

库函数使用说明

0.1 结构体	1
1 文件索引	3
1.1 文件列表	3
2 结构体说明	5
2.1 ADC_InitTypeDef结构体 参考	5
2.1.1 详细描述	6
2.1.2 结构体成员变量说明	6
2.1.2.1 CSMP	6
2.1.2.2 DATA_ALIGN	6
2.1.2.3 GAIN	6
2.1.2.4 GEN	6
2.1.2.5 HTH	7
2.1.2.6 IDLE_PRI	7
2.1.2.7 IE	7
2.1.2.8 IS1	7
2.1.2.9 LTH	7
2.1.2.10 NSMP	7
2.1.2.11 OVSR	8
2.1.2.12 RE	8
2.1.2.13 S1	8
2.1.2.14 S2	8
2.1.2.15 TCNT	8
2.1.2.16 TRIG	8
2.1.2.17 TROVS	9
2.2 ADC_TypeDef结构体 参考	9
2.2.1 详细描述	11
2.2.2 结构体成员变量说明	11
2.2.2.1 AMC0	11
2.2.2.2 AMC1	11
2.2.2.3 CFG	11
2.2.2.4 CHN0	11
2.2.2.5 CHN1	11
2.2.2.6 CHN2	12
2.2.2.7 CHNT	12
2.2.2.8 DAT0	12
2.2.2.9 DAT1	12
2.2.2.10 DAT10	12
2.2.2.11 DAT11	12
2.2.2.12 DAT2	13
2.2.2.13 DAT3	13
2.2.2.14 DAT4	13

2.2.2.15 DAT5	13
2.2.2.16 DAT6	13
2.2.2.17 DAT7	13
2.2.2.18 DAT8	14
2.2.2.19 DAT9	14
2.2.2.20 DC0	14
2.2.2.21 DC1	14
2.2.2.22 GAIN	14
2.2.2.23 GEN	14
2.2.2.24 HTH	15
2.2.2.25 IE	15
2.2.2.26 IF	15
2.2.2.27 LTH	15
2.2.2.28 RESERVED	15
2.2.2.29 RESERVED1	15
2.2.2.30 RESERVED2	16
2.2.2.31 RESERVED3	16
2.2.2.32 SWT	16
2.2.2.33 TRIG	16
2.3 AON_TypeDef结构体 参考	16
2.3.1 详细描述	17
2.3.2 结构体成员变量说明	17
2.3.2.1 BKP_REG0	17
2.3.2.2 BKP_REG1	17
2.3.2.3 EVT_RCD	18
2.3.2.4 IO_WAKE_EN	18
2.3.2.5 IO_WAKE_POL	18
2.3.2.6 PWR_CFG	18
2.4 CLU_InitTypeDef结构体 参考	18
2.4.1 详细描述	19
2.4.2 结构体成员变量说明	19
2.4.2.1 CLKINV	19
2.4.2.2 CLKSEL	19
2.4.2.3 EN	20
2.4.2.4 FIE	20
2.4.2.5 FN	20
2.4.2.6 MXA	20
2.4.2.7 MXB	20
2.4.2.8 OEN	20
2.4.2.9 OUTSEL	21
2.4.2.10 RIE	21
2.5 CLU_TypeDef结构体 参考	21

2.5.1 详细描述	22
2.5.2 结构体成员变量说明	22
2.5.2.1 CF	22
2.5.2.2 EN	22
2.5.2.3 FN	22
2.5.2.4 IE	22
2.5.2.5 IF	23
2.5.2.6 MX	23
2.5.2.7 OUT	23
2.6 CMP_InitTypeDef结构体 参考	23
2.6.1 详细描述	24
2.6.2 结构体成员变量说明	24
2.6.2.1 CLK_COM_DIV	24
2.6.2.2 CMP0_CFG	24
2.6.2.3 CMP1_CFG	24
2.6.2.4 HYS	24
2.7 CMP_SingleTypeDef结构体 参考	25
2.7.1 详细描述	25
2.7.2 结构体成员变量说明	26
2.7.2.1 CHN0P_PWM_EN	26
2.7.2.2 CHN1P_PWM_EN	26
2.7.2.3 CHN2P_PWM_EN	26
2.7.2.4 CHN3P_PWM_EN	26
2.7.2.5 EN	26
2.7.2.6 FIL_CLK_DIV16	26
2.7.2.7 FIL_CLK_DIV2	27
2.7.2.8 IE	27
2.7.2.9 IRQ_TRIG	27
2.7.2.10 POL	27
2.7.2.11 RE	27
2.7.2.12 SELN	27
2.7.2.13 SELP	28
2.7.2.14 W_PWM_POL	28
2.8 CMP_TypeDef结构体 参考	28
2.8.1 详细描述	29
2.8.2 结构体成员变量说明	29
2.8.2.1 BLCWIN	29
2.8.2.2 CFG	29
2.8.2.3 DATA	29
2.8.2.4 IE	29
2.8.2.5 IF	30
2.8.2.6 TCLK	30

2.9 CRC_InitTypeDef结构体 参考	30
2.9.1 详细描述	30
2.9.2 结构体成员变量说明	31
2.9.2.1 INIT	31
2.9.2.2 null	31
2.9.2.3 POLY	31
2.9.2.4 REFIN	31
2.9.2.5 REFOUT	31
2.9.2.6 WIDTH	31
2.9.2.7 XOROUT	32
2.10 CRC_TypeDef结构体 参考	32
2.10.1 详细描述	32
2.10.2 结构体成员变量说明	32
2.10.2.1 CR	32
2.10.2.2 DR	33
2.10.2.3 INIT	33
2.10.2.4 POL	33
2.11 DAC_InitTypeDef结构体 参考	33
2.11.1 详细描述	33
2.11.2 结构体成员变量说明	34
2.11.2.1 DAC_RANGE	34
2.11.2.2 DAC_STEP	34
2.11.2.3 DAC_STEP_TRG_x	34
2.11.2.4 DACOUT_EN	34
2.12 DMA_InitTypeDef结构体 参考	34
2.12.1 详细描述	35
2.12.2 结构体成员变量说明	35
2.12.2.1 CIRC	35
2.12.2.2 DADR	35
2.12.2.3 DBTW	36
2.12.2.4 DINC	36
2.12.2.5 EN	36
2.12.2.6 IRQ_EN	36
2.12.2.7 REN	36
2.12.2.8 RMODE	36
2.12.2.9 SADR	37
2.12.2.10 SBTW	37
2.12.2.11 SINC	37
2.12.2.12 TMS	37
2.13 DMA_TypeDef结构体 参考	37
2.13.1 详细描述	38
2.13.2 结构体成员变量说明	38

2.13.2.1 CCR	38
2.13.2.2 CTMS	38
2.13.2.3 DADR	38
2.13.2.4 REN	38
2.13.2.5 SADR	38
2.14 DMAMODULE_TypeDef结构体 参考	39
2.14.1 详细描述	39
2.14.2 结构体成员变量说明	39
2.14.2.1 CTRL	39
2.14.2.2 IE	39
2.14.2.3 IF	39
2.15 DSP_TypeDef结构体 参考	40
2.15.1 详细描述	40
2.15.2 结构体成员变量说明	40
2.15.2.1 DID	40
2.15.2.2 DID1	41
2.15.2.3 DIS	41
2.15.2.4 DIS1	41
2.15.2.5 QUO	41
2.15.2.6 QUO1	41
2.15.2.7 RAD	41
2.15.2.8 REM	42
2.15.2.9 RESERVED	42
2.15.2.10 RESERVED1	42
2.15.2.11 RESERVED2	42
2.15.2.12 SQRT	42
2.16 EEPROM_InitTypeDef结构体 参考	42
2.16.1 详细描述	43
2.16.2 结构体成员变量说明	43
2.16.2.1 addr	43
2.16.2.2 ADDR_LEN_x	43
2.16.2.3 EEPROM_BAUD_x	43
2.16.2.4 IO_SWITCH	44
2.16.2.5 RDY_RLS	44
2.17 EXTI_TypeDef结构体 参考	44
2.17.1 详细描述	45
2.17.2 结构体成员变量说明	45
2.17.2.1 CLKO	45
2.17.2.2 CR0	45
2.17.2.3 CR1	45
2.17.2.4 IE	45
2.17.2.5 IF	46

2.17.2.6 SWAP	46
2.18 FLASH_TypeDef结构体 参考	46
2.18.1 详细描述	47
2.18.2 结构体成员变量说明	47
2.18.2.1 ADDR	47
2.18.2.2 CFG	48
2.18.2.3 EPSM	48
2.18.2.4 EPSN	48
2.18.2.5 ERASE	48
2.18.2.6 LIB_ADDR	49
2.18.2.7 NADDR	49
2.18.2.8 NCFG	49
2.18.2.9 NERASE	49
2.18.2.10 NKEY	49
2.18.2.11 NRDATA	50
2.18.2.12 NWDATA	50
2.18.2.13 PROTECT	50
2.18.2.14 RDATA	50
2.18.2.15 READY	50
2.18.2.16 RESERVED	50
2.18.2.17 SIZE	51
2.18.2.18 WDATA	51
2.19 GPIO_InitTypeDef结构体 参考	51
2.19.1 详细描述	51
2.19.2 结构体成员变量说明	51
2.19.2.1 GPIO_Mode	51
2.19.2.2 GPIO_PFLT	52
2.19.2.3 GPIO_Pin	52
2.19.2.4 GPIO_PODEna	52
2.19.2.5 GPIO_PuPd	52
2.20 GPIO_TypeDef结构体 参考	52
2.20.1 详细描述	54
2.20.2 结构体成员变量说明	54
2.20.2.1 BRR	54
2.20.2.2 BSRR	54
2.20.2.3 F3210	55
2.20.2.4 F7654	55
2.20.2.5 FBA98	55
2.20.2.6 FFEDC	55
2.20.2.7 PDI	55
2.20.2.8 PDO	55
2.20.2.9 PFLT	56

2.20.2.10 PIE	56
2.20.2.11 PODE	56
2.20.2.12 POE	56
2.20.2.13 PUE	57
2.20.2.14 RESERVED	57
2.21 HALL_InitTypeDef结构体 参考	57
2.21.1 详细描述	57
2.21.2 结构体成员变量说明	58
2.21.2.1 Capture_IRQ_Ena	58
2.21.2.2 ClockDivision	58
2.21.2.3 CountTH	58
2.21.2.4 Filter75_Ena	58
2.21.2.5 FilterLen	58
2.21.2.6 HALL_Ena	58
2.21.2.7 OverFlow_IRQ_Ena	59
2.21.2.8 softIE	59
2.22 HALL_TypeDef结构体 参考	59
2.22.1 详细描述	59
2.22.2 结构体成员变量说明	60
2.22.2.1 CFG	60
2.22.2.2 CNT	60
2.22.2.3 INFO	60
2.22.2.4 TH	60
2.22.2.5 WIDTH	60
2.23 I2C_InitTypeDef结构体 参考	61
2.23.1 详细描述	61
2.23.2 结构体成员变量说明	61
2.23.2.1 ADDR	61
2.23.2.2 ADRCMP	62
2.23.2.3 BaudRate	62
2.23.2.4 BURST_ADDR_CMP	62
2.23.2.5 BURST_NACK	62
2.23.2.6 BUS_ERR_IE	62
2.23.2.7 BUSRT_EN	62
2.23.2.8 IE	63
2.23.2.9 MST_MODE	63
2.23.2.10 SLV_MODE	63
2.23.2.11 STOP_IE	63
2.23.2.12 TC_IE	63
2.24 I2C_TypeDef结构体 参考	63
2.24.1 详细描述	64
2.24.2 结构体成员变量说明	64

2.24.2.1 ADDR	64
2.24.2.2 BCR	64
2.24.2.3 BSIZE	65
2.24.2.4 CFG	65
2.24.2.5 DATA	65
2.24.2.6 MSCR	65
2.24.2.7 SCR	65
2.25 iictmpval结构体 参考	65
2.25.1 详细描述	66
2.25.2 结构体成员变量说明	66
2.25.2.1 nackflg	66
2.25.2.2 rxdata	66
2.25.2.3 rxdatap	66
2.25.2.4 state	66
2.25.2.5 txdata	66
2.25.2.6 txdatap	66
2.26 IWDG_InitTypeDef结构体 参考	67
2.26.1 详细描述	67
2.26.2 结构体成员变量说明	67
2.26.2.1 DWK_EN	67
2.26.2.2 RTH	67
2.26.2.3 WDG_EN	67
2.26.2.4 WTH	68
2.27 IWDG_TypeDef结构体 参考	68
2.27.1 详细描述	68
2.27.2 结构体成员变量说明	68
2.27.2.1 CFG	68
2.27.2.2 CLR	69
2.27.2.3 CNT	69
2.27.2.4 PSW	69
2.27.2.5 RTH	69
2.27.2.6 WTH	69
2.28 MCPWM_InitTypeDef结构体 参考	69
2.28.1 详细描述	70
2.28.2 结构体成员变量说明	70
2.28.2.1 AUEN	70
2.28.2.2 CLK_DIV	70
2.28.2.3 COUNT	70
2.28.2.4 DTHN	71
2.28.2.5 DTHP	71
2.28.2.6 EIE	71
2.28.2.7 EN	71

2.28.2.8 FAIL0	71
2.28.2.9 FAIL1	71
2.28.2.10 FAIL_IO	71
2.28.2.11 FLT_DIV	71
2.28.2.12 IE	72
2.28.2.13 IO0	72
2.28.2.14 IO1	72
2.28.2.15 IO2	72
2.28.2.16 IO3	72
2.28.2.17 RE	72
2.28.2.18 TH	72
2.28.2.19 TMR0	72
2.28.2.20 TMR1	73
2.28.2.21 TMR2	73
2.28.2.22 TMR3	73
2.28.2.23 TR	73
2.29 MCPWM_TypeDef结构体 参考	73
2.29.1 详细描述	76
2.29.2 结构体成员变量说明	76
2.29.2.1 AUEN	76
2.29.2.2 CH_DEF	76
2.29.2.3 CH_FAIL	76
2.29.2.4 CH_MSK	77
2.29.2.5 CNT0	77
2.29.2.6 DTH00	77
2.29.2.7 DTH01	77
2.29.2.8 EIE	77
2.29.2.9 EIF	77
2.29.2.10 EVT0	78
2.29.2.11 FCNT	78
2.29.2.12 FLT	78
2.29.2.13 IE0	78
2.29.2.14 IF0	79
2.29.2.15 IO01	79
2.29.2.16 IO23	79
2.29.2.17 PP	79
2.29.2.18 PRT	80
2.29.2.19 RE	80
2.29.2.20 RESERVED	80
2.29.2.21 RESERVED1	80
2.29.2.22 RESERVED2	80
2.29.2.23 RESERVED3	80

2.29.2.24 RESERVED4	81
2.29.2.25 RESERVED5	81
2.29.2.26 RESERVED6	81
2.29.2.27 RESERVED7	81
2.29.2.28 RESERVED8	81
2.29.2.29 SDCFG	81
2.29.2.30 SW_FAIL	81
2.29.2.31 TCLK	82
2.29.2.32 TH0	82
2.29.2.33 TH00	82
2.29.2.34 TH01	82
2.29.2.35 TH10	82
2.29.2.36 TH11	82
2.29.2.37 TH20	83
2.29.2.38 TH21	83
2.29.2.39 TH30	83
2.29.2.40 TH31	83
2.29.2.41 TMR0	83
2.29.2.42 TMR1	83
2.29.2.43 TMR2	84
2.29.2.44 TMR3	84
2.29.2.45 UPDATE	84
2.30 OPA_InitTypeDef结构体 参考	84
2.30.1 详细描述	85
2.30.2 结构体成员变量说明	85
2.30.2.1 OPA_CLEna	85
2.30.2.2 OPA_Gain	85
2.30.2.3 OPA_IT	85
2.31 QEP_InitTypeDef结构体 参考	85
2.31.1 详细描述	86
2.31.2 结构体成员变量说明	86
2.31.2.1 ClockDiv	86
2.31.2.2 FE_CNT_EN	86
2.31.2.3 Filter	86
2.31.2.4 IRQEna	87
2.31.2.5 Mode	87
2.31.2.6 TH	87
2.31.2.7 ZC	87
2.32 QEP_TypeDef结构体 参考	87
2.32.1 详细描述	88
2.32.2 结构体成员变量说明	88
2.32.2.1 CFG	88

2.32.2.2 CNT	88
2.32.2.3 IE	88
2.32.2.4 IF	88
2.32.2.5 TH	89
2.33 SIF_InitTypeDef结构体 参考	89
2.33.1 详细描述	90
2.33.2 结构体成员变量说明	90
2.33.2.1 SIF_DMA_EN	90
2.33.2.2 SIF_DONE	90
2.33.2.3 SIF_DONE_VLD	90
2.33.2.4 SIF_EN	90
2.33.2.5 SIF_IRQ_EN	90
2.33.2.6 SIF_IRQ_IF	91
2.33.2.7 SIF_MSB	91
2.33.2.8 SIF_RATIO	91
2.33.2.9 SIF_SYNC	91
2.33.2.10 SIF_SYNC_PULSE	91
2.33.2.11 SIF_SYNC_VLD	91
2.33.2.12 SIF_TDTH1	92
2.33.2.13 SIF_TOSC	92
2.33.2.14 SIF_TSTH1	92
2.34 SIF_TypeDef结构体 参考	92
2.34.1 详细描述	93
2.34.2 结构体成员变量说明	93
2.34.2.1 CFG	93
2.34.2.2 IRQ	93
2.34.2.3 TDTH1	93
2.34.2.4 TOSC	93
2.34.2.5 TSTH1	93
2.34.2.6 WDATA	94
2.35 SPI_InitTypeDef结构体 参考	94
2.35.1 详细描述	94
2.35.2 结构体成员变量说明	95
2.35.2.1 BaudRate	95
2.35.2.2 ByteLength	95
2.35.2.3 CPHA	95
2.35.2.4 CPOL	95
2.35.2.5 CS	95
2.35.2.6 Duplex	95
2.35.2.7 EN	96
2.35.2.8 ENDIAN	96
2.35.2.9 IRQEna	96

2.35.2.10 Mode	96
2.35.2.11 TRANS_MODE	96
2.35.2.12 Trig	96
2.36 SPI_TypeDef结构体 参考	97
2.36.1 详细描述	97
2.36.2 结构体成员变量说明	97
2.36.2.1 CFG	97
2.36.2.2 DIV	97
2.36.2.3 IE	98
2.36.2.4 RX_DATA	98
2.36.2.5 SIZE	98
2.36.2.6 TX_DATA	98
2.37 SYS_TypeDef结构体 参考	98
2.37.1 详细描述	100
2.37.2 结构体成员变量说明	101
2.37.2.1 AFE_ADC	101
2.37.2.2 AFE_DAC0	101
2.37.2.3 AFE_DAC0_AMC	101
2.37.2.4 AFE_DAC0_DC	101
2.37.2.5 AFE_DAC1	101
2.37.2.6 AFE_DAC1_AMC	101
2.37.2.7 AFE_DAC1_DC	102
2.37.2.8 AFE_DAC_CTRL	102
2.37.2.9 AFE_DBG	102
2.37.2.10 AFE_INFO	102
2.37.2.11 AFE_REG0	102
2.37.2.12 AFE_REG1	102
2.37.2.13 AFE_REG2	103
2.37.2.14 AFE_REG3	103
2.37.2.15 AFE_REG4	103
2.37.2.16 AFE_REG5	103
2.37.2.17 AFE_REG7	103
2.37.2.18 CLK_CFG	104
2.37.2.19 CLK_DIV0	104
2.37.2.20 CLK_DIV1	104
2.37.2.21 CLK_DIV2	104
2.37.2.22 CLK_FEN	104
2.37.2.23 DBG_CFG	104
2.37.2.24 FLSE	105
2.37.2.25 FLSP	105
2.37.2.26 IF	105
2.37.2.27 IO_CFG	105

2.37.2.28 NMI_TH	105
2.37.2.29 OPA_SEL	105
2.37.2.30 PROTECT	106
2.37.2.31 RAM_LOCK	106
2.37.2.32 RESERVED	106
2.37.2.33 RESERVED1	106
2.37.2.34 RESERVED2	106
2.37.2.35 RESERVED3	106
2.37.2.36 RESERVED41	107
2.37.2.37 RESERVED5	107
2.37.2.38 RESERVED6	107
2.37.2.39 RESERVED7	107
2.37.2.40 SFT_RST	107
2.38 TIMER_TimerInitTypeDef结构体 参考	107
2.38.1 详细描述	109
2.38.2 结构体成员变量说明	109
2.38.2.1 CAP0_CLR_EN	109
2.38.2.2 CAP1_CLR_EN	110
2.38.2.3 CENTER	110
2.38.2.4 CH0_DEFAULT	110
2.38.2.5 CH0_FE_CAP_EN	110
2.38.2.6 CH0_RE_CAP_EN	110
2.38.2.7 CH0_WorkMode	110
2.38.2.8 CH0Output	111
2.38.2.9 CH1_DEFAULT	111
2.38.2.10 CH1_FE_CAP_EN	111
2.38.2.11 CH1_RE_CAP_EN	111
2.38.2.12 CH1_WorkMode	111
2.38.2.13 CH1Output	111
2.38.2.14 CHN0	112
2.38.2.15 CHN1	112
2.38.2.16 ClockDiv	112
2.38.2.17 CMP0_TRIGGER_MODE	112
2.38.2.18 CMP1_TRIGGER_MODE	112
2.38.2.19 CNT	112
2.38.2.20 DIR	113
2.38.2.21 EN	113
2.38.2.22 ETON	113
2.38.2.23 EVT	113
2.38.2.24 FAIL_EN	113
2.38.2.25 FAIL_POL	113
2.38.2.26 FAIL_SEL	114

2.38.2.27 FLT	114
2.38.2.28 GATE_EN	114
2.38.2.29 HALT_PRT	114
2.38.2.30 IE	114
2.38.2.31 MOE	114
2.38.2.32 ONE_TRIG	115
2.38.2.33 RL_EN	115
2.38.2.34 SHADOW	115
2.38.2.35 SRC0	115
2.38.2.36 SRC1	115
2.38.2.37 TH	115
2.38.2.38 XCLK_EN	116
2.39 TIMER_TypeDef结构体 参考	116
2.39.1 详细描述	117
2.39.2 结构体成员变量说明	117
2.39.2.1 CFG0	117
2.39.2.2 CFG1	117
2.39.2.3 CFG2	117
2.39.2.4 CHN0	117
2.39.2.5 CHN1	118
2.39.2.6 CNT	118
2.39.2.7 EVT	118
2.39.2.8 FLT	118
2.39.2.9 IE	118
2.39.2.10 IF	118
2.39.2.11 IO	119
2.39.2.12 RESERVED	119
2.39.2.13 TH	119
2.40 UART_InitTypeDef结构体 参考	119
2.40.1 详细描述	120
2.40.2 结构体成员变量说明	120
2.40.2.1 ABD_EN	120
2.40.2.2 BaudRate	120
2.40.2.3 Bit9Value	120
2.40.2.4 CK_EN	121
2.40.2.5 DMARE	121
2.40.2.6 DUPLEX	121
2.40.2.7 FirstSend	121
2.40.2.8 IRQEna	121
2.40.2.9 LBDL	121
2.40.2.10 LIN_EN	122
2.40.2.11 Match485Addr	122

2.40.2.12 MultiDropEna	122
2.40.2.13 ParityMode	122
2.40.2.14 RXD_INV	122
2.40.2.15 StopBits	122
2.40.2.16 TXD_INV	123
2.40.2.17 WordLength	123
2.41 UART_TypeDef结构体 参考	123
2.41.1 详细描述	124
2.41.2 结构体成员变量说明	124
2.41.2.1 ADR	124
2.41.2.2 BUFF	124
2.41.2.3 CTRL	124
2.41.2.4 DIVH	124
2.41.2.5 DIVL	124
2.41.2.6 IE	125
2.41.2.7 IF	125
2.41.2.8 IOC	125
2.41.2.9 RE	125
2.41.2.10 STT	125
3 文件说明	127
3.1 include/lks32mc09x_adc.h 文件参考	127
3.1.1 宏定义说明	131
3.1.1.1 ADC_IE_AWD0	131
3.1.1.2 ADC_IE_HERR	131
3.1.1.3 ADC_IE_ISF	131
3.1.1.4 ADC_IE_SERR	132
3.1.1.5 ADC_IE_SF1	132
3.1.1.6 ADC_IE_SF2	132
3.1.1.7 ADC_IF_ALL	132
3.1.1.8 ADC_IF_AWD0	132
3.1.1.9 ADC_IF_HERR	132
3.1.1.10 ADC_IF_ISF	133
3.1.1.11 ADC_IF_SERR	133
3.1.1.12 ADC_IF_SF1	133
3.1.1.13 ADC_IF_SF2	133
3.1.1.14 ADC_RE_AWD0	133
3.1.1.15 ADC_RE_HERR	133
3.1.1.16 ADC_RE_ISF	134
3.1.1.17 ADC_RE_SERR	134
3.1.1.18 ADC_RE_SF1	134
3.1.1.19 ADC_RE_SF2	134

3.1.1.20	ADC_TRIG.CL_IT0	134
3.1.1.21	ADC_TRIG.CL_IT1	134
3.1.1.22	ADC_TRIG.CL_IT2	135
3.1.1.23	ADC_TRIG.CL_IT3	135
3.1.1.24	ADC_TRIG.CL_NT0	135
3.1.1.25	ADC_TRIG.CL_NT1	135
3.1.1.26	ADC_TRIG.CL_NT2	135
3.1.1.27	ADC_TRIG.CL_NT3	135
3.1.1.28	ADC_TRIG.MCPWM0_T0	136
3.1.1.29	ADC_TRIG.MCPWM0_T0_IEN	136
3.1.1.30	ADC_TRIG.MCPWM0_T1	136
3.1.1.31	ADC_TRIG.MCPWM0_T1_IEN	136
3.1.1.32	ADC_TRIG.MCPWM0_T2	136
3.1.1.33	ADC_TRIG.MCPWM0_T2_IEN	136
3.1.1.34	ADC_TRIG.MCPWM0_T3	137
3.1.1.35	ADC_TRIG.MCPWM0_T3_IEN	137
3.1.1.36	ADC_TRIG.SOFT	137
3.1.1.37	ADC_TRIG.TIMER0_CMP0	137
3.1.1.38	ADC_TRIG.TIMER0_CMP0_IEN	137
3.1.1.39	ADC_TRIG.TIMER0_CMP1	137
3.1.1.40	ADC_TRIG.TIMER0_CMP1_IEN	138
3.1.1.41	ADC_TRIG.TIMER1_CMP0	138
3.1.1.42	ADC_TRIG.TIMER1_CMP1	138
3.1.1.43	ADC_TRIG.TIMER2_CMP0_IEN	138
3.1.1.44	ADC_TRIG.TIMER2_CMP1_IEN	138
3.1.2	枚举类型说明	138
3.1.2.1	ADC.CHN_x	138
3.1.2.2	ADC.DAT_x	140
3.1.2.3	ADC.GAIN_x	140
3.1.2.4	ADC.RANGE_x	141
3.1.3	函数说明	141
3.1.3.1	ADC.ClearIRQFlag()	141
3.1.3.2	ADC.Disable()	141
3.1.3.3	ADC.Enable()	141
3.1.3.4	ADC.GetConversionValue()	142
3.1.3.5	ADC.GetConversionValueS32()	143
3.1.3.6	ADC.GetIRQFlag()	143
3.1.3.7	ADC.Init()	144
3.1.3.8	ADC.Reset()	144
3.1.3.9	ADC.SetChn()	145
3.1.3.10	ADC.SetGain()	145
3.1.3.11	ADC.SetRange()	146

3.1.3.12 ADC_SoftIdleTrig()	146
3.1.3.13 ADC_SoftTrig()	147
3.1.3.14 ADC_StructInit()	147
3.2 lks32mc09x_adc.h	147
3.3 include/lks32mc09x_clu.h 文件参考	150
3.3.1 宏定义说明	154
3.3.1.1 CLU_FN_0	154
3.3.1.2 CLU_FN_1	155
3.3.1.3 CLU_FN_A	155
3.3.1.4 CLU_FN_A.B	155
3.3.1.5 CLU_FN_A.B.C	155
3.3.1.6 CLU_FN_A.B.NOTC	155
3.3.1.7 CLU_FN_A.C	155
3.3.1.8 CLU_FN_A.NOTB	156
3.3.1.9 CLU_FN_A.NOTB.C	156
3.3.1.10 CLU_FN_A.NOTB.NOTC	156
3.3.1.11 CLU_FN_A.NOTC	156
3.3.1.12 CLU_FN_B	156
3.3.1.13 CLU_FN_B.C	156
3.3.1.14 CLU_FN_B.NOTC	157
3.3.1.15 CLU_FN_C	157
3.3.1.16 CLU_FN_NOTA	157
3.3.1.17 CLU_FN_NOTA.B	157
3.3.1.18 CLU_FN_NOTA.B.C	157
3.3.1.19 CLU_FN_NOTA.B.NOTC	157
3.3.1.20 CLU_FN_NOTA.C	158
3.3.1.21 CLU_FN_NOTA.NOTB	158
3.3.1.22 CLU_FN_NOTA.NOTB.C	158
3.3.1.23 CLU_FN_NOTA.NOTB.NOTC	158
3.3.1.24 CLU_FN_NOTA.NOTC	158
3.3.1.25 CLU_FN_NOTB	158
3.3.1.26 CLU_FN_NOTB.C	159
3.3.1.27 CLU_FN_NOTB.NOTC	159
3.3.1.28 CLU_FN_NOTC	159
3.3.1.29 CLU_MODEL_CLU0	159
3.3.1.30 CLU_MODEL_CLU1	159
3.3.1.31 CLU_MODEL_CLU2	159
3.3.1.32 CLU_MODEL_CLU3	160
3.3.1.33 CLU_MXA_CLU0_CLU0OUT	160
3.3.1.34 CLU_MXA_CLU0_CLU1OUT	160
3.3.1.35 CLU_MXA_CLU0_CLU2OUT	160
3.3.1.36 CLU_MXA_CLU0_CLU3OUT	160

3.3.1.37 CLU_MXA_CLU0.P0_0	160
3.3.1.38 CLU_MXA_CLU0.P0_12	160
3.3.1.39 CLU_MXA_CLU0.P0_14	161
3.3.1.40 CLU_MXA_CLU0.P0_3	161
3.3.1.41 CLU_MXA_CLU0.P0_5	161
3.3.1.42 CLU_MXA_CLU0.P0_7	161
3.3.1.43 CLU_MXA_CLU0.P1_0	161
3.3.1.44 CLU_MXA_CLU0.P2_12	161
3.3.1.45 CLU_MXA_CLU0.PWM0	161
3.3.1.46 CLU_MXA_CLU0.PWM1	161
3.3.1.47 CLU_MXA_CLU0.TIMER0ch0ovfl	162
3.3.1.48 CLU_MXA_CLU0.TIMER1ch0ovfl	162
3.3.1.49 CLU_MXA_CLU1.CLU0OUT	162
3.3.1.50 CLU_MXA_CLU1.CLU1OUT	162
3.3.1.51 CLU_MXA_CLU1.CLU2OUT	162
3.3.1.52 CLU_MXA_CLU1.CLU3OUT	162
3.3.1.53 CLU_MXA_CLU1.P0_0	162
3.3.1.54 CLU_MXA_CLU1.P0_11	162
3.3.1.55 CLU_MXA_CLU1.P0_14	163
3.3.1.56 CLU_MXA_CLU1.P0_15	163
3.3.1.57 CLU_MXA_CLU1.P0_2	163
3.3.1.58 CLU_MXA_CLU1.P0_5	163
3.3.1.59 CLU_MXA_CLU1.P0_6	163
3.3.1.60 CLU_MXA_CLU1.P2_12	163
3.3.1.61 CLU_MXA_CLU1.PWM2	163
3.3.1.62 CLU_MXA_CLU1.PWM3	163
3.3.1.63 CLU_MXA_CLU1.TIMER0ch1ovfl	164
3.3.1.64 CLU_MXA_CLU1.TIMER1ch1ovfl	164
3.3.1.65 CLU_MXA_CLU2.CLU0OUT	164
3.3.1.66 CLU_MXA_CLU2.CLU1OUT	164
3.3.1.67 CLU_MXA_CLU2.CLU2OUT	164
3.3.1.68 CLU_MXA_CLU2.CLU3OUT	164
3.3.1.69 CLU_MXA_CLU2.P1_10	164
3.3.1.70 CLU_MXA_CLU2.P1_13	164
3.3.1.71 CLU_MXA_CLU2.P1_15	165
3.3.1.72 CLU_MXA_CLU2.P2_1	165
3.3.1.73 CLU_MXA_CLU2.P2_14	165
3.3.1.74 CLU_MXA_CLU2.P2_4	165
3.3.1.75 CLU_MXA_CLU2.P2_6	165
3.3.1.76 CLU_MXA_CLU2.P2_8	165
3.3.1.77 CLU_MXA_CLU2.PWM0	165
3.3.1.78 CLU_MXA_CLU2.PWM2	165

3.3.1.79 CLU_MXA_CLU2_TIMER2ch0ovfl	166
3.3.1.80 CLU_MXA_CLU2_TIMER3ch0ovfl	166
3.3.1.81 CLU_MXA_CLU3_CLU0OUT	166
3.3.1.82 CLU_MXA_CLU3_CLU1OUT	166
3.3.1.83 CLU_MXA_CLU3_CLU2OUT	166
3.3.1.84 CLU_MXA_CLU3_CLU3OUT	166
3.3.1.85 CLU_MXA_CLU3_P1_15	166
3.3.1.86 CLU_MXA_CLU3_P2_0	166
3.3.1.87 CLU_MXA_CLU3_P2_1	167
3.3.1.88 CLU_MXA_CLU3_P2_13	167
3.3.1.89 CLU_MXA_CLU3_P2_3	167
3.3.1.90 CLU_MXA_CLU3_P2_6	167
3.3.1.91 CLU_MXA_CLU3_P2_7	167
3.3.1.92 CLU_MXA_CLU3_P2_8	167
3.3.1.93 CLU_MXA_CLU3_PWM1	167
3.3.1.94 CLU_MXA_CLU3_PWM3	168
3.3.1.95 CLU_MXA_CLU3_TIMER2ch1ovfl	168
3.3.1.96 CLU_MXA_CLU3_TIMER3ch1ovfl	168
3.3.1.97 CLU_MXB_CLU0_ADBUSY0	168
3.3.1.98 CLU_MXB_CLU0_CLU0OUT	168
3.3.1.99 CLU_MXB_CLU0_CLU1OUT	168
3.3.1.100 CLU_MXB_CLU0_CLU2OUT	168
3.3.1.101 CLU_MXB_CLU0_CLU3OUT	168
3.3.1.102 CLU_MXB_CLU0_CMP0_OUT	169
3.3.1.103 CLU_MXB_CLU0_P0_11	169
3.3.1.104 CLU_MXB_CLU0_P0_13	169
3.3.1.105 CLU_MXB_CLU0_P0_15	169
3.3.1.106 CLU_MXB_CLU0_P0_2	169
3.3.1.107 CLU_MXB_CLU0_P0_4	169
3.3.1.108 CLU_MXB_CLU0_P0_6	169
3.3.1.109 CLU_MXB_CLU0_P2_11	169
3.3.1.110 CLU_MXB_CLU0_P2_7	170
3.3.1.111 CLU_MXB_CLU0_PWM3	170
3.3.1.112 CLU_MXB_CLU0_SPI_MOSI	170
3.3.1.113 CLU_MXB_CLU1_ADBUSY0	170
3.3.1.114 CLU_MXB_CLU1_CLU0OUT	170
3.3.1.115 CLU_MXB_CLU1_CLU1OUT	170
3.3.1.116 CLU_MXB_CLU1_CLU2OUT	170
3.3.1.117 CLU_MXB_CLU1_CLU3OUT	170
3.3.1.118 CLU_MXB_CLU1_CMP1_OUT	171
3.3.1.119 CLU_MXB_CLU1_P0_12	171
3.3.1.120 CLU_MXB_CLU1_P0_13	171

3.3.1.121 CLU_MXB_CLU1_P0.3	171
3.3.1.122 CLU_MXB_CLU1_P0.4	171
3.3.1.123 CLU_MXB_CLU1_P0.7	171
3.3.1.124 CLU_MXB_CLU1_P1.0	171
3.3.1.125 CLU_MXB_CLU1_P2.11	171
3.3.1.126 CLU_MXB_CLU1_P2.7	172
3.3.1.127 CLU_MXB_CLU1_PWM1	172
3.3.1.128 CLU_MXB_CLU1_SPIMISO	172
3.3.1.129 CLU_MXB_CLU2_CLU0OUT	172
3.3.1.130 CLU_MXB_CLU2_CLU1OUT	172
3.3.1.131 CLU_MXB_CLU2_CLU2OUT	172
3.3.1.132 CLU_MXB_CLU2_CLU3OUT	172
3.3.1.133 CLU_MXB_CLU2_I2CSDA	172
3.3.1.134 CLU_MXB_CLU2_P1.11	173
3.3.1.135 CLU_MXB_CLU2_P1.14	173
3.3.1.136 CLU_MXB_CLU2_P2.0	173
3.3.1.137 CLU_MXB_CLU2_P2.13	173
3.3.1.138 CLU_MXB_CLU2_P2.15	173
3.3.1.139 CLU_MXB_CLU2_P2.3	173
3.3.1.140 CLU_MXB_CLU2_P2.5	173
3.3.1.141 CLU_MXB_CLU2_P3.9	173
3.3.1.142 CLU_MXB_CLU2_PWM1	174
3.3.1.143 CLU_MXB_CLU2_PWM3	174
3.3.1.144 CLU_MXB_CLU2_SPICLK	174
3.3.1.145 CLU_MXB_CLU3_CLU0OUT	174
3.3.1.146 CLU_MXB_CLU3_CLU1OUT	174
3.3.1.147 CLU_MXB_CLU3_CLU2OUT	174
3.3.1.148 CLU_MXB_CLU3_CLU3OUT	174
3.3.1.149 CLU_MXB_CLU3_I2CSCL	174
3.3.1.150 CLU_MXB_CLU3_P1.10	175
3.3.1.151 CLU_MXB_CLU3_P1.11	175
3.3.1.152 CLU_MXB_CLU3_P1.13	175
3.3.1.153 CLU_MXB_CLU3_P1.14	175
3.3.1.154 CLU_MXB_CLU3_P2.14	175
3.3.1.155 CLU_MXB_CLU3_P2.15	175
3.3.1.156 CLU_MXB_CLU3_P2.4	175
3.3.1.157 CLU_MXB_CLU3_P2.5	175
3.3.1.158 CLU_MXB_CLU3_PWM0	176
3.3.1.159 CLU_MXB_CLU3_PWM2	176
3.3.1.160 CLU_MXB_CLU3_SPICSN	176
3.3.2 函数说明	176
3.3.2.1 CLU_Disable()	176

3.3.2.2 CLU_Enable()	177
3.3.2.3 CLU_Init()	177
3.3.2.4 CLU_ReadOutValue()	178
3.3.2.5 CLU_StructInit()	178
3.4 lks32mc09x_clu.h	178
3.5 include/lks32mc09x_cmp.h 文件参考	181
3.5.1 宏定义说明	184
3.5.1.1 CMP_CHN_0	184
3.5.1.2 CMP_CHN_1	184
3.5.1.3 CMP_CLK_DIV_0	184
3.5.1.4 CMP_CLK_DIV_1	185
3.5.1.5 CMP_CLK_DIV_2	185
3.5.1.6 CMP_CLK_DIV_3	185
3.5.1.7 CMP_CLK_DIV_4	185
3.5.1.8 CMP_HYS_0mV	185
3.5.1.9 CMP_HYS_20mV	185
3.5.1.10 CMP_IE_CMP0	185
3.5.1.11 CMP_IE_CMP1	186
3.5.1.12 CMP_IF_CMP0	186
3.5.1.13 CMP_IF_CMP1	186
3.5.1.14 CMP_RE_CMP0	186
3.5.1.15 CMP_RE_CMP1	186
3.5.1.16 CMP_SELN_0_DAC0	186
3.5.1.17 CMP_SELN_0_DAC1	186
3.5.1.18 CMP_SELN_0_HS_MID0	187
3.5.1.19 CMP_SELN_0_IN_CMP0	187
3.5.1.20 CMP_SELN_0_VBGP	187
3.5.1.21 CMP_SELN_1_DAC0	187
3.5.1.22 CMP_SELN_1_DAC1	187
3.5.1.23 CMP_SELN_1_HS_MID1	187
3.5.1.24 CMP_SELN_1_IN_CMP1	188
3.5.1.25 CMP_SELN_1_VBGP	188
3.5.1.26 CMP_SELP_0_CMP0_IP0	188
3.5.1.27 CMP_SELP_0_CMP0_IP1	188
3.5.1.28 CMP_SELP_0_CMP0_IP2	188
3.5.1.29 CMP_SELP_0_CMP0_IP3	188
3.5.1.30 CMP_SELP_0_CMP0_IP4	189
3.5.1.31 CMP_SELP_0_CMP1_IP1	189
3.5.1.32 CMP_SELP_0_CMP1_IP2	189
3.5.1.33 CMP_SELP_0_CMP1_IP3	189
3.5.1.34 CMP_SELP_0_DAC0_OUT	189
3.5.1.35 CMP_SELP_0_OPA0_IP	189

3.5.1.36	CMP_SEL0_OPA0_OUTP	190
3.5.1.37	CMP_SEL0_OPA1_IP	190
3.5.1.38	CMP_SEL0_OPA1_OUTP	190
3.5.1.39	CMP_SEL0_OUT_OPA_CMP	190
3.5.1.40	CMP_SEL1_CMP0_IP1	190
3.5.1.41	CMP_SEL1_CMP0_IP2	190
3.5.1.42	CMP_SEL1_CMP0_IP3	191
3.5.1.43	CMP_SEL1_CMP1_IP0	191
3.5.1.44	CMP_SEL1_CMP1_IP1	191
3.5.1.45	CMP_SEL1_CMP1_IP2	191
3.5.1.46	CMP_SEL1_CMP1_IP3	191
3.5.1.47	CMP_SEL1_DAC1_OUT	191
3.5.1.48	CMP_SEL1_OPA2_IP	192
3.5.1.49	CMP_SEL1_OPA2_OUTP	192
3.5.1.50	CMP_SEL1_OPA3_IP	192
3.5.1.51	CMP_SEL1_OPA3_OUTP	192
3.5.1.52	CMP_SEL1_OPA_OUT_CMP	192
3.5.1.53	CMP_SIGNAL_EDGE_N	192
3.5.1.54	CMP_SIGNAL_EDGE_P	193
3.5.1.55	CMP_SIGNAL_LEVEL_H	193
3.5.1.56	CMP_SIGNAL_LEVEL_L	193
3.5.1.57	CMP_SIGNAL_NONE	193
3.5.1.58	CMP_WIN_LEVEL_H	193
3.5.1.59	CMP_WIN_LEVEL_L	193
3.5.1.60	CMP_WIN_SIGNAL_MCPWM0_0	193
3.5.1.61	CMP_WIN_SIGNAL_MCPWM0_1	193
3.5.1.62	CMP_WIN_SIGNAL_MCPWM0_2	194
3.5.1.63	CMP_WIN_SIGNAL_MCPWM0_3	194
3.5.1.64	CMP_WIN_SIGNAL_NONE	194
3.5.2	函数说明	194
3.5.2.1	CMP_ClearIRQFlag()	194
3.5.2.2	CMP_Disable()	194
3.5.2.3	CMP_Enable()	195
3.5.2.4	CMP_GetFilterValue()	196
3.5.2.5	CMP_GetIRQFlag()	196
3.5.2.6	CMP_GetValue()	197
3.5.2.7	CMP_Init()	197
3.5.2.8	CMP_SetFilterWidth()	198
3.5.2.9	CMP_SetHysteresis()	198
3.5.2.10	CMP_SetIRQSignal()	199
3.5.2.11	CMP_SetSignal()	199
3.5.2.12	CMP_SetWindowSignal()	200

3.5.2.13 CMP_StructInit()	201
3.6 lks32mc09x_cmp.h	201
3.7 include/lks32mc09x_crc.h 文件参考	203
3.7.1 宏定义说明	205
3.7.1.1 CRC_MODEL_CRC_16_CCITT	205
3.7.1.2 CRC_MODEL_CRC_16_CCITT_FALSE	205
3.7.1.3 CRC_MODEL_CRC_16_DNP	205
3.7.1.4 CRC_MODEL_CRC_16_IBMx1	205
3.7.1.5 CRC_MODEL_CRC_16_MAXIM	205
3.7.1.6 CRC_MODEL_CRC_16_MODBU	205
3.7.1.7 CRC_MODEL_CRC_16_USBx1	206
3.7.1.8 CRC_MODEL_CRC_16_X25	206
3.7.1.9 CRC_MODEL_CRC_16_XMODEM	206
3.7.1.10 CRC_MODEL_CRC_32	206
3.7.1.11 CRC_MODEL_CRC_32_MPEG_2	206
3.7.1.12 CRC_MODEL_CRC_8	206
3.7.1.13 CRC_MODEL_CRC_8_ITU	207
3.7.1.14 CRC_MODEL_CRC_8_MAXIMx	207
3.7.1.15 CRC_MODEL_CRC_8_ROHC	207
3.7.1.16 CRC_WIDTH_16	207
3.7.1.17 CRC_WIDTH_32	207
3.7.1.18 CRC_WIDTH_7	207
3.7.1.19 CRC_WIDTH_8	208
3.7.2 函数说明	208
3.7.2.1 CRC_Calc()	208
3.7.2.2 CRC_CalcCommon()	208
3.7.2.3 CRC_Disable()	209
3.7.2.4 CRC_Enable()	210
3.7.2.5 CRC_Init()	210
3.7.2.6 CRC_StructInit()	211
3.8 lks32mc09x_crc.h	211
3.9 include/lks32mc09x_dac.h 文件参考	212
3.9.1 宏定义说明	214
3.9.1.1 DAC_CHANNEL_0	214
3.9.1.2 DAC_CHANNEL_1	215
3.9.1.3 DAC_RANGE_0_1V2	215
3.9.1.4 DAC_RANGE_0_4V76	215
3.9.1.5 DAC_RANGE_1_1V2	215
3.9.1.6 DAC_RANGE_1_3V	215
3.9.1.7 DAC_STEP_1	215
3.9.1.8 DAC_STEP_2	216
3.9.1.9 DAC_STEP_3	216

3.9.1.10 DAC_STEP__0	216
3.9.1.11 DAC_STEP__1	216
3.9.1.12 DAC_STEP__2	216
3.9.1.13 DAC_STEP__3	216
3.9.1.14 DAC_STEP__4	217
3.9.1.15 DAC_STEP_TRG.NONE	217
3.9.1.16 DAC_STEP_TRG.TIMER0	217
3.9.1.17 DAC_STEP_TRG.TIMER1	217
3.9.1.18 DAC_STEP_TRG.TIMER2	217
3.9.1.19 DAC_STEP_TRG.TIMER3	217
3.9.2 函数说明	217
3.9.2.1 DAC.Cmd()	217
3.9.2.2 DAC.Disable()	218
3.9.2.3 DAC.Enable()	219
3.9.2.4 DAC.Init()	219
3.9.2.5 DAC.Output()	220
3.9.2.6 DAC.OutputValue()	220
3.9.2.7 DAC.OutputVoltage()	221
3.9.2.8 DAC.Range()	222
3.9.2.9 DAC.Reset()	222
3.9.2.10 DAC.SetRange()	223
3.9.2.11 DAC.Step()	224
3.9.2.12 DAC.StructInit()	224
3.9.2.13 DAC.Value()	226
3.9.2.14 DAC.Voltage()	226
3.10 lks32mc09x_dac.h	227
3.11 include/lks32mc09x_dma.h 文件参考	228
3.11.1 宏定义说明	231
3.11.1.1 DMA.BYTE_TRANS	231
3.11.1.2 DMA.HALFWORD_TRANS	231
3.11.1.3 DMA.IE_CH0	232
3.11.1.4 DMA.IE_CH1	232
3.11.1.5 DMA.IE_CH2	232
3.11.1.6 DMA.IE_CH3	232
3.11.1.7 DMA.IF_CH0	232
3.11.1.8 DMA.IF_CH1	232
3.11.1.9 DMA.IF_CH2	232
3.11.1.10 DMA.IF_CH3	233
3.11.1.11 DMA.MEMORY_2_PERI	233
3.11.1.12 DMA.PERI_2_MEMORY	233
3.11.1.13 DMA.REN_ADC0	233
3.11.1.14 DMA.REN_ADC1	233

3.11.1.15 DMA_REN_CMP	233
3.11.1.16 DMA_REN_DSP	233
3.11.1.17 DMA_REN_GPIO	234
3.11.1.18 DMA_REN_HALL0	234
3.11.1.19 DMA_REN_I2C0_RX	234
3.11.1.20 DMA_REN_I2C0_TX	234
3.11.1.21 DMA_REN_MCPWM_RE0	234
3.11.1.22 DMA_REN_MCPWM_RE1	234
3.11.1.23 DMA_REN_SIF	234
3.11.1.24 DMA_REN_SPI_RX	234
3.11.1.25 DMA_REN_SPI_TX	235
3.11.1.26 DMA_REN_SW	235
3.11.1.27 DMA_REN_TIMER0	235
3.11.1.28 DMA_REN_TIMER1	235
3.11.1.29 DMA_REN_TIMER2	235
3.11.1.30 DMA_REN_TIMER3	235
3.11.1.31 DMA_REN_UART0_RX	235
3.11.1.32 DMA_REN_UART0_TX	235
3.11.1.33 DMA_REN_UART1_RX	236
3.11.1.34 DMA_REN_UART1_TX	236
3.11.1.35 DMA_REN_UART2_RX	236
3.11.1.36 DMA_REN_UART2_TX	236
3.11.1.37 DMA_REQ_ADC0	236
3.11.1.38 DMA_REQ_ADC1	236
3.11.1.39 DMA_REQ_CMP	236
3.11.1.40 DMA_REQ_DSP	237
3.11.1.41 DMA_REQ_GPIO	237
3.11.1.42 DMA_REQ_HALL	237
3.11.1.43 DMA_REQ_I2C0_RX	237
3.11.1.44 DMA_REQ_I2C0_TX	237
3.11.1.45 DMA_REQ_MCPWM0	237
3.11.1.46 DMA_REQ_MCPWM1	238
3.11.1.47 DMA_REQ_SIF	238
3.11.1.48 DMA_REQ_SPI_RX	238
3.11.1.49 DMA_REQ_SPI_TX	238
3.11.1.50 DMA_REQ_SW	238
3.11.1.51 DMA_REQ_TIMER0	238
3.11.1.52 DMA_REQ_TIMER1	239
3.11.1.53 DMA_REQ_TIMER2	239
3.11.1.54 DMA_REQ_TIMER3	239
3.11.1.55 DMA_REQ_UART0_RX	239
3.11.1.56 DMA_REQ_UART0_TX	239

3.11.1.57 DMA_REQ_UART1_RX	239
3.11.1.58 DMA_REQ_UART1_TX	240
3.11.1.59 DMA_TCIE	240
3.11.1.60 DMA_TEIE	240
3.11.1.61 DMA_WORD_TRANS	240
3.11.2 函数说明	240
3.11.2.1 DMA_CHx_EN()	240
3.11.2.2 DMA_ClearIRQFlag()	241
3.11.2.3 DMA_Disable()	241
3.11.2.4 DMA_Enable()	241
3.11.2.5 DMA_GetFlag()	241
3.11.2.6 DMA_GetIRQFlag()	242
3.11.2.7 DMA_Init()	243
3.11.2.8 DMA_Mem2Reg()	243
3.11.2.9 DMA_Memcpy()	244
3.11.2.10 DMA_Reset()	246
3.11.2.11 DMA_SetAddr()	246
3.11.2.12 DMA_SetLength()	246
3.11.2.13 DMA_SetRequest()	247
3.11.2.14 DMA_StructInit()	247
3.11.2.15 DMA_Trigger()	248
3.12 lks32mc09x_dma.h	248
3.13 include/lks32mc09x_dsp.h 文件参考	250
3.13.1 函数说明	251
3.13.1.1 DSP_Disable()	251
3.13.1.2 DSP_Div()	251
3.13.1.3 DSP_Enable()	252
3.13.1.4 DSP_Init()	252
3.13.1.5 DSP_Mod()	253
3.13.1.6 DSP_Reset()	253
3.13.1.7 DSP_Rms()	254
3.13.1.8 DSP_Sqrt()	254
3.14 lks32mc09x_dsp.h	254
3.15 include/lks32mc09x_eeprom.h 文件参考	255
3.15.1 宏定义说明	256
3.15.1.1 EEPROM_ADDR_LEN_16	256
3.15.1.2 EEPROM_ADDR_LEN_8	257
3.15.1.3 EEPROM_BAUD_100K	257
3.15.1.4 EEPROM_BAUD_1M	257
3.15.1.5 EEPROM_BAUD_1M2	257
3.15.1.6 EEPROM_BAUD_400K	257
3.15.2 函数说明	258

