

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 DATA	6
3.1.2.4 IE	6
3.1.2.5 IF	6
3.1.2.6 TCLK	6
3.2 CRC_Model_Params结构体 参考	6
3.2.1 详细描述	7
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	8
3.3.1 详细描述	9
3.3.2 结构体成员变量说明	9
3.3.2.1 AMC0	9
3.3.2.2 AMC1	9
3.3.2.3 CFG	10
3.3.2.4 CHN0	10
3.3.2.5 CHN1	10
3.3.2.6 CHN2	10
3.3.2.7 CHNT	10
3.3.2.8 DAT	11
3.3.2.9 DC0	11
3.3.2.10 DC1	11
3.3.2.11 GAIN	11
3.3.2.12 GEN	11
3.3.2.13 HTH	12
3.3.2.14 IE	12
3.3.2.15 IF	12

3.3.2.16 LTH	12
3.3.2.17 RESERVED	12
3.3.2.18 RESERVED1	12
3.3.2.19 RESERVED2	13
3.3.2.20 RESERVED3	13
3.3.2.21 SWT	13
3.3.2.22 TRIG	13
3.4 HAL_AON_TypeDef结构体 参考	13
3.4.1 详细描述	14
3.4.2 结构体成员变量说明	14
3.4.2.1 BKP_REG0	14
3.4.2.2 BKP_REG1	14
3.4.2.3 EVT_RCD	15
3.4.2.4 IO_WAKE_EN	15
3.4.2.5 IO_WAKE_POL	15
3.4.2.6 PWR_CFG	15
3.5 HAL_CLU_TypeDef结构体 参考	15
3.5.1 详细描述	16
3.5.2 结构体成员变量说明	16
3.5.2.1 CF	16
3.5.2.2 EN	17
3.5.2.3 FN	17
3.5.2.4 IE	17
3.5.2.5 IF	17
3.5.2.6 MX	17
3.5.2.7 OUT	18
3.6 HAL_CMP_TypeDef结构体 参考	18
3.6.1 详细描述	18
3.6.2 结构体成员变量说明	19
3.6.2.1 BLCWIN	19
3.6.2.2 CFG	19
3.6.2.3 DATA	19
3.6.2.4 IE	19
3.6.2.5 IF	19
3.6.2.6 TCLK	20
3.7 HAL_CRC_TypeDef结构体 参考	20
3.7.1 详细描述	20
3.7.2 结构体成员变量说明	20
3.7.2.1 CR	20
3.7.2.2 DR	21
3.7.2.3 INIT	21
3.7.2.4 POL	21

3.8 HAL_DMA_TypeDef结构体 参考	21
3.8.1 详细描述	22
3.8.2 结构体成员变量说明	22
3.8.2.1 CCR	22
3.8.2.2 CTMS	22
3.8.2.3 DADR	22
3.8.2.4 REN	22
3.8.2.5 SADR	22
3.9 HAL_DMAMODULE_TypeDef结构体 参考	23
3.9.1 详细描述	23
3.9.2 结构体成员变量说明	23
3.9.2.1 CTRL	23
3.9.2.2 IE	23
3.9.2.3 IF	23
3.10 HAL_DSP_TypeDef结构体 参考	24
3.10.1 详细描述	24
3.10.2 结构体成员变量说明	24
3.10.2.1 DID	24
3.10.2.2 DID1	25
3.10.2.3 DIS	25
3.10.2.4 DIS1	25
3.10.2.5 QUO	25
3.10.2.6 QUO1	25
3.10.2.7 RAD	25
3.10.2.8 REM	26
3.10.2.9 RESERVED	26
3.10.2.10 RESERVED1	26
3.10.2.11 RESERVED2	26
3.10.2.12 SQRT	26
3.11 HAL_EXTI_TypeDef结构体 参考	26
3.11.1 详细描述	27
3.11.2 结构体成员变量说明	27
3.11.2.1 CLKO	27
3.11.2.2 CR0	27
3.11.2.3 CR1	28
3.11.2.4 IE	28
3.11.2.5 IF	28
3.11.2.6 SWAP	28
3.12 HAL_FLASH_TypeDef结构体 参考	28
3.12.1 详细描述	30
3.12.2 结构体成员变量说明	30
3.12.2.1 ADDR	30

3.12.2.2 CFG	30
3.12.2.3 EPSM	30
3.12.2.4 EPSN	31
3.12.2.5 ERASE	31
3.12.2.6 LIB_ADDR	31
3.12.2.7 NADDR	31
3.12.2.8 NCFG	31
3.12.2.9 NERASE	32
3.12.2.10 NKEY	32
3.12.2.11 NRDATA	32
3.12.2.12 NWDATA	32
3.12.2.13 PROTECT	32
3.12.2.14 RDATA	32
3.12.2.15 READY	33
3.12.2.16 RESERVED	33
3.12.2.17 SIZE	33
3.12.2.18 WDATA	33
3.13 HAL_GPIO_TypeDef结构体 参考	33
3.13.1 详细描述	35
3.13.2 结构体成员变量说明	35
3.13.2.1 BRR	35
3.13.2.2 BSRR	35
3.13.2.3 F3210	36
3.13.2.4 F7654	36
3.13.2.5 FBA98	36
3.13.2.6 FFEDC	36
3.13.2.7 PDI	36
3.13.2.8 PDO	36
3.13.2.9 PFLT	37
3.13.2.10 PIE	37
3.13.2.11 PODE	37
3.13.2.12 POE	37
3.13.2.13 PUE	38
3.13.2.14 RESERVED	38
3.14 HAL_HALL_TypeDef结构体 参考	38
3.14.1 详细描述	38
3.14.2 结构体成员变量说明	39
3.14.2.1 CFG	39
3.14.2.2 CNT	39
3.14.2.3 INFO	39
3.14.2.4 TH	39
3.14.2.5 WIDTH	39

3.15 HAL_I2C_TypeDef结构体 参考	40
3.15.1 详细描述	40
3.15.2 结构体成员变量说明	40
3.15.2.1 ADDR	40
3.15.2.2 BCR	41
3.15.2.3 BSIZE	41
3.15.2.4 CFG	41
3.15.2.5 DATA	41
3.15.2.6 MSCR	41
3.15.2.7 SCR	41
3.16 HAL_IWDG_TypeDef结构体 参考	42
3.16.1 详细描述	42
3.16.2 结构体成员变量说明	42
3.16.2.1 CFG	42
3.16.2.2 CLR	43
3.16.2.3 CNT	43
3.16.2.4 PSW	43
3.16.2.5 RTH	43
3.16.2.6 WTH	43
3.17 HAL_MCPWM_TypeDef结构体 参考	44
3.17.1 详细描述	46
3.17.2 结构体成员变量说明	46
3.17.2.1 AUEN	46
3.17.2.2 CH_DEF	46
3.17.2.3 CH_FAIL	47
3.17.2.4 CH_MSK	47
3.17.2.5 CNT0	47
3.17.2.6 DTH00	47
3.17.2.7 DTH01	47
3.17.2.8 EIE	47
3.17.2.9 EIF	48
3.17.2.10 EVT0	48
3.17.2.11 FCNT	48
3.17.2.12 FLT	48
3.17.2.13 IE0	48
3.17.2.14 IF0	49
3.17.2.15 IO01	49
3.17.2.16 IO23	49
3.17.2.17 PP	49
3.17.2.18 PRT	50
3.17.2.19 RE	50
3.17.2.20 RESERVED	50

3.17.2.21 RESERVED1	50
3.17.2.22 RESERVED2	50
3.17.2.23 RESERVED3	50
3.17.2.24 RESERVED4	51
3.17.2.25 RESERVED5	51
3.17.2.26 RESERVED6	51
3.17.2.27 RESERVED7	51
3.17.2.28 RESERVED8	51
3.17.2.29 SDCFG	51
3.17.2.30 SW_FAIL	51
3.17.2.31 TCLK	52
3.17.2.32 TH	52
3.17.2.33 TH0	52
3.17.2.34 TMR0	52
3.17.2.35 TMR1	52
3.17.2.36 TMR2	52
3.17.2.37 TMR3	53
3.17.2.38 UPDATE	53
3.18 HAL_QEP_TypeDef结构体 参考	53
3.18.1 详细描述	54
3.18.2 结构体成员变量说明	54
3.18.2.1 CFG	54
3.18.2.2 CNT	54
3.18.2.3 IE	54
3.18.2.4 IF	54
3.18.2.5 TH	55
3.19 HAL_SIF_TypeDef结构体 参考	55
3.19.1 详细描述	55
3.19.2 结构体成员变量说明	55
3.19.2.1 CFG	55
3.19.2.2 IRQ	56
3.19.2.3 TDTH1	56
3.19.2.4 TOSC	56
3.19.2.5 TSTH1	56
3.19.2.6 WDATA	56
3.20 HAL_SPI_TypeDef结构体 参考	56
3.20.1 详细描述	57
3.20.2 结构体成员变量说明	57
3.20.2.1 CFG	57
3.20.2.2 DIV	57
3.20.2.3 IE	57
3.20.2.4 RX_DATA	58

3.20.2.5 SIZE	58
3.20.2.6 TX_DATA	58
3.21 HAL_SYS_TypeDef结构体 参考	58
3.21.1 详细描述	60
3.21.2 结构体成员变量说明	60
3.21.2.1 AFE_ADC	60
3.21.2.2 AFE_DAC0	61
3.21.2.3 AFE_DAC0_AMC	61
3.21.2.4 AFE_DAC0_DC	61
3.21.2.5 AFE_DAC1	61
3.21.2.6 AFE_DAC1_AMC	61
3.21.2.7 AFE_DAC1_DC	61
3.21.2.8 AFE_DAC_CTRL	62
3.21.2.9 AFE_DBG	62
3.21.2.10 AFE_INFO	62
3.21.2.11 AFE_REG0	62
3.21.2.12 AFE_REG1	62
3.21.2.13 AFE_REG2	63
3.21.2.14 AFE_REG3	63
3.21.2.15 AFE_REG4	63
3.21.2.16 AFE_REG5	63
3.21.2.17 AFE_REG7	63
3.21.2.18 CLK_CFG	64
3.21.2.19 CLK_DIV0	64
3.21.2.20 CLK_DIV1	64
3.21.2.21 CLK_DIV2	64
3.21.2.22 CLK_FEN	64
3.21.2.23 DBG_CFG	64
3.21.2.24 FLSE	65
3.21.2.25 FLSP	65
3.21.2.26 IF	65
3.21.2.27 IO_CFG	65
3.21.2.28 NMI_TH	65
3.21.2.29 OPA_SEL	65
3.21.2.30 PROTECT	66
3.21.2.31 RAM_LOCK	66
3.21.2.32 RESERVED	66
3.21.2.33 RESERVED1	66
3.21.2.34 RESERVED2	66
3.21.2.35 RESERVED3	66
3.21.2.36 RESERVED41	67
3.21.2.37 RESERVED5	67

3.21.2.38 RESERVED6	67
3.21.2.39 RESERVED7	67
3.21.2.40 SFT_RST	67
3.22 HAL_TIMER_TypeDef结构体 参考	67
3.22.1 详细描述	68
3.22.2 结构体成员变量说明	69
3.22.2.1 CFG0	69
3.22.2.2 CFG1	69
3.22.2.3 CFG2	69
3.22.2.4 CHN0	69
3.22.2.5 CHN1	70
3.22.2.6 CNT	70
3.22.2.7 EVT	70
3.22.2.8 FLT	70
3.22.2.9 IE	70
3.22.2.10 IF	70
3.22.2.11 IO	71
3.22.2.12 RESERVED	71
3.22.2.13 TH	71
3.23 HAL_UART_TypeDef结构体 参考	71
3.23.1 详细描述	72
3.23.2 结构体成员变量说明	72
3.23.2.1 ADR	72
3.23.2.2 BUFF	72
3.23.2.3 CTRL	72
3.23.2.4 DIVH	72
3.23.2.5 DIVL	72
3.23.2.6 IE	73
3.23.2.7 IF	73
3.23.2.8 IOC	73
3.23.2.9 RE	73
3.23.2.10 STT	73
4 文件说明	75
4.1 hal_lks32mc09x.c 文件参考	75
4.1.1 宏定义说明	76
4.1.1.1 _HAL_LKS32MC05X_C_	76
4.2 hal_lks32mc09x.c	76
4.3 hal_lks32mc09x.h 文件参考	76
4.3.1 宏定义说明	78
4.3.1.1 HAL_ADC_IRQHandler	78
4.3.1.2 HAL_ADC_IRQHandler_00	79

4.3.1.3 HAL_ADC_IRQHandler_10	79
4.3.1.4 HAL_ADC_IRQHandler_20	79
4.3.1.5 HAL_CAN_IRQHandler	79
4.3.1.6 HAL_CAN_IRQHandler_00	79
4.3.1.7 HAL_CL_IRQHandler	79
4.3.1.8 HAL_CL_IRQHandler_00	79
4.3.1.9 HAL_CMP_IRQHandler	79
4.3.1.10 HAL_CMP_IRQHandler_00	80
4.3.1.11 HAL_CMP_IRQHandler_10	80
4.3.1.12 HAL_CMP_IRQHandler_20	80
4.3.1.13 HAL_CMP_IRQHandler_30	80
4.3.1.14 HAL_CMP_IRQHandler_40	80
4.3.1.15 HAL_CMP_IRQHandler_50	80
4.3.1.16 HAL_DMA_IRQHandler	80
4.3.1.17 HAL_DMA_IRQHandler_00	80
4.3.1.18 HAL_DSP_IRQHandler	81
4.3.1.19 HAL_DSP_IRQHandler_00	81
4.3.1.20 HAL_FMAC_IRQHandler_00	81
4.3.1.21 HAL_GPIO_IRQHandler	81
4.3.1.22 HAL_GPIO_IRQHandler_00	81
4.3.1.23 HAL_HALL_IRQHandler	81
4.3.1.24 HAL_HALL_IRQHandler_00	81
4.3.1.25 HAL_HALL_IRQHandler_10	81
4.3.1.26 HAL_I2C_IRQHandler	82
4.3.1.27 HAL_I2C_IRQHandler_00	82
4.3.1.28 HAL_I2C_IRQHandler_10	82
4.3.1.29 HAL_INIT_DEFAULE_MCLK	82
4.3.1.30 HAL_INIT_DEFAULE_MCLK_MS	82
4.3.1.31 HAL_INIT_DEFAULE_MCLK_US	82
4.3.1.32 HAL_INIT_IRQHandler	82
4.3.1.33 HAL_INIT_IRQHandler_00	83
4.3.1.34 HAL_MCPWM_IRQHandler	83
4.3.1.35 HAL_MCPWM_IRQHandler_00	83
4.3.1.36 HAL_MCPWM_IRQHandler_01	83
4.3.1.37 HAL_MCPWM_IRQHandler_10	83
4.3.1.38 HAL_MCPWM_IRQHandler_11	83
4.3.1.39 HAL_PWRDN_IRQHandler	83
4.3.1.40 HAL_PWRDN_IRQHandler_00	83
4.3.1.41 HAL_QEP_IRQHandler	84
4.3.1.42 HAL_QEP_IRQHandler_00	84
4.3.1.43 HAL_QEP_IRQHandler_10	84
4.3.1.44 HAL_QEP_IRQHandler_20	84

4.3.1.45 HAL_QEP_IRQHandler_30	84
4.3.1.46 HAL_SIF_IRQHandler	84
4.3.1.47 HAL_SIF_IRQHandler_00	84
4.3.1.48 HAL_SPI_IRQHandler	84
4.3.1.49 HAL_SPI_IRQHandler_00	85
4.3.1.50 HAL_SPI_IRQHandler_10	85
4.3.1.51 HAL_SW_IRQHandler	85
4.3.1.52 HAL_SW_IRQHandler_00	85
4.3.1.53 HAL_TIMER_IRQHandler	85
4.3.1.54 HAL_TIMER_IRQHandler_00	85
4.3.1.55 HAL_TIMER_IRQHandler_10	85
4.3.1.56 HAL_TIMER_IRQHandler_20	86
4.3.1.57 HAL_TIMER_IRQHandler_30	86
4.3.1.58 HAL_TIMER_IRQHandler_40	86
4.3.1.59 HAL_UART_IRQHandler	86
4.3.1.60 HAL_UART_IRQHandler_00	86
4.3.1.61 HAL_UART_IRQHandler_10	86
4.3.1.62 HAL_UART_IRQHandler_20	86
4.3.1.63 HAL_WAKE_IRQHandler	86
4.3.1.64 HAL_WAKE_IRQHandler_00	87
4.3.1.65 HAL_WWDG_IRQHandler_00	87
4.4 hal_lks32mc09x.h	87
4.5 hal_lksmcu.h 文件参考	88
4.6 hal_lksmcu.h	88
4.7 include/hal_lks32mc09x_reg.h 文件参考	89
4.7.1 宏定义说明	91
4.7.1.1 AON0	91
4.7.1.2 CLU0	91
4.7.1.3 DMA_CHN0	91
4.7.1.4 DMA_CHN1	91
4.7.1.5 DMA_CHN2	91
4.7.1.6 DMA_CHN3	91
4.7.1.7 DMA_TOP	91
4.7.1.8 EXTI0	92
4.7.1.9 LKS32MC09X	92
4.7.1.10 QEP0	92
4.7.1.11 SIF0	92
4.7.2 变量说明	92
4.7.2.1 HAL_ADC	92
4.7.2.2 HAL_CMP	92
4.7.2.3 HAL_CRC	92
4.7.2.4 HAL_DMA	93

4.7.2.5 HAL_DSP	93
4.7.2.6 HAL_FLASH	93
4.7.2.7 HAL_GPIO	93
4.7.2.8 HAL_HALL	93
4.7.2.9 HAL_I2C	93
4.7.2.10 HAL_IWDG	93
4.7.2.11 HAL_MCPWM	93
4.7.2.12 HAL_QEP	94
4.7.2.13 HAL_SPI	94
4.7.2.14 HAL_SYS	94
4.7.2.15 HAL_TIMER	94
4.7.2.16 HAL_UART	94
4.8 hal_lks32mc09x_reg.h	94
4.9 include/hal_lksmcu_adc.h 文件参考	98
4.9.1 枚举类型说明	101
4.9.1.1 HAL_ADC_PORT_x	101
4.9.1.2 HAL_ADC_REFERENCE_x	102
4.9.1.3 HAL_ADC_SAMP_NUMBER_x	102
4.9.1.4 HAL_ADC_SAMP_SEQUENCE_x	102
4.9.1.5 HAL_ADC_TRIGGER_x	103
4.9.1.6 HAL_ADC_x	103
4.9.2 函数说明	104
4.9.2.1 HAL_ADC_Disable()	104
4.9.2.2 HAL_ADC_Enable()	104
4.9.2.3 HAL_ADC_EndSampling()	104
4.9.2.4 HAL_ADC_GetRange()	105
4.9.2.5 HAL_ADC_GetSamplingValues()	106
4.9.2.6 HAL_ADC_GetValuesUnit()	106
4.9.2.7 HAL_ADC_ManualSampling()	107
4.9.2.8 HAL_ADC_Reset()	107
4.9.2.9 HAL_ADC_SetGpioIn()	108
4.9.2.10 HAL_ADC_SetGpioTriggerOut()	108
4.9.2.11 HAL_ADC_SetReference()	109
4.9.2.12 HAL_ADC_SetSamplingChannel()	109
4.9.2.13 HAL_ADC_SetSamplingQuantity()	110
4.9.2.14 HAL_ADC_SetTriggerSignal()	111
4.9.2.15 HAL_ADC_SoftTrigger()	112
4.10 hal_lksmcu_adc.h	112
4.11 include/hal_lksmcu_can.h 文件参考	115
4.12 hal_lksmcu_can.h	115
4.13 include/hal_lksmcu_clk.h 文件参考	115
4.13.1 枚举类型说明	116

4.13.1.1 HAL_CLK_FREQ_x	116
4.13.2 函数说明	117
4.13.2.1 HAL_CLK_GetMcuFreq()	117
4.13.2.2 HAL_CLK_SetMcuFreq()	118
4.14 hal_lksmcu_clk.h	119
4.15 include/hal_lksmcu_cmp.h 文件参考	120
4.15.1 枚举类型说明	121
4.15.1.1 HAL_CMP_CHANNEL_INx	121
4.15.1.2 HAL_CMP_CHANNEL_IPx	121
4.15.1.3 HAL_CMP_x	122
4.15.2 函数说明	123
4.15.2.1 HAL_CMP_Disable()	123
4.15.2.2 HAL_CMP_Enable()	123
4.15.2.3 HAL_CMP_GetData()	124
4.15.2.4 HAL_CMP_Reset()	124
4.15.2.5 HAL_CMP_SetChannel()	125
4.15.2.6 HAL_CMP_SetDigitalFilter()	125
4.15.2.7 HAL_CMP_SetGpioIn()	125
4.15.2.8 HAL_CMP_SetGpioOut()	126
4.16 hal_lksmcu_cmp.h	127
4.17 include/hal_lksmcu_crc.h 文件参考	128
4.17.1 枚举类型说明	129
4.17.1.1 HAL_CRC_MODEL_x	129
4.17.1.2 HAL_CRC_x	129
4.17.2 函数说明	129
4.17.2.1 HAL_CRC_Calc()	129
4.17.2.2 HAL_CRC_Disable()	130
4.17.2.3 HAL_CRC_Enable()	131
4.17.2.4 HAL_CRC_Reset()	131
4.18 hal_lksmcu_crc.h	131
4.19 include/hal_lksmcu_dac.h 文件参考	132
4.19.1 枚举类型说明	133
4.19.1.1 HAL_DAC_x	133
4.19.2 函数说明	133
4.19.2.1 HAL_DAC_Disable()	133
4.19.2.2 HAL_DAC_Enable()	133
4.19.2.3 HAL_DAC_Out2GpioDisable()	134
4.19.2.4 HAL_DAC_Out2GpioEnable()	134
4.19.2.5 HAL_DAC_Reset()	134
4.19.2.6 HAL_DAC_SetGpioOut()	134
4.19.2.7 HAL_DAC_SetOutVal()	135
4.19.2.8 HAL_DAC_SetOutVoltageV()	135

4.20 hal_lksmcu_dac.h	136
4.21 include/hal_lksmcu_dma.h 文件参考	136
4.21.1 枚举类型说明	137
4.21.1.1 HAL_DMA_x	137
4.21.2 函数说明	137
4.21.2.1 HAL_DMA_Disable()	137
4.21.2.2 HAL_DMA_Enable()	138
4.21.2.3 HAL_DMA_I2cRx()	138
4.21.2.4 HAL_DMA_I2cTx()	138
4.21.2.5 HAL_DMA_MemCopy()	138
4.21.2.6 HAL_DMA_Reset()	138
4.21.2.7 HAL_DMA_SpiRx()	138
4.21.2.8 HAL_DMA_SpiRxTx()	139
4.21.2.9 HAL_DMA_SpiTx()	139
4.21.2.10 HAL_DMA_UartRx()	139
4.21.2.11 HAL_DMA_UartTx()	139
4.22 hal_lksmcu_dma.h	140
4.23 include/hal_lksmcu_dsp.h 文件参考	140
4.23.1 枚举类型说明	141
4.23.1.1 HAL_DSP_x	141
4.23.2 函数说明	141
4.23.2.1 HAL_DSP_Cos()	141
4.23.2.2 HAL_DSP_Disable()	142
4.23.2.3 HAL_DSP_Div()	142
4.23.2.4 HAL_DSP_Enable()	142
4.23.2.5 HAL_DSP_Reset()	142
4.23.2.6 HAL_DSP_Sin()	143
4.23.2.7 HAL_DSP_SinCos()	143
4.23.2.8 HAL_DSP_Sqrt()	143
4.24 hal_lksmcu_dsp.h	144
4.25 include/hal_lksmcu_flash.h 文件参考	144
4.25.1 枚举类型说明	145
4.25.1.1 HAL_FLASH_x	145
4.25.2 函数说明	146
4.25.2.1 HAL_FLASH_Disable()	146
4.25.2.2 HAL_FLASH_DisableCache()	146
4.25.2.3 HAL_FLASH_Enable()	146
4.25.2.4 HAL_FLASH_EnableCache()	146
4.25.2.5 HAL_FLASH_Erase()	147
4.25.2.6 HAL_FLASH_Program()	147
4.25.2.7 HAL_FLASH_Read()	148
4.25.2.8 HAL_FLASH_ReadSectorSize()	149

4.25.2.9 HAL_FLASH_ReadUint16()	149
4.25.2.10 HAL_FLASH_ReadUint32()	149
4.25.2.11 HAL_FLASH_ReadUint8()	150
4.25.2.12 HAL_FLASH_Reset()	150
4.25.2.13 HAL_FLASH_SecurityParameterRead()	151
4.25.2.14 HAL_FLASH_SecurityParameterSave()	152
4.25.2.15 HAL_FLASH_Update()	152
4.26 hal_lksmcu_flash.h	153
4.27 include/hal_lksmcu_gpio.h 文件参考	153
4.27.1 枚举类型说明	156
4.27.1.1 HAL_GPIO_AFx	156
4.27.1.2 HAL_GPIO_MODE_x	157
4.27.1.3 HAL_GPIO_Px	157
4.27.2 函数说明	160
4.27.2.1 HAL_GPIO_Disable()	160
4.27.2.2 HAL_GPIO_Enable()	160
4.27.2.3 HAL_GPIO_GetInData()	160
4.27.2.4 HAL_GPIO_GetOutData()	160
4.27.2.5 HAL_GPIO_Invert()	161
4.27.2.6 HAL_GPIO_Reset()	161
4.27.2.7 HAL_GPIO_RstReuseGpioDisable()	161
4.27.2.8 HAL_GPIO_RstReuseGpioEnable()	162
4.27.2.9 HAL_GPIO_SetAf()	162
4.27.2.10 HAL_GPIO_SetHigh()	163
4.27.2.11 HAL_GPIO_SetLow()	164
4.27.2.12 HAL_GPIO_SetMode()	164
4.27.2.13 HAL_GPIO_SwdReuseGpioDisable()	165
4.27.2.14 HAL_GPIO_SwdReuseGpioEnable()	166
4.27.2.15 HAL_GPIO_Write()	166
4.28 hal_lksmcu_gpio.h	166
4.29 include/hal_lksmcu_hall.h 文件参考	168
4.29.1 枚举类型说明	170
4.29.1.1 HAL_HALL_x	170
4.29.2 函数说明	170
4.29.2.1 HAL_HALL_Disable()	170
4.29.2.2 HAL_HALL_Enable()	170
4.29.2.3 HAL_HALL_ReadHallData()	171
4.29.2.4 HAL_HALL_ReadHallWidth()	171
4.29.2.5 HAL_HALL_ReadTh()	171
4.29.2.6 HAL_HALL_Reset()	172
4.29.2.7 HAL_HALL_SetFilter()	172
4.29.2.8 HAL_HALL_SetFilteruS()	173

4.29.2.9 HAL_HALL_SetFreq()	173
4.29.2.10 HAL_HALL_SetGpio()	174
4.29.2.11 HAL_HALL_SetTh()	174
4.29.2.12 HAL_HALL_StartCounting()	175
4.29.2.13 HAL_HALL_StopCounting()	175
4.30 hal_lksmcu_hall.h	175
4.31 include/hal_lksmcu_i2c.h 文件参考	176
4.31.1 枚举类型说明	177
4.31.1.1 HAL_I2C_x	177
4.31.2 函数说明	177
4.31.2.1 HAL_I2C_BusReset()	177
4.31.2.2 HAL_I2C_Disable()	178
4.31.2.3 HAL_I2C_Enable()	178
4.31.2.4 HAL_I2C_Reset()	179
4.31.2.5 HAL_I2C_RxData()	179
4.31.2.6 HAL_I2C_SetBaud()	179
4.31.2.7 HAL_I2C_SetGpioScl()	180
4.31.2.8 HAL_I2C_SetGpioSda()	181
4.31.2.9 HAL_I2C_TxData()	181
4.32 hal_lksmcu_i2c.h	182
4.33 include/hal_lksmcu_init.h 文件参考	182
4.33.1 枚举类型说明	182
4.33.1.1 HAL_INIT_POWER_x	182
4.33.2 函数说明	183
4.33.2.1 HAL_INIT_PowerReady()	183
4.33.2.2 HAL_INIT_SysInit()	183
4.34 hal_lksmcu_init.h	183
4.35 include/hal_lksmcu_irq.h 文件参考	183
4.35.1 枚举类型说明	184
4.35.1.1 HAL_IRQ_MODULE_x	184
4.35.2 函数说明	185
4.35.2.1 HAL_IRQ_Disable()	185
4.35.2.2 HAL_IRQ_Enable()	186
4.35.2.3 HAL_IRQ_ModuleDisable()	186
4.35.2.4 HAL_IRQ_ModuleEnable()	186
4.35.2.5 HAL_IRQ_Reset()	187
4.36 hal_lksmcu_irq.h	187
4.37 include/hal_lksmcu_irqfun.h 文件参考	188
4.37.1 函数说明	189
4.37.1.1 HAL_ADC_IrqSamp1()	189
4.37.1.2 HAL_ADC_IrqSamp1Clear()	190
4.37.1.3 HAL_ADC_IrqSamp1Disable()	190

4.37.1.4 HAL_ADC_IrqSamp1Enable()	190
4.37.1.5 HAL_ADC_IrqSamp1Flg()	190
4.37.1.6 HAL_ADC_IrqSamp2()	191
4.37.1.7 HAL_ADC_IrqSamp2Clear()	191
4.37.1.8 HAL_ADC_IrqSamp2Disable()	191
4.37.1.9 HAL_ADC_IrqSamp2Enable()	192
4.37.1.10 HAL_ADC_IrqSamp2Flg()	192
4.37.1.11 HAL_HALL_IrqChange()	192
4.37.1.12 HAL_HALL_IrqChangeClear()	193
4.37.1.13 HAL_HALL_IrqChangeDisable()	193
4.37.1.14 HAL_HALL_IrqChangeEnable()	193
4.37.1.15 HAL_HALL_IrqChangeFlg()	194
4.37.1.16 HAL_HALL_IrqOver()	194
4.37.1.17 HAL_HALL_IrqOverClear()	194
4.37.1.18 HAL_HALL_IrqOverDisable()	195
4.37.1.19 HAL_HALL_IrqOverEnable()	195
4.37.1.20 HAL_HALL_IrqOverFlg()	195
4.38 hal_lksmcu_irqfun.h	195
4.39 include/hal_lksmcu_iwdg.h 文件参考	196
4.39.1 枚举类型说明	197
4.39.1.1 HAL_IWDG_x	197
4.39.2 函数说明	197
4.39.2.1 HAL_IWDG_Disable()	197
4.39.2.2 HAL_IWDG_Enable()	198
4.39.2.3 HAL_IWDG_Feed()	198
4.39.2.4 HAL_IWDG_Reset()	199
4.39.2.5 HAL_IWDG_SetResetTimemS()	199
4.40 hal_lksmcu_iwdg.h	199
4.41 include/hal_lksmcu_mcpwm.h 文件参考	200
4.41.1 枚举类型说明	202
4.41.1.1 HAL_MCPWM_DEFAULT_OUTPUT_STATE_x	202
4.41.1.2 HAL_MCPWM_FAIL_x	202
4.41.1.3 HAL_MCPWM_OUTPUT_STATE_x	202
4.41.1.4 HAL_MCPWM_POLARITY_x	203
4.41.1.5 HAL_MCPWM_PORT_x	203
4.41.1.6 HAL_MCPWM_x	204
4.41.2 函数说明	204
4.41.2.1 HAL_MCPWM_Disable()	204
4.41.2.2 HAL_MCPWM_Enable()	204
4.41.2.3 HAL_MCPWM_GetFreq()	204
4.41.2.4 HAL_MCPWM_GetThCnt()	206
4.41.2.5 HAL_MCPWM_IrqFail()	207

4.41.2.6 HAL_MCPWM_IrqFailClear()	207
4.41.2.7 HAL_MCPWM_IrqFailDisable()	207
4.41.2.8 HAL_MCPWM_IrqFailEnable()	207
4.41.2.9 HAL_MCPWM_IrqOverflow()	207
4.41.2.10 HAL_MCPWM_IrqOverflowClear()	207
4.41.2.11 HAL_MCPWM_IrqOverflowDisable()	207
4.41.2.12 HAL_MCPWM_IrqOverflowEnable()	208
4.41.2.13 HAL_MCPWM_IrqZeroCross()	208
4.41.2.14 HAL_MCPWM_IrqZeroCrossClear()	208
4.41.2.15 HAL_MCPWM_IrqZeroCrossDisable()	208
4.41.2.16 HAL_MCPWM_IrqZeroCrossEnable()	208
4.41.2.17 HAL_MCPWM_Reset()	208
4.41.2.18 HAL_MCPWM_SetAdcTrig()	209
4.41.2.19 HAL_MCPWM_SetAdcTrigQ15()	209
4.41.2.20 HAL_MCPWM_SetDead()	209
4.41.2.21 HAL_MCPWM_SetDefaultState()	210
4.41.2.22 HAL_MCPWM_SetDrivePolarity()	210
4.41.2.23 HAL_MCPWM_SetFailSignal()	211
4.41.2.24 HAL_MCPWM_SetFreq()	211
4.41.2.25 HAL_MCPWM_SetGpioFail()	212
4.41.2.26 HAL_MCPWM_SetGpioOut()	212
4.41.2.27 HAL_MCPWM_SetOutputDisable()	213
4.41.2.28 HAL_MCPWM_SetOutputEnable()	213
4.41.2.29 HAL_MCPWM_SetOutState()	214
4.41.2.30 HAL_MCPWM_SetPwm()	214
4.41.2.31 HAL_MCPWM_SetPwmQ15()	215
4.41.2.32 HAL_MCPWM_StartCounting()	215
4.41.2.33 HAL_MCPWM_StopCounting()	216
4.42 hal_lksmcu_mcpwm.h	216
4.43 include/hal_lksmcu_opa.h 文件参考	217
4.43.1 宏定义说明	219
4.43.1.1 HAL_OPA_OPAX_2_ADCCHN	219
4.43.1.2 HAL_OPA_READ_GAIN	219
4.43.2 枚举类型说明	219
4.43.2.1 HAL_OPA_GAIN_x	219
4.43.2.2 HAL_OPA_x	220
4.43.3 函数说明	221
4.43.3.1 HAL_OPA_Disable()	221
4.43.3.2 HAL_OPA_Enable()	221
4.43.3.3 HAL_OPA_Out2GpioDisable()	221
4.43.3.4 HAL_OPA_Out2GpioEnable()	221
4.43.3.5 HAL_OPA_Reset()	222

4.43.3.6 HAL_OPA_SetGain()	222
4.43.3.7 HAL_OPA_SetGpioIn()	222
4.43.3.8 HAL_OPA_SetGpioOut()	223
4.44 hal_lksmcu_opa.h	223
4.45 include/hal_lksmcu_qep.h 文件参考	224
4.45.1 枚举类型说明	226
4.45.1.1 HAL_QEP_MODE_x	226
4.45.1.2 HAL_QEP_x	226
4.45.2 函数说明	226
4.45.2.1 HAL_QEP_Disable()	226
4.45.2.2 HAL_QEP_Enable()	227
4.45.2.3 HAL_QEP_ReadCnt()	227
4.45.2.4 HAL_QEP_Reset()	227
4.45.2.5 HAL_QEP_ResetCnt()	228
4.45.2.6 HAL_QEP_SetGpio()	228
4.45.2.7 HAL_QEP_SetMode()	229
4.45.2.8 HAL_QEP_SetTh()	229
4.46 hal_lksmcu_qep.h	229
4.47 include/hal_lksmcu_reg.h 文件参考	230
4.47.1 详细描述	230
4.47.2 宏定义说明	230
4.47.2.1 HAL_REG_IF_WRITE	230
4.47.2.2 HAL_REG_INVERT	231
4.47.2.3 HAL_REG_READ	231
4.47.2.4 HAL_REG_READ_BIT	231
4.47.2.5 HAL_REG_READ_BIT_POS_MASK	231
4.47.2.6 HAL_REG_RESET	231
4.47.2.7 HAL_REG_SET	231
4.47.2.8 HAL_REG_SYS_LOCK	232
4.47.2.9 HAL_REG_SYS_UNLOCK	232
4.47.2.10 HAL_REG_WRITE	232
4.47.2.11 HAL_REG_WRITE_BIT	232
4.47.2.12 HAL_REG_WRITE_BIT_MASK_POS	232
4.48 hal_lksmcu_reg.h	232
4.49 include/hal_lksmcu_reset.h 文件参考	233
4.49.1 函数说明	233
4.49.1.1 HAL_RESET_ResetCore()	233
4.49.1.2 HAL_RESET_ResetGlobal()	233
4.49.1.3 HAL_RESET_ResetRegister()	234
4.50 hal_lksmcu_reset.h	234
4.51 include/hal_lksmcu_spi.h 文件参考	234
4.51.1 枚举类型说明	236

4.51.1.1 HAL_SPI_MODE_x	236
4.51.1.2 HAL_SPI_x	236
4.51.2 函数说明	237
4.51.2.1 HAL_SPI_Disable()	237
4.51.2.2 HAL_SPI_Enable()	237
4.51.2.3 HAL_SPI_Reset()	237
4.51.2.4 HAL_SPI_RxData()	237
4.51.2.5 HAL_SPI_RxDataWidth16()	238
4.51.2.6 HAL_SPI_RxTxData()	238
4.51.2.7 HAL_SPI_RxTxDataWidth16()	239
4.51.2.8 HAL_SPI_SetBaud()	239
4.51.2.9 HAL_SPI_SetGpioClk()	240
4.51.2.10 HAL_SPI_SetGpioCs()	241
4.51.2.11 HAL_SPI_SetGpioDi()	241
4.51.2.12 HAL_SPI_SetGpioDo()	242
4.51.2.13 HAL_SPI_SetGpioMiso()	242
4.51.2.14 HAL_SPI_SetGpioMosi()	243
4.51.2.15 HAL_SPI_SetMode()	244
4.51.2.16 HAL_SPI_TxData()	244
4.51.2.17 HAL_SPI_TxDataWidth16()	245
4.52 hal_lksmcu_spi.h	245
4.53 include/hal_lksmcu_timer.h 文件参考	246
4.53.1 枚举类型说明	247
4.53.1.1 HAL_TIMER_MODE_x	247
4.53.1.2 HAL_TIMER_x	248
4.53.2 函数说明	248
4.53.2.1 HAL_TIMER_Disable()	248
4.53.2.2 HAL_TIMER_Enable()	248
4.53.2.3 HAL_TIMER_GetFreq()	249
4.53.2.4 HAL_TIMER_GetThCnt()	249
4.53.2.5 HAL_TIMER_IrqCompare0()	249
4.53.2.6 HAL_TIMER_IrqCompare0Clear()	249
4.53.2.7 HAL_TIMER_IrqCompare0Disable()	250
4.53.2.8 HAL_TIMER_IrqCompare0Enable()	250
4.53.2.9 HAL_TIMER_IrqCompare1()	250
4.53.2.10 HAL_TIMER_IrqCompare1Clear()	250
4.53.2.11 HAL_TIMER_IrqCompare1Disable()	250
4.53.2.12 HAL_TIMER_IrqCompare1Enable()	250
4.53.2.13 HAL_TIMER_IrqOverflow()	250
4.53.2.14 HAL_TIMER_IrqOverflowClear()	251
4.53.2.15 HAL_TIMER_IrqOverflowDisable()	251
4.53.2.16 HAL_TIMER_IrqOverflowEnable()	251

4.53.2.17 HAL_TIMER_Reset()	251
4.53.2.18 HAL_TIMER_SetFreq()	251
4.53.2.19 HAL_TIMER_SetGpioIn()	252
4.53.2.20 HAL_TIMER_SetGpioOut()	252
4.53.2.21 HAL_TIMER_SetPeriodCount()	253
4.53.2.22 HAL_TIMER_SetPeriodmS()	254
4.53.2.23 HAL_TIMER_SetPerioduS()	254
4.53.2.24 HAL_TIMER_SetPwm()	255
4.53.2.25 HAL_TIMER_SetPwmQ15()	255
4.53.2.26 HAL_TIMER_SetWorkMode()	256
4.53.2.27 HAL_TIMER_StartCounting()	256
4.53.2.28 HAL_TIMER_StopCounting()	256
4.54 hal_lksmcu_timer.h	256
4.55 include/hal_lksmcu_tmp.h 文件参考	257
4.55.1 枚举类型说明	258
4.55.1.1 HAL_TMP_x	258
4.55.2 函数说明	258
4.55.2.1 HAL_TMP_Adc2Temperature()	258
4.55.2.2 HAL_TMP_Adc2Temperature01()	259
4.55.2.3 HAL_TMP_Disable()	260
4.55.2.4 HAL_TMP_Enable()	260
4.55.2.5 HAL_TMP_GetTemperature()	261
4.55.2.6 HAL_TMP_GetTemperature01()	261
4.55.2.7 HAL_TMP_Reset()	262
4.56 hal_lksmcu_trim.h	263
4.57 include/hal_lksmcu_trim.h 文件参考	263
4.57.1 函数说明	264
4.57.1.1 HAL_TRIM_Read()	264
4.57.1.2 HAL_TRIM_ReadId()	265
4.57.1.3 HAL_TRIM_ReadIdCrc32()	265
4.58 hal_lksmcu_trim.h	266
4.59 include/hal_lksmcu_uart.h 文件参考	266
4.59.1 枚举类型说明	267
4.59.1.1 HAL_UART_MODE_x	267
4.59.1.2 HAL_UART_x	268
4.59.2 函数说明	268
4.59.2.1 HAL_UART_Disable()	268
4.59.2.2 HAL_UART_Enable()	268
4.59.2.3 HAL_UART_IrqCheckErr()	268
4.59.2.4 HAL_UART_IrqCheckErrClear()	269
4.59.2.5 HAL_UART_IrqCheckErrDisable()	269
4.59.2.6 HAL_UART_IrqCheckErrEnable()	269

4.59.2.7 HAL_UART_IrqReceiveOver()	269
4.59.2.8 HAL_UART_IrqReceiveOverClear()	269
4.59.2.9 HAL_UART_IrqReceiveOverDisable()	269
4.59.2.10 HAL_UART_IrqReceiveOverEnable()	269
4.59.2.11 HAL_UART_IrqSendOver()	270
4.59.2.12 HAL_UART_IrqSendOverClear()	270
4.59.2.13 HAL_UART_IrqSendOverDisable()	270
4.59.2.14 HAL_UART_IrqSendOverEnable()	270
4.59.2.15 HAL_UART_IrqStopErr()	270
4.59.2.16 HAL_UART_IrqStopErrClear()	270
4.59.2.17 HAL_UART_IrqStopErrDisable()	270
4.59.2.18 HAL_UART_IrqStopErrEnable()	271
4.59.2.19 HAL_UART_ReadBuffer()	271
4.59.2.20 HAL_UART_Reset()	271
4.59.2.21 HAL_UART_SetBaud()	271
4.59.2.22 HAL_UART_SetGpioRx()	272
4.59.2.23 HAL_UART_SetGpioTx()	272
4.59.2.24 HAL_UART_SetMode()	273
4.59.2.25 HAL_UART_TxData()	273
4.59.2.26 HAL_UART_WriteBuffer()	273
4.60 hal_lksmcu_uart.h	274
4.61 include/hal_lksmcu_wake.h 文件参考	275
4.61.1 枚举类型说明	275
4.61.1.1 HAL_WAKE_UP_IO_x	275
4.61.2 函数说明	276
4.61.2.1 HAL_WAKE_DeepSleepIoTimeWakeUp()	276
4.61.2.2 HAL_WAKE_DeepSleepIoWake()	277
4.61.2.3 HAL_WAKE_DeepSleepTimeWakeUp()	277
4.62 hal_lksmcu_wake.h	277
4.63 readme.md 文件参考	278
4.64 source/hal_lks32mc09x_adc.c 文件参考	278
4.64.1 详细描述	279
4.64.2 函数说明	280
4.64.2.1 HAL_ADC_Disable()	280
4.64.2.2 HAL_ADC_Enable()	281
4.64.2.3 HAL_ADC_EndSampling()	281
4.64.2.4 HAL_ADC_GetRange()	281
4.64.2.5 HAL_ADC_GetSamplingValues()	282
4.64.2.6 HAL_ADC_GetValuesUnit()	283
4.64.2.7 HAL_ADC_ManualSampling()	283
4.64.2.8 HAL_ADC_Reset()	284
4.64.2.9 HAL_ADC_SetGpioIn()	284

4.64.2.10 HAL_ADC_SetGpioTriggerOut()	285
4.64.2.11 HAL_ADC_SetReference()	286
4.64.2.12 HAL_ADC_SetSamplingChannel()	286
4.64.2.13 HAL_ADC_SetSamplingQuantity()	287
4.64.2.14 HAL_ADC_SetTriggerSignal()	288
4.64.2.15 HAL_ADC_SoftTrigger()	289
4.64.2.16 HAL_ADC_UpdateTrig()	289
4.64.3 变量说明	290
4.64.3.1 adc_soft_reg	290
4.65 hal_lks32mc09x_adc.c	290
4.66 source/hal_lks32mc09x_clk.c 文件参考	294
4.66.1 函数说明	295
4.66.1.1 HAL_CLK_GetMcuFreq()	295
4.66.1.2 HAL_CLK_SetMcuFreq()	296
4.66.2 变量说明	297
4.66.2.1 HAL_CLK_mclk	297
4.67 hal_lks32mc09x_clk.c	298
4.68 source/hal_lks32mc09x_cmp.c 文件参考	299
4.68.1 函数说明	300
4.68.1.1 HAL_CMP_Disable()	300
4.68.1.2 HAL_CMP_Enable()	301
4.68.1.3 HAL_CMP_GetData()	301
4.68.1.4 HAL_CMP_Reset()	301
4.68.1.5 HAL_CMP_SetChannel()	302
4.68.1.6 HAL_CMP_SetDigitalFilter()	302
4.68.1.7 HAL_CMP_SetGpioIn()	303
4.68.1.8 HAL_CMP_SetGpioOut()	303
4.69 hal_lks32mc09x_cmp.c	304
4.70 source/hal_lks32mc09x_crc.c 文件参考	306
4.70.1 枚举类型说明	307
4.70.1.1 CRC_Width	307
4.70.2 函数说明	308
4.70.2.1 HAL_CRC_Calc()	308
4.70.2.2 HAL_CRC_Disable()	309
4.70.2.3 HAL_CRC_Enable()	309
4.70.2.4 HAL_CRC_Reflect()	309
4.70.2.5 HAL_CRC_Reset()	310
4.70.3 变量说明	310
4.70.3.1 crcModels	310
4.71 hal_lks32mc09x_crc.c	311
4.72 source/hal_lks32mc09x_dac.c 文件参考	312
4.72.1 函数说明	313

4.72.1.1 HAL_DAC_Disable()	313
4.72.1.2 HAL_DAC_Enable()	314
4.72.1.3 HAL_DAC_Out2GpioDisable()	314
4.72.1.4 HAL_DAC_Out2GpioEnable()	314
4.72.1.5 HAL_DAC_Reset()	314
4.72.1.6 HAL_DAC_SetGpioOut()	315
4.72.1.7 HAL_DAC_SetOutVal()	315
4.72.1.8 HAL_DAC_SetOutVoltageV()	316
4.73 hal_lks32mc09x_dac.c	316
4.74 source/hal_lks32mc09x_dma.c 文件参考	318
4.74.1 函数说明	319
4.74.1.1 HAL_DMA_Disable()	319
4.74.1.2 HAL_DMA_Enable()	319
4.74.1.3 HAL_DMA_I2cRx()	319
4.74.1.4 HAL_DMA_I2cTx()	319
4.74.1.5 HAL_DMA_MemCopy()	320
4.74.1.6 HAL_DMA_Reset()	320
4.74.1.7 HAL_DMA_SpiRx()	320
4.74.1.8 HAL_DMA_SpiRxTx()	320
4.74.1.9 HAL_DMA_SpiTx()	320
4.74.1.10 HAL_DMA_UartRx()	321
4.74.1.11 HAL_DMA_UartTx()	321
4.74.2 变量说明	321
4.74.2.1 HAL_DMA_ena	321
4.75 hal_lks32mc09x_dma.c	321
4.76 source/hal_lks32mc09x_dsp.c 文件参考	323
4.76.1 函数说明	324
4.76.1.1 HAL_DSP_Cos()	324
4.76.1.2 HAL_DSP_Disable()	324
4.76.1.3 HAL_DSP_Div()	324
4.76.1.4 HAL_DSP_Enable()	324
4.76.1.5 HAL_DSP_Reset()	325
4.76.1.6 HAL_DSP_Sin()	325
4.76.1.7 HAL_DSP_SinCos()	325
4.76.1.8 HAL_DSP_Sqrt()	326
4.77 hal_lks32mc09x_dsp.c	326
4.78 source/hal_lks32mc09x_flash.c 文件参考	327
4.78.1 函数说明	329
4.78.1.1 HAL_FLASH_Disable()	329
4.78.1.2 HAL_FLASH_DisableCache()	329
4.78.1.3 HAL_FLASH_Enable()	329
4.78.1.4 HAL_FLASH_EnableCache()	330

4.78.1.5 HAL_FLASH.Erase()	330
4.78.1.6 HAL_FLASH.Program()	331
4.78.1.7 HAL_FLASH.Read()	331
4.78.1.8 HAL_FLASH.ReadSectorSize()	332
4.78.1.9 HAL_FLASH.ReadUint16()	332
4.78.1.10 HAL_FLASH.ReadUint32()	333
4.78.1.11 HAL_FLASH.ReadUint8()	333
4.78.1.12 HAL_FLASH.Reset()	334
4.78.1.13 HAL_FLASH.SecurityParameterCrc()	334
4.78.1.14 HAL_FLASH.SecurityParameterRead()	335
4.78.1.15 HAL_FLASH.SecurityParameterSave()	335
4.78.1.16 HAL_FLASH.Update()	336
4.78.2 变量说明	337
4.78.2.1 HAL_FLASH.data	337
4.79 hal_iks32mc09x_flash.c	337
4.80 source/hal_iks32mc09x_gpio.c 文件参考	341
4.80.1 函数说明	342
4.80.1.1 HAL_GPIO.Disable()	342
4.80.1.2 HAL_GPIO.Enable()	342
4.80.1.3 HAL_GPIO.GetInData()	342
4.80.1.4 HAL_GPIO.GetOutData()	343
4.80.1.5 HAL_GPIO.Invert()	343
4.80.1.6 HAL_GPIO.Reset()	344
4.80.1.7 HAL_GPIO.RstReuseGpioDisable()	344
4.80.1.8 HAL_GPIO.RstReuseGpioEnable()	344
4.80.1.9 HAL_GPIO.SetAf()	344
4.80.1.10 HAL_GPIO.SetHigh()	345
4.80.1.11 HAL_GPIO.SetLow()	346
4.80.1.12 HAL_GPIO.SetMode()	346
4.80.1.13 HAL_GPIO.SwdReuseGpioDisable()	347
4.80.1.14 HAL_GPIO.SwdReuseGpioEnable()	348
4.80.1.15 HAL_GPIO.Write()	348
4.81 hal_iks32mc09x_gpio.c	348
4.82 source/hal_iks32mc09x_hall.c 文件参考	351
4.82.1 函数说明	352
4.82.1.1 HAL_HALL.Disable()	352
4.82.1.2 HAL_HALL.Enable()	352
4.82.1.3 HAL_HALL.ReadHallData()	353
4.82.1.4 HAL_HALL.ReadHallWidth()	353
4.82.1.5 HAL_HALL.ReadTh()	353
4.82.1.6 HAL_HALL.Reset()	354
4.82.1.7 HAL_HALL.SetFilter()	354

4.82.1.8 HAL_HALL_SetFilteruS()	355
4.82.1.9 HAL_HALL_SetFreq()	355
4.82.1.10 HAL_HALL_SetGpio()	356
4.82.1.11 HAL_HALL_SetTh()	356
4.82.1.12 HAL_HALL_StartCounting()	357
4.82.1.13 HAL_HALL_StopCounting()	357
4.83 hal_lks32mc09x_hall.c	357
4.84 source/hal_lks32mc09x_i2c.c 文件参考	359
4.84.1 函数说明	360
4.84.1.1 HAL_I2C_BusReset()	360
4.84.1.2 HAL_I2C_Disable()	361
4.84.1.3 HAL_I2C_Enable()	361
4.84.1.4 HAL_I2C_Reset()	361
4.84.1.5 HAL_I2C_RxData()	361
4.84.1.6 HAL_I2C_SetBaud()	362
4.84.1.7 HAL_I2C_SetGpioScl()	362
4.84.1.8 HAL_I2C_SetGpioSda()	363
4.84.1.9 HAL_I2C_TxData()	364
4.84.2 变量说明	364
4.84.2.1 hal_baud	364
4.84.2.2 hal_i2cio	365
4.85 hal_lks32mc09x_i2c.c	365
4.86 source/hal_lks32mc09x_init.c 文件参考	367
4.86.1 函数说明	368
4.86.1.1 HAL_INIT_PowerReady()	368
4.86.1.2 HAL_INIT_SysInit()	368
4.87 hal_lks32mc09x_init.c	368
4.88 source/hal_lks32mc09x_irq.c 文件参考	369
4.88.1 函数说明	369
4.88.1.1 HAL_IRQ_Disable()	369
4.88.1.2 HAL_IRQ_Enable()	370
4.88.1.3 HAL_IRQ_ModuleDisable()	370
4.88.1.4 HAL_IRQ_ModuleEnable()	371
4.88.1.5 HAL_IRQ_Reset()	371
4.89 hal_lks32mc09x_irq.c	371
4.90 source/hal_lks32mc09x_irqfun.c 文件参考	372
4.90.1 函数说明	373
4.90.1.1 HAL_ADC_IrqSamp1()	373
4.90.1.2 HAL_ADC_IrqSamp1Clear()	374
4.90.1.3 HAL_ADC_IrqSamp1Disable()	374
4.90.1.4 HAL_ADC_IrqSamp1Enable()	374
4.90.1.5 HAL_ADC_IrqSamp1Flg()	375

4.90.1.6 HAL_ADC_IrqSamp2()	375
4.90.1.7 HAL_ADC_IrqSamp2Clear()	375
4.90.1.8 HAL_ADC_IrqSamp2Disable()	376
4.90.1.9 HAL_ADC_IrqSamp2Enable()	376
4.90.1.10 HAL_ADC_IrqSamp2Flg()	376
4.90.1.11 HAL_HALL_IrqChange()	377
4.90.1.12 HAL_HALL_IrqChangeClear()	377
4.90.1.13 HAL_HALL_IrqChangeDisable()	377
4.90.1.14 HAL_HALL_IrqChangeEnable()	377
4.90.1.15 HAL_HALL_IrqChangeFlg()	379
4.90.1.16 HAL_HALL_IrqOver()	379
4.90.1.17 HAL_HALL_IrqOverClear()	379
4.90.1.18 HAL_HALL_IrqOverDisable()	380
4.90.1.19 HAL_HALL_IrqOverEnable()	380
4.90.1.20 HAL_HALL_IrqOverFlg()	380
4.91 hal_lks32mc09x_irqfun.c	381
4.92 source/hal_lks32mc09x_iwdg.c 文件参考	382
4.92.1 函数说明	383
4.92.1.1 HAL_IWDG_Disable()	383
4.92.1.2 HAL_IWDG_Enable()	384
4.92.1.3 HAL_IWDG_Feed()	384
4.92.1.4 HAL_IWDG_Reset()	385
4.92.1.5 HAL_IWDG_SetResetTimemS()	385
4.93 hal_lks32mc09x_iwdg.c	386
4.94 source/hal_lks32mc09x_mcpwm.c 文件参考	386
4.94.1 函数说明	388
4.94.1.1 HAL_MCPWM_Disable()	388
4.94.1.2 HAL_MCPWM_Enable()	389
4.94.1.3 HAL_MCPWM_GetFreq()	389
4.94.1.4 HAL_MCPWM_GetThCnt()	389
4.94.1.5 HAL_MCPWM_IrqFail()	390
4.94.1.6 HAL_MCPWM_IrqFailClear()	390
4.94.1.7 HAL_MCPWM_IrqFailDisable()	390
4.94.1.8 HAL_MCPWM_IrqFailEnable()	390
4.94.1.9 HAL_MCPWM_IrqOverflow()	391
4.94.1.10 HAL_MCPWM_IrqOverflowClear()	391
4.94.1.11 HAL_MCPWM_IrqOverflowDisable()	391
4.94.1.12 HAL_MCPWM_IrqOverflowEnable()	391
4.94.1.13 HAL_MCPWM_IrqZeroCross()	391
4.94.1.14 HAL_MCPWM_IrqZeroCrossClear()	391
4.94.1.15 HAL_MCPWM_IrqZeroCrossDisable()	391
4.94.1.16 HAL_MCPWM_IrqZeroCrossEnable()	392

4.94.1.17 HAL_MCPWM_Reset()	392
4.94.1.18 HAL_MCPWM_SetAdcTrig()	392
4.94.1.19 HAL_MCPWM_SetAdcTrigQ15()	392
4.94.1.20 HAL_MCPWM_SetDead()	393
4.94.1.21 HAL_MCPWM_SetDefaultState()	393
4.94.1.22 HAL_MCPWM_SetDrivePolarity()	394
4.94.1.23 HAL_MCPWM_SetFailSignal()	394
4.94.1.24 HAL_MCPWM_SetFreq()	395
4.94.1.25 HAL_MCPWM_SetGpioFail()	395
4.94.1.26 HAL_MCPWM_SetGpioOut()	396
4.94.1.27 HAL_MCPWM_SetOutputDisable()	396
4.94.1.28 HAL_MCPWM_SetOutputEnable()	397
4.94.1.29 HAL_MCPWM_SetOutState()	397
4.94.1.30 HAL_MCPWM_SetPwm()	398
4.94.1.31 HAL_MCPWM_SetPwmQ15()	398
4.94.1.32 HAL_MCPWM_StartCounting()	399
4.94.1.33 HAL_MCPWM_StopCounting()	399
4.94.1.34 hal_MCPWM_UpdatePolarityState()	399
4.94.2 变量说明	401
4.94.2.1 mcpwm_polarity_state	401
4.95 hal_lks32mc09x_mcpwm.c	402
4.96 source/hal_lks32mc09x_opa.c 文件参考	406
4.96.1 函数说明	408
4.96.1.1 HAL_OPA_Disable()	408
4.96.1.2 HAL_OPA_Enable()	408
4.96.1.3 HAL_OPA_Out2GpioDisable()	408
4.96.1.4 HAL_OPA_Out2GpioEnable()	408
4.96.1.5 HAL_OPA_Reset()	409
4.96.1.6 HAL_OPA_SetGain()	409
4.96.1.7 HAL_OPA_SetGpioIn()	409
4.96.1.8 HAL_OPA_SetGpioOut()	410
4.97 hal_lks32mc09x_opa.c	410
4.98 source/hal_lks32mc09x_qep.c 文件参考	412
4.98.1 函数说明	413
4.98.1.1 HAL_QEP_Disable()	413
4.98.1.2 HAL_QEP_Enable()	414
4.98.1.3 HAL_QEP_ReadCnt()	414
4.98.1.4 HAL_QEP_Reset()	414
4.98.1.5 HAL_QEP_ResetCnt()	415
4.98.1.6 HAL_QEP_SetGpio()	415
4.98.1.7 HAL_QEP_SetMode()	416
4.98.1.8 HAL_QEP_SetTh()	416

4.99 hal_lks32mc09x_qep.c	416
4.100 source/hal_lks32mc09x_reg.c 文件参考	417
4.100.1 变量说明	419
4.100.1.1 HAL_ADC	419
4.100.1.2 HAL_CMP	419
4.100.1.3 HAL_CRC	419
4.100.1.4 HAL_DMA	419
4.100.1.5 HAL_DSP	419
4.100.1.6 HAL_FLASH	419
4.100.1.7 HAL_GPIO	419
4.100.1.8 HAL_HALL	420
4.100.1.9 HAL_I2C	420
4.100.1.10 HAL_IWDG	420
4.100.1.11 HAL_MCPWM	420
4.100.1.12 HAL_QEP	420
4.100.1.13 HAL_SPI	420
4.100.1.14 HAL_SYS	420
4.100.1.15 HAL_TIMER	420
4.100.1.16 HAL_UART	421
4.101 hal_lks32mc09x_reg.c	421
4.102 source/hal_lks32mc09x_reset.c 文件参考	421
4.102.1 函数说明	422
4.102.1.1 HAL_RESET_ResetCore()	422
4.102.1.2 HAL_RESET_ResetGlobal()	422
4.102.1.3 HAL_RESET_ResetRegister()	423
4.103 hal_lks32mc09x_reset.c	423
4.104 source/hal_lks32mc09x_spi.c 文件参考	423
4.104.1 函数说明	425
4.104.1.1 HAL_SPI_Disable()	425
4.104.1.2 HAL_SPI_Enable()	425
4.104.1.3 HAL_SPI_Reset()	425
4.104.1.4 HAL_SPI_RxData()	425
4.104.1.5 HAL_SPI_RxDataWidth16()	427
4.104.1.6 HAL_SPI_RxTxData()	427
4.104.1.7 HAL_SPI_RxTxDataWidth16()	428
4.104.1.8 HAL_SPI_SetBaud()	428
4.104.1.9 HAL_SPI_SetGpioClk()	429
4.104.1.10 HAL_SPI_SetGpioCs()	429
4.104.1.11 HAL_SPI_SetGpioDi()	430
4.104.1.12 HAL_SPI_SetGpioDo()	431
4.104.1.13 HAL_SPI_SetGpioMiso()	431
4.104.1.14 HAL_SPI_SetGpioMosi()	432

4.104.1.15 HAL_SPI_SetMode()	433
4.104.1.16 HAL_SPI_TxData()	433
4.104.1.17 HAL_SPI_TxDataWidth16()	434
4.105 hal_lks32mc09x_spi.c	434
4.106 source/hal_lks32mc09x_test.c 文件参考	437
4.106.1 函数说明	438
4.106.1.1 HAL_TestRegAddr()	438
4.107 hal_lks32mc09x_test.c	438
4.108 source/hal_lks32mc09x_timer.c 文件参考	438
4.108.1 函数说明	440
4.108.1.1 HAL_TIMER_Disable()	440
4.108.1.2 HAL_TIMER_Enable()	440
4.108.1.3 HAL_TIMER_GetFreq()	440
4.108.1.4 HAL_TIMER_GetThCnt()	441
4.108.1.5 HAL_TIMER_IrqCompare0()	441
4.108.1.6 HAL_TIMER_IrqCompare0Clear()	441
4.108.1.7 HAL_TIMER_IrqCompare0Disable()	441
4.108.1.8 HAL_TIMER_IrqCompare0Enable()	441
4.108.1.9 HAL_TIMER_IrqCompare1()	442
4.108.1.10 HAL_TIMER_IrqCompare1Clear()	442
4.108.1.11 HAL_TIMER_IrqCompare1Disable()	442
4.108.1.12 HAL_TIMER_IrqCompare1Enable()	442
4.108.1.13 HAL_TIMER_IrqOverflow()	442
4.108.1.14 HAL_TIMER_IrqOverflowClear()	442
4.108.1.15 HAL_TIMER_IrqOverflowDisable()	442
4.108.1.16 HAL_TIMER_IrqOverflowEnable()	443
4.108.1.17 HAL_TIMER_Reset()	443
4.108.1.18 HAL_TIMER_SetFreq()	443
4.108.1.19 HAL_TIMER_SetGpioIn()	443
4.108.1.20 HAL_TIMER_SetGpioOut()	444
4.108.1.21 HAL_TIMER_SetPeriodCount()	445
4.108.1.22 HAL_TIMER_SetPeriodmS()	445
4.108.1.23 HAL_TIMER_SetPerioduS()	446
4.108.1.24 HAL_TIMER_SetPwm()	446
4.108.1.25 HAL_TIMER_SetPwmQ15()	447
4.108.1.26 HAL_TIMER_SetWorkMode()	447
4.108.1.27 HAL_TIMER_StartCounting()	447
4.108.1.28 HAL_TIMER_StopCounting()	447
4.109 hal_lks32mc09x_timer.c	448
4.110 source/hal_lks32mc09x_tmp.c 文件参考	452
4.110.1 函数说明	453
4.110.1.1 HAL_TMP_Adc2Temperature()	453

4.110.1.2 HAL_TMP_Adc2Temperature01()	454
4.110.1.3 HAL_TMP_Disable()	454
4.110.1.4 HAL_TMP_Enable()	455
4.110.1.5 HAL_TMP_GetTemperature()	455
4.110.1.6 HAL_TMP_GetTemperature01()	456
4.110.1.7 HAL_TMP_Reset()	457
4.110.2 变量说明	457
4.110.2.1 tmp_coef_a	457
4.110.2.2 tmp_coef_b	457
4.111 hal_lks32mc09x_tmp.c	457
4.112 source/hal_lks32mc09x_trim.c 文件参考	458
4.112.1 函数说明	459
4.112.1.1 HAL_TRIM_Read()	459
4.112.1.2 HAL_TRIM_ReadId()	460
4.112.1.3 HAL_TRIM_ReadIdCrc32()	461
4.112.1.4 HAL_TRIM_ReadMcu09()	461
4.113 hal_lks32mc09x_trim.c	462
4.114 source/hal_lks32mc09x_uart.c 文件参考	463
4.114.1 函数说明	464
4.114.1.1 HAL_UART_Disable()	464
4.114.1.2 HAL_UART_Enable()	465
4.114.1.3 HAL_UART_IrqCheckErr()	465
4.114.1.4 HAL_UART_IrqCheckErrClear()	465
4.114.1.5 HAL_UART_IrqCheckErrDisable()	465
4.114.1.6 HAL_UART_IrqCheckErrEnable()	465
4.114.1.7 HAL_UART_IrqReceiveOver()	465
4.114.1.8 HAL_UART_IrqReceiveOverClear()	465
4.114.1.9 HAL_UART_IrqReceiveOverDisable()	466
4.114.1.10 HAL_UART_IrqReceiveOverEnable()	466
4.114.1.11 HAL_UART_IrqSendOver()	466
4.114.1.12 HAL_UART_IrqSendOverClear()	466
4.114.1.13 HAL_UART_IrqSendOverDisable()	466
4.114.1.14 HAL_UART_IrqSendOverEnable()	466
4.114.1.15 HAL_UART_IrqStopErr()	466
4.114.1.16 HAL_UART_IrqStopErrClear()	467
4.114.1.17 HAL_UART_IrqStopErrDisable()	467
4.114.1.18 HAL_UART_IrqStopErrEnable()	467
4.114.1.19 HAL_UART_ReadBuffer()	467
4.114.1.20 HAL_UART_Reset()	467
4.114.1.21 HAL_UART_SetBaud()	467
4.114.1.22 HAL_UART_SetGpioRx()	468
4.114.1.23 HAL_UART_SetGpioTx()	468

4.114.1.24 HAL_UART_SetMode()	469
4.114.1.25 HAL_UART_TxData()	469
4.114.1.26 HAL_UART_WriteBuffer()	469
4.115 hal_lks32mc09x_uart.c	470
4.116 source/hal_lks32mc09x_wake.c 文件参考	474
4.116.1 函数说明	475
4.116.1.1 HAL_WAKE_DeepSleeploTimeWakeUp()	475
4.116.1.2 HAL_WAKE_DeepSleeploWake()	476
4.116.1.3 HAL_WAKE_DeepSleepTimeWakeUp()	476
4.117 hal_lks32mc09x_wake.c	477

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 (ADC0)	8
HAL_AON_TypeDef	
AON寄存器结构体句柄	13
HAL_CLU_TypeDef	
CL0寄存器	15
HAL_CMP_TypeDef	
比较器寄存器 (CMP)	18
HAL_CRC_TypeDef	
CRC0寄存器 (CRC0)	20
HAL_DMA_TypeDef	
DMA寄存器	21
HAL_DMAMODULE_TypeDef	
DMA控制状态寄存器对象定义	23
HAL_DSP_TypeDef	
DSP0寄存器结构体 (DSP0)	24
HAL_EXTI_TypeDef	
EXTI寄存器 (EXTI)	26
HAL_FLASH_TypeDef	
FLASH寄存器 (FLASH)	28
HAL_GPIO_TypeDef	
GPIO寄存器	33
HAL_HALL_TypeDef	
HALL0寄存器 (HALL0)	38
HAL_I2C_TypeDef	
I2C0寄存器 (I2C0)	40
HAL_IWDG_TypeDef	
IWDG寄存器结构体句柄	42
HAL_MCPWM_TypeDef	
MCPWM0寄存器 (MCPWM0)	44
HAL_QEP_TypeDef	
QEP0寄存器 (QEP0)	53

HAL_SIF_TypeDef	
SIF0寄存器 (SIF0)	55
HAL_SPI_TypeDef	
SPI0寄存器 (SPI0)	56
HAL_SYS_TypeDef	
SYS寄存器 (SYS)	58
HAL_TIMER_TypeDef	
TIMER0寄存器 (TIMER0)	67
HAL_UART_TypeDef	
UART寄存器	71

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_lks32mc09x.c	75
hal_lks32mc09x.h	76
hal_lksmcu.h	88
include/hal_lks32mc09x_reg.h	89
include/hal_lksmcu_adc.h	98
include/hal_lksmcu_can.h	115
include/hal_lksmcu_clk.h	115
include/hal_lksmcu_cmp.h	120
include/hal_lksmcu_crc.h	128
include/hal_lksmcu_dac.h	132
include/hal_lksmcu_dma.h	136
include/hal_lksmcu_dsp.h	140
include/hal_lksmcu_flash.h	144
include/hal_lksmcu_gpio.h	153
include/hal_lksmcu_hall.h	168
include/hal_lksmcu_i2c.h	176
include/hal_lksmcu_init.h	182
include/hal_lksmcu_irq.h	183
include/hal_lksmcu_irqfun.h	188
include/hal_lksmcu_iwdg.h	196
include/hal_lksmcu_mcpwm.h	200
include/hal_lksmcu_opa.h	217
include/hal_lksmcu_qep.h	224
include/hal_lksmcu_reg.h	
HAL库配置和内部参数	230
include/hal_lksmcu_reset.h	233
include/hal_lksmcu_spi.h	234
include/hal_lksmcu_timer.h	246
include/hal_lksmcu_tmp.h	257
include/hal_lksmcu_trim.h	263
include/hal_lksmcu_uart.h	266
include/hal_lksmcu_wake.h	275
source/hal_lks32mc09x_adc.c	
虚拟实现,用来在芯片之外约束函数运行效果	278
source/hal_lks32mc09x_clk.c	294

source/hal_lks32mc09x_cmp.c	299
source/hal_lks32mc09x_crc.c	306
source/hal_lks32mc09x_dac.c	312
source/hal_lks32mc09x_dma.c	318
source/hal_lks32mc09x_dsp.c	323
source/hal_lks32mc09x_flash.c	327
source/hal_lks32mc09x_gpio.c	341
source/hal_lks32mc09x_hall.c	351
source/hal_lks32mc09x_i2c.c	359
source/hal_lks32mc09x_init.c	367
source/hal_lks32mc09x_irq.c	369
source/hal_lks32mc09x_irqfun.c	372
source/hal_lks32mc09x_iwdg.c	382
source/hal_lks32mc09x_mcpwm.c	386
source/hal_lks32mc09x_opa.c	406
source/hal_lks32mc09x_qep.c	412
source/hal_lks32mc09x_reg.c	417
source/hal_lks32mc09x_reset.c	421
source/hal_lks32mc09x_spi.c	423
source/hal_lks32mc09x_test.c	437
source/hal_lks32mc09x_timer.c	438
source/hal_lks32mc09x_tmp.c	452
source/hal_lks32mc09x_trim.c	458
source/hal_lks32mc09x_uart.c	463
source/hal_lks32mc09x_wake.c	474

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
- volatile uint32_t **DATA**
0x14 比较器输出数值寄存器

3.1.1 详细描述

ADC寄存器结构体句柄

在文件 [hal_lks32mc09x_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_lks32mc09x_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_lks32mc09x_cmp.c](#) 第 15 行定义.

3.1.2.3 DATA

```
volatile uint32_t DATA
```

0x14 比较器输出数值寄存器

在文件 [hal_lks32mc09x_cmp.c](#) 第 17 行定义.

3.1.2.4 IE

```
volatile uint32_t IE
```

0x00 比较器中断使能寄存器

在文件 [hal_lks32mc09x_cmp.c](#) 第 12 行定义.

3.1.2.5 IF

```
volatile uint32_t IF
```

0x04 比较器中断标志寄存器

在文件 [hal_lks32mc09x_cmp.c](#) 第 13 行定义.

3.1.2.6 TCLK

```
volatile uint32_t TCLK
```

0x08 比较器分频时钟控制寄存器

在文件 [hal_lks32mc09x_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_lks32mc09x_crc.c](#) 第 15 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_lks32mc09x_crc.c](#) 第 18 行定义.

3.2.2.2 polynomial

```
uint32_t polynomial
```

在文件 [hal_lks32mc09x_crc.c](#) 第 17 行定义.

3.2.2.3 reflectIn

```
uint8_t reflectIn
```

在文件 [hal_lks32mc09x_crc.c](#) 第 20 行定义.

3.2.2.4 reflectOut

```
uint8_t reflectOut
```

在文件 [hal_lks32mc09x_crc.c](#) 第 21 行定义.

3.2.2.5 width

```
CRC_Width width
```

在文件 [hal_lks32mc09x_crc.c](#) 第 22 行定义.

3.2.2.6 xorOut

```
uint32_t xorOut
```

在文件 [hal_lks32mc09x_crc.c](#) 第 19 行定义.

3.3 HAL_ADC_TypeDef结构体 参考

ADC0寄存器 (ADC0)

```
#include <hal_lks32mc09x_reg.h>
```

成员变量

- `__IOM uint32_t DAT [12]`
 < ADC0结构体
- `__IM uint32_t RESERVED [8]`
- `__IOM uint32_t CHN0`
 数据0/1/2/3样本信号源 [15:12] ADC0第3次采样正端输入信号选择 [11:8] ADC0第2次采样正端输入信号选择 [7:4] ADC0第1次采样正端输入信号选择 [3:0] ADC0第0次采样正端输入信号选择
- `__IOM uint32_t CHN1`
 数据4/5/6/7样本信号源 [15:12] ADC0第7次采样正端输入信号选择或第4次空闲采样正端输入信号选择 [11:8] ADC0第6次采样正端输入信号选择或第5次空闲采样正端输入信号选择 [7:4] ADC0第5次采样正端输入信号选择 [3:0] ADC0第4次采样正端输入信号选择
- `__IOM uint32_t CHN2`
 数据8/9/10/11样本信号源 [15:12] ADC0第11次常规采样正端输入信号选择或第0次空闲采样正端输入信号选择 [11:8] ADC0第10次采样正端输入信号选择或第1次空闲采样正端输入信号选择 [7:4] ADC0第9次采样正端输入信号选择或第2次空闲采样正端输入信号选择 [3:0] ADC0第8次采样正端输入信号选择或第3次空闲采样正端输入信号选择
- `__IM uint32_t RESERVED1`
- `__IOM uint32_t CHNT`
 第0/1轮/空闲通道编号 [11:8] 空闲采样次数 [7:4] 第二段常规采样次数 [3:0] 第一段常规采样次数
- `__IOM uint32_t GAIN`
 样本增益选择 [11] DAT11增益选择,0:±3.6V量程,1:±5V量程 [10] DAT10增益选择,0:±3.6V量程,1:±5V量程 [9] DAT9增益选择,0:±3.6V量程,1:±5V量程 [8] DAT8增益选择,0:±3.6V量程,1:±5V量程 [7] DAT7增益选择,0:±3.6V量程,1:±5V量程 [6] DAT6增益选择,0:±3.6V量程,1:±5V量程 [5] DAT5增益选择,0:±3.6V量程,1:±5V量程 [4] DAT4增益选择,0:±3.6V量程,1:±5V量程 [3] DAT3增益选择,0:±3.6V量程,1:±5V量程 [2] DAT2增益选择,0:±3.6V量程,1:±5V量程 [1] DAT1增益选择,0:±3.6V量程,1:±5V量程 [0] DAT0增益选择,0:±3.6V量程,1:±5V量程
- `__IM uint32_t RESERVED2 [3]`
- `__IOM uint32_t CFG`
 配置 [12] 0:单段采样,1:两段采样 [11] 状态机复位控制信号 [10] ADC.DAT对齐方式 [9] 空闲触发优先级 [8] 连续采样模式 [7:4] 触发一次采样所需的事件数 [3] 过采样触发模式 [2:0] 过采样率
- `__IOM uint32_t TRIG`
 样本触发配置 [27] CL_OUTPUT[3]触发ADC空闲采样使能 [26] CL_OUTPUT[2]触发ADC空闲采样使能 [25] CL_OUTPUT[1]触发ADC空闲采样使能 [24] CL_OUTPUT[0]触发ADC空闲采样使能 [23] TIMER2比较/捕获事件1触发ADC空闲采样使能 [22] TIMER2比较/捕获事件0触发ADC空闲采样使能 [21] TIMER1比较/捕获事件1触发ADC空闲采样使能 [20] TIMER0比较/捕获事件0触发ADC空闲采样使能 [19] MCPWM0 T3事件触发ADC空闲采样使能 [18] MCPWM0 T2事件触发ADC空闲采样使能 [17] MCPWM0 T1事件触发ADC空闲采样使能 [16] MCPWM0 T0事件触发ADC空闲采样使能 [11] CL_OUTPUT[3]触发ADC常规采样使能 [10] CL_OUTPUT[2]触发ADC常规采样使能 [9] CL_OUTPUT[1]触发ADC常规采样使能 [8] CL_OUTPUT[0]触发ADC常规采样使能 [7] TIMER2比较/捕获事件1触发ADC常规采样使能 [6] TIMER2比较/捕获事件0触发ADC常规采样使能 [5] TIMER1比较/捕获事件1触发ADC常规采样使能 [4] TIMER0比较/捕获事件0触发ADC常规采样使能 [3] MCPWM0 T3事件触发ADC常规采样使能 [2] MCPWM0 T2事件触发ADC常规采样使能 [1] MCPWM0 T1事件触发ADC常规采样使能 [0] MCPWM0 T0事件触发ADC常规采样使能
- `__IOM uint32_t SWT`
 软件触发 [15:0] 写入数据为0x5AA5时,产生一次常规触发; 写入数据位0xF00F时,产生一次空闲触发
- `__IOM uint32_t DC0`
 低增益时S/H A的直流偏移 [15:0] ADC采样电路DC offset
- `__IOM uint32_t AMC0`
 低增益时S/H A的放大校准 [9:0] ADC采样电路增益校正系数

- `__IOM uint32_t DC1`
高增益时S/H A的直流偏移 [15:0] ADC采样电路DC offset
- `__IOM uint32_t AMC1`
高增益时S/H A的放大校准 [9:0] ADC采样电路增益校正系数
- `__IOM uint32_t IE`
中断使能 [15] 空闲采样完成DMA请求使能 [14] 硬件触发发生在非空闲状态DMA请求使能 [13] 软件触发发生在非空闲状态DMA请求使能 [10] 阈值监测0超限DMA请求使能 [9] 第二段采样完成DMA请求使能 [8] 第一段采样完成DMA请求使能 [7] 空闲采样完成中断使能 [6] 硬件触发发生在非空闲状态中断使能 [5] 软件触发发生在非空闲状态中断使能 [2] 阈值监测0超限中断使能 [1] 第二段常规采样完成中断使能 [0] 第一段常规采样完成中断使能
- `__IOM uint32_t IF`
中断标志 [7] 空闲采样完成中断标志 [6] 硬件触发发生在非空闲状态中断标志 [5] 软件触发发生在非空闲状态中断标志 [2] 阈值监测0超限中断标志 [1] 第二段常规采样完成中断标志 [0] 第一段常规采样完成中断标志
- `__IOM uint32_t RESERVED3` [11]
- `__IOM uint32_t LTH`
数据低阈值 [15:12] 符号扩展 [11:0] ADC模拟看门狗0下阈值
- `__IOM uint32_t HTH`
数据高阈值 [15:12] 符号扩展 [11:0] ADC模拟看门狗0上阈值
- `__IOM uint32_t GEN`
数据阈值保护使能 [15:0] ADC模拟看门狗0对应使能位 BIT0: DAT0看门狗监测使能 BIT1: DAT1看门狗监测使能 ... BIT7: DAT13看门狗监测使能

3.3.1 详细描述

ADC0寄存器 (ADC0)

在文件 [hal_lks32mc09x_reg.h](#) 第 12 行定义.

3.3.2 结构体成员变量说明

3.3.2.1 AMC0

```
__IOM uint32_t AMC0
```

低增益时S/H A的放大校准 [9:0] ADC采样电路增益校正系数

在文件 [hal_lks32mc09x_reg.h](#) 第 27 行定义.

3.3.2.2 AMC1

```
__IOM uint32_t AMC1
```

高增益时S/H A的放大校准 [9:0] ADC采样电路增益校正系数

在文件 [hal_lks32mc09x_reg.h](#) 第 29 行定义.

3.3.2.3 CFG

```
--IOM uint32_t CFG
```

配置 [12:0]:单段采样,1:两段采样 [11] 状态机复位控制信号 [10] ADC.DAT对齐方式 [9] 空闲触发优先级 [8] 连续采样模式 [7:4] 触发一次采样所需的事件数 [3] 过采样触发模式 [2:0] 过采样率

在文件 [hal_lks32mc09x_reg.h](#) 第 23 行定义.

3.3.2.4 CHN0

```
--IOM uint32_t CHN0
```

数据0/1/2/3样本信号源 [15:12] ADC0第3次采样正端输入信号选择 [11:8] ADC0第2次采样正端输入信号选择 [7:4] ADC0第1次采样正端输入信号选择 [3:0] ADC0第0次采样正端输入信号选择

在文件 [hal_lks32mc09x_reg.h](#) 第 16 行定义.

3.3.2.5 CHN1

```
--IOM uint32_t CHN1
```

数据4/5/6/7样本信号源 [15:12] ADC0第7次采样正端输入信号选择或第4次空闲采样正端输入信号选择 [11:8] ADC0第6次采样正端输入信号选择或第5次空闲采样正端输入信号选择 [7:4] ADC0第5次采样正端输入信号选择 [3:0] ADC0第4次采样正端输入信号选择

在文件 [hal_lks32mc09x_reg.h](#) 第 17 行定义.

3.3.2.6 CHN2

```
--IOM uint32_t CHN2
```

数据8/9/10/11样本信号源 [15:12] ADC0第11次常规采样正端输入信号选择或第0次空闲采样正端输入信号选择 [11:8] ADC0第10次采样正端输入信号选择或第1次空闲采样正端输入信号选择 [7:4] ADC0第9次采样正端输入信号选择或第2次空闲采样正端输入信号选择 [3:0] ADC0第8次采样正端输入信号选择或第3次空闲采样正端输入信号选择

在文件 [hal_lks32mc09x_reg.h](#) 第 18 行定义.

3.3.2.7 CHNT

```
--IOM uint32_t CHNT
```

第0/1轮/空闲通道编号 [11:8] 空闲采样次数 [7:4] 第二段常规采样次数 [3:0] 第一段常规采样次数

在文件 [hal_lks32mc09x_reg.h](#) 第 20 行定义.

3.3.2.8 DAT

```
--IOM uint32_t DAT[12]
```

< ADC0结构体

第0个样本数据 [15:0] ADC0第0次采样数据

在文件 [hal_lks32mc09x_reg.h](#) 第 14 行定义.

3.3.2.9 DC0

```
--IOM uint32_t DC0
```

低增益时S/H A的直流偏移 [15:0] ADC采样电路DC offset

在文件 [hal_lks32mc09x_reg.h](#) 第 26 行定义.

3.3.2.10 DC1

```
--IOM uint32_t DC1
```

高增益时S/H A的直流偏移 [15:0] ADC采样电路DC offset

在文件 [hal_lks32mc09x_reg.h](#) 第 28 行定义.

3.3.2.11 GAIN

```
--IOM uint32_t GAIN
```

样本增益选择 [11] DAT11增益选择,0:±3.6V量程,1:±5V量程 [10] DAT10增益选择,0:±3.6V量程,1:±5V量程 [9] DAT9增益选择,0:±3.6V量程,1:±5V量程 [8] DAT8增益选择,0:±3.6V量程,1:±5V量程 [7] DAT7增益选择,0:±3.6V量程,1:±5V量程 [6] DAT6增益选择,0:±3.6V量程,1:±5V量程 [5] DAT5增益选择,0:±3.6V量程,1:±5V量程 [4] DAT4增益选择,0:±3.6V量程,1:±5V量程 [3] DAT3增益选择,0:±3.6V量程,1:±5V量程 [2] DAT2增益选择,0:±3.6V量程,1:±5V量程 [1] DAT1增益选择,0:±3.6V量程,1:±5V量程 [0] DAT0增益选择,0:±3.6V量程,1:±5V量程

在文件 [hal_lks32mc09x_reg.h](#) 第 21 行定义.

3.3.2.12 GEN

```
--IOM uint32_t GEN
```

数据阈值保护使能 [15:0] ADC模拟看门狗0对应使能位 BIT0: DAT0看门狗监测使能 BIT1: DAT1看门狗监测使能 ... BIT7: DAT13看门狗监测使能

在文件 [hal_lks32mc09x_reg.h](#) 第 35 行定义.

3.3.2.13 HTH

```
--IOM uint32_t HTH
```

数据高阈值 [15:12] 符号扩展 [11:0] ADC模拟看门狗0上阈值

在文件 [hal_lks32mc09x_reg.h](#) 第 34 行定义.

3.3.2.14 IE

```
--IOM uint32_t IE
```

中断使能 [15] 空闲采样完成DMA请求使能 [14] 硬件触发发生在非空闲状态DMA请求使能 [13] 软件触发发生在非空闲状态DMA请求使能 [10] 阈值监测0超限DMA请求使能 [9] 第二段采样完成DMA请求使能 [8] 第一段采样完成DMA请求使能 [7] 空闲采样完成中断使能 [6] 硬件触发发生在非空闲状态中断使能 [5] 软件触发发生在非空闲状态中断使能 [2] 阈值监测0超限中断使能 [1] 第二段常规采样完成中断使能 [0] 第一段常规采样完成中断使能

在文件 [hal_lks32mc09x_reg.h](#) 第 30 行定义.

3.3.2.15 IF

```
--IOM uint32_t IF
```

中断标志 [7] 空闲采样完成中断标志 [6] 硬件触发发生在非空闲状态中断标志 [5] 软件触发发生在非空闲状态中断标志 [2] 阈值监测0超限中断标志 [1] 第二段常规采样完成中断标志 [0] 第一段常规采样完成中断标志

在文件 [hal_lks32mc09x_reg.h](#) 第 31 行定义.

3.3.2.16 LTH

```
--IOM uint32_t LTH
```

数据低阈值 [15:12] 符号扩展 [11:0] ADC模拟看门狗0下阈值

在文件 [hal_lks32mc09x_reg.h](#) 第 33 行定义.

3.3.2.17 RESERVED

```
--IM uint32_t RESERVED[8]
```

在文件 [hal_lks32mc09x_reg.h](#) 第 15 行定义.

3.3.2.18 RESERVED1

```
--IM uint32_t RESERVED1
```

在文件 [hal_lks32mc09x_reg.h](#) 第 19 行定义.

3.3.2.19 RESERVED2

```
--IM uint32_t RESERVED2[3]
```

在文件 [hal_lks32mc09x_reg.h](#) 第 22 行定义.

3.3.2.20 RESERVED3

```
--IM uint32_t RESERVED3[11]
```

在文件 [hal_lks32mc09x_reg.h](#) 第 32 行定义.

3.3.2.21 SWT

```
--OM uint32_t SWT
```

软件触发 [15:0] 写入数据为0x5AA5时,产生一次常规触发; 写入数据位0xF00F时,产生一次空闲触发

在文件 [hal_lks32mc09x_reg.h](#) 第 25 行定义.

3.3.2.22 TRIG

```
--IOM uint32_t TRIG
```

样本触发配置 [27] CL_OUTPUT[3]触发ADC空闲采样使能 [26] CL_OUTPUT[2]触发ADC空闲采样使能 [25] CL_OUTPUT[1]触发ADC空闲采样使能 [24] CL_OUTPUT[0]触发ADC空闲采样使能 [23] TIMER2比较/捕获事件1触发ADC空闲采样使能 [22] TIMER2比较/捕获事件0触发ADC空闲采样使能 [21] TIMER1比较/捕获事件1触发ADC空闲采样使能 [20] TIMER0比较/捕获事件0触发ADC空闲采样使能 [19] MCPWM0 T3事件触发ADC空闲采样使能 [18] MCPWM0 T2事件触发ADC空闲采样使能 [17] MCPWM0 T1事件触发ADC空闲采样使能 [16] MCPWM0 T0事件触发ADC空闲采样使能 [11] CL_OUTPUT[3]触发ADC常规采样使能 [10] CL_OUTPUT[2]触发ADC常规采样使能 [9] CL_OUTPUT[1]触发ADC常规采样使能 [8] CL_OUTPUT[0]触发ADC常规采样使能 [7] TIMER2比较/捕获事件1触发ADC常规采样使能 [6] TIMER2比较/捕获事件0触发ADC常规采样使能 [5] TIMER1比较/捕获事件1触发ADC常规采样使能 [4] TIMER0比较/捕获事件0触发ADC常规采样使能 [3] MCPWM0 T3事件触发ADC常规采样使能 [2] MCPWM0 T2事件触发ADC常规采样使能 [1] MCPWM0 T1事件触发ADC常规采样使能 [0] MCPWM0 T0事件触发ADC常规采样使能

在文件 [hal_lks32mc09x_reg.h](#) 第 24 行定义.

3.4 HAL_AON_TypeDef结构体 参考

AON寄存器结构体句柄

```
#include <hal_lks32mc09x_reg.h>
```

成员变量

- `__IOM uint32_t PWR_CFG`
功耗管理配置寄存器 [1] *IOWK.FLT* IO 唤醒信号滤波使能,默认使能
- `__IOM uint32_t EVT.RCD`
事件记录寄存器 [13] 深度休眠记录,高表示发生过 [12] 休眠记录,高表示发生过 [9] *IWDG* 定时唤醒记录,高表示 *Deep Sleep* 休眠被 *IWDG* 定时唤醒 [8] *IO* 唤醒或 *CLU* 唤醒记录,高表示 *Deep Sleep* 休眠被 *AON.IO.WAKE* 唤醒 [3] 独立看门狗复位发生记录,高表示发生过 [2] 按键复位发生记录,高表示发生过 [0] *POR* 复位发生记录,高表示发生过
- `__IOM uint32_t IO_WAKE_POL`
IO 唤醒极性寄存器 [7] *P2[0]* 外部唤醒极性 [6] *P2[4]* 外部唤醒极性 [5] *P2[7]* 外部唤醒极性 [4] *P0[14]* 外部唤醒极性 [3] *P0[13]* 外部唤醒极性 [2] *P0[12]* 外部唤醒极性 [1] *P0[2]* 外部唤醒极性 [0] *P0[0]* 外部唤醒极性
- `__IOM uint32_t IO_WAKE_EN`
IO 唤醒使能寄存器 [11] *CLUOUT3* 作为唤醒使能 [10] *CLUOUT2* 作为唤醒使能 [9] *CLUOUT1* 作为唤醒使能 [8] *CLUOUT0* 作为唤醒使能 [7] *P2[0]* 外部唤醒使能 [6] *P2[4]* 外部唤醒使能 [5] *P2[7]* 外部唤醒使能 [4] *P0[14]* 外部唤醒使能 [3] *P0[13]* 外部唤醒使能 [2] *P0[12]* 外部唤醒使能 [1] *P0[2]* 外部唤醒使能 [0] *P0[0]* 外部唤醒使能
- `__IOM uint32_t BKP_REG0`
后备寄存器 0
- `__IOM uint32_t BKP_REG1`
后备寄存器 1

3.4.1 详细描述

AON寄存器结构体句柄

在文件 `hal_lks32mc09x_reg.h` 第 366 行定义.

3.4.2 结构体成员变量说明

3.4.2.1 BKP_REG0

```
__IOM uint32_t BKP_REG0
```

后备寄存器 0

在文件 `hal_lks32mc09x_reg.h` 第 372 行定义.

3.4.2.2 BKP_REG1

```
__IOM uint32_t BKP_REG1
```

后备寄存器 1

在文件 `hal_lks32mc09x_reg.h` 第 373 行定义.

3.4.2.3 EVT_RCD

```
--IOM uint32_t EVT_RCD
```

事件记录寄存器 [13] 深度休眠记录,高表示发生过 [12] 休眠记录,高表示发生过 [9] IWDG 定时唤醒记录,高表示 Deep Sleep 休眠被 IWDG 定时唤醒 [8] IO唤醒或CLU 唤醒记录,高表示 Deep Sleep 休眠被 AON_IO_WAKE_EN 中对应的使能信号唤醒 [3] 独立看门狗复位发生记录,高表示发生过 [2] 按键复位发生记录,高表示发生过 [0] POR 复位发生记录,高表示发生过

在文件 [hal_lks32mc09x_reg.h](#) 第 369 行定义.

3.4.2.4 IO_WAKE_EN

```
--IOM uint32_t IO_WAKE_EN
```

IO 唤醒使能寄存器 [11] CLUOUT3 作为唤醒使能 [10] CLUOUT2 作为唤醒使能 [9] CLUOUT1 作为唤醒使能 [8] CLUOUT0 作为唤醒使能 [7] P2[0]外部唤醒使能 [6] P2[4]外部唤醒使能 [5] P2[7]外部唤醒使能 [4] P0[14]外部唤醒使能 [3] P0[13]外部唤醒使能 [2] P0[12]外部唤醒使能 [1] P0[2]外部唤醒使能 [0] P0[0]外部唤醒使能

在文件 [hal_lks32mc09x_reg.h](#) 第 371 行定义.

3.4.2.5 IO_WAKE_POL

```
--IOM uint32_t IO_WAKE_POL
```

IO 唤醒极性寄存器 [7] P2[0]外部唤醒极性 [6] P2[4]外部唤醒极性 [5] P2[7]外部唤醒极性 [4] P0[14]外部唤醒极性 [3] P0[13]外部唤醒极性 [2] P0[12]外部唤醒极性 [1] P0[2]外部唤醒极性 [0] P0[0]外部唤醒极性

在文件 [hal_lks32mc09x_reg.h](#) 第 370 行定义.

3.4.2.6 PWR_CFG

```
--IOM uint32_t PWR_CFG
```

功耗管理配置寄存器 [1] IOWK_FLT IO 唤醒信号滤波使能,默认使能

在文件 [hal_lks32mc09x_reg.h](#) 第 368 行定义.

3.5 HAL_CLU_TypeDef结构体 参考

CL0寄存器

```
#include <hal_lks32mc09x_reg.h>
```

成员变量

- `__IOM uint32_t EN`
 < CL0结构体
- `__IOM uint32_t IE`
 CL0中断使能寄存器 [7] CLU3上升沿触发中断使能 [6] CLU3下降沿触发中断使能 [5] CLU2上升沿触发中断使能 [4] CLU2下降沿触发中断使能 [3] CLU1上升沿触发中断使能 [2] CLU1下降沿触发中断使能 [1] CLU0上升沿触发中断使能 [0] CLU0下降沿触发中断使能
- `__IOM uint32_t IF`
 CL0中断标志寄存器 [7] CLU3上升沿中断标志位 [6] CLU3下降沿中断标志位 [5] CLU2上升沿中断标志位 [4] CLU2下降沿中断标志位 [3] CLU1上升沿中断标志位 [2] CLU1下降沿中断标志位 [1] CLU0上升沿中断标志位 [0] CLU0下降沿中断标志位
- `__IOM uint32_t OUT`
 CL0输出寄存器 [3] CLU3输出状态 [2] CLU2输出状态 [1] CLU1输出状态 [0] CLU0输出状态
- `__IOM uint32_t MX`
 CL0多路选择寄存器 [31:28] CLU3 A输入多路复用选择 [27:24] CLU3 B输入多路复用选择 [23:20] CLU2 A输入多路复用选择 [19:16] CLU2 B输入多路复用选择 [15:12] CLU1 A输入多路复用选择 [11:8] CLU1 B输入多路复用选择 [7:4] CLU0 A输入多路复用选择 [3:0] CLU0 B输入多路复用选择
- `__IOM uint32_t FN`
 CL0功能选择寄存器 [31:24] CLU3查找表功能选择 [23:16] CLU2查找表功能选择 [15:8] CLU1查找表功能选择 [7:0] CLU0查找表功能选择
- `__IOM uint32_t CF`
 CL0配置寄存器 [31] CLU3输出选取 [30] CLU3输出使能 [27] CLU3 D触发器复位 [26] CLU3 D触发器时钟电平反向 [25:24] CLU3 D触发器时钟选择 [23] CLU2输出选取 [22] CLU2输出使能 [19] CLU2 D触发器复位 [18] CLU2 D触发器时钟电平反向 [17:16] CLU2 D触发器时钟选择 [15] CLU1输出选取 [14] CLU1输出使能 [11] CLU1 D触发器复位 [10] CLU1 D触发器时钟电平反向 [9:8] CLU1 D触发器时钟选择 [7] CLU0输出选取 [6] CLU0输出使能 [3] CLU0 D触发器复位 [2] CLU0 D触发器时钟电平反向 [1:0] CLU0 D触发器时钟选择

3.5.1 详细描述

CL0寄存器

在文件 [hal_iks32mc09x_reg.h](#) 第 52 行定义。

3.5.2 结构体成员变量说明

3.5.2.1 CF

```
__IOM uint32_t CF
```

CL0配置寄存器 [31] CLU3输出选取 [30] CLU3输出使能 [27] CLU3 D触发器复位 [26] CLU3 D触发器时钟电平反向 [25:24] CLU3 D触发器时钟选择 [23] CLU2输出选取 [22] CLU2输出使能 [19] CLU2 D触发器复位 [18] CLU2 D触发器时钟电平反向 [17:16] CLU2 D触发器时钟选择 [15] CLU1输出选取 [14] CLU1输出使能 [11] CLU1 D触发器复位 [10] CLU1 D触发器时钟电平反向 [9:8] CLU1 D触发器时钟选择 [7] CLU0输出选取 [6] CLU0输出使能 [3] CLU0 D触发器复位 [2] CLU0 D触发器时钟电平反向 [1:0] CLU0 D触发器时钟选择

在文件 [hal_iks32mc09x_reg.h](#) 第 60 行定义。

3.5.2.2 EN

__IOM uint32_t EN

< CL0结构体

CL0使能寄存器 [3] CLU3使能 [2] CLU2使能 [1] CLU1使能 [0] CLU0使能

在文件 [hal_lks32mc09x_reg.h](#) 第 54 行定义.

3.5.2.3 FN

__IOM uint32_t FN

CL0功能选择寄存器 [31:24] CLU3查找表功能选择 [23:16] CLU2查找表功能选择 [15:8] CLU1查找表功能选择 [7:0] CLU0查找表功能选择

在文件 [hal_lks32mc09x_reg.h](#) 第 59 行定义.

3.5.2.4 IE

__IOM uint32_t IE

CL0中断使能寄存器 [7] CLU3上升沿触发中断使能 [6] CLU3下降沿触发中断使能 [5] CLU2上升沿触发中断使能 [4] CLU2下降沿触发中断使能 [3] CLU1上升沿触发中断使能 [2] CLU1下降沿触发中断使能 [1] CLU0上升沿触发中断使能 [0] CLU0下降沿触发中断使能

在文件 [hal_lks32mc09x_reg.h](#) 第 55 行定义.

3.5.2.5 IF

__IOM uint32_t IF

CL0中断标志寄存器 [7] CLU3上升沿中断标志位 [6] CLU3下降沿中断标志位 [5] CLU2上升沿中断标志位 [4] CLU2下降沿中断标志位 [3] CLU1上升沿中断标志位 [2] CLU1下降沿中断标志位 [1] CLU0上升沿中断标志位 [0] CLU0下降沿中断标志位

在文件 [hal_lks32mc09x_reg.h](#) 第 56 行定义.

3.5.2.6 MX

__IOM uint32_t MX

CL0多路选择寄存器 [31:28] CLU3 A输入多路复用选择 [27:24] CLU3 B输入多路复用选择 [23:20] CLU2 A输入多路复用选择 [19:16] CLU2 B输入多路复用选择 [15:12] CLU1 A输入多路复用选择 [11:8] CLU1 B输入多路复用选择 [7:4] CLU0 A输入多路复用选择 [3:0] CLU0 B输入多路复用选择

在文件 [hal_lks32mc09x_reg.h](#) 第 58 行定义.

3.5.2.7 OUT

```
--IOM uint32_t OUT
```

CL0输出寄存器 [3] CLU3输出状态 [2] CLU2输出状态 [1] CLU1输出状态 [0] CLU0输出状态

在文件 [hal_lks32mc09x_reg.h](#) 第 57 行定义。

3.6 HAL_CMP_TypeDef结构体 参考

比较器寄存器 (CMP)

```
#include <hal_lks32mc09x_reg.h>
```

成员变量

- [__IOM uint32_t IE](#)
 < CMP 结构体
- [__IOM uint32_t IF](#)
 比较器中断标志 [1] 比较器1中断标志 [0] 比较器0中断标志
- [__IOM uint32_t TCLK](#)
 比较器时钟控制 [26:24] 比较器0/1共用滤波时钟分频系数 [15:12] 比较器1滤波时钟分频 [11] 比较器1滤波时钟使能 [10:8] 比较器1滤波时钟分频 [7:4] 比较器0滤波时钟分频 [3] 比较器0滤波时钟使能 [2:0] 比较器0滤波时钟分频
- [__IOM uint32_t CFG](#)
 比较器配置 [7] 比较器1开窗PWM信号极性选择 [6] 比较器1中断触发类型 [5] 比较器1信号输入使能 [4] 比较器1极性选择 [3] 比较器0开窗PWM信号极性选择 [2] 比较器0中断触发类型 [1] 比较器0信号输入使能 [0] 比较器0极性选择
- [__IOM uint32_t BLCWIN](#)
 比较器黑窗口控制 [31:8] 保留 [7] 使用MCPWM模块CHN3.P通道输出的P管开关控制信号作为比较器1开窗使能 [6] 使用MCPWM模块CHN2.P通道输出的P管开关控制信号作为比较器1开窗使能 [5] 使用MCPWM模块CHN1.P通道输出的P管开关控制信号作为比较器1开窗使能 [4] 使用MCPWM模块CHN0.P通道输出的P管开关控制信号作为比较器1开窗使能 [3] 使用MCPWM模块CHN3.P通道输出的P管开关控制信号作为比较器0开窗使能 [2] 使用MCPWM模块CHN2.P通道输出的P管开关控制信号作为比较器0开窗使能 [1] 使用MCPWM模块CHN1.P通道输出的P管开关控制信号作为比较器0开窗使能 [0] 使用MCPWM模块CHN0.P通道输出的P管开关控制信号作为比较器0开窗使能
- [__IOM uint32_t DATA](#)
 比较器输出数据 [9] 比较器1经过滤波后的信号 [8] 比较器0经过滤波后的信号 [1] 比较器1原始输出信号 [0] 比较器0原始输出信号

3.6.1 详细描述

比较器寄存器 (CMP)

在文件 [hal_lks32mc09x_reg.h](#) 第 40 行定义。

3.6.2 结构体成员变量说明

3.6.2.1 BLCWIN

```
--IOM uint32_t BLCWIN
```

比较器黑窗口控制 [31:8] 保留 [7] 使用MCPWM模块CHN3.P通道输出的P管开关控制信号作为比较器1开窗使能 [6] 使用MCPWM模块CHN2.P通道输出的P管开关控制信号作为比较器1开窗使能 [5] 使用MCPWM模块CHN1.P通道输出的P管开关控制信号作为比较器1开窗使能 [4] 使用MCPWM模块CHN0.P通道输出的P管开关控制信号作为比较器1开窗使能 [3] 使用MCPWM模块CHN3.P通道输出的P管开关控制信号作为比较器0开窗使能 [2] 使用MCPWM模块CHN2.P通道输出的P管开关控制信号作为比较器0开窗使能 [1] 使用MCPWM模块CHN1.P通道输出的P管开关控制信号作为比较器0开窗使能 [0] 使用MCPWM模块CHN0.P通道输出的P管开关控制信号作为比较器0开窗使能

在文件 [hal_lks32mc09x_reg.h](#) 第 46 行定义.

3.6.2.2 CFG

```
--IOM uint32_t CFG
```

比较器配置 [7] 比较器1开窗PWM信号极性选择 [6] 比较器1中断触发类型 [5] 比较器1信号输入使能 [4] 比较器1极性选择 [3] 比较器0开窗PWM信号极性选择 [2] 比较器0中断触发类型 [1] 比较器0信号输入使能 [0] 比较器0极性选择

在文件 [hal_lks32mc09x_reg.h](#) 第 45 行定义.

3.6.2.3 DATA

```
--IOM uint32_t DATA
```

比较器输出数据 [9] 比较器1经过滤波后的信号 [8] 比较器0经过滤波后的信号 [1] 比较器1原始输出信号 [0] 比较器0原始输出信号

在文件 [hal_lks32mc09x_reg.h](#) 第 47 行定义.

3.6.2.4 IE

```
--IOM uint32_t IE
```

< CMP 结构体

比较器中断使能 [9] 比较器1 DMA请求使能 [8] 比较器0 DMA请求使能 [1] 比较器1中断使能 [0] 比较器0中断使能

在文件 [hal_lks32mc09x_reg.h](#) 第 42 行定义.

3.6.2.5 IF

```
--IOM uint32_t IF
```

比较器中断标志 [1] 比较器1中断标志 [0] 比较器0中断标志

在文件 [hal_lks32mc09x_reg.h](#) 第 43 行定义.

3.6.2.6 TCLK

```
--IOM uint32_t TCLK
```

比较器时钟控制 [26:24] 比较器0/1共用滤波时钟分频系数 [15:12] 比较器1滤波时钟分频 [11] 比较器1滤波时钟使能 [10:8] 比较器1滤波时钟分频 [7:4] 比较器0滤波时钟分频 [3] 比较器0滤波时钟使能 [2:0] 比较器0滤波时钟分频

在文件 [hal_lks32mc09x_reg.h](#) 第 44 行定义。

3.7 HAL_CRC_TypeDef结构体 参考

CRC0寄存器 (CRC0)

```
#include <hal_lks32mc09x_reg.h>
```

成员变量

- [__IOM uint32_t DR](#)
 < CRC0结构体
- [__IOM uint32_t CR](#)
 CRC控制寄存器 [12] 是否将CRC校验后的编码反转后输出 [9:8] 待编码数据反转类型 [5:4] 输出编码(多项式)位宽 [0] 与输入信息码进行CRC计算的数据来源
- [__IOM uint32_t INIT](#)
 CRC初始值寄存器 [31:0] 存放初始码
- [__IOM uint32_t POL](#)
 CRC系数寄存器 [31:0] 存放生成多项式对应的生成码

3.7.1 详细描述

CRC0寄存器 (CRC0)

在文件 [hal_lks32mc09x_reg.h](#) 第 65 行定义。

3.7.2 结构体成员变量说明

3.7.2.1 CR

```
--IOM uint32_t CR
```

CRC控制寄存器 [12] 是否将CRC校验后的编码反转后输出 [9:8] 待编码数据反转类型 [5:4] 输出编码(多项式)位宽 [0] 与输入信息码进行CRC计算的数据来源

在文件 [hal_lks32mc09x_reg.h](#) 第 68 行定义。

3.7.2.2 DR

__IOM uint32_t DR

< CRC0结构体

CRC数据寄存器 [31:0] 存放待编码的信息码和经CRC校验后的编码

在文件 [hal_lks32mc09x_reg.h](#) 第 67 行定义.

3.7.2.3 INIT

__IOM uint32_t INIT

CRC初始值寄存器 [31:0] 存放初始码

在文件 [hal_lks32mc09x_reg.h](#) 第 69 行定义.

3.7.2.4 POL

__IOM uint32_t POL

CRC系数寄存器 [31:0] 存放生成多项式对应的生成码

在文件 [hal_lks32mc09x_reg.h](#) 第 70 行定义.

3.8 HAL_DMA_TypeDef结构体 参考

DMA寄存器

```
#include <hal_lks32mc09x_reg.h>
```

成员变量

- __IOM uint32_t **CCR**
DMA通道0配置 [11:10] 源地址访问位宽 [9:8] 目的地址访问位宽 [6] 源地址递增方式 [4] 目的地址递增方式 [3] 循环模式 [1] 传输模式 [0] 通道使能
- __IOM uint32_t **REN**
DMA通道0请求使能 [11:0] DMA通道0数据搬运次数
- __IOM uint32_t **CTMS**
DMA通道0传输次数(保留字段)
- __IOM uint32_t **SADR**
DMA通道0源地址 [31:0] DMA通道x源地址
- __IOM uint32_t **DADR**
DMA通道0目的地址 [31:0] DMA通道x目的地址

3.8.1 详细描述

DMA寄存器

在文件 [hal_lks32mc09x_reg.h](#) 第 75 行定义.

3.8.2 结构体成员变量说明

3.8.2.1 CCR

```
--IOM uint32_t CCR
```

DMA通道0配置 [11:10] 源地址访问位宽 [9:8] 目的地址访问位宽 [6] 源地址递增方式 [4] 目的地址递增方式 [3] 循环模式 [1] 传输模式 [0] 通道使能

在文件 [hal_lks32mc09x_reg.h](#) 第 77 行定义.

3.8.2.2 CTMS

```
--IOM uint32_t CTMS
```

DMA通道0传输次数(保留字段)

在文件 [hal_lks32mc09x_reg.h](#) 第 79 行定义.

3.8.2.3 DADR

```
--IOM uint32_t DADR
```

DMA通道0目的地址 [31:0] DMA通道x目的地址

在文件 [hal_lks32mc09x_reg.h](#) 第 81 行定义.

3.8.2.4 REN

```
--IOM uint32_t REN
```

DMA通道0请求使能 [11:0] DMA通道0数据搬运次数

在文件 [hal_lks32mc09x_reg.h](#) 第 78 行定义.

3.8.2.5 SADR

```
--IOM uint32_t SADR
```

DMA通道0源地址 [31:0] DMA通道x源地址

在文件 [hal_lks32mc09x_reg.h](#) 第 80 行定义.

3.9 HAL_DMAMODULE_TypeDef结构体 参考

DMA控制状态寄存器对象定义

```
#include <hal_lks32mc09x_reg.h>
```

成员变量

- `__IOM uint32_t CTRL`
DMA控制 [0] DMA整体使能
- `__IOM uint32_t IE`
DMA中断使能 [3] 通道3完成中断使能 [2] 通道2完成中断使能 [1] 通道1完成中断使能 [0] 通道0完成中断使能
- `__IOM uint32_t IF`
DMA中断标志 [3] 通道3完成中断标志 [2] 通道2完成中断标志 [1] 通道1完成中断标志 [0] 通道0完成中断标志

3.9.1 详细描述

DMA控制状态寄存器对象定义

在文件 [hal_lks32mc09x_reg.h](#) 第 86 行定义.

3.9.2 结构体成员变量说明

3.9.2.1 CTRL

```
__IOM uint32_t CTRL
```

DMA控制 [0] DMA整体使能

在文件 [hal_lks32mc09x_reg.h](#) 第 88 行定义.

3.9.2.2 IE

```
__IOM uint32_t IE
```

DMA中断使能 [3] 通道3完成中断使能 [2] 通道2完成中断使能 [1] 通道1完成中断使能 [0] 通道0完成中断使能

在文件 [hal_lks32mc09x_reg.h](#) 第 89 行定义.

3.9.2.3 IF

```
__IOM uint32_t IF
```

DMA中断标志 [3] 通道3完成中断标志 [2] 通道2完成中断标志 [1] 通道1完成中断标志 [0] 通道0完成中断标志

在文件 [hal_lks32mc09x_reg.h](#) 第 90 行定义.

3.10 HAL_DSP_TypeDef结构体 参考

DSP0寄存器结构体 (DSP0)

```
#include <hal_lks32mc09x_reg.h>
```

成员变量

- __IM uint32_t **RESERVED** [1032]
 < DSP0 结构体
- __IOM uint32_t **DID**
 DSP 0次被除数 [31:0] DSP除法被除数寄存器,中断优先级高
- __IOM uint32_t **DIS**
 DSP 0次除数 [31:0] DSP除法除数寄存器,中断优先级高
- __IM uint32_t **QUO**
 DSP 0次商 [31:0] DSP商寄存器,中断优先级高
- __IM uint32_t **REM**
 DSP 0次余数 [31:0] DSP除法余数寄存器
- __IOM uint32_t **DID1**
 DSP 1次被除数 [31:0] DSP除法被除数寄存器,中断优先级低
- __IOM uint32_t **DIS1**
 DSP 1次除数 [31:0] DSP除法除数寄存器,中断优先级低
- __IM uint32_t **QUO1**
 DSP 1次商 [31:0] DSP商寄存器,中断优先级低
- __IM uint32_t **RESERVED1**
 保留位
- __IOM uint32_t **RAD**
 DSP 被开方数 [31:0] 协处理器被开方数寄存器
- __IM uint16_t **SQRT**
 DSP 平方根 [15:0] 协处理器平方根寄存器 [31:16] 保留,读出时恒为0
- __IM uint16_t **RESERVED2**
 保留位

3.10.1 详细描述

DSP0寄存器结构体 (DSP0)

在文件 [hal_lks32mc09x_reg.h](#) 第 95 行定义.

3.10.2 结构体成员变量说明

3.10.2.1 DID

```
__IOM uint32_t DID
```

DSP 0次被除数 [31:0] DSP除法被除数寄存器,中断优先级高

在文件 [hal_lks32mc09x_reg.h](#) 第 98 行定义.

3.10.2.2 DID1

```
--IOM uint32_t DID1
```

DSP 1次被除数 [31:0] DSP除法被除数寄存器,中断优先级低

在文件 [hal_lks32mc09x_reg.h](#) 第 102 行定义.

3.10.2.3 DIS

```
--IOM uint32_t DIS
```

DSP 0次除数 [31:0] DSP除法除数寄存器,中断优先级高

在文件 [hal_lks32mc09x_reg.h](#) 第 99 行定义.

3.10.2.4 DIS1

```
--IOM uint32_t DIS1
```

DSP 1次除数 [31:0] DSP除法除数寄存器,中断优先级低

在文件 [hal_lks32mc09x_reg.h](#) 第 103 行定义.

3.10.2.5 QUO

```
--IM uint32_t QUO
```

DSP 0次商 [31:0] DSP商寄存器,中断优先级高

在文件 [hal_lks32mc09x_reg.h](#) 第 100 行定义.

3.10.2.6 QUO1

```
--IM uint32_t QUO1
```

DSP 1次商 [31:0] DSP商寄存器,中断优先级低

在文件 [hal_lks32mc09x_reg.h](#) 第 104 行定义.

3.10.2.7 RAD

```
--IOM uint32_t RAD
```

DSP 被开方数 [31:0] 协处理器被开方数寄存器

在文件 [hal_lks32mc09x_reg.h](#) 第 106 行定义.

3.10.2.8 REM

```
--IM uint32_t REM
```

DSP 0次余数 [31:0] DSP除法余数寄存器

在文件 [hal_lks32mc09x_reg.h](#) 第 101 行定义.

3.10.2.9 RESERVED

```
--IM uint32_t RESERVED[1032]
```

< DSP0 结构体

保留位

在文件 [hal_lks32mc09x_reg.h](#) 第 97 行定义.

3.10.2.10 RESERVED1

```
--IM uint32_t RESERVED1
```

保留位

在文件 [hal_lks32mc09x_reg.h](#) 第 105 行定义.

3.10.2.11 RESERVED2

```
--IM uint16_t RESERVED2
```

保留位

在文件 [hal_lks32mc09x_reg.h](#) 第 108 行定义.

3.10.2.12 SQRT

```
--IM uint16_t SQRT
```

DSP 平方根 [15:0] 协处理器平方根寄存器 [31:16] 保留,读出时恒为0

在文件 [hal_lks32mc09x_reg.h](#) 第 107 行定义.

3.11 HAL_EXTI_TypeDef结构体 参考

EXTI寄存器 (EXTI)

```
#include <hal_lks32mc09x_reg.h>
```

成员变量

- `_IOM uint32_t CR0`
< EXT结构体
- `_IOM uint32_t CR1`
GPIO外部中断触发类型1 [15:14] GPIO 2[0]/P2[0]外部触发类型选择 [13:12] GPIO 2[13]/P2[13]外部触发类型选择 [11:10] GPIO 2[6]/P2[6]外部触发类型选择 [9:8] GPIO 2[5]/P2[5]外部触发类型选择 [7:6] GPIO 2[4]/P2[4]外部触发类型选择 [5:4] GPIO 2[3]/P2[3]外部触发类型选择 [3:2] GPIO 2[9]/P2[9]外部触发类型选择 [1:0] GPIO 2[7]/P2[7]外部触发类型选择
- `_IOM uint32_t IE`
GPIO中断使能 [10] GPIO2 DMA 请求使能 [9] GPIO1 DMA 请求使能 [8] GPIO0 DMA 请求使能 [2] GPIO2中断使能 [1] GPIO1中断使能 [0] GPIO0中断使能
- `_IOM uint32_t IF`
GPIO中断标志 [15] GPIO 2[0] / P2[0] 外部中断标志 [14] GPIO 2[13] / P2[13] 外部中断标志 [13] GPIO 2[6] / P2[6] 外部中断标志 [12] GPIO 2[5] / P2[5] 外部中断标志 [11] GPIO 2[4] / P2[4] 外部中断标志 [10] GPIO 2[3] / P2[3] 外部中断标志 [9] GPIO 2[9] / P2[9] 外部中断标志 [8] GPIO 2[7] / P2[7] 外部中断标志 [7] GPIO 1[0] / P1[0] 外部中断标志 [6] GPIO 0[15] / P0[15] 外部中断标志 [5] GPIO 0[14] / P0[14] 外部中断标志 [4] GPIO 0[13] / P0[13] 外部中断标志 [3] GPIO 0[12] / P0[12] 外部中断标志 [2] GPIO 0[11] / P0[11] 外部中断标志 [1] GPIO 0[2] / P0[2] 外部中断标志 [0] GPIO 0[0] / P0[0] 外部中断标志
- `_IOM uint32_t CLK0`
GPIO输出时钟选择 [4] 晶振时钟输出使能 [3] ADC时钟输出使能 [2] PLL输出使能 [1] HRC输出使能 [0] LRC输出使能
- `_IOM uint32_t SWAP`
MCPWM交换选项 [1:0] PWM信号交换控制 [31:0] 存放待编码的信息码和经CRC校验后的编码

3.11.1 详细描述

EXTI寄存器 (EXTI)

在文件 [hal_lks32mc09x_reg.h](#) 第 113 行定义。

3.11.2 结构体成员变量说明

3.11.2.1 CLK0

```
--IOM uint32_t CLK0
```

GPIO输出时钟选择 [4] 晶振时钟输出使能 [3] ADC时钟输出使能 [2] PLL输出使能 [1] HRC输出使能 [0] LRC输出使能

在文件 [hal_lks32mc09x_reg.h](#) 第 119 行定义。

3.11.2.2 CR0

```
--IOM uint32_t CR0
```

< EXTI结构体

GPIO外部中断触发类型0 [15:14] GPIO 1[0]/ P1[0]外部触发类型选择 [13:12] GPIO 0[15]/ P0[15]外部触发类型选择 [11:10] GPIO 0[14]/ P0[14]外部触发类型选择 [9:8] GPIO 0[13]/ P0[13]外部触发类型选择 [7:6] GPIO 0[12]/ P0[12]外部触发类型选择 [5:4] GPIO 0[11]/ P0[11]外部触发类型选择 [3:2] GPIO 0[2]/ P0[2]外部触发类型选择 [1:0] GPIO 0[0]/ P0[0]外部触发类型选择

在文件 [hal_lks32mc09x_reg.h](#) 第 115 行定义。

3.11.2.3 CR1

```
--IOM uint32_t CR1
```

GPIO外部中断触发类型1 [15:14] GPIO 2[0]/P2[0]外部触发类型选择 [13:12] GPIO 2[13]/P2[13]外部触发类型选择 [11:10] GPIO 2[6]/P2[6]外部触发类型选择 [9:8] GPIO 2[5]/P2[5]外部触发类型选择 [7:6] GPIO 2[4]/P2[4]外部触发类型选择 [5:4] GPIO 2[3]/P2[3]外部触发类型选择 [3:2] GPIO 2[9]/P2[9]外部触发类型选择 [1:0] GPIO 2[7]/P2[7]外部触发类型选择

在文件 [hal_lks32mc09x_reg.h](#) 第 116 行定义.

3.11.2.4 IE

```
--IOM uint32_t IE
```

GPIO中断使能 [10] GPIO2 DMA 请求使能 [9] GPIO1 DMA 请求使能 [8] GPIO0 DMA 请求使能 [2] GPIO2中断使能 [1] GPIO1中断使能 [0] GPIO0中断使能

在文件 [hal_lks32mc09x_reg.h](#) 第 117 行定义.

3.11.2.5 IF

```
--IOM uint32_t IF
```

GPIO中断标志 [15] GPIO 2[0] / P2[0] 外部中断标志 [14] GPIO 2[13] / P2[13] 外部中断标志 [13] GPIO 2[6] / P2[6] 外部中断标志 [12] GPIO 2[5] / P2[5] 外部中断标志 [11] GPIO 2[4] / P2[4] 外部中断标志 [10] GPIO 2[3] / P2[3] 外部中断标志 [9] GPIO 2[9] / P2[9] 外部中断标志 [8] GPIO 2[7] / P2[7] 外部中断标志 [7] GPIO 1[0] / P1[0] 外部中断标志 [6] GPIO 0[15] / P0[15] 外部中断标志 [5] GPIO 0[14] / P0[14] 外部中断标志 [4] GPIO 0[13] / P0[13] 外部中断标志 [3] GPIO 0[12] / P0[12] 外部中断标志 [2] GPIO 0[11] / P0[11] 外部中断标志 [1] GPIO 0[2] / P0[2] 外部中断标志 [0] GPIO 0[0] / P0[0] 外部中断标志

在文件 [hal_lks32mc09x_reg.h](#) 第 118 行定义.

3.11.2.6 SWAP

```
--IOM uint32_t SWAP
```

MCPWM交换选项 [1:0] PWM信号交换控制 [31:0] 存放待编码的信息码和经CRC校验后的编码

在文件 [hal_lks32mc09x_reg.h](#) 第 120 行定义.

3.12 HAL_FLASH_TypeDef结构体 参考

FLASH寄存器 (FLASH)

```
#include <hal_lks32mc09x_reg.h>
```

成员变量

- `__IOM uint32_t CFG`
 < FLASH结构体
- `__IOM uint32_t ADDR`
 Flash访问地址寄存器 [14:0] 地址寄存器
- `__OM uint32_t WDATA`
 Flash写数据寄存器 [31:0] 执行写入操作,写入FLASH的值
- `__IM uint32_t RDATA`
 Flash读数据寄存器 [31:0] 执行读取操作,读出FLASH的值
- `__OM uint32_t ERASE`
 Flash擦除寄存器 [31:0] 写入0x7654DCBA,触发擦除操作
- `__IM uint32_t PROTECT`
 读取以更新Flash保护状态 [31:0] 读取该寄存器,更新加密/解密状态
- `__IM uint32_t READY`
 Flash忙闲状态 [0] 1:FLASH处于Idle状态; 0:FLASH处于Busy状态
- `__IM uint32_t SIZE`
 Flash大小寄存器 [13] FLASH容量寄存器 [1] Flash存在不可读、可执行的库区域 [0] Flash存在可读、可执行、但禁止擦除的ROM区域
- `__IOM uint32_t NCFG`
 Flash NVR配置寄存器 [31] FLASH擦除使能 [27] FLASH编程使能 [23] FLASH地址递增使能(此寄存器,只能读取不能修改) [19] FLASH预取加速使能(此寄存器,只能读取不能修改) [15] FLASH擦除类型选择(此寄存器,只能读取不能修改) [11] 访问FLASH区域选择(修改此寄存器,无论写入值是0还是1,最终,只能被强制写成1) [7] 读取FLASH数据,等待开关(此寄存器,只能读取不能修改) [6:0] 编程/擦除时间基数寄存器(此寄存器,只能读取不能修改)
- `__IOM uint32_t NADDR`
 Flash NVR访问地址寄存器 [14:0] 地址寄存器
- `__OM uint32_t NWDATA`
 Flash NVR写数据寄存器 [31:0] 执行写入操作,写入FLASH的值
- `__IM uint32_t NRDATA`
 Flash NVR读数据寄存器 [31:0] 执行读取操作,读出FLASH的值
- `__OM uint32_t NERASE`
 Flash NVR擦除寄存器 [31:0] 写入0x7654DCBA,触发擦除操作
- `__OM uint32_t NKEY`
 Flash NVR密钥寄存器 [3:0] 当FLASH处于保护状态时,无法通过SWD读取,编程和擦除FLASH。此时,用户可以对此寄存器写入特殊值(0x0000.000F),使能开关,配置FLASH.NCFG/NADDR/NWDATA/NRDATA/NERASE寄存器,对NVR Sector 2进行读取,编程和擦除。
- `__IOM uint32_t EPSM`
 Main区域Sector擦除保护寄存器 [31] Main区域Sector 248~255擦除保护 [30] Main区域Sector 247~240擦除保护 [29] Main区域Sector 239~232擦除保护 [28] Main区域Sector 231~224擦除保护 [27] Main区域Sector 223~216擦除保护 [26] Main区域Sector 215~208擦除保护 [25] Main区域Sector 207~200擦除保护 [24] Main区域Sector 199_192擦除保护 [23] Main区域Sector 191_184擦除保护 [22] Main区域Sector 183_176擦除保护 [21] Main区域Sector 175_168擦除保护 [20] Main区域Sector 167_160擦除保护 [19] Main区域Sector 159_152擦除保护 [18] Main区域Sector 151_144擦除保护 [17] Main区域Sector 143_136擦除保护 [16] Main区域Sector 135_128擦除保护 [15] Main区域Sector 127_120擦除保护 [14] Main区域Sector 119_112擦除保护 [13] Main区域Sector 111~104擦除保护 [12] Main区域Sector 103_96擦除保护 [11] Main区域Sector 95_88擦除保护 [10] Main区域Sector 87_80擦除保护 [9] Main区域Sector 79_72擦除保护 [8] Main区域Sector 71_64擦除保护 [7] Main区域Sector 62~56擦除保护 [6] Main区域Sector 55_48擦除保护 [5] Main区域Sector 47_40擦除保护 [4] Main区域Sector 39_32擦除保护 [3] Main区域Sector 31_24擦除保护 [2] Main区域Sector 23_16擦除保护 [1] Main区域Sector 8~15擦除保护 [0] Main区域Sector 0~7擦除保护
- `__IM uint32_t RESERVED`
- `__IOM uint32_t EPSN`

NVR区域Sector擦除保护寄存器 [14] NVR区域Sector 14擦除保护 [13] NVR区域Sector 13擦除保护 [12] NVR区域Sector 12擦除保护 [11] NVR区域Sector 11擦除保护 [10] NVR区域Sector 10擦除保护 [9] NVR区域Sector 9擦除保护 [8] NVR区域Sector 8擦除保护 [7] NVR区域Sector 7擦除保护 [6] NVR区域Sector 6擦除保护 [5] NVR区域Sector 5擦除保护 [4] NVR区域Sector 4擦除保护 [3] NVR区域Sector 3擦除保护 [2] NVR区域Sector 2擦除保护 [1] NVR区域Sector 1擦除保护 [0] NVR区域Sector 0擦除保护

- __IOM uint32_t LIB_ADDR

库代码区域寄存器 [15:8] 库代码终止Sector [7:0] 库代码起始Sector

3.12.1 详细描述

FLASH寄存器 (FLASH)

在文件 [hal_lks32mc09x_reg.h](#) 第 125 行定义.

