrupt handler.
- **CH_IRQ_HANDLER** (DMA2_Stream3_IRQHandler)
DMA2 stream 3 shared interrupt handler.
- **CH_IRQ_HANDLER** (DMA2_Stream4_IRQHandler)
DMA2 stream 4 shared interrupt handler.
- **CH_IRQ_HANDLER** (DMA2_Stream5_IRQHandler)
DMA2 stream 5 shared interrupt handler.
- **CH_IRQ_HANDLER** (DMA2_Stream6_IRQHandler)
DMA2 stream 6 shared interrupt handler.
- **CH_IRQ_HANDLER** (DMA2_Stream7_IRQHandler)
DMA2 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 0x000000FF`
Mask of the DMA1 streams in dma_streams_mask.
- `#define STM32_DMA2_STREAMS_MASK 0x0000FF00`
Mask of the DMA2 streams in dma_streams_mask.
- `#define STM32_DMA_CR_RESET_VALUE 0x00000000`
Post-reset value of the stream CR register.
- `#define STM32_DMA_FCR_RESET_VALUE 0x00000021`
Post-reset value of the stream FCR register.

8.49 stm32_dma.h File Reference

8.49.1 Detailed Description

Enhanced-DMA helper driver header.

Note

This file requires definitions from the ST STM32F4xx header file `stm32f4xx.h`.

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 16`
Total number of DMA streams.
- `#define STM32_DMA_ISR_MASK 0x3D`
Mask of the ISR bits passed to the DMA callback functions.
- `#define STM32_DMA_GETCHANNEL(id, c) (((c) >> (((id) & 7) * 4)) & 7)`
Returns the channel associated to the specified stream.
- `#define STM32_DMA_STREAM_ID(dma, stream) (((dma) - 1) * 8) + (stream)`
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.

DMA streams identifiers

- `#define STM32_DMA_STREAM(id) (&_stm32_dma_streams[id])`
Returns a pointer to a `stm32_dma_stream_t` structure.
- `#define STM32_DMA1_STREAM0 STM32_DMA_STREAM(0)`
- `#define STM32_DMA1_STREAM1 STM32_DMA_STREAM(1)`
- `#define STM32_DMA1_STREAM2 STM32_DMA_STREAM(2)`
- `#define STM32_DMA1_STREAM3 STM32_DMA_STREAM(3)`
- `#define STM32_DMA1_STREAM4 STM32_DMA_STREAM(4)`
- `#define STM32_DMA1_STREAM5 STM32_DMA_STREAM(5)`
- `#define STM32_DMA1_STREAM6 STM32_DMA_STREAM(6)`
- `#define STM32_DMA1_STREAM7 STM32_DMA_STREAM(7)`
- `#define STM32_DMA2_STREAM0 STM32_DMA_STREAM(8)`
- `#define STM32_DMA2_STREAM1 STM32_DMA_STREAM(9)`
- `#define STM32_DMA2_STREAM2 STM32_DMA_STREAM(10)`
- `#define STM32_DMA2_STREAM3 STM32_DMA_STREAM(11)`
- `#define STM32_DMA2_STREAM4 STM32_DMA_STREAM(12)`
- `#define STM32_DMA2_STREAM5 STM32_DMA_STREAM(13)`
- `#define STM32_DMA2_STREAM6 STM32_DMA_STREAM(14)`
- `#define STM32_DMA2_STREAM7 STM32_DMA_STREAM(15)`

CR register constants common to all DMA types

- `#define STM32_DMA_CR_EN DMA_SxCR_EN`
- `#define STM32_DMA_CR_TEIE DMA_SxCR_TEIE`
- `#define STM32_DMA_CR_HTIE DMA_SxCR_HTIE`
- `#define STM32_DMA_CR_TCIE DMA_SxCR_TCIE`
- `#define STM32_DMA_CR_DIR_MASK DMA_SxCR_DIR`
- `#define STM32_DMA_CR_DIR_P2M 0`
- `#define STM32_DMA_CR_DIR_M2P DMA_SxCR_DIR_0`
- `#define STM32_DMA_CR_DIR_M2M DMA_SxCR_DIR_1`
- `#define STM32_DMA_CR_CIRC DMA_SxCR_CIRC`
- `#define STM32_DMA_CR_PINC DMA_SxCR_PINC`
- `#define STM32_DMA_CR_MINC DMA_SxCR_MINC`
- `#define STM32_DMA_CR_PSIZE_MASK DMA_SxCR_PSIZE`
- `#define STM32_DMA_CR_PSIZE_BYTE 0`
- `#define STM32_DMA_CR_PSIZE_HWORD DMA_SxCR_PSIZE_0`
- `#define STM32_DMA_CR_PSIZE_WORD DMA_SxCR_PSIZE_1`
- `#define STM32_DMA_CR_MSIZE_MASK DMA_SxCR_MSIZE`
- `#define STM32_DMA_CR_MSIZE_BYTE 0`
- `#define STM32_DMA_CR_MSIZE_HWORD DMA_SxCR_MSIZE_0`
- `#define STM32_DMA_CR_MSIZE_WORD DMA_SxCR_MSIZE_1`
- `#define STM32_DMA_CR_SIZE_MASK`
- `#define STM32_DMA_CR_PL_MASK DMA_SxCR_PL`
- `#define STM32_DMA_CR_PL(n) ((n) << 16)`

CR register constants only found in STM32F2xx/STM32F4xx

- `#define STM32_DMA_CR_DMEIE DMA_SxCR_DMEIE`
- `#define STM32_DMA_CR_PFCTRL DMA_SxCR_PFCTRL`
- `#define STM32_DMA_CR_PINCOS DMA_SxCR_PINCOS`
- `#define STM32_DMA_CR_DBM DMA_SxCR_DBM`
- `#define STM32_DMA_CR_CT DMA_SxCR_CT`
- `#define STM32_DMA_CR_PBURST_MASK DMA_SxCR_PBURST`
- `#define STM32_DMA_CR_PBURST_SINGLE 0`
- `#define STM32_DMA_CR_PBURST_INCR4 DMA_SxCR_PBURST_0`
- `#define STM32_DMA_CR_PBURST_INCR8 DMA_SxCR_PBURST_1`
- `#define STM32_DMA_CR_PBURST_INCR16 (DMA_SxCR_PBURST_0 | DMA_SxCR_PBURST_1)`
- `#define STM32_DMA_CR_MBURST_MASK DMA_SxCR_MBURST`
- `#define STM32_DMA_CR_MBURST_SINGLE 0`
- `#define STM32_DMA_CR_MBURST_INCR4 DMA_SxCR_MBURST_0`
- `#define STM32_DMA_CR_MBURST_INCR8 DMA_SxCR_MBURST_1`
- `#define STM32_DMA_CR_MBURST_INCR16 (DMA_SxCR_MBURST_0 | DMA_SxCR_MBURST_1)`
- `#define STM32_DMA_CR_CHSEL_MASK DMA_SxCR_CHSEL`
- `#define STM32_DMA_CR_CHSEL(n) ((n) << 25)`

FCR register constants only found in STM32F2xx/STM32F4xx

- #define **STM32_DMA_FCR_FEIE** DMA_SxFCR_FEIE
- #define **STM32_DMA_FCR_FS_MASK** DMA_SxFCR_FS
- #define **STM32_DMA_FCR_DMDIS** DMA_SxFCR_DMDIS
- #define **STM32_DMA_FCR_FTH_MASK** DMA_SxFCR_FTH
- #define **STM32_DMA_FCR_FTH_1Q** 0
- #define **STM32_DMA_FCR_FTH_HALF** DMA_SxFCR_FTH_0
- #define **STM32_DMA_FCR_FTH_3Q** DMA_SxFCR_FTH_1
- #define **STM32_DMA_FCR_FTH_FULL** (DMA_SxFCR_FTH_0 | DMA_SxFCR_FTH_1)

Status flags passed to the ISR callbacks

- #define **STM32_DMA_ISR_FEIF** DMA_LISR_FEIF0
- #define **STM32_DMA_ISR_DMEIF** DMA_LISR_DMEIF0
- #define **STM32_DMA_ISR_TEIF** DMA_LISR_TEIF0
- #define **STM32_DMA_ISR_HTIF** DMA_LISR_HTIF0
- #define **STM32_DMA_ISR_TCIF** DMA_LISR_TCIF0

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 **dmaStreamSetMemory1**(dmastp, addr)
Associates an alternate memory destination to a DMA stream.
- #define **dmaStreamSetTransactionSize**(dmastp, size)
Sets the number of transfers to be performed.
- #define **dmaStreamGetTransactionSize**(dmastp) ((size_t)((dmastp)->stream->NDTR))
Returns the number of transfers to be performed.
- #define **dmaStreamSetMode**(dmastp, mode)
Programs the stream mode settings.
- #define **dmaStreamSetFIFO**(dmastp, mode)
Programs the stream FIFO 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.

8.50 stm32_rcc.h File Reference

8.50.1 Detailed Description

RCC helper driver header.

Note

This file requires definitions from the ST header file `stm32f4xx.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 rccEnableAHB1(mask, lp)`
Enables the clock of one or more peripheral on the AHB1 bus.
- `#define rccDisableAHB1(mask, lp)`
Disables the clock of one or more peripheral on the AHB1 bus.
- `#define rccResetAHB1(mask)`
Resets one or more peripheral on the AHB1 bus.
- `#define rccEnableAHB2(mask, lp)`
Enables the clock of one or more peripheral on the AHB2 bus.
- `#define rccDisableAHB2(mask, lp)`
Disables the clock of one or more peripheral on the AHB2 bus.
- `#define rccResetAHB2(mask)`
Resets one or more peripheral on the AHB2 bus.
- `#define rccEnableAHB3(mask, lp)`
Enables the clock of one or more peripheral on the AHB3 (FSMC) bus.
- `#define rccDisableAHB3(mask, lp)`
Disables the clock of one or more peripheral on the AHB3 (FSMC) bus.
- `#define rccResetAHB3(mask)`
Resets one or more peripheral on the AHB3 (FSMC) 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.
- `#define rccEnableADC2(lp) rccEnableAPB2(RCC_APB2ENR_ADC2EN, lp)`
Enables the ADC2 peripheral clock.
- `#define rccDisableADC2(lp) rccDisableAPB2(RCC_APB2ENR_ADC2EN, lp)`
Disables the ADC2 peripheral clock.
- `#define rccResetADC2() rccResetAPB2(RCC_APB2RSTR_ADC2RST)`
Resets the ADC2 peripheral.
- `#define rccEnableADC3(lp) rccEnableAPB2(RCC_APB2ENR_ADC3EN, lp)`
Enables the ADC3 peripheral clock.
- `#define rccDisableADC3(lp) rccDisableAPB2(RCC_APB2ENR_ADC3EN, lp)`
Disables the ADC3 peripheral clock.
- `#define rccResetADC3() rccResetAPB2(RCC_APB2RSTR_ADC3RST)`
Resets the ADC3 peripheral.

DMA peripheral specific RCC operations

- `#define rccEnableDMA1(lp) rccEnableAHB1(RCC_AHB1ENR_DMA1EN, lp)`
Enables the DMA1 peripheral clock.
- `#define rccDisableDMA1(lp) rccDisableAHB1(RCC_AHB1ENR_DMA1EN, lp)`
Disables the DMA1 peripheral clock.
- `#define rccResetDMA1() rccResetAHB1(RCC_AHB1RSTR_DMA1RST)`
Resets the DMA1 peripheral.
- `#define rccEnableDMA2(lp) rccEnableAHB1(RCC_AHB1ENR_DMA2EN, lp)`
Enables the DMA2 peripheral clock.
- `#define rccDisableDMA2(lp) rccDisableAHB1(RCC_AHB1ENR_DMA2EN, lp)`
Disables the DMA2 peripheral clock.
- `#define rccResetDMA2() rccResetAHB1(RCC_AHB1RSTR_DMA2RST)`
Resets the DMA2 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.

CAN peripherals specific RCC operations

- `#define rccEnableCAN1(lp) rccEnableAPB1(RCC_APB1ENR_CAN1EN, lp)`
Enables the CAN1 peripheral clock.
- `#define rccDisableCAN1(lp) rccDisableAPB1(RCC_APB1ENR_CAN1EN, lp)`
Disables the CAN1 peripheral clock.
- `#define rccResetCAN1() rccResetAPB1(RCC_APB1RSTR_CAN1RST)`
Resets the CAN1 peripheral.
- `#define rccEnableCAN2(lp) rccEnableAPB1(RCC_APB1ENR_CAN2EN, lp)`
Enables the CAN2 peripheral clock.
- `#define rccDisableCAN2(lp) rccDisableAPB1(RCC_APB1ENR_CAN2EN, lp)`
Disables the CAN2 peripheral clock.
- `#define rccResetCAN2() rccResetAPB1(RCC_APB1RSTR_CAN2RST)`
Resets the CAN2 peripheral.

ETH peripheral specific RCC operations

- `#define rccEnableETH(lp)`
Enables the ETH peripheral clock.
- `#define rccDisableETH(lp)`
Disables the ETH peripheral clock.
- `#define rccResetETH() rccResetAHB1(RCC_AHB1RSTR_ETHMACRST)`
Resets the ETH peripheral.

I2C peripherals specific RCC operations

- `#define rccEnableI2C1(lp) rccEnableAPB1(RCC_APB1ENR_I2C1EN, lp)`
Enables the I2C1 peripheral clock.
- `#define rccDisableI2C1(lp) rccDisableAPB1(RCC_APB1ENR_I2C1EN, lp)`
Disables the I2C1 peripheral clock.
- `#define rccResetI2C1() rccResetAPB1(RCC_APB1RSTR_I2C1RST)`
Resets the I2C1 peripheral.
- `#define rccEnableI2C2(lp) rccEnableAPB1(RCC_APB1ENR_I2C2EN, lp)`
Enables the I2C2 peripheral clock.
- `#define rccDisableI2C2(lp) rccDisableAPB1(RCC_APB1ENR_I2C2EN, lp)`
Disables the I2C2 peripheral clock.

- `#define rccResetI2C2()` `rccResetAPB1(RCC_APB1RSTR_I2C2RST)`
Resets the I2C2 peripheral.
- `#define rccEnableI2C3(lp)` `rccEnableAPB1(RCC_APB1ENR_I2C3EN, lp)`
Enables the I2C3 peripheral clock.
- `#define rccDisableI2C3(lp)` `rccDisableAPB1(RCC_APB1ENR_I2C3EN, lp)`
Disables the I2C3 peripheral clock.
- `#define rccResetI2C3()` `rccResetAPB1(RCC_APB1RSTR_I2C3RST)`
Resets the I2C3 peripheral.

OTG peripherals specific RCC operations

- `#define rccEnableOTG_FS(lp)` `rccEnableAHB2(RCC_AHB2LPENR_OTGFSLPEN, lp)`
Enables the OTG_FS peripheral clock.
- `#define rccDisableOTG_FS(lp)` `rccEnableAHB2(RCC_AHB2LPENR_OTGFSLPEN, lp)`
Disables the OTG_FS peripheral clock.
- `#define rccResetOTG_FS()` `rccResetAHB2(RCC_AHB2RSTR_OTGFSRST)`
Resets the OTG_FS peripheral.

SDIO peripheral specific RCC operations

- `#define rccEnableSDIO(lp)` `rccEnableAPB2(RCC_APB2ENR_SDIOEN, lp)`
Enables the SDIO peripheral clock.
- `#define rccDisableSDIO(lp)` `rccDisableAPB2(RCC_APB2ENR_SDIOEN, lp)`
Disables the SDIO peripheral clock.
- `#define rccResetSDIO()` `rccResetAPB2(RCC_APB2RSTR_SDIORST)`
Resets the SDIO peripheral.

SPI peripherals specific RCC operations

- `#define rccEnableSPI1(lp)` `rccEnableAPB2(RCC_APB2ENR_SPI1EN, lp)`
Enables the SPI1 peripheral clock.
- `#define rccDisableSPI1(lp)` `rccDisableAPB2(RCC_APB2ENR_SPI1EN, lp)`
Disables the SPI1 peripheral clock.
- `#define rccResetSPI1()` `rccResetAPB2(RCC_APB2RSTR_SPI1RST)`
Resets the SPI1 peripheral.
- `#define rccEnableSPI2(lp)` `rccEnableAPB1(RCC_APB1ENR_SPI2EN, lp)`
Enables the SPI2 peripheral clock.
- `#define rccDisableSPI2(lp)` `rccDisableAPB1(RCC_APB1ENR_SPI2EN, lp)`
Disables the SPI2 peripheral clock.
- `#define rccResetSPI2()` `rccResetAPB1(RCC_APB1RSTR_SPI2RST)`
Resets the SPI2 peripheral.
- `#define rccEnableSPI3(lp)` `rccEnableAPB1(RCC_APB1ENR_SPI3EN, lp)`
Enables the SPI3 peripheral clock.
- `#define rccDisableSPI3(lp)` `rccDisableAPB1(RCC_APB1ENR_SPI3EN, lp)`
Disables the SPI3 peripheral clock.
- `#define rccResetSPI3()` `rccResetAPB1(RCC_APB1RSTR_SPI3RST)`
Resets the SPI3 peripheral.

TIM peripherals specific RCC operations

- `#define rccEnableTIM1(lp)` `rccEnableAPB2(RCC_APB2ENR_TIM1EN, lp)`
Enables the TIM1 peripheral clock.
- `#define rccDisableTIM1(lp)` `rccDisableAPB2(RCC_APB2ENR_TIM1EN, lp)`
Disables the TIM1 peripheral clock.
- `#define rccResetTIM1()` `rccResetAPB2(RCC_APB2RSTR_TIM1RST)`
Resets the TIM1 peripheral.
- `#define rccEnableTIM2(lp)` `rccEnableAPB1(RCC_APB1ENR_TIM2EN, lp)`
Enables the TIM2 peripheral clock.
- `#define rccDisableTIM2(lp)` `rccDisableAPB1(RCC_APB1ENR_TIM2EN, lp)`
Disables the TIM2 peripheral clock.

