

库函数使用说明

1 结构体索引	1
1.1 结构体	1
2 文件索引	3
2.1 文件列表	3
3 结构体说明	5
3.1 CMP_TypeDef结构体 参考	5
3.1.1 详细描述	5
3.1.2 结构体成员变量说明	5
3.1.2.1 BLCWIN	5
3.1.2.2 CFG	6
3.1.2.3 IE	6
3.1.2.4 IF	6
3.1.2.5 TCLK	6
3.2 CRC_Model.Params结构体 参考	6
3.2.1 详细描述	6
3.2.2 结构体成员变量说明	7
3.2.2.1 initValue	7
3.2.2.2 polynomial	7
3.2.2.3 reflectIn	7
3.2.2.4 reflectOut	7
3.2.2.5 width	7
3.2.2.6 xorOut	7
3.3 HAL_ADC_TypeDef结构体 参考	7
3.3.1 详细描述	8
3.3.2 结构体成员变量说明	9
3.3.2.1 AMC_A0	9
3.3.2.2 AMC_A1	9
3.3.2.3 AMC_B0	9
3.3.2.4 AMC_B1	9
3.3.2.5 CFG	9
3.3.2.6 CHN	9
3.3.2.7 CHNT0	10
3.3.2.8 CHNT1	10
3.3.2.9 DAT	10
3.3.2.10 DAT0_TH	10
3.3.2.11 DC_A0	10
3.3.2.12 DC_A1	10
3.3.2.13 DC_B0	11
3.3.2.14 DC_B1	11
3.3.2.15 GAIN_0	11
3.3.2.16 GAIN_1	11

3.3.2.17 IE	11
3.3.2.18 IF	11
3.3.2.19 null	12
3.3.2.20 SWT	12
3.3.2.21 TRIG	12
3.4 HAL_CAN_TypeDef结构体 参考	12
3.4.1 详细描述	13
3.4.2 结构体成员变量说明	14
3.4.2.1 ALC	14
3.4.2.2 BTR0	14
3.4.2.3 BTR1	14
3.4.2.4 CMR	14
3.4.2.5 ECC	14
3.4.2.6 EWLR	14
3.4.2.7 IER	15
3.4.2.8 IR	15
3.4.2.9 MOD	15
3.4.2.10 null0	15
3.4.2.11 null1	15
3.4.2.12 null2	15
3.4.2.13 null3	15
3.4.2.14 RBSA	16
3.4.2.15 RMC	16
3.4.2.16 RXERR	16
3.4.2.17 SR	16
3.4.2.18 TXERR	16
3.4.2.19 TXRX0	16
3.4.2.20 TXRX1	17
3.4.2.21 TXRX2	17
3.4.2.22 TXRX3	17
3.4.2.23 TXRX4	17
3.4.2.24 TXRX5	17
3.4.2.25 TXRX6	17
3.4.2.26 TXRX7	18
3.4.2.27 TXRX8	18
3.4.2.28 TXRX9	18
3.4.2.29 TXRXA	18
3.4.2.30 TXRXB	18
3.4.2.31 TXRXC	18
3.5 HAL_CMP_TypeDef结构体 参考	19
3.5.1 详细描述	19
3.5.2 结构体成员变量说明	19

3.5.2.1 BLCWIN	19
3.5.2.2 CFG	19
3.5.2.3 DATA	20
3.5.2.4 IE	20
3.5.2.5 IF	20
3.5.2.6 TCLK	20
3.6 HAL_CRC_TypeDef结构体 参考	20
3.6.1 详细描述	21
3.6.2 结构体成员变量说明	21
3.6.2.1 CR	21
3.6.2.2 DR	21
3.6.2.3 INIT	21
3.6.2.4 POL	21
3.7 HAL_DMA_TypeDef结构体 参考	21
3.7.1 详细描述	22
3.7.2 结构体成员变量说明	22
3.7.2.1 CCR	22
3.7.2.2 CMAR	22
3.7.2.3 CPAR	22
3.7.2.4 CTMS	22
3.8 HAL_DSP_TypeDef结构体 参考	23
3.8.1 详细描述	23
3.8.2 结构体成员变量说明	23
3.8.2.1 ATAN	23
3.8.2.2 COS	24
3.8.2.3 MOD	24
3.8.2.4 RAD	24
3.8.2.5 RESERVED	24
3.8.2.6 RESERVED1	24
3.8.2.7 SC	24
3.8.2.8 SIN	25
3.8.2.9 SQRT	25
3.8.2.10 THETA	25
3.8.2.11 XY	25
3.9 HAL_FLASH_TypeDef结构体 参考	25
3.9.1 详细描述	26
3.9.2 结构体成员变量说明	26
3.9.2.1 ADDR	26
3.9.2.2 CFG	26
3.9.2.3 ERASE	26
3.9.2.4 PROTECT	26
3.9.2.5 RDATA	26

3.9.2.6 READY	27
3.9.2.7 WDATA	27
3.10 HAL_GPIO_TypeDef结构体 参考	27
3.10.1 详细描述	28
3.10.2 结构体成员变量说明	28
3.10.2.1 F3210	28
3.10.2.2 F7654	28
3.10.2.3 FBA98	28
3.10.2.4 FFEDC	28
3.10.2.5 LCKR	28
3.10.2.6 null1	29
3.10.2.7 PDI	29
3.10.2.8 PDO	29
3.10.2.9 PIE	29
3.10.2.10 PODE	29
3.10.2.11 POE	29
3.10.2.12 PUE	30
3.11 HAL_HALL_TypeDef结构体 参考	30
3.11.1 详细描述	30
3.11.2 结构体成员变量说明	30
3.11.2.1 CFG	30
3.11.2.2 CNT	31
3.11.2.3 INFO	31
3.11.2.4 TH	31
3.11.2.5 WIDTH	31
3.12 HAL_I2C_TypeDef结构体 参考	31
3.12.1 详细描述	32
3.12.2 结构体成员变量说明	32
3.12.2.1 ADDR	32
3.12.2.2 BCR	32
3.12.2.3 CFG	32
3.12.2.4 DATA	32
3.12.2.5 MSCR	32
3.12.2.6 SCR	33
3.13 HAL_IWDG_TypeDef结构体 参考	33
3.13.1 详细描述	33
3.13.2 结构体成员变量说明	33
3.13.2.1 WDT_CLR	33
3.14 HAL_MCPWM_TypeDef结构体 参考	33
3.14.1 详细描述	34
3.14.2 结构体成员变量说明	34
3.14.2.1 CNT	34

3.14.2.2 DTH00	35
3.14.2.3 DTH01	35
3.14.2.4 DTH10	35
3.14.2.5 DTH11	35
3.14.2.6 DTH20	35
3.14.2.7 DTH21	35
3.14.2.8 DTH30	35
3.14.2.9 DTH31	35
3.14.2.10 EIE	36
3.14.2.11 EIF	36
3.14.2.12 FAIL	36
3.14.2.13 FILT	36
3.14.2.14 IE0	36
3.14.2.15 IF0	36
3.14.2.16 IO01	36
3.14.2.17 IO23	36
3.14.2.18 null1	37
3.14.2.19 null2	37
3.14.2.20 null3	37
3.14.2.21 PP	37
3.14.2.22 PRT	37
3.14.2.23 RE	37
3.14.2.24 SDCFG	37
3.14.2.25 TCLK	37
3.14.2.26 TH	38
3.14.2.27 TH0	38
3.14.2.28 TMR0	38
3.14.2.29 TMR1	38
3.14.2.30 TMR2	38
3.14.2.31 TMR3	38
3.14.2.32 UPDATE	38
3.15 HAL_QEP_TypeDef结构体 参考	38
3.15.1 详细描述	39
3.15.2 结构体成员变量说明	39
3.15.2.1 CFG	39
3.15.2.2 CNT	39
3.15.2.3 TH	39
3.16 HAL_SPI_TypeDef结构体 参考	39
3.16.1 详细描述	40
3.16.2 结构体成员变量说明	40
3.16.2.1 BAUD	40
3.16.2.2 CFG	40

3.16.2.3 IE	40
3.16.2.4 RXDATA	41
3.16.2.5 SIZE	41
3.16.2.6 TXDATA	41
3.17 HAL_SYS_TypeDef结构体 参考	41
3.17.1 详细描述	43
3.17.2 结构体成员变量说明	43
3.17.2.1 AFE_CMP	43
3.17.2.2 AFE_DAC	43
3.17.2.3 AFE_DAC_AMC	43
3.17.2.4 AFE_DAC_DC	43
3.17.2.5 AFE_REG0	44
3.17.2.6 AFE_REG1	44
3.17.2.7 AFE_REG2	44
3.17.2.8 AFE_REG3	44
3.17.2.9 AFE_REG4	44
3.17.2.10 AFE_REG5	44
3.17.2.11 AFE_REG6	45
3.17.2.12 AFE_REG7	45
3.17.2.13 AFE_REG8	45
3.17.2.14 AFE_REG9	45
3.17.2.15 AFE_REGA	45
3.17.2.16 AFE_REGB	45
3.17.2.17 AFE_REGC	46
3.17.2.18 CLK_CFG	46
3.17.2.19 CLK_DIV0	46
3.17.2.20 CLK_DIV2	46
3.17.2.21 CLK_FEN	46
3.17.2.22 CLK_SLP	46
3.17.2.23 CLR_RST	47
3.17.2.24 PROTECT	47
3.17.2.25 RESERVED	47
3.17.2.26 RESERVED1	47
3.17.2.27 RESERVED2	47
3.17.2.28 RESERVED3	47
3.17.2.29 RESERVED4	47
3.17.2.30 RESERVED5	48
3.17.2.31 RST_CFG	48
3.17.2.32 RST_SRC	48
3.17.2.33 SFT_RST	48
3.17.2.34 TMP_A	48
3.17.2.35 TMP_B	48

3.17.2.36 TRIM	49
3.17.2.37 WDT_CLR	49
3.17.2.38 WDT_CNT	49
3.17.2.39 WDT_PSW	49
3.17.2.40 WDT_TH	49
3.18 HAL_TIMER_TypeDef结构体 参考	49
3.18.1 详细描述	50
3.18.2 结构体成员变量说明	50
3.18.2.1 CFG	50
3.18.2.2 CMPT0	50
3.18.2.3 CMPT1	50
3.18.2.4 CNT	51
3.18.2.5 EVT	51
3.18.2.6 TH	51
3.19 HAL_UART_TypeDef结构体 参考	51
3.19.1 详细描述	52
3.19.2 结构体成员变量说明	52
3.19.2.1 ADR	52
3.19.2.2 BUFF	52
3.19.2.3 CTRL	52
3.19.2.4 DIVH	52
3.19.2.5 DIVL	52
3.19.2.6 IE	53
3.19.2.7 IF	53
3.19.2.8 IOC	53
3.19.2.9 STT	53
4 文件说明	55
4.1 hal_lks32mc08x.c 文件参考	55
4.1.1 宏定义说明	56
4.1.1.1 __HAL_LKS32MC08x_C__	56
4.2 hal_lks32mc08x.c	56
4.3 hal_lks32mc08x.h 文件参考	56
4.4 hal_lks32mc08x.h	57
4.5 hal_lksmcu.h 文件参考	58
4.6 hal_lksmcu.h	58
4.7 include/hal_lks32mc08x_reg.h 文件参考	58
4.7.1 宏定义说明	60
4.7.1.1 HAL_ADC_TRIG_STATE_ADC0_EN	60
4.7.1.2 HAL_ADC_TRIG_STATE_ADC1_EN	60
4.7.1.3 HAL_ADC_TRIG_STATE_MCPWM_TRIG_EN	60
4.7.1.4 HAL_ADC_TRIG_STATE_TIMER_TRIG_EN	60

4.7.1.5 HAL_ADC_TRIG_STATE_TWO_SEG_TRIG_EN	61
4.7.1.6 HAL_LKS32MC08X_REG	61
4.7.1.7 HAL_MCPWM_STATE_H_DEFAULT_BIT	61
4.7.1.8 HAL_MCPWM_STATE_H_MOS_EN_BIT	61
4.7.1.9 HAL_MCPWM_STATE_L_DEFAULT_BIT	61
4.7.1.10 HAL_MCPWM_STATE_L_MOS_EN_BIT	61
4.7.1.11 HAL_MCPWM_STATE_STATE	61
4.7.1.12 HAL_MCPWM_STATE_STATE_MASK	62
4.7.1.13 HAL_MCPWM_STATE_STATE_MAX	62
4.7.1.14 HAL_MCPWM_STATE_STATE_POS	62
4.7.1.15 HAL_MCPWM_STATE_SW_BIT	62
4.7.2 变量说明	62
4.7.2.1 HAL_ADC	62
4.7.2.2 HAL_CAN	62
4.7.2.3 HAL_CMP	62
4.7.2.4 HAL_CRC	63
4.7.2.5 HAL_DMA	63
4.7.2.6 HAL_DSP	63
4.7.2.7 HAL_FLASH	63
4.7.2.8 HAL_GPIO	63
4.7.2.9 HAL_HALL	63
4.7.2.10 HAL_I2C	63
4.7.2.11 HAL_IWDG	63
4.7.2.12 HAL_MCPWM	64
4.7.2.13 HAL_QEP	64
4.7.2.14 HAL_SPI	64
4.7.2.15 HAL_SYS	64
4.7.2.16 HAL_TIMER	64
4.7.2.17 HAL_UART	64
4.8 hal_lks32mc08x_reg.h	65
4.9 include/hal_lksmcu_adc.h 文件参考	68
4.9.1 枚举类型说明	71
4.9.1.1 HAL_ADC_CHANNEL_x	71
4.9.1.2 HAL_ADC_REFERENCE_x	71
4.9.1.3 HAL_ADC_SAMP_NUMBER_x	72
4.9.1.4 HAL_ADC_SAMP_SEQUENCE_x	72
4.9.1.5 HAL_ADC_TRIGGER_x	73
4.9.1.6 HAL_ADC_x	73
4.9.2 函数说明	74
4.9.2.1 HAL_ADC_Disable()	74
4.9.2.2 HAL_ADC_Enable()	74
4.9.2.3 HAL_ADC_EndSampling()	74

4.9.2.4 HAL_ADC_GetRange()	75
4.9.2.5 HAL_ADC_GetSamplingValues()	76
4.9.2.6 HAL_ADC_GetValuesUnit()	76
4.9.2.7 HAL_ADC_ManualSampling()	77
4.9.2.8 HAL_ADC_Reset()	77
4.9.2.9 HAL_ADC_SetGpioIn()	78
4.9.2.10 HAL_ADC_SetGpioTriggerOut()	78
4.9.2.11 HAL_ADC_SetReference()	79
4.9.2.12 HAL_ADC_SetSamplingChannel()	80
4.9.2.13 HAL_ADC_SetSamplingQuantity()	80
4.9.2.14 HAL_ADC_SetTriggerSignal()	81
4.9.2.15 HAL_ADC_SoftTrigger()	82
4.9.2.16 HAL_ADC_StateReset()	82
4.10 hal_lksmcu_adc.h	83
4.11 include/hal_lksmcu_can.h 文件参考	84
4.11.1 枚举类型说明	87
4.11.1.1 HAL_CAN_AE_x	87
4.11.1.2 HAL_CAN_ERROR_x	87
4.11.1.3 HAL_CAN_FRAME_x	87
4.11.1.4 HAL_CAN_IDEFilter_x	88
4.11.1.5 HAL_CAN_STBMode_x	88
4.11.1.6 HAL_CAN_STBTrig_x	88
4.11.1.7 HAL_CAN_TSSTAT_x	89
4.11.1.8 HAL_CAN_x	89
4.11.2 函数说明	89
4.11.2.1 HAL_CAN_ArbitrationLostCap()	89
4.11.2.2 HAL_CAN_Busoff()	89
4.11.2.3 HAL_CAN_Disable()	89
4.11.2.4 HAL_CAN_Enable()	90
4.11.2.5 HAL_CAN_ErrorPassive()	90
4.11.2.6 HAL_CAN_GetBaud()	90
4.11.2.7 HAL_CAN_GetClockDiv()	90
4.11.2.8 HAL_CAN_GetClockFreq()	91
4.11.2.9 HAL_CAN_IDx_Filter()	91
4.11.2.10 HAL_CAN_IDx_FilterDisable()	92
4.11.2.11 HAL_CAN_IDx_FilterEnable()	92
4.11.2.12 HAL_CAN_KindOfError()	92
4.11.2.13 HAL_CAN_LBME_Disable()	92
4.11.2.14 HAL_CAN_LBME_Enable()	92
4.11.2.15 HAL_CAN_LBML_Disable()	92
4.11.2.16 HAL_CAN_LBML_Enable()	93
4.11.2.17 HAL_CAN_LOM_Disable()	93

4.11.2.18 HAL_CAN_LOM_Enable()	93
4.11.2.19 HAL_CAN_Reset()	93
4.11.2.20 HAL_CAN_ROverflow()	93
4.11.2.21 HAL_CAN_RxData()	94
4.11.2.22 HAL_CAN_RxDataBuf()	94
4.11.2.23 HAL_CAN_RxDataFdf()	94
4.11.2.24 HAL_CAN_RxDataID()	94
4.11.2.25 HAL_CAN_RxDataIde()	94
4.11.2.26 HAL_CAN_RxDataLen()	94
4.11.2.27 HAL_CAN_RxDataRtr()	94
4.11.2.28 HAL_CAN_RxErrorCnt()	95
4.11.2.29 HAL_CAN_SetBaud()	95
4.11.2.30 HAL_CAN_SetClockDiv()	95
4.11.2.31 HAL_CAN_SetFastBaud()	95
4.11.2.32 HAL_CAN_SetGpioRx()	96
4.11.2.33 HAL_CAN_SetGpioTx()	96
4.11.2.34 HAL_CAN_SetSTBMode()	96
4.11.2.35 HAL_CAN_SetTDC()	97
4.11.2.36 HAL_CAN_SS_Disable()	97
4.11.2.37 HAL_CAN_SS_Enable()	97
4.11.2.38 HAL_CAN_STBFull()	97
4.11.2.39 HAL_CAN_TSSStat()	97
4.11.2.40 HAL_CAN_TxCancel()	97
4.11.2.41 HAL_CAN_TxData()	98
4.11.2.42 HAL_CAN_TxData_STBTrig()	98
4.11.2.43 HAL_CAN_TxData_STBWrite()	98
4.11.2.44 HAL_CAN_TxErrorCnt()	98
4.11.2.45 HAL_CAN_Txing()	99
4.12 hal_lksmcu_can.h	99
4.13 include/hal_lksmcu_clk.h 文件参考	101
4.13.1 枚举类型说明	102
4.13.1.1 HAL_CLK_FREQ_x	102
4.13.2 函数说明	102
4.13.2.1 HAL_CLK_GetMcuFreq()	102
4.13.2.2 HAL_CLK_SetMcuFreq()	103
4.14 hal_lksmcu_clk.h	104
4.15 include/hal_lksmcu_cmp.h 文件参考	105
4.15.1 枚举类型说明	106
4.15.1.1 HAL_CMP_CHANNEL_INx	106
4.15.1.2 HAL_CMP_CHANNEL_IPx	106
4.15.1.3 HAL_CMP_HYS_x	107
4.15.1.4 HAL_CMP_MODE_x	107

4.15.1.5 HAL_CMP_x	107
4.15.2 函数说明	108
4.15.2.1 HAL_CMP_Disable()	108
4.15.2.2 HAL_CMP_Enable()	108
4.15.2.3 HAL_CMP_GetData()	109
4.15.2.4 HAL_CMP_Reset()	109
4.15.2.5 HAL_CMP_SetChannel()	110
4.15.2.6 HAL_CMP_SetDigitalFilter()	110
4.15.2.7 HAL_CMP_SetGpioIn()	110
4.15.2.8 HAL_CMP_SetGpioOut()	111
4.15.2.9 HAL_CMP_SetHysteresis()	112
4.15.2.10 HAL_CMP_SetMode()	112
4.16 hal_lksmcu_cmp.h	112
4.17 include/hal_lksmcu_crc.h 文件参考	114
4.17.1 枚举类型说明	114
4.17.1.1 HAL_CRC_MODEL_x	114
4.17.1.2 HAL_CRC_x	115
4.17.2 函数说明	115
4.17.2.1 HAL_CRC_Calc()	115
4.17.2.2 HAL_CRC_Disable()	116
4.17.2.3 HAL_CRC_Enable()	116
4.17.2.4 HAL_CRC_Reset()	117
4.18 hal_lksmcu_crc.h	117
4.19 include/hal_lksmcu_dac.h 文件参考	117
4.19.1 枚举类型说明	119
4.19.1.1 HAL_DAC_RANGE_x	119
4.19.1.2 HAL_DAC_x	119
4.19.2 函数说明	119
4.19.2.1 HAL_DAC_Disable()	119
4.19.2.2 HAL_DAC_Enable()	120
4.19.2.3 HAL_DAC_GetOutValMax()	120
4.19.2.4 HAL_DAC_GetRange()	120
4.19.2.5 HAL_DAC_Out2GpioDisable()	121
4.19.2.6 HAL_DAC_Out2GpioEnable()	121
4.19.2.7 HAL_DAC_Reset()	121
4.19.2.8 HAL_DAC_SetGpioOut()	122
4.19.2.9 HAL_DAC_SetOutVal()	123
4.19.2.10 HAL_DAC_SetOutVoltageV()	123
4.19.2.11 HAL_DAC_SetRange()	124
4.20 hal_lksmcu_dac.h	124
4.21 include/hal_lksmcu_dma.h 文件参考	125
4.21.1 枚举类型说明	126

4.21.1.1 HAL_DMA_x	126
4.21.2 函数说明	126
4.21.2.1 HAL_DMA_Disable()	126
4.21.2.2 HAL_DMA_Enable()	127
4.21.2.3 HAL_DMA_I2cRx()	127
4.21.2.4 HAL_DMA_I2cTx()	127
4.21.2.5 HAL_DMA_MemCopy()	127
4.21.2.6 HAL_DMA_Reset()	127
4.21.2.7 HAL_DMA_SpiRx()	128
4.21.2.8 HAL_DMA_SpiRxTx()	128
4.21.2.9 HAL_DMA_SpiTx()	129
4.21.2.10 HAL_DMA_UartRx()	129
4.21.2.11 HAL_DMA_UartTx()	129
4.22 hal_lksmcu_dma.h	129
4.23 include/hal_lksmcu_dsp.h 文件参考	130
4.23.1 枚举类型说明	131
4.23.1.1 HAL_DSP_x	131
4.23.2 函数说明	131
4.23.2.1 HAL_DSP_Cos()	131
4.23.2.2 HAL_DSP_Disable()	132
4.23.2.3 HAL_DSP_Div()	132
4.23.2.4 HAL_DSP_Enable()	133
4.23.2.5 HAL_DSP_Reset()	133
4.23.2.6 HAL_DSP_Sin()	133
4.23.2.7 HAL_DSP_SinCos()	134
4.23.2.8 HAL_DSP_Sqrt()	134
4.24 hal_lksmcu_dsp.h	134
4.25 include/hal_lksmcu_flash.h 文件参考	135
4.25.1 枚举类型说明	136
4.25.1.1 HAL_FLASH_x	136
4.25.2 函数说明	136
4.25.2.1 HAL_FLASH_Disable()	136
4.25.2.2 HAL_FLASH_DisableCache()	137
4.25.2.3 HAL_FLASH_Enable()	137
4.25.2.4 HAL_FLASH_EnableCache()	137
4.25.2.5 HAL_FLASH_Erase()	138
4.25.2.6 HAL_FLASH_Program()	138
4.25.2.7 HAL_FLASH_Read()	139
4.25.2.8 HAL_FLASH_ReadSectorSize()	140
4.25.2.9 HAL_FLASH_ReadUint16()	140
4.25.2.10 HAL_FLASH_ReadUint32()	141
4.25.2.11 HAL_FLASH_ReadUint8()	141

4.25.2.12 HAL_FLASH_Reset()	142
4.25.2.13 HAL_FLASH_SecurityParameterRead()	142
4.25.2.14 HAL_FLASH_SecurityParameterSave()	143
4.25.2.15 HAL_FLASH_Update()	144
4.26 hal_lksmcu_flash.h	144
4.27 include/hal_lksmcu_gpio.h 文件参考	145
4.27.1 枚举类型说明	147
4.27.1.1 HAL_GPIO_AFx	147
4.27.1.2 HAL_GPIO_MODE_x	148
4.27.1.3 HAL_GPIO_Px	149
4.27.2 函数说明	151
4.27.2.1 HAL_GPIO_Disable()	151
4.27.2.2 HAL_GPIO_Enable()	151
4.27.2.3 HAL_GPIO_GetInData()	151
4.27.2.4 HAL_GPIO_GetOutData()	152
4.27.2.5 HAL_GPIO_Invert()	152
4.27.2.6 HAL_GPIO_Reset()	153
4.27.2.7 HAL_GPIO_RstReuseGpioDisable()	153
4.27.2.8 HAL_GPIO_RstReuseGpioEnable()	153
4.27.2.9 HAL_GPIO_SetAf()	153
4.27.2.10 HAL_GPIO_SetHigh()	155
4.27.2.11 HAL_GPIO_SetLow()	155
4.27.2.12 HAL_GPIO_SetMode()	155
4.27.2.13 HAL_GPIO_SwdReuseGpioDisable()	156
4.27.2.14 HAL_GPIO_SwdReuseGpioEnable()	157
4.27.2.15 HAL_GPIO_Write()	157
4.28 hal_lksmcu_gpio.h	157
4.29 include/hal_lksmcu_hall.h 文件参考	159
4.29.1 枚举类型说明	161
4.29.1.1 HAL_HALL_x	161
4.29.2 函数说明	161
4.29.2.1 HAL_HALL_Disable()	161
4.29.2.2 HAL_HALL_Enable()	161
4.29.2.3 HAL_HALL_GetCNT()	162
4.29.2.4 HAL_HALL_ReadHallData()	162
4.29.2.5 HAL_HALL_ReadHallWidth()	162
4.29.2.6 HAL_HALL_ReadTh()	163
4.29.2.7 HAL_HALL_Reset()	163
4.29.2.8 HAL_HALL_SetFilter()	163
4.29.2.9 HAL_HALL_SetFilter75()	164
4.29.2.10 HAL_HALL_SetFilteruS()	164
4.29.2.11 HAL_HALL_SetFreq()	165

4.29.2.12 HAL_HALL_SetGpio()	165
4.29.2.13 HAL_HALL_SetTh()	166
4.29.2.14 HAL_HALL_StartCounting()	166
4.29.2.15 HAL_HALL_StopCounting()	167
4.30 hal_lksmcu_hall.h	167
4.31 include/hal_lksmcu_i2c.h 文件参考	167
4.31.1 枚举类型说明	169
4.31.1.1 HAL_I2C_x	169
4.31.2 函数说明	169
4.31.2.1 HAL_I2C_BusReset()	169
4.31.2.2 HAL_I2C_Disable()	170
4.31.2.3 HAL_I2C_Enable()	170
4.31.2.4 HAL_I2C_GetBaud()	171
4.31.2.5 HAL_I2C_GetClockDiv()	171
4.31.2.6 HAL_I2C_GetClockFreq()	171
4.31.2.7 HAL_I2C_Reset()	172
4.31.2.8 HAL_I2C_RxData()	172
4.31.2.9 HAL_I2C_SetBaud()	173
4.31.2.10 HAL_I2C_SetClockDiv()	173
4.31.2.11 HAL_I2C_SetGpioScl()	173
4.31.2.12 HAL_I2C_SetGpioSda()	174
4.31.2.13 HAL_I2C_TxData()	174
4.32 hal_lksmcu_i2c.h	175
4.33 include/hal_lksmcu_init.h 文件参考	175
4.33.1 宏定义说明	176
4.33.1.1 HAL_INIT_DEFAULE_MCLK	176
4.33.1.2 HAL_INIT_DEFAULE_MCLK_MS	176
4.33.1.3 HAL_INIT_DEFAULE_MCLK_US	176
4.33.2 枚举类型说明	176
4.33.2.1 HAL_INIT_POWER_x	176
4.33.3 函数说明	177
4.33.3.1 HAL_INIT_PowerReady()	177
4.33.3.2 HAL_INIT_SysInit()	177
4.34 hal_lksmcu_init.h	178
4.35 include/hal_lksmcu_irq.h 文件参考	178
4.35.1 宏定义说明	179
4.35.1.1 HAL_IRQHandler_ADC0	179
4.35.1.2 HAL_IRQHandler_CAN0	180
4.35.1.3 HAL_IRQHandler_CMP0	180
4.35.1.4 HAL_IRQHandler_DMA0	180
4.35.1.5 HAL_IRQHandler_DSP0	180
4.35.1.6 HAL_IRQHandler_GPIO0	180

4.35.1.7 HAL_IRQHandler_HALL0	180
4.35.1.8 HAL_IRQHandler_I2C0	180
4.35.1.9 HAL_IRQHandler_INIT0	180
4.35.1.10 HAL_IRQHandler_MCPWM0	181
4.35.1.11 HAL_IRQHandler_PWRDN0	181
4.35.1.12 HAL_IRQHandler_QEP0	181
4.35.1.13 HAL_IRQHandler_QEP1	181
4.35.1.14 HAL_IRQHandler_SIF0	181
4.35.1.15 HAL_IRQHandler_SPI0	181
4.35.1.16 HAL_IRQHandler_TIMER0	181
4.35.1.17 HAL_IRQHandler_TIMER1	182
4.35.1.18 HAL_IRQHandler_TIMER2	182
4.35.1.19 HAL_IRQHandler_TIMER3	182
4.35.1.20 HAL_IRQHandler_UART0	182
4.35.1.21 HAL_IRQHandler_UART1	182
4.35.1.22 HAL_IRQHandler_WAKE0	182
4.35.2 枚举类型说明	182
4.35.2.1 HAL_IRQ_MODULE_x	182
4.35.3 函数说明	183
4.35.3.1 HAL_IRQ_Disable()	183
4.35.3.2 HAL_IRQ_Enable()	184
4.35.3.3 HAL_IRQ_ModuleClear()	184
4.35.3.4 HAL_IRQ_ModuleDisable()	185
4.35.3.5 HAL_IRQ_ModuleEnable()	185
4.35.3.6 HAL_IRQ_Reset()	185
4.36 hal_lksmcu_irq.h	186
4.37 include/hal_lksmcu_irqfun.h 文件参考	188
4.37.1 函数说明	199
4.37.1.1 HAL_ADC_IrqHardErr()	199
4.37.1.2 HAL_ADC_IrqHardErrClear()	199
4.37.1.3 HAL_ADC_IrqHardErrDisable()	199
4.37.1.4 HAL_ADC_IrqHardErrEnable()	200
4.37.1.5 HAL_ADC_IrqHardErrFlg()	200
4.37.1.6 HAL_ADC_IrqIdleSamp()	200
4.37.1.7 HAL_ADC_IrqIdleSampClear()	201
4.37.1.8 HAL_ADC_IrqIdleSampDisable()	201
4.37.1.9 HAL_ADC_IrqIdleSampEnable()	201
4.37.1.10 HAL_ADC_IrqIdleSampFlg()	201
4.37.1.11 HAL_ADC_IrqSamp1()	201
4.37.1.12 HAL_ADC_IrqSamp1Clear()	202
4.37.1.13 HAL_ADC_IrqSamp1Disable()	202
4.37.1.14 HAL_ADC_IrqSamp1Enable()	202

4.37.1.15 HAL_ADC_IrqSamp1Flg()	202
4.37.1.16 HAL_ADC_IrqSamp2()	203
4.37.1.17 HAL_ADC_IrqSamp2Clear()	203
4.37.1.18 HAL_ADC_IrqSamp2Disable()	203
4.37.1.19 HAL_ADC_IrqSamp2Enable()	204
4.37.1.20 HAL_ADC_IrqSamp2Flg()	204
4.37.1.21 HAL_ADC_IrqSamp3()	204
4.37.1.22 HAL_ADC_IrqSamp3Clear()	205
4.37.1.23 HAL_ADC_IrqSamp3Disable()	205
4.37.1.24 HAL_ADC_IrqSamp3Enable()	205
4.37.1.25 HAL_ADC_IrqSamp3Flg()	206
4.37.1.26 HAL_ADC_IrqSamp4()	206
4.37.1.27 HAL_ADC_IrqSamp4Clear()	206
4.37.1.28 HAL_ADC_IrqSamp4Disable()	207
4.37.1.29 HAL_ADC_IrqSamp4Enable()	207
4.37.1.30 HAL_ADC_IrqSamp4Flg()	207
4.37.1.31 HAL_ADC_IrqSoftErr()	207
4.37.1.32 HAL_ADC_IrqSoftErrClear()	208
4.37.1.33 HAL_ADC_IrqSoftErrDisable()	208
4.37.1.34 HAL_ADC_IrqSoftErrEnable()	208
4.37.1.35 HAL_ADC_IrqSoftErrFlg()	209
4.37.1.36 HAL_ADC_IrqWatchdog()	209
4.37.1.37 HAL_ADC_IrqWatchdog1()	209
4.37.1.38 HAL_ADC_IrqWatchdog1Clear()	209
4.37.1.39 HAL_ADC_IrqWatchdog1Disable()	210
4.37.1.40 HAL_ADC_IrqWatchdog1Enable()	210
4.37.1.41 HAL_ADC_IrqWatchdog1Flg()	210
4.37.1.42 HAL_ADC_IrqWatchdogClear()	210
4.37.1.43 HAL_ADC_IrqWatchdogDisable()	210
4.37.1.44 HAL_ADC_IrqWatchdogEnable()	211
4.37.1.45 HAL_ADC_IrqWatchdogFlg()	211
4.37.1.46 HAL_CAN_IrqAll()	211
4.37.1.47 HAL_CAN_IrqAllClear()	212
4.37.1.48 HAL_CAN_IrqAllDisable()	212
4.37.1.49 HAL_CAN_IrqAllEnable()	212
4.37.1.50 HAL_CAN_IrqAllFlg()	212
4.37.1.51 HAL_CAN_IrqRxDone()	213
4.37.1.52 HAL_CAN_IrqRxDoneClear()	213
4.37.1.53 HAL_CAN_IrqRxDoneDisable()	213
4.37.1.54 HAL_CAN_IrqRxDoneEnable()	214
4.37.1.55 HAL_CAN_IrqRxDoneFlg()	214
4.37.1.56 HAL_CAN_IrqTxDone()	214

4.37.1.57 HAL_CAN_IrqTxDoneClear()	215
4.37.1.58 HAL_CAN_IrqTxDoneDisable()	215
4.37.1.59 HAL_CAN_IrqTxDoneEnable()	215
4.37.1.60 HAL_CAN_IrqTxDoneFlg()	216
4.37.1.61 HAL_CMP_IrqCompare()	216
4.37.1.62 HAL_CMP_IrqCompareClear()	216
4.37.1.63 HAL_CMP_IrqCompareDisable()	217
4.37.1.64 HAL_CMP_IrqCompareEnable()	217
4.37.1.65 HAL_CMP_IrqCompareFlg()	217
4.37.1.66 HAL_DMA_IrqDone()	218
4.37.1.67 HAL_DMA_IrqDoneClear()	218
4.37.1.68 HAL_DMA_IrqDoneDisable()	218
4.37.1.69 HAL_DMA_IrqDoneEnable()	219
4.37.1.70 HAL_DMA_IrqDoneFlg()	219
4.37.1.71 HAL_DMA_IrqDoneTrig()	219
4.37.1.72 HAL_DMA_IrqErr()	219
4.37.1.73 HAL_DMA_IrqErrClear()	220
4.37.1.74 HAL_DMA_IrqErrDisable()	220
4.37.1.75 HAL_DMA_IrqErrEnable()	220
4.37.1.76 HAL_DMA_IrqErrFlg()	221
4.37.1.77 HAL_DMA_IrqErrTrig()	221
4.37.1.78 HAL_GPIO_Irq()	221
4.37.1.79 HAL_GPIO_IrqClear()	222
4.37.1.80 HAL_GPIO_IrqDisable()	222
4.37.1.81 HAL_GPIO_IrqEnable()	222
4.37.1.82 HAL_GPIO_IrqFlg()	222
4.37.1.83 HAL_GPIO_IrqTrig()	223
4.37.1.84 HAL_HALL_IrqChange()	223
4.37.1.85 HAL_HALL_IrqChangeClear()	223
4.37.1.86 HAL_HALL_IrqChangeDisable()	224
4.37.1.87 HAL_HALL_IrqChangeEnable()	224
4.37.1.88 HAL_HALL_IrqChangeFlg()	224
4.37.1.89 HAL_HALL_IrqOver()	224
4.37.1.90 HAL_HALL_IrqOverClear()	225
4.37.1.91 HAL_HALL_IrqOverDisable()	225
4.37.1.92 HAL_HALL_IrqOverEnable()	225
4.37.1.93 HAL_HALL_IrqOverFlg()	226
4.37.1.94 HAL_I2C_IrqBusErr()	226
4.37.1.95 HAL_I2C_IrqBusErrClear()	226
4.37.1.96 HAL_I2C_IrqBusErrDisable()	227
4.37.1.97 HAL_I2C_IrqBusErrEnable()	227
4.37.1.98 HAL_I2C_IrqBusErrFlg()	227

4.37.1.99 HAL_I2C_IrqDone()	228
4.37.1.100 HAL_I2C_IrqDoneClear()	228
4.37.1.101 HAL_I2C_IrqDoneDisable()	228
4.37.1.102 HAL_I2C_IrqDoneEnable()	229
4.37.1.103 HAL_I2C_IrqDoneFlg()	229
4.37.1.104 HAL_I2C_IrqStop()	229
4.37.1.105 HAL_I2C_IrqStopClear()	230
4.37.1.106 HAL_I2C_IrqStopDisable()	230
4.37.1.107 HAL_I2C_IrqStopEnable()	230
4.37.1.108 HAL_I2C_IrqStopFlg()	231
4.37.1.109 HAL_MCPWM_IrqFail()	231
4.37.1.110 HAL_MCPWM_IrqFail0()	231
4.37.1.111 HAL_MCPWM_IrqFail0Clear()	232
4.37.1.112 HAL_MCPWM_IrqFail0Disable()	232
4.37.1.113 HAL_MCPWM_IrqFail0Enable()	232
4.37.1.114 HAL_MCPWM_IrqFail0Flg()	233
4.37.1.115 HAL_MCPWM_IrqFail1()	233
4.37.1.116 HAL_MCPWM_IrqFail1Clear()	233
4.37.1.117 HAL_MCPWM_IrqFail1Disable()	234
4.37.1.118 HAL_MCPWM_IrqFail1Enable()	234
4.37.1.119 HAL_MCPWM_IrqFail1Flg()	234
4.37.1.120 HAL_MCPWM_IrqFailClear()	235
4.37.1.121 HAL_MCPWM_IrqFailDisable()	235
4.37.1.122 HAL_MCPWM_IrqFailEnable()	235
4.37.1.123 HAL_MCPWM_IrqFailFlg()	235
4.37.1.124 HAL_MCPWM_IrqOverflow()	236
4.37.1.125 HAL_MCPWM_IrqOverflowClear()	236
4.37.1.126 HAL_MCPWM_IrqOverflowDisable()	236
4.37.1.127 HAL_MCPWM_IrqOverflowEnable()	237
4.37.1.128 HAL_MCPWM_IrqOverflowFlg()	237
4.37.1.129 HAL_MCPWM_IrqShort()	237
4.37.1.130 HAL_MCPWM_IrqShortClear()	238
4.37.1.131 HAL_MCPWM_IrqShortDisable()	238
4.37.1.132 HAL_MCPWM_IrqShortEnable()	238
4.37.1.133 HAL_MCPWM_IrqShortFlg()	239
4.37.1.134 HAL_MCPWM_IrqTH00()	239
4.37.1.135 HAL_MCPWM_IrqTH00Clear()	239
4.37.1.136 HAL_MCPWM_IrqTH00Disable()	240
4.37.1.137 HAL_MCPWM_IrqTH00Enable()	240
4.37.1.138 HAL_MCPWM_IrqTH00Flg()	240
4.37.1.139 HAL_MCPWM_IrqTH01()	240
4.37.1.140 HAL_MCPWM_IrqTH01Clear()	241

4.37.1.141 HAL_MCPWM_IrqTH01Disable()	241
4.37.1.142 HAL_MCPWM_IrqTH01Enable()	241
4.37.1.143 HAL_MCPWM_IrqTH01Flg()	242
4.37.1.144 HAL_MCPWM_IrqTH10()	242
4.37.1.145 HAL_MCPWM_IrqTH10Clear()	242
4.37.1.146 HAL_MCPWM_IrqTH10Disable()	243
4.37.1.147 HAL_MCPWM_IrqTH10Enable()	243
4.37.1.148 HAL_MCPWM_IrqTH10Flg()	243
4.37.1.149 HAL_MCPWM_IrqTH11()	244
4.37.1.150 HAL_MCPWM_IrqTH11Clear()	244
4.37.1.151 HAL_MCPWM_IrqTH11Disable()	244
4.37.1.152 HAL_MCPWM_IrqTH11Enable()	244
4.37.1.153 HAL_MCPWM_IrqTH11Flg()	245
4.37.1.154 HAL_MCPWM_IrqTH20()	245
4.37.1.155 HAL_MCPWM_IrqTH20Clear()	245
4.37.1.156 HAL_MCPWM_IrqTH20Disable()	246
4.37.1.157 HAL_MCPWM_IrqTH20Enable()	246
4.37.1.158 HAL_MCPWM_IrqTH20Flg()	246
4.37.1.159 HAL_MCPWM_IrqTH21()	247
4.37.1.160 HAL_MCPWM_IrqTH21Clear()	247
4.37.1.161 HAL_MCPWM_IrqTH21Disable()	247
4.37.1.162 HAL_MCPWM_IrqTH21Enable()	248
4.37.1.163 HAL_MCPWM_IrqTH21Flg()	248
4.37.1.164 HAL_MCPWM_IrqTH30()	248
4.37.1.165 HAL_MCPWM_IrqTH30Clear()	249
4.37.1.166 HAL_MCPWM_IrqTH30Disable()	249
4.37.1.167 HAL_MCPWM_IrqTH30Enable()	249
4.37.1.168 HAL_MCPWM_IrqTH30Flg()	249
4.37.1.169 HAL_MCPWM_IrqTH31()	250
4.37.1.170 HAL_MCPWM_IrqTH31Clear()	250
4.37.1.171 HAL_MCPWM_IrqTH31Disable()	250
4.37.1.172 HAL_MCPWM_IrqTH31Enable()	251
4.37.1.173 HAL_MCPWM_IrqTH31Flg()	251
4.37.1.174 HAL_MCPWM_IrqTH40()	251
4.37.1.175 HAL_MCPWM_IrqTH40Clear()	251
4.37.1.176 HAL_MCPWM_IrqTH40Disable()	252
4.37.1.177 HAL_MCPWM_IrqTH40Enable()	252
4.37.1.178 HAL_MCPWM_IrqTH40Flg()	252
4.37.1.179 HAL_MCPWM_IrqTH41()	252
4.37.1.180 HAL_MCPWM_IrqTH41Clear()	252
4.37.1.181 HAL_MCPWM_IrqTH41Disable()	252
4.37.1.182 HAL_MCPWM_IrqTH41Enable()	252

4.37.1.183 HAL_MCPWM_IrqTH41Flg()	253
4.37.1.184 HAL_MCPWM_IrqTH50()	253
4.37.1.185 HAL_MCPWM_IrqTH50Clear()	253
4.37.1.186 HAL_MCPWM_IrqTH50Disable()	253
4.37.1.187 HAL_MCPWM_IrqTH50Enable()	253
4.37.1.188 HAL_MCPWM_IrqTH50Flg()	253
4.37.1.189 HAL_MCPWM_IrqTH51()	253
4.37.1.190 HAL_MCPWM_IrqTH51Clear()	254
4.37.1.191 HAL_MCPWM_IrqTH51Disable()	254
4.37.1.192 HAL_MCPWM_IrqTH51Enable()	254
4.37.1.193 HAL_MCPWM_IrqTH51Flg()	254
4.37.1.194 HAL_MCPWM_IrqTmr0()	254
4.37.1.195 HAL_MCPWM_IrqTmr0Clear()	255
4.37.1.196 HAL_MCPWM_IrqTmr0Disable()	255
4.37.1.197 HAL_MCPWM_IrqTmr0Enable()	255
4.37.1.198 HAL_MCPWM_IrqTmr0Flg()	255
4.37.1.199 HAL_MCPWM_IrqTmr1()	256
4.37.1.200 HAL_MCPWM_IrqTmr1Clear()	256
4.37.1.201 HAL_MCPWM_IrqTmr1Disable()	256
4.37.1.202 HAL_MCPWM_IrqTmr1Enable()	257
4.37.1.203 HAL_MCPWM_IrqTmr1Flg()	257
4.37.1.204 HAL_MCPWM_IrqTmr2()	257
4.37.1.205 HAL_MCPWM_IrqTmr2Clear()	258
4.37.1.206 HAL_MCPWM_IrqTmr2Disable()	258
4.37.1.207 HAL_MCPWM_IrqTmr2Enable()	258
4.37.1.208 HAL_MCPWM_IrqTmr2Flg()	259
4.37.1.209 HAL_MCPWM_IrqTmr3()	259
4.37.1.210 HAL_MCPWM_IrqTmr3Clear()	259
4.37.1.211 HAL_MCPWM_IrqTmr3Disable()	260
4.37.1.212 HAL_MCPWM_IrqTmr3Enable()	260
4.37.1.213 HAL_MCPWM_IrqTmr3Flg()	260
4.37.1.214 HAL_MCPWM_IrqUpdate()	260
4.37.1.215 HAL_MCPWM_IrqUpdateClear()	261
4.37.1.216 HAL_MCPWM_IrqUpdateDisable()	261
4.37.1.217 HAL_MCPWM_IrqUpdateEnable()	261
4.37.1.218 HAL_MCPWM_IrqUpdateFlg()	262
4.37.1.219 HAL_MCPWM_IrqZeroCross()	262
4.37.1.220 HAL_MCPWM_IrqZeroCrossClear()	262
4.37.1.221 HAL_MCPWM_IrqZeroCrossDisable()	263
4.37.1.222 HAL_MCPWM_IrqZeroCrossEnable()	263
4.37.1.223 HAL_MCPWM_IrqZeroCrossFlg()	263
4.37.1.224 HAL_SPI_IrqDone()	264

4.37.1.225 HAL_SPI_IrqDoneClear()	264
4.37.1.226 HAL_SPI_IrqDoneDisable()	264
4.37.1.227 HAL_SPI_IrqDoneEnable()	265
4.37.1.228 HAL_SPI_IrqDoneFlg()	265
4.37.1.229 HAL_SPI_IrqErr()	265
4.37.1.230 HAL_SPI_IrqErrClear()	266
4.37.1.231 HAL_SPI_IrqErrDisable()	266
4.37.1.232 HAL_SPI_IrqErrEnable()	266
4.37.1.233 HAL_SPI_IrqErrFlg()	266
4.37.1.234 HAL_SPI_IrqOverflow()	267
4.37.1.235 HAL_SPI_IrqOverflowClear()	267
4.37.1.236 HAL_SPI_IrqOverflowDisable()	267
4.37.1.237 HAL_SPI_IrqOverflowEnable()	268
4.37.1.238 HAL_SPI_IrqOverflowFlg()	268
4.37.1.239 HAL_SW_Irq()	268
4.37.1.240 HAL_SW_IrqClear()	269
4.37.1.241 HAL_SW_IrqDisable()	269
4.37.1.242 HAL_SW_IrqEnable()	269
4.37.1.243 HAL_SW_IrqFlg()	269
4.37.1.244 HAL_SW_IrqTrig()	269
4.37.1.245 HAL_TIMER_IrqCompare0()	269
4.37.1.246 HAL_TIMER_IrqCompare0Clear()	270
4.37.1.247 HAL_TIMER_IrqCompare0Disable()	270
4.37.1.248 HAL_TIMER_IrqCompare0Enable()	270
4.37.1.249 HAL_TIMER_IrqCompare0Flg()	271
4.37.1.250 HAL_TIMER_IrqCompare1()	271
4.37.1.251 HAL_TIMER_IrqCompare1Clear()	271
4.37.1.252 HAL_TIMER_IrqCompare1Disable()	272
4.37.1.253 HAL_TIMER_IrqCompare1Enable()	272
4.37.1.254 HAL_TIMER_IrqCompare1Flg()	272
4.37.1.255 HAL_TIMER_IrqOverflow()	273
4.37.1.256 HAL_TIMER_IrqOverflowClear()	273
4.37.1.257 HAL_TIMER_IrqOverflowDisable()	273
4.37.1.258 HAL_TIMER_IrqOverflowEnable()	274
4.37.1.259 HAL_TIMER_IrqOverflowFlg()	274
4.37.1.260 HAL_UART_IrqAbd()	274
4.37.1.261 HAL_UART_IrqAbdClear()	274
4.37.1.262 HAL_UART_IrqAbdDisable()	275
4.37.1.263 HAL_UART_IrqAbdEnable()	275
4.37.1.264 HAL_UART_IrqAbdFlg()	275
4.37.1.265 HAL_UART_IrqCheckErr()	275
4.37.1.266 HAL_UART_IrqCheckErrClear()	275

4.37.1.267 HAL_UART_IrqCheckErrDisable()	276
4.37.1.268 HAL_UART_IrqCheckErrEnable()	276
4.37.1.269 HAL_UART_IrqCheckErrFlg()	276
4.37.1.270 HAL_UART_IrqIdle()	277
4.37.1.271 HAL_UART_IrqIdleClear()	277
4.37.1.272 HAL_UART_IrqIdleDisable()	277
4.37.1.273 HAL_UART_IrqIdleEnable()	277
4.37.1.274 HAL_UART_IrqIdleFlg()	277
4.37.1.275 HAL_UART_IrqLbd()	277
4.37.1.276 HAL_UART_IrqLbdClear()	278
4.37.1.277 HAL_UART_IrqLbdDisable()	278
4.37.1.278 HAL_UART_IrqLbdEnable()	278
4.37.1.279 HAL_UART_IrqLbdFlg()	278
4.37.1.280 HAL_UART_IrqRxDone()	278
4.37.1.281 HAL_UART_IrqRxDoneClear()	279
4.37.1.282 HAL_UART_IrqRxDoneDisable()	279
4.37.1.283 HAL_UART_IrqRxDoneEnable()	279
4.37.1.284 HAL_UART_IrqRxDoneFlg()	279
4.37.1.285 HAL_UART_IrqRxOv()	280
4.37.1.286 HAL_UART_IrqRxOvClear()	280
4.37.1.287 HAL_UART_IrqRxOvDisable()	280
4.37.1.288 HAL_UART_IrqRxOvEnable()	280
4.37.1.289 HAL_UART_IrqRxOvFlg()	280
4.37.1.290 HAL_UART_IrqStopErr()	280
4.37.1.291 HAL_UART_IrqStopErrClear()	281
4.37.1.292 HAL_UART_IrqStopErrDisable()	281
4.37.1.293 HAL_UART_IrqStopErrEnable()	281
4.37.1.294 HAL_UART_IrqStopErrFlg()	282
4.37.1.295 HAL_UART_IrqTxBufEmpty()	282
4.37.1.296 HAL_UART_IrqTxBufEmptyClear()	282
4.37.1.297 HAL_UART_IrqTxBufEmptyDisable()	283
4.37.1.298 HAL_UART_IrqTxBufEmptyEnable()	283
4.37.1.299 HAL_UART_IrqTxBufEmptyFlg()	283
4.37.1.300 HAL_UART_IrqTxDone()	284
4.37.1.301 HAL_UART_IrqTxDoneClear()	284
4.37.1.302 HAL_UART_IrqTxDoneDisable()	284
4.37.1.303 HAL_UART_IrqTxDoneEnable()	285
4.37.1.304 HAL_UART_IrqTxDoneFlg()	285
4.37.1.305 HAL_UART_IrqTxOv()	285
4.37.1.306 HAL_UART_IrqTxOvClear()	286
4.37.1.307 HAL_UART_IrqTxOvDisable()	286
4.37.1.308 HAL_UART_IrqTxOvEnable()	286

4.37.1.309 HAL_UART_IrqTxOvFlg()	286
4.38 hal_lksmcu_irqfun.h	286
4.39 include/hal_lksmcu_iwdg.h 文件参考	291
4.39.1 枚举类型说明	291
4.39.1.1 HAL_IWDG_x	291
4.39.2 函数说明	292
4.39.2.1 HAL_IWDG_Disable()	292
4.39.2.2 HAL_IWDG_Enable()	292
4.39.2.3 HAL_IWDG_Feed()	293
4.39.2.4 HAL_IWDG_Reset()	293
4.39.2.5 HAL_IWDG_SetResetTimemS()	294
4.40 hal_lksmcu_iwdg.h	294
4.41 include/hal_lksmcu_mcpwm.h 文件参考	294
4.41.1 枚举类型说明	297
4.41.1.1 HAL_MCPWM_DEFAULT_OUTPUT_STATE_x	297
4.41.1.2 HAL_MCPWM_FAIL_IN_x	297
4.41.1.3 HAL_MCPWM_FAIL_x	298
4.41.1.4 HAL_MCPWM_OUTPUT_STATE_x	298
4.41.1.5 HAL_MCPWM_POLARITY_x	298
4.41.1.6 HAL_MCPWM_PORT_x	299
4.41.1.7 HAL_MCPWM_SWAP_x	299
4.41.1.8 HAL_MCPWM_UPDATE_x	299
4.41.1.9 HAL_MCPWM_x	300
4.41.2 函数说明	300
4.41.2.1 HAL_MCPWM_Disable()	300
4.41.2.2 HAL_MCPWM_Enable()	300
4.41.2.3 HAL_MCPWM_GetClockDiv()	301
4.41.2.4 HAL_MCPWM_GetClockFreq()	301
4.41.2.5 HAL_MCPWM_GetFreq()	302
4.41.2.6 HAL_MCPWM_GetThCnt()	302
4.41.2.7 HAL_MCPWM_Reset()	303
4.41.2.8 HAL_MCPWM_SetAdcTrig()	303
4.41.2.9 HAL_MCPWM_SetAdcTrigQ15()	304
4.41.2.10 HAL_MCPWM_SetClockDiv()	304
4.41.2.11 HAL_MCPWM_SetDead()	305
4.41.2.12 HAL_MCPWM_SetDefaultState()	305
4.41.2.13 HAL_MCPWM_SetDrivePolarity()	306
4.41.2.14 HAL_MCPWM_SetDrivePolarityPort()	306
4.41.2.15 HAL_MCPWM_SetFailFilter()	307
4.41.2.16 HAL_MCPWM_SetFailSignal()	307
4.41.2.17 HAL_MCPWM_SetFreq()	308
4.41.2.18 HAL_MCPWM_SetGpioFail()	308

4.41.2.19 HAL_MCPWM_SetGpioOut()	309
4.41.2.20 HAL_MCPWM_SetOutputDisable()	310
4.41.2.21 HAL_MCPWM_SetOutputEnable()	310
4.41.2.22 HAL_MCPWM_SetOutState()	310
4.41.2.23 HAL_MCPWM_SetPwm()	311
4.41.2.24 HAL_MCPWM_SetPwmQ15()	311
4.41.2.25 HAL_MCPWM_SetRemap()	312
4.41.2.26 HAL_MCPWM_SetSwap()	312
4.41.2.27 HAL_MCPWM_SetSwapPort()	313
4.41.2.28 HAL_MCPWM_SetThCnt()	314
4.41.2.29 HAL_MCPWM_SetUpdateMode()	314
4.41.2.30 HAL_MCPWM_StartCounting()	314
4.41.2.31 HAL_MCPWM_StartCountingSync()	315
4.41.2.32 HAL_MCPWM_StopCounting()	315
4.42 hal_lksmcu_mcpwm.h	316
4.43 include/hal_lksmcu_opa.h 文件参考	317
4.43.1 宏定义说明	319
4.43.1.1 HAL_OPA_OPAX_2_ADCCHN	319
4.43.1.2 HAL_OPA_READ_GAIN	319
4.43.2 枚举类型说明	319
4.43.2.1 HAL_OPA_GAIN_x	319
4.43.2.2 HAL_OPA_x	320
4.43.3 函数说明	320
4.43.3.1 HAL_OPA_Disable()	320
4.43.3.2 HAL_OPA_Enable()	320
4.43.3.3 HAL_OPA_Out2GpioDisable()	320
4.43.3.4 HAL_OPA_Out2GpioEnable()	321
4.43.3.5 HAL_OPA_Reset()	321
4.43.3.6 HAL_OPA_SetGain()	321
4.43.3.7 HAL_OPA_SetGpioIn()	321
4.43.3.8 HAL_OPA_SetGpioOut()	322
4.44 hal_lksmcu_opa.h	322
4.45 include/hal_lksmcu_qep.h 文件参考	323
4.45.1 枚举类型说明	325
4.45.1.1 HAL_QEP_MODE_x	325
4.45.1.2 HAL_QEP_x	325
4.45.2 函数说明	325
4.45.2.1 HAL_QEP_Disable()	325
4.45.2.2 HAL_QEP_Enable()	326
4.45.2.3 HAL_QEP_GetClockDiv()	326
4.45.2.4 HAL_QEP_GetClockFreq()	327
4.45.2.5 HAL_QEP_ReadCnt()	327

4.45.2.6 HAL_QEP_Reset()	327
4.45.2.7 HAL_QEP_ResetCnt()	328
4.45.2.8 HAL_QEP_SetClockDiv()	328
4.45.2.9 HAL_QEP_SetGpio()	328
4.45.2.10 HAL_QEP_SetMode()	329
4.45.2.11 HAL_QEP_SetTh()	329
4.46 hal_lksmcu_qep.h	330
4.47 include/hal_lksmcu_reg.h 文件参考	330
4.47.1 详细描述	331
4.47.2 宏定义说明	331
4.47.2.1 HAL_REG_IF_WRITE	331
4.47.2.2 HAL_REG_READ	331
4.47.2.3 HAL_REG_READ_ADDR	331
4.47.2.4 HAL_REG_READ_BIT	331
4.47.2.5 HAL_REG_READ_BIT_MASK_POS	331
4.47.2.6 HAL_REG_RESET	332
4.47.2.7 HAL_REG_SET	332
4.47.2.8 HAL_REG_SYS_LOCK	332
4.47.2.9 HAL_REG_SYS_UNLOCK	332
4.47.2.10 HAL_REG_WRITE	332
4.47.2.11 HAL_REG_WRITE_BIT	332
4.47.2.12 HAL_REG_WRITE_BIT_MASK_POS	333
4.48 hal_lksmcu_reg.h	333
4.49 include/hal_lksmcu_reset.h 文件参考	333
4.49.1 函数说明	333
4.49.1.1 HAL_RESET_ResetCore()	333
4.49.1.2 HAL_RESET_ResetGlobal()	334
4.49.1.3 HAL_RESET_ResetRegister()	334
4.50 hal_lksmcu_reset.h	334
4.51 include/hal_lksmcu_spi.h 文件参考	335
4.51.1 枚举类型说明	336
4.51.1.1 HAL_SPI_MODE_x	336
4.51.1.2 HAL_SPI_x	337
4.51.2 函数说明	337
4.51.2.1 HAL_SPI_Disable()	337
4.51.2.2 HAL_SPI_Enable()	337
4.51.2.3 HAL_SPI_GetBaud()	338
4.51.2.4 HAL_SPI_GetClockDiv()	338
4.51.2.5 HAL_SPI_GetClockFreq()	339
4.51.2.6 HAL_SPI_Reset()	339
4.51.2.7 HAL_SPI_RxData()	339
4.51.2.8 HAL_SPI_RxDataWidth16()	340

4.51.2.9 HAL_SPI_RxTxData()	341
4.51.2.10 HAL_SPI_RxTxDataWidth16()	342
4.51.2.11 HAL_SPI_SetBaud()	342
4.51.2.12 HAL_SPI_SetClockDiv()	343
4.51.2.13 HAL_SPI_SetGpioClk()	343
4.51.2.14 HAL_SPI_SetGpioCs()	344
4.51.2.15 HAL_SPI_SetGpioDi()	345
4.51.2.16 HAL_SPI_SetGpioDo()	345
4.51.2.17 HAL_SPI_SetGpioMiso()	346
4.51.2.18 HAL_SPI_SetGpioMosi()	347
4.51.2.19 HAL_SPI_SetMode()	347
4.51.2.20 HAL_SPI_TxData()	348
4.51.2.21 HAL_SPI_TxDataWidth16()	348
4.52 hal_lksmcu_spi.h	349
4.53 include/hal_lksmcu_timer.h 文件参考	350
4.53.1 枚举类型说明	351
4.53.1.1 HAL_TIMER_MODE_x	351
4.53.1.2 HAL_TIMER_x	351
4.53.2 函数说明	352
4.53.2.1 HAL_TIMER_Disable()	352
4.53.2.2 HAL_TIMER_Enable()	352
4.53.2.3 HAL_TIMER_GetClockDiv()	353
4.53.2.4 HAL_TIMER_GetClockFreq()	353
4.53.2.5 HAL_TIMER_GetFreq()	353
4.53.2.6 HAL_TIMER_GetThCnt()	354
4.53.2.7 HAL_TIMER_Reset()	354
4.53.2.8 HAL_TIMER_SetClockDiv()	354
4.53.2.9 HAL_TIMER_SetFreq()	354
4.53.2.10 HAL_TIMER_SetGpioIn()	355
4.53.2.11 HAL_TIMER_SetGpioOut()	356
4.53.2.12 HAL_TIMER_SetPeriodCount()	356
4.53.2.13 HAL_TIMER_SetPeriodmS()	357
4.53.2.14 HAL_TIMER_SetPerioduS()	357
4.53.2.15 HAL_TIMER_SetPwm()	358
4.53.2.16 HAL_TIMER_SetPwmQ15()	358
4.53.2.17 HAL_TIMER_SetWorkMode()	359
4.53.2.18 HAL_TIMER_StartCounting()	359
4.53.2.19 HAL_TIMER_StopCounting()	359
4.54 hal_lksmcu_timer.h	360
4.55 include/hal_lksmcu_tmp.h 文件参考	360
4.55.1 枚举类型说明	362
4.55.1.1 HAL_TMP_x	362

4.55.2 函数说明	362
4.55.2.1 HAL_TMP_Adc2Temperature()	362
4.55.2.2 HAL_TMP_Adc2Temperature01()	363
4.55.2.3 HAL_TMP_Disable()	363
4.55.2.4 HAL_TMP_Enable()	364
4.55.2.5 HAL_TMP_GetTemperature()	364
4.55.2.6 HAL_TMP_GetTemperature01()	365
4.55.2.7 HAL_TMP_Reset()	366
4.56 hal_lksmcu_tmp.h	366
4.57 include/hal_lksmcu_trim.h 文件参考	367
4.57.1 函数说明	367
4.57.1.1 HAL_TRIM_Read()	367
4.57.1.2 HAL_TRIM_ReadId()	368
4.57.1.3 HAL_TRIM_ReadIdCrc32()	369
4.58 hal_lksmcu_trim.h	369
4.59 include/hal_lksmcu_uart.h 文件参考	370
4.59.1 枚举类型说明	371
4.59.1.1 HAL_UART_MODE_x	371
4.59.1.2 HAL_UART_POL_x	371
4.59.1.3 HAL_UART_x	372
4.59.2 函数说明	372
4.59.2.1 HAL_UART_Disable()	372
4.59.2.2 HAL_UART_Enable()	372
4.59.2.3 HAL_UART_GetBaud()	372
4.59.2.4 HAL_UART_GetClockDiv()	373
4.59.2.5 HAL_UART_GetClockFreq()	373
4.59.2.6 HAL_UART_ReadBuffer()	373
4.59.2.7 HAL_UART_Reset()	374
4.59.2.8 HAL_UART_SetBaud()	374
4.59.2.9 HAL_UART_SetClockDiv()	374
4.59.2.10 HAL_UART_SetGpioRx()	374
4.59.2.11 HAL_UART_SetGpioTx()	375
4.59.2.12 HAL_UART_SetMode()	375
4.59.2.13 HAL_UART_SetPolarity()	375
4.59.2.14 HAL_UART_TxData()	376
4.59.2.15 HAL_UART_WriteBuffer()	376
4.60 hal_lksmcu_uart.h	376
4.61 include/hal_lksmcu_wake.h 文件参考	377
4.61.1 枚举类型说明	378
4.61.1.1 HAL_WAKE_UP_IO_x	378
4.61.2 函数说明	378
4.61.2.1 HAL_WAKE_DeepSleepToTimeWakeUp()	378

4.61.2.2 HAL_WAKE_DeepSleepIoWake()	379
4.61.2.3 HAL_WAKE_DeepSleepTimeWakeUp()	379
4.62 hal_lksmcu_wake.h	380
4.63 readme.md 文件参考	381
4.64 source/hal_lks32mc08x_adc.c 文件参考	381
4.64.1 详细描述	382
4.64.2 函数说明	383
4.64.2.1 HAL_ADC_Disable()	383
4.64.2.2 HAL_ADC_Enable()	383
4.64.2.3 HAL_ADC_EndSampling()	383
4.64.2.4 HAL_ADC_GetRange()	384
4.64.2.5 HAL_ADC_GetSamplingValues()	384
4.64.2.6 HAL_ADC_GetValuesUnit()	385
4.64.2.7 HAL_ADC_ManualSampling()	386
4.64.2.8 HAL_ADC_Reset()	387
4.64.2.9 HAL_ADC_SetGpioIn()	387
4.64.2.10 HAL_ADC_SetGpioTriggerOut()	387
4.64.2.11 HAL_ADC_SetReference()	389
4.64.2.12 HAL_ADC_SetSamplingChannel()	390
4.64.2.13 HAL_ADC_SetSamplingQuantity()	390
4.64.2.14 HAL_ADC_SetTriggerSignal()	391
4.64.2.15 HAL_ADC_SoftTrigger()	392
4.64.2.16 HAL_ADC_StateReset()	392
4.64.2.17 HAL_ADC_UpdateTrig()	393
4.64.3 变量说明	393
4.64.3.1 HAL_ADC_TRIG_STATE	393
4.65 hal_lks32mc08x_adc.c	393
4.66 source/hal_lks32mc08x_can.c 文件参考	397
4.66.1 函数说明	399
4.66.1.1 CAN_Write_TBUF()	399
4.66.1.2 GetBaudParam()	400
4.66.1.3 HAL_CAN_ArbitrationLostCap()	400
4.66.1.4 HAL_CAN_Busoff()	400
4.66.1.5 HAL_CAN_Disable()	401
4.66.1.6 HAL_CAN_Enable()	401
4.66.1.7 HAL_CAN_ErrorPassive()	401
4.66.1.8 HAL_CAN_GetBaud()	401
4.66.1.9 HAL_CAN_GetClockDiv()	401
4.66.1.10 HAL_CAN_GetClockFreq()	402
4.66.1.11 HAL_CAN_IDx_Filter()	402
4.66.1.12 HAL_CAN_IDx_FilterDisable()	403
4.66.1.13 HAL_CAN_IDx_FilterEnable()	403

4.66.1.14 HAL_CAN_KindOfError()	403
4.66.1.15 HAL_CAN_LBME_Disable()	403
4.66.1.16 HAL_CAN_LBME_Enable()	403
4.66.1.17 HAL_CAN_LBMI_Disable()	403
4.66.1.18 HAL_CAN_LBMI_Enable()	404
4.66.1.19 HAL_CAN_LOM_Disable()	404
4.66.1.20 HAL_CAN_LOM_Enable()	404
4.66.1.21 HAL_CAN_Reset()	404
4.66.1.22 HAL_CAN_ROverflow()	404
4.66.1.23 HAL_CAN_RxData()	405
4.66.1.24 HAL_CAN_RxDataBuf()	405
4.66.1.25 HAL_CAN_RxDataFdf()	405
4.66.1.26 HAL_CAN_RxDataID()	405
4.66.1.27 HAL_CAN_RxDataIde()	405
4.66.1.28 HAL_CAN_RxDataLen()	405
4.66.1.29 HAL_CAN_RxDataRtr()	405
4.66.1.30 HAL_CAN_RxErrorCnt()	406
4.66.1.31 HAL_CAN_SetBaud()	406
4.66.1.32 HAL_CAN_SetClockDiv()	406
4.66.1.33 HAL_CAN_SetFastBaud()	406
4.66.1.34 HAL_CAN_SetGpioRx()	407
4.66.1.35 HAL_CAN_SetGpioTx()	407
4.66.1.36 HAL_CAN_SetSTBMode()	407
4.66.1.37 HAL_CAN_SetTDC()	408
4.66.1.38 HAL_CAN_SS_Disable()	408
4.66.1.39 HAL_CAN_SS_Enable()	408
4.66.1.40 HAL_CAN_STBFull()	408
4.66.1.41 HAL_CAN_TSSStat()	408
4.66.1.42 HAL_CAN_TxCancel()	408
4.66.1.43 HAL_CAN_TxData()	409
4.66.1.44 HAL_CAN_TxData_STBTrig()	409
4.66.1.45 HAL_CAN_TxData_STBWrite()	409
4.66.1.46 HAL_CAN_TxErrorCnt()	409
4.66.1.47 HAL_CAN_Txing()	410
4.66.2 变量说明	410
4.66.2.1 baud.cnt	410
4.66.2.2 baud.param	410
4.66.2.3 hal_rx_buf	410
4.66.2.4 hal_rx_id	410
4.66.2.5 hal_rx_ide	410
4.66.2.6 hal_rx_len	411
4.66.2.7 hal_rx_rtr	411

4.67 hal_lks32mc08x_can.c	411
4.68 source/hal_lks32mc08x_clk.c 文件参考	418
4.68.1 函数说明	419
4.68.1.1 HAL_CLK_GetMcuFreq()	419
4.68.1.2 HAL_CLK_SetMcuFreq()	420
4.68.2 变量说明	421
4.68.2.1 HAL_CLK_mclk	421
4.69 hal_lks32mc08x_clk.c	421
4.70 source/hal_lks32mc08x_cmp.c 文件参考	422
4.70.1 函数说明	423
4.70.1.1 HAL_CMP_Disable()	423
4.70.1.2 HAL_CMP_Enable()	424
4.70.1.3 HAL_CMP_GetData()	424
4.70.1.4 HAL_CMP_Reset()	425
4.70.1.5 HAL_CMP_SetChannel()	425
4.70.1.6 HAL_CMP_SetDigitalFilter()	425
4.70.1.7 HAL_CMP_SetGpioIn()	426
4.70.1.8 HAL_CMP_SetGpioOut()	426
4.70.1.9 HAL_CMP_SetHysteresis()	427
4.70.1.10 HAL_CMP_SetMode()	427
4.71 hal_lks32mc08x_cmp.c	428
4.72 source/hal_lks32mc08x_crc.c 文件参考	430
4.72.1 枚举类型说明	431
4.72.1.1 CRC_Width	431
4.72.2 函数说明	432
4.72.2.1 HAL_CRC_Calc()	432
4.72.2.2 HAL_CRC_Disable()	433
4.72.2.3 HAL_CRC_Enable()	433
4.72.2.4 HAL_CRC_Reflect()	433
4.72.2.5 HAL_CRC_Reset()	434
4.72.3 变量说明	434
4.72.3.1 crcModels	434
4.73 hal_lks32mc08x_crc.c	435
4.74 source/hal_lks32mc08x_dac.c 文件参考	436
4.74.1 函数说明	438
4.74.1.1 HAL_DAC_Disable()	438
4.74.1.2 HAL_DAC_Enable()	438
4.74.1.3 HAL_DAC_GetOutValMax()	438
4.74.1.4 HAL_DAC_GetRange()	439
4.74.1.5 HAL_DAC_Out2GpioDisable()	439
4.74.1.6 HAL_DAC_Out2GpioEnable()	439
4.74.1.7 HAL_DAC_Reset()	440

4.74.1.8 HAL_DAC_SetGpioOut()	440
4.74.1.9 HAL_DAC_SetOutVal()	441
4.74.1.10 HAL_DAC_SetOutVoltageV()	441
4.74.1.11 HAL_DAC_SetRange()	442
4.75 hal_lks32mc08x_dac.c	443
4.76 source/hal_lks32mc08x_dma.c 文件参考	444
4.76.1 函数说明	445
4.76.1.1 HAL_DMA_Disable()	445
4.76.1.2 HAL_DMA_Enable()	446
4.76.1.3 HAL_DMA_I2cRx()	446
4.76.1.4 HAL_DMA_I2cTx()	446
4.76.1.5 HAL_DMA_MemCopy()	446
4.76.1.6 HAL_DMA_Reset()	447
4.76.1.7 HAL_DMA_SpiRx()	447
4.76.1.8 HAL_DMA_SpiRxTx()	447
4.76.1.9 HAL_DMA_SpiTx()	448
4.76.1.10 HAL_DMA_UartRx()	448
4.76.1.11 HAL_DMA_UartTx()	448
4.77 hal_lks32mc08x_dma.c	449
4.78 source/hal_lks32mc08x_dsp.c 文件参考	450
4.78.1 函数说明	451
4.78.1.1 HAL_DSP_Cos()	451
4.78.1.2 HAL_DSP_Disable()	452
4.78.1.3 HAL_DSP_Div()	452
4.78.1.4 HAL_DSP_Enable()	452
4.78.1.5 HAL_DSP_Reset()	453
4.78.1.6 HAL_DSP_Sin()	453
4.78.1.7 HAL_DSP_SinCos()	453
4.78.1.8 HAL_DSP_Sqrt()	454
4.79 hal_lks32mc08x_dsp.c	454
4.80 source/hal_lks32mc08x_flash.c 文件参考	455
4.80.1 函数说明	457
4.80.1.1 HAL_FLASH_Disable()	457
4.80.1.2 HAL_FLASH_DisableCache()	457
4.80.1.3 HAL_FLASH_Enable()	457
4.80.1.4 HAL_FLASH_EnableCache()	458
4.80.1.5 HAL_FLASH_Erase()	458
4.80.1.6 HAL_FLASH_Program()	459
4.80.1.7 HAL_FLASH_Read()	459
4.80.1.8 HAL_FLASH_ReadSectorSize()	460
4.80.1.9 HAL_FLASH_ReadUint16()	461
4.80.1.10 HAL_FLASH_ReadUint32()	461

4.80.1.11 HAL_FLASH_ReadUint8()	462
4.80.1.12 HAL_FLASH_Reset()	462
4.80.1.13 HAL_FLASH_SecurityParameterCrc()	462
4.80.1.14 HAL_FLASH_SecurityParameterRead()	463
4.80.1.15 HAL_FLASH_SecurityParameterSave()	464
4.80.1.16 HAL_FLASH_Update()	464
4.80.2 变量说明	465
4.80.2.1 HAL_FLASH_data	465
4.81 hal_lks32mc08x_flash.c	465
4.82 source/hal_lks32mc08x_gpio.c 文件参考	469
4.82.1 函数说明	470
4.82.1.1 HAL_GPIO_Disable()	470
4.82.1.2 HAL_GPIO_Enable()	470
4.82.1.3 HAL_GPIO_GetInData()	470
4.82.1.4 HAL_GPIO_GetOutData()	471
4.82.1.5 HAL_GPIO_Invert()	471
4.82.1.6 HAL_GPIO_Reset()	472
4.82.1.7 HAL_GPIO_RstReuseGpioDisable()	472
4.82.1.8 HAL_GPIO_RstReuseGpioEnable()	472
4.82.1.9 HAL_GPIO_SetAf()	472
4.82.1.10 HAL_GPIO_SetHigh()	474
4.82.1.11 HAL_GPIO_SetLow()	474
4.82.1.12 HAL_GPIO_SetMode()	474
4.82.1.13 HAL_GPIO_SwdReuseGpioDisable()	475
4.82.1.14 HAL_GPIO_SwdReuseGpioEnable()	476
4.82.1.15 HAL_GPIO_Write()	476
4.83 hal_lks32mc08x_gpio.c	476
4.84 source/hal_lks32mc08x_hall.c 文件参考	480
4.84.1 函数说明	481
4.84.1.1 HAL_HALL_Disable()	481
4.84.1.2 HAL_HALL_Enable()	481
4.84.1.3 HAL_HALL_GetCNT()	482
4.84.1.4 HAL_HALL_ReadHallData()	482
4.84.1.5 HAL_HALL_ReadHallWidth()	482
4.84.1.6 HAL_HALL_ReadTh()	483
4.84.1.7 HAL_HALL_Reset()	483
4.84.1.8 HAL_HALL_SetFilter()	483
4.84.1.9 HAL_HALL_SetFilter75()	484
4.84.1.10 HAL_HALL_SetFilteruS()	484
4.84.1.11 HAL_HALL_SetFreq()	485
4.84.1.12 HAL_HALL_SetGpio()	485
4.84.1.13 HAL_HALL_SetTh()	486

4.84.1.14 HAL_HALL_StartCounting()	486
4.84.1.15 HAL_HALL_StopCounting()	487
4.84.2 变量说明	487
4.84.2.1 hal_hall_filter	487
4.84.2.2 hal_hall_th	487
4.85 hal_lks32mc08x_hall.c	487
4.86 source/hal_lks32mc08x_i2c.c 文件参考	489
4.86.1 函数说明	490
4.86.1.1 HAL_I2C_BusReset()	490
4.86.1.2 HAL_I2C_Disable()	491
4.86.1.3 HAL_I2C_Enable()	491
4.86.1.4 HAL_I2C_GetBaud()	492
4.86.1.5 HAL_I2C_GetClockDiv()	492
4.86.1.6 HAL_I2C_GetClockFreq()	492
4.86.1.7 HAL_I2C_Reset()	493
4.86.1.8 HAL_I2C_RxData()	493
4.86.1.9 HAL_I2C_SetBaud()	494
4.86.1.10 HAL_I2C_SetClockDiv()	494
4.86.1.11 HAL_I2C_SetGpioScl()	494
4.86.1.12 HAL_I2C_SetGpioSda()	495
4.86.1.13 HAL_I2C_TxData()	495
4.86.2 变量说明	496
4.86.2.1 hal_baud	496
4.86.2.2 hal_i2cio	496
4.87 hal_lks32mc08x_i2c.c	496
4.88 source/hal_lks32mc08x_init.c 文件参考	499
4.88.1 函数说明	500
4.88.1.1 HAL_INIT_PowerReady()	500
4.88.1.2 HAL_INIT_SysInit()	500
4.89 hal_lks32mc08x_init.c	500
4.90 source/hal_lks32mc08x_irq.c 文件参考	501
4.90.1 函数说明	502
4.90.1.1 HAL_IRQ_Disable()	502
4.90.1.2 HAL_IRQ_Enable()	502
4.90.1.3 HAL_IRQ_ModuleClear()	503
4.90.1.4 HAL_IRQ_ModuleDisable()	503
4.90.1.5 HAL_IRQ_ModuleEnable()	503
4.90.1.6 HAL_IRQ_Reset()	504
4.91 hal_lks32mc08x_irq.c	505
4.92 source/hal_lks32mc08x_irqfun.c 文件参考	505
4.92.1 函数说明	517
4.92.1.1 HAL_ADC_IrqHardErr()	517

4.92.1.2 HAL_ADC_IrqHardErrClear()	517
4.92.1.3 HAL_ADC_IrqHardErrDisable()	517
4.92.1.4 HAL_ADC_IrqHardErrEnable()	518
4.92.1.5 HAL_ADC_IrqHardErrFlg()	518
4.92.1.6 HAL_ADC_IrqIdleSamp()	518
4.92.1.7 HAL_ADC_IrqIdleSampClear()	518
4.92.1.8 HAL_ADC_IrqIdleSampDisable()	519
4.92.1.9 HAL_ADC_IrqIdleSampEnable()	519
4.92.1.10 HAL_ADC_IrqIdleSampFlg()	519
4.92.1.11 HAL_ADC_IrqSamp1()	519
4.92.1.12 HAL_ADC_IrqSamp1Clear()	519
4.92.1.13 HAL_ADC_IrqSamp1Disable()	520
4.92.1.14 HAL_ADC_IrqSamp1Enable()	520
4.92.1.15 HAL_ADC_IrqSamp1Flg()	520
4.92.1.16 HAL_ADC_IrqSamp2()	521
4.92.1.17 HAL_ADC_IrqSamp2Clear()	521
4.92.1.18 HAL_ADC_IrqSamp2Disable()	521
4.92.1.19 HAL_ADC_IrqSamp2Enable()	522
4.92.1.20 HAL_ADC_IrqSamp2Flg()	522
4.92.1.21 HAL_ADC_IrqSamp3()	522
4.92.1.22 HAL_ADC_IrqSamp3Clear()	523
4.92.1.23 HAL_ADC_IrqSamp3Disable()	523
4.92.1.24 HAL_ADC_IrqSamp3Enable()	523
4.92.1.25 HAL_ADC_IrqSamp3Flg()	523
4.92.1.26 HAL_ADC_IrqSamp4()	524
4.92.1.27 HAL_ADC_IrqSamp4Clear()	524
4.92.1.28 HAL_ADC_IrqSamp4Disable()	524
4.92.1.29 HAL_ADC_IrqSamp4Enable()	525
4.92.1.30 HAL_ADC_IrqSamp4Flg()	525
4.92.1.31 HAL_ADC_IrqSoftErr()	525
4.92.1.32 HAL_ADC_IrqSoftErrClear()	526
4.92.1.33 HAL_ADC_IrqSoftErrDisable()	526
4.92.1.34 HAL_ADC_IrqSoftErrEnable()	526
4.92.1.35 HAL_ADC_IrqSoftErrFlg()	527
4.92.1.36 HAL_ADC_IrqWatchdog()	527
4.92.1.37 HAL_ADC_IrqWatchdog1()	527
4.92.1.38 HAL_ADC_IrqWatchdog1Clear()	527
4.92.1.39 HAL_ADC_IrqWatchdog1Disable()	528
4.92.1.40 HAL_ADC_IrqWatchdog1Enable()	528
4.92.1.41 HAL_ADC_IrqWatchdog1Flg()	528
4.92.1.42 HAL_ADC_IrqWatchdogClear()	528
4.92.1.43 HAL_ADC_IrqWatchdogDisable()	528

4.92.1.44 HAL_ADC_IrqWatchdogEnable()	529
4.92.1.45 HAL_ADC_IrqWatchdogFlg()	529
4.92.1.46 HAL_CAN_IrqAll()	529
4.92.1.47 HAL_CAN_IrqAllClear()	530
4.92.1.48 HAL_CAN_IrqAllDisable()	530
4.92.1.49 HAL_CAN_IrqAllEnable()	530
4.92.1.50 HAL_CAN_IrqAllFlg()	530
4.92.1.51 HAL_CAN_IrqRxDone()	531
4.92.1.52 HAL_CAN_IrqRxDoneClear()	531
4.92.1.53 HAL_CAN_IrqRxDoneDisable()	531
4.92.1.54 HAL_CAN_IrqRxDoneEnable()	532
4.92.1.55 HAL_CAN_IrqRxDoneFlg()	532
4.92.1.56 HAL_CAN_IrqTxDone()	532
4.92.1.57 HAL_CAN_IrqTxDoneClear()	533
4.92.1.58 HAL_CAN_IrqTxDoneDisable()	533
4.92.1.59 HAL_CAN_IrqTxDoneEnable()	533
4.92.1.60 HAL_CAN_IrqTxDoneFlg()	534
4.92.1.61 HAL_CMP_IrqCompare()	534
4.92.1.62 HAL_CMP_IrqCompareClear()	534
4.92.1.63 HAL_CMP_IrqCompareDisable()	535
4.92.1.64 HAL_CMP_IrqCompareEnable()	535
4.92.1.65 HAL_CMP_IrqCompareFlg()	535
4.92.1.66 HAL_DMA_IrqDone()	536
4.92.1.67 HAL_DMA_IrqDoneClear()	536
4.92.1.68 HAL_DMA_IrqDoneDisable()	536
4.92.1.69 HAL_DMA_IrqDoneEnable()	537
4.92.1.70 HAL_DMA_IrqDoneFlg()	537
4.92.1.71 HAL_DMA_IrqErr()	537
4.92.1.72 HAL_DMA_IrqErrClear()	538
4.92.1.73 HAL_DMA_IrqErrDisable()	538
4.92.1.74 HAL_DMA_IrqErrEnable()	538
4.92.1.75 HAL_DMA_IrqErrFlg()	539
4.92.1.76 HAL_GPIO_Irq()	539
4.92.1.77 HAL_GPIO_IrqClear()	539
4.92.1.78 HAL_GPIO_IrqDisable()	540
4.92.1.79 HAL_GPIO_IrqEnable()	540
4.92.1.80 HAL_GPIO_IrqFlg()	540
4.92.1.81 HAL_HALL_IrqChange()	541
4.92.1.82 HAL_HALL_IrqChangeClear()	541
4.92.1.83 HAL_HALL_IrqChangeDisable()	541
4.92.1.84 HAL_HALL_IrqChangeEnable()	542
4.92.1.85 HAL_HALL_IrqChangeFlg()	542

4.92.1.86 HAL_HALL_IrqOver()	542
4.92.1.87 HAL_HALL_IrqOverClear()	543
4.92.1.88 HAL_HALL_IrqOverDisable()	543
4.92.1.89 HAL_HALL_IrqOverEnable()	543
4.92.1.90 HAL_HALL_IrqOverFlg()	543
4.92.1.91 HAL_I2C_IrqBusErr()	544
4.92.1.92 HAL_I2C_IrqBusErrClear()	544
4.92.1.93 HAL_I2C_IrqBusErrDisable()	544
4.92.1.94 HAL_I2C_IrqBusErrEnable()	545
4.92.1.95 HAL_I2C_IrqBusErrFlg()	545
4.92.1.96 HAL_I2C_IrqDone()	545
4.92.1.97 HAL_I2C_IrqDoneClear()	546
4.92.1.98 HAL_I2C_IrqDoneDisable()	546
4.92.1.99 HAL_I2C_IrqDoneEnable()	546
4.92.1.100 HAL_I2C_IrqDoneFlg()	547
4.92.1.101 HAL_I2C_IrqStop()	547
4.92.1.102 HAL_I2C_IrqStopClear()	547
4.92.1.103 HAL_I2C_IrqStopDisable()	548
4.92.1.104 HAL_I2C_IrqStopEnable()	548
4.92.1.105 HAL_I2C_IrqStopFlg()	548
4.92.1.106 HAL_MCPWM_IrqFail()	549
4.92.1.107 HAL_MCPWM_IrqFail0()	549
4.92.1.108 HAL_MCPWM_IrqFail0Clear()	549
4.92.1.109 HAL_MCPWM_IrqFail0Disable()	550
4.92.1.110 HAL_MCPWM_IrqFail0Enable()	550
4.92.1.111 HAL_MCPWM_IrqFail0Flg()	550
4.92.1.112 HAL_MCPWM_IrqFail1()	551
4.92.1.113 HAL_MCPWM_IrqFail1Clear()	551
4.92.1.114 HAL_MCPWM_IrqFail1Disable()	551
4.92.1.115 HAL_MCPWM_IrqFail1Enable()	552
4.92.1.116 HAL_MCPWM_IrqFail1Flg()	552
4.92.1.117 HAL_MCPWM_IrqFailClear()	552
4.92.1.118 HAL_MCPWM_IrqFailDisable()	553
4.92.1.119 HAL_MCPWM_IrqFailEnable()	553
4.92.1.120 HAL_MCPWM_IrqFailFlg()	553
4.92.1.121 HAL_MCPWM_IrqOverflow()	553
4.92.1.122 HAL_MCPWM_IrqOverflowClear()	554
4.92.1.123 HAL_MCPWM_IrqOverflowDisable()	554
4.92.1.124 HAL_MCPWM_IrqOverflowEnable()	554
4.92.1.125 HAL_MCPWM_IrqOverflowFlg()	555
4.92.1.126 HAL_MCPWM_IrqShort()	555
4.92.1.127 HAL_MCPWM_IrqShortClear()	555

4.92.1.128 HAL_MCPWM_IrqShortDisable()	556
4.92.1.129 HAL_MCPWM_IrqShortEnable()	556
4.92.1.130 HAL_MCPWM_IrqShortFlg()	556
4.92.1.131 HAL_MCPWM_IrqTH00()	557
4.92.1.132 HAL_MCPWM_IrqTH00Clear()	557
4.92.1.133 HAL_MCPWM_IrqTH00Disable()	557
4.92.1.134 HAL_MCPWM_IrqTH00Enable()	557
4.92.1.135 HAL_MCPWM_IrqTH00Flg()	558
4.92.1.136 HAL_MCPWM_IrqTH01()	558
4.92.1.137 HAL_MCPWM_IrqTH01Clear()	558
4.92.1.138 HAL_MCPWM_IrqTH01Disable()	559
4.92.1.139 HAL_MCPWM_IrqTH01Enable()	559
4.92.1.140 HAL_MCPWM_IrqTH01Flg()	559
4.92.1.141 HAL_MCPWM_IrqTH10()	560
4.92.1.142 HAL_MCPWM_IrqTH10Clear()	560
4.92.1.143 HAL_MCPWM_IrqTH10Disable()	560
4.92.1.144 HAL_MCPWM_IrqTH10Enable()	561
4.92.1.145 HAL_MCPWM_IrqTH10Flg()	561
4.92.1.146 HAL_MCPWM_IrqTH11()	561
4.92.1.147 HAL_MCPWM_IrqTH11Clear()	562
4.92.1.148 HAL_MCPWM_IrqTH11Disable()	562
4.92.1.149 HAL_MCPWM_IrqTH11Enable()	562
4.92.1.150 HAL_MCPWM_IrqTH11Flg()	562
4.92.1.151 HAL_MCPWM_IrqTH20()	563
4.92.1.152 HAL_MCPWM_IrqTH20Clear()	563
4.92.1.153 HAL_MCPWM_IrqTH20Disable()	563
4.92.1.154 HAL_MCPWM_IrqTH20Enable()	564
4.92.1.155 HAL_MCPWM_IrqTH20Flg()	564
4.92.1.156 HAL_MCPWM_IrqTH21()	564
4.92.1.157 HAL_MCPWM_IrqTH21Clear()	565
4.92.1.158 HAL_MCPWM_IrqTH21Disable()	565
4.92.1.159 HAL_MCPWM_IrqTH21Enable()	565
4.92.1.160 HAL_MCPWM_IrqTH21Flg()	566
4.92.1.161 HAL_MCPWM_IrqTH30()	566
4.92.1.162 HAL_MCPWM_IrqTH30Clear()	566
4.92.1.163 HAL_MCPWM_IrqTH30Disable()	567
4.92.1.164 HAL_MCPWM_IrqTH30Enable()	567
4.92.1.165 HAL_MCPWM_IrqTH30Flg()	567
4.92.1.166 HAL_MCPWM_IrqTH31()	567
4.92.1.167 HAL_MCPWM_IrqTH31Clear()	568
4.92.1.168 HAL_MCPWM_IrqTH31Disable()	568
4.92.1.169 HAL_MCPWM_IrqTH31Enable()	568

4.92.1.170 HAL_MCPWM_IrqTH31Flg()	569
4.92.1.171 HAL_MCPWM_IrqTH40()	569
4.92.1.172 HAL_MCPWM_IrqTH40Clear()	569
4.92.1.173 HAL_MCPWM_IrqTH40Disable()	569
4.92.1.174 HAL_MCPWM_IrqTH40Enable()	569
4.92.1.175 HAL_MCPWM_IrqTH40Flg()	570
4.92.1.176 HAL_MCPWM_IrqTH41()	570
4.92.1.177 HAL_MCPWM_IrqTH41Clear()	570
4.92.1.178 HAL_MCPWM_IrqTH41Disable()	570
4.92.1.179 HAL_MCPWM_IrqTH41Enable()	570
4.92.1.180 HAL_MCPWM_IrqTH41Flg()	570
4.92.1.181 HAL_MCPWM_IrqTH50()	570
4.92.1.182 HAL_MCPWM_IrqTH50Clear()	571
4.92.1.183 HAL_MCPWM_IrqTH50Disable()	571
4.92.1.184 HAL_MCPWM_IrqTH50Enable()	571
4.92.1.185 HAL_MCPWM_IrqTH50Flg()	571
4.92.1.186 HAL_MCPWM_IrqTH51()	571
4.92.1.187 HAL_MCPWM_IrqTH51Clear()	571
4.92.1.188 HAL_MCPWM_IrqTH51Disable()	571
4.92.1.189 HAL_MCPWM_IrqTH51Enable()	572
4.92.1.190 HAL_MCPWM_IrqTH51Flg()	572
4.92.1.191 HAL_MCPWM_IrqTmr0()	572
4.92.1.192 HAL_MCPWM_IrqTmr0Clear()	572
4.92.1.193 HAL_MCPWM_IrqTmr0Disable()	572
4.92.1.194 HAL_MCPWM_IrqTmr0Enable()	573
4.92.1.195 HAL_MCPWM_IrqTmr0Flg()	573
4.92.1.196 HAL_MCPWM_IrqTmr1()	573
4.92.1.197 HAL_MCPWM_IrqTmr1Clear()	574
4.92.1.198 HAL_MCPWM_IrqTmr1Disable()	574
4.92.1.199 HAL_MCPWM_IrqTmr1Enable()	574
4.92.1.200 HAL_MCPWM_IrqTmr1Flg()	575
4.92.1.201 HAL_MCPWM_IrqTmr2()	575
4.92.1.202 HAL_MCPWM_IrqTmr2Clear()	575
4.92.1.203 HAL_MCPWM_IrqTmr2Disable()	576
4.92.1.204 HAL_MCPWM_IrqTmr2Enable()	576
4.92.1.205 HAL_MCPWM_IrqTmr2Flg()	576
4.92.1.206 HAL_MCPWM_IrqTmr3()	577
4.92.1.207 HAL_MCPWM_IrqTmr3Clear()	577
4.92.1.208 HAL_MCPWM_IrqTmr3Disable()	577
4.92.1.209 HAL_MCPWM_IrqTmr3Enable()	577
4.92.1.210 HAL_MCPWM_IrqTmr3Flg()	578
4.92.1.211 HAL_MCPWM_IrqUpdate()	578

4.92.1.212 HAL_MCPWM_IrqUpdateClear()	578
4.92.1.213 HAL_MCPWM_IrqUpdateDisable()	579
4.92.1.214 HAL_MCPWM_IrqUpdateEnable()	579
4.92.1.215 HAL_MCPWM_IrqUpdateFlg()	579
4.92.1.216 HAL_MCPWM_IrqZeroCross()	580
4.92.1.217 HAL_MCPWM_IrqZeroCrossClear()	580
4.92.1.218 HAL_MCPWM_IrqZeroCrossDisable()	580
4.92.1.219 HAL_MCPWM_IrqZeroCrossEnable()	581
4.92.1.220 HAL_MCPWM_IrqZeroCrossFlg()	581
4.92.1.221 HAL_SPI_IrqDone()	581
4.92.1.222 HAL_SPI_IrqDoneClear()	582
4.92.1.223 HAL_SPI_IrqDoneDisable()	582
4.92.1.224 HAL_SPI_IrqDoneEnable()	582
4.92.1.225 HAL_SPI_IrqDoneFlg()	583
4.92.1.226 HAL_SPI_IrqErr()	583
4.92.1.227 HAL_SPI_IrqErrClear()	583
4.92.1.228 HAL_SPI_IrqErrDisable()	584
4.92.1.229 HAL_SPI_IrqErrEnable()	584
4.92.1.230 HAL_SPI_IrqErrFlg()	584
4.92.1.231 HAL_SPI_IrqOverflow()	585
4.92.1.232 HAL_SPI_IrqOverflowClear()	585
4.92.1.233 HAL_SPI_IrqOverflowDisable()	585
4.92.1.234 HAL_SPI_IrqOverflowEnable()	586
4.92.1.235 HAL_SPI_IrqOverflowFlg()	586
4.92.1.236 HAL_SW_Irq()	586
4.92.1.237 HAL_SW_IrqClear()	586
4.92.1.238 HAL_SW_IrqDisable()	587
4.92.1.239 HAL_SW_IrqEnable()	587
4.92.1.240 HAL_SW_IrqFlg()	587
4.92.1.241 HAL_SW_IrqTrig()	587
4.92.1.242 HAL_TIMER_IrqCompare0()	587
4.92.1.243 HAL_TIMER_IrqCompare0Clear()	588
4.92.1.244 HAL_TIMER_IrqCompare0Disable()	588
4.92.1.245 HAL_TIMER_IrqCompare0Enable()	588
4.92.1.246 HAL_TIMER_IrqCompare0Flg()	588
4.92.1.247 HAL_TIMER_IrqCompare1()	589
4.92.1.248 HAL_TIMER_IrqCompare1Clear()	589
4.92.1.249 HAL_TIMER_IrqCompare1Disable()	589
4.92.1.250 HAL_TIMER_IrqCompare1Enable()	590
4.92.1.251 HAL_TIMER_IrqCompare1Flg()	590
4.92.1.252 HAL_TIMER_IrqOverflow()	590
4.92.1.253 HAL_TIMER_IrqOverflowClear()	591

4.92.1.254 HAL_TIMER_IrqOverflowDisable()	591
4.92.1.255 HAL_TIMER_IrqOverflowEnable()	591
4.92.1.256 HAL_TIMER_IrqOverflowFlg()	592
4.92.1.257 HAL_UART_IrqAbd()	592
4.92.1.258 HAL_UART_IrqAbdClear()	592
4.92.1.259 HAL_UART_IrqAbdDisable()	592
4.92.1.260 HAL_UART_IrqAbdEnable()	593
4.92.1.261 HAL_UART_IrqAbdFlg()	593
4.92.1.262 HAL_UART_IrqCheckErr()	593
4.92.1.263 HAL_UART_IrqCheckErrClear()	593
4.92.1.264 HAL_UART_IrqCheckErrDisable()	594
4.92.1.265 HAL_UART_IrqCheckErrEnable()	594
4.92.1.266 HAL_UART_IrqCheckErrFlg()	594
4.92.1.267 HAL_UART_IrqIdle()	595
4.92.1.268 HAL_UART_IrqIdleClear()	595
4.92.1.269 HAL_UART_IrqIdleDisable()	595
4.92.1.270 HAL_UART_IrqIdleEnable()	595
4.92.1.271 HAL_UART_IrqIdleFlg()	595
4.92.1.272 HAL_UART_IrqLbd()	595
4.92.1.273 HAL_UART_IrqLbdClear()	595
4.92.1.274 HAL_UART_IrqLbdDisable()	596
4.92.1.275 HAL_UART_IrqLbdEnable()	596
4.92.1.276 HAL_UART_IrqLbdFlg()	596
4.92.1.277 HAL_UART_IrqRxDone()	596
4.92.1.278 HAL_UART_IrqRxDoneClear()	596
4.92.1.279 HAL_UART_IrqRxDoneDisable()	597
4.92.1.280 HAL_UART_IrqRxDoneEnable()	597
4.92.1.281 HAL_UART_IrqRxDoneFlg()	597
4.92.1.282 HAL_UART_IrqRxOv()	598
4.92.1.283 HAL_UART_IrqRxOvClear()	598
4.92.1.284 HAL_UART_IrqRxOvDisable()	598
4.92.1.285 HAL_UART_IrqRxOvEnable()	598
4.92.1.286 HAL_UART_IrqRxOvFlg()	598
4.92.1.287 HAL_UART_IrqStopErr()	598
4.92.1.288 HAL_UART_IrqStopErrClear()	599
4.92.1.289 HAL_UART_IrqStopErrDisable()	599
4.92.1.290 HAL_UART_IrqStopErrEnable()	599
4.92.1.291 HAL_UART_IrqStopErrFlg()	600
4.92.1.292 HAL_UART_IrqTxBufEmpty()	600
4.92.1.293 HAL_UART_IrqTxBufEmptyClear()	600
4.92.1.294 HAL_UART_IrqTxBufEmptyDisable()	601
4.92.1.295 HAL_UART_IrqTxBufEmptyEnable()	601

4.92.1.296 HAL_UART_IrqTxBufEmptyFlg()	601
4.92.1.297 HAL_UART_IrqTxDone()	602
4.92.1.298 HAL_UART_IrqTxDoneClear()	602
4.92.1.299 HAL_UART_IrqTxDoneDisable()	602
4.92.1.300 HAL_UART_IrqTxDoneEnable()	603
4.92.1.301 HAL_UART_IrqTxDoneFlg()	603
4.92.1.302 HAL_UART_IrqTxOv()	603
4.92.1.303 HAL_UART_IrqTxOvClear()	604
4.92.1.304 HAL_UART_IrqTxOvDisable()	604
4.92.1.305 HAL_UART_IrqTxOvEnable()	604
4.92.1.306 HAL_UART_IrqTxOvFlg()	604
4.93 hal_lks32mc08x_irqfun.c	604
4.94 source/hal_lks32mc08x_iwdg.c 文件参考	638
4.94.1 函数说明	639
4.94.1.1 HAL_IWDG_Disable()	639
4.94.1.2 HAL_IWDG_Enable()	639
4.94.1.3 HAL_IWDG_Feed()	640
4.94.1.4 HAL_IWDG_Reset()	640
4.94.1.5 HAL_IWDG_SetResetTimemS()	641
4.94.2 变量说明	641
4.94.2.1 hal_iwdg_cfg	641
4.95 hal_lks32mc08x_iwdg.c	641
4.96 source/hal_lks32mc08x_mcpwm.c 文件参考	642
4.96.1 函数说明	644
4.96.1.1 HAL_MCPWM_Disable()	644
4.96.1.2 HAL_MCPWM_Enable()	645
4.96.1.3 HAL_MCPWM_GetClockDiv()	645
4.96.1.4 HAL_MCPWM_GetClockFreq()	645
4.96.1.5 HAL_MCPWM_GetFreq()	646
4.96.1.6 HAL_MCPWM_GetThCnt()	647
4.96.1.7 HAL_MCPWM_Reset()	647
4.96.1.8 HAL_MCPWM_SetAdcTrig()	648
4.96.1.9 HAL_MCPWM_SetAdcTrigQ15()	648
4.96.1.10 HAL_MCPWM_SetClockDiv()	649
4.96.1.11 HAL_MCPWM_SetDead()	649
4.96.1.12 HAL_MCPWM_SetDefaultState()	650
4.96.1.13 HAL_MCPWM_SetDrivePolarity()	650
4.96.1.14 HAL_MCPWM_SetDrivePolarityPort()	651
4.96.1.15 HAL_MCPWM_SetFailFilter()	651
4.96.1.16 HAL_MCPWM_SetFailSignal()	652
4.96.1.17 HAL_MCPWM_SetFreq()	652
4.96.1.18 HAL_MCPWM_SetGpioFail()	653

4.96.1.19 HAL_MCPWM_SetGpioOut()	653
4.96.1.20 HAL_MCPWM_SetOutputDisable()	654
4.96.1.21 HAL_MCPWM_SetOutputEnable()	654
4.96.1.22 HAL_MCPWM_SetOutState()	654
4.96.1.23 HAL_MCPWM_SetPwm()	655
4.96.1.24 HAL_MCPWM_SetPwmQ15()	656
4.96.1.25 HAL_MCPWM_SetRemap()	656
4.96.1.26 HAL_MCPWM_SetSwap()	656
4.96.1.27 HAL_MCPWM_SetSwapPort()	657
4.96.1.28 HAL_MCPWM_SetThCnt()	658
4.96.1.29 HAL_MCPWM_SetUpdateMode()	658
4.96.1.30 HAL_MCPWM_StartCounting()	658
4.96.1.31 HAL_MCPWM_StartCountingSync()	659
4.96.1.32 HAL_MCPWM_StopCounting()	660
4.96.1.33 HAL_MCPWM_UpdatePolarityState()	660
4.96.2 变量说明	661
4.96.2.1 HAL_MCPWM_STATE	661
4.96.2.2 HAL_MCPWM_STATE_INIT_FLG	661
4.97 hal_lks32mc08x_mcpwm.c	661
4.98 source/hal_lks32mc08x_opa.c 文件参考	667
4.98.1 函数说明	668
4.98.1.1 HAL_OPA_Disable()	668
4.98.1.2 HAL_OPA_Enable()	668
4.98.1.3 HAL_OPA_Out2GpioDisable()	668
4.98.1.4 HAL_OPA_Out2GpioEnable()	669
4.98.1.5 HAL_OPA_Reset()	669
4.98.1.6 HAL_OPA_SetGain()	669
4.98.1.7 HAL_OPA_SetGpioIn()	669
4.98.1.8 HAL_OPA_SetGpioOut()	670
4.99 hal_lks32mc08x_opa.c	671
4.100 source/hal_lks32mc08x_qep.c 文件参考	673
4.100.1 函数说明	674
4.100.1.1 HAL_QEP_Disable()	674
4.100.1.2 HAL_QEP_Enable()	674
4.100.1.3 HAL_QEP_GetClockDiv()	674
4.100.1.4 HAL_QEP_GetClockFreq()	675
4.100.1.5 HAL_QEP_ReadCnt()	675
4.100.1.6 HAL_QEP_Reset()	676
4.100.1.7 HAL_QEP_ResetCnt()	676
4.100.1.8 HAL_QEP_SetClockDiv()	676
4.100.1.9 HAL_QEP_SetGpio()	677
4.100.1.10 HAL_QEP_SetMode()	677

4.100.1.11 HAL_QEP_SetTh()	677
4.101 hal_lks32mc08x_qep.c	678
4.102 source/hal_lks32mc08x_reg.c 文件参考	679
4.102.1 变量说明	680
4.102.1.1 HAL_ADC	680
4.102.1.2 HAL_CAN	681
4.102.1.3 HAL_CMP	681
4.102.1.4 HAL_CRC	681
4.102.1.5 HAL_DMA	681
4.102.1.6 HAL_DSP	681
4.102.1.7 HAL_FLASH	681
4.102.1.8 HAL_GPIO	681
4.102.1.9 HAL_HALL	682
4.102.1.10 HAL_I2C	682
4.102.1.11 HAL_IWDG	682
4.102.1.12 HAL_MCPWM	682
4.102.1.13 HAL_QEP	682
4.102.1.14 HAL_SPI	682
4.102.1.15 HAL_SYS	682
4.102.1.16 HAL_TIMER	683
4.102.1.17 HAL_UART	683
4.103 hal_lks32mc08x_reg.c	683
4.104 source/hal_lks32mc08x_reset.c 文件参考	683
4.104.1 函数说明	684
4.104.1.1 HAL_RESET_ResetCore()	684
4.104.1.2 HAL_RESET_ResetGlobal()	684
4.104.1.3 HAL_RESET_ResetRegister()	685
4.105 hal_lks32mc08x_reset.c	685
4.106 source/hal_lks32mc08x_spi.c 文件参考	685
4.106.1 函数说明	687
4.106.1.1 HAL_SPI_Disable()	687
4.106.1.2 HAL_SPI_Enable()	687
4.106.1.3 HAL_SPI_GetBaud()	687
4.106.1.4 HAL_SPI_GetClockDiv()	688
4.106.1.5 HAL_SPI_GetClockFreq()	688
4.106.1.6 HAL_SPI_Reset()	689
4.106.1.7 HAL_SPI_RxData()	689
4.106.1.8 HAL_SPI_RxDataWidth16()	690
4.106.1.9 HAL_SPI_RxTxData()	691
4.106.1.10 HAL_SPI_RxTxDataWidth16()	691
4.106.1.11 HAL_SPI_SetBaud()	692
4.106.1.12 HAL_SPI_SetClockDiv()	692

4.106.1.13 HAL_SPI_SetGpioClk()	693
4.106.1.14 HAL_SPI_SetGpioCs()	693
4.106.1.15 HAL_SPI_SetGpioDi()	694
4.106.1.16 HAL_SPI_SetGpioDo()	695
4.106.1.17 HAL_SPI_SetGpioMiso()	695
4.106.1.18 HAL_SPI_SetGpioMosi()	696
4.106.1.19 HAL_SPI_SetMode()	697
4.106.1.20 HAL_SPI_TxData()	697
4.106.1.21 HAL_SPI_TxDataWidth16()	698
4.107 hal_lks32mc08x_spi.c	698
4.108 source/hal_lks32mc08x_timer.c 文件参考	702
4.108.1 函数说明	703
4.108.1.1 HAL_TIMER_Disable()	703
4.108.1.2 HAL_TIMER_Enable()	703
4.108.1.3 HAL_TIMER_GetClockDiv()	704
4.108.1.4 HAL_TIMER_GetClockFreq()	704
4.108.1.5 HAL_TIMER_GetFreq()	704
4.108.1.6 HAL_TIMER_GetThCnt()	705
4.108.1.7 HAL_TIMER_Reset()	705
4.108.1.8 HAL_TIMER_SetClockDiv()	705
4.108.1.9 HAL_TIMER_SetFreq()	705
4.108.1.10 HAL_TIMER_SetGpioIn()	706
4.108.1.11 HAL_TIMER_SetGpioOut()	707
4.108.1.12 HAL_TIMER_SetPeriodCount()	707
4.108.1.13 HAL_TIMER_SetPeriodmS()	708
4.108.1.14 HAL_TIMER_SetPerioduS()	708
4.108.1.15 HAL_TIMER_SetPwm()	709
4.108.1.16 HAL_TIMER_SetPwmQ15()	709
4.108.1.17 HAL_TIMER_SetWorkMode()	710
4.108.1.18 HAL_TIMER_StartCounting()	710
4.108.1.19 HAL_TIMER_StopCounting()	710
4.108.2 变量说明	710
4.108.2.1 hal_timer_ena	710
4.109 hal_lks32mc08x_timer.c	711
4.110 source/hal_lks32mc08x_tmp.c 文件参考	714
4.110.1 函数说明	715
4.110.1.1 HAL_TMP_Adc2Temperature()	715
4.110.1.2 HAL_TMP_Adc2Temperature01()	716
4.110.1.3 HAL_TMP_Disable()	717
4.110.1.4 HAL_TMP_Enable()	717
4.110.1.5 HAL_TMP_GetTemperature()	718
4.110.1.6 HAL_TMP_GetTemperature01()	718

4.110.1.7 HAL_TMP_Reset()	719
4.110.2 变量说明	719
4.110.2.1 tmp_coef_a	719
4.110.2.2 tmp_coef_b	720
4.111 hal_lks32mc08x_tmp.c	720
4.112 source/hal_lks32mc08x_trim.c 文件参考	721
4.112.1 函数说明	721
4.112.1.1 HAL_TRIM_Read()	721
4.112.1.2 HAL_TRIM_ReadId()	722
4.112.1.3 HAL_TRIM_ReadIdCrc32()	723
4.112.1.4 HAL_TRIM_ReadMcu08()	723
4.113 hal_lks32mc08x_trim.c	724
4.114 source/hal_lks32mc08x_uart.c 文件参考	725
4.114.1 函数说明	726
4.114.1.1 HAL_UART_Disable()	726
4.114.1.2 HAL_UART_Enable()	726
4.114.1.3 HAL_UART_GetBaud()	726
4.114.1.4 HAL_UART_GetClockDiv()	726
4.114.1.5 HAL_UART_GetClockFreq()	727
4.114.1.6 HAL_UART_ReadBuffer()	727
4.114.1.7 HAL_UART_Reset()	727
4.114.1.8 HAL_UART_SetBaud()	728
4.114.1.9 HAL_UART_SetClockDiv()	728
4.114.1.10 HAL_UART_SetGpioRx()	728
4.114.1.11 HAL_UART_SetGpioTx()	729
4.114.1.12 HAL_UART_SetMode()	729
4.114.1.13 HAL_UART_SetPolarity()	729
4.114.1.14 HAL_UART_TxData()	730
4.114.1.15 HAL_UART_WriteBuffer()	730
4.114.2 变量说明	730
4.114.2.1 hal_baud_set	730
4.115 hal_lks32mc08x_uart.c	730
4.116 source/hal_lks32mc08x_wake.c 文件参考	734
4.116.1 函数说明	735
4.116.1.1 HAL_WAKE_DeepSleepIoTimeWakeUp()	735
4.116.1.2 HAL_WAKE_DeepSleepIoWake()	735
4.116.1.3 HAL_WAKE_DeepSleepTimeWakeUp()	736
4.117 hal_lks32mc08x_wake.c	736

Chapter 1

结构体索引

1.1 结构体

这里列出了所有结构体，并附带简要说明：

CMP_TypeDef	
ADC寄存器结构体句柄	5
CRC_Model_Params	6
HAL_ADC_TypeDef	
ADC0寄存器定义	7
HAL_CAN_TypeDef	
CAN寄存器定义	12
HAL_CMP_TypeDef	
CMP寄存器定义	19
HAL_CRC_TypeDef	
CRC寄存器定义	20
HAL_DMA_TypeDef	
DMA寄存器定义	21
HAL_DSP_TypeDef	
DSP寄存器定义	23
HAL_FLASH_TypeDef	
FLASH寄存器定义	25
HAL_GPIO_TypeDef	
GPIO寄存器定义	27
HAL_HALL_TypeDef	
HALL寄存器定义	30
HAL_I2C_TypeDef	
I2C寄存器定义	31
HAL_IWDG_TypeDef	
HAL_MCPWM_TypeDef	
MCPWM寄存器定义	33
HAL_QEP_TypeDef	
QEP寄存器定义	38
HAL_SPI_TypeDef	
SPI寄存器定义	39
HAL_SYS_TypeDef	
SYS寄存器定义	41
HAL_TIMER_TypeDef	
TIMER寄存器定义	49
HAL_UART_TypeDef	
UART寄存器定义	51

Chapter 2

文件索引

2.1 文件列表

这里列出了所有文件，并附带简要说明:

hal_lks32mc08x.c	55
hal_lks32mc08x.h	56
hal_lksmcu.h	58
include/hal_lks32mc08x_reg.h	58
include/hal_lksmcu_adc.h	68
include/hal_lksmcu_can.h	84
include/hal_lksmcu_clk.h	101
include/hal_lksmcu_cmp.h	105
include/hal_lksmcu_crc.h	114
include/hal_lksmcu_dac.h	117
include/hal_lksmcu_dma.h	125
include/hal_lksmcu_dsp.h	130
include/hal_lksmcu_flash.h	135
include/hal_lksmcu_gpio.h	145
include/hal_lksmcu_hall.h	159
include/hal_lksmcu_i2c.h	167
include/hal_lksmcu_init.h	175
include/hal_lksmcu_irq.h	178
include/hal_lksmcu_irqfun.h	188
include/hal_lksmcu_iwdg.h	291
include/hal_lksmcu_mcpwm.h	294
include/hal_lksmcu_opa.h	317
include/hal_lksmcu_qep.h	323
include/hal_lksmcu_reg.h	
HAL库配置和内部参数	330
include/hal_lksmcu_reset.h	333
include/hal_lksmcu_spi.h	335
include/hal_lksmcu_timer.h	350
include/hal_lksmcu_tmp.h	360
include/hal_lksmcu_trim.h	367
include/hal_lksmcu_uart.h	370
include/hal_lksmcu_wake.h	377
source/hal_lks32mc08x_adc.c	
Lks32mc08x	381
source/hal_lks32mc08x_can.c	397

source/hal_lks32mc08x_clk.c	418
source/hal_lks32mc08x_cmp.c	422
source/hal_lks32mc08x_crc.c	430
source/hal_lks32mc08x_dac.c	436
source/hal_lks32mc08x_dma.c	444
source/hal_lks32mc08x_dsp.c	450
source/hal_lks32mc08x_flash.c	455
source/hal_lks32mc08x_gpio.c	469
source/hal_lks32mc08x_hall.c	480
source/hal_lks32mc08x_i2c.c	489
source/hal_lks32mc08x_init.c	499
source/hal_lks32mc08x_irq.c	501
source/hal_lks32mc08x_irqfun.c	505
source/hal_lks32mc08x_iwdg.c	638
source/hal_lks32mc08x_mcpwm.c	642
source/hal_lks32mc08x_opa.c	667
source/hal_lks32mc08x_qep.c	673
source/hal_lks32mc08x_reg.c	679
source/hal_lks32mc08x_reset.c	683
source/hal_lks32mc08x_spi.c	685
source/hal_lks32mc08x_timer.c	702
source/hal_lks32mc08x_tmp.c	714
source/hal_lks32mc08x_trim.c	721
source/hal_lks32mc08x_uart.c	725
source/hal_lks32mc08x_wake.c	734

Chapter 3

结构体说明

3.1 CMP_TypeDef结构体 参考

ADC寄存器结构体句柄

成员变量

- volatile uint32_t **IE**
0x00 比较器中断使能寄存器
- volatile uint32_t **IF**
0x04 比较器中断标志寄存器
- volatile uint32_t **TCLK**
0x08 比较器分频时钟控制寄存器
- volatile uint32_t **CFG**
0x0C 比较器控制寄存器
- volatile uint32_t **BLCWIN**
0x10 比较器开窗控制寄存器

3.1.1 详细描述

ADC寄存器结构体句柄

在文件 [hal_lks32mc08x_cmp.c](#) 第 10 行定义.

3.1.2 结构体成员变量说明

3.1.2.1 BLCWIN

volatile uint32_t BLCWIN

0x10 比较器开窗控制寄存器

在文件 [hal_lks32mc08x_cmp.c](#) 第 16 行定义.

3.1.2.2 CFG

```
volatile uint32_t CFG
```

0x0C 比较器控制寄存器

在文件 [hal_lks32mc08x_cmp.c](#) 第 15 行定义.

3.1.2.3 IE

```
volatile uint32_t IE
```

0x00 比较器中断使能寄存器

在文件 [hal_lks32mc08x_cmp.c](#) 第 12 行定义.

3.1.2.4 IF

```
volatile uint32_t IF
```

0x04 比较器中断标志寄存器

在文件 [hal_lks32mc08x_cmp.c](#) 第 13 行定义.

3.1.2.5 TCLK

```
volatile uint32_t TCLK
```

0x08 比较器分频时钟控制寄存器

在文件 [hal_lks32mc08x_cmp.c](#) 第 14 行定义.

3.2 CRC_Model_Params结构体 参考

成员变量

- uint32_t polynomial
- uint32_t initValue
- uint32_t xorOut
- uint8_t reflectIn
- uint8_t reflectOut
- CRC_Width width

3.2.1 详细描述

在文件 [hal_lks32mc08x_crc.c](#) 第 14 行定义.

3.2.2 结构体成员变量说明

3.2.2.1 initValue

`uint32_t initValue`

在文件 `hal_lks32mc08x_crc.c` 第 17 行定义.

3.2.2.2 polynomial

`uint32_t polynomial`

在文件 `hal_lks32mc08x_crc.c` 第 16 行定义.

3.2.2.3 reflectIn

`uint8_t reflectIn`

在文件 `hal_lks32mc08x_crc.c` 第 19 行定义.

3.2.2.4 reflectOut

`uint8_t reflectOut`

在文件 `hal_lks32mc08x_crc.c` 第 20 行定义.

3.2.2.5 width

`CRC_Width width`

在文件 `hal_lks32mc08x_crc.c` 第 21 行定义.

3.2.2.6 xorOut

`uint32_t xorOut`

在文件 `hal_lks32mc08x_crc.c` 第 18 行定义.

3.3 HAL_ADC_TypeDef结构体 参考

ADC0寄存器定义

```
#include <hal_lks32mc08x_reg.h>
```

成员变量

- volatile uint32_t **DAT** [20]
数据寄存器0
- volatile uint32_t **CHN** [10]
采样信号来源寄存器0
- volatile uint32_t **CHNT0**
分段通道数寄存器0 设置每段采样数量
- volatile uint32_t **CHNT1**
分段通道数寄存器1 设置每段采样数量
- volatile uint32_t **null** [2]
保留
- volatile uint32_t **GAIN_0**
增益选择寄存器0
- volatile uint32_t **GAIN_1**
增益选择寄存器1
- volatile uint32_t **DC_A0**
直流偏置寄存器0 采样保持电路A对应2/3倍增益通道偏置
- volatile uint32_t **DC_A1**
直流偏置寄存器1 采样保持电路A对应1倍增益通道偏置
- volatile uint32_t **AMC_A0**
增益校正寄存器0 采样保持电路A对应2/3倍增益通道增益校正
- volatile uint32_t **AMC_A1**
增益校正寄存器0 采样保持电路A对应1倍增益通道增益校正
- volatile uint32_t **DC_B0**
直流偏置寄存器0 采样保持电路B对应2/3倍增益通道偏置
- volatile uint32_t **DC_B1**
直流偏置寄存器1 采样保持电路B对应1倍增益通道偏置
- volatile uint32_t **AMC_B0**
增益校正寄存器0 采样保持电路B对应2/3倍增益通道增益校正
- volatile uint32_t **AMC_B1**
增益校正寄存器0 采样保持电路B对应1倍增益通道增益校正
- volatile uint32_t **IE**
中断寄存器
- volatile uint32_t **IF**
中断标志位寄存器
- volatile uint32_t **CFG**
模式配置寄存器
- volatile uint32_t **TRIG**
触发控制寄存器
- volatile uint32_t **SWT**
软件触发寄存器
- volatile uint32_t **DAT0_TH**
通道0阈值寄存器

3.3.1 详细描述

ADC0寄存器定义

在文件 [hal_lks32mc08x_reg.h](#) 第 149 行定义.

3.3.2 结构体成员变量说明

3.3.2.1 AMC_A0

```
volatile uint32_t AMC_A0
```

增益校正寄存器0 采样保持电路A对应2/3倍增益通道增益校正
在文件 [hal_lks32mc08x_reg.h](#) 第 160 行定义.

3.3.2.2 AMC_A1

```
volatile uint32_t AMC_A1
```

增益校正寄存器0 采样保持电路A对应 1 倍增益通道增益校正
在文件 [hal_lks32mc08x_reg.h](#) 第 161 行定义.

3.3.2.3 AMC_B0

```
volatile uint32_t AMC_B0
```

增益校正寄存器0 采样保持电路B对应2/3倍增益通道增益校正
在文件 [hal_lks32mc08x_reg.h](#) 第 164 行定义.

3.3.2.4 AMC_B1

```
volatile uint32_t AMC_B1
```

增益校正寄存器0 采样保持电路B对应 1 倍增益通道增益校正
在文件 [hal_lks32mc08x_reg.h](#) 第 165 行定义.

3.3.2.5 CFG

```
volatile uint32_t CFG
```

模式配置寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 168 行定义.

3.3.2.6 CHN

```
volatile uint32_t CHN[10]
```

采样信号来源寄存器0

在文件 [hal_lks32mc08x_reg.h](#) 第 152 行定义.

3.3.2.7 CHNT0

```
volatile uint32_t CHNT0
```

分段通道数寄存器0 设置每段采样数量

在文件 [hal_lks32mc08x.reg.h](#) 第 153 行定义.

3.3.2.8 CHNT1

```
volatile uint32_t CHNT1
```

分段通道数寄存器1 设置每段采样数量

在文件 [hal_lks32mc08x.reg.h](#) 第 154 行定义.

3.3.2.9 DAT

```
volatile uint32_t DAT[20]
```

数据寄存器0

在文件 [hal_lks32mc08x.reg.h](#) 第 151 行定义.

3.3.2.10 DAT0.TH

```
volatile uint32_t DAT0.TH
```

通道0阈值寄存器

在文件 [hal_lks32mc08x.reg.h](#) 第 171 行定义.

3.3.2.11 DC_A0

```
volatile uint32_t DC_A0
```

直流偏置寄存器0 采样保持电路A对应2/3倍增益通道偏置

在文件 [hal_lks32mc08x.reg.h](#) 第 158 行定义.

3.3.2.12 DC_A1

```
volatile uint32_t DC_A1
```

直流偏置寄存器1 采样保持电路A对应 1 倍增益通道偏置

在文件 [hal_lks32mc08x.reg.h](#) 第 159 行定义.

3.3.2.13 DC_B0

```
volatile uint32_t DC_B0
```

直流偏置寄存器0 采样保持电路B对应2/3倍增益通道偏置

在文件 [hal_lks32mc08x_reg.h](#) 第 162 行定义.

3.3.2.14 DC_B1

```
volatile uint32_t DC_B1
```

直流偏置寄存器1 采样保持电路B对应1倍增益通道偏置

在文件 [hal_lks32mc08x_reg.h](#) 第 163 行定义.

3.3.2.15 GAIN_0

```
volatile uint32_t GAIN_0
```

增益选择寄存器0

在文件 [hal_lks32mc08x_reg.h](#) 第 156 行定义.

3.3.2.16 GAIN_1

```
volatile uint32_t GAIN_1
```

增益选择寄存器1

在文件 [hal_lks32mc08x_reg.h](#) 第 157 行定义.

3.3.2.17 IE

```
volatile uint32_t IE
```

中断寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 166 行定义.

3.3.2.18 IF

```
volatile uint32_t IF
```

中断标志位寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 167 行定义.

3.3.2.19 null

```
volatile uint32_t null[2]
```

保留

在文件 [hal_lks32mc08x_reg.h](#) 第 155 行定义.

3.3.2.20 SWT

```
volatile uint32_t SWT
```

软件触发寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 170 行定义.

3.3.2.21 TRIG

```
volatile uint32_t TRIG
```

触发控制寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 169 行定义.

3.4 HAL_CAN_TypeDef结构体 参考

CAN寄存器定义

```
#include <hal_lks32mc08x_reg.h>
```

成员变量

- volatile uint32_t [MOD](#)
CAN 模式寄存器
- volatile uint32_t [CMR](#)
CAN 命令寄存器
- volatile uint32_t [SR](#)
CAN 状态寄存器
- volatile uint32_t [IR](#)
CAN 中断状态寄存器
- volatile uint32_t [IER](#)
CAN 中断控制寄存器
- volatile uint32_t [null0](#)
- volatile uint32_t [BTR0](#)
CAN 总线时序控制寄存器 0
- volatile uint32_t [BTR1](#)
CAN 总线时序控制寄存器 1
- volatile uint32_t [null1](#)

- volatile uint32_t null2
- volatile uint32_t null3
- volatile uint32_t ALC
CAN 仲裁丢失捕捉寄存器
- volatile uint32_t ECC
CAN 错误码捕捉寄存器
- volatile uint32_t EWLR
CAN 错误&警告门限值设置寄存器
- volatile uint32_t RXERR
CAN 接收错误计数器
- volatile uint32_t TXERR
CAN 发送错误计数器
- volatile uint32_t TXRX0
正常工作模式下, CAN 发送帧格式寄存器0/CAN 接收帧格式寄存器0
- volatile uint32_t TXRX1
正常工作模式下, CAN 发送数据寄存器0/CAN 接收数据寄存器1
- volatile uint32_t TXRX2
正常工作模式下, CAN 发送数据寄存器2/CAN 接收数据寄存器2
- volatile uint32_t TXRX3
正常工作模式下, CAN 发送数据寄存器3/CAN 接收数据寄存器3
- volatile uint32_t TXRX4
正常工作模式下, CAN 发送数据寄存器4/CAN 接收数据寄存器4
- volatile uint32_t TXRX5
正常工作模式下, CAN 发送数据寄存器5/CAN 接收数据寄存器5
- volatile uint32_t TXRX6
正常工作模式下, CAN 发送数据寄存器6/CAN 接收数据寄存器6
- volatile uint32_t TXRX7
正常工作模式下, CAN 发送数据寄存器7/CAN 接收数据寄存器7
- volatile uint32_t TXRX8
正常工作模式下, CAN 发送数据寄存器8/CAN 接收数据寄存器8
- volatile uint32_t TXRX9
正常工作模式下, CAN 发送数据寄存器9/CAN 接收数据寄存器9
- volatile uint32_t TXRXA
正常工作模式下, CAN 发送数据寄存器10/CAN 接收数据寄存器10
- volatile uint32_t TXRXB
正常工作模式下, CAN 发送数据寄存器11/CAN 接收数据寄存器11
- volatile uint32_t TXRXC
正常工作模式下, CAN 发送数据寄存器12/CAN 接收数据寄存器12
- volatile uint32_t RMC
CAN FIFO 有效接收信息计数器
- volatile uint32_t RBSA
CAN 第一条有效接收信息在 FIFO 中的地址寄存器

3.4.1 详细描述

CAN寄存器定义

在文件 [hal_lks32mc08x_reg.h](#) 第 284 行定义.

3.4.2 结构体成员变量说明

3.4.2.1 ALC

```
volatile uint32_t ALC
```

CAN 仲裁丢失捕捉寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 299 行定义.

3.4.2.2 BTR0

```
volatile uint32_t BTR0
```

CAN 总线时序控制寄存器 0

在文件 [hal_lks32mc08x_reg.h](#) 第 293 行定义.

3.4.2.3 BTR1

```
volatile uint32_t BTR1
```

CAN 总线时序控制寄存器 1

在文件 [hal_lks32mc08x_reg.h](#) 第 294 行定义.

3.4.2.4 CMR

```
volatile uint32_t CMR
```

CAN 命令寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 287 行定义.

3.4.2.5 ECC

```
volatile uint32_t ECC
```

CAN 错误码捕捉寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 301 行定义.

3.4.2.6 EWLR

```
volatile uint32_t EWLR
```

CAN 错误&警告门限值设置寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 302 行定义.

3.4.2.7 IER

```
volatile uint32_t IER
```

CAN 中断控制寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 291 行定义.

3.4.2.8 IR

```
volatile uint32_t IR
```

CAN 中断状态寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 289 行定义.

3.4.2.9 MOD

```
volatile uint32_t MOD
```

CAN 模式寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 286 行定义.

3.4.2.10 null0

```
volatile uint32_t null0
```

在文件 [hal_lks32mc08x_reg.h](#) 第 292 行定义.

3.4.2.11 null1

```
volatile uint32_t null1
```

在文件 [hal_lks32mc08x_reg.h](#) 第 296 行定义.

3.4.2.12 null2

```
volatile uint32_t null2
```

在文件 [hal_lks32mc08x_reg.h](#) 第 297 行定义.

3.4.2.13 null3

```
volatile uint32_t null3
```

在文件 [hal_lks32mc08x_reg.h](#) 第 298 行定义.

3.4.2.14 RBSA

```
volatile uint32_t RBSA
```

CAN 第一条有效接收信息在 FIFO 中的地址寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 323 行定义.

3.4.2.15 RMC

```
volatile uint32_t RMC
```

CAN FIFO 有效接收信息计数器

在文件 [hal_lks32mc08x_reg.h](#) 第 322 行定义.

3.4.2.16 RXERR

```
volatile uint32_t RXERR
```

CAN 接收错误计数器

在文件 [hal_lks32mc08x_reg.h](#) 第 303 行定义.

3.4.2.17 SR

```
volatile uint32_t SR
```

CAN 状态寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 288 行定义.

3.4.2.18 TXERR

```
volatile uint32_t TXERR
```

CAN 发送错误计数器

在文件 [hal_lks32mc08x_reg.h](#) 第 304 行定义.

3.4.2.19 TXRX0

```
volatile uint32_t TXRX0
```

正常工作模式下, CAN 发送帧格式寄存器0/CAN 接收帧格式寄存器0

在文件 [hal_lks32mc08x_reg.h](#) 第 306 行定义.

3.4.2.20 TXRX1

```
volatile uint32_t TXRX1
```

正常工作模式下，CAN 发送数据寄存器0/CAN 接收数据寄存器1

在文件 [hal_lks32mc08x_reg.h](#) 第 307 行定义.

3.4.2.21 TXRX2

```
volatile uint32_t TXRX2
```

正常工作模式下，CAN 发送数据寄存器2/CAN 接收数据寄存器2

在文件 [hal_lks32mc08x_reg.h](#) 第 308 行定义.

3.4.2.22 TXRX3

```
volatile uint32_t TXRX3
```

正常工作模式下，CAN 发送数据寄存器3/CAN 接收数据寄存器3

在文件 [hal_lks32mc08x_reg.h](#) 第 309 行定义.

3.4.2.23 TXRX4

```
volatile uint32_t TXRX4
```

正常工作模式下，CAN 发送数据寄存器4/CAN 接收数据寄存器4

在文件 [hal_lks32mc08x_reg.h](#) 第 311 行定义.

3.4.2.24 TXRX5

```
volatile uint32_t TXRX5
```

正常工作模式下，CAN 发送数据寄存器5/CAN 接收数据寄存器5

在文件 [hal_lks32mc08x_reg.h](#) 第 312 行定义.

3.4.2.25 TXRX6

```
volatile uint32_t TXRX6
```

正常工作模式下，CAN 发送数据寄存器6/CAN 接收数据寄存器6

在文件 [hal_lks32mc08x_reg.h](#) 第 313 行定义.

3.4.2.26 TXRX7

```
volatile uint32_t TXRX7
```

正常工作模式下, CAN 发送数据寄存器7/CAN 接收数据寄存器7

在文件 [hal_lks32mc08x_reg.h](#) 第 314 行定义.

3.4.2.27 TXRX8

```
volatile uint32_t TXRX8
```

正常工作模式下, CAN 发送数据寄存器8/CAN 接收数据寄存器8

在文件 [hal_lks32mc08x_reg.h](#) 第 316 行定义.

3.4.2.28 TXRX9

```
volatile uint32_t TXRX9
```

正常工作模式下, CAN 发送数据寄存器9/CAN 接收数据寄存器9

在文件 [hal_lks32mc08x_reg.h](#) 第 317 行定义.

3.4.2.29 TXRXA

```
volatile uint32_t TXRXA
```

正常工作模式下, CAN 发送数据寄存器10/CAN 接收数据寄存器10

在文件 [hal_lks32mc08x_reg.h](#) 第 318 行定义.

3.4.2.30 TXRXB

```
volatile uint32_t TXRXB
```

正常工作模式下, CAN 发送数据寄存器11/CAN 接收数据寄存器11

在文件 [hal_lks32mc08x_reg.h](#) 第 319 行定义.

3.4.2.31 TXRXC

```
volatile uint32_t TXRXC
```

正常工作模式下, CAN 发送数据寄存器 12/CAN 接收数据寄存器 12

在文件 [hal_lks32mc08x_reg.h](#) 第 321 行定义.

3.5 HAL_CMP_TypeDef结构体 参考

CMP寄存器定义

```
#include <hal_lks32mc08x_reg.h>
```

成员变量

- volatile uint32_t **IE**
0x00 中断使能寄存器
- volatile uint32_t **IF**
0x04 中断标志寄存器
- volatile uint32_t **TCLK**
0x08 时钟分频控制寄存器
- volatile uint32_t **CFG**
0x0C 配置寄存器
- volatile uint32_t **BLCWIN**
0x10 黑窗口控制寄存器
- volatile uint32_t **DATA**
0x14 数据寄存器

3.5.1 详细描述

CMP寄存器定义

在文件 [hal_lks32mc08x_reg.h](#) 第 114 行定义.

3.5.2 结构体成员变量说明

3.5.2.1 BLCWIN

```
volatile uint32_t BLCWIN
```

0x10 黑窗口控制寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 120 行定义.

3.5.2.2 CFG

```
volatile uint32_t CFG
```

0x0C 配置寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 119 行定义.

3.5.2.3 DATA

```
volatile uint32_t DATA
```

0x14 数据寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 121 行定义.

3.5.2.4 IE

```
volatile uint32_t IE
```

0x00 中断使能寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 116 行定义.

3.5.2.5 IF

```
volatile uint32_t IF
```

0x04 中断标志寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 117 行定义.

3.5.2.6 TCLK

```
volatile uint32_t TCLK
```

0x08 时钟分频控制寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 118 行定义.

3.6 HAL_CRC_TypeDef结构体 参考

CRC寄存器定义

```
#include <hal_lks32mc08x_reg.h>
```

成员变量

- volatile uint32_t [DR](#)
0x00 CRC数据
- volatile uint32_t [CR](#)
0x04 CRC控制
- volatile uint32_t [INIT](#)
0x08 CRC初始值
- volatile uint32_t [POL](#)
0x0C CRC系数

3.6.1 详细描述

CRC寄存器定义

在文件 [hal_lks32mc08x_reg.h](#) 第 257 行定义.

3.6.2 结构体成员变量说明

3.6.2.1 CR

```
volatile uint32_t CR
```

0x04 CRC控制

在文件 [hal_lks32mc08x_reg.h](#) 第 260 行定义.

3.6.2.2 DR

```
volatile uint32_t DR
```

0x00 CRC数据

在文件 [hal_lks32mc08x_reg.h](#) 第 259 行定义.

3.6.2.3 INIT

```
volatile uint32_t INIT
```

0x08 CRC初始值

在文件 [hal_lks32mc08x_reg.h](#) 第 261 行定义.

3.6.2.4 POL

```
volatile uint32_t POL
```

0x0C CRC系数

在文件 [hal_lks32mc08x_reg.h](#) 第 262 行定义.

3.7 HAL_DMA_TypeDef结构体 参考

DMA寄存器定义

```
#include <hal_lks32mc08x_reg.h>
```

成员变量

- volatile uint32_t **CCR**
DMA通道0通道配置寄存器
- volatile uint32_t **CTMS**
DMA通道0传输次数寄存器
- volatile uint32_t **CPAR**
DMA通道0外设地址寄存器
- volatile uint32_t **CMAR**
DMA通道0内存地址寄存器

3.7.1 详细描述

DMA寄存器定义

在文件 [hal_lks32mc08x_reg.h](#) 第 139 行定义.

3.7.2 结构体成员变量说明

3.7.2.1 CCR

```
volatile uint32_t CCR
```

DMA通道0通道配置寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 141 行定义.

3.7.2.2 CMAR

```
volatile uint32_t CMAR
```

DMA通道0内存地址寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 144 行定义.

3.7.2.3 CPAR

```
volatile uint32_t CPAR
```

DMA通道0外设地址寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 143 行定义.

3.7.2.4 CTMS

```
volatile uint32_t CTMS
```

DMA通道0传输次数寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 142 行定义.

3.8 HAL_DSP_TypeDef结构体 参考

DSP寄存器定义

```
#include <hal_lks32mc08x_reg.h>
```

成员变量

- volatile uint32_t **SC**
0x00 DSP被除数
- volatile uint32_t **THETA**
0x04 CORDIC $\check{c}$ st $\check{h}$ eta
- volatile uint32_t **XY**
0x08 CORDIC的坐标x,y
- volatile uint32_t **RESERVED**
- volatile uint32_t **SIN**
0x10 CORDIC $\check{c}$ ssin
- volatile uint32_t **COS**
0x14 CORDIC $\check{c}$ scos
- volatile uint32_t **MOD**
0x18 CORDIC的模
- volatile uint32_t **ATAN**
0x1C CORDIC $\check{c}$ sarctan
- volatile uint32_t **RESERVED1** [4]
- volatile uint32_t **RAD**
0x30 DSP被开方数
- volatile uint16_t **SQRT**
0x34 DSP平方根

3.8.1 详细描述

DSP寄存器定义

在文件 [hal_lks32mc08x_reg.h](#) 第 329 行定义.

3.8.2 结构体成员变量说明

3.8.2.1 ATAN

```
volatile uint32_t ATAN
```

0x1C CORDIC $\check{c}$ sarctan

在文件 [hal_lks32mc08x_reg.h](#) 第 338 行定义.

3.8.2.2 COS

```
volatile uint32_t COS
```

0x14 CORDIC的cos

在文件 [hal_lks32mc08x_reg.h](#) 第 336 行定义.

3.8.2.3 MOD

```
volatile uint32_t MOD
```

0x18 CORDIC的模

在文件 [hal_lks32mc08x_reg.h](#) 第 337 行定义.

3.8.2.4 RAD

```
volatile uint32_t RAD
```

0x30 DSP被开方数

在文件 [hal_lks32mc08x_reg.h](#) 第 340 行定义.

3.8.2.5 RESERVED

```
volatile uint32_t RESERVED
```

在文件 [hal_lks32mc08x_reg.h](#) 第 334 行定义.

3.8.2.6 RESERVED1

```
volatile uint32_t RESERVED1[4]
```

在文件 [hal_lks32mc08x_reg.h](#) 第 339 行定义.

3.8.2.7 SC

```
volatile uint32_t SC
```

0x00 DSP被除数

在文件 [hal_lks32mc08x_reg.h](#) 第 331 行定义.

3.8.2.8 SIN

volatile uint32_t SIN

0x10 CORDICçšsin

在文件 [hal_lks32mc08x_reg.h](#) 第 335 行定义.

3.8.2.9 SQRT

volatile uint16_t SQRT

0x34 DSP平方根

在文件 [hal_lks32mc08x_reg.h](#) 第 341 行定义.

3.8.2.10 THETA

volatile uint32_t THETA

0x04 CORDICçštheta

在文件 [hal_lks32mc08x_reg.h](#) 第 332 行定义.

3.8.2.11 XY

volatile uint32_t XY

0x08 CORDIC的坐标x,y

在文件 [hal_lks32mc08x_reg.h](#) 第 333 行定义.

3.9 HAL_FLASH_TypeDef结构体 参考

FLASH寄存器定义

```
#include <hal_lks32mc08x_reg.h>
```

成员变量

- volatile uint32_t [CFG](#)
0x00 Flash配置寄存器
- volatile uint32_t [ADDR](#)
0x04 Flash访问地址寄存器
- volatile uint32_t [WDATA](#)
0x08 Flash写数据寄存器
- volatile uint32_t [RDATA](#)
0x0C Flash读数据寄存器
- volatile uint32_t [ERASE](#)
0x10 Flash触发擦除操作寄存器
- volatile uint32_t [PROTECT](#)
0x14 读取更新Flash保护状态
- volatile uint32_t [READY](#)
0x18 Flash忙/空闲状态

3.9.1 详细描述

FLASH寄存器定义

在文件 [hal_lks32mc08x_reg.h](#) 第 75 行定义.

3.9.2 结构体成员变量说明

3.9.2.1 ADDR

```
volatile uint32_t ADDR
```

0x04 Flash访问地址寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 78 行定义.

3.9.2.2 CFG

```
volatile uint32_t CFG
```

0x00 Flash配置寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 77 行定义.

3.9.2.3 ERASE

```
volatile uint32_t ERASE
```

0x10 Flash触发擦除操作寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 81 行定义.

3.9.2.4 PROTECT

```
volatile uint32_t PROTECT
```

0x14 读取更新Flash保护状态

在文件 [hal_lks32mc08x_reg.h](#) 第 82 行定义.

3.9.2.5 RDATA

```
volatile uint32_t RDATA
```

0x0C Flash读数据寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 80 行定义.

3.9.2.6 READY

```
volatile uint32_t READY
```

0x18 Flash忙/空闲状态

在文件 [hal_lks32mc08x.reg.h](#) 第 83 行定义.

3.9.2.7 WDATA

```
volatile uint32_t WDATA
```

0x08 Flash写数据寄存器

在文件 [hal_lks32mc08x.reg.h](#) 第 79 行定义.

3.10 HAL_GPIO_TypeDef结构体 参考

GPIO寄存器定义

```
#include <hal_lks32mc08x.reg.h>
```

成员变量

- volatile uint32_t [PIE](#)
输入使能
- volatile uint32_t [POE](#)
输出使能
- volatile uint32_t [PDI](#)
输入数据
- volatile uint32_t [PDO](#)
输出数据
- volatile uint32_t [PUE](#)
上拉使能
- volatile uint32_t [null1](#)
- volatile uint32_t [PODE](#)
开漏使能
- volatile uint32_t [LCKR](#)
配置锁定
- volatile uint32_t [F3210](#)
功能选择 *GPIO.Pin.0-3*
- volatile uint32_t [F7654](#)
功能选择 *GPIO.Pin.4-7*
- volatile uint32_t [FBA98](#)
功能选择 *GPIO.Pin.8-11*
- volatile uint32_t [FFEDC](#)
功能选择 *GPIO.Pin.12-15*

3.10.1 详细描述

GPIO寄存器定义

在文件 [hal_lks32mc08x_reg.h](#) 第 238 行定义.

3.10.2 结构体成员变量说明

3.10.2.1 F3210

```
volatile uint32_t F3210
```

功能选择 GPIO.Pin_0-3

在文件 [hal_lks32mc08x_reg.h](#) 第 248 行定义.

3.10.2.2 F7654

```
volatile uint32_t F7654
```

功能选择 GPIO.Pin_4-7

在文件 [hal_lks32mc08x_reg.h](#) 第 249 行定义.

3.10.2.3 FBA98

```
volatile uint32_t FBA98
```

功能选择 GPIO.Pin_8-11

在文件 [hal_lks32mc08x_reg.h](#) 第 250 行定义.

3.10.2.4 FFEDC

```
volatile uint32_t FFEDC
```

功能选择 GPIO.Pin_12-15

在文件 [hal_lks32mc08x_reg.h](#) 第 251 行定义.

3.10.2.5 LCKR

```
volatile uint32_t LCKR
```

配置锁定

在文件 [hal_lks32mc08x_reg.h](#) 第 247 行定义.

3.10.2.6 null1

```
volatile uint32_t null1
```

在文件 [hal_lks32mc08x_reg.h](#) 第 245 行定义.

3.10.2.7 PDI

```
volatile uint32_t PDI
```

输入数据

在文件 [hal_lks32mc08x_reg.h](#) 第 242 行定义.

3.10.2.8 PDO

```
volatile uint32_t PDO
```

输出数据

在文件 [hal_lks32mc08x_reg.h](#) 第 243 行定义.

3.10.2.9 PIE

```
volatile uint32_t PIE
```

输入使能

在文件 [hal_lks32mc08x_reg.h](#) 第 240 行定义.

3.10.2.10 PODE

```
volatile uint32_t PODE
```

开漏使能

在文件 [hal_lks32mc08x_reg.h](#) 第 246 行定义.

3.10.2.11 POE

```
volatile uint32_t POE
```

输出使能

在文件 [hal_lks32mc08x_reg.h](#) 第 241 行定义.

3.10.2.12 PUE

```
volatile uint32_t PUE
```

上拉使能

在文件 [hal_lks32mc08x_reg.h](#) 第 244 行定义.

3.11 HAL_HALL_TypeDef结构体 参考

HALL寄存器定义

```
#include <hal_lks32mc08x_reg.h>
```

成员变量

- volatile uint32_t [CFG](#)
0x00 配置寄存器
- volatile uint32_t [INFO](#)
0x04 信息寄存器
- volatile uint32_t [WIDTH](#)
0x08 宽度寄存器
- volatile uint32_t [TH](#)
0x0C 阈值寄存器
- volatile uint32_t [CNT](#)
0x10 计数寄存器

3.11.1 详细描述

HALL寄存器定义

在文件 [hal_lks32mc08x_reg.h](#) 第 127 行定义.

3.11.2 结构体成员变量说明

3.11.2.1 CFG

```
volatile uint32_t CFG
```

0x00 配置寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 129 行定义.

3.11.2.2 CNT

`volatile uint32_t CNT`

0x10 计数寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 133 行定义.

3.11.2.3 INFO

`volatile uint32_t INFO`

0x04 信息寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 130 行定义.

3.11.2.4 TH

`volatile uint32_t TH`

0x0C 阈值寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 132 行定义.

3.11.2.5 WIDTH

`volatile uint32_t WIDTH`

0x08 宽度寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 131 行定义.

3.12 HAL_I2C_TypeDef结构体 参考

I2C寄存器定义

```
#include <hal_lks32mc08x_reg.h>
```

成员变量

- `volatile uint32_t ADDR`
0x00 设备地址寄存器
- `volatile uint32_t CFG`
0x04 配置寄存器
- `volatile uint32_t SCR`
0x08 状态控制寄存器
- `volatile uint32_t DATA`
0x0C 数据寄存器
- `volatile uint32_t MSCR`
0x10 主状态控制寄存器
- `volatile uint32_t BCR`
0x14 突发控制寄存器

3.12.1 详细描述

I2C寄存器定义

在文件 [hal_lks32mc08x_reg.h](#) 第 101 行定义.

3.12.2 结构体成员变量说明

3.12.2.1 ADDR

```
volatile uint32_t ADDR
```

0x00 设备地址寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 103 行定义.

3.12.2.2 BCR

```
volatile uint32_t BCR
```

0x14 突发控制寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 108 行定义.

3.12.2.3 CFG

```
volatile uint32_t CFG
```

0x04 配置寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 104 行定义.

3.12.2.4 DATA

```
volatile uint32_t DATA
```

0x0C 数据寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 106 行定义.

3.12.2.5 MSCR

```
volatile uint32_t MSCR
```

0x10 主状态控制寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 107 行定义.

3.12.2.6 SCR

`volatile uint32_t SCR`

0x08 状态控制寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 105 行定义.

3.13 HAL_IWDG_TypeDef结构体 参考

```
#include <hal_lks32mc08x_reg.h>
```

成员变量

- `volatile uint32_t WDT_CLR`
0x00 看门狗清零寄存器

3.13.1 详细描述

在文件 [hal_lks32mc08x_reg.h](#) 第 344 行定义.

3.13.2 结构体成员变量说明

3.13.2.1 WDT_CLR

`volatile uint32_t WDT_CLR`

0x00 看门狗清零寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 346 行定义.

3.14 HAL_MCPWM_TypeDef结构体 参考

MCPWM寄存器定义

```
#include <hal_lks32mc08x_reg.h>
```

成员变量

- volatile uint32_t TH [8]
- volatile uint32_t TMR0
- volatile uint32_t TMR1
- volatile uint32_t TMR2
- volatile uint32_t TMR3
- volatile uint32_t TH0
- volatile uint32_t UPDATE
- volatile uint32_t IE0
- volatile uint32_t IF0
- volatile uint32_t EIE
- volatile uint32_t EIF
- volatile uint32_t RE
- volatile uint32_t PP
- volatile uint32_t IO01
- volatile uint32_t IO23
- volatile uint32_t SDCFG
- volatile uint32_t null1 [1]
- volatile uint32_t TCLK
- volatile uint32_t FAIL
- volatile uint32_t FILT
- volatile uint32_t null2 [2]
- volatile uint32_t PRT
- volatile uint32_t CNT
- volatile uint32_t null3 [1]
- volatile uint32_t DTH00
- volatile uint32_t DTH01
- volatile uint32_t DTH10
- volatile uint32_t DTH11
- volatile uint32_t DTH20
- volatile uint32_t DTH21
- volatile uint32_t DTH30
- volatile uint32_t DTH31

3.14.1 详细描述

MCPWM寄存器定义

在文件 [hal_lks32mc08x_reg.h](#) 第 199 行定义.

3.14.2 结构体成员变量说明

3.14.2.1 CNT

```
volatile uint32_t CNT
```

在文件 [hal_lks32mc08x_reg.h](#) 第 223 行定义.

3.14.2.2 DTH00

```
volatile uint32_t DTH00
```

在文件 [hal_lks32mc08x_reg.h](#) 第 225 行定义.

3.14.2.3 DTH01

```
volatile uint32_t DTH01
```

在文件 [hal_lks32mc08x_reg.h](#) 第 226 行定义.

3.14.2.4 DTH10

```
volatile uint32_t DTH10
```

在文件 [hal_lks32mc08x_reg.h](#) 第 227 行定义.

3.14.2.5 DTH11

```
volatile uint32_t DTH11
```

在文件 [hal_lks32mc08x_reg.h](#) 第 228 行定义.

3.14.2.6 DTH20

```
volatile uint32_t DTH20
```

在文件 [hal_lks32mc08x_reg.h](#) 第 229 行定义.

3.14.2.7 DTH21

```
volatile uint32_t DTH21
```

在文件 [hal_lks32mc08x_reg.h](#) 第 230 行定义.

3.14.2.8 DTH30

```
volatile uint32_t DTH30
```

在文件 [hal_lks32mc08x_reg.h](#) 第 231 行定义.

3.14.2.9 DTH31

```
volatile uint32_t DTH31
```

在文件 [hal_lks32mc08x_reg.h](#) 第 232 行定义.

3.14.2.10 EIE

```
volatile uint32_t EIE
```

在文件 [hal_lks32mc08x_reg.h](#) 第 210 行定义.

3.14.2.11 EIF

```
volatile uint32_t EIF
```

在文件 [hal_lks32mc08x_reg.h](#) 第 211 行定义.

3.14.2.12 FAIL

```
volatile uint32_t FAIL
```

在文件 [hal_lks32mc08x_reg.h](#) 第 219 行定义.

3.14.2.13 FILT

```
volatile uint32_t FILT
```

在文件 [hal_lks32mc08x_reg.h](#) 第 220 行定义.

3.14.2.14 IE0

```
volatile uint32_t IE0
```

在文件 [hal_lks32mc08x_reg.h](#) 第 208 行定义.

3.14.2.15 IF0

```
volatile uint32_t IF0
```

在文件 [hal_lks32mc08x_reg.h](#) 第 209 行定义.

3.14.2.16 IO01

```
volatile uint32_t IO01
```

在文件 [hal_lks32mc08x_reg.h](#) 第 214 行定义.

3.14.2.17 IO23

```
volatile uint32_t IO23
```

在文件 [hal_lks32mc08x_reg.h](#) 第 215 行定义.

3.14.2.18 null1

```
volatile uint32_t null1[1]
```

在文件 [hal_lks32mc08x_reg.h](#) 第 217 行定义.

3.14.2.19 null2

```
volatile uint32_t null2[2]
```

在文件 [hal_lks32mc08x_reg.h](#) 第 221 行定义.

3.14.2.20 null3

```
volatile uint32_t null3[1]
```

在文件 [hal_lks32mc08x_reg.h](#) 第 224 行定义.

3.14.2.21 PP

```
volatile uint32_t PP
```

在文件 [hal_lks32mc08x_reg.h](#) 第 213 行定义.

3.14.2.22 PRT

```
volatile uint32_t PRT
```

在文件 [hal_lks32mc08x_reg.h](#) 第 222 行定义.

3.14.2.23 RE

```
volatile uint32_t RE
```

在文件 [hal_lks32mc08x_reg.h](#) 第 212 行定义.

3.14.2.24 SDCFG

```
volatile uint32_t SDCFG
```

在文件 [hal_lks32mc08x_reg.h](#) 第 216 行定义.

3.14.2.25 TCLK

```
volatile uint32_t TCLK
```

在文件 [hal_lks32mc08x_reg.h](#) 第 218 行定义.

3.14.2.26 TH

```
volatile uint32_t TH[8]
```

在文件 [hal_lks32mc08x_reg.h](#) 第 201 行定义.

3.14.2.27 TH0

```
volatile uint32_t TH0
```

在文件 [hal_lks32mc08x_reg.h](#) 第 206 行定义.

3.14.2.28 TMR0

```
volatile uint32_t TMR0
```

在文件 [hal_lks32mc08x_reg.h](#) 第 202 行定义.

3.14.2.29 TMR1

```
volatile uint32_t TMR1
```

在文件 [hal_lks32mc08x_reg.h](#) 第 203 行定义.

3.14.2.30 TMR2

```
volatile uint32_t TMR2
```

在文件 [hal_lks32mc08x_reg.h](#) 第 204 行定义.

3.14.2.31 TMR3

```
volatile uint32_t TMR3
```

在文件 [hal_lks32mc08x_reg.h](#) 第 205 行定义.

3.14.2.32 UPDATE

```
volatile uint32_t UPDATE
```

在文件 [hal_lks32mc08x_reg.h](#) 第 207 行定义.

3.15 HAL_QEP_TypeDef结构体 参考

QEP寄存器定义

```
#include <hal_lks32mc08x_reg.h>
```

成员变量

- `volatile uint32_t CFG`
定时器配置寄存器
- `volatile uint32_t TH`
定时器计数门限寄存器
- `volatile uint32_t CNT`
定时器计数值寄存器

3.15.1 详细描述

QEP寄存器定义

在文件 `hal_lks32mc08x_reg.h` 第 189 行定义.

3.15.2 结构体成员变量说明

3.15.2.1 CFG

```
volatile uint32_t CFG
```

定时器配置寄存器

在文件 `hal_lks32mc08x_reg.h` 第 191 行定义.

3.15.2.2 CNT

```
volatile uint32_t CNT
```

定时器计数值寄存器

在文件 `hal_lks32mc08x_reg.h` 第 193 行定义.

3.15.2.3 TH

```
volatile uint32_t TH
```

定时器计数门限寄存器

在文件 `hal_lks32mc08x_reg.h` 第 192 行定义.

3.16 HAL_SPI_TypeDef结构体 参考

SPI寄存器定义

```
#include <hal_lks32mc08x_reg.h>
```

成员变量

- volatile uint32_t **CFG**
0x00 配置寄存器
- volatile uint32_t **IE**
0x04 中断寄存器
- volatile uint32_t **BAUD**
0x08 波特率寄存器
- volatile uint32_t **TXDATA**
0x0C 发送数据寄存器
- volatile uint32_t **RXDATA**
0x10 接收数据寄存器
- volatile uint32_t **SIZE**
0x14 传输数据长度寄存器

3.16.1 详细描述

SPI寄存器定义

在文件 [hal_lks32mc08x_reg.h](#) 第 88 行定义.

3.16.2 结构体成员变量说明

3.16.2.1 BAUD

```
volatile uint32_t BAUD
```

0x08 波特率寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 92 行定义.

3.16.2.2 CFG

```
volatile uint32_t CFG
```

0x00 配置寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 90 行定义.

3.16.2.3 IE

```
volatile uint32_t IE
```

0x04 中断寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 91 行定义.

3.16.2.4 RXDATA

```
volatile uint32_t RXDATA
```

0x10 接收数据寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 94 行定义.

3.16.2.5 SIZE

```
volatile uint32_t SIZE
```

0x14 传输数据长度寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 95 行定义.

3.16.2.6 TXDATA

```
volatile uint32_t TXDATA
```

0x0C 发送数据寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 93 行定义.

3.17 HAL_SYS_TypeDef结构体 参考

SYS寄存器定义

```
#include <hal_lks32mc08x_reg.h>
```

成员变量

- volatile uint32_t [WDT_PSW](#)
0x00 看门狗密码
- volatile uint32_t [WDT_CLR](#)
0x04 看门狗喂狗
- volatile uint32_t [WDT_TH](#)
0x08 看门狗计数器阈值
- volatile uint32_t [WDT_CNT](#)
0x0C 看门狗计数器值
- volatile uint32_t [RESERVED](#) [2]
- volatile uint32_t [AFE_CMP](#)
0x18 比较器输出测试
- volatile uint32_t [RESERVED1](#)
- volatile uint32_t [AFE_REG0](#)
0x20 模拟寄存器0
- volatile uint32_t [AFE_REG1](#)
0x24 模拟寄存器1

- volatile uint32_t [AFE_REG2](#)
0x28 模拟寄存器2
- volatile uint32_t [AFE_REG3](#)
0x2C 模拟寄存器3
- volatile uint32_t [AFE_REG4](#)
0x30 模拟寄存器4
- volatile uint32_t [AFE_REG5](#)
0x34 模拟寄存器5
- volatile uint32_t [AFE_REG6](#)
0x38 模拟寄存器6
- volatile uint32_t [AFE_REG7](#)
0x3C 模拟寄存器7
- volatile uint32_t [AFE_REG8](#)
0x40 模拟寄存器8
- volatile uint32_t [AFE_REG9](#)
0x44 模拟寄存器9
- volatile uint32_t [AFE_REGA](#)
0x48 模拟寄存器A
- volatile uint32_t [AFE_REGB](#)
0x4C 模拟寄存器B
- volatile uint32_t [AFE_REGC](#)
0x50 模拟寄存器C
- volatile uint32_t [TMP_A](#)
0x54 温度传感器幅度校准A
- volatile uint32_t [TMP_B](#)
0x58 温度传感器偏移校准B
- volatile uint32_t [RESERVED2](#)
- volatile uint32_t [AFE_DAC](#)
0x60 DAC数字输入
- volatile uint32_t [AFE_DAC_AMC](#)
0x64 DAC放大校准
- volatile uint32_t [AFE_DAC_DC](#)
0x68 DAC直流偏移
- volatile uint32_t [RESERVED3](#) [5]
- volatile uint32_t [CLK_CFG](#)
0x80 时钟配置
- volatile uint32_t [RST_CFG](#)
0x84 杂项配置
- volatile uint32_t [RST_SRC](#)
0x88 复位源记录
- volatile uint32_t [CLR_RST](#)
0x8C 复位源记录清除
- volatile uint32_t [CLK_DIV0](#)
0x90 I2C时钟分频比
- volatile uint32_t [RESERVED4](#)
- volatile uint32_t [CLK_DIV2](#)
0x98 UART时钟分频比
- volatile uint32_t [CLK_FEN](#)
0x9C 外设时钟使能
- volatile uint32_t [CLK_SLP](#)
0xA0 休眠模式进入

- volatile uint32_t **RESERVED5**
- volatile uint32_t **TRIM**
0xA8 系统修剪
- volatile uint32_t **SFT_RST**
0xAC 外设软复位
- volatile uint32_t **PROTECT**
0xB0 系统寄存器保护

3.17.1 详细描述

SYS寄存器定义

在文件 [hal_lks32mc08x_reg.h](#) 第 28 行定义.

3.17.2 结构体成员变量说明

3.17.2.1 AFE_CMP

```
volatile uint32_t AFE_CMP
```

0x18 比较器输出测试

在文件 [hal_lks32mc08x_reg.h](#) 第 35 行定义.

3.17.2.2 AFE_DAC

```
volatile uint32_t AFE_DAC
```

0x60 DAC数字输入

在文件 [hal_lks32mc08x_reg.h](#) 第 53 行定义.

3.17.2.3 AFE_DAC_AMC

```
volatile uint32_t AFE_DAC_AMC
```

0x64 DAC放大校准

在文件 [hal_lks32mc08x_reg.h](#) 第 54 行定义.

3.17.2.4 AFE_DAC_DC

```
volatile uint32_t AFE_DAC_DC
```

0x68 DAC直流偏移

在文件 [hal_lks32mc08x_reg.h](#) 第 55 行定义.

3.17.2.5 AFE_REG0

```
volatile uint32_t AFE_REG0
```

0x20 模拟寄存器0

在文件 [hal_lks32mc08x.reg.h](#) 第 37 行定义.

3.17.2.6 AFE_REG1

```
volatile uint32_t AFE_REG1
```

0x24 模拟寄存器1

在文件 [hal_lks32mc08x.reg.h](#) 第 38 行定义.

3.17.2.7 AFE_REG2

```
volatile uint32_t AFE_REG2
```

0x28 模拟寄存器2

在文件 [hal_lks32mc08x.reg.h](#) 第 39 行定义.

3.17.2.8 AFE_REG3

```
volatile uint32_t AFE_REG3
```

0x2C 模拟寄存器3

在文件 [hal_lks32mc08x.reg.h](#) 第 40 行定义.

3.17.2.9 AFE_REG4

```
volatile uint32_t AFE_REG4
```

0x30 模拟寄存器4

在文件 [hal_lks32mc08x.reg.h](#) 第 41 行定义.

3.17.2.10 AFE_REG5

```
volatile uint32_t AFE_REG5
```

0x34 模拟寄存器5

在文件 [hal_lks32mc08x.reg.h](#) 第 42 行定义.

3.17.2.11 AFE_REG6

```
volatile uint32_t AFE_REG6
```

0x38 模拟寄存器6

在文件 [hal_lks32mc08x.reg.h](#) 第 43 行定义.

3.17.2.12 AFE_REG7

```
volatile uint32_t AFE_REG7
```

0x3C 模拟寄存器7

在文件 [hal_lks32mc08x.reg.h](#) 第 44 行定义.

3.17.2.13 AFE_REG8

```
volatile uint32_t AFE_REG8
```

0x40 模拟寄存器8

在文件 [hal_lks32mc08x.reg.h](#) 第 45 行定义.

3.17.2.14 AFE_REG9

```
volatile uint32_t AFE_REG9
```

0x44 模拟寄存器9

在文件 [hal_lks32mc08x.reg.h](#) 第 46 行定义.

3.17.2.15 AFE_REGA

```
volatile uint32_t AFE_REGA
```

0x48 模拟寄存器A

在文件 [hal_lks32mc08x.reg.h](#) 第 47 行定义.

3.17.2.16 AFE_REGB

```
volatile uint32_t AFE_REGB
```

0x4C 模拟寄存器B

在文件 [hal_lks32mc08x.reg.h](#) 第 48 行定义.

3.17.2.17 AFE_REGC

```
volatile uint32_t AFE_REGC
```

0x50 模拟寄存器C

在文件 [hal_lks32mc08x.reg.h](#) 第 49 行定义.

3.17.2.18 CLK_CFG

```
volatile uint32_t CLK_CFG
```

0x80 时钟配置

在文件 [hal_lks32mc08x.reg.h](#) 第 57 行定义.

3.17.2.19 CLK_DIV0

```
volatile uint32_t CLK_DIV0
```

0x90 I2C时钟分频比

在文件 [hal_lks32mc08x.reg.h](#) 第 61 行定义.

3.17.2.20 CLK_DIV2

```
volatile uint32_t CLK_DIV2
```

0x98 UART时钟分频比

在文件 [hal_lks32mc08x.reg.h](#) 第 63 行定义.

3.17.2.21 CLK_FEN

```
volatile uint32_t CLK_FEN
```

0x9C 外设时钟使能

在文件 [hal_lks32mc08x.reg.h](#) 第 64 行定义.

3.17.2.22 CLK_SLP

```
volatile uint32_t CLK_SLP
```

0xA0 休眠模式进入

在文件 [hal_lks32mc08x.reg.h](#) 第 65 行定义.

3.17.2.23 CLR_RST

```
volatile uint32_t CLR_RST
```

0x8C 复位源记录清除

在文件 [hal_lks32mc08x_reg.h](#) 第 60 行定义.

3.17.2.24 PROTECT

```
volatile uint32_t PROTECT
```

0xB0 系统寄存器保护

在文件 [hal_lks32mc08x_reg.h](#) 第 69 行定义.

3.17.2.25 RESERVED

```
volatile uint32_t RESERVED[2]
```

在文件 [hal_lks32mc08x_reg.h](#) 第 34 行定义.

3.17.2.26 RESERVED1

```
volatile uint32_t RESERVED1
```

在文件 [hal_lks32mc08x_reg.h](#) 第 36 行定义.

3.17.2.27 RESERVED2

```
volatile uint32_t RESERVED2
```

在文件 [hal_lks32mc08x_reg.h](#) 第 52 行定义.

3.17.2.28 RESERVED3

```
volatile uint32_t RESERVED3[5]
```

在文件 [hal_lks32mc08x_reg.h](#) 第 56 行定义.

3.17.2.29 RESERVED4

```
volatile uint32_t RESERVED4
```

在文件 [hal_lks32mc08x_reg.h](#) 第 62 行定义.

3.17.2.30 RESERVED5

```
volatile uint32_t RESERVED5
```

在文件 [hal_lks32mc08x_reg.h](#) 第 66 行定义.

3.17.2.31 RST_CFG

```
volatile uint32_t RST_CFG
```

0x84 杂项配置

在文件 [hal_lks32mc08x_reg.h](#) 第 58 行定义.

3.17.2.32 RST_SRC

```
volatile uint32_t RST_SRC
```

0x88 复位源记录

在文件 [hal_lks32mc08x_reg.h](#) 第 59 行定义.

3.17.2.33 SFT_RST

```
volatile uint32_t SFT_RST
```

0xAC 外设软复位

在文件 [hal_lks32mc08x_reg.h](#) 第 68 行定义.

3.17.2.34 TMP_A

```
volatile uint32_t TMP_A
```

0x54 温度传感器幅度校准A

在文件 [hal_lks32mc08x_reg.h](#) 第 50 行定义.

3.17.2.35 TMP_B

```
volatile uint32_t TMP_B
```

0x58 温度传感器偏移校准B

在文件 [hal_lks32mc08x_reg.h](#) 第 51 行定义.

3.17.2.36 TRIM

```
volatile uint32_t TRIM
```

0xA8 系统修剪

在文件 [hal_lks32mc08x_reg.h](#) 第 67 行定义.

3.17.2.37 WDT_CLR

```
volatile uint32_t WDT_CLR
```

0x04 看门狗喂狗

在文件 [hal_lks32mc08x_reg.h](#) 第 31 行定义.

3.17.2.38 WDT_CNT

```
volatile uint32_t WDT_CNT
```

0x0C 看门狗计数器值

在文件 [hal_lks32mc08x_reg.h](#) 第 33 行定义.

3.17.2.39 WDT_PSW

```
volatile uint32_t WDT_PSW
```

0x00 看门狗密码

在文件 [hal_lks32mc08x_reg.h](#) 第 30 行定义.

3.17.2.40 WDT_TH

```
volatile uint32_t WDT_TH
```

0x08 看门狗计数器阈值

在文件 [hal_lks32mc08x_reg.h](#) 第 32 行定义.

3.18 HAL_TIMER_TypeDef结构体 参考

TIMER寄存器定义

```
#include <hal_lks32mc08x_reg.h>
```

成员变量

- volatile uint32_t [CFG](#)
定时器配置寄存器
- volatile uint32_t [TH](#)
定时器计数门限寄存器
- volatile uint32_t [CNT](#)
定时器计数值寄存器
- volatile uint32_t [CMPT0](#)
定时器比较/捕获寄存器 0
- volatile uint32_t [CMPT1](#)
定时器比较/捕获寄存器 1
- volatile uint32_t [EVT](#)
定时器外部事件选择寄存器

3.18.1 详细描述

TIMER寄存器定义

在文件 [hal_lks32mc08x_reg.h](#) 第 177 行定义.

3.18.2 结构体成员变量说明

3.18.2.1 CFG

```
volatile uint32_t CFG
```

定时器配置寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 179 行定义.

3.18.2.2 CMPT0

```
volatile uint32_t CMPT0
```

定时器比较/捕获寄存器 0

在文件 [hal_lks32mc08x_reg.h](#) 第 182 行定义.

3.18.2.3 CMPT1

```
volatile uint32_t CMPT1
```

定时器比较/捕获寄存器 1

在文件 [hal_lks32mc08x_reg.h](#) 第 183 行定义.

3.18.2.4 CNT

`volatile uint32_t CNT`

定时器计数值寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 181 行定义.

3.18.2.5 EVT

`volatile uint32_t EVT`

定时器外部事件选择寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 184 行定义.

3.18.2.6 TH

`volatile uint32_t TH`

定时器计数门限寄存器

在文件 [hal_lks32mc08x_reg.h](#) 第 180 行定义.

3.19 HAL_UART_TypeDef结构体 参考

UART寄存器定义

```
#include <hal_lks32mc08x_reg.h>
```

成员变量

- `volatile uint32_t CTRL`
0x00 UART控制
- `volatile uint32_t DIVH`
0x04 UART波特率高字节
- `volatile uint32_t DIVL`
0x08 UART波特率低字节
- `volatile uint32_t BUFF`
0x0C UART读/写缓冲区
- `volatile uint32_t ADR`
0x10 UART匹配地址(如RS485)
- `volatile uint32_t STT`
0x14 UART状态
- `volatile uint32_t IE`
0x18 UART中断使能
- `volatile uint32_t IF`
0x1C UART中断标志
- `volatile uint32_t IOC`
0x20 UART IO控制

3.19.1 详细描述

UART寄存器定义

在文件 [hal_lks32mc08x_reg.h](#) 第 268 行定义.

3.19.2 结构体成员变量说明

3.19.2.1 ADR

```
volatile uint32_t ADR
```

0x10 UART匹配地址(如RS485)

在文件 [hal_lks32mc08x_reg.h](#) 第 274 行定义.

3.19.2.2 BUFF

```
volatile uint32_t BUFF
```

0x0C UART读/写缓冲区

在文件 [hal_lks32mc08x_reg.h](#) 第 273 行定义.

3.19.2.3 CTRL

```
volatile uint32_t CTRL
```

0x00 UART控制

在文件 [hal_lks32mc08x_reg.h](#) 第 270 行定义.

3.19.2.4 DIVH

```
volatile uint32_t DIVH
```

0x04 UART波特率高字节

在文件 [hal_lks32mc08x_reg.h](#) 第 271 行定义.

3.19.2.5 DIVL

```
volatile uint32_t DIVL
```

0x08 UART波特率低字节

在文件 [hal_lks32mc08x_reg.h](#) 第 272 行定义.

3.19.2.6 IE

`volatile uint32_t IE`

0x18 UART中断使能

在文件 [hal_lks32mc08x_reg.h](#) 第 276 行定义.

3.19.2.7 IF

`volatile uint32_t IF`

0x1C UART中断标志

在文件 [hal_lks32mc08x_reg.h](#) 第 277 行定义.

3.19.2.8 IOC

`volatile uint32_t IOC`

0x20 UART IO控制

在文件 [hal_lks32mc08x_reg.h](#) 第 278 行定义.

3.19.2.9 STT

`volatile uint32_t STT`

0x14 UART状态

在文件 [hal_lks32mc08x_reg.h](#) 第 275 行定义.

Chapter 4

文件说明

4.1 hal_lks32mc08x.c 文件参考

```
#include "source/hal_lks32mc08x_adc.c"
#include "source/hal_lks32mc08x_clk.c"
#include "source/hal_lks32mc08x_cmp.c"
#include "source/hal_lks32mc08x_crc.c"
#include "source/hal_lks32mc08x_dac.c"
#include "source/hal_lks32mc08x_dma.c"
#include "source/hal_lks32mc08x_dsp.c"
#include "source/hal_lks32mc08x_flash.c"
#include "source/hal_lks32mc08x_gpio.c"
#include "source/hal_lks32mc08x_hall.c"
#include "source/hal_lks32mc08x_i2c.c"
#include "source/hal_lks32mc08x_irq.c"
#include "source/hal_lks32mc08x_init.c"
#include "source/hal_lks32mc08x_iwdg.c"
#include "source/hal_lks32mc08x_mcpwm.c"
#include "source/hal_lks32mc08x_opa.c"
#include "source/hal_lks32mc08x_qep.c"
#include "source/hal_lks32mc08x_reset.c"
#include "source/hal_lks32mc08x_reg.c"
#include "source/hal_lks32mc08x_spi.c"
#include "source/hal_lks32mc08x_timer.c"
#include "source/hal_lks32mc08x_tmp.c"
#include "source/hal_lks32mc08x_trim.c"
#include "source/hal_lks32mc08x_uart.c"
#include "source/hal_lks32mc08x_can.c"
#include "source/hal_lks32mc08x_wake.c"
#include "source/hal_lks32mc08x_irqfun.c"
```

hal_lks32mc08x.c 的引用(Include)关系图:



宏定义

- #define _HAL_LKS32MC08X_C_

4.1.1 宏定义说明

4.1.1.1 __HAL_LKS32MC08x_C__

```
#define __HAL_LKS32MC08x_C__
```

在文件 `hal_lks32mc08x.c` 第 2 行定义。

4.2 hal_lks32mc08x.c

[浏览该文件的文档。](#)

```
00001 #ifndef __HAL_LKS32MC08x_C__
00002 #define __HAL_LKS32MC08x_C__
00003 #include "source/hal_lks32mc08x_adc.c"
00004 #include "source/hal_lks32mc08x_clk.c"
00005 #include "source/hal_lks32mc08x_cmp.c"
00006 #include "source/hal_lks32mc08x_crc.c"
00007 #include "source/hal_lks32mc08x_dac.c"
00008 #include "source/hal_lks32mc08x_dma.c"
00009 #include "source/hal_lks32mc08x_dsp.c"
00010 #include "source/hal_lks32mc08x_flash.c"
00011 #include "source/hal_lks32mc08x_gpio.c"
00012 #include "source/hal_lks32mc08x_hall.c"
00013 #include "source/hal_lks32mc08x_i2c.c"
00014 #include "source/hal_lks32mc08x_irq.c"
00015 #include "source/hal_lks32mc08x_init.c"
00016 #include "source/hal_lks32mc08x_iwdg.c"
00017 #include "source/hal_lks32mc08x_mcpwm.c"
00018 #include "source/hal_lks32mc08x_opa.c"
00019 #include "source/hal_lks32mc08x_qep.c"
00020 #include "source/hal_lks32mc08x_reset.c"
00021 #include "source/hal_lks32mc08x_reg.c"
00022 #include "source/hal_lks32mc08x_spi.c"
00023 #include "source/hal_lks32mc08x_timer.c"
00024 #include "source/hal_lks32mc08x_tmp.c"
00025 #include "source/hal_lks32mc08x_trim.c"
00026 #include "source/hal_lks32mc08x_uart.c"
00027 #include "source/hal_lks32mc08x_can.c"
00028 #include "source/hal_lks32mc08x_wake.c"
00029 #include "source/hal_lks32mc08x_irqfun.c"
00030
00031 #endif
```

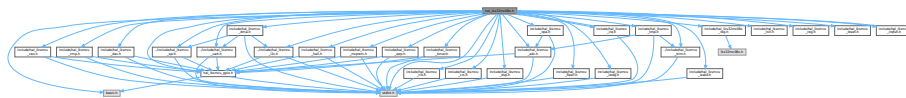
4.3 hal_lks32mc08x.h 文件参考

```
#include "basic.h"
#include "stdint.h"
#include "include/hal_lks32mc08x_reg.h"
#include "include/hal_lksmcu_adc.h"
#include "include/hal_lksmcu_can.h"
#include "include/hal_lksmcu_clk.h"
#include "include/hal_lksmcu_cmp.h"
#include "include/hal_lksmcu_crc.h"
#include "include/hal_lksmcu_dac.h"
#include "include/hal_lksmcu_dma.h"
#include "include/hal_lksmcu_dsp.h"
#include "include/hal_lksmcu_flash.h"
#include "include/hal_lksmcu_gpio.h"
#include "include/hal_lksmcu_hall.h"
#include "include/hal_lksmcu_i2c.h"
#include "include/hal_lksmcu_init.h"
#include "include/hal_lksmcu_iwdg.h"
```



```
#include "include/hal_lksmcu_irq.h"
#include "include/hal_lksmcu_mcpwm.h"
#include "include/hal_lksmcu_opa.h"
#include "include/hal_lksmcu_gep.h"
#include "include/hal_lksmcu_reg.h"
#include "include/hal_lksmcu_reset.h"
#include "include/hal_lksmcu_spi.h"
#include "include/hal_lksmcu_timer.h"
#include "include/hal_lksmcu_tmp.h"
#include "include/hal_lksmcu_trim.h"
#include "include/hal_lksmcu_uart.h"
#include "include/hal_lksmcu_wake.h"
#include "include/hal_lksmcu_irqfun.h"
```

hal_lks32mc08x.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



4.4 hal_lks32mc08x.h

[浏览该文件的文档.](#)

```
00001 #include "basic.h"
00002 #include "stdint.h"
00003 #include "include/hal_lks32mc08x_reg.h"
00004
00005 #ifndef __HAL_LKS32MC08X_H__
00006 #define __HAL_LKS32MC08X_H__
00007
00008 #include "include/hal_lksmcu_adc.h"
00009 #include "include/hal_lksmcu_can.h"
00010 #include "include/hal_lksmcu_clk.h"
00011 #include "include/hal_lksmcu_cmp.h"
00012 #include "include/hal_lksmcu_crc.h"
00013 #include "include/hal_lksmcu_dac.h"
00014 #include "include/hal_lksmcu_dma.h"
00015 #include "include/hal_lksmcu_dsp.h"
00016 #include "include/hal_lksmcu_flash.h"
00017 #include "include/hal_lksmcu_gpio.h"
00018 #include "include/hal_lksmcu_hall.h"
00019 #include "include/hal_lksmcu_i2c.h"
00020 #include "include/hal_lksmcu_init.h"
00021 #include "include/hal_lksmcu_iwdg.h"
00022 #include "include/hal_lksmcu_irq.h"
00023 #include "include/hal_lksmcu_mcpwm.h"
00024 #include "include/hal_lksmcu_opa.h"
00025 #include "include/hal_lksmcu_gep.h"
00026 #include "include/hal_lksmcu_reg.h"
00027 #include "include/hal_lksmcu_reset.h"
00028 #include "include/hal_lksmcu_spi.h"
00029 #include "include/hal_lksmcu_timer.h"
00030 #include "include/hal_lksmcu_tmp.h"
00031 #include "include/hal_lksmcu_trim.h"
00032 #include "include/hal_lksmcu_uart.h"
00033 #include "include/hal_lksmcu_wake.h"
00034 #include "include/hal_lksmcu_irqfun.h"
00035 #endif
```

4.5 hal_lksmcu.h 文件参考

```
#include "hal_lks32mc08x.h"
```

hal_lksmcu.h 的引用(Include)关系图:



4.6 hal_lksmcu.h

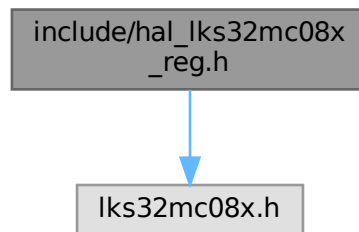
[浏览该文件的文档.](#)

```
00001 #include "hal_lks32mc08x.h"
```

4.7 include/hal_lks32mc08x_reg.h 文件参考

```
#include "lks32mc08x.h"
```

hal_lks32mc08x_reg.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



结构体

- struct [HAL_SYS_TypeDef](#)
SYS寄存器定义
- struct [HAL_FLASH_TypeDef](#)
FLASH寄存器定义

- struct [HAL_SPI_TypeDef](#)
SPI寄存器定义
- struct [HAL_I2C_TypeDef](#)
I2C寄存器定义
- struct [HAL_CMP_TypeDef](#)
CMP寄存器定义
- struct [HAL_HALL_TypeDef](#)
HALL寄存器定义
- struct [HAL_DMA_TypeDef](#)
DMA寄存器定义
- struct [HAL_ADC_TypeDef](#)
ADC0寄存器定义
- struct [HAL_TIMER_TypeDef](#)
TIMER寄存器定义
- struct [HAL_QEP_TypeDef](#)
QEP寄存器定义
- struct [HAL_MCPWM_TypeDef](#)
MCPWM寄存器定义
- struct [HAL_GPIO_TypeDef](#)
GPIO寄存器定义
- struct [HAL_CRC_TypeDef](#)
CRC寄存器定义
- struct [HAL_UART_TypeDef](#)
UART寄存器定义
- struct [HAL_CAN_TypeDef](#)
CAN寄存器定义
- struct [HAL_DSP_TypeDef](#)
DSP寄存器定义
- struct [HAL_IWDG_TypeDef](#)

宏定义

- #define [HAL_LKS32MC08X_REG](#)
- #define [HAL_MCPWM_STATE_SW_BIT](#) BIT0
- #define [HAL_MCPWM_STATE_H_MOS_EN_BIT](#) BIT1
- #define [HAL_MCPWM_STATE_L_MOS_EN_BIT](#) BIT2
- #define [HAL_MCPWM_STATE_H_DEFAULT_BIT](#) BIT3
- #define [HAL_MCPWM_STATE_L_DEFAULT_BIT](#) BIT4
- #define [HAL_MCPWM_STATE_STATE_MAX](#) ((uint32_t)0x7)
- #define [HAL_MCPWM_STATE_STATE_POS](#) 5
- #define [HAL_MCPWM_STATE_STATE_MASK](#) (uint32_t)((uint32_t)0x7 << [HAL_MCPWM_STATE_STATE_POS](#))
- #define [HAL_MCPWM_STATE_STATE\(x\)](#) (uint32_t)(((x) & 0x7) << [HAL_MCPWM_STATE_STATE_POS](#))
- #define [HAL_ADC_TRIG_STATE_ADC0_EN](#) BIT0
- #define [HAL_ADC_TRIG_STATE_ADC1_EN](#) BIT1
- #define [HAL_ADC_TRIG_STATE_MCPWM_TRIG_EN](#) BIT2
- #define [HAL_ADC_TRIG_STATE_TIMER_TRIG_EN](#) BIT3
- #define [HAL_ADC_TRIG_STATE_TWO_SEG_TRIG_EN](#) BIT4

变量

- [HAL_SYS_TypeDef](#) *const [HAL_SYS](#) []
- [HAL_FLASH_TypeDef](#) *const [HAL_FLASH](#) []
- [HAL_SPI_TypeDef](#) *const [HAL_SPI](#) []
- [HAL_I2C_TypeDef](#) *const [HAL_I2C](#) []
- [HAL_CMP_TypeDef](#) *const [HAL_CMP](#) []
- [HAL_HALL_TypeDef](#) *const [HAL_HALL](#) []
- [HAL_ADC_TypeDef](#) *const [HAL_ADC](#) []
- [HAL_TIMER_TypeDef](#) *const [HAL_TIMER](#) []
- [HAL_QEP_TypeDef](#) *const [HAL_QEP](#) []
- [HAL_MCPWM_TypeDef](#) *const [HAL_MCPWM](#) []
- [HAL_GPIO_TypeDef](#) *const [HAL_GPIO](#) []
- [HAL_CRC_TypeDef](#) *const [HAL_CRC](#) []
- [HAL_UART_TypeDef](#) *const [HAL_UART](#) []
- [HAL_CAN_TypeDef](#) *const [HAL_CAN](#) []
- [HAL_DSP_TypeDef](#) *const [HAL_DSP](#) []
- [HAL_IWDG_TypeDef](#) *const [HAL_IWDG](#) []
- [HAL_DMA_TypeDef](#) *const [HAL_DMA](#) []

4.7.1 宏定义说明

4.7.1.1 HAL_ADC_TRIG_STATE_ADC0_EN

```
#define HAL_ADC_TRIG_STATE_ADC0_EN BIT0
```

在文件 [hal_lks32mc08x_reg.h](#) 第 19 行定义.

4.7.1.2 HAL_ADC_TRIG_STATE_ADC1_EN

```
#define HAL_ADC_TRIG_STATE_ADC1_EN BIT1
```

在文件 [hal_lks32mc08x_reg.h](#) 第 20 行定义.

4.7.1.3 HAL_ADC_TRIG_STATE_MCPWM_TRIG_EN

```
#define HAL_ADC_TRIG_STATE_MCPWM_TRIG_EN BIT2
```

在文件 [hal_lks32mc08x_reg.h](#) 第 21 行定义.

4.7.1.4 HAL_ADC_TRIG_STATE_TIMER_TRIG_EN

```
#define HAL_ADC_TRIG_STATE_TIMER_TRIG_EN BIT3
```

在文件 [hal_lks32mc08x_reg.h](#) 第 22 行定义.

4.7.1.5 HAL_ADC_TRIG_STATE_TWO_SEG_TRIG_EN

```
#define HAL_ADC_TRIG_STATE_TWO_SEG_TRIG_EN BIT4
```

在文件 [hal_lks32mc08x_reg.h](#) 第 23 行定义.

4.7.1.6 HAL_LKS32MC08X_REG

```
#define HAL_LKS32MC08X_REG
```

在文件 [hal_lks32mc08x_reg.h](#) 第 4 行定义.

4.7.1.7 HAL_MCPWM_STATE_H_DEFAULT_BIT

```
#define HAL_MCPWM_STATE_H_DEFAULT_BIT BIT3
```

在文件 [hal_lks32mc08x_reg.h](#) 第 12 行定义.

4.7.1.8 HAL_MCPWM_STATE_H_MOS_EN_BIT

```
#define HAL_MCPWM_STATE_H_MOS_EN_BIT BIT1
```

在文件 [hal_lks32mc08x_reg.h](#) 第 10 行定义.

4.7.1.9 HAL_MCPWM_STATE_L_DEFAULT_BIT

```
#define HAL_MCPWM_STATE_L_DEFAULT_BIT BIT4
```

在文件 [hal_lks32mc08x_reg.h](#) 第 13 行定义.

4.7.1.10 HAL_MCPWM_STATE_L_MOS_EN_BIT

```
#define HAL_MCPWM_STATE_L_MOS_EN_BIT BIT2
```

在文件 [hal_lks32mc08x_reg.h](#) 第 11 行定义.

4.7.1.11 HAL_MCPWM_STATE_STATE

```
#define HAL_MCPWM_STATE_STATE(  
    x ) (uint32_t)((x) & 0x7) << HAL_MCPWM_STATE_STATE_POS)
```

在文件 [hal_lks32mc08x_reg.h](#) 第 17 行定义.

4.7.1.12 HAL_MCPWM_STATE_STATE_MASK

```
#define HAL_MCPWM_STATE_STATE_MASK (uint32_t) ((uint32_t) 0x7 << HAL_MCPWM_STATE_STATE_POS)
```

在文件 [hal_lks32mc08x_reg.h](#) 第 16 行定义.

4.7.1.13 HAL_MCPWM_STATE_STATE_MAX

```
#define HAL_MCPWM_STATE_STATE_MAX ((uint32_t) 0x7)
```

在文件 [hal_lks32mc08x_reg.h](#) 第 14 行定义.

4.7.1.14 HAL_MCPWM_STATE_STATE_POS

```
#define HAL_MCPWM_STATE_STATE_POS 5
```

在文件 [hal_lks32mc08x_reg.h](#) 第 15 行定义.

4.7.1.15 HAL_MCPWM_STATE_SW_BIT

```
#define HAL_MCPWM_STATE_SW_BIT BIT0
```

在文件 [hal_lks32mc08x_reg.h](#) 第 9 行定义.

4.7.2 变量说明

4.7.2.1 HAL_ADC

```
HAL_ADC_TypeDef* const HAL_ADC[] [extern]
```

在文件 [hal_lks32mc08x_reg.c](#) 第 14 行定义.

4.7.2.2 HAL_CAN

```
HAL_CAN_TypeDef* const HAL_CAN[] [extern]
```

在文件 [hal_lks32mc08x_reg.c](#) 第 21 行定义.

4.7.2.3 HAL_CMP

```
HAL_CMP_TypeDef* const HAL_CMP[] [extern]
```

在文件 [hal_lks32mc08x_reg.c](#) 第 12 行定义.

4.7.2.4 HAL_CRC

`HAL_CRC_TypeDef* const HAL_CRC[] [extern]`

在文件 `hal_lks32mc08x_reg.c` 第 19 行定义.

4.7.2.5 HAL_DMA

`HAL_DMA_TypeDef* const HAL_DMA[] [extern]`

在文件 `hal_lks32mc08x_reg.c` 第 24 行定义.

4.7.2.6 HAL_DSP

`HAL_DSP_TypeDef* const HAL_DSP[] [extern]`

在文件 `hal_lks32mc08x_reg.c` 第 22 行定义.

4.7.2.7 HAL_FLASH

`HAL_FLASH_TypeDef* const HAL_FLASH[] [extern]`

在文件 `hal_lks32mc08x_reg.c` 第 9 行定义.

4.7.2.8 HAL_GPIO

`HAL_GPIO_TypeDef* const HAL_GPIO[] [extern]`

在文件 `hal_lks32mc08x_reg.c` 第 18 行定义.

4.7.2.9 HAL_HALL

`HAL_HALL_TypeDef* const HAL_HALL[] [extern]`

在文件 `hal_lks32mc08x_reg.c` 第 13 行定义.

4.7.2.10 HAL_I2C

`HAL_I2C_TypeDef* const HAL_I2C[] [extern]`

在文件 `hal_lks32mc08x_reg.c` 第 11 行定义.

4.7.2.11 HAL_IWDG

`HAL_IWDG_TypeDef* const HAL_IWDG[] [extern]`

在文件 `hal_lks32mc08x_reg.c` 第 23 行定义.

4.7.2.12 HAL_MCPWM

```
HAL_MCPWM_TypeDef* const HAL_MCPWM[] [extern]
```

在文件 [hal_lks32mc08x_reg.c](#) 第 17 行定义.

4.7.2.13 HAL_QEP

```
HAL_QEP_TypeDef* const HAL_QEP[] [extern]
```

在文件 [hal_lks32mc08x_reg.c](#) 第 16 行定义.

4.7.2.14 HAL_SPI

```
HAL_SPI_TypeDef* const HAL_SPI[] [extern]
```

在文件 [hal_lks32mc08x_reg.c](#) 第 10 行定义.

4.7.2.15 HAL_SYS

```
HAL_SYS_TypeDef* const HAL_SYS[] [extern]
```

在文件 [hal_lks32mc08x_reg.c](#) 第 8 行定义.

4.7.2.16 HAL_TIMER

```
HAL_TIMER_TypeDef* const HAL_TIMER[] [extern]
```

在文件 [hal_lks32mc08x_reg.c](#) 第 15 行定义.

4.7.2.17 HAL_UART

```
HAL_UART_TypeDef* const HAL_UART[] [extern]
```

在文件 [hal_lks32mc08x_reg.c](#) 第 20 行定义.

4.8 hal_lks32mc08x_reg.h

[浏览该文件的文档.](#)

```

00001
00002 #ifndef __HAL_LKS32MC08X_REG_H__
00003 #define __HAL_LKS32MC08X_REG_H__
00004 #define HAL_LKS32MC08X_REG
00005 #ifndef HAL_TEST_REG
00006 #include "lks32mc08x.h"
00007 #endif
00008
00009 #define HAL_MCPWM_STATE_SW_BIT BIT0 // 通
    道交换使能 0 交换 1 交换
00010 #define HAL_MCPWM_STATE_H_MOS_EN_BIT BIT1 // 上管
    有效电平 0 上管低有效 1 上管高有效
00011 #define HAL_MCPWM_STATE_L_MOS_EN_BIT BIT2 // 下管
    有效电平 0 下管低有效 1 下管高有效
00012 #define HAL_MCPWM_STATE_H_DEFAULT_BIT BIT3 // 默认
    上管电平 0 关机后上管关闭 1 关机后上管打开
00013 #define HAL_MCPWM_STATE_L_DEFAULT_BIT BIT4 // 默认
    下管电平 0 关机后下管关闭 1 关机后下管打开
00014 #define HAL_MCPWM_STATE_STATE_MAX ((uint32_t)0x7) //
    HAL_MCPWM_STATE_STATE最大值
00015 #define HAL_MCPWM_STATE_STATE_POS 5 //
    HAL_MCPWM_STATE_STATE起始位
00016 #define HAL_MCPWM_STATE_STATE_MASK (uint32_t)((uint32_t)0x7 << HAL_MCPWM_STATE_STATE_POS) //
    HAL_MCPWM_STATE_STATE位掩码
00017 #define HAL_MCPWM_STATE_STATE(x) (uint32_t)((x) & 0x7) << HAL_MCPWM_STATE_STATE_POS // 默认
    输出状态 枚举
00018
00019 #define HAL_ADC_TRIG_STATE_ADC0_EN BIT0 // bit0 adc0使能
00020 #define HAL_ADC_TRIG_STATE_ADC1_EN BIT1 // bit1 adc1使能
00021 #define HAL_ADC_TRIG_STATE_MCPWM_TRIG_EN BIT2 // bit2 mcpwm触发
00022 #define HAL_ADC_TRIG_STATE_TIMER_TRIG_EN BIT3 // bit3 timer触发
00023 #define HAL_ADC_TRIG_STATE_TWO_SEG_TRIG_EN BIT4 // bit4 两段式触发使能
00024
00028 typedef struct
00029 {
00030     volatile uint32_t WDT_PSW;
00031     volatile uint32_t WDT_CLR;
00032     volatile uint32_t WDT_TH;
00033     volatile uint32_t WDT_CNT;
00034     volatile uint32_t RESERVED[2]; //
00035     volatile uint32_t AFE_CMP;
00036     volatile uint32_t RESERVED1; //
00037     volatile uint32_t AFE_REG0;
00038     volatile uint32_t AFE_REG1;
00039     volatile uint32_t AFE_REG2;
00040     volatile uint32_t AFE_REG3;
00041     volatile uint32_t AFE_REG4;
00042     volatile uint32_t AFE_REG5;
00043     volatile uint32_t AFE_REG6;
00044     volatile uint32_t AFE_REG7;
00045     volatile uint32_t AFE_REG8;
00046     volatile uint32_t AFE_REG9;
00047     volatile uint32_t AFE_REGA;
00048     volatile uint32_t AFE_REGB;
00049     volatile uint32_t AFE_REGC;
00050     volatile uint32_t TMP_A;
00051     volatile uint32_t TMP_B;
00052     volatile uint32_t RESERVED2; //
00053     volatile uint32_t AFE_DAC;
00054     volatile uint32_t AFE_DAC_AMC;
00055     volatile uint32_t AFE_DAC_DC;
00056     volatile uint32_t RESERVED3[5]; //
00057     volatile uint32_t CLK_CFG;
00058     volatile uint32_t RST_CFG;
00059     volatile uint32_t RST_SRC;
00060     volatile uint32_t CLR_RST;
00061     volatile uint32_t CLK_DIV0;
00062     volatile uint32_t RESERVED4; //
00063     volatile uint32_t CLK_DIV2;
00064     volatile uint32_t CLK_FEN;
00065     volatile uint32_t CLK_SLP;
00066     volatile uint32_t RESERVED5; //
00067     volatile uint32_t TRIM;
00068     volatile uint32_t SFT_RST;
00069     volatile uint32_t PROTECT;
00070 } HAL_SYS_TypeDef;
00071
00075 typedef struct
00076 {
00077     volatile uint32_t CFG;
00078     volatile uint32_t ADDR;
00079     volatile uint32_t WDATA;

```

```

00080     volatile uint32_t RDATA;
00081     volatile uint32_t ERASE;
00082     volatile uint32_t PROTECT;
00083     volatile uint32_t READY;
00084 } HAL_FLASH_TypeDef;
00088 typedef struct
00089 {
00090     volatile uint32_t CFG;
00091     volatile uint32_t IE;
00092     volatile uint32_t BAUD;
00093     volatile uint32_t TXDATA;
00094     volatile uint32_t RXDATA;
00095     volatile uint32_t SIZE;
00096 } HAL_SPI_TypeDef;
00097
00101 typedef struct
00102 {
00103     volatile uint32_t ADDR;
00104     volatile uint32_t CFG;
00105     volatile uint32_t SCR;
00106     volatile uint32_t DATA;
00107     volatile uint32_t MSCR;
00108     volatile uint32_t BCR;
00109 } HAL_I2C_TypeDef;
00110
00114 typedef struct
00115 {
00116     volatile uint32_t IE;
00117     volatile uint32_t IF;
00118     volatile uint32_t TCLK;
00119     volatile uint32_t CFG;
00120     volatile uint32_t BLCWIN;
00121     volatile uint32_t DATA;
00122 } HAL_CMP_TypeDef;
00123
00127 typedef struct
00128 {
00129     volatile uint32_t CFG;
00130     volatile uint32_t INFO;
00131     volatile uint32_t WIDTH;
00132     volatile uint32_t TH;
00133     volatile uint32_t CNT;
00134 } HAL_HALL_TypeDef;
00135
00139 typedef struct
00140 {
00141     volatile uint32_t CCR;
00142     volatile uint32_t CTMS;
00143     volatile uint32_t CPAR;
00144     volatile uint32_t CMAR;
00145 } HAL_DMA_TypeDef;
00149 typedef struct
00150 {
00151     volatile uint32_t DAT[20];
00152     volatile uint32_t CHN[10];
00153     volatile uint32_t CHNT0;
00154     volatile uint32_t CHNT1;
00155     volatile uint32_t null[2];
00156     volatile uint32_t GAIN_0;
00157     volatile uint32_t GAIN_1;
00158     volatile uint32_t DC_A0;
00159     volatile uint32_t DC_A1;
00160     volatile uint32_t AMC_A0;
00161     volatile uint32_t AMC_A1;
00162     volatile uint32_t DC_B0;
00163     volatile uint32_t DC_B1;
00164     volatile uint32_t AMC_B0;
00165     volatile uint32_t AMC_B1;
00166     volatile uint32_t IE;
00167     volatile uint32_t IF;
00168     volatile uint32_t CFG;
00169     volatile uint32_t TRIG;
00170     volatile uint32_t SWT;
00171     volatile uint32_t DAT0_TH;
00172 } HAL_ADC_TypeDef;
00173
00177 typedef struct
00178 {
00179     volatile uint32_t CFG;
00180     volatile uint32_t TH;
00181     volatile uint32_t CNT;
00182     volatile uint32_t CMPT0;
00183     volatile uint32_t CMPT1;
00184     volatile uint32_t EVT;
00185 } HAL_TIMER_TypeDef;
00189 typedef struct
00190 {

```

```

00191     volatile uint32_t CFG;
00192     volatile uint32_t TH;
00193     volatile uint32_t CNT;
00194 } HAL_QEP.TypeDef;
00195
00199 typedef struct
00200 {
00201     volatile uint32_t TH[8];      // 0x00
00202     volatile uint32_t TMR0;       // 0x20
00203     volatile uint32_t TMR1;       // 0x24
00204     volatile uint32_t TMR2;       // 0x28
00205     volatile uint32_t TMR3;       // 0x2C
00206     volatile uint32_t TH0;        // 0x30
00207     volatile uint32_t UPDATE;     // 0x34
00208     volatile uint32_t IE0;        // 0x38
00209     volatile uint32_t IF0;        // 0x3C
00210     volatile uint32_t EIE;        // 0x40
00211     volatile uint32_t EIF;        // 0x44
00212     volatile uint32_t RE;         // 0x48
00213     volatile uint32_t PP;         // 0x4C
00214     volatile uint32_t IO01;       // 0x50
00215     volatile uint32_t IO23;       // 0x54
00216     volatile uint32_t SDCFG;      // 0x58
00217     volatile uint32_t null1[1];    //
00218     volatile uint32_t TCLK;       // 0x60
00219     volatile uint32_t FAIL;       // 0x64
00220     volatile uint32_t FILT;       // 0x68
00221     volatile uint32_t null2[2];    //
00222     volatile uint32_t PRT;        // 0x74
00223     volatile uint32_t CNT;        // 0x78
00224     volatile uint32_t null3[1];    //
00225     volatile uint32_t DTH00;      // 0x80
00226     volatile uint32_t DTH01;      // 0x84
00227     volatile uint32_t DTH10;      // 0x88
00228     volatile uint32_t DTH11;      // 0x8C
00229     volatile uint32_t DTH20;      // 0x90
00230     volatile uint32_t DTH21;      // 0x94
00231     volatile uint32_t DTH30;      // 0x98
00232     volatile uint32_t DTH31;      // 0x9C
00233 } HAL_MCPWM.TypeDef;
00234
00238 typedef struct
00239 {
00240     volatile uint32_t PIE;
00241     volatile uint32_t POE;
00242     volatile uint32_t PDI;
00243     volatile uint32_t PDO;
00244     volatile uint32_t PUE;
00245     volatile uint32_t null1; // 地址对齐
00246     volatile uint32_t PODE;
00247     volatile uint32_t LCKR;
00248     volatile uint32_t F3210;
00249     volatile uint32_t F7654;
00250     volatile uint32_t FBA98;
00251     volatile uint32_t FFEDC;
00252 } HAL_GPIO.TypeDef;
00253
00257 typedef struct
00258 {
00259     volatile uint32_t DR;
00260     volatile uint32_t CR;
00261     volatile uint32_t INIT;
00262     volatile uint32_t POL;
00263 } HAL_CRC.TypeDef;
00264
00268 typedef struct
00269 {
00270     volatile uint32_t CTRL;
00271     volatile uint32_t DIVH;
00272     volatile uint32_t DIVL;
00273     volatile uint32_t BUFF;
00274     volatile uint32_t ADR;
00275     volatile uint32_t STT;
00276     volatile uint32_t IE;
00277     volatile uint32_t IF;
00278     volatile uint32_t IOC;
00279 } HAL_UART.TypeDef;
00280
00284 typedef struct
00285 {
00286     volatile uint32_t MOD;
00287     volatile uint32_t CMR;
00288     volatile uint32_t SR;
00289     volatile uint32_t IR;
00290
00291     volatile uint32_t IER;
00292     volatile uint32_t null0;

```

```

00293     volatile uint32_t BTR0;
00294     volatile uint32_t BTR1;
00295
00296     volatile uint32_t null1;
00297     volatile uint32_t null2;
00298     volatile uint32_t null3;
00299     volatile uint32_t ALC;
00300
00301     volatile uint32_t ECC;
00302     volatile uint32_t EWLR;
00303     volatile uint32_t RXERR;
00304     volatile uint32_t TXERR;
00305
00306     volatile uint32_t TXRX0;
00307     volatile uint32_t TXRX1;
00308     volatile uint32_t TXRX2;
00309     volatile uint32_t TXRX3;
00310
00311     volatile uint32_t TXRX4;
00312     volatile uint32_t TXRX5;
00313     volatile uint32_t TXRX6;
00314     volatile uint32_t TXRX7;
00315
00316     volatile uint32_t TXRX8;
00317     volatile uint32_t TXRX9;
00318     volatile uint32_t TXRXA;
00319     volatile uint32_t TXRXB;
00320
00321     volatile uint32_t TXRXC;
00322     volatile uint32_t RMC;
00323     volatile uint32_t RBSA;
00324 } HAL_CAN_TypeDef;
00325
00329 typedef struct
00330 {
00331     volatile uint32_t SC;
00332     volatile uint32_t THETA;
00333     volatile uint32_t XY;
00334     volatile uint32_t RESERVED;    //
00335     volatile uint32_t SIN;
00336     volatile uint32_t COS;
00337     volatile uint32_t MOD;
00338     volatile uint32_t ATAN;
00339     volatile uint32_t RESERVED1[4]; //
00340     volatile uint32_t RAD;
00341     volatile uint16_t SQRT;
00342 } HAL_DSP_TypeDef;
00343
00344 typedef struct
00345 {
00346     volatile uint32_t WDT_CLR;
00347 } HAL_IWDG_TypeDef;
00348
00349 extern HAL_SYS_TypeDef *const HAL_SYS[];
00350 extern HAL_FLASH_TypeDef *const HAL_FLASH[];
00351 extern HAL_SPI_TypeDef *const HAL_SPI[];
00352 extern HAL_I2C_TypeDef *const HAL_I2C[];
00353 extern HAL_CMP_TypeDef *const HAL_CMP[];
00354 extern HAL_HALL_TypeDef *const HAL_HALL[];
00355 extern HAL_ADC_TypeDef *const HAL_ADC[];
00356 extern HAL_TIMER_TypeDef *const HAL_TIMER[];
00357 extern HAL_QEP_TypeDef *const HAL_QEP[];
00358 extern HAL_MCPWM_TypeDef *const HAL_MCPWM[];
00359 extern HAL_GPIO_TypeDef *const HAL_GPIO[];
00360 extern HAL_CRC_TypeDef *const HAL_CRC[];
00361 extern HAL_UART_TypeDef *const HAL_UART[];
00362 extern HAL_CAN_TypeDef *const HAL_CAN[];
00363 extern HAL_DSP_TypeDef *const HAL_DSP[];
00364 extern HAL_IWDG_TypeDef *const HAL_IWDG[];
00365 extern HAL_DMA_TypeDef *const HAL_DMA[];
00366 #endif

```

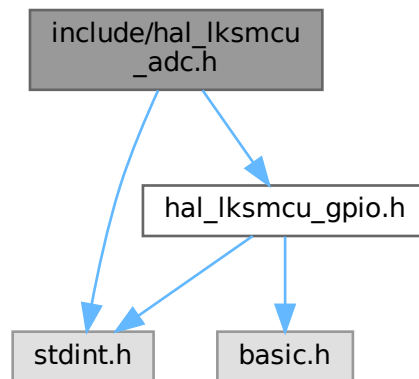
4.9 include/hal_lksmcu_adc.h 文件参考

```

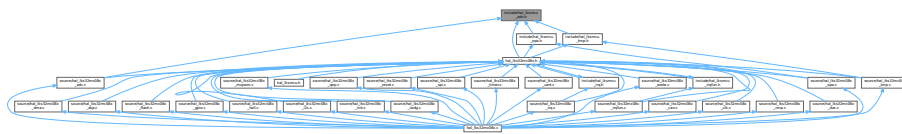
#include "stdint.h"
#include "hal_lksmcu_gpio.h"

```

hal_lksmcu_adc.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



枚举

- enum `HAL_ADC_CHANNEL_x` {
`HAL_ADC_CHANNEL_0` , `HAL_ADC_CHANNEL_1` , `HAL_ADC_CHANNEL_2` , `HAL_ADC_CHANNEL_3` ,
`HAL_ADC_CHANNEL_4` , `HAL_ADC_CHANNEL_5` , `HAL_ADC_CHANNEL_6` , `HAL_ADC_CHANNEL_7` ,
`HAL_ADC_CHANNEL_8` , `HAL_ADC_CHANNEL_9` , `HAL_ADC_CHANNEL_10` , `HAL_ADC_CHANNEL_11` ,
`HAL_ADC_CHANNEL_12` , `HAL_ADC_CHANNEL_13` , `HAL_ADC_CHANNEL_14` , `HAL_ADC_CHANNEL_15` ,
`HAL_ADC_CHANNEL_16` , `HAL_ADC_CHANNEL_17` , `HAL_ADC_CHANNEL_18` , `HAL_ADC_CHANNEL_19` ,
`HAL_ADC_CHANNEL_OPA0` = `HAL_ADC_CHANNEL_0` , `HAL_ADC_CHANNEL_OPA1` = `HAL_ADC_CHANNEL_1` , `HAL_ADC_CHANNEL_OPA2` = `HAL_ADC_CHANNEL_2` , `HAL_ADC_CHANNEL_OPA3` = `HAL_ADC_CHANNEL_3` ,
`HAL_ADC_CHANNEL_TMP0` = `HAL_ADC_CHANNEL_18` , `HAL_ADC_CHANNEL_AGND` = `HAL_ADC_CHANNEL_19` }
 主频
- enum `HAL_ADC_SAMP_SEQUENCE_x` {
`HAL_ADC_SAMP_SEQUENCE_0` , `HAL_ADC_SAMP_SEQUENCE_1` , `HAL_ADC_SAMP_SEQUENCE_2` ,
`HAL_ADC_SAMP_SEQUENCE_3` ,
`HAL_ADC_SAMP_SEQUENCE_4` , `HAL_ADC_SAMP_SEQUENCE_5` , `HAL_ADC_SAMP_SEQUENCE_6` ,
`HAL_ADC_SAMP_SEQUENCE_7` ,
`HAL_ADC_SAMP_SEQUENCE_8` , `HAL_ADC_SAMP_SEQUENCE_9` , `HAL_ADC_SAMP_SEQUENCE_10` ,
`HAL_ADC_SAMP_SEQUENCE_11` ,
`HAL_ADC_SAMP_SEQUENCE_12` , `HAL_ADC_SAMP_SEQUENCE_13` , `HAL_ADC_SAMP_SEQUENCE_14` ,
`HAL_ADC_SAMP_SEQUENCE_15` ,
`HAL_ADC_SAMP_SEQUENCE_16` , `HAL_ADC_SAMP_SEQUENCE_17` , `HAL_ADC_SAMP_SEQUENCE_18` ,
`HAL_ADC_SAMP_SEQUENCE_19` }

ADC采样序列

- enum `HAL_ADC_x` { `HAL_ADC_0` , `HAL_ADC_1` , `HAL_ADC_2` , `HAL_ADC_3` }
- enum `HAL_ADC_SAMP_NUMBER_x` {
`HAL_ADC_SAMP_NUMBER_0` , `HAL_ADC_SAMP_NUMBER_1` , `HAL_ADC_SAMP_NUMBER_2` , `HAL_ADC_SAMP_NUMBER_3`
, `HAL_ADC_SAMP_NUMBER_4` , `HAL_ADC_SAMP_NUMBER_5` , `HAL_ADC_SAMP_NUMBER_6` , `HAL_ADC_SAMP_NUMBER_7`
, `HAL_ADC_SAMP_NUMBER_8` , `HAL_ADC_SAMP_NUMBER_9` , `HAL_ADC_SAMP_NUMBER_10` , `HAL_ADC_SAMP_NUMBER_11`
, `HAL_ADC_SAMP_NUMBER_12` , `HAL_ADC_SAMP_NUMBER_13` , `HAL_ADC_SAMP_NUMBER_14` ,
`HAL_ADC_SAMP_NUMBER_15` ,
`HAL_ADC_SAMP_NUMBER_MAX` = `HAL_ADC_SAMP_NUMBER_15` }
- enum `HAL_ADC_TRIGGER_x` {
`HAL_ADC_TRIGGER_MCPWM0` , `HAL_ADC_TRIGGER_TIMER0` , `HAL_ADC_TRIGGER_SOFT` , `HAL_ADC_TRIGGER_MCPWM1`
= `HAL_ADC_TRIGGER_MCPWM0` ,
`HAL_ADC_TRIGGER_TIMER` = `HAL_ADC_TRIGGER_TIMER0` }
- enum `HAL_ADC_REFERENCE_x` { `HAL_ADC_REFERENCE_INTERNAL` , `HAL_ADC_REFERENCE_INTERNAL_3_6`
= `HAL_ADC_REFERENCE_INTERNAL` , `HAL_ADC_REFERENCE_INTERNAL_2_4` , `HAL_ADC_REFERENCE_EXTERNAL`
}

函数

- void `HAL_ADC_Reset` (`HAL_ADC_x`)
初始化指定的ADC模块
- void `HAL_ADC_Enable` (`HAL_ADC_x`)
启用指定的ADC模块
- void `HAL_ADC_Disable` (`HAL_ADC_x`)
禁用指定的ADC模块
- int16_t `HAL_ADC_GetSamplingValues` (`HAL_ADC_x`, `HAL_ADC_SAMP_SEQUENCE_x`)
获取指定序列号的采样值
- void `HAL_ADC_SetSamplingChannel` (`HAL_ADC_x`, `HAL_ADC_SAMP_SEQUENCE_x`, `HAL_ADC_CHANNEL_x`)
设置指定序列号的采样通道
- void `HAL_ADC_SetSamplingQuantity` (`HAL_ADC_x`, `HAL_ADC_SAMP_NUMBER_x`, `HAL_ADC_SAMP_NUMBER_x`)
设置采样数量
- void `HAL_ADC_SetTriggerSignal` (`HAL_ADC_x`, `HAL_ADC_TRIGGER_x`)
设置触发信号
- void `HAL_ADC_EndSampling` (`HAL_ADC_x`)
结束采样
- void `HAL_ADC_SetReference` (`HAL_ADC_x`, `HAL_ADC_REFERENCE_x`)
设置ADC基准源
- void `HAL_ADC_StateReset` (`HAL_ADC_x`)
重置ADC状态
- float `HAL_ADC_GetRange` (`HAL_ADC_x`)
获取adc量程,使用芯片供电作为adc基准时固定返回5.0
- float `HAL_ADC_GetValuesUnit` (`HAL_ADC_x`)
获取采样值单位
- void `HAL_ADC_SoftTrigger` (`HAL_ADC_x`)
软件触发ADC开始采样
- int16_t `HAL_ADC_ManualSampling` (`HAL_ADC_x`, `HAL_ADC_CHANNEL_x`)
手动采样
- void `HAL_ADC_SetGpioIn` (`HAL_ADC_x`, `HAL_GPIO_Px`)
设置指定的GPIO为ADC输入
- void `HAL_ADC_SetGpioTriggerOut` (`HAL_ADC_x`, `HAL_GPIO_Px`)
设置指定的GPIO为ADC触发信号输出

4.9.1 枚举类型说明

4.9.1.1 HAL_ADC_CHANNEL_x

enum [HAL_ADC_CHANNEL_x](#)

主频

枚举值

HAL_ADC_CHANNEL_0	ADC采样端口0
HAL_ADC_CHANNEL_1	ADC采样端口1
HAL_ADC_CHANNEL_2	ADC采样端口2
HAL_ADC_CHANNEL_3	ADC采样端口3
HAL_ADC_CHANNEL_4	ADC采样端口4
HAL_ADC_CHANNEL_5	ADC采样端口5
HAL_ADC_CHANNEL_6	ADC采样端口6
HAL_ADC_CHANNEL_7	ADC采样端口7
HAL_ADC_CHANNEL_8	ADC采样端口8
HAL_ADC_CHANNEL_9	ADC采样端口9
HAL_ADC_CHANNEL_10	ADC采样端口10
HAL_ADC_CHANNEL_11	ADC采样端口11
HAL_ADC_CHANNEL_12	ADC采样端口12
HAL_ADC_CHANNEL_13	ADC采样端口13
HAL_ADC_CHANNEL_14	ADC采样端口14
HAL_ADC_CHANNEL_15	ADC采样端口15
HAL_ADC_CHANNEL_16	ADC采样端口16
HAL_ADC_CHANNEL_17	ADC采样端口17
HAL_ADC_CHANNEL_18	ADC采样端口18
HAL_ADC_CHANNEL_19	ADC采样端口19
HAL_ADC_CHANNEL_OPA0	ADC采样端口运放0
HAL_ADC_CHANNEL_OPA1	ADC采样端口运放1
HAL_ADC_CHANNEL_OPA2	ADC采样端口运放2
HAL_ADC_CHANNEL_OPA3	ADC采样端口运放3
HAL_ADC_CHANNEL_TMP0	仅ADC0 ADC采样端口温度传感器
HAL_ADC_CHANNEL_AGN0	ADC采样内部地

在文件 [hal_lksmcu_adc.h](#) 第 6 行定义.