3.15.2.1 EEPROM_Disable()	258
3.15.2.2 EEPROM_Enable()	258
3.15.2.3 EEPROM_Init()	259
3.15.2.4 EEPROM_InitDefault()	259
3.15.2.5 EEPROM_Read()	259
3.15.2.6 EEPROM_ReadData()	260
3.15.2.7 EEPROM_Reset()	261
3.15.2.8 EEPROM_StructInit()	261
3.15.2.9 EEPROM_Write()	261
3.15.2.10 EEPROM_WriteData()	262
3.16 lks32mc09x_eeprom.h	263
3.17 include/lks32mc09x_exti.h 文件参考	263
3.17.1 宏定义说明	265
3.17.1.1 EXTI_CLKO_0	265
3.17.1.2 EXTI_CLKO_ADC	265
3.17.1.3 EXTI_CLKO_HSE	265
3.17.1.4 EXTI_CLKO_HSI	265
3.17.1.5 EXTI_CLKO_LSI	266
3.17.1.6 EXTI_CLKO_PLL	266
3.17.1.7 EXTI_CR_P0_0	266
3.17.1.8 EXTI_CR_P0_11	266
3.17.1.9 EXTI_CR_P0_12	266
3.17.1.10 EXTI_CR_P0_13	266
3.17.1.11 EXTI_CR_P0_14	266
3.17.1.12 EXTI_CR_P0_15	266
3.17.1.13 EXTI_CR_P0_2	267
3.17.1.14 EXTI_CR_P1_0	267
3.17.1.15 EXTI_CR_P2_0	267
3.17.1.16 EXTI_CR_P2_13	267
3.17.1.17 EXTI_CR_P2_3	267
3.17.1.18 EXTI_CR_P2_4	267
3.17.1.19 EXTI_CR_P2_5	267
3.17.1.20 EXTI_CR_P2_6	267
3.17.1.21 EXTI_CR_P2_7	268
3.17.1.22 EXTI_CR_P2_9	268
3.17.1.23 EXTI_IF_P0_0	268
3.17.1.24 EXTI_IF_P0_11	268
3.17.1.25 EXTI_IF_P0_12	268
3.17.1.26 EXTI_IF_P0_13	268
3.17.1.27 EXTI_IF_P0_14	268
3.17.1.28 EXTI_IF_P0_15	268
3.17.1.29 EXTI_IF_P0_2	269

3.17.1.30 EXTI_IF_P1_0	269
3.17.1.31 EXTI_IF_P2_0	269
3.17.1.32 EXTI_IF_P2_13	269
3.17.1.33 EXTI_IF_P2_3	269
3.17.1.34 EXTI_IF_P2_4	269
3.17.1.35 EXTI_IF_P2_5	269
3.17.1.36 EXTI_IF_P2_6	269
3.17.1.37 EXTI_IF_P2_7	270
3.17.1.38 EXTI_IF_P2_9	270
3.17.2 枚举类型说明	270
3.17.2.1 EXTI_Mode_TypeDef	270
3.17.3 函数说明	270
3.17.3.1 EXTI_ClearIRQFlag()	270
3.17.3.2 EXTI_DisableIRQFlag()	271
3.17.3.3 EXTI_EnableIRQFlag()	271
3.17.3.4 EXTI_GetIRQFlag()	272
3.17.3.5 EXTI_Reset()	272
3.17.3.6 EXTI_SetClko()	272
3.17.3.7 EXTI_TriggerConfig()	273
3.18 lks32mc09x_exti.h	273
3.19 include/lks32mc09x_flash.h 文件参考	274
3.19.1 宏定义说明	280
3.19.1.1 FLASH_SECTOR	280
3.19.1.2 FLASH_SECTOR_0	280
3.19.1.3 FLASH_SECTOR_1	280
3.19.1.4 FLASH_SECTOR_10	280
3.19.1.5 FLASH_SECTOR_100	281
3.19.1.6 FLASH_SECTOR_101	281
3.19.1.7 FLASH_SECTOR_102	281
3.19.1.8 FLASH_SECTOR_103	281
3.19.1.9 FLASH_SECTOR_104	281
3.19.1.10 FLASH_SECTOR_105	281
3.19.1.11 FLASH_SECTOR_106	281
3.19.1.12 FLASH_SECTOR_107	281
3.19.1.13 FLASH_SECTOR_108	282
3.19.1.14 FLASH_SECTOR_109	282
3.19.1.15 FLASH_SECTOR_11	282
3.19.1.16 FLASH_SECTOR_110	282
3.19.1.17 FLASH_SECTOR_111	282
3.19.1.18 FLASH_SECTOR_112	282
3.19.1.19 FLASH_SECTOR_113	282
3.19.1.20 FLASH_SECTOR_114	282

3.19.1.21 FLASH_SECTOR_115	283
3.19.1.22 FLASH_SECTOR_116	283
3.19.1.23 FLASH_SECTOR_117	283
3.19.1.24 FLASH_SECTOR_118	283
3.19.1.25 FLASH_SECTOR_119	283
3.19.1.26 FLASH_SECTOR_12	283
3.19.1.27 FLASH_SECTOR_120	283
3.19.1.28 FLASH_SECTOR_121	283
3.19.1.29 FLASH_SECTOR_122	284
3.19.1.30 FLASH_SECTOR_123	284
3.19.1.31 FLASH_SECTOR_124	284
3.19.1.32 FLASH_SECTOR_125	284
3.19.1.33 FLASH_SECTOR_126	284
3.19.1.34 FLASH_SECTOR_127	284
3.19.1.35 FLASH_SECTOR_128	284
3.19.1.36 FLASH_SECTOR_129	284
3.19.1.37 FLASH_SECTOR_13	285
3.19.1.38 FLASH_SECTOR_130	285
3.19.1.39 FLASH_SECTOR_131	285
3.19.1.40 FLASH_SECTOR_132	285
3.19.1.41 FLASH_SECTOR_133	285
3.19.1.42 FLASH_SECTOR_134	285
3.19.1.43 FLASH_SECTOR_135	285
3.19.1.44 FLASH_SECTOR_136	285
3.19.1.45 FLASH_SECTOR_137	286
3.19.1.46 FLASH_SECTOR_138	286
3.19.1.47 FLASH_SECTOR_139	286
3.19.1.48 FLASH_SECTOR_14	286
3.19.1.49 FLASH_SECTOR_140	286
3.19.1.50 FLASH_SECTOR_141	286
3.19.1.51 FLASH_SECTOR_142	286
3.19.1.52 FLASH_SECTOR_143	286
3.19.1.53 FLASH_SECTOR_144	287
3.19.1.54 FLASH_SECTOR_145	287
3.19.1.55 FLASH_SECTOR_146	287
3.19.1.56 FLASH_SECTOR_147	287
3.19.1.57 FLASH_SECTOR_148	287
3.19.1.58 FLASH_SECTOR_149	287
3.19.1.59 FLASH_SECTOR_15	287
3.19.1.60 FLASH_SECTOR_150	287
3.19.1.61 FLASH_SECTOR_151	288
3.19.1.62 FLASH_SECTOR_152	288

3.19.1.63 FLASH_SECTOR_153	288
3.19.1.64 FLASH_SECTOR_154	288
3.19.1.65 FLASH_SECTOR_155	288
3.19.1.66 FLASH_SECTOR_156	288
3.19.1.67 FLASH_SECTOR_157	288
3.19.1.68 FLASH_SECTOR_158	288
3.19.1.69 FLASH_SECTOR_159	289
3.19.1.70 FLASH_SECTOR_160	289
3.19.1.71 FLASH_SECTOR_161	289
3.19.1.72 FLASH_SECTOR_162	289
3.19.1.73 FLASH_SECTOR_163	289
3.19.1.74 FLASH_SECTOR_164	289
3.19.1.75 FLASH_SECTOR_165	289
3.19.1.76 FLASH_SECTOR_166	290
3.19.1.77 FLASH_SECTOR_167	290
3.19.1.78 FLASH_SECTOR_168	290
3.19.1.79 FLASH_SECTOR_169	290
3.19.1.80 FLASH_SECTOR_170	290
3.19.1.81 FLASH_SECTOR_171	290
3.19.1.82 FLASH_SECTOR_172	290
3.19.1.83 FLASH_SECTOR_173	291
3.19.1.84 FLASH_SECTOR_174	291
3.19.1.85 FLASH_SECTOR_175	291
3.19.1.86 FLASH_SECTOR_176	291
3.19.1.87 FLASH_SECTOR_177	291
3.19.1.88 FLASH_SECTOR_178	291
3.19.1.89 FLASH_SECTOR_179	291
3.19.1.90 FLASH_SECTOR_180	292
3.19.1.91 FLASH_SECTOR_181	292
3.19.1.92 FLASH_SECTOR_182	292
3.19.1.93 FLASH_SECTOR_183	292
3.19.1.94 FLASH_SECTOR_184	292
3.19.1.95 FLASH_SECTOR_185	292
3.19.1.96 FLASH_SECTOR_186	292
3.19.1.97 FLASH_SECTOR_187	292
3.19.1.98 FLASH_SECTOR_188	293
3.19.1.99 FLASH_SECTOR_189	293
3.19.1.100 FLASH_SECTOR_190	293

3.19.1.105 FLASH_SECTOR_191	293
3.19.1.106 FLASH_SECTOR_192	293
3.19.1.107 FLASH_SECTOR_193	293
3.19.1.108 FLASH_SECTOR_194	293
3.19.1.109 FLASH_SECTOR_195	294
3.19.1.110 FLASH_SECTOR_196	294
3.19.1.111 FLASH_SECTOR_197	294
3.19.1.112 FLASH_SECTOR_198	294
3.19.1.113 FLASH_SECTOR_199	294
3.19.1.114 FLASH_SECTOR_2	294
3.19.1.115 FLASH_SECTOR_20	294
3.19.1.116 FLASH_SECTOR_200	294
3.19.1.117 FLASH_SECTOR_201	295
3.19.1.118 FLASH_SECTOR_202	295
3.19.1.119 FLASH_SECTOR_203	295
3.19.1.120 FLASH_SECTOR_204	295
3.19.1.121 FLASH_SECTOR_205	295
3.19.1.122 FLASH_SECTOR_206	295
3.19.1.123 FLASH_SECTOR_207	295
3.19.1.124 FLASH_SECTOR_208	295
3.19.1.125 FLASH_SECTOR_209	296
3.19.1.126 FLASH_SECTOR_21	296
3.19.1.127 FLASH_SECTOR_210	296
3.19.1.128 FLASH_SECTOR_211	296
3.19.1.129 FLASH_SECTOR_212	296
3.19.1.130 FLASH_SECTOR_213	296
3.19.1.131 FLASH_SECTOR_214	296
3.19.1.132 FLASH_SECTOR_215	296
3.19.1.133 FLASH_SECTOR_216	297
3.19.1.134 FLASH_SECTOR_217	297
3.19.1.135 FLASH_SECTOR_218	297
3.19.1.136 FLASH_SECTOR_219	297
3.19.1.137 FLASH_SECTOR_22	297
3.19.1.138 FLASH_SECTOR_220	297
3.19.1.139 FLASH_SECTOR_221	297
3.19.1.140 FLASH_SECTOR_222	297
3.19.1.141 FLASH_SECTOR_223	298
3.19.1.142 FLASH_SECTOR_224	298
3.19.1.143 FLASH_SECTOR_225	298
3.19.1.144 FLASH_SECTOR_226	298
3.19.1.145 FLASH_SECTOR_227	298
3.19.1.146 FLASH_SECTOR_228	298

3.19.1.147 FLASH_SECTOR_229	298
3.19.1.148 FLASH_SECTOR_23	298
3.19.1.149 FLASH_SECTOR_230	299
3.19.1.150 FLASH_SECTOR_231	299
3.19.1.151 FLASH_SECTOR_232	299
3.19.1.152 FLASH_SECTOR_233	299
3.19.1.153 FLASH_SECTOR_234	299
3.19.1.154 FLASH_SECTOR_235	299
3.19.1.155 FLASH_SECTOR_236	299
3.19.1.156 FLASH_SECTOR_237	299
3.19.1.157 FLASH_SECTOR_238	300
3.19.1.158 FLASH_SECTOR_239	300
3.19.1.159 FLASH_SECTOR_24	300
3.19.1.160 FLASH_SECTOR_240	300
3.19.1.161 FLASH_SECTOR_241	300
3.19.1.162 FLASH_SECTOR_242	300
3.19.1.163 FLASH_SECTOR_243	300
3.19.1.164 FLASH_SECTOR_244	300
3.19.1.165 FLASH_SECTOR_245	301
3.19.1.166 FLASH_SECTOR_246	301
3.19.1.167 FLASH_SECTOR_247	301
3.19.1.168 FLASH_SECTOR_248	301
3.19.1.169 FLASH_SECTOR_249	301
3.19.1.170 FLASH_SECTOR_25	301
3.19.1.171 FLASH_SECTOR_250	301
3.19.1.172 FLASH_SECTOR_251	301
3.19.1.173 FLASH_SECTOR_252	302
3.19.1.174 FLASH_SECTOR_253	302
3.19.1.175 FLASH_SECTOR_254	302
3.19.1.176 FLASH_SECTOR_255	302
3.19.1.177 FLASH_SECTOR_26	302
3.19.1.178 FLASH_SECTOR_27	302
3.19.1.179 FLASH_SECTOR_28	302
3.19.1.180 FLASH_SECTOR_29	302
3.19.1.181 FLASH_SECTOR_3	303
3.19.1.182 FLASH_SECTOR_30	303
3.19.1.183 FLASH_SECTOR_31	303
3.19.1.184 FLASH_SECTOR_32	303
3.19.1.185 FLASH_SECTOR_33	303
3.19.1.186 FLASH_SECTOR_34	303
3.19.1.187 FLASH_SECTOR_35	303
3.19.1.188 FLASH_SECTOR_36	303

3.19.1.189 FLASH_SECTOR_37	304
3.19.1.190 FLASH_SECTOR_38	304
3.19.1.191 FLASH_SECTOR_39	304
3.19.1.192 FLASH_SECTOR_4	304
3.19.1.193 FLASH_SECTOR_40	304
3.19.1.194 FLASH_SECTOR_41	304
3.19.1.195 FLASH_SECTOR_42	304
3.19.1.196 FLASH_SECTOR_43	304
3.19.1.197 FLASH_SECTOR_44	305
3.19.1.198 FLASH_SECTOR_45	305
3.19.1.199 FLASH_SECTOR_46	305
3.19.1.200 FLASH_SECTOR_47	305
3.19.1.201 FLASH_SECTOR_48	305
3.19.1.202 FLASH_SECTOR_49	305
3.19.1.203 FLASH_SECTOR_5	305
3.19.1.204 FLASH_SECTOR_50	305
3.19.1.205 FLASH_SECTOR_51	306
3.19.1.206 FLASH_SECTOR_52	306
3.19.1.207 FLASH_SECTOR_53	306
3.19.1.208 FLASH_SECTOR_54	306
3.19.1.209 FLASH_SECTOR_55	306
3.19.1.210 FLASH_SECTOR_56	306
3.19.1.211 FLASH_SECTOR_57	306
3.19.1.212 FLASH_SECTOR_58	306
3.19.1.213 FLASH_SECTOR_59	307
3.19.1.214 FLASH_SECTOR_6	307
3.19.1.215 FLASH_SECTOR_60	307
3.19.1.216 FLASH_SECTOR_61	307
3.19.1.217 FLASH_SECTOR_62	307
3.19.1.218 FLASH_SECTOR_63	307
3.19.1.219 FLASH_SECTOR_64	307
3.19.1.220 FLASH_SECTOR_65	307
3.19.1.221 FLASH_SECTOR_66	308
3.19.1.222 FLASH_SECTOR_67	308
3.19.1.223 FLASH_SECTOR_68	308
3.19.1.224 FLASH_SECTOR_69	308
3.19.1.225 FLASH_SECTOR_7	308
3.19.1.226 FLASH_SECTOR_70	308
3.19.1.227 FLASH_SECTOR_71	308
3.19.1.228 FLASH_SECTOR_72	308
3.19.1.229 FLASH_SECTOR_73	309
3.19.1.230 FLASH_SECTOR_74	309

3.19.1.231 FLASH_SECTOR_75	309
3.19.1.232 FLASH_SECTOR_76	309
3.19.1.233 FLASH_SECTOR_77	309
3.19.1.234 FLASH_SECTOR_78	309
3.19.1.235 FLASH_SECTOR_79	309
3.19.1.236 FLASH_SECTOR_8	309
3.19.1.237 FLASH_SECTOR_80	310
3.19.1.238 FLASH_SECTOR_81	310
3.19.1.239 FLASH_SECTOR_82	310
3.19.1.240 FLASH_SECTOR_83	310
3.19.1.241 FLASH_SECTOR_84	310
3.19.1.242 FLASH_SECTOR_85	310
3.19.1.243 FLASH_SECTOR_86	310
3.19.1.244 FLASH_SECTOR_87	310
3.19.1.245 FLASH_SECTOR_88	311
3.19.1.246 FLASH_SECTOR_89	311
3.19.1.247 FLASH_SECTOR_9	311
3.19.1.248 FLASH_SECTOR_90	311
3.19.1.249 FLASH_SECTOR_91	311
3.19.1.250 FLASH_SECTOR_92	311
3.19.1.251 FLASH_SECTOR_93	311
3.19.1.252 FLASH_SECTOR_94	311
3.19.1.253 FLASH_SECTOR_95	312
3.19.1.254 FLASH_SECTOR_96	312
3.19.1.255 FLASH_SECTOR_97	312
3.19.1.256 FLASH_SECTOR_98	312
3.19.1.257 FLASH_SECTOR_99	312
3.19.1.258 FLASH_SECTOR_END_128K	312
3.19.1.259 FLASH_SECTOR_END_32K	312
3.19.1.260 FLASH_SECTOR_END_64K	312
3.19.2 函数说明	312
3.19.2.1 FLASH_Erase()	312
3.19.2.2 FLASH_EraseNvr()	313
3.19.2.3 FLASH_Read()	313
3.19.2.4 FLASH_ReadNvr()	314
3.19.2.5 FLASH_Write()	314
3.19.2.6 FLASH_WriteNvr()	314
3.20 lks32mc09x_flash.h	315
3.21 include/lks32mc09x_gpio.h 文件参考	319
3.21.1 宏定义说明	323
3.21.1.1 AF0_GPIO	323
3.21.1.2 AF10_CAN	323

3.21.1.3 AF11_SIF	324
3.21.1.4 AF12_CL	324
3.21.1.5 AF1_SYSAF	324
3.21.1.6 AF2_HALL	324
3.21.1.7 AF3_MCPWM	324
3.21.1.8 AF4_UART	324
3.21.1.9 AF5_SPI	325
3.21.1.10 AF6_I2C	325
3.21.1.11 AF7_TIMER01	325
3.21.1.12 AF8_TIMER23	325
3.21.1.13 AF9_DEBUG	325
3.21.1.14 GPIO_AF_ADC0_TRIGGER	325
3.21.1.15 GPIO_AF_ADC1_TRIGGER	326
3.21.1.16 GPIO_AF_AFE	326
3.21.1.17 GPIO_AF_CAN	326
3.21.1.18 GPIO_AF_CL	326
3.21.1.19 GPIO_AF_CLKO	326
3.21.1.20 GPIO_AF_CLU	326
3.21.1.21 GPIO_AF_CMP_OUT	327
3.21.1.22 GPIO_AF_GPIO	327
3.21.1.23 GPIO_AF_HALL	327
3.21.1.24 GPIO_AF_I2C	327
3.21.1.25 GPIO_AF_MCPWM	327
3.21.1.26 GPIO_AF_QEP0	327
3.21.1.27 GPIO_AF_SIF	328
3.21.1.28 GPIO_AF_SPI	328
3.21.1.29 GPIO_AF_TIMER0	328
3.21.1.30 GPIO_AF_TIMER1	328
3.21.1.31 GPIO_AF_TIMER2	328
3.21.1.32 GPIO_AF_UART	328
3.21.1.33 GPIO_Pin_0	329
3.21.1.34 GPIO_Pin_1	329
3.21.1.35 GPIO_Pin_10	329
3.21.1.36 GPIO_Pin_11	329
3.21.1.37 GPIO_Pin_12	329
3.21.1.38 GPIO_Pin_13	329
3.21.1.39 GPIO_Pin_14	330
3.21.1.40 GPIO_Pin_15	330
3.21.1.41 GPIO_Pin_2	330
3.21.1.42 GPIO_Pin_3	330
3.21.1.43 GPIO_Pin_4	330
3.21.1.44 GPIO_Pin_5	330

3.21.1.45 GPIO_Pin_6	331
3.21.1.46 GPIO_Pin_7	331
3.21.1.47 GPIO_Pin_8	331
3.21.1.48 GPIO_Pin_9	331
3.21.1.49 GPIO_Pin_ALL	331
3.21.1.50 GPIO_Pin_NONE	331
3.21.1.51 GPIO_PinSource_0	332
3.21.1.52 GPIO_PinSource_1	332
3.21.1.53 GPIO_PinSource_10	332
3.21.1.54 GPIO_PinSource_11	332
3.21.1.55 GPIO_PinSource_12	332
3.21.1.56 GPIO_PinSource_13	332
3.21.1.57 GPIO_PinSource_14	333
3.21.1.58 GPIO_PinSource_15	333
3.21.1.59 GPIO_PinSource_2	333
3.21.1.60 GPIO_PinSource_3	333
3.21.1.61 GPIO_PinSource_4	333
3.21.1.62 GPIO_PinSource_5	333
3.21.1.63 GPIO_PinSource_6	334
3.21.1.64 GPIO_PinSource_7	334
3.21.1.65 GPIO_PinSource_8	334
3.21.1.66 GPIO_PinSource_9	334
3.21.2 枚举类型说明	334
3.21.2.1 GPIO_Bit_Action	334
3.21.2.2 GPIO_Mode_TypeDef	335
3.21.2.3 GPIO_PuPd_TypeDef	335
3.21.3 函数说明	335
3.21.3.1 GPIO_Config()	335
3.21.3.2 GPIO_Disable()	336
3.21.3.3 GPIO_Enable()	336
3.21.3.4 GPIO_Init()	337
3.21.3.5 GPIO_LockRegister()	337
3.21.3.6 GPIO_PinAFConfig()	338
3.21.3.7 GPIO_ReadInputData()	338
3.21.3.8 GPIO_ReadInputDataBit()	339
3.21.3.9 GPIO_ReadOutputData()	339
3.21.3.10 GPIO_ReadOutputDataBit()	339
3.21.3.11 GPIO_Reset()	340
3.21.3.12 GPIO_ResetBits()	340
3.21.3.13 GPIO_SetBits()	340
3.21.3.14 GPIO_StructInit()	341
3.21.3.15 GPIO_ToggleBits()	341

3.21.3.16 GPIO_UnlockRegister()	342
3.21.3.17 GPIO_Write()	342
3.21.3.18 GPIO_WriteBit()	342
3.22 lks32mc09x_gpio.h	343
3.23 include/lks32mc09x_hall.h 文件参考	344
3.23.1 宏定义说明	346
3.23.1.1 HALL_CAPTURE_EVENT	346
3.23.1.2 HALL_CLK_DIV1	346
3.23.1.3 HALL_CLK_DIV2	347
3.23.1.4 HALL_CLK_DIV4	347
3.23.1.5 HALL_CLK_DIV8	347
3.23.1.6 HALL_OVERFLOW_EVENT	347
3.23.2 函数说明	347
3.23.2.1 HALL_Clear_IRQ()	347
3.23.2.2 HALL_Cmd()	348
3.23.2.3 HALL_GetCaptureValue()	348
3.23.2.4 HALL_GetCount()	348
3.23.2.5 HALL_GetFilterValue()	348
3.23.2.6 HALL_GetFlag()	348
3.23.2.7 HALL_GetIRQFlag()	349
3.23.2.8 HALL_Init()	349
3.23.2.9 HALL_IsCaptureEvent()	350
3.23.2.10 HALL_IsOverFlowEvent()	350
3.23.2.11 HALL_SoftTrigger()	350
3.23.2.12 HALL_StructInit()	350
3.24 lks32mc09x_hall.h	351
3.25 include/lks32mc09x_i2c.h 文件参考	352
3.25.1 宏定义说明	354
3.25.1.1 I2C_BUS_STATE_ADDR_DATA	354
3.25.1.2 I2C_BUS_STATE_Done	354
3.25.1.3 I2C_BUS_STATE_LOST_ARB	354
3.25.1.4 I2C_BUS_STATE_RX_ACK	354
3.25.1.5 I2C_BUS_STATE_STOP_EVT	354
3.25.1.6 I2C_BUS_STATE_STT_ERR	355
3.25.1.7 I2C_IF_ADDR_CMP	355
3.25.1.8 I2C_IF_BUS_ERR	355
3.25.1.9 I2C_IF_NACK	355
3.25.1.10 I2C_IF_STOP	355
3.25.1.11 I2C_IF_TC	355
3.25.2 函数说明	355
3.25.2.1 I2c_ClearBusState()	355
3.25.2.2 I2C_ClearIRQFlag()	356

3.25.2.3 I2C_Disable()	356
3.25.2.4 I2C_Enable()	357
3.25.2.5 I2C_GetFlag()	357
3.25.2.6 I2C_GetIRQFlag()	358
3.25.2.7 I2C_Init()	358
3.25.2.8 I2c_ReadBusState()	359
3.25.2.9 I2C_ReadData()	359
3.25.2.10 I2C_SendData()	360
3.25.2.11 I2C_SetBaudRate()	360
3.25.2.12 I2C_StructInit()	361
3.25.2.13 I2C_TableAck()	361
3.25.2.14 I2C_TableIrq()	362
3.25.2.15 I2C_TableRead()	362
3.25.2.16 I2C_TableSend()	362
3.25.2.17 I2C_TableState()	363
3.26 lks32mc09x_i2c.h	363
3.27 include/lks32mc09x_iwdg.h 文件参考	364
3.27.1 宏定义说明	365
3.27.1.1 IWDG_SECOND2IWDGCNT	365
3.27.1.2 IWDG_SECOND2IWDGCNT_SAT	366
3.27.2 函数说明	366
3.27.2.1 IWDG_DISABLE()	366
3.27.2.2 IWDG_ENABLE()	366
3.27.2.3 IWDG_Feed()	366
3.27.2.4 IWDG_Init()	366
3.27.2.5 IWDG_StructInit()	367
3.27.2.6 IWDG_Update()	367
3.28 lks32mc09x_iwdg.h	367
3.29 include/lks32mc09x_lib.h 文件参考	368
3.30 lks32mc09x_lib.h	368
3.31 include/lks32mc09x_mcpwm.h 文件参考	369
3.31.1 宏定义说明	374
3.31.1.1 MCPWM_AUEN_ALL	374
3.31.1.2 MCPWM_AUEN_CNT0	375
3.31.1.3 MCPWM_AUEN_DEFAULT	375
3.31.1.4 MCPWM_AUEN_IO	375
3.31.1.5 MCPWM_AUEN_IO012	375
3.31.1.6 MCPWM_AUEN_IO3	375
3.31.1.7 MCPWM_AUEN_TH0	375
3.31.1.8 MCPWM_AUEN_TH00	375
3.31.1.9 MCPWM_AUEN_TH01	375
3.31.1.10 MCPWM_AUEN_TH10	376

3.31.1.11 MCPWM_AUEN_TH11	376
3.31.1.12 MCPWM_AUEN_TH20	376
3.31.1.13 MCPWM_AUEN_TH21	376
3.31.1.14 MCPWM_AUEN_TH30	376
3.31.1.15 MCPWM_AUEN_TH31	376
3.31.1.16 MCPWM_AUEN_TMR0	376
3.31.1.17 MCPWM_AUEN_TMR1	376
3.31.1.18 MCPWM_AUEN_TMR2	377
3.31.1.19 MCPWM_AUEN_TMR3	377
3.31.1.20 MCPWM_CLK_DIV_1	377
3.31.1.21 MCPWM_CLK_DIV_128	377
3.31.1.22 MCPWM_CLK_DIV_16	377
3.31.1.23 MCPWM_CLK_DIV_2	377
3.31.1.24 MCPWM_CLK_DIV_32	377
3.31.1.25 MCPWM_CLK_DIV_4	377
3.31.1.26 MCPWM_CLK_DIV_64	378
3.31.1.27 MCPWM_CLK_DIV_8	378
3.31.1.28 MCPWM_COUNT_EVT_TIMER0_CMP0	378
3.31.1.29 MCPWM_COUNT_EVT_TIMER0_CMP1	378
3.31.1.30 MCPWM_COUNT_EVT_TIMER1_CMP0	378
3.31.1.31 MCPWM_COUNT_EVT_TIMER1_CMP1	378
3.31.1.32 MCPWM_COUNT_EVT_TIMER2_CMP0	378
3.31.1.33 MCPWM_COUNT_EVT_TIMER2_CMP1	378
3.31.1.34 MCPWM_COUNT_OFF	379
3.31.1.35 MCPWM_COUNT_START	379
3.31.1.36 MCPWM_EIE_FAIL0	379
3.31.1.37 MCPWM_EIE_FAIL1	379
3.31.1.38 MCPWM{EIF_FAIL0	379
3.31.1.39 MCPWM{EIF_FAIL1	379
3.31.1.40 MCPWM_FAIL0_BKIN0_H	379
3.31.1.41 MCPWM_FAIL0_BKIN0_L	379
3.31.1.42 MCPWM_FAIL0_CLU_OUT0_H	380
3.31.1.43 MCPWM_FAIL0_CLU_OUT0_L	380
3.31.1.44 MCPWM_FAIL0_CLU_OUT1_H	380
3.31.1.45 MCPWM_FAIL0_CLU_OUT1_L	380
3.31.1.46 MCPWM_FAIL0_CMP0_H	380
3.31.1.47 MCPWM_FAIL0_CMP0_L	380
3.31.1.48 MCPWM_FAIL0_OFF	380
3.31.1.49 MCPWM_FAIL1_BKIN1_H	380
3.31.1.50 MCPWM_FAIL1_BKIN1_L	381
3.31.1.51 MCPWM_FAIL1_CLU_OUT2_H	381
3.31.1.52 MCPWM_FAIL1_CLU_OUT2_L	381

3.31.1.53 MCPWM_FAIL1_CLU_OUT3_H	381
3.31.1.54 MCPWM_FAIL1_CLU_OUT3_L	381
3.31.1.55 MCPWM_FAIL1_CMP1_H	381
3.31.1.56 MCPWM_FAIL1_CMP1_L	381
3.31.1.57 MCPWM_FAIL1_OFF	381
3.31.1.58 MCPWM_FAIL_IO_HIGH	382
3.31.1.59 MCPWM_FAIL_IO_LOW	382
3.31.1.60 MCPWM_FAIL_IO_OFF	382
3.31.1.61 MCPWM_GET_IRQ_FAIL0	382
3.31.1.62 MCPWM_GET_IRQ_FAIL1	382
3.31.1.63 MCPWM_GET_IRQ_T0	382
3.31.1.64 MCPWM_GET_IRQ_T1	382
3.31.1.65 MCPWM_GET_IRQ_TH00	383
3.31.1.66 MCPWM_GET_IRQ_TH01	383
3.31.1.67 MCPWM_GET_IRQ_TH10	383
3.31.1.68 MCPWM_GET_IRQ_TH11	383
3.31.1.69 MCPWM_GET_IRQ_TH20	383
3.31.1.70 MCPWM_GET_IRQ_TH21	383
3.31.1.71 MCPWM_GET_IRQ_TMR0	383
3.31.1.72 MCPWM_GET_IRQ_TMR1	384
3.31.1.73 MCPWM_GET_IRQ_TMR2	384
3.31.1.74 MCPWM_GET_IRQ_TMR3	384
3.31.1.75 MCPWM_GET_IRQ_UP	384
3.31.1.76 MCPWM_IE_T0	384
3.31.1.77 MCPWM_IE_T1	384
3.31.1.78 MCPWM_IE_TH00	384
3.31.1.79 MCPWM_IE_TH01	385
3.31.1.80 MCPWM_IE_TH10	385
3.31.1.81 MCPWM_IE_TH11	385
3.31.1.82 MCPWM_IE_TH20	385
3.31.1.83 MCPWM_IE_TH21	385
3.31.1.84 MCPWM_IE_TMR0	385
3.31.1.85 MCPWM_IE_TMR1	385
3.31.1.86 MCPWM_IE_TMR2	385
3.31.1.87 MCPWM_IE_TMR3	386
3.31.1.88 MCPWM_IE_UP	386
3.31.1.89 MCPWM_IF_T0	386
3.31.1.90 MCPWM_IF_T1	386
3.31.1.91 MCPWM_IF_TH00	386
3.31.1.92 MCPWM_IF_TH01	386
3.31.1.93 MCPWM_IF_TH10	386
3.31.1.94 MCPWM_IF_TH11	386

3.31.1.95 MCPWM_IF_TH20	387
3.31.1.96 MCPWM_IF_TH21	387
3.31.1.97 MCPWM_IF_TMR0	387
3.31.1.98 MCPWM_IF_TMR1	387
3.31.1.99 MCPWM_IF_TMR2	387
3.31.1.100 MCPWM_IF_TMR3	387
3.31.1.101 MCPWM_IF_UP	387
3.31.1.102 MCPWM_IO_DISABLE	387
3.31.1.103 MCPWM_IO_HH_HIGH	388
3.31.1.104 MCPWM_IO_HH_HPWM	388
3.31.1.105 MCPWM_IO_HH_LOW	388
3.31.1.106 MCPWM_IO_HH_LPWM	388
3.31.1.107 MCPWM_IO_HH_OFF	388
3.31.1.108 MCPWM_IO_HH_PWM	388
3.31.1.109 MCPWM_IO_HHS_HIGH	388
3.31.1.110 MCPWM_IO_HHS_HPWM	388
3.31.1.111 MCPWM_IO_HHS_LOW	389
3.31.1.112 MCPWM_IO_HHS_LPWM	389
3.31.1.113 MCPWM_IO_HHS_OFF	389
3.31.1.114 MCPWM_IO_HHS_PWM	389
3.31.1.115 MCPWM_IO_HL_HIGH	389
3.31.1.116 MCPWM_IO_HL_HPWM	389
3.31.1.117 MCPWM_IO_HL_LOW	389
3.31.1.118 MCPWM_IO_HL_LPWM	389
3.31.1.119 MCPWM_IO_HL_OFF	390
3.31.1.120 MCPWM_IO_HL_PWM	390
3.31.1.121 MCPWM_IO_HLS_HIGH	390
3.31.1.122 MCPWM_IO_HLS_HPWM	390
3.31.1.123 MCPWM_IO_HLS_LOW	390
3.31.1.124 MCPWM_IO_HLS_LPWM	390
3.31.1.125 MCPWM_IO_HLS_OFF	390
3.31.1.126 MCPWM_IO_HLS_PWM	390
3.31.1.127 MCPWM_IO_LH_HIGH	391
3.31.1.128 MCPWM_IO_LH_HPWM	391
3.31.1.129 MCPWM_IO_LH_LOW	391
3.31.1.130 MCPWM_IO_LH_LPWM	391
3.31.1.131 MCPWM_IO_LH_OFF	391
3.31.1.132 MCPWM_IO_LH_PWM	391
3.31.1.133 MCPWM_IO_LHS_HIGH	391
3.31.1.134 MCPWM_IO_LHS_HPWM	391
3.31.1.135 MCPWM_IO_LHS_LOW	392
3.31.1.136 MCPWM_IO_LHS_LPWM	392

3.31.1.137 MCPWM_IO_LHS_OFF	392
3.31.1.138 MCPWM_IO_LHS_PWM	392
3.31.1.139 MCPWM_IO_LL_HIGH	392
3.31.1.140 MCPWM_IO_LL_HPWM	392
3.31.1.141 MCPWM_IO_LL_LOW	392
3.31.1.142 MCPWM_IO_LL_LPWM	392
3.31.1.143 MCPWM_IO_LL_OFF	393
3.31.1.144 MCPWM_IO_LL_PWM	393
3.31.1.145 MCPWM_IO_LLS_HIGH	393
3.31.1.146 MCPWM_IO_LLS_HPWM	393
3.31.1.147 MCPWM_IO_LLS_LOW	393
3.31.1.148 MCPWM_IO_LLS_LPWM	393
3.31.1.149 MCPWM_IO_LLS_OFF	393
3.31.1.150 MCPWM_IO_LLS_PWM	393
3.31.1.151 MCPWM_IO_NP	394
3.31.1.152 MCPWM_IO_NS	394
3.31.1.153 MCPWM_IO_PN_SW	394
3.31.1.154 MCPWM_IO_PP	394
3.31.1.155 MCPWM_IO_PS	394
3.31.1.156 MCPWM_IO_SCTRLN	394
3.31.1.157 MCPWM_IO_SCTRLP	394
3.31.1.158 MCPWM_IO_WM	394
3.31.1.159 MCPWM_IRQ_CLEAR_FAIL0	395
3.31.1.160 MCPWM_IRQ_CLEAR_FAIL1	395
3.31.1.161 MCPWM_IRQ_CLEAR_T0	395
3.31.1.162 MCPWM_IRQ_CLEAR_T1	395
3.31.1.163 MCPWM_IRQ_CLEAR_TH00	395
3.31.1.164 MCPWM_IRQ_CLEAR_TH01	395
3.31.1.165 MCPWM_IRQ_CLEAR_TH10	395
3.31.1.166 MCPWM_IRQ_CLEAR_TH11	396
3.31.1.167 MCPWM_IRQ_CLEAR_TH20	396
3.31.1.168 MCPWM_IRQ_CLEAR_TH21	396
3.31.1.169 MCPWM_IRQ_CLEAR_TMR0	396
3.31.1.170 MCPWM_IRQ_CLEAR_TMR1	396
3.31.1.171 MCPWM_IRQ_CLEAR_TMR2	396
3.31.1.172 MCPWM_IRQ_CLEAR_TMR3	396
3.31.1.173 MCPWM_IRQ_CLEAR_UP	397
3.31.1.174 MCPWM_IRQ_FAIL0	397
3.31.1.175 MCPWM_IRQ_FAIL1	397
3.31.1.176 MCPWM_IRQ_T0	397
3.31.1.177 MCPWM_IRQ_T1	397
3.31.1.178 MCPWM_IRQ_TH00	397

3.31.1.179 MCPWM_IRQ_TH01	397
3.31.1.180 MCPWM_IRQ_TH10	398
3.31.1.181 MCPWM_IRQ_TH11	398
3.31.1.182 MCPWM_IRQ_TH20	398
3.31.1.183 MCPWM_IRQ_TH21	398
3.31.1.184 MCPWM_IRQ_TMR0	398
3.31.1.185 MCPWM_IRQ_TMR1	398
3.31.1.186 MCPWM_IRQ_TMR2	398
3.31.1.187 MCPWM_IRQ_TMR3	399
3.31.1.188 MCPWM_IRQ_UP	399
3.31.1.189 MCPWM_TR_T0	399
3.31.1.190 MCPWM_TR_T1	399
3.31.2 函数说明	399
3.31.2.1 MCPWM_ClearErrorIRQFlag()	399
3.31.2.2 MCPWM_ClearIRQFlag()	399
3.31.2.3 MCPWM_Disable()	400
3.31.2.4 MCPWM_Enable()	401
3.31.2.5 MCPWM_GetCountTh()	401
3.31.2.6 MCPWM_GetCountVal()	402
3.31.2.7 MCPWM_GetEifFlag()	402
3.31.2.8 MCPWM_GetErrorIRQFlag()	402
3.31.2.9 MCPWM_GetIfFlag()	403
3.31.2.10 MCPWM_GetIRQFlag()	403
3.31.2.11 MCPWM_Init()	404
3.31.2.12 MCPWM_Reset()	404
3.31.2.13 MCPWM_SetIOConfig()	405
3.31.2.14 MCPWM_SetOutputState()	406
3.31.2.15 MCPWM_SetOutputVal()	406
3.31.2.16 MCPWM_SetTMR()	406
3.31.2.17 MCPWM_StartCount()	407
3.31.2.18 MCPWM_StopCount()	407
3.31.2.19 MCPWM_StructInit()	407
3.32 lks32mc09x_mcpwm.h	408
3.33 include/lks32mc09x_opa.h 文件参考	412
3.33.1 宏定义说明	414
3.33.1.1 OPA_CHANNEL_0	414
3.33.1.2 OPA_CHANNEL_1	414
3.33.1.3 OPA_CHANNEL_2	415
3.33.1.4 OPA_CHANNEL_3	415
3.33.1.5 OPA_GAIN_16	415
3.33.1.6 OPA_GAIN_33	415
3.33.1.7 OPA_GAIN_4	415

3.33.1.8 OPA_GAIN_8	415
3.33.1.9 OPA_IT_1_0	416
3.33.1.10 OPA_IT_1_2	416
3.33.1.11 OPA_IT_1_5	416
3.33.1.12 OPA_IT_2_0	416
3.33.1.13 OPA_OUT_CMP_OUT0N	416
3.33.1.14 OPA_OUT_CMP_OUT0P	416
3.33.1.15 OPA_OUT_CMP_OUT1N	417
3.33.1.16 OPA_OUT_CMP_OUT1P	417
3.33.1.17 OPA_OUT_CMP_OUT2N	417
3.33.1.18 OPA_OUT_CMP_OUT2P	417
3.33.1.19 OPA_OUT_CMP_OUT3N	417
3.33.1.20 OPA_OUT_CMP_OUT3P	417
3.33.2 函数说明	417
3.33.2.1 OPA_Disable()	417
3.33.2.2 OPA_Enable()	418
3.33.2.3 OPA_Init()	418
3.33.2.4 OPA_OutCmpSignal()	419
3.33.2.5 OPA_SetGain()	419
3.33.2.6 OPA_SetIt()	421
3.33.2.7 OPA_StructInit()	422
3.34 lks32mc09x_opa.h	422
3.35 include/lks32mc09x_qep.h 文件参考	423
3.35.1 宏定义说明	425
3.35.1.1 QEP_CLK_DIV1	425
3.35.1.2 QEP_CLK_DIV128	425
3.35.1.3 QEP_CLK_DIV16	425
3.35.1.4 QEP_CLK_DIV2	426
3.35.1.5 QEP_CLK_DIV32	426
3.35.1.6 QEP_CLK_DIV4	426
3.35.1.7 QEP_CLK_DIV64	426
3.35.1.8 QEP_CLK_DIV8	426
3.35.1.9 QEP_IF_OVERFLOW	426
3.35.1.10 QEP_IF_UNDERFLOW	427
3.35.1.11 QEP_IRQEna_None	427
3.35.1.12 QEP_IRQEna_OverFlow	427
3.35.1.13 QEP_IRQEna_UnderFlow	427
3.35.1.14 QEP_Mode_CCWCW_UpDown	427
3.35.1.15 QEP_Mode_CCWSIGN_UpDown	427
3.35.1.16 QEP_Mode_T1	428
3.35.1.17 QEP_Mode_T1_T2	428
3.35.1.18 QEP_ZC_EDEG_ALL	428

3.35.1.19 QEP_ZC_EDEG_N	428
3.35.1.20 QEP_ZC_EDEG_P	428
3.35.1.21 QEP_ZC_LEVEL_H	428
3.35.1.22 QEP_ZC_LEVEL_L	429
3.35.1.23 QEP_ZC_NONE	429
3.35.2 函数说明	429
3.35.2.1 QEP_ClearIRQFlag()	429
3.35.2.2 QEP_Cmd()	429
3.35.2.3 QEP_GetCount()	430
3.35.2.4 QEP_GetFlag()	430
3.35.2.5 QEP_GetIRQFlag()	431
3.35.2.6 QEP_Init()	431
3.35.2.7 QEP_SetMode()	431
3.35.2.8 QEP_StructInit()	432
3.36 lks32mc09x_qep.h	432
3.37 include/lks32mc09x_reg.h 文件参考	433
3.37.1 宏定义说明	436
3.37.1.1 ADC	436
3.37.1.2 ADC0	436
3.37.1.3 ADC1	436
3.37.1.4 AON	436
3.37.1.5 AON0	436
3.37.1.6 CL0	436
3.37.1.7 CLU	437
3.37.1.8 CLU0	437
3.37.1.9 CMP	437
3.37.1.10 CMP0	437
3.37.1.11 CRC0	437
3.37.1.12 DMA0	437
3.37.1.13 DMA1	437
3.37.1.14 DMA2	437
3.37.1.15 DMA3	438
3.37.1.16 DMA_CHN0	438
3.37.1.17 DMA_CHN1	438
3.37.1.18 DMA_CHN2	438
3.37.1.19 DMA_CHN3	438
3.37.1.20 DMA_TOP	438
3.37.1.21 DMAREG	438
3.37.1.22 DSP	438
3.37.1.23 DSP0	439
3.37.1.24 DSP_REG	439
3.37.1.25 EXTI	439

3.37.1.26 EXTI0	439
3.37.1.27 FLASH	439
3.37.1.28 FLASH0	439
3.37.1.29 GPIO0	439
3.37.1.30 GPIO1	439
3.37.1.31 GPIO2	440
3.37.1.32 HALL	440
3.37.1.33 HALL0	440
3.37.1.34 I2C	440
3.37.1.35 I2C0	440
3.37.1.36 IWDG	440
3.37.1.37 IWDG0	440
3.37.1.38 MCPWM	440
3.37.1.39 MCPWM0	441
3.37.1.40 QEP	441
3.37.1.41 QEP0	441
3.37.1.42 REG_IF_WRITE	441
3.37.1.43 REG_READ	441
3.37.1.44 REG_READ_BIT	441
3.37.1.45 REG_READ_BIT_POS_MASK	441
3.37.1.46 REG_RESET	442
3.37.1.47 REG_SET	442
3.37.1.48 REG_WRITE	442
3.37.1.49 REG_WRITE_BIT	442
3.37.1.50 REG_WRITE_BIT_POS_MASK	442
3.37.1.51 SIF	442
3.37.1.52 SIF0	443
3.37.1.53 SPI	443
3.37.1.54 SPI0	443
3.37.1.55 SYS	443
3.37.1.56 SYS0	443
3.37.1.57 TIMER0	443
3.37.1.58 TIMER1	443
3.37.1.59 TIMER16_0	443
3.37.1.60 TIMER16_1	444
3.37.1.61 TIMER2	444
3.37.1.62 TIMER32_0	444
3.37.1.63 UART	444
3.37.1.64 UART0	444
3.37.1.65 UART1	444
3.37.1.66 UART2	444
3.37.1.67 UTIMER0	444

3.37.1.68 UTIMER1	445
3.37.1.69 UTIMER16_0	445
3.37.1.70 UTIMER16_1	445
3.37.1.71 UTIMER2	445
3.37.1.72 UTIMER32_0	445
3.38 lks32mc09x_reg.h	445
3.39 include/lks32mc09x_sif.h 文件参考	450
3.39.1 宏定义说明	452
3.39.1.1 SIF_DMA_Enable	452
3.39.1.2 SIF_IE_Enable	452
3.39.1.3 SIF_IF_CLEAR	452
3.39.2 函数说明	452
3.39.2.1 SIF_Clear_Irq()	452
3.39.2.2 SIF_DMA_IsEnable()	452
3.39.2.3 SIF_Init()	453
3.39.2.4 SIF_Irq_IsEnable()	453
3.39.2.5 SIF_Senddata()	454
3.39.2.6 SIF_StructInit()	454
3.39.2.7 SIF_TH_set()	454
3.39.2.8 SIF_Timebase_set()	454
3.40 lks32mc09x_sif.h	455
3.41 include/lks32mc09x_spi.h 文件参考	456
3.41.1 宏定义说明	457
3.41.1.1 SPI_DMA_DISABLE	457
3.41.1.2 SPI_DMA_ENABLE	458
3.41.1.3 SPI_FIRSTSEND_LSB	458
3.41.1.4 SPI_FIRSTSEND_MSB	458
3.41.1.5 SPI_IF_DataOver	458
3.41.1.6 SPI_IF_SSErr	458
3.41.1.7 SPI_IF_TranDone	458
3.41.1.8 SPI_IRQEna_DataOver	459
3.41.1.9 SPI_IRQEna_Enable	459
3.41.1.10 SPI_IRQEna_SSErr	459
3.41.1.11 SPI_IRQEna_TranDone	459
3.41.2 枚举类型说明	459
3.41.2.1 SPI_Duplex	459
3.41.2.2 SPI_Mode	460
3.41.3 函数说明	460
3.41.3.1 SPI_ClearIRQFlag()	460
3.41.3.2 SPI_GetIRQFlag()	460
3.41.3.3 SPI_Init()	461
3.41.3.4 SPI_ReadData()	461

3.41.3.5 SPI_SendData()	462
3.41.3.6 SPI_StructInit()	462
3.42 lks32mc09x_spi.h	462
3.43 include/lks32mc09x_sys.h 文件参考	463
3.43.1 宏定义说明	465
3.43.1.1 SYS_MCLK_RC_12M	465
3.43.1.2 SYS_MCLK_RC_24M	465
3.43.1.3 SYS_MCLK_RC_48M	465
3.43.1.4 SYS_MCLK_RC_8M	465
3.43.1.5 SYS_MCLK_RC_96M	465
3.43.1.6 SYS_MCLK_XTAL_12M	466
3.43.1.7 SYS_MCLK_XTAL_24M	466
3.43.1.8 SYS_MCLK_XTAL_48M	466
3.43.1.9 SYS_MCLK_XTAL_96M	466
3.43.1.10 SYS_MODULE_ADC0	466
3.43.1.11 SYS_MODULE_ADC1	466
3.43.1.12 SYS_MODULE_CL0	466
3.43.1.13 SYS_MODULE_CMP	466
3.43.1.14 SYS_MODULE_CRC0	467
3.43.1.15 SYS_MODULE_DMA0	467
3.43.1.16 SYS_MODULE_DSP0	467
3.43.1.17 SYS_MODULE_EEPROM	467
3.43.1.18 SYS_MODULE_GPIO	467
3.43.1.19 SYS_MODULE_HALL0	467
3.43.1.20 SYS_MODULE_I2C0	467
3.43.1.21 SYS_MODULE_MCPWM0	467
3.43.1.22 SYS_MODULE_QEP0	468
3.43.1.23 SYS_MODULE_SIF0	468
3.43.1.24 SYS_MODULE_SPI0	468
3.43.1.25 SYS_MODULE_TIMER0	468
3.43.1.26 SYS_MODULE_TIMER1	468
3.43.1.27 SYS_MODULE_TIMER2	468
3.43.1.28 SYS_MODULE_UART0	468
3.43.1.29 SYS_MODULE_UART1	468
3.43.1.30 SYS_MODULE_UART2	469
3.43.1.31 SYS_PVDSEL_2.8V	469
3.43.1.32 SYS_PVDSEL_3.49V	469
3.43.1.33 SYS_PVDSEL_3.5V	469
3.43.1.34 SYS_PVDSEL_3.74V	469
3.43.1.35 SYS_PVDSEL_3.75V	469
3.43.1.36 SYS_PVDSEL_4.0V	469
3.43.1.37 SYS_PVDSEL_DAC	469

3.43.1.38 SYS_PVDSEL_OFF	470
3.43.2 函数说明	470
3.43.2.1 SYS_I2CClkDiv()	470
3.43.2.2 SYS_MclkChoice()	470
3.43.2.3 SYS_ModuleClockCmd()	471
3.43.2.4 SYS_ReadMcuClk()	472
3.43.2.5 SYS_ResetIO()	473
3.43.2.6 SYS_ResetPeripheral()	473
3.43.2.7 SYS_ResetPeripheralInternal()	474
3.43.2.8 SYS_SetPowerFailureDetection()	474
3.43.2.9 SYS_SoftReset_all()	475
3.43.2.10 SYS_SoftResetModule()	475
3.43.2.11 SYS_SPIClkDiv()	476
3.43.2.12 SYS_UartClkDiv()	476
3.43.2.13 SYS_WaitPowerReady()	478
3.44 lks32mc09x_sys.h	479
3.45 include/lks32mc09x_timer.h 文件参考	481
3.45.1 宏定义说明	482
3.45.1.1 TIMER_CH.MODE_CAP	482
3.45.1.2 TIMER_CH.MODE_Capture	482
3.45.1.3 TIMER_CH.MODE_CMP	483
3.45.1.4 TIMER_CH.MODE_Comparison	483
3.45.1.5 TIMER_IF_CH0	483
3.45.1.6 TIMER_IF_CH1	483
3.45.1.7 TIMER_IF_FAIL	483
3.45.1.8 TIMER_IF_ZC	483
3.45.1.9 TIMER_StatisticsStart	483
3.45.1.10 TIMER_StatisticsStop	484
3.45.2 函数说明	484
3.45.2.1 TIMER_ClearIRQFlag()	484
3.45.2.2 TIMER_CMP0()	484
3.45.2.3 TIMER_CMP1()	484
3.45.2.4 TIMER_Disable()	485
3.45.2.5 TIMER_Enable()	485
3.45.2.6 TIMER_GetFlag()	486
3.45.2.7 TIMER_GetIRQFlag()	487
3.45.2.8 TIMER_Init()	487
3.45.2.9 TIMER_SetOutputState()	487
3.45.2.10 TIMER_StructInit()	488
3.45.2.11 TIMER_Timer_GetCMPT0()	488
3.45.2.12 TIMER_Timer_GetCMPT1()	488
3.45.2.13 TIMER_Timer_GetCount()	489