- `#define rccResetTIM2() rccResetAPB1(RCC_APB1RSTR_TIM2RST)`
Resets the TIM2 peripheral.
- `#define rccEnableTIM3(lp) rccEnableAPB1(RCC_APB1ENR_TIM3EN, lp)`
Enables the TIM3 peripheral clock.
- `#define rccDisableTIM3(lp) rccDisableAPB1(RCC_APB1ENR_TIM3EN, lp)`
Disables the TIM3 peripheral clock.
- `#define rccResetTIM3() rccResetAPB1(RCC_APB1RSTR_TIM3RST)`
Resets the TIM3 peripheral.
- `#define rccEnableTIM4(lp) rccEnableAPB1(RCC_APB1ENR_TIM4EN, lp)`
Enables the TIM4 peripheral clock.
- `#define rccDisableTIM4(lp) rccDisableAPB1(RCC_APB1ENR_TIM4EN, lp)`
Disables the TIM4 peripheral clock.
- `#define rccResetTIM4() rccResetAPB1(RCC_APB1RSTR_TIM4RST)`
Resets the TIM4 peripheral.
- `#define rccEnableTIM5(lp) rccEnableAPB1(RCC_APB1ENR_TIM5EN, lp)`
Enables the TIM5 peripheral clock.
- `#define rccDisableTIM5(lp) rccDisableAPB1(RCC_APB1ENR_TIM5EN, lp)`
Disables the TIM5 peripheral clock.
- `#define rccResetTIM5() rccResetAPB1(RCC_APB1RSTR_TIM5RST)`
Resets the TIM5 peripheral.
- `#define rccEnableTIM8(lp) rccEnableAPB2(RCC_APB2ENR_TIM8EN, lp)`
Enables the TIM8 peripheral clock.
- `#define rccDisableTIM8(lp) rccDisableAPB2(RCC_APB2ENR_TIM8EN, lp)`
Disables the TIM8 peripheral clock.
- `#define rccResetTIM8() rccResetAPB2(RCC_APB2RSTR_TIM8RST)`
Resets the TIM8 peripheral.

USART/UART peripherals specific RCC operations

- `#define rccEnableUSART1(lp) rccEnableAPB2(RCC_APB2ENR_USART1EN, lp)`
Enables the USART1 peripheral clock.
- `#define rccDisableUSART1(lp) rccDisableAPB2(RCC_APB2ENR_USART1EN, lp)`
Disables the USART1 peripheral clock.
- `#define rccResetUSART1() rccResetAPB2(RCC_APB2RSTR_USART1RST)`
Resets the USART1 peripheral.
- `#define rccEnableUSART2(lp) rccEnableAPB1(RCC_APB1ENR_USART2EN, lp)`
Enables the USART2 peripheral clock.
- `#define rccDisableUSART2(lp) rccDisableAPB1(RCC_APB1ENR_USART2EN, lp)`
Disables the USART2 peripheral clock.
- `#define rccResetUSART2() rccResetAPB1(RCC_APB1RSTR_USART2RST)`
Resets the USART2 peripheral.
- `#define rccEnableUSART3(lp) rccEnableAPB1(RCC_APB1ENR_USART3EN, lp)`
Enables the USART3 peripheral clock.
- `#define rccDisableUSART3(lp) rccDisableAPB1(RCC_APB1ENR_USART3EN, lp)`
Disables the USART3 peripheral clock.
- `#define rccResetUSART3() rccResetAPB1(RCC_APB1RSTR_USART3RST)`
Resets the USART3 peripheral.
- `#define rccEnableUSART6(lp) rccEnableAPB2(RCC_APB2ENR_USART6EN, lp)`
Enables the USART6 peripheral clock.
- `#define rccDisableUSART6(lp) rccDisableAPB2(RCC_APB2ENR_USART6EN, lp)`
Disables the USART6 peripheral clock.
- `#define rccEnableUART4(lp) rccEnableAPB1(RCC_APB1ENR_UART4EN, lp)`
Enables the UART4 peripheral clock.
- `#define rccDisableUART4(lp) rccDisableAPB1(RCC_APB1ENR_UART4EN, lp)`
Disables the UART4 peripheral clock.
- `#define rccResetUART4() rccResetAPB1(RCC_APB1RSTR_UART4RST)`
Resets the UART4 peripheral.
- `#define rccEnableUART5(lp) rccEnableAPB1(RCC_APB1ENR_UART5EN, lp)`
Enables the UART5 peripheral clock.

- `#define rccDisableUART5(ip) rccDisableAPB1(RCC_APB1ENR_UART5EN, ip)`
Disables the UART5 peripheral clock.
- `#define rccResetUART5() rccResetAPB1(RCC_APB1RSTR_UART5RST)`
Resets the UART5 peripheral.
- `#define rccResetUSART6() rccResetAPB2(RCC_APB2RSTR_USART6RST)`
Resets the USART6 peripheral.

8.51 tm.c File Reference

8.51.1 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.

8.52 tm.h File Reference

8.52.1 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.

8.53 uart.c File Reference

8.53.1 Detailed Description

UART Driver code.

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

Functions

- **void uartInit (void)**
UART Driver initialization.
- **void uartObjectInit (UARTDriver *uartp)**
Initializes the standard part of a `UARTDriver` structure.
- **void uartStart (UARTDriver *uartp, const UARTConfig *config)**
Configures and activates the UART peripheral.
- **void uartStop (UARTDriver *uartp)**
Deactivates the UART peripheral.
- **void uartStartSend (UARTDriver *uartp, size_t n, const void *txbuf)**
Starts a transmission on the UART peripheral.
- **void uartStartSendl (UARTDriver *uartp, size_t n, const void *txbuf)**
Starts a transmission on the UART peripheral.
- **size_t uartStopSend (UARTDriver *uartp)**
Stops any ongoing transmission.
- **size_t uartStopSendl (UARTDriver *uartp)**
Stops any ongoing transmission.
- **void uartStartReceive (UARTDriver *uartp, size_t n, void *rxbuf)**
Starts a receive operation on the UART peripheral.
- **void uartStartReceive1 (UARTDriver *uartp, size_t n, void *rxbuf)**
Starts a receive operation on the UART peripheral.
- **size_t uartStopReceive (UARTDriver *uartp)**
Stops any ongoing receive operation.
- **size_t uartStopReceive1 (UARTDriver *uartp)**
Stops any ongoing receive operation.

8.54 uart.h File Reference

8.54.1 Detailed Description

UART Driver macros and structures.

```
#include "uart_lld.h"
```

Functions

- void `uartInit` (void)

UART Driver initialization.
- void `uartObjectInit` (`UARTDriver` *uartp)

Initializes the standard part of a `UARTDriver` structure.
- void `uartStart` (`UARTDriver` *uartp, const `UARTConfig` *config)

Configures and activates the UART peripheral.
- void `uartStop` (`UARTDriver` *uartp)

Deactivates the UART peripheral.
- void `uartStartSend` (`UARTDriver` *uartp, size_t n, const void *txbuf)

Starts a transmission on the UART peripheral.
- void `uartStartSendl` (`UARTDriver` *uartp, size_t n, const void *txbuf)

Starts a transmission on the UART peripheral.
- size_t `uartStopSend` (`UARTDriver` *uartp)

Stops any ongoing transmission.
- size_t `uartStopSendl` (`UARTDriver` *uartp)

Stops any ongoing transmission.
- void `uartStartReceive` (`UARTDriver` *uartp, size_t n, void *rxbuf)

Starts a receive operation on the UART peripheral.
- void `uartStartReceive1` (`UARTDriver` *uartp, size_t n, void *rxbuf)

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

Stops any ongoing receive operation.
- size_t `uartStopReceive1` (`UARTDriver` *uartp)

Stops any ongoing receive operation.

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.

8.55 uart_lld.c File Reference

8.55.1 Detailed Description

STM32 low level UART driver code.

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

Functions

- **CH_IRQ_HANDLER (USART1_IRQHandler)**
USART1 IRQ handler.
- **CH_IRQ_HANDLER (USART2_IRQHandler)**
USART2 IRQ handler.
- **CH_IRQ_HANDLER (USART3_IRQHandler)**
USART3 IRQ handler.
- **void uart_lld_init (void)**
Low level UART driver initialization.
- **void uart_lld_start (UARTDriver *uartp)**
Configures and activates the UART peripheral.
- **void uart_lld_stop (UARTDriver *uartp)**
Deactivates the UART peripheral.
- **void uart_lld_start_send (UARTDriver *uartp, size_t n, const void *txbuf)**
Starts a transmission on the UART peripheral.
- **size_t uart_lld_stop_send (UARTDriver *uartp)**
Stops any ongoing transmission.
- **void uart_lld_start_receive (UARTDriver *uartp, size_t n, void *rdbuf)**
Starts a receive operation on the UART peripheral.
- **size_t uart_lld_stop_receive (UARTDriver *uartp)**
Stops any ongoing receive operation.

Variables

- **UARTDriver UARD1**
USART1 UART driver identifier.
- **UARTDriver UARD2**
USART2 UART driver identifier.
- **UARTDriver UARD3**
USART3 UART driver identifier.

8.56 uart_lld.h File Reference

8.56.1 Detailed Description

STM32 low level UART driver header.

Data Structures

- **struct UARTConfig**
Driver configuration structure.
- **struct UARTDriver**
Structure representing an UART driver.

Functions

- void `uart_ll_init` (void)

Low level UART driver initialization.
- void `uart_ll_start` (`UARTDriver` *uartp)

Configures and activates the UART peripheral.
- void `uart_ll_stop` (`UARTDriver` *uartp)

Deactivates the UART peripheral.
- void `uart_ll_start_send` (`UARTDriver` *uartp, `size_t` n, const void *txbuf)

Starts a transmission on the UART peripheral.
- `size_t` `uart_ll_stop_send` (`UARTDriver` *uartp)

Stops any ongoing transmission.
- void `uart_ll_start_receive` (`UARTDriver` *uartp, `size_t` n, void *rxbuf)

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

Stops any ongoing receive operation.

Defines

Configuration options

- `#define STM32_UART_USE_USART1 TRUE`

UART driver on USART1 enable switch.
- `#define STM32_UART_USE_USART2 TRUE`

UART driver on USART2 enable switch.
- `#define STM32_UART_USE_USART3 TRUE`

UART driver on USART3 enable switch.
- `#define STM32_UART_USART1_IRQ_PRIORITY 12`

USART1 interrupt priority level setting.
- `#define STM32_UART_USART2_IRQ_PRIORITY 12`

USART2 interrupt priority level setting.
- `#define STM32_UART_USART3_IRQ_PRIORITY 12`

USART3 interrupt priority level setting.
- `#define STM32_UART_USART1_DMA_PRIORITY 0`

USART1 DMA priority (0..3|lowest..highest).
- `#define STM32_UART_USART2_DMA_PRIORITY 0`

USART2 DMA priority (0..3|lowest..highest).
- `#define STM32_UART_USART3_DMA_PRIORITY 0`

USART3 DMA priority (0..3|lowest..highest).
- `#define STM32_UART_DMA_ERROR_HOOK(uartp) chSysHalt()`

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.

Index

_IOBUS_DATA
 PAL Driver, 153

_adc_isr_error_code
 ADC Driver, 33

_adc_isr_full_code
 ADC Driver, 32

_adc_isr_half_code
 ADC Driver, 32

_adc_isr_i
 ADC Driver, 30

_adc_reset_s
 ADC Driver, 31

_adc_timeout_isr
 ADC Driver, 32

_adc_wakeup_isr
 ADC Driver, 31

_icu_isr_invoke_period_cb
 ICU Driver, 126

_icu_isr_invoke_width_cb
 ICU Driver, 126

_pal_lld_init
 PAL Driver, 151

_pal_lld_setgroupmode
 PAL Driver, 151

_sdc_wait_for_transfer_state
 SDC Driver, 201

_serial_driver_data
 Serial Driver, 195

_serial_driver_methods
 Serial Driver, 190

_spi_isr_code
 SPI Driver, 223

_spi_wait_s
 SPI Driver, 222

_spi_wakeup_isr
 SPI Driver, 222

_stm32_dma_streams
 STM32F4xx DMA Support, 262

adc
 ADCDriver, 304

ADC Driver, 15

 _adc_isr_error_code, 33

 _adc_isr_full_code, 32

 _adc_isr_half_code, 32

 adc_reset_i, 30

 adc_reset_s, 31

 adc_timeout_isr, 32

 adc_wakeup_isr, 31

 ADC_ACTIVE, 42

 ADC_COMPLETE, 42

 ADC_ERR_DMAFAILURE, 42

 ADC_ERR_OVERFLOW, 42

 ADC_ERROR, 42

 ADC_READY, 42

 ADC_STOP, 42

 ADC_UNINIT, 41

 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_IN2, 34

 ADC_CHANNEL_IN3, 34

 ADC_CHANNEL_IN4, 34

 ADC_CHANNEL_IN5, 34

 ADC_CHANNEL_IN6, 34

 ADC_CHANNEL_IN7, 34

 ADC_CHANNEL_IN8, 34

 ADC_CHANNEL_IN9, 34

 ADC_CHANNEL_SENSOR, 35

 ADC_CHANNEL_VBAT, 35

 ADC_CHANNEL_VREFINT, 35

 adc_channels_num_t, 41

 ADC_CR2_EXTSEL_SRC, 34

 adc_lld_init, 27

 adc_lld_start, 28

 adc_lld_start_conversion, 29

 adc_lld_stop, 28

 adc_lld_stop_conversion, 29

 ADC_SAMPLE_112, 36

 ADC_SAMPLE_144, 36

 ADC_SAMPLE_15, 36

 ADC_SAMPLE_28, 36

 ADC_SAMPLE_3, 35

 ADC_SAMPLE_480, 36

 ADC_SAMPLE_56, 36

 ADC_SAMPLE_84, 36

 ADC_SMPR1_SMP_AN10, 40

 ADC_SMPR1_SMP_AN11, 40

 ADC_SMPR1_SMP_AN12, 40

 ADC_SMPR1_SMP_AN13, 40

 ADC_SMPR1_SMP_AN14, 40

 ADC_SMPR1_SMP_AN15, 40

 ADC_SMPR1_SMP_SENSOR, 40

 ADC_SMPR1_SMP_VBAT, 41

 ADC_SMPR1_SMP_VREF, 41

ADC_SMPR2_SMP_AN0, 39
ADC_SMPR2_SMP_AN1, 39
ADC_SMPR2_SMP_AN2, 39
ADC_SMPR2_SMP_AN3, 39
ADC_SMPR2_SMP_AN4, 39
ADC_SMPR2_SMP_AN5, 40
ADC_SMPR2_SMP_AN6, 40
ADC_SMPR2_SMP_AN7, 40
ADC_SMPR2_SMP_AN8, 40
ADC_SMPR2_SMP_AN9, 40
ADC_SQR1_NUM_CH, 38
ADC_SQR1_SQ13_N, 39
ADC_SQR1_SQ14_N, 39
ADC_SQR1_SQ15_N, 39
ADC_SQR1_SQ16_N, 39
ADC_SQR2_SQ10_N, 39
ADC_SQR2_SQ11_N, 39
ADC_SQR2_SQ12_N, 39
ADC_SQR2_SQ7_N, 38
ADC_SQR2_SQ8_N, 38
ADC_SQR2_SQ9_N, 38
ADC_SQR3_SQ1_N, 38
ADC_SQR3_SQ2_N, 38
ADC_SQR3_SQ3_N, 38
ADC_SQR3_SQ4_N, 38
ADC_SQR3_SQ5_N, 38
ADC_SQR3_SQ6_N, 38
ADC_USE_MUTUAL_EXCLUSION, 30
ADC_USE_WAIT, 30
adcAcquireBus, 26
adccallback_t, 41
adcConvert, 27
ADCD1, 30
ADCD2, 30
ADCD3, 30
ADCDriver, 41
adcerror_t, 42
adcerrorcallback_t, 41
adcInit, 22
adcObjectInit, 22
adcReleaseBus, 26
adcsample_t, 41
adcStart, 23
adcStartConversion, 24
adcStartConversionl, 24
adcstate_t, 41
adcSTM32DisableTSVREFE, 29
adcSTM32DisableVBATE, 30
adcSTM32EnableTSVREFE, 29
adcSTM32EnableVBATE, 29
adcStop, 23
adcStopConversion, 25
adcStopConversionl, 25
CH_IRQ_HANDLER, 27
STM32_ADC_ADC1_DMA IRQ_PRIORITY, 37
STM32_ADC_ADC1_DMA_PRIORITY, 37
STM32_ADC_ADC1_DMA_STREAM, 37
STM32_ADC_ADC2_DMA IRQ_PRIORITY, 38
STM32_ADC_ADC2_DMA_PRIORITY, 37
STM32_ADC_ADC2_DMA_STREAM, 37
STM32_ADC_ADC3_DMA IRQ_PRIORITY, 38
STM32_ADC_ADC3_DMA_PRIORITY, 37
STM32_ADC_ADC3_DMA_STREAM, 37
STM32_ADC_ADCPRE, 36
STM32_ADC_IRQ_PRIORITY, 37
STM32_ADC_USE_ADC1, 36
STM32_ADC_USE_ADC2, 36
STM32_ADC_USE_ADC3, 37
STM32_ADCCLK_MAX, 34
STM32_ADCCLK_MIN, 33
adc.c, 347
adc.h, 348
ADC1_2_3_IRQHandler
 HAL Driver, 98
ADC_ACTIVE
 ADC Driver, 42
ADC_COMPLETE
 ADC Driver, 42
ADC_ERR_DMAFAILURE
 ADC Driver, 42
ADC_ERR_OVERFLOW
 ADC Driver, 42
ADC_ERROR
 ADC Driver, 42
ADC_READY
 ADC Driver, 42
ADC_STOP
 ADC Driver, 42
ADC_UNINIT
 ADC Driver, 41
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_IN2
 ADC Driver, 34
ADC_CHANNEL_IN3
 ADC Driver, 34
ADC_CHANNEL_IN4
 ADC Driver, 34
ADC_CHANNEL_IN5
 ADC Driver, 34
ADC_CHANNEL_IN6
 ADC Driver, 34
ADC_CHANNEL_IN7
 ADC Driver, 34