4.9.1.2 HAL_ADC_REFERENCE_x

enum [HAL_ADC_REFERENCE_x](#)

枚举值

HAL_ADC_REFERENCE_INTERNAL	使用内部基准源作为adc基准

枚举值

HAL_ADC_REFERENCE_INTERNAL_↔ 3_6	使用内部3.6V基准源作为adc基准
HAL_ADC_REFERENCE_INTERNAL_↔ 2_4	使用内部2.4V基准源作为adc基准
HAL_ADC_REFERENCE_EXTERNAL	使用芯片供电作为adc基准

在文件 [hal_lksmcu_adc.h](#) 第 116 行定义.

4.9.1.3 HAL_ADC_SAMP_NUMBER_x

enum [HAL_ADC_SAMP_NUMBER_x](#)

枚举值

HAL_ADC_SAMP_NUMBER_0	ADC采样0次
HAL_ADC_SAMP_NUMBER_1	ADC采样1次
HAL_ADC_SAMP_NUMBER_2	ADC采样2次
HAL_ADC_SAMP_NUMBER_3	ADC采样3次
HAL_ADC_SAMP_NUMBER_4	ADC采样4次
HAL_ADC_SAMP_NUMBER_5	ADC采样5次
HAL_ADC_SAMP_NUMBER_6	ADC采样6次
HAL_ADC_SAMP_NUMBER_7	ADC采样7次
HAL_ADC_SAMP_NUMBER_8	ADC采样8次
HAL_ADC_SAMP_NUMBER_9	ADC采样9次
HAL_ADC_SAMP_NUMBER_10	ADC采样10次
HAL_ADC_SAMP_NUMBER_11	ADC采样11次
HAL_ADC_SAMP_NUMBER_12	ADC采样12次
HAL_ADC_SAMP_NUMBER_13	ADC采样13次
HAL_ADC_SAMP_NUMBER_14	ADC采样14次
HAL_ADC_SAMP_NUMBER_15	ADC采样15次
HAL_ADC_SAMP_NUMBER_MAX	ADC采样15次

在文件 [hal_lksmcu_adc.h](#) 第 77 行定义.

4.9.1.4 HAL_ADC_SAMP_SEQUENCE_x

enum [HAL_ADC_SAMP_SEQUENCE_x](#)

ADC采样序列

枚举值

HAL_ADC_SAMP_SEQUENCE_0	ADC第0次采样
HAL_ADC_SAMP_SEQUENCE_1	ADC第1次采样

枚举值

HAL_ADC_SAMP_SEQUENCE_2	ADC第2次采样
HAL_ADC_SAMP_SEQUENCE_3	ADC第3次采样
HAL_ADC_SAMP_SEQUENCE_4	ADC第4次采样
HAL_ADC_SAMP_SEQUENCE_5	ADC第5次采样
HAL_ADC_SAMP_SEQUENCE_6	ADC第6次采样
HAL_ADC_SAMP_SEQUENCE_7	ADC第7次采样
HAL_ADC_SAMP_SEQUENCE_8	ADC第8次采样
HAL_ADC_SAMP_SEQUENCE_9	ADC第9次采样
HAL_ADC_SAMP_SEQUENCE_10	ADC第10次采样
HAL_ADC_SAMP_SEQUENCE_11	ADC第11次采样
HAL_ADC_SAMP_SEQUENCE_12	ADC第12次采样
HAL_ADC_SAMP_SEQUENCE_13	ADC第13次采样
HAL_ADC_SAMP_SEQUENCE_14	ADC第14次采样
HAL_ADC_SAMP_SEQUENCE_15	ADC第15次采样
HAL_ADC_SAMP_SEQUENCE_16	ADC第16次采样
HAL_ADC_SAMP_SEQUENCE_17	ADC第17次采样
HAL_ADC_SAMP_SEQUENCE_18	ADC第18次采样
HAL_ADC_SAMP_SEQUENCE_19	ADC第19次采样

在文件 [hal_lksmcu_adc.h](#) 第 47 行定义.

4.9.1.5 HAL_ADC_TRIGGER_x

enum [HAL_ADC_TRIGGER_x](#)

枚举值

HAL_ADC_TRIGGER_MCPWM0	使用MCPWM触发ADC采样
HAL_ADC_TRIGGER_TIMER0	使用TIMER触发ADC采样
HAL_ADC_TRIGGER_SOFT	使用软件触发ADC采样
HAL_ADC_TRIGGER_MCPWM	使用MCPWM触发ADC采样
HAL_ADC_TRIGGER_TIMER	使用TIMER触发ADC采样

在文件 [hal_lksmcu_adc.h](#) 第 97 行定义.

4.9.1.6 HAL_ADC_x

enum [HAL_ADC_x](#)

枚举值

HAL_↔ ADC_0	ADC模块0

枚举值

HAL↔ ADC_1	ADC模块1
HAL↔ ADC_2	ADC模块2
HAL↔ ADC_3	ADC模块3

在文件 [hal_lksmcu_adc.h](#) 第 70 行定义.

4.9.2 函数说明

4.9.2.1 HAL_ADC_Disable()

```
void HAL_ADC_Disable (
    HAL_ADC_x adc_x )
```

禁用指定的ADC模块

参数

adc↔ _x	要禁用的ADC模块编号

在文件 [hal_lks32mc08x_adc.c](#) 第 89 行定义.

4.9.2.2 HAL_ADC_Enable()

```
void HAL_ADC_Enable (
    HAL_ADC_x adc_x )
```

启用指定的ADC模块

参数

adc↔ _x	要启用的ADC模块编号

在文件 [hal_lks32mc08x_adc.c](#) 第 74 行定义.

4.9.2.3 HAL_ADC_EndSampling()

```
void HAL_ADC_EndSampling (
    HAL_ADC_x adc_x )
```

结束采样

参数

<code>adc_x</code>	ADC模块编号

在文件 [hal_lks32mc08x_adc.c](#) 第 244 行定义.

这是这个函数的调用关系图:



4.9.2.4 HAL_ADC_GetRange()

```
float HAL_ADC_GetRange (
    HAL_ADC_x adc_x )
```

获取adc量程,使用芯片供电作为adc基准时固定返回5.0

参数

<code>adc_x</code>	ADC模块编号

返回

ADC量程

在文件 [hal_lks32mc08x_adc.c](#) 第 350 行定义.

这是这个函数的调用关系图:



4.9.2.5 HAL_ADC_GetSamplingValues()

```
int16_t HAL_ADC_GetSamplingValues (
    HAL_ADC_x adc_x,
    HAL_ADC_SAMP_SEQUENCE_x sequence_x )
```

获取指定序列号的采样值

参数

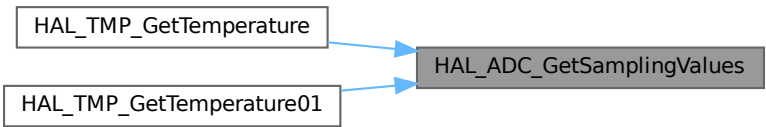
<i>adc_x</i>	ADC模块编号
<i>sequence_x</i>	采样序列号

返回

返回对应序列号的采样值

在文件 [hal_lks32mc08x_adc.c](#) 第 106 行定义.

这是这个函数的调用关系图:



4.9.2.6 HAL_ADC_GetValuesUnit()

```
float HAL_ADC_GetValuesUnit (
    HAL_ADC_x adc_x )
```

获取采样值单位

参数

<i>adc_x</i>	ADC模块编号

返回

采样值单位

在文件 [hal_lks32mc08x_adc.c](#) 第 381 行定义.

函数调用图:



4.9.2.7 HAL_ADC_ManualSampling()

```
int16_t HAL_ADC_ManualSampling (
    HAL_ADC_x adc_x,
    HAL_ADC_CHANNEL_x channel_x )
```

手动采样

参数

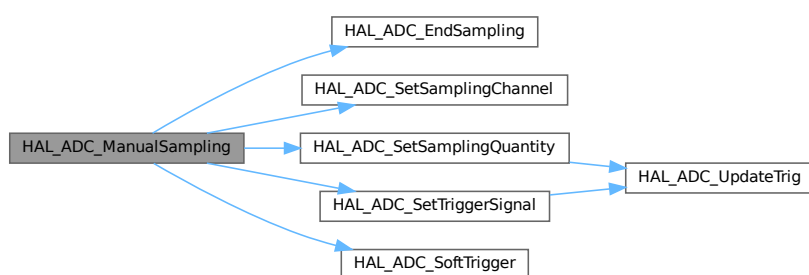
<i>n</i>	采样通道编号

返回

示例返回值,实际应返回采样值

在文件 [hal_lks32mc08x_adc.c](#) 第 263 行定义.

函数调用图:



4.9.2.8 HAL_ADC_Reset()

```
void HAL_ADC_Reset (
    HAL_ADC_x adc_x )
```

初始化指定的ADC模块

参数

<i>adc_x</i>	要初始化的ADC模块编号

在文件 [hal_lks32mc08x_adc.c](#) 第 65 行定义.

4.9.2.9 HAL_ADC_SetGpioIn()

```
void HAL_ADC_SetGpioIn (
    HAL_ADC_TypeDef* adc_x,
    GPIO_TypeDef* pin_x )
```

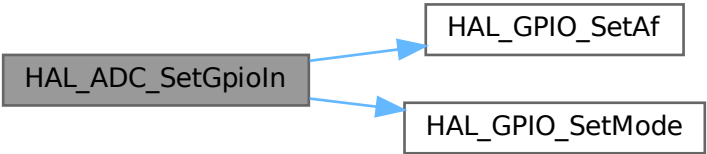
设置指定的GPIO为ADC输入

参数

<i>adc_x</i>	ADC模块编号
<i>pin_x</i>	GPIO编号

在文件 [hal_lks32mc08x_adc.c](#) 第 398 行定义.

函数调用图:



4.9.2.10 HAL_ADC_SetGpioTriggerOut()

```
void HAL_ADC_SetGpioTriggerOut (
    HAL_ADC_TypeDef* adc_x,
    GPIO_TypeDef* pin_x )
```

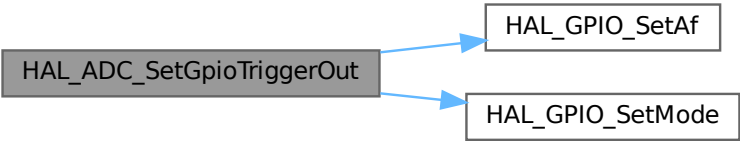
设置指定的GPIO为ADC触发信号输出

参数

<i>adc</i> ↔ _x	ADC模块编号
<i>pin</i> ↔ _x	GPIO编号

在文件 [hal_lks32mc08x_adc.c](#) 第 410 行定义.

函数调用图:



4.9.2.11 HAL_ADC.SetReference()

```
void HAL_ADC_SetReference (
    HAL_ADC_x adc_x,
    HAL_ADC_REFERENCE_x reference_x )
```

设置ADC基准源

参数

<i>adc_x</i>	ADC模块编号
<i>reference</i> ↔ _x	基准源

在文件 [hal_lks32mc08x_adc.c](#) 第 294 行定义.

函数调用图:



4.9.2.12 HAL_ADC_SetSamplingChannel()

```
void HAL_ADC_SetSamplingChannel (
    HAL_ADC_x adc_x,
    HAL_ADC_SAMP_SEQUENCE_x sequence_x,
    HAL_ADC_CHANNEL_x channel_x )
```

设置指定序列号的采样通道

参数

<i>adc_x</i>	ADC模块编号
<i>sequence_x</i>	采样序列号
<i>channel_x</i>	采样通道端口

在文件 [hal_lks32mc08x_adc.c](#) 第 117 行定义.

这是这个函数的调用关系图:



4.9.2.13 HAL_ADC_SetSamplingQuantity()

```
void HAL_ADC_SetSamplingQuantity (
    HAL_ADC_x adc_x,
    HAL_ADC_SAMP_NUMBER_x sample1_number,
    HAL_ADC_SAMP_NUMBER_x sample2_number )
```

设置采样数量

参数

<i>adc_x</i>	ADC模块编号
<i>sample_number</i>	采样数量

在文件 [hal_lks32mc08x_adc.c](#) 第 189 行定义.

函数调用图:



这是这个函数的调用关系图:



4.9.2.14 HAL_ADC_SetTriggerSignal()

```
void HAL_ADC_SetTriggerSignal (
    HAL_ADC_x adc-x,
    HAL_ADC_TRIGGER_x trigger )
```

设置触发信号

参数

<i>adc-x</i>	ADC模块编号
<i>trigger</i>	触发信号类型

- < 使用MCPWM触发ADC采样
- < 使用TIMER触发ADC采样
- < 使用软件触发ADC采样

在文件 [hal_lks32mc08x_adc.c](#) 第 221 行定义.

函数调用图:



这是这个函数的调用关系图:



4.9.2.15 HAL_ADC_SoftTrigger()

```
void HAL_ADC_SoftTrigger (
    HAL_ADC_x adc_x )
```

软件触发ADC开始采样

参数

adc_x	ADC模块编号

在文件 [hal_lks32mc08x_adc.c](#) 第 253 行定义.

这是这个函数的调用关系图:



4.9.2.16 HAL_ADC_StateReset()

```
void HAL_ADC_StateReset (
    HAL_ADC_x adc_x )
```

重置ADC状态

参数

adc_x	ADC模块编号

在文件 [hal_lks32mc08x_adc.c](#) 第 340 行定义.

4.10 hal_lksmcu_adc.h

[浏览该文件的文档.](#)

```

00001 #ifndef __HAL_LKSMCU_ADC_H__
00002 #define __HAL_LKSMCU_ADC_H__
00003 #include "stdint.h"
00004 #include "hal_lksmcu_gpio.h"
00006 typedef enum
00007 {
00008     HAL_ADC_CHANNEL_0,
00009     HAL_ADC_CHANNEL_1,
00010     HAL_ADC_CHANNEL_2,
00011     HAL_ADC_CHANNEL_3,
00012     HAL_ADC_CHANNEL_4,
00013     HAL_ADC_CHANNEL_5,
00014     HAL_ADC_CHANNEL_6,
00015     HAL_ADC_CHANNEL_7,
00016     HAL_ADC_CHANNEL_8,
00017     HAL_ADC_CHANNEL_9,
00018     HAL_ADC_CHANNEL_10,
00019     HAL_ADC_CHANNEL_11,
00020     HAL_ADC_CHANNEL_12,
00021     HAL_ADC_CHANNEL_13,
00022     HAL_ADC_CHANNEL_14,
00023     HAL_ADC_CHANNEL_15,
00024     HAL_ADC_CHANNEL_16,
00025     HAL_ADC_CHANNEL_17,
00026     HAL_ADC_CHANNEL_18,
00027     HAL_ADC_CHANNEL_19,
00028
00029     HAL_ADC_CHANNEL_OPA0 = HAL_ADC_CHANNEL_0,
00030     HAL_ADC_CHANNEL_OPA1 = HAL_ADC_CHANNEL_1,
00031     HAL_ADC_CHANNEL_OPA2 = HAL_ADC_CHANNEL_2,
00032     HAL_ADC_CHANNEL_OPA3 = HAL_ADC_CHANNEL_3,
00033     HAL_ADC_CHANNEL_TMP0 = HAL_ADC_CHANNEL_18,
00034     HAL_ADC_CHANNEL_AGN0 = HAL_ADC_CHANNEL_19,
00035
00036 #ifdef HAL_TEST_NOT_SUPPORTED
00037     HAL_ADC_CHANNEL_VDD,
00038     HAL_ADC_CHANNEL_DAC0,
00039     HAL_ADC_CHANNEL_DAC1,
00040     HAL_ADC_CHANNEL_REF,
00041     HAL_ADC_CHANNEL_OPA4,
00042     HAL_ADC_CHANNEL_OPA5,
00043     HAL_ADC_CHANNEL_LDO12,
00044 #endif
00045 } HAL_ADC_CHANNEL_x;
00047 typedef enum
00048 {
00049     HAL_ADC_SAMP_SEQUENCE_0,
00050     HAL_ADC_SAMP_SEQUENCE_1,
00051     HAL_ADC_SAMP_SEQUENCE_2,
00052     HAL_ADC_SAMP_SEQUENCE_3,
00053     HAL_ADC_SAMP_SEQUENCE_4,
00054     HAL_ADC_SAMP_SEQUENCE_5,
00055     HAL_ADC_SAMP_SEQUENCE_6,
00056     HAL_ADC_SAMP_SEQUENCE_7,
00057     HAL_ADC_SAMP_SEQUENCE_8,
00058     HAL_ADC_SAMP_SEQUENCE_9,
00059     HAL_ADC_SAMP_SEQUENCE_10,
00060     HAL_ADC_SAMP_SEQUENCE_11,
00061     HAL_ADC_SAMP_SEQUENCE_12,
00062     HAL_ADC_SAMP_SEQUENCE_13,
00063     HAL_ADC_SAMP_SEQUENCE_14,
00064     HAL_ADC_SAMP_SEQUENCE_15,
00065     HAL_ADC_SAMP_SEQUENCE_16,
00066     HAL_ADC_SAMP_SEQUENCE_17,
00067     HAL_ADC_SAMP_SEQUENCE_18,
00068     HAL_ADC_SAMP_SEQUENCE_19,
00069 } HAL_ADC_SAMP_SEQUENCE_x;
00070 typedef enum
00071 {
00072     HAL_ADC_0,
00073     HAL_ADC_1,
00074     HAL_ADC_2,
00075     HAL_ADC_3,
00076 } HAL_ADC_x;
00077 typedef enum
00078 {
00079     HAL_ADC_SAMP_NUMBER_0,
00080     HAL_ADC_SAMP_NUMBER_1,
00081     HAL_ADC_SAMP_NUMBER_2,
00082     HAL_ADC_SAMP_NUMBER_3,
00083     HAL_ADC_SAMP_NUMBER_4,
00084     HAL_ADC_SAMP_NUMBER_5,

```

```

00085     HAL_ADC_SAMP_NUMBER_6,
00086     HAL_ADC_SAMP_NUMBER_7,
00087     HAL_ADC_SAMP_NUMBER_8,
00088     HAL_ADC_SAMP_NUMBER_9,
00089     HAL_ADC_SAMP_NUMBER_10,
00090     HAL_ADC_SAMP_NUMBER_11,
00091     HAL_ADC_SAMP_NUMBER_12,
00092     HAL_ADC_SAMP_NUMBER_13,
00093     HAL_ADC_SAMP_NUMBER_14,
00094     HAL_ADC_SAMP_NUMBER_15,
00095     HAL_ADC_SAMP_NUMBER_MAX = HAL_ADC_SAMP_NUMBER_15,
00096 } HAL_ADC_SAMP_NUMBER_x;
00097 typedef enum
00098 {
00099     HAL_ADC_TRIGGER_MCPWM0,
00100     HAL_ADC_TRIGGER_TIMER0,
00101     HAL_ADC_TRIGGER_SOFT,
00102
00103     HAL_ADC_TRIGGER_MCPWM = HAL_ADC_TRIGGER_MCPWM0,
00104     HAL_ADC_TRIGGER_TIMER = HAL_ADC_TRIGGER_TIMER0,
00105
00106 #ifdef HAL_TEST_NOT_SUPPORTED
00107     HAL_ADC_TRIGGER_MCPWM0_1,
00108     HAL_ADC_TRIGGER_MCPWM1_1,
00109     HAL_ADC_TRIGGER_MCPWM1,
00110     HAL_ADC_TRIGGER_TIMER1,
00111     HAL_ADC_TRIGGER_TIMER2,
00112     HAL_ADC_TRIGGER_TIMER3,
00113     HAL_ADC_TRIGGER_TIMER4,
00114 #endif
00115 } HAL_ADC_TRIGGER_x;
00116 typedef enum
00117 {
00118     HAL_ADC_REFERENCE_INTERNAL,
00119     HAL_ADC_REFERENCE_INTERNAL_3_6 = HAL_ADC_REFERENCE_INTERNAL,
00120     HAL_ADC_REFERENCE_INTERNAL_2_4,
00121     HAL_ADC_REFERENCE_EXTERNAL,
00122 #ifdef HAL_TEST_NOT_SUPPORTED
00123     HAL_ADC_REFERENCE_INTERNAL_3_3
00124 #endif
00125 } HAL_ADC_REFERENCE_x;
00126
00127 // 初始化
00128 void HAL_ADC_Reset (HAL_ADC_x); // 模块复位
00129 void HAL_ADC_Enable (HAL_ADC_x); // 模块打开
00130 void HAL_ADC_Disable (HAL_ADC_x); // 模块关闭
00131
00132 // 自动采样配置
00133 int16_t HAL_ADC_GetSamplingValues (HAL_ADC_x, HAL_ADC_SAMP_SEQUENCE_x); // 获取采样值 返
    回第n次采样结果
00134 void HAL_ADC_SetSamplingChannel (HAL_ADC_x, HAL_ADC_SAMP_SEQUENCE_x, HAL_ADC_CHANNEL_x); // 设置采样通道, 输
    入参数 第n次采样channel通道
00135 void HAL_ADC_SetSamplingQuantity (HAL_ADC_x, HAL_ADC_SAMP_NUMBER_x, HAL_ADC_SAMP_NUMBER_x); // 设置采样数量
00136 void HAL_ADC_SetTriggerSignal (HAL_ADC_x, HAL_ADC_TRIGGER_x); // 设置触发信号
00137 void HAL_ADC_EndSampling (HAL_ADC_x); // 中止当前采
    样
00138 void HAL_ADC_SetReference (HAL_ADC_x, HAL_ADC_REFERENCE_x); // 设置ADC基准
    源
00139 void HAL_ADC_StateReset (HAL_ADC_x); // ADC状态机复
    位
00140 float HAL_ADC_GetRange (HAL_ADC_x); // 获取adc量
    程, 使用芯片供电作为adc基准时固定返回5.0
00141 float HAL_ADC_GetValuesUnit (HAL_ADC_x); // 获取采样值
    单位
00142 // 手动采样配置
00143 void HAL_ADC_SoftTrigger (HAL_ADC_x); // 软件触发ADC开始采样
00144 int16_t HAL_ADC_ManualSampling (HAL_ADC_x, HAL_ADC_CHANNEL_x); // 手动采样 需要先设置采样通道为软件触发
00145
00146 // IO配置
00147 void HAL_ADC_SetGpioIn (HAL_ADC_x, HAL_GPIO_Px); // 设置指定的GPIO为ADC输入
00148 void HAL_ADC_SetGpioTriggerOut (HAL_ADC_x, HAL_GPIO_Px); // 设置指定的GPIO为ADC触发信号输出
00149
00150 #endif

```

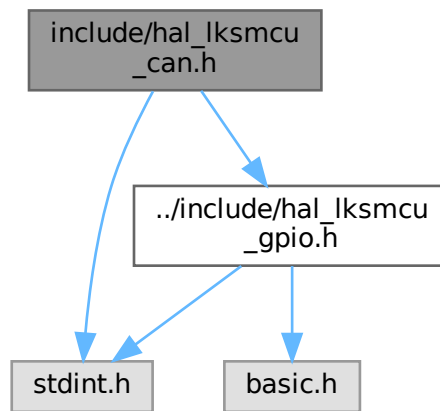
4.11 include/hal_lksmcu_can.h 文件参考

```

#include <stdint.h>
#include "../include/hal_lksmcu_gpio.h"

```

hal_lksmcu_can.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



枚举

- enum `HAL_CAN_x` { `HAL_CAN_0` }
- enum `HAL_CAN_FRAME_x` {
`HAL_CAN_FRAME_IDE0_RTR0` , `HAL_CAN_FRAME_IDE0_RTR1` , `HAL_CAN_FRAME_IDE1_RTR0` ,
`HAL_CAN_FRAME_IDE1_RTR1` ,
`HAL_CAN_FRAME_IDE0_RTR0_FD1_BRS0` , `HAL_CAN_FRAME_IDE0_RTR0_FD1_BRS1` , `HAL_CAN_FRAME_IDE1_RTR0_FD1`
, `HAL_CAN_FRAME_IDE1_RTR0_FD1_BRS1` }
- enum `HAL_CAN_ERROR_x` {
`HAL_CAN_ERROR_None` = 0 , `HAL_CAN_ERROR.Bit` = 1 , `HAL_CAN_ERROR.Form` = 2 , `HAL_CAN_ERROR.Stuff`
= 3 ,
`HAL_CAN_ERROR.Ack` = 4 , `HAL_CAN_ERROR.Crc` = 5 , `HAL_CAN_ERROR.Unknown` = 6 }
- enum `HAL_CAN_TSSTAT_x` { `HAL_CAN_TSSTAT_Empty` = 0 , `HAL_CAN_TSSTAT_LessHalf` = 1 ,
`HAL_CAN_TSSTAT_Half` = 2 , `HAL_CAN_TSSTAT_Full` = 3 }
- enum `HAL_CAN_AE_x` {
`HAL_CAN_AE_0` , `HAL_CAN_AE_1` , `HAL_CAN_AE_2` , `HAL_CAN_AE_3` ,
`HAL_CAN_AE_4` , `HAL_CAN_AE_5` , `HAL_CAN_AE_6` , `HAL_CAN_AE_7` ,
`HAL_CAN_AE_8` , `HAL_CAN_AE_9` , `HAL_CAN_AE_10` , `HAL_CAN_AE_11` ,
`HAL_CAN_AE_12` , `HAL_CAN_AE_13` , `HAL_CAN_AE_14` , `HAL_CAN_AE_15` }
- enum `HAL_CAN_STBTrig_x` { `HAL_CAN_STBTrig_One` , `HAL_CAN_STBTrig_All` }
- enum `HAL_CAN_STBMode_x` { `HAL_CAN_STBMode_FIFO` , `HAL_CAN_STBMode_Priority` }
- enum `HAL_CAN_IDEFilter_x` { `HAL_CAN_IDEFilter_All` , `HAL_CAN_IDEFilter_St` , `HAL_CAN_IDEFilter_Ex` }

函数

- void [HAL_CAN_Reset](#) ([HAL_CAN_x](#) can_x)
- void [HAL_CAN_Enable](#) ([HAL_CAN_x](#) can_x)
- void [HAL_CAN_Disable](#) ([HAL_CAN_x](#) can_x)
- void [HAL_CAN_SetBaud](#) ([HAL_CAN_x](#) can_x, uint32_t baud)
- void [HAL_CAN_SetClockDiv](#) ([HAL_CAN_x](#) can_x, uint32_t div)
- uint32_t [HAL_CAN_GetClockDiv](#) ([HAL_CAN_x](#) can_x)
- uint32_t [HAL_CAN_GetClockFreq](#) ([HAL_CAN_x](#) can_x)
- uint32_t [HAL_CAN_GetBaud](#) ([HAL_CAN_x](#) can_x)
- void [HAL_CAN_SetFastBaud](#) ([HAL_CAN_x](#) can_x, uint32_t fbaud)
- void [HAL_CAN_SetTDC](#) ([HAL_CAN_x](#) can_x, uint8_t sspoff)
- bool [HAL_CAN_TxData](#) ([HAL_CAN_x](#) can_x, uint32_t id, [HAL_CAN_FRAME_x](#) frame_x, uint8_t *buf, uint32_t len)
- bool [HAL_CAN_TxData_STBWrite](#) ([HAL_CAN_x](#) can_x, uint32_t id, [HAL_CAN_FRAME_x](#) frame_x, uint8_t *buf, uint32_t len)
- bool [HAL_CAN_TxData_STBTrig](#) ([HAL_CAN_x](#) can_x, [HAL_CAN_STBTrig_x](#) trig)
- void [HAL_CAN_TxCancel](#) ([HAL_CAN_x](#) can_x)
- bool [HAL_CAN_Txing](#) ([HAL_CAN_x](#) can_x)
- bool [HAL_CAN_STBFull](#) ([HAL_CAN_x](#) can_x)
- [HAL_CAN_TSSTAT_x](#) [HAL_CAN_TSSStat](#) ([HAL_CAN_x](#) can_x)
- bool [HAL_CAN_RxData](#) ([HAL_CAN_x](#) can_x)
- uint32_t [HAL_CAN_RxDataID](#) ([HAL_CAN_x](#) can_x)
- uint8_t [HAL_CAN_RxDataBuf](#) ([HAL_CAN_x](#) can_x, uint8_t *buf)
- uint8_t [HAL_CAN_RxDataLen](#) ([HAL_CAN_x](#) can_x)
- uint8_t [HAL_CAN_RxDataIde](#) ([HAL_CAN_x](#) can_x)
- uint8_t [HAL_CAN_RxDataRtr](#) ([HAL_CAN_x](#) can_x)
- uint8_t [HAL_CAN_RxDataFdf](#) ([HAL_CAN_x](#) can_x)
- void [HAL_CAN_IDx_Filter](#) ([HAL_CAN_x](#) can_x, u8 x, u32 id, u32 mask, [HAL_CAN_IDEFilter_x](#) ide_filter)
- CAN接收尝试刷疆
- void [HAL_CAN_IDx_FilterEnable](#) ([HAL_CAN_x](#) can_x, [HAL_CAN_AE_x](#) ae_x)
- void [HAL_CAN_IDx_FilterDisable](#) ([HAL_CAN_x](#) can_x, [HAL_CAN_AE_x](#) ae_x)
- void [HAL_CAN_SetSTBMode](#) ([HAL_CAN_x](#) can_x, [HAL_CAN_STBMode_x](#) mode_x)
- void [HAL_CAN_LOM_Enable](#) ([HAL_CAN_x](#) can_x)
- 监听模式使能/禁止
- void [HAL_CAN_LOM_Disable](#) ([HAL_CAN_x](#) can_x)
- void [HAL_CAN_LBME_Enable](#) ([HAL_CAN_x](#) can_x)
- 内部回环模式使能/禁止
- void [HAL_CAN_LBME_Disable](#) ([HAL_CAN_x](#) can_x)
- void [HAL_CAN_LBME_Enable](#) ([HAL_CAN_x](#) can_x)
- 外部回环模式使能/禁止
- void [HAL_CAN_LBME_Disable](#) ([HAL_CAN_x](#) can_x)
- void [HAL_CAN_SS_Enable](#) ([HAL_CAN_x](#) can_x)
- 单次传输模式使能/禁止
- void [HAL_CAN_SS_Disable](#) ([HAL_CAN_x](#) can_x)
- uint8_t [HAL_CAN_RxErrorCnt](#) ([HAL_CAN_x](#) can_x)
- uint8_t [HAL_CAN_TxErrorCnt](#) ([HAL_CAN_x](#) can_x)
- uint8_t [HAL_CAN_ErrorPassive](#) ([HAL_CAN_x](#) can_x)
- uint8_t [HAL_CAN_Busoff](#) ([HAL_CAN_x](#) can_x)
- uint8_t [HAL_CAN_ArbitrationLostCap](#) ([HAL_CAN_x](#) can_x)
- uint8_t [HAL_CAN_ROverflow](#) ([HAL_CAN_x](#) can_x)
- [HAL_CAN_ERROR_x](#) [HAL_CAN_KindOfError](#) ([HAL_CAN_x](#) can_x)
- void [HAL_CAN_SetGpioTx](#) ([HAL_CAN_x](#) can_x, [HAL_GPIO_Px](#) gpio_x)
- void [HAL_CAN_SetGpioRx](#) ([HAL_CAN_x](#) can_x, [HAL_GPIO_Px](#) gpio_x)

4.11.1 枚举类型说明

4.11.1.1 HAL_CAN_AE_x

enum [HAL_CAN_AE_x](#)

枚举值

HAL_CAN_AE_0	
HAL_CAN_AE_1	
HAL_CAN_AE_2	
HAL_CAN_AE_3	
HAL_CAN_AE_4	
HAL_CAN_AE_5	
HAL_CAN_AE_6	
HAL_CAN_AE_7	
HAL_CAN_AE_8	
HAL_CAN_AE_9	
HAL_CAN_AE_10	
HAL_CAN_AE_11	
HAL_CAN_AE_12	
HAL_CAN_AE_13	
HAL_CAN_AE_14	
HAL_CAN_AE_15	

在文件 [hal_lksmcu_can.h](#) 第 49 行定义.

4.11.1.2 HAL_CAN_ERROR_x

enum [HAL_CAN_ERROR_x](#)

枚举值

HAL_CAN_ERROR_None	无错误
HAL_CAN_ERROR_Bit	位错误
HAL_CAN_ERROR_Form	形式错误
HAL_CAN_ERROR_Stuff	填充错误
HAL_CAN_ERROR_Ack	应答错误
HAL_CAN_ERROR_Crc	CRC 错误
HAL_CAN_ERROR_Unknown	其他错误

在文件 [hal_lksmcu_can.h](#) 第 30 行定义.

4.11.1.3 HAL_CAN_FRAME_x

enum [HAL_CAN_FRAME_x](#)

枚举值

HAL_CAN_FRAME_IDE0_RTR0	ide=0 rtr=0
HAL_CAN_FRAME_IDE0_RTR1	ide=0 rtr=1
HAL_CAN_FRAME_IDE1_RTR0	ide=1 rtr=0
HAL_CAN_FRAME_IDE1_RTR1	ide=1 rtr=1
HAL_CAN_FRAME_IDE0_RTR0_FD1_BRS0	ide=0 rtr=0 fd=1 brs=0
HAL_CAN_FRAME_IDE0_RTR0_FD1_BRS1	ide=0 rtr=0 fd=1 brs=1
HAL_CAN_FRAME_IDE1_RTR0_FD1_BRS0	ide=1 rtr=0 fd=1 brs=0
HAL_CAN_FRAME_IDE1_RTR0_FD1_BRS1	ide=1 rtr=0 fd=1 brs=1

在文件 [hal_lksmcu_can.h](#) 第 13 行定义.

4.11.1.4 HAL_CAN_IDEFilter_x

enum [HAL_CAN_IDEFilter_x](#)

枚举值

HAL_CAN_IDEFilter_All	
HAL_CAN_IDEFilter_St	
HAL_CAN_IDEFilter_Ex	

在文件 [hal_lksmcu_can.h](#) 第 89 行定义.

4.11.1.5 HAL_CAN_STBMode_x

enum [HAL_CAN_STBMode_x](#)

枚举值

HAL_CAN_STBMode_FIFO	
HAL_CAN_STBMode_Priority	

在文件 [hal_lksmcu_can.h](#) 第 82 行定义.

4.11.1.6 HAL_CAN_STBTrig_x

enum [HAL_CAN_STBTrig_x](#)

枚举值

HAL_CAN_STBTrig_One	
HAL_CAN_STBTrig_All	

在文件 [hal_lksmcu_can.h](#) 第 76 行定义.

4.11.1.7 HAL_CAN_TSSTAT_x

enum [HAL_CAN_TSSTAT_x](#)

枚举值

HAL_CAN_TSSTAT_Empty	
HAL_CAN_TSSTAT_LessHalf	
HAL_CAN_TSSTAT_Half	
HAL_CAN_TSSTAT_Full	

在文件 [hal_lksmcu_can.h](#) 第 41 行定义.

4.11.1.8 HAL_CAN_x

enum [HAL_CAN_x](#)

枚举值

HAL_↔ CAN_0	

在文件 [hal_lksmcu_can.h](#) 第 7 行定义.

4.11.2 函数说明

4.11.2.1 HAL_CAN_ArbitrationLostCap()

```
uint8_t HAL_CAN_ArbitrationLostCap (
    HAL\_CAN\_x can_x )
```

在文件 [hal_lks32mc08x_can.c](#) 第 488 行定义.

4.11.2.2 HAL_CAN_Busoff()

```
uint8_t HAL_CAN_Busoff (
    HAL\_CAN\_x can_x )
```

在文件 [hal_lks32mc08x_can.c](#) 第 483 行定义.

4.11.2.3 HAL_CAN_Disable()

```
void HAL_CAN_Disable (
    HAL\_CAN\_x can_x )
```

在文件 [hal_lks32mc08x_can.c](#) 第 59 行定义.

4.11.2.4 HAL_CAN.Enable()

```
void HAL_CAN_Enable (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 40 行定义.

4.11.2.5 HAL_CAN.ErrorPassive()

```
uint8_t HAL_CAN_ErrorPassive (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 478 行定义.

4.11.2.6 HAL_CAN.GetBaud()

```
uint32_t HAL_CAN_GetBaud (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 120 行定义.

函数调用图:



4.11.2.7 HAL_CAN.GetClockDiv()

```
uint32_t HAL_CAN_GetClockDiv (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 108 行定义.

这是这个函数的调用关系图:

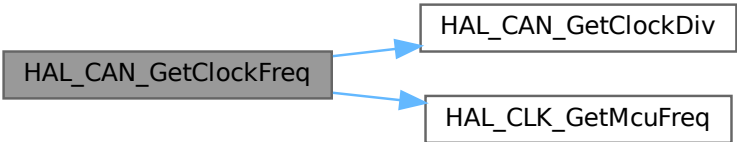


4.11.2.8 HAL_CAN_GetClockFreq()

```
uint32_t HAL_CAN_GetClockFreq (
    HAL_CAN_x can_x )
```

在文件 `hal_lks32mc08x.can.c` 第 114 行定义.

函数调用图:



这是这个函数的调用关系图:



4.11.2.9 HAL_CAN_IDx.Filter()

```
void HAL_CAN_IDx_Filter (
    HAL_CAN_x can_x,
    u8 ae_x,
    u32 id,
    u32 mask,
    HAL_CAN_IDEFilter_x ide_filter )
```

CAN滤波尝试剧疆

参数

08x的CAN滤波模拟为两个通道,	ae_x=0时使用单滤波, ae_x=1时使用双滤波。双滤波时, 对于扩展帧, 只匹配高16位。

在文件 `hal_lks32mc08x.can.c` 第 295 行定义.

4.11.2.10 HAL_CAN_IDx_FilterDisable()

```
void HAL_CAN_IDx_FilterDisable (
    HAL_CAN_x can_x,
    HAL_CAN_AE_x ae_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 389 行定义.

4.11.2.11 HAL_CAN_IDx_FilterEnable()

```
void HAL_CAN_IDx_FilterEnable (
    HAL_CAN_x can_x,
    HAL_CAN_AE_x ae_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 383 行定义.

4.11.2.12 HAL_CAN_KindOfError()

```
HAL_CAN_ERROR_x HAL_CAN_KindOfError (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 498 行定义.

4.11.2.13 HAL_CAN_LBME_Disable()

```
void HAL_CAN_LBME_Disable (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 448 行定义.

4.11.2.14 HAL_CAN_LBME_Enable()

```
void HAL_CAN_LBME_Enable (
    HAL_CAN_x can_x )
```

外部回环模式使能/禁止

更新记录

自发自收，自应答，需将芯片RX和TX接入CAN收发器，收发器可以不挂入总线。应在CAN初始化和ID滤波设置之后再使能。

在文件 [hal_lks32mc08x.can.c](#) 第 443 行定义.

4.11.2.15 HAL_CAN_LBMI_Disable()

```
void HAL_CAN_LBMI_Disable (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 432 行定义.

4.11.2.16 HAL_CAN_LBMI_Enable()

```
void HAL_CAN_LBMI_Enable (
    HAL_CAN_x can_x )
```

内部回环模式使能/禁止

更新记录

自发自收，无需自应答，芯片RX和TX引脚可悬空。应在CAN初始化和ID滤波设置之后再使能。

在文件 [hal_lks32mc08x.can.c](#) 第 427 行定义.

4.11.2.17 HAL_CAN_LOM_Disable()

```
void HAL_CAN_LOM_Disable (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 416 行定义.

4.11.2.18 HAL_CAN_LOM_Enable()

```
void HAL_CAN_LOM_Enable (
    HAL_CAN_x can_x )
```

监听模式使能/禁止

更新记录

如果设置了 TPE、TSONE 或 TSALL，则无法设置 LOM。如果 LOM 启用且 LBME 禁用，则无法启动传输。LOM=1 和 LBME=0 禁止所有传输。LOM=1 和 LBME=1 禁止应答相应接收到的帧和错误帧，但可以发送正常通信中，禁止使能该位。

在文件 [hal_lks32mc08x.can.c](#) 第 411 行定义.

4.11.2.19 HAL_CAN_Reset()

```
void HAL_CAN_Reset (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 19 行定义.

4.11.2.20 HAL_CAN_ROverflow()

```
uint8_t HAL_CAN_ROverflow (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 493 行定义.

4.11.2.21 HAL_CAN_RxData()

```
bool HAL_CAN_RxData (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 193 行定义.

4.11.2.22 HAL_CAN_RxDataBuf()

```
uint8_t HAL_CAN_RxDataBuf (
    HAL_CAN_x can_x,
    uint8_t * buf )
```

在文件 [hal_lks32mc08x.can.c](#) 第 256 行定义.

4.11.2.23 HAL_CAN_RxDataFdf()

```
uint8_t HAL_CAN_RxDataFdf (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 282 行定义.

4.11.2.24 HAL_CAN_RxDataID()

```
uint32_t HAL_CAN_RxDataID (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 251 行定义.

4.11.2.25 HAL_CAN_RxDataIde()

```
uint8_t HAL_CAN_RxDataIde (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 272 行定义.

4.11.2.26 HAL_CAN_RxDataLen()

```
uint8_t HAL_CAN_RxDataLen (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 267 行定义.

4.11.2.27 HAL_CAN_RxDataRtr()

```
uint8_t HAL_CAN_RxDataRtr (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 277 行定义.

4.11.2.28 HAL_CAN_RxErrorCnt()

```
uint8_t HAL_CAN_RxErrorCnt (
    HAL_CAN_x can_x )
```

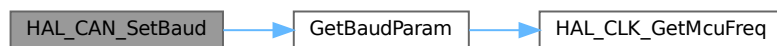
在文件 [hal_lks32mc08x.can.c](#) 第 468 行定义.

4.11.2.29 HAL_CAN_SetBaud()

```
void HAL_CAN_SetBaud (
    HAL_CAN_x can_x,
    uint32_t baud )
```

在文件 [hal_lks32mc08x.can.c](#) 第 77 行定义.

函数调用图:



4.11.2.30 HAL_CAN_SetClockDiv()

```
void HAL_CAN_SetClockDiv (
    HAL_CAN_x can_x,
    uint32_t div )
```

在文件 [hal_lks32mc08x.can.c](#) 第 99 行定义.

4.11.2.31 HAL_CAN_SetFastBaud()

```
void HAL_CAN_SetFastBaud (
    HAL_CAN_x can_x,
    uint32_t fbaud )
```

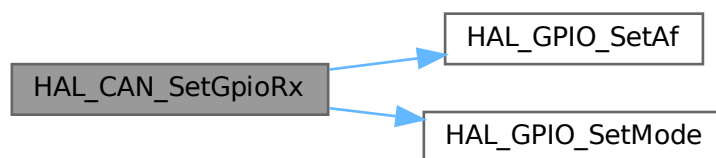
在文件 [hal_lks32mc08x.can.c](#) 第 92 行定义.

4.11.2.32 HAL_CAN_SetGpioRx()

```
void HAL_CAN_SetGpioRx (
    HAL_CAN_x can_x,
    HAL_GPIO_Px gpio_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 522 行定义.

函数调用图:

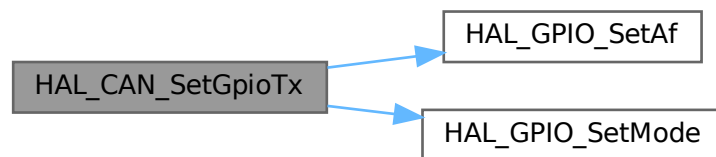


4.11.2.33 HAL_CAN_SetGpioTx()

```
void HAL_CAN_SetGpioTx (
    HAL_CAN_x can_x,
    HAL_GPIO_Px gpio_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 515 行定义.

函数调用图:



4.11.2.34 HAL_CAN_SetSTBMode()

```
void HAL_CAN_SetSTBMode (
    HAL_CAN_x can_x,
    HAL_CAN_STBMode_x mode_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 396 行定义.

4.11.2.35 HAL_CAN_SetTDC()

```
void HAL_CAN_SetTDC (
    HAL_CAN_x can_x,
    uint8_t sspoff )
```

在文件 [hal_lks32mc08x.can.c](#) 第 129 行定义.

4.11.2.36 HAL_CAN_SS_Disable()

```
void HAL_CAN_SS_Disable (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 463 行定义.

4.11.2.37 HAL_CAN_SS_Enable()

```
void HAL_CAN_SS_Enable (
    HAL_CAN_x can_x )
```

单次传输模式使能/禁止

更新记录

08x不支持

在文件 [hal_lks32mc08x.can.c](#) 第 458 行定义.

4.11.2.38 HAL_CAN_STBFull()

```
bool HAL_CAN_STBFull (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 177 行定义.

4.11.2.39 HAL_CAN_TSStat()

```
HAL_CAN_TSSTAT_x HAL_CAN_TSStat (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 183 行定义.

4.11.2.40 HAL_CAN_TxCancel()

```
void HAL_CAN_TxCancel (
    HAL_CAN_x can_x )
```

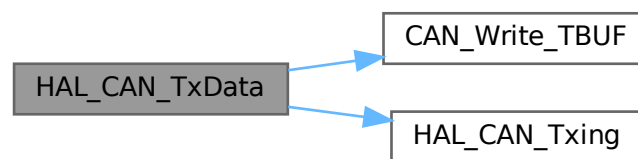
在文件 [hal_lks32mc08x.can.c](#) 第 167 行定义.

4.11.2.41 HAL_CAN_TxData()

```
bool HAL_CAN_TxData (
    HAL_CAN_x can_x,
    uint32_t id,
    HAL_CAN_FRAME_x frame_x,
    uint8_t * buf,
    uint32_t len )
```

在文件 [hal_lks32mc08x.can.c](#) 第 135 行定义.

函数调用图:



4.11.2.42 HAL_CAN_TxData_STBTrig()

```
bool HAL_CAN_TxData_STBTrig (
    HAL_CAN_x can_x,
    HAL_CAN_STBTrig_x trig )
```

在文件 [hal_lks32mc08x.can.c](#) 第 160 行定义.

4.11.2.43 HAL_CAN_TxData_STBWrite()

```
bool HAL_CAN_TxData_STBWrite (
    HAL_CAN_x can_x,
    uint32_t id,
    HAL_CAN_FRAME_x frame_x,
    uint8_t * buf,
    uint32_t len )
```

在文件 [hal_lks32mc08x.can.c](#) 第 149 行定义.

4.11.2.44 HAL_CAN_TxErrorCnt()

```
uint8_t HAL_CAN_TxErrorCnt (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 473 行定义.

4.11.2.45 HAL_CAN_Txing()

```
bool HAL_CAN_Txing (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x_can.c](#) 第 172 行定义.

这是这个函数的调用关系图:



4.12 hal_lksmcu_can.h

[浏览该文件的文档.](#)

```

00001 #ifndef __HAL_LKSMCU_CAN_H__
00002 #define __HAL_LKSMCU_CAN_H__
00003
00004 #include <stdint.h>
00005 #include "../include/hal_lksmcu_gpio.h"
00006
00007 typedef enum
00008 {
00009     HAL_CAN_0,
00010 } HAL_CAN_x;
00011
00012 /* CAN帧的属性 */
00013 typedef enum
00014 {
00015     /* 扩展帧、远程帧 */
00016     HAL_CAN_FRAME_IDE0_RTR0,
00017     HAL_CAN_FRAME_IDE0_RTR1,
00018     HAL_CAN_FRAME_IDE1_RTR0,
00019     HAL_CAN_FRAME_IDE1_RTR1,
00020
00021     /* 仅适用于CANFD */
00022     HAL_CAN_FRAME_IDE0_RTR0_FDL_BRS0,
00023     HAL_CAN_FRAME_IDE0_RTR0_FDL_BRS1,
00024
00025     HAL_CAN_FRAME_IDE1_RTR0_FDL_BRS0,
00026     HAL_CAN_FRAME_IDE1_RTR0_FDL_BRS1,
00027 } HAL_CAN_FRAME_x;
00028
00029 /* Error of kind */
00030 typedef enum
00031 {
00032     HAL_CAN_ERROR_None = 0,
00033     HAL_CAN_ERROR_Bit = 1,
00034     HAL_CAN_ERROR_Form = 2,
00035     HAL_CAN_ERROR_Stuff = 3,
00036     HAL_CAN_ERROR_Ack = 4,
00037     HAL_CAN_ERROR_Crc = 5,
00038     HAL_CAN_ERROR_Unknown = 6,
00039 } HAL_CAN_ERROR_x;
00040
00041 typedef enum
00042 {
00043     HAL_CAN_TSSTAT_Empty = 0,
00044     HAL_CAN_TSSTAT_LessHalf = 1,
00045     HAL_CAN_TSSTAT_Half = 2,
00046     HAL_CAN_TSSTAT_Full = 3,
00047 } HAL_CAN_TSSTAT_x;
00048
00049 typedef enum
00050 {
```

```

00051     HAL_CAN_AE_0,
00052     HAL_CAN_AE_1,
00053
00054     /* 以下仅在07x支持 */
00055     HAL_CAN_AE_2,
00056     HAL_CAN_AE_3,
00057
00058     /* 以下仅在45x支持 */
00059     HAL_CAN_AE_4,
00060     HAL_CAN_AE_5,
00061     HAL_CAN_AE_6,
00062     HAL_CAN_AE_7,
00063
00064     HAL_CAN_AE_8,
00065     HAL_CAN_AE_9,
00066     HAL_CAN_AE_10,
00067     HAL_CAN_AE_11,
00068
00069     HAL_CAN_AE_12,
00070     HAL_CAN_AE_13,
00071     HAL_CAN_AE_14,
00072     HAL_CAN_AE_15,
00073 } HAL_CAN_AE_x;
00074
00075 /* STB发送时, 需要填入的触发参数 */
00076 typedef enum
00077 {
00078     HAL_CAN_STBTrig_One, /* 触发STB发送一帧 */
00079     HAL_CAN_STBTrig_All, /* 触发STB发送所有帧 */
00080 } HAL_CAN_STBTrig_x;
00081
00082 typedef enum
00083 {
00084     HAL_CAN_STBMode_FIFO, /* 发送缓冲区FIFO模式 */
00085     HAL_CAN_STBMode_Priority, /* 发送缓冲区优先级模式 */
00086 } HAL_CAN_STBMode_x;
00087
00088 /* 设置ID过滤时, 对标准帧和扩展帧的设置选项 */
00089 typedef enum
00090 {
00091     HAL_CAN_IDEFilter_All, /* 接收标准帧+扩展帧 */
00092     HAL_CAN_IDEFilter_St, /* 只接收标准帧 */
00093     HAL_CAN_IDEFilter_Ex, /* 只接收扩展帧 */
00094 } HAL_CAN_IDEFilter_x;
00095
00096 void HAL_CAN_Reset(HAL_CAN_x can_x);
00097 void HAL_CAN_Enable(HAL_CAN_x can_x);
00098 void HAL_CAN_Disable(HAL_CAN_x can_x);
00099
00100 void HAL_CAN_SetBaud(HAL_CAN_x can_x, uint32_t baud);
00101 void HAL_CAN_SetClockDiv(HAL_CAN_x can_x, uint32_t div); /* 设置CAN预分频 div=实际除数(1~64)
00102 uint32_t HAL_CAN_GetClockDiv(HAL_CAN_x can_x); /* 获取CAN预分频实际除数
00103 uint32_t HAL_CAN_GetClockFreq(HAL_CAN_x can_x); /* 获取分频后工作时钟(Hz)
00104 uint32_t HAL_CAN_GetBaud(HAL_CAN_x can_x); /* 获取当前标称波特率(bps)
00105 void HAL_CAN_SetFastBaud(HAL_CAN_x can_x, uint32_t fbaud);
00106 void HAL_CAN_SetTDC(HAL_CAN_x can_x, uint8_t sspoff);
00107
00108 /* PTB方式发送CAN数据 */
00109 bool HAL_CAN_TxData(HAL_CAN_x can_x, uint32_t id, HAL_CAN_FRAME_x frame_x, uint8_t *buf, uint32_t len);
00110
00111 /* STB方式发送数据, 写数据和触发分两步执行 */
00112 bool HAL_CAN_TxData_STBWrite(HAL_CAN_x can_x, uint32_t id, HAL_CAN_FRAME_x frame_x, uint8_t *buf, uint32_t len);
00113 bool HAL_CAN_TxData_STBTrig(HAL_CAN_x can_x, HAL_CAN_STBTrig_x trig);
00114
00115 void HAL_CAN_TxCancel(HAL_CAN_x can_x);
00116
00117 bool HAL_CAN_Txing(HAL_CAN_x can_x);
00118 bool HAL_CAN_STBFull(HAL_CAN_x can_x);
00119 HAL_CAN_TSSTAT_x HAL_CAN_TSStat(HAL_CAN_x can_x);
00120
00121 bool HAL_CAN_RxData(HAL_CAN_x can_x);
00122 uint32_t HAL_CAN_RxDataID(HAL_CAN_x can_x);
00123 uint8_t HAL_CAN_RxDataBuf(HAL_CAN_x can_x, uint8_t *buf);
00124 uint8_t HAL_CAN_RxDataLen(HAL_CAN_x can_x);
00125 uint8_t HAL_CAN_RxDataIde(HAL_CAN_x can_x);
00126 uint8_t HAL_CAN_RxDataRtr(HAL_CAN_x can_x);
00127 uint8_t HAL_CAN_RxDataFdf(HAL_CAN_x can_x);
00128
00129 void HAL_CAN_IDx_Filter(HAL_CAN_x can_x, u8 x, u32 id, u32 mask, HAL_CAN_IDEFilter_x ide_filter);
00130 void HAL_CAN_IDx_FilterEnable(HAL_CAN_x can_x, HAL_CAN_AE_x ae_x);
00131 void HAL_CAN_IDx_FilterDisable(HAL_CAN_x can_x, HAL_CAN_AE_x ae_x);
00132
00133 void HAL_CAN_SetSTBMode(HAL_CAN_x can_x, HAL_CAN_STBMode_x mode_x);
00134
00135 void HAL_CAN_LOM_Enable(HAL_CAN_x can_x);
00136 void HAL_CAN_LOM_Disable(HAL_CAN_x can_x);

```

```

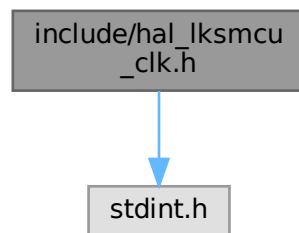
00137
00138 void HAL_CAN_LBMI_Enable(HAL_CAN_x can_x);
00139 void HAL_CAN_LBMI_Disable(HAL_CAN_x can_x);
00140
00141 void HAL_CAN_LBME_Enable(HAL_CAN_x can_x);
00142 void HAL_CAN_LBME_Disable(HAL_CAN_x can_x);
00143
00144 void HAL_CAN_SS_Enable(HAL_CAN_x can_x);
00145 void HAL_CAN_SS_Disable(HAL_CAN_x can_x);
00146
00147 uint8_t HAL_CAN_RxErrorCnt(HAL_CAN_x can_x);
00148 uint8_t HAL_CAN_TxErrorCnt(HAL_CAN_x can_x);
00149 uint8_t HAL_CAN_ErrorPassive(HAL_CAN_x can_x);
00150 uint8_t HAL_CAN_Busoff(HAL_CAN_x can_x);
00151 uint8_t HAL_CAN_ArbitrationLostCap(HAL_CAN_x can_x);
00152 uint8_t HAL_CAN_ROverflow(HAL_CAN_x can_x);
00153 HAL_CAN_ERROR_x HAL_CAN_KindOfError(HAL_CAN_x can_x);
00154
00155 void HAL_CAN_SetGpioTx(HAL_CAN_x can_x, HAL_GPIO_Px gpio_x);
00156 void HAL_CAN_SetGpioRx(HAL_CAN_x can_x, HAL_GPIO_Px gpio_x);
00157
00158 #endif

```

4.13 include/hal_lksmcu_clk.h 文件参考

#include "stdint.h"

hal_lksmcu_clk.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



枚举

- enum HAL_CLK_FREQ_x {
 HAL_CLK_FREQ_RC_4MHZ, HAL_CLK_FREQ_RC_12MHZ, HAL_CLK_FREQ_RC_24MHZ, HAL_CLK_FREQ_RC_48MHZ,
 HAL_CLK_FREQ_RC_96MHZ }
 主频

函数

- void [HAL_CLK_SetMcuFreq \(HAL_CLK_FREQ_x\)](#)
设置主频
- uint32_t [HAL_CLK_GetMcuFreq \(void\)](#)
获取主频

4.13.1 枚举类型说明

4.13.1.1 HAL_CLK_FREQ_x

enum [HAL_CLK_FREQ_x](#)

主频

枚举值

HAL_CLK_FREQ_RC_4MHZ	主频4Mhz
HAL_CLK_FREQ_RC_12MHZ	主频12Mhz
HAL_CLK_FREQ_RC_24MHZ	主频24Mhz
HAL_CLK_FREQ_RC_48MHZ	主频48Mhz
HAL_CLK_FREQ_RC_96MHZ	主频96Mhz

在文件 [hal_lksmcu_clk.h](#) 第 6 行定义.

4.13.2 函数说明

4.13.2.1 HAL_CLK_GetMcuFreq()

```
uint32_t HAL_CLK_GetMcuFreq (  
    void )
```

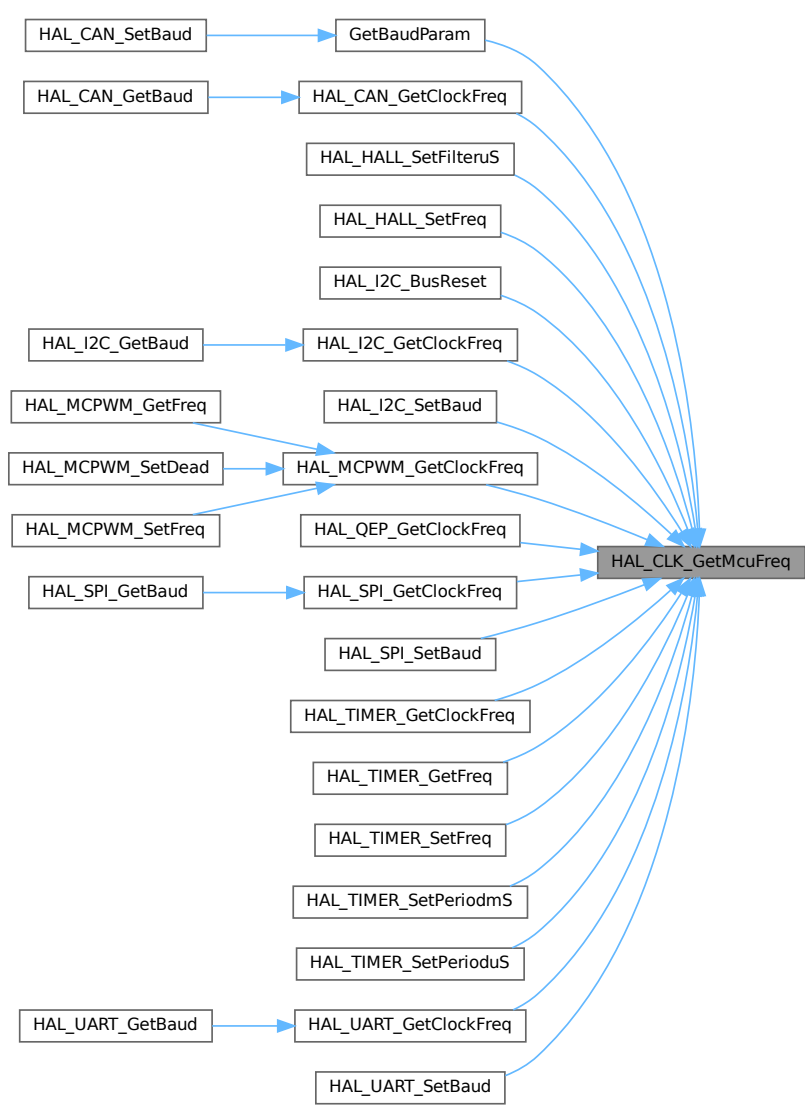
获取主频

返回

主频

在文件 [hal_lks32mc08x_clk.c](#) 第 81 行定义.

这是这个函数的调用关系图:



4.13.2.2 HAL_CLK.SetMcuFreq()

```
void HAL_CLK_SetMcuFreq (
    HAL_CLK_FREQ_x  FREQ_x )
```

设置主频

参数

<i>FREQ_x</i>	主频
_x	

< 主频4Mhz

< 主频12Mhz

< 主频24Mhz

< 主频48Mhz

< 主频96Mhz

在文件 [hal_lks32mc08x.clk.c](#) 第 11 行定义.

这是这个函数的调用关系图:



4.14 hal_lksmcu_clk.h

[浏览该文件的文档.](#)

```

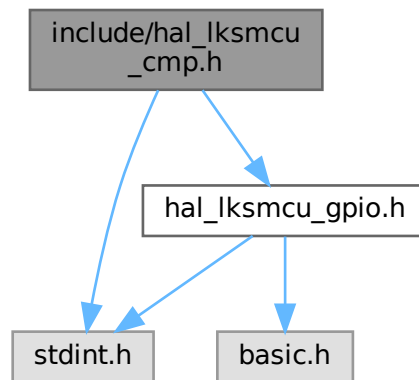
00001 #ifndef __HAL_LKSMCU_CLK_H__
00002 #define __HAL_LKSMCU_CLK_H__
00003 #include "stdint.h"
00004
00006 typedef enum
00007 {
00008     HAL_CLK_FREQ_RC_4MHZ,
00009     HAL_CLK_FREQ_RC_12MHZ,
00010     HAL_CLK_FREQ_RC_24MHZ,
00011     HAL_CLK_FREQ_RC_48MHZ,
00012     HAL_CLK_FREQ_RC_96MHZ,
00013
00014 #ifdef HAL_TEST_NOT_SUPPORTED
00015     HAL_CLK_FREQ_RC_6MHZ,
00016     HAL_CLK_FREQ_RC_8MHZ,
00017     HAL_CLK_FREQ_RC_9MHZ,
00018     HAL_CLK_FREQ_RC_16MHZ,
00019     HAL_CLK_FREQ_RC_18MHZ,
00020     HAL_CLK_FREQ_RC_32MHZ,
00021     HAL_CLK_FREQ_RC_36MHZ,
00022     HAL_CLK_FREQ_RC_64MHZ,
00023     HAL_CLK_FREQ_RC_72MHZ,
00024     HAL_CLK_FREQ_RC_192MHZ,
00025
00026     HAL_CLK_FREQ_XTAL_6MHZ,
00027     HAL_CLK_FREQ_XTAL_12MHZ,
00028     HAL_CLK_FREQ_XTAL_24MHZ,
00029     HAL_CLK_FREQ_XTAL_48MHZ,
00030     HAL_CLK_FREQ_XTAL_96MHZ,
00031     HAL_CLK_FREQ_XTAL_192MHZ,
00032 #endif
00033 } HAL_CLK_FREQ_x;
00034 void HAL_CLK_SetMcuFreq(HAL_CLK_FREQ_x);
00035 uint32_t HAL_CLK_GetMcuFreq(void);
00036 #endif
  
```


4.15 include/hal_lksmcu_cmp.h 文件参考

```
#include "stdint.h"
```

```
#include "hal_lksmcu_gpio.h"
```

hal_lksmcu_cmp.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



枚举

- enum HAL_CMP_x {
HAL_CMP_0, HAL_CMP_1, HAL_CMP_2, HAL_CMP_3,
HAL_CMP_4, HAL_CMP_5 }
- enum HAL_CMP_CHANNEL_IPx {
HAL_CMP_CHANNEL_IP0 = 0, HAL_CMP_CHANNEL_IP_OPA0_IP = 1, HAL_CMP_CHANNEL_IP_OPA3_IP =
1, HAL_CMP_CHANNEL_IP_OPA0_OUTP = 2,
HAL_CMP_CHANNEL_IP_OPA2_OUTP = 2, HAL_CMP_CHANNEL_IP_OPA1_OUTP = 3, HAL_CMP_CHANNEL_IP_OPA3_OUTP
= 3, HAL_CMP_CHANNEL_IP1 = 4,
HAL_CMP_CHANNEL_IP2 = 5, HAL_CMP_CHANNEL_IP3 = 6, HAL_CMP_CHANNEL_IP4 = 7,
HAL_CMP_CHANNEL_IP_AVSS = 7 }
- enum HAL_CMP_CHANNEL_INx {
HAL_CMP_CHANNEL_IN_0, HAL_CMP_CHANNEL_IN_REF, HAL_CMP_CHANNEL_IN_DAC0, HAL_CMP_CHANNEL_IN_MID
,
HAL_CMP_CHANNEL_IN_DAC = HAL_CMP_CHANNEL_IN_DAC0 }
- enum HAL_CMP_MODE_x { HAL_CMP_MODE_RISING_EDGE = 0x00U, HAL_CMP_MODE_FALLING_EDGE
= 0x01U, HAL_CMP_MODE_HIGH_LEVEL = 0x02U, HAL_CMP_MODE_LOW_LEVEL = 0x03U }
- enum HAL_CMP_HYS_x { HAL_CMP_HYS_20mV = 0, HAL_CMP_HYS_0mV = 1 }

函数

- void [HAL_CMP_Reset](#) (HAL_CMP_x)
模块复位
- void [HAL_CMP_Enable](#) (HAL_CMP_x)
模块打开
- void [HAL_CMP_Disable](#) (HAL_CMP_x)
模块关闭
- void [HAL_CMP_SetChannel](#) (HAL_CMP_x, HAL_CMP_CHANNEL_IPx, HAL_CMP_CHANNEL_INx)
模块关闭
- uint32_t [HAL_CMP_SetDigitalFilter](#) (HAL_CMP_x, uint32_t filter)
获取CMP模块的数字滤波宽度
- void [HAL_CMP_SetMode](#) (HAL_CMP_x, HAL_CMP_MODE_x)
设置CMP模块工作模式
- void [HAL_CMP_SetHysteresis](#) (HAL_CMP_x, HAL_CMP_HYS_x)
设置比较器回差
- void [HAL_CMP_SetGpioOut](#) (HAL_CMP_x, HAL_GPIO_Px cmpgpio)
设置CMP模块的GPIO
- void [HAL_CMP_SetGpioIn](#) (HAL_CMP_x, HAL_GPIO_Px cmpgpio)
设置CMP模块的GPIO
- uint8_t [HAL_CMP_GetData](#) (HAL_CMP_x)
获取CMP比较结果

4.15.1 枚举类型说明

4.15.1.1 HAL_CMP_CHANNEL_INx

enum [HAL_CMP_CHANNEL_INx](#)

枚举值

HAL_CMP_CHANNEL_IN_0	CMPN选择为外部负端输入
HAL_CMP_CHANNEL_IN_REF	CMPN选择为内部基准源
HAL_CMP_CHANNEL_IN_DAC0	CMPN选择为DAC输出
HAL_CMP_CHANNEL_IN_MID	CMPN选择为反电动势虚拟中心点
HAL_CMP_CHANNEL_IN_DAC	CMPN选择为DAC输出

在文件 [hal_lksmcu_cmp.h](#) 第 57 行定义。

4.15.1.2 HAL_CMP_CHANNEL_IPx

enum [HAL_CMP_CHANNEL_IPx](#)

枚举值

HAL_CMP_CHANNEL_IP0	CMP模块外部正端输入0
HAL_CMP_CHANNEL_IP_OPA0_IP	仅CMP0 运放0正端输入

枚举值

HAL_CMP_CHANNEL_IP_OPA3_IP	仅CMP1 运放3正端输入
HAL_CMP_CHANNEL_IP_OPA0_OUTP	仅CMP0 运放0正端输出
HAL_CMP_CHANNEL_IP_OPA2_OUTP	仅CMP1 运放2正端输出
HAL_CMP_CHANNEL_IP_OPA1_OUTP	仅CMP0 运放1正端输出
HAL_CMP_CHANNEL_IP_OPA3_OUTP	仅CMP1 运放3正端输出
HAL_CMP_CHANNEL_IP1	CMP模块外部正端输入1
HAL_CMP_CHANNEL_IP2	CMP模块外部正端输入2
HAL_CMP_CHANNEL_IP3	CMP模块外部正端输入3
HAL_CMP_CHANNEL_IP4	仅CMP0 CMP模块外部正端输入4
HAL_CMP_CHANNEL_IP_AVSS	仅CMP1 模拟地

在文件 [hal_lksmcu_cmp.h](#) 第 15 行定义.

4.15.1.3 HAL_CMP_HYS_x

enum [HAL_CMP_HYS_x](#)

枚举值

HAL_CMP_HYS_20mV	回差20mV(默认)
HAL_CMP_HYS_0mV	关闭回差

在文件 [hal_lksmcu_cmp.h](#) 第 77 行定义.

4.15.1.4 HAL_CMP_MODE_x

enum [HAL_CMP_MODE_x](#)

枚举值

HAL_CMP_MODE_RISING_EDGE	上升沿触发模式
HAL_CMP_MODE_FALLING_EDGE	下降沿触发模式
HAL_CMP_MODE_HIGH_LEVEL	高电平检测模式
HAL_CMP_MODE_LOW_LEVEL	低电平检测模式

在文件 [hal_lksmcu_cmp.h](#) 第 70 行定义.

4.15.1.5 HAL_CMP_x

enum [HAL_CMP_x](#)

枚举值

HAL_↔ CMP_0	CMP模块0
HAL_↔ CMP_1	CMP模块1
HAL_↔ CMP_2	CMP模块2
HAL_↔ CMP_3	CMP模块3
HAL_↔ CMP_4	CMP模块4
HAL_↔ CMP_5	CMP模块5

在文件 [hal_lksmcu_cmp.h](#) 第 5 行定义.

4.15.2 函数说明

4.15.2.1 HAL_CMP_Disable()

```
void HAL_CMP_Disable (
    HAL_CMP_x cmp_x )
```

模块关闭

参数

cmp↔ _x	使用的CMP模块编号

在文件 [hal_lks32mc08x_cmp.c](#) 第 56 行定义.

这是这个函数的调用关系图:



4.15.2.2 HAL_CMP_Enable()

```
void HAL_CMP_Enable (
    HAL_CMP_x cmp_x )
```

模块打开

参数

<i>cmp</i> _↔ <i>_x</i>	使用的CMP模块编号

在文件 [hal_lks32mc08x_cmp.c](#) 第 31 行定义.

4.15.2.3 HAL_CMP_GetData()

```
uint8_t HAL_CMP_GetData (
    HAL_CMP_x cmp_x )
```

获取CMP比较结果

参数

<i>cmp</i> _↔ <i>_x</i>	CMP模块编号

返回
比较结果

在文件 [hal_lks32mc08x_cmp.c](#) 第 200 行定义.

4.15.2.4 HAL_CMP_Reset()

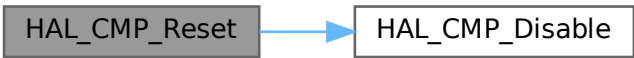
```
void HAL_CMP_Reset (
    HAL_CMP_x cmp_x )
```

模块复位

参数

<i>cmp</i> _↔ <i>_x</i>	使用的CMP模块编号

在文件 [hal_lks32mc08x_cmp.c](#) 第 22 行定义.
函数调用图:



4.15.2.5 HAL_CMP_SetChannel()

```
void HAL_CMP_SetChannel (
    HAL_CMP_x cmp_x,
    HAL_CMP_CHANNEL_IPx channel_p_x,
    HAL_CMP_CHANNEL_INx channel_n_x )
```

模块关闭

参数

<i>cmp_x</i>	使用的CMP模块编号
<i>channel_↔ p_x</i>	正端通道
<i>channel_↔ n_x</i>	负端通道

在文件 [hal_lks32mc08x_cmp.c](#) 第 83 行定义.

4.15.2.6 HAL_CMP_SetDigitalFilter()

```
uint32_t HAL_CMP_SetDigitalFilter (
    HAL_CMP_x cmp_x,
    uint32_t filter )
```

获取CMP模块的数字滤波宽度

参数

<i>cmp_↔ _x</i>	CMP模块编号
<i>filter</i>	滤波宽度

返回

实际滤波宽度

在文件 [hal_lks32mc08x_cmp.c](#) 第 114 行定义.

4.15.2.7 HAL_CMP_SetGpioIn()

```
void HAL_CMP_SetGpioIn (
    HAL_CMP_x cmp_x,
    HAL_GPIO_Px cmp_gpio )
```

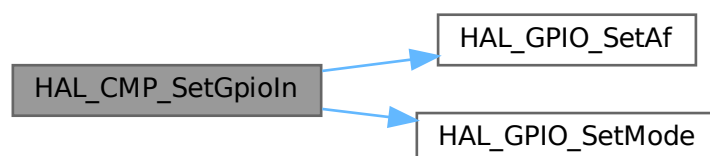
设置CMP模块的GPIO

参数

<i>cmp_x</i>	CMP模块编号
<i>cmpgpio</i>	GPIO编号

在文件 [hal_lks32mc08x_cmp.c](#) 第 188 行定义.

函数调用图:



4.15.2.8 HAL_CMP_SetGpioOut()

```
void HAL_CMP_SetGpioOut (
    HAL_CMP_x cmp_x,
    HAL_GPIO_Px cmpgpio )
```

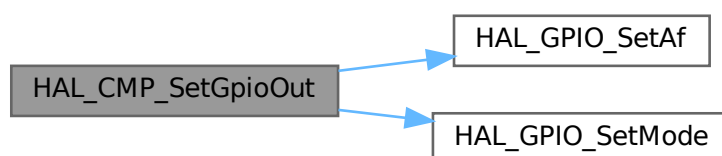
设置CMP模块的GPIO

参数

<i>cmp_x</i>	I2C模块编号
<i>cmpgpio</i>	GPIO编号

在文件 [hal_lks32mc08x_cmp.c](#) 第 176 行定义.

函数调用图:



4.15.2.9 HAL_CMP_SetHysteresis()

```
void HAL_CMP_SetHysteresis (
    HAL_CMP_x cmp_x,
    HAL_CMP_HYS_x hys_x )
```

设置比较器回差

参数

<i>cmp_x</i>	CMP模块编号(回差为所有CMP模块共用配置,与 <i>cmp_x</i> 无关,本参数保留暂不使用)
<i>hys_x</i>	回差选择

在文件 [hal_lks32mc08x.cmp.c](#) 第 229 行定义.

4.15.2.10 HAL_CMP_SetMode()

```
void HAL_CMP_SetMode (
    HAL_CMP_x cmp_x,
    HAL_CMP_MODE_x mode_x )
```

设置CMP模块工作模式

参数

<i>cmp_x</i>	CMP模块编号
<i>mode_x</i>	工作模式

在文件 [hal_lks32mc08x.cmp.c](#) 第 149 行定义.

4.16 hal_lksmcu_cmp.h

[浏览该文件的文档.](#)

```
00001 #ifndef __HAL_LKSMCU_CMP_H__
00002 #define __HAL_LKSMCU_CMP_H__
00003 #include "stdint.h"
00004 #include "hal_lksmcu_gpio.h"
00005 typedef enum
00006 {
00007     HAL_CMP_0,
00008     HAL_CMP_1,
00009     HAL_CMP_2,
00010     HAL_CMP_3,
00011     HAL_CMP_4,
00012     HAL_CMP_5,
00013 } HAL_CMP_x;
00014
00015 typedef enum
00016 {
00017     HAL_CMP_CHANNEL_IP0 = 0,
00018     HAL_CMP_CHANNEL_IP.OPA0_IP = 1,
00019     HAL_CMP_CHANNEL_IP.OPA3_IP = 1,
00020     HAL_CMP_CHANNEL_IP.OPA0_OUTP = 2,
```



```

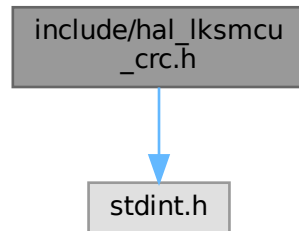
00021     HAL_CMP_CHANNEL_IP_OPA2_OUTP = 2,
00022     HAL_CMP_CHANNEL_IP_OPA1_OUTP = 3,
00023     HAL_CMP_CHANNEL_IP_OPA3_OUTP = 3,
00024     HAL_CMP_CHANNEL_IP1          = 4,
00025     HAL_CMP_CHANNEL_IP2          = 5,
00026     HAL_CMP_CHANNEL_IP3          = 6,
00027     HAL_CMP_CHANNEL_IP4          = 7,
00028     HAL_CMP_CHANNEL_IP_AVSS      = 7,
00029
00030 #ifdef HAL_TEST_NOT_SUPPORTED
00031     HAL_CMP_CHANNEL_IP5,
00032     HAL_CMP_CHANNEL_IP6,
00033     HAL_CMP_CHANNEL_IP7,
00034     HAL_CMP_CHANNEL_IP_DAC0,
00035     HAL_CMP_CHANNEL_IP_DAC1,
00036     HAL_CMP_CHANNEL_IP_OPA_OUT_CMP0,
00037     HAL_CMP_CHANNEL_IP_OPA0_IN,
00038     HAL_CMP_CHANNEL_IP_OPA0_OUTN,
00039     HAL_CMP_CHANNEL_IP_OPA1_IP,
00040     HAL_CMP_CHANNEL_IP_OPA1_IN,
00041     HAL_CMP_CHANNEL_IP_OPA1_OUTN,
00042     HAL_CMP_CHANNEL_IP_OPA2_IP,
00043     HAL_CMP_CHANNEL_IP_OPA2_IN,
00044     HAL_CMP_CHANNEL_IP_OPA2_OUTN,
00045     HAL_CMP_CHANNEL_IP_OPA3_IN,
00046     HAL_CMP_CHANNEL_IP_OPA3_OUTN,
00047     HAL_CMP_CHANNEL_IP_OPA4_IP,
00048     HAL_CMP_CHANNEL_IP_OPA4_IN,
00049     HAL_CMP_CHANNEL_IP_OPA4_OUTP,
00050     HAL_CMP_CHANNEL_IP_OPA4_OUTN,
00051     HAL_CMP_CHANNEL_IP_OPA5_IP,
00052     HAL_CMP_CHANNEL_IP_OPA5_IN,
00053     HAL_CMP_CHANNEL_IP_OPA5_OUTP,
00054     HAL_CMP_CHANNEL_IP_OPA5_OUTN,
00055 #endif
00056 } HAL_CMP_CHANNEL_IPx;
00057 typedef enum
00058 {
00059     HAL_CMP_CHANNEL_IN_0,
00060     HAL_CMP_CHANNEL_IN_REF,
00061     HAL_CMP_CHANNEL_IN_DAC0,
00062     HAL_CMP_CHANNEL_IN_MID,
00063
00064     HAL_CMP_CHANNEL_IN_DAC = HAL_CMP_CHANNEL_IN_DAC0,
00065
00066 #ifdef HAL_TEST_NOT_SUPPORTED
00067     HAL_CMP_CHANNEL_IN_DAC1,
00068 #endif
00069 } HAL_CMP_CHANNEL_INx;
00070 typedef enum
00071 {
00072     HAL_CMP_MODE_RISING_EDGE = 0x00U,
00073     HAL_CMP_MODE_FALLING_EDGE = 0x01U,
00074     HAL_CMP_MODE_HIGH_LEVEL = 0x02U,
00075     HAL_CMP_MODE_LOW_LEVEL = 0x03U
00076 } HAL_CMP_MODE_x;
00077 typedef enum
00078 {
00079     HAL_CMP_HYS_20mV = 0,
00080     HAL_CMP_HYS_0mV = 1,
00081 } HAL_CMP_HYS_x;
00082 // 初始化
00083 void HAL_CMP_Reset(HAL_CMP_x); // 模块复位
00084 void HAL_CMP_Enable(HAL_CMP_x); // 模块打开
00085 void HAL_CMP_Disable(HAL_CMP_x); // 模块关闭
00086
00087 void HAL_CMP_SetChannel(HAL_CMP_x, HAL_CMP_CHANNEL_IPx, HAL_CMP_CHANNEL_INx); // 设置CMP输入信号通道
00088 uint32_t HAL_CMP_SetDigitalFilter(HAL_CMP_x, uint32_t filter); // 设置数字滤波宽度 0-0xffff
// 单位MCU时钟周期,根据芯片不同,实际滤波宽度可能会有差异,返回值,实际滤波宽度
00089 void HAL_CMP_SetMode(HAL_CMP_x, HAL_CMP_MODE_x); // 设置CMP中断触发模式
00090 void HAL_CMP_SetHysteresis(HAL_CMP_x, HAL_CMP_HYS_x); // 设置比较器回差
00091 void HAL_CMP_SetGpioOut(HAL_CMP_x, HAL_GPIO_Px cmpgpio);
00092 void HAL_CMP_SetGpioIn(HAL_CMP_x, HAL_GPIO_Px cmpgpio);
00093 uint8_t HAL_CMP_GetData(HAL_CMP_x);
00094 #endif

```

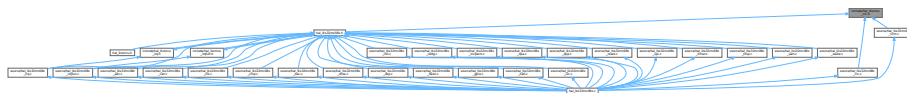
4.17 include/hal_lksmcu_crc.h 文件参考

```
#include "stdint.h"
```

hal_lksmcu_crc.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



枚举

- enum `HAL_CRC_x` { `HAL_CRC_0` }
- enum `HAL_CRC_MODEL_x` {
`HAL_CRC_MODEL_CRC8`, `HAL_CRC_MODEL_CRC8_ITU`, `HAL_CRC_MODEL_CRC8_ROHC`, `HAL_CRC_MODEL_CRC8_MAX`,
`HAL_CRC_MODEL_CRC16_IBMx1`, `HAL_CRC_MODEL_CRC16_MAXIM`, `HAL_CRC_MODEL_CRC16_USBx1`,
`HAL_CRC_MODEL_CRC16_MODBU`,
`HAL_CRC_MODEL_CRC16_CCITT`, `HAL_CRC_MODEL_CRC16_CCITT_FALSE`, `HAL_CRC_MODEL_CRC16_X25`,
`HAL_CRC_MODEL_CRC16_XMODEM`,
`HAL_CRC_MODEL_CRC16_DNP`, `HAL_CRC_MODEL_CRC32`, `HAL_CRC_MODEL_CRC32_MPEG_2` }

函数

- void `HAL_CRC_Reset` (`HAL_CRC_x`)
CRC模块复位
- void `HAL_CRC_Enable` (`HAL_CRC_x`)
CRC模块使能
- void `HAL_CRC_Disable` (`HAL_CRC_x`)
CRC模块失能
- uint32_t `HAL_CRC_Calc` (`HAL_CRC_x`, `HAL_CRC_MODEL_x`, uint8_t *data, uint32_t len)
CRC计算

4.17.1 枚举类型说明

4.17.1.1 HAL_CRC_MODEL_x

```
enum HAL_CRC_MODEL_x
```

枚举值

HAL_CRC_MODEL_CRC8	8 07 00 00 false false
HAL_CRC_MODEL_CRC8_ITU	8 07 00 55 false false
HAL_CRC_MODEL_CRC8_ROHC	8 07 FF 00 true true
HAL_CRC_MODEL_CRC8_MAXIMx	8 31 00 00 true true
HAL_CRC_MODEL_CRC16_IBMx1	16 8005 0000 0000 true true
HAL_CRC_MODEL_CRC16_MAXIM	16 8005 0000 FFFF true true
HAL_CRC_MODEL_CRC16_USBx1	16 8005 FFFF FFFF true true
HAL_CRC_MODEL_CRC16_MODBU	16 8005 FFFF 0000 true true
HAL_CRC_MODEL_CRC16_CCITT	16 1021 0000 0000 true true
HAL_CRC_MODEL_CRC16_CCITT_FALSE	16 1021 FFFF 0000 false false
HAL_CRC_MODEL_CRC16_X25	16 1021 FFFF FFFF true true
HAL_CRC_MODEL_CRC16_XMODEM	16 1021 0000 0000 false false
HAL_CRC_MODEL_CRC16_DNP	16 3D65 0000 FFFF true true
HAL_CRC_MODEL_CRC32	32 04C11DB7 FFFFFFFF FFFFFFFF true true
HAL_CRC_MODEL_CRC32_MPEG_2	32 04C11DB7 FFFFFFFF 00000000 false false

在文件 [hal_lksmcu_crc.h](#) 第 10 行定义.

4.17.1.2 HAL_CRC_x

enum [HAL_CRC_x](#)

枚举值

HAL_↔ CRC_0	CRC模块0

在文件 [hal_lksmcu_crc.h](#) 第 5 行定义.

4.17.2 函数说明

4.17.2.1 HAL_CRC.Calc()

```
uint32_t HAL_CRC_Calc (
    HAL\_CRC\_x CRC_x,
    HAL\_CRC\_MODEL\_x model,
    uint8_t * data,
    uint32_t len )
```

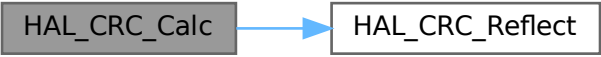
CRC计算

参数

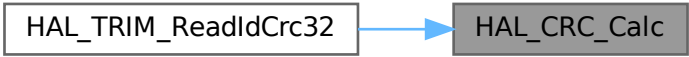
<i>CRC</i> ↔ <i>_x</i>	CRC模块编号
<i>model</i>	CRC模型
<i>data</i>	数据地址
	数据长度

在文件 `hal_lks32mc08x_crc.c` 第 90 行定义.

函数调用图:



这是这个函数的调用关系图:



4.17.2.2 HAL_CRC.Disable()

```
void HAL_CRC_Disable (
    HAL_CRC_x CRC_x )
```

CRC模块失能

参数

CRC↔ _x	CRC模块编号

在文件 `hal_lks32mc08x_crc.c` 第 78 行定义.

4.17.2.3 HAL_CRC.Enable()

```
void HAL_CRC_Enable (
    HAL_CRC_x CRC_x )
```

CRC模块使能

参数

CRC↔ _x	CRC模块编号

在文件 `hal_lks32mc08x_crc.c` 第 69 行定义.

4.17.2.4 HAL_CRC_Reset()

```
void HAL_CRC_Reset (
    HAL_CRC_x CRC_x )
```

CRC模块复位

参数

<code>CRC_x</code>	CRC模块编号

在文件 `hal_lks32mc08x_crc.c` 第 60 行定义.

4.18 hal_lksmcu_crc.h

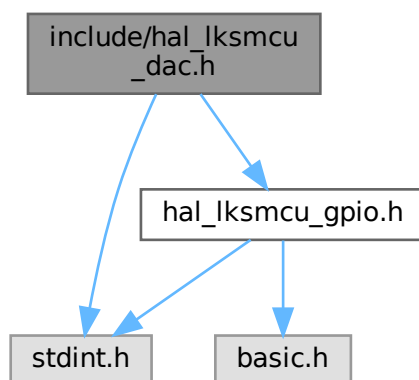
[浏览该文件的文档.](#)

```
00001 #ifndef __HAL_LKSMCU_CRC_H__
00002 #define __HAL_LKSMCU_CRC_H__
00003
00004 #include "stdint.h"
00005 typedef enum
00006 {
00007     HAL_CRC_0,
00008 } HAL_CRC_x;
00009
00010 typedef enum
00011 {
00012     HAL_CRC_MODEL_CRC8,
00013     HAL_CRC_MODEL_CRC8_ITU,
00014     HAL_CRC_MODEL_CRC8_ROHC,
00015     HAL_CRC_MODEL_CRC8_MAXIMx,
00016     HAL_CRC_MODEL_CRC16_IBMx1,
00017     HAL_CRC_MODEL_CRC16_MAXIM,
00018     HAL_CRC_MODEL_CRC16_USBx1,
00019     HAL_CRC_MODEL_CRC16_MODBU,
00020     HAL_CRC_MODEL_CRC16_CCITT,
00021     HAL_CRC_MODEL_CRC16_CCITT_FALSE,
00022     HAL_CRC_MODEL_CRC16_X25,
00023     HAL_CRC_MODEL_CRC16_XMODEM,
00024     HAL_CRC_MODEL_CRC16_DNP,
00025     HAL_CRC_MODEL_CRC32,
00026     HAL_CRC_MODEL_CRC32_MPEG_2,
00027 } HAL_CRC_MODEL_x;
00028 // 初始化
00029 void HAL_CRC_Reset(HAL_CRC_x); // 模块复位
00030 void HAL_CRC_Enable(HAL_CRC_x); // 模块打开
00031 void HAL_CRC_Disable(HAL_CRC_x); // 模块关闭
00032 // 计算
00033 uint32_t HAL_CRC_Calc(HAL_CRC_x, HAL_CRC_MODEL_x, uint8_t *data, uint32_t len); // 计算crc,03使用软件实现,其
    余芯片使用硬件实现
00034 #endif
```

4.19 include/hal_lksmcu_dac.h 文件参考

```
#include "stdint.h"
#include "hal_lksmcu_gpio.h"
```

hal_lksmcu_dac.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



枚举

- enum `HAL_DAC_x` { `HAL_DAC_0` }
- enum `HAL_DAC_RANGE_x` { `HAL_DAC_RANGE_1_2` = 1, `HAL_DAC_RANGE_3_0V` = 0, `HAL_DAC_RANGE_4_85V` = 2 }

函数

- void `HAL_DAC_Reset` (`HAL_DAC_x`)
DAC模块复位
- void `HAL_DAC_Enable` (`HAL_DAC_x`)
DAC模块使能
- void `HAL_DAC_Disable` (`HAL_DAC_x`)
DAC模块失能
- void `HAL_DAC_SetOutVal` (`HAL_DAC_x`, `uint16_t val`)
设置DAC模块输出值
- `uint16_t HAL_DAC_GetOutValMax` (`HAL_DAC_x`)
获取DAC模块输出最大值
- void `HAL_DAC_SetRange` (`HAL_DAC_x`, `HAL_DAC_RANGE_x`)
设置DAC模块输出量程
- float `HAL_DAC_GetRange` (`HAL_DAC_x`)
获取DAC模块输出量程
- void `HAL_DAC_SetOutVologemV` (`HAL_DAC_x`, `uint16_t vol_mV`)

- 设置DAC模块输出电压
- void HAL_DAC_Out2GpioEnable (HAL_DAC_x)
输出到GPIO使能
 - void HAL_DAC_Out2GpioDisable (HAL_DAC_x)
输出到GPIO关闭
 - void HAL_DAC_SetGpioOut (HAL_DAC_x, HAL_GPIO_Px)
设置DAC输出到GPIO

4.19.1 枚举类型说明

4.19.1.1 HAL_DAC_RANGE_x

enum HAL_DAC_RANGE_x

枚举值

HAL_DAC_RANGE_1_2	量程1.2V
HAL_DAC_RANGE_3_0V	量程3.0V
HAL_DAC_RANGE_4_85V	量程4.85V

在文件 [hal_lksmcu_dac.h](#) 第 15 行定义.

4.19.1.2 HAL_DAC_x

enum HAL_DAC_x

枚举值

HAL↔ DAC_0	DAC模块0

在文件 [hal_lksmcu_dac.h](#) 第 6 行定义.

4.19.2 函数说明

4.19.2.1 HAL_DAC_Disable()

```
void HAL_DAC_Disable (
    HAL_DAC_x dac_x )
```

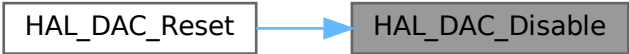
DAC模块失能

参数

dac↔ _x	DAC模块

在文件 [hal_lks32mc08x.dac.c](#) 第 32 行定义.

这是这个函数的调用关系图:



4.19.2.2 HAL_DAC.Enable()

```
void HAL_DAC_Enable (
    HAL_DAC_x dac_x )
```

DAC模块使能

参数

<i>dac_x</i>	DAC模块

在文件 [hal_lks32mc08x.dac.c](#) 第 19 行定义.

4.19.2.3 HAL_DAC.GetOutValMax()

```
uint16_t HAL_DAC_GetOutValMax (
    HAL_DAC_x dac_x )
```

获取DAC模块输出最大值

参数

<i>dac_x</i>	DAC模块

返回

DAC输出最大值

在文件 [hal_lks32mc08x.dac.c](#) 第 46 行定义.

4.19.2.4 HAL_DAC.GetRange()

```
float HAL_DAC_GetRange (
    HAL_DAC_x dac_x )
```


获取DAC模块输出量程

参数

$dac_{\_x}$	DAC模块

返回

输出量程

在文件 [hal_lks32mc08x_dac.c](#) 第 95 行定义.

4.19.2.5 HAL_DAC_Out2GpioDisable()

```
void HAL_DAC_Out2GpioDisable (
    HAL_DAC_x dac_x )
```

输出到GPIO关闭

参数

$dac_{\_x}$	DAC模块

在文件 [hal_lks32mc08x_dac.c](#) 第 181 行定义.

4.19.2.6 HAL_DAC_Out2GpioEnable()

```
void HAL_DAC_Out2GpioEnable (
    HAL_DAC_x dac_x )
```

输出到GPIO使能

参数

$dac_{\_x}$	DAC模块

在文件 [hal_lks32mc08x_dac.c](#) 第 162 行定义.

4.19.2.7 HAL_DAC_Reset()

```
void HAL_DAC_Reset (
    HAL_DAC_x dac_x )
```

DAC模块复位

参数

<i>dac_x</i>	DAC模块

在文件 [hal_lks32mc08x.dac.c](#) 第 11 行定义.

函数调用图:



4.19.2.8 HAL_DAC.SetGpioOut()

```
void HAL_DAC_SetGpioOut (
    HAL_DAC_x dac_x,
    HAL_GPIO_Px px )
```

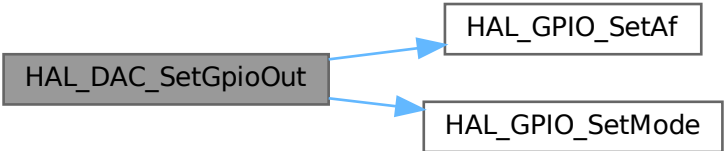
设置DAC输出到GPIO

参数

<i>dac_x</i>	DAC模块
<i>px</i>	GPIO

在文件 [hal_lks32mc08x.dac.c](#) 第 202 行定义.

函数调用图:



4.19.2.9 HAL_DAC_SetOutVal()

```
void HAL_DAC_SetOutVal (
    HAL_DAC_TypeDef dac_x,
    uint16_t val )
```

设置DAC模块输出值

参数

<i>dac_x</i>	DAC模块
<i>val</i>	DAC输出值

在文件 [hal_lks32mc08x.dac.c](#) 第 56 行定义.

这是这个函数的调用关系图:



4.19.2.10 HAL_DAC_SetOutVoltageMv()

```
void HAL_DAC_SetOutVoltageMv (
    HAL_DAC_TypeDef dac_x,
    uint16_t vol_mv )
```

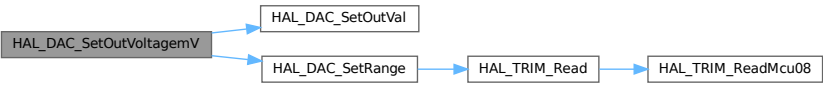
设置DAC模块输出电压

参数

<i>dac_x</i>	DAC模块
<i>vol_mv</i>	输出电压，单位mV

在文件 [hal_lks32mc08x.dac.c](#) 第 120 行定义.

函数调用图:



4.19.2.11 HAL_DAC_SetRange()

```
void HAL_DAC_SetRange (
    HAL_DAC_TypeDef dac_x,
    HAL_DAC_RANGE_TypeDef range )
```

设置DAC模块输出量程

参数

<i>dac_x</i>	DAC模块
<i>range</i>	输出量程

在文件 `hal_lks32mc08x_dac.c` 第 66 行定义.

函数调用图:



这是这个函数的调用关系图:



4.20 hal_lksmcu_dac.h

[浏览该文件的文档.](#)

```
00001 #ifndef __HAL_LKSMCU_DAC_H__
00002 #define __HAL_LKSMCU_DAC_H__
00003 #include "stdint.h"
00004 #include "hal_lksmcu_gpio.h"
00005
00006 typedef enum
00007 {
00008     HAL_DAC_0,
00009 #ifdef HAL_TEST_NOT_SUPPORTED
00010     HAL_DAC_1,
00011     HAL_DAC_2,
00012     HAL_DAC_3,
00013 #endif
00014 } HAL_DAC_TypeDef;
00015 typedef enum
00016 {
00017     HAL_DAC_RANGE_1_2 = 1,
00018     HAL_DAC_RANGE_3_0V = 0,
```

```

00019     HAL_DAC_RANGE_4.85V = 2,
00020 #ifdef HAL_TEST_NOT_SUPPORTED
00021     HAL_DAC_RANGE_4.76V,
00022     HAL_DAC_RANGE_4.8V,
00023 #endif
00024 } HAL_DAC_RANGE_x;
00025 void HAL_DAC_Reset(HAL_DAC_x); // 模块复位
00026 void HAL_DAC_Enable(HAL_DAC_x); // 模块打开
00027 void HAL_DAC_Disable(HAL_DAC_x); // 模块关闭
00028
00029 void HAL_DAC_SetOutVal(HAL_DAC_x, uint16_t val);
00030 uint16_t HAL_DAC_GetOutValMax(HAL_DAC_x);
00031 void HAL_DAC_SetRange(HAL_DAC_x, HAL_DAC_RANGE_x);
00032 float HAL_DAC_GetRange(HAL_DAC_x);
00033
00034 void HAL_DAC_SetOutVoltageMv(HAL_DAC_x, uint16_t vol_mv);
00035
00036 void HAL_DAC_Out2GpioEnable(HAL_DAC_x);
00037 void HAL_DAC_Out2GpioDisable(HAL_DAC_x);
00038 void HAL_DAC_SetGpioOut(HAL_DAC_x, HAL_GPIO_Px);
00039 #endif

```

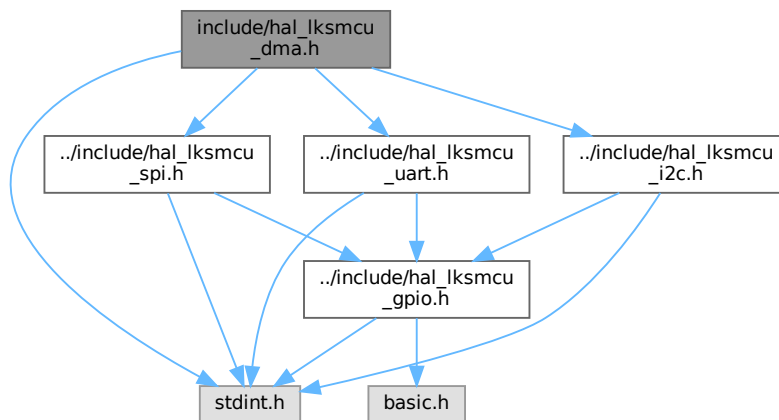
4.21 include/hal_lksmcu_dma.h 文件参考

```

#include "../include/hal_lksmcu_spi.h"
#include "../include/hal_lksmcu_uart.h"
#include "../include/hal_lksmcu_i2c.h"
#include "stdint.h"

```

hal_lksmcu_dma.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



枚举

- enum HAL_DMA_x { HAL_DMA_0, HAL_DMA_1, HAL_DMA_2, HAL_DMA_3 }

函数

- void [HAL_DMA_Reset](#) ([HAL_DMA_x](#))
- void [HAL_DMA_Enable](#) ([HAL_DMA_x](#))
- void [HAL_DMA_Disable](#) ([HAL_DMA_x](#))
- void [HAL_DMA_MemCopy](#) ([HAL_DMA_x](#), void *start, void *end, uint32_t len)
- void [HAL_DMA_UartTx](#) ([HAL_DMA_x](#), [HAL_UART_x](#), uint8_t *start, uint32_t len)
- void [HAL_DMA_SpiTx](#) ([HAL_DMA_x](#), [HAL_SPI_x](#), uint8_t *start, uint32_t len)
- void [HAL_DMA_I2cTx](#) ([HAL_DMA_x](#), [HAL_I2C_x](#), uint8_t *start, uint32_t len)
- void [HAL_DMA_UartRx](#) ([HAL_DMA_x](#), [HAL_UART_x](#), uint8_t *start, uint32_t len)
- void [HAL_DMA_SpiRx](#) ([HAL_DMA_x](#), [HAL_SPI_x](#), uint8_t *start, uint32_t len)
- void [HAL_DMA_I2cRx](#) ([HAL_DMA_x](#), [HAL_I2C_x](#), uint8_t *start, uint32_t len)
- void [HAL_DMA_SpiRxBx](#) ([HAL_DMA_x](#), [HAL_DMA_x](#), [HAL_SPI_x](#), uint8_t *txdata, uint8_t *rxdata, uint32_t len)

4.21.1 枚举类型说明

4.21.1.1 HAL_DMA_x

enum [HAL_DMA_x](#)

枚举值

HAL_DMA_0	DMA模块0
HAL_DMA_1	DMA模块1
HAL_DMA_2	DMA模块2
HAL_DMA_3	DMA模块3

在文件 [hal_lksmcu_dma.h](#) 第 8 行定义.

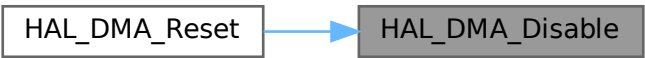
4.21.2 函数说明

4.21.2.1 HAL_DMA_Disable()

```
void HAL_DMA_Disable (  
    HAL\_DMA\_x DMA_x )
```

在文件 [hal_lks32mc08x_dma.c](#) 第 21 行定义.

这是这个函数的调用关系图:



4.21.2.2 HAL_DMA_Enable()

```
void HAL_DMA_Enable (
    HAL_DMA_x DMA_x )
```

在文件 [hal_lks32mc08x_dma.c](#) 第 14 行定义.

4.21.2.3 HAL_DMA_I2cRx()

```
void HAL_DMA_I2cRx (
    HAL_DMA_x DMA_x,
    HAL_I2C_x I2Cx,
    uint8_t * start,
    uint32_t len )
```

在文件 [hal_lks32mc08x_dma.c](#) 第 115 行定义.

4.21.2.4 HAL_DMA_I2cTx()

```
void HAL_DMA_I2cTx (
    HAL_DMA_x DMA_x,
    HAL_I2C_x I2Cx,
    uint8_t * start,
    uint32_t len )
```

在文件 [hal_lks32mc08x_dma.c](#) 第 72 行定义.

4.21.2.5 HAL_DMA_MemCopy()

```
void HAL_DMA_MemCopy (
    HAL_DMA_x DMA_x,
    void * start,
    void * end,
    uint32_t len )
```

在文件 [hal_lks32mc08x_dma.c](#) 第 27 行定义.

4.21.2.6 HAL_DMA_Reset()

```
void HAL_DMA_Reset (
    HAL_DMA_x DMA_x )
```

在文件 [hal_lks32mc08x_dma.c](#) 第 9 行定义.

函数调用图:



4.21.2.7 HAL_DMA_SpiRx()

```
void HAL_DMA_SpiRx (
    HAL_DMA_x DMA_x,
    HAL_SPI_x SPI_x,
    uint8_t * start,
    uint32_t len )
```

在文件 [hal_lks32mc08x_dma.c](#) 第 105 行定义.

这是这个函数的调用关系图:

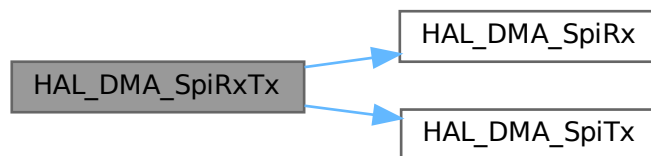


4.21.2.8 HAL_DMA_SpiRxTx()

```
void HAL_DMA_SpiRxTx (
    HAL_DMA_x DMA1_x,
    HAL_DMA_x DMA2_x,
    HAL_SPI_x SPI_x,
    uint8_t * txdata,
    uint8_t * rxdata,
    uint32_t len )
```

在文件 [hal_lks32mc08x_dma.c](#) 第 125 行定义.

函数调用图:



4.21.2.9 HAL_DMA_SpiTx()

```
void HAL_DMA_SpiTx (
    HAL_DMA_x DMA_x,
    HAL_SPI_x SPI_x,
    uint8_t * start,
    uint32_t len )
```

在文件 [hal_lks32mc08x_dma.c](#) 第 60 行定义.

这是这个函数的调用关系图:



4.21.2.10 HAL_DMA_UartRx()

```
void HAL_DMA_UartRx (
    HAL_DMA_x DMA_x,
    HAL_UART_x UART_x,
    uint8_t * start,
    uint32_t len )
```

在文件 [hal_lks32mc08x_dma.c](#) 第 84 行定义.

4.21.2.11 HAL_DMA_UartTx()

```
void HAL_DMA_UartTx (
    HAL_DMA_x DMA_x,
    HAL_UART_x UART_x,
    uint8_t * start,
    uint32_t len )
```

在文件 [hal_lks32mc08x_dma.c](#) 第 37 行定义.

4.22 hal_lksmcu_dma.h

[浏览该文件的文档.](#)

```
00001 #ifndef __HAL_LKSMCU_DMA_H__
00002 #define __HAL_LKSMCU_DMA_H__
00003
00004 #include "../include/hal_lksmcu_spi.h"
00005 #include "../include/hal_lksmcu_uart.h"
00006 #include "../include/hal_lksmcu_i2c.h"
00007 #include "stdint.h"
00008 typedef enum
00009 {
```

```

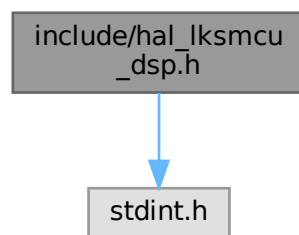
00010     HAL_DMA_0,
00011     HAL_DMA_1,
00012     HAL_DMA_2,
00013     HAL_DMA_3,
00014
00015 #ifndef HAL_TEST_NOT_SUPPORTED
00016     HAL_DMA_4,
00017     HAL_DMA_5,
00018     HAL_DMA_6,
00019     HAL_DMA_7,
00020 #endif
00021 } HAL_DMA_x;
00022
00023 // 初始化
00024 void HAL_DMA_Reset(HAL_DMA_x); // 模块复位
00025 void HAL_DMA_Enable(HAL_DMA_x); // 模块打开
00026 void HAL_DMA_Disable(HAL_DMA_x); // 模块关闭
00027
00028 void HAL_DMA_MemCopy(HAL_DMA_x, void *start, void *end, uint32_t len); // 内存复制
00029
00030 void HAL_DMA_UartTx(HAL_DMA_x, HAL_UART_x, uint8_t *start, uint32_t len); // dma使用uart发送数据,需要先初始化
好对应的uart模块
00031 void HAL_DMA_SpiTx(HAL_DMA_x, HAL_SPI_x, uint8_t *start, uint32_t len); // dma使用spi发送数据,需要先初始化
好对应的spi模块
00032 void HAL_DMA_I2cTx(HAL_DMA_x, HAL_I2C_x, uint8_t *start, uint32_t len); // dma使用i2c发送数据,需要先初始化
好对应的i2c模块
00033 void HAL_DMA_UartRx(HAL_DMA_x, HAL_UART_x, uint8_t *start, uint32_t len); // dma使用uart接收数据,需要先初始化
好对应的uart模块
00034 void HAL_DMA_SpiRx(HAL_DMA_x, HAL_SPI_x, uint8_t *start, uint32_t len); // dma使用spi接收数据,需要先初始化
好对应的spi模块
00035 void HAL_DMA_I2cRx(HAL_DMA_x, HAL_I2C_x, uint8_t *start, uint32_t len); // dma使用i2c接收数据,需要先初始化
好对应的i2c模块
00036
00037 void HAL_DMA_SpiRxTx(HAL_DMA_x, HAL_DMA_x, HAL_SPI_x, uint8_t *txdata, uint8_t *rxdata, uint32_t len); //
使用两个dma模块和一个spi模块,同时发送&接收数据,需要先初始化好对应的spi模块
00038
00039 #endif

```

4.23 include/hal_lksmcu_dsp.h 文件参考

```
#include "stdint.h"
```

hal_lksmcu_dsp.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



枚举

- enum HAL_DSP_x { HAL_DSP_0 }

函数

- void HAL_DSP_Reset (HAL_DSP_x)
DSP复位
- void HAL_DSP_Enable (HAL_DSP_x)
DSP使能
- void HAL_DSP_Disable (HAL_DSP_x)
DSP复位
- int16_t HAL_DSP_Sin (HAL_DSP_x, int16_t angle)
DSP正弦计算
- int16_t HAL_DSP_Cos (HAL_DSP_x, int16_t angle)
DSP余弦计算
- void HAL_DSP_SinCos (HAL_DSP_x, int16_t angle, int16_t *sin, int16_t *cos)
DSP正弦余弦计算
- uint16_t HAL_DSP_Sqrt (HAL_DSP_x, uint32_t)
DSP开方
- int32_t HAL_DSP_Div (HAL_DSP_x, int32_t, int32_t)
DSP除法

4.23.1 枚举类型说明

4.23.1.1 HAL_DSP_x

enum HAL_DSP_x

枚举值

HAL↔ DSP_0	DSP模块0

在文件 hal_lksmcu_dsp.h 第 5 行定义.

4.23.2 函数说明

4.23.2.1 HAL_DSP_Cos()

```
int16_t HALDSP_Cos (
    HAL_DSP_x dsp_x,
    int16_t angle )
```

DSP余弦计算

参数

$dsp_{\leftarrow x}$	DSP选择
$angle$	角度

返回值

余弦结果	

在文件 [hal_lks32mc08x.dsp.c](#) 第 70 行定义.

4.23.2.2 HAL_DSP_Disable()

```
void HAL_DSP_Disable (
    HAL_DSP_x dsp_x )
```

DSP复位

参数

$dsp_{\leftarrow x}$	DSP选择

在文件 [hal_lks32mc08x.dsp.c](#) 第 40 行定义.

4.23.2.3 HAL_DSP_Div()

```
int32_t HAL_DSP_Div (
    HAL_DSP_x dsp_x,
    int32_t a,
    int32_t b )
```

DSP除法

参数

$dsp_{\leftarrow x}$	DSP选择
a	被除数
b	除数

返回值

$\hat{a} \cdot t$	

在文件 [hal_lks32mc08x.dsp.c](#) 第 113 行定义.

4.23.2.4 HAL_DSP_Enable()

```
void HAL_DSP_Enable (
    HAL_DSP_x dsp_x )
```

DSP使能

参数

<i>dsp_x</i>	DSP选择

在文件 [hal_lks32mc08x.dsp.c](#) 第 26 行定义.

4.23.2.5 HAL_DSP_Reset()

```
void HAL_DSP_Reset (
    HAL_DSP_x dsp_x )
```

DSP复位

参数

<i>dsp_x</i>	DSP选择

在文件 [hal_lks32mc08x.dsp.c](#) 第 11 行定义.

4.23.2.6 HAL_DSP_Sin()

```
int16_t HAL_DSP_Sin (
    HAL_DSP_x dsp_x,
    int16_t angle )
```

DSP正弦计算

参数

<i>dsp_x</i>	DSP选择
<i>angle</i>	角度

返回值

正弦结果	

在文件 [hal_lks32mc08x.dsp.c](#) 第 56 行定义.

4.23.2.7 HAL_DSP_SinCos()

```
void HAL_DSP_SinCos (
    HAL_DSP_x dsp_x,
    int16_t angle,
    int16_t * sin,
    int16_t * cos )
```

DSP正弦余弦计算

参数

<i>dsp_x</i>	DSP选择
<i>angle</i>	角度
<i>sin</i>	正弦结果
<i>cos</i>	余弦结果

在文件 [hal_lks32mc08x.dsp.c](#) 第 85 行定义.

4.23.2.8 HAL_DSP_Sqrt()

```
uint16_t HAL_DSP_Sqrt (
    HAL_DSP_x dsp_x,
    uint32_t x )
```

DSP开方

参数

<i>dsp_x</i>	DSP选择
<i>x</i>	输入

返回值

开方结果	

在文件 [hal_lks32mc08x.dsp.c](#) 第 99 行定义.

4.24 hal_lksmcu_dsp.h

[浏览该文件的文档.](#)

```
00001 #ifndef __HAL_LKSMCU_DSP_H__
00002 #define __HAL_LKSMCU_DSP_H__
00003 #include "stdint.h"
00004
00005 typedef enum
00006 {
```

```

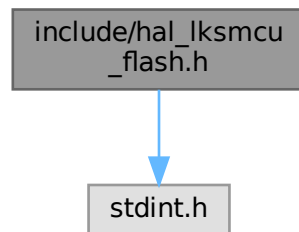
00007     HAL_DSP_0,
00008 } HAL_DSP_x;
00009 void HAL_DSP_Reset (HAL_DSP_x); // 模块复位
00010 void HAL_DSP_Enable (HAL_DSP_x); // 模块打开
00011 void HAL_DSP_Disable (HAL_DSP_x); // 模块关闭
00012
00013 int16_t HAL_DSP_Sin (HAL_DSP_x, int16_t angle);
00014 int16_t HAL_DSP_Cos (HAL_DSP_x, int16_t angle);
00015 void HAL_DSP_SinCos (HAL_DSP_x, int16_t angle, int16_t *sin, int16_t *cos);
00016 uint16_t HAL_DSP_Sqrt (HAL_DSP_x, uint32_t);
00017 int32_t HAL_DSP_Div (HAL_DSP_x, int32_t, int32_t);
00018 #endif

```

4.25 include/hal_lksmcu_flash.h 文件参考

#include "stdint.h"

hal_lksmcu_flash.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



枚举

- enum HAL_FLASH_x { HAL_FLASH_0 , HAL_FLASH_1 }

函数

- void HAL_FLASH_Reset (HAL_FLASH_x)
模块复位,预留接口
- void HAL_FLASH_Enable (HAL_FLASH_x)
模块打开,预留接口
- void HAL_FLASH_Disable (HAL_FLASH_x)
模块关闭,预留接口
- void HAL_FLASH_EnableCache (HAL_FLASH_x)
使能flash缓存加速

- void [HAL_FLASH_DisableCache \(HAL_FLASH_x\)](#)
关闭flash缓存加速
- uint32_t [HAL_FLASH_ReadSectorSize \(HAL_FLASH_x\)](#)
获取一个扇区的大小,和HAL_FLASH_SECTOR_SIZE等效
- void [HAL_FLASH_Read \(HAL_FLASH_x, uint32_t flashaddr, void *data, uint32_t len\)](#)
读FLASH取数据
- uint8_t [HAL_FLASH_ReadUint8 \(HAL_FLASH_x, uint32_t flashaddr\)](#)
读FLASH取数据
- uint16_t [HAL_FLASH_ReadUint16 \(HAL_FLASH_x, uint32_t flashaddr\)](#)
读FLASH取数据
- uint32_t [HAL_FLASH_ReadUint32 \(HAL_FLASH_x, uint32_t flashaddr\)](#)
读FLASH取数据
- void [HAL_FLASH_Erase \(HAL_FLASH_x, uint32_t flashaddr, uint32_t len\)](#)
扇区擦除,len只能是扇区大小的整数倍,不够一个扇区的部分会被忽略
- void [HAL_FLASH_Program \(HAL_FLASH_x, uint32_t flashaddr, uint8_t *data, uint32_t len\)](#)
编程
- void [HAL_FLASH_Update \(HAL_FLASH_x, uint32_t flashaddr, uint8_t *data, uint32_t len\)](#)
更新数据,等效先擦除再编程
- void [HAL_FLASH_SecurityParameterSave \(HAL_FLASH_x, void *defaultdata, void *data, uint32_t len\)](#)
保存安全参数
- uint32_t [HAL_FLASH_SecurityParameterRead \(HAL_FLASH_x, void *defaultdata, void *data, uint32_t len\)](#)
读取安全参数

4.25.1 枚举类型说明

4.25.1.1 HAL_FLASH_x

enum [HAL_FLASH_x](#)

枚举值

HAL_↔ FLASH_0	内部flash
HAL_↔ FLASH_1	外部flash

在文件 [hal_lksmcu_flash.h](#) 第 5 行定义.

4.25.2 函数说明

4.25.2.1 HAL_FLASH_Disable()

```
void HAL_FLASH_Disable (
    HAL\_FLASH\_x flash )
```

模块关闭,预留接口

参数

<i>flash</i>	模块类型

在文件 [hal_lks32mc08x.flash.c](#) 第 30 行定义.

4.25.2.2 HAL_FLASH_DisableCache()

```
void HAL_FLASH_DisableCache (
    HAL_FLASH_x_t flash )
```

关闭flash缓存加速

参数

<i>flash</i>	模块类型

在文件 [hal_lks32mc08x.flash.c](#) 第 49 行定义.

4.25.2.3 HAL_FLASH_Enable()

```
void HAL_FLASH_Enable (
    HAL_FLASH_x_t flash )
```

模块打开,预留接口

参数

<i>flash</i>	模块类型

在文件 [hal_lks32mc08x.flash.c](#) 第 21 行定义.

4.25.2.4 HAL_FLASH_EnableCache()

```
void HAL_FLASH_EnableCache (
    HAL_FLASH_x_t flash )
```

使能flash缓存加速

参数

<i>flash</i>	模块类型

在文件 [hal_lks32mc08x.flash.c](#) 第 39 行定义.

这是这个函数的调用关系图:



4.25.2.5 HAL_FLASH_Erase()

```

void HAL_FLASH_Erase (
    HAL_FLASH_X flash,
    uint32_t flashaddr,
    uint32_t len )
  
```

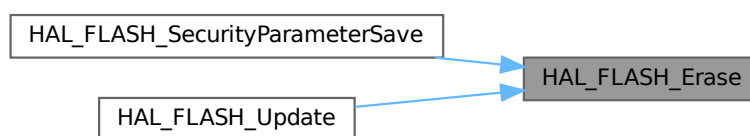
扇区擦除, *len* 只能是扇区大小的整数倍, 不够一个扇区的部分会被忽略

参数

<i>flash</i>	模块类型
<i>flashaddr</i>	flash地址
<i>len</i>	擦除长度

在文件 [hal_lks32mc08x_flash.c](#) 第 124 行定义.

这是这个函数的调用关系图:



4.25.2.6 HAL_FLASH_Program()

```

void HAL_FLASH_Program (
    HAL_FLASH_X flash,
    uint32_t flashaddr,
    uint8_t * data,
    uint32_t len )
  
```

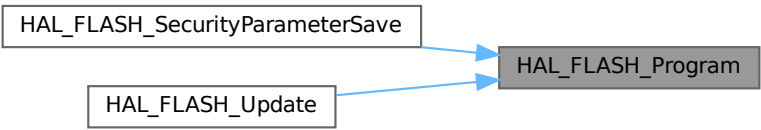
编程

参数

<i>flash</i>	模块类型
<i>flashaddr</i>	flash地址,低两位会被忽略
<i>data</i>	数据指针
<i>len</i>	数据长度

在文件 [hal_lks32mc08x.flash.c](#) 第 146 行定义.

这是这个函数的调用关系图:



4.25.2.7 HAL_FLASH_Read()

```
void HAL_FLASH_Read (
    HAL_FLASH_TypeDef * flash,
    uint32_t flashaddr,
    void * data,
    uint32_t len )
```

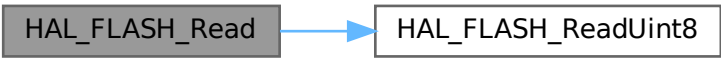
读FLASH取数据

参数

<i>flash</i>	模块类型
<i>flashaddr</i>	flash地址
<i>data</i>	读取的数据指针
<i>len</i>	读取长度

在文件 [hal_lks32mc08x.flash.c](#) 第 73 行定义.

函数调用图:



这是这个函数的调用关系图:



4.25.2.8 HAL_FLASH_ReadSectorSize()

```
uint32_t HAL_FLASH_ReadSectorSize (
    HAL_FLASH_x flash )
```

获取一个扇区的大小,和HAL_FLASH_SECTOR_SIZE等效

参数

flash	模块类型

返回

uint32_t 扇区大小

在文件 [hal_lks32mc08x_flash.c](#) 第 60 行定义.

4.25.2.9 HAL_FLASH_ReadUint16()

```
uint16_t HAL_FLASH_ReadUint16 (
    HAL_FLASH_x flash,
    uint32_t flashaddr )
```

读FLASH取数据

参数

flash	模块类型
flashaddr	flash地址

返回

读取的数据

在文件 [hal_lks32mc08x.flash.c](#) 第 100 行定义.

4.25.2.10 HAL_FLASH_ReadUint32()

```
uint32_t HAL_FLASH_ReadUint32 (
    HAL_FLASH_TypeDef * flash,
    uint32_t flashaddr )
```

读FLASH取数据

参数

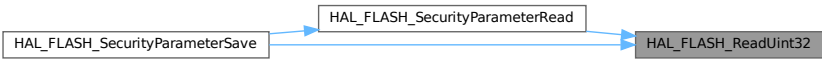
<i>flash</i>	模块类型
<i>flashaddr</i>	flash地址

返回

读取的数据

在文件 [hal_lks32mc08x.flash.c](#) 第 112 行定义.

这是这个函数的调用关系图:



4.25.2.11 HAL_FLASH_ReadUint8()

```
uint8_t HAL_FLASH_ReadUint8 (
    HAL_FLASH_TypeDef * flash,
    uint32_t flashaddr )
```

读FLASH取数据

参数

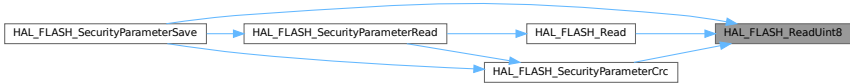
<i>flash</i>	模块类型
<i>flashaddr</i>	flash地址

返回

读取的数据

在文件 [hal_lks32mc08x.flash.c](#) 第 88 行定义.

这是这个函数的调用关系图:



4.25.2.12 HAL_FLASH.Reset()

```
void HAL_FLASH.Reset (
    HAL_FLASH_x flash )
```

模块复位,预留接口

参数

<i>flash</i>	模块类型

在文件 [hal_lks32mc08x.flash.c](#) 第 12 行定义.

4.25.2.13 HAL_FLASH.SecurityParameterRead()

```
uint32_t HAL_FLASH.SecurityParameterRead (
    HAL_FLASH_x flash,
    void * defaultdata,
    void * data,
    uint32_t len )
```

读取安全参数

参数

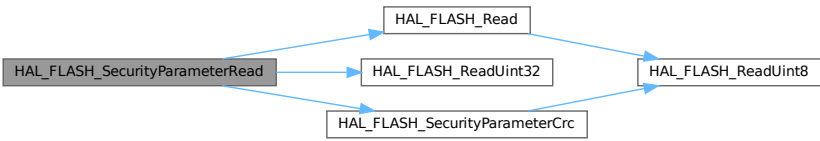
<i>flash</i>	模块类型
<i>data</i>	数据指针
<i>len</i>	数据长度

返回

使用的flash地址

在文件 [hal_lks32mc08x.flash.c](#) 第 310 行定义.

函数调用图:



这是这个函数的调用关系图:



4.25.2.14 HAL_FLASH_SecurityParameterSave()

```
void HAL_FLASH_SecurityParameterSave (
    HAL_FLASH_TypeDef * flash,
    void * defaultdata,
    void * data,
    uint32_t len )
```

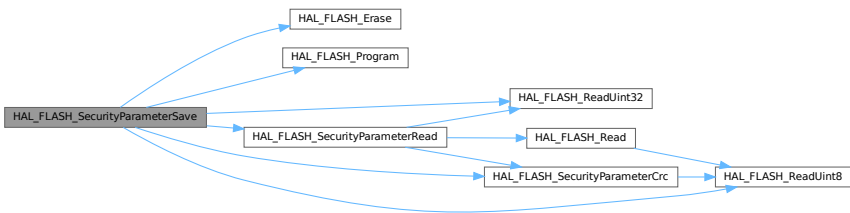
保存安全参数

参数

<i>flash</i>	模块类型
<i>defaultdata</i>	默认值
<i>data</i>	待写入的数据
<i>len</i>	数据长度

在文件 [hal_lks32mc08x_flash.c](#) 第 216 行定义.

函数调用图:



4.25.2.15 HAL_FLASH_Update()

```
void HAL_FLASH_Update (
    HAL_FLASH_x flash,
    uint32_t flashaddr,
    uint8_t * data,
    uint32_t len )
```

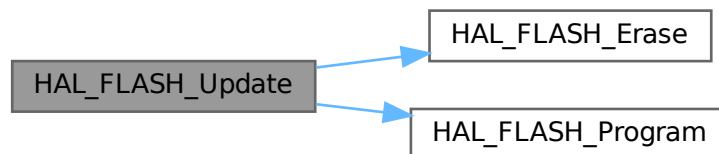
更新数据,等效先擦除再编程

参数

<i>flash</i>	模块类型
<i>flashaddr</i>	flash地址
<i>data</i>	数据指针
<i>len</i>	数据长度

在文件 [hal_lks32mc08x_flash.c](#) 第 172 行定义.

函数调用图:



4.26 hal_lksmcu_flash.h

[浏览该文件的文档.](#)

```
00001 #ifndef __HAL_LKSMCU_FLASH_H__
00002 #define __HAL_LKSMCU_FLASH_H__
00003
00004 #include "stdint.h"
00005 typedef enum
00006 {
00007     HAL_FLASH_0,
00008     HAL_FLASH_1,
00009 } HAL_FLASH_x;
00010 // 初始化
00011 void HAL_FLASH_Reset (HAL_FLASH_x); // 模块复位,预留接口
00012 void HAL_FLASH_Enable (HAL_FLASH_x); // 模块打开,预留接口
00013 void HAL_FLASH_Disable (HAL_FLASH_x); // 模块关闭,预留接口
00014
00015 void HAL_FLASH_EnableCache (HAL_FLASH_x); // 使能flash缓存加
    速
00016 void HAL_FLASH_DisableCache (HAL_FLASH_x); // 关闭flash缓存加
    速
00017 uint32_t HAL_FLASH_ReadSectorSize (HAL_FLASH_x); // 获取一个扇区的大
    小,和HAL_FLASH_SECTOR_SIZE等效
00018 void HAL_FLASH_Read (HAL_FLASH_x, uint32_t flashaddr, void *data, uint32_t len); // 任意长度
00019 uint8_t HAL_FLASH_ReadUint8 (HAL_FLASH_x, uint32_t flashaddr); // 8~(U8)
00020 uint16_t HAL_FLASH_ReadUint16 (HAL_FLASH_x, uint32_t flashaddr); // 16~(U16)
```



```

00021 uint32_t HAL_FLASH_ReadUint32(HAL_FLASH_x, uint32_t flashaddr); // 读(U32)
00022 void HAL_FLASH_Erase(HAL_FLASH_x, uint32_t flashaddr, uint32_t len); // 扇区擦除, len只能是
    扇区大小的整数倍, 不够一个扇区的部分会被忽略
00023 void HAL_FLASH_Program(HAL_FLASH_x, uint32_t flashaddr, uint8_t *data, uint32_t len); // 编程
00024 void HAL_FLASH_Update(HAL_FLASH_x, uint32_t flashaddr, uint8_t *data, uint32_t len); // 更新数据, 等效先擦除
    再编程
00025
00026 // 安全参数 (调用时自动分配地址)
00027 // 支持大小为 500的uint8_t的数据
00028 // 保留3个word用来校验
00029 // 状态+CRC32+保留位
00030 // 调用时必须传入defaultdata (确保极端情况下, data的数据安全)
00031 void HAL_FLASH_SecurityParameterSave(HAL_FLASH_x, void *defaultdata, void *data, uint32_t len); //
    保存数据 (仅变化时保存, 最多可能会关闭中断20ms)
00032 uint32_t HAL_FLASH_SecurityParameterRead(HAL_FLASH_x, void *defaultdata, void *data, uint32_t len); // 读
    数据
00033
00034 #endif

```

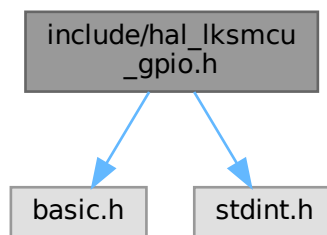
4.27 include/hal_lksmcu_gpio.h 文件参考

```

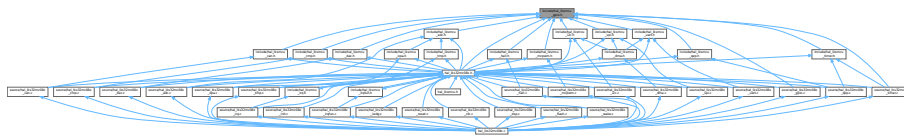
#include "basic.h"
#include "stdint.h"

```

hal_lksmcu_gpio.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



枚举

- enum HAL_GPIO_Px {
 HAL_GPIO_P0_0 = 0x0000, HAL_GPIO_P0_1 = 0x0001, HAL_GPIO_P0_2 = 0x0002, HAL_GPIO_P0_3 = 0x0003,
 HAL_GPIO_P0_4 = 0x0004, HAL_GPIO_P0_5 = 0x0005, HAL_GPIO_P0_6 = 0x0006, HAL_GPIO_P0_7 = 0x0007,
 HAL_GPIO_P0_8 = 0x0008, HAL_GPIO_P0_9 = 0x0009, HAL_GPIO_P0_10 = 0x000a, HAL_GPIO_P0_11 = 0x000b,
 HAL_GPIO_P0_12 = 0x000c, HAL_GPIO_P0_13 = 0x000d, HAL_GPIO_P0_14 = 0x000e, HAL_GPIO_P0_15 =

```

0x000f ,
HAL_GPIO_P1_0 = 0x0100 , HAL_GPIO_P1_1 = 0x0101 , HAL_GPIO_P1_2 = 0x0102 , HAL_GPIO_P1_3 =
0x0103 ,
HAL_GPIO_P1_4 = 0x0104 , HAL_GPIO_P1_5 = 0x0105 , HAL_GPIO_P1_6 = 0x0106 , HAL_GPIO_P1_7 =
0x0107 ,
HAL_GPIO_P1_8 = 0x0108 , HAL_GPIO_P1_9 = 0x0109 , HAL_GPIO_P1_10 = 0x010a , HAL_GPIO_P1_11 =
0x010b ,
HAL_GPIO_P1_12 = 0x010c , HAL_GPIO_P1_13 = 0x010d , HAL_GPIO_P1_14 = 0x010e , HAL_GPIO_P1_15 =
0x010f ,
HAL_GPIO_P2_0 = 0x0200 , HAL_GPIO_P2_1 = 0x0201 , HAL_GPIO_P2_2 = 0x0202 , HAL_GPIO_P2_3 =
0x0203 ,
HAL_GPIO_P2_4 = 0x0204 , HAL_GPIO_P2_5 = 0x0205 , HAL_GPIO_P2_6 = 0x0206 , HAL_GPIO_P2_7 =
0x0207 ,
HAL_GPIO_P2_8 = 0x0208 , HAL_GPIO_P2_9 = 0x0209 , HAL_GPIO_P2_10 = 0x020a , HAL_GPIO_P2_11 =
0x020b ,
HAL_GPIO_P2_12 = 0x020c , HAL_GPIO_P2_13 = 0x020d , HAL_GPIO_P2_14 = 0x020e , HAL_GPIO_P2_15 =
0x020f ,
HAL_GPIO_P3_0 = 0x0300 , HAL_GPIO_P3_1 = 0x0301 , HAL_GPIO_P3_2 = 0x0302 , HAL_GPIO_P3_3 =
0x0303 ,
HAL_GPIO_P3_4 = 0x0304 , HAL_GPIO_P3_5 = 0x0305 , HAL_GPIO_P3_6 = 0x0306 , HAL_GPIO_P3_7 =
0x0307 ,
HAL_GPIO_P3_8 = 0x0308 , HAL_GPIO_P3_9 = 0x0309 , HAL_GPIO_P3_10 = 0x030a , HAL_GPIO_P3_11 =
0x030b ,
HAL_GPIO_P3_12 = 0x030c , HAL_GPIO_P3_13 = 0x030d , HAL_GPIO_P3_14 = 0x030e , HAL_GPIO_P3_15 =
0x030f ,
HAL_GPIO_P4_0 = 0x0400 , HAL_GPIO_P4_1 = 0x0401 , HAL_GPIO_P4_2 = 0x0402 , HAL_GPIO_P4_3 =
0x0403 ,
HAL_GPIO_P4_4 = 0x0404 , HAL_GPIO_P4_5 = 0x0405 , HAL_GPIO_P4_6 = 0x0406 , HAL_GPIO_P4_7 =
0x0407 ,
HAL_GPIO_P4_8 = 0x0408 , HAL_GPIO_P4_9 = 0x0409 , HAL_GPIO_P4_10 = 0x040a , HAL_GPIO_P4_11 =
0x040b ,
HAL_GPIO_P4_12 = 0x040c , HAL_GPIO_P4_13 = 0x040d , HAL_GPIO_P4_14 = 0x040e , HAL_GPIO_P4_15 =
0x040f ,
HAL_GPIO_P5_0 = 0x0500 , HAL_GPIO_P5_1 = 0x0501 , HAL_GPIO_P5_2 = 0x0502 , HAL_GPIO_P5_3 =
0x0503 ,
HAL_GPIO_P5_4 = 0x0504 , HAL_GPIO_P5_5 = 0x0505 , HAL_GPIO_P5_6 = 0x0506 , HAL_GPIO_P5_7 =
0x0507 ,
HAL_GPIO_P5_8 = 0x0508 , HAL_GPIO_P5_9 = 0x0509 , HAL_GPIO_P5_10 = 0x050a , HAL_GPIO_P5_11 =
0x050b ,
HAL_GPIO_P5_12 = 0x050c , HAL_GPIO_P5_13 = 0x050d , HAL_GPIO_P5_14 = 0x050e , HAL_GPIO_P5_15 =
0x050f }

```

GPIO口定义

- enum HAL_GPIO_AFx {
 HAL_GPIO_AF_ADC = 0 , HAL_GPIO_AF_OPA = 0 , HAL_GPIO_AF_OPA_OUT = 0 , HAL_GPIO_AF_CMP = 0 ,
 HAL_GPIO_AF_DAC = 0 , HAL_GPIO_AF_GPIO = 0 , HAL_GPIO_AF_CLK = 0 , HAL_GPIO_AF_CMP_OUT = 1
 ,
 HAL_GPIO_AF_CLK_OUT = 1 , HAL_GPIO_AF_HALL = 2 , HAL_GPIO_AF_MCPWM = 3 , HAL_GPIO_AF_UART
 = 4 ,
 HAL_GPIO_AF_SPI = 5 , HAL_GPIO_AF_I2C = 6 , HAL_GPIO_AF_TIMER0 = 7 , HAL_GPIO_AF_TIMER1 = 7 ,
 HAL_GPIO_AF_TIMER2 = 8 , HAL_GPIO_AF_TIMER3 = 8 , HAL_GPIO_AF_QEP0 = 8 , HAL_GPIO_AF_QEP1
 = 8 ,
 HAL_GPIO_AF_ADC_TRIG = 9 , HAL_GPIO_AF_CAN = 10 , HAL_GPIO_AF_SIF = 11 }
- enum HAL_GPIO_MODE_x {
 HAL_GPIO_MODE_IN_EN = BIT0 , HAL_GPIO_MODE_OUT_EN = BIT1 , HAL_GPIO_MODE_UP_EN = BIT2 ,
 HAL_GPIO_MODE_PODE_EN = BIT3 ,
 HAL_GPIO_MODE_DEFAULT_0 = BIT4 , HAL_GPIO_MODE_DEFAULT_1 = BIT5 , HAL_GPIO_MODE_AFE = 0
 , HAL_GPIO_MODE_NONE = HAL_GPIO_MODE_AFE ,
 HAL_GPIO_MODE_IN = HAL_GPIO_MODE_IN_EN , HAL_GPIO_MODE_INUP = HAL_GPIO_MODE_IN | HAL_GPIO_MODE_INUP

```

_GPIO_MODE_UP_EN , HAL_GPIO_MODE_OUT = HAL_GPIO_MODE_OUT_EN , HAL_GPIO_MODE_INOUT
= HAL_GPIO_MODE_IN | HAL_GPIO_MODE_OUT | HAL_GPIO_MODE_PODE_EN ,
HAL_GPIO_MODE_INUPOUT = HAL_GPIO_MODE_INOUT | HAL_GPIO_MODE_UP_EN | HAL_GPIO_
_MODE_PODE_EN , HAL_GPIO_MODE_OUT_0 = HAL_GPIO_MODE_OUT | HAL_GPIO_MODE_
DEFAULT_0 , HAL_GPIO_MODE_INOUT_0 = HAL_GPIO_MODE_INOUT | HAL_GPIO_MODE_DEFAULT_0 ,
HAL_GPIO_MODE_INUPOUT_0 = HAL_GPIO_MODE_INUPOUT | HAL_GPIO_MODE_DEFAULT_0 ,
HAL_GPIO_MODE_OUT_1 = HAL_GPIO_MODE_OUT | HAL_GPIO_MODE_DEFAULT_1 , HAL_GPIO_MODE_INOUT_1
= HAL_GPIO_MODE_INOUT | HAL_GPIO_MODE_DEFAULT_1 , HAL_GPIO_MODE_INUPOUT_1 = HAL_
_GPIO_MODE_INUPOUT | HAL_GPIO_MODE_DEFAULT_1 }

```

*GPIO*工作模式定义

函数

- void **HAL_GPIO_Reset** (void)
模块复位,执行后恢复所有的*GPIO*为悬空状态(*SWD*、*JTAG*和*RST*口为默认状态)
- void **HAL_GPIO_Enable** (void)
启用*GPIO*模块,使能*GPIO*时钟和必要的资源
- void **HAL_GPIO_Disable** (void)
禁用*GPIO*模块,关闭*GPIO*时钟和释放资源
- void **HAL_GPIO_SetMode** (HAL_GPIO_Px, HAL_GPIO_MODE_x)
设置指定*GPIO*的工作模式
- void **HAL_GPIO_SetHigh** (HAL_GPIO_Px)
将指定的*GPIO*端口输出高电平(1)
- void **HAL_GPIO_SetLow** (HAL_GPIO_Px)
将指定的*GPIO*端口输出低电平(0)
- void **HAL_GPIO_Write** (HAL_GPIO_Px, uint8_t)
向指定的*GPIO*端口写入0或1
- void **HAL_GPIO_Invert** (HAL_GPIO_Px)
翻转指定的*GPIO*端口的输出状态
- uint8_t **HAL_GPIO_GetOutData** (HAL_GPIO_Px)
获取当前指定*GPIO*端口的输出值
- uint8_t **HAL_GPIO_GetInData** (HAL_GPIO_Px)
获取当前指定*GPIO*端口的输入值
- void **HAL_GPIO_SetAf** (HAL_GPIO_Px, HAL_GPIO_AFx)
配置*GPIO*的特殊功能,如*UART*、*I2C*、*SPI*等
- void **HAL_GPIO_SwdReuseGpioEnable** (void)
将*SWD*接口配置为普通*GPIO*口
- void **HAL_GPIO_SwdReuseGpioDisable** (void)
将*SWD*接口恢复为*SWD*功能
- void **HAL_GPIO_RstReuseGpioEnable** (void)
将*RST*接口配置为普通*GPIO*口
- void **HAL_GPIO_RstReuseGpioDisable** (void)
将*RST*接口恢复为*RST*功能

4.27.1 枚举类型说明

4.27.1.1 HAL_GPIO_AFx

enum **HAL_GPIO_AFx**

枚举值

HAL_GPIO_AF_ADC	
HAL_GPIO_AF_OPA	
HAL_GPIO_AF_OPA_OUT	
HAL_GPIO_AF_CMP	
HAL_GPIO_AF_DAC	
HAL_GPIO_AF_GPIO	
HAL_GPIO_AF_CLK	
HAL_GPIO_AF_CMP_OUT	
HAL_GPIO_AF_CLK_OUT	
HAL_GPIO_AF_HALL	
HAL_GPIO_AF_MCPWM	
HAL_GPIO_AF_UART	
HAL_GPIO_AF_SPI	
HAL_GPIO_AF_I2C	
HAL_GPIO_AF_TIMER0	
HAL_GPIO_AF_TIMER1	
HAL_GPIO_AF_TIMER2	
HAL_GPIO_AF_TIMER3	
HAL_GPIO_AF_QEP0	
HAL_GPIO_AF_QEP1	
HAL_GPIO_AF_ADC_TRIG	
HAL_GPIO_AF_CAN	
HAL_GPIO_AF_SIF	

在文件 [hal_lksmcu_gpio.h](#) 第 105 行定义.

4.27.1.2 HAL_GPIO_MODE_x

enum [HAL_GPIO_MODE_x](#)

GPIO工作模式定义

枚举值

HAL_GPIO_MODE_IN_EN	
HAL_GPIO_MODE_OUT_EN	
HAL_GPIO_MODE_UP_EN	
HAL_GPIO_MODE_PODE_EN	
HAL_GPIO_MODE_DEFAULT_0	
HAL_GPIO_MODE_DEFAULT_1	
HAL_GPIO_MODE_AFE	GPIO设置为模拟接口
HAL_GPIO_MODE_NONE	GPIO设置为不使用 输出输出均关闭
HAL_GPIO_MODE_IN	GPIO设置为输入模式
HAL_GPIO_MODE_INUP	GPIO设置为上拉输入模式(输入模式,打开上拉电阻)
HAL_GPIO_MODE_OUT	GPIO设置为输出模式
HAL_GPIO_MODE_INOUT	GPIO设置为输入输出模式(可以输出为0或悬空,悬空时输入有效)

枚举值

HAL_GPIO_MODE_INUPOUT	GPIO设置为上拉输入输出模式(可以输出为0或上拉,上拉时输入有效,线与模式)
HAL_GPIO_MODE_OUT_0	GPIO设置为输出模式 初始化完成后io状态为低电平
HAL_GPIO_MODE_INOUT_0	GPIO设置为输入输出模式(可以输出为0或悬空,悬空时输入有效) 初始化完成后io状态为低电平
HAL_GPIO_MODE_↔ INUPOUT_0	GPIO设置为上拉输入输出模式(可以输出为0或上拉,上拉时输入有效,线与模式) 初始化完成后io状态为低电平
HAL_GPIO_MODE_OUT_1	GPIO设置为输出模式 初始化完成后io状态为高电平
HAL_GPIO_MODE_INOUT_1	GPIO设置为输入输出模式(可以输出为0或悬空,悬空时输入有效) 初始化完成后io状态为悬空
HAL_GPIO_MODE_↔ INUPOUT_1	GPIO设置为上拉输入输出模式(可以输出为0或上拉,上拉时输入有效,线与模式) 初始化完成后io状态为上拉

在文件 [hal_lksmcu_gpio.h](#) 第 145 行定义.

4.27.1.3 HAL_GPIO_Px

enum [HAL_GPIO_Px](#)

GPIO口定义

枚举值

HAL_GPIO_P0.0	P0.0
HAL_GPIO_P0.1	P0.1
HAL_GPIO_P0.2	P0.2
HAL_GPIO_P0.3	P0.3
HAL_GPIO_P0.4	P0.4
HAL_GPIO_P0.5	P0.5
HAL_GPIO_P0.6	P0.6
HAL_GPIO_P0.7	P0.7
HAL_GPIO_P0.8	P0.8
HAL_GPIO_P0.9	P0.9
HAL_GPIO_P0.10	P0.10
HAL_GPIO_P0.11	P0.11
HAL_GPIO_P0.12	P0.12
HAL_GPIO_P0.13	P0.13
HAL_GPIO_P0.14	P0.14
HAL_GPIO_P0.15	P0.15
HAL_GPIO_P1.0	P1.0
HAL_GPIO_P1.1	P1.1
HAL_GPIO_P1.2	P1.2
HAL_GPIO_P1.3	P1.3
HAL_GPIO_P1.4	P1.4
HAL_GPIO_P1.5	P1.5
HAL_GPIO_P1.6	P1.6
HAL_GPIO_P1.7	P1.7
HAL_GPIO_P1.8	P1.8

枚举值

HAL_GPIO_P1_9	P1.9
HAL_GPIO_P1_10	P1.10
HAL_GPIO_P1_11	P1.11
HAL_GPIO_P1_12	P1.12
HAL_GPIO_P1_13	P1.13
HAL_GPIO_P1_14	P1.14
HAL_GPIO_P1_15	P1.15
HAL_GPIO_P2_0	P2.0
HAL_GPIO_P2_1	P2.1
HAL_GPIO_P2_2	P2.2
HAL_GPIO_P2_3	P2.3
HAL_GPIO_P2_4	P2.4
HAL_GPIO_P2_5	P2.5
HAL_GPIO_P2_6	P2.6
HAL_GPIO_P2_7	P2.7
HAL_GPIO_P2_8	P2.8
HAL_GPIO_P2_9	P2.9
HAL_GPIO_P2_10	P2.10
HAL_GPIO_P2_11	P2.11
HAL_GPIO_P2_12	P2.12
HAL_GPIO_P2_13	P2.13
HAL_GPIO_P2_14	P2.14
HAL_GPIO_P2_15	P2.15
HAL_GPIO_P3_0	P3.0
HAL_GPIO_P3_1	P3.1
HAL_GPIO_P3_2	P3.2
HAL_GPIO_P3_3	P3.3
HAL_GPIO_P3_4	P3.4
HAL_GPIO_P3_5	P3.5
HAL_GPIO_P3_6	P3.6
HAL_GPIO_P3_7	P3.7
HAL_GPIO_P3_8	P3.8
HAL_GPIO_P3_9	P3.9
HAL_GPIO_P3_10	P3.10
HAL_GPIO_P3_11	P3.11
HAL_GPIO_P3_12	P3.12
HAL_GPIO_P3_13	P3.13
HAL_GPIO_P3_14	P3.14
HAL_GPIO_P3_15	P3.15
HAL_GPIO_P4_0	P4.0
HAL_GPIO_P4_1	P4.1
HAL_GPIO_P4_2	P4.2
HAL_GPIO_P4_3	P4.3
HAL_GPIO_P4_4	P4.4
HAL_GPIO_P4_5	P4.5
HAL_GPIO_P4_6	P4.6
HAL_GPIO_P4_7	P4.7
HAL_GPIO_P4_8	P4.8
HAL_GPIO_P4_9	P4.9
HAL_GPIO_P4_10	P4.10

枚举值

HAL_GPIO_P4_11	P4.11
HAL_GPIO_P4_12	P4.12
HAL_GPIO_P4_13	P4.13
HAL_GPIO_P4_14	P4.14
HAL_GPIO_P4_15	P4.15
HAL_GPIO_P5_0	P5.0
HAL_GPIO_P5_1	P5.1
HAL_GPIO_P5_2	P5.2
HAL_GPIO_P5_3	P5.3
HAL_GPIO_P5_4	P5.4
HAL_GPIO_P5_5	P5.5
HAL_GPIO_P5_6	P5.6
HAL_GPIO_P5_7	P5.7
HAL_GPIO_P5_8	P5.8
HAL_GPIO_P5_9	P5.9
HAL_GPIO_P5_10	P5.10
HAL_GPIO_P5_11	P5.11
HAL_GPIO_P5_12	P5.12
HAL_GPIO_P5_13	P5.13
HAL_GPIO_P5_14	P5.14
HAL_GPIO_P5_15	P5.15

在文件 [hal_lksmcu_gpio.h](#) 第 6 行定义.

4.27.2 函数说明

4.27.2.1 HAL_GPIO_Disable()

```
void HAL_GPIO_Disable (
    void )
```

禁用GPIO模块,关闭GPIO时钟和释放资源

在文件 [hal_lks32mc08x_gpio.c](#) 第 67 行定义.

4.27.2.2 HAL_GPIO_Enable()

```
void HAL_GPIO_Enable (
    void )
```

启用GPIO模块,使能GPIO时钟和必要的资源

在文件 [hal_lks32mc08x_gpio.c](#) 第 60 行定义.

4.27.2.3 HAL_GPIO_GetInData()

```
uint8_t HAL_GPIO_GetInData (
    HAL_GPIO_Px px )
```

获取当前指定GPIO端口的输入值

参数

<i>px</i>	指定的GPIO端口

返回

当前GPIO端口的输入值(0或1)

在文件 [hal_lks32mc08x_gpio.c](#) 第 217 行定义.

4.27.2.4 HAL_GPIO_GetOutData()

```
uint8_t HAL_GPIO_GetOutData (
    HAL_GPIO_Px px )
```

获取当前指定GPIO端口的输出值

参数

<i>px</i>	指定的GPIO端口

返回

当前GPIO端口的输出值(0或1)

在文件 [hal_lks32mc08x_gpio.c](#) 第 194 行定义.

4.27.2.5 HAL_GPIO_Invert()

```
void HAL_GPIO_Invert (
    HAL_GPIO_Px px )
```

翻转指定的GPIO端口的输出状态

参数

<i>px</i>	指定的GPIO端口

在文件 [hal_lks32mc08x_gpio.c](#) 第 174 行定义.

这是这个函数的调用关系图:



4.27.2.6 HAL_GPIO_Reset()

```
void HAL_GPIO_Reset (
    void )
```

模块复位,执行后恢复所有的GPIO为悬空状态(SWD、JTAG和RST口为默认状态)

在文件 [hal_lks32mc08x_gpio.c](#) 第 9 行定义.

4.27.2.7 HAL_GPIO_RstReuseGpioDisable()

```
void HAL_GPIO_RstReuseGpioDisable (
    void )
```

将RST接口恢复为RST功能

在文件 [hal_lks32mc08x_gpio.c](#) 第 340 行定义.

4.27.2.8 HAL_GPIO_RstReuseGpioEnable()

```
void HAL_GPIO_RstReuseGpioEnable (
    void )
```

将RST接口配置为普通GPIO口

在文件 [hal_lks32mc08x_gpio.c](#) 第 327 行定义.

4.27.2.9 HAL_GPIO_SetAf()

```
void HAL_GPIO_SetAf (
    HAL_GPIO_Px px,
    HAL_GPIO_Afx af )
```

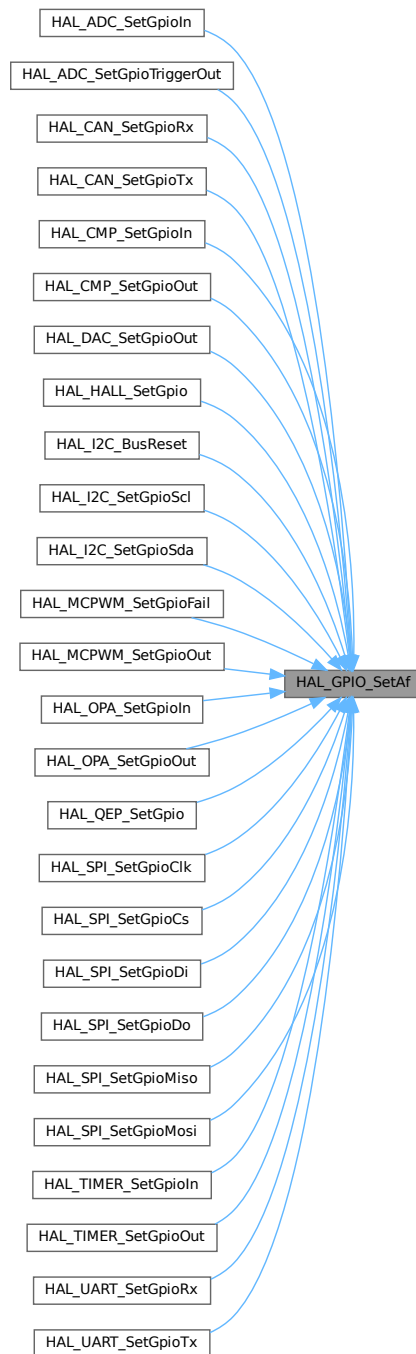
配置GPIO的特殊功能,如UART、I2C、SPI等

参数

<i>px</i>	指定的GPIO端口
<i>af</i>	要配置的特殊功能代码

在文件 [hal_lks32mc08x_gpio.c](#) 第 240 行定义.

这是这个函数的调用关系图:



4.27.2.10 HAL_GPIO_SetHigh()

```
void HAL_GPIO_SetHigh (
    HAL_GPIO_Px px )
```

将指定的GPIO端口输出高电平(1)

参数

<i>px</i>	指定的GPIO端口

注解

在INOUT模式下设为悬空,INUPOUT模式下设为上拉

在文件 [hal_lks32mc08x_gpio.c](#) 第 136 行定义.

4.27.2.11 HAL_GPIO_SetLow()

```
void HAL_GPIO_SetLow (
    HAL_GPIO_Px px )
```

将指定的GPIO端口输出低电平(0)

参数

<i>px</i>	指定的GPIO端口

在文件 [hal_lks32mc08x_gpio.c](#) 第 148 行定义.

4.27.2.12 HAL_GPIO_SetMode()

```
void HAL_GPIO_SetMode (
    HAL_GPIO_Px px,
    HAL_GPIO_MODE_x mode )
```

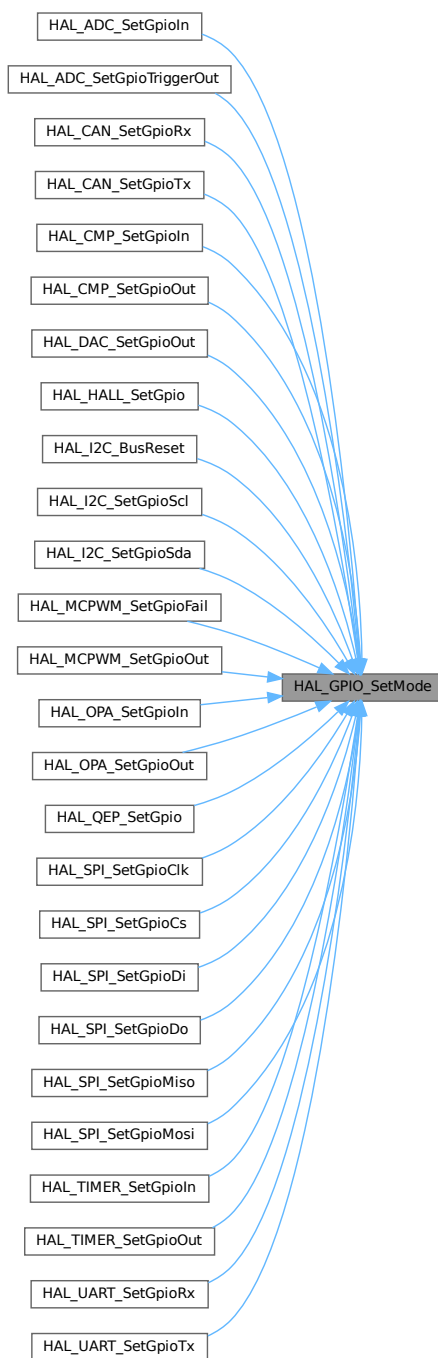
设置指定GPIO的工作模式

参数

<i>px</i>	指定的GPIO端口
<i>mode</i>	要设置的工作模式

在文件 [hal_lks32mc08x_gpio.c](#) 第 76 行定义.

这是这个函数的调用关系图:



4.27.2.13 HAL_GPIO_SwdReuseGpioDisable()

```
void HAL_GPIO_SwdReuseGpioDisable (
    void )
```

将SWD接口恢复为SWD功能

在文件 [hal_lks32mc08x_gpio.c](#) 第 314 行定义。

4.27.2.14 HAL_GPIO_SwdReuseGpioEnable()

```
void HAL_GPIO_SwdReuseGpioEnable (
    void )
```

将SWD接口配置为普通GPIO口

在文件 [hal_lks32mc08x_gpio.c](#) 第 301 行定义.

4.27.2.15 HAL_GPIO_Write()

```
void HAL_GPIO_Write (
    HAL_GPIO_Px px,
    uint8_t value )
```

向指定的GPIO端口写入0或1

参数

<i>px</i>	指定的GPIO端口
<i>value</i>	要写入的值(0或1)

在文件 [hal_lks32mc08x_gpio.c](#) 第 161 行定义.

4.28 hal_lksmcu_gpio.h

[浏览该文件的文档.](#)

```
00001 #ifndef __HAL_LKSMCU_GPIO_H__
00002 #define __HAL_LKSMCU_GPIO_H__
00003 #include "basic.h"
00004 #include "stdint.h"
00006 typedef enum
00007 {
00008     HAL_GPIO_P0_0 = 0x0000,
00009     HAL_GPIO_P0_1 = 0x0001,
00010     HAL_GPIO_P0_2 = 0x0002,
00011     HAL_GPIO_P0_3 = 0x0003,
00012     HAL_GPIO_P0_4 = 0x0004,
00013     HAL_GPIO_P0_5 = 0x0005,
00014     HAL_GPIO_P0_6 = 0x0006,
00015     HAL_GPIO_P0_7 = 0x0007,
00016     HAL_GPIO_P0_8 = 0x0008,
00017     HAL_GPIO_P0_9 = 0x0009,
00018     HAL_GPIO_P0_10 = 0x000a,
00019     HAL_GPIO_P0_11 = 0x000b,
00020     HAL_GPIO_P0_12 = 0x000c,
00021     HAL_GPIO_P0_13 = 0x000d,
00022     HAL_GPIO_P0_14 = 0x000e,
00023     HAL_GPIO_P0_15 = 0x000f,
00024     HAL_GPIO_P1_0 = 0x0100,
00025     HAL_GPIO_P1_1 = 0x0101,
00026     HAL_GPIO_P1_2 = 0x0102,
00027     HAL_GPIO_P1_3 = 0x0103,
00028     HAL_GPIO_P1_4 = 0x0104,
00029     HAL_GPIO_P1_5 = 0x0105,
00030     HAL_GPIO_P1_6 = 0x0106,
00031     HAL_GPIO_P1_7 = 0x0107,
00032     HAL_GPIO_P1_8 = 0x0108,
00033     HAL_GPIO_P1_9 = 0x0109,
00034     HAL_GPIO_P1_10 = 0x010a,
00035     HAL_GPIO_P1_11 = 0x010b,
00036     HAL_GPIO_P1_12 = 0x010c,
00037     HAL_GPIO_P1_13 = 0x010d,
```

```

00038     HAL_GPIO_P1_14 = 0x010e,
00039     HAL_GPIO_P1_15 = 0x010f,
00040     HAL_GPIO_P2_0  = 0x0200,
00041     HAL_GPIO_P2_1  = 0x0201,
00042     HAL_GPIO_P2_2  = 0x0202,
00043     HAL_GPIO_P2_3  = 0x0203,
00044     HAL_GPIO_P2_4  = 0x0204,
00045     HAL_GPIO_P2_5  = 0x0205,
00046     HAL_GPIO_P2_6  = 0x0206,
00047     HAL_GPIO_P2_7  = 0x0207,
00048     HAL_GPIO_P2_8  = 0x0208,
00049     HAL_GPIO_P2_9  = 0x0209,
00050     HAL_GPIO_P2_10 = 0x020a,
00051     HAL_GPIO_P2_11 = 0x020b,
00052     HAL_GPIO_P2_12 = 0x020c,
00053     HAL_GPIO_P2_13 = 0x020d,
00054     HAL_GPIO_P2_14 = 0x020e,
00055     HAL_GPIO_P2_15 = 0x020f,
00056     HAL_GPIO_P3_0  = 0x0300,
00057     HAL_GPIO_P3_1  = 0x0301,
00058     HAL_GPIO_P3_2  = 0x0302,
00059     HAL_GPIO_P3_3  = 0x0303,
00060     HAL_GPIO_P3_4  = 0x0304,
00061     HAL_GPIO_P3_5  = 0x0305,
00062     HAL_GPIO_P3_6  = 0x0306,
00063     HAL_GPIO_P3_7  = 0x0307,
00064     HAL_GPIO_P3_8  = 0x0308,
00065     HAL_GPIO_P3_9  = 0x0309,
00066     HAL_GPIO_P3_10 = 0x030a,
00067     HAL_GPIO_P3_11 = 0x030b,
00068     HAL_GPIO_P3_12 = 0x030c,
00069     HAL_GPIO_P3_13 = 0x030d,
00070     HAL_GPIO_P3_14 = 0x030e,
00071     HAL_GPIO_P3_15 = 0x030f,
00072     HAL_GPIO_P4_0  = 0x0400,
00073     HAL_GPIO_P4_1  = 0x0401,
00074     HAL_GPIO_P4_2  = 0x0402,
00075     HAL_GPIO_P4_3  = 0x0403,
00076     HAL_GPIO_P4_4  = 0x0404,
00077     HAL_GPIO_P4_5  = 0x0405,
00078     HAL_GPIO_P4_6  = 0x0406,
00079     HAL_GPIO_P4_7  = 0x0407,
00080     HAL_GPIO_P4_8  = 0x0408,
00081     HAL_GPIO_P4_9  = 0x0409,
00082     HAL_GPIO_P4_10 = 0x040a,
00083     HAL_GPIO_P4_11 = 0x040b,
00084     HAL_GPIO_P4_12 = 0x040c,
00085     HAL_GPIO_P4_13 = 0x040d,
00086     HAL_GPIO_P4_14 = 0x040e,
00087     HAL_GPIO_P4_15 = 0x040f,
00088     HAL_GPIO_P5_0  = 0x0500,
00089     HAL_GPIO_P5_1  = 0x0501,
00090     HAL_GPIO_P5_2  = 0x0502,
00091     HAL_GPIO_P5_3  = 0x0503,
00092     HAL_GPIO_P5_4  = 0x0504,
00093     HAL_GPIO_P5_5  = 0x0505,
00094     HAL_GPIO_P5_6  = 0x0506,
00095     HAL_GPIO_P5_7  = 0x0507,
00096     HAL_GPIO_P5_8  = 0x0508,
00097     HAL_GPIO_P5_9  = 0x0509,
00098     HAL_GPIO_P5_10 = 0x050a,
00099     HAL_GPIO_P5_11 = 0x050b,
00100     HAL_GPIO_P5_12 = 0x050c,
00101     HAL_GPIO_P5_13 = 0x050d,
00102     HAL_GPIO_P5_14 = 0x050e,
00103     HAL_GPIO_P5_15 = 0x050f,
00104 } HAL_GPIO_Px;
00105 typedef enum
00106 {
00107     HAL_GPIO_AF_ADC      = 0, // ADC输入
00108     HAL_GPIO_AF_OPA      = 0, // 运放输入
00109     HAL_GPIO_AF_OPA_OUT  = 0, // 运放输出
00110     HAL_GPIO_AF_CMP      = 0, // 比较器输入
00111     HAL_GPIO_AF_DAC      = 0, // DAC输出,需要DAC同步设置DAC输出使能
00112     HAL_GPIO_AF_GPIO     = 0, // GPIO功能
00113     HAL_GPIO_AF_CLK      = 0, // 时钟输入
00114
00115     HAL_GPIO_AF_CMP_OUT  = 1, // 比较器比较输出
00116     HAL_GPIO_AF_CLK_OUT  = 1, // 时钟输出
00117
00118     HAL_GPIO_AF_HALL     = 2, // 霍尔输入
00119     HAL_GPIO_AF_MCPWM    = 3, // MCPWMFAIL输入 MCPWM输出
00120     HAL_GPIO_AF_UART     = 4, // UART输入输出
00121     HAL_GPIO_AF_SPI      = 5, // CS CLK MOSI MISO
00122     HAL_GPIO_AF_I2C      = 6, // SCLâ€™ESDA
00123     HAL_GPIO_AF_TIMER0   = 7, // TIMER0输入或输出
00124     HAL_GPIO_AF_TIMER1   = 7, // TIMER1输入或输出

```

```

00125     HAL_GPIO_AF_TIMER2 = 8, // TIMER2输入或输出
00126     HAL_GPIO_AF_TIMER3 = 8, // TIMER3输入或输出
00127     HAL_GPIO_AF_QEP0  = 8, // 编码器输入
00128     HAL_GPIO_AF_QEP1  = 8, // 编码器输入
00129
00130     HAL_GPIO_AF_ADC_TRIG = 9, // adc触发信号输出
00131     HAL_GPIO_AF_CAN      = 10, // CAN输入输出
00132     HAL_GPIO_AF_SIF      = 11, // SIF信号输出
00133
00134 #ifdef HAL_TEST_NOT_SUPPORTED
00135     HAL_GPIO_AF_CL      = 0, // CL输出 不支持
00136     HAL_GPIO_AF_TIMER4 = 0, // TIMER4输入或输出 不支持
00137     HAL_GPIO_AF_TIMER5 = 0, // TIMER5输入或输出 不支持
00138     HAL_GPIO_AF_QEP2   = 0, // 编码器输入 不支持
00139     HAL_GPIO_AF_QEP3   = 0, // 编码器输入 不支持
00140 #endif
00141 } HAL_GPIO_Afx;
00142 typedef enum
00143 {
00144     HAL_GPIO_MODE_IN_EN      = BIT0, // 输入使能
00145     HAL_GPIO_MODE_OUT_EN     = BIT1, // 输出使能
00146     HAL_GPIO_MODE_UP_EN      = BIT2, // 上拉使能
00147     HAL_GPIO_MODE_PODE_EN     = BIT3, // 开漏使能
00148     HAL_GPIO_MODE_DEFAULT_0   = BIT4, // 初始化输出为低电平
00149     HAL_GPIO_MODE_DEFAULT_1   = BIT5, // 初始化输出为高电平
00150
00151     HAL_GPIO_MODE_AFE         = 0,
00152     HAL_GPIO_MODE_NONE       = HAL_GPIO_MODE_AFE,
00153     HAL_GPIO_MODE_IN         = HAL_GPIO_MODE_IN_EN,
00154     HAL_GPIO_MODE_INUP       = HAL_GPIO_MODE_IN | HAL_GPIO_MODE_UP_EN,
00155     HAL_GPIO_MODE_OUT        = HAL_GPIO_MODE_OUT_EN,
00156     HAL_GPIO_MODE_INOUT      = HAL_GPIO_MODE_IN | HAL_GPIO_MODE_OUT | HAL_GPIO_MODE_PODE_EN,
00157     HAL_GPIO_MODE_INUPOUT    = HAL_GPIO_MODE_INUP | HAL_GPIO_MODE_PODE_EN,
00158     HAL_GPIO_MODE_OUT_0      = HAL_GPIO_MODE_OUT | HAL_GPIO_MODE_DEFAULT_0,
00159     HAL_GPIO_MODE_INOUT_0    = HAL_GPIO_MODE_INOUT | HAL_GPIO_MODE_DEFAULT_0,
00160     HAL_GPIO_MODE_INUPOUT_0  = HAL_GPIO_MODE_INUPOUT | HAL_GPIO_MODE_DEFAULT_0,
00161     HAL_GPIO_MODE_OUT_1      = HAL_GPIO_MODE_OUT | HAL_GPIO_MODE_DEFAULT_1,
00162     HAL_GPIO_MODE_INOUT_1    = HAL_GPIO_MODE_INOUT | HAL_GPIO_MODE_DEFAULT_1,
00163     HAL_GPIO_MODE_INUPOUT_1  = HAL_GPIO_MODE_INUPOUT | HAL_GPIO_MODE_DEFAULT_1,
00164 } HAL_GPIO_MODE_x;
00165 // 模块初始化
00166 void HAL_GPIO_Reset(void); // 模块复位, 执行后恢复所有的GPIO为悬空状态 (SWD、JTAG和RST口为默认状态)
00167 void HAL_GPIO_Enable(void); // 模块打开
00168 void HAL_GPIO_Disable(void); // 模块关闭
00169
00170 // 输入输出
00171 void HAL_GPIO_SetMode(HAL_GPIO_Px, HAL_GPIO_MODE_x); // 设置指定GPIO的工作模式
00172 void HAL_GPIO_SetHigh(HAL_GPIO_Px); // 输出1 (INOUT和INUPOUT模式下分别为悬空和上拉)
00173 void HAL_GPIO_SetLow(HAL_GPIO_Px); // 输出0
00174 void HAL_GPIO_Write(HAL_GPIO_Px, uint8_t); // 输出0或1
00175 void HAL_GPIO_Invert(HAL_GPIO_Px); // 翻转指定的GPIO
00176 uint8_t HAL_GPIO_GetOutData(HAL_GPIO_Px); // 获取当前GPIO的输出值, 返回值0或1
00177 uint8_t HAL_GPIO_GetInData(HAL_GPIO_Px); // 获取当前GPIO的输入值, 返回值0或1
00178
00179 // 内部使用
00180 void HAL_GPIO_SetAf(HAL_GPIO_Px, HAL_GPIO_Afx); // 配置GPIO的特殊功能 (UART、I2C、SPI等, 由对应模块的初始化调用)
00181 // 特殊GPIO口配置
00182 void HAL_GPIO_SwdReuseGpioEnable(void); // 设置SWD接口为普通GPIO口
00183 void HAL_GPIO_SwdReuseGpioDisable(void); // 设置SWD接口为SWD功能
00184 void HAL_GPIO_RstReuseGpioEnable(void); // 设置RST接口为普通GPIO口
00185 void HAL_GPIO_RstReuseGpioDisable(void); // 设置RST接口为RST功能
00186 #endif

```

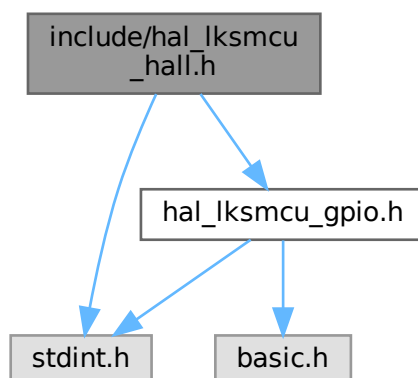
4.29 include/hal_lksmcu_hall.h 文件参考

```

#include "stdint.h"
#include "hal_lksmcu_gpio.h"

```

hal_lksmcu_hall.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



枚举

- enum `HAL_HALL_x` { `HAL_HALL_0` , `HAL_HALL_1` }

函数

- void `HAL_HALL_Reset` (`HAL_HALL_x`)
模块复位
- void `HAL_HALL_Enable` (`HAL_HALL_x`)
模块打开
- void `HAL_HALL_Disable` (`HAL_HALL_x`)
模块关闭
- void `HAL_HALL_SetFreq` (`HAL_HALL_x`, uint32_t freq)
设置hall频率
- void `HAL_HALL_SetTh` (`HAL_HALL_x`, uint32_t th)
设置hall阈值
- void `HAL_HALL_StartCounting` (`HAL_HALL_x`)
开始计数 `HAL_HALL_Enable`后默认开始计数
- void `HAL_HALL_StopCounting` (`HAL_HALL_x`)
停止计数(同时计数器回到0)
- void `HAL_HALL_SetFilter` (`HAL_HALL_x`, uint32_t filter_mclk)
滤波频率设置
- void `HAL_HALL_SetFilteruS` (`HAL_HALL_x`, uint32_t filter_uS)

- 设置滤波, 单位: μS
- void [HAL_HALL_SetFilter75](#) ([HAL_HALL_x](#), uint8_t ena)
设置7/5滤波使能(第一级滤波,连续采样7次,5次值一致)
- uint32_t [HAL_HALL_ReadTh](#) ([HAL_HALL_x](#))
获取当前的周期计数值
- uint8_t [HAL_HALL_ReadHallData](#) ([HAL_HALL_x](#))
获取当前的数据
- uint32_t [HAL_HALL_ReadHallWidth](#) ([HAL_HALL_x](#))
获取当前的宽度
- uint32_t [HAL_HALL_GetCNT](#) ([HAL_HALL_x](#))
获取当前计数器的值
- void [HAL_HALL_SetGpio](#) ([HAL_HALL_x](#), [HAL_GPIO_Px](#))
设置输入引脚

4.29.1 枚举类型说明

4.29.1.1 HAL_HALL_x

enum [HAL_HALL_x](#)

枚举值

HAL_↔ HALL_0	
HAL_↔ HALL_1	

在文件 [hal_lksmcu_hall.h](#) 第 5 行定义.

4.29.2 函数说明

4.29.2.1 HAL_HALL_Disable()

```
void HAL_HALL_Disable (
    HAL\_HALL\_x hall )
```

模块关闭

参数

<i>hall</i>	模块号

在文件 [hal_lks32mc08x_hall.c](#) 第 41 行定义.

4.29.2.2 HAL_HALL_Enable()

```
void HAL_HALL_Enable (
    HAL\_HALL\_x hall )
```

模块打开

参数

<i>hall</i>	模块号

在文件 [hal_lks32mc08x.hall.c](#) 第 29 行定义.

4.29.2.3 HAL_HALL_GetCNT()

```
uint32_t HAL_HALL_GetCNT (
    HAL_HALL_x hall )
```

获取当前hall计数器的值

参数

<i>hall</i>	模块号

返回

计数值

在文件 [hal_lks32mc08x.hall.c](#) 第 124 行定义.

4.29.2.4 HAL_HALL_ReadHallData()

```
uint8_t HAL_HALL_ReadHallData (
    HAL_HALL_x hall )
```

获取当前hall的数据

参数

<i>hall</i>	模块号

返回

数据值

在文件 [hal_lks32mc08x.hall.c](#) 第 104 行定义.

4.29.2.5 HAL_HALL_ReadHallWidth()

```
uint32_t HAL_HALL_ReadHallWidth (
    HAL_HALL_x hall )
```

获取当前hall的宽度

参数

<i>hall</i>	模块号

返回

宽度值

在文件 [hal_lks32mc08x.hall.c](#) 第 114 行定义.

4.29.2.6 HAL_HALL_ReadTh()

```
uint32_t HAL_HALL_ReadTh (
    HAL_HALL_x hall )
```

获取当前hall的周期计数值

参数

<i>hall</i>	模块号

返回

周期计数值

在文件 [hal_lks32mc08x.hall.c](#) 第 94 行定义.

4.29.2.7 HAL_HALL_Reset()

```
void HAL_HALL_Reset (
    HAL_HALL_x hall )
```

模块复位

参数

<i>hall</i>	模块号

在文件 [hal_lks32mc08x.hall.c](#) 第 14 行定义.

4.29.2.8 HAL_HALL_SetFilter()

```
void HAL_HALL_SetFilter (
    HAL_HALL_x hall_x,
    uint32_t filter_mclk )
```

璁剧疆婁ゆ尝

参数

<i>hall</i>	模块号
<i>filter_mclk</i>	滤波值

在文件 [hal_lks32mc08x_hall.c](#) 第 165 行定义.

这是这个函数的调用关系图:



4.29.2.9 HAL_HALL_SetFilter75()

```
void HAL_HALL_SetFilter75 (
    HAL_HALL_x hall_x,
    uint8_t ena )
```

设置7/5滤波使能(第一级滤波,连续采样7次,5次值一致)

参数

<i>hall_x</i>	模块号
<i>ena</i>	0:关闭(默认) 非0:使能

在文件 [hal_lks32mc08x_hall.c](#) 第 210 行定义.

4.29.2.10 HAL_HALL_SetFilteruS()

```
void HAL_HALL_SetFilteruS (
    HAL_HALL_x hall,
    uint32_t filter_uS )
```

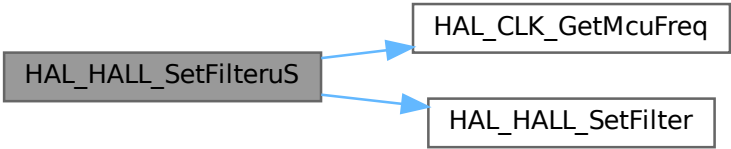
设置滤波, 单位: uS

参数

<i>hall</i>	模块号
<i>filter_uS</i>	滤波值

在文件 `hal_lks32mc08x.hall.c` 第 199 行定义.

函数调用图:



4.29.2.11 HAL_HALL_SetFreq()

```
void HAL_HALL_SetFreq (
    HAL_HALL_x hall,
    uint32_t freq )
```

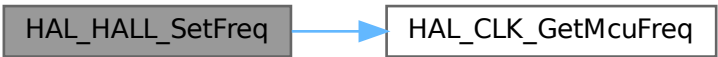
设置hall频率

参数

<i>hall</i>	模块号
<i>freq</i>	频率值

在文件 `hal_lks32mc08x.hall.c` 第 54 行定义.

函数调用图:



4.29.2.12 HAL_HALL_SetGpio()

```
void HAL_HALL_SetGpio (
    HAL_HALL_x hall,
    HAL_GPIO_Px px )
```

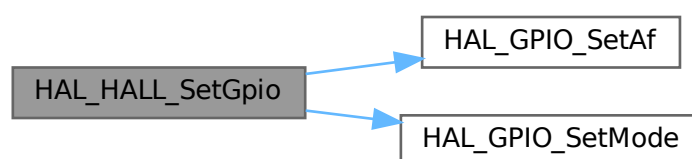
设置hall输入引脚

参数

<i>hall</i>	模块号
<i>px</i>	引脚

在文件 [hal_lks32mc08x.hall.c](#) 第 134 行定义.

函数调用图:



4.29.2.13 HAL_HALL_SetTh()

```
void HAL_HALL_SetTh (
    HAL_HALL_x hall_x,
    uint32_t th )
```

设置hall阈值

参数

<i>hall</i>	模块号
<i>th</i>	阈值

在文件 [hal_lks32mc08x.hall.c](#) 第 66 行定义.

4.29.2.14 HAL_HALL_StartCounting()

```
void HAL_HALL_StartCounting (
    HAL_HALL_x hall )
```

开始计数 HAL_HALL_Enable后默认开始计数

参数

<i>hall</i>	模块号

在文件 `hal_lks32mc08x_hall.c` 第 145 行定义.

4.29.2.15 HAL_HALL_StopCounting()

```
void HAL_HALL_StopCounting (
    HAL_HALL_x hall )
```

停止计数(同时计数器回到0)

参数

<i>hall</i>	模块号

在文件 `hal_lks32mc08x_hall.c` 第 154 行定义.

4.30 hal_lksmcu_hall.h

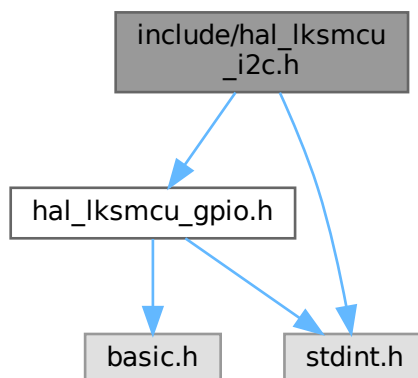
浏览该文件的文档.

```
00001 #ifndef __HAL_LKSMCU_HALL_H__
00002 #define __HAL_LKSMCU_HALL_H__
00003 #include "stdint.h"
00004 #include "hal_lksmcu_gpio.h"
00005 typedef enum
00006 {
00007     HAL_HALL_0,
00008     HAL_HALL_1,
00009 } HAL_HALL_x;
00010 // 初始化
00011 void HAL_HALL_Reset(HAL_HALL_x); // 模块复位
00012 void HAL_HALL_Enable(HAL_HALL_x); // 模块打开
00013 void HAL_HALL_Disable(HAL_HALL_x); // 模块关闭
00014 //
00015 void HAL_HALL_SetFreq(HAL_HALL_x, uint32_t freq); // 设置hall频率 SetFreq和SetTh配置其中一个即可
00016 void HAL_HALL_SetTh(HAL_HALL_x, uint32_t th); // 设置hall阈值 SetFreq和SetTh配置其中一个即可
00017 void HAL_HALL_StartCounting(HAL_HALL_x); // 开始计数 HAL_HALL_Enable后默认开始计数
00018 void HAL_HALL_StopCounting(HAL_HALL_x); // 停止计数(同时计数器回到0)
00019 void HAL_HALL_SetFilter(HAL_HALL_x, uint32_t filter_mclk); // 设置滤波, 单位: MCU时钟
00020 void HAL_HALL_SetFilteruS(HAL_HALL_x, uint32_t filter_uS); // 设置滤波, 单位: uS
00021 void HAL_HALL_SetFilter75(HAL_HALL_x, uint8_t ena); // 设置7/5滤波使能, 0:关闭(默认), 非0:使能
00022
00023 uint32_t HAL_HALL_ReadTh(HAL_HALL_x); // 获取当前hall的周期计数值
00024 uint8_t HAL_HALL_ReadHallData(HAL_HALL_x); // 获取当前hall的数据
00025 uint32_t HAL_HALL_ReadHallWidth(HAL_HALL_x); // 获取当前hall捕获到的宽度
00026 uint32_t HAL_HALL_GetCNT(HAL_HALL_x); // 获取当前hall计数器的值
00027
00028 void HAL_HALL_SetGpio(HAL_HALL_x, HAL_GPIO_Px); // 设置hall输入引脚
00029 #endif
```

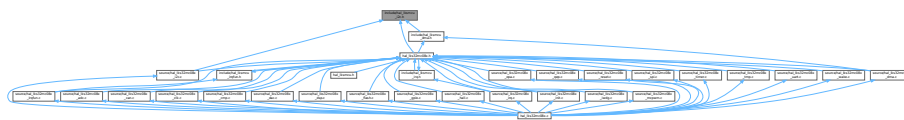
4.31 include/hal_lksmcu_i2c.h 文件参考

```
#include "hal_lksmcu_gpio.h"
#include "stdint.h"
```

hal_lksmcu_i2c.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



枚举

- enum `HAL_I2C_x` { `HAL_I2C_0` , `HAL_I2C_1` }

函数

- void `HAL_I2C_Reset` (`HAL_I2C_x`)
复位I2C模块
- void `HAL_I2C_Enable` (`HAL_I2C_x`)
打开I2C模块
- void `HAL_I2C_Disable` (`HAL_I2C_x`)
关闭I2C模块
- void `HAL_I2C_SetBaud` (`HAL_I2C_x`, uint32_t baud)
设置I2C模块波特率
- void `HAL_I2C_SetClockDiv` (`HAL_I2C_x` i2c_x, uint32_t div)
- uint32_t `HAL_I2C_GetClockDiv` (`HAL_I2C_x` i2c_x)
- uint32_t `HAL_I2C_GetClockFreq` (`HAL_I2C_x` i2c_x)
- uint32_t `HAL_I2C_GetBaud` (`HAL_I2C_x` i2c_x)
- uint8_t `HAL_I2C_TxData` (`HAL_I2C_x`, uint8_t addr, uint8_t *data, uint32_t len)
发送数据到I2C设备
- uint8_t `HAL_I2C_RxData` (`HAL_I2C_x`, uint8_t addr, uint8_t *data, uint32_t len)
从I2C设备接收数据

- void HAL_I2C_SetGpioScl (HAL_I2C_x, HAL_GPIO_Px i2cgpio)
 设置I2C模块的GPIO
- void HAL_I2C_SetGpioSda (HAL_I2C_x, HAL_GPIO_Px i2cgpio)
 设置I2C模块的GPIO
- void HAL_I2C_BusReset (HAL_I2C_x)
 复位I2C总线

4.31.1 枚举类型说明

4.31.1.1 HAL_I2C_x

```
enum HAL_I2C_x
```

枚举值

HAL_I2C↔ C_0	
HAL_I2C↔ C_1	

在文件 [hal_lksmcu_i2c.h](#) 第 5 行定义.

4.31.2 函数说明

4.31.2.1 HAL_I2C_BusReset()

```
void HAL_I2C_BusReset (  
    HAL_I2C_x i2c_x )
```

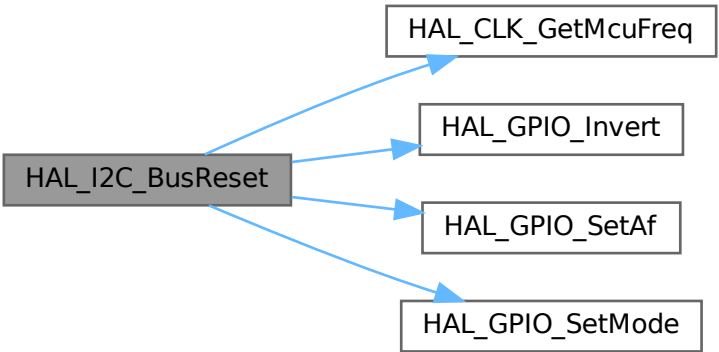
复位I2C总线

参数

i2c↔ _x	I2C模块编号

在文件 [hal_lks32mc08x_i2c.c](#) 第 234 行定义.

函数调用图:



4.31.2.2 HAL_I2C.Disable()

```
void HAL_I2C_Disable (
    HAL_I2C_x i2c_x )
```

关闭I2C模块

参数

<i>i2c</i> _x	I2C模块编号

在文件 [hal_lks32mc08x_i2c.c](#) 第 38 行定义.

4.31.2.3 HAL_I2C.Enable()

```
void HAL_I2C_Enable (
    HAL_I2C_x i2c_x )
```

打开I2C模块

参数

<i>i2c</i> _x	I2C模块编号

在文件 [hal_lks32mc08x_i2c.c](#) 第 26 行定义.

4.31.2.4 HAL_I2C_GetBaud()

```
uint32_t HAL_I2C_GetBaud (
    HAL_I2C_x i2c_x )
```

在文件 [hal_lks32mc08x_i2c.c](#) 第 86 行定义.

函数调用图:



4.31.2.5 HAL_I2C_GetClockDiv()

```
uint32_t HAL_I2C_GetClockDiv (
    HAL_I2C_x i2c_x )
```

在文件 [hal_lks32mc08x_i2c.c](#) 第 73 行定义.

这是这个函数的调用关系图:

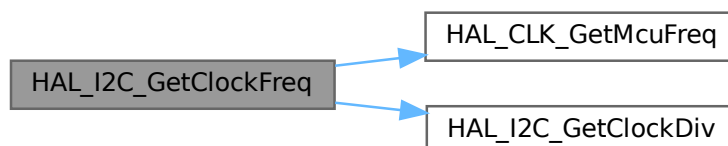


4.31.2.6 HAL_I2C_GetClockFreq()

```
uint32_t HAL_I2C_GetClockFreq (
    HAL_I2C_x i2c_x )
```

在文件 [hal_lks32mc08x_i2c.c](#) 第 80 行定义.

函数调用图:



这是这个函数的调用关系图:



4.31.2.7 HAL_I2C_Reset()

```
void HAL_I2C_Reset (
    HAL_I2C_x i2c_x )
```

复位I2C模块

参数

<i>i2c_x</i>	I2C模块编号

在文件 [hal_lks32mc08x_i2c.c](#) 第 13 行定义.

4.31.2.8 HAL_I2C_RxData()

```
uint8_t HAL_I2C_RxData (
    HAL_I2C_x i2c_x,
    uint8_t addr,
    uint8_t * data,
    uint32_t len )
```

从I2C设备接收数据

参数

<i>i2c_x</i>	I2C模块编号
<i>addr</i>	设备地址
<i>data</i>	数据缓冲区
<i>len</i>	数据长度

返回

0:成功 非0:失败

在文件 [hal_lks32mc08x_i2c.c](#) 第 160 行定义.

4.31.2.9 HAL_I2C_SetBaud()

```
void HAL_I2C_SetBaud (
    HAL_I2C_x i2c_x,
    uint32_t baud )
```

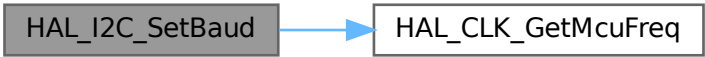
设置I2C模块波特率

参数

<i>i2c_x</i>	I2C模块编号
<i>baud</i>	波特率

在文件 `hal_lks32mc08x_i2c.c` 第 51 行定义.

函数调用图:



4.31.2.10 HAL_I2C_SetClockDiv()

```
void HAL_I2C_SetClockDiv (
    HAL_I2C_x i2c_x,
    uint32_t div )
```

在文件 `hal_lks32mc08x_i2c.c` 第 63 行定义.

4.31.2.11 HAL_I2C_SetGpioScl()

```
void HAL_I2C_SetGpioScl (
    HAL_I2C_x i2c_x,
    HAL_GPIO_Px i2cgpio )
```

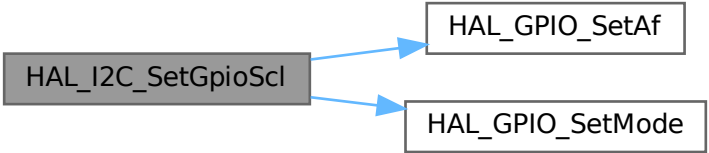
设置I2C模块的GPIO

参数

<i>i2c_x</i>	I2C模块编号
<i>i2cgpio</i>	GPIO编号

在文件 [hal_lks32mc08x.i2c.c](#) 第 209 行定义.

函数调用图:



4.31.2.12 HAL_I2C.SetGpioSda()

```
void HAL_I2C_SetGpioSda (
    HAL_I2C_x i2c_x,
    HAL_GPIO_Px i2cgpio )
```

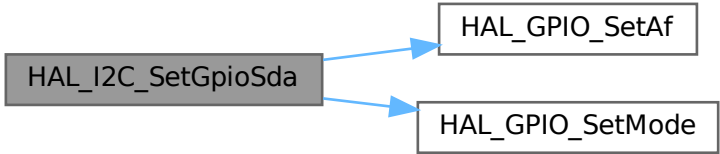
设置I2C模块的GPIO

参数

<i>i2c_x</i>	I2C模块编号
<i>i2cgpio</i>	GPIO编号

在文件 [hal_lks32mc08x.i2c.c](#) 第 222 行定义.

函数调用图:



4.31.2.13 HAL_I2C.TxDData()

```
uint8_t HAL_I2C_TxDData (
    HAL_I2C_x i2c_x,
```

```
uint8_t  addr,  
uint8_t * data,  
uint32_t len )
```

发送数据到I2C设备

参数

<i>i2c</i> _X	I2C模块编号
<i>addr</i>	设备地址
<i>data</i>	数据缓冲区
<i>len</i>	数据长度

返回

0:成功 非0:失败

在文件 `hal_lks32mc08x_i2c.c` 第 99 行定义.

4.32 hal_ksmcu_i2c.h

[浏览该文件的文档.](#)

```

00001 #ifndef __HAL_LKSMCU_I2C_H__
00002 #define __HAL_LKSMCU_I2C_H__
00003 #include "hal_lksmcu_gpio.h"
00004 #include "stdint.h"
00005 typedef enum
00006 {
00007     HAL_I2C_0,
00008     HAL_I2C_1,
00009 } HAL_I2C_x;
00010 // 初始化
00011 void HAL_I2C_Reset(HAL_I2C_x); // 模块复位
00012 void HAL_I2C_Enable(HAL_I2C_x); // 模块打开
00013 void HAL_I2C_Disable(HAL_I2C_x); // 模块关闭
00014 //
00015 void HAL_I2C_SetBaud(HAL_I2C_x, uint32_t baud);
00016 void HAL_I2C_SetClockDiv(HAL_I2C_x i2c_x, uint32_t div); // 设置I2C全局工作时钟分频 div=实际除数(I2C0/1共享)
00017 uint32_t HAL_I2C_GetClockDiv(HAL_I2C_x i2c_x); // 获取I2C工作时钟分频实际除数
00018 uint32_t HAL_I2C_GetClockFreq(HAL_I2C_x i2c_x); // 获取I2C工作时钟频率(Hz)
00019 uint32_t HAL_I2C_GetBaud(HAL_I2C_x i2c_x); // 获取当前SCL波特率(Hz)
00020 uint8_t HAL_I2C_TxData(HAL_I2C_x, uint8_t addr, uint8_t *data, uint32_t len);
00021 uint8_t HAL_I2C_RxData(HAL_I2C_x, uint8_t addr, uint8_t *data, uint32_t len);
00022
00023 void HAL_I2C_SetGpioScl(HAL_I2C_x, HAL_GPIO_Px i2c_gpio);
00024 void HAL_I2C_SetGpioSda(HAL_I2C_x, HAL_GPIO_Px i2c_gpio);
00025
00026 void HAL_I2C_BusReset(HAL_I2C_x); // 复位总线
00027
00028 #endif

```

4.33 include/hal_lksmcu_init.h 文件参考

此图展示该文件直接或间接的被哪些文件引用了:



宏定义

- `#define HAL_INIT_DEFAULE_MCLK 96000000`
芯片的默认时钟频率
- `#define HAL_INIT_DEFAULE_MCLK_US 96`
- `#define HAL_INIT_DEFAULE_MCLK_MS 96000`

枚举

- `enum HAL_INIT_POWER_x { HAL_INIT_POWER_3_3V = 0 , HAL_INIT_POWER_5V }`

函数

- `void HAL_Init_SysInit (void)`
- `void HAL_Init_PowerReady (HAL_INIT_POWER_x powerx)`

4.33.1 宏定义说明

4.33.1.1 HAL_INIT_DEFAULE_MCLK

```
#define HAL_INIT_DEFAULE_MCLK 96000000
```

芯片的默认时钟频率

在文件 `hal_lksmcu_init.h` 第 6 行定义.

4.33.1.2 HAL_INIT_DEFAULE_MCLK_MS

```
#define HAL_INIT_DEFAULE_MCLK_MS 96000
```

在文件 `hal_lksmcu_init.h` 第 8 行定义.

4.33.1.3 HAL_INIT_DEFAULE_MCLK_US

```
#define HAL_INIT_DEFAULE_MCLK_US 96
```

在文件 `hal_lksmcu_init.h` 第 7 行定义.

4.33.2 枚举类型说明

4.33.2.1 HAL_INIT_POWER_x

```
enum HAL_INIT_POWER_x
```


枚举值

HAL_INIT_POWER_3V	
HAL_INIT_POWER_5V	

在文件 [hal_lksmcu_init.h](#) 第 9 行定义.

4.33.3 函数说明

4.33.3.1 HAL_INIT_PowerReady()

```
void HAL_INIT_PowerReady (
    HAL_INIT_POWER_x powerx )
```

在文件 [hal_lks32mc08x_init.c](#) 第 12 行定义.

函数调用图:

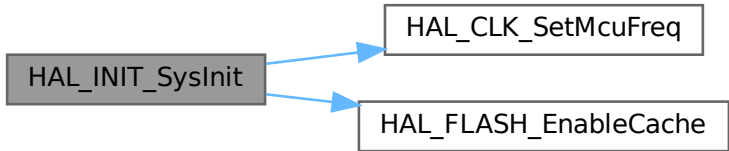


4.33.3.2 HAL_INIT_SysInit()

```
void HAL_INIT_SysInit (
    void )
```

在文件 [hal_lks32mc08x_init.c](#) 第 6 行定义.

函数调用图:



4.34 hal_lksmcu_init.h

浏览该文件的文档。

```

00001 #ifndef __HAL_LKSMCU_INIT_H__
00002 #define __HAL_LKSMCU_INIT_H__
00006 #define HAL_INIT_DEFAULT_MCLK 96000000
00007 #define HAL_INIT_DEFAULT_MCLK_US 96 // 上电后的默认时钟1us分频系数
00008 #define HAL_INIT_DEFAULT_MCLK_MS 96000 // 上电后的默认时钟1ms分频系数
00009 typedef enum
00010 {
00011     HAL_INIT_POWER_3_3V = 0,
00012     HAL_INIT_POWER_5V,
00013 } HAL_INIT_POWER_x;
00014 void HAL_INIT_SysInit(void);
00015 void HAL_INIT_PowerReady(HAL_INIT_POWER_x powerx);
00016
00017 #endif

```

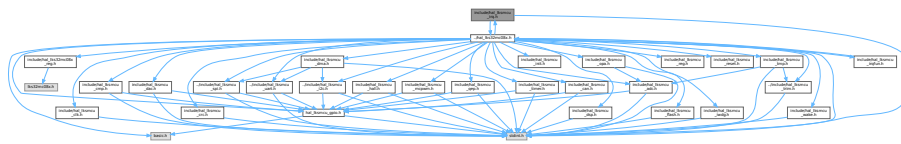
4.35 include/hal_lksmcu_irq.h 文件参考

```

#include "stdint.h"
#include "../hal_lks32mc08x.h"

```

hal_lksmcu_irq.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



宏定义

- #define HAL_IRQHandler_TIMER0 TIMER0_IRQHandler
中断函数名
- #define HAL_IRQHandler_TIMER1 TIMER1_IRQHandler
- #define HAL_IRQHandler_TIMER2 TIMER2_IRQHandler
- #define HAL_IRQHandler_TIMER3 TIMER3_IRQHandler
- #define HAL_IRQHandler_MCPWM0 MCPWM_IRQHandler
- #define HAL_IRQHandler_I2C0 I2C0_IRQHandler
- #define HAL_IRQHandler_SPI0 SPI0_IRQHandler
- #define HAL_IRQHandler_GPIO0 GPIO_IRQHandler
- #define HAL_IRQHandler_HALL0 HALL_IRQHandler
- #define HAL_IRQHandler_UART0 UART0_IRQHandler
- #define HAL_IRQHandler_UART1 UART1_IRQHandler
- #define HAL_IRQHandler_CMP0 CMP_IRQHandler
- #define HAL_IRQHandler_ADC0 ADC0_IRQHandler
- #define HAL_IRQHandler_WAKE0 WAKE_IRQHandler
- #define HAL_IRQHandler_QEP0 ENCODER0_IRQHandler

- #define HAL_IRQHandler_QEP1 ENCODER1_IRQHandler
- #define HAL_IRQHandler_DSP0 DSP_IRQHandler
- #define HAL_IRQHandler_DMA0 DMA_IRQHandler
- #define HAL_IRQHandler_CAN0 CAN_IRQHandler
- #define HAL_IRQHandler_SIF0 SIF_IRQHandler
- #define HAL_IRQHandler_PWRDN0 PWRDN_IRQHandler
- #define HAL_IRQHandler_INIT0 PWRDN_IRQHandler

枚举

- enum HAL_IRQ_MODULE_x {
 HAL_IRQ_MODULE_NonMaskableInt = -14, HAL_IRQ_MODULE_HardFault = -13, HAL_IRQ_MODULE_SVCall = -5, HAL_IRQ_MODULE_PendSV = -2,
 HAL_IRQ_MODULE_SysTick = -1, HAL_IRQ_MODULE_TIMER0 = 0, HAL_IRQ_MODULE_TIMER1 = 1, HAL_IRQ_MODULE_TIMER2 = 2,
 HAL_IRQ_MODULE_TIMER3 = 3, HAL_IRQ_MODULE_MCPWM0 = 13, HAL_IRQ_MODULE_I2C0 = 6, HAL_IRQ_MODULE_SPI0 = 10,
 HAL_IRQ_MODULE_GPIO0 = 7, HAL_IRQ_MODULE_HALL0 = 9, HAL_IRQ_MODULE_UART0 = 8, HAL_IRQ_MODULE_UART1 = 14,
 HAL_IRQ_MODULE_CMP0 = 15, HAL_IRQ_MODULE_ADC0 = 11, HAL_IRQ_MODULE_WAKE0 = 16, HAL_IRQ_MODULE_QEP0 = 4,
 HAL_IRQ_MODULE_QEP1 = 5, HAL_IRQ_MODULE_DMA0 = 18, HAL_IRQ_MODULE_SIF0 = 20, HAL_IRQ_MODULE_INIT0 = 17,
 HAL_IRQ_MODULE_CAN0 = 19, HAL_IRQ_MODULE_DSP0 = 12 }
 中断序号

函数

- void HAL_IRQ_Reset (void)
 重置中断(等效于关闭中断)
- void HAL_IRQ_Enable (void)
 使能全局中断
- void HAL_IRQ_Disable (void)
 关闭全局中断
- void HAL_IRQ_ModuleEnable (HAL_IRQ_MODULE_x, uint8_t priority)
 使能某个模块的中断
- void HAL_IRQ_ModuleDisable (HAL_IRQ_MODULE_x)
 关闭某个模块的中断
- void HAL_IRQ_ModuleClear (HAL_IRQ_MODULE_x)
 清除某个模块的中断

4.35.1 宏定义说明

4.35.1.1 HAL_IRQHandler_ADC0

```
#define HAL_IRQHandler_ADC0 ADC0_IRQHandler
```

在文件 `hal_lksmcu_irq.h` 第 81 行定义.

4.35.1.2 HAL_IRQHandler.CAN0

```
#define HAL_IRQHandler_CAN0 CAN_IRQHandler
```

在文件 [hal_lksmcu_irq.h](#) 第 87 行定义.

4.35.1.3 HAL_IRQHandler.CMP0

```
#define HAL_IRQHandler_CMP0 CMP_IRQHandler
```

在文件 [hal_lksmcu_irq.h](#) 第 80 行定义.

4.35.1.4 HAL_IRQHandler.DMA0

```
#define HAL_IRQHandler_DMA0 DMA_IRQHandler
```

在文件 [hal_lksmcu_irq.h](#) 第 86 行定义.

4.35.1.5 HAL_IRQHandler.DSP0

```
#define HAL_IRQHandler_DSP0 DSP_IRQHandler
```

在文件 [hal_lksmcu_irq.h](#) 第 85 行定义.

4.35.1.6 HAL_IRQHandler.GPIO0

```
#define HAL_IRQHandler_GPIO0 GPIO_IRQHandler
```

在文件 [hal_lksmcu_irq.h](#) 第 76 行定义.

4.35.1.7 HAL_IRQHandler.HALL0

```
#define HAL_IRQHandler_HALL0 HALL_IRQHandler
```

在文件 [hal_lksmcu_irq.h](#) 第 77 行定义.

4.35.1.8 HAL_IRQHandler.I2C0

```
#define HAL_IRQHandler_I2C0 I2C0_IRQHandler
```

在文件 [hal_lksmcu_irq.h](#) 第 74 行定义.

4.35.1.9 HAL_IRQHandler.INIT0

```
#define HAL_IRQHandler_INIT0 PWRDN_IRQHandler
```

在文件 [hal_lksmcu_irq.h](#) 第 90 行定义.

4.35.1.10 HAL_IRQHandler.MCPWM0

```
#define HAL_IRQHandler_MCPWM0 MCPWM_IRQHandler
```

在文件 [hal_lksmcu_irq.h](#) 第 73 行定义.

4.35.1.11 HAL_IRQHandler.PWRDN0

```
#define HAL_IRQHandler_PWRDN0 PWRDN_IRQHandler
```

在文件 [hal_lksmcu_irq.h](#) 第 89 行定义.

4.35.1.12 HAL_IRQHandler.QEP0

```
#define HAL_IRQHandler_QEP0 ENCODER0_IRQHandler
```

在文件 [hal_lksmcu_irq.h](#) 第 83 行定义.

4.35.1.13 HAL_IRQHandler.QEP1

```
#define HAL_IRQHandler_QEP1 ENCODER1_IRQHandler
```

在文件 [hal_lksmcu_irq.h](#) 第 84 行定义.

4.35.1.14 HAL_IRQHandler.SIF0

```
#define HAL_IRQHandler_SIF0 SIF_IRQHandler
```

在文件 [hal_lksmcu_irq.h](#) 第 88 行定义.

4.35.1.15 HAL_IRQHandler.SPI0

```
#define HAL_IRQHandler_SPI0 SPI0_IRQHandler
```

在文件 [hal_lksmcu_irq.h](#) 第 75 行定义.

4.35.1.16 HAL_IRQHandler.TIMER0

```
#define HAL_IRQHandler_TIMER0 TIMER0_IRQHandler
```

中断函数名

在文件 [hal_lksmcu_irq.h](#) 第 69 行定义.

4.35.1.17 HAL_IRQHandler_TIMER1

```
#define HAL_IRQHandler_TIMER1 TIMER1_IRQHandler
```

在文件 [hal_lksmcu_irq.h](#) 第 70 行定义.

4.35.1.18 HAL_IRQHandler_TIMER2

```
#define HAL_IRQHandler_TIMER2 TIMER2_IRQHandler
```

在文件 [hal_lksmcu_irq.h](#) 第 71 行定义.

4.35.1.19 HAL_IRQHandler_TIMER3

```
#define HAL_IRQHandler_TIMER3 TIMER3_IRQHandler
```

在文件 [hal_lksmcu_irq.h](#) 第 72 行定义.