3.45.2.14 TIMER_TimerCmd()	489
3.46 lks32mc09x_timer.h	489
3.47 include/lks32mc09x_tmp.h 文件参考	491
3.47.1 函数说明	492
3.47.1.1 TMP_Disable()	492
3.47.1.2 TMP_Enable()	492
3.47.1.3 TMP_GetCurrentTemperature()	492
3.47.1.4 TMP_Init()	493
3.47.1.5 TMP_Reset()	493
3.48 lks32mc09x_tmp.h	494
3.49 include/lks32mc09x_trim.h 文件参考	494
3.49.1 宏定义说明	496
3.49.1.1 NVR_ADDR_ADC_0_AMC0	496
3.49.1.2 NVR_ADDR_ADC_0_AMC1	496
3.49.1.3 NVR_ADDR_ADC_0_DC0	496
3.49.1.4 NVR_ADDR_ADC_0_DC1	496
3.49.1.5 NVR_ADDR_ADC_1_AMC0	496
3.49.1.6 NVR_ADDR_ADC_1_AMC1	496
3.49.1.7 NVR_ADDR_ADC_1_DC0	496
3.49.1.8 NVR_ADDR_ADC_1_DC1	497
3.49.1.9 NVR_ADDR_AFE_REG8	497
3.49.1.10 NVR_ADDR_AFE_REG9	497
3.49.1.11 NVR_ADDR_AFE_REGA	497
3.49.1.12 NVR_ADDR_AFE_REGB	497
3.49.1.13 NVR_ADDR_AFE_REGC	497
3.49.1.14 NVR_ADDR_AMTE_CNT	497
3.49.1.15 NVR_ADDR_AMTE_TMP	497
3.49.1.16 NVR_ADDR_AMTE_VERSION	498
3.49.1.17 NVR_ADDR_CHIP_BOOT_VERSION	498
3.49.1.18 NVR_ADDR_CHIP_FLASH_SIZE	498
3.49.1.19 NVR_ADDR_CHIP_ID0	498
3.49.1.20 NVR_ADDR_CHIP_ID1	498
3.49.1.21 NVR_ADDR_CHIP_ID2	498
3.49.1.22 NVR_ADDR_CHIP_ID3	498
3.49.1.23 NVR_ADDR_CHIP_ID4	498
3.49.1.24 NVR_ADDR_CHIP_MODEL	499
3.49.1.25 NVR_ADDR_DAC_0_AMC_1_20V	499
3.49.1.26 NVR_ADDR_DAC_0_AMC_4.8V	499
3.49.1.27 NVR_ADDR_DAC_0_DC_1_20V	499
3.49.1.28 NVR_ADDR_DAC_0_DC_4.8V	499
3.49.1.29 NVR_ADDR_DAC_1_AMC_1_20V	499
3.49.1.30 NVR_ADDR_DAC_1_AMC_3.0V	499

3.49.1.31 NVR_ADDR_DAC_1_DC_1_20V	499
3.49.1.32 NVR_ADDR_DAC_1_DC_3_0V	500
3.49.1.33 NVR_ADDR_OPA_0	500
3.49.1.34 NVR_ADDR_OPA_0_160K_40K_GAIN	500
3.49.1.35 NVR_ADDR_OPA_0_160K_40K_RES	500
3.49.1.36 NVR_ADDR_OPA_0_320K_40K_GAIN	500
3.49.1.37 NVR_ADDR_OPA_0_320K_40K_RES	500
3.49.1.38 NVR_ADDR_OPA_0_640K_40K_GAIN	500
3.49.1.39 NVR_ADDR_OPA_0_640K_40K_RES	500
3.49.1.40 NVR_ADDR_OPA_0_660K_20K_GAIN	501
3.49.1.41 NVR_ADDR_OPA_0_660K_20K_RES	501
3.49.1.42 NVR_ADDR_OPA_0_OFFSET_H	501
3.49.1.43 NVR_ADDR_OPA_0_OFFSET_L	501
3.49.1.44 NVR_ADDR_OPA_1	501
3.49.1.45 NVR_ADDR_OPA_1_160K_40K_GAIN	501
3.49.1.46 NVR_ADDR_OPA_1_160K_40K_RES	501
3.49.1.47 NVR_ADDR_OPA_1_320K_40K_GAIN	501
3.49.1.48 NVR_ADDR_OPA_1_320K_40K_RES	502
3.49.1.49 NVR_ADDR_OPA_1_640K_40K_GAIN	502
3.49.1.50 NVR_ADDR_OPA_1_640K_40K_RES	502
3.49.1.51 NVR_ADDR_OPA_1_660K_20K_GAIN	502
3.49.1.52 NVR_ADDR_OPA_1_660K_20K_RES	502
3.49.1.53 NVR_ADDR_OPA_1_OFFSET_H	502
3.49.1.54 NVR_ADDR_OPA_1_OFFSET_L	502
3.49.1.55 NVR_ADDR_OPA_2	502
3.49.1.56 NVR_ADDR_OPA_2_160K_40K_GAIN	503
3.49.1.57 NVR_ADDR_OPA_2_160K_40K_RES	503
3.49.1.58 NVR_ADDR_OPA_2_320K_40K_GAIN	503
3.49.1.59 NVR_ADDR_OPA_2_320K_40K_RES	503
3.49.1.60 NVR_ADDR_OPA_2_640K_40K_GAIN	503
3.49.1.61 NVR_ADDR_OPA_2_640K_40K_RES	503
3.49.1.62 NVR_ADDR_OPA_2_660K_20K_GAIN	503
3.49.1.63 NVR_ADDR_OPA_2_660K_20K_RES	503
3.49.1.64 NVR_ADDR_OPA_2_OFFSET_H	504
3.49.1.65 NVR_ADDR_OPA_2_OFFSET_L	504
3.49.1.66 NVR_ADDR_OPA_3	504
3.49.1.67 NVR_ADDR_OPA_3_160K_40K_GAIN	504
3.49.1.68 NVR_ADDR_OPA_3_160K_40K_RES	504
3.49.1.69 NVR_ADDR_OPA_3_320K_40K_GAIN	504
3.49.1.70 NVR_ADDR_OPA_3_320K_40K_RES	504
3.49.1.71 NVR_ADDR_OPA_3_640K_40K_GAIN	504
3.49.1.72 NVR_ADDR_OPA_3_640K_40K_RES	505

3.49.1.73 NVR_ADDR_OPA_3_660K_20K_GAIN	505
3.49.1.74 NVR_ADDR_OPA_3_660K_20K_RES	505
3.49.1.75 NVR_ADDR_OPA_3_OFFSET_H	505
3.49.1.76 NVR_ADDR_OPA_3_OFFSET_L	505
3.49.1.77 NVR_ADDR_TMP_B	505
3.49.1.78 NVR_ADDR_TMP_K	505
3.49.2 函数说明	506
3.49.2.1 TRIM_Read()	506
3.50 lks32mc09x_trim.h	506
3.51 include/lks32mc09x_uart.h 文件参考	508
3.51.1 宏定义说明	510
3.51.1.1 RX_DMA_RE	510
3.51.1.2 TX_BUF_DMA_RE	511
3.51.1.3 TX_DMA_RE	511
3.51.1.4 UART_FIRSTSEND_LSB	511
3.51.1.5 UART_FIRSTSEND_MSB	511
3.51.1.6 UART_IF_ABD	511
3.51.1.7 UART_IF_CheckError	511
3.51.1.8 UART_IF_IDLE	512
3.51.1.9 UART_IF_LBD	512
3.51.1.10 UART_IF_RcvOver	512
3.51.1.11 UART_IF_RX_OV	512
3.51.1.12 UART_IF_SendBufEmpty	512
3.51.1.13 UART_IF_SendOver	512
3.51.1.14 UART_IF_StopError	513
3.51.1.15 UART_IF_TX_OV	513
3.51.1.16 UART_IRQEna_ABD	513
3.51.1.17 UART_IRQEna_CheckError	513
3.51.1.18 UART_IRQEna_IDLE	513
3.51.1.19 UART_IRQEna_LBD	513
3.51.1.20 UART_IRQEna_RcvOver	514
3.51.1.21 UART_IRQEna_RX_OV	514
3.51.1.22 UART_IRQEna_SendBuffEmpty	514
3.51.1.23 UART_IRQEna_SendOver	514
3.51.1.24 UART_IRQEna_StopError	514
3.51.1.25 UART_IRQEna_TX_OV	514
3.51.1.26 UART_STOPBITS_1b	515
3.51.1.27 UART_STOPBITS_2b	515
3.51.1.28 UART_WORDLENGTH_8b	515
3.51.1.29 UART_WORDLENGTH_9b	515
3.51.2 枚举类型说明	515
3.51.2.1 UART_ParityMode	515

3.51.3 函数说明	516
3.51.3.1 UART_ClearIRQFlag()	516
3.51.3.2 UART_GetIRQFlag()	516
3.51.3.3 UART_Init()	516
3.51.3.4 UART_ReadData()	517
3.51.3.5 UART_RecvnData()	518
3.51.3.6 UART_SendAddr_485()	518
3.51.3.7 UART_SendData()	519
3.51.3.8 UART_SendData_485()	520
3.51.3.9 UART_SendnData()	521
3.51.3.10 UART_SetBaud()	521
3.51.3.11 UART_SetClkDiv()	522
3.51.3.12 UART_StructInit()	523
3.52 lks32mc09x_uart.h	523
3.53 include/lks32mc09x_wake.h 文件参考	525
3.53.1 宏定义说明	527
3.53.1.1 SYS_WakeInterval_025s	527
3.53.1.2 SYS_WakeInterval_05s	527
3.53.1.3 SYS_WakeInterval_16s	527
3.53.1.4 SYS_WakeInterval_1s	527
3.53.1.5 SYS_WakeInterval_2s	527
3.53.1.6 SYS_WakeInterval_32s	527
3.53.1.7 SYS_WakeInterval_4s	528
3.53.1.8 SYS_WakeInterval_8s	528
3.53.1.9 WAKE_FLT_DI	528
3.53.1.10 WAKE_FLT_EN	528
3.53.1.11 WAKE_IO_CLUOUT0	528
3.53.1.12 WAKE_IO_CLUOUT1	528
3.53.1.13 WAKE_IO_CLUOUT2	529
3.53.1.14 WAKE_IO_CLUOUT3	529
3.53.1.15 WAKE_IO_P0_0	529
3.53.1.16 WAKE_IO_P0_12	529
3.53.1.17 WAKE_IO_P0_13	529
3.53.1.18 WAKE_IO_P0_14	529
3.53.1.19 WAKE_IO_P0_2	530
3.53.1.20 WAKE_IO_P2_0	530
3.53.1.21 WAKE_IO_P2_4	530
3.53.1.22 WAKE_IO_P2_7	530
3.53.2 枚举类型说明	530
3.53.2.1 WAKE_TRIG_x	530
3.53.3 函数说明	531
3.53.3.1 WAKE_EVTCLRSleep()	531

3.53.3.2 WAKE_FallSleep()	531
3.53.3.3 WAKE_SetWakeInterval()	531
3.53.3.4 WAKE_SetWakeIO()	531
3.54 lks32mc09x_wake.h	532
3.55 source/lks32mc09x_adc.c 文件参考	533
3.55.1 函数说明	534
3.55.1.1 ADC_ClearIRQFlag()	534
3.55.1.2 ADC_Disable()	534
3.55.1.3 ADC_Enable()	534
3.55.1.4 ADC_GetConversionValue()	535
3.55.1.5 ADC_GetConversionValueS32()	535
3.55.1.6 ADC_GetIRQFlag()	536
3.55.1.7 ADC_Init()	536
3.55.1.8 ADC_Reset()	537
3.55.1.9 ADC_SetChn()	538
3.55.1.10 ADC_SetGain()	538
3.55.1.11 ADC_SetRange()	538
3.55.1.12 ADC_SoftIdleTrig()	539
3.55.1.13 ADC_SoftTrig()	539
3.55.1.14 ADC_StructInit()	539
3.56 lks32mc09x_adc.c	540
3.57 source/lks32mc09x_clu.c 文件参考	543
3.57.1 函数说明	544
3.57.1.1 CLU_Disable()	544
3.57.1.2 CLU_Enable()	545
3.57.1.3 CLU_Init()	545
3.57.1.4 CLU_ReadOutValue()	546
3.57.1.5 CLU_StructInit()	546
3.58 lks32mc09x_clu.c	546
3.59 source/lks32mc09x_cmp.c 文件参考	547
3.59.1 函数说明	549
3.59.1.1 CMP_ClearIRQFlag()	549
3.59.1.2 CMP_Disable()	549
3.59.1.3 CMP_Enable()	550
3.59.1.4 CMP_GetFilterValue()	550
3.59.1.5 CMP_GetIRQFlag()	550
3.59.1.6 CMP_GetValue()	551
3.59.1.7 CMP_Init()	551
3.59.1.8 CMP_SetFilterWidth()	552
3.59.1.9 CMP_SetHysteresis()	552
3.59.1.10 CMP_SetIRQSignal()	553
3.59.1.11 CMP_SetSignal()	553

3.59.1.12 CMP_SetWindowSignal()	554
3.59.1.13 CMP_StructInit()	555
3.60 lks32mc09x_cmp.c	555
3.61 source/lks32mc09x_crc.c 文件参考	559
3.61.1 函数说明	560
3.61.1.1 CRC_Calc()	560
3.61.1.2 CRC_CalcCommon()	561
3.61.1.3 CRC_Disable()	561
3.61.1.4 CRC_Enable()	562
3.61.1.5 CRC_Init()	563
3.61.1.6 CRC_StructInit()	563
3.61.2 变量说明	564
3.61.2.1 CRC_XOROUT	564
3.62 lks32mc09x_crc.c	564
3.63 source/lks32mc09x_dac.c 文件参考	567
3.63.1 函数说明	569
3.63.1.1 DAC_Cmd()	569
3.63.1.2 DAC_Disable()	569
3.63.1.3 DAC_Enable()	570
3.63.1.4 DAC_Init()	571
3.63.1.5 DAC_Output()	571
3.63.1.6 DAC_OutputValue()	572
3.63.1.7 DAC_OutputVoltage()	572
3.63.1.8 DAC_Range()	573
3.63.1.9 DAC_Reset()	574
3.63.1.10 DAC_SetRange()	574
3.63.1.11 DAC_Step()	575
3.63.1.12 DAC_StructInit()	576
3.63.1.13 DAC_Value()	576
3.63.1.14 DAC_Voltage()	577
3.64 lks32mc09x_dac.c	578
3.65 source/lks32mc09x_dma.c 文件参考	581
3.65.1 函数说明	582
3.65.1.1 DMA_CHx_EN()	582
3.65.1.2 DMA_ClearIRQFlag()	583
3.65.1.3 DMA_Disable()	583
3.65.1.4 DMA_Enable()	583
3.65.1.5 DMA_GetFlag()	583
3.65.1.6 DMA_GetIRQFlag()	584
3.65.1.7 DMA_Init()	585
3.65.1.8 DMA_Mem2Reg()	585
3.65.1.9 DMA_Memcpy()	586

3.65.1.10 DMA_Reset()	588
3.65.1.11 DMA_SetAddr()	588
3.65.1.12 DMA_SetLength()	588
3.65.1.13 DMA_SetRequest()	589
3.65.1.14 DMA_StructInit()	589
3.65.1.15 DMA_Trigger()	589
3.66 lks32mc09x_dma.c	590
3.67 source/lks32mc09x_dsp.c 文件参考	593
3.67.1 函数说明	594
3.67.1.1 DSP_Disable()	594
3.67.1.2 DSP_Div()	595
3.67.1.3 DSP_Enable()	595
3.67.1.4 DSP_Init()	596
3.67.1.5 DSP_Mod()	596
3.67.1.6 DSP_Reset()	597
3.67.1.7 DSP_Rms()	597
3.67.1.8 DSP_Sqrt()	598
3.68 lks32mc09x_dsp.c	598
3.69 source/lks32mc09x_eeprom.c 文件参考	599
3.69.1 宏定义说明	600
3.69.1.1 EEPROM_CFG	600
3.69.1.2 EEPROM_RDATA	600
3.69.2 函数说明	600
3.69.2.1 EEPROM_Disable()	600
3.69.2.2 EEPROM_Enable()	601
3.69.2.3 EEPROM_Init()	601
3.69.2.4 EEPROM_InitDefault()	602
3.69.2.5 EEPROM_Read()	602
3.69.2.6 EEPROM_ReadData()	603
3.69.2.7 EEPROM_Reset()	603
3.69.2.8 EEPROM_StructInit()	604
3.69.2.9 EEPROM_Write()	604
3.69.2.10 EEPROM_WriteData()	605
3.70 lks32mc09x_eeprom.c	605
3.71 source/lks32mc09x_exti.c 文件参考	607
3.71.1 函数说明	608
3.71.1.1 EXTI_ClearIRQFlag()	608
3.71.1.2 EXTI_DisableIRQFlag()	608
3.71.1.3 EXTI_EnableIRQFlag()	608
3.71.1.4 EXTI_GetIRQFlag()	609
3.71.1.5 EXTI_Reset()	609
3.71.1.6 EXTI_SetClko()	609

3.71.1.7 EXTI_TriggerConfig()	610
3.71.2 变量说明	610
3.71.2.1 exti_ie_flg	610
3.72 lks32mc09x_exti.c	610
3.73 source/lks32mc09x_flash.c 文件参考	612
3.73.1 函数说明	612
3.73.1.1 FLASH_Erase()	612
3.73.1.2 FLASH_EraseNvr()	613
3.73.1.3 FLASH_Read()	613
3.73.1.4 FLASH_ReadNvr()	613
3.73.1.5 FLASH_Write()	614
3.73.1.6 FLASH_WriteNvr()	614
3.74 lks32mc09x_flash.c	615
3.75 source/lks32mc09x_gpio.c 文件参考	616
3.75.1 函数说明	617
3.75.1.1 GPIO_Config()	617
3.75.1.2 GPIO_Disable()	618
3.75.1.3 GPIO_Enable()	619
3.75.1.4 GPIO_Init()	619
3.75.1.5 GPIO_LockRegister()	620
3.75.1.6 GPIO_PinAFConfig()	620
3.75.1.7 GPIO_ReadInputData()	621
3.75.1.8 GPIO_ReadInputDataBit()	621
3.75.1.9 GPIO_ReadOutputData()	622
3.75.1.10 GPIO_ReadOutputDataBit()	622
3.75.1.11 GPIO_Reset()	622
3.75.1.12 GPIO_ResetBits()	623
3.75.1.13 GPIO_SetBits()	623
3.75.1.14 GPIO_StructInit()	623
3.75.1.15 GPIO_ToggleBits()	624
3.75.1.16 GPIO_UnlockRegister()	624
3.75.1.17 GPIO_Write()	625
3.75.1.18 GPIO_WriteBit()	625
3.76 lks32mc09x_gpio.c	625
3.77 source/lks32mc09x_hall.c 文件参考	628
3.77.1 函数说明	629
3.77.1.1 HALL_Clear_IRQ()	629
3.77.1.2 HALL_Cmd()	629
3.77.1.3 HALL_GetCaptureValue()	629
3.77.1.4 HALL_GetCount()	629
3.77.1.5 HALL_GetFilterValue()	630
3.77.1.6 HALL_GetFlag()	630

3.77.1.7 HALL_GetIRQFlag()	630
3.77.1.8 HALL_Init()	631
3.77.1.9 HALL_IsCaptureEvent()	631
3.77.1.10 HALL_IsOverFlowEvent()	631
3.77.1.11 HALL_SoftTrigger()	632
3.77.1.12 HALL_StructInit()	632
3.78 lks32mc09x_hall.c	632
3.79 source/lks32mc09x_i2c.c 文件参考	633
3.79.1 函数说明	635
3.79.1.1 I2C_ClearBusState()	635
3.79.1.2 I2C_ClearIRQFlag()	635
3.79.1.3 I2C_Disable()	636
3.79.1.4 I2C_Enable()	636
3.79.1.5 I2C_GetFlag()	637
3.79.1.6 I2C_GetIRQFlag()	637
3.79.1.7 I2C_Init()	638
3.79.1.8 I2C_ReadBusState()	638
3.79.1.9 I2C_ReadData()	639
3.79.1.10 I2C_SendData()	639
3.79.1.11 I2C_SetBaudRate()	639
3.79.1.12 I2C_StructInit()	640
3.79.1.13 I2C_TableAck()	641
3.79.1.14 I2C_TableIrq()	641
3.79.1.15 I2C_TableRead()	641
3.79.1.16 I2C_TableSend()	641
3.79.1.17 I2C_TableState()	642
3.79.2 变量说明	642
3.79.2.1 I2C_TableVariable	642
3.80 lks32mc09x_i2c.c	643
3.81 source/lks32mc09x_iwdg.c 文件参考	646
3.81.1 函数说明	647
3.81.1.1 IWDG_DISABLE()	647
3.81.1.2 IWDG_ENABLE()	647
3.81.1.3 IWDG_Feed()	647
3.81.1.4 IWDG_Init()	647
3.81.1.5 IWDG_StructInit()	648
3.81.1.6 IWDG_Update()	648
3.82 lks32mc09x_iwdg.c	648
3.83 source/lks32mc09x_mcpwm.c 文件参考	649
3.83.1 函数说明	650
3.83.1.1 MCPWM_ClearErrorIRQFlag()	650
3.83.1.2 MCPWM_ClearIRQFlag()	651

3.83.1.3 MCPWM_Disable()	651
3.83.1.4 MCPWM_Enable()	652
3.83.1.5 MCPWM_GetCountTh()	653
3.83.1.6 MCPWM_GetCountVal()	653
3.83.1.7 MCPWM_GetEifFlag()	653
3.83.1.8 MCPWM_GetErrorIRQFlag()	654
3.83.1.9 MCPWM_GetIfFlag()	654
3.83.1.10 MCPWM_GetIRQFlag()	655
3.83.1.11 MCPWM_Init()	655
3.83.1.12 MCPWM_Reset()	656
3.83.1.13 MCPWM_SetIOConfig()	656
3.83.1.14 MCPWM_SetOutputState()	657
3.83.1.15 MCPWM_SetOutputVal()	657
3.83.1.16 MCPWM_SetTMR()	657
3.83.1.17 MCPWM_StartCount()	658
3.83.1.18 MCPWM_StopCount()	658
3.83.1.19 MCPWM_StructInit()	658
3.83.1.20 MCPWM_TmpUpdateFaillo()	659
3.83.2 变量说明	659
3.83.2.1 mcpwm_default_io	659
3.84 lks32mc09x_mcpwm.c	659
3.85 source/lks32mc09x_opa.c 文件参考	663
3.85.1 函数说明	663
3.85.1.1 OPA_Disable()	663
3.85.1.2 OPA_Enable()	664
3.85.1.3 OPA_Init()	664
3.85.1.4 OPA_OutCmpSignal()	665
3.85.1.5 OPA_SetGain()	665
3.85.1.6 OPA_SetIt()	667
3.85.1.7 OPA_StructInit()	667
3.86 lks32mc09x_opa.c	668
3.87 source/lks32mc09x_qep.c 文件参考	669
3.87.1 函数说明	670
3.87.1.1 QEP_ClearIRQFlag()	670
3.87.1.2 QEP_Cmd()	670
3.87.1.3 QEP_GetCount()	671
3.87.1.4 QEP_GetFlag()	671
3.87.1.5 QEP_GetIRQFlag()	672
3.87.1.6 QEP_Init()	672
3.87.1.7 QEP_SetMode()	672
3.87.1.8 QEP_StructInit()	673
3.88 lks32mc09x_qep.c	673

3.89 source/lks32mc09x_sif.c 文件参考	674
3.89.1 函数说明	675
3.89.1.1 SIF_Clear_Irq()	675
3.89.1.2 SIF_DMA_IsEnable()	676
3.89.1.3 SIF_Init()	676
3.89.1.4 SIF_Irq_IsEnable()	676
3.89.1.5 SIF_Senddata()	677
3.89.1.6 SIF_StructInit()	677
3.89.1.7 SIF_TH_set()	677
3.89.1.8 SIF_Timebase_set()	678
3.90 lks32mc09x_sif.c	678
3.91 source/lks32mc09x_spi.c 文件参考	679
3.91.1 函数说明	680
3.91.1.1 SPI_ClearIRQFlag()	680
3.91.1.2 SPI_GetIRQFlag()	680
3.91.1.3 SPI_Init()	681
3.91.1.4 SPI_ReadData()	681
3.91.1.5 SPI_SendData()	681
3.91.1.6 SPI_StructInit()	682
3.92 lks32mc09x_spi.c	682
3.93 source/lks32mc09x_sys.c 文件参考	683
3.93.1 函数说明	684
3.93.1.1 SYS_I2CClkDiv()	684
3.93.1.2 SYS_MclkChoice()	684
3.93.1.3 SYS_ModuleClockCmd()	685
3.93.1.4 SYS_ReadMcuClk()	686
3.93.1.5 SYS_ResetIO()	687
3.93.1.6 SYS_ResetPeripheral()	687
3.93.1.7 SYS_ResetPeripheralInternal()	688
3.93.1.8 SYS_SetPowerFailureDetection()	688
3.93.1.9 SYS_SoftReset_all()	689
3.93.1.10 SYS_SoftResetModule()	689
3.93.1.11 SYS_SPIClkDiv()	690
3.93.1.12 SYS_UartClkDiv()	690
3.93.1.13 SYS_WaitPowerReady()	691
3.94 lks32mc09x_sys.c	692
3.95 source/lks32mc09x_timer.c 文件参考	695
3.95.1 函数说明	696
3.95.1.1 TIMER_ClearIRQFlag()	696
3.95.1.2 TIMER_CMP0()	696
3.95.1.3 TIMER_CMP1()	696
3.95.1.4 TIMER_Disable()	697

3.95.1.5	TIMER_Enable()	697
3.95.1.6	TIMER_GetFlag()	698
3.95.1.7	TIMER_GetIRQFlag()	699
3.95.1.8	TIMER_Init()	699
3.95.1.9	TIMER_SetOutputState()	699
3.95.1.10	TIMER_StructInit()	700
3.95.1.11	TIMER_Timer_GetCMPT0()	700
3.95.1.12	TIMER_Timer_GetCMPT1()	700
3.95.1.13	TIMER_Timer_GetCount()	701
3.95.1.14	TIMER_TimerCmd()	701
3.96	lks32mc09x_timer.c	701
3.97	source/lks32mc09x_tmp.c 文件参考	704
3.97.1	函数说明	705
3.97.1.1	TMP_Disable()	705
3.97.1.2	TMP_Enable()	705
3.97.1.3	TMP_GetCurrentTemperature()	705
3.97.1.4	TMP_Init()	706
3.97.1.5	TMP_Reset()	706
3.97.2	变量说明	707
3.97.2.1	tmp_b	707
3.97.2.2	tmp_k	707
3.98	lks32mc09x_tmp.c	707
3.99	source/lks32mc09x_uart.c 文件参考	708
3.99.1	函数说明	709
3.99.1.1	UART_ClearIRQFlag()	709
3.99.1.2	UART_GetIRQFlag()	709
3.99.1.3	UART_Init()	709
3.99.1.4	UART_ReadBaud()	710
3.99.1.5	UART_ReadData()	710
3.99.1.6	UART_RecvnData()	711
3.99.1.7	UART_SendAddr_485()	712
3.99.1.8	UART_SendData()	712
3.99.1.9	UART_SendData_485()	713
3.99.1.10	UART_SendnData()	714
3.99.1.11	UART_SetBaud()	714
3.99.1.12	UART_SetClkDiv()	715
3.99.1.13	UART_StructInit()	716
3.100	lks32mc09x_uart.c	716
3.101	source/lks32mc09x_wake.c 文件参考	718
3.101.1	函数说明	718
3.101.1.1	WAKE_EVTCLRSleep()	718
3.101.1.2	WAKE_FallSleep()	719

3.101.1.3 WAKE_SetWakeInterval()	719
3.101.1.4 WAKE_SetWakeIO()	719
3.102 lks32mc09x_wake.c	720

0.1 结构体

这里列出了所有结构体，并附带简要说明：

ADC_InitTypeDef	ADC初始化结构体定义	5
ADC_TypeDef	ADC0寄存器 (ADC0)	9
AON_TypeDef	AON寄存器结构体句柄	16
CLU_InitTypeDef		18
CLU_TypeDef	CL0寄存器	21
CMP_InitTypeDef		23
CMP_SingleTypeDef	CMP初始化结构体句柄，初始化时请定义该句柄，并用其它地址来传参	25
CMP_TypeDef	比较器寄存器 (CMP)	28
CRC_InitTypeDef	CRC初始化结构体	30
CRC_TypeDef	CRC0寄存器 (CRC0)	32
DAC_InitTypeDef		33
DMA_InitTypeDef	DMA初始化结构体句柄，初始化时请定义该句柄，并用其它地址来传参	34
DMA_TypeDef	DMA寄存器	37
DMAMODULE_TypeDef	DMA控制状态寄存器对象定义	39
DSP_TypeDef	DSP0寄存器结构体 (DSP0)	40
EEPROM_InitTypeDef	EEPROM初始化结构体定义	42
EXTI_TypeDef	EXTI寄存器 (EXTI)	44
FLASH_TypeDef	FLASH寄存器 (FLASH)	46
GPIO_InitTypeDef	GPIO功能配置结构体句柄	51
GPIO_TypeDef	GPIO寄存器	52
HALL_InitTypeDef	HALL初始化结构体句柄，初始化时请定义该句柄，并用其它地址来传参	57
HALL_TypeDef	HALL0寄存器 (HALL0)	59
I2C_InitTypeDef		61
I2C_TypeDef	I2C0寄存器 (I2C0)	63
iicmpval		65
IWDG_InitTypeDef	IWDG初始化结构体句柄	67
IWDG_TypeDef	IWDG寄存器结构体句柄	68
MCPWM_InitTypeDef		69
MCPWM_TypeDef	MCPWM0寄存器 (MCPWM0)	73
OPA_InitTypeDef	OPA初始化结构体句柄，初始化时请定义该句柄，并用其它地址来传参	84

QEP_InitTypeDef	
QEP初始化结构体定义	85
QEP_TypeDef	
QEP0寄存器 (QEP0)	87
SIF_InitTypeDef	
SIF初始化结构体定义	89
SIF_TypeDef	
SIF0寄存器 (SIF0)	92
SPI_InitTypeDef	
SPI模块初始化结构体	94
SPI_TypeDef	
SPI0寄存器 (SPI0)	97
SYS_TypeDef	
SYS寄存器 (SYS)	98
TIMER_TimerInitTypeDef	
Timer初始化结构体	107
TIMER_TypeDef	
TIMER0寄存器 (TIMER0)	116
UART_InitTypeDef	
UART 初始化结构体定义	119
UART_TypeDef	
UART寄存器	123

Chapter 1

文件索引

1.1 文件列表

这里列出了所有文件，并附带简要说明:

include/lks32mc09x_adc.h	127
include/lks32mc09x_clu.h	150
include/lks32mc09x_cmp.h	181
include/lks32mc09x_crc.h	203
include/lks32mc09x_dac.h	212
include/lks32mc09x_dma.h	228
include/lks32mc09x_dsp.h	250
include/lks32mc09x_eeprom.h	255
include/lks32mc09x_exti.h	263
include/lks32mc09x_flash.h	274
include/lks32mc09x_gpio.h	319
include/lks32mc09x_hall.h	344
include/lks32mc09x_i2c.h	352
include/lks32mc09x_iwdg.h	364
include/lks32mc09x_lib.h	368
include/lks32mc09x_mcpwm.h	369
include/lks32mc09x_opa.h	412
include/lks32mc09x_qep.h	423
include/lks32mc09x_reg.h	433
include/lks32mc09x_sif.h	450
include/lks32mc09x_spi.h	456
include/lks32mc09x_sys.h	463
include/lks32mc09x_timer.h	481
include/lks32mc09x_tmp.h	491
include/lks32mc09x_trim.h	494
include/lks32mc09x_uart.h	508
include/lks32mc09x_wake.h	525
source/lks32mc09x_adc.c	533
source/lks32mc09x_clu.c	543
source/lks32mc09x_cmp.c	547
source/lks32mc09x_crc.c	559
source/lks32mc09x_dac.c	567
source/lks32mc09x_dma.c	581
source/lks32mc09x_dsp.c	593
source/lks32mc09x_eeprom.c	599

source/lks32mc09x_exti.c	607
source/lks32mc09x_flash.c	612
source/lks32mc09x_gpio.c	616
source/lks32mc09x_hall.c	628
source/lks32mc09x_i2c.c	633
source/lks32mc09x_iwdg.c	646
source/lks32mc09x_mcpwm.c	649
source/lks32mc09x_opa.c	663
source/lks32mc09x_qep.c	669
source/lks32mc09x_sif.c	674
source/lks32mc09x_spi.c	679
source/lks32mc09x_sys.c	683
source/lks32mc09x_timer.c	695
source/lks32mc09x_tmp.c	704
source/lks32mc09x_uart.c	708
source/lks32mc09x_wake.c	718

Chapter 2

结构体说明

2.1 ADC_InitTypeDef结构体 参考

ADC初始化结构体定义

```
#include <lks32mc09x_adc.h>
```

成员变量

- uint16_t **IE**
中断使能
- uint16_t **RE**
DMA请求使能
- uint16_t **NSMP**
两段采样使能
- uint16_t **DATA_ALIGN**
DAT右对齐使能
- uint16_t **CSMP**
连续采样使能
- uint16_t **TCNT**
触发一次采样所需的事件数
- uint16_t **TROVS**
手动触发过采样使能，开启后一次采样需要多次触发
- uint16_t **OVSR**
过采样率
- uint32_t **TRIG**
触发信号
- uint16_t **S1**
第一段常规采样次数，1表示触发1次采样1次，2表示触发1次采样2次
- uint16_t **S2**
第二段常规采样次数
- uint16_t **IS1**
空闲采样次数
- uint16_t **GAIN**
低增益（7.2V量程）使能

- `uint16_t LTH`
ADC 模拟看门狗 0 下阈值
- `uint16_t HTH`
ADC 模拟看门狗 0 上阈值
- `uint16_t GEN`
ADC 模拟看门狗 0 对应使能位
- `uint16_t IDLE_PRI`
空闲触发高优先级使能，默认为 0，空闲触发优先级低于常规触发

2.1.1 详细描述

ADC 初始化结构体定义

在文件 `lks32mc09x_adc.h` 第 9 行定义.

2.1.2 结构体成员变量说明

2.1.2.1 CSMP

`uint16_t CSMP`

连续采样使能

在文件 `lks32mc09x_adc.h` 第 15 行定义.

2.1.2.2 DATA_ALIGN

`uint16_t DATA_ALIGN`

DAT 右对齐使能

在文件 `lks32mc09x_adc.h` 第 14 行定义.

2.1.2.3 GAIN

`uint16_t GAIN`

低增益（7.2V 量程）使能

在文件 `lks32mc09x_adc.h` 第 23 行定义.

2.1.2.4 GEN

`uint16_t GEN`

ADC 模拟看门狗 0 对应使能位

在文件 `lks32mc09x_adc.h` 第 26 行定义.

2.1.2.5 HTH

`uint16_t HTH`

ADC 模拟看门狗 0 上阈值

在文件 [lks32mc09x_adc.h](#) 第 25 行定义.

2.1.2.6 IDLE_PRI

`uint16_t IDLE_PRI`

空闲触发高优先级使能，默认为0，空闲触发优先级低于常规触发

在文件 [lks32mc09x_adc.h](#) 第 27 行定义.

2.1.2.7 IE

`uint16_t IE`

中断使能

在文件 [lks32mc09x_adc.h](#) 第 11 行定义.

2.1.2.8 IS1

`uint16_t IS1`

空闲采样次数

在文件 [lks32mc09x_adc.h](#) 第 22 行定义.

2.1.2.9 LTH

`uint16_t LTH`

ADC 模拟看门狗 0 下阈值

在文件 [lks32mc09x_adc.h](#) 第 24 行定义.

2.1.2.10 NSMP

`uint16_t NSMP`

两段采样使能

在文件 [lks32mc09x_adc.h](#) 第 13 行定义.

2.1.2.11 OVSR

uint16_t OVSR

过采样率

在文件 [lks32mc09x_adc.h](#) 第 18 行定义.

2.1.2.12 RE

uint16_t RE

DMA请求使能

在文件 [lks32mc09x_adc.h](#) 第 12 行定义.

2.1.2.13 S1

uint16_t S1

第一段常规采样次数, 1表示触发1次采样1次, 2表示触发1次采样2次

在文件 [lks32mc09x_adc.h](#) 第 20 行定义.

2.1.2.14 S2

uint16_t S2

第二段常规采样次数

在文件 [lks32mc09x_adc.h](#) 第 21 行定义.

2.1.2.15 TCNT

uint16_t TCNT

触发一次采样所需的事件数

在文件 [lks32mc09x_adc.h](#) 第 16 行定义.

2.1.2.16 TRIG

uint32_t TRIG

触发信号

在文件 [lks32mc09x_adc.h](#) 第 19 行定义.

2.1.2.17 TROVS

```
uint16_t TROVS
```

手动触发过采样使能，开启后一次采样需要多次触发

在文件 [lks32mc09x_adc.h](#) 第 17 行定义。

2.2 ADC_TypeDef结构体 参考

ADC0寄存器 (ADC0)

```
#include <lks32mc09x_reg.h>
```

成员变量

- `__IOM uint32_t DAT0`
 < ADC0结构体
- `__IOM uint32_t DAT1`
 第1个样本数据 [15:0] ADC0第1次采样数据
- `__IOM uint32_t DAT2`
 第2个样本数据 [15:0] ADC0第2次采样数据
- `__IOM uint32_t DAT3`
 第3个样本数据 [15:0] ADC0第3次采样数据
- `__IOM uint32_t DAT4`
 第4个样本数据 [15:0] ADC0第4次采样数据
- `__IOM uint32_t DAT5`
 第5个样本数据 [15:0] ADC0第5次采样数据
- `__IOM uint32_t DAT6`
 第6个nsm/第6个ism数据 [15:0] ADC0第6次采样数据/第5次空闲采样数据
- `__IOM uint32_t DAT7`
 第7个nsm/第4个ism数据 [15:0] ADC0第7次采样数据/第4次空闲采样数据
- `__IOM uint32_t DAT8`
 第8个nsm/第3个ism数据 [15:0] ADC0第8次采样数据/第3次空闲采样数据
- `__IOM uint32_t DAT9`
 第9个nsm/第2个ism数据 [15:0] ADC0第9次采样数据/第2次空闲采样数据
- `__IOM uint32_t DAT10`
 第10个nsm/第1个ism数据 [15:0] ADC0第10次采样数据/第1次空闲采样数据
- `__IOM uint32_t DAT11`
 第11个nsm/第0个ism数据 [15:0] ADC0第11次采样数据/第0次空闲采样数据
- `__IOM 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 [15] CL_OUTPUT[3]触发ADC常规采样使能 [14] CL_OUTPUT[2]触发ADC常规采样使能 [13] CL_OUTPUT[1]触发ADC常规采样使能 [12] CL_OUTPUT[0]触发ADC常规采样使能 [11] TIMER2比较/捕获事件1触发ADC常规采样使能 [10] TIMER2比较/捕获事件0触发ADC常规采样使能 [9] TIMER1比较/捕获事件1触发ADC常规采样使能 [8] TIMER0比较/捕获事件0触发ADC常规采样使能 [7] MCPWM0 T3事件触发ADC常规采样使能 [6] MCPWM0 T2事件触发ADC常规采样使能 [5] MCPWM0 T1事件触发ADC常规采样使能 [4] MCPWM0 T0事件触发ADC常规采样使能 [3] CL_OUTPUT[3]触发ADC常规采样使能 [2] CL_OUTPUT[2]触发ADC常规采样使能 [1] CL_OUTPUT[1]触发ADC常规采样使能 [0] CL_OUTPUT[0]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

- __IM 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

2.2.1 详细描述

ADC0寄存器 (ADC0)

在文件 [lks32mc09x_reg.h](#) 第 8 行定义.

2.2.2 结构体成员变量说明

2.2.2.1 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 [lks32mc09x_reg.h](#) 第 34 行定义.

2.2.2.2 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 [lks32mc09x_reg.h](#) 第 36 行定义.

2.2.2.3 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 [lks32mc09x_reg.h](#) 第 30 行定义.

2.2.2.4 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 [lks32mc09x_reg.h](#) 第 23 行定义.

2.2.2.5 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 [lks32mc09x_reg.h](#) 第 24 行定义.

2.2.2.6 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 [lks32mc09x_reg.h](#) 第 25 行定义.

2.2.2.7 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 [lks32mc09x_reg.h](#) 第 27 行定义.

2.2.2.8 DAT0

__IOM uint32_t DAT0

< ADC0结构体

第0个样本数据 [15:0] ADC0第0次采样数据

在文件 [lks32mc09x_reg.h](#) 第 10 行定义.

2.2.2.9 DAT1

__IOM uint32_t DAT1

第1个样本数据 [15:0] ADC0第1次采样数据

在文件 [lks32mc09x_reg.h](#) 第 11 行定义.

2.2.2.10 DAT10

__IOM uint32_t DAT10

第10个nsmp/第1个ismp数据 [15:0] ADC0第10次采样数据/第1次空闲采样数据

在文件 [lks32mc09x_reg.h](#) 第 20 行定义.

2.2.2.11 DAT11

__IOM uint32_t DAT11

第11个nsmp/第0个ismp数据 [15:0] ADC0第11次采样数据/第0次空闲采样数据

在文件 [lks32mc09x_reg.h](#) 第 21 行定义.

2.2.2.12 DAT2

```
--IOM uint32_t DAT2
```

第2个样本数据 [15:0] ADC0第2次采样数据

在文件 [lks32mc09x_reg.h](#) 第 12 行定义.

2.2.2.13 DAT3

```
--IOM uint32_t DAT3
```

第3个样本数据 [15:0] ADC0第3次采样数据

在文件 [lks32mc09x_reg.h](#) 第 13 行定义.

2.2.2.14 DAT4

```
--IOM uint32_t DAT4
```

第4个样本数据 [15:0] ADC0第4次采样数据

在文件 [lks32mc09x_reg.h](#) 第 14 行定义.

2.2.2.15 DAT5

```
--IOM uint32_t DAT5
```

第5个样本数据 [15:0] ADC0第5次采样数据

在文件 [lks32mc09x_reg.h](#) 第 15 行定义.

2.2.2.16 DAT6

```
--IOM uint32_t DAT6
```

第6个nsmpl/第6个ismpl数据 [15:0] ADC0第6次采样数据/第5次空闲采样数据

在文件 [lks32mc09x_reg.h](#) 第 16 行定义.

2.2.2.17 DAT7

```
--IOM uint32_t DAT7
```

第7个nsmpl/第4个ismpl数据 [15:0] ADC0第7次采样数据/第4次空闲采样数据

在文件 [lks32mc09x_reg.h](#) 第 17 行定义.

2.2.2.18 DAT8

--IOM uint32_t DAT8

第8个nsmp/第3个ismp数据 [15:0] ADC0第8次采样数据/第3次空闲采样数据

在文件 [lks32mc09x_reg.h](#) 第 18 行定义.

2.2.2.19 DAT9

--IOM uint32_t DAT9

第9个nsmp/第2个ismp数据 [15:0] ADC0第9次采样数据/第2次空闲采样数据

在文件 [lks32mc09x_reg.h](#) 第 19 行定义.

2.2.2.20 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 [lks32mc09x_reg.h](#) 第 33 行定义.

2.2.2.21 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 [lks32mc09x_reg.h](#) 第 35 行定义.

2.2.2.22 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 [lks32mc09x_reg.h](#) 第 28 行定义.

2.2.2.23 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 [lks32mc09x_reg.h](#) 第 42 行定义.

2.2.2.24 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 [lks32mc09x_reg.h](#) 第 41 行定义.

2.2.2.25 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 [lks32mc09x_reg.h](#) 第 37 行定义.

2.2.2.26 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 [lks32mc09x_reg.h](#) 第 38 行定义.

2.2.2.27 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 [lks32mc09x_reg.h](#) 第 40 行定义.

2.2.2.28 RESERVED

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

在文件 [lks32mc09x_reg.h](#) 第 22 行定义.

2.2.2.29 RESERVED1

```
--IM uint32_t RESERVED1
```

在文件 [lks32mc09x_reg.h](#) 第 26 行定义.

2.2.2.30 RESERVED2

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

在文件 [lks32mc09x_reg.h](#) 第 29 行定义.

2.2.2.31 RESERVED3

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

在文件 [lks32mc09x_reg.h](#) 第 39 行定义.

2.2.2.32 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 [lks32mc09x_reg.h](#) 第 32 行定义.

2.2.2.33 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 [15] CL_OUTPUT[3]触发ADC常规采样使能 [14] CL_OUTPUT[2]触发ADC常规采样使能 [13] CL_OUTPUT[1]触发ADC常规采样使能 [12] CL_OUTPUT[0]触发ADC常规采样使能 [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 [lks32mc09x_reg.h](#) 第 31 行定义.

2.3 AON_TypeDef结构体 参考

AON寄存器结构体句柄

```
#include <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* 唤醒 [7] *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
- `__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

2.3.1 详细描述

AON寄存器结构体句柄

在文件 [lks32mc09x_reg.h](#) 第 380 行定义.

2.3.2 结构体成员变量说明

2.3.2.1 BKP_REG0

```
__IOM uint32_t BKP_REG0
```

后备寄存器 0

在文件 [lks32mc09x_reg.h](#) 第 386 行定义.

2.3.2.2 BKP_REG1

```
__IOM uint32_t BKP_REG1
```

后备寄存器 1

在文件 [lks32mc09x_reg.h](#) 第 387 行定义.

2.3.2.3 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 [lks32mc09x_reg.h](#) 第 383 行定义.

2.3.2.4 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 [lks32mc09x_reg.h](#) 第 385 行定义.

2.3.2.5 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 [lks32mc09x_reg.h](#) 第 384 行定义.

2.3.2.6 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 [lks32mc09x_reg.h](#) 第 382 行定义.

2.4 CLU_InitTypeDef结构体 参考

```
#include <lks32mc09x_clu.h>
```

成员变量

- `uint32_t EN`
模块使能 0:关闭模块 1:打开模块
- `uint32_t FIE`
下降沿中断使能
- `uint32_t RIE`
上升沿中断使能
- `uint32_t MXA`
A输入复用选择 格式 `CLU_MXA_CLUa_b` a取值范围为[0 1 2 3] 对应不同的CLU b表示选择不同的输入信号
- `uint32_t MXB`
B输入复用选择 格式 `CLU_MXB_CLUa_b` a取值范围为[0 1 2 3] 对应不同的CLU b表示选择不同的输入信号
- `uint32_t FN`
真值表 格式 `CLU_FN_x` x取值范围为[A NOTA A_NOTB ...]对应为最简与或式下的最小相 例如 $y=a+bc+!c$ 表示为 `CLU_FN_A|CLU_FN.B.C|CLU_FN.NOTC`
- `uint32_t OUTSEL`
输出选择 0: D 触发器输出 1: LUT 输出
- `uint32_t OEN`
输出使能 0: 关闭 1: 使能
- `uint32_t CLKINV`
D 触发器时钟电平 0: 同相 1: 反向
- `uint32_t CLKSEL`
D 触发器时钟选择 0x0: 进位输入 0x1: MXA 输入 0x2: ALTCLK0 0x3: ALTCLK1

2.4.1 详细描述

在文件 `lks32mc09x.clu.h` 第 6 行定义.

2.4.2 结构体成员变量说明

2.4.2.1 CLKINV

`uint32_t CLKINV`

D 触发器时钟电平 0: 同相 1: 反向

在文件 `lks32mc09x.clu.h` 第 16 行定义.

2.4.2.2 CLKSEL

`uint32_t CLKSEL`

D 触发器时钟选择 0x0: 进位输入 0x1: MXA 输入 0x2: ALTCLK0 0x3: ALTCLK1

在文件 `lks32mc09x.clu.h` 第 17 行定义.

2.4.2.3 EN

uint32_t EN

模块使能 0关闭模块 1打开模块

在文件 [lks32mc09x_clu.h](#) 第 8 行定义.

2.4.2.4 FIE

uint32_t FIE

下降沿中断使能

在文件 [lks32mc09x_clu.h](#) 第 9 行定义.

2.4.2.5 FN

uint32_t FN

真值表 格式 CLU_FN_x x取值范围为[A NOTA A.NOTB ...]对应为最简与或式下的最小相 例如 $y=a+bc+!c$ 表示为 CLU_FN_A|CLU_FN_B.C|CLU_FN_NOTC

在文件 [lks32mc09x_clu.h](#) 第 13 行定义.

2.4.2.6 MXA

uint32_t MXA

A输入复用选择 格式 CLU_MXA.CLUa_b a取值范围为[0 1 2 3] 对应不同的CLU b表示选择不同的输入信号

在文件 [lks32mc09x_clu.h](#) 第 11 行定义.

2.4.2.7 MXB

uint32_t MXB

B输入复用选择 格式 CLU_MXB.CLUa_b a取值范围为[0 1 2 3] 对应不同的CLU b表示选择不同的输入信号

在文件 [lks32mc09x_clu.h](#) 第 12 行定义.

2.4.2.8 OEN

uint32_t OEN

输出使能 0: 关闭 1: 使能

在文件 [lks32mc09x_clu.h](#) 第 15 行定义.

2.4.2.9 OUTSEL

uint32_t OUTSEL

输出选择 0: D 触发器输出 1: LUT 输出

在文件 [lks32mc09x_clu.h](#) 第 14 行定义.

2.4.2.10 RIE

uint32_t RIE

上升沿中断使能

在文件 [lks32mc09x_clu.h](#) 第 10 行定义.

2.5 CLU_TypeDef结构体 参考

CL0寄存器

```
#include <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`

2.5.1 详细描述

CL0寄存器

在文件 [lks32mc09x_reg.h](#) 第 59 行定义.

2.5.2 结构体成员变量说明

2.5.2.1 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 [lks32mc09x_reg.h](#) 第 67 行定义.

2.5.2.2 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 [lks32mc09x_reg.h](#) 第 61 行定义.

2.5.2.3 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 [lks32mc09x_reg.h](#) 第 66 行定义.

2.5.2.4 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 [lks32mc09x_reg.h](#) 第 62 行定义.

2.5.2.5 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 [lks32mc09x_reg.h](#) 第 63 行定义.

2.5.2.6 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 [lks32mc09x_reg.h](#) 第 65 行定义.

2.5.2.7 OUT

```
--IOM uint32_t OUT
```

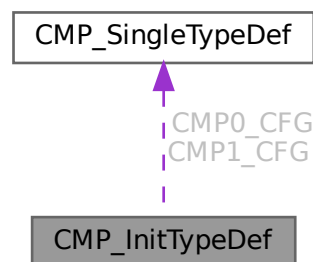
CL0输出寄存器 [3] CLU3输出状态 [2] CLU2输出状态 [1] CLU1输出状态 [0] CLU0输出状态

在文件 [lks32mc09x_reg.h](#) 第 64 行定义.

2.6 CMP_InitTypeDef结构体 参考

```
#include <lks32mc09x_cmp.h>
```

CMP_InitTypeDef 的协作图:



成员变量

- u8 **HYS**
比较器回差选择
- u32 **CLK_COM_DIV**
比较器共用分频系数，取值0-4
- **CMP_SingleTypeDef CMP0_CFG**
CMP0配置
- **CMP_SingleTypeDef CMP1_CFG**
CMP1配置

2.6.1 详细描述

在文件 [lks32mc09x_cmp.h](#) 第 25 行定义.

2.6.2 结构体成员变量说明

2.6.2.1 CLK_COM_DIV

u32 CLK_COM_DIV

比较器共用分频系数，取值0-4

在文件 [lks32mc09x_cmp.h](#) 第 28 行定义.

2.6.2.2 CMP0_CFG

CMP_SingleTypeDef CMP0_CFG

CMP0配置

在文件 [lks32mc09x_cmp.h](#) 第 29 行定义.

2.6.2.3 CMP1_CFG

CMP_SingleTypeDef CMP1_CFG

CMP1配置

在文件 [lks32mc09x_cmp.h](#) 第 30 行定义.

2.6.2.4 HYS

u8 HYS

比较器回差选择

在文件 [lks32mc09x_cmp.h](#) 第 27 行定义.

2.7 CMP_SingleTypeDef结构体 参考

CMP初始化结构体句柄，初始化时请定义该句柄，并用其它地址来传参

```
#include <lks32mc09x_cmp.h>
```

成员变量

- u16 SELP
比较器信号正端选择
- u16 SELN
比较器信号负端选择
- u32 FIL_CLK_DIV16
比较器012滤波系数，赋值范围0-0xF
- u32 FIL_CLK_DIV2
比较器012滤波系数，0x0:1分频，0x1:2分频,0x2:4分频,0x3:8分频，0x4:16分频，0x5:32分频,0x6:64分频,0x7:128分频
- u16 POL
比较器有效输出极性选择
- u16 W_PWM_POL
比较器开窗 PWM 信号极性选择,比较器 0/1
- u16 IRQ_TRIG
比较器边沿/电平触发中断选择 0电平触发 1边沿触发
- FuncState EN
比较器使能
DISABLE:失能，ENABLE: 使能
- FuncState IE
比较器中断使能，DISABLE:失能，ENABLE: 使能
- FuncState RE
比较器DMA 请求使能，DISABLE:失能，ENABLE: 使能
- FuncState CHN3P_PWM.EN
MCPWM 模块 CHN3_P 通道使能比较器 0/1 开窗，DISABLE:失能，ENABLE: 使能
- FuncState CHN2P_PWM.EN
MCPWM 模块 CHN2_P 通道使能比较器 0/1 开窗，DISABLE:失能，ENABLE: 使能
- FuncState CHN1P_PWM.EN
MCPWM 模块 CHN1_P 通道使能比较器 0/1 开窗，DISABLE:失能，ENABLE: 使能
- FuncState CHN0P_PWM.EN
MCPWM 模块 CHN0_P 通道使能比较器 0/1 开窗，DISABLE:失能，ENABLE: 使能

2.7.1 详细描述

CMP初始化结构体句柄，初始化时请定义该句柄，并用其它地址来传参

在文件 lks32mc09x_cmp.h 第 8 行定义.