3.12.2 结构体成员变量说明

3.12.2.1 ADDR

```
__IOM uint32_t ADDR
```

Flash访问地址寄存器 [14:0] 地址寄存器

在文件 [hal_lks32mc09x_reg.h](#) 第 128 行定义.

3.12.2.2 CFG

```
__IOM uint32_t CFG
```

< FLASH结构体

Flash配置寄存器 [31] FLASH擦除使能 [27] FLASH编程使能 [23] FLASH地址递增使能 [19] FLASH预取加速使能 [15] FLASH擦除类型选择 [11] 访问FLASH区域选择 [7] 读取FLASH数据,等待开关 [6:0] 编程/擦除时间基数寄存器

在文件 [hal_lks32mc09x_reg.h](#) 第 127 行定义.

3.12.2.3 EPSM

```
__IOM uint32_t EPSM
```

Main区域Sector擦除保护寄存器 [31] Main区域Sector 248~255擦除保护 [30] Main区域Sector 247~240擦除保护 [29] Main区域Sector 239~232擦除保护 [28] Main区域Sector 231~224擦除保护 [27] Main区域Sector 223~216擦除保护 [26] Main区域Sector 215~208擦除保护 [25] Main区域Sector 207~200擦除保护 [24] Main区域Sector 199_192擦除保护 [23] Main区域Sector 191_184擦除保护 [22] Main区域Sector 183_176擦除保护 [21] Main区域Sector 175_168擦除保护 [20] Main区域Sector 167_160擦除保护 [19] Main区域Sector 159_152擦除保护 [18] Main区域Sector 151_144擦除保护 [17] Main区域Sector 143_136擦除保护 [16] Main区域Sector 135_128擦除保护 [15] Main区域Sector 127_120擦除保护 [14] Main区域Sector 119_112擦除保护 [13] Main区域Sector 111_104擦除保护 [12] Main区域Sector 103_96擦除保护 [11] Main区域Sector 95_88擦除保护 [10] Main区域Sector 87_80擦除保护 [9] Main区域Sector 79_72擦除保护 [8] Main区域Sector 71_64擦除保护 [7] Main区域Sector 62_56擦除保护 [6] Main区域Sector 55_48擦除保护 [5] Main区域Sector 47_40擦除保护 [4] Main区域Sector 39_32擦除保护 [3] Main区域Sector 31_24擦除保护 [2] Main区域Sector 23_16擦除保护 [1] Main区域Sector 8~15擦除保护 [0] Main区域Sector 0~7擦除保护

在文件 [hal_lks32mc09x_reg.h](#) 第 141 行定义.

3.12.2.4 EPSN

```
--IOM uint32_t EPSN
```

NVR区域Sector擦除保护寄存器 [14] NVR区域Sector 14擦除保护 [13] NVR区域Sector 13擦除保护 [12] NVR区域Sector 12擦除保护 [11] NVR区域Sector 11擦除保护 [10] NVR区域Sector 10擦除保护 [9] NVR区域Sector 9擦除保护 [8] NVR区域Sector 8擦除保护 [7] NVR区域Sector 7擦除保护 [6] NVR区域Sector 6擦除保护 [5] NVR区域Sector 5擦除保护 [4] NVR区域Sector 4擦除保护 [3] NVR区域Sector 3擦除保护 [2] NVR区域Sector 2擦除保护 [1] NVR区域Sector 1擦除保护 [0] NVR区域Sector 0擦除保护

在文件 [hal_lks32mc09x.reg.h](#) 第 143 行定义.

3.12.2.5 ERASE

```
--OM uint32_t ERASE
```

Flash擦除寄存器 [31:0] 写入0x7654DCBA,触发擦除操作

在文件 [hal_lks32mc09x.reg.h](#) 第 131 行定义.

3.12.2.6 LIB_ADDR

```
--IOM uint32_t LIB_ADDR
```

库代码区域寄存器 [15:8] 库代码终止Sector [7:0] 库代码起始Sector

在文件 [hal_lks32mc09x.reg.h](#) 第 144 行定义.

3.12.2.7 NADDR

```
--IOM uint32_t NADDR
```

Flash NVR访问地址寄存器 [14:0] 地址寄存器

在文件 [hal_lks32mc09x.reg.h](#) 第 136 行定义.

3.12.2.8 NCFG

```
--IOM uint32_t NCFG
```

Flash NVR配置寄存器 [31] FLASH擦除使能 [27] FLASH编程使能 [23] FLASH地址递增使能(此寄存器,只能读取不能修改) [19] FLASH预取加速使能(此寄存器,只能读取不能修改) [15] FLASH擦除类型选择(此寄存器,只能读取不能修改) [11] 访问FLASH区域选择(修改此寄存器,无论写入值是0还是1,最终,只能被强制写成1) [7] 读取FLASH数据,等待开关(此寄存器,只能读取不能修改) [6:0] 编程/擦除时间基数寄存器(此寄存器,只能读取不能修改)

在文件 [hal_lks32mc09x.reg.h](#) 第 135 行定义.

3.12.2.9 NERASE

```
__OM uint32_t NERASE
```

Flash NVR擦除寄存器 [31:0] 写入0x7654DCBA,触发擦除操作

在文件 [hal_lks32mc09x_reg.h](#) 第 139 行定义.

3.12.2.10 NKEY

```
__OM uint32_t NKEY
```

Flash NVR密钥寄存器 [3:0] 当FLASH处于保护状态时,无法通过SWD读取,编程和擦除FLASH。此时,用户可以对此寄存器写入特殊值(0x0000_000F),使能开关,配置FLASH_NCFG/NADDR/NWDATA/NRDATA/NERASE寄存器,对NVR Sector 2进行读取,编程和擦除。

在文件 [hal_lks32mc09x_reg.h](#) 第 140 行定义.

3.12.2.11 NRDATA

```
__IM uint32_t NRDATA
```

Flash NVR读数据寄存器 [31:0] 执行读取操作,读出FLASH的值

在文件 [hal_lks32mc09x_reg.h](#) 第 138 行定义.

3.12.2.12 NWDATA

```
__OM uint32_t NWDATA
```

Flash NVR写数据寄存器 [31:0] 执行写入操作,写入FLASH的值

在文件 [hal_lks32mc09x_reg.h](#) 第 137 行定义.

3.12.2.13 PROTECT

```
__IM uint32_t PROTECT
```

读取以更新Flash保护状态 [31:0] 读取该寄存器,更新加密/解密状态

在文件 [hal_lks32mc09x_reg.h](#) 第 132 行定义.

3.12.2.14 RDATA

```
__IM uint32_t RDATA
```

Flash读数据寄存器 [31:0] 执行读取操作,读出FLASH的值

在文件 [hal_lks32mc09x_reg.h](#) 第 130 行定义.

3.12.2.15 READY

```
--IM uint32_t READY
```

Flash忙/闲状态 [0] 1:FLASH处于Idle状态; 0:FLASH处于Busy状态

在文件 [hal_lks32mc09x_reg.h](#) 第 133 行定义.

3.12.2.16 RESERVED

```
--IM uint32_t RESERVED
```

在文件 [hal_lks32mc09x_reg.h](#) 第 142 行定义.

3.12.2.17 SIZE

```
--IM uint32_t SIZE
```

Flash大小寄存器 [13] FLASH容量寄存器 [1] Flash存在不可读、可执行的库区域 [0] Flash存在可读、可执行、但禁止擦除的ROM区域

在文件 [hal_lks32mc09x_reg.h](#) 第 134 行定义.

3.12.2.18 WDATA

```
--OM uint32_t WDATA
```

Flash写数据寄存器 [31:0] 执行写入操作,写入FLASH的值

在文件 [hal_lks32mc09x_reg.h](#) 第 129 行定义.

3.13 HAL_GPIO_TypeDef结构体 参考

GPIO寄存器

```
#include <hal_lks32mc09x_reg.h>
```

成员变量

• `_IOM uint32_t PIE`

GPIO0 输入使能 [15] GPIO x[15] / Px[15] 输入使能 [14] GPIO x[14] / Px[14] 输入使能 [13] GPIO x[13] / Px[13] 输入使能 [12] GPIO x[12] / Px[12] 输入使能 [11] GPIO x[11] / Px[11] 输入使能 [10] GPIO x[10] / Px[10] 输入使能 [9] GPIO x[9] / Px[9] 输入使能 [8] GPIO x[8] / Px[8] 输入使能 [7] GPIO x[7] / Px[7] 输入使能 [6] GPIO x[6] / Px[6] 输入使能 [5] GPIO x[5] / Px[5] 输入使能 [4] GPIO x[4] / Px[4] 输入使能 [3] GPIO x[3] / Px[3] 输入使能 [2] GPIO x[2] / Px[2] 输入使能 [1] GPIO x[1] / Px[1] 输入使能 [0] GPIO x[0] / Px[0] 输入使能

• `_IOM uint32_t POE`

GPIO0 输出使能 [15] GPIO x[15] / Px[15] 输出使能 [14] GPIO x[14] / Px[14] 输出使能 [13] GPIO x[13] / Px[13] 输出使能 [12] GPIO x[12] / Px[12] 输出使能 [11] GPIO x[11] / Px[11] 输出使能 [10] GPIO x[10] / Px[10] 输出使能 [9] GPIO x[9] / Px[9] 输出使能 [8] GPIO x[8] / Px[8] 输出使能 [7] GPIO x[7] / Px[7] 输出使能 [6] GPIO x[6] / Px[6] 输出使能 [5] GPIO x[5] / Px[5] 输出使能 [4] GPIO x[4] / Px[4] 输出使能 [3] GPIO x[3] / Px[3] 输出使能 [2] GPIO x[2] / Px[2] 输出使能 [1] GPIO x[1] / Px[1] 输出使能 [0] GPIO x[0] / Px[0] 输出使能

• `_IOM uint32_t PDI`

GPIO0 输入数据 [15:0] GPIO x 输入数据

• `_IOM uint32_t PDO`

GPIO0 输出数据 [15:0] GPIO x 输出数据

• `_IOM uint32_t PUE`

GPIO0 上拉使能 [15] GPIO x[15] / Px[15] 上拉使能 [14] GPIO x[14] / Px[14] 上拉使能 [13] GPIO x[13] / Px[13] 上拉使能 [12] GPIO x[12] / Px[12] 上拉使能 [11] GPIO x[11] / Px[11] 上拉使能 [10] GPIO x[10] / Px[10] 上拉使能 [9] GPIO x[9] / Px[9] 上拉使能 [8] GPIO x[8] / Px[8] 上拉使能 [7] GPIO x[7] / Px[7] 上拉使能 [6] GPIO x[6] / Px[6] 上拉使能 [5] GPIO x[5] / Px[5] 上拉使能 [4] GPIO x[4] / Px[4] 上拉使能 [3] GPIO x[3] / Px[3] 上拉使能 [2] GPIO x[2] / Px[2] 上拉使能 [1] GPIO x[1] / Px[1] 上拉使能 [0] GPIO x[0] / Px[0] 上拉使能

• `_IOM uint32_t RESERVED`• `_IOM uint32_t PODE`

GPIO0 开漏使能 [15] GPIO x[15] / Px[15] 开漏使能 [14] GPIO x[14] / Px[14] 开漏使能 [13] GPIO x[13] / Px[13] 开漏使能 [12] GPIO x[12] / Px[12] 开漏使能 [11] GPIO x[11] / Px[11] 开漏使能 [10] GPIO x[10] / Px[10] 开漏使能 [9] GPIO x[9] / Px[9] 开漏使能 [8] GPIO x[8] / Px[8] 开漏使能 [7] GPIO x[7] / Px[7] 开漏使能 [6] GPIO x[6] / Px[6] 开漏使能 [5] GPIO x[5] / Px[5] 开漏使能 [4] GPIO x[4] / Px[4] 开漏使能 [3] GPIO x[3] / Px[3] 开漏使能 [2] GPIO x[2] / Px[2] 开漏使能 [1] GPIO x[1] / Px[1] 开漏使能 [0] GPIO x[0] / Px[0] 开漏使能

• `_IOM uint32_t PFLT`

GPIO0 滤波使能 [15] GPIO x[15] / Px[15] 滤波使能 [14] GPIO x[14] / Px[14] 滤波使能 [13] GPIO x[13] / Px[13] 滤波使能 [12] GPIO x[12] / Px[12] 滤波使能 [11] GPIO x[11] / Px[11] 滤波使能 [10] GPIO x[10] / Px[10] 滤波使能 [9] GPIO x[9] / Px[9] 滤波使能 [8] GPIO x[8] / Px[8] 滤波使能 [7] GPIO x[7] / Px[7] 滤波使能 [6] GPIO x[6] / Px[6] 滤波使能 [5] GPIO x[5] / Px[5] 滤波使能 [4] GPIO x[4] / Px[4] 滤波使能 [3] GPIO x[3] / Px[3] 滤波使能 [2] GPIO x[2] / Px[2] 滤波使能 [1] GPIO x[1] / Px[1] 滤波使能 [0] GPIO x[0] / Px[0] 滤波使能

• `_IOM uint32_t F3210`

GPIO0[3:0] 功能选择 [15:12] GPIO0.3 功能选择 [11:8] GPIO0.2 功能选择 [7:4] GPIO0.1 功能选择 [3:0] GPIO0.0 功能选择

• `_IOM uint32_t F7654`

GPIO0[7:4] 功能选择 [15:12] GPIO0.7 功能选择 [11:8] GPIO0.6 功能选择 [7:4] GPIO0.5 功能选择 [3:0] GPIO0.4 功能选择

• `_IOM uint32_t FBA98`

GPIO0[11:8] 功能选择 [15:12] GPIO0.11 功能选择 [11:8] GPIO0.11 功能选择 [7:4] GPIO0.10 功能选择 [3:0] GPIO0.9 功能选择

• `_IOM uint32_t FFEDC`

GPIO0[15:12] 功能选择 [15:12] GPIO0.15 功能选择 [11:8] GPIO0.14 功能选择 [7:4] GPIO0.13 功能选择 [3:0] GPIO0.12 功能选择

• `_IOM uint32_t BSRR`

GPIO0 位设置复位寄存器 [31] 写 1 将 GPIO x[15] 清零, 写 0 无作用 [30] 写 1 将 GPIO x[14] 清零, 写 0 无作用 [29] 写 1 将 GPIO x[13] 清零, 写 0 无作用 [28] 写 1 将 GPIO x[12] 清零, 写 0 无作用 [27] 写 1 将 GPIO x[11] 清零, 写 0 无作用 [26] 写 1 将 GPIO x[10] 清零, 写 0 无作用 [25] 写 1 将 GPIO x[9] 清零, 写 0 无作用 [24] 写 1 将 GPIO x[8] 清零, 写 0 无作用 [23] 写 1 将 GPIO x[7] 清零, 写 0 无作用 [22] 写 1 将 GPIO x[6] 清零, 写 0 无作用 [21] 写 1 将 GPIO x[5] 清零, 写 0 无作用 [20] 写 1 将 GPIO x[4] 清零, 写 0 无作用 [19] 写 1 将 GPIO x[3] 清零, 写 0 无作用 [18] 写 1 将 GPIO x[2] 清零, 写 0 无作用 [17] 写 1 将 GPIO x[1] 清零, 写 0 无作用 [16] 写 1 将 GPIO x[0] 清零, 写 0 无作用 [15] 写 1 将 GPIO x[15] 置 1, 写 0 无作用 [14] 写 1 将 GPIO x[14] 置 1, 写 0 无作用 [13] 写 1 将 GPIO x[13] 置 1, 写 0 无作用 [12] 写 1 将 GPIO x[12] 置 1, 写 0 无作用

[11] 写1将GPIO x[11]置1,写0无作用 [10] 写1将GPIO x[10]置1,写0无作用 [9] 写1将GPIO x[9]置1,写0无作用 [8] 写1将GPIO x[8]置1,写0无作用 [7] 写1将GPIO x[7]置1,写0无作用 [6] 写1将GPIO x[6]置1,写0无作用 [5] 写1将GPIO x[5]置1,写0无作用 [4] 写1将GPIO x[4]置1,写0无作用 [3] 写1将GPIO x[3]置1,写0无作用 [2] 写1将GPIO x[2]置1,写0无作用 [1] 写1将GPIO x[1]置1,写0无作用 [0] 写1将GPIO x[0]置1,写0无作用

- `_OM uint32_t BRR`

GPIO0 位复位寄存器 [15] 写1将GPIO x[15]清零,写0无作用 [14] 写1将GPIO x[14]清零,写0无作用 [13] 写1将GPIO x[13]清零,写0无作用 [12] 写1将GPIO x[12]清零,写0无作用 [11] 写1将GPIO x[11]清零,写0无作用 [10] 写1将GPIO x[10]清零,写0无作用 [9] 写1将GPIO x[9]清零,写0无作用 [8] 写1将GPIO x[8]清零,写0无作用 [7] 写1将GPIO x[7]清零,写0无作用 [6] 写1将GPIO x[6]清零,写0无作用 [5] 写1将GPIO x[5]清零,写0无作用 [4] 写1将GPIO x[4]清零,写0无作用 [3] 写1将GPIO x[3]清零,写0无作用 [2] 写1将GPIO x[2]清零,写0无作用 [1] 写1将GPIO x[1]清零,写0无作用 [0] 写1将GPIO x[0]清零,写0无作用

3.13.1 详细描述

GPIO寄存器

在文件 [hal_lks32mc09x_reg.h](#) 第 149 行定义。

3.13.2 结构体成员变量说明

3.13.2.1 BRR

`_OM uint32_t BRR`

GPIO0 位复位寄存器 [15] 写1将GPIO x[15]清零,写0无作用 [14] 写1将GPIO x[14]清零,写0无作用 [13] 写1将GPIO x[13]清零,写0无作用 [12] 写1将GPIO x[12]清零,写0无作用 [11] 写1将GPIO x[11]清零,写0无作用 [10] 写1将GPIO x[10]清零,写0无作用 [9] 写1将GPIO x[9]清零,写0无作用 [8] 写1将GPIO x[8]清零,写0无作用 [7] 写1将GPIO x[7]清零,写0无作用 [6] 写1将GPIO x[6]清零,写0无作用 [5] 写1将GPIO x[5]清零,写0无作用 [4] 写1将GPIO x[4]清零,写0无作用 [3] 写1将GPIO x[3]清零,写0无作用 [2] 写1将GPIO x[2]清零,写0无作用 [1] 写1将GPIO x[1]清零,写0无作用 [0] 写1将GPIO x[0]清零,写0无作用

在文件 [hal_lks32mc09x_reg.h](#) 第 164 行定义。

3.13.2.2 BSRR

`_OM uint32_t BSRR`

GPIO0 位设置复位寄存器 [31] 写1将GPIO x[15]清零,写0无作用 [30] 写1将GPIO x[14]清零,写0无作用 [29] 写1将GPIO x[13]清零,写0无作用 [28] 写1将GPIO x[12]清零,写0无作用 [27] 写1将GPIO x[11]清零,写0无作用 [26] 写1将GPIO x[10]清零,写0无作用 [25] 写1将GPIO x[9]清零,写0无作用 [24] 写1将GPIO x[8]清零,写0无作用 [23] 写1将GPIO x[7]清零,写0无作用 [22] 写1将GPIO x[6]清零,写0无作用 [21] 写1将GPIO x[5]清零,写0无作用 [20] 写1将GPIO x[4]清零,写0无作用 [19] 写1将GPIO x[3]清零,写0无作用 [18] 写1将GPIO x[2]清零,写0无作用 [17] 写1将GPIO x[1]清零,写0无作用 [16] 写1将GPIO x[0]清零,写0无作用 [15] 写1将GPIO x[15]置1,写0无作用 [14] 写1将GPIO x[14]置1,写0无作用 [13] 写1将GPIO x[13]置1,写0无作用 [12] 写1将GPIO x[12]置1,写0无作用 [11] 写1将GPIO x[11]置1,写0无作用 [10] 写1将GPIO x[10]置1,写0无作用 [9] 写1将GPIO x[9]置1,写0无作用 [8] 写1将GPIO x[8]置1,写0无作用 [7] 写1将GPIO x[7]置1,写0无作用 [6] 写1将GPIO x[6]置1,写0无作用 [5] 写1将GPIO x[5]置1,写0无作用 [4] 写1将GPIO x[4]置1,写0无作用 [3] 写1将GPIO x[3]置1,写0无作用 [2] 写1将GPIO x[2]置1,写0无作用 [1] 写1将GPIO x[1]置1,写0无作用 [0] 写1将GPIO x[0]置1,写0无作用

在文件 [hal_lks32mc09x_reg.h](#) 第 163 行定义。

3.13.2.3 F3210

```
--IOM uint32_t F3210
```

GPIO0[3:0] 功能选择 [15:12] GPIO0.3功能选择 [11:8] GPIO0.2功能选择 [7:4] GPIO0.1功能选择 [3:0] GPIO0.0功能选择

在文件 [hal_lks32mc09x_reg.h](#) 第 159 行定义.

3.13.2.4 F7654

```
--IOM uint32_t F7654
```

GPIO0[7:4] 功能选择 [15:12] GPIO0.7功能选择 [11:8] GPIO0.6功能选择 [7:4] GPIO0.5功能选择 [3:0] GPIO0.4功能选择

在文件 [hal_lks32mc09x_reg.h](#) 第 160 行定义.

3.13.2.5 FBA98

```
--IOM uint32_t FBA98
```

GPIO0[11:8] 功能选择 [15:12] GPIO0.11功能选择 [11:8] GPIO0.11功能选择 [7:4] GPIO0.10功能选择 [3:0] GPIO0.9功能选择

在文件 [hal_lks32mc09x_reg.h](#) 第 161 行定义.

3.13.2.6 FFEDC

```
--IOM uint32_t FFEDC
```

GPIO0[15:12] 功能选择 [15:12] GPIO0.15功能选择 [11:8] GPIO0.14功能选择 [7:4] GPIO0.13功能选择 [3:0] GPIO0.12功能选择

在文件 [hal_lks32mc09x_reg.h](#) 第 162 行定义.

3.13.2.7 PDI

```
--IOM uint32_t PDI
```

GPIO0 输入数据 [15:0] GPIO x输入数据

在文件 [hal_lks32mc09x_reg.h](#) 第 153 行定义.

3.13.2.8 PDO

```
--IOM uint32_t PDO
```

GPIO0 输出数据 [15:0] GPIO x输出数据

在文件 [hal_lks32mc09x_reg.h](#) 第 154 行定义.

3.13.2.9 PFLT

--IOM uint32_t PFLT

GPIO0 滤波使能 [15] GPIO x[15] / Px[15] 滤波使能 [14] GPIO x[14] / Px[14] 滤波使能 [13] GPIO x[13] / Px[13] 滤波使能 [12] GPIO x[12] / Px[12] 滤波使能 [11] GPIO x[11] / Px[11] 滤波使能 [10] GPIO x[10] / Px[10] 滤波使能 [9] GPIO x[9] / Px[9] 滤波使能 [8] GPIO x[8] / Px[8] 滤波使能 [7] GPIO x[7] / Px[7] 滤波使能 [6] GPIO x[6] / Px[6] 滤波使能 [5] GPIO x[5] / Px[5] 滤波使能 [4] GPIO x[4] / Px[4] 滤波使能 [3] GPIO x[3] / Px[3] 滤波使能 [2] GPIO x[2] / Px[2] 滤波使能 [1] GPIO x[1] / Px[1] 滤波使能 [0] GPIO x[0] / Px[0] 滤波使能

在文件 [hal_lks32mc09x_reg.h](#) 第 158 行定义。

3.13.2.10 PIE

--IOM uint32_t PIE

GPIO0 输入使能 [15] GPIO x[15] / Px[15] 输入使能 [14] GPIO x[14] / Px[14] 输入使能 [13] GPIO x[13] / Px[13] 输入使能 [12] GPIO x[12] / Px[12] 输入使能 [11] GPIO x[11] / Px[11] 输入使能 [10] GPIO x[10] / Px[10] 输入使能 [9] GPIO x[9] / Px[9] 输入使能 [8] GPIO x[8] / Px[8] 输入使能 [7] GPIO x[7] / Px[7] 输入使能 [6] GPIO x[6] / Px[6] 输入使能 [5] GPIO x[5] / Px[5] 输入使能 [4] GPIO x[4] / Px[4] 输入使能 [3] GPIO x[3] / Px[3] 输入使能 [2] GPIO x[2] / Px[2] 输入使能 [1] GPIO x[1] / Px[1] 输入使能 [0] GPIO x[0] / Px[0] 输入使能

在文件 [hal_lks32mc09x_reg.h](#) 第 151 行定义。

3.13.2.11 PODE

--IOM uint32_t PODE

GPIO0 开漏使能 [15] GPIO x[15] / Px[15] 开漏使能 [14] GPIO x[14] / Px[14] 开漏使能 [13] GPIO x[13] / Px[13] 开漏使能 [12] GPIO x[12] / Px[12] 开漏使能 [11] GPIO x[11] / Px[11] 开漏使能 [10] GPIO x[10] / Px[10] 开漏使能 [9] GPIO x[9] / Px[9] 开漏使能 [8] GPIO x[8] / Px[8] 开漏使能 [7] GPIO x[7] / Px[7] 开漏使能 [6] GPIO x[6] / Px[6] 开漏使能 [5] GPIO x[5] / Px[5] 开漏使能 [4] GPIO x[4] / Px[4] 开漏使能 [3] GPIO x[3] / Px[3] 开漏使能 [2] GPIO x[2] / Px[2] 开漏使能 [1] GPIO x[1] / Px[1] 开漏使能 [0] GPIO x[0] / Px[0] 开漏使能

在文件 [hal_lks32mc09x_reg.h](#) 第 157 行定义。

3.13.2.12 POE

--IOM uint32_t POE

GPIO0 输出使能 [15] GPIO x[15] / Px[15] 输出使能 [14] GPIO x[14] / Px[14] 输出使能 [13] GPIO x[13] / Px[13] 输出使能 [12] GPIO x[12] / Px[12] 输出使能 [11] GPIO x[11] / Px[11] 输出使能 [10] GPIO x[10] / Px[10] 输出使能 [9] GPIO x[9] / Px[9] 输出使能 [8] GPIO x[8] / Px[8] 输出使能 [7] GPIO x[7] / Px[7] 输出使能 [6] GPIO x[6] / Px[6] 输出使能 [5] GPIO x[5] / Px[5] 输出使能 [4] GPIO x[4] / Px[4] 输出使能 [3] GPIO x[3] / Px[3] 输出使能 [2] GPIO x[2] / Px[2] 输出使能 [1] GPIO x[1] / Px[1] 输出使能 [0] GPIO x[0] / Px[0] 输出使能

在文件 [hal_lks32mc09x_reg.h](#) 第 152 行定义。

3.13.2.13 PUE

```
--IOM uint32_t PUE
```

GPIO0 上拉使能 [15] GPIO x[15] / Px[15] 上拉使能 [14] GPIO x[14] / Px[14] 上拉使能 [13] GPIO x[13] / Px[13] 上拉使能 [12] GPIO x[12] / Px[12] 上拉使能 [11] GPIO x[11] / Px[11] 上拉使能 [10] GPIO x[10] / Px[10] 上拉使能 [9] GPIO x[9] / Px[9] 上拉使能 [8] GPIO x[8] / Px[8] 上拉使能 [7] GPIO x[7] / Px[7] 上拉使能 [6] GPIO x[6] / Px[6] 上拉使能 [5] GPIO x[5] / Px[5] 上拉使能 [4] GPIO x[4] / Px[4] 上拉使能 [3] GPIO x[3] / Px[3] 上拉使能 [2] GPIO x[2] / Px[2] 上拉使能 [1] GPIO x[1] / Px[1] 上拉使能 [0] GPIO x[0] / Px[0] 上拉使能

在文件 [hal_lks32mc09x.reg.h](#) 第 155 行定义.

3.13.2.14 RESERVED

```
--IM uint32_t RESERVED
```

在文件 [hal_lks32mc09x.reg.h](#) 第 156 行定义.

3.14 HAL_HALL_TypeDef结构体 参考

HALL0寄存器 (HALL0)

```
#include <hal_lks32mc09x_reg.h>
```

成员变量

- `--IOM uint32_t CFG`
 < HALL0结构体
- `--IOM uint32_t INFO`
 霍尔信息寄存器 [18] 软件触发HALL信号变化中断 [17] HALL计数器溢出事件标志 [16] HALL信号变化事件标志 [15:11] 系统保留,必须写入0,读出0 [10:8] HALL值,未滤波结果 [7:3] 系统保留,必须写入0,读出0 [2:0] HALL值,滤波结果
- `--IM uint32_t WIDTH`
 霍尔宽度计数值寄存器 [23:0] HALL 宽度计数器值
- `--IOM uint32_t TH`
 霍尔模块计数器门限值寄存器 [23:0] HALL计数器门限值
- `--IOM uint32_t CNT`
 霍尔计数寄存器 [23:0] HALL计数值,写入任意值可清零

3.14.1 详细描述

HALL0寄存器 (HALL0)

在文件 [hal_lks32mc09x.reg.h](#) 第 169 行定义.

3.14.2 结构体成员变量说明

3.14.2.1 CFG

`__IOM uint32_t CFG`

< HALL0结构体

霍尔配置寄存器 [30] 软件触发HALL信号变化中断使能 [29] HALL计数器溢出中断使能开关 [28] HALL信号变化中断使能开关 [27] HALL计数器溢出DMA请求使能开关 [25] HALL信号变化DMA请求使能开关 [24] HALL模块使能开关 [20] 7/5滤波开关 [17:16] HALL时钟分频系数 [14:0] 滤波宽度

在文件 [hal_lks32mc09x_reg.h](#) 第 171 行定义.

3.14.2.2 CNT

`__IOM uint32_t CNT`

霍尔计数寄存器 [23:0] HALL计数值,写入任意值可清零

在文件 [hal_lks32mc09x_reg.h](#) 第 175 行定义.

3.14.2.3 INFO

`__IOM uint32_t INFO`

霍尔信息寄存器 [18] 软件触发HALL信号变化中断 [17] HALL计数器溢出事件标志 [16] HALL信号变化事件标志 [15:11] 系统保留,必须写入0,读出0 [10:8] HALL值,未滤波结果 [7:3] 系统保留,必须写入0,读出0 [2:0] HALL值,滤波结果

在文件 [hal_lks32mc09x_reg.h](#) 第 172 行定义.

3.14.2.4 TH

`__IOM uint32_t TH`

霍尔模块计数器门限值寄存器 [23:0] HALL计数器门限值

在文件 [hal_lks32mc09x_reg.h](#) 第 174 行定义.

3.14.2.5 WIDTH

`__IM uint32_t WIDTH`

霍尔宽度计数值寄存器 [23:0] HALL 宽度计数器值

在文件 [hal_lks32mc09x_reg.h](#) 第 173 行定义.

3.15 HAL_I2C_TypeDef结构体 参考

I2C0寄存器 (I2C0)

```
#include <hal_lks32mc09x_reg.h>
```

成员变量

- `__IOM uint32_t ADDR`
 < I2C0 结构体
- `__IOM uint32_t CFG`
 I2C配置 [7] I2C中断使能信号 [6] I2C数据传输完成中断使能信号 [5] I2C总线错误事件中断使能信号 [4] I2CSTOP事件中断使能信号 [1] I2C主模式使能信号 [0] I2C从模式使能信号
- `__IOM uint32_t SCR`
 I2C状态控制 [7] 总线错误状态标志位 [6] 总线仲裁丢失状态标志位 [5] STOP事件状态标志位 [4] 下一字节ACK位 [3] 地址字节标志位 [2] 发送或接收控制位 [1] 接收响应标志位 [0] 传输完成状态标志位
- `__IOM uint32_t DATA`
 I2C数据寄存器 [7:0] 数据寄存器,用于主模式发送/从模式发送/主模式接收/从模式接收
- `__IOM uint32_t MSCR`
 I2C主状态控制 [3] I2C总线,闲忙状态 [2] 主模式争抢总线标志位 [1] 再次触发START事件 [0] 触发START事件并发送地址数据至总线
- `__IOM uint32_t BCR`
 I2C传输控制寄存器 [7] I2C传输,NACK事件中断使能信号 [6] I2C传输,硬件地址匹配中断使能信号 [5] I2C多数数据传输使能 [4] I2C多数数据传输,从模式执行DMA方式发送,触发硬件预取第一个字节
- `__IOM uint32_t BSIZ`
 I2C传输长度寄存器 [4:0] I2C数据传输长度寄存器,用于多字节传输 实际传输字节数 = $B[3:0] + 1$

3.15.1 详细描述

I2C0寄存器 (I2C0)

在文件 [hal_lks32mc09x_reg.h](#) 第 180 行定义.

3.15.2 结构体成员变量说明

3.15.2.1 ADDR

```
__IOM uint32_t ADDR
```

< I2C0 结构体

I2C设备地址 [7] I2C硬件地址比较使能开关 [6:0] 主模式下,用于配置需要进行通信的从机地址; 从模式下,用于配置该节点的从机地址;

在文件 [hal_lks32mc09x_reg.h](#) 第 182 行定义.

3.15.2.2 BCR

```
--IOM uint32_t BCR
```

I2C传输控制寄存器 [7] I2C传输,NACK事件中断使能信号 [6] I2C传输,硬件地址匹配中断使能信号 [5] I2C多数据传输使能 [4] I2C多数据传输,从模式执行DMA方式发送,触发硬件预取第一个字节

在文件 [hal_lks32mc09x_reg.h](#) 第 187 行定义.

3.15.2.3 BSIZE

```
--IOM uint32_t BSIZE
```

I2C传输长度寄存器 [4:0] I2C数据传输长度寄存器,用于多字节传输 实际传输字节数 = B[3:0] + 1

在文件 [hal_lks32mc09x_reg.h](#) 第 188 行定义.

3.15.2.4 CFG

```
--IOM uint32_t CFG
```

I2C配置 [7] I2C中断使能信号 [6] I2C数据传输完成中断使能信号 [5] I2C总线错误事件中断使能信号 [4] I2CSTOP事件中断使能信号 [1] I2C主模式使能信号 [0] I2C从模式使能信号

在文件 [hal_lks32mc09x_reg.h](#) 第 183 行定义.

3.15.2.5 DATA

```
--IOM uint32_t DATA
```

I2C数据寄存器 [7:0] 数据寄存器,用于主模式发送/从模式发送/主模式接收/从模式接收

在文件 [hal_lks32mc09x_reg.h](#) 第 185 行定义.

3.15.2.6 MSCR

```
--IOM uint32_t MSCR
```

I2C主状态控制 [3] I2C总线,闲忙状态 [2] 主模式争抢总线标志位 [1] 再次触发START事件 [0] 触发START事件并发送地址数据至总线

在文件 [hal_lks32mc09x_reg.h](#) 第 186 行定义.

3.15.2.7 SCR

```
--IOM uint32_t SCR
```

I2C状态控制 [7] 总线错误状态标志位 [6] 总线仲裁丢失状态标志位 [5] STOP事件状态标志位 [4] 下一字节ACK位 [3] 地址字节标志位 [2] 发送或接收控制位 [1] 接收响应标志位 [0] 传输完成状态标志位

在文件 [hal_lks32mc09x_reg.h](#) 第 184 行定义.

3.16 HAL_IWDG_TypeDef结构体 参考

IWDG寄存器结构体句柄

```
#include <hal_lks32mc09x_reg.h>
```

成员变量

- `__IO uint32_t PSW`
独立看门狗密码寄存器 [15:0] 写入0xA6B4,才能对IWDG.CLR/ IWDG.RTH等进行写操作,对IWDG.CLR或IWDG.RTH的写操作会将密码清空,因此每次对看门狗IWDG.CLR/ IWDG.RTH寄存器进行写操作前都需要重新写入密码
- `__IO uint32_t CFG`
独立看门狗配置寄存器 [4] 深度休眠定时唤醒使能,0:禁用,1:使能 [0] WDG.EN 独立看门狗使能,0:禁用,1:使能。默认使能,写1置位,写0同时向[15:8]写入0x3C可清零。当看门狗被禁用时,不再产生复位信号,但仍可计数并产生定时唤醒信号
- `__IO uint32_t CLR`
独立看门狗清零寄存器 [15:0] 写入字节 16'b0111_1001_1000_110B0,高15位为密码,密码正确时,B[0]才能写入B[0]写入1,则重置WDT计数器为TH,且该bit写入后自动清零,写入0无效
- `__IO uint32_t WTH`
独立看门狗定时唤醒门限寄存器 [20:8] 看门狗定时唤醒门限值,看门狗使用64kHz LRC时钟从IWDG.RTH开始计数,递减计数至WTH产生唤醒信号。[7:0]恒为0
- `__IO uint32_t RTH`
独立看门狗超时复位门限寄存器 [20:8] 看门狗超时复位门限值,也是重新装载值。看门狗使用64kHz LRC时钟从RTH开始计数,计数至0x7产生复位。向该寄存器写入0x0,会被硬件强制改写为0x100。先向IWDG.CLR PSW写入正确密码才能改写IWDG.RTH寄存器。改写RTH同样具有重置看门狗计数器的作用,看门狗会从新的IWDG.RTH开始计数。[7:0]恒为0
- `__IO uint32_t CNT`
独立看门狗当前计数值寄存器 [20:0] 看门狗当前计数值,此数值≤ IWDG.TH

3.16.1 详细描述

IWDG寄存器结构体句柄

在文件 [hal_lks32mc09x_reg.h](#) 第 353 行定义。

3.16.2 结构体成员变量说明

3.16.2.1 CFG

```
__IO uint32_t CFG
```

独立看门狗配置寄存器 [4] 深度休眠定时唤醒使能,0:禁用,1:使能 [0] WDG.EN 独立看门狗使能,0:禁用,1:使能。默认使能,写1置位,写0同时向[15:8]写入0x3C可清零。当看门狗被禁用时,不再产生复位信号,但仍可计数并产生定时唤醒信号

在文件 [hal_lks32mc09x_reg.h](#) 第 356 行定义。

3.16.2.2 CLR

```
--IO uint32_t CLR
```

独立看门狗清零寄存器 [15:0] 写入字节 16'b0111_1001_1000_110B0,高15位为密码,密码正确时,B[0]才能写入B[0]写入1,则重置WDT计数器为TH,且该bit写入后自动清零,写入0无效

在文件 [hal_lks32mc09x_reg.h](#) 第 357 行定义.

3.16.2.3 CNT

```
--IO uint32_t CNT
```

独立看门狗当前计数值寄存器 [20:0] 看门狗当前计数值,此数值 $\leq$ IWDG.TH

在文件 [hal_lks32mc09x_reg.h](#) 第 360 行定义.

3.16.2.4 PSW

```
--IO uint32_t PSW
```

独立看门狗密码寄存器 [15:0] 写入0xA6B4,才能对IWDG_CLR/ IWDG_RTH等进行写操作,对IWDG_CLR或IWDG_RTH的写操作会将密码清空,因此每次对看门狗IWDG_CLR/ IWDG_RTH寄存器进行写操作前都需要重新写入密码

在文件 [hal_lks32mc09x_reg.h](#) 第 355 行定义.

3.16.2.5 RTH

```
--IO uint32_t RTH
```

独立看门狗超时复位门限寄存器 [20:8] 看门狗超时复位门限值,也是重新装载值。看门狗使用64kHz LRC时钟从RTH开始计数,计数至0x7产生复位。向该寄存器写入0x0,会被硬件强制改写为0x100。先向IWDG_PSW写入正确密码才能改写IWDG_RTH寄存器。改写RTH同样具有重置看门狗计数器的作用,看门狗会从新的IWDG_RTH开始计数。[7:0] 恒为0

在文件 [hal_lks32mc09x_reg.h](#) 第 359 行定义.

3.16.2.6 WTH

```
--IO uint32_t WTH
```

独立看门狗定时唤醒门限寄存器 [20:8] 看门狗定时唤醒门限值,看门狗使用64kHz LRC时钟从IWDG_RTH开始计数,递减计数至WTH产生唤醒信号。[7:0] 恒为0

在文件 [hal_lks32mc09x_reg.h](#) 第 358 行定义.

3.17 HAL_MCPWM_TypeDef结构体 参考

MCPWM0寄存器 (MCPWM0)

```
#include <hal_lks32mc09x_reg.h>
```

成员变量

- `__IOM uint32_t TH [8]`
 < MCPWM0结构体
- `__IM uint32_t RESERVED [4]`
- `__IOM uint32_t TMR0`
 ADC触发时间0 [15:0]ADC采样定时器比较门限0寄存器
- `__IOM uint32_t TMR1`
 ADC触发时间1 [15:0]ADC采样定时器比较门限1寄存器
- `__IOM uint32_t TMR2`
 ADC触发时间2 [15:0]ADC采样定时器比较门限2寄存器
- `__IOM uint32_t TMR3`
 ADC触发时间3 [15:0]ADC采样定时器比较门限3寄存器
- `__IOM uint32_t TH0`
 基础计数器0 15位无符号阈值 [14:0]MCPWM时基0计数器门限值
- `__IM uint32_t RESERVED1`
- `__IOM uint32_t CNT0`
 基础计数器0 16位有符号阈值 [15:0]向此寄存器写入,更改时基0计数器的设定值
- `__IM uint32_t RESERVED2`
- `__IOM uint32_t UPDATE`
 掩码寄存器手动重载配置 [28] 写1产生软件(手动)触发,将MCPWM0.IO3/4/5由预装载寄存器加载到MCPWM运行所使用的影子寄存器 [25] 写1产生软件(手动)触发,将MCPWM0.TMR3由预装载寄存器加载到MCPWM运行所使用的影子寄存器 [24] 写1产生软件(手动)触发,将MCPWM0.TMR2由预装载寄存器加载到MCPWM运行所使用的影子寄存器 [17] 写1产生软件(手动)触发,将MCPWM0.TH31由预装载寄存器加载到MCPWM运行所使用的影子寄存器 [16] 写1产生软件(手动)触发,将MCPWM0.TH30由预装载寄存器加载到MCPWM运行所使用的影子寄存器 [12] 写1产生软件(手动)触发,将MCPWM0.IO0/1/2由预装载寄存器加载到MCPWM运行所使用的影子寄存器 [11] 写1产生软件(手动)触发,将MCPWM0.CNT0由预装载寄存器加载到MCPWM运行所使用的影子寄存器 [10] 写1产生软件(手动)触发,将MCPWM0.TH0由预装载寄存器加载到MCPWM运行所使用的影子寄存器 [9] 写1产生软件(手动)触发,将MCPWM0.TMR1由预装载寄存器加载到MCPWM运行所使用的影子寄存器 [8] 写1产生软件(手动)触发,将MCPWM0.TMR0由预装载寄存器加载到MCPWM运行所使用的影子寄存器 [5] 写1产生软件(手动)触发,将MCPWM0.TH21由预装载寄存器加载到MCPWM运行所使用的影子寄存器 [4] 写1产生软件(手动)触发,将MCPWM0.TH20由预装载寄存器加载到MCPWM运行所使用的影子寄存器 [3] 写1产生软件(手动)触发,将MCPWM0.TH11由预装载寄存器加载到MCPWM运行所使用的影子寄存器 [2] 写1产生软件(手动)触发,将MCPWM0.TH10由预装载寄存器加载到MCPWM运行所使用的影子寄存器 [1] 写1产生软件(手动)触发,将MCPWM0.TH01由预装载寄存器加载到MCPWM运行所使用的影子寄存器 [0] 写1产生软件(手动)触发,将MCPWM0.TH00由预装载寄存器加载到MCPWM运行所使用的影子寄存器
- `__IOM uint32_t FCNT`
 若MCPWM.FAIL012[15] = 1,当fail0/1发生时,将MCPWM.CNT0存储到MCPWM.FCNT [15:0] 若MCPWM0.CH←FAIL[15]=1,当发生fail0/1事件时,记录MCPWM0.CNT0值
- `__IOM uint32_t EVT0`
 MCPWM基础0外部触发寄存器 [15] TIMER3 CMP1事件触发时基0开始计数 [14] TIMER3 CMP0事件触发时基0开始计数 [13] TIMER2 CMP1事件触发时基0开始计数 [12] TIMER2 CMP0事件触发时基0开始计数 [11] TIMER1 CMP1事件触发时基0开始计数 [10] TIMER1 CMP0事件触发时基0开始计数 [9] TIMER0 CMP1事件触发时基0开始计数 [8] TIMER0 CMP0事件触发时基0开始计数 [3] MCPWM0 TMR3事件触发时基0开始计数 [2] MCPWM0 TMR2事件触发时基0开始计数 [1] MCPWM0 TMR1事件触发时基0开始计数 [0] MCPWM0 TMR0事件触发时基0开始计数
- `__IM uint32_t RESERVED3`
- `__IOM uint32_t DTH00`

- 通道0/1/2 N死区控制寄存器,10位无符号 [9:0] MCPWM CH0/1/2 P通道死区宽度控制寄存器
- `_IOM uint32_t DTH01`
 - 通道0/1/2 P死区控制寄存器,10位无符号 [9:0] MCPWM CH0/1/2 N通道死区宽度控制寄存器
- `_IM uint32_t RESERVED4` [2]
- `_IOM uint32_t FLT`
 - FAIL信号滤波配置 [7:0] 滤波时钟分频寄存器
- `_IOM uint32_t SDCFG`
 - 影子寄存器重载配置 [6] 更新事件是否自动清除MCPWM0 EIF[5:4]并置位MCPWM0.CH012.FAIL.MOE [5] 时基0 t1(过零)事件更新使能 [4] 时基0 t0(起点)事件更新使能 [3:0] 时基0更新间隔
- `_IOM uint32_t AUEN`
 - 自动更新使能 [25] MCPWM0.TMR3自动加载使能 [24] MCPWM0.TMR2自动加载使能 [17] MCPWM0.TH31自动加载使能 [16] MCPWM0.TH30自动加载使能 [12] MCPWM0.IO0/1/2自动加载使能 [11] MCPWM0.CNT0自动加载使能 [10] MCPWM0.TH0自动加载使能 [9] MCPWM0.TMR1自动加载使能 [8] MCPWM0.TMR0自动加载使能 [5] MCPWM0.TH21自动加载使能 [4] MCPWM0.TH20自动加载使能 [3] MCPWM0.TH11自动加载使能 [2] MCPWM0.TH10自动加载使能 [1] MCPWM0.TH01自动加载使能 [0] MCPWM0.TH00自动加载使能
- `_IOM uint32_t TCLK`
 - MCPWM时钟配置寄存器 [14:12] MCPWM工作时钟分频寄存器 [8] 时基0外部触发使能 [6] MCPWM时基0计数器使能开关 [2] MCPWM工作时钟使能
- `_IOM uint32_t IE0`
 - 基础计数器0中断源使能 [14] 时基0更新事件中断源使能 [13] MCPWM0.CNT0等于MCPWM0.TMR3中断源使能 [12] MCPWM0.CNT0等于MCPWM0.TMR2中断源使能 [11] MCPWM0.CNT0等于MCPWM0.TMR1中断源使能 [10] MCPWM0.CNT0等于MCPWM0.TMR0中断源使能 [9] MCPWM0.CNT0等于MCPWM0.TH31中断源使能 [8] MCPWM0.CNT0等于MCPWM0.TH30中断源使能 [7] MCPWM0.CNT0等于MCPWM0.TH21中断源使能 [6] MCPWM0.CNT0等于MCPWM0.TH20中断源使能 [5] MCPWM0.CNT0等于MCPWM0.TH11中断源使能 [4] MCPWM0.CNT0等于MCPWM0.TH10中断源使能 [3] MCPWM0.CNT0等于MCPWM0.TH01中断源使能 [2] MCPWM0.CNT0等于MCPWM0.TH00中断源使能 [1] 时基0 T1事件,MCPWM0.CNT0的计数值回0中断源使能 [0] 时基0 T0事件,MCPWM0.CNT0的计数值等于MCPWM0.TH0中断源使能
- `_IOM uint32_t IF0`
 - 基础计数器0中断标志 [14] 时基0更新事件MCPWM0.TH0/MCPWM0.TH00~MCPWM0.TH21/MCPWM0.TMR等寄存器更新到MCPWM实际运行系统的中断事件 [13] MCPWM0.CNT0等于MCPWM0.TMR3中断事件 [12] MCPWM0.CNT0等于MCPWM0.TMR2中断事件 [11] MCPWM0.CNT0等于MCPWM0.TMR1中断事件 [10] MCPWM0.CNT0等于MCPWM0.TMR0中断事件 [7] MCPWM0.CNT0等于MCPWM0.TH21中断事件 [6] MCPWM0.CNT0等于MCPWM0.TH20中断事件 [5] MCPWM0.CNT0等于MCPWM0.TH11中断事件 [4] MCPWM0.CNT0等于MCPWM0.TH10中断事件 [3] MCPWM0.CNT0等于MCPWM0.TH01中断事件 [2] MCPWM0.CNT0等于MCPWM0.TH00中断事件 [1] t1事件,MCPWM0.CNT0等于0中断事件 [0] t0事件,MCPWM0.CNT0等于MCPWM0.TH中断事件
- `_IM uint32_t RESERVED5` [2]
- `_IOM uint32_t EIE`
 - 错误中断源使能 [5] FAIL1中断源使能 [4] FAIL0中断源使能
- `_IOM uint32_t EIF`
 - 错误中断标志 [5] FAIL1中断源事件 [4] FAIL0中断源事件
- `_IOM uint32_t RE`
 - DMA请求使能 [13] 时基0 T1事件DMA请求1使能 [12] 时基0 T0事件DMA请求1使能 [11] MCPWM计数器命中TMR3,DMA请求1使能 [10] MCPWM计数器命中TMR2,DMA请求1使能 [9] MCPWM计数器命中TMR1,DMA请求1使能 [8] MCPWM计数器命中TMR0,DMA请求1使能 [5] 时基0 T1事件DMA请求0使能 [4] 时基0 T0事件DMA请求0使能 [3] MCPWM计数器命中TMR3,DMA请求0使能 [2] MCPWM计数器命中TMR2,DMA请求0使能 [1] MCPWM计数器命中TMR1,DMA请求0使能 [0] MCPWM计数器命中TMR0,DMA请求0使能
- `_IOM uint32_t PP`
 - 推挽配置 [3] IO3推挽模式使能信号 [2] IO2推挽模式使能信号 [1] IO1推挽模式使能信号 [0] IO0推挽模式使能信号
- `_IOM uint32_t IO01`
 - 通道0/1设置 [15] CH1工作模式选择 [14] CH1的P和N通道输出互换选择 [13] 当CH1.PS=1时,输出到CH1 P通道的值 [12] 当CH1.NS=1时,输出到CH1 N通道的值 [11] CH1 P来源 [10] CH1 N来源 [9] CH1 P极性选择 [8] CH1 N极性选择 [7] CH0工作模式选择 [6] CH0的P和N通道输出互换选择 [5] 当CH0.PS =1时,输出到CH0 P通道的值 [4] 当CH0.NS =1时,输出到CH0 N通道的值 [3] CH0 P来源 [2] CH0 N来源 [1] CH0 P极性选择 [0] CH0 N极性选择
- `_IOM uint32_t IO23`

通道2/3设置 [15] CH3工作模式选择 [14] CH3的P和N通道输出互换选择 [13] 当CH3.PS = 1时,输出到CH3 P通道的值 [12] 当CH3.NS = 1时,输出到CH3 N通道的值 [11] CH3 P来源 [10] CH3 N来源 [9] CH3 P极性选择 [8] CH3 N极性选择 [7] CH2工作模式选择 [6] CH2的P和N通道输出互换选择 [5] 当CH2.PS = 1时,输出到CH2 P通道的值 [4] 当CH2.NS = 1时,输出到CH2 N通道的值 [3] CH2 P来源 [2] CH2 N来源 [1] CH2 P极性选择 [0] CH2 N极性选择

- __IM uint32_t RESERVED6 [2]

- __IOM uint32_t CH_FAIL

MCPWM通道短路控制寄存器 [15] 当发生fail0/1事件时,将MCPWM0.CNT0值存入MCPWM0.FCNT [11:10] FAIL1来源选择 [9:8] FAIL0来源选择 [7] MCU进入HALT状态,MCPWM输出值选择 [6] MOE控制MCPWM通道P和N输出值 [5] FAIL1输入使能 [4] FAIL0输入使能 [3] FAIL1极性选择 [2] FAIL0极性选择

- __IM uint32_t RESERVED7

- __IOM uint32_t CH_DEF

MCPWM通道默认值 [7] CH3 P通道默认值 [6] CH3 N通道默认值 [5] CH2.P通道默认值 [4] CH2.N通道默认值 [3] CH1.P通道默认值 [2] CH1.N通道默认值 [1] CH0.P通道默认值 [0] CH0.N通道默认值

- __IOM uint32_t CH_MSK

MCPWM通道故障掩码 [7] CH3.P通道FAIL事件使能 [6] CH3.N通道FAIL事件使能 [5] CH2.P通道FAIL事件使能 [4] CH2.N通道FAIL事件使能 [3] CH1.P通道FAIL事件使能 [2] CH1.N通道 FAIL事件使能 [1] CH0.P通道FAIL事件使能 [0] CH0.N通道FAIL事件使能

- __OM uint32_t PRT

MCPWM寄存器保护 [15:0] 写入0xDEAD,解除MCPWM寄存器写保护; 写入其它值,MCPWM寄存器进入写保护

- __IM uint32_t RESERVED8 [3]

- __IOM uint32_t SW_FAIL

软件故障测试寄存器 [5] FAIL1软件测试信号使能 [4] FAIL0软件测试信号使能 [1] FAIL1软件测试信号 [0] FAIL0软件测试信号

3.17.1 详细描述

MCPWM0寄存器 (MCPWM0)

在文件 [hal_lks32mc09x_reg.h](#) 第 193 行定义.

3.17.2 结构体成员变量说明

3.17.2.1 AUEN

__IOM uint32_t AUEN

自动更新使能 [25] MCPWM0.TMR3自动加载使能 [24] MCPWM0.TMR2自动加载使能 [17] MCPWM0.TH31自动加载使能 [16] MCPWM0.TH30自动加载使能 [12] MCPWM0.IO0/1/2自动加载使能 [11] MCPWM0.CNT0自动加载使能 [10] MCPWM0.TH0自动加载使能 [9] MCPWM0.TMR1自动加载使能 [8] MCPWM0.TMR0自动加载使能 [5] MCPWM0.TH21自动加载使能 [4] MCPWM0.TH20自动加载使能 [3] MCPWM0.TH11自动加载使能 [2] MCPWM0.TH10自动加载使能 [1] MCPWM0.TH01自动加载使能 [0] MCPWM0.TH00自动加载使能

在文件 [hal_lks32mc09x_reg.h](#) 第 214 行定义.

3.17.2.2 CH_DEF

__IOM uint32_t CH_DEF

MCPWM通道默认值 [7] CH3 P通道默认值 [6] CH3 N通道默认值 [5] CH2.P通道默认值 [4] CH2.N通道默认值 [3] CH1.P通道默认值 [2] CH1.N通道默认值 [1] CH0.P通道默认值 [0] CH0.N通道默认值

在文件 [hal_lks32mc09x_reg.h](#) 第 228 行定义.

3.17.2.3 CH_FAIL

```
--IOM uint32_t CH_FAIL
```

MCPWM通道短路控制寄存器 [15] 当发生fail0/1事件时,将MCPWM0.CNT0值存入MCPWM0.FCNT [11:10] FAIL1来源选择 [9:8] FAIL0来源选择 [7] MCU进入HALT状态,MCPWM输出值选择 [6] MOE控制MCPWM通道P和N输出值 [5] FAIL1输入使能 [4] FAIL0输入使能 [3] FAIL1极性选择 [2] FAIL0极性选择

在文件 [hal_lks32mc09x_reg.h](#) 第 226 行定义.

3.17.2.4 CH_MSK

```
--IOM uint32_t CH_MSK
```

MCPWM通道故障掩码 [7] CH3_P 通道FAIL事件使能 [6] CH3_N 通道FAIL事件使能 [5] CH2_P 通道FAIL事件使能 [4] CH2_N 通道FAIL事件使能 [3] CH1_P 通道FAIL事件使能 [2] CH1_N通道 FAIL事件使能 [1] CH0_P 通道FAIL事件使能 [0] CH0_N 通道FAIL事件使能

在文件 [hal_lks32mc09x_reg.h](#) 第 229 行定义.

3.17.2.5 CNT0

```
--IOM uint32_t CNT0
```

基础计数器0 16位有符号阈值 [15:0]向此寄存器写入,更改时基0计数器的设定值

在文件 [hal_lks32mc09x_reg.h](#) 第 203 行定义.

3.17.2.6 DTH00

```
--IOM uint32_t DTH00
```

通道0/1/2 N死区控制寄存器,10位无符号 [9:0] MCPWM CH0/1/2 P通道死区宽度控制寄存器

在文件 [hal_lks32mc09x_reg.h](#) 第 209 行定义.

3.17.2.7 DTH01

```
--IOM uint32_t DTH01
```

通道0/1/2 P死区控制寄存器,10位无符号 [9:0] MCPWM CH0/1/2 N通道死区宽度控制寄存器

在文件 [hal_lks32mc09x_reg.h](#) 第 210 行定义.

3.17.2.8 EIE

```
--IOM uint32_t EIE
```

错误中断源使能 [5] FAIL1中断源使能 [4] FAIL0中断源使能

在文件 [hal_lks32mc09x_reg.h](#) 第 219 行定义.

3.17.2.9 EIF

--IOM uint32_t EIF

错误中断标志 [5] FAIL1中断源事件 [4] FAIL0中断源事件

在文件 [hal_iks32mc09x_reg.h](#) 第 220 行定义.

3.17.2.10 EVT0

--IOM uint32_t EVT0

MCPWM基础0外部触发寄存器 [15] TIMER3 CMP1事件触发时基0开始计数 [14] TIMER3 CMP0事件触发时基0开始计数 [13] TIMER2 CMP1事件触发时基0开始计数 [12] TIMER2 CMP0事件触发时基0开始计数 [11] TIMER1 CMP1事件触发时基0开始计数 [10] TIMER1 CMP0事件触发时基0开始计数 [9] TIMER0 CMP1事件触发时基0开始计数 [8] TIMER0 CMP0事件触发时基0开始计数 [3] MCPWM0 TMR3事件触发时基0开始计数 [2] MCPWM0 TMR2事件触发时基0开始计数 [1] MCPWM0 TMR1事件触发时基0开始计数 [0] MCPWM0 TMR0事件触发时基0开始计数

在文件 [hal_iks32mc09x_reg.h](#) 第 207 行定义.

3.17.2.11 FCNT

--IOM uint32_t FCNT

若MCPWM_FAIL012[15] = 1,当fail0/1发生时,将MCPWM_CNT0存储到MCPWM_FCNT [15:0] 若MCPWM0_CH↔ FAIL[15]=1,当发生fail0/1事件时,记录MCPWM0_CNT0值

在文件 [hal_iks32mc09x_reg.h](#) 第 206 行定义.

3.17.2.12 FLT

--IOM uint32_t FLT

FAIL信号滤波配置 [7:0] 滤波时钟分频寄存器

在文件 [hal_iks32mc09x_reg.h](#) 第 212 行定义.

3.17.2.13 IE0

--IOM uint32_t IE0

基础计数器0中断源使能 [14] 时基0更新事件中断源使能 [13] MCPWM0_CNT0等于MCPWM0_TMR3中断源使能 [12] MCPWM0_CNT0等于MCPWM0_TMR2中断源使能 [11] MCPWM0_CNT0等于MCPWM0_TMR1中断源使能 [10] MCPWM0_CNT0等于MCPWM0_TMR0中断源使能 [9] MCPWM0_CNT0等于MCPWM0_TH31中断源使能 [8] MCPWM0_CNT0等于MCPWM0_TH30中断源使能 [7] MCPWM0_CNT0等于MCPWM0_TH21中断源使能 [6] MCPWM0_CNT0等于MCPWM0_TH20中断源使能 [5] MCPWM0_CNT0等于MCPWM0_TH11中断源使能 [4] MCPWM0_CNT0等于MCPWM0_TH10中断源使能 [3] MCPWM0_CNT0等于MCPWM0_TH01中断源使能 [2] MCPWM0_CNT0等于MCPWM0_TH00中断源使能 [1] 时基0 T1事件,MCPWM0_CNT0的计数值回0中断源使能 [0] 时基0 T0事件,MCPWM0_CNT0的计数值等于MCPWM0_TH0中断源使能

在文件 [hal_iks32mc09x_reg.h](#) 第 216 行定义.

3.17.2.14 IF0

```
--IOM uint32_t IF0
```

基础计数器0中断标志 [14] 时基0更新事件MCPWM0_TH0/MCPWM0_TH00~MCPWM0_TH21/MCPWM0_TMR等寄存器更新到MCPWM实际运行系统的中断事件 [13] MCPWM0_CNT0等于MCPWM0_TMR3中断事件 [12] MCPWM0_CNT0等于MCPWM0_TMR2中断事件 [11] MCPWM0_CNT0等于MCPWM0_TMR1中断事件 [10] MCPWM0_CNT0等于MCPWM0_TMR0中断事件 [7] MCPWM0_CNT0等于MCPWM0_TH21中断事件 [6] MCPWM0_CNT0等于MCPWM0_TH20中断事件 [5] MCPWM0_CNT0等于MCPWM0_TH11中断事件 [4] MCPWM0_CNT0等于MCPWM0_TH10中断事件 [3] MCPWM0_CNT0等于MCPWM0_TH01中断事件 [2] MCPWM0_CNT0等于MCPWM0_TH00中断事件 [1] t1事件,MCPWM0_CNT0等于0中断事件 [0] t0事件,MCPWM0_CNT0等于MCPWM0_TH中断事件

在文件 [hal_lks32mc09x.reg.h](#) 第 217 行定义.

3.17.2.15 IO01

```
--IOM uint32_t IO01
```

通道0/1设置 [15] CH1工作模式选择 [14] CH1的P和N通道输出互换选择 [13] 当CH1_PS=1时,输出到CH1 P通道的值 [12] 当CH1_NS=1时,输出到CH1 N通道的值 [11] CH1 P来源 [10] CH1 N来源 [9] CH1 P极性选择 [8] CH1 N极性选择 [7] CH0工作模式选择 [6] CH0的P和N通道输出互换选择 [5] 当CH0_PS =1时,输出到CH0 P通道的值 [4] 当CH0_NS =1时,输出到CH0 N通道的值 [3] CH0 P来源 [2] CH0 N来源 [1] CH0 P极性选择 [0] CH0 N极性选择

在文件 [hal_lks32mc09x.reg.h](#) 第 223 行定义.

3.17.2.16 IO23

```
--IOM uint32_t IO23
```

通道2/3设置 [15] CH3工作模式选择 [14] CH3的P和N通道输出互换选择 [13] 当CH3_PS =1时,输出到CH3 P通道的值 [12] 当CH3_NS =1时,输出到CH3 N通道的值 [11] CH3 P来源 [10] CH3 N来源 [9] CH3 P极性选择 [8] CH3 N极性选择 [7] CH2工作模式选择 [6] CH2的P和N通道输出互换选择 [5] 当CH2_PS =1时,输出到CH2 P通道的值 [4] 当CH2_NS =1时,输出到CH2 N通道的值 [3] CH2 P来源 [2] CH2 N来源 [1] CH2 P极性选择 [0] CH2 N极性选择

在文件 [hal_lks32mc09x.reg.h](#) 第 224 行定义.

3.17.2.17 PP

```
--IOM uint32_t PP
```

推挽配置 [3] IO3推挽模式使能信号 [2] IO2推挽模式使能信号 [1] IO1推挽模式使能信号 [0] IO0推挽模式使能信号

在文件 [hal_lks32mc09x.reg.h](#) 第 222 行定义.

3.17.2.18 PRT

```
--OM uint32_t PRT
```

MCPWM寄存器保护 [15:0] 写入0xDEAD,解除MCPWM寄存器写保护; 写入其它值,MCPWM寄存器进入写保护

在文件 [hal_lks32mc09x_reg.h](#) 第 230 行定义.

3.17.2.19 RE

```
--IOM uint32_t RE
```

DMA请求使能 [13] 时基0 T1事件DMA请求1使能 [12] 时基0 T0事件DMA请求1使能 [11] MCPWM计数器命中TMR3,DMA请求1使能 [10] MCPWM计数器命中TMR2,DMA请求1使能 [9] MCPWM计数器命中TMR1,DMA请求1使能 [8] MCPWM计数器命中TMR0,DMA请求1使能 [5] 时基0 T1事件DMA请求0使能 [4] 时基0 T0事件DMA请求0使能 [3] MCPWM计数器命中TMR3,DMA请求0使能 [2] MCPWM计数器命中TMR2,DMA请求0使能 [1] MCPWM计数器命中TMR1,DMA请求0使能 [0] MCPWM计数器命中TMR0,DMA请求0使能

在文件 [hal_lks32mc09x_reg.h](#) 第 221 行定义.

3.17.2.20 RESERVED

```
--IM uint32_t RESERVED[4]
```

在文件 [hal_lks32mc09x_reg.h](#) 第 196 行定义.

3.17.2.21 RESERVED1

```
--IM uint32_t RESERVED1
```

在文件 [hal_lks32mc09x_reg.h](#) 第 202 行定义.

3.17.2.22 RESERVED2

```
--IM uint32_t RESERVED2
```

在文件 [hal_lks32mc09x_reg.h](#) 第 204 行定义.

3.17.2.23 RESERVED3

```
--IM uint32_t RESERVED3
```

在文件 [hal_lks32mc09x_reg.h](#) 第 208 行定义.

3.17.2.24 RESERVED4

```
--IM uint32_t RESERVED4[2]
```

在文件 [hal_lks32mc09x_reg.h](#) 第 211 行定义.

3.17.2.25 RESERVED5

```
--IM uint32_t RESERVED5[2]
```

在文件 [hal_lks32mc09x_reg.h](#) 第 218 行定义.

3.17.2.26 RESERVED6

```
--IM uint32_t RESERVED6[2]
```

在文件 [hal_lks32mc09x_reg.h](#) 第 225 行定义.

3.17.2.27 RESERVED7

```
--IM uint32_t RESERVED7
```

在文件 [hal_lks32mc09x_reg.h](#) 第 227 行定义.

3.17.2.28 RESERVED8

```
--IM uint32_t RESERVED8[3]
```

在文件 [hal_lks32mc09x_reg.h](#) 第 231 行定义.

3.17.2.29 SDCFG

```
--IOM uint32_t SDCFG
```

影子寄存器重载配置 [6] 更新事件是否自动清除MCPWM0_EIF[5:4]并置位MCPWM0_CH012_FAIL.MOE [5] 时基0 t1(过零)事件更新使能 [4] 时基0 t0(起点)事件更新使能 [3:0] 时基0更新间隔

在文件 [hal_lks32mc09x_reg.h](#) 第 213 行定义.

3.17.2.30 SW_FAIL

```
--IOM uint32_t SW_FAIL
```

软件故障测试寄存器 [5] FAIL1软件测试信号使能 [4] FAIL0软件测试信号使能 [1] FAIL1软件测试信号 [0] FAIL0软件测试信号

在文件 [hal_lks32mc09x_reg.h](#) 第 232 行定义.

3.17.2.31 TCLK

```
--IOM uint32_t TCLK
```

MCPWM时钟配置寄存器 [14:12] MCPWM工作时钟分频寄存器 [8] 时基0外部触发使能 [6] MCPWM时基0计数器使能开关 [2] MCPWM工作时钟使能

在文件 [hal_lks32mc09x_reg.h](#) 第 215 行定义.

3.17.2.32 TH

```
--IOM uint32_t TH[8]
```

< MCPWM0结构体

MCPWM 比较门限值

在文件 [hal_lks32mc09x_reg.h](#) 第 195 行定义.

3.17.2.33 TH0

```
--IOM uint32_t TH0
```

基础计数器0 15位无符号阈值 [14:0]MCPWM时基0计数器门限值

在文件 [hal_lks32mc09x_reg.h](#) 第 201 行定义.

3.17.2.34 TMR0

```
--IOM uint32_t TMR0
```

ADC触发时间0 [15:0]ADC采样定时器比较门限0寄存器

在文件 [hal_lks32mc09x_reg.h](#) 第 197 行定义.

3.17.2.35 TMR1

```
--IOM uint32_t TMR1
```

ADC触发时间1 [15:0]ADC采样定时器比较门限1寄存器

在文件 [hal_lks32mc09x_reg.h](#) 第 198 行定义.

3.17.2.36 TMR2

```
--IOM uint32_t TMR2
```

ADC触发时间2 [15:0]ADC采样定时器比较门限2寄存器

在文件 [hal_lks32mc09x_reg.h](#) 第 199 行定义.

3.17.2.37 TMR3

```
--IOM uint32_t TMR3
```

ADC触发时间3 [15:0]ADC采样定时器比较门限3寄存器

在文件 [hal_lks32mc09x_reg.h](#) 第 200 行定义。

3.17.2.38 UPDATE

```
--OM uint32_t UPDATE
```

掩码寄存器手动重载配置 [28] 写1产生软件(手动)触发,将MCPWM0.IO3/4/5由预装载寄存器加载到MCPWM运行所使用的影子寄存器 [25] 写1产生软件(手动)触发,将MCPWM0.TMR3由预装载寄存器加载到MCPWM运行所使用的影子寄存器 [24] 写1产生软件(手动)触发,将MCPWM0.TMR2由预装载寄存器加载到MCPWM运行所使用的影子寄存器 [17] 写1产生软件(手动)触发,将MCPWM0.TH31由预装载寄存器加载到MCPWM运行所使用的影子寄存器 [16] 写1产生软件(手动)触发,将MCPWM0.TH30由预装载寄存器加载到MCPWM运行所使用的影子寄存器 [12] 写1产生软件(手动)触发,将MCPWM0.IO0/1/2由预装载寄存器加载到MCPWM运行所使用的影子寄存器 [11] 写1产生软件(手动)触发,将MCPWM0.CNT0由预装载寄存器加载到MCPWM运行所使用的影子寄存器 [10] 写1产生软件(手动)触发,将MCPWM0.TH0由预装载寄存器加载到MCPWM运行所使用的影子寄存器 [9] 写1产生软件(手动)触发,将MCPWM0.TMR1由预装载寄存器加载到MCPWM运行所使用的影子寄存器 [8] 写1产生软件(手动)触发,将MCPWM0.TMR0由预装载寄存器加载到MCPWM运行所使用的影子寄存器 [5] 写1产生软件(手动)触发,将MCPWM0.TH21由预装载寄存器加载到MCPWM运行所使用的影子寄存器 [4] 写1产生软件(手动)触发,将MCPWM0.TH20由预装载寄存器加载到MCPWM运行所使用的影子寄存器 [3] 写1产生软件(手动)触发,将MCPWM0.TH11由预装载寄存器加载到MCPWM运行所使用的影子寄存器 [2] 写1产生软件(手动)触发,将MCPWM0.TH10由预装载寄存器加载到MCPWM运行所使用的影子寄存器 [1] 写1产生软件(手动)触发,将MCPWM0.TH01由预装载寄存器加载到MCPWM运行所使用的影子寄存器 [0] 写1产生软件(手动)触发,将MCPWM0.TH00由预装载寄存器加载到MCPWM运行所使用的影子寄存器

在文件 [hal_lks32mc09x_reg.h](#) 第 205 行定义。

3.18 HAL_QEP_TypeDef结构体 参考

QEP0寄存器 (QEP0)

```
#include <hal_lks32mc09x_reg.h>
```

成员变量

- `--IOM uint32_t CFG`
 < QEP0结构体
- `--IOM uint32_t TH`
 编码器计数阈值 [15:0] 计数门限 TH。编码器将从0计数到TH,然后重新开始,或者从TH向下计数到0然后重新开始
- `--IOM uint32_t CNT`
 编码器当前计数值 [15:0] QEP0计数值
- `--IOM uint32_t IE`
 QEP中断使能 [1] 上溢出中断使能 [0] 下溢出中断使能
- `--IOM uint32_t IF`
 QEP中断标志 [1] 上溢出中断标志 [0] 下溢出中断标志

3.18.1 详细描述

QEP0寄存器 (QEP0)

在文件 [hal_lks32mc09x_reg.h](#) 第 237 行定义.

3.18.2 结构体成员变量说明

3.18.2.1 CFG

```
--IOM uint32_t CFG
```

< QEP0结构体

QEP配置 [23:16] 编码器的滤波系数 [15] QEP模块整体使能 [10] CCW+SIGN/CCW+CW两种模式下,是否在下降沿进行计数(上升沿总是计数) [9:8] 编码器模式选择 [3] Z信号清零极性选择:低电平/下降沿清零使能 [2] Z信号清零极性选择:高电平/上升沿清零使能 [1] Z信号电平清零QEP计数器使能 [0] Z信号边沿清零QEP计数器使能

在文件 [hal_lks32mc09x_reg.h](#) 第 239 行定义.

3.18.2.2 CNT

```
--IOM uint32_t CNT
```

编码器当前计数值 [15:0] QEP0计数值

在文件 [hal_lks32mc09x_reg.h](#) 第 241 行定义.

3.18.2.3 IE

```
--IOM uint32_t IE
```

QEP中断使能 [1] 上溢出中断使能 [0] 下溢出中断使能

在文件 [hal_lks32mc09x_reg.h](#) 第 242 行定义.

3.18.2.4 IF

```
--IOM uint32_t IF
```

QEP中断标志 [1] 上溢出中断标志 [0] 下溢出中断标志

在文件 [hal_lks32mc09x_reg.h](#) 第 243 行定义.

3.18.2.5 TH

```
--IOM uint32_t TH
```

编码器计数阈值 [15:0] 计数门限TH。编码器将从0计数到TH,然后重新开始,或者从TH向下计数到0然后重新开始

在文件 [hal_lks32mc09x_reg.h](#) 第 240 行定义。

3.19 HAL_SIF_TypeDef结构体 参考

SIF0寄存器 (SIF0)

```
#include <hal_lks32mc09x_reg.h>
```

成员变量

- `--IOM uint32_t CFG`
 < SIF0 结构体
- `--IOM uint32_t TOSC`
 SIF 时间基准寄存器 [11:0] SIF模块时基设置
- `--IOM uint32_t TSTH1`
 SIF 同步字符周期 [9:0] SIF模块同步信号周期
- `--IOM uint32_t TDTTH1`
 SIF 完成字符周期 [3:0] SIF模块结束信号周期
- `--IOM uint32_t IRQ`
 SIF 中断寄存器 [4] SIF模块中断标志位 [1] SIF模块DMA使能控制位 [0] SIF模块中断使能控制位
- `--IOM uint32_t WDATA`
 SIF 写数据寄存器,写入以开始传输 [7:0] SIF模块数据寄存器

3.19.1 详细描述

SIF0寄存器 (SIF0)

在文件 [hal_lks32mc09x_reg.h](#) 第 248 行定义。

3.19.2 结构体成员变量说明

3.19.2.1 CFG

```
--IOM uint32_t CFG
```

< SIF0 结构体

SIF 配置寄存器 [7] SIF模块数据传输,完成信号默认电平 [6] SIF模块数据传输,同步信号默认电平 [5] SIF模块数据传输,同步信号是否产生脉冲信号 [4] SIF模块数据传输,结束信号控制位 [3] SIF模块数据传输,同步信号控制位 [2] SIF模块数据占空比控制位 [1] SIF模块数据大小头控制位 [0] SIF模块使能位

在文件 [hal_lks32mc09x_reg.h](#) 第 250 行定义。

3.19.2.2 IRQ

```
--IOM uint32_t IRQ
```

SIF 中断寄存器 [4] SIF模块中断标志位 [1] SIF模块DMA使能控制位 [0] SIF模块中断使能控制位

在文件 [hal_lks32mc09x_reg.h](#) 第 254 行定义.

3.19.2.3 TDTH1

```
--IOM uint32_t TDTH1
```

SIF 完成字符周期 [3:0] SIF模块结束信号周期

在文件 [hal_lks32mc09x_reg.h](#) 第 253 行定义.

3.19.2.4 TOSC

```
--IOM uint32_t TOSC
```

SIF 时间基准寄存器 [11:0] SIF模块时基设置

在文件 [hal_lks32mc09x_reg.h](#) 第 251 行定义.

3.19.2.5 TSTH1

```
--IOM uint32_t TSTH1
```

SIF 同步字符周期 [9:0] SIF模块同步信号周期

在文件 [hal_lks32mc09x_reg.h](#) 第 252 行定义.

3.19.2.6 WDATA

```
--IOM uint32_t WDATA
```

SIF 写数据寄存器,写入以开始传输 [7:0] SIF模块数据寄存器

在文件 [hal_lks32mc09x_reg.h](#) 第 255 行定义.

3.20 HAL_SPI_TypeDef结构体 参考

SPI0寄存器 (SPI0)

```
#include <hal_lks32mc09x_reg.h>
```

成员变量

- `__IOM uint32_t CFG`
< SPI0 结构体
- `__IOM uint32_t IE`
SPI 中断使能 [7] SPI中断使能开关 [6] SPI传输,完成事件中断使能信号 [5] SPI传输,异常事件中断使能信号 [4] SPI传输,溢出事件中断使能信号 [3] 传输触发选择 [2] SPI传输,完成事件 [1] SPI传输,异常事件 [0] SPI传输,溢出事件
- `__IOM uint32_t DIV`
SPI 时钟分频 [15] SPI数据搬移方式 [11:0] SPI传输波特率配置
- `__IOM uint32_t TX_DATA`
SPI 发送数据 [15:0] SPI数据发送寄存器
- `__IOM uint32_t RX_DATA`
SPI 接收数据 [15:0] SPI数据接收寄存器
- `__IOM uint32_t SIZE`
SPI 传输位数 [4:0] 字节长度寄存器

3.20.1 详细描述

SPI0寄存器 (SPI0)

在文件 [hal_lks32mc09x_reg.h](#) 第 260 行定义.

3.20.2 结构体成员变量说明

3.20.2.1 CFG

`__IOM uint32_t CFG`

< SPI0 结构体

SPI 控制寄存器 [7:6] 半双工模式设置 [5] SPI从设备下,片选信号来源 [4] SPI主从模式选择 [3] SPI相位选择 [2] SPI极性选择 [1] SPI模块传输顺序 [0] SPI模块使能信号

在文件 [hal_lks32mc09x_reg.h](#) 第 262 行定义.

3.20.2.2 DIV

`__IOM uint32_t DIV`

SPI 时钟分频 [15] SPI数据搬移方式 [11:0] SPI传输波特率配置

在文件 [hal_lks32mc09x_reg.h](#) 第 264 行定义.

3.20.2.3 IE

`__IOM uint32_t IE`

SPI 中断使能 [7] SPI中断使能开关 [6] SPI传输,完成事件中断使能信号 [5] SPI传输,异常事件中断使能信号 [4] SPI传输,溢出事件中断使能信号 [3] 传输触发选择 [2] SPI传输,完成事件 [1] SPI传输,异常事件 [0] SPI传输,溢出事件

在文件 [hal_lks32mc09x_reg.h](#) 第 263 行定义.

3.20.2.4 RX_DATA

```
--IOM uint32_t RX_DATA
```

SPI 接收数据 [15:0] SPI数据接收寄存器

在文件 [hal_lks32mc09x_reg.h](#) 第 266 行定义.

3.20.2.5 SIZE

```
--IOM uint32_t SIZE
```

SPI 传输位数 [4:0] 字节长度寄存器

在文件 [hal_lks32mc09x_reg.h](#) 第 267 行定义.

3.20.2.6 TX_DATA

```
--IOM uint32_t TX_DATA
```

SPI 发送数据 [15:0] SPI数据发送寄存器

在文件 [hal_lks32mc09x_reg.h](#) 第 265 行定义.

3.21 HAL_SYS_TypeDef结构体 参考

SYS寄存器 (SYS)

```
#include <hal_lks32mc09x_reg.h>
```

成员变量

- `_IM uint32_t AFE_ADC`
 < SYS 结构体
- `_IM uint32_t AFE_INFO`
 芯片版本信息 [15:8] 芯片型号信息 [7:0] 芯片版本信息
- `_IM uint32_t AFE_DBG`
 AFE调试寄存器 [15] 供电低于掉电监测阈值 [14:0] 保留位
- `_IOM uint32_t OPA_SEL`
 OPA选择 [15] ADC和运放模式 [14:13] 掉电检测阈值 V_{th} 选择 [12] 掉电检测基准源选择 [11] 掉电检测电路开关 [10] OPA 内部IP/IN短接以测试零漂 [9:8] OPA偏置电流调节 [7:6] 运放3反馈电阻 [5:4] 运放2反馈电阻 [3:2] 运放1反馈电阻 [1:0] 运放0反馈电阻
- `_IOM uint32_t AFE_REG0`
 模拟寄存器0 [15] ADC和运放模式 [14:13] 掉电检测阈值 V_{th} 选择 [12] 掉电检测基准源选择 [11] 掉电检测电路开关 [10] OPA 内部IP/IN短接以测试零漂 [9:8] OPA偏置电流调节 [7:6] 运放3反馈电阻 [5:4] 运放2反馈电阻 [3:2] 运放1反馈电阻 [1:0] 运放0反馈电阻
- `_IOM uint32_t AFE_REG1`

- 模拟寄存器1 [15] OPA3开关 [14] OPA2开关 [13] OPA1开关 [12] OPA0开关 [11] BandGap开关 [10] 8MHz RCH时钟开关 [9] 晶振低通滤波关闭 [8] 0:不输出 1:使能 1.5V LDO输出到IO P0.0 [7] 8BIT DAC量程选择 [6] 12BIT DAC量程选择 [5] 保留位,必须为0 [4] 晶振选择反相器或者施密特 [3:1] SW电压测量时片内电阻阻值选择 [0] 1: 比较器延迟为:15ns(pos),32ns(neg) 0:比较器延迟为:92ns(pos),320ns(neg)
- `_IOM uint32_t AFE_REG2`
模拟寄存器2 [15] 使能测量反电动势 [14:13] 测量反电动势的零温度系数电阻连接关系 [12] 保留位 [11:10] 使能比较器三个正输入端中点信号到输出 [9] 保留位 [8] 晶体起振电路电阻调节 [7] 使能DAC1输出到IO [6] 使能DAC0输出到IO [5] 使能VREF输出到IO [4] 使能VBGP输出到IO [3:0] 使能_OPAx输出信号送至IO口,或使能OPAx.OUT输出信号送至CMP
 - `_IOM uint32_t AFE_REG3`
模拟寄存器3 [15:12] 比较器1正端信号选择 [11:8] 比较器0正端信号选择 [7] 比较器回差选择 [6:4] 比较器1信号负端选择 [3] 保留位,必须为0 [2:0] 比较器0信号负端选择
 - `_IOM uint32_t AFE_REG4`
模拟寄存器4 [12] ADC测量运放通道半波使能 [4] 使用外部输入电源作为ADC REF
 - `_IOM uint32_t AFE_REG5`
模拟寄存器5 [15:10] 保留位,必须为0 [9:8] 电源复位阈值Vth选择 [7] 比较器1开关 [6] 比较器0开关 [5] PLL开关 [4] 晶体起振电路开关 [3] 温度传感器开关 [2] 8BIT DAC开关 [1] 12BIT DAC开关 [0] 保留位,必须为0
 - `_IOM uint32_t RESERVED`
 - `_IOM uint32_t AFE_REG7`
模拟寄存器7 [15:10] 保留位,必须为0 [9] ADC使能位 [8] ADC采样保持电路使能位 [7] 保留位,必须为0 [6:4] PLL输出频率选择 [3:0] 保留位,必须为0
 - `_IOM uint32_t RESERVED1` [11]
 - `_IOM uint32_t AFE_DAC_CTRL`
DAC控制寄存器 [6:4] SYS_AFE_DAC0步进值 [3] TIMER3过零事件触发DAC0步进使能 [2] TIMER2过零事件触发DAC0步进使能 [1] TIMER1过零事件触发DAC0步进使能 [0] TIMER0过零事件触发DAC0步进使能
 - `_IOM uint32_t AFE_DAC0`
DAC0数字输入 [11:0] DAC0待转换的数字量输入
 - `_IOM uint32_t AFE_DAC1`
DAC1数字量寄存器 [7:0] DAC1待转换的数字量输入
 - `_IOM uint32_t AFE_DAC0_AMC`
DAC0增益校正,10bit无符号定点数,B[9]为整数,B[8:0]为小数
 - `_IOM uint32_t AFE_DAC0_DC`
DAC0直流偏置,10bit有符号数,B[9]为符号位
 - `_IOM uint32_t AFE_DAC1_AMC`
DAC1增益校正,10bit无符号定点数,B[9]为整数,B[8:0]为小数
 - `_IOM uint32_t AFE_DAC1_DC`
DAC1直流偏置,10bit有符号数,B[9]为符号位
 - `_IOM uint32_t RESERVED2` [2]
 - `_IOM uint32_t CLK_CFG`
时钟配置 [14] PLLPDN设置为1后,经过一段时间,PLL_LOCK为1 [13] 晶振时钟在被开启后是否停振过 [12] 晶振时钟当前是否已经停振 [9:8] 系统时钟MCLK的来源选择信号 [7:0] PLL输出分频控制
 - `_IOM uint32_t IO_CFG`
IO配置 [6] SWD复用控制信号 [5] RSTn/P0.2复用控制信号
 - `_IOM uint32_t DBG_CFG`
调试配置 [31:16] 向此位段写入0x5AA5 触发软件中断 [15] 软件中断标志 [14] Debug软复位是否复位除Flash/SYS_AFE以外的外设寄存器 [11] 调试模式下CPU halt状态TIMER3停止 [10] 调试模式下CPU halt状态TIMER2停止 [9] 调试模式下CPU halt状态TIMER1停止 [8] 调试模式下CPU halt状态TIMER0停止 [5] 调试模式下CPU halt状态独立看门狗停止 [1] 调试STOP(DEEPSLEEP)模式 [0] 调试睡眠(SLEEP)模式
 - `_IOM uint32_t RESERVED3`
 - `_IOM uint32_t CLK_DIV0`
I2C时钟分频 [15:0] I2C工作时钟=MCLK/(CLK_DIV0+1)
 - `_IOM uint32_t CLK_DIV1`
 - `_IOM uint32_t CLK_DIV2`
UART时钟分频 [15:0] UART工作时钟=MCLK/(CLK_DIV2+1)

- `__IOM uint32_t CLK_FEN`

外设时钟使能 [24:22] 保留 [21] `CL0` 模块时钟控制信号 [20] `SIF0` 模块时钟控制信号 [19] `DSP0` 模块时钟控制信号 [16] `CRC0` 模块时钟控制信号 [15] `UART2` 模块时钟控制信号 [13] `UART1` 模块时钟控制信号 [12] `UART0` 模块时钟控制信号 [11] `GPIO` 模块时钟控制信号 [10] `MCPWM0` 模块时钟控制信号 [9] `QEP0` 模块时钟控制信号 [7] `TIMER2` 模块时钟控制信号 [5] `TIMER1` 模块时钟控制信号 [4] `TIMER0` 模块时钟控制信号 [3] `HALL0` 模块时钟控制信号 [2] `CMP` 模块时钟控制信号 [1] `I2C0` 模块时钟控制信号 [0] `SPI0` 模块时钟控制信号

- `__IM uint32_t RESERVED41`

- `__IOM uint32_t SFT_RST`

外设软复位 [26] `EEPROM` 模块软复位信号 [25:24] 保留 [23] `ADC1` 数字接口模块软复位信号 [22] `ADC0` 数字接口模块软复位信号 [21] `CL0` 模块软复位信号 [20] `SIF0` 模块软复位信号 [19] `DMA0` 模块软复位信号 [17] `DSP0` 模块软复位信号 [16] `CRC0` 数字接口模块软复位信号 [15] `UART1` 模块软复位信号 [13] `UART1` 模块软复位信号 [12] `UART0` 模块软复位信号 [11] `GPIO` 模块软复位信号 [10] `MCPWM0` 模块软复位信号 [9] `QEP1` 模块软复位信号 [8] `QEP0` 模块软复位信号 [7] `TIMER3` 模块软复位信号 [6] `TIMER2` 模块软复位信号 [5] `TIMER1` 模块软复位信号 [4] `TIMER0` 模块软复位信号 [3] `HALL0` 模块软复位信号 [2] `CMP` 模块软复位信号 [1] `I2C0` 模块软复位信号 [0] `SPI0` 模块软复位信号

- `__OM uint32_t PROTECT`

系统寄存器保护 [15:0] 写入 `0x7a83` 解除保护,其他值加上保护,除 `SYS_AFE_DAC0`、`SYS_AFE_DAC0_AMC`、`SYS_AFE_DAC0_DC`、`SYS_AFE_DAC1`、`SYS_AFE_DAC1_AMC`、`SYS_AFE_DAC1_DC` 外,其他系统寄存器(`SYS_`开头的寄存器,包括时钟复位管理以及模拟寄存器)受写保护

- `__IM uint32_t RESERVED5`

- `__IOM uint32_t NMI_TH`

`NMI` 间隔阈值 [1] `NMI` 定时中断标志 [0] `NMI` 间隔阈值

- `__IOM uint32_t IF`

系统中断 [1] 电源过低中断标志 [0] `NMI` 定时中断标志

- `__IOM uint32_t RAM_LOCK`

`RAM` 写保护寄存器 [31:8] 只有在 [31:8] 写入 `0x5a6b7c` 的同时,才能改写 [7:0] [7] `0x1C00~0x1FFF` `RAM` 空间写入锁定 [6] `0x1800~0x1BFF` `RAM` 空间写入锁定 [5] `0x1400~0x17FF` `RAM` 空间写入锁定 [4] `0x1000~0x13FF` `RAM` 空间写入锁定 [3] `0x0C00~0x0FFF` `RAM` 空间写入锁定 [2] `0x0800~0x0BFF` `RAM` 空间写入锁定 [1] `0x0400~0x07FF` `RAM` 空间写入锁定 [0] `0x0000~0x03FF` `RAM` 空间写入锁定

- `__IM uint32_t RESERVED6`

- `__IM uint32_t RESERVED7` [4]

- `__IOM uint32_t FLSE`

- `__IOM uint32_t FLSP`

3.21.1 详细描述

`SYS` 寄存器 (`SYS`)

在文件 `hal_lks32mc09x_reg.h` 第 272 行定义.

3.21.2 结构体成员变量说明

3.21.2.1 AFE_ADC

`__IM uint32_t AFE_ADC`

< `SYS` 结构体

`ADC` 原始数据 [11:0] `ADC` 输出的原始数据

在文件 `hal_lks32mc09x_reg.h` 第 274 行定义.

3.21.2.2 AFE_DAC0

```
--IOM uint32_t AFE_DAC0
```

DAC0数字输入 [11:0] DAC0待转换的数字量输入

在文件 [hal_lks32mc09x_reg.h](#) 第 288 行定义.

3.21.2.3 AFE_DAC0_AMC

```
--IOM uint32_t AFE_DAC0_AMC
```

DAC0增益校正,10bit无符号定点数,B[9]为整数,B[8:0]为小数

在文件 [hal_lks32mc09x_reg.h](#) 第 290 行定义.

3.21.2.4 AFE_DAC0_DC

```
--IOM uint32_t AFE_DAC0_DC
```

DAC0直流偏置,10bit有符号数,B[9]为符号位

在文件 [hal_lks32mc09x_reg.h](#) 第 291 行定义.

3.21.2.5 AFE_DAC1

```
--IOM uint32_t AFE_DAC1
```

DAC1数字量寄存器 [7:0] DAC1待转换的数字量输入

在文件 [hal_lks32mc09x_reg.h](#) 第 289 行定义.

3.21.2.6 AFE_DAC1_AMC

```
--IOM uint32_t AFE_DAC1_AMC
```

DAC1增益校正,10bit无符号定点数,B[9]为整数,B[8:0]为小数

在文件 [hal_lks32mc09x_reg.h](#) 第 292 行定义.

3.21.2.7 AFE_DAC1_DC

```
--IOM uint32_t AFE_DAC1_DC
```

DAC1直流偏置,10bit 有符号数,B[9]为符号位

在文件 [hal_lks32mc09x_reg.h](#) 第 293 行定义.

3.21.2.8 AFE_DAC_CTRL

```
--IOM uint32_t AFE_DAC_CTRL
```

DAC控制寄存器 [6:4] SYS_AFE_DAC0步进值 [3] TIMER3过零事件触发DAC0步进使能 [2] TIMER2过零事件触发DAC0步进使能 [1] TIMER1过零事件触发DAC0步进使能 [0] TIMER0过零事件触发DAC0步进使能

在文件 [hal_lks32mc09x.reg.h](#) 第 287 行定义.

3.21.2.9 AFE_DBG

```
--IM uint32_t AFE_DBG
```

AFE调试寄存器 [15] 供电低于掉电监测阈值 [14:0] 保留位

在文件 [hal_lks32mc09x.reg.h](#) 第 276 行定义.

3.21.2.10 AFE_INFO

```
--IM uint32_t AFE_INFO
```

芯片版本信息 [15:8] 芯片型号信息 [7:0] 芯片版本信息

在文件 [hal_lks32mc09x.reg.h](#) 第 275 行定义.

3.21.2.11 AFE_REG0

```
--IOM uint32_t AFE_REG0
```

模拟寄存器0 [15] ADC和运放模式 [14:13] 掉电检测阈值Vth选择 [12] 掉电检测基准源选择 [11] 掉电检测电路开关 [10] OPA 内部IP/IN短接以测试零漂 [9:8] OPA偏置电流调节 [7:6] 运放3反馈电阻 [5:4] 运放2反馈电阻 [3:2] 运放1反馈电阻 [1:0] 运放0反馈电阻

在文件 [hal_lks32mc09x.reg.h](#) 第 278 行定义.

3.21.2.12 AFE_REG1

```
--IOM uint32_t AFE_REG1
```

模拟寄存器1 [15] OPA3开关 [14] OPA2开关 [13] OPA1开关 [12] OPA0开关 [11] BandGap开关 [10] 8MHz RCH时钟开关 [9] 晶振低通滤波关闭 [8] 0:不输出1:使能1.5V LDO输出到IO P0_0 [7] 8BIT DAC量程选择 [6] 12BIT DAC量程选择 [5] 保留位,必须为0 [4] 晶振选择反相器或者施密特 [3:1] SW电压测量时片内电阻阻值选择 [0] 1: 比较器延迟为:15ns(pos),32ns(neg)0:比较器延迟为:92ns(pos),320ns(neg)

在文件 [hal_lks32mc09x.reg.h](#) 第 279 行定义.

3.21.2.13 AFE_REG2

```
--IOM uint32_t AFE_REG2
```

模拟寄存器2 [15] 使能测量反电动势 [14:13] 测量反电动势的零温度系数电阻连接关系 [12] 保留位 [11:10] 使能比较器三个正输入端中点信号到输出 [9] 保留位 [8] 晶体起振电路电阻调节 [7] 使能DAC1输出到IO [6] 使能DAC0输出到IO [5] 使能VREF输出到IO [4] 使能VBGP输出到IO [3:0] 使能.OPAx输出信号送至IO口,或使能OPAx_OUT输出信号送至CMP

在文件 [hal_lks32mc09x_reg.h](#) 第 280 行定义.

3.21.2.14 AFE_REG3

```
--IOM uint32_t AFE_REG3
```

模拟寄存器3 [15:12] 比较器1正端信号选择 [11:8] 比较器0正端信号选择 [7] 比较器回差选择 [6:4] 比较器1信号负端选择 [3] 保留位,必须为0 [2:0] 比较器0信号负端选择

在文件 [hal_lks32mc09x_reg.h](#) 第 281 行定义.

3.21.2.15 AFE_REG4

```
--IOM uint32_t AFE_REG4
```

模拟寄存器4 [12] ADC测量运放通道半波使能 [4] 使用外部输入电源作为ADC REF

在文件 [hal_lks32mc09x_reg.h](#) 第 282 行定义.

3.21.2.16 AFE_REG5

```
--IOM uint32_t AFE_REG5
```

模拟寄存器5 [15:10] 保留位,必须为0 [9:8] 电源复位阈值Vth选择 [7] 比较器1开关 [6] 比较器0开关 [5] PLL开关 [4] 晶体起振电路开关 [3] 温度传感器开关 [2] 8BIT DAC开关 [1] 12BIT DAC开关 [0] 保留位,必须为0

在文件 [hal_lks32mc09x_reg.h](#) 第 283 行定义.

3.21.2.17 AFE_REG7

```
--IOM uint32_t AFE_REG7
```

模拟寄存器7 [15:10] 保留位,必须为0 [9] ADC使能位 [8] ADC采样保持电路使能位 [7] 保留位,必须为0 [6:4] PLL输出频率选择 [3:0] 保留位,必须为0

在文件 [hal_lks32mc09x_reg.h](#) 第 285 行定义.

3.21.2.18 CLK_CFG

```
--IOM uint32_t CLK_CFG
```

时钟配置 [14] PLLPDN设置为1后,经过一段时间,PLL_LOCK为1 [13] 晶振时钟在被开启后是否停振过 [12] 晶振时钟当前是否已经停振 [9:8] 系统时钟MCLK的来源选择信号 [7:0] PLL输出分频控制

在文件 [hal_lks32mc09x_reg.h](#) 第 295 行定义.

3.21.2.19 CLK_DIV0

```
--IOM uint32_t CLK_DIV0
```

I2C时钟分频 [15:0] I2C工作时钟= $MCLK/(CLK\_DIV0+1)$

在文件 [hal_lks32mc09x_reg.h](#) 第 299 行定义.

3.21.2.20 CLK_DIV1

```
--IOM uint32_t CLK_DIV1
```

在文件 [hal_lks32mc09x_reg.h](#) 第 300 行定义.

3.21.2.21 CLK_DIV2

```
--IOM uint32_t CLK_DIV2
```

UART时钟分频 [15:0] UART工作时钟= $MCLK/(CLK\_DIV2+1)$

在文件 [hal_lks32mc09x_reg.h](#) 第 301 行定义.

3.21.2.22 CLK_FEN

```
--IOM uint32_t CLK_FEN
```

外设时钟使能 [24:22] 保留 [21] CL0模块时钟控制信号 [20] SIF0模块时钟控制信号 [19] DSP0模块时钟控制信号 [16] CRC0模块时钟控制信号 [15] UART2模块时钟控制信号 [13] UART1模块时钟控制信号 [12] UART0模块时钟控制信号 [11] GPIO模块时钟控制信号 [10] MCPWM0模块时钟控制信号 [9] QEP0模块时钟控制信号 [7] TIMER2模块时钟控制信号 [5] TIMER1模块时钟控制信号 [4] TIMER0模块时钟控制信号 [3] HALLO模块时钟控制信号 [2] CMP模块时钟控制信号 [1] I2C0模块时钟控制信号 [0] SPI0模块时钟控制信号

在文件 [hal_lks32mc09x_reg.h](#) 第 302 行定义.

3.21.2.23 DBG_CFG

```
--IOM uint32_t DBG_CFG
```

调试配置 [31:16] 向此位段写入0x5AA5 触发软件中断 [15] 软件中断标志 [14] Debug软复位是否复位除Flash/SYS_AFE以外的外设寄存器 [11] 调试模式下CPU halt状态TIMER3停止 [10] 调试模式下CPU halt状态TIMER2停止 [9] 调试模式下CPU halt状态TIMER1停止 [8] 调试模式下CPU halt状态TIMER0停止 [5] 调试模式下CPU halt状态独立看门狗停止 [1] 调试STOP(DEEPSLEEP)模式 [0] 调试睡眠(SLEEP)模式

在文件 [hal_lks32mc09x_reg.h](#) 第 297 行定义.

3.21.2.24 FLSE

```
--IOM uint32_t FLSE
```

在文件 [hal_lks32mc09x_reg.h](#) 第 312 行定义.

3.21.2.25 FLSP

```
--IOM uint32_t FLSP
```

在文件 [hal_lks32mc09x_reg.h](#) 第 313 行定义.

3.21.2.26 IF

```
--IOM uint32_t IF
```

系统中断 [1] 电源过低中断标志 [0] NMI定时中断标志

在文件 [hal_lks32mc09x_reg.h](#) 第 308 行定义.

3.21.2.27 IO_CFG

```
--IOM uint32_t IO_CFG
```

IO配置 [6] SWD复用控制信号 [5] RSTn/P0.2复用控制信号

在文件 [hal_lks32mc09x_reg.h](#) 第 296 行定义.

3.21.2.28 NMI_TH

```
--IOM uint32_t NMI_TH
```

NMI间隔阈值 [1] NMI定时中断标志 [0] NMI间隔阈值

在文件 [hal_lks32mc09x_reg.h](#) 第 307 行定义.

3.21.2.29 OPA_SEL

```
--IOM uint32_t OPA_SEL
```

OPA选择 [15] ADC和运放模式 [14:13] 掉电检测阈值Vth选择 [12] 掉电检测基准源选择 [11] 掉电检测电路开关 [10] OPA 内部IP/IN短接以测试零漂 [9:8] OPA偏置电流调节 [7:6] 运放3反馈电阻 [5:4] 运放2反馈电阻 [3:2] 运放1反馈电阻 [1:0] 运放0反馈电阻

在文件 [hal_lks32mc09x_reg.h](#) 第 277 行定义.

3.21.2.30 PROTECT

```
--OM uint32_t PROTECT
```

系统寄存器保护 [15:0] 写入0x7a83解除保护,其他值加上保护,除SYS_AFE_DAC0、SYS_AFE_DAC0_AMC、SYS_AFE_DAC0_DC、SYS_AFE_DAC1、SYS_AFE_DAC1_AMC、SYS_AFE_DAC1_DC外,其他系统寄存器(SYS开头的寄存器,包括时钟复位管理以及模拟寄存器)受写保护

在文件 [hal_lks32mc09x_reg.h](#) 第 305 行定义.

3.21.2.31 RAM_LOCK

```
--IOM uint32_t RAM_LOCK
```

RAM写保护寄存器 [31:8] 只有在[31:8]写入0x5a6b7c的同时,才能改写[7:0] [7] 0x1C00~0x1FFF RAM空间写入锁定 [6] 0x1800~0x1BFF RAM空间写入锁定 [5] 0x1400~0x17FF RAM空间写入锁定 [4] 0x1000~0x13FF RAM空间写入锁定 [3] 0x0C00~0x0FFF RAM空间写入锁定 [2] 0x0800~0x0BFF RAM空间写入锁定 [1] 0x0400~0x07FF RAM空间写入锁定 [0] 0x0000~0x03FF RAM空间写入锁定

在文件 [hal_lks32mc09x_reg.h](#) 第 309 行定义.

3.21.2.32 RESERVED

```
--IM uint32_t RESERVED
```

在文件 [hal_lks32mc09x_reg.h](#) 第 284 行定义.

3.21.2.33 RESERVED1

```
--IM uint32_t RESERVED1[11]
```

在文件 [hal_lks32mc09x_reg.h](#) 第 286 行定义.

3.21.2.34 RESERVED2

```
--IM uint32_t RESERVED2[2]
```

在文件 [hal_lks32mc09x_reg.h](#) 第 294 行定义.

3.21.2.35 RESERVED3

```
--IM uint32_t RESERVED3
```

在文件 [hal_lks32mc09x_reg.h](#) 第 298 行定义.

3.21.2.36 RESERVED41

```
--IM uint32_t RESERVED41
```

在文件 [hal_lks32mc09x_reg.h](#) 第 303 行定义.

3.21.2.37 RESERVED5

```
--IM uint32_t RESERVED5
```

在文件 [hal_lks32mc09x_reg.h](#) 第 306 行定义.

3.21.2.38 RESERVED6

```
--IM uint32_t RESERVED6
```

在文件 [hal_lks32mc09x_reg.h](#) 第 310 行定义.

3.21.2.39 RESERVED7

```
--IM uint32_t RESERVED7[4]
```

在文件 [hal_lks32mc09x_reg.h](#) 第 311 行定义.

3.21.2.40 SFT_RST

```
--IOM uint32_t SFT_RST
```

外设软复位 [26] EEPROM模块软复位信号 [25:24] 保留 [23] ADC1数字接口模块软复位信号 [22] ADC0数字接口模块软复位信号 [21] CL0模块软复位信号 [20] SIF0模块软复位信号 [19] DMA0模块软复位信号 [17] DSP0模块软复位信号 [16] CRC0数字接口模块软复位信号 [15] UART1模块软复位信号 [13] UART1模块软复位信号 [12] UART0模块软复位信号 [11] GPIO模块软复位信号 [10] MCPWM0模块软复位信号 [9] QEP1模块软复位信号 [8] QEP0模块软复位信号 [7] TIMER3模块软复位信号 [6] TIMER2模块软复位信号 [5] TIMER1模块软复位信号 [4] TIMER0模块软复位信号 [3] HALL0模块软复位信号 [2] CMP模块软复位信号 [1] I2C0模块软复位信号 [0] SPI0模块软复位信号

在文件 [hal_lks32mc09x_reg.h](#) 第 304 行定义.

3.22 HAL_TIMER_TypeDef结构体 参考

TIMER0寄存器 (TIMER0)

```
#include <hal_lks32mc09x_reg.h>
```

成员变量

- `__IOM uint32_t CFG0`
`< TIMER0` 结构体, `timer0`和`timer1`是16位的, `timer2`是32位的
- `__IOM uint32_t CFG1`
 定时器通道1配置 [10:9] 用于选择递增计数时触发还是递减计数的时候触发 `adc`采样 [8] 当发生 `CAP1`捕获事件时,清零 `TIMERx.CNT` [7:4] `TIMER`捕获模式通道1信号来源 [3] `TIMER`通道1在比较模式下的输出极性控制 [2] `TIMER`通道1的工作模式选择 [1] `TIMER`通道1下降沿捕获事件使能 [0] `TIMER`通道1上升沿捕获事件使能
- `__IOM uint32_t CFG2`
 定时器配置2 [15] `TIMER`模块整体使能 [14] 单次触发模式下是否可以在一个周期内再次触发 [13] 当 `RL←EN=1`时,使用 `CMP1`作为 `TMIN` [11] 软件更新启用影子寄存器时,向 `UPDATE`写1将 `TH/CMP0/CMP1`的预装载值加载到影子寄存器 `STH/SCMP0/SCMP1`之中 [10] 0: 不启用影子寄存器,软件写入 `TH/CMP0/CMP1`时直接更新对应的影子寄存器 1: 启用影子寄存器,软件写入 `TH/CMP0/CMP1`时进仅仅更新预装载值,当 `TIMER`发生过零事件时才将预装载值写入影子寄存器 [9] 单次发送模式 [8] 中心计数模式使能 [7] 0: 0-> `TH`递增计数, 1: `TH`->递减计数 [6:4] `TIMERx.CNT`频率配置 [3] `TIMERx.CNT`外部启动使能 [2] `TIMER`暂停使能 [1] `TIMER`重装使能 [0] `TIMER`时钟源
- `__IOM uint32_t RESERVED`
- `__IOM uint32_t TH`
 定时器计数阈值 [15:0] `TIMER`计数器计数门限
- `__IOM uint32_t CNT`
 定时器当前计数值 [15:0] `TIMER 0`计数器当前计数值
- `__IOM uint32_t CHN0`
 定时器通道0 `CMP/CAP` 阈值 [15:0] `TIMER`通道0工作在比较模式时,当计数器计数值等于 `CMP0`时,发生比较事件
- `__IOM uint32_t CHN1`
 定时器通道1 `CMP/CAP` 阈值 [15:0] `TIMER`通道1工作在比较模式时,当计数器计数值等于 `CMP1`时,发生比较事件
- `__IOM uint32_t EVT`
 定时器外部触发事件 [7:4] 外部事件发生次数阈值 [3:0] `TIMER`外部事件选择寄存器
- `__IOM uint32_t FLT`
 定时器滤波控制寄存器 [7:0] 通道0/1信号滤波宽度选择
- `__IOM uint32_t IE`
 定时器中断使能 [11] `TIMER FAIL`事件 `DMA`请求使能 [10] `TIMER CNT`过0 `DMA`请求使能 [9] `TIMER`通道1比较/捕获 `DMA`请求使能 [8] `TIMER`通道0比较/捕获 `DMA`请求使能 [3] `TIMER FAIL`事件中断使能 [2] `TIMER CNT`过0中断使能 [1] `TIMER`通道1比较/捕获中断使能 [0] `TIMER`通道0比较/捕获中断使能
- `__IOM uint32_t IF`
 定时器中断标志 [3] `TIMER Fail`事件中断标志 [2] `TIMER CNT`过0中断标志 [1] `TIMER`通道1比较/捕获中断标志 [0] `TIMER`通道0比较/捕获中断标志
- `__IOM uint32_t IO`
 定时器 `IO`控制寄存器 [9] 发生 `Fail`事件 `MOE`清零后, `CH1`通道输出值 [8] 发生 `Fail`事件 `MOE`清零后, `CH0`通道输出值 [7] `MCU`进入 `HALT`状态, `TIMER`输出值选择 [6] `TIMER`通道0/1输出使能 [2] `FAIL`信号选择 [1] `FAIL`信号极性 [0] `FAIL`信号使能

3.22.1 详细描述

TIMER0寄存器 (TIMER0)

在文件 `hal_lks32mc09x.reg.h` 第 318 行定义.

3.22.2 结构体成员变量说明

3.22.2.1 CFG0

```
__IOM uint32_t CFG0
```

< TIMER0 结构体,timer0和timer1是16位的,timer2是32位的

定时器通道0配置 [10:9] 用于选择递增计数时触发还是递减计数的时候触发adc采样 [8] 当发生CAP捕获事件时,清零TIMERx_CNT [7:4] TIMER捕获模式通道0信号来源 [3] TIMER通道0在比较模式下的输出极性控制 [2] TIMER通道0的工作模式选择 [1] TIMER通道0下降沿捕获事件使能 [0] TIMER通道0上升沿捕获事件使能

在文件 [hal_lks32mc09x_reg.h](#) 第 320 行定义.

3.22.2.2 CFG1

```
__IOM uint32_t CFG1
```

定时器通道1配置 [10:9] 用于选择递增计数时触发还是递减计数的时候触发adc采样 [8] 当发生CAP1捕获事件时,清零TIMERx_CNT [7:4] TIMER捕获模式通道1信号来源 [3] TIMER通道1在比较模式下的输出极性控制 [2] TIMER通道1的工作模式选择 [1] TIMER通道1下降沿捕获事件使能 [0] TIMER通道1上升沿捕获事件使能

在文件 [hal_lks32mc09x_reg.h](#) 第 321 行定义.

3.22.2.3 CFG2

```
__IOM uint32_t CFG2
```

定时器配置2 [15] TIMER模块整体使能 [14] 单次触发模式下是否可以在一个周期内再次触发 [13] 当RL_EN=1时,使用CMP1作为TMIN [11] 软件更新启用影子寄存器时,向UPDATE写1将TH/CMP0/CMP1的预装载值加载到影子寄存器STH/SCMP0/SCMP1之中 [10] 0: 不启用影子寄存器,软件写入TH/CMP0/CMP1时直接更新对应的影子寄存器1: 启用影子寄存器,软件写入TH/CMP0/CMP1时进仅仅更新预装载值,当TIMER发生过零事件时才将预装载值写入影子寄存器 [9] 单次发送模式 [8] 中心计数模式使能 [7] 0: 0->TH递增计数,1:TH->递减计数 [6:4] TIMERx_CNT频率配置 [3] TIMERx_CNT外部启动使能 [2] TIMER暂停使能 [1] TIMER重装使能 [0] TIMER 时钟源

在文件 [hal_lks32mc09x_reg.h](#) 第 322 行定义.

3.22.2.4 CHN0

```
__IOM uint32_t CHN0
```

定时器通道0 CMP/CAP 阈值 [15:0] TIMER 通道0工作在比较模式时,当计数器计数值等于CMP0时,发生比较事件

在文件 [hal_lks32mc09x_reg.h](#) 第 326 行定义.

3.22.2.5 CHN1

```
--IOM uint32_t CHN1
```

定时器通道1 CMP/CAP 阈值 [15:0] TIMER 通道1工作在比较模式时,当计数器计数值等于CMP1时,发生比较事件

在文件 [hal_lks32mc09x_reg.h](#) 第 327 行定义.

3.22.2.6 CNT

```
--IOM uint32_t CNT
```

定时器当前计数值 [15:0] TIMER 0计数器当前计数值

在文件 [hal_lks32mc09x_reg.h](#) 第 325 行定义.

3.22.2.7 EVT

```
--IOM uint32_t EVT
```

定时器外部触发事件 [7:4] 外部事件发生次数阈值 [3:0] TIMER外部事件选择寄存器

在文件 [hal_lks32mc09x_reg.h](#) 第 328 行定义.

3.22.2.8 FLT

```
--IOM uint32_t FLT
```

定时器滤波控制寄存器 [7:0] 通道0/1信号滤波宽度选择

在文件 [hal_lks32mc09x_reg.h](#) 第 329 行定义.

3.22.2.9 IE

```
--IOM uint32_t IE
```

定时器中断使能 [11] TIMER FAIL事件 DMA请求使能 [10] TIMER CNT过0 DMA请求使能 [9] TIMER通道1比较/捕获 DMA请求使能 [8] TIMER通道0比较/捕获 DMA请求使能 [3] TIMER FAIL事件中中断使能 [2] TIMER CNT过0中断使能 [1] TIMER通道1比较/捕获中断使能 [0] TIMER通道0比较/捕获中断使能

在文件 [hal_lks32mc09x_reg.h](#) 第 330 行定义.

3.22.2.10 IF

```
--IOM uint32_t IF
```

定时器中断标志 [3] TIMER Fail事件中中断标志 [2] TIMER CNT过0中断标志 [1] TIMER通道1比较/捕获中断标志 [0] TIMER通道0比较/捕获中断标志

在文件 [hal_lks32mc09x_reg.h](#) 第 331 行定义.

3.22.2.11 IO

```
__IOM uint32_t IO
```

定时器IO控制寄存器 [9] 发生Fail事件MOE清零后,CH1通道输出值 [8] 发生Fail事件MOE清零后,CH0通道输出值 [7] MCU进入HALT状态,TIMER输出值选择 [6] TIMER通道0/1输出使能 [2] FAIL信号选择 [1] FAIL信号极性 [0] FAIL信号使能

在文件 [hal_lks32mc09x_reg.h](#) 第 332 行定义.

3.22.2.12 RESERVED

```
__IM uint32_t RESERVED
```

在文件 [hal_lks32mc09x_reg.h](#) 第 323 行定义.

3.22.2.13 TH

```
__IOM uint32_t TH
```

定时器计数阈值 [15:0] TIMER 计数器计数门限

在文件 [hal_lks32mc09x_reg.h](#) 第 324 行定义.

3.23 HAL_UART_TypeDef结构体 参考

UART寄存器

```
#include <hal_lks32mc09x_reg.h>
```

成员变量

- `__IOM uint32_t CTRL`
UART控制 [7] 双工模式 [6] LIN模式使能 [5] 使能Multi-drop [4] 数据校验开关 [3] 奇偶校验配置 [2] 数据发送顺序配置 [1] 停止位长度配置 [0] 数据长度配置
- `__IOM uint32_t DIVH`
UART波特率高字节 [7:0] 波特率设置高字节
- `__IOM uint32_t DIVL`
UART波特率低字节 [7:0] 波特率设置低字节
- `__IOM uint32_t BUFF`
UART读/写缓冲区 [8:0] 写:发送数据缓存; 读:接收数据寄存器
- `__IOM uint32_t ADR`
UART匹配地址 [7:0] 多机通讯时的从机地址
- `__IOM uint32_t STT`
UART状态 [3] 1:UART已检测到起始符并正处于接收状态0:UART接收端空闲 [2] Multi-drop模式下,地址匹配标志位 [1] 发送完成标志位 [0] 发送缓存状态位
- `__IOM uint32_t RE`
UART DMA请求使能 [2] 发送缓冲区空DMA请求开关 [1] 接收完成DMA请求开关 [0] 发送完成DMA请求开关
- `__IOM uint32_t IE`
UART中断使能 [9] 波特率自适应中断使能 [8] 空闲帧中断使能 [7] LIN break character检测中断使能 [6] 接收缓冲区溢出中断使能 [5] 发送缓冲区溢出中断使能 [4] 校验错误中断开关 [3] 停止位错误中断开关 [2] 发送缓冲区空中断开关 [1] 接收完成中断开关 [0] 发送完成中断开关
- `__IOM uint32_t IF`
UART中断标志 [9] 波特率自适应使能情况下,发生波特率更新事件 [8] 空闲帧中断标志 [7] LIN break character检测中断标志 [6] 接收缓冲区溢出中断标志 [5] 发送缓冲区溢出中断标志 [4] 校验错误中断标志 [3] 停止位错误中断标志 [2] 发送缓冲区空中断标志 [1] 接收完成中断标志 [0] 发送完成中断标志
- `__IOM uint32_t IOC`
UART IO控制 [7] LIN模式下,写1发送一次break character [6] LIN break character检测长度 [4] 波特率自适应使能 [1] TXD输出极性使能开关 [0] RXD输入极性使能开关

3.23.1 详细描述

UART寄存器

在文件 [hal_lks32mc09x_reg.h](#) 第 337 行定义.

3.23.2 结构体成员变量说明

3.23.2.1 ADR

```
--IOM uint32_t ADR
```

UART匹配地址 [7:0] 多机通讯时的从机地址

在文件 [hal_lks32mc09x_reg.h](#) 第 343 行定义.

3.23.2.2 BUFF

```
--IOM uint32_t BUFF
```

UART读/写缓冲区 [8:0] 写:发送数据缓存; 读:接收数据寄存器

在文件 [hal_lks32mc09x_reg.h](#) 第 342 行定义.

3.23.2.3 CTRL

```
--IOM uint32_t CTRL
```

UART控制 [7] 双工模式 [6] LIN模式使能 [5] 使能Multi-drop [4] 数据校验开关 [3] 奇偶校验配置 [2] 数据发送顺序配置 [1] 停止位长度配置 [0] 数据长度配置

在文件 [hal_lks32mc09x_reg.h](#) 第 339 行定义.

3.23.2.4 DIVH

```
--IOM uint32_t DIVH
```

UART波特率高字节 [7:0] 波特率设置高字节

在文件 [hal_lks32mc09x_reg.h](#) 第 340 行定义.

3.23.2.5 DIVL

```
--IOM uint32_t DIVL
```

UART波特率低字节 [7:0] 波特率设置低字节

在文件 [hal_lks32mc09x_reg.h](#) 第 341 行定义.

3.23.2.6 IE

--IOM uint32_t IE

UART中断使能 [9] 波特率自适应中断使能 [8] 空闲帧中断使能 [7] LIN break character检测中断使能 [6] 接收缓冲区溢出中断使能 [5] 发送缓冲区溢出中断使能 [4] 校验错误中断开关 [3] 停止位错误中断开关 [2] 发送缓冲区空中断开关 [1] 接收完成中断开关 [0] 发送完成中断开关

在文件 [hal_lks32mc09x_reg.h](#) 第 346 行定义.

3.23.2.7 IF

--IOM uint32_t IF

UART中断标志 [9] 波特率自适应使能情况下,发生波特率更新事件 [8] 空闲帧中断标志 [7] LIN break character检测中断标志 [6] 接收缓冲区溢出中断标志 [5] 发送缓冲区溢出中断标志 [4] 校验错误中断标志 [3] 停止位错误中断标志 [2] 发送缓冲区空中断标志 [1] 接收完成中断标志 [0] 发送完成中断标志

在文件 [hal_lks32mc09x_reg.h](#) 第 347 行定义.

3.23.2.8 IOC

--IOM uint32_t IOC

UART IO控制 [7] LIN模式下,写1发送一次break character [6] LIN break character检测长度 [4] 波特率自适应使能 [1] TXD输出极性使能开关 [0] RXD输入极性使能开关

在文件 [hal_lks32mc09x_reg.h](#) 第 348 行定义.

3.23.2.9 RE

--IOM uint32_t RE

UART DMA请求使能 [2] 发送缓冲区空DMA请求开关 [1] 接收完成DMA请求开关 [0] 发送完成DMA请求开关

在文件 [hal_lks32mc09x_reg.h](#) 第 345 行定义.

3.23.2.10 STT

--IOM uint32_t STT

UART状态 [3] 1:UART已检测到起始符并正处于接收状态0:UART接收端空闲 [2] Multi-drop模式下,地址匹配标志位 [1] 发送完成标志位 [0] 发送缓存状态位

在文件 [hal_lks32mc09x_reg.h](#) 第 344 行定义.