4.35.1.20 HAL_IRQHandler_UART0

```
#define HAL_IRQHandler_UART0 UART0_IRQHandler
```

在文件 [hal_lksmcu_irq.h](#) 第 78 行定义.

4.35.1.21 HAL_IRQHandler_UART1

```
#define HAL_IRQHandler_UART1 UART1_IRQHandler
```

在文件 [hal_lksmcu_irq.h](#) 第 79 行定义.

4.35.1.22 HAL_IRQHandler_WAKE0

```
#define HAL_IRQHandler_WAKE0 WAKE_IRQHandler
```

在文件 [hal_lksmcu_irq.h](#) 第 82 行定义.

4.35.2 枚举类型说明

4.35.2.1 HAL_IRQ_MODULE_x

```
enum HAL_IRQ_MODULE_x
```

中断序号

枚举值

HAL_IRQ_MODULE_NonMaskableInt	
HAL_IRQ_MODULE_HardFault	
HAL_IRQ_MODULE_SVCall	
HAL_IRQ_MODULE_PendSV	
HAL_IRQ_MODULE_SysTick	
HAL_IRQ_MODULE_TIMER0	
HAL_IRQ_MODULE_TIMER1	
HAL_IRQ_MODULE_TIMER2	
HAL_IRQ_MODULE_TIMER3	
HAL_IRQ_MODULE_MCPWM0	
HAL_IRQ_MODULE_I2C0	
HAL_IRQ_MODULE_SPI0	
HAL_IRQ_MODULE_GPIO0	
HAL_IRQ_MODULE_HALL0	
HAL_IRQ_MODULE_UART0	
HAL_IRQ_MODULE_UART1	
HAL_IRQ_MODULE_CMP0	
HAL_IRQ_MODULE_ADC0	
HAL_IRQ_MODULE_WAKE0	
HAL_IRQ_MODULE_QEP0	
HAL_IRQ_MODULE_QEP1	
HAL_IRQ_MODULE_DMA0	
HAL_IRQ_MODULE_SIF0	
HAL_IRQ_MODULE_INIT0	
HAL_IRQ_MODULE_CAN0	
HAL_IRQ_MODULE_DSP0	

在文件 [hal_lksmcu_irq.h](#) 第 10 行定义.

4.35.3 函数说明

4.35.3.1 HAL_IRQ_Disable()

```
void HAL_IRQ_Disable (
    void )
```

关闭全局中断

参数

<i>irq</i>	中断编号

在文件 [hal_lks32mc08x_irq.c](#) 第 29 行定义.

这是这个函数的调用关系图:



4.35.3.2 HAL_IRQ_Enable()

```
void HAL_IRQ_Enable (
    void )
```

使能全局中断

参数

<i>irq</i>	中断编号

在文件 [hal_lks32mc08x_irq.c](#) 第 20 行定义.

4.35.3.3 HAL_IRQ_ModuleClear()

```
void HAL_IRQ_ModuleClear (
    HAL_IRQ_MODULE_x module )
```

清除某个模块的中断

参数

<i>module</i>	模块编号

在文件 [hal_lks32mc08x_irq.c](#) 第 57 行定义.

这是这个函数的调用关系图:



4.35.3.4 HAL_IRQ_ModuleDisable()

```
void HAL_IRQ_ModuleDisable (
    HAL_IRQ_MODULE_x module )
```

关闭某个模块的中断

参数

<i>module</i>	模块编号

在文件 [hal_lks32mc08x_irq.c](#) 第 49 行定义.

4.35.3.5 HAL_IRQ_ModuleEnable()

```
void HAL_IRQ_ModuleEnable (
    HAL_IRQ_MODULE_x module,
    uint8_t priority )
```

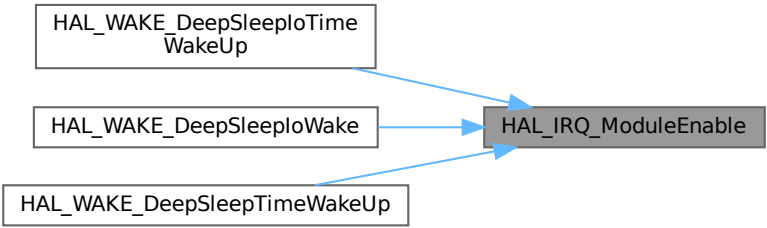
使能某个模块的中断

参数

<i>module</i>	模块编号
<i>priority</i>	中断优先级

在文件 [hal_lks32mc08x_irq.c](#) 第 39 行定义.

这是这个函数的调用关系图:



4.35.3.6 HAL_IRQ_Reset()

```
void HAL_IRQ_Reset (
    void )
```

重置中断(等效于关闭中断)

参数

<i>irq</i>	中断编号

在文件 `hal_lks32mc08x_irq.c` 第 11 行定义.

函数调用图:



4.36 hal_lksmcu_irq.h

[浏览该文件的文档.](#)

```
00001 #ifndef __HAL_LKSMCU_IRQ_H__
00002 #define __HAL_LKSMCU_IRQ_H__
00003
00004 #include "stdint.h"
00005 #include "../hal_lks32mc08x.h"
00006
00010 typedef enum
00011 {
00012     // 来自arm内核的模块
00013     HAL_IRQ_MODULE_NonMaskableInt = -14,
00014     HAL_IRQ_MODULE_HardFault      = -13,
00015     HAL_IRQ_MODULE_SVCall         = -5,
00016     HAL_IRQ_MODULE_PendSV         = -2,
00017     HAL_IRQ_MODULE_SysTick        = -1,
00018     // 芯片外设
00019     HAL_IRQ_MODULE_TIMER0 = 0,
00020     HAL_IRQ_MODULE_TIMER1 = 1,
00021     HAL_IRQ_MODULE_TIMER2 = 2,
00022     HAL_IRQ_MODULE_TIMER3 = 3,
00023     HAL_IRQ_MODULE_MCPWM0 = 13,
00024     HAL_IRQ_MODULE_I2C0   = 6,
00025     HAL_IRQ_MODULE_SPI0   = 10,
00026     HAL_IRQ_MODULE_GPIO0  = 7,
00027     HAL_IRQ_MODULE_HALL0  = 9,
00028     HAL_IRQ_MODULE_UART0  = 8,
00029     HAL_IRQ_MODULE_UART1  = 14,
00030     HAL_IRQ_MODULE_CMP0   = 15,
00031     HAL_IRQ_MODULE_ADC0   = 11,
00032     HAL_IRQ_MODULE_WAKE0  = 16,
00033     HAL_IRQ_MODULE_QEP0   = 4,
00034     HAL_IRQ_MODULE_QEP1   = 5,
00035     HAL_IRQ_MODULE_DMA0   = 18,
00036     HAL_IRQ_MODULE_SIF0   = 20,
00037     HAL_IRQ_MODULE_INIT0  = 17,
00038     HAL_IRQ_MODULE_CAN0   = 19,
00039     HAL_IRQ_MODULE_DSP0   = 12,
00040
00041 #ifdef HAL_TEST_NOT_SUPPORTED
00042     HAL_IRQ_MODULE_MCPWM0_1, // 不支持
00043     HAL_IRQ_MODULE_SW0,     // 不支持
00044     HAL_IRQ_MODULE_UART2,   // 不支持
00045     HAL_IRQ_MODULE_CMP1,    // 不支持
00046     HAL_IRQ_MODULE_ADC1,    // 不支持
00047     HAL_IRQ_MODULE_CL0,     // 不支持
00048     HAL_IRQ_MODULE_TIMER4,  // 不支持
00049     HAL_IRQ_MODULE_MCPWM1,  // 不支持
00050     HAL_IRQ_MODULE_MCPWM1_1, // 不支持
00051     HAL_IRQ_MODULE_I2C1,    // 不支持
00052     HAL_IRQ_MODULE_SPI1,    // 不支持

```

```

00053     HAL_IRQ_MODULE_HALL1,    // 不支持
00054     HAL_IRQ_MODULE_CMP2,      // 不支持
00055     HAL_IRQ_MODULE_CMP3,      // 不支持
00056     HAL_IRQ_MODULE_CMP4,      // 不支持
00057     HAL_IRQ_MODULE_CMP5,      // 不支持
00058     HAL_IRQ_MODULE_ADC2,      // 不支持
00059     HAL_IRQ_MODULE_QEP2,      // 不支持
00060     HAL_IRQ_MODULE_QEP3,      // 不支持
00061     HAL_IRQ_MODULE_FMAC0,     // 不支持
00062     HAL_IRQ_MODULE_WWDG0,     // 不支持
00063 #endif
00064 } HAL_IRQ_MODULE_x;
00065
00069 #define HAL_IRQHandler_TIMER0  TIMER0_IRQHandler
00070 #define HAL_IRQHandler_TIMER1  TIMER1_IRQHandler
00071 #define HAL_IRQHandler_TIMER2  TIMER2_IRQHandler
00072 #define HAL_IRQHandler_TIMER3  TIMER3_IRQHandler
00073 #define HAL_IRQHandler_MCPWM0  MCPWM_IRQHandler
00074 #define HAL_IRQHandler_I2C0    I2C0_IRQHandler
00075 #define HAL_IRQHandler_SPI0    SPI0_IRQHandler
00076 #define HAL_IRQHandler_GPIO0  GPIO_IRQHandler
00077 #define HAL_IRQHandler_HALL0   HALL_IRQHandler
00078 #define HAL_IRQHandler_UART0   UART0_IRQHandler
00079 #define HAL_IRQHandler_UART1   UART1_IRQHandler
00080 #define HAL_IRQHandler_CMP0    CMP_IRQHandler
00081 #define HAL_IRQHandler_ADC0    ADC0_IRQHandler
00082 #define HAL_IRQHandler_WAKE0   WAKE_IRQHandler
00083 #define HAL_IRQHandler_QEP0    ENCODER0_IRQHandler
00084 #define HAL_IRQHandler_QEP1    ENCODER1_IRQHandler
00085 #define HAL_IRQHandler_DSP0    DSP_IRQHandler
00086 #define HAL_IRQHandler_DMA0    DMA_IRQHandler
00087 #define HAL_IRQHandler_CAN0    CAN_IRQHandler
00088 #define HAL_IRQHandler_SIF0    SIF_IRQHandler
00089 #define HAL_IRQHandler_PWRDN0  PWRDN_IRQHandler
00090 #define HAL_IRQHandler_INIT0   PWRDN_IRQHandler
00091
00092 #ifdef HAL_TEST_NOT_SUPPORTED
00093 #define HAL_IRQHandler_SW0      HAL_IRQHandler_SW0_Null    // 不支持
00094 #define HAL_IRQHandler_CL0      HAL_IRQHandler_CL0_Null    // 不支持
00095 #define HAL_IRQHandler_UART2    HAL_IRQHandler_UART2_Null  // 不支持
00096 #define HAL_IRQHandler_CMP1     HAL_IRQHandler_CMP1_Null   // 不支持
00097 #define HAL_IRQHandler_MCPWM0_1 HAL_IRQHandler_MCPWM0_1_Null // 不支持
00098 #define HAL_IRQHandler_ADC1     HAL_IRQHandler_ADC1_Null   // 不支持
00099 #define HAL_IRQHandler_I2C1     HAL_IRQHandler_I2C1_Null   // 不支持
00100 #define HAL_IRQHandler_HALL1    HAL_IRQHandler_HALL1_Null  // 不支持
00101 #define HAL_IRQHandler_CMP2     HAL_IRQHandler_CMP2_Null   // 不支持
00102 #define HAL_IRQHandler_CMP3     HAL_IRQHandler_CMP3_Null   // 不支持
00103 #define HAL_IRQHandler_CMP4     HAL_IRQHandler_CMP4_Null   // 不支持
00104 #define HAL_IRQHandler_CMP5     HAL_IRQHandler_CMP5_Null   // 不支持
00105 #define HAL_IRQHandler_MCPWM1    HAL_IRQHandler_MCPWM1_Null // 不支持
00106 #define HAL_IRQHandler_MCPWM1_1 HAL_IRQHandler_MCPWM1_1_Null // 不支持
00107 #define HAL_IRQHandler_TIMER4    HAL_IRQHandler_TIMER4_Null // 不支持
00108 #define HAL_IRQHandler_QEP2     HAL_IRQHandler_QEP2_Null   // 不支持
00109 #define HAL_IRQHandler_QEP3     HAL_IRQHandler_QEP3_Null   // 不支持
00110 #define HAL_IRQHandler_ADC2     HAL_IRQHandler_ADC2_Null   // 不支持
00111 #define HAL_IRQHandler_SPI1     HAL_IRQHandler_SPI1_Null   // 不支持
00112 #define HAL_IRQHandler_WWDG0    HAL_IRQHandler_WWDG0_Null  // 不支持
00113 #define HAL_IRQHandler_FMAC0     HAL_IRQHandler_FMAC0_Null  // 不支持
00114 #endif
00115
00116 void HAL_IRQ_Reset(void); // 重置中断(等效于关闭中断)
00117 void HAL_IRQ_Enable(void); // 使能全局中断
00118 void HAL_IRQ_Disable(void); // 关闭全局中断
00119
00120 void HAL_IRQ_ModuleEnable(HAL_IRQ_MODULE_x, uint8_t priority); // 使能某个模块的中断
00121 void HAL_IRQ_ModuleDisable(HAL_IRQ_MODULE_x); // 关闭某个模块的中断
00122
00123 void HAL_IRQ_ModuleClear(HAL_IRQ_MODULE_x);
00124 #endif

```

4.37 include/hal_lksmcu_irqfun.h 文件参考

```
#include "../hal_lks32mc08x.h"
```

hal_lksmcu_irqfun.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



函数

- uint8_t HAL_HALL_IrqChange (HAL_HALL_x)
中断处理函数
- uint8_t HAL_HALL_IrqChangeFlg (HAL_HALL_x hall)
中断信号 (和 HAL_HALL_IrqChange 的区别是不判断中断使能)
- void HAL_HALL_IrqChangeEnable (HAL_HALL_x)
中断使能
- void HAL_HALL_IrqChangeDisable (HAL_HALL_x hall)
中断禁止
- void HAL_HALL_IrqChangeClear (HAL_HALL_x hall)
中断清除
- uint8_t HAL_HALL_IrqOver (HAL_HALL_x)
中断处理函数
- uint8_t HAL_HALL_IrqOverFlg (HAL_HALL_x hall)
中断信号 (和 HAL_HALL_IrqOver 的区别是不判断中断使能)
- void HAL_HALL_IrqOverEnable (HAL_HALL_x)
中断使能
- void HAL_HALL_IrqOverDisable (HAL_HALL_x hall)
中断禁止
- void HAL_HALL_IrqOverClear (HAL_HALL_x hall)
中断清除
- uint8_t HAL_ADC_IrqSamp1 (HAL_ADC_x)
中断处理函数
- uint8_t HAL_ADC_IrqSamp1Flg (HAL_ADC_x adc)
中断信号 (和 HAL_ADC_IrqSamp1 的区别是不判断中断使能)
- void HAL_ADC_IrqSamp1Enable (HAL_ADC_x)
中断使能
- void HAL_ADC_IrqSamp1Disable (HAL_ADC_x adc)
中断禁止
- void HAL_ADC_IrqSamp1Clear (HAL_ADC_x adc)
中断清除

- uint8_t HAL_ADC_IrqSamp2 (HAL_ADC_x)
中断处理函数
- uint8_t HAL_ADC_IrqSamp2Flg (HAL_ADC_x adc)
中断信号 (和 HAL_ADC_IrqSamp2 的区别是不判断中断使能)
- void HAL_ADC_IrqSamp2Enable (HAL_ADC_x)
中断使能
- void HAL_ADC_IrqSamp2Disable (HAL_ADC_x adc)
中断禁止
- void HAL_ADC_IrqSamp2Clear (HAL_ADC_x adc)
中断清除
- uint8_t HAL_MCPWM_IrqOverflow (HAL_MCPWM_x)
中断处理函数
- uint8_t HAL_MCPWM_IrqOverflowFlg (HAL_MCPWM_x mcpwmx)
中断信号 (和 HAL_MCPWM_IrqOverflow 的区别是不判断中断使能)
- void HAL_MCPWM_IrqOverflowEnable (HAL_MCPWM_x)
中断使能
- void HAL_MCPWM_IrqOverflowDisable (HAL_MCPWM_x mcpwmx)
中断禁止
- void HAL_MCPWM_IrqOverflowClear (HAL_MCPWM_x mcpwmx)
中断清除
- uint8_t HAL_MCPWM_IrqZeroCross (HAL_MCPWM_x)
中断处理函数
- uint8_t HAL_MCPWM_IrqZeroCrossFlg (HAL_MCPWM_x mcpwmx)
中断信号 (和 HAL_MCPWM_IrqZeroCross 的区别是不判断中断使能)
- void HAL_MCPWM_IrqZeroCrossEnable (HAL_MCPWM_x)
中断使能
- void HAL_MCPWM_IrqZeroCrossDisable (HAL_MCPWM_x mcpwmx)
中断禁止
- void HAL_MCPWM_IrqZeroCrossClear (HAL_MCPWM_x mcpwmx)
中断清除
- uint8_t HAL_MCPWM_IrqFail0 (HAL_MCPWM_x)
中断处理函数
- uint8_t HAL_MCPWM_IrqFail0Flg (HAL_MCPWM_x mcpwmx)
中断信号 (和 HAL_MCPWM_IrqFail0 的区别是不判断中断使能)
- void HAL_MCPWM_IrqFail0Enable (HAL_MCPWM_x)
中断使能
- void HAL_MCPWM_IrqFail0Disable (HAL_MCPWM_x mcpwmx)
中断禁止
- void HAL_MCPWM_IrqFail0Clear (HAL_MCPWM_x mcpwmx)
中断清除
- uint8_t HAL_MCPWM_IrqFail1 (HAL_MCPWM_x)
中断处理函数
- uint8_t HAL_MCPWM_IrqFail1Flg (HAL_MCPWM_x mcpwmx)
中断信号 (和 HAL_MCPWM_IrqFail1 的区别是不判断中断使能)
- void HAL_MCPWM_IrqFail1Enable (HAL_MCPWM_x)
中断使能
- void HAL_MCPWM_IrqFail1Disable (HAL_MCPWM_x mcpwmx)
中断禁止
- void HAL_MCPWM_IrqFail1Clear (HAL_MCPWM_x mcpwmx)
中断清除
- uint8_t HAL_MCPWM_IrqFail (HAL_MCPWM_x)

中断处理函数

- uint8_t HAL_MCPWM_IrqFailFlg (HAL_MCPWM_x mcpwmx)
中断信号 (和 HAL_MCPWM_IrqFail 的区别是不判断中断使能)
- void HAL_MCPWM_IrqFailEnable (HAL_MCPWM_x)
中断使能
- void HAL_MCPWM_IrqFailDisable (HAL_MCPWM_x mcpwmx)
中断禁止
- void HAL_MCPWM_IrqFailClear (HAL_MCPWM_x mcpwmx)
中断清除
- uint8_t HAL_CMP_IrqCompare (HAL_CMP_x)
中断处理函数
- uint8_t HAL_CMP_IrqCompareFlg (HAL_CMP_x cmp_x)
中断信号 (和 HAL_CMP_IrqCompare 的区别是不判断中断使能)
- void HAL_CMP_IrqCompareEnable (HAL_CMP_x)
中断使能
- void HAL_CMP_IrqCompareDisable (HAL_CMP_x cmp_x)
中断禁止
- void HAL_CMP_IrqCompareClear (HAL_CMP_x cmp_x)
中断清除
- uint8_t HAL_TIMER_IrqOverflow (HAL_TIMER_x)
中断处理函数
- uint8_t HAL_TIMER_IrqOverflowFlg (HAL_TIMER_x timer_x)
中断信号 (和 HAL_TIMER_IrqOverflow 的区别是不判断中断使能)
- void HAL_TIMER_IrqOverflowEnable (HAL_TIMER_x)
中断使能
- void HAL_TIMER_IrqOverflowDisable (HAL_TIMER_x timer_x)
中断禁止
- void HAL_TIMER_IrqOverflowClear (HAL_TIMER_x timer_x)
中断清除
- uint8_t HAL_TIMER_IrqCompare0 (HAL_TIMER_x)
中断处理函数
- uint8_t HAL_TIMER_IrqCompare0Flg (HAL_TIMER_x timer_x)
中断信号 (和 HAL_TIMER_IrqCompare0 的区别是不判断中断使能)
- void HAL_TIMER_IrqCompare0Enable (HAL_TIMER_x)
中断使能
- void HAL_TIMER_IrqCompare0Disable (HAL_TIMER_x timer_x)
中断禁止
- void HAL_TIMER_IrqCompare0Clear (HAL_TIMER_x timer_x)
中断清除
- uint8_t HAL_TIMER_IrqCompare1 (HAL_TIMER_x)
中断处理函数
- uint8_t HAL_TIMER_IrqCompare1Flg (HAL_TIMER_x timer_x)
中断信号 (和 HAL_TIMER_IrqCompare1 的区别是不判断中断使能)
- void HAL_TIMER_IrqCompare1Enable (HAL_TIMER_x)
中断使能
- void HAL_TIMER_IrqCompare1Disable (HAL_TIMER_x timer_x)
中断禁止
- void HAL_TIMER_IrqCompare1Clear (HAL_TIMER_x timer_x)
中断清除
- uint8_t HAL_UART_IrqTxDone (HAL_UART_x)
中断处理函数

- `uint8_t HAL_UART_IrqTxDoneFlg (HAL_UART_x uart_x)`
中断信号（和 `HAL_UART_IrqTxDone` 的区别是不判断中断使能）
- `void HAL_UART_IrqTxDoneEnable (HAL_UART_x)`
中断使能
- `void HAL_UART_IrqTxDoneDisable (HAL_UART_x uart_x)`
中断禁止
- `void HAL_UART_IrqTxDoneClear (HAL_UART_x uart_x)`
中断清除
- `uint8_t HAL_UART_IrqRxDone (HAL_UART_x)`
中断处理函数
- `uint8_t HAL_UART_IrqRxDoneFlg (HAL_UART_x uart_x)`
中断信号（和 `HAL_UART_IrqRxDone` 的区别是不判断中断使能）
- `void HAL_UART_IrqRxDoneEnable (HAL_UART_x)`
中断使能
- `void HAL_UART_IrqRxDoneDisable (HAL_UART_x uart_x)`
中断禁止
- `void HAL_UART_IrqRxDoneClear (HAL_UART_x uart_x)`
中断清除
- `uint8_t HAL_UART_IrqStopErr (HAL_UART_x)`
中断处理函数
- `uint8_t HAL_UART_IrqStopErrFlg (HAL_UART_x uart_x)`
中断信号（和 `HAL_UART_IrqStopErr` 的区别是不判断中断使能）
- `void HAL_UART_IrqStopErrEnable (HAL_UART_x)`
中断使能
- `void HAL_UART_IrqStopErrDisable (HAL_UART_x uart_x)`
中断禁止
- `void HAL_UART_IrqStopErrClear (HAL_UART_x uart_x)`
中断清除
- `uint8_t HAL_UART_IrqCheckErr (HAL_UART_x)`
中断处理函数
- `uint8_t HAL_UART_IrqCheckErrFlg (HAL_UART_x uart_x)`
中断信号（和 `HAL_UART_IrqCheckErr` 的区别是不判断中断使能）
- `void HAL_UART_IrqCheckErrEnable (HAL_UART_x)`
中断使能
- `void HAL_UART_IrqCheckErrDisable (HAL_UART_x uart_x)`
中断禁止
- `void HAL_UART_IrqCheckErrClear (HAL_UART_x uart_x)`
中断清除
- `uint8_t HAL_CAN_IrqRxDone (HAL_CAN_x)`
中断处理函数
- `uint8_t HAL_CAN_IrqRxDoneFlg (HAL_CAN_x can_x)`
中断信号（和 `HAL_CAN_IrqRxDone` 的区别是不判断中断使能）
- `void HAL_CAN_IrqRxDoneEnable (HAL_CAN_x)`
中断使能
- `void HAL_CAN_IrqRxDoneDisable (HAL_CAN_x can_x)`
中断禁止
- `void HAL_CAN_IrqRxDoneClear (HAL_CAN_x can_x)`
中断清除
- `uint8_t HAL_CAN_IrqTxDone (HAL_CAN_x)`
中断处理函数
- `uint8_t HAL_CAN_IrqTxDoneFlg (HAL_CAN_x can_x)`

- 中断信号（和 `HAL_CAN_IrqTxDone` 的区别是不判断中断使能）
- void `HAL_CAN_IrqTxDoneEnable` (`HAL_CAN_x`)
中断使能
- void `HAL_CAN_IrqTxDoneDisable` (`HAL_CAN_x can_x`)
中断禁止
- void `HAL_CAN_IrqTxDoneClear` (`HAL_CAN_x can_x`)
中断清除
- uint8_t `HAL_CAN_IrqAll` (`HAL_CAN_x`)
中断处理函数
- uint8_t `HAL_CAN_IrqAllFlg` (`HAL_CAN_x can_x`)
中断信号（和 `HAL_CAN_IrqAll` 的区别是不判断中断使能）
- void `HAL_CAN_IrqAllEnable` (`HAL_CAN_x`)
中断使能
- void `HAL_CAN_IrqAllDisable` (`HAL_CAN_x can_x`)
中断禁止
- void `HAL_CAN_IrqAllClear` (`HAL_CAN_x can_x`)
中断清除
- uint8_t `HAL_UART_IrqTxBufEmpty` (`HAL_UART_x`)
中断处理函数
- uint8_t `HAL_UART_IrqTxBufEmptyFlg` (`HAL_UART_x uart_x`)
中断信号（和 `HAL_UART_IrqTxBufEmpty` 的区别是不判断中断使能）
- void `HAL_UART_IrqTxBufEmptyEnable` (`HAL_UART_x`)
中断使能
- void `HAL_UART_IrqTxBufEmptyDisable` (`HAL_UART_x uart_x`)
中断禁止
- void `HAL_UART_IrqTxBufEmptyClear` (`HAL_UART_x uart_x`)
中断清除
- uint8_t `HAL_ADC_IrqSamp3` (`HAL_ADC_x`)
中断处理函数
- uint8_t `HAL_ADC_IrqSamp3Flg` (`HAL_ADC_x adc`)
中断信号（和 `HAL_ADC_IrqSamp3` 的区别是不判断中断使能）
- void `HAL_ADC_IrqSamp3Enable` (`HAL_ADC_x`)
中断使能
- void `HAL_ADC_IrqSamp3Disable` (`HAL_ADC_x adc`)
中断禁止
- void `HAL_ADC_IrqSamp3Clear` (`HAL_ADC_x adc`)
中断清除
- uint8_t `HAL_ADC_IrqSamp4` (`HAL_ADC_x`)
中断处理函数
- uint8_t `HAL_ADC_IrqSamp4Flg` (`HAL_ADC_x adc`)
中断信号（和 `HAL_ADC_IrqSamp4` 的区别是不判断中断使能）
- void `HAL_ADC_IrqSamp4Enable` (`HAL_ADC_x`)
中断使能
- void `HAL_ADC_IrqSamp4Disable` (`HAL_ADC_x adc`)
中断禁止
- void `HAL_ADC_IrqSamp4Clear` (`HAL_ADC_x adc`)
中断清除
- uint8_t `HAL_ADC_IrqWatchdog` (`HAL_ADC_x`)
中断处理函数
- uint8_t `HAL_ADC_IrqWatchdogFlg` (`HAL_ADC_x adc`)
中断信号（和 `HAL_ADC_IrqWatchdog` 的区别是不判断中断使能）

- void [HAL_ADC_IrqWatchdogEnable \(HAL_ADC_x\)](#)
中断使能
- void [HAL_ADC_IrqWatchdogDisable \(HAL_ADC_x adc\)](#)
中断禁止
- void [HAL_ADC_IrqWatchdogClear \(HAL_ADC_x adc\)](#)
中断清除
- uint8_t [HAL_ADC_IrqHardErr \(HAL_ADC_x\)](#)
中断处理函数
- uint8_t [HAL_ADC_IrqHardErrFlg \(HAL_ADC_x adc\)](#)
中断信号（和 [HAL_ADC_IrqHardErr](#) 的区别是不判断中断使能）
- void [HAL_ADC_IrqHardErrEnable \(HAL_ADC_x\)](#)
中断使能
- void [HAL_ADC_IrqHardErrDisable \(HAL_ADC_x adc\)](#)
中断禁止
- void [HAL_ADC_IrqHardErrClear \(HAL_ADC_x adc\)](#)
中断清除
- uint8_t [HAL_ADC_IrqSoftErr \(HAL_ADC_x\)](#)
中断处理函数
- uint8_t [HAL_ADC_IrqSoftErrFlg \(HAL_ADC_x adc\)](#)
中断信号（和 [HAL_ADC_IrqSoftErr](#) 的区别是不判断中断使能）
- void [HAL_ADC_IrqSoftErrEnable \(HAL_ADC_x\)](#)
中断使能
- void [HAL_ADC_IrqSoftErrDisable \(HAL_ADC_x adc\)](#)
中断禁止
- void [HAL_ADC_IrqSoftErrClear \(HAL_ADC_x adc\)](#)
中断清除
- uint8_t [HAL_MCPWM_IrqUpdate \(HAL_MCPWM_x\)](#)
中断处理函数
- uint8_t [HAL_MCPWM_IrqUpdateFlg \(HAL_MCPWM_x mcpwmx\)](#)
中断信号（和 [HAL_MCPWM_IrqUpdate](#) 的区别是不判断中断使能）
- void [HAL_MCPWM_IrqUpdateEnable \(HAL_MCPWM_x\)](#)
中断使能
- void [HAL_MCPWM_IrqUpdateDisable \(HAL_MCPWM_x mcpwmx\)](#)
中断禁止
- void [HAL_MCPWM_IrqUpdateClear \(HAL_MCPWM_x mcpwmx\)](#)
中断清除
- uint8_t [HAL_MCPWM_IrqTmr0 \(HAL_MCPWM_x\)](#)
中断处理函数
- uint8_t [HAL_MCPWM_IrqTmr0Flg \(HAL_MCPWM_x mcpwmx\)](#)
中断信号（和 [HAL_MCPWM_IrqTmr0](#) 的区别是不判断中断使能）
- void [HAL_MCPWM_IrqTmr0Enable \(HAL_MCPWM_x\)](#)
中断使能
- void [HAL_MCPWM_IrqTmr0Disable \(HAL_MCPWM_x mcpwmx\)](#)
中断禁止
- void [HAL_MCPWM_IrqTmr0Clear \(HAL_MCPWM_x mcpwmx\)](#)
中断清除
- uint8_t [HAL_MCPWM_IrqTmr1 \(HAL_MCPWM_x\)](#)
中断处理函数
- uint8_t [HAL_MCPWM_IrqTmr1Flg \(HAL_MCPWM_x mcpwmx\)](#)
中断信号（和 [HAL_MCPWM_IrqTmr1](#) 的区别是不判断中断使能）
- void [HAL_MCPWM_IrqTmr1Enable \(HAL_MCPWM_x\)](#)

- 中断使能
- void [HAL_MCPWM_IrqTmr1Disable](#) (HAL_MCPWM_x mcpwmx)
- 中断禁止
- void [HAL_MCPWM_IrqTmr1Clear](#) (HAL_MCPWM_x mcpwmx)
- 中断清除
- uint8_t [HAL_MCPWM_IrqTmr2](#) (HAL_MCPWM_x)
- 中断处理函数
- uint8_t [HAL_MCPWM_IrqTmr2Flg](#) (HAL_MCPWM_x mcpwmx)
- 中断信号 (和 [HAL_MCPWM_IrqTmr2](#) 的区别是不判断中断使能)
- void [HAL_MCPWM_IrqTmr2Enable](#) (HAL_MCPWM_x)
- 中断使能
- void [HAL_MCPWM_IrqTmr2Disable](#) (HAL_MCPWM_x mcpwmx)
- 中断禁止
- void [HAL_MCPWM_IrqTmr2Clear](#) (HAL_MCPWM_x mcpwmx)
- 中断清除
- uint8_t [HAL_MCPWM_IrqTmr3](#) (HAL_MCPWM_x)
- 中断处理函数
- uint8_t [HAL_MCPWM_IrqTmr3Flg](#) (HAL_MCPWM_x mcpwmx)
- 中断信号 (和 [HAL_MCPWM_IrqTmr3](#) 的区别是不判断中断使能)
- void [HAL_MCPWM_IrqTmr3Enable](#) (HAL_MCPWM_x)
- 中断使能
- void [HAL_MCPWM_IrqTmr3Disable](#) (HAL_MCPWM_x mcpwmx)
- 中断禁止
- void [HAL_MCPWM_IrqTmr3Clear](#) (HAL_MCPWM_x mcpwmx)
- 中断清除
- uint8_t [HAL_MCPWM_IrqTH00](#) (HAL_MCPWM_x)
- 中断处理函数
- uint8_t [HAL_MCPWM_IrqTH00Flg](#) (HAL_MCPWM_x mcpwmx)
- 中断信号 (和 [HAL_MCPWM_IrqTH00](#) 的区别是不判断中断使能)
- void [HAL_MCPWM_IrqTH00Enable](#) (HAL_MCPWM_x)
- 中断使能
- void [HAL_MCPWM_IrqTH00Disable](#) (HAL_MCPWM_x mcpwmx)
- 中断禁止
- void [HAL_MCPWM_IrqTH00Clear](#) (HAL_MCPWM_x mcpwmx)
- 中断清除
- uint8_t [HAL_MCPWM_IrqTH01](#) (HAL_MCPWM_x)
- 中断处理函数
- uint8_t [HAL_MCPWM_IrqTH01Flg](#) (HAL_MCPWM_x mcpwmx)
- 中断信号 (和 [HAL_MCPWM_IrqTH01](#) 的区别是不判断中断使能)
- void [HAL_MCPWM_IrqTH01Enable](#) (HAL_MCPWM_x)
- 中断使能
- void [HAL_MCPWM_IrqTH01Disable](#) (HAL_MCPWM_x mcpwmx)
- 中断禁止
- void [HAL_MCPWM_IrqTH01Clear](#) (HAL_MCPWM_x mcpwmx)
- 中断清除
- uint8_t [HAL_MCPWM_IrqTH10](#) (HAL_MCPWM_x)
- 中断处理函数
- uint8_t [HAL_MCPWM_IrqTH10Flg](#) (HAL_MCPWM_x mcpwmx)
- 中断信号 (和 [HAL_MCPWM_IrqTH10](#) 的区别是不判断中断使能)
- void [HAL_MCPWM_IrqTH10Enable](#) (HAL_MCPWM_x)
- 中断使能

- void [HAL_MCPWM_IrqTH10Disable](#) (HAL_MCPWM_x mcpwmx)
中断禁止
- void [HAL_MCPWM_IrqTH10Clear](#) (HAL_MCPWM_x mcpwmx)
中断清除
- uint8_t [HAL_MCPWM_IrqTH11](#) (HAL_MCPWM_x)
中断处理函数
- uint8_t [HAL_MCPWM_IrqTH11Flg](#) (HAL_MCPWM_x mcpwmx)
中断信号 (和 [HAL_MCPWM_IrqTH11](#) 的区别是不判断中断使能)
- void [HAL_MCPWM_IrqTH11Enable](#) (HAL_MCPWM_x)
中断使能
- void [HAL_MCPWM_IrqTH11Disable](#) (HAL_MCPWM_x mcpwmx)
中断禁止
- void [HAL_MCPWM_IrqTH11Clear](#) (HAL_MCPWM_x mcpwmx)
中断清除
- uint8_t [HAL_MCPWM_IrqTH20](#) (HAL_MCPWM_x)
中断处理函数
- uint8_t [HAL_MCPWM_IrqTH20Flg](#) (HAL_MCPWM_x mcpwmx)
中断信号 (和 [HAL_MCPWM_IrqTH20](#) 的区别是不判断中断使能)
- void [HAL_MCPWM_IrqTH20Enable](#) (HAL_MCPWM_x)
中断使能
- void [HAL_MCPWM_IrqTH20Disable](#) (HAL_MCPWM_x mcpwmx)
中断禁止
- void [HAL_MCPWM_IrqTH20Clear](#) (HAL_MCPWM_x mcpwmx)
中断清除
- uint8_t [HAL_MCPWM_IrqTH21](#) (HAL_MCPWM_x)
中断处理函数
- uint8_t [HAL_MCPWM_IrqTH21Flg](#) (HAL_MCPWM_x mcpwmx)
中断信号 (和 [HAL_MCPWM_IrqTH21](#) 的区别是不判断中断使能)
- void [HAL_MCPWM_IrqTH21Enable](#) (HAL_MCPWM_x)
中断使能
- void [HAL_MCPWM_IrqTH21Disable](#) (HAL_MCPWM_x mcpwmx)
中断禁止
- void [HAL_MCPWM_IrqTH21Clear](#) (HAL_MCPWM_x mcpwmx)
中断清除
- uint8_t [HAL_MCPWM_IrqTH30](#) (HAL_MCPWM_x)
中断处理函数
- uint8_t [HAL_MCPWM_IrqTH30Flg](#) (HAL_MCPWM_x mcpwmx)
中断信号 (和 [HAL_MCPWM_IrqTH30](#) 的区别是不判断中断使能)
- void [HAL_MCPWM_IrqTH30Enable](#) (HAL_MCPWM_x)
中断使能
- void [HAL_MCPWM_IrqTH30Disable](#) (HAL_MCPWM_x mcpwmx)
中断禁止
- void [HAL_MCPWM_IrqTH30Clear](#) (HAL_MCPWM_x mcpwmx)
中断清除
- uint8_t [HAL_MCPWM_IrqTH31](#) (HAL_MCPWM_x)
中断处理函数
- uint8_t [HAL_MCPWM_IrqTH31Flg](#) (HAL_MCPWM_x mcpwmx)
中断信号 (和 [HAL_MCPWM_IrqTH31](#) 的区别是不判断中断使能)
- void [HAL_MCPWM_IrqTH31Enable](#) (HAL_MCPWM_x)
中断使能
- void [HAL_MCPWM_IrqTH31Disable](#) (HAL_MCPWM_x mcpwmx)

- 中断禁止
- void [HAL_MCPWM_IrqTH31Clear](#) (HAL_MCPWM_x mcpwmx)
- 中断清除
- uint8_t [HAL_MCPWM_IrqShort](#) (HAL_MCPWM_x)
- 中断处理函数
- uint8_t [HAL_MCPWM_IrqShortFlg](#) (HAL_MCPWM_x mcpwmx)
- 中断信号（和 [HAL_MCPWM_IrqShort](#) 的区别是不判断中断使能）
- void [HAL_MCPWM_IrqShortEnable](#) (HAL_MCPWM_x)
- 中断使能
- void [HAL_MCPWM_IrqShortDisable](#) (HAL_MCPWM_x mcpwmx)
- 中断禁止
- void [HAL_MCPWM_IrqShortClear](#) (HAL_MCPWM_x mcpwmx)
- 中断清除
- uint8_t [HAL_I2C_IrqDone](#) (HAL_I2C_x)
- 中断处理函数
- uint8_t [HAL_I2C_IrqDoneFlg](#) (HAL_I2C_x i2c_x)
- 中断信号（和 [HAL_I2C_IrqDone](#) 的区别是不判断中断使能）
- void [HAL_I2C_IrqDoneEnable](#) (HAL_I2C_x)
- 中断使能
- void [HAL_I2C_IrqDoneDisable](#) (HAL_I2C_x i2c_x)
- 中断禁止
- void [HAL_I2C_IrqDoneClear](#) (HAL_I2C_x i2c_x)
- 中断清除
- uint8_t [HAL_I2C_IrqBusErr](#) (HAL_I2C_x)
- 中断处理函数
- uint8_t [HAL_I2C_IrqBusErrFlg](#) (HAL_I2C_x i2c_x)
- 中断信号（和 [HAL_I2C_IrqBusErr](#) 的区别是不判断中断使能）
- void [HAL_I2C_IrqBusErrEnable](#) (HAL_I2C_x)
- 中断使能
- void [HAL_I2C_IrqBusErrDisable](#) (HAL_I2C_x i2c_x)
- 中断禁止
- void [HAL_I2C_IrqBusErrClear](#) (HAL_I2C_x i2c_x)
- 中断清除
- uint8_t [HAL_I2C_IrqStop](#) (HAL_I2C_x)
- 中断处理函数
- uint8_t [HAL_I2C_IrqStopFlg](#) (HAL_I2C_x i2c_x)
- 中断信号（和 [HAL_I2C_IrqStop](#) 的区别是不判断中断使能）
- void [HAL_I2C_IrqStopEnable](#) (HAL_I2C_x)
- 中断使能
- void [HAL_I2C_IrqStopDisable](#) (HAL_I2C_x i2c_x)
- 中断禁止
- void [HAL_I2C_IrqStopClear](#) (HAL_I2C_x i2c_x)
- 中断清除
- uint8_t [HAL_SPI_IrqDone](#) (HAL_SPI_x)
- 中断处理函数
- uint8_t [HAL_SPI_IrqDoneFlg](#) (HAL_SPI_x spi_x)
- 中断信号（和 [HAL_SPI_IrqDone](#) 的区别是不判断中断使能）
- void [HAL_SPI_IrqDoneEnable](#) (HAL_SPI_x)
- 中断使能
- void [HAL_SPI_IrqDoneDisable](#) (HAL_SPI_x spi_x)
- 中断禁止

- void [HAL_SPI_IrqDoneClear](#) (HAL_SPI_x spi_x)
中断清除
- uint8_t [HAL_SPI_IrqErr](#) (HAL_SPI_x)
中断处理函数
- uint8_t [HAL_SPI_IrqErrFlg](#) (HAL_SPI_x spi_x)
中断信号（和 [HAL_SPI_IrqErr](#) 的区别是不判断中断使能）
- void [HAL_SPI_IrqErrEnable](#) (HAL_SPI_x)
中断使能
- void [HAL_SPI_IrqErrDisable](#) (HAL_SPI_x spi_x)
中断禁止
- void [HAL_SPI_IrqErrClear](#) (HAL_SPI_x spi_x)
中断清除
- uint8_t [HAL_SPI_IrqOverflow](#) (HAL_SPI_x)
中断处理函数
- uint8_t [HAL_SPI_IrqOverflowFlg](#) (HAL_SPI_x spi_x)
中断信号（和 [HAL_SPI_IrqOverflow](#) 的区别是不判断中断使能）
- void [HAL_SPI_IrqOverflowEnable](#) (HAL_SPI_x)
中断使能
- void [HAL_SPI_IrqOverflowDisable](#) (HAL_SPI_x spi_x)
中断禁止
- void [HAL_SPI_IrqOverflowClear](#) (HAL_SPI_x spi_x)
中断清除
- uint8_t [HAL_DMA_IrqDone](#) (HAL_DMA_x)
中断处理函数
- uint8_t [HAL_DMA_IrqDoneFlg](#) (HAL_DMA_x dma_x)
中断信号（同 [HAL_DMA_IrqDone](#)）
- void [HAL_DMA_IrqDoneEnable](#) (HAL_DMA_x)
中断使能（本模块无独立使能位，由NVIC控制）
- void [HAL_DMA_IrqDoneDisable](#) (HAL_DMA_x dma_x)
中断禁止（本模块无独立使能位，由NVIC控制）
- void [HAL_DMA_IrqDoneClear](#) (HAL_DMA_x dma_x)
中断清除
- void [HAL_DMA_IrqDoneTrig](#) (HAL_DMA_x)
- uint8_t [HAL_DMA_IrqErr](#) (HAL_DMA_x)
中断处理函数
- uint8_t [HAL_DMA_IrqErrFlg](#) (HAL_DMA_x dma_x)
中断信号（同 [HAL_DMA_IrqErr](#)）
- void [HAL_DMA_IrqErrEnable](#) (HAL_DMA_x)
中断使能（本模块无独立使能位，由NVIC控制）
- void [HAL_DMA_IrqErrDisable](#) (HAL_DMA_x dma_x)
中断禁止（本模块无独立使能位，由NVIC控制）
- void [HAL_DMA_IrqErrClear](#) (HAL_DMA_x dma_x)
中断清除
- void [HAL_DMA_IrqErrTrig](#) (HAL_DMA_x)
- uint8_t [HAL_GPIO_Irq](#) (uint8_t)
中断处理函数
- uint8_t [HAL_GPIO_IrqFlg](#) (uint8_t line)
中断信号（同 [HAL_GPIO_Irq](#)）
- void [HAL_GPIO_IrqEnable](#) (uint8_t)
中断使能（本模块无独立使能位，由NVIC控制）
- void [HAL_GPIO_IrqDisable](#) (uint8_t line)

中断禁止（本模块无独立使能位，由NVIC控制）

- void HAL_GPIO_IrqClear (uint8_t line)

中断清除

- void HAL_GPIO_IrqTrig (uint8_t)
- uint8_t HAL_ADC_IrqIdleSamp (HAL_ADC_x)
- uint8_t HAL_ADC_IrqIdleSampFlg (HAL_ADC_x adc)
- void HAL_ADC_IrqIdleSampEnable (HAL_ADC_x)
- void HAL_ADC_IrqIdleSampDisable (HAL_ADC_x adc)
- void HAL_ADC_IrqIdleSampClear (HAL_ADC_x adc)
- uint8_t HAL_ADC_IrqWatchdog1 (HAL_ADC_x)
- uint8_t HAL_ADC_IrqWatchdog1Flg (HAL_ADC_x adc)
- void HAL_ADC_IrqWatchdog1Enable (HAL_ADC_x)
- void HAL_ADC_IrqWatchdog1Disable (HAL_ADC_x adc)
- void HAL_ADC_IrqWatchdog1Clear (HAL_ADC_x adc)
- uint8_t HAL_MCPWM_IrqTH40 (HAL_MCPWM_x)
- uint8_t HAL_MCPWM_IrqTH40Flg (HAL_MCPWM_x mcpwmx)
- void HAL_MCPWM_IrqTH40Enable (HAL_MCPWM_x)
- void HAL_MCPWM_IrqTH40Disable (HAL_MCPWM_x mcpwmx)
- void HAL_MCPWM_IrqTH40Clear (HAL_MCPWM_x mcpwmx)
- uint8_t HAL_MCPWM_IrqTH41 (HAL_MCPWM_x)
- uint8_t HAL_MCPWM_IrqTH41Flg (HAL_MCPWM_x mcpwmx)
- void HAL_MCPWM_IrqTH41Enable (HAL_MCPWM_x)
- void HAL_MCPWM_IrqTH41Disable (HAL_MCPWM_x mcpwmx)
- void HAL_MCPWM_IrqTH41Clear (HAL_MCPWM_x mcpwmx)
- uint8_t HAL_MCPWM_IrqTH50 (HAL_MCPWM_x)
- uint8_t HAL_MCPWM_IrqTH50Flg (HAL_MCPWM_x mcpwmx)
- void HAL_MCPWM_IrqTH50Enable (HAL_MCPWM_x)
- void HAL_MCPWM_IrqTH50Disable (HAL_MCPWM_x mcpwmx)
- void HAL_MCPWM_IrqTH50Clear (HAL_MCPWM_x mcpwmx)
- uint8_t HAL_MCPWM_IrqTH51 (HAL_MCPWM_x)
- uint8_t HAL_MCPWM_IrqTH51Flg (HAL_MCPWM_x mcpwmx)
- void HAL_MCPWM_IrqTH51Enable (HAL_MCPWM_x)
- void HAL_MCPWM_IrqTH51Disable (HAL_MCPWM_x mcpwmx)
- void HAL_MCPWM_IrqTH51Clear (HAL_MCPWM_x mcpwmx)
- uint8_t HAL_SW_Irq (uint8_t)
- uint8_t HAL_SW_IrqFlg (uint8_t x)
- void HAL_SW_IrqEnable (uint8_t)
- void HAL_SW_IrqDisable (uint8_t x)
- void HAL_SW_IrqClear (uint8_t x)
- void HAL_SW_IrqTrig (uint8_t)
- uint8_t HAL_UART_IrqAbd (HAL_UART_x)
- uint8_t HAL_UART_IrqAbdFlg (HAL_UART_x uart_x)
- void HAL_UART_IrqAbdEnable (HAL_UART_x)
- void HAL_UART_IrqAbdDisable (HAL_UART_x uart_x)
- void HAL_UART_IrqAbdClear (HAL_UART_x uart_x)
- uint8_t HAL_UART_IrqIdle (HAL_UART_x)
- uint8_t HAL_UART_IrqIdleFlg (HAL_UART_x uart_x)
- void HAL_UART_IrqIdleEnable (HAL_UART_x)
- void HAL_UART_IrqIdleDisable (HAL_UART_x uart_x)
- void HAL_UART_IrqIdleClear (HAL_UART_x uart_x)
- uint8_t HAL_UART_IrqLbd (HAL_UART_x)
- uint8_t HAL_UART_IrqLbdFlg (HAL_UART_x uart_x)
- void HAL_UART_IrqLbdEnable (HAL_UART_x)
- void HAL_UART_IrqLbdDisable (HAL_UART_x uart_x)

- void [HAL_UART_IrqLbdClear](#) (HAL_UART_x uart_x)
- uint8_t [HAL_UART_IrqRxOv](#) (HAL_UART_x)
- uint8_t [HAL_UART_IrqRxOvFlg](#) (HAL_UART_x uart_x)
- void [HAL_UART_IrqRxOvEnable](#) (HAL_UART_x)
- void [HAL_UART_IrqRxOvDisable](#) (HAL_UART_x uart_x)
- void [HAL_UART_IrqRxOvClear](#) (HAL_UART_x uart_x)
- uint8_t [HAL_UART_IrqTxOv](#) (HAL_UART_x)
- uint8_t [HAL_UART_IrqTxOvFlg](#) (HAL_UART_x uart_x)
- void [HAL_UART_IrqTxOvEnable](#) (HAL_UART_x)
- void [HAL_UART_IrqTxOvDisable](#) (HAL_UART_x uart_x)
- void [HAL_UART_IrqTxOvClear](#) (HAL_UART_x uart_x)

4.37.1 函数说明

4.37.1.1 HAL_ADC_IrqHardErr()

```
uint8_t HAL_ADC_IrqHardErr (
    HAL_ADC_x adc )
```

中断处理函数

参数

<i>adc</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1786 行定义.

4.37.1.2 HAL_ADC_IrqHardErrClear()

```
void HAL_ADC_IrqHardErrClear (
    HAL_ADC_x adc )
```

中断清除

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1847 行定义.

4.37.1.3 HAL_ADC_IrqHardErrDisable()

```
void HAL_ADC_IrqHardErrDisable (
    HAL_ADC_x adc )
```

中断禁止

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1836 行定义.

4.37.1.4 HAL_ADC_IrqHardErrEnable()

```
void HAL_ADC_IrqHardErrEnable (
    HAL_ADC_x adc )
```

中断使能

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1825 行定义.

4.37.1.5 HAL_ADC_IrqHardErrFlg()

```
uint8_t HAL_ADC_IrqHardErrFlg (
    HAL_ADC_x adc )
```

中断信号（和 HAL_ADC_IrqHardErr 的区别是不判断中断使能）

参数

<i>adc</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1807 行定义.

4.37.1.6 HAL_ADC_IrqIdleSamp()

```
uint8_t HAL_ADC_IrqIdleSamp (
    HAL_ADC_x adc )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3655 行定义.

4.37.1.7 HAL_ADC_IrqIdleSampClear()

```
void HAL_ADC_IrqIdleSampClear (
    HAL_ADC_x adc )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3673 行定义.

4.37.1.8 HAL_ADC_IrqIdleSampDisable()

```
void HAL_ADC_IrqIdleSampDisable (
    HAL_ADC_x adc )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3669 行定义.

4.37.1.9 HAL_ADC_IrqIdleSampEnable()

```
void HAL_ADC_IrqIdleSampEnable (
    HAL_ADC_x adc )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3665 行定义.

4.37.1.10 HAL_ADC_IrqIdleSampFlg()

```
uint8_t HAL_ADC_IrqIdleSampFlg (
    HAL_ADC_x adc )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3660 行定义.

4.37.1.11 HAL_ADC_IrqSamp1()

```
uint8_t HAL_ADC_IrqSamp1 (
    HAL_ADC_x adc )
```

中断处理函数

参数

<i>adc</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 160 行定义.

4.37.1.12 HAL_ADC_IrqSamp1Clear()

```
void HAL_ADC_IrqSamp1Clear (
    HAL_ADC_x adc )
```

中断清除

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 221 行定义.

4.37.1.13 HAL_ADC_IrqSamp1Disable()

```
void HAL_ADC_IrqSamp1Disable (
    HAL_ADC_x adc )
```

中断禁止

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 210 行定义.

4.37.1.14 HAL_ADC_IrqSamp1Enable()

```
void HAL_ADC_IrqSamp1Enable (
    HAL_ADC_x adc )
```

中断使能

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 199 行定义.

4.37.1.15 HAL_ADC_IrqSamp1Flg()

```
uint8_t HAL_ADC_IrqSamp1Flg (
    HAL_ADC_x adc )
```

中断信号（和 HAL_ADC_IrqSamp1 的区别是不判断中断使能）

参数

<i>adc</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 181 行定义.

4.37.1.16 HAL_ADC_IrqSamp2()

```
uint8_t HAL_ADC_IrqSamp2 (
    HAL_ADC_x adc )
```

中断处理函数

参数

<i>adc</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 233 行定义.

4.37.1.17 HAL_ADC_IrqSamp2Clear()

```
void HAL_ADC_IrqSamp2Clear (
    HAL_ADC_x adc )
```

中断清除

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 294 行定义.

4.37.1.18 HAL_ADC_IrqSamp2Disable()

```
void HAL_ADC_IrqSamp2Disable (
    HAL_ADC_x adc )
```

中断禁止

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 283 行定义.

4.37.1.19 HAL_ADC_IrqSamp2Enable()

```
void HAL_ADC_IrqSamp2Enable (  
    HAL_ADC_x adc )
```

中断使能

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 272 行定义.

4.37.1.20 HAL_ADC_IrqSamp2Flg()

```
uint8_t HAL_ADC_IrqSamp2Flg (  
    HAL_ADC_x adc )
```

中断信号（和 HAL_ADC_IrqSamp2 的区别是不判断中断使能）

参数

<i>adc</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 254 行定义.

4.37.1.21 HAL_ADC_IrqSamp3()

```
uint8_t HAL_ADC_IrqSamp3 (  
    HAL_ADC_x adc )
```

中断处理函数

参数

<i>adc</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1567 行定义.

4.37.1.22 HAL_ADC_IrqSamp3Clear()

```
void HAL_ADC_IrqSamp3Clear (
    HAL_ADC_x adc )
```

中断清除

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1628 行定义.

4.37.1.23 HAL_ADC_IrqSamp3Disable()

```
void HAL_ADC_IrqSamp3Disable (
    HAL_ADC_x adc )
```

中断禁止

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1617 行定义.

4.37.1.24 HAL_ADC_IrqSamp3Enable()

```
void HAL_ADC_IrqSamp3Enable (
    HAL_ADC_x adc )
```

中断使能

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1606 行定义.

4.37.1.25 HAL_ADC_IrqSamp3Flg()

```
uint8_t HAL_ADC_IrqSamp3Flg (  
    HAL_ADC_x adc )
```

中断信号（和 HAL_ADC_IrqSamp3 的区别是不判断中断使能）

参数

<i>adc</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1588 行定义.

4.37.1.26 HAL_ADC_IrqSamp4()

```
uint8_t HAL_ADC_IrqSamp4 (  
    HAL_ADC_x adc )
```

中断处理函数

参数

<i>adc</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1640 行定义.

4.37.1.27 HAL_ADC_IrqSamp4Clear()

```
void HAL_ADC_IrqSamp4Clear (  
    HAL_ADC_x adc )
```

中断清除

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1701 行定义.

4.37.1.28 HAL_ADC_IrqSamp4Disable()

```
void HAL_ADC_IrqSamp4Disable (
    HAL_ADC_x adc )
```

中断禁止

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1690 行定义.

4.37.1.29 HAL_ADC_IrqSamp4Enable()

```
void HAL_ADC_IrqSamp4Enable (
    HAL_ADC_x adc )
```

中断使能

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1679 行定义.

4.37.1.30 HAL_ADC_IrqSamp4Flg()

```
uint8_t HAL_ADC_IrqSamp4Flg (
    HAL_ADC_x adc )
```

中断信号（和 HAL_ADC_IrqSamp4 的区别是不判断中断使能）

参数

<i>adc</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1661 行定义.

4.37.1.31 HAL_ADC_IrqSoftErr()

```
uint8_t HAL_ADC_IrqSoftErr (
    HAL_ADC_x adc )
```

中断处理函数

参数

<i>adc</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1859 行定义.

4.37.1.32 HAL_ADC_IrqSoftErrClear()

```
void HAL_ADC_IrqSoftErrClear (
    HAL_ADC_x adc )
```

中断清除

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1920 行定义.

4.37.1.33 HAL_ADC_IrqSoftErrDisable()

```
void HAL_ADC_IrqSoftErrDisable (
    HAL_ADC_x adc )
```

中断禁止

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1909 行定义.

4.37.1.34 HAL_ADC_IrqSoftErrEnable()

```
void HAL_ADC_IrqSoftErrEnable (
    HAL_ADC_x adc )
```

中断使能

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1898 行定义.

4.37.1.35 HAL_ADC_IrqSoftErrFlg()

```
uint8_t HAL_ADC_IrqSoftErrFlg (  
    HAL_ADC_x adc )
```

中断信号（和 HAL_ADC_IrqSoftErr 的区别是不判断中断使能）

参数

<i>adc</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1880 行定义.

4.37.1.36 HAL_ADC_IrqWatchdog()

```
uint8_t HAL_ADC_IrqWatchdog (  
    HAL_ADC_x adc )
```

中断处理函数

参数

<i>adc</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1713 行定义.

4.37.1.37 HAL_ADC_IrqWatchdog1()

```
uint8_t HAL_ADC_IrqWatchdog1 (  
    HAL_ADC_x adc )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3678 行定义.

4.37.1.38 HAL_ADC_IrqWatchdog1Clear()

```
void HAL_ADC_IrqWatchdog1Clear (  
    HAL_ADC_x adc )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3696 行定义.

4.37.1.39 HAL_ADC_IrqWatchdog1Disable()

```
void HAL_ADC_IrqWatchdog1Disable (
    HAL_ADC_x adc )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3692 行定义.

4.37.1.40 HAL_ADC_IrqWatchdog1Enable()

```
void HAL_ADC_IrqWatchdog1Enable (
    HAL_ADC_x adc )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3688 行定义.

4.37.1.41 HAL_ADC_IrqWatchdog1Flg()

```
uint8_t HAL_ADC_IrqWatchdog1Flg (
    HAL_ADC_x adc )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3683 行定义.

4.37.1.42 HAL_ADC_IrqWatchdogClear()

```
void HAL_ADC_IrqWatchdogClear (
    HAL_ADC_x adc )
```

中断清除

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1774 行定义.

4.37.1.43 HAL_ADC_IrqWatchdogDisable()

```
void HAL_ADC_IrqWatchdogDisable (
    HAL_ADC_x adc )
```

中断禁止

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1763 行定义.

4.37.1.44 HAL_ADC_IrqWatchdogEnable()

```
void HAL_ADC_IrqWatchdogEnable (
    HAL_ADC_x adc )
```

中断使能

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1752 行定义.

4.37.1.45 HAL_ADC_IrqWatchdogFlg()

```
uint8_t HAL_ADC_IrqWatchdogFlg (
    HAL_ADC_x adc )
```

中断信号（和 HAL_ADC_IrqWatchdog 的区别是不判断中断使能）

参数

<i>adc</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1734 行定义.

4.37.1.46 HAL_CAN_IrqAll()

```
uint8_t HAL_CAN_IrqAll (
    HAL_CAN_x can_x )
```

中断处理函数

参数

<i>can_x</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1421 行定义.

4.37.1.47 HAL_CAN_IrqAllClear()

```
void HAL_CAN_IrqAllClear (
    HAL_CAN_x can_x )
```

中断清除

参数

can_x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1482 行定义.

4.37.1.48 HAL_CAN_IrqAllDisable()

```
void HAL_CAN_IrqAllDisable (
    HAL_CAN_x can_x )
```

中断禁止

参数

can_x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1471 行定义.

4.37.1.49 HAL_CAN_IrqAllEnable()

```
void HAL_CAN_IrqAllEnable (
    HAL_CAN_x can_x )
```

中断使能

参数

can_x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1460 行定义.

4.37.1.50 HAL_CAN_IrqAllFlg()

```
uint8_t HAL_CAN_IrqAllFlg (
    HAL_CAN_x can_x )
```

中断信号（和 HAL_CAN_IrqAll 的区别是不判断中断使能）

参数

can_x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1442 行定义.

4.37.1.51 HAL_CAN_IrqRxDone()

```
uint8_t HAL_CAN_IrqRxDone (
    HAL_CAN_x can_x )
```

中断处理函数

参数

can_x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1275 行定义.

4.37.1.52 HAL_CAN_IrqRxDoneClear()

```
void HAL_CAN_IrqRxDoneClear (
    HAL_CAN_x can_x )
```

中断清除

参数

can_x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1336 行定义.

4.37.1.53 HAL_CAN_IrqRxDoneDisable()

```
void HAL_CAN_IrqRxDoneDisable (
    HAL_CAN_x can_x )
```

中断禁止

参数

<i>can</i> _{<i>x</i>}	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1325 行定义.

4.37.1.54 HAL_CAN_IrqRxDoneEnable()

```
void HAL_CAN_IrqRxDoneEnable (
    HAL_CAN_x can_x )
```

中断使能

参数

<i>can</i> _{<i>x</i>}	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1314 行定义.

4.37.1.55 HAL_CAN_IrqRxDoneFlg()

```
uint8_t HAL_CAN_IrqRxDoneFlg (
    HAL_CAN_x can_x )
```

中断信号（和 HAL_CAN_IrqRxDone 的区别是不判断中断使能）

参数

<i>can</i> _{<i>x</i>}	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1296 行定义.

4.37.1.56 HAL_CAN_IrqTxDone()

```
uint8_t HAL_CAN_IrqTxDone (
    HAL_CAN_x can_x )
```

中断处理函数

参数

can_x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1348 行定义.

4.37.1.57 HAL_CAN_IrqTxDoneClear()

```
void HAL_CAN_IrqTxDoneClear (
    HAL_CAN_x can_x )
```

中断清除

参数

can_x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1409 行定义.

4.37.1.58 HAL_CAN_IrqTxDoneDisable()

```
void HAL_CAN_IrqTxDoneDisable (
    HAL_CAN_x can_x )
```

中断禁止

参数

can_x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1398 行定义.

4.37.1.59 HAL_CAN_IrqTxDoneEnable()

```
void HAL_CAN_IrqTxDoneEnable (
    HAL_CAN_x can_x )
```

中断使能

参数

<i>can</i> ↔ _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1387 行定义.

4.37.1.60 HAL_CAN_IrqTxDoneFlg()

```
uint8_t HAL_CAN_IrqTxDoneFlg (
    HAL_CAN_x can_x )
```

中断信号（和 HAL_CAN_IrqTxDone 的区别是不判断中断使能）

参数

<i>can</i> ↔ _x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1369 行定义.

4.37.1.61 HAL_CMP_IrqCompare()

```
uint8_t HAL_CMP_IrqCompare (
    HAL_CMP_x cmp_x )
```

中断处理函数

参数

<i>cmp</i> ↔ _x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 691 行定义.

4.37.1.62 HAL_CMP_IrqCompareClear()

```
void HAL_CMP_IrqCompareClear (
    HAL_CMP_x cmp_x )
```

中断清除

参数

<i>cmp</i> ↔ _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 752 行定义.

4.37.1.63 HAL_CMP_IrqCompareDisable()

```
void HAL_CMP_IrqCompareDisable (
    HAL_CMP_x cmp_x )
```

中断禁止

参数

<i>cmp</i> ↔ _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 741 行定义.

4.37.1.64 HAL_CMP_IrqCompareEnable()

```
void HAL_CMP_IrqCompareEnable (
    HAL_CMP_x cmp_x )
```

中断使能

参数

<i>cmp</i> ↔ _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 730 行定义.

4.37.1.65 HAL_CMP_IrqCompareFlg()

```
uint8_t HAL_CMP_IrqCompareFlg (
    HAL_CMP_x cmp_x )
```

中断信号（和 HAL_CMP_IrqCompare 的区别是不判断中断使能）

参数

<i>cmp</i> ↔ _x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 712 行定义.

4.37.1.66 HAL_DMA_IrqDone()

```
uint8_t HAL_DMA_IrqDone (
    HAL_DMA_x dma_x )
```

中断处理函数

参数

<i>dma</i> ↔ _x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3448 行定义.

4.37.1.67 HAL_DMA_IrqDoneClear()

```
void HAL_DMA_IrqDoneClear (
    HAL_DMA_x dma_x )
```

中断清除

参数

<i>dma</i> ↔ _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3505 行定义.

4.37.1.68 HAL_DMA_IrqDoneDisable()

```
void HAL_DMA_IrqDoneDisable (
    HAL_DMA_x dma_x )
```

中断禁止（本模块无独立使能位，由NVIC控制）

参数

<i>dma</i> ↔ _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3496 行定义.

4.37.1.69 HAL_DMA_IrqDoneEnable()

```
void HAL_DMA_IrqDoneEnable (
    HAL_DMA_x dma_x )
```

中断使能（本模块无独立使能位，由NVIC控制）

参数

<i>dma</i> ↔ _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3487 行定义.

4.37.1.70 HAL_DMA_IrqDoneFlg()

```
uint8_t HAL_DMA_IrqDoneFlg (
    HAL_DMA_x dma_x )
```

中断信号（同 HAL_DMA_IrqDone）

参数

<i>dma</i> ↔ _x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3468 行定义.

4.37.1.71 HAL_DMA_IrqDoneTrig()

```
void HAL_DMA_IrqDoneTrig (
    HAL_DMA_x )
```

4.37.1.72 HAL_DMA_IrqErr()

```
uint8_t HAL_DMA_IrqErr (
    HAL_DMA_x dma_x )
```

中断处理函数

参数

dma_{x}	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3518 行定义.

4.37.1.73 HAL_DMA_IrqErrClear()

```
void HAL_DMA_IrqErrClear (
    HAL_DMA_x dma_x )
```

中断清除

参数

dma_{x}	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3575 行定义.

4.37.1.74 HAL_DMA_IrqErrDisable()

```
void HAL_DMA_IrqErrDisable (
    HAL_DMA_x dma_x )
```

中断禁止（本模块无独立使能位，由NVIC控制）

参数

dma_{x}	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3566 行定义.

4.37.1.75 HAL_DMA_IrqErrEnable()

```
void HAL_DMA_IrqErrEnable (
    HAL_DMA_x dma_x )
```

中断使能（本模块无独立使能位，由NVIC控制）

参数

<i>dma</i> ↔ <i>_x</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3557 行定义.

4.37.1.76 HAL_DMA_IrqErrFlg()

```
uint8_t HAL_DMA_IrqErrFlg (
    HAL_DMA_x dma_x )
```

中断信号 (同 HAL_DMA_IrqErr)

参数

<i>dma</i> ↔ <i>_x</i>	模块编号

返回

1表示中断发生, 0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3538 行定义.

4.37.1.77 HAL_DMA_IrqErrTrig()

```
void HAL_DMA_IrqErrTrig (
    HAL_DMA_x )
```

4.37.1.78 HAL_GPIO_Irq()

```
uint8_t HAL_GPIO_Irq (
    uint8_t line )
```

中断处理函数

参数

<i>line</i>	模块编号

返回

1表示中断发生, 0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3588 行定义.

4.37.1.79 HAL_GPIO_IrqClear()

```
void HAL_GPIO_IrqClear (
    uint8_t line )
```

中断清除

参数

<i>line</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3645 行定义.

4.37.1.80 HAL_GPIO_IrqDisable()

```
void HAL_GPIO_IrqDisable (
    uint8_t line )
```

中断禁止（本模块无独立使能位，由NVIC控制）

参数

<i>line</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3636 行定义.

4.37.1.81 HAL_GPIO_IrqEnable()

```
void HAL_GPIO_IrqEnable (
    uint8_t line )
```

中断使能（本模块无独立使能位，由NVIC控制）

参数

<i>line</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3627 行定义.

4.37.1.82 HAL_GPIO_IrqFlg()

```
uint8_t HAL_GPIO_IrqFlg (
    uint8_t line )
```

中断信号（同 HAL_GPIO_Irq）

参数

<i>line</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3608 行定义.

4.37.1.83 HAL_GPIO_IrqTrig()

```
void HAL_GPIO_IrqTrig (
    uint8_t )
```

4.37.1.84 HAL_HALL_IrqChange()

```
uint8_t HAL_HALL_IrqChange (
    HAL_HALL_x hall )
```

中断处理函数

参数

<i>hall</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 14 行定义.

4.37.1.85 HAL_HALL_IrqChangeClear()

```
void HAL_HALL_IrqChangeClear (
    HAL_HALL_x hall )
```

中断清除

参数

<i>hall</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 75 行定义.

4.37.1.86 HAL_HALL_IrqChangeDisable()

```
void HAL_HALL_IrqChangeDisable (
    HAL_HALL_x hall )
```

中断禁止

参数

<i>hall</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 64 行定义.

4.37.1.87 HAL_HALL_IrqChangeEnable()

```
void HAL_HALL_IrqChangeEnable (
    HAL_HALL_x hall )
```

中断使能

参数

<i>hall</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 53 行定义.

4.37.1.88 HAL_HALL_IrqChangeFlg()

```
uint8_t HAL_HALL_IrqChangeFlg (
    HAL_HALL_x hall )
```

中断信号（和 HAL_HALL_IrqChange 的区别是不判断中断使能）

参数

<i>hall</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 35 行定义.

4.37.1.89 HAL_HALL_IrqOver()

```
uint8_t HAL_HALL_IrqOver (
    HAL_HALL_x hall )
```

中断处理函数

参数

<i>hall</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 87 行定义.

4.37.1.90 HAL_HALL_IrqOverClear()

```
void HAL_HALL_IrqOverClear (
    HAL_HALL_x hall )
```

中断清除

参数

<i>hall</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 148 行定义.

4.37.1.91 HAL_HALL_IrqOverDisable()

```
void HAL_HALL_IrqOverDisable (
    HAL_HALL_x hall )
```

中断禁止

参数

<i>hall</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 137 行定义.

4.37.1.92 HAL_HALL_IrqOverEnable()

```
void HAL_HALL_IrqOverEnable (
    HAL_HALL_x hall )
```

中断使能

参数

<i>hall</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 126 行定义.

4.37.1.93 HAL_HALL_IrqOverFlg()

```
uint8_t HAL_HALL_IrqOverFlg (
    HAL_HALL_x hall )
```

中断信号（和 HAL_HALL_IrqOver 的区别是不判断中断使能）

参数

<i>hall</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 108 行定义.

4.37.1.94 HAL_I2C_IrqBusErr()

```
uint8_t HAL_I2C_IrqBusErr (
    HAL_I2C_x i2c_x )
```

中断处理函数

参数

<i>i2c_x</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3083 行定义.

4.37.1.95 HAL_I2C_IrqBusErrClear()

```
void HAL_I2C_IrqBusErrClear (
    HAL_I2C_x i2c_x )
```

中断清除

参数

$i2c\_x$	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3144 行定义.

4.37.1.96 HAL_I2C_IrqBusErrDisable()

```
void HAL_I2C_IrqBusErrDisable (  
    HAL_I2C_x i2c_x )
```

中断禁止

参数

$i2c\_x$	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3133 行定义.

4.37.1.97 HAL_I2C_IrqBusErrEnable()

```
void HAL_I2C_IrqBusErrEnable (  
    HAL_I2C_x i2c_x )
```

中断使能

参数

$i2c\_x$	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3122 行定义.

4.37.1.98 HAL_I2C_IrqBusErrFlg()

```
uint8_t HAL_I2C_IrqBusErrFlg (  
    HAL_I2C_x i2c_x )
```

中断信号（和 HAL_I2C_IrqBusErr 的区别是不判断中断使能）

参数

$i2c\_x$	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3104 行定义.

4.37.1.99 HAL_I2C_IrqDone()

```
uint8_t HAL_I2C_IrqDone (
    HAL_I2C_x i2c_x )
```

中断处理函数

参数

$i2c\_x$	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3010 行定义.

4.37.1.100 HAL_I2C_IrqDoneClear()

```
void HAL_I2C_IrqDoneClear (
    HAL_I2C_x i2c_x )
```

中断清除

参数

$i2c\_x$	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3071 行定义.

4.37.1.101 HAL_I2C_IrqDoneDisable()

```
void HAL_I2C_IrqDoneDisable (
    HAL_I2C_x i2c_x )
```

中断禁止

参数

$i2c\_x$	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3060 行定义.

4.37.1.102 HAL_I2C_IrqDoneEnable()

```
void HAL_I2C_IrqDoneEnable (
    HAL_I2C_x i2c_x )
```

中断使能

参数

$i2c\_x$	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3049 行定义.

4.37.1.103 HAL_I2C_IrqDoneFlg()

```
uint8_t HAL_I2C_IrqDoneFlg (
    HAL_I2C_x i2c_x )
```

中断信号（和 HAL_I2C_IrqDone 的区别是不判断中断使能）

参数

$i2c\_x$	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3031 行定义.

4.37.1.104 HAL_I2C_IrqStop()

```
uint8_t HAL_I2C_IrqStop (
    HAL_I2C_x i2c_x )
```

中断处理函数

参数

$i2c\_x$	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3156 行定义.

4.37.1.105 HAL_I2C_IrqStopClear()

```
void HAL_I2C_IrqStopClear (
    HAL_I2C_x i2c_x )
```

中断清除

参数

$i2c\_x$	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3217 行定义.

4.37.1.106 HAL_I2C_IrqStopDisable()

```
void HAL_I2C_IrqStopDisable (
    HAL_I2C_x i2c_x )
```

中断禁止

参数

$i2c\_x$	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3206 行定义.

4.37.1.107 HAL_I2C_IrqStopEnable()

```
void HAL_I2C_IrqStopEnable (
    HAL_I2C_x i2c_x )
```

中断使能

参数

<i>i2c</i> ↔ _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3195 行定义.

4.37.1.108 HAL_I2C_IrqStopFlg()

```
uint8_t HAL_I2C_IrqStopFlg (  
    HAL_I2C_x i2c_x )
```

中断信号（和 HAL_I2C_IrqStop 的区别是不判断中断使能）

参数

<i>i2c</i> ↔ _x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3177 行定义.

4.37.1.109 HAL_MCPWM_IrqFail()

```
uint8_t HAL_MCPWM_IrqFail (  
    HAL_MCPWM_x mcpwm_x )
```

中断处理函数

参数

<i>mcpwm_x</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 614 行定义.

4.37.1.110 HAL_MCPWM_IrqFail0()

```
uint8_t HAL_MCPWM_IrqFail0 (  
    HAL_MCPWM_x mcpwm_x )
```

中断处理函数

参数

<i>mcpwm</i> x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 460 行定义.

4.37.1.111 HAL_MCPWM_IrqFail0Clear()

```
void HAL_MCPWM_IrqFail0Clear (
    HAL_MCPWM_x mcpwm )
```

中断清除

参数

<i>mcpwm</i> x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 525 行定义.

4.37.1.112 HAL_MCPWM_IrqFail0Disable()

```
void HAL_MCPWM_IrqFail0Disable (
    HAL_MCPWM_x mcpwm )
```

中断禁止

参数

<i>mcpwm</i> x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 512 行定义.

4.37.1.113 HAL_MCPWM_IrqFail0Enable()

```
void HAL_MCPWM_IrqFail0Enable (
    HAL_MCPWM_x mcpwm )
```

中断使能

参数

<i>mcpwm</i> x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 499 行定义.

4.37.1.114 HAL_MCPWM_IrqFail0Flg()

```
uint8_t HAL_MCPWM_IrqFail0Flg (  
    HAL_MCPWM_X mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqFail0 的区别是不判断中断使能）

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 481 行定义.

4.37.1.115 HAL_MCPWM_IrqFail1()

```
uint8_t HAL_MCPWM_IrqFail1 (  
    HAL_MCPWM_X mcpwmx )
```

中断处理函数

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 537 行定义.

4.37.1.116 HAL_MCPWM_IrqFail1Clear()

```
void HAL_MCPWM_IrqFail1Clear (  
    HAL_MCPWM_X mcpwmx )
```

中断清除

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 602 行定义.

4.37.1.117 HAL_MCPWM_IrqFail1Disable()

```
void HAL_MCPWM_IrqFail1Disable (
    HAL_MCPWM_x mcpwmx )
```

中断禁止

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 589 行定义.

4.37.1.118 HAL_MCPWM_IrqFail1Enable()

```
void HAL_MCPWM_IrqFail1Enable (
    HAL_MCPWM_x mcpwmx )
```

中断使能

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 576 行定义.

4.37.1.119 HAL_MCPWM_IrqFail1Flg()

```
uint8_t HAL_MCPWM_IrqFail1Flg (
    HAL_MCPWM_x mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqFail1 的区别是不判断中断使能）

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 558 行定义.

4.37.1.120 HAL_MCPWM_IrqFailClear()

```
void HAL_MCPWM_IrqFailClear (
    HAL_MCPWM_TypeDef * mcpwm )
```

中断清除

参数

<i>mcpwm</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 679 行定义.

4.37.1.121 HAL_MCPWM_IrqFailDisable()

```
void HAL_MCPWM_IrqFailDisable (
    HAL_MCPWM_TypeDef * mcpwm )
```

中断禁止

参数

<i>mcpwm</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 666 行定义.

4.37.1.122 HAL_MCPWM_IrqFailEnable()

```
void HAL_MCPWM_IrqFailEnable (
    HAL_MCPWM_TypeDef * mcpwm )
```

中断使能

参数

<i>mcpwm</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 653 行定义.

4.37.1.123 HAL_MCPWM_IrqFailFlg()

```
uint8_t HAL_MCPWM_IrqFailFlg (
    HAL_MCPWM_TypeDef * mcpwm )
```

中断信号（和 HAL_MCPWM_IrqFail 的区别是不判断中断使能）

参数

<i>mcpwm</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 635 行定义.

4.37.1.124 HAL_MCPWM_IrqOverflow()

```
uint8_t HALMCPWM_IrqOverflow (
    HAL_MCPWM_x mcpwm )
```

中断处理函数

参数

<i>mcpwm</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 306 行定义.

4.37.1.125 HAL_MCPWM_IrqOverflowClear()

```
void HALMCPWM_IrqOverflowClear (
    HAL_MCPWM_x mcpwm )
```

中断清除

参数

<i>mcpwm</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 371 行定义.

4.37.1.126 HAL_MCPWM_IrqOverflowDisable()

```
void HALMCPWM_IrqOverflowDisable (
    HAL_MCPWM_x mcpwm )
```

中断禁止

参数

<i>mcpwm</i> x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 358 行定义.

4.37.1.127 HAL_MCPWM_IrqOverflowEnable()

```
void HAL_MCPWM_IrqOverflowEnable (
    HAL_MCPWM_x mcpwm )
```

中断使能

参数

<i>mcpwm</i> x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 345 行定义.

4.37.1.128 HAL_MCPWM_IrqOverflowFlg()

```
uint8_t HAL_MCPWM_IrqOverflowFlg (
    HAL_MCPWM_x mcpwm )
```

中断信号（和 HAL_MCPWM_IrqOverflow 的区别是不判断中断使能）

参数

<i>mcpwm</i> x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 327 行定义.

4.37.1.129 HAL_MCPWM_IrqShort()

```
uint8_t HAL_MCPWM_IrqShort (
    HAL_MCPWM_x mcpwm )
```

中断处理函数

参数

<i>mcpwm</i> x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2933 行定义.

4.37.1.130 HAL_MCPWM_IrqShortClear()

```
void HAL_MCPWM_IrqShortClear (
    HAL_MCPWM_X mcpwmx )
```

中断清除

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2998 行定义.

4.37.1.131 HAL_MCPWM_IrqShortDisable()

```
void HAL_MCPWM_IrqShortDisable (
    HAL_MCPWM_X mcpwmx )
```

中断禁止

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2985 行定义.

4.37.1.132 HAL_MCPWM_IrqShortEnable()

```
void HAL_MCPWM_IrqShortEnable (
    HAL_MCPWM_X mcpwmx )
```

中断使能

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2972 行定义.

4.37.1.133 HAL_MCPWM_IrqShortFlg()

```
uint8_t HAL_MCPWM_IrqShortFlg (  
    HAL_MCPWM_x mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqShort 的区别是不判断中断使能）

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2954 行定义.

4.37.1.134 HAL_MCPWM_IrqTH00()

```
uint8_t HAL_MCPWM_IrqTH00 (  
    HAL_MCPWM_x mcpwmx )
```

中断处理函数

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2317 行定义.

4.37.1.135 HAL_MCPWM_IrqTH00Clear()

```
void HAL_MCPWM_IrqTH00Clear (  
    HAL_MCPWM_x mcpwmx )
```

中断清除

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2382 行定义.

4.37.1.136 HAL_MCPWM_IrqTH00Disable()

```
void HAL_MCPWM_IrqTH00Disable (
    HAL_MCPWM_x mcpwmx )
```

中断禁止

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2369 行定义.

4.37.1.137 HAL_MCPWM_IrqTH00Enable()

```
void HAL_MCPWM_IrqTH00Enable (
    HAL_MCPWM_x mcpwmx )
```

中断使能

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2356 行定义.

4.37.1.138 HAL_MCPWM_IrqTH00Flg()

```
uint8_t HAL_MCPWM_IrqTH00Flg (
    HAL_MCPWM_x mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqTH00 的区别是不判断中断使能）

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2338 行定义.

4.37.1.139 HAL_MCPWM_IrqTH01()

```
uint8_t HAL_MCPWM_IrqTH01 (
    HAL_MCPWM_x mcpwmx )
```

中断处理函数

参数

<i>mcpwm</i> x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2394 行定义.

4.37.1.140 HAL_MCPWM_IrqTH01Clear()

```
void HAL_MCPWM_IrqTH01Clear (
    HAL_MCPWM_x mcpwm )
```

中断清除

参数

<i>mcpwm</i> x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2459 行定义.

4.37.1.141 HAL_MCPWM_IrqTH01Disable()

```
void HAL_MCPWM_IrqTH01Disable (
    HAL_MCPWM_x mcpwm )
```

中断禁止

参数

<i>mcpwm</i> x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2446 行定义.

4.37.1.142 HAL_MCPWM_IrqTH01Enable()

```
void HAL_MCPWM_IrqTH01Enable (
    HAL_MCPWM_x mcpwm )
```

中断使能

参数

<i>mcpwm</i> x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2433 行定义.

4.37.1.143 HAL_MCPWM_IrqTH01Flg()

```
uint8_t HAL_MCPWM_IrqTH01Flg (
    HAL_MCPWM_x mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqTH01 的区别是不判断中断使能）

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2415 行定义.

4.37.1.144 HAL_MCPWM_IrqTH10()

```
uint8_t HAL_MCPWM_IrqTH10 (
    HAL_MCPWM_x mcpwmx )
```

中断处理函数

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2471 行定义.

4.37.1.145 HAL_MCPWM_IrqTH10Clear()

```
void HAL_MCPWM_IrqTH10Clear (
    HAL_MCPWM_x mcpwmx )
```

中断清除

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2536 行定义.

4.37.1.146 HAL_MCPWM_IrqTH10Disable()

```
void HAL_MCPWM_IrqTH10Disable (
    HAL_MCPWM_x mcpwmx )
```

中断禁止

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2523 行定义.

4.37.1.147 HAL_MCPWM_IrqTH10Enable()

```
void HAL_MCPWM_IrqTH10Enable (
    HAL_MCPWM_x mcpwmx )
```

中断使能

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2510 行定义.

4.37.1.148 HAL_MCPWM_IrqTH10Flg()

```
uint8_t HAL_MCPWM_IrqTH10Flg (
    HAL_MCPWM_x mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqTH10 的区别是不判断中断使能）

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2492 行定义.

4.37.1.149 HAL_MCPWM_IrqTH11()

```
uint8_t HAL_MCPWM_IrqTH11 (
    HAL_MCPWM_x mcpwmx )
```

中断处理函数

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2548 行定义.

4.37.1.150 HAL_MCPWM_IrqTH11Clear()

```
void HAL_MCPWM_IrqTH11Clear (
    HAL_MCPWM_x mcpwmx )
```

中断清除

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2613 行定义.

4.37.1.151 HAL_MCPWM_IrqTH11Disable()

```
void HAL_MCPWM_IrqTH11Disable (
    HAL_MCPWM_x mcpwmx )
```

中断禁止

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2600 行定义.

4.37.1.152 HAL_MCPWM_IrqTH11Enable()

```
void HAL_MCPWM_IrqTH11Enable (
    HAL_MCPWM_x mcpwmx )
```

中断使能

参数

<i>mcpwm</i> x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2587 行定义.

4.37.1.153 HAL_MCPWM_IrqTH11Flg()

```
uint8_t HAL_MCPWM_IrqTH11Flg (
    HAL_MCPWM_x mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqTH11 的区别是不判断中断使能）

参数

<i>mcpwm</i> x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2569 行定义.

4.37.1.154 HAL_MCPWM_IrqTH20()

```
uint8_t HAL_MCPWM_IrqTH20 (
    HAL_MCPWM_x mcpwmx )
```

中断处理函数

参数

<i>mcpwm</i> x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2625 行定义.

4.37.1.155 HAL_MCPWM_IrqTH20Clear()

```
void HAL_MCPWM_IrqTH20Clear (
    HAL_MCPWM_x mcpwmx )
```

中断清除

参数

<i>mcpwm</i> x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2690 行定义.

4.37.1.156 HAL_MCPWM_IrqTH20Disable()

```
void HAL_MCPWM_IrqTH20Disable (
    HAL_MCPWM_x mcpwmx )
```

中断禁止

参数

<i>mcpwm</i> x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2677 行定义.

4.37.1.157 HAL_MCPWM_IrqTH20Enable()

```
void HAL_MCPWM_IrqTH20Enable (
    HAL_MCPWM_x mcpwmx )
```

中断使能

参数

<i>mcpwm</i> x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2664 行定义.

4.37.1.158 HAL_MCPWM_IrqTH20Flg()

```
uint8_t HAL_MCPWM_IrqTH20Flg (
    HAL_MCPWM_x mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqTH20 的区别是不判断中断使能）

参数

<i>mcpwm</i> x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2646 行定义.

4.37.1.159 HAL_MCPWM_IrqTH21()

```
uint8_t HAL_MCPWM_IrqTH21 (
    HAL_MCPWM_x mcpwmx )
```

中断处理函数

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2702 行定义.

4.37.1.160 HAL_MCPWM_IrqTH21Clear()

```
void HAL_MCPWM_IrqTH21Clear (
    HAL_MCPWM_x mcpwmx )
```

中断清除

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2767 行定义.

4.37.1.161 HAL_MCPWM_IrqTH21Disable()

```
void HAL_MCPWM_IrqTH21Disable (
    HAL_MCPWM_x mcpwmx )
```

中断禁止

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2754 行定义.

4.37.1.162 HAL_MCPWM_IrqTH21Enable()

```
void HAL_MCPWM_IrqTH21Enable (
    HAL_MCPWM_x mcpwmx )
```

中断使能

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2741 行定义.

4.37.1.163 HAL_MCPWM_IrqTH21Flg()

```
uint8_t HAL_MCPWM_IrqTH21Flg (
    HAL_MCPWM_x mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqTH21 的区别是不判断中断使能）

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2723 行定义.

4.37.1.164 HAL_MCPWM_IrqTH30()

```
uint8_t HAL_MCPWM_IrqTH30 (
    HAL_MCPWM_x mcpwmx )
```

中断处理函数

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2779 行定义.

4.37.1.165 HAL_MCPWM_IrqTH30Clear()

```
void HAL_MCPWM_IrqTH30Clear (
    HAL_MCPWM_x mcpwmx )
```

中断清除

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2844 行定义.

4.37.1.166 HAL_MCPWM_IrqTH30Disable()

```
void HAL_MCPWM_IrqTH30Disable (
    HAL_MCPWM_x mcpwmx )
```

中断禁止

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2831 行定义.

4.37.1.167 HAL_MCPWM_IrqTH30Enable()

```
void HAL_MCPWM_IrqTH30Enable (
    HAL_MCPWM_x mcpwmx )
```

中断使能

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2818 行定义.

4.37.1.168 HAL_MCPWM_IrqTH30Flg()

```
uint8_t HAL_MCPWM_IrqTH30Flg (
    HAL_MCPWM_x mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqTH30 的区别是不判断中断使能）

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2800 行定义.

4.37.1.169 HAL_MCPWM_IrqTH31()

```
uint8_t HAL_MCPWM_IrqTH31 (
    HAL_MCPWM_x mcpwmx )
```

中断处理函数

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2856 行定义.

4.37.1.170 HAL_MCPWM_IrqTH31Clear()

```
void HAL_MCPWM_IrqTH31Clear (
    HAL_MCPWM_x mcpwmx )
```

中断清除

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2921 行定义.

4.37.1.171 HAL_MCPWM_IrqTH31Disable()

```
void HAL_MCPWM_IrqTH31Disable (
    HAL_MCPWM_x mcpwmx )
```

中断禁止

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2908 行定义.

4.37.1.172 HAL_MCPWM_IrqTH31Enable()

```
void HAL_MCPWM_IrqTH31Enable (
    HAL_MCPWM_x mcpwmx )
```

中断使能

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2895 行定义.

4.37.1.173 HAL_MCPWM_IrqTH31Flg()

```
uint8_t HAL_MCPWM_IrqTH31Flg (
    HAL_MCPWM_x mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqTH31 的区别是不判断中断使能）

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2877 行定义.

4.37.1.174 HAL_MCPWM_IrqTH40()

```
uint8_t HAL_MCPWM_IrqTH40 (
    HAL_MCPWM_x mcpwmx )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3701 行定义.

4.37.1.175 HAL_MCPWM_IrqTH40Clear()

```
void HAL_MCPWM_IrqTH40Clear (
    HAL_MCPWM_x mcpwmx )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3719 行定义.

4.37.1.176 HAL_MCPWM_IrqTH40Disable()

```
void HAL_MCPWM_IrqTH40Disable (
    HAL_MCPWM_TypeDef * mcpwm )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3715 行定义.

4.37.1.177 HAL_MCPWM_IrqTH40Enable()

```
void HAL_MCPWM_IrqTH40Enable (
    HAL_MCPWM_TypeDef * mcpwm )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3711 行定义.

4.37.1.178 HAL_MCPWM_IrqTH40Flg()

```
uint8_t HAL_MCPWM_IrqTH40Flg (
    HAL_MCPWM_TypeDef * mcpwm )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3706 行定义.

4.37.1.179 HAL_MCPWM_IrqTH41()

```
uint8_t HAL_MCPWM_IrqTH41 (
    HAL_MCPWM_TypeDef * mcpwm )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3724 行定义.

4.37.1.180 HAL_MCPWM_IrqTH41Clear()

```
void HAL_MCPWM_IrqTH41Clear (
    HAL_MCPWM_TypeDef * mcpwm )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3742 行定义.

4.37.1.181 HAL_MCPWM_IrqTH41Disable()

```
void HAL_MCPWM_IrqTH41Disable (
    HAL_MCPWM_TypeDef * mcpwm )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3738 行定义.

4.37.1.182 HAL_MCPWM_IrqTH41Enable()

```
void HAL_MCPWM_IrqTH41Enable (
    HAL_MCPWM_TypeDef * mcpwm )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3734 行定义.

4.37.1.183 HAL_MCPWM_IrqTH41Flg()

```
uint8_t HAL_MCPWM_IrqTH41Flg (
    HAL_MCPWM_x mcpwmx )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3729 行定义.

4.37.1.184 HAL_MCPWM_IrqTH50()

```
uint8_t HAL_MCPWM_IrqTH50 (
    HAL_MCPWM_x mcpwmx )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3747 行定义.

4.37.1.185 HAL_MCPWM_IrqTH50Clear()

```
void HAL_MCPWM_IrqTH50Clear (
    HAL_MCPWM_x mcpwmx )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3765 行定义.

4.37.1.186 HAL_MCPWM_IrqTH50Disable()

```
void HAL_MCPWM_IrqTH50Disable (
    HAL_MCPWM_x mcpwmx )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3761 行定义.

4.37.1.187 HAL_MCPWM_IrqTH50Enable()

```
void HAL_MCPWM_IrqTH50Enable (
    HAL_MCPWM_x mcpwmx )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3757 行定义.

4.37.1.188 HAL_MCPWM_IrqTH50Flg()

```
uint8_t HAL_MCPWM_IrqTH50Flg (
    HAL_MCPWM_x mcpwmx )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3752 行定义.

4.37.1.189 HAL_MCPWM_IrqTH51()

```
uint8_t HAL_MCPWM_IrqTH51 (
    HAL_MCPWM_x mcpwmx )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3770 行定义.

4.37.1.190 HAL_MCPWM_IrqTH51Clear()

```
void HAL_MCPWM_IrqTH51Clear (
    HAL_MCPWM_x mcpwmx )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3788 行定义.

4.37.1.191 HAL_MCPWM_IrqTH51Disable()

```
void HAL_MCPWM_IrqTH51Disable (
    HAL_MCPWM_x mcpwmx )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3784 行定义.

4.37.1.192 HAL_MCPWM_IrqTH51Enable()

```
void HAL_MCPWM_IrqTH51Enable (
    HAL_MCPWM_x mcpwmx )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3780 行定义.

4.37.1.193 HAL_MCPWM_IrqTH51Flg()

```
uint8_t HAL_MCPWM_IrqTH51Flg (
    HAL_MCPWM_x mcpwmx )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3775 行定义.

4.37.1.194 HAL_MCPWM_IrqTmr0()

```
uint8_t HAL_MCPWM_IrqTmr0 (
    HAL_MCPWM_x mcpwmx )
```

中断处理函数

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2009 行定义.

4.37.1.195 HAL_MCPWM_IrqTmr0Clear()

```
void HAL_MCPWM_IrqTmr0Clear (
    HAL_MCPWM_x mcpwmx )
```

中断清除

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2074 行定义.

4.37.1.196 HAL_MCPWM_IrqTmr0Disable()

```
void HAL_MCPWM_IrqTmr0Disable (
    HAL_MCPWM_x mcpwmx )
```

中断禁止

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2061 行定义.

4.37.1.197 HAL_MCPWM_IrqTmr0Enable()

```
void HAL_MCPWM_IrqTmr0Enable (
    HAL_MCPWM_x mcpwmx )
```

中断使能

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2048 行定义.

4.37.1.198 HAL_MCPWM_IrqTmr0Flg()

```
uint8_t HAL_MCPWM_IrqTmr0Flg (
    HAL_MCPWM_x mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqTmr0 的区别是不判断中断使能）

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2030 行定义.

4.37.1.199 HAL_MCPWM_IrqTmr1()

```
uint8_t HAL_MCPWM_IrqTmr1 (  
    HAL_MCPWM_X mcpwmx )
```

中断处理函数

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2086 行定义.

4.37.1.200 HAL_MCPWM_IrqTmr1Clear()

```
void HAL_MCPWM_IrqTmr1Clear (  
    HAL_MCPWM_X mcpwmx )
```

中断清除

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2151 行定义.

4.37.1.201 HAL_MCPWM_IrqTmr1Disable()

```
void HAL_MCPWM_IrqTmr1Disable (  
    HAL_MCPWM_X mcpwmx )
```

中断禁止

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2138 行定义.

4.37.1.202 HAL_MCPWM_IrqTmr1Enable()

```
void HAL_MCPWM_IrqTmr1Enable (  
    HAL\_MCPWM\_x mcpwmx )
```

中断使能

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2125 行定义.

4.37.1.203 HAL_MCPWM_IrqTmr1Flg()

```
uint8_t HAL_MCPWM_IrqTmr1Flg (  
    HAL\_MCPWM\_x mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqTmr1 的区别是不判断中断使能）

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2107 行定义.

4.37.1.204 HAL_MCPWM_IrqTmr2()

```
uint8_t HAL_MCPWM_IrqTmr2 (  
    HAL\_MCPWM\_x mcpwmx )
```

中断处理函数

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2163 行定义.

4.37.1.205 HAL_MCPWM_IrqTmr2Clear()

```
void HAL_MCPWM_IrqTmr2Clear (
    HAL_MCPWM_x mcpwmx )
```

中断清除

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2228 行定义.

4.37.1.206 HAL_MCPWM_IrqTmr2Disable()

```
void HAL_MCPWM_IrqTmr2Disable (
    HAL_MCPWM_x mcpwmx )
```

中断禁止

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2215 行定义.

4.37.1.207 HAL_MCPWM_IrqTmr2Enable()

```
void HAL_MCPWM_IrqTmr2Enable (
    HAL_MCPWM_x mcpwmx )
```

中断使能

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2202 行定义.

4.37.1.208 HAL_MCPWM_IrqTmr2Flg()

```
uint8_t HAL_MCPWM_IrqTmr2Flg (
    HAL_MCPWM_x mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqTmr2 的区别是不判断中断使能）

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2184 行定义.

4.37.1.209 HAL_MCPWM_IrqTmr3()

```
uint8_t HAL_MCPWM_IrqTmr3 (
    HAL_MCPWM_x mcpwmx )
```

中断处理函数

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2240 行定义.

4.37.1.210 HAL_MCPWM_IrqTmr3Clear()

```
void HAL_MCPWM_IrqTmr3Clear (
    HAL_MCPWM_x mcpwmx )
```

中断清除

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2305 行定义.

4.37.1.211 HAL_MCPWM_IrqTmr3Disable()

```
void HAL_MCPWM_IrqTmr3Disable (
    HAL_MCPWM_x mcpwmx )
```

中断禁止

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2292 行定义.

4.37.1.212 HAL_MCPWM_IrqTmr3Enable()

```
void HAL_MCPWM_IrqTmr3Enable (
    HAL_MCPWM_x mcpwmx )
```

中断使能

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2279 行定义.

4.37.1.213 HAL_MCPWM_IrqTmr3Flg()

```
uint8_t HAL_MCPWM_IrqTmr3Flg (
    HAL_MCPWM_x mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqTmr3 的区别是不判断中断使能）

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2261 行定义.

4.37.1.214 HAL_MCPWM_IrqUpdate()

```
uint8_t HAL_MCPWM_IrqUpdate (
    HAL_MCPWM_x mcpwmx )
```

中断处理函数

参数

<i>mcpwm</i> x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1932 行定义.

4.37.1.215 HAL_MCPWM_IrqUpdateClear()

```
void HAL_MCPWM_IrqUpdateClear (
    HAL_MCPWM_x mcpwm )
```

中断清除

参数

<i>mcpwm</i> x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1997 行定义.

4.37.1.216 HAL_MCPWM_IrqUpdateDisable()

```
void HAL_MCPWM_IrqUpdateDisable (
    HAL_MCPWM_x mcpwm )
```

中断禁止

参数

<i>mcpwm</i> x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1984 行定义.

4.37.1.217 HAL_MCPWM_IrqUpdateEnable()

```
void HAL_MCPWM_IrqUpdateEnable (
    HAL_MCPWM_x mcpwm )
```

中断使能

参数

<i>mcpwm</i> x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1971 行定义.

4.37.1.218 HAL_MCPWM_IrqUpdateFlg()

```
uint8_t HAL_MCPWM_IrqUpdateFlg (
    HAL_MCPWM_x mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqUpdate 的区别是不判断中断使能）

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1953 行定义.

4.37.1.219 HAL_MCPWM_IrqZeroCross()

```
uint8_t HAL_MCPWM_IrqZeroCross (
    HAL_MCPWM_x mcpwmx )
```

中断处理函数

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 383 行定义.

4.37.1.220 HAL_MCPWM_IrqZeroCrossClear()

```
void HAL_MCPWM_IrqZeroCrossClear (
    HAL_MCPWM_x mcpwmx )
```

中断清除

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 448 行定义.

4.37.1.221 HAL_MCPWM_IrqZeroCrossDisable()

```
void HAL_MCPWM_IrqZeroCrossDisable (
    HAL_MCPWM_x mcpwmx )
```

中断禁止

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 435 行定义.

4.37.1.222 HAL_MCPWM_IrqZeroCrossEnable()

```
void HAL_MCPWM_IrqZeroCrossEnable (
    HAL_MCPWM_x mcpwmx )
```

中断使能

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 422 行定义.

4.37.1.223 HAL_MCPWM_IrqZeroCrossFlg()

```
uint8_t HAL_MCPWM_IrqZeroCrossFlg (
    HAL_MCPWM_x mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqZeroCross 的区别是不判断中断使能）

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 404 行定义.

4.37.1.224 HAL_SPI_IrqDone()

```
uint8_t HAL_SPI_IrqDone (
    HAL_SPI_TypeDef* spi_x )
```

中断处理函数

参数

spi_x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3229 行定义.

4.37.1.225 HAL_SPI_IrqDoneClear()

```
void HAL_SPI_IrqDoneClear (
    HAL_SPI_TypeDef* spi_x )
```

中断清除

参数

spi_x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3290 行定义.

4.37.1.226 HAL_SPI_IrqDoneDisable()

```
void HAL_SPI_IrqDoneDisable (
    HAL_SPI_TypeDef* spi_x )
```

中断禁止

参数

spi_x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3279 行定义.

4.37.1.227 HAL_SPI_IrqDoneEnable()

```
void HAL_SPI_IrqDoneEnable (
    HAL_SPI_TypeDef* spi_x )
```

中断使能

参数

spi_x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3268 行定义.

4.37.1.228 HAL_SPI_IrqDoneFlg()

```
uint8_t HAL_SPI_IrqDoneFlg (
    HAL_SPI_TypeDef* spi_x )
```

中断信号（和 HAL_SPI_IrqDone 的区别是不判断中断使能）

参数

spi_x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3250 行定义.

4.37.1.229 HAL_SPI_IrqErr()

```
uint8_t HAL_SPI_IrqErr (
    HAL_SPI_TypeDef* spi_x )
```

中断处理函数

参数

spi_x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3302 行定义.

4.37.1.230 HAL_SPI_IrqErrClear()

```
void HAL_SPI_IrqErrClear (
    HAL_SPI_x spi_x )
```

中断清除

参数

spi_x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3363 行定义.

4.37.1.231 HAL_SPI_IrqErrDisable()

```
void HAL_SPI_IrqErrDisable (
    HAL_SPI_x spi_x )
```

中断禁止

参数

spi_x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3352 行定义.

4.37.1.232 HAL_SPI_IrqErrEnable()

```
void HAL_SPI_IrqErrEnable (
    HAL_SPI_x spi_x )
```

中断使能

参数

spi_x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3341 行定义.

4.37.1.233 HAL_SPI_IrqErrFlg()

```
uint8_t HAL_SPI_IrqErrFlg (
    HAL_SPI_x spi_x )
```

中断信号（和 HAL_SPI_IrqErr 的区别是不判断中断使能）

参数

$\text{spi}_{\leftarrow x}$	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3323 行定义.

4.37.1.234 HAL_SPI_IrqOverflow()

```
uint8_t HAL_SPI_IrqOverflow (
    HAL_SPI_x spi_x )
```

中断处理函数

参数

$\text{spi}_{\leftarrow x}$	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3375 行定义.

4.37.1.235 HAL_SPI_IrqOverflowClear()

```
void HAL_SPI_IrqOverflowClear (
    HAL_SPI_x spi_x )
```

中断清除

参数

$\text{spi}_{\leftarrow x}$	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3436 行定义.

4.37.1.236 HAL_SPI_IrqOverflowDisable()

```
void HAL_SPI_IrqOverflowDisable (
    HAL_SPI_x spi_x )
```

中断禁止

参数

spi_x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3425 行定义.

4.37.1.237 HAL_SPI_IrqOverflowEnable()

```
void HAL_SPI_IrqOverflowEnable (
    HAL_SPI_x spi_x )
```

中断使能

参数

spi_x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3414 行定义.

4.37.1.238 HAL_SPI_IrqOverflowFlg()

```
uint8_t HAL_SPI_IrqOverflowFlg (
    HAL_SPI_x spi_x )
```

中断信号（和 HAL_SPI_IrqOverflow 的区别是不判断中断使能）

参数

spi_x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3396 行定义.

4.37.1.239 HAL_SW_Irq()

```
uint8_t HAL_SW_Irq (
    uint8_t x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3793 行定义.

4.37.1.240 HAL_SW_IrqClear()

```
void HAL_SW_IrqClear (
    uint8_t x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3811 行定义.

4.37.1.241 HAL_SW_IrqDisable()

```
void HAL_SW_IrqDisable (
    uint8_t x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3807 行定义.

4.37.1.242 HAL_SW_IrqEnable()

```
void HAL_SW_IrqEnable (
    uint8_t x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3803 行定义.

4.37.1.243 HAL_SW_IrqFlg()

```
uint8_t HAL_SW_IrqFlg (
    uint8_t x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3798 行定义.

4.37.1.244 HAL_SW_IrqTrig()

```
void HAL_SW_IrqTrig (
    uint8_t x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3816 行定义.

4.37.1.245 HAL_TIMER_IrqCompare0()

```
uint8_t HAL_TIMER_IrqCompare0 (
    HAL_TIMER_x timer_x )
```

中断处理函数

参数

<i>timer</i> ↔ _x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 837 行定义.

4.37.1.246 HAL_TIMER_IrqCompare0Clear()

```
void HAL_TIMER_IrqCompare0Clear (
    HAL_TIMER_X timer_x )
```

中断清除

参数

<i>timer</i> ↔ _X	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 898 行定义.

4.37.1.247 HAL_TIMER_IrqCompare0Disable()

```
void HAL_TIMER_IrqCompare0Disable (
    HAL_TIMER_X timer_x )
```

中断禁止

参数

<i>timer</i> ↔ _X	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 887 行定义.

4.37.1.248 HAL_TIMER_IrqCompare0Enable()

```
void HAL_TIMER_IrqCompare0Enable (
    HAL_TIMER_X timer_x )
```

中断使能

参数

<i>timer</i> ↔ _X	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 876 行定义.

4.37.1.249 HAL_TIMER_IrqCompare0Flg()

```
uint8_t HAL_TIMER_IrqCompare0Flg (  
    HAL_TIMER_x timer_x )
```

中断信号（和 HAL_TIMER_IrqCompare0 的区别是不判断中断使能）

参数

<i>timer</i> ↔ _x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 858 行定义.

4.37.1.250 HAL_TIMER_IrqCompare1()

```
uint8_t HAL_TIMER_IrqCompare1 (  
    HAL_TIMER_x timer_x )
```

中断处理函数

参数

<i>timer</i> ↔ _x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 910 行定义.

4.37.1.251 HAL_TIMER_IrqCompare1Clear()

```
void HAL_TIMER_IrqCompare1Clear (  
    HAL_TIMER_x timer_x )
```

中断清除

参数

<i>timer</i> ↔ _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 971 行定义.

4.37.1.252 HAL_TIMER_IrqCompare1Disable()

```
void HAL_TIMER_IrqCompare1Disable (
    HAL_TIMER_x timer_x )
```

中断禁止

参数

<i>timer_x</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 960 行定义.

4.37.1.253 HAL_TIMER_IrqCompare1Enable()

```
void HAL_TIMER_IrqCompare1Enable (
    HAL_TIMER_x timer_x )
```

中断使能

参数

<i>timer_x</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 949 行定义.

4.37.1.254 HAL_TIMER_IrqCompare1Flg()

```
uint8_t HAL_TIMER_IrqCompare1Flg (
    HAL_TIMER_x timer_x )
```

中断信号（和 HAL_TIMER_IrqCompare1 的区别是不判断中断使能）

参数

<i>timer_x</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 931 行定义.

4.37.1.255 HAL_TIMER_IrqOverflow()

```
uint8_t HAL_TIMER_IrqOverflow (
    HAL_TIMER_x timer_x )
```

中断处理函数

参数

<i>timer_x</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 764 行定义.

4.37.1.256 HAL_TIMER_IrqOverflowClear()

```
void HAL_TIMER_IrqOverflowClear (
    HAL_TIMER_x timer_x )
```

中断清除

参数

<i>timer_x</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 825 行定义.

4.37.1.257 HAL_TIMER_IrqOverflowDisable()

```
void HAL_TIMER_IrqOverflowDisable (
    HAL_TIMER_x timer_x )
```

中断禁止

参数

<i>timer_x</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 814 行定义.

4.37.1.258 HAL_TIMER_IrqOverflowEnable()

```
void HAL_TIMER_IrqOverflowEnable (
    HAL_TIMER_x timer_x )
```

中断使能

参数

<i>timer_x</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 803 行定义.

4.37.1.259 HAL_TIMER_IrqOverflowFlg()

```
uint8_t HAL_TIMER_IrqOverflowFlg (
    HAL_TIMER_x timer_x )
```

中断信号（和 HAL_TIMER_IrqOverflow 的区别是不判断中断使能）

参数

<i>timer_x</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 785 行定义.

4.37.1.260 HAL_UART_IrqAbd()

```
uint8_t HAL_UART_IrqAbd (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3821 行定义.

4.37.1.261 HAL_UART_IrqAbdClear()

```
void HAL_UART_IrqAbdClear (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3839 行定义.

4.37.1.262 HAL_UART_IrqAbdDisable()

```
void HAL_UART_IrqAbdDisable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3835 行定义.

4.37.1.263 HAL_UART_IrqAbdEnable()

```
void HAL_UART_IrqAbdEnable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3831 行定义.

4.37.1.264 HAL_UART_IrqAbdFlg()

```
uint8_t HAL_UART_IrqAbdFlg (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3826 行定义.

4.37.1.265 HAL_UART_IrqCheckErr()

```
uint8_t HAL_UART_IrqCheckErr (
    HAL_UART_x uart_x )
```

中断处理函数

参数

<i>uart_x</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1202 行定义.

4.37.1.266 HAL_UART_IrqCheckErrClear()

```
void HAL_UART_IrqCheckErrClear (
    HAL_UART_x uart_x )
```

中断清除

参数

<i>uart</i> _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1263 行定义.

4.37.1.267 HAL_UART_IrqCheckErrDisable()

```
void HAL_UART_IrqCheckErrDisable (
    HAL_UART_x uart_x )
```

中断禁止

参数

<i>uart</i> _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1252 行定义.

4.37.1.268 HAL_UART_IrqCheckErrEnable()

```
void HAL_UART_IrqCheckErrEnable (
    HAL_UART_x uart_x )
```

中断使能

参数

<i>uart</i> _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1241 行定义.

4.37.1.269 HAL_UART_IrqCheckErrFlg()

```
uint8_t HAL_UART_IrqCheckErrFlg (
    HAL_UART_x uart_x )
```

中断信号（和 HAL_UART_IrqCheckErr 的区别是不判断中断使能）

参数

<i>uart</i> _x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1223 行定义.

4.37.1.270 HAL_UART_IrqIdle()

```
uint8_t HAL_UART_IrqIdle (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3844 行定义.

4.37.1.271 HAL_UART_IrqIdleClear()

```
void HAL_UART_IrqIdleClear (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3862 行定义.

4.37.1.272 HAL_UART_IrqIdleDisable()

```
void HAL_UART_IrqIdleDisable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3858 行定义.

4.37.1.273 HAL_UART_IrqIdleEnable()

```
void HAL_UART_IrqIdleEnable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3854 行定义.

4.37.1.274 HAL_UART_IrqIdleFlg()

```
uint8_t HAL_UART_IrqIdleFlg (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3849 行定义.

4.37.1.275 HAL_UART_IrqLbd()

```
uint8_t HAL_UART_IrqLbd (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3867 行定义.

4.37.1.276 HAL_UART_IrqLbdClear()

```
void HAL_UART_IrqLbdClear (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3885 行定义.

4.37.1.277 HAL_UART_IrqLbdDisable()

```
void HAL_UART_IrqLbdDisable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3881 行定义.

4.37.1.278 HAL_UART_IrqLbdEnable()

```
void HAL_UART_IrqLbdEnable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3877 行定义.

4.37.1.279 HAL_UART_IrqLbdFlg()

```
uint8_t HAL_UART_IrqLbdFlg (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3872 行定义.

4.37.1.280 HAL_UART_IrqRxDone()

```
uint8_t HAL_UART_IrqRxDone (
    HAL_UART_x uart_x )
```

中断处理函数

参数

<i>uart_x</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1056 行定义.

4.37.1.281 HAL_UART_IrqRxDoneClear()

```
void HAL_UART_IrqRxDoneClear (
    HAL_UART_x uart_x )
```

中断清除

参数

<i>uart</i> _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1117 行定义.

4.37.1.282 HAL_UART_IrqRxDoneDisable()

```
void HAL_UART_IrqRxDoneDisable (
    HAL_UART_x uart_x )
```

中断禁止

参数

<i>uart</i> _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1106 行定义.

4.37.1.283 HAL_UART_IrqRxDoneEnable()

```
void HAL_UART_IrqRxDoneEnable (
    HAL_UART_x uart_x )
```

中断使能

参数

<i>uart</i> _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1095 行定义.

4.37.1.284 HAL_UART_IrqRxDoneFlg()

```
uint8_t HAL_UART_IrqRxDoneFlg (
    HAL_UART_x uart_x )
```

中断信号（和 HAL_UART_IrqRxDone 的区别是不判断中断使能）

参数

<code>uart_</code> <code>x</code>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1077 行定义.

4.37.1.285 HAL_UART_IrqRxOv()

```
uint8_t HAL_UART_IrqRxOv (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3890 行定义.

4.37.1.286 HAL_UART_IrqRxOvClear()

```
void HAL_UART_IrqRxOvClear (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3908 行定义.

4.37.1.287 HAL_UART_IrqRxOvDisable()

```
void HAL_UART_IrqRxOvDisable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3904 行定义.

4.37.1.288 HAL_UART_IrqRxOvEnable()

```
void HAL_UART_IrqRxOvEnable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3900 行定义.

4.37.1.289 HAL_UART_IrqRxOvFlg()

```
uint8_t HAL_UART_IrqRxOvFlg (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3895 行定义.

4.37.1.290 HAL_UART_IrqStopErr()

```
uint8_t HAL_UART_IrqStopErr (
    HAL_UART_x uart_x )
```

中断处理函数

参数

<i>uart</i> _{<i>_x</i>}	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1129 行定义.

4.37.1.291 HAL_UART_IrqStopErrClear()

```
void HAL_UART_IrqStopErrClear (
    HAL_UART_<i>x</i> <i>uart_x</i> )
```

中断清除

参数

<i>uart</i> _{<i>_x</i>}	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1190 行定义.

4.37.1.292 HAL_UART_IrqStopErrDisable()

```
void HAL_UART_IrqStopErrDisable (
    HAL_UART_<i>x</i> <i>uart_x</i> )
```

中断禁止

参数

<i>uart</i> _{<i>_x</i>}	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1179 行定义.

4.37.1.293 HAL_UART_IrqStopErrEnable()

```
void HAL_UART_IrqStopErrEnable (
    HAL_UART_<i>x</i> <i>uart_x</i> )
```

中断使能

参数

<i>uart</i> ↔ _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1168 行定义.

4.37.1.294 HAL_UART_IrqStopErrFlg()

```
uint8_t HAL_UART_IrqStopErrFlg (  
    HAL_UART_x uart_x )
```

中断信号（和 HAL_UART_IrqStopErr 的区别是不判断中断使能）

参数

<i>uart</i> ↔ _x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1150 行定义.

4.37.1.295 HAL_UART_IrqTxBufEmpty()

```
uint8_t HAL_UART_IrqTxBufEmpty (  
    HAL_UART_x uart_x )
```

中断处理函数

参数

<i>uart</i> ↔ _x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1494 行定义.

4.37.1.296 HAL_UART_IrqTxBufEmptyClear()

```
void HAL_UART_IrqTxBufEmptyClear (  
    HAL_UART_x uart_x )
```

中断清除

参数

<i>uart</i> _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1555 行定义.

4.37.1.297 HAL_UART_IrqTxBufEmptyDisable()

```
void HAL_UART_IrqTxBufEmptyDisable (
    HAL_UART_x uart_x )
```

中断禁止

参数

<i>uart</i> _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1544 行定义.

4.37.1.298 HAL_UART_IrqTxBufEmptyEnable()

```
void HAL_UART_IrqTxBufEmptyEnable (
    HAL_UART_x uart_x )
```

中断使能

参数

<i>uart</i> _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1533 行定义.

4.37.1.299 HAL_UART_IrqTxBufEmptyFlg()

```
uint8_t HAL_UART_IrqTxBufEmptyFlg (
    HAL_UART_x uart_x )
```

中断信号（和 HAL_UART_IrqTxBufEmpty 的区别是不判断中断使能）

参数

<i>uart</i> _x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1515 行定义.

4.37.1.300 HAL_UART_IrqTxDone()

```
uint8_t HAL_UART_IrqTxDone (
    HAL_UART_<_x uart_<_x )
```

中断处理函数

参数

<i>uart_<_x</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 983 行定义.

4.37.1.301 HAL_UART_IrqTxDoneClear()

```
void HAL_UART_IrqTxDoneClear (
    HAL_UART_<_x uart_<_x )
```

中断清除

参数

<i>uart_<_x</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1044 行定义.

4.37.1.302 HAL_UART_IrqTxDoneDisable()

```
void HAL_UART_IrqTxDoneDisable (
    HAL_UART_<_x uart_<_x )
```

中断禁止

参数

<i>uart</i> _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1033 行定义.

4.37.1.303 HAL_UART_IrqTxDoneEnable()

```
void HAL_UART_IrqTxDoneEnable (  
    HAL_UART_<i>x</i> <i>uart_x</i> )
```

中断使能

参数

<i>uart</i> _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1022 行定义.

4.37.1.304 HAL_UART_IrqTxDoneFlg()

```
uint8_t HAL_UART_IrqTxDoneFlg (  
    HAL_UART_<i>x</i> <i>uart_x</i> )
```

中断信号（和 HAL_UART_IrqTxDone 的区别是不判断中断使能）

参数

<i>uart</i> _x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1004 行定义.

4.37.1.305 HAL_UART_IrqTxOv()

```
uint8_t HAL_UART_IrqTxOv (  
    HAL_UART_<i>x</i> <i>uart_x</i> )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3913 行定义.

4.37.1.306 HAL_UART_IrqTxOvClear()

```
void HAL_UART_IrqTxOvClear (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3931 行定义.

4.37.1.307 HAL_UART_IrqTxOvDisable()

```
void HAL_UART_IrqTxOvDisable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3927 行定义.

4.37.1.308 HAL_UART_IrqTxOvEnable()

```
void HAL_UART_IrqTxOvEnable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3923 行定义.

4.37.1.309 HAL_UART_IrqTxOvFlg()

```
uint8_t HAL_UART_IrqTxOvFlg (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3918 行定义.

4.38 hal_lksmcu_irqfun.h

[浏览该文件的文档.](#)

```
00001
00002 #ifndef __HAL_LKSMCU_IRQFUN_H__
00003 #define __HAL_LKSMCU_IRQFUN_H__
00004
00005 #include "../hal_lks32mc08x.h"
00006
00007 uint8_t HAL_HALL_IrqChange (HAL_HALL_x hall);
00008 uint8_t HAL_HALL_IrqChangeFlg (HAL_HALL_x hall);
00009 void HAL_HALL_IrqChangeEnable (HAL_HALL_x hall);
00010 void HAL_HALL_IrqChangeDisable (HAL_HALL_x hall);
00011 void HAL_HALL_IrqChangeClear (HAL_HALL_x hall);
00012
00013 uint8_t HAL_HALL_IrqOver (HAL_HALL_x hall);
00014 uint8_t HAL_HALL_IrqOverFlg (HAL_HALL_x hall);
00015 void HAL_HALL_IrqOverEnable (HAL_HALL_x hall);
00016 void HAL_HALL_IrqOverDisable (HAL_HALL_x hall);
00017 void HAL_HALL_IrqOverClear (HAL_HALL_x hall);
00018
00019 uint8_t HAL_ADC_IrqSamp1 (HAL_ADC_x adc);
00020 uint8_t HAL_ADC_IrqSamp1Flg (HAL_ADC_x adc);
00021 void HAL_ADC_IrqSamp1Enable (HAL_ADC_x adc);
00022 void HAL_ADC_IrqSamp1Disable (HAL_ADC_x adc);
00023 void HAL_ADC_IrqSamp1Clear (HAL_ADC_x adc);
00024
00025 uint8_t HAL_ADC_IrqSamp2 (HAL_ADC_x adc);
00026 uint8_t HAL_ADC_IrqSamp2Flg (HAL_ADC_x adc);
00027 void HAL_ADC_IrqSamp2Enable (HAL_ADC_x adc);
00028 void HAL_ADC_IrqSamp2Disable (HAL_ADC_x adc);
00029 void HAL_ADC_IrqSamp2Clear (HAL_ADC_x adc);
```

```
00030
00031 uint8_t HAL_MCPWM_IrqOverflow(HAL_MCPWM_x);
00032 uint8_t HAL_MCPWM_IrqOverflowFlg(HAL_MCPWM_x mcpwmx);
00033 void HAL_MCPWM_IrqOverflowEnable(HAL_MCPWM_x);
00034 void HAL_MCPWM_IrqOverflowDisable(HAL_MCPWM_x mcpwmx);
00035 void HAL_MCPWM_IrqOverflowClear(HAL_MCPWM_x mcpwmx);
00036
00037 uint8_t HAL_MCPWM_IrqZeroCross(HAL_MCPWM_x);
00038 uint8_t HAL_MCPWM_IrqZeroCrossFlg(HAL_MCPWM_x mcpwmx);
00039 void HAL_MCPWM_IrqZeroCrossEnable(HAL_MCPWM_x);
00040 void HAL_MCPWM_IrqZeroCrossDisable(HAL_MCPWM_x mcpwmx);
00041 void HAL_MCPWM_IrqZeroCrossClear(HAL_MCPWM_x mcpwmx);
00042
00043 uint8_t HAL_MCPWM_IrqFail0(HAL_MCPWM_x);
00044 uint8_t HAL_MCPWM_IrqFail0Flg(HAL_MCPWM_x mcpwmx);
00045 void HAL_MCPWM_IrqFail0Enable(HAL_MCPWM_x);
00046 void HAL_MCPWM_IrqFail0Disable(HAL_MCPWM_x mcpwmx);
00047 void HAL_MCPWM_IrqFail0Clear(HAL_MCPWM_x mcpwmx);
00048
00049 uint8_t HAL_MCPWM_IrqFail1(HAL_MCPWM_x);
00050 uint8_t HAL_MCPWM_IrqFail1Flg(HAL_MCPWM_x mcpwmx);
00051 void HAL_MCPWM_IrqFail1Enable(HAL_MCPWM_x);
00052 void HAL_MCPWM_IrqFail1Disable(HAL_MCPWM_x mcpwmx);
00053 void HAL_MCPWM_IrqFail1Clear(HAL_MCPWM_x mcpwmx);
00054
00055 uint8_t HAL_MCPWM_IrqFail(HAL_MCPWM_x);
00056 uint8_t HAL_MCPWM_IrqFailFlg(HAL_MCPWM_x mcpwmx);
00057 void HAL_MCPWM_IrqFailEnable(HAL_MCPWM_x);
00058 void HAL_MCPWM_IrqFailDisable(HAL_MCPWM_x mcpwmx);
00059 void HAL_MCPWM_IrqFailClear(HAL_MCPWM_x mcpwmx);
00060
00061 uint8_t HAL_CMP_IrqCompare(HAL_CMP_x);
00062 uint8_t HAL_CMP_IrqCompareFlg(HAL_CMP_x cmp_x);
00063 void HAL_CMP_IrqCompareEnable(HAL_CMP_x);
00064 void HAL_CMP_IrqCompareDisable(HAL_CMP_x cmp_x);
00065 void HAL_CMP_IrqCompareClear(HAL_CMP_x cmp_x);
00066
00067 uint8_t HAL_TIMER_IrqOverflow(HAL_TIMER_x);
00068 uint8_t HAL_TIMER_IrqOverflowFlg(HAL_TIMER_x timer_x);
00069 void HAL_TIMER_IrqOverflowEnable(HAL_TIMER_x);
00070 void HAL_TIMER_IrqOverflowDisable(HAL_TIMER_x timer_x);
00071 void HAL_TIMER_IrqOverflowClear(HAL_TIMER_x timer_x);
00072
00073 uint8_t HAL_TIMER_IrqCompare0(HAL_TIMER_x);
00074 uint8_t HAL_TIMER_IrqCompare0Flg(HAL_TIMER_x timer_x);
00075 void HAL_TIMER_IrqCompare0Enable(HAL_TIMER_x);
00076 void HAL_TIMER_IrqCompare0Disable(HAL_TIMER_x timer_x);
00077 void HAL_TIMER_IrqCompare0Clear(HAL_TIMER_x timer_x);
00078
00079 uint8_t HAL_TIMER_IrqCompare1(HAL_TIMER_x);
00080 uint8_t HAL_TIMER_IrqCompare1Flg(HAL_TIMER_x timer_x);
00081 void HAL_TIMER_IrqCompare1Enable(HAL_TIMER_x);
00082 void HAL_TIMER_IrqCompare1Disable(HAL_TIMER_x timer_x);
00083 void HAL_TIMER_IrqCompare1Clear(HAL_TIMER_x timer_x);
00084
00085 uint8_t HAL_UART_IrqTxDone(HAL_UART_x);
00086 uint8_t HAL_UART_IrqTxDoneFlg(HAL_UART_x uart_x);
00087 void HAL_UART_IrqTxDoneEnable(HAL_UART_x);
00088 void HAL_UART_IrqTxDoneDisable(HAL_UART_x uart_x);
00089 void HAL_UART_IrqTxDoneClear(HAL_UART_x uart_x);
00090
00091 uint8_t HAL_UART_IrqRxDone(HAL_UART_x);
00092 uint8_t HAL_UART_IrqRxDoneFlg(HAL_UART_x uart_x);
00093 void HAL_UART_IrqRxDoneEnable(HAL_UART_x);
00094 void HAL_UART_IrqRxDoneDisable(HAL_UART_x uart_x);
00095 void HAL_UART_IrqRxDoneClear(HAL_UART_x uart_x);
00096
00097 uint8_t HAL_UART_IrqStopErr(HAL_UART_x);
00098 uint8_t HAL_UART_IrqStopErrFlg(HAL_UART_x uart_x);
00099 void HAL_UART_IrqStopErrEnable(HAL_UART_x);
00100 void HAL_UART_IrqStopErrDisable(HAL_UART_x uart_x);
00101 void HAL_UART_IrqStopErrClear(HAL_UART_x uart_x);
00102
00103 uint8_t HAL_UART_IrqCheckErr(HAL_UART_x);
00104 uint8_t HAL_UART_IrqCheckErrFlg(HAL_UART_x uart_x);
00105 void HAL_UART_IrqCheckErrEnable(HAL_UART_x);
00106 void HAL_UART_IrqCheckErrDisable(HAL_UART_x uart_x);
00107 void HAL_UART_IrqCheckErrClear(HAL_UART_x uart_x);
00108
00109 uint8_t HAL_CAN_IrqRxDone(HAL_CAN_x);
00110 uint8_t HAL_CAN_IrqRxDoneFlg(HAL_CAN_x can_x);
00111 void HAL_CAN_IrqRxDoneEnable(HAL_CAN_x);
00112 void HAL_CAN_IrqRxDoneDisable(HAL_CAN_x can_x);
00113 void HAL_CAN_IrqRxDoneClear(HAL_CAN_x can_x);
00114
00115 uint8_t HAL_CAN_IrqTxDone(HAL_CAN_x);
00116 uint8_t HAL_CAN_IrqTxDoneFlg(HAL_CAN_x can_x);
```

```

00117 void HAL_CAN_IrqTxDoneEnable (HAL_CAN_x);
00118 void HAL_CAN_IrqTxDoneDisable (HAL_CAN_x can_x);
00119 void HAL_CAN_IrqTxDoneClear (HAL_CAN_x can_x);
00120
00121 uint8_t HAL_CAN_IrqAll (HAL_CAN_x);
00122 uint8_t HAL_CAN_IrqAllFlg (HAL_CAN_x can_x);
00123 void HAL_CAN_IrqAllEnable (HAL_CAN_x);
00124 void HAL_CAN_IrqAllDisable (HAL_CAN_x can_x);
00125 void HAL_CAN_IrqAllClear (HAL_CAN_x can_x);
00126
00127 uint8_t HAL_UART_IrqTxBufEmpty (HAL_UART_x);
00128 uint8_t HAL_UART_IrqTxBufEmptyFlg (HAL_UART_x uart_x);
00129 void HAL_UART_IrqTxBufEmptyEnable (HAL_UART_x);
00130 void HAL_UART_IrqTxBufEmptyDisable (HAL_UART_x uart_x);
00131 void HAL_UART_IrqTxBufEmptyClear (HAL_UART_x uart_x);
00132
00133 uint8_t HAL_ADC_IrqSamp3 (HAL_ADC_x);
00134 uint8_t HAL_ADC_IrqSamp3Flg (HAL_ADC_x adc);
00135 void HAL_ADC_IrqSamp3Enable (HAL_ADC_x);
00136 void HAL_ADC_IrqSamp3Disable (HAL_ADC_x adc);
00137 void HAL_ADC_IrqSamp3Clear (HAL_ADC_x adc);
00138
00139 uint8_t HAL_ADC_IrqSamp4 (HAL_ADC_x);
00140 uint8_t HAL_ADC_IrqSamp4Flg (HAL_ADC_x adc);
00141 void HAL_ADC_IrqSamp4Enable (HAL_ADC_x);
00142 void HAL_ADC_IrqSamp4Disable (HAL_ADC_x adc);
00143 void HAL_ADC_IrqSamp4Clear (HAL_ADC_x adc);
00144
00145 uint8_t HAL_ADC_IrqWatchdog (HAL_ADC_x);
00146 uint8_t HAL_ADC_IrqWatchdogFlg (HAL_ADC_x adc);
00147 void HAL_ADC_IrqWatchdogEnable (HAL_ADC_x);
00148 void HAL_ADC_IrqWatchdogDisable (HAL_ADC_x adc);
00149 void HAL_ADC_IrqWatchdogClear (HAL_ADC_x adc);
00150
00151 uint8_t HAL_ADC_IrqHardErr (HAL_ADC_x);
00152 uint8_t HAL_ADC_IrqHardErrFlg (HAL_ADC_x adc);
00153 void HAL_ADC_IrqHardErrEnable (HAL_ADC_x);
00154 void HAL_ADC_IrqHardErrDisable (HAL_ADC_x adc);
00155 void HAL_ADC_IrqHardErrClear (HAL_ADC_x adc);
00156
00157 uint8_t HAL_ADC_IrqSoftErr (HAL_ADC_x);
00158 uint8_t HAL_ADC_IrqSoftErrFlg (HAL_ADC_x adc);
00159 void HAL_ADC_IrqSoftErrEnable (HAL_ADC_x);
00160 void HAL_ADC_IrqSoftErrDisable (HAL_ADC_x adc);
00161 void HAL_ADC_IrqSoftErrClear (HAL_ADC_x adc);
00162
00163 uint8_t HAL_MCPWM_IrqUpdate (HAL_MCPWM_x);
00164 uint8_t HAL_MCPWM_IrqUpdateFlg (HAL_MCPWM_x mcpwm_x);
00165 void HAL_MCPWM_IrqUpdateEnable (HAL_MCPWM_x);
00166 void HAL_MCPWM_IrqUpdateDisable (HAL_MCPWM_x mcpwm_x);
00167 void HAL_MCPWM_IrqUpdateClear (HAL_MCPWM_x mcpwm_x);
00168
00169 uint8_t HAL_MCPWM_IrqTmr0 (HAL_MCPWM_x);
00170 uint8_t HAL_MCPWM_IrqTmr0Flg (HAL_MCPWM_x mcpwm_x);
00171 void HAL_MCPWM_IrqTmr0Enable (HAL_MCPWM_x);
00172 void HAL_MCPWM_IrqTmr0Disable (HAL_MCPWM_x mcpwm_x);
00173 void HAL_MCPWM_IrqTmr0Clear (HAL_MCPWM_x mcpwm_x);
00174
00175 uint8_t HAL_MCPWM_IrqTmr1 (HAL_MCPWM_x);
00176 uint8_t HAL_MCPWM_IrqTmr1Flg (HAL_MCPWM_x mcpwm_x);
00177 void HAL_MCPWM_IrqTmr1Enable (HAL_MCPWM_x);
00178 void HAL_MCPWM_IrqTmr1Disable (HAL_MCPWM_x mcpwm_x);
00179 void HAL_MCPWM_IrqTmr1Clear (HAL_MCPWM_x mcpwm_x);
00180
00181 uint8_t HAL_MCPWM_IrqTmr2 (HAL_MCPWM_x);
00182 uint8_t HAL_MCPWM_IrqTmr2Flg (HAL_MCPWM_x mcpwm_x);
00183 void HAL_MCPWM_IrqTmr2Enable (HAL_MCPWM_x);
00184 void HAL_MCPWM_IrqTmr2Disable (HAL_MCPWM_x mcpwm_x);
00185 void HAL_MCPWM_IrqTmr2Clear (HAL_MCPWM_x mcpwm_x);
00186
00187 uint8_t HAL_MCPWM_IrqTmr3 (HAL_MCPWM_x);
00188 uint8_t HAL_MCPWM_IrqTmr3Flg (HAL_MCPWM_x mcpwm_x);
00189 void HAL_MCPWM_IrqTmr3Enable (HAL_MCPWM_x);
00190 void HAL_MCPWM_IrqTmr3Disable (HAL_MCPWM_x mcpwm_x);
00191 void HAL_MCPWM_IrqTmr3Clear (HAL_MCPWM_x mcpwm_x);
00192
00193 uint8_t HAL_MCPWM_IrqTH00 (HAL_MCPWM_x);
00194 uint8_t HAL_MCPWM_IrqTH00Flg (HAL_MCPWM_x mcpwm_x);
00195 void HAL_MCPWM_IrqTH00Enable (HAL_MCPWM_x);
00196 void HAL_MCPWM_IrqTH00Disable (HAL_MCPWM_x mcpwm_x);
00197 void HAL_MCPWM_IrqTH00Clear (HAL_MCPWM_x mcpwm_x);
00198
00199 uint8_t HAL_MCPWM_IrqTH01 (HAL_MCPWM_x);
00200 uint8_t HAL_MCPWM_IrqTH01Flg (HAL_MCPWM_x mcpwm_x);
00201 void HAL_MCPWM_IrqTH01Enable (HAL_MCPWM_x);
00202 void HAL_MCPWM_IrqTH01Disable (HAL_MCPWM_x mcpwm_x);
00203 void HAL_MCPWM_IrqTH01Clear (HAL_MCPWM_x mcpwm_x);

```



```
00204
00205 uint8_t HAL_MCPWM_IrqTH10(HAL_MCPWM_x);
00206 uint8_t HAL_MCPWM_IrqTH10Flg(HAL_MCPWM_x mcpwmx);
00207 void HAL_MCPWM_IrqTH10Enable(HAL_MCPWM_x);
00208 void HAL_MCPWM_IrqTH10Disable(HAL_MCPWM_x mcpwmx);
00209 void HAL_MCPWM_IrqTH10Clear(HAL_MCPWM_x mcpwmx);
00210
00211 uint8_t HAL_MCPWM_IrqTH11(HAL_MCPWM_x);
00212 uint8_t HAL_MCPWM_IrqTH11Flg(HAL_MCPWM_x mcpwmx);
00213 void HAL_MCPWM_IrqTH11Enable(HAL_MCPWM_x);
00214 void HAL_MCPWM_IrqTH11Disable(HAL_MCPWM_x mcpwmx);
00215 void HAL_MCPWM_IrqTH11Clear(HAL_MCPWM_x mcpwmx);
00216
00217 uint8_t HAL_MCPWM_IrqTH20(HAL_MCPWM_x);
00218 uint8_t HAL_MCPWM_IrqTH20Flg(HAL_MCPWM_x mcpwmx);
00219 void HAL_MCPWM_IrqTH20Enable(HAL_MCPWM_x);
00220 void HAL_MCPWM_IrqTH20Disable(HAL_MCPWM_x mcpwmx);
00221 void HAL_MCPWM_IrqTH20Clear(HAL_MCPWM_x mcpwmx);
00222
00223 uint8_t HAL_MCPWM_IrqTH21(HAL_MCPWM_x);
00224 uint8_t HAL_MCPWM_IrqTH21Flg(HAL_MCPWM_x mcpwmx);
00225 void HAL_MCPWM_IrqTH21Enable(HAL_MCPWM_x);
00226 void HAL_MCPWM_IrqTH21Disable(HAL_MCPWM_x mcpwmx);
00227 void HAL_MCPWM_IrqTH21Clear(HAL_MCPWM_x mcpwmx);
00228
00229 uint8_t HAL_MCPWM_IrqTH30(HAL_MCPWM_x);
00230 uint8_t HAL_MCPWM_IrqTH30Flg(HAL_MCPWM_x mcpwmx);
00231 void HAL_MCPWM_IrqTH30Enable(HAL_MCPWM_x);
00232 void HAL_MCPWM_IrqTH30Disable(HAL_MCPWM_x mcpwmx);
00233 void HAL_MCPWM_IrqTH30Clear(HAL_MCPWM_x mcpwmx);
00234
00235 uint8_t HAL_MCPWM_IrqTH31(HAL_MCPWM_x);
00236 uint8_t HAL_MCPWM_IrqTH31Flg(HAL_MCPWM_x mcpwmx);
00237 void HAL_MCPWM_IrqTH31Enable(HAL_MCPWM_x);
00238 void HAL_MCPWM_IrqTH31Disable(HAL_MCPWM_x mcpwmx);
00239 void HAL_MCPWM_IrqTH31Clear(HAL_MCPWM_x mcpwmx);
00240
00241 uint8_t HAL_MCPWM_IrqShort(HAL_MCPWM_x);
00242 uint8_t HAL_MCPWM_IrqShortFlg(HAL_MCPWM_x mcpwmx);
00243 void HAL_MCPWM_IrqShortEnable(HAL_MCPWM_x);
00244 void HAL_MCPWM_IrqShortDisable(HAL_MCPWM_x mcpwmx);
00245 void HAL_MCPWM_IrqShortClear(HAL_MCPWM_x mcpwmx);
00246
00247 uint8_t HAL_I2C_IrqDone(HAL_I2C_x);
00248 uint8_t HAL_I2C_IrqDoneFlg(HAL_I2C_x i2c_x);
00249 void HAL_I2C_IrqDoneEnable(HAL_I2C_x);
00250 void HAL_I2C_IrqDoneDisable(HAL_I2C_x i2c_x);
00251 void HAL_I2C_IrqDoneClear(HAL_I2C_x i2c_x);
00252
00253 uint8_t HAL_I2C_IrqBusErr(HAL_I2C_x);
00254 uint8_t HAL_I2C_IrqBusErrFlg(HAL_I2C_x i2c_x);
00255 void HAL_I2C_IrqBusErrEnable(HAL_I2C_x);
00256 void HAL_I2C_IrqBusErrDisable(HAL_I2C_x i2c_x);
00257 void HAL_I2C_IrqBusErrClear(HAL_I2C_x i2c_x);
00258
00259 uint8_t HAL_I2C_IrqStop(HAL_I2C_x);
00260 uint8_t HAL_I2C_IrqStopFlg(HAL_I2C_x i2c_x);
00261 void HAL_I2C_IrqStopEnable(HAL_I2C_x);
00262 void HAL_I2C_IrqStopDisable(HAL_I2C_x i2c_x);
00263 void HAL_I2C_IrqStopClear(HAL_I2C_x i2c_x);
00264
00265 uint8_t HAL_SPI_IrqDone(HAL_SPI_x);
00266 uint8_t HAL_SPI_IrqDoneFlg(HAL_SPI_x spi_x);
00267 void HAL_SPI_IrqDoneEnable(HAL_SPI_x);
00268 void HAL_SPI_IrqDoneDisable(HAL_SPI_x spi_x);
00269 void HAL_SPI_IrqDoneClear(HAL_SPI_x spi_x);
00270
00271 uint8_t HAL_SPI_IrqErr(HAL_SPI_x);
00272 uint8_t HAL_SPI_IrqErrFlg(HAL_SPI_x spi_x);
00273 void HAL_SPI_IrqErrEnable(HAL_SPI_x);
00274 void HAL_SPI_IrqErrDisable(HAL_SPI_x spi_x);
00275 void HAL_SPI_IrqErrClear(HAL_SPI_x spi_x);
00276
00277 uint8_t HAL_SPI_IrqOverflow(HAL_SPI_x);
00278 uint8_t HAL_SPI_IrqOverflowFlg(HAL_SPI_x spi_x);
00279 void HAL_SPI_IrqOverflowEnable(HAL_SPI_x);
00280 void HAL_SPI_IrqOverflowDisable(HAL_SPI_x spi_x);
00281 void HAL_SPI_IrqOverflowClear(HAL_SPI_x spi_x);
00282
00283 uint8_t HAL_DMA_IrqDone(HAL_DMA_x);
00284 uint8_t HAL_DMA_IrqDoneFlg(HAL_DMA_x dma_x);
00285 void HAL_DMA_IrqDoneEnable(HAL_DMA_x);
00286 void HAL_DMA_IrqDoneDisable(HAL_DMA_x dma_x);
00287 void HAL_DMA_IrqDoneClear(HAL_DMA_x dma_x);
00288 void HAL_DMA_IrqDoneTrig(HAL_DMA_x);
00289
00290 uint8_t HAL_DMA_IrqErr(HAL_DMA_x);
```

```
00291 uint8_t HAL_DMA_IrqErrFlg(HAL_DMA_x dma_x);
00292 void HAL_DMA_IrqErrEnable(HAL_DMA_x);
00293 void HAL_DMA_IrqErrDisable(HAL_DMA_x dma_x);
00294 void HAL_DMA_IrqErrClear(HAL_DMA_x dma_x);
00295 void HAL_DMA_IrqErrTrig(HAL_DMA_x);
00296
00297 uint8_t HAL_GPIO_Irq(uint8_t);
00298 uint8_t HAL_GPIO_IrqFlg(uint8_t line);
00299 void HAL_GPIO_IrqEnable(uint8_t);
00300 void HAL_GPIO_IrqDisable(uint8_t line);
00301 void HAL_GPIO_IrqClear(uint8_t line);
00302 void HAL_GPIO_IrqTrig(uint8_t);
00303
00304 // 硬件不支持的中断(空实现)
00305
00306 uint8_t HAL_ADC_IrqIdleSamp(HAL_ADC_x);
00307 uint8_t HAL_ADC_IrqIdleSampFlg(HAL_ADC_x adc);
00308 void HAL_ADC_IrqIdleSampEnable(HAL_ADC_x);
00309 void HAL_ADC_IrqIdleSampDisable(HAL_ADC_x adc);
00310 void HAL_ADC_IrqIdleSampClear(HAL_ADC_x adc);
00311
00312 uint8_t HAL_ADC_IrqWatchdog1(HAL_ADC_x);
00313 uint8_t HAL_ADC_IrqWatchdog1Flg(HAL_ADC_x adc);
00314 void HAL_ADC_IrqWatchdog1Enable(HAL_ADC_x);
00315 void HAL_ADC_IrqWatchdog1Disable(HAL_ADC_x adc);
00316 void HAL_ADC_IrqWatchdog1Clear(HAL_ADC_x adc);
00317
00318 uint8_t HAL_MCPWM_IrqTH40(HAL_MCPWM_x);
00319 uint8_t HAL_MCPWM_IrqTH40Flg(HAL_MCPWM_x mcpwm_x);
00320 void HAL_MCPWM_IrqTH40Enable(HAL_MCPWM_x);
00321 void HAL_MCPWM_IrqTH40Disable(HAL_MCPWM_x mcpwm_x);
00322 void HAL_MCPWM_IrqTH40Clear(HAL_MCPWM_x mcpwm_x);
00323
00324 uint8_t HAL_MCPWM_IrqTH41(HAL_MCPWM_x);
00325 uint8_t HAL_MCPWM_IrqTH41Flg(HAL_MCPWM_x mcpwm_x);
00326 void HAL_MCPWM_IrqTH41Enable(HAL_MCPWM_x);
00327 void HAL_MCPWM_IrqTH41Disable(HAL_MCPWM_x mcpwm_x);
00328 void HAL_MCPWM_IrqTH41Clear(HAL_MCPWM_x mcpwm_x);
00329
00330 uint8_t HAL_MCPWM_IrqTH50(HAL_MCPWM_x);
00331 uint8_t HAL_MCPWM_IrqTH50Flg(HAL_MCPWM_x mcpwm_x);
00332 void HAL_MCPWM_IrqTH50Enable(HAL_MCPWM_x);
00333 void HAL_MCPWM_IrqTH50Disable(HAL_MCPWM_x mcpwm_x);
00334 void HAL_MCPWM_IrqTH50Clear(HAL_MCPWM_x mcpwm_x);
00335
00336 uint8_t HAL_MCPWM_IrqTH51(HAL_MCPWM_x);
00337 uint8_t HAL_MCPWM_IrqTH51Flg(HAL_MCPWM_x mcpwm_x);
00338 void HAL_MCPWM_IrqTH51Enable(HAL_MCPWM_x);
00339 void HAL_MCPWM_IrqTH51Disable(HAL_MCPWM_x mcpwm_x);
00340 void HAL_MCPWM_IrqTH51Clear(HAL_MCPWM_x mcpwm_x);
00341
00342 uint8_t HAL_SW_Irq(uint8_t);
00343 uint8_t HAL_SW_IrqFlg(uint8_t x);
00344 void HAL_SW_IrqEnable(uint8_t);
00345 void HAL_SW_IrqDisable(uint8_t x);
00346 void HAL_SW_IrqClear(uint8_t x);
00347
00348 void HAL_SW_IrqTrig(uint8_t);
00349
00350 uint8_t HAL_UART_IrqAbd(HAL_UART_x);
00351 uint8_t HAL_UART_IrqAbdFlg(HAL_UART_x uart_x);
00352 void HAL_UART_IrqAbdEnable(HAL_UART_x);
00353 void HAL_UART_IrqAbdDisable(HAL_UART_x uart_x);
00354 void HAL_UART_IrqAbdClear(HAL_UART_x uart_x);
00355
00356 uint8_t HAL_UART_IrqIdle(HAL_UART_x);
00357 uint8_t HAL_UART_IrqIdleFlg(HAL_UART_x uart_x);
00358 void HAL_UART_IrqIdleEnable(HAL_UART_x);
00359 void HAL_UART_IrqIdleDisable(HAL_UART_x uart_x);
00360 void HAL_UART_IrqIdleClear(HAL_UART_x uart_x);
00361
00362 uint8_t HAL_UART_IrqLbd(HAL_UART_x);
00363 uint8_t HAL_UART_IrqLbdFlg(HAL_UART_x uart_x);
00364 void HAL_UART_IrqLbdEnable(HAL_UART_x);
00365 void HAL_UART_IrqLbdDisable(HAL_UART_x uart_x);
00366 void HAL_UART_IrqLbdClear(HAL_UART_x uart_x);
00367
00368 uint8_t HAL_UART_IrqRxOv(HAL_UART_x);
00369 uint8_t HAL_UART_IrqRxOvFlg(HAL_UART_x uart_x);
00370 void HAL_UART_IrqRxOvEnable(HAL_UART_x);
00371 void HAL_UART_IrqRxOvDisable(HAL_UART_x uart_x);
00372 void HAL_UART_IrqRxOvClear(HAL_UART_x uart_x);
00373
00374 uint8_t HAL_UART_IrqTxOv(HAL_UART_x);
00375 uint8_t HAL_UART_IrqTxOvFlg(HAL_UART_x uart_x);
00376 void HAL_UART_IrqTxOvEnable(HAL_UART_x);
00377 void HAL_UART_IrqTxOvDisable(HAL_UART_x uart_x);
```

```

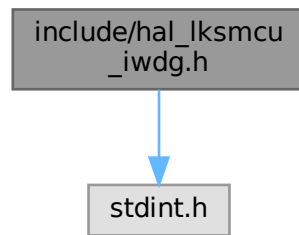
00378 void HAL_UART_IrqTxOvClear(HAL_UART_x uart_x);
00379
00380 #endif

```

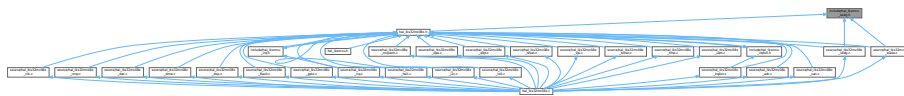
4.39 include/hal_lksmcu_iwdg.h 文件参考

```
#include "stdint.h"
```

hal_lksmcu_iwdg.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



枚举

- enum HAL_IWDG_x { HAL_IWDG_0 }

函数

- void HAL_IWDG_Reset (HAL_IWDG_x)
模块复位
- void HAL_IWDG_Enable (HAL_IWDG_x)
模块打开
- void HAL_IWDG_Disable (HAL_IWDG_x)
模块关闭
- void HAL_IWDG_SetResetTimemS (HAL_IWDG_x, int32_t ms)
设置看门狗超时复位时间
- void HAL_IWDG_Feed (HAL_IWDG_x)
喂狗

4.39.1 枚举类型说明

4.39.1.1 HAL_IWDG_x

```
enum HAL_IWDG_x
```

枚举值

HAL_IWDG_0

在文件 [hal_lksmcu_iwdg.h](#) 第 4 行定义.

4.39.2 函数说明

4.39.2.1 HAL_IWDG_Disable()

```
void HAL_IWDG_Disable (
    HAL_IWDG_x iwdg )
```

模块关闭

参数

<i>iwdg</i>	看门狗模块

在文件 [hal_lks32mc08x_iwdg.c](#) 第 32 行定义.

这是这个函数的调用关系图:



4.39.2.2 HAL_IWDG_Enable()

```
void HAL_IWDG_Enable (
    HAL_IWDG_x iwdg )
```

模块打开

参数

<i>iwdg</i>	看门狗模块

在文件 [hal_lks32mc08x_iwdg.c](#) 第 20 行定义.

4.39.2.3 HAL_IWDG_Feed()

```
void HAL_IWDG_Feed (
    HAL_IWDG_x iwdg )
```

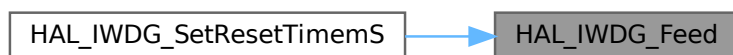
喂狗

参数

<i>iwdg</i>	看门狗模块

在文件 [hal_lks32mc08x.iwdg.c](#) 第 71 行定义.

这是这个函数的调用关系图:



4.39.2.4 HAL_IWDG_Reset()

```
void HAL_IWDG_Reset (
    HAL_IWDG_x iwdg )
```

模块复位

参数

<i>iwdg</i>	看门狗模块

在文件 [hal_lks32mc08x.iwdg.c](#) 第 11 行定义.

函数调用图:



4.39.2.5 HAL_IWDG_SetResetTimemS()

```
void HAL_IWDG_SetResetTimemS (
    HAL_IWDG_x iwdg,
    int32_t ms )
```

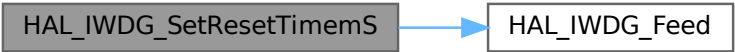
设置看门狗超时复位时间

参数

<i>iwdg</i>	看门狗模块
<i>ms</i>	超时复位时间,单位mS

在文件 [hal_lks32mc08x_iwdg.c](#) 第 45 行定义.

函数调用图:



4.40 hal_lksmcu_iwdg.h

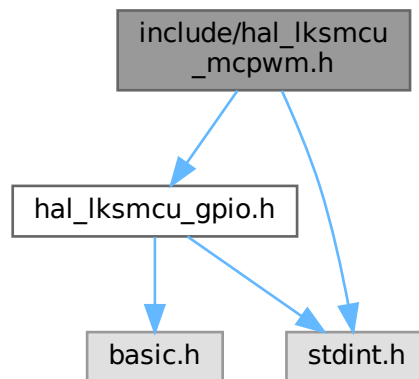
[浏览该文件的文档.](#)

```
00001 #ifndef __HAL_LKSMCU_IWDG_H__
00002 #define __HAL_LKSMCU_IWDG_H__
00003 #include "stdint.h"
00004 typedef enum
00005 {
00006     HAL_IWDG_0,
00007 } HAL_IWDG_x;
00008 // 初始化
00009 void HAL_IWDG_Reset(HAL_IWDG_x); // 模块复位
00010 void HAL_IWDG_Enable(HAL_IWDG_x); // 模块打开
00011 void HAL_IWDG_Disable(HAL_IWDG_x); // 模块关闭
00012 //
00013 void HAL_IWDG_SetResetTimemS(HAL_IWDG_x, int32_t ms); // 设置看门狗超时复位时间,单位mS,实际复位时间向上取整,例如
// 设置为130mS,实际一步为128mS,则按照256mS进行配置
00014 void HAL_IWDG_Feed(HAL_IWDG_x); // 喂狗
00015 #endif
```

4.41 include/hal_lksmcu_mcpwm.h 文件参考

```
#include "hal_lksmcu_gpio.h"
#include "stdint.h"
```

hal_lksmcu_mcpwm.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



枚举

- enum HAL_MCPWM_x { HAL_MCPWM_0 }
MCPWM模块
- enum HAL_MCPWM_PORT_x { HAL_MCPWM_PORT_0, HAL_MCPWM_PORT_1, HAL_MCPWM_PORT_2, HAL_MCPWM_PORT_3 }
输出端口
- enum HAL_MCPWM_SWAP_x { HAL_MCPWM_SWAP_DISABLE = 0, HAL_MCPWM_SWAP_ENABLE }
输出PN交换配置
- enum HAL_MCPWM_POLARITY_x { HAL_MCPWM_POLARITY_PL_NL, HAL_MCPWM_POLARITY_PH_NL, HAL_MCPWM_POLARITY_PL_NH, HAL_MCPWM_POLARITY_PH_NH }
极性
- enum HAL_MCPWM_OUTPUT_STATE_x { HAL_MCPWM_OUTPUT_STATE_PWM, HAL_MCPWM_OUTPUT_STATE_OFF, HAL_MCPWM_OUTPUT_STATE_LOW, HAL_MCPWM_OUTPUT_STATE_HIGH, HAL_MCPWM_OUTPUT_STATE_LOW_PWM, HAL_MCPWM_OUTPUT_STATE_HIGH_PWM }
输出状态
- enum HAL_MCPWM_DEFAULT_OUTPUT_STATE_x { HAL_MCPWM_DEFAULT_OUTPUT_STATE_OFF, HAL_MCPWM_DEFAULT_OUTPUT_STATE_LOW, HAL_MCPWM_DEFAULT_OUTPUT_STATE_HIGH }
默认输出电平（触发FAIL或关闭输出后的状态）
- enum HAL_MCPWM_FAIL_x { HAL_MCPWM_FAIL_0, HAL_MCPWM_FAIL_1 }
MCPWM模块FAIL通道

- enum `HAL_MCPWM_FAIL_IN_x` {
`HAL_MCPWM_FAIL_IN_IO_HIGH` = 0x01 , `HAL_MCPWM_FAIL_IN_IO_LOW` = 0x02 , `HAL_MCPWM_FAIL_IN_CMP0_HIGH` = 0x03 , `HAL_MCPWM_FAIL_IN_CMP0_LOW` = 0x04 ,
`HAL_MCPWM_FAIL_IN_CMP1_HIGH` = 0x03 , `HAL_MCPWM_FAIL_IN_CMP1_LOW` = 0x04 }
 FAIL信号来源
- enum `HAL_MCPWM_UPDATE_x` { `HAL_MCPWM_UPDATE_T0` = 0 , `HAL_MCPWM_UPDATE_T1` = 1 ,
`HAL_MCPWM_UPDATE_T0_AND_T1` = 2 }
 影子寄存器自动更新事件

函数

- void `HAL_MCPWM_Reset` (`HAL_MCPWM_x`)
 复位`mcpwm`模块
- void `HAL_MCPWM_Enable` (`HAL_MCPWM_x`)
`mcpwm`模块使能
- void `HAL_MCPWM_Disable` (`HAL_MCPWM_x`)
`mcpwm`模块关闭
- void `HAL_MCPWM_SetFreq` (`HAL_MCPWM_x`, uint32_t freq)
 设置`pwm`频率
- void `HAL_MCPWM_SetThCnt` (`HAL_MCPWM_x`, uint32_t th)
 设置计数器最大值
- void `HAL_MCPWM_SetUpdateMode` (`HAL_MCPWM_x`, `HAL_MCPWM_UPDATE_x`, uint32_t intv)
 设置影子寄存器自动更新时刻
- uint32_t `HAL_MCPWM_GetFreq` (`HAL_MCPWM_x`)
 获取`pwm`频率
- uint32_t `HAL_MCPWM_GetThCnt` (`HAL_MCPWM_x`)
 获取计数器最大值
- void `HAL_MCPWM_SetClockDiv` (`HAL_MCPWM_x` `mcpwm_x`, uint32_t div)
- uint32_t `HAL_MCPWM_GetClockDiv` (`HAL_MCPWM_x` `mcpwm_x`)
- uint32_t `HAL_MCPWM_GetClockFreq` (`HAL_MCPWM_x` `mcpwm_x`)
- void `HAL_MCPWM_SetDead` (`HAL_MCPWM_x`, uint32_t pdead_nS, uint32_t ndead_nS)
 设置死区
- void `HAL_MCPWM_SetOutputDisable` (`HAL_MCPWM_x`)
 关闭输出
- void `HAL_MCPWM_SetOutputEnable` (`HAL_MCPWM_x`)
 打开输出
- void `HAL_MCPWM_StartCounting` (`HAL_MCPWM_x`)
 开始计数
- void `HAL_MCPWM_StopCounting` (`HAL_MCPWM_x`)
 停止计数(同时计数器回到-th)
- void `HAL_MCPWM_StartCountingSync` (`HAL_MCPWM_x` `mcpwm1_x`, `HAL_MCPWM_x` `mcpwm2_x`, int16_t t0, int16_t t1)
 跟随另一个计数器启动, 周期不同时会自动映射相位差
- void `HAL_MCPWM_SetPwm` (`HAL_MCPWM_x`, `HAL_MCPWM_PORT_x`, int32_t t0, int32_t t1)
 设置占空比
- void `HAL_MCPWM_SetPwmQ15` (`HAL_MCPWM_x`, `HAL_MCPWM_PORT_x`, int16_t t0, int16_t t1)
- void `HAL_MCPWM_SetOutState` (`HAL_MCPWM_x`, `HAL_MCPWM_PORT_x`, `HAL_MCPWM_OUTPUT_STATE_x`)
 设置输出状态
- void `HAL_MCPWM_SetAdcTrig` (`HAL_MCPWM_x`, int32_t t0, int32_t t1)
 设置`pwm`输出ADC采样时间
- void `HAL_MCPWM_SetAdcTrigQ15` (`HAL_MCPWM_x`, int32_t t0, int32_t t1)

- 设置输出ADC采样时间 $q15$ 格式输入
- void `HAL_MCPWM_SetDrivePolarity` (`HAL_MCPWM_x`, `HAL_MCPWM_POLARITY_x`)
设置极性
- void `HAL_MCPWM_SetDrivePolarityPort` (`HAL_MCPWM_x`, `HAL_MCPWM_PORT_x`, `HAL_MCPWM_POLARITY_x`)
设置单个输出通道的极性
- void `HAL_MCPWM_SetSwap` (`HAL_MCPWM_x`, `HAL_MCPWM_SWAP_x`)
设置PN是否交换
- void `HAL_MCPWM_SetSwapPort` (`HAL_MCPWM_x`, `HAL_MCPWM_PORT_x`, `HAL_MCPWM_SWAP_x`)
设置单个输出通道的PN是否交换
- void `HAL_MCPWM_SetRemap` (`HAL_MCPWM_x`, `uint8_t mode`)
MCPWM通道重映射(通道输出到GPIO的路由重排,与`HAL_MCPWM_SetSwap`的PN交换无关)
- void `HAL_MCPWM_SetDefaultState` (`HAL_MCPWM_x`, `HAL_MCPWM_DEFAULT_OUTPUT_STATE_x`)
设置默认状态
- void `HAL_MCPWM_SetGpioOut` (`HAL_MCPWM_x`, `HAL_GPIO_Px`)
设置用于输出的gpio
- void `HAL_MCPWM_SetGpioFail` (`HAL_MCPWM_x`, `HAL_GPIO_Px`)
配置用于故障输入的gpio
- void `HAL_MCPWM_SetFailSignal` (`HAL_MCPWM_x`, `HAL_MCPWM_FAIL_x`, `HAL_MCPWM_FAIL_IN_x`)
设置FAIL信号来源
- void `HAL_MCPWM_SetFailFilter` (`HAL_MCPWM_x`, `uint32_t cnt`)
设置FAIL信号滤波宽度

4.41.1 枚举类型说明

4.41.1.1 HAL_MCPWM_DEFAULT_OUTPUT_STATE_x

enum `HAL_MCPWM_DEFAULT_OUTPUT_STATE_x`

默认输出电平（触发FAIL或关闭输出后的状态）

枚举值

<code>HAL_MCPWM_DEFAULT_OUTPUT_STATE_OFF</code>	关闭输出
<code>HAL_MCPWM_DEFAULT_OUTPUT_STATE_LOW</code>	下管恒开
<code>HAL_MCPWM_DEFAULT_OUTPUT_STATE_HIGH</code>	上管恒开

在文件 `hal_lksmcu_mcpwm.h` 第 64 行定义.

4.41.1.2 HAL_MCPWM_FAIL_IN_x

enum `HAL_MCPWM_FAIL_IN_x`

FAIL信号来源

枚举值

<code>HAL_MCPWM_FAIL_IN_IO_HIGH</code>	外部FAIL信号为高电平后mcpwm关闭输出
<code>HAL_MCPWM_FAIL_IN_IO_LOW</code>	外部FAIL信号为低电平后mcpwm关闭输出

枚举值

HAL_MCPWM_FAIL_IN_CMP0_HIGH	仅FAIL0 CMP0为高电平后mcpwm关闭输出
HAL_MCPWM_FAIL_IN_CMP0_LOW	仅FAIL0 CMP0为低电平后mcpwm关闭输出
HAL_MCPWM_FAIL_IN_CMP1_HIGH	仅FAIL1 CMP1为高电平后mcpwm关闭输出
HAL_MCPWM_FAIL_IN_CMP1_LOW	仅FAIL1 CMP1为低电平后mcpwm关闭输出

在文件 [hal_lksmcu_mcpwm.h](#) 第 87 行定义.

4.41.1.3 HAL_MCPWM_FAIL_x

enum [HAL_MCPWM_FAIL_x](#)

MCPWM模块FAIL通道

枚举值

HAL_MCPWM_↔ FAIL_0	FAIL0
HAL_MCPWM_↔ FAIL_1	FAIL1

在文件 [hal_lksmcu_mcpwm.h](#) 第 74 行定义.

4.41.1.4 HAL_MCPWM_OUTPUT_STATE_x

enum [HAL_MCPWM_OUTPUT_STATE_x](#)

输出状态

枚举值

HAL_MCPWM_OUTPUT_STATE_PWM	上下管对称pwm
HAL_MCPWM_OUTPUT_STATE_OFF	关闭输出
HAL_MCPWM_OUTPUT_STATE_LOW	下管恒开
HAL_MCPWM_OUTPUT_STATE_HIGH	上管恒开
HAL_MCPWM_OUTPUT_STATE_LOW_PWM	下管pwm
HAL_MCPWM_OUTPUT_STATE_HIGH_PWM	上管pwm

在文件 [hal_lksmcu_mcpwm.h](#) 第 52 行定义.

4.41.1.5 HAL_MCPWM_POLARITY_x

enum [HAL_MCPWM_POLARITY_x](#)

极性

枚举值

HAL_MCPWM_POLARITY_PL_NL	上管低有效,下管低有效
HAL_MCPWM_POLARITY_PH_NL	上管高有效,下管低有效
HAL_MCPWM_POLARITY_PL_NH	上管低有效,下管高有效
HAL_MCPWM_POLARITY_PH_NH	上管高有效,下管高有效

在文件 [hal_lksmcu_mcpwm.h](#) 第 42 行定义.

4.41.1.6 HAL_MCPWM_PORT_x

enum [HAL_MCPWM_PORT_x](#)

输出端口

枚举值

HAL_MCPWM_↔ PORT_0	输出端口0
HAL_MCPWM_↔ PORT_1	输出端口1
HAL_MCPWM_↔ PORT_2	输出端口2
HAL_MCPWM_↔ PORT_3	输出端口3

在文件 [hal_lksmcu_mcpwm.h](#) 第 20 行定义.

4.41.1.7 HAL_MCPWM_SWAP_x

enum [HAL_MCPWM_SWAP_x](#)

输出PN交换配置

枚举值

HAL_MCPWM_SWAP_DISABLE	取消交换PN通道
HAL_MCPWM_SWAP_ENABLE	交换PN通道

在文件 [hal_lksmcu_mcpwm.h](#) 第 34 行定义.

4.41.1.8 HAL_MCPWM_UPDATE_x

enum [HAL_MCPWM_UPDATE_x](#)

影子寄存器自动更新事件

枚举值

HAL_MCPWM_UPDATE_T0	起点更新(默认)
HAL_MCPWM_UPDATE_T1	过零更新
HAL_MCPWM_UPDATE_T0_AND_T1	起点和过零都更新

在文件 [hal_lksmcu_mcpwm.h](#) 第 114 行定义.

4.41.1.9 HAL_MCPWM_x

enum [HAL_MCPWM_x](#)

MCPWM模块

枚举值

HAL_↔ MCPWM_0	MCPWM模块0

在文件 [hal_lksmcu_mcpwm.h](#) 第 8 行定义.

4.41.2 函数说明

4.41.2.1 HAL_MCPWM_Disable()

```
void HAL_MCPWM_Disable (
    HAL\_MCPWM\_x mcpwm_x )
```

mcpwm模块关闭

参数

<i>mcpwm</i> ↔ _x	操作的模块编号

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 197 行定义.

4.41.2.2 HAL_MCPWM_Enable()

```
void HAL_MCPWM_Enable (
    HAL\_MCPWM\_x mcpwm_x )
```

mcpwm模块使能

参数

<i>mcpwm</i> ↔ _x	操作的模块编号

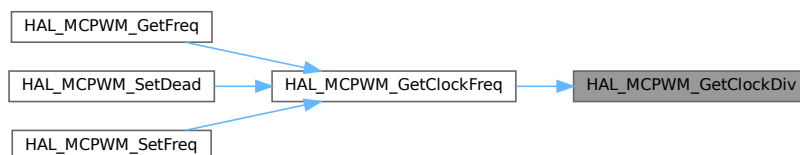
在文件 [hal_lks32mc08x.mcpwm.c](#) 第 178 行定义.

4.41.2.3 HAL_MCPWM_GetClockDiv()

```
uint32_t HAL_MCPWM_GetClockDiv (
    HAL_MCPWM_x mcpwm_x )
```

在文件 [hal_lks32mc08x.mcpwm.c](#) 第 278 行定义.

这是这个函数的调用关系图:

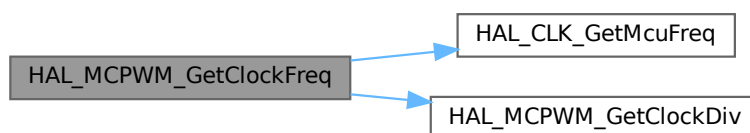


4.41.2.4 HAL_MCPWM_GetClockFreq()

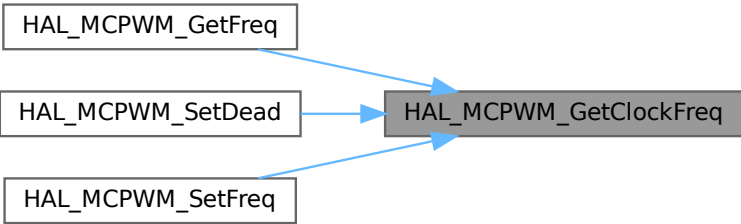
```
uint32_t HAL_MCPWM_GetClockFreq (
    HAL_MCPWM_x mcpwm_x )
```

在文件 [hal_lks32mc08x.mcpwm.c](#) 第 284 行定义.

函数调用图:



这是这个函数的调用关系图:



4.41.2.5 HAL_MCPWM_GetFreq()

```
uint32_t HAL_MCPWM_GetFreq (
    HAL_MCPWM_TypeDef * mcpwm_x )
```

获取pwm频率

参数

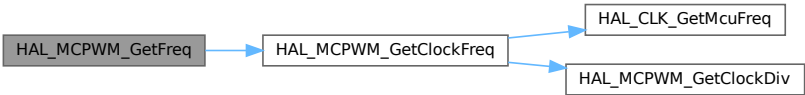
<i>mcpwm_x</i>	操作的模块编号

返回

当前pwm的频率

在文件 `hal_lks32mc08x_mcpwm.c` 第 255 行定义.

函数调用图:



4.41.2.6 HAL_MCPWM_GetThCnt()

```
uint32_t HAL_MCPWM_GetThCnt (
    HAL_MCPWM_TypeDef * mcpwm_x )
```

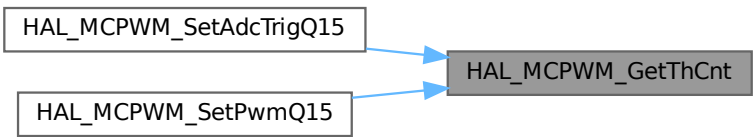
获取计数器最大值

参数

<i>mcpwm</i> ↔ _x	操作的模块编号

在文件 [hal_lks32mc08x.mcpwm.c](#) 第 293 行定义.

这是这个函数的调用关系图:



4.41.2.7 HAL_MCPWM_Reset()

```
void HALMCPWM_Reset (
    HAL_MCPWM_x mcpwm_x )
```

复位mcpwm模块

参数

<i>mcpwm</i> ↔ _x	操作的模块编号

在文件 [hal_lks32mc08x.mcpwm.c](#) 第 163 行定义.

4.41.2.8 HAL_MCPWM_SetAdcTrig()

```
void HALMCPWM_SetAdcTrig (
    HAL_MCPWM_x mcpwm_x,
    int32_t t0,
    int32_t t1 )
```

设置pwm输出ADC采样时间

参数

<i>mcpwm</i> ↔ _x	操作的模块编号
t0	采样时间
t1	采样时间

在文件 `hal_lks32mc08x.mcpwm.c` 第 578 行定义.

这是这个函数的调用关系图:



4.41.2.9 HAL_MCPWM_SetAdcTrigQ15()

```
void HAL_MCPWM_SetAdcTrigQ15 (
    HAL_MCPWM_x mcpwm_x,
    int32_t t0,
    int32_t t1 )
```

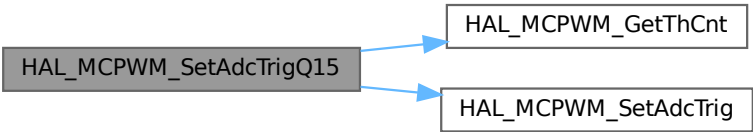
设置pwm输出ADC采样时间q15格式输入

参数

<i>mcpwm_x</i>	操作的模块编号
<i>t0</i>	采样时间q15格式输入
<i>t1</i>	采样时间q15格式输入

在文件 `hal_lks32mc08x.mcpwm.c` 第 589 行定义.

函数调用图:



4.41.2.10 HAL_MCPWM_SetClockDiv()

```
void HAL_MCPWM_SetClockDiv (
    HAL_MCPWM_x mcpwm_x,
    uint32_t div )
```

在文件 `hal_lks32mc08x.mcpwm.c` 第 265 行定义.

4.41.2.11 HAL_MCPWM_SetDead()

```
void HAL_MCPWM_SetDead (
    HAL_MCPWM_TypeDef* mcpwm_x,
    uint32_t pdead_ns,
    uint32_t ndead_ns )
```

设置死区

参数

<i>mcpwm_x</i>	操作的模块编号
<i>pdead_ns</i>	下管关闭上管打开的死区宽度,单位nS
<i>ndead_ns</i>	上管关闭下管打开的死区宽度,单位nS

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 304 行定义.

函数调用图:



4.41.2.12 HAL_MCPWM_SetDefaultState()

```
void HAL_MCPWM_SetDefaultState (
    HAL_MCPWM_TypeDef* mcpwm_x,
    HAL_MCPWM_DEFAULT_OUTPUT_STATE_TypeDef* mcpwm_default_output_state_x )
```

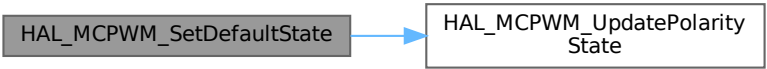
设置默认状态

参数

<i>mcpwm_x</i>	操作的模块编号
<i>mcpwm_default_output_state_x</i>	默认输出状态

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 478 行定义.

函数调用图:



4.41.2.13 HAL_MCPWM_SetDrivePolarity()

```
void HAL_MCPWM_SetDrivePolarity (
    HAL_MCPWM_x mcpwm_x,
    HAL_MCPWM_POLARITY_x mcpwm_polarity_x )
```

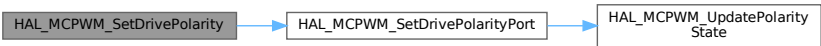
设置pwm极性

参数

<i>mcpwm_x</i>	操作的模块编号
<i>mcpwm_x</i> ↔ <i>polarity_x</i>	第x个pwm输出端口

在文件 [hal_lks32mc08x.mcpwm.c](#) 第 418 行定义.

函数调用图:



4.41.2.14 HAL_MCPWM_SetDrivePolarityPort()

```
void HAL_MCPWM_SetDrivePolarityPort (
    HAL_MCPWM_x mcpwm_x,
    HAL_MCPWM_PORT_x mcpwm_port_x,
    HAL_MCPWM_POLARITY_x mcpwm_polarity_x )
```

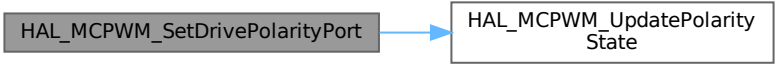
设置单个输出通道的pwm极性

参数

<i>mcpwm_x</i>	操作的模块编号
<i>mcpwm_port_x</i>	第x个pwm输出端口
<i>mcpwm_x</i> ↔ <i>polarity_x</i>	输出极性

在文件 `hal_lks32mc08x_mcpwm.c` 第 431 行定义.

函数调用图:



这是这个函数的调用关系图:



4.41.2.15 HAL.MCPWM.SetFailFilter()

```
void HALMCPWM.SetFailFilter (
    HALMCPWM_x mcpwm_x,
    uint32_t cnt )
```

设置FAIL信号滤波宽度

参数

<i>mcpwm_x</i>	操作的模块编号
<i>cnt</i>	滤波宽度

在文件 `hal_lks32mc08x_mcpwm.c` 第 602 行定义.

4.41.2.16 HAL.MCPWM.SetFailSignal()

```
void HALMCPWM.SetFailSignal (
    HALMCPWM_x mcpwm_x,
    HALMCPWM_FAIL_x fail_x,
    HALMCPWM_FAIL_IN_x fail_in_x )
```

设置FAIL信号来源

参数

<i>mcpwm</i> _{<i>_X</i>}	操作的模块编号
<i>fail</i> _{<i>_X</i>}	FAIL信号来源

< 外部FAIL信号为高电平后mcpwm关闭输出

< 外部FAIL信号为低电平后mcpwm关闭输出

< CMP0为高电平后mcpwm关闭输出

< CMP0为低电平后mcpwm关闭输出

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 536 行定义.

4.41.2.17 HAL_MCPWM_SetFreq()

```
void HAL_MCPWM_SetFreq (
    HAL_MCPWM_X mcpwm_X,
    uint32_t freq )
```

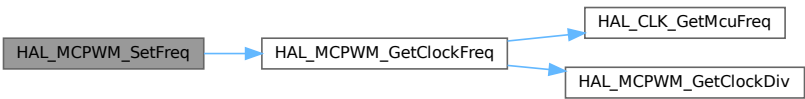
设置pwm频率

参数

<i>mcpwm</i> _{<i>_X</i>}	操作的模块编号

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 228 行定义.

函数调用图:



4.41.2.18 HAL_MCPWM_SetGpioFail()

```
void HAL_MCPWM_SetGpioFail (
    HAL_MCPWM_X mcpwm_X,
    HAL_GPIO_X px )
```

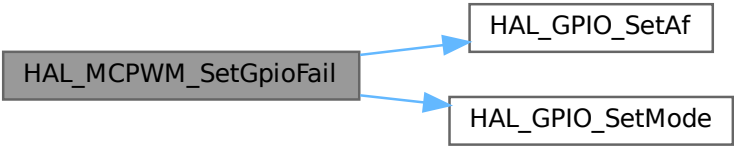
配置用于故障输入的gpio

参数

<i>mcpwm</i> _{<i>_x</i>}	MCPWM模块编号
<i>px</i>	使用的GPIO口

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 524 行定义.

函数调用图:



4.41.2.19 HAL_MCPWM_SetGpioOut()

```
void HAL_MCPWM_SetGpioOut (
    HAL_MCPWM_ x,
    HAL_GPIO_Px px )
```

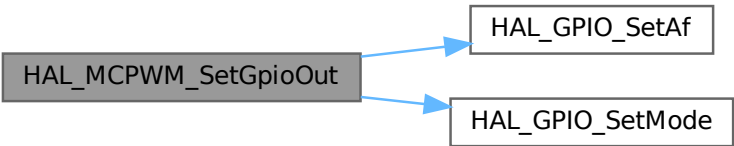
设置用于输出的gpio

参数

<i>mcpwm</i> _{<i>_x</i>}	MCPWM模块编号
<i>px</i>	使用的GPIO口

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 513 行定义.

函数调用图:



4.41.2.20 HAL_MCPWM_SetOutputDisable()

```
void HAL_MCPWM_SetOutputDisable (
    HAL_MCPWM_x mcpwm_x )
```

关闭输出

参数

<i>mcpwm_x</i>	操作的模块编号

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 325 行定义.

4.41.2.21 HAL_MCPWM_SetOutputEnable()

```
void HAL_MCPWM_SetOutputEnable (
    HAL_MCPWM_x mcpwm_x )
```

打开输出

参数

<i>mcpwm_x</i>	操作的模块编号

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 336 行定义.

4.41.2.22 HAL_MCPWM_SetOutState()

```
void HAL_MCPWM_SetOutState (
    HAL_MCPWM_x mcpwm_x,
    HAL_MCPWM_PORT_x mcpwm_port_x,
    HAL_MCPWM_OUTPUT_STATE_x mcpwm_output_state_x )
```

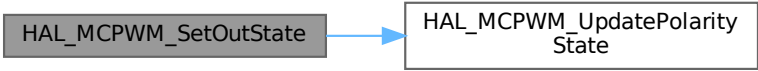
设置输出状态

参数

<i>mcpwm_x</i>	操作的模块编号
<i>mcpwm_port_x</i>	第x个pwm输出端口
<i>mcpwm_output_state_x</i>	输出状态

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 406 行定义.

函数调用图:



4.41.2.23 HAL.MCPWM.SetPwm()

```
void HAL_MCPWM_SetPwm (
    HAL_MCPWM_x mcpwm_x,
    HAL_MCPWM_PORT_x mcpwm_port_x,
    int32_t t0,
    int32_t t1 )
```

设置占空比

参数

<i>mcpwm_x</i>	操作的模块编号
<i>mcpwm_port_x</i>	第x个pwm输出端口
<i>t0</i>	pwm打开时刻
<i>t1</i>	pwm关闭时刻

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 386 行定义.

这是这个函数的调用关系图:

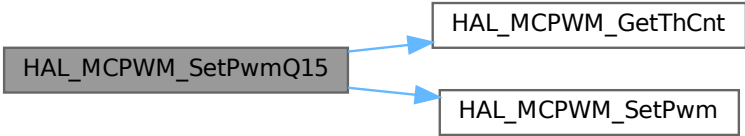


4.41.2.24 HAL.MCPWM.SetPwmQ15()

```
void HAL_MCPWM_SetPwmQ15 (
    HAL_MCPWM_x mcpwm_x,
    HAL_MCPWM_PORT_x mcpwm_port_x,
    int16_t t0,
    int16_t t1 )
```

在文件 `hal_lks32mc08x_mcpwm.c` 第 392 行定义.

函数调用图:



4.41.2.25 HAL_MCPWM_SetRemap()

```
void HALMCPWM_SetRemap (
    HAL_MCPWM_x mcpwm_x,
    uint8_t mode )
```

MCPWM通道重映射(通道输出到GPIO的路由重排,与HAL_MCPWM_SetSwap的PN交换无关)

参数

<i>mcpwm_x</i>	操作的模块编号
<i>mode</i>	0:片外预驱(默认排布) 1:封装排布1 2:封装排布2

注解

本系列无通道重映射功能,本函数为空操作

在文件 `hal_lks32mc08x_mcpwm.c` 第 467 行定义.

4.41.2.26 HAL_MCPWM_SetSwap()

```
void HALMCPWM_SetSwap (
    HAL_MCPWM_x mcpwm_x,
    HAL_MCPWM_SWAP_x mcpwm_swap_x )
```

设置PN是否交换

参数

<i>mcpwm_x</i>	操作的模块编号
<i>mcpwm_swap_x</i>	PN是否交换

在文件 `hal_lks32mc08x_mcpwm.c` 第 442 行定义.

函数调用图:



4.41.2.27 HAL.MCPWM.SetSwapPort()

```
void HALMCPWM.SetSwapPort (
    HAL_MCPWM_x mcpwm_x,
    HAL_MCPWM_PORT_x mcpwm_port_x,
    HAL_MCPWM_SWAP_x mcpwm_swap_x )
```

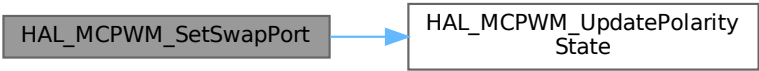
设置单个输出通道的PN是否交换

参数

<i>mcpwm_x</i>	操作的模块编号
<i>mcpwm_port_x</i>	第x个pwm输出端口
<i>mcpwm_swap_x</i>	PN是否交换

在文件 `hal_lks32mc08x_mcpwm.c` 第 455 行定义.

函数调用图:



这是这个函数的调用关系图:



4.41.2.28 HAL.MCPWM.SetThCnt()

```
void HAL_MCPWM_SetThCnt (
    HAL_MCPWM_x mcpwm_x,
    uint32_t th )
```

设置计数器最大值

参数

<i>mcpwm_x</i>	操作的模块编号
<i>th</i>	最大值

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 215 行定义.

4.41.2.29 HAL.MCPWM.SetUpdateMode()

```
void HAL_MCPWM_SetUpdateMode (
    HAL_MCPWM_x mcpwm_x,
    HAL_MCPWM_UPDATE_x update_x,
    uint32_t intv )
```

设置影子寄存器自动更新时刻

参数

<i>mcpwm_x</i>	操作的模块编号
<i>update_x</i>	更新事件选择
<i>intv</i>	更新间隔0~15,每intv+1次更新事件加载一次,0表示每次事件都更新

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 617 行定义.

4.41.2.30 HAL.MCPWM.StartCounting()

```
void HAL_MCPWM_StartCounting (
    HAL_MCPWM_x mcpwm_x )
```

开始计数

参数

<i>mcpwm_x</i>	操作的模块编号

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 347 行定义.

这是这个函数的调用关系图:



4.41.2.31 HAL_MCPWM_StartCountingSync()

```
void HALMCPWM.StartCountingSync (
    HAL_MCPWM_x mcpwm1_x,
    HAL_MCPWM_x mcpwm2_x,
    int16_t phase )
```

跟随另一个计数器启动，周期不同时会自动映射相位差

参数

<i>mcpwm_x</i>	操作的模块编号
<i>mcpwm_x_followed</i>	被跟随的模块编号
<i>phase</i>	相位差，0表示同相位，-32768或32767对应正交（执行时所有mcpwm都会暂停）

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 371 行定义.

函数调用图:



4.41.2.32 HAL_MCPWM_StopCounting()

```
void HALMCPWM.StopCounting (
    HAL_MCPWM_x mcpwm_x )
```

停止计数(同时计数器回到-th)

参数

<i>mcpwm_x</i>	操作的模块编号

在文件 `hal_lks32mc08x_mcpwm.c` 第 358 行定义.

4.42 hal_lksmcu_mcpwm.h

[浏览该文件的文档.](#)

```

00001 #ifndef __HAL_LKSMCU_MCPWM_H__
00002 #define __HAL_LKSMCU_MCPWM_H__
00003 #include "hal_lksmcu_gpio.h"
00004 #include "stdint.h"
00008 typedef enum
00009 {
00010     HAL_MCPWM_0,
00011 #ifdef HAL_TEST_NOT_SUPPORTED
00012     HAL_MCPWM_0_1,
00013     HAL_MCPWM_1,
00014     HAL_MCPWM_1_1,
00015 #endif
00016 } HAL_MCPWM_x;
00020 typedef enum
00021 {
00022     HAL_MCPWM_PORT_0,
00023     HAL_MCPWM_PORT_1,
00024     HAL_MCPWM_PORT_2,
00025     HAL_MCPWM_PORT_3,
00026 #ifdef HAL_TEST_NOT_SUPPORTED
00027     HAL_MCPWM_PORT_4,
00028     HAL_MCPWM_PORT_5,
00029 #endif
00030 } HAL_MCPWM_PORT_x;
00034 typedef enum
00035 {
00036     HAL_MCPWM_SWAP_DISABLE = 0,
00037     HAL_MCPWM_SWAP_ENABLE,
00038 } HAL_MCPWM_SWAP_x;
00042 typedef enum
00043 {
00044     HAL_MCPWM_POLARITY_PL_NL,
00045     HAL_MCPWM_POLARITY_PH_NL,
00046     HAL_MCPWM_POLARITY_PL_NH,
00047     HAL_MCPWM_POLARITY_PH_NH,
00048 } HAL_MCPWM_POLARITY_x;
00052 typedef enum
00053 {
00054     HAL_MCPWM_OUTPUT_STATE_PWM,
00055     HAL_MCPWM_OUTPUT_STATE_OFF,
00056     HAL_MCPWM_OUTPUT_STATE_LOW,
00057     HAL_MCPWM_OUTPUT_STATE_HIGH,
00058     HAL_MCPWM_OUTPUT_STATE_LOW_PWM,
00059     HAL_MCPWM_OUTPUT_STATE_HIGH_PWM,
00060 } HAL_MCPWM_OUTPUT_STATE_x;
00064 typedef enum
00065 {
00066     HAL_MCPWM_DEFAULT_OUTPUT_STATE_OFF,
00067     HAL_MCPWM_DEFAULT_OUTPUT_STATE_LOW,
00068     HAL_MCPWM_DEFAULT_OUTPUT_STATE_HIGH,
00069 } HAL_MCPWM_DEFAULT_OUTPUT_STATE_x;
00070
00074 typedef enum
00075 {
00076     HAL_MCPWM_FAIL_0,
00077     HAL_MCPWM_FAIL_1,
00078 #ifdef HAL_TEST_NOT_SUPPORTED
00079     HAL_MCPWM_FAIL_2,
00080     HAL_MCPWM_FAIL_3,
00081 #endif
00082 } HAL_MCPWM_FAIL_x;
00083
00087 typedef enum
00088 {
00089     HAL_MCPWM_FAIL_IN_IO_HIGH = 0x01,
00090     HAL_MCPWM_FAIL_IN_IO_LOW = 0x02,
00091     HAL_MCPWM_FAIL_IN_CMP0_HIGH = 0x03,
00092     HAL_MCPWM_FAIL_IN_CMP0_LOW = 0x04,
00093     HAL_MCPWM_FAIL_IN_CMP1_HIGH = 0x03,
00094     HAL_MCPWM_FAIL_IN_CMP1_LOW = 0x04,
00095 #ifdef HAL_TEST_NOT_SUPPORTED
00096     HAL_MCPWM_FAIL_IN_CLU0_HIGH,
00097     HAL_MCPWM_FAIL_IN_CLU0_LOW,
00098     HAL_MCPWM_FAIL_IN_CLU1_HIGH,
00099     HAL_MCPWM_FAIL_IN_CLU1_LOW,
00100     HAL_MCPWM_FAIL_IN_CLU2_HIGH,

```

```

00101     HAL_MCPWM_FAIL_IN_CLU2_LOW,
00102     HAL_MCPWM_FAIL_IN_CLU3_HIGH,
00103     HAL_MCPWM_FAIL_IN_CLU3_LOW,
00104     HAL_MCPWM_FAIL_IN_CMP2_HIGH,
00105     HAL_MCPWM_FAIL_IN_CMP2_LOW,
00106     HAL_MCPWM_FAIL_IN_CMP3_HIGH,
00107     HAL_MCPWM_FAIL_IN_CMP3_LOW,
00108 #endif
00109 } HAL_MCPWM_FAIL_IN_x;
00110
00111 typedef enum
00112 {
00113     HAL_MCPWM_UPDATE_T0      = 0,
00114     HAL_MCPWM_UPDATE_T1      = 1,
00115     HAL_MCPWM_UPDATE_T0_AND_T1 = 2,
00116 } HAL_MCPWM_UPDATE_x;
00117
00118 // 初始化
00119 void HAL_MCPWM_Reset(HAL_MCPWM_x); // 模块复位
00120 void HAL_MCPWM_Enable(HAL_MCPWM_x); // 模块打开
00121 void HAL_MCPWM_Disable(HAL_MCPWM_x); // 模块关闭
00122
00123 void HAL_MCPWM_SetFreq(HAL_MCPWM_x, uint32_t freq); // 设置pwm频率
00124 void HAL_MCPWM_SetThCnt(HAL_MCPWM_x, uint32_t th); // 设置计数器最大值
00125 void HAL_MCPWM_SetUpdateMode(HAL_MCPWM_x, HAL_MCPWM_UPDATE_x, uint32_t intv); // 设置影子寄存器自动更新时刻
00126 uint32_t HAL_MCPWM_GetFreq(HAL_MCPWM_x); // 获取当前的pwm频率
00127 uint32_t HAL_MCPWM_GetThCnt(HAL_MCPWM_x); // 获取计数器最大值
00128 void HAL_MCPWM_SetClockDiv(HAL_MCPWM_x mcpwm_x, uint32_t div); // 设置时钟分频 div=1/2/4/8
00129 uint32_t HAL_MCPWM_GetClockDiv(HAL_MCPWM_x mcpwm_x); // 获取时钟分频实际除数
00130 uint32_t HAL_MCPWM_GetClockFreq(HAL_MCPWM_x mcpwm_x); // 获取分频后工作时钟 (Hz)
00131 void HAL_MCPWM_SetDead(HAL_MCPWM_x, uint32_t pdead_nS, uint32_t ndead_nS); // 设置死区
00132
00133 void HAL_MCPWM_SetOutputDisable(HAL_MCPWM_x); // 关闭输出
00134 void HAL_MCPWM_SetOutputEnable(HAL_MCPWM_x); // 打开输出
00135 void HAL_MCPWM_StartCounting(HAL_MCPWM_x); // 开始计数
00136 void HAL_MCPWM_StopCounting(HAL_MCPWM_x); // 停止计数
00137 // 数(同时计数器回到-th)
00138 void HAL_MCPWM_StartCountingSync(HAL_MCPWM_x mcpwm1_x, HAL_MCPWM_x mcpwm2_x, int16_t phase); // 跟随另一个cnt启动, 需要先启动mcpwm1_x
00139 void HAL_MCPWM_SetPwm(HAL_MCPWM_x, HAL_MCPWM_PORT_x, int32_t t0, int32_t t1); // 设置pwm波
00140 // 形q15格式输入
00141 void HAL_MCPWM_SetPwmQ15(HAL_MCPWM_x, HAL_MCPWM_PORT_x, int16_t t0, int16_t t1); // 设置输出状态
00142 void HAL_MCPWM_SetOutState(HAL_MCPWM_x, HAL_MCPWM_PORT_x, HAL_MCPWM_OUTPUT_STATE_x); // 设置pwm输出
00143 void HAL_MCPWM_SetAdcTrig(HAL_MCPWM_x, int32_t t0, int32_t t1); // 设置pwm输出ADC采样时间
00144 void HAL_MCPWM_SetAdcTrigQ15(HAL_MCPWM_x, int32_t t0, int32_t t1); // 设置pwm输出ADC采样时间q15格式输入
00145
00146 void HAL_MCPWM_SetDrivePolarity(HAL_MCPWM_x, HAL_MCPWM_POLARITY_x); // 设置pwm极性
00147 void HAL_MCPWM_SetDrivePolarityPort(HAL_MCPWM_x, HAL_MCPWM_PORT_x, HAL_MCPWM_POLARITY_x); // 设置单个输出通道的pwm极性
00148 void HAL_MCPWM_SetSwap(HAL_MCPWM_x, HAL_MCPWM_SWAP_x); // 设置PN是否交换
00149 void HAL_MCPWM_SetSwapPort(HAL_MCPWM_x, HAL_MCPWM_PORT_x, HAL_MCPWM_SWAP_x); // 设置单个输出通道的PN是否交换
00150
00151 void HAL_MCPWM_SetRemap(HAL_MCPWM_x, uint8_t mode); // 设置默认电平
00152 void HAL_MCPWM_SetDefaultState(HAL_MCPWM_x, HAL_MCPWM_DEFAULT_OUTPUT_STATE_x); // 设置用于输出的gpio
00153 void HAL_MCPWM_SetGpioOut(HAL_MCPWM_x, HAL_GPIO_Px); // 设置用于故障输入的gpio
00154 void HAL_MCPWM_SetGpioFail(HAL_MCPWM_x, HAL_GPIO_Px); // 设置FAIL信号来源
00155 void HAL_MCPWM_SetFailSignal(HAL_MCPWM_x, HAL_MCPWM_FAIL_x, HAL_MCPWM_FAIL_IN_x); // 设置FAIL信号滤波宽度
00156 void HAL_MCPWM_SetFailFilter(HAL_MCPWM_x, uint32_t cnt); // 设置FAIL信号滤波宽度
00157
00158 #endif

```

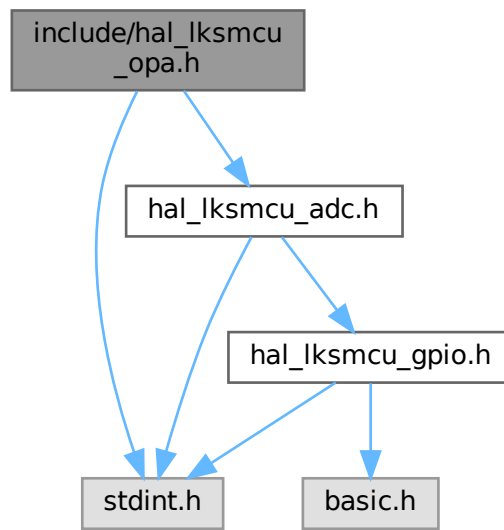
4.43 include/hal_lksmcu_opa.h 文件参考

```

#include "stdint.h"
#include "hal_lksmcu_adc.h"

```

hal_lksmcu_opa.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



宏定义

- #define HAL_OPA_OPAX_2_ADCCHN(OPAX) ((OPAX) == HAL_OPA_0B ? HAL_ADC_CHANNEL_0 ↔ : (HAL_ADC_CHANNEL_x)(OPAX))
- #define HAL_OPA_READ_GAIN(GAIN_x, res)
芯片的运放放大倍数

枚举

- enum HAL_OPA_x {
HAL_OPA_0, HAL_OPA_1, HAL_OPA_2, HAL_OPA_3,
HAL_OPA_4, HAL_OPA_5, HAL_OPA_0B }
- enum HAL_OPA_GAIN_x {
HAL_OPA_GAIN_200_10_4 = 0, HAL_OPA_GAIN_190_20_4 = 1, HAL_OPA_GAIN_180_30_4 = 2,
HAL_OPA_GAIN_170_40_4 = 3,
HAL_OPA_GAIN_HIGH = HAL_OPA_GAIN_200_10_4, HAL_OPA_GAIN_MEDIUM = HAL_OPA_GAIN_190_20_4,
HAL_OPA_GAIN_LOW = HAL_OPA_GAIN_180_30_4 }

函数

- void [HAL_OPA_Reset](#) ([HAL_OPA_x](#))
- void [HAL_OPA_Enable](#) ([HAL_OPA_x](#))
- void [HAL_OPA_Disable](#) ([HAL_OPA_x](#))
- void [HAL_OPA_SetGain](#) ([HAL_OPA_x](#), [HAL_OPA_GAIN_x](#))
- void [HAL_OPA_Out2GpioEnable](#) ([HAL_OPA_x](#))
OPA输出到IO
- void [HAL_OPA_Out2GpioDisable](#) ([HAL_OPA_x](#))
OPA停止输出到IO
- void [HAL_OPA_SetGpioIn](#) ([HAL_OPA_x](#), [HAL_GPIO_Px](#))
OPA设置IO输入
- void [HAL_OPA_SetGpioOut](#) ([HAL_OPA_x](#), [HAL_GPIO_Px](#))
OPA设置IO输出

4.43.1 宏定义说明

4.43.1.1 HAL_OPA_OPAX_2_ADCCHN

```
#define HAL_OPA_OPAX_2_ADCCHN(  
    OPAX ) ((OPAX) == HAL_OPA_0B ? HAL_ADC_CHANNEL_0 : (HAL_ADC_CHANNEL_x)(OPAX))
```

在文件 [hal_lksmcu_opa.h](#) 第 62 行定义.

4.43.1.2 HAL_OPA_READ_GAIN

```
#define HAL_OPA_READ_GAIN(  
    GAIN_x,  
    res )
```

值:

```
== HAL_OPA_GAIN_HIGH ? (200.0f / (10.4f + (float)res)) : (GAIN_x == HAL_OPA_GAIN_MEDIUM) ? (190.0f /  
(20.4f + (float)res)) \  
(180.0f / (30.4f + (float)res))
```

芯片的运放放大倍数

在文件 [hal_lksmcu_opa.h](#) 第 66 行定义.

4.43.2 枚举类型说明

4.43.2.1 HAL_OPA_GAIN_x

```
enum HAL_OPA_GAIN_x
```

枚举值

HAL_OPA_GAIN_200_↔ 10_4	
HAL_OPA_GAIN_190_↔	
 南京凌鸥创芯电子有限公司 LINKO SEMICONDUCTOR CO., LTD. HAL_OPA_GAIN_180_↔ 30_4	
HAL_OPA_GAIN_170_↔ 40_4	

在文件 [hal_lksmcu_opa.h](#) 第 16 行定义.

4.43.2.2 HAL_OPA_x

enum [HAL_OPA_x](#)

枚举值

HAL_OPA_0	ADC模块0
HAL_OPA_1	ADC模块1
HAL_OPA_2	ADC模块2
HAL_OPA_3	ADC模块3
HAL_OPA_4	ADC模块4
HAL_OPA_5	ADC模块5
HAL_OPA_0B	ADC模块0B

在文件 [hal_lksmcu_opa.h](#) 第 5 行定义.

4.43.3 函数说明

4.43.3.1 HAL_OPA_Disable()

```
void HAL_OPA_Disable (
    HAL\_OPA\_x opa_x )
```

在文件 [hal_lks32mc08x_opa.c](#) 第 37 行定义.

4.43.3.2 HAL_OPA_Enable()

```
void HAL_OPA_Enable (
    HAL\_OPA\_x opa_x )
```

在文件 [hal_lks32mc08x_opa.c](#) 第 12 行定义.

4.43.3.3 HAL_OPA_Out2GpioDisable()

```
void HAL_OPA_Out2GpioDisable (
    HAL\_OPA\_x opa_x )
```

OPA停止输出到IO

参数

opa_x	运放序号

在文件 [hal_lks32mc08x_opa.c](#) 第 119 行定义.

4.43.3.4 HAL_OPA_Out2GpioEnable()

```
void HAL_OPA_Out2GpioEnable (
    HAL_OPA_x opa_x )
```

OPA输出到IO

参数

opa_x	运放序号

在文件 [hal_lks32mc08x_opa.c](#) 第 91 行定义.

4.43.3.5 HAL_OPA_Reset()

```
void HAL_OPA_Reset (
    HAL_OPA_x opa_x )
```

在文件 [hal_lks32mc08x_opa.c](#) 第 7 行定义.

4.43.3.6 HAL_OPA_SetGain()

```
void HAL_OPA_SetGain (
    HAL_OPA_x opa_x,
    HAL_OPA_GAIN_x gain_x )
```

在文件 [hal_lks32mc08x_opa.c](#) 第 62 行定义.

4.43.3.7 HAL_OPA_SetGpioIn()

```
void HAL_OPA_SetGpioIn (
    HAL_OPA_x opa_x,
    HAL_GPIO_Px px )
```

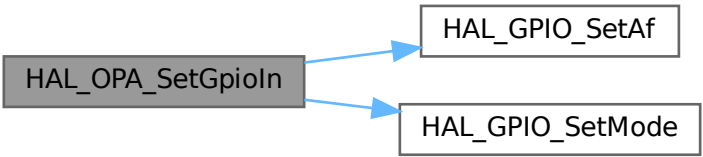
OPA设置IO输入

参数

opa_x	运放序号
px	IO序号

在文件 [hal_lks32mc08x_opa.c](#) 第 133 行定义.

函数调用图:



4.43.3.8 HAL_OPA_SetGpioOut()

```
void HAL_OPA_SetGpioOut (
    HAL_OPA_x opa_x,
    HAL_GPIO_Px px )
```

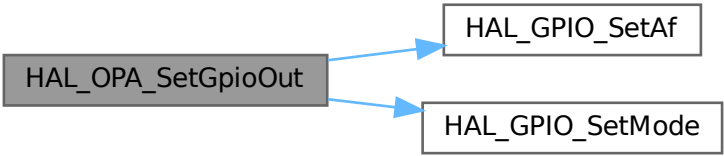
OPA设置IO输出

参数

<i>opa_x</i>	运放序号
<i>px</i>	IO序号

在文件 [hal_lks32mc08x_opa.c](#) 第 145 行定义.

函数调用图:



4.44 hal_lksmcu_opa.h

[浏览该文件的文档.](#)

```
00001 #ifndef __HAL_LKSMCU_OPA_H__
00002 #define __HAL_LKSMCU_OPA_H__
```

```

00003 #include "stdint.h"
00004 #include "hal_lksmcu_adc.h"
00005 typedef enum
00006 {
00007     HAL_OPA_0,
00008     HAL_OPA_1,
00009     HAL_OPA_2,
00010     HAL_OPA_3,
00011     HAL_OPA_4,
00012     HAL_OPA_5,
00013     HAL_OPA_0B,
00014 } HAL_OPA_x;
00015
00016 typedef enum
00017 {
00018     // 08
00019     HAL_OPA_GAIN_200_10_4 = 0,
00020     HAL_OPA_GAIN_190_20_4 = 1,
00021     HAL_OPA_GAIN_180_30_4 = 2,
00022     HAL_OPA_GAIN_170_40_4 = 3,
00023
00024     HAL_OPA_GAIN_HIGH = HAL_OPA_GAIN_200_10_4,
00025     HAL_OPA_GAIN_MEDIUM = HAL_OPA_GAIN_190_20_4,
00026     HAL_OPA_GAIN_LOW = HAL_OPA_GAIN_180_30_4,
00027
00028 #ifdef HAL_TEST_NOT_SUPPORTED
00029     HAL_OPA_GAIN_200_10,
00030     HAL_OPA_GAIN_190_20,
00031     HAL_OPA_GAIN_180_30,
00032     HAL_OPA_GAIN_170_40,
00033     HAL_OPA_GAIN_200_10_6,
00034     HAL_OPA_GAIN_190_20_6,
00035     HAL_OPA_GAIN_180_30_6,
00036     HAL_OPA_GAIN_170_40_6,
00037     HAL_OPA_GAIN_660_20,
00038     HAL_OPA_GAIN_640_40,
00039     HAL_OPA_GAIN_320_40,
00040     HAL_OPA_GAIN_160_40,
00041     HAL_OPA_GAIN_20_10,
00042     HAL_OPA_GAIN_40_10,
00043     HAL_OPA_GAIN_80_10,
00044     HAL_OPA_GAIN_160_10,
00045     HAL_OPA_GAIN_320_10,
00046     HAL_OPA_GAIN_320_5,
00047 #endif
00048 } HAL_OPA_GAIN_x;
00049
00050 // 初始化
00051 void HAL_OPA_Reset(HAL_OPA_x); // 模块复位
00052 void HAL_OPA_Enable(HAL_OPA_x); // 模块打开
00053 void HAL_OPA_Disable(HAL_OPA_x); // 模块关闭
00054
00055 void HAL_OPA_SetGain(HAL_OPA_x, HAL_OPA_GAIN_x); // 设置增益
00056 void HAL_OPA_Out2GpioEnable(HAL_OPA_x);
00057 void HAL_OPA_Out2GpioDisable(HAL_OPA_x);
00058 void HAL_OPA_SetGpioIn(HAL_OPA_x, HAL_GPIO_Px);
00059 void HAL_OPA_SetGpioOut(HAL_OPA_x, HAL_GPIO_Px);
00060
00061 #define HAL_OPA_OPAX_2_ADCCHN(OPAX) ((OPAX) == HAL_OPA_0B ? HAL_ADC_CHANNEL_0 : (HAL_ADC_CHANNEL_x)(OPAX))
00062 #define HAL_OPA_READ_GAIN(GAIN_x, res) (GAIN_x == HAL_OPA_GAIN_HIGH ? (200.0f / (10.4f + (float)res)) :
00063     (GAIN_x == HAL_OPA_GAIN_MEDIUM ? (190.0f / (20.4f + (float)res)) \
00064     : (180.0f / (30.4f + (float)res)))
00065
00066 #endif

```

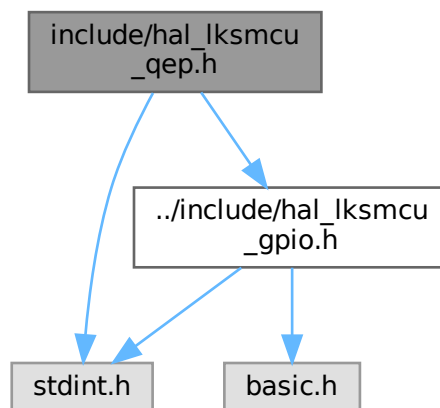
4.45 include/hal_lksmcu_gpi.h 文件参考

```

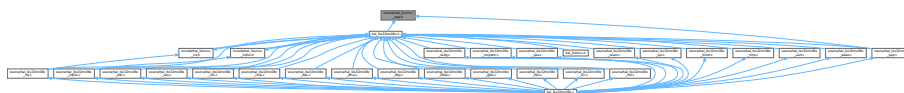
#include <stdint.h>
#include "../include/hal_lksmcu_gpio.h"

```

hal_lksmcu_qep.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



枚举

- enum HAL_QEP_x { HAL_QEP_0 , HAL_QEP_1 , HAL_QEP_2 , HAL_QEP_3 }
- enum HAL_QEP_MODE_x { HAL_QEP_MODE_QUADRATURE , HAL_QEP_MODE_SYMBOL_PULSE , HAL_QEP_MODE_DUAL_PULSE }

函数

- void HAL_QEP_Reset (HAL_QEP_x)
模块复位
- void HAL_QEP_Enable (HAL_QEP_x)
模块打开
- void HAL_QEP_Disable (HAL_QEP_x)
模块关闭
- void HAL_QEP_SetTh (HAL_QEP_x, uint32_t th)
设置最大计数值
- void HAL_QEP_SetMode (HAL_QEP_x, HAL_QEP_MODE_x)
设置计数模式
- uint32_t HAL_QEP_ReadCnt (HAL_QEP_x)
获取当前的计数值
- void HAL_QEP_ResetCnt (HAL_QEP_x)
复位计数值到0

- void HAL_QEP_SetClockDiv (HAL_QEP_x qep_x, uint32_t div)
 - uint32_t HAL_QEP_GetClockDiv (HAL_QEP_x qep_x)
 - uint32_t HAL_QEP_GetClockFreq (HAL_QEP_x qep_x)
 - void HAL_QEP_SetGpio (HAL_QEP_x, HAL_GPIO_Px)
- 设置QEP模块的引脚

4.45.1 枚举类型说明

4.45.1.1 HAL_QEP_MODE_x

enum HAL_QEP_MODE_x

枚举值

HAL_QEP_MODE_QUADRATURE	
HAL_QEP_MODE_SYMBOL_PULSE	
HAL_QEP_MODE_DUAL_PULSE	

在文件 `hal_lksmcu_qep.h` 第 13 行定义.

4.45.1.2 HAL_QEP_x

enum HAL_QEP_x

枚举值

HAL_QEP_0	
HAL_QEP_1	
HAL_QEP_2	
HAL_QEP_3	

在文件 `hal_lksmcu_qep.h` 第 5 行定义.

4.45.2 函数说明

4.45.2.1 HAL_QEP_Disable()

```
void HAL_QEP_Disable (
    HAL_QEP_x qep )
```

模块关闭

参数

<i>qep</i>	QEP模块

在文件 [hal_lks32mc08x.qep.c](#) 第 46 行定义.

4.45.2.2 HAL_QEP_Enable()

```
void HAL_QEP_Enable (
    HAL_QEP_x qep )
```

模块打开

参数

<i>qep</i>	QEP模块

在文件 [hal_lks32mc08x.qep.c](#) 第 25 行定义.

函数调用图:



4.45.2.3 HAL_QEP_GetClockDiv()

```
uint32_t HAL_QEP_GetClockDiv (
    HAL_QEP_x qep_x )
```

在文件 [hal_lks32mc08x.qep.c](#) 第 110 行定义.

函数调用图:



这是这个函数的调用关系图:

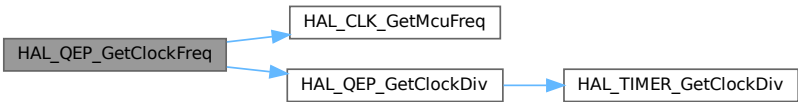


4.45.2.4 HAL_QEP_GetClockFreq()

```
uint32_t HAL_QEP_GetClockFreq (
    HAL_QEP_x qep_x )
```

在文件 [hal_lks32mc08x.qep.c](#) 第 116 行定义.