ADC_CHANNEL_IN8
 ADC Driver, 34
ADC_CHANNEL_IN9
 ADC Driver, 34
ADC_CHANNEL_SENSOR
 ADC Driver, 35
ADC_CHANNEL_VBAT
 ADC Driver, 35
ADC_CHANNEL_VREFINT
 ADC Driver, 35
adc_channels_num_t
 ADC Driver, 41
ADC_CR2_EXTSEL_SRC
 ADC Driver, 34
adc_lld.c, 349
adc_lld.h, 350
adc_lld_init
 ADC Driver, 27
adc_lld_start
 ADC Driver, 28
adc_lld_start_conversion
 ADC Driver, 29
adc_lld_stop
 ADC Driver, 28
adc_lld_stop_conversion
 ADC Driver, 29
ADC_SAMPLE_112
 ADC Driver, 36
ADC_SAMPLE_144
 ADC Driver, 36
ADC_SAMPLE_15
 ADC Driver, 36
ADC_SAMPLE_28
 ADC Driver, 36
ADC_SAMPLE_3
 ADC Driver, 35
ADC_SAMPLE_480
 ADC Driver, 36
ADC_SAMPLE_56
 ADC Driver, 36
ADC_SAMPLE_84
 ADC Driver, 36
ADC_SMPR1_SMP_AN10
 ADC Driver, 40
ADC_SMPR1_SMP_AN11
 ADC Driver, 40
ADC_SMPR1_SMP_AN12
 ADC Driver, 40
ADC_SMPR1_SMP_AN13
 ADC Driver, 40
ADC_SMPR1_SMP_AN14
 ADC Driver, 40
ADC_SMPR1_SMP_AN15
 ADC Driver, 40
ADC_SMPR1_SMP_SENSOR
 ADC Driver, 40
ADC_SMPR1_SMP_VBAT
 ADC Driver, 41
ADC_SMPR1_SMP_VREF
 ADC Driver, 41
ADC_SMPR2_SMP_AN0
 ADC Driver, 39
ADC_SMPR2_SMP_AN1
 ADC Driver, 39
ADC_SMPR2_SMP_AN2
 ADC Driver, 39
ADC_SMPR2_SMP_AN3
 ADC Driver, 39
ADC_SMPR2_SMP_AN4
 ADC Driver, 39
ADC_SMPR2_SMP_AN5
 ADC Driver, 40
ADC_SMPR2_SMP_AN6
 ADC Driver, 40
ADC_SMPR2_SMP_AN7
 ADC Driver, 40
ADC_SMPR2_SMP_AN8
 ADC Driver, 40
ADC_SMPR2_SMP_AN9
 ADC Driver, 40
ADC_SQR1_NUM_CH
 ADC Driver, 38
ADC_SQR1_SQ13_N
 ADC Driver, 39
ADC_SQR1_SQ14_N
 ADC Driver, 39
ADC_SQR1_SQ15_N
 ADC Driver, 39
ADC_SQR1_SQ16_N
 ADC Driver, 39
ADC_SQR2_SQ10_N
 ADC Driver, 39
ADC_SQR2_SQ11_N
 ADC Driver, 39
ADC_SQR2_SQ12_N
 ADC Driver, 39
ADC_SQR2_SQ7_N
 ADC Driver, 39
ADC_SQR3_SQ1_N
 ADC Driver, 38
ADC_SQR3_SQ2_N
 ADC Driver, 38
ADC_SQR2_SQ8_N
 ADC Driver, 38
ADC_SQR2_SQ9_N
 ADC Driver, 38
ADC_SQR3_SQ3_N
 ADC Driver, 38
ADC_SQR3_SQ4_N
 ADC Driver, 38
ADC_SQR3_SQ5_N
 ADC Driver, 38
ADC_SQR3_SQ6_N
 ADC Driver, 38
ADC_USE_MUTUAL_EXCLUSION
 ADC Driver, 30
 Configuration, 13
ADC_USE_WAIT

ADC Driver, 30
Configuration, 12
adcAcquireBus
 ADC Driver, 26
adccallback_t
 ADC Driver, 41
ADCCfg, 299
ADCCConversionGroup, 299
 circular, 301
 cr1, 301
 cr2, 301
 end_cb, 301
 error_cb, 301
 num_channels, 301
 smpr1, 301
 smpr2, 301
 sqr1, 302
 sqr2, 302
 sqr3, 302
adcConvert
 ADC Driver, 27
ADCD1
 ADC Driver, 30
ADCD2
 ADC Driver, 30
ADCD3
 ADC Driver, 30
ADCDriver, 302
 adc, 304
 ADC Driver, 41
 config, 304
 depth, 304
 dmamode, 305
 dmastp, 305
 grpp, 304
 mutex, 304
 samples, 304
 state, 304
 thread, 304
adcerror_t
 ADC Driver, 42
adcerrorcallback_t
 ADC Driver, 41
adclInit
 ADC Driver, 22
adcObjectInit
 ADC Driver, 22
adcReleaseBus
 ADC Driver, 26
adcsample_t
 ADC Driver, 41
adcStart
 ADC Driver, 23
adcStartConversion
 ADC Driver, 24
adcStartConversionl
 ADC Driver, 24
adcstate_t
 ADC Driver, 41

adcSTM32DisableTSVREFE
 ADC Driver, 29
adcSTM32DisableVBATE
 ADC Driver, 30
adcSTM32EnableTSVREFE
 ADC Driver, 29
adcSTM32EnableVBATE
 ADC Driver, 29
adcStop
 ADC Driver, 23
adcStopConversion
 ADC Driver, 25
adcStopConversionl
 ADC Driver, 25
afrh
 stm32_gpio_setup_t, 339
afrl
 stm32_gpio_setup_t, 339

bdtr
 PWMConfig, 328
best
 TimeMeasurement, 340

callback
 GPTConfig, 311
 PWMChannelConfig, 326
 PWMConfig, 328

CAN Driver, 42
 CAN_READY, 48
 CAN_SLEEP, 48
 CAN_STARTING, 48
 CAN_STOP, 48
 CAN_UNINIT, 48
 CAN_BUS_OFF_ERROR, 47
 CAN_FRAMING_ERROR, 47
 CAN_LIMIT_ERROR, 47
 CAN_LIMIT_WARNING, 47
 CAN_OVERFLOW_ERROR, 47
 CAN_USE_SLEEP_MODE, 47
canAddFlagsI, 47
canGetAndClearFlags, 46
canInit, 44
canObjectInit, 44
canReceive, 45
canSleep, 46
canStart, 44
canstate_t, 48
canStop, 45
canTransmit, 45
canWakeup, 47

can.c, 354
can.h, 355
CAN1_RX0_IRQHandler
 HAL Driver, 98
CAN1_RX1_IRQHandler
 HAL Driver, 98
CAN1_SCE_IRQHandler
 HAL Driver, 98

CAN1_TX_IRQHandler
 HAL Driver, 98
CAN2_RX0_IRQHandler
 HAL Driver, 102
CAN2_RX1_IRQHandler
 HAL Driver, 102
CAN2_SCE_IRQHandler
 HAL Driver, 102
CAN2_TX_IRQHandler
 HAL Driver, 101
CAN_READY
 CAN Driver, 48
CAN_SLEEP
 CAN Driver, 48
CAN_STARTING
 CAN Driver, 48
CAN_STOP
 CAN Driver, 48
CAN_UNINIT
 CAN Driver, 48
CAN_BUS_OFF_ERROR
 CAN Driver, 47
CAN_FRAMING_ERROR
 CAN Driver, 47
CAN_LIMIT_ERROR
 CAN Driver, 47
CAN_LIMIT_WARNING
 CAN Driver, 47
CAN_OVERFLOW_ERROR
 CAN Driver, 47
CAN_USE_SLEEP_MODE
 CAN Driver, 47
 Configuration, 13
canAddFlags()
 CAN Driver, 47
canGetAndClearFlags()
 CAN Driver, 46
canInit()
 CAN Driver, 44
canObjectInit()
 CAN Driver, 44
canReceive()
 CAN Driver, 45
canSleep()
 CAN Driver, 46
canStart()
 CAN Driver, 44
canstate_t
 CAN Driver, 48
canStop()
 CAN Driver, 45
canTransmit()
 CAN Driver, 45
canWakeup()
 CAN Driver, 47
cb
 EXTChannelConfig, 306
CH_IRQ_HANDLER
 ADC Driver, 27
EXT Driver, 53–55
Serial Driver, 187, 188
STM32F4xx DMA Support, 259–261
UART Driver, 240
channels
 EXTConfig, 307
 PWMConfig, 328
circular
 ADCConversionGroup, 301
clock
 GPTDriver, 313
 ICUDriver, 317
 PWMDriver, 331
cnt
 MMCDriver, 322
config
 ADCDriver, 304
 EXTDriver, 308
 GPTDriver, 313
 ICUDriver, 317
 MMCDriver, 321
 PWMDriver, 331
 SPIDriver, 337
 UARTDriver, 345
Configuration, 9
 ADC_USE_MUTUAL_EXCLUSION, 13
 ADC_USE_WAIT, 12
 CAN_USE_SLEEP_MODE, 13
 HAL_USE_ADC, 11
 HAL_USE_CAN, 11
 HAL_USE_EXT, 11
 HAL_USE_GPT, 11
 HAL_USE_I2C, 11
 HAL_USE_ICU, 12
 HAL_USE_MAC, 12
 HAL_USE_MMCSPI, 12
 HAL_USE_PAL, 11
 HAL_USE_PWM, 12
 HAL_USE_RTC, 12
 HAL_USE_SDC, 12
 HAL_USE_SERIAL, 12
 HAL_USE_SERIALUSB, 12
 HAL_USE_SPI, 12
 HAL_USE_UART, 12
 HAL_USE_USB, 12
 I2C_USE_MUTUAL_EXCLUSION, 13
 MAC_USE_EVENTS, 13
 MMC_NICE_WAITING, 13
 MMC_POLLING_DELAY, 13
 MMC_POLLING_INTERVAL, 13
 MMC_SECTOR_SIZE, 13
 MMC_USE_SPI_POLLING, 13
 SDC_INIT_RETRY, 13
 SDC_MMCSUPPORT, 14
 SDC_NICE_WAITING, 14
 SERIAL_BUFFERS_SIZE, 14
 SERIAL_DEFAULT_BITRATE, 14
 SERIAL_USB_BUFFERS_SIZE, 14
 SPI_USE_MUTUAL_EXCLUSION, 14

SPI_USE_WAIT, 14
cr1
 ADCConversionGroup, 301
 SPIConfig, 335
 UARTConfig, 342
cr2
 ADCConversionGroup, 301
 PWMConfig, 328
 UARTConfig, 342
cr3
 UARTConfig, 343
CRYP_IRQHandler
 HAL Driver, 103
DCMI_IRQHandler
 HAL Driver, 103
depth
 ADCDriver, 304
DMA1_Stream0_IRQHandler
 HAL Driver, 97
DMA1_Stream1_IRQHandler
 HAL Driver, 97
DMA1_Stream2_IRQHandler
 HAL Driver, 97
DMA1_Stream3_IRQHandler
 HAL Driver, 97
DMA1_Stream4_IRQHandler
 HAL Driver, 97
DMA1_Stream5_IRQHandler
 HAL Driver, 98
DMA1_Stream6_IRQHandler
 HAL Driver, 98
DMA1_Stream7_IRQHandler
 HAL Driver, 100
DMA2_Stream0_IRQHandler
 HAL Driver, 101
DMA2_Stream1_IRQHandler
 HAL Driver, 101
DMA2_Stream2_IRQHandler
 HAL Driver, 101
DMA2_Stream3_IRQHandler
 HAL Driver, 101
DMA2_Stream4_IRQHandler
 HAL Driver, 101
DMA2_Stream5_IRQHandler
 HAL Driver, 102
DMA2_Stream6_IRQHandler
 HAL Driver, 102
DMA2_Stream7_IRQHandler
 HAL Driver, 102
dmalinit
 STM32F4xx DMA Support, 261
dmamode
 ADCDriver, 305
 UARTDriver, 345
dmarx
 SPIDriver, 337
 UARTDriver, 345
dmaStartMemcpy

 STM32F4xx DMA Support, 269
dmastp
 ADCDriver, 305
dmaStreamAllocate
 STM32F4xx DMA Support, 261
dmaStreamClearInterrupt
 STM32F4xx DMA Support, 269
dmaStreamDisable
 STM32F4xx DMA Support, 268
dmaStreamEnable
 STM32F4xx DMA Support, 268
dmaStreamGetTransactionSize
 STM32F4xx DMA Support, 266
dmaStreamRelease
 STM32F4xx DMA Support, 262
dmaStreamSetFIFO
 STM32F4xx DMA Support, 267
dmaStreamSetMemory0
 STM32F4xx DMA Support, 265
dmaStreamSetMemory1
 STM32F4xx DMA Support, 265
dmaStreamSetMode
 STM32F4xx DMA Support, 267
dmaStreamSetPeripheral
 STM32F4xx DMA Support, 264
dmaStreamSetTransactionSize
 STM32F4xx DMA Support, 266
dmatx
 SPIDriver, 337
 UARTDriver, 345
dmaWaitCompletion
 STM32F4xx DMA Support, 270
end_cb
 ADCConversionGroup, 301
 SPIConfig, 335
error_cb
 ADCConversionGroup, 301
ETH_IRQHandler
 HAL Driver, 101
ETH_WKUP_IRQHandler
 HAL Driver, 101
expchannel_t
 EXT Driver, 59
EXT Driver, 48
 CH_IRQ_HANDLER, 53–55
 expchannel_t, 59
 EXT_CHANNELS_MASK, 56
 ext_lld_channel_disable, 56
 ext_lld_channel_enable, 56
 ext_lld_init, 55
 ext_lld_start, 55
 ext_lld_stop, 55
 EXT_MAX_CHANNELS, 56
 EXT_MODE_EXTI, 56
 EXT_MODE_GPIOA, 57
 EXT_MODE_GPIOB, 57
 EXT_MODE_GPIOC, 57
 EXT_MODE_GPIOD, 57

EXT_MODE_GPIOE, 57
EXT_MODE_GPIOF, 57
EXT_MODE_GPIOG, 57
EXT_MODE_GPIOH, 57
EXT_MODE_GPIOI, 57
extcallback_t, 59
extChannelDisable, 53
extChannelEnable, 53
EXTD1, 56
extInit, 51
extObjectInit, 52
extStart, 52
extStop, 52
STM32_EXT EXTI0_IRQ_PRIORITY, 57
STM32_EXT EXTI10_15_IRQ_PRIORITY, 58
STM32_EXT EXTI16_IRQ_PRIORITY, 58
STM32_EXT EXTI17_IRQ_PRIORITY, 58
STM32_EXT EXTI18_IRQ_PRIORITY, 58
STM32_EXT EXTI19_IRQ_PRIORITY, 58
STM32_EXT EXTI11_IRQ_PRIORITY, 57
STM32_EXT EXTI20_IRQ_PRIORITY, 58
STM32_EXT EXTI21_IRQ_PRIORITY, 58
STM32_EXT EXTI22_IRQ_PRIORITY, 58
STM32_EXT EXTI2_IRQ_PRIORITY, 57
STM32_EXT EXTI3_IRQ_PRIORITY, 58
STM32_EXT EXTI4_IRQ_PRIORITY, 58
STM32_EXT EXTI5_9_IRQ_PRIORITY, 58
ext.c, 356
EXT_CHANNELS_MASK
 EXT Driver, 56
ext_lld.c, 356
ext_lld.h, 357
ext_lld_channel_disable
 EXT Driver, 56
ext_lld_channel_enable
 EXT Driver, 56
ext_lld_init
 EXT Driver, 55
ext_lld_start
 EXT Driver, 55
ext_lld_stop
 EXT Driver, 55
EXT_MAX_CHANNELS
 EXT Driver, 56
EXT_MODE_EXTI
 EXT Driver, 56
EXT_MODE_GPIOA
 EXT Driver, 57
EXT_MODE_GPIOB
 EXT Driver, 57
EXT_MODE_GPIOC
 EXT Driver, 57
EXT_MODE_GPIOD
 EXT Driver, 57
EXT_MODE_GPIOE
 EXT Driver, 57
EXT_MODE_GPIOF
 EXT Driver, 57
EXT_MODE_GPIOG
 EXT Driver, 57
EXT Driver, 57
EXT MODE GPIOH
 EXT Driver, 57
EXT MODE GPIOI
 EXT Driver, 57
extcallback_t
 EXT Driver, 59
extChannelDisable
 EXT Driver, 53
extChannelEnable
 EXT Driver, 53
EXTConfig, 306
 channels, 307
 exti, 307
EXTD1
 EXT Driver, 56
EXTDriver, 308
 config, 308
 state, 308
exti
 EXTConfig, 307
EXTI0_IRQHandler
 HAL Driver, 97
EXTI15_10_IRQHandler
 HAL Driver, 100
EXTI1_IRQHandler
 HAL Driver, 97
EXTI2_IRQHandler
 HAL Driver, 97
EXTI3_IRQHandler
 HAL Driver, 97
EXTI4_IRQHandler
 HAL Driver, 97
EXTI9_5_IRQHandler
 HAL Driver, 98
extInit
 EXT Driver, 51
extObjectInit
 EXT Driver, 52
extStart
 EXT Driver, 52
extStop
 EXT Driver, 52
FLASH_IRQHandler
 HAL Driver, 97
FPU_IRQHandler
 HAL Driver, 103
frequency
 GPTConfig, 311
 ICUConfig, 315
 PWMConfig, 328
FSMC_IRQHandler
 HAL Driver, 100
GPIO_TypeDef, 309

GPT Driver, 59
 GPT_CONTINUOUS, 72
 GPT_ONESHOT, 72
 GPT_READY, 72
 GPT_STOP, 72
 GPT_UNINIT, 72
 gpt_lld_init, 68
 gpt_lld_polled_delay, 69
 gpt_lld_start, 68
 gpt_lld_start_timer, 68
 gpt_lld_stop, 68
 gpt_lld_stop_timer, 69
 gptcallback_t, 72
 gptcnt_t, 72
 GPTD1, 69
 GPTD2, 69
 GPTD3, 69
 GPTD4, 70
 GPTD5, 70
 GPTD8, 70
 GPTDriver, 72
 gptfreq_t, 72
 gptInit, 63
 gptObjectInit, 63
 gptPolledDelay, 67
 gptStart, 63
 gptStartContinuous, 64
 gptStartContinuousl, 65
 gptStartOneShot, 65
 gptStartOneShotl, 66
 gptstate_t, 72
 gptStop, 64
 gptStopTimer, 66
 gptStopTimerl, 67
 STM32_GPT_TIM1_IRQ_PRIORITY, 71
 STM32_GPT_TIM2_IRQ_PRIORITY, 71
 STM32_GPT_TIM3_IRQ_PRIORITY, 71
 STM32_GPT_TIM4_IRQ_PRIORITY, 71
 STM32_GPT_TIM5_IRQ_PRIORITY, 71
 STM32_GPT_TIM8_IRQ_PRIORITY, 72
 STM32_GPT_USE_TIM1, 70
 STM32_GPT_USE_TIM2, 70
 STM32_GPT_USE_TIM3, 70
 STM32_GPT_USE_TIM4, 71
 STM32_GPT_USE_TIM5, 71
 STM32_GPT_USE_TIM8, 71
gpt.c, 359
gpt.h, 359
GPT_CONTINUOUS
 GPT Driver, 72
GPT_ONESHOT
 GPT Driver, 72
GPT_READY
 GPT Driver, 72
GPT_STOP
 GPT Driver, 72
GPT_UNINIT
 GPT Driver, 72
gpt_lld.c, 360
gpt_lld.h, 361
gpt_lld_init
 GPT Driver, 68
gpt_lld_polled_delay
 GPT Driver, 69
gpt_lld_start
 GPT Driver, 68
gpt_lld_start_timer
 GPT Driver, 68
gpt_lld_stop
 GPT Driver, 68
gpt_lld_stop_timer
 GPT Driver, 69
gptcallback_t
 GPT Driver, 72
gptcnt_t
 GPT Driver, 72
GPTConfig, 309
 callback, 311
 frequency, 311
GPTD1
 GPT Driver, 69
GPTD2
 GPT Driver, 69
GPTD3
 GPT Driver, 69
GPTD4
 GPT Driver, 70
GPTD5
 GPT Driver, 70
GPTD8
 GPT Driver, 70
GPTDriver, 311
 clock, 313
 config, 313
 GPT Driver, 72
 state, 313
 tim, 313
gptfreq_t
 GPT Driver, 72
gptInit
 GPT Driver, 63
gptObjectInit
 GPT Driver, 63
gptPolledDelay
 GPT Driver, 67
gptStart
 GPT Driver, 63
gptStartContinuous
 GPT Driver, 64
gptStartContinuousl
 GPT Driver, 65
gptStartOneShot
 GPT Driver, 65
gptStartOneShotl
 GPT Driver, 66
gptstate_t
 GPT Driver, 72
gptStop