Chapter 4

文件说明

4.1 hal_lks32mc09x.c 文件参考

```
#include "source/hal_lks32mc09x_adc.c"
#include "source/hal_lks32mc09x_clk.c"
#include "source/hal_lks32mc09x_cmp.c"
#include "source/hal_lks32mc09x_crc.c"
#include "source/hal_lks32mc09x_dac.c"
#include "source/hal_lks32mc09x_dma.c"
#include "source/hal_lks32mc09x_dsp.c"
#include "source/hal_lks32mc09x_flash.c"
#include "source/hal_lks32mc09x_gpio.c"
#include "source/hal_lks32mc09x_hall.c"
#include "source/hal_lks32mc09x_i2c.c"
#include "source/hal_lks32mc09x_init.c"
#include "source/hal_lks32mc09x_iwdg.c"
#include "source/hal_lks32mc09x_irq.c"
#include "source/hal_lks32mc09x_mcpwm.c"
#include "source/hal_lks32mc09x_opa.c"
#include "source/hal_lks32mc09x_qep.c"
#include "source/hal_lks32mc09x_reset.c"
#include "source/hal_lks32mc09x_reg.c"
#include "source/hal_lks32mc09x_spi.c"
#include "source/hal_lks32mc09x_test.c"
#include "source/hal_lks32mc09x_timer.c"
#include "source/hal_lks32mc09x_tmp.c"
#include "source/hal_lks32mc09x_trim.c"
#include "source/hal_lks32mc09x_uart.c"
#include "source/hal_lks32mc09x_wake.c"
#include "source/hal_lks32mc09x_irqfun.c"
```

hal_lks32mc09x.c 的引用(Include)关系图:



宏定义

- #define _HAL_LKS32MC05X_C_

4.1.1 宏定义说明

4.1.1.1 __HAL_LKS32MC05X_C__

```
#define __HAL_LKS32MC05X_C__
```

在文件 `hal_lks32mc09x.c` 第 2 行定义。

4.2 hal_lks32mc09x.c

[浏览该文件的文档。](#)

```
00001 #ifndef __HAL_LKS32MC05X_C__
00002 #define __HAL_LKS32MC05X_C__
00003 #include "source/hal_lks32mc09x_adc.c"
00004 #include "source/hal_lks32mc09x_clk.c"
00005 #include "source/hal_lks32mc09x_cmp.c"
00006 #include "source/hal_lks32mc09x_crc.c"
00007 #include "source/hal_lks32mc09x_dac.c"
00008 #include "source/hal_lks32mc09x_dma.c"
00009 #include "source/hal_lks32mc09x_dsp.c"
00010 #include "source/hal_lks32mc09x_flash.c"
00011 #include "source/hal_lks32mc09x_gpio.c"
00012 #include "source/hal_lks32mc09x_hall.c"
00013 #include "source/hal_lks32mc09x_i2c.c"
00014 #include "source/hal_lks32mc09x_init.c"
00015 #include "source/hal_lks32mc09x_iwdg.c"
00016 #include "source/hal_lks32mc09x_irq.c"
00017 #include "source/hal_lks32mc09x_mcpwm.c"
00018 #include "source/hal_lks32mc09x_opa.c"
00019 #include "source/hal_lks32mc09x_qep.c"
00020 #include "source/hal_lks32mc09x_reset.c"
00021 #include "source/hal_lks32mc09x_reg.c"
00022 #include "source/hal_lks32mc09x_spi.c"
00023 #include "source/hal_lks32mc09x_test.c"
00024 #include "source/hal_lks32mc09x_timer.c"
00025 #include "source/hal_lks32mc09x_tpm.c"
00026 #include "source/hal_lks32mc09x_trim.c"
00027 #include "source/hal_lks32mc09x_uart.c"
00028 #include "source/hal_lks32mc09x_wake.c"
00029 #include "source/hal_lks32mc09x_irqfun.c"
00030
00031 #endif
```

4.3 hal_lks32mc09x.h 文件参考

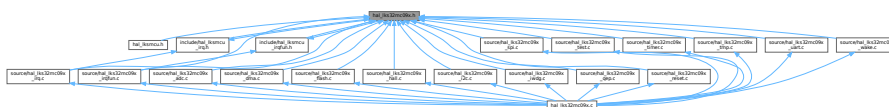
```
#include "basic.h"
#include "stdint.h"
#include "lks32mc09x.h"
#include "include/hal_lksmcu_adc.h"
#include "include/hal_lksmcu_can.h"
#include "include/hal_lksmcu_clk.h"
#include "include/hal_lksmcu_cmp.h"
#include "include/hal_lksmcu_crc.h"
#include "include/hal_lksmcu_dac.h"
#include "include/hal_lksmcu_dma.h"
#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_qep.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_ks32mc09x.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_TIMER_IRQHandler_00 TIMER0_IRQHandler

统一中断函数接口 命名方式: *HAL*-模块名称.*IRQHandler*-模块序号中断序号。例如*MCPWM*模块有两个*CNT*,会产生两个中断。这里分别对应*MCPWM0.IRQHandler*和*MCPWM1.IRQHandler*。为方便使用,末尾的*_00*可以省略。

- #define [HAL_TIMER_IRQHandler_10](#) TIMER1_IRQHandler
- #define [HAL_TIMER_IRQHandler_00](#) TIMER2_IRQHandler
- #define [HAL_MCPWM_IRQHandler_00](#) MCPWM0_IRQHandler
- #define [HAL_MCPWM_IRQHandler_01](#) MCPWM1_IRQHandler
- #define [HAL_I2C_IRQHandler_00](#) I2C0_IRQHandler
- #define [HAL_SPI_IRQHandler_00](#) SPI0_IRQHandler
- #define [HAL_GPIO_IRQHandler_00](#) GPIO_IRQHandler
- #define [HAL_HALL_IRQHandler_00](#) HALL0_IRQHandler
- #define [HAL_UART_IRQHandler_00](#) UART0_IRQHandler
- #define [HAL_UART_IRQHandler_10](#) UART1_IRQHandler
- #define [HAL_UART_IRQHandler_20](#) UART2_IRQHandler
- #define [HAL_CMP_IRQHandler_00](#) CMP0_IRQHandler
- #define [HAL_CMP_IRQHandler_10](#) CMP1_IRQHandler
- #define [HAL_ADC_IRQHandler_00](#) ADC0_IRQHandler
- #define [HAL_ADC_IRQHandler_10](#) ADC1_IRQHandler
- #define [HAL_WAKE_IRQHandler_00](#) WAKE_IRQHandler
- #define [HAL_QEP_IRQHandler_00](#) QEP0_IRQHandler
- #define [HAL_DMA_IRQHandler_00](#) DMA0_IRQHandler
- #define [HAL_SIF_IRQHandler_00](#) SIF0_IRQHandler
- #define [HAL_SW_IRQHandler_00](#) SW_IRQHandler

- #define HAL_PWRDN_IRQHandler_00 PWRDN_IRQHandler
- #define HAL_INIT_IRQHandler_00 HAL_PWRDN_IRQHandler_00
- #define HAL_CL_IRQHandler_00 CL0_IRQHandler
- #define HAL_TIMER_IRQHandler HAL_TIMER_IRQHandler_00
- #define HAL_MCPWM_IRQHandler HAL_MCPWM_IRQHandler_00
- #define HAL_I2C_IRQHandler HAL_I2C_IRQHandler_00
- #define HAL_SPI_IRQHandler HAL_SPI_IRQHandler_00
- #define HAL_GPIO_IRQHandler HAL_GPIO_IRQHandler_00
- #define HAL_HALL_IRQHandler HAL_HALL_IRQHandler_00
- #define HAL_UART_IRQHandler HAL_UART_IRQHandler_00
- #define HAL_CMP_IRQHandler HAL_CMP_IRQHandler_00
- #define HAL_ADC_IRQHandler HAL_ADC_IRQHandler_00
- #define HAL_DMA_IRQHandler HAL_DMA_IRQHandler_00
- #define HAL_WAKE_IRQHandler HAL_WAKE_IRQHandler_00
- #define HAL_QEP_IRQHandler HAL_QEP_IRQHandler_00
- #define HAL_SIF_IRQHandler HAL_SIF_IRQHandler_00
- #define HAL_SW_IRQHandler HAL_SW_IRQHandler_00
- #define HAL_PWRDN_IRQHandler HAL_PWRDN_IRQHandler_00
- #define HAL_INIT_IRQHandler HAL_PWRDN_IRQHandler_00
- #define HAL_CL_IRQHandler HAL_CL_IRQHandler_00
- #define HAL_TIMER_IRQHandler_30 HAL_TIMER_IRQHandler_30_Null
- #define HAL_QEP_IRQHandler_10 HAL_QEP_IRQHandler_10_Null
- #define HAL_DSP_IRQHandler_00 HAL_DSP_IRQHandler_00_Null
- #define HAL_CAN_IRQHandler_00 HAL_CAN_IRQHandler_00_Null
- #define HAL_DSP_IRQHandler HAL_DSP_IRQHandler_Null
- #define HAL_CAN_IRQHandler HAL_CAN_IRQHandler_Null
- #define HAL_I2C_IRQHandler_10 HAL_I2C_IRQHandler_10_Null
- #define HAL_HALL_IRQHandler_10 HAL_HALL_IRQHandler_10_Null
- #define HAL_CMP_IRQHandler_20 HAL_CMP_IRQHandler_20_Null
- #define HAL_CMP_IRQHandler_30 HAL_CMP_IRQHandler_30_Null
- #define HAL_CMP_IRQHandler_40 HAL_CMP_IRQHandler_40_Null
- #define HAL_CMP_IRQHandler_50 HAL_CMP_IRQHandler_50_Null
- #define HAL_MCPWM_IRQHandler_10 HAL_MCPWM_IRQHandler_10_Null
- #define HAL_MCPWM_IRQHandler_11 HAL_MCPWM_IRQHandler_11_Null
- #define HAL_TIMER_IRQHandler_40 HAL_TIMER_IRQHandler_40_Null
- #define HAL_QEP_IRQHandler_20 HAL_QEP_IRQHandler_20_Null
- #define HAL_QEP_IRQHandler_30 HAL_QEP_IRQHandler_30_Null
- #define HAL_ADC_IRQHandler_20 HAL_ADC_IRQHandler_20_Null
- #define HAL_SPI_IRQHandler_10 HAL_SPI_IRQHandler_10_Null
- #define HAL_WWDG_IRQHandler_00 HAL_WWDG_IRQHandler_00_Null
- #define HAL_FMAC_IRQHandler_00 HAL_FMAC_IRQHandler_00_Null
- #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

4.3.1 宏定义说明

4.3.1.1 HAL_ADC_IRQHandler

```
#define HAL_ADC_IRQHandler HAL_ADC_IRQHandler_00
```

在文件 [hal_iks32mc09x.h](#) 第 50 行定义。

4.3.1.2 HAL_ADC_IRQHandler_00

```
#define HAL_ADC_IRQHandler_00 ADC0_IRQHandler
```

在文件 [hal_lks32mc09x.h](#) 第 31 行定义.

4.3.1.3 HAL_ADC_IRQHandler_10

```
#define HAL_ADC_IRQHandler_10 ADC1_IRQHandler
```

在文件 [hal_lks32mc09x.h](#) 第 32 行定义.

4.3.1.4 HAL_ADC_IRQHandler_20

```
#define HAL_ADC_IRQHandler_20 HAL_ADC_IRQHandler_20_Null
```

在文件 [hal_lks32mc09x.h](#) 第 78 行定义.

4.3.1.5 HAL_CAN_IRQHandler

```
#define HAL_CAN_IRQHandler HAL_CAN_IRQHandler_Null
```

在文件 [hal_lks32mc09x.h](#) 第 66 行定义.

4.3.1.6 HAL_CAN_IRQHandler_00

```
#define HAL_CAN_IRQHandler_00 HAL_CAN_IRQHandler_00_Null
```

在文件 [hal_lks32mc09x.h](#) 第 64 行定义.

4.3.1.7 HAL_CL_IRQHandler

```
#define HAL_CL_IRQHandler HAL_CL_IRQHandler_00
```

在文件 [hal_lks32mc09x.h](#) 第 58 行定义.

4.3.1.8 HAL_CL_IRQHandler_00

```
#define HAL_CL_IRQHandler_00 CL0_IRQHandler
```

在文件 [hal_lks32mc09x.h](#) 第 40 行定义.

4.3.1.9 HAL_CMP_IRQHandler

```
#define HAL_CMP_IRQHandler HAL_CMP_IRQHandler_00
```

在文件 [hal_lks32mc09x.h](#) 第 49 行定义.

4.3.1.10 HAL_CMP_IRQHandler_00

```
#define HAL_CMP_IRQHandler_00 CMP0_IRQHandler
```

在文件 [hal_lks32mc09x.h](#) 第 29 行定义.

4.3.1.11 HAL_CMP_IRQHandler_10

```
#define HAL_CMP_IRQHandler_10 CMP1_IRQHandler
```

在文件 [hal_lks32mc09x.h](#) 第 30 行定义.

4.3.1.12 HAL_CMP_IRQHandler_20

```
#define HAL_CMP_IRQHandler_20 HAL_CMP_IRQHandler_20_Null
```

在文件 [hal_lks32mc09x.h](#) 第 69 行定义.

4.3.1.13 HAL_CMP_IRQHandler_30

```
#define HAL_CMP_IRQHandler_30 HAL_CMP_IRQHandler_30_Null
```

在文件 [hal_lks32mc09x.h](#) 第 70 行定义.

4.3.1.14 HAL_CMP_IRQHandler_40

```
#define HAL_CMP_IRQHandler_40 HAL_CMP_IRQHandler_40_Null
```

在文件 [hal_lks32mc09x.h](#) 第 71 行定义.

4.3.1.15 HAL_CMP_IRQHandler_50

```
#define HAL_CMP_IRQHandler_50 HAL_CMP_IRQHandler_50_Null
```

在文件 [hal_lks32mc09x.h](#) 第 72 行定义.

4.3.1.16 HAL_DMA_IRQHandler

```
#define HAL_DMA_IRQHandler HAL_DMA_IRQHandler_00
```

在文件 [hal_lks32mc09x.h](#) 第 51 行定义.

4.3.1.17 HAL_DMA_IRQHandler_00

```
#define HAL_DMA_IRQHandler_00 DMA0_IRQHandler
```

在文件 [hal_lks32mc09x.h](#) 第 35 行定义.

4.3.1.18 HAL_DSP_IRQHandler

```
#define HAL_DSP_IRQHandler HAL_DSP_IRQHandler_Null
```

在文件 [hal_lks32mc09x.h](#) 第 65 行定义.

4.3.1.19 HAL_DSP_IRQHandler_00

```
#define HAL_DSP_IRQHandler_00 HAL_DSP_IRQHandler_00_Null
```

在文件 [hal_lks32mc09x.h](#) 第 63 行定义.

4.3.1.20 HAL_FMAC_IRQHandler_00

```
#define HAL_FMAC_IRQHandler_00 HAL_FMAC_IRQHandler_00_Null
```

在文件 [hal_lks32mc09x.h](#) 第 81 行定义.

4.3.1.21 HAL_GPIO_IRQHandler

```
#define HAL_GPIO_IRQHandler HAL_GPIO_IRQHandler_00
```

在文件 [hal_lks32mc09x.h](#) 第 46 行定义.

4.3.1.22 HAL_GPIO_IRQHandler_00

```
#define HAL_GPIO_IRQHandler_00 GPIO_IRQHandler
```

在文件 [hal_lks32mc09x.h](#) 第 24 行定义.

4.3.1.23 HAL_HALL_IRQHandler

```
#define HAL_HALL_IRQHandler HAL_HALL_IRQHandler_00
```

在文件 [hal_lks32mc09x.h](#) 第 47 行定义.

4.3.1.24 HAL_HALL_IRQHandler_00

```
#define HAL_HALL_IRQHandler_00 HALL0_IRQHandler
```

在文件 [hal_lks32mc09x.h](#) 第 25 行定义.

4.3.1.25 HAL_HALL_IRQHandler_10

```
#define HAL_HALL_IRQHandler_10 HAL_HALL_IRQHandler_10_Null
```

在文件 [hal_lks32mc09x.h](#) 第 68 行定义.

4.3.1.26 HAL_I2C_IRQHandler

```
#define HAL_I2C_IRQHandler HAL_I2C_IRQHandler_00
```

在文件 [hal_lks32mc09x.h](#) 第 44 行定义.

4.3.1.27 HAL_I2C_IRQHandler_00

```
#define HAL_I2C_IRQHandler_00 I2C0_IRQHandler
```

在文件 [hal_lks32mc09x.h](#) 第 22 行定义.

4.3.1.28 HAL_I2C_IRQHandler_10

```
#define HAL_I2C_IRQHandler_10 HAL_I2C_IRQHandler_10_Null
```

在文件 [hal_lks32mc09x.h](#) 第 67 行定义.

4.3.1.29 HAL_INIT_DEFAULE_MCLK

```
#define HAL_INIT_DEFAULE_MCLK 96000000
```

芯片的默认时钟频率

在文件 [hal_lks32mc09x.h](#) 第 86 行定义.

4.3.1.30 HAL_INIT_DEFAULE_MCLK_MS

```
#define HAL_INIT_DEFAULE_MCLK_MS 96000
```

在文件 [hal_lks32mc09x.h](#) 第 88 行定义.

4.3.1.31 HAL_INIT_DEFAULE_MCLK_US

```
#define HAL_INIT_DEFAULE_MCLK_US 96
```

在文件 [hal_lks32mc09x.h](#) 第 87 行定义.

4.3.1.32 HAL_INIT_IRQHandler

```
#define HAL_INIT_IRQHandler HAL_PWRDN_IRQHandler_00
```

在文件 [hal_lks32mc09x.h](#) 第 57 行定义.

4.3.1.33 HAL_INIT_IRQHandler_00

```
#define HAL_INIT_IRQHandler_00 HAL_PWRDN_IRQHandler_00
```

在文件 [hal_lks32mc09x.h](#) 第 39 行定义.

4.3.1.34 HAL_MCPWM_IRQHandler

```
#define HAL_MCPWM_IRQHandler HAL_MCPWM_IRQHandler_00
```

在文件 [hal_lks32mc09x.h](#) 第 43 行定义.

4.3.1.35 HAL_MCPWM_IRQHandler_00

```
#define HAL_MCPWM_IRQHandler_00 MCPWM0_IRQHandler
```

在文件 [hal_lks32mc09x.h](#) 第 20 行定义.

4.3.1.36 HAL_MCPWM_IRQHandler_01

```
#define HAL_MCPWM_IRQHandler_01 MCPWM1_IRQHandler
```

在文件 [hal_lks32mc09x.h](#) 第 21 行定义.

4.3.1.37 HAL_MCPWM_IRQHandler_10

```
#define HAL_MCPWM_IRQHandler_10 HAL_MCPWM_IRQHandler_10_Null
```

在文件 [hal_lks32mc09x.h](#) 第 73 行定义.

4.3.1.38 HAL_MCPWM_IRQHandler_11

```
#define HAL_MCPWM_IRQHandler_11 HAL_MCPWM_IRQHandler_11_Null
```

在文件 [hal_lks32mc09x.h](#) 第 74 行定义.

4.3.1.39 HAL_PWRDN_IRQHandler

```
#define HAL_PWRDN_IRQHandler HAL_PWRDN_IRQHandler_00
```

在文件 [hal_lks32mc09x.h](#) 第 56 行定义.

4.3.1.40 HAL_PWRDN_IRQHandler_00

```
#define HAL_PWRDN_IRQHandler_00 PWRDN_IRQHandler
```

在文件 [hal_lks32mc09x.h](#) 第 38 行定义.

4.3.1.41 HAL_QEP_IRQHandler

```
#define HAL_QEP_IRQHandler HAL_QEP_IRQHandler_00
```

在文件 [hal_lks32mc09x.h](#) 第 53 行定义.

4.3.1.42 HAL_QEP_IRQHandler_00

```
#define HAL_QEP_IRQHandler_00 QEP0_IRQHandler
```

在文件 [hal_lks32mc09x.h](#) 第 34 行定义.

4.3.1.43 HAL_QEP_IRQHandler_10

```
#define HAL_QEP_IRQHandler_10 HAL_QEP_IRQHandler_10_Null
```

在文件 [hal_lks32mc09x.h](#) 第 62 行定义.

4.3.1.44 HAL_QEP_IRQHandler_20

```
#define HAL_QEP_IRQHandler_20 HAL_QEP_IRQHandler_20_Null
```

在文件 [hal_lks32mc09x.h](#) 第 76 行定义.

4.3.1.45 HAL_QEP_IRQHandler_30

```
#define HAL_QEP_IRQHandler_30 HAL_QEP_IRQHandler_30_Null
```

在文件 [hal_lks32mc09x.h](#) 第 77 行定义.

4.3.1.46 HAL_SIF_IRQHandler

```
#define HAL_SIF_IRQHandler HAL_SIF_IRQHandler_00
```

在文件 [hal_lks32mc09x.h](#) 第 54 行定义.

4.3.1.47 HAL_SIF_IRQHandler_00

```
#define HAL_SIF_IRQHandler_00 SIF0_IRQHandler
```

在文件 [hal_lks32mc09x.h](#) 第 36 行定义.

4.3.1.48 HAL_SPI_IRQHandler

```
#define HAL_SPI_IRQHandler HAL_SPI_IRQHandler_00
```

在文件 [hal_lks32mc09x.h](#) 第 45 行定义.

4.3.1.49 HAL_SPI_IRQHandler_00

```
#define HAL_SPI_IRQHandler_00 SPI0_IRQHandler
```

在文件 [hal_lks32mc09x.h](#) 第 23 行定义.

4.3.1.50 HAL_SPI_IRQHandler_10

```
#define HAL_SPI_IRQHandler_10 HAL_SPI_IRQHandler_10_Null
```

在文件 [hal_lks32mc09x.h](#) 第 79 行定义.

4.3.1.51 HAL_SW_IRQHandler

```
#define HAL_SW_IRQHandler HAL_SW_IRQHandler_00
```

在文件 [hal_lks32mc09x.h](#) 第 55 行定义.

4.3.1.52 HAL_SW_IRQHandler_00

```
#define HAL_SW_IRQHandler_00 SW_IRQHandler
```

在文件 [hal_lks32mc09x.h](#) 第 37 行定义.

4.3.1.53 HAL_TIMER_IRQHandler

```
#define HAL_TIMER_IRQHandler HAL_TIMER_IRQHandler_00
```

在文件 [hal_lks32mc09x.h](#) 第 42 行定义.

4.3.1.54 HAL_TIMER_IRQHandler_00

```
#define HAL_TIMER_IRQHandler_00 TIMER0_IRQHandler
```

统一中断函数接口 命名方式: HAL_模块名称_IRQHandler_模块序号中断序号。例如MCPWM模块有有两个CNT,会产生两个中断。这里分别对应MCPWM0_IRQHandler和MCPWM1_IRQHandler。为方便使用,末尾的_00可以省略。

在文件 [hal_lks32mc09x.h](#) 第 17 行定义.

4.3.1.55 HAL_TIMER_IRQHandler_10

```
#define HAL_TIMER_IRQHandler_10 TIMER1_IRQHandler
```

在文件 [hal_lks32mc09x.h](#) 第 18 行定义.

4.3.1.56 HAL_TIMER_IRQHandler_20

```
#define HAL_TIMER_IRQHandler_20 TIMER2_IRQHandler
```

在文件 [hal_lks32mc09x.h](#) 第 19 行定义.

4.3.1.57 HAL_TIMER_IRQHandler_30

```
#define HAL_TIMER_IRQHandler_30 HAL_TIMER_IRQHandler_30_Null
```

在文件 [hal_lks32mc09x.h](#) 第 61 行定义.

4.3.1.58 HAL_TIMER_IRQHandler_40

```
#define HAL_TIMER_IRQHandler_40 HAL_TIMER_IRQHandler_40_Null
```

在文件 [hal_lks32mc09x.h](#) 第 75 行定义.

4.3.1.59 HAL_UART_IRQHandler

```
#define HAL_UART_IRQHandler HAL_UART_IRQHandler_00
```

在文件 [hal_lks32mc09x.h](#) 第 48 行定义.

4.3.1.60 HAL_UART_IRQHandler_00

```
#define HAL_UART_IRQHandler_00 UART0_IRQHandler
```

在文件 [hal_lks32mc09x.h](#) 第 26 行定义.

4.3.1.61 HAL_UART_IRQHandler_10

```
#define HAL_UART_IRQHandler_10 UART1_IRQHandler
```

在文件 [hal_lks32mc09x.h](#) 第 27 行定义.

4.3.1.62 HAL_UART_IRQHandler_20

```
#define HAL_UART_IRQHandler_20 UART2_IRQHandler
```

在文件 [hal_lks32mc09x.h](#) 第 28 行定义.

4.3.1.63 HAL_WAKE_IRQHandler

```
#define HAL_WAKE_IRQHandler HAL_WAKE_IRQHandler_00
```

在文件 [hal_lks32mc09x.h](#) 第 52 行定义.

4.3.1.64 HAL_WAKE_IRQHandler_00

```
#define HAL_WAKE_IRQHandler_00 WAKE_IRQHandler
```

在文件 [hal_lks32mc09x.h](#) 第 33 行定义。

4.3.1.65 HAL_WWDG_IRQHandler_00

```
#define HAL_WWDG_IRQHandler_00 HAL_WWDG_IRQHandler_00_Null
```

在文件 [hal_lks32mc09x.h](#) 第 80 行定义。

4.4 hal_lks32mc09x.h

[浏览该文件的文档。](#)

```
00001 #include "basic.h"
00002 #include "stdint.h"
00003 #include "lks32mc09x.h"
00004
00005 #ifndef __HAL_LKS32MC09X_H__
00006 #define __HAL_LKS32MC09X_H__
00007
00017 #define HAL_TIMER_IRQHandler_00 TIMER0_IRQHandler
00018 #define HAL_TIMER_IRQHandler_10 TIMER1_IRQHandler
00019 #define HAL_TIMER_IRQHandler_20 TIMER2_IRQHandler
00020 #define HAL_MCPWM_IRQHandler_00 MCPWM0_IRQHandler
00021 #define HAL_MCPWM_IRQHandler_01 MCPWM1_IRQHandler
00022 #define HAL_I2C_IRQHandler_00 I2C0_IRQHandler
00023 #define HAL_SPI_IRQHandler_00 SPI0_IRQHandler
00024 #define HAL_GPIO_IRQHandler_00 GPIO_IRQHandler
00025 #define HAL_HALL_IRQHandler_00 HALL0_IRQHandler
00026 #define HAL_UART_IRQHandler_00 UART0_IRQHandler
00027 #define HAL_UART_IRQHandler_10 UART1_IRQHandler
00028 #define HAL_UART_IRQHandler_20 UART2_IRQHandler
00029 #define HAL_CMP_IRQHandler_00 CMP0_IRQHandler
00030 #define HAL_CMP_IRQHandler_10 CMP1_IRQHandler
00031 #define HAL_ADC_IRQHandler_00 ADC0_IRQHandler
00032 #define HAL_ADC_IRQHandler_10 ADC1_IRQHandler
00033 #define HAL_WAKE_IRQHandler_00 WAKE_IRQHandler
00034 #define HAL_QEP_IRQHandler_00 QEP0_IRQHandler
00035 #define HAL_DMA_IRQHandler_00 DMA0_IRQHandler
00036 #define HAL_SIF_IRQHandler_00 SIF0_IRQHandler
00037 #define HAL_SW_IRQHandler_00 SW_IRQHandler
00038 #define HAL_PWRDN_IRQHandler_00 PWRDN_IRQHandler
00039 #define HAL_INIT_IRQHandler_00 HAL_PWRDN_IRQHandler_00
00040 #define HAL_CL_IRQHandler_00 CL0_IRQHandler
00041 // 简化写法
00042 #define HAL_TIMER_IRQHandler HAL_TIMER_IRQHandler_00
00043 #define HAL_MCPWM_IRQHandler HAL_MCPWM_IRQHandler_00
00044 #define HAL_I2C_IRQHandler HAL_I2C_IRQHandler_00
00045 #define HAL_SPI_IRQHandler HAL_SPI_IRQHandler_00
00046 #define HAL_GPIO_IRQHandler HAL_GPIO_IRQHandler_00
00047 #define HAL_HALL_IRQHandler HAL_HALL_IRQHandler_00
00048 #define HAL_UART_IRQHandler HAL_UART_IRQHandler_00
00049 #define HAL_CMP_IRQHandler HAL_CMP_IRQHandler_00
00050 #define HAL_ADC_IRQHandler HAL_ADC_IRQHandler_00
00051 #define HAL_DMA_IRQHandler HAL_DMA_IRQHandler_00
00052 #define HAL_WAKE_IRQHandler HAL_WAKE_IRQHandler_00
00053 #define HAL_QEP_IRQHandler HAL_QEP_IRQHandler_00
00054 #define HAL_SIF_IRQHandler HAL_SIF_IRQHandler_00
00055 #define HAL_SW_IRQHandler HAL_SW_IRQHandler_00
00056 #define HAL_PWRDN_IRQHandler HAL_PWRDN_IRQHandler_00
00057 #define HAL_INIT_IRQHandler HAL_PWRDN_IRQHandler_00
00058 #define HAL_CL_IRQHandler HAL_CL_IRQHandler_00
00059
00060 // 当前芯片不支持的中断函数
00061 #define HAL_TIMER_IRQHandler_30 HAL_TIMER_IRQHandler_30_Null
00062 #define HAL_QEP_IRQHandler_10 HAL_QEP_IRQHandler_10_Null
00063 #define HAL_DSP_IRQHandler_00 HAL_DSP_IRQHandler_00_Null
00064 #define HAL_CAN_IRQHandler_00 HAL_CAN_IRQHandler_00_Null
00065 #define HAL_DSP_IRQHandler HAL_DSP_IRQHandler_Null
00066 #define HAL_CAN_IRQHandler HAL_CAN_IRQHandler_Null
00067 #define HAL_I2C_IRQHandler_10 HAL_I2C_IRQHandler_10_Null
```

```

00068 #define HAL_HALL_IRQHandler_10 HAL_HALL_IRQHandler_10_Null
00069 #define HAL_CMP_IRQHandler_20 HAL_CMP_IRQHandler_20_Null
00070 #define HAL_CMP_IRQHandler_30 HAL_CMP_IRQHandler_30_Null
00071 #define HAL_CMP_IRQHandler_40 HAL_CMP_IRQHandler_40_Null
00072 #define HAL_CMP_IRQHandler_50 HAL_CMP_IRQHandler_50_Null
00073 #define HAL_MCPWM_IRQHandler_10 HAL_MCPWM_IRQHandler_10_Null
00074 #define HAL_MCPWM_IRQHandler_11 HAL_MCPWM_IRQHandler_11_Null
00075 #define HAL_TIMER_IRQHandler_40 HAL_TIMER_IRQHandler_40_Null
00076 #define HAL_QEP_IRQHandler_20 HAL_QEP_IRQHandler_20_Null
00077 #define HAL_QEP_IRQHandler_30 HAL_QEP_IRQHandler_30_Null
00078 #define HAL_ADC_IRQHandler_20 HAL_ADC_IRQHandler_20_Null
00079 #define HAL_SPI_IRQHandler_10 HAL_SPI_IRQHandler_10_Null
00080 #define HAL_WWDG_IRQHandler_00 HAL_WWDG_IRQHandler_00_Null
00081 #define HAL_FMAC_IRQHandler_00 HAL_FMAC_IRQHandler_00_Null
00082
00086 #define HAL_INIT_DEFAULT_MCLK 96000000
00087 #define HAL_INIT_DEFAULT_MCLK_US 96 // 上电后的默认时钟1us分频系数
00088 #define HAL_INIT_DEFAULT_MCLK_MS 96000 // 上电后的默认时钟1ms分频系数
00089
00090 #include "include/hal_lksmcu_adc.h"
00091 #include "include/hal_lksmcu_can.h"
00092 #include "include/hal_lksmcu_clk.h"
00093 #include "include/hal_lksmcu_cmp.h"
00094 #include "include/hal_lksmcu_crc.h"
00095 #include "include/hal_lksmcu_dac.h"
00096 #include "include/hal_lksmcu_dma.h"
00097 #include "include/hal_lksmcu_dsp.h"
00098 #include "include/hal_lksmcu_flash.h"
00099 #include "include/hal_lksmcu_gpio.h"
00100 #include "include/hal_lksmcu_hall.h"
00101 #include "include/hal_lksmcu_i2c.h"
00102 #include "include/hal_lksmcu_init.h"
00103 #include "include/hal_lksmcu_iwdg.h"
00104 #include "include/hal_lksmcu_irq.h"
00105 #include "include/hal_lksmcu_mcpwm.h"
00106 #include "include/hal_lksmcu_opa.h"
00107 #include "include/hal_lksmcu_qep.h"
00108 #include "include/hal_lksmcu_reg.h"
00109 #include "include/hal_lksmcu_reset.h"
00110 #include "include/hal_lksmcu_spi.h"
00111 #include "include/hal_lksmcu_timer.h"
00112 #include "include/hal_lksmcu_tmp.h"
00113 #include "include/hal_lksmcu_trim.h"
00114 #include "include/hal_lksmcu_uart.h"
00115 #include "include/hal_lksmcu_wake.h"
00116 #include "include/hal_lksmcu_irqfun.h"
00117 #endif

```

4.5 hal_lksmcu.h 文件参考

#include "hal_lks32mc09x.h"
hal_lksmcu.h 的引用(Include)关系图:



4.6 hal_lksmcu.h

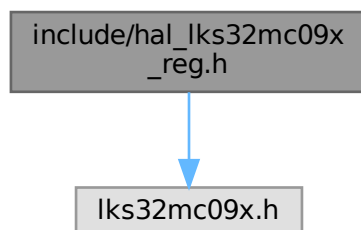
[浏览该文件的文档.](#)

```
00001 #include "hal_lks32mc09x.h"
```

4.7 include/hal_lks32mc09x_reg.h 文件参考

```
#include "lks32mc09x.h"
```

hal_lks32mc09x_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_ADC_TypeDef](#)
ADC0寄存器 (ADC0)
- struct [HAL_CMP_TypeDef](#)
比较器寄存器 (CMP)
- struct [HAL_CLU_TypeDef](#)
CL0寄存器
- struct [HAL_CRC_TypeDef](#)
CRC0寄存器 (CRC0)
- struct [HAL_DMA_TypeDef](#)
DMA寄存器
- struct [HAL_DMAMODULE_TypeDef](#)
DMA控制状态寄存器对象定义
- struct [HAL_DSP_TypeDef](#)
DSP0寄存器结构体 (DSP0)
- struct [HAL_EXTI_TypeDef](#)
EXTI寄存器 (EXTI)
- struct [HAL_FLASH_TypeDef](#)
FLASH寄存器 (FLASH)
- struct [HAL_GPIO_TypeDef](#)
GPIO寄存器
- struct [HAL_HALL_TypeDef](#)
HALL0寄存器 (HALL0)
- struct [HAL_I2C_TypeDef](#)

- *I2C0寄存器 (I2C0)*
- struct [HAL_MCPWM_TypeDef](#)
MCPWM0寄存器 (MCPWM0)
- struct [HAL_QEP_TypeDef](#)
QEP0寄存器 (QEP0)
- struct [HAL_SIF_TypeDef](#)
SIF0寄存器 (SIF0)
- struct [HAL_SPI_TypeDef](#)
SPI0寄存器 (SPI0)
- struct [HAL_SYS_TypeDef](#)
SYS寄存器 (SYS)
- struct [HAL_TIMER_TypeDef](#)
TIMER0寄存器 (TIMER0)
- struct [HAL_UART_TypeDef](#)
UART寄存器
- struct [HAL_IWDG_TypeDef](#)
IWDG寄存器结构体句柄
- struct [HAL_AON_TypeDef](#)
AON寄存器结构体句柄

宏定义

- #define [LKS32MC09X](#)
- #define [CLU0](#) (([HAL_CLU_TypeDef](#) *)CL0.BASE)
- #define [DMA_CHN0](#) (([HAL_DMA_TypeDef](#) *)DMA0.BASE)
- #define [DMA_CHN1](#) (([HAL_DMA_TypeDef](#) *) (DMA0.BASE + 0x20))
- #define [DMA_CHN2](#) (([HAL_DMA_TypeDef](#) *) (DMA0.BASE + 0x40))
- #define [DMA_CHN3](#) (([HAL_DMA_TypeDef](#) *) (DMA0.BASE + 0x60))
- #define [DMA_TOP](#) (([HAL_DMAMODULE_TypeDef](#) *) (DMA0.BASE + 0x80))
- #define [EXTI0](#) (([HAL_EXTI_TypeDef](#) *)EXTI.BASE)
- #define [QEP0](#) (([HAL_QEP_TypeDef](#) *)QEP0.BASE)
- #define [SIF0](#) (([HAL_SIF_TypeDef](#) *) (SIF0.BASE))
- #define [AON0](#) (([HAL_AON_TypeDef](#) *)AON.BASE)

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_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 AON0

```
#define AON0 ((HAL_AON_TypeDef *)AON_BASE)
```

在文件 [hal_lks32mc09x_reg.h](#) 第 385 行定义.

4.7.1.2 CLU0

```
#define CLU0 ((HAL_CLU_TypeDef *)CL0_BASE)
```

在文件 [hal_lks32mc09x_reg.h](#) 第 376 行定义.

4.7.1.3 DMA_CHN0

```
#define DMA_CHN0 ((HAL_DMA_TypeDef *)DMA0_BASE)
```

在文件 [hal_lks32mc09x_reg.h](#) 第 377 行定义.

4.7.1.4 DMA_CHN1

```
#define DMA_CHN1 ((HAL_DMA_TypeDef *) (DMA0_BASE + 0x20))
```

在文件 [hal_lks32mc09x_reg.h](#) 第 378 行定义.

4.7.1.5 DMA_CHN2

```
#define DMA_CHN2 ((HAL_DMA_TypeDef *) (DMA0_BASE + 0x40))
```

在文件 [hal_lks32mc09x_reg.h](#) 第 379 行定义.

4.7.1.6 DMA_CHN3

```
#define DMA_CHN3 ((HAL_DMA_TypeDef *) (DMA0_BASE + 0x60))
```

在文件 [hal_lks32mc09x_reg.h](#) 第 380 行定义.

4.7.1.7 DMA_TOP

```
#define DMA_TOP ((HAL_DMAMODULE_TypeDef *) (DMA0_BASE + 0x80))
```

在文件 [hal_lks32mc09x_reg.h](#) 第 381 行定义.

4.7.1.8 EXTI0

```
#define EXTI0 ((HAL_EXTI_TypeDef *)EXTI_BASE)
```

在文件 [hal_lks32mc09x_reg.h](#) 第 382 行定义.

4.7.1.9 LKS32MC09X

```
#define LKS32MC09X
```

在文件 [hal_lks32mc09x_reg.h](#) 第 6 行定义.

4.7.1.10 QEP0

```
#define QEP0 ((HAL_QEP_TypeDef *)QEP0_BASE)
```

在文件 [hal_lks32mc09x_reg.h](#) 第 383 行定义.

4.7.1.11 SIF0

```
#define SIF0 ((HAL_SIF_TypeDef *)SIF0_BASE)
```

在文件 [hal_lks32mc09x_reg.h](#) 第 384 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_lks32mc09x_reg.c](#) 第 13 行定义.

4.7.2.2 HAL_CMP

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

在文件 [hal_lks32mc09x_reg.c](#) 第 11 行定义.

4.7.2.3 HAL_CRC

```
HAL_CRC_TypeDef* const HAL_CRC[] [extern]
```

在文件 [hal_lks32mc09x_reg.c](#) 第 18 行定义.

4.7.2.4 HAL_DMA

`HAL_DMA_TypeDef* const HAL_DMA[] [extern]`

在文件 `hal_lks32mc09x_reg.c` 第 22 行定义.

4.7.2.5 HAL_DSP

`HAL_DSP_TypeDef* const HAL_DSP[] [extern]`

在文件 `hal_lks32mc09x_reg.c` 第 20 行定义.

4.7.2.6 HAL_FLASH

`HAL_FLASH_TypeDef* const HAL_FLASH[] [extern]`

在文件 `hal_lks32mc09x_reg.c` 第 8 行定义.

4.7.2.7 HAL_GPIO

`HAL_GPIO_TypeDef* const HAL_GPIO[] [extern]`

在文件 `hal_lks32mc09x_reg.c` 第 17 行定义.

4.7.2.8 HAL_HALL

`HAL_HALL_TypeDef* const HAL_HALL[] [extern]`

在文件 `hal_lks32mc09x_reg.c` 第 12 行定义.

4.7.2.9 HAL_I2C

`HAL_I2C_TypeDef* const HAL_I2C[] [extern]`

在文件 `hal_lks32mc09x_reg.c` 第 10 行定义.

4.7.2.10 HAL_IWDG

`HAL_IWDG_TypeDef* const HAL_IWDG[] [extern]`

在文件 `hal_lks32mc09x_reg.c` 第 21 行定义.

4.7.2.11 HAL_MCPWM

`HAL_MCPWM_TypeDef* const HAL_MCPWM[] [extern]`

在文件 `hal_lks32mc09x_reg.c` 第 16 行定义.

4.7.2.12 HAL_QEP

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

在文件 [hal_lks32mc09x_reg.c](#) 第 15 行定义.

4.7.2.13 HAL_SPI

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

在文件 [hal_lks32mc09x_reg.c](#) 第 9 行定义.

4.7.2.14 HAL_SYS

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

在文件 [hal_lks32mc09x_reg.c](#) 第 7 行定义.

4.7.2.15 HAL_TIMER

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

在文件 [hal_lks32mc09x_reg.c](#) 第 14 行定义.

4.7.2.16 HAL_UART

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

在文件 [hal_lks32mc09x_reg.c](#) 第 19 行定义.

4.8 hal_lks32mc09x_reg.h

[浏览该文件的文档.](#)

```
00001
00002 #ifndef __HAL_LKS32MC09X_REG_H__
00003 #define __HAL_LKS32MC09X_REG_H__
00004 #include "lks32mc09x.h"
00005 #ifndef LKS32MC09X
00006 #define LKS32MC09X
00007 #endif
00008
00012 typedef struct
00013 {
00014     __IOM uint32_t DAT[12];
00015     __IM uint32_t RESERVED[8]; //
00016     __IOM uint32_t CHN0;
00017     __IOM uint32_t CHN1;
00018     __IOM uint32_t CHN2;
00019     __IM uint32_t RESERVED1; //
00020     __IOM uint32_t CHNT;
00021     __IOM uint32_t GAIN;
00022     __IM uint32_t RESERVED2[3]; //
00023     __IOM uint32_t CFG;
00024     __IOM uint32_t TRIG;
00025     __OM uint32_t SWT;
00026     __IOM uint32_t DC0;
```



```

00027     __IOM uint32_t  AMCO;
00028     __IOM uint32_t  DC1;
00029     __IOM uint32_t  AMC1;
00030     __IOM uint32_t  IE;
00031     __IOM uint32_t  IF;
00032     __IM uint32_t  RESERVED3[11]; //
00033     __IOM uint32_t  LTH;
00034     __IOM uint32_t  HTH;
00035     __IOM uint32_t  GEN;
00036 } HAL_ADC_TypeDef;
00040 typedef struct
00041 {
00042     __IOM uint32_t  IE;
00043     __IOM uint32_t  IF;
00044     __IOM uint32_t  TCLK;
00045     __IOM uint32_t  CFG;
00046     __IOM uint32_t  BLCWIN;
00047     __IOM uint32_t  DATA;
00048 } HAL_CMP_TypeDef;
00052 typedef struct
00053 {
00054     __IOM uint32_t  EN;
00055     __IOM uint32_t  IE;
00056     __IOM uint32_t  IF;
00057     __IOM uint32_t  OUT;
00058     __IOM uint32_t  MX;
00059     __IOM uint32_t  FN;
00060     __IOM uint32_t  CF;
00061 } HAL_CLU_TypeDef;
00065 typedef struct
00066 {
00067     __IOM uint32_t  DR;
00068     __IOM uint32_t  CR;
00069     __IOM uint32_t  INIT;
00070     __IOM uint32_t  POL;
00071 } HAL_CRC_TypeDef;
00075 typedef struct
00076 {
00077     __IOM uint32_t  CCR;
00078     __IOM uint32_t  REN;
00079     __IOM uint32_t  CTMS;
00080     __IOM uint32_t  SADR;
00081     __IOM uint32_t  DADR;
00082 } HAL_DMA_TypeDef;
00086 typedef struct
00087 {
00088     __IOM uint32_t  CTRL;
00089     __IOM uint32_t  IE;
00090     __IOM uint32_t  IF;
00091 } HAL_DMAMODULE_TypeDef;
00095 typedef struct
00096 {
00097     __IM uint32_t  RESERVED[1032];
00098     __IOM uint32_t  DID;
00099     __IOM uint32_t  DIS;
00100     __IM uint32_t  QUO;
00101     __IM uint32_t  REM;
00102     __IOM uint32_t  DID1;
00103     __IOM uint32_t  DIS1;
00104     __IM uint32_t  QUO1;
00105     __IM uint32_t  RESERVED1;
00106     __IOM uint32_t  RAD;
00107     __IM uint16_t  SQRT;
00108     __IM uint16_t  RESERVED2;
00109 } HAL_DSP_TypeDef;
00113 typedef struct
00114 {
00115     __IOM uint32_t  CR0;
00116     __IOM uint32_t  CR1;
00117     __IOM uint32_t  IE;
00118     __IOM uint32_t  IF;
00119     __IOM uint32_t  CLKO;
00120     __IOM uint32_t  SWAP;
00121 } HAL_EXTI_TypeDef;
00125 typedef struct
00126 {
00127     __IOM uint32_t  CFG;
00128     __IOM uint32_t  ADDR;
00129     __IOM uint32_t  WDATA;
00130     __IM uint32_t  RDATA;
00131     __IOM uint32_t  ERASE;
00132     __IM uint32_t  PROTECT;
00133     __IM uint32_t  READY;
00134     __IM uint32_t  SIZE;
00135     __IOM uint32_t  NCFG;
00136     __IOM uint32_t  NADDR;
00137     __IOM uint32_t  NWDATA;

```

```

00138     __IM uint32_t NRDATA;
00139     __OM uint32_t NERASE;
00140     __OM uint32_t NKEY;
00141     __IOM uint32_t EPSM;
00142     __IM uint32_t RESERVED; //
00143     __IOM uint32_t EPSN;
00144     __IOM uint32_t LIB_ADDR;
00145 } HAL_FLASH_TypeDef;
00149 typedef struct
00150 {
00151     __IOM uint32_t PIE;
00152     __IOM uint32_t POE;
00153     __IOM uint32_t PDI;
00154     __IOM uint32_t PDO;
00155     __IOM uint32_t PUE;
00156     __IM uint32_t RESERVED; //
00157     __IOM uint32_t PODE;
00158     __IOM uint32_t PFLT;
00159     __IOM uint32_t F3210;
00160     __IOM uint32_t F7654;
00161     __IOM uint32_t FBA98;
00162     __IOM uint32_t FFEDC;
00163     __OM uint32_t BSRR;
00164     __OM uint32_t BRR;
00165 } HAL_GPIO_TypeDef;
00169 typedef struct
00170 {
00171     __IOM uint32_t CFG;
00172     __IOM uint32_t INFO;
00173     __IM uint32_t WIDTH;
00174     __IOM uint32_t TH;
00175     __IOM uint32_t CNT;
00176 } HAL_HALL_TypeDef;
00180 typedef struct
00181 {
00182     __IOM uint32_t ADDR;
00183     __IOM uint32_t CFG;
00184     __IOM uint32_t SCR;
00185     __IOM uint32_t DATA;
00186     __IOM uint32_t MSCR;
00187     __IOM uint32_t BCR;
00188     __IOM uint32_t BSIZ;
00189 } HAL_I2C_TypeDef;
00193 typedef struct
00194 {
00195     __IOM uint32_t TH[8];
00196     __IM uint32_t RESERVED[4]; //
00197     __IOM uint32_t TMR0;
00198     __IOM uint32_t TMR1;
00199     __IOM uint32_t TMR2;
00200     __IOM uint32_t TMR3;
00201     __IOM uint32_t TH0;
00202     __IM uint32_t RESERVED1; //
00203     __IOM uint32_t CNT0;
00204     __IM uint32_t RESERVED2; //
00205     __OM uint32_t UPDATE;
00206     __IOM uint32_t FCNT;
00207     __IOM uint32_t EVT0;
00208     __IM uint32_t RESERVED3; //
00209     __IOM uint32_t DTH00;
00210     __IOM uint32_t DTH01;
00211     __IM uint32_t RESERVED4[2]; //
00212     __IOM uint32_t FLT;
00213     __IOM uint32_t SDCFG;
00214     __IOM uint32_t AUEN;
00215     __IOM uint32_t TCLK;
00216     __IOM uint32_t IE0;
00217     __IOM uint32_t IF0;
00218     __IM uint32_t RESERVED5[2]; //
00219     __IOM uint32_t EIE;
00220     __IOM uint32_t EIF;
00221     __IOM uint32_t RE;
00222     __IOM uint32_t PP;
00223     __IOM uint32_t IO01;
00224     __IOM uint32_t IO23;
00225     __IM uint32_t RESERVED6[2]; //
00226     __IOM uint32_t CH_FAIL;
00227     __IM uint32_t RESERVED7; //
00228     __IOM uint32_t CH_DEF;
00229     __IOM uint32_t CH_MSK;
00230     __OM uint32_t PRT;
00231     __IM uint32_t RESERVED8[3]; //
00232     __IOM uint32_t SW_FAIL;
00233 } HAL_MCPWM_TypeDef;
00237 typedef struct
00238 {
00239     __IOM uint32_t CFG;

```

```

00240     __IOM uint32_t TH;
00241     __IOM uint32_t CNT;
00242     __IOM uint32_t IE;
00243     __IOM uint32_t IF;
00244 } HAL_QEP_TypeDef;
00248 typedef struct
00249 {
00250     __IOM uint32_t CFG;
00251     __IOM uint32_t TOSC;
00252     __IOM uint32_t TSTH1;
00253     __IOM uint32_t TDTH1;
00254     __IOM uint32_t IRQ;
00255     __IOM uint32_t WDATA;
00256 } HAL_SIF_TypeDef;
00260 typedef struct
00261 {
00262     __IOM uint32_t CFG;
00263     __IOM uint32_t IE;
00264     __IOM uint32_t DIV;
00265     __IOM uint32_t TX_DATA;
00266     __IOM uint32_t RX_DATA;
00267     __IOM uint32_t SIZE;
00268 } HAL_SPI_TypeDef;
00272 typedef struct
00273 {
00274     __IM uint32_t AFE_ADC;
00275     __IM uint32_t AFE_INFO;
00276     __IM uint32_t AFE_DBG;
00277     __IOM uint32_t OPA_SEL;
00278     __IOM uint32_t AFE_REG0;
00279     __IOM uint32_t AFE_REG1;
00280     __IOM uint32_t AFE_REG2;
00281     __IOM uint32_t AFE_REG3;
00282     __IOM uint32_t AFE_REG4;
00283     __IOM uint32_t AFE_REG5;
00284     __IM uint32_t RESERVED; //
00285     __IOM uint32_t AFE_REG7;
00286     __IM uint32_t RESERVED1[11]; //
00287     __IOM uint32_t AFE_DAC_CTRL;
00288     __IOM uint32_t AFE_DAC0;
00289     __IOM uint32_t AFE_DAC1;
00290     __IOM uint32_t AFE_DAC0_AMC;
00291     __IOM uint32_t AFE_DAC0_DC;
00292     __IOM uint32_t AFE_DAC1_AMC;
00293     __IOM uint32_t AFE_DAC1_DC;
00294     __IM uint32_t RESERVED2[2]; //
00295     __IOM uint32_t CLK_CFG;
00296     __IOM uint32_t IO_CFG;
00297     __IOM uint32_t DBG_CFG;
00298     __IM uint32_t RESERVED3; //
00299     __IOM uint32_t CLK_DIV0;
00300     __IOM uint32_t CLK_DIV1; //
00301     __IOM uint32_t CLK_DIV2;
00302     __IOM uint32_t CLK_FEN;
00303     __IM uint32_t RESERVED41; //
00304     __IOM uint32_t SFT_RST;
00305     __IOM uint32_t PROTECT;
00306     __IM uint32_t RESERVED5; //
00307     __IOM uint32_t NMI_TH;
00308     __IOM uint32_t IF;
00309     __IOM uint32_t RAM_LOCK;
00310     __IM uint32_t RESERVED6; //
00311     __IM uint32_t RESERVED7[4]; //
00312     __IOM uint32_t FLSE;
00313     __IOM uint32_t FLSP;
00314 } HAL_SYS_TypeDef;
00318 typedef struct
00319 {
00320     __IOM uint32_t CFG0;
00321     __IOM uint32_t CFG1;
00322     __IOM uint32_t CFG2;
00323     __IM uint32_t RESERVED; //
00324     __IOM uint32_t TH;
00325     __IOM uint32_t CNT;
00326     __IOM uint32_t CHN0;
00327     __IOM uint32_t CHN1;
00328     __IOM uint32_t EVT;
00329     __IOM uint32_t FLT;
00330     __IOM uint32_t IE;
00331     __IOM uint32_t IF;
00332     __IOM uint32_t IO;
00333 } HAL_TIMER_TypeDef;
00337 typedef struct
00338 {
00339     __IOM uint32_t CTRL;
00340     __IOM uint32_t DIVH;
00341     __IOM uint32_t DIVL;

```

```

00342     __IOM uint32_t  BUF;
00343     __IOM uint32_t  AD;
00344     __IOM uint32_t  ST;
00345     __IOM uint32_t  RE;
00346     __IOM uint32_t  IE;
00347     __IOM uint32_t  IF;
00348     __IOM uint32_t  IOC;
00349 } HAL_UART_TypeDef;
00353 typedef struct
00354 {
00355     __IO uint32_t  PSW;
00356     __IO uint32_t  CFG;
00357     __IO uint32_t  CLR;
00358     __IO uint32_t  WTH;
00359     __IO uint32_t  RTH;
00360     __IO uint32_t  CNT;
00361 } HAL_IWDG_TypeDef;
00362
00366 typedef struct
00367 {
00368     __IOM uint32_t  PWR_CFG;
00369     __IOM uint32_t  EVT_RCD;
00370     __IOM uint32_t  IO_WAKE_POL;
00371     __IOM uint32_t  IO_WAKE_EN;
00372     __IOM uint32_t  BKP_REG0;
00373     __IOM uint32_t  BKP_REG1;
00374 } HAL_AON_TypeDef;
00375
00376 #define CLU0      ((HAL_CLU_TypeDef *) CLU0_BASE)
00377 #define DMA_CHN0  ((HAL_DMA_TypeDef *) DMA0_BASE)
00378 #define DMA_CHN1  ((HAL_DMA_TypeDef *) (DMA0_BASE + 0x20))
00379 #define DMA_CHN2  ((HAL_DMA_TypeDef *) (DMA0_BASE + 0x40))
00380 #define DMA_CHN3  ((HAL_DMA_TypeDef *) (DMA0_BASE + 0x60))
00381 #define DMA_TOP   ((HAL_DMAMODULE_TypeDef *) (DMA0_BASE + 0x80))
00382 #define EXTI0     ((HAL_EXTI_TypeDef *) EXTI_BASE)
00383 #define QEP0      ((HAL_QEP_TypeDef *) QEP0_BASE)
00384 #define SIF0      ((HAL_SIF_TypeDef *) (SIF0_BASE))
00385 #define AON0      ((HAL_AON_TypeDef *) AON_BASE)
00386
00387 extern HAL_SYS_TypeDef *const HAL_SYS[];
00388 extern HAL_FLASH_TypeDef *const HAL_FLASH[];
00389 extern HAL_SPI_TypeDef *const HAL_SPI[];
00390 extern HAL_I2C_TypeDef *const HAL_I2C[];
00391 extern HAL_CMP_TypeDef *const HAL_CMP[];
00392 extern HAL_HALL_TypeDef *const HAL_HALL[];
00393 extern HAL_ADC_TypeDef *const HAL_ADC[];
00394 extern HAL_TIMER_TypeDef *const HAL_TIMER[];
00395 extern HAL_QEP_TypeDef *const HAL_QEP[];
00396 extern HAL_MCPWM_TypeDef *const HAL_MCPWM[];
00397 extern HAL_GPIO_TypeDef *const HAL_GPIO[];
00398 extern HAL_CRC_TypeDef *const HAL_CRC[];
00399 extern HAL_UART_TypeDef *const HAL_UART[];
00400 extern HAL_DSP_TypeDef *const HAL_DSP[];
00401 extern HAL_IWDG_TypeDef *const HAL_IWDG[];
00402 extern HAL_DMA_TypeDef *const HAL_DMA[];
00403 #endif

```

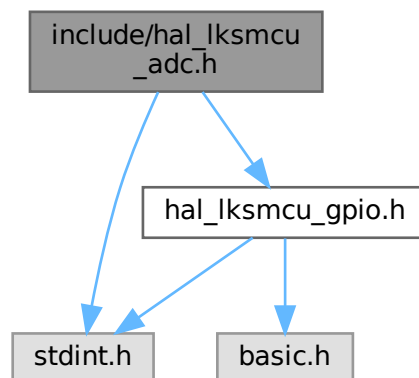
4.9 include/hal_lksmcu_adc.h 文件参考

```

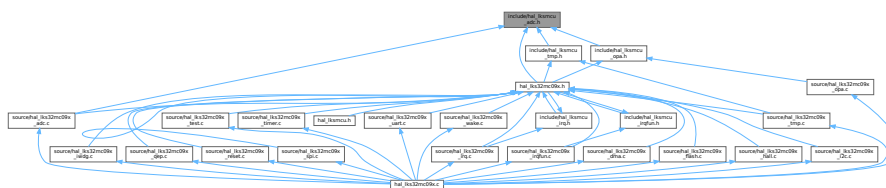
#include "stdint.h"
#include "hal_lksmcu_gpio.h"

```

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:



枚挙

- enum HAL_ADC_PORT_x {
 HAL_ADC_PORT_CH0 , HAL_ADC_PORT_CH1 , HAL_ADC_PORT_CH2 , HAL_ADC_PORT_CH3 ,
 HAL_ADC_PORT_CH4 , HAL_ADC_PORT_CH5 , HAL_ADC_PORT_CH6 , HAL_ADC_PORT_CH7 ,
 HAL_ADC_PORT_CH8 , HAL_ADC_PORT_CH9 , HAL_ADC_PORT_CH10 , HAL_ADC_PORT_CH11 ,
 HAL_ADC_PORT_CH12 , HAL_ADC_PORT_CH13 , HAL_ADC_PORT_CH14 , HAL_ADC_PORT_CH15 ,
 HAL_ADC_PORT_OPA0 = HAL_ADC_PORT_CH0 , HAL_ADC_PORT_OPA1 = HAL_ADC_PORT_CH1 ,
 HAL_ADC_PORT_OPA2 = HAL_ADC_PORT_CH2 , HAL_ADC_PORT_OPA3 = HAL_ADC_PORT_CH3 ,
 HAL_ADC_PORT_REF = HAL_ADC_PORT_CH9 , HAL_ADC_PORT_DAC0 = HAL_ADC_PORT_CH14 ,
 HAL_ADC_PORT_AGND = HAL_ADC_PORT_CH15 , HAL_ADC_PORT_TMP0 = HAL_ADC_PORT_CH14
 ,
 HAL_ADC_PORT_VDD = HAL_ADC_PORT_CH15 , HAL_ADC_PORT_DAC1 , HAL_ADC_PORT_OPA4 ,
 HAL_ADC_PORT_OPA5 ,
 HAL_ADC_PORT_CH16 , HAL_ADC_PORT_CH17 , HAL_ADC_PORT_CH18 , HAL_ADC_PORT_CH19 ,
 HAL_ADC_PORT_LDO12 }
 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_MCPWM , HAL_ADC_TRIGGER_TIMER , HAL_ADC_TRIGGER_SOFT
}
- enum HAL_ADC_REFERENCE_x { HAL_ADC_REFERENCE_INTERNAL , 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_PORT_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
- 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_PORT_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_PORT_x

enum HAL_ADC_PORT_x

主频

枚举值

HAL_ADC_PORT_CH0	ADC采样端口0
HAL_ADC_PORT_CH1	ADC采样端口1
HAL_ADC_PORT_CH2	ADC采样端口2
HAL_ADC_PORT_CH3	ADC采样端口3
HAL_ADC_PORT_CH4	ADC采样端口4
HAL_ADC_PORT_CH5	ADC采样端口5
HAL_ADC_PORT_CH6	ADC采样端口6
HAL_ADC_PORT_CH7	ADC采样端口7
HAL_ADC_PORT_CH8	ADC采样端口8
HAL_ADC_PORT_CH9	ADC采样端口9
HAL_ADC_PORT_CH10	ADC采样端口10
HAL_ADC_PORT_CH11	ADC采样端口11
HAL_ADC_PORT_CH12	ADC采样端口12
HAL_ADC_PORT_CH13	ADC采样端口13
HAL_ADC_PORT_CH14	ADC采样端口14
HAL_ADC_PORT_CH15	ADC采样端口15
HAL_ADC_PORT_OPA0	ADC采样端口运放0
HAL_ADC_PORT_OPA1	ADC采样端口运放1
HAL_ADC_PORT_OPA2	ADC采样端口运放2
HAL_ADC_PORT_OPA3	ADC采样端口运放3
HAL_ADC_PORT_REF	仅ADC0 ADC采样芯片内部基准电压
HAL_ADC_PORT_DAC0	仅ADC0 ADC采样DAC0输出
HAL_ADC_PORT_AGN0	仅ADC0 ADC采样内部地
HAL_ADC_PORT_TMP0	仅ADC1 ADC采样端口温度传感器
HAL_ADC_PORT_VDD	仅ADC1 ADC采样供电
HAL_ADC_PORT_DAC1	ADC采样DAC1输出 不支持 这里保留接口定义
HAL_ADC_PORT_OPA4	ADC采样端口运放4 不支持 这里保留接口定义
HAL_ADC_PORT_OPA5	ADC采样端口运放5 不支持 这里保留接口定义
HAL_ADC_PORT_CH16	ADC采样端口16 不支持 这里保留接口定义
HAL_ADC_PORT_CH17	ADC采样端口17 不支持 这里保留接口定义
HAL_ADC_PORT_CH18	ADC采样端口18 不支持 这里保留接口定义
HAL_ADC_PORT_CH19	ADC采样端口19 不支持 这里保留接口定义
HAL_ADC_PORT_LDO12	ADC采样芯片LDO12(1.2V)内核供电 不支持 这里保留接口定义

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
HAL_ADC_REFERENCE_EXTERNAL	使用芯片供电作为adc基准

在文件 [hal_lksmcu_adc.h](#) 第 137 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 111 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

枚举值

HAL_ADC_SAMP_SEQUENCE_1	ADC第1次采样
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 81 行定义.

4.9.1.5 HAL_ADC_TRIGGER_x

enum [HAL_ADC_TRIGGER_x](#)

枚举值

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

在文件 [hal_lksmcu_adc.h](#) 第 131 行定义.

4.9.1.6 HAL_ADC_x

enum [HAL_ADC_x](#)

枚举值

HAL_↔ ADC_0	ADC模块0
HAL_↔ ADC_1	ADC模块1

枚举值

HAL↔ ADC_2	ADC模块2
HAL↔ ADC_3	ADC模块3

在文件 [hal_lksmcu_adc.h](#) 第 104 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_lks32mc09x_adc.c](#) 第 115 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_lks32mc09x_adc.c](#) 第 87 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

adc_x	ADC模块编号

在文件 [hal_lks32mc09x_adc.c](#) 第 261 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

adc_x	ADC模块编号

返回

ADC量程

在文件 [hal_lks32mc09x_adc.c](#) 第 339 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_TypeDef ADC,
    HAL_ADC_Sequencer_TypeDef Sequence )
```

获取指定序列号的采样值

参数

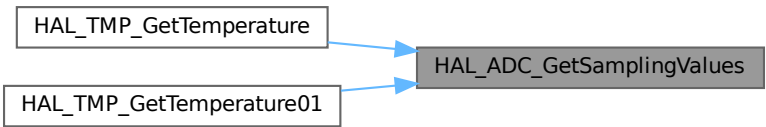
ADC	ADC模块编号
Sequence	采样序列号

返回

返回对应序列号的采样值

在文件 `hal_lks32mc09x_adc.c` 第 149 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_TypeDef ADC )
```

获取采样值单位

参数

ADC	ADC模块编号

返回

采样值单位

在文件 `hal_lks32mc09x_adc.c` 第 363 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_PORT_x port_x )
```

手动采样

参数

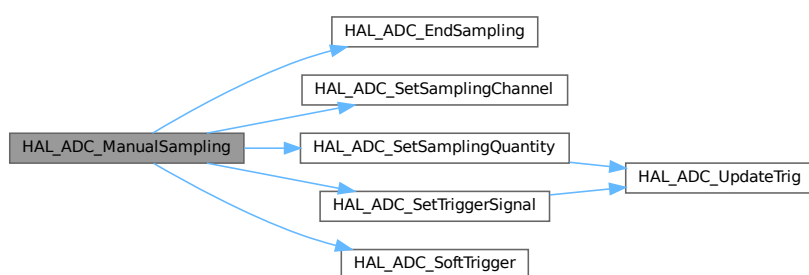
<i>n</i>	采样通道编号

返回

示例返回值,实际应返回采样值

在文件 [hal_lks32mc09x_adc.c](#) 第 279 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_lks32mc09x_adc.c](#) 第 62 行定义.

4.9.2.9 HAL_ADC_SetGpioIn()

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

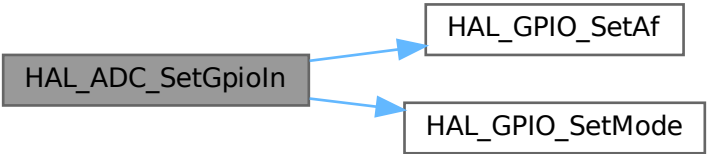
设置指定的GPIO为ADC输入

参数

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

在文件 [hal_lks32mc09x_adc.c](#) 第 380 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 )
```

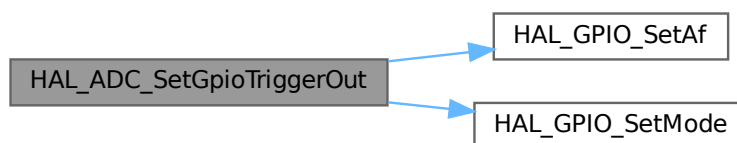
设置指定的GPIO为ADC触发信号输出

参数

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

在文件 [hal_lks32mc09x_adc.c](#) 第 391 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_lks32mc09x_adc.c](#) 第 309 行定义.

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_PORT_x port_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>port_x</i>	采样通道端口

< ADC第0次采样

< ADC第1次采样

< ADC第2次采样

< ADC第3次采样

< ADC第4次采样

< ADC第5次采样

< ADC第6次采样

< ADC第7次采样

< ADC第8次采样

< ADC第9次采样

< ADC第10次采样

< ADC第11次采样

在文件 [hal_lks32mc09x_adc.c](#) 第 160 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_lks32mc09x_adc.c](#) 第 210 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_lks32mc09x_adc.c](#) 第 240 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

<i>adc_x</i>	ADC模块编号

在文件 [hal_lks32mc09x_adc.c](#) 第 270 行定义.

这是这个函数的调用关系图:



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     // ADC0采样通道
00009     // 0x0: OPA0 输出
00010     // 0x1: OPA1 输出
00011     // 0x2: OPA2 输出
00012     // 0x3: OPA3 输出
00013     // 0x4: ADC01.CH4
00014     // 0x5: ADC01.CH5
00015     // 0x6: ADC01.CH6
00016     // 0x7: ADC01.CH7
00017     // 0x8: ADC0.CH8
00018     // 0x9: REF24
00019     // 0xA: ADC01.CH10
00020     // 0xB: ADC01.CH11
00021     // 0xC: ADC0.CH12
00022     // 0xD: ADC0.CH13
00023     // 0xE: DAC0 输出
00024     // 0xF: 内部地AVSS
00025
00026     // ADC1采样通道
00027     // 0x0: OPA0输出
00028     // 0x1: OPA1输出
00029     // 0x2: OPA2输出
00030     // 0x3: OPA3输出
00031     // 0x4: ADC01.CH4
00032     // 0x5: ADC01.CH5
00033     // 0x6: ADC01.CH6
00034     // 0x7: ADC01.CH7
00035     // 0x8: ADC1.CH8
00036     // 0x9: ADC1.CH9
00037     // 0xA: ADC01.CH10
00038     // 0xB: ADC01.CH11
00039     // 0xC: ADC1.CH12
00040     // 0xD: ADC1.CH13
00041     // 0xE: 温度传感器
00042     // 0xF: 电源AVDD
00043
00044     HAL_ADC_PORT.CH0,
00045     HAL_ADC_PORT.CH1,
00046     HAL_ADC_PORT.CH2,
00047     HAL_ADC_PORT.CH3,
00048     HAL_ADC_PORT.CH4,
00049     HAL_ADC_PORT.CH5,
00050     HAL_ADC_PORT.CH6,
00051     HAL_ADC_PORT.CH7,
00052     HAL_ADC_PORT.CH8,
00053     HAL_ADC_PORT.CH9,
00054     HAL_ADC_PORT.CH10,
00055     HAL_ADC_PORT.CH11,
00056     HAL_ADC_PORT.CH12,
00057     HAL_ADC_PORT.CH13,
00058     HAL_ADC_PORT.CH14,
00059     HAL_ADC_PORT.CH15,
00060
00061     HAL_ADC_PORT.OPA0 = HAL_ADC_PORT.CH0,
00062     HAL_ADC_PORT.OPA1 = HAL_ADC_PORT.CH1,
00063     HAL_ADC_PORT.OPA2 = HAL_ADC_PORT.CH2,
00064     HAL_ADC_PORT.OPA3 = HAL_ADC_PORT.CH3,
00065     HAL_ADC_PORT.REF = HAL_ADC_PORT.CH9,
00066     HAL_ADC_PORT.DAC0 = HAL_ADC_PORT.CH14,
00067     HAL_ADC_PORT.AGND = HAL_ADC_PORT.CH15,
00068     HAL_ADC_PORT.TMP0 = HAL_ADC_PORT.CH14,
00069     HAL_ADC_PORT.VDD = HAL_ADC_PORT.CH15,
00070
00071     HAL_ADC_PORT.DAC1,
00072     HAL_ADC_PORT.OPA4,
00073     HAL_ADC_PORT.OPA5,
00074     HAL_ADC_PORT.CH16,
00075     HAL_ADC_PORT.CH17,
00076     HAL_ADC_PORT.CH18,
00077     HAL_ADC_PORT.CH19,
00078     HAL_ADC_PORT.LDO12,
00079 } HAL_ADC_PORT_x;
00081 typedef enum
00082 {
00083     HAL_ADC_SAMP.SEQUENCE_0,
00084     HAL_ADC_SAMP.SEQUENCE_1,
00085     HAL_ADC_SAMP.SEQUENCE_2,
00086     HAL_ADC_SAMP.SEQUENCE_3,
00087     HAL_ADC_SAMP.SEQUENCE_4,
00088     HAL_ADC_SAMP.SEQUENCE_5,
00089     HAL_ADC_SAMP.SEQUENCE_6,

```

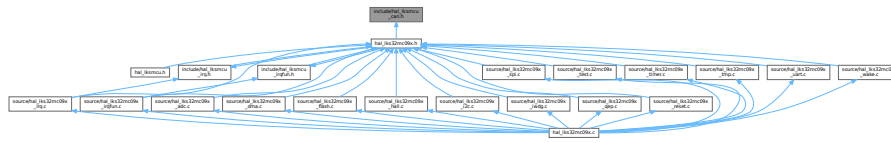
```

00090     HAL_ADC_SAMP_SEQUENCE_7,
00091     HAL_ADC_SAMP_SEQUENCE_8,
00092     HAL_ADC_SAMP_SEQUENCE_9,
00093     HAL_ADC_SAMP_SEQUENCE_10,
00094     HAL_ADC_SAMP_SEQUENCE_11,
00095     HAL_ADC_SAMP_SEQUENCE_12,
00096     HAL_ADC_SAMP_SEQUENCE_13,
00097     HAL_ADC_SAMP_SEQUENCE_14,
00098     HAL_ADC_SAMP_SEQUENCE_15,
00099     HAL_ADC_SAMP_SEQUENCE_16,
00100     HAL_ADC_SAMP_SEQUENCE_17,
00101     HAL_ADC_SAMP_SEQUENCE_18,
00102     HAL_ADC_SAMP_SEQUENCE_19,
00103 } HAL_ADC_SAMP_SEQUENCE_x;
00104 typedef enum
00105 {
00106     HAL_ADC_0,
00107     HAL_ADC_1,
00108     HAL_ADC_2,
00109     HAL_ADC_3,
00110 } HAL_ADC_x;
00111 typedef enum
00112 {
00113     HAL_ADC_SAMP_NUMBER_0,
00114     HAL_ADC_SAMP_NUMBER_1,
00115     HAL_ADC_SAMP_NUMBER_2,
00116     HAL_ADC_SAMP_NUMBER_3,
00117     HAL_ADC_SAMP_NUMBER_4,
00118     HAL_ADC_SAMP_NUMBER_5,
00119     HAL_ADC_SAMP_NUMBER_6,
00120     HAL_ADC_SAMP_NUMBER_7,
00121     HAL_ADC_SAMP_NUMBER_8,
00122     HAL_ADC_SAMP_NUMBER_9,
00123     HAL_ADC_SAMP_NUMBER_10,
00124     HAL_ADC_SAMP_NUMBER_11,
00125     HAL_ADC_SAMP_NUMBER_12,
00126     HAL_ADC_SAMP_NUMBER_13,
00127     HAL_ADC_SAMP_NUMBER_14,
00128     HAL_ADC_SAMP_NUMBER_15,
00129     HAL_ADC_SAMP_NUMBER_MAX = HAL_ADC_SAMP_NUMBER_15,
00130 } HAL_ADC_SAMP_NUMBER_x;
00131 typedef enum
00132 {
00133     HAL_ADC_TRIGGER_MCPWM,
00134     HAL_ADC_TRIGGER_TIMER,
00135     HAL_ADC_TRIGGER_SOFT,
00136 } HAL_ADC_TRIGGER_x;
00137 typedef enum
00138 {
00139     HAL_ADC_REFERENCE_INTERNAL,
00140     HAL_ADC_REFERENCE_EXTERNAL,
00141 } HAL_ADC_REFERENCE_x;
00142
00143 // 初始化
00144 void HAL_ADC_Reset(HAL_ADC_x); // 模块复位
00145 void HAL_ADC_Enable(HAL_ADC_x); // 模块打开
00146 void HAL_ADC_Disable(HAL_ADC_x); // 模块关闭
00147
00148 // 自动采样配置
00149 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
00150 void HAL_ADC_SetSamplingChannel(HAL_ADC_x, HAL_ADC_SAMP_SEQUENCE_x, HAL_ADC_PORT_x); // 设置采样通道, 输
    入参数 第n次采样channel通道
00151 void HAL_ADC_SetSamplingQuantity(HAL_ADC_x, HAL_ADC_SAMP_NUMBER_x, HAL_ADC_SAMP_NUMBER_x); // 设置采样数量
00152 void HAL_ADC_SetTriggerSignal(HAL_ADC_x, HAL_ADC_TRIGGER_x); // 设置触发信号
00153 void HAL_ADC_EndSampling(HAL_ADC_x); // 中止当前采
    样
00154 void HAL_ADC_SetReference(HAL_ADC_x, HAL_ADC_REFERENCE_x); // 设置ADC基准
    源
00155 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
00156 float HAL_ADC_GetValuesUnit(HAL_ADC_x); // 获取采样值
    单位
00157 // 手动采样配置
00158 void HAL_ADC_SoftTrigger(HAL_ADC_x); // 软件触发ADC开始采样
00159 int16_t HAL_ADC_ManualSampling(HAL_ADC_x, HAL_ADC_PORT_x); // 手动采样 需要先设置采样通道为软件触发
00160
00161 // IO配置
00162 void HAL_ADC_SetGpioIn(HAL_ADC_x, HAL_GPIO_Px); // 设置指定的GPIO为ADC输入
00163 void HAL_ADC_SetGpioTriggerOut(HAL_ADC_x, HAL_GPIO_Px); // 设置指定的GPIO为ADC触发信号输出
00164
00165 #endif

```

4.11 include/hal_lksmcu_can.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:



4.12 hal_ksmcu_can.h

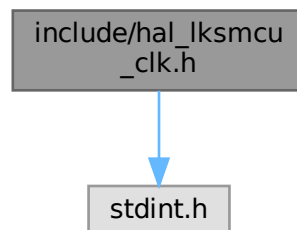
[浏览该文件的文档.](#)

```
00001 #ifndef __HAL_LKSMCU_CAN_H__
00002 #define __HAL_LKSMCU_CAN_H__
00003 #endif
```

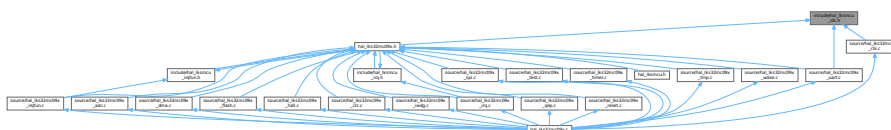
4.13 include/hal_lksmcu_clk.h 文件参考

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

hal_ksmcu_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_8MHZ , HAL_CLK_FREQ_RC_9MHZ , HAL_CLK_FREQ_RC_12MHZ , HAL_CLK_FREQ_RC_16MHZ
    ,
    HAL_CLK_FREQ_RC_18MHZ , HAL_CLK_FREQ_RC_24MHZ , HAL_CLK_FREQ_RC_32MHZ , HAL_CLK_FREQ_RC_36MHZ
    ,
    HAL_CLK_FREQ_RC_48MHZ , HAL_CLK_FREQ_RC_64MHZ , HAL_CLK_FREQ_RC_72MHZ , HAL_CLK_FREQ_RC_96MHZ
    ,
    HAL_CLK_FREQ_XTAL_6MHZ , HAL_CLK_FREQ_XTAL_12MHZ , HAL_CLK_FREQ_XTAL_24MHZ ,
    HAL_CLK_FREQ_XTAL_48MHZ ,
    HAL_CLK_FREQ_XTAL_96MHZ , HAL_CLK_FREQ_RC_4MHZ , HAL_CLK_FREQ_RC_6MHZ , HAL_CLK_FREQ_RC_192MHZ
    ,
    HAL_CLK_FREQ_XTAL_192MHZ }
    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_8MHZ	主频8Mhz
HAL_CLK_FREQ_RC_9MHZ	主频9Mhz
HAL_CLK_FREQ_RC_12MHZ	主频12Mhz
HAL_CLK_FREQ_RC_16MHZ	主频16Mhz
HAL_CLK_FREQ_RC_18MHZ	主频18Mhz
HAL_CLK_FREQ_RC_24MHZ	主频24Mhz
HAL_CLK_FREQ_RC_32MHZ	主频32Mhz
HAL_CLK_FREQ_RC_36MHZ	主频36Mhz
HAL_CLK_FREQ_RC_48MHZ	主频48Mhz
HAL_CLK_FREQ_RC_64MHZ	主频64Mhz
HAL_CLK_FREQ_RC_72MHZ	主频72Mhz
HAL_CLK_FREQ_RC_96MHZ	主频96Mhz
HAL_CLK_FREQ_XTAL_6MHZ	主频6Mhz 不支持 这里保留接口定义
HAL_CLK_FREQ_XTAL_12MHZ	主频12Mhz 不支持 这里保留接口定义
HAL_CLK_FREQ_XTAL_24MHZ	主频24Mhz 不支持 这里保留接口定义
HAL_CLK_FREQ_XTAL_48MHZ	主频48Mhz 不支持 这里保留接口定义

枚举值

HAL_CLK_FREQ_XTAL_96MHZ	主频96Mhz 不支持 这里保留接口定义
HAL_CLK_FREQ_RC_4MHZ	主频4Mhz 不支持 这里保留接口定义
HAL_CLK_FREQ_RC_6MHZ	主频6Mhz 不支持 这里保留接口定义
HAL_CLK_FREQ_RC_192MHZ	主频192Mhz 不支持 这里保留接口定义
HAL_CLK_FREQ_XTAL_192MHZ	主频192Mhz 不支持 这里保留接口定义

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 )
```

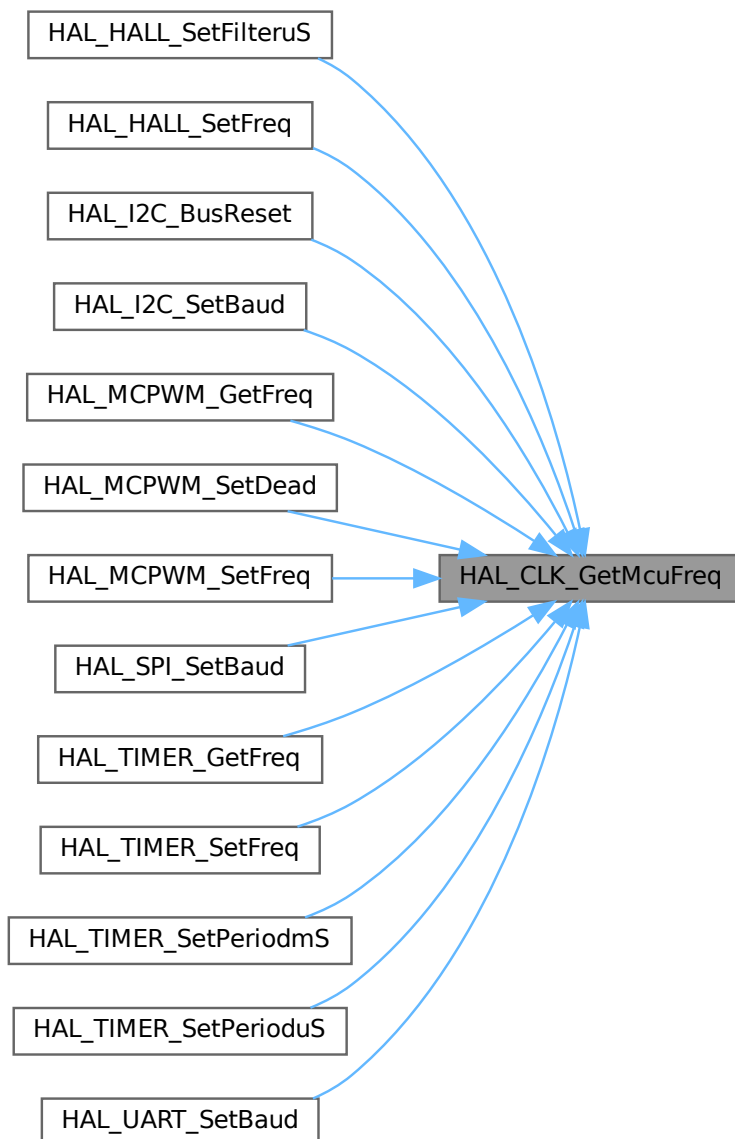
获取主频

返回

主频

在文件 [hal_lks32mc09x.clk.c](#) 第 90 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</i> _x	主频

< 主频4Mhz

< 主频12Mhz

< 主频24Mhz

< 主频48Mhz

< 主频96Mhz

< 主频18Mhz

< 主频36Mhz

< 主频72Mhz

< 主频16Mhz

< 主频32Mhz

< 主频64Mhz

< 主频96Mhz

在文件 [hal_lks32mc09x_clk.c](#) 第 11 行定义.

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_8MHZ,
00009     HAL_CLK_FREQ_RC_9MHZ,
00010     HAL_CLK_FREQ_RC_12MHZ,
00011     HAL_CLK_FREQ_RC_16MHZ,
00012     HAL_CLK_FREQ_RC_18MHZ,
00013     HAL_CLK_FREQ_RC_24MHZ,
00014     HAL_CLK_FREQ_RC_32MHZ,
00015     HAL_CLK_FREQ_RC_36MHZ,
00016     HAL_CLK_FREQ_RC_48MHZ,
00017     HAL_CLK_FREQ_RC_64MHZ,
00018     HAL_CLK_FREQ_RC_72MHZ,
00019     HAL_CLK_FREQ_RC_96MHZ,
00020
00021     HAL_CLK_FREQ_XTAL_6MHZ,
00022     HAL_CLK_FREQ_XTAL_12MHZ,
00023     HAL_CLK_FREQ_XTAL_24MHZ,
00024     HAL_CLK_FREQ_XTAL_48MHZ,
00025     HAL_CLK_FREQ_XTAL_96MHZ,
00026
00027     HAL_CLK_FREQ_RC_4MHZ,
00028     HAL_CLK_FREQ_RC_6MHZ,
00029     HAL_CLK_FREQ_RC_192MHZ,
00030     HAL_CLK_FREQ_XTAL_192MHZ,
00031 } HAL_CLK_FREQ_x;
00032 void HAL_CLK_SetMcuFreq(HAL_CLK_FREQ_x);
00033 uint32_t HAL_CLK_GetMcuFreq(void);
00034 #endif

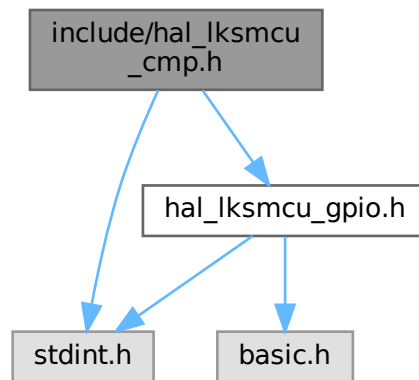
```

4.15 include/hal_lksmcu_cmp.h 文件参考

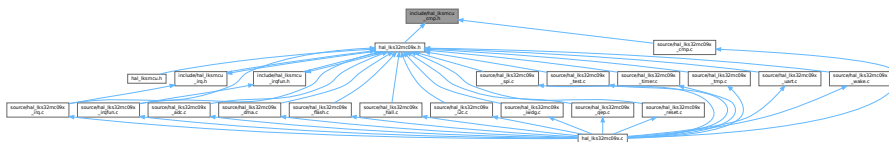
```
#include "stdint.h"
```

```
#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 = 4, HAL_CMP_CHANNEL_IP1 = 5, HAL_CMP_CHANNEL_IP2 = 6,
HAL_CMP_CHANNEL_IP3 = 7,
HAL_CMP_CHANNEL_IP4 = 8, HAL_CMP_CHANNEL_IP5 = 11, HAL_CMP_CHANNEL_IP6 = 12,
HAL_CMP_CHANNEL_IP7 = 13,
HAL_CMP_CHANNEL_IP_OPA0_IP = 0, HAL_CMP_CHANNEL_IP_OPA1_IP = 1, HAL_CMP_CHANNEL_IP_OPA2_IP
= 0, HAL_CMP_CHANNEL_IP_OPA3_IP = 1,
HAL_CMP_CHANNEL_IP_OPA0_OUTP = 2, HAL_CMP_CHANNEL_IP_OPA1_OUTP = 3, HAL_CMP_CHANNEL_IP_OPA2_OUTP
= 2, HAL_CMP_CHANNEL_IP_OPA3_OUTP = 3,
HAL_CMP_CHANNEL_IP_DAC0 = 9, HAL_CMP_CHANNEL_IP_DAC1 = 9, HAL_CMP_CHANNEL_IP_OPA_OUT_CMP0
= 10, HAL_CMP_CHANNEL_IP_AVSS,
HAL_CMP_CHANNEL_IP_OPA0.IN, HAL_CMP_CHANNEL_IP_OPA0.OUTN, HAL_CMP_CHANNEL_IP_OPA1.IN
, HAL_CMP_CHANNEL_IP_OPA1.OUTN,
HAL_CMP_CHANNEL_IP_OPA2.IN, HAL_CMP_CHANNEL_IP_OPA2.OUTN, HAL_CMP_CHANNEL_IP_OPA3.IN

```
, HAL_CMP_CHANNEL_IP_OPA3_OUTN ,
HAL_CMP_CHANNEL_IP_OPA4_IP , HAL_CMP_CHANNEL_IP_OPA4_IN , HAL_CMP_CHANNEL_IP_OPA4_OUTP
, HAL_CMP_CHANNEL_IP_OPA4_OUTN ,
HAL_CMP_CHANNEL_IP_OPA5_IP , HAL_CMP_CHANNEL_IP_OPA5_IN , HAL_CMP_CHANNEL_IP_OPA5_OUTP
, HAL_CMP_CHANNEL_IP_OPA5_OUTN }
• enum HAL_CMP_CHANNEL_INx { HAL_CMP_CHANNEL_IN_0 , HAL_CMP_CHANNEL_IN_REF , HAL_CMP_CHANNEL_IN_DAC
, HAL_CMP_CHANNEL_IN_MID }
```

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_SetGpioOut \(HAL_CMP_x, HAL_GPIO_Px i2cgpio\)](#)
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_DAC	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

在文件 [hal_lksmcu_cmp.h](#) 第 62 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_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_IP5	CMP模块外部正端输入5
HAL_CMP_CHANNEL_IP6	CMP模块外部正端输入6
HAL_CMP_CHANNEL_IP7	CMP模块外部正端输入6
HAL_CMP_CHANNEL_IP_OPA0_IP	仅CMP0 运放0正端输入
HAL_CMP_CHANNEL_IP_OPA1_IP	仅CMP0 运放1正端输入
HAL_CMP_CHANNEL_IP_OPA2_IP	仅CMP1 运放2正端输入
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_OPA1_OUTP	仅CMP0 运放1正端输出
HAL_CMP_CHANNEL_IP_OPA2_OUTP	仅CMP1 运放2正端输出
HAL_CMP_CHANNEL_IP_OPA3_OUTP	仅CMP1 运放3正端输出
HAL_CMP_CHANNEL_IP_DAC0	仅CMP0 DAC0输出
HAL_CMP_CHANNEL_IP_DAC1	仅CMP1 DAC1输出
HAL_CMP_CHANNEL_IP_OPA_OUT_CMP0	可以软件配置的运放输出0
HAL_CMP_CHANNEL_IP_AVSS	模拟地 不支持
HAL_CMP_CHANNEL_IP_OPA0.IN	运放0负端输入 不支持
HAL_CMP_CHANNEL_IP_OPA0.OUTN	运放0负端输出 不支持
HAL_CMP_CHANNEL_IP_OPA1.IN	运放1负端输入 不支持
HAL_CMP_CHANNEL_IP_OPA1.OUTN	运放1负端输出 不支持
HAL_CMP_CHANNEL_IP_OPA2.IN	运放2负端输入 不支持
HAL_CMP_CHANNEL_IP_OPA2.OUTN	运放2负端输出 不支持
HAL_CMP_CHANNEL_IP_OPA3.IN	运放3负端输入 不支持
HAL_CMP_CHANNEL_IP_OPA3.OUTN	运放3负端输出 不支持
HAL_CMP_CHANNEL_IP_OPA4_IP	运放4正端输入 不支持
HAL_CMP_CHANNEL_IP_OPA4.IN	运放4负端输入 不支持
HAL_CMP_CHANNEL_IP_OPA4_OUTP	运放4正端输出 不支持
HAL_CMP_CHANNEL_IP_OPA4.OUTN	运放4负端输出 不支持
HAL_CMP_CHANNEL_IP_OPA5_IP	运放4正端输入 不支持
HAL_CMP_CHANNEL_IP_OPA5.IN	运放5负端输入 不支持
HAL_CMP_CHANNEL_IP_OPA5_OUTP	运放5正端输出 不支持
HAL_CMP_CHANNEL_IP_OPA5.OUTN	运放5负端输出 不支持

在文件 [hal_lksmcu_cmp.h](#) 第 15 行定义。

4.15.1.3 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()

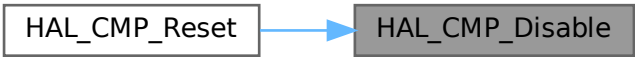
```
void HAL_CMP_Disable (
    HAL_CMP_x cmp_x )
```

模块关闭

参数

cmp↔ _x	使用的CMP模块编号

在文件 [hal_lks32mc09x_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_lks32mc09x_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_lks32mc09x_cmp.c](#) 第 174 行定义.

4.15.2.4 HAL_CMP_Reset()

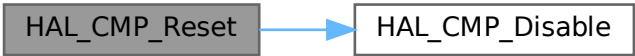
```
void HAL_CMP_Reset (
    HAL_CMP_x cmp_x )
```

模块复位

参数

<i>cmp</i> _↔ <i>_x</i>	使用的CMP模块编号

在文件 [hal_lks32mc09x_cmp.c](#) 第 23 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_lks32mc09x_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_lks32mc09x_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 )
```

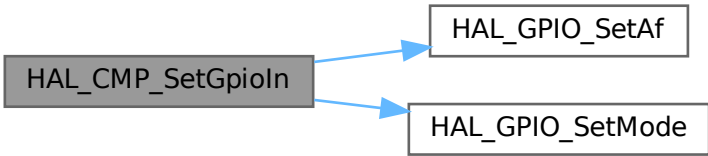
设置CMP模块的GPIO

参数

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

在文件 [hal_lks32mc09x_cmp.c](#) 第 162 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 )
```

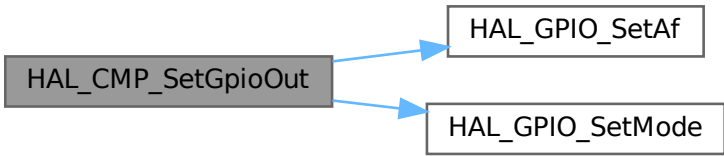
设置CMP模块的GPIO

参数

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

在文件 [hal_lks32mc09x_cmp.c](#) 第 150 行定义.

函数调用图:



4.16 hal_lksmcu_cmp.h

浏览该文件的文档.

```

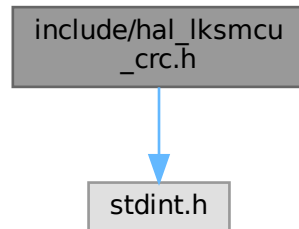
00001 #ifndef __HAL_LKSMCU_CMP_H__
00002 #define __HAL_LKSMCU_CMP_H__
00003 #include "stdint.h"
00004 #include "hal_lksmcu_gpio.h"
00005 typedef enum
00006 {
00007     HAL_CMP_0,
00008     HAL_CMP_1,
00009     HAL_CMP_2,
00010     HAL_CMP_3,
00011     HAL_CMP_4,
00012     HAL_CMP_5,
00013 } HAL_CMP_x;
00014
00015 typedef enum
00016 {
00017     // 比较器1正端信号选择
00018     // 1001: DAC1.OUT
00019     // 1010: OPA.OUT_CMP
00020     // CMP0
00021     // 1001: DAC0.OUT
00022     // 1010: OUT.OPA_CMP
00023     HAL_CMP_CHANNEL_IP0 = 4,
00024     HAL_CMP_CHANNEL_IP1 = 5,
00025     HAL_CMP_CHANNEL_IP2 = 6,
00026     HAL_CMP_CHANNEL_IP3 = 7,
00027     HAL_CMP_CHANNEL_IP4 = 8,
00028     HAL_CMP_CHANNEL_IP5 = 11,
00029     HAL_CMP_CHANNEL_IP6 = 12,
00030     HAL_CMP_CHANNEL_IP7 = 13,
00031     HAL_CMP_CHANNEL_IP.OPA0.IP = 0,
00032     HAL_CMP_CHANNEL_IP.OPA1.IP = 1,
00033     HAL_CMP_CHANNEL_IP.OPA2.IP = 0,
00034     HAL_CMP_CHANNEL_IP.OPA3.IP = 1,
00035     HAL_CMP_CHANNEL_IP.OPA0.OUTP = 2,
00036     HAL_CMP_CHANNEL_IP.OPA1.OUTP = 3,
00037     HAL_CMP_CHANNEL_IP.OPA2.OUTP = 2,
00038     HAL_CMP_CHANNEL_IP.OPA3.OUTP = 3,
00039     HAL_CMP_CHANNEL_IP.DAC0 = 9,
00040     HAL_CMP_CHANNEL_IP.DAC1 = 9,
00041     HAL_CMP_CHANNEL_IP.OPA.OUT_CMP0 = 10,
00042
00043     HAL_CMP_CHANNEL_IP.AVSS,
00044     HAL_CMP_CHANNEL_IP.OPA0.IN,
00045     HAL_CMP_CHANNEL_IP.OPA0.OUTN,
00046     HAL_CMP_CHANNEL_IP.OPA1.IN,
00047     HAL_CMP_CHANNEL_IP.OPA1.OUTN,
00048     HAL_CMP_CHANNEL_IP.OPA2.IN,
00049     HAL_CMP_CHANNEL_IP.OPA2.OUTN,
00050     HAL_CMP_CHANNEL_IP.OPA3.IN,
00051     HAL_CMP_CHANNEL_IP.OPA3.OUTN,
00052     HAL_CMP_CHANNEL_IP.OPA4.IP,
00053     HAL_CMP_CHANNEL_IP.OPA4.IN,
00054     HAL_CMP_CHANNEL_IP.OPA4.OUTP,
00055     HAL_CMP_CHANNEL_IP.OPA4.OUTN,
00056     HAL_CMP_CHANNEL_IP.OPA5.IP,
00057     HAL_CMP_CHANNEL_IP.OPA5.IN,
00058     HAL_CMP_CHANNEL_IP.OPA5.OUTP,
00059     HAL_CMP_CHANNEL_IP.OPA5.OUTN,
00060 } HAL_CMP_CHANNEL_IPx;
00061
00062 typedef enum
00063 {
00064     HAL_CMP_CHANNEL_IN_0,
00065     HAL_CMP_CHANNEL_IN_REF,
00066     HAL_CMP_CHANNEL_IN_DAC,
00067     HAL_CMP_CHANNEL_IN_MID,
00068 } HAL_CMP_CHANNEL_INx;
00069 // 初始化
00070 void HAL_CMP_Reset(HAL_CMP_x); // 模块复位
00071 void HAL_CMP_Enable(HAL_CMP_x); // 模块打开
00072 void HAL_CMP_Disable(HAL_CMP_x); // 模块关闭
00073
00074 void HAL_CMP_SetChannel(HAL_CMP_x, HAL_CMP_CHANNEL_IPx, HAL_CMP_CHANNEL_INx); // 设置CMP输入信号通道
00075 uint32_t HAL_CMP_SetDigitalFilter(HAL_CMP_x, uint32_t filter); // 设置数字滤波宽度 0-0xffff
00076 // 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
00077 void HAL_CMP_SetGpioOut(HAL_CMP_x, HAL_GPIO_Px i2cgpio);
00078 void HAL_CMP_SetGpioIn(HAL_CMP_x, HAL_GPIO_Px cmpgpio);
00079 uint8_t HAL_CMP_GetData(HAL_CMP_x);
00080 #endif

```

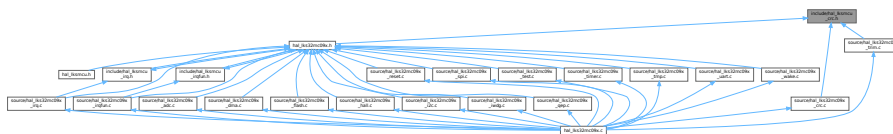
4.17 include/hal_lksmcu_crc.h 文件参考

```
#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

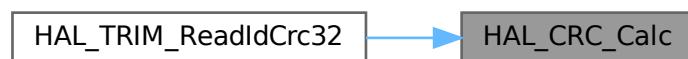
<i>CRC</i> ↔ <i>_x</i>	CRC模块编号
<i>model</i>	CRC模型
<i>data</i>	数据地址
<i>len</i>	数据长度

在文件 [hal_lks32mc09x_crc.c](#) 第 96 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

<i>CRC</i> ↔ <i>_x</i>	CRC模块编号

在文件 [hal_lks32mc09x_crc.c](#) 第 82 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

$CRC \leftrightarrow$ _x	CRC模块编号

在文件 [hal_lks32mc09x_crc.c](#) 第 71 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

$CRC \leftrightarrow$ _x	CRC模块编号

在文件 [hal_lks32mc09x_crc.c](#) 第 60 行定义.

4.18 hal_lksmcu_crc.h

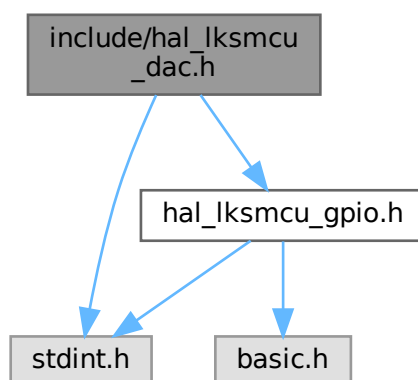
[浏览该文件的文档.](#)

```
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
```

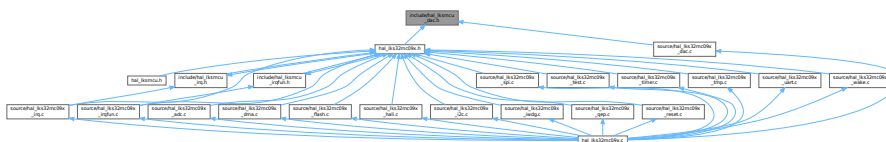
```
00029 void HAL_CRC_Reset(HAL_CRC_t x); // 模块复位
00030 void HAL_CRC_Enable(HAL_CRC_t x); // 模块打开
00031 void HAL_CRC_Disable(HAL_CRC_t x); // 模块关闭
00032 // 计算
00033 uint32_t HAL_CRC_Calc(HAL_CRC_t x, HAL_CRC_MODEL_t x, uint8_t *data, uint32_t len); // 计算crc,03使用软件实现,其余芯片使用硬件实现
00034 #endif
```

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 , HAL_DAC_1 , HAL_DAC_2 , HAL_DAC_3 }

函数

- void HAL_DAC_Reset (HAL_DAC_x)
 - void HAL_DAC_Enable (HAL_DAC_x)
 - void HAL_DAC_Disable (HAL_DAC_x)
 - void HAL_DAC_SetOutVal (HAL_DAC_x, uint16_t val)
 - void HAL_DAC_SetOutVoltageMv (HAL_DAC_x, uint16_t vol_mv)
 - void HAL_DAC_Out2GpioEnable (HAL_DAC_x)
 - void HAL_DAC_Out2GpioDisable (HAL_DAC_x)
 - 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_x

enum [HAL_DAC_x](#)

枚举值

HAL_↔ DAC_0	DAC模块0
HAL_↔ DAC_1	DAC模块1
HAL_↔ DAC_2	DAC模块2
HAL_↔ DAC_3	DAC模块3

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 )
```

在文件 [hal_lks32mc09x_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 )
```

在文件 [hal_lks32mc09x_dac.c](#) 第 13 行定义.

4.19.2.3 HAL_DAC_Out2GpioDisable()

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

在文件 [hal_lks32mc09x.dac.c](#) 第 133 行定义.

4.19.2.4 HAL_DAC_Out2GpioEnable()

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

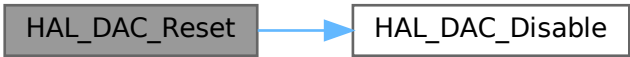
在文件 [hal_lks32mc09x.dac.c](#) 第 115 行定义.

4.19.2.5 HAL_DAC_Reset()

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

在文件 [hal_lks32mc09x.dac.c](#) 第 8 行定义.

函数调用图:



4.19.2.6 HAL_DAC_SetGpioOut()

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

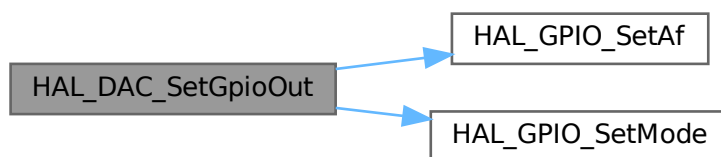
设置DAC输出到GPIO

参数

<i>dac</i>	DAC模块
<i>px</i>	GPIO

在文件 [hal_lks32mc09x.dac.c](#) 第 157 行定义.

函数调用图:



4.19.2.7 HAL_DAC_SetOutVal()

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

在文件 [hal_lks32mc09x_dac.c](#) 第 51 行定义.

这是这个函数的调用关系图:



4.19.2.8 HAL_DAC_SetOutVoltageM()

```
void HAL_DAC_SetOutVoltageM (
    HAL_DAC_TypeDef dac_x,
    uint16_t vol_mV )
```

在文件 [hal_lks32mc09x_dac.c](#) 第 65 行定义.

函数调用图:



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     HAL_DAC_1,
00010     HAL_DAC_2,
00011     HAL_DAC_3,
00012 } HAL_DAC_x;
00013 void HAL_DAC_Reset (HAL_DAC_x); // 模块复位
00014 void HAL_DAC_Enable (HAL_DAC_x); // 模块打开
00015 void HAL_DAC_Disable (HAL_DAC_x); // 模块关闭
00016
00017 void HAL_DAC_SetOutVal (HAL_DAC_x, uint16_t val);
00018 void HAL_DAC_SetOutVoltageMv (HAL_DAC_x, uint16_t vol_mv);
00019 void HAL_DAC_Out2GpioEnable (HAL_DAC_x);
00020 void HAL_DAC_Out2GpioDisable (HAL_DAC_x);
00021 void HAL_DAC_SetGpioOut (HAL_DAC_x, HAL_GPIO_Px);
00022 #endif

```

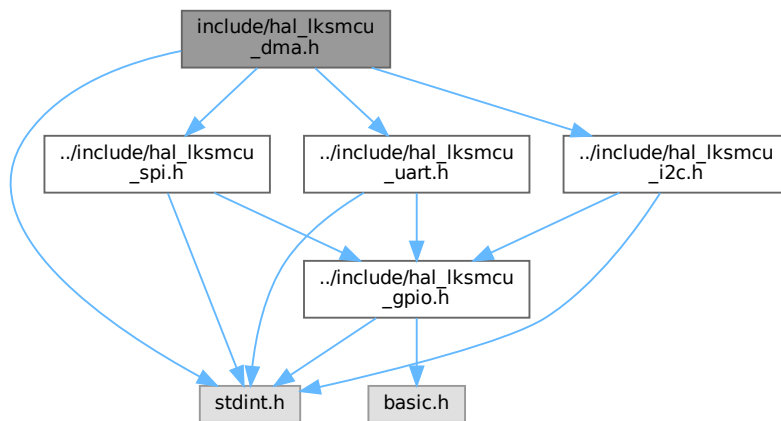
4.21 include/hal_lksmcu_dma.h 文件参考

```

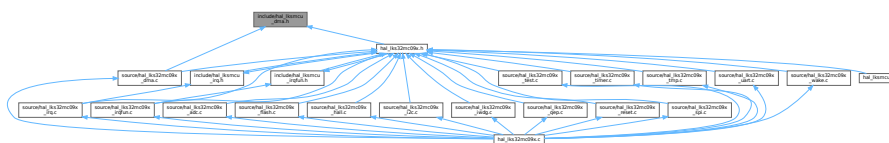
#include "../include/hal_lksmcu_spi.h"
#include "../include/hal_lksmcu_uart.h"
#include "../include/hal_lksmcu_i2c.h"
#include "stdint.h"

```

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_SpiRxCtx` (`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

<code>HAL_DMA_0</code>	DMA模块0
<code>HAL_DMA_1</code>	DMA模块1
<code>HAL_DMA_2</code>	DMA模块2
<code>HAL_DMA_3</code>	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_lks32mc09x_dma.c` 第 29 行定义.

4.21.2.2 HAL_DMA_Enable()

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

在文件 [hal_lks32mc09x_dma.c](#) 第 22 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_lks32mc09x_dma.c](#) 第 104 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_lks32mc09x_dma.c](#) 第 74 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_lks32mc09x_dma.c](#) 第 39 行定义.

4.21.2.6 HAL_DMA_Reset()

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

在文件 [hal_lks32mc09x_dma.c](#) 第 10 行定义.

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_lks32mc09x_dma.c](#) 第 94 行定义.

4.21.2.8 HAL_DMA_SpiRxDx()

```
void HAL_DMA_SpiRxDx (
    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_lks32mc09x_dma.c](#) 第 114 行定义.

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_lks32mc09x_dma.c](#) 第 64 行定义.

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_lks32mc09x_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_lks32mc09x_dma.c](#) 第 54 行定义.

4.22 hal_lksmcu_dma.h

[浏览该文件的文档.](#)

```

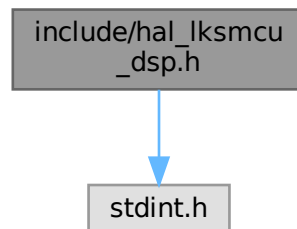
00001 #ifndef __HAL_LKSMCU_DMA_H__
00002 #define __HAL_LKSMCU_DMA_H__
00003
00004 #include "../include/hal_lksmcu_spi.h"
00005 #include "../include/hal_lksmcu_uart.h"
00006 #include "../include/hal_lksmcu_i2c.h"
00007 #include "stdint.h"
00008 typedef enum
00009 {
00010     HAL_DMA_0,
00011     HAL_DMA_1,
00012     HAL_DMA_2,
00013     HAL_DMA_3,
00014 } HAL_DMA_x;
00015
00016 // 初始化
00017 void HAL_DMA_Reset(HAL_DMA_x); // 模块复位
00018 void HAL_DMA_Enable(HAL_DMA_x); // 模块打开
00019 void HAL_DMA_Disable(HAL_DMA_x); // 模块关闭
00020
00021 void HAL_DMA_MemCopy(HAL_DMA_x, void *start, void *end, uint32_t len); // 内存复制
00022
00023 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
00024 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
00025 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
00026 void HAL_DMA_UartRx(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
00027 void HAL_DMA_SpiRx(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
00028 void HAL_DMA_I2cRx(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
00029
00030 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
00031
00032 #endif

```

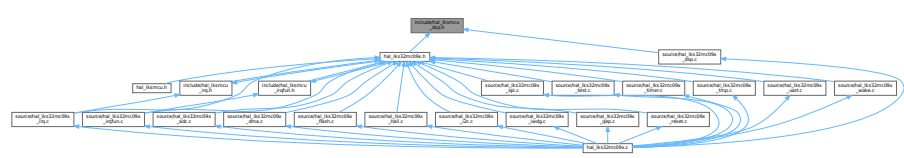
4.23 include/hal_lksmcu_dsp.h 文件参考

```
#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`)
- void `HAL_DSP_Enable` (`HAL_DSP_x`)
- void `HAL_DSP_Disable` (`HAL_DSP_x`)
- int16_t `HAL_DSP_Sin` (`HAL_DSP_x`, int16_t angle)
- int16_t `HAL_DSP_Cos` (`HAL_DSP_x`, int16_t angle)
- void `HAL_DSP_SinCos` (`HAL_DSP_x`, int16_t angle, int16_t *sin, int16_t *cos)
- uint16_t `HAL_DSP_Sqrt` (`HAL_DSP_x`, uint32_t)
- int32_t `HAL_DSP_Div` (`HAL_DSP_x`, int32_t, int32_t)

4.23.1 枚举类型说明

4.23.1.1 HAL_DSP_x

enum `HAL_DSP_x`

枚举值

<code>HAL_↔ DSP_0</code>	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 HAL_DSP_Cos (  
    HAL_DSP_x dsp_x,  
    int16_t angle )
```

在文件 `hal_lks32mc09x_dsp.c` 第 42 行定义.

函数调用图:



4.23.2.2 HAL_DSP_Disable()

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

在文件 [hal_lks32mc09x.dsp.c](#) 第 25 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 )
```

在文件 [hal_lks32mc09x.dsp.c](#) 第 126 行定义.

4.23.2.4 HAL_DSP_Enable()

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

在文件 [hal_lks32mc09x.dsp.c](#) 第 16 行定义.

4.23.2.5 HAL_DSP_Reset()

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

在文件 [hal_lks32mc09x.dsp.c](#) 第 6 行定义.

4.23.2.6 HAL_DSP_Sin()

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

在文件 [hal_lks32mc09x.dsp.c](#) 第 34 行定义.

函数调用图:

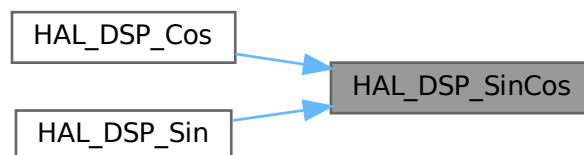


4.23.2.7 HAL_DSP_SinCos()

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

在文件 [hal_lks32mc09x.dsp.c](#) 第 50 行定义.

这是这个函数的调用关系图:



4.23.2.8 HAL_DSP_Sqrt()

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

在文件 [hal_lks32mc09x.dsp.c](#) 第 120 行定义.

4.24 hal_ksmcu_dsp.h

[浏览该文件的文档.](#)

```

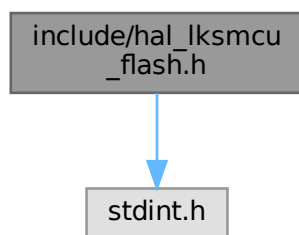
00001 #ifndef __HAL_LKSMCU_DSP_H__
00002 #define __HAL_LKSMCU_DSP_H__
00003 #include "stdint.h"
00004
00005 typedef enum
00006 {
00007     HAL_DSP_0,
00008 } HAL_DSP_x;
00009 void HAL_DSP_Reset(HAL_DSP_x); // 模块复位
00010 void HAL_DSP_Enable(HAL_DSP_x); // 模块打开
00011 void HAL_DSP_Disable(HAL_DSP_x); // 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

```

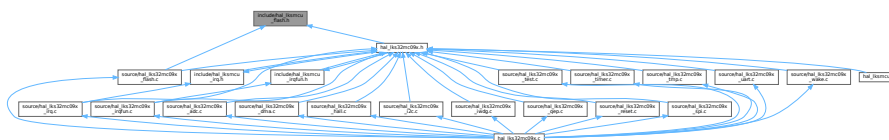
4.25 include/hal_lksmcu_flash.h 文件参考

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

hal_ksmcu_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_lks32mc09x_flash.c](#) 第 30 行定义.

4.25.2.2 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_lks32mc09x_flash.c](#) 第 49 行定义.

4.25.2.3 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_lks32mc09x_flash.c](#) 第 21 行定义.

4.25.2.4 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_lks32mc09x.flash.c](#) 第 39 行定义.

4.25.2.5 HAL_FLASH_Erase()

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

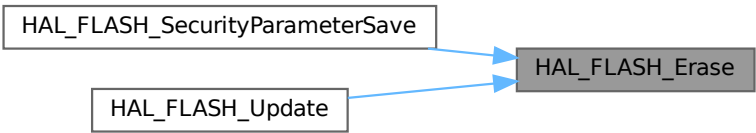
扇区擦除,len只能是扇区大小的整数倍,不够一个扇区的部分会被忽略

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_lks32mc09x.flash.c](#) 第 122 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 )
```

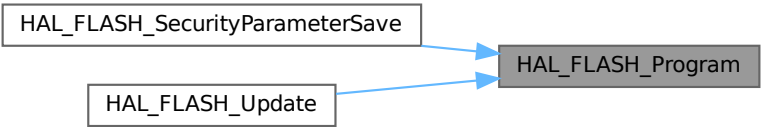
编程

参数

<i>flash</i>	模块类型
<i>flashaddr</i>	flash地址,低两位会被忽略
 南京凌鸥创芯电子有限公司 LINKO SEMICONDUCTOR CO.,LTD.	
<i>len</i>	数据长度

在文件 `hal_lks32mc09x.flash.c` 第 150 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_X flash,
    uint32_t flashaddr,
    void * data,
    uint32_t len )
```

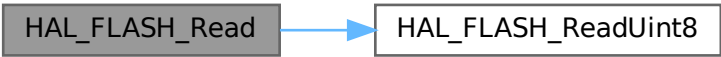
读FLASH取数据

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_lks32mc09x.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

<i>flash</i>	模块类型

返回

uint32_t 扇区大小

在文件 [hal_lks32mc09x_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

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

返回

读取的数据

在文件 [hal_lks32mc09x_flash.c](#) 第 99 行定义.

4.25.2.10 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_lks32mc09x.flash.c` 第 110 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_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_lks32mc09x.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_lks32mc09x.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

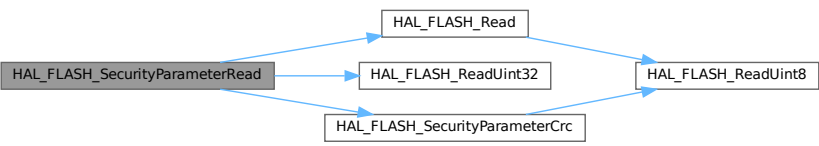
参数

<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_lks32mc09x.flash.c](#) 第 313 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_x_t flash,
    void * defaultdata,
    void * data,
    uint32_t len )
```

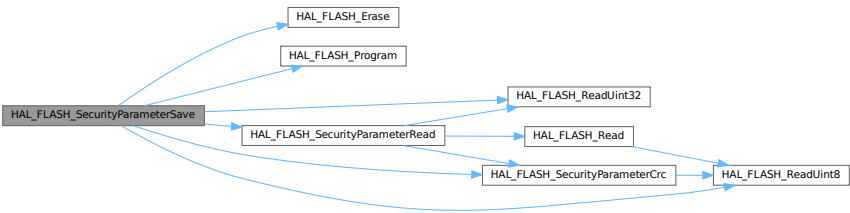
保存安全参数

参数

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

在文件 [hal_lks32mc09x_flash.c](#) 第 222 行定义.

函数调用图:



4.25.2.15 HAL_FLASH_Update()

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

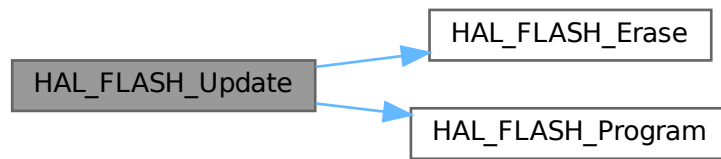
更新数据,等效先擦除再编程

参数

flash	模块类型
flashaddr	flash地址
data	数据指针
len	数据长度

在文件 [hal_lks32mc09x_flash.c](#) 第 180 行定义.

函数调用图:



4.26 hal_lksmcu_flash.h

浏览该文件的文档.

```

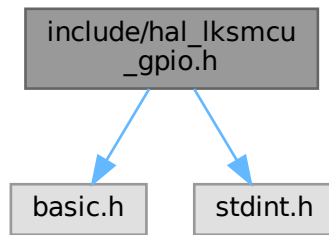
00001 #ifndef __HAL_LKSMCU_FLASH_H__
00002 #define __HAL_LKSMCU_FLASH_H__
00003
00004 #include "stdint.h"
00005 typedef enum
00006 {
00007     HAL_FLASH_0,
00008     HAL_FLASH_1,
00009 } HAL_FLASH_x;
00010
00011 // 初始化
00012 void HAL_FLASH_Reset(HAL_FLASH_x); // 模块复位, 预留接口
00013 void HAL_FLASH_Enable(HAL_FLASH_x); // 模块打开, 预留接口
00014 void HAL_FLASH_Disable(HAL_FLASH_x); // 模块关闭, 预留接口
00015
00016 void HAL_FLASH_EnableCache(HAL_FLASH_x); // 使能flash缓存加
速
00017 void HAL_FLASH_DisableCache(HAL_FLASH_x); // 关闭flash缓存加
速
00018 uint32_t HAL_FLASH_ReadSectorSize(HAL_FLASH_x); // 获取一个扇区的大
小, 和HAL_FLASH_SECTOR_SIZE等效
00019 void HAL_FLASH_Read(HAL_FLASH_x, uint32_t flashaddr, void *data, uint32_t len); // 任意长度
00020 uint8_t HAL_FLASH_ReadUint8(HAL_FLASH_x, uint32_t flashaddr); // 0~(U8)
00021 uint16_t HAL_FLASH_ReadUint16(HAL_FLASH_x, uint32_t flashaddr); // 0~(U16)
00022 uint32_t HAL_FLASH_ReadUint32(HAL_FLASH_x, uint32_t flashaddr); // 0~(U32)
00023 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
00024 void HAL_FLASH_Program(HAL_FLASH_x, uint32_t flashaddr, uint8_t *data, uint32_t len); // 编程
00025 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
00026
00027 // 安全参数 (调用时自动分配地址
00028 // 支持大小为 500的uint8_t的数据
00029 // 保留3个word用来校验
00030 // 状态+CRC32+保留位
00031 // 调用时必须传入defaultdata (确保极端情况下, data的数据安全)
00032 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)
00033 uint32_t HAL_FLASH_SecurityParameterRead(HAL_FLASH_x, void *defaultdata, void *data, uint32_t len); // 读
数据
00034
00035 #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_QEP = 8 , HAL_GPIO_AF_ADC_TRIG
 = 9 ,
 HAL_GPIO_AF_CAN = 10 , HAL_GPIO_AF_SIF = 11 , HAL_GPIO_AF_CL = 12 , HAL_GPIO_AF_TIMER4 = 0 ,
 HAL_GPIO_AF_TIMER5 = 0 }
- enum HAL_GPIO_MODE_x {
 HAL_GPIO_MODE_AFE = 0 , HAL_GPIO_MODE_NONE = HAL_GPIO_MODE_AFE , HAL_GPIO_MODE_IN =
 BIT0 , HAL_GPIO_MODE_INUP = HAL_GPIO_MODE_IN | BIT2 ,
 HAL_GPIO_MODE_OUT = BIT1 , HAL_GPIO_MODE_INOUT = HAL_GPIO_MODE_IN | HAL_GPIO_
 _MODE_OUT | BIT3 , HAL_GPIO_MODE_INUPOUT = HAL_GPIO_MODE_INOUT | BIT2 | BIT3 ,
 HAL_GPIO_MODE_OUT_0 = HAL_GPIO_MODE_OUT | BIT4 ,
 HAL_GPIO_MODE_INOUT_0 = HAL_GPIO_MODE_INOUT | BIT4 , HAL_GPIO_MODE_INUPOUT_0 =
 HAL_GPIO_MODE_INUPOUT | BIT4 , HAL_GPIO_MODE_OUT_1 = HAL_GPIO_MODE_OUT | BIT5 ,
 HAL_GPIO_MODE_INOUT_1 = HAL_GPIO_MODE_INOUT | BIT5 ,
 HAL_GPIO_MODE_INUPOUT_1 = HAL_GPIO_MODE_INUPOUT | BIT5 }

GPIO工作模式定义 BIT0 输入使能 BIT1 输出使能 BIT2 上拉使能 BIT3 开漏使能 BIT4 初始化输出为低电平
 BIT5 初始化输出为高电平

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

<code>HAL_GPIO_AF_ADC</code>	
<code>HAL_GPIO_AF_OPA</code>	
<code>HAL_GPIO_AF_OPA_OUT</code>	
<code>HAL_GPIO_AF_CMP</code>	
<code>HAL_GPIO_AF_DAC</code>	
<code>HAL_GPIO_AF_GPIO</code>	
<code>HAL_GPIO_AF_CLK</code>	
<code>HAL_GPIO_AF_CMP_OUT</code>	
<code>HAL_GPIO_AF_CLK_OUT</code>	
<code>HAL_GPIO_AF_HALL</code>	
<code>HAL_GPIO_AF_MCPWM</code>	
<code>HAL_GPIO_AF_UART</code>	
<code>HAL_GPIO_AF_SPI</code>	
<code>HAL_GPIO_AF_I2C</code>	
<code>HAL_GPIO_AF_TIMER0</code>	
<code>HAL_GPIO_AF_TIMER1</code>	
<code>HAL_GPIO_AF_TIMER2</code>	

枚举值

HAL_GPIO_AF_TIMER3	
HAL_GPIO_AF_QEP	
HAL_GPIO_AF_ADC_TRIG	
HAL_GPIO_AF_CAN	
HAL_GPIO_AF_SIF	
HAL_GPIO_AF_CL	
HAL_GPIO_AF_TIMER4	
HAL_GPIO_AF_TIMER5	

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 BIT0 输入使能 BIT1 输出使能 BIT2 上拉使能 BIT3 开漏使能 BIT4 初始化输出为低电平 BIT5 初始化输出为高电平

枚举值

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)
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 147 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
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

枚举值

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
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_lks32mc09x_gpio.c](#) 第 35 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_lks32mc09x_gpio.c](#) 第 23 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_lks32mc09x_gpio.c](#) 第 177 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_lks32mc09x_gpio.c](#) 第 155 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_lks32mc09x_gpio.c](#) 第 143 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_lks32mc09x_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_lks32mc09x_gpio.c](#) 第 295 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_lks32mc09x_gpio.c](#) 第 283 行定义.

4.27.2.9 HAL_GPIO_SetAf()

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

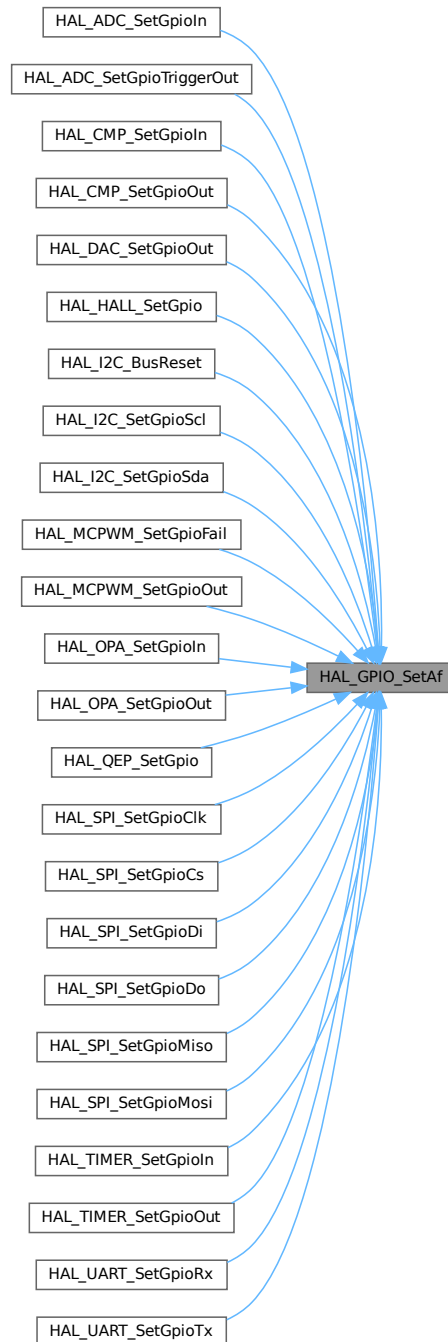
配置GPIO的特殊功能,如UART、I2C、SPI等

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_lks32mc09x_gpio.c](#) 第 199 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_lks32mc09x_gpio.c](#) 第 108 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_lks32mc09x_gpio.c](#) 第 119 行定义.

4.27.2.12 HAL_GPIO_SetMode()

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

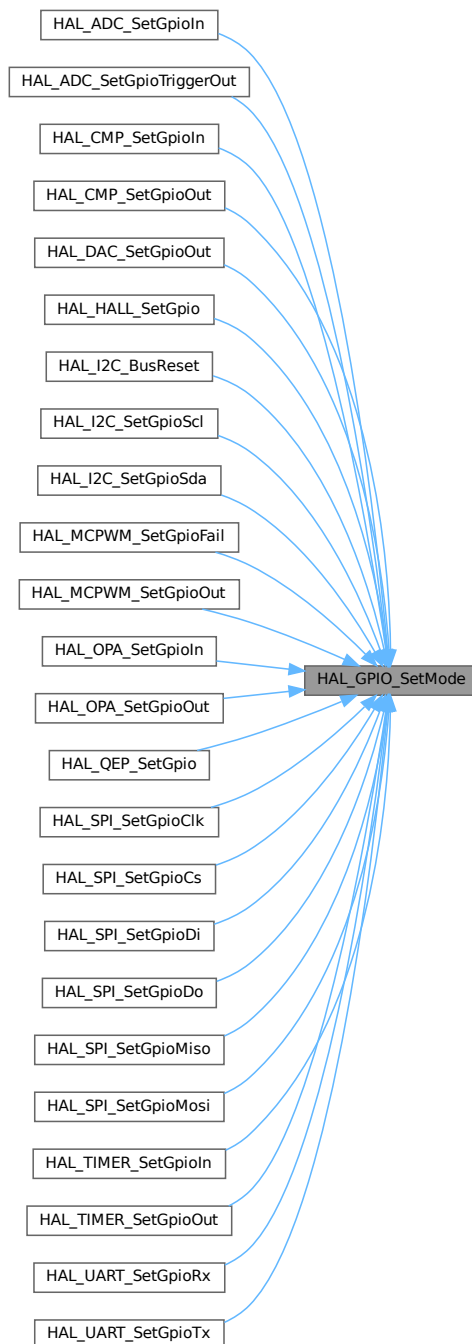
设置指定GPIO的工作模式

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_lks32mc09x_gpio.c](#) 第 49 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_lks32mc09x_gpio.c](#) 第 271 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_lks32mc09x_gpio.c](#) 第 259 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_lks32mc09x_gpio.c](#) 第 131 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_QEP    = 8, // 编码器输入
00128
00129     HAL_GPIO_AF_ADC_TRIG = 9, // adc触发信号输出
00130     HAL_GPIO_AF_CAN      = 10, // CAN输入输出
00131     HAL_GPIO_AF_SIF      = 11, // SIF信号输出
00132     HAL_GPIO_AF_CL       = 12, // CL输出
00133
00134     HAL_GPIO_AF_TIMER4 = 0, // TIMER4输入或输出 不支持 这里保留接口定义
00135     HAL_GPIO_AF_TIMER5 = 0, // TIMER5输入或输出 不支持 这里保留接口定义
00136 } HAL_GPIO_Afx;
00147 typedef enum
00148 {
00149     HAL_GPIO_MODE_AFE          = 0,
00150     HAL_GPIO_MODE_NONE        = HAL_GPIO_MODE_AFE,
00151     HAL_GPIO_MODE_IN          = BIT0,
00152     HAL_GPIO_MODE_INUP        = HAL_GPIO_MODE_IN | BIT2,
00153     HAL_GPIO_MODE_OUT         = BIT1,
00154     HAL_GPIO_MODE_INOUT       = HAL_GPIO_MODE_IN | HAL_GPIO_MODE_OUT | BIT3,
00155     HAL_GPIO_MODE_INUPOUT     = HAL_GPIO_MODE_INOUT | BIT2 | BIT3,
00156     HAL_GPIO_MODE_OUT_0       = HAL_GPIO_MODE_OUT | BIT4,
00157     HAL_GPIO_MODE_INOUT_0     = HAL_GPIO_MODE_INOUT | BIT4,
00158     HAL_GPIO_MODE_INUPOUT_0   = HAL_GPIO_MODE_INUPOUT | BIT4,
00159     HAL_GPIO_MODE_OUT_1       = HAL_GPIO_MODE_OUT | BIT5,
00160     HAL_GPIO_MODE_INOUT_1     = HAL_GPIO_MODE_INOUT | BIT5,
00161     HAL_GPIO_MODE_INUPOUT_1   = HAL_GPIO_MODE_INUPOUT | BIT5,
00162 } HAL_GPIO_MODE_x;
00163 // 模块初始化
00164 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)
00165 void HAL_GPIO_Enable(void); // 模块打开
00166 void HAL_GPIO_Disable(void); // 模块关闭
00167
00168 // 输入输出
00169 void HAL_GPIO_SetMode(HAL_GPIO_Px, HAL_GPIO_MODE_x); // 设置指定GPIO的工作模式
00170 void HAL_GPIO_SetHigh(HAL_GPIO_Px); // 输出1 (INOUT和INUPOUT模式下分别为悬空和上拉)
00171 void HAL_GPIO_SetLow(HAL_GPIO_Px); // 输出0
00172 void HAL_GPIO_Write(HAL_GPIO_Px, uint8_t); // 输出0或1
00173 void HAL_GPIO_Invert(HAL_GPIO_Px); // 翻转指定的GPIO
00174 uint8_t HAL_GPIO_GetOutData(HAL_GPIO_Px); // 获取当前GPIO的输出值, 返回值0或1
00175 uint8_t HAL_GPIO_GetInData(HAL_GPIO_Px); // 获取当前GPIO的输入值, 返回值0或1
00176
00177 // 内部使用
00178 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)
00179 // 特殊GPIO口配置
00180 void HAL_GPIO_SwdReuseGpioEnable(void); // 设置SWD接口为普通GPIO口
00181 void HAL_GPIO_SwdReuseGpioDisable(void); // 设置SWD接口为SWD功能
00182 void HAL_GPIO_RstReuseGpioEnable(void); // 设置RST接口为普通GPIO口
00183 void HAL_GPIO_RstReuseGpioDisable(void); // 设置RST接口为RST功能
00184 #endif

```

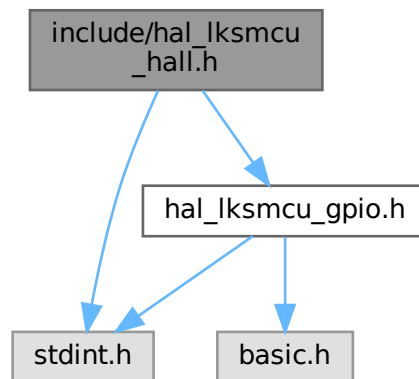
4.29 include/hal_lksmcu_hall.h 文件参考

```

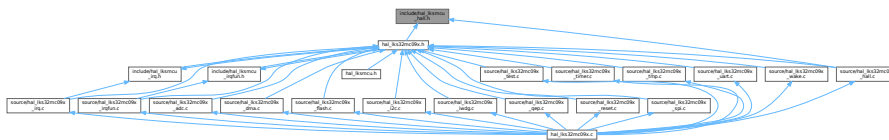
#include "stdint.h"
#include "hal_lksmcu_gpio.h"

```

hal_ksmcu_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`)

聰劇疆婁ゆ尝

- void [HAL_HALL_SetFilteruS](#) ([HAL_HALL_x](#), uint32_t filter_uS)
设置滤波, 单位: μS
- uint32_t [HAL_HALL_ReadTh](#) ([HAL_HALL_x](#))
获取当前 *hall* 的周期计数值
- uint8_t [HAL_HALL_ReadHallData](#) ([HAL_HALL_x](#))
获取当前 *hall* 的数据
- uint32_t [HAL_HALL_ReadHallWidth](#) ([HAL_HALL_x](#))
获取当前 *hall* 的宽度
- void [HAL_HALL_SetGpio](#) ([HAL_HALL_x](#), [HAL_GPIO_Px](#))
设置 *hall* 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_lks32mc09x_hall.c](#) 第 40 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_lks32mc09x_hall.c](#) 第 28 行定义.