函数调用图:



4.45.2.5 HAL_QEP_ReadCnt()

```
uint32_t HAL_QEP_ReadCnt (
    HAL_QEP_x qep )
```

获取当前的计数值

参数

<i>qep</i>	QEP模块

返回

当前的计数值

在文件 [hal_lks32mc08x.qep.c](#) 第 89 行定义.

4.45.2.6 HAL_QEP_Reset()

```
void HAL_QEP_Reset (
    HAL_QEP_x qep )
```

模块复位

参数

<i>qep</i>	QEP模块

在文件 [hal_lks32mc08x_qep.c](#) 第 14 行定义.

4.45.2.7 HAL_QEP_ResetCnt()

```
void HAL_QEP_ResetCnt (
    HAL_QEP_x qep )
```

复位计数值到0

参数

<i>qep</i>	QEP模块

在文件 [hal_lks32mc08x_qep.c](#) 第 98 行定义.

4.45.2.8 HAL_QEP_SetClockDiv()

```
void HAL_QEP_SetClockDiv (
    HAL_QEP_x qep_x,
    uint32_t div )
```

在文件 [hal_lks32mc08x_qep.c](#) 第 104 行定义.

函数调用图:



4.45.2.9 HAL_QEP_SetGpio()

```
void HAL_QEP_SetGpio (
    HAL_QEP_x qep_x,
    HAL_GPIO_Px px )
```

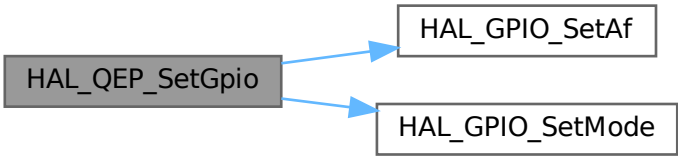
设置QEP模块的引脚

参数

<i>qepx</i>	QEP模块
<i>px</i>	引脚

在文件 [hal_lks32mc08x.qep.c](#) 第 126 行定义.

函数调用图:



4.45.2.10 HAL.QEP_SetMode()

```
void HAL_QEP_SetMode (
    HAL_QEP_x qep,
    HAL_QEP_MODE_x mode )
```

设置计数模式

参数

<i>qep</i>	QEP模块
<i>mode</i>	计数模式

在文件 [hal_lks32mc08x.qep.c](#) 第 66 行定义.

4.45.2.11 HAL.QEP_SetTh()

```
void HAL_QEP_SetTh (
    HAL_QEP_x qep,
    uint32_t th )
```

设置最大计数值

参数

<i>qep</i>	QEP模块
<i>th</i>	最大计数值

在文件 `hal_lks32mc08x.qep.c` 第 56 行定义.

4.46 hal_lksmcu_qep.h

浏览该文件的文档.

```
00001 #ifndef __HAL_LKSMCU_QEP_H__
00002 #define __HAL_LKSMCU_QEP_H__
00003 #include <stdint.h>
00004 #include "../include/hal_lksmcu_gpio.h"
00005 typedef enum
00006 {
00007     HAL_QEP_0,
00008     HAL_QEP_1,
00009     HAL_QEP_2,
00010     HAL_QEP_3,
00011 } HAL_QEP_x;
00012
00013 typedef enum
00014 {
00015     HAL_QEP_MODE_QUADRATURE, // 正交编码计数模式
00016     HAL_QEP_MODE_SYMBOL_PULSE, // 符号加脉冲计数模式
00017     HAL_QEP_MODE_DUAL_PULSE, // 双脉冲计数模式
00018 } HAL_QEP_MODE_x;
00019 void HAL_QEP_Reset(HAL_QEP_x); // 模块复位
00020 void HAL_QEP_Enable(HAL_QEP_x); // 模块打开
00021 void HAL_QEP_Disable(HAL_QEP_x); // 模块关闭
00022
00023 void HAL_QEP_SetTh(HAL_QEP_x, uint32_t th); // 设置最大计数值
00024 void HAL_QEP_SetMode(HAL_QEP_x, HAL_QEP_MODE_x); // 设置计数模式
00025 uint32_t HAL_QEP_ReadCnt(HAL_QEP_x); // 获取当前的计数值
00026 void HAL_QEP_ResetCnt(HAL_QEP_x); // 复位计数值到0
00027 void HAL_QEP_SetClockDiv(HAL_QEP_x qep_x, uint32_t div); // 设置时钟分频 div=1/2/4/8 (复用TIMER2/3分频)
00028 uint32_t HAL_QEP_GetClockDiv(HAL_QEP_x qep_x); // 获取时钟分频实际除数
00029 uint32_t HAL_QEP_GetClockFreq(HAL_QEP_x qep_x); // 获取分频后工作时钟 (Hz)
00030
00031 void HAL_QEP_SetGpio(HAL_QEP_x, HAL_GPIO_Px); // 设置QEP模块的引脚
00032 #endif
```

4.47 include/hal_lksmcu_reg.h 文件参考

HAL库配置和内部参数

此图展示该文件直接或间接的被哪些文件引用了:



宏定义

- #define HAL_REG_WRITE(reg, mask) reg = (mask)
- #define HAL_REG_READ(reg) (reg)
- #define HAL_REG_READ_ADDR(reg) (&reg)
- #define HAL_REG_WRITE_BIT(reg, mask, val) reg = ((reg) & ~(mask)) | ((val) & (mask))
- #define HAL_REG_WRITE_BIT_MASK_POS(reg, name, val) reg = ((reg) & ~(name##_MASK)) | (name(val))
- #define HAL_REG_READ_BIT_MASK_POS(reg, mask) (((reg) & (mask##_MASK)) >> mask##_POS)
- #define HAL_REG_READ_BIT(reg, mask) ((reg) & (mask))
- #define HAL_REG_SET(reg, mask) reg |= (mask)
- #define HAL_REG_RESET(reg, mask) reg &= ~(mask)
- #define HAL_REG_SYS_LOCK() SYS_WR_PROTECT = 0
- #define HAL_REG_SYS_UNLOCK() SYS_WR_PROTECT = 0x7a83
- #define HAL_REG_IF_WRITE(reg, mask, condition) reg = (condition) ? ((reg) | (mask)) : ((reg) & ~(mask))

4.47.1 详细描述

HAL库配置和内部参数

在文件 [hal_lksmcu_reg.h](#) 中定义.

4.47.2 宏定义说明

4.47.2.1 HAL_REG_IF_WRITE

```
#define HAL_REG_IF_WRITE(  
    reg,  
    mask,  
    condition ) reg = (condition) ? ((reg) | (mask)) : ((reg) & ~(mask))
```

在文件 [hal_lksmcu_reg.h](#) 第 19 行定义.

4.47.2.2 HAL_REG_READ

```
#define HAL_REG_READ(  
    reg ) (reg)
```

在文件 [hal_lksmcu_reg.h](#) 第 9 行定义.

4.47.2.3 HAL_REG_READ_ADDR

```
#define HAL_REG_READ_ADDR(  
    reg ) (&reg)
```

在文件 [hal_lksmcu_reg.h](#) 第 10 行定义.

4.47.2.4 HAL_REG_READ_BIT

```
#define HAL_REG_READ_BIT(  
    reg,  
    mask ) (((reg) & (mask))
```

在文件 [hal_lksmcu_reg.h](#) 第 14 行定义.

4.47.2.5 HAL_REG_READ_BIT_MASK_POS

```
#define HAL_REG_READ_BIT_MASK_POS(  
    reg,  
    mask ) (((reg) & (mask##_MASK)) >> mask##_POS)
```

在文件 [hal_lksmcu_reg.h](#) 第 13 行定义.

4.47.2.6 HAL_REG_RESET

```
#define HAL_REG_RESET(  
    reg,  
    mask ) reg &= ~(mask)
```

在文件 [hal_lksmcu_reg.h](#) 第 16 行定义.

4.47.2.7 HAL_REG_SET

```
#define HAL_REG_SET(  
    reg,  
    mask ) reg |= (mask)
```

在文件 [hal_lksmcu_reg.h](#) 第 15 行定义.

4.47.2.8 HAL_REG_SYS_LOCK

```
#define HAL_REG_SYS_LOCK( ) SYS_WR_PROTECT = 0
```

在文件 [hal_lksmcu_reg.h](#) 第 17 行定义.

4.47.2.9 HAL_REG_SYS_UNLOCK

```
#define HAL_REG_SYS_UNLOCK( ) SYS_WR_PROTECT = 0x7a83
```

在文件 [hal_lksmcu_reg.h](#) 第 18 行定义.

4.47.2.10 HAL_REG_WRITE

```
#define HAL_REG_WRITE(  
    reg,  
    mask ) reg = (mask)
```

在文件 [hal_lksmcu_reg.h](#) 第 8 行定义.

4.47.2.11 HAL_REG_WRITE_BIT

```
#define HAL_REG_WRITE_BIT(  
    reg,  
    mask,  
    val ) reg = ((reg) & ~(mask)) | ((val) & (mask))
```

在文件 [hal_lksmcu_reg.h](#) 第 11 行定义.

4.47.2.12 HAL_REG_WRITE_BIT_MASK_POS

```
#define HAL_REG_WRITE_BIT_MASK_POS(  
    reg,  
    name,  
    val ) reg = ((reg) & ~(name##_MASK)) | (name(val))
```

在文件 `hal_iksmcu_reg.h` 第 12 行定义.

4.48 hal_ksmcu_reg.h

[浏览该文件的文档.](#)

```

00001 #ifndef __HAL_LKSMCU_REG_H__
00002 #define __HAL_LKSMCU_REG_H__
00003
00008 #define HAL_REG_WRITE(reg, mask)                reg = (mask)
00009 #define HAL_REG_READ(reg)                       (reg)
00010 #define HAL_REG_READ_ADDR(reg)                  (&reg)
00011 #define HAL_REG_WRITE_BIT(reg, mask, val)        reg = ((reg) & ~(mask)) | ((val) & (mask))
00012 #define HAL_REG_WRITE_BIT_MASK_POS(reg, name, val) reg = ((reg) & ~(name##_MASK)) | (name(val))
00013 #define HAL_REG_READ_BIT_MASK_POS(reg, mask)      (((reg) & (mask##_MASK)) >> mask##_POS)
00014 #define HAL_REG_READ_BIT(reg, mask)              ((reg) & (mask))
00015 #define HAL_REG_SET(reg, mask)                   reg |= (mask)
00016 #define HAL_REG_RESET(reg, mask)                 reg &= ~(mask)
00017 #define HAL_REG_SYS_LOCK()                       SYS_WR_PROTECT = 0
00018 #define HAL_REG_SYS_UNLOCK()                     SYS_WR_PROTECT = 0x7a83
00019 #define HAL_REG_IF_WRITE(reg, mask, condition)    reg = (condition) ? ((reg) | (mask)) : ((reg) &
~(mask))
00020
00021 #endif

```

4.49 include/hal_iksmcu_reset.h 文件参考

此图展示该文件直接或间接的被哪些文件引用了:



函数

- void **HAL_RESET_ResetCore** (void)
复位内核
- void **HAL_RESET_ResetRegister** (void)
寄存器复位
- void **HAL_RESET_ResetGlobal** (void)
全局复位(先复位寄存器再复位内核)

4.49.1 函数说明

4.49.1.1 HAL_RESET_ResetCore()

```
void HAL_RESET_ResetCore (
    void )
```

复位内核

在文件 `hal_ks32mc08x_reset.c` 第 10 行定义.

4.49.1.2 HAL_RESET_ResetGlobal()

```
void HAL_RESET_ResetGlobal (
    void )
```

全局复位(先复位寄存器再复位内核)

在文件 [hal_lks32mc08x.reset.c](#) 第 35 行定义.

函数调用图:



4.49.1.3 HAL_RESET_ResetRegister()

```
void HAL_RESET_ResetRegister (
    void )
```

寄存器复位

在文件 [hal_lks32mc08x.reset.c](#) 第 18 行定义.

这是这个函数的调用关系图:



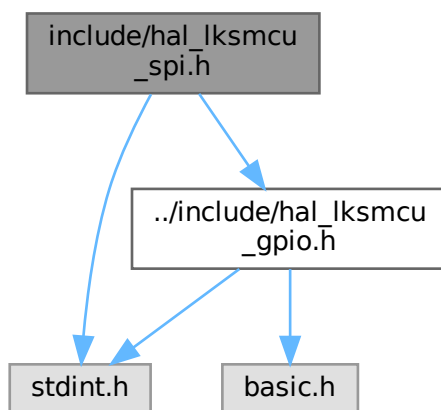
4.50 hal_lksmcu_reset.h

[浏览该文件的文档.](#)

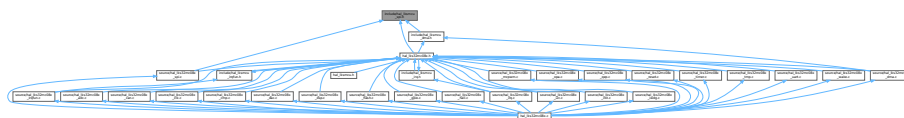
```
00001 #ifndef __HAL_LKSMCU_RESET_H__
00002 #define __HAL_LKSMCU_RESET_H__
00003 void HAL_RESET_ResetCore(void); // 复位内核
00004 void HAL_RESET_ResetRegister(void); // 寄存器复位
00005 void HAL_RESET_ResetGlobal(void); // 全局复位 (先复位寄存器再复位内核)
00006 #endif
```

4.51 include/hal_lksmcu_spi.h 文件参考

```
#include "stdint.h"
#include "../include/hal_lksmcu_gpio.h"
hal_lksmcu_spi.h 的引用(Include)关系图:
```



此图展示该文件直接或间接的被哪些文件引用了:



枚举

- enum HAL_SPI_x { HAL_SPI_0 , HAL_SPI_1 }
- enum HAL_SPI_MODE_x {
 HAL_SPI_MODE_MASTER_LSB_00 = 0 , HAL_SPI_MODE_MASTER_LSB_01 , HAL_SPI_MODE_MASTER_LSB_10 ,
 HAL_SPI_MODE_MASTER_LSB_11 ,
 HAL_SPI_MODE_MASTER_MSB_00 , HAL_SPI_MODE_MASTER_MSB_01 , HAL_SPI_MODE_MASTER_MSB_10 ,
 HAL_SPI_MODE_MASTER_MSB_11 ,
 HAL_SPI_MODE_SLAVE_LSB_00 , HAL_SPI_MODE_SLAVE_LSB_01 , HAL_SPI_MODE_SLAVE_LSB_10 ,
 HAL_SPI_MODE_SLAVE_LSB_11 ,
 HAL_SPI_MODE_SLAVE_MSB_00 , HAL_SPI_MODE_SLAVE_MSB_01 , HAL_SPI_MODE_SLAVE_MSB_10 ,
 HAL_SPI_MODE_SLAVE_MSB_11 }

函数

- void HAL_SPI_Reset (HAL_SPI_x)
 模块复位
- void HAL_SPI_Enable (HAL_SPI_x)

- 模块打开
- void [HAL_SPI_Disable](#) ([HAL_SPI_x](#))
- 模块关闭
- void [HAL_SPI_SetBaud](#) ([HAL_SPI_x](#) spi, uint32_t baud)
- 设置波特率
- void [HAL_SPI_SetClockDiv](#) ([HAL_SPI_x](#) spi, uint32_t div)
- uint32_t [HAL_SPI_GetClockDiv](#) ([HAL_SPI_x](#) spi)
- uint32_t [HAL_SPI_GetClockFreq](#) ([HAL_SPI_x](#) spi)
- uint32_t [HAL_SPI_GetBaud](#) ([HAL_SPI_x](#) spi)
- void [HAL_SPI_SetMode](#) ([HAL_SPI_x](#), [HAL_SPI_MODE_x](#))
- 设置SPI模式
- void [HAL_SPI_SetGpioCs](#) ([HAL_SPI_x](#), [HAL_GPIO_Px](#) pin)
- 初始化CS引脚
- void [HAL_SPI_SetGpioMosi](#) ([HAL_SPI_x](#), [HAL_GPIO_Px](#) pin)
- 初始化Mosi引脚
- void [HAL_SPI_SetGpioMiso](#) ([HAL_SPI_x](#), [HAL_GPIO_Px](#) pin)
- 初始化Miso引脚
- void [HAL_SPI_SetGpioClk](#) ([HAL_SPI_x](#), [HAL_GPIO_Px](#) pin)
- 初始化Clk引脚
- void [HAL_SPI_SetGpioDi](#) ([HAL_SPI_x](#), [HAL_GPIO_Px](#) pin)
- 初始化SPI的输入引脚
- void [HAL_SPI_SetGpioDo](#) ([HAL_SPI_x](#), [HAL_GPIO_Px](#) pin)
- 初始化SPI的输出引脚
- uint8_t [HAL_SPI_TxData](#) ([HAL_SPI_x](#), uint8_t *data, uint32_t len)
- 发送数据
- uint8_t [HAL_SPI_RxData](#) ([HAL_SPI_x](#), uint8_t *data, uint32_t len)
- 接收数据
- uint8_t [HAL_SPI_RxTxData](#) ([HAL_SPI_x](#), uint8_t *rxdata, uint8_t *txdata, uint32_t len)
- 发送和接收数据
- uint8_t [HAL_SPI_TxDataWidth16](#) ([HAL_SPI_x](#), uint16_t *data, uint32_t len)
- 发送数据
- uint8_t [HAL_SPI_RxDataWidth16](#) ([HAL_SPI_x](#), uint16_t *data, uint32_t len)
- 接收数据
- uint8_t [HAL_SPI_RxTxDataWidth16](#) ([HAL_SPI_x](#), uint16_t *rxdata, uint16_t *txdata, uint32_t len)
- 发送和接收数据

4.51.1 枚举类型说明

4.51.1.1 HAL_SPI_MODE_x

enum [HAL_SPI_MODE_x](#)

枚举值

HAL_SPI_MODE_MASTER_LSB_00	主模式 低位先发 CPOL0 CPHA0
HAL_SPI_MODE_MASTER_LSB_01	主模式 低位先发 CPOL0 CPHA1
HAL_SPI_MODE_MASTER_LSB_10	主模式 低位先发 CPOL1 CPHA0
HAL_SPI_MODE_MASTER_LSB_11	主模式 低位先发 CPOL1 CPHA1
HAL_SPI_MODE_MASTER_MSB_00	主模式 高位先发 CPOL0 CPHA0
HAL_SPI_MODE_MASTER_MSB_01	主模式 高位先发 CPOL0 CPHA1

枚举值

HAL_SPI_MODE_MASTER_MSB_10	主模式 高位先发 CPOL1 CPHA0
HAL_SPI_MODE_MASTER_MSB_11	主模式 高位先发 CPOL1 CPHA1
HAL_SPI_MODE_SLAVE_LSB_00	从模式 低位先发 CPOL0 CPHA0
HAL_SPI_MODE_SLAVE_LSB_01	从模式 低位先发 CPOL0 CPHA1
HAL_SPI_MODE_SLAVE_LSB_10	从模式 低位先发 CPOL1 CPHA0
HAL_SPI_MODE_SLAVE_LSB_11	从模式 低位先发 CPOL1 CPHA1
HAL_SPI_MODE_SLAVE_MSB_00	从模式 高位先发 CPOL0 CPHA0
HAL_SPI_MODE_SLAVE_MSB_01	从模式 高位先发 CPOL0 CPHA1
HAL_SPI_MODE_SLAVE_MSB_10	从模式 高位先发 CPOL1 CPHA0
HAL_SPI_MODE_SLAVE_MSB_11	从模式 高位先发 CPOL1 CPHA1

在文件 [hal_lksmcu_spi.h](#) 第 10 行定义.

4.51.1.2 HAL_SPI_x

enum [HAL_SPI_x](#)

枚举值

HAL_↔ SPI_0	
HAL_↔ SPI_1	

在文件 [hal_lksmcu_spi.h](#) 第 5 行定义.

4.51.2 函数说明

4.51.2.1 HAL_SPI_Disable()

```
void HAL_SPI_Disable (
    HAL\_SPI\_x spi )
```

模块关闭

参数

<i>spi</i>	SPI模块号

在文件 [hal_lks32mc08x_spi.c](#) 第 37 行定义.

4.51.2.2 HAL_SPI_Enable()

```
void HAL_SPI_Enable (
```

```
HAL_SPI_<x> spi )
```

模块打开

参数

<i>spi</i>	SPI模块号

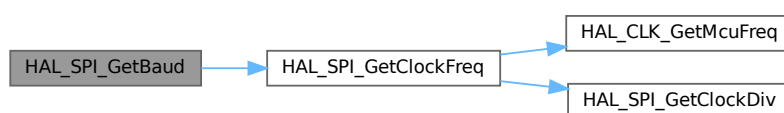
在文件 [hal_lks32mc08x_spi.c](#) 第 25 行定义.

4.51.2.3 HAL_SPI_GetBaud()

```
uint32_t HAL_SPI_GetBaud (
    HAL_SPI_<x> spi )
```

在文件 [hal_lks32mc08x_spi.c](#) 第 76 行定义.

函数调用图:



4.51.2.4 HAL_SPI_GetClockDiv()

```
uint32_t HAL_SPI_GetClockDiv (
    HAL_SPI_<x> spi )
```

在文件 [hal_lks32mc08x_spi.c](#) 第 64 行定义.

这是这个函数的调用关系图:

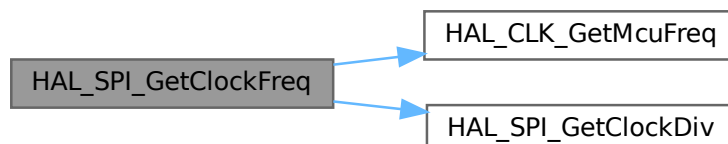


4.51.2.5 HAL_SPI_GetClockFreq()

```
uint32_t HAL_SPI_GetClockFreq (  
    HAL_SPI_TypeDef* spi )
```

在文件 [hal_lks32mc08x_spi.c](#) 第 70 行定义.

函数调用图:



这是这个函数的调用关系图:



4.51.2.6 HAL_SPI_Reset()

```
void HAL_SPI_Reset (  
    HAL_SPI_TypeDef* spi )
```

模块复位

参数

<i>spi</i>	SPI模块号

在文件 [hal_lks32mc08x_spi.c](#) 第 12 行定义.

4.51.2.7 HAL_SPI_RxData()

```
uint8_t HAL_SPI_RxData (  
    HAL_SPI_TypeDef* spi,
```

```
uint8_t * data,
uint32_t len )
```

接收数据

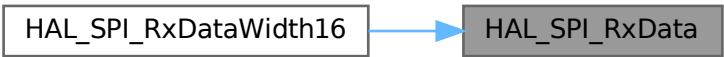
参数

<i>spi</i>	SPI模块号
<i>data</i>	接收缓冲区
<i>len</i>	数据长度

返回
接收状态

在文件 [hal_lks32mc08x_spi.c](#) 第 287 行定义.

这是这个函数的调用关系图:



4.51.2.8 HAL_SPI_RxDataWidth16()

```
uint8_t HAL_SPI_RxDataWidth16 (
    HAL_SPI_x spi,
    uint16_t * data,
    uint32_t len )
```

接收数据

参数

<i>spi</i>	SPI模块号
<i>data</i>	接收缓冲区
<i>len</i>	数据长度，单位2字节

返回
接收状态

注解

lks32mc08x不支持16位模式，这里用8位模式发送

在文件 [hal_lks32mc08x_spi.c](#) 第 354 行定义.

函数调用图:



4.51.2.9 HAL_SPI_RxTxData()

```
uint8_t HAL_SPI_RxTxData (
    HAL_SPI_x spi,
    uint8_t * rxdata,
    uint8_t * txdata,
    uint32_t len )
```

发送和接收数据

参数

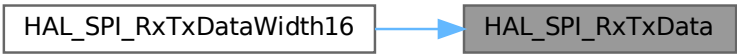
<i>spi</i>	SPI模块号
<i>rxdata</i>	接收缓冲区
<i>txdata</i>	要发送的数据
<i>len</i>	数据长度

返回

发送和接收状态

在文件 [hal_lks32mc08x_spi.c](#) 第 314 行定义.

这是这个函数的调用关系图:



4.51.2.10 HAL_SPI.RxTxDataWidth16()

```
uint8_t HAL_SPI_RxTxDataWidth16 (
    HAL_SPI_TypeDef * spi,
    uint16_t * rxdata,
    uint16_t * txdata,
    uint32_t len )
```

发送和接收数据

参数

<i>spi</i>	SPI模块号
<i>rxdata</i>	接收缓冲区
<i>txdata</i>	要发送的数据
<i>len</i>	数据长度，单位2字节

返回

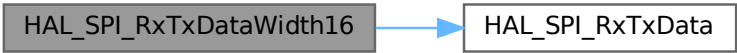
发送和接收状态

注解

lks32mc08x不支持16位模式，这里用8位模式发送

在文件 [hal_lks32mc08x_spi.c](#) 第 369 行定义.

函数调用图:



4.51.2.11 HAL_SPI.SetBaud()

```
void HAL_SPI_SetBaud (
    HAL_SPI_TypeDef * spi,
    uint32_t baud )
```

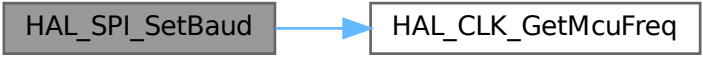
设置波特率

参数

<i>spi</i>	SPI模块号

在文件 `hal_lks32mc08x_spi.c` 第 49 行定义.

函数调用图:



4.51.2.12 HAL_SPI.SetClockDiv()

```
void HAL_SPI_SetClockDiv (
    HAL_SPI_x spi,
    uint32_t div )
```

在文件 `hal_lks32mc08x_spi.c` 第 57 行定义.

4.51.2.13 HAL_SPI.SetGpioClk()

```
void HAL_SPI_SetGpioClk (
    HAL_SPI_x spi,
    HAL_GPIO_Px pin )
```

初始化Clk引脚

参数

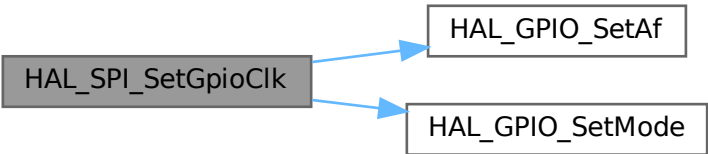
<i>spi</i>	SPI模块号
<i>pin</i>	GPIO引脚

参见

需要先初始化SPI

在文件 [hal_lks32mc08x_spi.c](#) 第 216 行定义.

函数调用图:



4.51.2.14 HAL_SPI.SetGpioCs()

```
void HAL_SPI_SetGpioCs (
    HAL_SPI_TypeDef* spi,
    GPIO_TypeDef* pin )
```

初始化CS引脚

参数

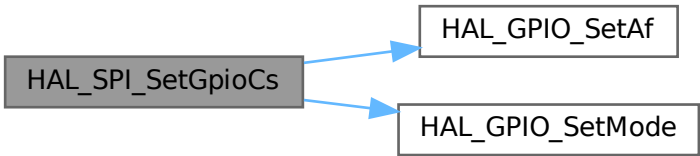
<i>spi</i>	SPI模块号
<i>pin</i>	GPIO引脚

参见

需要先初始化SPI

在文件 [hal_lks32mc08x_spi.c](#) 第 150 行定义.

函数调用图:



4.51.2.15 HAL_SPI.SetGpioDi()

```
void HAL_SPI_SetGpioDi (
    HAL_SPI_TypeDef* spi,
    GPIO_TypeDef* pin )
```

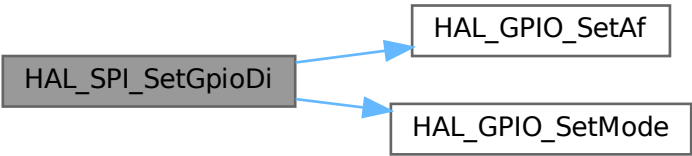
初始化SPI的输入引脚

参数

spi	SPI模块号
pin	GPIO引脚

在文件 [hal_lks32mc08x_spi.c](#) 第 237 行定义.

函数调用图:



4.51.2.16 HAL_SPI.SetGpioDo()

```
void HAL_SPI_SetGpioDo (
    HAL_SPI_TypeDef* spi,
    GPIO_TypeDef* pin )
```

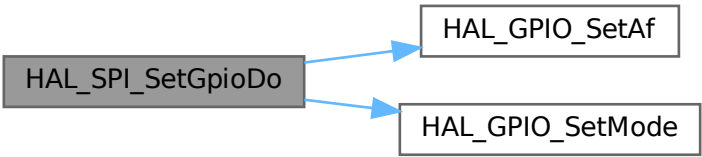
初始化SPI的输出引脚

参数

spi	SPI模块号
pin	GPIO引脚

在文件 [hal_lks32mc08x_spi.c](#) 第 249 行定义.

函数调用图:



4.51.2.17 HAL_SPI_SetGpioMiso()

```
void HAL_SPI_SetGpioMiso (
    HAL_SPI_TypeDef* spi,
    GPIO_TypeDef* pin )
```

初始化Miso引脚

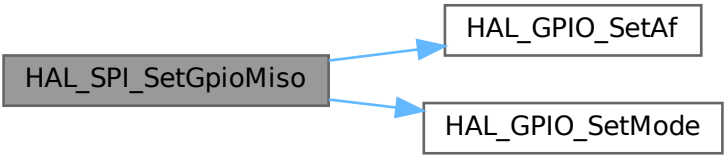
参数

<i>spi</i>	SPI模块号
<i>pin</i>	GPIO引脚

参见
需要先初始化SPI

在文件 [hal_lks32mc08x_spi.c](#) 第 194 行定义.

函数调用图:



4.51.2.18 HAL_SPI.SetGpioMosi()

```
void HAL_SPI_SetGpioMosi (
    HAL_SPI_TypeDef* spi,
    GPIO_TypeDef* pin )
```

初始化Mosi引脚

参数

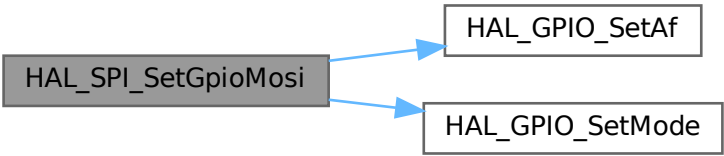
spi	SPI模块号
pin	GPIO引脚

参见

需要先初始化SPI

在文件 [hal_lks32mc08x_spi.c](#) 第 172 行定义.

函数调用图:



4.51.2.19 HAL_SPI.SetMode()

```
void HAL_SPI_SetMode (
    HAL_SPI_TypeDef* spi,
    HAL_SPI_Mode_TypeDef mode )
```

设置SPI模式

参数

spi	SPI模块号
mode	SPI模式

在文件 [hal_lks32mc08x_spi.c](#) 第 86 行定义.

4.51.2.20 HAL_SPI_TxData()

```
uint8_t HAL_SPI_TxData (
    HAL_SPI_TypeDef * spi,
    uint8_t * data,
    uint32_t len )
```

发送数据

参数

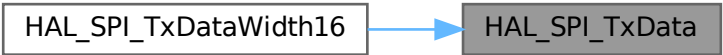
<i>spi</i>	SPI模块号
<i>data</i>	要发送的数据
<i>len</i>	数据长度

返回

发送状态

在文件 [hal_lks32mc08x_spi.c](#) 第 263 行定义.

这是这个函数的调用关系图:



4.51.2.21 HAL_SPI_TxDataWidth16()

```
uint8_t HAL_SPI_TxDataWidth16 (
    HAL_SPI_TypeDef * spi,
    uint16_t * data,
    uint32_t len )
```

发送数据

参数

<i>spi</i>	SPI模块号
<i>data</i>	要发送的数据
<i>len</i>	数据长度，单位2字节

返回

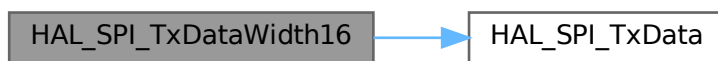
发送状态

注解

lks32mc08x不支持16位模式，这里用8位模式发送

在文件 `hal_lks32mc08x_spi.c` 第 340 行定义。

函数调用图:



4.52 hal_lksmcu_spi.h

[浏览该文件的文档.](#)

```

00001 #ifndef __HAL_LKSMCU_SPI_H__
00002 #define __HAL_LKSMCU_SPI_H__
00003 #include "stdint.h"
00004 #include "../include/hal_lksmcu_gpio.h"
00005 typedef enum
00006 {
00007     HAL_SPI_0,
00008     HAL_SPI_1,
00009 } HAL_SPI_x;
00010 typedef enum
00011 {
00012     HAL_SPI_MODE_MASTER_LSB_00 = 0,
00013     HAL_SPI_MODE_MASTER_LSB_01,
00014     HAL_SPI_MODE_MASTER_LSB_10,
00015     HAL_SPI_MODE_MASTER_LSB_11,
00016     HAL_SPI_MODE_MASTER_MSB_00,
00017     HAL_SPI_MODE_MASTER_MSB_01,
00018     HAL_SPI_MODE_MASTER_MSB_10,
00019     HAL_SPI_MODE_MASTER_MSB_11,
00020     HAL_SPI_MODE_SLAVE_LSB_00,
00021     HAL_SPI_MODE_SLAVE_LSB_01,
00022     HAL_SPI_MODE_SLAVE_LSB_10,
00023     HAL_SPI_MODE_SLAVE_LSB_11,
00024     HAL_SPI_MODE_SLAVE_MSB_00,
00025     HAL_SPI_MODE_SLAVE_MSB_01,
00026     HAL_SPI_MODE_SLAVE_MSB_10,
00027     HAL_SPI_MODE_SLAVE_MSB_11,
00028 } HAL_SPI_MODE_x;
00029 // 初始化
00030 void HAL_SPI_Reset(HAL_SPI_x); // 模块复位
00031 void HAL_SPI_Enable(HAL_SPI_x); // 模块打开
00032 void HAL_SPI_Disable(HAL_SPI_x); // 模块关闭
00033
00034 void HAL_SPI_SetBaud(HAL_SPI_x spi, uint32_t baud);
00035 void HAL_SPI_SetClockDiv(HAL_SPI_x spi, uint32_t div); // 设置时钟分频 div=实际除数(2,4,6...)
00036 uint32_t HAL_SPI_GetClockDiv(HAL_SPI_x spi); // 获取时钟分频实际除数
00037 uint32_t HAL_SPI_GetClockFreq(HAL_SPI_x spi); // 获取SCK时钟频率(Hz)
00038 uint32_t HAL_SPI_GetBaud(HAL_SPI_x spi); // 获取当前波特率(=SCK频率)
00039 void HAL_SPI_SetMode(HAL_SPI_x, HAL_SPI_MODE_x);
00040
00041 void HAL_SPI_SetGpioCs(HAL_SPI_x, HAL_GPIO_Px pin);
00042 void HAL_SPI_SetGpioMosi(HAL_SPI_x, HAL_GPIO_Px pin);
00043 void HAL_SPI_SetGpioMiso(HAL_SPI_x, HAL_GPIO_Px pin);
00044 void HAL_SPI_SetGpioClk(HAL_SPI_x, HAL_GPIO_Px pin);
  
```

```

00045
00046 void HAL_SPI_SetGpioDi(HAL_SPI_x, HAL_GPIO_Px pin); // 不自动识别模式, 直接指定输入或输出状态
00047 void HAL_SPI_SetGpioDo(HAL_SPI_x, HAL_GPIO_Px pin); // 不自动识别模式, 直接指定输入或输出状态
00048
00049 uint8_t HAL_SPI_TxData(HAL_SPI_x, uint8_t *data, uint32_t len);
00050 uint8_t HAL_SPI_RxData(HAL_SPI_x, uint8_t *data, uint32_t len);
00051 uint8_t HAL_SPI_RxTxData(HAL_SPI_x, uint8_t *rxdata, uint8_t *txdata, uint32_t len);
00052 uint8_t HAL_SPI_TxDataWidth16(HAL_SPI_x, uint16_t *data, uint32_t len);
00053 uint8_t HAL_SPI_RxDataWidth16(HAL_SPI_x, uint16_t *data, uint32_t len);
00054 uint8_t HAL_SPI_RxTxDataWidth16(HAL_SPI_x, uint16_t *rxdata, uint16_t *txdata, uint32_t len);
00055 #endif

```

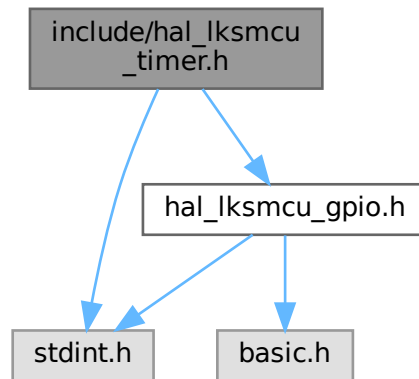
4.53 include/hal_lksmcu_timer.h 文件参考

```

#include "stdint.h"
#include "hal_lksmcu_gpio.h"

```

hal_lksmcu_timer.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



枚举

- enum HAL_TIMER_x {
 HAL_TIMER_0, HAL_TIMER_1, HAL_TIMER_2, HAL_TIMER_3,
 HAL_TIMER_4 }
- enum HAL_TIMER_MODE_x { HAL_TIMER_MODE_COUNT, HAL_TIMER_MODE_PWMOUT, HAL_TIMER_MODE_PWMIN }

函数

- void [HAL_TIMER_Reset](#) ([HAL_TIMER_x](#))
- void [HAL_TIMER_Enable](#) ([HAL_TIMER_x](#))
- void [HAL_TIMER_Disable](#) ([HAL_TIMER_x](#))
- void [HAL_TIMER_SetPeriodCount](#) ([HAL_TIMER_x](#), uint32_t count)
设置定时器周期
- void [HAL_TIMER_SetFreq](#) ([HAL_TIMER_x](#), uint32_t freq)
设置定时器频率
- void [HAL_TIMER_SetPeriodms](#) ([HAL_TIMER_x](#), uint32_t ms)
设置定时器周期(*ms*)
- void [HAL_TIMER_SetPeriodus](#) ([HAL_TIMER_x](#), uint32_t us)
设置定时器周期(*us*)
- uint32_t [HAL_TIMER_GetFreq](#) ([HAL_TIMER_x](#))
- uint32_t [HAL_TIMER_GetThCnt](#) ([HAL_TIMER_x](#))
- void [HAL_TIMER_SetClockDiv](#) ([HAL_TIMER_x](#) timer_x, uint32_t div)
- uint32_t [HAL_TIMER_GetClockDiv](#) ([HAL_TIMER_x](#) timer_x)
- uint32_t [HAL_TIMER_GetClockFreq](#) ([HAL_TIMER_x](#) timer_x)
- void [HAL_TIMER_StartCounting](#) ([HAL_TIMER_x](#))
- void [HAL_TIMER_StopCounting](#) ([HAL_TIMER_x](#))
- void [HAL_TIMER_SetPwm](#) ([HAL_TIMER_x](#) timer_x, int32_t t0, int32_t t1)
- void [HAL_TIMER_SetPwmQ15](#) ([HAL_TIMER_x](#) timer_x, int16_t t0, int16_t t1)
- void [HAL_TIMER_SetWorkMode](#) ([HAL_TIMER_x](#) timer_x, [HAL_TIMER_MODE_x](#) mode_x)
- void [HAL_TIMER_SetGpioOut](#) ([HAL_TIMER_x](#) timer_x, [HAL_GPIO_Px](#) px)
设置定时器输出 *gpio*
- void [HAL_TIMER_SetGpioIn](#) ([HAL_TIMER_x](#) timer_x, [HAL_GPIO_Px](#) px)
设置定时器输入 *gpio*

4.53.1 枚举类型说明

4.53.1.1 HAL_TIMER_MODE_x

enum [HAL_TIMER_MODE_x](#)

枚举值

HAL_TIMER_MODE_COUNT	计数模式
HAL_TIMER_MODE_PWMOUT	PWM输出模式
HAL_TIMER_MODE_PWMIN	PWM输入模式

在文件 [hal_lksmcu_timer.h](#) 第 14 行定义.

4.53.1.2 HAL_TIMER_x

enum [HAL_TIMER_x](#)

枚举值

HAL↔ TIMER_0	TIMER模块0
HAL↔ TIMER_1	TIMER模块1
HAL↔ TIMER_2	TIMER模块2
HAL↔ TIMER_3	TIMER模块3
HAL↔ TIMER_4	TIMER模块3

在文件 [hal_lksmcu_timer.h](#) 第 6 行定义.

4.53.2 函数说明

4.53.2.1 HAL_TIMER_Disable()

```
void HAL_TIMER_Disable (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc08x_timer.c](#) 第 27 行定义.

4.53.2.2 HAL_TIMER_Enable()

```
void HAL_TIMER_Enable (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc08x_timer.c](#) 第 16 行定义.

这是这个函数的调用关系图:

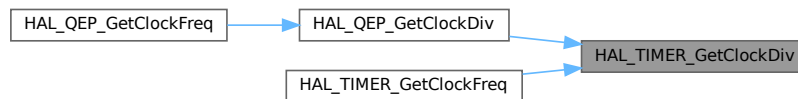


4.53.2.3 HAL_TIMER_GetClockDiv()

```
uint32_t HAL_TIMER_GetClockDiv (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc08x.timer.c](#) 第 154 行定义.

这是这个函数的调用关系图:

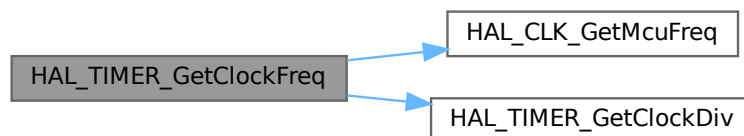


4.53.2.4 HAL_TIMER_GetClockFreq()

```
uint32_t HAL_TIMER_GetClockFreq (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc08x.timer.c](#) 第 167 行定义.

函数调用图:



4.53.2.5 HAL_TIMER_GetFreq()

```
uint32_t HAL_TIMER_GetFreq (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc08x.timer.c](#) 第 123 行定义.

函数调用图:



4.53.2.6 HAL_TIMER_GetThCnt()

```
uint32_t HAL_TIMER_GetThCnt (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc08x.timer.c](#) 第 173 行定义.

这是这个函数的调用关系图:



4.53.2.7 HAL_TIMER_Reset()

```
void HAL_TIMER_Reset (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc08x.timer.c](#) 第 11 行定义.

4.53.2.8 HAL_TIMER_SetClockDiv()

```
void HAL_TIMER_SetClockDiv (
    HAL_TIMER_x timer_x,
    uint32_t div )
```

在文件 [hal_lks32mc08x.timer.c](#) 第 136 行定义.

这是这个函数的调用关系图:



4.53.2.9 HAL_TIMER_SetFreq()

```
void HAL_TIMER_SetFreq (
    HAL_TIMER_x timer_x,
    uint32_t freq )
```

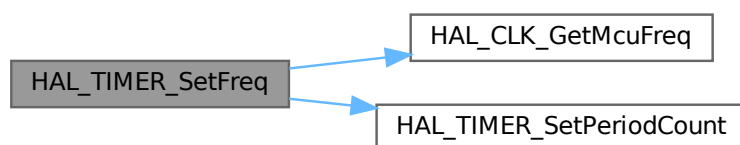
设置定时器频率

参数

<i>timer</i> ↔ _x	定时器编号
<i>freq</i>	定时器频率

在文件 [hal_lks32mc08x_timer.c](#) 第 92 行定义.

函数调用图:



4.53.2.10 HAL_TIMER_SetGpioIn()

```
void HAL_TIMER_SetGpioIn (
    HAL_TIMER_x timer_x,
    HAL_GPIO_Px px )
```

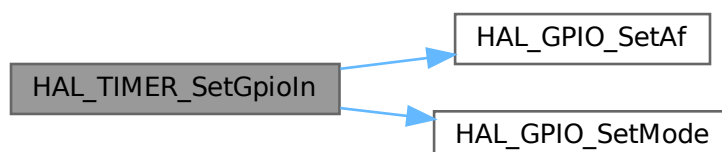
设置定时器输入gpio

参数

<i>timer</i> ↔ _x	定时器编号
<i>px</i>	gpio编号

在文件 [hal_lks32mc08x_timer.c](#) 第 296 行定义.

函数调用图:



4.53.2.11 HAL_TIMER_SetGpioOut()

```
void HAL_TIMER_SetGpioOut (
    HAL_TIMER_x timer_x,
    HAL_GPIO_Px px )
```

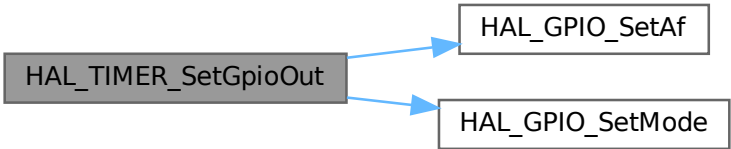
设置定时器输出gpio

参数

<i>timer_x</i>	定时器编号
<i>px</i>	gpio编号

在文件 [hal_lks32mc08x.timer.c](#) 第 266 行定义.

函数调用图:



4.53.2.12 HAL_TIMER_SetPeriodCount()

```
void HAL_TIMER_SetPeriodCount (
    HAL_TIMER_x timer_x,
    uint32_t count )
```

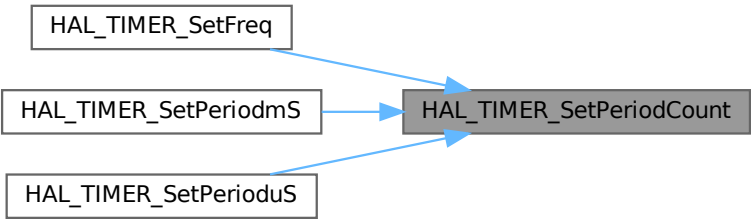
设置定时器周期

参数

<i>timer_x</i>	定时器编号
<i>count</i>	定时器周期

在文件 [hal_lks32mc08x.timer.c](#) 第 45 行定义.

这是这个函数的调用关系图:



4.53.2.13 HAL_TIMER_SetPeriodmS()

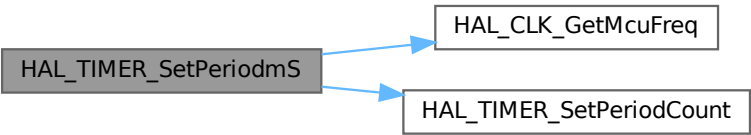
```
void HAL_TIMER.SetPeriodmS (
    HAL_TIMER.x timer_x,
    uint32_t ms )
```

设置定时器周期(ms)

参数

<i>timer_x</i>	定时器编号
<i>ms</i>	定时器周期(ms)

在文件 [hal_lks32mc08x.timer.c](#) 第 104 行定义.
函数调用图:



4.53.2.14 HAL_TIMER_SetPerioduS()

```
void HAL_TIMER.SetPerioduS (
    HAL_TIMER.x timer_x,
    uint32_t us )
```

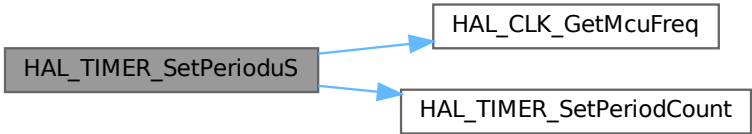
设置定时器周期(us)

参数

<i>timer</i> ↔ <i>_x</i>	定时器编号
<i>us</i>	定时器周期(us)

在文件 [hal_lks32mc08x.timer.c](#) 第 116 行定义.

函数调用图:



4.53.2.15 HAL_TIMER_SetPwm()

```
void HAL_TIMER_SetPwm (
    HAL_TIMER_x timer_x,
    int32_t t0,
    int32_t t1 )
```

在文件 [hal_lks32mc08x.timer.c](#) 第 221 行定义.

4.53.2.16 HAL_TIMER_SetPwmQ15()

```
void HAL_TIMER_SetPwmQ15 (
    HAL_TIMER_x timer_x,
    int16_t t0,
    int16_t t1 )
```

在文件 [hal_lks32mc08x.timer.c](#) 第 227 行定义.

函数调用图:



4.53.2.17 HAL_TIMER_SetWorkMode()

```
void HAL_TIMER_SetWorkMode (
    HAL_TIMER_x timer_x,
    HAL_TIMER_MODE_x mode_x )
```

< 计数模式

< PWM输出模式

< PWM输入模式

在文件 [hal_lks32mc08x_timer.c](#) 第 236 行定义.

4.53.2.18 HAL_TIMER_StartCounting()

```
void HAL_TIMER_StartCounting (
    HAL_TIMER_x timer_x )
```

< TIMER模块0

< TIMER模块1

< TIMER模块2

< TIMER模块3

在文件 [hal_lks32mc08x_timer.c](#) 第 178 行定义.

4.53.2.19 HAL_TIMER_StopCounting()

```
void HAL_TIMER_StopCounting (
    HAL_TIMER_x timer_x )
```

< TIMER模块0

< TIMER模块1

< TIMER模块2

< TIMER模块3

在文件 [hal_lks32mc08x_timer.c](#) 第 199 行定义.

4.54 hal_lksmcu_timer.h

浏览该文件的文档.

```

00001 #ifndef __HAL_LKSMCU_TIMER_H__
00002 #define __HAL_LKSMCU_TIMER_H__
00003 #include "stdint.h"
00004 #include "hal_lksmcu_gpio.h"
00005
00006 typedef enum
00007 {
00008     HAL_TIMER_0,
00009     HAL_TIMER_1,
00010     HAL_TIMER_2,
00011     HAL_TIMER_3,
00012     HAL_TIMER_4,
00013 } HAL_TIMER_x;
00014 typedef enum
00015 {
00016     HAL_TIMER_MODE_COUNT,
00017     HAL_TIMER_MODE_PWMOUT,
00018     HAL_TIMER_MODE_PWMIN,
00019 } HAL_TIMER_MODE_x;
00020
00021 void HAL_TIMER_Reset(HAL_TIMER_x); // 模块复位
00022 void HAL_TIMER_Enable(HAL_TIMER_x); // 模块打开
00023 void HAL_TIMER_Disable(HAL_TIMER_x); // 模块关闭
00024
00025 void HAL_TIMER_SetPeriodCount(HAL_TIMER_x, uint32_t count); // 设置周期(计数值)
00026 void HAL_TIMER_SetFreq(HAL_TIMER_x, uint32_t freq); // 设置频率
00027 void HAL_TIMER_SetPeriodms(HAL_TIMER_x, uint32_t ms); // 设置周期(ms)
00028 void HAL_TIMER_SetPeriodus(HAL_TIMER_x, uint32_t us); // 设置周期(us)
00029
00030 uint32_t HAL_TIMER_GetFreq(HAL_TIMER_x); // 获取当前的pwm频率
00031 uint32_t HAL_TIMER_GetThCnt(HAL_TIMER_x); // 获取计数器最大值
00032 void HAL_TIMER_SetClockDiv(HAL_TIMER_x timer_x, uint32_t div); // 设置时钟分频 div=1/2/4/8
00033 uint32_t HAL_TIMER_GetClockDiv(HAL_TIMER_x timer_x); // 获取时钟分频实际除数
00034 uint32_t HAL_TIMER_GetClockFreq(HAL_TIMER_x timer_x); // 获取分频后工作时钟(Hz)
00035 void HAL_TIMER_StartCounting(HAL_TIMER_x); // 开始计数
00036 void HAL_TIMER_StopCounting(HAL_TIMER_x); // 停止计数(同时计数器回到0)
00037 void HAL_TIMER_SetPwm(HAL_TIMER_x timer_x, int32_t t0, int32_t t1); // 设置pwm波形 仅PWM模式下有效
00038 void HAL_TIMER_SetPwmQ15(HAL_TIMER_x timer_x, int16_t t0, int16_t t1); // 设置pwm波形q15格式输入 仅PWM模
    式下有效
00039 void HAL_TIMER_SetWorkMode(HAL_TIMER_x timer_x, HAL_TIMER_MODE_x mode_x); // 设置工作模式
00040
00041 void HAL_TIMER_SetGpioOut(HAL_TIMER_x timer_x, HAL_GPIO_Px px);
00042 void HAL_TIMER_SetGpioIn(HAL_TIMER_x timer_x, HAL_GPIO_Px px);
00043 #endif

```

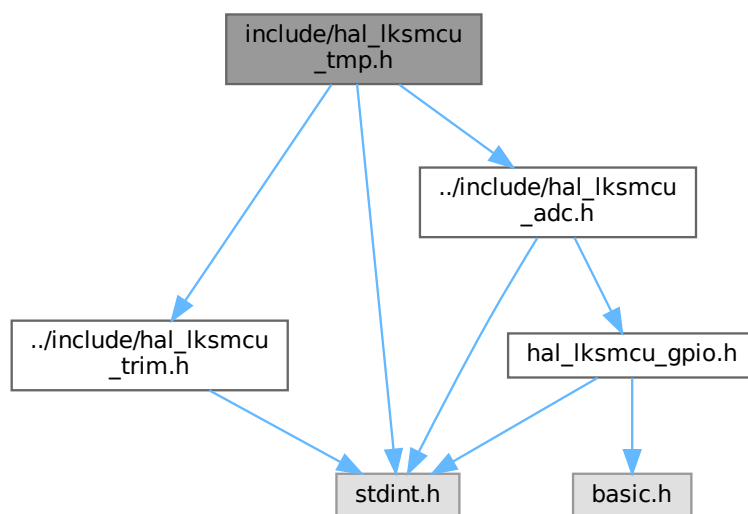
4.55 include/hal_lksmcu_tmp.h 文件参考

```

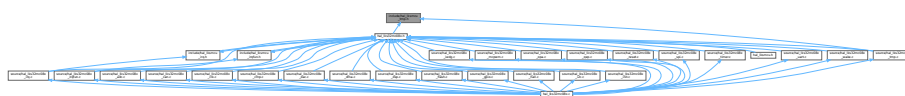
#include "../include/hal_lksmcu_trim.h"
#include "../include/hal_lksmcu_adc.h"
#include "stdint.h"

```


hal_lksmcu_tmp.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



枚举

- enum `HAL_TMP_x` { `HAL_TMP_0` , `HAL_TMP_1` }

函数

- void `HAL_TMP_Reset` (`HAL_TMP_x`)
模块复位
- void `HAL_TMP_Enable` (`HAL_TMP_x`)
模块打开
- void `HAL_TMP_Disable` (`HAL_TMP_x`)
模块关闭
- int32_t `HAL_TMP_GetTemperature` (`HAL_TMP_x`, `HAL_ADC_x`, `HAL_ADC_SAMP_SEQUENCE_x`)
获取当前芯片内部温度传感器测到的温度,单位摄氏度
- int32_t `HAL_TMP_GetTemperature01` (`HAL_TMP_x`, `HAL_ADC_x`, `HAL_ADC_SAMP_SEQUENCE_x`)
获取当前芯片内部温度传感器测到的温度,单位0.1摄氏度
- int32_t `HAL_TMP_Adc2Temperature` (`HAL_TMP_x`, int32_t `adc_value`)
`adc`值转温度
- int32_t `HAL_TMP_Adc2Temperature01` (`HAL_TMP_x`, int32_t `adc_value`)
`adc`值转温度

4.55.1 枚举类型说明

4.55.1.1 HAL_TMP_x

enum HAL_TMP_x

枚举值

HAL_↔ TMP_0	
HAL_↔ TMP_1	

在文件 [hal_lksmcu.tmp.h](#) 第 7 行定义.

4.55.2 函数说明

4.55.2.1 HAL_TMP_Adc2Temperature()

```
int32_t HAL_TMP_Adc2Temperature (
    HAL_TMP_x TMP_x,
    int32_t adc_value )
```

adc值转温度

参数

<i>TMP_x</i>	模块编号
<i>adc_value</i>	ADC值, 左对齐 (-32767 到32767)

返回

当前温度,单位摄氏度

在文件 [hal_lks32mc08x.tmp.c](#) 第 85 行定义.

函数调用图:



这是这个函数的调用关系图:



4.55.2.2 HAL_TMP_Adc2Temperature01()

```
int32_t HAL_TMP_Adc2Temperature01 (
    HAL_TMP_x TMP_x,
    int32_t adc_value )
```

adc值转温度

参数

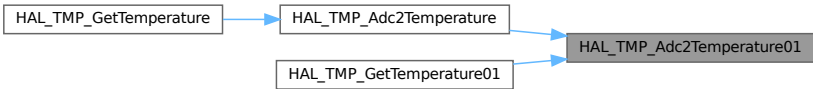
<i>TMP_x</i>	模块编号
<i>adc_value</i>	ADC值, 左对齐 (-32767 到32767)

返回

当前温度,单位0.1摄氏度

在文件 [hal_lks32mc08x.tmp.c](#) 第 96 行定义.

这是这个函数的调用关系图:



4.55.2.3 HAL_TMP_Disable()

```
void HAL_TMP_Disable (
    HAL_TMP_x TMP_x )
```

模块关闭

参数

<i>TMP</i> ↔ _X	模块编号

在文件 [hal_lks32mc08x.tmp.c](#) 第 37 行定义.

这是这个函数的调用关系图:



4.55.2.4 HAL_TMP.Enable()

```
void HAL_TMP_Enable (
    HAL_TMP_X TMP_X )
```

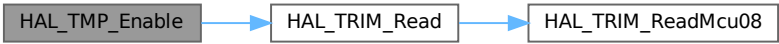
模块打开

参数

<i>TMP</i> ↔ _X	模块编号

在文件 [hal_lks32mc08x.tmp.c](#) 第 23 行定义.

函数调用图:



4.55.2.5 HAL_TMP.GetTemperature()

```
int32_t HAL_TMP_GetTemperature (
    HAL_TMP_X TMP_X,
    HAL_ADC_X adc,
    HAL_ADC_SAMP_SEQUENCE_X sequence )
```

获取当前芯片内部温度传感器测到的温度,单位摄氏度

参数

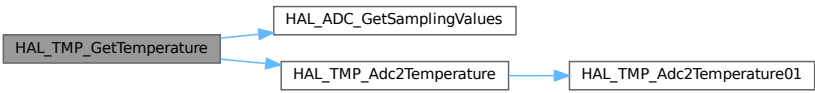
<i>TMP_x</i>	模块编号
<i>adc</i>	ADC编号
<i>sequence</i>	ADC采样序列

返回

当前温度,单位摄氏度

在文件 [hal_lks32mc08x.tmp.c](#) 第 52 行定义.

函数调用图:



4.55.2.6 HAL_TMP_GetTemperature01()

```
int32_t HAL_TMP_GetTemperature01 (
    HAL_TMP_x TMP_x,
    HAL_ADC_x adc,
    HAL_ADC_SAMP_SEQUENCE_x sequence )
```

获取当前芯片内部温度传感器测到的温度,单位0.1摄氏度

参数

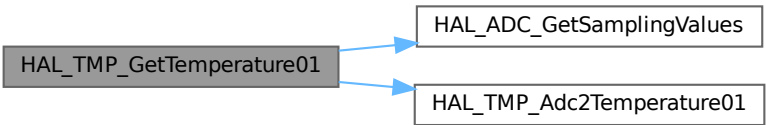
<i>TMP_x</i>	模块编号
<i>adc</i>	ADC编号
<i>sequence</i>	ADC采样序列

返回

当前温度,单位0.1摄氏度

在文件 [hal_lks32mc08x.tmp.c](#) 第 69 行定义.

函数调用图:



4.55.2.7 HAL_TMP_Reset()

```
void HAL_TMP_Reset (
    HAL_TMP_x TMP_x )
```

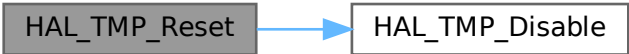
模块复位

参数

<i>TMP</i> ↔	模块编号
<i>_x</i>	

在文件 [hal_lks32mc08x.tmp.c](#) 第 14 行定义.

函数调用图:



4.56 hal_lksmcu.tmp.h

[浏览该文件的文档.](#)

```
00001
00002 #ifndef __HAL_LKSMCU_TMP_H__
00003 #define __HAL_LKSMCU_TMP_H__
00004 #include "../include/hal_lksmcu.trim.h"
00005 #include "../include/hal_lksmcu.adc.h"
00006 #include "stdint.h"
00007 typedef enum
00008 {
00009     HAL_TMP_0,
00010     HAL_TMP_1,
00011 } HAL_TMP_x;
00012 // 初始化
00013 void HAL_TMP_Reset (HAL_TMP_x); // 模块复位
00014 void HAL_TMP_Enable (HAL_TMP_x); // 模块打开
```

```

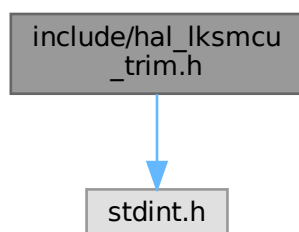
00015 void HAL_TMP_Disable(HAL_TMP_x); // 模块关闭
00016
00017 int32_t HAL_TMP_GetTemperature(HAL_TMP_x, HAL_ADC_x, HAL_ADC_SAMP_SEQUENCE_x); // 获取当前芯片内部温度传感器
测到的温度,单位摄氏度,需要配置ADC并采样温度传感器通道
00018 int32_t HAL_TMP_GetTemperature01(HAL_TMP_x, HAL_ADC_x, HAL_ADC_SAMP_SEQUENCE_x); // 获取当前芯片内部温度传感器
测到的温度,单位0.1摄氏度,需要配置ADC并采样温度传感器通道
00019
00020 int32_t HAL_TMP_Adc2Temperature(HAL_TMP_x, int32_t adc_value); // 芯片内部温度传感器ADC值转温度 单位度
00021 int32_t HAL_TMP_Adc2Temperature01(HAL_TMP_x, int32_t adc_value); // 芯片内部温度传感器ADC值转温度 单位0.1度
00022 #endif

```

4.57 include/hal_lksmcu_trim.h 文件参考

```
#include "stdint.h"
```

hal_lksmcu_trim.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



函数

- uint32_t **HAL_TRIM_Read** (uint32_t addr)
读取MCU03的校准参数
- uint32_t **HAL_TRIM_ReadId** (uint32_t idx)
读取芯片唯一ID
- uint32_t **HAL_TRIM_ReadIdCrc32** (void)
读取芯片唯一ID的CRC32值

4.57.1 函数说明

4.57.1.1 HAL_TRIM_Read()

```
uint32_t HAL_TRIM_Read (
    uint32_t addr )
```

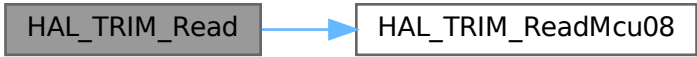
读取MCU03的校准参数

参数

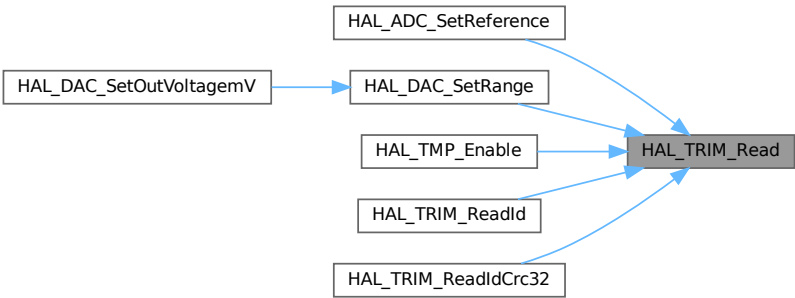
<i>addr</i>	校准参数的地址

在文件 [hal_lks32mc08x.trim.c](#) 第 64 行定义.

函数调用图:



这是这个函数的调用关系图:



4.57.1.2 HAL_TRIM_ReadId()

```
uint32_t HAL_TRIM_ReadId (
    uint32_t idx )
```

读取芯片唯一ID

参数

<i>idx</i>	0:ID0,1:ID1,2:ID2,3:ID3,4:ID4

返回

唯一ID

在文件 [hal_lks32mc08x.trim.c](#) 第 74 行定义.

函数调用图:



4.57.1.3 HAL_TRIM_ReadIdCrc32()

```
uint32_t HAL_TRIM_ReadIdCrc32 (
    void )
```

读取芯片唯一ID的CRC32值

返回

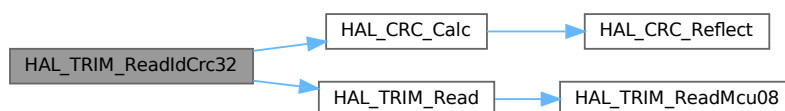
唯一ID的CRC32值

注解

相较于HAL_TRIM_ReadId，更方便存储和判断，但理论上存在重复的可能（1/0xffffffff）

在文件 [hal_lks32mc08x.trim.c](#) 第 84 行定义.

函数调用图:



4.58 hal_lksmcu_trim.h

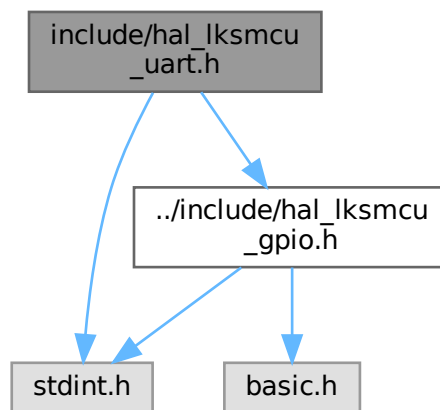
[浏览该文件的文档.](#)

```

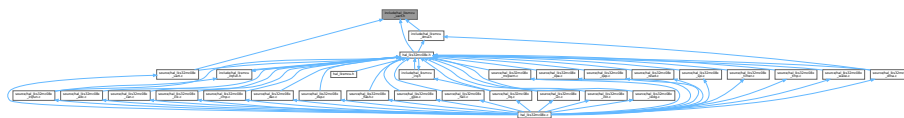
00001
00002 #ifndef __HAL_LKSMCU_TRIM_H__
00003 #define __HAL_LKSMCU_TRIM_H__
00004
00005 #include "stdint.h"
00006 uint32_t HAL_TRIM_Read(uint32_t addr);
00007 uint32_t HAL_TRIM_ReadId(uint32_t idx);
00008 uint32_t HAL_TRIM_ReadIdCrc32(void);
00009
00010 #endif
```

4.59 include/hal_lksmcu_uart.h 文件参考

```
#include "stdint.h"
#include "../include/hal_lksmcu_gpio.h"
hal_lksmcu_uart.h 的引用(Include)关系图:
```



此图展示该文件直接或间接的被哪些文件引用了:



枚举

- enum HAL_UART_x { HAL_UART_0 , HAL_UART_1 }
- enum HAL_UART_MODE_x {
HAL_UART_MODE_LSB_STOP1_NONE , HAL_UART_MODE_LSB_STOP2_NONE , HAL_UART_MODE_LSB_STOP1_ODD ,
HAL_UART_MODE_LSB_STOP2_ODD ,
HAL_UART_MODE_LSB_STOP1_EVEN , HAL_UART_MODE_LSB_STOP2_EVEN , HAL_UART_MODE_MSB_STOP1_NONE ,
HAL_UART_MODE_MSB_STOP2_NONE ,
HAL_UART_MODE_MSB_STOP1_ODD , HAL_UART_MODE_MSB_STOP2_ODD , HAL_UART_MODE_MSB_STOP1_EVEN ,
HAL_UART_MODE_MSB_STOP2_EVEN }
- enum HAL_UART_POL_x { HAL_UART_POL_RX_POS_TX_POS , HAL_UART_POL_RX_POS_TX_NEG ,
HAL_UART_POL_RX_NEG_TX_POS , HAL_UART_POL_RX_NEG_TX_NEG }

函数

- void HAL_UART_Reset (HAL_UART_x)
- void HAL_UART_Enable (HAL_UART_x)
- void HAL_UART_Disable (HAL_UART_x)

- void HAL_UART_SetBaud (HAL_UART_x, uint32_t baud)
- void HAL_UART_SetClockDiv (HAL_UART_x UART_x, uint32_t div)
- uint32_t HAL_UART_GetClockDiv (HAL_UART_x UART_x)
- uint32_t HAL_UART_GetClockFreq (HAL_UART_x UART_x)
- uint32_t HAL_UART_GetBaud (HAL_UART_x UART_x)
- void HAL_UART_TxData (HAL_UART_x, uint8_t *data, uint32_t len)
- void HAL_UART_SetMode (HAL_UART_x, HAL_UART_MODE_x)
- void HAL_UART_SetPolarity (HAL_UART_x, HAL_UART_POL_x)
- 设置IO极性
- uint8_t HAL_UART_ReadBuffer (HAL_UART_x)
- void HAL_UART_WriteBuffer (HAL_UART_x, uint8_t buff)
- void HAL_UART_SetGpioTx (HAL_UART_x, HAL_GPIO_Px)
- void HAL_UART_SetGpioRx (HAL_UART_x, HAL_GPIO_Px)

4.59.1 枚举类型说明

4.59.1.1 HAL_UART_MODE_x

enum HAL_UART_MODE_x

枚举值

HAL_UART_MODE_LSB.STOP1.NONE	低位先发 1位停止位 无校验
HAL_UART_MODE_LSB.STOP2.NONE	低位先发 2位停止位 无校验
HAL_UART_MODE_LSB.STOP1.ODD	低位先发 1位停止位 奇校验
HAL_UART_MODE_LSB.STOP2.ODD	低位先发 2位停止位 奇校验
HAL_UART_MODE_LSB.STOP1.EVEN	低位先发 1位停止位 偶校验
HAL_UART_MODE_LSB.STOP2.EVEN	低位先发 2位停止位 偶校验
HAL_UART_MODE_MSB.STOP1.NONE	高位先发 1位停止位 无校验
HAL_UART_MODE_MSB.STOP2.NONE	高位先发 2位停止位 无校验
HAL_UART_MODE_MSB.STOP1.ODD	高位先发 1位停止位 奇校验
HAL_UART_MODE_MSB.STOP2.ODD	高位先发 2位停止位 奇校验
HAL_UART_MODE_MSB.STOP1.EVEN	高位先发 1位停止位 偶校验
HAL_UART_MODE_MSB.STOP2.EVEN	高位先发 2位停止位 偶校验

在文件 `hal_lksmcu_uart.h` 第 14 行定义.

4.59.1.2 HAL_UART_POL_x

enum HAL_UART_POL_x

枚举值

HAL_UART_POL_RX.POS.TX_POS	
HAL_UART_POL_RX.POS.TX_NEG	
HAL_UART_POL_RX.NEG.TX_POS	
HAL_UART_POL_RX.NEG.TX_NEG	

在文件 `hal_lksmcu_uart.h` 第 30 行定义.

4.59.1.3 HAL_UART_x

enum `HAL_UART_x`

枚举值

HAL_↔ UART_0	
HAL_↔ UART_1	

在文件 `hal_lksmcu_uart.h` 第 6 行定义.

4.59.2 函数说明

4.59.2.1 HAL_UART_Disable()

```
void HAL_UART_Disable (
    HAL_UART_x UART_x )
```

在文件 `hal_lks32mc08x_uart.c` 第 64 行定义.

4.59.2.2 HAL_UART_Enable()

```
void HAL_UART_Enable (
    HAL_UART_x UART_x )
```

在文件 `hal_lks32mc08x_uart.c` 第 39 行定义.

4.59.2.3 HAL_UART_GetBaud()

```
uint32_t HAL_UART_GetBaud (
    HAL_UART_x UART_x )
```

在文件 `hal_lks32mc08x_uart.c` 第 158 行定义.

函数调用图:

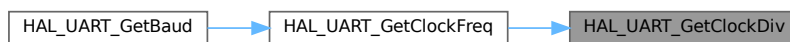


4.59.2.4 HAL_UART_GetClockDiv()

```
uint32_t HAL_UART_GetClockDiv (
    HAL_UART_x UART_x )
```

在文件 [hal_lks32mc08x_uart.c](#) 第 145 行定义.

这是这个函数的调用关系图:

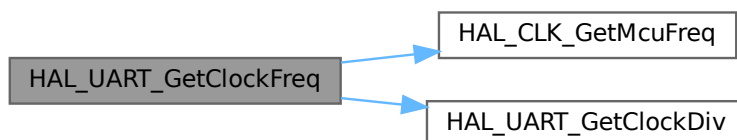


4.59.2.5 HAL_UART_GetClockFreq()

```
uint32_t HAL_UART_GetClockFreq (
    HAL_UART_x UART_x )
```

在文件 [hal_lks32mc08x_uart.c](#) 第 152 行定义.

函数调用图:



这是这个函数的调用关系图:



4.59.2.6 HAL_UART_ReadBuffer()

```
uint8_t HAL_UART_ReadBuffer (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_uart.c](#) 第 279 行定义.

4.59.2.7 HAL_UART_Reset()

```
void HAL_UART_Reset (
    HAL_UART_x UART_x )
```

在文件 [hal_lks32mc08x_uart.c](#) 第 12 行定义.

4.59.2.8 HAL_UART_SetBaud()

```
void HAL_UART_SetBaud (
    HAL_UART_x UART_x,
    uint32_t baud )
```

在文件 [hal_lks32mc08x_uart.c](#) 第 80 行定义.

函数调用图:



4.59.2.9 HAL_UART_SetClockDiv()

```
void HAL_UART_SetClockDiv (
    HAL_UART_x UART_x,
    uint32_t div )
```

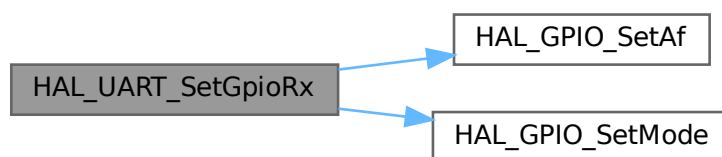
在文件 [hal_lks32mc08x_uart.c](#) 第 135 行定义.

4.59.2.10 HAL_UART_SetGpioRx()

```
void HAL_UART_SetGpioRx (
    HAL_UART_x uart_x,
    HAL_GPIO_Px px )
```

在文件 [hal_lks32mc08x_uart.c](#) 第 293 行定义.

函数调用图:

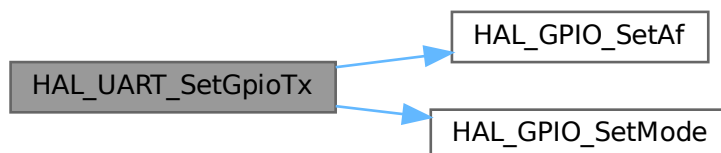


4.59.2.11 HAL_UART_SetGpioTx()

```
void HAL_UART_SetGpioTx (
    HAL_UART_<x> uart_<x>,
    HAL_GPIO_P<x> p<x> )
```

在文件 [hal_lks32mc08x_uart.c](#) 第 287 行定义.

函数调用图:



4.59.2.12 HAL_UART_SetMode()

```
void HAL_UART_SetMode (
    HAL_UART_<x> uart_<x>,
    HAL_UART_MODE_<x> mode_<x> )
```

- < 低位先发 1位停止位 无校验
- < 低位先发 2位停止位 无校验
- < 低位先发 1位停止位 奇校验
- < 低位先发 2位停止位 奇校验
- < 低位先发 1位停止位 偶校验
- < 低位先发 2位停止位 偶校验
- < 高位先发 1位停止位 无校验
- < 高位先发 2位停止位 无校验
- < 高位先发 1位停止位 奇校验
- < 高位先发 2位停止位 奇校验
- < 高位先发 1位停止位 偶校验
- < 高位先发 2位停止位 偶校验

在文件 [hal_lks32mc08x_uart.c](#) 第 178 行定义.

4.59.2.13 HAL_UART_SetPolarity()

```
void HAL_UART_SetPolarity (
    HAL_UART_<x> uart_<x>,
    HAL_UART_POL_<x> pol )
```

设置IO极性

参数

<i>uart_x</i>	UART模块编号
<i>pol</i>	极性配置

在文件 [hal_lks32mc08x_uart.c](#) 第 255 行定义。

4.59.2.14 HAL_UART_TxData()

```
void HAL_UART_TxData (
    HAL_UART_x UART_x,
    uint8_t * data,
    uint32_t len )
```

在文件 [hal_lks32mc08x_uart.c](#) 第 166 行定义。

4.59.2.15 HAL_UART_WriteBuffer()

```
void HAL_UART_WriteBuffer (
    HAL_UART_x uart_x,
    uint8_t buff )
```

在文件 [hal_lks32mc08x_uart.c](#) 第 283 行定义。

4.60 hal_lksmcu_uart.h

[浏览该文件的文档。](#)

```
00001
00002 #ifndef __HAL_LKSMCU_UART_H__
00003 #define __HAL_LKSMCU_UART_H__
00004 #include "stdint.h"
00005 #include "../include/hal_lksmcu_gpio.h"
00006 typedef enum
00007 {
00008     HAL_UART_0,
00009     HAL_UART_1,
00010 #ifdef HAL_TEST_NOT_SUPPORTED
00011     HAL_UART_2,
00012 #endif
00013 } HAL_UART_x;
00014 typedef enum
00015 {
00016     HAL_UART_MODE_LSB_STOP1_NONE,
00017     HAL_UART_MODE_LSB_STOP2_NONE,
00018     HAL_UART_MODE_LSB_STOP1_ODD,
00019     HAL_UART_MODE_LSB_STOP2_ODD,
00020     HAL_UART_MODE_LSB_STOP1_EVEN,
00021     HAL_UART_MODE_LSB_STOP2_EVEN,
00022     HAL_UART_MODE_MSB_STOP1_NONE,
00023     HAL_UART_MODE_MSB_STOP2_NONE,
00024     HAL_UART_MODE_MSB_STOP1_ODD,
00025     HAL_UART_MODE_MSB_STOP2_ODD,
00026     HAL_UART_MODE_MSB_STOP1_EVEN,
00027     HAL_UART_MODE_MSB_STOP2_EVEN,
00028 } HAL_UART_MODE_x;
00029
00030 typedef enum
00031 {
00032     HAL_UART_POL_RX_POS_TX_POS, // RX正逻辑 TX正逻辑 复位后的默认值
00033     HAL_UART_POL_RX_POS_TX_NEG, // RX正逻辑 TX负逻辑
00034     HAL_UART_POL_RX_NEG_TX_POS, // RX负逻辑 TX正逻辑
```



```

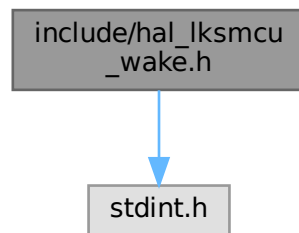
00035     HAL_UART_POL_RX_NEG_TX_NEG // RX负逻辑 TX负逻辑
00036 } HAL_UART_POL_x;
00037 // 初始化
00038 void HAL_UART_Reset(HAL_UART_x); // 模块复位
00039 void HAL_UART_Enable(HAL_UART_x); // 模块打开
00040 void HAL_UART_Disable(HAL_UART_x); // 模块关闭
00041 //
00042 void HAL_UART_SetBaud(HAL_UART_x, uint32_t baud);
00043 void HAL_UART_SetClockDiv(HAL_UART_x UART_x, uint32_t div); // 设置UART全局工作时钟分频 div=实际除
数(UART0/1共享)
00044 uint32_t HAL_UART_GetClockDiv(HAL_UART_x UART_x); // 获取UART工作时钟分频实际除数
00045 uint32_t HAL_UART_GetClockFreq(HAL_UART_x UART_x); // 获取UART工作时钟频率(Hz)
00046 uint32_t HAL_UART_GetBaud(HAL_UART_x UART_x); // 获取当前波特率
00047 void HAL_UART_TxData(HAL_UART_x, uint8_t *data, uint32_t len);
00048 void HAL_UART_SetMode(HAL_UART_x, HAL_UART_MODE_x);
00049 void HAL_UART_SetPolarity(HAL_UART_x, HAL_UART_POL_x);
00050
00051 uint8_t HAL_UART_ReadBuffer(HAL_UART_x);
00052 void HAL_UART_WriteBuffer(HAL_UART_x, uint8_t buff);
00053 void HAL_UART_SetGpioTx(HAL_UART_x, HAL_GPIO_Px); // 设置用于输出的gpio
00054 void HAL_UART_SetGpioRx(HAL_UART_x, HAL_GPIO_Px); // 设置用于输出的gpio
00055
00056 #endif

```

4.61 include/hal_lksmcu_wake.h 文件参考

#include "stdint.h"

hal_lksmcu_wake.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



枚举

- enum HAL_WAKE_UP_IO_x {
 HAL_WAKE_UP_IO_P0_0_Low = 0x0001, HAL_WAKE_UP_IO_P0_0_High = 0x0101, HAL_WAKE_UP_IO_P0_1_Low = 0x0002, HAL_WAKE_UP_IO_P0_1_High = 0x0102,
 HAL_WAKE_UP_IO_P1_0_Low = 0x0004, HAL_WAKE_UP_IO_P1_0_High = 0x0104, HAL_WAKE_UP_IO_P1_1_Low = 0x0008, HAL_WAKE_UP_IO_P1_1_High = 0x0108 }

函数

- void HAL_WAKE_DeepSleepIoWake (HAL_WAKE_UP_IO_x)
进入深度休眠,并且使用IO唤醒
- void HAL_WAKE_DeepSleepTimeWakeUp (uint32_t ms)
进入深度休眠,并且使用定时唤醒
- void HAL_WAKE_DeepSleepIoTimeWakeUp (HAL_WAKE_UP_IO_x, uint32_t ms)
进入深度休眠,并且使用IO唤醒和定时唤醒

4.61.1 枚举类型说明

4.61.1.1 HAL_WAKE_UP_IO_x

enum HAL_WAKE_UP_IO_x

枚举值

HAL_WAKE_UP_IO_P0_0_Low	
HAL_WAKE_UP_IO_P0_0_High	
HAL_WAKE_UP_IO_P0_1_Low	
HAL_WAKE_UP_IO_P0_1_High	
HAL_WAKE_UP_IO_P1_0_Low	
HAL_WAKE_UP_IO_P1_0_High	
HAL_WAKE_UP_IO_P1_1_Low	
HAL_WAKE_UP_IO_P1_1_High	

在文件 [hal_lksmcu_wake.h](#) 第 5 行定义.

4.61.2 函数说明

4.61.2.1 HAL_WAKE_DeepSleepIoTimeWakeUp()

```
void HAL_WAKE_DeepSleepIoTimeWakeUp (
    HAL_WAKE_UP_IO_x wake_up_io,
    uint32_t ms )
```

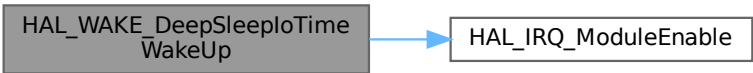
进入深度休眠,并且使用IO唤醒和定时唤醒

参数

<i>wake_up_</i> <i>_io</i>	指定唤醒IO引脚及其电平
<i>ms</i>	指定定时唤醒的时间,单位为毫秒

在文件 [hal_lks32mc08x_wake.c](#) 第 155 行定义.

函数调用图:



4.61.2.2 HAL_WAKE_DeepSleepIoWake()

```
void HAL_WAKE_DeepSleepIoWake (
    HAL_WAKE_UP_IO_x wake_up_io )
```

进入深度休眠,并且使用IO唤醒

参数

<i>wake_up_io</i>	指定唤醒IO引脚及其电平

在文件 [hal_lks32mc08x.wake.c](#) 第 12 行定义.

函数调用图:



4.61.2.3 HAL_WAKE_DeepSleepTimeWakeUp()

```
void HAL_WAKE_DeepSleepTimeWakeUp (
    uint32_t ms )
```

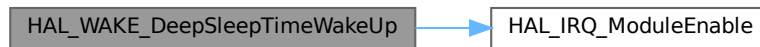
进入深度休眠,并且使用定时唤醒

参数

<i>ms</i>	指定定时唤醒的时间,单位为毫秒

在文件 [hal_lks32mc08x.wake.c](#) 第 62 行定义.

函数调用图:



4.62 hal_lksmcu_wake.h

[浏览该文件的文档.](#)

```

00001
00002 #ifndef __HAL_LKSMCU_WEAK_H__
00003 #define __HAL_LKSMCU_WEAK_H__
00004 #include "stdint.h"
00005 typedef enum
00006 {
00007
00008     HAL_WAKE_UP_IO_P0_0_Low = 0x0001,
00009     HAL_WAKE_UP_IO_P0_0_High = 0x0101,
00010     HAL_WAKE_UP_IO_P0_1_Low = 0x0002,
00011     HAL_WAKE_UP_IO_P0_1_High = 0x0102,
00012     HAL_WAKE_UP_IO_P1_0_Low = 0x0004,
00013     HAL_WAKE_UP_IO_P1_0_High = 0x0104,
00014     HAL_WAKE_UP_IO_P1_1_Low = 0x0008,
00015     HAL_WAKE_UP_IO_P1_1_High = 0x0108,
00016
00017 #ifdef HAL_TEST_NOT_SUPPORTED
00018     HAL_WAKE_UP_IO_P0_2_Low, // 不支持
00019     HAL_WAKE_UP_IO_P0_2_High, // 不支持
00020     HAL_WAKE_UP_IO_P0_8_Low, // 不支持
00021     HAL_WAKE_UP_IO_P0_8_High, // 不支持
00022     HAL_WAKE_UP_IO_P0_9_Low, // 不支持
00023     HAL_WAKE_UP_IO_P0_9_High, // 不支持
00024     HAL_WAKE_UP_IO_P0_10_Low, // 不支持
00025     HAL_WAKE_UP_IO_P0_10_High, // 不支持
00026     HAL_WAKE_UP_IO_P1_5_Low, // 不支持
00027     HAL_WAKE_UP_IO_P1_5_High, // 不支持
00028     HAL_WAKE_UP_IO_P1_8_Low, // 不支持
00029     HAL_WAKE_UP_IO_P1_8_High, // 不支持
00030     HAL_WAKE_UP_IO_P1_9_Low, // 不支持
00031     HAL_WAKE_UP_IO_P1_9_High, // 不支持
00032
00033     HAL_WAKE_UP_IO_P0_6_Low, // 不支持
00034     HAL_WAKE_UP_IO_P0_6_High, // 不支持
00035     HAL_WAKE_UP_IO_P0_11_Low, // 不支持
00036     HAL_WAKE_UP_IO_P0_11_High, // 不支持
00037     HAL_WAKE_UP_IO_P0_14_Low, // 不支持
00038     HAL_WAKE_UP_IO_P0_14_High, // 不支持
00039     HAL_WAKE_UP_IO_P2_4_Low, // 不支持
00040     HAL_WAKE_UP_IO_P2_4_High, // 不支持
00041     HAL_WAKE_UP_IO_P2_7_Low, // 不支持
00042     HAL_WAKE_UP_IO_P2_7_High, // 不支持
00043     HAL_WAKE_UP_IO_P2_15_Low, // 不支持
00044     HAL_WAKE_UP_IO_P2_15_High, // 不支持
00045     HAL_WAKE_UP_IO_CLU0_Low, // 不支持
00046     HAL_WAKE_UP_IO_CLU0_High, // 不支持
00047     HAL_WAKE_UP_IO_CLU1_Low, // 不支持
00048     HAL_WAKE_UP_IO_CLU1_High, // 不支持
00049     HAL_WAKE_UP_IO_CLU2_Low, // 不支持
00050     HAL_WAKE_UP_IO_CLU2_High, // 不支持
00051     HAL_WAKE_UP_IO_CLU3_Low, // 不支持
00052     HAL_WAKE_UP_IO_CLU3_High, // 不支持
00053     HAL_WAKE_UP_IO_P0_12_Low, // 不支持
00054     HAL_WAKE_UP_IO_P0_12_High, // 不支持
00055     HAL_WAKE_UP_IO_P0_13_Low, // 不支持
00056     HAL_WAKE_UP_IO_P0_13_High, // 不支持
00057     HAL_WAKE_UP_IO_P2_0_Low, // 不支持
00058     HAL_WAKE_UP_IO_P2_0_High, // 不支持
00059     HAL_WAKE_UP_IO_P0_3_Low, // 不支持
00060     HAL_WAKE_UP_IO_P0_3_High, // 不支持
00061     HAL_WAKE_UP_IO_P0_4_Low, // 不支持
00062     HAL_WAKE_UP_IO_P0_4_High, // 不支持
00063     HAL_WAKE_UP_IO_P0_5_Low, // 不支持
  
```

```
00064     HAL_WAKE_UP_IO_P0_5_High, // 不支持
00065     HAL_WAKE_UP_IO_P1_11_Low, // 不支持
00066     HAL_WAKE_UP_IO_P1_11_High, // 不支持
00067     HAL_WAKE_UP_IO_P2_14_Low, // 不支持
00068     HAL_WAKE_UP_IO_P2_14_High, // 不支持
00069     HAL_WAKE_UP_IO_P3_2_Low, // 不支持
00070     HAL_WAKE_UP_IO_P3_2_High, // 不支持
00071 #endif
00072
00073 } HAL_WAKE_UP_IO_x;
00074
00075 void HAL_WAKE_DeepSleepIoWake(HAL_WAKE_UP_IO_x);
00076 void HAL_WAKE_DeepSleepTimeWakeUp(uint32_t ms);
00077 void HAL_WAKE_DeepSleepIoTimeWakeUp(HAL_WAKE_UP_IO_x, uint32_t ms);
00078 #endif
```

4.63 readme.md 文件参考

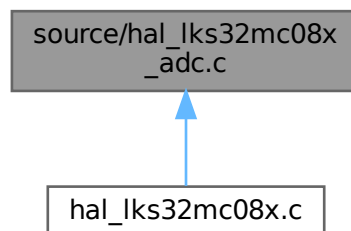
4.64 source/hal_lks32mc08x_adc.c 文件参考

lks32mc08x

```
#include "../include/hal_lksmcu_adc.h"
#include "../hal_lks32mc08x.h"
#include "lks32mc08x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc08x_reg.h"
hal_lks32mc08x_adc.c 的引用(Include)关系图:
```



此图展示该文件直接或间接的被哪些文件引用了:



函数

- static void [HAL_ADC_UpdateTrig](#) ([HAL_ADC_x](#) adc_x)
内部使用,更新`HAL_ADC.TRIG.STATE`描述的触发配置
- void [HAL_ADC_Reset](#) ([HAL_ADC_x](#) adc_x)
初始化指定的ADC模块
- void [HAL_ADC_Enable](#) ([HAL_ADC_x](#) adc_x)
启用指定的ADC模块
- void [HAL_ADC_Disable](#) ([HAL_ADC_x](#) adc_x)
禁用指定的ADC模块
- int16_t [HAL_ADC_GetSamplingValues](#) ([HAL_ADC_x](#) adc_x, [HAL_ADC_SAMP_SEQUENCE_x](#) sequence_x)
获取指定序列号的采样值
- void [HAL_ADC_SetSamplingChannel](#) ([HAL_ADC_x](#) adc_x, [HAL_ADC_SAMP_SEQUENCE_x](#) sequence_x, [HAL_ADC_CHANNEL_x](#) channel_x)
设置指定序列号的采样通道
- void [HAL_ADC_SetSamplingQuantity](#) ([HAL_ADC_x](#) adc_x, [HAL_ADC_SAMP_NUMBER_x](#) sample1_number, [HAL_ADC_SAMP_NUMBER_x](#) sample2_number)
设置采样数量
- void [HAL_ADC_SetTriggerSignal](#) ([HAL_ADC_x](#) adc_x, [HAL_ADC_TRIGGER_x](#) trigger)
设置触发信号
- void [HAL_ADC_EndSampling](#) ([HAL_ADC_x](#) adc_x)
结束采样
- void [HAL_ADC_SoftTrigger](#) ([HAL_ADC_x](#) adc_x)
软件触发ADC开始采样
- int16_t [HAL_ADC_ManualSampling](#) ([HAL_ADC_x](#) adc_x, [HAL_ADC_CHANNEL_x](#) channel_x)
手动采样
- void [HAL_ADC_SetReference](#) ([HAL_ADC_x](#) adc_x, [HAL_ADC_REFERENCE_x](#) reference_x)
设置ADC基准源
- void [HAL_ADC_StateReset](#) ([HAL_ADC_x](#) adc_x)
重置ADC状态
- float [HAL_ADC_GetRange](#) ([HAL_ADC_x](#) adc_x)
获取adc量程,使用芯片供电作为adc基准时固定返回5.0
- float [HAL_ADC_GetValuesUnit](#) ([HAL_ADC_x](#) adc_x)
获取采样值单位
- void [HAL_ADC_SetGpioIn](#) ([HAL_ADC_x](#) adc_x, [HAL_GPIO_Px](#) pin_x)
设置指定的GPIO为ADC输入
- void [HAL_ADC_SetGpioTriggerOut](#) ([HAL_ADC_x](#) adc_x, [HAL_GPIO_Px](#) pin_x)
设置指定的GPIO为ADC触发信号输出

变量

- static volatile uint8_t [HAL_ADC_TRIG_STATE](#) = 0

4.64.1 详细描述

lks32mc08x

在文件 [hal_lks32mc08x_adc.c](#) 中定义.

4.64.2 函数说明

4.64.2.1 HAL_ADC_Disable()

```
void HAL_ADC_Disable (
    HAL_ADC_x adc_x )
```

禁用指定的ADC模块

参数

<i>adc</i> _x	要禁用的ADC模块编号

在文件 [hal_lks32mc08x_adc.c](#) 第 89 行定义.

4.64.2.2 HAL_ADC_Enable()

```
void HAL_ADC_Enable (
    HAL_ADC_x adc_x )
```

启用指定的ADC模块

参数

<i>adc</i> _x	要启用的ADC模块编号

在文件 [hal_lks32mc08x_adc.c](#) 第 74 行定义.

4.64.2.3 HAL_ADC_EndSampling()

```
void HAL_ADC_EndSampling (
    HAL_ADC_x adc_x )
```

结束采样

参数

<i>adc</i> _x	ADC模块编号

在文件 [hal_lks32mc08x_adc.c](#) 第 244 行定义.

这是这个函数的调用关系图:



4.64.2.4 HAL_ADC.GetRange()

```
float HAL_ADC_GetRange (
    HAL_ADC_x adc_x )
```

获取adc量程,使用芯片供电作为adc基准时固定返回5.0

参数

adc↔ _x	ADC模块编号

返回

ADC量程

在文件 [hal_lks32mc08x_adc.c](#) 第 350 行定义.

这是这个函数的调用关系图:



4.64.2.5 HAL_ADC.GetSamplingValues()

```
int16_t HAL_ADC_GetSamplingValues (
    HAL_ADC_x adc_x,
    HAL_ADC_SAMP_SEQUENCE_x sequence_x )
```

获取指定序列号的采样值

参数

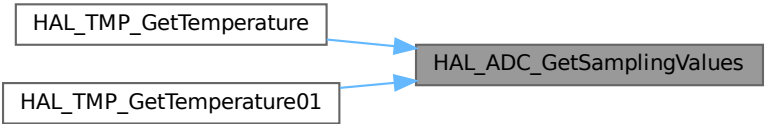
<i>adc_x</i>	ADC模块编号
<i>sequence_x</i>	采样序列号

返回

返回对应序列号的采样值

在文件 [hal_lks32mc08x_adc.c](#) 第 106 行定义.

这是这个函数的调用关系图:



4.64.2.6 HAL_ADC_GetValuesUnit()

```
float HAL_ADC_GetValuesUnit (
    HAL_ADC_x adc_x )
```

获取采样值单位

参数

<i>adc_x</i>	ADC模块编号

返回

采样值单位

在文件 [hal_lks32mc08x_adc.c](#) 第 381 行定义.

函数调用图:



4.64.2.7 HAL_ADC_ManualSampling()

```
int16_t HAL_ADC_ManualSampling (
    HAL_ADC_x adc_x,
    HAL_ADC_CHANNEL_x channel_x )
```

手动采样

参数

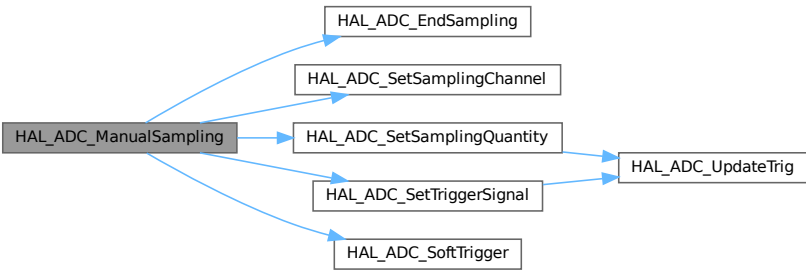
<i>n</i>	采样通道编号

返回

示例返回值,实际应返回采样值

在文件 [hal_lks32mc08x_adc.c](#) 第 263 行定义.

函数调用图:



4.64.2.8 HAL_ADC.Reset()

```
void HAL_ADC_Reset (
    HAL_ADC_x adc_x )
```

初始化指定的ADC模块

参数

<i>adc_x</i>	要初始化的ADC模块编号

在文件 [hal_lks32mc08x_adc.c](#) 第 65 行定义.

4.64.2.9 HAL_ADC.SetGpioIn()

```
void HAL_ADC_SetGpioIn (
    HAL_ADC_x adc_x,
    HAL_GPIO_Px pin_x )
```

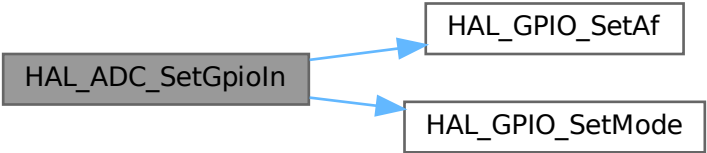
设置指定的GPIO为ADC输入

参数

<i>adc_x</i>	ADC模块编号
<i>pin_x</i>	GPIO编号

在文件 [hal_lks32mc08x_adc.c](#) 第 398 行定义.

函数调用图:



4.64.2.10 HAL_ADC.SetGpioTriggerOut()

```
void HAL_ADC_SetGpioTriggerOut (
    HAL_ADC_x adc_x,
    HAL_GPIO_Px pin_x )
```

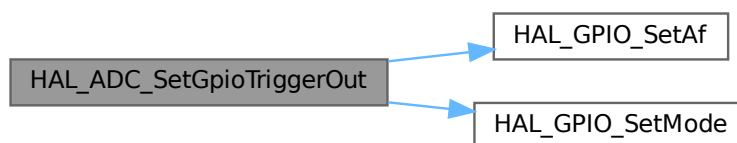
设置指定的GPIO为ADC触发信号输出

参数

<i>adc</i> ↔ _x	ADC模块编号
<i>pin</i> ↔ _x	GPIO编号

在文件 [hal_lks32mc08x_adc.c](#) 第 410 行定义.

函数调用图:



4.64.2.11 HAL_ADC_SetReference()

```
void HAL_ADC_SetReference (
    HAL_ADC_x adc_x,
    HAL_ADC_REFERENCE_x reference_x )
```

设置ADC基准源

参数

<i>adc_x</i>	ADC模块编号
<i>reference</i> ↔ _x	基准源

在文件 [hal_lks32mc08x_adc.c](#) 第 294 行定义.

函数调用图:



4.64.2.12 HAL_ADC_SetSamplingChannel()

```
void HAL_ADC_SetSamplingChannel (
    HAL_ADC_x adc_x,
    HAL_ADC_SAMP_SEQUENCE_x sequence_x,
    HAL_ADC_CHANNEL_x channel_x )
```

设置指定序列号的采样通道

参数

<i>adc_x</i>	ADC模块编号
<i>sequence_x</i>	采样序列号
<i>channel_x</i>	采样通道端口

在文件 [hal_lks32mc08x_adc.c](#) 第 117 行定义.

这是这个函数的调用关系图:



4.64.2.13 HAL_ADC_SetSamplingQuantity()

```
void HAL_ADC_SetSamplingQuantity (
    HAL_ADC_x adc_x,
    HAL_ADC_SAMP_NUMBER_x sample1_number,
    HAL_ADC_SAMP_NUMBER_x sample2_number )
```

设置采样数量

参数

<i>adc_x</i>	ADC模块编号
<i>sample_number</i>	采样数量

在文件 [hal_lks32mc08x_adc.c](#) 第 189 行定义.

函数调用图:



这是这个函数的调用关系图:



4.64.2.14 HAL_ADC_SetTriggerSignal()

```
void HAL_ADC_SetTriggerSignal (
    HAL_ADC_x adc-x,
    HAL_ADC_TRIGGER_x trigger )
```

设置触发信号

参数

<i>adc-x</i>	ADC模块编号
<i>trigger</i>	触发信号类型

- < 使用MCPWM触发ADC采样
- < 使用TIMER触发ADC采样
- < 使用软件触发ADC采样

在文件 [hal_lks32mc08x_adc.c](#) 第 221 行定义.

函数调用图:



这是这个函数的调用关系图:



4.64.2.15 HAL_ADC_SoftTrigger()

```
void HAL_ADC_SoftTrigger (
    HAL_ADC_x adc_x )
```

软件触发ADC开始采样

参数

adc_x	ADC模块编号

在文件 [hal_lks32mc08x_adc.c](#) 第 253 行定义.

这是这个函数的调用关系图:



4.64.2.16 HAL_ADC_StateReset()

```
void HAL_ADC_StateReset (
    HAL_ADC_x adc_x )
```

重置ADC状态

参数

adc_x	ADC模块编号

在文件 [hal_lks32mc08x_adc.c](#) 第 340 行定义.

4.64.2.17 HAL_ADC_UpdateTrig()

```
static void HAL_ADC_UpdateTrig (  
    HAL_ADC_x adc_x ) [static]
```

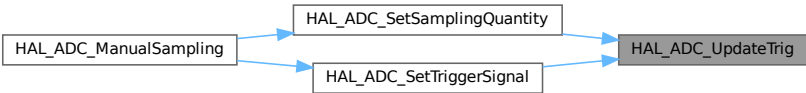
内部使用,更新HAL_ADC_TRIG_STATE描述的触发配置

参数

<code>adc_x</code>	使用的ADC模块编号

在文件 `hal_lks32mc08x_adc.c` 第 17 行定义.

这是这个函数的调用关系图:



4.64.3 变量说明

4.64.3.1 HAL_ADC_TRIG_STATE

```
volatile uint8_t HAL_ADC_TRIG_STATE = 0 [static]
```

在文件 `hal_lks32mc08x_adc.c` 第 11 行定义.

4.65 hal_lks32mc08x_adc.c

浏览该文件的文档.

```
00001  
00005 #include "../include/hal_lksmcu_adc.h"  
00006 #include "../hal_lks32mc08x.h"  
00007 #include "lks32mc08x-user-manual.h"  
00008 #include "../include/hal_lksmcu_reg.h"  
00009 #include "../include/hal_lks32mc08x_reg.h"  
00010  
00011 static volatile uint8_t HAL_ADC_TRIG_STATE = 0; // bit2 mcpwm触发 bit3 timer触发 bit4 两段式触发使能  
00012  
00017 static void HAL_ADC_UpdateTrig(HAL_ADC_x adc_x)  
00018 {  
00019     uint32_t mask = ADC0_TRIG_TRG_MODE_MASK |  
00020                     ADC0_TRIG_UTIMER3_CMP1_E_BIT |  
00021                     ADC0_TRIG_UTIMER3_CMP0_E_BIT |  
00022                     ADC0_TRIG_UTIMER2_CMP1_E_BIT |  
00023                     ADC0_TRIG_UTIMER2_CMP0_E_BIT |  
00024                     ADC0_TRIG_MCPWM_TRG3_E_BIT |  
00025                     ADC0_TRIG_MCPWM_TRG2_E_BIT |  
00026                     ADC0_TRIG_MCPWM_TRG1_E_BIT |  
00027                     ADC0_TRIG_MCPWM_TRG0_E_BIT;  
00028     // bit2 mcpwm触发 bit3 timer触发 bit4 两段式触发使能  
00029     if (HAL_REG_READ_BIT(HAL_ADC_TRIG_STATE, HAL_ADC_TRIG_STATE_TWO_SEG_TRIG_EN))  
00030     {
```

```

00031         if (HAL_REG_READ_BIT(HAL_ADC_TRIG_STATE, HAL_ADC_TRIG_STATE_MCPWM_TRIG_EN)) // 两段 mcpwm触发
00032         {
00033             HAL_REG_WRITE_BIT(HAL_ADC[adc_x]->TRIG, mask, ADC0_TRIG_MCPWM_TRG0_E_BIT |
ADC0_TRIG_MCPWM_TRG1_E_BIT | ADC0_TRIG_TRG_MODE(1));
00034         }
00035         else if (HAL_REG_READ_BIT(HAL_ADC_TRIG_STATE, HAL_ADC_TRIG_STATE_TIMER_TRIG_EN)) // 两段 timer触发
00036         {
00037             HAL_REG_WRITE_BIT(HAL_ADC[adc_x]->TRIG, mask, ADC0_TRIG_UTIMER2_CMP0_E_BIT |
ADC0_TRIG_UTIMER2_CMP1_E_BIT | ADC0_TRIG_TRG_MODE(1));
00038         }
00039         else // 两段 软件触发
00040         {
00041             HAL_REG_WRITE_BIT(HAL_ADC[adc_x]->TRIG, mask, ADC0_TRIG_TRG_MODE(1));
00042         }
00043     }
00044     else
00045     {
00046         if (HAL_REG_READ_BIT(HAL_ADC_TRIG_STATE, HAL_ADC_TRIG_STATE_MCPWM_TRIG_EN)) // 一段 mcpwm触发
00047         {
00048             HAL_REG_WRITE_BIT(HAL_ADC[adc_x]->TRIG, mask, ADC0_TRIG_MCPWM_TRG0_E_BIT);
00049         }
00050         else if (HAL_REG_READ_BIT(HAL_ADC_TRIG_STATE, HAL_ADC_TRIG_STATE_TIMER_TRIG_EN)) // 一段 timer触发
00051         {
00052             HAL_REG_WRITE_BIT(HAL_ADC[adc_x]->TRIG, mask, ADC0_TRIG_UTIMER2_CMP0_E_BIT);
00053         }
00054         else // 一段 软件触发
00055         {
00056             HAL_REG_WRITE_BIT(HAL_ADC[adc_x]->TRIG, mask, 0);
00057         }
00058     }
00059 }
00060
00065 void HAL_ADC_Reset(HAL_ADC_x adc_x)
00066 {
00067     (void)adc_x;
00068 }
00069
00074 void HAL_ADC_Enable(HAL_ADC_x adc_x)
00075 {
00076     (void)adc_x;
00077     // 系统寄存器解锁
00078     HAL_REG_SYS_UNLOCK();
00079     // SYS_AFE_REG5[0] ADC 开关控制
00080     HAL_REG_SET(SYS_AFE_REG5, SYS_AFE_REG5_ADCPDN_BIT);
00081     // 系统寄存器上锁
00082     HAL_REG_SYS_LOCK();
00083 }
00084
00089 void HAL_ADC_Disable(HAL_ADC_x adc_x)
00090 {
00091     (void)adc_x;
00092     // 系统寄存器解锁
00093     HAL_REG_SYS_UNLOCK();
00094     // SYS_AFE_REG5[0] ADC 开关控制
00095     HAL_REG_RESET(SYS_AFE_REG5, SYS_AFE_REG5_ADCPDN_BIT);
00096     // 系统寄存器上锁
00097     HAL_REG_SYS_LOCK();
00098 }
00099
00106 int16_t HAL_ADC_GetSamplingValues(HAL_ADC_x adc_x, HAL_ADC_SAMP_SEQUENCE_x sequence_x)
00107 {
00108     return HAL_REG_READ(HAL_ADC[adc_x]->DAT[sequence_x]);
00109 }
00110
00117 void HAL_ADC_SetSamplingChannel(HAL_ADC_x adc_x, HAL_ADC_SAMP_SEQUENCE_x sequence_x, HAL_ADC_CHANNEL_x
channel_x)
00118 {
00119     switch (sequence_x)
00120     {
00121         case HAL_ADC_SAMP_SEQUENCE_0:
00122             HAL_REG_WRITE_BIT_MASK_POS(HAL_ADC[adc_x]->CHN[0], ADC0_CHN0_DS0, channel_x);
00123             break;
00124         case HAL_ADC_SAMP_SEQUENCE_1:
00125             HAL_REG_WRITE_BIT_MASK_POS(HAL_ADC[adc_x]->CHN[0], ADC0_CHN0_DS1, channel_x);
00126             break;
00127         case HAL_ADC_SAMP_SEQUENCE_2:
00128             HAL_REG_WRITE_BIT_MASK_POS(HAL_ADC[adc_x]->CHN[1], ADC0_CHN1_DS2, channel_x);
00129             break;
00130         case HAL_ADC_SAMP_SEQUENCE_3:
00131             HAL_REG_WRITE_BIT_MASK_POS(HAL_ADC[adc_x]->CHN[1], ADC0_CHN1_DS3, channel_x);
00132             break;
00133         case HAL_ADC_SAMP_SEQUENCE_4:
00134             HAL_REG_WRITE_BIT_MASK_POS(HAL_ADC[adc_x]->CHN[2], ADC0_CHN2_DS4, channel_x);
00135             break;
00136         case HAL_ADC_SAMP_SEQUENCE_5:
00137             HAL_REG_WRITE_BIT_MASK_POS(HAL_ADC[adc_x]->CHN[2], ADC0_CHN2_DS5, channel_x);
00138             break;

```

```

00139         case HAL_ADC_SAMP_SEQUENCE_6:
00140             HAL_REG_WRITE_BIT_MASK_POS(HAL_ADC[adc_x] -> CHN[3], ADC0_CHN3_DS6, channel_x);
00141             break;
00142         case HAL_ADC_SAMP_SEQUENCE_7:
00143             HAL_REG_WRITE_BIT_MASK_POS(HAL_ADC[adc_x] -> CHN[3], ADC0_CHN3_DS7, channel_x);
00144             break;
00145         case HAL_ADC_SAMP_SEQUENCE_8:
00146             HAL_REG_WRITE_BIT_MASK_POS(HAL_ADC[adc_x] -> CHN[4], ADC0_CHN4_DS8, channel_x);
00147             break;
00148         case HAL_ADC_SAMP_SEQUENCE_9:
00149             HAL_REG_WRITE_BIT_MASK_POS(HAL_ADC[adc_x] -> CHN[4], ADC0_CHN4_DS9, channel_x);
00150             break;
00151         case HAL_ADC_SAMP_SEQUENCE_10:
00152             HAL_REG_WRITE_BIT_MASK_POS(HAL_ADC[adc_x] -> CHN[5], ADC0_CHN5_DS10, channel_x);
00153             break;
00154         case HAL_ADC_SAMP_SEQUENCE_11:
00155             HAL_REG_WRITE_BIT_MASK_POS(HAL_ADC[adc_x] -> CHN[5], ADC0_CHN5_DS11, channel_x);
00156             break;
00157         case HAL_ADC_SAMP_SEQUENCE_12:
00158             HAL_REG_WRITE_BIT_MASK_POS(HAL_ADC[adc_x] -> CHN[6], ADC0_CHN6_DS12, channel_x);
00159             break;
00160         case HAL_ADC_SAMP_SEQUENCE_13:
00161             HAL_REG_WRITE_BIT_MASK_POS(HAL_ADC[adc_x] -> CHN[6], ADC0_CHN6_DS13, channel_x);
00162             break;
00163         case HAL_ADC_SAMP_SEQUENCE_14:
00164             HAL_REG_WRITE_BIT_MASK_POS(HAL_ADC[adc_x] -> CHN[7], ADC0_CHN7_DS14, channel_x);
00165             break;
00166         case HAL_ADC_SAMP_SEQUENCE_15:
00167             HAL_REG_WRITE_BIT_MASK_POS(HAL_ADC[adc_x] -> CHN[7], ADC0_CHN7_DS15, channel_x);
00168             break;
00169         case HAL_ADC_SAMP_SEQUENCE_16:
00170             HAL_REG_WRITE_BIT_MASK_POS(HAL_ADC[adc_x] -> CHN[8], ADC0_CHN8_DS16, channel_x);
00171             break;
00172         case HAL_ADC_SAMP_SEQUENCE_17:
00173             HAL_REG_WRITE_BIT_MASK_POS(HAL_ADC[adc_x] -> CHN[8], ADC0_CHN8_DS17, channel_x);
00174             break;
00175         case HAL_ADC_SAMP_SEQUENCE_18:
00176             HAL_REG_WRITE_BIT_MASK_POS(HAL_ADC[adc_x] -> CHN[9], ADC0_CHN9_DS18, channel_x);
00177             break;
00178         case HAL_ADC_SAMP_SEQUENCE_19:
00179             HAL_REG_WRITE_BIT_MASK_POS(HAL_ADC[adc_x] -> CHN[9], ADC0_CHN9_DS19, channel_x);
00180             break;
00181     }
00182 }
00183
00189 void HAL_ADC_SetSamplingQuantity(HAL_ADC_x adc_x, HAL_ADC_SAMP_NUMBER_x sample1_number,
    HAL_ADC_SAMP_NUMBER_x sample2_number)
00190 {
00191     // 第一段采样数量为0时, 芯片不进行采样
00192     if (sample1_number == 0)
00193     {
00194         sample2_number = HAL_ADC_SAMP_NUMBER_0;
00195     }
00196     // 两段采样数量之和不能大于10个
00197     if (sample1_number + sample2_number > 20)
00198     {
00199         sample2_number = (HAL_ADC_SAMP_NUMBER_x)(20 - sample1_number);
00200     }
00201
00202     HAL_REG_WRITE_BIT(HAL_ADC[adc_x] -> CHNT0, ADC0_CHNT0_S1_MASK, sample1_number);
00203     HAL_REG_WRITE_BIT(HAL_ADC[adc_x] -> CHNT0, ADC0_CHNT0_S2_MASK, sample2_number << 8);
00204
00205     if (sample2_number)
00206     {
00207         HAL_REG_SET(HAL_ADC_TRIG_STATE, HAL_ADC_TRIG_STATE_TWO_SEG_TRIG_EN);
00208     }
00209     else
00210     {
00211         HAL_REG_RESET(HAL_ADC_TRIG_STATE, HAL_ADC_TRIG_STATE_TWO_SEG_TRIG_EN);
00212     }
00213     HAL_ADC_UpdateTrig(adc_x);
00214 }
00215
00221 void HAL_ADC_SetTriggerSignal(HAL_ADC_x adc_x, HAL_ADC_TRIGGER_x trigger)
00222 {
00223     switch (trigger)
00224     {
00225         case HAL_ADC_TRIGGER_MCPWM:
00226             HAL_REG_WRITE_BIT(HAL_ADC_TRIG_STATE, HAL_ADC_TRIG_STATE_MCPWM_TRIG_EN |
    HAL_ADC_TRIG_STATE_TIMER_TRIG_EN, HAL_ADC_TRIG_STATE_MCPWM_TRIG_EN);
00227             break;
00228         case HAL_ADC_TRIGGER_TIMER:
00229             HAL_REG_WRITE_BIT(HAL_ADC_TRIG_STATE, HAL_ADC_TRIG_STATE_MCPWM_TRIG_EN |
    HAL_ADC_TRIG_STATE_TIMER_TRIG_EN, HAL_ADC_TRIG_STATE_TIMER_TRIG_EN);
00230             break;
00231         case HAL_ADC_TRIGGER_SOFT:
00232             HAL_REG_WRITE_BIT(HAL_ADC_TRIG_STATE, HAL_ADC_TRIG_STATE_MCPWM_TRIG_EN |

```

```

        HAL_ADC_TRIG_STATE_TIMER_TRIG_EN, 0);
00233         break;
00234     default:
00235         break;
00236     }
00237     HAL_ADC_UpdateTrig(adcx);
00238 }
00239
00244 void HAL_ADC_EndSampling(HAL_ADC_x adc_x)
00245 {
00246     HAL_REG_SET(HAL_ADC[adc_x] -> CFG, ADC0_CFG_FSM_RESET_BIT);
00247 }
00248
00253 void HAL_ADC_SoftTrigger(HAL_ADC_x adc_x)
00254 {
00255     HAL_REG_WRITE(HAL_ADC[adc_x] -> SWT, 0x5aa5);
00256 }
00257
00263 int16_t HAL_ADC_ManualSampling(HAL_ADC_x adc_x, HAL_ADC_CHANNEL_x channel_x) // 手动采样 需要先设置采样通道为软件触发
00264 {
00265     int16_t data;
00266     uint32_t i;
00267     HAL_ADC_SetTriggerSignal(adc_x, HAL_ADC_TRIGGER_SOFT);
00268     HAL_ADC_EndSampling(adc_x);
00269     HAL_ADC_SetSamplingQuantity(adc_x, HAL_ADC_SAMP_NUMBER_1, HAL_ADC_SAMP_NUMBER_0);
00270     HAL_ADC_SetSamplingChannel(adc_x, HAL_ADC_SAMP_SEQUENCE_0, channel_x);
00271     HAL_REG_WRITE(HAL_ADC[adc_x] -> IE, 0);
00272     HAL_REG_WRITE(HAL_ADC[adc_x] -> IF, 0xffff);
00273     HAL_ADC_SoftTrigger(adc_x);
00274     i = 3000;
00275     while (HAL_REG_READ(HAL_ADC[adc_x] -> IF) == 0)
00276     {
00277         i--;
00278         __NOP();
00279         if (i == 0)
00280         {
00281             break;
00282         }
00283     }
00284     HAL_REG_WRITE(HAL_ADC[adc_x] -> IF, 0xffff);
00285     data = HAL_REG_READ(HAL_ADC[adc_x] -> DAT[0]);
00286     return data;
00287 }
00288
00294 void HAL_ADC_SetReference(HAL_ADC_x adc_x, HAL_ADC_REFERENCE_x reference_x)
00295 {
00296     (void)adc_x;
00297     switch (reference_x)
00298     {
00299         case HAL_ADC_REFERENCE_INTERNAL: // 使用内部基准源作为adc基准
00300
00301             // 系统寄存器解锁
00302             HAL_REG_SYS_UNLOCK();
00303             // SYS_AFE_REG1[3] ADC 基准电压调节
00304             // SYS_AFE_REG1[4] 使用外部输入电源作为 ADC REF
00305             HAL_REG_WRITE_BIT(SYS_AFE_REG1, SYS_AFE_REG1_GAIN_REF_BIT | SYS_AFE_REG1_REF2VDD_BIT, 0);
00306             // 系统寄存器上锁
00307             HAL_REG_SYS_LOCK();
00308
00309             break;
00310         case HAL_ADC_REFERENCE_EXTERNAL: // 使用外部供电作为adc基准
00311
00312             // 系统寄存器解锁
00313             HAL_REG_SYS_UNLOCK();
00314             // SYS_AFE_REG1[3] ADC 基准电压调节
00315             // SYS_AFE_REG1[4] 使用外部输入电源作为 ADC REF
00316             HAL_REG_WRITE_BIT(SYS_AFE_REG1, SYS_AFE_REG1_GAIN_REF_BIT | SYS_AFE_REG1_REF2VDD_BIT,
SYS_AFE_REG1_REF2VDD_BIT);
00317             // 系统寄存器上锁
00318             HAL_REG_SYS_LOCK();
00319
00320             break;
00321         default:
00322             break;
00323     }
00324
00325     HAL_REG_WRITE(ADC0_DC_A0, HAL_TRIM_Read(0x00000310) << 4);
00326     HAL_REG_WRITE(ADC0_DC_A1, HAL_TRIM_Read(0x00000314) << 4);
00327     HAL_REG_WRITE(ADC0_AMC_A0, HAL_TRIM_Read(0x00000318));
00328     HAL_REG_WRITE(ADC0_AMC_A1, HAL_TRIM_Read(0x0000031c));
00329
00330     HAL_REG_WRITE(ADC0_DC_B0, HAL_TRIM_Read(0x00000320) << 4);
00331     HAL_REG_WRITE(ADC0_DC_B1, HAL_TRIM_Read(0x00000324) << 4);
00332     HAL_REG_WRITE(ADC0_AMC_B0, HAL_TRIM_Read(0x00000328));
00333     HAL_REG_WRITE(ADC0_AMC_B1, HAL_TRIM_Read(0x0000032c));
00334 }

```

```

00335
00340 void HAL_ADC_StateReset (HAL_ADC_x adc_x)
00341 {
00342     HAL_REG_SET (HAL_ADC[adc_x]->CFG, ADC0_CFG_FSM_RESET_BIT);
00343 }
00344
00350 float HAL_ADC_GetRange (HAL_ADC_x adc_x)
00351 {
00352     uint32_t val;
00353     float range = 5.0f;
00354     (void)adc_x;
00355     val = HAL_REG_READ_BIT (SYS_AFE_REG1, SYS_AFE_REG1_GAIN_REF_BIT | SYS_AFE_REG1_REF2VDD_BIT);
00356     val >>= 3;
00357     switch (val)
00358     {
00359         case 0:
00360             range = 3.6f;
00361             break;
00362         case 1:
00363             range = 1.8f;
00364             break;
00365         case 2:
00366             range = 5.0f;
00367             break;
00368         case 3:
00369             range = 5.0f;
00370             break;
00371     }
00372
00373     return range;
00374 }
00375
00381 float HAL_ADC_GetValuesUnit (HAL_ADC_x adc_x)
00382 {
00383     if (HAL_REG_READ_BIT (HAL_ADC[adc_x]->CFG, ADC0_CFG_DATA_ALIGN_BIT))
00384     {
00385         return HAL_ADC_GetRange (adc_x) / 0x07ff;
00386     }
00387     else
00388     {
00389         return HAL_ADC_GetRange (adc_x) / 0x7fff;
00390     }
00391 }
00392
00398 void HAL_ADC_SetGpioIn (HAL_ADC_x adc_x, HAL_GPIO_Px pin_x)
00399 {
00400     (void)adc_x;
00401     HAL_GPIO_SetAf (pin_x, HAL_GPIO_AF_ADC);
00402     HAL_GPIO_SetMode (pin_x, HAL_GPIO_MODE_AFE);
00403 }
00404
00410 void HAL_ADC_SetGpioTriggerOut (HAL_ADC_x adc_x, HAL_GPIO_Px pin_x)
00411 {
00412     (void)adc_x;
00413     HAL_GPIO_SetAf (pin_x, HAL_GPIO_AF_ADC_TRIG);
00414     HAL_GPIO_SetMode (pin_x, HAL_GPIO_MODE_OUT);
00415 }

```

4.66 source/hal_lks32mc08x_can.c 文件参考

```

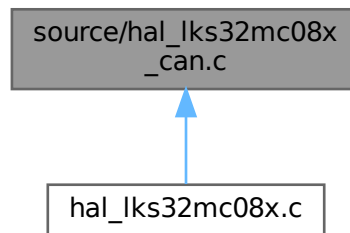
#include "../include/hal_lksmcu_clk.h"
#include "../include/hal_lksmcu_can.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc08x_reg.h"
#include "../hal_lks32mc08x.h"
#include "lks32mc08x_user_manual.h"
#include "stdint.h"

```

hal_lks32mc08x_can.c 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



函数

- static bool [GetBaudParam](#) (u32 baud, u8 *presc, u8 *sjw, u8 *seg1, u8 *seg2)
- static void [CAN_Write_TBUF](#) (HAL_CAN_x can_x, u32 id, HAL_CAN_FRAME_x frame_x, u8 *msg, u8 len)
- void [HAL_CAN_Reset](#) (HAL_CAN_x can_x)
- void [HAL_CAN_Enable](#) (HAL_CAN_x can_x)
- void [HAL_CAN_Disable](#) (HAL_CAN_x can_x)
- void [HAL_CAN_SetBaud](#) (HAL_CAN_x can_x, uint32_t baud)
- void [HAL_CAN_SetFastBaud](#) (HAL_CAN_x can_x, uint32_t fbaud)
- void [HAL_CAN_SetClockDiv](#) (HAL_CAN_x can_x, uint32_t div)
- uint32_t [HAL_CAN_GetClockDiv](#) (HAL_CAN_x can_x)
- uint32_t [HAL_CAN_GetClockFreq](#) (HAL_CAN_x can_x)
- uint32_t [HAL_CAN_GetBaud](#) (HAL_CAN_x can_x)
- void [HAL_CAN_SetTDC](#) (HAL_CAN_x can_x, uint8_t sspoff)
- bool [HAL_CAN_TxData](#) (HAL_CAN_x can_x, uint32_t id, HAL_CAN_FRAME_x frame_x, uint8_t *buf, uint32_t len)
- bool [HAL_CAN_TxData_STBWrite](#) (HAL_CAN_x can_x, uint32_t id, HAL_CAN_FRAME_x frame_x, uint8_t *buf, uint32_t len)
- bool [HAL_CAN_TxData_STBTrig](#) (HAL_CAN_x can_x, HAL_CAN_STBTrig_x trig)
- void [HAL_CAN_TxCancel](#) (HAL_CAN_x can_x)
- bool [HAL_CAN_Txing](#) (HAL_CAN_x can_x)
- bool [HAL_CAN_STBFull](#) (HAL_CAN_x can_x)
- HAL_CAN_TSSTAT_x HAL_CAN_TSSStat (HAL_CAN_x can_x)
- bool [HAL_CAN_RxData](#) (HAL_CAN_x can_x)
- uint32_t [HAL_CAN_RxDataID](#) (HAL_CAN_x can_x)
- uint8_t [HAL_CAN_RxDataBuf](#) (HAL_CAN_x can_x, uint8_t *buf)
- uint8_t [HAL_CAN_RxDataLen](#) (HAL_CAN_x can_x)
- uint8_t [HAL_CAN_RxDatalde](#) (HAL_CAN_x can_x)
- uint8_t [HAL_CAN_RxDataRtr](#) (HAL_CAN_x can_x)
- uint8_t [HAL_CAN_RxDataFdf](#) (HAL_CAN_x can_x)
- void [HAL_CAN.IDx.Filter](#) (HAL_CAN_x can_x, u8 ae_x, u32 id, u32 mask, HAL_CAN.IDEFilter_x ide_filter)
- void [HAL_CAN.IDx.FilterEnable](#) (HAL_CAN_x can_x, HAL_CAN.AE_x ae_x)
- void [HAL_CAN.IDx.FilterDisable](#) (HAL_CAN_x can_x, HAL_CAN.AE_x ae_x)
- void [HAL_CAN_SetSTBMode](#) (HAL_CAN_x can_x, HAL_CAN.STBMode_x mode_x)
- void [HAL_CAN.LOM.Enable](#) (HAL_CAN_x can_x)

CAN模块尝试刷写

监听模式使能/禁止

- void [HAL_CAN_LOM_Disable](#) ([HAL_CAN_x](#) can_x)
- void [HAL_CAN_LBMI_Enable](#) ([HAL_CAN_x](#) can_x)
内部回环模式使能/禁止
- void [HAL_CAN_LBMI_Disable](#) ([HAL_CAN_x](#) can_x)
- void [HAL_CAN_LBME_Enable](#) ([HAL_CAN_x](#) can_x)
外部回环模式使能/禁止
- void [HAL_CAN_LBME_Disable](#) ([HAL_CAN_x](#) can_x)
- void [HAL_CAN_SS_Enable](#) ([HAL_CAN_x](#) can_x)
单次传输模式使能/禁止
- void [HAL_CAN_SS_Disable](#) ([HAL_CAN_x](#) can_x)
- uint8_t [HAL_CAN_RxErrorCnt](#) ([HAL_CAN_x](#) can_x)
- uint8_t [HAL_CAN_TxErrorCnt](#) ([HAL_CAN_x](#) can_x)
- uint8_t [HAL_CAN_ErrorPassive](#) ([HAL_CAN_x](#) can_x)
- uint8_t [HAL_CAN_Busoff](#) ([HAL_CAN_x](#) can_x)
- uint8_t [HAL_CAN_ArbitrationLostCap](#) ([HAL_CAN_x](#) can_x)
- uint8_t [HAL_CAN_ROverflow](#) ([HAL_CAN_x](#) can_x)
- [HAL_CAN_ERROR_x](#) [HAL_CAN_KindOfError](#) ([HAL_CAN_x](#) can_x)
- void [HAL_CAN_SetGpioTx](#) ([HAL_CAN_x](#) can_x, [HAL_GPIO_Px](#) px)
- void [HAL_CAN_SetGpioRx](#) ([HAL_CAN_x](#) can_x, [HAL_GPIO_Px](#) px)

变量

- static uint32_t [hal_rx_id](#) [1] = {0}
- static uint8_t [hal_rx_ide](#) [1] = {0}
- static uint8_t [hal_rx_rtr](#) [1] = {0}
- static uint8_t [hal_rx_len](#) [1] = {0}
- static uint8_t [hal_rx_buf](#) [1][8] = {0}
- static const u16 [baud_cnt](#) = 12 * 1
- static const u16 [baud_param](#) [][6]

4.66.1 函数说明

4.66.1.1 CAN_Write_TBUF()

```
static void CAN_Write_TBUF (
    HAL\_CAN\_x can_x,
    u32 id,
    HAL\_CAN\_FRAME\_x frame_x,
    u8 * msg,
    u8 len ) [static]
```

在文件 [hal_lks32mc08x.can.c](#) 第 606 行定义。

这是这个函数的调用关系图:



4.66.1.2 GetBaudParam()

```
static bool GetBaudParam (
    u32 baud,
    u8 * presc,
    u8 * sjw,
    u8 * seg1,
    u8 * seg2 ) [static]
```

在文件 [hal_lks32mc08x.can.c](#) 第 587 行定义.

函数调用图:



这是这个函数的调用关系图:



4.66.1.3 HAL_CAN_ArbitrationLostCap()

```
uint8_t HAL_CAN_ArbitrationLostCap (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 488 行定义.

4.66.1.4 HAL_CAN_Busoff()

```
uint8_t HAL_CAN_Busoff (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 483 行定义.

4.66.1.5 HAL_CAN_Disable()

```
void HAL_CAN_Disable (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 59 行定义.

4.66.1.6 HAL_CAN_Enable()

```
void HAL_CAN_Enable (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 40 行定义.

4.66.1.7 HAL_CAN_ErrorPassive()

```
uint8_t HAL_CAN_ErrorPassive (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 478 行定义.

4.66.1.8 HAL_CAN_GetBaud()

```
uint32_t HAL_CAN_GetBaud (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 120 行定义.

函数调用图:

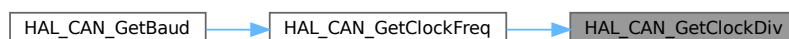


4.66.1.9 HAL_CAN_GetClockDiv()

```
uint32_t HAL_CAN_GetClockDiv (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 108 行定义.

这是这个函数的调用关系图:

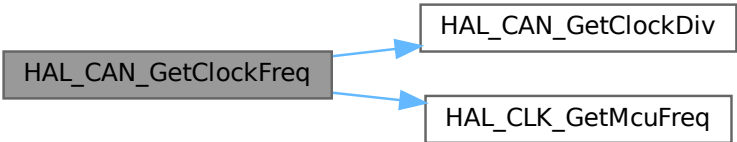


4.66.1.10 HAL_CAN_GetClockFreq()

```
uint32_t HAL_CAN_GetClockFreq (
    HAL_CAN_x can_x )
```

在文件 `hal_lks32mc08x.can.c` 第 114 行定义.

函数调用图:



这是这个函数的调用关系图:



4.66.1.11 HAL_CAN_IDx_Filter()

```
void HAL_CAN_IDx_Filter (
    HAL_CAN_x can_x,
    u8 ae_x,
    u32 id,
    u32 mask,
    HAL_CAN_IDEFilter_x ide_filter )
```

CAN滤波尝试

参数

08x的CAN滤波模拟为两个通道,	ae_x=0时使用单滤波, ae_x=1时使用双滤波。双滤波时, 对于扩展帧, 只匹配高16位。

在文件 `hal_lks32mc08x.can.c` 第 295 行定义.

4.66.1.12 HAL_CAN_IDx_FilterDisable()

```
void HAL_CAN_IDx_FilterDisable (
    HAL_CAN_x can_x,
    HAL_CAN_AE_x ae_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 389 行定义.

4.66.1.13 HAL_CAN_IDx_FilterEnable()

```
void HAL_CAN_IDx_FilterEnable (
    HAL_CAN_x can_x,
    HAL_CAN_AE_x ae_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 383 行定义.

4.66.1.14 HAL_CAN_KindOfError()

```
HAL_CAN_ERROR_x HAL_CAN_KindOfError (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 498 行定义.

4.66.1.15 HAL_CAN_LBME_Disable()

```
void HAL_CAN_LBME_Disable (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 448 行定义.

4.66.1.16 HAL_CAN_LBME_Enable()

```
void HAL_CAN_LBME_Enable (
    HAL_CAN_x can_x )
```

外部回环模式使能/禁止

更新记录

自发自收，自应答，需将芯片RX和TX接入CAN收发器，收发器可以不挂入总线。应在CAN初始化和ID滤波设置之后再使能。

在文件 [hal_lks32mc08x.can.c](#) 第 443 行定义.

4.66.1.17 HAL_CAN_LBMI_Disable()

```
void HAL_CAN_LBMI_Disable (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 432 行定义.

4.66.1.18 HAL_CAN_LBMI_Enable()

```
void HAL_CAN_LBMI_Enable (
    HAL_CAN_x can_x )
```

内部回环模式使能/禁止

更新记录

自发自收，无需自应答，芯片RX和TX引脚可悬空。应在CAN初始化和ID滤波设置之后再使能。

在文件 [hal_lks32mc08x.can.c](#) 第 427 行定义.

4.66.1.19 HAL_CAN_LOM_Disable()

```
void HAL_CAN_LOM_Disable (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 416 行定义.

4.66.1.20 HAL_CAN_LOM_Enable()

```
void HAL_CAN_LOM_Enable (
    HAL_CAN_x can_x )
```

监听模式使能/禁止

更新记录

如果设置了 TPE、TSONE 或 TSALL，则无法设置 LOM。如果 LOM 启用且 LBME 禁用，则无法启动传输。LOM=1 和 LBME=0 禁止所有传输。LOM=1 和 LBME=1 禁止应答相应接收到的帧和错误帧，但可以发送正常通信中，禁止使能该位。

在文件 [hal_lks32mc08x.can.c](#) 第 411 行定义.

4.66.1.21 HAL_CAN_Reset()

```
void HAL_CAN_Reset (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 19 行定义.

4.66.1.22 HAL_CAN_ROverflow()

```
uint8_t HAL_CAN_ROverflow (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 493 行定义.

4.66.1.23 HAL_CAN_RxData()

```
bool HAL_CAN_RxData (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 193 行定义.

4.66.1.24 HAL_CAN_RxDataBuf()

```
uint8_t HAL_CAN_RxDataBuf (
    HAL_CAN_x can_x,
    uint8_t * buf )
```

在文件 [hal_lks32mc08x.can.c](#) 第 256 行定义.

4.66.1.25 HAL_CAN_RxDataFdf()

```
uint8_t HAL_CAN_RxDataFdf (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 282 行定义.

4.66.1.26 HAL_CAN_RxDataID()

```
uint32_t HAL_CAN_RxDataID (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 251 行定义.

4.66.1.27 HAL_CAN_RxDatalde()

```
uint8_t HAL_CAN_RxDatalde (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 272 行定义.

4.66.1.28 HAL_CAN_RxDataLen()

```
uint8_t HAL_CAN_RxDataLen (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 267 行定义.

4.66.1.29 HAL_CAN_RxDataRtr()

```
uint8_t HAL_CAN_RxDataRtr (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 277 行定义.

4.66.1.30 HAL_CAN_RxErrorCnt()

```
uint8_t HAL_CAN_RxErrorCnt (
    HAL_CAN_x can_x )
```

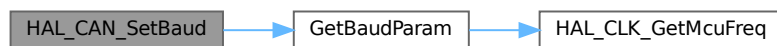
在文件 [hal_lks32mc08x.can.c](#) 第 468 行定义.

4.66.1.31 HAL_CAN_SetBaud()

```
void HAL_CAN_SetBaud (
    HAL_CAN_x can_x,
    uint32_t baud )
```

在文件 [hal_lks32mc08x.can.c](#) 第 77 行定义.

函数调用图:



4.66.1.32 HAL_CAN_SetClockDiv()

```
void HAL_CAN_SetClockDiv (
    HAL_CAN_x can_x,
    uint32_t div )
```

在文件 [hal_lks32mc08x.can.c](#) 第 99 行定义.

4.66.1.33 HAL_CAN_SetFastBaud()

```
void HAL_CAN_SetFastBaud (
    HAL_CAN_x can_x,
    uint32_t fbaud )
```

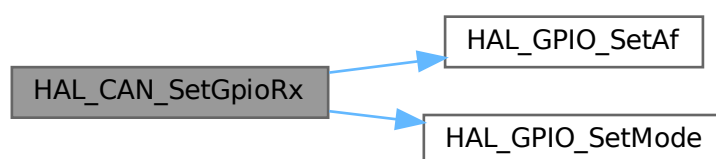
在文件 [hal_lks32mc08x.can.c](#) 第 92 行定义.

4.66.1.34 HAL_CAN_SetGpioRx()

```
void HAL_CAN_SetGpioRx (
    HAL_CAN_x can_x,
    HAL_GPIO_Px px )
```

在文件 [hal_lks32mc08x.can.c](#) 第 522 行定义.

函数调用图:

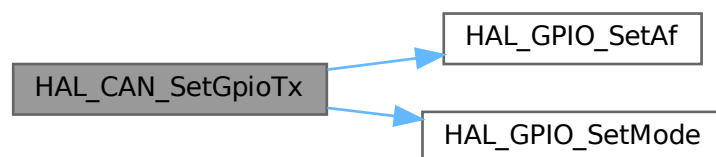


4.66.1.35 HAL_CAN_SetGpioTx()

```
void HAL_CAN_SetGpioTx (
    HAL_CAN_x can_x,
    HAL_GPIO_Px px )
```

在文件 [hal_lks32mc08x.can.c](#) 第 515 行定义.

函数调用图:



4.66.1.36 HAL_CAN_SetSTBMode()

```
void HAL_CAN_SetSTBMode (
    HAL_CAN_x can_x,
    HAL_CAN_STBMode_x mode_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 396 行定义.

4.66.1.37 HAL_CAN_SetTDC()

```
void HAL_CAN_SetTDC (
    HAL_CAN_x can_x,
    uint8_t sspoff )
```

在文件 [hal_lks32mc08x.can.c](#) 第 129 行定义.

4.66.1.38 HAL_CAN_SS_Disable()

```
void HAL_CAN_SS_Disable (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 463 行定义.

4.66.1.39 HAL_CAN_SS_Enable()

```
void HAL_CAN_SS_Enable (
    HAL_CAN_x can_x )
```

单次传输模式使能/禁止

更新记录

08x不支持

在文件 [hal_lks32mc08x.can.c](#) 第 458 行定义.

4.66.1.40 HAL_CAN_STBFull()

```
bool HAL_CAN_STBFull (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 177 行定义.

4.66.1.41 HAL_CAN_TSStat()

```
HAL_CAN_TSSTAT_x HAL_CAN_TSStat (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 183 行定义.

4.66.1.42 HAL_CAN_TxCancel()

```
void HAL_CAN_TxCancel (
    HAL_CAN_x can_x )
```

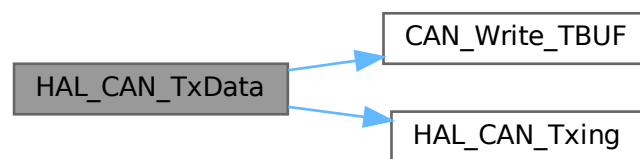
在文件 [hal_lks32mc08x.can.c](#) 第 167 行定义.

4.66.1.43 HAL_CAN_TxData()

```
bool HAL_CAN_TxData (
    HAL_CAN_x can_x,
    uint32_t id,
    HAL_CAN_FRAME_x frame_x,
    uint8_t * buf,
    uint32_t len )
```

在文件 [hal_lks32mc08x.can.c](#) 第 135 行定义.

函数调用图:



4.66.1.44 HAL_CAN_TxData_STBTrig()

```
bool HAL_CAN_TxData_STBTrig (
    HAL_CAN_x can_x,
    HAL_CAN_STBTrig_x trig )
```

在文件 [hal_lks32mc08x.can.c](#) 第 160 行定义.

4.66.1.45 HAL_CAN_TxData_STBWrite()

```
bool HAL_CAN_TxData_STBWrite (
    HAL_CAN_x can_x,
    uint32_t id,
    HAL_CAN_FRAME_x frame_x,
    uint8_t * buf,
    uint32_t len )
```

在文件 [hal_lks32mc08x.can.c](#) 第 149 行定义.

4.66.1.46 HAL_CAN_TxErrorCnt()

```
uint8_t HAL_CAN_TxErrorCnt (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 473 行定义.

4.66.1.47 HAL_CAN_Txing()

```
bool HAL_CAN_Txing (
    HAL_CAN_x can_x )
```

在文件 [hal_lks32mc08x.can.c](#) 第 172 行定义.

这是这个函数的调用关系图:



4.66.2 变量说明

4.66.2.1 baud_cnt

```
const u16 baud_cnt = 12 * 1 [static]
```

在文件 [hal_lks32mc08x.can.c](#) 第 547 行定义.

4.66.2.2 baud_param

```
const u16 baud_param[][6] [static]
```

在文件 [hal_lks32mc08x.can.c](#) 第 548 行定义.

4.66.2.3 hal_rx_buf

```
uint8_t hal_rx_buf[1][8] = {0} [static]
```

在文件 [hal_lks32mc08x.can.c](#) 第 13 行定义.

4.66.2.4 hal_rx_id

```
uint32_t hal_rx_id[1] = {0} [static]
```

在文件 [hal_lks32mc08x.can.c](#) 第 9 行定义.

4.66.2.5 hal_rx_ide

```
uint8_t hal_rx_ide[1] = {0} [static]
```

在文件 [hal_lks32mc08x.can.c](#) 第 10 行定义.

4.66.2.6 hal_rx_len

```
uint8_t hal_rx_len[1] = {0} [static]
```

在文件 [hal_lks32mc08x_can.c](#) 第 12 行定义。

4.66.2.7 hal_rx_rtr

```
uint8_t hal_rx_rtr[1] = {0} [static]
```

在文件 [hal_lks32mc08x_can.c](#) 第 11 行定义。

4.67 hal_lks32mc08x_can.c

[浏览该文件的文档。](#)

```
00001 #include "../include/hal_lksmcu_clk.h"
00002 #include "../include/hal_lksmcu_can.h"
00003 #include "../include/hal_lksmcu_reg.h"
00004 #include "../include/hal_lks32mc08x_reg.h"
00005 #include "../hal_lks32mc08x.h"
00006 #include "lks32mc08x_user_manual.h"
00007 #include "stdint.h"
00008
00009 static uint32_t hal_rx_id[1] = {0};
00010 static uint8_t hal_rx_idc[1] = {0};
00011 static uint8_t hal_rx_rtr[1] = {0};
00012 static uint8_t hal_rx_len[1] = {0};
00013 static uint8_t hal_rx_buf[1][8] = {0};
00014
00015 static bool GetBaudParam(u32 baud, u8 *presc, u8 *sjw, u8 *seg1, u8 *seg2);
00016 static void CAN_Write_TBUF(HAL_CAN_x can_x, u32 id, HAL_CAN_FRAME_x frame_x, u8 *msg, u8 len);
00017
00018 /* 模块复位 */
00019 void HAL_CAN_Reset(HAL_CAN_x can_x)
00020 {
00021     switch (can_x)
00022     {
00023         case HAL_CAN_0:
00024             // 系统寄存器解锁
00025             HAL_REG_SYS_UNLOCK();
00026             HAL_REG_SET(SYS_SFT_RST, SYS_SFT_RST_CAN_SFT_RST_BIT);
00027             HAL_REG_RESET(SYS_SFT_RST, SYS_SFT_RST_CAN_SFT_RST_BIT);
00028             // 系统寄存器上锁
00029             HAL_REG_SYS_LOCK();
00030
00031             HAL_REG_SET(HAL_CAN[can_x]->MOD, CAN_MOD_MODE_BIT); // 复位模式
00032             HAL_REG_RESET(HAL_CAN[can_x]->MOD, CAN_MOD_MODE_BIT); // 正常工作模式
00033             break;
00034         default:
00035             break;
00036     }
00037 }
00038
00039 /* 模块使能 */
00040 void HAL_CAN_Enable(HAL_CAN_x can_x)
00041 {
00042     switch (can_x)
00043     {
00044         case HAL_CAN_0:
00045             // 系统寄存器解锁
00046             HAL_REG_SYS_UNLOCK();
00047             HAL_REG_SET(SYS_CLK_FEN, SYS_CLK_FEN_CAN_CLK_EN_BIT);
00048             // 系统寄存器上锁
00049             HAL_REG_SYS_LOCK();
00050
00051             HAL_REG_RESET(HAL_CAN[can_x]->MOD, CAN_MOD_MODE_BIT); // 正常工作模式
00052             break;
00053         default:
00054             break;
00055     }
00056 }
00057
00058 /* 模块关闭 */
```

```

00059 void HAL_CAN_Disable(HAL_CAN_x can_x)
00060 {
00061     switch (can_x)
00062     {
00063         case HAL_CAN_0:
00064             // 系统寄存器解锁
00065             HAL_REG_SYS_UNLOCK();
00066             HAL_REG_RESET(SYS_CLK_FEN, SYS_CLK_FEN_CAN_CLK_EN_BIT);
00067             // 系统寄存器上锁
00068             HAL_REG_SYS_LOCK();
00069
00070             HAL_REG_SET(HAL_CAN[can_x]->MOD, CAN_MOD_MODE_BIT); // 复位模式
00071             break;
00072         default:
00073             break;
00074     }
00075 }
00076
00077 void HAL_CAN_SetBaud(HAL_CAN_x can_x, uint32_t baud)
00078 {
00079     u8 presc, sjw, seg1, seg2;
00080
00081     HAL_REG_SET(HAL_CAN[can_x]->MOD, CAN_MOD_MODE_BIT); // 复位模式
00082     if (baud > 0 && GetBaudParam(baud, &presc, &sjw, &seg1, &seg2))
00083     {
00084         HAL_REG_WRITE(HAL_CAN[can_x]->BTR0, CAN_BTR0_BAUDRATE(presc) + CAN_BTR0_SJW(sjw));
00085         HAL_REG_WRITE(HAL_CAN[can_x]->BTR1, CAN_BTR1_SEG1(seg1) + CAN_BTR1_SEG2(seg2));
00086     }
00087
00088     HAL_REG_RESET(HAL_CAN[can_x]->MOD, CAN_MOD_MODE_BIT); // 正常工作模式
00089 }
00090
00091 // 08x不支持FD
00092 void HAL_CAN_SetFastBaud(HAL_CAN_x can_x, uint32_t fbaud)
00093 {
00094     (void)can_x;
00095     (void)fbaud;
00096 }
00097
00098 // 设置CAN预分频 (只改BTR0的BAUDRATE, 位时序参数不变)
00099 void HAL_CAN_SetClockDiv(HAL_CAN_x can_x, uint32_t div)
00100 {
00101     if (div < 2) div = 2;
00102     HAL_REG_SET(HAL_CAN[can_x]->MOD, CAN_MOD_MODE_BIT); // 复位模式
00103     HAL_REG_WRITE_BIT_MASK_POS(HAL_CAN[can_x]->BTR0, CAN_BTR0_BAUDRATE, div / 2 - 1);
00104     HAL_REG_RESET(HAL_CAN[can_x]->MOD, CAN_MOD_MODE_BIT); // 正常工作模式
00105 }
00106
00107 // 获取CAN预分频实际除数
00108 uint32_t HAL_CAN_GetClockDiv(HAL_CAN_x can_x)
00109 {
00110     return 2 * (HAL_REG_READ_BIT_MASK_POS(HAL_CAN[can_x]->BTR0, CAN_BTR0_BAUDRATE) + 1);
00111 }
00112
00113 // 获取分频后工作时钟
00114 uint32_t HAL_CAN_GetClockFreq(HAL_CAN_x can_x)
00115 {
00116     return HAL_CLK_GetMcuFreq() / HAL_CAN_GetClockDiv(can_x);
00117 }
00118
00119 // 获取当前标称波特率
00120 uint32_t HAL_CAN_GetBaud(HAL_CAN_x can_x)
00121 {
00122     uint32_t clk = HAL_CAN_GetClockFreq(can_x);
00123     uint32_t seg1 = HAL_REG_READ_BIT_MASK_POS(HAL_CAN[can_x]->BTR1, CAN_BTR1_SEG1);
00124     uint32_t seg2 = HAL_REG_READ_BIT_MASK_POS(HAL_CAN[can_x]->BTR1, CAN_BTR1_SEG2);
00125     return clk / (1 + (seg1 + 2) + (seg2 + 1));
00126 }
00127
00128 // 08x不支持TDC
00129 void HAL_CAN_SetTDC(HAL_CAN_x can_x, uint8_t sspoff)
00130 {
00131     (void)can_x;
00132     (void)sspoff;
00133 }
00134
00135 bool HAL_CAN_TxData(HAL_CAN_x can_x, uint32_t id, HAL_CAN_FRAME_x frame_x, uint8_t *buf, uint32_t len)
00136 {
00137     if (HAL_CAN_Txing(can_x))
00138     {
00139         return false;
00140     }
00141
00142     CAN_Write_TBUF(can_x, id, frame_x, buf, len);
00143     HAL_REG_SET(HAL_CAN[can_x]->CMR, CAN_CMR_TRANS_REQ_BIT); // CAN发送传输请求
00144
00145     return true;

```

```

00146 }
00147
00148 // 08x不支持缓冲发送
00149 bool HAL_CAN_TxData_STBWrite(HAL_CAN_x can_x, uint32_t id, HAL_CAN_FRAME_x frame_x, uint8_t *buf, uint32_t
    len)
00150 {
00151     (void)can_x;
00152     (void)id;
00153     (void)frame_x;
00154     (void)buf;
00155     (void)len;
00156     return false;
00157 }
00158
00159 // 08x不支持缓冲发送
00160 bool HAL_CAN_TxData_STBTrig(HAL_CAN_x can_x, HAL_CAN_STBTrig_x trig)
00161 {
00162     (void)can_x;
00163     (void)trig;
00164     return false;
00165 }
00166
00167 void HAL_CAN_TxCancel(HAL_CAN_x can_x)
00168 {
00169     HAL_REG_SET(HAL_CAN[can_x]->CMR, CAN_CMR_INTR_TRANS_BIT); // 取消未执行的发送传输
00170 }
00171
00172 bool HAL_CAN_Txing(HAL_CAN_x can_x)
00173 {
00174     return HAL_REG_READ_BIT(HAL_CAN[can_x]->SR, CAN_SR_TXING_BIT); // CAN正在发送一帧数据
00175 }
00176
00177 bool HAL_CAN_STBFull(HAL_CAN_x can_x)
00178 {
00179     (void)can_x;
00180     return false;
00181 }
00182
00183 HAL_CAN_TSSTAT_x HAL_CAN_TSStat(HAL_CAN_x can_x)
00184 {
00185     (void)can_x;
00186     return HAL_CAN_TSSTAT_Empty;
00187 }
00188
00189 /*
00190 * 从接收缓冲区读出数据, 暂存到全局变量中
00191 * 返回值: 成功读到数据返回1, 无数据返回0
00192 */
00193 bool HAL_CAN_RxData(HAL_CAN_x can_x)
00194 {
00195     // 存有一帧或多帧数据, 可以通过 RFIFO 寄存器读取
00196     if (HAL_REG_READ_BIT(HAL_CAN[can_x]->SR, CAN_SR_DATA_AVAIL_BIT))
00197     {
00198         u8 reg_inf = HAL_REG_READ(HAL_CAN[can_x]->TXRX0);
00199         if (reg_inf & BIT6)
00200         {
00201             hal_rx_rtr[can_x] = 1;
00202         }
00203         else
00204         {
00205             hal_rx_rtr[can_x] = 0;
00206         }
00207         hal_rx_len[can_x] = reg_inf & 0x0F;
00208         if (reg_inf & BIT7) // EFF (扩展) 帧
00209         {
00210             hal_rx_id[can_x] = 1;
00211             hal_rx_id[can_x] = (HAL_REG_READ(HAL_CAN[can_x]->TXRX1) << 21);
00212             hal_rx_id[can_x] |= (HAL_REG_READ(HAL_CAN[can_x]->TXRX2) << 13);
00213             hal_rx_id[can_x] |= (HAL_REG_READ(HAL_CAN[can_x]->TXRX3) << 5);
00214             hal_rx_id[can_x] |= (HAL_REG_READ(HAL_CAN[can_x]->TXRX4) >> 3);
00215
00216             hal_rx_buf[can_x][0] = HAL_REG_READ(HAL_CAN[can_x]->TXRX5);
00217             hal_rx_buf[can_x][1] = HAL_REG_READ(HAL_CAN[can_x]->TXRX6);
00218             hal_rx_buf[can_x][2] = HAL_REG_READ(HAL_CAN[can_x]->TXRX7);
00219             hal_rx_buf[can_x][3] = HAL_REG_READ(HAL_CAN[can_x]->TXRX8);
00220             hal_rx_buf[can_x][4] = HAL_REG_READ(HAL_CAN[can_x]->TXRX9);
00221             hal_rx_buf[can_x][5] = HAL_REG_READ(HAL_CAN[can_x]->TXRXA);
00222             hal_rx_buf[can_x][6] = HAL_REG_READ(HAL_CAN[can_x]->TXRXB);
00223             hal_rx_buf[can_x][7] = HAL_REG_READ(HAL_CAN[can_x]->TXRXC);
00224         }
00225     }
00226     else // SFF (标准) 帧
00227     {
00228         hal_rx_id[can_x] = 0;
00229         hal_rx_id[can_x] = HAL_REG_READ(HAL_CAN[can_x]->TXRX1) << 3;
00230         hal_rx_id[can_x] |= HAL_REG_READ(HAL_CAN[can_x]->TXRX2) >> 5;
00231     }

```

```

00232         hal_rx_buf[can_x][0] = HAL_REG_READ(HAL_CAN[can_x]->TXRX3);
00233         hal_rx_buf[can_x][1] = HAL_REG_READ(HAL_CAN[can_x]->TXRX4);
00234         hal_rx_buf[can_x][2] = HAL_REG_READ(HAL_CAN[can_x]->TXRX5);
00235         hal_rx_buf[can_x][3] = HAL_REG_READ(HAL_CAN[can_x]->TXRX6);
00236         hal_rx_buf[can_x][4] = HAL_REG_READ(HAL_CAN[can_x]->TXRX7);
00237         hal_rx_buf[can_x][5] = HAL_REG_READ(HAL_CAN[can_x]->TXRX8);
00238         hal_rx_buf[can_x][6] = HAL_REG_READ(HAL_CAN[can_x]->TXRX9);
00239         hal_rx_buf[can_x][7] = HAL_REG_READ(HAL_CAN[can_x]->TXRXA);
00240     }
00241
00242     HAL_REG_SET(HAL_CAN[can_x]->CMR, CAN_CMR_RELEASE_FIFO_BIT); // 释放接收缓冲区
00243     return true; // 接收到有效数据
00244 }
00245 else
00246 {
00247     return false; // 未接收到数据
00248 }
00249 }
00250
00251 uint32_t HAL_CAN_RxDataID(HAL_CAN_x can_x)
00252 {
00253     return hal_rx_id[can_x];
00254 }
00255
00256 uint8_t HAL_CAN_RxDataBuf(HAL_CAN_x can_x, uint8_t *buf)
00257 {
00258     uint8_t len = hal_rx_len[can_x];
00259     for (uint8_t i = 0; i < len; i++)
00260     {
00261         buf[i] = hal_rx_buf[can_x][i];
00262     }
00263     return len;
00264 }
00265
00266 uint8_t HAL_CAN_RxDataLen(HAL_CAN_x can_x)
00267 {
00268     return hal_rx_len[can_x];
00269 }
00270
00271 uint8_t HAL_CAN_RxDataIde(HAL_CAN_x can_x)
00272 {
00273     return hal_rx_ide[can_x];
00274 }
00275
00276 uint8_t HAL_CAN_RxDataRtr(HAL_CAN_x can_x)
00277 {
00278     return hal_rx_rtr[can_x];
00279 }
00280
00281 uint8_t HAL_CAN_RxDataFdf(HAL_CAN_x can_x)
00282 {
00283     (void)can_x;
00284     return 0;
00285 }
00286
00287 void HAL_CAN_IDx_Filter(HAL_CAN_x can_x, u8 ae_x, u32 id, u32 mask, HAL_CAN_IDEFilter_x ide_filter)
00288 {
00289     u32 acr = id;
00290     u32 amr = mask;
00291     u8 rtr_acr = 0;
00292     u8 rtr_amr = 1; // rtr位不需要匹配
00293     u8 byte1_acr = 0;
00294     u8 byte1_amr = 0xFF;
00295     u8 byte2_acr = 0;
00296     u8 byte2_amr = 0xFF;
00297
00298     if (ae_x > HAL_CAN_AE_1)
00299     {
00300         return;
00301     }
00302
00303     HAL_REG_SET(HAL_CAN[can_x]->MOD, CAN_MOD_MODE_BIT); // 复位模式
00304     if (ae_x == HAL_CAN_AE_0) // 打开HAL_CAN_AE_0, 单滤波ID
00305     {
00306         HAL_REG_SET(HAL_CAN[can_x]->MOD, CAN_MOD_FLT_ID_BIT);
00307
00308         if (ide_filter == HAL_CAN_IDEFilter_Ex) // 长ID扩展帧
00309         {
00310             acr &= 0x1fffffff;
00311             amr &= 0x1fffffff;
00312
00313             /*可接收的ID寄存器, 当输入帧的 ID 同此匹配, 则被 CAN 模块接受
00314             配合 AMR 寄存器, 可同一类 ID 匹配。*/
00315             HAL_REG_WRITE(CAN_ACR3, (u8)(acr << 3) | (rtr_acr << 2));
00316             HAL_REG_WRITE(CAN_ACR2, (u8)(acr >> 5));
00317             HAL_REG_WRITE(CAN_ACR1, (u8)(acr >> 13));

```

```

00326     HAL_REG_WRITE(CAN_ACR0, (u8)(acr >> 21));
00327     /*配合 ACR 寄存器, 同输入 ID 进行匹配。AMR 寄存器某一位为 0, 标识
00328     ID 对应位需要同 ACR 匹配; 为 1, 标识 ID 对应位不需要同 ACR 匹配*/
00329     HAL_REG_WRITE(CAN_AMR3, (u8)((amr << 3) | (rtr_amr << 2) | 0x03));
00330     HAL_REG_WRITE(CAN_AMR2, (u8)(amr >> 5));
00331     HAL_REG_WRITE(CAN_AMR1, (u8)(amr >> 13));
00332     HAL_REG_WRITE(CAN_AMR0, (u8)(amr >> 21));
00333 }
00334 else // 短ID标准帧或扩展帧ID高位
00335 {
00336     acr &= 0x7ff;
00337     amr &= 0x7ff;
00338
00339     HAL_REG_WRITE(CAN_ACR0, (u8)(acr >> 3));
00340     HAL_REG_WRITE(CAN_ACR1, (u8)(acr << 5) | (rtr_acr << 4));
00341     HAL_REG_WRITE(CAN_ACR2, byte1_acr);
00342     HAL_REG_WRITE(CAN_ACR3, byte2_acr);
00343
00344     HAL_REG_WRITE(CAN_AMR0, (u8)(amr >> 3));
00345     HAL_REG_WRITE(CAN_AMR1, (u8)(amr << 5) | (rtr_amr << 4) | 0x0f);
00346     HAL_REG_WRITE(CAN_AMR2, byte1_amr);
00347     HAL_REG_WRITE(CAN_AMR3, byte2_amr);
00348 }
00349 }
00350 else if (ae_x == HAL_CAN_AE_1) // 打开HAL_CAN_AE_1, 表示开启双滤波
00351 {
00352     HAL_REG_RESET(HAL_CAN[can_x]->MOD, CAN_MOD_FLT_ID_BIT);
00353
00354     if (ide_filter == HAL_CAN_IDE_Filter_Ex) // 长ID扩展帧
00355     {
00356         acr &= 0x1fffffff;
00357         amr &= 0x1fffffff;
00358
00359         HAL_REG_WRITE(CAN_ACR3, (u8)(acr >> 13));
00360         HAL_REG_WRITE(CAN_ACR2, (u8)(acr >> 21));
00361
00362         HAL_REG_WRITE(CAN_AMR3, (u8)(amr >> 13));
00363         HAL_REG_WRITE(CAN_AMR2, (u8)(amr >> 21));
00364     }
00365     else // 短ID标准帧或扩展帧ID高位
00366     {
00367         acr &= 0x7ff;
00368         amr &= 0x7ff;
00369
00370         HAL_REG_RESET(CAN_ACR3, 0xf0);
00371         HAL_REG_WRITE(CAN_ACR2, (u8)(acr >> 3));
00372         HAL_REG_SET(CAN_ACR3, (u8)(acr << 5) | (rtr_acr << 4));
00373
00374         HAL_REG_RESET(CAN_AMR3, 0xf0);
00375         HAL_REG_WRITE(CAN_AMR2, (u8)(amr >> 3));
00376         HAL_REG_SET(CAN_AMR3, (u8)(amr << 5) | (rtr_amr >> 4));
00377     }
00378 }
00379
00380 HAL_REG_RESET(HAL_CAN[can_x]->MOD, CAN_MOD_MODE_BIT); // 正常工作模式
00381 }
00382
00383 void HAL_CAN_IDx_FilterEnable(HAL_CAN_x can_x, HAL_CAN_AE_x ae_x)
00384 {
00385     (void)can_x;
00386     (void)ae_x;
00387 }
00388
00389 void HAL_CAN_IDx_FilterDisable(HAL_CAN_x can_x, HAL_CAN_AE_x ae_x)
00390 {
00391     (void)can_x;
00392     (void)ae_x;
00393 }
00394
00395 /* 仅在45x下支持 */
00396 void HAL_CAN_SetSTBMode(HAL_CAN_x can_x, HAL_CAN_STBMode_x mode_x) /* 设置STB发送缓冲区模式 */
00397 {
00398     (void)can_x;
00399     (void)mode_x;
00400 }
00401
00411 void HAL_CAN_LOM_Enable(HAL_CAN_x can_x)
00412 {
00413     HAL_REG_SET(HAL_CAN[can_x]->MOD, CAN_MOD_FUNC_MODE_BIT);
00414 }
00415
00416 void HAL_CAN_LOM_Disable(HAL_CAN_x can_x)
00417 {
00418     HAL_REG_RESET(HAL_CAN[can_x]->MOD, CAN_MOD_FUNC_MODE_BIT);
00419 }
00420
00427 void HAL_CAN_LBMI_Enable(HAL_CAN_x can_x)

```

```

00428 {
00429     HAL_REG_SET(HAL_CAN[can_x]->MOD, CAN_MOD.TEST_MODE_BIT);
00430 }
00431
00432 void HAL_CAN_LBMI_Disable(HAL_CAN_x can_x)
00433 {
00434     HAL_REG_RESET(HAL_CAN[can_x]->MOD, CAN_MOD.TEST_MODE_BIT);
00435 }
00436
00443 void HAL_CAN_LBME_Enable(HAL_CAN_x can_x)
00444 {
00445     (void)can_x;
00446 }
00447
00448 void HAL_CAN_LBME_Disable(HAL_CAN_x can_x)
00449 {
00450     (void)can_x;
00451 }
00452
00458 void HAL_CAN_SS_Enable(HAL_CAN_x can_x) /* 单次传输模式使能/禁止 */
00459 {
00460     (void)can_x;
00461 }
00462
00463 void HAL_CAN_SS_Disable(HAL_CAN_x can_x)
00464 {
00465     (void)can_x;
00466 }
00467
00468 uint8_t HAL_CAN_RxErrorCnt(HAL_CAN_x can_x)
00469 {
00470     return HAL_REG_READ_BIT_MASK_POS(HAL_CAN[can_x]->RXERR, CAN_RXERR_RXERR);
00471 }
00472
00473 uint8_t HAL_CAN_TxErrorCnt(HAL_CAN_x can_x)
00474 {
00475     return HAL_REG_READ_BIT_MASK_POS(HAL_CAN[can_x]->TXERR, CAN_TXERR_TXERR);
00476 }
00477
00478 uint8_t HAL_CAN_ErrorPassive(HAL_CAN_x can_x)
00479 {
00480     return HAL_REG_READ_BIT(HAL_CAN[can_x]->IR, CAN_IR_M_ERR_IF_BIT);
00481 }
00482
00483 uint8_t HAL_CAN_Busoff(HAL_CAN_x can_x)
00484 {
00485     return HAL_REG_READ_BIT(HAL_CAN[can_x]->SR, CAN_SR_ON_BUS_BIT) > 0;
00486 }
00487
00488 uint8_t HAL_CAN_ArbitrationLostCap(HAL_CAN_x can_x)
00489 {
00490     return HAL_REG_READ_BIT_MASK_POS(HAL_CAN[can_x]->ALC, CAN_ALC_LOST_ARB);
00491 }
00492
00493 uint8_t HAL_CAN_ROverflow(HAL_CAN_x can_x)
00494 {
00495     return HAL_REG_READ_BIT(HAL_CAN[can_x]->SR, CAN_SR_RFIFO_EMPTY_BIT) > 0;
00496 }
00497
00498 HAL_CAN_ERROR_x HAL_CAN_KindOfError(HAL_CAN_x can_x)
00499 {
00500     uint8_t koer = HAL_REG_READ_BIT_MASK_POS(HAL_CAN[can_x]->ECC, CAN_ECC_BUS_ERR_TYPE);
00501     switch (koer)
00502     {
00503     case 0:
00504         return HAL_CAN_ERROR_Bit;
00505     case 1:
00506         return HAL_CAN_ERROR_Form;
00507     case 2:
00508         return HAL_CAN_ERROR_Stuff;
00509     case 3:
00510         return HAL_CAN_ERROR_Unknown;
00511     default:
00512         return HAL_CAN_ERROR_Unknown;
00513     }
00514 }
00515 void HAL_CAN_SetGpioTx(HAL_CAN_x can_x, HAL_GPIO_Px px)
00516 {
00517     (void)can_x;
00518     HAL_GPIO_SetAf(px, HAL_GPIO_AF_CAN);
00519     HAL_GPIO_SetMode(px, HAL_GPIO_MODE_OUT);
00520 }
00521
00522 void HAL_CAN_SetGpioRx(HAL_CAN_x can_x, HAL_GPIO_Px px)
00523 {
00524     (void)can_x;
00525     HAL_GPIO_SetAf(px, HAL_GPIO_AF_CAN);

```



```

00526     HAL_GPIO_SetMode(px, HAL_GPIO_MODE_IN);
00527 }
00528
00529 /*
00530 波特率计算:
00531  $TQ = 2 * Tclk * (PRESC + 1) = 2 * (PRESC + 1) / Fclk$ 
00532  $Tseg1 = TQ * (SEG1 + 2)$ 
00533  $Tseg2 = TQ * (SEG2 + 1)$ 
00534  $CAN\_BAUD = 1 / (1 * TQ + Tseg1 + Tseg2) = 1 / (TQ * (1 + SEG1 + 1 + SEG2 + 1)) = 1 / (TQ * (SEG1 + SEG2 + 3))$ 
00535  $SEG1 + SEG2 = 1 / (CAN\_BAUD * TQ) - 3 = Fclk / (CAN\_BAUD * 2 * (PRESC + 1)) - 3$ 
00536
00537 只要SEG1 + SEG2的总和固定, 波特率就正确。
00538 另外: SEG1 > SEG2, SEG1略大于SEG2
00539     SEG2 >= SJW
00540 采样点位置:  $(1 * TQ + Tseg1) / (1 * TQ + Tseg1 + Tseg2) = (SEG1 + 3) / (SEG1 + SEG2 + 4)$ , 应当在0.7~0.8之间
00541
00542 example:
00543     Fclock = 96000000; PRESC = 5; CAN_BAUD = 500000;
00544     SEG = 96000000 / ((5+1) * 2 * 500000) - 3 = 13
00545     SEG1 + SEG2 = 12 + 1 = 11 + 2 = ... = 8 + 5
00546 */
00547 static const u16 baud_cnt = 12 * 1;
00548 static const u16 baud_param[][6] = {
00549     // mclk baud presc sjw seg1 seg2
00550     // 96MHz 主频
00551     {96, 1000, 5, 1, 4, 1}, // 77.78%
00552     {96, 800, 5, 2, 5, 2}, // 72.73%
00553     {96, 666, 5, 1, 7, 2}, // 76.92%
00554     {96, 500, 5, 2, 10, 3}, // 76.47%
00555     {96, 400, 5, 2, 13, 4}, // 76.19%
00556     {96, 250, 11, 2, 10, 3}, // 76.47%
00557     {96, 200, 11, 2, 13, 4}, // 76.19%
00558     {96, 125, 23, 2, 10, 3}, // 76.47%
00559     {96, 100, 23, 2, 13, 4}, // 76.19%
00560     {96, 80, 29, 2, 13, 4}, // 76.19%
00561     {96, 50, 47, 2, 13, 4}, // 76.19%
00562     {96, 40, 59, 2, 13, 4}, // 76.19%
00563     // {96, 25, 95, 2, 13, 4}, // 76.19%
00564     // {96, 20, 119, 2, 13, 4}, // 76.19%
00565     // {96, 10, 119, 6, 29, 8}, // 78.05%
00566     // {96, 5, 119, 13, 61, 16}, // 79.01%
00567
00568     // 48MHz
00569     // {48, 1000, 1, 2, 7, 2}, // 76.92%
00570     // {48, 800, 2, 2, 5, 2}, // 72.73%
00571     // {48, 666, 2, 1, 7, 2}, // 76.92%
00572     // {48, 500, 2, 2, 10, 3}, // 76.47%
00573     // {48, 400, 2, 2, 13, 4}, // 76.19%
00574     // {48, 250, 5, 2, 10, 3}, // 76.47%
00575     // {48, 200, 5, 2, 13, 4}, // 76.19%
00576     // {48, 125, 11, 2, 10, 3}, // 76.47%
00577     // {48, 100, 11, 2, 13, 4}, // 76.19%
00578     // {48, 80, 14, 2, 13, 4}, // 76.19%
00579     // {48, 50, 23, 2, 13, 4}, // 76.19%
00580     // {48, 40, 29, 2, 13, 4}, // 76.19%
00581     // {48, 25, 47, 2, 13, 4}, // 76.19%
00582     // {48, 20, 59, 2, 13, 4}, // 76.19%
00583     // {48, 10, 59, 6, 29, 8}, // 78.05%
00584     // {48, 5, 59, 13, 61, 16}, // 79.91%
00585 };
00586
00587 static bool GetBaudParam(u32 baud, u8 *presc, u8 *sjw, u8 *seg1, u8 *seg2)
00588 {
00589     u16 i;
00590     u32 mclk_mega = HAL_CLK_GetMcuFreq() / 1000000;
00591     for (i = 0; i < baud_cnt; i++)
00592     {
00593         if (baud_param[i][0] == mclk_mega && baud_param[i][1] == baud)
00594         {
00595             *presc = baud_param[i][2];
00596             *sjw = baud_param[i][3];
00597             *seg1 = baud_param[i][4];
00598             *seg2 = baud_param[i][5];
00599             return true;
00600         }
00601     }
00602     return false;
00603 }
00604
00605 static void CAN_Write_TBUF(HAL_CAN_x can_x, u32 id, HAL_CAN_FRAME_x frame_x, u8 *msg, u8 len)
00606 {
00607     u8 frame_inf = 0;
00608     u8 ide = 0;
00609     u8 rtr = 0;
00610
00611

```

```

00612     if (len > 8)
00613     {
00614         len = 8;
00615     }
00616
00617     switch (frame_x)
00618     {
00619         case HAL_CAN_FRAME_IDE0_RTR0:
00620             ide = 0;
00621             rtr = 0;
00622             break;
00623         case HAL_CAN_FRAME_IDE0_RTR1:
00624             ide = 0;
00625             rtr = 1;
00626             break;
00627         case HAL_CAN_FRAME_IDE1_RTR0:
00628             ide = 1;
00629             rtr = 0;
00630             break;
00631         case HAL_CAN_FRAME_IDE1_RTR1:
00632             ide = 1;
00633             rtr = 1;
00634             break;
00635         default:
00636             break;
00637     }
00638
00639     frame_inf |= ide << 7; // 0标准/1扩展帧
00640     frame_inf |= rtr << 6; // 0,数据帧;1,遥控帧
00641     frame_inf |= len << 0; // 发送数据长度
00642     HAL_REG_WRITE(HAL_CAN[can_x]->TXRX0, frame_inf);
00643
00644     if (ide == 0)
00645     {
00646         id &= 0x7FF; // 标准帧, 11位ID
00647
00648         HAL_REG_WRITE(HAL_CAN[can_x]->TXRX1, (u8)(id >> 3)); // TX ID0
00649         HAL_REG_WRITE(HAL_CAN[can_x]->TXRX2, (u8)((id & 0X07) << 5)); // TX ID1
00650         HAL_REG_WRITE(HAL_CAN[can_x]->TXRX3, msg[0]); // TX DATA0
00651         HAL_REG_WRITE(HAL_CAN[can_x]->TXRX4, msg[1]); // TX DATA1
00652         HAL_REG_WRITE(HAL_CAN[can_x]->TXRX5, msg[2]); // TX DATA2
00653         HAL_REG_WRITE(HAL_CAN[can_x]->TXRX6, msg[3]); // TX DATA3
00654         HAL_REG_WRITE(HAL_CAN[can_x]->TXRX7, msg[4]); // TX DATA4
00655         HAL_REG_WRITE(HAL_CAN[can_x]->TXRX8, msg[5]); // TX DATA5
00656         HAL_REG_WRITE(HAL_CAN[can_x]->TXRX9, msg[6]); // TX DATA6
00657         HAL_REG_WRITE(HAL_CAN[can_x]->TXRXA, msg[7]); // TX DATA7
00658     }
00659     else
00660     {
00661         id &= 0x1FFFFFF; // 扩展帧, 29位ID
00662
00663         HAL_REG_WRITE(HAL_CAN[can_x]->TXRX1, (u8)(id >> 21)); // TX ID0
00664         HAL_REG_WRITE(HAL_CAN[can_x]->TXRX2, (u8)((id >> 13) & 0xFF)); // TX ID1
00665         HAL_REG_WRITE(HAL_CAN[can_x]->TXRX3, (u8)((id >> 5) & 0xFF)); // TX ID2
00666         HAL_REG_WRITE(HAL_CAN[can_x]->TXRX4, (u8)((id & 0xFF) << 3)); // TX ID3
00667         HAL_REG_WRITE(HAL_CAN[can_x]->TXRX5, msg[0]); // TX DATA0
00668         HAL_REG_WRITE(HAL_CAN[can_x]->TXRX6, msg[1]); // TX DATA1
00669         HAL_REG_WRITE(HAL_CAN[can_x]->TXRX7, msg[2]); // TX DATA2
00670         HAL_REG_WRITE(HAL_CAN[can_x]->TXRX8, msg[3]); // TX DATA3
00671         HAL_REG_WRITE(HAL_CAN[can_x]->TXRX9, msg[4]); // TX DATA4
00672         HAL_REG_WRITE(HAL_CAN[can_x]->TXRXA, msg[5]); // TX DATA5
00673         HAL_REG_WRITE(HAL_CAN[can_x]->TXRXB, msg[6]); // TX DATA6
00674         HAL_REG_WRITE(HAL_CAN[can_x]->TXRXC, msg[7]); // TX DATA7
00675     }
00676 }

```

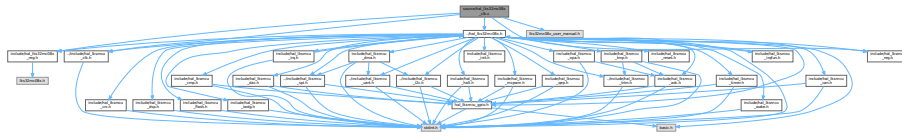
4.68 source/hal_lks32mc08x_clk.c 文件参考

```

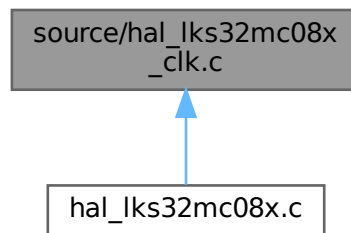
#include "../include/hal_lksmcu_clk.h"
#include "../hal_lks32mc08x.h"
#include "lks32mc08x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc08x_reg.h"

```

hal_lks32mc08x_clk.c 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



函数

- void [HAL_CLK_SetMcuFreq](#) ([HAL_CLK_FREQ_x](#) [FREQ_x](#))
设置主频
- [uint32_t](#) [HAL_CLK_GetMcuFreq](#) (void)
获取主频

变量

- static volatile [uint32_t](#) [HAL_CLK_mclk](#) = 96000000

4.68.1 函数说明

4.68.1.1 HAL_CLK_GetMcuFreq()

```
uint32_t HAL_CLK_GetMcuFreq (
    void )
```

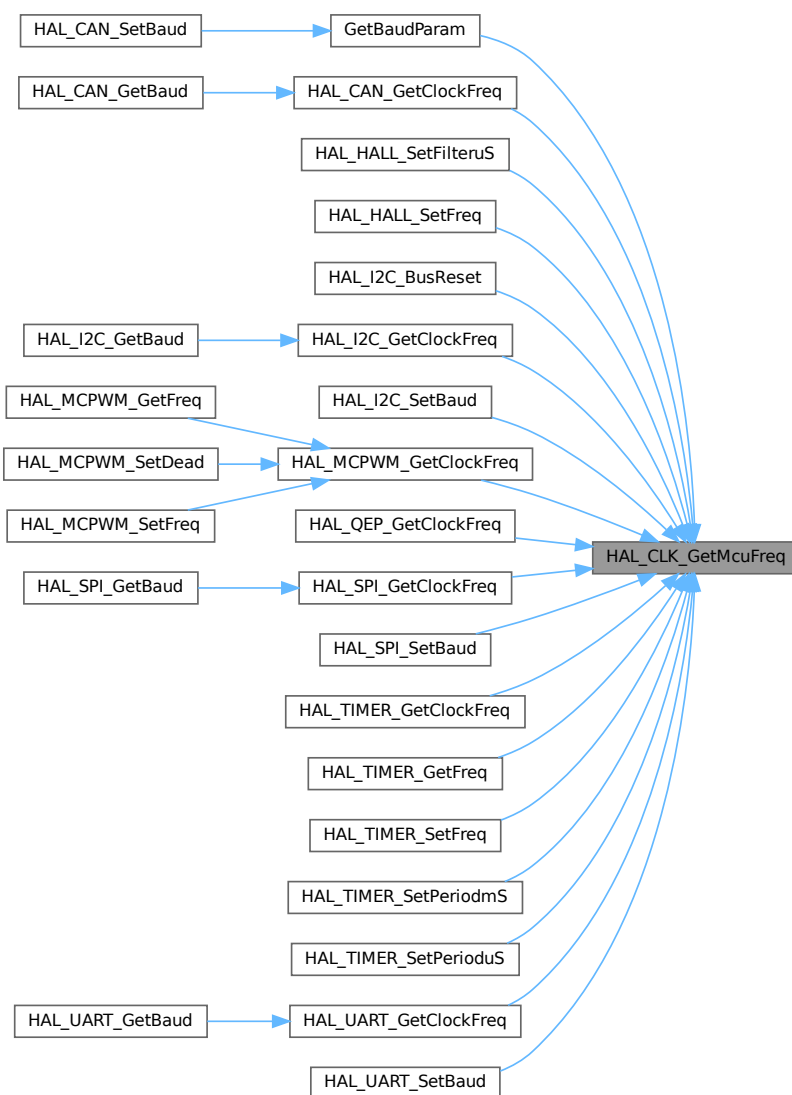
获取主频

返回

主频

在文件 [hal_lks32mc08x.clk.c](#) 第 81 行定义.

这是这个函数的调用关系图:



4.68.1.2 HAL_CLK_SetMcuFreq()

```
void HAL_CLK_SetMcuFreq (
    HAL_CLK_FREQ_x FREQ_x )
```

设置主频

参数

<i>FREQ</i> _x	主频

< 主频4Mhz

< 主频12Mhz

< 主频24Mhz

< 主频48Mhz

< 主频96Mhz

在文件 [hal_lks32mc08x.clk.c](#) 第 11 行定义.

这是这个函数的调用关系图:



4.68.2 变量说明

4.68.2.1 HAL_CLK_mclk

`volatile uint32_t HAL_CLK_mclk = 96000000 [static]`

在文件 [hal_lks32mc08x.clk.c](#) 第 6 行定义.

4.69 hal_lks32mc08x.clk.c

[浏览该文件的文档.](#)

```
00001 #include "../include/hal_lksmcu.clk.h"
00002 #include "../hal_lks32mc08x.h"
00003 #include "lks32mc08x.user_manual.h"
00004 #include "../include/hal_lksmcu.reg.h"
00005 #include "../include/hal_lks32mc08x.reg.h"
00006 static volatile uint32_t HAL_CLK_mclk = 96000000;
00011 void HAL_CLK_SetMcuFreq (HAL_CLK_FREQ_x FREQ_x)
00012 {
00013     uint32_t t_cnt;
00014     HAL_REG_SYS_UNLOCK();
00015
00016     switch (FREQ_x)
00017     {
00018     case HAL_CLK_FREQ_RC_4MHZ:
00019         HAL_CLK_mclk = 4000000;
00020         HAL_REG_WRITE_BIT_MASK_POS (SYS_CLK_CFG, SYS_CLK_CFG_CLK_DIV, 0x00);
00021         HAL_REG_RESET (SYS_CLK_CFG, SYS_CLK_CFG_CLK_SEL_BIT);
00022         for (t_cnt = 0; t_cnt < 4000; t_cnt++)
```

```

00023     {
00024         __NOP();
00025     }
00026     HAL_REG_RESET(SYS_AFE_REG5, SYS_AFE_REG5_PLLPDN_BIT);
00027     break;
00028     case HAL_CLK_FREQ_RC_12MHZ:
00029         HAL_CLK_mclk = 12000000;
00030         HAL_REG_SET(SYS_AFE_REG5, SYS_AFE_REG5_PLLPDN_BIT);
00031         for (t_cnt = 0; t_cnt < 200; t_cnt++)
00032         {
00033             __NOP();
00034         }
00035         HAL_REG_SET(SYS_CLK_CFG, SYS_CLK_CFG_CLK_SEL_BIT);
00036         HAL_REG_WRITE_BIT_MASK_POS(SYS_CLK_CFG, SYS_CLK_CFG_CLK_DIV, 0x01);
00037         break;
00038     case HAL_CLK_FREQ_RC_24MHZ:
00039         HAL_CLK_mclk = 24000000;
00040         HAL_REG_SET(SYS_AFE_REG5, SYS_AFE_REG5_PLLPDN_BIT);
00041         for (t_cnt = 0; t_cnt < 200; t_cnt++)
00042         {
00043             __NOP();
00044         }
00045         HAL_REG_SET(SYS_CLK_CFG, SYS_CLK_CFG_CLK_SEL_BIT);
00046         HAL_REG_WRITE_BIT_MASK_POS(SYS_CLK_CFG, SYS_CLK_CFG_CLK_DIV, 0x11);
00047         break;
00048     case HAL_CLK_FREQ_RC_48MHZ:
00049         HAL_CLK_mclk = 48000000;
00050         HAL_REG_SET(SYS_AFE_REG5, SYS_AFE_REG5_PLLPDN_BIT);
00051         for (t_cnt = 0; t_cnt < 200; t_cnt++)
00052         {
00053             __NOP();
00054         }
00055         HAL_REG_SET(SYS_CLK_CFG, SYS_CLK_CFG_CLK_SEL_BIT);
00056         HAL_REG_WRITE_BIT_MASK_POS(SYS_CLK_CFG, SYS_CLK_CFG_CLK_DIV, 0x55);
00057         break;
00058     case HAL_CLK_FREQ_RC_96MHZ:
00059     default:
00060         HAL_CLK_mclk = 96000000;
00061         HAL_REG_SET(SYS_AFE_REG5, SYS_AFE_REG5_PLLPDN_BIT);
00062         for (t_cnt = 0; t_cnt < 200; t_cnt++)
00063         {
00064             __NOP();
00065         }
00066         HAL_REG_SET(SYS_CLK_CFG, SYS_CLK_CFG_CLK_SEL_BIT);
00067         HAL_REG_WRITE_BIT_MASK_POS(SYS_CLK_CFG, SYS_CLK_CFG_CLK_DIV, 0xff);
00068         break;
00069     }
00070     for (t_cnt = 0; t_cnt < 200; t_cnt++)
00071     {
00072         __NOP();
00073     }
00074     HAL_REG_SYS_LOCK();
00075 }
00076
00081 uint32_t HAL_CLK_GetMcuFreq(void)
00082 {
00083     return HAL_CLK_mclk;
00084 }

```

4.70 source/hal_lks32mc08x_cmp.c 文件参考

```

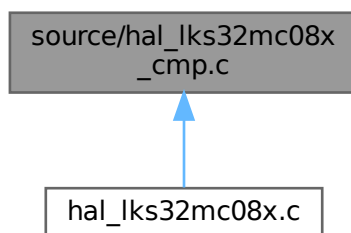
#include "../include/hal_lksmcu_cmp.h"
#include "../hal_lks32mc08x.h"
#include "lks32mc08x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc08x_reg.h"

```

hal_lks32mc08x_cmp.c 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了：



结构体

- **struct CMP_TypeDef**
ADC寄存器结构体句柄

函数

- **void HAL_CMP_Reset (HAL_CMP_x cmp_x)**
模块复位
- **void HAL_CMP_Enable (HAL_CMP_x cmp_x)**
模块打开
- **void HAL_CMP_Disable (HAL_CMP_x cmp_x)**
模块关闭
- **void HAL_CMP_SetChannel (HAL_CMP_x cmp_x, HAL_CMP_CHANNEL_IPx channel_p_x, HAL_CMP_CHANNEL_INx channel_n_x)**
模块关闭
- **uint32_t HAL_CMP_SetDigitalFilter (HAL_CMP_x cmp_x, uint32_t filter)**
获取CMP模块的数字滤波宽度
- **void HAL_CMP_SetMode (HAL_CMP_x cmp_x, HAL_CMP_MODE_x mode_x)**
设置CMP模块工作模式
- **void HAL_CMP_SetGpioOut (HAL_CMP_x cmp_x, HAL_GPIO_Px cmpgpio)**
设置CMP模块的GPIO
- **void HAL_CMP_SetGpioIn (HAL_CMP_x cmp_x, HAL_GPIO_Px cmpgpio)**
设置CMP模块的GPIO
- **uint8_t HAL_CMP_GetData (HAL_CMP_x cmp_x)**
获取CMP比较结果
- **void HAL_CMP_SetHysteresis (HAL_CMP_x cmp_x, HAL_CMP_HYS_x hys_x)**
设置比较器回差

4.70.1 函数说明

4.70.1.1 HAL_CMP_Disable()

```
void HAL_CMP_Disable (
    HAL_CMP_x cmp_x )
```

模块关闭

参数

<i>cmp</i> ↔ _x	使用的CMP模块编号

在文件 [hal_lks32mc08x_cmp.c](#) 第 56 行定义.

这是这个函数的调用关系图:



4.70.1.2 HAL_CMP.Enable()

```
void HAL_CMP_Enable (
    HAL_CMP_x cmp_x )
```

模块打开

参数

<i>cmp</i> ↔ _x	使用的CMP模块编号

在文件 [hal_lks32mc08x_cmp.c](#) 第 31 行定义.

4.70.1.3 HAL_CMP.GetData()

```
uint8_t HAL_CMP_GetData (
    HAL_CMP_x cmp_x )
```

获取CMP比较结果

参数

<i>cmp</i> ↔ _x	CMP模块编号

返回

比较结果

在文件 [hal_lks32mc08x_cmp.c](#) 第 200 行定义.

4.70.1.4 HAL_CMP_Reset()

```
void HAL_CMP_Reset (
    HAL_CMP_x cmp_x )
```

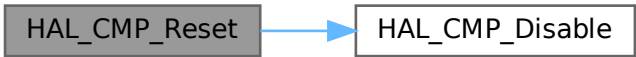
模块复位

参数

<i>cmp_x</i>	使用的CMP模块编号

在文件 [hal_lks32mc08x_cmp.c](#) 第 22 行定义.

函数调用图:



4.70.1.5 HAL_CMP_SetChannel()

```
void HAL_CMP_SetChannel (
    HAL_CMP_x cmp_x,
    HAL_CMP_CHANNEL_IPx channel_p_x,
    HAL_CMP_CHANNEL_INx channel_n_x )
```

模块关闭

参数

<i>cmp_x</i>	使用的CMP模块编号
<i>channel_p_x</i>	正端通道
<i>channel_n_x</i>	负端通道

在文件 [hal_lks32mc08x_cmp.c](#) 第 83 行定义.

4.70.1.6 HAL_CMP_SetDigitalFilter()

```
uint32_t HAL_CMP_SetDigitalFilter (
    HAL_CMP_x cmp_x,
    uint32_t filter )
```

获取CMP模块的数字滤波宽度

参数

<i>cmp_x</i>	CMP模块编号
<i>filter</i>	滤波宽度

返回

实际滤波宽度

在文件 [hal_lks32mc08x.cmp.c](#) 第 114 行定义.

4.70.1.7 HAL_CMP_SetGpioIn()

```
void HAL_CMP_SetGpioIn (
    HAL_CMP_x cmp_x,
    HAL_GPIO_Px cmpgpio )
```

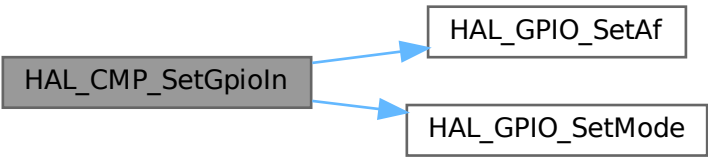
设置CMP模块的GPIO

参数

<i>cmp_x</i>	CMP模块编号
<i>cmpgpio</i>	GPIO编号

在文件 [hal_lks32mc08x.cmp.c](#) 第 188 行定义.

函数调用图:



4.70.1.8 HAL_CMP_SetGpioOut()

```
void HAL_CMP_SetGpioOut (
    HAL_CMP_x cmp_x,
    HAL_GPIO_Px cmpgpio )
```

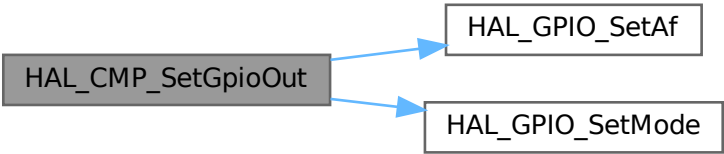
设置CMP模块的GPIO

参数

<i>cmp_x</i>	I2C模块编号
<i>cmpgpio</i>	GPIO编号

在文件 [hal_lks32mc08x_cmp.c](#) 第 176 行定义.

函数调用图:



4.70.1.9 HAL_CMP_SetHysteresis()

```
void HAL_CMP_SetHysteresis (
    HAL_CMP_x cmp_x,
    HAL_CMP_HYS_x hys_x )
```

设置比较器回差

参数

<i>cmp_x</i>	CMP模块编号(回差为所有CMP模块共用配置,与 <i>cmp_x</i> 无关,本参数保留暂不使用)
<i>hys_x</i>	回差选择

在文件 [hal_lks32mc08x_cmp.c](#) 第 229 行定义.

4.70.1.10 HAL_CMP_SetMode()

```
void HAL_CMP_SetMode (
    HAL_CMP_x cmp_x,
    HAL_CMP_MODE_x mode_x )
```

设置CMP模块工作模式

参数

<i>cmp_x</i>	CMP模块编号
<i>mode_x</i>	工作模式

在文件 `hal_lks32mc08x_cmp.c` 第 149 行定义。

4.71 hal_lks32mc08x_cmp.c

[浏览该文件的文档。](#)

```

00001 #include "../include/hal_lksmcu_cmp.h"
00002 #include "../hal_lks32mc08x.h"
00003 #include "lks32mc08x_user_manual.h"
00004 #include "../include/hal_lksmcu_reg.h"
00005 #include "../include/hal_lks32mc08x_reg.h"
00006
00010 typedef struct
00011 {
00012     volatile uint32_t IE;
00013     volatile uint32_t IF;
00014     volatile uint32_t TCLK;
00015     volatile uint32_t CFG;
00016     volatile uint32_t BLCWIN;
00017 } CMP_TypeDef;
00022 void HAL_CMP_Reset(HAL_CMP_x cmp_x)
00023 {
00024     HAL_CMP_Disable(cmp_x);
00025 }
00026
00031 void HAL_CMP_Enable(HAL_CMP_x cmp_x)
00032 {
00033     // 系统寄存器解锁
00034     HAL_REG_SYS_UNLOCK();
00035     switch (cmp_x)
00036     {
00037         case HAL_CMP_0:
00038             HAL_REG_SET(SYS_AFE_REG5, SYS_AFE_REG5_CMP0PDN_BIT);
00039             HAL_REG_SET(HAL_CMP[0]->CFG, CMP_CFG_CMP0_IN_EN_BIT);
00040             break;
00041         case HAL_CMP_1:
00042             HAL_REG_SET(SYS_AFE_REG5, SYS_AFE_REG5_CMP1PDN_BIT);
00043             HAL_REG_SET(HAL_CMP[0]->CFG, CMP_CFG_CMP1_IN_EN_BIT);
00044             break;
00045         default:
00046             break;
00047     }
00048     // 系统寄存器上锁
00049     HAL_REG_SYS_LOCK();
00050 }
00051
00056 void HAL_CMP_Disable(HAL_CMP_x cmp_x)
00057 {
00058     // 系统寄存器解锁
00059     HAL_REG_SYS_UNLOCK();
00060     switch (cmp_x)
00061     {
00062         case HAL_CMP_0:
00063             HAL_REG_RESET(SYS_AFE_REG5, SYS_AFE_REG5_CMP0PDN_BIT);
00064             HAL_REG_RESET(HAL_CMP[0]->CFG, CMP_CFG_CMP0_IN_EN_BIT);
00065             break;
00066         case HAL_CMP_1:
00067             HAL_REG_RESET(SYS_AFE_REG5, SYS_AFE_REG5_CMP1PDN_BIT);
00068             HAL_REG_RESET(HAL_CMP[0]->CFG, CMP_CFG_CMP1_IN_EN_BIT);
00069             break;
00070         default:
00071             break;
00072     }
00073     // 系统寄存器上锁
00074     HAL_REG_SYS_LOCK();
00075 }
00076
00083 void HAL_CMP_SetChannel(HAL_CMP_x cmp_x, HAL_CMP_CHANNEL_IPx channel_p_x, HAL_CMP_CHANNEL_INx channel_n_x)
00084 {
00085     switch (cmp_x)

```

```

00086     {
00087         case HAL_CMP_0:
00088             // 系统寄存器解锁
00089             HAL_REG_SYS_UNLOCK();
00090             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG3, SYS_AFE_REG3_CMP0_SELP, channel_p.x);
00091             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG3, SYS_AFE_REG3_CMP0_SELN, channel_n.x);
00092             // 系统寄存器上锁
00093             HAL_REG_SYS_LOCK();
00094             break;
00095         case HAL_CMP_1:
00096             // 系统寄存器解锁
00097             HAL_REG_SYS_UNLOCK();
00098             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG3, SYS_AFE_REG3_CMP1_SELP, channel_p.x);
00099             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG3, SYS_AFE_REG3_CMP1_SELN, channel_n.x);
00100             // 系统寄存器上锁
00101             HAL_REG_SYS_LOCK();
00102             break;
00103         default:
00104             break;
00105     }
00106 }
00107
00114 uint32_t HAL_CMP_SetDigitalFilter(HAL_CMP_x cmpx, uint32_t filter)
00115 {
00116     (void)cmpx;
00117     if (filter == 0)
00118     {
00119
00120         HAL_REG_RESET(HAL_CMP[0]->TCLK, CMP_TCLK_CLK_EN_BIT | CMP_TCLK_FIL_CLK_DIV16_MASK |
00121             CMP_TCLK_FIL_CLK_DIV1248_MASK);
00122     }
00123     else
00124     {
00125         uint16_t div_2 = 0;
00126         for (; filter > 0xf; )
00127         {
00128             filter >>= 1;
00129             div_2++;
00130         }
00131         if (div_2 > CMP_TCLK_FIL_CLK_DIV1248_MAX)
00132         {
00133             div_2 = CMP_TCLK_FIL_CLK_DIV1248_MAX;
00134             filter = CMP_TCLK_FIL_CLK_DIV16_MAX;
00135         }
00136         HAL_REG_SET(HAL_CMP[0]->TCLK, CMP_TCLK_CLK_EN_BIT);
00137         HAL_REG_WRITE_BIT_MASK_POS(HAL_CMP[0]->TCLK, CMP_TCLK_FIL_CLK_DIV1248, div_2);
00138         HAL_REG_WRITE_BIT_MASK_POS(HAL_CMP[0]->TCLK, CMP_TCLK_FIL_CLK_DIV16, filter);
00139         filter = filter << div_2;
00140     }
00141     return filter;
00142 }
00143
00149 void HAL_CMP_SetMode(HAL_CMP_x cmp_x, HAL_CMP_MODE_x mode_x)
00150 {
00151     switch (mode_x)
00152     {
00153         case HAL_CMP_MODE_RISING_EDGE: // 上升沿触发
00154             HAL_REG_SET(HAL_CMP[cmp_x]->CFG, CMP_CFG_CMP0_IRQ_TRIG_BIT);
00155             HAL_REG_RESET(HAL_CMP[cmp_x]->CFG, CMP_CFG_CMP0_POL_BIT);
00156             break;
00157         case HAL_CMP_MODE_FALLING_EDGE: // 下降沿触发
00158             HAL_REG_SET(HAL_CMP[cmp_x]->CFG, CMP_CFG_CMP0_IRQ_TRIG_BIT);
00159             HAL_REG_SET(HAL_CMP[cmp_x]->CFG, CMP_CFG_CMP0_POL_BIT);
00160             break;
00161         case HAL_CMP_MODE_HIGH_LEVEL: // 高电平有效
00162             HAL_REG_RESET(HAL_CMP[cmp_x]->CFG, CMP_CFG_CMP0_IRQ_TRIG_BIT);
00163             HAL_REG_RESET(HAL_CMP[cmp_x]->CFG, CMP_CFG_CMP0_POL_BIT);
00164             break;
00165         case HAL_CMP_MODE_LOW_LEVEL: // 低电平有效
00166             HAL_REG_RESET(HAL_CMP[cmp_x]->CFG, CMP_CFG_CMP0_IRQ_TRIG_BIT);
00167             HAL_REG_SET(HAL_CMP[cmp_x]->CFG, CMP_CFG_CMP0_POL_BIT);
00168             break;
00169     }
00170 }
00176 void HAL_CMP_SetGpioOut(HAL_CMP_x cmp_x, HAL_GPIO_Px cmp_gpio)
00177 {
00178     (void)cmp_x;
00179     HAL_GPIO_SetAf(cmp_gpio, HAL_GPIO_AF_CMP_OUT);
00180     HAL_GPIO_SetMode(cmp_gpio, HAL_GPIO_MODE_OUT);
00181 }
00182
00188 void HAL_CMP_SetGpioIn(HAL_CMP_x cmp_x, HAL_GPIO_Px cmp_gpio)
00189 {
00190     (void)cmp_x;
00191     HAL_GPIO_SetAf(cmp_gpio, HAL_GPIO_AF_CMP);
00192     HAL_GPIO_SetMode(cmp_gpio, HAL_GPIO_MODE_AFE);

```

```

00193 }
00194
00200 uint8_t HAL_CMP_GetData(HAL_CMP_x cmp_x)
00201 {
00202     uint32_t flg = 0;
00203     switch (cmp_x)
00204     {
00205         case HAL_CMP_0:
00206             flg = HAL_REG_READ_BIT(SYS_AFE_CMP, SYS_AFE_CMP_CMP0_RESULT_BIT);
00207             break;
00208         case HAL_CMP_1:
00209             flg = HAL_REG_READ_BIT(SYS_AFE_CMP, SYS_AFE_CMP_CMP1_RESULT_BIT);
00210             break;
00211         default:
00212             break;
00213     }
00214     if (flg == 0)
00215     {
00216         return 0;
00217     }
00218     else
00219     {
00220         return 1;
00221     }
00222 }
00223
00229 void HAL_CMP_SetHysteresis(HAL_CMP_x cmp_x, HAL_CMP_HYS_x hys_x)
00230 {
00231     (void)cmp_x;
00232     // 系统寄存器解锁
00233     HAL_REG_SYS_UNLOCK();
00234     HAL_REG_IF_WRITE(SYS_AFE_REG3, SYS_AFE_REG3_CMP_HYS_BIT, hys_x);
00235     // 系统寄存器上锁
00236     HAL_REG_SYS_LOCK();
00237 }

```

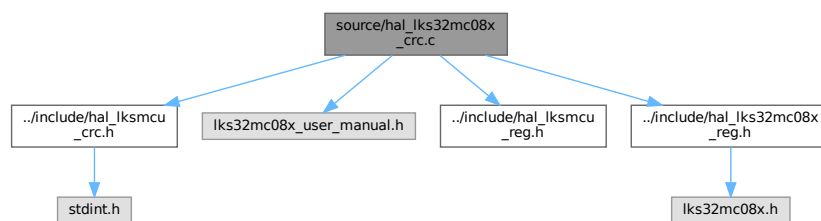
4.72 source/hal_lks32mc08x_crc.c 文件参考

```

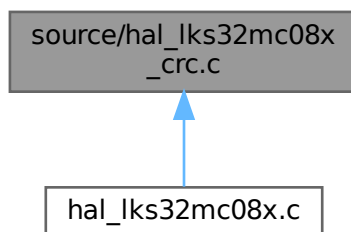
#include "../include/hal_lksmcu_crc.h"
#include "lks32mc08x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc08x_reg.h"

```

hal_lks32mc08x_crc.c 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



结构体

- struct [CRC_Model_Params](#)

枚举

- enum [CRC_Width](#) { [CRC_WIDTH_8](#) , [CRC_WIDTH_16](#) , [CRC_WIDTH_32](#) }

函数

- static uint32_t [HAL_CRC_Reflect](#) (uint32_t data, uint8_t width)
- void [HAL_CRC_Reset](#) ([HAL_CRC_x](#) CRC_x)
*CRC*模块复位
- void [HAL_CRC_Enable](#) ([HAL_CRC_x](#) CRC_x)
*CRC*模块使能
- void [HAL_CRC_Disable](#) ([HAL_CRC_x](#) CRC_x)
*CRC*模块失能
- uint32_t [HAL_CRC_Calc](#) ([HAL_CRC_x](#) CRC_x, [HAL_CRC_MODEL_x](#) model, uint8_t *data, uint32_t len)
*CRC*计算

变量

- static const [CRC_Model_Params](#) [crcModels](#) []

4.72.1 枚举类型说明

4.72.1.1 CRC_Width

enum [CRC_Width](#)

枚举值

CRC_WIDTH_8	
CRC_WIDTH_16	
CRC_WIDTH_32	

在文件 [hal_lks32mc08x_crc.c](#) 第 7 行定义.

4.72.2 函数说明

4.72.2.1 HAL_CRC_Calc()

```
uint32_t HAL_CRC_Calc (
    HAL_CRC_TypeDef * CRCx,
    HAL_CRC_MODEL_TypeDef * model,
    uint8_t * data,
    uint32_t len )
```

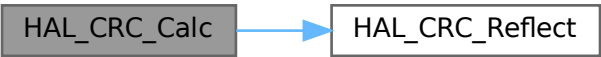
CRC计算

参数

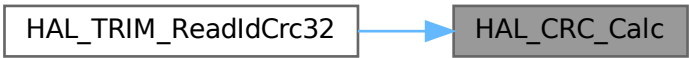
<i>CRCx</i>	CRC模块编号
<i>model</i>	CRC模型
<i>data</i>	数据地址
<i>len</i>	数据长度

在文件 [hal_lks32mc08x_crc.c](#) 第 90 行定义.

函数调用图:



这是这个函数的调用关系图:



4.72.2.2 HAL_CRC.Disable()

```
void HAL_CRC_Disable (
    HAL_CRC_x CRC_x )
```

CRC模块失能

参数

CRC↔ _x	CRC模块编号

在文件 [hal_lks32mc08x_crc.c](#) 第 78 行定义.

4.72.2.3 HAL_CRC.Enable()

```
void HAL_CRC_Enable (
    HAL_CRC_x CRC_x )
```

CRC模块使能

参数

CRC↔ _x	CRC模块编号

在文件 [hal_lks32mc08x_crc.c](#) 第 69 行定义.

4.72.2.4 HAL_CRC.Reflect()

```
static uint32_t HAL_CRC_Reflect (
    uint32_t data,
    uint8_t width ) [static]
```

在文件 [hal_lks32mc08x_crc.c](#) 第 42 行定义.

这是这个函数的调用关系图:



4.72.2.5 HAL_CRC.Reset()

```
void HAL_CRC_Reset (
    HAL_CRC_x CRC_x )
```

CRC模块复位

参数

CRC↔ _x	CRC模块编号

在文件 [hal_lks32mc08x_crc.c](#) 第 60 行定义.

4.72.3 变量说明

4.72.3.1 crcModels

```
const CRC_Model_Params crcModels[] [static]
```

```
初始值:
= {
    {0x07, 0x00, 0x00, 0, 0, CRC_WIDTH_8},
    {0x07, 0x00, 0x55, 0, 0, CRC_WIDTH_8},
    {0x07, 0xFF, 0x00, 1, 1, CRC_WIDTH_8},
    {0x31, 0x00, 0x00, 1, 1, CRC_WIDTH_8},
    {0x8005, 0x0000, 0x0000, 1, 1, CRC_WIDTH_16},
    {0x8005, 0x0000, 0xFFFF, 1, 1, CRC_WIDTH_16},
    {0x8005, 0xFFFF, 0xFFFF, 1, 1, CRC_WIDTH_16},
    {0x8005, 0xFFFF, 0x0000, 1, 1, CRC_WIDTH_16},
    {0x1021, 0x0000, 0x0000, 1, 1, CRC_WIDTH_16},
    {0x1021, 0xFFFF, 0x0000, 0, 0, CRC_WIDTH_16},
    {0x1021, 0xFFFF, 0xFFFF, 1, 1, CRC_WIDTH_16},
    {0x1021, 0x0000, 0x0000, 0, 0, CRC_WIDTH_16},
    {0x3D65, 0x0000, 0xFFFF, 1, 1, CRC_WIDTH_16},
    {0x04C11DB7, 0xFFFFFFFF, 0xFFFFFFFF, 1, 1, CRC_WIDTH_32},
    {0x04C11DB7, 0xFFFFFFFF, 0x00000000, 0, 0, CRC_WIDTH_32},
}
```

在文件 [hal_lks32mc08x_crc.c](#) 第 24 行定义.