2.7.2 结构体成员变量说明

2.7.2.1 CHN0P_PWM_EN

FuncState CHN0P_PWM_EN

MCPWM 模块 CHN0_P 通道使能比较器 0/1 开窗, DISABLE:失能, ENABLE: 使能
在文件 [lks32mc09x_cmp.h](#) 第 23 行定义.

2.7.2.2 CHN1P_PWM_EN

FuncState CHN1P_PWM_EN

MCPWM 模块 CHN1_P 通道使能比较器 0/1 开窗, DISABLE:失能, ENABLE: 使能
在文件 [lks32mc09x_cmp.h](#) 第 22 行定义.

2.7.2.3 CHN2P_PWM_EN

FuncState CHN2P_PWM_EN

MCPWM 模块 CHN2_P 通道使能比较器 0/1 开窗, DISABLE:失能, ENABLE: 使能
在文件 [lks32mc09x_cmp.h](#) 第 21 行定义.

2.7.2.4 CHN3P_PWM_EN

FuncState CHN3P_PWM_EN

MCPWM 模块 CHN3_P 通道使能比较器 0/1 开窗, DISABLE:失能, ENABLE: 使能
在文件 [lks32mc09x_cmp.h](#) 第 20 行定义.

2.7.2.5 EN

FuncState EN

比较器使能
DISABLE:失能, ENABLE: 使能

在文件 [lks32mc09x_cmp.h](#) 第 17 行定义.

2.7.2.6 FIL_CLK_DIV16

u32 FIL_CLK_DIV16

比较器012滤波系数, 赋值范围0-0xF

在文件 [lks32mc09x_cmp.h](#) 第 12 行定义.

2.7.2.7 FIL_CLK_DIV2

u32 FIL_CLK_DIV2

比较器012滤波系数，0x0:1分频，0x1:2分频,0x2:4分频,0x3:8分频，0x4:16分频，0x5:32分频,0x6:64分频,0x7:128分频

在文件 [lks32mc09x_cmp.h](#) 第 13 行定义.

2.7.2.8 IE

FuncState IE

比较器中断使能，DISABLE:失能，ENABLE：使能

在文件 [lks32mc09x_cmp.h](#) 第 18 行定义.

2.7.2.9 IRQ_TRIG

u16 IRQ_TRIG

比较器边沿/电平触发中断选择 0电平触发 1边沿触发

在文件 [lks32mc09x_cmp.h](#) 第 16 行定义.

2.7.2.10 POL

u16 POL

比较器有效输出极性选择

在文件 [lks32mc09x_cmp.h](#) 第 14 行定义.

2.7.2.11 RE

FuncState RE

比较器DMA 请求使能，DISABLE:失能，ENABLE：使能

在文件 [lks32mc09x_cmp.h](#) 第 19 行定义.

2.7.2.12 SELN

u16 SELN

比较器信号负端选择

在文件 [lks32mc09x_cmp.h](#) 第 11 行定义.

2.7.2.13 SELP

u16 SELP

比较器信号正端选择

在文件 [lks32mc09x_cmp.h](#) 第 10 行定义.

2.7.2.14 W_PWM_POL

u16 W_PWM_POL

比较器开窗 PWM 信号极性选择,比较器 0/1

在文件 [lks32mc09x_cmp.h](#) 第 15 行定义.

2.8 CMP_TypeDef结构体 参考

比较器寄存器 (CMP)

```
#include <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

2.8.1 详细描述

比较器寄存器 (CMP)

在文件 [lks32mc09x_reg.h](#) 第 47 行定义.

2.8.2 结构体成员变量说明

2.8.2.1 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 [lks32mc09x_reg.h](#) 第 53 行定义.

2.8.2.2 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 [lks32mc09x_reg.h](#) 第 52 行定义.

2.8.2.3 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 [lks32mc09x_reg.h](#) 第 54 行定义.

2.8.2.4 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 [lks32mc09x_reg.h](#) 第 49 行定义.

2.8.2.5 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 [lks32mc09x_reg.h](#) 第 50 行定义.

2.8.2.6 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 [lks32mc09x_reg.h](#) 第 51 行定义.

2.9 CRC.InitTypeDef结构体 参考

CRC初始化结构体

```
#include <lks32mc09x_crc.h>
```

成员变量

- [uint32_t POLY](#)
多项式
- [uint32_t INIT](#)
初始值
- [uint32_t XOROUT](#)
结果异或值
- [uint8_t REFIN](#)
输入数据反转
- [uint8_t REFOUT](#)
输出数据反转
- [uint8_t WIDTH](#)
掩码
- [uint8_t null](#)
地址对齐

2.9.1 详细描述

CRC初始化结构体

在文件 [lks32mc09x_crc.h](#) 第 9 行定义.

2.9.2 结构体成员变量说明

2.9.2.1 INIT

`uint32_t INIT`

初始值

在文件 [lks32mc09x_crc.h](#) 第 12 行定义.

2.9.2.2 null

`uint8_t null`

地址对齐

在文件 [lks32mc09x_crc.h](#) 第 17 行定义.

2.9.2.3 POLY

`uint32_t POLY`

多项式

在文件 [lks32mc09x_crc.h](#) 第 11 行定义.

2.9.2.4 REFIN

`uint8_t REFIN`

输入数据反转

在文件 [lks32mc09x_crc.h](#) 第 14 行定义.

2.9.2.5 REFOUT

`uint8_t REFOUT`

输出数据反转

在文件 [lks32mc09x_crc.h](#) 第 15 行定义.

2.9.2.6 WIDTH

`uint8_t WIDTH`

掩码

在文件 [lks32mc09x_crc.h](#) 第 16 行定义.

2.9.2.7 XOROUT

uint32_t XOROUT

结果异或值

在文件 [lks32mc09x_crc.h](#) 第 13 行定义.

2.10 CRC_TypeDef结构体 参考

CRC0寄存器 (CRC0)

```
#include <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

2.10.1 详细描述

CRC0寄存器 (CRC0)

在文件 [lks32mc09x_reg.h](#) 第 72 行定义.

2.10.2 结构体成员变量说明

2.10.2.1 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 [lks32mc09x_reg.h](#) 第 75 行定义.

2.10.2.2 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 [lks32mc09x_reg.h](#) 第 74 行定义.

2.10.2.3 INIT

```
--IOM uint32_t INIT
```

CRC初始值寄存器 [31:0] 存放初始码

在文件 [lks32mc09x_reg.h](#) 第 76 行定义.

2.10.2.4 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 [lks32mc09x_reg.h](#) 第 77 行定义.

2.11 DAC_InitTypeDef结构体 参考

```
#include <lks32mc09x_dac.h>
```

成员变量

- uint8_t [DAC_RANGE](#)
DAC量程 可选参数: *DAC_RANGE_0_4V76* *DAC_RANGE_0_1V2* *DAC_RANGE_1_3V* *DAC_RANGE_1_1V2*
- uint8_t [DACOUT_EN](#)
DAC输出到IO使能
- int8_t [DAC_STEP](#)
步进值, 3bit 有符号数, 范围-4~3
- uint8_t [DAC_STEP_TRG_x](#)
UTIMER0/1/2/3触发步进使能,*DISABLE*, 不使能; *ENABLE*, 使能

2.11.1 详细描述

在文件 [lks32mc09x_dac.h](#) 第 40 行定义.

2.11.2 结构体成员变量说明

2.11.2.1 DAC_RANGE

`uint8_t DAC_RANGE`

DAC量程 可选参数: `DAC_RANGE_0_4V76` `DAC_RANGE_0_1V2` `DAC_RANGE_1_3V` `DAC_RANGE_1_1V2`

在文件 [lks32mc09x-dac.h](#) 第 42 行定义.

2.11.2.2 DAC_STEP

`int8_t DAC_STEP`

步进值, 3bit 有符号数, 范围-4~3

在文件 [lks32mc09x-dac.h](#) 第 44 行定义.

2.11.2.3 DAC_STEP_TRG_x

`uint8_t DAC_STEP_TRG_x`

UTIMER0/1/2/3触发步进使能,DISABLE, 不使能; ENABLE, 使能

在文件 [lks32mc09x-dac.h](#) 第 45 行定义.

2.11.2.4 DACOUT_EN

`uint8_t DACOUT_EN`

DAC输出到IO使能

在文件 [lks32mc09x-dac.h](#) 第 43 行定义.

2.12 DMA_InitTypeDef结构体 参考

DMA初始化结构体句柄, 初始化时请定义该句柄, 并用其它地址来传参

```
#include <lks32mc09x-dma.h>
```

成员变量

- `uint8_t IRQ_EN`
- `uint8_t SBTW`
源地址访问位宽 0: Byte 1: Halfword 2: Word
- `uint8_t DBTW`
目的地址访问位宽 0: Byte 1: Halfword 2: Word
- `uint8_t SINC`
源地址递增使能
- `uint8_t DINC`
目的地址递增使能
- `uint8_t CIRC`
循环模式使能
- `uint8_t RMODE`
多轮传输使能, 多个DMA 请求完成传输
- `uint8_t EN`
通道使能
- `uint16_t TMS`
DMA 通道 x 数据搬运次数。
- `uint32_t SADR`
DMA 通道 x 源地址
- `uint32_t DADR`
DMA 通道 x 目的地址
- `uint32_t REN`
DMA 触发源 DMA.REN_x

2.12.1 详细描述

DMA初始化结构体句柄, 初始化时请定义该句柄, 并用其它地址来传参

在文件 `lks32mc09x_dma.h` 第 37 行定义.

2.12.2 结构体成员变量说明

2.12.2.1 CIRC

```
uint8_t CIRC
```

循环模式使能

在文件 `lks32mc09x_dma.h` 第 44 行定义.

2.12.2.2 DADR

```
uint32_t DADR
```

DMA 通道 x 目的地址

在文件 `lks32mc09x_dma.h` 第 49 行定义.

2.12.2.3 DBTW

uint8_t DBTW

目的地址访问位宽 0: Byte 1: Halfword 2: Word

在文件 [lks32mc09x_dma.h](#) 第 41 行定义.

2.12.2.4 DINC

uint8_t DINC

目的地址递增使能

在文件 [lks32mc09x_dma.h](#) 第 43 行定义.

2.12.2.5 EN

uint8_t EN

通道使能

在文件 [lks32mc09x_dma.h](#) 第 46 行定义.

2.12.2.6 IRQ_EN

uint8_t IRQ_EN

在文件 [lks32mc09x_dma.h](#) 第 39 行定义.

2.12.2.7 REN

uint32_t REN

DMA 触发源 DMA_REN_x

在文件 [lks32mc09x_dma.h](#) 第 50 行定义.

2.12.2.8 RMODE

uint8_t RMODE

多轮传输使能, 多个DMA 请求完成传输

在文件 [lks32mc09x_dma.h](#) 第 45 行定义.

2.12.2.9 SADR

uint32_t SADR

DMA 通道 x 源地址

在文件 [lks32mc09x_dma.h](#) 第 48 行定义.

2.12.2.10 SBTW

uint8_t SBTW

源地址访问位宽 0: Byte 1: Halfword 2: Word

在文件 [lks32mc09x_dma.h](#) 第 40 行定义.

2.12.2.11 SINC

uint8_t SINC

源地址递增使能

在文件 [lks32mc09x_dma.h](#) 第 42 行定义.

2.12.2.12 TMS

uint16_t TMS

DMA 通道 x 数据搬运次数。

在文件 [lks32mc09x_dma.h](#) 第 47 行定义.

2.13 DMA_TypeDef结构体 参考

DMA寄存器

```
#include <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

2.13.1 详细描述

DMA寄存器

在文件 [lks32mc09x_reg.h](#) 第 82 行定义.

2.13.2 结构体成员变量说明

2.13.2.1 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 [lks32mc09x_reg.h](#) 第 84 行定义.

2.13.2.2 CTMS

```
--IOM uint32_t CTMS
```

DMA通道0传输次数（保留字段）

在文件 [lks32mc09x_reg.h](#) 第 86 行定义.

2.13.2.3 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 [lks32mc09x_reg.h](#) 第 88 行定义.

2.13.2.4 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 [lks32mc09x_reg.h](#) 第 85 行定义.

2.13.2.5 SADR

```
--IOM uint32_t SADR
```

DMA通道0源地址 [31:0] DMA通道x源地址

在文件 [lks32mc09x_reg.h](#) 第 87 行定义.

2.14 DMAMODULE_TypeDef结构体 参考

DMA控制状态寄存器对象定义

```
#include <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

2.14.1 详细描述

DMA控制状态寄存器对象定义

在文件 [lks32mc09x_reg.h](#) 第 93 行定义.

2.14.2 结构体成员变量说明

2.14.2.1 CTRL

```
__IOM uint32_t CTRL
```

DMA控制 [0] DMA整体使能

在文件 [lks32mc09x_reg.h](#) 第 95 行定义.

2.14.2.2 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 [lks32mc09x_reg.h](#) 第 96 行定义.

2.14.2.3 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 [lks32mc09x_reg.h](#) 第 97 行定义.

2.15 DSP_Typedef结构体 参考

DSP0寄存器结构体 (DSP0)

```
#include <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

2.15.1 详细描述

DSP0寄存器结构体 (DSP0)

在文件 `lks32mc09x_reg.h` 第 102 行定义.

2.15.2 结构体成员变量说明

2.15.2.1 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 `lks32mc09x_reg.h` 第 105 行定义.

2.15.2.2 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 [lks32mc09x_reg.h](#) 第 109 行定义.

2.15.2.3 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 [lks32mc09x_reg.h](#) 第 106 行定义.

2.15.2.4 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 [lks32mc09x_reg.h](#) 第 110 行定义.

2.15.2.5 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 [lks32mc09x_reg.h](#) 第 107 行定义.

2.15.2.6 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 [lks32mc09x_reg.h](#) 第 111 行定义.

2.15.2.7 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 [lks32mc09x_reg.h](#) 第 113 行定义.

2.15.2.8 REM

```
--IM uint32_t REM
```

DSP 0次余数 [31:0] DSP除法余数寄存器

在文件 [lks32mc09x_reg.h](#) 第 108 行定义.

2.15.2.9 RESERVED

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

< DSP0 结构体

保留位

在文件 [lks32mc09x_reg.h](#) 第 104 行定义.

2.15.2.10 RESERVED1

```
--IM uint32_t RESERVED1
```

保留位

在文件 [lks32mc09x_reg.h](#) 第 112 行定义.

2.15.2.11 RESERVED2

```
--IM uint16_t RESERVED2
```

保留位

在文件 [lks32mc09x_reg.h](#) 第 115 行定义.

2.15.2.12 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 [lks32mc09x_reg.h](#) 第 114 行定义.

2.16 EEPROM_InitTypeDef结构体 参考

EEPROM初始化结构体定义

```
#include <lks32mc09x_eeprom.h>
```

成员变量

- `uint32_t addr`
eeprom地址
- `uint32_t EEPROM_BAUD_x`
通信波特率选择
- `uint32_t RDY_RLS`
软件等待选择信号, *disable*读写时不响应中断, 使能&失能不影响函数执行效果, 使能后读写时可以响应中断
- `uint32_t IO_SWITCH`
通信引脚交换使能
- `uint32_t ADDR_LEN_x`
eeprom数据地址长度

2.16.1 详细描述

EEPROM初始化结构体定义

在文件 `lks32mc09x_eeprom.h` 第 16 行定义.

2.16.2 结构体成员变量说明

2.16.2.1 addr

`uint32_t addr`

eeprom地址

在文件 `lks32mc09x_eeprom.h` 第 18 行定义.

2.16.2.2 ADDR_LEN_x

`uint32_t ADDR_LEN_x`

eeprom数据地址长度

在文件 `lks32mc09x_eeprom.h` 第 22 行定义.

2.16.2.3 EEPROM_BAUD_x

`uint32_t EEPROM_BAUD_x`

通信波特率选择

在文件 `lks32mc09x_eeprom.h` 第 19 行定义.

2.16.2.4 IO_SWITCH

uint32_t IO_SWITCH

通信引脚交换使能

在文件 [lks32mc09x_eeprom.h](#) 第 21 行定义。

2.16.2.5 RDY_RLS

uint32_t RDY_RLS

软件等待选择信号，disable读写时不响应中断，使能&失能不影响函数执行效果，使能后读写时可以响应中断

在文件 [lks32mc09x_eeprom.h](#) 第 20 行定义。

2.17 EXTI_TypeDef结构体 参考

EXTI寄存器 (EXTI)

```
#include <lks32mc09x_reg.h>
```

成员变量

- __IOM uint32_t [CR0](#)
 < EXTI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 [CLKO](#)
 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

2.17.1 详细描述

EXTI寄存器 (EXTI)

在文件 [lks32mc09x_reg.h](#) 第 120 行定义.

2.17.2 结构体成员变量说明

2.17.2.1 CLKO

__IOM uint32_t CLKO

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 [lks32mc09x_reg.h](#) 第 126 行定义.

2.17.2.2 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 [lks32mc09x_reg.h](#) 第 122 行定义.

2.17.2.3 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 [lks32mc09x_reg.h](#) 第 123 行定义.

2.17.2.4 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 [lks32mc09x_reg.h](#) 第 124 行定义.

2.17.2.5 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 [lks32mc09x_reg.h](#) 第 125 行定义。

2.17.2.6 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 [lks32mc09x_reg.h](#) 第 127 行定义。

2.18 FLASH_TypeDef结构体 参考

FLASH寄存器 (FLASH)

```
#include <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

- `_IOM uint32_t NWDATA`

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

- `_IOM uint32_t NRDATA`

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

- `_IOM uint32_t NERASE`

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

- `_IOM 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

- `_IOM 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

2.18.1 详细描述

FLASH寄存器 (FLASH)

在文件 `lks32mc09x_reg.h` 第 132 行定义。

2.18.2 结构体成员变量说明

2.18.2.1 ADDR

`_IOM uint32_t ADDR`

Flash访问地址寄存器 [14:0] 地址寄存器

在文件 `lks32mc09x_reg.h` 第 135 行定义。

2.18.2.2 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 [lks32mc09x_reg.h](#) 第 134 行定义。

2.18.2.3 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 [lks32mc09x_reg.h](#) 第 148 行定义。

2.18.2.4 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 [lks32mc09x_reg.h](#) 第 150 行定义。

2.18.2.5 ERASE

--IOM uint32_t ERASE

Flash擦除寄存器 [31:0] 写入0x7654DCBA，触发擦除操作

在文件 [lks32mc09x_reg.h](#) 第 138 行定义。

2.18.2.6 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 [lks32mc09x_reg.h](#) 第 151 行定义.

2.18.2.7 NADDR

```
--IOM uint32_t NADDR
```

Flash NVR访问地址寄存器 [14:0] 地址寄存器

在文件 [lks32mc09x_reg.h](#) 第 143 行定义.

2.18.2.8 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 [lks32mc09x_reg.h](#) 第 142 行定义.

2.18.2.9 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 [lks32mc09x_reg.h](#) 第 146 行定义.

2.18.2.10 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 [lks32mc09x_reg.h](#) 第 147 行定义.

2.18.2.11 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 [lks32mc09x_reg.h](#) 第 145 行定义.

2.18.2.12 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 [lks32mc09x_reg.h](#) 第 144 行定义.

2.18.2.13 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 [lks32mc09x_reg.h](#) 第 139 行定义.

2.18.2.14 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 [lks32mc09x_reg.h](#) 第 137 行定义.

2.18.2.15 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 [lks32mc09x_reg.h](#) 第 140 行定义.

2.18.2.16 RESERVED

```
--IM uint32_t RESERVED
```

在文件 [lks32mc09x_reg.h](#) 第 149 行定义.

2.18.2.17 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 [lks32mc09x_reg.h](#) 第 141 行定义.

2.18.2.18 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 [lks32mc09x_reg.h](#) 第 136 行定义.

2.19 GPIO_InitTypeDef结构体 参考

GPIO功能配置结构体句柄

`#include <lks32mc09x_gpio.h>`

成员变量

- `uint32_t GPIO_Pin`
要配置的引脚
- `GPIO_Mode_TypeDef GPIO_Mode`
GPIO模式
- `GPIO_PuPd_TypeDef GPIO_PuPd`
上拉使能
- `uint32_t GPIO_PODEna`
开漏使能
- `uint32_t GPIO_PFLT`
滤波使能

2.19.1 详细描述

GPIO功能配置结构体句柄

在文件 [lks32mc09x_gpio.h](#) 第 38 行定义.

2.19.2 结构体成员变量说明

2.19.2.1 GPIO_Mode

`GPIO_Mode_TypeDef GPIO_Mode`

GPIO模式

在文件 [lks32mc09x_gpio.h](#) 第 41 行定义.

2.19.2.2 GPIO_PFLT

`uint32_t GPIO_PFLT`

滤波使能

在文件 [lks32mc09x_gpio.h](#) 第 44 行定义.

2.19.2.3 GPIO_Pin

`uint32_t GPIO_Pin`

要配置的引脚

在文件 [lks32mc09x_gpio.h](#) 第 40 行定义.

2.19.2.4 GPIO_PODEna

`uint32_t GPIO_PODEna`

开漏使能

在文件 [lks32mc09x_gpio.h](#) 第 43 行定义.

2.19.2.5 GPIO_PuPd

`GPIO_PuPd_TypeDef GPIO_PuPd`

上拉使能

在文件 [lks32mc09x_gpio.h](#) 第 42 行定义.

2.20 GPIO_TypeDef结构体 参考

GPIO寄存器

```
#include <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` **PCODE**

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

2.20.1 详细描述

GPIO寄存器

在文件 `lks32mc09x_reg.h` 第 156 行定义.

2.20.2 结构体成员变量说明

2.20.2.1 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 `lks32mc09x_reg.h` 第 171 行定义.

2.20.2.2 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 `lks32mc09x_reg.h` 第 170 行定义.

2.20.2.3 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 [lks32mc09x_reg.h](#) 第 166 行定义.

2.20.2.4 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 [lks32mc09x_reg.h](#) 第 167 行定义.

2.20.2.5 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 [lks32mc09x_reg.h](#) 第 168 行定义.

2.20.2.6 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 [lks32mc09x_reg.h](#) 第 169 行定义.

2.20.2.7 PDI

```
--IOM uint32_t PDI
```

GPIO0 输入数据 [15:0] GPIO x输入数据

在文件 [lks32mc09x_reg.h](#) 第 160 行定义.

2.20.2.8 PDO

```
--IOM uint32_t PDO
```

GPIO0 输出数据 [15:0] GPIO x输出数据

在文件 [lks32mc09x_reg.h](#) 第 161 行定义.

2.20.2.9 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 [lks32mc09x_reg.h](#) 第 165 行定义。

2.20.2.10 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 [lks32mc09x_reg.h](#) 第 158 行定义。

2.20.2.11 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 [lks32mc09x_reg.h](#) 第 164 行定义。

2.20.2.12 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 [lks32mc09x_reg.h](#) 第 159 行定义。

2.20.2.13 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 [lks32mc09x_reg.h](#) 第 162 行定义。

2.20.2.14 RESERVED

```
--IM uint32_t RESERVED
```

在文件 [lks32mc09x_reg.h](#) 第 163 行定义。

2.21 HALL_InitTypeDef结构体 参考

HALL初始化结构体句柄，初始化时请定义该句柄，并用其它地址来传参

```
#include <lks32mc09x_hall.h>
```

成员变量

- `uint32_t CountTH`
HALL计数器门限值，有效位为24位
- `uint16_t FilterLen`
滤波长度,0对应长度1,1023对应长度1024滤波长度
- `uint8_t ClockDivision`
分频 0~3:1 / 2 / 4 / 8
- `uint8_t Filter75.Ena`
使能第一级7/5滤波,高电平有效,ENABLE: 使能 DISABLE: 失能
- `uint8_t HALL.Ena`
使能HALL,高电平有效,ENABLE: 使能 DISABLE: 失能
- `uint8_t Capture_IRQ.Ena`
HALL信号变化中断使能,高电平有效,ENABLE: 使能 DISABLE: 失能
- `uint8_t OverFlow_IRQ.Ena`
HALL计数器溢出中断使能,高电平有效,ENABLE: 使能 DISABLE: 失能
- `uint8_t softIE`
软件中断使能,ENABLE: 使能 DISABLE: 失能

2.21.1 详细描述

HALL初始化结构体句柄，初始化时请定义该句柄，并用其它地址来传参

在文件 [lks32mc09x_hall.h](#) 第 10 行定义。

2.21.2 结构体成员变量说明

2.21.2.1 Capture_IRQ_Ena

uint8_t Capture_IRQ_Ena

HALL信号变化中断使能,高电平有效,ENABLE: 使能 DISABLE: 失能

在文件 [lks32mc09x_hall.h](#) 第 17 行定义.

2.21.2.2 ClockDivision

uint8_t ClockDivision

分频 0~3:1 /2 /4 /8

在文件 [lks32mc09x_hall.h](#) 第 14 行定义.

2.21.2.3 CountTH

uint32_t CountTH

HALL计数器门限值, 有效位为24位

在文件 [lks32mc09x_hall.h](#) 第 12 行定义.

2.21.2.4 Filter75_Ena

uint8_t Filter75_Ena

使能第一级7/5滤波,高电平有效,ENABLE: 使能 DISABLE: 失能

在文件 [lks32mc09x_hall.h](#) 第 15 行定义.

2.21.2.5 FilterLen

uint16_t FilterLen

滤波长度,0对应长度1,1023对应长度1024滤波长度

在文件 [lks32mc09x_hall.h](#) 第 13 行定义.

2.21.2.6 HALL_Ena

uint8_t HALL_Ena

使能HALL,高电平有效,ENABLE: 使能 DISABLE: 失能

在文件 [lks32mc09x_hall.h](#) 第 16 行定义.

2.21.2.7 OverFlow_IRQ_Ena

uint8_t OverFlow_IRQ_Ena

HALL计数器溢出中断使能,高电平有效,ENABLE: 使能 DISABLE: 失能

在文件 [lks32mc09x_hall.h](#) 第 18 行定义.

2.21.2.8 softIE

uint8_t softIE

软件中断使能 ,ENABLE: 使能 DISABLE: 失能

在文件 [lks32mc09x_hall.h](#) 第 19 行定义.

2.22 HALL_TypeDef结构体 参考

HALL0寄存器 (HALL0)

```
#include <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
- __IOM 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

2.22.1 详细描述

HALL0寄存器 (HALL0)

在文件 [lks32mc09x_reg.h](#) 第 176 行定义.

2.22.2 结构体成员变量说明

2.22.2.1 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 [lks32mc09x_reg.h](#) 第 178 行定义.

2.22.2.2 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 [lks32mc09x_reg.h](#) 第 182 行定义.

2.22.2.3 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 [lks32mc09x_reg.h](#) 第 179 行定义.

2.22.2.4 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 [lks32mc09x_reg.h](#) 第 181 行定义.

2.22.2.5 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 [lks32mc09x_reg.h](#) 第 180 行定义.

2.23 I2C_InitTypeDef结构体 参考

```
#include <lks32mc09x_i2c.h>
```

成员变量

- uint32_t **ADRCMP**
I2C 硬件地址比较使能开关
- uint32_t **ADDR**
从模式下，本节点的从机地址
- uint32_t **MST_MODE**
I2C 主模式使能信号
- uint32_t **SLV_MODE**
I2C 从模式使能信号
- uint32_t **BUSRT_EN**
I2C DMA传输使能
- uint32_t **BaudRate**
I2C 波特率
- uint32_t **IE**
I2C 中断使能信号
- uint32_t **TC_IE**
I2C 数据传输完成中断使能
- uint32_t **BUS_ERR_IE**
I2C 总线错误事件中断使能信号
- uint32_t **STOP_IE**
I2C STOP 事件中断使能信号
- uint32_t **BURST_NACK**
NACK 事件中断使能
- uint32_t **BURST_ADDR_CMP**
硬件地址匹配中断使能

2.23.1 详细描述

在文件 [lks32mc09x_i2c.h](#) 第 22 行定义.

2.23.2 结构体成员变量说明

2.23.2.1 ADDR

```
uint32_t ADDR
```

从模式下，本节点的从机地址

在文件 [lks32mc09x_i2c.h](#) 第 25 行定义.

2.23.2.2 ADRCMP

uint32_t ADRCMP

I2C 硬件地址比较使能开关

在文件 [lks32mc09x_i2c.h](#) 第 24 行定义.

2.23.2.3 BaudRate

uint32_t BaudRate

I2C 波特率

在文件 [lks32mc09x_i2c.h](#) 第 29 行定义.

2.23.2.4 BURST_ADDR_CMP

uint32_t BURST_ADDR_CMP

硬件地址匹配中断使能

在文件 [lks32mc09x_i2c.h](#) 第 35 行定义.

2.23.2.5 BURST_NACK

uint32_t BURST_NACK

NACK 事件中断使能

在文件 [lks32mc09x_i2c.h](#) 第 34 行定义.

2.23.2.6 BUS_ERR_IE

uint32_t BUS_ERR_IE

I2C 总线错误事件中断使能信号

在文件 [lks32mc09x_i2c.h](#) 第 32 行定义.

2.23.2.7 BUSRT_EN

uint32_t BUSRT_EN

I2C DMA传输使能

在文件 [lks32mc09x_i2c.h](#) 第 28 行定义.

2.23.2.8 IE

`uint32_t IE`

I2C 中断使能信号

在文件 [lks32mc09x_i2c.h](#) 第 30 行定义.

2.23.2.9 MST_MODE

`uint32_t MST_MODE`

I2C 主模式使能信号

在文件 [lks32mc09x_i2c.h](#) 第 26 行定义.

2.23.2.10 SLV_MODE

`uint32_t SLV_MODE`

I2C 从模式使能信号

在文件 [lks32mc09x_i2c.h](#) 第 27 行定义.

2.23.2.11 STOP_IE

`uint32_t STOP_IE`

I2C STOP 事件中断使能信号

在文件 [lks32mc09x_i2c.h](#) 第 33 行定义.

2.23.2.12 TC_IE

`uint32_t TC_IE`

I2C 数据传输完成中断使能

在文件 [lks32mc09x_i2c.h](#) 第 31 行定义.

2.24 I2C_TypeDef结构体 参考

I2C0寄存器 (I2C0)

```
#include <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
- `__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 = $B[3:0] + 1$

2.24.1 详细描述

I2C0寄存器 (I2C0)

在文件 `lks32mc09x_reg.h` 第 187 行定义.

2.24.2 结构体成员变量说明

2.24.2.1 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 `lks32mc09x_reg.h` 第 189 行定义.

2.24.2.2 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 `lks32mc09x_reg.h` 第 194 行定义.

2.24.2.3 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 [lks32mc09x_reg.h](#) 第 195 行定义。

2.24.2.4 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 [lks32mc09x_reg.h](#) 第 190 行定义。

2.24.2.5 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 [lks32mc09x_reg.h](#) 第 192 行定义。

2.24.2.6 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 [lks32mc09x_reg.h](#) 第 193 行定义。

2.24.2.7 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 [lks32mc09x_reg.h](#) 第 191 行定义。

2.25 iictmpval结构体 参考

成员变量

- volatile uint8_t * [txdata](#)
- volatile uint8_t [txdatap](#)
- volatile uint8_t * [rxdata](#)
- volatile uint8_t [rxdatap](#)
- volatile uint8_t [state](#)
- volatile uint8_t [nackflg](#)

2.25.1 详细描述

在文件 [lks32mc09x_i2c.c](#) 第 232 行定义.

2.25.2 结构体成员变量说明

2.25.2.1 nackflg

```
volatile uint8_t nackflg
```

在文件 [lks32mc09x_i2c.c](#) 第 239 行定义.

2.25.2.2 rxdata

```
volatile uint8_t* rxdata
```

在文件 [lks32mc09x_i2c.c](#) 第 236 行定义.

2.25.2.3 rxdatap

```
volatile uint8_t rxdatap
```

在文件 [lks32mc09x_i2c.c](#) 第 237 行定义.

2.25.2.4 state

```
volatile uint8_t state
```

在文件 [lks32mc09x_i2c.c](#) 第 238 行定义.

2.25.2.5 txdata

```
volatile uint8_t* txdata
```

在文件 [lks32mc09x_i2c.c](#) 第 234 行定义.

2.25.2.6 txdatap

```
volatile uint8_t txdatap
```

在文件 [lks32mc09x_i2c.c](#) 第 235 行定义.

2.26 IWDG_InitTypeDef结构体 参考

IWDG初始化结构体句柄

```
#include <lks32mc09x_iwdg.h>
```

成员变量

- `uint32_t DWK_EN`
深度休眠定时唤醒使能，0:禁用，1:使能
- `uint32_t WDG_EN`
独立看门狗使能，0:禁用，1:使能
- `uint32_t WTH`
看门狗定时唤醒门限值（21位计数器，但低8位恒为0）
- `uint32_t RTH`
看门狗超时复位门限值（21位计数器，但低8位恒为0）

2.26.1 详细描述

IWDG初始化结构体句柄

在文件 [lks32mc09x_iwdg.h](#) 第 9 行定义.

2.26.2 结构体成员变量说明

2.26.2.1 DWK_EN

```
uint32_t DWK_EN
```

深度休眠定时唤醒使能，0:禁用，1:使能

在文件 [lks32mc09x_iwdg.h](#) 第 11 行定义.

2.26.2.2 RTH

```
uint32_t RTH
```

看门狗超时复位门限值（21位计数器，但低8位恒为0）

在文件 [lks32mc09x_iwdg.h](#) 第 14 行定义.

2.26.2.3 WDG_EN

```
uint32_t WDG_EN
```

独立看门狗使能，0:禁用，1:使能

在文件 [lks32mc09x_iwdg.h](#) 第 12 行定义.

2.26.2.4 WTH

uint32_t WTH

看门狗定时唤醒门限值（21位计数器，但低8位恒为0）

在文件 [lks32mc09x_iwdg.h](#) 第 13 行定义。

2.27 IWDG_TypeDef结构体 参考

IWDG寄存器结构体句柄

```
#include <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_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

2.27.1 详细描述

IWDG寄存器结构体句柄

在文件 [lks32mc09x_reg.h](#) 第 367 行定义。

2.27.2 结构体成员变量说明

2.27.2.1 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 [lks32mc09x_reg.h](#) 第 370 行定义。

2.27.2.2 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 [lks32mc09x_reg.h](#) 第 371 行定义。

2.27.2.3 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 [lks32mc09x_reg.h](#) 第 374 行定义。

2.27.2.4 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 [lks32mc09x_reg.h](#) 第 369 行定义。

2.27.2.5 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 [lks32mc09x_reg.h](#) 第 373 行定义。

2.27.2.6 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 [lks32mc09x_reg.h](#) 第 372 行定义。

2.28 MCPWM_InitTypeDef结构体 参考

```
#include <lks32mc09x_mcpwm.h>
```

成员变量

- uint16_t EN
- uint32_t COUNT
- uint16_t CLK_DIV
- uint16_t TH
- uint16_t TR
- int16_t TMR0
- int16_t TMR1
- int16_t TMR2
- int16_t TMR3
- uint16_t DTHP
- uint16_t DTHN
- uint16_t FLT_DIV
- uint16_t IO0
- uint16_t IO1
- uint16_t IO2
- uint16_t IO3
- uint32_t AUEN
- uint32_t IE
- uint32_t EIE
- uint32_t RE
- uint32_t FAIL_IO
- uint32_t FAIL0
- uint32_t FAIL1

2.28.1 详细描述

在文件 [lks32mc09x_mcpwm.h](#) 第 5 行定义.

2.28.2 结构体成员变量说明

2.28.2.1 AUEN

uint32_t AUEN

在文件 [lks32mc09x_mcpwm.h](#) 第 24 行定义.

2.28.2.2 CLK_DIV

uint16_t CLK_DIV

在文件 [lks32mc09x_mcpwm.h](#) 第 10 行定义.

2.28.2.3 COUNT

uint32_t COUNT

在文件 [lks32mc09x_mcpwm.h](#) 第 9 行定义.

2.28.2.4 DTHN

uint16_t DTHN

在文件 [lks32mc09x_mcpwm.h](#) 第 18 行定义.

2.28.2.5 DTHP

uint16_t DTHP

在文件 [lks32mc09x_mcpwm.h](#) 第 17 行定义.

2.28.2.6 EIE

uint32_t EIE

在文件 [lks32mc09x_mcpwm.h](#) 第 26 行定义.

2.28.2.7 EN

uint16_t EN

在文件 [lks32mc09x_mcpwm.h](#) 第 8 行定义.

2.28.2.8 FAIL0

uint32_t FAIL0

在文件 [lks32mc09x_mcpwm.h](#) 第 29 行定义.

2.28.2.9 FAIL1

uint32_t FAIL1

在文件 [lks32mc09x_mcpwm.h](#) 第 30 行定义.

2.28.2.10 FAIL_IO

uint32_t FAIL_IO

在文件 [lks32mc09x_mcpwm.h](#) 第 28 行定义.

2.28.2.11 FLT_DIV

uint16_t FLT_DIV

在文件 [lks32mc09x_mcpwm.h](#) 第 19 行定义.

2.28.2.12 IE

uint32_t IE

在文件 [lks32mc09x_mcpwm.h](#) 第 25 行定义.

2.28.2.13 IO0

uint16_t IO0

在文件 [lks32mc09x_mcpwm.h](#) 第 20 行定义.

2.28.2.14 IO1

uint16_t IO1

在文件 [lks32mc09x_mcpwm.h](#) 第 21 行定义.

2.28.2.15 IO2

uint16_t IO2

在文件 [lks32mc09x_mcpwm.h](#) 第 22 行定义.

2.28.2.16 IO3

uint16_t IO3

在文件 [lks32mc09x_mcpwm.h](#) 第 23 行定义.

2.28.2.17 RE

uint32_t RE

在文件 [lks32mc09x_mcpwm.h](#) 第 27 行定义.

2.28.2.18 TH

uint16_t TH

在文件 [lks32mc09x_mcpwm.h](#) 第 11 行定义.

2.28.2.19 TMR0

int16_t TMR0

在文件 [lks32mc09x_mcpwm.h](#) 第 13 行定义.

2.28.2.20 TMR1

int16_t TMR1

在文件 [lks32mc09x_mcpwm.h](#) 第 14 行定义.

2.28.2.21 TMR2

int16_t TMR2

在文件 [lks32mc09x_mcpwm.h](#) 第 15 行定义.

2.28.2.22 TMR3

int16_t TMR3

在文件 [lks32mc09x_mcpwm.h](#) 第 16 行定义.

2.28.2.23 TR

uint16_t TR

在文件 [lks32mc09x_mcpwm.h](#) 第 12 行定义.

2.29 MCPWM_TypeDef结构体 参考

MCPWM0寄存器 (MCPWM0)

```
#include <lks32mc09x_reg.h>
```

成员变量

- __IOM uint32_t [TH00](#)
 < MCPWM0结构体
- __IOM uint32_t [TH01](#)
 CH0 N 16位阈值 [15:0]MCPWM CH0.N比较门限值
- __IOM uint32_t [TH10](#)
 CH1 P 16位阈值 [15:0]MCPWM CH1.P比较门限值
- __IOM uint32_t [TH11](#)
 CH1 N 16位阈值 [15:0]MCPWM CH1.N比较门限值
- __IOM uint32_t [TH20](#)
 CH2 P 16位阈值 [15:0]MCPWM CH2.P比较门限值
- __IOM uint32_t [TH21](#)
 CH2 N 16位阈值 [15:0]MCPWM CH2.N比较门限值
- __IOM uint32_t [TH30](#)
 CH3 P 16位阈值 [15:0]MCPWM CH3.P比较门限值
- __IOM uint32_t [TH31](#)

- CH3 N 16位阈值 [15:0]MCPWM CH3.N比较门限值
 - __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
- `__IOM 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
- `__IOM 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

2.29.1 详细描述

MCPWM0寄存器 (MCPWM0)

在文件 `lks32mc09x_reg.h` 第 200 行定义.

2.29.2 结构体成员变量说明

2.29.2.1 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 `lks32mc09x_reg.h` 第 228 行定义.

2.29.2.2 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 `lks32mc09x_reg.h` 第 242 行定义.

2.29.2.3 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 `lks32mc09x_reg.h` 第 240 行定义.

2.29.2.4 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 [lks32mc09x_reg.h](#) 第 243 行定义.

2.29.2.5 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 [lks32mc09x_reg.h](#) 第 217 行定义.

2.29.2.6 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 [lks32mc09x_reg.h](#) 第 223 行定义.

2.29.2.7 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 [lks32mc09x_reg.h](#) 第 224 行定义.

2.29.2.8 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 [lks32mc09x_reg.h](#) 第 233 行定义.

2.29.2.9 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 [lks32mc09x_reg.h](#) 第 234 行定义.

2.29.2.10 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 [lks32mc09x_reg.h](#) 第 221 行定义。

2.29.2.11 FCNT

```
--IOM uint32_t FCNT
```

若MCPWM_FAIL0[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 [lks32mc09x_reg.h](#) 第 220 行定义。

2.29.2.12 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 [lks32mc09x_reg.h](#) 第 226 行定义。

2.29.2.13 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 [lks32mc09x_reg.h](#) 第 230 行定义。

2.29.2.14 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 [lks32mc09x_reg.h](#) 第 231 行定义.

2.29.2.15 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 [lks32mc09x_reg.h](#) 第 237 行定义.

2.29.2.16 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 [lks32mc09x_reg.h](#) 第 238 行定义.

2.29.2.17 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 [lks32mc09x_reg.h](#) 第 236 行定义.

2.29.2.18 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 [lks32mc09x_reg.h](#) 第 244 行定义.

2.29.2.19 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 [lks32mc09x_reg.h](#) 第 235 行定义.

2.29.2.20 RESERVED

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

在文件 [lks32mc09x_reg.h](#) 第 210 行定义.

2.29.2.21 RESERVED1

```
--IM uint32_t RESERVED1
```

在文件 [lks32mc09x_reg.h](#) 第 216 行定义.

2.29.2.22 RESERVED2

```
--IM uint32_t RESERVED2
```

在文件 [lks32mc09x_reg.h](#) 第 218 行定义.

2.29.2.23 RESERVED3

```
--IM uint32_t RESERVED3
```

在文件 [lks32mc09x_reg.h](#) 第 222 行定义.

2.29.2.24 RESERVED4

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

在文件 [lks32mc09x_reg.h](#) 第 225 行定义.

2.29.2.25 RESERVED5

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

在文件 [lks32mc09x_reg.h](#) 第 232 行定义.

2.29.2.26 RESERVED6

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

在文件 [lks32mc09x_reg.h](#) 第 239 行定义.

2.29.2.27 RESERVED7

```
--IM uint32_t RESERVED7
```

在文件 [lks32mc09x_reg.h](#) 第 241 行定义.

2.29.2.28 RESERVED8

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

在文件 [lks32mc09x_reg.h](#) 第 245 行定义.

2.29.2.29 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 [lks32mc09x_reg.h](#) 第 227 行定义.

2.29.2.30 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 [lks32mc09x_reg.h](#) 第 246 行定义.

2.29.2.31 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 [lks32mc09x_reg.h](#) 第 229 行定义.

2.29.2.32 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 [lks32mc09x_reg.h](#) 第 215 行定义.

2.29.2.33 TH00

```
--IOM uint32_t TH00
```

< MCPWM0结构体

CH0 P 16位阈值 [15:0]MCPWM CH0.P比较门限值

在文件 [lks32mc09x_reg.h](#) 第 202 行定义.

2.29.2.34 TH01

```
--IOM uint32_t TH01
```

CH0 N 16位阈值 [15:0]MCPWM CH0.N比较门限值

在文件 [lks32mc09x_reg.h](#) 第 203 行定义.

2.29.2.35 TH10

```
--IOM uint32_t TH10
```

CH1 P 16位阈值 [15:0]MCPWM CH1.P比较门限值

在文件 [lks32mc09x_reg.h](#) 第 204 行定义.

2.29.2.36 TH11

```
--IOM uint32_t TH11
```

CH1 N 16位阈值 [15:0]MCPWM CH1.N比较门限值

在文件 [lks32mc09x_reg.h](#) 第 205 行定义.

2.29.2.37 TH20

--IOM uint32_t TH20

CH2 P 16位阈值 [15:0]MCPWM CH2.P比较门限值

在文件 [lks32mc09x_reg.h](#) 第 206 行定义.

2.29.2.38 TH21

--IOM uint32_t TH21

CH2 N 16位阈值 [15:0]MCPWM CH2.N比较门限值

在文件 [lks32mc09x_reg.h](#) 第 207 行定义.

2.29.2.39 TH30

--IOM uint32_t TH30

CH3 P 16位阈值 [15:0]MCPWM CH3.P比较门限值

在文件 [lks32mc09x_reg.h](#) 第 208 行定义.

2.29.2.40 TH31

--IOM uint32_t TH31

CH3 N 16位阈值 [15:0]MCPWM CH3.N比较门限值

在文件 [lks32mc09x_reg.h](#) 第 209 行定义.

2.29.2.41 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 [lks32mc09x_reg.h](#) 第 211 行定义.

2.29.2.42 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 [lks32mc09x_reg.h](#) 第 212 行定义.

2.29.2.43 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 [lks32mc09x_reg.h](#) 第 213 行定义。

2.29.2.44 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 [lks32mc09x_reg.h](#) 第 214 行定义。

2.29.2.45 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 [lks32mc09x_reg.h](#) 第 219 行定义。

2.30 OPA.InitTypeDef结构体 参考

OPA初始化结构体句柄，初始化时请定义该句柄，并用其它地址来传参

```
#include <lks32mc09x_opa.h>
```

成员变量

- `uint32_t OPA_IT`
运放偏置电流调节 `OPA_IT_1_0` `OPA_IT_1_2` `OPA_IT_1_5` `OPA_IT_2_0`
- `uint32_t OPA_Gain`
运放增益 `OPA_GAIN_33` `OPA_GAIN_16` `OPA_GAIN_8` `OPA_GAIN_4`
- `uint32_t OPA_CLEna`
运放使能 `Enable` `Disable`

2.30.1 详细描述

OPA初始化结构体句柄，初始化时请定义该句柄，并用其它地址来传参

在文件 `lks32mc09x_opa.h` 第 13 行定义.

2.30.2 结构体成员变量说明

2.30.2.1 OPA_CLEna

`uint32_t OPA_CLEna`

运放使能 `Enable` `Disable`

在文件 `lks32mc09x_opa.h` 第 17 行定义.

2.30.2.2 OPA_Gain

`uint32_t OPA_Gain`

运放增益 `OPA_GAIN_33` `OPA_GAIN_16` `OPA_GAIN_8` `OPA_GAIN_4`

在文件 `lks32mc09x_opa.h` 第 16 行定义.

2.30.2.3 OPA_IT

`uint32_t OPA_IT`

运放偏置电流调节 `OPA_IT_1_0` `OPA_IT_1_2` `OPA_IT_1_5` `OPA_IT_2_0`

在文件 `lks32mc09x_opa.h` 第 15 行定义.

2.31 QEP_InitTypeDef结构体 参考

QEP初始化结构体定义

```
#include <lks32mc09x-qep.h>
```

成员变量

- `uint32_t Mode`
编码器模式选择
- `uint32_t FE_CNT_EN`
下降沿计数使能（上升沿始终计数），*ENABLE*使能 *DISABLE*失能
- `uint16_t TH`
*QEP*计数门限寄存器
- `uint32_t ClockDiv`
时钟分频
- `uint32_t Filter`
信号输入滤波，设置为1时滤波宽度是1*8个时钟，取值范围0-15
- `uint32_t IRQEna`
中断使能
- `uint32_t ZC`
Z信号配置

2.31.1 详细描述

*QEP*初始化结构体定义

在文件 `lks32mc09x_qep.h` 第 9 行定义.

2.31.2 结构体成员变量说明

2.31.2.1 ClockDiv

`uint32_t ClockDiv`

时钟分频

在文件 `lks32mc09x_qep.h` 第 14 行定义.

2.31.2.2 FE_CNT_EN

`uint32_t FE_CNT_EN`

下降沿计数使能（上升沿始终计数），*ENABLE*使能 *DISABLE*失能

在文件 `lks32mc09x_qep.h` 第 12 行定义.

2.31.2.3 Filter

`uint32_t Filter`

信号输入滤波，设置为1时滤波宽度是1*8个时钟，取值范围0-15

在文件 `lks32mc09x_qep.h` 第 15 行定义.

2.31.2.4 IRQEna

uint32_t IRQEna

中断使能

在文件 [lks32mc09x_qep.h](#) 第 16 行定义.

2.31.2.5 Mode

uint32_t Mode

编码器模式选择

在文件 [lks32mc09x_qep.h](#) 第 11 行定义.

2.31.2.6 TH

uint16_t TH

QEP计数门限寄存器

在文件 [lks32mc09x_qep.h](#) 第 13 行定义.

2.31.2.7 ZC

uint32_t ZC

Z信号配置

在文件 [lks32mc09x_qep.h](#) 第 17 行定义.

2.32 QEP_TypeDef结构体 参考

QEP0寄存器 (QEP0)

```
#include <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

2.32.1 详细描述

QEP0寄存器 (QEP0)

在文件 [lks32mc09x_reg.h](#) 第 251 行定义.

2.32.2 结构体成员变量说明

2.32.2.1 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 [lks32mc09x_reg.h](#) 第 253 行定义.

2.32.2.2 CNT

```
__IOM uint32_t CNT
```

编码器当前计数值 [15:0] QEP0计数值

在文件 [lks32mc09x_reg.h](#) 第 255 行定义.

2.32.2.3 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 [lks32mc09x_reg.h](#) 第 256 行定义.

2.32.2.4 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 [lks32mc09x_reg.h](#) 第 257 行定义.

2.32.2.5 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 [lks32mc09x_reg.h](#) 第 254 行定义。

2.33 SIF_InitTypeDef结构体 参考

SIF初始化结构体定义

```
#include <lks32mc09x_sif.h>
```

成员变量

- `uint8_t SIF_DONE`
传输完成后默认电平 1: 高电平 0: 低电平
- `uint8_t SIF_SYNC`
同步信号默认电平 1: 高电平 0: 低电平
- `uint8_t SIF_SYNC_PULSE`
同步信号是否有电平反转 1: 有 0: 无
- `uint8_t SIF_DONE_VLD`
有无结束信号 1: 有 0: 无
- `uint8_t SIF_SYNC_VLD`
有无同步信号 1: 有 0: 无
- `uint8_t SIF_RATIO`
数据占空比 1: 3: 1 0: 2: 1
- `uint8_t SIF_MSB`
数据传送顺序 1: 高字节在前 0: 低字节在前
- `uint8_t SIF_EN`
SIF模块使能, 高有效
- `uint16_t SIF_TOSC`
时基设置 $SIF\_TOSC + 1$ 333NS为单位 最大 $4096 * 333 = 1364US$
- `uint16_t SIF_TSTH1`
同步时长 $(SIF\_TSTH1 + 1) * SIF\_TOSC$ 最大 $1024 * SIF\_TOSC * 32$
- `uint8_t SIF_TDTH1`
结束信号时长 $(SIF\_TDTH1 + 1) MS$
- `uint8_t SIF_IRQ_IF`
SIF中断标志位 1: 有 0: 无
- `uint8_t SIF_DMA_EN`
DMA 传输使能 高有效
- `uint8_t SIF_IRQ_EN`
SIF中断使能 高有效

2.33.1 详细描述

SIF初始化结构体定义

在文件 [lks32mc09x_sif.h](#) 第 9 行定义.

2.33.2 结构体成员变量说明

2.33.2.1 SIF_DMA_EN

```
uint8_t SIF_DMA_EN
```

DMA 传输使能 高有效

在文件 [lks32mc09x_sif.h](#) 第 23 行定义.

2.33.2.2 SIF_DONE

```
uint8_t SIF_DONE
```

传输完成后默认电平 1: 高电平 0: 低电平

在文件 [lks32mc09x_sif.h](#) 第 11 行定义.

2.33.2.3 SIF_DONE_VLD

```
uint8_t SIF_DONE_VLD
```

有无结束信号 1: 有 0: 无

在文件 [lks32mc09x_sif.h](#) 第 14 行定义.

2.33.2.4 SIF_EN

```
uint8_t SIF_EN
```

SIF模块使能, 高有效

在文件 [lks32mc09x_sif.h](#) 第 18 行定义.

2.33.2.5 SIF_IRQ_EN

```
uint8_t SIF_IRQ_EN
```

SIF中断使能 高有效

在文件 [lks32mc09x_sif.h](#) 第 24 行定义.

2.33.2.6 SIF_IRQ_IF

uint8_t SIF_IRQ_IF

SIF中断标志位 1: 有 0: 无

在文件 [lks32mc09x_sif.h](#) 第 22 行定义.

2.33.2.7 SIF_MSB

uint8_t SIF_MSB

数据传送顺序 1: 高字节在前 0: 低字节在前

在文件 [lks32mc09x_sif.h](#) 第 17 行定义.

2.33.2.8 SIF_RATIO

uint8_t SIF_RATIO

数据占空比 1: 3: 1 0: 2: 1

在文件 [lks32mc09x_sif.h](#) 第 16 行定义.

2.33.2.9 SIF_SYNC

uint8_t SIF_SYNC

同步信号默认电平 1: 高电平 0: 低电平

在文件 [lks32mc09x_sif.h](#) 第 12 行定义.

2.33.2.10 SIF_SYNC_PULSE

uint8_t SIF_SYNC_PULSE

同步信号是否有电平反转 1: 有 0: 无

在文件 [lks32mc09x_sif.h](#) 第 13 行定义.

2.33.2.11 SIF_SYNC_VLD

uint8_t SIF_SYNC_VLD

有无同步信号 1: 有 0: 无

在文件 [lks32mc09x_sif.h](#) 第 15 行定义.

2.33.2.12 SIF_TDTH1

uint8_t SIF_TDTH1

结束信号时长 (SIF_TDTH1 + 1) MS

在文件 [lks32mc09x_sif.h](#) 第 21 行定义.

2.33.2.13 SIF_TOSC

uint16_t SIF_TOSC

时基设置 SIF_TOSC+1 333NS为单位 最大 $4096 * 333 = 1364US$

在文件 [lks32mc09x_sif.h](#) 第 19 行定义.

2.33.2.14 SIF_TSTH1

uint16_t SIF_TSTH1

同步时长 (SIF_TSTH1 + 1) * SIF_TOSC 最大 $1024 * SIF\_TOSC * 32$

在文件 [lks32mc09x_sif.h](#) 第 20 行定义.

2.34 SIF_TypeDef结构体 参考

SIF0寄存器 (SIF0)

```
#include <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 [TDTH1](#)
 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

2.34.1 详细描述

SIF0寄存器 (SIF0)

在文件 [lks32mc09x_reg.h](#) 第 262 行定义.