GPT Driver, 64
gptStopTimer
 GPT Driver, 66
gptStopTimerl
 GPT Driver, 67
grpp
 ADCDriver, 304

HAL, 7
HAL Driver, 72
 ADC1_2_3_IRQHandler, 98
 CAN1_RX0_IRQHandler, 98
 CAN1_RX1_IRQHandler, 98
 CAN1_SCE_IRQHandler, 98
 CAN1_TX_IRQHandler, 98
 CAN2_RX0_IRQHandler, 102
 CAN2_RX1_IRQHandler, 102
 CAN2_SCE_IRQHandler, 102
 CAN2_TX_IRQHandler, 101
 CRYP_IRQHandler, 103
 DCMI_IRQHandler, 103
 DMA1_Stream0_IRQHandler, 97
 DMA1_Stream1_IRQHandler, 97
 DMA1_Stream2_IRQHandler, 97
 DMA1_Stream3_IRQHandler, 97
 DMA1_Stream4_IRQHandler, 97
 DMA1_Stream5_IRQHandler, 98
 DMA1_Stream6_IRQHandler, 98
 DMA1_Stream7_IRQHandler, 100
 DMA2_Stream0_IRQHandler, 101
 DMA2_Stream1_IRQHandler, 101
 DMA2_Stream2_IRQHandler, 101
 DMA2_Stream3_IRQHandler, 101
 DMA2_Stream4_IRQHandler, 101
 DMA2_Stream5_IRQHandler, 102
 DMA2_Stream6_IRQHandler, 102
 DMA2_Stream7_IRQHandler, 102
 ETH_IRQHandler, 101
 ETH_WKUP_IRQHandler, 101
 EXTI0_IRQHandler, 97
 EXTI15_10_IRQHandler, 100
 EXTI1_IRQHandler, 97
 EXTI2_IRQHandler, 97
 EXTI3_IRQHandler, 97
 EXTI4_IRQHandler, 97
 EXTI9_5_IRQHandler, 98
 FLASH_IRQHandler, 97
 FPU_IRQHandler, 103
 FSMC_IRQHandler, 100
 HAL_IMPLEMENTS_COUNTERS, 88
 hal_lld_get_counter_frequency, 109
 hal_lld_get_counter_value, 109
 hal_lld_init, 86
 halclock_t, 110
 halGetCounterFrequency, 88
 halGetCounterValue, 87
 hallInit, 83
 hallsCounterWithin, 84
 halPolledDelay, 85

 halrcnt_t, 110
 HASH_RNG_IRQHandler, 103
 I2C1_ER_IRQHandler, 99
 I2C1_EV_IRQHandler, 99
 I2C2_ER_IRQHandler, 99
 I2C2_EV_IRQHandler, 99
 I2C3_ER_IRQHandler, 102
 I2C3_EV_IRQHandler, 102
 MS2RTT, 87
 OTG_FS_IRQHandler, 102
 OTG_FS_WKUP_IRQHandler, 100
 OTG_HS_EP1_IN_IRQHandler, 102
 OTG_HS_EP1_OUT_IRQHandler, 102
 OTG_HS_IRQHandler, 103
 OTG_HS_WKUP_IRQHandler, 103
 PVD_IRQHandler, 96
 RCC_IRQHandler, 97
 RTC_Alarm_IRQHandler, 100
 RTC_WKUP_IRQHandler, 96
 S2RTT, 86
 SDIO_IRQHandler, 100
 SPI1_IRQHandler, 99
 SPI2_IRQHandler, 99
 SPI3_IRQHandler, 100
 STM32_0WS_THRESHOLD, 106
 STM32_ACTIVATE_PLL, 107
 STM32_ACTIVATE_PLLI2S, 108
 STM32_CLOCK48_REQUIRED, 104
 stm32_clock_init, 86
 STM32_FLASHBITS, 109
 STM32_HCLK, 107
 STM32_HPRE, 105
 STM32_HPRE_DIV1, 91
 STM32_HPRE_DIV128, 92
 STM32_HPRE_DIV16, 92
 STM32_HPRE_DIV2, 91
 STM32_HPRE_DIV256, 92
 STM32_HPRE_DIV4, 92
 STM32_HPRE_DIV512, 92
 STM32_HPRE_DIV64, 92
 STM32_HPRE_DIV8, 92
 STM32_HPRE_MASK, 91
 STM32_HSE_ENABLED, 104
 STM32_HSECLK_MAX, 88
 STM32_HSECLK_MIN, 88
 STM32_HSEDIVCLK, 109
 STM32_HSI_ENABLED, 103
 STM32_HSICLK, 89
 STM32_I2SSRC, 106
 STM32_I2SSRC_CKIN, 94
 STM32_I2SSRC_MASK, 94
 STM32_I2SSRC_PLLI2S, 94
 STM32_LSE_ENABLED, 104
 STM32_LSECLK_MAX, 88
 STM32_LSECLK_MIN, 88
 STM32_LSI_ENABLED, 104
 STM32_LSICLK, 89
 STM32_MCO1CLK, 108
 STM32_MCO1DIVCLK, 108

STM32_MCO1PRE, 105
STM32_MCO1PRE_DIV1, 94
STM32_MCO1PRE_DIV2, 94
STM32_MCO1PRE_DIV3, 94
STM32_MCO1PRE_DIV4, 94
STM32_MCO1PRE_DIV5, 94
STM32_MCO1PRE_MASK, 94
STM32_MCO1SEL, 105
STM32_MCO1SEL_HSE, 93
STM32_MCO1SEL_HSI, 93
STM32_MCO1SEL_LSE, 93
STM32_MCO1SEL_MASK, 93
STM32_MCO1SEL_PLL, 94
STM32_MCO2CLK, 108
STM32_MCO2DIVCLK, 108
STM32_MCO2PRE, 106
STM32_MCO2PRE_DIV1, 94
STM32_MCO2PRE_DIV2, 95
STM32_MCO2PRE_DIV3, 95
STM32_MCO2PRE_DIV4, 95
STM32_MCO2PRE_DIV5, 95
STM32_MCO2PRE_MASK, 94
STM32_MCO2SEL, 106
STM32_MCO2SEL_HSE, 95
STM32_MCO2SEL_MASK, 95
STM32_MCO2SEL_PLL, 95
STM32_MCO2SEL_PLLI2S, 95
STM32_MCO2SEL_SYSCLK, 95
STM32_NO_INIT, 103
STM32_PCLK1, 107
STM32_PCLK1_MAX, 89
STM32_PCLK2, 108
STM32_PCLK2_MAX, 89
STM32_PLL48CLK, 109
STM32_PLLCLKIN, 107
STM32_PLLCLKOUT, 107
STM32_PLLI2SCLKOUT, 108
STM32_PLLI2SN, 108
STM32_PLLI2SN_MASK, 96
STM32_PLLI2SN_VALUE, 106
STM32_PLLI2SR, 108
STM32_PLLI2SR_MASK, 96
STM32_PLLI2SR_VALUE, 106
STM32_PLLI2SVCO, 108
STM32_PLLIN_MAX, 88
STM32_PLLIN_MIN, 89
STM32_PLLM, 107
STM32_PLLM_VALUE, 104
STM32_PLLN, 107
STM32_PLLN_VALUE, 104
STM32_PLLOUT_MAX, 89
STM32_PLLOUT_MIN, 89
STM32_PLLP, 107
STM32_PLLP_DIV2, 90
STM32_PLLP_DIV4, 91
STM32_PLLP_DIV6, 91
STM32_PLLP_DIV8, 91
STM32_PLLP_MASK, 90
STM32_PLLP_VALUE, 104
STM32_PLLQ, 107
STM32_PLLQ_VALUE, 105
STM32_PLLSRC, 104
STM32_PLLSRC_HSE, 91
STM32_PLLSRC_HSI, 91
STM32_PLLVCO, 107
STM32_PLLVCO_MAX, 89
STM32_PLLVCO_MIN, 89
STM32_PLS, 103
STM32_PLS_LEV0, 90
STM32_PLS_LEV1, 90
STM32_PLS_LEV2, 90
STM32_PLS_LEV3, 90
STM32_PLS_LEV4, 90
STM32_PLS_LEV5, 90
STM32_PLS_LEV6, 90
STM32_PLS_LEV7, 90
STM32_PLS_MASK, 90
STM32_PPREG1, 105
STM32_PPREG1_DIV1, 92
STM32_PPREG1_DIV16, 93
STM32_PPREG1_DIV2, 92
STM32_PPREG1_DIV4, 92
STM32_PPREG1_DIV8, 92
STM32_PPREG1_MASK, 92
STM32_PPREG2, 105
STM32_PPREG2_DIV1, 93
STM32_PPREG2_DIV16, 93
STM32_PPREG2_DIV2, 93
STM32_PPREG2_DIV4, 93
STM32_PPREG2_DIV8, 93
STM32_PPREG2_MASK, 93
STM32_PVD_ENABLE, 103
STM32_RTC_HSE, 96
STM32_RTC_LSE, 95
STM32_RTC_LSI, 95
STM32_RTC_NO_CLOCK, 95
STM32_RTCCLK, 109
STM32_RTCPRE, 108
STM32_RTCPRE_MASK, 93
STM32_RTCPRE_VALUE, 105
STM32_RTCSEL, 105
STM32_RTCSEL_HSEDIV, 96
STM32_RTCSEL_LSE, 96
STM32_RTCSEL_LSI, 96
STM32_RTCSEL_MASK, 96
STM32_RTCSEL_NO_CLOCK, 96
STM32_SPII2S_MAX, 89
STM32_SW, 104
STM32_SW_HSE, 91
STM32_SW_HSI, 91
STM32_SW_MASK, 91
STM32_SW_PLL, 91
STM32_SYSCLK, 107
STM32_SYSCLK_MAX, 106
STM32_TIMCLK1, 109
STM32_TIMCLK2, 109
STM32_VOS, 103
STM32_VOS_HIGH, 90

STM32_VOS_LOW, 89
STM32_VOS_MASK, 89
TAMP_STAMP_IRQHandler, 96
TIM1_BRK_IRQHandler, 98
TIM1_CC_IRQHandler, 98
TIM1_TRG_COM_IRQHandler, 98
TIM1_UP_IRQHandler, 98
TIM2_IRQHandler, 99
TIM3_IRQHandler, 99
TIM4_IRQHandler, 99
TIM5_IRQHandler, 100
TIM6_IRQHandler, 101
TIM7_IRQHandler, 101
TIM8_BRK_IRQHandler, 100
TIM8_CC_IRQHandler, 100
TIM8_TRG_COM_IRQHandler, 100
TIM8_UP_IRQHandler, 100
UART4_IRQHandler, 101
UART5_IRQHandler, 101
US2RTT, 87
USART1_IRQHandler, 99
USART2_IRQHandler, 99
USART3_IRQHandler, 99
USART6_IRQHandler, 102
WWDG_IRQHandler, 96
hal.c, 362
hal.h, 363
HAL_IMPLEMENTANTS_COUNTERS
 HAL Driver, 88
hal_lld.c, 364
hal_lld.h, 364
hal_lld_get_counter_frequency
 HAL Driver, 109
hal_lld_get_counter_value
 HAL Driver, 109
hal_lld_init
 HAL Driver, 86
HAL_USE_ADC
 Configuration, 11
HAL_USE_CAN
 Configuration, 11
HAL_USE_EXT
 Configuration, 11
HAL_USE_GPT
 Configuration, 11
HAL_USE_I2C
 Configuration, 11
HAL_USE_ICU
 Configuration, 12
HAL_USE_MAC
 Configuration, 12
HAL_USE_MMC_SPI
 Configuration, 12
HAL_USE_PAL
 Configuration, 11
HAL_USE_PWM
 Configuration, 12
HAL_USE_RTC
 Configuration, 12
HAL_USE_SDC
 Configuration, 12
HAL_USE_SERIAL
 Configuration, 12
HAL_USE_SERIAL_USB
 Configuration, 12
HAL_USE_SPI
 Configuration, 12
HAL_USE_UART
 Configuration, 12
HAL_USE_USB
 Configuration, 12
halclock_t
 HAL Driver, 110
halconf.h, 372
halGetCounterFrequency
 HAL Driver, 88
halGetCounterValue
 HAL Driver, 87
hallinit
 HAL Driver, 83
hallsCounterWithin
 HAL Driver, 84
halPolledDelay
 HAL Driver, 85
halrtcnt_t
 HAL Driver, 110
HASH_RNG_IRQHandler
 HAL Driver, 103
hscfg
 MMCDriver, 321
I2C Driver, 110
 I2C_ACTIVE_RX, 116
 I2C_ACTIVE_TX, 116
 I2C_READY, 116
 I2C_STOP, 116
 I2C_UNINIT, 116
 I2C_USE_MUTUAL_EXCLUSION, 116
 i2cAcquireBus, 114
 I2CD_ACK_FAILURE, 115
 I2CD_ARBITRATION_LOST, 115
 I2CD_BUS_ERROR, 115
 I2CD_NO_ERROR, 115
 I2CD_OVERRUN, 115
 I2CD_PEC_ERROR, 115
 I2CD_SMB_ALERT, 116
 I2CD_TIMEOUT, 115
 i2cGetErrors, 113
 i2cInit, 112
 i2cMasterReceive, 116
 i2cMasterReceiveTimeout, 114
 i2cMasterTransmit, 116
 i2cMasterTransmitTimeout, 113
 i2cObjectInit, 112
 i2cReleaseBus, 115
 i2cStart, 113
 i2cstate_t, 116
 i2cStop, 113

i2c.c, 374
i2c.h, 375
I2C1_ER_IRQHandler
 HAL Driver, 99
I2C1_EV_IRQHandler
 HAL Driver, 99
I2C2_ER_IRQHandler
 HAL Driver, 99
I2C2_EV_IRQHandler
 HAL Driver, 99
I2C3_ER_IRQHandler
 HAL Driver, 102
I2C3_EV_IRQHandler
 HAL Driver, 102
I2C_ACTIVE_RX
 I2C Driver, 116
I2C_ACTIVE_TX
 I2C Driver, 116
I2C_READY
 I2C Driver, 116
I2C_STOP
 I2C Driver, 116
I2C_UNINIT
 I2C Driver, 116
I2C_USE_MUTUAL_EXCLUSION
 Configuration, 13
 I2C Driver, 116
i2cAcquireBus
 I2C Driver, 114
I2CD_ACK_FAILURE
 I2C Driver, 115
I2CD_ARBITRATION_LOST
 I2C Driver, 115
I2CD_BUS_ERROR
 I2C Driver, 115
I2CD_NO_ERROR
 I2C Driver, 115
I2CD_OVERRUN
 I2C Driver, 115
I2CD_PEC_ERROR
 I2C Driver, 115
I2CD_SMB_ALERT
 I2C Driver, 116
I2CD_TIMEOUT
 I2C Driver, 115
i2cGetErrors
 I2C Driver, 113
i2cInit
 I2C Driver, 112
i2cMasterReceive
 I2C Driver, 116
i2cMasterReceiveTimeout
 I2C Driver, 114
i2cMasterTransmit
 I2C Driver, 116
i2cMasterTransmitTimeout
 I2C Driver, 113
i2cObjectInit
 I2C Driver, 112
i2cReleaseBus
 I2C Driver, 115
i2cStart
 I2C Driver, 113
i2cstate_t
 I2C Driver, 116
i2cStop
 I2C Driver, 113
ICU Driver, 116
 _icu_isr_invoke_period_cb, 126
 _icu_isr_invoke_width_cb, 126
 ICU_ACTIVE, 129
 ICU_IDLE, 129
 ICU_INPUT_ACTIVE_HIGH, 129
 ICU_INPUT_ACTIVE_LOW, 129
 ICU_READY, 129
 ICU_STOP, 129
 ICU_UNINIT, 129
 ICU_WAITING, 129
 icu_lld_disable, 123
 icu_lld_enable, 123
 icu_lld_get_period, 128
 icu_lld_get_width, 128
 icu_lld_init, 122
 icu_lld_start, 123
 icu_lld_stop, 123
 icucallback_t, 128
 icucnt_t, 129
 ICUD1, 124
 ICUD2, 124
 ICUD3, 124
 ICUD4, 124
 ICUD5, 124
 ICUD8, 124
 icuDisable, 122
 icuDisablel, 125
 ICUDriver, 128
 icuEnable, 121
 icuEnablel, 125
 icufreq_t, 129
 icuGetPeriodl, 125
 icuGetWidthl, 125
 icuInit, 120
 icumode_t, 129
 icuObjectInit, 120
 icuStart, 121
 icustate_t, 129
 icuStop, 121
 STM32_ICU_TIM1_IRQ_PRIORITY, 127
 STM32_ICU_TIM2_IRQ_PRIORITY, 127
 STM32_ICU_TIM3_IRQ_PRIORITY, 127
 STM32_ICU_TIM4_IRQ_PRIORITY, 127
 STM32_ICU_TIM5_IRQ_PRIORITY, 128
 STM32_ICU_TIM8_IRQ_PRIORITY, 128
 STM32_ICU_USE_TIM1, 126
 STM32_ICU_USE_TIM2, 126
 STM32_ICU_USE_TIM3, 127
 STM32_ICU_USE_TIM4, 127
 STM32_ICU_USE_TIM5, 127