4.73 hal_lks32mc08x_crc.c

浏览该文件的文档。

```

00001 #include "../include/hal_lksmcu_crc.h"
00002 #include "lks32mc08x_user_manual.h"
00003 #include "../include/hal_lksmcu_reg.h"
00004 #include "../include/hal_lks32mc08x_reg.h"
00005
00006 // 仅内部使用
00007 typedef enum
00008 {
00009     CRC_WIDTH_8,
00010     CRC_WIDTH_16,
00011     CRC_WIDTH_32
00012 } CRC_Width;
00013 // 仅内部使用
00014 typedef struct
00015 {
00016     uint32_t polynomial; // 公式
00017     uint32_t initValue; // 初始值
00018     uint32_t xorOut; // 结果异或值
00019     uint8_t reflectIn; // 输入反转使能
00020     uint8_t reflectOut; // 输出反转使能
00021     CRC_Width width; // 数据宽度
00022 } CRC_Model_Params;
00023 // 仅内部使用
00024 static const CRC_Model_Params crcModels[] = {
00025     {0x07, 0x00, 0x00, 0, 0, CRC_WIDTH_8}, // CRC8
00026     {0x07, 0x00, 0x55, 0, 0, CRC_WIDTH_8}, // CRC8_ITU
00027     {0x07, 0xFF, 0x00, 1, 1, CRC_WIDTH_8}, // CRC8_ROHC
00028     {0x31, 0x00, 0x00, 1, 1, CRC_WIDTH_8}, // CRC8_MAXIMx
00029     {0x8005, 0x0000, 0x0000, 1, 1, CRC_WIDTH_16}, // CRC16_IBMx1
00030     {0x8005, 0x0000, 0xFFFF, 1, 1, CRC_WIDTH_16}, // CRC16_MAXIM
00031     {0x8005, 0xFFFF, 0xFFFF, 1, 1, CRC_WIDTH_16}, // CRC16_USBx1
00032     {0x8005, 0xFFFF, 0x0000, 1, 1, CRC_WIDTH_16}, // CRC16_MODBU
00033     {0x1021, 0x0000, 0x0000, 1, 1, CRC_WIDTH_16}, // CRC16_CCITT
00034     {0x1021, 0xFFFF, 0x0000, 0, 0, CRC_WIDTH_16}, // CRC16_CCITT_FALSE
00035     {0x1021, 0xFFFF, 0xFFFF, 1, 1, CRC_WIDTH_16}, // CRC16_X25
00036     {0x1021, 0x0000, 0x0000, 0, 0, CRC_WIDTH_16}, // CRC16_XMODEM
00037     {0x3D65, 0x0000, 0xFFFF, 1, 1, CRC_WIDTH_16}, // CRC16_DNP
00038     {0x04C11DB7, 0xFFFFFFFF, 0xFFFFFFFF, 1, 1, CRC_WIDTH_32}, // CRC32
00039     {0x04C11DB7, 0xFFFFFFFF, 0x00000000, 0, 0, CRC_WIDTH_32}, // CRC32_MPEG2
00040 };
00041 // 仅内部使用
00042 static uint32_t HAL_CRC_Reflect(uint32_t data, uint8_t width)
00043 {
00044     uint32_t result = 0;
00045     uint8_t i;
00046     for (i = 0; i < width; i++)
00047     {
00048         if (data & (1 << i))
00049         {
00050             result |= (1 << (width - 1 - i));
00051         }
00052     }
00053     return result;
00054 }
00055
00060 void HAL_CRC_Reset(HAL_CRC_x CRC_x)
00061 {
00062     (void)CRC_x;
00063 }
00064
00069 void HAL_CRC_Enable(HAL_CRC_x CRC_x)
00070 {
00071     (void)CRC_x;
00072 }
00073
00078 void HAL_CRC_Disable(HAL_CRC_x CRC_x)
00079 {
00080     (void)CRC_x;
00081 }
00082
00090 uint32_t HAL_CRC_Calc(HAL_CRC_x CRC_x, HAL_CRC_MODEL_x model, uint8_t *data, uint32_t len)
00091 {
00092     const CRC_Model_Params *params = &crcModels[model];
00093     uint32_t crc = params->initValue;
00094     uint8_t width_bytes = (params->width == CRC_WIDTH_8) ? 1 : (params->width ==
CRC_WIDTH_16) ? 2
00095     : 4;
00096     uint8_t width_bits = width_bytes * 8;
00097     uint32_t top_bit = 1 << (width_bits - 1);
00098     uint32_t mask = (width_bits < 32) ? (uint32_t)((1 << width_bits) - 1) :
0xFFFFFFFF;

```

```

00099     uint32_t i;
00100     (void)CRC_x;
00101     if (model >= sizeof(crcModels) / sizeof(crcModels[0]))
00102     {
00103         return 0; // 无效模型
00104     }
00105
00106     for (i = 0; i < len; i++)
00107     {
00108         uint32_t j;
00109         uint8_t byte = data[i];
00110         if (params->reflectIn)
00111         {
00112             byte = HAL_CRC_Reflect(byte, 8);
00113         }
00114
00115         // 根据CRC宽度处理输入
00116         if (params->width == CRC_WIDTH_8)
00117         {
00118             crc ^= (uint32_t)byte;
00119         }
00120         else if (params->width == CRC_WIDTH_16)
00121         {
00122             crc ^= (uint32_t)byte << 8; // 假设数据是大端序输入
00123         }
00124         else
00125         {
00126             crc ^= (uint32_t)byte << 24;
00127         }
00128         for (j = 0; j < 8; j++)
00129         {
00130             if (crc & top_bit)
00131             {
00132                 crc = (crc << 1) ^ params->polynomial;
00133             }
00134             else
00135             {
00136                 crc <<= 1;
00137             }
00138             crc &= mask; // 确保只保留有效位
00139         }
00140     }
00141
00142     if (params->reflectOut)
00143     {
00144         crc = HAL_CRC_Reflect(crc, width_bits);
00145     }
00146
00147     crc ^= params->xorOut;
00148
00149     // 根据宽度返回适当大小的值
00150     if (params->width == CRC_WIDTH_8)
00151     {
00152         return crc & 0xFF;
00153     }
00154     else if (params->width == CRC_WIDTH_16)
00155     {
00156         return crc & 0xFFFF;
00157     }
00158     return crc;
00159 }

```

4.74 source/hal_lks32mc08x_dac.c 文件参考

```

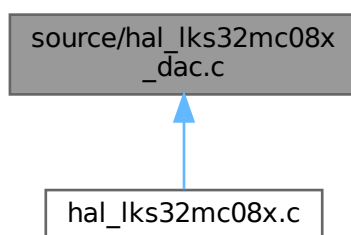
#include "../include/hal_lksmcu_dac.h"
#include "../include/hal_lksmcu_trim.h"
#include "../hal_lks32mc08x.h"
#include "lks32mc08x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc08x_reg.h"

```

hal_lks32mc08x_dac.c 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



函数

- void **HAL_DAC_Reset** (HAL_DAC_x dac_x)
DAC模块复位
- void **HAL_DAC_Enable** (HAL_DAC_x dac_x)
DAC模块使能
- void **HAL_DAC_Disable** (HAL_DAC_x dac_x)
DAC模块失能
- uint16_t **HAL_DAC_GetOutValMax** (HAL_DAC_x dac_x)
获取DAC模块输出最大值
- void **HAL_DAC_SetOutVal** (HAL_DAC_x dac_x, uint16_t val)
设置DAC模块输出值
- void **HAL_DAC_SetRange** (HAL_DAC_x dac_x, HAL_DAC_RANGE_x range)
设置DAC模块输出量程
- float **HAL_DAC_GetRange** (HAL_DAC_x dac_x)
获取DAC模块输出量程
- void **HAL_DAC_SetOutVoltageV** (HAL_DAC_x dac_x, uint16_t vol_mV)
设置DAC模块输出电压
- void **HAL_DAC_Out2GpioEnable** (HAL_DAC_x dac_x)
输出到GPIO使能
- void **HAL_DAC_Out2GpioDisable** (HAL_DAC_x dac_x)
输出到GPIO关闭
- void **HAL_DAC_SetGpioOut** (HAL_DAC_x dac_x, HAL_GPIO_Px px)
设置DAC输出到GPIO

4.74.1 函数说明

4.74.1.1 HAL_DAC_Disable()

```
void HAL_DAC_Disable (
    HAL_DAC_x dac_x )
```

DAC模块失能

参数

dac_x	DAC模块

在文件 [hal_lks32mc08x_dac.c](#) 第 32 行定义.

这是这个函数的调用关系图:



4.74.1.2 HAL_DAC_Enable()

```
void HAL_DAC_Enable (
    HAL_DAC_x dac_x )
```

DAC模块使能

参数

dac_x	DAC模块

在文件 [hal_lks32mc08x_dac.c](#) 第 19 行定义.

4.74.1.3 HAL_DAC_GetOutValMax()

```
uint16_t HAL_DAC_GetOutValMax (
    HAL_DAC_x dac_x )
```

获取DAC模块输出最大值

参数

dac_{-x}	DAC模块

返回

DAC输出最大值

在文件 [hal_lks32mc08x_dac.c](#) 第 46 行定义.

4.74.1.4 HAL_DAC_GetRange()

```
float HAL_DAC_GetRange (
    HAL_DAC_x dac_x )
```

获取DAC模块输出量程

参数

dac_{-x}	DAC模块

返回

输出量程

在文件 [hal_lks32mc08x_dac.c](#) 第 95 行定义.

4.74.1.5 HAL_DAC_Out2GpioDisable()

```
void HAL_DAC_Out2GpioDisable (
    HAL_DAC_x dac_x )
```

输出到GPIO关闭

参数

dac_{-x}	DAC模块

在文件 [hal_lks32mc08x_dac.c](#) 第 181 行定义.

4.74.1.6 HAL_DAC_Out2GpioEnable()

```
void HAL_DAC_Out2GpioEnable (
    HAL_DAC_x dac_x )
```

输出到GPIO使能

参数

<i>dac</i> ↔ <i>_x</i>	DAC模块

在文件 [hal_lks32mc08x_dac.c](#) 第 162 行定义.

4.74.1.7 HAL_DAC_Reset()

```
void HAL_DAC_Reset (
    HAL_DAC_x dac_x )
```

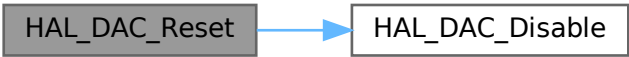
DAC模块复位

参数

<i>dac</i> ↔ <i>_x</i>	DAC模块

在文件 [hal_lks32mc08x_dac.c](#) 第 11 行定义.

函数调用图:



4.74.1.8 HAL_DAC_SetGpioOut()

```
void HAL_DAC_SetGpioOut (
    HAL_DAC_x dac_x,
    HAL_GPIO_Px px )
```

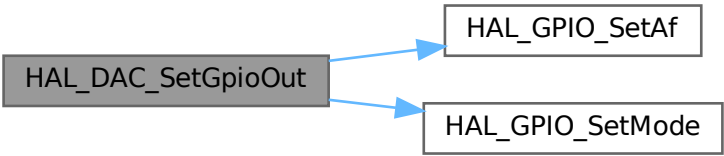
设置DAC输出到GPIO

参数

<i>dac</i> ↔ <i>_x</i>	DAC模块
<i>px</i>	GPIO

在文件 `hal_lks32mc08x.dac.c` 第 202 行定义.

函数调用图:



4.74.1.9 HAL_DAC.SetOutVal()

```
void HAL_DAC_SetOutVal (
    HAL_DAC_x dac_x,
    uint16_t val )
```

设置DAC模块输出值

参数

<i>dac_x</i>	DAC模块
<i>val</i>	DAC输出值

在文件 `hal_lks32mc08x.dac.c` 第 56 行定义.

这是这个函数的调用关系图:



4.74.1.10 HAL_DAC.SetOutVoltageMv()

```
void HAL_DAC_SetOutVoltageMv (
    HAL_DAC_x dac_x,
    uint16_t vol_mv )
```

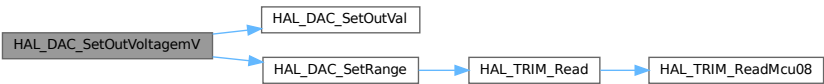
设置DAC模块输出电压

参数

<i>dac_x</i>	DAC模块
<i>vol_mV</i>	输出电压，单位mV

在文件 [hal_lks32mc08x_dac.c](#) 第 120 行定义.

函数调用图:



4.74.1.11 HAL_DAC_SetRange()

```
void HAL_DAC_SetRange (
    HAL_DAC_↔_x dac_x,
    HAL_DAC_RANGE_↔_x range )
```

设置DAC模块输出量程

参数

<i>dac↔_x</i>	DAC模块
<i>range</i>	输出量程

在文件 [hal_lks32mc08x_dac.c](#) 第 66 行定义.

函数调用图:



这是这个函数的调用关系图:



4.75 hal_lks32mc08x_dac.c

[浏览该文件的文档。](#)

```

00001 #include "../include/hal_lksmcu_dac.h"
00002 #include "../include/hal_lksmcu_trim.h"
00003 #include "../hal_lks32mc08x.h"
00004 #include "lks32mc08x_user_manual.h"
00005 #include "../include/hal_lksmcu_reg.h"
00006 #include "../include/hal_lks32mc08x_reg.h"
00011 void HAL_DAC_Reset(HAL_DAC_x dac_x)
00012 {
00013     HAL_DAC_Disable(dac_x);
00014 }
00019 void HAL_DAC_Enable(HAL_DAC_x dac_x)
00020 {
00021     (void)dac_x;
00022     // 系统寄存器解锁
00023     HAL_REG_SYS_UNLOCK();
00024     HAL_REG_SET(SYS_AFE_REG5, SYS_AFE_REG5_DAC12BPDN_BIT);
00025     // 系统寄存器上锁
00026     HAL_REG_SYS_LOCK();
00027 }
00032 void HAL_DAC_Disable(HAL_DAC_x dac_x)
00033 {
00034     (void)dac_x;
00035     // 系统寄存器解锁
00036     HAL_REG_SYS_UNLOCK();
00037     HAL_REG_RESET(SYS_AFE_REG5, SYS_AFE_REG5_DAC12BPDN_BIT);
00038     // 系统寄存器上锁
00039     HAL_REG_SYS_LOCK();
00040 }
00046 uint16_t HAL_DAC_GetOutValMax(HAL_DAC_x dac_x)
00047 {
00048     (void)dac_x;
00049     return SYS_AFE_DAC_DAC_IN_MAX;
00050 }
00056 void HAL_DAC_SetOutVal(HAL_DAC_x dac_x, uint16_t val)
00057 {
00058     (void)dac_x;
00059     HAL_REG_WRITE(SYS_AFE_DAC, val);
00060 }
00066 void HAL_DAC_SetRange(HAL_DAC_x dac_x, HAL_DAC_RANGE_x range)
00067 {
00068     (void)dac_x;
00069     switch (range)
00070     {
00071     case HAL_DAC_RANGE_1.2:
00072         HAL_REG_WRITE(SYS_AFE_DAC_DC, HAL_TRIM_Read(0x00000344));
00073         HAL_REG_WRITE(SYS_AFE_DAC_AMC, HAL_TRIM_Read(0x00000334));
00074         break;
00075     case HAL_DAC_RANGE_3.0V:
00076         HAL_REG_WRITE(SYS_AFE_DAC_DC, HAL_TRIM_Read(0x00000340));
00077         HAL_REG_WRITE(SYS_AFE_DAC_AMC, HAL_TRIM_Read(0x00000330));
00078         break;
00079     case HAL_DAC_RANGE_4.85V:
00080         HAL_REG_WRITE(SYS_AFE_DAC_DC, HAL_TRIM_Read(0x00000348));
00081         HAL_REG_WRITE(SYS_AFE_DAC_AMC, HAL_TRIM_Read(0x00000338));
00082         break;
00083     default:
00084         break;
00085     }
00086     HAL_REG_SYS_UNLOCK();
00087     HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG1, SYS_AFE_REG1_DAC_GAIN, range);
00088     HAL_REG_SYS_UNLOCK();
00089 }
00095 float HAL_DAC_GetRange(HAL_DAC_x dac_x)
00096 {
00097     float val = 0;
00098     (void)dac_x;
00099     switch (HAL_REG_READ_BIT_MASK_POS(SYS_AFE_REG1, SYS_AFE_REG1_DAC_GAIN))
00100     {
00101     case 0:
00102         val = 3.0f;
00103         break;
00104     case 1:
00105         val = 1.2f;
00106         break;
00107     case 2:
00108         val = 4.85f;
00109         break;
00110     default:
00111         break;
00112     }
00113     return val;
00114 }

```

```

00120 void HAL_DAC_SetOutVtagemV(HAL_DAC_x dac_x, uint16_t vol_mV)
00121 {
00122     uint32_t val;
00123     (void)dac_x;
00124     if (vol_mV < 1000)
00125     {
00126         val = vol_mV;
00127         val = val * SYS_AFE_DAC_DAC_IN_MAX / 1200;
00128         if (val > SYS_AFE_DAC_DAC_IN_MAX)
00129         {
00130             val = SYS_AFE_DAC_DAC_IN_MAX;
00131         }
00132         HAL_DAC_SetRange(dac_x, HAL_DAC_RANGE_1_2);
00133         HAL_DAC_SetOutVal(dac_x, val);
00134     }
00135     else if (vol_mV < 2500)
00136     {
00137         val = vol_mV;
00138         val = val * SYS_AFE_DAC_DAC_IN_MAX / 3000;
00139         if (val > SYS_AFE_DAC_DAC_IN_MAX)
00140         {
00141             val = SYS_AFE_DAC_DAC_IN_MAX;
00142         }
00143         HAL_DAC_SetRange(dac_x, HAL_DAC_RANGE_3_0V);
00144         HAL_DAC_SetOutVal(dac_x, val);
00145     }
00146     else
00147     {
00148         val = vol_mV;
00149         val = val * SYS_AFE_DAC_DAC_IN_MAX / 4850;
00150         if (val > SYS_AFE_DAC_DAC_IN_MAX)
00151         {
00152             val = SYS_AFE_DAC_DAC_IN_MAX;
00153         }
00154         HAL_DAC_SetRange(dac_x, HAL_DAC_RANGE_4_85V);
00155         HAL_DAC_SetOutVal(dac_x, val);
00156     }
00157 }
00162 void HAL_DAC_Out2GpioEnable(HAL_DAC_x dac_x)
00163 {
00164     // 系统寄存器解锁
00165     HAL_REG_SYS_UNLOCK();
00166     switch (dac_x)
00167     {
00168         case HAL_DAC_0:
00169             HAL_REG_SET(SYS_AFE_REG3, SYS_AFE_REG3_DACOUT_EN_BIT);
00170             break;
00171         default:
00172             break;
00173     }
00174     // 系统寄存器上锁
00175     HAL_REG_SYS_LOCK();
00176 }
00181 void HAL_DAC_Out2GpioDisable(HAL_DAC_x dac_x)
00182 {
00183     // 系统寄存器解锁
00184     HAL_REG_SYS_UNLOCK();
00185     switch (dac_x)
00186     {
00187         case HAL_DAC_0:
00188             HAL_REG_SET(SYS_AFE_REG3, SYS_AFE_REG3_DACOUT_EN_BIT);
00189             break;
00190         default:
00191             break;
00192     }
00193     // 系统寄存器上锁
00194     HAL_REG_SYS_LOCK();
00195 }
00196
00202 void HAL_DAC_SetGpioOut(HAL_DAC_x dac_x, HAL_GPIO_Px px)
00203 {
00204     (void)dac_x;
00205
00206     HAL_GPIO_SetAf(px, HAL_GPIO_AF_DAC);
00207     HAL_GPIO_SetMode(px, HAL_GPIO_MODE_AFE);
00208 }

```

4.76 source/hal_lks32mc08x_dma.c 文件参考

```

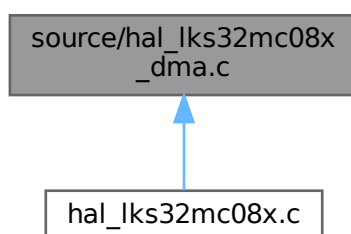
#include "../include/hal_lksmcu_dma.h"
#include "../hal_lks32mc08x.h"

```

```
#include "lks32mc08x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc08x_reg.h"
hal_lks32mc08x_dma.c 的引用(Include)关系图:
```



此图展示该文件直接或间接的被哪些文件引用了:



函数

- void [HAL_DMA_Reset](#) ([HAL_DMA_x](#) DMA_x)
- void [HAL_DMA_Enable](#) ([HAL_DMA_x](#) DMA_x)
- void [HAL_DMA_Disable](#) ([HAL_DMA_x](#) DMA_x)
- void [HAL_DMA_MemCopy](#) ([HAL_DMA_x](#) DMA_x, void *start, void *end, uint32_t len)
- void [HAL_DMA_UartTx](#) ([HAL_DMA_x](#) DMA_x, [HAL_UART_x](#) UART_x, uint8_t *start, uint32_t len)
- void [HAL_DMA_SpiTx](#) ([HAL_DMA_x](#) DMA_x, [HAL_SPI_x](#) SPI_x, uint8_t *start, uint32_t len)
- void [HAL_DMA_I2cTx](#) ([HAL_DMA_x](#) DMA_x, [HAL_I2C_x](#) I2Cx, uint8_t *start, uint32_t len)
- void [HAL_DMA_UartRx](#) ([HAL_DMA_x](#) DMA_x, [HAL_UART_x](#) UART_x, uint8_t *start, uint32_t len)
- void [HAL_DMA_SpiRx](#) ([HAL_DMA_x](#) DMA_x, [HAL_SPI_x](#) SPI_x, uint8_t *start, uint32_t len)
- void [HAL_DMA_I2cRx](#) ([HAL_DMA_x](#) DMA_x, [HAL_I2C_x](#) I2Cx, uint8_t *start, uint32_t len)
- void [HAL_DMA_SpiRxTx](#) ([HAL_DMA_x](#) DMA1_x, [HAL_DMA_x](#) DMA2_x, [HAL_SPI_x](#) SPI_x, uint8_t *txdata, uint8_t *rxdata, uint32_t len)

4.76.1 函数说明

4.76.1.1 HAL_DMA_Disable()

```
void HAL_DMA_Disable (
    HAL\_DMA\_x DMA_x )
```

在文件 [hal_lks32mc08x_dma.c](#) 第 21 行定义.

这是这个函数的调用关系图:



4.76.1.2 HAL_DMA_Enable()

```
void HAL_DMA_Enable (
    HAL_DMA_x DMA_x )
```

在文件 [hal_lks32mc08x_dma.c](#) 第 14 行定义.

4.76.1.3 HAL_DMA_I2cRx()

```
void HAL_DMA_I2cRx (
    HAL_DMA_x DMA_x,
    HAL_I2C_x I2Cx,
    uint8_t * start,
    uint32_t len )
```

在文件 [hal_lks32mc08x_dma.c](#) 第 115 行定义.

4.76.1.4 HAL_DMA_I2cTx()

```
void HAL_DMA_I2cTx (
    HAL_DMA_x DMA_x,
    HAL_I2C_x I2Cx,
    uint8_t * start,
    uint32_t len )
```

在文件 [hal_lks32mc08x_dma.c](#) 第 72 行定义.

4.76.1.5 HAL_DMA_MemCopy()

```
void HAL_DMA_MemCopy (
    HAL_DMA_x DMA_x,
    void * start,
    void * end,
    uint32_t len )
```

在文件 [hal_lks32mc08x_dma.c](#) 第 27 行定义.

4.76.1.6 HAL_DMA_Reset()

```
void HAL_DMA_Reset (
    HAL_DMA_x DMA_x )
```

在文件 [hal_lks32mc08x_dma.c](#) 第 9 行定义.

函数调用图:



4.76.1.7 HAL_DMA_SpiRx()

```
void HAL_DMA_SpiRx (
    HAL_DMA_x DMA_x,
    HAL_SPI_x SPI_x,
    uint8_t * start,
    uint32_t len )
```

在文件 [hal_lks32mc08x_dma.c](#) 第 105 行定义.

这是这个函数的调用关系图:

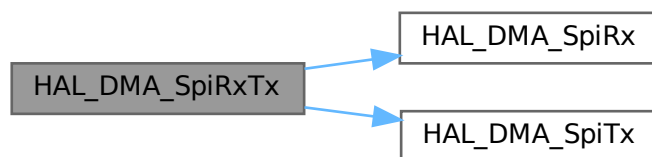


4.76.1.8 HAL_DMA_SpiRxTx()

```
void HAL_DMA_SpiRxTx (
    HAL_DMA_x DMA1_x,
    HAL_DMA_x DMA2_x,
    HAL_SPI_x SPI_x,
    uint8_t * txdata,
    uint8_t * rxdata,
    uint32_t len )
```

在文件 [hal_lks32mc08x_dma.c](#) 第 125 行定义.

函数调用图:



4.76.1.9 HAL_DMA_SpiTx()

```
void HAL_DMA_SpiTx (  
    HAL_DMA_x DMA_x,  
    HAL_SPI_x SPI_x,  
    uint8_t * start,  
    uint32_t len )
```

在文件 [hal_lks32mc08x.dma.c](#) 第 60 行定义.

这是这个函数的调用关系图:



4.76.1.10 HAL_DMA_UartRx()

```
void HAL_DMA_UartRx (  
    HAL_DMA_x DMA_x,  
    HAL_UART_x UART_x,  
    uint8_t * start,  
    uint32_t len )
```

在文件 [hal_lks32mc08x.dma.c](#) 第 84 行定义.

4.76.1.11 HAL_DMA_UartTx()

```
void HAL_DMA_UartTx (  
    HAL_DMA_x DMA_x,  
    HAL_UART_x UART_x,  
    uint8_t * start,  
    uint32_t len )
```

在文件 [hal_lks32mc08x.dma.c](#) 第 37 行定义.

4.77 hal_lks32mc08x_dma.c

[浏览该文件的文档.](#)

```

00001 #include "../include/hal_lksmcu_dma.h"
00002 #include "../hal_lks32mc08x.h"
00003 #include "lks32mc08x_user_manual.h"
00004 #include "../include/hal_lksmcu_reg.h"
00005 #include "../include/hal_lks32mc08x_reg.h"
00006 // 注意: 在使用I2C、UART或SPI通信时, 不支持指定DMA通道
00007 // 初始化
00008 // 模块复位
00009 void HAL_DMA_Reset (HAL_DMA_x DMA_x)
00010 {
00011     HAL_DMA_Disable (DMA_x);
00012 }
00013 // 模块打开
00014 void HAL_DMA_Enable (HAL_DMA_x DMA_x)
00015 {
00016     (void)DMA_x;
00017     HAL_REG_SET (DMA_CTRL, DMA_CTRL_EN_BIT);
00018     HAL_REG_RESET (DMA_CTRL, DMA_CTRL_PRIORITY_BIT);
00019 }
00020 // 模块关闭
00021 void HAL_DMA_Disable (HAL_DMA_x DMA_x)
00022 {
00023     HAL_REG_RESET (HAL_DMA [DMA_x] -> CCR, DMA_CCRx_EN_BIT);
00024     HAL_REG_RESET (DMA_CTRL, DMA_CTRL_EN_BIT);
00025 }
00026 // 内存复制 (08DMA不支持内存到内存, 这里用软件实现)
00027 void HAL_DMA_MemCopy (HAL_DMA_x DMA_x, void *start, void *end, uint32_t len)
00028 {
00029     uint32_t i;
00030     (void)DMA_x;
00031     for (i = 0; i < len; i++)
00032     {
00033         ((uint8_t *)end)[i] = ((uint8_t *)start)[i];
00034     }
00035 }
00036 // dma使用uart发送数据, 需要先初始化好对应的uart模块
00037 void HAL_DMA_UartTx (HAL_DMA_x DMA_x, HAL_UART_x UART_x, uint8_t *start, uint32_t len)
00038 {
00039     if (UART_x == HAL_UART_0)
00040     {
00041         DMA_x = HAL_DMA_2;
00042     }
00043     else if (UART_x == HAL_UART_1)
00044     {
00045         DMA_x = HAL_DMA_3;
00046     }
00047     else
00048     {
00049         return;
00050     }
00051     HAL_REG_WRITE (HAL_DMA [DMA_x] -> CMAR, (uint32_t)(uintptr_t)start);
00052     HAL_REG_WRITE (HAL_DMA [DMA_x] -> CPAR, (uint32_t)(uintptr_t)(&(HAL_UART [UART_x] -> BUFF)));
00053     HAL_REG_WRITE_BIT_MASK_POS (HAL_DMA [DMA_x] -> CTMS, DMA_CTMSx_ROUND, len);
00054     HAL_REG_WRITE_BIT_MASK_POS (HAL_DMA [DMA_x] -> CTMS, DMA_CTMSx_TIMES, 1);
00055     HAL_REG_WRITE (HAL_DMA [DMA_x] -> CCR, DMA_CCRx_REQ_EN (BIT0) | DMA_CCRx_MBTW (0) | DMA_CCRx_PBTW (0) |
DMA_CCRx_MINC_BIT | DMA_CCRx_DIR_BIT | DMA_CCRx_EN_BIT);
00056     HAL_REG_SET (HAL_DMA [DMA_x] -> CCR, DMA_CCRx_SW_TRIG_BIT);
00057     HAL_REG_RESET (HAL_DMA [DMA_x] -> CCR, DMA_CCRx_SW_TRIG_BIT);
00058 }
00059 // dma使用spi发送数据, 需要先初始化好对应的spi模块
00060 void HAL_DMA_SpiTx (HAL_DMA_x DMA_x, HAL_SPI_x SPI_x, uint8_t *start, uint32_t len)
00061 {
00062     (void)DMA_x;
00063     HAL_REG_WRITE (HAL_DMA [2] -> CMAR, (uint32_t)(uintptr_t)start);
00064     HAL_REG_WRITE (HAL_DMA [2] -> CPAR, (uint32_t)(uintptr_t)(&(HAL_SPI [SPI_x] -> TXDATA)));
00065     HAL_REG_WRITE_BIT_MASK_POS (HAL_DMA [2] -> CTMS, DMA_CTMSx_ROUND, len);
00066     HAL_REG_WRITE_BIT_MASK_POS (HAL_DMA [2] -> CTMS, DMA_CTMSx_TIMES, 1);
00067     HAL_REG_WRITE (HAL_DMA [2] -> CCR, DMA_CCRx_REQ_EN (BIT1) | DMA_CCRx_MBTW (0) | DMA_CCRx_PBTW (0) |
DMA_CCRx_MINC_BIT | DMA_CCRx_DIR_BIT | DMA_CCRx_EN_BIT);
00068     HAL_REG_SET (HAL_DMA [2] -> CCR, DMA_CCRx_SW_TRIG_BIT);
00069     HAL_REG_RESET (HAL_DMA [2] -> CCR, DMA_CCRx_SW_TRIG_BIT);
00070 }
00071 // dma使用i2c发送数据, 需要先初始化好对应的i2c模块
00072 void HAL_DMA_I2cTx (HAL_DMA_x DMA_x, HAL_I2C_x I2Cx, uint8_t *start, uint32_t len)
00073 {
00074     (void)DMA_x;
00075     HAL_REG_WRITE (HAL_DMA [3] -> CMAR, (uint32_t)(uintptr_t)start);
00076     HAL_REG_WRITE (HAL_DMA [3] -> CPAR, (uint32_t)(uintptr_t)(&(HAL_I2C [I2Cx] -> DATA)));
00077     HAL_REG_WRITE_BIT_MASK_POS (HAL_DMA [3] -> CTMS, DMA_CTMSx_ROUND, len);
00078     HAL_REG_WRITE_BIT_MASK_POS (HAL_DMA [3] -> CTMS, DMA_CTMSx_TIMES, 1);
00079     HAL_REG_WRITE (HAL_DMA [3] -> CCR, DMA_CCRx_REQ_EN (BIT2) | DMA_CCRx_MBTW (0) | DMA_CCRx_PBTW (0) |
DMA_CCRx_MINC_BIT | DMA_CCRx_DIR_BIT | DMA_CCRx_EN_BIT);

```

```

00080     HAL_REG_SET(HAL_DMA[3]->CCR, DMA_CCRx.SW_TRIG_BIT);
00081     HAL_REG_RESET(HAL_DMA[3]->CCR, DMA_CCRx.SW_TRIG_BIT);
00082 }
00083 // dma使用uart接收数据,需要先初始化好对应的uart模块
00084 void HAL_DMA_UartRx(HAL_DMA_x DMA_x, HAL_UART_x UART_x, uint8_t *start, uint32_t len)
00085 {
00086     if (UART_x == HAL_UART_0)
00087     {
00088         DMA_x = HAL_DMA_2;
00089     }
00090     else if (UART_x == HAL_UART_1)
00091     {
00092         DMA_x = HAL_DMA_3;
00093     }
00094     else
00095     {
00096         return;
00097     }
00098     HAL_REG_WRITE(HAL_DMA[DMA_x]->CMAR, (uint32_t)(uintptr_t)start);
00099     HAL_REG_WRITE(HAL_DMA[DMA_x]->CPAR, (uint32_t)(uintptr_t)(HAL_UART[UART_x]->BUFF));
00100     HAL_REG_WRITE_BIT_MASK_POS(HAL_DMA[DMA_x]->CTMS, DMA_CTMSx_ROUND, len);
00101     HAL_REG_WRITE_BIT_MASK_POS(HAL_DMA[DMA_x]->CTMS, DMA_CTMSx_TIMES, 1);
00102     HAL_REG_WRITE(HAL_DMA[DMA_x]->CCR, DMA_CCRx_REQ_EN(BIT0) | DMA_CCRx_MBTW(0) | DMA_CCRx_PBTW(0) |
DMA_CCRx_MINC_BIT | DMA_CCRx_EN_BIT);
00103 }
00104 // dma使用spi接收数据,需要先初始化好对应的spi模块
00105 void HAL_DMA_SpiRx(HAL_DMA_x DMA_x, HAL_SPI_x SPI_x, uint8_t *start, uint32_t len)
00106 {
00107     (void)DMA_x;
00108     HAL_REG_WRITE(HAL_DMA[1]->CMAR, (uint32_t)(uintptr_t)start);
00109     HAL_REG_WRITE(HAL_DMA[1]->CPAR, (uint32_t)(uintptr_t)(HAL_SPI[SPI_x]->RXDATA));
00110     HAL_REG_WRITE_BIT_MASK_POS(HAL_DMA[1]->CTMS, DMA_CTMSx_ROUND, len);
00111     HAL_REG_WRITE_BIT_MASK_POS(HAL_DMA[1]->CTMS, DMA_CTMSx_TIMES, 1);
00112     HAL_REG_WRITE(HAL_DMA[1]->CCR, DMA_CCRx_REQ_EN(BIT0) | DMA_CCRx_MBTW(0) | DMA_CCRx_PBTW(0) |
DMA_CCRx_MINC_BIT | DMA_CCRx_EN_BIT);
00113 }
00114 // dma使用i2c接收数据,需要先初始化好对应的i2c模块
00115 void HAL_DMA_I2cRx(HAL_DMA_x DMA_x, HAL_I2C_x I2C_x, uint8_t *start, uint32_t len)
00116 {
00117     (void)DMA_x;
00118     HAL_REG_WRITE(HAL_DMA[3]->CMAR, (uint32_t)(uintptr_t)start);
00119     HAL_REG_WRITE(HAL_DMA[3]->CPAR, (uint32_t)(uintptr_t)(HAL_I2C[I2C_x]->DATA));
00120     HAL_REG_WRITE_BIT_MASK_POS(HAL_DMA[3]->CTMS, DMA_CTMSx_ROUND, len);
00121     HAL_REG_WRITE_BIT_MASK_POS(HAL_DMA[3]->CTMS, DMA_CTMSx_TIMES, 1);
00122     HAL_REG_WRITE(HAL_DMA[3]->CCR, DMA_CCRx_REQ_EN(BIT2) | DMA_CCRx_MBTW(0) | DMA_CCRx_PBTW(0) |
DMA_CCRx_MINC_BIT | DMA_CCRx_EN_BIT);
00123 }
00124 // 使用两个dma模块和一个spi模块,同时发送&接收数据,需要先初始化好对应的spi模块
00125 void HAL_DMA_SpiRxTx(HAL_DMA_x DMA1_x, HAL_DMA_x DMA2_x, HAL_SPI_x SPI_x, uint8_t *txdata, uint8_t *rxdata,
uint32_t len)
00126 {
00127     HAL_DMA_SpiRx(DMA1_x, SPI_x, rxdata, len);
00128     HAL_DMA_SpiTx(DMA2_x, SPI_x, txdata, len);
00129 }

```

4.78 source/hal_lks32mc08x_dsp.c 文件参考

```

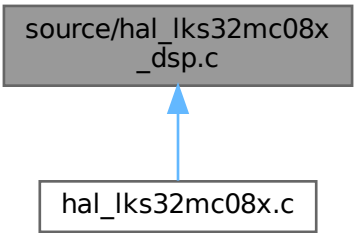
#include "../include/hal_lksmcu_dsp.h"
#include "../hal_lks32mc08x.h"
#include "lks32mc08x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc08x_reg.h"

```

hal_lks32mc08x_dsp.c 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



函数

- void [HAL_DSP_Reset](#) ([HAL_DSP_x](#) dsp_x)
DSP复位
- void [HAL_DSP_Enable](#) ([HAL_DSP_x](#) dsp_x)
DSP使能
- void [HAL_DSP_Disable](#) ([HAL_DSP_x](#) dsp_x)
DSP复位
- int16_t [HAL_DSP_Sin](#) ([HAL_DSP_x](#) dsp_x, int16_t angle)
DSP正弦计算
- int16_t [HAL_DSP_Cos](#) ([HAL_DSP_x](#) dsp_x, int16_t angle)
DSP余弦计算
- void [HAL_DSP_SinCos](#) ([HAL_DSP_x](#) dsp_x, int16_t angle, int16_t *sin, int16_t *cos)
DSP正弦余弦计算
- uint16_t [HAL_DSP_Sqrt](#) ([HAL_DSP_x](#) dsp_x, uint32_t x)
DSP开方
- int32_t [HAL_DSP_Div](#) ([HAL_DSP_x](#) dsp_x, int32_t a, int32_t b)
DSP除法

4.78.1 函数说明

4.78.1.1 HAL_DSP_Cos()

```
int16_t HAL_DSP_Cos (  
    HAL\_DSP\_x dsp_x,  
    int16_t angle )
```

DSP余弦计算

参数

<i>dsp_x</i>	DSP选择
<i>angle</i>	角度

返回值

余弦结果	

在文件 [hal_lks32mc08x.dsp.c](#) 第 70 行定义.

4.78.1.2 HAL_DSP_Disable()

```
void HAL_DSP_Disable (
    HAL_DSP_x dsp_x )
```

DSP复位

参数

dsp_x	DSP选择

在文件 [hal_lks32mc08x.dsp.c](#) 第 40 行定义.

4.78.1.3 HAL_DSP_Div()

```
int32_t HAL_DSP_Div (
    HAL_DSP_x dsp_x,
    int32_t a,
    int32_t b )
```

DSP除法

参数

dsp_x	DSP选择
a	被除数
b	除数

返回值

$a \div b$	

在文件 [hal_lks32mc08x.dsp.c](#) 第 113 行定义.

4.78.1.4 HAL_DSP_Enable()

```
void HAL_DSP_Enable (
    HAL_DSP_x dsp_x )
```

DSP使能

参数

<i>dsp_x</i>	DSP选择

在文件 [hal_lks32mc08x.dsp.c](#) 第 26 行定义.

4.78.1.5 HAL_DSP_Reset()

```
void HAL_DSP_Reset (
    HAL_DSP_x dsp_x )
```

DSP复位

参数

<i>dsp_x</i>	DSP选择

在文件 [hal_lks32mc08x.dsp.c](#) 第 11 行定义.

4.78.1.6 HAL_DSP_Sin()

```
int16_t HAL_DSP_Sin (
    HAL_DSP_x dsp_x,
    int16_t angle )
```

DSP正弦计算

参数

<i>dsp_x</i>	DSP选择
<i>angle</i>	角度

返回值

正弦结果	

在文件 [hal_lks32mc08x.dsp.c](#) 第 56 行定义.

4.78.1.7 HAL_DSP_SinCos()

```
void HAL_DSP_SinCos (
    HAL_DSP_x dsp_x,
    int16_t angle,
```

```
int16_t * sin,  
int16_t * cos )
```

DSP正弦余弦计算

参数

<i>dsp_x</i>	DSP选择
<i>angle</i>	角度
<i>sin</i>	正弦结果
<i>cos</i>	余弦结果

在文件 [hal_lks32mc08x_dsp.c](#) 第 85 行定义.

4.78.1.8 HAL_DSP_Sqrt()

```
uint16_t HAL_DSP_Sqrt (  
    HAL_DSP_x dsp_x,  
    uint32_t x )
```

DSP开方

参数

<i>dsp_x</i>	DSP选择
<i>x</i>	输入

返回值

开方结果	

在文件 [hal_lks32mc08x_dsp.c](#) 第 99 行定义.

4.79 hal_lks32mc08x_dsp.c

[浏览该文件的文档.](#)

```
00001 #include "../include/hal_lksmcu_dsp.h"  
00002 #include "../hal_lks32mc08x.h"  
00003 #include "lks32mc08x.user_manual.h"  
00004 #include "../include/hal_lksmcu_reg.h"  
00005 #include "../include/hal_lks32mc08x_reg.h"  
00006  
00011 void HAL_DSP_Reset (HAL_DSP_x dsp_x)  
00012 {  
00013     (void) dsp_x;  
00014     // 系统寄存器解锁  
00015     HAL_REG_SYS_UNLOCK();  
00016     HAL_REG_SET(SYS_SFT_RST, SYS_SFT_RST_DSP_SFT_RST_BIT);  
00017     HAL_REG_RESET(SYS_SFT_RST, SYS_SFT_RST_DSP_SFT_RST_BIT);  
00018     // 系统寄存器上锁  
00019     HAL_REG_SYS_LOCK();
```

```

00020 }
00021
00026 void HAL_DSP_Enable(HAL_DSP_x dsp_x)
00027 {
00028     (void)dsp_x;
00029     // 系统寄存器解锁
00030     HAL_REG_SYS_UNLOCK();
00031     HAL_REG_SET(SYS_CLK_FEN, SYS_CLK_FEN_DSP_CLK_EN_BIT);
00032     // 系统寄存器上锁
00033     HAL_REG_SYS_LOCK();
00034 }
00035
00040 void HAL_DSP_Disable(HAL_DSP_x dsp_x)
00041 {
00042     (void)dsp_x;
00043     // 系统寄存器解锁
00044     HAL_REG_SYS_UNLOCK();
00045     HAL_REG_RESET(SYS_CLK_FEN, SYS_CLK_FEN_DSP_CLK_EN_BIT);
00046     // 系统寄存器上锁
00047     HAL_REG_SYS_LOCK();
00048 }
00049
00056 int16_t HAL_DSP_Sin(HAL_DSP_x dsp_x, int16_t angle)
00057 {
00058     (void)dsp_x;
00059     HAL_REG_WRITE(DSP_SC, DSP_SC_PAUSED_BIT | DSP_SC_CORDIC_MODE_BIT);
00060     HAL_REG_WRITE(DSP_THETA, angle);
00061     return HAL_REG_READ(DSP_SIN);
00062 }
00063
00070 int16_t HAL_DSP_Cos(HAL_DSP_x dsp_x, int16_t angle)
00071 {
00072     (void)dsp_x;
00073     HAL_REG_WRITE(DSP_SC, DSP_SC_PAUSED_BIT | DSP_SC_CORDIC_MODE_BIT);
00074     HAL_REG_WRITE(DSP_THETA, angle);
00075     return HAL_REG_READ(DSP_COS);
00076 }
00077
00085 void HAL_DSP_SinCos(HAL_DSP_x dsp_x, int16_t angle, int16_t *sin, int16_t *cos)
00086 {
00087     (void)dsp_x;
00088     HAL_REG_WRITE(DSP_SC, DSP_SC_PAUSED_BIT | DSP_SC_CORDIC_MODE_BIT);
00089     HAL_REG_WRITE(DSP_THETA, angle);
00090     *sin = HAL_REG_READ(DSP_SIN);
00091     *cos = HAL_REG_READ(DSP_COS);
00092 }
00099 uint16_t HAL_DSP_Sqrt(HAL_DSP_x dsp_x, uint32_t x)
00100 {
00101     (void)dsp_x;
00102     HAL_REG_WRITE(DSP_RAD, x);
00103     return HAL_REG_READ(DSP_SQRT);
00104 }
00105
00113 int32_t HAL_DSP_Div(HAL_DSP_x dsp_x, int32_t a, int32_t b)
00114 {
00115     (void)dsp_x;
00116     if (b == 0)
00117     {
00118         return 0;
00119     }
00120     else if (b == 1)
00121     {
00122         return a;
00123     }
00124     else if ((b > 32767) || (b < -32767))
00125     {
00126         return a / b;
00127     }
00128     else
00129     {
00130         HAL_REG_WRITE(DSP_DID, a);
00131         HAL_REG_WRITE(DSP_DIS, b);
00132         return HAL_REG_READ(DSP_QUO);
00133     }
00134 }

```

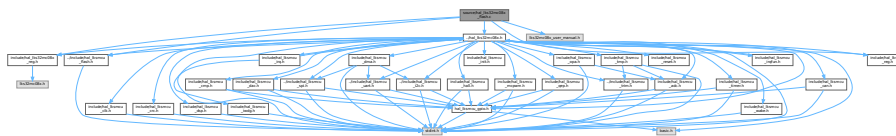
4.80 source/hal_lks32mc08x_flash.c 文件参考

```

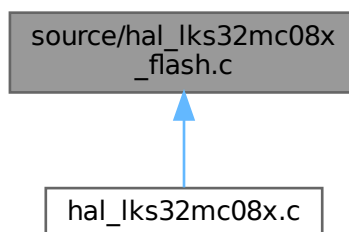
#include "../include/hal_lksmcu_flash.h"
#include "../hal_lks32mc08x.h"

```

```
#include "lks32mc08x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc08x_reg.h"
hal_lks32mc08x_flash.c 的引用(Include)关系图:
```



此图展示该文件直接或间接的被哪些文件引用了:



函数

- void **HAL_FLASH_Reset** (HAL_FLASH_x flash)
模块复位,预留接口
- void **HAL_FLASH_Enable** (HAL_FLASH_x flash)
模块打开,预留接口
- void **HAL_FLASH_Disable** (HAL_FLASH_x flash)
模块关闭,预留接口
- void **HAL_FLASH_EnableCache** (HAL_FLASH_x flash)
使能flash缓存加速
- void **HAL_FLASH_DisableCache** (HAL_FLASH_x flash)
关闭flash缓存加速
- uint32_t **HAL_FLASH_ReadSectorSize** (HAL_FLASH_x flash)
获取一个扇区的大小,和HAL_FLASH_SECTOR_SIZE等效
- void **HAL_FLASH_Read** (HAL_FLASH_x flash, uint32_t flashaddr, void *data, uint32_t len)
读FLASH取数据
- uint8_t **HAL_FLASH_ReadUint8** (HAL_FLASH_x flash, uint32_t flashaddr)
读FLASH取数据
- uint16_t **HAL_FLASH_ReadUint16** (HAL_FLASH_x flash, uint32_t flashaddr)
读FLASH取数据
- uint32_t **HAL_FLASH_ReadUint32** (HAL_FLASH_x flash, uint32_t flashaddr)
读FLASH取数据
- void **HAL_FLASH_Erase** (HAL_FLASH_x flash, uint32_t flashaddr, uint32_t len)

扇区擦除,*len*只能是扇区大小的整数倍,不够一个扇区的部分会被忽略

- void **HAL_FLASH_Program** (**HAL_FLASH_x** flash, uint32_t flashaddr, uint8_t *data, uint32_t len)
编程
- void **HAL_FLASH_Update** (**HAL_FLASH_x** flash, uint32_t flashaddr, uint8_t *data, uint32_t len)
更新数据,等效先擦除再编程
- static uint32_t **HAL_FLASH_SecurityParameterCrc** (**HAL_FLASH_x** flash, uint32_t addr)
- void **HAL_FLASH_SecurityParameterSave** (**HAL_FLASH_x** flash, void *defaultdata, void *data, uint32_t len)
保存安全参数
- uint32_t **HAL_FLASH_SecurityParameterRead** (**HAL_FLASH_x** flash, void *defaultdata, void *data, uint32_t len)
读取安全参数

变量

- static volatile const uint8_t **HAL_FLASH_data** [3 * 512] = {0}

4.80.1 函数说明

4.80.1.1 HAL_FLASH_Disable()

```
void HAL_FLASH_Disable (
    HAL_FLASH_x flash )
```

模块关闭,预留接口

参数

<i>flash</i>	模块类型

在文件 **hal_lks32mc08x_flash.c** 第 30 行定义.

4.80.1.2 HAL_FLASH_DisableCache()

```
void HAL_FLASH_DisableCache (
    HAL_FLASH_x flash )
```

关闭flash缓存加速

参数

<i>flash</i>	模块类型

在文件 **hal_lks32mc08x_flash.c** 第 49 行定义.

4.80.1.3 HAL_FLASH_Enable()

```
void HAL_FLASH_Enable (
    HAL_FLASH_x flash )
```

模块打开,预留接口

参数

<i>flash</i>	模块类型

在文件 [hal_lks32mc08x_flash.c](#) 第 21 行定义.

4.80.1.4 HAL_FLASH_EnableCache()

```
void HAL_FLASH_EnableCache (
    HAL_FLASH_x flash )
```

使能flash缓存加速

参数

<i>flash</i>	模块类型

在文件 [hal_lks32mc08x_flash.c](#) 第 39 行定义.

这是这个函数的调用关系图:



4.80.1.5 HAL_FLASH_Erase()

```
void HAL_FLASH_Erase (
    HAL_FLASH_x flash,
    uint32_t flashaddr,
    uint32_t len )
```

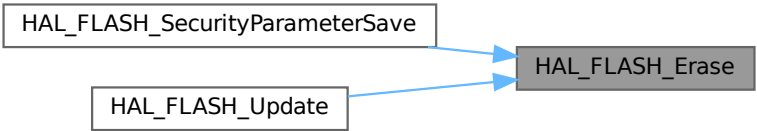
扇区擦除,len只能是扇区大小的整数倍,不够一个扇区的部分会被忽略

参数

<i>flash</i>	模块类型
<i>flashaddr</i>	flash地址
<i>len</i>	擦除长度

在文件 [hal_lks32mc08x.flash.c](#) 第 124 行定义.

这是这个函数的调用关系图:



4.80.1.6 HAL_FLASH_Program()

```
void HAL_FLASH_Program (
    HAL_FLASH_x flash,
    uint32_t flashaddr,
    uint8_t * data,
    uint32_t len )
```

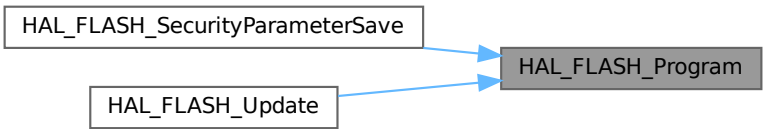
编程

参数

<i>flash</i>	模块类型
<i>flashaddr</i>	flash地址,低两位会被忽略
<i>data</i>	数据指针
<i>len</i>	数据长度

在文件 [hal_lks32mc08x.flash.c](#) 第 146 行定义.

这是这个函数的调用关系图:



4.80.1.7 HAL_FLASH_Read()

```
void HAL_FLASH_Read (
    HAL_FLASH_x flash,
```

```
uint32_t flashaddr,
void * data,
uint32_t len )
```

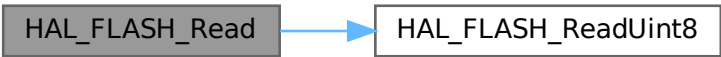
读FLASH取数据

参数

<i>flash</i>	模块类型
<i>flashaddr</i>	flash地址
<i>data</i>	读取的数据指针
<i>len</i>	读取长度

在文件 [hal_lks32mc08x.flash.c](#) 第 73 行定义.

函数调用图:



这是这个函数的调用关系图:



4.80.1.8 HAL_FLASH_ReadSectorSize()

```
uint32_t HAL_FLASH_ReadSectorSize (
    HAL_FLASH_x flash )
```

获取一个扇区的大小,和HAL_FLASH_SECTOR_SIZE等效

参数

<i>flash</i>	模块类型

返回

uint32_t 扇区大小

在文件 [hal_lks32mc08x.flash.c](#) 第 60 行定义.

4.80.1.9 HAL_FLASH_ReadUint16()

```
uint16_t HAL_FLASH_ReadUint16 (
    HAL_FLASH_x flash,
    uint32_t flashaddr )
```

读FLASH取数据

参数

<i>flash</i>	模块类型
<i>flashaddr</i>	flash地址

返回
读取的数据

在文件 [hal_lks32mc08x.flash.c](#) 第 100 行定义.

4.80.1.10 HAL_FLASH_ReadUint32()

```
uint32_t HAL_FLASH_ReadUint32 (
    HAL_FLASH_x flash,
    uint32_t flashaddr )
```

读FLASH取数据

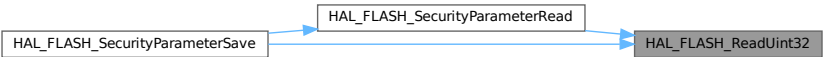
参数

<i>flash</i>	模块类型
<i>flashaddr</i>	flash地址

返回
读取的数据

在文件 [hal_lks32mc08x.flash.c](#) 第 112 行定义.

这是这个函数的调用关系图:



4.80.1.11 HAL_FLASH_ReadUint8()

```
uint8_t HAL_FLASH_ReadUint8 (
    HAL_FLASH_x flash,
    uint32_t flashaddr )
```

读FLASH取数据

参数

<i>flash</i>	模块类型
<i>flashaddr</i>	flash地址

返回

读取的数据

在文件 `hal_ks32mc08x_flash.c` 第 88 行定义.

这是这个函数的调用关系图:



4.80.1.12 HAL_FLASH_Reset()

```
void HAL_FLASH_Reset (
    HAL_FLASH_x flash )
```

模块复位,预留接口

参数

<i>flash</i>	模块类型

在文件 `hal_ks32mc08x_flash.c` 第 12 行定义.

4.80.1.13 HAL_FLASH_SecurityParameterCrc()

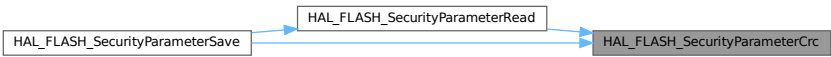
```
static uint32_t HAL_FLASH_SecurityParameterCrc (
    HAL_FLASH_x flash,
    uint32_t addr ) [static]
```

在文件 `hal_iks32mc08x_flash.c` 第 179 行定义.

函数调用图:



这是这个函数的调用关系图:



4.80.1.14 HAL_FLASH_SecurityParameterRead()

```
uint32_t HAL_FLASH_SecurityParameterRead (
    HAL_FLASH_x flash,
    void * defaultdata,
    void * data,
    uint32_t len )
```

读取安全参数

参数

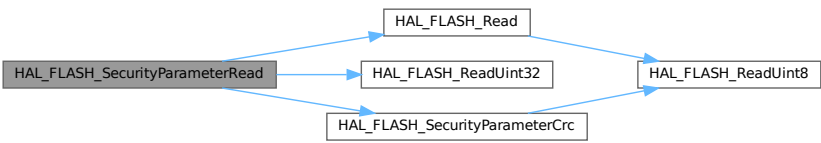
<i>flash</i>	模块类型
<i>data</i>	数据指针
<i>len</i>	数据长度

返回

使用的flash地址

在文件 [hal_lks32mc08x.flash.c](#) 第 310 行定义.

函数调用图:



这是这个函数的调用关系图:



4.80.1.15 HAL_FLASH_SecurityParameterSave()

```
void HAL_FLASH_SecurityParameterSave (
    HAL_FLASH_x flash,
    void * defaultdata,
    void * data,
    uint32_t len )
```

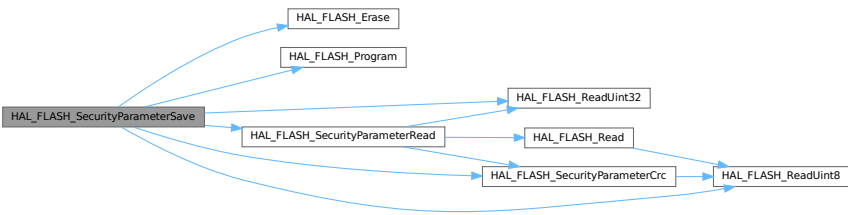
保存安全参数

参数

<i>flash</i>	模块类型
<i>defaultdata</i>	默认值
<i>data</i>	待写入的数据
<i>len</i>	数据长度

在文件 [hal_lks32mc08x.flash.c](#) 第 216 行定义.

函数调用图:



4.80.1.16 HAL_FLASH_Update()

```
void HAL_FLASH_Update (
    HAL_FLASH_x flash,
    uint32_t flashaddr,
    uint8_t * data,
    uint32_t len )
```

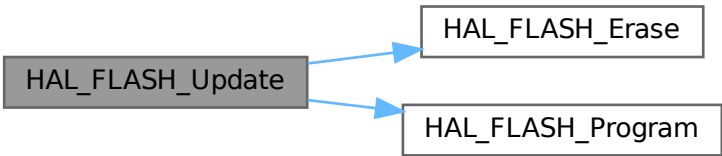
更新数据,等效先擦除再编程

参数

<i>flash</i>	模块类型
<i>flashaddr</i>	flash地址
<i>data</i>	数据指针
<i>len</i>	数据长度

在文件 [hal_lks32mc08x.flash.c](#) 第 172 行定义.

函数调用图:



4.80.2 变量说明

4.80.2.1 HAL_FLASH_data

```
volatile const uint8_t HAL_FLASH_data[3 * 512] = {0} [static]
```

在文件 [hal_lks32mc08x.flash.c](#) 第 7 行定义.

4.81 hal_lks32mc08x_flash.c

[浏览该文件的文档.](#)

```
00001 #include "../include/hal_lksmcu.flash.h"
00002 #include "../hal_lks32mc08x.h"
00003 #include "lks32mc08x.user_manual.h"
00004 #include "../include/hal_lksmcu.reg.h"
00005 #include "../include/hal_lks32mc08x.reg.h"
00006 // 使用三个扇区（确保在随机分配地址时，能分配到至少两个扇区）
00007 static volatile const uint8_t HAL_FLASH_data[3 * 512] = {0};
00012 void HAL_FLASH_Reset(HAL_FLASH_x flash)
00013 {
00014     (void) flash;
00015 }
00016
00021 void HAL_FLASH_Enable(HAL_FLASH_x flash)
00022 {
00023     (void) flash;
00024 }
00025
00030 void HAL_FLASH_Disable(HAL_FLASH_x flash)
00031 {
00032     (void) flash;
00033 }
00034
00039 void HAL_FLASH_EnableCache(HAL_FLASH_x flash)
00040 {
```

```

00041     (void)flash;
00042     HAL_REG_SET(FLASH_CFG, FLASH_CFG_PREF_BIT);
00043 }
00044
00049 void HAL_FLASH_DisableCache(HAL_FLASH_x flash)
00050 {
00051     (void)flash;
00052     HAL_REG_RESET(FLASH_CFG, FLASH_CFG_PREF_BIT);
00053 }
00054
00060 uint32_t HAL_FLASH_ReadSectorSize(HAL_FLASH_x flash)
00061 {
00062     (void)flash;
00063     return 512;
00064 }
00065
00073 void HAL_FLASH_Read(HAL_FLASH_x flash, uint32_t flashaddr, void *data, uint32_t len)
00074 {
00075     uint32_t i;
00076     for (i = 0; i < len; i++)
00077     {
00078         ((uint8_t *)data)[i] = HAL_FLASH_ReadUint8(flash, flashaddr + i);
00079     }
00080 }
00081
00088 uint8_t HAL_FLASH_ReadUint8(HAL_FLASH_x flash, uint32_t flashaddr)
00089 {
00090     (void)flash;
00091     return REG8(flashaddr);
00092 }
00093
00100 uint16_t HAL_FLASH_ReadUint16(HAL_FLASH_x flash, uint32_t flashaddr)
00101 {
00102     (void)flash;
00103     return REG16(flashaddr);
00104 }
00105
00112 uint32_t HAL_FLASH_ReadUint32(HAL_FLASH_x flash, uint32_t flashaddr)
00113 {
00114     (void)flash;
00115     return REG32(flashaddr);
00116 }
00117
00124 void HAL_FLASH_Erase(HAL_FLASH_x flash, uint32_t flashaddr, uint32_t len)
00125 {
00126     (void)flash;
00127     HAL_REG_RESET(FLASH_CFG, FLASH_CFG_ERS_TYPE_BIT);
00128     HAL_REG_RESET(FLASH_CFG, FLASH_CFG_REGION_BIT);
00129     HAL_REG_SET(FLASH_CFG, FLASH_CFG_ERS_EN_BIT);
00130
00131     for (; ((int32_t)len) > 0; len -= 512, flashaddr += 512)
00132     {
00133         HAL_REG_WRITE(FLASH_ADDR, flashaddr);
00134         HAL_REG_WRITE(FLASH_ERASE, 0x7654dcba);
00135     }
00136     HAL_REG_RESET(FLASH_CFG, FLASH_CFG_ERS_EN_BIT);
00137 }
00138
00146 void HAL_FLASH_Program(HAL_FLASH_x flash, uint32_t flashaddr, uint8_t *data, uint32_t len)
00147 {
00148     uint8_t *p = data;
00149     (void)flash;
00150     HAL_REG_RESET(FLASH_CFG, FLASH_CFG_REGION_BIT); // 访问main区域
00151     HAL_REG_SET(FLASH_CFG, FLASH_CFG_PRG_EN_BIT); // 打开编程使能
00152     HAL_REG_SET(FLASH_CFG, FLASH_CFG_ADR_INC_BIT); // 打开地址递增使能
00153     flashaddr &= ~0x3;
00154     HAL_REG_WRITE(FLASH_ADDR, flashaddr); // 写入初始地址
00155     for (; len > 0; len -= 4)
00156     {
00157         uint32_t data = p[0] | (p[1] << 8) | (p[2] << 16) | (p[3] << 24);
00158         HAL_REG_WRITE(FLASH_WDATA, data);
00159         p += 4;
00160     }
00161     HAL_REG_RESET(FLASH_CFG, FLASH_CFG_PRG_EN_BIT); // 关闭编程使能
00162     HAL_REG_RESET(FLASH_CFG, FLASH_CFG_ADR_INC_BIT); // 关闭地址递增使能
00163 }
00164
00172 void HAL_FLASH_Update(HAL_FLASH_x flash, uint32_t flashaddr, uint8_t *data, uint32_t len)
00173 {
00174     HAL_FLASH_Erase(flash, flashaddr, len);
00175     HAL_FLASH_Program(flash, flashaddr, data, len);
00176 }
00177
00178 // 仅内部使用, 计算CRC32
00179 static uint32_t HAL_FLASH_SecurityParameterCrc(HAL_FLASH_x flash, uint32_t addr)
00180 {
00181     uint32_t crc
        = 0xFFFFFFFF; // CRC32初始值

```

```

00182     const uint32_t polynomial = 0xEDB88320; // CRC32多项式
00183
00184     uint32_t i;
00185     for (i = 0; i < 504; i++)
00186     {
00187         uint32_t j;
00188         uint8_t byte = HAL_FLASH_ReadUint8(flash, addr + i);
00189
00190         // 将字节与CRC的高8位异或
00191         crc ^= byte;
00192         // 处理每个bit
00193         for (j = 0; j < 8; j++)
00194         {
00195             if (crc & 1)
00196             {
00197                 crc = (crc >> 1) ^ polynomial;
00198             }
00199             else
00200             {
00201                 crc >>= 1;
00202             }
00203         }
00204     }
00205
00206     return ~crc; // 取反得到最终结果
00207 }
00208
00216 void HAL_FLASH_SecurityParameterSave(HAL_FLASH_x flash, void *defaultdata, void *data, uint32_t len)
00217 {
00218     uint32_t addr0 = ((uint32_t)(uintptr_t)HAL_FLASH_data + 512) & ~0x1fff;
00219     uint32_t addr1 = addr0 + 512;
00220     uint32_t flg = 0;
00221     uint32_t addr = HAL_FLASH_SecurityParameterRead(flash, 0, 0, 0);
00222     if (addr != 0xffffffff)
00223     {
00224         uint32_t i;
00225         for (i = 0; i < len; i++)
00226         {
00227             // 判断待写入的数据和flash保存的数据是否一致
00228             if (((uint8_t *)data)[i] != HAL_FLASH_ReadUint8(flash, addr + i))
00229             {
00230                 flg = 1;
00231                 break;
00232             }
00233         }
00234     }
00235     else
00236     {
00237         uint32_t i;
00238         for (i = 0; i < len; i++)
00239         {
00240             // 判断待写入的数据和默认数据是否一致
00241             if (((uint8_t *)data)[i] != ((uint8_t *)defaultdata)[i])
00242             {
00243                 flg = 1;
00244                 break;
00245             }
00246         }
00247     }
00248     if (flg != 0) // 需要保存数据
00249     {
00250         uint32_t cnt;
00251         uint32_t cnt1;
00252         uint32_t crc;
00253         if (addr == addr0)
00254         {
00255             cnt = HAL_FLASH_ReadUint32(flash, addr0 + 500);
00256             cnt1 = cnt + 1;
00257             if (cnt1 == 0) // 溢出后从1开始计数
00258             {
00259                 cnt1 = 1;
00260             }
00261             HAL_FLASH_Erase(flash, addr1, 512); // 擦除另一个扇区
00262             HAL_FLASH_Program(flash, addr1, data, len); // 写入用户数据
00263             HAL_FLASH_Program(flash, addr1 + 500, (void *)&cnt1, 4); // 写入计数
00264             crc = HAL_FLASH_SecurityParameterCrc(flash, addr1); // 计算CRC结果
00265             HAL_FLASH_Program(flash, addr1 + 504, (void *)&crc, 4); // 写入crc
00266
00267             if (cnt == 0xffffffff) // 溢出后清空另一个扇区 (确保HAL_FLASH_SecurityParameterRead判断正确)
00268             {
00269                 HAL_FLASH_Erase(flash, addr0, 512);
00270             }
00271         }
00272         else if (addr == addr1)
00273         {
00274             cnt = HAL_FLASH_ReadUint32(flash, addr0 + 500);
00275             cnt1 = cnt + 1;

```

```

00276         if (cnt1 == 0) // 溢出后从1开始计数
00277         {
00278             cnt1 = 1;
00279         }
00280         HAL_FLASH_Erase(flash, addr0, 512);
00281         HAL_FLASH_Program(flash, addr0, data, len);
00282         HAL_FLASH_Program(flash, addr0 + 500, (void *)&cnt1, 4);
00283         crc = HAL_FLASH_SecurityParameterCrc(flash, addr0);
00284         HAL_FLASH_Program(flash, addr0 + 504, (void *)&crc, 4);
00285
00286         if (cnt == 0xffffffff) // 溢出后清空另一个扇区 (确保HAL_FLASH_SecurityParameterRead判断正确)
00287         {
00288             HAL_FLASH_Erase(flash, addr0, 512);
00289         }
00290     }
00291     else
00292     {
00293         cnt = 1; // 第一次写入时, 写入数据到扇区0, 并
从1开始计数
00294         HAL_FLASH_Erase(flash, addr1, 512); // 擦除另一个扇区
00295         HAL_FLASH_Program(flash, addr1, data, len); // 写入用户数据
00296         HAL_FLASH_Program(flash, addr1 + 500, (void *)&cnt, 4); // 写入计数
00297         crc = HAL_FLASH_SecurityParameterCrc(flash, addr1); // 计算CRC结果
00298         HAL_FLASH_Program(flash, addr1 + 504, (void *)&crc, 4); // 写入crc
00299     }
00300 }
00301 }
00302
00310 uint32_t HAL_FLASH_SecurityParameterRead(HAL_FLASH_t flash, void *defaultdata, void *data, uint32_t len)
00311 {
00312     uint32_t addr0 = ((uint32_t)(uintptr_t)HAL_FLASH_data + 512) & ~0x1fff;
00313     uint32_t addr1 = addr0 + 512;
00314     uint32_t addr = 0;
00315     uint32_t val0 = REG32(addr0 + 500);
00316     uint32_t val1 = REG32(addr1 + 500);
00317     if (val0 > val1)
00318     {
00319         // 优先使用扇区0
00320         if (HAL_FLASH_SecurityParameterCrc(flash, addr0) == HAL_FLASH_ReadUint32(flash, addr0 + 504))
00321         { // 确保校验通过
00322             addr = addr0;
00323         }
00324         else if (HAL_FLASH_SecurityParameterCrc(flash, addr1) == HAL_FLASH_ReadUint32(flash, addr1 +
504))
00325         {
00326             addr = addr1;
00327         }
00328         else
00329         {
00330             // 两个扇区数据都丢失, 则返回默认参数
00331             // 运行时flash被误擦除
00332             // flash未初始化
00333             // flash保存的参数和默认值一致 (如果flash数据不做任何修改, 则不会触发保存的动作)
00334             addr = 0xffffffff;
00335         }
00336     }
00337 }
00338 else
00339 {
00340     // 优先使用扇区1
00341     if (HAL_FLASH_SecurityParameterCrc(flash, addr1) == HAL_FLASH_ReadUint32(flash, addr1 + 504))
00342     { // 确保校验通过
00343         addr = addr1;
00344     }
00345     else if (HAL_FLASH_SecurityParameterCrc(flash, addr0) == HAL_FLASH_ReadUint32(flash, addr0 +
504))
00346     {
00347         addr = addr0;
00348     }
00349     else
00350     {
00351         // 两个扇区数据都丢失, 则返回默认参数
00352         // 运行时flash被误擦除
00353         // flash未初始化
00354         // flash保存的参数和默认值一致 (如果flash数据不做任何修改, 则不会触发保存的动作)
00355         addr = 0xffffffff;
00356     }
00357 }
00358 if (len != 0)
00359 {
00360     if (addr != 0xffffffff)
00361     {
00362         HAL_FLASH_Read(flash, addr, data, len);
00363     }
00364     else
00365     {
00366         uint32_t i;

```

```

00367         for (i = 0; i < len; i++)
00368         {
00369             ((uint8_t *)data)[i] = ((uint8_t *)defaultdata)[i];
00370         }
00371     }
00372 }
00373 return addr;
00374 }

```

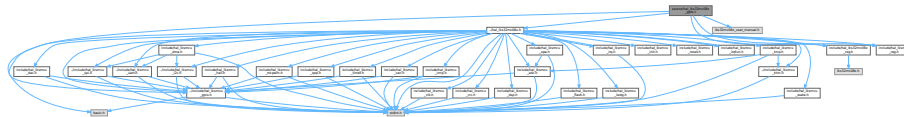
4.82 source/hal_lks32mc08x_gpio.c 文件参考

```

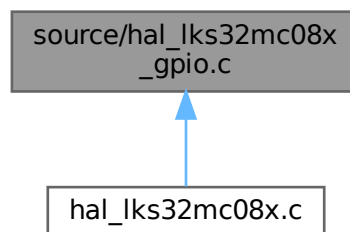
#include "../include/hal_lksmcu_gpio.h"
#include "../hal_lks32mc08x.h"
#include "lks32mc08x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc08x_reg.h"

```

hal_lks32mc08x_gpio.c 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



函数

- void **HAL_GPIO_Reset** ()
模块复位,执行后恢复所有的GPIO为悬空状态(SWD、JTAG和RST口为默认状态)
- void **HAL_GPIO_Enable** ()
启用GPIO模块,使能GPIO时钟和必要的资源
- void **HAL_GPIO_Disable** ()
禁用GPIO模块,关闭GPIO时钟和释放资源
- void **HAL_GPIO_SetMode** (HAL_GPIO_Px px, HAL_GPIO_MODE_x mode)
设置指定GPIO的工作模式
- void **HAL_GPIO_SetHigh** (HAL_GPIO_Px px)
将指定的GPIO端口输出高电平(1)

- void [HAL_GPIO_SetLow](#) ([HAL_GPIO_Px](#) px)
将指定的GPIO端口输出低电平(0)
- void [HAL_GPIO_Write](#) ([HAL_GPIO_Px](#) px, uint8_t value)
向指定的GPIO端口写入0或1
- void [HAL_GPIO_Invert](#) ([HAL_GPIO_Px](#) px)
翻转指定的GPIO端口的输出状态
- uint8_t [HAL_GPIO_GetOutData](#) ([HAL_GPIO_Px](#) px)
获取当前指定GPIO端口的输出值
- uint8_t [HAL_GPIO_GetInData](#) ([HAL_GPIO_Px](#) px)
获取当前指定GPIO端口的输入值
- void [HAL_GPIO_SetAf](#) ([HAL_GPIO_Px](#) px, [HAL_GPIO_Af](#) af)
配置GPIO的特殊功能,如UART、I2C、SPI等
- void [HAL_GPIO_SwdReuseGpioEnable](#) (void)
将SWD接口配置为普通GPIO口
- void [HAL_GPIO_SwdReuseGpioDisable](#) (void)
将SWD接口恢复为SWD功能
- void [HAL_GPIO_RstReuseGpioEnable](#) (void)
将RST接口配置为普通GPIO口
- void [HAL_GPIO_RstReuseGpioDisable](#) (void)
将RST接口恢复为RST功能

4.82.1 函数说明

4.82.1.1 HAL_GPIO_Disable()

```
void HAL_GPIO_Disable (
    void )
```

禁用GPIO模块,关闭GPIO时钟和释放资源

在文件 [hal_lks32mc08x_gpio.c](#) 第 67 行定义.

4.82.1.2 HAL_GPIO_Enable()

```
void HAL_GPIO_Enable (
    void )
```

启用GPIO模块,使能GPIO时钟和必要的资源

在文件 [hal_lks32mc08x_gpio.c](#) 第 60 行定义.

4.82.1.3 HAL_GPIO_GetInData()

```
uint8_t HAL_GPIO_GetInData (
    HAL\_GPIO\_Px px )
```

获取当前指定GPIO端口的输入值

参数

<i>px</i>	指定的GPIO端口

返回

当前GPIO端口的输入值(0或1)

在文件 [hal_lks32mc08x_gpio.c](#) 第 217 行定义.

4.82.1.4 HAL_GPIO_GetOutData()

```
uint8_t HAL_GPIO_GetOutData (
    HAL_GPIO_Px px )
```

获取当前指定GPIO端口的输出值

参数

<i>px</i>	指定的GPIO端口

返回

当前GPIO端口的输出值(0或1)

在文件 [hal_lks32mc08x_gpio.c](#) 第 194 行定义.

4.82.1.5 HAL_GPIO_Invert()

```
void HAL_GPIO_Invert (
    HAL_GPIO_Px px )
```

翻转指定的GPIO端口的输出状态

参数

<i>px</i>	指定的GPIO端口

在文件 [hal_lks32mc08x_gpio.c](#) 第 174 行定义.

这是这个函数的调用关系图:



4.82.1.6 HAL_GPIO_Reset()

```
void HAL_GPIO_Reset (
    void )
```

模块复位,执行后恢复所有的GPIO为悬空状态(SWD、JTAG和RST口为默认状态)

在文件 [hal_lks32mc08x_gpio.c](#) 第 9 行定义.

4.82.1.7 HAL_GPIO_RstReuseGpioDisable()

```
void HAL_GPIO_RstReuseGpioDisable (
    void )
```

将RST接口恢复为RST功能

在文件 [hal_lks32mc08x_gpio.c](#) 第 340 行定义.

4.82.1.8 HAL_GPIO_RstReuseGpioEnable()

```
void HAL_GPIO_RstReuseGpioEnable (
    void )
```

将RST接口配置为普通GPIO口

在文件 [hal_lks32mc08x_gpio.c](#) 第 327 行定义.

4.82.1.9 HAL_GPIO_SetAf()

```
void HAL_GPIO_SetAf (
    HAL_GPIO_Px px,
    HAL_GPIO_Afx af )
```

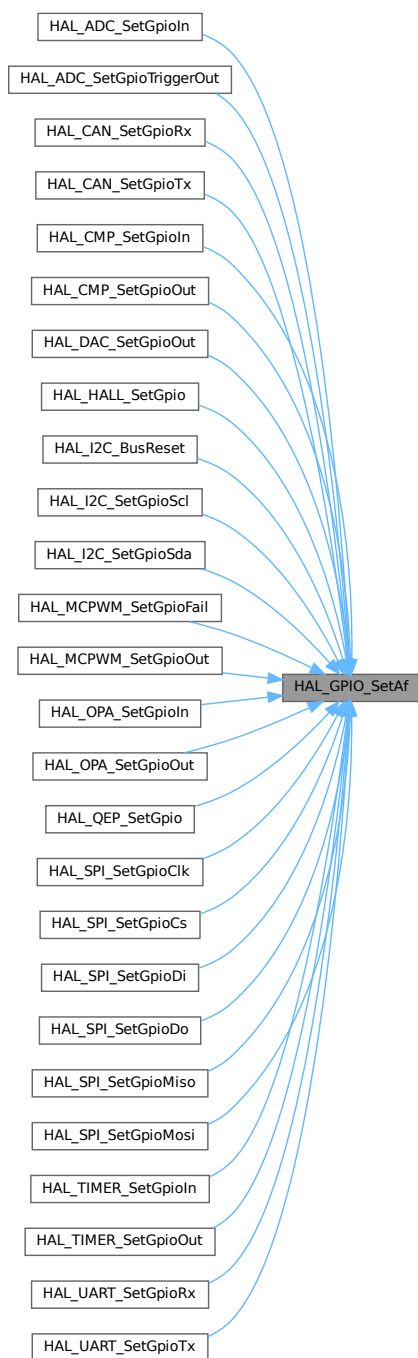
配置GPIO的特殊功能,如UART、I2C、SPI等

参数

<i>px</i>	指定的GPIO端口
<i>af</i>	要配置的特殊功能代码

在文件 [hal_lks32mc08x_gpio.c](#) 第 240 行定义。

这是这个函数的调用关系图：



4.82.1.10 HAL_GPIO_SetHigh()

```
void HAL_GPIO_SetHigh (
    HAL_GPIO_Px px )
```

将指定的GPIO端口输出高电平(1)

参数

<i>px</i>	指定的GPIO端口

注解

在INOUT模式下设为悬空,INUPOUT模式下设为上拉

在文件 [hal_lks32mc08x_gpio.c](#) 第 136 行定义.

4.82.1.11 HAL_GPIO_SetLow()

```
void HAL_GPIO_SetLow (
    HAL_GPIO_Px px )
```

将指定的GPIO端口输出低电平(0)

参数

<i>px</i>	指定的GPIO端口

在文件 [hal_lks32mc08x_gpio.c](#) 第 148 行定义.

4.82.1.12 HAL_GPIO_SetMode()

```
void HAL_GPIO_SetMode (
    HAL_GPIO_Px px,
    HAL_GPIO_MODE_x mode )
```

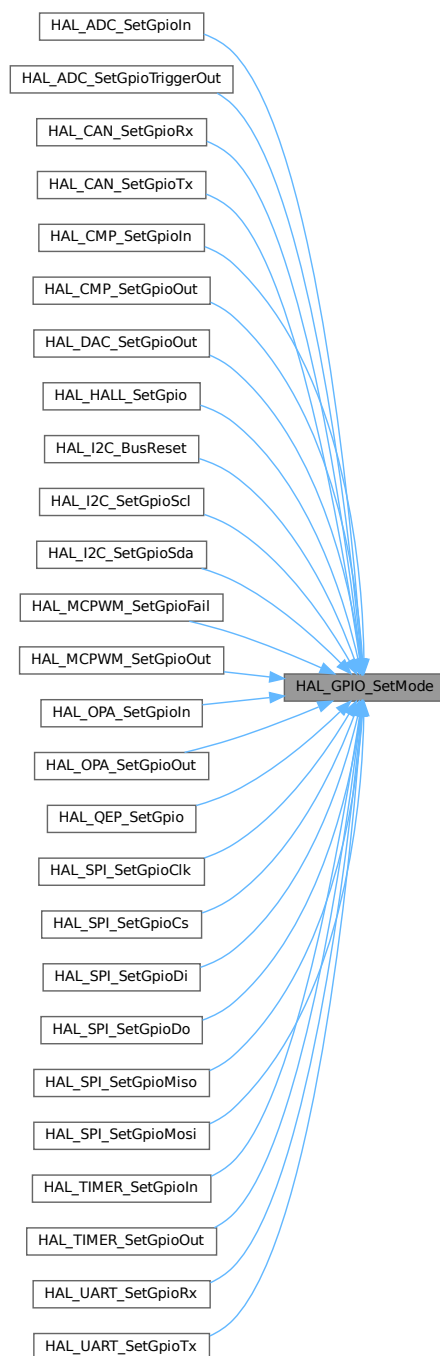
设置指定GPIO的工作模式

参数

<i>px</i>	指定的GPIO端口
<i>mode</i>	要设置的工作模式

在文件 [hal_lks32mc08x_gpio.c](#) 第 76 行定义.

这是这个函数的调用关系图:



4.82.1.13 HAL_GPIO_SwdReuseGpioDisable()

```
void HAL_GPIO_SwdReuseGpioDisable (
    void )
```

将SWD接口恢复为SWD功能

在文件 [hal_lks32mc08x_gpio.c](#) 第 314 行定义。

4.82.1.14 HAL_GPIO_SwdReuseGpioEnable()

```
void HAL_GPIO_SwdReuseGpioEnable (
    void )
```

将SWD接口配置为普通GPIO口

在文件 [hal_lks32mc08x_gpio.c](#) 第 301 行定义。

4.82.1.15 HAL_GPIO_Write()

```
void HAL_GPIO_Write (
    HAL_GPIO_Px px,
    uint8_t value )
```

向指定的GPIO端口写入0或1

参数

<i>px</i>	指定的GPIO端口
<i>value</i>	要写入的值(0或1)

在文件 [hal_lks32mc08x_gpio.c](#) 第 161 行定义。

4.83 hal_lks32mc08x_gpio.c

[浏览该文件的文档。](#)

```
00001 #include "../include/hal_lksmcu_gpio.h"
00002 #include "../hal_lks32mc08x.h"
00003 #include "lks32mc08x_user_manual.h"
00004 #include "../include/hal_lksmcu_reg.h"
00005 #include "../include/hal_lks32mc08x_reg.h"
00009 void HAL_GPIO_Reset ()
00010 {
00011     // 系统寄存器解锁
00012     HAL_REG_SYS_UNLOCK();
00013     HAL_REG_WRITE(HAL_GPIO[0]->PIE, 0);
00014     HAL_REG_WRITE(HAL_GPIO[0]->POE, 0);
00015     HAL_REG_WRITE(HAL_GPIO[0]->PDO, 0);
00016     HAL_REG_WRITE(HAL_GPIO[0]->PUE, 0);
00017     HAL_REG_WRITE(HAL_GPIO[0]->PODE, 0);
00018     HAL_REG_WRITE(HAL_GPIO[0]->F3210, 0);
00019     HAL_REG_WRITE(HAL_GPIO[0]->F7654, 0);
00020     HAL_REG_WRITE(HAL_GPIO[0]->FBA98, 0);
00021     HAL_REG_WRITE(HAL_GPIO[0]->FFEDC, 0);
00022
00023     HAL_REG_WRITE(HAL_GPIO[1]->PIE, 0);
00024     HAL_REG_WRITE(HAL_GPIO[1]->POE, 0);
00025     HAL_REG_WRITE(HAL_GPIO[1]->PDO, 0);
00026     HAL_REG_WRITE(HAL_GPIO[1]->PUE, 0);
00027     HAL_REG_WRITE(HAL_GPIO[1]->PODE, 0);
00028     HAL_REG_WRITE(HAL_GPIO[1]->F3210, 0);
00029     HAL_REG_WRITE(HAL_GPIO[1]->F7654, 0);
00030     HAL_REG_WRITE(HAL_GPIO[1]->FBA98, 0);
00031     HAL_REG_WRITE(HAL_GPIO[1]->FFEDC, 0);
00032
00033     HAL_REG_WRITE(HAL_GPIO[2]->PIE, 0);
00034     HAL_REG_WRITE(HAL_GPIO[2]->POE, 0);
00035     HAL_REG_WRITE(HAL_GPIO[2]->PDO, 0);
00036     HAL_REG_WRITE(HAL_GPIO[2]->PUE, 0);
00037     HAL_REG_WRITE(HAL_GPIO[2]->PODE, 0);
00038     HAL_REG_WRITE(HAL_GPIO[2]->F3210, 0);
00039     HAL_REG_WRITE(HAL_GPIO[2]->F7654, 0);
```

```

00040     HAL_REG_WRITE(HAL_GPIO[2]->FBA98, 0);
00041     HAL_REG_WRITE(HAL_GPIO[2]->FFEDC, 0);
00042
00043     HAL_REG_WRITE(HAL_GPIO[3]->PIE, 0);
00044     HAL_REG_WRITE(HAL_GPIO[3]->POE, 0);
00045     HAL_REG_WRITE(HAL_GPIO[3]->PDO, 0);
00046     HAL_REG_WRITE(HAL_GPIO[3]->PUE, 0);
00047     HAL_REG_WRITE(HAL_GPIO[3]->PODE, 0);
00048     HAL_REG_WRITE(HAL_GPIO[3]->F3210, 0);
00049     HAL_REG_WRITE(HAL_GPIO[3]->F7654, 0);
00050     HAL_REG_WRITE(HAL_GPIO[3]->FBA98, 0);
00051     HAL_REG_WRITE(HAL_GPIO[3]->FFEDC, 0);
00052     HAL_REG_RESET(SYS_RST_CFG, SYS_RST_CFG_RST_IO_BIT);
00053     // 系统寄存器上锁
00054     HAL_REG_SYS_LOCK();
00055 }
00056
00060 void HAL_GPIO_Enable()
00061 {
00062 }
00063
00067 void HAL_GPIO_Disable()
00068 {
00069 }
00070
00076 void HAL_GPIO_SetMode(HAL_GPIO_Px px, HAL_GPIO_MODE_x mode)
00077 {
00078     uint8_t p    = (px >> 8) & 0xff;
00079     uint8_t x    = (px >> 0) & 0xff;
00080     uint16_t bitx = 1 << x;
00081     // 默认电平设置
00082     if (HAL_REG_READ_BIT(mode, HAL_GPIO_MODE_DEFAULT_0))
00083     {
00084         HAL_REG_RESET(HAL_GPIO[p]->PDO, bitx);
00085     }
00086     else if (HAL_REG_READ_BIT(mode, HAL_GPIO_MODE_DEFAULT_1))
00087     {
00088         HAL_REG_SET(HAL_GPIO[p]->PDO, bitx);
00089     }
00090     else
00091     {
00092     }
00093     // 开漏设置
00094     if (HAL_REG_READ_BIT(mode, HAL_GPIO_MODE_PODE_EN))
00095     {
00096         HAL_REG_SET(HAL_GPIO[p]->PODE, bitx);
00097     }
00098     else
00099     {
00100         HAL_REG_RESET(HAL_GPIO[p]->PODE, bitx);
00101     }
00102     // 上拉设置
00103     if (HAL_REG_READ_BIT(mode, HAL_GPIO_MODE_UP_EN))
00104     {
00105         HAL_REG_SET(HAL_GPIO[p]->PUE, bitx);
00106     }
00107     else
00108     {
00109         HAL_REG_RESET(HAL_GPIO[p]->PUE, bitx);
00110     }
00111     // 输出设置
00112     if (HAL_REG_READ_BIT(mode, HAL_GPIO_MODE_OUT_EN))
00113     {
00114         HAL_REG_SET(HAL_GPIO[p]->POE, bitx);
00115     }
00116     else
00117     {
00118         HAL_REG_RESET(HAL_GPIO[p]->POE, bitx);
00119     }
00120     // 输入使能
00121     if (HAL_REG_READ_BIT(mode, HAL_GPIO_MODE_IN_EN))
00122     {
00123         HAL_REG_SET(HAL_GPIO[p]->PIE, bitx);
00124     }
00125     else
00126     {
00127         HAL_REG_RESET(HAL_GPIO[p]->PIE, bitx);
00128     }
00129 }
00130
00136 void HAL_GPIO_SetHigh(HAL_GPIO_Px px)
00137 {
00138     uint8_t p    = (px >> 8) & 0xff;
00139     uint8_t x    = (px >> 0) & 0xff;
00140     uint32_t bitx = 1 << x;
00141     HAL_REG_SET(HAL_GPIO[p]->PDO, bitx);
00142 }

```

```

00143
00148 void HAL_GPIO_SetLow(HAL_GPIO_TypeDef px)
00149 {
00150     uint8_t p      = (px >> 8) & 0xff;
00151     uint8_t x      = (px >> 0) & 0xff;
00152     uint32_t bitx = 1 << x;
00153     HAL_REG_RESET(HAL_GPIO[p]->PDO, bitx);
00154 }
00155
00161 void HAL_GPIO_Write(HAL_GPIO_TypeDef px, uint8_t value)
00162 {
00163     uint8_t p      = (px >> 8) & 0xff;
00164     uint8_t x      = (px >> 0) & 0xff;
00165     uint32_t bitx = 1 << x;
00166     uint32_t valx = value << x;
00167     HAL_REG_WRITE_BIT(HAL_GPIO[p]->PDO, bitx, valx);
00168 }
00169
00174 void HAL_GPIO_Invert(HAL_GPIO_TypeDef px)
00175 {
00176     uint8_t p      = (px >> 8) & 0xff;
00177     uint8_t x      = (px >> 0) & 0xff;
00178     uint32_t bitx = 1 << x;
00179     if (HAL_REG_READ_BIT(HAL_GPIO[p]->PDO, bitx))
00180     {
00181         HAL_REG_RESET(HAL_GPIO[p]->PDO, bitx);
00182     }
00183     else
00184     {
00185         HAL_REG_SET(HAL_GPIO[p]->PDO, bitx);
00186     }
00187 }
00188
00194 uint8_t HAL_GPIO_GetOutData(HAL_GPIO_TypeDef px)
00195 {
00196     uint8_t p      = (px >> 8) & 0xff;
00197     uint8_t x      = (px >> 0) & 0xff;
00198     uint32_t bitx = 1 << x;
00199     uint32_t valx = HAL_REG_READ_BIT(HAL_GPIO[p]->PDO, bitx);
00200     uint8_t result;
00201     if (valx)
00202     {
00203         result = 1;
00204     }
00205     else
00206     {
00207         result = 0;
00208     }
00209     return result;
00210 }
00211
00217 uint8_t HAL_GPIO_GetInData(HAL_GPIO_TypeDef px)
00218 {
00219     uint8_t p      = (px >> 8) & 0xff;
00220     uint8_t x      = (px >> 0) & 0xff;
00221     uint32_t bitx = 1 << x;
00222     uint32_t valx = HAL_REG_READ_BIT(HAL_GPIO[p]->PDI, bitx);
00223     uint8_t result;
00224     if (valx)
00225     {
00226         result = 1;
00227     }
00228     else
00229     {
00230         result = 0;
00231     }
00232     return result;
00233 }
00234
00240 void HAL_GPIO_SetAf(HAL_GPIO_TypeDef px, HAL_GPIO_Af_TypeDef af)
00241 {
00242     uint8_t p = (px >> 8) & 0xff;
00243     uint8_t x = (px >> 0) & 0xff;
00244     af &= 0xf; // 限制af的取值范围为0-15
00245     switch (x)
00246     {
00247     case 0:
00248         HAL_REG_WRITE_BIT_MASK_POS(HAL_GPIO[p]->F3210, GPIOx_F3210_F0, af);
00249         break;
00250     case 1:
00251         HAL_REG_WRITE_BIT_MASK_POS(HAL_GPIO[p]->F3210, GPIOx_F3210_F1, af);
00252         break;
00253     case 2:
00254         HAL_REG_WRITE_BIT_MASK_POS(HAL_GPIO[p]->F3210, GPIOx_F3210_F2, af);
00255         break;
00256     case 3:
00257         HAL_REG_WRITE_BIT_MASK_POS(HAL_GPIO[p]->F3210, GPIOx_F3210_F3, af);

```

```

00258         break;
00259     case 4:
00260         HAL_REG_WRITE_BIT_MASK_POS(HAL_GPIO[p]->F7654, GPIOx_F7654_F4, af);
00261         break;
00262     case 5:
00263         HAL_REG_WRITE_BIT_MASK_POS(HAL_GPIO[p]->F7654, GPIOx_F7654_F5, af);
00264         break;
00265     case 6:
00266         HAL_REG_WRITE_BIT_MASK_POS(HAL_GPIO[p]->F7654, GPIOx_F7654_F6, af);
00267         break;
00268     case 7:
00269         HAL_REG_WRITE_BIT_MASK_POS(HAL_GPIO[p]->F7654, GPIOx_F7654_F7, af);
00270         break;
00271     case 8:
00272         HAL_REG_WRITE_BIT_MASK_POS(HAL_GPIO[p]->FBA98, GPIOx_FBA98_F8, af);
00273         break;
00274     case 9:
00275         HAL_REG_WRITE_BIT_MASK_POS(HAL_GPIO[p]->FBA98, GPIOx_FBA98_F9, af);
00276         break;
00277     case 10:
00278         HAL_REG_WRITE_BIT_MASK_POS(HAL_GPIO[p]->FBA98, GPIOx_FBA98_F10, af);
00279         break;
00280     case 11:
00281         HAL_REG_WRITE_BIT_MASK_POS(HAL_GPIO[p]->FBA98, GPIOx_FBA98_F11, af);
00282         break;
00283     case 12:
00284         HAL_REG_WRITE_BIT_MASK_POS(HAL_GPIO[p]->FFEDC, GPIOx_FFEDC_F12, af);
00285         break;
00286     case 13:
00287         HAL_REG_WRITE_BIT_MASK_POS(HAL_GPIO[p]->FFEDC, GPIOx_FFEDC_F13, af);
00288         break;
00289     case 14:
00290         HAL_REG_WRITE_BIT_MASK_POS(HAL_GPIO[p]->FFEDC, GPIOx_FFEDC_F14, af);
00291         break;
00292     case 15:
00293         HAL_REG_WRITE_BIT_MASK_POS(HAL_GPIO[p]->FFEDC, GPIOx_FFEDC_F15, af);
00294         break;
00295     }
00296 }
00297
00301 void HAL_GPIO_SwdReuseGpioEnable(void)
00302 {
00303     // 系统寄存器解锁
00304     HAL_REG_SYS_UNLOCK();
00305     // SYS.IO.CFG[6] SWD 复用控制信号,默认配置为 SWD 1:P1.8 复用为 SWCLK,P1.9 复用为 SWDIO
00306     HAL_REG_RESET(SYS_RST_CFG, SYS_RST_CFG_RST_IO_BIT);
00307     // 系统寄存器上锁
00308     HAL_REG_SYS_LOCK();
00309 }
00310
00314 void HAL_GPIO_SwdReuseGpioDisable(void)
00315 {
00316     // 系统寄存器解锁
00317     HAL_REG_SYS_UNLOCK();
00318     // SYS.IO.CFG[6] SWD 复用控制信号,默认配置为 SWD 1:P1.8 复用为 SWCLK,P1.9 复用为 SWDIO
00319     HAL_REG_SET(SYS_RST_CFG, SYS_RST_CFG_RST_IO_BIT);
00320     // 系统寄存器上锁
00321     HAL_REG_SYS_LOCK();
00322 }
00323
00327 void HAL_GPIO_RstReuseGpioEnable(void)
00328 {
00329     // 系统寄存器解锁
00330     HAL_REG_SYS_UNLOCK();
00331     // SYS.IO.CFG[5] RSTn/P0.2 复用控制信号,默认配置为 RSTn 0:RSTn
00332     HAL_REG_SET(SYS_RST_CFG, SYS_RST_CFG_RST_IO_BIT);
00333     // 系统寄存器上锁
00334     HAL_REG_SYS_LOCK();
00335 }
00336
00340 void HAL_GPIO_RstReuseGpioDisable(void)
00341 {
00342     // 系统寄存器解锁
00343     HAL_REG_SYS_UNLOCK();
00344     // SYS.IO.CFG[5] RSTn/P0.2 复用控制信号,默认配置为 RSTn 0:RSTn
00345     HAL_REG_RESET(SYS_RST_CFG, SYS_RST_CFG_RST_IO_BIT);
00346     // 系统寄存器上锁
00347     HAL_REG_SYS_LOCK();
00348 }

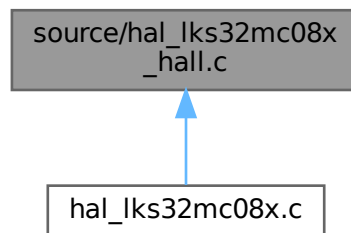
```

4.84 source/hal_lks32mc08x_hall.c 文件参考

```
#include "../include/hal_lksmcu_hall.h"
#include "../hal_lks32mc08x.h"
#include "lks32mc08x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc08x_reg.h"
hal_lks32mc08x_hall.c 的引用(Include)关系图:
```



此图展示该文件直接或间接的被哪些文件引用了:



函数

- void [HAL_HALL_Reset](#) (HAL_HALL_x hall)
模块复位
- void [HAL_HALL_Enable](#) (HAL_HALL_x hall)
模块打开
- void [HAL_HALL_Disable](#) (HAL_HALL_x hall)
模块关闭
- void [HAL_HALL_SetFreq](#) (HAL_HALL_x hall, uint32_t freq)
设置hall频率
- void [HAL_HALL_SetTh](#) (HAL_HALL_x hall_x, uint32_t th)
设置hall阈值
- uint32_t [HAL_HALL_ReadTh](#) (HAL_HALL_x hall)
获取当前hall的周期计数值
- uint8_t [HAL_HALL_ReadHallData](#) (HAL_HALL_x hall)
获取当前hall的数据
- uint32_t [HAL_HALL_ReadHallWidth](#) (HAL_HALL_x hall)
获取当前hall的宽度
- uint32_t [HAL_HALL_GetCNT](#) (HAL_HALL_x hall)

获取当前 *hall* 计数器的值

- void [HAL_HALL_SetGpio](#) ([HAL_HALL_x](#) hall, [HAL_GPIO_Px](#) px)
设置 *hall* 输入引脚
- void [HAL_HALL_StartCounting](#) ([HAL_HALL_x](#) hall)
开始计数 *HAL_HALL_Enable* 后默认开始计数
- void [HAL_HALL_StopCounting](#) ([HAL_HALL_x](#) hall)
停止计数(同时计数器回到0)
- void [HAL_HALL_SetFilter](#) ([HAL_HALL_x](#) hall_x, uint32_t filter_mclk)
设置滤波器
- void [HAL_HALL_SetFilteruS](#) ([HAL_HALL_x](#) hall, uint32_t filter_uS)
设置滤波, 单位: *uS*
- void [HAL_HALL_SetFilter75](#) ([HAL_HALL_x](#) hall_x, uint8_t ena)
设置75滤波使能(第一级滤波,连续采样7次,5次值一致)

变量

- static uint32_t [hal_hall_th](#) = 0
- static uint32_t [hal_hall_filter](#) = 0

4.84.1 函数说明

4.84.1.1 HAL_HALL_Disable()

```
void HAL_HALL_Disable (
    HAL\_HALL\_x hall )
```

模块关闭

参数

<i>hall</i>	模块号

在文件 [hal_lks32mc08x.hall.c](#) 第 41 行定义.

4.84.1.2 HAL_HALL_Enable()

```
void HAL_HALL_Enable (
    HAL\_HALL\_x hall )
```

模块打开

参数

<i>hall</i>	模块号

在文件 [hal_lks32mc08x.hall.c](#) 第 29 行定义.

4.84.1.3 HAL_HALL_GetCNT()

```
uint32_t HAL_HALL_GetCNT (
    HAL_HALL_x hall )
```

获取当前hall计数器的值

参数

<i>hall</i>	模块号

返回

计数值

在文件 [hal_lks32mc08x.hall.c](#) 第 124 行定义.

4.84.1.4 HAL_HALL_ReadHallData()

```
uint8_t HAL_HALL_ReadHallData (
    HAL_HALL_x hall )
```

获取当前hall的数据

参数

<i>hall</i>	模块号

返回

数据值

在文件 [hal_lks32mc08x.hall.c](#) 第 104 行定义.

4.84.1.5 HAL_HALL_ReadHallWidth()

```
uint32_t HAL_HALL_ReadHallWidth (
    HAL_HALL_x hall )
```

获取当前hall的宽度

参数

<i>hall</i>	模块号

返回

宽度值

在文件 [hal_lks32mc08x.hall.c](#) 第 114 行定义.

4.84.1.6 HAL_HALL_ReadTh()

```
uint32_t HAL_HALL_ReadTh (
    HAL_HALL_x hall )
```

获取当前hall的周期计数值

参数

<i>hall</i>	模块号

返回

周期计数值

在文件 [hal_lks32mc08x.hall.c](#) 第 94 行定义.

4.84.1.7 HAL_HALL_Reset()

```
void HAL_HALL_Reset (
    HAL_HALL_x hall )
```

模块复位

参数

<i>hall</i>	模块号

在文件 [hal_lks32mc08x.hall.c](#) 第 14 行定义.

4.84.1.8 HAL_HALL_SetFilter()

```
void HAL_HALL_SetFilter (
    HAL_HALL_x hall_x,
    uint32_t filter_mclk )
```

璁剧疆婁ゆ尝

参数

<i>hall</i>	模块号
<i>filter_mclk</i>	滤波值

在文件 `hal_lks32mc08x.hall.c` 第 165 行定义.

这是这个函数的调用关系图:



4.84.1.9 HAL_HALL_SetFilter75()

```
void HAL_HALL_SetFilter75 (
    HAL_HALL_x hall_x,
    uint8_t ena )
```

设置7/5滤波使能(第一级滤波,连续采样7次,5次值一致)

参数

<i>hall_x</i>	模块号
<i>ena</i>	0:关闭(默认) 非0:使能

在文件 `hal_lks32mc08x.hall.c` 第 210 行定义.

4.84.1.10 HAL_HALL_SetFilteruS()

```
void HAL_HALL_SetFilteruS (
    HAL_HALL_x hall,
    uint32_t filter_uS )
```

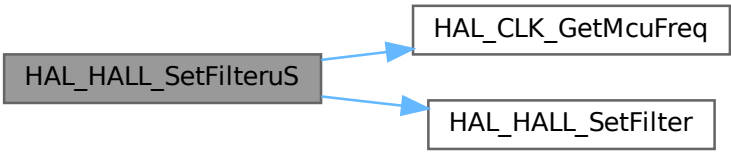
设置滤波，单位： uS

参数

<i>hall</i>	模块号
<i>filter_uS</i>	滤波值

在文件 `hal_lks32mc08x.hall.c` 第 199 行定义.

函数调用图:



4.84.1.11 HAL_HALL_SetFreq()

```
void HAL_HALL_SetFreq (
    HAL_HALL_x hall,
    uint32_t freq )
```

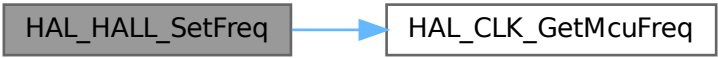
设置hall频率

参数

hall	模块号
freq	频率值

在文件 [hal_lks32mc08x_hall.c](#) 第 54 行定义.

函数调用图:



4.84.1.12 HAL_HALL_SetGpio()

```
void HAL_HALL_SetGpio (
    HAL_HALL_x hall,
    HAL_GPIO_Px px )
```

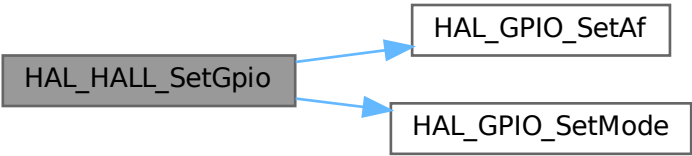
设置hall输入引脚

参数

<i>hall</i>	模块号
<i>px</i>	引脚

在文件 [hal_lks32mc08x.hall.c](#) 第 134 行定义.

函数调用图:



4.84.1.13 HAL_HALL_SetTh()

```
void HAL_HALL_SetTh (
    HAL_HALL_x hall_x,
    uint32_t th )
```

设置hall阈值

参数

<i>hall</i>	模块号
<i>th</i>	阈值

在文件 [hal_lks32mc08x.hall.c](#) 第 66 行定义.

4.84.1.14 HAL_HALL_StartCounting()

```
void HAL_HALL_StartCounting (
    HAL_HALL_x hall )
```

开始计数 HAL_HALL_Enable后默认开始计数

参数

<i>hall</i>	模块号

在文件 [hal_lks32mc08x_hall.c](#) 第 145 行定义.

4.84.1.15 HAL_HALL_StopCounting()

```
void HAL_HALL_StopCounting (
    HAL_HALL_x hall )
```

停止计数(同时计数器回到0)

参数

<i>hall</i>	模块号

在文件 [hal_lks32mc08x_hall.c](#) 第 154 行定义.

4.84.2 变量说明

4.84.2.1 hal_hall_filter

```
uint32_t hal_hall_filter = 0 [static]
```

在文件 [hal_lks32mc08x_hall.c](#) 第 9 行定义.

4.84.2.2 hal_hall_th

```
uint32_t hal_hall_th = 0 [static]
```

在文件 [hal_lks32mc08x_hall.c](#) 第 8 行定义.

4.85 hal_lks32mc08x_hall.c

[浏览该文件的文档.](#)

```
00001
00002 #include "../include/hal_lksmcu_hall.h"
00003
00004 #include "../hal_lks32mc08x.h"
00005 #include "lks32mc08x_user_manual.h"
00006 #include "../include/hal_lksmcu_reg.h"
00007 #include "../include/hal_lks32mc08x_reg.h"
00008 static uint32_t hal_hall_th = 0;
00009 static uint32_t hal_hall_filter = 0;
00014 void HAL_HALL_Reset(HAL_HALL_x hall)
00015 {
00016     (void)hall;
00017     // 系统寄存器解锁
00018     HAL_REG_SYS_UNLOCK();
00019     HAL_REG_SET(SYS_SFT_RST, SYS_SFT_RST_HALL_SFT_RST_BIT);
00020     HAL_REG_RESET(SYS_SFT_RST, SYS_SFT_RST_HALL_SFT_RST_BIT);
00021     // 系统寄存器上锁
00022     HAL_REG_SYS_LOCK();
00023 }
00024
00029 void HAL_HALL_Enable(HAL_HALL_x hall)
00030 {
00031     HAL_REG_SYS_UNLOCK();
00032     HAL_REG_SET(SYS_CLK_FEN, SYS_CLK_FEN_HALL_CLK_EN_BIT);
```

```

00033     HAL_REG_SET(HAL_HALL[hall]->CFG, HALL_CFG_HALL_EN_BIT);
00034     HAL_REG_SYS_LOCK();
00035 }
00036
00041 void HAL_HALL_Disable(HAL_HALL_x hall)
00042 {
00043     HAL_REG_SYS_UNLOCK();
00044     HAL_REG_RESET(HAL_HALL[hall]->CFG, HALL_CFG_HALL_EN_BIT);
00045     HAL_REG_RESET(SYS_CLK_FEN, SYS_CLK_FEN_HALL_CLK_EN_BIT);
00046     HAL_REG_SYS_LOCK();
00047 }
00048
00054 void HAL_HALL_SetFreq(HAL_HALL_x hall, uint32_t freq)
00055 {
00056     uint32_t mclk = HAL_CLK_GetMcuFreq();
00057     uint32_t th = (mclk / freq);
00058     HAL_REG_WRITE(HAL_HALL[hall]->TH, th);
00059 }
00060
00066 void HAL_HALL_SetTh(HAL_HALL_x hall_x, uint32_t th)
00067 {
00068     uint32_t div = 0;
00069     uint32_t filter = 0;
00070     hal_hall.th = th;
00071     th = hal_hall.th;
00072     filter = hal_hall.filter;
00073     // 自动计算分配系数
00074     for (; (th > HALL_TH_TH_MAX) || (filter > HALL_CFG_FIL_LEN_MAX);)
00075     {
00076         th >>= 1;
00077         filter >>= 1;
00078         div++;
00079         if (div > HALL_CFG_CLK_DIV_MAX)
00080         {
00081             div = HALL_CFG_CLK_DIV_MAX;
00082         }
00083     }
00084     HAL_REG_WRITE_BIT_MASK_POS(HAL_HALL[hall_x]->CFG, HALL_CFG_CLK_DIV, div);
00085     HAL_REG_WRITE_BIT_MASK_POS(HAL_HALL[hall_x]->CFG, HALL_CFG_FIL_LEN, filter);
00086     HAL_REG_WRITE_BIT_MASK_POS(HAL_HALL[hall_x]->TH, HALL_TH_TH, th);
00087 }
00088
00094 uint32_t HAL_HALL_ReadTh(HAL_HALL_x hall)
00095 {
00096     return HAL_REG_READ(HAL_HALL[hall]->TH);
00097 }
00098
00104 uint8_t HAL_HALL_ReadHallData(HAL_HALL_x hall)
00105 {
00106     return HAL_REG_READ(HAL_HALL[hall]->INFO);
00107 }
00108
00114 uint32_t HAL_HALL_ReadHallWidth(HAL_HALL_x hall)
00115 {
00116     return HAL_REG_READ(HAL_HALL[hall]->WIDTH);
00117 }
00118
00124 uint32_t HAL_HALL_GetCNT(HAL_HALL_x hall)
00125 {
00126     return HAL_REG_READ(HAL_HALL[hall]->CNT);
00127 }
00128
00134 void HAL_HALL_SetGpio(HAL_HALL_x hall, HAL_GPIO_Px px)
00135 {
00136     (void)hall;
00137     HAL_GPIO_SetMode(px, HAL_GPIO_MODE_IN);
00138     HAL_GPIO_SetAf(px, HAL_GPIO_AF_HALL);
00139 }
00140
00145 void HAL_HALL_StartCounting(HAL_HALL_x hall)
00146 {
00147     HAL_REG_SET(HAL_HALL[hall]->CFG, HALL_CFG_HALL_EN_BIT);
00148 }
00149
00154 void HAL_HALL_StopCounting(HAL_HALL_x hall)
00155 {
00156     HAL_REG_RESET(HAL_HALL[hall]->CFG, HALL_CFG_HALL_EN_BIT);
00157     HAL_REG_WRITE(HAL_HALL[hall]->CNT, 0);
00158 }
00159
00165 void HAL_HALL_SetFilter(HAL_HALL_x hall_x, uint32_t filter_mclk)
00166 {
00167     uint32_t div = 0;
00168     uint32_t filter = 0;
00169     uint32_t th = 0;
00170     if (filter_mclk != 0)
00171     {

```



```

00172     filter_mclk -= 1;
00173 }
00174 hal_hall_filter = filter_mclk;
00175
00176 th = hal_hall_th;
00177 filter = hal_hall_filter;
00178 // 自动计算分配系数
00179 for (; (th > HALL_TH_TH_MAX) || (filter > HALL_CFG_FIL_LEN_MAX);)
00180 {
00181     th >>= 1;
00182     filter >>= 1;
00183     div++;
00184     if (div > HALL_CFG_CLK_DIV_MAX)
00185     {
00186         div = HALL_CFG_CLK_DIV_MAX;
00187     }
00188 }
00189 HAL_REG_WRITE_BIT_MASK_POS(HAL_HALL[hall_x]->CFG, HALL_CFG_CLK_DIV, div);
00190 HAL_REG_WRITE_BIT_MASK_POS(HAL_HALL[hall_x]->CFG, HALL_CFG_FIL_LEN, filter);
00191 HAL_REG_WRITE_BIT_MASK_POS(HAL_HALL[hall_x]->TH, HALL_TH_TH, th);
00192 }
00193
00199 void HAL_HALL_SetFilteruS(HAL_HALL_x hall, uint32_t filter_uS)
00200 {
00201     uint32_t clk = HAL_CLK_GetMcuFreq();
00202     HAL_HALL_SetFilter(hall, (clk / 1000000) * filter_uS);
00203 }
00204
00210 void HAL_HALL_SetFilter75(HAL_HALL_x hall_x, uint8_t ena)
00211 {
00212     HAL_REG_IF_WRITE(HAL_HALL[hall_x]->CFG, HALL_CFG_FIL_75_BIT, ena);
00213 }

```

4.86 source/hal_lks32mc08x_i2c.c 文件参考

```

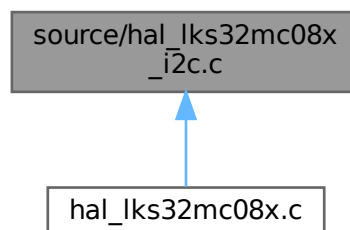
#include "../include/hal_lksmcu_i2c.h"
#include "../hal_lks32mc08x.h"
#include "lks32mc08x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc08x_reg.h"

```

hal_lks32mc08x_i2c.c 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



函数

- void [HAL_I2C_Reset](#) ([HAL_I2C_x](#) i2c_x)
复位I2C模块
- void [HAL_I2C_Enable](#) ([HAL_I2C_x](#) i2c_x)
打开I2C模块
- void [HAL_I2C_Disable](#) ([HAL_I2C_x](#) i2c_x)
关闭I2C模块
- void [HAL_I2C_SetBaud](#) ([HAL_I2C_x](#) i2c_x, uint32_t baud)
设置I2C模块波特率
- void [HAL_I2C_SetClockDiv](#) ([HAL_I2C_x](#) i2c_x, uint32_t div)
- uint32_t [HAL_I2C_GetClockDiv](#) ([HAL_I2C_x](#) i2c_x)
- uint32_t [HAL_I2C_GetClockFreq](#) ([HAL_I2C_x](#) i2c_x)
- uint32_t [HAL_I2C_GetBaud](#) ([HAL_I2C_x](#) i2c_x)
- uint8_t [HAL_I2C_TxData](#) ([HAL_I2C_x](#) i2c_x, uint8_t addr, uint8_t *data, uint32_t len)
发送数据到I2C设备
- uint8_t [HAL_I2C_RxData](#) ([HAL_I2C_x](#) i2c_x, uint8_t addr, uint8_t *data, uint32_t len)
从I2C设备接收数据
- void [HAL_I2C_SetGpioScl](#) ([HAL_I2C_x](#) i2c_x, [HAL_GPIO_Px](#) i2cgpio)
设置I2C模块的GPIO
- void [HAL_I2C_SetGpioSda](#) ([HAL_I2C_x](#) i2c_x, [HAL_GPIO_Px](#) i2cgpio)
设置I2C模块的GPIO
- void [HAL_I2C_BusReset](#) ([HAL_I2C_x](#) i2c_x)
复位I2C总线

变量

- static [HAL_GPIO_Px](#) [hal_i2cio](#) [1][2]
- static uint32_t [hal_baud](#) [1]

4.86.1 函数说明

4.86.1.1 HAL_I2C_BusReset()

```
void HAL_I2C_BusReset (  
    HAL\_I2C\_x i2c_x )
```

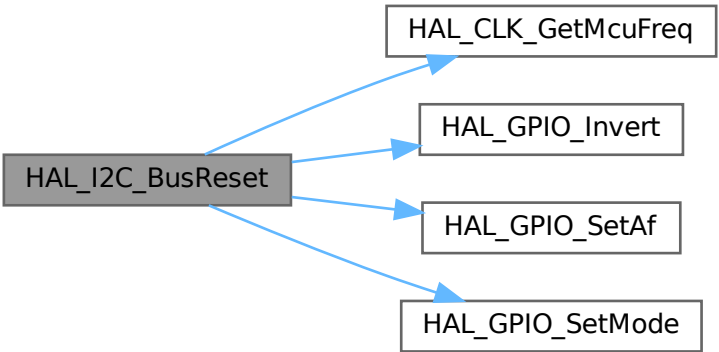
复位I2C总线

参数

i2c_x	I2C模块编号

在文件 [hal_lks32mc08x_i2c.c](#) 第 234 行定义.

函数调用图:



4.86.1.2 HAL_I2C.Disable()

```
void HAL_I2C_Disable (
    HAL_I2C_x i2c_x )
```

关闭I2C模块

参数

<i>i2c</i> _x	I2C模块编号

在文件 [hal_lks32mc08x_i2c.c](#) 第 38 行定义.

4.86.1.3 HAL_I2C.Enable()

```
void HAL_I2C_Enable (
    HAL_I2C_x i2c_x )
```

打开I2C模块

参数

<i>i2c</i> _x	I2C模块编号

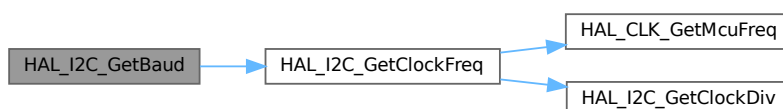
在文件 [hal_lks32mc08x_i2c.c](#) 第 26 行定义.

4.86.1.4 HAL_I2C_GetBaud()

```
uint32_t HAL_I2C_GetBaud (
    HAL_I2C_x i2c_x )
```

在文件 [hal_lks32mc08x_i2c.c](#) 第 86 行定义.

函数调用图:



4.86.1.5 HAL_I2C_GetClockDiv()

```
uint32_t HAL_I2C_GetClockDiv (
    HAL_I2C_x i2c_x )
```

在文件 [hal_lks32mc08x_i2c.c](#) 第 73 行定义.

这是这个函数的调用关系图:

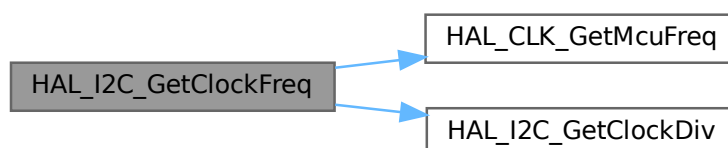


4.86.1.6 HAL_I2C_GetClockFreq()

```
uint32_t HAL_I2C_GetClockFreq (
    HAL_I2C_x i2c_x )
```

在文件 [hal_lks32mc08x_i2c.c](#) 第 80 行定义.

函数调用图:



这是这个函数的调用关系图:



4.86.1.7 HAL_I2C_Reset()

```
void HAL_I2C_Reset (
    HAL_I2C_x i2c_x )
```

复位I2C模块

参数

<i>i2c_x</i>	I2C模块编号

在文件 [hal_lks32mc08x_i2c.c](#) 第 13 行定义.

4.86.1.8 HAL_I2C_RxData()

```
uint8_t HAL_I2C_RxData (
    HAL_I2C_x i2c_x,
    uint8_t addr,
    uint8_t * data,
    uint32_t len )
```

从I2C设备接收数据

参数

<i>i2c_x</i>	I2C模块编号
<i>addr</i>	设备地址
<i>data</i>	数据缓冲区
<i>len</i>	数据长度

返回

0:成功 非0:失败

在文件 [hal_lks32mc08x_i2c.c](#) 第 160 行定义.

4.86.1.9 HAL_I2C_SetBaud()

```
void HAL_I2C_SetBaud (
    HAL_I2C_x i2c_x,
    uint32_t baud )
```

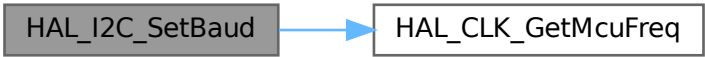
设置I2C模块波特率

参数

<i>i2c_x</i>	I2C模块编号
<i>baud</i>	波特率

在文件 [hal_lks32mc08x_i2c.c](#) 第 51 行定义.

函数调用图:



4.86.1.10 HAL_I2C_SetClockDiv()

```
void HAL_I2C_SetClockDiv (
    HAL_I2C_x i2c_x,
    uint32_t div )
```

在文件 [hal_lks32mc08x_i2c.c](#) 第 63 行定义.

4.86.1.11 HAL_I2C_SetGpioScl()

```
void HAL_I2C_SetGpioScl (
    HAL_I2C_x i2c_x,
    HAL_GPIO_Px i2cgpio )
```

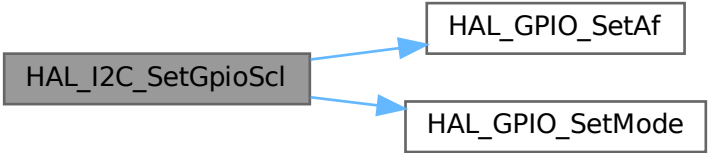
设置I2C模块的GPIO

参数

<i>i2c_x</i>	I2C模块编号
<i>i2cgpio</i>	GPIO编号

在文件 [hal_lks32mc08x.i2c.c](#) 第 209 行定义.

函数调用图:



4.86.1.12 HAL_I2C.SetGpioSda()

```
void HAL_I2C_SetGpioSda (
    HAL_I2C_x i2c_x,
    HAL_GPIO_Px i2cgpio )
```

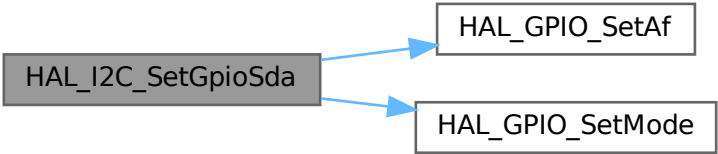
设置I2C模块的GPIO

参数

<i>i2c_x</i>	I2C模块编号
<i>i2cgpio</i>	GPIO编号

在文件 [hal_lks32mc08x.i2c.c](#) 第 222 行定义.

函数调用图:



4.86.1.13 HAL_I2C.TxData()

```
uint8_t HAL_I2C_TxData (
    HAL_I2C_x i2c_x,
```

```
uint8_t  addr,
uint8_t * data,
uint32_t len )
```

发送数据到I2C设备

参数

<i>i2c_x</i>	I2C模块编号
<i>addr</i>	设备地址
<i>data</i>	数据缓冲区
<i>len</i>	数据长度

返回

0:成功 非0:失败

在文件 [hal_lks32mc08x_i2c.c](#) 第 99 行定义.

4.86.2 变量说明

4.86.2.1 hal_baud

```
uint32_t hal_baud[1] [static]
```

在文件 [hal_lks32mc08x_i2c.c](#) 第 8 行定义.

4.86.2.2 hal_i2cio

```
HAL_GPIO_Px hal_i2cio[1][2] [static]
```

在文件 [hal_lks32mc08x_i2c.c](#) 第 7 行定义.

4.87 hal_lks32mc08x_i2c.c

[浏览该文件的文档.](#)

```
00001
00002 #include "../include/hal_lksmcu_i2c.h"
00003 #include "../hal_lks32mc08x.h"
00004 #include "lks32mc08x-user-manual.h"
00005 #include "../include/hal_lksmcu_reg.h"
00006 #include "../include/hal_lks32mc08x_reg.h"
00007 static HAL_GPIO_Px hal_i2cio[1][2];
00008 static uint32_t hal_baud[1];
00013 void HAL_I2C_Reset(HAL_I2C_x i2c_x)
00014 {
00015     (void)i2c_x;
00016     HAL_REG_SYS_UNLOCK();
00017     HAL_REG_SET(SYS_SFT_RST, SYS_SFT_RST_I2C_SFT_RST_BIT);
00018     HAL_REG_RESET(SYS_SFT_RST, SYS_SFT_RST_I2C_SFT_RST_BIT);
00019     HAL_REG_SYS_LOCK();
00020 }
00021
```



```

00026 void HAL_I2C_Enable(HAL_I2C_x i2c_x)
00027 {
00028     (void)i2c_x;
00029     HAL_REG_SYS_UNLOCK();
00030     HAL_REG_SET(SYS_CLK_FEN, SYS_CLK_FEN_I2C_CLK_EN_BIT);
00031     HAL_REG_SYS_LOCK();
00032 }
00033
00038 void HAL_I2C_Disable(HAL_I2C_x i2c_x)
00039 {
00040     (void)i2c_x;
00041     HAL_REG_SYS_UNLOCK();
00042     HAL_REG_RESET(SYS_CLK_FEN, SYS_CLK_FEN_I2C_CLK_EN_BIT);
00043     HAL_REG_SYS_LOCK();
00044 }
00045
00051 void HAL_I2C_SetBaud(HAL_I2C_x i2c_x, uint32_t baud)
00052 {
00053     uint32_t BaudRate;
00054     (void)i2c_x;
00055     hal_baud[i2c_x] = baud;
00056     BaudRate = ((uint32_t)HAL_CLK_GetMcuFreq() + ((baud * 17) >> 1)) / (baud * 17) - 1;
00057     HAL_REG_SYS_UNLOCK();
00058     HAL_REG_WRITE(SYS_CLK_DIV0, BaudRate);
00059     HAL_REG_SYS_LOCK();
00060 }
00061
00062 // 设置I2C全局工作时钟分频 (I2C0/1共享)
00063 void HAL_I2C_SetClockDiv(HAL_I2C_x i2c_x, uint32_t div)
00064 {
00065     (void)i2c_x;
00066     if (div < 1) div = 1;
00067     HAL_REG_SYS_UNLOCK();
00068     HAL_REG_WRITE(SYS_CLK_DIV0, div - 1);
00069     HAL_REG_SYS_LOCK();
00070 }
00071
00072 // 获取I2C工作时钟分频实际除数
00073 uint32_t HAL_I2C_GetClockDiv(HAL_I2C_x i2c_x)
00074 {
00075     (void)i2c_x;
00076     return HAL_REG_READ_BIT_MASK_POS(SYS_CLK_DIV0, SYS_CLK_DIV0_DIV0) + 1;
00077 }
00078
00079 // 获取I2C工作时钟频率
00080 uint32_t HAL_I2C_GetClockFreq(HAL_I2C_x i2c_x)
00081 {
00082     return HAL_CLK_GetMcuFreq() / HAL_I2C_GetClockDiv(i2c_x);
00083 }
00084
00085 // 获取当前SCL波特率
00086 uint32_t HAL_I2C_GetBaud(HAL_I2C_x i2c_x)
00087 {
00088     return HAL_I2C_GetClockFreq(i2c_x) / 17;
00089 }
00090
00099 uint8_t HAL_I2C_TxData(HAL_I2C_x i2c_x, uint8_t addr, uint8_t *data, uint32_t len)
00100 {
00101     uint32_t i = 0;
00102     HAL_REG_SET(HAL_I2C[i2c_x]->CFG, I2C0_CFG_MST_MODE_BIT);
00103     if (HAL_REG_READ_BIT(HAL_I2C[i2c_x]->MSCR, I2C0_MSCR_BUSY_BIT))
00104     {
00105         return 1; // 发送数据时,i2c总线忙碌
00106     }
00107     if (len == 0)
00108     {
00109         return 1;
00110     }
00111     HAL_REG_WRITE(HAL_I2C[i2c_x]->SCR, 0);
00112     HAL_REG_WRITE(HAL_I2C[i2c_x]->MSCR, I2C0_MSCR_START_BIT); // 触发I2C发送数据
00113     while (HAL_REG_READ_BIT(HAL_I2C[i2c_x]->MSCR, I2C0_MSCR_MST_CHECK_BIT) == 0)
00114     {
00115     }
00116     HAL_REG_WRITE(HAL_I2C[i2c_x]->DATA, addr);
00117     while (HAL_REG_READ(HAL_I2C[i2c_x]->SCR) == 0) // 等待地址发送完成
00118     {
00119     }
00120
00121     if (HAL_REG_READ_BIT(HAL_I2C[i2c_x]->SCR, I2C0_SCR_RX_ACK_BIT)) // 收到nack
00122     {
00123         HAL_REG_WRITE(HAL_I2C[i2c_x]->SCR, 0);
00124         while (HAL_REG_READ_BIT(HAL_I2C[i2c_x]->MSCR, I2C0_MSCR_BUSY_BIT)) // 等待stop信号发送完成
00125         {
00126         }
00127         return 1;
00128     }
00129     for (i = 0; i < len; i++)

```

```

00130     {
00131         HAL_REG_WRITE(HAL_I2C[i2c_x]->DATA, data[i]);
00132         HAL_REG_WRITE(HAL_I2C[i2c_x]->SCR, I2C0.SCR.DATA_DIR_BIT); // 触发发送
00133         while (HAL_REG_READ_BIT(HAL_I2C[i2c_x]->SCR, I2C0.SCR.Done_BIT) == 0) // 等待数据发送完成
00134         {
00135         }
00136         if (HAL_REG_READ_BIT(HAL_I2C[i2c_x]->SCR, I2C0.SCR.RX_ACK_BIT)) // 收到nack
00137         {
00138             HAL_REG_WRITE(HAL_I2C[i2c_x]->SCR, 0);
00139             while (HAL_REG_READ_BIT(HAL_I2C[i2c_x]->MSCR, I2C0.MSCR.BUSY_BIT)) // 等待stop信号发送完成
00140             {
00141             }
00142             return 1;
00143         }
00144     }
00145     HAL_REG_WRITE(HAL_I2C[i2c_x]->SCR, 0);
00146     while (HAL_REG_READ_BIT(HAL_I2C[i2c_x]->MSCR, I2C0.MSCR.BUSY_BIT)) // 等待stop信号发送完成
00147     {
00148     }
00149     return 0;
00150 }
00151
00160 uint8_t HAL_I2C_RxData(HAL_I2C_x i2c_x, uint8_t addr, uint8_t *data, uint32_t len)
00161 {
00162     uint32_t i;
00163     HAL_REG_SET(HAL_I2C[i2c_x]->CFG, I2C0.CFG.MST.MODE_BIT);
00164     if (HAL_REG_READ_BIT(HAL_I2C[i2c_x]->MSCR, I2C0.MSCR.BUSY_BIT))
00165     {
00166         return 1; // 发送数据时,i2c总线忙碌
00167     }
00168     if (len == 0)
00169     {
00170         return 1;
00171     }
00172     HAL_REG_WRITE(HAL_I2C[i2c_x]->SCR, 0);
00173     HAL_REG_WRITE(HAL_I2C[i2c_x]->MSCR, I2C0.MSCR.START_BIT); // 触发I2C发送数据
00174     while (HAL_REG_READ_BIT(HAL_I2C[i2c_x]->MSCR, I2C0.MSCR.MST.CHECK_BIT) == 0) // 等待start信号发送完成
00175     {
00176     }
00177     HAL_REG_WRITE(HAL_I2C[i2c_x]->DATA, addr | 1); // 发送地址数据
00178     while (HAL_REG_READ_BIT(HAL_I2C[i2c_x]->SCR) == 0) // 等待地址发送完成
00179     {
00180     }
00181     if (HAL_REG_READ_BIT(HAL_I2C[i2c_x]->SCR, I2C0.SCR.RX_ACK_BIT)) // 收到nack
00182     {
00183         HAL_REG_WRITE(HAL_I2C[i2c_x]->SCR, 0);
00184         while (HAL_REG_READ_BIT(HAL_I2C[i2c_x]->MSCR, I2C0.MSCR.BUSY_BIT)) // 等待stop信号发送完成
00185         {
00186         }
00187         return 1;
00188     }
00189     for (i = 0; i < len; i++)
00190     {
00191         HAL_REG_WRITE(HAL_I2C[i2c_x]->SCR, I2C0.SCR.BYTE_CMPLT_BIT); // 向从机发送ack响应
00192         while (HAL_REG_READ_BIT(HAL_I2C[i2c_x]->SCR, I2C0.SCR.Done_BIT) == 0) // 等待数据发送完成
00193         {
00194         }
00195         data[i] = HAL_REG_READ(HAL_I2C[i2c_x]->DATA);
00196     }
00197     HAL_REG_WRITE(HAL_I2C[i2c_x]->SCR, 0);
00198     while (HAL_REG_READ_BIT(HAL_I2C[i2c_x]->MSCR, I2C0.MSCR.BUSY_BIT)) // 等待stop信号发送完成
00199     {
00200     }
00201     return 0;
00202 }
00203
00209 void HAL_I2C_SetGpioScl(HAL_I2C_x i2c_x, HAL_GPIO_Px i2cgpio)
00210 {
00211     (void)i2c_x;
00212     HAL_GPIO_SetAf(i2cgpio, HAL_GPIO_AF_I2C);
00213     HAL_GPIO_SetMode(i2cgpio, HAL_GPIO_MODE_IN);
00214     hal_i2cio[i2c_x][0] = i2cgpio; // 保存I2C使用的Io口
00215 }
00216
00222 void HAL_I2C_SetGpioSda(HAL_I2C_x i2c_x, HAL_GPIO_Px i2cgpio)
00223 {
00224     (void)i2c_x;
00225     HAL_GPIO_SetAf(i2cgpio, HAL_GPIO_AF_I2C);
00226     HAL_GPIO_SetMode(i2cgpio, HAL_GPIO_MODE_IN);
00227     hal_i2cio[i2c_x][1] = i2cgpio; // 保存I2C使用的Io口
00228 }
00229
00234 void HAL_I2C_BusReset(HAL_I2C_x i2c_x)
00235 {
00236     uint32_t div;
00237     uint32_t i;
00238     uint32_t j;

```

```

00239 // 还原成普通IO口
00240 HAL_GPIO_SetAf(hal_i2cio[i2c.x][0], HAL_GPIO_AF_GPIO);
00241 HAL_GPIO_SetAf(hal_i2cio[i2c.x][1], HAL_GPIO_AF_GPIO);
00242 HAL_GPIO_SetMode(hal_i2cio[i2c.x][0], HAL_GPIO_MODE_INUPOUT_1);
00243 HAL_GPIO_SetMode(hal_i2cio[i2c.x][1], HAL_GPIO_MODE_INUP);
00244 // 翻转10次SCL
00245 div = (HAL_CLK_GetMcuFreq() / hal_baud[i2c.x]) >> 4;
00246 for (i = 0; i < 20; i++)
00247 {
00248     HAL_GPIO_Invert(hal_i2cio[i2c.x][0]);
00249     for (j = div; j--;) // 一个大于波特率的软件延迟
00250     {
00251         __NOP();
00252     }
00253 }
00254 // 复用回I2C
00255 HAL_GPIO_SetAf(hal_i2cio[i2c.x][0], HAL_GPIO_AF_I2C);
00256 HAL_GPIO_SetAf(hal_i2cio[i2c.x][1], HAL_GPIO_AF_I2C);
00257 HAL_GPIO_SetMode(hal_i2cio[i2c.x][0], HAL_GPIO_MODE_IN);
00258 HAL_GPIO_SetMode(hal_i2cio[i2c.x][1], HAL_GPIO_MODE_IN);
00259 }

```

4.88 source/hal_lks32mc08x_init.c 文件参考

```

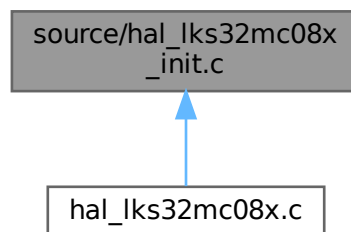
#include "../include/hal_lksmcu_init.h"
#include "../hal_lks32mc08x.h"
#include "lks32mc08x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc08x_reg.h"

```

hal_lks32mc08x_init.c 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



函数

- void HAL_INIT_SysInit (void)
- void HAL_INIT_PowerReady (HAL_INIT_POWER_x powerx)

4.88.1 函数说明

4.88.1.1 HAL_INIT_PowerReady()

```
void HAL_INIT_PowerReady (
    HAL_INIT_POWER_x powerx )
```

在文件 `hal_lks32mc08x_init.c` 第 12 行定义.

函数调用图:

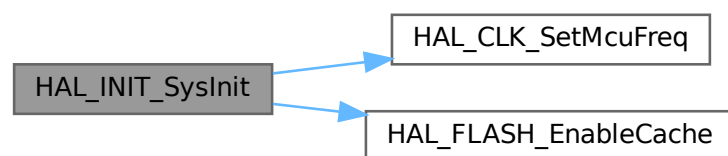


4.88.1.2 HAL_INIT_SysInit()

```
void HAL_INIT_SysInit (
    void )
```

在文件 `hal_lks32mc08x_init.c` 第 6 行定义.

函数调用图:



4.89 hal_lks32mc08x_init.c

[浏览该文件的文档.](#)

```
00001 #include "../include/hal_lksmcu_init.h"
00002 #include "../hal_lks32mc08x.h"
00003 #include "lks32mc08x_user_manual.h"
00004 #include "../include/hal_lksmcu_reg.h"
00005 #include "../include/hal_lks32mc08x_reg.h"
00006 void HAL_INIT_SysInit(void)
00007 {
00008     HAL_CLK_SetMcuFreq(HAL_CLK_FREQ_RC_96MHZ);
00009     HAL_FLASH_EnableCache(HAL_FLASH_0);
```

```

00010 }
00011
00012 void HAL_INIT_PowerReady(HAL_INIT_POWER_x powerx)
00013 {
00014     switch (powerx)
00015     {
00016         case HAL_INIT_POWER_3_3V: // 等待3.3v电源就绪, 判断阈值为2.8V
00017             __disable_irq();
00018             // 不支持3.3v掉电检测
00019             break;
00020         case HAL_INIT_POWER_5V: // 等待5.0v电源就绪, 判断阈值为4.0V
00021             __disable_irq();
00022             // 设置掉电检测电压
00023             HAL_REG_SYS_UNLOCK();
00024             HAL_REG_WRITE_BIT_MASK_POS(HAL_SYS[0]->AFE_REG6, SYS_AFE_REG6_PVDSEL, 2);
00025             HAL_REG_RESET(HAL_SYS[0]->AFE_REG6, (SYS_AFE_REG6_VSR_PDT_BIT | SYS_AFE_REG6_PD_PDT_BIT));
00026             HAL_REG_SYS_LOCK();
00027             // 等待电源就绪
00028             while (HAL_REG_READ_BIT(HAL_SYS[0]->AFE_CMP, SYS_AFE_CMP_PWR_WEAK_BIT))
00029             {
00030                 __NOP();
00031             }
00032             break;
00033     }
00034     // 清除中断标志位
00035     HAL_IRQ_ModuleClear(HAL_IRQ_MODULE_INIT0);
00036 }

```

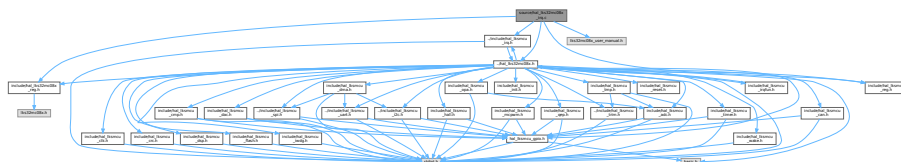
4.90 source/hal_lks32mc08x_irq.c 文件参考

```

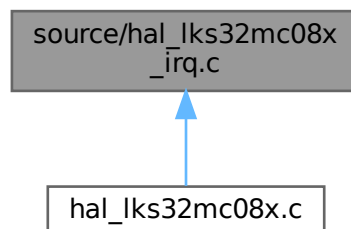
#include "../include/hal_lksmcu_irq.h"
#include "../hal_lks32mc08x.h"
#include "lks32mc08x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc08x_reg.h"

```

hal_lks32mc08x_irq.c 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



函数

- void [HAL_IRQ_Reset](#) (void)
重置中断(等效于关闭中断)
- void [HAL_IRQ_Enable](#) (void)
使能全局中断
- void [HAL_IRQ_Disable](#) (void)
关闭全局中断
- void [HAL_IRQ_ModuleEnable](#) (HAL_IRQ_MODULE_x module, uint8_t priority)
使能某个模块的中断
- void [HAL_IRQ_ModuleDisable](#) (HAL_IRQ_MODULE_x module)
关闭某个模块的中断
- void [HAL_IRQ_ModuleClear](#) (HAL_IRQ_MODULE_x module)
清除某个模块的中断

4.90.1 函数说明

4.90.1.1 HAL_IRQ_Disable()

```
void HAL_IRQ_Disable (  
    void )
```

关闭全局中断

参数

<i>irq</i>	中断编号

在文件 [hal_lks32mc08x_irq.c](#) 第 29 行定义.

这是这个函数的调用关系图:



4.90.1.2 HAL_IRQ_Enable()

```
void HAL_IRQ_Enable (  
    void )
```

使能全局中断

参数

<i>irq</i>	中断编号

在文件 [hal_lks32mc08x_irq.c](#) 第 20 行定义.

4.90.1.3 HAL_IRQ_ModuleClear()

```
void HAL_IRQ_ModuleClear (
    HAL_IRQ_MODULE_x module )
```

清除某个模块的中断

参数

<i>module</i>	模块编号

在文件 [hal_lks32mc08x_irq.c](#) 第 57 行定义.

这是这个函数的调用关系图:



4.90.1.4 HAL_IRQ_ModuleDisable()

```
void HAL_IRQ_ModuleDisable (
    HAL_IRQ_MODULE_x module )
```

关闭某个模块的中断

参数

<i>module</i>	模块编号

在文件 [hal_lks32mc08x_irq.c](#) 第 49 行定义.

4.90.1.5 HAL_IRQ_ModuleEnable()

```
void HAL_IRQ_ModuleEnable (
    HAL_IRQ_MODULE_x module,
    uint8_t priority )
```

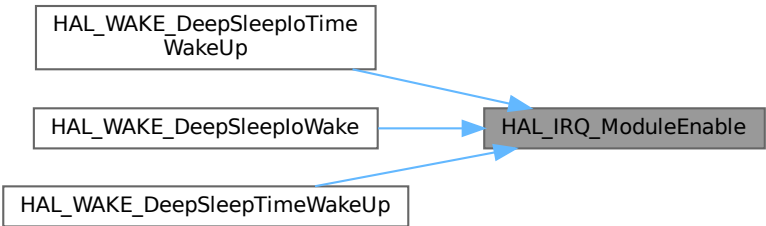
使能某个模块的中断

参数

<i>module</i>	模块编号
<i>priority</i>	中断优先级

在文件 [hal_lks32mc08x_irq.c](#) 第 39 行定义.

这是这个函数的调用关系图:



4.90.1.6 HAL_IRQ_Reset()

```
void HAL_IRQ_Reset (
    void )
```

重置中断(等效于关闭中断)

参数

<i>irq</i>	中断编号

在文件 [hal_lks32mc08x_irq.c](#) 第 11 行定义.

函数调用图:



4.91 hal_lks32mc08x_irq.c

[浏览该文件的文档.](#)

```

00001 #include "../include/hal_lksmcu_irq.h"
00002 #include "../hal_lks32mc08x.h"
00003 #include "lks32mc08x_user_manual.h"
00004 #include "../include/hal_lksmcu_reg.h"
00005 #include "../include/hal_lks32mc08x_reg.h"
00006
00011 void HAL_IRQ_Reset(void)
00012 {
00013     HAL_IRQ_Disable();
00014 }
00015
00020 void HAL_IRQ_Enable(void)
00021 {
00022     __enable_irq();
00023 }
00024
00029 void HAL_IRQ_Disable(void)
00030 {
00031     __disable_irq();
00032 }
00033
00039 void HAL_IRQ_ModuleEnable(HAL_IRQ_MODULE_x module, uint8_t priority)
00040 {
00041     NVIC.SetPriority((IRQn.Type)module, priority);
00042     NVIC.EnableIRQ((IRQn.Type)module);
00043 }
00044
00049 void HAL_IRQ_ModuleDisable(HAL_IRQ_MODULE_x module)
00050 {
00051     NVIC.DisableIRQ((IRQn.Type)module);
00052 }
00057 void HAL_IRQ_ModuleClear(HAL_IRQ_MODULE_x module)
00058 {
00059     NVIC.ClearPendingIRQ((IRQn.Type)module);
00060 }

```

4.92 source/hal_lks32mc08x_irqfun.c 文件参考

```

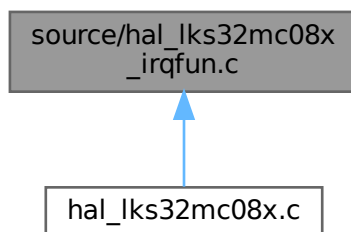
#include "../hal_lks32mc08x.h"
#include "lks32mc08x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc08x_reg.h"
#include "../include/hal_lksmcu_irqfun.h"

```

hal_lks32mc08x_irqfun.c 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了：



函数

- uint8_t HAL_HALL_IrqChange (HAL_HALL_x hall)
中断处理函数
- uint8_t HAL_HALL_IrqChangeFlg (HAL_HALL_x hall)
中断信号（和 HAL_HALL_IrqChange 的区别是不判断中断使能）
- void HAL_HALL_IrqChangeEnable (HAL_HALL_x hall)
中断使能
- void HAL_HALL_IrqChangeDisable (HAL_HALL_x hall)
中断禁止
- void HAL_HALL_IrqChangeClear (HAL_HALL_x hall)
中断清除
- uint8_t HAL_HALL_IrqOver (HAL_HALL_x hall)
中断处理函数
- uint8_t HAL_HALL_IrqOverFlg (HAL_HALL_x hall)
中断信号（和 HAL_HALL_IrqOver 的区别是不判断中断使能）
- void HAL_HALL_IrqOverEnable (HAL_HALL_x hall)
中断使能
- void HAL_HALL_IrqOverDisable (HAL_HALL_x hall)
中断禁止
- void HAL_HALL_IrqOverClear (HAL_HALL_x hall)
中断清除
- uint8_t HAL_ADC_IrqSamp1 (HAL_ADC_x adc)
中断处理函数
- uint8_t HAL_ADC_IrqSamp1Flg (HAL_ADC_x adc)
中断信号（和 HAL_ADC_IrqSamp1 的区别是不判断中断使能）
- void HAL_ADC_IrqSamp1Enable (HAL_ADC_x adc)
中断使能
- void HAL_ADC_IrqSamp1Disable (HAL_ADC_x adc)
中断禁止
- void HAL_ADC_IrqSamp1Clear (HAL_ADC_x adc)
中断清除
- uint8_t HAL_ADC_IrqSamp2 (HAL_ADC_x adc)
中断处理函数

- uint8_t HAL_ADC_IrqSamp2Flg (HAL_ADC_x adc)
中断信号（和 HAL_ADC_IrqSamp2 的区别是不判断中断使能）
- void HAL_ADC_IrqSamp2Enable (HAL_ADC_x adc)
中断使能
- void HAL_ADC_IrqSamp2Disable (HAL_ADC_x adc)
中断禁止
- void HAL_ADC_IrqSamp2Clear (HAL_ADC_x adc)
中断清除
- uint8_t HAL_MCPWM_IrqOverflow (HAL_MCPWM_x mcpwmx)
中断处理函数
- uint8_t HAL_MCPWM_IrqOverflowFlg (HAL_MCPWM_x mcpwmx)
中断信号（和 HAL_MCPWM_IrqOverflow 的区别是不判断中断使能）
- void HAL_MCPWM_IrqOverflowEnable (HAL_MCPWM_x mcpwmx)
中断使能
- void HAL_MCPWM_IrqOverflowDisable (HAL_MCPWM_x mcpwmx)
中断禁止
- void HAL_MCPWM_IrqOverflowClear (HAL_MCPWM_x mcpwmx)
中断清除
- uint8_t HAL_MCPWM_IrqZeroCross (HAL_MCPWM_x mcpwmx)
中断处理函数
- uint8_t HAL_MCPWM_IrqZeroCrossFlg (HAL_MCPWM_x mcpwmx)
中断信号（和 HAL_MCPWM_IrqZeroCross 的区别是不判断中断使能）
- void HAL_MCPWM_IrqZeroCrossEnable (HAL_MCPWM_x mcpwmx)
中断使能
- void HAL_MCPWM_IrqZeroCrossDisable (HAL_MCPWM_x mcpwmx)
中断禁止
- void HAL_MCPWM_IrqZeroCrossClear (HAL_MCPWM_x mcpwmx)
中断清除
- uint8_t HAL_MCPWM_IrqFail0 (HAL_MCPWM_x mcpwmx)
中断处理函数
- uint8_t HAL_MCPWM_IrqFail0Flg (HAL_MCPWM_x mcpwmx)
中断信号（和 HAL_MCPWM_IrqFail0 的区别是不判断中断使能）
- void HAL_MCPWM_IrqFail0Enable (HAL_MCPWM_x mcpwmx)
中断使能
- void HAL_MCPWM_IrqFail0Disable (HAL_MCPWM_x mcpwmx)
中断禁止
- void HAL_MCPWM_IrqFail0Clear (HAL_MCPWM_x mcpwmx)
中断清除
- uint8_t HAL_MCPWM_IrqFail1 (HAL_MCPWM_x mcpwmx)
中断处理函数
- uint8_t HAL_MCPWM_IrqFail1Flg (HAL_MCPWM_x mcpwmx)
中断信号（和 HAL_MCPWM_IrqFail1 的区别是不判断中断使能）
- void HAL_MCPWM_IrqFail1Enable (HAL_MCPWM_x mcpwmx)
中断使能
- void HAL_MCPWM_IrqFail1Disable (HAL_MCPWM_x mcpwmx)
中断禁止
- void HAL_MCPWM_IrqFail1Clear (HAL_MCPWM_x mcpwmx)
中断清除
- uint8_t HAL_MCPWM_IrqFail (HAL_MCPWM_x mcpwmx)
中断处理函数
- uint8_t HAL_MCPWM_IrqFailFlg (HAL_MCPWM_x mcpwmx)

- 中断信号（和 *HAL_MCPWM_IrqFail* 的区别是不判断中断使能）
- void [HAL_MCPWM_IrqFailEnable](#) (HAL_MCPWM_x mcpwmx)
中断使能
- void [HAL_MCPWM_IrqFailDisable](#) (HAL_MCPWM_x mcpwmx)
中断禁止
- void [HAL_MCPWM_IrqFailClear](#) (HAL_MCPWM_x mcpwmx)
中断清除
- uint8_t [HAL_CMP_IrqCompare](#) (HAL_CMP_x cmp_x)
中断处理函数
- uint8_t [HAL_CMP_IrqCompareFlg](#) (HAL_CMP_x cmp_x)
中断信号（和 *HAL_CMP_IrqCompare* 的区别是不判断中断使能）
- void [HAL_CMP_IrqCompareEnable](#) (HAL_CMP_x cmp_x)
中断使能
- void [HAL_CMP_IrqCompareDisable](#) (HAL_CMP_x cmp_x)
中断禁止
- void [HAL_CMP_IrqCompareClear](#) (HAL_CMP_x cmp_x)
中断清除
- uint8_t [HAL_TIMER_IrqOverflow](#) (HAL_TIMER_x timer_x)
中断处理函数
- uint8_t [HAL_TIMER_IrqOverflowFlg](#) (HAL_TIMER_x timer_x)
中断信号（和 *HAL_TIMER_IrqOverflow* 的区别是不判断中断使能）
- void [HAL_TIMER_IrqOverflowEnable](#) (HAL_TIMER_x timer_x)
中断使能
- void [HAL_TIMER_IrqOverflowDisable](#) (HAL_TIMER_x timer_x)
中断禁止
- void [HAL_TIMER_IrqOverflowClear](#) (HAL_TIMER_x timer_x)
中断清除
- uint8_t [HAL_TIMER_IrqCompare0](#) (HAL_TIMER_x timer_x)
中断处理函数
- uint8_t [HAL_TIMER_IrqCompare0Flg](#) (HAL_TIMER_x timer_x)
中断信号（和 *HAL_TIMER_IrqCompare0* 的区别是不判断中断使能）
- void [HAL_TIMER_IrqCompare0Enable](#) (HAL_TIMER_x timer_x)
中断使能
- void [HAL_TIMER_IrqCompare0Disable](#) (HAL_TIMER_x timer_x)
中断禁止
- void [HAL_TIMER_IrqCompare0Clear](#) (HAL_TIMER_x timer_x)
中断清除
- uint8_t [HAL_TIMER_IrqCompare1](#) (HAL_TIMER_x timer_x)
中断处理函数
- uint8_t [HAL_TIMER_IrqCompare1Flg](#) (HAL_TIMER_x timer_x)
中断信号（和 *HAL_TIMER_IrqCompare1* 的区别是不判断中断使能）
- void [HAL_TIMER_IrqCompare1Enable](#) (HAL_TIMER_x timer_x)
中断使能
- void [HAL_TIMER_IrqCompare1Disable](#) (HAL_TIMER_x timer_x)
中断禁止
- void [HAL_TIMER_IrqCompare1Clear](#) (HAL_TIMER_x timer_x)
中断清除
- uint8_t [HAL_UART_IrqTxDone](#) (HAL_UART_x uart_x)
中断处理函数
- uint8_t [HAL_UART_IrqTxDoneFlg](#) (HAL_UART_x uart_x)
中断信号（和 *HAL_UART_IrqTxDone* 的区别是不判断中断使能）

- void `HAL_UART_IrqTxDoneEnable` (`HAL_UART_x` `uart_x`)
中断使能
- void `HAL_UART_IrqTxDoneDisable` (`HAL_UART_x` `uart_x`)
中断禁止
- void `HAL_UART_IrqTxDoneClear` (`HAL_UART_x` `uart_x`)
中断清除
- uint8_t `HAL_UART_IrqRxDone` (`HAL_UART_x` `uart_x`)
中断处理函数
- uint8_t `HAL_UART_IrqRxDoneFlg` (`HAL_UART_x` `uart_x`)
中断信号（和 `HAL_UART_IrqRxDone` 的区别是不判断中断使能）
- void `HAL_UART_IrqRxDoneEnable` (`HAL_UART_x` `uart_x`)
中断使能
- void `HAL_UART_IrqRxDoneDisable` (`HAL_UART_x` `uart_x`)
中断禁止
- void `HAL_UART_IrqRxDoneClear` (`HAL_UART_x` `uart_x`)
中断清除
- uint8_t `HAL_UART_IrqStopErr` (`HAL_UART_x` `uart_x`)
中断处理函数
- uint8_t `HAL_UART_IrqStopErrFlg` (`HAL_UART_x` `uart_x`)
中断信号（和 `HAL_UART_IrqStopErr` 的区别是不判断中断使能）
- void `HAL_UART_IrqStopErrEnable` (`HAL_UART_x` `uart_x`)
中断使能
- void `HAL_UART_IrqStopErrDisable` (`HAL_UART_x` `uart_x`)
中断禁止
- void `HAL_UART_IrqStopErrClear` (`HAL_UART_x` `uart_x`)
中断清除
- uint8_t `HAL_UART_IrqCheckErr` (`HAL_UART_x` `uart_x`)
中断处理函数
- uint8_t `HAL_UART_IrqCheckErrFlg` (`HAL_UART_x` `uart_x`)
中断信号（和 `HAL_UART_IrqCheckErr` 的区别是不判断中断使能）
- void `HAL_UART_IrqCheckErrEnable` (`HAL_UART_x` `uart_x`)
中断使能
- void `HAL_UART_IrqCheckErrDisable` (`HAL_UART_x` `uart_x`)
中断禁止
- void `HAL_UART_IrqCheckErrClear` (`HAL_UART_x` `uart_x`)
中断清除
- uint8_t `HAL_CAN_IrqRxDone` (`HAL_CAN_x` `can_x`)
中断处理函数
- uint8_t `HAL_CAN_IrqRxDoneFlg` (`HAL_CAN_x` `can_x`)
中断信号（和 `HAL_CAN_IrqRxDone` 的区别是不判断中断使能）
- void `HAL_CAN_IrqRxDoneEnable` (`HAL_CAN_x` `can_x`)
中断使能
- void `HAL_CAN_IrqRxDoneDisable` (`HAL_CAN_x` `can_x`)
中断禁止
- void `HAL_CAN_IrqRxDoneClear` (`HAL_CAN_x` `can_x`)
中断清除
- uint8_t `HAL_CAN_IrqTxDone` (`HAL_CAN_x` `can_x`)
中断处理函数
- uint8_t `HAL_CAN_IrqTxDoneFlg` (`HAL_CAN_x` `can_x`)
中断信号（和 `HAL_CAN_IrqTxDone` 的区别是不判断中断使能）
- void `HAL_CAN_IrqTxDoneEnable` (`HAL_CAN_x` `can_x`)

- 中断使能
- void [HAL_CAN_IrqTxDoneDisable](#) (HAL_CAN_x can_x)
- 中断禁止
- void [HAL_CAN_IrqTxDoneClear](#) (HAL_CAN_x can_x)
- 中断清除
- uint8_t [HAL_CAN_IrqAll](#) (HAL_CAN_x can_x)
- 中断处理函数
- uint8_t [HAL_CAN_IrqAllFlg](#) (HAL_CAN_x can_x)
- 中断信号（和 [HAL_CAN_IrqAll](#) 的区别是不判断中断使能）
- void [HAL_CAN_IrqAllEnable](#) (HAL_CAN_x can_x)
- 中断使能
- void [HAL_CAN_IrqAllDisable](#) (HAL_CAN_x can_x)
- 中断禁止
- void [HAL_CAN_IrqAllClear](#) (HAL_CAN_x can_x)
- 中断清除
- uint8_t [HAL_UART_IrqTxBufEmpty](#) (HAL_UART_x uart_x)
- 中断处理函数
- uint8_t [HAL_UART_IrqTxBufEmptyFlg](#) (HAL_UART_x uart_x)
- 中断信号（和 [HAL_UART_IrqTxBufEmpty](#) 的区别是不判断中断使能）
- void [HAL_UART_IrqTxBufEmptyEnable](#) (HAL_UART_x uart_x)
- 中断使能
- void [HAL_UART_IrqTxBufEmptyDisable](#) (HAL_UART_x uart_x)
- 中断禁止
- void [HAL_UART_IrqTxBufEmptyClear](#) (HAL_UART_x uart_x)
- 中断清除
- uint8_t [HAL_ADC_IrqSamp3](#) (HAL_ADC_x adc)
- 中断处理函数
- uint8_t [HAL_ADC_IrqSamp3Flg](#) (HAL_ADC_x adc)
- 中断信号（和 [HAL_ADC_IrqSamp3](#) 的区别是不判断中断使能）
- void [HAL_ADC_IrqSamp3Enable](#) (HAL_ADC_x adc)
- 中断使能
- void [HAL_ADC_IrqSamp3Disable](#) (HAL_ADC_x adc)
- 中断禁止
- void [HAL_ADC_IrqSamp3Clear](#) (HAL_ADC_x adc)
- 中断清除
- uint8_t [HAL_ADC_IrqSamp4](#) (HAL_ADC_x adc)
- 中断处理函数
- uint8_t [HAL_ADC_IrqSamp4Flg](#) (HAL_ADC_x adc)
- 中断信号（和 [HAL_ADC_IrqSamp4](#) 的区别是不判断中断使能）
- void [HAL_ADC_IrqSamp4Enable](#) (HAL_ADC_x adc)
- 中断使能
- void [HAL_ADC_IrqSamp4Disable](#) (HAL_ADC_x adc)
- 中断禁止
- void [HAL_ADC_IrqSamp4Clear](#) (HAL_ADC_x adc)
- 中断清除
- uint8_t [HAL_ADC_IrqWatchdog](#) (HAL_ADC_x adc)
- 中断处理函数
- uint8_t [HAL_ADC_IrqWatchdogFlg](#) (HAL_ADC_x adc)
- 中断信号（和 [HAL_ADC_IrqWatchdog](#) 的区别是不判断中断使能）
- void [HAL_ADC_IrqWatchdogEnable](#) (HAL_ADC_x adc)
- 中断使能

- void [HAL_ADC_IrqWatchdogDisable](#) (HAL_ADC_x adc)
中断禁止
- void [HAL_ADC_IrqWatchdogClear](#) (HAL_ADC_x adc)
中断清除
- uint8_t [HAL_ADC_IrqHardErr](#) (HAL_ADC_x adc)
中断处理函数
- uint8_t [HAL_ADC_IrqHardErrFlg](#) (HAL_ADC_x adc)
中断信号（和 [HAL_ADC_IrqHardErr](#) 的区别是不判断中断使能）
- void [HAL_ADC_IrqHardErrEnable](#) (HAL_ADC_x adc)
中断使能
- void [HAL_ADC_IrqHardErrDisable](#) (HAL_ADC_x adc)
中断禁止
- void [HAL_ADC_IrqHardErrClear](#) (HAL_ADC_x adc)
中断清除
- uint8_t [HAL_ADC_IrqSoftErr](#) (HAL_ADC_x adc)
中断处理函数
- uint8_t [HAL_ADC_IrqSoftErrFlg](#) (HAL_ADC_x adc)
中断信号（和 [HAL_ADC_IrqSoftErr](#) 的区别是不判断中断使能）
- void [HAL_ADC_IrqSoftErrEnable](#) (HAL_ADC_x adc)
中断使能
- void [HAL_ADC_IrqSoftErrDisable](#) (HAL_ADC_x adc)
中断禁止
- void [HAL_ADC_IrqSoftErrClear](#) (HAL_ADC_x adc)
中断清除
- uint8_t [HAL_MCPWM_IrqUpdate](#) (HAL_MCPWM_x mcpwmx)
中断处理函数
- uint8_t [HAL_MCPWM_IrqUpdateFlg](#) (HAL_MCPWM_x mcpwmx)
中断信号（和 [HAL_MCPWM_IrqUpdate](#) 的区别是不判断中断使能）
- void [HAL_MCPWM_IrqUpdateEnable](#) (HAL_MCPWM_x mcpwmx)
中断使能
- void [HAL_MCPWM_IrqUpdateDisable](#) (HAL_MCPWM_x mcpwmx)
中断禁止
- void [HAL_MCPWM_IrqUpdateClear](#) (HAL_MCPWM_x mcpwmx)
中断清除
- uint8_t [HAL_MCPWM_IrqTmr0](#) (HAL_MCPWM_x mcpwmx)
中断处理函数
- uint8_t [HAL_MCPWM_IrqTmr0Flg](#) (HAL_MCPWM_x mcpwmx)
中断信号（和 [HAL_MCPWM_IrqTmr0](#) 的区别是不判断中断使能）
- void [HAL_MCPWM_IrqTmr0Enable](#) (HAL_MCPWM_x mcpwmx)
中断使能
- void [HAL_MCPWM_IrqTmr0Disable](#) (HAL_MCPWM_x mcpwmx)
中断禁止
- void [HAL_MCPWM_IrqTmr0Clear](#) (HAL_MCPWM_x mcpwmx)
中断清除
- uint8_t [HAL_MCPWM_IrqTmr1](#) (HAL_MCPWM_x mcpwmx)
中断处理函数
- uint8_t [HAL_MCPWM_IrqTmr1Flg](#) (HAL_MCPWM_x mcpwmx)
中断信号（和 [HAL_MCPWM_IrqTmr1](#) 的区别是不判断中断使能）
- void [HAL_MCPWM_IrqTmr1Enable](#) (HAL_MCPWM_x mcpwmx)
中断使能
- void [HAL_MCPWM_IrqTmr1Disable](#) (HAL_MCPWM_x mcpwmx)

- 中断禁止
- void [HAL_MCPWM_IrqTmr1Clear](#) (HAL_MCPWM_x mcpwmx)
- 中断清除
- uint8_t [HAL_MCPWM_IrqTmr2](#) (HAL_MCPWM_x mcpwmx)
- 中断处理函数
- uint8_t [HAL_MCPWM_IrqTmr2Flg](#) (HAL_MCPWM_x mcpwmx)
- 中断信号（和 [HAL_MCPWM_IrqTmr2](#) 的区别是不判断中断使能）
- void [HAL_MCPWM_IrqTmr2Enable](#) (HAL_MCPWM_x mcpwmx)
- 中断使能
- void [HAL_MCPWM_IrqTmr2Disable](#) (HAL_MCPWM_x mcpwmx)
- 中断禁止
- void [HAL_MCPWM_IrqTmr2Clear](#) (HAL_MCPWM_x mcpwmx)
- 中断清除
- uint8_t [HAL_MCPWM_IrqTmr3](#) (HAL_MCPWM_x mcpwmx)
- 中断处理函数
- uint8_t [HAL_MCPWM_IrqTmr3Flg](#) (HAL_MCPWM_x mcpwmx)
- 中断信号（和 [HAL_MCPWM_IrqTmr3](#) 的区别是不判断中断使能）
- void [HAL_MCPWM_IrqTmr3Enable](#) (HAL_MCPWM_x mcpwmx)
- 中断使能
- void [HAL_MCPWM_IrqTmr3Disable](#) (HAL_MCPWM_x mcpwmx)
- 中断禁止
- void [HAL_MCPWM_IrqTmr3Clear](#) (HAL_MCPWM_x mcpwmx)
- 中断清除
- uint8_t [HAL_MCPWM_IrqTH00](#) (HAL_MCPWM_x mcpwmx)
- 中断处理函数
- uint8_t [HAL_MCPWM_IrqTH00Flg](#) (HAL_MCPWM_x mcpwmx)
- 中断信号（和 [HAL_MCPWM_IrqTH00](#) 的区别是不判断中断使能）
- void [HAL_MCPWM_IrqTH00Enable](#) (HAL_MCPWM_x mcpwmx)
- 中断使能
- void [HAL_MCPWM_IrqTH00Disable](#) (HAL_MCPWM_x mcpwmx)
- 中断禁止
- void [HAL_MCPWM_IrqTH00Clear](#) (HAL_MCPWM_x mcpwmx)
- 中断清除
- uint8_t [HAL_MCPWM_IrqTH01](#) (HAL_MCPWM_x mcpwmx)
- 中断处理函数
- uint8_t [HAL_MCPWM_IrqTH01Flg](#) (HAL_MCPWM_x mcpwmx)
- 中断信号（和 [HAL_MCPWM_IrqTH01](#) 的区别是不判断中断使能）
- void [HAL_MCPWM_IrqTH01Enable](#) (HAL_MCPWM_x mcpwmx)
- 中断使能
- void [HAL_MCPWM_IrqTH01Disable](#) (HAL_MCPWM_x mcpwmx)
- 中断禁止
- void [HAL_MCPWM_IrqTH01Clear](#) (HAL_MCPWM_x mcpwmx)
- 中断清除
- uint8_t [HAL_MCPWM_IrqTH10](#) (HAL_MCPWM_x mcpwmx)
- 中断处理函数
- uint8_t [HAL_MCPWM_IrqTH10Flg](#) (HAL_MCPWM_x mcpwmx)
- 中断信号（和 [HAL_MCPWM_IrqTH10](#) 的区别是不判断中断使能）
- void [HAL_MCPWM_IrqTH10Enable](#) (HAL_MCPWM_x mcpwmx)
- 中断使能
- void [HAL_MCPWM_IrqTH10Disable](#) (HAL_MCPWM_x mcpwmx)
- 中断禁止

- void `HAL_MCPWM_IrqTH10Clear` (`HAL_MCPWM_x` `mcpwmx`)
中断清除
- uint8_t `HAL_MCPWM_IrqTH11` (`HAL_MCPWM_x` `mcpwmx`)
中断处理函数
- uint8_t `HAL_MCPWM_IrqTH11Flg` (`HAL_MCPWM_x` `mcpwmx`)
中断信号（和 `HAL_MCPWM_IrqTH11` 的区别是不判断中断使能）
- void `HAL_MCPWM_IrqTH11Enable` (`HAL_MCPWM_x` `mcpwmx`)
中断使能
- void `HAL_MCPWM_IrqTH11Disable` (`HAL_MCPWM_x` `mcpwmx`)
中断禁止
- void `HAL_MCPWM_IrqTH11Clear` (`HAL_MCPWM_x` `mcpwmx`)
中断清除
- uint8_t `HAL_MCPWM_IrqTH20` (`HAL_MCPWM_x` `mcpwmx`)
中断处理函数
- uint8_t `HAL_MCPWM_IrqTH20Flg` (`HAL_MCPWM_x` `mcpwmx`)
中断信号（和 `HAL_MCPWM_IrqTH20` 的区别是不判断中断使能）
- void `HAL_MCPWM_IrqTH20Enable` (`HAL_MCPWM_x` `mcpwmx`)
中断使能
- void `HAL_MCPWM_IrqTH20Disable` (`HAL_MCPWM_x` `mcpwmx`)
中断禁止
- void `HAL_MCPWM_IrqTH20Clear` (`HAL_MCPWM_x` `mcpwmx`)
中断清除
- uint8_t `HAL_MCPWM_IrqTH21` (`HAL_MCPWM_x` `mcpwmx`)
中断处理函数
- uint8_t `HAL_MCPWM_IrqTH21Flg` (`HAL_MCPWM_x` `mcpwmx`)
中断信号（和 `HAL_MCPWM_IrqTH21` 的区别是不判断中断使能）
- void `HAL_MCPWM_IrqTH21Enable` (`HAL_MCPWM_x` `mcpwmx`)
中断使能
- void `HAL_MCPWM_IrqTH21Disable` (`HAL_MCPWM_x` `mcpwmx`)
中断禁止
- void `HAL_MCPWM_IrqTH21Clear` (`HAL_MCPWM_x` `mcpwmx`)
中断清除
- uint8_t `HAL_MCPWM_IrqTH30` (`HAL_MCPWM_x` `mcpwmx`)
中断处理函数
- uint8_t `HAL_MCPWM_IrqTH30Flg` (`HAL_MCPWM_x` `mcpwmx`)
中断信号（和 `HAL_MCPWM_IrqTH30` 的区别是不判断中断使能）
- void `HAL_MCPWM_IrqTH30Enable` (`HAL_MCPWM_x` `mcpwmx`)
中断使能
- void `HAL_MCPWM_IrqTH30Disable` (`HAL_MCPWM_x` `mcpwmx`)
中断禁止
- void `HAL_MCPWM_IrqTH30Clear` (`HAL_MCPWM_x` `mcpwmx`)
中断清除
- uint8_t `HAL_MCPWM_IrqTH31` (`HAL_MCPWM_x` `mcpwmx`)
中断处理函数
- uint8_t `HAL_MCPWM_IrqTH31Flg` (`HAL_MCPWM_x` `mcpwmx`)
中断信号（和 `HAL_MCPWM_IrqTH31` 的区别是不判断中断使能）
- void `HAL_MCPWM_IrqTH31Enable` (`HAL_MCPWM_x` `mcpwmx`)
中断使能
- void `HAL_MCPWM_IrqTH31Disable` (`HAL_MCPWM_x` `mcpwmx`)
中断禁止
- void `HAL_MCPWM_IrqTH31Clear` (`HAL_MCPWM_x` `mcpwmx`)

- 中断清除
- uint8_t HAL_MCPWM_IrqShort (HAL_MCPWM_x mcpwmx)
- 中断处理函数
- uint8_t HAL_MCPWM_IrqShortFlg (HAL_MCPWM_x mcpwmx)
- 中断信号 (和 HAL_MCPWM_IrqShort 的区别是不判断中断使能)
- void HAL_MCPWM_IrqShortEnable (HAL_MCPWM_x mcpwmx)
- 中断使能
- void HAL_MCPWM_IrqShortDisable (HAL_MCPWM_x mcpwmx)
- 中断禁止
- void HAL_MCPWM_IrqShortClear (HAL_MCPWM_x mcpwmx)
- 中断清除
- uint8_t HAL_I2C_IrqDone (HAL_I2C_x i2c_x)
- 中断处理函数
- uint8_t HAL_I2C_IrqDoneFlg (HAL_I2C_x i2c_x)
- 中断信号 (和 HAL_I2C_IrqDone 的区别是不判断中断使能)
- void HAL_I2C_IrqDoneEnable (HAL_I2C_x i2c_x)
- 中断使能
- void HAL_I2C_IrqDoneDisable (HAL_I2C_x i2c_x)
- 中断禁止
- void HAL_I2C_IrqDoneClear (HAL_I2C_x i2c_x)
- 中断清除
- uint8_t HAL_I2C_IrqBusErr (HAL_I2C_x i2c_x)
- 中断处理函数
- uint8_t HAL_I2C_IrqBusErrFlg (HAL_I2C_x i2c_x)
- 中断信号 (和 HAL_I2C_IrqBusErr 的区别是不判断中断使能)
- void HAL_I2C_IrqBusErrEnable (HAL_I2C_x i2c_x)
- 中断使能
- void HAL_I2C_IrqBusErrDisable (HAL_I2C_x i2c_x)
- 中断禁止
- void HAL_I2C_IrqBusErrClear (HAL_I2C_x i2c_x)
- 中断清除
- uint8_t HAL_I2C_IrqStop (HAL_I2C_x i2c_x)
- 中断处理函数
- uint8_t HAL_I2C_IrqStopFlg (HAL_I2C_x i2c_x)
- 中断信号 (和 HAL_I2C_IrqStop 的区别是不判断中断使能)
- void HAL_I2C_IrqStopEnable (HAL_I2C_x i2c_x)
- 中断使能
- void HAL_I2C_IrqStopDisable (HAL_I2C_x i2c_x)
- 中断禁止
- void HAL_I2C_IrqStopClear (HAL_I2C_x i2c_x)
- 中断清除
- uint8_t HAL_SPI_IrqDone (HAL_SPI_x spi_x)
- 中断处理函数
- uint8_t HAL_SPI_IrqDoneFlg (HAL_SPI_x spi_x)
- 中断信号 (和 HAL_SPI_IrqDone 的区别是不判断中断使能)
- void HAL_SPI_IrqDoneEnable (HAL_SPI_x spi_x)
- 中断使能
- void HAL_SPI_IrqDoneDisable (HAL_SPI_x spi_x)
- 中断禁止
- void HAL_SPI_IrqDoneClear (HAL_SPI_x spi_x)
- 中断清除

- `uint8_t HAL_SPI_IrqErr (HAL_SPI_x spi_x)`
中断处理函数
- `uint8_t HAL_SPI_IrqErrFlg (HAL_SPI_x spi_x)`
中断信号（和 `HAL_SPI_IrqErr` 的区别是不判断中断使能）
- `void HAL_SPI_IrqErrEnable (HAL_SPI_x spi_x)`
中断使能
- `void HAL_SPI_IrqErrDisable (HAL_SPI_x spi_x)`
中断禁止
- `void HAL_SPI_IrqErrClear (HAL_SPI_x spi_x)`
中断清除
- `uint8_t HAL_SPI_IrqOverflow (HAL_SPI_x spi_x)`
中断处理函数
- `uint8_t HAL_SPI_IrqOverflowFlg (HAL_SPI_x spi_x)`
中断信号（和 `HAL_SPI_IrqOverflow` 的区别是不判断中断使能）
- `void HAL_SPI_IrqOverflowEnable (HAL_SPI_x spi_x)`
中断使能
- `void HAL_SPI_IrqOverflowDisable (HAL_SPI_x spi_x)`
中断禁止
- `void HAL_SPI_IrqOverflowClear (HAL_SPI_x spi_x)`
中断清除
- `uint8_t HAL_DMA_IrqDone (HAL_DMA_x dma_x)`
中断处理函数
- `uint8_t HAL_DMA_IrqDoneFlg (HAL_DMA_x dma_x)`
中断信号（同 `HAL_DMA_IrqDone`）
- `void HAL_DMA_IrqDoneEnable (HAL_DMA_x dma_x)`
中断使能（本模块无独立使能位，由NVIC控制）
- `void HAL_DMA_IrqDoneDisable (HAL_DMA_x dma_x)`
中断禁止（本模块无独立使能位，由NVIC控制）
- `void HAL_DMA_IrqDoneClear (HAL_DMA_x dma_x)`
中断清除
- `uint8_t HAL_DMA_IrqErr (HAL_DMA_x dma_x)`
中断处理函数
- `uint8_t HAL_DMA_IrqErrFlg (HAL_DMA_x dma_x)`
中断信号（同 `HAL_DMA_IrqErr`）
- `void HAL_DMA_IrqErrEnable (HAL_DMA_x dma_x)`
中断使能（本模块无独立使能位，由NVIC控制）
- `void HAL_DMA_IrqErrDisable (HAL_DMA_x dma_x)`
中断禁止（本模块无独立使能位，由NVIC控制）
- `void HAL_DMA_IrqErrClear (HAL_DMA_x dma_x)`
中断清除
- `uint8_t HAL_GPIO_Irq (uint8_t line)`
中断处理函数
- `uint8_t HAL_GPIO_IrqFlg (uint8_t line)`
中断信号（同 `HAL_GPIO_Irq`）
- `void HAL_GPIO_IrqEnable (uint8_t line)`
中断使能（本模块无独立使能位，由NVIC控制）
- `void HAL_GPIO_IrqDisable (uint8_t line)`
中断禁止（本模块无独立使能位，由NVIC控制）
- `void HAL_GPIO_IrqClear (uint8_t line)`
中断清除
- `uint8_t HAL_ADC_IrqIdleSamp (HAL_ADC_x adc)`

- uint8_t HAL_ADC_IrqlIdleSampFlg (HAL_ADC_x adc)
- void HAL_ADC_IrqlIdleSampEnable (HAL_ADC_x adc)
- void HAL_ADC_IrqlIdleSampDisable (HAL_ADC_x adc)
- void HAL_ADC_IrqlIdleSampClear (HAL_ADC_x adc)
- uint8_t HAL_ADC_IrqWatchdog1 (HAL_ADC_x adc)
- uint8_t HAL_ADC_IrqWatchdog1Flg (HAL_ADC_x adc)
- void HAL_ADC_IrqWatchdog1Enable (HAL_ADC_x adc)
- void HAL_ADC_IrqWatchdog1Disable (HAL_ADC_x adc)
- void HAL_ADC_IrqWatchdog1Clear (HAL_ADC_x adc)
- uint8_t HAL_MCPWM_IrqTH40 (HAL_MCPWM_x mcpwmx)
- uint8_t HAL_MCPWM_IrqTH40Flg (HAL_MCPWM_x mcpwmx)
- void HAL_MCPWM_IrqTH40Enable (HAL_MCPWM_x mcpwmx)
- void HAL_MCPWM_IrqTH40Disable (HAL_MCPWM_x mcpwmx)
- void HAL_MCPWM_IrqTH40Clear (HAL_MCPWM_x mcpwmx)
- uint8_t HAL_MCPWM_IrqTH41 (HAL_MCPWM_x mcpwmx)
- uint8_t HAL_MCPWM_IrqTH41Flg (HAL_MCPWM_x mcpwmx)
- void HAL_MCPWM_IrqTH41Enable (HAL_MCPWM_x mcpwmx)
- void HAL_MCPWM_IrqTH41Disable (HAL_MCPWM_x mcpwmx)
- void HAL_MCPWM_IrqTH41Clear (HAL_MCPWM_x mcpwmx)
- uint8_t HAL_MCPWM_IrqTH50 (HAL_MCPWM_x mcpwmx)
- uint8_t HAL_MCPWM_IrqTH50Flg (HAL_MCPWM_x mcpwmx)
- void HAL_MCPWM_IrqTH50Enable (HAL_MCPWM_x mcpwmx)
- void HAL_MCPWM_IrqTH50Disable (HAL_MCPWM_x mcpwmx)
- void HAL_MCPWM_IrqTH50Clear (HAL_MCPWM_x mcpwmx)
- uint8_t HAL_MCPWM_IrqTH51 (HAL_MCPWM_x mcpwmx)
- uint8_t HAL_MCPWM_IrqTH51Flg (HAL_MCPWM_x mcpwmx)
- void HAL_MCPWM_IrqTH51Enable (HAL_MCPWM_x mcpwmx)
- void HAL_MCPWM_IrqTH51Disable (HAL_MCPWM_x mcpwmx)
- void HAL_MCPWM_IrqTH51Clear (HAL_MCPWM_x mcpwmx)
- uint8_t HAL_SW_Irq (uint8_t x)
- uint8_t HAL_SW_IrqFlg (uint8_t x)
- void HAL_SW_IrqEnable (uint8_t x)
- void HAL_SW_IrqDisable (uint8_t x)
- void HAL_SW_IrqClear (uint8_t x)
- void HAL_SW_IrqTrig (uint8_t x)
- uint8_t HAL_UART_IrqAbd (HAL_UART_x uart_x)
- uint8_t HAL_UART_IrqAbdFlg (HAL_UART_x uart_x)
- void HAL_UART_IrqAbdEnable (HAL_UART_x uart_x)
- void HAL_UART_IrqAbdDisable (HAL_UART_x uart_x)
- void HAL_UART_IrqAbdClear (HAL_UART_x uart_x)
- uint8_t HAL_UART_IrqlIdle (HAL_UART_x uart_x)
- uint8_t HAL_UART_IrqlIdleFlg (HAL_UART_x uart_x)
- void HAL_UART_IrqlIdleEnable (HAL_UART_x uart_x)
- void HAL_UART_IrqlIdleDisable (HAL_UART_x uart_x)
- void HAL_UART_IrqlIdleClear (HAL_UART_x uart_x)
- uint8_t HAL_UART_IrqLbd (HAL_UART_x uart_x)
- uint8_t HAL_UART_IrqLbdFlg (HAL_UART_x uart_x)
- void HAL_UART_IrqLbdEnable (HAL_UART_x uart_x)
- void HAL_UART_IrqLbdDisable (HAL_UART_x uart_x)
- void HAL_UART_IrqLbdClear (HAL_UART_x uart_x)
- uint8_t HAL_UART_IrqRxOv (HAL_UART_x uart_x)
- uint8_t HAL_UART_IrqRxOvFlg (HAL_UART_x uart_x)
- void HAL_UART_IrqRxOvEnable (HAL_UART_x uart_x)
- void HAL_UART_IrqRxOvDisable (HAL_UART_x uart_x)
- void HAL_UART_IrqRxOvClear (HAL_UART_x uart_x)

- uint8_t HAL_UART_IrqTxOv (HAL_UART_x uart_x)
- uint8_t HAL_UART_IrqTxOvFlg (HAL_UART_x uart_x)
- void HAL_UART_IrqTxOvEnable (HAL_UART_x uart_x)
- void HAL_UART_IrqTxOvDisable (HAL_UART_x uart_x)
- void HAL_UART_IrqTxOvClear (HAL_UART_x uart_x)

4.92.1 函数说明

4.92.1.1 HAL_ADC_IrqHardErr()

```
uint8_t HAL_ADC_IrqHardErr (
    HAL_ADC_x adc )
```

中断处理函数

参数

<i>adc</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1786 行定义.

4.92.1.2 HAL_ADC_IrqHardErrClear()

```
void HAL_ADC_IrqHardErrClear (
    HAL_ADC_x adc )
```

中断清除

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1847 行定义.

4.92.1.3 HAL_ADC_IrqHardErrDisable()

```
void HAL_ADC_IrqHardErrDisable (
    HAL_ADC_x adc )
```

中断禁止

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1836 行定义.

4.92.1.4 HAL_ADC_IrqHardErrEnable()

```
void HAL_ADC_IrqHardErrEnable (
    HAL_ADC_x adc )
```

中断使能

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1825 行定义.

4.92.1.5 HAL_ADC_IrqHardErrFlg()

```
uint8_t HAL_ADC_IrqHardErrFlg (
    HAL_ADC_x adc )
```

中断信号（和 HAL_ADC_IrqHardErr 的区别是不判断中断使能）

参数

<i>adc</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1807 行定义.

4.92.1.6 HAL_ADC_IrqIdleSamp()

```
uint8_t HAL_ADC_IrqIdleSamp (
    HAL_ADC_x adc )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3655 行定义.

4.92.1.7 HAL_ADC_IrqIdleSampClear()

```
void HAL_ADC_IrqIdleSampClear (
    HAL_ADC_x adc )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3673 行定义.

4.92.1.8 HAL_ADC_IrqIdleSampDisable()

```
void HAL_ADC_IrqIdleSampDisable (
    HAL_ADC_x adc )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3669 行定义.

4.92.1.9 HAL_ADC_IrqIdleSampEnable()

```
void HAL_ADC_IrqIdleSampEnable (
    HAL_ADC_x adc )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3665 行定义.

4.92.1.10 HAL_ADC_IrqIdleSampFlg()

```
uint8_t HAL_ADC_IrqIdleSampFlg (
    HAL_ADC_x adc )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3660 行定义.

4.92.1.11 HAL_ADC_IrqSamp1()

```
uint8_t HAL_ADC_IrqSamp1 (
    HAL_ADC_x adc )
```

中断处理函数

参数

<i>adc</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 160 行定义.

4.92.1.12 HAL_ADC_IrqSamp1Clear()

```
void HAL_ADC_IrqSamp1Clear (
    HAL_ADC_x adc )
```

中断清除

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 221 行定义.

4.92.1.13 HAL_ADC_IrqSamp1Disable()

```
void HAL_ADC_IrqSamp1Disable (
    HAL_ADC_x adc )
```

中断禁止

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 210 行定义.

4.92.1.14 HAL_ADC_IrqSamp1Enable()

```
void HAL_ADC_IrqSamp1Enable (
    HAL_ADC_x adc )
```

中断使能

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 199 行定义.

4.92.1.15 HAL_ADC_IrqSamp1Flg()

```
uint8_t HAL_ADC_IrqSamp1Flg (
    HAL_ADC_x adc )
```

中断信号（和 HAL_ADC_IrqSamp1 的区别是不判断中断使能）

参数

<i>adc</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 181 行定义.

4.92.1.16 HAL_ADC_IrqSamp2()

```
uint8_t HAL_ADC_IrqSamp2 (
    HAL_ADC_x adc )
```

中断处理函数

参数

<i>adc</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 233 行定义.

4.92.1.17 HAL_ADC_IrqSamp2Clear()

```
void HAL_ADC_IrqSamp2Clear (
    HAL_ADC_x adc )
```

中断清除

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 294 行定义.

4.92.1.18 HAL_ADC_IrqSamp2Disable()

```
void HAL_ADC_IrqSamp2Disable (
    HAL_ADC_x adc )
```

中断禁止

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 283 行定义.

4.92.1.19 HAL_ADC_IrqSamp2Enable()

```
void HAL_ADC_IrqSamp2Enable (
    HAL_ADC_x adc )
```

中断使能

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 272 行定义.

4.92.1.20 HAL_ADC_IrqSamp2Flg()

```
uint8_t HAL_ADC_IrqSamp2Flg (
    HAL_ADC_x adc )
```

中断信号（和 HAL_ADC_IrqSamp2 的区别是不判断中断使能）

参数

<i>adc</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 254 行定义.

4.92.1.21 HAL_ADC_IrqSamp3()

```
uint8_t HAL_ADC_IrqSamp3 (
    HAL_ADC_x adc )
```

中断处理函数

参数

<i>adc</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1567 行定义.

4.92.1.22 HAL_ADC_IrqSamp3Clear()

```
void HAL_ADC_IrqSamp3Clear (
    HAL_ADC_x adc )
```

中断清除

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1628 行定义.

4.92.1.23 HAL_ADC_IrqSamp3Disable()

```
void HAL_ADC_IrqSamp3Disable (
    HAL_ADC_x adc )
```

中断禁止

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1617 行定义.

4.92.1.24 HAL_ADC_IrqSamp3Enable()

```
void HAL_ADC_IrqSamp3Enable (
    HAL_ADC_x adc )
```

中断使能

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1606 行定义.

4.92.1.25 HAL_ADC_IrqSamp3Flg()

```
uint8_t HAL_ADC_IrqSamp3Flg (
    HAL_ADC_x adc )
```

中断信号（和 HAL_ADC_IrqSamp3 的区别是不判断中断使能）

参数

<i>adc</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1588 行定义.

4.92.1.26 HAL_ADC_IrqSamp4()

```
uint8_t HAL_ADC_IrqSamp4 (
    HAL_ADC_x adc )
```

中断处理函数

参数

<i>adc</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1640 行定义.

4.92.1.27 HAL_ADC_IrqSamp4Clear()

```
void HAL_ADC_IrqSamp4Clear (
    HAL_ADC_x adc )
```

中断清除

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1701 行定义.

4.92.1.28 HAL_ADC_IrqSamp4Disable()

```
void HAL_ADC_IrqSamp4Disable (
    HAL_ADC_x adc )
```

中断禁止

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1690 行定义.

4.92.1.29 HAL_ADC_IrqSamp4Enable()

```
void HAL_ADC_IrqSamp4Enable (  
    HAL_ADC_x adc )
```

中断使能

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1679 行定义.

4.92.1.30 HAL_ADC_IrqSamp4Flg()

```
uint8_t HAL_ADC_IrqSamp4Flg (  
    HAL_ADC_x adc )
```

中断信号（和 HAL_ADC_IrqSamp4 的区别是不判断中断使能）

参数

<i>adc</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1661 行定义.

4.92.1.31 HAL_ADC_IrqSoftErr()

```
uint8_t HAL_ADC_IrqSoftErr (  
    HAL_ADC_x adc )
```

中断处理函数

参数

<i>adc</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1859 行定义.

4.92.1.32 HAL_ADC_IrqSoftErrClear()

```
void HAL_ADC_IrqSoftErrClear (
    HAL_ADC_x adc )
```

中断清除

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1920 行定义.

4.92.1.33 HAL_ADC_IrqSoftErrDisable()

```
void HAL_ADC_IrqSoftErrDisable (
    HAL_ADC_x adc )
```

中断禁止

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1909 行定义.

4.92.1.34 HAL_ADC_IrqSoftErrEnable()

```
void HAL_ADC_IrqSoftErrEnable (
    HAL_ADC_x adc )
```

中断使能

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1898 行定义.

4.92.1.35 HAL_ADC_IrqSoftErrFlg()

```
uint8_t HAL_ADC_IrqSoftErrFlg (
    HAL_ADC_x adc )
```

中断信号（和 HAL_ADC_IrqSoftErr 的区别是不判断中断使能）

参数

<i>adc</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1880 行定义.

4.92.1.36 HAL_ADC_IrqWatchdog()

```
uint8_t HAL_ADC_IrqWatchdog (
    HAL_ADC_x adc )
```

中断处理函数

参数

<i>adc</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1713 行定义.

4.92.1.37 HAL_ADC_IrqWatchdog1()

```
uint8_t HAL_ADC_IrqWatchdog1 (
    HAL_ADC_x adc )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3678 行定义.

4.92.1.38 HAL_ADC_IrqWatchdog1Clear()

```
void HAL_ADC_IrqWatchdog1Clear (
    HAL_ADC_x adc )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3696 行定义.

4.92.1.39 HAL_ADC_IrqWatchdog1Disable()

```
void HAL_ADC_IrqWatchdog1Disable (
    HAL_ADC_x adc )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3692 行定义.

4.92.1.40 HAL_ADC_IrqWatchdog1Enable()

```
void HAL_ADC_IrqWatchdog1Enable (
    HAL_ADC_x adc )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3688 行定义.

4.92.1.41 HAL_ADC_IrqWatchdog1Flg()

```
uint8_t HAL_ADC_IrqWatchdog1Flg (
    HAL_ADC_x adc )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3683 行定义.

4.92.1.42 HAL_ADC_IrqWatchdogClear()

```
void HAL_ADC_IrqWatchdogClear (
    HAL_ADC_x adc )
```

中断清除

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1774 行定义.

4.92.1.43 HAL_ADC_IrqWatchdogDisable()

```
void HAL_ADC_IrqWatchdogDisable (
    HAL_ADC_x adc )
```

中断禁止

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1763 行定义.

4.92.1.44 HAL_ADC_IrqWatchdogEnable()

```
void HAL_ADC_IrqWatchdogEnable (
    HAL_ADC_x adc )
```

中断使能

参数

<i>adc</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1752 行定义.

4.92.1.45 HAL_ADC_IrqWatchdogFlg()

```
uint8_t HAL_ADC_IrqWatchdogFlg (
    HAL_ADC_x adc )
```

中断信号 (和 HAL_ADC_IrqWatchdog 的区别是不判断中断使能)

参数

<i>adc</i>	模块编号

返回

1表示中断发生, 0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1734 行定义.

4.92.1.46 HAL_CAN_IrqAll()

```
uint8_t HAL_CAN_IrqAll (
    HAL_CAN_x can_x )
```

中断处理函数

参数

<i>can_x</i>	模块编号

返回

1表示中断发生, 0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1421 行定义.

4.92.1.47 HAL_CAN_IrqAllClear()

```
void HAL_CAN_IrqAllClear (
    HAL_CAN_x can_x )
```

中断清除

参数

can_x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1482 行定义.

4.92.1.48 HAL_CAN_IrqAllDisable()

```
void HAL_CAN_IrqAllDisable (
    HAL_CAN_x can_x )
```

中断禁止

参数

can_x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1471 行定义.

4.92.1.49 HAL_CAN_IrqAllEnable()

```
void HAL_CAN_IrqAllEnable (
    HAL_CAN_x can_x )
```

中断使能

参数

can_x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1460 行定义.

4.92.1.50 HAL_CAN_IrqAllFlg()

```
uint8_t HAL_CAN_IrqAllFlg (
    HAL_CAN_x can_x )
```

中断信号（和 HAL_CAN_IrqAll 的区别是不判断中断使能）

参数

can_x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1442 行定义.

4.92.1.51 HAL_CAN_IrqRxDone()

```
uint8_t HAL_CAN_IrqRxDone (
    HAL_CAN_x can_x )
```

中断处理函数

参数

can_x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1275 行定义.

4.92.1.52 HAL_CAN_IrqRxDoneClear()

```
void HAL_CAN_IrqRxDoneClear (
    HAL_CAN_x can_x )
```

中断清除

参数

can_x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1336 行定义.

4.92.1.53 HAL_CAN_IrqRxDoneDisable()

```
void HAL_CAN_IrqRxDoneDisable (
    HAL_CAN_x can_x )
```

中断禁止

参数

<i>can</i> _{<i>x</i>}	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1325 行定义.

4.92.1.54 HAL_CAN_IrqRxDoneEnable()

```
void HAL_CAN_IrqRxDoneEnable (
    HAL_CAN_x can_x )
```

中断使能

参数

<i>can</i> _{<i>x</i>}	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1314 行定义.

4.92.1.55 HAL_CAN_IrqRxDoneFlg()

```
uint8_t HAL_CAN_IrqRxDoneFlg (
    HAL_CAN_x can_x )
```

中断信号（和 HAL_CAN_IrqRxDone 的区别是不判断中断使能）

参数

<i>can</i> _{<i>x</i>}	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1296 行定义.

4.92.1.56 HAL_CAN_IrqTxDone()

```
uint8_t HAL_CAN_IrqTxDone (
    HAL_CAN_x can_x )
```

中断处理函数

参数

can_x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1348 行定义.

4.92.1.57 HAL_CAN_IrqTxDoneClear()

```
void HAL_CAN_IrqTxDoneClear (
    HAL_CAN_x can_x )
```

中断清除

参数

can_x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1409 行定义.

4.92.1.58 HAL_CAN_IrqTxDoneDisable()

```
void HAL_CAN_IrqTxDoneDisable (
    HAL_CAN_x can_x )
```

中断禁止

参数

can_x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1398 行定义.

4.92.1.59 HAL_CAN_IrqTxDoneEnable()

```
void HAL_CAN_IrqTxDoneEnable (
    HAL_CAN_x can_x )
```

中断使能

参数

can_x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1387 行定义.

4.92.1.60 HAL_CAN_IrqTxDoneFlg()

```
uint8_t HAL_CAN_IrqTxDoneFlg (
    HAL_CAN_x can_x )
```

中断信号（和 HAL_CAN_IrqTxDone 的区别是不判断中断使能）

参数

can_x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1369 行定义.

4.92.1.61 HAL_CMP_IrqCompare()

```
uint8_t HAL_CMP_IrqCompare (
    HAL_CMP_x cmp_x )
```

中断处理函数

参数

cmp_x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 691 行定义.

4.92.1.62 HAL_CMP_IrqCompareClear()

```
void HAL_CMP_IrqCompareClear (
    HAL_CMP_x cmp_x )
```

中断清除

参数

<i>cmp</i> ↔ _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 752 行定义.

4.92.1.63 HAL_CMP_IrqCompareDisable()

```
void HAL_CMP_IrqCompareDisable (
    HAL_CMP_x cmp_x )
```

中断禁止

参数

<i>cmp</i> ↔ _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 741 行定义.

4.92.1.64 HAL_CMP_IrqCompareEnable()

```
void HAL_CMP_IrqCompareEnable (
    HAL_CMP_x cmp_x )
```

中断使能

参数

<i>cmp</i> ↔ _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 730 行定义.

4.92.1.65 HAL_CMP_IrqCompareFlg()

```
uint8_t HAL_CMP_IrqCompareFlg (
    HAL_CMP_x cmp_x )
```

中断信号（和 HAL_CMP_IrqCompare 的区别是不判断中断使能）

参数

<i>cmp</i> ↔ _x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 712 行定义.

4.92.1.66 HAL_DMA_IrqDone()

```
uint8_t HAL_DMA_IrqDone (
    HAL_DMA_x dma_x )
```

中断处理函数

参数

<i>dma</i> ↔ _x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3448 行定义.

4.92.1.67 HAL_DMA_IrqDoneClear()

```
void HAL_DMA_IrqDoneClear (
    HAL_DMA_x dma_x )
```

中断清除

参数

<i>dma</i> ↔ _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3505 行定义.

4.92.1.68 HAL_DMA_IrqDoneDisable()

```
void HAL_DMA_IrqDoneDisable (
    HAL_DMA_x dma_x )
```

中断禁止（本模块无独立使能位，由NVIC控制）

参数

<i>dma</i> ↔ _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3496 行定义.

4.92.1.69 HAL_DMA_IrqDoneEnable()

```
void HAL_DMA_IrqDoneEnable (
    HAL_DMA_x dma_x )
```

中断使能（本模块无独立使能位，由NVIC控制）

参数

<i>dma</i> ↔ _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3487 行定义.

4.92.1.70 HAL_DMA_IrqDoneFlg()

```
uint8_t HAL_DMA_IrqDoneFlg (
    HAL_DMA_x dma_x )
```

中断信号（同 HAL_DMA_IrqDone）

参数

<i>dma</i> ↔ _x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3468 行定义.

4.92.1.71 HAL_DMA_IrqErr()

```
uint8_t HAL_DMA_IrqErr (
    HAL_DMA_x dma_x )
```

中断处理函数

参数

dma_{x}	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3518 行定义.

4.92.1.72 HAL_DMA_IrqErrClear()

```
void HAL_DMA_IrqErrClear (
    HAL_DMA_x dma_x )
```

中断清除

参数

dma_{x}	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3575 行定义.

4.92.1.73 HAL_DMA_IrqErrDisable()

```
void HAL_DMA_IrqErrDisable (
    HAL_DMA_x dma_x )
```

中断禁止（本模块无独立使能位，由NVIC控制）

参数

dma_{x}	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3566 行定义.

4.92.1.74 HAL_DMA_IrqErrEnable()

```
void HAL_DMA_IrqErrEnable (
    HAL_DMA_x dma_x )
```

中断使能（本模块无独立使能位，由NVIC控制）

参数

<i>dma</i> ↔ _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3557 行定义.

4.92.1.75 HAL_DMA_IrqErrFlg()

```
uint8_t HAL_DMA_IrqErrFlg (
    HAL_DMA_x dma_x )
```

中断信号 (同 HAL_DMA_IrqErr)

参数

<i>dma</i> ↔ _x	模块编号

返回

1表示中断发生, 0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3538 行定义.

4.92.1.76 HAL_GPIO_Irq()

```
uint8_t HAL_GPIO_Irq (
    uint8_t line )
```

中断处理函数

参数

<i>line</i>	模块编号

返回

1表示中断发生, 0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3588 行定义.

4.92.1.77 HAL_GPIO_IrqClear()

```
void HAL_GPIO_IrqClear (
    uint8_t line )
```

中断清除

参数

<i>line</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3645 行定义.

4.92.1.78 HAL_GPIO_IrqDisable()

```
void HAL_GPIO_IrqDisable (
    uint8_t line )
```

中断禁止（本模块无独立使能位，由NVIC控制）

参数

<i>line</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3636 行定义.

4.92.1.79 HAL_GPIO_IrqEnable()

```
void HAL_GPIO_IrqEnable (
    uint8_t line )
```

中断使能（本模块无独立使能位，由NVIC控制）

参数

<i>line</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3627 行定义.

4.92.1.80 HAL_GPIO_IrqFlg()

```
uint8_t HAL_GPIO_IrqFlg (
    uint8_t line )
```

中断信号（同 HAL_GPIO_Irq）

参数

<i>line</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3608 行定义.

4.92.1.81 HAL_HALL_IrqChange()

```
uint8_t HAL_HALL_IrqChange (
    HAL_HALL_x hall )
```

中断处理函数

参数

<i>hall</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 14 行定义.

4.92.1.82 HAL_HALL_IrqChangeClear()

```
void HAL_HALL_IrqChangeClear (
    HAL_HALL_x hall )
```

中断清除

参数

<i>hall</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 75 行定义.

4.92.1.83 HAL_HALL_IrqChangeDisable()

```
void HAL_HALL_IrqChangeDisable (
    HAL_HALL_x hall )
```

中断禁止

参数

<i>hall</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 64 行定义.

4.92.1.84 HAL_HALL_IrqChangeEnable()

```
void HAL_HALL_IrqChangeEnable (
    HAL_HALL_x hall )
```

中断使能

参数

<i>hall</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 53 行定义.

4.92.1.85 HAL_HALL_IrqChangeFlg()

```
uint8_t HAL_HALL_IrqChangeFlg (
    HAL_HALL_x hall )
```

中断信号（和 HAL_HALL_IrqChange 的区别是不判断中断使能）

参数

<i>hall</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 35 行定义.

4.92.1.86 HAL_HALL_IrqOver()

```
uint8_t HAL_HALL_IrqOver (
    HAL_HALL_x hall )
```

中断处理函数

参数

<i>hall</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 87 行定义.

4.92.1.87 HAL_HALL_IrqOverClear()

```
void HAL_HALL_IrqOverClear (
    HAL_HALL_x hall )
```

中断清除

参数

<i>hall</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 148 行定义.

4.92.1.88 HAL_HALL_IrqOverDisable()

```
void HAL_HALL_IrqOverDisable (
    HAL_HALL_x hall )
```

中断禁止

参数

<i>hall</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 137 行定义.

4.92.1.89 HAL_HALL_IrqOverEnable()

```
void HAL_HALL_IrqOverEnable (
    HAL_HALL_x hall )
```

中断使能

参数

<i>hall</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 126 行定义.

4.92.1.90 HAL_HALL_IrqOverFlg()

```
uint8_t HAL_HALL_IrqOverFlg (
    HAL_HALL_x hall )
```

中断信号（和 HAL_HALL_IrqOver 的区别是不判断中断使能）

参数

<i>hall</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 108 行定义.

4.92.1.91 HAL_I2C_IrqBusErr()

```
uint8_t HAL_I2C_IrqBusErr (
    HAL_I2C_x i2c_x )
```

中断处理函数

参数

<i>i2c_x</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3083 行定义.

4.92.1.92 HAL_I2C_IrqBusErrClear()

```
void HAL_I2C_IrqBusErrClear (
    HAL_I2C_x i2c_x )
```

中断清除

参数

<i>i2c_x</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3144 行定义.

4.92.1.93 HAL_I2C_IrqBusErrDisable()

```
void HAL_I2C_IrqBusErrDisable (
    HAL_I2C_x i2c_x )
```

中断禁止

参数

$i2c\_x$	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3133 行定义.

4.92.1.94 HAL_I2C_IrqBusErrEnable()

```
void HAL_I2C_IrqBusErrEnable (
    HAL_I2C_x i2c_x )
```

中断使能

参数

$i2c\_x$	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3122 行定义.

4.92.1.95 HAL_I2C_IrqBusErrFlg()

```
uint8_t HAL_I2C_IrqBusErrFlg (
    HAL_I2C_x i2c_x )
```

中断信号（和 HAL_I2C_IrqBusErr 的区别是不判断中断使能）

参数

$i2c\_x$	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3104 行定义.

4.92.1.96 HAL_I2C_IrqDone()

```
uint8_t HAL_I2C_IrqDone (
    HAL_I2C_x i2c_x )
```

中断处理函数

参数

$i2c\_x$	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3010 行定义.

4.92.1.97 HAL_I2C_IrqDoneClear()

```
void HAL_I2C_IrqDoneClear (
    HAL_I2C_x i2c_x )
```

中断清除

参数

$i2c\_x$	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3071 行定义.

4.92.1.98 HAL_I2C_IrqDoneDisable()

```
void HAL_I2C_IrqDoneDisable (
    HAL_I2C_x i2c_x )
```

中断禁止

参数

$i2c\_x$	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3060 行定义.

4.92.1.99 HAL_I2C_IrqDoneEnable()

```
void HAL_I2C_IrqDoneEnable (
    HAL_I2C_x i2c_x )
```

中断使能

参数

<i>i2c</i> ↔ _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3049 行定义.

4.92.1.100 HAL_I2C_IrqDoneFlg()

```
uint8_t HAL_I2C_IrqDoneFlg (
    HAL_I2C_x i2c_x )
```

中断信号（和 HAL_I2C_IrqDone 的区别是不判断中断使能）

参数

<i>i2c</i> ↔ _x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3031 行定义.

4.92.1.101 HAL_I2C_IrqStop()

```
uint8_t HAL_I2C_IrqStop (
    HAL_I2C_x i2c_x )
```

中断处理函数

参数

<i>i2c</i> ↔ _x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3156 行定义.

4.92.1.102 HAL_I2C_IrqStopClear()

```
void HAL_I2C_IrqStopClear (
    HAL_I2C_x i2c_x )
```

中断清除

参数

$i2c\_x$	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3217 行定义.

4.92.1.103 HAL_I2C_IrqStopDisable()

```
void HAL_I2C_IrqStopDisable (
    HAL_I2C_x i2c_x )
```

中断禁止

参数

$i2c\_x$	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3206 行定义.

4.92.1.104 HAL_I2C_IrqStopEnable()

```
void HAL_I2C_IrqStopEnable (
    HAL_I2C_x i2c_x )
```

中断使能

参数

$i2c\_x$	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3195 行定义.

4.92.1.105 HAL_I2C_IrqStopFlg()

```
uint8_t HAL_I2C_IrqStopFlg (
    HAL_I2C_x i2c_x )
```

中断信号（和 HAL_I2C_IrqStop 的区别是不判断中断使能）

参数

$i2c\_x$	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3177 行定义.

4.92.1.106 HAL_MCPWM_IrqFail()

```
uint8_t HALMCPWM_IrqFail (
    HAL_MCPWM_x mcpwmx )
```

中断处理函数

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 614 行定义.

4.92.1.107 HAL_MCPWM_IrqFail0()

```
uint8_t HALMCPWM_IrqFail0 (
    HAL_MCPWM_x mcpwmx )
```

中断处理函数

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 460 行定义.

4.92.1.108 HAL_MCPWM_IrqFail0Clear()

```
void HALMCPWM_IrqFail0Clear (
    HAL_MCPWM_x mcpwmx )
```

中断清除

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 525 行定义.

4.92.1.109 HAL_MCPWM_IrqFail0Disable()

```
void HAL_MCPWM_IrqFail0Disable (  
    HAL_MCPWM_x mcpwmx )
```

中断禁止

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 512 行定义.

4.92.1.110 HAL_MCPWM_IrqFail0Enable()

```
void HAL_MCPWM_IrqFail0Enable (  
    HAL_MCPWM_x mcpwmx )
```

中断使能

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 499 行定义.

4.92.1.111 HAL_MCPWM_IrqFail0Flg()

```
uint8_t HAL_MCPWM_IrqFail0Flg (  
    HAL_MCPWM_x mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqFail0 的区别是不判断中断使能）

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 481 行定义.

4.92.1.112 HAL_MCPWM_IrqFail1()

```
uint8_t HALMCPWM_IrqFail1 (
    HAL_MCPWM_x mcpwmx )
```

中断处理函数

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 537 行定义.

4.92.1.113 HAL_MCPWM_IrqFail1Clear()

```
void HALMCPWM_IrqFail1Clear (
    HAL_MCPWM_x mcpwmx )
```

中断清除

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 602 行定义.

4.92.1.114 HAL_MCPWM_IrqFail1Disable()

```
void HALMCPWM_IrqFail1Disable (
    HAL_MCPWM_x mcpwmx )
```

中断禁止

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 589 行定义.

4.92.1.115 HAL_MCPWM_IrqFail1Enable()

```
void HAL_MCPWM_IrqFail1Enable (
    HAL_MCPWM_x mcpwmx )
```

中断使能

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 576 行定义.

4.92.1.116 HAL_MCPWM_IrqFail1Flg()

```
uint8_t HAL_MCPWM_IrqFail1Flg (
    HAL_MCPWM_x mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqFail1 的区别是不判断中断使能）

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 558 行定义.

4.92.1.117 HAL_MCPWM_IrqFailClear()

```
void HAL_MCPWM_IrqFailClear (
    HAL_MCPWM_x mcpwmx )
```

中断清除

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 679 行定义.

4.92.1.118 HAL_MCPWM_IrqFailDisable()

```
void HAL_MCPWM_IrqFailDisable (
    HAL_MCPWM_x mcpwmx )
```

中断禁止

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 666 行定义.

4.92.1.119 HAL_MCPWM_IrqFailEnable()

```
void HAL_MCPWM_IrqFailEnable (
    HAL_MCPWM_x mcpwmx )
```

中断使能

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 653 行定义.

4.92.1.120 HAL_MCPWM_IrqFailFlg()

```
uint8_t HAL_MCPWM_IrqFailFlg (
    HAL_MCPWM_x mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqFail 的区别是不判断中断使能）

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 635 行定义.

4.92.1.121 HAL_MCPWM_IrqOverflow()

```
uint8_t HAL_MCPWM_IrqOverflow (
    HAL_MCPWM_x mcpwmx )
```

中断处理函数



参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 306 行定义.

4.92.1.122 HAL_MCPWM_IrqOverflowClear()

```
void HAL_MCPWM_IrqOverflowClear (
    HAL_MCPWM_X mcpwmx )
```

中断清除

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 371 行定义.

4.92.1.123 HAL_MCPWM_IrqOverflowDisable()

```
void HAL_MCPWM_IrqOverflowDisable (
    HAL_MCPWM_X mcpwmx )
```

中断禁止

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 358 行定义.

4.92.1.124 HAL_MCPWM_IrqOverflowEnable()

```
void HAL_MCPWM_IrqOverflowEnable (
    HAL_MCPWM_X mcpwmx )
```

中断使能

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 345 行定义.

4.92.1.125 HAL_MCPWM_IrqOverflowFlg()

```
uint8_t HAL_MCPWM_IrqOverflowFlg (  
    HAL_MCPWM_x mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqOverflow 的区别是不判断中断使能）

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 327 行定义.

4.92.1.126 HAL_MCPWM_IrqShort()

```
uint8_t HAL_MCPWM_IrqShort (  
    HAL_MCPWM_x mcpwmx )
```

中断处理函数

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2933 行定义.

4.92.1.127 HAL_MCPWM_IrqShortClear()

```
void HAL_MCPWM_IrqShortClear (  
    HAL_MCPWM_x mcpwmx )
```

中断清除

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2998 行定义.

4.92.1.128 HAL_MCPWM_IrqShortDisable()

```
void HAL_MCPWM_IrqShortDisable (
    HAL_MCPWM_x mcpwmx )
```

中断禁止

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2985 行定义.

4.92.1.129 HAL_MCPWM_IrqShortEnable()

```
void HAL_MCPWM_IrqShortEnable (
    HAL_MCPWM_x mcpwmx )
```

中断使能

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2972 行定义.

4.92.1.130 HAL_MCPWM_IrqShortFlg()

```
uint8_t HAL_MCPWM_IrqShortFlg (
    HAL_MCPWM_x mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqShort 的区别是不判断中断使能）

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2954 行定义.

4.92.1.131 HAL_MCPWM_IrqTH00()

```
uint8_t HAL_MCPWM_IrqTH00 (
    HAL_MCPWM_x mcpwmx )
```

中断处理函数

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2317 行定义.

4.92.1.132 HAL_MCPWM_IrqTH00Clear()

```
void HAL_MCPWM_IrqTH00Clear (
    HAL_MCPWM_x mcpwmx )
```

中断清除

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2382 行定义.

4.92.1.133 HAL_MCPWM_IrqTH00Disable()

```
void HAL_MCPWM_IrqTH00Disable (
    HAL_MCPWM_x mcpwmx )
```

中断禁止

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2369 行定义.

4.92.1.134 HAL_MCPWM_IrqTH00Enable()

```
void HAL_MCPWM_IrqTH00Enable (
    HAL_MCPWM_x mcpwmx )
```

中断使能

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2356 行定义.

4.92.1.135 HAL_MCPWM_IrqTH00Flg()

```
uint8_t HAL_MCPWM_IrqTH00Flg (  
    HAL_MCPWM_x mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqTH00 的区别是不判断中断使能）

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2338 行定义.

4.92.1.136 HAL_MCPWM_IrqTH01()

```
uint8_t HAL_MCPWM_IrqTH01 (  
    HAL_MCPWM_x mcpwmx )
```

中断处理函数

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2394 行定义.

4.92.1.137 HAL_MCPWM_IrqTH01Clear()

```
void HAL_MCPWM_IrqTH01Clear (  
    HAL_MCPWM_x mcpwmx )
```

中断清除

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2459 行定义.

4.92.1.138 HAL_MCPWM_IrqTH01Disable()

```
void HAL_MCPWM_IrqTH01Disable (  
    HAL_MCPWM_x mcpwmx )
```

中断禁止

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2446 行定义.

4.92.1.139 HAL_MCPWM_IrqTH01Enable()

```
void HAL_MCPWM_IrqTH01Enable (  
    HAL_MCPWM_x mcpwmx )
```

中断使能

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2433 行定义.