2.34.2 结构体成员变量说明

2.34.2.1 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 [lks32mc09x_reg.h](#) 第 264 行定义.

2.34.2.2 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 [lks32mc09x_reg.h](#) 第 268 行定义.

2.34.2.3 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 [lks32mc09x_reg.h](#) 第 267 行定义.

2.34.2.4 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 [lks32mc09x_reg.h](#) 第 265 行定义.

2.34.2.5 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 [lks32mc09x_reg.h](#) 第 266 行定义.

2.34.2.6 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 [lks32mc09x_reg.h](#) 第 269 行定义.

2.35 SPI_InitTypeDef结构体 参考

SPI模块初始化结构体

```
#include <lks32mc09x_spi.h>
```

成员变量

- [SPI_Duplex Duplex](#)
数据传输模式选择
- [uint8_t CS](#)
从设备下片选信号来源: 0, 恒为0; 1, 来源于主设备
- [SPI_Mode Mode](#)
主从模式选择
- [uint8_t CPHA](#)
相位选择
- [uint8_t CPOL](#)
极性选择
- [uint8_t ENDIAN](#)
传输顺序选择: 0对应MSB高位先传输; 1对应LSB低位先传输
- [uint8_t EN](#)
模块使能
- [uint8_t IRQEna](#)
中断使能
- [uint8_t Trig](#)
传输触发方式选择: 1, 外部触发; 0, 内部自动执行 (仅主模式有效)
- [uint8_t TRANS.MODE](#)
数据搬运方式选择: 0, DMA搬运; 1, MCU搬运
- [uint16_t BaudRate](#)
波特率设置, 取值范围3-4095
- [uint8_t ByteLength](#)
数据长度

2.35.1 详细描述

SPI模块初始化结构体

在文件 [lks32mc09x_spi.h](#) 第 28 行定义.

2.35.2 结构体成员变量说明

2.35.2.1 BaudRate

`uint16_t BaudRate`

波特率设置，取值范围3-4095

在文件 [lks32mc09x_spi.h](#) 第 40 行定义.

2.35.2.2 ByteLength

`uint8_t ByteLength`

数据长度

在文件 [lks32mc09x_spi.h](#) 第 41 行定义.

2.35.2.3 CPHA

`uint8_t CPHA`

相位选择

在文件 [lks32mc09x_spi.h](#) 第 33 行定义.

2.35.2.4 CPOL

`uint8_t CPOL`

极性选择

在文件 [lks32mc09x_spi.h](#) 第 34 行定义.

2.35.2.5 CS

`uint8_t CS`

从设备下片选信号来源: 0, 恒为0; 1, 来源于主设备

在文件 [lks32mc09x_spi.h](#) 第 31 行定义.

2.35.2.6 Duplex

[SPI_Duplex](#) `Duplex`

数据传输模式选择

在文件 [lks32mc09x_spi.h](#) 第 30 行定义.

2.35.2.7 EN

uint8_t EN

模块使能

在文件 [lks32mc09x_spi.h](#) 第 36 行定义.

2.35.2.8 ENDIAN

uint8_t ENDIAN

传输顺序选择: 0对应MSB高位先传输; 1对应LSB低位先传输

在文件 [lks32mc09x_spi.h](#) 第 35 行定义.

2.35.2.9 IRQEna

uint8_t IRQEna

中断使能

在文件 [lks32mc09x_spi.h](#) 第 37 行定义.

2.35.2.10 Mode

[SPI_Mode](#) Mode

主从模式选择

在文件 [lks32mc09x_spi.h](#) 第 32 行定义.

2.35.2.11 TRANS_MODE

uint8_t TRANS_MODE

数据搬运方式选择: 0, DMA搬运; 1, MCU搬运

在文件 [lks32mc09x_spi.h](#) 第 39 行定义.

2.35.2.12 Trig

uint8_t Trig

传输触发方式选择: 1, 外部触发; 0, 内部自动执行 (仅主模式有效)

在文件 [lks32mc09x_spi.h](#) 第 38 行定义.

2.36 SPI_TypeDef结构体 参考

SPI0寄存器 (SPI0)

```
#include <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

2.36.1 详细描述

SPI0寄存器 (SPI0)

在文件 [lks32mc09x_reg.h](#) 第 274 行定义.

2.36.2 结构体成员变量说明

2.36.2.1 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 [lks32mc09x_reg.h](#) 第 276 行定义.

2.36.2.2 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 [lks32mc09x_reg.h](#) 第 278 行定义.

2.36.2.3 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 [lks32mc09x_reg.h](#) 第 277 行定义.

2.36.2.4 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 [lks32mc09x_reg.h](#) 第 280 行定义.

2.36.2.5 SIZE

```
--IOM uint32_t SIZE
```

SPI 传输位数 [4:0] 字节长度寄存器

在文件 [lks32mc09x_reg.h](#) 第 281 行定义.

2.36.2.6 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 [lks32mc09x_reg.h](#) 第 279 行定义.

2.37 SYS_TypeDef结构体 参考

SYS寄存器 (SYS)

```
#include <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
- `_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
- `_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
- `_IM 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
- `_IM 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

- __IM 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
- __IM 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] HAL0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] HAL0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**

2.37.1 详细描述

SYS寄存器 (SYS)

在文件 [lks32mc09x_reg.h](#) 第 286 行定义.

2.37.2 结构体成员变量说明

2.37.2.1 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 [lks32mc09x_reg.h](#) 第 288 行定义.

2.37.2.2 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 [lks32mc09x_reg.h](#) 第 302 行定义.

2.37.2.3 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 [lks32mc09x_reg.h](#) 第 304 行定义.

2.37.2.4 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 [lks32mc09x_reg.h](#) 第 305 行定义.

2.37.2.5 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 [lks32mc09x_reg.h](#) 第 303 行定义.

2.37.2.6 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 [lks32mc09x_reg.h](#) 第 306 行定义.

2.37.2.7 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 [lks32mc09x_reg.h](#) 第 307 行定义.

2.37.2.8 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 [lks32mc09x_reg.h](#) 第 301 行定义.

2.37.2.9 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 [lks32mc09x_reg.h](#) 第 290 行定义.

2.37.2.10 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 [lks32mc09x_reg.h](#) 第 289 行定义.

2.37.2.11 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 [lks32mc09x_reg.h](#) 第 292 行定义.

2.37.2.12 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 [lks32mc09x_reg.h](#) 第 293 行定义.

2.37.2.13 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 [lks32mc09x_reg.h](#) 第 294 行定义.

2.37.2.14 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 [lks32mc09x_reg.h](#) 第 295 行定义.

2.37.2.15 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 [lks32mc09x_reg.h](#) 第 296 行定义.

2.37.2.16 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 [lks32mc09x_reg.h](#) 第 297 行定义.

2.37.2.17 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 [lks32mc09x_reg.h](#) 第 299 行定义.

2.37.2.18 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 [lks32mc09x_reg.h](#) 第 309 行定义.

2.37.2.19 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 [lks32mc09x_reg.h](#) 第 313 行定义.

2.37.2.20 CLK_DIV1

```
--IOM uint32_t CLK_DIV1
```

在文件 [lks32mc09x_reg.h](#) 第 314 行定义.

2.37.2.21 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 [lks32mc09x_reg.h](#) 第 315 行定义.

2.37.2.22 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 [lks32mc09x_reg.h](#) 第 316 行定义.

2.37.2.23 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 [lks32mc09x_reg.h](#) 第 311 行定义.

2.37.2.24 FLSE

```
--IOM uint32_t FLSE
```

在文件 [lks32mc09x_reg.h](#) 第 326 行定义.

2.37.2.25 FLSP

```
--IOM uint32_t FLSP
```

在文件 [lks32mc09x_reg.h](#) 第 327 行定义.

2.37.2.26 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 [lks32mc09x_reg.h](#) 第 322 行定义.

2.37.2.27 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 [lks32mc09x_reg.h](#) 第 310 行定义.

2.37.2.28 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 [lks32mc09x_reg.h](#) 第 321 行定义.

2.37.2.29 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 [lks32mc09x_reg.h](#) 第 291 行定义.

2.37.2.30 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 [lks32mc09x_reg.h](#) 第 319 行定义.

2.37.2.31 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 [lks32mc09x_reg.h](#) 第 323 行定义.

2.37.2.32 RESERVED

```
--IM uint32_t RESERVED
```

在文件 [lks32mc09x_reg.h](#) 第 298 行定义.

2.37.2.33 RESERVED1

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

在文件 [lks32mc09x_reg.h](#) 第 300 行定义.

2.37.2.34 RESERVED2

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

在文件 [lks32mc09x_reg.h](#) 第 308 行定义.

2.37.2.35 RESERVED3

```
--IM uint32_t RESERVED3
```

在文件 [lks32mc09x_reg.h](#) 第 312 行定义.

2.37.2.36 RESERVED41

```
--IM uint32_t RESERVED41
```

在文件 [lks32mc09x_reg.h](#) 第 317 行定义.

2.37.2.37 RESERVED5

```
--IM uint32_t RESERVED5
```

在文件 [lks32mc09x_reg.h](#) 第 320 行定义.

2.37.2.38 RESERVED6

```
--IM uint32_t RESERVED6
```

在文件 [lks32mc09x_reg.h](#) 第 324 行定义.

2.37.2.39 RESERVED7

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

在文件 [lks32mc09x_reg.h](#) 第 325 行定义.

2.37.2.40 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 [lks32mc09x_reg.h](#) 第 318 行定义.

2.38 TIMER_TimerInitTypeDef结构体 参考

Timer初始化结构体

```
#include <lks32mc09x_timer.h>
```

成员变量

- uint8_t **EN**
定时器使能, *ENABLE*使能, *DISABLE*失能, 影响*timer*的*cnt*是否计数
- uint32_t **CAP1_CLR_EN**
当发生*CAP1*捕获事件时, 清零 *Timer* 计数器, 高有效
- uint32_t **CAP0_CLR_EN**
当发生*CAP0*捕获事件时, 清零 *Timer* 计数器, 高有效
- uint32_t **ONE_TRIG**
单次发送模式, 此位需要在*Timer*比较模式下使用, 且对应定时器使能需设置为0
- uint32_t **CENTER**
中心计数模式使能, 0:*Timer*向上从0计数至*TH*, 然后回0, 或*Timer*向下从*TH*计数至0, 然后回到*TH*, 1:*Timer*向上从0计数至*TH*, 然后向下计数至0
- uint32_t **DIR**
0:0->*TH* 递增计数, 1: *TH*->递减计数
- uint32_t **ClockDiv**
*Timer*计数器频率配置, 0:1分频, 1:2分频, 2:4分频, 3:8分频, 4:16分频, 5:32分频, 6:64分频, 7:128分频
- uint32_t **ETON**
*Timer*计数器计数外部启动使能, 0: 自动运行, 1:外部事件触发计数
- uint32_t **GATE_EN**
*Timer*暂停使能, 0: 不暂停 1:当外部信号为低时, *Timer*暂停计数, 外部信号根据*TIMER1_EVT.EVT_SRC* 进行选择
- uint32_t **RL_EN**
*Timer*重装使能, 0: 禁用外部事件重装, 1: 使能外部事件重装
- uint32_t **XCLK_EN**
*Timer*时钟源, 0: 芯片内部时钟, 1:外部时钟
- uint32_t **SRC1**
Timer 捕获模式通道1信号来源, 0:*Timer*通道0输入信号, 1:*Timer*通道1输入信号, 2:*CLU0*输出信号, 3:*CLU1*输出信号, 4:*CLU2*输出信号, 5:*CLU3*输出信号, 6:比较器0输出信号, 7:比较器1输出信号, 8:比较器2输出信号, 9:*Timer*通道0和1的异或信号
- uint32_t **CH1Output**
Timer 通道1在比较模式下的输出极性控制, 当计数器 *CNT*<*CMP1*时的输出值, 0: 低电平, 1: 高电平
- uint32_t **CH1_WorkMode**
Timer 通道1的工作模式选择, 0: 比较模式, 1: 捕获模式
- uint32_t **CH1_FE_CAP_EN**
*Timer*通道1下降沿捕获事件使能, 1:使能, 0:关闭
- uint32_t **CH1_RE_CAP_EN**
*Timer*通道1上升沿捕获事件使能, 1:使能, 0:关闭
- uint32_t **SRC0**
Timer 捕获模式通道0信号来源
- uint32_t **CH0Output**
Timer 通道0在比较模式下的输出极性控制, 当计数器 *CNT*<*CMP1*时的输出值, 0: 低电平, 1: 高电平
- uint32_t **CH0_WorkMode**
Timer 通道0的工作模式选择, 0: 比较模式, 1: 捕获模式
- uint32_t **CH0_FE_CAP_EN**
*Timer*通道0下降沿捕获事件使能, 1:使能, 0:关闭
- uint32_t **CH0_RE_CAP_EN**
*Timer*通道0上升沿捕获事件使能, 1:使能, 0:关闭
- uint32_t **TH**
*Timer*计数器计数门限
- uint32_t **CNT**
*Timer*计数器当前计数值

- `uint32_t CHN0`
通道0比较捕获寄存器
- `uint32_t CHN1`
通道1比较捕获寄存器
- `uint32_t EVT`
*Timer*外部事件选择
- `uint32_t FLT`
通道0/1信号滤波宽度选择。取值范围0~255,滤波公式= $Timer.FLT \times 8 \times$ 定时器时钟
- `uint32_t IE`
中断使能
- `uint32_t CMP0_TRIGGER_MODE`
0: 通道0比较事件在CNT递增或递减均产生*adc*触发信号, 1递增时触发, 2递减时触发
- `uint32_t CMP1_TRIGGER_MODE`
0: 通道1比较事件在CNT递增或递减均产生*adc*触发信号, 1递增时触发, 2递减时触发
- `uint32_t SHADOW`
0: 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时进位仅更新预装载值, 当*Timer*发生过零事件时才将预装载值写入影子寄存器
- `uint32_t CH1_DEFAULT`
定时器0发生*Fail*事件MOE清零后, CH1通道输出值
- `uint32_t CH0_DEFAULT`
定时器0发生*Fail*事件MOE清零后, CH0通道输出值
- `uint32_t HALT_PRT`
MCU进入HALT状态, *Timer*输出值选择。1:正常输出; 0:输出CH1_DEFAULT和CH0_DEFAULT。
- `uint32_t FAIL_SEL`
FAIL信号选择, 0: *TIMER0_FAIL*, 来自GPIO, 1: *CLU0*输出
- `uint32_t FAIL_POL`
FAIL信号极性, 0: 高电平FAIL, 1: 低电平FAIL
- `uint32_t FAIL_EN`
FAIL信号使能, 0: 禁用FAIL, 1: 使能FAIL
- `uint32_t MOE`
*TIMER*通道0/1输出使能 1:输出正常信号 0:输出CH0_DEFAULT和CH1_DEFAULT默认值

2.38.1 详细描述

*Timer*初始化结构体

在文件 `lks32mc09x_timer.h` 第 9 行定义.

2.38.2 结构体成员变量说明

2.38.2.1 CAP0_CLR_EN

`uint32_t CAP0_CLR_EN`

当发生CAP0捕获事件时, 清零 *Timer* 计数器, 高有效

在文件 `lks32mc09x_timer.h` 第 13 行定义.

2.38.2.2 CAP1_CLR_EN

uint32_t CAP1_CLR_EN

当发生CAP1捕获事件时，清零 Timer 计数器，高有效

在文件 [lks32mc09x_timer.h](#) 第 12 行定义.

2.38.2.3 CENTER

uint32_t CENTER

中心计数模式使能,0:Timer向上从0计数至TH，然后回0，或Timer向下从TH计数至0，然后回到TH，1↔:Timer向上从0计数至TH，然后向下计数至0

在文件 [lks32mc09x_timer.h](#) 第 15 行定义.

2.38.2.4 CH0_DEFAULT

uint32_t CH0_DEFAULT

定时器0发生Fail事件MOE清零后，CH0通道输出值

在文件 [lks32mc09x_timer.h](#) 第 44 行定义.

2.38.2.5 CH0_FE_CAP_EN

uint32_t CH0_FE_CAP_EN

Timer通道0下升沿捕获事件使能,1:使能,0:关闭

在文件 [lks32mc09x_timer.h](#) 第 30 行定义.

2.38.2.6 CH0_RE_CAP_EN

uint32_t CH0_RE_CAP_EN

Timer通道0上升沿捕获事件使能,1:使能,0:关闭

在文件 [lks32mc09x_timer.h](#) 第 31 行定义.

2.38.2.7 CH0_WorkMode

uint32_t CH0_WorkMode

Timer 通道0的工作模式选择,0: 比较模式 ,1: 捕获模式

在文件 [lks32mc09x_timer.h](#) 第 29 行定义.

2.38.2.8 CH0Output

uint32_t CH0Output

Timer 通道0在比较模式下的输出极性控制，当计数器 $CNT < CMP1$ 时的输出值，0：低电平，1：高电平

在文件 [lks32mc09x_timer.h](#) 第 28 行定义.

2.38.2.9 CH1_DEFAULT

uint32_t CH1_DEFAULT

定时器0发生Fail事件MOE清零后，CH1通道输出值

在文件 [lks32mc09x_timer.h](#) 第 43 行定义.

2.38.2.10 CH1_FE_CAP_EN

uint32_t CH1_FE_CAP_EN

Timer通道1下降沿捕获事件使能,1:使能,0:关闭

在文件 [lks32mc09x_timer.h](#) 第 25 行定义.

2.38.2.11 CH1_RE_CAP_EN

uint32_t CH1_RE_CAP_EN

Timer通道1上升沿捕获事件使能,1:使能,0:关闭

在文件 [lks32mc09x_timer.h](#) 第 26 行定义.

2.38.2.12 CH1_WorkMode

uint32_t CH1_WorkMode

Timer 通道1的工作模式选择,0: 比较模式 ,1: 捕获模式

在文件 [lks32mc09x_timer.h](#) 第 24 行定义.

2.38.2.13 CH1Output

uint32_t CH1Output

Timer 通道1在比较模式下的输出极性控制，当计数器 $CNT < CMP1$ 时的输出值，0：低电平，1：高电平

在文件 [lks32mc09x_timer.h](#) 第 23 行定义.

2.38.2.14 CHN0

uint32_t CHN0

通道0比较捕获寄存器

在文件 [lks32mc09x_timer.h](#) 第 34 行定义.

2.38.2.15 CHN1

uint32_t CHN1

通道1比较捕获寄存器

在文件 [lks32mc09x_timer.h](#) 第 35 行定义.

2.38.2.16 ClockDiv

uint32_t ClockDiv

Timer计数器频率配置,0:1分频,1:2分频,2:4分频,3:8分频,4:16分频,5:32分频,6:64分频,7:128分频

在文件 [lks32mc09x_timer.h](#) 第 17 行定义.

2.38.2.17 CMP0.TRIGGER_MODE

uint32_t CMP0.TRIGGER_MODE

0: 通道0比较事件在CNT递增或递减均产生adc触发信号, 1递增时触发, 2递减时触发

在文件 [lks32mc09x_timer.h](#) 第 40 行定义.

2.38.2.18 CMP1.TRIGGER_MODE

uint32_t CMP1.TRIGGER_MODE

0: 通道1比较事件在CNT递增或递减均产生adc触发信号, 1递增时触发, 2递减时触发

在文件 [lks32mc09x_timer.h](#) 第 41 行定义.

2.38.2.19 CNT

uint32_t CNT

Timer计数器当前计数值

在文件 [lks32mc09x_timer.h](#) 第 33 行定义.

2.38.2.20 DIR

uint32_t DIR

0:0->TH 递增计数, 1: TH->递减计数

在文件 [lks32mc09x_timer.h](#) 第 16 行定义.

2.38.2.21 EN

uint8_t EN

定时器使能, ENABLE使能, DISABLE失能, 影响timer的cnt是否计数

在文件 [lks32mc09x_timer.h](#) 第 11 行定义.

2.38.2.22 ETON

uint32_t ETON

Timer计数器计数外部启动使能,0: 自动运行,1:外部事件触发计数

在文件 [lks32mc09x_timer.h](#) 第 18 行定义.

2.38.2.23 EVT

uint32_t EVT

Timer外部事件选择

在文件 [lks32mc09x_timer.h](#) 第 36 行定义.

2.38.2.24 FAIL_EN

uint32_t FAIL_EN

FAIL 信号使能,0: 禁用 FAIL,1: 使能 FAIL

在文件 [lks32mc09x_timer.h](#) 第 48 行定义.

2.38.2.25 FAIL_POL

uint32_t FAIL_POL

FAIL信号极性,0: 高电平FAIL,1: 低电平FAIL

在文件 [lks32mc09x_timer.h](#) 第 47 行定义.

2.38.2.26 FAIL_SEL

```
uint32_t FAIL_SEL
```

FAIL信号选择, 0: TIMER0_FAIL, 来自GPIO, 1: CLU0输出

在文件 [lks32mc09x_timer.h](#) 第 46 行定义.

2.38.2.27 FLT

```
uint32_t FLT
```

通道0/1信号滤波宽度选择。取值范围0~255,滤波公式=Timer_FLT*8*定时器时钟

在文件 [lks32mc09x_timer.h](#) 第 37 行定义.

2.38.2.28 GATE_EN

```
uint32_t GATE_EN
```

Timer暂停使能, 0: 不暂停 1:当外部信号为低时, Timer暂停计数, 外部信号根据TIMER1_EVT.EVT_SRC进行选择

在文件 [lks32mc09x_timer.h](#) 第 19 行定义.

2.38.2.29 HALT_PRT

```
uint32_t HALT_PRT
```

MCU 进入HALT状态, Timer输出值选择。1:正常输出; 0:输出 CH1_DEFAULT 和 CH0_DEFAULT。

在文件 [lks32mc09x_timer.h](#) 第 45 行定义.

2.38.2.30 IE

```
uint32_t IE
```

中断使能

在文件 [lks32mc09x_timer.h](#) 第 38 行定义.

2.38.2.31 MOE

```
uint32_t MOE
```

TIMER 通道 0/1 输出使能 1:输出正常信号 0:输出 CH0_DEFAULT 和 CH1_DEFAULT 默认值

在文件 [lks32mc09x_timer.h](#) 第 49 行定义.

2.38.2.32 ONE_TRIG

```
uint32_t ONE_TRIG
```

单次发送模式，此位需要在Timer比较模式下使用，且对应定时器使能需设置为0

在文件 [lks32mc09x_timer.h](#) 第 14 行定义.

2.38.2.33 RL_EN

```
uint32_t RL_EN
```

Timer重装使能,0: 禁用外部事件重装, 1: 使能外部事件重装

在文件 [lks32mc09x_timer.h](#) 第 20 行定义.

2.38.2.34 SHADOW

```
uint32_t SHADOW
```

0: 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

在文件 [lks32mc09x_timer.h](#) 第 42 行定义.

2.38.2.35 SRC0

```
uint32_t SRC0
```

Timer 捕获模式通道0信号来源

在文件 [lks32mc09x_timer.h](#) 第 27 行定义.

2.38.2.36 SRC1

```
uint32_t SRC1
```

Timer 捕获模式通道1信号来源,0:Timer通道0输入信号,1:Timer通道1输入信号,2:CLU0输出信号,3:CLU1输出信号,4:CLU2输出信号,5:CLU3输出信号,6:比较器0输出信号,7:比较器1输出信号,8:比较器2输出信号,9:Timer通道0和1的异或信号

在文件 [lks32mc09x_timer.h](#) 第 22 行定义.

2.38.2.37 TH

```
uint32_t TH
```

Timer计数器计数门限

在文件 [lks32mc09x_timer.h](#) 第 32 行定义.

2.38.2.38 XCLK_EN

uint32_t XCLK_EN

Timer时钟源,0: 芯片内部时钟,1:外部时钟

在文件 lks32mc09x_timer.h 第 21 行定义.

2.39 TIMER_TypeDef结构体 参考

TIMER0寄存器 (TIMER0)

```
#include <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] TIMER0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

2.39.1 详细描述

TIMER0寄存器 (TIMER0)

在文件 [lks32mc09x_reg.h](#) 第 332 行定义.

2.39.2 结构体成员变量说明

2.39.2.1 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 [lks32mc09x_reg.h](#) 第 334 行定义.

2.39.2.2 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 [lks32mc09x_reg.h](#) 第 335 行定义.

2.39.2.3 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 [lks32mc09x_reg.h](#) 第 336 行定义.

2.39.2.4 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 [lks32mc09x_reg.h](#) 第 340 行定义.

2.39.2.5 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 [lks32mc09x_reg.h](#) 第 341 行定义.

2.39.2.6 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 [lks32mc09x_reg.h](#) 第 339 行定义.

2.39.2.7 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 [lks32mc09x_reg.h](#) 第 342 行定义.

2.39.2.8 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 [lks32mc09x_reg.h](#) 第 343 行定义.

2.39.2.9 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 [lks32mc09x_reg.h](#) 第 344 行定义.

2.39.2.10 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 [lks32mc09x_reg.h](#) 第 345 行定义.

2.39.2.11 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 [lks32mc09x_reg.h](#) 第 346 行定义.

2.39.2.12 RESERVED

```
--IM uint32_t RESERVED
```

在文件 [lks32mc09x_reg.h](#) 第 337 行定义.

2.39.2.13 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 [lks32mc09x_reg.h](#) 第 338 行定义.

2.40 UART_InitTypeDef结构体 参考

UART 初始化结构体定义

```
#include <lks32mc09x_uart.h>
```

成员变量

- uint32_t [BaudRate](#)
波特率
- uint8_t [WordLength](#)
数据长度
- uint8_t [StopBits](#)
停止位长度
- uint8_t [FirstSend](#)
发送数据大小端设置
- [UART_ParityMode](#) [ParityMode](#)
奇偶校验
- uint8_t [MultiDropEna](#)
使能*Multi-drop*, 0:Disable 1:Enable
- uint8_t [Bit9Value](#)
*Multi-drop Master*模式时, 第9个数据位值
- uint8_t [DUPLEX](#)
双工选择, 0:全双工 1:半双工

- `uint16_t Match485Addr`
用作485通信时的匹配地址
- `uint16_t IRQEna`
中断使能寄存器
- `uint8_t DMARE`
DMA 请求使能
- `uint8_t RXD_INV`
接收电平取反
- `uint8_t TXD_INV`
发送电平取反
- `uint8_t ABD_EN`
LIN波特率自适应使能
- `uint8_t LBDL`
LIN同步段检测长度 0:10bits 1:11bits
- `uint8_t LIN_EN`
LIN 模式使能
- `uint8_t CK_EN`
数据校验开关

2.40.1 详细描述

UART 初始化结构体定义

在文件 `lks32mc09x_uart.h` 第 15 行定义.

2.40.2 结构体成员变量说明

2.40.2.1 ABD_EN

```
uint8_t ABD_EN
```

LIN波特率自适应使能

在文件 `lks32mc09x_uart.h` 第 30 行定义.

2.40.2.2 BaudRate

```
uint32_t BaudRate
```

波特率

在文件 `lks32mc09x_uart.h` 第 17 行定义.

2.40.2.3 Bit9Value

```
uint8_t Bit9Value
```

Multi-drop Master模式时, 第9个数据位值

在文件 `lks32mc09x_uart.h` 第 23 行定义.

2.40.2.4 CK_EN

uint8_t CK_EN

数据校验开关

在文件 [lks32mc09x_uart.h](#) 第 33 行定义.

2.40.2.5 DMARE

uint8_t DMARE

DMA 请求使能

在文件 [lks32mc09x_uart.h](#) 第 27 行定义.

2.40.2.6 DUPLEX

uint8_t DUPLEX

双工选择, 0:全双工 1:半双工

在文件 [lks32mc09x_uart.h](#) 第 24 行定义.

2.40.2.7 FirstSend

uint8_t FirstSend

发送数据大小端设置

在文件 [lks32mc09x_uart.h](#) 第 20 行定义.

2.40.2.8 IRQEna

uint16_t IRQEna

中断使能寄存器

在文件 [lks32mc09x_uart.h](#) 第 26 行定义.

2.40.2.9 LBDL

uint8_t LBDL

LIN同步段检测长度 0:10bits 1:11bits

在文件 [lks32mc09x_uart.h](#) 第 31 行定义.

2.40.2.10 LIN_EN

uint8_t LIN_EN

LIN 模式使能

在文件 [lks32mc09x_uart.h](#) 第 32 行定义.

2.40.2.11 Match485Addr

uint16_t Match485Addr

用作485通信时的匹配地址

在文件 [lks32mc09x_uart.h](#) 第 25 行定义.

2.40.2.12 MultiDropEna

uint8_t MultiDropEna

使能Multi-drop, 0:Disable 1:Enable

在文件 [lks32mc09x_uart.h](#) 第 22 行定义.

2.40.2.13 ParityMode

UART_ParityMode ParityMode

奇偶校验

在文件 [lks32mc09x_uart.h](#) 第 21 行定义.

2.40.2.14 RXD_INV

uint8_t RXD_INV

接收电平取反

在文件 [lks32mc09x_uart.h](#) 第 28 行定义.

2.40.2.15 StopBits

uint8_t StopBits

停止位长度

在文件 [lks32mc09x_uart.h](#) 第 19 行定义.

2.40.2.16 TXD_INV

uint8_t TXD_INV

发送电平取反

在文件 [lks32mc09x_uart.h](#) 第 29 行定义.

2.40.2.17 WordLength

uint8_t WordLength

数据长度

在文件 [lks32mc09x_uart.h](#) 第 18 行定义.

2.41 UART_TypeDef结构体 参考

UART寄存器

```
#include <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

2.41.1 详细描述

UART寄存器

在文件 [lks32mc09x_reg.h](#) 第 351 行定义.

2.41.2 结构体成员变量说明

2.41.2.1 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 [lks32mc09x_reg.h](#) 第 357 行定义.

2.41.2.2 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 [lks32mc09x_reg.h](#) 第 356 行定义.

2.41.2.3 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 [lks32mc09x_reg.h](#) 第 353 行定义.

2.41.2.4 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 [lks32mc09x_reg.h](#) 第 354 行定义.

2.41.2.5 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 [lks32mc09x_reg.h](#) 第 355 行定义.

2.41.2.6 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 [lks32mc09x_reg.h](#) 第 360 行定义.

2.41.2.7 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 [lks32mc09x_reg.h](#) 第 361 行定义.

2.41.2.8 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 [lks32mc09x_reg.h](#) 第 362 行定义.

2.41.2.9 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 [lks32mc09x_reg.h](#) 第 359 行定义.

2.41.2.10 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 [lks32mc09x_reg.h](#) 第 358 行定义.

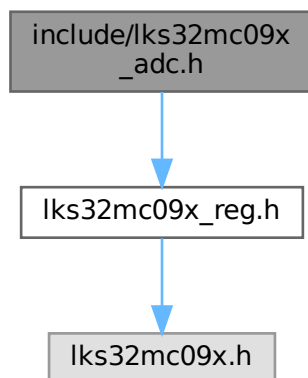
Chapter 3

文件说明

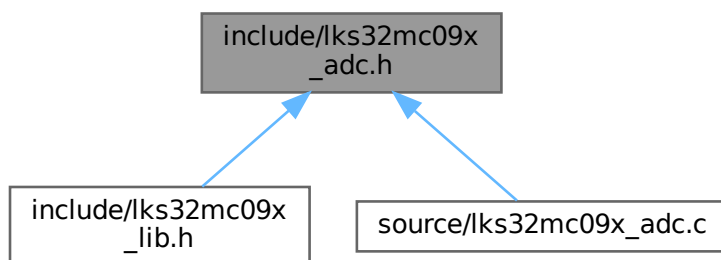
3.1 include/lks32mc09x_adc.h 文件参考

```
#include "lks32mc09x_reg.h"
```

lks32mc09x_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：



结构体

- struct **ADC_InitTypeDef**
ADC初始化结构体定义

宏定义

- #define **ADC_RE_ISF** BIT15
DMA请求使能宏定义
- #define **ADC_RE_HERR** BIT14
硬件触发发生在非空闲状态 DMA 请求使能
- #define **ADC_RE_SERR** BIT13
软件触发发生在非空闲状态 DMA 请求使能
- #define **ADC_RE_AWD0** BIT10
阈值监测 0 超限 DMA 请求使能
- #define **ADC_RE_SF2** BIT9
第二段采样完成 DMA 请求使能
- #define **ADC_RE_SF1** BIT8
第一段采样完成 DMA 请求使能
- #define **ADC_IE_ISF** BIT7
DMA中断使能宏定义
- #define **ADC_IE_HERR** BIT6
硬件触发发生在非空闲状态中断使能
- #define **ADC_IE_SERR** BIT5
软件触发发生在非空闲状态中断使能
- #define **ADC_IE_AWD0** BIT2
阈值监测 0 超限中断使能
- #define **ADC_IE_SF2** BIT1
第二段常规采样完成中断使能
- #define **ADC_IE_SF1** BIT0
第一段常规采样完成中断使能
- #define **ADC_IF_ISF** BIT7
中断标志位宏定义

- **#define ADC_IF_HERR** BIT6
硬件触发发生在非空闲状态中断标志
- **#define ADC_IF_SERR** BIT5
软件触发发生在非空闲状态中断标志
- **#define ADC_IF_AWD0** BIT2
阈值监测 0 超限中断标志
- **#define ADC_IF_SF2** BIT1
第二段常规采样完成中断标志
- **#define ADC_IF_SF1** BIT0
第一段常规采样完成中断标志
- **#define ADC_IF_ALL** (0x7f)
ADC全部中断标志位——可用于初始化
- **#define ADC_TRIG.CL_IT3** BIT27
空闲采样使能宏定义
- **#define ADC_TRIG.CL_IT2** BIT26
*CL_OUTPUT[2]*触发 ADC 空闲采样使能，高有效
- **#define ADC_TRIG.CL_IT1** BIT25
*CL_OUTPUT[1]*触发 ADC 空闲采样使能，高有效
- **#define ADC_TRIG.CL_IT0** BIT24
*CL_OUTPUT[0]*触发 ADC 空闲采样使能，高有效
- **#define ADC_TRIG.TIMER2.CMP1_IEN** BIT23
TIMER2/TIMER1 比较/捕获事件 1 触发 ADC 空闲采样使能，高有效
- **#define ADC_TRIG.TIMER2.CMP0_IEN** BIT22
TIMER2/TIMER1 比较/捕获事件 0 触发 ADC 空闲采样使能，高有效
- **#define ADC_TRIG.TIMER0.CMP1_IEN** BIT21
TIMER1/TIMER0 比较/捕获事件 1 触发 ADC 空闲采样使能，高有效
- **#define ADC_TRIG.TIMER0.CMP0_IEN** BIT20
TIMER0 比较/捕获事件 0 触发 ADC 空闲采样使能，高有效
- **#define ADC_TRIG.MCPWM0.T3_IEN** BIT19
MCPWM0 T3 事件触发 ADC 空闲采样使能，高有效
- **#define ADC_TRIG.MCPWM0.T2_IEN** BIT18
MCPWM0 T2 事件触发 ADC 空闲采样使能，高有效
- **#define ADC_TRIG.MCPWM0.T1_IEN** BIT17
MCPWM0 T1 事件触发 ADC 空闲采样使能，高有效
- **#define ADC_TRIG.MCPWM0.T0_IEN** BIT16
MCPWM0 T0 事件触发 ADC 空闲采样使能，高有效
- **#define ADC_TRIG.CL_NT3** BIT11
常规采样使能宏定义
- **#define ADC_TRIG.CL_NT2** BIT10
*CL_OUTPUT[2]*触发 ADC 常规采样
- **#define ADC_TRIG.CL_NT1** BIT9
*CL_OUTPUT[1]*触发 ADC 常规采样
- **#define ADC_TRIG.CL_NT0** BIT8
*CL_OUTPUT[0]*触发 ADC 常规采样
- **#define ADC_TRIG.TIMER1.CMP1** BIT7
TIMER1 比较事件 1 触发 ADC 常规采样
- **#define ADC_TRIG.TIMER1.CMP0** BIT6
TIMER1 比较事件 0 触发 ADC 常规采样
- **#define ADC_TRIG.TIMER0.CMP1** BIT5
TIMER0 比较事件 1 触发 ADC 常规采样
- **#define ADC_TRIG.TIMER0.CMP0** BIT4

- *TIMER0* 比较事件 0 触发 *ADC* 常规采样
- #define **ADC_TRIG_MCPWM0_T3** BIT3
MCPWM0 T3 事件触发 *ADC* 常规采样
- #define **ADC_TRIG_MCPWM0_T2** BIT2
MCPWM0 T2 事件触发 *ADC* 常规采样
- #define **ADC_TRIG_MCPWM0_T1** BIT1
MCPWM0 T1 事件触发 *ADC* 常规采样
- #define **ADC_TRIG_MCPWM0_T0** BIT0
MCPWM0 T0 事件触发 *ADC* 常规采样
- #define **ADC_TRIG_SOFT** 0
仅软件触发 *ADC* 常规采样

枚举

- enum **ADC_CHN_x** {
ADC_CHN_0 = 0, ADC_CHN_1, ADC_CHN_2, ADC_CHN_3,
ADC_CHN_4, ADC_CHN_5, ADC_CHN_6, ADC_CHN_7,
ADC_CHN_8, ADC_CHN_9, ADC_CHN_10, ADC_CHN_11,
ADC_CHN_12, ADC_CHN_13, ADC_CHN_14, ADC_CHN_15,
ADC0_CHN_OPA0 = ADC_CHN_0, ADC0_CHN_OPA1 = ADC_CHN_1, ADC0_CHN_OPA2 = ADC_CHN_2,
ADC0_CHN_OPA3 = ADC_CHN_3,
ADC0_CHN_ADC01_CH4 = ADC_CHN_4, ADC0_CHN_ADC01_CH5 = ADC_CHN_5, ADC0_CHN_ADC01_CH6
= ADC_CHN_6, ADC0_CHN_ADC01_CH7 = ADC_CHN_7,
ADC0_CHN_ADC01_CH8 = ADC_CHN_8, ADC0_CHN_REF24 = ADC_CHN_9, ADC0_CHN_ADC01_CH10 =
ADC_CHN_10, ADC0_CHN_ADC01_CH11 = ADC_CHN_11,
ADC0_CHN_ADC01_CH12 = ADC_CHN_12, ADC0_CHN_ADC01_CH13 = ADC_CHN_13, ADC0_CHN_DAC =
ADC_CHN_14, ADC0_CHN_AVSS = ADC_CHN_15,
ADC1_CHN_OPA0 = ADC_CHN_0, ADC1_CHN_OPA1 = ADC_CHN_1, ADC1_CHN_OPA2 = ADC_CHN_2,
ADC1_CHN_OPA3 = ADC_CHN_3,
ADC1_CHN_ADC01_CH4 = ADC_CHN_4, ADC1_CHN_ADC01_CH5 = ADC_CHN_5, ADC1_CHN_ADC01_CH6
= ADC_CHN_6, ADC1_CHN_ADC01_CH7 = ADC_CHN_7,
ADC1_CHN_ADC01_CH8 = ADC_CHN_8, ADC1_CHN_ADC01_CH9 = ADC_CHN_9, ADC1_CHN_ADC01_CH10
= ADC_CHN_10, ADC1_CHN_ADC01_CH11 = ADC_CHN_11,
ADC1_CHN_ADC01_CH12 = ADC_CHN_12, ADC1_CHN_ADC01_CH13 = ADC_CHN_13, ADC1_CHN_TMP =
ADC_CHN_14, ADC1_CHN_AVDD = ADC_CHN_15 }
ADC采样通道序号枚举定义
- enum **ADC_DAT_x** {
ADC_DAT_0 = 0, ADC_DAT_1, ADC_DAT_2, ADC_DAT_3,
ADC_DAT_4, ADC_DAT_5, ADC_DAT_6, ADC_DAT_7,
ADC_DAT_8, ADC_DAT_9, ADC_DAT_10, ADC_DAT_11 }
ADC数据寄存器序号枚举定义
- enum **ADC_GAIN_x** { **ADC_GAIN_3V6** = 0, **ADC_GAIN_7V2** = 1 }
ADC增益序号枚举定义
- enum **ADC_RANGE_x** { **ADC_RANGE_3_3** = 0, **ADC_RANGE_3_6** = 1, **ADC_RANGE_EXTERNAL** = 2 }
ADC基准枚举定义

函数

- void **ADC_Enable** (ADC_TypeDef *ADCx)
使能ADC模块
- void **ADC_Disable** (ADC_TypeDef *ADCx)
禁用ADC模块
- void **ADC_Reset** (ADC_TypeDef *ADCx)

- 复位ADC模块
- void **ADC_Init** (ADC_TypeDef *ADCx, ADC_InitTypeDef *ADC_InitStruct)
初始化ADC模块
- void **ADC_StructInit** (ADC_InitTypeDef *ADC_InitStruct)
将ADC初始化结构体置为默认值
- uint16_t **ADC_GetIRQFlag** (ADC_TypeDef *ADCx, uint16_t INT_flag)
获取ADC中断标志位状态
- void **ADC_ClearIRQFlag** (ADC_TypeDef *ADCx, uint16_t INT_flag)
清除ADC中断标志位
- void **ADC_SetGain** (ADC_TypeDef *ADCx, ADC_DAT_x DATNum, ADC_GAIN_x gainx)
设置ADC通道的增益
- s16 **ADC_GetConversionValue** (ADC_TypeDef *ADCx, ADC_DAT_x DATNum)
获取ADC通道的转换值
- int32_t **ADC_GetConversionValueS32** (ADC_TypeDef *ADCx, ADC_DAT_x DATNum)
获取ADC通道的转换值(按int32返回, 用来采样adc通道4、5、6、7)
- void **ADC_SoftTrig** (ADC_TypeDef *ADCx)
执行软件触发ADC采样, 采样次数取决于ADC_InitTypeDef里的S1的配置
- void **ADC_SoftIdleTrig** (ADC_TypeDef *ADCx)
执行软件触发ADC空闲采样
- void **ADC_SetChn** (ADC_TypeDef *ADCx, ADC_DAT_x datx, ADC_CHN_x chnx)
设置第每次的采样通道, 例如ADC_SetChn(ADC0,ADC_DAT_0,ADC_CHN_0),把adc第一次采样的通道设置为通道0
- void **ADC_SetRange** (ADC_TypeDef *ADCx, ADC_RANGE_x rangex)
设置ADC基准电压

3.1.1 宏定义说明

3.1.1.1 ADC_IE_AWD0

```
#define ADC_IE_AWD0 BIT2
```

阈值监测 0 超限中断使能

在文件 [lks32mc09x_adc.h](#) 第 46 行定义.

3.1.1.2 ADC_IE_HERR

```
#define ADC_IE_HERR BIT6
```

硬件触发发生在非空闲状态中断使能

在文件 [lks32mc09x_adc.h](#) 第 44 行定义.

3.1.1.3 ADC_IE_ISF

```
#define ADC_IE_ISF BIT7
```

DMA中断使能宏定义

空闲采样完成中断使能

在文件 [lks32mc09x_adc.h](#) 第 43 行定义.

3.1.1.4 ADC_IE_SERR

```
#define ADC_IE_SERR BIT5
```

软件触发发生在非空闲状态中断使能

在文件 [lks32mc09x_adc.h](#) 第 45 行定义.

3.1.1.5 ADC_IE_SF1

```
#define ADC_IE_SF1 BIT0
```

第一段常规采样完成中断使能

在文件 [lks32mc09x_adc.h](#) 第 48 行定义.

3.1.1.6 ADC_IE_SF2

```
#define ADC_IE_SF2 BIT1
```

第二段常规采样完成中断使能

在文件 [lks32mc09x_adc.h](#) 第 47 行定义.

3.1.1.7 ADC_IF_ALL

```
#define ADC_IF_ALL (0x7f)
```

ADC全部中断标志位——可用于初始化

在文件 [lks32mc09x_adc.h](#) 第 59 行定义.

3.1.1.8 ADC_IF_AWD0

```
#define ADC_IF_AWD0 BIT2
```

阈值监测 0 超限中断标志

在文件 [lks32mc09x_adc.h](#) 第 56 行定义.

3.1.1.9 ADC_IF_HERR

```
#define ADC_IF_HERR BIT6
```

硬件触发发生在非空闲状态中断标志

在文件 [lks32mc09x_adc.h](#) 第 54 行定义.

3.1.1.10 ADC_IF_ISF

```
#define ADC_IF_ISF BIT7
```

中断标志位宏定义

空闲采样完成中断标志

在文件 [lks32mc09x_adc.h](#) 第 53 行定义.

3.1.1.11 ADC_IF_SERR

```
#define ADC_IF_SERR BIT5
```

软件触发发生在非空闲状态中断标志

在文件 [lks32mc09x_adc.h](#) 第 55 行定义.

3.1.1.12 ADC_IF_SF1

```
#define ADC_IF_SF1 BIT0
```

第一段常规采样完成中断标志

在文件 [lks32mc09x_adc.h](#) 第 58 行定义.

3.1.1.13 ADC_IF_SF2

```
#define ADC_IF_SF2 BIT1
```

第二段常规采样完成中断标志

在文件 [lks32mc09x_adc.h](#) 第 57 行定义.

3.1.1.14 ADC_RE_AWD0

```
#define ADC_RE_AWD0 BIT10
```

阈值监测 0 超限 DMA 请求使能

在文件 [lks32mc09x_adc.h](#) 第 36 行定义.

3.1.1.15 ADC_RE_HERR

```
#define ADC_RE_HERR BIT14
```

硬件触发发生在非空闲状态 DMA 请求使能

在文件 [lks32mc09x_adc.h](#) 第 34 行定义.

3.1.1.16 ADC_RE_ISF

```
#define ADC_RE_ISF BIT15
```

DMA请求使能宏定义

空闲采样完成 DMA 请求使能

在文件 [lks32mc09x_adc.h](#) 第 33 行定义.

3.1.1.17 ADC_RE_SERR

```
#define ADC_RE_SERR BIT13
```

软件触发发生在非空闲状态 DMA 请求使能

在文件 [lks32mc09x_adc.h](#) 第 35 行定义.

3.1.1.18 ADC_RE_SF1

```
#define ADC_RE_SF1 BIT8
```

第一段采样完成 DMA 请求使能

在文件 [lks32mc09x_adc.h](#) 第 38 行定义.

3.1.1.19 ADC_RE_SF2

```
#define ADC_RE_SF2 BIT9
```

第二段采样完成 DMA 请求使能

在文件 [lks32mc09x_adc.h](#) 第 37 行定义.

3.1.1.20 ADC_TRIG_CL_IT0

```
#define ADC_TRIG_CL_IT0 BIT24
```

CL.OUTPUT[0]触发 ADC 空闲采样使能, 高有效

在文件 [lks32mc09x_adc.h](#) 第 67 行定义.

3.1.1.21 ADC_TRIG_CL_IT1

```
#define ADC_TRIG_CL_IT1 BIT25
```

CL.OUTPUT[1]触发 ADC 空闲采样使能, 高有效

在文件 [lks32mc09x_adc.h](#) 第 66 行定义.

3.1.1.22 ADC_TRIG_CL.IT2

```
#define ADC_TRIG_CL_IT2 BIT26
```

CL.OUTPUT[2]触发 ADC 空闲采样使能，高有效
在文件 [lks32mc09x_adc.h](#) 第 65 行定义.

3.1.1.23 ADC_TRIG_CL.IT3

```
#define ADC_TRIG_CL_IT3 BIT27
```

空闲采样使能宏定义

CL.OUTPUT[3]触发 ADC 空闲采样使能，高有效
在文件 [lks32mc09x_adc.h](#) 第 64 行定义.

3.1.1.24 ADC_TRIG_CL.NT0

```
#define ADC_TRIG_CL_NT0 BIT8
```

CL.OUTPUT[0]触发 ADC 常规采样
在文件 [lks32mc09x_adc.h](#) 第 83 行定义.

3.1.1.25 ADC_TRIG_CL.NT1

```
#define ADC_TRIG_CL_NT1 BIT9
```

CL.OUTPUT[1]触发 ADC 常规采样
在文件 [lks32mc09x_adc.h](#) 第 82 行定义.

3.1.1.26 ADC_TRIG_CL.NT2

```
#define ADC_TRIG_CL_NT2 BIT10
```

CL.OUTPUT[2]触发 ADC 常规采样
在文件 [lks32mc09x_adc.h](#) 第 81 行定义.

3.1.1.27 ADC_TRIG_CL.NT3

```
#define ADC_TRIG_CL_NT3 BIT11
```

常规采样使能宏定义

CL.OUTPUT[3]触发 ADC 常规采样
在文件 [lks32mc09x_adc.h](#) 第 80 行定义.

3.1.1.28 ADC_TRIG_MCPWM0_T0

```
#define ADC_TRIG_MCPWM0_T0 BIT0
```

MCPWM0 T0 事件触发 ADC 常规采样

在文件 [lks32mc09x_adc.h](#) 第 91 行定义.

3.1.1.29 ADC_TRIG_MCPWM0_T0_IEN

```
#define ADC_TRIG_MCPWM0_T0_IEN BIT16
```

MCPWM0 T0 事件触发 ADC 空闲采样使能，高有效

在文件 [lks32mc09x_adc.h](#) 第 75 行定义.

3.1.1.30 ADC_TRIG_MCPWM0_T1

```
#define ADC_TRIG_MCPWM0_T1 BIT1
```

MCPWM0 T1 事件触发 ADC 常规采样

在文件 [lks32mc09x_adc.h](#) 第 90 行定义.

3.1.1.31 ADC_TRIG_MCPWM0_T1_IEN

```
#define ADC_TRIG_MCPWM0_T1_IEN BIT17
```

MCPWM0 T1 事件触发 ADC 空闲采样使能，高有效

在文件 [lks32mc09x_adc.h](#) 第 74 行定义.

3.1.1.32 ADC_TRIG_MCPWM0_T2

```
#define ADC_TRIG_MCPWM0_T2 BIT2
```

MCPWM0 T2 事件触发 ADC 常规采样

在文件 [lks32mc09x_adc.h](#) 第 89 行定义.

3.1.1.33 ADC_TRIG_MCPWM0_T2_IEN

```
#define ADC_TRIG_MCPWM0_T2_IEN BIT18
```

MCPWM0 T2 事件触发 ADC 空闲采样使能，高有效

在文件 [lks32mc09x_adc.h](#) 第 73 行定义.

3.1.1.34 ADC_TRIG_MCPWM0_T3

```
#define ADC_TRIG_MCPWM0_T3 BIT3
```

MCPWM0 T3 事件触发 ADC 常规采样

在文件 [lks32mc09x_adc.h](#) 第 88 行定义.

3.1.1.35 ADC_TRIG_MCPWM0_T3_IEN

```
#define ADC_TRIG_MCPWM0_T3_IEN BIT19
```

MCPWM0 T3 事件触发 ADC 空闲采样使能, 高有效

在文件 [lks32mc09x_adc.h](#) 第 72 行定义.

3.1.1.36 ADC_TRIG_SOFT

```
#define ADC_TRIG_SOFT 0
```

仅软件触发 ADC 常规采样

在文件 [lks32mc09x_adc.h](#) 第 92 行定义.

3.1.1.37 ADC_TRIG_TIMER0_CMP0

```
#define ADC_TRIG_TIMER0_CMP0 BIT4
```

TIMER0 比较事件 0 触发 ADC 常规采样

在文件 [lks32mc09x_adc.h](#) 第 87 行定义.

3.1.1.38 ADC_TRIG_TIMER0_CMP0_IEN

```
#define ADC_TRIG_TIMER0_CMP0_IEN BIT20
```

TIMER0 比较/捕获事件 0 触发 ADC 空闲采样使能, 高有效

在文件 [lks32mc09x_adc.h](#) 第 71 行定义.

3.1.1.39 ADC_TRIG_TIMER0_CMP1

```
#define ADC_TRIG_TIMER0_CMP1 BIT5
```

TIMER0 比较事件 1 触发 ADC 常规采样

在文件 [lks32mc09x_adc.h](#) 第 86 行定义.