STM32_ICU_USE_TIM8, [127](#)
icu.c, [376](#)
icu.h, [376](#)
ICU_ACTIVE
 ICU Driver, [129](#)
ICU_IDLE
 ICU Driver, [129](#)
ICU_INPUT_ACTIVE_HIGH
 ICU Driver, [129](#)
ICU_INPUT_ACTIVE_LOW
 ICU Driver, [129](#)
ICU_READY
 ICU Driver, [129](#)
ICU_STOP
 ICU Driver, [129](#)
ICU_UNINIT
 ICU Driver, [129](#)
ICU_WAITING
 ICU Driver, [129](#)
icu_lld.c, [378](#)
icu_lld.h, [378](#)
icu_lld_disable
 ICU Driver, [123](#)
icu_lld_enable
 ICU Driver, [123](#)
icu_lld_get_period
 ICU Driver, [128](#)
icu_lld_get_width
 ICU Driver, [128](#)
icu_lld_init
 ICU Driver, [122](#)
icu_lld_start
 ICU Driver, [123](#)
icu_lld_stop
 ICU Driver, [123](#)
icucallback_t
 ICU Driver, [128](#)
icucnt_t
 ICU Driver, [129](#)
ICUConfig, [313](#)
 frequency, [315](#)
 mode, [315](#)
 period_cb, [315](#)
 width_cb, [315](#)
ICUD1
 ICU Driver, [124](#)
ICUD2
 ICU Driver, [124](#)
ICUD3
 ICU Driver, [124](#)
ICUD4
 ICU Driver, [124](#)
ICUD5
 ICU Driver, [124](#)
ICUD8
 ICU Driver, [124](#)
icuDisable
 ICU Driver, [122](#)
icuDisableI

 ICU Driver, [125](#)
ICUDriver, [315](#)
 clock, [317](#)
 config, [317](#)
 ICU Driver, [128](#)
 state, [317](#)
 tim, [317](#)
icuEnable
 ICU Driver, [121](#)
icuEnableI
 ICU Driver, [125](#)
icufreq_t
 ICU Driver, [129](#)
icuGetPeriodI
 ICU Driver, [125](#)
icuGetWidthI
 ICU Driver, [125](#)
icuInit
 ICU Driver, [120](#)
icumode_t
 ICU Driver, [129](#)
icuObjectInit
 ICU Driver, [120](#)
icuStart
 ICU Driver, [121](#)
icustate_t
 ICU Driver, [129](#)
icuStop
 ICU Driver, [121](#)
ifcr
 stm32_dma_stream_t, [338](#)
inserted_event
 MMCDriver, [321](#)
IOBus, [317](#)
 mask, [318](#)
 offset, [319](#)
 portid, [318](#)
IOBUS_DECL
 PAL Driver, [153](#)
iomode_t
 PAL Driver, [164](#)
IOPORT1
 PAL Driver, [160](#)
IOPORT2
 PAL Driver, [160](#)
IOPORT3
 PAL Driver, [160](#)
IOPORT4
 PAL Driver, [160](#)
IOPORT5
 PAL Driver, [161](#)
IOPORT6
 PAL Driver, [161](#)
IOPORT7
 PAL Driver, [161](#)
IOPORT8
 PAL Driver, [161](#)
IOPORT9
 PAL Driver, [161](#)

ioportid_t
 PAL Driver, 164

ioportmask_t
 PAL Driver, 163

is_inserted
 MMCDriver, 321

is_protected
 MMCDriver, 321

ishift
 stm32_dma_stream_t, 338

last
 TimeMeasurement, 340

lscfg
 MMCDriver, 321

MAC Driver, 129
 MAC_ACTIVE, 134
 MAC_STOP, 134
 MAC_UNINIT, 134
 MAC_USE_EVENTS, 133
 MACDriver, 134
 macGetReceiveEventSource, 133
 macInit, 130
 macObjectInit, 130
 macPollLinkStatus, 133
 macReadReceiveDescriptor, 134
 macReleaseReceiveDescriptor, 132
 macReleaseTransmitDescriptor, 132
 macStart, 131
 macstate_t, 134
 macStop, 131
 macWaitReceiveDescriptor, 132
 macWaitTransmitDescriptor, 131
 macWriteTransmitDescriptor, 133

mac.c, 380

mac.h, 380

MAC_ACTIVE
 MAC Driver, 134

MAC_STOP
 MAC Driver, 134

MAC_UNINIT
 MAC Driver, 134

MAC_USE_EVENTS
 Configuration, 13
 MAC Driver, 133

MACDriver
 MAC Driver, 134

macGetReceiveEventSource
 MAC Driver, 133

macInit
 MAC Driver, 130

macObjectInit
 MAC Driver, 130

macPollLinkStatus
 MAC Driver, 133

macReadReceiveDescriptor
 MAC Driver, 134

macReleaseReceiveDescriptor
 MAC Driver, 134

macReleaseTransmitDescriptor
 MAC Driver, 132

macStart
 MAC Driver, 131

macstate_t
 MAC Driver, 134

macStop
 MAC Driver, 131

macWaitReceiveDescriptor
 MAC Driver, 132

macWaitTransmitDescriptor
 MAC Driver, 131

macWriteTransmitDescriptor
 MAC Driver, 133

MAC Driver, 132

macReleaseTransmitDescriptor
 MAC Driver, 132

macStart
 MAC Driver, 131

macstate_t
 MAC Driver, 134

macStop
 MAC Driver, 131

macWaitReceiveDescriptor
 MAC Driver, 132

macWaitTransmitDescriptor
 MAC Driver, 131

macWriteTransmitDescriptor
 MAC Driver, 133

mask
 IOBus, 318

MMC over SPI Driver, 134
 MMC_INSERTED, 145
 MMC_READING, 145
 MMC_READY, 145
 MMC_STOP, 145
 MMC_UNINIT, 145
 MMC_WAIT, 145
 MMC_WRITING, 145
 MMC_NICE_WAITING, 144
 MMC_POLLING_DELAY, 144
 MMC_POLLING_INTERVAL, 144
 MMC_SECTOR_SIZE, 144

mmcConnect, 138

mmcDisconnect, 138

mmcGetDriverState, 144

mmcInit, 137

mmclsWriteProtected, 144

mmcObjectInit, 137

mmcquery_t, 145

mmcSequentialRead, 140

mmcSequentialWrite, 142

mmcStart, 137

mmcStartSequentialRead, 139

mmcStartSequentialWrite, 141

mmcstate_t, 145

mmcStop, 137

mmcStopSequentialRead, 140

mmcStopSequentialWrite, 143

MMC_INSERTED
 MMC over SPI Driver, 145

MMC_READING
 MMC over SPI Driver, 145

MMC_READY
 MMC over SPI Driver, 145

MMC_STOP
 MMC over SPI Driver, 145

MMC_UNINIT
 MMC over SPI Driver, 145

MMC_WAIT
 MMC over SPI Driver, 145

MMC_WRITING
 MMC over SPI Driver, 145

MMC_NICE_WAITING
 Configuration, 13
 MMC over SPI Driver, 144

MMC_POLLING_DELAY
 Configuration, 13
 MMC over SPI Driver, 144

MMC_POLLING_INTERVAL
 Configuration, 13
 MMC over SPI Driver, 144

MMC_SECTOR_SIZE
 Configuration, 13
 MMC over SPI Driver, 144

mmc_spi.c, 381

mmc_spi.h, 382

MMC_USE_SPI_POLLING
 Configuration, 13

MMCConfig, 319

mmcConnect
 MMC over SPI Driver, 138

mmcDisconnect
 MMC over SPI Driver, 138

MMCDriver, 319

 cnt, 322
 config, 321
 hscfg, 321
 inserted_event, 321
 is_inserted, 321
 is_protected, 321
 lscfg, 321
 removed_event, 322
 spip, 321
 state, 321
 vt, 322

mmcGetDriverState
 MMC over SPI Driver, 144

mmclnit
 MMC over SPI Driver, 137

mmclsWriteProtected
 MMC over SPI Driver, 144

mmcObjectInit
 MMC over SPI Driver, 137

mmcquery_t
 MMC over SPI Driver, 145

mmcSequentialRead
 MMC over SPI Driver, 140

mmcSequentialWrite
 MMC over SPI Driver, 142

mmcStart
 MMC over SPI Driver, 137

mmcStartSequentialRead
 MMC over SPI Driver, 139

mmcStartSequentialWrite
 MMC over SPI Driver, 141

mmcstate_t
 MMC over SPI Driver, 145

mmcStop
 MMC over SPI Driver, 137

mmcStopSequentialRead
 MMC over SPI Driver, 140

mmcStopSequentialWrite
 MMC over SPI Driver, 143

mmcStopSequentialWrite
 MMC over SPI Driver, 143

mode
 EXTChannelConfig, 306
 ICUConfig, 315
 PWMChannelConfig, 326

moder
 stm32_gpio_setup_t, 339

MS2RTT
 HAL Driver, 87

mutex
 ADCDriver, 304
 SPIDriver, 337

num_channels
 ADCConversionGroup, 301

odr
 stm32_gpio_setup_t, 339

offset
 IOBus, 319

ospeedr
 stm32_gpio_setup_t, 339

OTG_FS_IRQHandler
 HAL Driver, 102

OTG_FS_WKUP_IRQHandler
 HAL Driver, 100

OTG_HS_EP1_IN_IRQHandler
 HAL Driver, 102

OTG_HS_EP1_OUT_IRQHandler
 HAL Driver, 102

OTG_HS_IRQHandler
 HAL Driver, 103

OTG_HS_WKUP_IRQHandler
 HAL Driver, 103

otyper
 stm32_gpio_setup_t, 339

PADData
 PALConfig, 323

PAL Driver, 145

 _IOBUS_DATA, 153
 _pal_lld_init, 151
 _pal_lld_setgroupmode, 151
 IOBUS_DECL, 153
 iomode_t, 164
 IOPORT1, 160
 IOPORT2, 160
 IOPORT3, 160
 IOPORT4, 160
 IOPORT5, 161
 IOPORT6, 161
 IOPORT7, 161
 IOPORT8, 161
 IOPORT9, 161
 ioportid_t, 164
 ioportmask_t, 163
 PAL_GROUP_MASK, 152
 PAL_HIGH, 152

PAL_IOPORTS_WIDTH, 160
pal_lld_clearport, 162
pal_lld_init, 161
pal_lld_readlatch, 161
pal_lld_readport, 161
pal_lld_setgroupmode, 163
pal_lld_setport, 162
pal_lld_writegroup, 163
pal_lld_writepad, 163
pal_lld_writeport, 162
PAL_LOW, 152
PAL_MODE_ALTERNATE, 159
PAL_MODE_INPUT, 152, 159
PAL_MODE_INPUT_ANALOG, 152, 160
PAL_MODE_INPUT_PULLDOWN, 152, 159
PAL_MODE_INPUT_PULLUP, 152, 159
PAL_MODE_OUTPUT_OPENDRAIN, 152, 160
PAL_MODE_OUTPUT_PUSH_PULL, 152, 160
PAL_MODE_RESET, 151, 159
PAL_MODE_UNCONNECTED, 151, 159
PAL_PORT_BIT, 152
PAL_WHOLE_PORT, 160
palClearPad, 158
palClearPort, 155
pallInit, 153
palReadBus, 150
palReadGroup, 155
palReadLatch, 154
palReadPad, 156
palReadPort, 153
palSetBusMode, 150
palSetGroupMode, 156
palSetPad, 157
palSetPadMode, 158
palSetPort, 154
palTogglePad, 158
palTogglePort, 155
palWriteBus, 150
palWriteGroup, 156
palWritePad, 157
palWritePort, 154
pal.c, 383
pal.h, 384
PAL_GROUP_MASK
 PAL Driver, 152
PAL_HIGH
 PAL Driver, 152
PAL_IOPORTS_WIDTH
 PAL Driver, 160
pal_lld.c, 385
pal_lld.h, 386
pal_lld_clearport
 PAL Driver, 162
pal_lld_init
 PAL Driver, 161
pal_lld_readlatch
 PAL Driver, 161
pal_lld_readport
 PAL Driver, 161
pal_lld_setgroupmode
 PAL Driver, 163
pal_lld_setport
 PAL Driver, 162
pal_lld_writegroup
 PAL Driver, 163
pal_lld_writepad
 PAL Driver, 163
pal_lld_writeport
 PAL Driver, 162
PAL_LOW
 PAL Driver, 152
PAL_MODE_ALTERNATE
 PAL Driver, 159
PAL_MODE_INPUT
 PAL Driver, 152, 159
PAL_MODE_INPUT_ANALOG
 PAL Driver, 152, 160
PAL_MODE_INPUT_PULLDOWN
 PAL Driver, 152, 159
PAL_MODE_INPUT_PULLUP
 PAL Driver, 152, 159
PAL_MODE_OUTPUT_OPENDRAIN
 PAL Driver, 152, 160
PAL_MODE_OUTPUT_PUSH_PULL
 PAL Driver, 152, 160
PAL_MODE_RESET
 PAL Driver, 151, 159
PAL_MODE_UNCONNECTED
 PAL Driver, 151, 159
PAL_PORT_BIT
 PAL Driver, 152
PAL_WHOLE_PORT
 PAL Driver, 160
palClearPad
 PAL Driver, 158
palClearPort
 PAL Driver, 155
PALConfig, 322
 PAData, 323
 PBData, 323
 PCData, 323
 PDData, 324
pallInit
 PAL Driver, 153
palReadBus
 PAL Driver, 150
palReadGroup
 PAL Driver, 155
palReadLatch
 PAL Driver, 154
palReadPad
 PAL Driver, 156
palReadPort
 PAL Driver, 153
palSetBusMode
 PAL Driver, 150
palSetGroupMode
 PAL Driver, 156

palSetPad
 PAL Driver, 157
palSetPadMode
 PAL Driver, 158
palSetPort
 PAL Driver, 154
palTogglePad
 PAL Driver, 158
palTogglePort
 PAL Driver, 155
palWriteBus
 PAL Driver, 150
palWriteGroup
 PAL Driver, 156
palWritePad
 PAL Driver, 157
palWritePort
 PAL Driver, 154
PBData
 PALConfig, 323
PCData
 PALConfig, 323
PDData
 PALConfig, 324
period
 PWMConfig, 328
 PWMDriver, 331
period_cb
 ICUConfig, 315
portid
 IOBus, 318
pupdr
 stm32_gpio_setup_t, 339
PVD_IRQHandler
 HAL Driver, 96
PWM Driver, 164
 PWM_READY, 180
 PWM_STOP, 180
 PWM_UNINIT, 180
 PWM_CHANNELS, 177
 PWM_COMPLEMENTARY_OUTPUT_ACTIVE_HIGH, 177
 PWM_COMPLEMENTARY_OUTPUT_ACTIVE_LOW, 177
 PWM_COMPLEMENTARY_OUTPUT_DISABLED, 177
 PWM_COMPLEMENTARY_OUTPUT_MASK, 177
 PWM_DEGREES_TO_WIDTH, 175
 PWM_FRACTION_TO_WIDTH, 174
 pwm_lld_change_period, 179
 pwm_lld_disable_channel, 172
 pwm_lld_enable_channel, 172
 pwm_lld_init, 171
 pwm_lld_start, 171
 pwm_lld_stop, 172
 PWM_OUTPUT_ACTIVE_HIGH, 174
 PWM_OUTPUT_ACTIVE_LOW, 174
 PWM_OUTPUT_DISABLED, 174
 PWM_OUTPUT_MASK, 174
 PWM_PERCENTAGE_TO_WIDTH, 175
 pwmcallback_t, 180
 pwmChangePeriod, 169
 pwmChangePeriodl, 175
 pwmchannel_t, 180
 pwmcnt_t, 180
 PWMD1, 173
 PWMD2, 173
 PWMD3, 173
 PWMD4, 173
 PWMD5, 173
 PWMD8, 174
 pwmDisableChannel, 171
 pwmDisableChannell, 176
 PWMDriver, 180
 pwmEnableChannel, 170
 pwmEnableChannell, 176
 pwmlInit, 168
 pwmmodet, 180
 pwmObjectInit, 168
 pwmStart, 169
 pwmstate_t, 180
 pwmStop, 169
 STM32_PWM_TIM1_IRQ_PRIORITY, 179
 STM32_PWM_TIM2_IRQ_PRIORITY, 179
 STM32_PWM_TIM3_IRQ_PRIORITY, 179
 STM32_PWM_TIM4_IRQ_PRIORITY, 179
 STM32_PWM_TIM5_IRQ_PRIORITY, 179
 STM32_PWM_TIM8_IRQ_PRIORITY, 179
 STM32_PWM_USE_ADVANCED, 178
 STM32_PWM_USE_TIM1, 178
 STM32_PWM_USE_TIM2, 178
 STM32_PWM_USE_TIM3, 178
 STM32_PWM_USE_TIM4, 178
 STM32_PWM_USE_TIM5, 178
 STM32_PWM_USE_TIM8, 179
 pwm.c, 388
 pwm.h, 388
 PWM_READY
 PWM Driver, 180
 PWM_STOP
 PWM Driver, 180
 PWM_UNINIT
 PWM Driver, 180
 PWM_CHANNELS
 PWM Driver, 177
 PWM_COMPLEMENTARY_OUTPUT_ACTIVE_HIGH
 PWM Driver, 177
 PWM_COMPLEMENTARY_OUTPUT_ACTIVE_LOW
 PWM Driver, 177
 PWM_COMPLEMENTARY_OUTPUT_DISABLED
 PWM Driver, 177
 PWM_COMPLEMENTARY_OUTPUT_MASK
 PWM Driver, 177
 PWM_DEGREES_TO_WIDTH
 PWM Driver, 175
 PWM_FRACTION_TO_WIDTH
 PWM Driver, 174
 pwm_lld.c, 390
 pwm_lld.h, 390

pwm_lld_change_period
 PWM Driver, 179

pwm_lld_disable_channel
 PWM Driver, 172

pwm_lld_enable_channel
 PWM Driver, 172

pwm_lld_init
 PWM Driver, 171

pwm_lld_start
 PWM Driver, 171

pwm_lld_stop
 PWM Driver, 172

PWM_OUTPUT_ACTIVE_HIGH
 PWM Driver, 174

PWM_OUTPUT_ACTIVE_LOW
 PWM Driver, 174

PWM_OUTPUT_DISABLED
 PWM Driver, 174

PWM_OUTPUT_MASK
 PWM Driver, 174

PWM_PERCENTAGE_TO_WIDTH
 PWM Driver, 175

pwmcallback_t
 PWM Driver, 180

pwmChangePeriod
 PWM Driver, 169

pwmChangePeriodl
 PWM Driver, 175

pwmclock_t
 PWM Driver, 180

PWMChannelConfig, 324
 callback, 326
 mode, 326

pwmcnt_t
 PWM Driver, 180

PWMConfig, 326
 bdtr, 328
 callback, 328
 channels, 328
 cr2, 328
 frequency, 328
 period, 328