4.92.1.140 HAL_MCPWM_IrqTH01Flg()

```
uint8_t HAL_MCPWM_IrqTH01Flg (  
    HAL_MCPWM_x mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqTH01 的区别是不判断中断使能）

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2415 行定义.

4.92.1.141 HAL_MCPWM_IrqTH10()

```
uint8_t HAL_MCPWM_IrqTH10 (
    HAL_MCPWM_x mcpwmx )
```

中断处理函数

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2471 行定义.

4.92.1.142 HAL_MCPWM_IrqTH10Clear()

```
void HAL_MCPWM_IrqTH10Clear (
    HAL_MCPWM_x mcpwmx )
```

中断清除

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2536 行定义.

4.92.1.143 HAL_MCPWM_IrqTH10Disable()

```
void HAL_MCPWM_IrqTH10Disable (
    HAL_MCPWM_x mcpwmx )
```

中断禁止

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2523 行定义.

4.92.1.144 HAL_MCPWM_IrqTH10Enable()

```
void HAL_MCPWM_IrqTH10Enable (
    HAL_MCPWM_x mcpwmx )
```

中断使能

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2510 行定义.

4.92.1.145 HAL_MCPWM_IrqTH10Flg()

```
uint8_t HAL_MCPWM_IrqTH10Flg (
    HAL_MCPWM_x mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqTH10 的区别是不判断中断使能）

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2492 行定义.

4.92.1.146 HAL_MCPWM_IrqTH11()

```
uint8_t HAL_MCPWM_IrqTH11 (
    HAL_MCPWM_x mcpwmx )
```

中断处理函数

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2548 行定义.

4.92.1.147 HAL_MCPWM_IrqTH11Clear()

```
void HAL_MCPWM_IrqTH11Clear (
    HAL_MCPWM_x mcpwmx )
```

中断清除

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2613 行定义.

4.92.1.148 HAL_MCPWM_IrqTH11Disable()

```
void HAL_MCPWM_IrqTH11Disable (
    HAL_MCPWM_x mcpwmx )
```

中断禁止

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2600 行定义.

4.92.1.149 HAL_MCPWM_IrqTH11Enable()

```
void HAL_MCPWM_IrqTH11Enable (
    HAL_MCPWM_x mcpwmx )
```

中断使能

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2587 行定义.

4.92.1.150 HAL_MCPWM_IrqTH11Flg()

```
uint8_t HAL_MCPWM_IrqTH11Flg (
    HAL_MCPWM_x mcpwmx )
```

中断信号 (和 HAL_MCPWM_IrqTH11 的区别是不判断中断使能)

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2569 行定义.

4.92.1.151 HAL_MCPWM_IrqTH20()

```
uint8_t HAL_MCPWM_IrqTH20 (
    HAL_MCPWM_x mcpwmx )
```

中断处理函数

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2625 行定义.

4.92.1.152 HAL_MCPWM_IrqTH20Clear()

```
void HAL_MCPWM_IrqTH20Clear (
    HAL_MCPWM_x mcpwmx )
```

中断清除

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2690 行定义.

4.92.1.153 HAL_MCPWM_IrqTH20Disable()

```
void HAL_MCPWM_IrqTH20Disable (
    HAL_MCPWM_x mcpwmx )
```

中断禁止

参数

<i>mcpwm</i> x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2677 行定义.

4.92.1.154 HAL_MCPWM_IrqTH20Enable()

```
void HAL_MCPWM_IrqTH20Enable (  
    HAL_MCPWM_x mcpwm )
```

中断使能

参数

<i>mcpwm</i> x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2664 行定义.

4.92.1.155 HAL_MCPWM_IrqTH20Flg()

```
uint8_t HAL_MCPWM_IrqTH20Flg (  
    HAL_MCPWM_x mcpwm )
```

中断信号（和 HAL_MCPWM_IrqTH20 的区别是不判断中断使能）

参数

<i>mcpwm</i> x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2646 行定义.

4.92.1.156 HAL_MCPWM_IrqTH21()

```
uint8_t HAL_MCPWM_IrqTH21 (  
    HAL_MCPWM_x mcpwm )
```

中断处理函数

参数

<i>mcpwm</i> x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2702 行定义.

4.92.1.157 HAL_MCPWM_IrqTH21Clear()

```
void HAL_MCPWM_IrqTH21Clear (
    HAL_MCPWM_x mcpwmx )
```

中断清除

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2767 行定义.

4.92.1.158 HAL_MCPWM_IrqTH21Disable()

```
void HAL_MCPWM_IrqTH21Disable (
    HAL_MCPWM_x mcpwmx )
```

中断禁止

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2754 行定义.

4.92.1.159 HAL_MCPWM_IrqTH21Enable()

```
void HAL_MCPWM_IrqTH21Enable (
    HAL_MCPWM_x mcpwmx )
```

中断使能

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2741 行定义.

4.92.1.160 HAL_MCPWM_IrqTH21Flg()

```
uint8_t HAL_MCPWM_IrqTH21Flg (
    HAL_MCPWM_x mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqTH21 的区别是不判断中断使能）

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2723 行定义.

4.92.1.161 HAL_MCPWM_IrqTH30()

```
uint8_t HAL_MCPWM_IrqTH30 (
    HAL_MCPWM_x mcpwmx )
```

中断处理函数

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2779 行定义.

4.92.1.162 HAL_MCPWM_IrqTH30Clear()

```
void HAL_MCPWM_IrqTH30Clear (
    HAL_MCPWM_x mcpwmx )
```

中断清除

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2844 行定义.

4.92.1.163 HAL_MCPWM_IrqTH30Disable()

```
void HAL_MCPWM_IrqTH30Disable (
    HAL_MCPWM_x mcpwmx )
```

中断禁止

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2831 行定义.

4.92.1.164 HAL_MCPWM_IrqTH30Enable()

```
void HAL_MCPWM_IrqTH30Enable (
    HAL_MCPWM_x mcpwmx )
```

中断使能

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2818 行定义.

4.92.1.165 HAL_MCPWM_IrqTH30Flg()

```
uint8_t HAL_MCPWM_IrqTH30Flg (
    HAL_MCPWM_x mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqTH30 的区别是不判断中断使能）

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2800 行定义.

4.92.1.166 HAL_MCPWM_IrqTH31()

```
uint8_t HAL_MCPWM_IrqTH31 (
    HAL_MCPWM_x mcpwmx )
```

中断处理函数

参数

<i>mcpwm</i> x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2856 行定义.

4.92.1.167 HAL_MCPWM_IrqTH31Clear()

```
void HAL_MCPWM_IrqTH31Clear (
    HAL_MCPWM_x mcpwm )
```

中断清除

参数

<i>mcpwm</i> x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2921 行定义.

4.92.1.168 HAL_MCPWM_IrqTH31Disable()

```
void HAL_MCPWM_IrqTH31Disable (
    HAL_MCPWM_x mcpwm )
```

中断禁止

参数

<i>mcpwm</i> x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2908 行定义.

4.92.1.169 HAL_MCPWM_IrqTH31Enable()

```
void HAL_MCPWM_IrqTH31Enable (
    HAL_MCPWM_x mcpwm )
```

中断使能

参数

<i>mcpwm</i> x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2895 行定义.

4.92.1.170 HAL_MCPWM_IrqTH31Flg()

```
uint8_t HAL_MCPWM_IrqTH31Flg (  
    HAL_MCPWM_x mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqTH31 的区别是不判断中断使能）

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2877 行定义.

4.92.1.171 HAL_MCPWM_IrqTH40()

```
uint8_t HAL_MCPWM_IrqTH40 (  
    HAL_MCPWM_x mcpwmx )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3701 行定义.

4.92.1.172 HAL_MCPWM_IrqTH40Clear()

```
void HAL_MCPWM_IrqTH40Clear (  
    HAL_MCPWM_x mcpwmx )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3719 行定义.

4.92.1.173 HAL_MCPWM_IrqTH40Disable()

```
void HAL_MCPWM_IrqTH40Disable (  
    HAL_MCPWM_x mcpwmx )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3715 行定义.

4.92.1.174 HAL_MCPWM_IrqTH40Enable()

```
void HAL_MCPWM_IrqTH40Enable (  
    HAL_MCPWM_x mcpwmx )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3711 行定义.

4.92.1.175 HAL_MCPWM_IrqTH40Flg()

```
uint8_t HAL_MCPWM_IrqTH40Flg (
    HAL_MCPWM_x mcpwmx )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3706 行定义.

4.92.1.176 HAL_MCPWM_IrqTH41()

```
uint8_t HAL_MCPWM_IrqTH41 (
    HAL_MCPWM_x mcpwmx )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3724 行定义.

4.92.1.177 HAL_MCPWM_IrqTH41Clear()

```
void HAL_MCPWM_IrqTH41Clear (
    HAL_MCPWM_x mcpwmx )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3742 行定义.

4.92.1.178 HAL_MCPWM_IrqTH41Disable()

```
void HAL_MCPWM_IrqTH41Disable (
    HAL_MCPWM_x mcpwmx )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3738 行定义.

4.92.1.179 HAL_MCPWM_IrqTH41Enable()

```
void HAL_MCPWM_IrqTH41Enable (
    HAL_MCPWM_x mcpwmx )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3734 行定义.

4.92.1.180 HAL_MCPWM_IrqTH41Flg()

```
uint8_t HAL_MCPWM_IrqTH41Flg (
    HAL_MCPWM_x mcpwmx )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3729 行定义.

4.92.1.181 HAL_MCPWM_IrqTH50()

```
uint8_t HAL_MCPWM_IrqTH50 (
    HAL_MCPWM_x mcpwmx )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3747 行定义.

4.92.1.182 HAL_MCPWM_IrqTH50Clear()

```
void HAL_MCPWM_IrqTH50Clear (
    HAL_MCPWM_TypeDef * mcpwm )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3765 行定义.

4.92.1.183 HAL_MCPWM_IrqTH50Disable()

```
void HAL_MCPWM_IrqTH50Disable (
    HAL_MCPWM_TypeDef * mcpwm )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3761 行定义.

4.92.1.184 HAL_MCPWM_IrqTH50Enable()

```
void HAL_MCPWM_IrqTH50Enable (
    HAL_MCPWM_TypeDef * mcpwm )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3757 行定义.

4.92.1.185 HAL_MCPWM_IrqTH50Flg()

```
uint8_t HAL_MCPWM_IrqTH50Flg (
    HAL_MCPWM_TypeDef * mcpwm )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3752 行定义.

4.92.1.186 HAL_MCPWM_IrqTH51()

```
uint8_t HAL_MCPWM_IrqTH51 (
    HAL_MCPWM_TypeDef * mcpwm )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3770 行定义.

4.92.1.187 HAL_MCPWM_IrqTH51Clear()

```
void HAL_MCPWM_IrqTH51Clear (
    HAL_MCPWM_TypeDef * mcpwm )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3788 行定义.

4.92.1.188 HAL_MCPWM_IrqTH51Disable()

```
void HAL_MCPWM_IrqTH51Disable (
    HAL_MCPWM_TypeDef * mcpwm )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3784 行定义.

4.92.1.189 HAL_MCPWM_IrqTH51Enable()

```
void HAL_MCPWM_IrqTH51Enable (
    HAL_MCPWM_x mcpwmx )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3780 行定义.

4.92.1.190 HAL_MCPWM_IrqTH51Flg()

```
uint8_t HAL_MCPWM_IrqTH51Flg (
    HAL_MCPWM_x mcpwmx )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3775 行定义.

4.92.1.191 HAL_MCPWM_IrqTmr0()

```
uint8_t HAL_MCPWM_IrqTmr0 (
    HAL_MCPWM_x mcpwmx )
```

中断处理函数

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2009 行定义.

4.92.1.192 HAL_MCPWM_IrqTmr0Clear()

```
void HAL_MCPWM_IrqTmr0Clear (
    HAL_MCPWM_x mcpwmx )
```

中断清除

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2074 行定义.

4.92.1.193 HAL_MCPWM_IrqTmr0Disable()

```
void HAL_MCPWM_IrqTmr0Disable (
```

```
HAL_MCPWM_x mcpwmx )
```

中断禁止

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2061 行定义.

4.92.1.194 HAL_MCPWM_IrqTmr0Enable()

```
void HAL_MCPWM_IrqTmr0Enable (  
    HAL_MCPWM_x mcpwmx )
```

中断使能

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2048 行定义.

4.92.1.195 HAL_MCPWM_IrqTmr0Flg()

```
uint8_t HAL_MCPWM_IrqTmr0Flg (  
    HAL_MCPWM_x mcpwmx )
```

中断信号 (和 HAL_MCPWM_IrqTmr0 的区别是不判断中断使能)

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生, 0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2030 行定义.

4.92.1.196 HAL_MCPWM_IrqTmr1()

```
uint8_t HAL_MCPWM_IrqTmr1 (  
    HAL_MCPWM_x mcpwmx )
```

中断处理函数

参数

<i>mcpwm</i> x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2086 行定义.

4.92.1.197 HAL_MCPWM_IrqTmr1Clear()

```
void HAL_MCPWM_IrqTmr1Clear (
    HAL_MCPWM_x mcpwm )
```

中断清除

参数

<i>mcpwm</i> x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2151 行定义.

4.92.1.198 HAL_MCPWM_IrqTmr1Disable()

```
void HAL_MCPWM_IrqTmr1Disable (
    HAL_MCPWM_x mcpwm )
```

中断禁止

参数

<i>mcpwm</i> x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2138 行定义.

4.92.1.199 HAL_MCPWM_IrqTmr1Enable()

```
void HAL_MCPWM_IrqTmr1Enable (
    HAL_MCPWM_x mcpwm )
```

中断使能

参数

<i>mcpwm</i> x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2125 行定义.

4.92.1.200 HAL_MCPWM_IrqTmr1Flg()

```
uint8_t HAL_MCPWM_IrqTmr1Flg (
    HAL_MCPWM_x mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqTmr1 的区别是不判断中断使能）

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2107 行定义.

4.92.1.201 HAL_MCPWM_IrqTmr2()

```
uint8_t HAL_MCPWM_IrqTmr2 (
    HAL_MCPWM_x mcpwmx )
```

中断处理函数

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2163 行定义.

4.92.1.202 HAL_MCPWM_IrqTmr2Clear()

```
void HAL_MCPWM_IrqTmr2Clear (
    HAL_MCPWM_x mcpwmx )
```

中断清除

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2228 行定义.

4.92.1.203 HAL_MCPWM_IrqTmr2Disable()

```
void HAL_MCPWM_IrqTmr2Disable (
    HAL_MCPWM_x mcpwmx )
```

中断禁止

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2215 行定义.

4.92.1.204 HAL_MCPWM_IrqTmr2Enable()

```
void HAL_MCPWM_IrqTmr2Enable (
    HAL_MCPWM_x mcpwmx )
```

中断使能

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2202 行定义.

4.92.1.205 HAL_MCPWM_IrqTmr2Flg()

```
uint8_t HAL_MCPWM_IrqTmr2Flg (
    HAL_MCPWM_x mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqTmr2 的区别是不判断中断使能）

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2184 行定义.

4.92.1.206 HAL_MCPWM_IrqTmr3()

```
uint8_t HAL_MCPWM_IrqTmr3 (
    HAL_MCPWM_x mcpwmx )
```

中断处理函数

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2240 行定义.

4.92.1.207 HAL_MCPWM_IrqTmr3Clear()

```
void HAL_MCPWM_IrqTmr3Clear (
    HAL_MCPWM_x mcpwmx )
```

中断清除

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2305 行定义.

4.92.1.208 HAL_MCPWM_IrqTmr3Disable()

```
void HAL_MCPWM_IrqTmr3Disable (
    HAL_MCPWM_x mcpwmx )
```

中断禁止

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2292 行定义.

4.92.1.209 HAL_MCPWM_IrqTmr3Enable()

```
void HAL_MCPWM_IrqTmr3Enable (
    HAL_MCPWM_x mcpwmx )
```

中断使能

参数

<i>mcpwm</i> x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2279 行定义.

4.92.1.210 HAL_MCPWM_IrqTmr3Flg()

```
uint8_t HAL_MCPWM_IrqTmr3Flg (
    HAL_MCPWM_x mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqTmr3 的区别是不判断中断使能）

参数

<i>mcpwm</i> x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 2261 行定义.

4.92.1.211 HAL_MCPWM_IrqUpdate()

```
uint8_t HAL_MCPWM_IrqUpdate (
    HAL_MCPWM_x mcpwmx )
```

中断处理函数

参数

<i>mcpwm</i> x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1932 行定义.

4.92.1.212 HAL_MCPWM_IrqUpdateClear()

```
void HAL_MCPWM_IrqUpdateClear (
    HAL_MCPWM_x mcpwmx )
```

中断清除

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1997 行定义.

4.92.1.213 HAL_MCPWM_IrqUpdateDisable()

```
void HAL_MCPWM_IrqUpdateDisable (
    HAL_MCPWM_x mcpwmx )
```

中断禁止

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1984 行定义.

4.92.1.214 HAL_MCPWM_IrqUpdateEnable()

```
void HAL_MCPWM_IrqUpdateEnable (
    HAL_MCPWM_x mcpwmx )
```

中断使能

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1971 行定义.

4.92.1.215 HAL_MCPWM_IrqUpdateFlg()

```
uint8_t HAL_MCPWM_IrqUpdateFlg (
    HAL_MCPWM_x mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqUpdate 的区别是不判断中断使能）

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1953 行定义.

4.92.1.216 HAL_MCPWM_IrqZeroCross()

```
uint8_t HALMCPWM_IrqZeroCross (
    HAL_MCPWM_x mcpwmx )
```

中断处理函数

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 383 行定义.

4.92.1.217 HAL_MCPWM_IrqZeroCrossClear()

```
void HALMCPWM_IrqZeroCrossClear (
    HAL_MCPWM_x mcpwmx )
```

中断清除

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 448 行定义.

4.92.1.218 HAL_MCPWM_IrqZeroCrossDisable()

```
void HALMCPWM_IrqZeroCrossDisable (
    HAL_MCPWM_x mcpwmx )
```

中断禁止

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 435 行定义.

4.92.1.219 HAL_MCPWM_IrqZeroCrossEnable()

```
void HAL_MCPWM_IrqZeroCrossEnable (
    HAL_MCPWM_x mcpwmx )
```

中断使能

参数

<i>mcpwmx</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 422 行定义.

4.92.1.220 HAL_MCPWM_IrqZeroCrossFlg()

```
uint8_t HAL_MCPWM_IrqZeroCrossFlg (
    HAL_MCPWM_x mcpwmx )
```

中断信号（和 HAL_MCPWM_IrqZeroCross 的区别是不判断中断使能）

参数

<i>mcpwmx</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 404 行定义.

4.92.1.221 HAL_SPI_IrqDone()

```
uint8_t HAL_SPI_IrqDone (
    HAL_SPI_x spi_x )
```

中断处理函数

参数

<i>spi_x</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3229 行定义.

4.92.1.222 HAL_SPI_IrqDoneClear()

```
void HAL_SPI_IrqDoneClear (
    HAL_SPI_x spi_x )
```

中断清除

参数

spi_x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3290 行定义.

4.92.1.223 HAL_SPI_IrqDoneDisable()

```
void HAL_SPI_IrqDoneDisable (
    HAL_SPI_x spi_x )
```

中断禁止

参数

spi_x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3279 行定义.

4.92.1.224 HAL_SPI_IrqDoneEnable()

```
void HAL_SPI_IrqDoneEnable (
    HAL_SPI_x spi_x )
```

中断使能

参数

spi_x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3268 行定义.

4.92.1.225 HAL_SPI_IrqDoneFlg()

```
uint8_t HAL_SPI_IrqDoneFlg (
    HAL_SPI_X spi_x )
```

中断信号（和 HAL_SPI_IrqDone 的区别是不判断中断使能）

参数

spi_x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3250 行定义.

4.92.1.226 HAL_SPI_IrqErr()

```
uint8_t HAL_SPI_IrqErr (
    HAL_SPI_X spi_x )
```

中断处理函数

参数

spi_x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3302 行定义.

4.92.1.227 HAL_SPI_IrqErrClear()

```
void HAL_SPI_IrqErrClear (
    HAL_SPI_X spi_x )
```

中断清除

参数

spi_x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3363 行定义.

4.92.1.228 HAL_SPI_IrqErrDisable()

```
void HAL_SPI_IrqErrDisable (
    HAL_SPI_X spi_x )
```

中断禁止

参数

spi_x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3352 行定义.

4.92.1.229 HAL_SPI_IrqErrEnable()

```
void HAL_SPI_IrqErrEnable (
    HAL_SPI_X spi_x )
```

中断使能

参数

spi_x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3341 行定义.

4.92.1.230 HAL_SPI_IrqErrFlg()

```
uint8_t HAL_SPI_IrqErrFlg (
    HAL_SPI_X spi_x )
```

中断信号（和 HAL_SPI_IrqErr 的区别是不判断中断使能）

参数

spi_x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3323 行定义.

4.92.1.231 HAL_SPI_IrqOverflow()

```
uint8_t HAL_SPI_IrqOverflow (
    HAL_SPI_TypeDef* spi_x )
```

中断处理函数

参数

spi_x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3375 行定义.

4.92.1.232 HAL_SPI_IrqOverflowClear()

```
void HAL_SPI_IrqOverflowClear (
    HAL_SPI_TypeDef* spi_x )
```

中断清除

参数

spi_x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3436 行定义.

4.92.1.233 HAL_SPI_IrqOverflowDisable()

```
void HAL_SPI_IrqOverflowDisable (
    HAL_SPI_TypeDef* spi_x )
```

中断禁止

参数

spi_x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3425 行定义.

4.92.1.234 HAL_SPI_IrqOverflowEnable()

```
void HAL_SPI_IrqOverflowEnable (
    HAL_SPI_TypeDef* spi_x )
```

中断使能

参数

<i>spi_x</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3414 行定义.

4.92.1.235 HAL_SPI_IrqOverflowFlg()

```
uint8_t HAL_SPI_IrqOverflowFlg (
    HAL_SPI_TypeDef* spi_x )
```

中断信号（和 HAL_SPI_IrqOverflow 的区别是不判断中断使能）

参数

<i>spi_x</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3396 行定义.

4.92.1.236 HAL_SW_Irq()

```
uint8_t HAL_SW_Irq (
    uint8_t x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3793 行定义.

4.92.1.237 HAL_SW_IrqClear()

```
void HAL_SW_IrqClear (
    uint8_t x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3811 行定义.

4.92.1.238 HAL_SW_IrqDisable()

```
void HAL_SW_IrqDisable (
    uint8_t x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3807 行定义.

4.92.1.239 HAL_SW_IrqEnable()

```
void HAL_SW_IrqEnable (
    uint8_t x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3803 行定义.

4.92.1.240 HAL_SW_IrqFlg()

```
uint8_t HAL_SW_IrqFlg (
    uint8_t x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3798 行定义.

4.92.1.241 HAL_SW_IrqTrig()

```
void HAL_SW_IrqTrig (
    uint8_t x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3816 行定义.

4.92.1.242 HAL_TIMER_IrqCompare0()

```
uint8_t HAL_TIMER_IrqCompare0 (
    HAL_TIMER_x timer_x )
```

中断处理函数

参数

<i>timer_x</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 837 行定义.

4.92.1.243 HAL_TIMER_IrqCompare0Clear()

```
void HAL_TIMER_IrqCompare0Clear (
    HAL_TIMER_x timer_x )
```

中断清除

参数

<i>timer</i> ↔ _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 898 行定义.

4.92.1.244 HAL_TIMER_IrqCompare0Disable()

```
void HAL_TIMER_IrqCompare0Disable (
    HAL_TIMER_x timer_x )
```

中断禁止

参数

<i>timer</i> ↔ _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 887 行定义.

4.92.1.245 HAL_TIMER_IrqCompare0Enable()

```
void HAL_TIMER_IrqCompare0Enable (
    HAL_TIMER_x timer_x )
```

中断使能

参数

<i>timer</i> ↔ _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 876 行定义.

4.92.1.246 HAL_TIMER_IrqCompare0Flg()

```
uint8_t HAL_TIMER_IrqCompare0Flg (
    HAL_TIMER_x timer_x )
```

中断信号（和 HAL_TIMER_IrqCompare0 的区别是不判断中断使能）

参数

$timer_{x}$	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 858 行定义.

4.92.1.247 HAL_TIMER_IrqCompare1()

```
uint8_t HAL_TIMER_IrqCompare1 (
    HAL_TIMER_x timer_x )
```

中断处理函数

参数

$timer_{x}$	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 910 行定义.

4.92.1.248 HAL_TIMER_IrqCompare1Clear()

```
void HAL_TIMER_IrqCompare1Clear (
    HAL_TIMER_x timer_x )
```

中断清除

参数

$timer_{x}$	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 971 行定义.

4.92.1.249 HAL_TIMER_IrqCompare1Disable()

```
void HAL_TIMER_IrqCompare1Disable (
    HAL_TIMER_x timer_x )
```

中断禁止

参数

<i>timer</i> ↔ _X	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 960 行定义.

4.92.1.250 HAL_TIMER_IrqCompare1Enable()

```
void HAL_TIMER_IrqCompare1Enable (
    HAL_TIMER_X timer_x )
```

中断使能

参数

<i>timer</i> ↔ _X	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 949 行定义.

4.92.1.251 HAL_TIMER_IrqCompare1Flg()

```
uint8_t HAL_TIMER_IrqCompare1Flg (
    HAL_TIMER_X timer_x )
```

中断信号（和 HAL_TIMER_IrqCompare1 的区别是不判断中断使能）

参数

<i>timer</i> ↔ _X	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 931 行定义.

4.92.1.252 HAL_TIMER_IrqOverflow()

```
uint8_t HAL_TIMER_IrqOverflow (
    HAL_TIMER_X timer_x )
```

中断处理函数

参数

<i>timer</i> ↔ _X	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 764 行定义.

4.92.1.253 HAL_TIMER_IrqOverflowClear()

```
void HAL_TIMER_IrqOverflowClear (
    HAL_TIMER_X timer_x )
```

中断清除

参数

<i>timer</i> ↔ _X	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 825 行定义.

4.92.1.254 HAL_TIMER_IrqOverflowDisable()

```
void HAL_TIMER_IrqOverflowDisable (
    HAL_TIMER_X timer_x )
```

中断禁止

参数

<i>timer</i> ↔ _X	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 814 行定义.

4.92.1.255 HAL_TIMER_IrqOverflowEnable()

```
void HAL_TIMER_IrqOverflowEnable (
    HAL_TIMER_X timer_x )
```

中断使能

参数

<i>timer</i> ↔ _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 803 行定义.

4.92.1.256 HAL_TIMER_IrqOverflowFlg()

```
uint8_t HAL_TIMER_IrqOverflowFlg (
    HAL_TIMER_x timer_x )
```

中断信号（和 HAL_TIMER_IrqOverflow 的区别是不判断中断使能）

参数

<i>timer</i> ↔ _x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 785 行定义.

4.92.1.257 HAL_UART_IrqAbd()

```
uint8_t HAL_UART_IrqAbd (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3821 行定义.

4.92.1.258 HAL_UART_IrqAbdClear()

```
void HAL_UART_IrqAbdClear (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3839 行定义.

4.92.1.259 HAL_UART_IrqAbdDisable()

```
void HAL_UART_IrqAbdDisable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3835 行定义.

4.92.1.260 HAL_UART_IrqAbdEnable()

```
void HAL_UART_IrqAbdEnable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3831 行定义.

4.92.1.261 HAL_UART_IrqAbdFlg()

```
uint8_t HAL_UART_IrqAbdFlg (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3826 行定义.

4.92.1.262 HAL_UART_IrqCheckErr()

```
uint8_t HAL_UART_IrqCheckErr (
    HAL_UART_x uart_x )
```

中断处理函数

参数

<i>uart_x</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1202 行定义.

4.92.1.263 HAL_UART_IrqCheckErrClear()

```
void HAL_UART_IrqCheckErrClear (
    HAL_UART_x uart_x )
```

中断清除

参数

<i>uart_x</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1263 行定义.

4.92.1.264 HAL_UART_IrqCheckErrDisable()

```
void HAL_UART_IrqCheckErrDisable (
    HAL_UART_x uart_x )
```

中断禁止

参数

<i>uart_x</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1252 行定义.

4.92.1.265 HAL_UART_IrqCheckErrEnable()

```
void HAL_UART_IrqCheckErrEnable (
    HAL_UART_x uart_x )
```

中断使能

参数

<i>uart_x</i>	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1241 行定义.

4.92.1.266 HAL_UART_IrqCheckErrFlg()

```
uint8_t HAL_UART_IrqCheckErrFlg (
    HAL_UART_x uart_x )
```

中断信号（和 HAL_UART_IrqCheckErr 的区别是不判断中断使能）

参数

<i>uart_x</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1223 行定义.

4.92.1.267 HAL_UART_IrqIdle()

```
uint8_t HAL_UART_IrqIdle (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3844 行定义.

4.92.1.268 HAL_UART_IrqIdleClear()

```
void HAL_UART_IrqIdleClear (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3862 行定义.

4.92.1.269 HAL_UART_IrqIdleDisable()

```
void HAL_UART_IrqIdleDisable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3858 行定义.

4.92.1.270 HAL_UART_IrqIdleEnable()

```
void HAL_UART_IrqIdleEnable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3854 行定义.

4.92.1.271 HAL_UART_IrqIdleFlg()

```
uint8_t HAL_UART_IrqIdleFlg (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3849 行定义.

4.92.1.272 HAL_UART_IrqLbd()

```
uint8_t HAL_UART_IrqLbd (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3867 行定义.

4.92.1.273 HAL_UART_IrqLbdClear()

```
void HAL_UART_IrqLbdClear (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3885 行定义.

4.92.1.274 HAL_UART_IrqLbdDisable()

```
void HAL_UART_IrqLbdDisable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3881 行定义.

4.92.1.275 HAL_UART_IrqLbdEnable()

```
void HAL_UART_IrqLbdEnable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3877 行定义.

4.92.1.276 HAL_UART_IrqLbdFlg()

```
uint8_t HAL_UART_IrqLbdFlg (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3872 行定义.

4.92.1.277 HAL_UART_IrqRxDone()

```
uint8_t HAL_UART_IrqRxDone (
    HAL_UART_x uart_x )
```

中断处理函数

参数

<i>uart_x</i>	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1056 行定义.

4.92.1.278 HAL_UART_IrqRxDoneClear()

```
void HAL_UART_IrqRxDoneClear (
    HAL_UART_x uart_x )
```

中断清除

参数

<i>uart</i> _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1117 行定义.

4.92.1.279 HAL_UART_IrqRxDoneDisable()

```
void HAL_UART_IrqRxDoneDisable (
    HAL_UART_x uart_x )
```

中断禁止

参数

<i>uart</i> _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1106 行定义.

4.92.1.280 HAL_UART_IrqRxDoneEnable()

```
void HAL_UART_IrqRxDoneEnable (
    HAL_UART_x uart_x )
```

中断使能

参数

<i>uart</i> _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1095 行定义.

4.92.1.281 HAL_UART_IrqRxDoneFlg()

```
uint8_t HAL_UART_IrqRxDoneFlg (
    HAL_UART_x uart_x )
```

中断信号（和 HAL_UART_IrqRxDone 的区别是不判断中断使能）

参数

<i>uart</i> _x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1077 行定义.

4.92.1.282 HAL_UART_IrqRxOv()

```
uint8_t HAL_UART_IrqRxOv (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3890 行定义.

4.92.1.283 HAL_UART_IrqRxOvClear()

```
void HAL_UART_IrqRxOvClear (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3908 行定义.

4.92.1.284 HAL_UART_IrqRxOvDisable()

```
void HAL_UART_IrqRxOvDisable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3904 行定义.

4.92.1.285 HAL_UART_IrqRxOvEnable()

```
void HAL_UART_IrqRxOvEnable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3900 行定义.

4.92.1.286 HAL_UART_IrqRxOvFlg()

```
uint8_t HAL_UART_IrqRxOvFlg (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3895 行定义.

4.92.1.287 HAL_UART_IrqStopErr()

```
uint8_t HAL_UART_IrqStopErr (
    HAL_UART_x uart_x )
```

中断处理函数

参数

<i>uart</i> _x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1129 行定义.

4.92.1.288 HAL_UART_IrqStopErrClear()

```
void HAL_UART_IrqStopErrClear (
    HAL_UART_x uart_x )
```

中断清除

参数

<i>uart</i> _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1190 行定义.

4.92.1.289 HAL_UART_IrqStopErrDisable()

```
void HAL_UART_IrqStopErrDisable (
    HAL_UART_x uart_x )
```

中断禁止

参数

<i>uart</i> _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1179 行定义.

4.92.1.290 HAL_UART_IrqStopErrEnable()

```
void HAL_UART_IrqStopErrEnable (
    HAL_UART_x uart_x )
```

中断使能

参数

<i>uart</i> ↔ _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1168 行定义.

4.92.1.291 HAL_UART_IrqStopErrFlg()

```
uint8_t HAL_UART_IrqStopErrFlg (  
    HAL_UART_x uart_x )
```

中断信号（和 HAL_UART_IrqStopErr 的区别是不判断中断使能）

参数

<i>uart</i> ↔ _x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1150 行定义.

4.92.1.292 HAL_UART_IrqTxBufEmpty()

```
uint8_t HAL_UART_IrqTxBufEmpty (  
    HAL_UART_x uart_x )
```

中断处理函数

参数

<i>uart</i> ↔ _x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1494 行定义.

4.92.1.293 HAL_UART_IrqTxBufEmptyClear()

```
void HAL_UART_IrqTxBufEmptyClear (  
    HAL_UART_x uart_x )
```

中断清除

参数

<i>uart</i> _{<i>_x</i>}	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1555 行定义.

4.92.1.294 HAL_UART_IrqTxBufEmptyDisable()

```
void HAL_UART_IrqTxBufEmptyDisable (
    HAL_UART_<i>x</i> <i>uart_x</i> )
```

中断禁止

参数

<i>uart</i> _{<i>_x</i>}	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1544 行定义.

4.92.1.295 HAL_UART_IrqTxBufEmptyEnable()

```
void HAL_UART_IrqTxBufEmptyEnable (
    HAL_UART_<i>x</i> <i>uart_x</i> )
```

中断使能

参数

<i>uart</i> _{<i>_x</i>}	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1533 行定义.

4.92.1.296 HAL_UART_IrqTxBufEmptyFlg()

```
uint8_t HAL_UART_IrqTxBufEmptyFlg (
    HAL_UART_<i>x</i> <i>uart_x</i> )
```

中断信号（和 HAL_UART_IrqTxBufEmpty 的区别是不判断中断使能）

参数

<i>uart</i> _{<i>_x</i>}	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1515 行定义.

4.92.1.297 HAL_UART_IrqTxDone()

```
uint8_t HAL_UART_IrqTxDone (
    HAL_UART_x uart_x )
```

中断处理函数

参数

<i>uart</i> _x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 983 行定义.

4.92.1.298 HAL_UART_IrqTxDoneClear()

```
void HAL_UART_IrqTxDoneClear (
    HAL_UART_x uart_x )
```

中断清除

参数

<i>uart</i> _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1044 行定义.

4.92.1.299 HAL_UART_IrqTxDoneDisable()

```
void HAL_UART_IrqTxDoneDisable (
    HAL_UART_x uart_x )
```

中断禁止

参数

<i>uart</i> _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1033 行定义.

4.92.1.300 HAL_UART_IrqTxDoneEnable()

```
void HAL_UART_IrqTxDoneEnable (  
    HAL_UART_<i>x</i> <i>uart_x</i> )
```

中断使能

参数

<i>uart</i> _x	模块编号

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1022 行定义.

4.92.1.301 HAL_UART_IrqTxDoneFlg()

```
uint8_t HAL_UART_IrqTxDoneFlg (  
    HAL_UART_<i>x</i> <i>uart_x</i> )
```

中断信号（和 HAL_UART_IrqTxDone 的区别是不判断中断使能）

参数

<i>uart</i> _x	模块编号

返回

1表示中断发生，0表示没有中断发生

在文件 [hal_lks32mc08x_irqfun.c](#) 第 1004 行定义.

4.92.1.302 HAL_UART_IrqTxOv()

```
uint8_t HAL_UART_IrqTxOv (  
    HAL_UART_<i>x</i> <i>uart_x</i> )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3913 行定义.

4.92.1.303 HAL_UART_IrqTxOvClear()

```
void HAL_UART_IrqTxOvClear (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3931 行定义。

4.92.1.304 HAL_UART_IrqTxOvDisable()

```
void HAL_UART_IrqTxOvDisable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3927 行定义。

4.92.1.305 HAL_UART_IrqTxOvEnable()

```
void HAL_UART_IrqTxOvEnable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3923 行定义。

4.92.1.306 HAL_UART_IrqTxOvFlg()

```
uint8_t HAL_UART_IrqTxOvFlg (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_irqfun.c](#) 第 3918 行定义。

4.93 hal_lks32mc08x_irqfun.c

[浏览该文件的文档.](#)

```
00001
00002
00003 #include "../hal_lks32mc08x.h"
00004 #include "lks32mc08x_user_manual.h"
00005 #include "../include/hal_lksmcu_reg.h"
00006 #include "../include/hal_lks32mc08x_reg.h"
00007 #include "../include/hal_lksmcu_irqfun.h"
00008
00014 uint8_t HAL_HALL_IrqChange(HAL_HALL_x hall)
00015 {
00016     volatile uint32_t *addr_if;
00017     volatile uint32_t *addr_ie;
00018     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_HALL[hall]->CFG);
00019     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_HALL[hall]->INFO);
00020     if ((HAL_REG_READ_BIT(*addr_ie, HALL_CFG_CHG_IE_BIT) && (HAL_REG_READ_BIT(*addr_if,
HALL_INFO_CHG_IF_BIT)))
00021     {
00022         return 1;
00023     }
00024     else
00025     {
00026         return 0;
00027     }
00028 }
00029
00035 uint8_t HAL_HALL_IrqChangeFlg(HAL_HALL_x hall)
00036 {
00037     volatile uint32_t *addr;
00038     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_HALL[hall]->INFO);
```

```

00039     if (HAL_REG_READ_BIT(*addr, HALL_INFO_CHG_IF_BIT))
00040     {
00041         return 1;
00042     }
00043     else
00044     {
00045         return 0;
00046     }
00047 }
00048
00053 void HAL_HALL_IrqChangeEnable(HAL_HALL_x hall)
00054 {
00055     volatile uint32_t *addr;
00056     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_HALL[hall]->CFG);
00057     HAL_REG_SET(*addr, HALL_CFG_CHG_IE_BIT);
00058 }
00059
00064 void HAL_HALL_IrqChangeDisable(HAL_HALL_x hall)
00065 {
00066     volatile uint32_t *addr;
00067     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_HALL[hall]->CFG);
00068     HAL_REG_RESET(*addr, HALL_CFG_CHG_IE_BIT);
00069 }
00070
00075 void HAL_HALL_IrqChangeClear(HAL_HALL_x hall)
00076 {
00077     volatile uint32_t *addr;
00078     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_HALL[hall]->INFO);
00079     HAL_REG_WRITE(*addr, HALL_INFO_CHG_IF_BIT);
00080 }
00081
00087 uint8_t HAL_HALL_IrqOver(HAL_HALL_x hall)
00088 {
00089     volatile uint32_t *addr_if;
00090     volatile uint32_t *addr_ie;
00091     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_HALL[hall]->CFG);
00092     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_HALL[hall]->INFO);
00093     if ((HAL_REG_READ_BIT(*addr_ie, HALL_CFG_OV_IE_BIT)) && (HAL_REG_READ_BIT(*addr_if,
HALL_INFO_OV_IF_BIT)))
00094     {
00095         return 1;
00096     }
00097     else
00098     {
00099         return 0;
00100     }
00101 }
00102
00108 uint8_t HAL_HALL_IrqOverFlg(HAL_HALL_x hall)
00109 {
00110     volatile uint32_t *addr;
00111     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_HALL[hall]->INFO);
00112     if (HAL_REG_READ_BIT(*addr, HALL_INFO_OV_IF_BIT))
00113     {
00114         return 1;
00115     }
00116     else
00117     {
00118         return 0;
00119     }
00120 }
00121
00126 void HAL_HALL_IrqOverEnable(HAL_HALL_x hall)
00127 {
00128     volatile uint32_t *addr;
00129     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_HALL[hall]->CFG);
00130     HAL_REG_SET(*addr, HALL_CFG_OV_IE_BIT);
00131 }
00132
00137 void HAL_HALL_IrqOverDisable(HAL_HALL_x hall)
00138 {
00139     volatile uint32_t *addr;
00140     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_HALL[hall]->CFG);
00141     HAL_REG_RESET(*addr, HALL_CFG_OV_IE_BIT);
00142 }
00143
00148 void HAL_HALL_IrqOverClear(HAL_HALL_x hall)
00149 {
00150     volatile uint32_t *addr;
00151     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_HALL[hall]->INFO);
00152     HAL_REG_WRITE(*addr, HALL_INFO_OV_IF_BIT);
00153 }
00154
00160 uint8_t HAL_ADC_IrqSamp1(HAL_ADC_x adc)
00161 {
00162     volatile uint32_t *addr_if;
00163     volatile uint32_t *addr_ie;

```

```

00164     addr.ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IE);
00165     addr.if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IF);
00166     if ((HAL_REG_READ_BIT(*addr.ie, ADC0_IE_S1FINISH_IE_BIT) && (HAL_REG_READ_BIT(*addr.if,
ADC0_IF_S1FINISH_IF_BIT)))
00167     {
00168         return 1;
00169     }
00170     else
00171     {
00172         return 0;
00173     }
00174 }
00175
00181 uint8_t HAL_ADC_IrqSamp1Flg(HAL_ADC_x adc)
00182 {
00183     volatile uint32_t *addr;
00184     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IF);
00185     if (HAL_REG_READ_BIT(*addr, ADC0_IF_S1FINISH_IF_BIT))
00186     {
00187         return 1;
00188     }
00189     else
00190     {
00191         return 0;
00192     }
00193 }
00194
00199 void HAL_ADC_IrqSamp1Enable(HAL_ADC_x adc)
00200 {
00201     volatile uint32_t *addr;
00202     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IE);
00203     HAL_REG_SET(*addr, ADC0_IE_S1FINISH_IE_BIT);
00204 }
00205
00210 void HAL_ADC_IrqSamp1Disable(HAL_ADC_x adc)
00211 {
00212     volatile uint32_t *addr;
00213     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IE);
00214     HAL_REG_RESET(*addr, ADC0_IE_S1FINISH_IE_BIT);
00215 }
00216
00221 void HAL_ADC_IrqSamp1Clear(HAL_ADC_x adc)
00222 {
00223     volatile uint32_t *addr;
00224     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IF);
00225     HAL_REG_WRITE(*addr, ADC0_IF_S1FINISH_IF_BIT);
00226 }
00227
00233 uint8_t HAL_ADC_IrqSamp2(HAL_ADC_x adc)
00234 {
00235     volatile uint32_t *addr.if;
00236     volatile uint32_t *addr.ie;
00237     addr.ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IE);
00238     addr.if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IF);
00239     if ((HAL_REG_READ_BIT(*addr.ie, ADC0_IE_S2FINISH_IE_BIT) && (HAL_REG_READ_BIT(*addr.if,
ADC0_IF_S2FINISH_IF_BIT)))
00240     {
00241         return 1;
00242     }
00243     else
00244     {
00245         return 0;
00246     }
00247 }
00248
00254 uint8_t HAL_ADC_IrqSamp2Flg(HAL_ADC_x adc)
00255 {
00256     volatile uint32_t *addr;
00257     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IF);
00258     if (HAL_REG_READ_BIT(*addr, ADC0_IF_S2FINISH_IF_BIT))
00259     {
00260         return 1;
00261     }
00262     else
00263     {
00264         return 0;
00265     }
00266 }
00267
00272 void HAL_ADC_IrqSamp2Enable(HAL_ADC_x adc)
00273 {
00274     volatile uint32_t *addr;
00275     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IE);
00276     HAL_REG_SET(*addr, ADC0_IE_S2FINISH_IE_BIT);
00277 }
00278
00283 void HAL_ADC_IrqSamp2Disable(HAL_ADC_x adc)

```

```

00284 {
00285     volatile uint32_t *addr;
00286     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IE);
00287     HAL_REG_RESET(*addr, ADC0_IE_S2FINISH_IE_BIT);
00288 }
00289
00294 void HAL_ADC_IrqSamp2Clear(HAL_ADC_x adc)
00295 {
00296     volatile uint32_t *addr;
00297     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IF);
00298     HAL_REG_WRITE(*addr, ADC0_IF_S2FINISH_IF_BIT);
00299 }
00300
00306 uint8_t HAL_MCPWM_IrqOverflow(HAL_MCPWM_x mcpwm)
00307 {
00308     volatile uint32_t *addr_if;
00309     volatile uint32_t *addr_ie;
00310     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwm & 0xf]->IE0);
00311     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwm & 0xf]->IF0);
00312     if ((HAL_REG_READ_BIT(*addr_ie, MCPWM_IE_T0_IE_BIT)) && (HAL_REG_READ_BIT(*addr_if,
MCPWM_IF_T0_IF_BIT)))
00313     {
00314         return 1;
00315     }
00316     else
00317     {
00318         return 0;
00319     }
00320 }
00321
00327 uint8_t HAL_MCPWM_IrqOverflowFlg(HAL_MCPWM_x mcpwm)
00328 {
00329     volatile uint32_t *addr;
00330     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwm & 0xf]->IF0);
00331     if (HAL_REG_READ_BIT(*addr, MCPWM_IF_T0_IF_BIT))
00332     {
00333         return 1;
00334     }
00335     else
00336     {
00337         return 0;
00338     }
00339 }
00340
00345 void HAL_MCPWM_IrqOverflowEnable(HAL_MCPWM_x mcpwm)
00346 {
00347     volatile uint32_t *addr;
00348     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwm & 0xf]->IE0);
00349     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
00350     HAL_REG_SET(*addr, MCPWM_IE_T0_IE_BIT);
00351     HAL_REG_WRITE(MCPWM_PRT, 0);
00352 }
00353
00358 void HAL_MCPWM_IrqOverflowDisable(HAL_MCPWM_x mcpwm)
00359 {
00360     volatile uint32_t *addr;
00361     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwm & 0xf]->IE0);
00362     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
00363     HAL_REG_RESET(*addr, MCPWM_IE_T0_IE_BIT);
00364     HAL_REG_WRITE(MCPWM_PRT, 0);
00365 }
00366
00371 void HAL_MCPWM_IrqOverflowClear(HAL_MCPWM_x mcpwm)
00372 {
00373     volatile uint32_t *addr;
00374     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwm & 0xf]->IF0);
00375     HAL_REG_WRITE(*addr, MCPWM_IF_T0_IF_BIT);
00376 }
00377
00383 uint8_t HAL_MCPWM_IrqZeroCross(HAL_MCPWM_x mcpwm)
00384 {
00385     volatile uint32_t *addr_if;
00386     volatile uint32_t *addr_ie;
00387     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwm & 0xf]->IE0);
00388     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwm & 0xf]->IF0);
00389     if ((HAL_REG_READ_BIT(*addr_ie, MCPWM_IE_T1_IE_BIT)) && (HAL_REG_READ_BIT(*addr_if,
MCPWM_IF_T1_IF_BIT)))
00390     {
00391         return 1;
00392     }
00393     else
00394     {
00395         return 0;
00396     }
00397 }
00398
00404 uint8_t HAL_MCPWM_IrqZeroCrossFlg(HAL_MCPWM_x mcpwm)

```

```

00405 {
00406     volatile uint32_t *addr;
00407     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
00408     if (HAL_REG_READ_BIT(*addr, MCPWM_IF_T1_IF_BIT))
00409     {
00410         return 1;
00411     }
00412     else
00413     {
00414         return 0;
00415     }
00416 }
00417
00422 void HAL_MCPWM_IrqZeroCrossEnable(HAL_MCPWM_x mcpwmx)
00423 {
00424     volatile uint32_t *addr;
00425     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
00426     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
00427     HAL_REG_SET(*addr, MCPWM_IE_T1_IE_BIT);
00428     HAL_REG_WRITE(MCPWM_PRT, 0);
00429 }
00430
00435 void HAL_MCPWM_IrqZeroCrossDisable(HAL_MCPWM_x mcpwmx)
00436 {
00437     volatile uint32_t *addr;
00438     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
00439     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
00440     HAL_REG_RESET(*addr, MCPWM_IE_T1_IE_BIT);
00441     HAL_REG_WRITE(MCPWM_PRT, 0);
00442 }
00443
00448 void HAL_MCPWM_IrqZeroCrossClear(HAL_MCPWM_x mcpwmx)
00449 {
00450     volatile uint32_t *addr;
00451     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
00452     HAL_REG_WRITE(*addr, MCPWM_IF_T1_IF_BIT);
00453 }
00454
00460 uint8_t HAL_MCPWM_IrqFail0(HAL_MCPWM_x mcpwmx)
00461 {
00462     volatile uint32_t *addr_if;
00463     volatile uint32_t *addr_ie;
00464     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->EIE);
00465     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->EIF);
00466     if ((HAL_REG_READ_BIT(*addr_ie, MCPWM_EIE_FAIL0_IE_BIT)) && (HAL_REG_READ_BIT(*addr_if,
MCPWM{EIF_FAIL0_IF_BIT})))
00467     {
00468         return 1;
00469     }
00470     else
00471     {
00472         return 0;
00473     }
00474 }
00475
00481 uint8_t HAL_MCPWM_IrqFail0Flg(HAL_MCPWM_x mcpwmx)
00482 {
00483     volatile uint32_t *addr;
00484     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->EIF);
00485     if (HAL_REG_READ_BIT(*addr, MCPWM{EIF_FAIL0_IF_BIT}))
00486     {
00487         return 1;
00488     }
00489     else
00490     {
00491         return 0;
00492     }
00493 }
00494
00499 void HAL_MCPWM_IrqFail0Enable(HAL_MCPWM_x mcpwmx)
00500 {
00501     volatile uint32_t *addr;
00502     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->EIE);
00503     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
00504     HAL_REG_SET(*addr, MCPWM_EIE_FAIL0_IE_BIT);
00505     HAL_REG_WRITE(MCPWM_PRT, 0);
00506 }
00507
00512 void HAL_MCPWM_IrqFail0Disable(HAL_MCPWM_x mcpwmx)
00513 {
00514     volatile uint32_t *addr;
00515     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->EIE);
00516     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
00517     HAL_REG_RESET(*addr, MCPWM_EIE_FAIL0_IE_BIT);
00518     HAL_REG_WRITE(MCPWM_PRT, 0);
00519 }
00520

```



```

00525 void HAL_MCPWM_IrqFail0Clear(HAL_MCPWM_x mcpwmx)
00526 {
00527     volatile uint32_t *addr;
00528     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->EIF);
00529     HAL_REG_WRITE(*addr, MCPWM_EIF_FAIL0_IF_BIT);
00530 }
00531
00537 uint8_t HAL_MCPWM_IrqFail1(HAL_MCPWM_x mcpwmx)
00538 {
00539     volatile uint32_t *addr_if;
00540     volatile uint32_t *addr_ie;
00541     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->EIE);
00542     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->EIF);
00543     if ((HAL_REG_READ_BIT(*addr_ie, MCPWM_EIE_FAIL1_IE_BIT)) && (HAL_REG_READ_BIT(*addr_if,
MCPWM_EIF_FAIL1_IF_BIT)))
00544     {
00545         return 1;
00546     }
00547     else
00548     {
00549         return 0;
00550     }
00551 }
00552
00558 uint8_t HAL_MCPWM_IrqFail1Flg(HAL_MCPWM_x mcpwmx)
00559 {
00560     volatile uint32_t *addr;
00561     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->EIF);
00562     if (HAL_REG_READ_BIT(*addr, MCPWM_EIF_FAIL1_IF_BIT))
00563     {
00564         return 1;
00565     }
00566     else
00567     {
00568         return 0;
00569     }
00570 }
00571
00576 void HAL_MCPWM_IrqFail1Enable(HAL_MCPWM_x mcpwmx)
00577 {
00578     volatile uint32_t *addr;
00579     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->EIE);
00580     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
00581     HAL_REG_SET(*addr, MCPWM_EIE_FAIL1_IE_BIT);
00582     HAL_REG_WRITE(MCPWM_PRT, 0);
00583 }
00584
00589 void HAL_MCPWM_IrqFail1Disable(HAL_MCPWM_x mcpwmx)
00590 {
00591     volatile uint32_t *addr;
00592     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->EIE);
00593     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
00594     HAL_REG_RESET(*addr, MCPWM_EIE_FAIL1_IE_BIT);
00595     HAL_REG_WRITE(MCPWM_PRT, 0);
00596 }
00597
00602 void HAL_MCPWM_IrqFail1Clear(HAL_MCPWM_x mcpwmx)
00603 {
00604     volatile uint32_t *addr;
00605     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->EIF);
00606     HAL_REG_WRITE(*addr, MCPWM_EIF_FAIL1_IF_BIT);
00607 }
00608
00614 uint8_t HAL_MCPWM_IrqFail(HAL_MCPWM_x mcpwmx)
00615 {
00616     volatile uint32_t *addr_if;
00617     volatile uint32_t *addr_ie;
00618     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->EIE);
00619     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->EIF);
00620     if ((HAL_REG_READ_BIT(*addr_ie, MCPWM_EIE_FAIL0_IE_BIT | MCPWM_EIE_FAIL1_IE_BIT)) &&
(HAL_REG_READ_BIT(*addr_if, MCPWM_EIF_FAIL0_IF_BIT | MCPWM_EIF_FAIL1_IF_BIT)))
00621     {
00622         return 1;
00623     }
00624     else
00625     {
00626         return 0;
00627     }
00628 }
00629
00635 uint8_t HAL_MCPWM_IrqFailFlg(HAL_MCPWM_x mcpwmx)
00636 {
00637     volatile uint32_t *addr;
00638     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->EIF);
00639     if (HAL_REG_READ_BIT(*addr, MCPWM_EIF_FAIL0_IF_BIT | MCPWM_EIF_FAIL1_IF_BIT))
00640     {
00641         return 1;

```

```

00642     }
00643     else
00644     {
00645         return 0;
00646     }
00647 }
00648
00653 void HAL_MCPWM_IrqFailEnable(HAL_MCPWM_x mcpwmx)
00654 {
00655     volatile uint32_t *addr;
00656     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->EIE);
00657     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
00658     HAL_REG_SET(*addr, MCPWM_EIE_FAIL0_IE_BIT | MCPWM_EIE_FAIL1_IE_BIT);
00659     HAL_REG_WRITE(MCPWM_PRT, 0);
00660 }
00661
00666 void HAL_MCPWM_IrqFailDisable(HAL_MCPWM_x mcpwmx)
00667 {
00668     volatile uint32_t *addr;
00669     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->EIE);
00670     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
00671     HAL_REG_RESET(*addr, MCPWM_EIE_FAIL0_IE_BIT | MCPWM_EIE_FAIL1_IE_BIT);
00672     HAL_REG_WRITE(MCPWM_PRT, 0);
00673 }
00674
00679 void HAL_MCPWM_IrqFailClear(HAL_MCPWM_x mcpwmx)
00680 {
00681     volatile uint32_t *addr;
00682     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->EIF);
00683     HAL_REG_WRITE(*addr, MCPWM{EIF_FAIL0_IF_BIT | MCPWM{EIF_FAIL1_IF_BIT});
00684 }
00685
00691 uint8_t HAL_CMP_IrqCompare(HAL_CMP_x cmp_x)
00692 {
00693     volatile uint32_t *addr_if;
00694     volatile uint32_t *addr_ie;
00695     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_CMP[0]->IE);
00696     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_CMP[0]->IF);
00697     if ((HAL_REG_READ_BIT(*addr_ie, CMP_IE_CMP0_IE_BIT << (cmp_x))) && (HAL_REG_READ_BIT(*addr_if,
CMP_IF_CMP0_IF_BIT << (cmp_x))))
00698     {
00699         return 1;
00700     }
00701     else
00702     {
00703         return 0;
00704     }
00705 }
00706
00712 uint8_t HAL_CMP_IrqCompareFlg(HAL_CMP_x cmp_x)
00713 {
00714     volatile uint32_t *addr;
00715     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_CMP[0]->IF);
00716     if (HAL_REG_READ_BIT(*addr, CMP_IF_CMP0_IF_BIT << (cmp_x)))
00717     {
00718         return 1;
00719     }
00720     else
00721     {
00722         return 0;
00723     }
00724 }
00725
00730 void HAL_CMP_IrqCompareEnable(HAL_CMP_x cmp_x)
00731 {
00732     volatile uint32_t *addr;
00733     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_CMP[0]->IE);
00734     HAL_REG_SET(*addr, CMP_IE_CMP0_IE_BIT << (cmp_x));
00735 }
00736
00741 void HAL_CMP_IrqCompareDisable(HAL_CMP_x cmp_x)
00742 {
00743     volatile uint32_t *addr;
00744     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_CMP[0]->IE);
00745     HAL_REG_RESET(*addr, CMP_IE_CMP0_IE_BIT << (cmp_x));
00746 }
00747
00752 void HAL_CMP_IrqCompareClear(HAL_CMP_x cmp_x)
00753 {
00754     volatile uint32_t *addr;
00755     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_CMP[0]->IF);
00756     HAL_REG_WRITE(*addr, CMP_IF_CMP0_IF_BIT << (cmp_x));
00757 }
00758
00764 uint8_t HAL_TIMER_IrqOverflow(HAL_TIMER_x timer_x)
00765 {
00766     volatile uint32_t *addr_if;

```

```

00767     volatile uint32_t *addr_ie;
00768     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(UTIMER_IE);
00769     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(UTIMER_IF);
00770     if ((HAL_REG_READ_BIT(*addr_ie, UTIMER_IE_T0_ZC_IE_BIT << (timer_x * 3))) &&
        (HAL_REG_READ_BIT(*addr_if, UTIMER_IF_T0_ZC_IF_BIT << (timer_x * 3))))
00771     {
00772         return 1;
00773     }
00774     else
00775     {
00776         return 0;
00777     }
00778 }
00779
00785 uint8_t HAL_TIMER_IrqOverflowFlg(HAL_TIMER_x timer_x)
00786 {
00787     volatile uint32_t *addr;
00788     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(UTIMER_IF);
00789     if (HAL_REG_READ_BIT(*addr, UTIMER_IF_T0_ZC_IF_BIT << (timer_x * 3)))
00790     {
00791         return 1;
00792     }
00793     else
00794     {
00795         return 0;
00796     }
00797 }
00798
00803 void HAL_TIMER_IrqOverflowEnable(HAL_TIMER_x timer_x)
00804 {
00805     volatile uint32_t *addr;
00806     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(UTIMER_IE);
00807     HAL_REG_SET(*addr, UTIMER_IE_T0_ZC_IE_BIT << (timer_x * 3));
00808 }
00809
00814 void HAL_TIMER_IrqOverflowDisable(HAL_TIMER_x timer_x)
00815 {
00816     volatile uint32_t *addr;
00817     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(UTIMER_IE);
00818     HAL_REG_RESET(*addr, UTIMER_IE_T0_ZC_IE_BIT << (timer_x * 3));
00819 }
00820
00825 void HAL_TIMER_IrqOverflowClear(HAL_TIMER_x timer_x)
00826 {
00827     volatile uint32_t *addr;
00828     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(UTIMER_IF);
00829     HAL_REG_WRITE(*addr, UTIMER_IF_T0_ZC_IF_BIT << (timer_x * 3));
00830 }
00831
00837 uint8_t HAL_TIMER_IrqCompare0(HAL_TIMER_x timer_x)
00838 {
00839     volatile uint32_t *addr_if;
00840     volatile uint32_t *addr_ie;
00841     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(UTIMER_IE);
00842     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(UTIMER_IF);
00843     if ((HAL_REG_READ_BIT(*addr_ie, UTIMER_IE_T0_CH0_IE_BIT << (timer_x * 3))) &&
        (HAL_REG_READ_BIT(*addr_if, UTIMER_IF_T0_CH0_IF_BIT << (timer_x * 3))))
00844     {
00845         return 1;
00846     }
00847     else
00848     {
00849         return 0;
00850     }
00851 }
00852
00858 uint8_t HAL_TIMER_IrqCompare0Flg(HAL_TIMER_x timer_x)
00859 {
00860     volatile uint32_t *addr;
00861     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(UTIMER_IF);
00862     if (HAL_REG_READ_BIT(*addr, UTIMER_IF_T0_CH0_IF_BIT << (timer_x * 3)))
00863     {
00864         return 1;
00865     }
00866     else
00867     {
00868         return 0;
00869     }
00870 }
00871
00876 void HAL_TIMER_IrqCompare0Enable(HAL_TIMER_x timer_x)
00877 {
00878     volatile uint32_t *addr;
00879     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(UTIMER_IE);
00880     HAL_REG_SET(*addr, UTIMER_IE_T0_CH0_IE_BIT << (timer_x * 3));
00881 }
00882

```

```

00887 void HAL_TIMER_IrqCompare0Disable(HAL_TIMER_x timer_x)
00888 {
00889     volatile uint32_t *addr;
00890     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(UTIMER_IE);
00891     HAL_REG_RESET(*addr, UTIMER_IE.T0.CH0.IE.BIT << (timer_x * 3));
00892 }
00893
00898 void HAL_TIMER_IrqCompare0Clear(HAL_TIMER_x timer_x)
00899 {
00900     volatile uint32_t *addr;
00901     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(UTIMER_IF);
00902     HAL_REG_WRITE(*addr, UTIMER_IF.T0.CH0.IF.BIT << (timer_x * 3));
00903 }
00904
00910 uint8_t HAL_TIMER_IrqCompare1(HAL_TIMER_x timer_x)
00911 {
00912     volatile uint32_t *addr_if;
00913     volatile uint32_t *addr_ie;
00914     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(UTIMER_IE);
00915     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(UTIMER_IF);
00916     if ((HAL_REG_READ_BIT(*addr_ie, UTIMER_IE.T0.CH1.IE.BIT << (timer_x * 3))) &&
        (HAL_REG_READ_BIT(*addr_if, UTIMER_IF.T0.CH1.IF.BIT << (timer_x * 3))))
00917     {
00918         return 1;
00919     }
00920     else
00921     {
00922         return 0;
00923     }
00924 }
00925
00931 uint8_t HAL_TIMER_IrqCompare1Flg(HAL_TIMER_x timer_x)
00932 {
00933     volatile uint32_t *addr;
00934     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(UTIMER_IF);
00935     if (HAL_REG_READ_BIT(*addr, UTIMER_IF.T0.CH1.IF.BIT << (timer_x * 3)))
00936     {
00937         return 1;
00938     }
00939     else
00940     {
00941         return 0;
00942     }
00943 }
00944
00949 void HAL_TIMER_IrqCompare1Enable(HAL_TIMER_x timer_x)
00950 {
00951     volatile uint32_t *addr;
00952     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(UTIMER_IE);
00953     HAL_REG_SET(*addr, UTIMER_IE.T0.CH1.IE.BIT << (timer_x * 3));
00954 }
00955
00960 void HAL_TIMER_IrqCompare1Disable(HAL_TIMER_x timer_x)
00961 {
00962     volatile uint32_t *addr;
00963     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(UTIMER_IE);
00964     HAL_REG_RESET(*addr, UTIMER_IE.T0.CH1.IE.BIT << (timer_x * 3));
00965 }
00966
00971 void HAL_TIMER_IrqCompare1Clear(HAL_TIMER_x timer_x)
00972 {
00973     volatile uint32_t *addr;
00974     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(UTIMER_IF);
00975     HAL_REG_WRITE(*addr, UTIMER_IF.T0.CH1.IF.BIT << (timer_x * 3));
00976 }
00977
00983 uint8_t HAL_UART_IrqTxDone(HAL_UART_x uart_x)
00984 {
00985     volatile uint32_t *addr_if;
00986     volatile uint32_t *addr_ie;
00987     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_UART[uart_x]->IE);
00988     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_UART[uart_x]->IF);
00989     if ((HAL_REG_READ_BIT(*addr_ie, UARTx.IE.TX.DONE.IE.BIT)) && (HAL_REG_READ_BIT(*addr_if,
        UARTx.IF.TX.DONE.IF.BIT)))
00990     {
00991         return 1;
00992     }
00993     else
00994     {
00995         return 0;
00996     }
00997 }
00998
01004 uint8_t HAL_UART_IrqTxDoneFlg(HAL_UART_x uart_x)
01005 {
01006     volatile uint32_t *addr;
01007     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_UART[uart_x]->IF);

```

```

01008     if (HAL_REG_READ_BIT(*addr, UARTx_IF_TX_DONE_IF_BIT))
01009     {
01010         return 1;
01011     }
01012     else
01013     {
01014         return 0;
01015     }
01016 }
01017
01022 void HAL_UART_IrqTxDoneEnable(HAL_UART_x uart_x)
01023 {
01024     volatile uint32_t *addr;
01025     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_UART[uart_x]->IE);
01026     HAL_REG_SET(*addr, UARTx_IE_TX_DONE_IE_BIT);
01027 }
01028
01033 void HAL_UART_IrqTxDoneDisable(HAL_UART_x uart_x)
01034 {
01035     volatile uint32_t *addr;
01036     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_UART[uart_x]->IE);
01037     HAL_REG_RESET(*addr, UARTx_IE_TX_DONE_IE_BIT);
01038 }
01039
01044 void HAL_UART_IrqTxDoneClear(HAL_UART_x uart_x)
01045 {
01046     volatile uint32_t *addr;
01047     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_UART[uart_x]->IF);
01048     HAL_REG_WRITE(*addr, UARTx_IF_TX_DONE_IF_BIT);
01049 }
01050
01056 uint8_t HAL_UART_IrqRxDone(HAL_UART_x uart_x)
01057 {
01058     volatile uint32_t *addr_if;
01059     volatile uint32_t *addr_ie;
01060     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_UART[uart_x]->IE);
01061     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_UART[uart_x]->IF);
01062     if ((HAL_REG_READ_BIT(*addr_ie, UARTx_IE_RX_DONE_IE_BIT)) && (HAL_REG_READ_BIT(*addr_if,
UARTx_IF_RX_DONE_IF_BIT)))
01063     {
01064         return 1;
01065     }
01066     else
01067     {
01068         return 0;
01069     }
01070 }
01071
01077 uint8_t HAL_UART_IrqRxDoneFlg(HAL_UART_x uart_x)
01078 {
01079     volatile uint32_t *addr;
01080     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_UART[uart_x]->IF);
01081     if (HAL_REG_READ_BIT(*addr, UARTx_IF_RX_DONE_IF_BIT))
01082     {
01083         return 1;
01084     }
01085     else
01086     {
01087         return 0;
01088     }
01089 }
01090
01095 void HAL_UART_IrqRxDoneEnable(HAL_UART_x uart_x)
01096 {
01097     volatile uint32_t *addr;
01098     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_UART[uart_x]->IE);
01099     HAL_REG_SET(*addr, UARTx_IE_RX_DONE_IE_BIT);
01100 }
01101
01106 void HAL_UART_IrqRxDoneDisable(HAL_UART_x uart_x)
01107 {
01108     volatile uint32_t *addr;
01109     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_UART[uart_x]->IE);
01110     HAL_REG_RESET(*addr, UARTx_IE_RX_DONE_IE_BIT);
01111 }
01112
01117 void HAL_UART_IrqRxDoneClear(HAL_UART_x uart_x)
01118 {
01119     volatile uint32_t *addr;
01120     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_UART[uart_x]->IF);
01121     HAL_REG_WRITE(*addr, UARTx_IF_RX_DONE_IF_BIT);
01122 }
01123
01129 uint8_t HAL_UART_IrqStopErr(HAL_UART_x uart_x)
01130 {
01131     volatile uint32_t *addr_if;
01132     volatile uint32_t *addr_ie;

```

```

01133     addr.ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_UART[uart_x]->IE);
01134     addr.if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_UART[uart_x]->IF);
01135     if ((HAL_REG_READ_BIT(*addr.ie, UARTx.IE_STOP_ERR_IE_BIT)) && (HAL_REG_READ_BIT(*addr.if,
UARTx.IF_STOP_ERR_IF_BIT)))
01136     {
01137         return 1;
01138     }
01139     else
01140     {
01141         return 0;
01142     }
01143 }
01144
01150 uint8_t HAL_UART_IrqStopErrFlg(HAL_UART_x uart_x)
01151 {
01152     volatile uint32_t *addr;
01153     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_UART[uart_x]->IF);
01154     if (HAL_REG_READ_BIT(*addr, UARTx.IF_STOP_ERR_IF_BIT))
01155     {
01156         return 1;
01157     }
01158     else
01159     {
01160         return 0;
01161     }
01162 }
01163
01168 void HAL_UART_IrqStopErrEnable(HAL_UART_x uart_x)
01169 {
01170     volatile uint32_t *addr;
01171     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_UART[uart_x]->IE);
01172     HAL_REG_SET(*addr, UARTx.IE_STOP_ERR_IE_BIT);
01173 }
01174
01179 void HAL_UART_IrqStopErrDisable(HAL_UART_x uart_x)
01180 {
01181     volatile uint32_t *addr;
01182     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_UART[uart_x]->IE);
01183     HAL_REG_RESET(*addr, UARTx.IE_STOP_ERR_IE_BIT);
01184 }
01185
01190 void HAL_UART_IrqStopErrClear(HAL_UART_x uart_x)
01191 {
01192     volatile uint32_t *addr;
01193     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_UART[uart_x]->IF);
01194     HAL_REG_WRITE(*addr, UARTx.IF_STOP_ERR_IF_BIT);
01195 }
01196
01202 uint8_t HAL_UART_IrqCheckErr(HAL_UART_x uart_x)
01203 {
01204     volatile uint32_t *addr.if;
01205     volatile uint32_t *addr.ie;
01206     addr.ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_UART[uart_x]->IE);
01207     addr.if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_UART[uart_x]->IF);
01208     if ((HAL_REG_READ_BIT(*addr.ie, UARTx.IE_CHK_ERR_IE_BIT)) && (HAL_REG_READ_BIT(*addr.if,
UARTx.IF_CHK_ERR_IF_BIT)))
01209     {
01210         return 1;
01211     }
01212     else
01213     {
01214         return 0;
01215     }
01216 }
01217
01223 uint8_t HAL_UART_IrqCheckErrFlg(HAL_UART_x uart_x)
01224 {
01225     volatile uint32_t *addr;
01226     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_UART[uart_x]->IF);
01227     if (HAL_REG_READ_BIT(*addr, UARTx.IF_CHK_ERR_IF_BIT))
01228     {
01229         return 1;
01230     }
01231     else
01232     {
01233         return 0;
01234     }
01235 }
01236
01241 void HAL_UART_IrqCheckErrEnable(HAL_UART_x uart_x)
01242 {
01243     volatile uint32_t *addr;
01244     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_UART[uart_x]->IE);
01245     HAL_REG_SET(*addr, UARTx.IE_CHK_ERR_IE_BIT);
01246 }
01247
01252 void HAL_UART_IrqCheckErrDisable(HAL_UART_x uart_x)

```

```

01253 {
01254     volatile uint32_t *addr;
01255     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_UART[uart_x]->IE);
01256     HAL_REG_RESET(*addr, UARTx_IE_CHK_ERR_IE_BIT);
01257 }
01258
01263 void HAL_UART_IrqCheckErrClear(HAL_UART_x uart_x)
01264 {
01265     volatile uint32_t *addr;
01266     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_UART[uart_x]->IF);
01267     HAL_REG_WRITE(*addr, UARTx_IF_CHK_ERR_IF_BIT);
01268 }
01269
01275 uint8_t HAL_CAN_IrqRxDone(HAL_CAN_x can_x)
01276 {
01277     volatile uint32_t *addr_if;
01278     volatile uint32_t *addr_ie;
01279     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_CAN[can_x]->IER);
01280     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_CAN[can_x]->IR);
01281     if ((HAL_REG_READ_BIT(*addr_ie, CAN_IER_TX_DONE_IE_BIT) && (HAL_REG_READ_BIT(*addr_if,
CAN_IR_TX_DONE_IF_BIT))))
01282     {
01283         return 1;
01284     }
01285     else
01286     {
01287         return 0;
01288     }
01289 }
01290
01296 uint8_t HAL_CAN_IrqRxDoneFlg(HAL_CAN_x can_x)
01297 {
01298     volatile uint32_t *addr;
01299     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_CAN[can_x]->IR);
01300     if (HAL_REG_READ_BIT(*addr, CAN_IR_TX_DONE_IF_BIT))
01301     {
01302         return 1;
01303     }
01304     else
01305     {
01306         return 0;
01307     }
01308 }
01309
01314 void HAL_CAN_IrqRxDoneEnable(HAL_CAN_x can_x)
01315 {
01316     volatile uint32_t *addr;
01317     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_CAN[can_x]->IER);
01318     HAL_REG_SET(*addr, CAN_IER_TX_DONE_IE_BIT);
01319 }
01320
01325 void HAL_CAN_IrqRxDoneDisable(HAL_CAN_x can_x)
01326 {
01327     volatile uint32_t *addr;
01328     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_CAN[can_x]->IER);
01329     HAL_REG_RESET(*addr, CAN_IER_TX_DONE_IE_BIT);
01330 }
01331
01336 void HAL_CAN_IrqRxDoneClear(HAL_CAN_x can_x)
01337 {
01338     volatile uint32_t *addr;
01339     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_CAN[can_x]->IR);
01340     HAL_REG_WRITE(*addr, CAN_IR_TX_DONE_IF_BIT);
01341 }
01342
01348 uint8_t HAL_CAN_IrqTxDone(HAL_CAN_x can_x)
01349 {
01350     volatile uint32_t *addr_if;
01351     volatile uint32_t *addr_ie;
01352     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_CAN[can_x]->IER);
01353     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_CAN[can_x]->IR);
01354     if ((HAL_REG_READ_BIT(*addr_ie, CAN_IER_RFIFO_N_EMPTY_IE_BIT) && (HAL_REG_READ_BIT(*addr_if,
CAN_IR_RFIFO_N_EMPTY_IF_BIT))))
01355     {
01356         return 1;
01357     }
01358     else
01359     {
01360         return 0;
01361     }
01362 }
01363
01369 uint8_t HAL_CAN_IrqTxDoneFlg(HAL_CAN_x can_x)
01370 {
01371     volatile uint32_t *addr;
01372     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_CAN[can_x]->IR);
01373     if (HAL_REG_READ_BIT(*addr, CAN_IR_RFIFO_N_EMPTY_IF_BIT))

```

```

01374     {
01375         return 1;
01376     }
01377     else
01378     {
01379         return 0;
01380     }
01381 }
01382
01387 void HAL_CAN_IrqTxDoneEnable(HAL_CAN_x can_x)
01388 {
01389     volatile uint32_t *addr;
01390     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_CAN[can_x]->IER);
01391     HAL_REG_SET(*addr, CAN_IER_RFIFO_N_EMPTY_IE_BIT);
01392 }
01393
01398 void HAL_CAN_IrqTxDoneDisable(HAL_CAN_x can_x)
01399 {
01400     volatile uint32_t *addr;
01401     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_CAN[can_x]->IER);
01402     HAL_REG_RESET(*addr, CAN_IER_RFIFO_N_EMPTY_IE_BIT);
01403 }
01404
01409 void HAL_CAN_IrqTxDoneClear(HAL_CAN_x can_x)
01410 {
01411     volatile uint32_t *addr;
01412     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_CAN[can_x]->IR);
01413     HAL_REG_WRITE(*addr, CAN_IR_RFIFO_N_EMPTY_IF_BIT);
01414 }
01415
01421 uint8_t HAL_CAN_IrqAll(HAL_CAN_x can_x)
01422 {
01423     volatile uint32_t *addr_if;
01424     volatile uint32_t *addr_ie;
01425     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_CAN[can_x]->IER);
01426     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_CAN[can_x]->IR);
01427     if ((HAL_REG_READ_BIT(*addr_ie, CAN_IER_BUS_ERR_IE_BIT | CAN_IER_LOST_ARB_IE_BIT | CAN_IER_M_ERR_IE_BIT |
CAN_IER_WK_IE_BIT | CAN_IER_RFIFO_OV_IE_BIT | CAN_IER_W_ERR_IE_BIT | CAN_IER_TX_DONE_IE_BIT |
CAN_IER_RFIFO_N_EMPTY_IE_BIT)) && (HAL_REG_READ_BIT(*addr_if, CAN_IR_BUS_ERR_IF_BIT | CAN_IR_LOST_ARB_IF_BIT |
CAN_IR_M_ERR_IF_BIT | CAN_IR_WK_IF_BIT | CAN_IR_RFIFO_OV_IF_BIT | CAN_IR_W_ERR_IF_BIT | CAN_IR_TX_DONE_IF_BIT |
CAN_IR_RFIFO_N_EMPTY_IF_BIT)))
01428     {
01429         return 1;
01430     }
01431     else
01432     {
01433         return 0;
01434     }
01435 }
01436
01442 uint8_t HAL_CAN_IrqAllFlg(HAL_CAN_x can_x)
01443 {
01444     volatile uint32_t *addr;
01445     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_CAN[can_x]->IR);
01446     if (HAL_REG_READ_BIT(*addr, CAN_IR_BUS_ERR_IF_BIT | CAN_IR_LOST_ARB_IF_BIT | CAN_IR_M_ERR_IF_BIT |
CAN_IR_WK_IF_BIT | CAN_IR_RFIFO_OV_IF_BIT | CAN_IR_W_ERR_IF_BIT | CAN_IR_TX_DONE_IF_BIT |
CAN_IR_RFIFO_N_EMPTY_IF_BIT))
01447     {
01448         return 1;
01449     }
01450     else
01451     {
01452         return 0;
01453     }
01454 }
01455
01460 void HAL_CAN_IrqAllEnable(HAL_CAN_x can_x)
01461 {
01462     volatile uint32_t *addr;
01463     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_CAN[can_x]->IER);
01464     HAL_REG_SET(*addr, CAN_IER_BUS_ERR_IE_BIT | CAN_IER_LOST_ARB_IE_BIT | CAN_IER_M_ERR_IE_BIT |
CAN_IER_WK_IE_BIT | CAN_IER_RFIFO_OV_IE_BIT | CAN_IER_W_ERR_IE_BIT | CAN_IER_TX_DONE_IE_BIT |
CAN_IER_RFIFO_N_EMPTY_IE_BIT);
01465 }
01466
01471 void HAL_CAN_IrqAllDisable(HAL_CAN_x can_x)
01472 {
01473     volatile uint32_t *addr;
01474     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_CAN[can_x]->IER);
01475     HAL_REG_RESET(*addr, CAN_IER_BUS_ERR_IE_BIT | CAN_IER_LOST_ARB_IE_BIT | CAN_IER_M_ERR_IE_BIT |
CAN_IER_WK_IE_BIT | CAN_IER_RFIFO_OV_IE_BIT | CAN_IER_W_ERR_IE_BIT | CAN_IER_TX_DONE_IE_BIT |
CAN_IER_RFIFO_N_EMPTY_IE_BIT);
01476 }
01477
01482 void HAL_CAN_IrqAllClear(HAL_CAN_x can_x)
01483 {
01484     volatile uint32_t *addr;

```



```

01485     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_CAN[can_x]->IR);
01486     HAL_REG_WRITE(*addr, CAN_IR_BUS_ERR_IF_BIT | CAN_IR_LOST_ARB_IF_BIT | CAN_IR_M_ERR_IF_BIT |
CAN_IR_WK_IF_BIT | CAN_IR_RFIFO_OV_IF_BIT | CAN_IR_W_ERR_IF_BIT | CAN_IR_TX_DONE_IF_BIT |
CAN_IR_RFIFO_N_EMPTY_IF_BIT);
01487 }
01488
01494 uint8_t HAL_UART_IrqTxBufEmpty(HAL_UART_x uart_x)
01495 {
01496     volatile uint32_t *addr_if;
01497     volatile uint32_t *addr_ie;
01498     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_UART[uart_x]->IE);
01499     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_UART[uart_x]->IF);
01500     if ((HAL_REG_READ_BIT(*addr_ie, UARTx_IE_TX_BUF_EMPTY_IE_BIT)) && (HAL_REG_READ_BIT(*addr_if,
UARTx_IF_TX_BUF_EMPTY_IF_BIT)))
01501     {
01502         return 1;
01503     }
01504     else
01505     {
01506         return 0;
01507     }
01508 }
01509
01515 uint8_t HAL_UART_IrqTxBufEmptyFlg(HAL_UART_x uart_x)
01516 {
01517     volatile uint32_t *addr;
01518     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_UART[uart_x]->IF);
01519     if (HAL_REG_READ_BIT(*addr, UARTx_IF_TX_BUF_EMPTY_IF_BIT))
01520     {
01521         return 1;
01522     }
01523     else
01524     {
01525         return 0;
01526     }
01527 }
01528
01533 void HAL_UART_IrqTxBufEmptyEnable(HAL_UART_x uart_x)
01534 {
01535     volatile uint32_t *addr;
01536     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_UART[uart_x]->IE);
01537     HAL_REG_SET(*addr, UARTx_IE_TX_BUF_EMPTY_IE_BIT);
01538 }
01539
01544 void HAL_UART_IrqTxBufEmptyDisable(HAL_UART_x uart_x)
01545 {
01546     volatile uint32_t *addr;
01547     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_UART[uart_x]->IE);
01548     HAL_REG_RESET(*addr, UARTx_IE_TX_BUF_EMPTY_IE_BIT);
01549 }
01550
01555 void HAL_UART_IrqTxBufEmptyClear(HAL_UART_x uart_x)
01556 {
01557     volatile uint32_t *addr;
01558     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_UART[uart_x]->IF);
01559     HAL_REG_WRITE(*addr, UARTx_IF_TX_BUF_EMPTY_IF_BIT);
01560 }
01561
01567 uint8_t HAL_ADC_IrqSamp3(HAL_ADC_x adc)
01568 {
01569     volatile uint32_t *addr_if;
01570     volatile uint32_t *addr_ie;
01571     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IE);
01572     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IF);
01573     if ((HAL_REG_READ_BIT(*addr_ie, ADC0_IE_S3FINISH_IE_BIT)) && (HAL_REG_READ_BIT(*addr_if,
ADC0_IF_S3FINISH_IF_BIT)))
01574     {
01575         return 1;
01576     }
01577     else
01578     {
01579         return 0;
01580     }
01581 }
01582
01588 uint8_t HAL_ADC_IrqSamp3Flg(HAL_ADC_x adc)
01589 {
01590     volatile uint32_t *addr;
01591     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IF);
01592     if (HAL_REG_READ_BIT(*addr, ADC0_IF_S3FINISH_IF_BIT))
01593     {
01594         return 1;
01595     }
01596     else
01597     {
01598         return 0;
01599     }

```

```

01600 }
01601
01606 void HAL_ADC_IrqSamp3Enable(HAL_ADC_x adc)
01607 {
01608     volatile uint32_t *addr;
01609     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IE);
01610     HAL_REG_SET(*addr, ADC0_IE_S3FINISH_IE_BIT);
01611 }
01612
01617 void HAL_ADC_IrqSamp3Disable(HAL_ADC_x adc)
01618 {
01619     volatile uint32_t *addr;
01620     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IE);
01621     HAL_REG_RESET(*addr, ADC0_IE_S3FINISH_IE_BIT);
01622 }
01623
01628 void HAL_ADC_IrqSamp3Clear(HAL_ADC_x adc)
01629 {
01630     volatile uint32_t *addr;
01631     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IF);
01632     HAL_REG_WRITE(*addr, ADC0_IF_S3FINISH_IF_BIT);
01633 }
01634
01640 uint8_t HAL_ADC_IrqSamp4(HAL_ADC_x adc)
01641 {
01642     volatile uint32_t *addr_if;
01643     volatile uint32_t *addr_ie;
01644     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IE);
01645     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IF);
01646     if ((HAL_REG_READ_BIT(*addr_ie, ADC0_IE_S4FINISH_IE_BIT) && (HAL_REG_READ_BIT(*addr_if,
ADC0_IF_S4FINISH_IF_BIT))))
01647     {
01648         return 1;
01649     }
01650     else
01651     {
01652         return 0;
01653     }
01654 }
01655
01661 uint8_t HAL_ADC_IrqSamp4Flg(HAL_ADC_x adc)
01662 {
01663     volatile uint32_t *addr;
01664     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IF);
01665     if (HAL_REG_READ_BIT(*addr, ADC0_IF_S4FINISH_IF_BIT))
01666     {
01667         return 1;
01668     }
01669     else
01670     {
01671         return 0;
01672     }
01673 }
01674
01679 void HAL_ADC_IrqSamp4Enable(HAL_ADC_x adc)
01680 {
01681     volatile uint32_t *addr;
01682     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IE);
01683     HAL_REG_SET(*addr, ADC0_IE_S4FINISH_IE_BIT);
01684 }
01685
01690 void HAL_ADC_IrqSamp4Disable(HAL_ADC_x adc)
01691 {
01692     volatile uint32_t *addr;
01693     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IE);
01694     HAL_REG_RESET(*addr, ADC0_IE_S4FINISH_IE_BIT);
01695 }
01696
01701 void HAL_ADC_IrqSamp4Clear(HAL_ADC_x adc)
01702 {
01703     volatile uint32_t *addr;
01704     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IF);
01705     HAL_REG_WRITE(*addr, ADC0_IF_S4FINISH_IF_BIT);
01706 }
01707
01713 uint8_t HAL_ADC_IrqWatchdog(HAL_ADC_x adc)
01714 {
01715     volatile uint32_t *addr_if;
01716     volatile uint32_t *addr_ie;
01717     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IE);
01718     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IF);
01719     if ((HAL_REG_READ_BIT(*addr_ie, ADC0_IE_TH_IE_BIT) && (HAL_REG_READ_BIT(*addr_if, ADC0_IF_TH_IF_BIT))))
01720     {
01721         return 1;
01722     }
01723     else
01724     {

```

```

01725         return 0;
01726     }
01727 }
01728
01734 uint8_t HAL_ADC_IrqWatchdogFlg(HAL_ADC_x adc)
01735 {
01736     volatile uint32_t *addr;
01737     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IF);
01738     if (HAL_REG_READ_BIT(*addr, ADC0_IF_TH_IF_BIT))
01739     {
01740         return 1;
01741     }
01742     else
01743     {
01744         return 0;
01745     }
01746 }
01747
01752 void HAL_ADC_IrqWatchdogEnable(HAL_ADC_x adc)
01753 {
01754     volatile uint32_t *addr;
01755     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IE);
01756     HAL_REG_SET(*addr, ADC0_IE_TH_IE_BIT);
01757 }
01758
01763 void HAL_ADC_IrqWatchdogDisable(HAL_ADC_x adc)
01764 {
01765     volatile uint32_t *addr;
01766     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IE);
01767     HAL_REG_RESET(*addr, ADC0_IE_TH_IE_BIT);
01768 }
01769
01774 void HAL_ADC_IrqWatchdogClear(HAL_ADC_x adc)
01775 {
01776     volatile uint32_t *addr;
01777     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IF);
01778     HAL_REG_WRITE(*addr, ADC0_IF_TH_IF_BIT);
01779 }
01780
01786 uint8_t HAL_ADC_IrqHardErr(HAL_ADC_x adc)
01787 {
01788     volatile uint32_t *addr_if;
01789     volatile uint32_t *addr_ie;
01790     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IE);
01791     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IF);
01792     if ((HAL_REG_READ_BIT(*addr_ie, ADC0_IE_HERR_IE_BIT)) && (HAL_REG_READ_BIT(*addr_if,
01793     ADC0_IF_HERR_IF_BIT)))
01794     {
01795         return 1;
01796     }
01797     else
01798     {
01799         return 0;
01800     }
01801 }
01807 uint8_t HAL_ADC_IrqHardErrFlg(HAL_ADC_x adc)
01808 {
01809     volatile uint32_t *addr;
01810     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IF);
01811     if (HAL_REG_READ_BIT(*addr, ADC0_IF_HERR_IF_BIT))
01812     {
01813         return 1;
01814     }
01815     else
01816     {
01817         return 0;
01818     }
01819 }
01820
01825 void HAL_ADC_IrqHardErrEnable(HAL_ADC_x adc)
01826 {
01827     volatile uint32_t *addr;
01828     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IE);
01829     HAL_REG_SET(*addr, ADC0_IE_HERR_IE_BIT);
01830 }
01831
01836 void HAL_ADC_IrqHardErrDisable(HAL_ADC_x adc)
01837 {
01838     volatile uint32_t *addr;
01839     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IE);
01840     HAL_REG_RESET(*addr, ADC0_IE_HERR_IE_BIT);
01841 }
01842
01847 void HAL_ADC_IrqHardErrClear(HAL_ADC_x adc)
01848 {
01849     volatile uint32_t *addr;

```

```

01850     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IF);
01851     HAL_REG_WRITE(*addr, ADC0_IF_HERR_IF_BIT);
01852 }
01853
01859 uint8_t HAL_ADC_IrqSoftErr(HAL_ADC_x adc)
01860 {
01861     volatile uint32_t *addr_if;
01862     volatile uint32_t *addr_ie;
01863     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IE);
01864     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IF);
01865     if ((HAL_REG_READ_BIT(*addr_ie, ADC0_IE_SERR_IE_BIT)) && (HAL_REG_READ_BIT(*addr_if,
ADC0_IF_SERR_IF_BIT)))
01866     {
01867         return 1;
01868     }
01869     else
01870     {
01871         return 0;
01872     }
01873 }
01874
01880 uint8_t HAL_ADC_IrqSoftErrFlg(HAL_ADC_x adc)
01881 {
01882     volatile uint32_t *addr;
01883     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IF);
01884     if (HAL_REG_READ_BIT(*addr, ADC0_IF_SERR_IF_BIT))
01885     {
01886         return 1;
01887     }
01888     else
01889     {
01890         return 0;
01891     }
01892 }
01893
01898 void HAL_ADC_IrqSoftErrEnable(HAL_ADC_x adc)
01899 {
01900     volatile uint32_t *addr;
01901     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IE);
01902     HAL_REG_SET(*addr, ADC0_IE_SERR_IE_BIT);
01903 }
01904
01909 void HAL_ADC_IrqSoftErrDisable(HAL_ADC_x adc)
01910 {
01911     volatile uint32_t *addr;
01912     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IE);
01913     HAL_REG_RESET(*addr, ADC0_IE_SERR_IE_BIT);
01914 }
01915
01920 void HAL_ADC_IrqSoftErrClear(HAL_ADC_x adc)
01921 {
01922     volatile uint32_t *addr;
01923     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_ADC[adc]->IF);
01924     HAL_REG_WRITE(*addr, ADC0_IF_SERR_IF_BIT);
01925 }
01926
01932 uint8_t HAL_MCPWM_IrqUpdate(HAL_MCPWM_x mcpwmx)
01933 {
01934     volatile uint32_t *addr_if;
01935     volatile uint32_t *addr_ie;
01936     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
01937     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
01938     if ((HAL_REG_READ_BIT(*addr_ie, MCPWM_IE_SHADE_IE_BIT)) && (HAL_REG_READ_BIT(*addr_if,
MCPWM_IF_SHADE_IF_BIT)))
01939     {
01940         return 1;
01941     }
01942     else
01943     {
01944         return 0;
01945     }
01946 }
01947
01953 uint8_t HAL_MCPWM_IrqUpdateFlg(HAL_MCPWM_x mcpwmx)
01954 {
01955     volatile uint32_t *addr;
01956     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
01957     if (HAL_REG_READ_BIT(*addr, MCPWM_IF_SHADE_IF_BIT))
01958     {
01959         return 1;
01960     }
01961     else
01962     {
01963         return 0;
01964     }
01965 }
01966

```

```

01971 void HAL_MCPWM_IrqUpdateEnable(HAL_MCPWM_x mcpwmx)
01972 {
01973     volatile uint32_t *addr;
01974     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
01975     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
01976     HAL_REG_SET(*addr, MCPWM_IE_SHADE_IE_BIT);
01977     HAL_REG_WRITE(MCPWM_PRT, 0);
01978 }
01979
01984 void HAL_MCPWM_IrqUpdateDisable(HAL_MCPWM_x mcpwmx)
01985 {
01986     volatile uint32_t *addr;
01987     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
01988     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
01989     HAL_REG_RESET(*addr, MCPWM_IE_SHADE_IE_BIT);
01990     HAL_REG_WRITE(MCPWM_PRT, 0);
01991 }
01992
01997 void HAL_MCPWM_IrqUpdateClear(HAL_MCPWM_x mcpwmx)
01998 {
01999     volatile uint32_t *addr;
02000     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
02001     HAL_REG_WRITE(*addr, MCPWM_IF_SHADE_IF_BIT);
02002 }
02003
02009 uint8_t HAL_MCPWM_IrqTmr0(HAL_MCPWM_x mcpwmx)
02010 {
02011     volatile uint32_t *addr_if;
02012     volatile uint32_t *addr_ie;
02013     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
02014     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
02015     if ((HAL_REG_READ_BIT(*addr_ie, MCPWM_IE_TMR0_IE_BIT)) && (HAL_REG_READ_BIT(*addr_if,
MCPWM_IF_TMR0_IF_BIT)))
02016     {
02017         return 1;
02018     }
02019     else
02020     {
02021         return 0;
02022     }
02023 }
02024
02030 uint8_t HAL_MCPWM_IrqTmr0Flg(HAL_MCPWM_x mcpwmx)
02031 {
02032     volatile uint32_t *addr;
02033     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
02034     if (HAL_REG_READ_BIT(*addr, MCPWM_IF_TMR0_IF_BIT))
02035     {
02036         return 1;
02037     }
02038     else
02039     {
02040         return 0;
02041     }
02042 }
02043
02048 void HAL_MCPWM_IrqTmr0Enable(HAL_MCPWM_x mcpwmx)
02049 {
02050     volatile uint32_t *addr;
02051     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
02052     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
02053     HAL_REG_SET(*addr, MCPWM_IE_TMR0_IE_BIT);
02054     HAL_REG_WRITE(MCPWM_PRT, 0);
02055 }
02056
02061 void HAL_MCPWM_IrqTmr0Disable(HAL_MCPWM_x mcpwmx)
02062 {
02063     volatile uint32_t *addr;
02064     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
02065     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
02066     HAL_REG_RESET(*addr, MCPWM_IE_TMR0_IE_BIT);
02067     HAL_REG_WRITE(MCPWM_PRT, 0);
02068 }
02069
02074 void HAL_MCPWM_IrqTmr0Clear(HAL_MCPWM_x mcpwmx)
02075 {
02076     volatile uint32_t *addr;
02077     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
02078     HAL_REG_WRITE(*addr, MCPWM_IF_TMR0_IF_BIT);
02079 }
02080
02086 uint8_t HAL_MCPWM_IrqTmr1(HAL_MCPWM_x mcpwmx)
02087 {
02088     volatile uint32_t *addr_if;
02089     volatile uint32_t *addr_ie;
02090     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
02091     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);

```

```

02092     if ((HAL_REG_READ_BIT(*addr_ie, MCPWM_IE_TMR1_IE_BIT)) && (HAL_REG_READ_BIT(*addr_if,
MCPWM_IF_TMR1_IF_BIT)))
02093     {
02094         return 1;
02095     }
02096     else
02097     {
02098         return 0;
02099     }
02100 }
02101
02107 uint8_t HAL_MCPWM_IrqTmr1Flg(HAL_MCPWM_x mcpwmx)
02108 {
02109     volatile uint32_t *addr;
02110     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
02111     if (HAL_REG_READ_BIT(*addr, MCPWM_IF_TMR1_IF_BIT))
02112     {
02113         return 1;
02114     }
02115     else
02116     {
02117         return 0;
02118     }
02119 }
02120
02125 void HAL_MCPWM_IrqTmr1Enable(HAL_MCPWM_x mcpwmx)
02126 {
02127     volatile uint32_t *addr;
02128     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
02129     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
02130     HAL_REG_SET(*addr, MCPWM_IE_TMR1_IE_BIT);
02131     HAL_REG_WRITE(MCPWM_PRT, 0);
02132 }
02133
02138 void HAL_MCPWM_IrqTmr1Disable(HAL_MCPWM_x mcpwmx)
02139 {
02140     volatile uint32_t *addr;
02141     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
02142     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
02143     HAL_REG_RESET(*addr, MCPWM_IE_TMR1_IE_BIT);
02144     HAL_REG_WRITE(MCPWM_PRT, 0);
02145 }
02146
02151 void HAL_MCPWM_IrqTmr1Clear(HAL_MCPWM_x mcpwmx)
02152 {
02153     volatile uint32_t *addr;
02154     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
02155     HAL_REG_WRITE(*addr, MCPWM_IF_TMR1_IF_BIT);
02156 }
02157
02163 uint8_t HAL_MCPWM_IrqTmr2(HAL_MCPWM_x mcpwmx)
02164 {
02165     volatile uint32_t *addr_if;
02166     volatile uint32_t *addr_ie;
02167     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
02168     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
02169     if ((HAL_REG_READ_BIT(*addr_ie, MCPWM_IE_TMR2_IE_BIT)) && (HAL_REG_READ_BIT(*addr_if,
MCPWM_IF_TMR2_IF_BIT)))
02170     {
02171         return 1;
02172     }
02173     else
02174     {
02175         return 0;
02176     }
02177 }
02178
02184 uint8_t HAL_MCPWM_IrqTmr2Flg(HAL_MCPWM_x mcpwmx)
02185 {
02186     volatile uint32_t *addr;
02187     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
02188     if (HAL_REG_READ_BIT(*addr, MCPWM_IF_TMR2_IF_BIT))
02189     {
02190         return 1;
02191     }
02192     else
02193     {
02194         return 0;
02195     }
02196 }
02197
02202 void HAL_MCPWM_IrqTmr2Enable(HAL_MCPWM_x mcpwmx)
02203 {
02204     volatile uint32_t *addr;
02205     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
02206     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
02207     HAL_REG_SET(*addr, MCPWM_IE_TMR2_IE_BIT);

```

```

02208     HAL_REG_WRITE(MCPWM_PRT, 0);
02209 }
02210
02215 void HAL_MCPWM_IrqTmr2Disable(HAL_MCPWM_x mcpwmx)
02216 {
02217     volatile uint32_t *addr;
02218     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
02219     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
02220     HAL_REG_RESET(*addr, MCPWM_IE_TMR2_IE_BIT);
02221     HAL_REG_WRITE(MCPWM_PRT, 0);
02222 }
02223
02228 void HAL_MCPWM_IrqTmr2Clear(HAL_MCPWM_x mcpwmx)
02229 {
02230     volatile uint32_t *addr;
02231     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
02232     HAL_REG_WRITE(*addr, MCPWM_IF_TMR2_IF_BIT);
02233 }
02234
02240 uint8_t HAL_MCPWM_IrqTmr3(HAL_MCPWM_x mcpwmx)
02241 {
02242     volatile uint32_t *addr_if;
02243     volatile uint32_t *addr_ie;
02244     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
02245     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
02246     if ((HAL_REG_READ_BIT(*addr_ie, MCPWM_IE_TMR3_IE_BIT)) && (HAL_REG_READ_BIT(*addr_if,
MCPWM_IF_TMR3_IF_BIT)))
02247     {
02248         return 1;
02249     }
02250     else
02251     {
02252         return 0;
02253     }
02254 }
02255
02261 uint8_t HAL_MCPWM_IrqTmr3Flg(HAL_MCPWM_x mcpwmx)
02262 {
02263     volatile uint32_t *addr;
02264     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
02265     if (HAL_REG_READ_BIT(*addr, MCPWM_IF_TMR3_IF_BIT))
02266     {
02267         return 1;
02268     }
02269     else
02270     {
02271         return 0;
02272     }
02273 }
02274
02279 void HAL_MCPWM_IrqTmr3Enable(HAL_MCPWM_x mcpwmx)
02280 {
02281     volatile uint32_t *addr;
02282     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
02283     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
02284     HAL_REG_SET(*addr, MCPWM_IE_TMR3_IE_BIT);
02285     HAL_REG_WRITE(MCPWM_PRT, 0);
02286 }
02287
02292 void HAL_MCPWM_IrqTmr3Disable(HAL_MCPWM_x mcpwmx)
02293 {
02294     volatile uint32_t *addr;
02295     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
02296     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
02297     HAL_REG_RESET(*addr, MCPWM_IE_TMR3_IE_BIT);
02298     HAL_REG_WRITE(MCPWM_PRT, 0);
02299 }
02300
02305 void HAL_MCPWM_IrqTmr3Clear(HAL_MCPWM_x mcpwmx)
02306 {
02307     volatile uint32_t *addr;
02308     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
02309     HAL_REG_WRITE(*addr, MCPWM_IF_TMR3_IF_BIT);
02310 }
02311
02317 uint8_t HAL_MCPWM_IrqTH00(HAL_MCPWM_x mcpwmx)
02318 {
02319     volatile uint32_t *addr_if;
02320     volatile uint32_t *addr_ie;
02321     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
02322     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
02323     if ((HAL_REG_READ_BIT(*addr_ie, MCPWM_IE_TH00_IE_BIT)) && (HAL_REG_READ_BIT(*addr_if,
MCPWM_IF_TH00_IF_BIT)))
02324     {
02325         return 1;
02326     }
02327     else

```

```

02328     {
02329         return 0;
02330     }
02331 }
02332
02333 uint8_t HAL_MCPWM_IrqTH00Flg(HAL_MCPWM_x mcpwm)
02334 {
02335     volatile uint32_t *addr;
02336     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwm & 0xf]->IF0);
02337     if (HAL_REG_READ_BIT(*addr, MCPWM_IF_TH00_IF_BIT))
02338     {
02339         return 1;
02340     }
02341     else
02342     {
02343         return 0;
02344     }
02345 }
02346
02347 void HAL_MCPWM_IrqTH00Enable(HAL_MCPWM_x mcpwm)
02348 {
02349     volatile uint32_t *addr;
02350     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwm & 0xf]->IE0);
02351     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
02352     HAL_REG_SET(*addr, MCPWM_IE_TH00_IE_BIT);
02353     HAL_REG_WRITE(MCPWM_PRT, 0);
02354 }
02355
02356 void HAL_MCPWM_IrqTH00Disable(HAL_MCPWM_x mcpwm)
02357 {
02358     volatile uint32_t *addr;
02359     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwm & 0xf]->IE0);
02360     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
02361     HAL_REG_RESET(*addr, MCPWM_IE_TH00_IE_BIT);
02362     HAL_REG_WRITE(MCPWM_PRT, 0);
02363 }
02364
02365 void HAL_MCPWM_IrqTH00Clear(HAL_MCPWM_x mcpwm)
02366 {
02367     volatile uint32_t *addr;
02368     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwm & 0xf]->IF0);
02369     HAL_REG_WRITE(*addr, MCPWM_IF_TH00_IF_BIT);
02370 }
02371
02372 uint8_t HAL_MCPWM_IrqTH01(HAL_MCPWM_x mcpwm)
02373 {
02374     volatile uint32_t *addr_if;
02375     volatile uint32_t *addr_ie;
02376     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwm & 0xf]->IE0);
02377     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwm & 0xf]->IF0);
02378     if ((HAL_REG_READ_BIT(*addr_ie, MCPWM_IE_TH01_IE_BIT)) && (HAL_REG_READ_BIT(*addr_if,
02379 MCPWM_IF_TH01_IF_BIT)))
02380     {
02381         return 1;
02382     }
02383     else
02384     {
02385         return 0;
02386     }
02387 }
02388
02389 uint8_t HAL_MCPWM_IrqTH01Flg(HAL_MCPWM_x mcpwm)
02390 {
02391     volatile uint32_t *addr;
02392     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwm & 0xf]->IF0);
02393     if (HAL_REG_READ_BIT(*addr, MCPWM_IF_TH01_IF_BIT))
02394     {
02395         return 1;
02396     }
02397     else
02398     {
02399         return 0;
02400     }
02401 }
02402
02403 void HAL_MCPWM_IrqTH01Enable(HAL_MCPWM_x mcpwm)
02404 {
02405     volatile uint32_t *addr;
02406     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwm & 0xf]->IE0);
02407     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
02408     HAL_REG_SET(*addr, MCPWM_IE_TH01_IE_BIT);
02409     HAL_REG_WRITE(MCPWM_PRT, 0);
02410 }
02411
02412 void HAL_MCPWM_IrqTH01Disable(HAL_MCPWM_x mcpwm)
02413 {
02414     volatile uint32_t *addr;

```



```

02449     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
02450     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
02451     HAL_REG_RESET(*addr, MCPWM_IE_TH01_IE_BIT);
02452     HAL_REG_WRITE(MCPWM_PRT, 0);
02453 }
02454
02459 void HAL_MCPWM_IrqTH01Clear(HAL_MCPWM_x mcpwmx)
02460 {
02461     volatile uint32_t *addr;
02462     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
02463     HAL_REG_WRITE(*addr, MCPWM_IF_TH01_IF_BIT);
02464 }
02465
02471 uint8_t HAL_MCPWM_IrqTH10(HAL_MCPWM_x mcpwmx)
02472 {
02473     volatile uint32_t *addr_if;
02474     volatile uint32_t *addr_ie;
02475     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
02476     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
02477     if ((HAL_REG_READ_BIT(*addr_ie, MCPWM_IE_TH10_IE_BIT)) && (HAL_REG_READ_BIT(*addr_if,
MCPWM_IF_TH10_IF_BIT)))
02478     {
02479         return 1;
02480     }
02481     else
02482     {
02483         return 0;
02484     }
02485 }
02486
02492 uint8_t HAL_MCPWM_IrqTH10Flg(HAL_MCPWM_x mcpwmx)
02493 {
02494     volatile uint32_t *addr;
02495     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
02496     if (HAL_REG_READ_BIT(*addr, MCPWM_IF_TH10_IF_BIT))
02497     {
02498         return 1;
02499     }
02500     else
02501     {
02502         return 0;
02503     }
02504 }
02505
02510 void HAL_MCPWM_IrqTH10Enable(HAL_MCPWM_x mcpwmx)
02511 {
02512     volatile uint32_t *addr;
02513     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
02514     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
02515     HAL_REG_SET(*addr, MCPWM_IE_TH10_IE_BIT);
02516     HAL_REG_WRITE(MCPWM_PRT, 0);
02517 }
02518
02523 void HAL_MCPWM_IrqTH10Disable(HAL_MCPWM_x mcpwmx)
02524 {
02525     volatile uint32_t *addr;
02526     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
02527     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
02528     HAL_REG_RESET(*addr, MCPWM_IE_TH10_IE_BIT);
02529     HAL_REG_WRITE(MCPWM_PRT, 0);
02530 }
02531
02536 void HAL_MCPWM_IrqTH10Clear(HAL_MCPWM_x mcpwmx)
02537 {
02538     volatile uint32_t *addr;
02539     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
02540     HAL_REG_WRITE(*addr, MCPWM_IF_TH10_IF_BIT);
02541 }
02542
02548 uint8_t HAL_MCPWM_IrqTH11(HAL_MCPWM_x mcpwmx)
02549 {
02550     volatile uint32_t *addr_if;
02551     volatile uint32_t *addr_ie;
02552     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
02553     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
02554     if ((HAL_REG_READ_BIT(*addr_ie, MCPWM_IE_TH11_IE_BIT)) && (HAL_REG_READ_BIT(*addr_if,
MCPWM_IF_TH11_IF_BIT)))
02555     {
02556         return 1;
02557     }
02558     else
02559     {
02560         return 0;
02561     }
02562 }
02563
02569 uint8_t HAL_MCPWM_IrqTH11Flg(HAL_MCPWM_x mcpwmx)

```

```

02570 {
02571     volatile uint32_t *addr;
02572     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
02573     if (HAL_REG_READ_BIT(*addr, MCPWM_IF_TH11_IF_BIT))
02574     {
02575         return 1;
02576     }
02577     else
02578     {
02579         return 0;
02580     }
02581 }
02582
02587 void HAL_MCPWM_IrqTH11Enable(HAL_MCPWM_x mcpwmx)
02588 {
02589     volatile uint32_t *addr;
02590     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
02591     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
02592     HAL_REG_SET(*addr, MCPWM_IE_TH11_IE_BIT);
02593     HAL_REG_WRITE(MCPWM_PRT, 0);
02594 }
02595
02600 void HAL_MCPWM_IrqTH11Disable(HAL_MCPWM_x mcpwmx)
02601 {
02602     volatile uint32_t *addr;
02603     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
02604     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
02605     HAL_REG_RESET(*addr, MCPWM_IE_TH11_IE_BIT);
02606     HAL_REG_WRITE(MCPWM_PRT, 0);
02607 }
02608
02613 void HAL_MCPWM_IrqTH11Clear(HAL_MCPWM_x mcpwmx)
02614 {
02615     volatile uint32_t *addr;
02616     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
02617     HAL_REG_WRITE(*addr, MCPWM_IF_TH11_IF_BIT);
02618 }
02619
02625 uint8_t HAL_MCPWM_IrqTH20(HAL_MCPWM_x mcpwmx)
02626 {
02627     volatile uint32_t *addr_if;
02628     volatile uint32_t *addr_ie;
02629     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
02630     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
02631     if ((HAL_REG_READ_BIT(*addr_ie, MCPWM_IE_TH20_IE_BIT)) && (HAL_REG_READ_BIT(*addr_if,
MCPWM_IF_TH20_IF_BIT)))
02632     {
02633         return 1;
02634     }
02635     else
02636     {
02637         return 0;
02638     }
02639 }
02640
02646 uint8_t HAL_MCPWM_IrqTH20Flg(HAL_MCPWM_x mcpwmx)
02647 {
02648     volatile uint32_t *addr;
02649     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
02650     if (HAL_REG_READ_BIT(*addr, MCPWM_IF_TH20_IF_BIT))
02651     {
02652         return 1;
02653     }
02654     else
02655     {
02656         return 0;
02657     }
02658 }
02659
02664 void HAL_MCPWM_IrqTH20Enable(HAL_MCPWM_x mcpwmx)
02665 {
02666     volatile uint32_t *addr;
02667     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
02668     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
02669     HAL_REG_SET(*addr, MCPWM_IE_TH20_IE_BIT);
02670     HAL_REG_WRITE(MCPWM_PRT, 0);
02671 }
02672
02677 void HAL_MCPWM_IrqTH20Disable(HAL_MCPWM_x mcpwmx)
02678 {
02679     volatile uint32_t *addr;
02680     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
02681     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
02682     HAL_REG_RESET(*addr, MCPWM_IE_TH20_IE_BIT);
02683     HAL_REG_WRITE(MCPWM_PRT, 0);
02684 }
02685

```

```

02690 void HAL_MCPWM_IrqTH20Clear(HAL_MCPWM_x mcpwmx)
02691 {
02692     volatile uint32_t *addr;
02693     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
02694     HAL_REG_WRITE(*addr, MCPWM_IF_TH20_IF_BIT);
02695 }
02696
02702 uint8_t HAL_MCPWM_IrqTH21(HAL_MCPWM_x mcpwmx)
02703 {
02704     volatile uint32_t *addr_if;
02705     volatile uint32_t *addr_ie;
02706     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
02707     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
02708     if ((HAL_REG_READ_BIT(*addr_ie, MCPWM_IE_TH21_IE_BIT)) && (HAL_REG_READ_BIT(*addr_if,
MCPWM_IF_TH21_IF_BIT)))
02709     {
02710         return 1;
02711     }
02712     else
02713     {
02714         return 0;
02715     }
02716 }
02717
02723 uint8_t HAL_MCPWM_IrqTH21Flg(HAL_MCPWM_x mcpwmx)
02724 {
02725     volatile uint32_t *addr;
02726     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
02727     if (HAL_REG_READ_BIT(*addr, MCPWM_IF_TH21_IF_BIT))
02728     {
02729         return 1;
02730     }
02731     else
02732     {
02733         return 0;
02734     }
02735 }
02736
02741 void HAL_MCPWM_IrqTH21Enable(HAL_MCPWM_x mcpwmx)
02742 {
02743     volatile uint32_t *addr;
02744     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
02745     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
02746     HAL_REG_SET(*addr, MCPWM_IE_TH21_IE_BIT);
02747     HAL_REG_WRITE(MCPWM_PRT, 0);
02748 }
02749
02754 void HAL_MCPWM_IrqTH21Disable(HAL_MCPWM_x mcpwmx)
02755 {
02756     volatile uint32_t *addr;
02757     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
02758     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
02759     HAL_REG_RESET(*addr, MCPWM_IE_TH21_IE_BIT);
02760     HAL_REG_WRITE(MCPWM_PRT, 0);
02761 }
02762
02767 void HAL_MCPWM_IrqTH21Clear(HAL_MCPWM_x mcpwmx)
02768 {
02769     volatile uint32_t *addr;
02770     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
02771     HAL_REG_WRITE(*addr, MCPWM_IF_TH21_IF_BIT);
02772 }
02773
02779 uint8_t HAL_MCPWM_IrqTH30(HAL_MCPWM_x mcpwmx)
02780 {
02781     volatile uint32_t *addr_if;
02782     volatile uint32_t *addr_ie;
02783     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
02784     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
02785     if ((HAL_REG_READ_BIT(*addr_ie, MCPWM_IE_TH30_IE_BIT)) && (HAL_REG_READ_BIT(*addr_if,
MCPWM_IF_TH30_IF_BIT)))
02786     {
02787         return 1;
02788     }
02789     else
02790     {
02791         return 0;
02792     }
02793 }
02794
02800 uint8_t HAL_MCPWM_IrqTH30Flg(HAL_MCPWM_x mcpwmx)
02801 {
02802     volatile uint32_t *addr;
02803     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
02804     if (HAL_REG_READ_BIT(*addr, MCPWM_IF_TH30_IF_BIT))
02805     {
02806         return 1;

```

```

02807     }
02808     else
02809     {
02810         return 0;
02811     }
02812 }
02813
02818 void HAL_MCPWM_IrqTH30Enable(HAL_MCPWM_x mcpwmx)
02819 {
02820     volatile uint32_t *addr;
02821     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
02822     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
02823     HAL_REG_SET(*addr, MCPWM_IE_TH30_IE_BIT);
02824     HAL_REG_WRITE(MCPWM_PRT, 0);
02825 }
02826
02831 void HAL_MCPWM_IrqTH30Disable(HAL_MCPWM_x mcpwmx)
02832 {
02833     volatile uint32_t *addr;
02834     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
02835     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
02836     HAL_REG_RESET(*addr, MCPWM_IE_TH30_IE_BIT);
02837     HAL_REG_WRITE(MCPWM_PRT, 0);
02838 }
02839
02844 void HAL_MCPWM_IrqTH30Clear(HAL_MCPWM_x mcpwmx)
02845 {
02846     volatile uint32_t *addr;
02847     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
02848     HAL_REG_WRITE(*addr, MCPWM_IF_TH30_IF_BIT);
02849 }
02850
02856 uint8_t HAL_MCPWM_IrqTH31(HAL_MCPWM_x mcpwmx)
02857 {
02858     volatile uint32_t *addr_if;
02859     volatile uint32_t *addr_ie;
02860     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
02861     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
02862     if ((HAL_REG_READ_BIT(*addr_ie, MCPWM_IE_TH31_IE_BIT) && (HAL_REG_READ_BIT(*addr_if,
MCPWM_IF_TH31_IF_BIT)))
02863     {
02864         return 1;
02865     }
02866     else
02867     {
02868         return 0;
02869     }
02870 }
02871
02877 uint8_t HAL_MCPWM_IrqTH31Flg(HAL_MCPWM_x mcpwmx)
02878 {
02879     volatile uint32_t *addr;
02880     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
02881     if (HAL_REG_READ_BIT(*addr, MCPWM_IF_TH31_IF_BIT))
02882     {
02883         return 1;
02884     }
02885     else
02886     {
02887         return 0;
02888     }
02889 }
02890
02895 void HAL_MCPWM_IrqTH31Enable(HAL_MCPWM_x mcpwmx)
02896 {
02897     volatile uint32_t *addr;
02898     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
02899     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
02900     HAL_REG_SET(*addr, MCPWM_IE_TH31_IE_BIT);
02901     HAL_REG_WRITE(MCPWM_PRT, 0);
02902 }
02903
02908 void HAL_MCPWM_IrqTH31Disable(HAL_MCPWM_x mcpwmx)
02909 {
02910     volatile uint32_t *addr;
02911     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IE0);
02912     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
02913     HAL_REG_RESET(*addr, MCPWM_IE_TH31_IE_BIT);
02914     HAL_REG_WRITE(MCPWM_PRT, 0);
02915 }
02916
02921 void HAL_MCPWM_IrqTH31Clear(HAL_MCPWM_x mcpwmx)
02922 {
02923     volatile uint32_t *addr;
02924     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwmx & 0xf]->IF0);
02925     HAL_REG_WRITE(*addr, MCPWM_IF_TH31_IF_BIT);
02926 }

```

```

02927
02933 uint8_t HAL_MCPWM_IrqShort(HAL_MCPWM_x mcpwm)
02934 {
02935     volatile uint32_t *addr_if;
02936     volatile uint32_t *addr_ie;
02937     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwm & 0xf]->EIE);
02938     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwm & 0xf]->EIF);
02939     if ((HAL_REG_READ_BIT(*addr_ie, MCPWM_EIE_CH0_SHORT_IE_BIT | MCPWM_EIE_CH1_SHORT_IE_BIT |
MCPWM_EIE_CH2_SHORT_IE_BIT | MCPWM_EIE_CH3_SHORT_IE_BIT)) && (HAL_REG_READ_BIT(*addr_if,
MCPWM{EIF_CH0_SHORT_IF_BIT | MCPWM{EIF_CH1_SHORT_IF_BIT | MCPWM{EIF_CH2_SHORT_IF_BIT |
MCPWM{EIF_CH3_SHORT_IF_BIT})))
02940     {
02941         return 1;
02942     }
02943     else
02944     {
02945         return 0;
02946     }
02947 }
02948
02954 uint8_t HAL_MCPWM_IrqShortFlg(HAL_MCPWM_x mcpwm)
02955 {
02956     volatile uint32_t *addr;
02957     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwm & 0xf]->EIF);
02958     if (HAL_REG_READ_BIT(*addr, MCPWM{EIF_CH0_SHORT_IF_BIT | MCPWM{EIF_CH1_SHORT_IF_BIT |
MCPWM{EIF_CH2_SHORT_IF_BIT | MCPWM{EIF_CH3_SHORT_IF_BIT}))
02959     {
02960         return 1;
02961     }
02962     else
02963     {
02964         return 0;
02965     }
02966 }
02967
02972 void HAL_MCPWM_IrqShortEnable(HAL_MCPWM_x mcpwm)
02973 {
02974     volatile uint32_t *addr;
02975     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwm & 0xf]->EIE);
02976     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
02977     HAL_REG_SET(*addr, MCPWM_EIE_CH0_SHORT_IE_BIT | MCPWM_EIE_CH1_SHORT_IE_BIT | MCPWM_EIE_CH2_SHORT_IE_BIT |
MCPWM_EIE_CH3_SHORT_IE_BIT);
02978     HAL_REG_WRITE(MCPWM_PRT, 0);
02979 }
02980
02985 void HAL_MCPWM_IrqShortDisable(HAL_MCPWM_x mcpwm)
02986 {
02987     volatile uint32_t *addr;
02988     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwm & 0xf]->EIE);
02989     HAL_REG_WRITE(MCPWM_PRT, 0xDEAD);
02990     HAL_REG_RESET(*addr, MCPWM_EIE_CH0_SHORT_IE_BIT | MCPWM_EIE_CH1_SHORT_IE_BIT | MCPWM_EIE_CH2_SHORT_IE_BIT
| MCPWM_EIE_CH3_SHORT_IE_BIT);
02991     HAL_REG_WRITE(MCPWM_PRT, 0);
02992 }
02993
02998 void HAL_MCPWM_IrqShortClear(HAL_MCPWM_x mcpwm)
02999 {
03000     volatile uint32_t *addr;
03001     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_MCPWM[mcpwm & 0xf]->EIF);
03002     HAL_REG_WRITE(*addr, MCPWM{EIF_CH0_SHORT_IF_BIT | MCPWM{EIF_CH1_SHORT_IF_BIT | MCPWM{EIF_CH2_SHORT_IF_BIT
| MCPWM{EIF_CH3_SHORT_IF_BIT});
03003 }
03004
03010 uint8_t HAL_I2C_IrqDone(HAL_I2C_x i2c)
03011 {
03012     volatile uint32_t *addr_if;
03013     volatile uint32_t *addr_ie;
03014     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_I2C[i2c]->CFG);
03015     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_I2C[i2c]->SCR);
03016     if ((HAL_REG_READ_BIT(*addr_ie, I2C0_CFG_IE_BIT | I2C0_CFG_TC_IE_BIT)) && (HAL_REG_READ_BIT(*addr_if,
I2C0_SCR_Done_BIT)))
03017     {
03018         return 1;
03019     }
03020     else
03021     {
03022         return 0;
03023     }
03024 }
03025
03031 uint8_t HAL_I2C_IrqDoneFlg(HAL_I2C_x i2c)
03032 {
03033     volatile uint32_t *addr;
03034     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_I2C[i2c]->SCR);
03035     if (HAL_REG_READ_BIT(*addr, I2C0_SCR_Done_BIT))
03036     {
03037         return 1;

```

```

03038     }
03039     else
03040     {
03041         return 0;
03042     }
03043 }
03044
03049 void HAL_I2C_IrqDoneEnable(HAL_I2C_x i2c_x)
03050 {
03051     volatile uint32_t *addr;
03052     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_I2C[i2c_x]->CFG);
03053     HAL_REG_SET(*addr, I2C0_CFG_IE_BIT | I2C0_CFG_TC_IE_BIT);
03054 }
03055
03060 void HAL_I2C_IrqDoneDisable(HAL_I2C_x i2c_x)
03061 {
03062     volatile uint32_t *addr;
03063     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_I2C[i2c_x]->CFG);
03064     HAL_REG_RESET(*addr, I2C0_CFG_TC_IE_BIT);
03065 }
03066
03071 void HAL_I2C_IrqDoneClear(HAL_I2C_x i2c_x)
03072 {
03073     volatile uint32_t *addr;
03074     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_I2C[i2c_x]->SCR);
03075     HAL_REG_RESET(*addr, I2C0_SCR_Done_BIT);
03076 }
03077
03083 uint8_t HAL_I2C_IrqBusErr(HAL_I2C_x i2c_x)
03084 {
03085     volatile uint32_t *addr_if;
03086     volatile uint32_t *addr_ie;
03087     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_I2C[i2c_x]->CFG);
03088     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_I2C[i2c_x]->SCR);
03089     if ((HAL_REG_READ_BIT(*addr_ie, I2C0_CFG_IE_BIT | I2C0_CFG_BUS_ERR_IE_BIT)) &&
03090         (HAL_REG_READ_BIT(*addr_if, I2C0_SCR_STT_ERR_BIT)))
03091     {
03092         return 1;
03093     }
03094     else
03095     {
03096         return 0;
03097     }
03098 }
03104 uint8_t HAL_I2C_IrqBusErrFlg(HAL_I2C_x i2c_x)
03105 {
03106     volatile uint32_t *addr;
03107     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_I2C[i2c_x]->SCR);
03108     if (HAL_REG_READ_BIT(*addr, I2C0_SCR_STT_ERR_BIT))
03109     {
03110         return 1;
03111     }
03112     else
03113     {
03114         return 0;
03115     }
03116 }
03117
03122 void HAL_I2C_IrqBusErrEnable(HAL_I2C_x i2c_x)
03123 {
03124     volatile uint32_t *addr;
03125     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_I2C[i2c_x]->CFG);
03126     HAL_REG_SET(*addr, I2C0_CFG_IE_BIT | I2C0_CFG_BUS_ERR_IE_BIT);
03127 }
03128
03133 void HAL_I2C_IrqBusErrDisable(HAL_I2C_x i2c_x)
03134 {
03135     volatile uint32_t *addr;
03136     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_I2C[i2c_x]->CFG);
03137     HAL_REG_RESET(*addr, I2C0_CFG_BUS_ERR_IE_BIT);
03138 }
03139
03144 void HAL_I2C_IrqBusErrClear(HAL_I2C_x i2c_x)
03145 {
03146     volatile uint32_t *addr;
03147     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_I2C[i2c_x]->SCR);
03148     HAL_REG_RESET(*addr, I2C0_SCR_STT_ERR_BIT);
03149 }
03150
03156 uint8_t HAL_I2C_IrqStop(HAL_I2C_x i2c_x)
03157 {
03158     volatile uint32_t *addr_if;
03159     volatile uint32_t *addr_ie;
03160     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_I2C[i2c_x]->CFG);
03161     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_I2C[i2c_x]->SCR);
03162     if ((HAL_REG_READ_BIT(*addr_ie, I2C0_CFG_IE_BIT | I2C0_CFG_STOP_IE_BIT)) && (HAL_REG_READ_BIT(*addr_if,

```

```

I2C0.SCR.STOP.EVT.BIT)))
03163     {
03164         return 1;
03165     }
03166     else
03167     {
03168         return 0;
03169     }
03170 }
03171
03177 uint8_t HAL_I2C_IrqStopFlg(HAL_I2C_TypeDef *I2Cx)
03178 {
03179     volatile uint32_t *addr;
03180     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_I2C[I2Cx] -> SCR);
03181     if (HAL_REG_READ_BIT(*addr, I2C0.SCR.STOP.EVT.BIT))
03182     {
03183         return 1;
03184     }
03185     else
03186     {
03187         return 0;
03188     }
03189 }
03190
03195 void HAL_I2C_IrqStopEnable(HAL_I2C_TypeDef *I2Cx)
03196 {
03197     volatile uint32_t *addr;
03198     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_I2C[I2Cx] -> CFG);
03199     HAL_REG_SET(*addr, I2C0.CFG.IE.BIT | I2C0.CFG.STOP.IE.BIT);
03200 }
03201
03206 void HAL_I2C_IrqStopDisable(HAL_I2C_TypeDef *I2Cx)
03207 {
03208     volatile uint32_t *addr;
03209     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_I2C[I2Cx] -> CFG);
03210     HAL_REG_RESET(*addr, I2C0.CFG.STOP.IE.BIT);
03211 }
03212
03217 void HAL_I2C_IrqStopClear(HAL_I2C_TypeDef *I2Cx)
03218 {
03219     volatile uint32_t *addr;
03220     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_I2C[I2Cx] -> SCR);
03221     HAL_REG_RESET(*addr, I2C0.SCR.STOP.EVT.BIT);
03222 }
03223
03229 uint8_t HAL_SPI_IrqDone(HAL_SPI_TypeDef *SPIx)
03230 {
03231     volatile uint32_t *addr_if;
03232     volatile uint32_t *addr_ie;
03233     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_SPI[SPIx] -> IE);
03234     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_SPI[SPIx] -> IE);
03235     if ((HAL_REG_READ_BIT(*addr_ie, SPI_IE_IE_BIT | SPI_IE_CMPLT_IE_BIT) && (HAL_REG_READ_BIT(*addr_if,
SPI_IE_CMPLT_IF_BIT)))
03236     {
03237         return 1;
03238     }
03239     else
03240     {
03241         return 0;
03242     }
03243 }
03244
03250 uint8_t HAL_SPI_IrqDoneFlg(HAL_SPI_TypeDef *SPIx)
03251 {
03252     volatile uint32_t *addr;
03253     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_SPI[SPIx] -> IE);
03254     if (HAL_REG_READ_BIT(*addr, SPI_IE_CMPLT_IF_BIT))
03255     {
03256         return 1;
03257     }
03258     else
03259     {
03260         return 0;
03261     }
03262 }
03263
03268 void HAL_SPI_IrqDoneEnable(HAL_SPI_TypeDef *SPIx)
03269 {
03270     volatile uint32_t *addr;
03271     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_SPI[SPIx] -> IE);
03272     HAL_REG_SET(*addr, SPI_IE_IE_BIT | SPI_IE_CMPLT_IE_BIT);
03273 }
03274
03279 void HAL_SPI_IrqDoneDisable(HAL_SPI_TypeDef *SPIx)
03280 {
03281     volatile uint32_t *addr;
03282     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_SPI[SPIx] -> IE);

```

```

03283     HAL_REG_RESET(*addr, SPI_IE_CMPLT_IE_BIT);
03284 }
03285
03290 void HAL_SPI_IrqDoneClear(HAL_SPI_TypeDef spi_x)
03291 {
03292     volatile uint32_t *addr;
03293     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_SPI[spi_x]->IE);
03294     HAL_REG_SET(*addr, SPI_IE_CMPLT_IF_BIT);
03295 }
03296
03302 uint8_t HAL_SPI_IrqErr(HAL_SPI_TypeDef spi_x)
03303 {
03304     volatile uint32_t *addr_if;
03305     volatile uint32_t *addr_ie;
03306     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_SPI[spi_x]->IE);
03307     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_SPI[spi_x]->IF);
03308     if ((HAL_REG_READ_BIT(*addr_ie, SPI_IE_IE_BIT | SPI_IE_AB_IE_BIT) && (HAL_REG_READ_BIT(*addr_if,
SPI_IE_AB_IF_BIT)))
03309     {
03310         return 1;
03311     }
03312     else
03313     {
03314         return 0;
03315     }
03316 }
03317
03323 uint8_t HAL_SPI_IrqErrFlg(HAL_SPI_TypeDef spi_x)
03324 {
03325     volatile uint32_t *addr;
03326     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_SPI[spi_x]->IE);
03327     if (HAL_REG_READ_BIT(*addr, SPI_IE_AB_IF_BIT))
03328     {
03329         return 1;
03330     }
03331     else
03332     {
03333         return 0;
03334     }
03335 }
03336
03341 void HAL_SPI_IrqErrEnable(HAL_SPI_TypeDef spi_x)
03342 {
03343     volatile uint32_t *addr;
03344     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_SPI[spi_x]->IE);
03345     HAL_REG_SET(*addr, SPI_IE_IE_BIT | SPI_IE_AB_IE_BIT);
03346 }
03347
03352 void HAL_SPI_IrqErrDisable(HAL_SPI_TypeDef spi_x)
03353 {
03354     volatile uint32_t *addr;
03355     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_SPI[spi_x]->IE);
03356     HAL_REG_RESET(*addr, SPI_IE_AB_IE_BIT);
03357 }
03358
03363 void HAL_SPI_IrqErrClear(HAL_SPI_TypeDef spi_x)
03364 {
03365     volatile uint32_t *addr;
03366     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_SPI[spi_x]->IE);
03367     HAL_REG_SET(*addr, SPI_IE_AB_IF_BIT);
03368 }
03369
03375 uint8_t HAL_SPI_IrqOverflow(HAL_SPI_TypeDef spi_x)
03376 {
03377     volatile uint32_t *addr_if;
03378     volatile uint32_t *addr_ie;
03379     addr_ie = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_SPI[spi_x]->IE);
03380     addr_if = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_SPI[spi_x]->IF);
03381     if ((HAL_REG_READ_BIT(*addr_ie, SPI_IE_IE_BIT | SPI_IE_OV_IE_BIT) && (HAL_REG_READ_BIT(*addr_if,
SPI_IE_OV_IF_BIT)))
03382     {
03383         return 1;
03384     }
03385     else
03386     {
03387         return 0;
03388     }
03389 }
03390
03396 uint8_t HAL_SPI_IrqOverflowFlg(HAL_SPI_TypeDef spi_x)
03397 {
03398     volatile uint32_t *addr;
03399     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_SPI[spi_x]->IE);
03400     if (HAL_REG_READ_BIT(*addr, SPI_IE_OV_IF_BIT))
03401     {
03402         return 1;
03403     }

```



```

03404     else
03405     {
03406         return 0;
03407     }
03408 }
03409
03414 void HAL_SPI_IrqOverflowEnable(HAL_SPI_TypeDef spi_x)
03415 {
03416     volatile uint32_t *addr;
03417     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_SPI(spi_x)->IE);
03418     HAL_REG_SET(*addr, SPI_IE_IE_BIT | SPI_IE_OV_IE_BIT);
03419 }
03420
03425 void HAL_SPI_IrqOverflowDisable(HAL_SPI_TypeDef spi_x)
03426 {
03427     volatile uint32_t *addr;
03428     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_SPI(spi_x)->IE);
03429     HAL_REG_RESET(*addr, SPI_IE_OV_IE_BIT);
03430 }
03431
03436 void HAL_SPI_IrqOverflowClear(HAL_SPI_TypeDef spi_x)
03437 {
03438     volatile uint32_t *addr;
03439     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(HAL_SPI(spi_x)->IE);
03440     HAL_REG_SET(*addr, SPI_IE_OV_IF_BIT);
03441 }
03442
03448 uint8_t HAL_DMA_IrqDone(HAL_DMA_TypeDef dma_x)
03449 {
03450     volatile uint32_t *addr;
03451     (void)dma_x;
03452     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(DMA_IF);
03453     if (HAL_REG_READ_BIT(*addr, DMA_IF_CH0_FIF_BIT << (dma_x)))
03454     {
03455         return 1;
03456     }
03457     else
03458     {
03459         return 0;
03460     }
03461 }
03462
03468 uint8_t HAL_DMA_IrqDoneFlg(HAL_DMA_TypeDef dma_x)
03469 {
03470     volatile uint32_t *addr;
03471     (void)dma_x;
03472     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(DMA_IF);
03473     if (HAL_REG_READ_BIT(*addr, DMA_IF_CH0_FIF_BIT << (dma_x)))
03474     {
03475         return 1;
03476     }
03477     else
03478     {
03479         return 0;
03480     }
03481 }
03482
03487 void HAL_DMA_IrqDoneEnable(HAL_DMA_TypeDef dma_x)
03488 {
03489     (void)dma_x;
03490 }
03491
03496 void HAL_DMA_IrqDoneDisable(HAL_DMA_TypeDef dma_x)
03497 {
03498     (void)dma_x;
03499 }
03500
03505 void HAL_DMA_IrqDoneClear(HAL_DMA_TypeDef dma_x)
03506 {
03507     volatile uint32_t *addr;
03508     (void)dma_x;
03509     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(DMA_IF);
03510     HAL_REG_SET(*addr, DMA_IF_CH0_FIF_BIT << (dma_x));
03511 }
03512
03518 uint8_t HAL_DMA_IrqErr(HAL_DMA_TypeDef dma_x)
03519 {
03520     volatile uint32_t *addr;
03521     (void)dma_x;
03522     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(DMA_IF);
03523     if (HAL_REG_READ_BIT(*addr, DMA_IF_CH0{EIF_BIT << (dma_x)))
03524     {
03525         return 1;
03526     }
03527     else
03528     {
03529         return 0;

```

```

03530     }
03531 }
03532
03538 uint8_t HAL_DMA_IrqErrFlg(HAL_DMA_x dma_x)
03539 {
03540     volatile uint32_t *addr;
03541     (void)dma_x;
03542     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(DMA_IF);
03543     if (HAL_REG_READ_BIT(*addr, DMA_IF_CH0_EIF_BIT << (dma_x)))
03544     {
03545         return 1;
03546     }
03547     else
03548     {
03549         return 0;
03550     }
03551 }
03552
03557 void HAL_DMA_IrqErrEnable(HAL_DMA_x dma_x)
03558 {
03559     (void)dma_x;
03560 }
03561
03566 void HAL_DMA_IrqErrDisable(HAL_DMA_x dma_x)
03567 {
03568     (void)dma_x;
03569 }
03570
03575 void HAL_DMA_IrqErrClear(HAL_DMA_x dma_x)
03576 {
03577     volatile uint32_t *addr;
03578     (void)dma_x;
03579     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(DMA_IF);
03580     HAL_REG_SET(*addr, DMA_IF_CH0_EIF_BIT << (dma_x));
03581 }
03582
03588 uint8_t HAL_GPIO_Irq(uint8_t line)
03589 {
03590     volatile uint32_t *addr;
03591     (void)line;
03592     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(EXTI_IF);
03593     if (HAL_REG_READ_BIT(*addr, EXTI_IF_IF0_BIT << (line)))
03594     {
03595         return 1;
03596     }
03597     else
03598     {
03599         return 0;
03600     }
03601 }
03602
03608 uint8_t HAL_GPIO_IrqFlg(uint8_t line)
03609 {
03610     volatile uint32_t *addr;
03611     (void)line;
03612     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(EXTI_IF);
03613     if (HAL_REG_READ_BIT(*addr, EXTI_IF_IF0_BIT << (line)))
03614     {
03615         return 1;
03616     }
03617     else
03618     {
03619         return 0;
03620     }
03621 }
03622
03627 void HAL_GPIO_IrqEnable(uint8_t line)
03628 {
03629     (void)line;
03630 }
03631
03636 void HAL_GPIO_IrqDisable(uint8_t line)
03637 {
03638     (void)line;
03639 }
03640
03645 void HAL_GPIO_IrqClear(uint8_t line)
03646 {
03647     volatile uint32_t *addr;
03648     (void)line;
03649     addr = (volatile uint32_t *)HAL_REG_READ_ADDR(EXTI_IF);
03650     HAL_REG_SET(*addr, EXTI_IF_IF0_BIT << (line));
03651 }
03652
03653 // 硬件不支持的中断(空实现)
03654
03655 uint8_t HAL_ADC_IrqIdleSamp(HAL_ADC_x adc)

```

```

03656 {
03657     (void)adc;
03658     return 0;
03659 }
03660 uint8_t HAL_ADC_IrqIdleSampFlg(HAL_ADC_x adc)
03661 {
03662     (void)adc;
03663     return 0;
03664 }
03665 void HAL_ADC_IrqIdleSampEnable(HAL_ADC_x adc)
03666 {
03667     (void)adc;
03668 }
03669 void HAL_ADC_IrqIdleSampDisable(HAL_ADC_x adc)
03670 {
03671     (void)adc;
03672 }
03673 void HAL_ADC_IrqIdleSampClear(HAL_ADC_x adc)
03674 {
03675     (void)adc;
03676 }
03677
03678 uint8_t HAL_ADC_IrqWatchdog1(HAL_ADC_x adc)
03679 {
03680     (void)adc;
03681     return 0;
03682 }
03683 uint8_t HAL_ADC_IrqWatchdog1Flg(HAL_ADC_x adc)
03684 {
03685     (void)adc;
03686     return 0;
03687 }
03688 void HAL_ADC_IrqWatchdog1Enable(HAL_ADC_x adc)
03689 {
03690     (void)adc;
03691 }
03692 void HAL_ADC_IrqWatchdog1Disable(HAL_ADC_x adc)
03693 {
03694     (void)adc;
03695 }
03696 void HAL_ADC_IrqWatchdog1Clear(HAL_ADC_x adc)
03697 {
03698     (void)adc;
03699 }
03700
03701 uint8_t HAL_MCPWM_IrqTH40(HAL_MCPWM_x mcpwm)
03702 {
03703     (void)mcpwm;
03704     return 0;
03705 }
03706 uint8_t HAL_MCPWM_IrqTH40Flg(HAL_MCPWM_x mcpwm)
03707 {
03708     (void)mcpwm;
03709     return 0;
03710 }
03711 void HAL_MCPWM_IrqTH40Enable(HAL_MCPWM_x mcpwm)
03712 {
03713     (void)mcpwm;
03714 }
03715 void HAL_MCPWM_IrqTH40Disable(HAL_MCPWM_x mcpwm)
03716 {
03717     (void)mcpwm;
03718 }
03719 void HAL_MCPWM_IrqTH40Clear(HAL_MCPWM_x mcpwm)
03720 {
03721     (void)mcpwm;
03722 }
03723
03724 uint8_t HAL_MCPWM_IrqTH41(HAL_MCPWM_x mcpwm)
03725 {
03726     (void)mcpwm;
03727     return 0;
03728 }
03729 uint8_t HAL_MCPWM_IrqTH41Flg(HAL_MCPWM_x mcpwm)
03730 {
03731     (void)mcpwm;
03732     return 0;
03733 }
03734 void HAL_MCPWM_IrqTH41Enable(HAL_MCPWM_x mcpwm)
03735 {
03736     (void)mcpwm;
03737 }
03738 void HAL_MCPWM_IrqTH41Disable(HAL_MCPWM_x mcpwm)
03739 {
03740     (void)mcpwm;
03741 }
03742 void HAL_MCPWM_IrqTH41Clear(HAL_MCPWM_x mcpwm)

```

```

03743 {
03744     (void)mcpwmx;
03745 }
03746
03747 uint8_t HAL_MCPWM_IrqTH50(HAL_MCPWM_x mcpwmx)
03748 {
03749     (void)mcpwmx;
03750     return 0;
03751 }
03752 uint8_t HAL_MCPWM_IrqTH50Flg(HAL_MCPWM_x mcpwmx)
03753 {
03754     (void)mcpwmx;
03755     return 0;
03756 }
03757 void HAL_MCPWM_IrqTH50Enable(HAL_MCPWM_x mcpwmx)
03758 {
03759     (void)mcpwmx;
03760 }
03761 void HAL_MCPWM_IrqTH50Disable(HAL_MCPWM_x mcpwmx)
03762 {
03763     (void)mcpwmx;
03764 }
03765 void HAL_MCPWM_IrqTH50Clear(HAL_MCPWM_x mcpwmx)
03766 {
03767     (void)mcpwmx;
03768 }
03769
03770 uint8_t HAL_MCPWM_IrqTH51(HAL_MCPWM_x mcpwmx)
03771 {
03772     (void)mcpwmx;
03773     return 0;
03774 }
03775 uint8_t HAL_MCPWM_IrqTH51Flg(HAL_MCPWM_x mcpwmx)
03776 {
03777     (void)mcpwmx;
03778     return 0;
03779 }
03780 void HAL_MCPWM_IrqTH51Enable(HAL_MCPWM_x mcpwmx)
03781 {
03782     (void)mcpwmx;
03783 }
03784 void HAL_MCPWM_IrqTH51Disable(HAL_MCPWM_x mcpwmx)
03785 {
03786     (void)mcpwmx;
03787 }
03788 void HAL_MCPWM_IrqTH51Clear(HAL_MCPWM_x mcpwmx)
03789 {
03790     (void)mcpwmx;
03791 }
03792
03793 uint8_t HAL_SW_Irq(uint8_t x)
03794 {
03795     (void)x;
03796     return 0;
03797 }
03798 uint8_t HAL_SW_IrqFlg(uint8_t x)
03799 {
03800     (void)x;
03801     return 0;
03802 }
03803 void HAL_SW_IrqEnable(uint8_t x)
03804 {
03805     (void)x;
03806 }
03807 void HAL_SW_IrqDisable(uint8_t x)
03808 {
03809     (void)x;
03810 }
03811 void HAL_SW_IrqClear(uint8_t x)
03812 {
03813     (void)x;
03814 }
03815
03816 void HAL_SW_IrqTrig(uint8_t x)
03817 {
03818     (void)x;
03819 }
03820
03821 uint8_t HAL_UART_IrqAbd(HAL_UART_x uart_x)
03822 {
03823     (void)uart_x;
03824     return 0;
03825 }
03826 uint8_t HAL_UART_IrqAbdFlg(HAL_UART_x uart_x)
03827 {
03828     (void)uart_x;
03829     return 0;

```

```

03830 }
03831 void HAL_UART_IrqAbdEnable(HAL_UART_x uart_x)
03832 {
03833     (void)uart_x;
03834 }
03835 void HAL_UART_IrqAbdDisable(HAL_UART_x uart_x)
03836 {
03837     (void)uart_x;
03838 }
03839 void HAL_UART_IrqAbdClear(HAL_UART_x uart_x)
03840 {
03841     (void)uart_x;
03842 }
03843
03844 uint8_t HAL_UART_IrqIdle(HAL_UART_x uart_x)
03845 {
03846     (void)uart_x;
03847     return 0;
03848 }
03849 uint8_t HAL_UART_IrqIdleFlg(HAL_UART_x uart_x)
03850 {
03851     (void)uart_x;
03852     return 0;
03853 }
03854 void HAL_UART_IrqIdleEnable(HAL_UART_x uart_x)
03855 {
03856     (void)uart_x;
03857 }
03858 void HAL_UART_IrqIdleDisable(HAL_UART_x uart_x)
03859 {
03860     (void)uart_x;
03861 }
03862 void HAL_UART_IrqIdleClear(HAL_UART_x uart_x)
03863 {
03864     (void)uart_x;
03865 }
03866
03867 uint8_t HAL_UART_IrqLbd(HAL_UART_x uart_x)
03868 {
03869     (void)uart_x;
03870     return 0;
03871 }
03872 uint8_t HAL_UART_IrqLbdFlg(HAL_UART_x uart_x)
03873 {
03874     (void)uart_x;
03875     return 0;
03876 }
03877 void HAL_UART_IrqLbdEnable(HAL_UART_x uart_x)
03878 {
03879     (void)uart_x;
03880 }
03881 void HAL_UART_IrqLbdDisable(HAL_UART_x uart_x)
03882 {
03883     (void)uart_x;
03884 }
03885 void HAL_UART_IrqLbdClear(HAL_UART_x uart_x)
03886 {
03887     (void)uart_x;
03888 }
03889
03890 uint8_t HAL_UART_IrqRxOv(HAL_UART_x uart_x)
03891 {
03892     (void)uart_x;
03893     return 0;
03894 }
03895 uint8_t HAL_UART_IrqRxOvFlg(HAL_UART_x uart_x)
03896 {
03897     (void)uart_x;
03898     return 0;
03899 }
03900 void HAL_UART_IrqRxOvEnable(HAL_UART_x uart_x)
03901 {
03902     (void)uart_x;
03903 }
03904 void HAL_UART_IrqRxOvDisable(HAL_UART_x uart_x)
03905 {
03906     (void)uart_x;
03907 }
03908 void HAL_UART_IrqRxOvClear(HAL_UART_x uart_x)
03909 {
03910     (void)uart_x;
03911 }
03912
03913 uint8_t HAL_UART_IrqTxOv(HAL_UART_x uart_x)
03914 {
03915     (void)uart_x;
03916     return 0;

```

```

03917 }
03918 uint8_t HAL_UART_IrqTxOvFlg(HAL_UART_x uart_x)
03919 {
03920     (void)uart_x;
03921     return 0;
03922 }
03923 void HAL_UART_IrqTxOvEnable(HAL_UART_x uart_x)
03924 {
03925     (void)uart_x;
03926 }
03927 void HAL_UART_IrqTxOvDisable(HAL_UART_x uart_x)
03928 {
03929     (void)uart_x;
03930 }
03931 void HAL_UART_IrqTxOvClear(HAL_UART_x uart_x)
03932 {
03933     (void)uart_x;
03934 }

```

4.94 source/hal_lks32mc08x_iwdg.c 文件参考

```

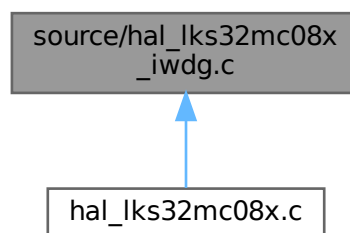
#include "../include/hal_lksmcu_iwdg.h"
#include "../hal_lks32mc08x.h"
#include "lks32mc08x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc08x_reg.h"

```

hal_lks32mc08x_iwdg.c 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



函数

- void **HAL_IWDG.Reset** (HAL_IWDG_x iwdg)
模块复位
- void **HAL_IWDG.Enable** (HAL_IWDG_x iwdg)
模块打开

- void [HAL_IWDG_Disable](#) ([HAL_IWDG_x](#) iwdg)
模块关闭
- void [HAL_IWDG_SetResetTimemS](#) ([HAL_IWDG_x](#) iwdg, int32_t ms)
设置看门狗超时复位时间
- void [HAL_IWDG_Feed](#) ([HAL_IWDG_x](#) iwdg)
喂狗

变量

- static uint32_t [hal_iwdg_cfg](#) = 0

4.94.1 函数说明

4.94.1.1 HAL_IWDG_Disable()

```
void HAL_IWDG_Disable (  
    HAL\_IWDG\_x iwdg )
```

模块关闭

参数

<i>iwdg</i>	看门狗模块

在文件 [hal_lks32mc08x_iwdg.c](#) 第 32 行定义.

这是这个函数的调用关系图:



4.94.1.2 HAL_IWDG_Enable()

```
void HAL_IWDG_Enable (  
    HAL\_IWDG\_x iwdg )
```

模块打开

参数

<i>iwdg</i>	看门狗模块

在文件 [hal_lks32mc08x_iwdg.c](#) 第 20 行定义.

4.94.1.3 HAL_IWDG_Feed()

```
void HAL_IWDG_Feed (
    HAL_IWDG_x iwdg )
```

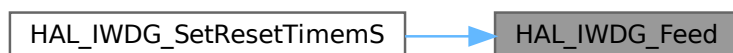
喂狗

参数

<i>iwdg</i>	看门狗模块

在文件 [hal_lks32mc08x_iwdg.c](#) 第 71 行定义.

这是这个函数的调用关系图:



4.94.1.4 HAL_IWDG_Reset()

```
void HAL_IWDG_Reset (
    HAL_IWDG_x iwdg )
```

模块复位

参数

<i>iwdg</i>	看门狗模块

在文件 [hal_lks32mc08x_iwdg.c](#) 第 11 行定义.

函数调用图:



4.94.1.5 HAL_IWDG_SetResetTimemS()

```
void HAL_IWDG_SetResetTimemS (
    HAL_IWDG_x iwdg,
    int32_t ms )
```

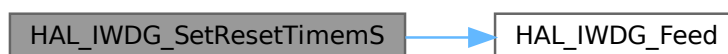
设置看门狗超时复位时间

参数

<i>iwdg</i>	看门狗模块
<i>ms</i>	超时复位时间,单位mS

在文件 [hal_lks32mc08x_iwdg.c](#) 第 45 行定义.

函数调用图:



4.94.2 变量说明

4.94.2.1 hal_iwdg_cfg

```
uint32_t hal_iwdg_cfg = 0 [static]
```

在文件 [hal_lks32mc08x_iwdg.c](#) 第 6 行定义.

4.95 hal_lks32mc08x_iwdg.c

[浏览该文件的文档.](#)

```
00001 #include "../include/hal_lksmcu_iwdg.h"
00002 #include "../hal_lks32mc08x.h"
00003 #include "lks32mc08x_user_manual.h"
00004 #include "../include/hal_lksmcu_reg.h"
00005 #include "../include/hal_lks32mc08x_reg.h"
00006 static uint32_t hal_iwdg_cfg = 0;
00011 void HAL_IWDG_Reset(HAL_IWDG_x iwdg)
00012 {
00013     HAL_IWDG_Disable(iwdg);
00014 }
00015
00020 void HAL_IWDG_Enable(HAL_IWDG_x iwdg)
00021 {
00022     (void)iwdg;
00023     HAL_REG_SYS_UNLOCK();
00024     HAL_REG_SET(SYS_RST_CFG, SYS_RST_CFG_WDT_EN_BIT);
00025     HAL_REG_SYS_LOCK();
00026 }
00027
00032 void HAL_IWDG_Disable(HAL_IWDG_x iwdg)
```

```

00033 {
00034     (void)wdg;
00035     HAL_REG_SYS_UNLOCK();
00036     HAL_REG_RESET(HAL_SYS[0] -> RST_CFG, SYS_RST_CFG_WDT_EN_BIT);
00037     HAL_REG_SYS_LOCK();
00038 }
00039
00045 void HAL_IWDG_SetResetTimemS(HAL_IWDG_x iwdg, int32_t ms)
00046 {
00047     (void)wdg;
00048     if (ms <= 2000)
00049     {
00050         hal_iwdg_cfg = 0x798F; // 2秒复位
00051     }
00052     else if (ms <= 4000)
00053     {
00054         hal_iwdg_cfg = 0x798D; // 4秒复位
00055     }
00056     else if (ms <= 8000)
00057     {
00058         hal_iwdg_cfg = 0x798B; // 8秒复位
00059     }
00060     else
00061     {
00062         hal_iwdg_cfg = 0x7989; // 64秒复位
00063     }
00064     HAL_IWDG_Feed(iwdg);
00065 }
00066
00071 void HAL_IWDG_Feed(HAL_IWDG_x iwdg)
00072 {
00073     (void)wdg;
00074     HAL_REG_WRITE(SYS_WR_PROTECT, 0xCAFE);
00075     HAL_REG_WRITE(SYS_WDT_CLR, hal_iwdg_cfg);
00076     HAL_REG_WRITE(SYS_WR_PROTECT, 0);
00077 }

```

4.96 source/hal_lks32mc08x_mcpwm.c 文件参考

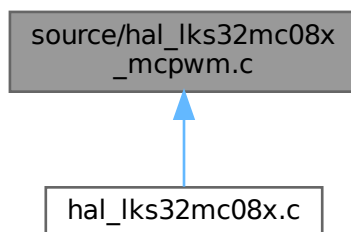
```

#include "../include/hal_lksmcu_mcpwm.h"
#include "../hal_lks32mc08x.h"
#include "lks32mc08x_user_manual.h"
#include "../include/hal_lksmcu_clk.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc08x_reg.h"
hal_lks32mc08x_mcpwm.c 的引用(Include)关系图:

```



此图展示该文件直接或间接的被哪些文件引用了：



函数

- void [HAL_MCPWM_UpdatePolarityState](#) (HAL_MCPWM_x mcpwm_x, HAL_MCPWM_PORT_x mcpwm_port_x)
设置FAIL信号来源
- void [HAL_MCPWM_Reset](#) (HAL_MCPWM_x mcpwm_x)
复位mcpwm模块
- void [HAL_MCPWM_Enable](#) (HAL_MCPWM_x mcpwm_x)
mcpwm模块使能
- void [HAL_MCPWM_Disable](#) (HAL_MCPWM_x mcpwm_x)
mcpwm模块关闭
- void [HAL_MCPWM_SetThCnt](#) (HAL_MCPWM_x mcpwm_x, uint32_t th)
设置计数器最大值
- void [HAL_MCPWM_SetFreq](#) (HAL_MCPWM_x mcpwm_x, uint32_t freq)
设置pwm频率
- uint32_t [HAL_MCPWM_GetFreq](#) (HAL_MCPWM_x mcpwm_x)
获取pwm频率
- void [HAL_MCPWM_SetClockDiv](#) (HAL_MCPWM_x mcpwm_x, uint32_t div)
- uint32_t [HAL_MCPWM_GetClockDiv](#) (HAL_MCPWM_x mcpwm_x)
- uint32_t [HAL_MCPWM_GetClockFreq](#) (HAL_MCPWM_x mcpwm_x)
- uint32_t [HAL_MCPWM_GetThCnt](#) (HAL_MCPWM_x mcpwm_x)
获取计数器最大值
- void [HAL_MCPWM_SetDead](#) (HAL_MCPWM_x mcpwm_x, uint32_t pdead_nS, uint32_t ndead_nS)
设置死区
- void [HAL_MCPWM_SetOutputDisable](#) (HAL_MCPWM_x mcpwm_x)
关闭输出
- void [HAL_MCPWM_SetOutputEnable](#) (HAL_MCPWM_x mcpwm_x)
打开输出
- void [HAL_MCPWM_StartCounting](#) (HAL_MCPWM_x mcpwm_x)
开始计数
- void [HAL_MCPWM_StopCounting](#) (HAL_MCPWM_x mcpwm_x)
停止计数(同时计数器回到-th)
- void [HAL_MCPWM_StartCountingSync](#) (HAL_MCPWM_x mcpwm1_x, HAL_MCPWM_x mcpwm2_x, int16_t phase)
跟随另一个计数器启动，周期不同时会自动映射相位差

- void `HAL_MCPWM_SetPwm` (`HAL_MCPWM_x` mcpwm_x, `HAL_MCPWM_PORT_x` mcpwm_port_x, int32_t t0, int32_t t1)
设置占空比
- void `HAL_MCPWM_SetPwmQ15` (`HAL_MCPWM_x` mcpwm_x, `HAL_MCPWM_PORT_x` mcpwm_port_x, int16_t t0, int16_t t1)
- void `HAL_MCPWM_SetOutState` (`HAL_MCPWM_x` mcpwm_x, `HAL_MCPWM_PORT_x` mcpwm_port_x, `HAL_MCPWM_OUTPUT_STATE_x` mcpwm_output_state_x)
设置输出状态
- void `HAL_MCPWM_SetDrivePolarity` (`HAL_MCPWM_x` mcpwm_x, `HAL_MCPWM_POLARITY_x` mcpwm_polarity_x)
设置pwm极性
- void `HAL_MCPWM_SetDrivePolarityPort` (`HAL_MCPWM_x` mcpwm_x, `HAL_MCPWM_PORT_x` mcpwm_port_x, `HAL_MCPWM_POLARITY_x` mcpwm_polarity_x)
设置单个输出通道的pwm极性
- void `HAL_MCPWM_SetSwap` (`HAL_MCPWM_x` mcpwm_x, `HAL_MCPWM_SWAP_x` mcpwm_swap_x)
设置PN是否交换
- void `HAL_MCPWM_SetSwapPort` (`HAL_MCPWM_x` mcpwm_x, `HAL_MCPWM_PORT_x` mcpwm_port_x, `HAL_MCPWM_SWAP_x` mcpwm_swap_x)
设置单个输出通道的PN是否交换
- void `HAL_MCPWM_SetRemap` (`HAL_MCPWM_x` mcpwm_x, uint8_t mode)
MCPWM通道重映射(通道输出到GPIO的路由重排,与HAL_MCPWM_SetSwap的PN交换无关)
- void `HAL_MCPWM_SetDefaultState` (`HAL_MCPWM_x` mcpwm_x, `HAL_MCPWM_DEFAULT_OUTPUT_STATE_x` mcpwm_default_output_state_x)
设置默认状态
- void `HAL_MCPWM_SetGpioOut` (`HAL_MCPWM_x` mcpwm_x, `HAL_GPIO_Px` px)
设置用于输出的gpio
- void `HAL_MCPWM_SetGpioFail` (`HAL_MCPWM_x` mcpwm_x, `HAL_GPIO_Px` px)
配置用于故障输入的gpio
- void `HAL_MCPWM_SetFailSignal` (`HAL_MCPWM_x` mcpwm_x, `HAL_MCPWM_FAIL_x` fail_x, `HAL_MCPWM_FAIL_IN_x` fail_in_x)
设置FAIL信号来源
- void `HAL_MCPWM_SetAdcTrig` (`HAL_MCPWM_x` mcpwm_x, int32_t t0, int32_t t1)
设置pwm输出ADC采样时间
- void `HAL_MCPWM_SetAdcTrigQ15` (`HAL_MCPWM_x` mcpwm_x, int32_t t0, int32_t t1)
设置pwm输出ADC采样时间q15格式输入
- void `HAL_MCPWM_SetFailFilter` (`HAL_MCPWM_x` mcpwm_x, uint32_t cnt)
设置FAIL信号滤波宽度
- void `HAL_MCPWM_SetUpdateMode` (`HAL_MCPWM_x` mcpwm_x, `HAL_MCPWM_UPDATE_x` update_x, uint32_t intv)
设置影子寄存器自动更新时刻

变量

- static uint16_t `HAL_MCPWM_STATE` [1][4] = {{0, 0, 0, 0}}
- static uint16_t `HAL_MCPWM_STATE_INIT_FLG` [1][4] = {{0, 0, 0, 0}}

4.96.1 函数说明

4.96.1.1 HAL_MCPWM_Disable()

```
void HAL_MCPWM_Disable (
    HAL_MCPWM_x mcpwm_x )
```

mcpwm模块关闭

参数

<i>mcpwm</i> ↔ _x	操作的模块编号

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 197 行定义.

4.96.1.2 HAL_MCPWM_Enable()

```
void HAL_MCPWM_Enable (
    HAL_MCPWM_x mcpwm_x )
```

mcpwm模块使能

参数

<i>mcpwm</i> ↔ _x	操作的模块编号

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 178 行定义.

4.96.1.3 HAL_MCPWM_GetClockDiv()

```
uint32_t HAL_MCPWM_GetClockDiv (
    HAL_MCPWM_x mcpwm_x )
```

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 278 行定义.

这是这个函数的调用关系图:

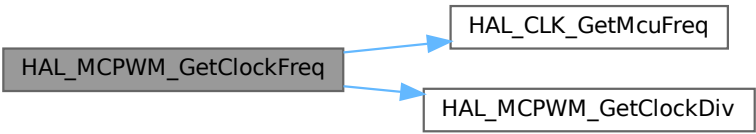


4.96.1.4 HAL_MCPWM_GetClockFreq()

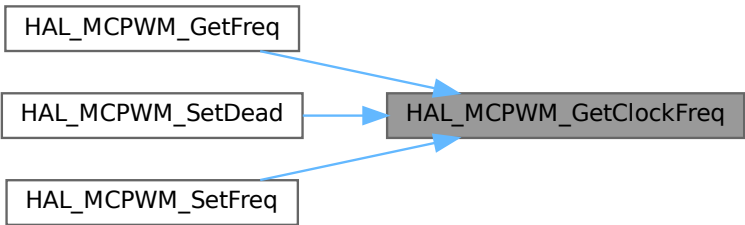
```
uint32_t HAL_MCPWM_GetClockFreq (
    HAL_MCPWM_x mcpwm_x )
```

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 284 行定义.

函数调用图:



这是这个函数的调用关系图:



4.96.1.5 HAL_MCPWM_GetFreq()

```
uint32_t HAL_MCPWM_GetFreq (
    HAL_MCPWM_X mcpwm_x )
```

获取pwm频率

参数

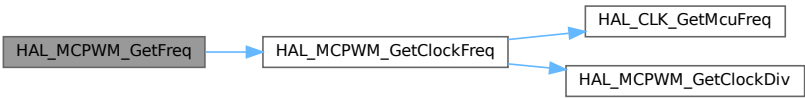
<i>mcpwm_x</i>	操作的模块编号

返回

当前pwm的频率

在文件 `hal_lks32mc08x_mcpwm.c` 第 255 行定义.

函数调用图:



4.96.1.6 HAL_MCPWM_GetThCnt()

```
uint32_t HAL_MCPWM_GetThCnt (
    HAL_MCPWM_x mcpwm_x )
```

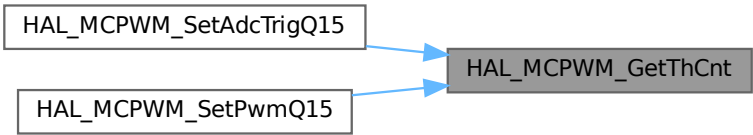
获取计数器最大值

参数

<i>mcpwm_x</i>	操作的模块编号

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 293 行定义.

这是这个函数的调用关系图:



4.96.1.7 HAL_MCPWM_Reset()

```
void HAL_MCPWM_Reset (
    HAL_MCPWM_x mcpwm_x )
```

复位mcpwm模块

参数

<i>mcpwm_x</i>	操作的模块编号

在文件 `hal_lks32mc08x_mcpwm.c` 第 163 行定义.

4.96.1.8 HAL_MCPWM_SetAdcTrig()

```
void HAL_MCPWM_SetAdcTrig (
    HAL_MCPWM_x mcpwm_x,
    int32_t t0,
    int32_t t1 )
```

设置pwm输出ADC采样时间

参数

<i>mcpwm_x</i>	操作的模块编号
<i>t0</i>	采样时间
<i>t1</i>	采样时间

在文件 `hal_lks32mc08x_mcpwm.c` 第 578 行定义.

这是这个函数的调用关系图:



4.96.1.9 HAL_MCPWM_SetAdcTrigQ15()

```
void HAL_MCPWM_SetAdcTrigQ15 (
    HAL_MCPWM_x mcpwm_x,
    int32_t t0,
    int32_t t1 )
```

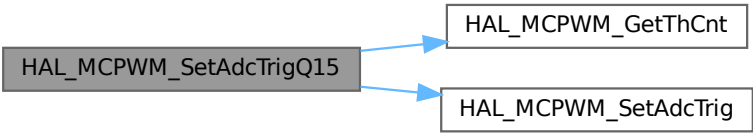
设置pwm输出ADC采样时间q15格式输入

参数

<i>mcpwm_x</i>	操作的模块编号
<i>t0</i>	采样时间q15格式输入
<i>t1</i>	采样时间q15格式输入

在文件 `hal_lks32mc08x_mcpwm.c` 第 589 行定义.

函数调用图:



4.96.1.10 HAL_MCPWM_SetClockDiv()

```
void HAL_MCPWM_SetClockDiv (
    HAL_MCPWM_x mcpwm_x,
    uint32_t div )
```

在文件 [hal_lks32mc08x.mcpwm.c](#) 第 265 行定义.

4.96.1.11 HAL_MCPWM_SetDead()

```
void HAL_MCPWM_SetDead (
    HAL_MCPWM_x mcpwm_x,
    uint32_t pdead_nS,
    uint32_t ndead_nS )
```

设置死区

参数

<i>mcpwm_x</i>	操作的模块编号
<i>pdead_nS</i>	下管关闭上管打开的死区宽度,单位nS
<i>ndead_nS</i>	上管关闭下管打开的死区宽度,单位nS

在文件 [hal_lks32mc08x.mcpwm.c](#) 第 304 行定义.

函数调用图:



4.96.1.12 HAL_MCPWM_SetDefaultState()

```
void HAL_MCPWM_SetDefaultState (
    HAL_MCPWM_X mcpwm_x,
    HAL_MCPWM_DEFAULT_OUTPUT_STATE_X mcpwm_default_output_state_x )
```

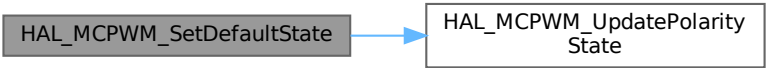
设置默认状态

参数

<i>mcpwm_x</i>	操作的模块编号
<i>mcpwm_default_output_state_x</i>	默认输出状态

在文件 [hal_lks32mc08x.mcpwm.c](#) 第 478 行定义.

函数调用图:



4.96.1.13 HAL_MCPWM_SetDrivePolarity()

```
void HAL_MCPWM_SetDrivePolarity (
    HAL_MCPWM_X mcpwm_x,
    HAL_MCPWM_POLARITY_X mcpwm_polarity_x )
```

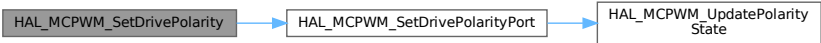
设置pwm极性

参数

<i>mcpwm_x</i>	操作的模块编号
<i>mcpwm_polarity_x</i>	第x个pwm输出端口

在文件 [hal_lks32mc08x.mcpwm.c](#) 第 418 行定义.

函数调用图:



4.96.1.14 HAL_MCPWM_SetDrivePolarityPort()

```
void HAL_MCPWM_SetDrivePolarityPort (
    HAL_MCPWM_x mcpwm_x,
    HAL_MCPWM_PORT_x mcpwm_port_x,
    HAL_MCPWM_POLARITY_x mcpwm_polarity_x )
```

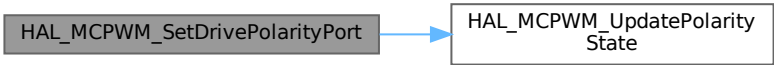
设置单个输出通道的pwm极性

参数

<i>mcpwm_x</i>	操作的模块编号
<i>mcpwm_port_x</i>	第x个pwm输出端口
<i>mcpwm_x</i> ↔ <i>polarity_x</i>	输出极性

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 431 行定义.

函数调用图:



这是这个函数的调用关系图:



4.96.1.15 HAL_MCPWM_SetFailFilter()

```
void HAL_MCPWM_SetFailFilter (
    HAL_MCPWM_x mcpwm_x,
    uint32_t cnt )
```

设置FAIL信号滤波宽度

参数

<i>mcpwm_x</i>	操作的模块编号
<i>cnt</i>	滤波宽度

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 602 行定义.

4.96.1.16 HAL_MCPWM_SetFailSignal()

```
void HAL_MCPWM_SetFailSignal (
    HAL_MCPWM_X mcpwm_x,
    HAL_MCPWM_FAIL_X fail_x,
    HAL_MCPWM_FAIL_IN_X fail_in_x )
```

设置FAIL信号来源

参数

<i>mcpwm_x</i>	操作的模块编号
<i>fail_x</i>	FAIL信号来源

- < 外部FAIL信号为高电平后mcpwm关闭输出
- < 外部FAIL信号为低电平后mcpwm关闭输出
- < CMP0为高电平后mcpwm关闭输出
- < CMP0为低电平后mcpwm关闭输出

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 536 行定义.

4.96.1.17 HAL_MCPWM_SetFreq()

```
void HAL_MCPWM_SetFreq (
    HAL_MCPWM_X mcpwm_x,
    uint32_t freq )
```

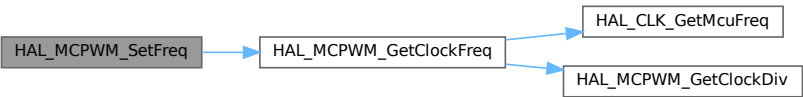
设置pwm频率

参数

<i>mcpwm_x</i>	操作的模块编号

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 228 行定义.

函数调用图:



4.96.1.18 HAL.MCPWM_SetGpioFail()

```
void HAL_MCPWM_SetGpioFail (
    HAL_MCPWM_X mcpwm_x,
    HAL_GPIO_Px px )
```

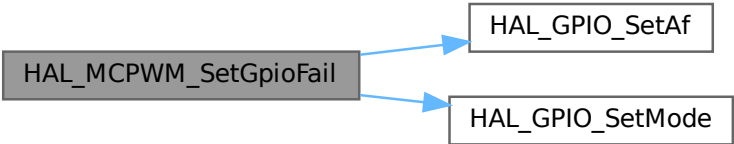
配置用于故障输入的gpio

参数

<i>mcpwm_x</i>	MCPWM模块编号
<i>px</i>	使用的GPIO口

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 524 行定义.

函数调用图:



4.96.1.19 HAL.MCPWM_SetGpioOut()

```
void HAL_MCPWM_SetGpioOut (
    HAL_MCPWM_X mcpwm_x,
    HAL_GPIO_Px px )
```

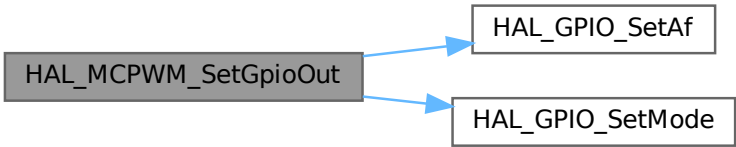
设置用于输出的gpio

参数

<i>mcpwm_x</i>	MCPWM模块编号
<i>px</i>	使用的GPIO口

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 513 行定义.

函数调用图:



4.96.1.20 HAL_MCPWM_SetOutputDisable()

```
void HAL_MCPWM_SetOutputDisable (
    HAL_MCPWM_x mcpwm_x )
```

关闭输出

参数

<i>mcpwm_x</i>	操作的模块编号

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 325 行定义.

4.96.1.21 HAL_MCPWM_SetOutputEnable()

```
void HAL_MCPWM_SetOutputEnable (
    HAL_MCPWM_x mcpwm_x )
```

打开输出

参数

<i>mcpwm_x</i>	操作的模块编号

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 336 行定义.

4.96.1.22 HAL_MCPWM_SetOutState()

```
void HAL_MCPWM_SetOutState (
    HAL_MCPWM_x mcpwm_x,
    HAL_MCPWM_PORT_x mcpwm_port_x,
    HAL_MCPWM_OUTPUT_STATE_x mcpwm_output_state_x )
```

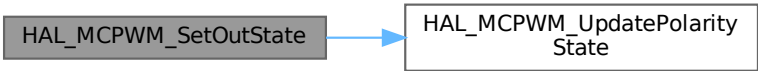
设置输出状态

参数

<i>mcpwm_x</i>	操作的模块编号
<i>mcpwm_port_x</i>	第x个pwm输出端口
<i>mcpwm_output_↔ state_x</i>	输出状态

在文件 [hal_lks32mc08x.mcpwm.c](#) 第 406 行定义.

函数调用图:



4.96.1.23 HAL_MCPWM_SetPwm()

```
void HAL_MCPWM_SetPwm (
    HAL_MCPWM_x mcpwm_x,
    HAL_MCPWM_PORT_x mcpwm_port_x,
    int32_t t0,
    int32_t t1 )
```

设置占空比

参数

<i>mcpwm_x</i>	操作的模块编号
<i>mcpwm_↔ port_x</i>	第x个pwm输出端口
<i>t0</i>	pwm打开时刻
<i>t1</i>	pwm关闭时刻

在文件 [hal_lks32mc08x.mcpwm.c](#) 第 386 行定义.

这是这个函数的调用关系图:

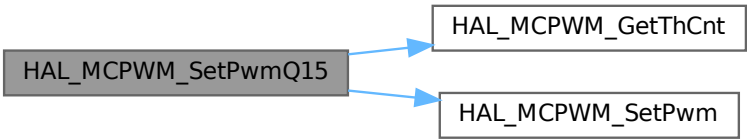


4.96.1.24 HAL_MCPWM_SetPwmQ15()

```
void HAL_MCPWM_SetPwmQ15 (
    HAL_MCPWM_x mcpwm_x,
    HAL_MCPWM_PORT_x mcpwm_port_x,
    int16_t t0,
    int16_t t1 )
```

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 392 行定义.

函数调用图:



4.96.1.25 HAL_MCPWM_SetRemap()

```
void HAL_MCPWM_SetRemap (
    HAL_MCPWM_x mcpwm_x,
    uint8_t mode )
```

MCPWM通道重映射(通道输出到GPIO的路由重排,与HAL_MCPWM_SetSwap的PN交换无关)

参数

<i>mcpwm_x</i>	操作的模块编号
<i>mode</i>	0:片外预驱(默认排布) 1:封装排布1 2:封装排布2

注解

本系列无通道重映射功能,本函数为空操作

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 467 行定义.

4.96.1.26 HAL_MCPWM_SetSwap()

```
void HAL_MCPWM_SetSwap (
    HAL_MCPWM_x mcpwm_x,
    HAL_MCPWM_SWAP_x mcpwm_swap_x )
```

设置PN是否交换

参数

<i>mcpwm_x</i>	操作的模块编号
<i>mcpwm_↔ swap_x</i>	PN是否交换

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 442 行定义.

函数调用图:



4.96.1.27 HAL_MCPWM_SetSwapPort()

```
void HAL_MCPWM_SetSwapPort (
    HAL_MCPWM_x mcpwm_x,
    HAL_MCPWM_PORT_x mcpwm_port_x,
    HAL_MCPWM_SWAP_x mcpwm_swap_x )
```

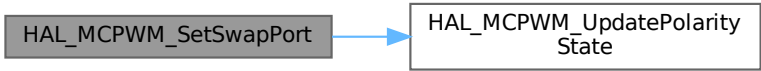
设置单个输出通道的PN是否交换

参数

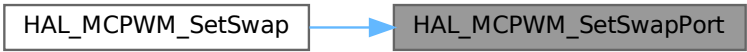
<i>mcpwm_x</i>	操作的模块编号
<i>mcpwm_port_x</i>	第x个pwm输出端口
<i>mcpwm_↔ swap_x</i>	PN是否交换

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 455 行定义.

函数调用图:



这是这个函数的调用关系图:



4.96.1.28 HAL_MCPWM_SetThCnt()

```
void HAL_MCPWM_SetThCnt (
    HAL_MCPWM_X mcpwm_x,
    uint32_t th )
```

设置计数器最大值

参数

<i>mcpwm_x</i>	操作的模块编号
<i>th</i>	最大值

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 215 行定义.

4.96.1.29 HAL_MCPWM_SetUpdateMode()

```
void HAL_MCPWM_SetUpdateMode (
    HAL_MCPWM_X mcpwm_x,
    HAL_MCPWM_UPDATE_X update_x,
    uint32_t intv )
```

设置影子寄存器自动更新时刻

参数

<i>mcpwm_x</i>	操作的模块编号
<i>update_x</i>	更新事件选择
<i>intv</i>	更新间隔0~15,每intv+1次更新事件加载一次,0表示每次事件都更新

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 617 行定义.

4.96.1.30 HAL_MCPWM_StartCounting()

```
void HAL_MCPWM_StartCounting (
```

`HAL_MCPWM_x mcpwm_x )`

开始计数

参数

<code>mcpwm_x</code>	操作的模块编号

在文件 `hal_lks32mc08x_mcpwm.c` 第 347 行定义.

这是这个函数的调用关系图:



4.96.1.31 HAL_MCPWM_StartCountingSync()

```
void HAL_MCPWM_StartCountingSync (
    HAL_MCPWM_x mcpwm1_x,
    HAL_MCPWM_x mcpwm2_x,
    int16_t phase )
```

跟随另一个计数器启动，周期不同时会自动映射相位差

参数

<code>mcpwm_x</code>	操作的模块编号
<code>mcpwm_x_followed</code>	被跟随的模块编号
<code>phase</code>	相位差，0表示同相位，-32768或32767对应正交（执行时所有mcpwm都会暂停）

在文件 `hal_lks32mc08x_mcpwm.c` 第 371 行定义.

函数调用图:



4.96.1.32 HAL_MCPWM_StopCounting()

```
void HAL_MCPWM_StopCounting (
    HAL_MCPWM_X mcpwm_x )
```

停止计数(同时计数器回到-th)

参数

mcpwm_x	操作的模块编号

在文件 [hal_lks32mc08x.mcpwm.c](#) 第 358 行定义.

4.96.1.33 HAL_MCPWM_UpdatePolarityState()

```
void HAL_MCPWM_UpdatePolarityState (
    HAL_MCPWM_X mcpwm_x,
    HAL_MCPWM_PORT_X mcpwm_port_x )
```

设置FAIL信号来源

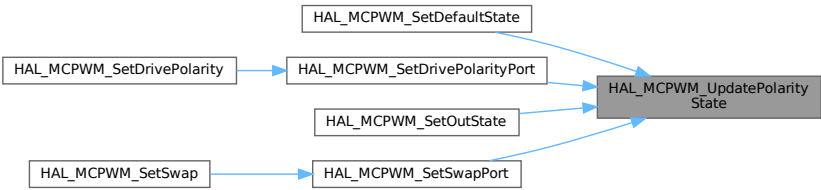
参数

mcpwm_x	操作的模块编号
fail_x	FAIL通道
fail_in_x	FAIL信号来源

- < 上下管对称pwm
- < 关闭输出
- < 下管恒开
- < 上管恒开
- < 下管pwm
- < 上管pwm

在文件 [hal_lks32mc08x.mcpwm.c](#) 第 17 行定义.

这是这个函数的调用关系图:



4.96.2 变量说明

4.96.2.1 HAL_MCPWM_STATE

```
uint16_t HAL_MCPWM_STATE[1][4] = {{0, 0, 0, 0}} [static]
```

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 8 行定义。

4.96.2.2 HAL_MCPWM_STATE_INIT_FLG

```
uint16_t HAL_MCPWM_STATE_INIT_FLG[1][4] = {{0, 0, 0, 0}} [static]
```

在文件 [hal_lks32mc08x_mcpwm.c](#) 第 9 行定义。

4.97 hal_lks32mc08x_mcpwm.c

[浏览该文件的文档。](#)

```
00001 #include "../include/hal_lksmcu_mcpwm.h"
00002 #include "../hal_lks32mc08x.h"
00003 #include "lks32mc08x_user_manual.h"
00004 #include "../include/hal_lksmcu_clk.h"
00005 #include "../include/hal_lksmcu_reg.h"
00006 #include "../include/hal_lks32mc08x_reg.h"
00007
00008 static uint16_t HAL_MCPWM_STATE[1][4] = {{0, 0, 0, 0}};
00009 static uint16_t HAL_MCPWM_STATE_INIT_FLG[1][4] = {{0, 0, 0, 0}}; // 初始化状态，只有对所有位都初始化完成后，才
    允许写入到Io中
00010
00017 void HAL_MCPWM_UpdatePolarityState(HAL_MCPWM_x mcpwm_x, HAL_MCPWM_PORT_x mcpwm_port_x)
00018 {
00019     uint8_t tmp[4] = {0, 0, 0, 0};
00020     uint8_t tmp1[4] = {0, 0, 0, 0};
00021     uint32_t io_mask;
00022     HAL_MCPWM_OUTPUT_STATE_x out_state;
00023
00024     if (HAL_REG_READ_BIT(HAL_MCPWM_STATE_INIT_FLG[0][mcpwm_port_x], HAL_MCPWM_STATE_SW_BIT) == 0)
00025     {
00026         return;
00027     }
00028     if (HAL_REG_READ_BIT(HAL_MCPWM_STATE_INIT_FLG[0][mcpwm_port_x], HAL_MCPWM_STATE_H_MOS_EN_BIT) == 0)
00029     {
00030         return;
00031     }
00032     if (HAL_REG_READ_BIT(HAL_MCPWM_STATE_INIT_FLG[0][mcpwm_port_x], HAL_MCPWM_STATE_L_MOS_EN_BIT) == 0)
00033     {
00034         return;
00035     }
00036     if (HAL_REG_READ_BIT(HAL_MCPWM_STATE_INIT_FLG[0][mcpwm_port_x], HAL_MCPWM_STATE_H_DEFAULT_BIT) == 0)
00037     {
00038         return;
00039     }
00040     if (HAL_REG_READ_BIT(HAL_MCPWM_STATE_INIT_FLG[0][mcpwm_port_x], HAL_MCPWM_STATE_L_DEFAULT_BIT) == 0)
00041     {
00042         return;
00043     }
00044     // 输出默认状态可以不配置，这里不对HAL_MCPWM_STATE_STATE_MASK进行判断，默认上下管上下管对称pwm
00045     // if (HAL_REG_READ_BIT(HAL_MCPWM_STATE_INIT_FLG[0][mcpwm_port_x], HAL_MCPWM_STATE_STATE_MASK) == 0)
00046     // {
00047     //     return;
00048     // }
00049     // MCPWMx_IO01_CH0_PP_BIT MCPWMx_IO01_CH0_NP_BIT 是负逻辑
00050     HAL_REG_IF_WRITE(tmp1[mcpwm_port_x], MCPWM_IO01_CH0_PP_SW_BIT, (HAL_MCPWM_STATE[0][mcpwm_port_x] &
    HAL_MCPWM_STATE_SW_BIT)); // 打开通道交换
00051     HAL_REG_IF_WRITE(tmp1[mcpwm_port_x], MCPWM_IO01_CH0_PP_BIT, !(HAL_MCPWM_STATE[0][mcpwm_port_x] &
    HAL_MCPWM_STATE_H_MOS_EN_BIT)); // 上管高有效
00052     HAL_REG_IF_WRITE(tmp1[mcpwm_port_x], MCPWM_IO01_CH0_NP_BIT, !(HAL_MCPWM_STATE[0][mcpwm_port_x] &
    HAL_MCPWM_STATE_L_MOS_EN_BIT)); // 下管高有效
00053
00054     out_state = (HAL_MCPWM_OUTPUT_STATE_x) HAL_REG_READ_BIT_MASK_POS(HAL_MCPWM_STATE[0][mcpwm_port_x],
    HAL_MCPWM_STATE_STATE);
00055     switch (out_state)
```

```

00056     {
00057         case HAL_MCPWM_OUTPUT_STATE_PWM:
00058             tmp1[mcpwm_port_x] |= 0;
00059             break;
00060         case HAL_MCPWM_OUTPUT_STATE_OFF:
00061             tmp1[mcpwm_port_x] |= MCPWM_IO23_CH2_NS_BIT | MCPWM_IO23_CH2_PS_BIT;
00062             break;
00063         case HAL_MCPWM_OUTPUT_STATE_LOW:
00064             tmp1[mcpwm_port_x] |= MCPWM_IO23_CH2_NS_BIT | MCPWM_IO23_CH2_PS_BIT |
MCPWM_IO23_CH2_SCTRLP_BIT;
00065             break;
00066         case HAL_MCPWM_OUTPUT_STATE_HIGH:
00067             tmp1[mcpwm_port_x] |= MCPWM_IO23_CH2_NS_BIT | MCPWM_IO23_CH2_PS_BIT |
MCPWM_IO23_CH2_SCTRLP_BIT;
00068             break;
00069         case HAL_MCPWM_OUTPUT_STATE_LOW_PWM:
00070             tmp1[mcpwm_port_x] |= MCPWM_IO23_CH2_PS_BIT;
00071             break;
00072         case HAL_MCPWM_OUTPUT_STATE_HIGH_PWM:
00073             tmp1[mcpwm_port_x] |= MCPWM_IO23_CH2_NS_BIT;
00074             break;
00075     }
00076
00077     if (HAL_REG_READ_BIT(HAL_MCPWM_STATE[0][mcpwm_port_x], HAL_MCPWM_STATE_L_DEFAULT_BIT))
00078     {
00079         // 关机后上管打开
00080         if (HAL_REG_READ_BIT(HAL_MCPWM_STATE[0][mcpwm_port_x], HAL_MCPWM_STATE_SW_BIT)) // gpio交换使能
00081         {
00082             HAL_REG_IF_WRITE(tmp[mcpwm_port_x], MCPWM_IO01_CH0_NP_BIT, (HAL_MCPWM_STATE[0][mcpwm_port_x] &
HAL_MCPWM_STATE_H_MOS_EN_BIT)); // CH0 N 通道默认值
00083             HAL_REG_IF_WRITE(tmp[mcpwm_port_x], MCPWM_IO01_CH0_PP_BIT, !(HAL_MCPWM_STATE[0][mcpwm_port_x] &
HAL_MCPWM_STATE_L_MOS_EN_BIT)); // CH0 P 通道默认值
00084         }
00085         else
00086         {
00087             HAL_REG_IF_WRITE(tmp[mcpwm_port_x], MCPWM_IO01_CH0_NP_BIT, !(HAL_MCPWM_STATE[0][mcpwm_port_x] &
HAL_MCPWM_STATE_L_MOS_EN_BIT)); // CH0 N 通道默认值
00088             HAL_REG_IF_WRITE(tmp[mcpwm_port_x], MCPWM_IO01_CH0_PP_BIT, (HAL_MCPWM_STATE[0][mcpwm_port_x] &
HAL_MCPWM_STATE_H_MOS_EN_BIT)); // CH0 P 通道默认值
00089         }
00090     }
00091     else if (HAL_REG_READ_BIT(HAL_MCPWM_STATE[0][mcpwm_port_x], HAL_MCPWM_STATE_H_DEFAULT_BIT))
00092     {
00093         // 关机后下管打开
00094         if (HAL_REG_READ_BIT(HAL_MCPWM_STATE[0][mcpwm_port_x], HAL_MCPWM_STATE_SW_BIT)) // gpio交换使能
00095         {
00096             HAL_REG_IF_WRITE(tmp[mcpwm_port_x], MCPWM_IO01_CH0_NP_BIT, !(HAL_MCPWM_STATE[0][mcpwm_port_x] &
HAL_MCPWM_STATE_H_MOS_EN_BIT)); // CH0 N 通道默认值
00097             HAL_REG_IF_WRITE(tmp[mcpwm_port_x], MCPWM_IO01_CH0_PP_BIT, (HAL_MCPWM_STATE[0][mcpwm_port_x] &
HAL_MCPWM_STATE_L_MOS_EN_BIT)); // CH0 P 通道默认值
00098         }
00099         else
00100         {
00101             HAL_REG_IF_WRITE(tmp[mcpwm_port_x], MCPWM_IO01_CH0_NP_BIT, (HAL_MCPWM_STATE[0][mcpwm_port_x] &
HAL_MCPWM_STATE_L_MOS_EN_BIT)); // CH0 N 通道默认值
00102             HAL_REG_IF_WRITE(tmp[mcpwm_port_x], MCPWM_IO01_CH0_PP_BIT, !(HAL_MCPWM_STATE[0][mcpwm_port_x] &
HAL_MCPWM_STATE_H_MOS_EN_BIT)); // CH0 P 通道默认值
00103         }
00104     }
00105     else
00106     {
00107         // 关机后gpio悬空
00108         // 关机后下管打开
00109         if (HAL_REG_READ_BIT(HAL_MCPWM_STATE[0][mcpwm_port_x], HAL_MCPWM_STATE_SW_BIT)) // gpio交换使能
00110         {
00111             HAL_REG_IF_WRITE(tmp[mcpwm_port_x], MCPWM_IO01_CH0_NP_BIT, !(HAL_MCPWM_STATE[0][mcpwm_port_x] &
HAL_MCPWM_STATE_H_MOS_EN_BIT)); // CH0 N 通道默认值
00112             HAL_REG_IF_WRITE(tmp[mcpwm_port_x], MCPWM_IO01_CH0_PP_BIT, !(HAL_MCPWM_STATE[0][mcpwm_port_x] &
HAL_MCPWM_STATE_L_MOS_EN_BIT)); // CH0 P 通道默认值
00113         }
00114         else
00115         {
00116             HAL_REG_IF_WRITE(tmp[mcpwm_port_x], MCPWM_IO01_CH0_NP_BIT, !(HAL_MCPWM_STATE[0][mcpwm_port_x] &
HAL_MCPWM_STATE_H_MOS_EN_BIT)); // CH0 N 通道默认值
00117             HAL_REG_IF_WRITE(tmp[mcpwm_port_x], MCPWM_IO01_CH0_PP_BIT, !(HAL_MCPWM_STATE[0][mcpwm_port_x] &
HAL_MCPWM_STATE_H_MOS_EN_BIT)); // CH0 P 通道默认值
00118         }
00119     }
00120
00121     io_mask = MCPWM_IO01_CH0_PN_SW_BIT |
00122             MCPWM_IO01_CH0_SCTRLP_BIT |
00123             MCPWM_IO01_CH0_SCTRLN_BIT |
00124             MCPWM_IO01_CH0_PS_BIT |
00125             MCPWM_IO01_CH0_NS_BIT |
00126             MCPWM_IO01_CH0_PP_BIT |
00127             MCPWM_IO01_CH0_NP_BIT;
00128

```

```

00129     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x & 0xf] -> PRT, 0xdead);
00130     switch (mcpwm_port_x)
00131     {
00132         case HAL_MCPWM_PORT_0:
00133             HAL_REG_IF_WRITE(HAL_MCPWM[mcpwm_x & 0xf] -> FAIL, MCPWM_FAIL_CH0N_DEFAULT_BIT, tmp[0] &
MCPWM_IO01_CH0_NP_BIT); // 默认输出电平
00134             HAL_REG_IF_WRITE(HAL_MCPWM[mcpwm_x & 0xf] -> FAIL, MCPWM_FAIL_CH0P_DEFAULT_BIT, tmp[0] &
MCPWM_IO01_CH0_PP_BIT); // 默认输出电平
00135             HAL_REG_WRITE_BIT(HAL_MCPWM[mcpwm_x & 0xf] -> IO01, io_mask, tmp[0]);
00136             break;
00137         case HAL_MCPWM_PORT_1:
00138             HAL_REG_IF_WRITE(HAL_MCPWM[mcpwm_x & 0xf] -> FAIL, MCPWM_FAIL_CH1N_DEFAULT_BIT, tmp[1] &
MCPWM_IO01_CH0_NP_BIT); // 默认输出电平
00139             HAL_REG_IF_WRITE(HAL_MCPWM[mcpwm_x & 0xf] -> FAIL, MCPWM_FAIL_CH1P_DEFAULT_BIT, tmp[1] &
MCPWM_IO01_CH0_PP_BIT); // 默认输出电平
00140             HAL_REG_WRITE_BIT(HAL_MCPWM[mcpwm_x & 0xf] -> IO01, io_mask << 8, tmp[1] << 8);
00141             break;
00142         case HAL_MCPWM_PORT_2:
00143             HAL_REG_IF_WRITE(HAL_MCPWM[mcpwm_x & 0xf] -> FAIL, MCPWM_FAIL_CH2N_DEFAULT_BIT, tmp[2] &
MCPWM_IO01_CH0_NP_BIT); // 默认输出电平
00144             HAL_REG_IF_WRITE(HAL_MCPWM[mcpwm_x & 0xf] -> FAIL, MCPWM_FAIL_CH2P_DEFAULT_BIT, tmp[2] &
MCPWM_IO01_CH0_PP_BIT); // 默认输出电平
00145             HAL_REG_WRITE_BIT(HAL_MCPWM[mcpwm_x & 0xf] -> IO23, io_mask, tmp[2]);
00146             break;
00147         case HAL_MCPWM_PORT_3:
00148             HAL_REG_IF_WRITE(HAL_MCPWM[mcpwm_x & 0xf] -> FAIL, MCPWM_FAIL_CH3N_DEFAULT_BIT, tmp[3] &
MCPWM_IO01_CH0_NP_BIT); // 默认输出电平
00149             HAL_REG_IF_WRITE(HAL_MCPWM[mcpwm_x & 0xf] -> FAIL, MCPWM_FAIL_CH3P_DEFAULT_BIT, tmp[3] &
MCPWM_IO01_CH0_PP_BIT); // 默认输出电平
00150             HAL_REG_WRITE_BIT(HAL_MCPWM[mcpwm_x & 0xf] -> IO23, io_mask << 8, tmp[3] << 8);
00151             break;
00152         default:
00153             break;
00154     }
00155     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x & 0xf] -> PRT, 0);
00157 }
00158
00163 void HAL_MCPWM_Reset(HAL_MCPWM_x mcpwm_x)
00164 {
00165     (void)mcpwm_x;
00166     // 系统寄存器解锁
00167     HAL_REG_SYS_UNLOCK();
00168     HAL_REG_SET(SYS_SFT_RST, SYS_SFT_RST_MCPWM_SFT_RST_BIT);
00169     HAL_REG_RESET(SYS_SFT_RST, SYS_SFT_RST_MCPWM_SFT_RST_BIT);
00170     // 系统寄存器上锁
00171     HAL_REG_SYS_LOCK();
00172 }
00173
00178 void HAL_MCPWM_Enable(HAL_MCPWM_x mcpwm_x)
00179 {
00180     // 系统寄存器解锁
00181     HAL_REG_SYS_UNLOCK();
00182     // SYS_CLK_FEN[3] MCPWM 模块整体时钟控制信号
00183     HAL_REG_SET(SYS_CLK_FEN, SYS_CLK_FEN_MCPWM_CLK_EN_BIT);
00184     // 系统寄存器上锁
00185     HAL_REG_SYS_LOCK();
00186
00187     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0xdead);
00188     HAL_REG_SET(HAL_MCPWM[mcpwm_x] -> TCLK, MCPWM_TCLK_CLK_EN_BIT);
00189     HAL_REG_SET(HAL_MCPWM[mcpwm_x] -> SDCFG, MCPWM_SDCFG_T0_UPDATE_EN_BIT);
00190     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0);
00191 }
00192
00197 void HAL_MCPWM_Disable(HAL_MCPWM_x mcpwm_x)
00198 {
00199     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0xdead);
00200     HAL_REG_RESET(HAL_MCPWM[mcpwm_x] -> TCLK, MCPWM_TCLK_CLK_EN_BIT);
00201     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0);
00202     // 系统寄存器解锁
00203     HAL_REG_SYS_UNLOCK();
00204     // SYS_CLK_FEN[3] MCPWM 模块整体时钟控制信号
00205     HAL_REG_RESET(SYS_CLK_FEN, SYS_CLK_FEN_MCPWM_CLK_EN_BIT);
00206     // 系统寄存器上锁
00207     HAL_REG_SYS_LOCK();
00208 }
00209
00215 void HAL_MCPWM_SetThCnt(HAL_MCPWM_x mcpwm_x, uint32_t th)
00216 {
00217     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x & 0xf] -> PRT, 0xdead);
00218
00219     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x & 0xf] -> TH0, th);
00220     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x & 0xf] -> UPDATE, MCPWM_UPDATE_TH_UPDATE_BIT);
00221
00222     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x & 0xf] -> PRT, 0);
00223 }
00228 void HAL_MCPWM_SetFreq(HAL_MCPWM_x mcpwm_x, uint32_t freq)

```

```

00229 {
00230     uint32_t mclk = HAL_MCPWM_GetClockFreq(mcpwm_x);
00231     uint32_t th = ((mclk / freq) - 1) / 2;
00232     uint32_t div = 0;
00233     for (; th > MCPWM_TH_TH_MAX; )
00234     {
00235         th = th >> 1;
00236         div++;
00237         if (div > MCPWM_TCLK_CLK_DIV_MAX)
00238         {
00239             th = MCPWM_TH_TH_MAX;
00240             div = MCPWM_TCLK_CLK_DIV_MAX;
00241         }
00242     }
00243     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0xdead);
00244     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> TH0, th);
00245     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> UPDATE, MCPWM_UPDATE_TH_UPDATE_BIT);
00246     HAL_REG_WRITE_BIT_MASK_POS(HAL_MCPWM[mcpwm_x] -> TCLK, MCPWM_TCLK_CLK_DIV, div);
00247     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0);
00248 }
00249
00255 uint32_t HAL_MCPWM_GetFreq(HAL_MCPWM_x mcpwm_x)
00256 {
00257     uint32_t mclk = HAL_MCPWM_GetClockFreq(mcpwm_x);
00258     uint16_t th = HAL_REG_READ(HAL_MCPWM[mcpwm_x] -> TH0);
00259     uint32_t div = HAL_REG_READ_BIT_MASK_POS(HAL_MCPWM[mcpwm_x] -> TCLK, MCPWM_TCLK_CLK_DIV);
00260     uint32_t freq = (mclk >> div) / (th * 2 + 1);
00261     return freq;
00262 }
00263
00264 // 设置时钟分频
00265 void HAL_MCPWM_SetClockDiv(HAL_MCPWM_x mcpwm_x, uint32_t div)
00266 {
00267     uint32_t shift = 0;
00268     while ((1u << shift) < div && shift < MCPWM_TCLK_CLK_DIV_MAX)
00269     {
00270         shift++;
00271     }
00272     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] & 0xf -> PRT, 0xdead);
00273     HAL_REG_WRITE_BIT_MASK_POS(HAL_MCPWM[mcpwm_x] & 0xf -> TCLK, MCPWM_TCLK_CLK_DIV, shift);
00274     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] & 0xf -> PRT, 0);
00275 }
00276
00277 // 获取时钟分频实际除数
00278 uint32_t HAL_MCPWM_GetClockDiv(HAL_MCPWM_x mcpwm_x)
00279 {
00280     return 1u << HAL_REG_READ_BIT_MASK_POS(HAL_MCPWM[mcpwm_x] & 0xf -> TCLK, MCPWM_TCLK_CLK_DIV);
00281 }
00282
00283 // 获取分频后工作时钟
00284 uint32_t HAL_MCPWM_GetClockFreq(HAL_MCPWM_x mcpwm_x)
00285 {
00286     return HAL_CLK_GetMcuFreq() / HAL_MCPWM_GetClockDiv(mcpwm_x);
00287 }
00288
00293 uint32_t HAL_MCPWM_GetThCnt(HAL_MCPWM_x mcpwm_x)
00294 {
00295     return HAL_REG_READ(HAL_MCPWM[mcpwm_x] -> TH0);
00296 }
00297
00304 void HAL_MCPWM_SetDead(HAL_MCPWM_x mcpwm_x, uint32_t pdead_ns, uint32_t ndead_ns)
00305 {
00306     uint32_t mclk = HAL_MCPWM_GetClockFreq(mcpwm_x);
00307     uint16_t pdead = (int64_t)pdead_ns * mclk / 1000000000;
00308     uint16_t ndead = (int64_t)ndead_ns * mclk / 1000000000;
00309     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0xdead);
00310     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> DTH0, pdead);
00311     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> DTH10, pdead);
00312     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> DTH20, pdead);
00313     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> DTH30, pdead);
00314     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> DTH01, ndead);
00315     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> DTH11, ndead);
00316     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> DTH21, ndead);
00317     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> DTH31, ndead);
00318     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0);
00319 }
00320
00325 void HAL_MCPWM_SetOutputDisable(HAL_MCPWM_x mcpwm_x)
00326 {
00327     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0xdead);
00328     HAL_REG_RESET(HAL_MCPWM[mcpwm_x] -> FAIL, MCPWM_FAIL_MCPWM_OE_BIT);
00329     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0);
00330 }
00331
00336 void HAL_MCPWM_SetOutputEnable(HAL_MCPWM_x mcpwm_x)
00337 {
00338     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0xdead);

```



```

00339     HAL_REG_SET(HAL_MCPWM[mcpwm_x] -> FAIL, MCPWM_FAIL_MCPWM_OE_BIT);
00340     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0);
00341 }
00342
00347 void HAL_MCPWM_StartCounting(HAL_MCPWM_x mcpwm_x)
00348 {
00349     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0xdead);
00350     HAL_REG_SET(HAL_MCPWM[mcpwm_x] -> TCLK, MCPWM_TCLK_BASE_CNT_EN_BIT);
00351     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0);
00352 }
00353
00358 void HAL_MCPWM_StopCounting(HAL_MCPWM_x mcpwm_x)
00359 {
00360     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0xdead);
00361     HAL_REG_RESET(HAL_MCPWM[mcpwm_x] -> TCLK, MCPWM_TCLK_BASE_CNT_EN_BIT);
00362     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0);
00363 }
00364
00371 void HAL_MCPWM_StartCountingSync(HAL_MCPWM_x mcpwm1_x, HAL_MCPWM_x mcpwm2_x, int16_t phase)
00372 {
00373     (void)mcpwm2_x;
00374     (void)phase;
00375     // 只有一个cnt, 这里直接启动
00376     HAL_MCPWM_StartCounting(mcpwm1_x);
00377     return;
00378 }
00386 void HAL_MCPWM_SetPwm(HAL_MCPWM_x mcpwm_x, HAL_MCPWM_PORT_x mcpwm_port_x, int32_t t0, int32_t t1)
00387 {
00388     // 使用公式写可以在内联的情况下, 实现和直接配置寄存器一样的效果
00389     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> TH[mcpwm_port_x * 2], t0);
00390     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> TH[mcpwm_port_x * 2 + 1], t1);
00391 }
00392 void HAL_MCPWM_SetPwmQ15(HAL_MCPWM_x mcpwm_x, HAL_MCPWM_PORT_x mcpwm_port_x, int16_t t0, int16_t t1)
00393 {
00394     uint32_t th = HAL_MCPWM_GetThCnt(mcpwm_x);
00395     int32_t cnt0 = (t0 * th) >> 15;
00396     int32_t cnt1 = (t1 * th) >> 15;
00397     HAL_MCPWM_SetPwm(mcpwm_x, mcpwm_port_x, cnt0, cnt1);
00398 }
00399
00406 void HAL_MCPWM_SetOutState(HAL_MCPWM_x mcpwm_x, HAL_MCPWM_PORT_x mcpwm_port_x, HAL_MCPWM_OUTPUT_STATE_x
mcpwm_output_state_x)
00407 {
00408     HAL_REG_WRITE_BIT_MASK_POS(HAL_MCPWM_STATE[mcpwm_x & 0xf][mcpwm_port_x], HAL_MCPWM_STATE_STATE,
mcpwm_output_state_x);
00409     HAL_REG_SET(HAL_MCPWM_STATE_INIT_FLG[mcpwm_x & 0xf][mcpwm_port_x], HAL_MCPWM_STATE_STATE_MASK);
00410     HAL_MCPWM_UpdatePolarityState(mcpwm_x, mcpwm_port_x);
00411 }
00412
00418 void HAL_MCPWM_SetDrivePolarity(HAL_MCPWM_x mcpwm_x, HAL_MCPWM_POLARITY_x mcpwm_polarity_x)
00419 {
00420     HAL_MCPWM_SetDrivePolarityPort(mcpwm_x, HAL_MCPWM_PORT_0, mcpwm_polarity_x);
00421     HAL_MCPWM_SetDrivePolarityPort(mcpwm_x, HAL_MCPWM_PORT_1, mcpwm_polarity_x);
00422     HAL_MCPWM_SetDrivePolarityPort(mcpwm_x, HAL_MCPWM_PORT_2, mcpwm_polarity_x);
00423     HAL_MCPWM_SetDrivePolarityPort(mcpwm_x, HAL_MCPWM_PORT_3, mcpwm_polarity_x);
00424 }
00431 void HAL_MCPWM_SetDrivePolarityPort(HAL_MCPWM_x mcpwm_x, HAL_MCPWM_PORT_x mcpwm_port_x,
HAL_MCPWM_POLARITY_x mcpwm_polarity_x)
00432 {
00433     HAL_REG_WRITE_BIT(HAL_MCPWM_STATE[mcpwm_x & 0xf][mcpwm_port_x], HAL_MCPWM_STATE_H_MOS_EN_BIT |
HAL_MCPWM_STATE_L_MOS_EN_BIT, mcpwm_polarity_x << 1);
00434     HAL_REG_SET(HAL_MCPWM_STATE_INIT_FLG[mcpwm_x & 0xf][mcpwm_port_x], HAL_MCPWM_STATE_H_MOS_EN_BIT |
HAL_MCPWM_STATE_L_MOS_EN_BIT);
00435     HAL_MCPWM_UpdatePolarityState(mcpwm_x, mcpwm_port_x);
00436 }
00442 void HAL_MCPWM_SetSwap(HAL_MCPWM_x mcpwm_x, HAL_MCPWM_SWAP_x mcpwm_swap_x)
00443 {
00444     HAL_MCPWM_SetSwapPort(mcpwm_x, HAL_MCPWM_PORT_0, mcpwm_swap_x);
00445     HAL_MCPWM_SetSwapPort(mcpwm_x, HAL_MCPWM_PORT_1, mcpwm_swap_x);
00446     HAL_MCPWM_SetSwapPort(mcpwm_x, HAL_MCPWM_PORT_2, mcpwm_swap_x);
00447     HAL_MCPWM_SetSwapPort(mcpwm_x, HAL_MCPWM_PORT_3, mcpwm_swap_x);
00448 }
00455 void HAL_MCPWM_SetSwapPort(HAL_MCPWM_x mcpwm_x, HAL_MCPWM_PORT_x mcpwm_port_x, HAL_MCPWM_SWAP_x
mcpwm_swap_x)
00456 {
00457     HAL_REG_IF_WRITE(HAL_MCPWM_STATE[mcpwm_x & 0xf][mcpwm_port_x], HAL_MCPWM_STATE_SW_BIT, mcpwm_swap_x);
00458     HAL_REG_SET(HAL_MCPWM_STATE_INIT_FLG[mcpwm_x & 0xf][mcpwm_port_x], HAL_MCPWM_STATE_SW_BIT);
00459     HAL_MCPWM_UpdatePolarityState(mcpwm_x, mcpwm_port_x);
00460 }
00467 void HAL_MCPWM_SetRemap(HAL_MCPWM_x mcpwm_x, uint8_t mode)
00468 {
00469     (void)mcpwm_x;
00470     (void)mode;
00471 }
00472
00478 void HAL_MCPWM_SetDefaultState(HAL_MCPWM_x mcpwm_x, HAL_MCPWM_DEFAULT_OUTPUT_STATE_x
mcpwm_default_output_state_x)

```

```

00479 {
00480     uint32_t val = 0;
00481     switch (mcpwm.default.output.state.x)
00482     {
00483         case HAL_MCPWM_DEFAULT_OUTPUT_STATE_OFF:
00484             break;
00485         case HAL_MCPWM_DEFAULT_OUTPUT_STATE_LOW:
00486             HAL_REG_SET(val, HAL_MCPWM_STATE_L_DEFAULT_BIT);
00487             break;
00488         case HAL_MCPWM_DEFAULT_OUTPUT_STATE_HIGH:
00489             HAL_REG_SET(val, HAL_MCPWM_STATE_H_DEFAULT_BIT);
00490             break;
00491     }
00492     HAL_REG_WRITE_BIT(HAL_MCPWM_STATE[mcpwm.x & 0xf][HAL_MCPWM_PORT_0], HAL_MCPWM_STATE_H_DEFAULT_BIT |
HAL_MCPWM_STATE_L_DEFAULT_BIT, val);
00493     HAL_REG_WRITE_BIT(HAL_MCPWM_STATE[mcpwm.x & 0xf][HAL_MCPWM_PORT_1], HAL_MCPWM_STATE_H_DEFAULT_BIT |
HAL_MCPWM_STATE_L_DEFAULT_BIT, val);
00494     HAL_REG_WRITE_BIT(HAL_MCPWM_STATE[mcpwm.x & 0xf][HAL_MCPWM_PORT_2], HAL_MCPWM_STATE_H_DEFAULT_BIT |
HAL_MCPWM_STATE_L_DEFAULT_BIT, val);
00495     HAL_REG_WRITE_BIT(HAL_MCPWM_STATE[mcpwm.x & 0xf][HAL_MCPWM_PORT_3], HAL_MCPWM_STATE_H_DEFAULT_BIT |
HAL_MCPWM_STATE_L_DEFAULT_BIT, val);
00496
00497     HAL_REG_SET(HAL_MCPWM_STATE_INIT_FLG[mcpwm.x & 0xf][HAL_MCPWM_PORT_0], HAL_MCPWM_STATE_H_DEFAULT_BIT |
HAL_MCPWM_STATE_L_DEFAULT_BIT);
00498     HAL_REG_SET(HAL_MCPWM_STATE_INIT_FLG[mcpwm.x & 0xf][HAL_MCPWM_PORT_1], HAL_MCPWM_STATE_H_DEFAULT_BIT |
HAL_MCPWM_STATE_L_DEFAULT_BIT);
00499     HAL_REG_SET(HAL_MCPWM_STATE_INIT_FLG[mcpwm.x & 0xf][HAL_MCPWM_PORT_2], HAL_MCPWM_STATE_H_DEFAULT_BIT |
HAL_MCPWM_STATE_L_DEFAULT_BIT);
00500     HAL_REG_SET(HAL_MCPWM_STATE_INIT_FLG[mcpwm.x & 0xf][HAL_MCPWM_PORT_3], HAL_MCPWM_STATE_H_DEFAULT_BIT |
HAL_MCPWM_STATE_L_DEFAULT_BIT);
00501
00502     HAL_MCPWM_UpdatePolarityState(mcpwm.x, HAL_MCPWM_PORT_0);
00503     HAL_MCPWM_UpdatePolarityState(mcpwm.x, HAL_MCPWM_PORT_1);
00504     HAL_MCPWM_UpdatePolarityState(mcpwm.x, HAL_MCPWM_PORT_2);
00505     HAL_MCPWM_UpdatePolarityState(mcpwm.x, HAL_MCPWM_PORT_3);
00506 }
00507
00513 void HAL_MCPWM_SetGpioOut(HAL_MCPWM_x mcpwm_x, HAL_GPIO_Px px)
00514 {
00515     (void)mcpwm_x;
00516     HAL_GPIO_SetAf(px, HAL_GPIO_AF_MCPWM);
00517     HAL_GPIO_SetMode(px, HAL_GPIO_MODE_OUT);
00518 }
00524 void HAL_MCPWM_SetGpioFail(HAL_MCPWM_x mcpwm_x, HAL_GPIO_Px px)
00525 {
00526     (void)mcpwm_x;
00527     HAL_GPIO_SetAf(px, HAL_GPIO_AF_MCPWM);
00528     HAL_GPIO_SetMode(px, HAL_GPIO_MODE_IN);
00529 }
00530
00536 void HAL_MCPWM_SetFailSignal(HAL_MCPWM_x mcpwm_x, HAL_MCPWM_FAIL_x fail_x, HAL_MCPWM_FAIL_IN_x fail_in_x)
00537 {
00538     uint32_t fail_mask;
00539     uint32_t fail_val;
00540     fail_mask = MCPWM_FAIL_FAIL0_EN_BIT | MCPWM_FAIL_FAIL0_POL_BIT | MCPWM_FAIL_FAIL0_SEL_BIT;
00541
00542     switch (fail_in_x)
00543     {
00544         case HAL_MCPWM_FAIL_IN_IO_HIGH:
00545             fail_val = MCPWM_FAIL_FAIL0_EN_BIT;
00546             break;
00547         case HAL_MCPWM_FAIL_IN_IO_LOW:
00548             fail_val = MCPWM_FAIL_FAIL0_EN_BIT | MCPWM_FAIL_FAIL0_POL_BIT;
00549             break;
00550         case HAL_MCPWM_FAIL_IN_CMP0_HIGH:
00551             fail_val = MCPWM_FAIL_FAIL0_EN_BIT | MCPWM_FAIL_FAIL0_SEL_BIT;
00552             break;
00553         case HAL_MCPWM_FAIL_IN_CMP0_LOW:
00554             fail_val = MCPWM_FAIL_FAIL0_EN_BIT | MCPWM_FAIL_FAIL0_POL_BIT | MCPWM_FAIL_FAIL0_SEL_BIT;
00555             break;
00556         default:
00557             fail_val = 0;
00558             break;
00559     }
00560     if (fail_x == HAL_MCPWM_FAIL_1)
00561     {
00562         fail_val <<= 1;
00563         fail_mask <<= 1;
00564     }
00565     if (mcpwm_x == HAL_MCPWM_0)
00566     {
00567         HAL_REG_WRITE(HAL_MCPWM[mcpwm_x & 0xf]->PRT, 0xdead);
00568         HAL_REG_WRITE_BIT(HAL_MCPWM[mcpwm_x & 0xf]->FAIL, fail_mask, fail_val);
00569         HAL_REG_WRITE(HAL_MCPWM[mcpwm_x & 0xf]->PRT, 0);
00570     }
00571 }
00578 void HAL_MCPWM_SetAdcTrig(HAL_MCPWM_x mcpwm_x, int32_t t0, int32_t t1)

```

```

00579 {
00580     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x]->TMR0, t0);
00581     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x]->TMR1, t1);
00582 }
00589 void HAL_MCPWM_SetAdcTrigQ15(HAL_MCPWM_x mcpwm_x, int32_t t0, int32_t t1)
00590 {
00591     uint32_t th = HAL_MCPWM_GetThCnt(mcpwm_x);
00592     int16_t tmr0 = ((int32_t)t0 * th) >> 15;
00593     int16_t tmr1 = ((int32_t)t1 * th) >> 15;
00594     HAL_MCPWM_SetAdcTrig(mcpwm_x, tmr0, tmr1);
00595 }
00596
00602 void HAL_MCPWM_SetFailFilter(HAL_MCPWM_x mcpwm_x, uint32_t cnt)
00603 {
00604     cnt >= 4;
00605     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] & 0xf]->PRT, 0xdead);
00606     HAL_REG_WRITE_BIT_MASK_POS(HAL_MCPWM[mcpwm_x] & 0xf]->TCLK, MCPWM_TCLK_IO_FLT_CLKDIV, cnt);
00607     HAL_REG_WRITE_BIT_MASK_POS(HAL_MCPWM[mcpwm_x] & 0xf]->TCLK, MCPWM_TCLK_CMP_FLT_CLKDIV, cnt);
00608     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] & 0xf]->PRT, 0);
00609 }
00610
00617 void HAL_MCPWM_SetUpdateMode(HAL_MCPWM_x mcpwm_x, HAL_MCPWM_UPDATE_x update_x, uint32_t intv)
00618 {
00619     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x]->PRT, 0xdead);
00620     HAL_REG_RESET(HAL_MCPWM[mcpwm_x]->SDCFG, MCPWM_SDCFG_T0_UPDATE_EN_BIT | MCPWM_SDCFG_T1_UPDATE_EN_BIT |
MCPWM_SDCFG_UPDATE_INTV_MASK);
00621     if (update_x != HAL_MCPWM_UPDATE_T1)
00622     {
00623         HAL_REG_SET(HAL_MCPWM[mcpwm_x]->SDCFG, MCPWM_SDCFG_T0_UPDATE_EN_BIT);
00624     }
00625     if (update_x != HAL_MCPWM_UPDATE_T0)
00626     {
00627         HAL_REG_SET(HAL_MCPWM[mcpwm_x]->SDCFG, MCPWM_SDCFG_T1_UPDATE_EN_BIT);
00628     }
00629     HAL_REG_WRITE_BIT_MASK_POS(HAL_MCPWM[mcpwm_x]->SDCFG, MCPWM_SDCFG_UPDATE_INTV, intv);
00630     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x]->PRT, 0);
00631 }

```

4.98 source/hal_lks32mc08x_opa.c 文件参考

```

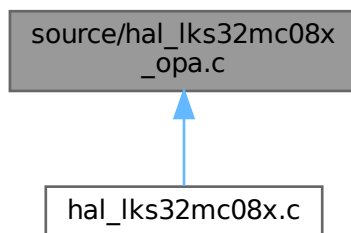
#include "../hal_lks32mc08x.h"
#include "../include/hal_lksmcu_opa.h"
#include "lks32mc08x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc08x_reg.h"

```

hal_lks32mc08x_opa.c 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



函数

- void [HAL_OPA_Reset](#) ([HAL_OPA_x](#) opa_x)
- void [HAL_OPA_Enable](#) ([HAL_OPA_x](#) opa_x)
- void [HAL_OPA_Disable](#) ([HAL_OPA_x](#) opa_x)
- void [HAL_OPA_SetGain](#) ([HAL_OPA_x](#) opa_x, [HAL_OPA_GAIN_x](#) gain_x)
- void [HAL_OPA_Out2GpioEnable](#) ([HAL_OPA_x](#) opa_x)
OPA输出到IO
- void [HAL_OPA_Out2GpioDisable](#) ([HAL_OPA_x](#) opa_x)
OPA停止输出到IO
- void [HAL_OPA_SetGpioIn](#) ([HAL_OPA_x](#) opa_x, [HAL_GPIO_Px](#) px)
OPA设置IO输入
- void [HAL_OPA_SetGpioOut](#) ([HAL_OPA_x](#) opa_x, [HAL_GPIO_Px](#) px)
OPA设置IO输出

4.98.1 函数说明

4.98.1.1 HAL_OPA_Disable()

```
void HAL_OPA_Disable (
    HAL\_OPA\_x opa_x )
```

在文件 [hal_lks32mc08x_opa.c](#) 第 37 行定义.