3.1.1.40 ADC_TRIG_TIMER0_CMP1_IEN

```
#define ADC_TRIG_TIMER0_CMP1_IEN BIT21
```

TIMER1/TIMER0 比较/捕获事件 1 触发 ADC 空闲采样使能，高有效
在文件 [lks32mc09x_adc.h](#) 第 70 行定义.

3.1.1.41 ADC_TRIG_TIMER1_CMP0

```
#define ADC_TRIG_TIMER1_CMP0 BIT6
```

TIMER1 比较事件 0 触发 ADC 常规采样
在文件 [lks32mc09x_adc.h](#) 第 85 行定义.

3.1.1.42 ADC_TRIG_TIMER1_CMP1

```
#define ADC_TRIG_TIMER1_CMP1 BIT7
```

TIMER1 比较事件 1 触发 ADC 常规采样
在文件 [lks32mc09x_adc.h](#) 第 84 行定义.

3.1.1.43 ADC_TRIG_TIMER2_CMP0_IEN

```
#define ADC_TRIG_TIMER2_CMP0_IEN BIT22
```

TIMER2/TIMER1 比较/捕获事件 0 触发 ADC 空闲采样使能，高有效
在文件 [lks32mc09x_adc.h](#) 第 69 行定义.

3.1.1.44 ADC_TRIG_TIMER2_CMP1_IEN

```
#define ADC_TRIG_TIMER2_CMP1_IEN BIT23
```

TIMER2/TIMER1 比较/捕获事件 1 触发 ADC 空闲采样使能，高有效
在文件 [lks32mc09x_adc.h](#) 第 68 行定义.

3.1.2 枚举类型说明

3.1.2.1 ADC_CHN_x

```
enum ADC_CHN_x
```

ADC采样通道序号枚举定义

枚举值

ADC.CHN_0	采样通道序号
ADC.CHN_1	采样通道序号
ADC.CHN_2	采样通道序号
ADC.CHN_3	采样通道序号
ADC.CHN_4	采样通道序号
ADC.CHN_5	采样通道序号
ADC.CHN_6	采样通道序号
ADC.CHN_7	采样通道序号
ADC.CHN_8	采样通道序号
ADC.CHN_9	采样通道序号
ADC.CHN_10	采样通道序号
ADC.CHN_11	采样通道序号
ADC.CHN_12	采样通道序号
ADC.CHN_13	采样通道序号
ADC.CHN_14	采样通道序号
ADC.CHN_15	采样通道序号
ADC0.CHN.OPA0	ADC0采样通道序号
ADC0.CHN.OPA1	ADC0采样通道序号
ADC0.CHN.OPA2	ADC0采样通道序号
ADC0.CHN.OPA3	ADC0采样通道序号
ADC0.CHN.ADC01.CH4	ADC0采样通道序号
ADC0.CHN.ADC01.CH5	ADC0采样通道序号
ADC0.CHN.ADC01.CH6	ADC0采样通道序号
ADC0.CHN.ADC01.CH7	ADC0采样通道序号
ADC0.CHN.ADC0.CH8	ADC0采样通道序号
ADC0.CHN.REF24	ADC0采样通道序号
ADC0.CHN.ADC01.CH10	ADC0采样通道序号
ADC0.CHN.ADC01.CH11	ADC0采样通道序号
ADC0.CHN.ADC0.CH12	ADC0采样通道序号
ADC0.CHN.ADC0.CH13	ADC0采样通道序号
ADC0.CHN.DAC	ADC0采样通道序号
ADC0.CHN.AVSS	ADC0采样通道序号
ADC1.CHN.OPA0	ADC1采样通道序号
ADC1.CHN.OPA1	ADC1采样通道序号
ADC1.CHN.OPA2	ADC1采样通道序号
ADC1.CHN.OPA3	ADC1采样通道序号
ADC1.CHN.ADC01.CH4	ADC1采样通道序号
ADC1.CHN.ADC01.CH5	ADC1采样通道序号
ADC1.CHN.ADC01.CH6	ADC1采样通道序号
ADC1.CHN.ADC01.CH7	ADC1采样通道序号
ADC1.CHN.ADC1.CH8	ADC1采样通道序号
ADC1.CHN.ADC1.CH9	ADC1采样通道序号
ADC1.CHN.ADC01.CH10	ADC1采样通道序号
ADC1.CHN.ADC01.CH11	ADC1采样通道序号

枚举值

ADC1_CHN_ADC1_CH12	ADC1采样通道序号
ADC1_CHN_ADC1_CH13	ADC1采样通道序号
ADC1_CHN_TMP	ADC1采样通道序号
ADC1_CHN_AVDD	ADC1采样通道序号

在文件 [lks32mc09x_adc.h](#) 第 97 行定义.

3.1.2.2 ADC_DAT_x

enum [ADC_DAT_x](#)

ADC数据寄存器序号枚举定义

枚举值

ADC_DAT_0	第0次采样信号对应结果寄存器序号
ADC_DAT_1	第1次采样信号对应结果寄存器序号
ADC_DAT_2	第2次采样信号对应结果寄存器序号
ADC_DAT_3	第3次采样信号对应结果寄存器序号
ADC_DAT_4	第4次采样信号对应结果寄存器序号
ADC_DAT_5	第5次采样信号对应结果寄存器序号
ADC_DAT_6	第6次采样信号对应结果寄存器序号
ADC_DAT_7	第7次采样信号对应结果寄存器序号
ADC_DAT_8	第8次采样信号对应结果寄存器序号
ADC_DAT_9	第9次采样信号对应结果寄存器序号
ADC_DAT_10	第10次采样信号对应结果寄存器序号
ADC_DAT_11	第11次采样信号对应结果寄存器序号

在文件 [lks32mc09x_adc.h](#) 第 154 行定义.

3.1.2.3 ADC_GAIN_x

enum [ADC_GAIN_x](#)

ADC增益序号枚举定义

枚举值

ADC_GAIN_3V6	3.6V 量程
ADC_GAIN_7V2	7.2V 量程

在文件 [lks32mc09x_adc.h](#) 第 173 行定义.

3.1.2.4 ADC_RANGE_x

enum ADC_RANGE_x

ADC基准枚举定义

枚举值

ADC_RANGE_3_3	3.3V基准
ADC_RANGE_3_6	3.6V基准
ADC_RANGE_EXTERNAL	外部基准

在文件 lks32mc09x_adc.h 第 182 行定义.

3.1.3 函数说明

3.1.3.1 ADC_ClearIRQFlag()

```
void ADC_ClearIRQFlag (
    ADC_TypeDef * ADCx,
    uint16_t INT_flag )
```

清除ADC中断标志位

参数

ADCx	ADC模块指针，指向ADC0或ADC1
INT_flag	中断标志位，可以是ADC_IF_ISF等

在文件 lks32mc09x_adc.c 第 151 行定义.

3.1.3.2 ADC_Disable()

```
void ADC_Disable (
    ADC_TypeDef * ADCx )
```

禁用ADC模块

参数

ADCx	ADC模块指针，指向ADC0或ADC1

在文件 lks32mc09x_adc.c 第 25 行定义.

3.1.3.3 ADC_Enable()

```
void ADC_Enable (
```

```
ADC_TypeDef * ADCx )
```

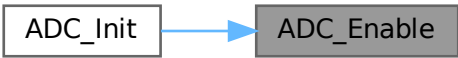
使能ADC模块

参数

ADCx	ADC模块指针，指向ADC0或ADC1

在文件 [lks32mc09x_adc.c](#) 第 10 行定义.

这是这个函数的调用关系图:



3.1.3.4 ADC_GetConversionValue()

```
s16 ADC_GetConversionValue (
    ADC_TypeDef * ADCx,
    ADC_DAT_x DATNum )
```

获取ADC通道的转换值

参数

ADCx	ADC模块指针，指向ADC0或ADC1
DATNum	数据寄存器序号，如ADC_DAT_0

返回

转换值

在文件 [lks32mc09x_adc.c](#) 第 173 行定义.

函数调用图:



3.1.3.5 ADC_GetConversionValueS32()

```
int32_t ADC_GetConversionValueS32 (
    ADC_TypeDef * ADCx,
    ADC_DAT_x DATNum )
```

获取ADC通道的转换值(按int32返回，用来采样adc通道4、5、6、7)

参数

ADCx	ADC模块指针，指向ADC0或ADC1
DATNum	数据寄存器序号，如ADC_DAT_0

返回
转换值

获取ADC通道的转换值(按int32返回，用来采样adc通道4、5、6、7)

参数

ADCx	ADC模块指针，指向ADC0或ADC1
DATNum	数据寄存器序号，如ADC_DAT_0

返回
转换值

在文件 [lks32mc09x_adc.c](#) 第 190 行定义.

这是这个函数的调用关系图:



3.1.3.6 ADC_GetIRQFlag()

```
uint16_t ADC_GetIRQFlag (
    ADC_TypeDef * ADCx,
    uint16_t INT_flag )
```

获取ADC中断标志位状态

参数

<i>ADCx</i>	ADC模块指针，指向ADC0或ADC1
<i>INT_flag</i>	中断标志位，可以是ADC_IF_ISF等

返回
中断标志位状态

在文件 [lks32mc09x_adc.c](#) 第 141 行定义.

3.1.3.7 ADC_Init()

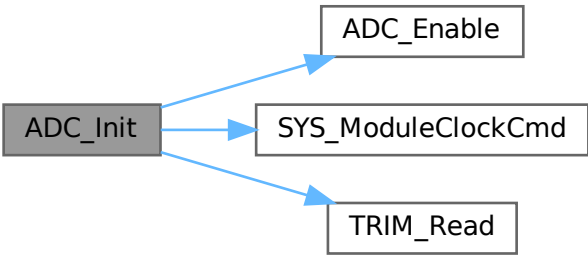
```
void ADC_Init (
    ADC_TypeDef * ADCx,
    ADC_InitTypeDef * ADC_InitStruct )
```

初始化ADC模块

参数

<i>ADCx</i>	ADC模块指针，指向ADC0或ADC1
<i>ADC_InitStruct</i>	指向ADC初始化结构体的指针

在文件 [lks32mc09x_adc.c](#) 第 61 行定义.
函数调用图:



3.1.3.8 ADC_Reset()

```
void ADC_Reset (
    ADC_TypeDef * ADCx )
```

复位ADC模块

参数

<i>ADCx</i>	ADC模块指针，指向ADC0或ADC1

在文件 [lks32mc09x_adc.c](#) 第 40 行定义.

函数调用图:



3.1.3.9 ADC_SetChn()

```
void ADC_SetChn (
    ADC_TypeDef * ADCx,
    ADC_DAT_x datx,
    ADC_CHN_x chnx )
```

设置第每次的采样通道，例如ADC_SetChn(ADC0,ADC_DAT_0,ADC_CHN_0),把adc第一次采样的通道设置为通道0

参数

<i>ADCx</i>	ADC模块指针，指向ADC0或ADC1
<i>datx</i>	数据寄存器序号，如ADC_DAT_0
<i>chnx</i>	ADC采样通道序号枚举定义，如ADC_CHN_0

在文件 [lks32mc09x_adc.c](#) 第 299 行定义.

3.1.3.10 ADC_SetGain()

```
void ADC_SetGain (
    ADC_TypeDef * ADCx,
    ADC_DAT_x DAT_x,
    ADC_GAIN_x gainx )
```

设置ADC通道的增益

参数

<i>ADCx</i>	ADC模块指针，指向ADC0或ADC1

参数

<i>DATNum</i>	数据寄存器序号，如ADC_DAT_0
<i>gainx</i>	增益选择，如ADC_GAIN_3V6
<i>ADCx</i>	ADC模块指针，指向ADC0或ADC1
<i>DAT_x</i>	数据寄存器序号，如ADC_DAT_0表示第0次采样结果
<i>gainx</i>	增益选择，如ADC_GAIN_3V6

在文件 [lks32mc09x_adc.c](#) 第 162 行定义.

3.1.3.11 ADC_SetRange()

```
void ADC_SetRange (
    ADC_TypeDef * ADCx,
    ADC_RANGE_x rangex )
```

设置ADC基准电压

参数

<i>ADCx</i>	ADC模块指针，指向ADC0或ADC1
<i>rangex</i>	基准选择
<i>ADCx</i>	ADC模块指针,注意: adc0和adc1共用同一个基准电压
<i>rangex</i>	基准选择

< 3.3V基准

< 外部基准

< 3.6V基准

在文件 [lks32mc09x_adc.c](#) 第 348 行定义.

3.1.3.12 ADC_SoftIdleTrig()

```
void ADC_SoftIdleTrig (
    ADC_TypeDef * ADCx )
```

执行软件触发ADC空闲采样

参数

<i>ADCx</i>	ADC模块指针，指向ADC0

在文件 [lks32mc09x_adc.c](#) 第 287 行定义.

3.1.3.13 ADC_SoftTrig()

```
void ADC_SoftTrig (
    ADC_TypeDef * ADCx )
```

执行软件触发ADC采样，采样次数取决于ADC.InitTypeDef里的S1的配置

参数

ADCx	ADC模块指针，指向ADC0或ADC1

在文件 [lks32mc09x_adc.c](#) 第 277 行定义.

3.1.3.14 ADC_StructInit()

```
void ADC_StructInit (
    ADC_InitTypeDef * ADC_InitStruct )
```

将ADC初始化结构体置为默认值

参数

ADC_InitStruct	指向ADC初始化结构体的指针

在文件 [lks32mc09x_adc.c](#) 第 126 行定义.

3.2 lks32mc09x_adc.h

[浏览该文件的文档.](#)

```
00001 #ifndef __LKS32MC09X_ADC_H__
00002 #define __LKS32MC09X_ADC_H__
00003
00004 #include "lks32mc09x_reg.h"
00005
00009 typedef struct
00010 {
00011     uint16_t IE;
00012     uint16_t RE;
00013     uint16_t NSMP;
00014     uint16_t DATA_ALIGN;
00015     uint16_t CSMP;
00016     uint16_t TCNT;
00017     uint16_t TROVS;
00018     uint16_t OVSR;
00019     uint32_t TRIG;
00020     uint16_t S1;
00021     uint16_t S2;
00022     uint16_t IS1;
00023     uint16_t GAIN;
00024     uint16_t LTH;
00025     uint16_t HTH;
00026     uint16_t GEN;
00027     uint16_t IDLE_PRI;
00028 } ADC_InitTypeDef;
00029
00033 #define ADC_RE_ISF BIT15
00034 #define ADC_RE_HERR BIT14
00035 #define ADC_RE_SERR BIT13
00036 #define ADC_RE_AWD0 BIT10
00037 #define ADC_RE_SF2 BIT9
```

```

00038 #define ADC_RE_SF1 BIT8
00039
00043 #define ADC_IE_ISF BIT7
00044 #define ADC_IE_HERR BIT6
00045 #define ADC_IE_SERR BIT5
00046 #define ADC_IE_AWD0 BIT2
00047 #define ADC_IE_SF2 BIT1
00048 #define ADC_IE_SF1 BIT0
00049
00053 #define ADC_IF_ISF BIT7
00054 #define ADC_IF_HERR BIT6
00055 #define ADC_IF_SERR BIT5
00056 #define ADC_IF_AWD0 BIT2
00057 #define ADC_IF_SF2 BIT1
00058 #define ADC_IF_SF1 BIT0
00059 #define ADC_IF_ALL (0x7f)
00060
00064 #define ADC_TRIG_CL_IT3 BIT27
00065 #define ADC_TRIG_CL_IT2 BIT26
00066 #define ADC_TRIG_CL_IT1 BIT25
00067 #define ADC_TRIG_CL_IT0 BIT24
00068 #define ADC_TRIG_TIMER2_CMP1_IEN BIT23
00069 #define ADC_TRIG_TIMER2_CMP0_IEN BIT22
00070 #define ADC_TRIG_TIMER0_CMP1_IEN BIT21
00071 #define ADC_TRIG_TIMER0_CMP0_IEN BIT20
00072 #define ADC_TRIG_MCPWM0_T3_IEN BIT19
00073 #define ADC_TRIG_MCPWM0_T2_IEN BIT18
00074 #define ADC_TRIG_MCPWM0_T1_IEN BIT17
00075 #define ADC_TRIG_MCPWM0_T0_IEN BIT16
00076
00080 #define ADC_TRIG_CL_NT3 BIT11
00081 #define ADC_TRIG_CL_NT2 BIT10
00082 #define ADC_TRIG_CL_NT1 BIT9
00083 #define ADC_TRIG_CL_NT0 BIT8
00084 #define ADC_TRIG_TIMER1_CMP1 BIT7
00085 #define ADC_TRIG_TIMER1_CMP0 BIT6
00086 #define ADC_TRIG_TIMER0_CMP1 BIT5
00087 #define ADC_TRIG_TIMER0_CMP0 BIT4
00088 #define ADC_TRIG_MCPWM0_T3 BIT3
00089 #define ADC_TRIG_MCPWM0_T2 BIT2
00090 #define ADC_TRIG_MCPWM0_T1 BIT1
00091 #define ADC_TRIG_MCPWM0_T0 BIT0
00092 #define ADC_TRIG_SOFT 0
00093
00097 typedef enum
00098 {
00099     ADC_CHN_0 = 0,
00100     ADC_CHN_1,
00101     ADC_CHN_2,
00102     ADC_CHN_3,
00103     ADC_CHN_4,
00104     ADC_CHN_5,
00105     ADC_CHN_6,
00106     ADC_CHN_7,
00107     ADC_CHN_8,
00108     ADC_CHN_9,
00109     ADC_CHN_10,
00110     ADC_CHN_11,
00111     ADC_CHN_12,
00112     ADC_CHN_13,
00113     ADC_CHN_14,
00114     ADC_CHN_15,
00115     // ADC0
00116     ADC0_CHN_OPA0 = ADC_CHN_0,
00117     ADC0_CHN_OPA1 = ADC_CHN_1,
00118     ADC0_CHN_OPA2 = ADC_CHN_2,
00119     ADC0_CHN_OPA3 = ADC_CHN_3,
00120     ADC0_CHN_ADC01_CH4 = ADC_CHN_4,
00121     ADC0_CHN_ADC01_CH5 = ADC_CHN_5,
00122     ADC0_CHN_ADC01_CH6 = ADC_CHN_6,
00123     ADC0_CHN_ADC01_CH7 = ADC_CHN_7,
00124     ADC0_CHN_ADC01_CH8 = ADC_CHN_8,
00125     ADC0_CHN_REF24 = ADC_CHN_9,
00126     ADC0_CHN_ADC01_CH10 = ADC_CHN_10,
00127     ADC0_CHN_ADC01_CH11 = ADC_CHN_11,
00128     ADC0_CHN_ADC01_CH12 = ADC_CHN_12,
00129     ADC0_CHN_ADC01_CH13 = ADC_CHN_13,
00130     ADC0_CHN_DAC = ADC_CHN_14,
00131     ADC0_CHN_AVSS = ADC_CHN_15,
00132     // ADC1
00133     ADC1_CHN_OPA0 = ADC_CHN_0,
00134     ADC1_CHN_OPA1 = ADC_CHN_1,
00135     ADC1_CHN_OPA2 = ADC_CHN_2,
00136     ADC1_CHN_OPA3 = ADC_CHN_3,
00137     ADC1_CHN_ADC01_CH4 = ADC_CHN_4,
00138     ADC1_CHN_ADC01_CH5 = ADC_CHN_5,
00139     ADC1_CHN_ADC01_CH6 = ADC_CHN_6,

```

```

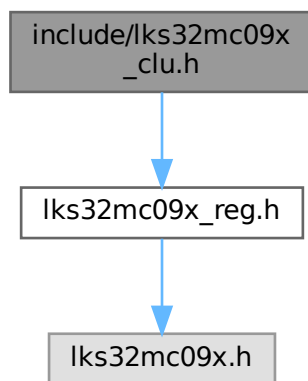
00140     ADC1.CHN_ADC01.CH7  = ADC.CHN_7,
00141     ADC1.CHN_ADC1.CH8   = ADC.CHN_8,
00142     ADC1.CHN_ADC1.CH9   = ADC.CHN_9,
00143     ADC1.CHN_ADC01.CH10 = ADC.CHN_10,
00144     ADC1.CHN_ADC01.CH11 = ADC.CHN_11,
00145     ADC1.CHN_ADC1.CH12  = ADC.CHN_12,
00146     ADC1.CHN_ADC1.CH13  = ADC.CHN_13,
00147     ADC1.CHN_TMP        = ADC.CHN_14,
00148     ADC1.CHN_AVDD       = ADC.CHN_15,
00149 } ADC.CHN_x;
00150
00154 typedef enum
00155 {
00156     ADC_DAT_0 = 0,
00157     ADC_DAT_1,
00158     ADC_DAT_2,
00159     ADC_DAT_3,
00160     ADC_DAT_4,
00161     ADC_DAT_5,
00162     ADC_DAT_6,
00163     ADC_DAT_7,
00164     ADC_DAT_8,
00165     ADC_DAT_9,
00166     ADC_DAT_10,
00167     ADC_DAT_11,
00168 } ADC_DAT_x;
00169
00173 typedef enum
00174 {
00175     ADC_GAIN_3V6 = 0,
00176     ADC_GAIN_7V2 = 1,
00177 } ADC_GAIN_x;
00178
00182 typedef enum
00183 {
00184     ADC_RANGE_3_3      = 0,
00185     ADC_RANGE_3_6      = 1,
00186     ADC_RANGE_EXTERNAL = 2,
00187 } ADC_RANGE_x;
00192 void ADC_Enable(ADC_TypeDef *ADCx);
00193
00198 void ADC_Disable(ADC_TypeDef *ADCx);
00199
00204 void ADC_Reset(ADC_TypeDef *ADCx);
00205
00211 void ADC_Init(ADC_TypeDef *ADCx, ADC_InitTypeDef *ADC_InitStruct);
00212
00217 void ADC_StructInit(ADC_InitTypeDef *ADC_InitStruct);
00218
00225 uint16_t ADC_GetIRQFlag(ADC_TypeDef *ADCx, uint16_t INT_flag);
00226
00232 void ADC_ClearIRQFlag(ADC_TypeDef *ADCx, uint16_t INT_flag);
00233
00240 void ADC_SetGain(ADC_TypeDef *ADCx, ADC_DAT_x DATNum, ADC_GAIN_x gainx);
00241
00248 s16 ADC_GetConversionValue(ADC_TypeDef *ADCx, ADC_DAT_x DATNum);
00249
00256 int32_t ADC_GetConversionValueS32(ADC_TypeDef *ADCx, ADC_DAT_x DATNum);
00257
00262 void ADC_SoftTrig(ADC_TypeDef *ADCx);
00263
00268 void ADC_SoftIdleTrig(ADC_TypeDef *ADCx);
00269
00276 void ADC_SetChn(ADC_TypeDef *ADCx, ADC_DAT_x datx, ADC_CHN_x chnx);
00277
00283 void ADC_SetRange(ADC_TypeDef *ADCx, ADC_RANGE_x rangex);
00284 #endif // __LKS32MC09X_ADC_H__

```

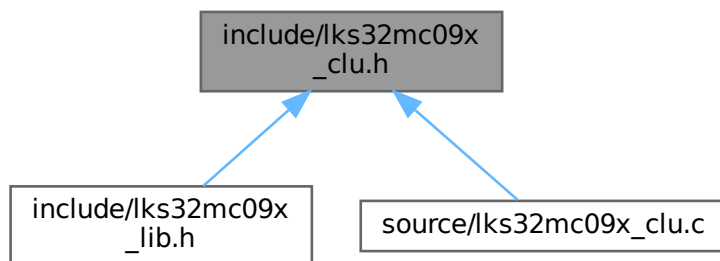
3.3 include/lks32mc09x_clu.h 文件参考

```
#include "lks32mc09x_reg.h"
```

lks32mc09x_clu.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 [CLU_InitTypeDef](#)

宏定义

- #define [CLU_MODEL_CLU0](#) BIT0
CLU0 CL0里面有4个CLU 这里用BIT0 BIT1 BIT2 BIT3做区分 不可修改
- #define [CLU_MODEL_CLU1](#) BIT1

CLU1

- #define `CLU_MODEL_CLU2` BIT2

CLU2

- #define `CLU_MODEL_CLU3` BIT3

CLU3

- #define `CLU_MXA_CLU0_CLU0OUT` 0
- #define `CLU_MXA_CLU0_CLU1OUT` 1
- #define `CLU_MXA_CLU0_CLU2OUT` 2
- #define `CLU_MXA_CLU0_CLU3OUT` 3
- #define `CLU_MXA_CLU0_TIMER0ch0ovfl` 4
- #define `CLU_MXA_CLU0_TIMER1ch0ovfl` 5
- #define `CLU_MXA_CLU0_PWM0` 6
- #define `CLU_MXA_CLU0_PWM1` 7
- #define `CLU_MXA_CLU0_P0_0` 8
- #define `CLU_MXA_CLU0_P0_3` 9
- #define `CLU_MXA_CLU0_P0_5` 10
- #define `CLU_MXA_CLU0_P0_7` 11
- #define `CLU_MXA_CLU0_P2_12` 12
- #define `CLU_MXA_CLU0_P0_12` 13
- #define `CLU_MXA_CLU0_P0_14` 14
- #define `CLU_MXA_CLU0_P1_0` 15
- #define `CLU_MXA_CLU1_CLU0OUT` 0
- #define `CLU_MXA_CLU1_CLU1OUT` 1
- #define `CLU_MXA_CLU1_CLU2OUT` 2
- #define `CLU_MXA_CLU1_CLU3OUT` 3
- #define `CLU_MXA_CLU1_TIMER0ch1ovfl` 4
- #define `CLU_MXA_CLU1_TIMER1ch1ovfl` 5
- #define `CLU_MXA_CLU1_PWM2` 6
- #define `CLU_MXA_CLU1_PWM3` 7
- #define `CLU_MXA_CLU1_P0_0` 8
- #define `CLU_MXA_CLU1_P0_2` 9
- #define `CLU_MXA_CLU1_P0_5` 10
- #define `CLU_MXA_CLU1_P0_6` 11
- #define `CLU_MXA_CLU1_P2_12` 12
- #define `CLU_MXA_CLU1_P0_11` 13
- #define `CLU_MXA_CLU1_P0_14` 14
- #define `CLU_MXA_CLU1_P0_15` 15
- #define `CLU_MXA_CLU2_CLU0OUT` 0
- #define `CLU_MXA_CLU2_CLU1OUT` 1
- #define `CLU_MXA_CLU2_CLU2OUT` 2
- #define `CLU_MXA_CLU2_CLU3OUT` 3
- #define `CLU_MXA_CLU2_TIMER2ch0ovfl` 4
- #define `CLU_MXA_CLU2_TIMER3ch0ovfl` 5
- #define `CLU_MXA_CLU2_PWM0` 6
- #define `CLU_MXA_CLU2_PWM2` 7
- #define `CLU_MXA_CLU2_P2_8` 8
- #define `CLU_MXA_CLU2_P1_13` 9
- #define `CLU_MXA_CLU2_P1_15` 10
- #define `CLU_MXA_CLU2_P1_10` 11
- #define `CLU_MXA_CLU2_P2_1` 12
- #define `CLU_MXA_CLU2_P2_4` 13
- #define `CLU_MXA_CLU2_P2_6` 14
- #define `CLU_MXA_CLU2_P2_14` 15
- #define `CLU_MXA_CLU3_CLU0OUT` 0

- #define CLU_MXA_CLU3_CLU1OUT 1
- #define CLU_MXA_CLU3_CLU2OUT 2
- #define CLU_MXA_CLU3_CLU3OUT 3
- #define CLU_MXA_CLU3_TIMER2ch1ovfl 4
- #define CLU_MXA_CLU3_TIMER3ch1ovfl 5
- #define CLU_MXA_CLU3_PWM1 6
- #define CLU_MXA_CLU3_PWM3 7
- #define CLU_MXA_CLU3_P2_8 8
- #define CLU_MXA_CLU3_P2_7 9
- #define CLU_MXA_CLU3_P1_15 10
- #define CLU_MXA_CLU3_P2_0 11
- #define CLU_MXA_CLU3_P2_1 12
- #define CLU_MXA_CLU3_P2_3 13
- #define CLU_MXA_CLU3_P2_6 14
- #define CLU_MXA_CLU3_P2_13 15

CLU_MXB信号选择

- #define CLU_MXB_CLU0_CLU0OUT 0
- #define CLU_MXB_CLU0_CLU1OUT 1
- #define CLU_MXB_CLU0_CLU2OUT 2
- #define CLU_MXB_CLU0_CLU3OUT 3
- #define CLU_MXB_CLU0_ADBUSY0 4
- #define CLU_MXB_CLU0_CMP0_OUT 5
- #define CLU_MXB_CLU0_PWM3 6
- #define CLU_MXB_CLU0_SPIMOSI 7
- #define CLU_MXB_CLU0_P0_2 8
- #define CLU_MXB_CLU0_P0_4 9
- #define CLU_MXB_CLU0_P0_6 10
- #define CLU_MXB_CLU0_P2_11 11
- #define CLU_MXB_CLU0_P0_11 12
- #define CLU_MXB_CLU0_P0_13 13
- #define CLU_MXB_CLU0_P0_15 14
- #define CLU_MXB_CLU0_P2_7 15
- #define CLU_MXB_CLU1_CLU0OUT 0
- #define CLU_MXB_CLU1_CLU1OUT 1
- #define CLU_MXB_CLU1_CLU2OUT 2
- #define CLU_MXB_CLU1_CLU3OUT 3
- #define CLU_MXB_CLU1_ADBUSY0 4
- #define CLU_MXB_CLU1_CMP1_OUT 5
- #define CLU_MXB_CLU1_PWM1 6
- #define CLU_MXB_CLU1_SPIMISO 7
- #define CLU_MXB_CLU1_P0_3 8
- #define CLU_MXB_CLU1_P0_4 9
- #define CLU_MXB_CLU1_P0_7 10
- #define CLU_MXB_CLU1_P2_11 11
- #define CLU_MXB_CLU1_P0_12 12
- #define CLU_MXB_CLU1_P0_13 13
- #define CLU_MXB_CLU1_P1_0 14
- #define CLU_MXB_CLU1_P2_7 15
- #define CLU_MXB_CLU2_CLU0OUT 0
- #define CLU_MXB_CLU2_CLU1OUT 1
- #define CLU_MXB_CLU2_CLU2OUT 2
- #define CLU_MXB_CLU2_CLU3OUT 3
- #define CLU_MXB_CLU2_PWM1 4
- #define CLU_MXB_CLU2_PWM3 5

- #define CLU_MXB_CLU2_I2CSDA 6
- #define CLU_MXB_CLU2_SPICLK 7
- #define CLU_MXB_CLU2_P3_9 8
- #define CLU_MXB_CLU2_P1_14 9
- #define CLU_MXB_CLU2_P2_0 10
- #define CLU_MXB_CLU2_P1_11 11
- #define CLU_MXB_CLU2_P2_3 12
- #define CLU_MXB_CLU2_P2_5 13
- #define CLU_MXB_CLU2_P2_13 14
- #define CLU_MXB_CLU2_P2_15 15
- #define CLU_MXB_CLU3_CLU0OUT 0
- #define CLU_MXB_CLU3_CLU1OUT 1
- #define CLU_MXB_CLU3_CLU2OUT 2
- #define CLU_MXB_CLU3_CLU3OUT 3
- #define CLU_MXB_CLU3_PWM0 4
- #define CLU_MXB_CLU3_PWM2 5
- #define CLU_MXB_CLU3_I2CSCL 6
- #define CLU_MXB_CLU3_SPICSN 7
- #define CLU_MXB_CLU3_P1_13 8
- #define CLU_MXB_CLU3_P1_14 9
- #define CLU_MXB_CLU3_P1_10 10
- #define CLU_MXB_CLU3_P1_11 11
- #define CLU_MXB_CLU3_P2_4 12
- #define CLU_MXB_CLU3_P2_5 13
- #define CLU_MXB_CLU3_P2_14 14
- #define CLU_MXB_CLU3_P2_15 15
- #define CLU_FN_0 0x0
CLU真值表的最小相 0 注意最小相之间只能用 | (C语言按位或)
- #define CLU_FN_1 0xff
CLU真值表的最小相 1 注意最小相之间只能用 | (C语言按位或)
- #define CLU_FN_A 0xf0
CLU真值表的最小相 a 注意最小相之间只能用 | (C语言按位或)
- #define CLU_FN_NOTA 0x0f
CLU真值表的最小相 !a 注意最小相之间只能用 | (C语言按位或)
- #define CLU_FN_B 0xcc
CLU真值表的最小相 b 注意最小相之间只能用 | (C语言按位或)
- #define CLU_FN_NOTB 0x33
CLU真值表的最小相 !b 注意最小相之间只能用 | (C语言按位或)
- #define CLU_FN_C 0xaa
CLU真值表的最小相 c 注意最小相之间只能用 | (C语言按位或)
- #define CLU_FN_NOTC 0x55
CLU真值表的最小相 !c 注意最小相之间只能用 | (C语言按位或)
- #define CLU_FN_A_B 0xc0
CLU真值表的最小相 a&b 注意最小相之间只能用 | (C语言按位或)
- #define CLU_FN_NOTA_NOTB 0x03
CLU真值表的最小相 !a&!b 注意最小相之间只能用 | (C语言按位或)
- #define CLU_FN_NOTA_B 0x0c
CLU真值表的最小相 !a&b 注意最小相之间只能用 | (C语言按位或)
- #define CLU_FN_A_NOTB 0x30
CLU真值表的最小相 a&!b 注意最小相之间只能用 | (C语言按位或)
- #define CLU_FN_A_C 0xa0
CLU真值表的最小相 a&c 注意最小相之间只能用 | (C语言按位或)

- `#define CLU_FN_NOTA_NOTC 0x05`
CLU真值表的最小相 *!a&!c* 注意最小相之间只能用 | (C语言按位或)
- `#define CLU_FN_NOTA_C 0x0a`
CLU真值表的最小相 *!a&c* 注意最小相之间只能用 | (C语言按位或)
- `#define CLU_FN_A_NOTC 0x50`
CLU真值表的最小相 *a&!c* 注意最小相之间只能用 | (C语言按位或)
- `#define CLU_FN_B_C 0x88`
CLU真值表的最小相 *b&c* 注意最小相之间只能用 | (C语言按位或)
- `#define CLU_FN_NOTB_NOTC 0x11`
CLU真值表的最小相 *!b&!c* 注意最小相之间只能用 | (C语言按位或)
- `#define CLU_FN_NOTB_C 0x22`
CLU真值表的最小相 *!b&c* 注意最小相之间只能用 | (C语言按位或)
- `#define CLU_FN_B_NOTC 0x44`
CLU真值表的最小相 *b&!c* 注意最小相之间只能用 | (C语言按位或)
- `#define CLU_FN_NOTA_NOTB_NOTC 0x01`
CLU真值表的最小相 *!a&!b&!c* 注意最小相之间只能用 | (C语言按位或)
- `#define CLU_FN_NOTA_NOTB_C 0x02`
CLU真值表的最小相 *!a&!b&c* 注意最小相之间只能用 | (C语言按位或)
- `#define CLU_FN_NOTA_B_NOTC 0x04`
CLU真值表的最小相 *!a&b&!c* 注意最小相之间只能用 | (C语言按位或)
- `#define CLU_FN_NOTA_B_C 0x08`
CLU真值表的最小相 *!a&b&c* 注意最小相之间只能用 | (C语言按位或)
- `#define CLU_FN_A_NOTB_NOTC 0x10`
CLU真值表的最小相 *a&!b&!c* 注意最小相之间只能用 | (C语言按位或)
- `#define CLU_FN_A_NOTB_C 0x20`
CLU真值表的最小相 *a&!b&c* 注意最小相之间只能用 | (C语言按位或)
- `#define CLU_FN_A_B_NOTC 0x40`
CLU真值表的最小相 *a&b&!c* 注意最小相之间只能用 | (C语言按位或)
- `#define CLU_FN_A_B_C 0x80`
CLU真值表的最小相 *a&b&c* 注意最小相之间只能用 | (C语言按位或)

函数

- `void CLU_Init (CLU_TypeDef *CLUx, uint8_t CLU_MODEL_CLUx, CLU_InitTypeDef *CLU_InitStruct)`
CLU计算初始化
- `void CLU_StructInit (CLU_InitTypeDef *CLU_InitStruct)`
CLU初始化结构体初始化
- `void CLU_Enable (CLU_TypeDef *CLUx)`
CLU模块时钟使能
- `void CLU_Disable (CLU_TypeDef *CLUx)`
CLU模块时钟关闭
- `uint8_t CLU_ReadOutValue (CLU_TypeDef *CLUx, uint8_t CLU_MODEL_CLUx)`
获取CLU的输出值

3.3.1 宏定义说明

3.3.1.1 CLU_FN_0

```
#define CLU_FN_0 0x0
```

CLU真值表的最小相 0 注意最小相之间只能用 | (C语言按位或)

在文件 [lks32mc09x_clu.h](#) 第 163 行定义.

3.3.1.2 CLU_FN_1

```
#define CLU_FN_1 0xff
```

CLU真值表的最小相 1 注意最小相之间只能用 | （C语言按位或）

在文件 [lks32mc09x.clu.h](#) 第 164 行定义.

3.3.1.3 CLU_FN_A

```
#define CLU_FN_A 0xf0
```

CLU真值表的最小相 a 注意最小相之间只能用 | （C语言按位或）

在文件 [lks32mc09x.clu.h](#) 第 165 行定义.

3.3.1.4 CLU_FN_A_B

```
#define CLU_FN_A_B 0xc0
```

CLU真值表的最小相 a&b 注意最小相之间只能用 | （C语言按位或）

在文件 [lks32mc09x.clu.h](#) 第 171 行定义.

3.3.1.5 CLU_FN_A_B_C

```
#define CLU_FN_A_B_C 0x80
```

CLU真值表的最小相 a&b&c 注意最小相之间只能用 | （C语言按位或）

在文件 [lks32mc09x.clu.h](#) 第 190 行定义.

3.3.1.6 CLU_FN_A_B_NOTC

```
#define CLU_FN_A_B_NOTC 0x40
```

CLU真值表的最小相 a&b&!c 注意最小相之间只能用 | （C语言按位或）

在文件 [lks32mc09x.clu.h](#) 第 189 行定义.

3.3.1.7 CLU_FN_A_C

```
#define CLU_FN_A_C 0xa0
```

CLU真值表的最小相 a&c 注意最小相之间只能用 | （C语言按位或）

在文件 [lks32mc09x.clu.h](#) 第 175 行定义.

3.3.1.8 CLU_FN_A_NOTB

```
#define CLU_FN_A_NOTB 0x30
```

CLU真值表的最小相 $a \& b$ 注意最小相之间只能用 | (C语言按位或)

在文件 [lks32mc09x.clu.h](#) 第 174 行定义.

3.3.1.9 CLU_FN_A_NOTB_C

```
#define CLU_FN_A_NOTB_C 0x20
```

CLU真值表的最小相 $a \& b \& c$ 注意最小相之间只能用 | (C语言按位或)

在文件 [lks32mc09x.clu.h](#) 第 188 行定义.

3.3.1.10 CLU_FN_A_NOTB_NOTC

```
#define CLU_FN_A_NOTB_NOTC 0x10
```

CLU真值表的最小相 $a \& b \& !c$ 注意最小相之间只能用 | (C语言按位或)

在文件 [lks32mc09x.clu.h](#) 第 187 行定义.

3.3.1.11 CLU_FN_A_NOTC

```
#define CLU_FN_A_NOTC 0x50
```

CLU真值表的最小相 $a \& !c$ 注意最小相之间只能用 | (C语言按位或)

在文件 [lks32mc09x.clu.h](#) 第 178 行定义.

3.3.1.12 CLU_FN_B

```
#define CLU_FN_B 0xcc
```

CLU真值表的最小相 b 注意最小相之间只能用 | (C语言按位或)

在文件 [lks32mc09x.clu.h](#) 第 167 行定义.

3.3.1.13 CLU_FN_B_C

```
#define CLU_FN_B_C 0x88
```

CLU真值表的最小相 $b \& c$ 注意最小相之间只能用 | (C语言按位或)

在文件 [lks32mc09x.clu.h](#) 第 179 行定义.

3.3.1.14 CLU_FN_B_NOTC

```
#define CLU_FN_B_NOTC 0x44
```

CLU真值表的最小相 b&lc 注意最小相之间只能用 | （C语言按位或）

在文件 [lks32mc09x.clu.h](#) 第 182 行定义.

3.3.1.15 CLU_FN_C

```
#define CLU_FN_C 0xaa
```

CLU真值表的最小相 c 注意最小相之间只能用 | （C语言按位或）

在文件 [lks32mc09x.clu.h](#) 第 169 行定义.

3.3.1.16 CLU_FN_NOTA

```
#define CLU_FN_NOTA 0x0f
```

CLU真值表的最小相 !a 注意最小相之间只能用 | （C语言按位或）

在文件 [lks32mc09x.clu.h](#) 第 166 行定义.

3.3.1.17 CLU_FN_NOTA_B

```
#define CLU_FN_NOTA_B 0x0c
```

CLU真值表的最小相 !a&b 注意最小相之间只能用 | （C语言按位或）

在文件 [lks32mc09x.clu.h](#) 第 173 行定义.

3.3.1.18 CLU_FN_NOTA_B_C

```
#define CLU_FN_NOTA_B_C 0x08
```

CLU真值表的最小相 !a&b&c 注意最小相之间只能用 | （C语言按位或）

在文件 [lks32mc09x.clu.h](#) 第 186 行定义.

3.3.1.19 CLU_FN_NOTA_B_NOTC

```
#define CLU_FN_NOTA_B_NOTC 0x04
```

CLU真值表的最小相 !a&b&lc 注意最小相之间只能用 | （C语言按位或）

在文件 [lks32mc09x.clu.h](#) 第 185 行定义.

3.3.1.20 CLU_FN_NOTA_C

```
#define CLU_FN_NOTA_C 0x0a
```

CLU真值表的最小相 **!a&c** 注意最小相之间只能用 | （C语言按位或）

在文件 [lks32mc09x.clu.h](#) 第 177 行定义.

3.3.1.21 CLU_FN_NOTA_NOTB

```
#define CLU_FN_NOTA_NOTB 0x03
```

CLU真值表的最小相 **!a&!b** 注意最小相之间只能用 | （C语言按位或）

在文件 [lks32mc09x.clu.h](#) 第 172 行定义.

3.3.1.22 CLU_FN_NOTA_NOTB_C

```
#define CLU_FN_NOTA_NOTB_C 0x02
```

CLU真值表的最小相 **!a&!b&c** 注意最小相之间只能用 | （C语言按位或）

在文件 [lks32mc09x.clu.h](#) 第 184 行定义.

3.3.1.23 CLU_FN_NOTA_NOTB_NOTC

```
#define CLU_FN_NOTA_NOTB_NOTC 0x01
```

CLU真值表的最小相 **!a&!b&!c** 注意最小相之间只能用 | （C语言按位或）

在文件 [lks32mc09x.clu.h](#) 第 183 行定义.

3.3.1.24 CLU_FN_NOTA_NOTC

```
#define CLU_FN_NOTA_NOTC 0x05
```

CLU真值表的最小相 **!a&!c** 注意最小相之间只能用 | （C语言按位或）

在文件 [lks32mc09x.clu.h](#) 第 176 行定义.

3.3.1.25 CLU_FN_NOTB

```
#define CLU_FN_NOTB 0x33
```

CLU真值表的最小相 **!b** 注意最小相之间只能用 | （C语言按位或）

在文件 [lks32mc09x.clu.h](#) 第 168 行定义.

3.3.1.26 CLU_FN_NOTB_C

```
#define CLU_FN_NOTB_C 0x22
```

CLU真值表的最小相**lb&c** 注意最小相之间只能用|（C语言按位或）

在文件 [lks32mc09x.clu.h](#) 第 181 行定义.

3.3.1.27 CLU_FN_NOTB_NOTC

```
#define CLU_FN_NOTB_NOTC 0x11
```

CLU真值表的最小相**lb&lc** 注意最小相之间只能用|（C语言按位或）

在文件 [lks32mc09x.clu.h](#) 第 180 行定义.

3.3.1.28 CLU_FN_NOTC

```
#define CLU_FN_NOTC 0x55
```

CLU真值表的最小相**lc** 注意最小相之间只能用|（C语言按位或）

在文件 [lks32mc09x.clu.h](#) 第 170 行定义.

3.3.1.29 CLU_MODEL_CLU0

```
#define CLU_MODEL_CLU0 BIT0
```

CLU0 CL0里面有4个CLU 这里用BIT0 BIT1 BIT2 BIT3做区分 不可修改

在文件 [lks32mc09x.clu.h](#) 第 20 行定义.

3.3.1.30 CLU_MODEL_CLU1

```
#define CLU_MODEL_CLU1 BIT1
```

CLU1

在文件 [lks32mc09x.clu.h](#) 第 21 行定义.

3.3.1.31 CLU_MODEL_CLU2

```
#define CLU_MODEL_CLU2 BIT2
```

CLU2

在文件 [lks32mc09x.clu.h](#) 第 22 行定义.

3.3.1.32 CLU_MODEL_CLU3

```
#define CLU_MODEL_CLU3 BIT3
```

CLU3

CLU.MXA信号选择

在文件 [lks32mc09x.clu.h](#) 第 25 行定义.

3.3.1.33 CLU_MXA_CLU0_CLU0OUT

```
#define CLU_MXA_CLU0_CLU0OUT 0
```

在文件 [lks32mc09x.clu.h](#) 第 26 行定义.

3.3.1.34 CLU_MXA_CLU0_CLU1OUT

```
#define CLU_MXA_CLU0_CLU1OUT 1
```

在文件 [lks32mc09x.clu.h](#) 第 27 行定义.

3.3.1.35 CLU_MXA_CLU0_CLU2OUT

```
#define CLU_MXA_CLU0_CLU2OUT 2
```

在文件 [lks32mc09x.clu.h](#) 第 28 行定义.

3.3.1.36 CLU_MXA_CLU0_CLU3OUT

```
#define CLU_MXA_CLU0_CLU3OUT 3
```

在文件 [lks32mc09x.clu.h](#) 第 29 行定义.

3.3.1.37 CLU_MXA_CLU0_P0_0

```
#define CLU_MXA_CLU0_P0_0 8
```

在文件 [lks32mc09x.clu.h](#) 第 34 行定义.

3.3.1.38 CLU_MXA_CLU0_P0_12

```
#define CLU_MXA_CLU0_P0_12 13
```

在文件 [lks32mc09x.clu.h](#) 第 39 行定义.

3.3.1.39 CLU_MXA_CLU0_P0_14

```
#define CLU_MXA_CLU0_P0_14 14
```

在文件 [lks32mc09x_clu.h](#) 第 40 行定义.

3.3.1.40 CLU_MXA_CLU0_P0_3

```
#define CLU_MXA_CLU0_P0_3 9
```

在文件 [lks32mc09x_clu.h](#) 第 35 行定义.

3.3.1.41 CLU_MXA_CLU0_P0_5

```
#define CLU_MXA_CLU0_P0_5 10
```

在文件 [lks32mc09x_clu.h](#) 第 36 行定义.

3.3.1.42 CLU_MXA_CLU0_P0_7

```
#define CLU_MXA_CLU0_P0_7 11
```

在文件 [lks32mc09x_clu.h](#) 第 37 行定义.

3.3.1.43 CLU_MXA_CLU0_P1_0

```
#define CLU_MXA_CLU0_P1_0 15
```

在文件 [lks32mc09x_clu.h](#) 第 41 行定义.

3.3.1.44 CLU_MXA_CLU0_P2_12

```
#define CLU_MXA_CLU0_P2_12 12
```

在文件 [lks32mc09x_clu.h](#) 第 38 行定义.

3.3.1.45 CLU_MXA_CLU0_PWM0

```
#define CLU_MXA_CLU0_PWM0 6
```

在文件 [lks32mc09x_clu.h](#) 第 32 行定义.

3.3.1.46 CLU_MXA_CLU0_PWM1

```
#define CLU_MXA_CLU0_PWM1 7
```

在文件 [lks32mc09x_clu.h](#) 第 33 行定义.

3.3.1.47 CLU_MXA_CLU0_TIMER0ch0ovfl

```
#define CLU_MXA_CLU0_TIMER0ch0ovfl 4
```

在文件 [lks32mc09x.clu.h](#) 第 30 行定义.

3.3.1.48 CLU_MXA_CLU0_TIMER1ch0ovfl

```
#define CLU_MXA_CLU0_TIMER1ch0ovfl 5
```

在文件 [lks32mc09x.clu.h](#) 第 31 行定义.

3.3.1.49 CLU_MXA_CLU1_CLU0OUT

```
#define CLU_MXA_CLU1_CLU0OUT 0
```

在文件 [lks32mc09x.clu.h](#) 第 43 行定义.

3.3.1.50 CLU_MXA_CLU1_CLU1OUT

```
#define CLU_MXA_CLU1_CLU1OUT 1
```

在文件 [lks32mc09x.clu.h](#) 第 44 行定义.

3.3.1.51 CLU_MXA_CLU1_CLU2OUT

```
#define CLU_MXA_CLU1_CLU2OUT 2
```

在文件 [lks32mc09x.clu.h](#) 第 45 行定义.

3.3.1.52 CLU_MXA_CLU1_CLU3OUT

```
#define CLU_MXA_CLU1_CLU3OUT 3
```

在文件 [lks32mc09x.clu.h](#) 第 46 行定义.

3.3.1.53 CLU_MXA_CLU1_P0_0

```
#define CLU_MXA_CLU1_P0_0 8
```

在文件 [lks32mc09x.clu.h](#) 第 51 行定义.

3.3.1.54 CLU_MXA_CLU1_P0_11

```
#define CLU_MXA_CLU1_P0_11 13
```

在文件 [lks32mc09x.clu.h](#) 第 56 行定义.

3.3.1.55 CLU_MXA_CLU1_P0_14

```
#define CLU_MXA_CLU1_P0_14 14
```

在文件 [lks32mc09x.clu.h](#) 第 57 行定义.

3.3.1.56 CLU_MXA_CLU1_P0_15

```
#define CLU_MXA_CLU1_P0_15 15
```

在文件 [lks32mc09x.clu.h](#) 第 58 行定义.

3.3.1.57 CLU_MXA_CLU1_P0_2

```
#define CLU_MXA_CLU1_P0_2 9
```

在文件 [lks32mc09x.clu.h](#) 第 52 行定义.

3.3.1.58 CLU_MXA_CLU1_P0_5

```
#define CLU_MXA_CLU1_P0_5 10
```

在文件 [lks32mc09x.clu.h](#) 第 53 行定义.

3.3.1.59 CLU_MXA_CLU1_P0_6

```
#define CLU_MXA_CLU1_P0_6 11
```

在文件 [lks32mc09x.clu.h](#) 第 54 行定义.

3.3.1.60 CLU_MXA_CLU1_P2_12

```
#define CLU_MXA_CLU1_P2_12 12
```

在文件 [lks32mc09x.clu.h](#) 第 55 行定义.

3.3.1.61 CLU_MXA_CLU1_PWM2

```
#define CLU_MXA_CLU1_PWM2 6
```

在文件 [lks32mc09x.clu.h](#) 第 49 行定义.

3.3.1.62 CLU_MXA_CLU1_PWM3

```
#define CLU_MXA_CLU1_PWM3 7
```

在文件 [lks32mc09x.clu.h](#) 第 50 行定义.

3.3.1.63 CLU_MXA_CLU1_TIMER0ch1ovfl

```
#define CLU_MXA_CLU1_TIMER0ch1ovfl 4
```

在文件 [lks32mc09x.clu.h](#) 第 47 行定义.

3.3.1.64 CLU_MXA_CLU1_TIMER1ch1ovfl

```
#define CLU_MXA_CLU1_TIMER1ch1ovfl 5
```

在文件 [lks32mc09x.clu.h](#) 第 48 行定义.

3.3.1.65 CLU_MXA_CLU2_CLU0OUT

```
#define CLU_MXA_CLU2_CLU0OUT 0
```

在文件 [lks32mc09x.clu.h](#) 第 60 行定义.

3.3.1.66 CLU_MXA_CLU2_CLU1OUT

```
#define CLU_MXA_CLU2_CLU1OUT 1
```

在文件 [lks32mc09x.clu.h](#) 第 61 行定义.

3.3.1.67 CLU_MXA_CLU2_CLU2OUT

```
#define CLU_MXA_CLU2_CLU2OUT 2
```

在文件 [lks32mc09x.clu.h](#) 第 62 行定义.

3.3.1.68 CLU_MXA_CLU2_CLU3OUT

```
#define CLU_MXA_CLU2_CLU3OUT 3
```

在文件 [lks32mc09x.clu.h](#) 第 63 行定义.

3.3.1.69 CLU_MXA_CLU2_P1_10

```
#define CLU_MXA_CLU2_P1_10 11
```

在文件 [lks32mc09x.clu.h](#) 第 71 行定义.

3.3.1.70 CLU_MXA_CLU2_P1_13

```
#define CLU_MXA_CLU2_P1_13 9
```

在文件 [lks32mc09x.clu.h](#) 第 69 行定义.

3.3.1.71 CLU_MXA_CLU2_P1_15

```
#define CLU_MXA_CLU2_P1_15 10
```

在文件 [lks32mc09x.clu.h](#) 第 70 行定义.