4.29.2.3 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_lks32mc09x_hall.c](#) 第 85 行定义.

4.29.2.4 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_lks32mc09x_hall.c](#) 第 95 行定义.

4.29.2.5 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_lks32mc09x.hall.c](#) 第 75 行定义.

4.29.2.6 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_lks32mc09x.hall.c](#) 第 13 行定义.

4.29.2.7 HAL_HALL_SetFilter()

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

璁剧疆婊0尝

参数

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

在文件 [hal_lks32mc09x.hall.c](#) 第 135 行定义.
这是这个函数的调用关系图:



4.29.2.8 HAL_HALL_SetFilteruS()

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

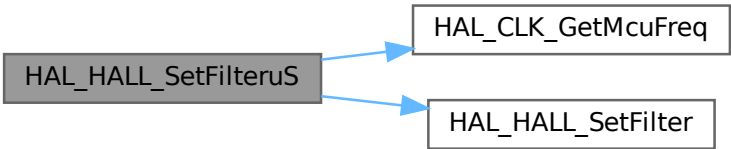
设置滤波，单位：uS

参数

hall	模块号
filter_uS	滤波值

在文件 [hal_lks32mc09x.hall.c](#) 第 167 行定义.

函数调用图:



4.29.2.9 HAL_HALL_SetFreq()

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

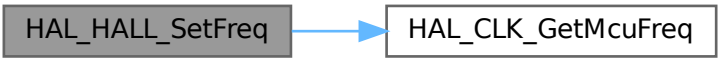
设置hall频率

参数

hall	模块号
freq	频率值

在文件 [hal_lks32mc09x.hall.c](#) 第 53 行定义.

函数调用图:



4.29.2.10 HAL_HALL_SetGpio()

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

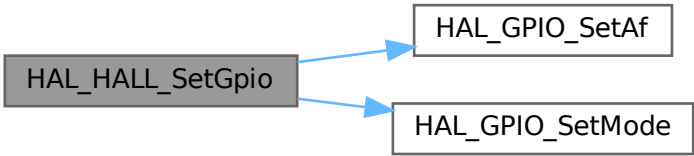
设置hall输入引脚

参数

hall	模块号
px	引脚

在文件 `hal_lks32mc09x_hall.c` 第 105 行定义.

函数调用图:



4.29.2.11 HAL_HALL_SetTh()

```
void HAL_HALL_SetTh (
    HAL_HALL_x hall,
    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_lks32mc09x_hall.c](#) 第 65 行定义.

4.29.2.12 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_lks32mc09x_hall.c](#) 第 116 行定义.

4.29.2.13 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_lks32mc09x_hall.c](#) 第 124 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
```

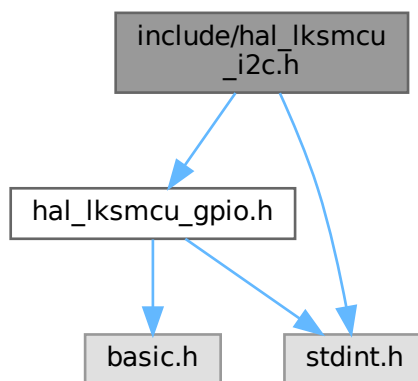
```
00017 void HAL_HALL_StartCounting(HAL_HALL_x); // 开始计数 HAL_HALL_Enable后默认开始计数
00018 void HAL_HALL_StopCounting(HAL_HALL_x); // 停止计数(同时计数器回到0)
00019 void HAL_HALL_SetFilter(HAL_HALL_x, uint32_t filter_mclk); // 设置滤波, 单位: MCU时钟
00020 void HAL_HALL_SetFilteruS(HAL_HALL_x, uint32_t filter_uS); // 设置滤波, 单位: uS
00021
00022 uint32_t HAL_HALL_ReadTh(HAL_HALL_x); // 获取当前hall的周期计数值
00023 uint8_t HAL_HALL_ReadHallData(HAL_HALL_x); // 获取当前hall的数据
00024 uint32_t HAL_HALL_ReadHallWidth(HAL_HALL_x); // 获取当前hall捕获到的宽度
00025
00026 void HAL_HALL_SetGpio(HAL_HALL_x, HAL_GPIO_Px); // 设置hall输入引脚
00027 #endif
```

4.31 include/hal_lksmcu_i2c.h 文件参考

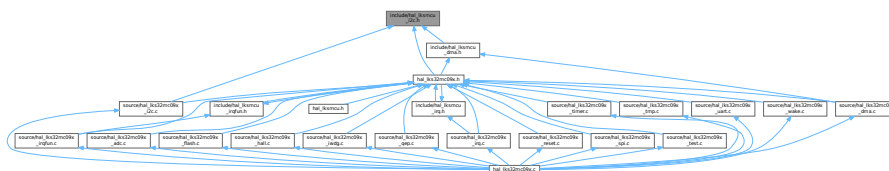
```
#include "hal_lksmcu_gpio.h"
```

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

hal_ksmcu_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
- 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 )
```

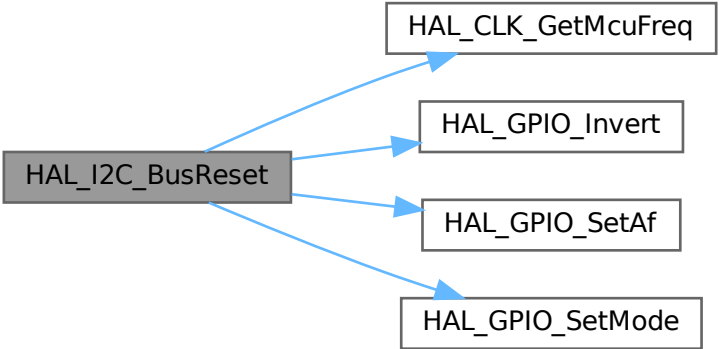
复位I2C总线

参数

i2c↔ _x	I2C模块编号

在文件 [hal_lks32mc09x_i2c.c](#) 第 201 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_x</i>	I2C模块编号

在文件 [hal_lks32mc09x_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_x</i>	I2C模块编号

在文件 [hal_lks32mc09x_i2c.c](#) 第 26 行定义.

4.31.2.4 HAL_I2C_Reset()

```
void HAL_I2C_Reset (
    HAL_I2C_TypeDef* i2c_x )
```

复位I2C模块

参数

<i>i2c_x</i>	I2C模块编号

在文件 [hal_lks32mc09x_i2c.c](#) 第 13 行定义.

4.31.2.5 HAL_I2C_RxData()

```
uint8_t HAL_I2C_RxData (
    HAL_I2C_TypeDef* 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_lks32mc09x_i2c.c](#) 第 130 行定义.

4.31.2.6 HAL_I2C_SetBaud()

```
void HAL_I2C_SetBaud (
    HAL_I2C_TypeDef* i2c_x,
    uint32_t baud )
```

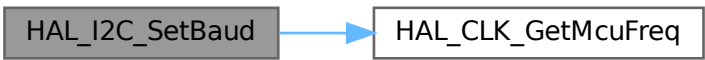
设置I2C模块波特率

参数

<i>i2c_x</i>	I2C模块编号
<i>baud</i>	波特率

在文件 [hal_lks32mc09x.i2c.c](#) 第 50 行定义.

函数调用图:



4.31.2.7 HAL_I2C_SetGpioScl()

```
void HAL_I2C_SetGpioScl (
    HAL_I2C_x i2c_x,
    HAL_GPIO_Px i2cgpio )
```

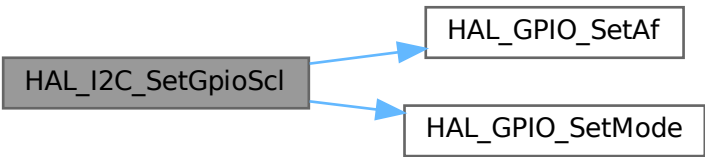
设置I2C模块的GPIO

参数

<i>i2c_x</i>	I2C模块编号
<i>i2cgpio</i>	GPIO编号

在文件 [hal_lks32mc09x.i2c.c](#) 第 178 行定义.

函数调用图:



4.31.2.8 HAL_I2C_SetGpioSda()

```
void HAL_I2C_SetGpioSda (
    HAL_I2C_x i2c_x,
    HAL_GPIO_Px i2cgpio )
```

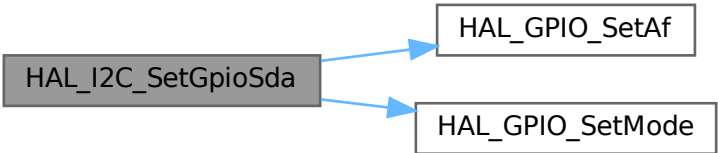
设置I2C模块的GPIO

参数

<i>i2c_x</i>	I2C模块编号
<i>i2cgpio</i>	GPIO编号

在文件 [hal_lks32mc09x_i2c.c](#) 第 190 行定义.

函数调用图:



4.31.2.9 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_lks32mc09x_i2c.c` 第 69 行定义.

4.32 hal_ksmcu_i2c.h

[浏览该文件的文档.](#)

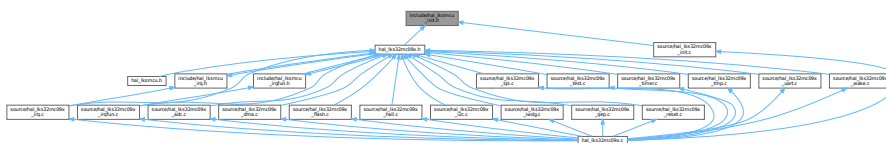
```

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 uint8_t HAL_I2C_TxData(HAL_I2C_x, uint8_t addr, uint8_t *data, uint32_t len);
00017 uint8_t HAL_I2C_RxData(HAL_I2C_x, uint8_t addr, uint8_t *data, uint32_t len);
00018 //
00019 void HAL_I2C_SetGpioScl(HAL_I2C_x, HAL_GPIO_Px i2cgpio);
00020 void HAL_I2C_SetGpioSda(HAL_I2C_x, HAL_GPIO_Px i2cgpio);
00021 //
00022 void HAL_I2C_BusReset(HAL_I2C_x); // 复位总线
00023 #endif

```

4.33 include/hal_iksmcu_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:



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 枚举类型说明

4.33.1.1 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 3 行定义.

4.33.2 函数说明

4.33.2.1 HAL_INIT_PowerReady()

```
void HAL_INIT_PowerReady (
    HAL_INIT_POWER_x powerx )
```

在文件 [hal_lks32mc09x_init.c](#) 第 14 行定义.

4.33.2.2 HAL_INIT_SysInit()

```
void HAL_INIT_SysInit (
    void )
```

在文件 [hal_lks32mc09x_init.c](#) 第 6 行定义.

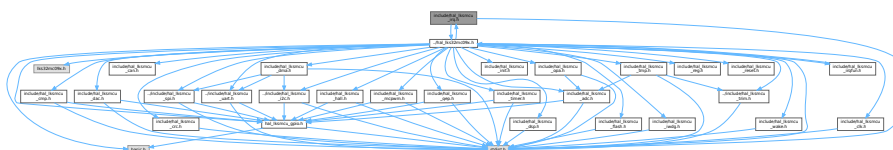
4.34 hal_lksmcu_init.h

[浏览该文件的文档.](#)

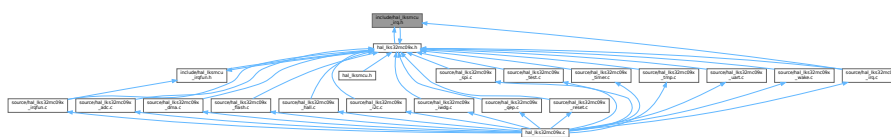
```
00001 #ifndef __HAL_LKSMCU_INIT_H__
00002 #define __HAL_LKSMCU_INIT_H__
00003 typedef enum
00004 {
00005     HAL_INIT_POWER_3V = 0,
00006     HAL_INIT_POWER_5V,
00007 } HAL_INIT_POWER_x;
00008 void HAL_INIT_SysInit(void);
00009 void HAL_INIT_PowerReady(HAL_INIT_POWER_x powerx);
00010
00011 #endif
```

4.35 include/hal_lksmcu_irq.h 文件参考

```
#include "stdint.h"
#include "../hal_lks32mc09x.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:



枚举

- enum **HAL_IRQ_MODULE_x** {
HAL_IRQ_MODULE_NonMaskableInt = NonMaskableInt_IRQn , **HAL_IRQ_MODULE_HardFault** = HardFault_IRQn , **HAL_IRQ_MODULE_SVCALL** = SVCALL_IRQn , **HAL_IRQ_MODULE_PendSV** = PendSV_IRQn ,
HAL_IRQ_MODULE_SysTick = SysTick_IRQn , **HAL_IRQ_MODULE_TIMER_00** = TIMER0_IRQn ,
HAL_IRQ_MODULE_TIMER_10 = TIMER1_IRQn , **HAL_IRQ_MODULE_TIMER_20** = TIMER2_IRQn ,
HAL_IRQ_MODULE_MCPWM_00 = MCPWM0_IRQn , **HAL_IRQ_MODULE_MCPWM_01** = MCPWM1_IRQn ,
HAL_IRQ_MODULE_I2C_00 = I2C0_IRQn , **HAL_IRQ_MODULE_SPI_00** = SPI0_IRQn ,
HAL_IRQ_MODULE_GPIO_00 = GPIO_IRQn , **HAL_IRQ_MODULE_HALL_00** = HALL0_IRQn , **HAL_IRQ_MODULE_UART_00** = UART0_IRQn , **HAL_IRQ_MODULE_UART_10** = UART1_IRQn ,
HAL_IRQ_MODULE_UART_20 = UART2_IRQn , **HAL_IRQ_MODULE_CMP_00** = CMP0_IRQn , **HAL_IRQ_MODULE_CMP_10** = CMP1_IRQn , **HAL_IRQ_MODULE_ADC_00** = ADC0_IRQn ,
HAL_IRQ_MODULE_ADC_10 = ADC1_IRQn , **HAL_IRQ_MODULE_WAKE_00** = WAKE_IRQn , **HAL_IRQ_MODULE_QEP_00** = QEP0_IRQn , **HAL_IRQ_MODULE_DMA_00** = DMA0_IRQn ,
HAL_IRQ_MODULE_SIF_00 = SIF0_IRQn , **HAL_IRQ_MODULE_SW_00** = SW_IRQn , **HAL_IRQ_MODULE_INIT_00** = PWRDN_IRQn , **HAL_IRQ_MODULE_CL_00** = CL0_IRQn }
 各个模块的中断编号 如果调用不存在中断的模块,则不做任何操作

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

4.35.1 枚举类型说明

4.35.1.1 HAL_IRQ_MODULE_x

enum **HAL_IRQ_MODULE_x**

各个模块的中断编号 如果调用不存在中断的模块,则不做任何操作

枚举值

HAL_IRQ_MODULE_NonMaskableInt

枚举值

HAL_IRQ_MODULE_HardFault	
HAL_IRQ_MODULE_SVCall	
HAL_IRQ_MODULE_PendSV	
HAL_IRQ_MODULE_SysTick	
HAL_IRQ_MODULE_TIMER_00	
HAL_IRQ_MODULE_TIMER_10	
HAL_IRQ_MODULE_TIMER_20	
HAL_IRQ_MODULE_MCPWM_00	
HAL_IRQ_MODULE_MCPWM_01	
HAL_IRQ_MODULE_I2C_00	
HAL_IRQ_MODULE_SPI_00	
HAL_IRQ_MODULE_GPIO_00	
HAL_IRQ_MODULE_HALL_00	
HAL_IRQ_MODULE_UART_00	
HAL_IRQ_MODULE_UART_10	
HAL_IRQ_MODULE_UART_20	
HAL_IRQ_MODULE_CMP_00	
HAL_IRQ_MODULE_CMP_10	
HAL_IRQ_MODULE_ADC_00	
HAL_IRQ_MODULE_ADC_10	
HAL_IRQ_MODULE_WAKE_00	
HAL_IRQ_MODULE_QEP_00	
HAL_IRQ_MODULE_DMA_00	
HAL_IRQ_MODULE_SIF_00	
HAL_IRQ_MODULE_SW_00	
HAL_IRQ_MODULE_INIT_00	
HAL_IRQ_MODULE_CL_00	

在文件 [hal_lksmcu_irq.h](#) 第 11 行定义.

4.35.2 函数说明

4.35.2.1 HAL_IRQ_Disable()

```
void HAL_IRQ_Disable (
    void )
```

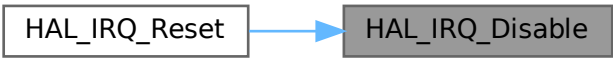
关闭全局中断

参数

<i>irq</i>	中断编号

在文件 [hal_lks32mc09x_irq.c](#) 第 29 行定义.

这是这个函数的调用关系图:



4.35.2.2 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_lks32mc09x_irq.c](#) 第 20 行定义.

4.35.2.3 HAL_IRQ_ModuleDisable()

```
void HAL_IRQ_ModuleDisable (
    HAL_IRQ_MODULE_x module )
```

关闭某个模块的中断

参数

<i>irq</i>	中断编号
<i>module</i>	模块编号
<i>number</i>	中断号

在文件 [hal_lks32mc09x_irq.c](#) 第 52 行定义.

4.35.2.4 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>irq</i>	中断编号
<i>module</i>	模块编号
<i>number</i>	中断号
<i>priority</i>	中断优先级

在文件 [hal_lks32mc09x_irq.c](#) 第 41 行定义.

4.35.2.5 HAL_IRQ_Reset()

```
void HAL_IRQ_Reset (
    void )
```

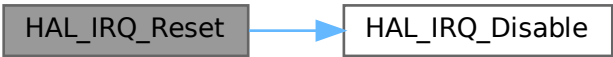
重置中断(等效于关闭中断)

参数

<i>irq</i>	中断编号

在文件 [hal_lks32mc09x_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_lks32mc09x.h"
00006
00011 typedef enum
00012 {
00013     // 来自arm内核的模块
00014     HAL_IRQ_MODULE.NonMaskableInt = NonMaskableInt_IRQn,
00015     HAL_IRQ_MODULE.HardFault      = HardFault_IRQn,
00016     HAL_IRQ_MODULE.SVCall         = SVCall_IRQn,
00017     HAL_IRQ_MODULE.PendSV         = PendSV_IRQn,
00018     HAL_IRQ_MODULE.SysTick        = SysTick_IRQn,
00019     // 芯片外设
00020     HAL_IRQ_MODULE.TIMER_00 = TIMER0_IRQn,
00021     HAL_IRQ_MODULE.TIMER_10 = TIMER1_IRQn,
00022     HAL_IRQ_MODULE.TIMER_20 = TIMER2_IRQn,
```

```

00023     HAL_IRQ_MODULE_MCPWM_00 = MCPWM0_IRQn,
00024     HAL_IRQ_MODULE_MCPWM_01 = MCPWM1_IRQn,
00025     HAL_IRQ_MODULE_I2C_00   = I2C0_IRQn,
00026     HAL_IRQ_MODULE_SPI_00    = SPI0_IRQn,
00027     HAL_IRQ_MODULE_GPIO_00   = GPIO_IRQn,
00028     HAL_IRQ_MODULE_HALL_00   = HALL0_IRQn,
00029     HAL_IRQ_MODULE_UART_00   = UART0_IRQn,
00030     HAL_IRQ_MODULE_UART_10   = UART1_IRQn,
00031     HAL_IRQ_MODULE_UART_20   = UART2_IRQn,
00032     HAL_IRQ_MODULE_CMP_00    = CMP0_IRQn,
00033     HAL_IRQ_MODULE_CMP_10    = CMP1_IRQn,
00034     HAL_IRQ_MODULE_ADC_00    = ADC0_IRQn,
00035     HAL_IRQ_MODULE_ADC_10    = ADC1_IRQn,
00036     HAL_IRQ_MODULE_WAKE_00   = WAKE_IRQn,
00037     HAL_IRQ_MODULE_QEP_00    = QEP0_IRQn,
00038     HAL_IRQ_MODULE_DMA_00    = DMA0_IRQn,
00039     HAL_IRQ_MODULE_SIF_00    = SIF0_IRQn,
00040     HAL_IRQ_MODULE_SW_00     = SW_IRQn,
00041     HAL_IRQ_MODULE_INIT_00   = PWRDN_IRQn,
00042     HAL_IRQ_MODULE_CL_00     = CL0_IRQn,
00043 } HAL_IRQ_MODULE_x;
00044
00045 void HAL_IRQ_Reset(void); // 重置中断(等效于关闭中断)
00046 void HAL_IRQ_Enable(void); // 使能全局中断
00047 void HAL_IRQ_Disable(void); // 关闭全局中断
00048
00049 void HAL_IRQ_ModuleEnable(HAL_IRQ_MODULE_x, uint8_t priority); // 使能某个模块的中断
00050 void HAL_IRQ_ModuleDisable(HAL_IRQ_MODULE_x); // 关闭某个模块的中断
00051
00052 #endif

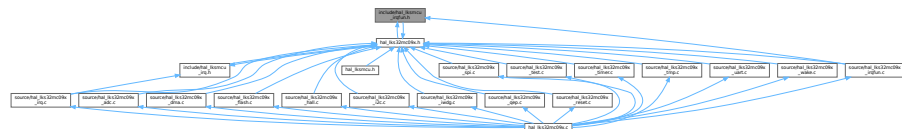
```

4.37 include/hal_lksmcu_irqfun.h 文件参考

#include "../hal_lks32mc09x.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

4.37.1 函数说明

4.37.1.1 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_lks32mc09x_irqfun.c](#) 第 128 行定义.

4.37.1.2 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_lks32mc09x_irqfun.c](#) 第 175 行定义.

4.37.1.3 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_lks32mc09x_irqfun.c](#) 第 167 行定义.

4.37.1.4 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_lks32mc09x_irqfun.c](#) 第 159 行定义.

4.37.1.5 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_lks32mc09x_irqfun.c](#) 第 144 行定义.

4.37.1.6 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_lks32mc09x_irqfun.c](#) 第 185 行定义.

4.37.1.7 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_lks32mc09x_irqfun.c](#) 第 232 行定义.

4.37.1.8 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_lks32mc09x_irqfun.c](#) 第 224 行定义.

4.37.1.9 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_lks32mc09x_irqfun.c](#) 第 216 行定义.

4.37.1.10 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_lks32mc09x_irqfun.c](#) 第 201 行定义.

4.37.1.11 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_lks32mc09x_irqfun.c](#) 第 14 行定义.

4.37.1.12 HAL_HALL_IrqChangeClear()

```
void HAL_HALL_IrqChangeClear (
    HAL_HALLx hall )
```

中断清除

参数

<i>hall</i>	模块编号

在文件 [hal_lks32mc09x_irqfun.c](#) 第 61 行定义.

4.37.1.13 HAL_HALL_IrqChangeDisable()

```
void HAL_HALL_IrqChangeDisable (
    HAL_HALLx hall )
```

中断禁止

参数

<i>hall</i>	模块编号

在文件 [hal_lks32mc09x_irqfun.c](#) 第 53 行定义.

4.37.1.14 HAL_HALL_IrqChangeEnable()

```
void HAL_HALL_IrqChangeEnable (
    HAL_HALLx hall )
```

中断使能

参数

<i>hall</i>	模块编号

在文件 [hal_lks32mc09x_irqfun.c](#) 第 45 行定义.

4.37.1.15 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

hall	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_lks32mc09x_irqfun.c](#) 第 30 行定义.

4.37.1.16 HAL_HALL_IrqOver()

```
uint8_t HAL_HALL_IrqOver (
    HAL_HALL_x hall )
```

中断处理函数

参数

hall	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_lks32mc09x_irqfun.c](#) 第 71 行定义.

4.37.1.17 HAL_HALL_IrqOverClear()

```
void HAL_HALL_IrqOverClear (
    HAL_HALL_x hall )
```

中断清除

参数

hall	模块编号

在文件 [hal_lks32mc09x_irqfun.c](#) 第 118 行定义.

4.37.1.18 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_lks32mc09x_irqfun.c](#) 第 110 行定义.

4.37.1.19 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_lks32mc09x_irqfun.c](#) 第 102 行定义.

4.37.1.20 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_lks32mc09x_irqfun.c](#) 第 87 行定义.

4.38 hal_lksmcu_irqfun.h

[浏览该文件的文档.](#)

```

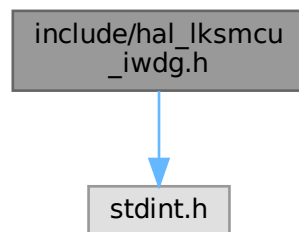
00001
00002 #ifndef __HAL_LKSMCU_IRQFUN_H__
00003 #define __HAL_LKSMCU_IRQFUN_H__
00004
00005 #include "../hal_lks32mc09x.h"
00006
00007 uint8_t HAL_HALL_IrqChange (HAL_HALL_x);
00008 uint8_t HAL_HALL_IrqChangeFlg (HAL_HALL_x hall);
00009 void HAL_HALL_IrqChangeEnable (HAL_HALL_x);
00010 void HAL_HALL_IrqChangeDisable (HAL_HALL_x hall);
00011 void HAL_HALL_IrqChangeClear (HAL_HALL_x hall);
00012
00013 uint8_t HAL_HALL_IrqOver (HAL_HALL_x);
00014 uint8_t HAL_HALL_IrqOverFlg (HAL_HALL_x hall);
00015 void HAL_HALL_IrqOverEnable (HAL_HALL_x);
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);
00020 uint8_t HAL_ADC_IrqSamp1Flg (HAL_ADC_x adc);
00021 void HAL_ADC_IrqSamp1Enable (HAL_ADC_x);
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);
00026 uint8_t HAL_ADC_IrqSamp2Flg (HAL_ADC_x adc);
00027 void HAL_ADC_IrqSamp2Enable (HAL_ADC_x);
00028 void HAL_ADC_IrqSamp2Disable (HAL_ADC_x adc);
00029 void HAL_ADC_IrqSamp2Clear (HAL_ADC_x adc);
00030
00031 #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_lks32mc09x_iwdg.c](#) 第 28 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_lks32mc09x_iwdg.c](#) 第 19 行定义.

4.39.2.3 HAL_IWDG_Feed()

```
void HAL_IWDG_Feed (
    HAL_IWDG_x iwdg )
```

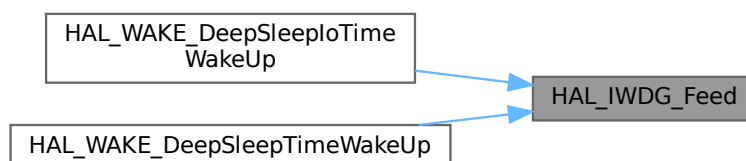
喂狗

参数

<i>iwdg</i>	看门狗模块

在文件 [hal_lks32mc09x_iwdg.c](#) 第 52 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 )
```

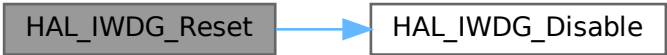
模块复位

参数

<i>iwdg</i>	看门狗模块

在文件 [hal_lks32mc09x_iwdg.c](#) 第 10 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_lks32mc09x_iwdg.c](#) 第 40 行定义.

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

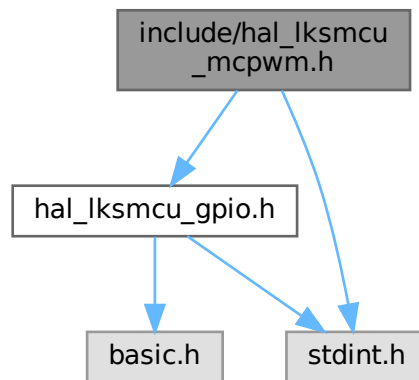
```

4.41 include/hal_lksmcu_mcpwm.h 文件参考

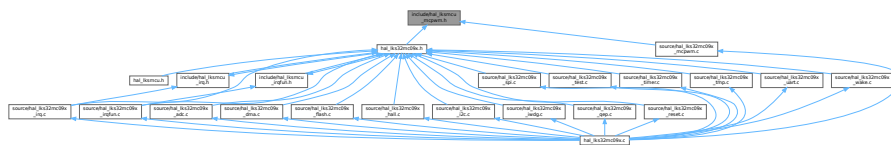
```
#include "hal_lksmcu_gpio.h"
```

```
#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, HAL_MCPWM_1 }
- enum HAL_MCPWM_PORT_x { HAL_MCPWM_PORT_0, HAL_MCPWM_PORT_1, HAL_MCPWM_PORT_2, HAL_MCPWM_PORT_3, HAL_MCPWM_PORT_4, HAL_MCPWM_PORT_5 }
- 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 }
- 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 }

- 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 }
- enum HAL_MCPWM_FAIL_x { HAL_MCPWM_FAIL_IO0_HIGH , HAL_MCPWM_FAIL_IO1_HIGH , HAL_MCPWM_FAIL_CMP0_HIGH , HAL_MCPWM_FAIL_CMP1_HIGH , HAL_MCPWM_FAIL_IO0_LOW , HAL_MCPWM_FAIL_IO1_LOW , HAL_MCPWM_FAIL_CMP0_LOW , HAL_MCPWM_FAIL_CMP1_LOW }

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
- 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_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_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)
- void HAL_MCPWM_SetAdcTrigQ15 (HAL_MCPWM_x, int32_t t0, int32_t t1)
- void HAL_MCPWM_SetDrivePolarity (HAL_MCPWM_x, HAL_MCPWM_POLARITY_x)
设置*pwm*极性
- 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)
- void HAL_MCPWM_SetFailSignal (HAL_MCPWM_x, HAL_MCPWM_FAIL_x)
设置*FAIL*信号来源
- uint8_t HAL_MCPWM_IrqOverflow (HAL_MCPWM_x)
- void HAL_MCPWM_IrqOverflowEnable (HAL_MCPWM_x)
- void HAL_MCPWM_IrqOverflowDisable (HAL_MCPWM_x)

- void [HAL_MCPWM_IrqOverflowClear](#) (HAL_MCPWM_x)
- uint8_t [HAL_MCPWM_IrqZeroCross](#) (HAL_MCPWM_x)
- void [HAL_MCPWM_IrqZeroCrossEnable](#) (HAL_MCPWM_x)
- void [HAL_MCPWM_IrqZeroCrossDisable](#) (HAL_MCPWM_x)
- void [HAL_MCPWM_IrqZeroCrossClear](#) (HAL_MCPWM_x)
- uint8_t [HAL_MCPWM_IrqFail](#) (HAL_MCPWM_x)
- void [HAL_MCPWM_IrqFailEnable](#) (HAL_MCPWM_x)
- void [HAL_MCPWM_IrqFailDisable](#) (HAL_MCPWM_x)
- void [HAL_MCPWM_IrqFailClear](#) (HAL_MCPWM_x)

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](#)

枚举值

HAL_MCPWM_DEFAULT_OUTPUT_STATE_OFF	关闭输出
HAL_MCPWM_DEFAULT_OUTPUT_STATE_LOW	下管恒开
HAL_MCPWM_DEFAULT_OUTPUT_STATE_HIGH	上管恒开

在文件 [hal_lksmcu_mcpwm.h](#) 第 36 行定义.

4.41.1.2 HAL_MCPWM_FAIL_x

enum [HAL_MCPWM_FAIL_x](#)

枚举值

HAL_MCPWM_FAIL_IO0_HIGH	
HAL_MCPWM_FAIL_IO1_HIGH	
HAL_MCPWM_FAIL_CMP0_HIGH	
HAL_MCPWM_FAIL_CMP1_HIGH	
HAL_MCPWM_FAIL_IO0_LOW	
HAL_MCPWM_FAIL_IO1_LOW	
HAL_MCPWM_FAIL_CMP0_LOW	
HAL_MCPWM_FAIL_CMP1_LOW	

在文件 [hal_lksmcu_mcpwm.h](#) 第 42 行定义.

4.41.1.3 HAL_MCPWM_OUTPUT_STATE_x

enum [HAL_MCPWM_OUTPUT_STATE_x](#)

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 27 行定义.

4.41.1.4 HAL_MCPWM_POLARITY_x

enum [HAL_MCPWM_POLARITY_x](#)

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 20 行定义.

4.41.1.5 HAL_MCPWM_PORT_x

enum [HAL_MCPWM_PORT_x](#)

枚举值

HAL_MCPWM_↔ PORT_0	输出端口
HAL_MCPWM_↔ PORT_1	输出端口
HAL_MCPWM_↔ PORT_2	输出端口
HAL_MCPWM_↔ PORT_3	输出端口
HAL_MCPWM_↔ PORT_4	输出端口
HAL_MCPWM_↔ PORT_5	输出端口

在文件 [hal_lksmcu_mcpwm.h](#) 第 11 行定义.

4.41.1.6 HAL_MCPWM_x

enum HAL_MCPWM_x

枚举值

HAL↔ MCPWM_0	MCPWM模块0
HAL↔ MCPWM_1	MCPWM模块1

在文件 [hal_lksmcu_mcpwm.h](#) 第 5 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

mcpwm↔ _x	操作的模块编号

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 191 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

mcpwm↔ _x	操作的模块编号

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 172 行定义.

4.41.2.3 HAL_MCPWM_GetFreq()

```
uint32_t HAL_MCPWM_GetFreq (
    HAL_MCPWM_x mcpwm_x )
```

获取pwm频率

参数

<i>mcpwm</i> ↔ _x	操作的模块编号

返回
当前pwm的频率

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 221 行定义.

函数调用图:



4.41.2.4 HAL_MCPWM_GetThCnt()

```
uint32_t HAL_MCPWM_GetThCnt (
    HAL_MCPWM_x mcpwm_x )
```

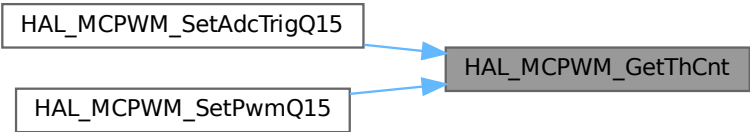
获取计数器最大值

参数

<i>mcpwm</i> ↔ _x	操作的模块编号

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 232 行定义.

这是这个函数的调用关系图:



4.41.2.5 HAL_MCPWM_IrqFail()

```
uint8_t HAL_MCPWM_IrqFail (
    HAL_MCPWM_TypeDef * mcpwm_x )
```

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 494 行定义.

4.41.2.6 HAL_MCPWM_IrqFailClear()

```
void HAL_MCPWM_IrqFailClear (
    HAL_MCPWM_TypeDef * mcpwm_x )
```

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 517 行定义.

4.41.2.7 HAL_MCPWM_IrqFailDisable()

```
void HAL_MCPWM_IrqFailDisable (
    HAL_MCPWM_TypeDef * mcpwm_x )
```

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 511 行定义.

4.41.2.8 HAL_MCPWM_IrqFailEnable()

```
void HAL_MCPWM_IrqFailEnable (
    HAL_MCPWM_TypeDef * mcpwm_x )
```

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 505 行定义.

4.41.2.9 HAL_MCPWM_IrqOverflow()

```
uint8_t HAL_MCPWM_IrqOverflow (
    HAL_MCPWM_TypeDef * mcpwm_x )
```

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 438 行定义.

4.41.2.10 HAL_MCPWM_IrqOverflowClear()

```
void HAL_MCPWM_IrqOverflowClear (
    HAL_MCPWM_TypeDef * mcpwm_x )
```

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 461 行定义.

4.41.2.11 HAL_MCPWM_IrqOverflowDisable()

```
void HAL_MCPWM_IrqOverflowDisable (
    HAL_MCPWM_TypeDef * mcpwm_x )
```

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 455 行定义.

4.41.2.12 HAL_MCPWM_IrqOverflowEnable()

```
void HAL_MCPWM_IrqOverflowEnable (
    HAL_MCPWM_X mcpwm_x )
```

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 449 行定义.

4.41.2.13 HAL_MCPWM_IrqZeroCross()

```
uint8_t HAL_MCPWM_IrqZeroCross (
    HAL_MCPWM_X mcpwm_x )
```

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 466 行定义.

4.41.2.14 HAL_MCPWM_IrqZeroCrossClear()

```
void HAL_MCPWM_IrqZeroCrossClear (
    HAL_MCPWM_X mcpwm_x )
```

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 489 行定义.

4.41.2.15 HAL_MCPWM_IrqZeroCrossDisable()

```
void HAL_MCPWM_IrqZeroCrossDisable (
    HAL_MCPWM_X mcpwm_x )
```

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 483 行定义.

4.41.2.16 HAL_MCPWM_IrqZeroCrossEnable()

```
void HAL_MCPWM_IrqZeroCrossEnable (
    HAL_MCPWM_X mcpwm_x )
```

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 477 行定义.

4.41.2.17 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_lks32mc09x_mcpwm.c` 第 158 行定义.

4.41.2.18 HAL_MCPWM_SetAdcTrig()

```
void HAL_MCPWM_SetAdcTrig (
    HAL_MCPWM_X mcpwm_x,
    int32_t t0,
    int32_t t1 )
```

在文件 `hal_lks32mc09x_mcpwm.c` 第 422 行定义.

4.41.2.19 HAL_MCPWM_SetAdcTrigQ15()

```
void HAL_MCPWM_SetAdcTrigQ15 (
    HAL_MCPWM_X mcpwm_x,
    int32_t t0,
    int32_t t1 )
```

在文件 `hal_lks32mc09x_mcpwm.c` 第 428 行定义.

函数调用图:



4.41.2.20 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_lks32mc09x_mcpwm.c` 第 242 行定义.

函数调用图:



4.41.2.21 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

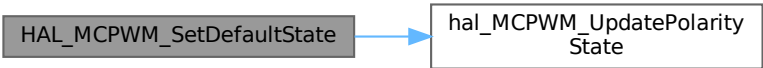
参数

mcpwm_x	操作的模块编号
mcpwm_default_output_state_x	默认输出状态

bit0 通道交换使能 0 交换 1 交换 bit1 上管有效电平 0 上管低有效 1 上管高有效 bit2 下管有效电平 0 下管低有效 1 下管高有效 bit3 默认上管电平 0 关机后上管关闭 1 关机后上管打开 bit4 默认下管电平 0 关机后下管关闭 1 关机后下管打开 bit5-bit7 输出状态 HAL_MCPWM_OUTPUT_STATE_x

在文件 [hal_lks32mc09x.mcpwm.c](#) 第 362 行定义.

函数调用图:



4.41.2.22 HAL_MCPWM_SetDrivePolarity()

```
void HAL_MCPWM_SetDrivePolarity (
    HAL_MCPWM_x mcpwm_x,
    HAL_MCPWM_POLARITY_x mcpwm_polarity_x )
```

设置pwm极性

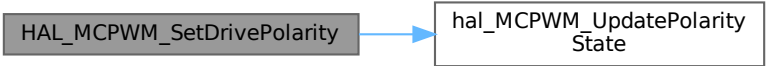
参数

<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

bit0 通道交换使能 0 交换 1 交换 bit1 上管有效电平 0 上管低有效 1 上管高有效 bit2 下管有效电平 0 下管低有效 1 下管高有效 bit3 默认上管电平 0 关机后上管关闭 1 关机后上管打开 bit4 默认下管电平 0 关机后下管关闭 1 关机后下管打开 bit5-bit7 输出状态 HAL_MCPWM_OUTPUT_STATE_x

在文件 [hal_lks32mc09x.mcpwm.c](#) 第 338 行定义.

函数调用图:



4.41.2.23 HAL_MCPWM_SetFailSignal()

```
void HAL_MCPWM_SetFailSignal (
    HAL_MCPWM_x mcpwm_x,
    HAL_MCPWM_FAIL_x fail_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

在文件 [hal_lks32mc09x.mcpwm.c](#) 第 416 行定义.

4.41.2.24 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_lks32mc09x_mcpwm.c](#) 第 207 行定义.

函数调用图:

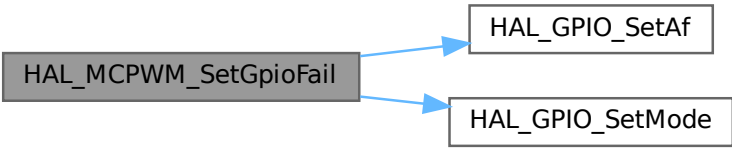


4.41.2.25 HAL_MCPWM_SetGpioFail()

```
void HAL_MCPWM_SetGpioFail (
    HAL_MCPWM_ x mcpwm_x,
    HAL_GPIO_Port px )
```

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 404 行定义.

函数调用图:



4.41.2.26 HAL_MCPWM_SetGpioOut()

```
void HAL_MCPWM_SetGpioOut (
    HAL_MCPWM_ x mcpwm_x,
    HAL_GPIO_Port px )
```

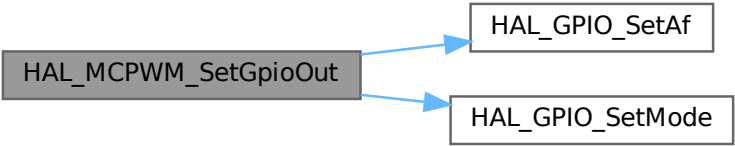
设置用于输出的gpio

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_lks32mc09x_mcpwm.c](#) 第 398 行定义.

函数调用图:



4.41.2.27 HAL_MCPWM_SetOutputDisable()

```
void HAL_MCPWM_SetOutputDisable (
    HAL_MCPWM_ x mcpwm_x )
```

关闭输出

参数

<i>mcpwm</i> _{<i>_x</i>}	操作的模块编号

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 257 行定义.

4.41.2.28 HAL_MCPWM_SetOutputEnable()

```
void HAL_MCPWM_SetOutputEnable (
    HAL_MCPWM_ x mcpwm_x )
```

打开输出

参数

<i>mcpwm</i> _{<i>_x</i>}	操作的模块编号

在文件 `hal_lks32mc09x_mcpwm.c` 第 267 行定义.

4.41.2.29 HAL.MCPWM.SetOutState()

```
void HALMCPWM.SetOutState (
    HALMCPWM_x mcpwm_x,
    HALMCPWM.PORT_x mcpwm_port_x,
    HALMCPWM.OUTPUT_STATE_x mcpwm_output_state_x )
```

设置输出状态

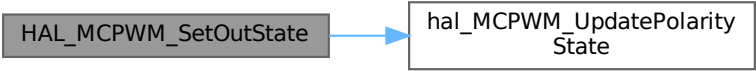
参数

<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

bit0 通道交换使能 0 交换 1 交换 bit1 上管有效电平 0 上管低有效 1 上管高有效 bit2 下管有效电平 0 下管低有效 1 下管高有效 bit3 默认上管电平 0 关机后上管关闭 1 关机后上管打开 bit4 默认下管电平 0 关机后下管关闭 1 关机后下管打开 bit5-bit7 输出状态 HAL.MCPWM.OUTPUT.STATE_x

在文件 `hal_lks32mc09x_mcpwm.c` 第 319 行定义.

函数调用图:



4.41.2.30 HAL.MCPWM.SetPwm()

```
void HALMCPWM.SetPwm (
    HALMCPWM_x mcpwm_x,
    HALMCPWM.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_lks32mc09x_mcpwm.c](#) 第 300 行定义.

这是这个函数的调用关系图:

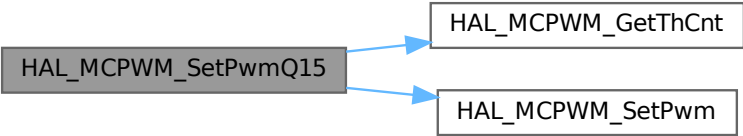


4.41.2.31 HAL.MCPWM.SetPwmQ15()

```
void HAL_MCPWM_SetPwmQ15 (
    HAL_MCPWM_TypeDef* mcpwm_x,
    HAL_MCPWM_Port_TypeDef* mcpwm_port_x,
    int16_t t0,
    int16_t t1 )
```

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 306 行定义.

函数调用图:



4.41.2.32 HAL.MCPWM.StartCounting()

```
void HAL_MCPWM_StartCounting (
    HAL_MCPWM_TypeDef* mcpwm_x )
```

开始计数

参数

<i>mcpwm_x</i>	操作的模块编号

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 277 行定义.

4.41.2.33 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

<i>mcpwm_x</i>	操作的模块编号

在文件 `hal_lks32mc09x_mcpwm.c` 第 287 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"
00005 typedef enum
00006 {
00007     HAL_MCPWM_0,
00008     HAL_MCPWM_1,
00009 } HAL_MCPWM_x;
00010
00011 typedef enum
00012 {
00013     HAL_MCPWM_PORT_0,
00014     HAL_MCPWM_PORT_1,
00015     HAL_MCPWM_PORT_2,
00016     HAL_MCPWM_PORT_3,
00017     HAL_MCPWM_PORT_4,
00018     HAL_MCPWM_PORT_5,
00019 } HAL_MCPWM_PORT_x;
00020 typedef enum
00021 {
00022     HAL_MCPWM_POLARITY_PL_NL,
00023     HAL_MCPWM_POLARITY_PH_NL,
00024     HAL_MCPWM_POLARITY_PL_NH,
00025     HAL_MCPWM_POLARITY_PH_NH,
00026 } HAL_MCPWM_POLARITY_x;
00027 typedef enum
00028 {
00029     HAL_MCPWM_OUTPUT_STATE_PWM,
00030     HAL_MCPWM_OUTPUT_STATE_OFF,
00031     HAL_MCPWM_OUTPUT_STATE_LOW,
00032     HAL_MCPWM_OUTPUT_STATE_HIGH,
00033     HAL_MCPWM_OUTPUT_STATE_LOW_PWM,
00034     HAL_MCPWM_OUTPUT_STATE_HIGH_PWM,
00035 } HAL_MCPWM_OUTPUT_STATE_x;
00036 typedef enum
00037 {
00038     HAL_MCPWM_DEFAULT_OUTPUT_STATE_OFF,
00039     HAL_MCPWM_DEFAULT_OUTPUT_STATE_LOW,
00040     HAL_MCPWM_DEFAULT_OUTPUT_STATE_HIGH,
00041 } HAL_MCPWM_DEFAULT_OUTPUT_STATE_x;
00042 typedef enum
00043 {
00044     HAL_MCPWM_FAIL_IO0_HIGH,
00045     HAL_MCPWM_FAIL_IO1_HIGH,
00046     HAL_MCPWM_FAIL_CMP0_HIGH,
00047     HAL_MCPWM_FAIL_CMP1_HIGH,
00048     HAL_MCPWM_FAIL_IO0_LOW,
00049     HAL_MCPWM_FAIL_IO1_LOW,
00050     HAL_MCPWM_FAIL_CMP0_LOW,
00051     HAL_MCPWM_FAIL_CMP1_LOW,
00052 } HAL_MCPWM_FAIL_x;
00053 // 初始化
00054 void HAL_MCPWM_Reset (HAL_MCPWM_x); // 模块复位
00055 void HAL_MCPWM_Enable (HAL_MCPWM_x); // 模块打开
```

```

00056 void HAL_MCPWM_Disable(HAL_MCPWM_x); // 模块关闭
00057
00058 void HAL_MCPWM_SetFreq(HAL_MCPWM_x, uint32_t freq); // 设置pwm频率
00059 uint32_t HAL_MCPWM_GetFreq(HAL_MCPWM_x); // 获取当前的pwm频率
00060 uint32_t HAL_MCPWM_GetThCnt(HAL_MCPWM_x); // 获取计数器最大值
00061 void HAL_MCPWM_SetDead(HAL_MCPWM_x, uint32_t pdead_nS, uint32_t ndead_nS); // 设置死区
00062
00063 void HAL_MCPWM_SetOutputDisable(HAL_MCPWM_x); // 关闭输出
00064 void HAL_MCPWM_SetOutputEnable(HAL_MCPWM_x); // 打开输出
00065 void HAL_MCPWM_StartCounting(HAL_MCPWM_x); // 开始计数
00066 void HAL_MCPWM_StopCounting(HAL_MCPWM_x); // 停止计数(同时计数
器回到-th)
00067 void HAL_MCPWM_SetPwm(HAL_MCPWM_x, HAL_MCPWM_PORT_x, int32_t t0, int32_t t1); // 设置pwm频率
00068 void HAL_MCPWM_SetPwmQ15(HAL_MCPWM_x, HAL_MCPWM_PORT_x, int16_t t0, int16_t t1); // 设置pwm波形q15格式输入
00069 void HAL_MCPWM_SetOutState(HAL_MCPWM_x, HAL_MCPWM_PORT_x, HAL_MCPWM_OUTPUT_STATE_x); // 设置输出状态
00070 void HAL_MCPWM_SetAdcTrig(HAL_MCPWM_x, int32_t t0, int32_t t1); // 设置pwm输出ADC采样时间
00071 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
00072
00073 void HAL_MCPWM_SetDrivePolarity(HAL_MCPWM_x, HAL_MCPWM_POLARITY_x); // 设置pwm极性
00074 void HAL_MCPWM_SetDefaultState(HAL_MCPWM_x, HAL_MCPWM_DEFAULT_OUTPUT_STATE_x); // 设置默认电平
00075 void HAL_MCPWM_SetGpioOut(HAL_MCPWM_x, HAL_GPIO_Px); // 设置用于输出的gpio
00076 void HAL_MCPWM_SetGpioFail(HAL_MCPWM_x, HAL_GPIO_Px); // 设置用于故障输入的gpio
00077 void HAL_MCPWM_SetFailSignal(HAL_MCPWM_x, HAL_MCPWM_FAIL_x); // 设置FAIL信号来源
00078
00079 uint8_t HAL_MCPWM_IrqOverflow(HAL_MCPWM_x);
00080 void HAL_MCPWM_IrqOverflowEnable(HAL_MCPWM_x);
00081 void HAL_MCPWM_IrqOverflowDisable(HAL_MCPWM_x);
00082 void HAL_MCPWM_IrqOverflowClear(HAL_MCPWM_x);
00083
00084 uint8_t HAL_MCPWM_IrqZeroCross(HAL_MCPWM_x);
00085 void HAL_MCPWM_IrqZeroCrossEnable(HAL_MCPWM_x);
00086 void HAL_MCPWM_IrqZeroCrossDisable(HAL_MCPWM_x);
00087 void HAL_MCPWM_IrqZeroCrossClear(HAL_MCPWM_x);
00088
00089 uint8_t HAL_MCPWM_IrqFail(HAL_MCPWM_x);
00090 void HAL_MCPWM_IrqFailEnable(HAL_MCPWM_x);
00091 void HAL_MCPWM_IrqFailDisable(HAL_MCPWM_x);
00092 void HAL_MCPWM_IrqFailClear(HAL_MCPWM_x);
00093 #endif

```

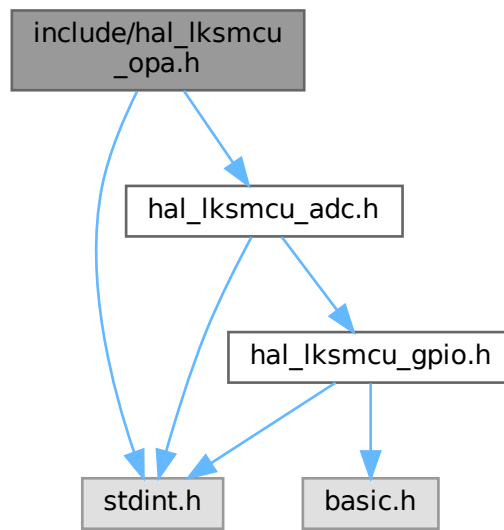
4.43 include/hal_lksmcu_opa.h 文件参考

```

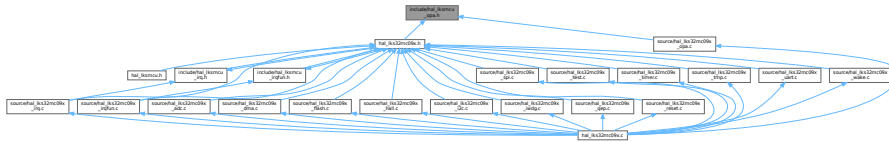
#include "stdint.h"
#include "hal_lksmcu_adc.h"

```

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_PORT_0 : (HAL_ADC_PORT_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_660_20 = 0, HAL_OPA_GAIN_640_40 = 1, HAL_OPA_GAIN_320_40 = 2, HAL_OPA_GAIN_160_40 = 3, HAL_OPA_GAIN_HIGH = HAL_OPA_GAIN_640_40, HAL_OPA_GAIN_MEDIUM = HAL_OPA_GAIN_320_40, HAL_OPA_GAIN_LOW = HAL_OPA_GAIN_160_40, HAL_OPA_GAIN_200_10, HAL_OPA_GAIN_190_20, HAL_OPA_GAIN_180_30, HAL_OPA_GAIN_170_40, HAL_OPA_GAIN_200_10_6 }`

```
HAL_OPA_GAIN_190_20_6, HAL_OPA_GAIN_180_30_6, HAL_OPA_GAIN_170_40_6, HAL_OPA_GAIN_200_10_4,
,
HAL_OPA_GAIN_190_20_4, HAL_OPA_GAIN_180_30_4, HAL_OPA_GAIN_170_40_4, HAL_OPA_GAIN_20_10,
HAL_OPA_GAIN_40_10, HAL_OPA_GAIN_80_10, HAL_OPA_GAIN_160_10, HAL_OPA_GAIN_320_10,
HAL_OPA_GAIN_320_5 }
```

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_PORT_0 : (HAL_ADC_PORT_x)(OPAX))
```

在文件 `hal_lksmcu_opa.h` 第 65 行定义.

4.43.1.2 HAL_OPA.READ_GAIN

```
#define HAL_OPA.READ_GAIN(  
    GAIN_x,  
    res )
```

值:

```
== HAL_OPA_GAIN_HIGH ? (640.0f / (40.0f + (float)res)) : (GAIN_x == HAL_OPA_GAIN_MEDIUM) ? (320.0f /  
(40.0f + (float)res)) \  
(160.0f / (40.0f + (float)res))
```

芯片的运放放大倍数

在文件 `hal_lksmcu_opa.h` 第 69 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_660_20	
HAL_OPA_GAIN_640_40	
HAL_OPA_GAIN_320_40	
HAL_OPA_GAIN_160_40	
HAL_OPA_GAIN_HIGH	高增益
HAL_OPA_GAIN_MEDIUM	中等增益
HAL_OPA_GAIN_LOW	低增益
HAL_OPA_GAIN_200_10	
HAL_OPA_GAIN_190_20	
HAL_OPA_GAIN_180_30	
HAL_OPA_GAIN_170_40	
HAL_OPA_GAIN_200_↔ 10_6	
HAL_OPA_GAIN_190_↔ 20_6	
HAL_OPA_GAIN_180_↔ 30_6	
HAL_OPA_GAIN_170_↔ 40_6	
HAL_OPA_GAIN_200_↔ 10_4	
HAL_OPA_GAIN_190_↔ 20_4	
HAL_OPA_GAIN_180_↔ 30_4	
HAL_OPA_GAIN_170_↔ 40_4	
HAL_OPA_GAIN_20_10	
HAL_OPA_GAIN_40_10	
HAL_OPA_GAIN_80_10	
HAL_OPA_GAIN_160_10	
HAL_OPA_GAIN_320_10	
HAL_OPA_GAIN_320_5	

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_lks32mc09x_opa.c` 第 38 行定义.

4.43.3.2 HAL_OPA_Enable()

```
void HAL_OPA_Enable (
    HAL_OPA_x opa_x )
```

在文件 `hal_lks32mc09x_opa.c` 第 13 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

<i>opa_x</i>	运放序号

使能_OPAx输出信号送至IO口,或使能OPA_x.OUT输出信号送至CMP。0000:不输出到IO,输出OPA0.OUT正端到OPA_OUT_CMP; 0001:不输出到IO,输出OPA1.OUT正端到OPA_OUT_CMP; 0010:不输出到IO,输出OPA2.OUT正端到OPA_OUT_CMP; 0011:不输出到IO,输出OPA3.OUT正端到OPA_OUT_CMP; 0100:不输出到IO,输出OPA0.OUT负端到OPA_OUT_CMP; 0101:不输出到IO,输出OPA1.OUT负端到OPA_OUT_CMP; 0110:不输出到IO,输出OPA2.OUT负端到OPA_OUT_CMP; 0111:不输出到IO,输出OPA3.OUT负端到OPA_OUT_CMP; 1000:输出OPA0.OUT正端到IO P2.7口,同时输出OPA0.OUT正端到OPA_OUT_CMP; 1001:输出OPA1.OUT正端到IO P2.7口,同时输出OPA1.OUT正端到OPA_OUT_CMP; 1010:输出OPA2.OUT正端到IO P2.7口,同时输出OPA2.OUT正端到OPA_OUT_CMP; 1011:输出OPA3.OUT正端到IO P2.7口,同时输出OPA3.OUT正端到OPA_OUT_CMP; 1100:输出OPA0.OUT负端到IO P2.7口,同时输出OPA0.OUT负端到OPA_OUT_CMP; 1101:输出OPA1.OUT负端到IO P2.7口,同时输出OPA1.OUT负端到OPA_OUT_CMP; 1110:输出OPA2.OUT负端到IO P2.7口,同时输出OPA2.OUT负端到OPA_OUT_CMP; 1111:输出OPA3.OUT负端到IO P2.7口,同时输出OPA3.OUT负端到OPA_OUT_CMP; 即OPAOUT_EN[3]控制是否将OPA_x输出到IO,OPAOUT_EN[2:0]选择OPA_x 输出到OPA_OUT_CMP

在文件 `hal_lks32mc09x_opa.c` 第 137 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

<i>opa</i> _{<i>x</i>}	运放序号

使能_{OPA_x}输出信号送至IO口,或使能_{OPA_x.OUT}输出信号送至CMP。0000:不输出到IO,输出OPA0.OUT正端到OPA.OUT_CMP; 0001:不输出到IO,输出OPA1.OUT正端到OPA.OUT_CMP; 0010:不输出到IO,输出OPA2.OUT正端到OPA.OUT_CMP; 0011:不输出到IO,输出OPA3.OUT正端到OPA.OUT_CMP; 0100:不输出到IO,输出OPA0.OUT负端到OPA.OUT_CMP; 0101:不输出到IO,输出OPA1.OUT负端到OPA.OUT_CMP; 0110:不输出到IO,输出OPA2.OUT负端到OPA.OUT_CMP; 0111:不输出到IO,输出OPA3.OUT负端到OPA.OUT_CMP; 1000:输出OPA0.OUT正端到IO P2.7口,同时输出OPA0.OUT正端到OPA.OUT_CMP; 1001:输出OPA1.OUT正端到IO P2.7口,同时输出OPA1.OUT正端到OPA.OUT_CMP; 1010:输出OPA2.OUT正端到IO P2.7口,同时输出OPA2.OUT正端到OPA.OUT_CMP; 1011:输出OPA3.OUT正端到IO P2.7口,同时输出OPA3.OUT正端到OPA.OUT_CMP; 1100:输出OPA0.OUT负端到IO P2.7口,同时输出OPA0.OUT负端到OPA.OUT_CMP; 1101:输出OPA1.OUT负端到IO P2.7口,同时输出OPA1.OUT负端到OPA.OUT_CMP; 1110:输出OPA2.OUT负端到IO P2.7口,同时输出OPA2.OUT负端到OPA.OUT_CMP; 1111:输出OPA3.OUT负端到IO P2.7口,同时输出OPA3.OUT负端到OPA.OUT_CMP; 即OPAOUT_EN[3]控制是否将OPA_x输出到IO,OPAOUT_EN[2:0]选择OPA_x输出到OPA.OUT_CMP

在文件 [hal_lks32mc09x_opa.c](#) 第 92 行定义.

4.43.3.5 HAL_OPA_Reset()

```
void HAL_OPA_Reset (
    HAL_OPA_x opa_x )
```

在文件 [hal_lks32mc09x_opa.c](#) 第 8 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_lks32mc09x_opa.c](#) 第 63 行定义.

4.43.3.7 HAL_OPA_SetGpioIn()

```
void HAL_OPA_SetGpioIn (
    HAL_OPA_x opa_x,
    HAL_GPIO_Px px )
```

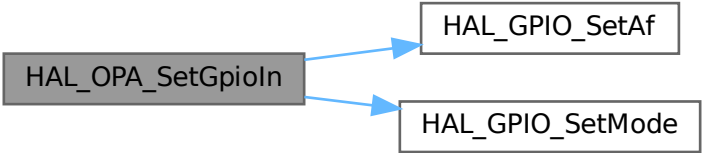
OPA设置IO输入

参数

<i>opa</i> _{<i>x</i>}	运放序号
<i>px</i>	IO序号

在文件 [hal_lks32mc09x_opa.c](#) 第 169 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 )
```

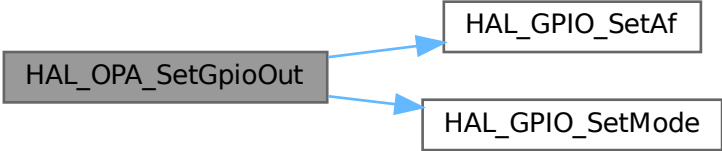
OPA设置IO输出

参数

<i>opa_x</i>	运放序号
<i>px</i>	IO序号

在文件 [hal_lks32mc09x_opa.c](#) 第 180 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     // 09
00019     HAL_OPA_GAIN_660_20 = 0,
00020     HAL_OPA_GAIN_640_40 = 1,
00021     HAL_OPA_GAIN_320_40 = 2,
00022     HAL_OPA_GAIN_160_40 = 3,
00023
00024     HAL_OPA_GAIN_HIGH = HAL_OPA_GAIN_640_40,
00025     HAL_OPA_GAIN_MEDIUM = HAL_OPA_GAIN_320_40,
00026     HAL_OPA_GAIN_LOW = HAL_OPA_GAIN_160_40,
00027
00028     // 03
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     // 05
00034     HAL_OPA_GAIN_200_10_6, // 不支持
00035     HAL_OPA_GAIN_190_20_6,
00036     HAL_OPA_GAIN_180_30_6,
00037     HAL_OPA_GAIN_170_40_6,
00038     // 08
00039     HAL_OPA_GAIN_200_10_4,
00040     HAL_OPA_GAIN_190_20_4,
00041     HAL_OPA_GAIN_180_30_4,
00042     HAL_OPA_GAIN_170_40_4,
00043
00044     // 45
00045     HAL_OPA_GAIN_20_10,
00046     HAL_OPA_GAIN_40_10,
00047     HAL_OPA_GAIN_80_10,
00048     HAL_OPA_GAIN_160_10,
00049     HAL_OPA_GAIN_320_10,
00050     HAL_OPA_GAIN_320_5,
00051
00052 } HAL_OPA_GAIN_x;
00053
00054 // 初始化
00055 void HAL_OPA_Reset(HAL_OPA_x); // 模块复位
00056 void HAL_OPA_Enable(HAL_OPA_x); // 模块打开
00057 void HAL_OPA_Disable(HAL_OPA_x); // 模块关闭
00058
00059 void HAL_OPA_SetGain(HAL_OPA_x, HAL_OPA_GAIN_x); // 设置增益
00060 void HAL_OPA_Out2GpioEnable(HAL_OPA_x);
00061 void HAL_OPA_Out2GpioDisable(HAL_OPA_x);
00062 void HAL_OPA_SetGpioIn(HAL_OPA_x, HAL_GPIO_Px);
00063 void HAL_OPA_SetGpioOut(HAL_OPA_x, HAL_GPIO_Px);
00064
00065 #define HAL_OPA_OPAX_2_ADCCHN(OPAX) ((OPAX) == HAL_OPA_0B ? HAL_ADC_PORT_0 : (HAL_ADC_PORT_x)(OPAX))
00069 #define HAL_OPA_READ_GAIN(GAIN_x, res) (GAIN_x == HAL_OPA_GAIN_HIGH ? (640.0f / (40.0f + (float)res)) :
    (GAIN_x == HAL_OPA_GAIN_MEDIUM) ? (320.0f / (40.0f + (float)res)) \
00070 : (160.0f / (40.0f + (float)res)))
00071
00072 #endif

```

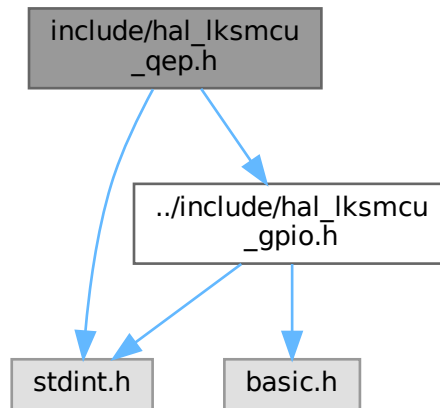
4.45 include/hal_lksmcu_qep.h 文件参考

```

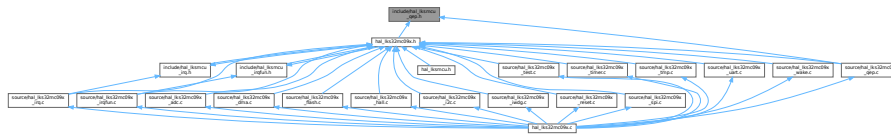
#include <stdint.h>
#include "../include/hal_lksmcu_gpio.h"

```

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_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_lks32mc09x_qep.c](#) 第 52 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_lks32mc09x_qep.c](#) 第 32 行定义.

函数调用图:



4.45.2.3 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_lks32mc09x_qep.c](#) 第 105 行定义.

4.45.2.4 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_lks32mc09x.qep.c](#) 第 13 行定义.

4.45.2.5 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_lks32mc09x.qep.c](#) 第 114 行定义.

4.45.2.6 HAL_QEP_SetGpio()

```
void HAL_QEP_SetGpio (
    HAL_QEP_x qepx,
    HAL_GPIO_Px px )
```

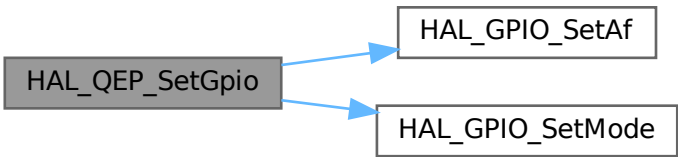
设置QEP模块的引脚

参数

<i>qepx</i>	QEP模块
<i>px</i>	引脚

在文件 [hal_lks32mc09x.qep.c](#) 第 124 行定义.

函数调用图:



4.45.2.7 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_lks32mc09x_qep.c](#) 第 82 行定义.

4.45.2.8 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_lks32mc09x_qep.c](#) 第 72 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
00028 void HAL_QEP_SetGpio (HAL_QEP_x, HAL_GPIO_Px);
00029
00030 #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_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_POS_MASK(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_INVERT(reg, mask) reg = ((reg) & ~(mask)) | ((~((reg) & (mask))) & (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 18 行定义.

4.47.2.2 HAL_REG_INVERT

```
#define HAL_REG_INVERT(  
    reg,  
    mask ) reg = ((reg) & ~(mask)) | ((~((reg) & (mask))) & (mask))
```

在文件 [hal_lksmcu_reg.h](#) 第 15 行定义.

4.47.2.3 HAL_REG_READ

```
#define HAL_REG_READ(  
    reg ) (reg)
```

在文件 [hal_lksmcu_reg.h](#) 第 8 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 12 行定义.

4.47.2.5 HAL_REG_READ_BIT_POS_MASK

```
#define HAL_REG_READ_BIT_POS_MASK(  
    reg,  
    mask ) (((reg) & (mask##_MASK)) >> mask##_POS)
```

在文件 [hal_lksmcu_reg.h](#) 第 11 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 14 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 13 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 16 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 17 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 7 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 9 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_lksmcu_reg.h](#) 第 10 行定义。

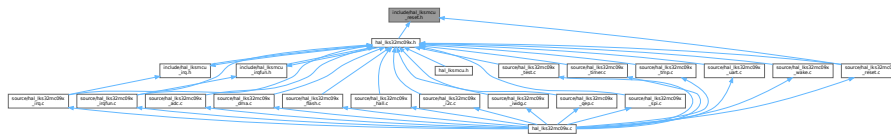
4.48 hal_lksmcu_reg.h

[浏览该文件的文档。](#)

```
00001 #ifndef __HAL_LKSMCU_REG_H__
00002 #define __HAL_LKSMCU_REG_H__
00007 #define HAL_REG_WRITE(reg, mask)          reg = (mask)
00008 #define HAL_REG_READ(reg)                (reg)
00009 #define HAL_REG_WRITE_BIT(reg, mask, val)  reg = ((reg) & ~(mask)) | ((val) & (mask))
00010 #define HAL_REG_WRITE_BIT_MASK_POS(reg, name, val)  reg = ((reg) & ~(name##_MASK)) | (name(val))
00011 #define HAL_REG_READ_BIT_POS_MASK(reg, mask)  (((reg) & (mask##_MASK)) >> mask##_POS)
00012 #define HAL_REG_READ_BIT(reg, mask)          ((reg) & (mask))
00013 #define HAL_REG_SET(reg, mask)              reg |= (mask)
00014 #define HAL_REG_RESET(reg, mask)            reg &= ~(mask)
00015 #define HAL_REG_INVERT(reg, mask)           reg = ((reg) & ~(mask)) | ((~((reg) & (mask))) & (mask))
00016 #define HAL_REG_SYS_LOCK()                 SYS_WR_PROTECT = 0
00017 #define HAL_REG_SYS_UNLOCK()               SYS_WR_PROTECT = 0x7a83
00018 #define HAL_REG_IF_WRITE(reg, mask, condition)  reg = (condition) ? ((reg) | (mask)) : ((reg) & ~(mask))
00019
00020 #endif
```

4.49 include/hal_lksmcu_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_lks32mc09x_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_lks32mc09x_reset.c](#) 第 29 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_lks32mc09x.reset.c](#) 第 17 行定义.

这是这个函数的调用关系图:



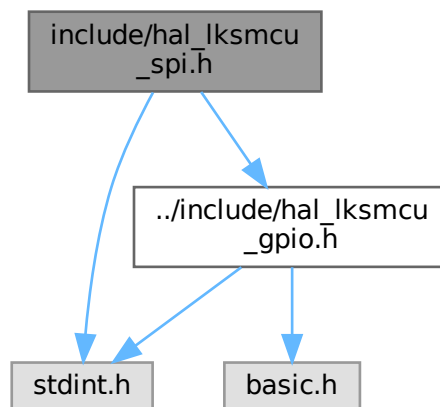
4.50 hal_lksmcu_reset.h

[浏览该文件的文档.](#)

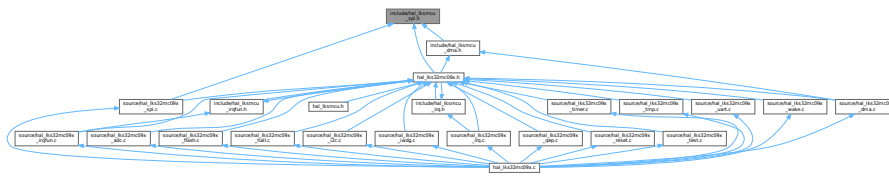
```
00001 #ifndef __HAL_LKSMCU_RESET_H__
00002 #define __HAL_LKSMCU_RESET_H__
00003 void HAL_RESET_ResetCore(void); // 复位内核
00004 void HAL_RESET_ResetRegister(void); // 寄存器复位
00005 void HAL_RESET_ResetGlobal(void); // 全局复位 (先复位寄存器再复位内核)
00006 #endif
```

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_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
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_lks32mc09x_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_lks32mc09x_spi.c](#) 第 25 行定义.

4.51.2.3 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_lks32mc09x_spi.c](#) 第 12 行定义.

4.51.2.4 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_lks32mc09x_spi.c](#) 第 257 行定义.

4.51.2.5 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

返回
接收状态

在文件 [hal_lks32mc09x_spi.c](#) 第 332 行定义.

4.51.2.6 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_lks32mc09x_spi.c](#) 第 284 行定义.

4.51.2.7 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

返回

发送和接收状态

在文件 [hal_lks32mc09x_spi.c](#) 第 357 行定义.

4.51.2.8 HAL_SPI_SetBaud()

```
void HAL_SPI_SetBaud (
    HAL_SPI_TypeDef * spi,
    uint32_t baud )
```

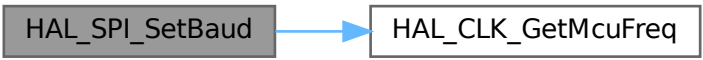
设置波特率

参数

<i>spi</i>	SPI模块号

在文件 [hal_lks32mc09x_spi.c](#) 第 49 行定义.

函数调用图:



4.51.2.9 HAL_SPI_SetGpioClk()

```
void HAL_SPI_SetGpioClk (  
    HAL_SPI_x spi,  
    HAL_GPIO_Px pin )
```

初始化Clk引脚

参数

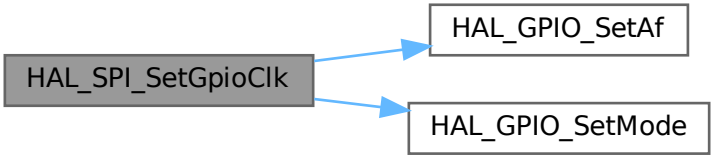
<i>spi</i>	SPI模块号
<i>pin</i>	GPIO引脚

参见

需要先初始化SPI

在文件 [hal_lks32mc09x_spi.c](#) 第 186 行定义.

函数调用图:



4.51.2.10 HAL_SPI_SetGpioCs()

```
void HAL_SPI_SetGpioCs (
    HAL_SPI_TypeDef* spi,
    GPIO_TypeDef* pin )
```

初始化CS引脚

参数

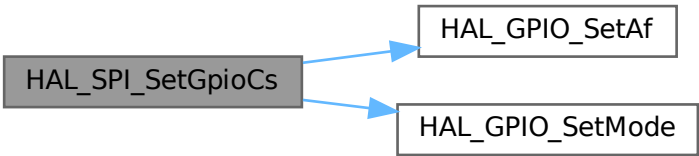
<i>spi</i>	SPI模块号
<i>pin</i>	GPIO引脚

参见

需要先初始化SPI

在文件 [hal_lks32mc09x_spi.c](#) 第 123 行定义.

函数调用图:



4.51.2.11 HAL_SPI_SetGpioDi()

```
void HAL_SPI_SetGpioDi (
    HAL_SPI_TypeDef* spi,
    GPIO_TypeDef* pin )
```

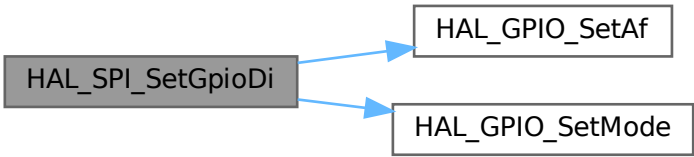
初始化SPI的输入引脚

参数

<i>spi</i>	SPI模块号
<i>pin</i>	GPIO引脚

在文件 [hal_lks32mc09x_spi.c](#) 第 206 行定义.

函数调用图:



4.51.2.12 HAL_SPI.SetGpioDo()

```
void HAL_SPI_SetGpioDo (
    HAL_SPI_x spi,
    HAL_GPIO_Px pin )
```

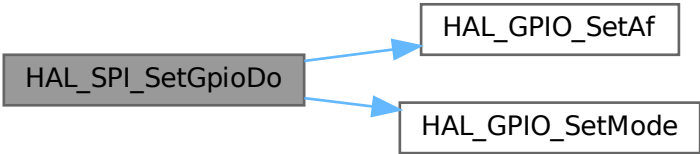
初始化SPI的输出引脚

参数

<i>spi</i>	SPI模块号
<i>pin</i>	GPIO引脚

在文件 [hal_lks32mc09x_spi.c](#) 第 217 行定义.

函数调用图:



4.51.2.13 HAL_SPI.SetGpioMiso()

```
void HAL_SPI_SetGpioMiso (
    HAL_SPI_x spi,
    HAL_GPIO_Px pin )
```

初始化Miso引脚

参数

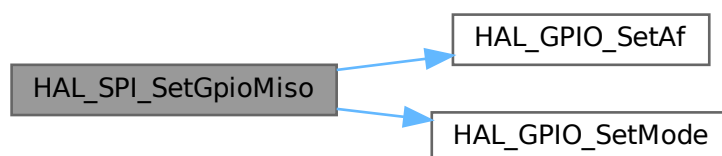
<i>spi</i>	SPI模块号
<i>pin</i>	GPIO引脚

参见

需要先初始化SPI

在文件 [hal_lks32mc09x_spi.c](#) 第 165 行定义.

函数调用图:



4.51.2.14 HAL_SPI.SetGpioMosi()

```
void HAL_SPI_SetGpioMosi (
    HAL_SPI_x spi,
    HAL_GPIO_Px pin )
```

初始化Mosi引脚

参数

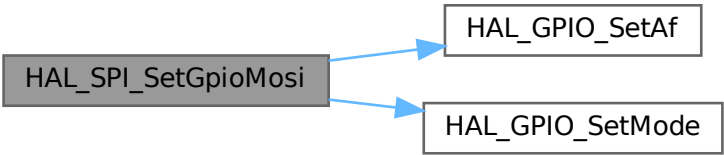
<i>spi</i>	SPI模块号
<i>pin</i>	GPIO引脚

参见

需要先初始化SPI

在文件 [hal_lks32mc09x_spi.c](#) 第 144 行定义.

函数调用图:



4.51.2.15 HAL_SPI.SetMode()

```
void HAL_SPI_SetMode (
    HAL_SPI_x spi,
    HAL_SPI_MODE_x 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_lks32mc09x_spi.c](#) 第 60 行定义.

4.51.2.16 HAL_SPI.TxData()

```
uint8_t HAL_SPI_TxData (
    HAL_SPI_x 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_lks32mc09x_spi.c](#) 第 231 行定义.

4.51.2.17 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

返回

发送状态

在文件 [hal_lks32mc09x_spi.c](#) 第 309 行定义。

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_TypeDef;
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_TypeDef;
00029 // 初始化
00030 void HAL_SPI_Reset(HAL_SPI_TypeDef* spi); // 模块复位
00031 void HAL_SPI_Enable(HAL_SPI_TypeDef* spi); // 模块打开
00032 void HAL_SPI_Disable(HAL_SPI_TypeDef* spi); // 模块关闭
00033
00034 void HAL_SPI_SetBaud(HAL_SPI_TypeDef* spi, uint32_t baud);
00035 void HAL_SPI_SetMode(HAL_SPI_TypeDef* spi, HAL_SPI_MODE_TypeDef mode);
00036
00037 void HAL_SPI_SetGpioCs(HAL_SPI_TypeDef* spi, HAL_GPIO_TypeDef* pin);
00038 void HAL_SPI_SetGpioMosi(HAL_SPI_TypeDef* spi, HAL_GPIO_TypeDef* pin);
00039 void HAL_SPI_SetGpioMiso(HAL_SPI_TypeDef* spi, HAL_GPIO_TypeDef* pin);
00040 void HAL_SPI_SetGpioClk(HAL_SPI_TypeDef* spi, HAL_GPIO_TypeDef* pin);
00041
00042 void HAL_SPI_SetGpioDi(HAL_SPI_TypeDef* spi, HAL_GPIO_TypeDef* pin); // 不自动识别模式，直接指定输入或输出状态
```

```

00043 void HAL_SPI_SetGpioDo(HAL_SPI_t, HAL_GPIO_t pin); // 不自动识别模式, 直接指定输入或输出状态
00044
00045 uint8_t HAL_SPI_TxData(HAL_SPI_t, uint8_t *data, uint32_t len);
00046 uint8_t HAL_SPI_RxData(HAL_SPI_t, uint8_t *data, uint32_t len);
00047 uint8_t HAL_SPI_RxTxData(HAL_SPI_t, uint8_t *rxdata, uint8_t *txdata, uint32_t len);
00048 uint8_t HAL_SPI_TxDataWidth16(HAL_SPI_t, uint16_t *data, uint32_t len);
00049 uint8_t HAL_SPI_RxDataWidth16(HAL_SPI_t, uint16_t *data, uint32_t len);
00050 uint8_t HAL_SPI_RxTxDataWidth16(HAL_SPI_t, uint16_t *rxdata, uint16_t *txdata, uint32_t len);
00051 #endif

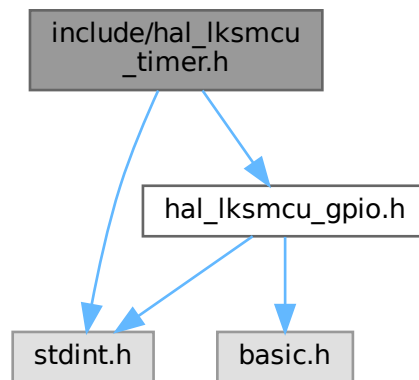
```

4.53 include/hal_lksmcu_timer.h 文件参考

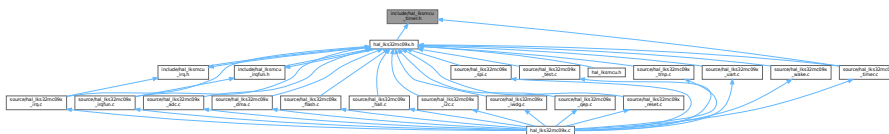
```
#include "stdint.h"
```

```
#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*)
- uint8_t [HAL_TIMER_IrqOverflow](#) ([HAL_TIMER_x](#))
- void [HAL_TIMER_IrqOverflowEnable](#) ([HAL_TIMER_x](#))
- void [HAL_TIMER_IrqOverflowDisable](#) ([HAL_TIMER_x](#))
- void [HAL_TIMER_IrqOverflowClear](#) ([HAL_TIMER_x](#))
- uint8_t [HAL_TIMER_IrqCompare0](#) ([HAL_TIMER_x](#))
- void [HAL_TIMER_IrqCompare0Enable](#) ([HAL_TIMER_x](#))
- void [HAL_TIMER_IrqCompare0Disable](#) ([HAL_TIMER_x](#))
- void [HAL_TIMER_IrqCompare0Clear](#) ([HAL_TIMER_x](#))
- uint8_t [HAL_TIMER_IrqCompare1](#) ([HAL_TIMER_x](#))
- void [HAL_TIMER_IrqCompare1Enable](#) ([HAL_TIMER_x](#))
- void [HAL_TIMER_IrqCompare1Disable](#) ([HAL_TIMER_x](#))
- void [HAL_TIMER_IrqCompare1Clear](#) ([HAL_TIMER_x](#))
- uint32_t [HAL_TIMER_GetFreq](#) ([HAL_TIMER_x](#))
- uint32_t [HAL_TIMER_GetThCnt](#) ([HAL_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 )
```

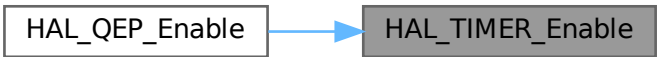
在文件 `hal_lks32mc09x_timer.c` 第 56 行定义.

4.53.2.2 HAL_TIMER_Enable()

```
void HAL_TIMER_Enable (
    HAL_TIMER_x timer_x )
```

在文件 `hal_lks32mc09x_timer.c` 第 34 行定义.

这是这个函数的调用关系图:



4.53.2.3 HAL_TIMER_GetFreq()

```
uint32_t HAL_TIMER_GetFreq (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc09x_timer.c](#) 第 157 行定义.

函数调用图:



4.53.2.4 HAL_TIMER_GetThCnt()

```
uint32_t HAL_TIMER_GetThCnt (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc09x_timer.c](#) 第 164 行定义.

这是这个函数的调用关系图:



4.53.2.5 HAL_TIMER_IrqCompare0()

```
uint8_t HAL_TIMER_IrqCompare0 (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc09x_timer.c](#) 第 267 行定义.

4.53.2.6 HAL_TIMER_IrqCompare0Clear()

```
void HAL_TIMER_IrqCompare0Clear (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc09x_timer.c](#) 第 289 行定义.

4.53.2.7 HAL_TIMER_IrqCompare0Disable()

```
void HAL_TIMER_IrqCompare0Disable (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc09x_timer.c](#) 第 285 行定义.

4.53.2.8 HAL_TIMER_IrqCompare0Enable()

```
void HAL_TIMER_IrqCompare0Enable (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc09x_timer.c](#) 第 281 行定义.

4.53.2.9 HAL_TIMER_IrqCompare1()

```
uint8_t HAL_TIMER_IrqCompare1 (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc09x_timer.c](#) 第 294 行定义.

4.53.2.10 HAL_TIMER_IrqCompare1Clear()

```
void HAL_TIMER_IrqCompare1Clear (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc09x_timer.c](#) 第 316 行定义.

4.53.2.11 HAL_TIMER_IrqCompare1Disable()

```
void HAL_TIMER_IrqCompare1Disable (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc09x_timer.c](#) 第 312 行定义.

4.53.2.12 HAL_TIMER_IrqCompare1Enable()

```
void HAL_TIMER_IrqCompare1Enable (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc09x_timer.c](#) 第 308 行定义.

4.53.2.13 HAL_TIMER_IrqOverflow()

```
uint8_t HAL_TIMER_IrqOverflow (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc09x_timer.c](#) 第 240 行定义.

4.53.2.14 HAL_TIMER_IrqOverflowClear()

```
void HAL_TIMER_IrqOverflowClear (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc09x_timer.c](#) 第 262 行定义.

4.53.2.15 HAL_TIMER_IrqOverflowDisable()

```
void HAL_TIMER_IrqOverflowDisable (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc09x_timer.c](#) 第 258 行定义.

4.53.2.16 HAL_TIMER_IrqOverflowEnable()

```
void HAL_TIMER_IrqOverflowEnable (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc09x_timer.c](#) 第 254 行定义.

4.53.2.17 HAL_TIMER_Reset()

```
void HAL_TIMER_Reset (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc09x_timer.c](#) 第 10 行定义.

4.53.2.18 HAL_TIMER_SetFreq()

```
void HAL_TIMER_SetFreq (
    HAL_TIMER_x timer_x,
    uint32_t freq )
```

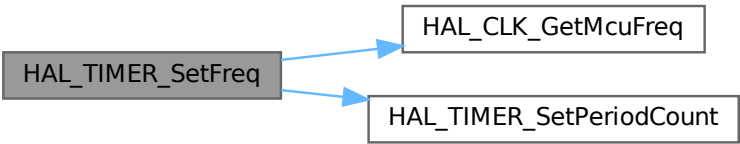
设置定时器频率

参数

<i>timer_x</i>	定时器编号
<i>freq</i>	定时器频率

在文件 [hal_lks32mc09x_timer.c](#) 第 127 行定义.

函数调用图:



4.53.2.19 HAL_TIMER_SetGpioIn()

```
void HAL_TIMER_SetGpioIn (
    HAL_TIMER_x timer_x,
    HAL_GPIO_Px px )
```

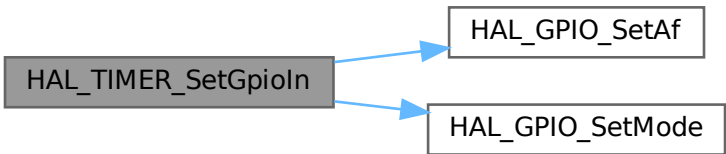
设置定时器输入gpio

参数

<i>timer_x</i>	定时器编号
<i>px</i>	gpio编号

在文件 [hal_lks32mc09x.timer.c](#) 第 355 行定义.

函数调用图:



4.53.2.20 HAL_TIMER_SetGpioOut()

```
void HAL_TIMER_SetGpioOut (
    HAL_TIMER_x timer_x,
    HAL_GPIO_Px px )
```

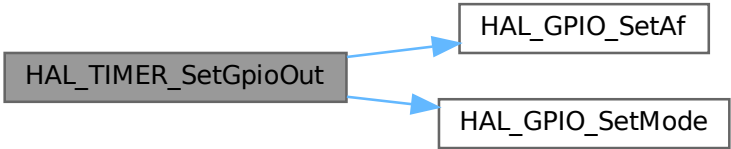
设置定时器输出gpio

参数

<i>timer</i> ↔ <i>_x</i>	定时器编号
<i>px</i>	gpio编号

在文件 [hal_lks32mc09x_timer.c](#) 第 326 行定义.

函数调用图:



4.53.2.21 HAL_TIMER.SetPeriodCount()

```
void HAL_TIMER.SetPeriodCount (
    HAL_TIMER_x timer_x,
    uint32_t count )
```

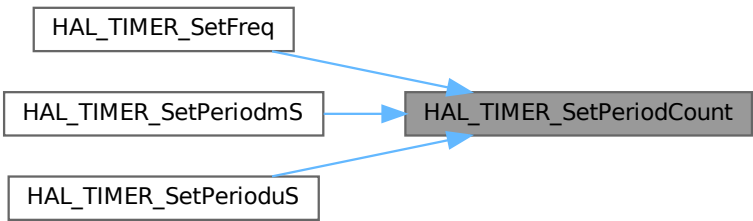
设置定时器周期

参数

<i>timer</i> ↔ <i>_x</i>	定时器编号
<i>count</i>	定时器周期

在文件 [hal_lks32mc09x_timer.c](#) 第 82 行定义.

这是这个函数的调用关系图:



4.53.2.22 HAL_TIMER_SetPeriodmS()

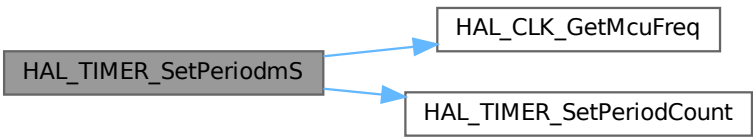
```
void HAL_TIMER.SetPeriodmS (
    HAL_TIMER.x timer_x,
    uint32_t ms )
```

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_lks32mc09x.timer.c](#) 第 139 行定义.
函数调用图:



4.53.2.23 HAL_TIMER_SetPerioduS()

```
void HAL_TIMER.SetPerioduS (
    HAL_TIMER.x timer_x,
    uint32_t us )
```

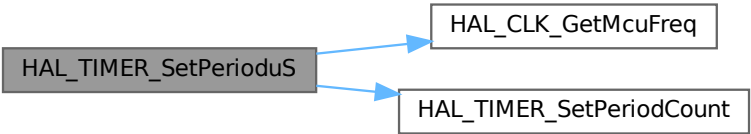
设置定时器周期(us)

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_lks32mc09x.timer.c](#) 第 150 行定义.

函数调用图:



4.53.2.24 HAL_TIMER_SetPwm()

```
void HAL_TIMER_SetPwm (
    HAL_TIMER_x timer_x,
    int32_t t0,
    int32_t t1 )
```

在文件 [hal_lks32mc09x.timer.c](#) 第 180 行定义.

4.53.2.25 HAL_TIMER_SetPwmQ15()

```
void HAL_TIMER_SetPwmQ15 (
    HAL_TIMER_x timer_x,
    int16_t t0,
    int16_t t1 )
```

在文件 [hal_lks32mc09x.timer.c](#) 第 186 行定义.

函数调用图:



4.53.2.26 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_lks32mc09x_timer.c](#) 第 195 行定义.

4.53.2.27 HAL_TIMER_StartCounting()

```
void HAL_TIMER_StartCounting (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc09x_timer.c](#) 第 169 行定义.

4.53.2.28 HAL_TIMER_StopCounting()

```
void HAL_TIMER_StopCounting (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc09x_timer.c](#) 第 174 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 uint8_t HAL_TIMER_IrqOverflow(HAL_TIMER_x);
00031 void HAL_TIMER_IrqOverflowEnable(HAL_TIMER_x);
```

```

00032 void HAL_TIMER_IrqOverflowDisable(HAL_TIMER_x);
00033 void HAL_TIMER_IrqOverflowClear(HAL_TIMER_x);
00034
00035 uint8_t HAL_TIMER_IrqCompare0(HAL_TIMER_x);
00036 void HAL_TIMER_IrqCompare0Enable(HAL_TIMER_x);
00037 void HAL_TIMER_IrqCompare0Disable(HAL_TIMER_x);
00038 void HAL_TIMER_IrqCompare0Clear(HAL_TIMER_x);
00039
00040 uint8_t HAL_TIMER_IrqCompare1(HAL_TIMER_x);
00041 void HAL_TIMER_IrqCompare1Enable(HAL_TIMER_x);
00042 void HAL_TIMER_IrqCompare1Disable(HAL_TIMER_x);
00043 void HAL_TIMER_IrqCompare1Clear(HAL_TIMER_x);
00044
00045 uint32_t HAL_TIMER_GetFreq(HAL_TIMER_x); // 获取当前的pwm频率
00046 uint32_t HAL_TIMER_GetThCnt(HAL_TIMER_x); // 获取计数器最大值
00047 void HAL_TIMER_StartCounting(HAL_TIMER_x); // 开始计数
00048 void HAL_TIMER_StopCounting(HAL_TIMER_x); // 停止计数 (同时计数器回到0)
00049 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
00050 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
00051 void HAL_TIMER_SetWorkMode(HAL_TIMER_x timer_x, HAL_TIMER_MODE_x mode_x); // 设置工作模式
00052
00053 void HAL_TIMER_SetGpioOut(HAL_TIMER_x timer_x, HAL_GPIO_Px px);
00054 void HAL_TIMER_SetGpioIn(HAL_TIMER_x timer_x, HAL_GPIO_Px px);
00055 #endif

```

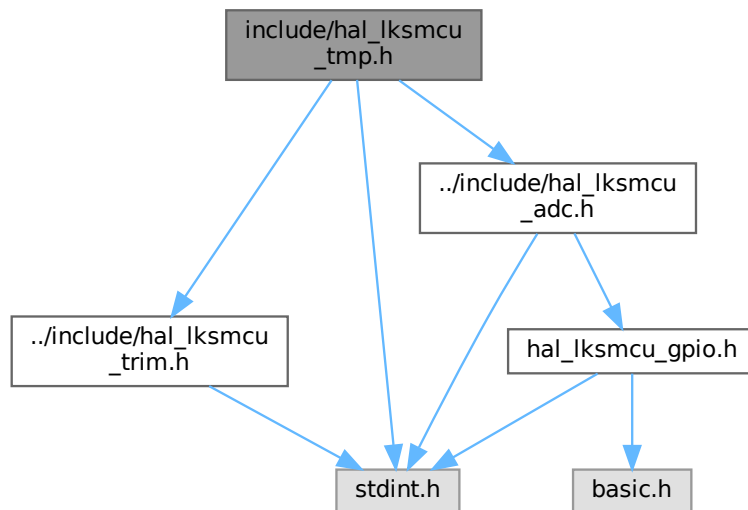
4.55 include/hal_lksmcu_tmp.h 文件参考

```

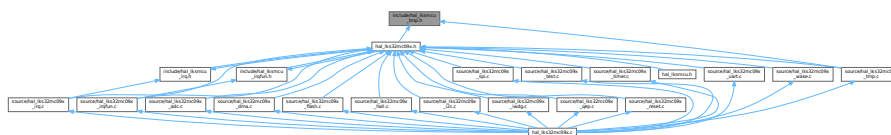
#include "../include/hal_lksmcu_trim.h"
#include "../include/hal_lksmcu_adc.h"
#include "stdint.h"

```

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

<code>HAL_↔ TMP_0</code>	
<code>HAL_↔ TMP_1</code>	

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_lks32mc09x_tmp.c](#) 第 84 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

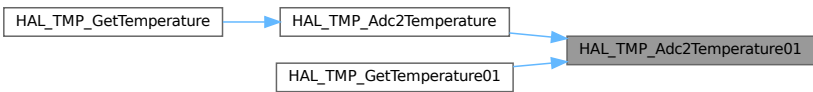
<i>TMP_x</i>	模块编号
<i>adc_value</i>	ADC值，左对齐（-32767 到32767）

返回

当前温度,单位0.1摄氏度

在文件 `hal_lks32mc09x.tmp.c` 第 94 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 )
```

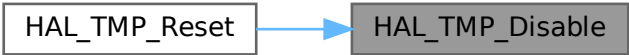
模块关闭

参数

<i>TMP_x</i>	模块编号

在文件 `hal_lks32mc09x.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 )
```

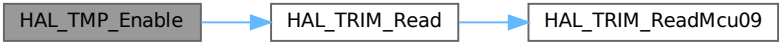
模块打开

参数

<i>TMP_x</i>	模块编号

在文件 [hal_lks32mc09x_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

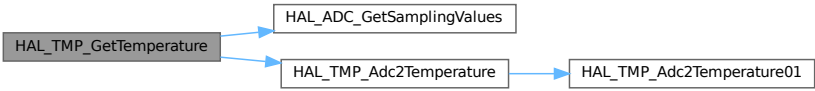
<i>TMP_x</i>	模块编号
<i>adc</i>	ADC编号
<i>sequence</i>	ADC采样序列

返回

当前温度,单位摄氏度

在文件 [hal_lks32mc09x_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

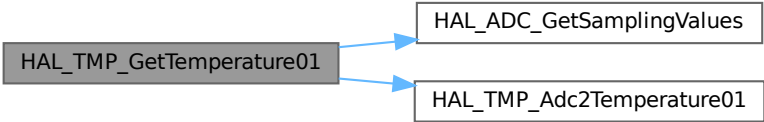
<i>TMP_x</i>	模块编号
<i>adc</i>	ADC编号
<i>sequence</i>	ADC采样序列

返回

当前温度,单位0.1摄氏度

在文件 [hal_lks32mc09x.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_x</i>	模块编号

在文件 [hal_lks32mc09x.tmp.c](#) 第 14 行定义.

函数调用图:



4.56 hal_ksmcu_tmp.h

[浏览该文件的文档.](#)

```

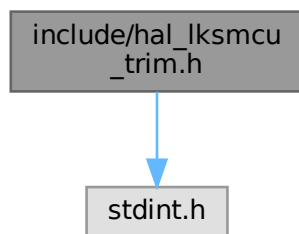
00001 #ifndef __HAL_LKSMCU_TMP_H__
00002 #ifndef __HAL_LKSMCU_TMP_H__
00003 #include "../include/hal_lksmcu_trim.h"
00004 #include "../include/hal_lksmcu_adc.h"
00005 #include "stdint.h"
00006 typedef enum
00007 {
00008     HAL_TMP_0,
00009     HAL_TMP_1,
00010 } HAL_TMP_x;
00011 // 初始化
00012 void HAL_TMP_Reset(HAL_TMP_x); // 模块复位
00013 void HAL_TMP_Enable(HAL_TMP_x); // 模块打开
00014 void HAL_TMP_Disable(HAL_TMP_x); // 模块关闭
00015
00016 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
00017 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
00019 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
00020
00021 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
00022 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
00023 #endif

```

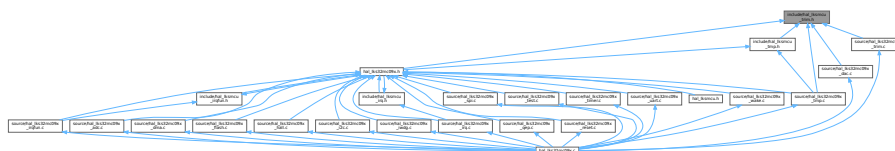
4.57 include/hal_iksmcu_trim.h 文件参考

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

hal_ksmcu_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 )
```

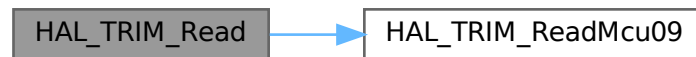
读取MCU03的校准参数

参数

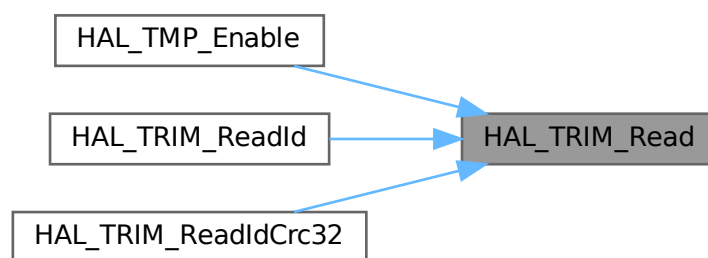
<i>addr</i>	校准参数的地址

在文件 [hal_lks32mc09x_trim.c](#) 第 92 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_lks32mc09x_trim.c](#) 第 102 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

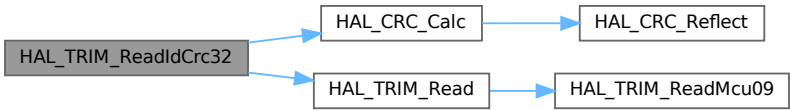
唯一ID的CRC32值

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_lks32mc09x_trim.c](#) 第 111 行定义.

函数调用图:



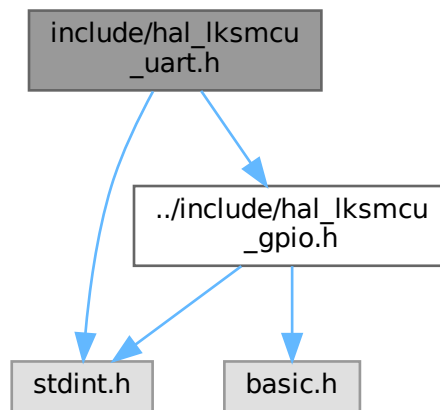
4.58 hal_lksmcu_trim.h

[浏览该文件的文档.](#)

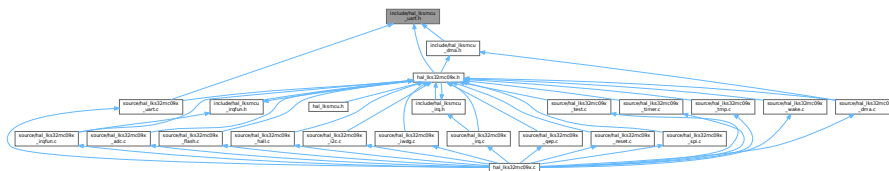
```
00001
00002 #ifndef __HAL_LKSMCU_TRIM_H__
00003 #define __HAL_LKSMCU_TRIM_H__
00004
00005 #include "stdint.h"
00006 uint32_t HAL_TRIM_Read(uint32_t addr);
00007 uint32_t HAL_TRIM_ReadId(uint32_t idx);
00008 uint32_t HAL_TRIM_ReadIdCrc32(void);
00009 #endif
```

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`, `HAL_UART_2` }

```

• 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 }

```

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_TxData (HAL_UART_x, uint8_t *data, uint32_t len)
• void HAL_UART_SetMode (HAL_UART_x, HAL_UART_MODE_x)
• 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)
• uint8_t HAL_UART_IrqSendOver (HAL_UART_x)
• void HAL_UART_IrqSendOverEnable (HAL_UART_x)
• void HAL_UART_IrqSendOverDisable (HAL_UART_x)
• void HAL_UART_IrqSendOverClear (HAL_UART_x)
• uint8_t HAL_UART_IrqReceiveOver (HAL_UART_x)
• void HAL_UART_IrqReceiveOverEnable (HAL_UART_x)
• void HAL_UART_IrqReceiveOverDisable (HAL_UART_x)
• void HAL_UART_IrqReceiveOverClear (HAL_UART_x)
• uint8_t HAL_UART_IrqStopErr (HAL_UART_x)
• void HAL_UART_IrqStopErrEnable (HAL_UART_x)
• void HAL_UART_IrqStopErrDisable (HAL_UART_x)
• void HAL_UART_IrqStopErrClear (HAL_UART_x)
• uint8_t HAL_UART_IrqCheckErr (HAL_UART_x)
• void HAL_UART_IrqCheckErrEnable (HAL_UART_x)
• void HAL_UART_IrqCheckErrDisable (HAL_UART_x)
• void HAL_UART_IrqCheckErrClear (HAL_UART_x)

```

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

枚举值

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 12 行定义.

4.59.1.2 HAL_UART_x

enum [HAL_UART_x](#)

枚举值

HAL_↔ UART_0	
HAL_↔ UART_1	
HAL_↔ UART_2	

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_lks32mc09x_uart.c](#) 第 80 行定义.

4.59.2.2 HAL_UART_Enable()

```
void HAL_UART_Enable (
    HAL\_UART\_x UART_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 47 行定义.

4.59.2.3 HAL_UART_IrqCheckErr()

```
uint8_t HAL_UART_IrqCheckErr (
    HAL\_UART\_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 316 行定义.

4.59.2.4 HAL_UART_IrqCheckErrClear()

```
void HAL_UART_IrqCheckErrClear (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 335 行定义.

4.59.2.5 HAL_UART_IrqCheckErrDisable()

```
void HAL_UART_IrqCheckErrDisable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 331 行定义.

4.59.2.6 HAL_UART_IrqCheckErrEnable()

```
void HAL_UART_IrqCheckErrEnable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 327 行定义.

4.59.2.7 HAL_UART_IrqReceiveOver()

```
uint8_t HAL_UART_IrqReceiveOver (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 266 行定义.

4.59.2.8 HAL_UART_IrqReceiveOverClear()

```
void HAL_UART_IrqReceiveOverClear (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 285 行定义.

4.59.2.9 HAL_UART_IrqReceiveOverDisable()

```
void HAL_UART_IrqReceiveOverDisable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 281 行定义.

4.59.2.10 HAL_UART_IrqReceiveOverEnable()

```
void HAL_UART_IrqReceiveOverEnable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 277 行定义.

4.59.2.11 HAL_UART_IrqSendOver()

```
uint8_t HAL_UART_IrqSendOver (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 241 行定义.

4.59.2.12 HAL_UART_IrqSendOverClear()

```
void HAL_UART_IrqSendOverClear (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 260 行定义.

4.59.2.13 HAL_UART_IrqSendOverDisable()

```
void HAL_UART_IrqSendOverDisable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 256 行定义.

4.59.2.14 HAL_UART_IrqSendOverEnable()

```
void HAL_UART_IrqSendOverEnable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 252 行定义.

4.59.2.15 HAL_UART_IrqStopErr()

```
uint8_t HAL_UART_IrqStopErr (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 291 行定义.

4.59.2.16 HAL_UART_IrqStopErrClear()

```
void HAL_UART_IrqStopErrClear (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 310 行定义.

4.59.2.17 HAL_UART_IrqStopErrDisable()

```
void HAL_UART_IrqStopErrDisable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 306 行定义.

4.59.2.18 HAL_UART_IrqStopErrEnable()

```
void HAL_UART_IrqStopErrEnable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 302 行定义.

4.59.2.19 HAL_UART_ReadBuffer()

```
uint8_t HAL_UART_ReadBuffer (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 219 行定义.

4.59.2.20 HAL_UART_Reset()

```
void HAL_UART_Reset (
    HAL_UART_x UART_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 11 行定义.

4.59.2.21 HAL_UART_SetBaud()

```
void HAL_UART_SetBaud (
    HAL_UART_x UART_x,
    uint32_t baud )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 112 行定义.

函数调用图:

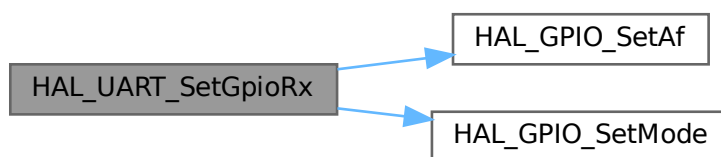


4.59.2.22 HAL_UART_SetGpioRx()

```
void HAL_UART_SetGpioRx (
    HAL_UART_x uart_x,
    HAL_GPIO_Px px )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 233 行定义.

函数调用图:

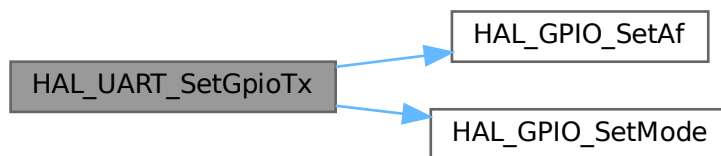


4.59.2.23 HAL_UART_SetGpioTx()

```
void HAL_UART_SetGpioTx (
    HAL_UART_x uart_x,
    HAL_GPIO_Px px )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 227 行定义.

函数调用图:



4.59.2.24 HAL_UART_SetMode()

```
void HAL_UART_SetMode (
    HAL_UART_TypeDef uart_x,
    HAL_UART_Mode_TypeDef 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_lks32mc09x_uart.c](#) 第 148 行定义.

4.59.2.25 HAL_UART_TxData()

```
void HAL_UART_TxData (
    HAL_UART_TypeDef UART_x,
    uint8_t * data,
    uint32_t len )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 136 行定义.

4.59.2.26 HAL_UART_WriteBuffer()

```
void HAL_UART_WriteBuffer (
    HAL_UART_TypeDef uart_x,
    uint8_t * buff )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 223 行定义.

4.60 hal_lksmcu_uart.h

[浏览该文件的文档.](#)

```

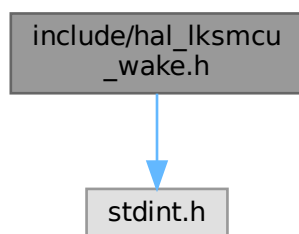
00001
00002 #ifndef __HAL_LKSMCU_UART_H__
00003 #define __HAL_LKSMCU_UART_H__
00004 #include "stdint.h"
00005 #include "../include/hal_lksmcu_gpio.h"
00006 typedef enum
00007 {
00008     HAL_UART_0,
00009     HAL_UART_1,
00010     HAL_UART_2,
00011 } HAL_UART_x;
00012 typedef enum
00013 {
00014     HAL_UART_MODE_LSB_STOP1_NONE,
00015     HAL_UART_MODE_LSB_STOP2_NONE,
00016     HAL_UART_MODE_LSB_STOP1_ODD,
00017     HAL_UART_MODE_LSB_STOP2_ODD,
00018     HAL_UART_MODE_LSB_STOP1_EVEN,
00019     HAL_UART_MODE_LSB_STOP2_EVEN,
00020     HAL_UART_MODE_MSB_STOP1_NONE,
00021     HAL_UART_MODE_MSB_STOP2_NONE,
00022     HAL_UART_MODE_MSB_STOP1_ODD,
00023     HAL_UART_MODE_MSB_STOP2_ODD,
00024     HAL_UART_MODE_MSB_STOP1_EVEN,
00025     HAL_UART_MODE_MSB_STOP2_EVEN,
00026 } HAL_UART_MODE_x;
00027 // 初始化
00028 void HAL_UART_Reset(HAL_UART_x); // 模块复位
00029 void HAL_UART_Enable(HAL_UART_x); // 模块打开
00030 void HAL_UART_Disable(HAL_UART_x); // 模块关闭
00031 //
00032 void HAL_UART_SetBaud(HAL_UART_x, uint32_t baud);
00033 void HAL_UART_TxData(HAL_UART_x, uint8_t *data, uint32_t len);
00034 void HAL_UART_SetMode(HAL_UART_x, HAL_UART_MODE_x);
00035 uint8_t HAL_UART_ReadBuffer(HAL_UART_x);
00036 void HAL_UART_WriteBuffer(HAL_UART_x, uint8_t buff);
00037 void HAL_UART_SetGpioTx(HAL_UART_x, HAL_GPIO_Px); // 设置用于输出的gpio
00038 void HAL_UART_SetGpioRx(HAL_UART_x, HAL_GPIO_Px); // 设置用于输出的gpio
00039
00040 // 中断
00041 // 仅支持发送完成、接受完成、停止位错误、校验错误
00042 //
00043 // 以下功能不提供接口, 如果需要使用, 可以使用标准库实现
00044 // 发送缓冲区空
00045 // 发送缓冲区溢出
00046 // 接受缓冲区溢出
00047 // LIN相关
00048
00049 // uart发送完成中断
00050 uint8_t HAL_UART_IrqSendOver(HAL_UART_x);
00051 void HAL_UART_IrqSendOverEnable(HAL_UART_x);
00052 void HAL_UART_IrqSendOverDisable(HAL_UART_x);
00053 void HAL_UART_IrqSendOverClear(HAL_UART_x);
00054
00055 // uart接受完成中断
00056 uint8_t HAL_UART_IrqReceiveOver(HAL_UART_x);
00057 void HAL_UART_IrqReceiveOverEnable(HAL_UART_x);
00058 void HAL_UART_IrqReceiveOverDisable(HAL_UART_x);
00059 void HAL_UART_IrqReceiveOverClear(HAL_UART_x);
00060
00061 // uart停止位错误中断
00062 uint8_t HAL_UART_IrqStopErr(HAL_UART_x);
00063 void HAL_UART_IrqStopErrEnable(HAL_UART_x);
00064 void HAL_UART_IrqStopErrDisable(HAL_UART_x);
00065 void HAL_UART_IrqStopErrClear(HAL_UART_x);
00066
00067 // uart校验错误中断
00068 uint8_t HAL_UART_IrqCheckErr(HAL_UART_x);
00069 void HAL_UART_IrqCheckErrEnable(HAL_UART_x);
00070 void HAL_UART_IrqCheckErrDisable(HAL_UART_x);
00071 void HAL_UART_IrqCheckErrClear(HAL_UART_x);
00072 #endif

```

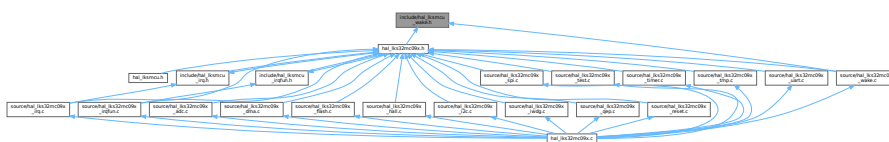
4.61 include/hal_lksmcu_wake.h 文件参考

```
#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_0.Low = 0x0001 , HAL_WAKE_UP_IO_0.High = 0x0101 , HAL_WAKE_UP_IO_1.Low = 0x0002 , HAL_WAKE_UP_IO_1.High = 0x0102 ,
 HAL_WAKE_UP_IO_2.Low = 0x0004 , HAL_WAKE_UP_IO_2.High = 0x0104 , HAL_WAKE_UP_IO_3.Low = 0x0008 , HAL_WAKE_UP_IO_3.High = 0x0108 ,
 HAL_WAKE_UP_IO_4.Low = 0x0010 , HAL_WAKE_UP_IO_4.High = 0x0110 , HAL_WAKE_UP_IO_5.Low = 0x0020 , HAL_WAKE_UP_IO_5.High = 0x0120 ,
 HAL_WAKE_UP_IO_6.Low = 0x0040 , HAL_WAKE_UP_IO_6.High = 0x0140 , HAL_WAKE_UP_IO_7.Low = 0x0080 , HAL_WAKE_UP_IO_7.High = 0x0180 }

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_0.Low	
HAL_WAKE_UP_IO_0.High	
HAL_WAKE_UP_IO_1.Low	
HAL_WAKE_UP_IO_1.High	
HAL_WAKE_UP_IO_2.Low	
HAL_WAKE_UP_IO_2.High	
HAL_WAKE_UP_IO_3.Low	
HAL_WAKE_UP_IO_3.High	
HAL_WAKE_UP_IO_4.Low	
HAL_WAKE_UP_IO_4.High	
HAL_WAKE_UP_IO_5.Low	
HAL_WAKE_UP_IO_5.High	
HAL_WAKE_UP_IO_6.Low	
HAL_WAKE_UP_IO_6.High	
HAL_WAKE_UP_IO_7.Low	
HAL_WAKE_UP_IO_7.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 )
```

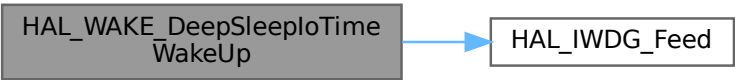
进入深度休眠,并且使用IO唤醒和定时唤醒

参数

<i>wake_up_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_lks32mc09x.wake.c` 第 147 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</i> <i>_io</i>	指定唤醒IO引脚及其电平

在文件 [hal_lks32mc09x.wake.c](#) 第 12 行定义.

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_lks32mc09x.wake.c](#) 第 84 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     HAL_WAKE_UP_IO_0_Low = 0x0001,
00008     HAL_WAKE_UP_IO_0_High = 0x0101,
00009     HAL_WAKE_UP_IO_1_Low = 0x0002,
00010     HAL_WAKE_UP_IO_1_High = 0x0102,
00011     HAL_WAKE_UP_IO_2_Low = 0x0004,
00012     HAL_WAKE_UP_IO_2_High = 0x0104,
00013     HAL_WAKE_UP_IO_3_Low = 0x0008,
```

```

00014     HAL_WAKE_UP_IO_3_High = 0x0108,
00015     HAL_WAKE_UP_IO_4_Low  = 0x0010,
00016     HAL_WAKE_UP_IO_4_High = 0x0110,
00017     HAL_WAKE_UP_IO_5_Low  = 0x0020,
00018     HAL_WAKE_UP_IO_5_High = 0x0120,
00019     HAL_WAKE_UP_IO_6_Low  = 0x0040,
00020     HAL_WAKE_UP_IO_6_High = 0x0140,
00021     HAL_WAKE_UP_IO_7_Low  = 0x0080,
00022     HAL_WAKE_UP_IO_7_High = 0x0180,
00023 } HAL_WAKE_UP_IO_x;
00024
00025 void HAL_WAKE_DeepSleepIoWake(HAL_WAKE_UP_IO_x);
00026 void HAL_WAKE_DeepSleepTimeWakeUp(uint32_t ms);
00027 void HAL_WAKE_DeepSleepIoTimeWakeUp(HAL_WAKE_UP_IO_x, uint32_t ms);
00028 #endif

```

4.63 readme.md 文件参考

4.64 source/hal_lks32mc09x_adc.c 文件参考

虚拟实现,用来在芯片之外约束函数运行效果

```

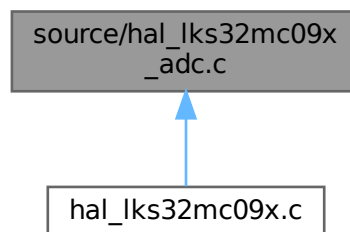
#include "../include/hal_lksmcu_adc.h"
#include "../hal_lks32mc09x.h"
#include "lks32mc09x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc09x_reg.h"

```

hal_lks32mc09x_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`adc.soft_reg`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_PORT_x](#) port_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_PORT_x](#) port_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
- 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 [adc_soft_reg](#) [2] = {0, 0}
ADC寄存器结构体句柄

4.64.1 详细描述

虚拟实现,用来在芯片之外约束函数运行效果

在文件 [hal_lks32mc09x_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

adc_x	要禁用的ADC模块编号

在文件 [hal_lks32mc09x_adc.c](#) 第 115 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

adc_x	要启用的ADC模块编号

在文件 [hal_lks32mc09x_adc.c](#) 第 87 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

adc_x	ADC模块编号

在文件 [hal_lks32mc09x_adc.c](#) 第 261 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

<i>adc</i> ↔ _x	ADC模块编号

返回

ADC量程

在文件 [hal_lks32mc09x_adc.c](#) 第 339 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

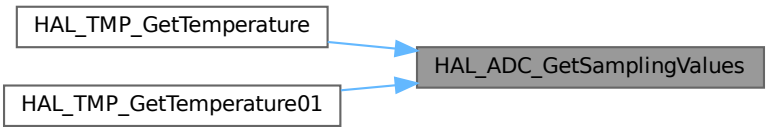
<i>adc_x</i>	ADC模块编号
<i>sequence</i> ↔ _x	采样序列号

返回

返回对应序列号的采样值

在文件 [hal_lks32mc09x_adc.c](#) 第 149 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_lks32mc09x_adc.c` 第 363 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_PORT_x port_x )
```

手动采样

参数

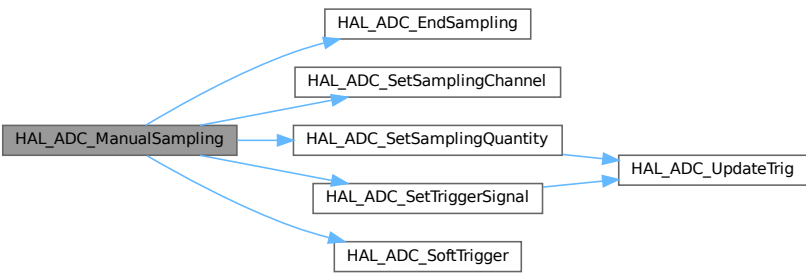
<i>n</i>	采样通道编号

返回

示例返回值,实际应返回采样值

在文件 [hal_lks32mc09x_adc.c](#) 第 279 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_lks32mc09x_adc.c](#) 第 62 行定义.