PWMD1
 PWM Driver, 173

PWMD2
 PWM Driver, 173

PWMD3
 PWM Driver, 173

PWMD4
 PWM Driver, 173

PWMD5
 PWM Driver, 173

PWMD8
 PWM Driver, 174

pwmDisableChannel
 PWM Driver, 171

pwmDisableChannell
 PWM Driver, 176

PWMDriver, 329

clock, 331

config, 331

period, 331

PWM Driver, 180

state, 331

tim, 331

pwmEnableChannel
 PWM Driver, 170

pwmEnableChannell
 PWM Driver, 176

pwmlinit
 PWM Driver, 168

pwmmode_t
 PWM Driver, 180

pwmObjectInit
 PWM Driver, 168

pwmStart
 PWM Driver, 169

pwmsstate_t
 PWM Driver, 180

pwmStop
 PWM Driver, 169

RCC_IRQHandler
 HAL Driver, 97

rccDisableADC1
 STM32F4xx RCC Support, 281

rccDisableADC2
 STM32F4xx RCC Support, 281

rccDisableADC3
 STM32F4xx RCC Support, 282

rccDisableAHB1
 STM32F4xx RCC Support, 278

rccDisableAHB2
 STM32F4xx RCC Support, 279

rccDisableAHB3
 STM32F4xx RCC Support, 280

rccDisableAPB1
 STM32F4xx RCC Support, 276

rccDisableAPB2
 STM32F4xx RCC Support, 277

rccDisableCAN1
 STM32F4xx RCC Support, 284

rccDisableCAN2
 STM32F4xx RCC Support, 285

rccDisableDMA1
 STM32F4xx RCC Support, 283

rccDisableDMA2
 STM32F4xx RCC Support, 283

rccDisableETH
 STM32F4xx RCC Support, 286

rccDisableI2C1
 STM32F4xx RCC Support, 286

rccDisableI2C2
 STM32F4xx RCC Support, 287

rccDisableI2C3
 STM32F4xx RCC Support, 287

rccDisableOTG_FS
 STM32F4xx RCC Support, 288

rccDisablePWRInterface
 STM32F4xx RCC Support, 284

rccDisableSDIO
 STM32F4xx RCC Support, 289

rccDisableSPI1
 STM32F4xx RCC Support, 289

rccDisableSPI2
 STM32F4xx RCC Support, 290

rccDisableSPI3
 STM32F4xx RCC Support, 290

rccDisableTIM1
 STM32F4xx RCC Support, 291

rccDisableTIM2
 STM32F4xx RCC Support, 292

rccDisableTIM3
 STM32F4xx RCC Support, 292

rccDisableTIM4
 STM32F4xx RCC Support, 293

rccDisableTIM5
 STM32F4xx RCC Support, 294

rccDisableTIM8
 STM32F4xx RCC Support, 294

rccDisableUART4
 STM32F4xx RCC Support, 297

rccDisableUART5
 STM32F4xx RCC Support, 298

rccDisableUSART1
 STM32F4xx RCC Support, 295

rccDisableUSART2
 STM32F4xx RCC Support, 295

rccDisableUSART3
 STM32F4xx RCC Support, 296

rccDisableUSART6
 STM32F4xx RCC Support, 297

rccEnableADC1
 STM32F4xx RCC Support, 281

rccEnableADC2
 STM32F4xx RCC Support, 281

rccEnableADC3
 STM32F4xx RCC Support, 282

rccEnableAHB1
 STM32F4xx RCC Support, 278

rccEnableAHB2
 STM32F4xx RCC Support, 279

rccEnableAHB3
 STM32F4xx RCC Support, 280

rccEnableAPB1
 STM32F4xx RCC Support, 276

rccEnableAPB2
 STM32F4xx RCC Support, 277

rccEnableCAN1
 STM32F4xx RCC Support, 284

rccEnableCAN2
 STM32F4xx RCC Support, 285

rccEnableDMA1
 STM32F4xx RCC Support, 282

rccEnableDMA2
 STM32F4xx RCC Support, 283

rccEnableETH

 STM32F4xx RCC Support, 285

rccEnableI2C1
 STM32F4xx RCC Support, 286

rccEnableI2C2
 STM32F4xx RCC Support, 287

rccEnableI2C3
 STM32F4xx RCC Support, 287

rccEnableOTG_FS
 STM32F4xx RCC Support, 288

rccEnablePWRInterface
 STM32F4xx RCC Support, 283

rccEnableSDIO
 STM32F4xx RCC Support, 288

rccEnableSPI1
 STM32F4xx RCC Support, 289

rccEnableSPI2
 STM32F4xx RCC Support, 290

rccEnableSPI3
 STM32F4xx RCC Support, 290

rccEnableTIM1
 STM32F4xx RCC Support, 291

rccEnableTIM2
 STM32F4xx RCC Support, 292

rccEnableTIM3
 STM32F4xx RCC Support, 292

rccEnableTIM4
 STM32F4xx RCC Support, 293

rccEnableTIM5
 STM32F4xx RCC Support, 293

rccEnableTIM8
 STM32F4xx RCC Support, 294

rccEnableUART4
 STM32F4xx RCC Support, 297

rccEnableUSART1
 STM32F4xx RCC Support, 295

rccEnableUSART2
 STM32F4xx RCC Support, 295

rccEnableUSART3
 STM32F4xx RCC Support, 295

rccEnableUSART8
 STM32F4xx RCC Support, 294

rccEnableUSART5
 STM32F4xx RCC Support, 298

rccEnableUSART1
 STM32F4xx RCC Support, 295

rccEnableUSART2
 STM32F4xx RCC Support, 295

rccEnableUSART3
 STM32F4xx RCC Support, 296

rccEnableUSART6
 STM32F4xx RCC Support, 296

rccResetADC1
 STM32F4xx RCC Support, 281

rccResetADC2
 STM32F4xx RCC Support, 282

rccResetADC3
 STM32F4xx RCC Support, 282

rccResetAHB1
 STM32F4xx RCC Support, 278

rccResetAHB2
 STM32F4xx RCC Support, 279

rccResetAHB3
 STM32F4xx RCC Support, 280

rccResetAPB1
 STM32F4xx RCC Support, 276

rccResetAPB2
 STM32F4xx RCC Support, 277

rccResetCAN1
 STM32F4xx RCC Support, 284

rccResetCAN2
 STM32F4xx RCC Support, 285

rccResetDMA1
 STM32F4xx RCC Support, 283

rccResetDMA2
 STM32F4xx RCC Support, 283

rccResetETH
 STM32F4xx RCC Support, 286

rccResetI2C1
 STM32F4xx RCC Support, 286

rccResetI2C2
 STM32F4xx RCC Support, 287

rccResetI2C3
 STM32F4xx RCC Support, 288

rccResetOTG_FS
 STM32F4xx RCC Support, 288

rccResetPWRInterface
 STM32F4xx RCC Support, 284

rccResetSDIO
 STM32F4xx RCC Support, 289

rccResetSPI1
 STM32F4xx RCC Support, 290

rccResetSPI2
 STM32F4xx RCC Support, 290

rccResetSPI3
 STM32F4xx RCC Support, 291

rccResetTIM1
 STM32F4xx RCC Support, 291

rccResetTIM2
 STM32F4xx RCC Support, 292

rccResetTIM3
 STM32F4xx RCC Support, 293

rccResetTIM4
 STM32F4xx RCC Support, 293

rccResetTIM5
 STM32F4xx RCC Support, 294

rccResetTIM8
 STM32F4xx RCC Support, 295

rccResetUART4
 STM32F4xx RCC Support, 297

rccResetUART5
 STM32F4xx RCC Support, 298

rccResetUSART1
 STM32F4xx RCC Support, 295

rccResetUSART2
 STM32F4xx RCC Support, 296

rccResetUSART3
 STM32F4xx RCC Support, 296

rccResetUSART6
 STM32F4xx RCC Support, 298

removed_event
 MMCDriver, 322

RTC_Alarm_IRQHandler
 HAL Driver, 100

RTC_WKUP_IRQHandler
 HAL Driver, 96

rxbuf

 UARTDriver, 345

rxchar_cb
 UARTConfig, 342

rxdmamode
 SPIDriver, 337

rxend_cb
 UARTConfig, 342

rxerr_cb
 UARTConfig, 342

rxstate
 UARTDriver, 345

S2RTT
 HAL Driver, 86

samples
 ADCDriver, 304

sc_cr1
 SerialConfig, 332

sc_cr2
 SerialConfig, 332

sc_cr3
 SerialConfig, 332

sc_speed
 SerialConfig, 332

SD1
 Serial Driver, 189

SD2
 Serial Driver, 189

SD3
 Serial Driver, 189

SD4
 Serial Driver, 189

SD5
 Serial Driver, 189

SD6
 Serial Driver, 189

SD_READY
 Serial Driver, 196

SD_STOP
 Serial Driver, 196

SD_UNINIT
 Serial Driver, 196

SD_BREAK_DETECTED
 Serial Driver, 190

SD_FRAMING_ERROR
 Serial Driver, 189

sd_lld_init
 Serial Driver, 188

sd_lld_start
 Serial Driver, 188

sd_lld_stop
 Serial Driver, 189

SD_NOISE_ERROR
 Serial Driver, 190

SD_OVERRUN_ERROR
 Serial Driver, 189

SD_PARITY_ERROR
 Serial Driver, 189

sdAsynchronousRead

Serial Driver, 193
sdAsynchronousWrite
 Serial Driver, 193
SDC Driver, 196
 _sdc_wait_for_transfer_state, 201
 SDC_ACTIVE, 205
 SDC_CONNECTING, 205
 SDC_DISCONNECTING, 205
 SDC_READING, 205
 SDC_READY, 204
 SDC_STOP, 204
 SDC_UNINIT, 204
 SDC_WRITING, 205
 SDC_BLOCK_SIZE, 202
 SDC_CMD8_PATTERN, 202
 SDC_INIT_RETRY, 202
 SDC_MMU_SUPPORT, 202
 SDC_MODE_CARDTYPE_MASK, 202
 SDC_MODE_CARDTYPE_MMIC, 202
 SDC_MODE_CARDTYPE_SDV11, 202
 SDC_MODE_CARDTYPE_SDV20, 202
 SDC_MODE_HIGH_CAPACITY, 202
 SDC_NICE_WAITING, 203
 SDC_R1_ERROR, 203
 SDC_R1_ERROR_MASK, 202
 SDC_R1_IS_CARD_LOCKED, 203
 SDC_R1_STS, 203
 sdcConnect, 199
 sdcDisconnect, 200
 sdcGetDriverState, 203
 sdclInit, 199
 sdclsCardInserted, 203
 sdclsWriteProtected, 204
 sdcObjectInit, 199
 sdcRead, 200
 sdcStart, 199
 sdccstate_t, 204
 sdcStop, 199
 sdcWrite, 201
sdc.c, 392
sdc.h, 393
SDC_ACTIVE
 SDC Driver, 205
SDC_CONNECTING
 SDC Driver, 205
SDC_DISCONNECTING
 SDC Driver, 205
SDC_READING
 SDC Driver, 205
SDC_READY
 SDC Driver, 204
SDC_STOP
 SDC Driver, 204
SDC_UNINIT
 SDC Driver, 204
SDC_WRITING
 SDC Driver, 205
SDC_BLOCK_SIZE
 SDC Driver, 202
SDC_CMD8_PATTERN
 SDC Driver, 202
SDC_INIT_RETRY
 Configuration, 13
 SDC Driver, 202
SDC_MMU_SUPPORT
 Configuration, 14
 SDC Driver, 202
SDC_MODE_CARDTYPE_MASK
 SDC Driver, 202
SDC_MODE_CARDTYPE_MMIC
 SDC Driver, 202
SDC_MODE_CARDTYPE_SDV11
 SDC Driver, 202
SDC_MODE_CARDTYPE_SDV20
 SDC Driver, 202
SDC_MODE_HIGH_CAPACITY
 SDC Driver, 202
SDC_NICE_WAITING
 Configuration, 14
 SDC Driver, 203
SDC_R1_ERROR
 SDC Driver, 203
SDC_R1_ERROR_MASK
 SDC Driver, 202
SDC_R1_IS_CARD_LOCKED
 SDC Driver, 203
SDC_R1_STS
 SDC Driver, 203
sdcConnect
 SDC Driver, 199
sdcDisconnect
 SDC Driver, 200
sdcGetDriverState
 SDC Driver, 203
sdclInit
 SDC Driver, 199
sdclsCardInserted
 SDC Driver, 203
sdclsWriteProtected
 SDC Driver, 204
sdcObjectInit
 SDC Driver, 199
sdcRead
 SDC Driver, 200
sdcStart
 SDC Driver, 199
sdccstate_t
 SDC Driver, 204
sdcStop
 SDC Driver, 199
sdcWrite
 SDC Driver, 201
sdGet
 Serial Driver, 191
sdGetTimeout
 Serial Driver, 192
sdGetWouldBlock
 Serial Driver, 190

sdIncomingData
 Serial Driver, 186

sdInit
 Serial Driver, 184

SDIO_IRQHandler
 HAL Driver, 100

sdObjectInit
 Serial Driver, 185

sdPut
 Serial Driver, 191

sdPutTimeout
 Serial Driver, 191

sdPutWouldBlock
 Serial Driver, 190

sdRead
 Serial Driver, 193

sdReadTimeout
 Serial Driver, 193

sdRequestData
 Serial Driver, 187

sdStart
 Serial Driver, 185

sdstate_t
 Serial Driver, 196

sdStop
 Serial Driver, 186

sdWrite
 Serial Driver, 192

sdWriteTimeout
 Serial Driver, 192

selfindex
 stm32_dma_stream_t, 338

Serial Driver, 181

- __serial_driver_data, 195
- __serial_driver_methods, 190
- CH_IRQ_HANDLER, 187, 188
- SD1, 189
- SD2, 189
- SD3, 189
- SD4, 189
- SD5, 189
- SD6, 189
- SD_READY, 196
- SD_STOP, 196
- SD_UNINIT, 196
- SD_BREAK_DETECTED, 190
- SD_FRAMING_ERROR, 189
- sd_lld_init, 188
- sd_lld_start, 188
- sd_lld_stop, 189
- SD_NOISE_ERROR, 190
- SD_OVERRUN_ERROR, 189
- SD_PARITY_ERROR, 189
- sdAsynchronousRead, 193
- sdAsynchronousWrite, 193
- sdGet, 191
- sdGetTimeout, 192
- sdGetWouldBlock, 190
- sdIncomingData, 186

sdInit, 184

sdObjectInit, 185

sdPut, 191

sdPutTimeout, 191

sdPutWouldBlock, 190

sdRead, 193

sdReadTimeout, 193

sdRequestData, 187

sdStart, 185

sdstate_t, 196

sdStop, 186

sdWrite, 192

sdWriteTimeout, 192

SERIAL_BUFFERS_SIZE, 190

SERIAL_DEFAULT_BITRATE, 190

SerialDriver, 196

- STM32_SERIAL_UART4_PRIORITY, 195
- STM32_SERIAL_UART5_PRIORITY, 195
- STM32_SERIAL_USART1_PRIORITY, 195
- STM32_SERIAL_USART2_PRIORITY, 195
- STM32_SERIAL_USART3_PRIORITY, 195
- STM32_SERIAL_USART6_PRIORITY, 195
- STM32_SERIAL_USE_UART4, 194
- STM32_SERIAL_USE_UART5, 194
- STM32_SERIAL_USE_USART1, 194
- STM32_SERIAL_USE_USART2, 194
- STM32_SERIAL_USE_USART3, 194
- STM32_SERIAL_USE_USART6, 195
- USART_CR2_STOP0P5_BITS, 196
- USART_CR2_STOP1_BITS, 196
- USART_CR2_STOP1P5_BITS, 196
- USART_CR2_STOP2_BITS, 196

serial.c, 394

serial.h, 394

SERIAL_BUFFERS_SIZE
 Configuration, 14
 Serial Driver, 190

SERIAL_DEFAULT_BITRATE
 Configuration, 14
 Serial Driver, 190

serial_lld.c, 396

serial_lld.h, 397

SERIAL_USB_BUFFERS_SIZE
 Configuration, 14

SerialConfig, 331

- sc_cr1, 332
- sc_cr2, 332
- sc_cr3, 332
- sc_speed, 332

SerialDriver, 332

- Serial Driver, 196
- vmt, 333

SerialDriverVMT, 333

smpr1
 ADCConversionGroup, 301

smpr2
 ADCConversionGroup, 301

speed
 UARTConfig, 342

spi
 SPIDriver, 337
SPI Driver, 205
 _spi_isr_code, 223
 _spi_wait_s, 222
 _spi_wakeup_isr, 222
 SPI_ACTIVE, 226
 SPI_COMPLETE, 226
 SPI_READY, 226
 SPI_STOP, 226
 SPI_UNINIT, 226
 spi_lld_exchange, 216
 spi_lld_ignore, 216
 spi_lld_init, 214
 spi_lld_polled_exchange, 218
 spi_lld_receive, 217
 spi_lld_select, 216
 spi_lld_send, 217
 spi_lld_start, 215
 spi_lld_stop, 215
 spi_lld_unselect, 216
 SPI_USE_MUTUAL_EXCLUSION, 218
 SPI_USE_WAIT, 218
 spiAcquireBus, 214
 spicallback_t, 226
 SPID1, 218
 SPID2, 218
 SPID3, 218
 SPIDriver, 226
 spiExchange, 213
 spilgnore, 212
 spilnit, 208
 spiObjectInit, 209
 spiPolledExchange, 222
 spiReceive, 213
 spiReleaseBus, 214
 spiSelect, 210
 spiSelectl, 219
 spiSend, 213
 spiStart, 209
 spiStartExchange, 211
 spiStartExchangel, 220
 spiStartIgnore, 210
 spiStartIgnorel, 219
 spiStartReceive, 212
 spiStartReceivel, 221
 spiStartSend, 211
 spiStartSendl, 220
 spistate_t, 226
 spiStop, 209
 spiUnselect, 210
 spiUnselectl, 219
 STM32_SPI_DMA_ERROR_HOOK, 225
 STM32_SPI_SPI1_DMA_PRIORITY, 224
 STM32_SPI_SPI1_IRQ_PRIORITY, 224
 STM32_SPI_SPI1_RX_DMA_STREAM, 225
 STM32_SPI_SPI1_TX_DMA_STREAM, 225
 STM32_SPI_SPI2_DMA_PRIORITY, 224
 STM32_SPI_SPI2_IRQ_PRIORITY, 224
STM32_SPI_SPI2_RX_DMA_STREAM, 225
STM32_SPI_SPI2_TX_DMA_STREAM, 225
STM32_SPI_SPI3_DMA_PRIORITY, 224
STM32_SPI_SPI3_IRQ_PRIORITY, 224
STM32_SPI_SPI3_RX_DMA_STREAM, 225
STM32_SPI_SPI3_TX_DMA_STREAM, 225
STM32_SPI_USE_SPI1, 223
STM32_SPI_USE_SPI2, 224
STM32_SPI_USE_SPI3, 224
spi.c, 398
spi.h, 399
SPI1_IRQHandler
 HAL Driver, 99
SPI2_IRQHandler
 HAL Driver, 99
SPI3_IRQHandler
 HAL Driver, 100
SPI_ACTIVE
 SPI Driver, 226
SPI_COMPLETE
 SPI Driver, 226
SPI_READY
 SPI Driver, 226
SPI_STOP
 SPI Driver, 226
SPI_UNINIT
 SPI Driver, 226
spi_lld.c, 400
spi_lld.h, 401
spi_lld_exchange
 SPI Driver, 216
spi_lld_ignore
 SPI Driver, 216
spi_lld_init
 SPI Driver, 214
spi_lld_polled_exchange
 SPI Driver, 218
spi_lld_receive
 SPI Driver, 217
spi_lld_select
 SPI Driver, 216
spi_lld_send
 SPI Driver, 217
spi_lld_start
 SPI Driver, 215
spi_lld_stop
 SPI Driver, 215
spi_lld_unselect
 SPI Driver, 216
SPI_USE_MUTUAL_EXCLUSION
 Configuration, 14
 SPI Driver, 218
SPI_USE_WAIT
 Configuration, 14
 SPI Driver, 218
spiAcquireBus
 SPI Driver, 214
spicallback_t
 SPI Driver, 226