3.3.1.72 CLU_MXA_CLU2_P2_1

```
#define CLU_MXA_CLU2_P2_1 12
```

在文件 [lks32mc09x.clu.h](#) 第 72 行定义.

3.3.1.73 CLU_MXA_CLU2_P2_14

```
#define CLU_MXA_CLU2_P2_14 15
```

在文件 [lks32mc09x.clu.h](#) 第 75 行定义.

3.3.1.74 CLU_MXA_CLU2_P2_4

```
#define CLU_MXA_CLU2_P2_4 13
```

在文件 [lks32mc09x.clu.h](#) 第 73 行定义.

3.3.1.75 CLU_MXA_CLU2_P2_6

```
#define CLU_MXA_CLU2_P2_6 14
```

在文件 [lks32mc09x.clu.h](#) 第 74 行定义.

3.3.1.76 CLU_MXA_CLU2_P2_8

```
#define CLU_MXA_CLU2_P2_8 8
```

在文件 [lks32mc09x.clu.h](#) 第 68 行定义.

3.3.1.77 CLU_MXA_CLU2_PWM0

```
#define CLU_MXA_CLU2_PWM0 6
```

在文件 [lks32mc09x.clu.h](#) 第 66 行定义.

3.3.1.78 CLU_MXA_CLU2_PWM2

```
#define CLU_MXA_CLU2_PWM2 7
```

在文件 [lks32mc09x.clu.h](#) 第 67 行定义.

3.3.1.79 CLU_MXA_CLU2_TIMER2ch0ovfl

```
#define CLU_MXA_CLU2_TIMER2ch0ovfl 4
```

在文件 [lks32mc09x.clu.h](#) 第 64 行定义.

3.3.1.80 CLU_MXA_CLU2_TIMER3ch0ovfl

```
#define CLU_MXA_CLU2_TIMER3ch0ovfl 5
```

在文件 [lks32mc09x.clu.h](#) 第 65 行定义.

3.3.1.81 CLU_MXA_CLU3_CLU0OUT

```
#define CLU_MXA_CLU3_CLU0OUT 0
```

在文件 [lks32mc09x.clu.h](#) 第 77 行定义.

3.3.1.82 CLU_MXA_CLU3_CLU1OUT

```
#define CLU_MXA_CLU3_CLU1OUT 1
```

在文件 [lks32mc09x.clu.h](#) 第 78 行定义.

3.3.1.83 CLU_MXA_CLU3_CLU2OUT

```
#define CLU_MXA_CLU3_CLU2OUT 2
```

在文件 [lks32mc09x.clu.h](#) 第 79 行定义.

3.3.1.84 CLU_MXA_CLU3_CLU3OUT

```
#define CLU_MXA_CLU3_CLU3OUT 3
```

在文件 [lks32mc09x.clu.h](#) 第 80 行定义.

3.3.1.85 CLU_MXA_CLU3_P1_15

```
#define CLU_MXA_CLU3_P1_15 10
```

在文件 [lks32mc09x.clu.h](#) 第 87 行定义.

3.3.1.86 CLU_MXA_CLU3_P2_0

```
#define CLU_MXA_CLU3_P2_0 11
```

在文件 [lks32mc09x.clu.h](#) 第 88 行定义.

3.3.1.87 CLU_MXA_CLU3_P2_1

```
#define CLU_MXA_CLU3_P2_1 12
```

在文件 [lks32mc09x.clu.h](#) 第 89 行定义.

3.3.1.88 CLU_MXA_CLU3_P2_13

```
#define CLU_MXA_CLU3_P2_13 15
```

CLU_MXB信号选择

在文件 [lks32mc09x.clu.h](#) 第 94 行定义.

3.3.1.89 CLU_MXA_CLU3_P2_3

```
#define CLU_MXA_CLU3_P2_3 13
```

在文件 [lks32mc09x.clu.h](#) 第 90 行定义.

3.3.1.90 CLU_MXA_CLU3_P2_6

```
#define CLU_MXA_CLU3_P2_6 14
```

在文件 [lks32mc09x.clu.h](#) 第 91 行定义.

3.3.1.91 CLU_MXA_CLU3_P2_7

```
#define CLU_MXA_CLU3_P2_7 9
```

在文件 [lks32mc09x.clu.h](#) 第 86 行定义.

3.3.1.92 CLU_MXA_CLU3_P2_8

```
#define CLU_MXA_CLU3_P2_8 8
```

在文件 [lks32mc09x.clu.h](#) 第 85 行定义.

3.3.1.93 CLU_MXA_CLU3_PWM1

```
#define CLU_MXA_CLU3_PWM1 6
```

在文件 [lks32mc09x.clu.h](#) 第 83 行定义.

3.3.1.94 CLU_MXA_CLU3_PWM3

```
#define CLU_MXA_CLU3_PWM3 7
```

在文件 [lks32mc09x.clu.h](#) 第 84 行定义.

3.3.1.95 CLU_MXA_CLU3_TIMER2ch1ovfl

```
#define CLU_MXA_CLU3_TIMER2ch1ovfl 4
```

在文件 [lks32mc09x.clu.h](#) 第 81 行定义.

3.3.1.96 CLU_MXA_CLU3_TIMER3ch1ovfl

```
#define CLU_MXA_CLU3_TIMER3ch1ovfl 5
```

在文件 [lks32mc09x.clu.h](#) 第 82 行定义.

3.3.1.97 CLU_MXB_CLU0_ADBUSY0

```
#define CLU_MXB_CLU0_ADBUSY0 4
```

在文件 [lks32mc09x.clu.h](#) 第 99 行定义.

3.3.1.98 CLU_MXB_CLU0_CLU0OUT

```
#define CLU_MXB_CLU0_CLU0OUT 0
```

在文件 [lks32mc09x.clu.h](#) 第 95 行定义.

3.3.1.99 CLU_MXB_CLU0_CLU1OUT

```
#define CLU_MXB_CLU0_CLU1OUT 1
```

在文件 [lks32mc09x.clu.h](#) 第 96 行定义.

3.3.1.100 CLU_MXB_CLU0_CLU2OUT

```
#define CLU_MXB_CLU0_CLU2OUT 2
```

在文件 [lks32mc09x.clu.h](#) 第 97 行定义.

3.3.1.101 CLU_MXB_CLU0_CLU3OUT

```
#define CLU_MXB_CLU0_CLU3OUT 3
```

在文件 [lks32mc09x.clu.h](#) 第 98 行定义.

3.3.1.102 CLU_MXB_CLU0_CMP0_OUT

```
#define CLU_MXB_CLU0_CMP0_OUT 5
```

在文件 [lks32mc09x.clu.h](#) 第 100 行定义.

3.3.1.103 CLU_MXB_CLU0_P0_11

```
#define CLU_MXB_CLU0_P0_11 12
```

在文件 [lks32mc09x.clu.h](#) 第 107 行定义.

3.3.1.104 CLU_MXB_CLU0_P0_13

```
#define CLU_MXB_CLU0_P0_13 13
```

在文件 [lks32mc09x.clu.h](#) 第 108 行定义.

3.3.1.105 CLU_MXB_CLU0_P0_15

```
#define CLU_MXB_CLU0_P0_15 14
```

在文件 [lks32mc09x.clu.h](#) 第 109 行定义.

3.3.1.106 CLU_MXB_CLU0_P0_2

```
#define CLU_MXB_CLU0_P0_2 8
```

在文件 [lks32mc09x.clu.h](#) 第 103 行定义.

3.3.1.107 CLU_MXB_CLU0_P0_4

```
#define CLU_MXB_CLU0_P0_4 9
```

在文件 [lks32mc09x.clu.h](#) 第 104 行定义.

3.3.1.108 CLU_MXB_CLU0_P0_6

```
#define CLU_MXB_CLU0_P0_6 10
```

在文件 [lks32mc09x.clu.h](#) 第 105 行定义.

3.3.1.109 CLU_MXB_CLU0_P2_11

```
#define CLU_MXB_CLU0_P2_11 11
```

在文件 [lks32mc09x.clu.h](#) 第 106 行定义.

3.3.1.110 CLU_MXB_CLU0_P2_7

```
#define CLU_MXB_CLU0_P2_7 15
```

在文件 [lks32mc09x_clu.h](#) 第 110 行定义.

3.3.1.111 CLU_MXB_CLU0_PWM3

```
#define CLU_MXB_CLU0_PWM3 6
```

在文件 [lks32mc09x_clu.h](#) 第 101 行定义.

3.3.1.112 CLU_MXB_CLU0_SPIMOSI

```
#define CLU_MXB_CLU0_SPIMOSI 7
```

在文件 [lks32mc09x_clu.h](#) 第 102 行定义.

3.3.1.113 CLU_MXB_CLU1_ADBUSY0

```
#define CLU_MXB_CLU1_ADBUSY0 4
```

在文件 [lks32mc09x_clu.h](#) 第 116 行定义.

3.3.1.114 CLU_MXB_CLU1_CLU0OUT

```
#define CLU_MXB_CLU1_CLU0OUT 0
```

在文件 [lks32mc09x_clu.h](#) 第 112 行定义.

3.3.1.115 CLU_MXB_CLU1_CLU1OUT

```
#define CLU_MXB_CLU1_CLU1OUT 1
```

在文件 [lks32mc09x_clu.h](#) 第 113 行定义.

3.3.1.116 CLU_MXB_CLU1_CLU2OUT

```
#define CLU_MXB_CLU1_CLU2OUT 2
```

在文件 [lks32mc09x_clu.h](#) 第 114 行定义.

3.3.1.117 CLU_MXB_CLU1_CLU3OUT

```
#define CLU_MXB_CLU1_CLU3OUT 3
```

在文件 [lks32mc09x_clu.h](#) 第 115 行定义.

3.3.1.118 CLU_MXB_CLU1_CMP1_OUT

```
#define CLU_MXB_CLU1_CMP1_OUT 5
```

在文件 [lks32mc09x.clu.h](#) 第 117 行定义.

3.3.1.119 CLU_MXB_CLU1_P0_12

```
#define CLU_MXB_CLU1_P0_12 12
```

在文件 [lks32mc09x.clu.h](#) 第 124 行定义.

3.3.1.120 CLU_MXB_CLU1_P0_13

```
#define CLU_MXB_CLU1_P0_13 13
```

在文件 [lks32mc09x.clu.h](#) 第 125 行定义.

3.3.1.121 CLU_MXB_CLU1_P0_3

```
#define CLU_MXB_CLU1_P0_3 8
```

在文件 [lks32mc09x.clu.h](#) 第 120 行定义.

3.3.1.122 CLU_MXB_CLU1_P0_4

```
#define CLU_MXB_CLU1_P0_4 9
```

在文件 [lks32mc09x.clu.h](#) 第 121 行定义.

3.3.1.123 CLU_MXB_CLU1_P0_7

```
#define CLU_MXB_CLU1_P0_7 10
```

在文件 [lks32mc09x.clu.h](#) 第 122 行定义.

3.3.1.124 CLU_MXB_CLU1_P1_0

```
#define CLU_MXB_CLU1_P1_0 14
```

在文件 [lks32mc09x.clu.h](#) 第 126 行定义.

3.3.1.125 CLU_MXB_CLU1_P2_11

```
#define CLU_MXB_CLU1_P2_11 11
```

在文件 [lks32mc09x.clu.h](#) 第 123 行定义.

3.3.1.126 CLU_MXB_CLU1_P2_7

```
#define CLU_MXB_CLU1_P2_7 15
```

在文件 [lks32mc09x.clu.h](#) 第 127 行定义.

3.3.1.127 CLU_MXB_CLU1_PWM1

```
#define CLU_MXB_CLU1_PWM1 6
```

在文件 [lks32mc09x.clu.h](#) 第 118 行定义.

3.3.1.128 CLU_MXB_CLU1_SPIMISO

```
#define CLU_MXB_CLU1_SPIMISO 7
```

在文件 [lks32mc09x.clu.h](#) 第 119 行定义.

3.3.1.129 CLU_MXB_CLU2_CLU0OUT

```
#define CLU_MXB_CLU2_CLU0OUT 0
```

在文件 [lks32mc09x.clu.h](#) 第 129 行定义.

3.3.1.130 CLU_MXB_CLU2_CLU1OUT

```
#define CLU_MXB_CLU2_CLU1OUT 1
```

在文件 [lks32mc09x.clu.h](#) 第 130 行定义.

3.3.1.131 CLU_MXB_CLU2_CLU2OUT

```
#define CLU_MXB_CLU2_CLU2OUT 2
```

在文件 [lks32mc09x.clu.h](#) 第 131 行定义.

3.3.1.132 CLU_MXB_CLU2_CLU3OUT

```
#define CLU_MXB_CLU2_CLU3OUT 3
```

在文件 [lks32mc09x.clu.h](#) 第 132 行定义.

3.3.1.133 CLU_MXB_CLU2_I2CSDA

```
#define CLU_MXB_CLU2_I2CSDA 6
```

在文件 [lks32mc09x.clu.h](#) 第 135 行定义.

3.3.1.134 CLU_MXB_CLU2_P1_11

```
#define CLU_MXB_CLU2_P1_11 11
```

在文件 [lks32mc09x.clu.h](#) 第 140 行定义.

3.3.1.135 CLU_MXB_CLU2_P1_14

```
#define CLU_MXB_CLU2_P1_14 9
```

在文件 [lks32mc09x.clu.h](#) 第 138 行定义.

3.3.1.136 CLU_MXB_CLU2_P2_0

```
#define CLU_MXB_CLU2_P2_0 10
```

在文件 [lks32mc09x.clu.h](#) 第 139 行定义.

3.3.1.137 CLU_MXB_CLU2_P2_13

```
#define CLU_MXB_CLU2_P2_13 14
```

在文件 [lks32mc09x.clu.h](#) 第 143 行定义.

3.3.1.138 CLU_MXB_CLU2_P2_15

```
#define CLU_MXB_CLU2_P2_15 15
```

在文件 [lks32mc09x.clu.h](#) 第 144 行定义.

3.3.1.139 CLU_MXB_CLU2_P2_3

```
#define CLU_MXB_CLU2_P2_3 12
```

在文件 [lks32mc09x.clu.h](#) 第 141 行定义.

3.3.1.140 CLU_MXB_CLU2_P2_5

```
#define CLU_MXB_CLU2_P2_5 13
```

在文件 [lks32mc09x.clu.h](#) 第 142 行定义.

3.3.1.141 CLU_MXB_CLU2_P3_9

```
#define CLU_MXB_CLU2_P3_9 8
```

在文件 [lks32mc09x.clu.h](#) 第 137 行定义.

3.3.1.142 CLU_MXB_CLU2_PWM1

```
#define CLU_MXB_CLU2_PWM1 4
```

在文件 [lks32mc09x_clu.h](#) 第 133 行定义.

3.3.1.143 CLU_MXB_CLU2_PWM3

```
#define CLU_MXB_CLU2_PWM3 5
```

在文件 [lks32mc09x_clu.h](#) 第 134 行定义.

3.3.1.144 CLU_MXB_CLU2_SPICLK

```
#define CLU_MXB_CLU2_SPICLK 7
```

在文件 [lks32mc09x_clu.h](#) 第 136 行定义.

3.3.1.145 CLU_MXB_CLU3_CLU0OUT

```
#define CLU_MXB_CLU3_CLU0OUT 0
```

在文件 [lks32mc09x_clu.h](#) 第 146 行定义.

3.3.1.146 CLU_MXB_CLU3_CLU1OUT

```
#define CLU_MXB_CLU3_CLU1OUT 1
```

在文件 [lks32mc09x_clu.h](#) 第 147 行定义.

3.3.1.147 CLU_MXB_CLU3_CLU2OUT

```
#define CLU_MXB_CLU3_CLU2OUT 2
```

在文件 [lks32mc09x_clu.h](#) 第 148 行定义.

3.3.1.148 CLU_MXB_CLU3_CLU3OUT

```
#define CLU_MXB_CLU3_CLU3OUT 3
```

在文件 [lks32mc09x_clu.h](#) 第 149 行定义.

3.3.1.149 CLU_MXB_CLU3_I2CSCL

```
#define CLU_MXB_CLU3_I2CSCL 6
```

在文件 [lks32mc09x_clu.h](#) 第 152 行定义.

3.3.1.150 CLU_MXB_CLU3_P1_10

```
#define CLU_MXB_CLU3_P1_10 10
```

在文件 [lks32mc09x_clu.h](#) 第 156 行定义.

3.3.1.151 CLU_MXB_CLU3_P1_11

```
#define CLU_MXB_CLU3_P1_11 11
```

在文件 [lks32mc09x_clu.h](#) 第 157 行定义.

3.3.1.152 CLU_MXB_CLU3_P1_13

```
#define CLU_MXB_CLU3_P1_13 8
```

在文件 [lks32mc09x_clu.h](#) 第 154 行定义.

3.3.1.153 CLU_MXB_CLU3_P1_14

```
#define CLU_MXB_CLU3_P1_14 9
```

在文件 [lks32mc09x_clu.h](#) 第 155 行定义.

3.3.1.154 CLU_MXB_CLU3_P2_14

```
#define CLU_MXB_CLU3_P2_14 14
```

在文件 [lks32mc09x_clu.h](#) 第 160 行定义.

3.3.1.155 CLU_MXB_CLU3_P2_15

```
#define CLU_MXB_CLU3_P2_15 15
```

在文件 [lks32mc09x_clu.h](#) 第 161 行定义.

3.3.1.156 CLU_MXB_CLU3_P2_4

```
#define CLU_MXB_CLU3_P2_4 12
```

在文件 [lks32mc09x_clu.h](#) 第 158 行定义.

3.3.1.157 CLU_MXB_CLU3_P2_5

```
#define CLU_MXB_CLU3_P2_5 13
```

在文件 [lks32mc09x_clu.h](#) 第 159 行定义.

3.3.1.158 CLU_MXB_CLU3_PWM0

```
#define CLU_MXB_CLU3_PWM0 4
```

在文件 [lks32mc09x.clu.h](#) 第 150 行定义.

3.3.1.159 CLU_MXB_CLU3_PWM2

```
#define CLU_MXB_CLU3_PWM2 5
```

在文件 [lks32mc09x.clu.h](#) 第 151 行定义.

3.3.1.160 CLU_MXB_CLU3_SPICSN

```
#define CLU_MXB_CLU3_SPICSN 7
```

在文件 [lks32mc09x.clu.h](#) 第 153 行定义.

3.3.2 函数说明

3.3.2.1 CLU_Disable()

```
void CLU_Disable (
    CLU_TypeDef * CLUx )
```

CLU模块时钟关闭

在文件 [lks32mc09x.clu.c](#) 第 79 行定义.

函数调用图:



3.3.2.2 CLU_Enable()

```
void CLU_Enable (
    CLU_TypeDef * CLUx )
```

CLU模块时钟使能

在文件 [lks32mc09x-clu.c](#) 第 68 行定义.

函数调用图:



这是这个函数的调用关系图:



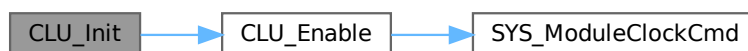
3.3.2.3 CLU_Init()

```
void CLU_Init (
    CLU_TypeDef * CLUx,
    uint8_t CLU_MODEL_CLUx,
    CLU_InitTypeDef * CLU_InitStruct )
```

CLU计算初始化

在文件 [lks32mc09x-clu.c](#) 第 8 行定义.

函数调用图:



3.3.2.4 CLU_ReadOutValue()

```
uint8_t CLU_ReadOutValue (
    CLU_TypeDef * CLUX,
    uint8_t CLU_MODEL_CLUX )
```

获取CLU的输出值

在文件 [lks32mc09x.clu.c](#) 第 90 行定义.

3.3.2.5 CLU_StructInit()

```
void CLU_StructInit (
    CLU_InitTypeDef * CLU_InitStruct )
```

CLU初始化结构体初始化

在文件 [lks32mc09x.clu.c](#) 第 56 行定义.

3.4 lks32mc09x_clu.h

[浏览该文件的文档.](#)

```
00001 #ifndef __LKS32MC09X_CLU_H__
00002 #define __LKS32MC09X_CLU_H__
00003
00004 #include "lks32mc09x_reg.h"
00005
00006 typedef struct
00007 {
00008     uint32_t EN;
00009     uint32_t FIE;
00010     uint32_t RIE;
00011     uint32_t MXA;
00012     uint32_t MXB;
00013     uint32_t FN;
00014     uint32_t OUTSEL;
00015     uint32_t OEN;
00016     uint32_t CLKINV;
00017     uint32_t CLKSEL;
00018 } CLU_InitTypeDef;
00019
00020 #define CLU_MODEL_CLU0 BIT0
00021 #define CLU_MODEL_CLU1 BIT1
00022 #define CLU_MODEL_CLU2 BIT2
00023 #define CLU_MODEL_CLU3 BIT3
00024
00026 #define CLU_MXA_CLU0_CLU0OUT 0
00027 #define CLU_MXA_CLU0_CLU1OUT 1
00028 #define CLU_MXA_CLU0_CLU2OUT 2
00029 #define CLU_MXA_CLU0_CLU3OUT 3
00030 #define CLU_MXA_CLU0_TIMER0ch0ovfl 4
00031 #define CLU_MXA_CLU0_TIMER1ch0ovfl 5
00032 #define CLU_MXA_CLU0_PWM0 6
00033 #define CLU_MXA_CLU0_PWM1 7
00034 #define CLU_MXA_CLU0_P0_0 8
00035 #define CLU_MXA_CLU0_P0_3 9
00036 #define CLU_MXA_CLU0_P0_5 10
00037 #define CLU_MXA_CLU0_P0_7 11
00038 #define CLU_MXA_CLU0_P2_12 12
00039 #define CLU_MXA_CLU0_P0_12 13
00040 #define CLU_MXA_CLU0_P0_14 14
00041 #define CLU_MXA_CLU0_P1_0 15
00042
00043 #define CLU_MXA_CLU1_CLU0OUT 0
00044 #define CLU_MXA_CLU1_CLU1OUT 1
00045 #define CLU_MXA_CLU1_CLU2OUT 2
00046 #define CLU_MXA_CLU1_CLU3OUT 3
00047 #define CLU_MXA_CLU1_TIMER0ch1ovfl 4
00048 #define CLU_MXA_CLU1_TIMER1ch1ovfl 5
```

```

00049 #define CLU.MXA.CLU1.PWM2        6
00050 #define CLU.MXA.CLU1.PWM3        7
00051 #define CLU.MXA.CLU1.P0.0        8
00052 #define CLU.MXA.CLU1.P0.2        9
00053 #define CLU.MXA.CLU1.P0.5       10
00054 #define CLU.MXA.CLU1.P0.6       11
00055 #define CLU.MXA.CLU1.P2.12      12
00056 #define CLU.MXA.CLU1.P0.11      13
00057 #define CLU.MXA.CLU1.P0.14      14
00058 #define CLU.MXA.CLU1.P0.15      15
00059
00060 #define CLU.MXA.CLU2.CLU0OUT      0
00061 #define CLU.MXA.CLU2.CLU1OUT      1
00062 #define CLU.MXA.CLU2.CLU2OUT      2
00063 #define CLU.MXA.CLU2.CLU3OUT      3
00064 #define CLU.MXA.CLU2.TIMER2ch0ovf1 4
00065 #define CLU.MXA.CLU2.TIMER3ch0ovf1 5
00066 #define CLU.MXA.CLU2.PWM0        6
00067 #define CLU.MXA.CLU2.PWM2        7
00068 #define CLU.MXA.CLU2.P2.8        8
00069 #define CLU.MXA.CLU2.P1.13       9
00070 #define CLU.MXA.CLU2.P1.15      10
00071 #define CLU.MXA.CLU2.P1.10      11
00072 #define CLU.MXA.CLU2.P2.1       12
00073 #define CLU.MXA.CLU2.P2.4       13
00074 #define CLU.MXA.CLU2.P2.6       14
00075 #define CLU.MXA.CLU2.P2.14      15
00076
00077 #define CLU.MXA.CLU3.CLU0OUT      0
00078 #define CLU.MXA.CLU3.CLU1OUT      1
00079 #define CLU.MXA.CLU3.CLU2OUT      2
00080 #define CLU.MXA.CLU3.CLU3OUT      3
00081 #define CLU.MXA.CLU3.TIMER2ch1ovf1 4
00082 #define CLU.MXA.CLU3.TIMER3ch1ovf1 5
00083 #define CLU.MXA.CLU3.PWM1        6
00084 #define CLU.MXA.CLU3.PWM3        7
00085 #define CLU.MXA.CLU3.P2.8        8
00086 #define CLU.MXA.CLU3.P2.7        9
00087 #define CLU.MXA.CLU3.P1.15      10
00088 #define CLU.MXA.CLU3.P2.0       11
00089 #define CLU.MXA.CLU3.P2.1       12
00090 #define CLU.MXA.CLU3.P2.3       13
00091 #define CLU.MXA.CLU3.P2.6       14
00092 #define CLU.MXA.CLU3.P2.13      15
00093
00095 #define CLU.MXB.CLU0.CLU0OUT      0
00096 #define CLU.MXB.CLU0.CLU1OUT      1
00097 #define CLU.MXB.CLU0.CLU2OUT      2
00098 #define CLU.MXB.CLU0.CLU3OUT      3
00099 #define CLU.MXB.CLU0.ADBUSY0      4
00100 #define CLU.MXB.CLU0.CMP0.OUT     5
00101 #define CLU.MXB.CLU0.PWM3        6
00102 #define CLU.MXB.CLU0.SPIMOSI      7
00103 #define CLU.MXB.CLU0.P0.2        8
00104 #define CLU.MXB.CLU0.P0.4        9
00105 #define CLU.MXB.CLU0.P0.6       10
00106 #define CLU.MXB.CLU0.P2.11      11
00107 #define CLU.MXB.CLU0.P0.11      12
00108 #define CLU.MXB.CLU0.P0.13      13
00109 #define CLU.MXB.CLU0.P0.15      14
00110 #define CLU.MXB.CLU0.P2.7       15
00111
00112 #define CLU.MXB.CLU1.CLU0OUT      0
00113 #define CLU.MXB.CLU1.CLU1OUT      1
00114 #define CLU.MXB.CLU1.CLU2OUT      2
00115 #define CLU.MXB.CLU1.CLU3OUT      3
00116 #define CLU.MXB.CLU1.ADBUSY0      4
00117 #define CLU.MXB.CLU1.CMP1.OUT     5
00118 #define CLU.MXB.CLU1.PWM1        6
00119 #define CLU.MXB.CLU1.SPIMISO      7
00120 #define CLU.MXB.CLU1.P0.3        8
00121 #define CLU.MXB.CLU1.P0.4        9
00122 #define CLU.MXB.CLU1.P0.7       10
00123 #define CLU.MXB.CLU1.P2.11      11
00124 #define CLU.MXB.CLU1.P0.12      12
00125 #define CLU.MXB.CLU1.P0.13      13
00126 #define CLU.MXB.CLU1.P1.0       14
00127 #define CLU.MXB.CLU1.P2.7       15
00128
00129 #define CLU.MXB.CLU2.CLU0OUT      0
00130 #define CLU.MXB.CLU2.CLU1OUT      1
00131 #define CLU.MXB.CLU2.CLU2OUT      2
00132 #define CLU.MXB.CLU2.CLU3OUT      3
00133 #define CLU.MXB.CLU2.PWM1        4
00134 #define CLU.MXB.CLU2.PWM3        5
00135 #define CLU.MXB.CLU2.I2CSDA      6
00136 #define CLU.MXB.CLU2.SPICKLK      7

```

```

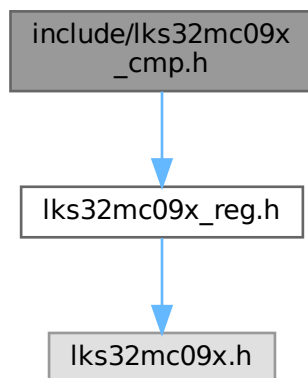
00137 #define CLU_MXB_CLU2_P3_9      8
00138 #define CLU_MXB_CLU2_P1_14      9
00139 #define CLU_MXB_CLU2_P2_0      10
00140 #define CLU_MXB_CLU2_P1_11     11
00141 #define CLU_MXB_CLU2_P2_3      12
00142 #define CLU_MXB_CLU2_P2_5      13
00143 #define CLU_MXB_CLU2_P2_13     14
00144 #define CLU_MXB_CLU2_P2_15     15
00145
00146 #define CLU_MXB_CLU3_CLU0OUT    0
00147 #define CLU_MXB_CLU3_CLU1OUT    1
00148 #define CLU_MXB_CLU3_CLU2OUT    2
00149 #define CLU_MXB_CLU3_CLU3OUT    3
00150 #define CLU_MXB_CLU3_PWM0       4
00151 #define CLU_MXB_CLU3_PWM2       5
00152 #define CLU_MXB_CLU3_I2CSCL     6
00153 #define CLU_MXB_CLU3_SPICSN     7
00154 #define CLU_MXB_CLU3_P1_13      8
00155 #define CLU_MXB_CLU3_P1_14      9
00156 #define CLU_MXB_CLU3_P1_10     10
00157 #define CLU_MXB_CLU3_P1_11     11
00158 #define CLU_MXB_CLU3_P2_4       12
00159 #define CLU_MXB_CLU3_P2_5       13
00160 #define CLU_MXB_CLU3_P2_14      14
00161 #define CLU_MXB_CLU3_P2_15      15
00162
00163 #define CLU_FN_0                 0x0
00164 #define CLU_FN_1                 0xff
00165 #define CLU_FN_A                 0xf0
00166 #define CLU_FN_NOTA              0x0f
00167 #define CLU_FN_B                 0xcc
00168 #define CLU_FN_NOTB              0x33
00169 #define CLU_FN_C                 0xaa
00170 #define CLU_FN_NOTC              0x55
00171 #define CLU_FN_A_B               0xc0
00172 #define CLU_FN_NOTA_NOTB         0x03
00173 #define CLU_FN_NOTA_B            0x0c
00174 #define CLU_FN_A_NOTB            0x30
00175 #define CLU_FN_A_C               0xa0
00176 #define CLU_FN_NOTA_NOTC         0x05
00177 #define CLU_FN_NOTA_C            0x0a
00178 #define CLU_FN_A_NOTC            0x50
00179 #define CLU_FN_B_C               0x88
00180 #define CLU_FN_NOTB_NOTC         0x11
00181 #define CLU_FN_NOTB_C            0x22
00182 #define CLU_FN_B_NOTC            0x44
00183 #define CLU_FN_NOTA_NOTB_NOTC    0x01
00184 #define CLU_FN_NOTA_NOTB_C       0x02
00185 #define CLU_FN_NOTA_B_NOTC       0x04
00186 #define CLU_FN_NOTA_B_C          0x08
00187 #define CLU_FN_A_NOTB_NOTC       0x10
00188 #define CLU_FN_A_NOTB_C          0x20
00189 #define CLU_FN_A_B_NOTC          0x40
00190 #define CLU_FN_A_B_C             0x80
00191
00192 void CLU_Init(CLU_TypeDef *CLUx, uint8_t CLU_MODEL_CLUx, CLU_InitTypeDef *CLU_InitStruct);
00193 void CLU_StructInit(CLU_InitTypeDef *CLU_InitStruct);
00194
00195 void CLU_Enable(CLU_TypeDef *CLUx);
00196 void CLU_Disable(CLU_TypeDef *CLUx);
00197
00198 uint8_t CLU_ReadOutV alue(CLU_TypeDef *CLUx, uint8_t CLU_MODEL_CLUx);
00199 #endif /* _LKS32MC08x_CLU_H */

```

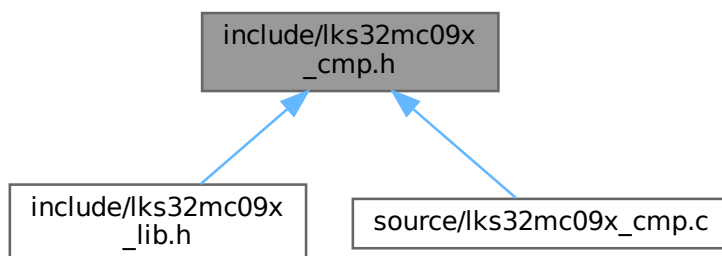
3.5 include/lks32mc09x_cmp.h 文件参考

```
#include "lks32mc09x_reg.h"
```

lks32mc09x_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:



结构体

- struct [CMP_SingleTypeDef](#)
*CMP*初始化结构体句柄，初始化时请定义该句柄，并用其它地址来传参
- struct [CMP_InitTypeDef](#)

宏定义

- #define [CMP_CHN_0](#) 0
- #define [CMP_CHN_1](#) 1

- #define CMP_SELFP_1_OPA2_IP 0
0000: OPA2.IP
- #define CMP_SELFP_1_OPA3_IP 1
0001: OPA3.IP
- #define CMP_SELFP_1_OPA2_OUTP 2
0010: OPA2.OUTP, OPA2 差分输出正端
- #define CMP_SELFP_1_OPA3_OUTP 3
0011: OPA3.OUTP, OPA3 差分输出正端
- #define CMP_SELFP_1_CMP1_IP0 4
0100: CMP1.IP0
- #define CMP_SELFP_1_CMP1_IP1 5
0101: CMP1.IP1
- #define CMP_SELFP_1_CMP1_IP2 6
0110: CMP1.IP2
- #define CMP_SELFP_1_CMP1_IP3 7
0111: CMP1.IP3
- #define CMP_SELFP_1_DAC1_OUT 9
1001: DAC1.OUT
- #define CMP_SELFP_1_OPA_OUT_CMP 10
1010: OPA_OUT_CMP
- #define CMP_SELFP_1_CMP0_IP1 11
1011: CMP0.IP1
- #define CMP_SELFP_1_CMP0_IP2 12
1100: CMP0.IP2
- #define CMP_SELFP_1_CMP0_IP3 13
1101: CMP0.IP3
- #define CMP_SELFP_0_OPA0_IP 0
0000: OPA0.IP
- #define CMP_SELFP_0_OPA1_IP 1
0001: OPA1.IP
- #define CMP_SELFP_0_OPA0_OUTP 2
0010: OPA0.OUTP, OPA0 差分输出正端
- #define CMP_SELFP_0_OPA1_OUTP 3
0011: OPA1.OUTP, OPA1 差分输出正端
- #define CMP_SELFP_0_CMP0_IP0 4
0100: CMP0.IP0
- #define CMP_SELFP_0_CMP0_IP1 5
0101: CMP0.IP1
- #define CMP_SELFP_0_CMP0_IP2 6
0110: IP_CMP0.IP2
- #define CMP_SELFP_0_CMP0_IP3 7
0111: IP_CMP0.IP3
- #define CMP_SELFP_0_CMP0_IP4 8
1000: IP_CMP0.IP4
- #define CMP_SELFP_0_DAC0_OUT 9
1001: DAC0.OUT
- #define CMP_SELFP_0_OUT_OPA_CMP 10
1010: OUT_OPA_CMP
- #define CMP_SELFP_0_CMP1_IP1 11
1011: IP_CMP1.IP1
- #define CMP_SELFP_0_CMP1_IP2 12

```

1100: IP_CMP1_IP2
• #define CMP_SEL0_CMP1_IP3 13
1101: IP_CMP1_IP3
• #define CMP_HYS_20mV 0
• #define CMP_HYS_0mV 1
• #define CMP_SELN_1_IN_CMP1 0
000:  $\dot{e}_i \dot{z}$  IN_CMP1
• #define CMP_SELN_1_VBGP 1
001:  $\dot{e}_i \dot{z}$  VBGP
• #define CMP_SELN_1_DAC1 2
010:  $\dot{e}_i \dot{z}$  DAC1 OUT
• #define CMP_SELN_1_DAC0 3
011:  $\dot{e}_i \dot{z}$  DAC0 OUT
• #define CMP_SELN_1_HS_MID1 4
100: 连 HS_MID1 (中性点的电阻阻值增加为约 30k)
• #define CMP_SELN_0_IN_CMP0 0
000:  $\dot{e}_i \dot{z}$  IN_CMP0
• #define CMP_SELN_0_VBGP 1
001:  $\dot{e}_i \dot{z}$  VBGP
• #define CMP_SELN_0_DAC1 2
010:  $\dot{e}_i \dot{z}$  DAC1 OUT
• #define CMP_SELN_0_DAC0 3
011:  $\dot{e}_i \dot{z}$  DAC0 OUT
• #define CMP_SELN_0_HS_MID0 4
100: 连 HS_MID0 (中性点的电阻阻值增加为约 30k)
• #define CMP_IF_CMP0 BIT0
• #define CMP_IF_CMP1 BIT1
• #define CMP_IE_CMP0 BIT0
• #define CMP_IE_CMP1 BIT1
• #define CMP_RE_CMP0 BIT8
• #define CMP_RE_CMP1 BIT9
• #define CMP_CLK_DIV_0 0
不分频
• #define CMP_CLK_DIV_1 1
1分频
• #define CMP_CLK_DIV_2 2
2分频
• #define CMP_CLK_DIV_3 3
3分频
• #define CMP_CLK_DIV_4 4
4分频
• #define CMP_SIGNAL_NONE 0
• #define CMP_SIGNAL_EDGE_P (BIT2 | BIT1)
• #define CMP_SIGNAL_EDGE_N (BIT2 | BIT1 | BIT0)
• #define CMP_SIGNAL_LEVEL_H (BIT1)
• #define CMP_SIGNAL_LEVEL_L (BIT1 | BIT0)
• #define CMP_WIN_SIGNAL_NONE 0
• #define CMP_WIN_SIGNAL_MCPWM0_0 (BIT0)
• #define CMP_WIN_SIGNAL_MCPWM0_1 (BIT1)
• #define CMP_WIN_SIGNAL_MCPWM0_2 (BIT2)
• #define CMP_WIN_SIGNAL_MCPWM0_3 (BIT3)
• #define CMP_WIN_LEVEL_H (1)
• #define CMP_WIN_LEVEL_L (0)

```

函数

- void **CMP_Enable** (uint8_t CMP_CHN_x)
使能比较器
- void **CMP_Disable** (uint8_t CMP_CHN_x)
失能比较器
- void **CMP_SetSignal** (uint8_t CMP_CHN_x, uint8_t CMP_SELN_x_x, uint8_t CMP_SELP_x_x)
设置比较器信号来源
- void **CMP_SetHysteresis** (uint8_t CMP_HYS_x)
设置比较器回差(两个比较器共用同一个回差)
- void **CMP_SetWindowSignal** (uint8_t CMP_CHN_x, uint8_t CMP_WIN_x, uint8_t CMP_WIN_LEVEL_x)
设置比较器开窗信号
- void **CMP_SetFilterWidth** (uint8_t CMP_CLK_DIV_x, uint16_t CMP0_FIL_x, uint16_t CMP1_FIL_x)
设置比较器的滤波宽度
- void **CMP_SetIRQSignal** (uint8_t CMP_CHN_x, uint8_t CMP_SIGNAL_x)
设置比较器中断信号
- void **CMP_ClearIRQFlag** (uint8_t CMP_CHN_x)
清除CMP中断标志
- uint8_t **CMP_GetIRQFlag** (uint8_t CMP_CHN_x)
读取CMP中断标志
- uint8_t **CMP_GetFilterValue** (uint8_t CMP_CHN_x)
读取滤波后的比较器值
- uint8_t **CMP_GetValue** (uint8_t CMP_CHN_x)
读取未滤波的比较器值
- void **CMP_StructInit** (CMP_InitTypeDef *CMP_InitStruct)
CMP结构体初始化
- void **CMP_Init** (CMP_InitTypeDef *CMP_InitStruct)
CMP初始化

3.5.1 宏定义说明

3.5.1.1 CMP_CHN_0

```
#define CMP_CHN_0 0
```

在文件 [lks32mc09x_cmp.h](#) 第 33 行定义.

3.5.1.2 CMP_CHN_1

```
#define CMP_CHN_1 1
```

在文件 [lks32mc09x_cmp.h](#) 第 34 行定义.

3.5.1.3 CMP_CLK_DIV_0

```
#define CMP_CLK_DIV_0 0
```

不分频

在文件 [lks32mc09x_cmp.h](#) 第 98 行定义.

3.5.1.4 CMP_CLK_DIV_1

```
#define CMP_CLK_DIV_1 1
```

1分频

在文件 [lks32mc09x_cmp.h](#) 第 99 行定义.

3.5.1.5 CMP_CLK_DIV_2

```
#define CMP_CLK_DIV_2 2
```

2分频

在文件 [lks32mc09x_cmp.h](#) 第 100 行定义.

3.5.1.6 CMP_CLK_DIV_3

```
#define CMP_CLK_DIV_3 3
```

3分频

在文件 [lks32mc09x_cmp.h](#) 第 101 行定义.

3.5.1.7 CMP_CLK_DIV_4

```
#define CMP_CLK_DIV_4 4
```

4分频

在文件 [lks32mc09x_cmp.h](#) 第 102 行定义.

3.5.1.8 CMP_HYS_0mV

```
#define CMP_HYS_0mV 1
```

在文件 [lks32mc09x_cmp.h](#) 第 69 行定义.

3.5.1.9 CMP_HYS_20mV

```
#define CMP_HYS_20mV 0
```

在文件 [lks32mc09x_cmp.h](#) 第 68 行定义.

3.5.1.10 CMP_IE_CMP0

```
#define CMP_IE_CMP0 BIT0
```

在文件 [lks32mc09x_cmp.h](#) 第 90 行定义.

3.5.1.11 CMP_IE_CMP1

```
#define CMP_IE_CMP1 BIT1
```

在文件 [lks32mc09x_cmp.h](#) 第 91 行定义.

3.5.1.12 CMP_IF_CMP0

```
#define CMP_IF_CMP0 BIT0
```

在文件 [lks32mc09x_cmp.h](#) 第 87 行定义.

3.5.1.13 CMP_IF_CMP1

```
#define CMP_IF_CMP1 BIT1
```

在文件 [lks32mc09x_cmp.h](#) 第 88 行定义.

3.5.1.14 CMP_RE_CMP0

```
#define CMP_RE_CMP0 BIT8
```

在文件 [lks32mc09x_cmp.h](#) 第 93 行定义.

3.5.1.15 CMP_RE_CMP1

```
#define CMP_RE_CMP1 BIT9
```

在文件 [lks32mc09x_cmp.h](#) 第 94 行定义.

3.5.1.16 CMP_SELN_0_DAC0

```
#define CMP_SELN_0_DAC0 3
```

011: DAC0 OUT

在文件 [lks32mc09x_cmp.h](#) 第 82 行定义.

3.5.1.17 CMP_SELN_0_DAC1

```
#define CMP_SELN_0_DAC1 2
```

010: DAC1 OUT

在文件 [lks32mc09x_cmp.h](#) 第 81 行定义.

3.5.1.18 CMP_SELN_0_HS_MID0

```
#define CMP_SELN_0_HS_MID0 4
```

100: 连 HS_MID0（中性点的电阻阻值增加为约 30k）

在文件 [lks32mc09x_cmp.h](#) 第 83 行定义.

3.5.1.19 CMP_SELN_0_IN_CMP0

```
#define CMP_SELN_0_IN_CMP0 0
```

000: 0 IN_CMP0

在文件 [lks32mc09x_cmp.h](#) 第 79 行定义.

3.5.1.20 CMP_SELN_0_VBGP

```
#define CMP_SELN_0_VBGP 1
```

001: 0 VBGP

在文件 [lks32mc09x_cmp.h](#) 第 80 行定义.

3.5.1.21 CMP_SELN_1_DAC0

```
#define CMP_SELN_1_DAC0 3
```

011: 0 DAC0 OUT

在文件 [lks32mc09x_cmp.h](#) 第 75 行定义.

3.5.1.22 CMP_SELN_1_DAC1

```
#define CMP_SELN_1_DAC1 2
```

010: 0 DAC1 OUT

在文件 [lks32mc09x_cmp.h](#) 第 74 行定义.

3.5.1.23 CMP_SELN_1_HS_MID1

```
#define CMP_SELN_1_HS_MID1 4
```

100: 连 HS_MID1（中性点的电阻阻值增加为约 30k）

在文件 [lks32mc09x_cmp.h](#) 第 76 行定义.

3.5.1.24 CMP_SELN_1_IN_CMP1

```
#define CMP_SELN_1_IN_CMP1 0
```

000: 000 IN_CMP1

在文件 [lks32mc09x_cmp.h](#) 第 72 行定义.

3.5.1.25 CMP_SELN_1_VBGP

```
#define CMP_SELN_1_VBGP 1
```

001: 001 VBGP

在文件 [lks32mc09x_cmp.h](#) 第 73 行定义.

3.5.1.26 CMP_SEL0_CMP0_IP0

```
#define CMP_SEL0_CMP0_IP0 4
```

0100: CMP0_IP0

在文件 [lks32mc09x_cmp.h](#) 第 56 行定义.

3.5.1.27 CMP_SEL0_CMP0_IP1

```
#define CMP_SEL0_CMP0_IP1 5
```

0101: CMP0_IP1

在文件 [lks32mc09x_cmp.h](#) 第 57 行定义.

3.5.1.28 CMP_SEL0_CMP0_IP2

```
#define CMP_SEL0_CMP0_IP2 6
```

0110: IP_CMP0_IP2

在文件 [lks32mc09x_cmp.h](#) 第 58 行定义.

3.5.1.29 CMP_SEL0_CMP0_IP3

```
#define CMP_SEL0_CMP0_IP3 7
```

0111: IP_CMP0_IP3

在文件 [lks32mc09x_cmp.h](#) 第 59 行定义.

3.5.1.30 CMP_SELP_0_CMP0_IP4

```
#define CMP_SELP_0_CMP0_IP4 8
```

1000: IP_CMP0_IP4

在文件 [lks32mc09x_cmp.h](#) 第 60 行定义.

3.5.1.31 CMP_SELP_0_CMP1_IP1

```
#define CMP_SELP_0_CMP1_IP1 11
```

1011: IP_CMP1_IP1

在文件 [lks32mc09x_cmp.h](#) 第 63 行定义.

3.5.1.32 CMP_SELP_0_CMP1_IP2

```
#define CMP_SELP_0_CMP1_IP2 12
```

1100: IP_CMP1_IP2

在文件 [lks32mc09x_cmp.h](#) 第 64 行定义.

3.5.1.33 CMP_SELP_0_CMP1_IP3

```
#define CMP_SELP_0_CMP1_IP3 13
```

1101: IP_CMP1_IP3

在文件 [lks32mc09x_cmp.h](#) 第 65 行定义.

3.5.1.34 CMP_SELP_0_DAC0_OUT

```
#define CMP_SELP_0_DAC0_OUT 9
```

1001: DAC0_OUT

在文件 [lks32mc09x_cmp.h](#) 第 61 行定义.

3.5.1.35 CMP_SELP_0_OPA0_IP

```
#define CMP_SELP_0_OPA0_IP 0
```

0000: OPA0_IP

在文件 [lks32mc09x_cmp.h](#) 第 52 行定义.

3.5.1.36 CMP_SEL_P0_OPA0_OUTP

```
#define CMP_SEL_P0_OPA0_OUTP 2
```

0010: OPA0_OUTP, OPA0 差分输出正端

在文件 [lks32mc09x_cmp.h](#) 第 54 行定义.

3.5.1.37 CMP_SEL_P0_OPA1_IP

```
#define CMP_SEL_P0_OPA1_IP 1
```

0001: OPA1_IP

在文件 [lks32mc09x_cmp.h](#) 第 53 行定义.

3.5.1.38 CMP_SEL_P0_OPA1_OUTP

```
#define CMP_SEL_P0_OPA1_OUTP 3
```

0011: OPA1_OUTP, OPA1 差分输出正端

在文件 [lks32mc09x_cmp.h](#) 第 55 行定义.

3.5.1.39 CMP_SEL_P0_OUT_OPA_CMP

```
#define CMP_SEL_P0_OUT_OPA_CMP 10
```

1010: OUT_OPA_CMP

在文件 [lks32mc09x_cmp.h](#) 第 62 行定义.

3.5.1.40 CMP_SEL_P1_CMP0_IP1

```
#define CMP_SEL_P1_CMP0_IP1 11
```

1011: CMP0_IP1

在文件 [lks32mc09x_cmp.h](#) 第 47 行定义.

3.5.1.41 CMP_SEL_P1_CMP0_IP2

```
#define CMP_SEL_P1_CMP0_IP2 12
```

1100: CMP0_IP2

在文件 [lks32mc09x_cmp.h](#) 第 48 行定义.

3.5.1.42 CMP_SEL_P1_CMP0_IP3

```
#define CMP_SEL_P1_CMP0_IP3 13
```

1101: CMP0_IP3

在文件 [lks32mc09x_cmp.h](#) 第 49 行定义.

3.5.1.43 CMP_SEL_P1_CMP1_IP0

```
#define CMP_SEL_P1_CMP1_IP0 4
```

0100: CMP1_IP0

在文件 [lks32mc09x_cmp.h](#) 第 41 行定义.

3.5.1.44 CMP_SEL_P1_CMP1_IP1

```
#define CMP_SEL_P1_CMP1_IP1 5
```

0101: CMP1_IP1

在文件 [lks32mc09x_cmp.h](#) 第 42 行定义.

3.5.1.45 CMP_SEL_P1_CMP1_IP2

```
#define CMP_SEL_P1_CMP1_IP2 6
```

0110: CMP1_IP2

在文件 [lks32mc09x_cmp.h](#) 第 43 行定义.

3.5.1.46 CMP_SEL_P1_CMP1_IP3

```
#define CMP_SEL_P1_CMP1_IP3 7
```

0111: CMP1_IP3

在文件 [lks32mc09x_cmp.h](#) 第 44 行定义.

3.5.1.47 CMP_SEL_P1_DAC1_OUT

```
#define CMP_SEL_P1_DAC1_OUT 9
```

1001: DAC1_OUT

在文件 [lks32mc09x_cmp.h](#) 第 45 行定义.

3.5.1.48 CMP_SELFP_1_OPA2_IP

```
#define CMP_SELFP_1_OPA2_IP 0
```

0000: OPA2_IP

在文件 [lks32mc09x_cmp.h](#) 第 37 行定义.

3.5.1.49 CMP_SELFP_1_OPA2_OUTP

```
#define CMP_SELFP_1_OPA2_OUTP 2
```

0010: OPA2_OUTP , OPA2 差分输出正端

在文件 [lks32mc09x_cmp.h](#) 第 39 行定义.

3.5.1.50 CMP_SELFP_1_OPA3_IP

```
#define CMP_SELFP_1_OPA3_IP 1
```

0001: OPA3_IP

在文件 [lks32mc09x_cmp.h](#) 第 38 行定义.

3.5.1.51 CMP_SELFP_1_OPA3_OUTP

```
#define CMP_SELFP_1_OPA3_OUTP 3
```

0011: OPA3_OUTP , OPA3 差分输出正端

在文件 [lks32mc09x_cmp.h](#) 第 40 行定义.

3.5.1.52 CMP_SELFP_1_OPA_OUT_CMP

```
#define CMP_SELFP_1_OPA_OUT_CMP 10
```

1010: OPA_OUT_CMP

在文件 [lks32mc09x_cmp.h](#) 第 46 行定义.

3.5.1.53 CMP_SIGNAL_EDGE_N

```
#define CMP_SIGNAL_EDGE_N (BIT2 | BIT1 | BIT0)
```

在文件 [lks32mc09x_cmp.h](#) 第 107 行定义.

3.5.1.54 CMP_SIGNAL_EDGE_P

```
#define CMP_SIGNAL_EDGE_P (BIT2 | BIT1)
```

在文件 [lks32mc09x_cmp.h](#) 第 106 行定义.

3.5.1.55 CMP_SIGNAL_LEVEL_H

```
#define CMP_SIGNAL_LEVEL_H (BIT1)
```

在文件 [lks32mc09x_cmp.h](#) 第 108 行定义.

3.5.1.56 CMP_SIGNAL_LEVEL_L

```
#define CMP_SIGNAL_LEVEL_L (BIT1 | BIT0)
```

在文件 [lks32mc09x_cmp.h](#) 第 109 行定义.

3.5.1.57 CMP_SIGNAL_NONE

```
#define CMP_SIGNAL_NONE 0
```

在文件 [lks32mc09x_cmp.h](#) 第 105 行定义.

3.5.1.58 CMP_WIN_LEVEL_H

```
#define CMP_WIN_LEVEL_H (1)
```

在文件 [lks32mc09x_cmp.h](#) 第 117 行定义.

3.5.1.59 CMP_WIN_LEVEL_L

```
#define CMP_WIN_LEVEL_L (0)
```

在文件 [lks32mc09x_cmp.h](#) 第 118 行定义.

3.5.1.60 CMP_WIN_SIGNAL_MCPWM0_0

```
#define CMP_WIN_SIGNAL_MCPWM0_0 (BIT0)
```

在文件 [lks32mc09x_cmp.h](#) 第 113 行定义.

3.5.1.61 CMP_WIN_SIGNAL_MCPWM0_1

```
#define CMP_WIN_SIGNAL_MCPWM0_1 (BIT1)
```

在文件 [lks32mc09x_cmp.h](#) 第 114 行定义.

3.5.1.62 CMP_WIN_SIGNAL_MCPWM0_2

```
#define CMP_WIN_SIGNAL_MCPWM0_2 (BIT2)
```

在文件 [lks32mc09x_cmp.h](#) 第 115 行定义.

3.5.1.63 CMP_WIN_SIGNAL_MCPWM0_3

```
#define CMP_WIN_SIGNAL_MCPWM0_3 (BIT3)
```

在文件 [lks32mc09x_cmp.h](#) 第 116 行定义.