4.64.2.9 HAL_ADC.SetGpioIn()

```
void HAL_ADC_SetGpioIn (
    HAL_ADC_x adc_x,
    HAL_GPIO_Px pin_x )
```

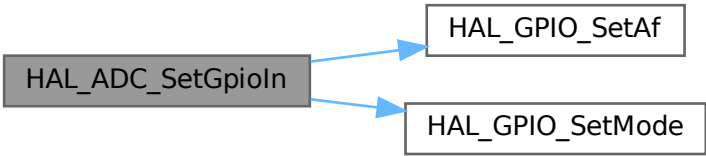
设置指定的GPIO为ADC输入

参数

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

在文件 [hal_lks32mc09x_adc.c](#) 第 380 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 )
```

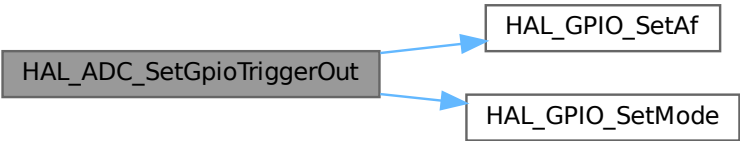
设置指定的GPIO为ADC触发信号输出

参数

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

在文件 [hal_lks32mc09x_adc.c](#) 第 391 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_x</i>	基准源

在文件 `hal_lks32mc09x_adc.c` 第 309 行定义.

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_PORT_x port_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>port_x</i>	采样通道端口

- < ADC第0次采样
- < ADC第1次采样

- < ADC第2次采样
- < ADC第3次采样
- < ADC第4次采样
- < ADC第5次采样
- < ADC第6次采样
- < ADC第7次采样
- < ADC第8次采样
- < ADC第9次采样
- < ADC第10次采样
- < ADC第11次采样

在文件 [hal_lks32mc09x_adc.c](#) 第 160 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_lks32mc09x_adc.c](#) 第 210 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_lks32mc09x_adc.c](#) 第 240 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

<i>adc_x</i>	ADC模块编号

在文件 [hal_lks32mc09x_adc.c](#) 第 270 行定义.

这是这个函数的调用关系图:



4.64.2.16 HAL_ADC_UpdateTrig()

```
static void HAL_ADC_UpdateTrig (
    HAL_ADC_x adc_x ) [static]
```

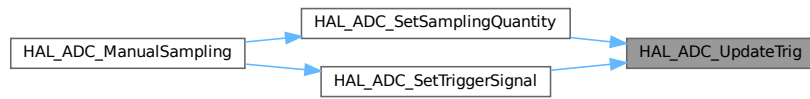
内部使用,更新adc.soft_reg描述的触发配置

参数

<i>adc_x</i>	使用的ADC模块编号

在文件 [hal_lks32mc09x_adc.c](#) 第 22 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 adc_soft_reg

```
volatile uint8_t adc_soft_reg[2] = {0, 0} [static]
```

ADC寄存器结构体句柄

在文件 [hal_lks32mc09x_adc.c](#) 第 16 行定义。

4.65 hal_lks32mc09x_adc.c

[浏览该文件的文档.](#)

```

00001
00005 #include "../include/hal_lksmcu_adc.h"
00006 #include "../hal_lks32mc09x.h"
00007 #include "lks32mc09x_user_manual.h"
00008 #include "../include/hal_lksmcu_reg.h"
00009 #include "../include/hal_lks32mc09x_reg.h"
00010
00015 // 虚拟寄存器
00016 static volatile uint8_t adc_soft_reg[2] = {0, 0}; // bit0 adc0使能 bit1 adc1使能 bit2 mcpwm触发 bit3
           timer触发 bit4 两段式触发使能
00017
00022 static void HAL_ADC_UpdateTrig(HAL_ADC_x adc_x)
00023 {
00024     // bit2 mcpwm触发 bit3 timer触发 bit4 两段式触发使能
00025     if (HAL_REG_READ_BIT(adc_soft_reg[adc_x], BIT4))
00026     {
00027         HAL_REG_SET(HAL_ADC[adc_x]->CFG, BIT12);
00028         if (HAL_REG_READ_BIT(adc_soft_reg[adc_x], BIT2)) // 两段 mcpwm触发
00029         {
00030             HAL_REG_WRITE_BIT(HAL_ADC[adc_x]->TRIG, BIT0 | BIT1 | BIT4 | BIT5, BIT0 | BIT1);
00031         }
00032         else if (HAL_REG_READ_BIT(adc_soft_reg[adc_x], BIT3)) // 两段 timer触发
00033         {
00034             HAL_REG_WRITE_BIT(HAL_ADC[adc_x]->TRIG, BIT0 | BIT1 | BIT4 | BIT5, BIT4 | BIT5);
00035         }
00036         else // 两段 软件触发
00037         {
00038             HAL_REG_WRITE_BIT(HAL_ADC[adc_x]->TRIG, BIT0 | BIT1 | BIT4 | BIT5, 0);
00039         }
00040     }
00041     else
00042     {
00043         HAL_REG_RESET(HAL_ADC[adc_x]->CFG, BIT12);
00044         if (HAL_REG_READ_BIT(adc_soft_reg[adc_x], BIT2)) // 一段 mcpwm触发
00045         {
00046             HAL_REG_WRITE_BIT(HAL_ADC[adc_x]->TRIG, BIT0 | BIT1 | BIT4 | BIT5, BIT0);
00047         }
00048         else if (HAL_REG_READ_BIT(adc_soft_reg[adc_x], BIT3)) // 一段 timer触发
00049         {
00050             HAL_REG_WRITE_BIT(HAL_ADC[adc_x]->TRIG, BIT0 | BIT1 | BIT4 | BIT5, BIT4);
00051         }
00052         else // 一段 软件触发
00053         {
00054             HAL_REG_WRITE_BIT(HAL_ADC[adc_x]->TRIG, BIT0 | BIT1 | BIT4 | BIT5, 0);
00055         }
00056     }
00057 }
  
```

```

00056     }
00057 }
00062 void HAL_ADC_Reset (HAL_ADC_x adc_x)
00063 {
00064     // 系统寄存器解锁
00065     HAL_REG_SYS_UNLOCK();
00066     switch (adc_x)
00067     {
00068         case HAL_ADC_0:
00069             HAL_REG_SET(SYS_SFT_RST, SYS_SFT_RST_ADC0_SFT_RST_BIT);
00070             HAL_REG_RESET(SYS_SFT_RST, SYS_SFT_RST_ADC0_SFT_RST_BIT);
00071             break;
00072         case HAL_ADC_1:
00073             HAL_REG_SET(SYS_SFT_RST, SYS_SFT_RST_ADC1_SFT_RST_BIT);
00074             HAL_REG_RESET(SYS_SFT_RST, SYS_SFT_RST_ADC1_SFT_RST_BIT);
00075             break;
00076         default:
00077             break;
00078     }
00079     // 系统寄存器上锁
00080     HAL_REG_SYS_LOCK();
00081 }
00082
00087 void HAL_ADC_Enable (HAL_ADC_x adc_x)
00088 {
00089     switch (adc_x)
00090     {
00091         case HAL_ADC_0:
00092             HAL_REG_SET(adc_soft_reg[adc_x], BIT0);
00093             HAL_REG_RESET(adc_soft_reg[adc_x], BIT0);
00094             break;
00095         case HAL_ADC_1:
00096             HAL_REG_SET(adc_soft_reg[adc_x], BIT1);
00097             HAL_REG_RESET(adc_soft_reg[adc_x], BIT1);
00098             break;
00099         default:
00100             break;
00101     }
00102     // 系统寄存器解锁
00103     HAL_REG_SYS_UNLOCK();
00104     // SYS_AFE_REG5[0] ADC 开启使能
00105     HAL_REG_SET(SYS_AFE_REG7, SYS_AFE_REG7_ADCPDN_BIT);
00106     HAL_REG_SET(SYS_AFE_REG7, SYS_AFE_REG7_SHAPDN_BIT);
00107     // 系统寄存器上锁
00108     HAL_REG_SYS_LOCK();
00109 }
00110
00115 void HAL_ADC_Disable (HAL_ADC_x adc_x)
00116 {
00117     switch (adc_x)
00118     {
00119         case HAL_ADC_0:
00120             HAL_REG_SET(adc_soft_reg[adc_x], BIT0);
00121             HAL_REG_RESET(adc_soft_reg[adc_x], BIT0);
00122             break;
00123         case HAL_ADC_1:
00124             HAL_REG_SET(adc_soft_reg[adc_x], BIT1);
00125             HAL_REG_RESET(adc_soft_reg[adc_x], BIT1);
00126             break;
00127         default:
00128             break;
00129     }
00130     // 因为adc只有一个使能位,所以
00131     // 只在两个adc同时关闭时才关闭adc
00132     if (HAL_REG_READ_BIT(adc_soft_reg[adc_x], BIT0 | BIT1) == 0)
00133     {
00134         // 系统寄存器解锁
00135         HAL_REG_SYS_UNLOCK();
00136         HAL_REG_RESET(SYS_AFE_REG7, SYS_AFE_REG7_ADCPDN_BIT);
00137         HAL_REG_RESET(SYS_AFE_REG7, SYS_AFE_REG7_SHAPDN_BIT);
00138         // 系统寄存器上锁
00139         HAL_REG_SYS_LOCK();
00140     }
00141 }
00142
00149 int16_t HAL_ADC_GetSamplingValues (HAL_ADC_x adc_x, HAL_ADC_SAMP_SEQUENCE_x sequence_x)
00150 {
00151     return HAL_ADC[adc_x]->DAT[sequence_x];
00152 }
00153
00160 void HAL_ADC_SetSamplingChannel (HAL_ADC_x adc_x, HAL_ADC_SAMP_SEQUENCE_x sequence_x, HAL_ADC_PORT_x port_x)
00161 {
00162     switch (sequence_x)
00163     {
00164         case HAL_ADC_SAMP_SEQUENCE_0:
00165             HAL_REG_WRITE_BIT(HAL_ADC[adc_x]->CHN0, 0x000f, port_x);
00166             break;

```

```

00167         case HAL_ADC_SAMP_SEQUENCE_1:
00168             HAL_REG_WRITE_BIT(HAL_ADC[adc_x]->CHN0, 0x00f0, port_x << 4);
00169             break;
00170         case HAL_ADC_SAMP_SEQUENCE_2:
00171             HAL_REG_WRITE_BIT(HAL_ADC[adc_x]->CHN0, 0x0f00, port_x << 8);
00172             break;
00173         case HAL_ADC_SAMP_SEQUENCE_3:
00174             HAL_REG_WRITE_BIT(HAL_ADC[adc_x]->CHN0, 0xf000, port_x << 12);
00175             break;
00176         case HAL_ADC_SAMP_SEQUENCE_4:
00177             HAL_REG_WRITE_BIT(HAL_ADC[adc_x]->CHN1, 0x000f, port_x);
00178             break;
00179         case HAL_ADC_SAMP_SEQUENCE_5:
00180             HAL_REG_WRITE_BIT(HAL_ADC[adc_x]->CHN1, 0x00f0, port_x << 4);
00181             break;
00182         case HAL_ADC_SAMP_SEQUENCE_6:
00183             HAL_REG_WRITE_BIT(HAL_ADC[adc_x]->CHN1, 0x0f00, port_x << 8);
00184             break;
00185         case HAL_ADC_SAMP_SEQUENCE_7:
00186             HAL_REG_WRITE_BIT(HAL_ADC[adc_x]->CHN1, 0xf000, port_x << 12);
00187             break;
00188         case HAL_ADC_SAMP_SEQUENCE_8:
00189             HAL_REG_WRITE_BIT(HAL_ADC[adc_x]->CHN2, 0x000f, port_x);
00190             break;
00191         case HAL_ADC_SAMP_SEQUENCE_9:
00192             HAL_REG_WRITE_BIT(HAL_ADC[adc_x]->CHN2, 0x00f0, port_x << 4);
00193             break;
00194         case HAL_ADC_SAMP_SEQUENCE_10:
00195             HAL_REG_WRITE_BIT(HAL_ADC[adc_x]->CHN2, 0x0f00, port_x << 8);
00196             break;
00197         case HAL_ADC_SAMP_SEQUENCE_11:
00198             HAL_REG_WRITE_BIT(HAL_ADC[adc_x]->CHN2, 0xf000, port_x << 12);
00199             break;
00200         default:
00201             break;
00202     }
00203 }
00204
00210 void HAL_ADC_SetSamplingQuantity(HAL_ADC_x adc_x, HAL_ADC_SAMP_NUMBER_x sample1_number,
    HAL_ADC_SAMP_NUMBER_x sample2_number)
00211 {
00212     // 第一段采样数量为0时, 芯片不进行采样
00213     if (sample1_number == HAL_ADC_SAMP_NUMBER_0)
00214     {
00215         sample2_number = HAL_ADC_SAMP_NUMBER_0;
00216     }
00217     // 两段采样数量之和不能大于10个
00218     if (sample1_number + sample2_number > 14)
00219     {
00220         sample2_number = (HAL_ADC_SAMP_NUMBER_x)(14 - sample1_number);
00221     }
00222     HAL_REG_WRITE_BIT(HAL_ADC[adc_x]->CHNT, 0x000f, sample1_number);
00223     HAL_REG_WRITE_BIT(HAL_ADC[adc_x]->CHNT, 0x00f0, sample2_number << 4);
00224     if (sample2_number)
00225     {
00226         HAL_REG_SET(adc_soft_reg[adc_x], BIT4);
00227     }
00228     else
00229     {
00230         HAL_REG_RESET(adc_soft_reg[adc_x], BIT4);
00231     }
00232     HAL_ADC_UpdateTrig(adc_x);
00233 }
00234
00240 void HAL_ADC_SetTriggerSignal(HAL_ADC_x adc_x, HAL_ADC_TRIGGER_x trigger)
00241 {
00242     switch (trigger)
00243     {
00244         case HAL_ADC_TRIGGER_MCPWM:
00245             HAL_REG_WRITE_BIT(adc_soft_reg[adc_x], BIT2 | BIT3, BIT2);
00246             break;
00247         case HAL_ADC_TRIGGER_TIMER:
00248             HAL_REG_WRITE_BIT(adc_soft_reg[adc_x], BIT2 | BIT3, BIT3);
00249             break;
00250         case HAL_ADC_TRIGGER_SOFT:
00251             HAL_REG_WRITE_BIT(adc_soft_reg[adc_x], BIT2 | BIT3, 0);
00252             break;
00253     }
00254     HAL_ADC_UpdateTrig(adc_x);
00255 }
00256
00261 void HAL_ADC_EndSampling(HAL_ADC_x adc_x)
00262 {
00263     HAL_REG_SET(HAL_ADC[adc_x]->CFG, BIT11);
00264 }
00265
00270 void HAL_ADC_SoftTrigger(HAL_ADC_x adc_x)

```

```

00271 {
00272     HAL_REG_WRITE(HAL_ADC[adc_x]->SWT, 0x5aa5);
00273 }
00279 int16_t HAL_ADC_ManualSampling(HAL_ADC_x adc_x, HAL_ADC_PORT_x port_x) // 手动采样 需要先设置采样通道为软件触发
00280 {
00281     int16_t data;
00282     HAL_ADC_SetTriggerSignal(adc_x, HAL_ADC_TRIGGER_SOFT);
00283     HAL_ADC_EndSampling(adc_x);
00284     HAL_ADC_SetSamplingQuantity(adc_x, HAL_ADC_SAMP_NUMBER_1, HAL_ADC_SAMP_NUMBER_0);
00285     HAL_ADC_SetSamplingChannel(adc_x, HAL_ADC_SAMP_SEQUENCE_0, port_x);
00286     HAL_REG_WRITE(HAL_ADC[adc_x]->IE, 0);
00287     HAL_REG_WRITE(HAL_ADC[adc_x]->IF, 0xffff);
00288     HAL_ADC_SoftTrigger(adc_x);
00289     uint32_t i = 3000;
00290     while (HAL_ADC[adc_x]->IF == 0)
00291     {
00292         i--;
00293         __NOP();
00294         if (i == 0)
00295         {
00296             break;
00297         }
00298     }
00299     HAL_REG_WRITE(HAL_ADC[adc_x]->IF, 0xffff);
00300     data = HAL_ADC[adc_x]->DAT[0];
00301     return data;
00302 }
00303
00309 void HAL_ADC_SetReference(HAL_ADC_x adc_x, HAL_ADC_REFERENCE_x reference_x)
00310 {
00311     (void)adc_x;
00312     switch (reference_x)
00313     {
00314         case HAL_ADC_REFERENCE_INTERNAL: // 使用内部基准源作为adc基准
00315             // 系统寄存器解锁
00316             HAL_REG_SYS_UNLOCK();
00317             // SYS_AFE_REG1[4] ADC 基准选择
00318             HAL_REG_WRITE_BIT(SYS_AFE_REG1, BIT4, 0);
00319             // 系统寄存器上锁
00320             HAL_REG_SYS_LOCK();
00321             break;
00322         case HAL_ADC_REFERENCE_EXTERNAL:
00323             // 系统寄存器解锁
00324             HAL_REG_SYS_UNLOCK();
00325             // SYS_AFE_REG1[4] ADC 基准选择
00326             HAL_REG_WRITE_BIT(SYS_AFE_REG1, BIT4, BIT4);
00327             // 系统寄存器上锁
00328             HAL_REG_SYS_LOCK();
00329             break;
00330     }
00331 }
00332
00339 float HAL_ADC_GetRange(HAL_ADC_x adc_x)
00340 {
00341     (void)adc_x;
00342     uint32_t val;
00343     float range = 5.0f;
00344     val = HAL_REG_READ_BIT(SYS_AFE_REG1, BIT4);
00345     val >= 4;
00346     switch (val)
00347     {
00348         case 0:
00349             range = 3.6f;
00350             break;
00351         case 1:
00352             range = 5.0f;
00353             break;
00354     }
00355     return range;
00356 }
00357
00363 float HAL_ADC_GetValuesUnit(HAL_ADC_x adc_x)
00364 {
00365     if (HAL_ADC[adc_x]->CFG & BIT10)
00366     {
00367         return HAL_ADC_GetRange(adc_x) / 0x07ff;
00368     }
00369     else
00370     {
00371         return HAL_ADC_GetRange(adc_x) / 0x7ff0;
00372     }
00373 }
00374
00380 void HAL_ADC_SetGpioIn(HAL_ADC_x adc_x, HAL_GPIO_Px pin_x)
00381 {
00382     (void)adc_x;

```

```

00383     HAL_GPIO_SetAf (pin_x, HAL_GPIO_AF_ADC);
00384     HAL_GPIO_SetMode (pin_x, HAL_GPIO_MODE_AFE);
00385 }
00391 void HAL_ADC_SetGpioTriggerOut (HAL_ADC_x adc_x, HAL_GPIO_Px pin_x)
00392 {
00393     (void)adc_x;
00394     HAL_GPIO_SetAf (pin_x, HAL_GPIO_AF_ADC_TRIG);
00395     HAL_GPIO_SetMode (pin_x, HAL_GPIO_MODE_OUT);
00396 }

```

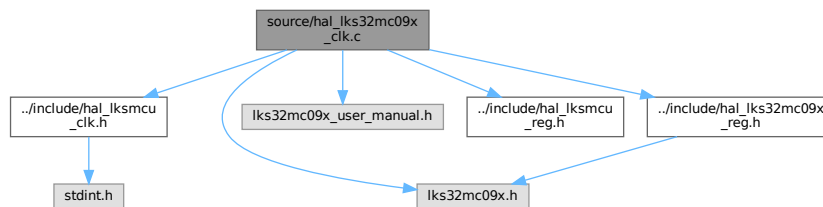
4.66 source/hal_lks32mc09x_clk.c 文件参考

```

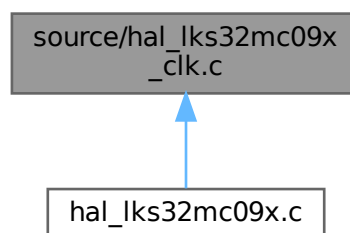
#include "../include/hal_lksmcu_clk.h"
#include "lks32mc09x.h"
#include "lks32mc09x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc09x_reg.h"

```

hal_lks32mc09x_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

4.66.1 函数说明

4.66.1.1 HAL_CLK_GetMcuFreq()

```
uint32_t HAL_CLK_GetMcuFreq (  
    void )
```

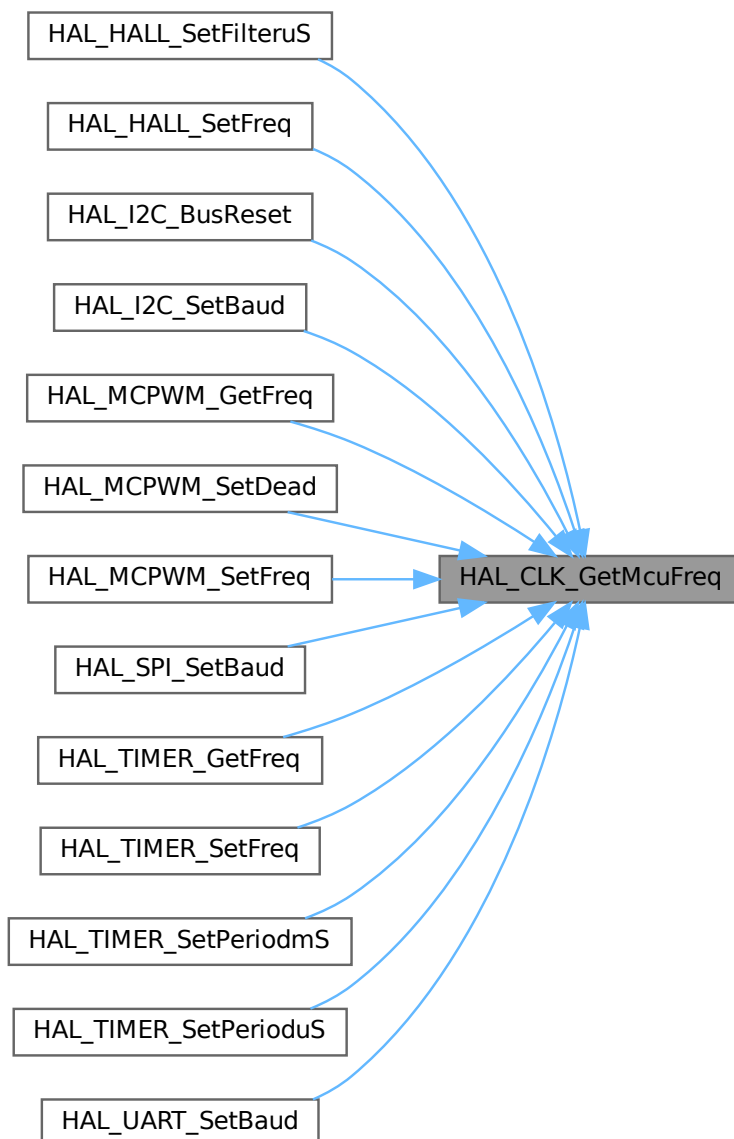
获取主频

返回

主频

在文件 [hal_lks32mc09x_clk.c](#) 第 90 行定义.

这是这个函数的调用关系图:



4.66.1.2 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

< ä¸'9Mhz

< ä¸'18Mhz

< ä¸'36Mhz

< ä¸'72Mhz

< ä¸'16Mhz

< ä¸'32Mhz

< ä¸'64Mhz

< 主频96Mhz

在文件 [hal_lks32mc09x.clk.c](#) 第 11 行定义.

4.66.2 变量说明

4.66.2.1 HAL_CLK_mclk

```
volatile uint32_t HAL_CLK_mclk = 96000000 [static]
```

在文件 [hal_lks32mc09x.clk.c](#) 第 6 行定义.

4.67 hal_lks32mc09x_clk.c

[浏览该文件的文档.](#)

```

00001 #include "../include/hal_lksmcu_clk.h"
00002 #include "lks32mc09x.h"
00003 #include "lks32mc09x_user_manual.h"
00004 #include "../include/hal_lksmcu_reg.h"
00005 #include "../include/hal_lks32mc09x_reg.h"
00006 static volatile uint32_t HAL_CLK_mclk = 96000000;
00011 void HAL_CLK_SetMcuFreq(HAL_CLK_FREQ_x FREQ_x)
00012 {
00013     // 000:96MHz即E32MHz
00014     // 010:72MHz即E24MHz
00015     // 101:64MHz即E32MHz
00016     HAL_REG_SYS_UNLOCK();
00017     switch (FREQ_x)
00018     {
00019         case HAL_CLK_FREQ_RC_8MHZ:
00020             HAL_CLK_mclk = 8000000;
00021             HAL_REG_WRITE(SYS_CLK_CFG, 0);
00022             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG7, SYS_AFE_REG7_PLL_FREQ, 0);
00023             break;
00024         case HAL_CLK_FREQ_RC_12MHZ:
00025             HAL_CLK_mclk = 12000000;
00026             HAL_REG_WRITE(SYS_CLK_CFG, 0x101);
00027             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG7, SYS_AFE_REG7_PLL_FREQ, 0);
00028             break;
00029         case HAL_CLK_FREQ_RC_24MHZ:
00030             HAL_CLK_mclk = 24000000;
00031             HAL_REG_WRITE(SYS_CLK_CFG, 0x111);
00032             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG7, SYS_AFE_REG7_PLL_FREQ, 0);
00033             break;
00034         case HAL_CLK_FREQ_RC_48MHZ:
00035             HAL_CLK_mclk = 48000000;
00036             HAL_REG_WRITE(SYS_CLK_CFG, 0x155);
00037             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG7, SYS_AFE_REG7_PLL_FREQ, 0);
00038             break;
00039
00040         case HAL_CLK_FREQ_RC_96MHZ:
00041             HAL_CLK_mclk = 96000000;
00042             HAL_REG_WRITE(SYS_CLK_CFG, 0x101);
00043             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG7, SYS_AFE_REG7_PLL_FREQ, 2);
00044             break;
00045         case HAL_CLK_FREQ_RC_18MHZ:
00046             HAL_CLK_mclk = 18000000;
00047             HAL_REG_WRITE(SYS_CLK_CFG, 0x111);
00048             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG7, SYS_AFE_REG7_PLL_FREQ, 2);
00049             break;
00050         case HAL_CLK_FREQ_RC_36MHZ:
00051             HAL_CLK_mclk = 36000000;
00052             HAL_REG_WRITE(SYS_CLK_CFG, 0x155);
00053             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG7, SYS_AFE_REG7_PLL_FREQ, 2);
00054             break;
00055         case HAL_CLK_FREQ_RC_72MHZ:
00056             HAL_CLK_mclk = 72000000;
00057             HAL_REG_WRITE(SYS_CLK_CFG, 0x1ff);
00058             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG7, SYS_AFE_REG7_PLL_FREQ, 2);
00059             break;
00060
00061         case HAL_CLK_FREQ_RC_16MHZ:
00062             HAL_CLK_mclk = 16000000;
00063             HAL_REG_WRITE(SYS_CLK_CFG, 0x111);
00064             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG7, SYS_AFE_REG7_PLL_FREQ, 5);
00065             break;
00066         case HAL_CLK_FREQ_RC_32MHZ:
00067             HAL_CLK_mclk = 32000000;
00068             HAL_REG_WRITE(SYS_CLK_CFG, 0x155);
00069             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG7, SYS_AFE_REG7_PLL_FREQ, 5);
00070             break;
00071         case HAL_CLK_FREQ_RC_64MHZ:
00072             HAL_CLK_mclk = 64000000;
00073             HAL_REG_WRITE(SYS_CLK_CFG, 0x1ff);
00074             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG7, SYS_AFE_REG7_PLL_FREQ, 5);
00075             break;
00076
00077         default:
00078             case HAL_CLK_FREQ_RC_96MHZ:
00079                 HAL_CLK_mclk = 96000000;
00080                 HAL_REG_WRITE(SYS_CLK_CFG, 0x1ff);
00081                 break;
00082     }
00083     HAL_REG_SYS_LOCK();
00084 }
00085
00090 uint32_t HAL_CLK_GetMcuFreq(void)

```

```

00091 {
00092     return 96000000;
00093 }

```

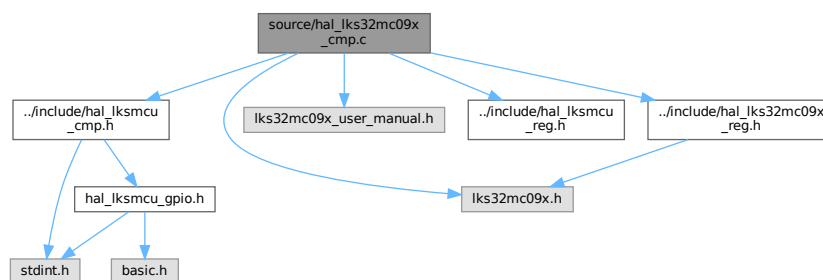
4.68 source/hal_lks32mc09x_cmp.c 文件参考

```

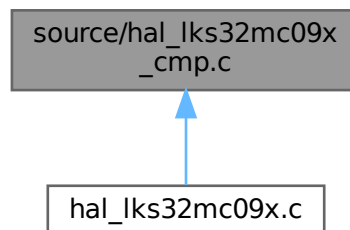
#include "../include/hal_lksmcu_cmp.h"
#include "lks32mc09x.h"
#include "lks32mc09x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc09x_reg.h"

```

hal_lks32mc09x_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` `cmpx`, uint32_t `filter`)
获取CMP模块的数字滤波宽度
- 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

4.68.1 函数说明

4.68.1.1 HAL_CMP_Disable()

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

模块关闭

参数

<code>cmp_x</code>	使用的CMP模块编号

在文件 `hal_lks32mc09x_cmp.c` 第 56 行定义.

这是这个函数的调用关系图:



4.68.1.2 HAL_CMP_Enable()

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

模块打开

参数

<i>cmp_x</i>	使用的CMP模块编号

在文件 [hal_lks32mc09x_cmp.c](#) 第 31 行定义.

4.68.1.3 HAL_CMP_GetData()

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

获取CMP比较结果

参数

<i>cmp_x</i>	CMP模块编号

返回

比较结果

在文件 [hal_lks32mc09x_cmp.c](#) 第 174 行定义.

4.68.1.4 HAL_CMP_Reset()

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

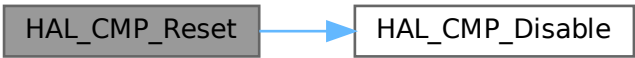
模块复位

参数

<i>cmp_x</i>	使用的CMP模块编号

在文件 [hal_lks32mc09x_cmp.c](#) 第 23 行定义.

函数调用图:



4.68.1.5 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_lks32mc09x.cmp.c](#) 第 83 行定义.

4.68.1.6 HAL_CMP_SetDigitalFilter()

```
uint32_t HAL_CMP_SetDigitalFilter (
    HAL_CMP_x cmpx,
    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_lks32mc09x.cmp.c](#) 第 114 行定义.

4.68.1.7 HAL_CMP_SetGpioIn()

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

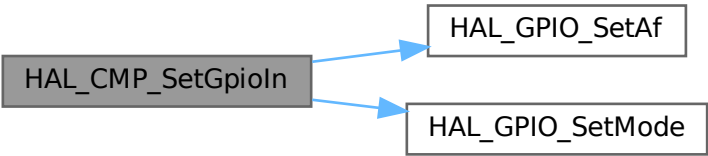
设置CMP模块的GPIO

参数

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

在文件 [hal_lks32mc09x_cmp.c](#) 第 162 行定义.

函数调用图:



4.68.1.8 HAL_CMP_SetGpioOut()

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

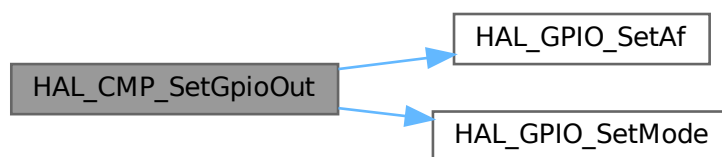
设置CMP模块的GPIO

参数

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

在文件 [hal_lks32mc09x_cmp.c](#) 第 150 行定义.

函数调用图:



4.69 hal_lks32mc09x_cmp.c

[浏览该文件的文档.](#)

```

00001 #include "../include/hal_lksmcu_cmp.h"
00002 #include "lks32mc09x.h"
00003 #include "lks32mc09x_user_manual.h"
00004 #include "../include/hal_lksmcu_reg.h"
00005 #include "../include/hal_lks32mc09x_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     volatile uint32_t DATA;
00018 } CMP_TypeDef;
00023 void HAL_CMP_Reset(HAL_CMP_x cmp_x)
00024 {
00025     HAL_CMP_Disable(cmp_x);
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     HAL_REG_SET(SYS_CLK_FEN, SYS_CLK_FEN_CMP_CLK_EN_BIT);
00049     // 系统寄存器上锁
00050     HAL_REG_SYS_LOCK();
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     HAL_REG_IF_WRITE(SYS_CLK_FEN, SYS_CLK_FEN_CMP_CLK_EN_BIT, !(SYS_AFE_REG5 & (SYS_AFE_REG5_CMP0PDN_BIT |
SYS_AFE_REG5_CMP1PDN_BIT)));
00074     // 系统寄存器上锁
00075     HAL_REG_SYS_LOCK();
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
00093             // 系统寄存器上锁
00094             HAL_REG_SYS_LOCK();
00095             break;
00096         case HAL_CMP_1:
00097             // 系统寄存器解锁
00098             HAL_REG_SYS_UNLOCK();
00099             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG3, SYS_AFE_REG3_CMP1_SELP, channel_p_x);
00100             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG3, SYS_AFE_REG3_CMP1_SELN, channel_n_x);
00101             // 系统寄存器上锁
00102             HAL_REG_SYS_LOCK();
00103             break;
00104         default:
00105             break;
00106     }
00107 }
00114 uint32_t HAL_CMP_SetDigitalFilter(HAL_CMP_x cmp_x, uint32_t filter)
00115 {
00116     (void)cmp_x;
00117     if (filter == 0)
00118     {
00119         HAL_REG_RESET(HAL_CMP[0]->TCLK, CMP_TCLK_COMM_DIV_MASK | CMP_TCLK_FIL1_CLK_DIV16_MASK |
CMP_TCLK_CLK1_EN_BIT | CMP_TCLK_FIL1_CLK_DIV2_MASK | CMP_TCLK_FIL0_CLK_DIV16_MASK | CMP_TCLK_CLK0_EN_BIT |
CMP_TCLK_FIL0_CLK_DIV2_MASK);
00120     }
00121     else
00122     {
00123         uint16_t div_2 = 0;
00124         for (; filter > 0xf; )
00125         {
00126             filter >>= 1;
00127             div_2++;
00128         }
00129         if (div_2 > CMP_TCLK_FIL0_CLK_DIV2_MAX)
00130         {
00131             div_2 = CMP_TCLK_FIL0_CLK_DIV2_MAX;
00132             filter = CMP_TCLK_FIL0_CLK_DIV16_MAX;
00133         }
00134         HAL_REG_RESET(HAL_CMP[0]->TCLK, CMP_TCLK_COMM_DIV_MASK);
00135         HAL_REG_SET(HAL_CMP[0]->TCLK, CMP_TCLK_CLK0_EN_BIT | CMP_TCLK_CLK1_EN_BIT);
00136         HAL_REG_WRITE_BIT_MASK_POS(HAL_CMP[0]->TCLK, CMP_TCLK_FIL0_CLK_DIV2, div_2);
00137         HAL_REG_WRITE_BIT_MASK_POS(HAL_CMP[0]->TCLK, CMP_TCLK_FIL0_CLK_DIV16, filter);
00138         HAL_REG_WRITE_BIT_MASK_POS(HAL_CMP[0]->TCLK, CMP_TCLK_FIL1_CLK_DIV2, div_2);
00139         HAL_REG_WRITE_BIT_MASK_POS(HAL_CMP[0]->TCLK, CMP_TCLK_FIL1_CLK_DIV16, filter);
00140
00141         filter = filter << div_2;
00142     }
00143     return filter;
00144 }
00150 void HAL_CMP_SetGpioOut(HAL_CMP_x cmp_x, HAL_GPIO_Px cmp_gpio)
00151 {
00152     (void)cmp_x;
00153     HAL_GPIO_SetAf(cmp_gpio, HAL_GPIO_AF_CMP_OUT);
00154     HAL_GPIO_SetMode(cmp_gpio, HAL_GPIO_MODE_OUT);
00155 }
00156
00162 void HAL_CMP_SetGpioIn(HAL_CMP_x cmp_x, HAL_GPIO_Px cmp_gpio)
00163 {
00164     (void)cmp_x;
00165     HAL_GPIO_SetAf(cmp_gpio, HAL_GPIO_AF_CMP);
00166     HAL_GPIO_SetMode(cmp_gpio, HAL_GPIO_MODE_AFE);
00167 }
00168
00174 uint8_t HAL_CMP_GetData(HAL_CMP_x cmp_x)
00175 {
00176     uint32_t flg = 0;
00177     switch (cmp_x)
00178     {
00179         case HAL_CMP_0:
00180             if (HAL_REG_READ_BIT(HAL_CMP[0]->TCLK, CMP_TCLK_CLK0_EN_BIT))
00181             {
00182                 // 打开滤波之后返回滤波后的结果

```

```

00183         flg = HAL_REG_READ_BIT(HAL_CMP[0]->DATA, CMP_DATA_CMP0_FLT_DATA_BIT);
00184     }
00185     else
00186     {
00187         // 如果没有配置滤波, 则返回滤波前的结果
00188         flg = HAL_REG_READ_BIT(HAL_CMP[0]->DATA, CMP_DATA_CMP0_RAW_DATA_BIT);
00189     }
00190     break;
00191 case HAL_CMP_1:
00192     if (HAL_REG_READ_BIT(HAL_CMP[0]->TCLK, CMP_TCLK_CLK1_EN_BIT))
00193     {
00194         // 打开滤波之后返回滤波后的结果
00195         flg = HAL_REG_READ_BIT(HAL_CMP[0]->DATA, CMP_DATA_CMP1_FLT_DATA_BIT);
00196     }
00197     else
00198     {
00199         // 如果没有配置滤波, 则返回滤波前的结果
00200         flg = HAL_REG_READ_BIT(HAL_CMP[0]->DATA, CMP_DATA_CMP1_RAW_DATA_BIT);
00201     }
00202     break;
00203 default:
00204     break;
00205 }
00206 if (flg == 0)
00207 {
00208     return 0;
00209 }
00210 else
00211 {
00212     return 1;
00213 }
00214 }

```

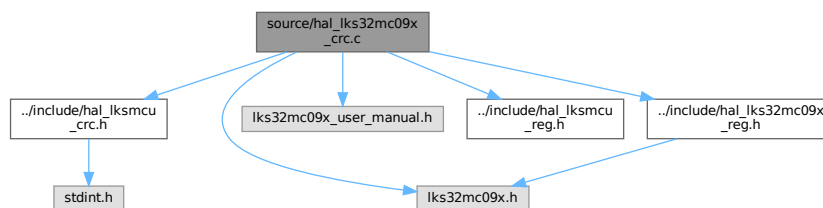
4.70 source/hal_lks32mc09x_crc.c 文件参考

```

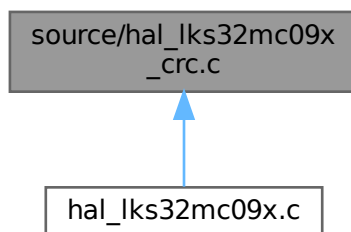
#include "../include/hal_lksmcu_crc.h"
#include "lks32mc09x.h"
#include "lks32mc09x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc09x_reg.h"

```

hal_lks32mc09x_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](#) []

4.70.1 枚举类型说明

4.70.1.1 CRC_Width

enum [CRC_Width](#)

枚举值

CRC_WIDTH_8	
CRC_WIDTH_16	
CRC_WIDTH_32	

在文件 [hal_lks32mc09x_crc.c](#) 第 8 行定义.

4.70.2 函数说明

4.70.2.1 HAL_CRC_Calc()

```
uint32_t HAL_CRC_Calc (
    HAL_CRC_TypeDef * CRCx,
    HAL_CRC_MODEL_TypeDef * model,
    uint8_t * data,
    uint32_t len )
```

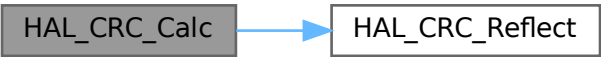
CRC计算

参数

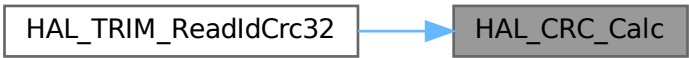
<i>CRCx</i>	CRC模块编号
<i>model</i>	CRC模型
<i>data</i>	数据地址
<i>len</i>	数据长度

在文件 [hal_lks32mc09x_crc.c](#) 第 96 行定义.

函数调用图:



这是这个函数的调用关系图:



4.70.2.2 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_lks32mc09x_crc.c](#) 第 82 行定义.

4.70.2.3 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_lks32mc09x_crc.c](#) 第 71 行定义.

4.70.2.4 HAL_CRC.Reflect()

```
static uint32_t HAL_CRC_Reflect (
    uint32_t data,
    uint8_t width ) [static]
```

在文件 [hal_lks32mc09x_crc.c](#) 第 43 行定义.

这是这个函数的调用关系图:



4.70.2.5 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_lks32mc09x_crc.c](#) 第 60 行定义.

4.70.3 变量说明

4.70.3.1 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_lks32mc09x_crc.c](#) 第 25 行定义.

4.71 hal_lks32mc09x_crc.c

[浏览该文件的文档。](#)

```

00001 #include "../include/hal_lksmcu_crc.h"
00002 #include "lks32mc09x.h"
00003 #include "lks32mc09x_user_manual.h"
00004 #include "../include/hal_lksmcu_reg.h"
00005 #include "../include/hal_lks32mc09x_reg.h"
00006
00007 // 仅内部使用
00008 typedef enum
00009 {
00010     CRC_WIDTH_8,
00011     CRC_WIDTH_16,
00012     CRC_WIDTH_32
00013 } CRC_Width;
00014 // 仅内部使用
00015 typedef struct
00016 {
00017     uint32_t polynomial; // 公式
00018     uint32_t initValue; // 初始值
00019     uint32_t xorOut; // 结果异或值
00020     uint8_t reflectIn; // 输入反转使能
00021     uint8_t reflectOut; // 输出反转使能
00022     CRC_Width width; // 数据宽度
00023 } CRC_Model_Params;
00024 // 仅内部使用
00025 static const CRC_Model_Params crcModels[] = {
00026     {0x07, 0x00, 0x00, 0, 0, CRC_WIDTH_8}, // CRC8
00027     {0x07, 0x00, 0x55, 0, 0, CRC_WIDTH_8}, // CRC8-ITU
00028     {0x07, 0xFF, 0x00, 1, 1, CRC_WIDTH_8}, // CRC8-ROHC
00029     {0x31, 0x00, 0x00, 1, 1, CRC_WIDTH_8}, // CRC8-MAXIMx
00030     {0x8005, 0x0000, 0x0000, 1, 1, CRC_WIDTH_16}, // CRC16-IBMx1
00031     {0x8005, 0x0000, 0xFFFF, 1, 1, CRC_WIDTH_16}, // CRC16-MAXIM
00032     {0x8005, 0xFFFF, 0xFFFF, 1, 1, CRC_WIDTH_16}, // CRC16-USBx1
00033     {0x8005, 0xFFFF, 0x0000, 1, 1, CRC_WIDTH_16}, // CRC16-MODBU
00034     {0x1021, 0x0000, 0x0000, 1, 1, CRC_WIDTH_16}, // CRC16-CCITT
00035     {0x1021, 0xFFFF, 0x0000, 0, 0, CRC_WIDTH_16}, // CRC16-CCITT-FALSE
00036     {0x1021, 0xFFFF, 0xFFFF, 1, 1, CRC_WIDTH_16}, // CRC16-X25
00037     {0x1021, 0x0000, 0x0000, 0, 0, CRC_WIDTH_16}, // CRC16-XMODEM
00038     {0x3D65, 0x0000, 0xFFFF, 1, 1, CRC_WIDTH_16}, // CRC16-DNP
00039     {0x04C11DB7, 0xFFFFFFFF, 0xFFFFFFFF, 1, 1, CRC_WIDTH_32}, // CRC32
00040     {0x04C11DB7, 0xFFFFFFFF, 0x00000000, 0, 0, CRC_WIDTH_32}, // CRC32-MPEG2
00041 };
00042 // 仅内部使用
00043 static uint32_t HAL_CRC_Reflect(uint32_t data, uint8_t width)
00044 {
00045     uint32_t result = 0;
00046     for (uint8_t 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     HAL_REG_SYS_UNLOCK();
00064     HAL_REG_SET(SYS_SFT_RST, SYS_SFT_RST_CRC0_SFT_RST_BIT);
00065     HAL_REG_SYS_LOCK();
00066 }
00071 void HAL_CRC_Enable(HAL_CRC_x CRC_x)
00072 {
00073     (void)CRC_x;
00074     HAL_REG_SYS_UNLOCK();
00075     HAL_REG_SET(SYS_CLK_FEN, SYS_CLK_FEN_CRC0_CLK_EN_BIT);
00076     HAL_REG_SYS_LOCK();
00077 }
00082 void HAL_CRC_Disable(HAL_CRC_x CRC_x)
00083 {
00084     (void)CRC_x;
00085     HAL_REG_SYS_UNLOCK();
00086     HAL_REG_RESET(SYS_CLK_FEN, SYS_CLK_FEN_CRC0_CLK_EN_BIT);
00087     HAL_REG_SYS_LOCK();
00088 }
00096 uint32_t HAL_CRC_Calc(HAL_CRC_x CRC_x, HAL_CRC_MODEL_x model, uint8_t *data, uint32_t len)
00097 {
00098     (void)CRC_x;
00099     if (model >= sizeof(crcModels) / sizeof(crcModels[0]))
00100     {
00101         return 0; // 无效模型
    
```

```

00102     }
00103
00104     const CRC_Model_Params *params = &crcModels[model];
00105     uint32_t crc = params->initValue;
00106     uint8_t width_bytes = (params->width == CRC_WIDTH_8) ? 1 : (params->width ==
CRC_WIDTH_16) ? 2
00107
: 4;
00108     uint8_t width_bits = width_bytes * 8;
00109     uint32_t top_bit = 1 << (width_bits - 1);
00110     uint32_t mask = (width_bits < 32) ? (uint32_t)((1 << width_bits) - 1) :
0xFFFFFFF;
00111
00112     for (uint32_t i = 0; i < len; i++)
00113     {
00114         uint8_t byte = data[i];
00115         if (params->reflectIn)
00116         {
00117             byte = HAL_CRC_Reflect(byte, 8);
00118         }
00119
00120         // 根据CRC宽度处理输入
00121         if (params->width == CRC_WIDTH_8)
00122         {
00123             crc ^= (uint32_t)byte;
00124         }
00125         else if (params->width == CRC_WIDTH_16)
00126         {
00127             crc ^= (uint32_t)byte << 8; // 假设数据是大端序输入
00128         }
00129         else
00130         {
00131             crc ^= (uint32_t)byte << 24;
00132         }
00133
00134         for (int j = 0; j < 8; j++)
00135         {
00136             if (crc & top_bit)
00137             {
00138                 crc = (crc << 1) ^ params->polynomial;
00139             }
00140             else
00141             {
00142                 crc <<= 1;
00143             }
00144             crc &= mask; // 确保只保留有效位
00145         }
00146     }
00147
00148     if (params->reflectOut)
00149     {
00150         crc = HAL_CRC_Reflect(crc, width_bits);
00151     }
00152
00153     crc ^= params->xorOut;
00154
00155     // 根据宽度返回适当大小的值
00156     if (params->width == CRC_WIDTH_8)
00157     {
00158         return crc & 0xFF;
00159     }
00160     else if (params->width == CRC_WIDTH_16)
00161     {
00162         return crc & 0xFFFF;
00163     }
00164     return crc;
00165 }

```

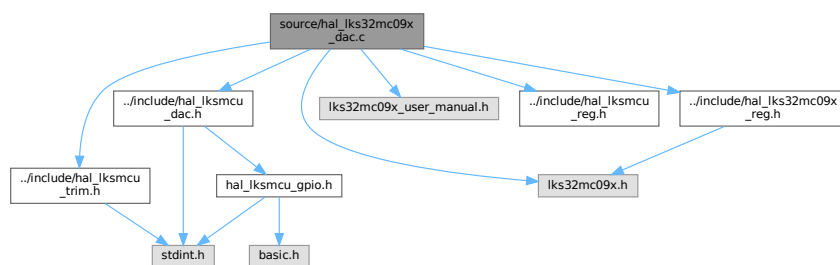
4.72 source/hal_lks32mc09x_dac.c 文件参考

```

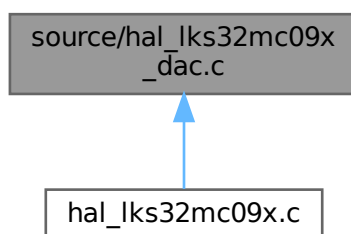
#include "../include/hal_lksmcu_dac.h"
#include "../include/hal_lksmcu_trim.h"
#include "lks32mc09x.h"
#include "lks32mc09x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc09x_reg.h"

```

hal_lks32mc09x_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)
- void [HAL_DAC_Enable](#) (HAL_DAC_x dac_x)
- void [HAL_DAC_Disable](#) (HAL_DAC_x dac_x)
- void [HAL_DAC_SetOutVal](#) (HAL_DAC_x dac_x, uint16_t val)
- void [HAL_DAC_SetOutVoltage](#) (HAL_DAC_x dac_x, uint16_t vol_mV)
- void [HAL_DAC_Out2GpioEnable](#) (HAL_DAC_x dac_x)
- void [HAL_DAC_Out2GpioDisable](#) (HAL_DAC_x dac_x)
- void [HAL_DAC_SetGpioOut](#) (HAL_DAC_x dac, HAL_GPIO_Px px)

设置DAC输出到GPIO

4.72.1 函数说明

4.72.1.1 HAL_DAC_Disable()

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

在文件 [hal_lks32mc09x_dac.c](#) 第 32 行定义.

这是这个函数的调用关系图:



4.72.1.2 HAL_DAC.Enable()

```
void HAL_DAC_Enable (
    HAL_DAC_TypeDef dac_x )
```

在文件 [hal_lks32mc09x_dac.c](#) 第 13 行定义.

4.72.1.3 HAL_DAC.Out2GpioDisable()

```
void HAL_DAC_Out2GpioDisable (
    HAL_DAC_TypeDef dac_x )
```

在文件 [hal_lks32mc09x_dac.c](#) 第 133 行定义.

4.72.1.4 HAL_DAC.Out2GpioEnable()

```
void HAL_DAC_Out2GpioEnable (
    HAL_DAC_TypeDef dac_x )
```

在文件 [hal_lks32mc09x_dac.c](#) 第 115 行定义.

4.72.1.5 HAL_DAC.Reset()

```
void HAL_DAC_Reset (
    HAL_DAC_TypeDef dac_x )
```

在文件 [hal_lks32mc09x_dac.c](#) 第 8 行定义.

函数调用图:



4.72.1.6 HAL_DAC_SetGpioOut()

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

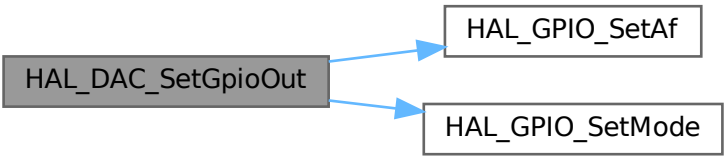
设置DAC输出到GPIO

参数

<i>dac</i>	DAC模块
<i>px</i>	GPIO

在文件 [hal_lks32mc09x.dac.c](#) 第 157 行定义.

函数调用图:



4.72.1.7 HAL_DAC_SetOutVal()

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

在文件 [hal_lks32mc09x.dac.c](#) 第 51 行定义.

这是这个函数的调用关系图:



4.72.1.8 HAL_DAC_SetOutVoltageV()

```
void HAL_DAC_SetOutVoltageV (
    HAL_DAC_x dac_x,
    uint16_t vol_mv )
```

在文件 `hal_lks32mc09x_dac.c` 第 65 行定义.

函数调用图:



4.73 hal_lks32mc09x_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 "lks32mc09x.h"
00004 #include "lks32mc09x_user_manual.h"
00005 #include "../include/hal_lksmcu_reg.h"
00006 #include "../include/hal_lks32mc09x_reg.h"
00007 // 模块复位
00008 void HAL_DAC_Reset (HAL_DAC_x dac_x)
00009 {
00010     HAL_DAC_Disable (dac_x);
00011 }
00012 // 模块打开
00013 void HAL_DAC_Enable (HAL_DAC_x dac_x)
00014 {
00015     // 系统寄存器解锁
00016     HAL_REG_SYS_UNLOCK();
00017     switch (dac_x)
00018     {
00019         case HAL_DAC_0:
00020             HAL_REG_SET (SYS_AFE_REG5, SYS_AFE_REG5_DAC0PDN_BIT);
00021             break;
00022         case HAL_DAC_1:
00023             HAL_REG_SET (SYS_AFE_REG5, SYS_AFE_REG5_DAC1PDN_BIT);
00024             break;
00025         default:
00026             break;
00027     }
00028     // 系统寄存器上锁
00029     HAL_REG_SYS_LOCK();
00030 }
00031 // 模块关闭
00032 void HAL_DAC_Disable (HAL_DAC_x dac_x)
00033 {
00034     // 系统寄存器解锁
00035     HAL_REG_SYS_UNLOCK();
00036     switch (dac_x)
00037     {
00038         case HAL_DAC_0:
00039             HAL_REG_RESET (SYS_AFE_REG5, SYS_AFE_REG5_DAC0PDN_BIT);
00040             break;
00041         case HAL_DAC_1:
00042             HAL_REG_RESET (SYS_AFE_REG5, SYS_AFE_REG5_DAC1PDN_BIT);
00043             break;
00044         default:
00045             break;
00046     }
00047     // 系统寄存器上锁
00048     HAL_REG_SYS_LOCK();
00049 }
```

```

00050
00051 void HAL_DAC_SetOutVal(HAL_DAC_x dac_x, uint16_t val)
00052 {
00053     switch (dac_x)
00054     {
00055         case HAL_DAC_0:
00056             HAL_REG_WRITE_BIT_MASK_POS(HAL_SYS[0]->AFE_DAC0, SYS_AFE_DAC0_DAC_IN, val);
00057             break;
00058         case HAL_DAC_1:
00059             HAL_REG_WRITE_BIT_MASK_POS(HAL_SYS[0]->AFE_DAC1, SYS_AFE_DAC1_DAC_IN, val >> 4); // DAC1是8位,
这里输入保持和dac0一致
00060             break;
00061         default:
00062             break;
00063     }
00064 }
00065 void HAL_DAC_SetOutVologemV(HAL_DAC_x dac_x, uint16_t vol_mV)
00066 {
00067     // 系统寄存器解锁
00068     int32_t val;
00069     HAL_REG_SYS_UNLOCK();
00070     switch (dac_x)
00071     {
00072         case HAL_DAC_0:
00073             if (vol_mV < 1000)
00074             {
00075                 val = vol_mV;
00076                 val = (val * 27853 + BIT12) >> 13;
00077                 if (val > 4095)
00078                 {
00079                     val = 4095;
00080                 }
00081                 if (val < 0)
00082                 {
00083                     val = 0;
00084                 }
00085                 // SYS_AFE_DAC0_DC = HAL_TRIM_Read(0x0000145C);
00086                 // SYS_AFE_DAC0_AMC = HAL_TRIM_Read(0x00001458);
00087                 HAL_REG_SET(SYS_AFE_REG1, SYS_AFE_REG1_DAC0_GAIN_BIT); // 1.2V量程
00088                 HAL_DAC_SetOutVal(dac_x, val);
00089             }
00090             else
00091             {
00092                 val = vol_mV;
00093                 val = val * 4095 / 4760;
00094                 if (val > 4095)
00095                 {
00096                     val = 4095;
00097                 }
00098                 if (val < 0)
00099                 {
00100                     val = 0;
00101                 }
00102                 // SYS_AFE_DAC0_DC = HAL_TRIM_Read(0x00001454);
00103                 // SYS_AFE_DAC0_AMC = HAL_TRIM_Read(0x00001450);
00104                 HAL_REG_RESET(SYS_AFE_REG1, SYS_AFE_REG1_DAC0_GAIN_BIT); // 4.76V量程
00105                 HAL_DAC_SetOutVal(dac_x, val);
00106             }
00107             break;
00108         default:
00109             break;
00110     }
00111     // 系统寄存器上锁
00112     HAL_REG_SYS_LOCK();
00113 }
00114
00115 void HAL_DAC_Out2GpioEnable(HAL_DAC_x dac_x)
00116 {
00117     // 系统寄存器解锁
00118     HAL_REG_SYS_UNLOCK();
00119     switch (dac_x)
00120     {
00121         case HAL_DAC_0:
00122             HAL_REG_SET(SYS_AFE_REG2, SYS_AFE_REG2_DAC0_OUT_EN_BIT);
00123             break;
00124         case HAL_DAC_1:
00125             HAL_REG_SET(SYS_AFE_REG2, SYS_AFE_REG2_DAC1_OUT_EN_BIT);
00126             break;
00127         default:
00128             break;
00129     }
00130     // 系统寄存器上锁
00131     HAL_REG_SYS_LOCK();
00132 }
00133 void HAL_DAC_Out2GpioDisable(HAL_DAC_x dac_x)
00134 {
00135     // 系统寄存器解锁

```

```

00136     HAL_REG_SYS_UNLOCK();
00137     switch (dac.x)
00138     {
00139         case HAL_DAC_0:
00140             HAL_REG_RESET(SYS_AFE_REG2, SYS_AFE_REG2_DAC0_OUT_EN_BIT);
00141             break;
00142         case HAL_DAC_1:
00143             HAL_REG_RESET(SYS_AFE_REG2, SYS_AFE_REG2_DAC1_OUT_EN_BIT);
00144             break;
00145         default:
00146             break;
00147     }
00148     // 系统寄存器上锁
00149     HAL_REG_SYS_LOCK();
00150 }
00151
00157 void HAL_DAC_SetGpioOut (HAL_DAC_x dac, HAL_GPIO_Px px)
00158 {
00159     (void)dac;
00160
00161     HAL_GPIO_SetAf (px, HAL_GPIO_AF_DAC);
00162     HAL_GPIO_SetMode (px, HAL_GPIO_MODE_AFE);
00163 }

```

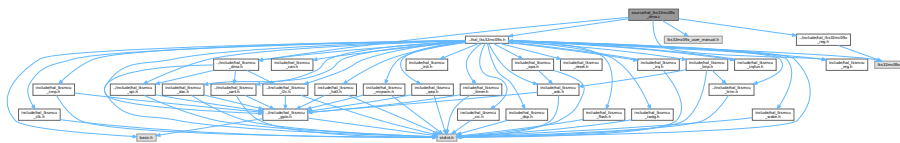
4.74 source/hal_lks32mc09x_dma.c 文件参考

```

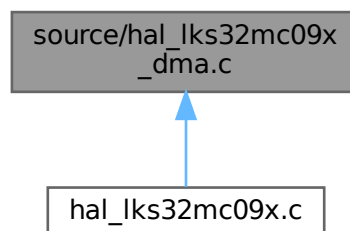
#include "../include/hal_lksmcu_dma.h"
#include "../hal_lks32mc09x.h"
#include "lks32mc09x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc09x_reg.h"

```

hal_lks32mc09x_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`)

变量

- static uint8_t `HAL_DMA_ena` = 0

4.74.1 函数说明

4.74.1.1 HAL_DMA_Disable()

```
void HAL_DMA_Disable (
    HAL_DMA_x DMA_x )
```

在文件 `hal_lks32mc09x.dma.c` 第 29 行定义.

4.74.1.2 HAL_DMA_Enable()

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

在文件 `hal_lks32mc09x.dma.c` 第 22 行定义.

4.74.1.3 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_lks32mc09x.dma.c` 第 104 行定义.

4.74.1.4 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_lks32mc09x.dma.c` 第 74 行定义.

4.74.1.5 HAL_DMA_MemCopy()

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

在文件 [hal_lks32mc09x_dma.c](#) 第 39 行定义.

4.74.1.6 HAL_DMA_Reset()

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

在文件 [hal_lks32mc09x_dma.c](#) 第 10 行定义.

4.74.1.7 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_lks32mc09x_dma.c](#) 第 94 行定义.

4.74.1.8 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_lks32mc09x_dma.c](#) 第 114 行定义.

4.74.1.9 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_lks32mc09x_dma.c](#) 第 64 行定义.

4.74.1.10 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_lks32mc09x_dma.c](#) 第 84 行定义.

4.74.1.11 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_lks32mc09x_dma.c](#) 第 54 行定义.

4.74.2 变量说明

4.74.2.1 HAL_DMA_ena

```
uint8_t HAL_DMA_ena = 0 [static]
```

在文件 [hal_lks32mc09x_dma.c](#) 第 9 行定义.

4.75 hal_lks32mc09x_dma.c

[浏览该文件的文档.](#)

```
00001 #include "../include/hal_lksmcu_dma.h"
00002 #include "../hal_lks32mc09x.h"
00003 #include "lks32mc09x_user_manual.h"
00004 #include "../include/hal_lksmcu_reg.h"
00005 #include "../include/hal_lks32mc09x_reg.h"
00006
00007 // 初始化
00008 // 模块复位
00009 static uint8_t HAL_DMA_ena = 0;
00010 void HAL_DMA_Reset (HAL_DMA_x DMA_x)
00011 {
00012     (void)DMA_x;
00013     // 系统寄存器解锁
00014     HAL_REG_SYS_UNLOCK();
00015
00016     HAL_REG_SET(SYS_SFT_RST, SYS_SFT_RST_DMA0_SFT_RST_BIT);
00017     HAL_REG_RESET(SYS_SFT_RST, SYS_SFT_RST_DMA0_SFT_RST_BIT);
00018     // 系统寄存器上锁
00019     HAL_REG_SYS_LOCK();
00020 }
00021 // 模块打开
00022 void HAL_DMA_Enable (HAL_DMA_x DMA_x)
00023 {
00024     (void)DMA_x;
00025     HAL_DMA_ena |= (1 << DMA_x);
00026     HAL_REG_SET(DMA0_CTRL, DMA_CTRL_EN_BIT);
00027 }
00028 // 模块关闭
00029 void HAL_DMA_Disable (HAL_DMA_x DMA_x)
00030 {
```

```

00031     (void)DMA_x;
00032     HAL_DMA_ena &= ~(1 << DMA_x);
00033     if (HAL_DMA_ena == 0)
00034     {
00035         HAL_REG_RESET(DMA0_CTRL, DMA_CTRL_EN_BIT);
00036     }
00037 }
00038 // 内存复制
00039 void HAL_DMA_MemCopy(HAL_DMA_x DMA_x, void *start, void *end, uint32_t len)
00040 {
00041     HAL_DMA [DMA_x]->CCR = 0;
00042     HAL_DMA [DMA_x]->SADR = (uint32_t)(uintptr_t)start;
00043     HAL_DMA [DMA_x]->DADR = (uint32_t)(uintptr_t)end;
00044     HAL_DMA [DMA_x]->CTMS = len;
00045     HAL_DMA [DMA_x]->CCR = DMA_CCRx_EN_BIT | DMA_CCRx_DINC_BIT | DMA_CCRx_SINC_BIT;
00046     HAL_REG_WRITE(DMA0_IF, DMA_IF_CH0_FIF_BIT << DMA_x);
00047     HAL_DMA [DMA_x]->REN = BIT31;
00048     while (HAL_REG_READ_BIT(DMA0_IF, DMA_IF_CH0_FIF_BIT << DMA_x) == 0)
00049     {
00050     }
00051     HAL_REG_WRITE(DMA0_IF, DMA_IF_CH0_FIF_BIT << DMA_x);
00052 }
00053 // dma使用uart发送数据,需要先初始化好对应的uart模块
00054 void HAL_DMA_UartTx(HAL_DMA_x DMA_x, HAL_UART_x UART_x, uint8_t *start, uint32_t len)
00055 {
00056     HAL_DMA [DMA_x]->CCR = 0;
00057     HAL_DMA [DMA_x]->SADR = (uint32_t)(uintptr_t)start;
00058     HAL_DMA [DMA_x]->DADR = (uint32_t)(uintptr_t)(&(HAL_UART [UART_x]->BUFF));
00059     HAL_DMA [DMA_x]->CTMS = len;
00060     HAL_DMA [DMA_x]->CCR = DMA_CCRx_EN_BIT | BIT1 | DMA_CCRx_SINC_BIT;
00061     HAL_DMA [DMA_x]->REN = BIT31 | BIT7;
00062 }
00063 // dma使用spi发送数据,需要先初始化好对应的spi模块
00064 void HAL_DMA_SpiTx(HAL_DMA_x DMA_x, HAL_SPI_x SPI_x, uint8_t *start, uint32_t len)
00065 {
00066     HAL_DMA [DMA_x]->CCR = 0;
00067     HAL_DMA [DMA_x]->SADR = (uint32_t)(uintptr_t)start;
00068     HAL_DMA [DMA_x]->DADR = (uint32_t)(uintptr_t)(&(HAL_SPI [SPI_x]->TX_DATA));
00069     HAL_DMA [DMA_x]->CTMS = len;
00070     HAL_DMA [DMA_x]->CCR = DMA_CCRx_EN_BIT | BIT1 | DMA_CCRx_SINC_BIT;
00071     HAL_DMA [DMA_x]->REN = BIT31 | BIT11;
00072 }
00073 // dma使用i2c发送数据,需要先初始化好对应的i2c模块
00074 void HAL_DMA_I2cTx(HAL_DMA_x DMA_x, HAL_I2C_x I2Cx, uint8_t *start, uint32_t len)
00075 {
00076     HAL_DMA [DMA_x]->CCR = 0;
00077     HAL_DMA [DMA_x]->SADR = (uint32_t)(uintptr_t)start;
00078     HAL_DMA [DMA_x]->DADR = (uint32_t)(uintptr_t)(&(HAL_I2C [I2Cx]->DATA));
00079     HAL_DMA [DMA_x]->CTMS = len;
00080     HAL_DMA [DMA_x]->CCR = DMA_CCRx_EN_BIT | BIT1 | DMA_CCRx_SINC_BIT;
00081     HAL_DMA [DMA_x]->REN = BIT31 | BIT11;
00082 }
00083 // dma使用uart发送数据,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     HAL_DMA [DMA_x]->CCR = 0;
00087     HAL_DMA [DMA_x]->SADR = (uint32_t)(uintptr_t)(&(HAL_UART [UART_x]->BUFF));
00088     HAL_DMA [DMA_x]->DADR = (uint32_t)(uintptr_t)start;
00089     HAL_DMA [DMA_x]->CTMS = len;
00090     HAL_DMA [DMA_x]->CCR = DMA_CCRx_EN_BIT | DMA_CCRx_RMODE_BIT | DMA_CCRx_DINC_BIT;
00091     HAL_DMA [DMA_x]->REN = BIT6;
00092 }
00093 // dma使用spi发送数据,需要先初始化好对应的spi模块
00094 void HAL_DMA_SpiRx(HAL_DMA_x DMA_x, HAL_SPI_x SPI_x, uint8_t *start, uint32_t len)
00095 {
00096     HAL_DMA [DMA_x]->CCR = 0;
00097     HAL_DMA [DMA_x]->SADR = (uint32_t)(uintptr_t)(&(HAL_SPI [SPI_x]->RX_DATA));
00098     HAL_DMA [DMA_x]->DADR = (uint32_t)(uintptr_t)start;
00099     HAL_DMA [DMA_x]->CTMS = len;
00100     HAL_DMA [DMA_x]->CCR = DMA_CCRx_EN_BIT | DMA_CCRx_RMODE_BIT | DMA_CCRx_DINC_BIT;
00101     HAL_DMA [DMA_x]->REN = BIT6;
00102 }
00103 // dma使用i2c发送数据,需要先初始化好对应的i2c模块
00104 void HAL_DMA_I2cRx(HAL_DMA_x DMA_x, HAL_I2C_x I2Cx, uint8_t *start, uint32_t len)
00105 {
00106     HAL_DMA [DMA_x]->CCR = 0;
00107     HAL_DMA [DMA_x]->SADR = (uint32_t)(uintptr_t)(&(HAL_I2C [I2Cx]->DATA));
00108     HAL_DMA [DMA_x]->DADR = (uint32_t)(uintptr_t)start;
00109     HAL_DMA [DMA_x]->CTMS = len;
00110     HAL_DMA [DMA_x]->CCR = DMA_CCRx_EN_BIT | DMA_CCRx_RMODE_BIT | DMA_CCRx_DINC_BIT;
00111     HAL_DMA [DMA_x]->REN = BIT6;
00112 }
00113 // 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
00114 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)
00115 {
00116     HAL_DMA [DMA1_x]->CCR = 0;

```

```

00117     HAL_DMA [DMA1_x]->SADR = (uint32_t) (uintptr_t) (&(HAL_SPI[SPI_x]->RX_DATA));
00118     HAL_DMA [DMA1_x]->DADR = (uint32_t) (uintptr_t) rxdata;
00119     HAL_DMA [DMA1_x]->CTMS = len;
00120     HAL_DMA [DMA1_x]->CCR = DMA_CCRx_EN_BIT | DMA_CCRx_RMODE_BIT | DMA_CCRx_DINC_BIT;
00121     HAL_DMA [DMA1_x]->REN = BIT6;
00122
00123     HAL_DMA [DMA2_x]->CCR = 0;
00124     HAL_DMA [DMA2_x]->SADR = (uint32_t) (uintptr_t) txdata;
00125     HAL_DMA [DMA2_x]->DADR = (uint32_t) (uintptr_t) (&(HAL_SPI[SPI_x]->TX_DATA));
00126     HAL_DMA [DMA2_x]->CTMS = len;
00127     HAL_DMA [DMA2_x]->CCR = DMA_CCRx_EN_BIT | DMA_CCRx_RMODE_BIT | DMA_CCRx_SINC_BIT;
00128     HAL_DMA [DMA2_x]->REN = BIT6;
00129 }

```

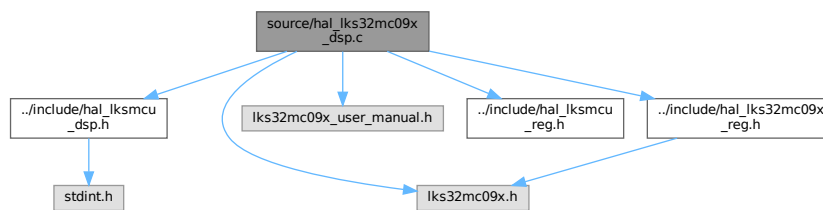
4.76 source/hal_lks32mc09x_dsp.c 文件参考

```

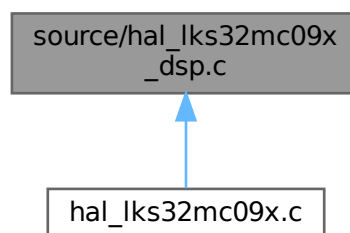
#include "../include/hal_lksmcu_dsp.h"
#include "lks32mc09x.h"
#include "lks32mc09x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc09x_reg.h"

```

hal_lks32mc09x_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)
- void HAL_DSP_Enable (HAL_DSP_x dsp_x)
- void HAL_DSP_Disable (HAL_DSP_x dsp_x)

- int16_t HAL_DSP_Sin (HAL_DSP_x dsp_x, int16_t angle)
- int16_t HAL_DSP_Cos (HAL_DSP_x dsp_x, int16_t angle)
- void HAL_DSP_SinCos (HAL_DSP_x dsp_x, int16_t angle, int16_t *sin, int16_t *cos)
- uint16_t HAL_DSP_Sqrt (HAL_DSP_x dsp_x, uint32_t x)
- int32_t HAL_DSP_Div (HAL_DSP_x dsp_x, int32_t a, int32_t b)

4.76.1 函数说明

4.76.1.1 HAL_DSP_Cos()

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

在文件 [hal_lks32mc09x.dsp.c](#) 第 42 行定义.

函数调用图:



4.76.1.2 HAL_DSP_Disable()

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

在文件 [hal_lks32mc09x.dsp.c](#) 第 25 行定义.

4.76.1.3 HAL_DSP_Div()

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

在文件 [hal_lks32mc09x.dsp.c](#) 第 126 行定义.

4.76.1.4 HAL_DSP_Enable()

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

在文件 [hal_lks32mc09x.dsp.c](#) 第 16 行定义.

4.76.1.5 HAL_DSP_Reset()

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

在文件 [hal_lks32mc09x.dsp.c](#) 第 6 行定义.

4.76.1.6 HAL_DSP_Sin()

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

在文件 [hal_lks32mc09x.dsp.c](#) 第 34 行定义.

函数调用图:

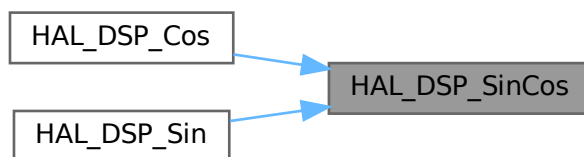


4.76.1.7 HAL_DSP_SinCos()

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

在文件 [hal_lks32mc09x.dsp.c](#) 第 50 行定义.

这是这个函数的调用关系图:



4.76.1.8 HAL_DSP_Sqrt()

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

在文件 [hal_lks32mc09x.dsp.c](#) 第 120 行定义。

4.77 hal_lks32mc09x.dsp.c

浏览该文件的文档。

```
00001 #include "../include/hal_lksmcu.dsp.h"
00002 #include "lks32mc09x.h"
00003 #include "lks32mc09x.user_manual.h"
00004 #include "../include/hal_lksmcu.reg.h"
00005 #include "../include/hal_lks32mc09x.reg.h"
00006 void HAL_DSP_Reset(HAL_DSP_x dsp_x)
00007 {
00008     (void)dsp_x;
00009     // 系统寄存器解锁
00010     HAL_REG_SYS_UNLOCK();
00011     HAL_REG_SET(SYS_SFT_RST, SYS_SFT_RST_DSP0_SFT_RST_BIT);
00012     HAL_REG_RESET(SYS_SFT_RST, SYS_SFT_RST_DSP0_SFT_RST_BIT);
00013     // 系统寄存器上锁
00014     HAL_REG_SYS_LOCK();
00015 }
00016 void HAL_DSP_Enable(HAL_DSP_x dsp_x)
00017 {
00018     (void)dsp_x;
00019     // 系统寄存器解锁
00020     HAL_REG_SYS_UNLOCK();
00021     HAL_REG_SET(SYS_CLK_FEN, SYS_CLK_FEN_DSP0_CLK_EN_BIT);
00022     // 系统寄存器上锁
00023     HAL_REG_SYS_LOCK();
00024 }
00025 void HAL_DSP_Disable(HAL_DSP_x dsp_x)
00026 {
00027     (void)dsp_x;
00028     // 系统寄存器解锁
00029     HAL_REG_SYS_UNLOCK();
00030     HAL_REG_RESET(SYS_CLK_FEN, SYS_CLK_FEN_DSP0_CLK_EN_BIT);
00031     // 系统寄存器上锁
00032     HAL_REG_SYS_LOCK();
00033 }
00034 int16_t HAL_DSP_Sin(HAL_DSP_x dsp_x, int16_t angle)
00035 {
00036     int16_t sin;
00037     int16_t cos;
00038
00039     HAL_DSP_SinCos(dsp_x, angle, &sin, &cos);
00040     return sin;
00041 }
00042 int16_t HAL_DSP_Cos(HAL_DSP_x dsp_x, int16_t angle)
00043 {
00044     int16_t sin;
00045     int16_t cos;
00046
00047     HAL_DSP_SinCos(dsp_x, angle, &sin, &cos);
00048     return cos;
00049 }
00050 void HAL_DSP_SinCos(HAL_DSP_x dsp_x, int16_t angle, int16_t *sin, int16_t *cos)
00051 {
00052     (void)dsp_x;
00053     static const s16 hSinCosTable[256] =
00054     {
00055         0x0000, 0x00C9, 0x0192, 0x025B, 0x0324, 0x03ED, 0x04B6, 0x057F,
00056         0x0648, 0x0711, 0x07D9, 0x08A2, 0x096B, 0x0A33, 0x0AFB, 0x0BC4,
00057         0x0C8C, 0x0D54, 0x0E1C, 0x0EE4, 0x0FAB, 0x1073, 0x113A, 0x1201,
00058         0x12C8, 0x138F, 0x1455, 0x151C, 0x15E2, 0x16A8, 0x176E, 0x1833,
00059         0x18F9, 0x19BE, 0x1A83, 0x1B47, 0x1C0C, 0x1CD0, 0x1D93, 0x1E57,
00060         0x1F1A, 0x1FDD, 0x209F, 0x2162, 0x2224, 0x22E5, 0x23A7, 0x2467,
00061         0x2528, 0x25E8, 0x26A8, 0x2768, 0x2827, 0x28E5, 0x29A4, 0x2A62,
00062         0x2B1F, 0x2BDC, 0x2C99, 0x2D55, 0x2E11, 0x2ECC, 0x2F87, 0x3042,
00063         0x30FC, 0x31B5, 0x326E, 0x3327, 0x33DF, 0x3497, 0x354E, 0x3604,
00064         0x36BA, 0x3770, 0x3825, 0x38D9, 0x398D, 0x3A40, 0x3AF3, 0x3BA5,
00065         0x3C57, 0x3D08, 0x3DB8, 0x3E68, 0x3F17, 0x3FC6, 0x4074, 0x4121,
00066         0x41CE, 0x427A, 0x4326, 0x43D1, 0x447B, 0x4524, 0x45CD, 0x4675,
```



```

00067      0x471D, 0x47C4, 0x486A, 0x490F, 0x49B4, 0x4A58, 0x4AFB, 0x4B9E,
00068      0x4C40, 0x4CE1, 0x4D81, 0x4E21, 0x4EC0, 0x4F5E, 0x4FFB, 0x5098,
00069      0x5134, 0x51CF, 0x5269, 0x5303, 0x539B, 0x5433, 0x54CA, 0x5560,
00070      0x55F6, 0x568A, 0x571E, 0x57B1, 0x5843, 0x58D4, 0x5964, 0x59F4,
00071      0x5A82, 0x5B10, 0x5B9D, 0x5C29, 0x5CB4, 0x5D3E, 0x5DC8, 0x5E50,
00072      0x5ED7, 0x5F5E, 0x5FE4, 0x6068, 0x60EC, 0x616F, 0x61F1, 0x6272,
00073      0x62F2, 0x6371, 0x63EF, 0x646C, 0x64E9, 0x6564, 0x65DE, 0x6657,
00074      0x66D0, 0x6747, 0x67BD, 0x6832, 0x68A7, 0x691A, 0x698C, 0x69FD,
00075      0x6A6E, 0x6ADD, 0x6B4B, 0x6BB8, 0x6C24, 0x6C8F, 0x6CF9, 0x6D62,
00076      0x6DCA, 0x6E31, 0x6E97, 0x6EFB, 0x6F5F, 0x6FC2, 0x7023, 0x7083,
00077      0x70E3, 0x7141, 0x719E, 0x71FA, 0x7255, 0x72AF, 0x7308, 0x735F,
00078      0x73B6, 0x740B, 0x7460, 0x74B3, 0x7505, 0x7556, 0x75A6, 0x75F4,
00079      0x7642, 0x768E, 0x76D9, 0x7723, 0x776C, 0x77B4, 0x77FB, 0x7840,
00080      0x7885, 0x78C8, 0x790A, 0x794A, 0x798A, 0x79C9, 0x7A06, 0x7A42,
00081      0x7AD, 0x7AB7, 0x7AEF, 0x7B27, 0x7B5D, 0x7B92, 0x7BC6, 0x7BF9,
00082      0x7C2A, 0x7C5A, 0x7C89, 0x7CB7, 0x7CE4, 0x7D0F, 0x7D3A, 0x7D63,
00083      0x7D8A, 0x7DB1, 0x7DD6, 0x7DFB, 0x7E1E, 0x7E3F, 0x7E60, 0x7E7F,
00084      0x7E9D, 0x7EBA, 0x7ED6, 0x7EF0, 0x7F0A, 0x7F22, 0x7F38, 0x7F4E,
00085      0x7F62, 0x7F75, 0x7F87, 0x7F98, 0x7FA7, 0x7FB5, 0x7FC2, 0x7FCE,
00086      0x7FD9, 0x7FE2, 0x7FEA, 0x7FF1, 0x7FF6, 0x7FFA, 0x7FFE, 0x7FFF};
00087
00088      u8 sector;
00089
00090      // 10 bit index computation //
00091      sector = ((u16)angle) >> 14;
00092      angle = angle >> 6;
00093
00094      switch (sector)
00095      {
00096          case 0: // U0_90:
00097              *sin = hSin.Cos.Table[(u8)(angle)];
00098              *cos = hSin.Cos.Table[(u8)(~angle)];
00099              break;
00100
00101          case 1: // U90_180:
00102              *sin = hSin.Cos.Table[(u8)(~angle)];
00103              *cos = -hSin.Cos.Table[(u8)(angle)];
00104              break;
00105
00106          case 2: // U180_270:
00107              *sin = -hSin.Cos.Table[(u8)(angle)];
00108              *cos = -hSin.Cos.Table[(u8)(~angle)];
00109              break;
00110
00111          case 3: // U270_360:
00112              *sin = -hSin.Cos.Table[(u8)(~angle)];
00113              *cos = hSin.Cos.Table[(u8)(angle)];
00114              break;
00115
00116          default:
00117              break;
00118      }
00119  }
00120  uint16_t HAL_DSP_Sqrt(HAL_DSP_x dsp_x, uint32_t x)
00121  {
00122      (void)dsp_x;
00123      DSP0_RAD = x;
00124      return DSP0_SQRT;
00125  }
00126  int32_t HAL_DSP_Div(HAL_DSP_x dsp_x, int32_t a, int32_t b)
00127  {
00128      (void)dsp_x;
00129      DSP0_DID = a;
00130      DSP0_DIS = b;
00131      return DSP0_QUO;
00132  }

```

4.78 source/hal_lks32mc09x_flash.c 文件参考

```

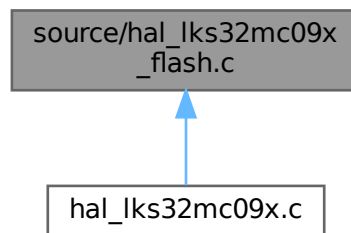
#include "../include/hal_lksmcu_flash.h"
#include "../hal_lks32mc09x.h"
#include "lks32mc09x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc09x_reg.h"

```

hal_lks32mc09x_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}

4.78.1 函数说明

4.78.1.1 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_lks32mc09x_flash.c](#) 第 30 行定义.

4.78.1.2 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_lks32mc09x_flash.c](#) 第 49 行定义.

4.78.1.3 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_lks32mc09x_flash.c](#) 第 21 行定义.

4.78.1.4 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_lks32mc09x_flash.c](#) 第 39 行定义.

4.78.1.5 HAL_FLASH_Erase()

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

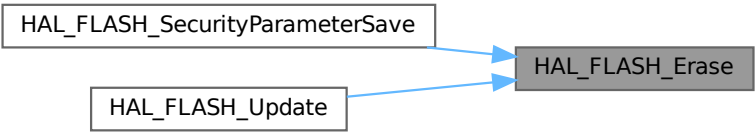
扇区擦除,len只能是扇区大小的整数倍,不够一个扇区的部分会被忽略

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_lks32mc09x_flash.c](#) 第 122 行定义.

这是这个函数的调用关系图:



4.78.1.6 HAL_FLASH_Program()

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

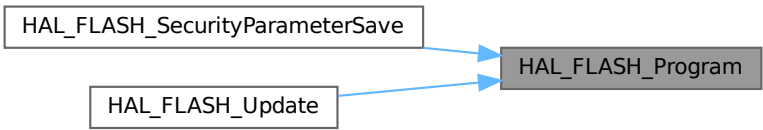
编程

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_lks32mc09x_flash.c](#) 第 150 行定义.

这是这个函数的调用关系图:



4.78.1.7 HAL_FLASH_Read()

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

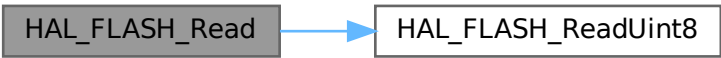
读FLASH取数据

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_lks32mc09x_flash.c](#) 第 73 行定义.

函数调用图:



这是这个函数的调用关系图:



4.78.1.8 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_lks32mc09x_flash.c](#) 第 60 行定义.

4.78.1.9 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_lks32mc09x.flash.c](#) 第 99 行定义.

4.78.1.10 HAL_FLASH_ReadUint32()

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

读FLASH取数据

参数

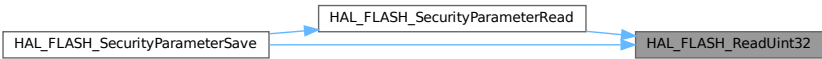
<i>flash</i>	模块类型
<i>flashaddr</i>	flash地址

返回

读取的数据

在文件 [hal_lks32mc09x.flash.c](#) 第 110 行定义.

这是这个函数的调用关系图:



4.78.1.11 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_lks32mc09x.flash.c](#) 第 88 行定义.

这是这个函数的调用关系图:



4.78.1.12 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_lks32mc09x.flash.c](#) 第 12 行定义.

4.78.1.13 HAL_FLASH.SecurityParameterCrc()

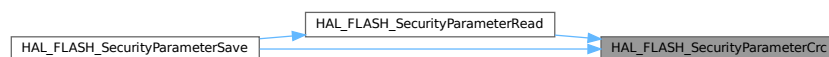
```
static uint32_t HAL_FLASH_SecurityParameterCrc (
    HAL_FLASH_X flash,
    uint32_t addr ) [static]
```

在文件 [hal_lks32mc09x.flash.c](#) 第 187 行定义.

函数调用图:



这是这个函数的调用关系图:



4.78.1.14 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

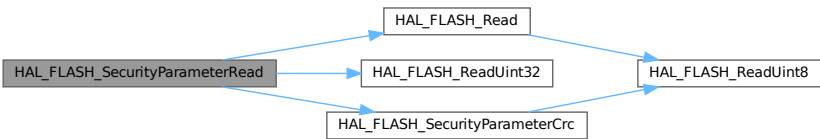
<i>flash</i>	模块类型
<i>data</i>	数据指针
<i>len</i>	数据长度

返回

使用的flash地址

在文件 [hal_lks32mc09x_flash.c](#) 第 313 行定义.

函数调用图:



这是这个函数的调用关系图:



4.78.1.15 HAL_FLASH_SecurityParameterSave()

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

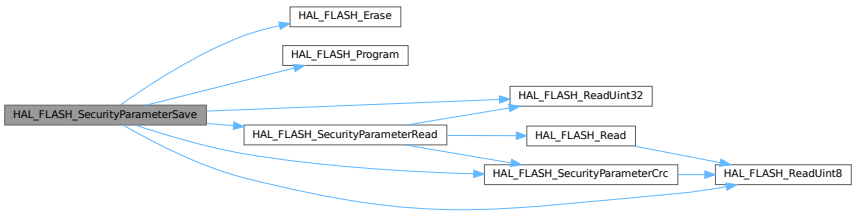
保存安全参数

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_lks32mc09x.flash.c` 第 222 行定义.

函数调用图:



4.78.1.16 HAL.FLASH.Update()

```
void HAL_FLASH.Update (
    HAL_FLASH_x flash,
    uint32_t flashaddr,
    uint8_t * data,
    uint32_t len )
```

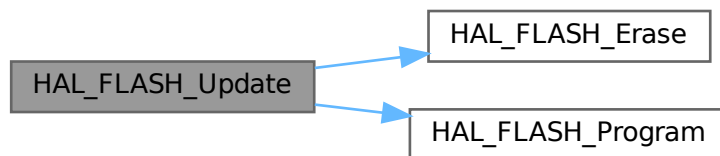
更新数据,等效先擦除再编程

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_lks32mc09x.flash.c` 第 180 行定义.

函数调用图:



4.78.2 变量说明

4.78.2.1 HAL_FLASH_data

```
volatile const uint8_t HAL_FLASH_data[3 * 512] = {0} [static]
```

在文件 [hal_lks32mc09x_flash.c](#) 第 7 行定义.

4.79 hal_lks32mc09x_flash.c

[浏览该文件的文档.](#)

```

00001 #include "../include/hal_lksmcu_flash.h"
00002 #include "../hal_lks32mc09x.h"
00003 #include "lks32mc09x.user_manual.h"
00004 #include "../include/hal_lksmcu_reg.h"
00005 #include "../include/hal_lks32mc09x_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, BIT19);
00043 }
00044
00049 void HAL_FLASH_DisableCache(HAL_FLASH_x flash)
00050 {
00051     (void)flash;
00052     HAL_REG_RESET(FLASH_CFG, BIT19);
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     (void)flash;
00076     uint32_t i;
00077     for (i = 0; i < len; i++)
00078     {
00079         ((uint8_t *)data)[i] = HAL_FLASH_ReadUint8(flash, flashaddr + i);
00080     }
00081 }
00082 uint8_t HAL_FLASH_ReadUint8(HAL_FLASH_x flash, uint32_t flashaddr)
00083 {
00084     (void)flash;
00085     return REG8(flashaddr);
00086 }
00087 uint16_t HAL_FLASH_ReadUint16(HAL_FLASH_x flash, uint32_t flashaddr)
00088 {
00089     (void)flash;
00090     return REG16(flashaddr);
00091 }
00092 uint32_t HAL_FLASH_ReadUint32(HAL_FLASH_x flash, uint32_t flashaddr)
00093 {
00094     (void)flash;
00095     return REG32(flashaddr);
00096 }
00097 void HAL_FLASH_Erase(HAL_FLASH_x flash, uint32_t flashaddr, uint32_t len)
00098 {
00099     (void)flash;
00100     uint32_t flg = __get_PRIMASK();
00101     __disable_irq();
00102     HAL_REG_SYS_UNLOCK();
00103     HAL_REG_WRITE(SYS_FLSE, 0xfca);
00104     HAL_REG_SET(FLASH_CFG, FLASH_CFG_ERS_EN_BIT);
00105     HAL_REG_RESET(FLASH_CFG, FLASH_CFG_ERS_TYPE_BIT);
00106     for (; len > 0; len -= 512, flashaddr += 512)
00107     {
00108         HAL_REG_WRITE(FLASH_ADDR, flashaddr);
00109         HAL_REG_WRITE(FLASH_ERASE, 0x7654dcba);
00110     }
00111     HAL_REG_WRITE(SYS_FLSE, 0);
00112     HAL_REG_RESET(FLASH_CFG, FLASH_CFG_ERS_EN_BIT);
00113     __set_PRIMASK(flq);
00114 }
00115 void HAL_FLASH_Program(HAL_FLASH_x flash, uint32_t flashaddr, uint8_t *data, uint32_t len)
00116 {
00117     (void)flash;
00118     uint8_t *p = data;
00119     uint32_t flg = __get_PRIMASK();
00120     __disable_irq();
00121     SYS_WR_PROTECT = 0x7a83;
00122     SYS_FLSP = 0xf35;
00123     FLASH_CFG |= BIT27 | BIT23;
00124     FLASH_ADDR = flashaddr;
00125     len &= ~3; // 低两位值忽略
00126     for (; len > 0; len -= 4)
00127     {
00128         FLASH_WDATA = p[0] | (p[1] << 8) | (p[2] << 16) | (p[3] << 24);
00129         p += 4;
00130     }
00131     SYS_FLSP = 0;
00132     FLASH_CFG &= ~(BIT27 | BIT23);
00133     SYS_WR_PROTECT = 0;
00134     __set_PRIMASK(flq);
00135 }
00136 void HAL_FLASH_Update(HAL_FLASH_x flash, uint32_t flashaddr, uint8_t *data, uint32_t len)
00137 {
00138     HAL_FLASH_Erase(flash, flashaddr, len);
00139     HAL_FLASH_Program(flash, flashaddr, data, len);
00140 }
00141 // 仅内部使用, 计算CRC32
00142 static uint32_t HAL_FLASH_SecurityParameterCrc(HAL_FLASH_x flash, uint32_t addr)
00143 {
00144     uint32_t crc = 0xffffffff; // CRC32初始值
00145     const uint32_t polynomial = 0xedb88320; // CRC32多项式
00146     for (uint32_t i = 0; i < 504; i++)
00147     {
00148         uint8_t byte = HAL_FLASH_ReadUint8(flash, addr + i);
00149         // 将字节与CRC的高8位异或
00150         crc ^= byte;
00151     }
00152 }

```

```

00199         // 处理每个bit
00200         for (uint32_t j = 0; j < 8; j++)
00201         {
00202             if (crc & 1)
00203             {
00204                 crc = (crc >> 1) ^ polynomial;
00205             }
00206             else
00207             {
00208                 crc >>= 1;
00209             }
00210         }
00211     }
00212
00213     return ~crc; // 取反得到最终结果
00214 }
00222 void HAL_FLASH_SecurityParameterSave(HAL_FLASH_x flash, void *defaultdata, void *data, uint32_t len)
00223 {
00224     uint32_t addr0 = ((uint32_t)(uintptr_t)HAL_FLASH_data + 512) & ~0x1ff;
00225     uint32_t addr1 = addr0 + 512;
00226     uint32_t flg = 0;
00227     uint32_t addr = HAL_FLASH_SecurityParameterRead(flash, 0, 0, 0);
00228     if (addr != 0xffffffff)
00229     {
00230         for (uint32_t i = 0; i < len; i++)
00231         {
00232             // 判断待写入的数据和flash保存的数据是否一致
00233             if ((uint8_t *)data[i] != HAL_FLASH_ReadUint8(flash, addr + i))
00234             {
00235                 flg = 1;
00236                 break;
00237             }
00238         }
00239     }
00240     else
00241     {
00242         for (uint32_t i = 0; i < len; i++)
00243         {
00244             // 判断待写入的数据和默认数据是否一致
00245             if ((uint8_t *)data[i] != ((uint8_t *)defaultdata)[i])
00246             {
00247                 flg = 1;
00248                 break;
00249             }
00250         }
00251     }
00252     if (flg != 0) // 需要保存数据
00253     {
00254         uint32_t cnt;
00255         uint32_t cnt1;
00256         uint32_t crc;
00257         if (addr == addr0)
00258         {
00259             cnt = HAL_FLASH_ReadUint32(flash, addr0 + 500);
00260             cnt1 = cnt + 1;
00261             if (cnt1 == 0) // 溢出后从1开始计数
00262             {
00263                 cnt1 = 1;
00264             }
00265             HAL_FLASH_Erase(flash, addr1, 512); // 擦除另一个扇区
00266             HAL_FLASH_Program(flash, addr1, data, len); // 写入用户数据
00267             HAL_FLASH_Program(flash, addr1 + 500, (void *)&cnt1, 4); // 写入计数
00268             crc = HAL_FLASH_SecurityParameterCrc(flash, addr1); // 计算CRC结果
00269             HAL_FLASH_Program(flash, addr1 + 504, (void *)&crc, 4); // 写入crc
00270
00271             if (cnt == 0xffffffff) // 溢出后清空另一个扇区 (确保HAL_FLASH_SecurityParameterRead判断正确)
00272             {
00273                 HAL_FLASH_Erase(flash, addr0, 512);
00274             }
00275         }
00276         else if (addr == addr1)
00277         {
00278             cnt = HAL_FLASH_ReadUint32(flash, addr0 + 500);
00279             cnt1 = cnt + 1;
00280             if (cnt1 == 0) // 溢出后从1开始计数
00281             {
00282                 cnt1 = 1;
00283             }
00284             HAL_FLASH_Erase(flash, addr0, 512);
00285             HAL_FLASH_Program(flash, addr0, data, len);
00286             HAL_FLASH_Program(flash, addr0 + 500, (void *)&cnt1, 4);
00287             crc = HAL_FLASH_SecurityParameterCrc(flash, addr0);
00288             HAL_FLASH_Program(flash, addr0 + 504, (void *)&crc, 4);
00289
00290             if (cnt == 0xffffffff) // 溢出后清空另一个扇区 (确保HAL_FLASH_SecurityParameterRead判断正确)
00291             {
00292                 HAL_FLASH_Erase(flash, addr0, 512);

```

```

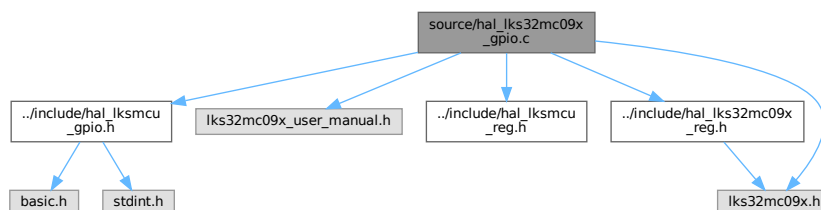
00293     }
00294     }
00295     else
00296     {
00297         cnt = 1; // 第一次写入时, 写入数据到扇区0, 并
// 从1开始计数
00298         HAL_FLASH_Erase(flash, addr1, 512); // 擦除另一个扇区
00299         HAL_FLASH_Program(flash, addr1, data, len); // 写入用户数据
00300         HAL_FLASH_Program(flash, addr1 + 500, (void *)&cnt, 4); // 写入计数
00301         crc = HAL_FLASH_SecurityParameterCrc(flash, addr1); // 计算CRC结果
00302         HAL_FLASH_Program(flash, addr1 + 504, (void *)&crc, 4); // 写入crc
00303     }
00304 }
00305 }
00313 uint32_t HAL_FLASH_SecurityParameterRead(HAL_FLASH_x flash, void *defaultdata, void *data, uint32_t len)
00314 {
00315     uint32_t addr0 = ((uint32_t)(uintptr_t)HAL_FLASH_data + 512) & ~0x1fff;
00316     uint32_t addr1 = addr0 + 512;
00317     uint32_t addr = 0;
00318     if (REG32(addr0 + 500) > REG32(addr1 + 500))
00319     {
00320         // 优先使用扇区0
00321         if (HAL_FLASH_SecurityParameterCrc(flash, addr0) == HAL_FLASH_ReadUint32(flash, addr0 + 504))
00322         { // 确保校验通过
00323             addr = addr0;
00324         }
00325         else if (HAL_FLASH_SecurityParameterCrc(flash, addr1) == HAL_FLASH_ReadUint32(flash, addr1 +
504))
00326         {
00327             addr = addr1;
00328         }
00329         else
00330         {
00331             // 两个扇区数据都丢失, 则返回默认参数
00332             // 运行时flash被误擦除
00333             // flash未初始化
00334             // 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)
00335             addr = 0xffffffff;
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 }
00359 if (len != 0)
00360 {
00361     if (addr != 0xffffffff)
00362     {
00363         HAL_FLASH_Read(flash, addr, data, len);
00364     }
00365     else
00366     {
00367         for (uint32_t i = 0; i < len; i++)
00368         {
00369             ((uint8_t *)data)[i] = ((uint8_t *)defaultdata)[i];
00370         }
00371     }
00372 }
00373 return addr;
00374 }

```

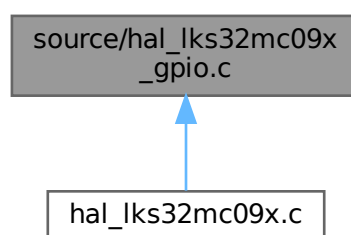
4.80 source/hal_lks32mc09x_gpio.c 文件参考

```
#include "../include/hal_lksmcu_gpio.h"
#include "lks32mc09x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc09x_reg.h"
#include "lks32mc09x.h"
```

hal_lks32mc09x_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_AFx](#) 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

4.80.1 函数说明

4.80.1.1 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_lks32mc09x_gpio.c](#) 第 35 行定义.

4.80.1.2 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_lks32mc09x_gpio.c](#) 第 23 行定义.

4.80.1.3 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_lks32mc09x_gpio.c](#) 第 177 行定义.

4.80.1.4 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_lks32mc09x_gpio.c](#) 第 155 行定义.

4.80.1.5 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_lks32mc09x_gpio.c](#) 第 143 行定义.

这是这个函数的调用关系图:



4.80.1.6 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_lks32mc09x_gpio.c](#) 第 9 行定义.

4.80.1.7 HAL_GPIO_RstReuseGpioDisable()

```
void HAL_GPIO_RstReuseGpioDisable (
    void )
```

将RST接口恢复为RST功能

在文件 [hal_lks32mc09x_gpio.c](#) 第 295 行定义.

4.80.1.8 HAL_GPIO_RstReuseGpioEnable()

```
void HAL_GPIO_RstReuseGpioEnable (
    void )
```

将RST接口配置为普通GPIO口

在文件 [hal_lks32mc09x_gpio.c](#) 第 283 行定义.

4.80.1.9 HAL_GPIO_SetAf()

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

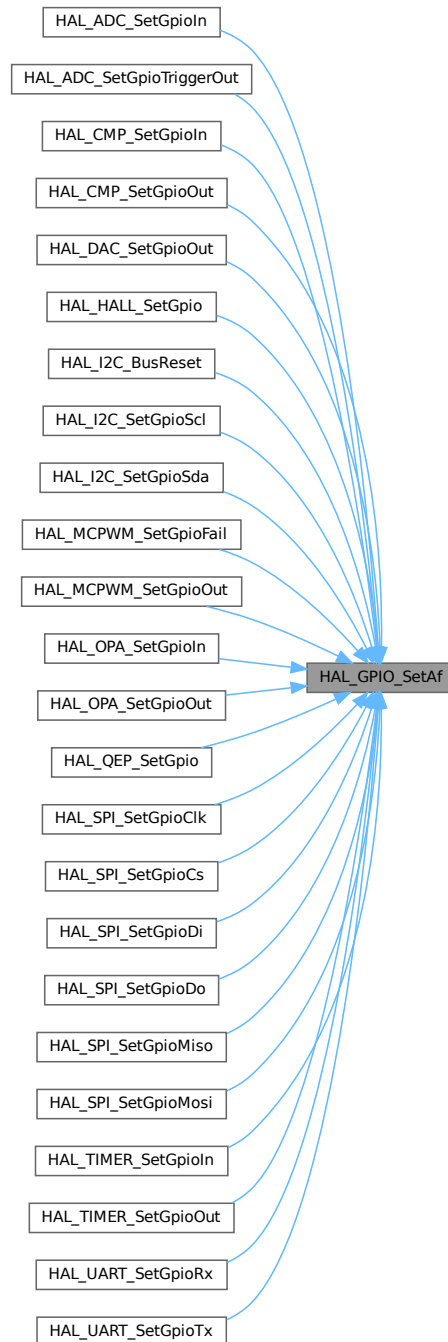
配置GPIO的特殊功能,如UART、I2C、SPI等

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_lks32mc09x_gpio.c](#) 第 199 行定义.

这是这个函数的调用关系图:



4.80.1.10 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_lks32mc09x_gpio.c](#) 第 108 行定义.

4.80.1.11 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_lks32mc09x_gpio.c](#) 第 119 行定义.

4.80.1.12 HAL_GPIO_SetMode()

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

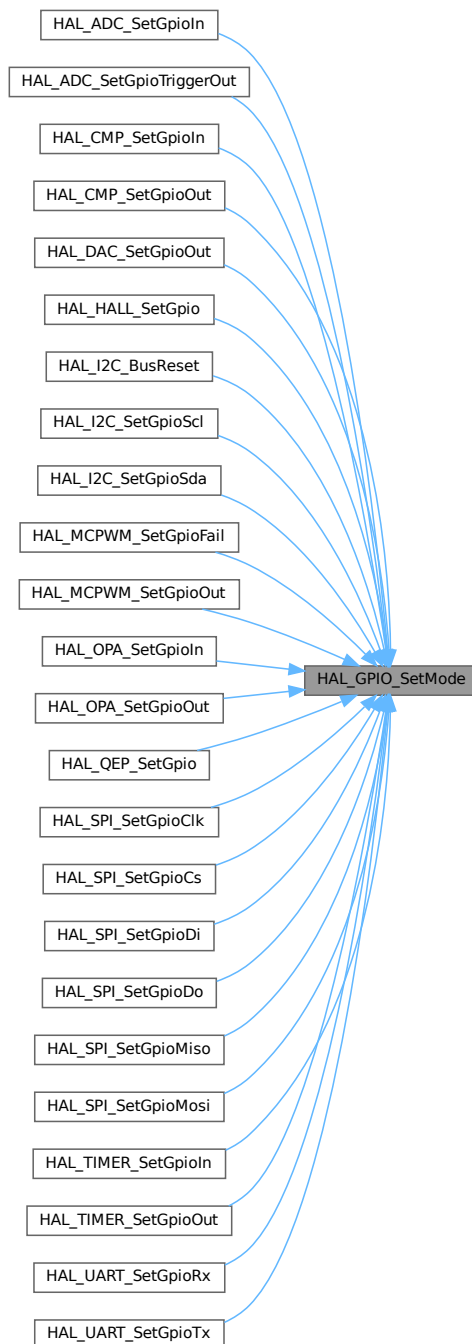
设置指定GPIO的工作模式

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_lks32mc09x_gpio.c](#) 第 49 行定义.

这是这个函数的调用关系图:



4.80.1.13 HAL_GPIO_SwdReuseGpioDisable()

```
void HAL_GPIO_SwdReuseGpioDisable (
    void )
```

将SWD接口恢复为SWD功能

在文件 [hal_lks32mc09x_gpio.c](#) 第 271 行定义.

4.80.1.14 HAL_GPIO_SwdReuseGpioEnable()

```
void HAL_GPIO_SwdReuseGpioEnable (
    void )
```

将SWD接口配置为普通GPIO口

在文件 [hal_lks32mc09x_gpio.c](#) 第 259 行定义。

4.80.1.15 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_lks32mc09x_gpio.c](#) 第 131 行定义。

4.81 hal_lks32mc09x_gpio.c

[浏览该文件的文档。](#)

```
00001 #include "../include/hal_lksmcu_gpio.h"
00002 #include "lks32mc09x_user_manual.h"
00003 #include "../include/hal_lksmcu_reg.h"
00004 #include "../include/hal_lks32mc09x_reg.h"
00005 #include "lks32mc09x.h"
00009 void HAL_GPIO_Reset()
00010 {
00011     // 系统寄存器解锁
00012     HAL_REG_SYS_UNLOCK();
00013     HAL_REG_SET(SYS_SFT_RST, SYS_SFT_RST_GPIO_SFT_RST_BIT);
00014     HAL_REG_RESET(SYS_SFT_RST, SYS_SFT_RST_GPIO_SFT_RST_BIT);
00015     HAL_REG_RESET(SYS_IO_CFG, BIT6);
00016     HAL_REG_RESET(SYS_IO_CFG, BIT5);
00017     // 系统寄存器上锁
00018     HAL_REG_SYS_LOCK();
00019 }
00023 void HAL_GPIO_Enable()
00024 {
00025     // 系统寄存器解锁
00026     HAL_REG_SYS_UNLOCK();
00027     // [11] GPIO模块时钟控制信号,默认关闭GPIO模块时钟 1:使能GPIO模块时钟 0:关闭GPIO模块时钟
00028     HAL_REG_SET(SYS_CLK_FEN, SYS_CLK_FEN_GPIO_CLK_EN_BIT);
00029     // 系统寄存器上锁
00030     HAL_REG_SYS_LOCK();
00031 }
00035 void HAL_GPIO_Disable()
00036 {
00037     // 系统寄存器解锁
00038     HAL_REG_SYS_UNLOCK();
00039     // GPIO.CLK_EN[11] GPIO 模块时钟控制信号,默认关闭 GPIO 模块时钟
00040     HAL_REG_RESET(SYS_CLK_FEN, SYS_CLK_FEN_GPIO_CLK_EN_BIT);
00041     // 系统寄存器上锁
00042     HAL_REG_SYS_LOCK();
00043 }
00049 void HAL_GPIO_SetMode(HAL_GPIO_Px px, HAL_GPIO_MODE_x mode)
00050 {
```

```

00051     uint8_t p      = (px >> 8) & 0xff;
00052     uint8_t x      = (px >> 0) & 0xff;
00053     uint16_t bitx = 1 << x;
00054     // 默认电平设置
00055     if (mode & BIT4)
00056     {
00057         HAL_REG_WRITE(HAL_GPIO[p]->BRR, bitx);
00058     }
00059     else if (mode & BIT5)
00060     {
00061         HAL_REG_WRITE(HAL_GPIO[p]->BSRR, bitx);
00062     }
00063     else
00064     {
00065     }
00066     // 开漏设置
00067     if (mode & BIT3)
00068     {
00069         HAL_REG_SET(HAL_GPIO[p]->PDE, bitx);
00070     }
00071     else
00072     {
00073         HAL_REG_RESET(HAL_GPIO[p]->PDE, bitx);
00074     }
00075     // 上拉设置
00076     if (mode & BIT2)
00077     {
00078         HAL_REG_SET(HAL_GPIO[p]->PUE, bitx);
00079     }
00080     else
00081     {
00082         HAL_REG_RESET(HAL_GPIO[p]->PUE, bitx);
00083     }
00084     // 输出设置
00085     if (mode & BIT1)
00086     {
00087         HAL_REG_SET(HAL_GPIO[p]->POE, bitx);
00088     }
00089     else
00090     {
00091         HAL_REG_RESET(HAL_GPIO[p]->POE, bitx);
00092     }
00093     // 输入使能
00094     if (mode & BIT0)
00095     {
00096         HAL_REG_SET(HAL_GPIO[p]->PIE, bitx);
00097     }
00098     else
00099     {
00100         HAL_REG_RESET(HAL_GPIO[p]->PIE, bitx);
00101     }
00102 }
00108 void HAL_GPIO_SetHigh(HAL_GPIO_Px px)
00109 {
00110     uint8_t p      = (px >> 8) & 0xff;
00111     uint8_t x      = (px >> 0) & 0xff;
00112     uint32_t bitx = 1 << x;
00113     HAL_REG_WRITE(HAL_GPIO[p]->BSRR, bitx);
00114 }
00119 void HAL_GPIO_SetLow(HAL_GPIO_Px px)
00120 {
00121     uint8_t p      = (px >> 8) & 0xff;
00122     uint8_t x      = (px >> 0) & 0xff;
00123     uint32_t bitx = 1 << x;
00124     HAL_REG_WRITE(HAL_GPIO[p]->BRR, bitx);
00125 }
00131 void HAL_GPIO_Write(HAL_GPIO_Px px, uint8_t value)
00132 {
00133     uint8_t p      = (px >> 8) & 0xff;
00134     uint8_t x      = (px >> 0) & 0xff;
00135     uint32_t bitx = 1 << x;
00136     uint32_t valx = value << x;
00137     HAL_REG_WRITE_BIT(HAL_GPIO[p]->PDO, bitx, valx);
00138 }
00143 void HAL_GPIO_Invert(HAL_GPIO_Px px)
00144 {
00145     uint8_t p      = (px >> 8) & 0xff;
00146     uint8_t x      = (px >> 0) & 0xff;
00147     uint32_t bitx = 1 << x;
00148     HAL_REG_INVERT(HAL_GPIO[p]->PDO, bitx);
00149 }
00155 uint8_t HAL_GPIO_GetOutData(HAL_GPIO_Px px)
00156 {
00157     uint8_t p      = (px >> 8) & 0xff;
00158     uint8_t x      = (px >> 0) & 0xff;
00159     uint32_t bitx = 1 << x;
00160     uint32_t valx = HAL_REG_READ_BIT(HAL_GPIO[p]->PDO, bitx);

```

```

00161     uint8_t result;
00162     if (valx)
00163     {
00164         result = 1;
00165     }
00166     else
00167     {
00168         result = 0;
00169     }
00170     return result;
00171 }
00177 uint8_t HAL_GPIO.GetInData(HAL_GPIO_Px px)
00178 {
00179     uint8_t p = (px >> 8) & 0xff;
00180     uint8_t x = (px >> 0) & 0xff;
00181     uint32_t bitx = 1 << x;
00182     uint32_t valx = HAL_REG_READ_BIT(HAL_GPIO[p]->PDI, bitx);
00183     uint8_t result;
00184     if (valx)
00185     {
00186         result = 1;
00187     }
00188     else
00189     {
00190         result = 0;
00191     }
00192     return result;
00193 }
00199 void HAL_GPIO.SetAf(HAL_GPIO_Px px, HAL_GPIO_Af af)
00200 {
00201     uint8_t p = (px >> 8) & 0xff;
00202     uint8_t x = (px >> 0) & 0xff;
00203     af &= 0xf; // 限制af的取值范围为0-15
00204     switch (x)
00205     {
00206     case 0:
00207         HAL_REG_WRITE_BIT(HAL_GPIO[p]->F3210, 0x000f, af);
00208         break;
00209     case 1:
00210         HAL_REG_WRITE_BIT(HAL_GPIO[p]->F3210, 0x00f0, af << 4);
00211         break;
00212     case 2:
00213         HAL_REG_WRITE_BIT(HAL_GPIO[p]->F3210, 0x0f00, af << 8);
00214         break;
00215     case 3:
00216         HAL_REG_WRITE_BIT(HAL_GPIO[p]->F3210, 0xf000, af << 12);
00217         break;
00218     case 4:
00219         HAL_REG_WRITE_BIT(HAL_GPIO[p]->F7654, 0x000f, af);
00220         break;
00221     case 5:
00222         HAL_REG_WRITE_BIT(HAL_GPIO[p]->F7654, 0x00f0, af << 4);
00223         break;
00224     case 6:
00225         HAL_REG_WRITE_BIT(HAL_GPIO[p]->F7654, 0x0f00, af << 8);
00226         break;
00227     case 7:
00228         HAL_REG_WRITE_BIT(HAL_GPIO[p]->F7654, 0xf000, af << 12);
00229         break;
00230     case 8:
00231         HAL_REG_WRITE_BIT(HAL_GPIO[p]->FBA98, 0x000f, af);
00232         break;
00233     case 9:
00234         HAL_REG_WRITE_BIT(HAL_GPIO[p]->FBA98, 0x00f0, af << 4);
00235         break;
00236     case 10:
00237         HAL_REG_WRITE_BIT(HAL_GPIO[p]->FBA98, 0x0f00, af << 8);
00238         break;
00239     case 11:
00240         HAL_REG_WRITE_BIT(HAL_GPIO[p]->FBA98, 0xf000, af << 12);
00241         break;
00242     case 12:
00243         HAL_REG_WRITE_BIT(HAL_GPIO[p]->FFEDC, 0x000f, af);
00244         break;
00245     case 13:
00246         HAL_REG_WRITE_BIT(HAL_GPIO[p]->FFEDC, 0x00f0, af << 4);
00247         break;
00248     case 14:
00249         HAL_REG_WRITE_BIT(HAL_GPIO[p]->FFEDC, 0x0f00, af << 8);
00250         break;
00251     case 15:
00252         HAL_REG_WRITE_BIT(HAL_GPIO[p]->FFEDC, 0xf000, af << 12);
00253         break;
00254     }
00255 }
00259 void HAL_GPIO.SwdReuseGpioEnable(void)
00260 {

```



```

00261 // 系统寄存器解锁
00262 HAL_REG_SYS_UNLOCK();
00263 // 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
00264 HAL_REG_RESET(SYS.IO.CFG, BIT6);
00265 // 系统寄存器上锁
00266 HAL_REG_SYS_LOCK();
00267 }
00271 void HAL_GPIO_SwdReuseGpioDisable(void)
00272 {
00273 // 系统寄存器解锁
00274 HAL_REG_SYS_UNLOCK();
00275 // 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
00276 HAL_REG_SET(SYS.IO.CFG, BIT6);
00277 // 系统寄存器上锁
00278 HAL_REG_SYS_LOCK();
00279 }
00283 void HAL_GPIO_RstReuseGpioEnable(void)
00284 {
00285 // 系统寄存器解锁
00286 HAL_REG_SYS_UNLOCK();
00287 // SYS.IO.CFG[5] RSTn/P0.2 复用控制信号,默认配置为 RSTn 0:RSTn
00288 HAL_REG_SET(SYS.IO.CFG, BIT5);
00289 // 系统寄存器上锁
00290 HAL_REG_SYS_LOCK();
00291 }
00295 void HAL_GPIO_RstReuseGpioDisable(void)
00296 {
00297 // 系统寄存器解锁
00298 HAL_REG_SYS_UNLOCK();
00299 // SYS.IO.CFG[5] RSTn/P0.2 复用控制信号,默认配置为 RSTn 0:RSTn
00300 HAL_REG_RESET(SYS.IO.CFG, BIT5);
00301 // 系统寄存器上锁
00302 HAL_REG_SYS_LOCK();
00303 }

```

4.82 source/hal_lks32mc09x_hall.c 文件参考

```

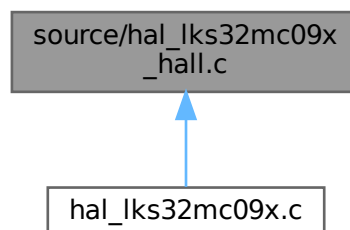
#include "../include/hal_lksmcu_hall.h"
#include "../hal_lks32mc09x.h"
#include "lks32mc09x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc09x_reg.h"

```

hal_lks32mc09x_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, 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
- 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, uint32_t filter_mclk)
臆刷疆婊ゆ尝
- void [HAL_HALL_SetFilteruS](#) ([HAL_HALL_x](#) hall, uint32_t filter_uS)
设置滤波, 单位: μS

4.82.1 函数说明

4.82.1.1 HAL_HALL_Disable()

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

模块关闭

参数

hall	模块号

在文件 [hal_lks32mc09x_hall.c](#) 第 40 行定义.

4.82.1.2 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_lks32mc09x_hall.c](#) 第 28 行定义.

4.82.1.3 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_lks32mc09x_hall.c](#) 第 85 行定义.

4.82.1.4 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_lks32mc09x_hall.c](#) 第 95 行定义.

4.82.1.5 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_lks32mc09x.hall.c](#) 第 75 行定义.

4.82.1.6 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_lks32mc09x.hall.c](#) 第 13 行定义.

4.82.1.7 HAL_HALL_SetFilter()

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

璁剧疆婊0尝

参数

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

在文件 [hal_lks32mc09x.hall.c](#) 第 135 行定义.
这是这个函数的调用关系图:



4.82.1.8 HAL_HALL_SetFilteruS()

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

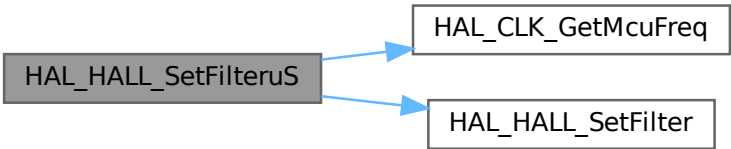
设置滤波，单位：uS

参数

hall	模块号
filter_uS	滤波值

在文件 [hal_lks32mc09x.hall.c](#) 第 167 行定义.

函数调用图:



4.82.1.9 HAL_HALL_SetFreq()

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

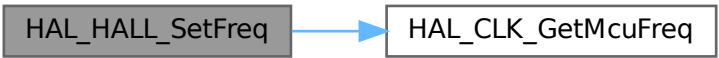
设置hall频率

参数

hall	模块号
freq	频率值

在文件 [hal_lks32mc09x.hall.c](#) 第 53 行定义.

函数调用图:



4.82.1.10 HAL_HALL_SetGpio()

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

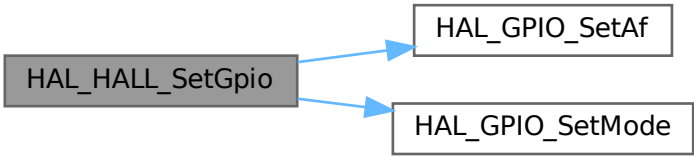
设置hall输入引脚

参数

hall	模块号
px	引脚

在文件 [hal_lks32mc09x_hall.c](#) 第 105 行定义.

函数调用图:



4.82.1.11 HAL_HALL_SetTh()

```
void HAL_HALL_SetTh (
    HAL_HALL_x hall,
    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_lks32mc09x_hall.c](#) 第 65 行定义.

4.82.1.12 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_lks32mc09x_hall.c](#) 第 116 行定义.

4.82.1.13 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_lks32mc09x_hall.c](#) 第 124 行定义.

4.83 hal_lks32mc09x_hall.c

[浏览该文件的文档.](#)

```
00001
00002 #include "../include/hal_lksmcu_hall.h"
00003
00004 #include "../hal_lks32mc09x.h"
00005 #include "lks32mc09x_user_manual.h"
00006 #include "../include/hal_lksmcu_reg.h"
00007 #include "../include/hal_lks32mc09x_reg.h"
00008
00013 void HAL_HALL_Reset(HAL_HALL_x hall)
00014 {
00015     (void)hall;
00016     // 系统寄存器解锁
00017     HAL_REG_SYS_UNLOCK();
00018     HAL_REG_SET(SYS_SFT_RST, SYS_SFT_RST_HALL0_SFT_RST_BIT);
00019     HAL_REG_RESET(SYS_SFT_RST, SYS_SFT_RST_HALL0_SFT_RST_BIT);
00020     // 系统寄存器上锁
```

```

00021     HAL_REG_SYS_UNLOCK();
00022 }
00023
00028 void HAL_HALL_Enable(HAL_HALL_x hall)
00029 {
00030     HAL_REG_SYS_UNLOCK();
00031     HAL_REG_SET(SYS_CLK_FEN, SYS_CLK_FEN_HALL0_CLK_EN_BIT);
00032     HAL_REG_SET(HAL_HALL[hall]->CFG, HALL0_CFG_HALL_EN_BIT);
00033     HAL_REG_SYS_LOCK();
00034 }
00035
00040 void HAL_HALL_Disable(HAL_HALL_x hall)
00041 {
00042     HAL_REG_SYS_UNLOCK();
00043     HAL_REG_RESET(HAL_HALL[hall]->CFG, HALL0_CFG_HALL_EN_BIT);
00044     HAL_REG_RESET(SYS_CLK_FEN, SYS_CLK_FEN_HALL0_CLK_EN_BIT);
00045     HAL_REG_SYS_LOCK();
00046 }
00047
00053 void HAL_HALL_SetFreq(HAL_HALL_x hall, uint32_t freq)
00054 {
00055     uint32_t mclk = HAL_CLK_GetMcuFreq();
00056     uint32_t th = (mclk / freq);
00057     HAL_REG_WRITE(HAL_HALL[hall]->TH, th);
00058 }
00059
00065 void HAL_HALL_SetTh(HAL_HALL_x hall, uint32_t th)
00066 {
00067     HAL_REG_WRITE(HAL_HALL[hall]->TH, th);
00068 }
00069
00075 uint32_t HAL_HALL_ReadTh(HAL_HALL_x hall)
00076 {
00077     return HAL_REG_READ(HAL_HALL[hall]->TH);
00078 }
00079
00085 uint8_t HAL_HALL_ReadHallData(HAL_HALL_x hall)
00086 {
00087     return HAL_REG_READ(HAL_HALL[hall]->INFO);
00088 }
00089
00095 uint32_t HAL_HALL_ReadHallWidth(HAL_HALL_x hall)
00096 {
00097     return HAL_REG_READ(HAL_HALL[hall]->WIDTH);
00098 }
00099
00105 void HAL_HALL_SetGpio(HAL_HALL_x hall, HAL_GPIO_Px px)
00106 {
00107     (void)hall;
00108     HAL_GPIO_SetMode(px, HAL_GPIO_MODE_IN);
00109     HAL_GPIO_SetAf(px, HAL_GPIO_AF_HALL);
00110 }
00111
00116 void HAL_HALL_StartCounting(HAL_HALL_x hall)
00117 {
00118     HAL_REG_SET(HAL_HALL[hall]->CFG, HALL0_CFG_HALL_EN_BIT);
00119 }
00124 void HAL_HALL_StopCounting(HAL_HALL_x hall)
00125 {
00126     HAL_REG_RESET(HAL_HALL[hall]->CFG, HALL0_CFG_HALL_EN_BIT);
00127     HAL_REG_WRITE(HAL_HALL[hall]->CNT, 0);
00128 }
00129
00135 void HAL_HALL_SetFilter(HAL_HALL_x hall, uint32_t filter_mclk)
00136 {
00137     if (filter_mclk < 5)
00138     {
00139         HAL_REG_RESET(HAL_HALL[hall]->CFG, HALL0_CFG_FIL75_BIT); // 小于5个cnt关闭7/5滤波
00140         HAL_REG_WRITE_BIT_MASK_POS(HAL_HALL[hall]->CFG, HALL0_CFG_CLK_DIV, 0);
00141         HAL_REG_WRITE_BIT_MASK_POS(HAL_HALL[hall]->CFG, HALL0_CFG_FIL_LEN, filter_mclk);
00142     }
00143     else
00144     {
00145         uint32_t div = 0;
00146         HAL_REG_SET(HAL_HALL[hall]->CFG, HALL0_CFG_FIL75_BIT); // 大于5个cnt打开7/5滤波
00147         // 自动计算分配系数
00148         for (; filter_mclk > HALL0_CFG_FIL_LEN_MAX;)
00149         {
00150             filter_mclk >>= 1;
00151             div++;
00152         }
00153         if (div > HALL0_CFG_CLK_DIV_MAX)
00154         {
00155             div = HALL0_CFG_CLK_DIV_MAX;
00156             filter_mclk = HALL0_CFG_FIL_LEN_MAX;
00157         }
00158         HAL_REG_WRITE_BIT_MASK_POS(HAL_HALL[hall]->CFG, HALL0_CFG_CLK_DIV, div);

```



```

00159     HAL_REG_WRITE_BIT_MASK_POS(HAL_HALL[hall]->CFG, HALLO_CFG_FILLEN, filter_mclk);
00160 }
00161 }
00167 void HAL_HALL_SetFilteruS(HAL_HALL_x hall, uint32_t filter_uS)
00168 {
00169     uint32_t clk = HAL_CLK_GetMcuFreq();
00170     HAL_HALL_SetFilter(hall, (clk / 1000000) * filter_uS);
00171 }

```

4.84 source/hal_lks32mc09x_i2c.c 文件参考

```

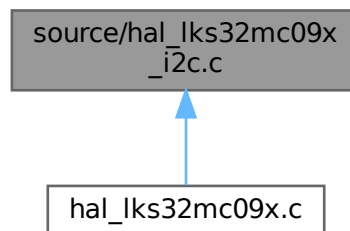
#include "../include/hal_lksmcu_i2c.h"
#include "../hal_lks32mc09x.h"
#include "lks32mc09x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc09x_reg.h"

```

hal_lks32mc09x_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
- 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]

4.84.1 函数说明

4.84.1.1 HAL_I2C_BusReset()

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

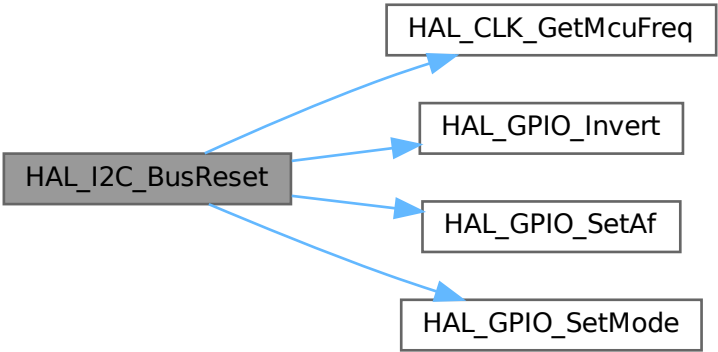
复位I2C总线

参数

i2c_x	I2C模块编号

在文件 [hal_lks32mc09x_i2c.c](#) 第 201 行定义.

函数调用图:



4.84.1.2 HAL_I2C_Disable()

```
void HAL_I2C_Disable (
    HAL_I2C_X i2c_x )
```

关闭I2C模块

参数

$i2c\_x$	I2C模块编号

在文件 [hal_lks32mc09x_i2c.c](#) 第 38 行定义.

4.84.1.3 HAL_I2C_Enable()

```
void HAL_I2C_Enable (
    HAL_I2C_X i2c_x )
```

打开I2C模块

参数

$i2c\_x$	I2C模块编号

在文件 [hal_lks32mc09x_i2c.c](#) 第 26 行定义.

4.84.1.4 HAL_I2C_Reset()

```
void HAL_I2C_Reset (
    HAL_I2C_X i2c_x )
```

复位I2C模块

参数

$i2c\_x$	I2C模块编号

在文件 [hal_lks32mc09x_i2c.c](#) 第 13 行定义.

4.84.1.5 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_lks32mc09x_i2c.c](#) 第 130 行定义.

4.84.1.6 HAL_I2C.SetBaud()

```
void HAL_I2C_SetBaud (
    HAL_I2C_x i2c_x,
    uint32_t baud )
```

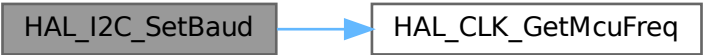
设置I2C模块波特率

参数

<i>i2c_x</i>	I2C模块编号
<i>baud</i>	波特率

在文件 [hal_lks32mc09x_i2c.c](#) 第 50 行定义.

函数调用图:



4.84.1.7 HAL_I2C.SetGpioScl()

```
void HAL_I2C_SetGpioScl (
```

```
HAL_I2C_x i2c_x,  
HAL_GPIO_Px i2cgpio )
```

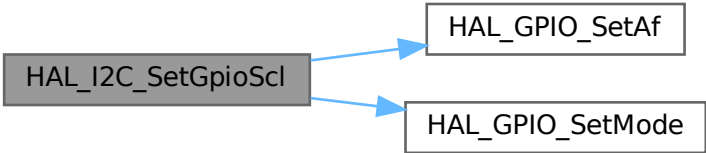
设置I2C模块的GPIO

参数

<i>i2c_x</i>	I2C模块编号
<i>i2cgpio</i>	GPIO编号

在文件 [hal_lks32mc09x_i2c.c](#) 第 178 行定义.

函数调用图:



4.84.1.8 HAL_I2C.SetGpioSda()

```
void HAL_I2C_SetGpioSda (  
    HAL_I2C_x i2c_x,  
    HAL_GPIO_Px i2cgpio )
```

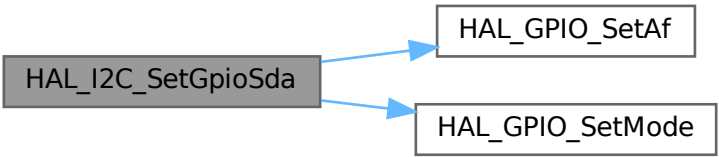
设置I2C模块的GPIO

参数

<i>i2c_x</i>	I2C模块编号
<i>i2cgpio</i>	GPIO编号

在文件 [hal_lks32mc09x_i2c.c](#) 第 190 行定义.

函数调用图:



4.84.1.9 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_lks32mc09x_i2c.c](#) 第 69 行定义.

4.84.2 变量说明

4.84.2.1 hal_baud

```
uint32_t hal_baud[1] [static]
```

在文件 [hal_lks32mc09x_i2c.c](#) 第 8 行定义.

4.84.2.2 hal_i2cio

```
HAL_GPIO_Px hal_i2cio[1][2] [static]
```

在文件 `hal_lks32mc09x_i2c.c` 第 7 行定义.