SPIConfig, 333
 cr1, 335
 end_cb, 335
 sspad, 335
 ssport, 335
SPID1
 SPI Driver, 218
SPID2
 SPI Driver, 218
SPID3
 SPI Driver, 218
SPIDriver, 335
 config, 337
 dmarx, 337
 dmatx, 337
 mutex, 337
 rxdmamode, 337
 spi, 337
 SPI Driver, 226
 state, 337
 thread, 337
 txdmamode, 337
spiExchange
 SPI Driver, 213
spilgnore
 SPI Driver, 212
spilinit
 SPI Driver, 208
spiObjectInit
 SPI Driver, 209
spip
 MMCDriver, 321
spiPolledExchange
 SPI Driver, 222
spiReceive
 SPI Driver, 213
spiReleaseBus
 SPI Driver, 214
spiSelect
 SPI Driver, 210
spiSelectl
 SPI Driver, 219
spiSend
 SPI Driver, 213
spiStart
 SPI Driver, 209
spiStartExchange
 SPI Driver, 211
spiStartExchangel
 SPI Driver, 220
spiStartIgnore
 SPI Driver, 210
spiStartIgnorel
 SPI Driver, 219
spiStartReceive
 SPI Driver, 212
spiStartReceivel
 SPI Driver, 221
spiStartSend
 SPI Driver, 211
 spiStartSendl
 SPI Driver, 220
 spistate_t
 SPI Driver, 226
 spiStop
 SPI Driver, 209
 spiUnselect
 SPI Driver, 210
 spiUnselectl
 SPI Driver, 219
 sqr1
 ADCConversionGroup, 302
 sqr2
 ADCConversionGroup, 302
 sqr3
 ADCConversionGroup, 302
 sspad
 SPIConfig, 335
 ssport
 SPIConfig, 335
 start
 TimeMeasurement, 340
 state
 ADCDriver, 304
 EXTDriver, 308
 GPTDriver, 313
 ICUDriver, 317
 MMCDriver, 321
 PWMDriver, 331
 SPIDriver, 337
 UARTDriver, 345
 stm32.h, 403
 STM32_0WS_THRESHOLD
 HAL Driver, 106
 STM32_ACTIVATE_PLL
 HAL Driver, 107
 STM32_ACTIVATE_PLLI2S
 HAL Driver, 108
 STM32_ADC_ADC1_DMA_IRQ_PRIORITY
 ADC Driver, 37
 STM32_ADC_ADC1_DMA_PRIORITY
 ADC Driver, 37
 STM32_ADC_ADC1_DMA_STREAM
 ADC Driver, 37
 STM32_ADC_ADC2_DMA_IRQ_PRIORITY
 ADC Driver, 38
 STM32_ADC_ADC2_DMA_PRIORITY
 ADC Driver, 37
 STM32_ADC_ADC2_DMA_STREAM
 ADC Driver, 37
 STM32_ADC_ADC3_DMA_IRQ_PRIORITY
 ADC Driver, 38
 STM32_ADC_ADC3_DMA_PRIORITY
 ADC Driver, 37
 STM32_ADC_ADC3_DMA_STREAM
 ADC Driver, 37
 STM32_ADC_ADCPRE
 ADC Driver, 36

STM32_ADC_IRQ_PRIORITY
 ADC Driver, 37
STM32_ADC_USE_ADC1
 ADC Driver, 36
STM32_ADC_USE_ADC2
 ADC Driver, 36
STM32_ADC_USE_ADC3
 ADC Driver, 37
STM32_ADCCLK_MAX
 ADC Driver, 34
STM32_ADCCLK_MIN
 ADC Driver, 33
STM32_CLOCK48_REQUIRED
 HAL Driver, 104
stm32_clock_init
 HAL Driver, 86
stm32_dma.c, 403
stm32_dma.h, 405
STM32_DMA1_STREAMS_MASK
 STM32F4xx DMA Support, 263
STM32_DMA2_STREAMS_MASK
 STM32F4xx DMA Support, 263
STM32_DMA_CR_RESET_VALUE
 STM32F4xx DMA Support, 263
STM32_DMA_FCR_RESET_VALUE
 STM32F4xx DMA Support, 263
STM32_DMA_GETCHANNEL
 STM32F4xx DMA Support, 263
STM32_DMA_IS_VALID_ID
 STM32F4xx DMA Support, 264
STM32_DMA_ISR_MASK
 STM32F4xx DMA Support, 263
STM32_DMA_STREAM
 STM32F4xx DMA Support, 264
STM32_DMA_STREAM_ID
 STM32F4xx DMA Support, 264
STM32_DMA_STREAM_ID_MSK
 STM32F4xx DMA Support, 264
stm32_dma_stream_t, 338
 ifcr, 338
 ishift, 338
 selfindex, 338
 stream, 338
 vector, 338
STM32_DMA_STREAMS
 STM32F4xx DMA Support, 263
stm32_dmaisr_t
 STM32F4xx DMA Support, 270
STM32_EXTI0_IRQ_PRIORITY
 EXT Driver, 57
STM32_EXTI10_15_IRQ_PRIORITY
 EXT Driver, 58
STM32_EXTI16_IRQ_PRIORITY
 EXT Driver, 58
STM32_EXTI17_IRQ_PRIORITY
 EXT Driver, 58
STM32_EXTI18_IRQ_PRIORITY
 EXT Driver, 58
STM32_EXTI19_IRQ_PRIORITY
 EXT Driver, 58
STM32_EXTI1_IRQ_PRIORITY
 EXT Driver, 58
STM32_EXTI2_IRQ_PRIORITY
 EXT Driver, 57
STM32_EXTI20_IRQ_PRIORITY
 EXT Driver, 58
STM32_EXTI21_IRQ_PRIORITY
 EXT Driver, 58
STM32_EXTI22_IRQ_PRIORITY
 EXT Driver, 58
STM32_EXTI23_IRQ_PRIORITY
 EXT Driver, 57
STM32_EXTI3_IRQ_PRIORITY
 EXT Driver, 58
STM32_EXTI4_IRQ_PRIORITY
 EXT Driver, 58
STM32_EXTI5_9_IRQ_PRIORITY
 EXT Driver, 58
STM32_FLASHBITS
 HAL Driver, 109
stm32_gpio_setup_t, 339
 afrh, 339
 afrl, 339
 moder, 339
 odr, 339
 ospeedr, 339
 otyper, 339
 pupdr, 339
STM32_GPT_TIM1_IRQ_PRIORITY
 GPT Driver, 71
STM32_GPT_TIM2_IRQ_PRIORITY
 GPT Driver, 71
STM32_GPT_TIM3_IRQ_PRIORITY
 GPT Driver, 71
STM32_GPT_TIM4_IRQ_PRIORITY
 GPT Driver, 71
STM32_GPT_TIM5_IRQ_PRIORITY
 GPT Driver, 71
STM32_GPT_TIM8_IRQ_PRIORITY
 GPT Driver, 72
STM32_GPT_USE_TIM1
 GPT Driver, 70
STM32_GPT_USE_TIM2
 GPT Driver, 70
STM32_GPT_USE_TIM3
 GPT Driver, 70
STM32_GPT_USE_TIM4
 GPT Driver, 71
STM32_GPT_USE_TIM5
 GPT Driver, 71
STM32_GPT_USE_TIM8
 GPT Driver, 71
STM32_HCLK
 HAL Driver, 107
STM32_HPRE
 HAL Driver, 105
STM32_HPRE_DIV1
 HAL Driver, 91
STM32_HPRE_DIV128
 HAL Driver, 92

STM32_HPREF_DIV16
 HAL Driver, 92
STM32_HPREF_DIV2
 HAL Driver, 91
STM32_HPREF_DIV256
 HAL Driver, 92
STM32_HPREF_DIV4
 HAL Driver, 92
STM32_HPREF_DIV512
 HAL Driver, 92
STM32_HPREF_DIV64
 HAL Driver, 92
STM32_HPREF_DIV8
 HAL Driver, 92
STM32_HPREF_MASK
 HAL Driver, 91
STM32_HSE_ENABLED
 HAL Driver, 104
STM32_HSECLK_MAX
 HAL Driver, 88
STM32_HSECLK_MIN
 HAL Driver, 88
STM32_HSEDIVCLK
 HAL Driver, 109
STM32_HSI_ENABLED
 HAL Driver, 103
STM32_HSICLK
 HAL Driver, 89
STM32_I2SSRC
 HAL Driver, 106
STM32_I2SSRC_CKIN
 HAL Driver, 94
STM32_I2SSRC_MASK
 HAL Driver, 94
STM32_I2SSRC_PLLI2S
 HAL Driver, 94
STM32_ICU_TIM1_IRQ_PRIORITY
 ICU Driver, 127
STM32_ICU_TIM2_IRQ_PRIORITY
 ICU Driver, 127
STM32_ICU_TIM3_IRQ_PRIORITY
 ICU Driver, 127
STM32_ICU_TIM4_IRQ_PRIORITY
 ICU Driver, 127
STM32_ICU_TIM5_IRQ_PRIORITY
 ICU Driver, 128
STM32_ICU_TIM8_IRQ_PRIORITY
 ICU Driver, 128
STM32_ICU_USE_TIM1
 ICU Driver, 126
STM32_ICU_USE_TIM2
 ICU Driver, 126
STM32_ICU_USE_TIM3
 ICU Driver, 127
STM32_ICU_USE_TIM4
 ICU Driver, 127
STM32_ICU_USE_TIM5
 ICU Driver, 127
STM32_ICU_USE_TIM8
 ICU Driver, 127
STM32_LSE_ENABLED
 HAL Driver, 104
STM32_LSECLK_MAX
 HAL Driver, 88
STM32_LSECLK_MIN
 HAL Driver, 88
STM32_LSI_ENABLED
 HAL Driver, 104
STM32_LSICLK
 HAL Driver, 89
STM32_MCO1CLK
 HAL Driver, 108
STM32_MCO1DIVCLK
 HAL Driver, 108
STM32_MCO1PRE
 HAL Driver, 105
STM32_MCO1PRE_DIV1
 HAL Driver, 94
STM32_MCO1PRE_DIV2
 HAL Driver, 94
STM32_MCO1PRE_DIV3
 HAL Driver, 94
STM32_MCO1PRE_DIV4
 HAL Driver, 94
STM32_MCO1PRE_DIV5
 HAL Driver, 94
STM32_MCO1PRE_MASK
 HAL Driver, 94
STM32_MCO1SEL
 HAL Driver, 105
STM32_MCO1SEL_HSE
 HAL Driver, 93
STM32_MCO1SEL_LSE
 HAL Driver, 93
STM32_MCO1SEL_MASK
 HAL Driver, 93
STM32_MCO1SEL_PLL
 HAL Driver, 94
STM32_MCO2CLK
 HAL Driver, 108
STM32_MCO2DIVCLK
 HAL Driver, 108
STM32_MCO2PRE
 HAL Driver, 106
STM32_MCO2PRE_DIV1
 HAL Driver, 94
STM32_MCO2PRE_DIV2
 HAL Driver, 95
STM32_MCO2PRE_DIV3
 HAL Driver, 95
STM32_MCO2PRE_DIV4
 HAL Driver, 95
STM32_MCO2PRE_DIV5
 HAL Driver, 95
STM32_MCO2PRE_MASK
 HAL Driver, 94

STM32_MCO2SEL
 HAL Driver, 106
STM32_MCO2SEL_HSE
 HAL Driver, 95
STM32_MCO2SEL_MASK
 HAL Driver, 95
STM32_MCO2SEL_PLL
 HAL Driver, 95
STM32_MCO2SEL_PLLI2S
 HAL Driver, 95
STM32_MCO2SEL_SYSCLK
 HAL Driver, 95
STM32_NO_INIT
 HAL Driver, 103
STM32_PCLK1
 HAL Driver, 107
STM32_PCLK1_MAX
 HAL Driver, 89
STM32_PCLK2
 HAL Driver, 108
STM32_PCLK2_MAX
 HAL Driver, 89
STM32_PLL48CLK
 HAL Driver, 109
STM32_PLLCLKIN
 HAL Driver, 107
STM32_PLLCLKOUT
 HAL Driver, 107
STM32_PLLI2SCLKOUT
 HAL Driver, 108
STM32_PLLI2SN
 HAL Driver, 108
STM32_PLLI2SN_MASK
 HAL Driver, 96
STM32_PLLI2SN_VALUE
 HAL Driver, 106
STM32_PLLI2SR
 HAL Driver, 108
STM32_PLLI2SR_MASK
 HAL Driver, 96
STM32_PLLI2SR_VALUE
 HAL Driver, 106
STM32_PLLI2SVCO
 HAL Driver, 108
STM32_PLLIN_MAX
 HAL Driver, 88
STM32_PLLIN_MIN
 HAL Driver, 89
STM32_PLLM
 HAL Driver, 107
STM32_PLLM_VALUE
 HAL Driver, 104
STM32_PLLN
 HAL Driver, 107
STM32_PLLN_VALUE
 HAL Driver, 104
STM32_PLLOUT_MAX
 HAL Driver, 89
STM32_PLLOUT_MIN
 HAL Driver, 89
STM32_PLLP
 HAL Driver, 107
STM32_PLLP_DIV2
 HAL Driver, 90
STM32_PLLP_DIV4
 HAL Driver, 91
STM32_PLLP_DIV6
 HAL Driver, 91
STM32_PLLP_DIV8
 HAL Driver, 91
STM32_PLLP_MASK
 HAL Driver, 90
STM32_PLLP_VALUE
 HAL Driver, 104
STM32_PLLQ
 HAL Driver, 107
STM32_PLLQ_VALUE
 HAL Driver, 105
STM32_PLLSRC
 HAL Driver, 104
STM32_PLLSRC_HSE
 HAL Driver, 91
STM32_PLLSRC_HSI
 HAL Driver, 91
STM32_PLLVCO
 HAL Driver, 107
STM32_PLLVCO_MAX
 HAL Driver, 89
STM32_PLLVCO_MIN
 HAL Driver, 89
STM32_PLS
 HAL Driver, 103
STM32_PLS_LEV0
 HAL Driver, 90
STM32_PLS_LEV1
 HAL Driver, 90
STM32_PLS_LEV2
 HAL Driver, 90
STM32_PLS_LEV3
 HAL Driver, 90
STM32_PLS_LEV4
 HAL Driver, 90
STM32_PLS_LEV5
 HAL Driver, 90
STM32_PLS_LEV6
 HAL Driver, 90
STM32_PLS_LEV7
 HAL Driver, 90
STM32_PLS_MASK
 HAL Driver, 90
STM32_PPREG1
 HAL Driver, 105
STM32_PPREG1_DIV1
 HAL Driver, 92
STM32_PPREG1_DIV16
 HAL Driver, 93
STM32_PPREG1_DIV2
 HAL Driver, 92