4.98.1.2 HAL_OPA_Enable()

```
void HAL_OPA_Enable (
    HAL\_OPA\_x opa_x )
```

在文件 [hal_lks32mc08x_opa.c](#) 第 12 行定义.

4.98.1.3 HAL_OPA_Out2GpioDisable()

```
void HAL_OPA_Out2GpioDisable (
    HAL\_OPA\_x opa_x )
```

OPA停止输出到IO

参数

<i>opa</i> _{<i>x</i>}	运放序号

在文件 [hal_lks32mc08x_opa.c](#) 第 119 行定义.

4.98.1.4 HAL_OPA_Out2GpioEnable()

```
void HAL_OPA_Out2GpioEnable (
    HAL_OPA_ x opa_x )
```

OPA输出到IO

参数

<i>opa</i> _{<i>x</i>}	运放序号

在文件 [hal_lks32mc08x_opa.c](#) 第 91 行定义.

4.98.1.5 HAL_OPA_Reset()

```
void HAL_OPA_Reset (
    HAL_OPA_ x opa_x )
```

在文件 [hal_lks32mc08x_opa.c](#) 第 7 行定义.

4.98.1.6 HAL_OPA_SetGain()

```
void HAL_OPA_SetGain (
    HAL_OPA_ x opa_x,
    HAL_OPA_GAIN_ x gain_x )
```

在文件 [hal_lks32mc08x_opa.c](#) 第 62 行定义.

4.98.1.7 HAL_OPA_SetGpioIn()

```
void HAL_OPA_SetGpioIn (
    HAL_OPA_ x opa_x,
    HAL_GPIO_Px px )
```

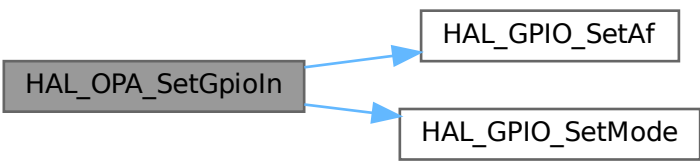
OPA设置IO输入

参数

<i>opa</i> _{<i>x</i>}	运放序号
<i>px</i>	IO序号

在文件 [hal_lks32mc08x_opa.c](#) 第 133 行定义.

函数调用图:



4.98.1.8 HAL_OPA_SetGpioOut()

```
void HAL_OPA_SetGpioOut (
    HAL_OPA_x opa_x,
    HAL_GPIO_Px px )
```

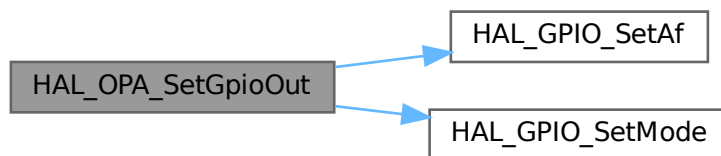
OPA设置IO输出

参数

<i>opa</i> _{<i>x</i>}	运放序号
<i>px</i>	IO序号

在文件 [hal_lks32mc08x_opa.c](#) 第 145 行定义.

函数调用图:



4.99 hal_lks32mc08x_opa.c

[浏览该文件的文档.](#)

```

00001 #include "../hal_lks32mc08x.h"
00002 #include "../include/hal_lksmcu_opa.h"
00003 #include "lks32mc08x_user_manual.h"
00004 #include "../include/hal_lksmcu_reg.h"
00005 #include "../include/hal_lks32mc08x_reg.h"
00006 // 模块复位
00007 void HAL_OPA_Reset(HAL_OPA_x opa_x)
00008 {
00009     (void)opa_x;
00010 }
00011 // 模块打开
00012 void HAL_OPA_Enable(HAL_OPA_x opa_x)
00013 {
00014     // 系统寄存器解锁
00015     HAL_REG_SYS_UNLOCK();
00016     switch (opa_x)
00017     {
00018         case HAL_OPA_0:
00019             HAL_REG_SET(SYS_AFE_REG5, SYS_AFE_REG5_OPA0PDN_BIT);
00020             break;
00021         case HAL_OPA_1:
00022             HAL_REG_SET(SYS_AFE_REG5, SYS_AFE_REG5_OPA1PDN_BIT);
00023             break;
00024         case HAL_OPA_2:
00025             HAL_REG_SET(SYS_AFE_REG5, SYS_AFE_REG5_OPA2PDN_BIT);
00026             break;
00027         case HAL_OPA_3:
00028             HAL_REG_SET(SYS_AFE_REG5, SYS_AFE_REG5_OPA3PDN_BIT);
00029             break;
00030         default:
00031             break;
00032     }
00033     // 系统寄存器上锁
00034     HAL_REG_SYS_LOCK();
00035 }
00036 // 模块关闭
00037 void HAL_OPA_Disable(HAL_OPA_x opa_x)
00038 {
00039     // 系统寄存器解锁
00040     HAL_REG_SYS_UNLOCK();
00041     switch (opa_x)
00042     {
00043         case HAL_OPA_0:
00044             HAL_REG_RESET(SYS_AFE_REG5, SYS_AFE_REG5_OPA0PDN_BIT);
00045             break;
00046         case HAL_OPA_1:
00047             HAL_REG_RESET(SYS_AFE_REG5, SYS_AFE_REG5_OPA1PDN_BIT);
00048             break;
00049         case HAL_OPA_2:
00050             HAL_REG_RESET(SYS_AFE_REG5, SYS_AFE_REG5_OPA2PDN_BIT);
00051             break;
00052         case HAL_OPA_3:
00053             HAL_REG_RESET(SYS_AFE_REG5, SYS_AFE_REG5_OPA3PDN_BIT);
00054             break;
00055         default:
00056             break;
  
```

```

00057     }
00058     // 系统寄存器上锁
00059     HAL_REG_SYS_LOCK();
00060 }
00061 // 设置增益
00062 void HAL_OPA_SetGain(HAL_OPA_x opa_x, HAL_OPA_GAIN_x gain_x)
00063 {
00064     // 系统寄存器解锁
00065     HAL_REG_SYS_UNLOCK();
00066     switch (opa_x)
00067     {
00068         case HAL_OPA_0:
00069             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG0, SYS_AFE_REG0_REF_OPA0, gain_x);
00070             break;
00071         case HAL_OPA_1:
00072             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG0, SYS_AFE_REG0_REF_OPA1, gain_x);
00073             break;
00074         case HAL_OPA_2:
00075             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG0, SYS_AFE_REG0_REF_OPA2, gain_x);
00076             break;
00077         case HAL_OPA_3:
00078             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG0, SYS_AFE_REG0_REF_OPA3, gain_x);
00079             break;
00080         default:
00081             break;
00082     }
00083     // 系统寄存器上锁
00084     HAL_REG_SYS_LOCK();
00085 }
00086
00091 void HAL_OPA_Out2GpioEnable(HAL_OPA_x opa_x)
00092 {
00093     // 使能OPAx输出信号送至IO口p2.7 000:不输出; 001:输出OPA0信号到IO口; 010:输出OPA1信号到IO口; 011:输出OPA2信号
    // 到IO口; 100:输出OPA3信号到IO口; 101~111: 禁止
00094     HAL_REG_SYS_UNLOCK();
00095     switch (opa_x)
00096     {
00097         case HAL_OPA_0:
00098             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG2, SYS_AFE_REG2_OPAOUT_EN, 1);
00099             break;
00100         case HAL_OPA_1:
00101             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG2, SYS_AFE_REG2_OPAOUT_EN, 2);
00102             break;
00103         case HAL_OPA_2:
00104             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG2, SYS_AFE_REG2_OPAOUT_EN, 3);
00105             break;
00106         case HAL_OPA_3:
00107             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG2, SYS_AFE_REG2_OPAOUT_EN, 4);
00108             break;
00109         default:
00110             break;
00111     }
00112     HAL_REG_SYS_LOCK();
00113 }
00114
00119 void HAL_OPA_Out2GpioDisable(HAL_OPA_x opa_x)
00120 {
00121     (void) opa_x;
00122     // 使能OPAx输出信号送至IO口p2.7 000:不输出; 001:输出OPA0信号到IO口; 010:输出OPA1信号到IO口; 011:输出OPA2信号
    // 到IO口; 100:输出OPA3信号到IO口; 101~111: 禁止
00123     HAL_REG_SYS_UNLOCK();
00124     HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG2, SYS_AFE_REG2_OPAOUT_EN, 0);
00125     HAL_REG_SYS_LOCK();
00126 }
00127
00133 void HAL_OPA_SetGpioIn(HAL_OPA_x opa_x, HAL_GPIO_Px px)
00134 {
00135     (void) opa_x;
00136     HAL_GPIO_SetAf(px, HAL_GPIO_AF_OPA);
00137     HAL_GPIO_SetMode(px, HAL_GPIO_MODE_AFE);
00138 }
00139
00145 void HAL_OPA_SetGpioOut(HAL_OPA_x opa_x, HAL_GPIO_Px px)
00146 {
00147     (void) opa_x;
00148     HAL_GPIO_SetAf(px, HAL_GPIO_AF_OPA);
00149     HAL_GPIO_SetMode(px, HAL_GPIO_MODE_AFE);
00150 }

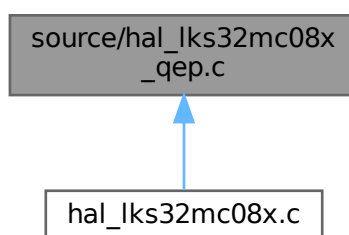
```


4.100 source/hal_lks32mc08x_qep.c 文件参考

```
#include "../include/hal_lksmcu_clk.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lksmcu_qep.h"
#include "../include/hal_lksmcu_timer.h"
#include "../hal_lks32mc08x.h"
#include "lks32mc08x_user_manual.h"
#include "../include/hal_lks32mc08x_reg.h"
hal_lks32mc08x_qep.c 的引用(Include)关系图:
```



此图展示该文件直接或间接的被哪些文件引用了:



函数

- void [HAL_QEP_Reset](#) (HAL_QEP_x qep)
模块复位
- void [HAL_QEP_Enable](#) (HAL_QEP_x qep)
模块打开
- void [HAL_QEP_Disable](#) (HAL_QEP_x qep)
模块关闭
- void [HAL_QEP_SetTh](#) (HAL_QEP_x qep, uint32_t th)
设置最大计数值
- void [HAL_QEP_SetMode](#) (HAL_QEP_x qep, HAL_QEP_MODE_x mode)
设置计数模式
- uint32_t [HAL_QEP_ReadCnt](#) (HAL_QEP_x qep)
获取当前的计数值
- void [HAL_QEP_ResetCnt](#) (HAL_QEP_x qep)
复位计数值到0
- void [HAL_QEP_SetClockDiv](#) (HAL_QEP_x qep_x, uint32_t div)
- uint32_t [HAL_QEP_GetClockDiv](#) (HAL_QEP_x qep_x)
- uint32_t [HAL_QEP_GetClockFreq](#) (HAL_QEP_x qep_x)
- void [HAL_QEP_SetGpio](#) (HAL_QEP_x qep_x, HAL_GPIO_Px px)
设置QEP模块的引脚

4.100.1 函数说明

4.100.1.1 HAL_QEP_Disable()

```
void HAL_QEP_Disable (
    HAL_QEP_x qep )
```

模块关闭

参数

qep	QEP模块

在文件 [hal_lks32mc08x.qep.c](#) 第 46 行定义.

4.100.1.2 HAL_QEP_Enable()

```
void HAL_QEP_Enable (
    HAL_QEP_x qep )
```

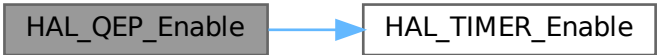
模块打开

参数

qep	QEP模块

在文件 [hal_lks32mc08x.qep.c](#) 第 25 行定义.

函数调用图:



4.100.1.3 HAL_QEP_GetClockDiv()

```
uint32_t HAL_QEP_GetClockDiv (
    HAL_QEP_x qep_x )
```

在文件 [hal_lks32mc08x.qep.c](#) 第 110 行定义.

函数调用图:



这是这个函数的调用关系图:

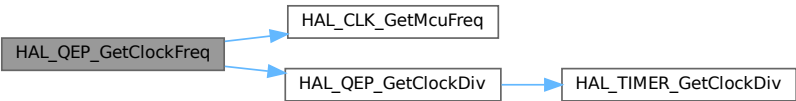


4.100.1.4 HAL_QEP_GetClockFreq()

```
uint32_t HAL_QEP_GetClockFreq (
    HAL_QEP_x qep_x )
```

在文件 [hal_lks32mc08x.qep.c](#) 第 116 行定义.

函数调用图:



4.100.1.5 HAL_QEP_ReadCnt()

```
uint32_t HAL_QEP_ReadCnt (
    HAL_QEP_x qep )
```

获取当前的计数值

参数

<i>qep</i>	QEP模块

返回

当前的计数值

在文件 [hal_lks32mc08x.qep.c](#) 第 89 行定义.

4.100.1.6 HAL_QEP_Reset()

```
void HAL_QEP_Reset (
    HAL_QEP_x qep )
```

模块复位

参数

<i>qep</i>	QEP模块

在文件 [hal_lks32mc08x.qep.c](#) 第 14 行定义.

4.100.1.7 HAL_QEP_ResetCnt()

```
void HAL_QEP_ResetCnt (
    HAL_QEP_x qep )
```

复位计数值到0

参数

<i>qep</i>	QEP模块

在文件 [hal_lks32mc08x.qep.c](#) 第 98 行定义.

4.100.1.8 HAL_QEP_SetClockDiv()

```
void HAL_QEP_SetClockDiv (
    HAL_QEP_x qep-x,
    uint32_t div )
```

在文件 [hal_lks32mc08x.qep.c](#) 第 104 行定义.

函数调用图:



4.100.1.9 HAL_QEP_SetGpio()

```
void HAL_QEP_SetGpio (
    HAL_QEP_x qepx,
    HAL_GPIO_Px px )
```

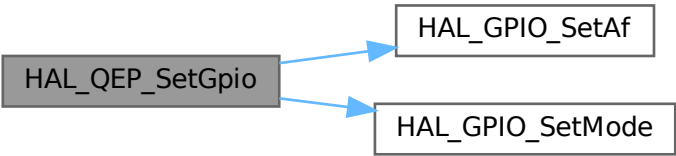
设置QEP模块的引脚

参数

<i>qepx</i>	QEP模块
<i>px</i>	引脚

在文件 [hal_lks32mc08x_qep.c](#) 第 126 行定义.

函数调用图:



4.100.1.10 HAL_QEP_SetMode()

```
void HAL_QEP_SetMode (
    HAL_QEP_x qep,
    HAL_QEP_MODE_x mode )
```

设置计数模式

参数

<i>qep</i>	QEP模块
<i>mode</i>	计数模式

在文件 [hal_lks32mc08x_qep.c](#) 第 66 行定义.

4.100.1.11 HAL_QEP_SetTh()

```
void HAL_QEP_SetTh (
    HAL_QEP_x qep,
    uint32_t th )
```

设置最大计数值

参数

<i>qep</i>	QEP模块
<i>th</i>	最大计数值

在文件 [hal_lks32mc08x_qep.c](#) 第 56 行定义.

4.101 hal_lks32mc08x_qep.c

[浏览该文件的文档.](#)

```

00001 #include "../include/hal_lksmcu_clk.h"
00002 #include "../include/hal_lksmcu_reg.h"
00003 #include "../include/hal_lksmcu_qep.h"
00004 #include "../include/hal_lksmcu_timer.h"
00005 #include "../hal_lks32mc08x.h"
00006 #include "lks32mc08x_user_manual.h"
00007 #include "../include/hal_lksmcu_reg.h"
00008 #include "../include/hal_lks32mc08x_reg.h"
00009
00014 void HAL_QEP_Reset (HAL_QEP_x qep)
00015 {
00016     HAL_REG_WRITE (HAL_QEP[qep]->CFG, 0);
00017     HAL_REG_WRITE (HAL_QEP[qep]->TH, 0);
00018     HAL_REG_WRITE (HAL_QEP[qep]->CNT, 0);
00019 }
00020
00025 void HAL_QEP_Enable (HAL_QEP_x qep)
00026 {
00027     switch (qep)
00028     {
00029         case HAL_QEP_0:
00030             HAL_TIMER_Enable (HAL_TIMER_2); // qep0使用时必须使能TIMER2
00031             HAL_REG_SET (UTIMER_CFG, UTIMER_CFG_ENC0_EN_BIT);
00032             break;
00033         case HAL_QEP_1:
00034             HAL_TIMER_Enable (HAL_TIMER_3); // qep1使用时必须使能TIMER3
00035             HAL_REG_SET (UTIMER_CFG, UTIMER_CFG_ENC1_EN_BIT);
00036             break;
00037         default:
00038             break;
00039     }
00040 }
00041
00046 void HAL_QEP_Disable (HAL_QEP_x qep)
00047 {
00048     (void)qep;
00049 }
00050
00056 void HAL_QEP_SetTh (HAL_QEP_x qep, uint32_t th)
00057 {
00058     HAL_REG_WRITE (HAL_QEP[qep]->TH, th);
00059 }
00060
00066 void HAL_QEP_SetMode (HAL_QEP_x qep, HAL_QEP_MODE_x mode)
00067 {
00068     switch (mode)
00069     {
00070         case HAL_QEP_MODE_QUADRATURE:
00071             HAL_REG_WRITE_BIT_MASK_POS (HAL_QEP[qep]->CFG, UTIMER_ECDx_CFG_MODE, 1);
00072             break;
00073         case HAL_QEP_MODE_SYMBOL_PULSE:
00074             HAL_REG_WRITE_BIT_MASK_POS (HAL_QEP[qep]->CFG, UTIMER_ECDx_CFG_MODE, 2);
00075             break;
00076         case HAL_QEP_MODE_DUAL_PULSE:
00077             HAL_REG_WRITE_BIT_MASK_POS (HAL_QEP[qep]->CFG, UTIMER_ECDx_CFG_MODE, 3);
00078             break;
00079         default:
00080             break;
00081     }
00082 }
00083
00089 uint32_t HAL_QEP_ReadCnt (HAL_QEP_x qep)
00090 {

```

```

00091     return HAL_REG_READ(HAL_QEP[qep]->CNT);
00092 }
00093
00098 void HAL_QEP_ResetCnt(HAL_QEP_x qep)
00099 {
00100     HAL_REG_WRITE(HAL_QEP[qep]->CNT, 0);
00101 }
00102
00103 // 设置时钟分频 (QEP复用TIMER2/3的CLK_DIV)
00104 void HAL_QEP_SetClockDiv(HAL_QEP_x qep_x, uint32_t div)
00105 {
00106     HAL_TIMER_SetClockDiv((qep_x == HAL_QEP_0) ? HAL_TIMER_2 : HAL_TIMER_3, div);
00107 }
00108
00109 // 获取时钟分频实际除数
00110 uint32_t HAL_QEP_GetClockDiv(HAL_QEP_x qep_x)
00111 {
00112     return HAL_TIMER_GetClockDiv((qep_x == HAL_QEP_0) ? HAL_TIMER_2 : HAL_TIMER_3);
00113 }
00114
00115 // 获取分频后工作时钟
00116 uint32_t HAL_QEP_GetClockFreq(HAL_QEP_x qep_x)
00117 {
00118     return HAL_CLK_GetMcuFreq() / HAL_QEP_GetClockDiv(qep_x);
00119 }
00120
00126 void HAL_QEP_SetGpio(HAL_QEP_x qep_x, HAL_GPIO_Px px)
00127 {
00128     (void)qep_x;
00129     HAL_GPIO_SetMode(px, HAL_GPIO_MODE_IN);
00130     HAL_GPIO_SetAf(px, HAL_GPIO_AF_QEP0);
00131 }

```

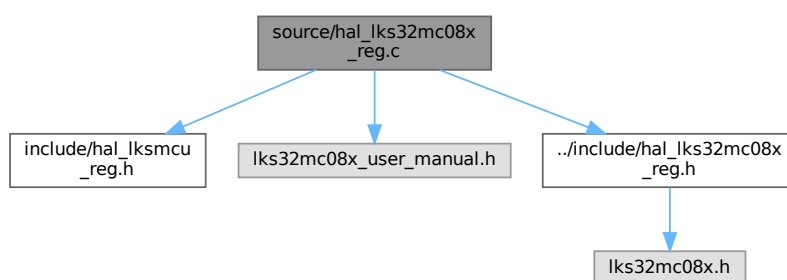
4.102 source/hal_lks32mc08x_reg.c 文件参考

```

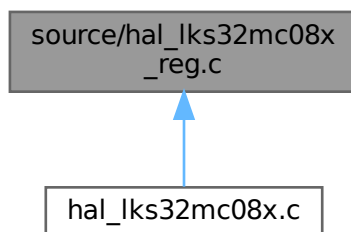
#include "include/hal_lksmcu_reg.h"
#include "lks32mc08x_user_manual.h"
#include "../include/hal_lks32mc08x_reg.h"

```

hal_lks32mc08x_reg.c 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



变量

- `HAL_SYS_TypeDef *const HAL_SYS [] = {(HAL_SYS_TypeDef *)MISC_BASE}`
- `HAL_FLASH_TypeDef *const HAL_FLASH [] = {(HAL_FLASH_TypeDef *)FLS_CR_BASE}`
- `HAL_SPI_TypeDef *const HAL_SPI [] = {(HAL_SPI_TypeDef *)SPI0_BASE}`
- `HAL_I2C_TypeDef *const HAL_I2C [] = {(HAL_I2C_TypeDef *)I2C0_BASE}`
- `HAL_CMP_TypeDef *const HAL_CMP [] = {(HAL_CMP_TypeDef *)CMP_BASE}`
- `HAL_HALL_TypeDef *const HAL_HALL [] = {(HAL_HALL_TypeDef *)HALL_BASE}`
- `HAL_ADC_TypeDef *const HAL_ADC [] = {(HAL_ADC_TypeDef *)ADC0_BASE}`
- `HAL_TIMER_TypeDef *const HAL_TIMER [] = {(HAL_TIMER_TypeDef *)UTIMER_BASE, (HAL_TIMER_TypeDef *) (UTIMER_BASE + 0x20), (HAL_TIMER_TypeDef *) (UTIMER_BASE + 0x40), (HAL_TIMER_TypeDef *) (UTIMER_BASE + 0x60)}`
- `HAL_QEP_TypeDef *const HAL_QEP [] = {(HAL_QEP_TypeDef *) (UTIMER_BASE + 0x80), (HAL_QEP_TypeDef *) (UTIMER_BASE + 0x90)}`
- `HAL_MCPWM_TypeDef *const HAL_MCPWM [] = {(HAL_MCPWM_TypeDef *)MCPWM0_BASE}`
- `HAL_GPIO_TypeDef *const HAL_GPIO [] = {(HAL_GPIO_TypeDef *)GPIO0_BASE, (HAL_GPIO_TypeDef *)GPIO1_BASE, (HAL_GPIO_TypeDef *)GPIO2_BASE, (HAL_GPIO_TypeDef *)GPIO3_BASE}`
- `HAL_CRC_TypeDef *const HAL_CRC [] = {(HAL_CRC_TypeDef *)CRC_BASE}`
- `HAL_UART_TypeDef *const HAL_UART [] = {(HAL_UART_TypeDef *)UART0_BASE, (HAL_UART_TypeDef *)UART1_BASE}`
- `HAL_CAN_TypeDef *const HAL_CAN [] = {(HAL_CAN_TypeDef *)CAN_BASE}`
- `HAL_DSP_TypeDef *const HAL_DSP [] = {(HAL_DSP_TypeDef *)DSP_BASE}`
- `HAL_IWDG_TypeDef *const HAL_IWDG [] = {(HAL_IWDG_TypeDef *) (MISC_BASE + 0x38)}`
- `HAL_DMA_TypeDef *const HAL_DMA [] = {(HAL_DMA_TypeDef *)DMA_BASE, (HAL_DMA_TypeDef *) (DMA_BASE + 0x10), (HAL_DMA_TypeDef *) (DMA_BASE + 0x20), (HAL_DMA_TypeDef *) (DMA_BASE + 0x30)}`

4.102.1 变量说明

4.102.1.1 HAL_ADC

`HAL_ADC_TypeDef* const HAL_ADC[] = {(HAL_ADC_TypeDef *)ADC0_BASE}`

在文件 `hal_lks32mc08x_reg.c` 第 14 行定义.

4.102.1.2 HAL_CAN

```
HAL_CAN_TypeDef* const HAL_CAN[] = {(HAL_CAN_TypeDef *)CAN_BASE}
```

在文件 [hal_lks32mc08x_reg.c](#) 第 21 行定义.

4.102.1.3 HAL_CMP

```
HAL_CMP_TypeDef* const HAL_CMP[] = {(HAL_CMP_TypeDef *)CMP_BASE}
```

在文件 [hal_lks32mc08x_reg.c](#) 第 12 行定义.

4.102.1.4 HAL_CRC

```
HAL_CRC_TypeDef* const HAL_CRC[] = {(HAL_CRC_TypeDef *)CRC_BASE}
```

在文件 [hal_lks32mc08x_reg.c](#) 第 19 行定义.

4.102.1.5 HAL_DMA

```
HAL_DMA_TypeDef* const HAL_DMA[] = {(HAL_DMA_TypeDef *)DMA_BASE, (HAL_DMA_TypeDef *) (DMA_BASE +  
0x10), (HAL_DMA_TypeDef *) (DMA_BASE + 0x20), (HAL_DMA_TypeDef *) (DMA_BASE + 0x30)}
```

在文件 [hal_lks32mc08x_reg.c](#) 第 24 行定义.

4.102.1.6 HAL_DSP

```
HAL_DSP_TypeDef* const HAL_DSP[] = {(HAL_DSP_TypeDef *)DSP_BASE}
```

在文件 [hal_lks32mc08x_reg.c](#) 第 22 行定义.

4.102.1.7 HAL_FLASH

```
HAL_FLASH_TypeDef* const HAL_FLASH[] = {(HAL_FLASH_TypeDef *)FLS_CR_BASE}
```

在文件 [hal_lks32mc08x_reg.c](#) 第 9 行定义.

4.102.1.8 HAL_GPIO

```
HAL_GPIO_TypeDef* const HAL_GPIO[] = {(HAL_GPIO_TypeDef *)GPIO0_BASE, (HAL_GPIO_TypeDef *)GPIO1_↔  
BASE, (HAL_GPIO_TypeDef *)GPIO2_BASE, (HAL_GPIO_TypeDef *)GPIO3_BASE}
```

在文件 [hal_lks32mc08x_reg.c](#) 第 18 行定义.

4.102.1.9 HAL_HALL

```
HAL_HALL_TypeDef* const HAL_HALL[] = {(HAL_HALL_TypeDef *)HALL_BASE}
```

在文件 [hal_lks32mc08x.reg.c](#) 第 13 行定义.

4.102.1.10 HAL_I2C

```
HAL_I2C_TypeDef* const HAL_I2C[] = {(HAL_I2C_TypeDef *)I2C0_BASE}
```

在文件 [hal_lks32mc08x.reg.c](#) 第 11 行定义.

4.102.1.11 HAL_IWDG

```
HAL_IWDG_TypeDef* const HAL_IWDG[] = {(HAL_IWDG_TypeDef *) (MISC_BASE + 0x38)}
```

在文件 [hal_lks32mc08x.reg.c](#) 第 23 行定义.

4.102.1.12 HAL_MCPWM

```
HAL_MCPWM_TypeDef* const HAL_MCPWM[] = {(HAL_MCPWM_TypeDef *)MCPWM0_BASE}
```

在文件 [hal_lks32mc08x.reg.c](#) 第 17 行定义.

4.102.1.13 HAL_QEP

```
HAL_QEP_TypeDef* const HAL_QEP[] = {(HAL_QEP_TypeDef *) (UTIMER_BASE + 0x80), (HAL_QEP_TypeDef *) (UTIMER_BASE + 0x90)}
```

在文件 [hal_lks32mc08x.reg.c](#) 第 16 行定义.

4.102.1.14 HAL_SPI

```
HAL_SPI_TypeDef* const HAL_SPI[] = {(HAL_SPI_TypeDef *)SPI0_BASE}
```

在文件 [hal_lks32mc08x.reg.c](#) 第 10 行定义.

4.102.1.15 HAL_SYS

```
HAL_SYS_TypeDef* const HAL_SYS[] = {(HAL_SYS_TypeDef *)MISC_BASE}
```

在文件 [hal_lks32mc08x.reg.c](#) 第 8 行定义.

4.102.1.16 HAL_TIMER

```
HAL_TIMER_TypeDef* const HAL_TIMER[] = {(HAL_TIMER_TypeDef *)UTIMER_BASE, (HAL_TIMER_TypeDef *)
*(UTIMER_BASE + 0x20), (HAL_TIMER_TypeDef *) (UTIMER_BASE + 0x40), (HAL_TIMER_TypeDef *) (UTIMER_
BASE + 0x60)}
```

在文件 `hal_lks32mc08x_reg.c` 第 15 行定义。

4.102.1.17 HAL_UART

```
HAL_UART_TypeDef* const HAL_UART[] = {(HAL_UART_TypeDef *)UART0_BASE, (HAL_UART_TypeDef *)UART1_
BASE}
```

在文件 `hal_lks32mc08x_reg.c` 第 20 行定义。

4.103 hal_lks32mc08x_reg.c

浏览该文件的文档。

```
00001
00002 #include "include/hal_lksmcu_reg.h"
00003 #include "lks32mc08x-user-manual.h"
00004 #include "../include/hal_lksmcu_reg.h"
00005 #include "../include/hal_lks32mc08x_reg.h"
00006
00007 #ifdef HAL_LKS32MC08X_REG
00008 HAL_SYS_TypeDef *const HAL_SYS[] = {(HAL_SYS_TypeDef *)MISC_BASE};
00009 HAL_FLASH_TypeDef *const HAL_FLASH[] = {(HAL_FLASH_TypeDef *)FLS_CR_BASE};
00010 HAL_SPI_TypeDef *const HAL_SPI[] = {(HAL_SPI_TypeDef *)SPI0_BASE};
00011 HAL_I2C_TypeDef *const HAL_I2C[] = {(HAL_I2C_TypeDef *)I2C0_BASE};
00012 HAL_CMP_TypeDef *const HAL_CMP[] = {(HAL_CMP_TypeDef *)CMP_BASE};
00013 HAL_HALL_TypeDef *const HAL_HALL[] = {(HAL_HALL_TypeDef *)HALL_BASE};
00014 HAL_ADC_TypeDef *const HAL_ADC[] = {(HAL_ADC_TypeDef *)ADC0_BASE};
00015 HAL_TIMER_TypeDef *const HAL_TIMER[] = {(HAL_TIMER_TypeDef *)UTIMER_BASE, (HAL_TIMER_TypeDef
*) (UTIMER_BASE + 0x20), (HAL_TIMER_TypeDef *) (UTIMER_BASE + 0x40), (HAL_TIMER_TypeDef *) (UTIMER_BASE +
0x60)};
00016 HAL_QEP_TypeDef *const HAL_QEP[] = {(HAL_QEP_TypeDef *) (UTIMER_BASE + 0x80), (HAL_QEP_TypeDef
*) (UTIMER_BASE + 0x90)};
00017 HAL_MCPWM_TypeDef *const HAL_MCPWM[] = {(HAL_MCPWM_TypeDef *)MCPWM0_BASE};
00018 HAL_GPIO_TypeDef *const HAL_GPIO[] = {(HAL_GPIO_TypeDef *)GPIO0_BASE, (HAL_GPIO_TypeDef *)GPIO1_BASE,
(HAL_GPIO_TypeDef *)GPIO2_BASE, (HAL_GPIO_TypeDef *)GPIO3_BASE};
00019 HAL_CRC_TypeDef *const HAL_CRC[] = {(HAL_CRC_TypeDef *)CRC_BASE};
00020 HAL_UART_TypeDef *const HAL_UART[] = {(HAL_UART_TypeDef *)UART0_BASE, (HAL_UART_TypeDef *)UART1_BASE};
00021 HAL_CAN_TypeDef *const HAL_CAN[] = {(HAL_CAN_TypeDef *)CAN_BASE};
00022 HAL_DSP_TypeDef *const HAL_DSP[] = {(HAL_DSP_TypeDef *)DSP_BASE};
00023 HAL_IWDG_TypeDef *const HAL_IWDG[] = {(HAL_IWDG_TypeDef *) (MISC_BASE + 0x38)};
00024 HAL_DMA_TypeDef *const HAL_DMA[] = {(HAL_DMA_TypeDef *)DMA_BASE, (HAL_DMA_TypeDef *) (DMA_BASE + 0x10),
(HAL_DMA_TypeDef *) (DMA_BASE + 0x20), (HAL_DMA_TypeDef *) (DMA_BASE + 0x30)};
00025 #endif
```

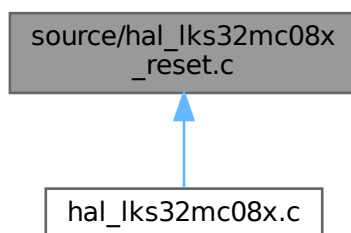
4.104 source/hal_lks32mc08x_reset.c 文件参考

```
#include "../include/hal_lksmcu_reset.h"
#include "../hal_lks32mc08x.h"
#include "lks32mc08x-user-manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc08x_reg.h"
```

hal_lks32mc08x_reset.c 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



函数

- void **HAL_RESET_ResetCore** ()
复位内核
- void **HAL_RESET_ResetRegister** ()
寄存器复位
- void **HAL_RESET_ResetGlobal** ()
全局复位(先复位寄存器再复位内核)

4.104.1 函数说明

4.104.1.1 HAL_RESET_ResetCore()

```
void HAL_RESET_ResetCore (  
    void )
```

复位内核

在文件 [hal_lks32mc08x.reset.c](#) 第 10 行定义.

4.104.1.2 HAL_RESET_ResetGlobal()

```
void HAL_RESET_ResetGlobal (  
    void )
```

全局复位(先复位寄存器再复位内核)

在文件 [hal_lks32mc08x.reset.c](#) 第 35 行定义.

函数调用图:



4.104.1.3 HAL_RESET_ResetRegister()

```
void HAL_RESET_ResetRegister (
    void )
```

寄存器复位

在文件 `hal_lks32mc08x.reset.c` 第 18 行定义。

这是这个函数的调用关系图:



4.105 hal_lks32mc08x.reset.c

[浏览该文件的文档.](#)

```

00001
00002 #include "../include/hal_lksmcu.reset.h"
00003 #include "../hal_lks32mc08x.h"
00004 #include "lks32mc08x.user_manual.h"
00005 #include "../include/hal_lksmcu.reg.h"
00006 #include "../include/hal_lks32mc08x.reg.h"
00010 void HAL_RESET_ResetCore()
00011 {
00012     NVIC.SystemReset();
00013 }
00014
00018 void HAL_RESET_ResetRegister()
00019 {
00020     // 系统寄存器解锁
00021     HAL_REG_SYS_UNLOCK();
00022     HAL_REG_WRITE(HAL_GPIO[0]->POE, 0);
00023     HAL_REG_WRITE(HAL_GPIO[1]->POE, 0);
00024     HAL_REG_WRITE(HAL_GPIO[2]->POE, 0);
00025     HAL_REG_WRITE(HAL_GPIO[3]->POE, 0);
00026     HAL_REG_SET(SYS_SFT_RST, 0xffff);
00027     HAL_REG_RESET(SYS_SFT_RST, 0);
00028     // 系统寄存器上锁
00029     HAL_REG_SYS_LOCK();
00030 }
00031
00035 void HAL_RESET_ResetGlobal()
00036 {
00037     HAL_RESET_ResetRegister();
00038     NVIC.SystemReset();
00039 }
```

4.106 source/hal_lks32mc08x_spi.c 文件参考

```

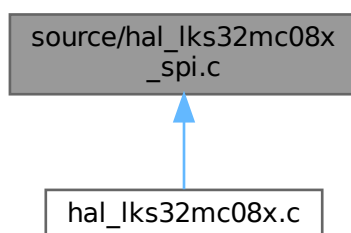
#include "../include/hal_lksmcu.spi.h"
#include "../hal_lks32mc08x.h"
#include "lks32mc08x.user_manual.h"
#include "../include/hal_lksmcu.reg.h"
```

```
#include "../include/hal_lks32mc08x_reg.h"
```

hal_lks32mc08x_spi.c 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



函数

- void **HAL_SPI_Reset** (HAL_SPI_x spi)
模块复位
- void **HAL_SPI_Enable** (HAL_SPI_x spi)
模块打开
- void **HAL_SPI_Disable** (HAL_SPI_x spi)
模块关闭
- void **HAL_SPI_SetBaud** (HAL_SPI_x spi, uint32_t baud)
设置波特率
- void **HAL_SPI_SetClockDiv** (HAL_SPI_x spi, uint32_t div)
- uint32_t **HAL_SPI_GetClockDiv** (HAL_SPI_x spi)
- uint32_t **HAL_SPI_GetClockFreq** (HAL_SPI_x spi)
- uint32_t **HAL_SPI_GetBaud** (HAL_SPI_x spi)
- void **HAL_SPI_SetMode** (HAL_SPI_x spi, HAL_SPI_MODE_x mode)
设置SPI模式
- void **HAL_SPI_SetGpioCs** (HAL_SPI_x spi, HAL_GPIO_Px pin)
初始化CS引脚
- void **HAL_SPI_SetGpioMosi** (HAL_SPI_x spi, HAL_GPIO_Px pin)
初始化Mosi引脚
- void **HAL_SPI_SetGpioMiso** (HAL_SPI_x spi, HAL_GPIO_Px pin)
初始化Miso引脚
- void **HAL_SPI_SetGpioClk** (HAL_SPI_x spi, HAL_GPIO_Px pin)
初始化Clk引脚
- void **HAL_SPI_SetGpioDi** (HAL_SPI_x spi, HAL_GPIO_Px pin)

- 初始化SPI的输入引脚
- void [HAL_SPI_SetGpioDo](#) ([HAL_SPI_x](#) spi, [HAL_GPIO_Px](#) pin)
- 初始化SPI的输出引脚
- uint8_t [HAL_SPI_TxData](#) ([HAL_SPI_x](#) spi, uint8_t *data, uint32_t len)
- 发送数据
- uint8_t [HAL_SPI_RxData](#) ([HAL_SPI_x](#) spi, uint8_t *data, uint32_t len)
- 接收数据
- uint8_t [HAL_SPI_RxTxData](#) ([HAL_SPI_x](#) spi, uint8_t *rxdata, uint8_t *txdata, uint32_t len)
- 发送和接收数据
- uint8_t [HAL_SPI_TxDataWidth16](#) ([HAL_SPI_x](#) spi, uint16_t *data, uint32_t len)
- 发送数据
- uint8_t [HAL_SPI_RxDataWidth16](#) ([HAL_SPI_x](#) spi, uint16_t *data, uint32_t len)
- 接收数据
- uint8_t [HAL_SPI_RxTxDataWidth16](#) ([HAL_SPI_x](#) spi, uint16_t *rxdata, uint16_t *txdata, uint32_t len)
- 发送和接收数据

4.106.1 函数说明

4.106.1.1 HAL_SPI.Disable()

```
void HAL_SPI_Disable (
    HAL\_SPI\_x spi )
```

模块关闭

参数

<i>spi</i>	SPI模块号

在文件 [hal_lks32mc08x_spi.c](#) 第 37 行定义.

4.106.1.2 HAL_SPI.Enable()

```
void HAL_SPI_Enable (
    HAL\_SPI\_x spi )
```

模块打开

参数

<i>spi</i>	SPI模块号

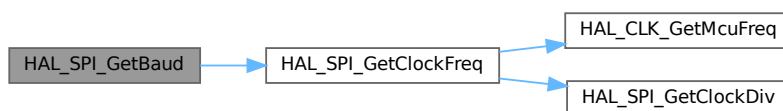
在文件 [hal_lks32mc08x_spi.c](#) 第 25 行定义.

4.106.1.3 HAL_SPI.GetBaud()

```
uint32_t HAL_SPI_GetBaud (
    HAL\_SPI\_x spi )
```

在文件 [hal_lks32mc08x_spi.c](#) 第 76 行定义.

函数调用图:



4.106.1.4 HAL_SPI.GetClockDiv()

```
uint32_t HAL_SPI_GetClockDiv (  
    HAL_SPI_TypeDef* spi )
```

在文件 [hal_lks32mc08x_spi.c](#) 第 64 行定义.

这是这个函数的调用关系图:

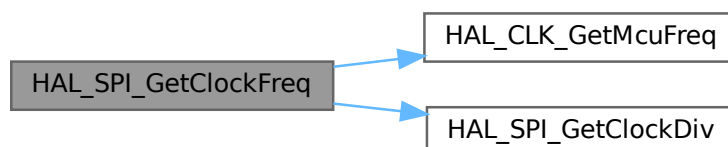


4.106.1.5 HAL_SPI.GetClockFreq()

```
uint32_t HAL_SPI_GetClockFreq (  
    HAL_SPI_TypeDef* spi )
```

在文件 [hal_lks32mc08x_spi.c](#) 第 70 行定义.

函数调用图:



这是这个函数的调用关系图:



4.106.1.6 HAL_SPI_Reset()

```
void HAL_SPI_Reset (
    HAL_SPI_x spi )
```

模块复位

参数

<i>spi</i>	SPI模块号

在文件 [hal_lks32mc08x_spi.c](#) 第 12 行定义.

4.106.1.7 HAL_SPI_RxData()

```
uint8_t HAL_SPI_RxData (
    HAL_SPI_x spi,
    uint8_t * data,
    uint32_t len )
```

接收数据

参数

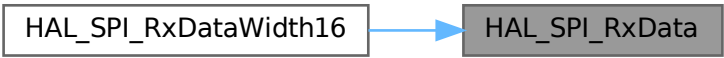
<i>spi</i>	SPI模块号
<i>data</i>	接收缓冲区
<i>len</i>	数据长度

返回

接收状态

在文件 [hal_lks32mc08x_spi.c](#) 第 287 行定义.

这是这个函数的调用关系图:



4.106.1.8 HAL_SPI_RxDataWidth16()

```
uint8_t HAL_SPI_RxDataWidth16 (
    HAL_SPI_TypeDef * spi,
    uint16_t * data,
    uint32_t len )
```

接收数据

参数

<i>spi</i>	SPI模块号
<i>data</i>	接收缓冲区
<i>len</i>	数据长度，单位2字节

返回

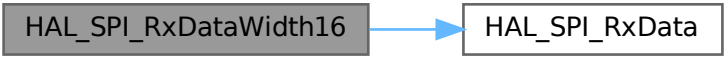
接收状态

注解

lks32mc08x不支持16位模式，这里用8位模式发送

在文件 [hal_lks32mc08x_spi.c](#) 第 354 行定义.

函数调用图:



4.106.1.9 HAL_SPI_RxTxData()

```
uint8_t HAL_SPI_RxTxData (
    HAL_SPI_x spi,
    uint8_t * rxdata,
    uint8_t * txdata,
    uint32_t len )
```

发送和接收数据

参数

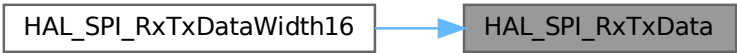
<i>spi</i>	SPI模块号
<i>rxdata</i>	接收缓冲区
<i>txdata</i>	要发送的数据
<i>len</i>	数据长度

返回

发送和接收状态

在文件 [hal_lks32mc08x_spi.c](#) 第 314 行定义.

这是这个函数的调用关系图:



4.106.1.10 HAL_SPI_RxTxDataWidth16()

```
uint8_t HAL_SPI_RxTxDataWidth16 (
    HAL_SPI_x spi,
    uint16_t * rxdata,
    uint16_t * txdata,
    uint32_t len )
```

发送和接收数据

参数

<i>spi</i>	SPI模块号
<i>rxdata</i>	接收缓冲区
<i>txdata</i>	要发送的数据
<i>len</i>	数据长度，单位2字节

返回

发送和接收状态

注解

lks32mc08x不支持16位模式，这里用8位模式发送

在文件 [hal_lks32mc08x_spi.c](#) 第 369 行定义.

函数调用图:



4.106.1.11 HAL_SPI_SetBaud()

```
void HAL_SPI_SetBaud (
    HAL_SPI_TypeDef* spi,
    uint32_t baud )
```

设置波特率

参数

<i>spi</i>	SPI模块号

在文件 [hal_lks32mc08x_spi.c](#) 第 49 行定义.

函数调用图:



4.106.1.12 HAL_SPI_SetClockDiv()

```
void HAL_SPI_SetClockDiv (
```

```
HAL_SPI_x spi,  
uint32_t div )
```

在文件 [hal_lks32mc08x_spi.c](#) 第 57 行定义.

4.106.1.13 HAL_SPI_SetGpioClk()

```
void HAL_SPI_SetGpioClk (  
    HAL_SPI_x spi,  
    HAL_GPIO_Px pin )
```

初始化Clk引脚

参数

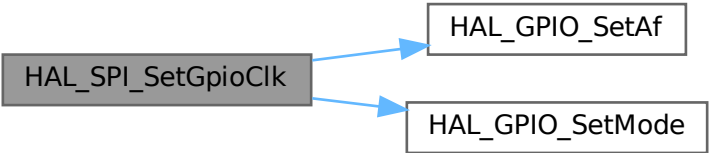
<i>spi</i>	SPI模块号
<i>pin</i>	GPIO引脚

参见

需要先初始化SPI

在文件 [hal_lks32mc08x_spi.c](#) 第 216 行定义.

函数调用图:



4.106.1.14 HAL_SPI_SetGpioCs()

```
void HAL_SPI_SetGpioCs (  
    HAL_SPI_x spi,  
    HAL_GPIO_Px pin )
```

初始化CS引脚

参数

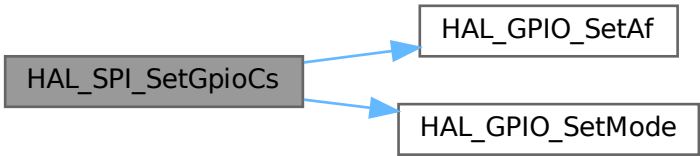
<i>spi</i>	SPI模块号
<i>pin</i>	GPIO引脚

参见

需要先初始化SPI

在文件 [hal_lks32mc08x_spi.c](#) 第 150 行定义.

函数调用图:



4.106.1.15 HAL_SPI_SetGpioDi()

```
void HAL_SPI_SetGpioDi (
    HAL_SPI_TypeDef* spi,
    HAL_GPIO_TypeDef* pin )
```

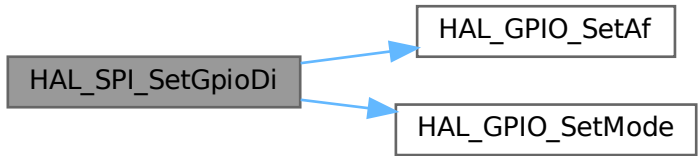
初始化SPI的输入引脚

参数

<i>spi</i>	SPI模块号
<i>pin</i>	GPIO引脚

在文件 [hal_lks32mc08x_spi.c](#) 第 237 行定义.

函数调用图:



4.106.1.16 HAL_SPI_SetGpioDo()

```
void HAL_SPI_SetGpioDo (
    HAL_SPI_TypeDef* spi,
    GPIO_TypeDef* pin )
```

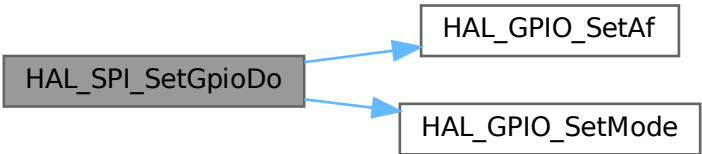
初始化SPI的输出引脚

参数

spi	SPI模块号
pin	GPIO引脚

在文件 [hal_lks32mc08x_spi.c](#) 第 249 行定义.

函数调用图:



4.106.1.17 HAL_SPI_SetGpioMiso()

```
void HAL_SPI_SetGpioMiso (
    HAL_SPI_TypeDef* spi,
    GPIO_TypeDef* pin )
```

初始化Miso引脚

参数

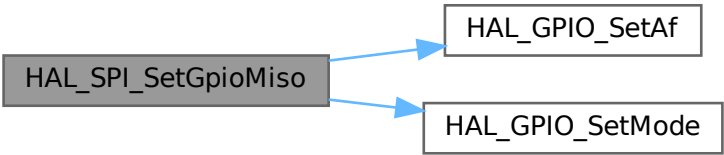
spi	SPI模块号
pin	GPIO引脚

参见

需要先初始化SPI

在文件 [hal_lks32mc08x_spi.c](#) 第 194 行定义.

函数调用图:



4.106.1.18 HAL_SPI_SetGpioMosi()

```
void HAL_SPI_SetGpioMosi (
    HAL_SPI_TypeDef* spi,
    GPIO_TypeDef* pin )
```

初始化Mosi引脚

参数

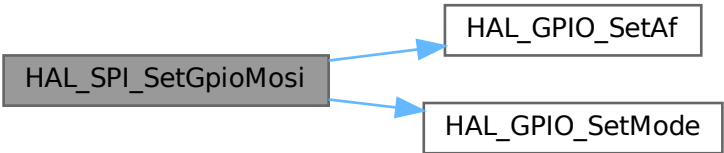
<i>spi</i>	SPI模块号
<i>pin</i>	GPIO引脚

参见

需要先初始化SPI

在文件 [hal_lks32mc08x_spi.c](#) 第 172 行定义.

函数调用图:



4.106.1.19 HAL_SPI_SetMode()

```
void HAL_SPI_SetMode (
    HAL_SPI_TypeDef* spi,
    HAL_SPI_Mode_TypeDef mode )
```

设置SPI模式

参数

<i>spi</i>	SPI模块号
<i>mode</i>	SPI模式

在文件 [hal_lks32mc08x_spi.c](#) 第 86 行定义.

4.106.1.20 HAL_SPI_TxData()

```
uint8_t HAL_SPI_TxData (
    HAL_SPI_TypeDef* spi,
    uint8_t* data,
    uint32_t len )
```

发送数据

参数

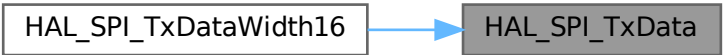
<i>spi</i>	SPI模块号
<i>data</i>	要发送的数据
<i>len</i>	数据长度

返回

发送状态

在文件 [hal_lks32mc08x_spi.c](#) 第 263 行定义.

这是这个函数的调用关系图:



4.106.1.21 HAL_SPI_TxDataWidth16()

```
uint8_t HAL_SPI_TxDataWidth16 (
    HAL_SPI_TypeDef * spi,
    uint16_t * data,
    uint32_t len )
```

发送数据

参数

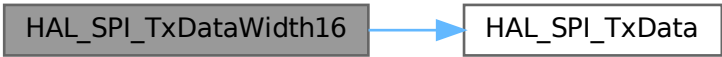
<i>spi</i>	SPI模块号
<i>data</i>	要发送的数据
<i>len</i>	数据长度，单位2字节

返回
发送状态

注解
lks32mc08x不支持16位模式，这里用8位模式发送

在文件 [hal_lks32mc08x_spi.c](#) 第 340 行定义.

函数调用图:



4.107 hal_lks32mc08x_spi.c

[浏览该文件的文档.](#)

```
00001
00002 #include "../include/hal_lksmcu_spi.h"
00003 #include "../hal_lks32mc08x.h"
00004 #include "lks32mc08x_user_manual.h"
00005 #include "../include/hal_lksmcu_reg.h"
00006 #include "../include/hal_lks32mc08x_reg.h"
00007
00012 void HAL_SPI_Reset (HAL_SPI_TypeDef * spi)
00013 {
00014     (void) spi;
00015     HAL_REG_SYS_UNLOCK();
00016     HAL_REG_SET(SYS_SFT_RST, SYS_SFT_RST_SPI_SFT_RST_BIT);
00017     HAL_REG_RESET(SYS_SFT_RST, SYS_SFT_RST_SPI_SFT_RST_BIT);
00018     HAL_REG_SYS_LOCK();
00019 }
00020
```

```

00025 void HAL_SPI_Enable(HAL_SPI_TypeDef spi)
00026 {
00027     HAL_REG_SYS_UNLOCK();
00028     HAL_REG_SET(SYS_CLK_FEN, SYS_CLK_FEN_SPI_CKLEN_BIT);
00029     HAL_REG_SYS_LOCK();
00030     HAL_REG_SET(HAL_SPI[spi]->CFG, SPI_CFG_EN_BIT);
00031 }
00032
00037 void HAL_SPI_Disable(HAL_SPI_TypeDef spi)
00038 {
00039     HAL_REG_RESET(HAL_SPI[spi]->CFG, SPI_CFG_EN_BIT);
00040     HAL_REG_SYS_UNLOCK();
00041     HAL_REG_RESET(SYS_CLK_FEN, SYS_CLK_FEN_SPI_CKLEN_BIT);
00042     HAL_REG_SYS_LOCK();
00043 }
00044
00049 void HAL_SPI_SetBaud(HAL_SPI_TypeDef spi, uint32_t baud)
00050 {
00051     uint32_t mclk = HAL_CLK_GetMcuFreq();
00052     uint32_t cnt = (mclk / (2 * baud)) - 1;
00053     HAL_REG_WRITE_BIT_MASK_POS(HAL_SPI[spi]->BAUD, SPI_DIV_BAUD, cnt);
00054 }
00055
00056 // 设置时钟分频
00057 void HAL_SPI_SetClockDiv(HAL_SPI_TypeDef spi, uint32_t div)
00058 {
00059     if (div < 2) div = 2;
00060     HAL_REG_WRITE_BIT_MASK_POS(HAL_SPI[spi]->BAUD, SPI_DIV_BAUD, div / 2 - 1);
00061 }
00062
00063 // 获取时钟分频实际除数
00064 uint32_t HAL_SPI_GetClockDiv(HAL_SPI_TypeDef spi)
00065 {
00066     return 2 * (HAL_REG_READ_BIT_MASK_POS(HAL_SPI[spi]->BAUD, SPI_DIV_BAUD) + 1);
00067 }
00068
00069 // 获取SCK时钟频率
00070 uint32_t HAL_SPI_GetClockFreq(HAL_SPI_TypeDef spi)
00071 {
00072     return HAL_CLK_GetMcuFreq() / HAL_SPI_GetClockDiv(spi);
00073 }
00074
00075 // 获取当前波特率(=SCK频率)
00076 uint32_t HAL_SPI_GetBaud(HAL_SPI_TypeDef spi)
00077 {
00078     return HAL_SPI_GetClockFreq(spi);
00079 }
00080
00086 void HAL_SPI_SetMode(HAL_SPI_TypeDef spi, HAL_SPI_Mode_TypeDef mode)
00087 {
00088     uint32_t mask = SPI_CFG_MS_BIT | SPI_CFG_SAMPLE_BIT | SPI_CFG_CLK_POL_BIT | SPI_CFG_ENDIAN_BIT;
00089     switch (mode)
00090     {
00091         case HAL_SPI_MODE_MASTER_LSB_00:
00092             HAL_REG_WRITE_BIT(HAL_SPI[spi]->CFG, mask, SPI_CFG_MS_BIT | SPI_CFG_ENDIAN_BIT);
00093             break;
00094         case HAL_SPI_MODE_MASTER_LSB_01:
00095             HAL_REG_WRITE_BIT(HAL_SPI[spi]->CFG, mask, SPI_CFG_MS_BIT | SPI_CFG_CLK_POL_BIT |
SPI_CFG_ENDIAN_BIT);
00096             break;
00097         case HAL_SPI_MODE_MASTER_LSB_10:
00098             HAL_REG_WRITE_BIT(HAL_SPI[spi]->CFG, mask, SPI_CFG_MS_BIT | SPI_CFG_SAMPLE_BIT |
SPI_CFG_ENDIAN_BIT);
00099             break;
00100         case HAL_SPI_MODE_MASTER_LSB_11:
00101             HAL_REG_WRITE_BIT(HAL_SPI[spi]->CFG, mask, SPI_CFG_MS_BIT | SPI_CFG_SAMPLE_BIT |
SPI_CFG_CLK_POL_BIT | SPI_CFG_ENDIAN_BIT);
00102             break;
00103         case HAL_SPI_MODE_MASTER_MSB_00:
00104             HAL_REG_WRITE_BIT(HAL_SPI[spi]->CFG, mask, SPI_CFG_MS_BIT | 0);
00105             break;
00106         case HAL_SPI_MODE_MASTER_MSB_01:
00107             HAL_REG_WRITE_BIT(HAL_SPI[spi]->CFG, mask, SPI_CFG_MS_BIT | SPI_CFG_CLK_POL_BIT);
00108             break;
00109         case HAL_SPI_MODE_MASTER_MSB_10:
00110             HAL_REG_WRITE_BIT(HAL_SPI[spi]->CFG, mask, SPI_CFG_MS_BIT | SPI_CFG_SAMPLE_BIT);
00111             break;
00112         case HAL_SPI_MODE_MASTER_MSB_11:
00113             HAL_REG_WRITE_BIT(HAL_SPI[spi]->CFG, mask, SPI_CFG_MS_BIT | SPI_CFG_SAMPLE_BIT |
SPI_CFG_CLK_POL_BIT);
00114             break;
00115         case HAL_SPI_MODE_SLAVE_LSB_00:
00116             HAL_REG_WRITE_BIT(HAL_SPI[spi]->CFG, mask, SPI_CFG_ENDIAN_BIT);
00117             break;
00118         case HAL_SPI_MODE_SLAVE_LSB_01:
00119             HAL_REG_WRITE_BIT(HAL_SPI[spi]->CFG, mask, SPI_CFG_CLK_POL_BIT | SPI_CFG_ENDIAN_BIT);
00120             break;

```

```

00121         case HAL_SPI_MODE_SLAVE_LSB_10:
00122             HAL_REG_WRITE_BIT(HAL_SPI[spi]->CFG, mask, SPI_CFG_SAMPLE_BIT | SPI_CFG_ENDIAN_BIT);
00123             break;
00124         case HAL_SPI_MODE_SLAVE_LSB_11:
00125             HAL_REG_WRITE_BIT(HAL_SPI[spi]->CFG, mask, SPI_CFG_SAMPLE_BIT | SPI_CFG_CLK_POL_BIT |
SPI_CFG_ENDIAN_BIT);
00126             break;
00127         case HAL_SPI_MODE_SLAVE_MSB_00:
00128             HAL_REG_WRITE_BIT(HAL_SPI[spi]->CFG, mask, 0);
00129             break;
00130         case HAL_SPI_MODE_SLAVE_MSB_01:
00131             HAL_REG_WRITE_BIT(HAL_SPI[spi]->CFG, mask, SPI_CFG_CLK_POL_BIT);
00132             break;
00133         case HAL_SPI_MODE_SLAVE_MSB_10:
00134             HAL_REG_WRITE_BIT(HAL_SPI[spi]->CFG, mask, SPI_CFG_SAMPLE_BIT);
00135             break;
00136         case HAL_SPI_MODE_SLAVE_MSB_11:
00137             HAL_REG_WRITE_BIT(HAL_SPI[spi]->CFG, mask, SPI_CFG_SAMPLE_BIT | SPI_CFG_CLK_POL_BIT);
00138             break;
00139         default:
00140             break;
00141     }
00142 }
00143
00150 void HAL_SPI_SetGpioCs(HAL_SPI_TypeDef spi, HAL_GPIO_TypeDef pin)
00151 {
00152     if (HAL_REG_READ_BIT(HAL_SPI[spi]->CFG, SPI_CFG_MS_BIT) == 0)
00153     {
00154         // 从模式
00155         HAL_GPIO_SetAf(pin, HAL_GPIO_AF_SPI);
00156         HAL_GPIO_SetMode(pin, HAL_GPIO_MODE_IN);
00157     }
00158     else
00159     {
00160         // 主模式
00161         HAL_GPIO_SetAf(pin, HAL_GPIO_AF_SPI);
00162         HAL_GPIO_SetMode(pin, HAL_GPIO_MODE_OUT);
00163     }
00164 }
00165
00172 void HAL_SPI_SetGpioMosi(HAL_SPI_TypeDef spi, HAL_GPIO_TypeDef pin)
00173 {
00174     if (HAL_REG_READ_BIT(HAL_SPI[spi]->CFG, SPI_CFG_MS_BIT) == 0)
00175     {
00176         // 从模式
00177         HAL_GPIO_SetAf(pin, HAL_GPIO_AF_SPI);
00178         HAL_GPIO_SetMode(pin, HAL_GPIO_MODE_IN);
00179     }
00180     else
00181     {
00182         // 主模式
00183         HAL_GPIO_SetAf(pin, HAL_GPIO_AF_SPI);
00184         HAL_GPIO_SetMode(pin, HAL_GPIO_MODE_OUT);
00185     }
00186 }
00187
00194 void HAL_SPI_SetGpioMiso(HAL_SPI_TypeDef spi, HAL_GPIO_TypeDef pin)
00195 {
00196     if (HAL_REG_READ_BIT(HAL_SPI[spi]->CFG, SPI_CFG_MS_BIT) == 0)
00197     {
00198         // 从模式
00199         HAL_GPIO_SetAf(pin, HAL_GPIO_AF_SPI);
00200         HAL_GPIO_SetMode(pin, HAL_GPIO_MODE_OUT);
00201     }
00202     else
00203     {
00204         // 主模式
00205         HAL_GPIO_SetAf(pin, HAL_GPIO_AF_SPI);
00206         HAL_GPIO_SetMode(pin, HAL_GPIO_MODE_IN);
00207     }
00208 }
00209
00216 void HAL_SPI_SetGpioClk(HAL_SPI_TypeDef spi, HAL_GPIO_TypeDef pin)
00217 {
00218     if (HAL_REG_READ_BIT(HAL_SPI[spi]->CFG, SPI_CFG_MS_BIT) == 0)
00219     {
00220         // 从模式
00221         HAL_GPIO_SetAf(pin, HAL_GPIO_AF_SPI);
00222         HAL_GPIO_SetMode(pin, HAL_GPIO_MODE_IN);
00223     }
00224     else
00225     {
00226         // 主模式
00227         HAL_GPIO_SetAf(pin, HAL_GPIO_AF_SPI);
00228         HAL_GPIO_SetMode(pin, HAL_GPIO_MODE_OUT);
00229     }
00230 }

```

```

00231
00237 void HAL_SPI_SetGpioDi(HAL_SPI_TypeDef spi, HAL_GPIO_TypeDef pin)
00238 {
00239     (void)spi;
00240     HAL_GPIO_SetAf(pin, HAL_GPIO_AF_SPI);
00241     HAL_GPIO_SetMode(pin, HAL_GPIO_MODE_IN);
00242 }
00243
00249 void HAL_SPI_SetGpioDo(HAL_SPI_TypeDef spi, HAL_GPIO_TypeDef pin)
00250 {
00251     (void)spi;
00252     HAL_GPIO_SetAf(pin, HAL_GPIO_AF_SPI);
00253     HAL_GPIO_SetMode(pin, HAL_GPIO_MODE_OUT);
00254 }
00255
00263 uint8_t HAL_SPI_TxData(HAL_SPI_TypeDef spi, uint8_t *data, uint32_t len)
00264 {
00265     uint32_t i = 0;
00266     HAL_REG_SET(HAL_SPI[spi]->BAUD, SPI_DIV_TRANS_MODE_BIT);
00267     HAL_REG_WRITE_BIT_MASK_POS(HAL_SPI[spi]->CFG, SPI_CFG_DUPLEX, 2);
00268     for (i = 0; i < len; i++)
00269     {
00270         HAL_REG_WRITE(HAL_SPI[spi]->SIZE, 1);
00271         HAL_REG_WRITE_BIT(HAL_SPI[spi]->IE, SPI_IE_CMPLT_IF_BIT | SPI_IE_AB_IF_BIT | SPI_IE_OV_IF_BIT,
SPI_IE_CMPLT_IF_BIT);
00272         HAL_REG_WRITE(HAL_SPI[spi]->TXDATA, data[i]);
00273         while (HAL_REG_READ(HAL_SPI[spi]->IE & SPI_IE_CMPLT_IF_BIT) == 0)
00274         {
00275         }
00276     }
00277     return 0;
00278 }
00279
00287 uint8_t HAL_SPI_RxData(HAL_SPI_TypeDef spi, uint8_t *data, uint32_t len)
00288 {
00289     uint32_t i = 0;
00290     HAL_REG_SET(HAL_SPI[spi]->BAUD, SPI_DIV_TRANS_MODE_BIT);
00291     HAL_REG_WRITE_BIT_MASK_POS(HAL_SPI[spi]->CFG, SPI_CFG_DUPLEX, 3);
00292     for (i = 0; i < len; i++)
00293     {
00294         HAL_REG_WRITE(HAL_SPI[spi]->SIZE, 1);
00295         HAL_REG_WRITE_BIT(HAL_SPI[spi]->IE, SPI_IE_CMPLT_IF_BIT | SPI_IE_AB_IF_BIT | SPI_IE_OV_IF_BIT,
SPI_IE_CMPLT_IF_BIT);
00296         HAL_REG_WRITE(HAL_SPI[spi]->TXDATA, 0); // 发送一个空字节,以触发接收
00297         while (HAL_REG_READ(HAL_SPI[spi]->IE & SPI_IE_CMPLT_IF_BIT) == 0)
00298         {
00299         }
00300         data[i] = HAL_REG_READ(HAL_SPI[spi]->RXDATA);
00301     }
00302     // 实现接收数据的代码
00303     return 0; // 返回接收状态
00304 }
00305
00314 uint8_t HAL_SPI_RxTxData(HAL_SPI_TypeDef spi, uint8_t *rxdata, uint8_t *txdata, uint32_t len)
00315 {
00316     uint32_t i = 0;
00317     HAL_REG_SET(HAL_SPI[spi]->BAUD, SPI_DIV_TRANS_MODE_BIT);
00318     HAL_REG_WRITE_BIT_MASK_POS(HAL_SPI[spi]->CFG, SPI_CFG_DUPLEX, 0);
00319     for (i = 0; i < len; i++)
00320     {
00321         HAL_REG_WRITE(HAL_SPI[spi]->SIZE, 1);
00322         HAL_REG_WRITE_BIT(HAL_SPI[spi]->IE, SPI_IE_CMPLT_IF_BIT | SPI_IE_AB_IF_BIT | SPI_IE_OV_IF_BIT,
SPI_IE_CMPLT_IF_BIT);
00323         HAL_REG_WRITE(HAL_SPI[spi]->TXDATA, txdata[i]);
00324         while (HAL_REG_READ(HAL_SPI[spi]->IE & SPI_IE_CMPLT_IF_BIT) == 0)
00325         {
00326         }
00327         rxdata[i] = HAL_REG_READ(HAL_SPI[spi]->RXDATA);
00328     }
00329     return 0; // 返回发送和接收状态
00330 }
00331
00340 uint8_t HAL_SPI_TxDataWidth16(HAL_SPI_TypeDef spi, uint16_t *data, uint32_t len)
00341 {
00342     HAL_SPI_TxData(spi, (uint8_t *)data, len * 2);
00343     return 0;
00344 }
00345
00354 uint8_t HAL_SPI_RxDataWidth16(HAL_SPI_TypeDef spi, uint16_t *data, uint32_t len)
00355 {
00356     HAL_SPI_RxData(spi, (uint8_t *)data, len * 2);
00357     return 0; // 返回接收状态
00358 }
00359
00369 uint8_t HAL_SPI_RxTxDataWidth16(HAL_SPI_TypeDef spi, uint16_t *rxdata, uint16_t *txdata, uint32_t len)
00370 {
00371     HAL_SPI_RxTxData(spi, (uint8_t *)rxdata, (uint8_t *)txdata, len * 2);

```

```
00372     return 0; // 返回发送和接收状态
00373 }
```

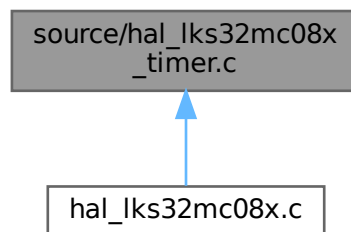
4.108 source/hal_lks32mc08x_timer.c 文件参考

```
#include "../include/hal_lksmcutimer.h"
#include "../include/hal_lksmcu_gpio.h"
#include "../hal_lks32mc08x.h"
#include "lks32mc08x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc08x_reg.h"
```

hal_lks32mc08x_timer.c 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



函数

- void **HAL_TIMER_Reset** (HAL_TIMER_x timer_x)
- void **HAL_TIMER_Enable** (HAL_TIMER_x timer_x)
- void **HAL_TIMER_Disable** (HAL_TIMER_x timer_x)
- void **HAL_TIMER_SetPeriodCount** (HAL_TIMER_x timer_x, uint32_t count)
设置定时器周期
- void **HAL_TIMER_SetFreq** (HAL_TIMER_x timer_x, uint32_t freq)
设置定时器频率
- void **HAL_TIMER_SetPeriodms** (HAL_TIMER_x timer_x, uint32_t ms)
设置定时器周期(ms)
- void **HAL_TIMER_SetPeriodus** (HAL_TIMER_x timer_x, uint32_t us)
设置定时器周期(us)
- uint32_t **HAL_TIMER_GetFreq** (HAL_TIMER_x timer_x)
- void **HAL_TIMER_SetClockDiv** (HAL_TIMER_x timer_x, uint32_t div)

- uint32_t HAL_TIMER_GetClockDiv (HAL_TIMER_x timer_x)
- uint32_t HAL_TIMER_GetClockFreq (HAL_TIMER_x timer_x)
- uint32_t HAL_TIMER_GetThCnt (HAL_TIMER_x timer_x)
- void HAL_TIMER_StartCounting (HAL_TIMER_x timer_x)
- void HAL_TIMER_StopCounting (HAL_TIMER_x timer_x)
- void HAL_TIMER_SetPwm (HAL_TIMER_x timer_x, int32_t t0, int32_t t1)
- void HAL_TIMER_SetPwmQ15 (HAL_TIMER_x timer_x, int16_t t0, int16_t t1)
- void HAL_TIMER_SetWorkMode (HAL_TIMER_x timer_x, HAL_TIMER_MODE_x mode_x)
- void HAL_TIMER_SetGpioOut (HAL_TIMER_x timer_x, HAL_GPIO_Px px)
设置定时器输出 *gpio*
- void HAL_TIMER_SetGpioIn (HAL_TIMER_x timer_x, HAL_GPIO_Px px)
设置定时器输入 *gpio*

变量

- static uint8_t hal_timer_ena = 0

4.108.1 函数说明

4.108.1.1 HAL_TIMER_Disable()

```
void HAL_TIMER_Disable (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc08x_timer.c](#) 第 27 行定义.

4.108.1.2 HAL_TIMER_Enable()

```
void HAL_TIMER_Enable (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc08x_timer.c](#) 第 16 行定义.

这是这个函数的调用关系图:

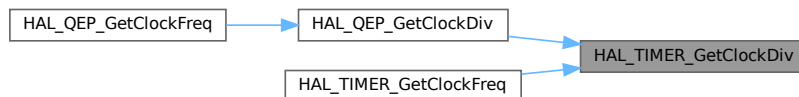


4.108.1.3 HAL_TIMER_GetClockDiv()

```
uint32_t HAL_TIMER_GetClockDiv (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc08x.timer.c](#) 第 154 行定义.

这是这个函数的调用关系图:

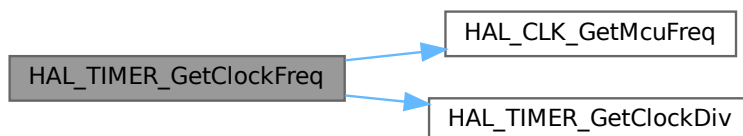


4.108.1.4 HAL_TIMER_GetClockFreq()

```
uint32_t HAL_TIMER_GetClockFreq (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc08x.timer.c](#) 第 167 行定义.

函数调用图:



4.108.1.5 HAL_TIMER_GetFreq()

```
uint32_t HAL_TIMER_GetFreq (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc08x.timer.c](#) 第 123 行定义.

函数调用图:



4.108.1.6 HAL_TIMER_GetThCnt()

```
uint32_t HAL_TIMER_GetThCnt (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc08x.timer.c](#) 第 173 行定义.

这是这个函数的调用关系图:



4.108.1.7 HAL_TIMER_Reset()

```
void HAL_TIMER_Reset (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc08x.timer.c](#) 第 11 行定义.

4.108.1.8 HAL_TIMER_SetClockDiv()

```
void HAL_TIMER_SetClockDiv (
    HAL_TIMER_x timer_x,
    uint32_t div )
```

在文件 [hal_lks32mc08x.timer.c](#) 第 136 行定义.

这是这个函数的调用关系图:



4.108.1.9 HAL_TIMER_SetFreq()

```
void HAL_TIMER_SetFreq (
    HAL_TIMER_x timer_x,
    uint32_t freq )
```

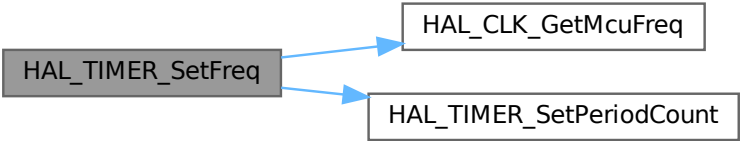
设置定时器频率

参数

<i>timer</i> ↔ <i>_x</i>	定时器编号
<i>freq</i>	定时器频率

在文件 [hal_lks32mc08x_timer.c](#) 第 92 行定义.

函数调用图:



4.108.1.10 HAL_TIMER.SetGpioIn()

```
void HAL_TIMER.SetGpioIn (
    HAL_TIMER_x timer_x,
    HAL_GPIO_Px px )
```

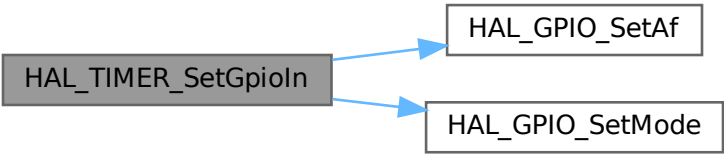
设置定时器输入gpio

参数

<i>timer</i> ↔ <i>_x</i>	定时器编号
<i>px</i>	gpio编号

在文件 [hal_lks32mc08x_timer.c](#) 第 296 行定义.

函数调用图:



4.108.1.11 HAL_TIMER.SetGpioOut()

```
void HAL_TIMER.SetGpioOut (
    HAL_TIMER_x timer_x,
    HAL_GPIO_Px px )
```

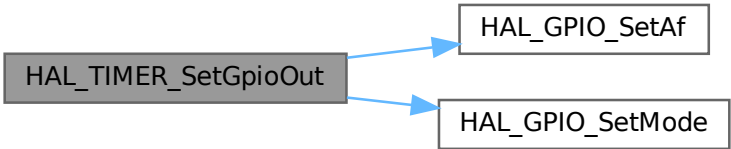
设置定时器输出gpio

参数

<i>timer_x</i>	定时器编号
<i>px</i>	gpio编号

在文件 [hal_lks32mc08x.timer.c](#) 第 266 行定义.

函数调用图:



4.108.1.12 HAL_TIMER.SetPeriodCount()

```
void HAL_TIMER.SetPeriodCount (
    HAL_TIMER_x timer_x,
    uint32_t count )
```

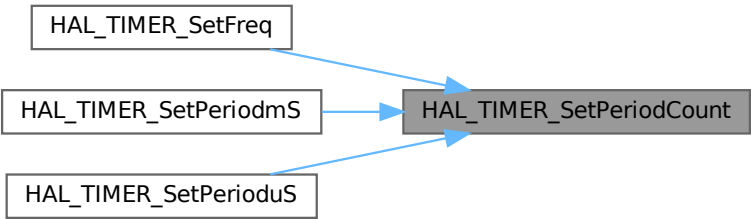
设置定时器周期

参数

<i>timer_x</i>	定时器编号
<i>count</i>	定时器周期

在文件 [hal_lks32mc08x.timer.c](#) 第 45 行定义.

这是这个函数的调用关系图:



4.108.1.13 HAL_TIMER.SetPeriodmS()

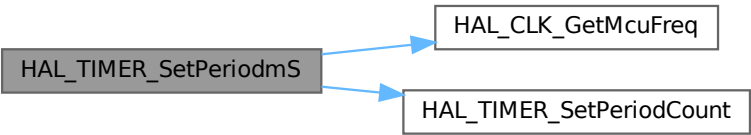
```
void HAL_TIMER.SetPeriodmS (
    HAL_TIMER.x timer_x,
    uint32_t ms )
```

设置定时器周期(ms)

参数

<i>timer_x</i>	定时器编号
<i>ms</i>	定时器周期(ms)

在文件 [hal_lks32mc08x.timer.c](#) 第 104 行定义.
函数调用图:



4.108.1.14 HAL_TIMER.SetPerioduS()

```
void HAL_TIMER.SetPerioduS (
    HAL_TIMER.x timer_x,
    uint32_t us )
```

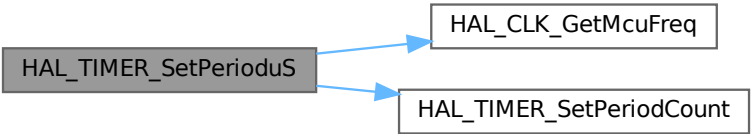
设置定时器周期(us)

参数

<i>timer</i> ↔ <i>_x</i>	定时器编号
<i>us</i>	定时器周期(us)

在文件 [hal_lks32mc08x.timer.c](#) 第 116 行定义.

函数调用图:



4.108.1.15 HAL_TIMER.SetPwm()

```
void HAL_TIMER.SetPwm (
    HAL_TIMER_x timer_x,
    int32_t t0,
    int32_t t1 )
```

在文件 [hal_lks32mc08x.timer.c](#) 第 221 行定义.

4.108.1.16 HAL_TIMER.SetPwmQ15()

```
void HAL_TIMER.SetPwmQ15 (
    HAL_TIMER_x timer_x,
    int16_t t0,
    int16_t t1 )
```

在文件 [hal_lks32mc08x.timer.c](#) 第 227 行定义.

函数调用图:



4.108.1.17 HAL_TIMER_SetWorkMode()

```
void HAL_TIMER_SetWorkMode (
    HAL_TIMER_x timer_x,
    HAL_TIMER_MODE_x mode_x )
```

< 计数模式

< PWM输出模式

< PWM输入模式

在文件 [hal_lks32mc08x.timer.c](#) 第 236 行定义.

4.108.1.18 HAL_TIMER_StartCounting()

```
void HAL_TIMER_StartCounting (
    HAL_TIMER_x timer_x )
```

< TIMER模块0

< TIMER模块1

< TIMER模块2

< TIMER模块3

在文件 [hal_lks32mc08x.timer.c](#) 第 178 行定义.

4.108.1.19 HAL_TIMER_StopCounting()

```
void HAL_TIMER_StopCounting (
    HAL_TIMER_x timer_x )
```

< TIMER模块0

< TIMER模块1

< TIMER模块2

< TIMER模块3

在文件 [hal_lks32mc08x.timer.c](#) 第 199 行定义.

4.108.2 变量说明

4.108.2.1 hal_timer_ena

```
uint8_t hal_timer_ena = 0 [static]
```

在文件 [hal_lks32mc08x.timer.c](#) 第 9 行定义.

4.109 hal_lks32mc08x_timer.c

[浏览该文件的文档.](#)

```

00001
00002 #include "../include/hal_lksmcu_timer.h"
00003 #include "../include/hal_lksmcu_gpio.h"
00004 #include "../hal_lks32mc08x.h"
00005 #include "lks32mc08x_user_manual.h"
00006 #include "../include/hal_lksmcu_reg.h"
00007 #include "../include/hal_lks32mc08x_reg.h"
00008
00009 static uint8_t hal_timer_ena = 0;
00010 // 模块复位
00011 void HAL_TIMER_Reset(HAL_TIMER_x timer_x)
00012 {
00013     (void)timer_x;
00014 }
00015 // 模块打开
00016 void HAL_TIMER_Enable(HAL_TIMER_x timer_x)
00017 {
00018     (void)timer_x;
00019     HAL_REG_SET(hal_timer_ena, 1 << timer_x);
00020     // 系统寄存器解锁
00021     HAL_REG_SYS_UNLOCK();
00022     HAL_REG_SET(SYS_CLK_FEN, SYS_CLK_FEN_UTIMER_CLK_EN_BIT);
00023     // 系统寄存器上锁
00024     HAL_REG_SYS_LOCK();
00025 }
00026 // 模块关闭
00027 void HAL_TIMER_Disable(HAL_TIMER_x timer_x)
00028 {
00029     HAL_REG_RESET(hal_timer_ena, 1 << timer_x);
00030     if (hal_timer_ena == 0)
00031     {
00032         // 系统寄存器解锁
00033         HAL_REG_SYS_UNLOCK();
00034         HAL_REG_RESET(SYS_CLK_FEN, SYS_CLK_FEN_UTIMER_CLK_EN_BIT);
00035         // 系统寄存器上锁
00036         HAL_REG_SYS_LOCK();
00037     }
00038 }
00039
00040 void HAL_TIMER_SetPeriodCount(HAL_TIMER_x timer_x, uint32_t count)
00041 {
00042     if (count > 0)
00043     {
00044         count = count - 1;
00045     }
00046     if ((timer_x == HAL_TIMER_0) || (timer_x == HAL_TIMER_1))
00047     {
00048         uint32_t div = 0;
00049         for (; count > UTIMER_UNT0_TH_UNT0_TH_MAX; )
00050         {
00051             count >>= 1;
00052             div++;
00053         }
00054         if (div > UTIMER_UNT0_CFG_CLK_DIV_MAX)
00055         {
00056             div = UTIMER_UNT0_CFG_CLK_DIV_MAX;
00057             count = UTIMER_UNT0_TH_UNT0_TH_MAX;
00058         }
00059         HAL_REG_WRITE_BIT_MASK_POS(HAL_TIMER[timer_x]->CFG, UTIMER_UNT0_CFG_CLK_DIV, div);
00060         HAL_REG_WRITE_BIT_MASK_POS(HAL_TIMER[timer_x]->TH, UTIMER_UNT0_TH_UNT0_TH, count);
00061     }
00062     else
00063     {
00064         // 对于32位的定时器, count不需要判断
00065         // uint32_t div = 0;
00066         // for (; count > UTIMER_UNT2_TH_UNT2_TH_MAX; )
00067         // {
00068         //     count >>= 1;
00069         //     div++;
00070         // }
00071         // if (div > UTIMER_UNT2_CFG_CLK_DIV_MAX)
00072         // {
00073         //     div = UTIMER_UNT2_CFG_CLK_DIV_MAX;
00074         //     count = UTIMER_UNT2_TH_UNT2_TH_MAX;
00075         // }
00076         HAL_REG_WRITE_BIT_MASK_POS(HAL_TIMER[timer_x]->CFG, UTIMER_UNT2_CFG_CLK_DIV, 0);
00077         HAL_REG_WRITE_BIT_MASK_POS(HAL_TIMER[timer_x]->TH, UTIMER_UNT2_TH_UNT2_TH, count);
00078     }
00079 }
00080
00081 void HAL_TIMER_SetFreq(HAL_TIMER_x timer_x, uint32_t freq)

```

```

00093 {
00094     uint32_t mclk = HAL_CLK_GetMcuFreq();
00095     uint32_t th   = (mclk / freq);
00096     HAL_TIMER_SetPeriodCount(timer_x, th);
00097 }
00098
00104 void HAL_TIMER_SetPeriodMS(HAL_TIMER_x timer_x, uint32_t ms)
00105 {
00106     uint32_t mclk = HAL_CLK_GetMcuFreq();
00107     uint32_t th   = (mclk / 1000) * ms;
00108     HAL_TIMER_SetPeriodCount(timer_x, th);
00109 }
00110
00116 void HAL_TIMER_SetPeriodUS(HAL_TIMER_x timer_x, uint32_t us)
00117 {
00118     uint32_t mclk = HAL_CLK_GetMcuFreq();
00119     uint32_t th   = (mclk / 1000000) * us;
00120     HAL_TIMER_SetPeriodCount(timer_x, th);
00121 }
00122 // 获取当前的pwm频率
00123 uint32_t HAL_TIMER_GetFreq(HAL_TIMER_x timer_x)
00124 {
00125     uint32_t mclk;
00126     uint32_t th;
00127     uint32_t freq;
00128     uint32_t div = HAL_REG_READ_BIT_MASK_POS(HAL_TIMER[timer_x]->CFG, UTIMER_UNT0_CFG_CLK_DIV);
00129
00130     mclk = HAL_CLK_GetMcuFreq();
00131     th   = HAL_REG_READ(HAL_TIMER[timer_x]->TH);
00132     freq = (mclk >> div) / (th + 1);
00133     return freq;
00134 }
00135 // 设置时钟分频
00136 void HAL_TIMER_SetClockDiv(HAL_TIMER_x timer_x, uint32_t div)
00137 {
00138     uint32_t shift = 0;
00139     while ((1u << shift) < div && shift < UTIMER_UNT0_CFG_CLK_DIV_MAX)
00140     {
00141         shift++;
00142     }
00143     if ((timer_x == HAL_TIMER_0) || (timer_x == HAL_TIMER_1))
00144     {
00145         HAL_REG_WRITE_BIT_MASK_POS(HAL_TIMER[timer_x]->CFG, UTIMER_UNT0_CFG_CLK_DIV, shift);
00146     }
00147     else
00148     {
00149         HAL_REG_WRITE_BIT_MASK_POS(HAL_TIMER[timer_x]->CFG, UTIMER_UNT2_CFG_CLK_DIV, shift);
00150     }
00151 }
00152
00153 // 获取时钟分频实际除数
00154 uint32_t HAL_TIMER_GetClockDiv(HAL_TIMER_x timer_x)
00155 {
00156     if ((timer_x == HAL_TIMER_0) || (timer_x == HAL_TIMER_1))
00157     {
00158         return 1u << HAL_REG_READ_BIT_MASK_POS(HAL_TIMER[timer_x]->CFG, UTIMER_UNT0_CFG_CLK_DIV);
00159     }
00160     else
00161     {
00162         return 1u << HAL_REG_READ_BIT_MASK_POS(HAL_TIMER[timer_x]->CFG, UTIMER_UNT2_CFG_CLK_DIV);
00163     }
00164 }
00165
00166 // 获取分频后工作时钟
00167 uint32_t HAL_TIMER_GetClockFreq(HAL_TIMER_x timer_x)
00168 {
00169     return HAL_CLK_GetMcuFreq() / HAL_TIMER_GetClockDiv(timer_x);
00170 }
00171
00172 // 获取计数器最大值
00173 uint32_t HAL_TIMER_GetThCnt(HAL_TIMER_x timer_x)
00174 {
00175     return HAL_REG_READ(HAL_TIMER[timer_x]->TH);
00176 }
00177 // 开始计数
00178 void HAL_TIMER_StartCounting(HAL_TIMER_x timer_x)
00179 {
00180     switch (timer_x)
00181     {
00182     case HAL_TIMER_0:
00183         HAL_REG_SET(UTIMER_CFG, UTIMER_CFG_TIMER0_EN_BIT);
00184         break;
00185     case HAL_TIMER_1:
00186         HAL_REG_SET(UTIMER_CFG, UTIMER_CFG_TIMER1_EN_BIT);
00187         break;
00188     case HAL_TIMER_2:
00189         HAL_REG_SET(UTIMER_CFG, UTIMER_CFG_TIMER2_EN_BIT);

```



```

00190         break;
00191     case HAL_TIMER_3:
00192         HAL_REG_SET(UTIMER_CFG, UTIMER_CFG_TIMER3_EN_BIT);
00193         break;
00194     default:
00195         break;
00196 }
00197 }
00198 // 停止计数(同时计数器回到0)
00199 void HAL_TIMER_StopCounting(HAL_TIMER_x timer_x)
00200 {
00201     switch (timer_x)
00202     {
00203     case HAL_TIMER_0:
00204         HAL_REG_RESET(UTIMER_CFG, UTIMER_CFG_TIMER0_EN_BIT);
00205         break;
00206     case HAL_TIMER_1:
00207         HAL_REG_RESET(UTIMER_CFG, UTIMER_CFG_TIMER1_EN_BIT);
00208         break;
00209     case HAL_TIMER_2:
00210         HAL_REG_RESET(UTIMER_CFG, UTIMER_CFG_TIMER2_EN_BIT);
00211         break;
00212     case HAL_TIMER_3:
00213         HAL_REG_RESET(UTIMER_CFG, UTIMER_CFG_TIMER3_EN_BIT);
00214         break;
00215     default:
00216         break;
00217     }
00218     HAL_REG_WRITE(HAL_TIMER[timer_x] -> CNT, 0);
00219 }
00220 // 设置pwm波形 仅PWM模式下有效
00221 void HAL_TIMER_SetPwm(HAL_TIMER_x timer_x, int32_t t0, int32_t t1)
00222 {
00223     HAL_REG_WRITE(HAL_TIMER[timer_x] -> CMPT0, t0);
00224     HAL_REG_WRITE(HAL_TIMER[timer_x] -> CMPT1, t1);
00225 }
00226 // 设置pwm波形q15格式输入 仅PWM模式下有效
00227 void HAL_TIMER_SetPwmQ15(HAL_TIMER_x timer_x, int16_t t0, int16_t t1)
00228 {
00229     uint32_t th = HAL_TIMER_GetThCnt(timer_x);
00230     t0 = t0 * th >> 15;
00231     t1 = t1 * th >> 15;
00232     HAL_REG_WRITE(HAL_TIMER[timer_x] -> CMPT0, t0);
00233     HAL_REG_WRITE(HAL_TIMER[timer_x] -> CMPT1, t1);
00234 }
00235 // 设置工作模式
00236 void HAL_TIMER_SetWorkMode(HAL_TIMER_x timer_x, HAL_TIMER_MODE_x mode_x)
00237 {
00238     switch (mode_x)
00239     {
00240     case HAL_TIMER_MODE_COUNT:
00241         HAL_REG_RESET(HAL_TIMER[timer_x] -> CFG, UTIMER_UNT0_CFG_CH0_MODE_BIT);
00242         HAL_REG_RESET(HAL_TIMER[timer_x] -> CFG, UTIMER_UNT0_CFG_CH1_MODE_BIT);
00243         break;
00244     case HAL_TIMER_MODE_PWMOUT:
00245         HAL_REG_RESET(HAL_TIMER[timer_x] -> CFG, UTIMER_UNT0_CFG_CH0_MODE_BIT);
00246         HAL_REG_RESET(HAL_TIMER[timer_x] -> CFG, UTIMER_UNT0_CFG_CH1_MODE_BIT);
00247         break;
00248     case HAL_TIMER_MODE_PWMIN:
00249         HAL_REG_SET(HAL_TIMER[timer_x] -> CFG, UTIMER_UNT0_CFG_CH0_MODE_BIT);
00250         HAL_REG_SET(HAL_TIMER[timer_x] -> CFG, UTIMER_UNT0_CFG_CH1_MODE_BIT);
00251         HAL_REG_SET(HAL_TIMER[timer_x] -> CFG, UTIMER_UNT0_CFG_CH1_FE_CAP_EN_BIT);
00252         HAL_REG_RESET(HAL_TIMER[timer_x] -> CFG, UTIMER_UNT0_CFG_CH1_RE_CAP_EN_BIT);
00253         HAL_REG_RESET(HAL_TIMER[timer_x] -> CFG, UTIMER_UNT0_CFG_CH0_FE_CAP_EN_BIT);
00254         HAL_REG_SET(HAL_TIMER[timer_x] -> CFG, UTIMER_UNT0_CFG_CH0_RE_CAP_EN_BIT);
00255         break;
00256     default:
00257         break;
00258     }
00259 }
00260
00261 void HAL_TIMER_SetGpioOut(HAL_TIMER_x timer_x, HAL_GPIO_Px px)
00262 {
00263     switch (timer_x)
00264     {
00265     case HAL_TIMER_0:
00266         HAL_GPIO_SetAf(px, HAL_GPIO_AF_TIMER0);
00267         HAL_GPIO_SetMode(px, HAL_GPIO_MODE_OUT);
00268         break;
00269     case HAL_TIMER_1:
00270         HAL_GPIO_SetAf(px, HAL_GPIO_AF_TIMER1);
00271         HAL_GPIO_SetMode(px, HAL_GPIO_MODE_OUT);
00272         break;
00273     case HAL_TIMER_2:
00274         HAL_GPIO_SetAf(px, HAL_GPIO_AF_TIMER2);
00275         HAL_GPIO_SetMode(px, HAL_GPIO_MODE_OUT);
00276         break;
00277     }
00278 }

```

```

00282         case HAL_TIMER_3:
00283             HAL_GPIO_SetAf(px, HAL_GPIO_AF_TIMER3);
00284             HAL_GPIO_SetMode(px, HAL_GPIO_MODE_OUT);
00285             break;
00286         default:
00287             break;
00288     }
00289 }
00290
00296 void HAL_TIMER_SetGpioIn(HAL_TIMER_x timer_x, HAL_GPIO_Px px)
00297 {
00298     switch (timer_x)
00299     {
00300         case HAL_TIMER_0:
00301             HAL_GPIO_SetAf(px, HAL_GPIO_AF_TIMER0);
00302             HAL_GPIO_SetMode(px, HAL_GPIO_MODE_IN);
00303             break;
00304         case HAL_TIMER_1:
00305             HAL_GPIO_SetAf(px, HAL_GPIO_AF_TIMER1);
00306             HAL_GPIO_SetMode(px, HAL_GPIO_MODE_IN);
00307             break;
00308         case HAL_TIMER_2:
00309             HAL_GPIO_SetAf(px, HAL_GPIO_AF_TIMER2);
00310             HAL_GPIO_SetMode(px, HAL_GPIO_MODE_IN);
00311             break;
00312         case HAL_TIMER_3:
00313             HAL_GPIO_SetAf(px, HAL_GPIO_AF_TIMER3);
00314             HAL_GPIO_SetMode(px, HAL_GPIO_MODE_IN);
00315             break;
00316         default:
00317             break;
00318     }
00319 }

```

4.110 source/hal_lks32mc08x_tmp.c 文件参考

```

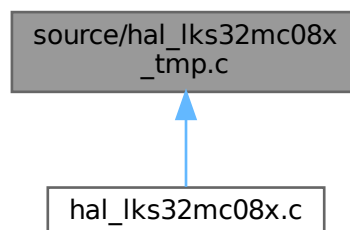
#include "../include/hal_lksmcu_tmp.h"
#include "../include/hal_lksmcu_trim.h"
#include "../hal_lks32mc08x.h"
#include "lks32mc08x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc08x_reg.h"

```

hal_lks32mc08x_tmp.c 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



函数

- void `HAL_TMP_Reset` (`HAL_TMP_x` `TMP_x`)
模块复位
- void `HAL_TMP_Enable` (`HAL_TMP_x` `TMP_x`)
模块打开
- void `HAL_TMP_Disable` (`HAL_TMP_x` `TMP_x`)
模块关闭
- int32_t `HAL_TMP_GetTemperature` (`HAL_TMP_x` `TMP_x`, `HAL_ADC_x` `adc`, `HAL_ADC_SAMP_SEQUENCE_x` `sequence`)
获取当前芯片内部温度传感器测到的温度,单位摄氏度
- int32_t `HAL_TMP_GetTemperature01` (`HAL_TMP_x` `TMP_x`, `HAL_ADC_x` `adc`, `HAL_ADC_SAMP_SEQUENCE_x` `sequence`)
获取当前芯片内部温度传感器测到的温度,单位0.1摄氏度
- int32_t `HAL_TMP_Adc2Temperature` (`HAL_TMP_x` `TMP_x`, int32_t `adc_value`)
adc值转温度
- int32_t `HAL_TMP_Adc2Temperature01` (`HAL_TMP_x` `TMP_x`, int32_t `adc_value`)
adc值转温度

变量

- static int32_t `tmp_coef_a` = 0
- static int32_t `tmp_coef_b` = 0

4.110.1 函数说明

4.110.1.1 HAL_TMP_Adc2Temperature()

```
int32_t HAL_TMP_Adc2Temperature (
    HAL_TMP_x TMP_x,
    int32_t adc_value )
```

adc值转温度

参数

<i>TMP_x</i>	模块编号
<i>adc_value</i>	ADC值，左对齐（-32767 到32767）

返回

当前温度,单位摄氏度

在文件 `hal_lks32mc08x_tmp.c` 第 85 行定义.

函数调用图:



这是这个函数的调用关系图:



4.110.1.2 HAL_TMP_Adc2Temperature01()

```
int32_t HAL_TMP_Adc2Temperature01 (
    HAL_TMP_x TMP_x,
    int32_t adc_value )
```

adc值转温度

参数

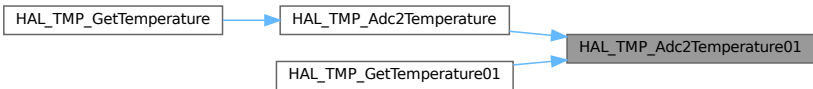
<i>TMP_x</i>	模块编号
<i>adc_value</i>	ADC值, 左对齐 (-32767 到32767)

返回

当前温度,单位0.1摄氏度

在文件 `hal_lks32mc08x.tmp.c` 第 96 行定义.

这是这个函数的调用关系图:



4.110.1.3 HAL_TMP_Disable()

```
void HAL_TMP_Disable (
    HAL_TMP_x TMP_x )
```

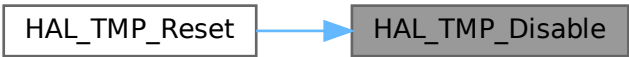
模块关闭

参数

<i>TMP</i> _{<i>x</i>}	模块编号

在文件 [hal_lks32mc08x.tmp.c](#) 第 37 行定义.

这是这个函数的调用关系图:



4.110.1.4 HAL_TMP_Enable()

```
void HAL_TMP_Enable (
    HAL_TMP_x TMP_x )
```

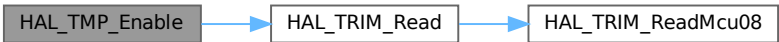
模块打开

参数

<i>TMP</i> _{<i>x</i>}	模块编号

在文件 [hal_lks32mc08x.tmp.c](#) 第 23 行定义.

函数调用图:



4.110.1.5 HAL_TMP_GetTemperature()

```
int32_t HAL_TMP_GetTemperature (
    HAL_TMP_x TMP_x,
    HAL_ADC_x adc,
    HAL_ADC_SAMP_SEQUENCE_x sequence )
```

获取当前芯片内部温度传感器测到的温度,单位摄氏度

参数

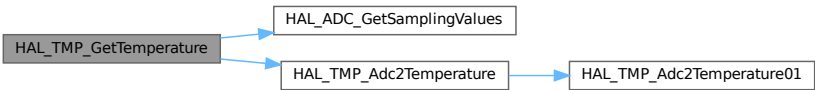
<i>TMP_x</i>	模块编号
<i>adc</i>	ADC编号
<i>sequence</i>	ADC采样序列

返回

当前温度,单位摄氏度

在文件 [hal_lks32mc08x.tmp.c](#) 第 52 行定义.

函数调用图:



4.110.1.6 HAL_TMP_GetTemperature01()

```
int32_t HAL_TMP_GetTemperature01 (
    HAL_TMP_x TMP_x,
    HAL_ADC_x adc,
    HAL_ADC_SAMP_SEQUENCE_x sequence )
```

获取当前芯片内部温度传感器测到的温度,单位0.1摄氏度

参数

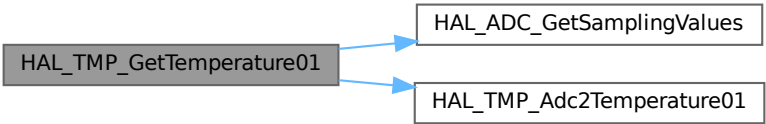
<i>TMP_x</i>	模块编号
<i>adc</i>	ADC编号
<i>sequence</i>	ADC采样序列

返回

当前温度,单位0.1摄氏度

在文件 [hal_lks32mc08x.tmp.c](#) 第 69 行定义.

函数调用图:



4.110.1.7 HAL_TMP_Reset()

```
void HAL_TMP_Reset (
    HAL_TMP_x TMP_x )
```

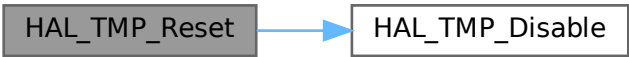
模块复位

参数

<i>TMP_x</i>	模块编号

在文件 [hal_lks32mc08x.tmp.c](#) 第 14 行定义.

函数调用图:



4.110.2 变量说明

4.110.2.1 tmp_coef_a

```
int32_t tmp_coef_a = 0 [static]
```

在文件 [hal_lks32mc08x.tmp.c](#) 第 8 行定义.

4.110.2.2 tmp_coef_b

```
int32_t tmp_coef_b = 0 [static]
```

在文件 `hal_lks32mc08x.tmp.c` 第 9 行定义。

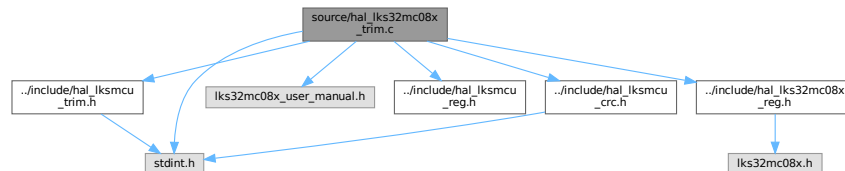
4.111 hal_lks32mc08x.tmp.c

[浏览该文件的文档.](#)

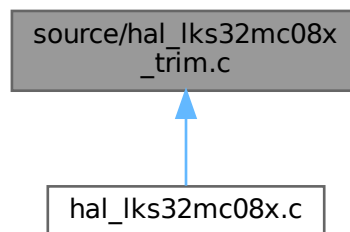
```
00001
00002 #include "../include/hal_lksmcu.tmp.h"
00003 #include "../include/hal_lksmcu.trim.h"
00004 #include "../hal_lks32mc08x.h"
00005 #include "lks32mc08x.user.manual.h"
00006 #include "../include/hal_lksmcu.reg.h"
00007 #include "../include/hal_lks32mc08x.reg.h"
00008 static int32_t tmp_coef_a = 0;
00009 static int32_t tmp_coef_b = 0;
00014 void HAL_TMP_Reset(HAL_TMP_x TMP_x)
00015 {
00016     HAL_TMP_Disable(TMP_x);
00017 }
00018
00023 void HAL_TMP_Enable(HAL_TMP_x TMP_x)
00024 {
00025     (void)TMP_x;
00026     HAL_REG_SYS_UNLOCK();
00027     HAL_REG_SET(SYS_AFE_REG5, SYS_AFE_REG5_TMP_PDN_BIT);
00028     HAL_REG_SYS_LOCK();
00029     tmp_coef_a = -HAL_TRIM_Read(0x00000398);
00030     tmp_coef_b = HAL_TRIM_Read(0x0000039C);
00031 }
00032
00037 void HAL_TMP_Disable(HAL_TMP_x TMP_x)
00038 {
00039     (void)TMP_x;
00040     HAL_REG_SYS_UNLOCK();
00041     HAL_REG_RESET(SYS_AFE_REG5, SYS_AFE_REG5_TMP_PDN_BIT);
00042     HAL_REG_SYS_LOCK();
00043 }
00044
00052 int32_t HAL_TMP_GetTemperature(HAL_TMP_x TMP_x, HAL_ADC_x adc, HAL_ADC_SAMP_SEQUENCE_x sequence)
00053 {
00054     // 实现获取温度的代码,单位摄氏度
00055     int32_t adc_data;
00056     int32_t tmp;
00057     adc_data = HAL_ADC_GetSamplingValues(adc, sequence);
00058     tmp = HAL_TMP_Adc2Temperature(TMP_x, adc_data);
00059     return tmp; // 返回温度值
00060 }
00061
00069 int32_t HAL_TMP_GetTemperature01(HAL_TMP_x TMP_x, HAL_ADC_x adc, HAL_ADC_SAMP_SEQUENCE_x sequence)
00070 {
00071     // 实现获取温度的代码,单位0.1摄氏度
00072     int32_t adc_data;
00073     int32_t tmp;
00074     adc_data = HAL_ADC_GetSamplingValues(adc, sequence);
00075     tmp = HAL_TMP_Adc2Temperature01(TMP_x, adc_data);
00076     return tmp; // 返回温度值
00077 }
00078
00085 int32_t HAL_TMP_Adc2Temperature(HAL_TMP_x TMP_x, int32_t adc_value)
00086 {
00087     return HAL_TMP_Adc2Temperature01(TMP_x, adc_value) / 10;
00088 }
00089
00096 int32_t HAL_TMP_Adc2Temperature01(HAL_TMP_x TMP_x, int32_t adc_value)
00097 {
00098     int32_t tmp;
00099     (void)TMP_x;
00100     tmp = (int32_t)tmp_coef_a * (adc_value >> 4) / 1000 + (int32_t)tmp_coef_b;
00101     return tmp; // 返回温度值
00102 }
```


4.112 source/hal_lks32mc08x_trim.c 文件参考

```
#include "../include/hal_lksmcu_trim.h"
#include "stdint.h"
#include "lks32mc08x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc08x_reg.h"
#include "../include/hal_lksmcu_crc.h"
hal_lks32mc08x_trim.c 的引用(Include)关系图:
```



此图展示该文件直接或间接的被哪些文件引用了:



函数

- uint32_t [HAL_TRIM_ReadMcu08](#) (uint32_t)
- uint32_t [HAL_TRIM_Read](#) (uint32_t addr)
读取MCU03的校准参数
- uint32_t [HAL_TRIM_ReadId](#) (uint32_t idx)
读取芯片唯一ID
- uint32_t [HAL_TRIM_ReadIdCrc32](#) (void)
读取芯片唯一ID的CRC32值

4.112.1 函数说明

4.112.1.1 HAL_TRIM_Read()

```
uint32_t HAL_TRIM_Read (
    uint32_t addr )
```

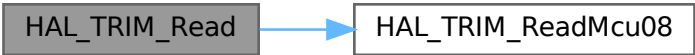
读取MCU03的校准参数

参数

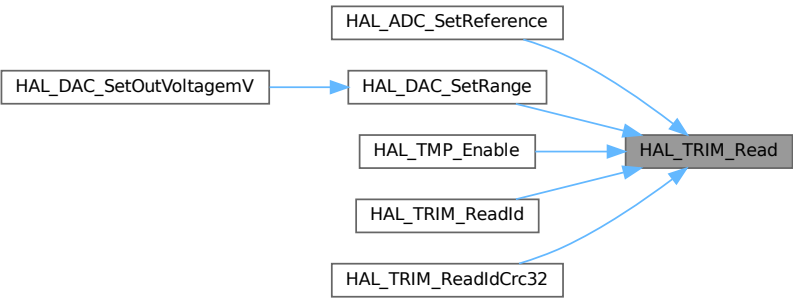
<i>addr</i>	校准参数的地址

在文件 [hal_lks32mc08x.trim.c](#) 第 64 行定义.

函数调用图:



这是这个函数的调用关系图:



4.112.1.2 HAL_TRIM_ReadId()

```
uint32_t HAL_TRIM_ReadId (
    uint32_t idx )
```

读取芯片唯一ID

参数

<i>idx</i>	0:ID0,1:ID1,2:ID2,3:ID3,4:ID4

返回

唯一ID

在文件 [hal_lks32mc08x.trim.c](#) 第 74 行定义.

函数调用图:



4.112.1.3 HAL_TRIM_ReadIdCrc32()

```
uint32_t HAL_TRIM_ReadIdCrc32 (
    void )
```

读取芯片唯一ID的CRC32值

返回

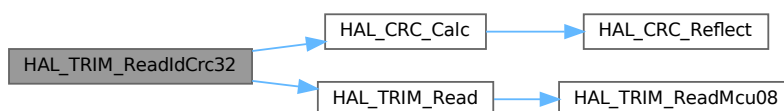
唯一ID的CRC32值

注解

相较于HAL_TRIM_ReadId，更方便存储和判断，但理论上存在重复的可能（1/0xffffffff）

在文件 [hal_lks32mc08x.trim.c](#) 第 84 行定义.

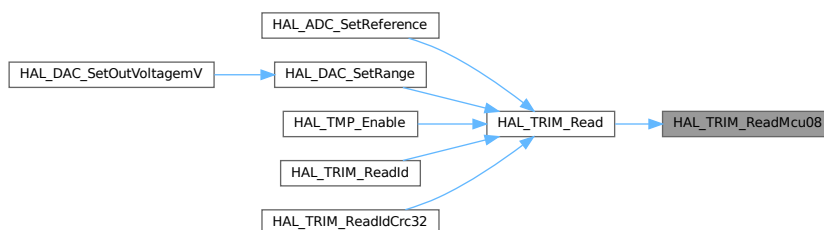
函数调用图:



4.112.1.4 HAL_TRIM_ReadMcu08()

```
uint32_t HAL_TRIM_ReadMcu08 (
    uint32_t )
```

这是这个函数的调用关系图:



4.113 hal_lks32mc08x_trim.c

[浏览该文件的文档。](#)

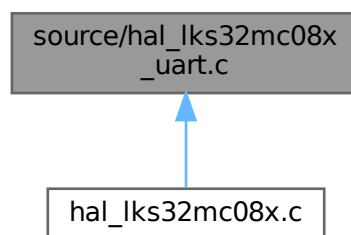
```
00001 #include "../include/hal_lksmcu_trim.h"
00002 #include "stdint.h"
00003 #include "lks32mc08x_user_manual.h"
00004 #include "../include/hal_lksmcu_reg.h"
00005 #include "../include/hal_lks32mc08x_reg.h"
00006 #include "../include/hal_lksmcu_crc.h"
00007 uint32_t HAL_TRIM_ReadMcu08(uint32_t);
00008 // 地址 内容
00009 // 0x00000310 ADC0_DC0 校准值
00010 // 0x00000314 ADC0_DC1 校准值
00011 // 0x00000318 ADC0_AMC0 校准值
00012 // 0x0000031C ADC0_AMC1 校准值
00013 // 0x00000320 ADC1_DC0 校准值
00014 // 0x00000324 ADC1_DC1 校准值
00015 // 0x00000328 ADC1_AMC0 校准值
00016 // 0x0000032C ADC1_AMC1 校准值
00017 // 0x00000330 DAC 选择 3.00V 档位, SYS_AFE_DAC_AMC 校准值(扩大512 倍结果)
00018 // 0x00000334 DAC 选择 1.20V 档位, SYS_AFE_DAC_AMC 校准值(扩大512 倍结果)
00019 // 0x00000338 DAC 选择 4.85V 档位, SYS_AFE_DAC_AMC 校准值(扩大512 倍结果)
00020 // 0x00000340 DAC 选择 3.00V 档位, SYS_AFE_DAC_DC 校准值
00021 // 0x00000344 DAC 选择 1.20V 档位, SYS_AFE_DAC_DC 校准值
00022 // 0x00000348 DAC 选择 4.85V 档位, SYS_AFE_DAC_DC 校准值
00023 // 0x00000350 OPA0, 200K 欧姆 VS 10.4K 欧姆, GAIN 校准值(扩大1000 倍结果), R0为0欧姆
00024 // 0x00000354 OPA0, 190K 欧姆 VS 20.4K 欧姆, GAIN 校准值(扩大1000 倍结果), R0为0欧姆
00025 // 0x00000358 OPA0, 180K 欧姆 VS 30.4K 欧姆, GAIN 校准值(扩大1000 倍结果), R0为0欧姆
00026 // 0x0000035C OPA0, 170K 欧姆 VS 40.4K 欧姆, GAIN 校准值(扩大1000 倍结果), R0为0欧姆
00027 // 0x00000360 OPA1, 200K 欧姆 VS 10.4K 欧姆, GAIN 校准值(扩大1000 倍结果), R0为0欧姆
00028 // 0x00000364 OPA1, 190K 欧姆 VS 20.4K 欧姆, GAIN 校准值(扩大1000 倍结果), R0为0欧姆
00029 // 0x00000368 OPA1, 180K 欧姆 VS 30.4K 欧姆, GAIN 校准值(扩大1000 倍结果), R0为0欧姆
00030 // 0x0000036C OPA1, 170K 欧姆 VS 40.4K 欧姆, GAIN 校准值(扩大1000 倍结果), R0为0欧姆
00031 // 0x00000370 OPA2, 200K 欧姆 VS 10.4K 欧姆, GAIN 校准值(扩大1000 倍结果), R0为0欧姆
00032 // 0x00000374 OPA2, 190K 欧姆 VS 20.4K 欧姆, GAIN 校准值(扩大1000 倍结果), R0为0欧姆
00033 // 0x00000378 OPA2, 180K 欧姆 VS 30.4K 欧姆, GAIN 校准值(扩大1000 倍结果), R0为0欧姆
00034 // 0x0000037C OPA2, 170K 欧姆 VS 40.4K 欧姆, GAIN 校准值(扩大1000 倍结果), R0为0欧姆
00035 // 0x00000380 OPA3, 200K 欧姆 VS 10.4K 欧姆, GAIN 校准值(扩大1000 倍结果), R0为0欧姆
00036 // 0x00000384 OPA3, 190K 欧姆 VS 20.4K 欧姆, GAIN 校准值(扩大1000 倍结果), R0为0欧姆
00037 // 0x00000388 OPA3, 180K 欧姆 VS 30.4K 欧姆, GAIN 校准值(扩大1000 倍结果), R0为0欧姆
00038 // 0x0000038C OPA3, 170K 欧姆 VS 40.4K 欧姆, GAIN 校准值(扩大1000 倍结果), R0为0欧姆
00039 // 0x00000398 温度传感器, 斜率校准值
00040 // 0x0000039C 温度传感器, 偏置校准值
00041 // 0x000003B0 高16-bit存放OPA1共模电压, 低16-bit存放OPA0共模电压(注意, 均是放大10000倍存入)
00042 // 0x000003B4 高16-bit存放OPA3共模电压, 低16-bit存放OPA2共模电压(注意, 均是放大10000倍存入)
00043 // 0x000002C0 OPA0, 200K 欧姆:10.4K 欧姆。高16位为R2实际电阻值, 低16位为R1实际电阻值(扩大100 倍结果)
00044 // 0x000002C4 OPA0, 190K 欧姆:20.4K 欧姆。高16位为R2实际电阻值, 低16位为R1实际电阻值(扩大100 倍结果)
00045 // 0x000002C8 OPA0, 180K 欧姆:30.4K 欧姆。高16位为R2实际电阻值, 低16位为R1实际电阻值(扩大100 倍结果)
00046 // 0x000002CC OPA0, 170K 欧姆:40.4K 欧姆。高16位为R2实际电阻值, 低16位为R1实际电阻值(扩大100 倍结果)
00047 // 0x000002D0 OPA1, 200K 欧姆:10.4K 欧姆。高16位为R2实际电阻值, 低16位为R1实际电阻值(扩大100 倍结果)
00048 // 0x000002D4 OPA1, 190K 欧姆:20.4K 欧姆。高16位为R2实际电阻值, 低16位为R1实际电阻值(扩大100 倍结果)
00049 // 0x000002D8 OPA1, 180K 欧姆:30.4K 欧姆。高16位为R2实际电阻值, 低16位为R1实际电阻值(扩大100 倍结果)
00050 // 0x000002DC OPA1, 170K 欧姆:40.4K 欧姆。高16位为R2实际电阻值, 低16位为R1实际电阻值(扩大100 倍结果)
00051 // 0x000002E0 OPA2, 200K 欧姆:10.4K 欧姆。高16位为R2实际电阻值, 低16位为R1实际电阻值(扩大100 倍结果)
00052 // 0x000002E4 OPA2, 190K 欧姆:20.4K 欧姆。高16位为R2实际电阻值, 低16位为R1实际电阻值(扩大100 倍结果)
00053 // 0x000002E8 OPA2, 180K 欧姆:30.4K 欧姆。高16位为R2实际电阻值, 低16位为R1实际电阻值(扩大100 倍结果)
00054 // 0x000002EC OPA2, 170K 欧姆:40.4K 欧姆。高16位为R2实际电阻值, 低16位为R1实际电阻值(扩大100 倍结果)
00055 // 0x000002F0 OPA3, 200K 欧姆:10.4K 欧姆。高16位为R2实际电阻值, 低16位为R1实际电阻值(扩大100 倍结果)
00056 // 0x000002F4 OPA3, 190K 欧姆:20.4K 欧姆。高16位为R2实际电阻值, 低16位为R1实际电阻值(扩大100 倍结果)
00057 // 0x000002F8 OPA3, 180K 欧姆:30.4K 欧姆。高16位为R2实际电阻值, 低16位为R1实际电阻值(扩大100 倍结果)
00058 // 0x000002FC OPA3, 170K 欧姆:40.4K 欧姆。高16位为R2实际电阻值, 低16位为R1实际电阻值(扩大100 倍结果)
00059
00064 uint32_t HAL_TRIM_Read(uint32_t addr)
00065 {
00066     return HAL_TRIM_ReadMcu08(addr);
00067 }
00068
00074 uint32_t HAL_TRIM_ReadId(uint32_t idx)
00075 {
00076     return HAL_TRIM_Read(0x0000084C + idx * 4);
00077 }
00078
00084 uint32_t HAL_TRIM_ReadIdCrc32(void)
00085 {
00086     uint32_t id[5];
00087     uint32_t i;
00088     for (i = 0; i < 5; i++)
00089     {
00090         id[i] = HAL_TRIM_Read(0x0000084C + i * 4);
00091     }
00092     return HAL_CRC_Calc(HAL_CRC_0, HAL_CRC_MODEL_CRC32, (uint8_t *)id, sizeof(id));
00093 }
```

4.114 source/hal_lks32mc08x_uart.c 文件参考

```
#include "../include/hal_lksmcu_clk.h"
#include "../include/hal_lksmcu_uart.h"
#include "../hal_lks32mc08x.h"
#include "stdint.h"
#include "lks32mc08x-user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc08x_reg.h"
hal_lks32mc08x_uart.c 的引用(Include)关系图:
```



此图展示该文件直接或间接的被哪些文件引用了:



函数

- void [HAL_UART_Reset](#) (HAL_UART_x UART_x)
- void [HAL_UART_Enable](#) (HAL_UART_x UART_x)
- void [HAL_UART_Disable](#) (HAL_UART_x UART_x)
- void [HAL_UART_SetBaud](#) (HAL_UART_x UART_x, uint32_t baud)
- void [HAL_UART_SetClockDiv](#) (HAL_UART_x UART_x, uint32_t div)
- uint32_t [HAL_UART_GetClockDiv](#) (HAL_UART_x UART_x)
- uint32_t [HAL_UART_GetClockFreq](#) (HAL_UART_x UART_x)
- uint32_t [HAL_UART_GetBaud](#) (HAL_UART_x UART_x)
- void [HAL_UART_TxData](#) (HAL_UART_x UART_x, uint8_t *data, uint32_t len)
- void [HAL_UART_SetMode](#) (HAL_UART_x uart_x, HAL_UART_MODE_x mode_x)
- void [HAL_UART_SetPolarity](#) (HAL_UART_x uart_x, HAL_UART_POL_x pol)
- 设置IO极性
- uint8_t [HAL_UART_ReadBuffer](#) (HAL_UART_x uart_x)
- void [HAL_UART_WriteBuffer](#) (HAL_UART_x uart_x, uint8_t buff)
- void [HAL_UART_SetGpioTx](#) (HAL_UART_x uart_x, HAL_GPIO_Px px)
- void [HAL_UART_SetGpioRx](#) (HAL_UART_x uart_x, HAL_GPIO_Px px)

变量

- static uint32_t [hal_baud_set](#) [2] = {0, 0}

4.114.1 函数说明

4.114.1.1 HAL_UART_Disable()

```
void HAL_UART_Disable (
    HAL_UART_x UART_x )
```

在文件 [hal_lks32mc08x_uart.c](#) 第 64 行定义.

4.114.1.2 HAL_UART_Enable()

```
void HAL_UART_Enable (
    HAL_UART_x UART_x )
```

在文件 [hal_lks32mc08x_uart.c](#) 第 39 行定义.

4.114.1.3 HAL_UART_GetBaud()

```
uint32_t HAL_UART_GetBaud (
    HAL_UART_x UART_x )
```

在文件 [hal_lks32mc08x_uart.c](#) 第 158 行定义.

函数调用图:

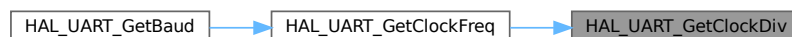


4.114.1.4 HAL_UART_GetClockDiv()

```
uint32_t HAL_UART_GetClockDiv (
    HAL_UART_x UART_x )
```

在文件 [hal_lks32mc08x_uart.c](#) 第 145 行定义.

这是这个函数的调用关系图:

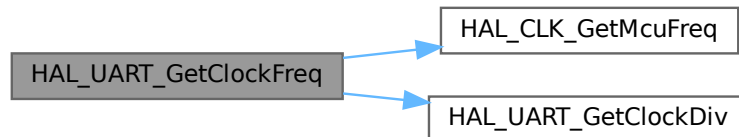


4.114.1.5 HAL_UART_GetClockFreq()

```
uint32_t HAL_UART_GetClockFreq (  
    HAL_UART_x UART_x )
```

在文件 [hal_lks32mc08x_uart.c](#) 第 152 行定义.

函数调用图:



这是这个函数的调用关系图:



4.114.1.6 HAL_UART_ReadBuffer()

```
uint8_t HAL_UART_ReadBuffer (  
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc08x_uart.c](#) 第 279 行定义.

4.114.1.7 HAL_UART_Reset()

```
void HAL_UART_Reset (  
    HAL_UART_x UART_x )
```

在文件 [hal_lks32mc08x_uart.c](#) 第 12 行定义.

4.114.1.8 HAL_UART_SetBaud()

```
void HAL_UART_SetBaud (
    HAL_UART_x UART_x,
    uint32_t baud )
```

在文件 [hal_lks32mc08x_uart.c](#) 第 80 行定义.

函数调用图:



4.114.1.9 HAL_UART_SetClockDiv()

```
void HAL_UART_SetClockDiv (
    HAL_UART_x UART_x,
    uint32_t div )
```

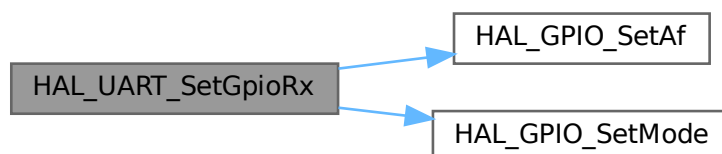
在文件 [hal_lks32mc08x_uart.c](#) 第 135 行定义.

4.114.1.10 HAL_UART_SetGpioRx()

```
void HAL_UART_SetGpioRx (
    HAL_UART_x uart_x,
    HAL_GPIO_Px px )
```

在文件 [hal_lks32mc08x_uart.c](#) 第 293 行定义.

函数调用图:

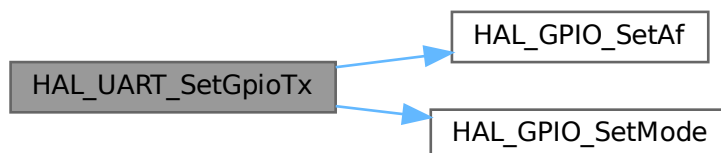


4.114.1.11 HAL_UART_SetGpioTx()

```
void HAL_UART_SetGpioTx (
    HAL_UART_<x> uart_<x>,
    HAL_GPIO_P<x> p<x> )
```

在文件 [hal_lks32mc08x_uart.c](#) 第 287 行定义.

函数调用图:



4.114.1.12 HAL_UART_SetMode()

```
void HAL_UART_SetMode (
    HAL_UART_<x> uart_<x>,
    HAL_UART_MODE_<x> mode_<x> )
```

- < 低位先发 1位停止位 无校验
- < 低位先发 2位停止位 无校验
- < 低位先发 1位停止位 奇校验
- < 低位先发 2位停止位 奇校验
- < 低位先发 1位停止位 偶校验
- < 低位先发 2位停止位 偶校验
- < 高位先发 1位停止位 无校验
- < 高位先发 2位停止位 无校验
- < 高位先发 1位停止位 奇校验
- < 高位先发 2位停止位 奇校验
- < 高位先发 1位停止位 偶校验
- < 高位先发 2位停止位 偶校验

在文件 [hal_lks32mc08x_uart.c](#) 第 178 行定义.

4.114.1.13 HAL_UART_SetPolarity()

```
void HAL_UART_SetPolarity (
    HAL_UART_<x> uart_<x>,
    HAL_UART_POL_<x> pol )
```

设置IO极性

参数

<i>uart_x</i>	UART模块编号
<i>pol</i>	极性配置

在文件 [hal_lks32mc08x_uart.c](#) 第 255 行定义.

4.114.1.14 HAL_UART_TxData()

```
void HAL_UART_TxData (
    HAL_UART_x UART_x,
    uint8_t * data,
    uint32_t len )
```

在文件 [hal_lks32mc08x_uart.c](#) 第 166 行定义.

4.114.1.15 HAL_UART_WriteBuffer()

```
void HAL_UART_WriteBuffer (
    HAL_UART_x uart_x,
    uint8_t buff )
```

在文件 [hal_lks32mc08x_uart.c](#) 第 283 行定义.

4.114.2 变量说明

4.114.2.1 hal_baud_set

```
uint32_t hal_baud_set[2] = {0, 0} [static]
```

在文件 [hal_lks32mc08x_uart.c](#) 第 9 行定义.

4.115 hal_lks32mc08x_uart.c

[浏览该文件的文档.](#)

```
00001 #include "../include/hal_lksmcu_clk.h"
00002 #include "../include/hal_lksmcu_uart.h"
00003 #include "../hal_lks32mc08x.h"
00004 #include "stdint.h"
00005 #include "lks32mc08x-user-manual.h"
00006 #include "../include/hal_lksmcu_reg.h"
00007 #include "../include/hal_lks32mc08x_reg.h"
00008
00009 static uint32_t hal_baud_set[2] = {0, 0};
00010 // 初始化
00011 // 模块复位
00012 void HAL_UART_Reset(HAL_UART_x UART_x)
00013 {
00014     switch (UART_x)
00015     {
00016         case HAL_UART_0:
00017             // 系统寄存器解锁
00018
```

```

00019         HAL_REG_SYS_UNLOCK();
00020         HAL_REG_SET(SYS_SFT_RST, SYS_SFT_RST_UART0_SFT_RST_BIT);
00021         HAL_REG_RESET(SYS_SFT_RST, SYS_SFT_RST_UART0_SFT_RST_BIT);
00022         // 系统寄存器上锁
00023         HAL_REG_SYS_LOCK();
00024         break;
00025     case HAL_UART_1:
00026
00027         // 系统寄存器解锁
00028         HAL_REG_SYS_UNLOCK();
00029         HAL_REG_SET(SYS_SFT_RST, SYS_SFT_RST_UART1_SFT_RST_BIT);
00030         HAL_REG_RESET(SYS_SFT_RST, SYS_SFT_RST_UART1_SFT_RST_BIT);
00031         // 系统寄存器上锁
00032         HAL_REG_SYS_LOCK();
00033         break;
00034     default:
00035         break;
00036 }
00037 }
00038 // 模块打开
00039 void HAL_UART_Enable(HAL_UART_x UART_x)
00040 {
00041     switch (UART_x)
00042     {
00043     case HAL_UART_0:
00044
00045         // 系统寄存器解锁
00046         HAL_REG_SYS_UNLOCK();
00047         HAL_REG_SET(SYS_CLK_FEN, SYS_CLK_FEN_UART0_CLK_EN_BIT);
00048         // 系统寄存器上锁
00049         HAL_REG_SYS_LOCK();
00050         break;
00051     case HAL_UART_1:
00052
00053         // 系统寄存器解锁
00054         HAL_REG_SYS_UNLOCK();
00055         HAL_REG_SET(SYS_CLK_FEN, SYS_CLK_FEN_UART1_CLK_EN_BIT);
00056         // 系统寄存器上锁
00057         HAL_REG_SYS_LOCK();
00058         break;
00059     default:
00060         break;
00061     }
00062 }
00063 // 模块关闭
00064 void HAL_UART_Disable(HAL_UART_x UART_x)
00065 {
00066     switch (UART_x)
00067     {
00068     case HAL_UART_0:
00069
00070         // 系统寄存器解锁
00071         HAL_REG_SYS_UNLOCK();
00072         HAL_REG_RESET(SYS_CLK_FEN, SYS_CLK_FEN_UART1_CLK_EN_BIT);
00073         // 系统寄存器上锁
00074         HAL_REG_SYS_LOCK();
00075         break;
00076     default:
00077         break;
00078     }
00079 }
00080 void HAL_UART_SetBaud(HAL_UART_x UART_x, uint32_t baud)
00081 {
00082     uint32_t mclk = HAL_CLK_GetMcuFreq();
00083     uint32_t div0;
00084     uint32_t div1;
00085     uint32_t divs = 1;
00086     uint32_t divmax = (UARTx_DIVH_DIVH_MAX << 8) | UARTx_DIVL_DIVL_MAX;
00087     if (baud > mclk)
00088     {
00089         baud = mclk;
00090     }
00091     hal_baud_set[UART_x] = baud;
00092
00093     if (hal_baud_set[0] != 0)
00094     {
00095         div0 = mclk / hal_baud_set[0];
00096     }
00097     else
00098     {
00099         div0 = 0;
00100     }
00101     if (hal_baud_set[1] != 0)
00102     {
00103         div1 = mclk / hal_baud_set[1];
00104     }
00105     else

```

```

00106     {
00107         div1 = 0;
00108     }
00109
00110     while ((div0 > divmax) || (div1 > divmax))
00111     {
00112         // 为了方便计算, 这里只考虑1 2 4 8 16 32 ... 这样的分频系数
00113         if (divs < SYS_CLK_DIV2_DIV2_MAX + 1)
00114         {
00115             divs <= 1;
00116         }
00117         else
00118         {
00119             divs = SYS_CLK_DIV2_DIV2_MAX + 1;
00120         }
00121         div0 >= 1;
00122         div1 >= 1;
00123     }
00124     divs = divs - 1;
00125     HAL_REG_SYS_UNLOCK();
00126     HAL_REG_WRITE_BIT_MASK_POS(SYS_CLK_DIV2, SYS_CLK_DIV2_DIV2, divs);
00127     HAL_REG_SYS_LOCK();
00128     HAL_REG_WRITE_BIT_MASK_POS(HAL_UART[0]->DIVH, UARTx_DIVH_DIVH, div0 >> 8);
00129     HAL_REG_WRITE_BIT_MASK_POS(HAL_UART[0]->DIVL, UARTx_DIVL_DIVL, div0 & 0xff);
00130     HAL_REG_WRITE_BIT_MASK_POS(HAL_UART[1]->DIVH, UARTx_DIVH_DIVH, div1 >> 8);
00131     HAL_REG_WRITE_BIT_MASK_POS(HAL_UART[1]->DIVL, UARTx_DIVL_DIVL, div1 & 0xff);
00132 }
00133
00134 // 设置UART全局工作时钟分频 (UART0/1共享)
00135 void HAL_UART_SetClockDiv(HAL_UART_x UART_x, uint32_t div)
00136 {
00137     (void)UART_x;
00138     if (div < 1) div = 1;
00139     HAL_REG_SYS_UNLOCK();
00140     HAL_REG_WRITE_BIT_MASK_POS(SYS_CLK_DIV2, SYS_CLK_DIV2_DIV2, div - 1);
00141     HAL_REG_SYS_LOCK();
00142 }
00143
00144 // 获取UART工作时钟分频实际除数
00145 uint32_t HAL_UART_GetClockDiv(HAL_UART_x UART_x)
00146 {
00147     (void)UART_x;
00148     return HAL_REG_READ_BIT_MASK_POS(SYS_CLK_DIV2, SYS_CLK_DIV2_DIV2) + 1;
00149 }
00150
00151 // 获取UART工作时钟频率
00152 uint32_t HAL_UART_GetClockFreq(HAL_UART_x UART_x)
00153 {
00154     return HAL_CLK_GetMcuFreq() / HAL_UART_GetClockDiv(UART_x);
00155 }
00156
00157 // 获取当前波特率
00158 uint32_t HAL_UART_GetBaud(HAL_UART_x UART_x)
00159 {
00160     uint32_t clk = HAL_UART_GetClockFreq(UART_x);
00161     uint32_t divh = HAL_REG_READ_BIT_MASK_POS(HAL_UART[UART_x]->DIVH, UARTx_DIVH_DIVH);
00162     uint32_t divl = HAL_REG_READ_BIT_MASK_POS(HAL_UART[UART_x]->DIVL, UARTx_DIVL_DIVL);
00163     return clk / ((divh << 8 | divl) + 1);
00164 }
00165
00166 void HAL_UART_TxData(HAL_UART_x UART_x, uint8_t *data, uint32_t len)
00167 {
00168     for (; len > 0; len--)
00169     {
00170         HAL_REG_WRITE(HAL_UART[UART_x]->BUFF, *data);
00171         data++;
00172         while (HAL_REG_READ_BIT(HAL_UART[UART_x]->STT, UARTx_STT_TX_BUF_EMPTY_BIT) == 0)
00173         {
00174         }
00175     }
00176 }
00177
00178 void HAL_UART_SetMode(HAL_UART_x uart_x, HAL_UART_MODE_x mode_x)
00179 {
00180     uint32_t mask = (UARTx_CTRL_CK_EN_BIT |
00181                     UARTx_CTRL_CK_TYPE_BIT |
00182                     UARTx_CTRL_BIT_ORDER_BIT |
00183                     UARTx_CTRL_STOP_LEN_BIT |
00184                     UARTx_CTRL_DAT_LEN_BIT);
00185
00186     switch (mode_x)
00187     {
00188     case HAL_UART_MODE_LSB_STOP1_NONE:
00189     {
00190         HAL_REG_WRITE_BIT(HAL_UART[uart_x]->CTRL, mask, 0);
00191         break;
00192     }
00193     }

```

```

00193         case HAL_UART_MODE_LSB_STOP2_NONE:
00194         {
00195             HAL_REG_WRITE_BIT(HAL_UART[uart_x]->CTRL, mask, UARTx_CTRL_STOP_LEN_BIT);
00196             break;
00197         }
00198         case HAL_UART_MODE_LSB_STOP1_ODD:
00199         {
00200             HAL_REG_WRITE_BIT(HAL_UART[uart_x]->CTRL, mask, UARTx_CTRL_CK_EN_BIT | UARTx_CTRL_CK_TYPE_BIT |
UARTx_CTRL_DAT_LEN_BIT);
00201             break;
00202         }
00203         case HAL_UART_MODE_LSB_STOP2_ODD:
00204         {
00205             HAL_REG_WRITE_BIT(HAL_UART[uart_x]->CTRL, mask, UARTx_CTRL_CK_EN_BIT | UARTx_CTRL_CK_TYPE_BIT |
UARTx_CTRL_STOP_LEN_BIT | UARTx_CTRL_DAT_LEN_BIT);
00206             break;
00207         }
00208         case HAL_UART_MODE_LSB_STOP1_EVEN:
00209         {
00210             HAL_REG_WRITE_BIT(HAL_UART[uart_x]->CTRL, mask, UARTx_CTRL_CK_EN_BIT | UARTx_CTRL_DAT_LEN_BIT);
00211             break;
00212         }
00213         case HAL_UART_MODE_LSB_STOP2_EVEN:
00214         {
00215             HAL_REG_WRITE_BIT(HAL_UART[uart_x]->CTRL, mask, UARTx_CTRL_CK_EN_BIT | UARTx_CTRL_STOP_LEN_BIT
| UARTx_CTRL_DAT_LEN_BIT);
00216             break;
00217         }
00218         case HAL_UART_MODE_MSB_STOP1_NONE:
00219         {
00220             HAL_REG_WRITE_BIT(HAL_UART[uart_x]->CTRL, mask, UARTx_CTRL_BIT_ORDER_BIT);
00221             break;
00222         }
00223         case HAL_UART_MODE_MSB_STOP2_NONE:
00224         {
00225             HAL_REG_WRITE_BIT(HAL_UART[uart_x]->CTRL, mask, UARTx_CTRL_BIT_ORDER_BIT |
UARTx_CTRL_STOP_LEN_BIT);
00226             break;
00227         }
00228         case HAL_UART_MODE_MSB_STOP1_ODD:
00229         {
00230             HAL_REG_WRITE_BIT(HAL_UART[uart_x]->CTRL, mask, UARTx_CTRL_CK_EN_BIT | UARTx_CTRL_CK_TYPE_BIT |
UARTx_CTRL_BIT_ORDER_BIT | UARTx_CTRL_DAT_LEN_BIT);
00231             break;
00232         }
00233         case HAL_UART_MODE_MSB_STOP2_ODD:
00234         {
00235             HAL_REG_WRITE_BIT(HAL_UART[uart_x]->CTRL, mask, UARTx_CTRL_CK_EN_BIT | UARTx_CTRL_CK_TYPE_BIT |
UARTx_CTRL_BIT_ORDER_BIT | UARTx_CTRL_STOP_LEN_BIT | UARTx_CTRL_DAT_LEN_BIT);
00236             break;
00237         }
00238         case HAL_UART_MODE_MSB_STOP1_EVEN:
00239         {
00240             HAL_REG_WRITE_BIT(HAL_UART[uart_x]->CTRL, mask, UARTx_CTRL_CK_EN_BIT | UARTx_CTRL_BIT_ORDER_BIT
| UARTx_CTRL_DAT_LEN_BIT);
00241             break;
00242         }
00243         case HAL_UART_MODE_MSB_STOP2_EVEN:
00244         {
00245             HAL_REG_WRITE_BIT(HAL_UART[uart_x]->CTRL, mask, UARTx_CTRL_CK_EN_BIT | UARTx_CTRL_BIT_ORDER_BIT
| UARTx_CTRL_STOP_LEN_BIT | UARTx_CTRL_DAT_LEN_BIT);
00246             break;
00247         }
00248     }
00249 }
00255 void HAL_UART_SetPolarity(HAL_UART_x uart_x, HAL_UART_POL_x pol)
00256 {
00257     switch (pol)
00258     {
00259         case HAL_UART_POL_RX_POS_TX_POS:
00260             HAL_REG_RESET(HAL_UART[uart_x]->IOC, UARTx_INV_RXD_INV_BIT);
00261             HAL_REG_RESET(HAL_UART[uart_x]->IOC, UARTx_INV_TXD_INV_BIT);
00262             break;
00263         case HAL_UART_POL_RX_POS_TX_NEG:
00264             HAL_REG_RESET(HAL_UART[uart_x]->IOC, UARTx_INV_RXD_INV_BIT);
00265             HAL_REG_SET(HAL_UART[uart_x]->IOC, UARTx_INV_TXD_INV_BIT);
00266             break;
00267         case HAL_UART_POL_RX_NEG_TX_POS:
00268             HAL_REG_SET(HAL_UART[uart_x]->IOC, UARTx_INV_RXD_INV_BIT);
00269             HAL_REG_RESET(HAL_UART[uart_x]->IOC, UARTx_INV_TXD_INV_BIT);
00270             break;
00271         case HAL_UART_POL_RX_NEG_TX_NEG:
00272             HAL_REG_SET(HAL_UART[uart_x]->IOC, UARTx_INV_RXD_INV_BIT);
00273             HAL_REG_SET(HAL_UART[uart_x]->IOC, UARTx_INV_TXD_INV_BIT);
00274             break;
00275         default:
00276             break;

```

```

00277     }
00278 }
00279 uint8_t HAL_UART_ReadBuffer(HAL_UART_x uart_x)
00280 {
00281     return HAL_REG_READ(HAL_UART[uart_x]->BUFF);
00282 }
00283 void HAL_UART_WriteBuffer(HAL_UART_x uart_x, uint8_t buff)
00284 {
00285     HAL_REG_WRITE(HAL_UART[uart_x]->BUFF, buff);
00286 }
00287 void HAL_UART_SetGpioTx(HAL_UART_x uart_x, HAL_GPIO_Px px)
00288 {
00289     (void)uart_x;
00290     HAL_GPIO_SetAf(px, HAL_GPIO_AF_UART);
00291     HAL_GPIO_SetMode(px, HAL_GPIO_MODE_OUT);
00292 }
00293 void HAL_UART_SetGpioRx(HAL_UART_x uart_x, HAL_GPIO_Px px)
00294 {
00295     (void)uart_x;
00296     HAL_GPIO_SetAf(px, HAL_GPIO_AF_UART);
00297     HAL_GPIO_SetMode(px, HAL_GPIO_MODE_IN);
00298 }

```

4.116 source/hal_lks32mc08x_wake.c 文件参考

```

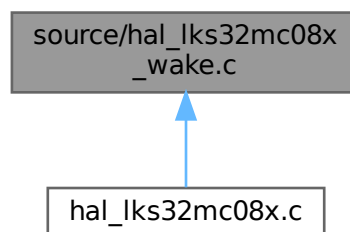
#include "../include/hal_lksmcu_iwdg.h"
#include "../include/hal_lksmcu_wake.h"
#include "../hal_lks32mc08x.h"
#include "lks32mc08x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc08x_reg.h"

```

hal_lks32mc08x_wake.c 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



函数

- void HAL_WAKE_DeepSleepIoWake (HAL_WAKE_UP_IO_x wake_up_io)

- 进入深度休眠,并且使用IO唤醒
- void HAL_WAKE_DeepSleepTimeWakeUp (uint32_t ms)

进入深度休眠,并且使用定时唤醒
 - void HAL_WAKE_DeepSleepIoTimeWakeUp (HAL_WAKE_UP_IO_x wake_up_io, uint32_t ms)

进入深度休眠,并且使用IO唤醒和定时唤醒

4.116.1 函数说明

4.116.1.1 HAL_WAKE_DeepSleepIoTimeWakeUp()

```
void HAL_WAKE_DeepSleepIoTimeWakeUp (  
    HAL_WAKE_UP_IO_x wake_up_io,  
    uint32_t ms )
```

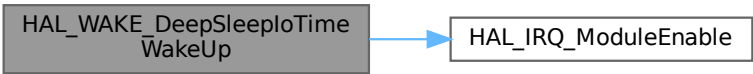
进入深度休眠,并且使用IO唤醒和定时唤醒

参数

<i>wake_up_io</i>	指定唤醒IO引脚及其电平
<i>ms</i>	指定定时唤醒的时间,单位为毫秒

在文件 [hal_lks32mc08x.wake.c](#) 第 155 行定义.

函数调用图:



4.116.1.2 HAL_WAKE_DeepSleepIoWake()

```
void HAL_WAKE_DeepSleepIoWake (  
    HAL_WAKE_UP_IO_x wake_up_io )
```

进入深度休眠,并且使用IO唤醒

参数

<i>wake_up_io</i>	指定唤醒IO引脚及其电平

在文件 [hal_lks32mc08x.wake.c](#) 第 12 行定义.

函数调用图:



4.116.1.3 HAL_WAKE_DeepSleepTimeWakeUp()

```
void HAL_WAKE_DeepSleepTimeWakeUp (
    uint32_t ms )
```

进入深度休眠,并且使用定时唤醒

参数

ms	指定定时唤醒的时间,单位为毫秒

在文件 [hal_lks32mc08x_wake.c](#) 第 62 行定义.

函数调用图:



4.117 hal_lks32mc08x_wake.c

[浏览该文件的文档.](#)

```
00001 #include "../include/hal_lksmcu_iwdg.h"
00002 #include "../include/hal_lksmcu_wake.h"
00003 #include "../hal_lks32mc08x.h"
00004 #include "lks32mc08x_user_manual.h"
00005 #include "../include/hal_lksmcu_reg.h"
00006 #include "../include/hal_lks32mc08x_reg.h"
00007
00012 void HAL_WAKE_DeepSleepIoWake (HAL_WAKE_UP_IO_x wake_up_io)
00013 {
00014     uint32_t t_cnt;
00015     uint32_t wake_irq_state = 0;
00016     uint32_t wake_sys_clk_cfg = 0;
00017     uint32_t wake_sys_clk_fen = 0;
00018     uint32_t wake_sys_afe_reg5 = 0;
00019     __disable_irq(); // 关闭所有中断
00020
00021     wake_irq_state = HAL_REG_READ (NVIC->ISER[0]); // 保存中断状态
00022     wake_sys_clk_cfg = HAL_REG_READ (SYS_CLK_CFG); // 保存时钟配置
00023     wake_sys_clk_fen = HAL_REG_READ (SYS_CLK_FEN); // 保存时钟使能
00024     wake_sys_afe_reg5 = HAL_REG_READ (SYS_AFE_REG5); // 保存模拟外设使能
```



```

00025
00026 HAL_REG_WRITE(GPIO_WAKE_POL, (wake_up_io >> 8) ? (wake_up_io & 0xFF) : 0); // 设置IO唤醒电平
00027 HAL_REG_WRITE(GPIO_WAKE_EN, wake_up_io & 0xFF); // 设置IO唤醒电平
00028
00029 HAL_REG_WRITE(NVIC->ICER[0], 0xFFFFFFFF); // 关闭所有中断
00030 HAL_REG_WRITE(NVIC->ICPR[0], 0xFFFFFFFF); // 清除所有中断标志
00031 HAL_IRQ_ModuleEnable(HAL_IRQ_MODULE_WAKE0, 0);
00032
00033 __enable_irq(); // 使能全局中断,此时仅唤醒中断使能
00034
00035 HAL_REG_SYS_UNLOCK();
00036 HAL_REG_WRITE(SYS_CLK_CFG, 0); // 关闭pll&时钟
00037 for (t_cnt = 0; t_cnt < 4000; t_cnt++)
00038 {
00039     __NOP();
00040 }
00041 HAL_REG_WRITE(SYS_CLK_FEN, 0); // 关闭所有数字外设时钟
00042 HAL_REG_WRITE(SYS_AFE_REG5, SYS_AFE_REG5_RCHPD_BIT | SYS_AFE_REG5_BGPPD_BIT); // 关闭所有模拟外设
00043 HAL_REG_WRITE(SYS_CLK_SLP, 0xDEAD); // 设置休眠模式为深度休眠
00044 __WFI(); // 进入深度休眠
00045
00046 HAL_REG_SYS_UNLOCK();
00047 HAL_REG_WRITE(SYS_AFE_REG5, wake_sys_afe_reg5); // 还原模拟外设使能
00048 for (t_cnt = 0; t_cnt < 400; t_cnt++)
00049 {
00050     __NOP();
00051 }
00052 HAL_REG_WRITE(SYS_CLK_CFG, wake_sys_clk_cfg); // 还原时钟配置
00053 HAL_REG_WRITE(SYS_CLK_FEN, wake_sys_clk_fen); // 还原时钟使能
00054 HAL_REG_WRITE(NVIC->ISER[0], wake_irq_state); // 恢复中断使能
00055 HAL_REG_SYS_LOCK();
00056 }
00057
00062 void HAL_WAKE_DeepSleepTimeWakeUp(uint32_t ms)
00063 {
00064     uint32_t t_cnt;
00065     uint32_t wake_irq_state = 0;
00066     uint32_t wake_sys_clk_cfg = 0;
00067     uint32_t wake_sys_clk_fen = 0;
00068     uint32_t wake_sys_afe_reg5 = 0;
00069     __disable_irq(); // 关闭所有中断
00070
00071     wake_irq_state = HAL_REG_READ(NVIC->ISER[0]); // 保存中断状态
00072     wake_sys_clk_cfg = HAL_REG_READ(SYS_CLK_CFG); // 保存时钟配置
00073     wake_sys_clk_fen = HAL_REG_READ(SYS_CLK_FEN); // 保存时钟使能
00074     wake_sys_afe_reg5 = HAL_REG_READ(SYS_AFE_REG5); // 保存模拟外设使能
00075
00076     HAL_REG_WRITE(GPIO_WAKE_POL, 0); // 设置IO唤醒电平
00077     HAL_REG_WRITE(GPIO_WAKE_EN, 0); // 设置IO唤醒引脚
00078
00079     HAL_REG_SYS_UNLOCK();
00080     // 0:0.25S
00081     // 1:0.5S
00082     // 2:1S
00083     // 3:2S
00084     // 4:4S
00085     // 5:8S
00086     // 6:16S
00087     // 7:32S
00088     if (ms < 250)
00089     {
00090         HAL_REG_WRITE_BIT_MASK_POS(SYS_RST_CFG, SYS_RST_CFG_WK_INTV, 0);
00091     }
00092     else if (ms < 500)
00093     {
00094         HAL_REG_WRITE_BIT_MASK_POS(SYS_RST_CFG, SYS_RST_CFG_WK_INTV, 1);
00095     }
00096     else if (ms < 1000)
00097     {
00098         HAL_REG_WRITE_BIT_MASK_POS(SYS_RST_CFG, SYS_RST_CFG_WK_INTV, 2);
00099     }
00100     else if (ms < 2000)
00101     {
00102         HAL_REG_WRITE_BIT_MASK_POS(SYS_RST_CFG, SYS_RST_CFG_WK_INTV, 3);
00103     }
00104     else if (ms < 4000)
00105     {
00106         HAL_REG_WRITE_BIT_MASK_POS(SYS_RST_CFG, SYS_RST_CFG_WK_INTV, 4);
00107     }
00108     else if (ms < 8000)
00109     {
00110         HAL_REG_WRITE_BIT_MASK_POS(SYS_RST_CFG, SYS_RST_CFG_WK_INTV, 5);
00111     }
00112     else if (ms < 16000)
00113     {
00114         HAL_REG_WRITE_BIT_MASK_POS(SYS_RST_CFG, SYS_RST_CFG_WK_INTV, 6);
00115     }

```

```

00116     else
00117     {
00118         HAL_REG_WRITE_BIT_MASK_POS(SYS_RST_CFG, SYS_RST_CFG_WK_INTV, 7);
00119     }
00120     HAL_REG_SYS_UNLOCK();
00121
00122     HAL_REG_WRITE(NVIC->ICER[0], 0xFFFFFFFF); // 关闭所有中断
00123     HAL_REG_WRITE(NVIC->ICPR[0], 0xFFFFFFFF); // 清除所有中断标志
00124     HAL_IRQ_ModuleEnable(HAL_IRQ_MODULE_WAKE0, 0);
00125
00126     __enable_irq(); // 使能全局中断, 此时仅唤醒中断使能
00127
00128     HAL_REG_SYS_UNLOCK();
00129     HAL_REG_WRITE(SYS_CLK_CFG, 0); // 关闭pll&时钟
00130     for (t_cnt = 0; t_cnt < 4000; t_cnt++)
00131     {
00132         __NOP();
00133     }
00134     HAL_REG_WRITE(SYS_CLK_FEN, 0);
00135     HAL_REG_WRITE(SYS_AFE_REG5, SYS_AFE_REG5_RCHPD_BIT | SYS_AFE_REG5_BGPPD_BIT); // 关闭所有数字外设时钟 // 关闭所有模拟外设 // 进入深度休眠
00136     __WFI();
00137
00138     HAL_REG_SYS_UNLOCK();
00139     HAL_REG_WRITE(SYS_AFE_REG5, wake_sys_afe_reg5); // 还原模拟外设使能
00140     for (t_cnt = 0; t_cnt < 400; t_cnt++)
00141     {
00142         __NOP();
00143     }
00144     HAL_REG_WRITE(SYS_CLK_CFG, wake_sys_clk_cfg); // 还原时钟配置
00145     HAL_REG_WRITE(SYS_CLK_FEN, wake_sys_clk_fen); // 还原时钟使能
00146     HAL_REG_WRITE(NVIC->ISER[0], wake_irq_state); // 恢复中断使能
00147     HAL_REG_SYS_LOCK();
00148 }
00149
00155 void HAL_WAKE_DeepSleepIoTimeWakeUp(HAL_WAKE_UP_IO_x wake_up_io, uint32_t ms)
00156 {
00157     uint32_t t_cnt;
00158     uint32_t wake_irq_state = 0;
00159     uint32_t wake_sys_clk_cfg = 0;
00160     uint32_t wake_sys_clk_fen = 0;
00161     uint32_t wake_sys_afe_reg5 = 0;
00162     __disable_irq(); // 关闭所有中断
00163
00164     wake_irq_state = HAL_REG_READ(NVIC->ISER[0]); // 保存中断状态
00165     wake_sys_clk_cfg = HAL_REG_READ(SYS_CLK_CFG); // 保存时钟配置
00166     wake_sys_clk_fen = HAL_REG_READ(SYS_CLK_FEN); // 保存时钟使能
00167     wake_sys_afe_reg5 = HAL_REG_READ(SYS_AFE_REG5); // 保存模拟外设使能
00168
00169     HAL_REG_WRITE(GPIO_WAKE_POL, wake_up_io >> 8); // 设置IO唤醒电平
00170     HAL_REG_WRITE(GPIO_WAKE_EN, wake_up_io & 0xFF); // 设置IO唤醒引脚
00171
00172     HAL_REG_SYS_UNLOCK();
00173     // 0:0.25S
00174     // 1:0.5S
00175     // 2:1S
00176     // 3:2S
00177     // 4:4S
00178     // 5:8S
00179     // 6:16S
00180     // 7:32S
00181     if (ms < 250)
00182     {
00183         HAL_REG_WRITE_BIT_MASK_POS(SYS_RST_CFG, SYS_RST_CFG_WK_INTV, 0);
00184     }
00185     else if (ms < 500)
00186     {
00187         HAL_REG_WRITE_BIT_MASK_POS(SYS_RST_CFG, SYS_RST_CFG_WK_INTV, 1);
00188     }
00189     else if (ms < 1000)
00190     {
00191         HAL_REG_WRITE_BIT_MASK_POS(SYS_RST_CFG, SYS_RST_CFG_WK_INTV, 2);
00192     }
00193     else if (ms < 2000)
00194     {
00195         HAL_REG_WRITE_BIT_MASK_POS(SYS_RST_CFG, SYS_RST_CFG_WK_INTV, 3);
00196     }
00197     else if (ms < 4000)
00198     {
00199         HAL_REG_WRITE_BIT_MASK_POS(SYS_RST_CFG, SYS_RST_CFG_WK_INTV, 4);
00200     }
00201     else if (ms < 8000)
00202     {
00203         HAL_REG_WRITE_BIT_MASK_POS(SYS_RST_CFG, SYS_RST_CFG_WK_INTV, 5);
00204     }
00205     else if (ms < 16000)
00206     {
00207         HAL_REG_WRITE_BIT_MASK_POS(SYS_RST_CFG, SYS_RST_CFG_WK_INTV, 6);

```

```

00208     }
00209     else
00210     {
00211         HAL_REG_WRITE_BIT_MASK_POS(SYS_RST_CFG, SYS_RST_CFG_WK_INTV, 7);
00212     }
00213     HAL_REG_SYS_LOCK();
00214
00215     HAL_REG_WRITE(NVIC->ICER[0], 0xFFFFFFFF); // 关闭所有中断
00216     HAL_REG_WRITE(NVIC->ICPR[0], 0xFFFFFFFF); // 清除所有中断标志
00217     HAL_IRQ_ModuleEnable(HAL_IRQ_MODULE_WAKE0, 0);
00218
00219     __enable_irq(); // 使能全局中断,此时仅唤醒中断使能
00220
00221     HAL_REG_SYS_UNLOCK();
00222     HAL_REG_WRITE(SYS_CLK_CFG, 0); // 关闭pll&时钟
00223     for (t_cnt = 0; t_cnt < 4000; t_cnt++)
00224     {
00225         __NOP();
00226     }
00227     HAL_REG_WRITE(SYS_CLK_FEN, 0);
00228     HAL_REG_WRITE(SYS_AFE_REG5, SYS_AFE_REG5_RCHPD_BIT | SYS_AFE_REG5_BGPPD_BIT); // 关闭所有模拟外设 // 关闭所有数字外设时钟
00229     __WFI(); // 进入深度休眠
00230
00231     HAL_REG_SYS_UNLOCK();
00232     HAL_REG_WRITE(SYS_AFE_REG5, wake_sys_afe_reg5); // 还原模拟外设使能
00233     for (t_cnt = 0; t_cnt < 400; t_cnt++)
00234     {
00235         __NOP();
00236     }
00237     HAL_REG_WRITE(SYS_CLK_CFG, wake_sys_clk_cfg); // 还原时钟配置
00238     HAL_REG_WRITE(SYS_CLK_FEN, wake_sys_clk_fen); // 还原时钟使能
00239     HAL_REG_WRITE(NVIC->ISER[0], wake_irq_state); // 恢复中断使能
00240     HAL_REG_SYS_LOCK();
00241 }

```


Index

__HAL_LKS32MC08x_C__
 hal_lks32mc08x.c, [56](#)

ADDR
 HAL_FLASH_TypeDef, [26](#)
 HAL_I2C_TypeDef, [32](#)

ADR
 HAL_UART_TypeDef, [52](#)

AFE_CMP
 HAL_SYS_TypeDef, [43](#)

AFE_DAC
 HAL_SYS_TypeDef, [43](#)

AFE_DAC_AMC
 HAL_SYS_TypeDef, [43](#)

AFE_DAC_DC
 HAL_SYS_TypeDef, [43](#)

AFE_REG0
 HAL_SYS_TypeDef, [43](#)

AFE_REG1
 HAL_SYS_TypeDef, [44](#)

AFE_REG2
 HAL_SYS_TypeDef, [44](#)

AFE_REG3
 HAL_SYS_TypeDef, [44](#)

AFE_REG4
 HAL_SYS_TypeDef, [44](#)

AFE_REG5
 HAL_SYS_TypeDef, [44](#)

AFE_REG6
 HAL_SYS_TypeDef, [44](#)

AFE_REG7
 HAL_SYS_TypeDef, [45](#)

AFE_REG8
 HAL_SYS_TypeDef, [45](#)

AFE_REG9
 HAL_SYS_TypeDef, [45](#)

AFE_REGA
 HAL_SYS_TypeDef, [45](#)

AFE_REGB
 HAL_SYS_TypeDef, [45](#)

AFE_REGC
 HAL_SYS_TypeDef, [45](#)

ALC
 HAL_CAN_TypeDef, [14](#)

AMC_A0
 HAL_ADC_TypeDef, [9](#)

AMC_A1
 HAL_ADC_TypeDef, [9](#)

AMC_B0
 HAL_ADC_TypeDef, [9](#)

AMC_B1
 HAL_ADC_TypeDef, [9](#)

ATAN
 HAL_DSP_TypeDef, [23](#)

BAUD
 HAL_SPI_TypeDef, [40](#)

baud_cnt
 hal_lks32mc08x_can.c, [410](#)

baud_param
 hal_lks32mc08x_can.c, [410](#)

BCR
 HAL_I2C_TypeDef, [32](#)

BLCWIN
 CMP_TypeDef, [5](#)
 HAL_CMP_TypeDef, [19](#)

BTR0
 HAL_CAN_TypeDef, [14](#)

BTR1
 HAL_CAN_TypeDef, [14](#)

BUFF
 HAL_UART_TypeDef, [52](#)

CAN_Write_TBUF
 hal_lks32mc08x_can.c, [399](#)

CCR
 HAL_DMA_TypeDef, [22](#)

CFG
 CMP_TypeDef, [5](#)
 HAL_ADC_TypeDef, [9](#)
 HAL_CMP_TypeDef, [19](#)
 HAL_FLASH_TypeDef, [26](#)
 HAL_HALL_TypeDef, [30](#)
 HAL_I2C_TypeDef, [32](#)
 HAL_QEP_TypeDef, [39](#)
 HAL_SPI_TypeDef, [40](#)
 HAL_TIMER_TypeDef, [50](#)

CHN
 HAL_ADC_TypeDef, [9](#)

CHNT0
 HAL_ADC_TypeDef, [9](#)

CHNT1
 HAL_ADC_TypeDef, [10](#)

CLK_CFG
 HAL_SYS_TypeDef, [46](#)

CLK_DIV0
 HAL_SYS_TypeDef, [46](#)

CLK_DIV2
 HAL_SYS_TypeDef, [46](#)

CLK_FEN

HAL_SYS_TypeDef, 46
 CLK_SLP
 HAL_SYS_TypeDef, 46
 CLR_RST
 HAL_SYS_TypeDef, 46
 CMAR
 HAL_DMA_TypeDef, 22
 CMP_TypeDef, 5
 BLCWIN, 5
 CFG, 5
 IE, 6
 IF, 6
 TCLK, 6
 CMPT0
 HAL_TIMER_TypeDef, 50
 CMPT1
 HAL_TIMER_TypeDef, 50
 CMR
 HAL_CAN_TypeDef, 14
 CNT
 HAL_HALL_TypeDef, 30
 HAL_MCPWM_TypeDef, 34
 HAL_QEP_TypeDef, 39
 HAL_TIMER_TypeDef, 50
 COS
 HAL_DSP_TypeDef, 23
 CPAR
 HAL_DMA_TypeDef, 22
 CR
 HAL_CRC_TypeDef, 21
 CRC_Model_Params, 6
 initValue, 7
 polynomial, 7
 reflectIn, 7
 reflectOut, 7
 width, 7
 xorOut, 7
 CRC_Width
 hal_lks32mc08x_crc.c, 431
 CRC_WIDTH_16
 hal_lks32mc08x_crc.c, 432
 CRC_WIDTH_32
 hal_lks32mc08x_crc.c, 432
 CRC_WIDTH_8
 hal_lks32mc08x_crc.c, 432
 crcModels
 hal_lks32mc08x_crc.c, 434
 CTMS
 HAL_DMA_TypeDef, 22
 CTRL
 HAL_UART_TypeDef, 52
 DAT
 HAL_ADC_TypeDef, 10
 DAT0_TH
 HAL_ADC_TypeDef, 10
 DATA
 HAL_CMP_TypeDef, 19
 HAL_I2C_TypeDef, 32
 DC_A0
 HAL_ADC_TypeDef, 10
 DC_A1
 HAL_ADC_TypeDef, 10
 DC_B0
 HAL_ADC_TypeDef, 10
 DC_B1
 HAL_ADC_TypeDef, 11
 DIVH
 HAL_UART_TypeDef, 52
 DIVL
 HAL_UART_TypeDef, 52
 DR
 HAL_CRC_TypeDef, 21
 DTH00
 HAL_MCPWM_TypeDef, 34
 DTH01
 HAL_MCPWM_TypeDef, 35
 DTH10
 HAL_MCPWM_TypeDef, 35
 DTH11
 HAL_MCPWM_TypeDef, 35
 DTH20
 HAL_MCPWM_TypeDef, 35
 DTH21
 HAL_MCPWM_TypeDef, 35
 DTH30
 HAL_MCPWM_TypeDef, 35
 DTH31
 HAL_MCPWM_TypeDef, 35
 ECC
 HAL_CAN_TypeDef, 14
 EIE
 HAL_MCPWM_TypeDef, 35
 EIF
 HAL_MCPWM_TypeDef, 36
 ERASE
 HAL_FLASH_TypeDef, 26
 EVT
 HAL_TIMER_TypeDef, 51
 EWLR
 HAL_CAN_TypeDef, 14
 F3210
 HAL_GPIO_TypeDef, 28
 F7654
 HAL_GPIO_TypeDef, 28
 FAIL
 HAL_MCPWM_TypeDef, 36
 FBA98
 HAL_GPIO_TypeDef, 28
 FFEDC
 HAL_GPIO_TypeDef, 28
 FILT
 HAL_MCPWM_TypeDef, 36
 GAIN_0
 HAL_ADC_TypeDef, 11

GAIN_1
 HAL_ADC_TypeDef, 11
GetBaudParam
 hal_lks32mc08x_can.c, 399

HAL_ADC
 hal_lks32mc08x_reg.c, 680
 hal_lks32mc08x_reg.h, 62

HAL_ADC_0
 hal_lksmcu_adc.h, 73

HAL_ADC_1
 hal_lksmcu_adc.h, 74

HAL_ADC_2
 hal_lksmcu_adc.h, 74

HAL_ADC_3
 hal_lksmcu_adc.h, 74

HAL_ADC_CHANNEL_0
 hal_lksmcu_adc.h, 71

HAL_ADC_CHANNEL_1
 hal_lksmcu_adc.h, 71

HAL_ADC_CHANNEL_10
 hal_lksmcu_adc.h, 71

HAL_ADC_CHANNEL_11
 hal_lksmcu_adc.h, 71

HAL_ADC_CHANNEL_12
 hal_lksmcu_adc.h, 71

HAL_ADC_CHANNEL_13
 hal_lksmcu_adc.h, 71

HAL_ADC_CHANNEL_14
 hal_lksmcu_adc.h, 71

HAL_ADC_CHANNEL_15
 hal_lksmcu_adc.h, 71

HAL_ADC_CHANNEL_16
 hal_lksmcu_adc.h, 71

HAL_ADC_CHANNEL_17
 hal_lksmcu_adc.h, 71

HAL_ADC_CHANNEL_18
 hal_lksmcu_adc.h, 71

HAL_ADC_CHANNEL_19
 hal_lksmcu_adc.h, 71

HAL_ADC_CHANNEL_2
 hal_lksmcu_adc.h, 71

HAL_ADC_CHANNEL_3
 hal_lksmcu_adc.h, 71

HAL_ADC_CHANNEL_4
 hal_lksmcu_adc.h, 71

HAL_ADC_CHANNEL_5
 hal_lksmcu_adc.h, 71

HAL_ADC_CHANNEL_6
 hal_lksmcu_adc.h, 71

HAL_ADC_CHANNEL_7
 hal_lksmcu_adc.h, 71

HAL_ADC_CHANNEL_8
 hal_lksmcu_adc.h, 71

HAL_ADC_CHANNEL_9
 hal_lksmcu_adc.h, 71

HAL_ADC_CHANNEL_AGND
 hal_lksmcu_adc.h, 71

HAL_ADC_CHANNEL_OPA0
 hal_lksmcu_adc.h, 71

HAL_ADC_CHANNEL_OPA1
 hal_lksmcu_adc.h, 71

HAL_ADC_CHANNEL_OPA2
 hal_lksmcu_adc.h, 71

HAL_ADC_CHANNEL_OPA3
 hal_lksmcu_adc.h, 71

HAL_ADC_CHANNEL_TMP0
 hal_lksmcu_adc.h, 71

HAL_ADC_CHANNEL_x
 hal_lksmcu_adc.h, 71

HAL_ADC_Disable
 hal_lks32mc08x_adc.c, 383
 hal_lksmcu_adc.h, 74

HAL_ADC_Enable
 hal_lks32mc08x_adc.c, 383
 hal_lksmcu_adc.h, 74

HAL_ADC_EndSampling
 hal_lks32mc08x_adc.c, 383
 hal_lksmcu_adc.h, 74

HAL_ADC_GetRange
 hal_lks32mc08x_adc.c, 384
 hal_lksmcu_adc.h, 75

HAL_ADC_GetSamplingValues
 hal_lks32mc08x_adc.c, 384
 hal_lksmcu_adc.h, 75

HAL_ADC_GetValuesUnit
 hal_lks32mc08x_adc.c, 385
 hal_lksmcu_adc.h, 76

HAL_ADC_IrqHardErr
 hal_lks32mc08x_irqfun.c, 517
 hal_lksmcu_irqfun.h, 199

HAL_ADC_IrqHardErrClear
 hal_lks32mc08x_irqfun.c, 517
 hal_lksmcu_irqfun.h, 199

HAL_ADC_IrqHardErrDisable
 hal_lks32mc08x_irqfun.c, 517
 hal_lksmcu_irqfun.h, 199

HAL_ADC_IrqHardErrEnable
 hal_lks32mc08x_irqfun.c, 518
 hal_lksmcu_irqfun.h, 200

HAL_ADC_IrqHardErrFlg
 hal_lks32mc08x_irqfun.c, 518
 hal_lksmcu_irqfun.h, 200

HAL_ADC_IrqIdleSamp
 hal_lks32mc08x_irqfun.c, 518
 hal_lksmcu_irqfun.h, 200

HAL_ADC_IrqIdleSampClear
 hal_lks32mc08x_irqfun.c, 518
 hal_lksmcu_irqfun.h, 200

HAL_ADC_IrqIdleSampDisable
 hal_lks32mc08x_irqfun.c, 518
 hal_lksmcu_irqfun.h, 201

HAL_ADC_IrqIdleSampEnable
 hal_lks32mc08x_irqfun.c, 519
 hal_lksmcu_irqfun.h, 201

HAL_ADC_IrqIdleSampFlg
 hal_lks32mc08x_irqfun.c, 519

- hal_lksmcu_irqfun.h, 201
- HAL_ADC_IrqSamp1
 - hal_lks32mc08x_irqfun.c, 519
 - hal_lksmcu_irqfun.h, 201
- HAL_ADC_IrqSamp1Clear
 - hal_lks32mc08x_irqfun.c, 519
 - hal_lksmcu_irqfun.h, 201
- HAL_ADC_IrqSamp1Disable
 - hal_lks32mc08x_irqfun.c, 520
 - hal_lksmcu_irqfun.h, 202
- HAL_ADC_IrqSamp1Enable
 - hal_lks32mc08x_irqfun.c, 520
 - hal_lksmcu_irqfun.h, 202
- HAL_ADC_IrqSamp1Flg
 - hal_lks32mc08x_irqfun.c, 520
 - hal_lksmcu_irqfun.h, 202
- HAL_ADC_IrqSamp2
 - hal_lks32mc08x_irqfun.c, 521
 - hal_lksmcu_irqfun.h, 203
- HAL_ADC_IrqSamp2Clear
 - hal_lks32mc08x_irqfun.c, 521
 - hal_lksmcu_irqfun.h, 203
- HAL_ADC_IrqSamp2Disable
 - hal_lks32mc08x_irqfun.c, 521
 - hal_lksmcu_irqfun.h, 203
- HAL_ADC_IrqSamp2Enable
 - hal_lks32mc08x_irqfun.c, 521
 - hal_lksmcu_irqfun.h, 204
- HAL_ADC_IrqSamp2Flg
 - hal_lks32mc08x_irqfun.c, 522
 - hal_lksmcu_irqfun.h, 204
- HAL_ADC_IrqSamp3
 - hal_lks32mc08x_irqfun.c, 522
 - hal_lksmcu_irqfun.h, 204
- HAL_ADC_IrqSamp3Clear
 - hal_lks32mc08x_irqfun.c, 522
 - hal_lksmcu_irqfun.h, 205
- HAL_ADC_IrqSamp3Disable
 - hal_lks32mc08x_irqfun.c, 523
 - hal_lksmcu_irqfun.h, 205
- HAL_ADC_IrqSamp3Enable
 - hal_lks32mc08x_irqfun.c, 523
 - hal_lksmcu_irqfun.h, 205
- HAL_ADC_IrqSamp3Flg
 - hal_lks32mc08x_irqfun.c, 523
 - hal_lksmcu_irqfun.h, 205
- HAL_ADC_IrqSamp4
 - hal_lks32mc08x_irqfun.c, 524
 - hal_lksmcu_irqfun.h, 206
- HAL_ADC_IrqSamp4Clear
 - hal_lks32mc08x_irqfun.c, 524
 - hal_lksmcu_irqfun.h, 206
- HAL_ADC_IrqSamp4Disable
 - hal_lks32mc08x_irqfun.c, 524
 - hal_lksmcu_irqfun.h, 206
- HAL_ADC_IrqSamp4Enable
 - hal_lks32mc08x_irqfun.c, 525
 - hal_lksmcu_irqfun.h, 207
- HAL_ADC_IrqSamp4Flg
 - hal_lks32mc08x_irqfun.c, 525
 - hal_lksmcu_irqfun.h, 207
- HAL_ADC_IrqSoftErr
 - hal_lks32mc08x_irqfun.c, 525
 - hal_lksmcu_irqfun.h, 207
- HAL_ADC_IrqSoftErrClear
 - hal_lks32mc08x_irqfun.c, 526
 - hal_lksmcu_irqfun.h, 208
- HAL_ADC_IrqSoftErrDisable
 - hal_lks32mc08x_irqfun.c, 526
 - hal_lksmcu_irqfun.h, 208
- HAL_ADC_IrqSoftErrEnable
 - hal_lks32mc08x_irqfun.c, 526
 - hal_lksmcu_irqfun.h, 208
- HAL_ADC_IrqSoftErrFlg
 - hal_lks32mc08x_irqfun.c, 526
 - hal_lksmcu_irqfun.h, 209
- HAL_ADC_IrqWatchdog
 - hal_lks32mc08x_irqfun.c, 527
 - hal_lksmcu_irqfun.h, 209
- HAL_ADC_IrqWatchdog1
 - hal_lks32mc08x_irqfun.c, 527
 - hal_lksmcu_irqfun.h, 209
- HAL_ADC_IrqWatchdog1Clear
 - hal_lks32mc08x_irqfun.c, 527
 - hal_lksmcu_irqfun.h, 209
- HAL_ADC_IrqWatchdog1Disable
 - hal_lks32mc08x_irqfun.c, 527
 - hal_lksmcu_irqfun.h, 209
- HAL_ADC_IrqWatchdog1Enable
 - hal_lks32mc08x_irqfun.c, 528
 - hal_lksmcu_irqfun.h, 210
- HAL_ADC_IrqWatchdog1Flg
 - hal_lks32mc08x_irqfun.c, 528
 - hal_lksmcu_irqfun.h, 210
- HAL_ADC_IrqWatchdogClear
 - hal_lks32mc08x_irqfun.c, 528
 - hal_lksmcu_irqfun.h, 210
- HAL_ADC_IrqWatchdogDisable
 - hal_lks32mc08x_irqfun.c, 528
 - hal_lksmcu_irqfun.h, 210
- HAL_ADC_IrqWatchdogEnable
 - hal_lks32mc08x_irqfun.c, 528
 - hal_lksmcu_irqfun.h, 210
- HAL_ADC_IrqWatchdogFlg
 - hal_lks32mc08x_irqfun.c, 529
 - hal_lksmcu_irqfun.h, 211
- HAL_ADC_ManualSampling
 - hal_lks32mc08x_adc.c, 386
 - hal_lksmcu_adc.h, 77
- HAL_ADC_REFERENCE_EXTERNAL
 - hal_lksmcu_adc.h, 72
- HAL_ADC_REFERENCE_INTERNAL
 - hal_lksmcu_adc.h, 71
- HAL_ADC_REFERENCE_INTERNAL_2.4
 - hal_lksmcu_adc.h, 72
- HAL_ADC_REFERENCE_INTERNAL_3.6

hal_lksmcu_adc.h, [72](#)
 HAL_ADC_REFERENCE_x
 hal_lksmcu_adc.h, [71](#)
 HAL_ADC_Reset
 hal_lks32mc08x_adc.c, [386](#)
 hal_lksmcu_adc.h, [77](#)
 HAL_ADC_SAMP_NUMBER_0
 hal_lksmcu_adc.h, [72](#)
 HAL_ADC_SAMP_NUMBER_1
 hal_lksmcu_adc.h, [72](#)
 HAL_ADC_SAMP_NUMBER_10
 hal_lksmcu_adc.h, [72](#)
 HAL_ADC_SAMP_NUMBER_11
 hal_lksmcu_adc.h, [72](#)
 HAL_ADC_SAMP_NUMBER_12
 hal_lksmcu_adc.h, [72](#)
 HAL_ADC_SAMP_NUMBER_13
 hal_lksmcu_adc.h, [72](#)
 HAL_ADC_SAMP_NUMBER_14
 hal_lksmcu_adc.h, [72](#)
 HAL_ADC_SAMP_NUMBER_15
 hal_lksmcu_adc.h, [72](#)
 HAL_ADC_SAMP_NUMBER_2
 hal_lksmcu_adc.h, [72](#)
 HAL_ADC_SAMP_NUMBER_3
 hal_lksmcu_adc.h, [72](#)
 HAL_ADC_SAMP_NUMBER_4
 hal_lksmcu_adc.h, [72](#)
 HAL_ADC_SAMP_NUMBER_5
 hal_lksmcu_adc.h, [72](#)
 HAL_ADC_SAMP_NUMBER_6
 hal_lksmcu_adc.h, [72](#)
 HAL_ADC_SAMP_NUMBER_7
 hal_lksmcu_adc.h, [72](#)
 HAL_ADC_SAMP_NUMBER_8
 hal_lksmcu_adc.h, [72](#)
 HAL_ADC_SAMP_NUMBER_9
 hal_lksmcu_adc.h, [72](#)
 HAL_ADC_SAMP_NUMBER_MAX
 hal_lksmcu_adc.h, [72](#)
 HAL_ADC_SAMP_NUMBER_x
 hal_lksmcu_adc.h, [72](#)
 HAL_ADC_SAMP_SEQUENCE_0
 hal_lksmcu_adc.h, [72](#)
 HAL_ADC_SAMP_SEQUENCE_1
 hal_lksmcu_adc.h, [72](#)
 HAL_ADC_SAMP_SEQUENCE_10
 hal_lksmcu_adc.h, [73](#)
 HAL_ADC_SAMP_SEQUENCE_11
 hal_lksmcu_adc.h, [73](#)
 HAL_ADC_SAMP_SEQUENCE_12
 hal_lksmcu_adc.h, [73](#)
 HAL_ADC_SAMP_SEQUENCE_13
 hal_lksmcu_adc.h, [73](#)
 HAL_ADC_SAMP_SEQUENCE_14
 hal_lksmcu_adc.h, [73](#)
 HAL_ADC_SAMP_SEQUENCE_15
 hal_lksmcu_adc.h, [73](#)
 HAL_ADC_SAMP_SEQUENCE_16
 hal_lksmcu_adc.h, [73](#)
 HAL_ADC_SAMP_SEQUENCE_17
 hal_lksmcu_adc.h, [73](#)
 HAL_ADC_SAMP_SEQUENCE_18
 hal_lksmcu_adc.h, [73](#)
 HAL_ADC_SAMP_SEQUENCE_19
 hal_lksmcu_adc.h, [73](#)
 HAL_ADC_SAMP_SEQUENCE_2
 hal_lksmcu_adc.h, [73](#)
 HAL_ADC_SAMP_SEQUENCE_3
 hal_lksmcu_adc.h, [73](#)
 HAL_ADC_SAMP_SEQUENCE_4
 hal_lksmcu_adc.h, [73](#)
 HAL_ADC_SAMP_SEQUENCE_5
 hal_lksmcu_adc.h, [73](#)
 HAL_ADC_SAMP_SEQUENCE_6
 hal_lksmcu_adc.h, [73](#)
 HAL_ADC_SAMP_SEQUENCE_7
 hal_lksmcu_adc.h, [73](#)
 HAL_ADC_SAMP_SEQUENCE_8
 hal_lksmcu_adc.h, [73](#)
 HAL_ADC_SAMP_SEQUENCE_9
 hal_lksmcu_adc.h, [73](#)
 HAL_ADC_SAMP_SEQUENCE_x
 hal_lksmcu_adc.h, [72](#)
 HAL_ADC_SetGpioIn
 hal_lks32mc08x_adc.c, [387](#)
 hal_lksmcu_adc.h, [78](#)
 HAL_ADC_SetGpioTriggerOut
 hal_lks32mc08x_adc.c, [387](#)
 hal_lksmcu_adc.h, [78](#)
 HAL_ADC_SetReference
 hal_lks32mc08x_adc.c, [389](#)
 hal_lksmcu_adc.h, [79](#)
 HAL_ADC_SetSamplingChannel
 hal_lks32mc08x_adc.c, [389](#)
 hal_lksmcu_adc.h, [79](#)
 HAL_ADC_SetSamplingQuantity
 hal_lks32mc08x_adc.c, [390](#)
 hal_lksmcu_adc.h, [80](#)
 HAL_ADC_SetTriggerSignal
 hal_lks32mc08x_adc.c, [391](#)
 hal_lksmcu_adc.h, [81](#)
 HAL_ADC_SoftTrigger
 hal_lks32mc08x_adc.c, [392](#)
 hal_lksmcu_adc.h, [82](#)
 HAL_ADC_StateReset
 hal_lks32mc08x_adc.c, [392](#)
 hal_lksmcu_adc.h, [82](#)
 HAL_ADC_TRIG.STATE
 hal_lks32mc08x_adc.c, [393](#)
 HAL_ADC_TRIG.STATE_ADC0_EN
 hal_lks32mc08x_reg.h, [60](#)
 HAL_ADC_TRIG.STATE_ADC1_EN
 hal_lks32mc08x_reg.h, [60](#)
 HAL_ADC_TRIG.STATE_MCPWM_TRIG_EN
 hal_lks32mc08x_reg.h, [60](#)

HAL_ADC_TRIG_STATE_TIMER_TRIG_EN
 hal_lks32mc08x_reg.h, 60
 HAL_ADC_TRIG_STATE_TWO_SEG_TRIG_EN
 hal_lks32mc08x_reg.h, 60
 HAL_ADC_TRIGGER_MCPWM
 hal_lksmcu_adc.h, 73
 HAL_ADC_TRIGGER_MCPWM0
 hal_lksmcu_adc.h, 73
 HAL_ADC_TRIGGER_SOFT
 hal_lksmcu_adc.h, 73
 HAL_ADC_TRIGGER_TIMER
 hal_lksmcu_adc.h, 73
 HAL_ADC_TRIGGER_TIMER0
 hal_lksmcu_adc.h, 73
 HAL_ADC_TRIGGER_x
 hal_lksmcu_adc.h, 73
 HAL_ADC_TypeDef, 7
 AMC_A0, 9
 AMC_A1, 9
 AMC_B0, 9
 AMC_B1, 9
 CFG, 9
 CHN, 9
 CHNT0, 9
 CHNT1, 10
 DAT, 10
 DAT0.TH, 10
 DC_A0, 10
 DC_A1, 10
 DC_B0, 10
 DC_B1, 11
 GAIN_0, 11
 GAIN_1, 11
 IE, 11
 IF, 11
 null, 11
 SWT, 12
 TRIG, 12
 HAL_ADC_UpdateTrig
 hal_lks32mc08x_adc.c, 392
 HAL_ADC_x
 hal_lksmcu_adc.h, 73
 hal_baud
 hal_lks32mc08x_i2c.c, 496
 hal_baud_set
 hal_lks32mc08x_uart.c, 730
 HAL_CAN
 hal_lks32mc08x_reg.c, 680
 hal_lks32mc08x_reg.h, 62
 HAL_CAN_0
 hal_lksmcu_can.h, 89
 HAL_CAN_AE_0
 hal_lksmcu_can.h, 87
 HAL_CAN_AE_1
 hal_lksmcu_can.h, 87
 HAL_CAN_AE_10
 hal_lksmcu_can.h, 87
 HAL_CAN_AE_11
 hal_lksmcu_can.h, 87
 HAL_CAN_AE_12
 hal_lksmcu_can.h, 87
 HAL_CAN_AE_13
 hal_lksmcu_can.h, 87
 HAL_CAN_AE_14
 hal_lksmcu_can.h, 87
 HAL_CAN_AE_15
 hal_lksmcu_can.h, 87
 HAL_CAN_AE_2
 hal_lksmcu_can.h, 87
 HAL_CAN_AE_3
 hal_lksmcu_can.h, 87
 HAL_CAN_AE_4
 hal_lksmcu_can.h, 87
 HAL_CAN_AE_5
 hal_lksmcu_can.h, 87
 HAL_CAN_AE_6
 hal_lksmcu_can.h, 87
 HAL_CAN_AE_7
 hal_lksmcu_can.h, 87
 HAL_CAN_AE_8
 hal_lksmcu_can.h, 87
 HAL_CAN_AE_9
 hal_lksmcu_can.h, 87
 HAL_CAN_AE_x
 hal_lksmcu_can.h, 87
 HAL_CAN_ArbitrationLostCap
 hal_lks32mc08x_can.c, 400
 hal_lksmcu_can.h, 89
 HAL_CAN_Busoff
 hal_lks32mc08x_can.c, 400
 hal_lksmcu_can.h, 89
 HAL_CAN_Disable
 hal_lks32mc08x_can.c, 400
 hal_lksmcu_can.h, 89
 HAL_CAN_Enable
 hal_lks32mc08x_can.c, 401
 hal_lksmcu_can.h, 89
 HAL_CAN_ERROR_Ack
 hal_lksmcu_can.h, 87
 HAL_CAN_ERROR_Bit
 hal_lksmcu_can.h, 87
 HAL_CAN_ERROR_Crc
 hal_lksmcu_can.h, 87
 HAL_CAN_ERROR_Form
 hal_lksmcu_can.h, 87
 HAL_CAN_ERROR_None
 hal_lksmcu_can.h, 87
 HAL_CAN_ERROR_Stuff
 hal_lksmcu_can.h, 87
 HAL_CAN_ERROR_Unknown
 hal_lksmcu_can.h, 87
 HAL_CAN_ERROR_x
 hal_lksmcu_can.h, 87
 HAL_CAN_ErrorPassive
 hal_lks32mc08x_can.c, 401
 hal_lksmcu_can.h, 90

HAL_CAN_FRAME_IDE0_RTR0
 hal_lksmcu_can.h, 88
 HAL_CAN_FRAME_IDE0_RTR0_FD1_BRS0
 hal_lksmcu_can.h, 88
 HAL_CAN_FRAME_IDE0_RTR0_FD1_BRS1
 hal_lksmcu_can.h, 88
 HAL_CAN_FRAME_IDE0_RTR1
 hal_lksmcu_can.h, 88
 HAL_CAN_FRAME_IDE1_RTR0
 hal_lksmcu_can.h, 88
 HAL_CAN_FRAME_IDE1_RTR0_FD1_BRS0
 hal_lksmcu_can.h, 88
 HAL_CAN_FRAME_IDE1_RTR0_FD1_BRS1
 hal_lksmcu_can.h, 88
 HAL_CAN_FRAME_IDE1_RTR1
 hal_lksmcu_can.h, 88
 HAL_CAN_FRAME_x
 hal_lksmcu_can.h, 87
 HAL_CAN_GetBaud
 hal_lks32mc08x_can.c, 401
 hal_lksmcu_can.h, 90
 HAL_CAN_GetClockDiv
 hal_lks32mc08x_can.c, 401
 hal_lksmcu_can.h, 90
 HAL_CAN_GetClockFreq
 hal_lks32mc08x_can.c, 401
 hal_lksmcu_can.h, 90
 HAL_CAN_IDEFilter_All
 hal_lksmcu_can.h, 88
 HAL_CAN_IDEFilter_Ext
 hal_lksmcu_can.h, 88
 HAL_CAN_IDEFilter_St
 hal_lksmcu_can.h, 88
 HAL_CAN_IDEFilter_x
 hal_lksmcu_can.h, 88
 HAL_CAN_IDx_Filter
 hal_lks32mc08x_can.c, 402
 hal_lksmcu_can.h, 91
 HAL_CAN_IDx_FilterDisable
 hal_lks32mc08x_can.c, 402
 hal_lksmcu_can.h, 91
 HAL_CAN_IDx_FilterEnable
 hal_lks32mc08x_can.c, 403
 hal_lksmcu_can.h, 92
 HAL_CAN_IrqAll
 hal_lks32mc08x_irqfun.c, 529
 hal_lksmcu_irqfun.h, 211
 HAL_CAN_IrqAllClear
 hal_lks32mc08x_irqfun.c, 529
 hal_lksmcu_irqfun.h, 211
 HAL_CAN_IrqAllDisable
 hal_lks32mc08x_irqfun.c, 530
 hal_lksmcu_irqfun.h, 212
 HAL_CAN_IrqAllEnable
 hal_lks32mc08x_irqfun.c, 530
 hal_lksmcu_irqfun.h, 212
 HAL_CAN_IrqAllFlg
 hal_lks32mc08x_irqfun.c, 530
 hal_lksmcu_irqfun.h, 212
 HAL_CAN_IrqRxDone
 hal_lks32mc08x_irqfun.c, 531
 hal_lksmcu_irqfun.h, 213
 HAL_CAN_IrqRxDoneClear
 hal_lks32mc08x_irqfun.c, 531
 hal_lksmcu_irqfun.h, 213
 HAL_CAN_IrqRxDoneDisable
 hal_lks32mc08x_irqfun.c, 531
 hal_lksmcu_irqfun.h, 213
 HAL_CAN_IrqRxDoneEnable
 hal_lks32mc08x_irqfun.c, 532
 hal_lksmcu_irqfun.h, 214
 HAL_CAN_IrqRxDoneFlg
 hal_lks32mc08x_irqfun.c, 532
 hal_lksmcu_irqfun.h, 214
 HAL_CAN_IrqTxDone
 hal_lks32mc08x_irqfun.c, 532
 hal_lksmcu_irqfun.h, 214
 HAL_CAN_IrqTxDoneClear
 hal_lks32mc08x_irqfun.c, 533
 hal_lksmcu_irqfun.h, 215
 HAL_CAN_IrqTxDoneDisable
 hal_lks32mc08x_irqfun.c, 533
 hal_lksmcu_irqfun.h, 215
 HAL_CAN_IrqTxDoneEnable
 hal_lks32mc08x_irqfun.c, 533
 hal_lksmcu_irqfun.h, 215
 HAL_CAN_IrqTxDoneFlg
 hal_lks32mc08x_irqfun.c, 534
 hal_lksmcu_irqfun.h, 216
 HAL_CAN_KindOfError
 hal_lks32mc08x_can.c, 403
 hal_lksmcu_can.h, 92
 HAL_CAN_LBME_Disable
 hal_lks32mc08x_can.c, 403
 hal_lksmcu_can.h, 92
 HAL_CAN_LBME_Enable
 hal_lks32mc08x_can.c, 403
 hal_lksmcu_can.h, 92
 HAL_CAN_LBMI_Disable
 hal_lks32mc08x_can.c, 403
 hal_lksmcu_can.h, 92
 HAL_CAN_LBMI_Enable
 hal_lks32mc08x_can.c, 403
 hal_lksmcu_can.h, 92
 HAL_CAN_LOM_Disable
 hal_lks32mc08x_can.c, 404
 hal_lksmcu_can.h, 93
 HAL_CAN_LOM_Enable
 hal_lks32mc08x_can.c, 404
 hal_lksmcu_can.h, 93
 HAL_CAN_Reset
 hal_lks32mc08x_can.c, 404
 hal_lksmcu_can.h, 93
 HAL_CAN_ROverflow
 hal_lks32mc08x_can.c, 404
 hal_lksmcu_can.h, 93

HAL_CAN_RxData
 hal_lks32mc08x_can.c, 404
 hal_lksmcu_can.h, 93

HAL_CAN_RxDataBuf
 hal_lks32mc08x_can.c, 405
 hal_lksmcu_can.h, 94

HAL_CAN_RxDataFdf
 hal_lks32mc08x_can.c, 405
 hal_lksmcu_can.h, 94

HAL_CAN_RxDataID
 hal_lks32mc08x_can.c, 405
 hal_lksmcu_can.h, 94

HAL_CAN_RxDataIde
 hal_lks32mc08x_can.c, 405
 hal_lksmcu_can.h, 94

HAL_CAN_RxDataLen
 hal_lks32mc08x_can.c, 405
 hal_lksmcu_can.h, 94

HAL_CAN_RxDataRtr
 hal_lks32mc08x_can.c, 405
 hal_lksmcu_can.h, 94

HAL_CAN_RxErrorCnt
 hal_lks32mc08x_can.c, 405
 hal_lksmcu_can.h, 94

HAL_CAN_SetBaud
 hal_lks32mc08x_can.c, 406
 hal_lksmcu_can.h, 95

HAL_CAN_SetClockDiv
 hal_lks32mc08x_can.c, 406
 hal_lksmcu_can.h, 95

HAL_CAN_SetFastBaud
 hal_lks32mc08x_can.c, 406
 hal_lksmcu_can.h, 95

HAL_CAN_SetGpioRx
 hal_lks32mc08x_can.c, 406
 hal_lksmcu_can.h, 95

HAL_CAN_SetGpioTx
 hal_lks32mc08x_can.c, 407
 hal_lksmcu_can.h, 96

HAL_CAN_SetSTBMode
 hal_lks32mc08x_can.c, 407
 hal_lksmcu_can.h, 96

HAL_CAN_SetTDC
 hal_lks32mc08x_can.c, 407
 hal_lksmcu_can.h, 96

HAL_CAN_SS_Disable
 hal_lks32mc08x_can.c, 408
 hal_lksmcu_can.h, 97

HAL_CAN_SS_Enable
 hal_lks32mc08x_can.c, 408
 hal_lksmcu_can.h, 97

HAL_CAN_STBFull
 hal_lks32mc08x_can.c, 408
 hal_lksmcu_can.h, 97

HAL_CAN_STBMode_FIFO
 hal_lksmcu_can.h, 88

HAL_CAN_STBMode_Priority
 hal_lksmcu_can.h, 88

HAL_CAN_STBMode_x
 hal_lksmcu_can.h, 88

HAL_CAN_STBTrig_All
 hal_lksmcu_can.h, 88

HAL_CAN_STBTrig_One
 hal_lksmcu_can.h, 88

HAL_CAN_STBTrig_x
 hal_lksmcu_can.h, 88

HAL_CAN_TStat
 hal_lks32mc08x_can.c, 408
 hal_lksmcu_can.h, 97

HAL_CAN_TSSTAT_Empty
 hal_lksmcu_can.h, 89

HAL_CAN_TSSTAT_Full
 hal_lksmcu_can.h, 89

HAL_CAN_TSSTAT_Half
 hal_lksmcu_can.h, 89

HAL_CAN_TSSTAT_LessHalf
 hal_lksmcu_can.h, 89

HAL_CAN_TSSTAT_x
 hal_lksmcu_can.h, 88

HAL_CAN_TxCancel
 hal_lks32mc08x_can.c, 408
 hal_lksmcu_can.h, 97

HAL_CAN_TxData
 hal_lks32mc08x_can.c, 408
 hal_lksmcu_can.h, 97

HAL_CAN_TxData_STBTrig
 hal_lks32mc08x_can.c, 409
 hal_lksmcu_can.h, 98

HAL_CAN_TxData_STBWrite
 hal_lks32mc08x_can.c, 409
 hal_lksmcu_can.h, 98

HAL_CAN_TxErrorCnt
 hal_lks32mc08x_can.c, 409
 hal_lksmcu_can.h, 98

HAL_CAN_Txing
 hal_lks32mc08x_can.c, 409
 hal_lksmcu_can.h, 98

HAL_CAN_TypeDef, 12
 ALC, 14
 BTR0, 14
 BTR1, 14
 CMR, 14
 ECC, 14
 EWLR, 14
 IER, 14
 IR, 15
 MOD, 15
 null0, 15
 null1, 15
 null2, 15
 null3, 15
 RBSA, 15
 RMC, 16
 RXERR, 16
 SR, 16
 TXERR, 16

TXRX0, 16
 TXRX1, 16
 TXRX2, 17
 TXRX3, 17
 TXRX4, 17
 TXRX5, 17
 TXRX6, 17
 TXRX7, 17
 TXRX8, 18
 TXRX9, 18
 TXRXA, 18
 TXRXB, 18
 TXRXC, 18
 HAL_CAN_x
 hal_lksmcu_can.h, 89
 HAL_CLK_FREQ_RC_12MHZ
 hal_lksmcu_clk.h, 102
 HAL_CLK_FREQ_RC_24MHZ
 hal_lksmcu_clk.h, 102
 HAL_CLK_FREQ_RC_48MHZ
 hal_lksmcu_clk.h, 102
 HAL_CLK_FREQ_RC_4MHZ
 hal_lksmcu_clk.h, 102
 HAL_CLK_FREQ_RC_96MHZ
 hal_lksmcu_clk.h, 102
 HAL_CLK_FREQ_x
 hal_lksmcu_clk.h, 102
 HAL_CLK_GetMcuFreq
 hal_lks32mc08x_clk.c, 419
 hal_lksmcu_clk.h, 102
 HAL_CLK_mclk
 hal_lks32mc08x_clk.c, 421
 HAL_CLK_SetMcuFreq
 hal_lks32mc08x_clk.c, 420
 hal_lksmcu_clk.h, 103
 HAL_CMP
 hal_lks32mc08x_reg.c, 681
 hal_lks32mc08x_reg.h, 62
 HAL_CMP_0
 hal_lksmcu_cmp.h, 108
 HAL_CMP_1
 hal_lksmcu_cmp.h, 108
 HAL_CMP_2
 hal_lksmcu_cmp.h, 108
 HAL_CMP_3
 hal_lksmcu_cmp.h, 108
 HAL_CMP_4
 hal_lksmcu_cmp.h, 108
 HAL_CMP_5
 hal_lksmcu_cmp.h, 108
 HAL_CMP_CHANNEL_IN_0
 hal_lksmcu_cmp.h, 106
 HAL_CMP_CHANNEL_IN_DAC
 hal_lksmcu_cmp.h, 106
 HAL_CMP_CHANNEL_IN_DAC0
 hal_lksmcu_cmp.h, 106
 HAL_CMP_CHANNEL_IN_MID
 hal_lksmcu_cmp.h, 106
 HAL_CMP_CHANNEL_IN_REF
 hal_lksmcu_cmp.h, 106
 HAL_CMP_CHANNEL_INx
 hal_lksmcu_cmp.h, 106
 HAL_CMP_CHANNEL_IP0
 hal_lksmcu_cmp.h, 106
 HAL_CMP_CHANNEL_IP1
 hal_lksmcu_cmp.h, 107
 HAL_CMP_CHANNEL_IP2
 hal_lksmcu_cmp.h, 107
 HAL_CMP_CHANNEL_IP3
 hal_lksmcu_cmp.h, 107
 HAL_CMP_CHANNEL_IP4
 hal_lksmcu_cmp.h, 107
 HAL_CMP_CHANNEL_IP_AVSS
 hal_lksmcu_cmp.h, 107
 HAL_CMP_CHANNEL_IP_OPA0_IP
 hal_lksmcu_cmp.h, 106
 HAL_CMP_CHANNEL_IP_OPA0_OUTP
 hal_lksmcu_cmp.h, 107
 HAL_CMP_CHANNEL_IP_OPA1_OUTP
 hal_lksmcu_cmp.h, 107
 HAL_CMP_CHANNEL_IP_OPA2_OUTP
 hal_lksmcu_cmp.h, 107
 HAL_CMP_CHANNEL_IP_OPA3_IP
 hal_lksmcu_cmp.h, 107
 HAL_CMP_CHANNEL_IP_OPA3_OUTP
 hal_lksmcu_cmp.h, 107
 HAL_CMP_CHANNEL_IPx
 hal_lksmcu_cmp.h, 106
 HAL_CMP_Disable
 hal_lks32mc08x_cmp.c, 423
 hal_lksmcu_cmp.h, 108
 HAL_CMP_Enable
 hal_lks32mc08x_cmp.c, 424
 hal_lksmcu_cmp.h, 108
 HAL_CMP_GetData
 hal_lks32mc08x_cmp.c, 424
 hal_lksmcu_cmp.h, 109
 HAL_CMP_HYS_0mV
 hal_lksmcu_cmp.h, 107
 HAL_CMP_HYS_20mV
 hal_lksmcu_cmp.h, 107
 HAL_CMP_HYS_x
 hal_lksmcu_cmp.h, 107
 HAL_CMP_IrqCompare
 hal_lks32mc08x_irqfun.c, 534
 hal_lksmcu_irqfun.h, 216
 HAL_CMP_IrqCompareClear
 hal_lks32mc08x_irqfun.c, 534
 hal_lksmcu_irqfun.h, 216
 HAL_CMP_IrqCompareDisable
 hal_lks32mc08x_irqfun.c, 535
 hal_lksmcu_irqfun.h, 217
 HAL_CMP_IrqCompareEnable
 hal_lks32mc08x_irqfun.c, 535
 hal_lksmcu_irqfun.h, 217
 HAL_CMP_IrqCompareFlg