4.85 hal_lks32mc09x_i2c.c

浏览该文件的文档.

```
00001
00002 #include "../include/hal_lksmcu_i2c.h"
00003 #include "../hal_lks32mc09x.h"
00004 #include "lks32mc09x.user_manual.h"
00005 #include "../include/hal_lksmcu_reg.h"
00006 #include "../include/hal_lks32mc09x_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_I2C0_SFT_RST_BIT);
00018     HAL_REG_RESET(SYS_SFT_RST, SYS_SFT_RST_I2C0_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_I2C0_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_I2C0_CLK_EN_BIT);
00043     HAL_REG_SYS_LOCK();
00044 }
00050 void HAL_I2C_SetBaud(HAL_I2C_x i2c_x, uint32_t baud)
00051 {
00052     (void)i2c_x;
00053     hal_baud[i2c_x] = baud;
00054     u32 BaudRate;
00055     BaudRate = ((u32)HAL_CLK_GetMcuFreq() + ((baud * 17) >> 1)) / (baud * 17) - 1;
00056     HAL_REG_SYS_UNLOCK();
00057     HAL_REG_WRITE(SYS_CLK_DIV0, BaudRate);
00058     HAL_REG_SYS_LOCK();
00059 }
00060
00069 uint8_t HAL_I2C_TxData(HAL_I2C_x i2c_x, uint8_t addr, uint8_t *data, uint32_t len)
00070 {
00071     uint32_t i = 0;
00072     HAL_REG_SET(HAL_I2C[i2c_x]->CFG, I2C0_CFG_MST_MODE_BIT);
00073     if (HAL_I2C[i2c_x]->MSCR & BIT3)
00074     {
00075         return 1; // 发送数据时,i2c总线忙碌
00076     }
00077     if (len == 0)
00078     {
00079         return 1; // 不支持仅发送一个地址
00080     }
00081     HAL_I2C[i2c_x]->SCR = 0;
00082     HAL_I2C[i2c_x]->MSCR = 1; // 触发I2C发送数据
00083     while ((HAL_I2C[i2c_x]->MSCR & BIT2) == 0)
00084     {
00085     }
00086     HAL_I2C[i2c_x]->DATA = addr;
00087     while (HAL_I2C[i2c_x]->SCR == 0) // 等待地址发送完成
00088     {
00089     }
00090
00091     if (HAL_I2C[i2c_x]->SCR & BIT1) // 收到nack
00092     {
00093         HAL_I2C[i2c_x]->SCR = 0;
00094         while (HAL_I2C[i2c_x]->MSCR & BIT3) // 等待stop信号发送完成
```

```

00095     {
00096     }
00097     return 1;
00098 }
00099 for (i = 0; i < len; i++)
00100 {
00101     HAL_I2C[i2c_x]->DATA = data[i];
00102     HAL_I2C[i2c_x]->SCR = BIT2; // 触发送
00103     while ((HAL_I2C[i2c_x]->SCR & BIT0) == 0) // 等待数据发送完成
00104     {
00105     }
00106     if (HAL_I2C[i2c_x]->SCR & BIT1) // 收到nack
00107     {
00108         HAL_I2C[i2c_x]->SCR = 0;
00109         while (HAL_I2C[i2c_x]->MSCR & BIT3) // 等待stop信号发送完成
00110         {
00111         }
00112         return 1;
00113     }
00114 }
00115 HAL_I2C[i2c_x]->SCR = 0;
00116 while (HAL_I2C[i2c_x]->MSCR & BIT3) // 等待stop信号发送完成
00117 {
00118 }
00119 return 0;
00120 }
00121
00130 uint8_t HAL_I2C_RxData(HAL_I2C_TypeDef* i2c_x, uint8_t* addr, uint8_t* data, uint32_t len)
00131 {
00132     uint32_t i;
00133     HAL_REG_SET(HAL_I2C[i2c_x]->CFG, I2C0_CFG_MST_MODE_BIT);
00134     if (HAL_I2C[i2c_x]->MSCR & BIT3)
00135     {
00136         return 1; // 发送数据时,i2c总线忙碌
00137     }
00138     if (len == 0)
00139     {
00140         return 1; // 不支持仅发送一个地址
00141     }
00142     HAL_I2C[i2c_x]->SCR = 0;
00143     HAL_I2C[i2c_x]->MSCR = 1; // 触发I2C发送数据
00144     while ((HAL_I2C[i2c_x]->MSCR & BIT2) == 0) // 等待start信号发送完成
00145     {
00146     }
00147     HAL_I2C[i2c_x]->DATA = addr | 1; // 发送地址数据
00148     while (HAL_I2C[i2c_x]->SCR == 0) // 等待地址发送完成
00149     {
00150     }
00151     if (HAL_I2C[i2c_x]->SCR & BIT1) // 收到nack
00152     {
00153         HAL_I2C[i2c_x]->SCR = 0;
00154         while (HAL_I2C[i2c_x]->MSCR & BIT3) // 等待stop信号发送完成
00155         {
00156         }
00157         return 1;
00158     }
00159     for (i = 0; i < len; i++)
00160     {
00161         HAL_REG_WRITE(HAL_I2C[i2c_x]->SCR, I2C0_SCR_BYTE_CMPLT_BIT);
00162         while ((HAL_I2C[i2c_x]->SCR & BIT0) == 0) // 等待数据发送完成
00163         {
00164         }
00165         data[i] = HAL_I2C[i2c_x]->DATA;
00166     }
00167     HAL_I2C[i2c_x]->SCR = 0;
00168     while (HAL_I2C[i2c_x]->MSCR & BIT3) // 等待stop信号发送完成
00169     {
00170     }
00171     return 0;
00172 }
00173 void HAL_I2C_SetGpioScl(HAL_I2C_TypeDef* i2c_x, HAL_GPIO_TypeDef* i2cgpio)
00174 {
00175     (void)i2c_x;
00176     HAL_GPIO_SetAf(i2cgpio, HAL_GPIO_AF_I2C);
00177     HAL_GPIO_SetMode(i2cgpio, HAL_GPIO_MODE_IN);
00178     hal_i2cio[i2c_x][0] = i2cgpio; // 保存I2C使用的IO口
00179 }
00180 void HAL_I2C_SetGpioSda(HAL_I2C_TypeDef* i2c_x, HAL_GPIO_TypeDef* i2cgpio)
00181 {
00182     (void)i2c_x;
00183     HAL_GPIO_SetAf(i2cgpio, HAL_GPIO_AF_I2C);
00184     HAL_GPIO_SetMode(i2cgpio, HAL_GPIO_MODE_IN);
00185     hal_i2cio[i2c_x][1] = i2cgpio; // 保存I2C使用的IO口
00186 }
00187 void HAL_I2C_BusReset(HAL_I2C_TypeDef* i2c_x)
00188 {
00189     uint32_t div;

```



```

00204     uint32_t i;
00205     uint32_t j;
00206     // 还原成普通IO口
00207     HAL_GPIO_SetAf(hal_i2cio[i2c.x][0], HAL_GPIO_AF_GPIO);
00208     HAL_GPIO_SetAf(hal_i2cio[i2c.x][1], HAL_GPIO_AF_GPIO);
00209     HAL_GPIO_SetMode(hal_i2cio[i2c.x][0], HAL_GPIO_MODE_INUPOUT_1);
00210     HAL_GPIO_SetMode(hal_i2cio[i2c.x][1], HAL_GPIO_MODE_INUP);
00211     // 翻转10次SCL
00212     div = (HAL_CLK_GetMcuFreq() / hal_baud[i2c.x]) >> 4;
00213     for (i = 0; i < 20; i++)
00214     {
00215         HAL_GPIO_Invert(hal_i2cio[i2c.x][0]);
00216         for (j = div; j--;) // 一个大于波特率的软件延迟
00217         {
00218             __NOP();
00219         }
00220     }
00221     // 复用回I2C
00222     HAL_GPIO_SetAf(hal_i2cio[i2c.x][0], HAL_GPIO_AF_I2C);
00223     HAL_GPIO_SetAf(hal_i2cio[i2c.x][1], HAL_GPIO_AF_I2C);
00224     HAL_GPIO_SetMode(hal_i2cio[i2c.x][0], HAL_GPIO_MODE_IN);
00225     HAL_GPIO_SetMode(hal_i2cio[i2c.x][1], HAL_GPIO_MODE_IN);
00226 }

```

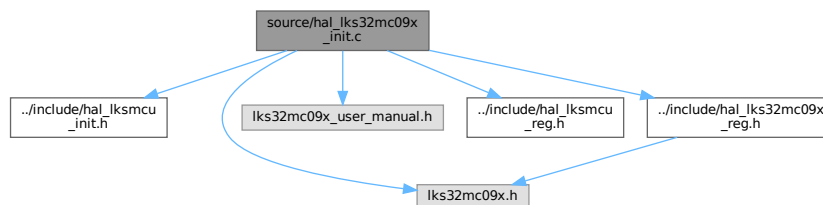
4.86 source/hal_lks32mc09x_init.c 文件参考

```

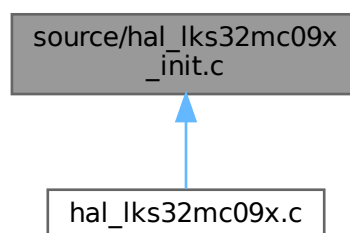
#include "../include/hal_lksmcu_init.h"
#include "lks32mc09x.h"
#include "lks32mc09x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc09x_reg.h"

```

hal_lks32mc09x_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)

4.86.1 函数说明

4.86.1.1 HAL_INIT_PowerReady()

```
void HAL_INIT_PowerReady (
    HAL_INIT_POWER_x powerx )
```

在文件 [hal_lks32mc09x_init.c](#) 第 14 行定义.

4.86.1.2 HAL_INIT_SysInit()

```
void HAL_INIT_SysInit (
    void )
```

在文件 [hal_lks32mc09x_init.c](#) 第 6 行定义.

4.87 hal_lks32mc09x_init.c

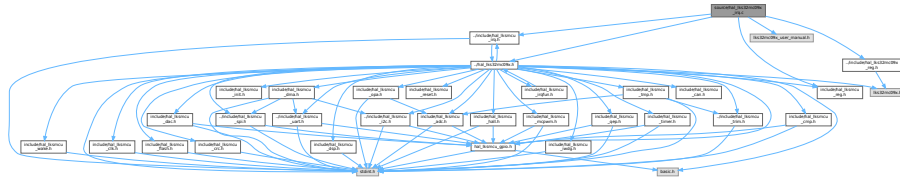
[浏览该文件的文档.](#)

```
00001 #include "../include/hal_lksmcu_init.h"
00002 #include "lks32mc09x.h"
00003 #include "lks32mc09x.user_manual.h"
00004 #include "../include/hal_lksmcu_reg.h"
00005 #include "../include/hal_lks32mc09x_reg.h"
00006 void HAL_INIT_SysInit(void)
00007 {
00008     IWDG.CFG          = 0x3c00;
00009     SYS_WR_PROTECT    = 0x7a83;    /* 解除系统寄存器写保护 */
00010     SYS_AFE_REG5      = BIT15;    /* BIT15: PLLPDN & PLL */
00011     SYS_CLK_CFG       = 0x000001ff; /* BIT8:0: CLK_HS, 1: PLL | BIT[7:0] CLK_DIV | 1fff对应96M时钟 */
00012 }
00013
00014 void HAL_INIT_PowerReady(HAL_INIT_POWER_x powerx)
00015 {
00016     switch (powerx)
00017     {
00018         case HAL_INIT_POWER_3_3V: // 等待3.3v电源就绪, 判断阈值为2.8V
00019             __disable_irq();
00020             // 设置掉电检测电压
00021             HAL_REG_SYS_UNLOCK();
00022             HAL_REG_WRITE_BIT_MASK_POS(HAL_SYS[0]->AFE_REG5, SYS_AFE_REG5_RSTSEL, 1);
00023             HAL_REG_WRITE_BIT_MASK_POS(HAL_SYS[0]->AFE_REG0, SYS_AFE_REG0_PVDSEL, 3);
00024             HAL_REG_RESET(HAL_SYS[0]->AFE_REG0, (SYS_AFE_REG0_PD_PDT_BIT | SYS_AFE_REG0_VSR_PDT_BIT));
00025             HAL_REG_SYS_LOCK();
00026             // 等待电源就绪
00027             while (HAL_REG_READ_BIT(HAL_SYS[0]->AFE_DBG, SYS_AFE_DBG_PWR_WEAK_BIT))
00028             {
00029                 __NOP();
00030             }
00031             break;
00032         case HAL_INIT_POWER_5V: // 等待5.0v电源就绪, 判断阈值为4.0V
00033             __disable_irq();
00034             // 设置掉电检测电压
00035             HAL_REG_SYS_UNLOCK();
00036             HAL_REG_WRITE_BIT_MASK_POS(HAL_SYS[0]->AFE_REG5, SYS_AFE_REG5_RSTSEL, 1);
00037             HAL_REG_WRITE_BIT_MASK_POS(HAL_SYS[0]->AFE_REG0, SYS_AFE_REG0_PVDSEL, 0);
00038             HAL_REG_RESET(HAL_SYS[0]->AFE_REG0, (SYS_AFE_REG0_PD_PDT_BIT | SYS_AFE_REG0_VSR_PDT_BIT));
00039             HAL_REG_SYS_LOCK();
00040             // 等待电源就绪
00041             while (HAL_REG_READ_BIT(HAL_SYS[0]->AFE_DBG, SYS_AFE_DBG_PWR_WEAK_BIT))
00042             {
00043                 __NOP();
00044             }
00045             break;
00046     }
00047     // 清除中断标志位
00048     NVIC_ClearPendingIRQ(PWRDN_IRQn);
00049 }
```

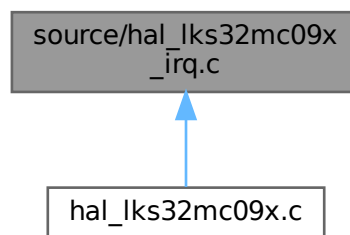
4.88 source/hal_lks32mc09x_irq.c 文件参考

```
#include "../include/hal_lksmcu_irq.h"
#include "../hal_lks32mc09x.h"
#include "lks32mc09x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc09x_reg.h"
```

hal_lks32mc09x_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

4.88.1 函数说明

4.88.1.1 HAL_IRQ_Disable()

```
void HAL_IRQ_Disable (
    void )
```

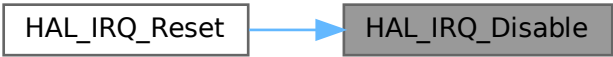
关闭全局中断

参数

<i>irq</i>	中断编号

在文件 [hal_lks32mc09x_irq.c](#) 第 29 行定义.

这是这个函数的调用关系图:



4.88.1.2 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_lks32mc09x_irq.c](#) 第 20 行定义.

4.88.1.3 HAL_IRQ_ModuleDisable()

```
void HAL_IRQ_ModuleDisable (  
    HAL_IRQ_MODULE_x module )
```

关闭某个模块的中断

参数

<i>irq</i>	中断编号
<i>module</i>	模块编号
<i>number</i>	中断号

在文件 [hal_lks32mc09x_irq.c](#) 第 52 行定义.

4.88.1.4 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>irq</i>	中断编号
<i>module</i>	模块编号
<i>number</i>	中断号
<i>priority</i>	中断优先级

在文件 [hal_lks32mc09x_irq.c](#) 第 41 行定义.

4.88.1.5 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_lks32mc09x_irq.c](#) 第 11 行定义.

函数调用图:



4.89 hal_lks32mc09x_irq.c

[浏览该文件的文档.](#)

```
00001 #include "../include/hal_lksmcu_irq.h"
00002 #include "../hal_lks32mc09x.h"
00003 #include "lks32mc09x_user_manual.h"
00004 #include "../include/hal_lksmcu_reg.h"
00005 #include "../include/hal_lks32mc09x_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
00041 void HAL_IRQ_ModuleEnable(HAL_IRQ_MODULE_x module, uint8_t priority)
00042 {
00043     NVIC.SetPriority((IRQn.Type)module, priority);
00044     NVIC.EnableIRQ((IRQn.Type)module);
00045 }
00052 void HAL_IRQ_ModuleDisable(HAL_IRQ_MODULE_x module)
00053 {
00054     NVIC.DisableIRQ((IRQn.Type)module);
00055 }

```

4.90 source/hal_lks32mc09x_irqfun.c 文件参考

```

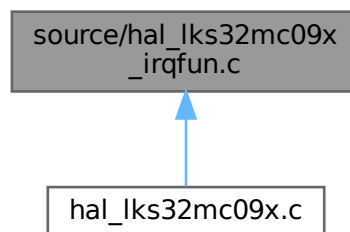
#include "../hal_lks32mc09x.h"
#include "lks32mc09x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc09x_reg.h"
#include "../include/hal_lksmcu_irqfun.h"

```

hal_lks32mc09x_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

4.90.1 函数说明

4.90.1.1 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_lks32mc09x_irqfun.c](#) 第 128 行定义.

4.90.1.2 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_lks32mc09x_irqfun.c](#) 第 175 行定义.

4.90.1.3 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_lks32mc09x_irqfun.c](#) 第 167 行定义.

4.90.1.4 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_lks32mc09x_irqfun.c](#) 第 159 行定义.

4.90.1.5 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_lks32mc09x_irqfun.c](#) 第 144 行定义.

4.90.1.6 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_lks32mc09x_irqfun.c](#) 第 185 行定义.

4.90.1.7 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_lks32mc09x_irqfun.c](#) 第 232 行定义.

4.90.1.8 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_lks32mc09x_irqfun.c](#) 第 224 行定义.

4.90.1.9 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_lks32mc09x_irqfun.c](#) 第 216 行定义.

4.90.1.10 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_lks32mc09x_irqfun.c](#) 第 201 行定义.

4.90.1.11 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_lks32mc09x_irqfun.c](#) 第 14 行定义.

4.90.1.12 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_lks32mc09x_irqfun.c](#) 第 61 行定义.

4.90.1.13 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_lks32mc09x_irqfun.c](#) 第 53 行定义.

4.90.1.14 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_lks32mc09x_irqfun.c](#) 第 45 行定义.

4.90.1.15 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_lks32mc09x_irqfun.c](#) 第 30 行定义.

4.90.1.16 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_lks32mc09x_irqfun.c](#) 第 71 行定义.

4.90.1.17 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_lks32mc09x_irqfun.c](#) 第 118 行定义.

4.90.1.18 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_lks32mc09x_irqfun.c](#) 第 110 行定义.

4.90.1.19 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_lks32mc09x_irqfun.c](#) 第 102 行定义.

4.90.1.20 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_lks32mc09x_irqfun.c` 第 87 行定义.

4.91 hal_lks32mc09x_irqfun.c

[浏览该文件的文档.](#)

```

00001
00002
00003 #include "../hal_lks32mc09x.h"
00004 #include "lks32mc09x_user_manual.h"
00005 #include "../include/hal_lksmcu_reg.h"
00006 #include "../include/hal_lks32mc09x_reg.h"
00007 #include "../include/hal_lksmcu_irqfun.h"
00008
00014 uint8_t HAL_HALL_IrqChange(HAL_HALLx hall)
00015 {
00016     if ((HAL_REG_READ_BIT(HAL_HALL[hall]->CFG, HALL0_CFG_CHG_IE_BIT)) &&
00017         (HAL_REG_READ_BIT(HAL_HALL[hall]->INFO, HALL0_INFO_CHG_IF_BIT)))
00018     {
00019         return 1;
00020     }
00021     else
00022     {
00023         return 0;
00024     }
00025 }
00030 uint8_t HAL_HALL_IrqChangeFlg(HAL_HALLx hall)
00031 {
00032     if (HAL_REG_READ_BIT(HAL_HALL[hall]->INFO, HALL0_INFO_CHG_IF_BIT))
00033     {
00034         return 1;
00035     }
00036     else
00037     {
00038         return 0;
00039     }
00040 }
00045 void HAL_HALL_IrqChangeEnable(HAL_HALLx hall)
00046 {
00047     HAL_REG_SET(HAL_HALL[hall]->CFG, HALL0_CFG_CHG_IE_BIT);
00048 }
00053 void HAL_HALL_IrqChangeDisable(HAL_HALLx hall)
00054 {
00055     HAL_REG_RESET(HAL_HALL[hall]->CFG, HALL0_CFG_CHG_IE_BIT);
00056 }
00061 void HAL_HALL_IrqChangeClear(HAL_HALLx hall)
00062 {
00063     HAL_REG_WRITE(HAL_HALL[hall]->INFO, HALL0_INFO_CHG_IF_BIT);
00064 }
00065
00071 uint8_t HAL_HALL_IrqOver(HAL_HALLx hall)
00072 {
00073     if ((HAL_REG_READ_BIT(HAL_HALL[hall]->CFG, HALL0_CFG_OV_IE_BIT)) &&
00074         (HAL_REG_READ_BIT(HAL_HALL[hall]->INFO, HALL0_INFO_OV_IF_BIT)))
00075     {
00076         return 1;
00077     }
00078     else
00079     {
00080         return 0;
00081     }
00082 }
00087 uint8_t HAL_HALL_IrqOverFlg(HAL_HALLx hall)
00088 {
00089     if (HAL_REG_READ_BIT(HAL_HALL[hall]->INFO, HALL0_INFO_OV_IF_BIT))
00090     {
00091         return 1;
00092     }
00093     else
00094     {
00095         return 0;
00096     }
00097 }
00102 void HAL_HALL_IrqOverEnable(HAL_HALLx hall)
00103 {
00104     HAL_REG_SET(HAL_HALL[hall]->CFG, HALL0_CFG_OV_IE_BIT);
00105 }
00110 void HAL_HALL_IrqOverDisable(HAL_HALLx hall)
00111 {
00112     HAL_REG_RESET(HAL_HALL[hall]->CFG, HALL0_CFG_OV_IE_BIT);
00113 }
00118 void HAL_HALL_IrqOverClear(HAL_HALLx hall)

```

```

00119 {
00120     HAL_REG_WRITE(HAL_HALL[hal1]->INFO, HAL0_INFO_OV_IF_BIT);
00121 }
00122
00128 uint8_t HAL_ADC_IrqSamp1(HAL_ADC_x adc)
00129 {
00130     if ((HAL_REG_READ_BIT(HAL_ADC[adc]->IE, ADC0_IE_SF1_IE_BIT)) && (HAL_REG_READ_BIT(HAL_ADC[adc]->IF,
00131     ADC0_IF_SF1_IF_BIT)))
00132     {
00133         return 1;
00134     }
00135     else
00136     {
00137         return 0;
00138     }
00139 }
00144 uint8_t HAL_ADC_IrqSamp1Flg(HAL_ADC_x adc)
00145 {
00146     if (HAL_REG_READ_BIT(HAL_ADC[adc]->IF, ADC0_IF_SF1_IF_BIT))
00147     {
00148         return 1;
00149     }
00150     else
00151     {
00152         return 0;
00153     }
00154 }
00159 void HAL_ADC_IrqSamp1Enable(HAL_ADC_x adc)
00160 {
00161     HAL_REG_SET(HAL_ADC[adc]->IE, ADC0_IE_SF1_IE_BIT);
00162 }
00167 void HAL_ADC_IrqSamp1Disable(HAL_ADC_x adc)
00168 {
00169     HAL_REG_RESET(HAL_ADC[adc]->IE, ADC0_IE_SF1_IE_BIT);
00170 }
00175 void HAL_ADC_IrqSamp1Clear(HAL_ADC_x adc)
00176 {
00177     HAL_REG_WRITE(HAL_ADC[adc]->IF, ADC0_IF_SF1_IF_BIT);
00178 }
00179
00185 uint8_t HAL_ADC_IrqSamp2(HAL_ADC_x adc)
00186 {
00187     if ((HAL_REG_READ_BIT(HAL_ADC[adc]->IE, ADC0_IE_SF2_IE_BIT)) && (HAL_REG_READ_BIT(HAL_ADC[adc]->IF,
00188     ADC0_IF_SF2_IF_BIT)))
00189     {
00190         return 1;
00191     }
00192     else
00193     {
00194         return 0;
00195     }
00196 }
00201 uint8_t HAL_ADC_IrqSamp2Flg(HAL_ADC_x adc)
00202 {
00203     if (HAL_REG_READ_BIT(HAL_ADC[adc]->IF, ADC0_IF_SF2_IF_BIT))
00204     {
00205         return 1;
00206     }
00207     else
00208     {
00209         return 0;
00210     }
00211 }
00216 void HAL_ADC_IrqSamp2Enable(HAL_ADC_x adc)
00217 {
00218     HAL_REG_SET(HAL_ADC[adc]->IE, ADC0_IE_SF2_IE_BIT);
00219 }
00224 void HAL_ADC_IrqSamp2Disable(HAL_ADC_x adc)
00225 {
00226     HAL_REG_RESET(HAL_ADC[adc]->IE, ADC0_IE_SF2_IE_BIT);
00227 }
00232 void HAL_ADC_IrqSamp2Clear(HAL_ADC_x adc)
00233 {
00234     HAL_REG_WRITE(HAL_ADC[adc]->IF, ADC0_IF_SF2_IF_BIT);
00235 }

```

4.92 source/hal_lks32mc09x_iwdg.c 文件参考

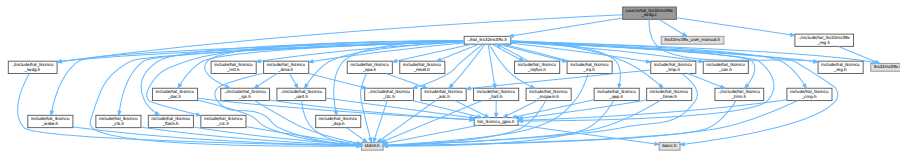
```

#include "../include/hal_lksmcu_iwdg.h"
#include "../hal_lks32mc09x.h"

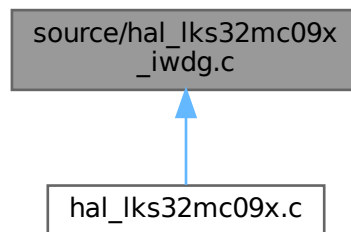
```



```
#include "lks32mc09x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc09x_reg.h"
hal_lks32mc09x_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

4.92.1 函数说明

4.92.1.1 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_lks32mc09x_iwdg.c](#) 第 28 行定义.

这是这个函数的调用关系图:



4.92.1.2 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_lks32mc09x_iwdg.c](#) 第 19 行定义.

4.92.1.3 HAL_IWDG_Feed()

```
void HAL_IWDG_Feed (
    HAL_IWDG_x iwdg )
```

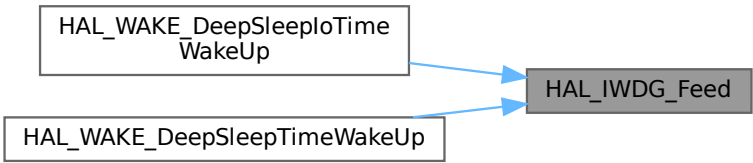
喂狗

参数

<i>iwdg</i>	看门狗模块

在文件 [hal_lks32mc09x_iwdg.c](#) 第 52 行定义.

这是这个函数的调用关系图:



4.92.1.4 HAL_IWDG_Reset()

```
void HAL_IWDG_Reset (
    HAL_IWDG_x iwdg )
```

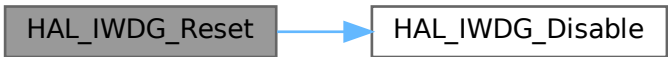
模块复位

参数

<i>iwdg</i>	看门狗模块

在文件 [hal_lks32mc09x.iwdg.c](#) 第 10 行定义.

函数调用图:



4.92.1.5 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_lks32mc09x_iwdg.c` 第 40 行定义。

4.93 hal_lks32mc09x_iwdg.c

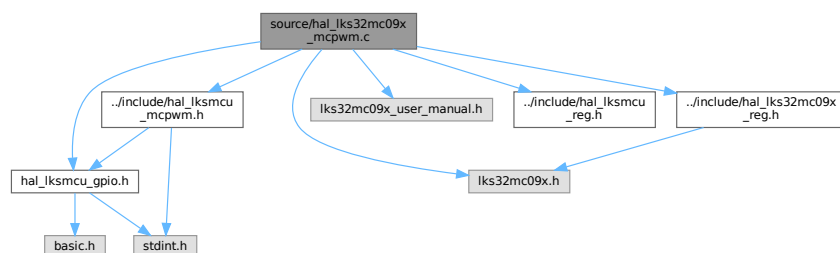
浏览该文件的文档。

```
00001 #include "../include/hal_lksmcu_iwdg.h"
00002 #include "../hal_lks32mc09x.h"
00003 #include "lks32mc09x_user_manual.h"
00004 #include "../include/hal_lksmcu_reg.h"
00005 #include "../include/hal_lks32mc09x_reg.h"
00010 void HAL_IWDG_Reset(HAL_IWDG_x iwdg)
00011 {
00012     HAL_IWDG_Disable(iwdg);
00013 }
00014
00019 void HAL_IWDG_Enable(HAL_IWDG_x iwdg)
00020 {
00021     HAL_REG_SET(HAL_IWDG[iwdg]->CFG, IWDG_CFG_WDG_EN_BIT);
00022 }
00023
00028 void HAL_IWDG_Disable(HAL_IWDG_x iwdg)
00029 {
00030     (void)iwdg;
00031     // 写0同时向[15:8]写入0x3c可清零
00032     HAL_REG_WRITE_BIT(IWDG_CFG, 0x3c00 | IWDG_CFG_WDG_EN_BIT, 0x3c00);
00033 }
00034
00040 void HAL_IWDG_SetResetTimemS(HAL_IWDG_x iwdg, int32_t ms)
00041 {
00042     uint32_t th;
00043     th = ((uint32_t)(32 * ms) >> IWDG_RTH_RTH_POS);
00044     HAL_REG_WRITE(HAL_IWDG[iwdg]->PSW, 0xA6B4);
00045     HAL_REG_WRITE_BIT_MASK_POS(IWDG_RTH, IWDG_RTH_RTH, th);
00046 }
00047
00052 void HAL_IWDG_Feed(HAL_IWDG_x iwdg)
00053 {
00054     (void)iwdg;
00055     HAL_REG_WRITE(HAL_IWDG[0]->PSW, 0xA6B4);
00056     HAL_REG_WRITE(HAL_IWDG[0]->CLR, 0x798D);
00057 }
```

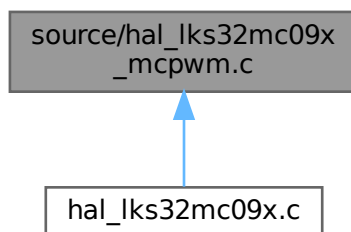
4.94 source/hal_lks32mc09x_mcpwm.c 文件参考

```
#include "../include/hal_lksmcu_mcpwm.h"
#include "../include/hal_lksmcu_gpio.h"
#include "lks32mc09x.h"
#include "lks32mc09x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc09x_reg.h"
```

hal_lks32mc09x_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)
内部使用,输出端口的状态
- 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_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
- 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_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)
设置占空比
- 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_x`↔
`polarity_x`)
设置- 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`)
- void `HAL_MCPWM_SetFailSignal` (`HAL_MCPWM_x` `mcpwm_x`, `HAL_MCPWM_FAIL_x` `fail_x`)
设置FAIL信号来源
- void `HAL_MCPWM_SetAdcTrig` (`HAL_MCPWM_x` `mcpwm_x`, `int32_t` `t0`, `int32_t` `t1`)
- void `HAL_MCPWM_SetAdcTrigQ15` (`HAL_MCPWM_x` `mcpwm_x`, `int32_t` `t0`, `int32_t` `t1`)
- `uint8_t` `HAL_MCPWM_IrqOverflow` (`HAL_MCPWM_x` `mcpwm_x`)
- void `HAL_MCPWM_IrqOverflowEnable` (`HAL_MCPWM_x` `mcpwm_x`)
- void `HAL_MCPWM_IrqOverflowDisable` (`HAL_MCPWM_x` `mcpwm_x`)
- void `HAL_MCPWM_IrqOverflowClear` (`HAL_MCPWM_x` `mcpwm_x`)
- `uint8_t` `HAL_MCPWM_IrqZeroCross` (`HAL_MCPWM_x` `mcpwm_x`)
- void `HAL_MCPWM_IrqZeroCrossEnable` (`HAL_MCPWM_x` `mcpwm_x`)
- void `HAL_MCPWM_IrqZeroCrossDisable` (`HAL_MCPWM_x` `mcpwm_x`)
- void `HAL_MCPWM_IrqZeroCrossClear` (`HAL_MCPWM_x` `mcpwm_x`)
- `uint8_t` `HAL_MCPWM_IrqFail` (`HAL_MCPWM_x` `mcpwm_x`)
- void `HAL_MCPWM_IrqFailEnable` (`HAL_MCPWM_x` `mcpwm_x`)
- void `HAL_MCPWM_IrqFailDisable` (`HAL_MCPWM_x` `mcpwm_x`)
- void `HAL_MCPWM_IrqFailClear` (`HAL_MCPWM_x` `mcpwm_x`)

变量

- static `uint16_t` `mcpwm_polarity_state` [1][6] = {{0, 0, 0, 0, 0, 0}}
- bit0* 通道交换使能 0 交换 1 交换 *bit1* 上管有效电平 0 上管低有效 1 上管高有效 *bit2* 下管有效电平 0 下管低有效 1 下管高有效 *bit3* 默认上管电平 0 关机后上管关闭 1 关机后上管打开 *bit4* 默认下管电平 0 关机后下管关闭 1 关机后下管打开 *bit5-bit7* 输出状态 `HAL_MCPWM_OUTPUT_STATE_x`

4.94.1 函数说明

4.94.1.1 HAL_MCPWM_Disable()

```
void HAL_MCPWM_Disable (
    HAL_MCPWM_x mcpwm_x )
```

mcpwm模块关闭

参数

<code>mcpwm_x</code> ↔ _x	操作的模块编号

在文件 `hal_iks32mc09x_mcpwm.c` 第 191 行定义.

4.94.1.2 HAL_MCPWM_Enable()

```
void HAL_MCPWM_Enable (
    HAL_MCPWM_X mcpwm_x )
```

mcpwm模块使能

参数

mcpwm↔ _x	操作的模块编号

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 172 行定义.

4.94.1.3 HAL_MCPWM_GetFreq()

```
uint32_t HAL_MCPWM_GetFreq (
    HAL_MCPWM_X mcpwm_x )
```

获取pwm频率

参数

mcpwm↔ _x	操作的模块编号

返回

当前pwm的频率

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 221 行定义.

函数调用图:



4.94.1.4 HAL_MCPWM_GetThCnt()

```
uint32_t HAL_MCPWM_GetThCnt (
    HAL_MCPWM_X mcpwm_x )
```

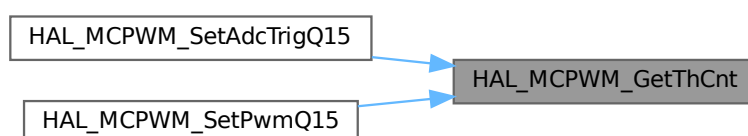
获取计数器最大值

参数

<code>mcpwm_x</code>	操作的模块编号

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 232 行定义.

这是这个函数的调用关系图:



4.94.1.5 HAL_MCPWM_IrqFail()

```
uint8_t HALMCPWM_IrqFail (
    HAL_MCPWM_x mcpwm_x )
```

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 494 行定义.

4.94.1.6 HAL_MCPWM_IrqFailClear()

```
void HALMCPWM_IrqFailClear (
    HAL_MCPWM_x mcpwm_x )
```

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 517 行定义.

4.94.1.7 HAL_MCPWM_IrqFailDisable()

```
void HALMCPWM_IrqFailDisable (
    HAL_MCPWM_x mcpwm_x )
```

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 511 行定义.

4.94.1.8 HAL_MCPWM_IrqFailEnable()

```
void HALMCPWM_IrqFailEnable (
    HAL_MCPWM_x mcpwm_x )
```

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 505 行定义.

4.94.1.9 HAL_MCPWM_IrqOverflow()

```
uint8_t HAL_MCPWM_IrqOverflow (
    HAL_MCPWM_X mcpwm_x )
```

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 438 行定义.

4.94.1.10 HAL_MCPWM_IrqOverflowClear()

```
void HAL_MCPWM_IrqOverflowClear (
    HAL_MCPWM_X mcpwm_x )
```

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 461 行定义.

4.94.1.11 HAL_MCPWM_IrqOverflowDisable()

```
void HAL_MCPWM_IrqOverflowDisable (
    HAL_MCPWM_X mcpwm_x )
```

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 455 行定义.

4.94.1.12 HAL_MCPWM_IrqOverflowEnable()

```
void HAL_MCPWM_IrqOverflowEnable (
    HAL_MCPWM_X mcpwm_x )
```

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 449 行定义.

4.94.1.13 HAL_MCPWM_IrqZeroCross()

```
uint8_t HAL_MCPWM_IrqZeroCross (
    HAL_MCPWM_X mcpwm_x )
```

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 466 行定义.

4.94.1.14 HAL_MCPWM_IrqZeroCrossClear()

```
void HAL_MCPWM_IrqZeroCrossClear (
    HAL_MCPWM_X mcpwm_x )
```

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 489 行定义.

4.94.1.15 HAL_MCPWM_IrqZeroCrossDisable()

```
void HAL_MCPWM_IrqZeroCrossDisable (
    HAL_MCPWM_X mcpwm_x )
```

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 483 行定义.

4.94.1.16 HAL_MCPWM_IrqZeroCrossEnable()

```
void HAL_MCPWM_IrqZeroCrossEnable (
    HAL_MCPWM_X mcpwm_x )
```

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 477 行定义.

4.94.1.17 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_lks32mc09x_mcpwm.c](#) 第 158 行定义.

4.94.1.18 HAL_MCPWM_SetAdcTrig()

```
void HAL_MCPWM_SetAdcTrig (
    HAL_MCPWM_X mcpwm_x,
    int32_t t0,
    int32_t t1 )
```

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 422 行定义.

4.94.1.19 HAL_MCPWM_SetAdcTrigQ15()

```
void HAL_MCPWM_SetAdcTrigQ15 (
    HAL_MCPWM_X mcpwm_x,
    int32_t t0,
    int32_t t1 )
```

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 428 行定义.

函数调用图:



4.94.1.20 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_lks32mc09x.mcpwm.c](#) 第 242 行定义.

函数调用图:



4.94.1.21 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

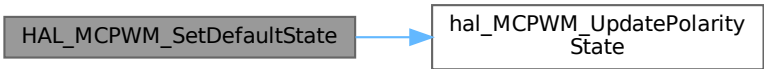
参数

<i>mcpwm_x</i>	操作的模块编号
<i>mcpwm_default_output_state_x</i>	默认输出状态

bit0 通道交换使能 0 交换 1 交换 bit1 上管有效电平 0 上管低有效 1 上管高有效 bit2 下管有效电平 0 下管低有效 1 下管高有效 bit3 默认上管电平 0 关机后上管关闭 1 关机后上管打开 bit4 默认下管电平 0 关机后下管关闭 1 关机后下管打开 bit5-bit7 输出状态 HAL_MCPWM_OUTPUT_STATE_x

在文件 [hal_lks32mc09x.mcpwm.c](#) 第 362 行定义.

函数调用图:



4.94.1.22 HAL_MCPWM_SetDrivePolarity()

```
void HAL_MCPWM_SetDrivePolarity (
    HAL_MCPWM_x mcpwm_x,
    HAL_MCPWM_POLARITY_x mcpwm_polarity_x )
```

设置pwm极性

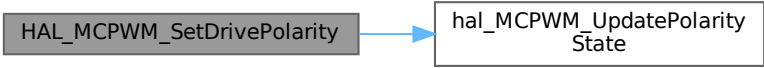
参数

<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

bit0 通道交换使能 0 交换 1 交换 bit1 上管有效电平 0 上管低有效 1 上管高有效 bit2 下管有效电平 0 下管低有效 1 下管高有效 bit3 默认上管电平 0 关机后上管关闭 1 关机后上管打开 bit4 默认下管电平 0 关机后下管关闭 1 关机后下管打开 bit5-bit7 输出状态 HAL_MCPWM_OUTPUT_STATE_x

在文件 [hal_iks32mc09x.mcpwm.c](#) 第 338 行定义.

函数调用图:



4.94.1.23 HAL_MCPWM_SetFailSignal()

```
void HAL_MCPWM_SetFailSignal (
    HAL_MCPWM_x mcpwm_x,
    HAL_MCPWM_FAIL_x fail_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_x</i>	FAIL信号来源

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 416 行定义.

4.94.1.24 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_lks32mc09x_mcpwm.c](#) 第 207 行定义.

函数调用图:

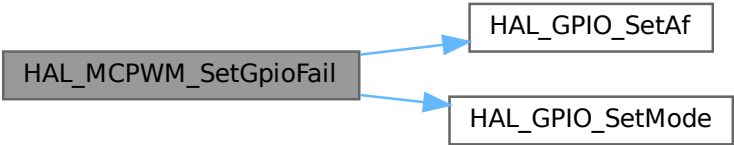


4.94.1.25 HAL_MCPWM_SetGpioFail()

```
void HAL_MCPWM_SetGpioFail (
    HAL_MCPWM_X mcpwm_x,
    HAL_GPIO_Px px )
```

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 404 行定义.

函数调用图:



4.94.1.26 HAL_MCPWM_SetGpioOut()

```
void HAL_MCPWM_SetGpioOut (
    HAL_MCPWM_TypeDef* mcpwm_x,
    GPIO_TypeDef* px )
```

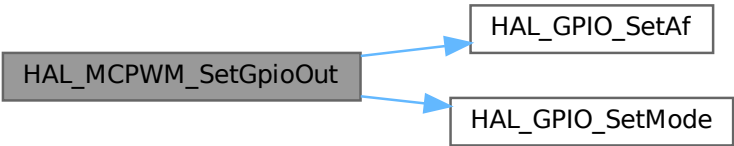
设置用于输出的gpio

参数

<i>mcpwm_x</i>	MCPWM模块编号
<i>px</i>	使用的GPIO口

在文件 `hal_lks32mc09x_mcpwm.c` 第 398 行定义.

函数调用图:



4.94.1.27 HAL_MCPWM_SetOutputDisable()

```
void HAL_MCPWM_SetOutputDisable (
    HAL_MCPWM_TypeDef* mcpwm_x )
```

关闭输出

参数

<i>mcpwm</i> ↔ <i>_x</i>	操作的模块编号

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 257 行定义.

4.94.1.28 HAL_MCPWM_SetOutputEnable()

```
void HAL_MCPWM_SetOutputEnable (
    HAL_MCPWM_x mcpwm_x )
```

打开输出

参数

<i>mcpwm</i> ↔ <i>_x</i>	操作的模块编号

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 267 行定义.

4.94.1.29 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

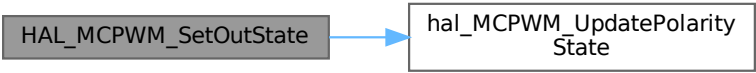
参数

<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</i> ↔ <i>state_x</i>	输出状态

bit0 通道交换使能 0 交换 1 交换 bit1 上管有效电平 0 上管低有效 1 上管高有效 bit2 下管有效电平 0 下管低有效 1 下管高有效 bit3 默认上管电平 0 关机后上管关闭 1 关机后上管打开 bit4 默认下管电平 0 关机后下管关闭 1 关机后下管打开 bit5-bit7 输出状态 HAL_MCPWM_OUTPUT_STATE_x

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 319 行定义.

函数调用图:



4.94.1.30 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_x</i> ↔ <i>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_lks32mc09x.mcpwm.c](#) 第 300 行定义.

这是这个函数的调用关系图:

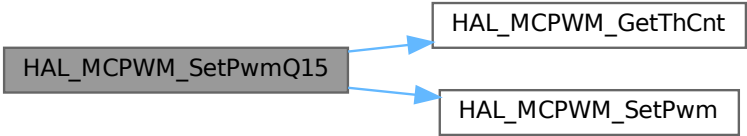


4.94.1.31 HAL.MCPWM.SetPwmQ15()

```
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_lks32mc09x_mcpwm.c](#) 第 306 行定义.

函数调用图:



4.94.1.32 HAL_MCPWM_StartCounting()

```
void HALMCPWM_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_lks32mc09x_mcpwm.c](#) 第 277 行定义.

4.94.1.33 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_lks32mc09x_mcpwm.c](#) 第 287 行定义.

4.94.1.34 hal_MCPWM_UpdatePolarityState()

```
void halMCPWM_UpdatePolarityState (
    HAL_MCPWM_x mcpwm_x,
    HAL_MCPWM_PORT_x mcpwm_port_x )
```

内部使用,输出端口的状态

参数

<i>mcpwm_x</i>	操作的模块编号
<i>mcpwm_x</i> <i>port_x</i>	第x个pwm输出端口

< 上下管对称pwm

< 关闭输出

< 下管恒开

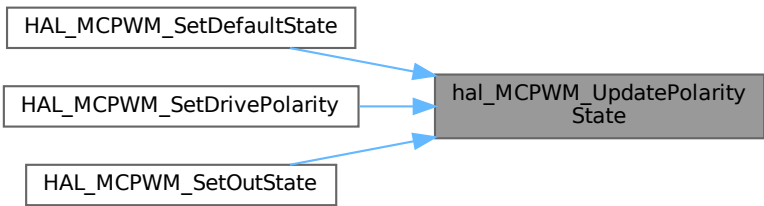
< 上管恒开

< 下管pwm

< 上管pwm

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 23 行定义.

这是这个函数的调用关系图:



4.94.2 变量说明

4.94.2.1 mcpwm_polarity_state

```
uint16_t mcpwm_polarity_state[1][6] = {{0, 0, 0, 0, 0, 0}} [static]
```

bit0 通道交换使能 0 交换 1 交换 bit1 上管有效电平 0 上管低有效 1 上管高有效 bit2 下管有效电平 0 下管低有效 1 下管高有效 bit3 默认上管电平 0 关机后上管关闭 1 关机后上管打开 bit4 默认下管电平 0 关机后下管关闭 1 关机后下管打开 bit5-bit7 输出状态 HAL_MCPWM_OUTPUT_STATE_x

在文件 [hal_lks32mc09x_mcpwm.c](#) 第 16 行定义.

4.95 hal_lks32mc09x_mcpwm.c

[浏览该文件的文档.](#)

```

00001 #include "../include/hal_lksmcu_mcpwm.h"
00002 #include "../include/hal_lksmcu_gpio.h"
00003 #include "lks32mc09x.h"
00004 #include "lks32mc09x_user_manual.h"
00005 #include "../include/hal_lksmcu_reg.h"
00006 #include "../include/hal_lks32mc09x_reg.h"
00007
00016 static uint16_t mcpwm_polarity_state[1][6] = {{0, 0, 0, 0, 0, 0}};
00017
00023 void hal_MCPWM_UpdatePolarityState(HAL_MCPWM_x mcpwm_x, HAL_MCPWM_PORT_x mcpwm_port_x)
00024 {
00025     (void)mcpwm_x;
00026     (void)mcpwm_port_x;
00027     uint8_t tmp[6] = {0, 0, 0, 0, 0, 0};
00028     uint8_t tmp1[6] = {0, 0, 0, 0, 0, 0};
00029     int32_t i;
00030     uint32_t ch_def = 0;
00031     uint32_t io01 = 0;
00032     uint32_t io23 = 0;
00033     uint32_t io45 = 0;
00034     HAL_REG_IF_WRITE(io01, BIT6, mcpwm_polarity_state[0][0] & BIT0); // 打开通道交换
00035     HAL_REG_IF_WRITE(io01, BIT14, mcpwm_polarity_state[0][1] & BIT0); // 打开通道交换
00036     HAL_REG_IF_WRITE(io23, BIT6, mcpwm_polarity_state[0][2] & BIT0); // 打开通道交换
00037     HAL_REG_IF_WRITE(io23, BIT14, mcpwm_polarity_state[0][3] & BIT0); // 打开通道交换
00038     HAL_REG_IF_WRITE(io45, BIT6, mcpwm_polarity_state[0][4] & BIT0); // 打开通道交换
00039     HAL_REG_IF_WRITE(io45, BIT14, mcpwm_polarity_state[0][5] & BIT0); // 打开通道交换
00040
00041     HAL_REG_IF_WRITE(io01, BIT1, !(mcpwm_polarity_state[0][0] & BIT1)); // 上管高有效
00042     HAL_REG_IF_WRITE(io01, BIT9, !(mcpwm_polarity_state[0][1] & BIT1)); // 上管高有效
00043     HAL_REG_IF_WRITE(io23, BIT1, !(mcpwm_polarity_state[0][2] & BIT1)); // 上管高有效
00044     HAL_REG_IF_WRITE(io23, BIT9, !(mcpwm_polarity_state[0][3] & BIT1)); // 上管高有效
00045     HAL_REG_IF_WRITE(io45, BIT1, !(mcpwm_polarity_state[0][4] & BIT1)); // 上管高有效
00046     HAL_REG_IF_WRITE(io45, BIT9, !(mcpwm_polarity_state[0][5] & BIT1)); // 上管高有效
00047
00048     HAL_REG_IF_WRITE(io01, BIT0, !(mcpwm_polarity_state[0][0] & BIT2)); // 下管高有效
00049     HAL_REG_IF_WRITE(io01, BIT8, !(mcpwm_polarity_state[0][1] & BIT2)); // 下管高有效
00050     HAL_REG_IF_WRITE(io23, BIT0, !(mcpwm_polarity_state[0][2] & BIT2)); // 下管高有效
00051     HAL_REG_IF_WRITE(io23, BIT8, !(mcpwm_polarity_state[0][3] & BIT2)); // 下管高有效
00052     HAL_REG_IF_WRITE(io45, BIT0, !(mcpwm_polarity_state[0][4] & BIT2)); // 下管高有效
00053     HAL_REG_IF_WRITE(io45, BIT8, !(mcpwm_polarity_state[0][5] & BIT2)); // 下管高有效
00054
00055     for (i = 0; i < 6; i++)
00056     {
00057         if (mcpwm_polarity_state[0][i] & BIT4)
00058         {
00059             // 关机后上管打开
00060             if (mcpwm_polarity_state[0][i] & BIT0) // gpio交换使能
00061             {
00062                 HAL_REG_IF_WRITE(tmp[i], BIT0, (mcpwm_polarity_state[0][i] & BIT1)); // CHO N 通道默认值
00063                 HAL_REG_IF_WRITE(tmp[i], BIT1, !(mcpwm_polarity_state[0][i] & BIT2)); // CHO P 通道默认值
00064             }
00065             else
00066             {
00067                 HAL_REG_IF_WRITE(tmp[i], BIT0, !(mcpwm_polarity_state[0][i] & BIT2)); // CHO N 通道默认值
00068                 HAL_REG_IF_WRITE(tmp[i], BIT1, (mcpwm_polarity_state[0][i] & BIT1)); // CHO P 通道默认值
00069             }
00070         }
00071         else if (mcpwm_polarity_state[0][i] & BIT3)
00072         {
00073             // 关机后下管打开
00074             if (mcpwm_polarity_state[0][i] & BIT0) // gpio交换使能
00075             {
00076                 HAL_REG_IF_WRITE(tmp[i], BIT0, !(mcpwm_polarity_state[0][i] & BIT1)); // CHO N 通道默认值
00077                 HAL_REG_IF_WRITE(tmp[i], BIT1, (mcpwm_polarity_state[0][i] & BIT2)); // CHO P 通道默认值
00078             }
00079             else
00080             {
00081                 HAL_REG_IF_WRITE(tmp[i], BIT0, (mcpwm_polarity_state[0][i] & BIT2)); // CHO N 通道默认值
00082                 HAL_REG_IF_WRITE(tmp[i], BIT1, !(mcpwm_polarity_state[0][i] & BIT1)); // CHO P 通道默认值
00083             }
00084         }
00085         else
00086         {
00087             // 关机后gpio悬空
00088             // 关机后下管打开
00089             if (mcpwm_polarity_state[0][i] & BIT0) // gpio交换使能
00090             {
00091                 HAL_REG_IF_WRITE(tmp[i], BIT0, !(mcpwm_polarity_state[0][i] & BIT1)); // CHO N 通道默认值
00092                 HAL_REG_IF_WRITE(tmp[i], BIT1, !(mcpwm_polarity_state[0][i] & BIT2)); // CHO P 通道默认值
00093             }
00094             else
00095             {

```

```

00096         HAL_REG_IF_WRITE(tmp[i], BIT0, !(mcpwm.polarity_state[0][i] & BIT2)); // CHO N 通道默认值
00097         HAL_REG_IF_WRITE(tmp[i], BIT1, !(mcpwm.polarity_state[0][i] & BIT1)); // CHO P 通道默认值
00098     }
00099 }
00100 }
00101 HAL_REG_IF_WRITE(ch_def, BIT0, tmp[0] & BIT0); // 默认输出状态
00102 HAL_REG_IF_WRITE(ch_def, BIT1, tmp[0] & BIT1); // 默认输出状态
00103 HAL_REG_IF_WRITE(ch_def, BIT2, tmp[1] & BIT0); // 默认输出状态
00104 HAL_REG_IF_WRITE(ch_def, BIT3, tmp[1] & BIT1); // 默认输出状态
00105 HAL_REG_IF_WRITE(ch_def, BIT4, tmp[2] & BIT0); // 默认输出状态
00106 HAL_REG_IF_WRITE(ch_def, BIT5, tmp[2] & BIT1); // 默认输出状态
00107 HAL_REG_IF_WRITE(ch_def, BIT6, tmp[3] & BIT0); // 默认输出状态
00108 HAL_REG_IF_WRITE(ch_def, BIT7, tmp[3] & BIT1); // 默认输出状态
00109 HAL_REG_IF_WRITE(ch_def, BIT8, tmp[4] & BIT0); // 默认输出状态
00110 HAL_REG_IF_WRITE(ch_def, BIT9, tmp[4] & BIT1); // 默认输出状态
00111 HAL_REG_IF_WRITE(ch_def, BIT10, tmp[5] & BIT0); // 默认输出状态
00112 HAL_REG_IF_WRITE(ch_def, BIT11, tmp[5] & BIT1); // 默认输出状态
00113
00114 for (i = 0; i < 6; i++)
00115 {
00116     HAL_MCPWM_OUTPUT_STATE_x out_state;
00117     out_state = (HAL_MCPWM_OUTPUT_STATE_x)(mcpwm.polarity_state[0][i] >> 5);
00118     switch (out_state)
00119     {
00120         case HAL_MCPWM_OUTPUT_STATE_PWM:
00121             tmp1[i] = 0;
00122             break;
00123         case HAL_MCPWM_OUTPUT_STATE_OFF:
00124             tmp1[i] = BIT2 | BIT3;
00125             break;
00126         case HAL_MCPWM_OUTPUT_STATE_LOW:
00127             tmp1[i] = BIT2 | BIT3 | BIT4;
00128             break;
00129         case HAL_MCPWM_OUTPUT_STATE_HIGH:
00130             tmp1[i] = BIT2 | BIT3 | BIT5;
00131             break;
00132         case HAL_MCPWM_OUTPUT_STATE_LOW_PWM:
00133             tmp1[i] = BIT3;
00134             break;
00135         case HAL_MCPWM_OUTPUT_STATE_HIGH_PWM:
00136             tmp1[i] = BIT2;
00137             break;
00138     }
00139 }
00140 HAL_REG_WRITE_BIT(io01, 0x003c, tmp1[0]);
00141 HAL_REG_WRITE_BIT(io01, 0x3c00, tmp1[1] << 8);
00142 HAL_REG_WRITE_BIT(io23, 0x003c, tmp1[2]);
00143 HAL_REG_WRITE_BIT(io23, 0x3c00, tmp1[3] << 8);
00144 HAL_REG_WRITE_BIT(io45, 0x003c, tmp1[4]);
00145 HAL_REG_WRITE_BIT(io45, 0x3c00, tmp1[5] << 8);
00146
00147 HAL_REG_WRITE(HAL_MCPWM[mcpwm_x]->PRT, 0xdead);
00148 HAL_REG_WRITE_BIT(HAL_MCPWM[mcpwm_x]->CH_DEF, 0xff00, ch_def);
00149 HAL_REG_WRITE_BIT(HAL_MCPWM[mcpwm_x]->IO01, 0x7f7f, io01);
00150 HAL_REG_WRITE_BIT(HAL_MCPWM[mcpwm_x]->IO23, 0x7f7f, io23);
00151 HAL_REG_WRITE(HAL_MCPWM[mcpwm_x]->PRT, 0);
00152 }
00153
00154 void HAL_MCPWM_Reset(HAL_MCPWM_x mcpwm_x)
00155 {
00156     (void)mcpwm_x;
00157     // 系统寄存器解锁
00158     HAL_REG_SYS_UNLOCK();
00159     HAL_REG_SET(SYS_SFT_RST, SYS_SFT_RST_MCPWM0_SFT_RST_BIT);
00160     HAL_REG_RESET(SYS_SFT_RST, SYS_SFT_RST_MCPWM0_SFT_RST_BIT);
00161     // 系统寄存器上锁
00162     HAL_REG_SYS_LOCK();
00163 }
00164
00165 void HAL_MCPWM_Enable(HAL_MCPWM_x mcpwm_x)
00166 {
00167     // 系统寄存器解锁
00168     HAL_REG_SYS_UNLOCK();
00169     // SYS_CLK_FEN[4] MCPWM 模块整体时钟控制信号
00170     HAL_REG_SET(SYS_CLK_FEN, SYS_CLK_FEN_MCPWM0_CLK_EN_BIT);
00171     // 系统寄存器上锁
00172     HAL_REG_SYS_LOCK();
00173
00174     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x]->PRT, 0xdead);
00175     HAL_REG_SET(HAL_MCPWM[mcpwm_x]->TCLK, BIT2);
00176     // HAL_REG_WRITE_BIT(HAL_MCPWM[mcpwm_x]->AUEN, 0xffff, 0x43f);
00177     HAL_REG_SET(HAL_MCPWM[mcpwm_x]->SDCFG, BIT4);
00178     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x]->PRT, 0);
00179 }
00180
00181 void HAL_MCPWM_Disable(HAL_MCPWM_x mcpwm_x)
00182 {
00183     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x]->PRT, 0xdead);
00184     HAL_REG_RESET(HAL_MCPWM[mcpwm_x]->TCLK, BIT2);
00185 }

```

```

00195     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0);
00196     // 系统寄存器解锁
00197     HAL_REG_SYS_UNLOCK();
00198     // SYS_CLK_FEN[4] MCPWM 模块整体时钟控制信号
00199     HAL_REG_RESET(SYS_CLK_FEN, SYS_CLK_FEN_MCPWM0_CLK_EN_BIT);
00200     // 系统寄存器上锁
00201     HAL_REG_SYS_LOCK();
00202 }
00207 void HAL_MCPWM_SetFreq(HAL_MCPWM_x mcpwm_x, uint32_t freq)
00208 {
00209     uint32_t mclk = HAL_CLK_GetMcuFreq();
00210     uint32_t th = ((mclk / freq) - 1) / 2;
00211     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0xdead);
00212     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> TH0, th);
00213     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0);
00214     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> UPDATE, BIT12);
00215 }
00221 uint32_t HAL_MCPWM_GetFreq(HAL_MCPWM_x mcpwm_x)
00222 {
00223     uint32_t mclk = HAL_CLK_GetMcuFreq();
00224     uint16_t th = HAL_MCPWM[mcpwm_x] -> TH0;
00225     uint32_t freq = mclk / (th * 2 + 1);
00226     return freq;
00227 }
00232 uint32_t HAL_MCPWM_GetThCnt(HAL_MCPWM_x mcpwm_x)
00233 {
00234     return HAL_MCPWM[mcpwm_x] -> TH0;
00235 }
00242 void HAL_MCPWM_SetDead(HAL_MCPWM_x mcpwm_x, uint32_t pdead_ns, uint32_t ndead_ns)
00243 {
00244     uint32_t mclk = HAL_CLK_GetMcuFreq();
00245     uint16_t pdead = (int64_t)pdead_ns * mclk / 1000000000;
00246     uint16_t ndead = (int64_t)ndead_ns * mclk / 1000000000;
00247     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0xdead);
00248     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> DTH00, pdead);
00249     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> DTH01, ndead);
00250     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0);
00251 }
00252
00257 void HAL_MCPWM_SetOutputDisable(HAL_MCPWM_x mcpwm_x)
00258 {
00259     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0xdead);
00260     HAL_REG_RESET(HAL_MCPWM[mcpwm_x] -> CH_FAIL, BIT6);
00261     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0);
00262 }
00267 void HAL_MCPWM_SetOutputEnable(HAL_MCPWM_x mcpwm_x)
00268 {
00269     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0xdead);
00270     HAL_REG_SET(HAL_MCPWM[mcpwm_x] -> CH_FAIL, BIT6);
00271     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0);
00272 }
00277 void HAL_MCPWM_StartCounting(HAL_MCPWM_x mcpwm_x)
00278 {
00279     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0xdead);
00280     HAL_REG_SET(HAL_MCPWM[mcpwm_x] -> TCLK, BIT6);
00281     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0);
00282 }
00287 void HAL_MCPWM_StopCounting(HAL_MCPWM_x mcpwm_x)
00288 {
00289     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0xdead);
00290     HAL_REG_RESET(HAL_MCPWM[mcpwm_x] -> TCLK, BIT6);
00291     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0);
00292 }
00300 void HAL_MCPWM_SetPwm(HAL_MCPWM_x mcpwm_x, HAL_MCPWM_PORT_x mcpwm_port_x, int32_t t0, int32_t t1)
00301 {
00302     // 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
00303     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> TH[mcpwm_port_x * 2], t0);
00304     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> TH[mcpwm_port_x * 2 + 1], t1);
00305 }
00306 void HAL_MCPWM_SetPwmQ15(HAL_MCPWM_x mcpwm_x, HAL_MCPWM_PORT_x mcpwm_port_x, int16_t t0, int16_t t1)
00307 {
00308     uint32_t th = HAL_MCPWM_GetThCnt(mcpwm_x);
00309     int32_t cnt0 = (t0 * th) >> 15;
00310     int32_t cnt1 = (t1 * th) >> 15;
00311     HAL_MCPWM_SetPwm(mcpwm_x, mcpwm_port_x, cnt0, cnt1);
00312 }
00319 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)
00320 {
00321     HAL_REG_WRITE_BIT(mcpwm_polarity_state[mcpwm_x][mcpwm_port_x], BIT5 | BIT6 | BIT7,
mcpwm_output_state_x << 5);
00322     HAL_MCPWM_UpdatePolarityState(mcpwm_x, mcpwm_port_x);
00323 }
00332
00338 void HAL_MCPWM_SetDrivePolarity(HAL_MCPWM_x mcpwm_x, HAL_MCPWM_POLARITY_x mcpwm_polarity_x)
00339 {
00340     HAL_REG_WRITE_BIT(mcpwm_polarity_state[mcpwm_x][HAL_MCPWM_PORT_0], BIT1 | BIT2, mcpwm_polarity_x <<

```

```

1);
00349 HAL_REG_WRITE_BIT(mcpwm_polarity_state[mcpwm_x][HAL_MCPWM_PORT_1], BIT1 | BIT2, mcpwm_polarity_x <<
1);
00350 HAL_REG_WRITE_BIT(mcpwm_polarity_state[mcpwm_x][HAL_MCPWM_PORT_2], BIT1 | BIT2, mcpwm_polarity_x <<
1);
00351 HAL_REG_WRITE_BIT(mcpwm_polarity_state[mcpwm_x][HAL_MCPWM_PORT_3], BIT1 | BIT2, mcpwm_polarity_x <<
1);
00352 HAL_MCPWM_UpdatePolarityState(mcpwm_x, HAL_MCPWM_PORT_0);
00353 HAL_MCPWM_UpdatePolarityState(mcpwm_x, HAL_MCPWM_PORT_1);
00354 HAL_MCPWM_UpdatePolarityState(mcpwm_x, HAL_MCPWM_PORT_2);
00355 HAL_MCPWM_UpdatePolarityState(mcpwm_x, HAL_MCPWM_PORT_3);
00356 }
00362 void HAL_MCPWM_SetDefaultState(HAL_MCPWM_x mcpwm_x, HAL_MCPWM_DEFAULT_OUTPUT_STATE_x
mcpwm_default_output_state_x)
00363 {
00364     uint32_t val = 0;
00365     switch (mcpwm_default_output_state_x)
00366     {
00367         case HAL_MCPWM_DEFAULT_OUTPUT_STATE_OFF:
00368             break;
00369         case HAL_MCPWM_DEFAULT_OUTPUT_STATE_LOW:
00370             HAL_REG_SET(val, BIT4);
00371             break;
00372         case HAL_MCPWM_DEFAULT_OUTPUT_STATE_HIGH:
00373             HAL_REG_SET(val, BIT3);
00374             break;
00375     }
00384 HAL_REG_WRITE_BIT(mcpwm_polarity_state[mcpwm_x][HAL_MCPWM_PORT_0], BIT3 | BIT4, val);
00385 HAL_REG_WRITE_BIT(mcpwm_polarity_state[mcpwm_x][HAL_MCPWM_PORT_1], BIT3 | BIT4, val);
00386 HAL_REG_WRITE_BIT(mcpwm_polarity_state[mcpwm_x][HAL_MCPWM_PORT_2], BIT3 | BIT4, val);
00387 HAL_REG_WRITE_BIT(mcpwm_polarity_state[mcpwm_x][HAL_MCPWM_PORT_3], BIT3 | BIT4, val);
00388 HAL_MCPWM_UpdatePolarityState(mcpwm_x, HAL_MCPWM_PORT_0);
00389 HAL_MCPWM_UpdatePolarityState(mcpwm_x, HAL_MCPWM_PORT_1);
00390 HAL_MCPWM_UpdatePolarityState(mcpwm_x, HAL_MCPWM_PORT_2);
00391 HAL_MCPWM_UpdatePolarityState(mcpwm_x, HAL_MCPWM_PORT_3);
00392 }
00398 void HAL_MCPWM_SetGpioOut(HAL_MCPWM_x mcpwm_x, HAL_GPIO_Px px)
00399 {
00400     (void)mcpwm_x;
00401     HAL_GPIO_SetAf(px, HAL_GPIO_AF_MCPWM);
00402     HAL_GPIO_SetMode(px, HAL_GPIO_MODE_OUT);
00403 }
00404 void HAL_MCPWM_SetGpioFail(HAL_MCPWM_x mcpwm_x, HAL_GPIO_Px px) // 设置用于故障输入的gpio
00405 {
00406     (void)mcpwm_x;
00407     HAL_GPIO_SetAf(px, HAL_GPIO_AF_MCPWM);
00408     HAL_GPIO_SetMode(px, HAL_GPIO_MODE_IN);
00409 }
00410
00416 void HAL_MCPWM_SetFailSignal(HAL_MCPWM_x mcpwm_x, HAL_MCPWM_FAIL_x fail_x)
00417 {
00418     (void)mcpwm_x;
00419     (void)fail_x;
00420 }
00421 // 设置pwm输出ADC采样时间
00422 void HAL_MCPWM_SetAdcTrig(HAL_MCPWM_x mcpwm_x, int32_t t0, int32_t t1)
00423 {
00424     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x]->TMR0, t0);
00425     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x]->TMR1, t1);
00426 }
00427 // 设置pwm输出ADC采样时间q15格式输入
00428 void HAL_MCPWM_SetAdcTrigQ15(HAL_MCPWM_x mcpwm_x, int32_t t0, int32_t t1)
00429 {
00430     uint32_t th = HAL_MCPWM_GetThCnt(mcpwm_x);
00431     int16_t tmr0 = ((int32_t)t0 * th) >> 15;
00432     int16_t tmr1 = ((int32_t)t1 * th) >> 15;
00433     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x]->PRT, 0xdead);
00434     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x]->TMR0, tmr0);
00435     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x]->TMR1, tmr1);
00436 }
00437
00438 uint8_t HAL_MCPWM_IrqOverflow(HAL_MCPWM_x mcpwm_x)
00439 {
00440     if (HAL_MCPWM[mcpwm_x]->IE0 & HAL_MCPWM[mcpwm_x]->IF0 & BIT0)
00441     {
00442         return 1;
00443     }
00444     else
00445     {
00446         return 0;
00447     }
00448 }
00449 void HAL_MCPWM_IrqOverflowEnable(HAL_MCPWM_x mcpwm_x)
00450 {
00451     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x]->PRT, 0xdead);
00452     HAL_REG_SET(HAL_MCPWM[mcpwm_x]->IE0, BIT0);
00453     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x]->PRT, 0);

```

```

00454 }
00455 void HAL_MCPWM_IrqOverflowDisable(HAL_MCPWM_x mcpwm_x)
00456 {
00457     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0xdead);
00458     HAL_REG_RESET(HAL_MCPWM[mcpwm_x] -> IE0, BIT0);
00459     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0);
00460 }
00461 void HAL_MCPWM_IrqOverflowClear(HAL_MCPWM_x mcpwm_x)
00462 {
00463     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> IF0, BIT0);
00464 }
00465
00466 uint8_t HAL_MCPWM_IrqZeroCross(HAL_MCPWM_x mcpwm_x)
00467 {
00468     if (HAL_MCPWM[mcpwm_x] -> IE0 & HAL_MCPWM[mcpwm_x] -> IF0 & BIT1)
00469     {
00470         return 1;
00471     }
00472     else
00473     {
00474         return 0;
00475     }
00476 }
00477 void HAL_MCPWM_IrqZeroCrossEnable(HAL_MCPWM_x mcpwm_x)
00478 {
00479     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0xdead);
00480     HAL_REG_SET(HAL_MCPWM[mcpwm_x] -> IE0, BIT1);
00481     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0);
00482 }
00483 void HAL_MCPWM_IrqZeroCrossDisable(HAL_MCPWM_x mcpwm_x)
00484 {
00485     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0xdead);
00486     HAL_REG_RESET(HAL_MCPWM[mcpwm_x] -> IE0, BIT1);
00487     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0);
00488 }
00489 void HAL_MCPWM_IrqZeroCrossClear(HAL_MCPWM_x mcpwm_x)
00490 {
00491     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> IF0, BIT1);
00492 }
00493
00494 uint8_t HAL_MCPWM_IrqFail(HAL_MCPWM_x mcpwm_x)
00495 {
00496     if (HAL_MCPWM[mcpwm_x] -> EIE & HAL_MCPWM[mcpwm_x] -> EIF & (BIT4 | BIT5))
00497     {
00498         return 1;
00499     }
00500     else
00501     {
00502         return 0;
00503     }
00504 }
00505 void HAL_MCPWM_IrqFailEnable(HAL_MCPWM_x mcpwm_x)
00506 {
00507     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0xdead);
00508     HAL_REG_SET(HAL_MCPWM[mcpwm_x] -> IE0, BIT4 | BIT5);
00509     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0);
00510 }
00511 void HAL_MCPWM_IrqFailDisable(HAL_MCPWM_x mcpwm_x)
00512 {
00513     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0xdead);
00514     HAL_REG_RESET(HAL_MCPWM[mcpwm_x] -> IE0, BIT4 | BIT5);
00515     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> PRT, 0);
00516 }
00517 void HAL_MCPWM_IrqFailClear(HAL_MCPWM_x mcpwm_x)
00518 {
00519     HAL_REG_WRITE(HAL_MCPWM[mcpwm_x] -> IF0, BIT4 | BIT5);
00520 }

```

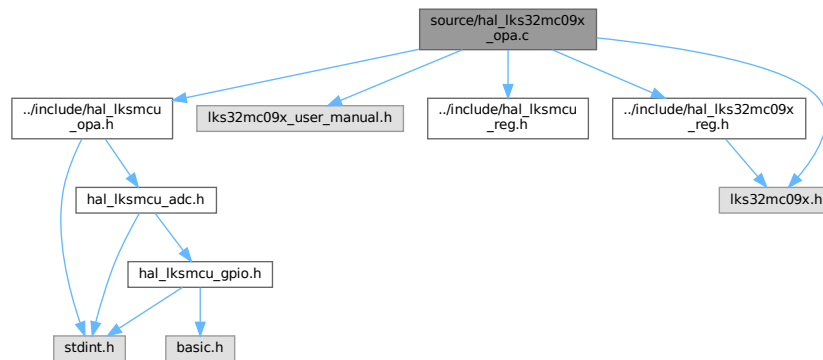
4.96 source/hal_lks32mc09x_opa.c 文件参考

```

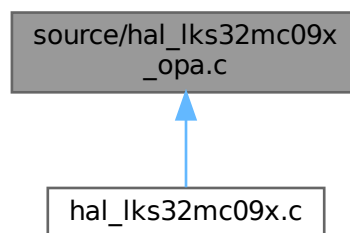
#include "../include/hal_lksmcu_opa.h"
#include "lks32mc09x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc09x_reg.h"
#include "lks32mc09x.h"

```


hal_iks32mc09x_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

4.96.1 函数说明

4.96.1.1 HAL_OPA_Disable()

```
void HAL_OPA_Disable (
    HAL_OPA_x opa_x )
```

在文件 [hal_lks32mc09x_opa.c](#) 第 38 行定义。

4.96.1.2 HAL_OPA_Enable()

```
void HAL_OPA_Enable (
    HAL_OPA_x opa_x )
```

在文件 [hal_lks32mc09x_opa.c](#) 第 13 行定义。

4.96.1.3 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

使能 OPA_x 输出信号送至IO口,或使能 $OPA_x.OUT$ 输出信号送至CMP。0000:不输出到IO,输出 $OPA0.OUT$ 正端到 $OPA.OUT\_CMP$; 0001:不输出到IO,输出 $OPA1.OUT$ 正端到 $OPA.OUT\_CMP$; 0010:不输出到IO,输出 $OPA2.OUT$ 正端到 $OPA.OUT\_CMP$; 0011:不输出到IO,输出 $OPA3.OUT$ 正端到 $OPA.OUT\_CMP$; 0100:不输出到IO,输出 $OPA0.OUT$ 负端到 $OPA.OUT\_CMP$; 0101:不输出到IO,输出 $OPA1.OUT$ 负端到 $OPA.OUT\_CMP$; 0110:不输出到IO,输出 $OPA2.OUT$ 负端到 $OPA.OUT\_CMP$; 0111:不输出到IO,输出 $OPA3.OUT$ 负端到 $OPA.OUT\_CMP$; 1000:输出 $OPA0.OUT$ 正端到IO P2.7口,同时输出 $OPA0.OUT$ 正端到 $OPA.OUT\_CMP$; 1001:输出 $OPA1.OUT$ 正端到IO P2.7口,同时输出 $OPA1.OUT$ 正端到 $OPA.OUT\_CMP$; 1010:输出 $OPA2.OUT$ 正端到IO P2.7口,同时输出 $OPA2.OUT$ 正端到 $OPA.OUT\_CMP$; 1011:输出 $OPA3.OUT$ 正端到IO P2.7口,同时输出 $OPA3.OUT$ 正端到 $OPA.OUT\_CMP$; 1100:输出 $OPA0.OUT$ 负端到IO P2.7口,同时输出 $OPA0.OUT$ 负端到 $OPA.OUT\_CMP$; 1101:输出 $OPA1.OUT$ 负端到IO P2.7口,同时输出 $OPA1.OUT$ 负端到 $OPA.OUT\_CMP$; 1110:输出 $OPA2.OUT$ 负端到IO P2.7口,同时输出 $OPA2.OUT$ 负端到 $OPA.OUT\_CMP$; 1111:输出 $OPA3.OUT$ 负端到IO P2.7口,同时输出 $OPA3.OUT$ 负端到 $OPA.OUT\_CMP$; 即 $OPAOUT\_EN[3]$ 控制是否将 OPA_x 输出到IO, $OPAOUT\_EN[2:0]$ 选择 OPA_x 输出到 $OPA.OUT\_CMP$

在文件 [hal_lks32mc09x_opa.c](#) 第 137 行定义。

4.96.1.4 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

使能_OPAx输出信号送至IO口,或使能OPA_x.OUT输出信号送至CMP。0000:不输出到IO,输出OPA0_OUT正端到OPA_OUT_CMP; 0001:不输出到IO,输出OPA1_OUT正端到OPA_OUT_CMP; 0010:不输出到IO,输出OPA2_OUT正端到OPA_OUT_CMP; 0011:不输出到IO,输出OPA3_OUT正端到OPA_OUT_CMP; 0100:不输出到IO,输出OPA0_OUT负端到OPA_OUT_CMP; 0101:不输出到IO,输出OPA1_OUT负端到OPA_OUT_CMP; 0110:不输出到IO,输出OPA2_OUT负端到OPA_OUT_CMP; 0111:不输出到IO,输出OPA3_OUT负端到OPA_OUT_CMP; 1000:输出OPA0_OUT正端到IO P2.7口,同时输出OPA0_OUT正端到OPA_OUT_CMP; 1001:输出OPA1_OUT正端到IO P2.7口,同时输出OPA1_OUT正端到OPA_OUT_CMP; 1010:输出OPA2_OUT正端到IO P2.7口,同时输出OPA2_OUT正端到OPA_OUT_CMP; 1011:输出OPA3_OUT正端到IO P2.7口,同时输出OPA3_OUT正端到OPA_OUT_CMP; 1100:输出OPA0_OUT负端到IO P2.7口,同时输出OPA0_OUT负端到OPA_OUT_CMP; 1101:输出OPA1_OUT负端到IO P2.7口,同时输出OPA1_OUT负端到OPA_OUT_CMP; 1110:输出OPA2_OUT负端到IO P2.7口,同时输出OPA2_OUT负端到OPA_OUT_CMP; 1111:输出OPA3_OUT负端到IO P2.7口,同时输出OPA3_OUT负端到OPA_OUT_CMP; 即OPAOUT_EN[3]控制是否将OPA_x输出到IO,OPAOUT_EN[2:0]选择OPA_x输出到OPA_OUT_CMP

在文件 [hal_lks32mc09x_opa.c](#) 第 92 行定义.

4.96.1.5 HAL_OPA_Reset()

```
void HAL_OPA_Reset (
    HAL_OPA_x opa_x )
```

在文件 [hal_lks32mc09x_opa.c](#) 第 8 行定义.

4.96.1.6 HAL_OPA_SetGain()

```
void HAL_OPA_SetGain (
    HAL_OPA_x opa_x,
    HAL_OPA_GAIN_x gain_x )
```

在文件 [hal_lks32mc09x_opa.c](#) 第 63 行定义.

4.96.1.7 HAL_OPA_SetGpioIn()

```
void HAL_OPA_SetGpioIn (
    HAL_OPA_x opa_x,
    HAL_GPIO_Px px )
```

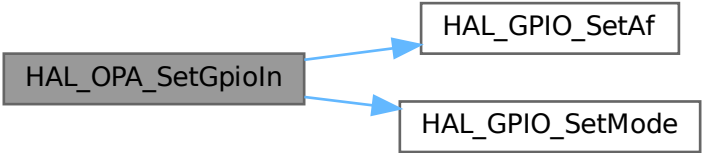
OPA设置IO输入

参数

<i>opa</i> ↵ <i>_x</i>	运放序号
<i>px</i>	IO序号

在文件 [hal_lks32mc09x_opa.c](#) 第 169 行定义.

函数调用图:



4.96.1.8 HAL_OPA_SetGpioOut()

```
void HAL_OPA_SetGpioOut (
    HAL_OPA_x opa_x,
    HAL_GPIO_Px px )
```

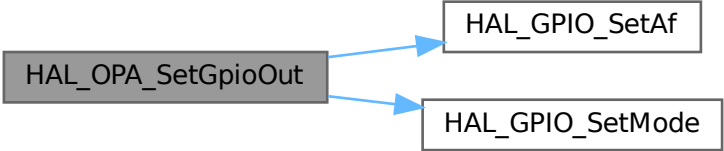
OPA设置IO输出

参数

<i>opa_x</i>	运放序号
<i>px</i>	IO序号

在文件 [hal_lks32mc09x_opa.c](#) 第 180 行定义.

函数调用图:



4.97 hal_lks32mc09x_opa.c

[浏览该文件的文档.](#)

```

00001 #include "../include/hal_lksmcu_opa.h"
00002 #include "lks32mc09x_user_manual.h"
00003 #include "../include/hal_lksmcu_reg.h"
00004 #include "../include/hal_lks32mc09x_reg.h"
00005 #include "lks32mc09x.h"
00006
00007 // 模块复位
00008 void HAL_OPA_Reset(HAL_OPA_x opa_x)
00009 {
00010     (void)opa_x;
00011 }
00012 // 模块打开
00013 void HAL_OPA_Enable(HAL_OPA_x opa_x)
00014 {
00015     // 系统寄存器解锁
00016     HAL_REG_SYS_UNLOCK();
00017     switch (opa_x)
00018     {
00019         case HAL_OPA_0:
00020             HAL_REG_SET(SYS_AFE_REG1, SYS_AFE_REG1_OPA0PDN_BIT);
00021             break;
00022         case HAL_OPA_1:
00023             HAL_REG_SET(SYS_AFE_REG1, SYS_AFE_REG1_OPA1PDN_BIT);
00024             break;
00025         case HAL_OPA_2:
00026             HAL_REG_SET(SYS_AFE_REG1, SYS_AFE_REG1_OPA2PDN_BIT);
00027             break;
00028         case HAL_OPA_3:
00029             HAL_REG_SET(SYS_AFE_REG1, SYS_AFE_REG1_OPA3PDN_BIT);
00030             break;
00031         default:
00032             break;
00033     }
00034     // 系统寄存器上锁
00035     HAL_REG_SYS_LOCK();
00036 }
00037 // 模块关闭
00038 void HAL_OPA_Disable(HAL_OPA_x opa_x)
00039 {
00040     // 系统寄存器解锁
00041     HAL_REG_SYS_UNLOCK();
00042     switch (opa_x)
00043     {
00044         case HAL_OPA_0:
00045             HAL_REG_RESET(SYS_AFE_REG1, SYS_AFE_REG1_OPA0PDN_BIT);
00046             break;
00047         case HAL_OPA_1:
00048             HAL_REG_RESET(SYS_AFE_REG1, SYS_AFE_REG1_OPA1PDN_BIT);
00049             break;
00050         case HAL_OPA_2:
00051             HAL_REG_RESET(SYS_AFE_REG1, SYS_AFE_REG1_OPA2PDN_BIT);
00052             break;
00053         case HAL_OPA_3:
00054             HAL_REG_RESET(SYS_AFE_REG1, SYS_AFE_REG1_OPA3PDN_BIT);
00055             break;
00056         default:
00057             break;
00058     }
00059     // 系统寄存器上锁
00060     HAL_REG_SYS_LOCK();
00061 }
00062 // 设置增益
00063 void HAL_OPA_SetGain(HAL_OPA_x opa_x, HAL_OPA_GAIN_x gain_x)
00064 {
00065     // 系统寄存器解锁
00066     HAL_REG_SYS_UNLOCK();
00067     switch (opa_x)
00068     {
00069         case HAL_OPA_0:
00070             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG0, SYS_AFE_REG0_RES_OPA0, gain_x);
00071             break;
00072         case HAL_OPA_1:
00073             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG0, SYS_AFE_REG0_RES_OPA1, gain_x);
00074             break;
00075         case HAL_OPA_2:
00076             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG0, SYS_AFE_REG0_RES_OPA2, gain_x);
00077             break;
00078         case HAL_OPA_3:
00079             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG0, SYS_AFE_REG0_RES_OPA3, gain_x);
00080             break;
00081         default:
00082             break;
00083     }
00084     // 系统寄存器上锁
00085     HAL_REG_SYS_LOCK();
00086 }
00087

```

```

00092 void HAL_OPA_Out2GpioEnable(HAL_OPA_x opa_x)
00093 {
00113     HAL_REG_SYS_UNLOCK();
00114     switch (opa_x)
00115     {
00116         case HAL_OPA_0:
00117             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG2, SYS_AFE_REG2_OPAOUT_EN, 8);
00118             break;
00119         case HAL_OPA_1:
00120             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG2, SYS_AFE_REG2_OPAOUT_EN, 9);
00121             break;
00122         case HAL_OPA_2:
00123             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG2, SYS_AFE_REG2_OPAOUT_EN, 10);
00124             break;
00125         case HAL_OPA_3:
00126             HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG2, SYS_AFE_REG2_OPAOUT_EN, 11);
00127             break;
00128         default:
00129             break;
00130     }
00131     HAL_REG_SYS_LOCK();
00132 }
00137 void HAL_OPA_Out2GpioDisable(HAL_OPA_x opa_x)
00138 {
00139     (void)opa_x;
00159     HAL_REG_SYS_UNLOCK();
00160     HAL_REG_WRITE_BIT_MASK_POS(SYS_AFE_REG2, SYS_AFE_REG2_OPAOUT_EN, 0);
00161     HAL_REG_SYS_LOCK();
00162 }
00163
00169 void HAL_OPA_SetGpioIn(HAL_OPA_x opa_x, HAL_GPIO_Px px)
00170 {
00171     (void)opa_x;
00172     HAL_GPIO_SetAf(px, HAL_GPIO_AF_OPA);
00173     HAL_GPIO_SetMode(px, HAL_GPIO_MODE_AFE);
00174 }
00180 void HAL_OPA_SetGpioOut(HAL_OPA_x opa_x, HAL_GPIO_Px px)
00181 {
00182     (void)opa_x;
00183     HAL_GPIO_SetAf(px, HAL_GPIO_AF_OPA);
00184     HAL_GPIO_SetMode(px, HAL_GPIO_MODE_AFE);
00185 }

```

4.98 source/hal_lks32mc09x_qep.c 文件参考

```

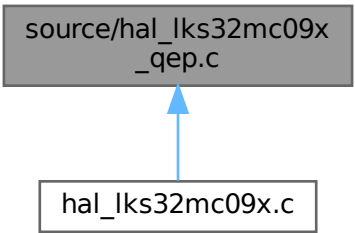
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lksmcu_qep.h"
#include "../include/hal_lksmcu_gpio.h"
#include "../hal_lks32mc09x.h"
#include "lks32mc09x_user_manual.h"
#include "../include/hal_lks32mc09x_reg.h"

```

hal_lks32mc09x_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_SetGpio](#) ([HAL_QEP_x](#) qepx, [HAL_GPIO_Px](#) px)
 设置QEP模块的引脚

4.98.1 函数说明

4.98.1.1 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_lks32mc09x_qep.c](#) 第 52 行定义.

4.98.1.2 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_lks32mc09x_qep.c](#) 第 32 行定义.

函数调用图:



4.98.1.3 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_lks32mc09x_qep.c](#) 第 105 行定义.

4.98.1.4 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_lks32mc09x.qep.c](#) 第 13 行定义.

4.98.1.5 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_lks32mc09x.qep.c](#) 第 114 行定义.

4.98.1.6 HAL_QEP_SetGpio()

```
void HAL_QEP_SetGpio (
    HAL_QEP_x qepx,
    HAL_GPIO_Px px )
```

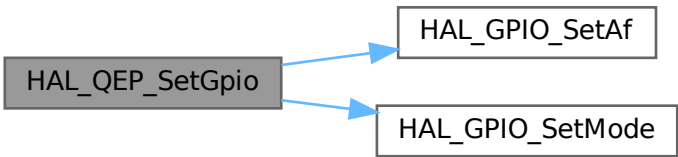
设置QEP模块的引脚

参数

<i>qepx</i>	QEP模块
<i>px</i>	引脚

在文件 [hal_lks32mc09x.qep.c](#) 第 124 行定义.

函数调用图:



4.98.1.7 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_lks32mc09x_qep.c](#) 第 82 行定义.

4.98.1.8 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_lks32mc09x_qep.c](#) 第 72 行定义.

4.99 hal_lks32mc09x_qep.c

[浏览该文件的文档.](#)

```
00001 #include "../include/hal_lksmcu.reg.h"
00002 #include "../include/hal_lksmcu.qep.h"
00003 #include "../include/hal_lksmcu.gpio.h"
00004 #include "../hal_lks32mc09x.h"
00005 #include "lks32mc09x-user-manual.h"
00006 #include "../include/hal_lksmcu.reg.h"
00007 #include "../include/hal_lks32mc09x_reg.h"
00008
00013 void HAL_QEP_Reset (HAL_QEP_x qep)
00014 {
00015     switch (qep)
00016     {
00017         case HAL_QEP_0:
00018             HAL_REG_SYS_UNLOCK();
00019             HAL_REG_SET(SYS_SFT_RST, SYS_SFT_RST_QEP0_SFT_RST_BIT);
00020             HAL_REG_RESET(SYS_SFT_RST, SYS_SFT_RST_QEP0_SFT_RST_BIT);
00021             HAL_REG_SYS_LOCK();
00022             break;
00023         default:
00024             break;
00025     }
00026 }
00027
00032 void HAL_QEP_Enable (HAL_QEP_x qep)
```

```

00033 {
00034     switch (qep)
00035     {
00036         case HAL_QEP_0:
00037             HAL_REG_SYS_UNLOCK();
00038             HAL_REG_SET(SYS_CLK_FEN, SYS_CLK_FEN_QEP0_CLK_EN_BIT);
00039             HAL_REG_SYS_LOCK();
00040             HAL_TIMER_Enable(HAL_TIMER_2); // // qep0使用时必须使能TIMER2
00041             break;
00042         default:
00043             break;
00044     }
00045     HAL_REG_SET(HAL_QEP[qep]->CFG, QEP0_CFG_EN_BIT);
00046 }
00047
00052 void HAL_QEP_Disable(HAL_QEP_x qep)
00053 {
00054     switch (qep)
00055     {
00056         case HAL_QEP_0:
00057             HAL_REG_SYS_UNLOCK();
00058             HAL_REG_RESET(SYS_CLK_FEN, SYS_CLK_FEN_QEP0_CLK_EN_BIT);
00059             HAL_REG_SYS_LOCK();
00060             break;
00061         default:
00062             break;
00063     }
00064     HAL_REG_RESET(HAL_QEP[qep]->CFG, QEP0_CFG_EN_BIT);
00065 }
00066
00072 void HAL_QEP_SetTh(HAL_QEP_x qep, uint32_t th)
00073 {
00074     HAL_QEP[qep]->TH = th;
00075 }
00076
00082 void HAL_QEP_SetMode(HAL_QEP_x qep, HAL_QEP_MODE_x mode)
00083 {
00084     switch (mode)
00085     {
00086         case HAL_QEP_MODE_QUADRATURE:
00087             HAL_REG_WRITE_BIT_MASK_POS(HAL_QEP[qep]->CFG, QEP0_CFG_MODE, 1);
00088             break;
00089         case HAL_QEP_MODE_SYMBOL_PULSE:
00090             HAL_REG_WRITE_BIT_MASK_POS(HAL_QEP[qep]->CFG, QEP0_CFG_MODE, 2);
00091             break;
00092         case HAL_QEP_MODE_DUAL_PULSE:
00093             HAL_REG_WRITE_BIT_MASK_POS(HAL_QEP[qep]->CFG, QEP0_CFG_MODE, 3);
00094             break;
00095         default:
00096             break;
00097     }
00098 }
00099
00105 uint32_t HAL_QEP_ReadCnt(HAL_QEP_x qep)
00106 {
00107     return HAL_QEP[qep]->CNT;
00108 }
00109
00114 void HAL_QEP_ResetCnt(HAL_QEP_x qep)
00115 {
00116     HAL_QEP[qep]->CNT = 0;
00117 }
00118
00124 void HAL_QEP_SetGpio(HAL_QEP_x qepx, HAL_GPIO_Px px)
00125 {
00126     (void)qepx;
00127     HAL_GPIO_SetMode(px, HAL_GPIO_MODE_IN);
00128     HAL_GPIO_SetAf(px, HAL_GPIO_AF_QEP);
00129 }

```

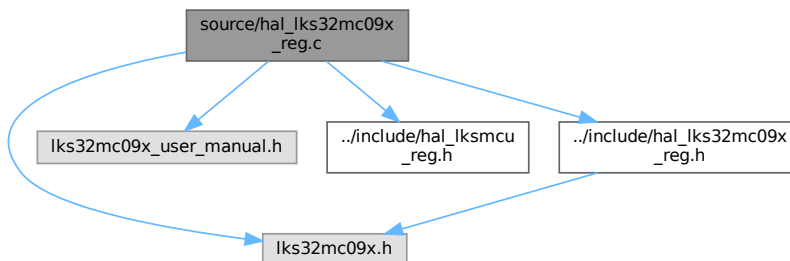
4.100 source/hal_lks32mc09x_reg.c 文件参考

```

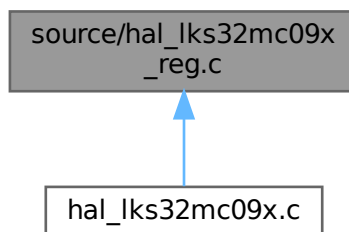
#include "lks32mc09x.h"
#include "lks32mc09x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc09x_reg.h"

```

hal_lks32mc09x_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 *)SYS_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 *)HALL0_BASE}`
- `HAL_ADC_TypeDef *const HAL_ADC [] = {(HAL_ADC_TypeDef *)ADC0_BASE, (HAL_ADC_TypeDef *)ADC1_BASE}`
- `HAL_TIMER_TypeDef *const HAL_TIMER [] = {(HAL_TIMER_TypeDef *)TIMER0_BASE, (HAL_TIMER_TypeDef *)TIMER1_BASE, (HAL_TIMER_TypeDef *)TIMER2_BASE}`
- `HAL_QEP_TypeDef *const HAL_QEP [] = {(HAL_QEP_TypeDef *)QEP0_BASE}`
- `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_CRC_TypeDef *const HAL_CRC [] = {(HAL_CRC_TypeDef *)CRC0_BASE}`
- `HAL_UART_TypeDef *const HAL_UART [] = {(HAL_UART_TypeDef *)UART0_BASE, (HAL_UART_TypeDef *)UART1_BASE}`
- `HAL_DSP_TypeDef *const HAL_DSP [] = {(HAL_DSP_TypeDef *)DSP0_BASE}`
- `HAL_IWDG_TypeDef *const HAL_IWDG [] = {(HAL_IWDG_TypeDef *)IWDG_BASE}`
- `HAL_DMA_TypeDef *const HAL_DMA [] = {(HAL_DMA_TypeDef *)DMA0_BASE + 0, (HAL_DMA_TypeDef *)DMA0_BASE + 0x20, (HAL_DMA_TypeDef *)DMA0_BASE + 0x40, (HAL_DMA_TypeDef *)DMA0_BASE + 0x60}`

4.100.1 变量说明

4.100.1.1 HAL_ADC

```
HAL_ADC_TypeDef* const HAL_ADC[] = {(HAL_ADC_TypeDef *)ADC0_BASE, (HAL_ADC_TypeDef *)ADC1_BASE}
```

在文件 [hal_lks32mc09x_reg.c](#) 第 13 行定义。

4.100.1.2 HAL_CMP

```
HAL_CMP_TypeDef* const HAL_CMP[] = {(HAL_CMP_TypeDef *)CMP_BASE}
```

在文件 [hal_lks32mc09x_reg.c](#) 第 11 行定义。

4.100.1.3 HAL_CRC

```
HAL_CRC_TypeDef* const HAL_CRC[] = {(HAL_CRC_TypeDef *)CRC0_BASE}
```

在文件 [hal_lks32mc09x_reg.c](#) 第 18 行定义。

4.100.1.4 HAL_DMA

```
HAL_DMA_TypeDef* const HAL_DMA[] = {(HAL_DMA_TypeDef *) (DMA0_BASE + 0), (HAL_DMA_TypeDef *) (DMA0←  
_BASE + 0x20), (HAL_DMA_TypeDef *) (DMA0_BASE + 0x40), (HAL_DMA_TypeDef *) (DMA0_BASE + 0x60)}
```

在文件 [hal_lks32mc09x_reg.c](#) 第 22 行定义。

4.100.1.5 HAL_DSP

```
HAL_DSP_TypeDef* const HAL_DSP[] = {(HAL_DSP_TypeDef *)DSP0_BASE}
```

在文件 [hal_lks32mc09x_reg.c](#) 第 20 行定义。

4.100.1.6 HAL_FLASH

```
HAL_FLASH_TypeDef* const HAL_FLASH[] = {(HAL_FLASH_TypeDef *)FLS_CR_BASE}
```

在文件 [hal_lks32mc09x_reg.c](#) 第 8 行定义。

4.100.1.7 HAL_GPIO

```
HAL_GPIO_TypeDef* const HAL_GPIO[] = {(HAL_GPIO_TypeDef *)GPIO0_BASE, (HAL_GPIO_TypeDef *)GPIO1←  
BASE, (HAL_GPIO_TypeDef *)GPIO2_BASE}
```

在文件 [hal_lks32mc09x_reg.c](#) 第 17 行定义。

4.100.1.8 HAL_HALL

```
HAL_HALL_TypeDef* const HAL_HALL[] = {(HAL_HALL_TypeDef *)HAL0_BASE}
```

在文件 [hal_lks32mc09x.reg.c](#) 第 12 行定义.

4.100.1.9 HAL_I2C

```
HAL_I2C_TypeDef* const HAL_I2C[] = {(HAL_I2C_TypeDef *)I2C0_BASE}
```

在文件 [hal_lks32mc09x.reg.c](#) 第 10 行定义.

4.100.1.10 HAL_IWDG

```
HAL_IWDG_TypeDef* const HAL_IWDG[] = {(HAL_IWDG_TypeDef *)IWDG_BASE}
```

在文件 [hal_lks32mc09x.reg.c](#) 第 21 行定义.

4.100.1.11 HAL_MCPWM

```
HAL_MCPWM_TypeDef* const HAL_MCPWM[] = {(HAL_MCPWM_TypeDef *)MCPWM0_BASE}
```

在文件 [hal_lks32mc09x.reg.c](#) 第 16 行定义.

4.100.1.12 HAL_QEP

```
HAL_QEP_TypeDef* const HAL_QEP[] = {(HAL_QEP_TypeDef *)QEP0_BASE}
```

在文件 [hal_lks32mc09x.reg.c](#) 第 15 行定义.

4.100.1.13 HAL_SPI

```
HAL_SPI_TypeDef* const HAL_SPI[] = {(HAL_SPI_TypeDef *)SPI0_BASE}
```

在文件 [hal_lks32mc09x.reg.c](#) 第 9 行定义.

4.100.1.14 HAL_SYS

```
HAL_SYS_TypeDef* const HAL_SYS[] = {(HAL_SYS_TypeDef *)SYS_BASE}
```

在文件 [hal_lks32mc09x.reg.c](#) 第 7 行定义.

4.100.1.15 HAL_TIMER

```
HAL_TIMER_TypeDef* const HAL_TIMER[] = {(HAL_TIMER_TypeDef *)TIMER0_BASE, (HAL_TIMER_TypeDef *)TIMER1_BASE, (HAL_TIMER_TypeDef *)TIMER2_BASE}
```

在文件 [hal_lks32mc09x.reg.c](#) 第 14 行定义.

4.100.1.16 HAL_UART

```
HAL_UART_TypeDef* const HAL_UART[] = { (HAL_UART_TypeDef *)UART0_BASE, (HAL_UART_TypeDef *)UART1_  
BASE }
```

在文件 `hal_lks32mc09x_reg.c` 第 19 行定义.

4.101 hal_iks32mc09x_reg.c

[浏览该文件的文档.](#)

```

00001
00002 #include "lks32mc09x.h"
00003 #include "lks32mc09x.user_manual.h"
00004 #include "../include/hal_lksmcu_reg.h"
00005 #include "../include/hal_lks32mc09x_reg.h"
00006
00007 HAL_SYS_TypeDef *const HAL_SYS[] = {(HAL_SYS_TypeDef *)SYS_BASE};
00008 HAL_FLASH_TypeDef *const HAL_FLASH[] = {(HAL_FLASH_TypeDef *)FLS_CR_BASE};
00009 HAL_SPI_TypeDef *const HAL_SPI[] = {(HAL_SPI_TypeDef *)SPI0_BASE};
00010 HAL_I2C_TypeDef *const HAL_I2C[] = {(HAL_I2C_TypeDef *)I2C0_BASE};
00011 HAL_CMP_TypeDef *const HAL_CMP[] = {(HAL_CMP_TypeDef *)CMP_BASE};
00012 HAL_HALL_TypeDef *const HAL_HALL[] = {(HAL_HALL_TypeDef *)HALL0_BASE};
00013 HAL_ADC_TypeDef *const HAL_ADC[] = {(HAL_ADC_TypeDef *)ADC0_BASE, (HAL_ADC_TypeDef *)ADC1_BASE};
00014 HAL_TIMER_TypeDef *const HAL_TIMER[] = {(HAL_TIMER_TypeDef *)TIMER0_BASE, (HAL_TIMER_TypeDef
*)TIMER1_BASE, (HAL_TIMER_TypeDef *)TIMER2_BASE};
00015 HAL_QEP_TypeDef *const HAL_QEP[] = {(HAL_QEP_TypeDef *)QEP0_BASE};
00016 HAL_MCPWM_TypeDef *const HAL_MCPWM[] = {(HAL_MCPWM_TypeDef *)MCPWM0_BASE};
00017 HAL_GPIO_TypeDef *const HAL_GPIO[] = {(HAL_GPIO_TypeDef *)GPIO0_BASE, (HAL_GPIO_TypeDef *)GPIO1_BASE,
(HAL_GPIO_TypeDef *)GPIO2_BASE};
00018 HAL_CRC_TypeDef *const HAL_CRC[] = {(HAL_CRC_TypeDef *)CRC0_BASE};
00019 HAL_UART_TypeDef *const HAL_UART[] = {(HAL_UART_TypeDef *)UART0_BASE, (HAL_UART_TypeDef *)UART1_BASE};
00020 HAL_DSP_TypeDef *const HAL_DSP[] = {(HAL_DSP_TypeDef *)DSP0_BASE};
00021 HAL_IWDG_TypeDef *const HAL_IWDG[] = {(HAL_IWDG_TypeDef *)IWDG_BASE};
00022 HAL_DMA_TypeDef *const HAL_DMA[] = {(HAL_DMA_TypeDef *)DMA0_BASE + 0, (HAL_DMA_TypeDef *)DMA0_BASE
+ 0x20, (HAL_DMA_TypeDef *)DMA0_BASE + 0x40, (HAL_DMA_TypeDef *)DMA0_BASE + 0x60};

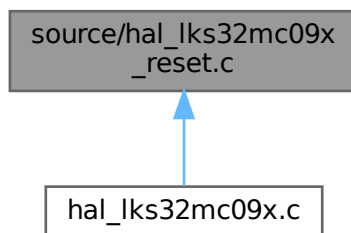
```

4.102 source/hal_lks32mc09x_reset.c 文件参考

```
#include "../include/hal_lksmcu_reset.h"
#include "../hal_lks32mc09x.h"
#include "lks32mc09x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc09x_reg.h"
hal_lks32mc09x_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)

4.102.1 函数说明

4.102.1.1 HAL_RESET_ResetCore()

```
void HAL_RESET_ResetCore (  
    void )
```

复位内核

在文件 [hal_lks32mc09x.reset.c](#) 第 10 行定义.

4.102.1.2 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_lks32mc09x.reset.c](#) 第 29 行定义.

函数调用图:



4.102.1.3 HAL_RESET_ResetRegister()

```
void HAL_RESET_ResetRegister (
    void )
```

寄存器复位

在文件 [hal_lks32mc09x.reset.c](#) 第 17 行定义.

这是这个函数的调用关系图:



4.103 hal_lks32mc09x.reset.c

[浏览该文件的文档.](#)

```

00001
00002 #include "../include/hal_lksmcu.reset.h"
00003 #include "../hal_lks32mc09x.h"
00004 #include "lks32mc09x.user_manual.h"
00005 #include "../include/hal_lksmcu.reg.h"
00006 #include "../include/hal_lks32mc09x.reg.h"
00010 void HAL_RESET_ResetCore()
00011 {
00012     NVIC.SystemReset();
00013 }
00017 void HAL_RESET_ResetRegister()
00018 {
00019     // 系统寄存器解锁
00020     HAL_REG_SYS_UNLOCK();
00021     HAL_REG_SET(SYS_SFT_RST, 0xffff);
00022     HAL_REG_RESET(SYS_SFT_RST, 0);
00023     // 系统寄存器上锁
00024     HAL_REG_SYS_LOCK();
00025 }
00029 void HAL_RESET_ResetGlobal()
00030 {
00031     HAL_RESET_ResetRegister();
00032     NVIC.SystemReset();
00033 }
```

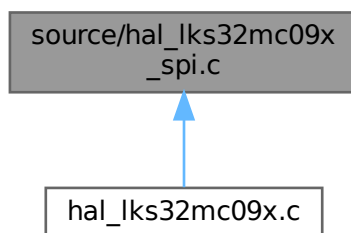
4.104 source/hal_lks32mc09x_spi.c 文件参考

```

#include "../include/hal_lksmcu_spi.h"
#include "../hal_lks32mc09x.h"
#include "lks32mc09x.user_manual.h"
#include "../include/hal_lksmcu.reg.h"
#include "../include/hal_lks32mc09x.reg.h"
hal_lks32mc09x_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_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

4.104.1 函数说明

4.104.1.1 HAL_SPI_Disable()

```
void HAL_SPI_Disable (
    HAL_SPI_TypeDef* spi )
```

模块关闭

参数

<i>spi</i>	SPI模块号

在文件 [hal_lks32mc09x_spi.c](#) 第 37 行定义.

4.104.1.2 HAL_SPI_Enable()

```
void HAL_SPI_Enable (
    HAL_SPI_TypeDef* spi )
```

模块打开

参数

<i>spi</i>	SPI模块号

在文件 [hal_lks32mc09x_spi.c](#) 第 25 行定义.

4.104.1.3 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_lks32mc09x_spi.c](#) 第 12 行定义.

4.104.1.4 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_lks32mc09x_spi.c](#) 第 257 行定义.

4.104.1.5 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

返回

接收状态

在文件 [hal_lks32mc09x_spi.c](#) 第 332 行定义.

4.104.1.6 HAL_SPI_RxTxData()

```
uint8_t HAL_SPI_RxTxData (
    HAL_SPI_TypeDef * 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

返回

发送和接收状态

在文件 [hal_lks32mc09x_spi.c](#) 第 284 行定义.

4.104.1.7 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

返回

发送和接收状态

在文件 [hal_lks32mc09x_spi.c](#) 第 357 行定义.

4.104.1.8 HAL_SPI_SetBaud()

```
void HAL_SPI_SetBaud (
    HAL_SPI_TypeDef * spi,
    uint32_t baud )
```

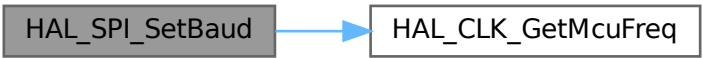
设置波特率

参数

<i>spi</i>	SPI模块号

在文件 [hal_lks32mc09x_spi.c](#) 第 49 行定义.

函数调用图:



4.104.1.9 HAL_SPI_SetGpioClk()

```
void HAL_SPI_SetGpioClk (
    HAL_SPI_TypeDef* spi,
    GPIO_TypeDef* pin )
```

初始化Clk引脚

参数

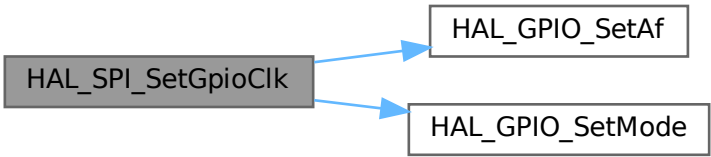
<i>spi</i>	SPI模块号
<i>pin</i>	GPIO引脚

参见

需要先初始化SPI

在文件 [hal_lks32mc09x_spi.c](#) 第 186 行定义.

函数调用图:



4.104.1.10 HAL_SPI_SetGpioCs()

```
void HAL_SPI_SetGpioCs (
    HAL_SPI_TypeDef* spi,
    GPIO_TypeDef* pin )
```

初始化CS引脚

参数

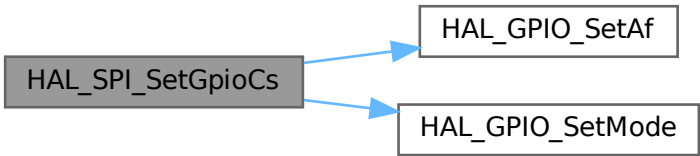
<i>spi</i>	SPI模块号
<i>pin</i>	GPIO引脚

参见

需要先初始化SPI

在文件 [hal_lks32mc09x_spi.c](#) 第 123 行定义.

函数调用图:



4.104.1.11 HAL_SPI_SetGpioDi()

```
void HAL_SPI_SetGpioDi (
    HAL_SPI_x spi,
    HAL_GPIO_Px pin )
```

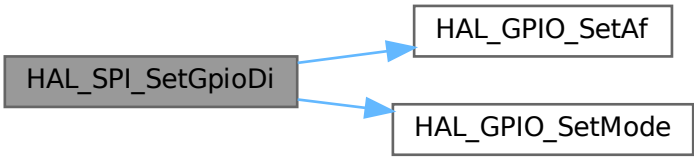
初始化SPI的输入引脚

参数

<i>spi</i>	SPI模块号
<i>pin</i>	GPIO引脚

在文件 [hal_lks32mc09x_spi.c](#) 第 206 行定义.

函数调用图:



4.104.1.12 HAL_SPI_SetGpioDo()

```
void HAL_SPI_SetGpioDo (
    HAL_SPI_x spi,
    HAL_GPIO_Px pin )
```

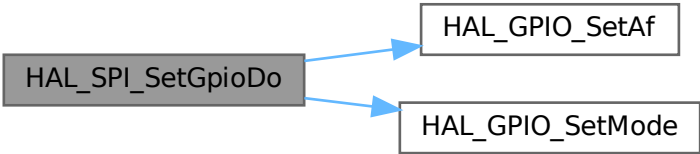
初始化SPI的输出引脚

参数

<i>spi</i>	SPI模块号
<i>pin</i>	GPIO引脚

在文件 [hal_lks32mc09x_spi.c](#) 第 217 行定义.

函数调用图:



4.104.1.13 HAL_SPI_SetGpioMiso()

```
void HAL_SPI_SetGpioMiso (
    HAL_SPI_x spi,
    HAL_GPIO_Px pin )
```

初始化Miso引脚

参数

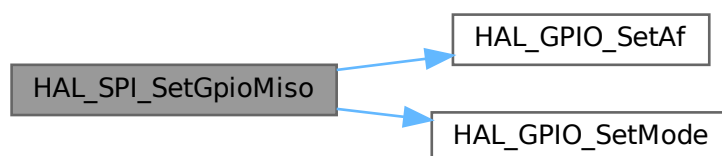
<i>spi</i>	SPI模块号
<i>pin</i>	GPIO引脚

参见

需要先初始化SPI

在文件 [hal_lks32mc09x_spi.c](#) 第 165 行定义.

函数调用图:



4.104.1.14 HAL_SPI_SetGpioMosi()

```
void HAL_SPI_SetGpioMosi (
    HAL_SPI_x spi,
    HAL_GPIO_Px pin )
```

初始化Mosi引脚

参数

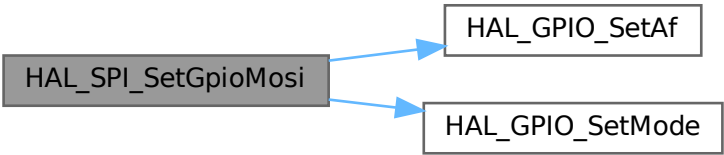
<i>spi</i>	SPI模块号
<i>pin</i>	GPIO引脚

参见

需要先初始化SPI

在文件 [hal_lks32mc09x_spi.c](#) 第 144 行定义.

函数调用图:



4.104.1.15 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_lks32mc09x_spi.c](#) 第 60 行定义.

4.104.1.16 HAL_SPI_TxData()

```
uint8_t HAL_SPI_TxData (
    HAL_SPI_TypeDef* spi,
    uint8_t* data,
    uint32_t len )
```

发送数据

参数

spi	SPI模块号
data	要发送的数据
len	数据长度

返回

发送状态

在文件 [hal_lks32mc09x_spi.c](#) 第 231 行定义.

4.104.1.17 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

返回

发送状态

在文件 [hal_lks32mc09x_spi.c](#) 第 309 行定义.

4.105 hal_lks32mc09x_spi.c

[浏览该文件的文档.](#)

```
00001
00002 #include "../include/hal_lksmcu_spi.h"
00003 #include "../hal_lks32mc09x.h"
00004 #include "lks32mc09x_user_manual.h"
00005 #include "../include/hal_lksmcu_reg.h"
00006 #include "../include/hal_lks32mc09x_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_SPI0_SFT_RST_BIT);
00017     HAL_REG_RESET(SYS_SFT_RST, SYS_SFT_RST_SPI0_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_SPI0_CLK_EN_BIT);
00029     HAL_REG_SYS_LOCK();
00030     HAL_REG_SET(HAL_SPI[spi]->CFG, SPI0_CFG_EN_BIT);
00031 }
00032
00037 void HAL_SPI_Disable(HAL_SPI_TypeDef* spi)
00038 {
00039     HAL_REG_RESET(HAL_SPI[spi]->CFG, SPI0_CFG_EN_BIT);
00040     HAL_REG_SYS_UNLOCK();
00041     HAL_REG_RESET(SYS_CLK_FEN, SYS_CLK_FEN_SPI0_CLK_EN_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]->DIV, SPI0_BAUD_BAUD, cnt);
00054 }
00060 void HAL_SPI_SetMode(HAL_SPI_TypeDef* spi, HAL_SPI_Mode_TypeDef mode)
00061 {
00062     uint32_t mask = SPI0_CFG_MS_BIT | SPI0_CFG_CPHA_BIT | SPI0_CFG_CPOL_BIT | SPI0_CFG_ENDIAN_BIT;
00063     switch (mode)
```

```

00064     {
00065         case HAL_SPI_MODE_MASTER_LSB_00:
00066             HAL_REG_WRITE_BIT(HAL_SPI[spi]->CFG, mask, SPI0_CFG_MS_BIT | SPI0_CFG_ENDIAN_BIT);
00067             break;
00068         case HAL_SPI_MODE_MASTER_LSB_01:
00069             HAL_REG_WRITE_BIT(HAL_SPI[spi]->CFG, mask, SPI0_CFG_MS_BIT | SPI0_CFG_CPOL_BIT |
SPI0_CFG_ENDIAN_BIT);
00070             break;
00071         case HAL_SPI_MODE_MASTER_LSB_10:
00072             HAL_REG_WRITE_BIT(HAL_SPI[spi]->CFG, mask, SPI0_CFG_MS_BIT | SPI0_CFG_CPHA_BIT |
SPI0_CFG_ENDIAN_BIT);
00073             break;
00074         case HAL_SPI_MODE_MASTER_LSB_11:
00075             HAL_REG_WRITE_BIT(HAL_SPI[spi]->CFG, mask, SPI0_CFG_MS_BIT | SPI0_CFG_CPHA_BIT |
SPI0_CFG_CPOL_BIT | SPI0_CFG_ENDIAN_BIT);
00076             break;
00077         case HAL_SPI_MODE_MASTER_MSB_00:
00078             HAL_REG_WRITE_BIT(HAL_SPI[spi]->CFG, mask, SPI0_CFG_MS_BIT | 0);
00079             break;
00080         case HAL_SPI_MODE_MASTER_MSB_01:
00081             HAL_REG_WRITE_BIT(HAL_SPI[spi]->CFG, mask, SPI0_CFG_MS_BIT | SPI0_CFG_CPOL_BIT);
00082             break;
00083         case HAL_SPI_MODE_MASTER_MSB_10:
00084             HAL_REG_WRITE_BIT(HAL_SPI[spi]->CFG, mask, SPI0_CFG_MS_BIT | SPI0_CFG_CPHA_BIT);
00085             break;
00086         case HAL_SPI_MODE_MASTER_MSB_11:
00087             HAL_REG_WRITE_BIT(HAL_SPI[spi]->CFG, mask, SPI0_CFG_MS_BIT | SPI0_CFG_CPHA_BIT |
SPI0_CFG_CPOL_BIT);
00088             break;
00089         case HAL_SPI_MODE_SLAVE_LSB_00:
00090             HAL_REG_WRITE_BIT(HAL_SPI[spi]->CFG, mask, SPI0_CFG_ENDIAN_BIT);
00091             break;
00092         case HAL_SPI_MODE_SLAVE_LSB_01:
00093             HAL_REG_WRITE_BIT(HAL_SPI[spi]->CFG, mask, SPI0_CFG_CPOL_BIT | SPI0_CFG_ENDIAN_BIT);
00094             break;
00095         case HAL_SPI_MODE_SLAVE_LSB_10:
00096             HAL_REG_WRITE_BIT(HAL_SPI[spi]->CFG, mask, SPI0_CFG_CPHA_BIT | SPI0_CFG_ENDIAN_BIT);
00097             break;
00098         case HAL_SPI_MODE_SLAVE_LSB_11:
00099             HAL_REG_WRITE_BIT(HAL_SPI[spi]->CFG, mask, SPI0_CFG_CPHA_BIT | SPI0_CFG_CPOL_BIT |
SPI0_CFG_ENDIAN_BIT);
00100             break;
00101         case HAL_SPI_MODE_SLAVE_MSB_00:
00102             HAL_REG_WRITE_BIT(HAL_SPI[spi]->CFG, mask, 0);
00103             break;
00104         case HAL_SPI_MODE_SLAVE_MSB_01:
00105             HAL_REG_WRITE_BIT(HAL_SPI[spi]->CFG, mask, SPI0_CFG_CPOL_BIT);
00106             break;
00107         case HAL_SPI_MODE_SLAVE_MSB_10:
00108             HAL_REG_WRITE_BIT(HAL_SPI[spi]->CFG, mask, SPI0_CFG_CPHA_BIT);
00109             break;
00110         case HAL_SPI_MODE_SLAVE_MSB_11:
00111             HAL_REG_WRITE_BIT(HAL_SPI[spi]->CFG, mask, SPI0_CFG_CPHA_BIT | SPI0_CFG_CPOL_BIT);
00112             break;
00113         default:
00114             break;
00115     }
00116 }
00123 void HAL_SPI_SetGpioCs(HAL_SPI_x spi, HAL_GPIO_Px pin)
00124 {
00125     if (HAL_REG_READ_BIT(HAL_SPI[spi]->CFG, SPI0_CFG_MS_BIT) == 0)
00126     {
00127         // 从模式
00128         HAL_GPIO_SetAf(pin, HAL_GPIO_AF_SPI);
00129         HAL_GPIO_SetMode(pin, HAL_GPIO_MODE_IN);
00130     }
00131     else
00132     {
00133         // 主模式
00134         HAL_GPIO_SetAf(pin, HAL_GPIO_AF_SPI);
00135         HAL_GPIO_SetMode(pin, HAL_GPIO_MODE_OUT);
00136     }
00137 }
00144 void HAL_SPI_SetGpioMosi(HAL_SPI_x spi, HAL_GPIO_Px pin)
00145 {
00146     if (HAL_REG_READ_BIT(HAL_SPI[spi]->CFG, SPI0_CFG_MS_BIT) == 0)
00147     {
00148         // 从模式
00149         HAL_GPIO_SetAf(pin, HAL_GPIO_AF_SPI);
00150         HAL_GPIO_SetMode(pin, HAL_GPIO_MODE_IN);
00151     }
00152     else
00153     {
00154         // 主模式
00155         HAL_GPIO_SetAf(pin, HAL_GPIO_AF_SPI);
00156         HAL_GPIO_SetMode(pin, HAL_GPIO_MODE_OUT);
00157     }

```

```

00158 }
00165 void HAL_SPI_SetGpioMiso(HAL_SPI_x spi, HAL_GPIO_Px pin)
00166 {
00167     if (HAL_REG_READ_BIT(HAL_SPI[spi]->CFG, SPI0_CFG_MS_BIT) == 0)
00168     {
00169         // 从模式
00170         HAL_GPIO_SetAf(pin, HAL_GPIO_AF_SPI);
00171         HAL_GPIO_SetMode(pin, HAL_GPIO_MODE_OUT);
00172     }
00173     else
00174     {
00175         // 主模式
00176         HAL_GPIO_SetAf(pin, HAL_GPIO_AF_SPI);
00177         HAL_GPIO_SetMode(pin, HAL_GPIO_MODE_IN);
00178     }
00179 }
00186 void HAL_SPI_SetGpioClk(HAL_SPI_x spi, HAL_GPIO_Px pin)
00187 {
00188     if (HAL_REG_READ_BIT(HAL_SPI[spi]->CFG, SPI0_CFG_MS_BIT) == 0)
00189     {
00190         // 从模式
00191         HAL_GPIO_SetAf(pin, HAL_GPIO_AF_SPI);
00192         HAL_GPIO_SetMode(pin, HAL_GPIO_MODE_IN);
00193     }
00194     else
00195     {
00196         // 主模式
00197         HAL_GPIO_SetAf(pin, HAL_GPIO_AF_SPI);
00198         HAL_GPIO_SetMode(pin, HAL_GPIO_MODE_OUT);
00199     }
00200 }
00206 void HAL_SPI_SetGpioDi(HAL_SPI_x spi, HAL_GPIO_Px pin)
00207 {
00208     (void)spi;
00209     HAL_GPIO_SetAf(pin, HAL_GPIO_AF_SPI);
00210     HAL_GPIO_SetMode(pin, HAL_GPIO_MODE_IN);
00211 }
00217 void HAL_SPI_SetGpioDo(HAL_SPI_x spi, HAL_GPIO_Px pin)
00218 {
00219     (void)spi;
00220     HAL_GPIO_SetAf(pin, HAL_GPIO_AF_SPI);
00221     HAL_GPIO_SetMode(pin, HAL_GPIO_MODE_OUT);
00222 }
00223
00231 uint8_t HAL_SPI_TxData(HAL_SPI_x spi, uint8_t *data, uint32_t len)
00232 {
00233     uint32_t i = 0;
00234     HAL_REG_WRITE(HAL_SPI[spi]->SIZE, 8);
00235     HAL_REG_SET(HAL_SPI[spi]->DIV, SPI0_BAUD_TRANS_MODE_BIT);
00236     HAL_REG_WRITE_BIT_MASK_POS(HAL_SPI[spi]->CFG, SPI0_CFG_DUPLEX, 2);
00237
00238     for (i = 0; i < len; i++)
00239     {
00240         HAL_REG_SET(HAL_SPI[spi]->IE, SPI0_IE_CMPLT_IF_BIT);
00241         HAL_REG_WRITE_BIT(HAL_SPI[spi]->IE, 0x07, BIT2);
00242         HAL_REG_WRITE(HAL_SPI[spi]->TX_DATA, data[i]);
00243         while (HAL_REG_READ_BIT(HAL_SPI[spi]->IE, SPI0_IE_CMPLT_IF_BIT) == 0)
00244         {
00245         }
00246     }
00247     return 0;
00248 }
00249
00257 uint8_t HAL_SPI_RxData(HAL_SPI_x spi, uint8_t *data, uint32_t len)
00258 {
00259     uint32_t i = 0;
00260     HAL_REG_WRITE(HAL_SPI[spi]->SIZE, 8);
00261     HAL_REG_SET(HAL_SPI[spi]->DIV, SPI0_BAUD_TRANS_MODE_BIT);
00262     HAL_REG_WRITE_BIT_MASK_POS(HAL_SPI[spi]->CFG, SPI0_CFG_DUPLEX, 3);
00263     for (i = 0; i < len; i++)
00264     {
00265         HAL_REG_WRITE_BIT(HAL_SPI[spi]->IE, 0x07, BIT2);
00266         HAL_REG_WRITE(HAL_SPI[spi]->TX_DATA, 0); // 发送一个空字节,以触发接收
00267         while (HAL_REG_READ(HAL_SPI[spi]->IE & BIT2) == 0)
00268         {
00269         }
00270         data[i] = HAL_REG_READ(HAL_SPI[spi]->RX_DATA);
00271     }
00272     // 实现接收数据的代码
00273     return 0; // 返回接收状态
00274 }
00275
00284 uint8_t HAL_SPI_RxTxData(HAL_SPI_x spi, uint8_t *rxdata, uint8_t *txdata, uint32_t len)
00285 {
00286     uint32_t i = 0;
00287     HAL_REG_WRITE(HAL_SPI[spi]->SIZE, 8);
00288     HAL_REG_SET(HAL_SPI[spi]->DIV, SPI0_BAUD_TRANS_MODE_BIT);

```

```

00289     HAL_REG_WRITE_BIT_MASK_POS(HAL_SPI[spi]->CFG, SPI0_CFG_DUPLEX, 0);
00290     for (i = 0; i < len; i++)
00291     {
00292         HAL_REG_WRITE_BIT(HAL_SPI[spi]->IE, 0x07, BIT2);
00293         HAL_REG_WRITE(HAL_SPI[spi]->TX_DATA, txdata[i]);
00294         while (HAL_REG_READ(HAL_SPI[spi]->IE & BIT2) == 0)
00295         {
00296         }
00297         rxdata[i] = HAL_REG_READ(HAL_SPI[spi]->RX_DATA);
00298     }
00299     return 0; // 返回发送和接收状态
00300 }
00301
00309 uint8_t HAL_SPI_TxDataWidth16(HAL_SPI_x spi, uint16_t *data, uint32_t len)
00310 {
00311     uint32_t i = 0;
00312     HAL_REG_WRITE(HAL_SPI[spi]->SIZE, 16);
00313     HAL_REG_SET(HAL_SPI[spi]->DIV, SPI0_BAUD_TRANS_MODE_BIT);
00314     HAL_REG_WRITE_BIT_MASK_POS(HAL_SPI[spi]->CFG, SPI0_CFG_DUPLEX, 2);
00315     for (i = 0; i < len; i++)
00316     {
00317         HAL_REG_WRITE_BIT(HAL_SPI[spi]->IE, 0x07, BIT2);
00318         HAL_REG_WRITE(HAL_SPI[spi]->TX_DATA, data[i]);
00319         while (HAL_REG_READ(HAL_SPI[spi]->IE & BIT2) == 0)
00320         {
00321         }
00322     }
00323     return 0;
00324 }
00332 uint8_t HAL_SPI_RxDataWidth16(HAL_SPI_x spi, uint16_t *data, uint32_t len)
00333 {
00334     uint32_t i = 0;
00335     HAL_REG_WRITE(HAL_SPI[spi]->SIZE, 16);
00336     HAL_REG_SET(HAL_SPI[spi]->DIV, SPI0_BAUD_TRANS_MODE_BIT);
00337     HAL_REG_WRITE_BIT_MASK_POS(HAL_SPI[spi]->CFG, SPI0_CFG_DUPLEX, 3);
00338     for (i = 0; i < len; i++)
00339     {
00340         HAL_REG_WRITE_BIT(HAL_SPI[spi]->IE, 0x07, BIT2);
00341         HAL_REG_WRITE(HAL_SPI[spi]->TX_DATA, 0); // 发送一个空字节, 以触发接收
00342         while (HAL_REG_READ(HAL_SPI[spi]->IE & BIT2) == 0)
00343         {
00344         }
00345         data[i] = HAL_REG_READ(HAL_SPI[spi]->RX_DATA);
00346     }
00347     return 0; // 返回接收状态
00348 }
00357 uint8_t HAL_SPI_RxTxDataWidth16(HAL_SPI_x spi, uint16_t *rxdata, uint16_t *txdata, uint32_t len)
00358 {
00359     uint32_t i = 0;
00360     HAL_REG_WRITE(HAL_SPI[spi]->SIZE, 16);
00361     HAL_REG_SET(HAL_SPI[spi]->DIV, SPI0_BAUD_TRANS_MODE_BIT);
00362     HAL_REG_WRITE_BIT_MASK_POS(HAL_SPI[spi]->CFG, SPI0_CFG_DUPLEX, 0);
00363     for (i = 0; i < len; i++)
00364     {
00365         HAL_REG_WRITE_BIT(HAL_SPI[spi]->IE, 0x07, BIT2);
00366         HAL_REG_WRITE(HAL_SPI[spi]->TX_DATA, txdata[i]);
00367         while (HAL_REG_READ(HAL_SPI[spi]->IE & BIT2) == 0)
00368         {
00369         }
00370         rxdata[i] = HAL_REG_READ(HAL_SPI[spi]->RX_DATA);
00371     }
00372     return 0; // 返回发送和接收状态
00373 }

```

4.106 source/hal_lks32mc09x_test.c 文件参考

```

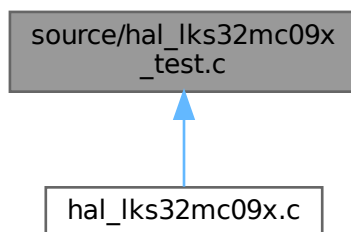
#include "../hal_lks32mc09x.h"
#include "lks32mc09x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc09x_reg.h"

```

hal_lks32mc09x_test.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_TestRegAddr](#) (void)

4.106.1 函数说明

4.106.1.1 HAL_TestRegAddr()

```
uint32_t HAL_TestRegAddr (  
    void )
```

在文件 [hal_lks32mc09x_test.c](#) 第 6 行定义.

4.107 hal_lks32mc09x_test.c

[浏览该文件的文档.](#)

```
00001  
00002 #include "../hal_lks32mc09x.h"  
00003 #include "lks32mc09x-user-manual.h"  
00004 #include "../include/hal_lksmcu-reg.h"  
00005 #include "../include/hal_lks32mc09x-reg.h"  
00006 uint32_t HAL_TestRegAddr(void)  
00007 {  
00008     return 0;  
00009 }
```

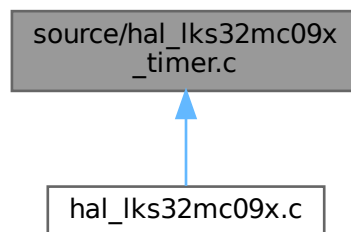
4.108 source/hal_lks32mc09x_timer.c 文件参考

```
#include "../include/hal_lksmcu-timer.h"  
#include "../include/hal_lksmcu-gpio.h"  
#include "../hal_lks32mc09x.h"  
#include "lks32mc09x-user-manual.h"  
#include "../include/hal_lksmcu-reg.h"
```


#include "../include/hal_lks32mc09x_reg.h"
 hal_lks32mc09x_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)
- 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)
- uint8_t [HAL_TIMER_IrqOverflow](#) (HAL_TIMER_x timer_x)
- void [HAL_TIMER_IrqOverflowEnable](#) (HAL_TIMER_x timer_x)
- void [HAL_TIMER_IrqOverflowDisable](#) (HAL_TIMER_x timer_x)
- void [HAL_TIMER_IrqOverflowClear](#) (HAL_TIMER_x timer_x)
- uint8_t [HAL_TIMER_IrqCompare0](#) (HAL_TIMER_x timer_x)
- void [HAL_TIMER_IrqCompare0Enable](#) (HAL_TIMER_x timer_x)

- void HAL_TIMER_IrqCompare0Disable (HAL_TIMER_x timer_x)
- void HAL_TIMER_IrqCompare0Clear (HAL_TIMER_x timer_x)
- uint8_t HAL_TIMER_IrqCompare1 (HAL_TIMER_x timer_x)
- void HAL_TIMER_IrqCompare1Enable (HAL_TIMER_x timer_x)
- void HAL_TIMER_IrqCompare1Disable (HAL_TIMER_x timer_x)
- void HAL_TIMER_IrqCompare1Clear (HAL_TIMER_x timer_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*

4.108.1 函数说明

4.108.1.1 HAL_TIMER_Disable()

```
void HAL_TIMER_Disable (
    HAL_TIMER_x timer_x )
```

在文件 `hal_lks32mc09x.timer.c` 第 56 行定义.

4.108.1.2 HAL_TIMER_Enable()

```
void HAL_TIMER_Enable (
    HAL_TIMER_x timer_x )
```

在文件 `hal_lks32mc09x.timer.c` 第 34 行定义.

这是这个函数的调用关系图:



4.108.1.3 HAL_TIMER_GetFreq()

```
uint32_t HAL_TIMER_GetFreq (
    HAL_TIMER_x timer_x )
```

在文件 `hal_lks32mc09x.timer.c` 第 157 行定义.

函数调用图:



4.108.1.4 HAL_TIMER_GetThCnt()

```
uint32_t HAL_TIMER_GetThCnt (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc09x_timer.c](#) 第 164 行定义.