STM32_PPREG1_DIV4
 HAL Driver, 92
STM32_PPREG1_DIV8
 HAL Driver, 92
STM32_PPREG1_MASK
 HAL Driver, 92
STM32_PPREG2
 HAL Driver, 105
STM32_PPREG2_DIV1
 HAL Driver, 93
STM32_PPREG2_DIV16
 HAL Driver, 93
STM32_PPREG2_DIV2
 HAL Driver, 93
STM32_PPREG2_DIV4
 HAL Driver, 93
STM32_PPREG2_DIV8
 HAL Driver, 93
STM32_PPREG2_MASK
 HAL Driver, 93
STM32_PVD_ENABLE
 HAL Driver, 103
STM32_PWM_TIM1_IRQ_PRIORITY
 PWM Driver, 179
STM32_PWM_TIM2_IRQ_PRIORITY
 PWM Driver, 179
STM32_PWM_TIM3_IRQ_PRIORITY
 PWM Driver, 179
STM32_PWM_TIM4_IRQ_PRIORITY
 PWM Driver, 179
STM32_PWM_TIM5_IRQ_PRIORITY
 PWM Driver, 179
STM32_PWM_TIM8_IRQ_PRIORITY
 PWM Driver, 179
STM32_PWM_USE_ADVANCED
 PWM Driver, 178
STM32_PWM_USE_TIM1
 PWM Driver, 178
STM32_PWM_USE_TIM2
 PWM Driver, 178
STM32_PWM_USE_TIM3
 PWM Driver, 178
STM32_PWM_USE_TIM4
 PWM Driver, 178
STM32_PWM_USE_TIM5
 PWM Driver, 178
STM32_PWM_USE_TIM8
 PWM Driver, 179
stm32_rcc.h, 407
STM32_RTC_HSE
 HAL Driver, 96
STM32_RTC_LSE
 HAL Driver, 95
STM32_RTC_LSI
 HAL Driver, 95
STM32_RTC_NO CLOCK
 HAL Driver, 95
STM32_RTCCLK
 HAL Driver, 109
STM32_RTCPRE
 HAL Driver, 108
STM32_RTCPRE_MASK
 HAL Driver, 93
STM32_RTCPRE_VALUE
 HAL Driver, 105
STM32_RTCSEL
 HAL Driver, 105
STM32_RTCSEL_HS DIV
 HAL Driver, 96
STM32_RTCSEL_LSE
 HAL Driver, 96
STM32_RTCSEL_LSI
 HAL Driver, 96
STM32_RTCSEL_MASK
 HAL Driver, 96
STM32_RTCSEL_NO CLOCK
 HAL Driver, 96
STM32_SERIAL_UART4_PRIORITY
 Serial Driver, 195
STM32_SERIAL_UART5_PRIORITY
 Serial Driver, 195
STM32_SERIAL_USART1_PRIORITY
 Serial Driver, 195
STM32_SERIAL_USART2_PRIORITY
 Serial Driver, 195
STM32_SERIAL_USART3_PRIORITY
 Serial Driver, 195
STM32_SERIAL_USART6_PRIORITY
 Serial Driver, 195
STM32_SERIAL_USE_UART4
 Serial Driver, 194
STM32_SERIAL_USE_UART5
 Serial Driver, 194
STM32_SERIAL_USE_USART1
 Serial Driver, 194
STM32_SERIAL_USE_USART2
 Serial Driver, 194
STM32_SERIAL_USE_USART3
 Serial Driver, 194
STM32_SERIAL_USE_USART6
 Serial Driver, 195
STM32_SPI_DMA_ERROR_HOOK
 SPI Driver, 225
STM32_SPI_SPI1_DMA_PRIORITY
 SPI Driver, 224
STM32_SPI_SPI1_IRQ_PRIORITY
 SPI Driver, 224
STM32_SPI_SPI1_RX_DMA_STREAM
 SPI Driver, 225
STM32_SPI_SPI1_TX_DMA_STREAM
 SPI Driver, 225
STM32_SPI_SPI2_DMA_PRIORITY
 SPI Driver, 224
STM32_SPI_SPI2_IRQ_PRIORITY
 SPI Driver, 224
STM32_SPI_SPI2_RX_DMA_STREAM
 SPI Driver, 225
STM32_SPI_SPI2_TX_DMA_STREAM

SPI Driver, 225
STM32_SPI_SPI3_DMA_PRIORITY
 SPI Driver, 224
STM32_SPI_SPI3_IRQ_PRIORITY
 SPI Driver, 224
STM32_SPI_SPI3_RX_DMA_STREAM
 SPI Driver, 225
STM32_SPI_SPI3_TX_DMA_STREAM
 SPI Driver, 225
STM32_SPI_USE_SPI1
 SPI Driver, 223
STM32_SPI_USE_SPI2
 SPI Driver, 224
STM32_SPI_USE_SPI3
 SPI Driver, 224
STM32_SPII2S_MAX
 HAL Driver, 89
STM32_SW
 HAL Driver, 104
STM32_SW_HSE
 HAL Driver, 91
STM32_SW_HSI
 HAL Driver, 91
STM32_SW_MASK
 HAL Driver, 91
STM32_SW_PLL
 HAL Driver, 91
STM32_SYSCLK
 HAL Driver, 107
STM32_SYSCLK_MAX
 HAL Driver, 106
stm32_tim_t, 340
STM32_TIMCLK1
 HAL Driver, 109
STM32_TIMCLK2
 HAL Driver, 109
STM32_UART_DMA_ERROR_HOOK
 UART Driver, 245
STM32_UART_USART1_DMA_PRIORITY
 UART Driver, 244
STM32_UART_USART1_IRQ_PRIORITY
 UART Driver, 244
STM32_UART_USART1_RX_DMA_STREAM
 UART Driver, 245
STM32_UART_USART1_TX_DMA_STREAM
 UART Driver, 245
STM32_UART_USART2_DMA_PRIORITY
 UART Driver, 244
STM32_UART_USART2_IRQ_PRIORITY
 UART Driver, 244
STM32_UART_USART2_RX_DMA_STREAM
 UART Driver, 245
STM32_UART_USART2_TX_DMA_STREAM
 UART Driver, 245
STM32_UART_USART3_DMA_PRIORITY
 UART Driver, 244
STM32_UART_USART3_IRQ_PRIORITY
 UART Driver, 244
STM32_UART_USART3_RX_DMA_STREAM
 UART Driver, 245
STM32_UART_USART3_TX_DMA_STREAM
 UART Driver, 245
STM32_VOS
 HAL Driver, 103
STM32_VOS_HIGH
 HAL Driver, 90
STM32_VOS_LOW
 HAL Driver, 89
STM32_VOS_MASK
 HAL Driver, 89
STM32F4xx ADC Support, 248
STM32F4xx CAN Support, 249
STM32F4xx DMA Support, 254
 _stm32_dma_streams, 262
 CH_IRQ_HANDLER, 259–261
 dmaInit, 261
 dmaStartMemcpy, 269
 dmaStreamAllocate, 261
 dmaStreamClearInterrupt, 269
 dmaStreamDisable, 268
 dmaStreamEnable, 268
 dmaStreamGetTransactionSize, 266
 dmaStreamRelease, 262
 dmaStreamSetFIFO, 267
 dmaStreamSetMemory0, 265
 dmaStreamSetMemory1, 265
 dmaStreamSetMode, 267
 dmaStreamSetPeripheral, 264
 dmaStreamSetTransactionSize, 266
 dmaWaitCompletion, 270
 STM32_DMA1_STREAMS_MASK, 263
 STM32_DMA2_STREAMS_MASK, 263
 STM32_DMA_CR_RESET_VALUE, 263
 STM32_DMA_FCR_RESET_VALUE, 263
 STM32_DMA_GETCHANNEL, 263
 STM32_DMA_IS_VALID_ID, 264
 STM32_DMA_ISR_MASK, 263
 STM32_DMA_STREAM, 264
 STM32_DMA_STREAM_ID, 264
 STM32_DMA_STREAM_ID_MSK, 264
 STM32_DMA_STREAMS, 263
 stm32_dmaisr_t, 270
STM32F4xx Drivers, 247
STM32F4xx EXT Support, 249
STM32F4xx GPT Support, 249
STM32F4xx ICU Support, 250
STM32F4xx Initialization Support, 248
STM32F4xx MAC Support, 250
STM32F4xx PAL Support, 250
STM32F4xx Platform Drivers, 254
STM32F4xx PWM Support, 252
STM32F4xx RCC Support, 271

rccDisableADC1, 281
rccDisableADC2, 281
rccDisableADC3, 282
rccDisableAHB1, 278
rccDisableAHB2, 279
rccDisableAHB3, 280
rccDisableAPB1, 276
rccDisableAPB2, 277
rccDisableCAN1, 284
rccDisableCAN2, 285
rccDisableDMA1, 283
rccDisableDMA2, 283
rccDisableETH, 286
rccDisableI2C1, 286
rccDisableI2C2, 287
rccDisableI2C3, 287
rccDisableOTG_FS, 288
rccDisablePWRInterface, 284
rccDisableSDIO, 289
rccDisableSPI1, 289
rccDisableSPI2, 290
rccDisableSPI3, 290
rccDisableTIM1, 291
rccDisableTIM2, 292
rccDisableTIM3, 292
rccDisableTIM4, 293
rccDisableTIM5, 294
rccDisableTIM8, 294
rccDisableUART4, 297
rccDisableUART5, 298
rccDisableUSART1, 295
rccDisableUSART2, 295
rccDisableUSART3, 296
rccDisableUSART6, 297
rccEnableADC1, 281
rccEnableADC2, 281
rccEnableADC3, 282
rccEnableAHB1, 278
rccEnableAHB2, 279
rccEnableAHB3, 280
rccEnableAPB1, 276
rccEnableAPB2, 277
rccEnableCAN1, 284
rccEnableCAN2, 285
rccEnableDMA1, 282
rccEnableDMA2, 283
rccEnableETH, 285
rccEnableI2C1, 286
rccEnableI2C2, 287
rccEnableI2C3, 287
rccEnableOTG_FS, 288
rccEnablePWRInterface, 283
rccEnableSDIO, 288
rccEnableSPI1, 289
rccEnableSPI2, 290
rccEnableSPI3, 290
rccEnableTIM1, 291
rccEnableTIM2, 292
rccEnableTIM3, 292
rccEnableTIM4, 293
rccEnableTIM5, 294
rccEnableTIM8, 294
rccEnableUART4, 297
rccEnableUART5, 298
rccEnableUSART1, 295
rccEnableUSART2, 295
rccEnableUSART3, 296
rccEnableUSART6, 296
rccResetADC1, 281
rccResetADC2, 282
rccResetADC3, 282
rccResetAHB1, 278
rccResetAHB2, 279
rccResetAHB3, 280
rccResetAPB1, 276
rccResetAPB2, 277
rccResetCAN1, 284
rccResetCAN2, 285
rccResetDMA1, 283
rccResetDMA2, 283
rccResetETH, 286
rccResetI2C1, 286
rccResetI2C2, 287
rccResetI2C3, 288
rccResetOTG_FS, 288
rccResetPWRInterface, 284
rccResetSDIO, 289
rccResetSPI1, 290
rccResetSPI2, 290
rccResetSPI3, 291
rccResetTIM1, 291
rccResetTIM2, 292
rccResetTIM3, 293
rccResetTIM4, 293
rccResetTIM5, 294
rccResetTIM8, 295
rccResetUART4, 297
rccResetUART5, 298
rccResetUSART1, 295
rccResetUSART2, 296
rccResetUSART3, 296
rccResetUSART6, 298
STM32F4xx SDC Support, 252
STM32F4xx Serial Support, 252
STM32F4xx SPI Support, 253
STM32F4xx UART Support, 254
stop
 TimeMeasurement, 340
stream
 stm32_dma_stream_t, 338
TAMP_STAMP_IRQHandler
 HAL Driver, 96
thread
 ADCDriver, 304
 SPIDriver, 337
tim
 GPTDriver, 313

ICUDriver, 317
PWMDriver, 331
TIM1_BRK_IRQHandler
 HAL Driver, 98
TIM1_CC_IRQHandler
 HAL Driver, 98
TIM1_TRG_COM_IRQHandler
 HAL Driver, 98
TIM1_UP_IRQHandler
 HAL Driver, 98
TIM2_IRQHandler
 HAL Driver, 99
TIM3_IRQHandler
 HAL Driver, 99
TIM4_IRQHandler
 HAL Driver, 99
TIM5_IRQHandler
 HAL Driver, 100
TIM6_IRQHandler
 HAL Driver, 101
TIM7_IRQHandler
 HAL Driver, 101
TIM8_BRK_IRQHandler
 HAL Driver, 100
TIM8_CC_IRQHandler
 HAL Driver, 100
TIM8_TRG_COM_IRQHandler
 HAL Driver, 100
TIM8_UP_IRQHandler
 HAL Driver, 100
Time Measurement Driver., 226
 TimeMeasurement, 228
 tmInit, 227
 tmObjectInit, 227
 tmStartMeasurement, 227
 tmStopMeasurement, 228
TimeMeasurement, 340
 best, 340
 last, 340
 start, 340
 stop, 340
 Time Measurement Driver., 228
 worst, 340
tm.c, 412
tm.h, 412
tmInit
 Time Measurement Driver., 227
tmObjectInit
 Time Measurement Driver., 227
tmStartMeasurement
 Time Measurement Driver., 227
tmStopMeasurement
 Time Measurement Driver., 228
txdmamode
 SPIDriver, 337
txend1_cb
 UARTConfig, 342
txend2_cb
 UARTConfig, 342
txstate
 UARTDriver, 345
UART Driver, 228
 CH_IRQ_HANDLER, 240
 STM32_UART_DMA_ERROR_HOOK, 245
 STM32_UART_USART1_DMA_PRIORITY, 244
 STM32_UART_USART1_IRQ_PRIORITY, 244
 STM32_UART_USART1_RX_DMA_STREAM, 245
 STM32_UART_USART1_TX_DMA_STREAM, 245
 STM32_UART_USART2_DMA_PRIORITY, 244
 STM32_UART_USART2_IRQ_PRIORITY, 244
 STM32_UART_USART2_RX_DMA_STREAM, 245
 STM32_UART_USART2_TX_DMA_STREAM, 245
 STM32_UART_USART3_DMA_PRIORITY, 244
 STM32_UART_USART3_IRQ_PRIORITY, 244
 STM32_UART_USART3_RX_DMA_STREAM, 245
 STM32_UART_USART3_TX_DMA_STREAM, 246
 STM32_UART_USE_USART1, 243
 STM32_UART_USE_USART2, 243
 STM32_UART_USE_USART3, 244
 UART_READY, 247
 UART_RX_ACTIVE, 247
 UART_RX_COMPLETE, 247
 UART_RX_IDLE, 247
 UART_STOP, 247
 UART_TX_ACTIVE, 247
 UART_TX_COMPLETE, 247
 UART_TX_IDLE, 247
 UART_UNINIT, 247
 UART_BREAK_DETECTED, 243
 UART_FRAMING_ERROR, 243
 uart_lld_init, 240
 uart_lld_start, 240
 uart_lld_start_receive, 242
 uart_lld_start_send, 241
 uart_lld_stop, 241
 uart_lld_stop_receive, 242
 uart_lld_stop_send, 241
 UART_NO_ERROR, 243
 UART_NOISE_ERROR, 243
 UART_OVERRUN_ERROR, 243
 UART_PARITY_ERROR, 243
 uartcb_t, 246
 uartccb_t, 246
 UARTD1, 242
 UARTD2, 243
 UARTD3, 243
 UARTDriver, 246
 uartecb_t, 246
 uartlags_t, 246
 uartInit, 233
 uartObjectInit, 233
 uartrxstate_t, 247
 uartStart, 234
 uartStartReceive, 237
 uartStartReceive1, 238
 uartStartSend, 235
 uartStartSend1, 235

uartstate_t, 247
uartStop, 234
uartStopReceive, 238
uartStopReceive1, 239
uartStopSend, 236
uartStopSend1, 236
uarttxstate_t, 247
uart.c, 413
uart.h, 413
UART4_IRQHandler
 HAL Driver, 101
UART5_IRQHandler
 HAL Driver, 101
UART_READY
 UART Driver, 247
UART_RX_ACTIVE
 UART Driver, 247
UART_RX_COMPLETE
 UART Driver, 247
UART_RX_IDLE
 UART Driver, 247
UART_STOP
 UART Driver, 247
UART_TX_ACTIVE
 UART Driver, 247
UART_TX_COMPLETE
 UART Driver, 247
UART_TX_IDLE
 UART Driver, 247
UART_UNINIT
 UART Driver, 247
UART_BREAK_DETECTED
 UART Driver, 243
UART_FRAMING_ERROR
 UART Driver, 243
uart_lld.c, 415
uart_lld.h, 415
uart_lld_init
 UART Driver, 240
uart_lld_start
 UART Driver, 240
uart_lld_start_receive
 UART Driver, 242
uart_lld_start_send
 UART Driver, 241
uart_lld_stop
 UART Driver, 241
uart_lld_stop_receive
 UART Driver, 242
uart_lld_stop_send
 UART Driver, 241
UART_NO_ERROR
 UART Driver, 243
UART_NOISE_ERROR
 UART Driver, 243
UART_OVERRUN_ERROR
 UART Driver, 243
UART_PARITY_ERROR
 UART Driver, 243
uartcb_t
 UART Driver, 246
uartccb_t
 UART Driver, 246
UARTConfig, 341
 cr1, 342
 cr2, 342
 cr3, 343
 rxchar_cb, 342
 rxend_cb, 342
 rxerr_cb, 342
 speed, 342
 txend1_cb, 342
 txend2_cb, 342
UARTD1
 UART Driver, 242
UARTD2
 UART Driver, 243
UARTD3
 UART Driver, 243
UARTDriver, 343
 config, 345
 dmemode, 345
 dmarx, 345
 dmatx, 345
 rdbuf, 345
 rxstate, 345
 state, 345
 txstate, 345
 UART Driver, 246
 usart, 345
uartecb_t
 UART Driver, 246
uartflags_t
 UART Driver, 246
uartInit
 UART Driver, 233
uartObjectInit
 UART Driver, 233
uartrxstate_t
 UART Driver, 247
uartStart
 UART Driver, 234
uartStartReceive
 UART Driver, 237
uartStartReceive1
 UART Driver, 238
uartStartSend
 UART Driver, 235
uartStartSend1
 UART Driver, 235
uartstate_t
 UART Driver, 247
uartStop
 UART Driver, 234
uartStopReceive
 UART Driver, 238
uartStopReceive1
 UART Driver, 239

uartStopSend
 UART Driver, [236](#)
uartStopSendl
 UART Driver, [236](#)
uarttxstate_t
 UART Driver, [247](#)
US2RTT
 HAL Driver, [87](#)
usart
 UARTDriver, [345](#)
USART1_IRQHandler
 HAL Driver, [99](#)
USART2_IRQHandler
 HAL Driver, [99](#)
USART3_IRQHandler
 HAL Driver, [99](#)
USART6_IRQHandler
 HAL Driver, [102](#)
USART_CR2_STOP0P5_BITS
 Serial Driver, [196](#)
USART_CR2_STOP1_BITS
 Serial Driver, [196](#)
USART_CR2_STOP1P5_BITS
 Serial Driver, [196](#)
USART_CR2_STOP2_BITS
 Serial Driver, [196](#)

vector
 stm32_dma_stream_t, [338](#)

vmt
 SerialDriver, [333](#)

vt
 MMCDriver, [322](#)

width_cb
 ICUConfig, [315](#)

worst
 TimeMeasurement, [340](#)

WWDG_IRQHandler
 HAL Driver, [96](#)