- hal_lks32mc08x_irqfun.c, 535
- hal_lksmcu_irqfun.h, 217
- HAL_CMP_MODE_FALLING_EDGE
 - hal_lksmcu_cmp.h, 107
- HAL_CMP_MODE_HIGH_LEVEL
 - hal_lksmcu_cmp.h, 107
- HAL_CMP_MODE_LOW_LEVEL
 - hal_lksmcu_cmp.h, 107
- HAL_CMP_MODE_RISING_EDGE
 - hal_lksmcu_cmp.h, 107
- HAL_CMP_MODE_x
 - hal_lksmcu_cmp.h, 107
- HAL_CMP_Reset
 - hal_lks32mc08x_cmp.c, 424
 - hal_lksmcu_cmp.h, 109
- HAL_CMP_SetChannel
 - hal_lks32mc08x_cmp.c, 425
 - hal_lksmcu_cmp.h, 110
- HAL_CMP_SetDigitalFilter
 - hal_lks32mc08x_cmp.c, 425
 - hal_lksmcu_cmp.h, 110
- HAL_CMP_SetGpioIn
 - hal_lks32mc08x_cmp.c, 426
 - hal_lksmcu_cmp.h, 110
- HAL_CMP_SetGpioOut
 - hal_lks32mc08x_cmp.c, 426
 - hal_lksmcu_cmp.h, 111
- HAL_CMP_SetHysteresis
 - hal_lks32mc08x_cmp.c, 427
 - hal_lksmcu_cmp.h, 111
- HAL_CMP_SetMode
 - hal_lks32mc08x_cmp.c, 427
 - hal_lksmcu_cmp.h, 112
- HAL_CMP_TypeDef, 19
 - BLCWIN, 19
 - CFG, 19
 - DATA, 19
 - IE, 20
 - IF, 20
 - TCLK, 20
- HAL_CMP_x
 - hal_lksmcu_cmp.h, 107
- HAL_CRC
 - hal_lks32mc08x_reg.c, 681
 - hal_lks32mc08x_reg.h, 62
- HAL_CRC_0
 - hal_lksmcu_crc.h, 115
- HAL_CRC_Calc
 - hal_lks32mc08x_crc.c, 432
 - hal_lksmcu_crc.h, 115
- HAL_CRC_Disable
 - hal_lks32mc08x_crc.c, 433
 - hal_lksmcu_crc.h, 116
- HAL_CRC_Enable
 - hal_lks32mc08x_crc.c, 433
 - hal_lksmcu_crc.h, 116
- HAL_CRC_MODEL_CRC16_CCITT
 - hal_lksmcu_crc.h, 115
- HAL_CRC_MODEL_CRC16_CCITT_FALSE
 - hal_lksmcu_crc.h, 115
- HAL_CRC_MODEL_CRC16_DNP
 - hal_lksmcu_crc.h, 115
- HAL_CRC_MODEL_CRC16_IBMx1
 - hal_lksmcu_crc.h, 115
- HAL_CRC_MODEL_CRC16_MAXIM
 - hal_lksmcu_crc.h, 115
- HAL_CRC_MODEL_CRC16_MODBU
 - hal_lksmcu_crc.h, 115
- HAL_CRC_MODEL_CRC16_USBx1
 - hal_lksmcu_crc.h, 115
- HAL_CRC_MODEL_CRC16_X25
 - hal_lksmcu_crc.h, 115
- HAL_CRC_MODEL_CRC16_XMODEM
 - hal_lksmcu_crc.h, 115
- HAL_CRC_MODEL_CRC32
 - hal_lksmcu_crc.h, 115
- HAL_CRC_MODEL_CRC32_MPEG_2
 - hal_lksmcu_crc.h, 115
- HAL_CRC_MODEL_CRC8
 - hal_lksmcu_crc.h, 115
- HAL_CRC_MODEL_CRC8_ITU
 - hal_lksmcu_crc.h, 115
- HAL_CRC_MODEL_CRC8_MAXIMx
 - hal_lksmcu_crc.h, 115
- HAL_CRC_MODEL_CRC8_ROHC
 - hal_lksmcu_crc.h, 115
- HAL_CRC_MODEL_x
 - hal_lksmcu_crc.h, 114
- HAL_CRC_Reflect
 - hal_lks32mc08x_crc.c, 433
- HAL_CRC_Reset
 - hal_lks32mc08x_crc.c, 434
 - hal_lksmcu_crc.h, 117
- HAL_CRC_TypeDef, 20
 - CR, 21
 - DR, 21
 - INIT, 21
 - POL, 21
- HAL_CRC_x
 - hal_lksmcu_crc.h, 115
- HAL_DAC_0
 - hal_lksmcu_dac.h, 119
- HAL_DAC_Disable
 - hal_lks32mc08x_dac.c, 438
 - hal_lksmcu_dac.h, 119
- HAL_DAC_Enable
 - hal_lks32mc08x_dac.c, 438
 - hal_lksmcu_dac.h, 120
- HAL_DAC_GetOutValMax
 - hal_lks32mc08x_dac.c, 438
 - hal_lksmcu_dac.h, 120
- HAL_DAC_GetRange
 - hal_lks32mc08x_dac.c, 439
 - hal_lksmcu_dac.h, 120
- HAL_DAC_Out2GpioDisable
 - hal_lks32mc08x_dac.c, 439

hal_lksmcu_dac.h, 121
 HAL_DAC_Out2GpioEnable
 hal_lks32mc08x_dac.c, 439
 hal_lksmcu_dac.h, 121
 HAL_DAC_RANGE_1_2
 hal_lksmcu_dac.h, 119
 HAL_DAC_RANGE_3_0V
 hal_lksmcu_dac.h, 119
 HAL_DAC_RANGE_4_85V
 hal_lksmcu_dac.h, 119
 HAL_DAC_RANGE_x
 hal_lksmcu_dac.h, 119
 HAL_DAC_Reset
 hal_lks32mc08x_dac.c, 440
 hal_lksmcu_dac.h, 121
 HAL_DAC_SetGpioOut
 hal_lks32mc08x_dac.c, 440
 hal_lksmcu_dac.h, 122
 HAL_DAC_SetOutVal
 hal_lks32mc08x_dac.c, 441
 hal_lksmcu_dac.h, 122
 HAL_DAC_SetOutVologemV
 hal_lks32mc08x_dac.c, 441
 hal_lksmcu_dac.h, 123
 HAL_DAC_SetRange
 hal_lks32mc08x_dac.c, 442
 hal_lksmcu_dac.h, 123
 HAL_DAC_x
 hal_lksmcu_dac.h, 119
 HAL_DMA
 hal_lks32mc08x_reg.c, 681
 hal_lks32mc08x_reg.h, 63
 HAL_DMA_0
 hal_lksmcu_dma.h, 126
 HAL_DMA_1
 hal_lksmcu_dma.h, 126
 HAL_DMA_2
 hal_lksmcu_dma.h, 126
 HAL_DMA_3
 hal_lksmcu_dma.h, 126
 HAL_DMA_Disable
 hal_lks32mc08x_dma.c, 445
 hal_lksmcu_dma.h, 126
 HAL_DMA_Enable
 hal_lks32mc08x_dma.c, 446
 hal_lksmcu_dma.h, 126
 HAL_DMA_I2cRx
 hal_lks32mc08x_dma.c, 446
 hal_lksmcu_dma.h, 127
 HAL_DMA_I2cTx
 hal_lks32mc08x_dma.c, 446
 hal_lksmcu_dma.h, 127
 HAL_DMA_IrqDone
 hal_lks32mc08x_irqfun.c, 536
 hal_lksmcu_irqfun.h, 218
 HAL_DMA_IrqDoneClear
 hal_lks32mc08x_irqfun.c, 536
 hal_lksmcu_irqfun.h, 218
 HAL_DMA_IrqDoneDisable
 hal_lks32mc08x_irqfun.c, 536
 hal_lksmcu_irqfun.h, 218
 HAL_DMA_IrqDoneEnable
 hal_lks32mc08x_irqfun.c, 537
 hal_lksmcu_irqfun.h, 219
 HAL_DMA_IrqDoneFlg
 hal_lks32mc08x_irqfun.c, 537
 hal_lksmcu_irqfun.h, 219
 HAL_DMA_IrqDoneTrig
 hal_lksmcu_irqfun.h, 219
 HAL_DMA_IrqErr
 hal_lks32mc08x_irqfun.c, 537
 hal_lksmcu_irqfun.h, 219
 HAL_DMA_IrqErrClear
 hal_lks32mc08x_irqfun.c, 538
 hal_lksmcu_irqfun.h, 220
 HAL_DMA_IrqErrDisable
 hal_lks32mc08x_irqfun.c, 538
 hal_lksmcu_irqfun.h, 220
 HAL_DMA_IrqErrEnable
 hal_lks32mc08x_irqfun.c, 538
 hal_lksmcu_irqfun.h, 220
 HAL_DMA_IrqErrFlg
 hal_lks32mc08x_irqfun.c, 539
 hal_lksmcu_irqfun.h, 221
 HAL_DMA_IrqErrTrig
 hal_lksmcu_irqfun.h, 221
 HAL_DMA_MemCopy
 hal_lks32mc08x_dma.c, 446
 hal_lksmcu_dma.h, 127
 HAL_DMA_Reset
 hal_lks32mc08x_dma.c, 446
 hal_lksmcu_dma.h, 127
 HAL_DMA_SpiRx
 hal_lks32mc08x_dma.c, 447
 hal_lksmcu_dma.h, 127
 HAL_DMA_SpiRxTx
 hal_lks32mc08x_dma.c, 447
 hal_lksmcu_dma.h, 128
 HAL_DMA_SpiTx
 hal_lks32mc08x_dma.c, 448
 hal_lksmcu_dma.h, 128
 HAL_DMA_TypeDef, 21
 CCR, 22
 CMAR, 22
 CPAR, 22
 CTMS, 22
 HAL_DMA_UartRx
 hal_lks32mc08x_dma.c, 448
 hal_lksmcu_dma.h, 129
 HAL_DMA_UartTx
 hal_lks32mc08x_dma.c, 448
 hal_lksmcu_dma.h, 129
 HAL_DMA_x
 hal_lksmcu_dma.h, 126
 HAL_DSP
 hal_lks32mc08x_reg.c, 681

hal_lks32mc08x_reg.h, 63

HAL_DSP_0

hal_lksmcu_dsp.h, 131

HAL_DSP_Cos

hal_lks32mc08x_dsp.c, 451

hal_lksmcu_dsp.h, 131

HAL_DSP_Disable

hal_lks32mc08x_dsp.c, 452

hal_lksmcu_dsp.h, 132

HAL_DSP_Div

hal_lks32mc08x_dsp.c, 452

hal_lksmcu_dsp.h, 132

HAL_DSP_Enable

hal_lks32mc08x_dsp.c, 452

hal_lksmcu_dsp.h, 132

HAL_DSP_Reset

hal_lks32mc08x_dsp.c, 453

hal_lksmcu_dsp.h, 133

HAL_DSP_Sin

hal_lks32mc08x_dsp.c, 453

hal_lksmcu_dsp.h, 133

HAL_DSP_SinCos

hal_lks32mc08x_dsp.c, 453

hal_lksmcu_dsp.h, 133

HAL_DSP_Sqrt

hal_lks32mc08x_dsp.c, 454

hal_lksmcu_dsp.h, 134

HAL_DSP_TypeDef, 23

ATAN, 23

COS, 23

MOD, 24

RAD, 24

RESERVED, 24

RESERVED1, 24

SC, 24

SIN, 24

SQRT, 25

THETA, 25

XY, 25

HAL_DSP_x

hal_lksmcu_dsp.h, 131

HAL_FLASH

hal_lks32mc08x_reg.c, 681

hal_lks32mc08x_reg.h, 63

HAL_FLASH_0

hal_lksmcu_flash.h, 136

HAL_FLASH_1

hal_lksmcu_flash.h, 136

HAL_FLASH_data

hal_lks32mc08x_flash.c, 465

HAL_FLASH_Disable

hal_lks32mc08x_flash.c, 457

hal_lksmcu_flash.h, 136

HAL_FLASH_DisableCache

hal_lks32mc08x_flash.c, 457

hal_lksmcu_flash.h, 137

HAL_FLASH_Enable

hal_lks32mc08x_flash.c, 457

hal_lksmcu_flash.h, 137

HAL_FLASH_Erase

hal_lks32mc08x_flash.c, 458

hal_lksmcu_flash.h, 138

HAL_FLASH_Program

hal_lks32mc08x_flash.c, 459

hal_lksmcu_flash.h, 138

HAL_FLASH_Read

hal_lks32mc08x_flash.c, 459

hal_lksmcu_flash.h, 139

HAL_FLASH_ReadSectorSize

hal_lks32mc08x_flash.c, 460

hal_lksmcu_flash.h, 140

HAL_FLASH_ReadUint16

hal_lks32mc08x_flash.c, 460

hal_lksmcu_flash.h, 140

HAL_FLASH_ReadUint32

hal_lks32mc08x_flash.c, 461

hal_lksmcu_flash.h, 141

HAL_FLASH_ReadUint8

hal_lks32mc08x_flash.c, 461

hal_lksmcu_flash.h, 141

HAL_FLASH_Reset

hal_lks32mc08x_flash.c, 462

hal_lksmcu_flash.h, 142

HAL_FLASH_SecurityParameterCrc

hal_lks32mc08x_flash.c, 462

HAL_FLASH_SecurityParameterRead

hal_lks32mc08x_flash.c, 463

hal_lksmcu_flash.h, 142

HAL_FLASH_SecurityParameterSave

hal_lks32mc08x_flash.c, 464

hal_lksmcu_flash.h, 143

HAL_FLASH_TypeDef, 25

ADDR, 26

CFG, 26

ERASE, 26

PROTECT, 26

RDATA, 26

READY, 26

WDATA, 27

HAL_FLASH_Update

hal_lks32mc08x_flash.c, 464

hal_lksmcu_flash.h, 143

HAL_FLASH_x

hal_lksmcu_flash.h, 136

HAL_GPIO

hal_lks32mc08x_reg.c, 681

hal_lks32mc08x_reg.h, 63

HAL_GPIO_AF_ADC

hal_lksmcu_gpio.h, 148

HAL_GPIO_AF_ADC_TRIG

hal_lksmcu_gpio.h, 148

HAL_GPIO_AF_CAN

hal_lksmcu_gpio.h, 148

HAL_GPIO_AF_CLK
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_AF_CLK_OUT
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_AF_CMP
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_AF_CMP_OUT
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_AF_DAC
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_AF_GPIO
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_AF_HALL
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_AF_I2C
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_AF_MCPWM
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_AF_OPA
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_AF_OPA_OUT
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_AF_QEP0
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_AF_QEP1
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_AF_SIF
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_AF_SPI
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_AF_TIMER0
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_AF_TIMER1
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_AF_TIMER2
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_AF_TIMER3
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_AF_UART
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_AFx
 hal_lksmcu_gpio.h, 147
 HAL_GPIO_Disable
 hal_lks32mc08x_gpio.c, 470
 hal_lksmcu_gpio.h, 151
 HAL_GPIO_Enable
 hal_lks32mc08x_gpio.c, 470
 hal_lksmcu_gpio.h, 151
 HAL_GPIO_GetInData
 hal_lks32mc08x_gpio.c, 470
 hal_lksmcu_gpio.h, 151
 HAL_GPIO_GetOutData
 hal_lks32mc08x_gpio.c, 471
 hal_lksmcu_gpio.h, 152
 HAL_GPIO_Invert
 hal_lks32mc08x_gpio.c, 471
 hal_lksmcu_gpio.h, 152
 HAL_GPIO_Irq
 hal_lks32mc08x_irqfun.c, 539
 hal_lksmcu_irqfun.h, 221
 HAL_GPIO_IrqClear
 hal_lks32mc08x_irqfun.c, 539
 hal_lksmcu_irqfun.h, 221
 HAL_GPIO_IrqDisable
 hal_lks32mc08x_irqfun.c, 540
 hal_lksmcu_irqfun.h, 222
 HAL_GPIO_IrqEnable
 hal_lks32mc08x_irqfun.c, 540
 hal_lksmcu_irqfun.h, 222
 HAL_GPIO_IrqFlg
 hal_lks32mc08x_irqfun.c, 540
 hal_lksmcu_irqfun.h, 222
 HAL_GPIO_IrqTrig
 hal_lksmcu_irqfun.h, 223
 HAL_GPIO_MODE_AFE
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_MODE_DEFAULT_0
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_MODE_DEFAULT_1
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_MODE_IN
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_MODE_IN_EN
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_MODE_INOUT
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_MODE_INOUT_0
 hal_lksmcu_gpio.h, 149
 HAL_GPIO_MODE_INOUT_1
 hal_lksmcu_gpio.h, 149
 HAL_GPIO_MODE_INUP
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_MODE_INUPOUT
 hal_lksmcu_gpio.h, 149
 HAL_GPIO_MODE_INUPOUT_0
 hal_lksmcu_gpio.h, 149
 HAL_GPIO_MODE_INUPOUT_1
 hal_lksmcu_gpio.h, 149
 HAL_GPIO_MODE_NONE
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_MODE_OUT
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_MODE_OUT_0
 hal_lksmcu_gpio.h, 149
 HAL_GPIO_MODE_OUT_1
 hal_lksmcu_gpio.h, 149
 HAL_GPIO_MODE_OUT_EN
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_MODE_PODE_EN
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_MODE_UP_EN
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_MODE_x
 hal_lksmcu_gpio.h, 148
 HAL_GPIO_P0_0
 hal_lksmcu_gpio.h, 149

HAL_GPIO_P0_1
hal_lksmcu_gpio.h, 149

HAL_GPIO_P0_10
hal_lksmcu_gpio.h, 149

HAL_GPIO_P0_11
hal_lksmcu_gpio.h, 149

HAL_GPIO_P0_12
hal_lksmcu_gpio.h, 149

HAL_GPIO_P0_13
hal_lksmcu_gpio.h, 149

HAL_GPIO_P0_14
hal_lksmcu_gpio.h, 149

HAL_GPIO_P0_15
hal_lksmcu_gpio.h, 149

HAL_GPIO_P0_2
hal_lksmcu_gpio.h, 149

HAL_GPIO_P0_3
hal_lksmcu_gpio.h, 149

HAL_GPIO_P0_4
hal_lksmcu_gpio.h, 149

HAL_GPIO_P0_5
hal_lksmcu_gpio.h, 149

HAL_GPIO_P0_6
hal_lksmcu_gpio.h, 149

HAL_GPIO_P0_7
hal_lksmcu_gpio.h, 149

HAL_GPIO_P0_8
hal_lksmcu_gpio.h, 149

HAL_GPIO_P0_9
hal_lksmcu_gpio.h, 149

HAL_GPIO_P1_0
hal_lksmcu_gpio.h, 149

HAL_GPIO_P1_1
hal_lksmcu_gpio.h, 149

HAL_GPIO_P1_10
hal_lksmcu_gpio.h, 150

HAL_GPIO_P1_11
hal_lksmcu_gpio.h, 150

HAL_GPIO_P1_12
hal_lksmcu_gpio.h, 150

HAL_GPIO_P1_13
hal_lksmcu_gpio.h, 150

HAL_GPIO_P1_14
hal_lksmcu_gpio.h, 150

HAL_GPIO_P1_15
hal_lksmcu_gpio.h, 150

HAL_GPIO_P1_2
hal_lksmcu_gpio.h, 149

HAL_GPIO_P1_3
hal_lksmcu_gpio.h, 149

HAL_GPIO_P1_4
hal_lksmcu_gpio.h, 149

HAL_GPIO_P1_5
hal_lksmcu_gpio.h, 149

HAL_GPIO_P1_6
hal_lksmcu_gpio.h, 149

HAL_GPIO_P1_7
hal_lksmcu_gpio.h, 149

HAL_GPIO_P1_8
hal_lksmcu_gpio.h, 149

HAL_GPIO_P1_9
hal_lksmcu_gpio.h, 150

HAL_GPIO_P2_0
hal_lksmcu_gpio.h, 150

HAL_GPIO_P2_1
hal_lksmcu_gpio.h, 150

HAL_GPIO_P2_10
hal_lksmcu_gpio.h, 150

HAL_GPIO_P2_11
hal_lksmcu_gpio.h, 150

HAL_GPIO_P2_12
hal_lksmcu_gpio.h, 150

HAL_GPIO_P2_13
hal_lksmcu_gpio.h, 150

HAL_GPIO_P2_14
hal_lksmcu_gpio.h, 150

HAL_GPIO_P2_15
hal_lksmcu_gpio.h, 150

HAL_GPIO_P2_2
hal_lksmcu_gpio.h, 150

HAL_GPIO_P2_3
hal_lksmcu_gpio.h, 150

HAL_GPIO_P2_4
hal_lksmcu_gpio.h, 150

HAL_GPIO_P2_5
hal_lksmcu_gpio.h, 150

HAL_GPIO_P2_6
hal_lksmcu_gpio.h, 150

HAL_GPIO_P2_7
hal_lksmcu_gpio.h, 150

HAL_GPIO_P2_8
hal_lksmcu_gpio.h, 150

HAL_GPIO_P2_9
hal_lksmcu_gpio.h, 150

HAL_GPIO_P3_0
hal_lksmcu_gpio.h, 150

HAL_GPIO_P3_1
hal_lksmcu_gpio.h, 150

HAL_GPIO_P3_10
hal_lksmcu_gpio.h, 150

HAL_GPIO_P3_11
hal_lksmcu_gpio.h, 150

HAL_GPIO_P3_12
hal_lksmcu_gpio.h, 150

HAL_GPIO_P3_13
hal_lksmcu_gpio.h, 150

HAL_GPIO_P3_14
hal_lksmcu_gpio.h, 150

HAL_GPIO_P3_15
hal_lksmcu_gpio.h, 150

HAL_GPIO_P3_2
hal_lksmcu_gpio.h, 150

HAL_GPIO_P3_3
hal_lksmcu_gpio.h, 150

HAL_GPIO_P3_4
hal_lksmcu_gpio.h, 150

HAL_GPIO_P3_5
 hal_lksmcu_gpio.h, 150

HAL_GPIO_P3_6
 hal_lksmcu_gpio.h, 150

HAL_GPIO_P3_7
 hal_lksmcu_gpio.h, 150

HAL_GPIO_P3_8
 hal_lksmcu_gpio.h, 150

HAL_GPIO_P3_9
 hal_lksmcu_gpio.h, 150

HAL_GPIO_P4_0
 hal_lksmcu_gpio.h, 150

HAL_GPIO_P4_1
 hal_lksmcu_gpio.h, 150

HAL_GPIO_P4_10
 hal_lksmcu_gpio.h, 150

HAL_GPIO_P4_11
 hal_lksmcu_gpio.h, 151

HAL_GPIO_P4_12
 hal_lksmcu_gpio.h, 151

HAL_GPIO_P4_13
 hal_lksmcu_gpio.h, 151

HAL_GPIO_P4_14
 hal_lksmcu_gpio.h, 151

HAL_GPIO_P4_15
 hal_lksmcu_gpio.h, 151

HAL_GPIO_P4_2
 hal_lksmcu_gpio.h, 150

HAL_GPIO_P4_3
 hal_lksmcu_gpio.h, 150

HAL_GPIO_P4_4
 hal_lksmcu_gpio.h, 150

HAL_GPIO_P4_5
 hal_lksmcu_gpio.h, 150

HAL_GPIO_P4_6
 hal_lksmcu_gpio.h, 150

HAL_GPIO_P4_7
 hal_lksmcu_gpio.h, 150

HAL_GPIO_P4_8
 hal_lksmcu_gpio.h, 150

HAL_GPIO_P4_9
 hal_lksmcu_gpio.h, 150

HAL_GPIO_P5_0
 hal_lksmcu_gpio.h, 151

HAL_GPIO_P5_1
 hal_lksmcu_gpio.h, 151

HAL_GPIO_P5_10
 hal_lksmcu_gpio.h, 151

HAL_GPIO_P5_11
 hal_lksmcu_gpio.h, 151

HAL_GPIO_P5_12
 hal_lksmcu_gpio.h, 151

HAL_GPIO_P5_13
 hal_lksmcu_gpio.h, 151

HAL_GPIO_P5_14
 hal_lksmcu_gpio.h, 151

HAL_GPIO_P5_15
 hal_lksmcu_gpio.h, 151

HAL_GPIO_P5_2
 hal_lksmcu_gpio.h, 151

HAL_GPIO_P5_3
 hal_lksmcu_gpio.h, 151

HAL_GPIO_P5_4
 hal_lksmcu_gpio.h, 151

HAL_GPIO_P5_5
 hal_lksmcu_gpio.h, 151

HAL_GPIO_P5_6
 hal_lksmcu_gpio.h, 151

HAL_GPIO_P5_7
 hal_lksmcu_gpio.h, 151

HAL_GPIO_P5_8
 hal_lksmcu_gpio.h, 151

HAL_GPIO_P5_9
 hal_lksmcu_gpio.h, 151

HAL_GPIO_Px
 hal_lksmcu_gpio.h, 149

HAL_GPIO_Reset
 hal_lks32mc08x_gpio.c, 472
 hal_lksmcu_gpio.h, 153

HAL_GPIO_RstReuseGpioDisable
 hal_lks32mc08x_gpio.c, 472
 hal_lksmcu_gpio.h, 153

HAL_GPIO_RstReuseGpioEnable
 hal_lks32mc08x_gpio.c, 472
 hal_lksmcu_gpio.h, 153

HAL_GPIO_SetAf
 hal_lks32mc08x_gpio.c, 472
 hal_lksmcu_gpio.h, 153

HAL_GPIO_SetHigh
 hal_lks32mc08x_gpio.c, 473
 hal_lksmcu_gpio.h, 154

HAL_GPIO_SetLow
 hal_lks32mc08x_gpio.c, 474
 hal_lksmcu_gpio.h, 155

HAL_GPIO_SetMode
 hal_lks32mc08x_gpio.c, 474
 hal_lksmcu_gpio.h, 155

HAL_GPIO_SwdReuseGpioDisable
 hal_lks32mc08x_gpio.c, 475
 hal_lksmcu_gpio.h, 156

HAL_GPIO_SwdReuseGpioEnable
 hal_lks32mc08x_gpio.c, 475
 hal_lksmcu_gpio.h, 156

HAL_GPIO_TypeDef, 27

 F3210, 28

 F7654, 28

 FBA98, 28

 FFEDC, 28

 LCKR, 28

 null1, 28

 PDI, 29

 PDO, 29

 PIE, 29

 PODE, 29

 POE, 29

 PUE, 29

HAL_GPIO_Write
 hal_iks32mc08x_gpio.c, 476
 hal_iksmcu_gpio.h, 157
 HAL_HALL
 hal_iks32mc08x_reg.c, 681
 hal_iks32mc08x_reg.h, 63
 HAL_HALL_0
 hal_iksmcu_hall.h, 161
 HAL_HALL_1
 hal_iksmcu_hall.h, 161
 HAL_HALL_Disable
 hal_iks32mc08x_hall.c, 481
 hal_iksmcu_hall.h, 161
 HAL_HALL_Enable
 hal_iks32mc08x_hall.c, 481
 hal_iksmcu_hall.h, 161
 hal_hall_filter
 hal_iks32mc08x_hall.c, 487
 HAL_HALL_GetCNT
 hal_iks32mc08x_hall.c, 481
 hal_iksmcu_hall.h, 162
 HAL_HALL_IrqChange
 hal_iks32mc08x_irqfun.c, 541
 hal_iksmcu_irqfun.h, 223
 HAL_HALL_IrqChangeClear
 hal_iks32mc08x_irqfun.c, 541
 hal_iksmcu_irqfun.h, 223
 HAL_HALL_IrqChangeDisable
 hal_iks32mc08x_irqfun.c, 541
 hal_iksmcu_irqfun.h, 223
 HAL_HALL_IrqChangeEnable
 hal_iks32mc08x_irqfun.c, 541
 hal_iksmcu_irqfun.h, 224
 HAL_HALL_IrqChangeFlg
 hal_iks32mc08x_irqfun.c, 542
 hal_iksmcu_irqfun.h, 224
 HAL_HALL_IrqOver
 hal_iks32mc08x_irqfun.c, 542
 hal_iksmcu_irqfun.h, 224
 HAL_HALL_IrqOverClear
 hal_iks32mc08x_irqfun.c, 542
 hal_iksmcu_irqfun.h, 225
 HAL_HALL_IrqOverDisable
 hal_iks32mc08x_irqfun.c, 543
 hal_iksmcu_irqfun.h, 225
 HAL_HALL_IrqOverEnable
 hal_iks32mc08x_irqfun.c, 543
 hal_iksmcu_irqfun.h, 225
 HAL_HALL_IrqOverFlg
 hal_iks32mc08x_irqfun.c, 543
 hal_iksmcu_irqfun.h, 226
 HAL_HALL_ReadHallData
 hal_iks32mc08x_hall.c, 482
 hal_iksmcu_hall.h, 162
 HAL_HALL_ReadHallWidth
 hal_iks32mc08x_hall.c, 482
 hal_iksmcu_hall.h, 162
 HAL_HALL_ReadTh
 hal_iks32mc08x_hall.c, 483
 hal_iksmcu_hall.h, 163
 HAL_HALL_Reset
 hal_iks32mc08x_hall.c, 483
 hal_iksmcu_hall.h, 163
 HAL_HALL_SetFilter
 hal_iks32mc08x_hall.c, 483
 hal_iksmcu_hall.h, 163
 HAL_HALL_SetFilter75
 hal_iks32mc08x_hall.c, 484
 hal_iksmcu_hall.h, 164
 HAL_HALL_SetFilteruS
 hal_iks32mc08x_hall.c, 484
 hal_iksmcu_hall.h, 164
 HAL_HALL_SetFreq
 hal_iks32mc08x_hall.c, 485
 hal_iksmcu_hall.h, 165
 HAL_HALL_SetGpio
 hal_iks32mc08x_hall.c, 485
 hal_iksmcu_hall.h, 165
 HAL_HALL_SetTh
 hal_iks32mc08x_hall.c, 486
 hal_iksmcu_hall.h, 166
 HAL_HALL_StartCounting
 hal_iks32mc08x_hall.c, 486
 hal_iksmcu_hall.h, 166
 HAL_HALL_StopCounting
 hal_iks32mc08x_hall.c, 487
 hal_iksmcu_hall.h, 167
 hal_hall_th
 hal_iks32mc08x_hall.c, 487
 HAL_HALL_TypeDef, 30
 CFG, 30
 CNT, 30
 INFO, 31
 TH, 31
 WIDTH, 31
 HAL_HALL_x
 hal_iksmcu_hall.h, 161
 HAL_I2C
 hal_iks32mc08x_reg.c, 682
 hal_iks32mc08x_reg.h, 63
 HAL_I2C_0
 hal_iksmcu_i2c.h, 169
 HAL_I2C_1
 hal_iksmcu_i2c.h, 169
 HAL_I2C_BusReset
 hal_iks32mc08x_i2c.c, 490
 hal_iksmcu_i2c.h, 169
 HAL_I2C_Disable
 hal_iks32mc08x_i2c.c, 491
 hal_iksmcu_i2c.h, 170
 HAL_I2C_Enable
 hal_iks32mc08x_i2c.c, 491
 hal_iksmcu_i2c.h, 170
 HAL_I2C_GetBaud
 hal_iks32mc08x_i2c.c, 491
 hal_iksmcu_i2c.h, 170

HAL_I2C_GetClockDiv
 hal_lks32mc08x_i2c.c, 492
 hal_lksmcu_i2c.h, 171
 HAL_I2C_GetClockFreq
 hal_lks32mc08x_i2c.c, 492
 hal_lksmcu_i2c.h, 171
 HAL_I2C_IrqBusErr
 hal_lks32mc08x_irqfun.c, 544
 hal_lksmcu_irqfun.h, 226
 HAL_I2C_IrqBusErrClear
 hal_lks32mc08x_irqfun.c, 544
 hal_lksmcu_irqfun.h, 226
 HAL_I2C_IrqBusErrDisable
 hal_lks32mc08x_irqfun.c, 544
 hal_lksmcu_irqfun.h, 227
 HAL_I2C_IrqBusErrEnable
 hal_lks32mc08x_irqfun.c, 545
 hal_lksmcu_irqfun.h, 227
 HAL_I2C_IrqBusErrFlg
 hal_lks32mc08x_irqfun.c, 545
 hal_lksmcu_irqfun.h, 227
 HAL_I2C_IrqDone
 hal_lks32mc08x_irqfun.c, 545
 hal_lksmcu_irqfun.h, 228
 HAL_I2C_IrqDoneClear
 hal_lks32mc08x_irqfun.c, 546
 hal_lksmcu_irqfun.h, 228
 HAL_I2C_IrqDoneDisable
 hal_lks32mc08x_irqfun.c, 546
 hal_lksmcu_irqfun.h, 228
 HAL_I2C_IrqDoneEnable
 hal_lks32mc08x_irqfun.c, 546
 hal_lksmcu_irqfun.h, 229
 HAL_I2C_IrqDoneFlg
 hal_lks32mc08x_irqfun.c, 547
 hal_lksmcu_irqfun.h, 229
 HAL_I2C_IrqStop
 hal_lks32mc08x_irqfun.c, 547
 hal_lksmcu_irqfun.h, 229
 HAL_I2C_IrqStopClear
 hal_lks32mc08x_irqfun.c, 547
 hal_lksmcu_irqfun.h, 230
 HAL_I2C_IrqStopDisable
 hal_lks32mc08x_irqfun.c, 548
 hal_lksmcu_irqfun.h, 230
 HAL_I2C_IrqStopEnable
 hal_lks32mc08x_irqfun.c, 548
 hal_lksmcu_irqfun.h, 230
 HAL_I2C_IrqStopFlg
 hal_lks32mc08x_irqfun.c, 548
 hal_lksmcu_irqfun.h, 231
 HAL_I2C_Reset
 hal_lks32mc08x_i2c.c, 493
 hal_lksmcu_i2c.h, 172
 HAL_I2C_RxData
 hal_lks32mc08x_i2c.c, 493
 hal_lksmcu_i2c.h, 172
 HAL_I2C_SetBaud
 hal_lks32mc08x_i2c.c, 493
 hal_lksmcu_i2c.h, 172
 HAL_I2C_SetClockDiv
 hal_lks32mc08x_i2c.c, 494
 hal_lksmcu_i2c.h, 173
 HAL_I2C_SetGpioScl
 hal_lks32mc08x_i2c.c, 494
 hal_lksmcu_i2c.h, 173
 HAL_I2C_SetGpioSda
 hal_lks32mc08x_i2c.c, 495
 hal_lksmcu_i2c.h, 174
 HAL_I2C_TxData
 hal_lks32mc08x_i2c.c, 495
 hal_lksmcu_i2c.h, 174
 HAL_I2C_TypeDef, 31
 ADDR, 32
 BCR, 32
 CFG, 32
 DATA, 32
 MSCR, 32
 SCR, 32
 HAL_I2C_x
 hal_lksmcu_i2c.h, 169
 hal_i2cio
 hal_lks32mc08x_i2c.c, 496
 HAL_INIT_DEFAULE_MCLK
 hal_lksmcu_init.h, 176
 HAL_INIT_DEFAULE_MCLK_MS
 hal_lksmcu_init.h, 176
 HAL_INIT_DEFAULE_MCLK_US
 hal_lksmcu_init.h, 176
 HAL_INIT_POWER_3.3V
 hal_lksmcu_init.h, 177
 HAL_INIT_POWER_5V
 hal_lksmcu_init.h, 177
 HAL_INIT_POWER_x
 hal_lksmcu_init.h, 176
 HAL_INIT_PowerReady
 hal_lks32mc08x_init.c, 500
 hal_lksmcu_init.h, 177
 HAL_INIT_SysInit
 hal_lks32mc08x_init.c, 500
 hal_lksmcu_init.h, 177
 HAL_IRQ_Disable
 hal_lks32mc08x_irq.c, 502
 hal_lksmcu_irq.h, 183
 HAL_IRQ_Enable
 hal_lks32mc08x_irq.c, 502
 hal_lksmcu_irq.h, 184
 HAL_IRQ_MODULE_ADC0
 hal_lksmcu_irq.h, 183
 HAL_IRQ_MODULE_CAN0
 hal_lksmcu_irq.h, 183
 HAL_IRQ_MODULE_CMP0
 hal_lksmcu_irq.h, 183
 HAL_IRQ_MODULE_DMA0
 hal_lksmcu_irq.h, 183
 HAL_IRQ_MODULE_DSP0

hal_lksmcu_irq.h, 183
HAL_IRQ_MODULE_GPIO0
hal_lksmcu_irq.h, 183
HAL_IRQ_MODULE_HALL0
hal_lksmcu_irq.h, 183
HAL_IRQ_MODULE_HardFault
hal_lksmcu_irq.h, 183
HAL_IRQ_MODULE_I2C0
hal_lksmcu_irq.h, 183
HAL_IRQ_MODULE_INIT0
hal_lksmcu_irq.h, 183
HAL_IRQ_MODULE_MCPWM0
hal_lksmcu_irq.h, 183
HAL_IRQ_MODULE_NonMaskableInt
hal_lksmcu_irq.h, 183
HAL_IRQ_MODULE_PendSV
hal_lksmcu_irq.h, 183
HAL_IRQ_MODULE_QEP0
hal_lksmcu_irq.h, 183
HAL_IRQ_MODULE_QEP1
hal_lksmcu_irq.h, 183
HAL_IRQ_MODULE_SIF0
hal_lksmcu_irq.h, 183
HAL_IRQ_MODULE_SPI0
hal_lksmcu_irq.h, 183
HAL_IRQ_MODULE_SVCall
hal_lksmcu_irq.h, 183
HAL_IRQ_MODULE_SysTick
hal_lksmcu_irq.h, 183
HAL_IRQ_MODULE_TIMER0
hal_lksmcu_irq.h, 183
HAL_IRQ_MODULE_TIMER1
hal_lksmcu_irq.h, 183
HAL_IRQ_MODULE_TIMER2
hal_lksmcu_irq.h, 183
HAL_IRQ_MODULE_TIMER3
hal_lksmcu_irq.h, 183
HAL_IRQ_MODULE_UART0
hal_lksmcu_irq.h, 183
HAL_IRQ_MODULE_UART1
hal_lksmcu_irq.h, 183
HAL_IRQ_MODULE_WAKE0
hal_lksmcu_irq.h, 183
HAL_IRQ_MODULE_x
hal_lksmcu_irq.h, 182
HAL_IRQ_ModuleClear
hal_lks32mc08x_irq.c, 503
hal_lksmcu_irq.h, 184
HAL_IRQ_ModuleDisable
hal_lks32mc08x_irq.c, 503
hal_lksmcu_irq.h, 184
HAL_IRQ_ModuleEnable
hal_lks32mc08x_irq.c, 503
hal_lksmcu_irq.h, 185
HAL_IRQ_Reset
hal_lks32mc08x_irq.c, 504
hal_lksmcu_irq.h, 185
HAL_IRQHandler_ADC0
hal_lksmcu_irq.h, 179
HAL_IRQHandler_CAN0
hal_lksmcu_irq.h, 179
HAL_IRQHandler_CMP0
hal_lksmcu_irq.h, 180
HAL_IRQHandler_DMA0
hal_lksmcu_irq.h, 180
HAL_IRQHandler_DSP0
hal_lksmcu_irq.h, 180
HAL_IRQHandler_GPIO0
hal_lksmcu_irq.h, 180
HAL_IRQHandler_HALL0
hal_lksmcu_irq.h, 180
HAL_IRQHandler_I2C0
hal_lksmcu_irq.h, 180
HAL_IRQHandler_INIT0
hal_lksmcu_irq.h, 180
HAL_IRQHandler_MCPWM0
hal_lksmcu_irq.h, 180
HAL_IRQHandler_PWRDN0
hal_lksmcu_irq.h, 181
HAL_IRQHandler_QEP0
hal_lksmcu_irq.h, 181
HAL_IRQHandler_QEP1
hal_lksmcu_irq.h, 181
HAL_IRQHandler_SIF0
hal_lksmcu_irq.h, 181
HAL_IRQHandler_SPI0
hal_lksmcu_irq.h, 181
HAL_IRQHandler_TIMER0
hal_lksmcu_irq.h, 181
HAL_IRQHandler_TIMER1
hal_lksmcu_irq.h, 181
HAL_IRQHandler_TIMER2
hal_lksmcu_irq.h, 182
HAL_IRQHandler_TIMER3
hal_lksmcu_irq.h, 182
HAL_IRQHandler_UART0
hal_lksmcu_irq.h, 182
HAL_IRQHandler_UART1
hal_lksmcu_irq.h, 182
HAL_IRQHandler_WAKE0
hal_lksmcu_irq.h, 182
HAL_IWDG
hal_lks32mc08x_reg.c, 682
hal_lks32mc08x_reg.h, 63
HAL_IWDG_0
hal_lksmcu_iwdg.h, 292
hal_iwdg_cfg
hal_lks32mc08x_iwdg.c, 641
HAL_IWDG_Disable
hal_lks32mc08x_iwdg.c, 639
hal_lksmcu_iwdg.h, 292
HAL_IWDG_Enable
hal_lks32mc08x_iwdg.c, 639
hal_lksmcu_iwdg.h, 292
HAL_IWDG_Feed
hal_lks32mc08x_iwdg.c, 640

hal_lksmcu_iwdg.h, 292
 HAL_IWDG_Reset
 hal_lks32mc08x_iwdg.c, 640
 hal_lksmcu_iwdg.h, 293
 HAL_IWDG_SetResetTimemS
 hal_lks32mc08x_iwdg.c, 640
 hal_lksmcu_iwdg.h, 293
 HAL_IWDG_TypeDef, 33
 WDT_CLR, 33
 HAL_IWDG_x
 hal_lksmcu_iwdg.h, 291
 hal_lks32mc08x.c, 55
 __HAL_LKS32MC08x_C__, 56
 hal_lks32mc08x.h, 56
 hal_lks32mc08x_adc.c
 HAL_ADC_Disable, 383
 HAL_ADC_Enable, 383
 HAL_ADC_EndSampling, 383
 HAL_ADC_GetRange, 384
 HAL_ADC_GetSamplingValues, 384
 HAL_ADC_GetValuesUnit, 385
 HAL_ADC_ManualSampling, 386
 HAL_ADC_Reset, 386
 HAL_ADC_SetGpioIn, 387
 HAL_ADC_SetGpioTriggerOut, 387
 HAL_ADC_SetReference, 389
 HAL_ADC_SetSamplingChannel, 389
 HAL_ADC_SetSamplingQuantity, 390
 HAL_ADC_SetTriggerSignal, 391
 HAL_ADC_SoftTrigger, 392
 HAL_ADC_StateReset, 392
 HAL_ADC_TRIG_STATE, 393
 HAL_ADC_UpdateTrig, 392
 hal_lks32mc08x_can.c
 baud_cnt, 410
 baud_param, 410
 CAN_Write_TBUF, 399
 GetBaudParam, 399
 HAL_CAN_ArbitrationLostCap, 400
 HAL_CAN_Busoff, 400
 HAL_CAN_Disable, 400
 HAL_CAN_Enable, 401
 HAL_CAN_ErrorPassive, 401
 HAL_CAN_GetBaud, 401
 HAL_CAN_GetClockDiv, 401
 HAL_CAN_GetClockFreq, 401
 HAL_CAN_IDx_Filter, 402
 HAL_CAN_IDx_FilterDisable, 402
 HAL_CAN_IDx_FilterEnable, 403
 HAL_CAN_KindOfError, 403
 HAL_CAN_LBME_Disable, 403
 HAL_CAN_LBME_Enable, 403
 HAL_CAN_LBMI_Disable, 403
 HAL_CAN_LBMI_Enable, 403
 HAL_CAN_LOM_Disable, 404
 HAL_CAN_LOM_Enable, 404
 HAL_CAN_Reset, 404
 HAL_CAN_ROverflow, 404
 HAL_CAN_RxData, 404
 HAL_CAN_RxDataBuf, 405
 HAL_CAN_RxDataFdf, 405
 HAL_CAN_RxDataID, 405
 HAL_CAN_RxDatalde, 405
 HAL_CAN_RxDataLen, 405
 HAL_CAN_RxDataRtr, 405
 HAL_CAN_RxErrorCnt, 405
 HAL_CAN_SetBaud, 406
 HAL_CAN_SetClockDiv, 406
 HAL_CAN_SetFastBaud, 406
 HAL_CAN_SetGpioRx, 406
 HAL_CAN_SetGpioTx, 407
 HAL_CAN_SetSTBMode, 407
 HAL_CAN_SetTDC, 407
 HAL_CAN_SS_Disable, 408
 HAL_CAN_SS_Enable, 408
 HAL_CAN_STBFull, 408
 HAL_CAN_TSSStat, 408
 HAL_CAN_TxCancel, 408
 HAL_CAN_TxData, 408
 HAL_CAN_TxData_STBTrig, 409
 HAL_CAN_TxData_STBWrite, 409
 HAL_CAN_TxErrorCnt, 409
 HAL_CAN_Txing, 409
 hal_rx_buf, 410
 hal_rx_id, 410
 hal_rx_ide, 410
 hal_rx_len, 410
 hal_rx_rtr, 411
 hal_lks32mc08x_clk.c
 HAL_CLK_GetMcuFreq, 419
 HAL_CLK_mclk, 421
 HAL_CLK_SetMcuFreq, 420
 hal_lks32mc08x_cmp.c
 HAL_CMP_Disable, 423
 HAL_CMP_Enable, 424
 HAL_CMP_GetData, 424
 HAL_CMP_Reset, 424
 HAL_CMP_SetChannel, 425
 HAL_CMP_SetDigitalFilter, 425
 HAL_CMP_SetGpioIn, 426
 HAL_CMP_SetGpioOut, 426
 HAL_CMP_SetHysteresis, 427
 HAL_CMP_SetMode, 427
 hal_lks32mc08x_crc.c
 CRC_Width, 431
 CRC_WIDTH_16, 432
 CRC_WIDTH_32, 432
 CRC_WIDTH_8, 432
 crcModels, 434
 HAL_CRC_Calc, 432
 HAL_CRC_Disable, 433
 HAL_CRC_Enable, 433
 HAL_CRC_Reflect, 433
 HAL_CRC_Reset, 434
 hal_lks32mc08x_dac.c
 HAL_DAC_Disable, 438

- HAL_DAC.Enable, 438
- HAL_DAC.GetOutValMax, 438
- HAL_DAC.GetRange, 439
- HAL_DAC.Out2GpioDisable, 439
- HAL_DAC.Out2GpioEnable, 439
- HAL_DAC.Reset, 440
- HAL_DAC.SetGpioOut, 440
- HAL_DAC.SetOutVal, 441
- HAL_DAC.SetOutVologemV, 441
- HAL_DAC.SetRange, 442
- hal_lks32mc08x_dma.c
 - HAL_DMA.Disable, 445
 - HAL_DMA.Enable, 446
 - HAL_DMA.I2cRx, 446
 - HAL_DMA.I2cTx, 446
 - HAL_DMA.MemCopy, 446
 - HAL_DMA.Reset, 446
 - HAL_DMA.SpiRx, 447
 - HAL_DMA.SpiRxTx, 447
 - HAL_DMA.SpiTx, 448
 - HAL_DMA.UartRx, 448
 - HAL_DMA.UartTx, 448
- hal_lks32mc08x_dsp.c
 - HAL_DSP.Cos, 451
 - HAL_DSP.Disable, 452
 - HAL_DSP.Div, 452
 - HAL_DSP.Enable, 452
 - HAL_DSP.Reset, 453
 - HAL_DSP.Sin, 453
 - HAL_DSP.SinCos, 453
 - HAL_DSP.Sqrt, 454
- hal_lks32mc08x_flash.c
 - HAL_FLASH.data, 465
 - HAL_FLASH.Disable, 457
 - HAL_FLASH.DisableCache, 457
 - HAL_FLASH.Enable, 457
 - HAL_FLASH.EnableCache, 458
 - HAL_FLASH.Erase, 458
 - HAL_FLASH.Program, 459
 - HAL_FLASH.Read, 459
 - HAL_FLASH.ReadSectorSize, 460
 - HAL_FLASH.ReadUint16, 460
 - HAL_FLASH.ReadUint32, 461
 - HAL_FLASH.ReadUint8, 461
 - HAL_FLASH.Reset, 462
 - HAL_FLASH.SecurityParameterCrc, 462
 - HAL_FLASH.SecurityParameterRead, 463
 - HAL_FLASH.SecurityParameterSave, 464
 - HAL_FLASH.Update, 464
- hal_lks32mc08x_gpio.c
 - HAL_GPIO.Disable, 470
 - HAL_GPIO.Enable, 470
 - HAL_GPIO.GetInData, 470
 - HAL_GPIO.GetOutData, 471
 - HAL_GPIO.Invert, 471
 - HAL_GPIO.Reset, 472
 - HAL_GPIO.RstReuseGpioDisable, 472
 - HAL_GPIO.RstReuseGpioEnable, 472
 - HAL_GPIO.SetAf, 472
 - HAL_GPIO.SetHigh, 473
 - HAL_GPIO.SetLow, 474
 - HAL_GPIO.SetMode, 474
 - HAL_GPIO.SwdReuseGpioDisable, 475
 - HAL_GPIO.SwdReuseGpioEnable, 475
 - HAL_GPIO.Write, 476
- hal_lks32mc08x_hall.c
 - HAL_HALL.Disable, 481
 - HAL_HALL.Enable, 481
 - hal_hall_filter, 487
 - HAL_HALL.GetCNT, 481
 - HAL_HALL.ReadHallData, 482
 - HAL_HALL.ReadHallWidth, 482
 - HAL_HALL.ReadTh, 483
 - HAL_HALL.Reset, 483
 - HAL_HALL.SetFilter, 483
 - HAL_HALL.SetFilter75, 484
 - HAL_HALL.SetFilteruS, 484
 - HAL_HALL.SetFreq, 485
 - HAL_HALL.SetGpio, 485
 - HAL_HALL.SetTh, 486
 - HAL_HALL.StartCounting, 486
 - HAL_HALL.StopCounting, 487
 - hal_hall_th, 487
- hal_lks32mc08x_i2c.c
 - hal_baud, 496
 - HAL_I2C.BusReset, 490
 - HAL_I2C.Disable, 491
 - HAL_I2C.Enable, 491
 - HAL_I2C.GetBaud, 491
 - HAL_I2C.GetClockDiv, 492
 - HAL_I2C.GetClockFreq, 492
 - HAL_I2C.Reset, 493
 - HAL_I2C.RxData, 493
 - HAL_I2C.SetBaud, 493
 - HAL_I2C.SetClockDiv, 494
 - HAL_I2C.SetGpioScl, 494
 - HAL_I2C.SetGpioSda, 495
 - HAL_I2C.TxData, 495
 - hal_i2cio, 496
- hal_lks32mc08x_init.c
 - HAL_INIT.PowerReady, 500
 - HAL_INIT.SysInit, 500
- hal_lks32mc08x_irq.c
 - HAL_IRQ.Disable, 502
 - HAL_IRQ.Enable, 502
 - HAL_IRQ.ModuleClear, 503
 - HAL_IRQ.ModuleDisable, 503
 - HAL_IRQ.ModuleEnable, 503
 - HAL_IRQ.Reset, 504
- hal_lks32mc08x_irqfun.c
 - HAL_ADC.IrqHardErr, 517
 - HAL_ADC.IrqHardErrClear, 517
 - HAL_ADC.IrqHardErrDisable, 517
 - HAL_ADC.IrqHardErrEnable, 518
 - HAL_ADC.IrqHardErrFlg, 518
 - HAL_ADC.IrqIdleSamp, 518

- HAL_ADC.IrqIdleSampClear, 518
- HAL_ADC.IrqIdleSampDisable, 518
- HAL_ADC.IrqIdleSampEnable, 519
- HAL_ADC.IrqIdleSampFlg, 519
- HAL_ADC.IrqSamp1, 519
- HAL_ADC.IrqSamp1Clear, 519
- HAL_ADC.IrqSamp1Disable, 520
- HAL_ADC.IrqSamp1Enable, 520
- HAL_ADC.IrqSamp1Flg, 520
- HAL_ADC.IrqSamp2, 521
- HAL_ADC.IrqSamp2Clear, 521
- HAL_ADC.IrqSamp2Disable, 521
- HAL_ADC.IrqSamp2Enable, 521
- HAL_ADC.IrqSamp2Flg, 522
- HAL_ADC.IrqSamp3, 522
- HAL_ADC.IrqSamp3Clear, 522
- HAL_ADC.IrqSamp3Disable, 523
- HAL_ADC.IrqSamp3Enable, 523
- HAL_ADC.IrqSamp3Flg, 523
- HAL_ADC.IrqSamp4, 524
- HAL_ADC.IrqSamp4Clear, 524
- HAL_ADC.IrqSamp4Disable, 524
- HAL_ADC.IrqSamp4Enable, 525
- HAL_ADC.IrqSamp4Flg, 525
- HAL_ADC.IrqSoftErr, 525
- HAL_ADC.IrqSoftErrClear, 526
- HAL_ADC.IrqSoftErrDisable, 526
- HAL_ADC.IrqSoftErrEnable, 526
- HAL_ADC.IrqSoftErrFlg, 526
- HAL_ADC.IrqWatchdog, 527
- HAL_ADC.IrqWatchdog1, 527
- HAL_ADC.IrqWatchdog1Clear, 527
- HAL_ADC.IrqWatchdog1Disable, 527
- HAL_ADC.IrqWatchdog1Enable, 528
- HAL_ADC.IrqWatchdog1Flg, 528
- HAL_ADC.IrqWatchdogClear, 528
- HAL_ADC.IrqWatchdogDisable, 528
- HAL_ADC.IrqWatchdogEnable, 528
- HAL_ADC.IrqWatchdogFlg, 529
- HAL_CAN.IrqAll, 529
- HAL_CAN.IrqAllClear, 529
- HAL_CAN.IrqAllDisable, 530
- HAL_CAN.IrqAllEnable, 530
- HAL_CAN.IrqAllFlg, 530
- HAL_CAN.IrqRxDone, 531
- HAL_CAN.IrqRxDoneClear, 531
- HAL_CAN.IrqRxDoneDisable, 531
- HAL_CAN.IrqRxDoneEnable, 532
- HAL_CAN.IrqRxDoneFlg, 532
- HAL_CAN.IrqTxDone, 532
- HAL_CAN.IrqTxDoneClear, 533
- HAL_CAN.IrqTxDoneDisable, 533
- HAL_CAN.IrqTxDoneEnable, 533
- HAL_CAN.IrqTxDoneFlg, 534
- HAL_CMP.IrqCompare, 534
- HAL_CMP.IrqCompareClear, 534
- HAL_CMP.IrqCompareDisable, 535
- HAL_CMP.IrqCompareEnable, 535
- HAL_CMP.IrqCompareFlg, 535
- HAL_DMA.IrqDone, 536
- HAL_DMA.IrqDoneClear, 536
- HAL_DMA.IrqDoneDisable, 536
- HAL_DMA.IrqDoneEnable, 537
- HAL_DMA.IrqDoneFlg, 537
- HAL_DMA.IrqErr, 537
- HAL_DMA.IrqErrClear, 538
- HAL_DMA.IrqErrDisable, 538
- HAL_DMA.IrqErrEnable, 538
- HAL_DMA.IrqErrFlg, 539
- HAL_GPIO.Irq, 539
- HAL_GPIO.IrqClear, 539
- HAL_GPIO.IrqDisable, 540
- HAL_GPIO.IrqEnable, 540
- HAL_GPIO.IrqFlg, 540
- HAL_HALL.IrqChange, 541
- HAL_HALL.IrqChangeClear, 541
- HAL_HALL.IrqChangeDisable, 541
- HAL_HALL.IrqChangeEnable, 541
- HAL_HALL.IrqChangeFlg, 542
- HAL_HALL.IrqOver, 542
- HAL_HALL.IrqOverClear, 542
- HAL_HALL.IrqOverDisable, 543
- HAL_HALL.IrqOverEnable, 543
- HAL_HALL.IrqOverFlg, 543
- HAL_I2C.IrqBusErr, 544
- HAL_I2C.IrqBusErrClear, 544
- HAL_I2C.IrqBusErrDisable, 544
- HAL_I2C.IrqBusErrEnable, 545
- HAL_I2C.IrqBusErrFlg, 545
- HAL_I2C.IrqDone, 545
- HAL_I2C.IrqDoneClear, 546
- HAL_I2C.IrqDoneDisable, 546
- HAL_I2C.IrqDoneEnable, 546
- HAL_I2C.IrqDoneFlg, 547
- HAL_I2C.IrqStop, 547
- HAL_I2C.IrqStopClear, 547
- HAL_I2C.IrqStopDisable, 548
- HAL_I2C.IrqStopEnable, 548
- HAL_I2C.IrqStopFlg, 548
- HAL_MCPWM.IrqFail, 549
- HAL_MCPWM.IrqFail0, 549
- HAL_MCPWM.IrqFail0Clear, 549
- HAL_MCPWM.IrqFail0Disable, 550
- HAL_MCPWM.IrqFail0Enable, 550
- HAL_MCPWM.IrqFail0Flg, 550
- HAL_MCPWM.IrqFail1, 551
- HAL_MCPWM.IrqFail1Clear, 551
- HAL_MCPWM.IrqFail1Disable, 551
- HAL_MCPWM.IrqFail1Enable, 552
- HAL_MCPWM.IrqFail1Flg, 552
- HAL_MCPWM.IrqFailClear, 552
- HAL_MCPWM.IrqFailDisable, 552
- HAL_MCPWM.IrqFailEnable, 553
- HAL_MCPWM.IrqFailFlg, 553
- HAL_MCPWM.IrqOverflow, 553
- HAL_MCPWM.IrqOverflowClear, 554

HAL_MCPWM_IrqOverflowDisable, 554
HAL_MCPWM_IrqOverflowEnable, 554
HAL_MCPWM_IrqOverflowFlg, 555
HAL_MCPWM_IrqShort, 555
HAL_MCPWM_IrqShortClear, 555
HAL_MCPWM_IrqShortDisable, 556
HAL_MCPWM_IrqShortEnable, 556
HAL_MCPWM_IrqShortFlg, 556
HAL_MCPWM_IrqTH00, 556
HAL_MCPWM_IrqTH00Clear, 557
HAL_MCPWM_IrqTH00Disable, 557
HAL_MCPWM_IrqTH00Enable, 557
HAL_MCPWM_IrqTH00Flg, 558
HAL_MCPWM_IrqTH01, 558
HAL_MCPWM_IrqTH01Clear, 558
HAL_MCPWM_IrqTH01Disable, 559
HAL_MCPWM_IrqTH01Enable, 559
HAL_MCPWM_IrqTH01Flg, 559
HAL_MCPWM_IrqTH10, 560
HAL_MCPWM_IrqTH10Clear, 560
HAL_MCPWM_IrqTH10Disable, 560
HAL_MCPWM_IrqTH10Enable, 561
HAL_MCPWM_IrqTH10Flg, 561
HAL_MCPWM_IrqTH11, 561
HAL_MCPWM_IrqTH11Clear, 562
HAL_MCPWM_IrqTH11Disable, 562
HAL_MCPWM_IrqTH11Enable, 562
HAL_MCPWM_IrqTH11Flg, 562
HAL_MCPWM_IrqTH20, 563
HAL_MCPWM_IrqTH20Clear, 563
HAL_MCPWM_IrqTH20Disable, 563
HAL_MCPWM_IrqTH20Enable, 564
HAL_MCPWM_IrqTH20Flg, 564
HAL_MCPWM_IrqTH21, 564
HAL_MCPWM_IrqTH21Clear, 565
HAL_MCPWM_IrqTH21Disable, 565
HAL_MCPWM_IrqTH21Enable, 565
HAL_MCPWM_IrqTH21Flg, 565
HAL_MCPWM_IrqTH30, 566
HAL_MCPWM_IrqTH30Clear, 566
HAL_MCPWM_IrqTH30Disable, 566
HAL_MCPWM_IrqTH30Enable, 567
HAL_MCPWM_IrqTH30Flg, 567
HAL_MCPWM_IrqTH31, 567
HAL_MCPWM_IrqTH31Clear, 568
HAL_MCPWM_IrqTH31Disable, 568
HAL_MCPWM_IrqTH31Enable, 568
HAL_MCPWM_IrqTH31Flg, 569
HAL_MCPWM_IrqTH40, 569
HAL_MCPWM_IrqTH40Clear, 569
HAL_MCPWM_IrqTH40Disable, 569
HAL_MCPWM_IrqTH40Enable, 569
HAL_MCPWM_IrqTH40Flg, 569
HAL_MCPWM_IrqTH41, 570
HAL_MCPWM_IrqTH41Clear, 570
HAL_MCPWM_IrqTH41Disable, 570
HAL_MCPWM_IrqTH41Enable, 570
HAL_MCPWM_IrqTH41Flg, 570
HAL_MCPWM_IrqTH50, 570
HAL_MCPWM_IrqTH50Clear, 570
HAL_MCPWM_IrqTH50Disable, 571
HAL_MCPWM_IrqTH50Enable, 571
HAL_MCPWM_IrqTH50Flg, 571
HAL_MCPWM_IrqTH51, 571
HAL_MCPWM_IrqTH51Clear, 571
HAL_MCPWM_IrqTH51Disable, 571
HAL_MCPWM_IrqTH51Enable, 571
HAL_MCPWM_IrqTH51Flg, 572
HAL_MCPWM_IrqTmr0, 572
HAL_MCPWM_IrqTmr0Clear, 572
HAL_MCPWM_IrqTmr0Disable, 572
HAL_MCPWM_IrqTmr0Enable, 573
HAL_MCPWM_IrqTmr0Flg, 573
HAL_MCPWM_IrqTmr1, 573
HAL_MCPWM_IrqTmr1Clear, 574
HAL_MCPWM_IrqTmr1Disable, 574
HAL_MCPWM_IrqTmr1Enable, 574
HAL_MCPWM_IrqTmr1Flg, 575
HAL_MCPWM_IrqTmr2, 575
HAL_MCPWM_IrqTmr2Clear, 575
HAL_MCPWM_IrqTmr2Disable, 576
HAL_MCPWM_IrqTmr2Enable, 576
HAL_MCPWM_IrqTmr2Flg, 576
HAL_MCPWM_IrqTmr3, 576
HAL_MCPWM_IrqTmr3Clear, 577
HAL_MCPWM_IrqTmr3Disable, 577
HAL_MCPWM_IrqTmr3Enable, 577
HAL_MCPWM_IrqTmr3Flg, 578
HAL_MCPWM_IrqUpdate, 578
HAL_MCPWM_IrqUpdateClear, 578
HAL_MCPWM_IrqUpdateDisable, 579
HAL_MCPWM_IrqUpdateEnable, 579
HAL_MCPWM_IrqUpdateFlg, 579
HAL_MCPWM_IrqZeroCross, 580
HAL_MCPWM_IrqZeroCrossClear, 580
HAL_MCPWM_IrqZeroCrossDisable, 580
HAL_MCPWM_IrqZeroCrossEnable, 581
HAL_MCPWM_IrqZeroCrossFlg, 581
HAL_SPI_IrqDone, 581
HAL_SPI_IrqDoneClear, 582
HAL_SPI_IrqDoneDisable, 582
HAL_SPI_IrqDoneEnable, 582
HAL_SPI_IrqDoneFlg, 582
HAL_SPI_IrqErr, 583
HAL_SPI_IrqErrClear, 583
HAL_SPI_IrqErrDisable, 584
HAL_SPI_IrqErrEnable, 584
HAL_SPI_IrqErrFlg, 584
HAL_SPI_IrqOverflow, 584
HAL_SPI_IrqOverflowClear, 585
HAL_SPI_IrqOverflowDisable, 585
HAL_SPI_IrqOverflowEnable, 585
HAL_SPI_IrqOverflowFlg, 586
HAL_SW_Irq, 586
HAL_SW_IrqClear, 586
HAL_SW_IrqDisable, 586

HAL_SW_IrqEnable, 587
HAL_SW_IrqFlg, 587
HAL_SW_IrqTrig, 587
HAL_TIMER_IrqCompare0, 587
HAL_TIMER_IrqCompare0Clear, 587
HAL_TIMER_IrqCompare0Disable, 588
HAL_TIMER_IrqCompare0Enable, 588
HAL_TIMER_IrqCompare0Flg, 588
HAL_TIMER_IrqCompare1, 589
HAL_TIMER_IrqCompare1Clear, 589
HAL_TIMER_IrqCompare1Disable, 589
HAL_TIMER_IrqCompare1Enable, 590
HAL_TIMER_IrqCompare1Flg, 590
HAL_TIMER_IrqOverflow, 590
HAL_TIMER_IrqOverflowClear, 591
HAL_TIMER_IrqOverflowDisable, 591
HAL_TIMER_IrqOverflowEnable, 591
HAL_TIMER_IrqOverflowFlg, 592
HAL_UART_IrqAbd, 592
HAL_UART_IrqAbdClear, 592
HAL_UART_IrqAbdDisable, 592
HAL_UART_IrqAbdEnable, 592
HAL_UART_IrqAbdFlg, 593
HAL_UART_IrqCheckErr, 593
HAL_UART_IrqCheckErrClear, 593
HAL_UART_IrqCheckErrDisable, 593
HAL_UART_IrqCheckErrEnable, 594
HAL_UART_IrqCheckErrFlg, 594
HAL_UART_IrqIdle, 594
HAL_UART_IrqIdleClear, 595
HAL_UART_IrqIdleDisable, 595
HAL_UART_IrqIdleEnable, 595
HAL_UART_IrqIdleFlg, 595
HAL_UART_IrqLbd, 595
HAL_UART_IrqLbdClear, 595
HAL_UART_IrqLbdDisable, 595
HAL_UART_IrqLbdEnable, 596
HAL_UART_IrqLbdFlg, 596
HAL_UART_IrqRxDone, 596
HAL_UART_IrqRxDoneClear, 596
HAL_UART_IrqRxDoneDisable, 597
HAL_UART_IrqRxDoneEnable, 597
HAL_UART_IrqRxDoneFlg, 597
HAL_UART_IrqRxOv, 598
HAL_UART_IrqRxOvClear, 598
HAL_UART_IrqRxOvDisable, 598
HAL_UART_IrqRxOvEnable, 598
HAL_UART_IrqRxOvFlg, 598
HAL_UART_IrqStopErr, 598
HAL_UART_IrqStopErrClear, 599
HAL_UART_IrqStopErrDisable, 599
HAL_UART_IrqStopErrEnable, 599
HAL_UART_IrqStopErrFlg, 600
HAL_UART_IrqTxBufEmpty, 600
HAL_UART_IrqTxBufEmptyClear, 600
HAL_UART_IrqTxBufEmptyDisable, 601
HAL_UART_IrqTxBufEmptyEnable, 601
HAL_UART_IrqTxBufEmptyFlg, 601
HAL_UART_IrqTxDone, 602
HAL_UART_IrqTxDoneClear, 602
HAL_UART_IrqTxDoneDisable, 602
HAL_UART_IrqTxDoneEnable, 603
HAL_UART_IrqTxDoneFlg, 603
HAL_UART_IrqTxOv, 603
HAL_UART_IrqTxOvClear, 603
HAL_UART_IrqTxOvDisable, 604
HAL_UART_IrqTxOvEnable, 604
HAL_UART_IrqTxOvFlg, 604
hal_ks32mc08x_iwdg.c
hal_iwdg_cfg, 641
HAL_IWDG_Disable, 639
HAL_IWDG_Enable, 639
HAL_IWDG_Feed, 640
HAL_IWDG_Reset, 640
HAL_IWDG_SetResetTimemS, 640
hal_ks32mc08x_mcpwm.c
HAL_MCPWM_Disable, 644
HAL_MCPWM_Enable, 645
HAL_MCPWM_GetClockDiv, 645
HAL_MCPWM_GetClockFreq, 645
HAL_MCPWM_GetFreq, 646
HAL_MCPWM_GetThCnt, 647
HAL_MCPWM_Reset, 647
HAL_MCPWM_SetAdcTrig, 648
HAL_MCPWM_SetAdcTrigQ15, 648
HAL_MCPWM_SetClockDiv, 649
HAL_MCPWM_SetDead, 649
HAL_MCPWM_SetDefaultState, 649
HAL_MCPWM_SetDrivePolarity, 650
HAL_MCPWM_SetDrivePolarityPort, 650
HAL_MCPWM_SetFailFilter, 651
HAL_MCPWM_SetFailSignal, 652
HAL_MCPWM_SetFreq, 652
HAL_MCPWM_SetGpioFail, 653
HAL_MCPWM_SetGpioOut, 653
HAL_MCPWM_SetOutputDisable, 654
HAL_MCPWM_SetOutputEnable, 654
HAL_MCPWM_SetOutState, 654
HAL_MCPWM_SetPwm, 655
HAL_MCPWM_SetPwmQ15, 655
HAL_MCPWM_SetRemap, 656
HAL_MCPWM_SetSwap, 656
HAL_MCPWM_SetSwapPort, 657
HAL_MCPWM_SetThCnt, 658
HAL_MCPWM_SetUpdateMode, 658
HAL_MCPWM_StartCounting, 658
HAL_MCPWM_StartCountingSync, 659
HAL_MCPWM_STATE, 661
HAL_MCPWM_STATE_INIT_FLG, 661
HAL_MCPWM_StopCounting, 659
HAL_MCPWM_UpdatePolarityState, 660
hal_ks32mc08x_opa.c
HAL_OPA_Disable, 668
HAL_OPA_Enable, 668
HAL_OPA_Out2GpioDisable, 668
HAL_OPA_Out2GpioEnable, 669

HAL_OPA_Reset, 669
 HAL_OPA_SetGain, 669
 HAL_OPA_SetGpioIn, 669
 HAL_OPA_SetGpioOut, 670
 hal_lks32mc08x_qep.c
 HAL_QEP_Disable, 674
 HAL_QEP_Enable, 674
 HAL_QEP_GetClockDiv, 674
 HAL_QEP_GetClockFreq, 675
 HAL_QEP_ReadCnt, 675
 HAL_QEP_Reset, 676
 HAL_QEP_ResetCnt, 676
 HAL_QEP_SetClockDiv, 676
 HAL_QEP_SetGpio, 676
 HAL_QEP_SetMode, 677
 HAL_QEP_SetTh, 677
 HAL_LKS32MC08X_REG
 hal_lks32mc08x_reg.h, 61
 hal_lks32mc08x_reg.c
 HAL_ADC, 680
 HAL_CAN, 680
 HAL_CMP, 681
 HAL_CRC, 681
 HAL_DMA, 681
 HAL_DSP, 681
 HAL_FLASH, 681
 HAL_GPIO, 681
 HAL_HALL, 681
 HAL_I2C, 682
 HAL_IWDG, 682
 HAL_MCPWM, 682
 HAL_QEP, 682
 HAL_SPI, 682
 HAL_SYS, 682
 HAL_TIMER, 682
 HAL_UART, 683
 hal_lks32mc08x_reg.h
 HAL_ADC, 62
 HAL_ADC_TRIG_STATE_ADC0_EN, 60
 HAL_ADC_TRIG_STATE_ADC1_EN, 60
 HAL_ADC_TRIG_STATE_MCPWM_TRIG_EN, 60
 HAL_ADC_TRIG_STATE_TIMER_TRIG_EN, 60
 HAL_ADC_TRIG_STATE_TWO_SEG_TRIG_EN, 60
 HAL_CAN, 62
 HAL_CMP, 62
 HAL_CRC, 62
 HAL_DMA, 63
 HAL_DSP, 63
 HAL_FLASH, 63
 HAL_GPIO, 63
 HAL_HALL, 63
 HAL_I2C, 63
 HAL_IWDG, 63
 HAL_LKS32MC08X_REG, 61
 HAL_MCPWM, 63
 HAL_MCPWM_STATE_H_DEFAULT_BIT, 61
 HAL_MCPWM_STATE_H_MOS_EN_BIT, 61
 HAL_MCPWM_STATE_L_DEFAULT_BIT, 61
 HAL_MCPWM_STATE_L_MOS_EN_BIT, 61
 HAL_MCPWM_STATE_STATE, 61
 HAL_MCPWM_STATE_STATE_MASK, 61
 HAL_MCPWM_STATE_STATE_MAX, 62
 HAL_MCPWM_STATE_STATE_POS, 62
 HAL_MCPWM_STATE_SW_BIT, 62
 HAL_QEP, 64
 HAL_SPI, 64
 HAL_SYS, 64
 HAL_TIMER, 64
 HAL_UART, 64
 hal_lks32mc08x_reset.c
 HAL_RESET_ResetCore, 684
 HAL_RESET_ResetGlobal, 684
 HAL_RESET_ResetRegister, 684
 hal_lks32mc08x_spi.c
 HAL_SPI_Disable, 687
 HAL_SPI_Enable, 687
 HAL_SPI_GetBaud, 687
 HAL_SPI_GetClockDiv, 688
 HAL_SPI_GetClockFreq, 688
 HAL_SPI_Reset, 689
 HAL_SPI_RxData, 689
 HAL_SPI_RxDataWidth16, 690
 HAL_SPI_RxTxData, 690
 HAL_SPI_RxTxDataWidth16, 691
 HAL_SPI_SetBaud, 692
 HAL_SPI_SetClockDiv, 692
 HAL_SPI_SetGpioClk, 693
 HAL_SPI_SetGpioCs, 693
 HAL_SPI_SetGpioDi, 694
 HAL_SPI_SetGpioDo, 694
 HAL_SPI_SetGpioMiso, 695
 HAL_SPI_SetGpioMosi, 696
 HAL_SPI_SetMode, 696
 HAL_SPI_TxData, 697
 HAL_SPI_TxDataWidth16, 697
 hal_lks32mc08x_timer.c
 HAL_TIMER_Disable, 703
 hal_timer_ena, 710
 HAL_TIMER_Enable, 703
 HAL_TIMER_GetClockDiv, 703
 HAL_TIMER_GetClockFreq, 704
 HAL_TIMER_GetFreq, 704
 HAL_TIMER_GetThCnt, 704
 HAL_TIMER_Reset, 705
 HAL_TIMER_SetClockDiv, 705
 HAL_TIMER_SetFreq, 705
 HAL_TIMER_SetGpioIn, 706
 HAL_TIMER_SetGpioOut, 707
 HAL_TIMER_SetPeriodCount, 707
 HAL_TIMER_SetPeriodmS, 708
 HAL_TIMER_SetPerioduS, 708
 HAL_TIMER_SetPwm, 709
 HAL_TIMER_SetPwmQ15, 709
 HAL_TIMER_SetWorkMode, 709
 HAL_TIMER_StartCounting, 710
 HAL_TIMER_StopCounting, 710

hal_lks32mc08x_tmp.c
 HAL_TMP_Adc2Temperature, 715
 HAL_TMP_Adc2Temperature01, 716
 HAL_TMP_Disable, 716
 HAL_TMP_Enable, 717
 HAL_TMP_GetTemperature, 717
 HAL_TMP_GetTemperature01, 718
 HAL_TMP_Reset, 719
 tmp_coef_a, 719
 tmp_coef_b, 719
 hal_lks32mc08x_trim.c
 HAL_TRIM_Read, 721
 HAL_TRIM_ReadId, 722
 HAL_TRIM_ReadIdCrc32, 723
 HAL_TRIM_ReadMcu08, 723
 hal_lks32mc08x_uart.c
 hal_baud_set, 730
 HAL_UART_Disable, 726
 HAL_UART_Enable, 726
 HAL_UART_GetBaud, 726
 HAL_UART_GetClockDiv, 726
 HAL_UART_GetClockFreq, 726
 HAL_UART_ReadBuffer, 727
 HAL_UART_Reset, 727
 HAL_UART_SetBaud, 727
 HAL_UART_SetClockDiv, 728
 HAL_UART_SetGpioRx, 728
 HAL_UART_SetGpioTx, 728
 HAL_UART_SetMode, 729
 HAL_UART_SetPolarity, 729
 HAL_UART_TxData, 730
 HAL_UART_WriteBuffer, 730
 hal_lks32mc08x_wake.c
 HAL_WAKE_DeepSleepToTimeWakeUp, 735
 HAL_WAKE_DeepSleepToWake, 735
 HAL_WAKE_DeepSleepTimeWakeUp, 736
 hal_lksmcu.h, 58
 hal_lksmcu_adc.h
 HAL_ADC_0, 73
 HAL_ADC_1, 74
 HAL_ADC_2, 74
 HAL_ADC_3, 74
 HAL_ADC_CHANNEL_0, 71
 HAL_ADC_CHANNEL_1, 71
 HAL_ADC_CHANNEL_10, 71
 HAL_ADC_CHANNEL_11, 71
 HAL_ADC_CHANNEL_12, 71
 HAL_ADC_CHANNEL_13, 71
 HAL_ADC_CHANNEL_14, 71
 HAL_ADC_CHANNEL_15, 71
 HAL_ADC_CHANNEL_16, 71
 HAL_ADC_CHANNEL_17, 71
 HAL_ADC_CHANNEL_18, 71
 HAL_ADC_CHANNEL_19, 71
 HAL_ADC_CHANNEL_2, 71
 HAL_ADC_CHANNEL_3, 71
 HAL_ADC_CHANNEL_4, 71
 HAL_ADC_CHANNEL_5, 71
 HAL_ADC_CHANNEL_6, 71
 HAL_ADC_CHANNEL_7, 71
 HAL_ADC_CHANNEL_8, 71
 HAL_ADC_CHANNEL_9, 71
 HAL_ADC_CHANNEL_AGND, 71
 HAL_ADC_CHANNEL_OPA0, 71
 HAL_ADC_CHANNEL_OPA1, 71
 HAL_ADC_CHANNEL_OPA2, 71
 HAL_ADC_CHANNEL_OPA3, 71
 HAL_ADC_CHANNEL_TMP0, 71
 HAL_ADC_CHANNEL_x, 71
 HAL_ADC_Disable, 74
 HAL_ADC_Enable, 74
 HAL_ADC_EndSampling, 74
 HAL_ADC_GetRange, 75
 HAL_ADC_GetSamplingValues, 75
 HAL_ADC_GetValuesUnit, 76
 HAL_ADC_ManualSampling, 77
 HAL_ADC_REFERENCE_EXTERNAL, 72
 HAL_ADC_REFERENCE_INTERNAL, 71
 HAL_ADC_REFERENCE_INTERNAL_2_4, 72
 HAL_ADC_REFERENCE_INTERNAL_3_6, 72
 HAL_ADC_REFERENCE_x, 71
 HAL_ADC_Reset, 77
 HAL_ADC_SAMP_NUMBER_0, 72
 HAL_ADC_SAMP_NUMBER_1, 72
 HAL_ADC_SAMP_NUMBER_10, 72
 HAL_ADC_SAMP_NUMBER_11, 72
 HAL_ADC_SAMP_NUMBER_12, 72
 HAL_ADC_SAMP_NUMBER_13, 72
 HAL_ADC_SAMP_NUMBER_14, 72
 HAL_ADC_SAMP_NUMBER_15, 72
 HAL_ADC_SAMP_NUMBER_2, 72
 HAL_ADC_SAMP_NUMBER_3, 72
 HAL_ADC_SAMP_NUMBER_4, 72
 HAL_ADC_SAMP_NUMBER_5, 72
 HAL_ADC_SAMP_NUMBER_6, 72
 HAL_ADC_SAMP_NUMBER_7, 72
 HAL_ADC_SAMP_NUMBER_8, 72
 HAL_ADC_SAMP_NUMBER_9, 72
 HAL_ADC_SAMP_NUMBER_MAX, 72
 HAL_ADC_SAMP_NUMBER_x, 72
 HAL_ADC_SAMP_SEQUENCE_0, 72
 HAL_ADC_SAMP_SEQUENCE_1, 72
 HAL_ADC_SAMP_SEQUENCE_10, 73
 HAL_ADC_SAMP_SEQUENCE_11, 73
 HAL_ADC_SAMP_SEQUENCE_12, 73
 HAL_ADC_SAMP_SEQUENCE_13, 73
 HAL_ADC_SAMP_SEQUENCE_14, 73
 HAL_ADC_SAMP_SEQUENCE_15, 73
 HAL_ADC_SAMP_SEQUENCE_16, 73
 HAL_ADC_SAMP_SEQUENCE_17, 73
 HAL_ADC_SAMP_SEQUENCE_18, 73
 HAL_ADC_SAMP_SEQUENCE_19, 73
 HAL_ADC_SAMP_SEQUENCE_2, 73
 HAL_ADC_SAMP_SEQUENCE_3, 73
 HAL_ADC_SAMP_SEQUENCE_4, 73
 HAL_ADC_SAMP_SEQUENCE_5, 73

- HAL_ADC_SAMP_SEQUENCE_6, 73
- HAL_ADC_SAMP_SEQUENCE_7, 73
- HAL_ADC_SAMP_SEQUENCE_8, 73
- HAL_ADC_SAMP_SEQUENCE_9, 73
- HAL_ADC_SAMP_SEQUENCE_x, 72
- HAL_ADC_SetGpioIn, 78
- HAL_ADC_SetGpioTriggerOut, 78
- HAL_ADC_SetReference, 79
- HAL_ADC_SetSamplingChannel, 79
- HAL_ADC_SetSamplingQuantity, 80
- HAL_ADC_SetTriggerSignal, 81
- HAL_ADC_SoftTrigger, 82
- HAL_ADC_StateReset, 82
- HAL_ADC_TRIGGER_MCPWM, 73
- HAL_ADC_TRIGGER_MCPWM0, 73
- HAL_ADC_TRIGGER_SOFT, 73
- HAL_ADC_TRIGGER_TIMER, 73
- HAL_ADC_TRIGGER_TIMER0, 73
- HAL_ADC_TRIGGER_x, 73
- HAL_ADC_x, 73
- hal_lksmcu_can.h
 - HAL_CAN_0, 89
 - HAL_CAN_AE_0, 87
 - HAL_CAN_AE_1, 87
 - HAL_CAN_AE_10, 87
 - HAL_CAN_AE_11, 87
 - HAL_CAN_AE_12, 87
 - HAL_CAN_AE_13, 87
 - HAL_CAN_AE_14, 87
 - HAL_CAN_AE_15, 87
 - HAL_CAN_AE_2, 87
 - HAL_CAN_AE_3, 87
 - HAL_CAN_AE_4, 87
 - HAL_CAN_AE_5, 87
 - HAL_CAN_AE_6, 87
 - HAL_CAN_AE_7, 87
 - HAL_CAN_AE_8, 87
 - HAL_CAN_AE_9, 87
 - HAL_CAN_AE_x, 87
 - HAL_CAN_ArbitrationLostCap, 89
 - HAL_CAN_Busoff, 89
 - HAL_CAN_Disable, 89
 - HAL_CAN_Enable, 89
 - HAL_CAN_ERROR_Ack, 87
 - HAL_CAN_ERROR_Bit, 87
 - HAL_CAN_ERROR_Crc, 87
 - HAL_CAN_ERROR_Form, 87
 - HAL_CAN_ERROR_None, 87
 - HAL_CAN_ERROR_Stuff, 87
 - HAL_CAN_ERROR_Unknown, 87
 - HAL_CAN_ERROR_x, 87
 - HAL_CAN_ErrorPassive, 90
 - HAL_CAN_FRAME_IDE0_RTR0, 88
 - HAL_CAN_FRAME_IDE0_RTR0_FD1_BRS0, 88
 - HAL_CAN_FRAME_IDE0_RTR0_FD1_BRS1, 88
 - HAL_CAN_FRAME_IDE0_RTR1, 88
 - HAL_CAN_FRAME_IDE1_RTR0, 88
 - HAL_CAN_FRAME_IDE1_RTR0_FD1_BRS0, 88
 - HAL_CAN_FRAME_IDE1_RTR0_FD1_BRS1, 88
 - HAL_CAN_FRAME_IDE1_RTR1, 88
 - HAL_CAN_FRAME_x, 87
 - HAL_CAN_GetBaud, 90
 - HAL_CAN_GetClockDiv, 90
 - HAL_CAN_GetClockFreq, 90
 - HAL_CAN_IDEFilter_All, 88
 - HAL_CAN_IDEFilter_Ex, 88
 - HAL_CAN_IDEFilter_St, 88
 - HAL_CAN_IDEFilter_x, 88
 - HAL_CAN_IDx_Filter, 91
 - HAL_CAN_IDx_FilterDisable, 91
 - HAL_CAN_IDx_FilterEnable, 92
 - HAL_CAN_KindOfError, 92
 - HAL_CAN_LBME_Disable, 92
 - HAL_CAN_LBME_Enable, 92
 - HAL_CAN_LBMI_Disable, 92
 - HAL_CAN_LBMI_Enable, 92
 - HAL_CAN_LOM_Disable, 93
 - HAL_CAN_LOM_Enable, 93
 - HAL_CAN_Reset, 93
 - HAL_CAN_ROverflow, 93
 - HAL_CAN_RxData, 93
 - HAL_CAN_RxDataBuf, 94
 - HAL_CAN_RxDataFdf, 94
 - HAL_CAN_RxDataID, 94
 - HAL_CAN_RxDatalde, 94
 - HAL_CAN_RxDataLen, 94
 - HAL_CAN_RxDataRtr, 94
 - HAL_CAN_RxErrorCnt, 94
 - HAL_CAN_SetBaud, 95
 - HAL_CAN_SetClockDiv, 95
 - HAL_CAN_SetFastBaud, 95
 - HAL_CAN_SetGpioRx, 95
 - HAL_CAN_SetGpioTx, 96
 - HAL_CAN_SetSTBMode, 96
 - HAL_CAN_SetTDC, 96
 - HAL_CAN_SS_Disable, 97
 - HAL_CAN_SS_Enable, 97
 - HAL_CAN_STBFull, 97
 - HAL_CAN_STBMode_FIFO, 88
 - HAL_CAN_STBMode_Priority, 88
 - HAL_CAN_STBMode_x, 88
 - HAL_CAN_STBTrig_All, 88
 - HAL_CAN_STBTrig_One, 88
 - HAL_CAN_STBTrig_x, 88
 - HAL_CAN_TSSStat, 97
 - HAL_CAN_TSSTAT_Empty, 89
 - HAL_CAN_TSSTAT_Full, 89
 - HAL_CAN_TSSTAT_Half, 89
 - HAL_CAN_TSSTAT_LessHalf, 89
 - HAL_CAN_TSSTAT_x, 88
 - HAL_CAN_TxCancel, 97
 - HAL_CAN_TxData, 97
 - HAL_CAN_TxData_STBTrig, 98
 - HAL_CAN_TxData_STBWrite, 98
 - HAL_CAN_TxErrorCnt, 98
 - HAL_CAN_Txing, 98

- HAL_CAN_x, 89
- hal_lksmcu_clk.h
 - HAL_CLK_FREQ_RC_12MHZ, 102
 - HAL_CLK_FREQ_RC_24MHZ, 102
 - HAL_CLK_FREQ_RC_48MHZ, 102
 - HAL_CLK_FREQ_RC_4MHZ, 102
 - HAL_CLK_FREQ_RC_96MHZ, 102
 - HAL_CLK_FREQ_x, 102
 - HAL_CLK_GetMcuFreq, 102
 - HAL_CLK_SetMcuFreq, 103
- hal_lksmcu_cmp.h
 - HAL_CMP_0, 108
 - HAL_CMP_1, 108
 - HAL_CMP_2, 108
 - HAL_CMP_3, 108
 - HAL_CMP_4, 108
 - HAL_CMP_5, 108
 - HAL_CMP_CHANNEL.IN_0, 106
 - HAL_CMP_CHANNEL.IN_DAC, 106
 - HAL_CMP_CHANNEL.IN_DAC0, 106
 - HAL_CMP_CHANNEL.IN_MID, 106
 - HAL_CMP_CHANNEL.IN_REF, 106
 - HAL_CMP_CHANNEL.INx, 106
 - HAL_CMP_CHANNEL.IP0, 106
 - HAL_CMP_CHANNEL.IP1, 107
 - HAL_CMP_CHANNEL.IP2, 107
 - HAL_CMP_CHANNEL.IP3, 107
 - HAL_CMP_CHANNEL.IP4, 107
 - HAL_CMP_CHANNEL.IP_AVSS, 107
 - HAL_CMP_CHANNEL.IP_OPA0_IP, 106
 - HAL_CMP_CHANNEL.IP_OPA0_OUTP, 107
 - HAL_CMP_CHANNEL.IP_OPA1_OUTP, 107
 - HAL_CMP_CHANNEL.IP_OPA2_OUTP, 107
 - HAL_CMP_CHANNEL.IP_OPA3_IP, 107
 - HAL_CMP_CHANNEL.IP_OPA3_OUTP, 107
 - HAL_CMP_CHANNEL.IPx, 106
 - HAL_CMP_Disable, 108
 - HAL_CMP_Enable, 108
 - HAL_CMP_GetData, 109
 - HAL_CMP_HYS_0mV, 107
 - HAL_CMP_HYS_20mV, 107
 - HAL_CMP_HYS_x, 107
 - HAL_CMP_MODE_FALLING_EDGE, 107
 - HAL_CMP_MODE_HIGH_LEVEL, 107
 - HAL_CMP_MODE_LOW_LEVEL, 107
 - HAL_CMP_MODE_RISING_EDGE, 107
 - HAL_CMP_MODE_x, 107
 - HAL_CMP_Reset, 109
 - HAL_CMP_SetChannel, 110
 - HAL_CMP_SetDigitalFilter, 110
 - HAL_CMP_SetGpioIn, 110
 - HAL_CMP_SetGpioOut, 111
 - HAL_CMP_SetHysteresis, 111
 - HAL_CMP_SetMode, 112
 - HAL_CMP_x, 107
- hal_lksmcu_crc.h
 - HAL_CRC_0, 115
 - HAL_CRC_Calc, 115
 - HAL_CRC_Disable, 116
 - HAL_CRC_Enable, 116
 - HAL_CRC_MODEL_CRC16_CCITT, 115
 - HAL_CRC_MODEL_CRC16_CCITT_FALSE, 115
 - HAL_CRC_MODEL_CRC16_DNP, 115
 - HAL_CRC_MODEL_CRC16_IBMx1, 115
 - HAL_CRC_MODEL_CRC16_MAXIM, 115
 - HAL_CRC_MODEL_CRC16_MODBU, 115
 - HAL_CRC_MODEL_CRC16_USBx1, 115
 - HAL_CRC_MODEL_CRC16_X25, 115
 - HAL_CRC_MODEL_CRC16_XMODEM, 115
 - HAL_CRC_MODEL_CRC32, 115
 - HAL_CRC_MODEL_CRC32_MPEG_2, 115
 - HAL_CRC_MODEL_CRC8, 115
 - HAL_CRC_MODEL_CRC8_ITU, 115
 - HAL_CRC_MODEL_CRC8_MAXIMx, 115
 - HAL_CRC_MODEL_CRC8_ROHC, 115
 - HAL_CRC_MODEL_x, 114
 - HAL_CRC_Reset, 117
 - HAL_CRC_x, 115
- hal_lksmcu_dac.h
 - HAL_DAC_0, 119
 - HAL_DAC_Disable, 119
 - HAL_DAC_Enable, 120
 - HAL_DAC_GetOutValMax, 120
 - HAL_DAC_GetRange, 120
 - HAL_DAC_Out2GpioDisable, 121
 - HAL_DAC_Out2GpioEnable, 121
 - HAL_DAC_RANGE_1_2, 119
 - HAL_DAC_RANGE_3_0V, 119
 - HAL_DAC_RANGE_4_85V, 119
 - HAL_DAC_RANGE_x, 119
 - HAL_DAC_Reset, 121
 - HAL_DAC_SetGpioOut, 122
 - HAL_DAC_SetOutVal, 122
 - HAL_DAC_SetOutVoltageV, 123
 - HAL_DAC_SetRange, 123
 - HAL_DAC_x, 119
- hal_lksmcu_dma.h
 - HAL_DMA_0, 126
 - HAL_DMA_1, 126
 - HAL_DMA_2, 126
 - HAL_DMA_3, 126
 - HAL_DMA_Disable, 126
 - HAL_DMA_Enable, 126
 - HAL_DMA_I2cRx, 127
 - HAL_DMA_I2cTx, 127
 - HAL_DMA_MemCopy, 127
 - HAL_DMA_Reset, 127
 - HAL_DMA_SpiRx, 127
 - HAL_DMA_SpiRxTx, 128
 - HAL_DMA_SpiTx, 128
 - HAL_DMA_UartRx, 129
 - HAL_DMA_UartTx, 129
 - HAL_DMA_x, 126
- hal_lksmcu_dsp.h
 - HAL_DSP_0, 131
 - HAL_DSP_Cos, 131

- HAL_DSP_Disable, 132
- HAL_DSP_Div, 132
- HAL_DSP_Enable, 132
- HAL_DSP_Reset, 133
- HAL_DSP_Sin, 133
- HAL_DSP_SinCos, 133
- HAL_DSP_Sqrt, 134
- HAL_DSP_x, 131
- hal_ksmcu_flash.h
 - HAL_FLASH_0, 136
 - HAL_FLASH_1, 136
 - HAL_FLASH_Disable, 136
 - HAL_FLASH_DisableCache, 137
 - HAL_FLASH_Enable, 137
 - HAL_FLASH_EnableCache, 137
 - HAL_FLASH_Erase, 138
 - HAL_FLASH_Program, 138
 - HAL_FLASH_Read, 139
 - HAL_FLASH_ReadSectorSize, 140
 - HAL_FLASH_ReadUint16, 140
 - HAL_FLASH_ReadUint32, 141
 - HAL_FLASH_ReadUint8, 141
 - HAL_FLASH_Reset, 142
 - HAL_FLASH_SecurityParameterRead, 142
 - HAL_FLASH_SecurityParameterSave, 143
 - HAL_FLASH_Update, 143
 - HAL_FLASH_x, 136
- hal_ksmcu_gpio.h
 - HAL_GPIO_AF_ADC, 148
 - HAL_GPIO_AF_ADC_TRIG, 148
 - HAL_GPIO_AF_CAN, 148
 - HAL_GPIO_AF_CLK, 148
 - HAL_GPIO_AF_CLK_OUT, 148
 - HAL_GPIO_AF_CMP, 148
 - HAL_GPIO_AF_CMP_OUT, 148
 - HAL_GPIO_AF_DAC, 148
 - HAL_GPIO_AF_GPIO, 148
 - HAL_GPIO_AF_HALL, 148
 - HAL_GPIO_AF_I2C, 148
 - HAL_GPIO_AF_MCPWM, 148
 - HAL_GPIO_AF_OPA, 148
 - HAL_GPIO_AF_OPA_OUT, 148
 - HAL_GPIO_AF_QEP0, 148
 - HAL_GPIO_AF_QEP1, 148
 - HAL_GPIO_AF_SIF, 148
 - HAL_GPIO_AF_SPI, 148
 - HAL_GPIO_AF_TIMER0, 148
 - HAL_GPIO_AF_TIMER1, 148
 - HAL_GPIO_AF_TIMER2, 148
 - HAL_GPIO_AF_TIMER3, 148
 - HAL_GPIO_AF_UART, 148
 - HAL_GPIO_AFx, 147
 - HAL_GPIO_Disable, 151
 - HAL_GPIO_Enable, 151
 - HAL_GPIO_GetInData, 151
 - HAL_GPIO_GetOutData, 152
 - HAL_GPIO_Invert, 152
 - HAL_GPIO_MODE_AFE, 148
 - HAL_GPIO_MODE_DEFAULT_0, 148
 - HAL_GPIO_MODE_DEFAULT_1, 148
 - HAL_GPIO_MODE_IN, 148
 - HAL_GPIO_MODE_IN_EN, 148
 - HAL_GPIO_MODE_INOUT, 148
 - HAL_GPIO_MODE_INOUT_0, 149
 - HAL_GPIO_MODE_INOUT_1, 149
 - HAL_GPIO_MODE_INUP, 148
 - HAL_GPIO_MODE_INUPOUT, 149
 - HAL_GPIO_MODE_INUPOUT_0, 149
 - HAL_GPIO_MODE_INUPOUT_1, 149
 - HAL_GPIO_MODE_NONE, 148
 - HAL_GPIO_MODE_OUT, 148
 - HAL_GPIO_MODE_OUT_0, 149
 - HAL_GPIO_MODE_OUT_1, 149
 - HAL_GPIO_MODE_OUT_EN, 148
 - HAL_GPIO_MODE_PODE_EN, 148
 - HAL_GPIO_MODE_UP_EN, 148
 - HAL_GPIO_MODE_x, 148
 - HAL_GPIO_P0_0, 149
 - HAL_GPIO_P0_1, 149
 - HAL_GPIO_P0_10, 149
 - HAL_GPIO_P0_11, 149
 - HAL_GPIO_P0_12, 149
 - HAL_GPIO_P0_13, 149
 - HAL_GPIO_P0_14, 149
 - HAL_GPIO_P0_15, 149
 - HAL_GPIO_P0_2, 149
 - HAL_GPIO_P0_3, 149
 - HAL_GPIO_P0_4, 149
 - HAL_GPIO_P0_5, 149
 - HAL_GPIO_P0_6, 149
 - HAL_GPIO_P0_7, 149
 - HAL_GPIO_P0_8, 149
 - HAL_GPIO_P0_9, 149
 - HAL_GPIO_P1_0, 149
 - HAL_GPIO_P1_1, 149
 - HAL_GPIO_P1_10, 150
 - HAL_GPIO_P1_11, 150
 - HAL_GPIO_P1_12, 150
 - HAL_GPIO_P1_13, 150
 - HAL_GPIO_P1_14, 150
 - HAL_GPIO_P1_15, 150
 - HAL_GPIO_P1_2, 149
 - HAL_GPIO_P1_3, 149
 - HAL_GPIO_P1_4, 149
 - HAL_GPIO_P1_5, 149
 - HAL_GPIO_P1_6, 149
 - HAL_GPIO_P1_7, 149
 - HAL_GPIO_P1_8, 149
 - HAL_GPIO_P1_9, 150
 - HAL_GPIO_P2_0, 150
 - HAL_GPIO_P2_1, 150
 - HAL_GPIO_P2_10, 150
 - HAL_GPIO_P2_11, 150
 - HAL_GPIO_P2_12, 150
 - HAL_GPIO_P2_13, 150
 - HAL_GPIO_P2_14, 150

- HAL_GPIO_P2.15, [150](#)
- HAL_GPIO_P2.2, [150](#)
- HAL_GPIO_P2.3, [150](#)
- HAL_GPIO_P2.4, [150](#)
- HAL_GPIO_P2.5, [150](#)
- HAL_GPIO_P2.6, [150](#)
- HAL_GPIO_P2.7, [150](#)
- HAL_GPIO_P2.8, [150](#)
- HAL_GPIO_P2.9, [150](#)
- HAL_GPIO_P3.0, [150](#)
- HAL_GPIO_P3.1, [150](#)
- HAL_GPIO_P3.10, [150](#)
- HAL_GPIO_P3.11, [150](#)
- HAL_GPIO_P3.12, [150](#)
- HAL_GPIO_P3.13, [150](#)
- HAL_GPIO_P3.14, [150](#)
- HAL_GPIO_P3.15, [150](#)
- HAL_GPIO_P3.2, [150](#)
- HAL_GPIO_P3.3, [150](#)
- HAL_GPIO_P3.4, [150](#)
- HAL_GPIO_P3.5, [150](#)
- HAL_GPIO_P3.6, [150](#)
- HAL_GPIO_P3.7, [150](#)
- HAL_GPIO_P3.8, [150](#)
- HAL_GPIO_P3.9, [150](#)
- HAL_GPIO_P4.0, [150](#)
- HAL_GPIO_P4.1, [150](#)
- HAL_GPIO_P4.10, [150](#)
- HAL_GPIO_P4.11, [151](#)
- HAL_GPIO_P4.12, [151](#)
- HAL_GPIO_P4.13, [151](#)
- HAL_GPIO_P4.14, [151](#)
- HAL_GPIO_P4.15, [151](#)
- HAL_GPIO_P4.2, [150](#)
- HAL_GPIO_P4.3, [150](#)
- HAL_GPIO_P4.4, [150](#)
- HAL_GPIO_P4.5, [150](#)
- HAL_GPIO_P4.6, [150](#)
- HAL_GPIO_P4.7, [150](#)
- HAL_GPIO_P4.8, [150](#)
- HAL_GPIO_P4.9, [150](#)
- HAL_GPIO_P5.0, [151](#)
- HAL_GPIO_P5.1, [151](#)
- HAL_GPIO_P5.10, [151](#)
- HAL_GPIO_P5.11, [151](#)
- HAL_GPIO_P5.12, [151](#)
- HAL_GPIO_P5.13, [151](#)
- HAL_GPIO_P5.14, [151](#)
- HAL_GPIO_P5.15, [151](#)
- HAL_GPIO_P5.2, [151](#)
- HAL_GPIO_P5.3, [151](#)
- HAL_GPIO_P5.4, [151](#)
- HAL_GPIO_P5.5, [151](#)
- HAL_GPIO_P5.6, [151](#)
- HAL_GPIO_P5.7, [151](#)
- HAL_GPIO_P5.8, [151](#)
- HAL_GPIO_P5.9, [151](#)
- HAL_GPIO_Px, [149](#)
- HAL_GPIO_Reset, [153](#)
- HAL_GPIO_RstReuseGpioDisable, [153](#)
- HAL_GPIO_RstReuseGpioEnable, [153](#)
- HAL_GPIO_SetAf, [153](#)
- HAL_GPIO_SetHigh, [154](#)
- HAL_GPIO_SetLow, [155](#)
- HAL_GPIO_SetMode, [155](#)
- HAL_GPIO_SwdReuseGpioDisable, [156](#)
- HAL_GPIO_SwdReuseGpioEnable, [156](#)
- HAL_GPIO_Write, [157](#)
- hal_lksmcu_hall.h
 - HAL_HALL_0, [161](#)
 - HAL_HALL_1, [161](#)
 - HAL_HALL_Disable, [161](#)
 - HAL_HALL_Enable, [161](#)
 - HAL_HALL_GetCNT, [162](#)
 - HAL_HALL_ReadHallData, [162](#)
 - HAL_HALL_ReadHallWidth, [162](#)
 - HAL_HALL_ReadTh, [163](#)
 - HAL_HALL_Reset, [163](#)
 - HAL_HALL_SetFilter, [163](#)
 - HAL_HALL_SetFilter75, [164](#)
 - HAL_HALL_SetFilteruS, [164](#)
 - HAL_HALL_SetFreq, [165](#)
 - HAL_HALL_SetGpio, [165](#)
 - HAL_HALL_SetTh, [166](#)
 - HAL_HALL_StartCounting, [166](#)
 - HAL_HALL_StopCounting, [167](#)
 - HAL_HALL_x, [161](#)
- hal_lksmcu_i2c.h
 - HAL_I2C_0, [169](#)
 - HAL_I2C_1, [169](#)
 - HAL_I2C_BusReset, [169](#)
 - HAL_I2C_Disable, [170](#)
 - HAL_I2C_Enable, [170](#)
 - HAL_I2C_GetBaud, [170](#)
 - HAL_I2C_GetClockDiv, [171](#)
 - HAL_I2C_GetClockFreq, [171](#)
 - HAL_I2C_Reset, [172](#)
 - HAL_I2C_RxData, [172](#)
 - HAL_I2C_SetBaud, [172](#)
 - HAL_I2C_SetClockDiv, [173](#)
 - HAL_I2C_SetGpioScl, [173](#)
 - HAL_I2C_SetGpioSda, [174](#)
 - HAL_I2C_TxData, [174](#)
 - HAL_I2C_x, [169](#)
- hal_lksmcu_init.h
 - HAL_INIT_DEFAULE_MCLK, [176](#)
 - HAL_INIT_DEFAULE_MCLK_MS, [176](#)
 - HAL_INIT_DEFAULE_MCLK_US, [176](#)
 - HAL_INIT_POWER_3.3V, [177](#)
 - HAL_INIT_POWER_5V, [177](#)
 - HAL_INIT_POWER_x, [176](#)
 - HAL_INIT_PowerReady, [177](#)
 - HAL_INIT_SysInit, [177](#)
- hal_lksmcu_irq.h
 - HAL_IRQ_Disable, [183](#)
 - HAL_IRQ_Enable, [184](#)

- HAL_IRQ_MODULE_ADC0, 183
- HAL_IRQ_MODULE_CAN0, 183
- HAL_IRQ_MODULE_CMP0, 183
- HAL_IRQ_MODULE_DMA0, 183
- HAL_IRQ_MODULE_DSP0, 183
- HAL_IRQ_MODULE_GPIO0, 183
- HAL_IRQ_MODULE_HALL0, 183
- HAL_IRQ_MODULE_HardFault, 183
- HAL_IRQ_MODULE_I2C0, 183
- HAL_IRQ_MODULE_INIT0, 183
- HAL_IRQ_MODULE_MCPWM0, 183
- HAL_IRQ_MODULE_NonMaskableInt, 183
- HAL_IRQ_MODULE_PendSV, 183
- HAL_IRQ_MODULE_QEP0, 183
- HAL_IRQ_MODULE_QEP1, 183
- HAL_IRQ_MODULE_SIF0, 183
- HAL_IRQ_MODULE_SPI0, 183
- HAL_IRQ_MODULE_SVCall, 183
- HAL_IRQ_MODULE_SysTick, 183
- HAL_IRQ_MODULE_TIMER0, 183
- HAL_IRQ_MODULE_TIMER1, 183
- HAL_IRQ_MODULE_TIMER2, 183
- HAL_IRQ_MODULE_TIMER3, 183
- HAL_IRQ_MODULE_UART0, 183
- HAL_IRQ_MODULE_UART1, 183
- HAL_IRQ_MODULE_WAKE0, 183
- HAL_IRQ_MODULE_x, 182
- HAL_IRQ_ModuleClear, 184
- HAL_IRQ_ModuleDisable, 184
- HAL_IRQ_ModuleEnable, 185
- HAL_IRQ_Reset, 185
- HAL_IRQHandler_ADC0, 179
- HAL_IRQHandler_CAN0, 179
- HAL_IRQHandler_CMP0, 180
- HAL_IRQHandler_DMA0, 180
- HAL_IRQHandler_DSP0, 180
- HAL_IRQHandler_GPIO0, 180
- HAL_IRQHandler_HALL0, 180
- HAL_IRQHandler_I2C0, 180
- HAL_IRQHandler_INIT0, 180
- HAL_IRQHandler_MCPWM0, 180
- HAL_IRQHandler_PWRDN0, 181
- HAL_IRQHandler_QEP0, 181
- HAL_IRQHandler_QEP1, 181
- HAL_IRQHandler_SIF0, 181
- HAL_IRQHandler_SPI0, 181
- HAL_IRQHandler_TIMER0, 181
- HAL_IRQHandler_TIMER1, 181
- HAL_IRQHandler_TIMER2, 182
- HAL_IRQHandler_TIMER3, 182
- HAL_IRQHandler_UART0, 182
- HAL_IRQHandler_UART1, 182
- HAL_IRQHandler_WAKE0, 182
- hal_lksmcu_irqfun.h
 - HAL_ADC_IrqHardErr, 199
 - HAL_ADC_IrqHardErrClear, 199
 - HAL_ADC_IrqHardErrDisable, 199
 - HAL_ADC_IrqHardErrEnable, 200
 - HAL_ADC_IrqHardErrFlg, 200
 - HAL_ADC_IrqIdleSamp, 200
 - HAL_ADC_IrqIdleSampClear, 200
 - HAL_ADC_IrqIdleSampDisable, 201
 - HAL_ADC_IrqIdleSampEnable, 201
 - HAL_ADC_IrqIdleSampFlg, 201
 - HAL_ADC_IrqSamp1, 201
 - HAL_ADC_IrqSamp1Clear, 201
 - HAL_ADC_IrqSamp1Disable, 202
 - HAL_ADC_IrqSamp1Enable, 202
 - HAL_ADC_IrqSamp1Flg, 202
 - HAL_ADC_IrqSamp2, 203
 - HAL_ADC_IrqSamp2Clear, 203
 - HAL_ADC_IrqSamp2Disable, 203
 - HAL_ADC_IrqSamp2Enable, 204
 - HAL_ADC_IrqSamp2Flg, 204
 - HAL_ADC_IrqSamp3, 204
 - HAL_ADC_IrqSamp3Clear, 205
 - HAL_ADC_IrqSamp3Disable, 205
 - HAL_ADC_IrqSamp3Enable, 205
 - HAL_ADC_IrqSamp3Flg, 205
 - HAL_ADC_IrqSamp4, 206
 - HAL_ADC_IrqSamp4Clear, 206
 - HAL_ADC_IrqSamp4Disable, 206
 - HAL_ADC_IrqSamp4Enable, 207
 - HAL_ADC_IrqSamp4Flg, 207
 - HAL_ADC_IrqSoftErr, 207
 - HAL_ADC_IrqSoftErrClear, 208
 - HAL_ADC_IrqSoftErrDisable, 208
 - HAL_ADC_IrqSoftErrEnable, 208
 - HAL_ADC_IrqSoftErrFlg, 209
 - HAL_ADC_IrqWatchdog, 209
 - HAL_ADC_IrqWatchdog1, 209
 - HAL_ADC_IrqWatchdog1Clear, 209
 - HAL_ADC_IrqWatchdog1Disable, 209
 - HAL_ADC_IrqWatchdog1Enable, 210
 - HAL_ADC_IrqWatchdog1Flg, 210
 - HAL_ADC_IrqWatchdogClear, 210
 - HAL_ADC_IrqWatchdogDisable, 210
 - HAL_ADC_IrqWatchdogEnable, 210
 - HAL_ADC_IrqWatchdogFlg, 211
 - HAL_CAN_IrqAll, 211
 - HAL_CAN_IrqAllClear, 211
 - HAL_CAN_IrqAllDisable, 212
 - HAL_CAN_IrqAllEnable, 212
 - HAL_CAN_IrqAllFlg, 212
 - HAL_CAN_IrqRxDone, 213
 - HAL_CAN_IrqRxDoneClear, 213
 - HAL_CAN_IrqRxDoneDisable, 213
 - HAL_CAN_IrqRxDoneEnable, 214
 - HAL_CAN_IrqRxDoneFlg, 214
 - HAL_CAN_IrqTxDone, 214
 - HAL_CAN_IrqTxDoneClear, 215
 - HAL_CAN_IrqTxDoneDisable, 215
 - HAL_CAN_IrqTxDoneEnable, 215
 - HAL_CAN_IrqTxDoneFlg, 216
 - HAL_CMP_IrqCompare, 216
 - HAL_CMP_IrqCompareClear, 216

HAL_CMP_IrqCompareDisable, 217
HAL_CMP_IrqCompareEnable, 217
HAL_CMP_IrqCompareFlg, 217
HAL_DMA_IrqDone, 218
HAL_DMA_IrqDoneClear, 218
HAL_DMA_IrqDoneDisable, 218
HAL_DMA_IrqDoneEnable, 219
HAL_DMA_IrqDoneFlg, 219
HAL_DMA_IrqDoneTrig, 219
HAL_DMA_IrqErr, 219
HAL_DMA_IrqErrClear, 220
HAL_DMA_IrqErrDisable, 220
HAL_DMA_IrqErrEnable, 220
HAL_DMA_IrqErrFlg, 221
HAL_DMA_IrqErrTrig, 221
HAL_GPIO_Irq, 221
HAL_GPIO_IrqClear, 221
HAL_GPIO_IrqDisable, 222
HAL_GPIO_IrqEnable, 222
HAL_GPIO_IrqFlg, 222
HAL_GPIO_IrqTrig, 223
HAL_HALL_IrqChange, 223
HAL_HALL_IrqChangeClear, 223
HAL_HALL_IrqChangeDisable, 223
HAL_HALL_IrqChangeEnable, 224
HAL_HALL_IrqChangeFlg, 224
HAL_HALL_IrqOver, 224
HAL_HALL_IrqOverClear, 225
HAL_HALL_IrqOverDisable, 225
HAL_HALL_IrqOverEnable, 225
HAL_HALL_IrqOverFlg, 226
HAL_I2C_IrqBusErr, 226
HAL_I2C_IrqBusErrClear, 226
HAL_I2C_IrqBusErrDisable, 227
HAL_I2C_IrqBusErrEnable, 227
HAL_I2C_IrqBusErrFlg, 227
HAL_I2C_IrqDone, 228
HAL_I2C_IrqDoneClear, 228
HAL_I2C_IrqDoneDisable, 228
HAL_I2C_IrqDoneEnable, 229
HAL_I2C_IrqDoneFlg, 229
HAL_I2C_IrqStop, 229
HAL_I2C_IrqStopClear, 230
HAL_I2C_IrqStopDisable, 230
HAL_I2C_IrqStopEnable, 230
HAL_I2C_IrqStopFlg, 231
HAL_MCPWM_IrqFail, 231
HAL_MCPWM_IrqFail0, 231
HAL_MCPWM_IrqFail0Clear, 232
HAL_MCPWM_IrqFail0Disable, 232
HAL_MCPWM_IrqFail0Enable, 232
HAL_MCPWM_IrqFail0Flg, 233
HAL_MCPWM_IrqFail1, 233
HAL_MCPWM_IrqFail1Clear, 233
HAL_MCPWM_IrqFail1Disable, 234
HAL_MCPWM_IrqFail1Enable, 234
HAL_MCPWM_IrqFail1Flg, 234
HAL_MCPWM_IrqFailClear, 234
HAL_MCPWM_IrqFailDisable, 235
HAL_MCPWM_IrqFailEnable, 235
HAL_MCPWM_IrqFailFlg, 235
HAL_MCPWM_IrqOverflow, 236
HAL_MCPWM_IrqOverflowClear, 236
HAL_MCPWM_IrqOverflowDisable, 236
HAL_MCPWM_IrqOverflowEnable, 237
HAL_MCPWM_IrqOverflowFlg, 237
HAL_MCPWM_IrqShort, 237
HAL_MCPWM_IrqShortClear, 238
HAL_MCPWM_IrqShortDisable, 238
HAL_MCPWM_IrqShortEnable, 238
HAL_MCPWM_IrqShortFlg, 238
HAL_MCPWM_IrqTH00, 239
HAL_MCPWM_IrqTH00Clear, 239
HAL_MCPWM_IrqTH00Disable, 239
HAL_MCPWM_IrqTH00Enable, 240
HAL_MCPWM_IrqTH00Flg, 240
HAL_MCPWM_IrqTH01, 240
HAL_MCPWM_IrqTH01Clear, 241
HAL_MCPWM_IrqTH01Disable, 241
HAL_MCPWM_IrqTH01Enable, 241
HAL_MCPWM_IrqTH01Flg, 242
HAL_MCPWM_IrqTH10, 242
HAL_MCPWM_IrqTH10Clear, 242
HAL_MCPWM_IrqTH10Disable, 243
HAL_MCPWM_IrqTH10Enable, 243
HAL_MCPWM_IrqTH10Flg, 243
HAL_MCPWM_IrqTH11, 243
HAL_MCPWM_IrqTH11Clear, 244
HAL_MCPWM_IrqTH11Disable, 244
HAL_MCPWM_IrqTH11Enable, 244
HAL_MCPWM_IrqTH11Flg, 245
HAL_MCPWM_IrqTH20, 245
HAL_MCPWM_IrqTH20Clear, 245
HAL_MCPWM_IrqTH20Disable, 246
HAL_MCPWM_IrqTH20Enable, 246
HAL_MCPWM_IrqTH20Flg, 246
HAL_MCPWM_IrqTH21, 247
HAL_MCPWM_IrqTH21Clear, 247
HAL_MCPWM_IrqTH21Disable, 247
HAL_MCPWM_IrqTH21Enable, 248
HAL_MCPWM_IrqTH21Flg, 248
HAL_MCPWM_IrqTH30, 248
HAL_MCPWM_IrqTH30Clear, 249
HAL_MCPWM_IrqTH30Disable, 249
HAL_MCPWM_IrqTH30Enable, 249
HAL_MCPWM_IrqTH30Flg, 249
HAL_MCPWM_IrqTH31, 250
HAL_MCPWM_IrqTH31Clear, 250
HAL_MCPWM_IrqTH31Disable, 250
HAL_MCPWM_IrqTH31Enable, 251
HAL_MCPWM_IrqTH31Flg, 251
HAL_MCPWM_IrqTH40, 251
HAL_MCPWM_IrqTH40Clear, 251
HAL_MCPWM_IrqTH40Disable, 251
HAL_MCPWM_IrqTH40Enable, 252
HAL_MCPWM_IrqTH40Flg, 252

HAL_MCPWM_IrqTH41, 252
HAL_MCPWM_IrqTH41Clear, 252
HAL_MCPWM_IrqTH41Disable, 252
HAL_MCPWM_IrqTH41Enable, 252
HAL_MCPWM_IrqTH41Flg, 252
HAL_MCPWM_IrqTH50, 253
HAL_MCPWM_IrqTH50Clear, 253
HAL_MCPWM_IrqTH50Disable, 253
HAL_MCPWM_IrqTH50Enable, 253
HAL_MCPWM_IrqTH50Flg, 253
HAL_MCPWM_IrqTH51, 253
HAL_MCPWM_IrqTH51Clear, 253
HAL_MCPWM_IrqTH51Disable, 254
HAL_MCPWM_IrqTH51Enable, 254
HAL_MCPWM_IrqTH51Flg, 254
HAL_MCPWM_IrqTmr0, 254
HAL_MCPWM_IrqTmr0Clear, 254
HAL_MCPWM_IrqTmr0Disable, 255
HAL_MCPWM_IrqTmr0Enable, 255
HAL_MCPWM_IrqTmr0Flg, 255
HAL_MCPWM_IrqTmr1, 256
HAL_MCPWM_IrqTmr1Clear, 256
HAL_MCPWM_IrqTmr1Disable, 256
HAL_MCPWM_IrqTmr1Enable, 257
HAL_MCPWM_IrqTmr1Flg, 257
HAL_MCPWM_IrqTmr2, 257
HAL_MCPWM_IrqTmr2Clear, 258
HAL_MCPWM_IrqTmr2Disable, 258
HAL_MCPWM_IrqTmr2Enable, 258
HAL_MCPWM_IrqTmr2Flg, 258
HAL_MCPWM_IrqTmr3, 259
HAL_MCPWM_IrqTmr3Clear, 259
HAL_MCPWM_IrqTmr3Disable, 259
HAL_MCPWM_IrqTmr3Enable, 260
HAL_MCPWM_IrqTmr3Flg, 260
HAL_MCPWM_IrqUpdate, 260
HAL_MCPWM_IrqUpdateClear, 261
HAL_MCPWM_IrqUpdateDisable, 261
HAL_MCPWM_IrqUpdateEnable, 261
HAL_MCPWM_IrqUpdateFlg, 262
HAL_MCPWM_IrqZeroCross, 262
HAL_MCPWM_IrqZeroCrossClear, 262
HAL_MCPWM_IrqZeroCrossDisable, 263
HAL_MCPWM_IrqZeroCrossEnable, 263
HAL_MCPWM_IrqZeroCrossFlg, 263
HAL_SPI_IrqDone, 263
HAL_SPI_IrqDoneClear, 264
HAL_SPI_IrqDoneDisable, 264
HAL_SPI_IrqDoneEnable, 264
HAL_SPI_IrqDoneFlg, 265
HAL_SPI_IrqErr, 265
HAL_SPI_IrqErrClear, 265
HAL_SPI_IrqErrDisable, 266
HAL_SPI_IrqErrEnable, 266
HAL_SPI_IrqErrFlg, 266
HAL_SPI_IrqOverflow, 267
HAL_SPI_IrqOverflowClear, 267
HAL_SPI_IrqOverflowDisable, 267
HAL_SPI_IrqOverflowEnable, 268
HAL_SPI_IrqOverflowFlg, 268
HAL_SW_Irq, 268
HAL_SW_IrqClear, 268
HAL_SW_IrqDisable, 269
HAL_SW_IrqEnable, 269
HAL_SW_IrqFlg, 269
HAL_SW_IrqTrig, 269
HAL_TIMER_IrqCompare0, 269
HAL_TIMER_IrqCompare0Clear, 270
HAL_TIMER_IrqCompare0Disable, 270
HAL_TIMER_IrqCompare0Enable, 270
HAL_TIMER_IrqCompare0Flg, 270
HAL_TIMER_IrqCompare1, 271
HAL_TIMER_IrqCompare1Clear, 271
HAL_TIMER_IrqCompare1Disable, 272
HAL_TIMER_IrqCompare1Enable, 272
HAL_TIMER_IrqCompare1Flg, 272
HAL_TIMER_IrqOverflow, 272
HAL_TIMER_IrqOverflowClear, 273
HAL_TIMER_IrqOverflowDisable, 273
HAL_TIMER_IrqOverflowEnable, 273
HAL_TIMER_IrqOverflowFlg, 274
HAL_UART_IrqAbd, 274
HAL_UART_IrqAbdClear, 274
HAL_UART_IrqAbdDisable, 274
HAL_UART_IrqAbdEnable, 275
HAL_UART_IrqAbdFlg, 275
HAL_UART_IrqCheckErr, 275
HAL_UART_IrqCheckErrClear, 275
HAL_UART_IrqCheckErrDisable, 276
HAL_UART_IrqCheckErrEnable, 276
HAL_UART_IrqCheckErrFlg, 276
HAL_UART_IrqIdle, 277
HAL_UART_IrqIdleClear, 277
HAL_UART_IrqIdleDisable, 277
HAL_UART_IrqIdleEnable, 277
HAL_UART_IrqIdleFlg, 277
HAL_UART_IrqLbd, 277
HAL_UART_IrqLbdClear, 277
HAL_UART_IrqLbdDisable, 278
HAL_UART_IrqLbdEnable, 278
HAL_UART_IrqLbdFlg, 278
HAL_UART_IrqRxDone, 278
HAL_UART_IrqRxDoneClear, 278
HAL_UART_IrqRxDoneDisable, 279
HAL_UART_IrqRxDoneEnable, 279
HAL_UART_IrqRxDoneFlg, 279
HAL_UART_IrqRxOv, 280
HAL_UART_IrqRxOvClear, 280
HAL_UART_IrqRxOvDisable, 280
HAL_UART_IrqRxOvEnable, 280
HAL_UART_IrqRxOvFlg, 280
HAL_UART_IrqStopErr, 280
HAL_UART_IrqStopErrClear, 281
HAL_UART_IrqStopErrDisable, 281
HAL_UART_IrqStopErrEnable, 281
HAL_UART_IrqStopErrFlg, 282

- HAL_UART_IrqTxBufEmpty, 282
- HAL_UART_IrqTxBufEmptyClear, 282
- HAL_UART_IrqTxBufEmptyDisable, 283
- HAL_UART_IrqTxBufEmptyEnable, 283
- HAL_UART_IrqTxBufEmptyFlg, 283
- HAL_UART_IrqTxDone, 284
- HAL_UART_IrqTxDoneClear, 284
- HAL_UART_IrqTxDoneDisable, 284
- HAL_UART_IrqTxDoneEnable, 285
- HAL_UART_IrqTxDoneFlg, 285
- HAL_UART_IrqTxOv, 285
- HAL_UART_IrqTxOvClear, 285
- HAL_UART_IrqTxOvDisable, 286
- HAL_UART_IrqTxOvEnable, 286
- HAL_UART_IrqTxOvFlg, 286
- hal_ksmcu_iwdg.h
 - HAL_IWDG_0, 292
 - HAL_IWDG_Disable, 292
 - HAL_IWDG_Enable, 292
 - HAL_IWDG_Feed, 292
 - HAL_IWDG_Reset, 293
 - HAL_IWDG_SetResetTimemS, 293
 - HAL_IWDG_x, 291
- hal_ksmcu_mcpwm.h
 - HAL_MCPWM_0, 300
 - HAL_MCPWM_DEFAULT_OUTPUT_STATE_HIGH, 297
 - HAL_MCPWM_DEFAULT_OUTPUT_STATE_LOW, 297
 - HAL_MCPWM_DEFAULT_OUTPUT_STATE_OFF, 297
 - HAL_MCPWM_DEFAULT_OUTPUT_STATE_x, 297
 - HAL_MCPWM_Disable, 300
 - HAL_MCPWM_Enable, 300
 - HAL_MCPWM_FAIL_0, 298
 - HAL_MCPWM_FAIL_1, 298
 - HAL_MCPWM_FAIL_IN_CMP0_HIGH, 298
 - HAL_MCPWM_FAIL_IN_CMP0_LOW, 298
 - HAL_MCPWM_FAIL_IN_CMP1_HIGH, 298
 - HAL_MCPWM_FAIL_IN_CMP1_LOW, 298
 - HAL_MCPWM_FAIL_IN_IO_HIGH, 297
 - HAL_MCPWM_FAIL_IN_IO_LOW, 297
 - HAL_MCPWM_FAIL_IN_x, 297
 - HAL_MCPWM_FAIL_x, 298
 - HAL_MCPWM_GetClockDiv, 301
 - HAL_MCPWM_GetClockFreq, 301
 - HAL_MCPWM_GetFreq, 302
 - HAL_MCPWM_GetThCnt, 302
 - HAL_MCPWM_OUTPUT_STATE_HIGH, 298
 - HAL_MCPWM_OUTPUT_STATE_HIGH_PWM, 298
 - HAL_MCPWM_OUTPUT_STATE_LOW, 298
 - HAL_MCPWM_OUTPUT_STATE_LOW_PWM, 298
 - HAL_MCPWM_OUTPUT_STATE_OFF, 298
 - HAL_MCPWM_OUTPUT_STATE_PWM, 298
 - HAL_MCPWM_OUTPUT_STATE_x, 298
 - HAL_MCPWM_POLARITY_PH_NH, 299
 - HAL_MCPWM_POLARITY_PH_NL, 299
 - HAL_MCPWM_POLARITY_PL_NH, 299
 - HAL_MCPWM_POLARITY_PL_NL, 299
 - HAL_MCPWM_POLARITY_x, 298
 - HAL_MCPWM_PORT_0, 299
 - HAL_MCPWM_PORT_1, 299
 - HAL_MCPWM_PORT_2, 299
 - HAL_MCPWM_PORT_3, 299
 - HAL_MCPWM_PORT_x, 299
 - HAL_MCPWM_Reset, 303
 - HAL_MCPWM_SetAdcTrig, 303
 - HAL_MCPWM_SetAdcTrigQ15, 304
 - HAL_MCPWM_SetClockDiv, 304
 - HAL_MCPWM_SetDead, 304
 - HAL_MCPWM_SetDefaultState, 305
 - HAL_MCPWM_SetDrivePolarity, 306
 - HAL_MCPWM_SetDrivePolarityPort, 306
 - HAL_MCPWM_SetFailFilter, 307
 - HAL_MCPWM_SetFailSignal, 307
 - HAL_MCPWM_SetFreq, 308
 - HAL_MCPWM_SetGpioFail, 308
 - HAL_MCPWM_SetGpioOut, 309
 - HAL_MCPWM_SetOutputDisable, 309
 - HAL_MCPWM_SetOutputEnable, 310
 - HAL_MCPWM_SetOutState, 310
 - HAL_MCPWM_SetPwm, 311
 - HAL_MCPWM_SetPwmQ15, 311
 - HAL_MCPWM_SetRemap, 312
 - HAL_MCPWM_SetSwap, 312
 - HAL_MCPWM_SetSwapPort, 313
 - HAL_MCPWM_SetThCnt, 313
 - HAL_MCPWM_SetUpdateMode, 314
 - HAL_MCPWM_StartCounting, 314
 - HAL_MCPWM_StartCountingSync, 315
 - HAL_MCPWM_StopCounting, 315
 - HAL_MCPWM_SWAP_DISABLE, 299
 - HAL_MCPWM_SWAP_ENABLE, 299
 - HAL_MCPWM_SWAP_x, 299
 - HAL_MCPWM_UPDATE_T0, 300
 - HAL_MCPWM_UPDATE_T0_AND_T1, 300
 - HAL_MCPWM_UPDATE_T1, 300
 - HAL_MCPWM_UPDATE_x, 299
 - HAL_MCPWM_x, 300
- hal_ksmcu_opa.h
 - HAL_OPA_0, 320
 - HAL_OPA_0B, 320
 - HAL_OPA_1, 320
 - HAL_OPA_2, 320
 - HAL_OPA_3, 320
 - HAL_OPA_4, 320
 - HAL_OPA_5, 320
 - HAL_OPA_Disable, 320
 - HAL_OPA_Enable, 320
 - HAL_OPA_GAIN_170_40_4, 319
 - HAL_OPA_GAIN_180_30_4, 319
 - HAL_OPA_GAIN_190_20_4, 319
 - HAL_OPA_GAIN_200_10_4, 319
 - HAL_OPA_GAIN_HIGH, 319
 - HAL_OPA_GAIN_LOW, 319
 - HAL_OPA_GAIN_MEDIUM, 319

- HAL_OPA_GAIN_x, 319
- HAL_OPA_OPAX_2_ADCCHN, 319
- HAL_OPA_Out2GpioDisable, 320
- HAL_OPA_Out2GpioEnable, 321
- HAL_OPA_READ_GAIN, 319
- HAL_OPA_Reset, 321
- HAL_OPA_SetGain, 321
- HAL_OPA_SetGpioIn, 321
- HAL_OPA_SetGpioOut, 322
- HAL_OPA_x, 320
- hal_lksmcu_qep.h
 - HAL_QEP_0, 325
 - HAL_QEP_1, 325
 - HAL_QEP_2, 325
 - HAL_QEP_3, 325
 - HAL_QEP_Disable, 325
 - HAL_QEP_Enable, 326
 - HAL_QEP_GetClockDiv, 326
 - HAL_QEP_GetClockFreq, 327
 - HAL_QEP_MODE_DUAL_PULSE, 325
 - HAL_QEP_MODE_QUADRATURE, 325
 - HAL_QEP_MODE_SYMBOL_PULSE, 325
 - HAL_QEP_MODE_x, 325
 - HAL_QEP_ReadCnt, 327
 - HAL_QEP_Reset, 327
 - HAL_QEP_ResetCnt, 328
 - HAL_QEP_SetClockDiv, 328
 - HAL_QEP_SetGpio, 328
 - HAL_QEP_SetMode, 329
 - HAL_QEP_SetTh, 329
 - HAL_QEP_x, 325
- hal_lksmcu_reg.h
 - HAL_REG_IF_WRITE, 331
 - HAL_REG_READ, 331
 - HAL_REG_READ_ADDR, 331
 - HAL_REG_READ_BIT, 331
 - HAL_REG_READ_BIT_MASK_POS, 331
 - HAL_REG_RESET, 331
 - HAL_REG_SET, 332
 - HAL_REG_SYS_LOCK, 332
 - HAL_REG_SYS_UNLOCK, 332
 - HAL_REG_WRITE, 332
 - HAL_REG_WRITE_BIT, 332
 - HAL_REG_WRITE_BIT_MASK_POS, 332
- hal_lksmcu_reset.h
 - HAL_RESET_ResetCore, 333
 - HAL_RESET_ResetGlobal, 333
 - HAL_RESET_ResetRegister, 334
- hal_lksmcu_spi.h
 - HAL_SPI_0, 337
 - HAL_SPI_1, 337
 - HAL_SPI_Disable, 337
 - HAL_SPI_Enable, 337
 - HAL_SPI_GetBaud, 338
 - HAL_SPI_GetClockDiv, 338
 - HAL_SPI_GetClockFreq, 338
 - HAL_SPI_MODE_MASTER_LSB_00, 336
 - HAL_SPI_MODE_MASTER_LSB_01, 336
 - HAL_SPI_MODE_MASTER_LSB_10, 336
 - HAL_SPI_MODE_MASTER_LSB_11, 336
 - HAL_SPI_MODE_MASTER_MSB_00, 336
 - HAL_SPI_MODE_MASTER_MSB_01, 336
 - HAL_SPI_MODE_MASTER_MSB_10, 337
 - HAL_SPI_MODE_MASTER_MSB_11, 337
 - HAL_SPI_MODE_SLAVE_LSB_00, 337
 - HAL_SPI_MODE_SLAVE_LSB_01, 337
 - HAL_SPI_MODE_SLAVE_LSB_10, 337
 - HAL_SPI_MODE_SLAVE_LSB_11, 337
 - HAL_SPI_MODE_SLAVE_MSB_00, 337
 - HAL_SPI_MODE_SLAVE_MSB_01, 337
 - HAL_SPI_MODE_SLAVE_MSB_10, 337
 - HAL_SPI_MODE_SLAVE_MSB_11, 337
 - HAL_SPI_MODE_x, 336
 - HAL_SPI_Reset, 339
 - HAL_SPI_RxData, 339
 - HAL_SPI_RxDataWidth16, 340
 - HAL_SPI_RxTxData, 341
 - HAL_SPI_RxTxDataWidth16, 341
 - HAL_SPI_SetBaud, 342
 - HAL_SPI_SetClockDiv, 343
 - HAL_SPI_SetGpioClk, 343
 - HAL_SPI_SetGpioCs, 344
 - HAL_SPI_SetGpioDi, 345
 - HAL_SPI_SetGpioDo, 345
 - HAL_SPI_SetGpioMiso, 346
 - HAL_SPI_SetGpioMosi, 346
 - HAL_SPI_SetMode, 347
 - HAL_SPI_TxData, 347
 - HAL_SPI_TxDataWidth16, 348
 - HAL_SPI_x, 337
- hal_lksmcu_timer.h
 - HAL_TIMER_0, 352
 - HAL_TIMER_1, 352
 - HAL_TIMER_2, 352
 - HAL_TIMER_3, 352
 - HAL_TIMER_4, 352
 - HAL_TIMER_Disable, 352
 - HAL_TIMER_Enable, 352
 - HAL_TIMER_GetClockDiv, 352
 - HAL_TIMER_GetClockFreq, 353
 - HAL_TIMER_GetFreq, 353
 - HAL_TIMER_GetThCnt, 353
 - HAL_TIMER_MODE_COUNT, 351
 - HAL_TIMER_MODE_PWMIN, 351
 - HAL_TIMER_MODE_PWMOUT, 351
 - HAL_TIMER_MODE_x, 351
 - HAL_TIMER_Reset, 354
 - HAL_TIMER_SetClockDiv, 354
 - HAL_TIMER_SetFreq, 354
 - HAL_TIMER_SetGpioIn, 355
 - HAL_TIMER_SetGpioOut, 356
 - HAL_TIMER_SetPeriodCount, 356
 - HAL_TIMER_SetPeriodmS, 357
 - HAL_TIMER_SetPerioduS, 357
 - HAL_TIMER_SetPwm, 358
 - HAL_TIMER_SetPwmQ15, 358

- HAL_TIMER_SetWorkMode, 358
- HAL_TIMER_StartCounting, 359
- HAL_TIMER_StopCounting, 359
- HAL_TIMER_x, 351
- hal_iksmcu_tmp.h
 - HAL_TMP_0, 362
 - HAL_TMP_1, 362
 - HAL_TMP_Adc2Temperature, 362
 - HAL_TMP_Adc2Temperature01, 363
 - HAL_TMP_Disable, 363
 - HAL_TMP_Enable, 364
 - HAL_TMP_GetTemperature, 364
 - HAL_TMP_GetTemperature01, 365
 - HAL_TMP_Reset, 366
 - HAL_TMP_x, 362
- hal_iksmcu_trim.h
 - HAL_TRIM_Read, 367
 - HAL_TRIM_ReadId, 368
 - HAL_TRIM_ReadIdCrc32, 369
- hal_iksmcu_uart.h
 - HAL_UART_0, 372
 - HAL_UART_1, 372
 - HAL_UART_Disable, 372
 - HAL_UART_Enable, 372
 - HAL_UART_GetBaud, 372
 - HAL_UART_GetClockDiv, 372
 - HAL_UART_GetClockFreq, 373
 - HAL_UART_MODE_LSB_STOP1_EVEN, 371
 - HAL_UART_MODE_LSB_STOP1_NONE, 371
 - HAL_UART_MODE_LSB_STOP1_ODD, 371
 - HAL_UART_MODE_LSB_STOP2_EVEN, 371
 - HAL_UART_MODE_LSB_STOP2_NONE, 371
 - HAL_UART_MODE_LSB_STOP2_ODD, 371
 - HAL_UART_MODE_MSB_STOP1_EVEN, 371
 - HAL_UART_MODE_MSB_STOP1_NONE, 371
 - HAL_UART_MODE_MSB_STOP1_ODD, 371
 - HAL_UART_MODE_MSB_STOP2_EVEN, 371
 - HAL_UART_MODE_MSB_STOP2_NONE, 371
 - HAL_UART_MODE_MSB_STOP2_ODD, 371
 - HAL_UART_MODE_x, 371
 - HAL_UART_POL_RX_NEG_TX_NEG, 371
 - HAL_UART_POL_RX_NEG_TX_POS, 371
 - HAL_UART_POL_RX_POS_TX_NEG, 371
 - HAL_UART_POL_RX_POS_TX_POS, 371
 - HAL_UART_POL_x, 371
 - HAL_UART_ReadBuffer, 373
 - HAL_UART_Reset, 373
 - HAL_UART_SetBaud, 374
 - HAL_UART_SetClockDiv, 374
 - HAL_UART_SetGpioRx, 374
 - HAL_UART_SetGpioTx, 374
 - HAL_UART_SetMode, 375
 - HAL_UART_SetPolarity, 375
 - HAL_UART_TxData, 376
 - HAL_UART_WriteBuffer, 376
 - HAL_UART_x, 372
- hal_iksmcu_wake.h
 - HAL_WAKE_DeepSleepIoWakeUp, 378
 - HAL_WAKE_DeepSleepTimeWakeUp, 379
 - HAL_WAKE_UP_IO_P0_0_High, 378
 - HAL_WAKE_UP_IO_P0_0_Low, 378
 - HAL_WAKE_UP_IO_P0_1_High, 378
 - HAL_WAKE_UP_IO_P0_1_Low, 378
 - HAL_WAKE_UP_IO_P1_0_High, 378
 - HAL_WAKE_UP_IO_P1_0_Low, 378
 - HAL_WAKE_UP_IO_P1_1_High, 378
 - HAL_WAKE_UP_IO_P1_1_Low, 378
 - HAL_WAKE_UP_IO_x, 378
- HAL_MCPWM
 - hal_iks32mc08x_reg.c, 682
 - hal_iks32mc08x_reg.h, 63
- HAL_MCPWM_0
 - hal_iksmcu_mcpwm.h, 300
- HAL_MCPWM_DEFAULT_OUTPUT_STATE_HIGH
 - hal_iksmcu_mcpwm.h, 297
- HAL_MCPWM_DEFAULT_OUTPUT_STATE_LOW
 - hal_iksmcu_mcpwm.h, 297
- HAL_MCPWM_DEFAULT_OUTPUT_STATE_OFF
 - hal_iksmcu_mcpwm.h, 297
- HAL_MCPWM_DEFAULT_OUTPUT_STATE_x
 - hal_iksmcu_mcpwm.h, 297
- HAL_MCPWM_Disable
 - hal_iks32mc08x_mcpwm.c, 644
 - hal_iksmcu_mcpwm.h, 300
- HAL_MCPWM_Enable
 - hal_iks32mc08x_mcpwm.c, 645
 - hal_iksmcu_mcpwm.h, 300
- HAL_MCPWM_FAIL_0
 - hal_iksmcu_mcpwm.h, 298
- HAL_MCPWM_FAIL_1
 - hal_iksmcu_mcpwm.h, 298
- HAL_MCPWM_FAIL_IN_CMP0_HIGH
 - hal_iksmcu_mcpwm.h, 298
- HAL_MCPWM_FAIL_IN_CMP0_LOW
 - hal_iksmcu_mcpwm.h, 298
- HAL_MCPWM_FAIL_IN_CMP1_HIGH
 - hal_iksmcu_mcpwm.h, 298
- HAL_MCPWM_FAIL_IN_CMP1_LOW
 - hal_iksmcu_mcpwm.h, 298
- HAL_MCPWM_FAIL_IN_IO_HIGH
 - hal_iksmcu_mcpwm.h, 297
- HAL_MCPWM_FAIL_IN_IO_LOW
 - hal_iksmcu_mcpwm.h, 297
- HAL_MCPWM_FAIL_IN_x
 - hal_iksmcu_mcpwm.h, 297
- HAL_MCPWM_FAIL_x
 - hal_iksmcu_mcpwm.h, 298
- HAL_MCPWM_GetClockDiv
 - hal_iks32mc08x_mcpwm.c, 645
 - hal_iksmcu_mcpwm.h, 301
- HAL_MCPWM_GetClockFreq
 - hal_iks32mc08x_mcpwm.c, 645
 - hal_iksmcu_mcpwm.h, 301
- HAL_MCPWM_GetFreq
 - hal_iks32mc08x_mcpwm.c, 646

hal_lksmcu_mcpwm.h, 302

HAL_MCPWM_GetThCnt
hal_lks32mc08x_mcpwm.c, 647
hal_lksmcu_mcpwm.h, 302

HAL_MCPWM_IrqFail
hal_lks32mc08x_irqfun.c, 549
hal_lksmcu_irqfun.h, 231

HAL_MCPWM_IrqFail0
hal_lks32mc08x_irqfun.c, 549
hal_lksmcu_irqfun.h, 231

HAL_MCPWM_IrqFail0Clear
hal_lks32mc08x_irqfun.c, 549
hal_lksmcu_irqfun.h, 232

HAL_MCPWM_IrqFail0Disable
hal_lks32mc08x_irqfun.c, 550
hal_lksmcu_irqfun.h, 232

HAL_MCPWM_IrqFail0Enable
hal_lks32mc08x_irqfun.c, 550
hal_lksmcu_irqfun.h, 232

HAL_MCPWM_IrqFail0Flg
hal_lks32mc08x_irqfun.c, 550
hal_lksmcu_irqfun.h, 233

HAL_MCPWM_IrqFail1
hal_lks32mc08x_irqfun.c, 551
hal_lksmcu_irqfun.h, 233

HAL_MCPWM_IrqFail1Clear
hal_lks32mc08x_irqfun.c, 551
hal_lksmcu_irqfun.h, 233

HAL_MCPWM_IrqFail1Disable
hal_lks32mc08x_irqfun.c, 551
hal_lksmcu_irqfun.h, 234

HAL_MCPWM_IrqFail1Enable
hal_lks32mc08x_irqfun.c, 552
hal_lksmcu_irqfun.h, 234

HAL_MCPWM_IrqFail1Flg
hal_lks32mc08x_irqfun.c, 552
hal_lksmcu_irqfun.h, 234

HAL_MCPWM_IrqFailClear
hal_lks32mc08x_irqfun.c, 552
hal_lksmcu_irqfun.h, 234

HAL_MCPWM_IrqFailDisable
hal_lks32mc08x_irqfun.c, 552
hal_lksmcu_irqfun.h, 235

HAL_MCPWM_IrqFailEnable
hal_lks32mc08x_irqfun.c, 553
hal_lksmcu_irqfun.h, 235

HAL_MCPWM_IrqFailFlg
hal_lks32mc08x_irqfun.c, 553
hal_lksmcu_irqfun.h, 235

HAL_MCPWM_IrqOverflow
hal_lks32mc08x_irqfun.c, 553
hal_lksmcu_irqfun.h, 236

HAL_MCPWM_IrqOverflowClear
hal_lks32mc08x_irqfun.c, 554
hal_lksmcu_irqfun.h, 236

HAL_MCPWM_IrqOverflowDisable
hal_lks32mc08x_irqfun.c, 554
hal_lksmcu_irqfun.h, 236

HAL_MCPWM_IrqOverflowEnable
hal_lks32mc08x_irqfun.c, 554
hal_lksmcu_irqfun.h, 237

HAL_MCPWM_IrqOverflowFlg
hal_lks32mc08x_irqfun.c, 555
hal_lksmcu_irqfun.h, 237

HAL_MCPWM_IrqShort
hal_lks32mc08x_irqfun.c, 555
hal_lksmcu_irqfun.h, 237

HAL_MCPWM_IrqShortClear
hal_lks32mc08x_irqfun.c, 555
hal_lksmcu_irqfun.h, 238

HAL_MCPWM_IrqShortDisable
hal_lks32mc08x_irqfun.c, 556
hal_lksmcu_irqfun.h, 238

HAL_MCPWM_IrqShortEnable
hal_lks32mc08x_irqfun.c, 556
hal_lksmcu_irqfun.h, 238

HAL_MCPWM_IrqShortFlg
hal_lks32mc08x_irqfun.c, 556
hal_lksmcu_irqfun.h, 238

HAL_MCPWM_IrqTH00
hal_lks32mc08x_irqfun.c, 556
hal_lksmcu_irqfun.h, 239

HAL_MCPWM_IrqTH00Clear
hal_lks32mc08x_irqfun.c, 557
hal_lksmcu_irqfun.h, 239

HAL_MCPWM_IrqTH00Disable
hal_lks32mc08x_irqfun.c, 557
hal_lksmcu_irqfun.h, 239

HAL_MCPWM_IrqTH00Enable
hal_lks32mc08x_irqfun.c, 557
hal_lksmcu_irqfun.h, 240

HAL_MCPWM_IrqTH00Flg
hal_lks32mc08x_irqfun.c, 558
hal_lksmcu_irqfun.h, 240

HAL_MCPWM_IrqTH01
hal_lks32mc08x_irqfun.c, 558
hal_lksmcu_irqfun.h, 240

HAL_MCPWM_IrqTH01Clear
hal_lks32mc08x_irqfun.c, 558
hal_lksmcu_irqfun.h, 241

HAL_MCPWM_IrqTH01Disable
hal_lks32mc08x_irqfun.c, 559
hal_lksmcu_irqfun.h, 241

HAL_MCPWM_IrqTH01Enable
hal_lks32mc08x_irqfun.c, 559
hal_lksmcu_irqfun.h, 241

HAL_MCPWM_IrqTH01Flg
hal_lks32mc08x_irqfun.c, 559
hal_lksmcu_irqfun.h, 242

HAL_MCPWM_IrqTH10
hal_lks32mc08x_irqfun.c, 560
hal_lksmcu_irqfun.h, 242

HAL_MCPWM_IrqTH10Clear
hal_lks32mc08x_irqfun.c, 560
hal_lksmcu_irqfun.h, 242

HAL_MCPWM_IrqTH10Disable

- hal_iks32mc08x_irqfun.c, 560
- hal_iksmcu_irqfun.h, 243
- HAL_MCPWM_IrqTH10Enable
 - hal_iks32mc08x_irqfun.c, 561
 - hal_iksmcu_irqfun.h, 243
- HAL_MCPWM_IrqTH10Flg
 - hal_iks32mc08x_irqfun.c, 561
 - hal_iksmcu_irqfun.h, 243
- HAL_MCPWM_IrqTH11
 - hal_iks32mc08x_irqfun.c, 561
 - hal_iksmcu_irqfun.h, 243
- HAL_MCPWM_IrqTH11Clear
 - hal_iks32mc08x_irqfun.c, 562
 - hal_iksmcu_irqfun.h, 244
- HAL_MCPWM_IrqTH11Disable
 - hal_iks32mc08x_irqfun.c, 562
 - hal_iksmcu_irqfun.h, 244
- HAL_MCPWM_IrqTH11Enable
 - hal_iks32mc08x_irqfun.c, 562
 - hal_iksmcu_irqfun.h, 244
- HAL_MCPWM_IrqTH11Flg
 - hal_iks32mc08x_irqfun.c, 562
 - hal_iksmcu_irqfun.h, 245
- HAL_MCPWM_IrqTH20
 - hal_iks32mc08x_irqfun.c, 563
 - hal_iksmcu_irqfun.h, 245
- HAL_MCPWM_IrqTH20Clear
 - hal_iks32mc08x_irqfun.c, 563
 - hal_iksmcu_irqfun.h, 245
- HAL_MCPWM_IrqTH20Disable
 - hal_iks32mc08x_irqfun.c, 563
 - hal_iksmcu_irqfun.h, 246
- HAL_MCPWM_IrqTH20Enable
 - hal_iks32mc08x_irqfun.c, 564
 - hal_iksmcu_irqfun.h, 246
- HAL_MCPWM_IrqTH20Flg
 - hal_iks32mc08x_irqfun.c, 564
 - hal_iksmcu_irqfun.h, 246
- HAL_MCPWM_IrqTH21
 - hal_iks32mc08x_irqfun.c, 564
 - hal_iksmcu_irqfun.h, 247
- HAL_MCPWM_IrqTH21Clear
 - hal_iks32mc08x_irqfun.c, 565
 - hal_iksmcu_irqfun.h, 247
- HAL_MCPWM_IrqTH21Disable
 - hal_iks32mc08x_irqfun.c, 565
 - hal_iksmcu_irqfun.h, 247
- HAL_MCPWM_IrqTH21Enable
 - hal_iks32mc08x_irqfun.c, 565
 - hal_iksmcu_irqfun.h, 248
- HAL_MCPWM_IrqTH21Flg
 - hal_iks32mc08x_irqfun.c, 565
 - hal_iksmcu_irqfun.h, 248
- HAL_MCPWM_IrqTH30
 - hal_iks32mc08x_irqfun.c, 566
 - hal_iksmcu_irqfun.h, 248
- HAL_MCPWM_IrqTH30Clear
 - hal_iks32mc08x_irqfun.c, 566
- hal_iksmcu_irqfun.h, 249
- HAL_MCPWM_IrqTH30Disable
 - hal_iks32mc08x_irqfun.c, 566
 - hal_iksmcu_irqfun.h, 249
- HAL_MCPWM_IrqTH30Enable
 - hal_iks32mc08x_irqfun.c, 567
 - hal_iksmcu_irqfun.h, 249
- HAL_MCPWM_IrqTH30Flg
 - hal_iks32mc08x_irqfun.c, 567
 - hal_iksmcu_irqfun.h, 249
- HAL_MCPWM_IrqTH31
 - hal_iks32mc08x_irqfun.c, 567
 - hal_iksmcu_irqfun.h, 250
- HAL_MCPWM_IrqTH31Clear
 - hal_iks32mc08x_irqfun.c, 568
 - hal_iksmcu_irqfun.h, 250
- HAL_MCPWM_IrqTH31Disable
 - hal_iks32mc08x_irqfun.c, 568
 - hal_iksmcu_irqfun.h, 250
- HAL_MCPWM_IrqTH31Enable
 - hal_iks32mc08x_irqfun.c, 568
 - hal_iksmcu_irqfun.h, 251
- HAL_MCPWM_IrqTH31Flg
 - hal_iks32mc08x_irqfun.c, 569
 - hal_iksmcu_irqfun.h, 251
- HAL_MCPWM_IrqTH40
 - hal_iks32mc08x_irqfun.c, 569
 - hal_iksmcu_irqfun.h, 251
- HAL_MCPWM_IrqTH40Clear
 - hal_iks32mc08x_irqfun.c, 569
 - hal_iksmcu_irqfun.h, 251
- HAL_MCPWM_IrqTH40Disable
 - hal_iks32mc08x_irqfun.c, 569
 - hal_iksmcu_irqfun.h, 251
- HAL_MCPWM_IrqTH40Enable
 - hal_iks32mc08x_irqfun.c, 569
 - hal_iksmcu_irqfun.h, 252
- HAL_MCPWM_IrqTH40Flg
 - hal_iks32mc08x_irqfun.c, 569
 - hal_iksmcu_irqfun.h, 252
- HAL_MCPWM_IrqTH41
 - hal_iks32mc08x_irqfun.c, 570
 - hal_iksmcu_irqfun.h, 252
- HAL_MCPWM_IrqTH41Clear
 - hal_iks32mc08x_irqfun.c, 570
 - hal_iksmcu_irqfun.h, 252
- HAL_MCPWM_IrqTH41Disable
 - hal_iks32mc08x_irqfun.c, 570
 - hal_iksmcu_irqfun.h, 252
- HAL_MCPWM_IrqTH41Enable
 - hal_iks32mc08x_irqfun.c, 570
 - hal_iksmcu_irqfun.h, 252
- HAL_MCPWM_IrqTH41Flg
 - hal_iks32mc08x_irqfun.c, 570
 - hal_iksmcu_irqfun.h, 252
- HAL_MCPWM_IrqTH50
 - hal_iks32mc08x_irqfun.c, 570
 - hal_iksmcu_irqfun.h, 253

- HAL_MCPWM_IrqTH50Clear
 - hal_iks32mc08x_irqfun.c, [570](#)
 - hal_iksmcu_irqfun.h, [253](#)
- HAL_MCPWM_IrqTH50Disable
 - hal_iks32mc08x_irqfun.c, [571](#)
 - hal_iksmcu_irqfun.h, [253](#)
- HAL_MCPWM_IrqTH50Enable
 - hal_iks32mc08x_irqfun.c, [571](#)
 - hal_iksmcu_irqfun.h, [253](#)
- HAL_MCPWM_IrqTH50Flg
 - hal_iks32mc08x_irqfun.c, [571](#)
 - hal_iksmcu_irqfun.h, [253](#)
- HAL_MCPWM_IrqTH51
 - hal_iks32mc08x_irqfun.c, [571](#)
 - hal_iksmcu_irqfun.h, [253](#)
- HAL_MCPWM_IrqTH51Clear
 - hal_iks32mc08x_irqfun.c, [571](#)
 - hal_iksmcu_irqfun.h, [253](#)
- HAL_MCPWM_IrqTH51Disable
 - hal_iks32mc08x_irqfun.c, [571](#)
 - hal_iksmcu_irqfun.h, [254](#)
- HAL_MCPWM_IrqTH51Enable
 - hal_iks32mc08x_irqfun.c, [571](#)
 - hal_iksmcu_irqfun.h, [254](#)
- HAL_MCPWM_IrqTH51Flg
 - hal_iks32mc08x_irqfun.c, [572](#)
 - hal_iksmcu_irqfun.h, [254](#)
- HAL_MCPWM_IrqTmr0
 - hal_iks32mc08x_irqfun.c, [572](#)
 - hal_iksmcu_irqfun.h, [254](#)
- HAL_MCPWM_IrqTmr0Clear
 - hal_iks32mc08x_irqfun.c, [572](#)
 - hal_iksmcu_irqfun.h, [254](#)
- HAL_MCPWM_IrqTmr0Disable
 - hal_iks32mc08x_irqfun.c, [572](#)
 - hal_iksmcu_irqfun.h, [255](#)
- HAL_MCPWM_IrqTmr0Enable
 - hal_iks32mc08x_irqfun.c, [573](#)
 - hal_iksmcu_irqfun.h, [255](#)
- HAL_MCPWM_IrqTmr0Flg
 - hal_iks32mc08x_irqfun.c, [573](#)
 - hal_iksmcu_irqfun.h, [255](#)
- HAL_MCPWM_IrqTmr1
 - hal_iks32mc08x_irqfun.c, [573](#)
 - hal_iksmcu_irqfun.h, [256](#)
- HAL_MCPWM_IrqTmr1Clear
 - hal_iks32mc08x_irqfun.c, [574](#)
 - hal_iksmcu_irqfun.h, [256](#)
- HAL_MCPWM_IrqTmr1Disable
 - hal_iks32mc08x_irqfun.c, [574](#)
 - hal_iksmcu_irqfun.h, [256](#)
- HAL_MCPWM_IrqTmr1Enable
 - hal_iks32mc08x_irqfun.c, [574](#)
 - hal_iksmcu_irqfun.h, [257](#)
- HAL_MCPWM_IrqTmr1Flg
 - hal_iks32mc08x_irqfun.c, [575](#)
 - hal_iksmcu_irqfun.h, [257](#)
- HAL_MCPWM_IrqTmr2
 - hal_iks32mc08x_irqfun.c, [575](#)
 - hal_iksmcu_irqfun.h, [257](#)
- HAL_MCPWM_IrqTmr2Clear
 - hal_iks32mc08x_irqfun.c, [575](#)
 - hal_iksmcu_irqfun.h, [258](#)
- HAL_MCPWM_IrqTmr2Disable
 - hal_iks32mc08x_irqfun.c, [576](#)
 - hal_iksmcu_irqfun.h, [258](#)
- HAL_MCPWM_IrqTmr2Enable
 - hal_iks32mc08x_irqfun.c, [576](#)
 - hal_iksmcu_irqfun.h, [258](#)
- HAL_MCPWM_IrqTmr2Flg
 - hal_iks32mc08x_irqfun.c, [576](#)
 - hal_iksmcu_irqfun.h, [258](#)
- HAL_MCPWM_IrqTmr3
 - hal_iks32mc08x_irqfun.c, [576](#)
 - hal_iksmcu_irqfun.h, [259](#)
- HAL_MCPWM_IrqTmr3Clear
 - hal_iks32mc08x_irqfun.c, [577](#)
 - hal_iksmcu_irqfun.h, [259](#)
- HAL_MCPWM_IrqTmr3Disable
 - hal_iks32mc08x_irqfun.c, [577](#)
 - hal_iksmcu_irqfun.h, [259](#)
- HAL_MCPWM_IrqTmr3Enable
 - hal_iks32mc08x_irqfun.c, [577](#)
 - hal_iksmcu_irqfun.h, [260](#)
- HAL_MCPWM_IrqTmr3Flg
 - hal_iks32mc08x_irqfun.c, [578](#)
 - hal_iksmcu_irqfun.h, [260](#)
- HAL_MCPWM_IrqUpdate
 - hal_iks32mc08x_irqfun.c, [578](#)
 - hal_iksmcu_irqfun.h, [260](#)
- HAL_MCPWM_IrqUpdateClear
 - hal_iks32mc08x_irqfun.c, [578](#)
 - hal_iksmcu_irqfun.h, [261](#)
- HAL_MCPWM_IrqUpdateDisable
 - hal_iks32mc08x_irqfun.c, [579](#)
 - hal_iksmcu_irqfun.h, [261](#)
- HAL_MCPWM_IrqUpdateEnable
 - hal_iks32mc08x_irqfun.c, [579](#)
 - hal_iksmcu_irqfun.h, [261](#)
- HAL_MCPWM_IrqUpdateFlg
 - hal_iks32mc08x_irqfun.c, [579](#)
 - hal_iksmcu_irqfun.h, [262](#)
- HAL_MCPWM_IrqZeroCross
 - hal_iks32mc08x_irqfun.c, [580](#)
 - hal_iksmcu_irqfun.h, [262](#)
- HAL_MCPWM_IrqZeroCrossClear
 - hal_iks32mc08x_irqfun.c, [580](#)
 - hal_iksmcu_irqfun.h, [262](#)
- HAL_MCPWM_IrqZeroCrossDisable
 - hal_iks32mc08x_irqfun.c, [580](#)
 - hal_iksmcu_irqfun.h, [263](#)
- HAL_MCPWM_IrqZeroCrossEnable
 - hal_iks32mc08x_irqfun.c, [581](#)
 - hal_iksmcu_irqfun.h, [263](#)
- HAL_MCPWM_IrqZeroCrossFlg
 - hal_iks32mc08x_irqfun.c, [581](#)

- hal_lksmcu_irqfun.h, 263
- HAL_MCPWM_OUTPUT_STATE_HIGH
 - hal_lksmcu_mcpwm.h, 298
- HAL_MCPWM_OUTPUT_STATE_HIGH_PWM
 - hal_lksmcu_mcpwm.h, 298
- HAL_MCPWM_OUTPUT_STATE_LOW
 - hal_lksmcu_mcpwm.h, 298
- HAL_MCPWM_OUTPUT_STATE_LOW_PWM
 - hal_lksmcu_mcpwm.h, 298
- HAL_MCPWM_OUTPUT_STATE_OFF
 - hal_lksmcu_mcpwm.h, 298
- HAL_MCPWM_OUTPUT_STATE_PWM
 - hal_lksmcu_mcpwm.h, 298
- HAL_MCPWM_OUTPUT_STATE_x
 - hal_lksmcu_mcpwm.h, 298
- HAL_MCPWM_POLARITY_PH_NH
 - hal_lksmcu_mcpwm.h, 299
- HAL_MCPWM_POLARITY_PH_NL
 - hal_lksmcu_mcpwm.h, 299
- HAL_MCPWM_POLARITY_PL_NH
 - hal_lksmcu_mcpwm.h, 299
- HAL_MCPWM_POLARITY_PL_NL
 - hal_lksmcu_mcpwm.h, 299
- HAL_MCPWM_POLARITY_x
 - hal_lksmcu_mcpwm.h, 298
- HAL_MCPWM_PORT_0
 - hal_lksmcu_mcpwm.h, 299
- HAL_MCPWM_PORT_1
 - hal_lksmcu_mcpwm.h, 299
- HAL_MCPWM_PORT_2
 - hal_lksmcu_mcpwm.h, 299
- HAL_MCPWM_PORT_3
 - hal_lksmcu_mcpwm.h, 299
- HAL_MCPWM_PORT_x
 - hal_lksmcu_mcpwm.h, 299
- HAL_MCPWM_Reset
 - hal_lks32mc08x_mcpwm.c, 647
 - hal_lksmcu_mcpwm.h, 303
- HAL_MCPWM_SetAdcTrig
 - hal_lks32mc08x_mcpwm.c, 648
 - hal_lksmcu_mcpwm.h, 303
- HAL_MCPWM_SetAdcTrigQ15
 - hal_lks32mc08x_mcpwm.c, 648
 - hal_lksmcu_mcpwm.h, 304
- HAL_MCPWM_SetClockDiv
 - hal_lks32mc08x_mcpwm.c, 649
 - hal_lksmcu_mcpwm.h, 304
- HAL_MCPWM_SetDead
 - hal_lks32mc08x_mcpwm.c, 649
 - hal_lksmcu_mcpwm.h, 304
- HAL_MCPWM_SetDefaultState
 - hal_lks32mc08x_mcpwm.c, 649
 - hal_lksmcu_mcpwm.h, 305
- HAL_MCPWM_SetDrivePolarity
 - hal_lks32mc08x_mcpwm.c, 650
 - hal_lksmcu_mcpwm.h, 306
- HAL_MCPWM_SetDrivePolarityPort
 - hal_lks32mc08x_mcpwm.c, 650
- hal_lksmcu_mcpwm.h, 306
- HAL_MCPWM_SetFailFilter
 - hal_lks32mc08x_mcpwm.c, 651
 - hal_lksmcu_mcpwm.h, 307
- HAL_MCPWM_SetFailSignal
 - hal_lks32mc08x_mcpwm.c, 652
 - hal_lksmcu_mcpwm.h, 307
- HAL_MCPWM_SetFreq
 - hal_lks32mc08x_mcpwm.c, 652
 - hal_lksmcu_mcpwm.h, 308
- HAL_MCPWM_SetGpioFail
 - hal_lks32mc08x_mcpwm.c, 653
 - hal_lksmcu_mcpwm.h, 308
- HAL_MCPWM_SetGpioOut
 - hal_lks32mc08x_mcpwm.c, 653
 - hal_lksmcu_mcpwm.h, 309
- HAL_MCPWM_SetOutputDisable
 - hal_lks32mc08x_mcpwm.c, 654
 - hal_lksmcu_mcpwm.h, 309
- HAL_MCPWM_SetOutputEnable
 - hal_lks32mc08x_mcpwm.c, 654
 - hal_lksmcu_mcpwm.h, 310
- HAL_MCPWM_SetOutState
 - hal_lks32mc08x_mcpwm.c, 654
 - hal_lksmcu_mcpwm.h, 310
- HAL_MCPWM_SetPwm
 - hal_lks32mc08x_mcpwm.c, 655
 - hal_lksmcu_mcpwm.h, 311
- HAL_MCPWM_SetPwmQ15
 - hal_lks32mc08x_mcpwm.c, 655
 - hal_lksmcu_mcpwm.h, 311
- HAL_MCPWM_SetRemap
 - hal_lks32mc08x_mcpwm.c, 656
 - hal_lksmcu_mcpwm.h, 312
- HAL_MCPWM_SetSwap
 - hal_lks32mc08x_mcpwm.c, 656
 - hal_lksmcu_mcpwm.h, 312
- HAL_MCPWM_SetSwapPort
 - hal_lks32mc08x_mcpwm.c, 657
 - hal_lksmcu_mcpwm.h, 313
- HAL_MCPWM_SetThCnt
 - hal_lks32mc08x_mcpwm.c, 658
 - hal_lksmcu_mcpwm.h, 313
- HAL_MCPWM_SetUpdateMode
 - hal_lks32mc08x_mcpwm.c, 658
 - hal_lksmcu_mcpwm.h, 314
- HAL_MCPWM_StartCounting
 - hal_lks32mc08x_mcpwm.c, 658
 - hal_lksmcu_mcpwm.h, 314
- HAL_MCPWM_StartCountingSync
 - hal_lks32mc08x_mcpwm.c, 659
 - hal_lksmcu_mcpwm.h, 315
- HAL_MCPWM_STATE
 - hal_lks32mc08x_mcpwm.c, 661
- HAL_MCPWM_STATE_H_DEFAULT_BIT
 - hal_lks32mc08x_reg.h, 61
- HAL_MCPWM_STATE_H_MOS_EN_BIT
 - hal_lks32mc08x_reg.h, 61

HAL_MCPWM_STATE_INIT_FLG
 hal_iks32mc08x_mcpwm.c, 661
 HAL_MCPWM_STATE_L_DEFAULT_BIT
 hal_iks32mc08x_reg.h, 61
 HAL_MCPWM_STATE_L_MOS_EN_BIT
 hal_iks32mc08x_reg.h, 61
 HAL_MCPWM_STATE_STATE
 hal_iks32mc08x_reg.h, 61
 HAL_MCPWM_STATE_STATE_MASK
 hal_iks32mc08x_reg.h, 61
 HAL_MCPWM_STATE_STATE_MAX
 hal_iks32mc08x_reg.h, 62
 HAL_MCPWM_STATE_STATE_POS
 hal_iks32mc08x_reg.h, 62
 HAL_MCPWM_STATE_SW_BIT
 hal_iks32mc08x_reg.h, 62
 HAL_MCPWM_StopCounting
 hal_iks32mc08x_mcpwm.c, 659
 hal_iksmcu_mcpwm.h, 315
 HAL_MCPWM_SWAP_DISABLE
 hal_iksmcu_mcpwm.h, 299
 HAL_MCPWM_SWAP_ENABLE
 hal_iksmcu_mcpwm.h, 299
 HAL_MCPWM_SWAP_x
 hal_iksmcu_mcpwm.h, 299
 HAL_MCPWM_TypeDef, 33
 CNT, 34
 DTH00, 34
 DTH01, 35
 DTH10, 35
 DTH11, 35
 DTH20, 35
 DTH21, 35
 DTH30, 35
 DTH31, 35
 EIE, 35
 EIF, 36
 FAIL, 36
 FILT, 36
 IE0, 36
 IF0, 36
 IO01, 36
 IO23, 36
 null1, 36
 null2, 37
 null3, 37
 PP, 37
 PRT, 37
 RE, 37
 SDCFG, 37
 TCLK, 37
 TH, 37
 TH0, 38
 TMR0, 38
 TMR1, 38
 TMR2, 38
 TMR3, 38
 UPDATE, 38
 HAL_MCPWM_UPDATE_T0
 hal_iksmcu_mcpwm.h, 300
 HAL_MCPWM_UPDATE_T0_AND_T1
 hal_iksmcu_mcpwm.h, 300
 HAL_MCPWM_UPDATE_T1
 hal_iksmcu_mcpwm.h, 300
 HAL_MCPWM_UPDATE_x
 hal_iksmcu_mcpwm.h, 299
 HAL_MCPWM_UpdatePolarityState
 hal_iks32mc08x_mcpwm.c, 660
 HAL_MCPWM_x
 hal_iksmcu_mcpwm.h, 300
 HAL_OPA_0
 hal_iksmcu_opa.h, 320
 HAL_OPA_0B
 hal_iksmcu_opa.h, 320
 HAL_OPA_1
 hal_iksmcu_opa.h, 320
 HAL_OPA_2
 hal_iksmcu_opa.h, 320
 HAL_OPA_3
 hal_iksmcu_opa.h, 320
 HAL_OPA_4
 hal_iksmcu_opa.h, 320
 HAL_OPA_5
 hal_iksmcu_opa.h, 320
 HAL_OPA_Disable
 hal_iks32mc08x_opa.c, 668
 hal_iksmcu_opa.h, 320
 HAL_OPA_Enable
 hal_iks32mc08x_opa.c, 668
 hal_iksmcu_opa.h, 320
 HAL_OPA_GAIN_170_40_4
 hal_iksmcu_opa.h, 319
 HAL_OPA_GAIN_180_30_4
 hal_iksmcu_opa.h, 319
 HAL_OPA_GAIN_190_20_4
 hal_iksmcu_opa.h, 319
 HAL_OPA_GAIN_200_10_4
 hal_iksmcu_opa.h, 319
 HAL_OPA_GAIN_HIGH
 hal_iksmcu_opa.h, 319
 HAL_OPA_GAIN_LOW
 hal_iksmcu_opa.h, 319
 HAL_OPA_GAIN_MEDIUM
 hal_iksmcu_opa.h, 319
 HAL_OPA_GAIN_x
 hal_iksmcu_opa.h, 319
 HAL_OPA_OPAX_2_ADCCHN
 hal_iksmcu_opa.h, 319
 HAL_OPA_Out2GpioDisable
 hal_iks32mc08x_opa.c, 668
 hal_iksmcu_opa.h, 320
 HAL_OPA_Out2GpioEnable
 hal_iks32mc08x_opa.c, 669
 hal_iksmcu_opa.h, 321
 HAL_OPA_READ_GAIN
 hal_iksmcu_opa.h, 319

HAL_OPA_Reset
 hal_lks32mc08x_opa.c, 669
 hal_lksmcu_opa.h, 321

HAL_OPA_SetGain
 hal_lks32mc08x_opa.c, 669
 hal_lksmcu_opa.h, 321

HAL_OPA_SetGpioIn
 hal_lks32mc08x_opa.c, 669
 hal_lksmcu_opa.h, 321

HAL_OPA_SetGpioOut
 hal_lks32mc08x_opa.c, 670
 hal_lksmcu_opa.h, 322

HAL_OPA_x
 hal_lksmcu_opa.h, 320

HAL_QEP
 hal_lks32mc08x_reg.c, 682
 hal_lks32mc08x_reg.h, 64

HAL_QEP_0
 hal_lksmcu_qep.h, 325

HAL_QEP_1
 hal_lksmcu_qep.h, 325

HAL_QEP_2
 hal_lksmcu_qep.h, 325

HAL_QEP_3
 hal_lksmcu_qep.h, 325

HAL_QEP_Disable
 hal_lks32mc08x_qep.c, 674
 hal_lksmcu_qep.h, 325

HAL_QEP_Enable
 hal_lks32mc08x_qep.c, 674
 hal_lksmcu_qep.h, 326

HAL_QEP_GetClockDiv
 hal_lks32mc08x_qep.c, 674
 hal_lksmcu_qep.h, 326

HAL_QEP_GetClockFreq
 hal_lks32mc08x_qep.c, 675
 hal_lksmcu_qep.h, 327

HAL_QEP_MODE_DUAL_PULSE
 hal_lksmcu_qep.h, 325

HAL_QEP_MODE_QUADRATURE
 hal_lksmcu_qep.h, 325

HAL_QEP_MODE_SYMBOL_PULSE
 hal_lksmcu_qep.h, 325

HAL_QEP_MODE_x
 hal_lksmcu_qep.h, 325

HAL_QEP_ReadCnt
 hal_lks32mc08x_qep.c, 675
 hal_lksmcu_qep.h, 327

HAL_QEP_Reset
 hal_lks32mc08x_qep.c, 676
 hal_lksmcu_qep.h, 327

HAL_QEP_ResetCnt
 hal_lks32mc08x_qep.c, 676
 hal_lksmcu_qep.h, 328

HAL_QEP_SetClockDiv
 hal_lks32mc08x_qep.c, 676
 hal_lksmcu_qep.h, 328

HAL_QEP_SetGpio
 hal_lks32mc08x_qep.c, 676
 hal_lksmcu_qep.h, 328

HAL_QEP_SetMode
 hal_lks32mc08x_qep.c, 677
 hal_lksmcu_qep.h, 329

HAL_QEP_SetTh
 hal_lks32mc08x_qep.c, 677
 hal_lksmcu_qep.h, 329

HAL_QEP_TypeDef, 38
 CFG, 39
 CNT, 39
 TH, 39

HAL_QEP_x
 hal_lksmcu_qep.h, 325

HAL_REG_IF_WRITE
 hal_lksmcu_reg.h, 331

HAL_REG_READ
 hal_lksmcu_reg.h, 331

HAL_REG_READ_ADDR
 hal_lksmcu_reg.h, 331

HAL_REG_READ_BIT
 hal_lksmcu_reg.h, 331

HAL_REG_READ_BIT_MASK_POS
 hal_lksmcu_reg.h, 331

HAL_REG_RESET
 hal_lksmcu_reg.h, 331

HAL_REG_SET
 hal_lksmcu_reg.h, 332

HAL_REG_SYS_LOCK
 hal_lksmcu_reg.h, 332

HAL_REG_SYS_UNLOCK
 hal_lksmcu_reg.h, 332

HAL_REG_WRITE
 hal_lksmcu_reg.h, 332

HAL_REG_WRITE_BIT
 hal_lksmcu_reg.h, 332

HAL_REG_WRITE_BIT_MASK_POS
 hal_lksmcu_reg.h, 332

HAL_RESET_ResetCore
 hal_lks32mc08x_reset.c, 684
 hal_lksmcu_reset.h, 333

HAL_RESET_ResetGlobal
 hal_lks32mc08x_reset.c, 684
 hal_lksmcu_reset.h, 333

HAL_RESET_ResetRegister
 hal_lks32mc08x_reset.c, 684
 hal_lksmcu_reset.h, 334

hal_rx_buf
 hal_lks32mc08x_can.c, 410

hal_rx_id
 hal_lks32mc08x_can.c, 410

hal_rx_ide
 hal_lks32mc08x_can.c, 410

hal_rx_len
 hal_lks32mc08x_can.c, 410

hal_rx_rtr
 hal_lks32mc08x_can.c, 411

HAL_SPI

hal_lks32mc08x_reg.c, 682
hal_lks32mc08x_reg.h, 64
HAL_SPI_0
hal_lksmcu_spi.h, 337
HAL_SPI_1
hal_lksmcu_spi.h, 337
HAL_SPI_Disable
hal_lks32mc08x_spi.c, 687
hal_lksmcu_spi.h, 337
HAL_SPI_Enable
hal_lks32mc08x_spi.c, 687
hal_lksmcu_spi.h, 337
HAL_SPI_GetBaud
hal_lks32mc08x_spi.c, 687
hal_lksmcu_spi.h, 338
HAL_SPI_GetClockDiv
hal_lks32mc08x_spi.c, 688
hal_lksmcu_spi.h, 338
HAL_SPI_GetClockFreq
hal_lks32mc08x_spi.c, 688
hal_lksmcu_spi.h, 338
HAL_SPI_IrqDone
hal_lks32mc08x_irqfun.c, 581
hal_lksmcu_irqfun.h, 263
HAL_SPI_IrqDoneClear
hal_lks32mc08x_irqfun.c, 582
hal_lksmcu_irqfun.h, 264
HAL_SPI_IrqDoneDisable
hal_lks32mc08x_irqfun.c, 582
hal_lksmcu_irqfun.h, 264
HAL_SPI_IrqDoneEnable
hal_lks32mc08x_irqfun.c, 582
hal_lksmcu_irqfun.h, 264
HAL_SPI_IrqDoneFlg
hal_lks32mc08x_irqfun.c, 582
hal_lksmcu_irqfun.h, 265
HAL_SPI_IrqErr
hal_lks32mc08x_irqfun.c, 583
hal_lksmcu_irqfun.h, 265
HAL_SPI_IrqErrClear
hal_lks32mc08x_irqfun.c, 583
hal_lksmcu_irqfun.h, 265
HAL_SPI_IrqErrDisable
hal_lks32mc08x_irqfun.c, 584
hal_lksmcu_irqfun.h, 266
HAL_SPI_IrqErrEnable
hal_lks32mc08x_irqfun.c, 584
hal_lksmcu_irqfun.h, 266
HAL_SPI_IrqErrFlg
hal_lks32mc08x_irqfun.c, 584
hal_lksmcu_irqfun.h, 266
HAL_SPI_IrqOverflow
hal_lks32mc08x_irqfun.c, 584
hal_lksmcu_irqfun.h, 267
HAL_SPI_IrqOverflowClear
hal_lks32mc08x_irqfun.c, 585
hal_lksmcu_irqfun.h, 267
HAL_SPI_IrqOverflowDisable
hal_lks32mc08x_irqfun.c, 585
hal_lksmcu_irqfun.h, 267
HAL_SPI_IrqOverflowEnable
hal_lks32mc08x_irqfun.c, 585
hal_lksmcu_irqfun.h, 268
HAL_SPI_IrqOverflowFlg
hal_lks32mc08x_irqfun.c, 586
hal_lksmcu_irqfun.h, 268
HAL_SPI_MODE_MASTER_LSB_00
hal_lksmcu_spi.h, 336
HAL_SPI_MODE_MASTER_LSB_01
hal_lksmcu_spi.h, 336
HAL_SPI_MODE_MASTER_LSB_10
hal_lksmcu_spi.h, 336
HAL_SPI_MODE_MASTER_LSB_11
hal_lksmcu_spi.h, 336
HAL_SPI_MODE_MASTER_MSB_00
hal_lksmcu_spi.h, 336
HAL_SPI_MODE_MASTER_MSB_01
hal_lksmcu_spi.h, 336
HAL_SPI_MODE_MASTER_MSB_10
hal_lksmcu_spi.h, 337
HAL_SPI_MODE_MASTER_MSB_11
hal_lksmcu_spi.h, 337
HAL_SPI_MODE_SLAVE_LSB_00
hal_lksmcu_spi.h, 337
HAL_SPI_MODE_SLAVE_LSB_01
hal_lksmcu_spi.h, 337
HAL_SPI_MODE_SLAVE_LSB_10
hal_lksmcu_spi.h, 337
HAL_SPI_MODE_SLAVE_LSB_11
hal_lksmcu_spi.h, 337
HAL_SPI_MODE_SLAVE_MSB_00
hal_lksmcu_spi.h, 337
HAL_SPI_MODE_SLAVE_MSB_01
hal_lksmcu_spi.h, 337
HAL_SPI_MODE_SLAVE_MSB_10
hal_lksmcu_spi.h, 337
HAL_SPI_MODE_SLAVE_MSB_11
hal_lksmcu_spi.h, 337
HAL_SPI_MODE_x
hal_lksmcu_spi.h, 336
HAL_SPI_Reset
hal_lks32mc08x_spi.c, 689
hal_lksmcu_spi.h, 339
HAL_SPI_RxData
hal_lks32mc08x_spi.c, 689
hal_lksmcu_spi.h, 339
HAL_SPI_RxDataWidth16
hal_lks32mc08x_spi.c, 690
hal_lksmcu_spi.h, 340
HAL_SPI_RxTxData
hal_lks32mc08x_spi.c, 690
hal_lksmcu_spi.h, 341
HAL_SPI_RxTxDataWidth16
hal_lks32mc08x_spi.c, 691
hal_lksmcu_spi.h, 341
HAL_SPI_SetBaud

- hal_lks32mc08x_spi.c, 692
- hal_lksmcu_spi.h, 342
- HAL_SPI_SetClockDiv
 - hal_lks32mc08x_spi.c, 692
 - hal_lksmcu_spi.h, 343
- HAL_SPI_SetGpioClk
 - hal_lks32mc08x_spi.c, 693
 - hal_lksmcu_spi.h, 343
- HAL_SPI_SetGpioCs
 - hal_lks32mc08x_spi.c, 693
 - hal_lksmcu_spi.h, 344
- HAL_SPI_SetGpioDi
 - hal_lks32mc08x_spi.c, 694
 - hal_lksmcu_spi.h, 345
- HAL_SPI_SetGpioDo
 - hal_lks32mc08x_spi.c, 694
 - hal_lksmcu_spi.h, 345
- HAL_SPI_SetGpioMiso
 - hal_lks32mc08x_spi.c, 695
 - hal_lksmcu_spi.h, 346
- HAL_SPI_SetGpioMosi
 - hal_lks32mc08x_spi.c, 696
 - hal_lksmcu_spi.h, 346
- HAL_SPI_SetMode
 - hal_lks32mc08x_spi.c, 696
 - hal_lksmcu_spi.h, 347
- HAL_SPI_TxData
 - hal_lks32mc08x_spi.c, 697
 - hal_lksmcu_spi.h, 347
- HAL_SPI_TxDataWidth16
 - hal_lks32mc08x_spi.c, 697
 - hal_lksmcu_spi.h, 348
- HAL_SPI_TypeDef, 39
 - BAUD, 40
 - CFG, 40
 - IE, 40
 - RXDATA, 40
 - SIZE, 41
 - TXDATA, 41
- HAL_SPI_x
 - hal_lksmcu_spi.h, 337
- HAL_SW_Irq
 - hal_lks32mc08x_irqfun.c, 586
 - hal_lksmcu_irqfun.h, 268
- HAL_SW_IrqClear
 - hal_lks32mc08x_irqfun.c, 586
 - hal_lksmcu_irqfun.h, 268
- HAL_SW_IrqDisable
 - hal_lks32mc08x_irqfun.c, 586
 - hal_lksmcu_irqfun.h, 269
- HAL_SW_IrqEnable
 - hal_lks32mc08x_irqfun.c, 587
 - hal_lksmcu_irqfun.h, 269
- HAL_SW_IrqFlg
 - hal_lks32mc08x_irqfun.c, 587
 - hal_lksmcu_irqfun.h, 269
- HAL_SW_IrqTrig
 - hal_lks32mc08x_irqfun.c, 587
- hal_lksmcu_irqfun.h, 269
- HAL_SYS
 - hal_lks32mc08x_reg.c, 682
 - hal_lks32mc08x_reg.h, 64
- HAL_SYS_TypeDef, 41
 - AFE_CMP, 43
 - AFE_DAC, 43
 - AFE_DAC_AMC, 43
 - AFE_DAC_DC, 43
 - AFE_REG0, 43
 - AFE_REG1, 44
 - AFE_REG2, 44
 - AFE_REG3, 44
 - AFE_REG4, 44
 - AFE_REG5, 44
 - AFE_REG6, 44
 - AFE_REG7, 45
 - AFE_REG8, 45
 - AFE_REG9, 45
 - AFE_REGA, 45
 - AFE_REGB, 45
 - AFE_REGC, 45
 - CLK_CFG, 46
 - CLK_DIV0, 46
 - CLK_DIV2, 46
 - CLK_FEN, 46
 - CLK_SLP, 46
 - CLR_RST, 46
 - PROTECT, 47
 - RESERVED, 47
 - RESERVED1, 47
 - RESERVED2, 47
 - RESERVED3, 47
 - RESERVED4, 47
 - RESERVED5, 47
 - RST_CFG, 48
 - RST_SRC, 48
 - SFT_RST, 48
 - TMP_A, 48
 - TMP_B, 48
 - TRIM, 48
 - WDT_CLR, 49
 - WDT_CNT, 49
 - WDT_PSW, 49
 - WDT_TH, 49
- HAL_TIMER
 - hal_lks32mc08x_reg.c, 682
 - hal_lks32mc08x_reg.h, 64
- HAL_TIMER_0
 - hal_lksmcu_timer.h, 352
- HAL_TIMER_1
 - hal_lksmcu_timer.h, 352
- HAL_TIMER_2
 - hal_lksmcu_timer.h, 352
- HAL_TIMER_3
 - hal_lksmcu_timer.h, 352
- HAL_TIMER_4
 - hal_lksmcu_timer.h, 352

HAL_TIMER_Disable
 hal_iks32mc08x_timer.c, 703
 hal_iksmcu_timer.h, 352
 hal_timer_ena
 hal_iks32mc08x_timer.c, 710
 HAL_TIMER_Enable
 hal_iks32mc08x_timer.c, 703
 hal_iksmcu_timer.h, 352
 HAL_TIMER_GetClockDiv
 hal_iks32mc08x_timer.c, 703
 hal_iksmcu_timer.h, 352
 HAL_TIMER_GetClockFreq
 hal_iks32mc08x_timer.c, 704
 hal_iksmcu_timer.h, 353
 HAL_TIMER_GetFreq
 hal_iks32mc08x_timer.c, 704
 hal_iksmcu_timer.h, 353
 HAL_TIMER_GetThCnt
 hal_iks32mc08x_timer.c, 704
 hal_iksmcu_timer.h, 353
 HAL_TIMER_IrqCompare0
 hal_iks32mc08x_irqfun.c, 587
 hal_iksmcu_irqfun.h, 269
 HAL_TIMER_IrqCompare0Clear
 hal_iks32mc08x_irqfun.c, 587
 hal_iksmcu_irqfun.h, 270
 HAL_TIMER_IrqCompare0Disable
 hal_iks32mc08x_irqfun.c, 588
 hal_iksmcu_irqfun.h, 270
 HAL_TIMER_IrqCompare0Enable
 hal_iks32mc08x_irqfun.c, 588
 hal_iksmcu_irqfun.h, 270
 HAL_TIMER_IrqCompare0Flg
 hal_iks32mc08x_irqfun.c, 588
 hal_iksmcu_irqfun.h, 270
 HAL_TIMER_IrqCompare1
 hal_iks32mc08x_irqfun.c, 589
 hal_iksmcu_irqfun.h, 271
 HAL_TIMER_IrqCompare1Clear
 hal_iks32mc08x_irqfun.c, 589
 hal_iksmcu_irqfun.h, 271
 HAL_TIMER_IrqCompare1Disable
 hal_iks32mc08x_irqfun.c, 589
 hal_iksmcu_irqfun.h, 272
 HAL_TIMER_IrqCompare1Enable
 hal_iks32mc08x_irqfun.c, 590
 hal_iksmcu_irqfun.h, 272
 HAL_TIMER_IrqCompare1Flg
 hal_iks32mc08x_irqfun.c, 590
 hal_iksmcu_irqfun.h, 272
 HAL_TIMER_IrqOverflow
 hal_iks32mc08x_irqfun.c, 590
 hal_iksmcu_irqfun.h, 272
 HAL_TIMER_IrqOverflowClear
 hal_iks32mc08x_irqfun.c, 591
 hal_iksmcu_irqfun.h, 273
 HAL_TIMER_IrqOverflowDisable
 hal_iks32mc08x_irqfun.c, 591
 hal_iksmcu_irqfun.h, 273
 HAL_TIMER_IrqOverflowEnable
 hal_iks32mc08x_irqfun.c, 591
 hal_iksmcu_irqfun.h, 273
 HAL_TIMER_IrqOverflowFlg
 hal_iks32mc08x_irqfun.c, 592
 hal_iksmcu_irqfun.h, 274
 HAL_TIMER_MODE_COUNT
 hal_iksmcu_timer.h, 351
 HAL_TIMER_MODE_PWMIN
 hal_iksmcu_timer.h, 351
 HAL_TIMER_MODE_PWMOUT
 hal_iksmcu_timer.h, 351
 HAL_TIMER_MODE_x
 hal_iksmcu_timer.h, 351
 HAL_TIMER_Reset
 hal_iks32mc08x_timer.c, 705
 hal_iksmcu_timer.h, 354
 HAL_TIMER_SetClockDiv
 hal_iks32mc08x_timer.c, 705
 hal_iksmcu_timer.h, 354
 HAL_TIMER_SetFreq
 hal_iks32mc08x_timer.c, 705
 hal_iksmcu_timer.h, 354
 HAL_TIMER_SetGpioIn
 hal_iks32mc08x_timer.c, 706
 hal_iksmcu_timer.h, 355
 HAL_TIMER_SetGpioOut
 hal_iks32mc08x_timer.c, 707
 hal_iksmcu_timer.h, 356
 HAL_TIMER_SetPeriodCount
 hal_iks32mc08x_timer.c, 707
 hal_iksmcu_timer.h, 356
 HAL_TIMER_SetPeriodmS
 hal_iks32mc08x_timer.c, 708
 hal_iksmcu_timer.h, 357
 HAL_TIMER_SetPerioduS
 hal_iks32mc08x_timer.c, 708
 hal_iksmcu_timer.h, 357
 HAL_TIMER_SetPwm
 hal_iks32mc08x_timer.c, 709
 hal_iksmcu_timer.h, 358
 HAL_TIMER_SetPwmQ15
 hal_iks32mc08x_timer.c, 709
 hal_iksmcu_timer.h, 358
 HAL_TIMER_SetWorkMode
 hal_iks32mc08x_timer.c, 709
 hal_iksmcu_timer.h, 358
 HAL_TIMER_StartCounting
 hal_iks32mc08x_timer.c, 710
 hal_iksmcu_timer.h, 359
 HAL_TIMER_StopCounting
 hal_iks32mc08x_timer.c, 710
 hal_iksmcu_timer.h, 359
 HAL_TIMER_TypeDef, 49
 CFG, 50
 CMPT0, 50
 CMPT1, 50

CNT, 50
EVT, 51
TH, 51
HAL_TIMER_x
 hal_lksmcu_timer.h, 351
HAL_TMP_0
 hal_lksmcu_tmp.h, 362
HAL_TMP_1
 hal_lksmcu_tmp.h, 362
HAL_TMP_Adc2Temperature
 hal_lks32mc08x_tmp.c, 715
 hal_lksmcu_tmp.h, 362
HAL_TMP_Adc2Temperature01
 hal_lks32mc08x_tmp.c, 716
 hal_lksmcu_tmp.h, 363
HAL_TMP_Disable
 hal_lks32mc08x_tmp.c, 716
 hal_lksmcu_tmp.h, 363
HAL_TMP_Enable
 hal_lks32mc08x_tmp.c, 717
 hal_lksmcu_tmp.h, 364
HAL_TMP_GetTemperature
 hal_lks32mc08x_tmp.c, 717
 hal_lksmcu_tmp.h, 364
HAL_TMP_GetTemperature01
 hal_lks32mc08x_tmp.c, 718
 hal_lksmcu_tmp.h, 365
HAL_TMP_Reset
 hal_lks32mc08x_tmp.c, 719
 hal_lksmcu_tmp.h, 366
HAL_TMP_x
 hal_lksmcu_tmp.h, 362
HAL_TRIM_Read
 hal_lks32mc08x_trim.c, 721
 hal_lksmcu_trim.h, 367
HAL_TRIM_ReadId
 hal_lks32mc08x_trim.c, 722
 hal_lksmcu_trim.h, 368
HAL_TRIM_ReadIdCrc32
 hal_lks32mc08x_trim.c, 723
 hal_lksmcu_trim.h, 369
HAL_TRIM_ReadMcu08
 hal_lks32mc08x_trim.c, 723
HAL_UART
 hal_lks32mc08x_reg.c, 683
 hal_lks32mc08x_reg.h, 64
HAL_UART_0
 hal_lksmcu_uart.h, 372
HAL_UART_1
 hal_lksmcu_uart.h, 372
HAL_UART_Disable
 hal_lks32mc08x_uart.c, 726
 hal_lksmcu_uart.h, 372
HAL_UART_Enable
 hal_lks32mc08x_uart.c, 726
 hal_lksmcu_uart.h, 372
HAL_UART_GetBaud
 hal_lks32mc08x_uart.c, 726
 hal_lksmcu_uart.h, 372
HAL_UART_GetClockDiv
 hal_lks32mc08x_uart.c, 726
 hal_lksmcu_uart.h, 372
HAL_UART_GetClockFreq
 hal_lks32mc08x_uart.c, 726
 hal_lksmcu_uart.h, 373
HAL_UART_IrqAbd
 hal_lks32mc08x_irqfun.c, 592
 hal_lksmcu_irqfun.h, 274
HAL_UART_IrqAbdClear
 hal_lks32mc08x_irqfun.c, 592
 hal_lksmcu_irqfun.h, 274
HAL_UART_IrqAbdDisable
 hal_lks32mc08x_irqfun.c, 592
 hal_lksmcu_irqfun.h, 274
HAL_UART_IrqAbdEnable
 hal_lks32mc08x_irqfun.c, 592
 hal_lksmcu_irqfun.h, 275
HAL_UART_IrqAbdFlg
 hal_lks32mc08x_irqfun.c, 593
 hal_lksmcu_irqfun.h, 275
HAL_UART_IrqCheckErr
 hal_lks32mc08x_irqfun.c, 593
 hal_lksmcu_irqfun.h, 275
HAL_UART_IrqCheckErrClear
 hal_lks32mc08x_irqfun.c, 593
 hal_lksmcu_irqfun.h, 275
HAL_UART_IrqCheckErrDisable
 hal_lks32mc08x_irqfun.c, 593
 hal_lksmcu_irqfun.h, 276
HAL_UART_IrqCheckErrEnable
 hal_lks32mc08x_irqfun.c, 594
 hal_lksmcu_irqfun.h, 276
HAL_UART_IrqCheckErrFlg
 hal_lks32mc08x_irqfun.c, 594
 hal_lksmcu_irqfun.h, 276
HAL_UART_IrqIdle
 hal_lks32mc08x_irqfun.c, 594
 hal_lksmcu_irqfun.h, 277
HAL_UART_IrqIdleClear
 hal_lks32mc08x_irqfun.c, 595
 hal_lksmcu_irqfun.h, 277
HAL_UART_IrqIdleDisable
 hal_lks32mc08x_irqfun.c, 595
 hal_lksmcu_irqfun.h, 277
HAL_UART_IrqIdleEnable
 hal_lks32mc08x_irqfun.c, 595
 hal_lksmcu_irqfun.h, 277
HAL_UART_IrqIdleFlg
 hal_lks32mc08x_irqfun.c, 595
 hal_lksmcu_irqfun.h, 277
HAL_UART_IrqLbd
 hal_lks32mc08x_irqfun.c, 595
 hal_lksmcu_irqfun.h, 277
HAL_UART_IrqLbdClear
 hal_lks32mc08x_irqfun.c, 595
 hal_lksmcu_irqfun.h, 277

- HAL_UART_IrqLbdDisable
 - hal_lks32mc08x_irqfun.c, 595
 - hal_lksmcu_irqfun.h, 278
- HAL_UART_IrqLbdEnable
 - hal_lks32mc08x_irqfun.c, 596
 - hal_lksmcu_irqfun.h, 278
- HAL_UART_IrqLbdFlg
 - hal_lks32mc08x_irqfun.c, 596
 - hal_lksmcu_irqfun.h, 278
- HAL_UART_IrqRxDone
 - hal_lks32mc08x_irqfun.c, 596
 - hal_lksmcu_irqfun.h, 278
- HAL_UART_IrqRxDoneClear
 - hal_lks32mc08x_irqfun.c, 596
 - hal_lksmcu_irqfun.h, 278
- HAL_UART_IrqRxDoneDisable
 - hal_lks32mc08x_irqfun.c, 597
 - hal_lksmcu_irqfun.h, 279
- HAL_UART_IrqRxDoneEnable
 - hal_lks32mc08x_irqfun.c, 597
 - hal_lksmcu_irqfun.h, 279
- HAL_UART_IrqRxDoneFlg
 - hal_lks32mc08x_irqfun.c, 597
 - hal_lksmcu_irqfun.h, 279
- HAL_UART_IrqRxOv
 - hal_lks32mc08x_irqfun.c, 598
 - hal_lksmcu_irqfun.h, 280
- HAL_UART_IrqRxOvClear
 - hal_lks32mc08x_irqfun.c, 598
 - hal_lksmcu_irqfun.h, 280
- HAL_UART_IrqRxOvDisable
 - hal_lks32mc08x_irqfun.c, 598
 - hal_lksmcu_irqfun.h, 280
- HAL_UART_IrqRxOvEnable
 - hal_lks32mc08x_irqfun.c, 598
 - hal_lksmcu_irqfun.h, 280
- HAL_UART_IrqRxOvFlg
 - hal_lks32mc08x_irqfun.c, 598
 - hal_lksmcu_irqfun.h, 280
- HAL_UART_IrqStopErr
 - hal_lks32mc08x_irqfun.c, 598
 - hal_lksmcu_irqfun.h, 280
- HAL_UART_IrqStopErrClear
 - hal_lks32mc08x_irqfun.c, 599
 - hal_lksmcu_irqfun.h, 281
- HAL_UART_IrqStopErrDisable
 - hal_lks32mc08x_irqfun.c, 599
 - hal_lksmcu_irqfun.h, 281
- HAL_UART_IrqStopErrEnable
 - hal_lks32mc08x_irqfun.c, 599
 - hal_lksmcu_irqfun.h, 281
- HAL_UART_IrqStopErrFlg
 - hal_lks32mc08x_irqfun.c, 600
 - hal_lksmcu_irqfun.h, 282
- HAL_UART_IrqTxBufEmpty
 - hal_lks32mc08x_irqfun.c, 600
 - hal_lksmcu_irqfun.h, 282
- HAL_UART_IrqTxBufEmptyClear
 - hal_lks32mc08x_irqfun.c, 600
 - hal_lksmcu_irqfun.h, 282
- HAL_UART_IrqTxBufEmptyDisable
 - hal_lks32mc08x_irqfun.c, 601
 - hal_lksmcu_irqfun.h, 283
- HAL_UART_IrqTxBufEmptyEnable
 - hal_lks32mc08x_irqfun.c, 601
 - hal_lksmcu_irqfun.h, 283
- HAL_UART_IrqTxBufEmptyFlg
 - hal_lks32mc08x_irqfun.c, 601
 - hal_lksmcu_irqfun.h, 283
- HAL_UART_IrqTxDone
 - hal_lks32mc08x_irqfun.c, 602
 - hal_lksmcu_irqfun.h, 284
- HAL_UART_IrqTxDoneClear
 - hal_lks32mc08x_irqfun.c, 602
 - hal_lksmcu_irqfun.h, 284
- HAL_UART_IrqTxDoneDisable
 - hal_lks32mc08x_irqfun.c, 602
 - hal_lksmcu_irqfun.h, 284
- HAL_UART_IrqTxDoneEnable
 - hal_lks32mc08x_irqfun.c, 603
 - hal_lksmcu_irqfun.h, 285
- HAL_UART_IrqTxDoneFlg
 - hal_lks32mc08x_irqfun.c, 603
 - hal_lksmcu_irqfun.h, 285
- HAL_UART_IrqTxOv
 - hal_lks32mc08x_irqfun.c, 603
 - hal_lksmcu_irqfun.h, 285
- HAL_UART_IrqTxOvClear
 - hal_lks32mc08x_irqfun.c, 603
 - hal_lksmcu_irqfun.h, 285
- HAL_UART_IrqTxOvDisable
 - hal_lks32mc08x_irqfun.c, 604
 - hal_lksmcu_irqfun.h, 286
- HAL_UART_IrqTxOvEnable
 - hal_lks32mc08x_irqfun.c, 604
 - hal_lksmcu_irqfun.h, 286
- HAL_UART_IrqTxOvFlg
 - hal_lks32mc08x_irqfun.c, 604
 - hal_lksmcu_irqfun.h, 286
- HAL_UART_MODE_LSB_STOP1_EVEN
 - hal_lksmcu_uart.h, 371
- HAL_UART_MODE_LSB_STOP1_NONE
 - hal_lksmcu_uart.h, 371
- HAL_UART_MODE_LSB_STOP1_ODD
 - hal_lksmcu_uart.h, 371
- HAL_UART_MODE_LSB_STOP2_EVEN
 - hal_lksmcu_uart.h, 371
- HAL_UART_MODE_LSB_STOP2_NONE
 - hal_lksmcu_uart.h, 371
- HAL_UART_MODE_LSB_STOP2_ODD
 - hal_lksmcu_uart.h, 371
- HAL_UART_MODE_MSB_STOP1_EVEN
 - hal_lksmcu_uart.h, 371
- HAL_UART_MODE_MSB_STOP1_NONE
 - hal_lksmcu_uart.h, 371
- HAL_UART_MODE_MSB_STOP1_ODD
 - hal_lksmcu_uart.h, 371

- hal_lksmcu_uart.h, 371
- HAL_UART_MODE_MSB_STOP2_EVEN
 - hal_lksmcu_uart.h, 371
- HAL_UART_MODE_MSB_STOP2_NONE
 - hal_lksmcu_uart.h, 371
- HAL_UART_MODE_MSB_STOP2_ODD
 - hal_lksmcu_uart.h, 371
- HAL_UART_MODE_x
 - hal_lksmcu_uart.h, 371
- HAL_UART_POL_RX_NEG_TX_NEG
 - hal_lksmcu_uart.h, 371
- HAL_UART_POL_RX_NEG_TX_POS
 - hal_lksmcu_uart.h, 371
- HAL_UART_POL_RX_POS_TX_NEG
 - hal_lksmcu_uart.h, 371
- HAL_UART_POL_RX_POS_TX_POS
 - hal_lksmcu_uart.h, 371
- HAL_UART_POL_x
 - hal_lksmcu_uart.h, 371
- HAL_UART_ReadBuffer
 - hal_lks32mc08x_uart.c, 727
 - hal_lksmcu_uart.h, 373
- HAL_UART_Reset
 - hal_lks32mc08x_uart.c, 727
 - hal_lksmcu_uart.h, 373
- HAL_UART_SetBaud
 - hal_lks32mc08x_uart.c, 727
 - hal_lksmcu_uart.h, 374
- HAL_UART_SetClockDiv
 - hal_lks32mc08x_uart.c, 728
 - hal_lksmcu_uart.h, 374
- HAL_UART_SetGpioRx
 - hal_lks32mc08x_uart.c, 728
 - hal_lksmcu_uart.h, 374
- HAL_UART_SetGpioTx
 - hal_lks32mc08x_uart.c, 728
 - hal_lksmcu_uart.h, 374
- HAL_UART_SetMode
 - hal_lks32mc08x_uart.c, 729
 - hal_lksmcu_uart.h, 375
- HAL_UART_SetPolarity
 - hal_lks32mc08x_uart.c, 729
 - hal_lksmcu_uart.h, 375
- HAL_UART_TxData
 - hal_lks32mc08x_uart.c, 730
 - hal_lksmcu_uart.h, 376
- HAL_UART_TypeDef, 51
 - ADR, 52
 - BUFF, 52
 - CTRL, 52
 - DIVH, 52
 - DIVL, 52
 - IE, 52
 - IF, 53
 - IOC, 53
 - STT, 53
- HAL_UART_WriteBuffer
 - hal_lks32mc08x_uart.c, 730
- hal_lksmcu_uart.h, 376
- HAL_UART_x
 - hal_lksmcu_uart.h, 372
- HAL_WAKE_DeepSleepIoTimeWakeUp
 - hal_lks32mc08x_wake.c, 735
 - hal_lksmcu_wake.h, 378
- HAL_WAKE_DeepSleepIoWake
 - hal_lks32mc08x_wake.c, 735
 - hal_lksmcu_wake.h, 379
- HAL_WAKE_DeepSleepTimeWakeUp
 - hal_lks32mc08x_wake.c, 736
 - hal_lksmcu_wake.h, 379
- HAL_WAKE_UP_IO_P0_0_High
 - hal_lksmcu_wake.h, 378
- HAL_WAKE_UP_IO_P0_0_Low
 - hal_lksmcu_wake.h, 378
- HAL_WAKE_UP_IO_P0_1_High
 - hal_lksmcu_wake.h, 378
- HAL_WAKE_UP_IO_P0_1_Low
 - hal_lksmcu_wake.h, 378
- HAL_WAKE_UP_IO_P1_0_High
 - hal_lksmcu_wake.h, 378
- HAL_WAKE_UP_IO_P1_0_Low
 - hal_lksmcu_wake.h, 378
- HAL_WAKE_UP_IO_P1_1_High
 - hal_lksmcu_wake.h, 378
- HAL_WAKE_UP_IO_P1_1_Low
 - hal_lksmcu_wake.h, 378
- HAL_WAKE_UP_IO_x
 - hal_lksmcu_wake.h, 378
- IE
 - CMP_TypeDef, 6
 - HAL_ADC_TypeDef, 11
 - HAL_CMP_TypeDef, 20
 - HAL_SPI_TypeDef, 40
 - HAL_UART_TypeDef, 52
- IE0
 - HAL_MCPWM_TypeDef, 36
- IER
 - HAL_CAN_TypeDef, 14
- IF
 - CMP_TypeDef, 6
 - HAL_ADC_TypeDef, 11
 - HAL_CMP_TypeDef, 20
 - HAL_UART_TypeDef, 53
- IF0
 - HAL_MCPWM_TypeDef, 36
- include/hal_lks32mc08x_reg.h, 58, 65
- include/hal_lksmcu_adc.h, 68, 83
- include/hal_lksmcu_can.h, 84, 99
- include/hal_lksmcu_clk.h, 101, 104
- include/hal_lksmcu_cmp.h, 105, 112
- include/hal_lksmcu_crc.h, 114, 117
- include/hal_lksmcu_dac.h, 117, 124
- include/hal_lksmcu_dma.h, 125, 129
- include/hal_lksmcu_dsp.h, 130, 134
- include/hal_lksmcu_flash.h, 135, 144
- include/hal_lksmcu_gpio.h, 145, 157

include/hal_lksmcu_hall.h, 159, 167
 include/hal_lksmcu_i2c.h, 167, 175
 include/hal_lksmcu_init.h, 175, 178
 include/hal_lksmcu_irq.h, 178, 186
 include/hal_lksmcu_irqfun.h, 188, 286
 include/hal_lksmcu_iwdg.h, 291, 294
 include/hal_lksmcu_mcpwm.h, 294, 316
 include/hal_lksmcu_opa.h, 317, 322
 include/hal_lksmcu_qep.h, 323, 330
 include/hal_lksmcu_reg.h, 330, 333
 include/hal_lksmcu_reset.h, 333, 334
 include/hal_lksmcu_spi.h, 335, 349
 include/hal_lksmcu_timer.h, 350, 360
 include/hal_lksmcu_tmp.h, 360, 366
 include/hal_lksmcu_trim.h, 367, 369
 include/hal_lksmcu_uart.h, 370, 376
 include/hal_lksmcu_wake.h, 377, 380
 INFO
 HAL_HALL_TypeDef, 31
 INIT
 HAL_CRC_TypeDef, 21
 initValue
 CRC_Model_Params, 7
 IO01
 HAL_MCPWM_TypeDef, 36
 IO23
 HAL_MCPWM_TypeDef, 36
 IOC
 HAL_UART_TypeDef, 53
 IR
 HAL_CAN_TypeDef, 15
 LCKR
 HAL_GPIO_TypeDef, 28
 MOD
 HAL_CAN_TypeDef, 15
 HAL_DSP_TypeDef, 24
 MSCR
 HAL_I2C_TypeDef, 32
 null
 HAL_ADC_TypeDef, 11
 null0
 HAL_CAN_TypeDef, 15
 null1
 HAL_CAN_TypeDef, 15
 HAL_GPIO_TypeDef, 28
 HAL_MCPWM_TypeDef, 36
 null2
 HAL_CAN_TypeDef, 15
 HAL_MCPWM_TypeDef, 37
 null3
 HAL_CAN_TypeDef, 15
 HAL_MCPWM_TypeDef, 37
 PDI
 HAL_GPIO_TypeDef, 29
 PDO
 HAL_GPIO_TypeDef, 29
 PIE
 HAL_GPIO_TypeDef, 29
 PODE
 HAL_GPIO_TypeDef, 29
 POE
 HAL_GPIO_TypeDef, 29
 POL
 HAL_CRC_TypeDef, 21
 polynomial
 CRC_Model_Params, 7
 PP
 HAL_MCPWM_TypeDef, 37
 PROTECT
 HAL_FLASH_TypeDef, 26
 HAL_SYS_TypeDef, 47
 PRT
 HAL_MCPWM_TypeDef, 37
 PUE
 HAL_GPIO_TypeDef, 29
 RAD
 HAL_DSP_TypeDef, 24
 RBSA
 HAL_CAN_TypeDef, 15
 RDATA
 HAL_FLASH_TypeDef, 26
 RE
 HAL_MCPWM_TypeDef, 37
 readme.md, 381
 READY
 HAL_FLASH_TypeDef, 26
 reflectIn
 CRC_Model_Params, 7
 reflectOut
 CRC_Model_Params, 7
 RESERVED
 HAL_DSP_TypeDef, 24
 HAL_SYS_TypeDef, 47
 RESERVED1
 HAL_DSP_TypeDef, 24
 HAL_SYS_TypeDef, 47
 RESERVED2
 HAL_SYS_TypeDef, 47
 RESERVED3
 HAL_SYS_TypeDef, 47
 RESERVED4
 HAL_SYS_TypeDef, 47
 RESERVED5
 HAL_SYS_TypeDef, 47
 RMC
 HAL_CAN_TypeDef, 16
 RST_CFG
 HAL_SYS_TypeDef, 48
 RST_SRC
 HAL_SYS_TypeDef, 48
 RXDATA
 HAL_SPI_TypeDef, 40
 RXERR

HAL_CAN_TypeDef, 16
 SC
 HAL_DSP_TypeDef, 24
 SCR
 HAL_I2C_TypeDef, 32
 SDCFG
 HAL_MCPWM_TypeDef, 37
 SFT_RST
 HAL_SYS_TypeDef, 48
 SIN
 HAL_DSP_TypeDef, 24
 SIZE
 HAL_SPI_TypeDef, 41
 source/hal_iks32mc08x_adc.c, 381, 393
 source/hal_iks32mc08x_can.c, 397, 411
 source/hal_iks32mc08x_clk.c, 418, 421
 source/hal_iks32mc08x_cmp.c, 422, 428
 source/hal_iks32mc08x_crc.c, 430, 435
 source/hal_iks32mc08x_dac.c, 436, 443
 source/hal_iks32mc08x_dma.c, 444, 449
 source/hal_iks32mc08x_dsp.c, 450, 454
 source/hal_iks32mc08x_flash.c, 455, 465
 source/hal_iks32mc08x_gpio.c, 469, 476
 source/hal_iks32mc08x_hall.c, 480, 487
 source/hal_iks32mc08x_i2c.c, 489, 496
 source/hal_iks32mc08x_init.c, 499, 500
 source/hal_iks32mc08x_irq.c, 501, 505
 source/hal_iks32mc08x_irqfun.c, 505, 604
 source/hal_iks32mc08x_iwdg.c, 638, 641
 source/hal_iks32mc08x_mcpwm.c, 642, 661
 source/hal_iks32mc08x_opa.c, 667, 671
 source/hal_iks32mc08x_qep.c, 673, 678
 source/hal_iks32mc08x_reg.c, 679, 683
 source/hal_iks32mc08x_reset.c, 683, 685
 source/hal_iks32mc08x_spi.c, 685, 698
 source/hal_iks32mc08x_timer.c, 702, 711
 source/hal_iks32mc08x_tmp.c, 714, 720
 source/hal_iks32mc08x_trim.c, 721, 724
 source/hal_iks32mc08x_uart.c, 725, 730
 source/hal_iks32mc08x_wake.c, 734, 736
 SQRT
 HAL_DSP_TypeDef, 25
 SR
 HAL_CAN_TypeDef, 16
 STT
 HAL_UART_TypeDef, 53
 SWT
 HAL_ADC_TypeDef, 12
 TCLK
 CMP_TypeDef, 6
 HAL_CMP_TypeDef, 20
 HAL_MCPWM_TypeDef, 37
 TH
 HAL_HALL_TypeDef, 31
 HAL_MCPWM_TypeDef, 37
 HAL_QEP_TypeDef, 39
 HAL_TIMER_TypeDef, 51
 TH0
 HAL_MCPWM_TypeDef, 38
 THETA
 HAL_DSP_TypeDef, 25
 TMP_A
 HAL_SYS_TypeDef, 48
 TMP_B
 HAL_SYS_TypeDef, 48
 tmp_coef_a
 hal_iks32mc08x_tmp.c, 719
 tmp_coef_b
 hal_iks32mc08x_tmp.c, 719
 TMR0
 HAL_MCPWM_TypeDef, 38
 TMR1
 HAL_MCPWM_TypeDef, 38
 TMR2
 HAL_MCPWM_TypeDef, 38
 TMR3
 HAL_MCPWM_TypeDef, 38
 TRIG
 HAL_ADC_TypeDef, 12
 TRIM
 HAL_SYS_TypeDef, 48
 TXDATA
 HAL_SPI_TypeDef, 41
 TXERR
 HAL_CAN_TypeDef, 16
 TXRX0
 HAL_CAN_TypeDef, 16
 TXRX1
 HAL_CAN_TypeDef, 16
 TXRX2
 HAL_CAN_TypeDef, 17
 TXRX3
 HAL_CAN_TypeDef, 17
 TXRX4
 HAL_CAN_TypeDef, 17
 TXRX5
 HAL_CAN_TypeDef, 17
 TXRX6
 HAL_CAN_TypeDef, 17
 TXRX7
 HAL_CAN_TypeDef, 17
 TXRX8
 HAL_CAN_TypeDef, 18
 TXRX9
 HAL_CAN_TypeDef, 18
 TXRXA
 HAL_CAN_TypeDef, 18
 TXRXB
 HAL_CAN_TypeDef, 18
 TXRXC
 HAL_CAN_TypeDef, 18
 UPDATE
 HAL_MCPWM_TypeDef, 38
 WDATA

HAL_FLASH_TypeDef, [27](#)
WDT_CLR
 HAL_IWDG_TypeDef, [33](#)
 HAL_SYS_TypeDef, [49](#)
WDT_CNT
 HAL_SYS_TypeDef, [49](#)
WDT_PSW
 HAL_SYS_TypeDef, [49](#)
WDT_TH
 HAL_SYS_TypeDef, [49](#)
WIDTH
 HAL_HALL_TypeDef, [31](#)
width
 CRC_Model_Params, [7](#)

xorOut
 CRC_Model_Params, [7](#)
XY
 HAL_DSP_TypeDef, [25](#)