这是这个函数的调用关系图:



4.108.1.5 HAL_TIMER_IrqCompare0()

```
uint8_t HAL_TIMER_IrqCompare0 (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc09x_timer.c](#) 第 267 行定义.

4.108.1.6 HAL_TIMER_IrqCompare0Clear()

```
void HAL_TIMER_IrqCompare0Clear (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc09x_timer.c](#) 第 289 行定义.

4.108.1.7 HAL_TIMER_IrqCompare0Disable()

```
void HAL_TIMER_IrqCompare0Disable (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc09x_timer.c](#) 第 285 行定义.

4.108.1.8 HAL_TIMER_IrqCompare0Enable()

```
void HAL_TIMER_IrqCompare0Enable (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc09x_timer.c](#) 第 281 行定义.

4.108.1.9 HAL_TIMER_IrqCompare1()

```
uint8_t HAL_TIMER_IrqCompare1 (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc09x_timer.c](#) 第 294 行定义.

4.108.1.10 HAL_TIMER_IrqCompare1Clear()

```
void HAL_TIMER_IrqCompare1Clear (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc09x_timer.c](#) 第 316 行定义.

4.108.1.11 HAL_TIMER_IrqCompare1Disable()

```
void HAL_TIMER_IrqCompare1Disable (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc09x_timer.c](#) 第 312 行定义.

4.108.1.12 HAL_TIMER_IrqCompare1Enable()

```
void HAL_TIMER_IrqCompare1Enable (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc09x_timer.c](#) 第 308 行定义.

4.108.1.13 HAL_TIMER_IrqOverflow()

```
uint8_t HAL_TIMER_IrqOverflow (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc09x_timer.c](#) 第 240 行定义.

4.108.1.14 HAL_TIMER_IrqOverflowClear()

```
void HAL_TIMER_IrqOverflowClear (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc09x_timer.c](#) 第 262 行定义.

4.108.1.15 HAL_TIMER_IrqOverflowDisable()

```
void HAL_TIMER_IrqOverflowDisable (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc09x_timer.c](#) 第 258 行定义.

4.108.1.16 HAL_TIMER_IrqOverflowEnable()

```
void HAL_TIMER_IrqOverflowEnable (
    HAL_TIMER_x timer_x )
```

在文件 `hal_lks32mc09x.timer.c` 第 254 行定义.

4.108.1.17 HAL_TIMER_Reset()

```
void HAL_TIMER_Reset (
    HAL_TIMER_x timer_x )
```

在文件 `hal_lks32mc09x.timer.c` 第 10 行定义.

4.108.1.18 HAL_TIMER_SetFreq()

```
void HAL_TIMER_SetFreq (
    HAL_TIMER_x timer_x,
    uint32_t freq )
```

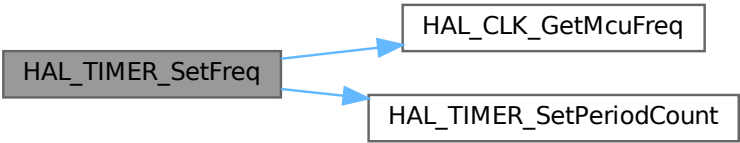
设置定时器频率

参数

<i>timer_x</i>	定时器编号
<i>freq</i>	定时器频率

在文件 `hal_lks32mc09x.timer.c` 第 127 行定义.

函数调用图:



4.108.1.19 HAL_TIMER_SetGpioIn()

```
void HAL_TIMER_SetGpioIn (
    HAL_TIMER_x timer_x,
    HAL_GPIO_Px px )
```

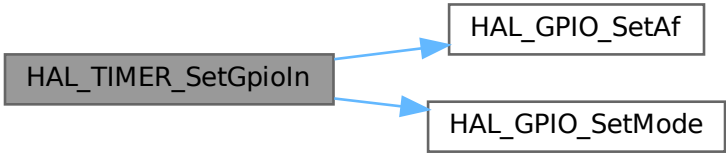
设置定时器输入gpio

参数

<i>timer</i> ↔ _x	定时器编号
<i>px</i>	gpio编号

在文件 [hal_lks32mc09x_timer.c](#) 第 355 行定义.

函数调用图:



4.108.1.20 HAL_TIMER_SetGpioOut()

```
void HAL_TIMER_SetGpioOut (
    HAL_TIMER_x timer_x,
    HAL_GPIO_Px px )
```

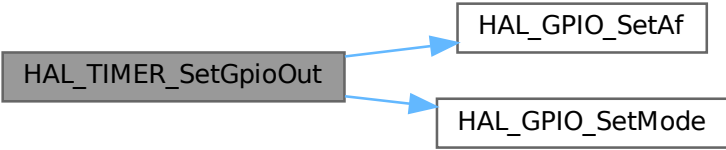
设置定时器输出gpio

参数

<i>timer</i> ↔ _x	定时器编号
<i>px</i>	gpio编号

在文件 [hal_lks32mc09x_timer.c](#) 第 326 行定义.

函数调用图:



4.108.1.21 HAL_TIMER.SetPeriodCount()

```
void HAL_TIMER.SetPeriodCount (
    HAL_TIMER.x timer_x,
    uint32_t count )
```

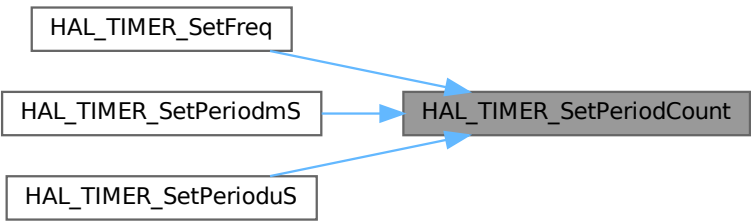
设置定时器周期

参数

<i>timer_x</i>	定时器编号
<i>count</i>	定时器周期

在文件 [hal_lks32mc09x_timer.c](#) 第 82 行定义.

这是这个函数的调用关系图:



4.108.1.22 HAL_TIMER.SetPeriodmS()

```
void HAL_TIMER.SetPeriodmS (
    HAL_TIMER.x timer_x,
    uint32_t ms )
```

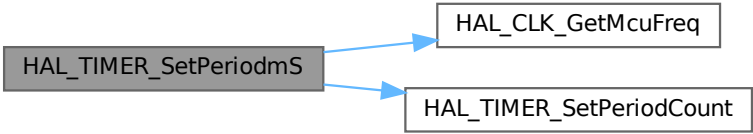
设置定时器周期(ms)

参数

<i>timer_x</i>	定时器编号
<i>ms</i>	定时器周期(ms)

在文件 [hal_lks32mc09x_timer.c](#) 第 139 行定义.

函数调用图:



4.108.1.23 HAL_TIMER_SetPerioduS()

```
void HAL_TIMER_SetPerioduS (
    HAL_TIMER_x timer_x,
    uint32_t us )
```

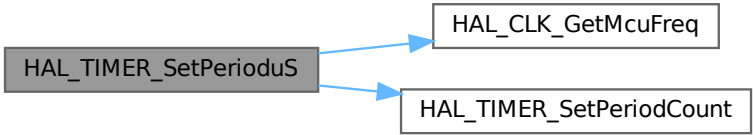
设置定时器周期(us)

参数

<i>timer_x</i>	定时器编号
<i>us</i>	定时器周期(us)

在文件 [hal_lks32mc09x.timer.c](#) 第 150 行定义.

函数调用图:



4.108.1.24 HAL_TIMER_SetPwm()

```
void HAL_TIMER_SetPwm (
    HAL_TIMER_x timer_x,
    int32_t t0,
    int32_t t1 )
```

在文件 [hal_lks32mc09x.timer.c](#) 第 180 行定义.

4.108.1.25 HAL_TIMER_SetPwmQ15()

```
void HAL_TIMER_SetPwmQ15 (
    HAL_TIMER_x timer_x,
    int16_t t0,
    int16_t t1 )
```

在文件 [hal_lks32mc09x.timer.c](#) 第 186 行定义.

函数调用图:



4.108.1.26 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_lks32mc09x.timer.c](#) 第 195 行定义.

4.108.1.27 HAL_TIMER_StartCounting()

```
void HAL_TIMER_StartCounting (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc09x.timer.c](#) 第 169 行定义.

4.108.1.28 HAL_TIMER_StopCounting()

```
void HAL_TIMER_StopCounting (
    HAL_TIMER_x timer_x )
```

在文件 [hal_lks32mc09x.timer.c](#) 第 174 行定义.

4.109 hal_lks32mc09x_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_lks32mc09x.h"
00005 #include "lks32mc09x_user_manual.h"
00006 #include "../include/hal_lksmcu_reg.h"
00007 #include "../include/hal_lks32mc09x_reg.h"
00008
00009 // 模块复位
00010 void HAL_TIMER_Reset(HAL_TIMER_x timer_x)
00011 {
00012     HAL_REG_SYS_UNLOCK();
00013     switch (timer_x)
00014     {
00015         case HAL_TIMER_0:
00016             HAL_REG_SET(SYS_SFT_RST, SYS_SFT_RST_TIMER0_SFT_RST_BIT);
00017             HAL_REG_RESET(SYS_SFT_RST, SYS_SFT_RST_TIMER0_SFT_RST_BIT);
00018             break;
00019         case HAL_TIMER_1:
00020             HAL_REG_SET(SYS_SFT_RST, SYS_SFT_RST_TIMER1_SFT_RST_BIT);
00021             HAL_REG_RESET(SYS_SFT_RST, SYS_SFT_RST_TIMER1_SFT_RST_BIT);
00022             break;
00023         case HAL_TIMER_2:
00024             HAL_REG_SET(SYS_SFT_RST, SYS_SFT_RST_TIMER2_SFT_RST_BIT);
00025             HAL_REG_RESET(SYS_SFT_RST, SYS_SFT_RST_TIMER2_SFT_RST_BIT);
00026             break;
00027         default:
00028             break;
00029     }
00030     // 系统寄存器上锁
00031     HAL_REG_SYS_LOCK();
00032 }
00033 // 模块打开
00034 void HAL_TIMER_Enable(HAL_TIMER_x timer_x)
00035 {
00036     // 系统寄存器解锁
00037     HAL_REG_SYS_UNLOCK();
00038     switch (timer_x)
00039     {
00040         case HAL_TIMER_0:
00041             HAL_REG_SET(SYS_CLK_FEN, SYS_CLK_FEN_TIMER0_CLK_EN_BIT);
00042             break;
00043         case HAL_TIMER_1:
00044             HAL_REG_SET(SYS_CLK_FEN, SYS_CLK_FEN_TIMER1_CLK_EN_BIT);
00045             break;
00046         case HAL_TIMER_2:
00047             HAL_REG_SET(SYS_CLK_FEN, SYS_CLK_FEN_TIMER2_CLK_EN_BIT);
00048             break;
00049         default:
00050             break;
00051     }
00052     // 系统寄存器上锁
00053     HAL_REG_SYS_LOCK();
00054 }
00055 // 模块关闭
00056 void HAL_TIMER_Disable(HAL_TIMER_x timer_x)
00057 {
00058     // 系统寄存器解锁
00059     HAL_REG_SYS_UNLOCK();
00060     switch (timer_x)
00061     {
00062         case HAL_TIMER_0:
00063             HAL_REG_RESET(SYS_CLK_FEN, SYS_CLK_FEN_TIMER0_CLK_EN_BIT);
00064             break;
00065         case HAL_TIMER_1:
00066             HAL_REG_RESET(SYS_CLK_FEN, SYS_CLK_FEN_TIMER1_CLK_EN_BIT);
00067             break;
00068         case HAL_TIMER_2:
00069             HAL_REG_RESET(SYS_CLK_FEN, SYS_CLK_FEN_TIMER2_CLK_EN_BIT);
00070             break;
00071         default:
00072             break;
00073     }
00074     // 系统寄存器上锁
00075     HAL_REG_SYS_LOCK();
00076 }
00082 void HAL_TIMER_SetPeriodCount(HAL_TIMER_x timer_x, uint32_t count)
00083 {
00084     if (count > 0)
00085     {
00086         count = count - 1;
00087     }

```

```

00088     if ((timer_x == HAL_TIMER_0) || (timer_x == HAL_TIMER_1))
00089     {
00090
00091         uint32_t div = 0;
00092         for (; count > TIMER0_TH_TH_MAX;)
00093         {
00094             count >>= 1;
00095             div++;
00096         }
00097         if (div > TIMER0_CFG2_CLK_DIV_MAX)
00098         {
00099             div = TIMER0_CFG2_CLK_DIV_MAX;
00100             count = TIMER0_TH_TH_MAX;
00101         }
00102         HAL_REG_WRITE_BIT_MASK_POS(HAL_TIMER[timer_x]->CFG2, TIMER0_CFG2_CLK_DIV, div);
00103         HAL_REG_WRITE_BIT_MASK_POS(HAL_TIMER[timer_x]->TH, TIMER0_TH_TH, count);
00104     }
00105     else
00106     {
00107         uint32_t div = 0;
00108         for (; count > TIMER2_TH_TH_MAX;)
00109         {
00110             count >>= 1;
00111             div++;
00112         }
00113         if (div > TIMER2_CFG2_CLK_DIV_MAX)
00114         {
00115             div = TIMER2_CFG2_CLK_DIV_MAX;
00116             count = TIMER2_TH_TH_MAX;
00117         }
00118         HAL_REG_WRITE_BIT_MASK_POS(HAL_TIMER[timer_x]->CFG2, TIMER2_CFG2_CLK_DIV, div);
00119         HAL_REG_WRITE_BIT_MASK_POS(HAL_TIMER[timer_x]->TH, TIMER2_TH_TH, count);
00120     }
00121 }
00127 void HAL_TIMER_SetFreq(HAL_TIMER_x timer_x, uint32_t freq)
00128 {
00129     uint32_t mclk = HAL_CLK_GetMcuFreq();
00130     uint32_t th = (mclk / freq);
00131     HAL_TIMER_SetPeriodCount(timer_x, th);
00132 }
00133
00139 void HAL_TIMER_SetPeriodms(HAL_TIMER_x timer_x, uint32_t ms)
00140 {
00141     uint32_t mclk = HAL_CLK_GetMcuFreq();
00142     uint32_t th = (mclk / 1000) * ms;
00143     HAL_TIMER_SetPeriodCount(timer_x, th);
00144 }
00150 void HAL_TIMER_SetPeriodus(HAL_TIMER_x timer_x, uint32_t us)
00151 {
00152     uint32_t mclk = HAL_CLK_GetMcuFreq();
00153     uint32_t th = (mclk / 1000000) * us;
00154     HAL_TIMER_SetPeriodCount(timer_x, th);
00155 }
00156 // 获取当前的pwm频率
00157 uint32_t HAL_TIMER_GetFreq(HAL_TIMER_x timer_x)
00158 {
00159     uint32_t mclk = HAL_CLK_GetMcuFreq();
00160     uint32_t freq = (mclk / HAL_TIMER[timer_x]->TH);
00161     return freq;
00162 }
00163 // 获取计数器最大值
00164 uint32_t HAL_TIMER_GetThCnt(HAL_TIMER_x timer_x)
00165 {
00166     return HAL_TIMER[timer_x]->TH;
00167 }
00168 // 开始计数
00169 void HAL_TIMER_StartCounting(HAL_TIMER_x timer_x)
00170 {
00171     HAL_REG_SET(HAL_TIMER[timer_x]->CFG2, TIMER0_CFG2_EN_BIT);
00172 }
00173 // 停止计数 (同时计数器回到0)
00174 void HAL_TIMER_StopCounting(HAL_TIMER_x timer_x)
00175 {
00176     HAL_REG_RESET(HAL_TIMER[timer_x]->CFG2, TIMER0_CFG2_EN_BIT);
00177     HAL_TIMER[timer_x]->CNT = 0;
00178 }
00179 // 设置pwm波形 仅PWM模式下有效
00180 void HAL_TIMER_SetPwm(HAL_TIMER_x timer_x, int32_t t0, int32_t t1)
00181 {
00182     HAL_REG_WRITE(HAL_TIMER[timer_x]->CHN0, t0);
00183     HAL_REG_WRITE(HAL_TIMER[timer_x]->CHN1, t1);
00184 }
00185 // 设置pwm波形q15格式输入 仅PWM模式下有效
00186 void HAL_TIMER_SetPwmQ15(HAL_TIMER_x timer_x, int16_t t0, int16_t t1)
00187 {
00188     uint32_t th = HAL_TIMER_GetThCnt(timer_x);
00189     t0 = t0 * th >> 15;

```

```

00190     t1         = t1 * th >> 15;
00191     HAL_REG_WRITE(HAL_TIMER[timer_x]->CHN0, t0);
00192     HAL_REG_WRITE(HAL_TIMER[timer_x]->CHN1, t1);
00193 }
00194 // 设置工作模式
00195 void HAL_TIMER_SetWorkMode(HAL_TIMER_x timer_x, HAL_TIMER_MODE_x mode_x)
00196 {
00197     switch (mode_x)
00198     {
00199     case HAL_TIMER_MODE_COUNT:
00200         HAL_REG_RESET(HAL_TIMER[timer_x]->CFG0, TIMER0_CFG0_CH_POL_BIT);
00201         HAL_REG_RESET(HAL_TIMER[timer_x]->CFG0, TIMER0_CFG0_CH_MODE_BIT);
00202         HAL_REG_RESET(HAL_TIMER[timer_x]->CFG0, TIMER0_CFG0_CH_FE_CAP_EN_BIT);
00203         HAL_REG_RESET(HAL_TIMER[timer_x]->CFG0, TIMER0_CFG0_CH_RE_CAP_EN_BIT);
00204
00205         HAL_REG_RESET(HAL_TIMER[timer_x]->CFG1, TIMER0_CFG1_CH_POL_BIT);
00206         HAL_REG_RESET(HAL_TIMER[timer_x]->CFG1, TIMER0_CFG1_CH_MODE_BIT);
00207         HAL_REG_RESET(HAL_TIMER[timer_x]->CFG1, TIMER0_CFG1_CH_FE_CAP_EN_BIT);
00208         HAL_REG_RESET(HAL_TIMER[timer_x]->CFG1, TIMER0_CFG1_CH_RE_CAP_EN_BIT);
00209         break;
00210     case HAL_TIMER_MODE_PWMOUT:
00211         HAL_REG_SET(HAL_TIMER[timer_x]->CFG0, TIMER0_CFG0_CH_POL_BIT);
00212         HAL_REG_SET(HAL_TIMER[timer_x]->CFG0, TIMER0_CFG0_CH_MODE_BIT);
00213         HAL_REG_RESET(HAL_TIMER[timer_x]->CFG0, TIMER0_CFG0_CH_FE_CAP_EN_BIT);
00214         HAL_REG_RESET(HAL_TIMER[timer_x]->CFG0, TIMER0_CFG0_CH_RE_CAP_EN_BIT);
00215
00216         HAL_REG_SET(HAL_TIMER[timer_x]->CFG1, TIMER0_CFG1_CH_POL_BIT);
00217         HAL_REG_SET(HAL_TIMER[timer_x]->CFG1, TIMER0_CFG1_CH_MODE_BIT);
00218         HAL_REG_RESET(HAL_TIMER[timer_x]->CFG1, TIMER0_CFG1_CH_FE_CAP_EN_BIT);
00219         HAL_REG_RESET(HAL_TIMER[timer_x]->CFG1, TIMER0_CFG1_CH_RE_CAP_EN_BIT);
00220
00221         HAL_REG_SET(HAL_TIMER[timer_x]->IO, TIMER0_IO_MOE_BIT);
00222         break;
00223     case HAL_TIMER_MODE_PWMIN:
00224         HAL_REG_RESET(HAL_TIMER[timer_x]->CFG0, TIMER0_CFG0_CH_POL_BIT);
00225         HAL_REG_RESET(HAL_TIMER[timer_x]->CFG0, TIMER0_CFG0_CH_MODE_BIT);
00226         HAL_REG_SET(HAL_TIMER[timer_x]->CFG0, TIMER0_CFG0_CH_FE_CAP_EN_BIT);
00227         HAL_REG_RESET(HAL_TIMER[timer_x]->CFG0, TIMER0_CFG0_CH_RE_CAP_EN_BIT);
00228
00229         HAL_REG_RESET(HAL_TIMER[timer_x]->CFG1, TIMER0_CFG1_CH_POL_BIT);
00230         HAL_REG_RESET(HAL_TIMER[timer_x]->CFG1, TIMER0_CFG1_CH_MODE_BIT);
00231         HAL_REG_RESET(HAL_TIMER[timer_x]->CFG1, TIMER0_CFG1_CH_FE_CAP_EN_BIT);
00232         HAL_REG_SET(HAL_TIMER[timer_x]->CFG1, TIMER0_CFG1_CH_RE_CAP_EN_BIT);
00233         break;
00234     default:
00235         break;
00236     }
00237 }
00238 }
00239
00240 uint8_t HAL_TIMER_IrqOverflow(HAL_TIMER_x timer_x)
00241 {
00242     uint32_t val1;
00243
00244     val1 = HAL_TIMER[timer_x]->IE & HAL_TIMER[timer_x]->IF;
00245     if (val1 & BIT2)
00246     {
00247         return 1;
00248     }
00249     else
00250     {
00251         return 0;
00252     }
00253 }
00254 void HAL_TIMER_IrqOverflowEnable(HAL_TIMER_x timer_x)
00255 {
00256     HAL_REG_SET(HAL_TIMER[timer_x]->IE, BIT2);
00257 }
00258 void HAL_TIMER_IrqOverflowDisable(HAL_TIMER_x timer_x)
00259 {
00260     HAL_REG_RESET(HAL_TIMER[timer_x]->IE, BIT2);
00261 }
00262 void HAL_TIMER_IrqOverflowClear(HAL_TIMER_x timer_x)
00263 {
00264     HAL_REG_WRITE(HAL_TIMER[timer_x]->IF, BIT2);
00265 }
00266
00267 uint8_t HAL_TIMER_IrqCompare0(HAL_TIMER_x timer_x)
00268 {
00269     uint32_t val1;
00270
00271     val1 = HAL_TIMER[timer_x]->IE & HAL_TIMER[timer_x]->IF;
00272     if (val1 & BIT0)
00273     {
00274         return 1;
00275     }
00276     else

```

```

00277     {
00278         return 0;
00279     }
00280 }
00281 void HAL_TIMER_IrqCompare0Enable(HAL_TIMER_x timer_x)
00282 {
00283     HAL_REG_SET(HAL_TIMER[timer_x]->IE, BIT0);
00284 }
00285 void HAL_TIMER_IrqCompare0Disable(HAL_TIMER_x timer_x)
00286 {
00287     HAL_REG_RESET(HAL_TIMER[timer_x]->IE, BIT0);
00288 }
00289 void HAL_TIMER_IrqCompare0Clear(HAL_TIMER_x timer_x)
00290 {
00291     HAL_REG_WRITE(HAL_TIMER[timer_x]->IF, BIT0);
00292 }
00293
00294 uint8_t HAL_TIMER_IrqCompare1(HAL_TIMER_x timer_x)
00295 {
00296     uint32_t val1;
00297
00298     val1 = HAL_TIMER[timer_x]->IE & HAL_TIMER[timer_x]->IF;
00299     if (val1 & BIT1)
00300     {
00301         return 1;
00302     }
00303     else
00304     {
00305         return 0;
00306     }
00307 }
00308 void HAL_TIMER_IrqCompare1Enable(HAL_TIMER_x timer_x)
00309 {
00310     HAL_REG_SET(HAL_TIMER[timer_x]->IE, BIT1);
00311 }
00312 void HAL_TIMER_IrqCompare1Disable(HAL_TIMER_x timer_x)
00313 {
00314     HAL_REG_RESET(HAL_TIMER[timer_x]->IE, BIT1);
00315 }
00316 void HAL_TIMER_IrqCompare1Clear(HAL_TIMER_x timer_x)
00317 {
00318     HAL_REG_WRITE(HAL_TIMER[timer_x]->IF, BIT1);
00319 }
00320
00321 void HAL_TIMER_SetGpioOut(HAL_TIMER_x timer_x, HAL_GPIO_Px px)
00322 {
00323     switch (timer_x)
00324     {
00325     case HAL_TIMER_0:
00326         HAL_GPIO_SetAf(px, HAL_GPIO_AF_TIMER0);
00327         HAL_GPIO_SetMode(px, HAL_GPIO_MODE_OUT);
00328         break;
00329     case HAL_TIMER_1:
00330         HAL_GPIO_SetAf(px, HAL_GPIO_AF_TIMER1);
00331         HAL_GPIO_SetMode(px, HAL_GPIO_MODE_OUT);
00332         break;
00333     case HAL_TIMER_2:
00334         HAL_GPIO_SetAf(px, HAL_GPIO_AF_TIMER2);
00335         HAL_GPIO_SetMode(px, HAL_GPIO_MODE_OUT);
00336         break;
00337     case HAL_TIMER_3:
00338         HAL_GPIO_SetAf(px, HAL_GPIO_AF_TIMER3);
00339         HAL_GPIO_SetMode(px, HAL_GPIO_MODE_OUT);
00340         break;
00341     default:
00342         break;
00343     }
00344 }
00345
00346 void HAL_TIMER_SetGpioIn(HAL_TIMER_x timer_x, HAL_GPIO_Px px)
00347 {
00348     switch (timer_x)
00349     {
00350     case HAL_TIMER_0:
00351         HAL_GPIO_SetAf(px, HAL_GPIO_AF_TIMER0);
00352         HAL_GPIO_SetMode(px, HAL_GPIO_MODE_IN);
00353         break;
00354     case HAL_TIMER_1:
00355         HAL_GPIO_SetAf(px, HAL_GPIO_AF_TIMER1);
00356         HAL_GPIO_SetMode(px, HAL_GPIO_MODE_IN);
00357         break;
00358     case HAL_TIMER_2:
00359         HAL_GPIO_SetAf(px, HAL_GPIO_AF_TIMER2);
00360         HAL_GPIO_SetMode(px, HAL_GPIO_MODE_IN);
00361         break;
00362     case HAL_TIMER_3:
00363         HAL_GPIO_SetAf(px, HAL_GPIO_AF_TIMER3);
00364         HAL_GPIO_SetMode(px, HAL_GPIO_MODE_IN);
00365     }
00366 }

```

```

00374         break;
00375     default:
00376         break;
00377 }
00378 }

```

4.110 source/hal_lks32mc09x_tmp.c 文件参考

```

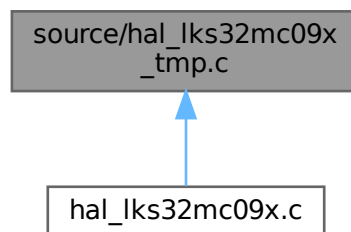
#include "../include/hal_lksmcu_tmp.h"
#include "../include/hal_lksmcu_trim.h"
#include "../hal_lks32mc09x.h"
#include "lks32mc09x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc09x_reg.h"

```

hal_lks32mc09x_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_lks32mc09x.tmp.c` 第 84 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

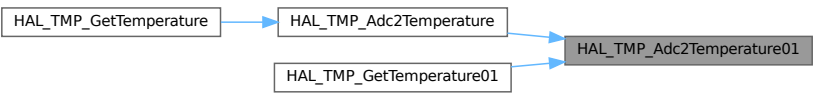
<i>TMP_x</i>	模块编号
<i>adc_value</i>	ADC值, 左对齐 (-32767 到32767)

返回

当前温度,单位0.1摄氏度

在文件 [hal_lks32mc09x.tmp.c](#) 第 94 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 )
```

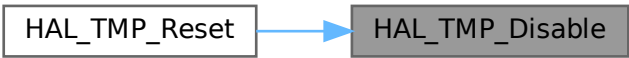
模块关闭

参数

<i>TMP_x</i>	模块编号

在文件 [hal_lks32mc09x.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_x</i>	模块编号

在文件 [hal_lks32mc09x.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

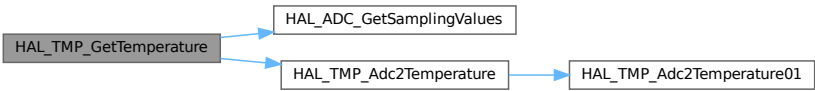
<i>TMP_x</i>	模块编号
<i>adc</i>	ADC编号
<i>sequence</i>	ADC采样序列

返回

当前温度,单位摄氏度

在文件 [hal_lks32mc09x.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

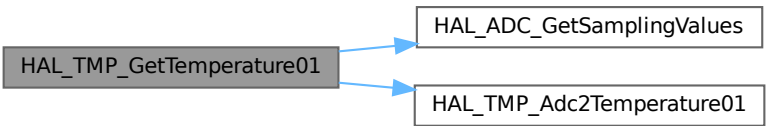
<i>TMP_x</i>	模块编号
<i>adc</i>	ADC编号
<i>sequence</i>	ADC采样序列

返回

当前温度,单位0.1摄氏度

在文件 [hal_lks32mc09x.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 )
```

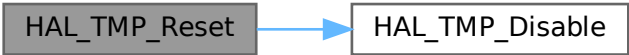
模块复位

参数

<i>HAL_TMP_x</i>	模块编号

在文件 [hal_lks32mc09x_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_lks32mc09x_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_lks32mc09x_tmp.c](#) 第 9 行定义.

4.111 hal_lks32mc09x_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_lks32mc09x.h"
00005 #include "lks32mc09x.user.manual.h"
00006 #include "../include/hal_lksmcu.reg.h"
00007 #include "../include/hal_lks32mc09x.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(0x000014D0);
00030     tmp_coef_b = HAL_TRIM_Read(0x000014D4);
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 }
00084 int32_t HAL_TMP_Adc2Temperature(HAL_TMP_x TMP_x, int32_t adc_value)
00085 {
00086     return HAL_TMP_Adc2Temperature01(TMP_x, adc_value) / 10;
00087 }
00094 int32_t HAL_TMP_Adc2Temperature01(HAL_TMP_x TMP_x, int32_t adc_value)
00095 {
00096     (void) TMP_x;
00097     int32_t tmp;
00098     tmp = (int32_t) tmp_coef_a * (adc_value >> 4) / 1000 + (int32_t) tmp_coef_b;
00099     return tmp; // 返回温度值
00100 }

```

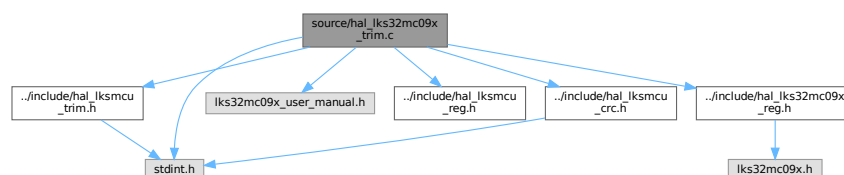
4.112 source/hal_lks32mc09x_trim.c 文件参考

```

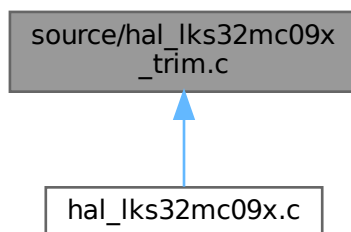
#include "../include/hal_lksmcu_trim.h"
#include "stdint.h"
#include "lks32mc09x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc09x_reg.h"
#include "../include/hal_lksmcu_crc.h"

```

hal_lks32mc09x_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_ReadMcu09](#) (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 )
```

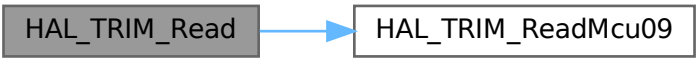
读取MCU03的校准参数

参数

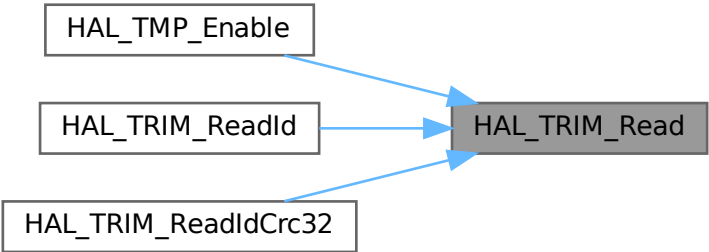
<i>addr</i>	校准参数的地址

在文件 [hal_lks32mc09x_trim.c](#) 第 92 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_lks32mc09x_trim.c](#) 第 102 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

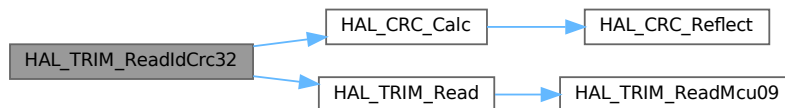
唯一ID的CRC32值

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_lks32mc09x_trim.c](#) 第 111 行定义.

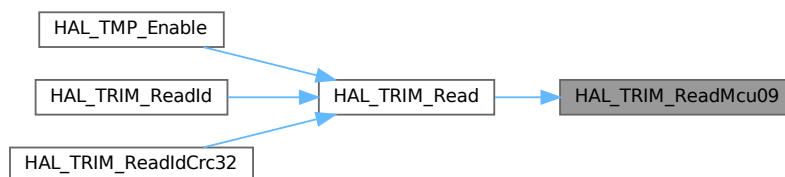
函数调用图:



4.112.1.4 HAL_TRIM_ReadMcu09()

```
uint32_t HAL_TRIM_ReadMcu09 (
    uint32_t )
```

这是这个函数的调用关系图:



4.113 hal_lks32mc09x_trim.c

[浏览该文件的文档.](#)

```
00001 #include "../include/hal_lksmcu_trim.h"
00002 #include "stdint.h"
00003 #include "lks32mc09x_user_manual.h"
00004 #include "../include/hal_lksmcu_reg.h"
00005 #include "../include/hal_lks32mc09x_reg.h"
00006 #include "../include/hal_lksmcu_crc.h"
00007 uint32_t HAL_TRIM_ReadMcu09(uint32_t);
00008
00009 // 地址 内容
00010 // 0x00001400 SYS_AFE_REG8 校准值, REF校准值
00011 // 0x00001404 SYS_AFE_REGA 校准值, LDO校准值
00012 // 0x00001408 SYS_AFE_REGB 校准值, RCL校准值
00013 // 0x0000140C SYS_AFE_REGC 校准值, RCH校准值, 5v供电
00014 // 0x00001410 SYS_AFE_REG9 校准值
00015 // 0x00001420 ADC0_DC0 校准值, 3.6V量程OFFSET
00016 // 0x00001424 ADC0_AMC0 校准值, 3.6V量程GAIN
00017 // 0x00001428 ADC0_DC1 校准值, 7.2V量程OFFSET
00018 // 0x0000142C ADC0_AMC1 校准值, 7.2V量程GAIN
00019 // 0x00001430 ADC1_DC0 校准值, 3.6V量程OFFSET
00020 // 0x00001434 ADC1_AMC0 校准值, 3.6V量程GAIN
00021 // 0x00001438 ADC1_DC1 校准值, 7.2V量程OFFSET
00022 // 0x0000143C ADC1_AMC1 校准值, 7.2V量程GAIN
00023 // 0x00001450 DAC0 选择4.8V 档位, SYS_AFE_DAC_AMC 校准值(扩大 512 倍结果)
00024 // 0x00001454 DAC0 选择4.8V 档位, SYS_AFE_DAC_DC 校准值
00025 // 0x00001458 DAC0 选择1.20V 档位, SYS_AFE_DAC_AMC 校准值(扩大 512 倍结果)
00026 // 0x0000145C DAC0 选择1.20V 档位, SYS_AFE_DAC_DC 校准值
00027 // 0x00001460 DAC1 选择3.0V 档位, SYS_AFE_DAC_AMC 校准值(扩大 512 倍结果)
00028 // 0x00001464 DAC1 选择3.0V 档位, SYS_AFE_DAC_DC 校准值
00029 // 0x00001468 DAC1 选择1.20V 档位, SYS_AFE_DAC_AMC 校准值(扩大 512 倍结果)
00030 // 0x0000146C DAC1 选择1.20V 档位, SYS_AFE_DAC_DC 校准值
00031 // 0x000014C0 OPA0, 共模电压(扩大10000 倍结果)
00032 // 0x000014C4 OPA1, 共模电压(扩大10000 倍结果)
00033 // 0x000014C8 OPA2, 共模电压(扩大10000 倍结果)
00034 // 0x000014CC OPA3, 共模电压(扩大10000 倍结果)
00035 // 0x00001480 OPA0, 660K 欧姆 VS 20K 欧姆, GAIN 校准值(扩大 1000 倍结果)
00036 // 0x00001484 OPA0, 640K 欧姆 VS 40K 欧姆, GAIN 校准值(扩大 1000 倍结果)
00037 // 0x00001488 OPA0, 320K 欧姆 VS 40K 欧姆, GAIN 校准值(扩大 1000 倍结果)
00038 // 0x0000148C OPA0, 160K 欧姆 VS 40K 欧姆, GAIN 校准值(扩大 1000 倍结果)
00039 // 0x00001490 OPA1, 660K 欧姆 VS 20K 欧姆, GAIN 校准值(扩大 1000 倍结果)
00040 // 0x00001494 OPA1, 640K 欧姆 VS 40K 欧姆, GAIN 校准值(扩大 1000 倍结果)
00041 // 0x00001498 OPA1, 320K 欧姆 VS 40K 欧姆, GAIN 校准值(扩大 1000 倍结果)
00042 // 0x0000149C OPA1, 160K 欧姆 VS 40K 欧姆, GAIN 校准值(扩大 1000 倍结果)
00043 // 0x000014A0 OPA2, 660K 欧姆 VS 20K 欧姆, GAIN 校准值(扩大 1000 倍结果)
00044 // 0x000014A4 OPA2, 640K 欧姆 VS 40K 欧姆, GAIN 校准值(扩大 1000 倍结果)
00045 // 0x000014A8 OPA2, 320K 欧姆 VS 40K 欧姆, GAIN 校准值(扩大 1000 倍结果)
00046 // 0x000014AC OPA2, 160K 欧姆 VS 40K 欧姆, GAIN 校准值(扩大 1000 倍结果)
00047 // 0x000014B0 OPA3, 660K 欧姆 VS 20K 欧姆, GAIN 校准值(扩大 1000 倍结果)
00048 // 0x000014B4 OPA3, 640K 欧姆 VS 40K 欧姆, GAIN 校准值(扩大 1000 倍结果)
00049 // 0x000014B8 OPA3, 320K 欧姆 VS 40K 欧姆, GAIN 校准值(扩大 1000 倍结果)
00050 // 0x000014BC OPA3, 160K 欧姆 VS 40K 欧姆, GAIN 校准值(扩大 1000 倍结果)
00051 // 0x00001500 OPA0, 660K 欧姆:20K 欧姆, 电阻校准值。高16位R2校准值(注意: 扩大10 倍结果) 低16位R1校准值(注意: 扩大10 倍结果)
00052 // 0x00001504 OPA0, 640K 欧姆:40K 欧姆, 电阻校准值。高16位R2校准值(注意: 扩大10 倍结果) 低16位R1校准值(注意: 扩大10 倍结果)
00053 // 0x00001508 OPA0, 320K 欧姆:40K 欧姆, 电阻校准值。高16位R2校准值(注意: 扩大10 倍结果) 低16位R1校准值(注意: 扩大10 倍结果)
00054 // 0x0000150C OPA0, 160K 欧姆:40K 欧姆, 电阻校准值。高16位R2校准值(注意: 扩大10 倍结果) 低16位R1校准值(注意: 扩大10 倍结果)
00055 // 0x00001510 OPA1, 660K 欧姆:20K 欧姆, 电阻校准值。高16位R2校准值(注意: 扩大10 倍结果) 低16位R1校准值(注意: 扩大10 倍结果)
00056 // 0x00001514 OPA1, 640K 欧姆:40K 欧姆, 电阻校准值。高16位R2校准值(注意: 扩大10 倍结果) 低16位R1校准值(注意: 扩大10 倍结果)
00057 // 0x00001518 OPA1, 320K 欧姆:40K 欧姆, 电阻校准值。高16位R2校准值(注意: 扩大10 倍结果) 低16位R1校准值(注意: 扩大10 倍结果)
00058 // 0x0000151C OPA1, 160K 欧姆:40K 欧姆, 电阻校准值。高16位R2校准值(注意: 扩大10 倍结果) 低16位R1校准值(注意: 扩大10 倍结果)
00059 // 0x00001520 OPA2, 660K 欧姆:20K 欧姆, 电阻校准值。高16位R2校准值(注意: 扩大10 倍结果) 低16位R1校准值(注意: 扩大10 倍结果)
00060 // 0x00001524 OPA2, 640K 欧姆:40K 欧姆, 电阻校准值。高16位R2校准值(注意: 扩大10 倍结果) 低16位R1校准值(注意: 扩大10 倍结果)
00061 // 0x00001528 OPA2, 320K 欧姆:40K 欧姆, 电阻校准值。高16位R2校准值(注意: 扩大10 倍结果) 低16位R1校准值(注意: 扩大10 倍结果)
00062 // 0x0000152C OPA2, 160K 欧姆:40K 欧姆, 电阻校准值。高16位R2校准值(注意: 扩大10 倍结果) 低16位R1校准值(注意: 扩大10 倍结果)
00063 // 0x00001530 OPA3, 660K 欧姆:20K 欧姆, 电阻校准值。高16位R2校准值(注意: 扩大10 倍结果) 低16位R1校准值(注意: 扩大10 倍结果)
00064 // 0x00001534 OPA3, 640K 欧姆:40K 欧姆, 电阻校准值。高16位R2校准值(注意: 扩大10 倍结果) 低16位R1校准值(注意: 扩大10 倍结果)
00065 // 0x00001538 OPA3, 320K 欧姆:40K 欧姆, 电阻校准值。高16位R2校准值(注意: 扩大10 倍结果) 低16位R1校准值(注意: 扩大10 倍结果)
00066 // 0x0000153C OPA3, 160K 欧姆:40K 欧姆, 电阻校准值。高16位R2校准值(注意: 扩大10 倍结果) 低16位R1校准值(注意: 扩大10 倍结果)
```



```

00067 // 0x00001560 OPA0 四个档位下Offset值(有符号类型,单位v,放大1000倍)Bit[31:16]4倍放大倍数 3档Bit[15:0]8倍放大倍数
      2档
00068 // 0x00001564 OPA0 四个档位下Offset值(有符号类型,单位v,放大1000倍)Bit[31:16]16倍放大倍数 1档Bit[15:0]33倍放大倍
      数 0档
00069 // 0x00001568 OPA1 四个档位下Offset值(有符号类型,单位v,放大1000倍)Bit[31:16]4倍放大倍数Bit[15:0]8倍放大倍数
00070 // 0x0000156C OPA1 四个档位下Offset值(有符号类型,单位v,放大1000倍)Bit[31:16]16倍放大倍数Bit[15:0]33倍放大倍数
00071 // 0x00001570 OPA2 四个档位下Offset值(有符号类型,单位v,放大1000倍)Bit[31:16]4倍放大倍数 3档Bit[15:0]8倍放大倍数
      2档
00072 // 0x00001574 OPA2 四个档位下Offset值(有符号类型,单位v,放大1000倍)Bit[31:16]16倍放大倍数 1档Bit[15:0]33倍放大倍
      数 0档
00073 // 0x00001578 OPA3 四个档位下Offset值(有符号类型,单位v,放大1000倍)Bit[31:16]4倍放大倍数Bit[15:0]8倍放大倍数
00074 // 0x0000157C OPA3 四个档位下Offset值(有符号类型,单位v,放大1000倍)Bit[31:16]16倍放大倍数Bit[15:0]33倍放大倍数
00075 // 0x000014D0 温度传感器,斜率校准值
00076 // 0x000014D4 温度传感器,偏置校准值
00077 // 0x000014D8 测试温度
00078 // 0x000014E0 芯片AMTE 测试次数
00079 // 0x000014E4 AMTE 测试程序版本号
00080 // 0x000014E8 LKS07X 芯片具体型号,070/1/2/3/5/6/....
00081 // 0x000014EC Boot 程序版本号
00082 // 0x000014F0 Flash大小标志,0:128k/1:64k/2:32k
00083 // 0x00001e84 ID0
00084 // 0x00001e88 ID1
00085 // 0x00001e8c ID2
00086 // 0x00001e90 ID3
00087 // 0x00001e94 ID4
00092 uint32_t HAL_TRIM_Read(uint32_t addr)
00093 {
00094     return HAL_TRIM_ReadMcu09(addr);
00095 }
00096
00102 uint32_t HAL_TRIM_ReadId(uint32_t idx)
00103 {
00104     return HAL_TRIM_Read(0x00001e84 + idx * 4);
00105 }
00111 uint32_t HAL_TRIM_ReadIdCrc32(void)
00112 {
00113     uint32_t id[5];
00114     for (int i = 0; i < 5; i++)
00115     {
00116         id[i] = HAL_TRIM_Read(0x00001e84 + i * 4);
00117     }
00118     return HAL_CRC_Calc(HAL_CRC_0, HAL_CRC_MODEL_CRC32, (uint8_t *)id, sizeof(id));
00119 }

```

4.114 source/hal_lks32mc09x_uart.c 文件参考

```

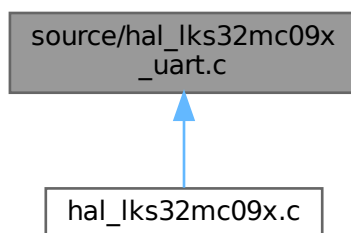
#include "../include/hal_lksmcu_clk.h"
#include "../include/hal_lksmcu_uart.h"
#include "../hal_lks32mc09x.h"
#include "stdint.h"
#include "lks32mc09x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc09x_reg.h"

```

hal_lks32mc09x_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_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)
- 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)
- uint8_t [HAL_UART_IrqSendOver](#) (HAL_UART_x uart_x)
- void [HAL_UART_IrqSendOverEnable](#) (HAL_UART_x uart_x)
- void [HAL_UART_IrqSendOverDisable](#) (HAL_UART_x uart_x)
- void [HAL_UART_IrqSendOverClear](#) (HAL_UART_x uart_x)
- uint8_t [HAL_UART_IrqReceiveOver](#) (HAL_UART_x uart_x)
- void [HAL_UART_IrqReceiveOverEnable](#) (HAL_UART_x uart_x)
- void [HAL_UART_IrqReceiveOverDisable](#) (HAL_UART_x uart_x)
- void [HAL_UART_IrqReceiveOverClear](#) (HAL_UART_x uart_x)
- uint8_t [HAL_UART_IrqStopErr](#) (HAL_UART_x uart_x)
- void [HAL_UART_IrqStopErrEnable](#) (HAL_UART_x uart_x)
- void [HAL_UART_IrqStopErrDisable](#) (HAL_UART_x uart_x)
- void [HAL_UART_IrqStopErrClear](#) (HAL_UART_x uart_x)
- uint8_t [HAL_UART_IrqCheckErr](#) (HAL_UART_x uart_x)
- void [HAL_UART_IrqCheckErrEnable](#) (HAL_UART_x uart_x)
- void [HAL_UART_IrqCheckErrDisable](#) (HAL_UART_x uart_x)
- void [HAL_UART_IrqCheckErrClear](#) (HAL_UART_x uart_x)

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_lks32mc09x_uart.c](#) 第 80 行定义。

4.114.1.2 HAL_UART_Enable()

```
void HAL_UART_Enable (
    HAL_UART_x UART_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 47 行定义.

4.114.1.3 HAL_UART_IrqCheckErr()

```
uint8_t HAL_UART_IrqCheckErr (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 316 行定义.

4.114.1.4 HAL_UART_IrqCheckErrClear()

```
void HAL_UART_IrqCheckErrClear (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 335 行定义.

4.114.1.5 HAL_UART_IrqCheckErrDisable()

```
void HAL_UART_IrqCheckErrDisable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 331 行定义.

4.114.1.6 HAL_UART_IrqCheckErrEnable()

```
void HAL_UART_IrqCheckErrEnable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 327 行定义.

4.114.1.7 HAL_UART_IrqReceiveOver()

```
uint8_t HAL_UART_IrqReceiveOver (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 266 行定义.

4.114.1.8 HAL_UART_IrqReceiveOverClear()

```
void HAL_UART_IrqReceiveOverClear (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 285 行定义.

4.114.1.9 HAL_UART_IrqReceiveOverDisable()

```
void HAL_UART_IrqReceiveOverDisable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 281 行定义.

4.114.1.10 HAL_UART_IrqReceiveOverEnable()

```
void HAL_UART_IrqReceiveOverEnable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 277 行定义.

4.114.1.11 HAL_UART_IrqSendOver()

```
uint8_t HAL_UART_IrqSendOver (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 241 行定义.

4.114.1.12 HAL_UART_IrqSendOverClear()

```
void HAL_UART_IrqSendOverClear (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 260 行定义.

4.114.1.13 HAL_UART_IrqSendOverDisable()

```
void HAL_UART_IrqSendOverDisable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 256 行定义.

4.114.1.14 HAL_UART_IrqSendOverEnable()

```
void HAL_UART_IrqSendOverEnable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 252 行定义.

4.114.1.15 HAL_UART_IrqStopErr()

```
uint8_t HAL_UART_IrqStopErr (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 291 行定义.

4.114.1.16 HAL_UART_IrqStopErrClear()

```
void HAL_UART_IrqStopErrClear (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 310 行定义.

4.114.1.17 HAL_UART_IrqStopErrDisable()

```
void HAL_UART_IrqStopErrDisable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 306 行定义.

4.114.1.18 HAL_UART_IrqStopErrEnable()

```
void HAL_UART_IrqStopErrEnable (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 302 行定义.

4.114.1.19 HAL_UART_ReadBuffer()

```
uint8_t HAL_UART_ReadBuffer (
    HAL_UART_x uart_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 219 行定义.

4.114.1.20 HAL_UART_Reset()

```
void HAL_UART_Reset (
    HAL_UART_x UART_x )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 11 行定义.

4.114.1.21 HAL_UART_SetBaud()

```
void HAL_UART_SetBaud (
    HAL_UART_x UART_x,
    uint32_t baud )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 112 行定义.

函数调用图:

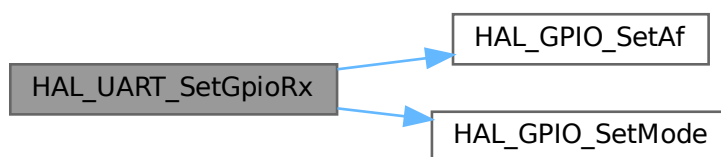


4.114.1.22 HAL_UART_SetGpioRx()

```
void HAL_UART_SetGpioRx (
    HAL_UART_x uart_x,
    HAL_GPIO_Px px )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 233 行定义.

函数调用图:

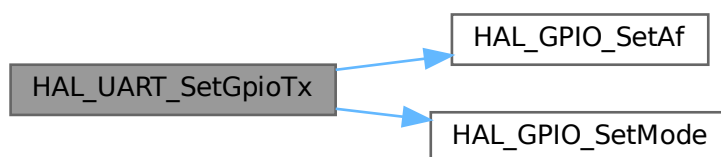


4.114.1.23 HAL_UART_SetGpioTx()

```
void HAL_UART_SetGpioTx (
    HAL_UART_x uart_x,
    HAL_GPIO_Px px )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 227 行定义.

函数调用图:



4.114.1.24 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_lks32mc09x_uart.c](#) 第 148 行定义.

4.114.1.25 HAL_UART_TxData()

```
void HAL_UART_TxData (
    HAL_UART_x UART_x,
    uint8_t * data,
    uint32_t len )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 136 行定义.

4.114.1.26 HAL_UART_WriteBuffer()

```
void HAL_UART_WriteBuffer (
    HAL_UART_x uart_x,
    uint8_t buff )
```

在文件 [hal_lks32mc09x_uart.c](#) 第 223 行定义.

4.115 hal_lks32mc09x_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_lks32mc09x.h"
00004 #include "stdint.h"
00005 #include "lks32mc09x_user_manual.h"
00006 #include "../include/hal_lksmcu_reg.h"
00007 #include "../include/hal_lks32mc09x_reg.h"
00008
00009 // 初始化
00010 // 模块复位
00011 void HAL_UART_Reset (HAL_UART_x UART_x)
00012 {
00013     switch (UART_x)
00014     {
00015         case HAL_UART_0:
00016             // 系统寄存器解锁
00017             HAL_REG_SYS_UNLOCK();
00018             HAL_REG_SET(SYS_SFT_RST, SYS_SFT_RST_UART0_SFT_RST_BIT);
00019             HAL_REG_RESET(SYS_SFT_RST, SYS_SFT_RST_UART0_SFT_RST_BIT);
00020             // 系统寄存器上锁
00021             HAL_REG_SYS_LOCK();
00022             break;
00023         case HAL_UART_1:
00024             // 系统寄存器解锁
00025             HAL_REG_SYS_UNLOCK();
00026             HAL_REG_SET(SYS_SFT_RST, SYS_SFT_RST_UART1_SFT_RST_BIT);
00027             HAL_REG_RESET(SYS_SFT_RST, SYS_SFT_RST_UART1_SFT_RST_BIT);
00028             // 系统寄存器上锁
00029             HAL_REG_SYS_LOCK();
00030             break;
00031         case HAL_UART_2:
00032             // 系统寄存器解锁
00033             HAL_REG_SYS_UNLOCK();
00034             HAL_REG_SET(SYS_SFT_RST, SYS_SFT_RST_UART2_SFT_RST_BIT);
00035             HAL_REG_RESET(SYS_SFT_RST, SYS_SFT_RST_UART2_SFT_RST_BIT);
00036             // 系统寄存器上锁
00037             HAL_REG_SYS_LOCK();
00038             break;
00039         default:
00040             break;
00041     }
00042 }
00043 // 模块打开
00044 void HAL_UART_Enable (HAL_UART_x UART_x)
00045 {
00046     switch (UART_x)
00047     {
00048         case HAL_UART_0:
00049             // 系统寄存器解锁
00050             HAL_REG_SYS_UNLOCK();
00051             HAL_REG_SET(SYS_CLK_FEN, SYS_CLK_FEN_UART0_CLK_EN_BIT);
00052             // 系统寄存器上锁
00053             HAL_REG_SYS_LOCK();
00054             break;
00055         case HAL_UART_1:
00056             // 系统寄存器解锁
00057             HAL_REG_SYS_UNLOCK();
00058             HAL_REG_SET(SYS_CLK_FEN, SYS_CLK_FEN_UART1_CLK_EN_BIT);
00059             // 系统寄存器上锁
00060             HAL_REG_SYS_LOCK();
00061             break;
00062         case HAL_UART_2:
00063             // 系统寄存器解锁
00064             HAL_REG_SYS_UNLOCK();
00065             HAL_REG_SET(SYS_CLK_FEN, SYS_CLK_FEN_UART2_CLK_EN_BIT);
00066             // 系统寄存器上锁
00067             HAL_REG_SYS_LOCK();
00068             break;
00069         default:
00070             break;
00071     }
00072 }
00073 // 模块关闭
00074 void HAL_UART_Disable (HAL_UART_x UART_x)
00075 {
00076     switch (UART_x)

```



```

00083     {
00084         case HAL_UART_0:
00085             // 系统寄存器解锁
00086             HAL_REG_SYS_UNLOCK();
00087             HAL_REG_RESET(SYS_CLK_FEN, SYS_CLK_FEN_UART0_CLK_EN_BIT);
00088             // 系统寄存器上锁
00089             HAL_REG_SYS_LOCK();
00090             break;
00091         case HAL_UART_1:
00092             // 系统寄存器解锁
00093             HAL_REG_SYS_UNLOCK();
00094             HAL_REG_RESET(SYS_CLK_FEN, SYS_CLK_FEN_UART1_CLK_EN_BIT);
00095             // 系统寄存器上锁
00096             HAL_REG_SYS_LOCK();
00097             break;
00098         case HAL_UART_2:
00099             // 系统寄存器解锁
00100             HAL_REG_SYS_UNLOCK();
00101             HAL_REG_RESET(SYS_CLK_FEN, SYS_CLK_FEN_UART2_CLK_EN_BIT);
00102             // 系统寄存器上锁
00103             HAL_REG_SYS_LOCK();
00104             break;
00105         default:
00106             break;
00107     }
00108 }
00109
00110 void HAL_UART_SetBaud(HAL_UART_x UART_x, uint32_t baud)
00111 {
00112     switch (UART_x)
00113     {
00114         case HAL_UART_0:
00115         {
00116             uint32_t mclk = HAL_CLK_GetMcuFreq();
00117             uint32_t div = mclk / baud;
00118             uint32_t div2 = 0;
00119             while (div & 0xffff0000)
00120             {
00121                 div2++;
00122                 div >>= 1;
00123             }
00124             HAL_REG_SYS_UNLOCK();
00125             HAL_REG_WRITE(SYS_CLK_DIV2, div2);
00126             HAL_REG_SYS_LOCK();
00127             HAL_REG_WRITE(HAL_UART[UART_x]->DIVH, div >> 8);
00128             HAL_REG_WRITE(HAL_UART[UART_x]->DIVL, div & 0xff);
00129         }
00130         default:
00131             break;
00132     }
00133 }
00134
00135 void HAL_UART_TxData(HAL_UART_x UART_x, uint8_t *data, uint32_t len)
00136 {
00137     for (; len > 0; len--)
00138     {
00139         HAL_REG_WRITE(HAL_UART[UART_x]->BUFF, *data);
00140         data++;
00141         while (HAL_REG_READ_BIT(HAL_UART[UART_x]->STT, BIT0) == 0)
00142         {
00143         }
00144     }
00145 }
00146
00147 void HAL_UART_SetMode(HAL_UART_x uart_x, HAL_UART_MODE_x mode_x)
00148 {
00149     uint32_t mask = (UARTx_CTRLCK_EN_BIT |
00150                     UARTx_CTRLCK_TYPE_BIT |
00151                     UARTx_CTRLBIT_ORDER_BIT |
00152                     UARTx_CTRLSTOP_LEN_BIT |
00153                     UARTx_CTRLBYTE_LEN_BIT);
00154     switch (mode_x)
00155     {
00156         case HAL_UART_MODE_LSB_STOP1_NONE:
00157         {
00158             HAL_REG_WRITE_BIT(HAL_UART[uart_x]->CTRL, mask, 0);
00159             break;
00160         }
00161         case HAL_UART_MODE_LSB_STOP2_NONE:
00162         {
00163             HAL_REG_WRITE_BIT(HAL_UART[uart_x]->CTRL, mask, UARTx_CTRLSTOP_LEN_BIT);
00164             break;
00165         }
00166         case HAL_UART_MODE_LSB_STOP1_ODD:
00167         {
00168             HAL_REG_WRITE_BIT(HAL_UART[uart_x]->CTRL, mask, UARTx_CTRLCK_EN_BIT | UARTx_CTRLCK_TYPE_BIT |

```

```

    UARTx->CTRL_BYTE_LEN_BIT);
00170         break;
00171     }
00172     case HAL_UART_MODE_LSB_STOP2_ODD:
00173     {
00174         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_BYTE_LEN_BIT);
00175         break;
00176     }
00177     case HAL_UART_MODE_LSB_STOP1_EVEN:
00178     {
00179         HAL_REG_WRITE_BIT(HAL_UART[uart_x] -> CTRL, mask, UARTx->CTRL_CK_EN_BIT |
UARTx->CTRL_BYTE_LEN_BIT);
00180         break;
00181     }
00182     case HAL_UART_MODE_LSB_STOP2_EVEN:
00183     {
00184         HAL_REG_WRITE_BIT(HAL_UART[uart_x] -> CTRL, mask, UARTx->CTRL_CK_EN_BIT | UARTx->CTRL_STOP_LEN_BIT
| UARTx->CTRL_BYTE_LEN_BIT);
00185         break;
00186     }
00187     case HAL_UART_MODE_MSB_STOP1_NONE:
00188     {
00189         HAL_REG_WRITE_BIT(HAL_UART[uart_x] -> CTRL, mask, UARTx->CTRL_BIT_ORDER_BIT);
00190         break;
00191     }
00192     case HAL_UART_MODE_MSB_STOP2_NONE:
00193     {
00194         HAL_REG_WRITE_BIT(HAL_UART[uart_x] -> CTRL, mask, UARTx->CTRL_BIT_ORDER_BIT |
UARTx->CTRL_STOP_LEN_BIT);
00195         break;
00196     }
00197     case HAL_UART_MODE_MSB_STOP1_ODD:
00198     {
00199         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_BYTE_LEN_BIT);
00200         break;
00201     }
00202     case HAL_UART_MODE_MSB_STOP2_ODD:
00203     {
00204         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_BYTE_LEN_BIT);
00205         break;
00206     }
00207     case HAL_UART_MODE_MSB_STOP1_EVEN:
00208     {
00209         HAL_REG_WRITE_BIT(HAL_UART[uart_x] -> CTRL, mask, UARTx->CTRL_CK_EN_BIT | UARTx->CTRL_BIT_ORDER_BIT
| UARTx->CTRL_BYTE_LEN_BIT);
00210         break;
00211     }
00212     case HAL_UART_MODE_MSB_STOP2_EVEN:
00213     {
00214         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_BYTE_LEN_BIT);
00215         break;
00216     }
00217     }
00218 }
00219 uint8_t HAL_UART_ReadBuffer(HAL_UART_x uart_x)
00220 {
00221     return HAL_REG_READ(HAL_UART[uart_x] -> BUFF);
00222 }
00223 void HAL_UART_WriteBuffer(HAL_UART_x uart_x, uint8_t buff)
00224 {
00225     HAL_REG_WRITE(HAL_UART[uart_x] -> BUFF, buff);
00226 }
00227 void HAL_UART_SetGpioTx(HAL_UART_x uart_x, HAL_GPIO_Px px)
00228 {
00229     (void)uart_x;
00230     HAL_GPIO_SetAf(px, HAL_GPIO_AF_UART);
00231     HAL_GPIO_SetMode(px, HAL_GPIO_MODE_OUT);
00232 }
00233 void HAL_UART_SetGpioRx(HAL_UART_x uart_x, HAL_GPIO_Px px)
00234 {
00235     (void)uart_x;
00236     HAL_GPIO_SetAf(px, HAL_GPIO_AF_UART);
00237     HAL_GPIO_SetMode(px, HAL_GPIO_MODE_IN);
00238 }
00239
00240 // uart发送完成中断
00241 uint8_t HAL_UART_IrqSendOver(HAL_UART_x uart_x)
00242 {
00243     if (HAL_UART[uart_x] -> IE & HAL_UART[uart_x] -> IF & BIT0)
00244     {
00245         return 1;
00246     }
00247     else

```

```

00248     {
00249         return 0;
00250     }
00251 }
00252 void HAL_UART_IrqSendOverEnable(HAL_UART_x uart_x)
00253 {
00254     HAL_REG_SET(HAL_UART[uart_x]->IE, BIT0);
00255 }
00256 void HAL_UART_IrqSendOverDisable(HAL_UART_x uart_x)
00257 {
00258     HAL_REG_RESET(HAL_UART[uart_x]->IE, BIT0);
00259 }
00260 void HAL_UART_IrqSendOverClear(HAL_UART_x uart_x)
00261 {
00262     HAL_REG_WRITE(HAL_UART[uart_x]->IF, BIT0);
00263 }
00264
00265 // uart接受完成中断
00266 uint8_t HAL_UART_IrqReceiveOver(HAL_UART_x uart_x)
00267 {
00268     if (HAL_UART[uart_x]->IE & HAL_UART[uart_x]->IF & BIT1)
00269     {
00270         return 1;
00271     }
00272     else
00273     {
00274         return 0;
00275     }
00276 }
00277 void HAL_UART_IrqReceiveOverEnable(HAL_UART_x uart_x)
00278 {
00279     HAL_REG_SET(HAL_UART[uart_x]->IE, BIT1);
00280 }
00281 void HAL_UART_IrqReceiveOverDisable(HAL_UART_x uart_x)
00282 {
00283     HAL_REG_RESET(HAL_UART[uart_x]->IE, BIT1);
00284 }
00285 void HAL_UART_IrqReceiveOverClear(HAL_UART_x uart_x)
00286 {
00287     HAL_REG_WRITE(HAL_UART[uart_x]->IF, BIT1);
00288 }
00289
00290 // uart停止位错误中断
00291 uint8_t HAL_UART_IrqStopErr(HAL_UART_x uart_x)
00292 {
00293     if (HAL_UART[uart_x]->IE & HAL_UART[uart_x]->IF & BIT3)
00294     {
00295         return 1;
00296     }
00297     else
00298     {
00299         return 0;
00300     }
00301 }
00302 void HAL_UART_IrqStopErrEnable(HAL_UART_x uart_x)
00303 {
00304     HAL_REG_SET(HAL_UART[uart_x]->IE, BIT3);
00305 }
00306 void HAL_UART_IrqStopErrDisable(HAL_UART_x uart_x)
00307 {
00308     HAL_REG_RESET(HAL_UART[uart_x]->IE, BIT3);
00309 }
00310 void HAL_UART_IrqStopErrClear(HAL_UART_x uart_x)
00311 {
00312     HAL_REG_WRITE(HAL_UART[uart_x]->IF, BIT3);
00313 }
00314
00315 // uart校验错误中断
00316 uint8_t HAL_UART_IrqCheckErr(HAL_UART_x uart_x)
00317 {
00318     if (HAL_UART[uart_x]->IE & HAL_UART[uart_x]->IF & BIT4)
00319     {
00320         return 1;
00321     }
00322     else
00323     {
00324         return 0;
00325     }
00326 }
00327 void HAL_UART_IrqCheckErrEnable(HAL_UART_x uart_x)
00328 {
00329     HAL_REG_SET(HAL_UART[uart_x]->IE, BIT4);
00330 }
00331 void HAL_UART_IrqCheckErrDisable(HAL_UART_x uart_x)
00332 {
00333     HAL_REG_RESET(HAL_UART[uart_x]->IE, BIT4);
00334 }

```

```

00335 void HAL_UART_IrqCheckErrClear(HAL_UART_x uart_x)
00336 {
00337     HAL_REG_WRITE(HAL_UART[uart_x] -> IF, BIT4);
00338 }

```

4.116 source/hal_lks32mc09x_wake.c 文件参考

```

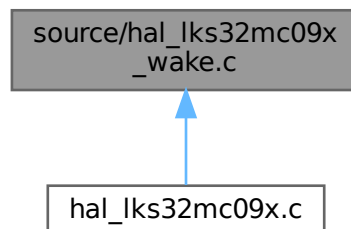
#include "../include/hal_lksmcu_iwdg.h"
#include "../include/hal_lksmcu_wake.h"
#include "../hal_lks32mc09x.h"
#include "lks32mc09x_user_manual.h"
#include "../include/hal_lksmcu_reg.h"
#include "../include/hal_lks32mc09x_reg.h"

```

hal_lks32mc09x_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 )
```

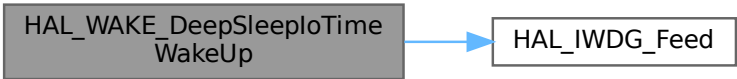
进入深度休眠,并且使用IO唤醒和定时唤醒

参数

<i>wake_up</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_lks32mc09x.wake.c](#) 第 147 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</i> ↔ <i>_io</i>	指定唤醒IO引脚及其电平

在文件 [hal_lks32mc09x.wake.c](#) 第 12 行定义.

4.116.1.3 HAL_WAKE_DeepSleepTimeWakeUp()

```
void HAL_WAKE_DeepSleepTimeWakeUp (
    uint32_t ms )
```

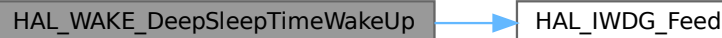
进入深度休眠,并且使用定时唤醒

参数

<i>ms</i>	指定定时唤醒的时间,单位为毫秒

在文件 [hal_lks32mc09x.wake.c](#) 第 84 行定义.

函数调用图:



4.117 hal_lks32mc09x_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_lks32mc09x.h"
00004 #include "lks32mc09x_user_manual.h"
00005 #include "../include/hal_lksmcu.reg.h"
00006 #include "../include/hal_lks32mc09x.reg.h"
00007
00012 void HAL_WAKE_DeepSleepIoWake(HAL_WAKE_UP_IO_x wake_up_io)
00013 {
00014     u32 t_cnt;
00015     uint32_t reg0; // 模拟配置
00016     uint32_t reg1; // 模拟配置
00017     uint32_t reg2; // 模拟配置
00018     uint32_t reg5; // 模拟配置
00019     uint32_t reg7; // 模拟配置
00020     uint32_t clkcfg; // 时钟配置
00021     uint32_t clk_fen; // 模块使能配置
00022     uint32_t saved_enable; // 中断配置
00023
00024     HAL_REG_WRITE_BIT(AON.IO.WAKE_EN, 0xff, wake_up_io & 0xff);
00025     HAL_REG_WRITE_BIT(AON.IO.WAKE_POL, 0xff, (wake_up_io >> 8) & 0xff);
00026
00027     __disable_irq(); // 关闭全局中断, 确保执行休眠期间, 不会进入其他任何中断
00028     reg0 = SYS_AFE_REG0; // 保存模拟使能配置
00029     reg1 = SYS_AFE_REG1; // 保存模拟使能配置
00030     reg2 = SYS_AFE_REG2; // 保存模拟使能配置
00031     reg5 = SYS_AFE_REG5; // 保存模拟使能配置
00032     reg7 = SYS_AFE_REG7; // 保存模拟使能配置
00033     clkcfg = SYS_CLK_CFG; // 保存时钟配置
00034     clk_fen = SYS_CLK_FEN; // 保存模块时钟使能配置
00035     saved_enable = NVIC->ISER[0]; // 保存当前中断配置
00036
00037     SYS_WR_PROTECT = 0x7a83; // 开启寄存器写使能
00038     NVIC->ICER[0] = 0xffffffff; // 关闭所有中断
00039     NVIC_EnableIRQ(WAKE_IRQn); /* 使能WAKE中断 */
00040     NVIC_SetPriority(WAKE_IRQn, 0); /* 配置WAKE中断优先级 */
00041     IWDG_PSW = 0xA6B4; // 打开看门狗使能
00042     IWDG_CLR = 0x798D; // 喂狗
00043     SYS_CLK_CFG = 0; // 切换到HRC时钟
00044     for (t_cnt = 0; t_cnt < 4000; t_cnt++) // 等待时钟稳定
00045     {
00046         __NOP();
00047     }
00048     SYS_CLK_FEN = 0; // 关闭数字模块时钟
00049     SYS_AFE_REG0 = 0; //
00050     SYS_AFE_REG1 = SYS_AFE_REG1_RCHPD_BIT | SYS_AFE_REG1_BGPPD_BIT; //
00051     SYS_AFE_REG2 = 0; //
00052     SYS_AFE_REG5 = 0; //
00053     SYS_AFE_REG7 = 0; //
00054     SYS_WR_PROTECT = 0x0; // 关闭系统寄存器写使能
00055     __enable_irq(); // 打开全局中断
00056     SCB->SCR |= (1UL << 2); // 使能深度休眠
00057
00058     __WFI(); // 休眠 wakeak
00059     __disable_irq(); // 唤醒后关闭全局中断
00060     NVIC_DisableIRQ(WAKE_IRQn); // 关闭唤醒中断
00061
00062     SYS_WR_PROTECT = 0x7a83; // 开启寄存器写使能
00063     SYS_AFE_REG0 = reg0; // 还原模拟使能配置
00064     SYS_AFE_REG1 = reg1; // 还原模拟使能配置
00065     SYS_AFE_REG2 = reg2; // 还原模拟使能配置
00066     SYS_AFE_REG5 = reg5; // 还原模拟使能配置
00067     SYS_AFE_REG7 = reg7; // 还原模拟使能配置
  
```

```

00068
00069     for (t_cnt = 0; t_cnt < 4000; t_cnt++) // 此时使用hrc, 等待>1ms, 确保PLL稳定
00070     {
00071         __NOP();
00072     }
00073     SYS_CLK_CFG = clkcfg; // 恢复时钟配置
00074     NVIC->ISER[0] = saved_enable; // 恢复中断使能
00075     SYS_CLK_FEN = clk_fen; // 打开其他模块
00076     SYS_WR_PROTECT = 0x0; // 打开写保护
00077     __enable_irq(); // 使能全局中断
00078 }
00079
00084 void HAL_WAKE_DeepSleepTimeWakeUp(uint32_t ms)
00085 {
00086     u32 t_cnt;
00087     uint32_t reg5; // 模拟配置
00088     uint32_t clkcfg; // 时钟配置
00089     uint32_t clk_fen; // 模块使能配置
00090     uint32_t saved_enable; // 中断配置
00091
00092     __disable_irq(); // 关闭全局中断, 确保执行休眠期间, 不会进入其他任何中断
00093     reg5 = SYS_AFE_REG5; // 保存模拟使能配置
00094     clkcfg = SYS_CLK_CFG; // 保存时钟配置
00095     clk_fen = SYS_CLK_FEN; // 保存模块时钟使能配置
00096     saved_enable = NVIC->ISER[0]; // 保存当前中断配置
00097
00098     SYS_WR_PROTECT = 0x7a83; // 开启寄存器写使能
00099     NVIC->ICER[0] = 0xffffffff; // 关闭所有中断
00100     NVIC_EnableIRQ(WAKE_IRQn); // 打开唤醒中断
00101     {
00102         int32_t th;
00103         th = ((uint32_t)(64 * ms + 0x800) & 0x1fff000);
00104         th = HAL_REG_READ(HAL_IWDG[HAL_IWDG_0]->RTH) - th;
00105         if (th < 0)
00106         {
00107             th = 0;
00108         }
00109         HAL_REG_WRITE(HAL_IWDG[HAL_IWDG_0]->WTH, th);
00110     }
00111     HAL_IWDG_Feed(HAL_IWDG_0); // 喂狗
00112     SYS_CLK_CFG = 0; // 切换到HRC时钟
00113     for (t_cnt = 0; t_cnt < 4000; t_cnt++) // 等待时钟稳定
00114     {
00115         __NOP();
00116     }
00117     SYS_CLK_FEN = 0; // 关闭数字模块时钟
00118     SYS_AFE_REG5 = BIT10 | BIT8; // 准备关闭 PLL, BGP, 以及其它模拟电路使能
00119     SYS_WR_PROTECT = 0x0; // 关闭系统寄存器写使能
00120
00121     __enable_irq(); // 打开全局中断
00122     SCB->SCR |= (1UL << 2); // 使能深度休眠
00123
00124     __WFI(); // 休眠 wakeak
00125     __disable_irq(); // 唤醒后关闭全局中断
00126     NVIC_DisableIRQ(WAKE_IRQn); // 关闭唤醒中断
00127
00128     SYS_WR_PROTECT = 0x7a83; // 开启寄存器写使能
00129     SYS_AFE_REG5 = reg5; // 开启 PLL, BGP
00130
00131     for (t_cnt = 0; t_cnt < 400; t_cnt++) // 此时使用hrc, 等待>1ms, 确保PLL稳定
00132     {
00133         __NOP();
00134     }
00135     SYS_CLK_CFG = clkcfg; // 恢复时钟配置
00136     NVIC->ISER[0] = saved_enable; // 恢复中断使能
00137     SYS_CLK_FEN = clk_fen; // 打开其他模块
00138     SYS_WR_PROTECT = 0x0; // 打开写保护
00139     __enable_irq(); // 使能全局中断
00140 }
00141
00147 void HAL_WAKE_DeepSleepIoTimeWakeUp(HAL_WAKE_UP_IO_x wake_up_io, uint32_t ms)
00148 {
00149     u32 t_cnt;
00150     uint32_t reg5; // 模拟配置
00151     uint32_t clkcfg; // 时钟配置
00152     uint32_t clk_fen; // 模块使能配置
00153     uint32_t saved_enable; // 中断配置
00154
00155     __disable_irq(); // 关闭全局中断, 确保执行休眠期间, 不会进入其他任何中断
00156     reg5 = SYS_AFE_REG5; // 保存模拟使能配置
00157     clkcfg = SYS_CLK_CFG; // 保存时钟配置
00158     clk_fen = SYS_CLK_FEN; // 保存模块时钟使能配置
00159     saved_enable = NVIC->ISER[0]; // 保存当前中断配置
00160
00161     SYS_WR_PROTECT = 0x7a83; // 开启寄存器写使能
00162     NVIC->ICER[0] = 0xffffffff; // 关闭所有中断
00163     NVIC_EnableIRQ(WAKE_IRQn); // 打开唤醒中断

```



```

00164     {
00165         int32_t th;
00166         th = ((uint32_t)(64 * ms + 0x800) & 0x1ff000);
00167         th = HAL_REG_READ(HAL_IWDG[HAL_IWDG_0]->RTH) - th;
00168         if (th < 0)
00169         {
00170             th = 0;
00171         }
00172         HAL_REG_WRITE(HAL_IWDG[HAL_IWDG_0]->WTH, th);
00173     }
00174     HAL_IWDG_Feed(HAL_IWDG_0); // 喂狗
00175     SYS_CLK_CFG = 0; // 切换到HRC时钟
00176     for (t_cnt = 0; t_cnt < 4000; t_cnt++) // 等待时钟稳定
00177     {
00178         __NOP();
00179     }
00180     SYS_CLK_FEN = 0; // 关闭数字模块时钟
00181     SYS_AFE_REG5 = BIT10 | BIT8; // 准备关闭 PLL,BGP,以及其它模拟电路使能
00182     SYS_WR_PROTECT = 0x0; // 关闭系统寄存器写使能
00183
00184     AON_IO_WAKE_POL = wake_up_io >> 8; // 设置IO唤醒电平
00185     AON_IO_WAKE_EN = wake_up_io & 0xFF; // 设置IO唤醒引脚
00186
00187     __enable_irq(); // 打开全局中断
00188     SCB->SCR |= (1UL << 2); // 使能深度休眠
00189
00190     __WFI(); // 休眠 wakeak
00191     __disable_irq(); // 唤醒后关闭全局中断
00192     NVIC_DisableIRQ(WAKE_IRQn); // 关闭唤醒中断
00193
00194     SYS_WR_PROTECT = 0x7a83; // 开启寄存器写使能
00195     SYS_AFE_REG5 = reg5; // 开启 PLL,BGP
00196
00197     for (t_cnt = 0; t_cnt < 400; t_cnt++) // 此时使用hrc, 等待>1mS, 确保PLL稳定
00198     {
00199         __NOP();
00200     }
00201     SYS_CLK_CFG = clkcfg; // 恢复时钟配置
00202     NVIC->ISER[0] = saved_enable; // 恢复中断使能
00203     SYS_CLK_FEN = clk_fen; // 打开其他模块
00204     SYS_WR_PROTECT = 0x0; // 打开写保护
00205     __enable_irq(); // 使能全局中断
00206 }

```


Index

__HAL_LKS32MC05X_C__
 hal_lks32mc09x.c, [76](#)

adc_soft_reg
 hal_lks32mc09x_adc.c, [290](#)

ADDR
 HAL_FLASH_TypeDef, [30](#)
 HAL_I2C_TypeDef, [40](#)

ADR
 HAL_UART_TypeDef, [72](#)

AFE_ADC
 HAL_SYS_TypeDef, [60](#)

AFE_DAC0
 HAL_SYS_TypeDef, [60](#)

AFE_DAC0_AMC
 HAL_SYS_TypeDef, [61](#)

AFE_DAC0_DC
 HAL_SYS_TypeDef, [61](#)

AFE_DAC1
 HAL_SYS_TypeDef, [61](#)

AFE_DAC1_AMC
 HAL_SYS_TypeDef, [61](#)

AFE_DAC1_DC
 HAL_SYS_TypeDef, [61](#)

AFE_DAC_CTRL
 HAL_SYS_TypeDef, [61](#)

AFE_DBG
 HAL_SYS_TypeDef, [62](#)

AFE_INFO
 HAL_SYS_TypeDef, [62](#)

AFE_REG0
 HAL_SYS_TypeDef, [62](#)

AFE_REG1
 HAL_SYS_TypeDef, [62](#)

AFE_REG2
 HAL_SYS_TypeDef, [62](#)

AFE_REG3
 HAL_SYS_TypeDef, [63](#)

AFE_REG4
 HAL_SYS_TypeDef, [63](#)

AFE_REG5
 HAL_SYS_TypeDef, [63](#)

AFE_REG7
 HAL_SYS_TypeDef, [63](#)

AMC0
 HAL_ADC_TypeDef, [9](#)

AMC1
 HAL_ADC_TypeDef, [9](#)

AON0
 hal_lks32mc09x_reg.h, [91](#)

AUEN
 HAL_MCPWM_TypeDef, [46](#)

BCR
 HAL_I2C_TypeDef, [40](#)

BKP_REG0
 HAL_AON_TypeDef, [14](#)

BKP_REG1
 HAL_AON_TypeDef, [14](#)

BLCWIN
 CMP_TypeDef, [5](#)
 HAL_CMP_TypeDef, [19](#)

BRR
 HAL_GPIO_TypeDef, [35](#)

BSIZE
 HAL_I2C_TypeDef, [41](#)

BSRR
 HAL_GPIO_TypeDef, [35](#)

BUFF
 HAL_UART_TypeDef, [72](#)

CCR
 HAL_DMA_TypeDef, [22](#)

CF
 HAL_CLU_TypeDef, [16](#)

CFG
 CMP_TypeDef, [5](#)
 HAL_ADC_TypeDef, [9](#)
 HAL_CMP_TypeDef, [19](#)
 HAL_FLASH_TypeDef, [30](#)
 HAL_HALL_TypeDef, [39](#)
 HAL_I2C_TypeDef, [41](#)
 HAL_IWDG_TypeDef, [42](#)
 HAL_QEP_TypeDef, [54](#)
 HAL_SIF_TypeDef, [55](#)
 HAL_SPI_TypeDef, [57](#)

CFG0
 HAL_TIMER_TypeDef, [69](#)

CFG1
 HAL_TIMER_TypeDef, [69](#)

CFG2
 HAL_TIMER_TypeDef, [69](#)

CH_DEF
 HAL_MCPWM_TypeDef, [46](#)

CH_FAIL
 HAL_MCPWM_TypeDef, [46](#)

CH_MSK
 HAL_MCPWM_TypeDef, [47](#)

CHN0
 HAL_ADC_TypeDef, [10](#)

- HAL_TIMER_TypeDef, 69
- CHN1
 - HAL_ADC_TypeDef, 10
 - HAL_TIMER_TypeDef, 69
- CHN2
 - HAL_ADC_TypeDef, 10
- CHNT
 - HAL_ADC_TypeDef, 10
- CLK_CFG
 - HAL_SYS_TypeDef, 63
- CLK_DIV0
 - HAL_SYS_TypeDef, 64
- CLK_DIV1
 - HAL_SYS_TypeDef, 64
- CLK_DIV2
 - HAL_SYS_TypeDef, 64
- CLK_FEN
 - HAL_SYS_TypeDef, 64
- CLKO
 - HAL_EXTI_TypeDef, 27
- CLR
 - HAL_IWDG_TypeDef, 42
- CLU0
 - hal_iks32mc09x_reg.h, 91
- CMP_TypeDef, 5
 - BLCWIN, 5
 - CFG, 5
 - DATA, 6
 - IE, 6
 - IF, 6
 - TCLK, 6
- CNT
 - HAL_HALL_TypeDef, 39
 - HAL_IWDG_TypeDef, 43
 - HAL_QEP_TypeDef, 54
 - HAL_TIMER_TypeDef, 70
- CNT0
 - HAL_MCPWM_TypeDef, 47
- CR
 - HAL_CRC_TypeDef, 20
- CR0
 - HAL_EXTI_TypeDef, 27
- CR1
 - HAL_EXTI_TypeDef, 27
- CRC_Model.Params, 6
 - initValue, 7
 - polynomial, 7
 - reflectIn, 7
 - reflectOut, 7
 - width, 7
 - xorOut, 7
- CRC_Width
 - hal_iks32mc09x_crc.c, 307
- CRC_WIDTH_16
 - hal_iks32mc09x_crc.c, 308
- CRC_WIDTH_32
 - hal_iks32mc09x_crc.c, 308
- CRC_WIDTH_8
- hal_iks32mc09x_crc.c, 308
- crcModels
 - hal_iks32mc09x_crc.c, 310
- CTMS
 - HAL_DMA_TypeDef, 22
- CTRL
 - HAL_DMAMODULE_TypeDef, 23
 - HAL_UART_TypeDef, 72
- DADR
 - HAL_DMA_TypeDef, 22
- DAT
 - HAL_ADC_TypeDef, 10
- DATA
 - CMP_TypeDef, 6
 - HAL_CMP_TypeDef, 19
 - HAL_I2C_TypeDef, 41
- DBG_CFG
 - HAL_SYS_TypeDef, 64
- DC0
 - HAL_ADC_TypeDef, 11
- DC1
 - HAL_ADC_TypeDef, 11
- DID
 - HAL_DSP_TypeDef, 24
- DID1
 - HAL_DSP_TypeDef, 24
- DIS
 - HAL_DSP_TypeDef, 25
- DIS1
 - HAL_DSP_TypeDef, 25
- DIV
 - HAL_SPI_TypeDef, 57
- DIVH
 - HAL_UART_TypeDef, 72
- DIVL
 - HAL_UART_TypeDef, 72
- DMA_CHN0
 - hal_iks32mc09x_reg.h, 91
- DMA_CHN1
 - hal_iks32mc09x_reg.h, 91
- DMA_CHN2
 - hal_iks32mc09x_reg.h, 91
- DMA_CHN3
 - hal_iks32mc09x_reg.h, 91
- DMA_TOP
 - hal_iks32mc09x_reg.h, 91
- DR
 - HAL_CRC_TypeDef, 20
- DTH00
 - HAL_MCPWM_TypeDef, 47
- DTH01
 - HAL_MCPWM_TypeDef, 47
- EIE
 - HAL_MCPWM_TypeDef, 47
- EIF
 - HAL_MCPWM_TypeDef, 47
- EN

HAL_CLU_TypeDef, 16
 EPSM
 HAL_FLASH_TypeDef, 30
 EPSN
 HAL_FLASH_TypeDef, 30
 ERASE
 HAL_FLASH_TypeDef, 31
 EVT
 HAL_TIMER_TypeDef, 70
 EVT0
 HAL_MCPWM_TypeDef, 48
 EVT_RCD
 HAL_AON_TypeDef, 14
 EXTI0
 hal_iks32mc09x_reg.h, 91

 F3210
 HAL_GPIO_TypeDef, 35
 F7654
 HAL_GPIO_TypeDef, 36
 FBA98
 HAL_GPIO_TypeDef, 36
 FCNT
 HAL_MCPWM_TypeDef, 48
 FFEDC
 HAL_GPIO_TypeDef, 36
 FLSE
 HAL_SYS_TypeDef, 64
 FLSP
 HAL_SYS_TypeDef, 65
 FLT
 HAL_MCPWM_TypeDef, 48
 HAL_TIMER_TypeDef, 70
 FN
 HAL_CLU_TypeDef, 17

 GAIN
 HAL_ADC_TypeDef, 11
 GEN
 HAL_ADC_TypeDef, 11

 HAL_ADC
 hal_iks32mc09x_reg.c, 419
 hal_iks32mc09x_reg.h, 92
 HAL_ADC_0
 hal_iksmcu_adc.h, 103
 HAL_ADC_1
 hal_iksmcu_adc.h, 103
 HAL_ADC_2
 hal_iksmcu_adc.h, 104
 HAL_ADC_3
 hal_iksmcu_adc.h, 104
 HAL_ADC_Disable
 hal_iks32mc09x_adc.c, 280
 hal_iksmcu_adc.h, 104
 HAL_ADC_Enable
 hal_iks32mc09x_adc.c, 281
 hal_iksmcu_adc.h, 104
 HAL_ADC_EndSampling
 hal_iks32mc09x_adc.c, 281
 hal_iksmcu_adc.h, 104
 HAL_ADC_GetRange
 hal_iks32mc09x_adc.c, 281
 hal_iksmcu_adc.h, 105
 HAL_ADC_GetSamplingValues
 hal_iks32mc09x_adc.c, 282
 hal_iksmcu_adc.h, 105
 HAL_ADC_GetValuesUnit
 hal_iks32mc09x_adc.c, 283
 hal_iksmcu_adc.h, 106
 HAL_ADC_IRQHandler
 hal_iks32mc09x.h, 78
 HAL_ADC_IRQHandler_00
 hal_iks32mc09x.h, 78
 HAL_ADC_IRQHandler_10
 hal_iks32mc09x.h, 79
 HAL_ADC_IRQHandler_20
 hal_iks32mc09x.h, 79
 HAL_ADC_IrqSamp1
 hal_iks32mc09x_irqfun.c, 373
 hal_iksmcu_irqfun.h, 189
 HAL_ADC_IrqSamp1Clear
 hal_iks32mc09x_irqfun.c, 374
 hal_iksmcu_irqfun.h, 190
 HAL_ADC_IrqSamp1Disable
 hal_iks32mc09x_irqfun.c, 374
 hal_iksmcu_irqfun.h, 190
 HAL_ADC_IrqSamp1Enable
 hal_iks32mc09x_irqfun.c, 374
 hal_iksmcu_irqfun.h, 190
 HAL_ADC_IrqSamp1Flg
 hal_iks32mc09x_irqfun.c, 375
 hal_iksmcu_irqfun.h, 190
 HAL_ADC_IrqSamp2
 hal_iks32mc09x_irqfun.c, 375
 hal_iksmcu_irqfun.h, 191
 HAL_ADC_IrqSamp2Clear
 hal_iks32mc09x_irqfun.c, 375
 hal_iksmcu_irqfun.h, 191
 HAL_ADC_IrqSamp2Disable
 hal_iks32mc09x_irqfun.c, 376
 hal_iksmcu_irqfun.h, 191
 HAL_ADC_IrqSamp2Enable
 hal_iks32mc09x_irqfun.c, 376
 hal_iksmcu_irqfun.h, 192
 HAL_ADC_IrqSamp2Flg
 hal_iks32mc09x_irqfun.c, 376
 hal_iksmcu_irqfun.h, 192
 HAL_ADC_ManualSampling
 hal_iks32mc09x_adc.c, 283
 hal_iksmcu_adc.h, 107
 HAL_ADC_PORT_AGND
 hal_iksmcu_adc.h, 101
 HAL_ADC_PORT_CH0
 hal_iksmcu_adc.h, 101
 HAL_ADC_PORT_CH1
 hal_iksmcu_adc.h, 101

HAL_ADC_PORT_CH10
 hal_lksmcu_adc.h, 101

HAL_ADC_PORT_CH11
 hal_lksmcu_adc.h, 101

HAL_ADC_PORT_CH12
 hal_lksmcu_adc.h, 101

HAL_ADC_PORT_CH13
 hal_lksmcu_adc.h, 101

HAL_ADC_PORT_CH14
 hal_lksmcu_adc.h, 101

HAL_ADC_PORT_CH15
 hal_lksmcu_adc.h, 101

HAL_ADC_PORT_CH16
 hal_lksmcu_adc.h, 101

HAL_ADC_PORT_CH17
 hal_lksmcu_adc.h, 101

HAL_ADC_PORT_CH18
 hal_lksmcu_adc.h, 101

HAL_ADC_PORT_CH19
 hal_lksmcu_adc.h, 101

HAL_ADC_PORT_CH2
 hal_lksmcu_adc.h, 101

HAL_ADC_PORT_CH3
 hal_lksmcu_adc.h, 101

HAL_ADC_PORT_CH4
 hal_lksmcu_adc.h, 101

HAL_ADC_PORT_CH5
 hal_lksmcu_adc.h, 101

HAL_ADC_PORT_CH6
 hal_lksmcu_adc.h, 101

HAL_ADC_PORT_CH7
 hal_lksmcu_adc.h, 101

HAL_ADC_PORT_CH8
 hal_lksmcu_adc.h, 101

HAL_ADC_PORT_CH9
 hal_lksmcu_adc.h, 101

HAL_ADC_PORT_DAC0
 hal_lksmcu_adc.h, 101

HAL_ADC_PORT_DAC1
 hal_lksmcu_adc.h, 101

HAL_ADC_PORT_LDO12
 hal_lksmcu_adc.h, 101

HAL_ADC_PORT_OPA0
 hal_lksmcu_adc.h, 101

HAL_ADC_PORT_OPA1
 hal_lksmcu_adc.h, 101

HAL_ADC_PORT_OPA2
 hal_lksmcu_adc.h, 101

HAL_ADC_PORT_OPA3
 hal_lksmcu_adc.h, 101

HAL_ADC_PORT_OPA4
 hal_lksmcu_adc.h, 101

HAL_ADC_PORT_OPA5
 hal_lksmcu_adc.h, 101

HAL_ADC_PORT_REF
 hal_lksmcu_adc.h, 101

HAL_ADC_PORT_TMP0
 hal_lksmcu_adc.h, 101

HAL_ADC_PORT_VDD
 hal_lksmcu_adc.h, 101

HAL_ADC_PORT_x
 hal_lksmcu_adc.h, 101

HAL_ADC_REFERENCE_EXTERNAL
 hal_lksmcu_adc.h, 102

HAL_ADC_REFERENCE_INTERNAL
 hal_lksmcu_adc.h, 102

HAL_ADC_REFERENCE_x
 hal_lksmcu_adc.h, 101

HAL_ADC_Reset
 hal_lks32mc09x_adc.c, 284
 hal_lksmcu_adc.h, 107

HAL_ADC_SAMP_NUMBER_0
 hal_lksmcu_adc.h, 102

HAL_ADC_SAMP_NUMBER_1
 hal_lksmcu_adc.h, 102

HAL_ADC_SAMP_NUMBER_10
 hal_lksmcu_adc.h, 102

HAL_ADC_SAMP_NUMBER_11
 hal_lksmcu_adc.h, 102

HAL_ADC_SAMP_NUMBER_12
 hal_lksmcu_adc.h, 102

HAL_ADC_SAMP_NUMBER_13
 hal_lksmcu_adc.h, 102

HAL_ADC_SAMP_NUMBER_14
 hal_lksmcu_adc.h, 102

HAL_ADC_SAMP_NUMBER_15
 hal_lksmcu_adc.h, 102

HAL_ADC_SAMP_NUMBER_2
 hal_lksmcu_adc.h, 102

HAL_ADC_SAMP_NUMBER_3
 hal_lksmcu_adc.h, 102

HAL_ADC_SAMP_NUMBER_4
 hal_lksmcu_adc.h, 102

HAL_ADC_SAMP_NUMBER_5
 hal_lksmcu_adc.h, 102

HAL_ADC_SAMP_NUMBER_6
 hal_lksmcu_adc.h, 102

HAL_ADC_SAMP_NUMBER_7
 hal_lksmcu_adc.h, 102

HAL_ADC_SAMP_NUMBER_8
 hal_lksmcu_adc.h, 102

HAL_ADC_SAMP_NUMBER_9
 hal_lksmcu_adc.h, 102

HAL_ADC_SAMP_NUMBER_MAX
 hal_lksmcu_adc.h, 102

HAL_ADC_SAMP_NUMBER_x
 hal_lksmcu_adc.h, 102

HAL_ADC_SAMP_SEQUENCE_0
 hal_lksmcu_adc.h, 102

HAL_ADC_SAMP_SEQUENCE_1
 hal_lksmcu_adc.h, 103

HAL_ADC_SAMP_SEQUENCE_10
 hal_lksmcu_adc.h, 103

HAL_ADC_SAMP_SEQUENCE_11
 hal_lksmcu_adc.h, 103

HAL_ADC_SAMP_SEQUENCE_12

[hal_lksmcu_adc.h](#), [103](#)
 HAL_ADC_SAMP_SEQUENCE_13
[hal_lksmcu_adc.h](#), [103](#)
 HAL_ADC_SAMP_SEQUENCE_14
[hal_lksmcu_adc.h](#), [103](#)
 HAL_ADC_SAMP_SEQUENCE_15
[hal_lksmcu_adc.h](#), [103](#)
 HAL_ADC_SAMP_SEQUENCE_16
[hal_lksmcu_adc.h](#), [103](#)
 HAL_ADC_SAMP_SEQUENCE_17
[hal_lksmcu_adc.h](#), [103](#)
 HAL_ADC_SAMP_SEQUENCE_18
[hal_lksmcu_adc.h](#), [103](#)
 HAL_ADC_SAMP_SEQUENCE_19
[hal_lksmcu_adc.h](#), [103](#)
 HAL_ADC_SAMP_SEQUENCE_2
[hal_lksmcu_adc.h](#), [103](#)
 HAL_ADC_SAMP_SEQUENCE_3
[hal_lksmcu_adc.h](#), [103](#)
 HAL_ADC_SAMP_SEQUENCE_4
[hal_lksmcu_adc.h](#), [103](#)
 HAL_ADC_SAMP_SEQUENCE_5
[hal_lksmcu_adc.h](#), [103](#)
 HAL_ADC_SAMP_SEQUENCE_6
[hal_lksmcu_adc.h](#), [103](#)
 HAL_ADC_SAMP_SEQUENCE_7
[hal_lksmcu_adc.h](#), [103](#)
 HAL_ADC_SAMP_SEQUENCE_8
[hal_lksmcu_adc.h](#), [103](#)
 HAL_ADC_SAMP_SEQUENCE_9
[hal_lksmcu_adc.h](#), [103](#)
 HAL_ADC_SAMP_SEQUENCE_x
[hal_lksmcu_adc.h](#), [102](#)
 HAL_ADC_SetGpioIn
[hal_lks32mc09x_adc.c](#), [284](#)
[hal_lksmcu_adc.h](#), [108](#)
 HAL_ADC_SetGpioTriggerOut
[hal_lks32mc09x_adc.c](#), [285](#)
[hal_lksmcu_adc.h](#), [108](#)
 HAL_ADC_SetReference
[hal_lks32mc09x_adc.c](#), [286](#)
[hal_lksmcu_adc.h](#), [109](#)
 HAL_ADC_SetSamplingChannel
[hal_lks32mc09x_adc.c](#), [286](#)
[hal_lksmcu_adc.h](#), [109](#)
 HAL_ADC_SetSamplingQuantity
[hal_lks32mc09x_adc.c](#), [287](#)
[hal_lksmcu_adc.h](#), [110](#)
 HAL_ADC_SetTriggerSignal
[hal_lks32mc09x_adc.c](#), [288](#)
[hal_lksmcu_adc.h](#), [111](#)
 HAL_ADC_SoftTrigger
[hal_lks32mc09x_adc.c](#), [289](#)
[hal_lksmcu_adc.h](#), [112](#)
 HAL_ADC_TRIGGER_MCPWM
[hal_lksmcu_adc.h](#), [103](#)
 HAL_ADC_TRIGGER_SOFT
[hal_lksmcu_adc.h](#), [103](#)
 HAL_ADC_TRIGGER_TIMER
[hal_lksmcu_adc.h](#), [103](#)
 HAL_ADC_TRIGGER_x
[hal_lksmcu_adc.h](#), [103](#)
 HAL_ADC_TypeDef, [8](#)
 AMC0, [9](#)
 AMC1, [9](#)
 CFG, [9](#)
 CHN0, [10](#)
 CHN1, [10](#)
 CHN2, [10](#)
 CHNT, [10](#)
 DAT, [10](#)
 DC0, [11](#)
 DC1, [11](#)
 GAIN, [11](#)
 GEN, [11](#)
 HTH, [11](#)
 IE, [12](#)
 IF, [12](#)
 LTH, [12](#)
 RESERVED, [12](#)
 RESERVED1, [12](#)
 RESERVED2, [12](#)
 RESERVED3, [13](#)
 SWT, [13](#)
 TRIG, [13](#)
 HAL_ADC_UpdateTrig
[hal_lks32mc09x_adc.c](#), [289](#)
 HAL_ADC_x
[hal_lksmcu_adc.h](#), [103](#)
 HAL_AON_TypeDef, [13](#)
 BKP_REG0, [14](#)
 BKP_REG1, [14](#)
 EVT_RCD, [14](#)
 IO_WAKE_EN, [15](#)
 IO_WAKE_POL, [15](#)
 PWR_CFG, [15](#)
 hal_baud
[hal_lks32mc09x_i2c.c](#), [364](#)
 HAL_CAN_IRQHandler
[hal_lks32mc09x.h](#), [79](#)
 HAL_CAN_IRQHandler_00
[hal_lks32mc09x.h](#), [79](#)
 HAL_CL_IRQHandler
[hal_lks32mc09x.h](#), [79](#)
 HAL_CL_IRQHandler_00
[hal_lks32mc09x.h](#), [79](#)
 HAL_CLK_FREQ_RC_12MHZ
[hal_lksmcu_clk.h](#), [116](#)
 HAL_CLK_FREQ_RC_16MHZ
[hal_lksmcu_clk.h](#), [116](#)
 HAL_CLK_FREQ_RC_18MHZ
[hal_lksmcu_clk.h](#), [116](#)
 HAL_CLK_FREQ_RC_192MHZ
[hal_lksmcu_clk.h](#), [117](#)
 HAL_CLK_FREQ_RC_24MHZ
[hal_lksmcu_clk.h](#), [116](#)

HAL_CLK_FREQ_RC_32MHZ
 hal_lksmcu_clk.h, 116
 HAL_CLK_FREQ_RC_36MHZ
 hal_lksmcu_clk.h, 116
 HAL_CLK_FREQ_RC_48MHZ
 hal_lksmcu_clk.h, 116
 HAL_CLK_FREQ_RC_4MHZ
 hal_lksmcu_clk.h, 117
 HAL_CLK_FREQ_RC_64MHZ
 hal_lksmcu_clk.h, 116
 HAL_CLK_FREQ_RC_6MHZ
 hal_lksmcu_clk.h, 117
 HAL_CLK_FREQ_RC_72MHZ
 hal_lksmcu_clk.h, 116
 HAL_CLK_FREQ_RC_8MHZ
 hal_lksmcu_clk.h, 116
 HAL_CLK_FREQ_RC_96MHZ
 hal_lksmcu_clk.h, 116
 HAL_CLK_FREQ_RC_9MHZ
 hal_lksmcu_clk.h, 116
 HAL_CLK_FREQ_x
 hal_lksmcu_clk.h, 116
 HAL_CLK_FREQ_XTAL_12MHZ
 hal_lksmcu_clk.h, 116
 HAL_CLK_FREQ_XTAL_192MHZ
 hal_lksmcu_clk.h, 117
 HAL_CLK_FREQ_XTAL_24MHZ
 hal_lksmcu_clk.h, 116
 HAL_CLK_FREQ_XTAL_48MHZ
 hal_lksmcu_clk.h, 116
 HAL_CLK_FREQ_XTAL_6MHZ
 hal_lksmcu_clk.h, 116
 HAL_CLK_FREQ_XTAL_96MHZ
 hal_lksmcu_clk.h, 117
 HAL_CLK_GetMcuFreq
 hal_lks32mc09x_clk.c, 295
 hal_lksmcu_clk.h, 117
 HAL_CLK_mclk
 hal_lks32mc09x_clk.c, 297
 HAL_CLK_SetMcuFreq
 hal_lks32mc09x_clk.c, 296
 hal_lksmcu_clk.h, 118
 HAL_CLU_TypeDef, 15
 CF, 16
 EN, 16
 FN, 17
 IE, 17
 IF, 17
 MX, 17
 OUT, 17
 HAL_CMP
 hal_lks32mc09x_reg.c, 419
 hal_lks32mc09x_reg.h, 92
 HAL_CMP_0
 hal_lksmcu_cmp.h, 123
 HAL_CMP_1
 hal_lksmcu_cmp.h, 123
 HAL_CMP_2
 hal_lksmcu_cmp.h, 123
 HAL_CMP_3
 hal_lksmcu_cmp.h, 123
 HAL_CMP_4
 hal_lksmcu_cmp.h, 123
 HAL_CMP_5
 hal_lksmcu_cmp.h, 123
 HAL_CMP_CHANNEL_IN_0
 hal_lksmcu_cmp.h, 121
 HAL_CMP_CHANNEL_IN_DAC
 hal_lksmcu_cmp.h, 121
 HAL_CMP_CHANNEL_IN_MID
 hal_lksmcu_cmp.h, 121
 HAL_CMP_CHANNEL_IN_REF
 hal_lksmcu_cmp.h, 121
 HAL_CMP_CHANNEL_INx
 hal_lksmcu_cmp.h, 121
 HAL_CMP_CHANNEL_IP0
 hal_lksmcu_cmp.h, 122
 HAL_CMP_CHANNEL_IP1
 hal_lksmcu_cmp.h, 122
 HAL_CMP_CHANNEL_IP2
 hal_lksmcu_cmp.h, 122
 HAL_CMP_CHANNEL_IP3
 hal_lksmcu_cmp.h, 122
 HAL_CMP_CHANNEL_IP4
 hal_lksmcu_cmp.h, 122
 HAL_CMP_CHANNEL_IP5
 hal_lksmcu_cmp.h, 122
 HAL_CMP_CHANNEL_IP6
 hal_lksmcu_cmp.h, 122
 HAL_CMP_CHANNEL_IP7
 hal_lksmcu_cmp.h, 122
 HAL_CMP_CHANNEL_IP_AVSS
 hal_lksmcu_cmp.h, 122
 HAL_CMP_CHANNEL_IP_DAC0
 hal_lksmcu_cmp.h, 122
 HAL_CMP_CHANNEL_IP_DAC1
 hal_lksmcu_cmp.h, 122
 HAL_CMP_CHANNEL_IP_OPA0_IN
 hal_lksmcu_cmp.h, 122
 HAL_CMP_CHANNEL_IP_OPA0_IP
 hal_lksmcu_cmp.h, 122
 HAL_CMP_CHANNEL_IP_OPA0_OUTN
 hal_lksmcu_cmp.h, 122
 HAL_CMP_CHANNEL_IP_OPA0_OUTP
 hal_lksmcu_cmp.h, 122
 HAL_CMP_CHANNEL_IP_OPA1_IN
 hal_lksmcu_cmp.h, 122
 HAL_CMP_CHANNEL_IP_OPA1_IP
 hal_lksmcu_cmp.h, 122
 HAL_CMP_CHANNEL_IP_OPA1_OUTN
 hal_lksmcu_cmp.h, 122
 HAL_CMP_CHANNEL_IP_OPA1_OUTP
 hal_lksmcu_cmp.h, 122
 HAL_CMP_CHANNEL_IP_OPA2_IN
 hal_lksmcu_cmp.h, 122
 HAL_CMP_CHANNEL_IP_OPA2_IP

[hal_lksmcu_cmp.h](#), 122
 HAL_CMP_CHANNEL_IP_OPA2.OUTN
[hal_lksmcu_cmp.h](#), 122
 HAL_CMP_CHANNEL_IP_OPA2.OUTPUT
[hal_lksmcu_cmp.h](#), 122
 HAL_CMP_CHANNEL_IP_OPA3.IN
[hal_lksmcu_cmp.h](#), 122
 HAL_CMP_CHANNEL_IP_OPA3.IP
[hal_lksmcu_cmp.h](#), 122
 HAL_CMP_CHANNEL_IP_OPA3.OUTN
[hal_lksmcu_cmp.h](#), 122
 HAL_CMP_CHANNEL_IP_OPA3.OUTPUT
[hal_lksmcu_cmp.h](#), 122
 HAL_CMP_CHANNEL_IP_OPA4.IN
[hal_lksmcu_cmp.h](#), 122
 HAL_CMP_CHANNEL_IP_OPA4.IP
[hal_lksmcu_cmp.h](#), 122
 HAL_CMP_CHANNEL_IP_OPA4.OUTN
[hal_lksmcu_cmp.h](#), 122
 HAL_CMP_CHANNEL_IP_OPA4.OUTPUT
[hal_lksmcu_cmp.h](#), 122
 HAL_CMP_CHANNEL_IP_OPA5.IN
[hal_lksmcu_cmp.h](#), 122
 HAL_CMP_CHANNEL_IP_OPA5.IP
[hal_lksmcu_cmp.h](#), 122
 HAL_CMP_CHANNEL_IP_OPA5.OUTN
[hal_lksmcu_cmp.h](#), 122
 HAL_CMP_CHANNEL_IP_OPA5.OUTPUT
[hal_lksmcu_cmp.h](#), 122
 HAL_CMP_CHANNEL_IP_OPA.OUT_CMP0
[hal_lksmcu_cmp.h](#), 122
 HAL_CMP_CHANNEL_IPx
[hal_lksmcu_cmp.h](#), 121
 HAL_CMP.Disable
[hal_lks32mc09x_cmp.c](#), 300
[hal_lksmcu_cmp.h](#), 123
 HAL_CMP.Enable
[hal_lks32mc09x_cmp.c](#), 300
[hal_lksmcu_cmp.h](#), 123
 HAL_CMP.GetData
[hal_lks32mc09x_cmp.c](#), 301
[hal_lksmcu_cmp.h](#), 124
 HAL_CMP.IRQHandler
[hal_lks32mc09x.h](#), 79
 HAL_CMP.IRQHandler_00
[hal_lks32mc09x.h](#), 79
 HAL_CMP.IRQHandler_10
[hal_lks32mc09x.h](#), 80
 HAL_CMP.IRQHandler_20
[hal_lks32mc09x.h](#), 80
 HAL_CMP.IRQHandler_30
[hal_lks32mc09x.h](#), 80
 HAL_CMP.IRQHandler_40
[hal_lks32mc09x.h](#), 80
 HAL_CMP.IRQHandler_50
[hal_lks32mc09x.h](#), 80
 HAL_CMP.Reset
[hal_lks32mc09x_cmp.c](#), 301

[hal_lksmcu_cmp.h](#), 124
 HAL_CMP.SetChannel
[hal_lks32mc09x_cmp.c](#), 302
[hal_lksmcu_cmp.h](#), 125
 HAL_CMP.SetDigitalFilter
[hal_lks32mc09x_cmp.c](#), 302
[hal_lksmcu_cmp.h](#), 125
 HAL_CMP.SetGpioIn
[hal_lks32mc09x_cmp.c](#), 302
[hal_lksmcu_cmp.h](#), 125
 HAL_CMP.SetGpioOut
[hal_lks32mc09x_cmp.c](#), 303
[hal_lksmcu_cmp.h](#), 126
 HAL_CMP.TypeDef, 18
 BLCWIN, 19
 CFG, 19
 DATA, 19
 IE, 19
 IF, 19
 TCLK, 19
 HAL_CMP_x
[hal_lksmcu_cmp.h](#), 122
 HAL_CRC
[hal_lks32mc09x_reg.c](#), 419
[hal_lks32mc09x_reg.h](#), 92
 HAL_CRC_0
[hal_lksmcu_crc.h](#), 129
 HAL_CRC.Calc
[hal_lks32mc09x_crc.c](#), 308
[hal_lksmcu_crc.h](#), 129
 HAL_CRC.Disable
[hal_lks32mc09x_crc.c](#), 309
[hal_lksmcu_crc.h](#), 130
 HAL_CRC.Enable
[hal_lks32mc09x_crc.c](#), 309
[hal_lksmcu_crc.h](#), 130
 HAL_CRC.MODEL_CRC16_CCITT
[hal_lksmcu_crc.h](#), 129
 HAL_CRC.MODEL_CRC16_CCITT.FALSE
[hal_lksmcu_crc.h](#), 129
 HAL_CRC.MODEL_CRC16_DNP
[hal_lksmcu_crc.h](#), 129
 HAL_CRC.MODEL_CRC16_IBMx1
[hal_lksmcu_crc.h](#), 129
 HAL_CRC.MODEL_CRC16_MAXIM
[hal_lksmcu_crc.h](#), 129
 HAL_CRC.MODEL_CRC16_MODBU
[hal_lksmcu_crc.h](#), 129
 HAL_CRC.MODEL_CRC16_USBx1
[hal_lksmcu_crc.h](#), 129
 HAL_CRC.MODEL_CRC16_X25
[hal_lksmcu_crc.h](#), 129
 HAL_CRC.MODEL_CRC16_XMODEM
[hal_lksmcu_crc.h](#), 129
 HAL_CRC.MODEL_CRC32
[hal_lksmcu_crc.h](#), 129
 HAL_CRC.MODEL_CRC32_MPEG_2
[hal_lksmcu_crc.h](#), 129

HAL_CRC_MODEL_CRC8
 hal_lksmcu_crc.h, 129
 HAL_CRC_MODEL_CRC8_ITU
 hal_lksmcu_crc.h, 129
 HAL_CRC_MODEL_CRC8_MAXIMx
 hal_lksmcu_crc.h, 129
 HAL_CRC_MODEL_CRC8_ROHC
 hal_lksmcu_crc.h, 129
 HAL_CRC_MODEL_x
 hal_lksmcu_crc.h, 129
 HAL_CRC_Reflect
 hal_lks32mc09x_crc.c, 309
 HAL_CRC_Reset
 hal_lks32mc09x_crc.c, 310
 hal_lksmcu_crc.h, 131
 HAL_CRC_TypeDef, 20
 CR, 20
 DR, 20
 INIT, 21
 POL, 21
 HAL_CRC_x
 hal_lksmcu_crc.h, 129
 HAL_DAC_0
 hal_lksmcu_dac.h, 133
 HAL_DAC_1
 hal_lksmcu_dac.h, 133
 HAL_DAC_2
 hal_lksmcu_dac.h, 133
 HAL_DAC_3
 hal_lksmcu_dac.h, 133
 HAL_DAC_Disable
 hal_lks32mc09x_dac.c, 313
 hal_lksmcu_dac.h, 133
 HAL_DAC_Enable
 hal_lks32mc09x_dac.c, 314
 hal_lksmcu_dac.h, 133
 HAL_DAC_Out2GpioDisable
 hal_lks32mc09x_dac.c, 314
 hal_lksmcu_dac.h, 133
 HAL_DAC_Out2GpioEnable
 hal_lks32mc09x_dac.c, 314
 hal_lksmcu_dac.h, 134
 HAL_DAC_Reset
 hal_lks32mc09x_dac.c, 314
 hal_lksmcu_dac.h, 134
 HAL_DAC_SetGpioOut
 hal_lks32mc09x_dac.c, 314
 hal_lksmcu_dac.h, 134
 HAL_DAC_SetOutVal
 hal_lks32mc09x_dac.c, 315
 hal_lksmcu_dac.h, 135
 HAL_DAC_SetOutVoltageMV
 hal_lks32mc09x_dac.c, 315
 hal_lksmcu_dac.h, 135
 HAL_DAC_x
 hal_lksmcu_dac.h, 133
 HAL_DMA
 hal_lks32mc09x_reg.c, 419
 hal_lks32mc09x_reg.h, 92
 HAL_DMA_0
 hal_lksmcu_dma.h, 137
 HAL_DMA_1
 hal_lksmcu_dma.h, 137
 HAL_DMA_2
 hal_lksmcu_dma.h, 137
 HAL_DMA_3
 hal_lksmcu_dma.h, 137
 HAL_DMA_Disable
 hal_lks32mc09x_dma.c, 319
 hal_lksmcu_dma.h, 137
 HAL_DMA_ena
 hal_lks32mc09x_dma.c, 321
 HAL_DMA_Enable
 hal_lks32mc09x_dma.c, 319
 hal_lksmcu_dma.h, 137
 HAL_DMA_I2cRx
 hal_lks32mc09x_dma.c, 319
 hal_lksmcu_dma.h, 138
 HAL_DMA_I2cTx
 hal_lks32mc09x_dma.c, 319
 hal_lksmcu_dma.h, 138
 HAL_DMA_IRQHandler
 hal_lks32mc09x.h, 80
 HAL_DMA_IRQHandler_00
 hal_lks32mc09x.h, 80
 HAL_DMA_MemCopy
 hal_lks32mc09x_dma.c, 319
 hal_lksmcu_dma.h, 138
 HAL_DMA_Reset
 hal_lks32mc09x_dma.c, 320
 hal_lksmcu_dma.h, 138
 HAL_DMA_SpiRx
 hal_lks32mc09x_dma.c, 320
 hal_lksmcu_dma.h, 138
 HAL_DMA_SpiRxTx
 hal_lks32mc09x_dma.c, 320
 hal_lksmcu_dma.h, 138
 HAL_DMA_SpiTx
 hal_lks32mc09x_dma.c, 320
 hal_lksmcu_dma.h, 139
 HAL_DMA_TypeDef, 21
 CCR, 22
 CTMS, 22
 DADR, 22
 REN, 22
 SADR, 22
 HAL_DMA_UartRx
 hal_lks32mc09x_dma.c, 320
 hal_lksmcu_dma.h, 139
 HAL_DMA_UartTx
 hal_lks32mc09x_dma.c, 321
 hal_lksmcu_dma.h, 139
 HAL_DMA_x
 hal_lksmcu_dma.h, 137
 HAL_DMAMODULE_TypeDef, 23
 CTRL, 23

IE, 23
 IF, 23
 HAL_DSP
 hal_iks32mc09x_reg.c, 419
 hal_iks32mc09x_reg.h, 93
 HAL_DSP_0
 hal_iksmcu_dsp.h, 141
 HAL_DSP_Cos
 hal_iks32mc09x_dsp.c, 324
 hal_iksmcu_dsp.h, 141
 HAL_DSP_Disable
 hal_iks32mc09x_dsp.c, 324
 hal_iksmcu_dsp.h, 142
 HAL_DSP_Div
 hal_iks32mc09x_dsp.c, 324
 hal_iksmcu_dsp.h, 142
 HAL_DSP_Enable
 hal_iks32mc09x_dsp.c, 324
 hal_iksmcu_dsp.h, 142
 HAL_DSP_IRQHandler
 hal_iks32mc09x.h, 80
 HAL_DSP_IRQHandler_00
 hal_iks32mc09x.h, 81
 HAL_DSP_Reset
 hal_iks32mc09x_dsp.c, 324
 hal_iksmcu_dsp.h, 142
 HAL_DSP_Sin
 hal_iks32mc09x_dsp.c, 325
 hal_iksmcu_dsp.h, 142
 HAL_DSP_SinCos
 hal_iks32mc09x_dsp.c, 325
 hal_iksmcu_dsp.h, 143
 HAL_DSP_Sqrt
 hal_iks32mc09x_dsp.c, 325
 hal_iksmcu_dsp.h, 143
 HAL_DSP_TypeDef, 24
 DID, 24
 DID1, 24
 DIS, 25
 DIS1, 25
 QUO, 25
 QUO1, 25
 RAD, 25
 REM, 25
 RESERVED, 26
 RESERVED1, 26
 RESERVED2, 26
 SQRT, 26
 HAL_DSP_x
 hal_iksmcu_dsp.h, 141
 HAL_EXTI_TypeDef, 26
 CLKO, 27
 CR0, 27
 CR1, 27
 IE, 28
 IF, 28
 SWAP, 28
 HAL_FLASH
 hal_iks32mc09x_reg.c, 419
 hal_iks32mc09x_reg.h, 93
 HAL_FLASH_0
 hal_iksmcu_flash.h, 145
 HAL_FLASH_1
 hal_iksmcu_flash.h, 145
 HAL_FLASH_data
 hal_iks32mc09x_flash.c, 337
 HAL_FLASH_Disable
 hal_iks32mc09x_flash.c, 329
 hal_iksmcu_flash.h, 146
 HAL_FLASH_DisableCache
 hal_iks32mc09x_flash.c, 329
 hal_iksmcu_flash.h, 146
 HAL_FLASH_Enable
 hal_iks32mc09x_flash.c, 329
 hal_iksmcu_flash.h, 146
 HAL_FLASH_EnableCache
 hal_iks32mc09x_flash.c, 330
 hal_iksmcu_flash.h, 146
 HAL_FLASH_Erase
 hal_iks32mc09x_flash.c, 330
 hal_iksmcu_flash.h, 147
 HAL_FLASH_Program
 hal_iks32mc09x_flash.c, 330
 hal_iksmcu_flash.h, 147
 HAL_FLASH_Read
 hal_iks32mc09x_flash.c, 331
 hal_iksmcu_flash.h, 148
 HAL_FLASH_ReadSectorSize
 hal_iks32mc09x_flash.c, 332
 hal_iksmcu_flash.h, 148
 HAL_FLASH_ReadUint16
 hal_iks32mc09x_flash.c, 332
 hal_iksmcu_flash.h, 149
 HAL_FLASH_ReadUint32
 hal_iks32mc09x_flash.c, 333
 hal_iksmcu_flash.h, 149
 HAL_FLASH_ReadUint8
 hal_iks32mc09x_flash.c, 333
 hal_iksmcu_flash.h, 150
 HAL_FLASH_Reset
 hal_iks32mc09x_flash.c, 334
 hal_iksmcu_flash.h, 150
 HAL_FLASH_SecurityParameterCrc
 hal_iks32mc09x_flash.c, 334
 HAL_FLASH_SecurityParameterRead
 hal_iks32mc09x_flash.c, 334
 hal_iksmcu_flash.h, 151
 HAL_FLASH_SecurityParameterSave
 hal_iks32mc09x_flash.c, 335
 hal_iksmcu_flash.h, 151
 HAL_FLASH_TypeDef, 28
 ADDR, 30
 CFG, 30
 EPSM, 30
 EPSN, 30
 ERASE, 31

LIB_ADDR, 31
NADDR, 31
NCFG, 31
NERASE, 31
NKEY, 32
NRDATA, 32
NWDATA, 32
PROTECT, 32
RDATA, 32
READY, 32
RESERVED, 33
SIZE, 33
WDATA, 33
HAL_FLASH_Update
 hal_iks32mc09x_flash.c, 336
 hal_iksmcu_flash.h, 152
HAL_FLASH_x
 hal_iksmcu_flash.h, 145
HAL_FMAC_IRQHandler_00
 hal_iks32mc09x.h, 81
HAL_GPIO
 hal_iks32mc09x_reg.c, 419
 hal_iks32mc09x_reg.h, 93
HAL_GPIO_AF_ADC
 hal_iksmcu_gpio.h, 156
HAL_GPIO_AF_ADC_TRIG
 hal_iksmcu_gpio.h, 157
HAL_GPIO_AF_CAN
 hal_iksmcu_gpio.h, 157
HAL_GPIO_AF_CL
 hal_iksmcu_gpio.h, 157
HAL_GPIO_AF_CLK
 hal_iksmcu_gpio.h, 156
HAL_GPIO_AF_CLK_OUT
 hal_iksmcu_gpio.h, 156
HAL_GPIO_AF_CMP
 hal_iksmcu_gpio.h, 156
HAL_GPIO_AF_CMP_OUT
 hal_iksmcu_gpio.h, 156
HAL_GPIO_AF_DAC
 hal_iksmcu_gpio.h, 156
HAL_GPIO_AF_GPIO
 hal_iksmcu_gpio.h, 156
HAL_GPIO_AF_HALL
 hal_iksmcu_gpio.h, 156
HAL_GPIO_AF_I2C
 hal_iksmcu_gpio.h, 156
HAL_GPIO_AF_MCPWM
 hal_iksmcu_gpio.h, 156
HAL_GPIO_AF_OPA
 hal_iksmcu_gpio.h, 156
HAL_GPIO_AF_OPA_OUT
 hal_iksmcu_gpio.h, 156
HAL_GPIO_AF_QEP
 hal_iksmcu_gpio.h, 157
HAL_GPIO_AF_SIF
 hal_iksmcu_gpio.h, 157
HAL_GPIO_AF_SPI
 hal_iksmcu_gpio.h, 156
 hal_iksmcu_gpio.h, 156
HAL_GPIO_AF_TIMER0
 hal_iksmcu_gpio.h, 156
HAL_GPIO_AF_TIMER1
 hal_iksmcu_gpio.h, 156
HAL_GPIO_AF_TIMER2
 hal_iksmcu_gpio.h, 156
HAL_GPIO_AF_TIMER3
 hal_iksmcu_gpio.h, 157
HAL_GPIO_AF_TIMER4
 hal_iksmcu_gpio.h, 157
HAL_GPIO_AF_TIMER5
 hal_iksmcu_gpio.h, 157
HAL_GPIO_AF_UART
 hal_iksmcu_gpio.h, 156
HAL_GPIO_AFx
 hal_iksmcu_gpio.h, 156
HAL_GPIO_Disable
 hal_iks32mc09x_gpio.c, 342
 hal_iksmcu_gpio.h, 160
HAL_GPIO_Enable
 hal_iks32mc09x_gpio.c, 342
 hal_iksmcu_gpio.h, 160
HAL_GPIO_GetInData
 hal_iks32mc09x_gpio.c, 342
 hal_iksmcu_gpio.h, 160
HAL_GPIO_GetOutData
 hal_iks32mc09x_gpio.c, 343
 hal_iksmcu_gpio.h, 160
HAL_GPIO_Invert
 hal_iks32mc09x_gpio.c, 343
 hal_iksmcu_gpio.h, 161
HAL_GPIO_IRQHandler
 hal_iks32mc09x.h, 81
HAL_GPIO_IRQHandler_00
 hal_iks32mc09x.h, 81
HAL_GPIO_MODE_AFE
 hal_iksmcu_gpio.h, 157
HAL_GPIO_MODE_IN
 hal_iksmcu_gpio.h, 157
HAL_GPIO_MODE_INOUT
 hal_iksmcu_gpio.h, 157
HAL_GPIO_MODE_INOUT_0
 hal_iksmcu_gpio.h, 157
HAL_GPIO_MODE_INOUT_1
 hal_iksmcu_gpio.h, 157
HAL_GPIO_MODE_INUP
 hal_iksmcu_gpio.h, 157
HAL_GPIO_MODE_INUPOUT
 hal_iksmcu_gpio.h, 157
HAL_GPIO_MODE_INUPOUT_0
 hal_iksmcu_gpio.h, 157
HAL_GPIO_MODE_INUPOUT_1
 hal_iksmcu_gpio.h, 157
HAL_GPIO_MODE_NONE
 hal_iksmcu_gpio.h, 157
HAL_GPIO_MODE_OUT
 hal_iksmcu_gpio.h, 157

HAL_GPIO_MODE_OUT_0
 hal_lksmcu_gpio.h, 157

HAL_GPIO_MODE_OUT_1
 hal_lksmcu_gpio.h, 157

HAL_GPIO_MODE_x
 hal_lksmcu_gpio.h, 157

HAL_GPIO_P0_0
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P0_1
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P0_10
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P0_11
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P0_12
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P0_13
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P0_14
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P0_15
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P0_2
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P0_3
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P0_4
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P0_5
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P0_6
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P0_7
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P0_8
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P0_9
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P1_0
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P1_1
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P1_10
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P1_11
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P1_12
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P1_13
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P1_14
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P1_15
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P1_2
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P1_3
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P1_4
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P1_5
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P1_6
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P1_7
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P1_8
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P1_9
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P2_0
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P2_1
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P2_10
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P2_11
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P2_12
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P2_13
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P2_14
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P2_15
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P2_2
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P2_3
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P2_4
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P2_5
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P2_6
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P2_7
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P2_8
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P2_9
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P3_0
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P3_1
 hal_lksmcu_gpio.h, 158

HAL_GPIO_P3_10
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P3_11
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P3_12
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P3_13
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P3_14
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P3_15
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P3_2
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P3_3
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P3_4
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P3_5
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P3_6
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P3_7
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P3_8
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P3_9
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P4_0
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P4_1
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P4_10
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P4_11
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P4_12
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P4_13
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P4_14
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P4_15
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P4_2
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P4_3
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P4_4
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P4_5
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P4_6
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P4_7
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P4_8
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P4_9
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P5_0
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P5_1
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P5_10
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P5_11
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P5_12
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P5_13
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P5_14
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P5_15
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P5_2
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P5_3
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P5_4
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P5_5
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P5_6
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P5_7
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P5_8
 hal_lksmcu_gpio.h, 159

HAL_GPIO_P5_9
 hal_lksmcu_gpio.h, 159

HAL_GPIO_Px
 hal_lksmcu_gpio.h, 157

HAL_GPIO_Reset
 hal_lks32mc09x_gpio.c, 343
 hal_lksmcu_gpio.h, 161

HAL_GPIO_RstReuseGpioDisable
 hal_lks32mc09x_gpio.c, 344
 hal_lksmcu_gpio.h, 161

HAL_GPIO_RstReuseGpioEnable
 hal_lks32mc09x_gpio.c, 344
 hal_lksmcu_gpio.h, 161

HAL_GPIO_SetAf
 hal_lks32mc09x_gpio.c, 344
 hal_lksmcu_gpio.h, 162

HAL_GPIO_SetHigh
 hal_lks32mc09x_gpio.c, 345
 hal_lksmcu_gpio.h, 163

HAL_GPIO_SetLow
 hal_lks32mc09x_gpio.c, 346
 hal_lksmcu_gpio.h, 164

HAL_GPIO_SetMode
 hal_lks32mc09x_gpio.c, 346
 hal_lksmcu_gpio.h, 164

HAL_GPIO_SwdReuseGpioDisable
 hal_lks32mc09x_gpio.c, 347
 hal_lksmcu_gpio.h, 165

HAL_GPIO_SwdReuseGpioEnable
 hal_lks32mc09x_gpio.c, 347
 hal_lksmcu_gpio.h, 165

HAL_GPIO_TypeDef, 33

BRR, 35

BSRR, 35

F3210, 35

F7654, 36

- FBA98, 36
- FFEDC, 36
- PDI, 36
- PDO, 36
- PFLT, 36
- PIE, 37
- PODE, 37
- POE, 37
- PUE, 37
- RESERVED, 38
- HAL_GPIO_Write
 - hal_iks32mc09x_gpio.c, 348
 - hal_iksmcu_gpio.h, 166
- HAL_HALL
 - hal_iks32mc09x_reg.c, 419
 - hal_iks32mc09x_reg.h, 93
- HAL_HALL_0
 - hal_iksmcu_hall.h, 170
- HAL_HALL_1
 - hal_iksmcu_hall.h, 170
- HAL_HALL_Disable
 - hal_iks32mc09x_hall.c, 352
 - hal_iksmcu_hall.h, 170
- HAL_HALL_Enable
 - hal_iks32mc09x_hall.c, 352
 - hal_iksmcu_hall.h, 170
- HAL_HALL_IrqChange
 - hal_iks32mc09x_irqfun.c, 376
 - hal_iksmcu_irqfun.h, 192
- HAL_HALL_IrqChangeClear
 - hal_iks32mc09x_irqfun.c, 377
 - hal_iksmcu_irqfun.h, 193
- HAL_HALL_IrqChangeDisable
 - hal_iks32mc09x_irqfun.c, 377
 - hal_iksmcu_irqfun.h, 193
- HAL_HALL_IrqChangeEnable
 - hal_iks32mc09x_irqfun.c, 377
 - hal_iksmcu_irqfun.h, 193
- HAL_HALL_IrqChangeFlg
 - hal_iks32mc09x_irqfun.c, 379
 - hal_iksmcu_irqfun.h, 193
- HAL_HALL_IRQHandler
 - hal_iks32mc09x.h, 81
- HAL_HALL_IRQHandler_00
 - hal_iks32mc09x.h, 81
- HAL_HALL_IRQHandler_10
 - hal_iks32mc09x.h, 81
- HAL_HALL_IrqOver
 - hal_iks32mc09x_irqfun.c, 379
 - hal_iksmcu_irqfun.h, 194
- HAL_HALL_IrqOverClear
 - hal_iks32mc09x_irqfun.c, 379
 - hal_iksmcu_irqfun.h, 194
- HAL_HALL_IrqOverDisable
 - hal_iks32mc09x_irqfun.c, 380
 - hal_iksmcu_irqfun.h, 194
- HAL_HALL_IrqOverEnable
 - hal_iks32mc09x_irqfun.c, 380
- hal_iksmcu_irqfun.h, 195
- HAL_HALL_IrqOverFlg
 - hal_iks32mc09x_irqfun.c, 380
 - hal_iksmcu_irqfun.h, 195
- HAL_HALL_ReadHallData
 - hal_iks32mc09x_hall.c, 353
 - hal_iksmcu_hall.h, 171
- HAL_HALL_ReadHallWidth
 - hal_iks32mc09x_hall.c, 353
 - hal_iksmcu_hall.h, 171
- HAL_HALL_ReadTh
 - hal_iks32mc09x_hall.c, 353
 - hal_iksmcu_hall.h, 171
- HAL_HALL_Reset
 - hal_iks32mc09x_hall.c, 354
 - hal_iksmcu_hall.h, 172
- HAL_HALL_SetFilter
 - hal_iks32mc09x_hall.c, 354
 - hal_iksmcu_hall.h, 172
- HAL_HALL_SetFilteruS
 - hal_iks32mc09x_hall.c, 354
 - hal_iksmcu_hall.h, 172
- HAL_HALL_SetFreq
 - hal_iks32mc09x_hall.c, 355
 - hal_iksmcu_hall.h, 173
- HAL_HALL_SetGpio
 - hal_iks32mc09x_hall.c, 356
 - hal_iksmcu_hall.h, 174
- HAL_HALL_SetTh
 - hal_iks32mc09x_hall.c, 356
 - hal_iksmcu_hall.h, 174
- HAL_HALL_StartCounting
 - hal_iks32mc09x_hall.c, 357
 - hal_iksmcu_hall.h, 175
- HAL_HALL_StopCounting
 - hal_iks32mc09x_hall.c, 357
 - hal_iksmcu_hall.h, 175
- HAL_HALL_TypeDef, 38
 - CFG, 39
 - CNT, 39
 - INFO, 39
 - TH, 39
 - WIDTH, 39
- HAL_HALL_x
 - hal_iksmcu_hall.h, 170
- HAL_I2C
 - hal_iks32mc09x_reg.c, 420
 - hal_iks32mc09x_reg.h, 93
- HAL_I2C_0
 - hal_iksmcu_i2c.h, 177
- HAL_I2C_1
 - hal_iksmcu_i2c.h, 177
- HAL_I2C_BusReset
 - hal_iks32mc09x_i2c.c, 360
 - hal_iksmcu_i2c.h, 177
- HAL_I2C_Disable
 - hal_iks32mc09x_i2c.c, 360
 - hal_iksmcu_i2c.h, 178

HAL_I2C_Enable
 hal_iks32mc09x_i2c.c, 361
 hal_iksmcu_i2c.h, 178

HAL_I2C_IRQHandler
 hal_iks32mc09x.h, 81

HAL_I2C_IRQHandler_00
 hal_iks32mc09x.h, 82

HAL_I2C_IRQHandler_10
 hal_iks32mc09x.h, 82

HAL_I2C_Reset
 hal_iks32mc09x_i2c.c, 361
 hal_iksmcu_i2c.h, 178

HAL_I2C_RxData
 hal_iks32mc09x_i2c.c, 361
 hal_iksmcu_i2c.h, 179

HAL_I2C_SetBaud
 hal_iks32mc09x_i2c.c, 362
 hal_iksmcu_i2c.h, 179

HAL_I2C_SetGpioScl
 hal_iks32mc09x_i2c.c, 362
 hal_iksmcu_i2c.h, 180

HAL_I2C_SetGpioSda
 hal_iks32mc09x_i2c.c, 363
 hal_iksmcu_i2c.h, 180

HAL_I2C_TxData
 hal_iks32mc09x_i2c.c, 364
 hal_iksmcu_i2c.h, 181

HAL_I2C_TypeDef, 40
 ADDR, 40
 BCR, 40
 BSIZE, 41
 CFG, 41
 DATA, 41
 MSCR, 41
 SCR, 41

HAL_I2C_x
 hal_iksmcu_i2c.h, 177

hal_i2cio
 hal_iks32mc09x_i2c.c, 364

HAL_INIT_DEFAULE_MCLK
 hal_iks32mc09x.h, 82

HAL_INIT_DEFAULE_MCLK_MS
 hal_iks32mc09x.h, 82

HAL_INIT_DEFAULE_MCLK_US
 hal_iks32mc09x.h, 82

HAL_INIT_IRQHandler
 hal_iks32mc09x.h, 82

HAL_INIT_IRQHandler_00
 hal_iks32mc09x.h, 82

HAL_INIT_POWER_3.3V
 hal_iksmcu_init.h, 183

HAL_INIT_POWER_5V
 hal_iksmcu_init.h, 183

HAL_INIT_POWER_x
 hal_iksmcu_init.h, 182

HAL_INIT_PowerReady
 hal_iks32mc09x_init.c, 368
 hal_iksmcu_init.h, 183

HAL_INIT_SysInit
 hal_iks32mc09x_init.c, 368
 hal_iksmcu_init.h, 183

HAL_IRQ_Disable
 hal_iks32mc09x_irq.c, 369
 hal_iksmcu_irq.h, 185

HAL_IRQ_Enable
 hal_iks32mc09x_irq.c, 370
 hal_iksmcu_irq.h, 186

HAL_IRQ_MODULE_ADC_00
 hal_iksmcu_irq.h, 185

HAL_IRQ_MODULE_ADC_10
 hal_iksmcu_irq.h, 185

HAL_IRQ_MODULE_CL_00
 hal_iksmcu_irq.h, 185

HAL_IRQ_MODULE_CMP_00
 hal_iksmcu_irq.h, 185

HAL_IRQ_MODULE_CMP_10
 hal_iksmcu_irq.h, 185

HAL_IRQ_MODULE_DMA_00
 hal_iksmcu_irq.h, 185

HAL_IRQ_MODULE_GPIO_00
 hal_iksmcu_irq.h, 185

HAL_IRQ_MODULE_HALL_00
 hal_iksmcu_irq.h, 185

HAL_IRQ_MODULE_HardFault
 hal_iksmcu_irq.h, 185

HAL_IRQ_MODULE_I2C_00
 hal_iksmcu_irq.h, 185

HAL_IRQ_MODULE_INIT_00
 hal_iksmcu_irq.h, 185

HAL_IRQ_MODULE_MCPWM_00
 hal_iksmcu_irq.h, 185

HAL_IRQ_MODULE_MCPWM_01
 hal_iksmcu_irq.h, 185

HAL_IRQ_MODULE_NonMaskableInt
 hal_iksmcu_irq.h, 184

HAL_IRQ_MODULE_PendSV
 hal_iksmcu_irq.h, 185

HAL_IRQ_MODULE_QEP_00
 hal_iksmcu_irq.h, 185

HAL_IRQ_MODULE_SIF_00
 hal_iksmcu_irq.h, 185

HAL_IRQ_MODULE_SPI_00
 hal_iksmcu_irq.h, 185

HAL_IRQ_MODULE_SVCall
 hal_iksmcu_irq.h, 185

HAL_IRQ_MODULE_SW_00
 hal_iksmcu_irq.h, 185

HAL_IRQ_MODULE_SysTick
 hal_iksmcu_irq.h, 185

HAL_IRQ_MODULE_TIMER_00
 hal_iksmcu_irq.h, 185

HAL_IRQ_MODULE_TIMER_10
 hal_iksmcu_irq.h, 185

HAL_IRQ_MODULE_TIMER_20
 hal_iksmcu_irq.h, 185

HAL_IRQ_MODULE_UART_00

hal_lksmcu_irq.h, 185
 HAL_IRQ_MODULE_UART_10
 hal_lksmcu_irq.h, 185
 HAL_IRQ_MODULE_UART_20
 hal_lksmcu_irq.h, 185
 HAL_IRQ_MODULE_WAKE_00
 hal_lksmcu_irq.h, 185
 HAL_IRQ_MODULE_x
 hal_lksmcu_irq.h, 184
 HAL_IRQ_ModuleDisable
 hal_lks32mc09x_irq.c, 370
 hal_lksmcu_irq.h, 186
 HAL_IRQ_ModuleEnable
 hal_lks32mc09x_irq.c, 370
 hal_lksmcu_irq.h, 186
 HAL_IRQ_Reset
 hal_lks32mc09x_irq.c, 371
 hal_lksmcu_irq.h, 187
 HAL_IWDG
 hal_lks32mc09x_reg.c, 420
 hal_lks32mc09x_reg.h, 93
 HAL_IWDG_0
 hal_lksmcu_iwdg.h, 197
 HAL_IWDG_Disable
 hal_lks32mc09x_iwdg.c, 383
 hal_lksmcu_iwdg.h, 197
 HAL_IWDG_Enable
 hal_lks32mc09x_iwdg.c, 384
 hal_lksmcu_iwdg.h, 198
 HAL_IWDG_Feed
 hal_lks32mc09x_iwdg.c, 384
 hal_lksmcu_iwdg.h, 198
 HAL_IWDG_Reset
 hal_lks32mc09x_iwdg.c, 385
 hal_lksmcu_iwdg.h, 199
 HAL_IWDG_SetResetTimemS
 hal_lks32mc09x_iwdg.c, 385
 hal_lksmcu_iwdg.h, 199
 HAL_IWDG_TypeDef, 42
 CFG, 42
 CLR, 42
 CNT, 43
 PSW, 43
 RTH, 43
 WTH, 43
 HAL_IWDG_x
 hal_lksmcu_iwdg.h, 197
 hal_lks32mc09x.c, 75
 __HAL_LKS32MC05X_C__, 76
 hal_lks32mc09x.h, 76
 HAL_ADC_IRQHandler, 78
 HAL_ADC_IRQHandler_00, 78
 HAL_ADC_IRQHandler_10, 79
 HAL_ADC_IRQHandler_20, 79
 HAL_CAN_IRQHandler, 79
 HAL_CAN_IRQHandler_00, 79
 HAL_CL_IRQHandler, 79
 HAL_CL_IRQHandler_00, 79
 HAL_CMP_IRQHandler, 79
 HAL_CMP_IRQHandler_00, 79
 HAL_CMP_IRQHandler_10, 80
 HAL_CMP_IRQHandler_20, 80
 HAL_CMP_IRQHandler_30, 80
 HAL_CMP_IRQHandler_40, 80
 HAL_CMP_IRQHandler_50, 80
 HAL_DMA_IRQHandler, 80
 HAL_DMA_IRQHandler_00, 80
 HAL_DSP_IRQHandler, 80
 HAL_DSP_IRQHandler_00, 81
 HAL_FMAC_IRQHandler_00, 81
 HAL_GPIO_IRQHandler, 81
 HAL_GPIO_IRQHandler_00, 81
 HAL_HALL_IRQHandler, 81
 HAL_HALL_IRQHandler_00, 81
 HAL_HALL_IRQHandler_10, 81
 HAL_I2C_IRQHandler, 81
 HAL_I2C_IRQHandler_00, 82
 HAL_I2C_IRQHandler_10, 82
 HAL_INIT_DEFAULE_MCLK, 82
 HAL_INIT_DEFAULE_MCLK_MS, 82
 HAL_INIT_DEFAULE_MCLK_US, 82
 HAL_INIT_IRQHandler, 82
 HAL_INIT_IRQHandler_00, 82
 HAL_MCPWM_IRQHandler, 83
 HAL_MCPWM_IRQHandler_00, 83
 HAL_MCPWM_IRQHandler_01, 83
 HAL_MCPWM_IRQHandler_10, 83
 HAL_MCPWM_IRQHandler_11, 83
 HAL_PWRDN_IRQHandler, 83
 HAL_PWRDN_IRQHandler_00, 83
 HAL_QEP_IRQHandler, 83
 HAL_QEP_IRQHandler_00, 84
 HAL_QEP_IRQHandler_10, 84
 HAL_QEP_IRQHandler_20, 84
 HAL_QEP_IRQHandler_30, 84
 HAL_SIF_IRQHandler, 84
 HAL_SIF_IRQHandler_00, 84
 HAL_SPI_IRQHandler, 84
 HAL_SPI_IRQHandler_00, 84
 HAL_SPI_IRQHandler_10, 85
 HAL_SW_IRQHandler, 85
 HAL_SW_IRQHandler_00, 85
 HAL_TIMER_IRQHandler, 85
 HAL_TIMER_IRQHandler_00, 85
 HAL_TIMER_IRQHandler_10, 85
 HAL_TIMER_IRQHandler_20, 85
 HAL_TIMER_IRQHandler_30, 86
 HAL_TIMER_IRQHandler_40, 86
 HAL_UART_IRQHandler, 86
 HAL_UART_IRQHandler_00, 86
 HAL_UART_IRQHandler_10, 86
 HAL_UART_IRQHandler_20, 86
 HAL_WAKE_IRQHandler, 86
 HAL_WAKE_IRQHandler_00, 86
 HAL_WWDG_IRQHandler_00, 87
 hal_lks32mc09x_adc.c

- adc_soft_reg, 290
- HAL_ADC_Disable, 280
- HAL_ADC_Enable, 281
- HAL_ADC_EndSampling, 281
- HAL_ADC_GetRange, 281
- HAL_ADC_GetSamplingValues, 282
- HAL_ADC_GetValuesUnit, 283
- HAL_ADC_ManualSampling, 283
- HAL_ADC_Reset, 284
- HAL_ADC_SetGpioIn, 284
- HAL_ADC_SetGpioTriggerOut, 285
- HAL_ADC_SetReference, 286
- HAL_ADC_SetSamplingChannel, 286
- HAL_ADC_SetSamplingQuantity, 287
- HAL_ADC_SetTriggerSignal, 288
- HAL_ADC_SoftTrigger, 289
- HAL_ADC_UpdateTrig, 289
- hal_iks32mc09x_clk.c
 - HAL_CLK_GetMcuFreq, 295
 - HAL_CLK_mclk, 297
 - HAL_CLK_SetMcuFreq, 296
- hal_iks32mc09x_cmp.c
 - HAL_CMP_Disable, 300
 - HAL_CMP_Enable, 300
 - HAL_CMP_GetData, 301
 - HAL_CMP_Reset, 301
 - HAL_CMP_SetChannel, 302
 - HAL_CMP_SetDigitalFilter, 302
 - HAL_CMP_SetGpioIn, 302
 - HAL_CMP_SetGpioOut, 303
- hal_iks32mc09x_crc.c
 - CRC_Width, 307
 - CRC_WIDTH_16, 308
 - CRC_WIDTH_32, 308
 - CRC_WIDTH_8, 308
 - crcModels, 310
 - HAL_CRC_Calc, 308
 - HAL_CRC_Disable, 309
 - HAL_CRC_Enable, 309
 - HAL_CRC_Reflect, 309
 - HAL_CRC_Reset, 310
- hal_iks32mc09x_dac.c
 - HAL_DAC_Disable, 313
 - HAL_DAC_Enable, 314
 - HAL_DAC_Out2GpioDisable, 314
 - HAL_DAC_Out2GpioEnable, 314
 - HAL_DAC_Reset, 314
 - HAL_DAC_SetGpioOut, 314
 - HAL_DAC_SetOutVal, 315
 - HAL_DAC_SetOutVotagemV, 315
- hal_iks32mc09x_dma.c
 - HAL_DMA_Disable, 319
 - HAL_DMA_ena, 321
 - HAL_DMA_Enable, 319
 - HAL_DMA_I2cRx, 319
 - HAL_DMA_I2cTx, 319
 - HAL_DMA_MemCopy, 319
 - HAL_DMA_Reset, 320
 - HAL_DMA_SpiRx, 320
 - HAL_DMA_SpiRxTx, 320
 - HAL_DMA_SpiTx, 320
 - HAL_DMA_UartRx, 320
 - HAL_DMA_UartTx, 321
- hal_iks32mc09x_dsp.c
 - HAL_DSP_Cos, 324
 - HAL_DSP_Disable, 324
 - HAL_DSP_Div, 324
 - HAL_DSP_Enable, 324
 - HAL_DSP_Reset, 324
 - HAL_DSP_Sin, 325
 - HAL_DSP_SinCos, 325
 - HAL_DSP_Sqrt, 325
- hal_iks32mc09x_flash.c
 - HAL_FLASH_data, 337
 - HAL_FLASH_Disable, 329
 - HAL_FLASH_DisableCache, 329
 - HAL_FLASH_Enable, 329
 - HAL_FLASH_EnableCache, 330
 - HAL_FLASH_Erase, 330
 - HAL_FLASH_Program, 330
 - HAL_FLASH_Read, 331
 - HAL_FLASH_ReadSectorSize, 332
 - HAL_FLASH_ReadUint16, 332
 - HAL_FLASH_ReadUint32, 333
 - HAL_FLASH_ReadUint8, 333
 - HAL_FLASH_Reset, 334
 - HAL_FLASH_SecurityParameterCrc, 334
 - HAL_FLASH_SecurityParameterRead, 334
 - HAL_FLASH_SecurityParameterSave, 335
 - HAL_FLASH_Update, 336
- hal_iks32mc09x_gpio.c
 - HAL_GPIO_Disable, 342
 - HAL_GPIO_Enable, 342
 - HAL_GPIO_GetInData, 342
 - HAL_GPIO_GetOutData, 343
 - HAL_GPIO_Invert, 343
 - HAL_GPIO_Reset, 343
 - HAL_GPIO_RstReuseGpioDisable, 344
 - HAL_GPIO_RstReuseGpioEnable, 344
 - HAL_GPIO_SetAf, 344
 - HAL_GPIO_SetHigh, 345
 - HAL_GPIO_SetLow, 346
 - HAL_GPIO_SetMode, 346
 - HAL_GPIO_SwdReuseGpioDisable, 347
 - HAL_GPIO_SwdReuseGpioEnable, 347
 - HAL_GPIO_Write, 348
- hal_iks32mc09x_hall.c
 - HAL_HALL_Disable, 352
 - HAL_HALL_Enable, 352
 - HAL_HALL_ReadHallData, 353
 - HAL_HALL_ReadHallWidth, 353
 - HAL_HALL_ReadTh, 353
 - HAL_HALL_Reset, 354
 - HAL_HALL_SetFilter, 354
 - HAL_HALL_SetFilteruS, 354
 - HAL_HALL_SetFreq, 355

- HAL_HALL_SetGpio, 356
- HAL_HALL_SetTh, 356
- HAL_HALL_StartCounting, 357
- HAL_HALL_StopCounting, 357
- hal_iks32mc09x_i2c.c
 - hal_baud, 364
 - HAL_I2C_BusReset, 360
 - HAL_I2C_Disable, 360
 - HAL_I2C_Enable, 361
 - HAL_I2C_Reset, 361
 - HAL_I2C_RxData, 361
 - HAL_I2C_SetBaud, 362
 - HAL_I2C_SetGpioScl, 362
 - HAL_I2C_SetGpioSda, 363
 - HAL_I2C_TxData, 364
 - hal_i2cio, 364
- hal_iks32mc09x_init.c
 - HAL_INIT_PowerReady, 368
 - HAL_INIT_SysInit, 368
- hal_iks32mc09x_irq.c
 - HAL_IRQ_Disable, 369
 - HAL_IRQ_Enable, 370
 - HAL_IRQ_ModuleDisable, 370
 - HAL_IRQ_ModuleEnable, 370
 - HAL_IRQ_Reset, 371
- hal_iks32mc09x_irqfun.c
 - HAL_ADC_IrqSamp1, 373
 - HAL_ADC_IrqSamp1Clear, 374
 - HAL_ADC_IrqSamp1Disable, 374
 - HAL_ADC_IrqSamp1Enable, 374
 - HAL_ADC_IrqSamp1Flg, 375
 - HAL_ADC_IrqSamp2, 375
 - HAL_ADC_IrqSamp2Clear, 375
 - HAL_ADC_IrqSamp2Disable, 376
 - HAL_ADC_IrqSamp2Enable, 376
 - HAL_ADC_IrqSamp2Flg, 376
 - HAL_HALL_IrqChange, 376
 - HAL_HALL_IrqChangeClear, 377
 - HAL_HALL_IrqChangeDisable, 377
 - HAL_HALL_IrqChangeEnable, 377
 - HAL_HALL_IrqChangeFlg, 379
 - HAL_HALL_IrqOver, 379
 - HAL_HALL_IrqOverClear, 379
 - HAL_HALL_IrqOverDisable, 380
 - HAL_HALL_IrqOverEnable, 380
 - HAL_HALL_IrqOverFlg, 380
- hal_iks32mc09x_iwdg.c
 - HAL_IWDG_Disable, 383
 - HAL_IWDG_Enable, 384
 - HAL_IWDG_Feed, 384
 - HAL_IWDG_Reset, 385
 - HAL_IWDG_SetResetTimemS, 385
- hal_iks32mc09x_mcpwm.c
 - HAL_MCPWM_Disable, 388
 - HAL_MCPWM_Enable, 388
 - HAL_MCPWM_GetFreq, 389
 - HAL_MCPWM_GetThCnt, 389
 - HAL_MCPWM_IrqFail, 390
 - HAL_MCPWM_IrqFailClear, 390
 - HAL_MCPWM_IrqFailDisable, 390
 - HAL_MCPWM_IrqFailEnable, 390
 - HAL_MCPWM_IrqOverflow, 390
 - HAL_MCPWM_IrqOverflowClear, 391
 - HAL_MCPWM_IrqOverflowDisable, 391
 - HAL_MCPWM_IrqOverflowEnable, 391
 - HAL_MCPWM_IrqZeroCross, 391
 - HAL_MCPWM_IrqZeroCrossClear, 391
 - HAL_MCPWM_IrqZeroCrossDisable, 391
 - HAL_MCPWM_IrqZeroCrossEnable, 391
 - HAL_MCPWM_Reset, 392
 - HAL_MCPWM_SetAdcTrig, 392
 - HAL_MCPWM_SetAdcTrigQ15, 392
 - HAL_MCPWM_SetDead, 392
 - HAL_MCPWM_SetDefaultState, 393
 - HAL_MCPWM_SetDrivePolarity, 394
 - HAL_MCPWM_SetFailSignal, 394
 - HAL_MCPWM_SetFreq, 395
 - HAL_MCPWM_SetGpioFail, 395
 - HAL_MCPWM_SetGpioOut, 396
 - HAL_MCPWM_SetOutputDisable, 396
 - HAL_MCPWM_SetOutputEnable, 397
 - HAL_MCPWM_SetOutState, 397
 - HAL_MCPWM_SetPwm, 398
 - HAL_MCPWM_SetPwmQ15, 398
 - HAL_MCPWM_StartCounting, 399
 - HAL_MCPWM_StopCounting, 399
 - hal_MCPWM_UpdatePolarityState, 399
 - mcpwm_polarity_state, 401
- hal_iks32mc09x_opa.c
 - HAL_OPA_Disable, 408
 - HAL_OPA_Enable, 408
 - HAL_OPA_Out2GpioDisable, 408
 - HAL_OPA_Out2GpioEnable, 408
 - HAL_OPA_Reset, 409
 - HAL_OPA_SetGain, 409
 - HAL_OPA_SetGpioIn, 409
 - HAL_OPA_SetGpioOut, 410
- hal_iks32mc09x_qep.c
 - HAL_QEP_Disable, 413
 - HAL_QEP_Enable, 413
 - HAL_QEP_ReadCnt, 414
 - HAL_QEP_Reset, 414
 - HAL_QEP_ResetCnt, 415
 - HAL_QEP_SetGpio, 415
 - HAL_QEP_SetMode, 415
 - HAL_QEP_SetTh, 416
- hal_iks32mc09x_reg.c
 - HAL_ADC, 419
 - HAL_CMP, 419
 - HAL_CRC, 419
 - HAL_DMA, 419
 - HAL_DSP, 419
 - HAL_FLASH, 419
 - HAL_GPIO, 419
 - HAL_HALL, 419
 - HAL_I2C, 420

- HAL_IWDG, 420
- HAL_MCPWM, 420
- HAL_QEP, 420
- HAL_SPI, 420
- HAL_SYS, 420
- HAL_TIMER, 420
- HAL_UART, 420
- hal_lks32mc09x_reg.h
 - AON0, 91
 - CLU0, 91
 - DMA_CHN0, 91
 - DMA_CHN1, 91
 - DMA_CHN2, 91
 - DMA_CHN3, 91
 - DMA_TOP, 91
 - EXTI0, 91
 - HAL_ADC, 92
 - HAL_CMP, 92
 - HAL_CRC, 92
 - HAL_DMA, 92
 - HAL_DSP, 93
 - HAL_FLASH, 93
 - HAL_GPIO, 93
 - HAL_HALL, 93
 - HAL_I2C, 93
 - HAL_IWDG, 93
 - HAL_MCPWM, 93
 - HAL_QEP, 93
 - HAL_SPI, 94
 - HAL_SYS, 94
 - HAL_TIMER, 94
 - HAL_UART, 94
 - LKS32MC09X, 92
 - QEP0, 92
 - SIF0, 92
- hal_lks32mc09x_reset.c
 - HAL_RESET_ResetCore, 422
 - HAL_RESET_ResetGlobal, 422
 - HAL_RESET_ResetRegister, 422
- hal_lks32mc09x_spi.c
 - HAL_SPI_Disable, 425
 - HAL_SPI_Enable, 425
 - HAL_SPI_Reset, 425
 - HAL_SPI_RxData, 425
 - HAL_SPI_RxDataWidth16, 427
 - HAL_SPI_RxTxData, 427
 - HAL_SPI_RxTxDataWidth16, 428
 - HAL_SPI_SetBaud, 428
 - HAL_SPI_SetGpioClk, 429
 - HAL_SPI_SetGpioCs, 429
 - HAL_SPI_SetGpioDi, 430
 - HAL_SPI_SetGpioDo, 431
 - HAL_SPI_SetGpioMiso, 431
 - HAL_SPI_SetGpioMosi, 432
 - HAL_SPI_SetMode, 433
 - HAL_SPI_TxData, 433
 - HAL_SPI_TxDataWidth16, 433
- hal_lks32mc09x_test.c
 - HAL_TestRegAddr, 438
- hal_lks32mc09x_timer.c
 - HAL_TIMER_Disable, 440
 - HAL_TIMER_Enable, 440
 - HAL_TIMER_GetFreq, 440
 - HAL_TIMER_GetThCnt, 440
 - HAL_TIMER_IrqCompare0, 441
 - HAL_TIMER_IrqCompare0Clear, 441
 - HAL_TIMER_IrqCompare0Disable, 441
 - HAL_TIMER_IrqCompare0Enable, 441
 - HAL_TIMER_IrqCompare1, 441
 - HAL_TIMER_IrqCompare1Clear, 442
 - HAL_TIMER_IrqCompare1Disable, 442
 - HAL_TIMER_IrqCompare1Enable, 442
 - HAL_TIMER_IrqOverflow, 442
 - HAL_TIMER_IrqOverflowClear, 442
 - HAL_TIMER_IrqOverflowDisable, 442
 - HAL_TIMER_IrqOverflowEnable, 442
 - HAL_TIMER_Reset, 443
 - HAL_TIMER_SetFreq, 443
 - HAL_TIMER_SetGpioIn, 443
 - HAL_TIMER_SetGpioOut, 444
 - HAL_TIMER_SetPeriodCount, 445
 - HAL_TIMER_SetPeriodmS, 445
 - HAL_TIMER_SetPerioduS, 446
 - HAL_TIMER_SetPwm, 446
 - HAL_TIMER_SetPwmQ15, 446
 - HAL_TIMER_SetWorkMode, 447
 - HAL_TIMER_StartCounting, 447
 - HAL_TIMER_StopCounting, 447
- hal_lks32mc09x_tmp.c
 - HAL_TMP_Adc2Temperature, 453
 - HAL_TMP_Adc2Temperature01, 453
 - HAL_TMP_Disable, 454
 - HAL_TMP_Enable, 455
 - HAL_TMP_GetTemperature, 455
 - HAL_TMP_GetTemperature01, 456
 - HAL_TMP_Reset, 456
 - tmp_coef_a, 457
 - tmp_coef_b, 457
- hal_lks32mc09x_trim.c
 - HAL_TRIM_Read, 459
 - HAL_TRIM_ReadId, 460
 - HAL_TRIM_ReadIdCrc32, 461
 - HAL_TRIM_ReadMcu09, 461
- hal_lks32mc09x_uart.c
 - HAL_UART_Disable, 464
 - HAL_UART_Enable, 464
 - HAL_UART_IrqCheckErr, 465
 - HAL_UART_IrqCheckErrClear, 465
 - HAL_UART_IrqCheckErrDisable, 465
 - HAL_UART_IrqCheckErrEnable, 465
 - HAL_UART_IrqReceiveOver, 465
 - HAL_UART_IrqReceiveOverClear, 465
 - HAL_UART_IrqReceiveOverDisable, 465
 - HAL_UART_IrqReceiveOverEnable, 466
 - HAL_UART_IrqSendOver, 466
 - HAL_UART_IrqSendOverClear, 466

- HAL_UART_IrqSendOverDisable, 466
- HAL_UART_IrqSendOverEnable, 466
- HAL_UART_IrqStopErr, 466
- HAL_UART_IrqStopErrClear, 466
- HAL_UART_IrqStopErrDisable, 467
- HAL_UART_IrqStopErrEnable, 467
- HAL_UART_ReadBuffer, 467
- HAL_UART_Reset, 467
- HAL_UART_SetBaud, 467
- HAL_UART_SetGpioRx, 467
- HAL_UART_SetGpioTx, 468
- HAL_UART_SetMode, 468
- HAL_UART_TxData, 469
- HAL_UART_WriteBuffer, 469
- hal_ks32mc09x_wake.c
 - HAL_WAKE_DeepSleepToTimeWakeUp, 475
 - HAL_WAKE_DeepSleepToWake, 476
 - HAL_WAKE_DeepSleepTimeWakeUp, 476
- hal_ksmcu.h, 88
- hal_ksmcu_adc.h
 - HAL_ADC_0, 103
 - HAL_ADC_1, 103
 - HAL_ADC_2, 104
 - HAL_ADC_3, 104
 - HAL_ADC_Disable, 104
 - HAL_ADC_Enable, 104
 - HAL_ADC_EndSampling, 104
 - HAL_ADC_GetRange, 105
 - HAL_ADC_GetSamplingValues, 105
 - HAL_ADC_GetValuesUnit, 106
 - HAL_ADC_ManualSampling, 107
 - HAL_ADC_PORT_AGND, 101
 - HAL_ADC_PORT_CH0, 101
 - HAL_ADC_PORT_CH1, 101
 - HAL_ADC_PORT_CH10, 101
 - HAL_ADC_PORT_CH11, 101
 - HAL_ADC_PORT_CH12, 101
 - HAL_ADC_PORT_CH13, 101
 - HAL_ADC_PORT_CH14, 101
 - HAL_ADC_PORT_CH15, 101
 - HAL_ADC_PORT_CH16, 101
 - HAL_ADC_PORT_CH17, 101
 - HAL_ADC_PORT_CH18, 101
 - HAL_ADC_PORT_CH19, 101
 - HAL_ADC_PORT_CH2, 101
 - HAL_ADC_PORT_CH3, 101
 - HAL_ADC_PORT_CH4, 101
 - HAL_ADC_PORT_CH5, 101
 - HAL_ADC_PORT_CH6, 101
 - HAL_ADC_PORT_CH7, 101
 - HAL_ADC_PORT_CH8, 101
 - HAL_ADC_PORT_CH9, 101
 - HAL_ADC_PORT_DAC0, 101
 - HAL_ADC_PORT_DAC1, 101
 - HAL_ADC_PORT_LDO12, 101
 - HAL_ADC_PORT_OPA0, 101
 - HAL_ADC_PORT_OPA1, 101
 - HAL_ADC_PORT_OPA2, 101
 - HAL_ADC_PORT_OPA3, 101
 - HAL_ADC_PORT_OPA4, 101
 - HAL_ADC_PORT_OPA5, 101
 - HAL_ADC_PORT_REF, 101
 - HAL_ADC_PORT_TMP0, 101
 - HAL_ADC_PORT_VDD, 101
 - HAL_ADC_PORT_x, 101
 - HAL_ADC_REFERENCE_EXTERNAL, 102
 - HAL_ADC_REFERENCE_INTERNAL, 102
 - HAL_ADC_REFERENCE_x, 101
 - HAL_ADC_Reset, 107
 - HAL_ADC_SAMP_NUMBER_0, 102
 - HAL_ADC_SAMP_NUMBER_1, 102
 - HAL_ADC_SAMP_NUMBER_10, 102
 - HAL_ADC_SAMP_NUMBER_11, 102
 - HAL_ADC_SAMP_NUMBER_12, 102
 - HAL_ADC_SAMP_NUMBER_13, 102
 - HAL_ADC_SAMP_NUMBER_14, 102
 - HAL_ADC_SAMP_NUMBER_15, 102
 - HAL_ADC_SAMP_NUMBER_2, 102
 - HAL_ADC_SAMP_NUMBER_3, 102
 - HAL_ADC_SAMP_NUMBER_4, 102
 - HAL_ADC_SAMP_NUMBER_5, 102
 - HAL_ADC_SAMP_NUMBER_6, 102
 - HAL_ADC_SAMP_NUMBER_7, 102
 - HAL_ADC_SAMP_NUMBER_8, 102
 - HAL_ADC_SAMP_NUMBER_9, 102
 - HAL_ADC_SAMP_NUMBER_MAX, 102
 - HAL_ADC_SAMP_NUMBER_x, 102
 - HAL_ADC_SAMP_SEQUENCE_0, 102
 - HAL_ADC_SAMP_SEQUENCE_1, 103
 - HAL_ADC_SAMP_SEQUENCE_10, 103
 - HAL_ADC_SAMP_SEQUENCE_11, 103
 - HAL_ADC_SAMP_SEQUENCE_12, 103
 - HAL_ADC_SAMP_SEQUENCE_13, 103
 - HAL_ADC_SAMP_SEQUENCE_14, 103
 - HAL_ADC_SAMP_SEQUENCE_15, 103
 - HAL_ADC_SAMP_SEQUENCE_16, 103
 - HAL_ADC_SAMP_SEQUENCE_17, 103
 - HAL_ADC_SAMP_SEQUENCE_18, 103
 - HAL_ADC_SAMP_SEQUENCE_19, 103
 - HAL_ADC_SAMP_SEQUENCE_2, 103
 - HAL_ADC_SAMP_SEQUENCE_3, 103
 - HAL_ADC_SAMP_SEQUENCE_4, 103
 - HAL_ADC_SAMP_SEQUENCE_5, 103
 - HAL_ADC_SAMP_SEQUENCE_6, 103
 - HAL_ADC_SAMP_SEQUENCE_7, 103
 - HAL_ADC_SAMP_SEQUENCE_8, 103
 - HAL_ADC_SAMP_SEQUENCE_9, 103
 - HAL_ADC_SAMP_SEQUENCE_x, 102
 - HAL_ADC_SetGpioIn, 108
 - HAL_ADC_SetGpioTriggerOut, 108
 - HAL_ADC_SetReference, 109
 - HAL_ADC_SetSamplingChannel, 109
 - HAL_ADC_SetSamplingQuantity, 110
 - HAL_ADC_SetTriggerSignal, 111
 - HAL_ADC_SoftTrigger, 112
 - HAL_ADC_TRIGGER_MCPWM, 103

- HAL_ADC_TRIGGER_SOFT, 103
- HAL_ADC_TRIGGER_TIMER, 103
- HAL_ADC_TRIGGER_x, 103
- HAL_ADC_x, 103
- hal_lksmcu_clk.h
 - HAL_CLK_FREQ_RC_12MHZ, 116
 - HAL_CLK_FREQ_RC_16MHZ, 116
 - HAL_CLK_FREQ_RC_18MHZ, 116
 - HAL_CLK_FREQ_RC_192MHZ, 117
 - HAL_CLK_FREQ_RC_24MHZ, 116
 - HAL_CLK_FREQ_RC_32MHZ, 116
 - HAL_CLK_FREQ_RC_36MHZ, 116
 - HAL_CLK_FREQ_RC_48MHZ, 116
 - HAL_CLK_FREQ_RC_4MHZ, 117
 - HAL_CLK_FREQ_RC_64MHZ, 116
 - HAL_CLK_FREQ_RC_6MHZ, 117
 - HAL_CLK_FREQ_RC_72MHZ, 116
 - HAL_CLK_FREQ_RC_8MHZ, 116
 - HAL_CLK_FREQ_RC_96MHZ, 116
 - HAL_CLK_FREQ_RC_9MHZ, 116
 - HAL_CLK_FREQ_x, 116
 - HAL_CLK_FREQ_XTAL_12MHZ, 116
 - HAL_CLK_FREQ_XTAL_192MHZ, 117
 - HAL_CLK_FREQ_XTAL_24MHZ, 116
 - HAL_CLK_FREQ_XTAL_48MHZ, 116
 - HAL_CLK_FREQ_XTAL_6MHZ, 116
 - HAL_CLK_FREQ_XTAL_96MHZ, 117
 - HAL_CLK_GetMcuFreq, 117
 - HAL_CLK_SetMcuFreq, 118
- hal_lksmcu_cmp.h
 - HAL_CMP_0, 123
 - HAL_CMP_1, 123
 - HAL_CMP_2, 123
 - HAL_CMP_3, 123
 - HAL_CMP_4, 123
 - HAL_CMP_5, 123
 - HAL_CMP_CHANNEL_IN_0, 121
 - HAL_CMP_CHANNEL_IN_DAC, 121
 - HAL_CMP_CHANNEL_IN_MID, 121
 - HAL_CMP_CHANNEL_IN_REF, 121
 - HAL_CMP_CHANNEL_INx, 121
 - HAL_CMP_CHANNEL_IP0, 122
 - HAL_CMP_CHANNEL_IP1, 122
 - HAL_CMP_CHANNEL_IP2, 122
 - HAL_CMP_CHANNEL_IP3, 122
 - HAL_CMP_CHANNEL_IP4, 122
 - HAL_CMP_CHANNEL_IP5, 122
 - HAL_CMP_CHANNEL_IP6, 122
 - HAL_CMP_CHANNEL_IP7, 122
 - HAL_CMP_CHANNEL_IP_AVSS, 122
 - HAL_CMP_CHANNEL_IP_DAC0, 122
 - HAL_CMP_CHANNEL_IP_DAC1, 122
 - HAL_CMP_CHANNEL_IP_OPA0_IN, 122
 - HAL_CMP_CHANNEL_IP_OPA0_IP, 122
 - HAL_CMP_CHANNEL_IP_OPA0_OUTN, 122
 - HAL_CMP_CHANNEL_IP_OPA0_OUTP, 122
 - HAL_CMP_CHANNEL_IP_OPA1_IN, 122
 - HAL_CMP_CHANNEL_IP_OPA1_IP, 122
- HAL_CMP_CHANNEL_IP_OPA1_OUTN, 122
- HAL_CMP_CHANNEL_IP_OPA1_OUTP, 122
- HAL_CMP_CHANNEL_IP_OPA2_IN, 122
- HAL_CMP_CHANNEL_IP_OPA2_IP, 122
- HAL_CMP_CHANNEL_IP_OPA2_OUTN, 122
- HAL_CMP_CHANNEL_IP_OPA2_OUTP, 122
- HAL_CMP_CHANNEL_IP_OPA3_IN, 122
- HAL_CMP_CHANNEL_IP_OPA3_IP, 122
- HAL_CMP_CHANNEL_IP_OPA3_OUTN, 122
- HAL_CMP_CHANNEL_IP_OPA3_OUTP, 122
- HAL_CMP_CHANNEL_IP_OPA4_IN, 122
- HAL_CMP_CHANNEL_IP_OPA4_IP, 122
- HAL_CMP_CHANNEL_IP_OPA4_OUTN, 122
- HAL_CMP_CHANNEL_IP_OPA4_OUTP, 122
- HAL_CMP_CHANNEL_IP_OPA5_IN, 122
- HAL_CMP_CHANNEL_IP_OPA5_IP, 122
- HAL_CMP_CHANNEL_IP_OPA5_OUTN, 122
- HAL_CMP_CHANNEL_IP_OPA5_OUTP, 122
- HAL_CMP_CHANNEL_IP_OPA_OUT_CMP0, 122
- HAL_CMP_CHANNEL_IPx, 121
- HAL_CMP_Disable, 123
- HAL_CMP_Enable, 123
- HAL_CMP_GetData, 124
- HAL_CMP_Reset, 124
- HAL_CMP_SetChannel, 125
- HAL_CMP_SetDigitalFilter, 125
- HAL_CMP_SetGpioIn, 125
- HAL_CMP_SetGpioOut, 126
- HAL_CMP_x, 122
- hal_lksmcu_crc.h
 - HAL_CRC_0, 129
 - HAL_CRC_Calc, 129
 - HAL_CRC_Disable, 130
 - HAL_CRC_Enable, 130
 - HAL_CRC_MODEL_CRC16_CCITT, 129
 - HAL_CRC_MODEL_CRC16_CCITT_FALSE, 129
 - HAL_CRC_MODEL_CRC16_DNP, 129
 - HAL_CRC_MODEL_CRC16_IBMx1, 129
 - HAL_CRC_MODEL_CRC16_MAXIM, 129
 - HAL_CRC_MODEL_CRC16_MODBU, 129
 - HAL_CRC_MODEL_CRC16_USBx1, 129
 - HAL_CRC_MODEL_CRC16_X25, 129
 - HAL_CRC_MODEL_CRC16_XMODEM, 129
 - HAL_CRC_MODEL_CRC32, 129
 - HAL_CRC_MODEL_CRC32_MPEG_2, 129
 - HAL_CRC_MODEL_CRC8, 129
 - HAL_CRC_MODEL_CRC8_ITU, 129
 - HAL_CRC_MODEL_CRC8_MAXIMx, 129
 - HAL_CRC_MODEL_CRC8_ROHC, 129
 - HAL_CRC_MODEL_x, 129
 - HAL_CRC_Reset, 131
 - HAL_CRC_x, 129
- hal_lksmcu_dac.h
 - HAL_DAC_0, 133
 - HAL_DAC_1, 133
 - HAL_DAC_2, 133
 - HAL_DAC_3, 133
 - HAL_DAC_Disable, 133

- HAL_DAC.Enable, 133
- HAL_DAC.Out2GpioDisable, 133
- HAL_DAC.Out2GpioEnable, 134
- HAL_DAC.Reset, 134
- HAL_DAC.SetGpioOut, 134
- HAL_DAC.SetOutVal, 135
- HAL_DAC.SetOutVoltageV, 135
- HAL_DAC_x, 133
- hal_lksmcu_dma.h
 - HAL_DMA_0, 137
 - HAL_DMA_1, 137
 - HAL_DMA_2, 137
 - HAL_DMA_3, 137
 - HAL_DMA.Disable, 137
 - HAL_DMA.Enable, 137
 - HAL_DMA.I2cRx, 138
 - HAL_DMA.I2cTx, 138
 - HAL_DMA.MemCopy, 138
 - HAL_DMA.Reset, 138
 - HAL_DMA.SpiRx, 138
 - HAL_DMA.SpiRxTx, 138
 - HAL_DMA.SpiTx, 139
 - HAL_DMA.UartRx, 139
 - HAL_DMA.UartTx, 139
 - HAL_DMA_x, 137
- hal_lksmcu_dsp.h
 - HAL_DSP_0, 141
 - HAL_DSP.Cos, 141
 - HAL_DSP.Disable, 142
 - HAL_DSP.Div, 142
 - HAL_DSP.Enable, 142
 - HAL_DSP.Reset, 142
 - HAL_DSP.Sin, 142
 - HAL_DSP.SinCos, 143
 - HAL_DSP.Sqrt, 143
 - HAL_DSP_x, 141
- hal_lksmcu_flash.h
 - HAL_FLASH_0, 145
 - HAL_FLASH_1, 145
 - HAL_FLASH.Disable, 146
 - HAL_FLASH.DisableCache, 146
 - HAL_FLASH.Enable, 146
 - HAL_FLASH.EnableCache, 146
 - HAL_FLASH.Erase, 147
 - HAL_FLASH.Program, 147
 - HAL_FLASH.Read, 148
 - HAL_FLASH.ReadSectorSize, 148
 - HAL_FLASH.ReadUint16, 149
 - HAL_FLASH.ReadUint32, 149
 - HAL_FLASH.ReadUint8, 150
 - HAL_FLASH.Reset, 150
 - HAL_FLASH.SecurityParameterRead, 151
 - HAL_FLASH.SecurityParameterSave, 151
 - HAL_FLASH.Update, 152
 - HAL_FLASH_x, 145
- hal_lksmcu_gpio.h
 - HAL_GPIO_AF_ADC, 156
 - HAL_GPIO_AF_ADC.TRIG, 157
 - HAL_GPIO_AF_CAN, 157
 - HAL_GPIO_AF_CL, 157
 - HAL_GPIO_AF_CLK, 156
 - HAL_GPIO_AF_CLK_OUT, 156
 - HAL_GPIO_AF_CMP, 156
 - HAL_GPIO_AF_CMP_OUT, 156
 - HAL_GPIO_AF_DAC, 156
 - HAL_GPIO_AF_GPIO, 156
 - HAL_GPIO_AF_HALL, 156
 - HAL_GPIO_AF_I2C, 156
 - HAL_GPIO_AF.MCPWM, 156
 - HAL_GPIO_AF.OPA, 156
 - HAL_GPIO_AF.OPA_OUT, 156
 - HAL_GPIO_AF.QEP, 157
 - HAL_GPIO_AF.SIF, 157
 - HAL_GPIO_AF.SPI, 156
 - HAL_GPIO_AF.TIMER0, 156
 - HAL_GPIO_AF.TIMER1, 156
 - HAL_GPIO_AF.TIMER2, 156
 - HAL_GPIO_AF.TIMER3, 157
 - HAL_GPIO_AF.TIMER4, 157
 - HAL_GPIO_AF.TIMER5, 157
 - HAL_GPIO_AF.UART, 156
 - HAL_GPIO_AFx, 156
 - HAL_GPIO.Disable, 160
 - HAL_GPIO.Enable, 160
 - HAL_GPIO.GetInData, 160
 - HAL_GPIO.GetOutData, 160
 - HAL_GPIO.Invert, 161
 - HAL_GPIO.MODE_AFE, 157
 - HAL_GPIO.MODE_IN, 157
 - HAL_GPIO.MODE.INOUT, 157
 - HAL_GPIO.MODE.INOUT_0, 157
 - HAL_GPIO.MODE.INOUT_1, 157
 - HAL_GPIO.MODE.INUP, 157
 - HAL_GPIO.MODE.INUPOUT, 157
 - HAL_GPIO.MODE.INUPOUT_0, 157
 - HAL_GPIO.MODE.INUPOUT_1, 157
 - HAL_GPIO.MODE.NONE, 157
 - HAL_GPIO.MODE.OUT, 157
 - HAL_GPIO.MODE.OUT_0, 157
 - HAL_GPIO.MODE.OUT_1, 157
 - HAL_GPIO.MODE_x, 157
 - HAL_GPIO.P0_0, 158
 - HAL_GPIO.P0_1, 158
 - HAL_GPIO.P0_10, 158
 - HAL_GPIO.P0_11, 158
 - HAL_GPIO.P0_12, 158
 - HAL_GPIO.P0_13, 158
 - HAL_GPIO.P0_14, 158
 - HAL_GPIO.P0_15, 158
 - HAL_GPIO.P0_2, 158
 - HAL_GPIO.P0_3, 158
 - HAL_GPIO.P0_4, 158
 - HAL_GPIO.P0_5, 158
 - HAL_GPIO.P0_6, 158
 - HAL_GPIO.P0_7, 158
 - HAL_GPIO.P0_8, 158

HAL_GPIO_P0.9, 158
HAL_GPIO_P1.0, 158
HAL_GPIO_P1.1, 158
HAL_GPIO_P1.10, 158
HAL_GPIO_P1.11, 158
HAL_GPIO_P1.12, 158
HAL_GPIO_P1.13, 158
HAL_GPIO_P1.14, 158
HAL_GPIO_P1.15, 158
HAL_GPIO_P1.2, 158
HAL_GPIO_P1.3, 158
HAL_GPIO_P1.4, 158
HAL_GPIO_P1.5, 158
HAL_GPIO_P1.6, 158
HAL_GPIO_P1.7, 158
HAL_GPIO_P1.8, 158
HAL_GPIO_P1.9, 158
HAL_GPIO_P2.0, 158
HAL_GPIO_P2.1, 158
HAL_GPIO_P2.10, 158
HAL_GPIO_P2.11, 158
HAL_GPIO_P2.12, 158
HAL_GPIO_P2.13, 158
HAL_GPIO_P2.14, 158
HAL_GPIO_P2.15, 158
HAL_GPIO_P2.2, 158
HAL_GPIO_P2.3, 158
HAL_GPIO_P2.4, 158
HAL_GPIO_P2.5, 158
HAL_GPIO_P2.6, 158
HAL_GPIO_P2.7, 158
HAL_GPIO_P2.8, 158
HAL_GPIO_P2.9, 158
HAL_GPIO_P3.0, 158
HAL_GPIO_P3.1, 158
HAL_GPIO_P3.10, 159
HAL_GPIO_P3.11, 159
HAL_GPIO_P3.12, 159
HAL_GPIO_P3.13, 159
HAL_GPIO_P3.14, 159
HAL_GPIO_P3.15, 159
HAL_GPIO_P3.2, 159
HAL_GPIO_P3.3, 159
HAL_GPIO_P3.4, 159
HAL_GPIO_P3.5, 159
HAL_GPIO_P3.6, 159
HAL_GPIO_P3.7, 159
HAL_GPIO_P3.8, 159
HAL_GPIO_P3.9, 159
HAL_GPIO_P4.0, 159
HAL_GPIO_P4.1, 159
HAL_GPIO_P4.10, 159
HAL_GPIO_P4.11, 159
HAL_GPIO_P4.12, 159
HAL_GPIO_P4.13, 159
HAL_GPIO_P4.14, 159
HAL_GPIO_P4.15, 159
HAL_GPIO_P4.2, 159
HAL_GPIO_P4.3, 159
HAL_GPIO_P4.4, 159
HAL_GPIO_P4.5, 159
HAL_GPIO_P4.6, 159
HAL_GPIO_P4.7, 159
HAL_GPIO_P4.8, 159
HAL_GPIO_P4.9, 159
HAL_GPIO_P5.0, 159
HAL_GPIO_P5.1, 159
HAL_GPIO_P5.10, 159
HAL_GPIO_P5.11, 159
HAL_GPIO_P5.12, 159
HAL_GPIO_P5.13, 159
HAL_GPIO_P5.14, 159
HAL_GPIO_P5.15, 159
HAL_GPIO_P5.2, 159
HAL_GPIO_P5.3, 159
HAL_GPIO_P5.4, 159
HAL_GPIO_P5.5, 159
HAL_GPIO_P5.6, 159
HAL_GPIO_P5.7, 159
HAL_GPIO_P5.8, 159
HAL_GPIO_P5.9, 159
HAL_GPIO_Px, 157
HAL_GPIO_Reset, 161
HAL_GPIO_RstReuseGpioDisable, 161
HAL_GPIO_RstReuseGpioEnable, 161
HAL_GPIO_SetAf, 162
HAL_GPIO_SetHigh, 163
HAL_GPIO_SetLow, 164
HAL_GPIO_SetMode, 164
HAL_GPIO_SwdReuseGpioDisable, 165
HAL_GPIO_SwdReuseGpioEnable, 165
HAL_GPIO_Write, 166
hal.lksmcu_hall.h
HAL_HALL_0, 170
HAL_HALL_1, 170
HAL_HALL_Disable, 170
HAL_HALL_Enable, 170
HAL_HALL_ReadHallData, 171
HAL_HALL_ReadHallWidth, 171
HAL_HALL_ReadTh, 171
HAL_HALL_Reset, 172
HAL_HALL_SetFilter, 172
HAL_HALL_SetFilteruS, 172
HAL_HALL_SetFreq, 173
HAL_HALL_SetGpio, 174
HAL_HALL_SetTh, 174
HAL_HALL_StartCounting, 175
HAL_HALL_StopCounting, 175
HAL_HALL_x, 170
hal.lksmcu_i2c.h
HAL_I2C_0, 177
HAL_I2C_1, 177
HAL_I2C_BusReset, 177
HAL_I2C_Disable, 178
HAL_I2C_Enable, 178
HAL_I2C_Reset, 178

- HAL_I2C_RxData, 179
- HAL_I2C_SetBaud, 179
- HAL_I2C_SetGpioScl, 180
- HAL_I2C_SetGpioSda, 180
- HAL_I2C_TxData, 181
- HAL_I2C_x, 177
- hal_lksmcu_init.h
 - HAL_INIT_POWER_3.3V, 183
 - HAL_INIT_POWER_5V, 183
 - HAL_INIT_POWER_x, 182
 - HAL_INIT_PowerReady, 183
 - HAL_INIT_SysInit, 183
- hal_lksmcu_irq.h
 - HAL_IRQ_Disable, 185
 - HAL_IRQ_Enable, 186
 - HAL_IRQ_MODULE_ADC_00, 185
 - HAL_IRQ_MODULE_ADC_10, 185
 - HAL_IRQ_MODULE_CL_00, 185
 - HAL_IRQ_MODULE_CMP_00, 185
 - HAL_IRQ_MODULE_CMP_10, 185
 - HAL_IRQ_MODULE_DMA_00, 185
 - HAL_IRQ_MODULE_GPIO_00, 185
 - HAL_IRQ_MODULE_HALL_00, 185
 - HAL_IRQ_MODULE_HardFault, 185
 - HAL_IRQ_MODULE_I2C_00, 185
 - HAL_IRQ_MODULE_INIT_00, 185
 - HAL_IRQ_MODULE_MCPWM_00, 185
 - HAL_IRQ_MODULE_MCPWM_01, 185
 - HAL_IRQ_MODULE_NonMaskableInt, 184
 - HAL_IRQ_MODULE_PendSV, 185
 - HAL_IRQ_MODULE_QEP_00, 185
 - HAL_IRQ_MODULE_SIF_00, 185
 - HAL_IRQ_MODULE_SPI_00, 185
 - HAL_IRQ_MODULE_SVCall, 185
 - HAL_IRQ_MODULE_SW_00, 185
 - HAL_IRQ_MODULE_SysTick, 185
 - HAL_IRQ_MODULE_TIMER_00, 185
 - HAL_IRQ_MODULE_TIMER_10, 185
 - HAL_IRQ_MODULE_TIMER_20, 185
 - HAL_IRQ_MODULE_UART_00, 185
 - HAL_IRQ_MODULE_UART_10, 185
 - HAL_IRQ_MODULE_UART_20, 185
 - HAL_IRQ_MODULE_WAKE_00, 185
 - HAL_IRQ_MODULE_x, 184
 - HAL_IRQ_ModuleDisable, 186
 - HAL_IRQ_ModuleEnable, 186
 - HAL_IRQ_Reset, 187
- hal_lksmcu_irqfun.h
 - HAL_ADC_IrqSamp1, 189
 - HAL_ADC_IrqSamp1Clear, 190
 - HAL_ADC_IrqSamp1Disable, 190
 - HAL_ADC_IrqSamp1Enable, 190
 - HAL_ADC_IrqSamp1Flg, 190
 - HAL_ADC_IrqSamp2, 191
 - HAL_ADC_IrqSamp2Clear, 191
 - HAL_ADC_IrqSamp2Disable, 191
 - HAL_ADC_IrqSamp2Enable, 192
 - HAL_ADC_IrqSamp2Flg, 192
- HAL_HALL_IrqChange, 192
- HAL_HALL_IrqChangeClear, 193
- HAL_HALL_IrqChangeDisable, 193
- HAL_HALL_IrqChangeEnable, 193
- HAL_HALL_IrqChangeFlg, 193
- HAL_HALL_IrqOver, 194
- HAL_HALL_IrqOverClear, 194
- HAL_HALL_IrqOverDisable, 194
- HAL_HALL_IrqOverEnable, 195
- HAL_HALL_IrqOverFlg, 195
- hal_lksmcu_iwdg.h
 - HAL_IWDG_0, 197
 - HAL_IWDG_Disable, 197
 - HAL_IWDG_Enable, 198
 - HAL_IWDG_Feed, 198
 - HAL_IWDG_Reset, 199
 - HAL_IWDG_SetResetTimemS, 199
 - HAL_IWDG_x, 197
- hal_lksmcu_mcpwm.h
 - HAL_MCPWM_0, 204
 - HAL_MCPWM_1, 204
 - HAL_MCPWM_DEFAULT_OUTPUT_STATE_HIGH, 202
 - HAL_MCPWM_DEFAULT_OUTPUT_STATE_LOW, 202
 - HAL_MCPWM_DEFAULT_OUTPUT_STATE_OFF, 202
 - HAL_MCPWM_DEFAULT_OUTPUT_STATE_x, 202
 - HAL_MCPWM_Disable, 204
 - HAL_MCPWM_Enable, 204
 - HAL_MCPWM_FAIL_CMP0_HIGH, 202
 - HAL_MCPWM_FAIL_CMP0_LOW, 202
 - HAL_MCPWM_FAIL_CMP1_HIGH, 202
 - HAL_MCPWM_FAIL_CMP1_LOW, 202
 - HAL_MCPWM_FAIL_IO0_HIGH, 202
 - HAL_MCPWM_FAIL_IO0_LOW, 202
 - HAL_MCPWM_FAIL_IO1_HIGH, 202
 - HAL_MCPWM_FAIL_IO1_LOW, 202
 - HAL_MCPWM_FAIL_x, 202
 - HAL_MCPWM_GetFreq, 204
 - HAL_MCPWM_GetThCnt, 206
 - HAL_MCPWM_IrqFail, 206
 - HAL_MCPWM_IrqFailClear, 207
 - HAL_MCPWM_IrqFailDisable, 207
 - HAL_MCPWM_IrqFailEnable, 207
 - HAL_MCPWM_IrqOverflow, 207
 - HAL_MCPWM_IrqOverflowClear, 207
 - HAL_MCPWM_IrqOverflowDisable, 207
 - HAL_MCPWM_IrqOverflowEnable, 207
 - HAL_MCPWM_IrqZeroCross, 208
 - HAL_MCPWM_IrqZeroCrossClear, 208
 - HAL_MCPWM_IrqZeroCrossDisable, 208
 - HAL_MCPWM_IrqZeroCrossEnable, 208
 - HAL_MCPWM_OUTPUT_STATE_HIGH, 203
 - HAL_MCPWM_OUTPUT_STATE_HIGH_PWM, 203
 - HAL_MCPWM_OUTPUT_STATE_LOW, 203
 - HAL_MCPWM_OUTPUT_STATE_LOW_PWM, 203
 - HAL_MCPWM_OUTPUT_STATE_OFF, 203

- HAL_MCPWM_OUTPUT_STATE_PWM, 203
- HAL_MCPWM_OUTPUT_STATE_x, 202
- HAL_MCPWM_POLARITY_PH_NH, 203
- HAL_MCPWM_POLARITY_PH_NL, 203
- HAL_MCPWM_POLARITY_PL_NH, 203
- HAL_MCPWM_POLARITY_PL_NL, 203
- HAL_MCPWM_POLARITY_x, 203
- HAL_MCPWM_PORT_0, 203
- HAL_MCPWM_PORT_1, 203
- HAL_MCPWM_PORT_2, 203
- HAL_MCPWM_PORT_3, 203
- HAL_MCPWM_PORT_4, 203
- HAL_MCPWM_PORT_5, 203
- HAL_MCPWM_PORT_x, 203
- HAL_MCPWM_Reset, 208
- HAL_MCPWM_SetAdcTrig, 209
- HAL_MCPWM_SetAdcTrigQ15, 209
- HAL_MCPWM_SetDead, 209
- HAL_MCPWM_SetDefaultState, 210
- HAL_MCPWM_SetDrivePolarity, 210
- HAL_MCPWM_SetFailSignal, 211
- HAL_MCPWM_SetFreq, 211
- HAL_MCPWM_SetGpioFail, 212
- HAL_MCPWM_SetGpioOut, 212
- HAL_MCPWM_SetOutputDisable, 213
- HAL_MCPWM_SetOutputEnable, 213
- HAL_MCPWM_SetOutState, 214
- HAL_MCPWM_SetPwm, 214
- HAL_MCPWM_SetPwmQ15, 215
- HAL_MCPWM_StartCounting, 215
- HAL_MCPWM_StopCounting, 215
- HAL_MCPWM_x, 203
- hal_lksmcu_opa.h
 - HAL_OPA_0, 220
 - HAL_OPA_0B, 220
 - HAL_OPA_1, 220
 - HAL_OPA_2, 220
 - HAL_OPA_3, 220
 - HAL_OPA_4, 220
 - HAL_OPA_5, 220
 - HAL_OPA_Disable, 221
 - HAL_OPA_Enable, 221
 - HAL_OPA_GAIN_160_10, 220
 - HAL_OPA_GAIN_160_40, 220
 - HAL_OPA_GAIN_170_40, 220
 - HAL_OPA_GAIN_170_40_4, 220
 - HAL_OPA_GAIN_170_40_6, 220
 - HAL_OPA_GAIN_180_30, 220
 - HAL_OPA_GAIN_180_30_4, 220
 - HAL_OPA_GAIN_180_30_6, 220
 - HAL_OPA_GAIN_190_20, 220
 - HAL_OPA_GAIN_190_20_4, 220
 - HAL_OPA_GAIN_190_20_6, 220
 - HAL_OPA_GAIN_200_10, 220
 - HAL_OPA_GAIN_200_10_4, 220
 - HAL_OPA_GAIN_200_10_6, 220
 - HAL_OPA_GAIN_20_10, 220
 - HAL_OPA_GAIN_320_10, 220
 - HAL_OPA_GAIN_320_40, 220
 - HAL_OPA_GAIN_320_5, 220
 - HAL_OPA_GAIN_40_10, 220
 - HAL_OPA_GAIN_640_40, 220
 - HAL_OPA_GAIN_660_20, 220
 - HAL_OPA_GAIN_80_10, 220
 - HAL_OPA_GAIN_HIGH, 220
 - HAL_OPA_GAIN_LOW, 220
 - HAL_OPA_GAIN_MEDIUM, 220
 - HAL_OPA_GAIN_x, 219
 - HAL_OPA_OPAX_2_ADCCHN, 219
 - HAL_OPA_Out2GpioDisable, 221
 - HAL_OPA_Out2GpioEnable, 221
 - HAL_OPA_READ_GAIN, 219
 - HAL_OPA_Reset, 222
 - HAL_OPA_SetGain, 222
 - HAL_OPA_SetGpioIn, 222
 - HAL_OPA_SetGpioOut, 223
 - HAL_OPA_x, 220
- hal_lksmcu_qep.h
 - HAL_QEP_0, 226
 - HAL_QEP_1, 226
 - HAL_QEP_2, 226
 - HAL_QEP_3, 226
 - HAL_QEP_Disable, 226
 - HAL_QEP_Enable, 226
 - HAL_QEP_MODE_DUAL_PULSE, 226
 - HAL_QEP_MODE_QUADRATURE, 226
 - HAL_QEP_MODE_SYMBOL_PULSE, 226
 - HAL_QEP_MODE_x, 226
 - HAL_QEP_ReadCnt, 227
 - HAL_QEP_Reset, 227
 - HAL_QEP_ResetCnt, 228
 - HAL_QEP_SetGpio, 228
 - HAL_QEP_SetMode, 228
 - HAL_QEP_SetTh, 229
 - HAL_QEP_x, 226
- hal_lksmcu_reg.h
 - HAL_REG_IF_WRITE, 230
 - HAL_REG_INVERT, 230
 - HAL_REG_READ, 231
 - HAL_REG_READ_BIT, 231
 - HAL_REG_READ_BIT_POS_MASK, 231
 - HAL_REG_RESET, 231
 - HAL_REG_SET, 231
 - HAL_REG_SYS_LOCK, 231
 - HAL_REG_SYS_UNLOCK, 232
 - HAL_REG_WRITE, 232
 - HAL_REG_WRITE_BIT, 232
 - HAL_REG_WRITE_BIT_MASK_POS, 232
- hal_lksmcu_reset.h
 - HAL_RESET_ResetCore, 233
 - HAL_RESET_ResetGlobal, 233
 - HAL_RESET_ResetRegister, 233
- hal_lksmcu_spi.h
 - HAL_SPI_0, 236
 - HAL_SPI_1, 236
 - HAL_SPI_Disable, 237

- HAL_SPI_Enable, 237
- HAL_SPI_MODE_MASTER_LSB_00, 236
- HAL_SPI_MODE_MASTER_LSB_01, 236
- HAL_SPI_MODE_MASTER_LSB_10, 236
- HAL_SPI_MODE_MASTER_LSB_11, 236
- HAL_SPI_MODE_MASTER_MSB_00, 236
- HAL_SPI_MODE_MASTER_MSB_01, 236
- HAL_SPI_MODE_MASTER_MSB_10, 236
- HAL_SPI_MODE_MASTER_MSB_11, 236
- HAL_SPI_MODE_SLAVE_LSB_00, 236
- HAL_SPI_MODE_SLAVE_LSB_01, 236
- HAL_SPI_MODE_SLAVE_LSB_10, 236
- HAL_SPI_MODE_SLAVE_LSB_11, 236
- HAL_SPI_MODE_SLAVE_MSB_00, 236
- HAL_SPI_MODE_SLAVE_MSB_01, 236
- HAL_SPI_MODE_SLAVE_MSB_10, 236
- HAL_SPI_MODE_SLAVE_MSB_11, 236
- HAL_SPI_MODE_x, 236
- HAL_SPI_Reset, 237
- HAL_SPI_RxData, 237
- HAL_SPI_RxDataWidth16, 238
- HAL_SPI_RxTxData, 238
- HAL_SPI_RxTxDataWidth16, 239
- HAL_SPI_SetBaud, 239
- HAL_SPI_SetGpioClk, 240
- HAL_SPI_SetGpioCs, 240
- HAL_SPI_SetGpioDi, 241
- HAL_SPI_SetGpioDo, 242
- HAL_SPI_SetGpioMiso, 242
- HAL_SPI_SetGpioMosi, 243
- HAL_SPI_SetMode, 244
- HAL_SPI_TxData, 244
- HAL_SPI_TxDataWidth16, 244
- HAL_SPI_x, 236
- hal_lksmcu_timer.h
 - HAL_TIMER_0, 248
 - HAL_TIMER_1, 248
 - HAL_TIMER_2, 248
 - HAL_TIMER_3, 248
 - HAL_TIMER_4, 248
 - HAL_TIMER_Disable, 248
 - HAL_TIMER_Enable, 248
 - HAL_TIMER_GetFreq, 248
 - HAL_TIMER_GetThCnt, 249
 - HAL_TIMER_IrqCompare0, 249
 - HAL_TIMER_IrqCompare0Clear, 249
 - HAL_TIMER_IrqCompare0Disable, 249
 - HAL_TIMER_IrqCompare0Enable, 250
 - HAL_TIMER_IrqCompare1, 250
 - HAL_TIMER_IrqCompare1Clear, 250
 - HAL_TIMER_IrqCompare1Disable, 250
 - HAL_TIMER_IrqCompare1Enable, 250
 - HAL_TIMER_IrqOverflow, 250
 - HAL_TIMER_IrqOverflowClear, 250
 - HAL_TIMER_IrqOverflowDisable, 251
 - HAL_TIMER_IrqOverflowEnable, 251
 - HAL_TIMER_MODE_COUNT, 247
 - HAL_TIMER_MODE_PWMIN, 247
 - HAL_TIMER_MODE_PWMOUT, 247
 - HAL_TIMER_MODE_x, 247
 - HAL_TIMER_Reset, 251
 - HAL_TIMER_SetFreq, 251
 - HAL_TIMER_SetGpioIn, 252
 - HAL_TIMER_SetGpioOut, 252
 - HAL_TIMER_SetPeriodCount, 253
 - HAL_TIMER_SetPeriodmS, 254
 - HAL_TIMER_SetPerioduS, 254
 - HAL_TIMER_SetPwm, 255
 - HAL_TIMER_SetPwmQ15, 255
 - HAL_TIMER_SetWorkMode, 255
 - HAL_TIMER_StartCounting, 256
 - HAL_TIMER_StopCounting, 256
 - HAL_TIMER_x, 247
- hal_lksmcu_tmp.h
 - HAL_TMP_0, 258
 - HAL_TMP_1, 258
 - HAL_TMP_Adc2Temperature, 258
 - HAL_TMP_Adc2Temperature01, 259
 - HAL_TMP_Disable, 260
 - HAL_TMP_Enable, 260
 - HAL_TMP_GetTemperature, 261
 - HAL_TMP_GetTemperature01, 261
 - HAL_TMP_Reset, 262
 - HAL_TMP_x, 258
- hal_lksmcu_trim.h
 - HAL_TRIM_Read, 264
 - HAL_TRIM_ReadId, 264
 - HAL_TRIM_ReadIdCrc32, 265
- hal_lksmcu_uart.h
 - HAL_UART_0, 268
 - HAL_UART_1, 268
 - HAL_UART_2, 268
 - HAL_UART_Disable, 268
 - HAL_UART_Enable, 268
 - HAL_UART_IrqCheckErr, 268
 - HAL_UART_IrqCheckErrClear, 268
 - HAL_UART_IrqCheckErrDisable, 269
 - HAL_UART_IrqCheckErrEnable, 269
 - HAL_UART_IrqReceiveOver, 269
 - HAL_UART_IrqReceiveOverClear, 269
 - HAL_UART_IrqReceiveOverDisable, 269
 - HAL_UART_IrqReceiveOverEnable, 269
 - HAL_UART_IrqSendOver, 269
 - HAL_UART_IrqSendOverClear, 270
 - HAL_UART_IrqSendOverDisable, 270
 - HAL_UART_IrqSendOverEnable, 270
 - HAL_UART_IrqStopErr, 270
 - HAL_UART_IrqStopErrClear, 270
 - HAL_UART_IrqStopErrDisable, 270
 - HAL_UART_IrqStopErrEnable, 270
 - HAL_UART_MODE_LSB_STOP1_EVEN, 267
 - HAL_UART_MODE_LSB_STOP1_NONE, 267
 - HAL_UART_MODE_LSB_STOP1_ODD, 267
 - HAL_UART_MODE_LSB_STOP2_EVEN, 267
 - HAL_UART_MODE_LSB_STOP2_NONE, 267
 - HAL_UART_MODE_LSB_STOP2_ODD, 267

- HAL_UART_MODE_MSB_STOP1_EVEN, 268
- HAL_UART_MODE_MSB_STOP1_NONE, 268
- HAL_UART_MODE_MSB_STOP1_ODD, 268
- HAL_UART_MODE_MSB_STOP2_EVEN, 268
- HAL_UART_MODE_MSB_STOP2_NONE, 268
- HAL_UART_MODE_MSB_STOP2_ODD, 268
- HAL_UART_MODE_x, 267
- HAL_UART_ReadBuffer, 271
- HAL_UART_Reset, 271
- HAL_UART_SetBaud, 271
- HAL_UART_SetGpioRx, 271
- HAL_UART_SetGpioTx, 272
- HAL_UART_SetMode, 272
- HAL_UART_TxData, 273
- HAL_UART_WriteBuffer, 273
- HAL_UART_x, 268
- hal_iksmcu_wake.h
 - HAL_WAKE_DeepSleepIoTimeWakeUp, 276
 - HAL_WAKE_DeepSleepIoWake, 276
 - HAL_WAKE_DeepSleepTimeWakeUp, 277
 - HAL_WAKE_UP_IO_0_High, 276
 - HAL_WAKE_UP_IO_0_Low, 276
 - HAL_WAKE_UP_IO_1_High, 276
 - HAL_WAKE_UP_IO_1_Low, 276
 - HAL_WAKE_UP_IO_2_High, 276
 - HAL_WAKE_UP_IO_2_Low, 276
 - HAL_WAKE_UP_IO_3_High, 276
 - HAL_WAKE_UP_IO_3_Low, 276
 - HAL_WAKE_UP_IO_4_High, 276
 - HAL_WAKE_UP_IO_4_Low, 276
 - HAL_WAKE_UP_IO_5_High, 276
 - HAL_WAKE_UP_IO_5_Low, 276
 - HAL_WAKE_UP_IO_6_High, 276
 - HAL_WAKE_UP_IO_6_Low, 276
 - HAL_WAKE_UP_IO_7_High, 276
 - HAL_WAKE_UP_IO_7_Low, 276
 - HAL_WAKE_UP_IO_x, 275
- HAL_MCPWM
 - hal_iks32mc09x_reg.c, 420
 - hal_iks32mc09x_reg.h, 93
- HAL_MCPWM_0
 - hal_iksmcu_mcpwm.h, 204
- HAL_MCPWM_1
 - hal_iksmcu_mcpwm.h, 204
- HAL_MCPWM_DEFAULT_OUTPUT_STATE_HIGH
 - hal_iksmcu_mcpwm.h, 202
- HAL_MCPWM_DEFAULT_OUTPUT_STATE_LOW
 - hal_iksmcu_mcpwm.h, 202
- HAL_MCPWM_DEFAULT_OUTPUT_STATE_OFF
 - hal_iksmcu_mcpwm.h, 202
- HAL_MCPWM_DEFAULT_OUTPUT_STATE_x
 - hal_iksmcu_mcpwm.h, 202
- HAL_MCPWM_Disable
 - hal_iks32mc09x_mcpwm.c, 388
 - hal_iksmcu_mcpwm.h, 204
- HAL_MCPWM_Enable
 - hal_iks32mc09x_mcpwm.c, 388
 - hal_iksmcu_mcpwm.h, 204
- HAL_MCPWM_FAIL_CMP0_HIGH
 - hal_iksmcu_mcpwm.h, 202
- HAL_MCPWM_FAIL_CMP0_LOW
 - hal_iksmcu_mcpwm.h, 202
- HAL_MCPWM_FAIL_CMP1_HIGH
 - hal_iksmcu_mcpwm.h, 202
- HAL_MCPWM_FAIL_CMP1_LOW
 - hal_iksmcu_mcpwm.h, 202
- HAL_MCPWM_FAIL_IO0_HIGH
 - hal_iksmcu_mcpwm.h, 202
- HAL_MCPWM_FAIL_IO0_LOW
 - hal_iksmcu_mcpwm.h, 202
- HAL_MCPWM_FAIL_IO1_HIGH
 - hal_iksmcu_mcpwm.h, 202
- HAL_MCPWM_FAIL_IO1_LOW
 - hal_iksmcu_mcpwm.h, 202
- HAL_MCPWM_FAIL_x
 - hal_iksmcu_mcpwm.h, 202
- HAL_MCPWM_GetFreq
 - hal_iks32mc09x_mcpwm.c, 389
 - hal_iksmcu_mcpwm.h, 204
- HAL_MCPWM_GetThCnt
 - hal_iks32mc09x_mcpwm.c, 389
 - hal_iksmcu_mcpwm.h, 206
- HAL_MCPWM_IrqFail
 - hal_iks32mc09x_mcpwm.c, 390
 - hal_iksmcu_mcpwm.h, 206
- HAL_MCPWM_IrqFailClear
 - hal_iks32mc09x_mcpwm.c, 390
 - hal_iksmcu_mcpwm.h, 207
- HAL_MCPWM_IrqFailDisable
 - hal_iks32mc09x_mcpwm.c, 390
 - hal_iksmcu_mcpwm.h, 207
- HAL_MCPWM_IrqFailEnable
 - hal_iks32mc09x_mcpwm.c, 390
 - hal_iksmcu_mcpwm.h, 207
- HAL_MCPWM_IRQHandler
 - hal_iks32mc09x.h, 83
- HAL_MCPWM_IRQHandler_00
 - hal_iks32mc09x.h, 83
- HAL_MCPWM_IRQHandler_01
 - hal_iks32mc09x.h, 83
- HAL_MCPWM_IRQHandler_10
 - hal_iks32mc09x.h, 83
- HAL_MCPWM_IRQHandler_11
 - hal_iks32mc09x.h, 83
- HAL_MCPWM_IrqOverflow
 - hal_iks32mc09x_mcpwm.c, 390
 - hal_iksmcu_mcpwm.h, 207
- HAL_MCPWM_IrqOverflowClear
 - hal_iks32mc09x_mcpwm.c, 391
 - hal_iksmcu_mcpwm.h, 207
- HAL_MCPWM_IrqOverflowDisable
 - hal_iks32mc09x_mcpwm.c, 391
 - hal_iksmcu_mcpwm.h, 207
- HAL_MCPWM_IrqOverflowEnable
 - hal_iks32mc09x_mcpwm.c, 391
 - hal_iksmcu_mcpwm.h, 207

HAL_MCPWM_IrqZeroCross
 hal_iks32mc09x_mcpwm.c, 391
 hal_iksmcu_mcpwm.h, 208
 HAL_MCPWM_IrqZeroCrossClear
 hal_iks32mc09x_mcpwm.c, 391
 hal_iksmcu_mcpwm.h, 208
 HAL_MCPWM_IrqZeroCrossDisable
 hal_iks32mc09x_mcpwm.c, 391
 hal_iksmcu_mcpwm.h, 208
 HAL_MCPWM_IrqZeroCrossEnable
 hal_iks32mc09x_mcpwm.c, 391
 hal_iksmcu_mcpwm.h, 208
 HAL_MCPWM_OUTPUT_STATE_HIGH
 hal_iksmcu_mcpwm.h, 203
 HAL_MCPWM_OUTPUT_STATE_HIGH_PWM
 hal_iksmcu_mcpwm.h, 203
 HAL_MCPWM_OUTPUT_STATE_LOW
 hal_iksmcu_mcpwm.h, 203
 HAL_MCPWM_OUTPUT_STATE_LOW_PWM
 hal_iksmcu_mcpwm.h, 203
 HAL_MCPWM_OUTPUT_STATE_OFF
 hal_iksmcu_mcpwm.h, 203
 HAL_MCPWM_OUTPUT_STATE_PWM
 hal_iksmcu_mcpwm.h, 203
 HAL_MCPWM_OUTPUT_STATE_x
 hal_iksmcu_mcpwm.h, 202
 HAL_MCPWM_POLARITY_PH_NH
 hal_iksmcu_mcpwm.h, 203
 HAL_MCPWM_POLARITY_PH_NL
 hal_iksmcu_mcpwm.h, 203
 HAL_MCPWM_POLARITY_PL_NH
 hal_iksmcu_mcpwm.h, 203
 HAL_MCPWM_POLARITY_PL_NL
 hal_iksmcu_mcpwm.h, 203
 HAL_MCPWM_POLARITY_x
 hal_iksmcu_mcpwm.h, 203
 HAL_MCPWM_PORT_0
 hal_iksmcu_mcpwm.h, 203
 HAL_MCPWM_PORT_1
 hal_iksmcu_mcpwm.h, 203
 HAL_MCPWM_PORT_2
 hal_iksmcu_mcpwm.h, 203
 HAL_MCPWM_PORT_3
 hal_iksmcu_mcpwm.h, 203
 HAL_MCPWM_PORT_4
 hal_iksmcu_mcpwm.h, 203
 HAL_MCPWM_PORT_5
 hal_iksmcu_mcpwm.h, 203
 HAL_MCPWM_PORT_x
 hal_iksmcu_mcpwm.h, 203
 HAL_MCPWM_Reset
 hal_iks32mc09x_mcpwm.c, 392
 hal_iksmcu_mcpwm.h, 208
 HAL_MCPWM_SetAdcTrig
 hal_iks32mc09x_mcpwm.c, 392
 hal_iksmcu_mcpwm.h, 209
 HAL_MCPWM_SetAdcTrigQ15
 hal_iks32mc09x_mcpwm.c, 392
 hal_iksmcu_mcpwm.h, 209
 HAL_MCPWM_SetDead
 hal_iks32mc09x_mcpwm.c, 392
 hal_iksmcu_mcpwm.h, 209
 HAL_MCPWM_SetDefaultState
 hal_iks32mc09x_mcpwm.c, 393
 hal_iksmcu_mcpwm.h, 210
 HAL_MCPWM_SetDrivePolarity
 hal_iks32mc09x_mcpwm.c, 394
 hal_iksmcu_mcpwm.h, 210
 HAL_MCPWM_SetFailSignal
 hal_iks32mc09x_mcpwm.c, 394
 hal_iksmcu_mcpwm.h, 211
 HAL_MCPWM_SetFreq
 hal_iks32mc09x_mcpwm.c, 395
 hal_iksmcu_mcpwm.h, 211
 HAL_MCPWM_SetGpioFail
 hal_iks32mc09x_mcpwm.c, 395
 hal_iksmcu_mcpwm.h, 212
 HAL_MCPWM_SetGpioOut
 hal_iks32mc09x_mcpwm.c, 396
 hal_iksmcu_mcpwm.h, 212
 HAL_MCPWM_SetOutputDisable
 hal_iks32mc09x_mcpwm.c, 396
 hal_iksmcu_mcpwm.h, 213
 HAL_MCPWM_SetOutputEnable
 hal_iks32mc09x_mcpwm.c, 397
 hal_iksmcu_mcpwm.h, 213
 HAL_MCPWM_SetOutState
 hal_iks32mc09x_mcpwm.c, 397
 hal_iksmcu_mcpwm.h, 214
 HAL_MCPWM_SetPwm
 hal_iks32mc09x_mcpwm.c, 398
 hal_iksmcu_mcpwm.h, 214
 HAL_MCPWM_SetPwmQ15
 hal_iks32mc09x_mcpwm.c, 398
 hal_iksmcu_mcpwm.h, 215
 HAL_MCPWM_StartCounting
 hal_iks32mc09x_mcpwm.c, 399
 hal_iksmcu_mcpwm.h, 215
 HAL_MCPWM_StopCounting
 hal_iks32mc09x_mcpwm.c, 399
 hal_iksmcu_mcpwm.h, 215
 HAL_MCPWM_TypeDef, 44
 AUEN, 46
 CH_DEF, 46
 CH_FAIL, 46
 CH_MSK, 47
 CNT0, 47
 DTH00, 47
 DTH01, 47
 EIE, 47
 EIF, 47
 EVT0, 48
 FCNT, 48
 FLT, 48
 IE0, 48
 IF0, 48

IO01, [49](#)
 IO23, [49](#)
 PP, [49](#)
 PRT, [49](#)
 RE, [50](#)
 RESERVED, [50](#)
 RESERVED1, [50](#)
 RESERVED2, [50](#)
 RESERVED3, [50](#)
 RESERVED4, [50](#)
 RESERVED5, [51](#)
 RESERVED6, [51](#)
 RESERVED7, [51](#)
 RESERVED8, [51](#)
 SDCFG, [51](#)
 SW_FAIL, [51](#)
 TCLK, [51](#)
 TH, [52](#)
 TH0, [52](#)
 TMR0, [52](#)
 TMR1, [52](#)
 TMR2, [52](#)
 TMR3, [52](#)
 UPDATE, [53](#)
 hal_MCPWM_UpdatePolarityState
 hal_lks32mc09x_mcpwm.c, [399](#)
 HAL_MCPWM_x
 hal_lksmcu_mcpwm.h, [203](#)
 HAL_OPA_0
 hal_lksmcu_opa.h, [220](#)
 HAL_OPA_0B
 hal_lksmcu_opa.h, [220](#)
 HAL_OPA_1
 hal_lksmcu_opa.h, [220](#)
 HAL_OPA_2
 hal_lksmcu_opa.h, [220](#)
 HAL_OPA_3
 hal_lksmcu_opa.h, [220](#)
 HAL_OPA_4
 hal_lksmcu_opa.h, [220](#)
 HAL_OPA_5
 hal_lksmcu_opa.h, [220](#)
 HAL_OPA_Disable
 hal_lks32mc09x_opa.c, [408](#)
 hal_lksmcu_opa.h, [221](#)
 HAL_OPA_Enable
 hal_lks32mc09x_opa.c, [408](#)
 hal_lksmcu_opa.h, [221](#)
 HAL_OPA_GAIN_160_10
 hal_lksmcu_opa.h, [220](#)
 HAL_OPA_GAIN_160_40
 hal_lksmcu_opa.h, [220](#)
 HAL_OPA_GAIN_170_40
 hal_lksmcu_opa.h, [220](#)
 HAL_OPA_GAIN_170_40_4
 hal_lksmcu_opa.h, [220](#)
 HAL_OPA_GAIN_170_40_6
 hal_lksmcu_opa.h, [220](#)
 HAL_OPA_GAIN_180_30
 hal_lksmcu_opa.h, [220](#)
 HAL_OPA_GAIN_180_30_4
 hal_lksmcu_opa.h, [220](#)
 HAL_OPA_GAIN_180_30_6
 hal_lksmcu_opa.h, [220](#)
 HAL_OPA_GAIN_190_20
 hal_lksmcu_opa.h, [220](#)
 HAL_OPA_GAIN_190_20_4
 hal_lksmcu_opa.h, [220](#)
 HAL_OPA_GAIN_190_20_6
 hal_lksmcu_opa.h, [220](#)
 HAL_OPA_GAIN_200_10
 hal_lksmcu_opa.h, [220](#)
 HAL_OPA_GAIN_200_10_4
 hal_lksmcu_opa.h, [220](#)
 HAL_OPA_GAIN_200_10_6
 hal_lksmcu_opa.h, [220](#)
 HAL_OPA_GAIN_20_10
 hal_lksmcu_opa.h, [220](#)
 HAL_OPA_GAIN_320_10
 hal_lksmcu_opa.h, [220](#)
 HAL_OPA_GAIN_320_40
 hal_lksmcu_opa.h, [220](#)
 HAL_OPA_GAIN_320_5
 hal_lksmcu_opa.h, [220](#)
 HAL_OPA_GAIN_40_10
 hal_lksmcu_opa.h, [220](#)
 HAL_OPA_GAIN_640_40
 hal_lksmcu_opa.h, [220](#)
 HAL_OPA_GAIN_660_20
 hal_lksmcu_opa.h, [220](#)
 HAL_OPA_GAIN_80_10
 hal_lksmcu_opa.h, [220](#)
 HAL_OPA_GAIN_HIGH
 hal_lksmcu_opa.h, [220](#)
 HAL_OPA_GAIN_LOW
 hal_lksmcu_opa.h, [220](#)
 HAL_OPA_GAIN_MEDIUM
 hal_lksmcu_opa.h, [220](#)
 HAL_OPA_GAIN_x
 hal_lksmcu_opa.h, [219](#)
 HAL_OPA_OPAX_2_ADCCHN
 hal_lksmcu_opa.h, [219](#)
 HAL_OPA_Out2GpioDisable
 hal_lks32mc09x_opa.c, [408](#)
 hal_lksmcu_opa.h, [221](#)
 HAL_OPA_Out2GpioEnable
 hal_lks32mc09x_opa.c, [408](#)
 hal_lksmcu_opa.h, [221](#)
 HAL_OPA_READ_GAIN
 hal_lksmcu_opa.h, [219](#)
 HAL_OPA_Reset
 hal_lks32mc09x_opa.c, [409](#)
 hal_lksmcu_opa.h, [222](#)
 HAL_OPA_SetGain
 hal_lks32mc09x_opa.c, [409](#)
 hal_lksmcu_opa.h, [222](#)

HAL_OPA_SetGpioIn
 hal_lks32mc09x_opa.c, 409
 hal_lksmcu_opa.h, 222

HAL_OPA_SetGpioOut
 hal_lks32mc09x_opa.c, 410
 hal_lksmcu_opa.h, 223

HAL_OPA_x
 hal_lksmcu_opa.h, 220

HAL_PWRDN_IRQHandler
 hal_lks32mc09x.h, 83

HAL_PWRDN_IRQHandler_00
 hal_lks32mc09x.h, 83

HAL_QEP
 hal_lks32mc09x_reg.c, 420
 hal_lks32mc09x_reg.h, 93

HAL_QEP_0
 hal_lksmcu_qep.h, 226

HAL_QEP_1
 hal_lksmcu_qep.h, 226

HAL_QEP_2
 hal_lksmcu_qep.h, 226

HAL_QEP_3
 hal_lksmcu_qep.h, 226

HAL_QEP_Disable
 hal_lks32mc09x_qep.c, 413
 hal_lksmcu_qep.h, 226

HAL_QEP_Enable
 hal_lks32mc09x_qep.c, 413
 hal_lksmcu_qep.h, 226

HAL_QEP_IRQHandler
 hal_lks32mc09x.h, 83

HAL_QEP_IRQHandler_00
 hal_lks32mc09x.h, 84

HAL_QEP_IRQHandler_10
 hal_lks32mc09x.h, 84

HAL_QEP_IRQHandler_20
 hal_lks32mc09x.h, 84

HAL_QEP_IRQHandler_30
 hal_lks32mc09x.h, 84

HAL_QEP_MODE_DUAL_PULSE
 hal_lksmcu_qep.h, 226

HAL_QEP_MODE_QUADRATURE
 hal_lksmcu_qep.h, 226

HAL_QEP_MODE_SYMBOL_PULSE
 hal_lksmcu_qep.h, 226

HAL_QEP_MODE_x
 hal_lksmcu_qep.h, 226

HAL_QEP_ReadCnt
 hal_lks32mc09x_qep.c, 414
 hal_lksmcu_qep.h, 227

HAL_QEP_Reset
 hal_lks32mc09x_qep.c, 414
 hal_lksmcu_qep.h, 227

HAL_QEP_ResetCnt
 hal_lks32mc09x_qep.c, 415
 hal_lksmcu_qep.h, 228

HAL_QEP_SetGpio
 hal_lks32mc09x_qep.c, 415
 hal_lksmcu_qep.h, 228

HAL_QEP_SetMode
 hal_lks32mc09x_qep.c, 415
 hal_lksmcu_qep.h, 228

HAL_QEP_SetTh
 hal_lks32mc09x_qep.c, 416
 hal_lksmcu_qep.h, 229

HAL_QEP_TypeDef, 53
 CFG, 54
 CNT, 54
 IE, 54
 IF, 54
 TH, 54

HAL_QEP_x
 hal_lksmcu_qep.h, 226

HAL_REG_IF_WRITE
 hal_lksmcu_reg.h, 230

HAL_REG_INVERT
 hal_lksmcu_reg.h, 230

HAL_REG_READ
 hal_lksmcu_reg.h, 231

HAL_REG_READ_BIT
 hal_lksmcu_reg.h, 231

HAL_REG_READ_BIT_POS_MASK
 hal_lksmcu_reg.h, 231

HAL_REG_RESET
 hal_lksmcu_reg.h, 231

HAL_REG_SET
 hal_lksmcu_reg.h, 231

HAL_REG_SYS_LOCK
 hal_lksmcu_reg.h, 231

HAL_REG_SYS_UNLOCK
 hal_lksmcu_reg.h, 232

HAL_REG_WRITE
 hal_lksmcu_reg.h, 232

HAL_REG_WRITE_BIT
 hal_lksmcu_reg.h, 232

HAL_REG_WRITE_BIT_MASK_POS
 hal_lksmcu_reg.h, 232

HAL_RESET_ResetCore
 hal_lks32mc09x_reset.c, 422
 hal_lksmcu_reset.h, 233

HAL_RESET_ResetGlobal
 hal_lks32mc09x_reset.c, 422
 hal_lksmcu_reset.h, 233

HAL_RESET_ResetRegister
 hal_lks32mc09x_reset.c, 422
 hal_lksmcu_reset.h, 233

HAL_SIF_IRQHandler
 hal_lks32mc09x.h, 84

HAL_SIF_IRQHandler_00
 hal_lks32mc09x.h, 84

HAL_SIF_TypeDef, 55
 CFG, 55
 IRQ, 55
 TDTH1, 56
 TOSC, 56
 TSTH1, 56

WDATA, 56
 HAL_SPI
 hal_iks32mc09x_reg.c, 420
 hal_iks32mc09x_reg.h, 94
 HAL_SPI_0
 hal_iksmcu_spi.h, 236
 HAL_SPI_1
 hal_iksmcu_spi.h, 236
 HAL_SPI_Disable
 hal_iks32mc09x_spi.c, 425
 hal_iksmcu_spi.h, 237
 HAL_SPI_Enable
 hal_iks32mc09x_spi.c, 425
 hal_iksmcu_spi.h, 237
 HAL_SPI_IRQHandler
 hal_iks32mc09x.h, 84
 HAL_SPI_IRQHandler_00
 hal_iks32mc09x.h, 84
 HAL_SPI_IRQHandler_10
 hal_iks32mc09x.h, 85
 HAL_SPI_MODE_MASTER_LSB_00
 hal_iksmcu_spi.h, 236
 HAL_SPI_MODE_MASTER_LSB_01
 hal_iksmcu_spi.h, 236
 HAL_SPI_MODE_MASTER_LSB_10
 hal_iksmcu_spi.h, 236
 HAL_SPI_MODE_MASTER_LSB_11
 hal_iksmcu_spi.h, 236
 HAL_SPI_MODE_MASTER_MSB_00
 hal_iksmcu_spi.h, 236
 HAL_SPI_MODE_MASTER_MSB_01
 hal_iksmcu_spi.h, 236
 HAL_SPI_MODE_MASTER_MSB_10
 hal_iksmcu_spi.h, 236
 HAL_SPI_MODE_MASTER_MSB_11
 hal_iksmcu_spi.h, 236
 HAL_SPI_MODE_SLAVE_LSB_00
 hal_iksmcu_spi.h, 236
 HAL_SPI_MODE_SLAVE_LSB_01
 hal_iksmcu_spi.h, 236
 HAL_SPI_MODE_SLAVE_LSB_10
 hal_iksmcu_spi.h, 236
 HAL_SPI_MODE_SLAVE_LSB_11
 hal_iksmcu_spi.h, 236
 HAL_SPI_MODE_SLAVE_MSB_00
 hal_iksmcu_spi.h, 236
 HAL_SPI_MODE_SLAVE_MSB_01
 hal_iksmcu_spi.h, 236
 HAL_SPI_MODE_SLAVE_MSB_10
 hal_iksmcu_spi.h, 236
 HAL_SPI_MODE_SLAVE_MSB_11
 hal_iksmcu_spi.h, 236
 HAL_SPI_MODE_x
 hal_iksmcu_spi.h, 236
 HAL_SPI_Reset
 hal_iks32mc09x_spi.c, 425
 hal_iksmcu_spi.h, 237
 HAL_SPI_RxData
 hal_iks32mc09x_spi.c, 425
 hal_iksmcu_spi.h, 237
 HAL_SPI_RxDataWidth16
 hal_iks32mc09x_spi.c, 427
 hal_iksmcu_spi.h, 238
 HAL_SPI_RxTxData
 hal_iks32mc09x_spi.c, 427
 hal_iksmcu_spi.h, 238
 HAL_SPI_RxTxDataWidth16
 hal_iks32mc09x_spi.c, 428
 hal_iksmcu_spi.h, 239
 HAL_SPI_SetBaud
 hal_iks32mc09x_spi.c, 428
 hal_iksmcu_spi.h, 239
 HAL_SPI_SetGpioCik
 hal_iks32mc09x_spi.c, 429
 hal_iksmcu_spi.h, 240
 HAL_SPI_SetGpioCs
 hal_iks32mc09x_spi.c, 429
 hal_iksmcu_spi.h, 240
 HAL_SPI_SetGpioDi
 hal_iks32mc09x_spi.c, 430
 hal_iksmcu_spi.h, 241
 HAL_SPI_SetGpioDo
 hal_iks32mc09x_spi.c, 431
 hal_iksmcu_spi.h, 242
 HAL_SPI_SetGpioMiso
 hal_iks32mc09x_spi.c, 431
 hal_iksmcu_spi.h, 242
 HAL_SPI_SetGpioMosi
 hal_iks32mc09x_spi.c, 432
 hal_iksmcu_spi.h, 243
 HAL_SPI_SetMode
 hal_iks32mc09x_spi.c, 433
 hal_iksmcu_spi.h, 244
 HAL_SPI_TxData
 hal_iks32mc09x_spi.c, 433
 hal_iksmcu_spi.h, 244
 HAL_SPI_TxDataWidth16
 hal_iks32mc09x_spi.c, 433
 hal_iksmcu_spi.h, 244
 HAL_SPI_TypeDef, 56
 CFG, 57
 DIV, 57
 IE, 57
 RX_DATA, 57
 SIZE, 58
 TX_DATA, 58
 HAL_SPI_x
 hal_iksmcu_spi.h, 236
 HAL_SW_IRQHandler
 hal_iks32mc09x.h, 85
 HAL_SW_IRQHandler_00
 hal_iks32mc09x.h, 85
 HAL_SYS
 hal_iks32mc09x_reg.c, 420
 hal_iks32mc09x_reg.h, 94
 HAL_SYS_TypeDef, 58

AFE_ADC, 60
 AFE_DAC0, 60
 AFE_DAC0_AMC, 61
 AFE_DAC0_DC, 61
 AFE_DAC1, 61
 AFE_DAC1_AMC, 61
 AFE_DAC1_DC, 61
 AFE_DAC_CTRL, 61
 AFE_DBG, 62
 AFE_INFO, 62
 AFE_REG0, 62
 AFE_REG1, 62
 AFE_REG2, 62
 AFE_REG3, 63
 AFE_REG4, 63
 AFE_REG5, 63
 AFE_REG7, 63
 CLK_CFG, 63
 CLK_DIV0, 64
 CLK_DIV1, 64
 CLK_DIV2, 64
 CLK_FEN, 64
 DBG_CFG, 64
 FLSE, 64
 FLSP, 65
 IF, 65
 IO_CFG, 65
 NMI_TH, 65
 OPA_SEL, 65
 PROTECT, 65
 RAM_LOCK, 66
 RESERVED, 66
 RESERVED1, 66
 RESERVED2, 66
 RESERVED3, 66
 RESERVED41, 66
 RESERVED5, 67
 RESERVED6, 67
 RESERVED7, 67
 SFT_RST, 67
 HAL_TestRegAddr
 hal_lks32mc09x_test.c, 438
 HAL_TIMER
 hal_lks32mc09x_reg.c, 420
 hal_lks32mc09x_reg.h, 94
 HAL_TIMER_0
 hal_lksmcu_timer.h, 248
 HAL_TIMER_1
 hal_lksmcu_timer.h, 248
 HAL_TIMER_2
 hal_lksmcu_timer.h, 248
 HAL_TIMER_3
 hal_lksmcu_timer.h, 248
 HAL_TIMER_4
 hal_lksmcu_timer.h, 248
 HAL_TIMER_Disable
 hal_lks32mc09x_timer.c, 440
 hal_lksmcu_timer.h, 248
 HAL_TIMER_Enable
 hal_lks32mc09x_timer.c, 440
 hal_lksmcu_timer.h, 248
 HAL_TIMER_GetFreq
 hal_lks32mc09x_timer.c, 440
 hal_lksmcu_timer.h, 248
 HAL_TIMER_GetThCnt
 hal_lks32mc09x_timer.c, 440
 hal_lksmcu_timer.h, 249
 HAL_TIMER_IrqCompare0
 hal_lks32mc09x_timer.c, 441
 hal_lksmcu_timer.h, 249
 HAL_TIMER_IrqCompare0Clear
 hal_lks32mc09x_timer.c, 441
 hal_lksmcu_timer.h, 249
 HAL_TIMER_IrqCompare0Disable
 hal_lks32mc09x_timer.c, 441
 hal_lksmcu_timer.h, 249
 HAL_TIMER_IrqCompare0Enable
 hal_lks32mc09x_timer.c, 441
 hal_lksmcu_timer.h, 250
 HAL_TIMER_IrqCompare1
 hal_lks32mc09x_timer.c, 441
 hal_lksmcu_timer.h, 250
 HAL_TIMER_IrqCompare1Clear
 hal_lks32mc09x_timer.c, 442
 hal_lksmcu_timer.h, 250
 HAL_TIMER_IrqCompare1Disable
 hal_lks32mc09x_timer.c, 442
 hal_lksmcu_timer.h, 250
 HAL_TIMER_IrqCompare1Enable
 hal_lks32mc09x_timer.c, 442
 hal_lksmcu_timer.h, 250
 HAL_TIMER_IRQHandler
 hal_lks32mc09x.h, 85
 HAL_TIMER_IRQHandler_00
 hal_lks32mc09x.h, 85
 HAL_TIMER_IRQHandler_10
 hal_lks32mc09x.h, 85
 HAL_TIMER_IRQHandler_20
 hal_lks32mc09x.h, 85
 HAL_TIMER_IRQHandler_30
 hal_lks32mc09x.h, 86
 HAL_TIMER_IRQHandler_40
 hal_lks32mc09x.h, 86
 HAL_TIMER_IrqOverflow
 hal_lks32mc09x_timer.c, 442
 hal_lksmcu_timer.h, 250
 HAL_TIMER_IrqOverflowClear
 hal_lks32mc09x_timer.c, 442
 hal_lksmcu_timer.h, 250
 HAL_TIMER_IrqOverflowDisable
 hal_lks32mc09x_timer.c, 442
 hal_lksmcu_timer.h, 251
 HAL_TIMER_IrqOverflowEnable
 hal_lks32mc09x_timer.c, 442
 hal_lksmcu_timer.h, 251
 HAL_TIMER_MODE_COUNT

hal_lksmcu_timer.h, 247
 HAL_TIMER_MODE_PWMIN
 hal_lksmcu_timer.h, 247
 HAL_TIMER_MODE_PWMOUT
 hal_lksmcu_timer.h, 247
 HAL_TIMER_MODE_x
 hal_lksmcu_timer.h, 247
 HAL_TIMER_Reset
 hal_lks32mc09x_timer.c, 443
 hal_lksmcu_timer.h, 251
 HAL_TIMER_SetFreq
 hal_lks32mc09x_timer.c, 443
 hal_lksmcu_timer.h, 251
 HAL_TIMER_SetGpioIn
 hal_lks32mc09x_timer.c, 443
 hal_lksmcu_timer.h, 252
 HAL_TIMER_SetGpioOut
 hal_lks32mc09x_timer.c, 444
 hal_lksmcu_timer.h, 252
 HAL_TIMER_SetPeriodCount
 hal_lks32mc09x_timer.c, 445
 hal_lksmcu_timer.h, 253
 HAL_TIMER_SetPeriodmS
 hal_lks32mc09x_timer.c, 445
 hal_lksmcu_timer.h, 254
 HAL_TIMER_SetPerioduS
 hal_lks32mc09x_timer.c, 446
 hal_lksmcu_timer.h, 254
 HAL_TIMER_SetPwm
 hal_lks32mc09x_timer.c, 446
 hal_lksmcu_timer.h, 255
 HAL_TIMER_SetPwmQ15
 hal_lks32mc09x_timer.c, 446
 hal_lksmcu_timer.h, 255
 HAL_TIMER_SetWorkMode
 hal_lks32mc09x_timer.c, 447
 hal_lksmcu_timer.h, 255
 HAL_TIMER_StartCounting
 hal_lks32mc09x_timer.c, 447
 hal_lksmcu_timer.h, 256
 HAL_TIMER_StopCounting
 hal_lks32mc09x_timer.c, 447
 hal_lksmcu_timer.h, 256
 HAL_TIMER_TypeDef, 67
 CFG0, 69
 CFG1, 69
 CFG2, 69
 CHN0, 69
 CHN1, 69
 CNT, 70
 EVT, 70
 FLT, 70
 IE, 70
 IF, 70
 IO, 70
 RESERVED, 71
 TH, 71
 HAL_TIMER_x
 hal_lksmcu_timer.h, 247
 HAL_TMP_0
 hal_lksmcu_tmp.h, 258
 HAL_TMP_1
 hal_lksmcu_tmp.h, 258
 HAL_TMP_Adc2Temperature
 hal_lks32mc09x_tmp.c, 453
 hal_lksmcu_tmp.h, 258
 HAL_TMP_Adc2Temperature01
 hal_lks32mc09x_tmp.c, 453
 hal_lksmcu_tmp.h, 259
 HAL_TMP_Disable
 hal_lks32mc09x_tmp.c, 454
 hal_lksmcu_tmp.h, 260
 HAL_TMP_Enable
 hal_lks32mc09x_tmp.c, 455
 hal_lksmcu_tmp.h, 260
 HAL_TMP_GetTemperature
 hal_lks32mc09x_tmp.c, 455
 hal_lksmcu_tmp.h, 261
 HAL_TMP_GetTemperature01
 hal_lks32mc09x_tmp.c, 456
 hal_lksmcu_tmp.h, 261
 HAL_TMP_Reset
 hal_lks32mc09x_tmp.c, 456
 hal_lksmcu_tmp.h, 262
 HAL_TMP_x
 hal_lksmcu_tmp.h, 258
 HAL_TRIM_Read
 hal_lks32mc09x_trim.c, 459
 hal_lksmcu_trim.h, 264
 HAL_TRIM_ReadId
 hal_lks32mc09x_trim.c, 460
 hal_lksmcu_trim.h, 264
 HAL_TRIM_ReadIdCrc32
 hal_lks32mc09x_trim.c, 461
 hal_lksmcu_trim.h, 265
 HAL_TRIM_ReadMcu09
 hal_lks32mc09x_trim.c, 461
 HAL_UART
 hal_lks32mc09x_reg.c, 420
 hal_lks32mc09x_reg.h, 94
 HAL_UART_0
 hal_lksmcu_uart.h, 268
 HAL_UART_1
 hal_lksmcu_uart.h, 268
 HAL_UART_2
 hal_lksmcu_uart.h, 268
 HAL_UART_Disable
 hal_lks32mc09x_uart.c, 464
 hal_lksmcu_uart.h, 268
 HAL_UART_Enable
 hal_lks32mc09x_uart.c, 464
 hal_lksmcu_uart.h, 268
 HAL_UART_IrqCheckErr
 hal_lks32mc09x_uart.c, 465
 hal_lksmcu_uart.h, 268
 HAL_UART_IrqCheckErrClear

hal_lks32mc09x_uart.c, 465
 hal_lksmcu_uart.h, 268
 HAL_UART_IrqCheckErrDisable
 hal_lks32mc09x_uart.c, 465
 hal_lksmcu_uart.h, 269
 HAL_UART_IrqCheckErrEnable
 hal_lks32mc09x_uart.c, 465
 hal_lksmcu_uart.h, 269
 HAL_UART_IRQHandler
 hal_lks32mc09x.h, 86
 HAL_UART_IRQHandler_00
 hal_lks32mc09x.h, 86
 HAL_UART_IRQHandler_10
 hal_lks32mc09x.h, 86
 HAL_UART_IRQHandler_20
 hal_lks32mc09x.h, 86
 HAL_UART_IrqReceiveOver
 hal_lks32mc09x_uart.c, 465
 hal_lksmcu_uart.h, 269
 HAL_UART_IrqReceiveOverClear
 hal_lks32mc09x_uart.c, 465
 hal_lksmcu_uart.h, 269
 HAL_UART_IrqReceiveOverDisable
 hal_lks32mc09x_uart.c, 465
 hal_lksmcu_uart.h, 269
 HAL_UART_IrqReceiveOverEnable
 hal_lks32mc09x_uart.c, 466
 hal_lksmcu_uart.h, 269
 HAL_UART_IrqSendOver
 hal_lks32mc09x_uart.c, 466
 hal_lksmcu_uart.h, 269
 HAL_UART_IrqSendOverClear
 hal_lks32mc09x_uart.c, 466
 hal_lksmcu_uart.h, 270
 HAL_UART_IrqSendOverDisable
 hal_lks32mc09x_uart.c, 466
 hal_lksmcu_uart.h, 270
 HAL_UART_IrqSendOverEnable
 hal_lks32mc09x_uart.c, 466
 hal_lksmcu_uart.h, 270
 HAL_UART_IrqStopErr
 hal_lks32mc09x_uart.c, 466
 hal_lksmcu_uart.h, 270
 HAL_UART_IrqStopErrClear
 hal_lks32mc09x_uart.c, 466
 hal_lksmcu_uart.h, 270
 HAL_UART_IrqStopErrDisable
 hal_lks32mc09x_uart.c, 467
 hal_lksmcu_uart.h, 270
 HAL_UART_IrqStopErrEnable
 hal_lks32mc09x_uart.c, 467
 hal_lksmcu_uart.h, 270
 HAL_UART_MODE_LSB_STOP1_EVEN
 hal_lksmcu_uart.h, 267
 HAL_UART_MODE_LSB_STOP1_NONE
 hal_lksmcu_uart.h, 267
 HAL_UART_MODE_LSB_STOP1_ODD
 hal_lksmcu_uart.h, 267
 HAL_UART_MODE_LSB_STOP2_EVEN
 hal_lksmcu_uart.h, 267
 HAL_UART_MODE_LSB_STOP2_NONE
 hal_lksmcu_uart.h, 267
 HAL_UART_MODE_LSB_STOP2_ODD
 hal_lksmcu_uart.h, 267
 HAL_UART_MODE_MSB_STOP1_EVEN
 hal_lksmcu_uart.h, 268
 HAL_UART_MODE_MSB_STOP1_NONE
 hal_lksmcu_uart.h, 268
 HAL_UART_MODE_MSB_STOP1_ODD
 hal_lksmcu_uart.h, 268
 HAL_UART_MODE_MSB_STOP2_EVEN
 hal_lksmcu_uart.h, 268
 HAL_UART_MODE_MSB_STOP2_NONE
 hal_lksmcu_uart.h, 268
 HAL_UART_MODE_MSB_STOP2_ODD
 hal_lksmcu_uart.h, 268
 HAL_UART_MODE_x
 hal_lksmcu_uart.h, 267
 HAL_UART_ReadBuffer
 hal_lks32mc09x_uart.c, 467
 hal_lksmcu_uart.h, 271
 HAL_UART_Reset
 hal_lks32mc09x_uart.c, 467
 hal_lksmcu_uart.h, 271
 HAL_UART_SetBaud
 hal_lks32mc09x_uart.c, 467
 hal_lksmcu_uart.h, 271
 HAL_UART_SetGpioRx
 hal_lks32mc09x_uart.c, 467
 hal_lksmcu_uart.h, 271
 HAL_UART_SetGpioTx
 hal_lks32mc09x_uart.c, 468
 hal_lksmcu_uart.h, 272
 HAL_UART_SetMode
 hal_lks32mc09x_uart.c, 468
 hal_lksmcu_uart.h, 272
 HAL_UART_TxData
 hal_lks32mc09x_uart.c, 469
 hal_lksmcu_uart.h, 273
 HAL_UART_TypeDef, 71
 ADR, 72
 BUFF, 72
 CTRL, 72
 DIVH, 72
 DIVL, 72
 IE, 72
 IF, 73
 IOC, 73
 RE, 73
 STT, 73
 HAL_UART_WriteBuffer
 hal_lks32mc09x_uart.c, 469
 hal_lksmcu_uart.h, 273
 HAL_UART_x
 hal_lksmcu_uart.h, 268
 HAL_WAKE_DeepSleepToTimeWakeUp

- hal_lks32mc09x_wake.c, 475
- hal_lksmcu_wake.h, 276
- HAL_WAKE_DeepSleepToWake
 - hal_lks32mc09x_wake.c, 476
 - hal_lksmcu_wake.h, 276
- HAL_WAKE_DeepSleepTimeWakeUp
 - hal_lks32mc09x_wake.c, 476
 - hal_lksmcu_wake.h, 277
- HAL_WAKE_IRQHandler
 - hal_lks32mc09x.h, 86
- HAL_WAKE_IRQHandler_00
 - hal_lks32mc09x.h, 86
- HAL_WAKE_UP_IO_0_High
 - hal_lksmcu_wake.h, 276
- HAL_WAKE_UP_IO_0_Low
 - hal_lksmcu_wake.h, 276
- HAL_WAKE_UP_IO_1_High
 - hal_lksmcu_wake.h, 276
- HAL_WAKE_UP_IO_1_Low
 - hal_lksmcu_wake.h, 276
- HAL_WAKE_UP_IO_2_High
 - hal_lksmcu_wake.h, 276
- HAL_WAKE_UP_IO_2_Low
 - hal_lksmcu_wake.h, 276
- HAL_WAKE_UP_IO_3_High
 - hal_lksmcu_wake.h, 276
- HAL_WAKE_UP_IO_3_Low
 - hal_lksmcu_wake.h, 276
- HAL_WAKE_UP_IO_4_High
 - hal_lksmcu_wake.h, 276
- HAL_WAKE_UP_IO_4_Low
 - hal_lksmcu_wake.h, 276
- HAL_WAKE_UP_IO_5_High
 - hal_lksmcu_wake.h, 276
- HAL_WAKE_UP_IO_5_Low
 - hal_lksmcu_wake.h, 276
- HAL_WAKE_UP_IO_6_High
 - hal_lksmcu_wake.h, 276
- HAL_WAKE_UP_IO_6_Low
 - hal_lksmcu_wake.h, 276
- HAL_WAKE_UP_IO_7_High
 - hal_lksmcu_wake.h, 276
- HAL_WAKE_UP_IO_7_Low
 - hal_lksmcu_wake.h, 276
- HAL_WAKE_UP_IO_x
 - hal_lksmcu_wake.h, 275
- HAL_WWDG_IRQHandler_00
 - hal_lks32mc09x.h, 87
- HTH
 - HAL_ADC_TypeDef, 11
- IE
 - CMP_TypeDef, 6
 - HAL_ADC_TypeDef, 12
 - HAL_CLU_TypeDef, 17
 - HAL_CMP_TypeDef, 19
 - HAL_DMAMODULE_TypeDef, 23
 - HAL_EXTI_TypeDef, 28
 - HAL_QEP_TypeDef, 54
 - HAL_SPI_TypeDef, 57
 - HAL_TIMER_TypeDef, 70
 - HAL_UART_TypeDef, 72
- IE0
 - HAL_MCPWM_TypeDef, 48
- IF
 - CMP_TypeDef, 6
 - HAL_ADC_TypeDef, 12
 - HAL_CLU_TypeDef, 17
 - HAL_CMP_TypeDef, 19
 - HAL_DMAMODULE_TypeDef, 23
 - HAL_EXTI_TypeDef, 28
 - HAL_QEP_TypeDef, 54
 - HAL_SYS_TypeDef, 65
 - HAL_TIMER_TypeDef, 70
 - HAL_UART_TypeDef, 73
- IF0
 - HAL_MCPWM_TypeDef, 48
- include/hal_lks32mc09x_reg.h, 89, 94
- include/hal_lksmcu_adc.h, 98, 112
- include/hal_lksmcu_can.h, 115
- include/hal_lksmcu_clk.h, 115, 119
- include/hal_lksmcu_cmp.h, 120, 127
- include/hal_lksmcu_crc.h, 128, 131
- include/hal_lksmcu_dac.h, 132, 136
- include/hal_lksmcu_dma.h, 136, 140
- include/hal_lksmcu_dsp.h, 140, 144
- include/hal_lksmcu_flash.h, 144, 153
- include/hal_lksmcu_gpio.h, 153, 166
- include/hal_lksmcu_hall.h, 168, 175
- include/hal_lksmcu_i2c.h, 176, 182
- include/hal_lksmcu_init.h, 182, 183
- include/hal_lksmcu_irq.h, 183, 187
- include/hal_lksmcu_irqfun.h, 188, 195
- include/hal_lksmcu_iwdg.h, 196, 199
- include/hal_lksmcu_mcpwm.h, 200, 216
- include/hal_lksmcu_opa.h, 217, 223
- include/hal_lksmcu_qep.h, 224, 229
- include/hal_lksmcu_reg.h, 230, 232
- include/hal_lksmcu_reset.h, 233, 234
- include/hal_lksmcu_spi.h, 234, 245
- include/hal_lksmcu_timer.h, 246, 256
- include/hal_lksmcu_tmp.h, 257, 263
- include/hal_lksmcu_trim.h, 263, 266
- include/hal_lksmcu_uart.h, 266, 274
- include/hal_lksmcu_wake.h, 275, 277
- INFO
 - HAL_HALL_TypeDef, 39
- INIT
 - HAL_CRC_TypeDef, 21
- initValue
 - CRC_Model.Params, 7
- IO
 - HAL_TIMER_TypeDef, 70
- IO01
 - HAL_MCPWM_TypeDef, 49
- IO23
 - HAL_MCPWM_TypeDef, 49

IO_CFG
 HAL_SYS_TypeDef, 65
 IO_WAKE_EN
 HAL_AON_TypeDef, 15
 IO_WAKE_POL
 HAL_AON_TypeDef, 15
 IOC
 HAL_UART_TypeDef, 73
 IRQ
 HAL_SIF_TypeDef, 55

 LIB_ADDR
 HAL_FLASH_TypeDef, 31
 LKS32MC09X
 hal_lks32mc09x_reg.h, 92
 LTH
 HAL_ADC_TypeDef, 12

 mcpwm_polarity_state
 hal_lks32mc09x_mcpwm.c, 401
 MSCR
 HAL_I2C_TypeDef, 41
 MX
 HAL_CLU_TypeDef, 17

 NADDR
 HAL_FLASH_TypeDef, 31
 NCFG
 HAL_FLASH_TypeDef, 31
 NERASE
 HAL_FLASH_TypeDef, 31
 NKEY
 HAL_FLASH_TypeDef, 32
 NMI_TH
 HAL_SYS_TypeDef, 65
 NRDATA
 HAL_FLASH_TypeDef, 32
 NWDATA
 HAL_FLASH_TypeDef, 32

 OPA_SEL
 HAL_SYS_TypeDef, 65
 OUT
 HAL_CLU_TypeDef, 17

 PDI
 HAL_GPIO_TypeDef, 36
 PDO
 HAL_GPIO_TypeDef, 36
 PFLT
 HAL_GPIO_TypeDef, 36
 PIE
 HAL_GPIO_TypeDef, 37
 PODE
 HAL_GPIO_TypeDef, 37
 POE
 HAL_GPIO_TypeDef, 37
 POL
 HAL_CRC_TypeDef, 21

 polynomial
 CRC_Model.Params, 7
 PP
 HAL_MCPWM_TypeDef, 49
 PROTECT
 HAL_FLASH_TypeDef, 32
 HAL_SYS_TypeDef, 65
 PRT
 HAL_MCPWM_TypeDef, 49
 PSW
 HAL_IWDG_TypeDef, 43
 PUE
 HAL_GPIO_TypeDef, 37
 PWR_CFG
 HAL_AON_TypeDef, 15

 QEP0
 hal_lks32mc09x_reg.h, 92
 QUO
 HAL_DSP_TypeDef, 25
 QUO1
 HAL_DSP_TypeDef, 25

 RAD
 HAL_DSP_TypeDef, 25
 RAM_LOCK
 HAL_SYS_TypeDef, 66
 RDATA
 HAL_FLASH_TypeDef, 32
 RE
 HAL_MCPWM_TypeDef, 50
 HAL_UART_TypeDef, 73
 readme.md, 278
 READY
 HAL_FLASH_TypeDef, 32
 reflectIn
 CRC_Model.Params, 7
 reflectOut
 CRC_Model.Params, 7
 REM
 HAL_DSP_TypeDef, 25
 REN
 HAL_DMA_TypeDef, 22
 RESERVED
 HAL_ADC_TypeDef, 12
 HAL_DSP_TypeDef, 26
 HAL_FLASH_TypeDef, 33
 HAL_GPIO_TypeDef, 38
 HAL_MCPWM_TypeDef, 50
 HAL_SYS_TypeDef, 66
 HAL_TIMER_TypeDef, 71
 RESERVED1
 HAL_ADC_TypeDef, 12
 HAL_DSP_TypeDef, 26
 HAL_MCPWM_TypeDef, 50
 HAL_SYS_TypeDef, 66
 RESERVED2
 HAL_ADC_TypeDef, 12
 HAL_DSP_TypeDef, 26

HAL_MCPWM_TypeDef, 50
 HAL_SYS_TypeDef, 66
 RESERVED3
 HAL_ADC_TypeDef, 13
 HAL_MCPWM_TypeDef, 50
 HAL_SYS_TypeDef, 66
 RESERVED4
 HAL_MCPWM_TypeDef, 50
 RESERVED41
 HAL_SYS_TypeDef, 66
 RESERVED5
 HAL_MCPWM_TypeDef, 51
 HAL_SYS_TypeDef, 67
 RESERVED6
 HAL_MCPWM_TypeDef, 51
 HAL_SYS_TypeDef, 67
 RESERVED7
 HAL_MCPWM_TypeDef, 51
 HAL_SYS_TypeDef, 67
 RESERVED8
 HAL_MCPWM_TypeDef, 51
 RTH
 HAL_IWDG_TypeDef, 43
 RX_DATA
 HAL_SPI_TypeDef, 57

 SADR
 HAL_DMA_TypeDef, 22
 SCR
 HAL_I2C_TypeDef, 41
 SDCFG
 HAL_MCPWM_TypeDef, 51
 SFT_RST
 HAL_SYS_TypeDef, 67
 SIF0
 hal_iks32mc09x_reg.h, 92
 SIZE
 HAL_FLASH_TypeDef, 33
 HAL_SPI_TypeDef, 58
 source/hal_iks32mc09x_adc.c, 278, 290
 source/hal_iks32mc09x_clk.c, 294, 298
 source/hal_iks32mc09x_cmp.c, 299, 304
 source/hal_iks32mc09x_crc.c, 306, 311
 source/hal_iks32mc09x_dac.c, 312, 316
 source/hal_iks32mc09x_dma.c, 318, 321
 source/hal_iks32mc09x_dsp.c, 323, 326
 source/hal_iks32mc09x_flash.c, 327, 337
 source/hal_iks32mc09x_gpio.c, 341, 348
 source/hal_iks32mc09x_hall.c, 351, 357
 source/hal_iks32mc09x_i2c.c, 359, 365
 source/hal_iks32mc09x_init.c, 367, 368
 source/hal_iks32mc09x_irq.c, 369, 371
 source/hal_iks32mc09x_irqfun.c, 372, 381
 source/hal_iks32mc09x_iwdg.c, 382, 386
 source/hal_iks32mc09x_mcpwm.c, 386, 402
 source/hal_iks32mc09x_opa.c, 406, 410
 source/hal_iks32mc09x_qep.c, 412, 416
 source/hal_iks32mc09x_reg.c, 417, 421
 source/hal_iks32mc09x_reset.c, 421, 423
 source/hal_iks32mc09x_spi.c, 423, 434
 source/hal_iks32mc09x_test.c, 437, 438
 source/hal_iks32mc09x_timer.c, 438, 448
 source/hal_iks32mc09x_tmp.c, 452, 457
 source/hal_iks32mc09x_trim.c, 458, 462
 source/hal_iks32mc09x_uart.c, 463, 470
 source/hal_iks32mc09x_wake.c, 474, 477
 SQRT
 HAL_DSP_TypeDef, 26
 STT
 HAL_UART_TypeDef, 73
 SW_FAIL
 HAL_MCPWM_TypeDef, 51
 SWAP
 HAL_EXTI_TypeDef, 28
 SWT
 HAL_ADC_TypeDef, 13

 TCLK
 CMP_TypeDef, 6
 HAL_CMP_TypeDef, 19
 HAL_MCPWM_TypeDef, 51
 TDTH1
 HAL_SIF_TypeDef, 56
 TH
 HAL_HALL_TypeDef, 39
 HAL_MCPWM_TypeDef, 52
 HAL_QEP_TypeDef, 54
 HAL_TIMER_TypeDef, 71
 TH0
 HAL_MCPWM_TypeDef, 52
 tmp_coef_a
 hal_iks32mc09x_tmp.c, 457
 tmp_coef_b
 hal_iks32mc09x_tmp.c, 457
 TMR0
 HAL_MCPWM_TypeDef, 52
 TMR1
 HAL_MCPWM_TypeDef, 52
 TMR2
 HAL_MCPWM_TypeDef, 52
 TMR3
 HAL_MCPWM_TypeDef, 52
 TOSC
 HAL_SIF_TypeDef, 56
 TRIG
 HAL_ADC_TypeDef, 13
 TSTH1
 HAL_SIF_TypeDef, 56
 TX_DATA
 HAL_SPI_TypeDef, 58

 UPDATE
 HAL_MCPWM_TypeDef, 53

 WDATA
 HAL_FLASH_TypeDef, 33
 HAL_SIF_TypeDef, 56
 WIDTH

HAL_HALL_TypeDef, [39](#)
width
CRC_Model_Params, [7](#)
WTH
HAL_IWDG_TypeDef, [43](#)
xorOut
CRC_Model_Params, [7](#)