3.5.1.64 CMP_WIN_SIGNAL_NONE

```
#define CMP_WIN_SIGNAL_NONE 0
```

在文件 [lks32mc09x_cmp.h](#) 第 112 行定义.

3.5.2 函数说明**3.5.2.1 CMP_ClearIRQFlag()**

```
void CMP_ClearIRQFlag (
    uint8_t CMP_CHN_x )
```

清除CMP中断标志

参数

<i>CMP_↔ CHN_x</i>	比较器通道选择, CMP_CHN.0, CMP_CHN.1

在文件 [lks32mc09x_cmp.c](#) 第 236 行定义.

3.5.2.2 CMP_Disable()

```
void CMP_Disable (
    uint8_t CMP_CHN_x )
```

失能比较器

参数

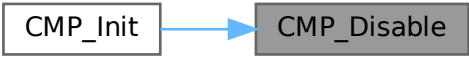
<i>CMP_↔ CHN_x</i>	比较器通道选择, CMP_CHN.0, CMP_CHN.1

在文件 [lks32mc09x_cmp.c](#) 第 38 行定义.

函数调用图:



这是这个函数的调用关系图:



3.5.2.3 CMP_Enable()

```
void CMP_Enable (
    uint8_t CMP_CHN_x )
```

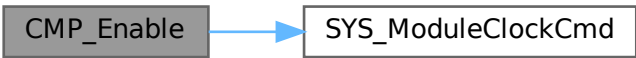
使能比较器

参数

<i>CMP_CHN_x</i>	比较器通道选择, CMP_CHN_0, CMP_CHN_1

在文件 [lks32mc09x_cmp.c](#) 第 10 行定义.

函数调用图:



这是这个函数的调用关系图:



3.5.2.4 CMP_GetFilterValue()

```
uint8_t CMP_GetFilterValue (
    uint8_t CMP_CHN_x )
```

读取滤波后的比较器值

返回

- : 滤波后的比较器值
- : 滤波后的比较器值 0或1

在文件 [lks32mc09x_cmp.c](#) 第 281 行定义.

3.5.2.5 CMP_GetIRQFlag()

```
uint8_t CMP_GetIRQFlag (
    uint8_t CMP_CHN_x )
```

读取CMP中断标志

参数

<i>CMP_↔</i> <i>CHN_x</i>	比较器通道选择, CMP_CHN_0, CMP_CHN_1

返回

- : 中断标志状态

参数

<i>CMP_↔</i> <i>CHN_x</i>	比较器通道选择, CMP_CHN_0, CMP_CHN_1

返回

: 中断标志状态 0或1

在文件 [lks32mc09x_cmp.c](#) 第 261 行定义.

3.5.2.6 CMP_GetValue()

```
uint8_t CMP_GetValue (
    uint8_t CMP_CHN_x )
```

读取未滤波的比较器值

返回

: 未滤波的比较器值

: 未滤波的比较器值 0或1

在文件 [lks32mc09x_cmp.c](#) 第 301 行定义.

3.5.2.7 CMP_Init()

```
void CMP_Init (
    CMP\_InitTypeDef * CMP_InitStruct )
```

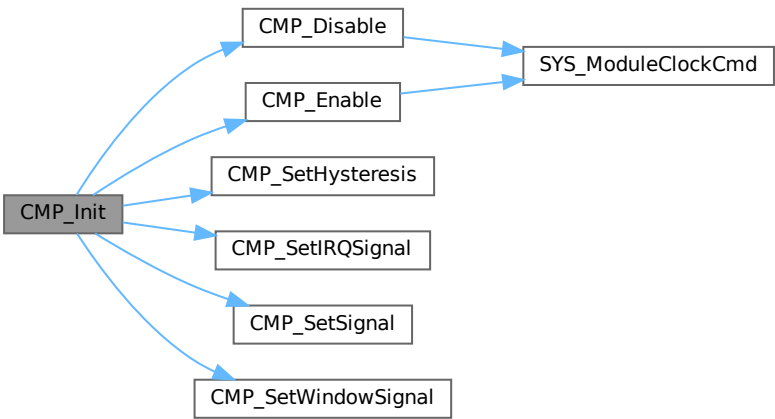
CMP初始化

参数

<i>CMP_InitStruct</i>	指向CMP初始化结构体的指针

在文件 [lks32mc09x_cmp.c](#) 第 332 行定义.

函数调用图:



3.5.2.8 CMP_SetFilterWidth()

```
void CMP_SetFilterWidth (
    uint8_t CMP_CLK_DIV_x,
    uint16_t CMP0_FIL_x,
    uint16_t CMP1_FIL_x )
```

设置比较器的滤波宽度

参数

<i>CMP_CLK_↔ DIV_x</i>	比较器时钟分频系数，CMP_CLK_DIV_0, CMP_CLK_DIV_1, CMP_CLK_DIV_2, CMP_CLK_DIV_3, CMP_CLK_DIV_4
<i>CMP0_FIL_x</i>	比较器0滤波宽度，取值范围0-2047，0表示关闭信号的输入
<i>CMP1_FIL_x</i>	比较器1滤波宽度，取值范围0-2047，0表示关闭信号的输入

在文件 [lks32mc09x_cmp.c](#) 第 149 行定义.

3.5.2.9 CMP_SetHysteresis()

```
void CMP_SetHysteresis (
    uint8_t CMP_HYS_x )
```

设置比较器回差(两个比较器共用同一个回差)

参数

<i>CMP_↔ HYS_x</i>	回差选择，CMP_HYS_20mV, CMP_HYS_0mV

在文件 [lks32mc09x_cmp.c](#) 第 104 行定义.

这是这个函数的调用关系图:



3.5.2.10 CMP_SetIRQSignal()

```
void CMP_SetIRQSignal (
    uint8_t CMP_CHN_x,
    uint8_t CMP_SIGNAL_x )
```

设置比较器中断信号

参数

<i>CMP_CHN_x</i>	比较器通道选择, CMP_CHN_0, CMP_CHN_1
<i>CMP_↔ SIGNAL_x</i>	中断信号选择, CMP_SIGNAL_NONE, CMP_SIGNAL_EDGE_P, CMP_SIGNAL_EDGE_N, CMP_SIGNAL_LEVEL_H, CMP_SIGNAL_LEVEL_L

在文件 [lks32mc09x_cmp.c](#) 第 195 行定义.

这是这个函数的调用关系图:



3.5.2.11 CMP_SetSignal()

```
void CMP_SetSignal (
    uint8_t CMP_CHN_x,
    uint8_t CMP_SELN_x_x,
    uint8_t CMP_SELP_x_x )
```

设置比较器信号来源

参数

<i>CMP_CHN_x</i>	比较器通道选择, CMP_CHN_0, CMP_CHN_1
<i>CMP_SELN_x↔x</i>	信号负端选择, 注意, 由于比较器的各个通道不完全一样, 所以需要指明使用的比较器
<i>CMP_SELP_x↔x</i>	信号正端选择, 注意, 由于比较器的各个通道不完全一样, 所以需要指明使用的比较器

在文件 [lks32mc09x_cmp.c](#) 第 72 行定义.

这是这个函数的调用关系图:



3.5.2.12 CMP_SetWindowSignal()

```
void CMP_SetWindowSignal (
    uint8_t CMP_CHN_x,
    uint8_t CMP_WIN_x,
    uint8_t CMP_WIN_LEVEL_x )
```

设置比较器开窗信号

参数

<i>CMP_CHN_x</i>	比较器通道选择, CMP_CHN_0, CMP_CHN_1
<i>CMP_WIN_x</i>	开窗信号选择, CMP_WIN_SIGNAL_NONE, CMP_WIN_SIGNAL_MCPWM0_0
<i>CMP_WIN_LEVEL_x↔LEVEL_x</i>	开窗电平选择, CMP_WIN_LEVEL_H, CMP_WIN_LEVEL_L

在文件 [lks32mc09x_cmp.c](#) 第 118 行定义.

这是这个函数的调用关系图:



3.5.2.13 CMP_StructInit()

```
void CMP_StructInit (
    CMP_InitTypeDef * CMP_InitStruct )
```

CMP结构体初始化

参数

<i>CMP_InitStruct</i>	指向CMP初始化结构体的指针

在文件 [lks32mc09x_cmp.c](#) 第 320 行定义.

3.6 lks32mc09x_cmp.h

[浏览该文件的文档.](#)

```
00001 #ifndef __LKS32MC09X_CMP_H__
00002 #define __LKS32MC09X_CMP_H__
00003 #include "lks32mc09x_reg.h"
00004
00008 typedef struct
00009 {
00010     u16 SELP;
00011     u16 SELN;
00012     u32 FIL_CLK_DIV16;
00013     u32 FIL_CLK_DIV2;
00014     u16 POL;
00015     u16 WPWM_POL;
00016     u16 IRQ_TRIG;
00017     FuncState EN;
00018     FuncState IE;
00019     FuncState RE;
00020     FuncState CHN3P_PWM_EN;
00021     FuncState CHN2P_PWM_EN;
00022     FuncState CHN1P_PWM_EN;
00023     FuncState CHNOP_PWM_EN;
00024 } CMP_SingleTypeDef;
00025 typedef struct
00026 {
00027     u8 HYS;
00028     u32 CLK_COM_DIV;
00029     CMP_SingleTypeDef CMP0_CFG;
00030     CMP_SingleTypeDef CMP1_CFG;
00031 } CMP_InitTypeDef;
00032
00033 #define CMP_CHN_0 0
00034 #define CMP_CHN_1 1
00035 // 比较器模拟部分配置
00036 // 比较器 1 正端信号选择
00037 #define CMP_SEL1_1_OPA2_IP 0
00038 #define CMP_SEL1_1_OPA3_IP 1
00039 #define CMP_SEL1_1_OPA2_OUTP 2
00040 #define CMP_SEL1_1_OPA3_OUTP 3
00041 #define CMP_SEL1_1_CMP1_IP0 4
00042 #define CMP_SEL1_1_CMP1_IP1 5
00043 #define CMP_SEL1_1_CMP1_IP2 6
00044 #define CMP_SEL1_1_CMP1_IP3 7
00045 #define CMP_SEL1_1_DAC1_OUT 9
00046 #define CMP_SEL1_1_OPA_OUT_CMP 10
00047 #define CMP_SEL1_1_CMP0_IP1 11
00048 #define CMP_SEL1_1_CMP0_IP2 12
00049 #define CMP_SEL1_1_CMP0_IP3 13
00050
00051 // 比较器 0 正端信号选择
00052 #define CMP_SEL0_0_OPA0_IP 0
00053 #define CMP_SEL0_0_OPA1_IP 1
00054 #define CMP_SEL0_0_OPA0_OUTP 2
00055 #define CMP_SEL0_0_OPA1_OUTP 3
00056 #define CMP_SEL0_0_CMP0_IP0 4
00057 #define CMP_SEL0_0_CMP0_IP1 5
00058 #define CMP_SEL0_0_CMP0_IP2 6
00059 #define CMP_SEL0_0_CMP0_IP3 7
```

```

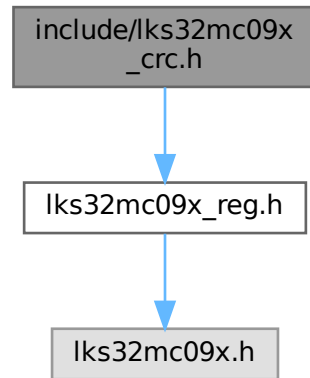
00060 #define CMP_SEL0_0_CMP0_IP4      8
00061 #define CMP_SEL0_0_DAC0_OUT      9
00062 #define CMP_SEL0_0_OUT_OPA_CMP 10
00063 #define CMP_SEL0_0_CMP1_IP1     11
00064 #define CMP_SEL0_0_CMP1_IP2     12
00065 #define CMP_SEL0_0_CMP1_IP3     13
00066
00067 // 比较器回差选择
00068 #define CMP_HYS_20mV 0 // 0:20mV
00069 #define CMP_HYS_0mV 1 // 0:20mV
00070
00071 // 比较器 1 信号负端选择
00072 #define CMP_SELN_1_IN_CMP1 0
00073 #define CMP_SELN_1_VBGP 1
00074 #define CMP_SELN_1_DAC1 2
00075 #define CMP_SELN_1_DAC0 3
00076 #define CMP_SELN_1_HS_MID1 4
00077
00078 // 比较器 0 信号负端选择
00079 #define CMP_SELN_0_IN_CMP0 0
00080 #define CMP_SELN_0_VBGP 1
00081 #define CMP_SELN_0_DAC1 2
00082 #define CMP_SELN_0_DAC0 3
00083 #define CMP_SELN_0_HS_MID0 4
00084
00085 // 比较器数字部分配置
00086 // 中断标志位
00087 #define CMP_IF_CMP0 BIT0
00088 #define CMP_IF_CMP1 BIT1
00089 // 中断使能
00090 #define CMP_IE_CMP0 BIT0
00091 #define CMP_IE_CMP1 BIT1
00092 // CMP事件DMA 请求使能
00093 #define CMP_RE_CMP0 BIT8
00094 #define CMP_RE_CMP1 BIT9
00095
00096 // 滤波宽度 1-2048可选, 不再提供宏定义
00097 // 模块时钟频率系数
00098 #define CMP_CLK_DIV_0 0
00099 #define CMP_CLK_DIV_1 1
00100 #define CMP_CLK_DIV_2 2
00101 #define CMP_CLK_DIV_3 3
00102 #define CMP_CLK_DIV_4 4
00103
00104 // 比较器中断信号选择
00105 #define CMP_SIGNAL_NONE 0 // 不产生中断信号
00106 #define CMP_SIGNAL_EDGE_P (BIT2 | BIT1) // 上升沿时产生中断信号
00107 #define CMP_SIGNAL_EDGE_N (BIT2 | BIT1 | BIT0) // 下降沿时产生中断信号
00108 #define CMP_SIGNAL_LEVEL_H (BIT1) // 高电平时产生中断信号, 此时除非高电平信号消失, 否则中断一直存在
00109 #define CMP_SIGNAL_LEVEL_L (BIT1 | BIT0) // 低电平时产生中断信号, 此时除非高电平信号消失, 否则中断一直存在
00110
00111 // 比较器开窗控制 (开窗控制同时影响MCPWM模块的FAIL信号和比较器本身的数字处理)
00112 #define CMP_WIN_SIGNAL_NONE 0 // 所有时刻比较器都运行
00113 #define CMP_WIN_SIGNAL_MCPWM0_0 (BIT0) // 在MCPWM0_0上管/关闭时不工作
00114 #define CMP_WIN_SIGNAL_MCPWM0_1 (BIT1) // 在MCPWM0_1上管/关闭时不工作
00115 #define CMP_WIN_SIGNAL_MCPWM0_2 (BIT2) // 在MCPWM0_2上管/关闭时不工作
00116 #define CMP_WIN_SIGNAL_MCPWM0_3 (BIT3) // 在MCPWM0_3上管/关闭时不工作
00117 #define CMP_WIN_LEVEL_H (1) // 上管打开时比较器不工作
00118 #define CMP_WIN_LEVEL_L (0) // 上管关闭时比较器不工作
00119
00120 // 模拟部分
00125 void CMP_Enable(uint8_t CMP_CHN_x);
00130 void CMP_Disable(uint8_t CMP_CHN_x);
00137 void CMP_SetSignal(uint8_t CMP_CHN_x, uint8_t CMP_SELN_x_x, uint8_t CMP_SEL0_x_x);
00138
00143 void CMP_SetHysteresis(uint8_t CMP_HYS_x);
00144
00145 // 数字部分
00152 void CMP_SetWindowSignal(uint8_t CMP_CHN_x, uint8_t CMP_WIN_x, uint8_t CMP_WIN_LEVEL_x);
00159 void CMP_SetFilterWidth(uint8_t CMP_CLK_DIV_x, uint16_t CMP0_FIL_x, uint16_t CMP1_FIL_x);
00160
00166 void CMP_SetIRQSignal(uint8_t CMP_CHN_x, uint8_t CMP_SIGNAL_x);
00171 void CMP_ClearIRQFlag(uint8_t CMP_CHN_x);
00172
00178 uint8_t CMP_GetIRQFlag(uint8_t CMP_CHN_x);
00179
00184 uint8_t CMP_GetFilterValue(uint8_t CMP_CHN_x);
00185
00190 uint8_t CMP_GetValue(uint8_t CMP_CHN_x);
00191
00192 void CMP_StructInit(CMP_InitTypeDef *CMP_InitStruct); /* CMP结构体初始化*/
00193 void CMP_Init(CMP_InitTypeDef *CMP_InitStruct); /* CMP初始化*/
00194 #endif

```

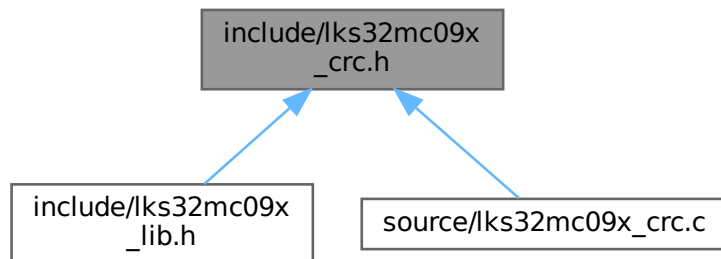
3.7 include/lks32mc09x_crc.h 文件参考

```
#include "lks32mc09x_reg.h"
```

lks32mc09x_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:



结构体

- struct [CRC_InitTypeDef](#)
CRC初始化结构体

宏定义

- #define [CRC_MODEL_CRC_8](#) 0
CRC模型定义

- #define CRC.MODEL_CRC_8.ITU 1
8 07 00 55 false false
- #define CRC.MODEL_CRC_8.ROHC 2
8 07 FF 00 true true
- #define CRC.MODEL_CRC_8.MAXIMx 3
8 31 00 00 true true
- #define CRC.MODEL_CRC_16.IBMx1 4
16 8005 0000 0000 true true
- #define CRC.MODEL_CRC_16.MAXIM 5
16 8005 0000 FFFF true true
- #define CRC.MODEL_CRC_16.USBx1 6
16 8005 FFFF FFFF true true
- #define CRC.MODEL_CRC_16.MOBU 7
16 8005 FFFF 0000 true true
- #define CRC.MODEL_CRC_16.CCITT 8
16 1021 0000 0000 true true
- #define CRC.MODEL_CRC_16.CCITT_FALSE 9
16 1021 FFFF 0000 false false
- #define CRC.MODEL_CRC_16.X25 10
16 1021 FFFF FFFF true true
- #define CRC.MODEL_CRC_16.XMODEM 11
16 1021 0000 0000 false false
- #define CRC.MODEL_CRC_16.DNP 12
16 3D65 0000 FFFF true true
- #define CRC.MODEL_CRC_32 13
32 04C11DB7 FFFFFFFF FFFFFFFF true true
- #define CRC.MODEL_CRC_32.MPEG_2 14
32 04C11DB7 FFFFFFFF 00000000 false false
- #define CRC.WIDTH_32 0
CRC宽度定义
- #define CRC.WIDTH_16 1
16bits
- #define CRC.WIDTH_8 2
8bits
- #define CRC.WIDTH_7 3
7bits

函数

- void CRC_Init (CRC_TypeDef *CRCx, CRC_InitTypeDef *CRC_InitStruct)
初始化CRC模块
- void CRC_StructInit (CRC_InitTypeDef *CRC_InitStruct)
初始化CRC结构体为默认值
- uint32_t CRC_Calc (CRC_TypeDef *CRCx, uint8_t *data, uint32_t len)
计算CRC值
- uint32_t CRC_CalcCommon (CRC_TypeDef *CRCx, uint8_t *data, uint32_t len, uint16_t CRC_MODEL_x)
使用常见模型计算CRC值
- void CRC_Enable (CRC_TypeDef *CRCx)
使能CRC模块
- void CRC_Disable (CRC_TypeDef *CRCx)
禁用CRC模块

3.7.1 宏定义说明

3.7.1.1 CRC_MODEL_CRC_16_CCITT

```
#define CRC_MODEL_CRC_16_CCITT 8
```

```
16 1021 0000 0000 true true
```

在文件 [lks32mc09x_crc.h](#) 第 31 行定义.

3.7.1.2 CRC_MODEL_CRC_16_CCITT_FALSE

```
#define CRC_MODEL_CRC_16_CCITT_FALSE 9
```

```
16 1021 FFFF 0000 false false
```

在文件 [lks32mc09x_crc.h](#) 第 32 行定义.

3.7.1.3 CRC_MODEL_CRC_16_DNP

```
#define CRC_MODEL_CRC_16_DNP 12
```

```
16 3D65 0000 FFFF true true
```

在文件 [lks32mc09x_crc.h](#) 第 35 行定义.

3.7.1.4 CRC_MODEL_CRC_16_IBMx1

```
#define CRC_MODEL_CRC_16_IBMx1 4
```

```
16 8005 0000 0000 true true
```

在文件 [lks32mc09x_crc.h](#) 第 27 行定义.

3.7.1.5 CRC_MODEL_CRC_16_MAXIM

```
#define CRC_MODEL_CRC_16_MAXIM 5
```

```
16 8005 0000 FFFF true true
```

在文件 [lks32mc09x_crc.h](#) 第 28 行定义.

3.7.1.6 CRC_MODEL_CRC_16_MODBU

```
#define CRC_MODEL_CRC_16_MODBU 7
```

```
16 8005 FFFF 0000 true true
```

在文件 [lks32mc09x_crc.h](#) 第 30 行定义.

3.7.1.7 CRC_MODEL_CRC_16_USBx1

```
#define CRC_MODEL_CRC_16_USBx1 6
```

```
16 8005 FFFF FFFF true true
```

在文件 [lks32mc09x_crc.h](#) 第 29 行定义.

3.7.1.8 CRC_MODEL_CRC_16_X25

```
#define CRC_MODEL_CRC_16_X25 10
```

```
16 1021 FFFF FFFF true true
```

在文件 [lks32mc09x_crc.h](#) 第 33 行定义.

3.7.1.9 CRC_MODEL_CRC_16_XMODEM

```
#define CRC_MODEL_CRC_16_XMODEM 11
```

```
16 1021 0000 0000 false false
```

在文件 [lks32mc09x_crc.h](#) 第 34 行定义.

3.7.1.10 CRC_MODEL_CRC_32

```
#define CRC_MODEL_CRC_32 13
```

```
32 04C11DB7 FFFFFFFF FFFFFFFF true true
```

在文件 [lks32mc09x_crc.h](#) 第 36 行定义.

3.7.1.11 CRC_MODEL_CRC_32_MPEG_2

```
#define CRC_MODEL_CRC_32_MPEG_2 14
```

```
32 04C11DB7 FFFFFFFF 00000000 false false
```

在文件 [lks32mc09x_crc.h](#) 第 37 行定义.

3.7.1.12 CRC_MODEL_CRC_8

```
#define CRC_MODEL_CRC_8 0
```

CRC模型定义

```
8 07 00 00 false false
```

在文件 [lks32mc09x_crc.h](#) 第 23 行定义.

3.7.1.13 CRC_MODEL_CRC_8_ITU

```
#define CRC_MODEL_CRC_8_ITU 1
```

8 07 00 55 false false

在文件 [lks32mc09x_crc.h](#) 第 24 行定义.

3.7.1.14 CRC_MODEL_CRC_8_MAXIMx

```
#define CRC_MODEL_CRC_8_MAXIMx 3
```

8 31 00 00 true true

在文件 [lks32mc09x_crc.h](#) 第 26 行定义.

3.7.1.15 CRC_MODEL_CRC_8_ROHC

```
#define CRC_MODEL_CRC_8_ROHC 2
```

8 07 FF 00 true true

在文件 [lks32mc09x_crc.h](#) 第 25 行定义.

3.7.1.16 CRC_WIDTH_16

```
#define CRC_WIDTH_16 1
```

16bits

在文件 [lks32mc09x_crc.h](#) 第 43 行定义.

3.7.1.17 CRC_WIDTH_32

```
#define CRC_WIDTH_32 0
```

CRC宽度定义

32bits

在文件 [lks32mc09x_crc.h](#) 第 42 行定义.

3.7.1.18 CRC_WIDTH_7

```
#define CRC_WIDTH_7 3
```

7bits

在文件 [lks32mc09x_crc.h](#) 第 45 行定义.

3.7.1.19 CRC_WIDTH_8

```
#define CRC_WIDTH_8 2

8bits
```

在文件 [lks32mc09x_crc.h](#) 第 44 行定义.

3.7.2 函数说明

3.7.2.1 CRC_Calc()

```
uint32_t CRC_Calc (
    CRC_TypeDef * CRCx,
    uint8_t * data,
    uint32_t len )
```

计算CRC值

参数

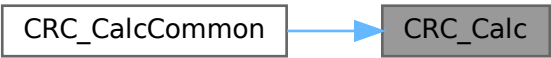
<i>CRCx</i>	CRC模块指针
<i>data</i>	数据指针
<i>len</i>	数据长度

返回

计算得到的CRC值

在文件 [lks32mc09x_crc.c](#) 第 79 行定义.

这是这个函数的调用关系图:



3.7.2.2 CRC_CalcCommon()

```
uint32_t CRC_CalcCommon (
    CRC_TypeDef * CRCx,
    uint8_t * data,
    uint32_t len,
    uint16_t CRC_MODEL_x )
```

使用常见模型计算CRC值

参数

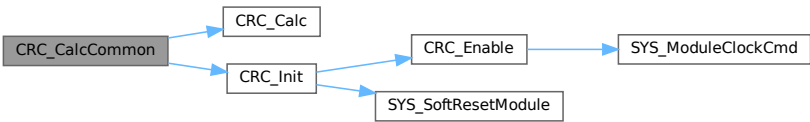
<i>CRCx</i>	CRC模块指针
<i>data</i>	数据指针
<i>len</i>	数据长度
<i>CRC_↔ MODEL_x</i>	CRC模型

返回

计算得到的CRC值

在文件 [lks32mc09x_crc.c](#) 第 106 行定义.

函数调用图:



3.7.2.3 CRC_Disable()

```
void CRC_Disable (
    CRC_TypeDef * CRCx )
```

禁用CRC模块

参数

<i>CRCx</i>	CRC模块指针

在文件 [lks32mc09x_crc.c](#) 第 304 行定义.

函数调用图:



3.7.2.4 CRC_Enable()

```
void CRC_Enable (
    CRC_TypeDef * CRCx )
```

使能CRC模块

参数

CRCx	CRC模块指针

在文件 [lks32mc09x_crc.c](#) 第 288 行定义.

函数调用图:



这是这个函数的调用关系图:



3.7.2.5 CRC_Init()

```
void CRC_Init (
    CRC_TypeDef * CRCx,
    CRC_InitTypeDef * CRC_InitStruct )
```

初始化CRC模块

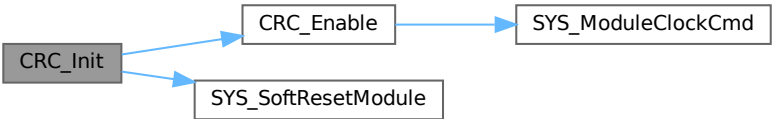
参数

CRCx	CRC模块指针
CRC_InitStruct	CRC初始化结构体指针

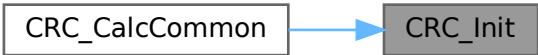
初始化CRC模块

在文件 `lks32mc09x_crc.c` 第 14 行定义.

函数调用图:



这是这个函数的调用关系图:



3.7.2.6 CRC_StructInit()

```
void CRC_StructInit (
    CRC_InitTypeDef * CRC_InitStruct )
```

初始化CRC结构体为默认值

参数

<code>CRC_InitStruct</code>	CRC初始化结构体指针

在文件 `lks32mc09x_crc.c` 第 61 行定义.

3.8 lks32mc09x_crc.h

[浏览该文件的文档.](#)

```
00001 #ifndef __LKS32MC09X_CRC_H__
00002 #define __LKS32MC09X_CRC_H__
00003
00004 #include "lks32mc09x_reg.h"
00005
00009 typedef struct
00010 {
00011     uint32_t POLY;
00012     uint32_t INIT;
00013     uint32_t XOROUT;
00014     uint8_t  REFIN;
```

```

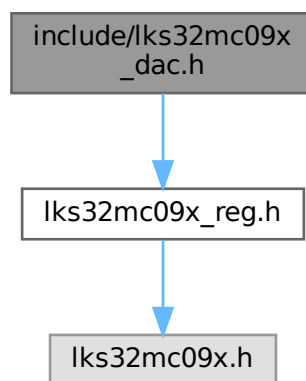
00015     uint8_t REFOUT;
00016     uint8_t WIDTH;
00017     uint8_t null;
00018 } CRC_InitTypeDef;
00019
00023 #define CRC_MODEL_CRC_8          0
00024 #define CRC_MODEL_CRC_8_ITU      1
00025 #define CRC_MODEL_CRC_8_ROHC     2
00026 #define CRC_MODEL_CRC_8_MAXIMx   3
00027 #define CRC_MODEL_CRC_16_IBMx1   4
00028 #define CRC_MODEL_CRC_16_MAXIM   5
00029 #define CRC_MODEL_CRC_16_USBx1   6
00030 #define CRC_MODEL_CRC_16_MODBU   7
00031 #define CRC_MODEL_CRC_16_CCITT   8
00032 #define CRC_MODEL_CRC_16_CCITT_FALSE 9
00033 #define CRC_MODEL_CRC_16_X25     10
00034 #define CRC_MODEL_CRC_16_XMODEM  11
00035 #define CRC_MODEL_CRC_16_DNP     12
00036 #define CRC_MODEL_CRC_32         13
00037 #define CRC_MODEL_CRC_32_MPEG_2  14
00038
00042 #define CRC_WIDTH_32 0
00043 #define CRC_WIDTH_16 1
00044 #define CRC_WIDTH_8  2
00045 #define CRC_WIDTH_7  3
00046
00052 void CRC_Init(CRC_TypeDef *CRCx, CRC_InitTypeDef *CRC_InitStruct);
00053
00058 void CRC_StructInit(CRC_InitTypeDef *CRC_InitStruct);
00059
00067 uint32_t CRC_Calc(CRC_TypeDef *CRCx, uint8_t *data, uint32_t len);
00068
00077 uint32_t CRC_CalcCommon(CRC_TypeDef *CRCx, uint8_t *data, uint32_t len, uint16_t CRC_MODEL_x);
00078
00083 void CRC_Enable(CRC_TypeDef *CRCx);
00084
00089 void CRC_Disable(CRC_TypeDef *CRCx);
00090
00091 #endif

```

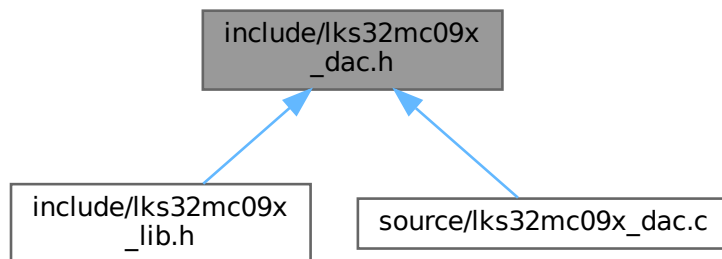
3.9 include/lks32mc09x_dac.h 文件参考

#include "lks32mc09x_reg.h"

lks32mc09x_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：



结构体

- struct [DAC_InitTypeDef](#)

宏定义

- #define [DAC_CHANNEL_0](#) 0
DAC模块设置
- #define [DAC_CHANNEL_1](#) 1
- #define [DAC_RANGE_0_4V76](#) 0
DAC量程
- #define [DAC_RANGE_0_1V2](#) 1
DAC0 1.2V量程
- #define [DAC_RANGE_1_3V](#) 0
DAC1 3V量程
- #define [DAC_RANGE_1_1V2](#) 1
DAC1 1.2V量程
- #define [DAC_STEP_TRG_NONE](#) (0)
DAC0步进
- #define [DAC_STEP_TRG_TIMER0](#) (1 << 0)
TIMER0 过零事件触发 DAC0 步进
- #define [DAC_STEP_TRG_TIMER1](#) (1 << 1)
TIMER1 过零事件触发 DAC0 步进
- #define [DAC_STEP_TRG_TIMER2](#) (1 << 2)
TIMER2 过零事件触发 DAC0 步进
- #define [DAC_STEP_TRG_TIMER3](#) (1 << 3)
TIMER3 过零事件触发 DAC0 步进
- #define [DAC_STEP__4](#) 0x4
DAC0步进
- #define [DAC_STEP__3](#) 0x5
DAC0 步进值 -3
- #define [DAC_STEP__2](#) 0x6
DAC0 步进值 -2

- `#define DAC_STEP_1 0x7`
DAC0 步进值 -1
- `#define DAC_STEP_0 0x0`
DAC0 步进值 0
- `#define DAC_STEP_1 0x1`
DAC0 步进值 1
- `#define DAC_STEP_2 0x2`
DAC0 步进值 2
- `#define DAC_STEP_3 0x3`
DAC0 步进值 3

函数

- `void DAC_Init (uint8_t DAC_Channel, DAC_InitTypeDef *DAC_InitStruct)`
DAC初始化
- `void DAC_StructInit (DAC_InitTypeDef *DAC_InitStruct)`
DAC初始化结构体初始化
- `void DAC_OutputValue (uint8_t DAC_Channel, uint32_t DACValue)`
DAC输出数值设置（数字量）
- `void DAC_SetRange (uint8_t DAC_Channel, uint32_t DAC_RANGE_x)`
设置DAC量程
- `void DAC_OutputVoltage (uint8_t DAC_Channel, uint16_t DACVoltage)`
DAC输出模拟量，自动配置量程&同时自动使能对应的模块
- `void DAC_Cmd (uint8_t DAC_Channel, uint8_t state)`
DAC通道时钟使能
- `void DAC_Enable (uint8_t DAC_Channel)`
DAC使能
- `void DAC_Disable (uint8_t DAC_Channel)`
DAC失能
- `void DAC_Reset (uint8_t DAC_Channel)`
DAC复位
- `void DAC_Output (uint8_t DAC_Channel, uint8_t state)`
DAC输出至IO口使能配置
- `uint32_t DAC_Value (uint8_t DAC_Channel)`
获取当前DAC输出值
- `uint16_t DAC_Voltage (uint8_t DAC_Channel)`
获取当前DAC输出电压
- `uint8_t DAC_Range (uint8_t DAC_Channel)`
获取当前DAC量程
- `void DAC_Step (uint8_t DAC_Channel, uint8_t DAC_STEP_x, uint8_t DAC_STEP_TRG_x)`
DAC0步进设置

3.9.1 宏定义说明

3.9.1.1 DAC_CHANNEL_0

```
#define DAC_CHANNEL_0 0
```

DAC模块设置

在文件 [lks32mc09x-dac.h](#) 第 8 行定义.

3.9.1.2 DAC_CHANNEL_1

```
#define DAC_CHANNEL_1 1
```

在文件 [lks32mc09x_dac.h](#) 第 9 行定义.

3.9.1.3 DAC_RANGE_0_1V2

```
#define DAC_RANGE_0_1V2 1
```

DAC0 1.2V量程

在文件 [lks32mc09x_dac.h](#) 第 17 行定义.

3.9.1.4 DAC_RANGE_0_4V76

```
#define DAC_RANGE_0_4V76 0
```

DAC量程

参见

SYS_AFE_REG1_DAC1_GAIN_BIT SYS_AFE_REG1_DAC0_GAIN_BIT DAC0 4.76V量程

在文件 [lks32mc09x_dac.h](#) 第 16 行定义.

3.9.1.5 DAC_RANGE_1_1V2

```
#define DAC_RANGE_1_1V2 1
```

DAC1 1.2V量程

在文件 [lks32mc09x_dac.h](#) 第 19 行定义.

3.9.1.6 DAC_RANGE_1_3V

```
#define DAC_RANGE_1_3V 0
```

DAC1 3V量程

在文件 [lks32mc09x_dac.h](#) 第 18 行定义.

3.9.1.7 DAC_STEP_1

```
#define DAC_STEP_1 0x1
```

DAC0 步进值 1

在文件 [lks32mc09x_dac.h](#) 第 36 行定义.

3.9.1.8 DAC_STEP_2

```
#define DAC_STEP_2 0x2
```

DAC0 步进值 2

在文件 [lks32mc09x_dac.h](#) 第 37 行定义.

3.9.1.9 DAC_STEP_3

```
#define DAC_STEP_3 0x3
```

DAC0 步进值 3

在文件 [lks32mc09x_dac.h](#) 第 38 行定义.

3.9.1.10 DAC_STEP_0

```
#define DAC_STEP_0 0x0
```

DAC0 步进值 0

在文件 [lks32mc09x_dac.h](#) 第 35 行定义.

3.9.1.11 DAC_STEP_1

```
#define DAC_STEP_1 0x7
```

DAC0 步进值 -1

在文件 [lks32mc09x_dac.h](#) 第 34 行定义.

3.9.1.12 DAC_STEP_2

```
#define DAC_STEP_2 0x6
```

DAC0 步进值 -2

在文件 [lks32mc09x_dac.h](#) 第 33 行定义.

3.9.1.13 DAC_STEP_3

```
#define DAC_STEP_3 0x5
```

DAC0 步进值 -3

在文件 [lks32mc09x_dac.h](#) 第 32 行定义.

3.9.1.14 DAC_STEP_4

```
#define DAC_STEP_4 0x4
```

DAC0步进

DAC0 步进值 -4

在文件 [lks32mc09x.dac.h](#) 第 31 行定义.

3.9.1.15 DAC_STEP_TRG_NONE

```
#define DAC_STEP_TRG_NONE (0)
```

DAC0步进

不触发DAC步进

在文件 [lks32mc09x.dac.h](#) 第 23 行定义.

3.9.1.16 DAC_STEP_TRG_TIMER0

```
#define DAC_STEP_TRG_TIMER0 (1 << 0)
```

TIMER0 过零事件触发 DAC0 步进

在文件 [lks32mc09x.dac.h](#) 第 24 行定义.

3.9.1.17 DAC_STEP_TRG_TIMER1

```
#define DAC_STEP_TRG_TIMER1 (1 << 1)
```

TIMER1 过零事件触发 DAC0 步进

在文件 [lks32mc09x.dac.h](#) 第 25 行定义.

3.9.1.18 DAC_STEP_TRG_TIMER2

```
#define DAC_STEP_TRG_TIMER2 (1 << 2)
```

TIMER2 过零事件触发 DAC0 步进

在文件 [lks32mc09x.dac.h](#) 第 26 行定义.

3.9.1.19 DAC_STEP_TRG_TIMER3

```
#define DAC_STEP_TRG_TIMER3 (1 << 3)
```

TIMER3 过零事件触发 DAC0 步进

在文件 [lks32mc09x.dac.h](#) 第 27 行定义.

3.9.2 函数说明

3.9.2.1 DAC_Cmd()

```
void DAC_Cmd (
    uint8_t DAC_Channel,
    uint8_t state )
```

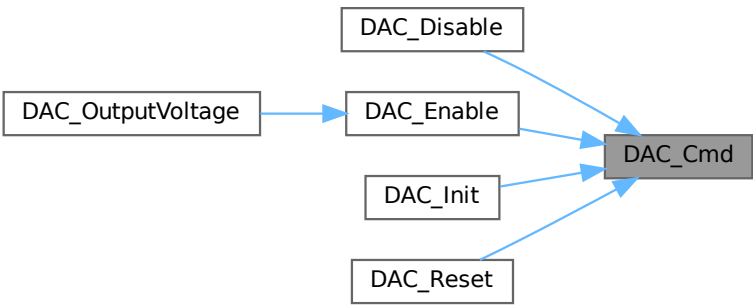
DAC通道时钟使能

参数

<i>DAC_Channel</i>	DAC通道选择
<i>state</i>	使能状态，1模块使能 0 模块关闭

在文件 [lks32mc09x_dac.c](#) 第 151 行定义.

这是这个函数的调用关系图:



3.9.2.2 DAC_Disable()

```
void DAC_Disable (
    uint8_t DAC_Channel )
```

DAC失能

参数

<i>DAC_Channel</i>	DAC通道选择

在文件 [lks32mc09x_dac.c](#) 第 199 行定义.

函数调用图:



3.9.2.3 DAC_Enable()

```
void DAC_Enable (
    uint8_t DAC_Channel )
```

DAC使能

参数

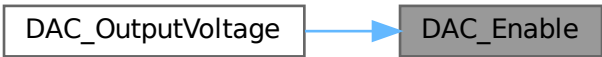
DAC_Channel	DAC通道选择

在文件 [lks32mc09x-dac.c](#) 第 191 行定义.

函数调用图:



这是这个函数的调用关系图:



3.9.2.4 DAC_Init()

```
void DAC_Init (
    uint8_t DAC_Channel,
    DAC_InitTypeDef * DAC_InitStruct )
```

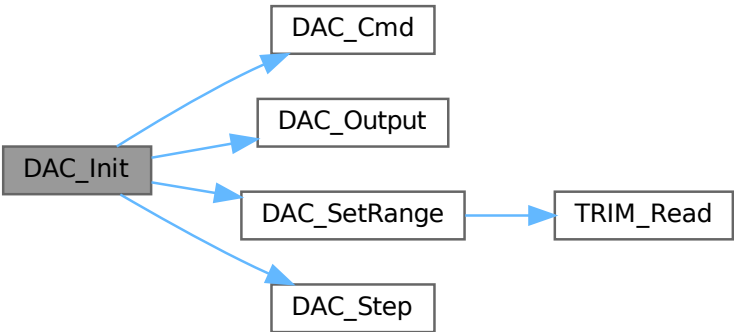
DAC初始化

参数

DAC_Channel	DAC通道选择
DAC_InitStruct	DAC初始化结构体

在文件 [lks32mc09x-dac.c](#) 第 9 行定义.

函数调用图:



3.9.2.5 DAC_Output()

```
void DAC_Output (
    uint8_t DAC_Channel,
    uint8_t state )
```

DAC输出至IO口使能配置

参数

DAC_Channel	DAC通道选择
state	1Dac电压p0.0口 0停止输出到P0.0

在文件 [lks32mc09x-dac.c](#) 第 218 行定义.

这是这个函数的调用关系图:



3.9.2.6 DAC_OutputValue()

```
void DAC_OutputValue (
```

```
uint8_t DAC_Channel,
uint32_t DACValue )
```

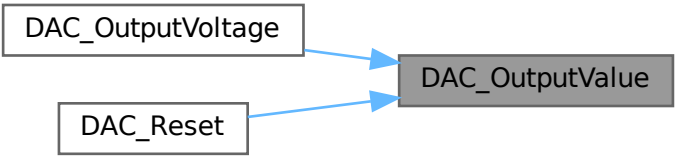
DAC输出数值设置（数字量）

参数

<i>DAC_Channel</i>	DAC通道选择
<i>DACValue</i>	DAC输出数字量

在文件 [lks32mc09x_dac.c](#) 第 33 行定义.

这是这个函数的调用关系图:



3.9.2.7 DAC_OutputVoltage()

```
void DAC_OutputVoltage (
    uint8_t DAC_Channel,
    uint16_t DACVoltage )
```

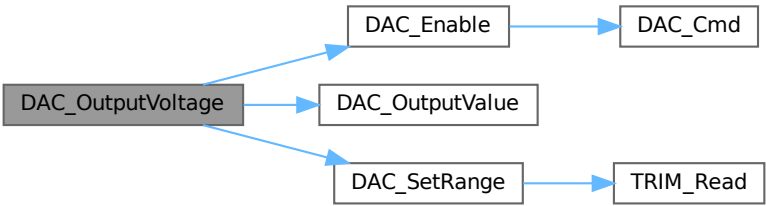
DAC输出模拟量，自动配置量程&同时自动使能对应的模块

参数

<i>DAC_Channel</i>	DAC通道选择
<i>DACVoltage</i>	DAC输出模拟量 单位mV

在文件 [lks32mc09x_dac.c](#) 第 106 行定义.

函数调用图:



3.9.2.8 DAC.Range()

```
uint8_t DAC.Range (
    uint8_t DAC.Channel )
```

获取当前DAC量程

参数

DAC.Channel	DAC通道选择

返回

DAC量程 dac0 0:4.85V 1:1.2V dac1 0:3V 1:1.2V

在文件 [lks32mc09x-dac.c](#) 第 310 行定义.

这是这个函数的调用关系图:



3.9.2.9 DAC.Reset()

```
void DAC.Reset (
    uint8_t DAC.Channel )
```

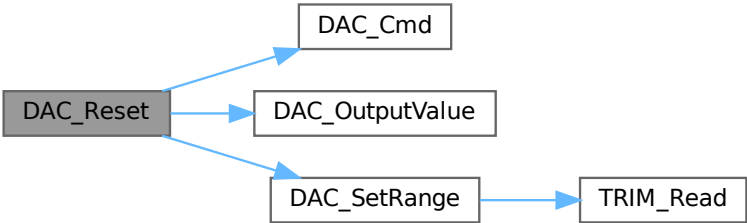
DAC复位

参数

<i>DAC_Channel</i>	DAC通道选择

在文件 [lks32mc09x_dac.c](#) 第 207 行定义.

函数调用图:



3.9.2.10 DAC.SetRange()

```
void DAC_SetRange (
    uint8_t DAC_Channel,
    uint32_t DAC_RANGE_x )
```

设置DAC量程

参数

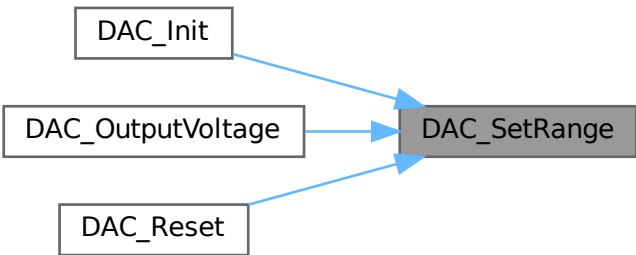
<i>DAC_Channel</i>	DAC通道选择
<i>DAC_RANGE_x</i>	DAC输出量程

在文件 [lks32mc09x_dac.c](#) 第 61 行定义.

函数调用图:



这是这个函数的调用关系图:



3.9.2.11 DAC.Step()

```
void DAC_Step (
    uint8_t DAC_Channel,
    uint8_t DAC_STEP_x,
    uint8_t DAC_STEP_TRG_x )
```

DAC0步进设置

参数

DAC_Channel	DAC通道选择
DAC_STEP_x	DAC0步进值
DAC_STEP↔TRG_x	DAC0步进触发源

在文件 [lks32mc09x-dac.c](#) 第 346 行定义.

这是这个函数的调用关系图:



3.9.2.12 DAC.StructInit()

```
void DAC_StructInit (
    DAC_InitTypeDef * DAC_InitStruct )
```


DAC初始化结构体初始化

参数

<i>DAC_InitStruct</i>	DAC初始化结构体

在文件 [lks32mc09x_dac.c](#) 第 20 行定义.

3.9.2.13 DAC_Value()

```
uint32_t DAC_Value (
    uint8_t DAC_Channel )
```

获取当前DAC输出值

参数

<i>DAC_Channel</i>	DAC通道选择

返回

DAC输出值

在文件 [lks32mc09x_dac.c](#) 第 254 行定义.

这是这个函数的调用关系图:



3.9.2.14 DAC_Voltage()

```
uint16_t DAC_Voltage (
    uint8_t DAC_Channel )
```

获取当前DAC输出电压

参数

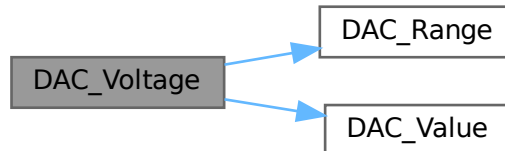
<i>DAC_Channel</i>	DAC通道选择

返回

DAC输出电压 单位mV

在文件 `lks32mc09x_dac.c` 第 270 行定义.

函数调用图:



3.10 lks32mc09x_dac.h

[浏览该文件的文档.](#)

```

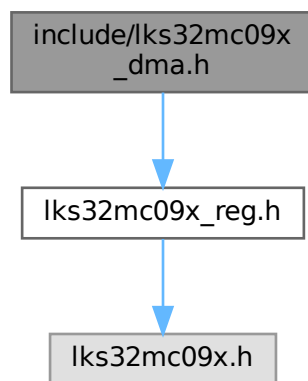
00001 #ifndef __LKS32MC09X_DAC_H__
00002 #define __LKS32MC09X_DAC_H__
00003
00004 #include "lks32mc09x_reg.h"
00008 #define DAC_CHANNEL_0 0 // 12位dac, 量程4.85V和1.2V可选
00009 #define DAC_CHANNEL_1 1 // 8位dac, 量程3V和1.2V可选
00016 #define DAC_RANGE_0_4V76 0
00017 #define DAC_RANGE_0_1V2 1
00018 #define DAC_RANGE_1_3V 0
00019 #define DAC_RANGE_1_1V2 1
00023 #define DAC_STEP_TRG_NONE (0)
00024 #define DAC_STEP_TRG_TIMER0 (1 << 0)
00025 #define DAC_STEP_TRG_TIMER1 (1 << 1)
00026 #define DAC_STEP_TRG_TIMER2 (1 << 2)
00027 #define DAC_STEP_TRG_TIMER3 (1 << 3)
00031 #define DAC_STEP_4 0x4
00032 #define DAC_STEP_3 0x5
00033 #define DAC_STEP_2 0x6
00034 #define DAC_STEP_1 0x7
00035 #define DAC_STEP_0 0x0
00036 #define DAC_STEP_1 0x1
00037 #define DAC_STEP_2 0x2
00038 #define DAC_STEP_3 0x3
00039
00040 typedef struct
00041 {
00042     uint8_t DAC_RANGE;
00043     uint8_t DACOUT_EN;
00044     int8_t DAC_STEP;
00045     uint8_t DAC_STEP_TRG_x;
00046 } DAC_InitTypeDef;
00047
00053 void DAC_Init(uint8_t DAC_Channel, DAC_InitTypeDef *DAC_InitStruct);
00054
00059 void DAC_StructInit(DAC_InitTypeDef *DAC_InitStruct);
00065 void DAC_OutputValue(uint8_t DAC_Channel, uint32_t DACValue);
00066
00072 void DAC_SetRange(uint8_t DAC_Channel, uint32_t DAC_RANGE_x);
00073
00079 void DAC_OutputVoltage(uint8_t DAC_Channel, uint16_t DACVoltage);
00080
00086 void DAC_Cmd(uint8_t DAC_Channel, uint8_t state);
00087
00092 void DAC_Enable(uint8_t DAC_Channel);
00097 void DAC_Disable(uint8_t DAC_Channel);
00102 void DAC_Reset(uint8_t DAC_Channel);
00108 void DAC_Output(uint8_t DAC_Channel, uint8_t state);
  
```

```
00109
00115 uint32_t DAC_Value(uint8_t DAC_Channel);
00121 uint16_t DAC_Voltage(uint8_t DAC_Channel);
00127 uint8_t DAC_Range(uint8_t DAC_Channel);
00134 void DAC_Step(uint8_t DAC_Channel, uint8_t DAC_STEP_x, uint8_t DAC_STEP_TRG_x);
00135 #endif
```

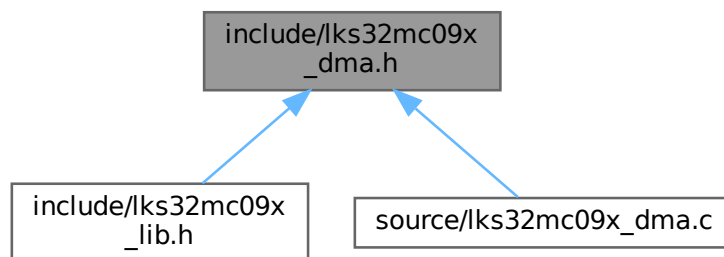
3.11 include/lks32mc09x_dma.h 文件参考

```
#include "lks32mc09x_reg.h"
```

lks32mc09x_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:



结构体

- struct [DMA_InitTypeDef](#)

*DMA*初始化结构体句柄，初始化时请定义该句柄，并用其它地址来传参

宏定义

- #define DMA.REN_ADC0 BIT0
DMA搬运使能源定义
- #define DMA.REN_ADC1 BIT1
- #define DMA.REN_DSP BIT3
- #define DMA.REN_TIMER0 BIT4
- #define DMA.REN_TIMER1 BIT5
- #define DMA.REN_TIMER2 BIT6
- #define DMA.REN_TIMER3 BIT7
- #define DMA.REN_SPI_RX BIT10
- #define DMA.REN_SPI_TX BIT11
- #define DMA.REN_MCPWM_RE0 BIT12
- #define DMA.REN_MCPWM_RE1 BIT13
- #define DMA.REN_I2C0_RX BIT14
- #define DMA.REN_I2C0_TX BIT15
- #define DMA.REN_UART0_RX BIT16
- #define DMA.REN_UART0_TX BIT17
- #define DMA.REN_UART1_RX BIT18
- #define DMA.REN_UART1_TX BIT19
- #define DMA.REN_UART2_RX BIT20
- #define DMA.REN_UART2_TX BIT21
- #define DMA.REN_HALL0 BIT24
- #define DMA.REN_SIF BIT25
- #define DMA.REN_CMP BIT26
- #define DMA.REN_GPIO BIT29
- #define DMA.REN_SW BIT31
- #define DMA.TCIE BIT0
传输完成中断使能，高有效
- #define DMA.TEIE BIT1
错误中断使能，高有效
- #define DMA.PERI_2_MEMORY 0
外设至内存
- #define DMA.MEMORY_2_PERI 1
内存至外设
- #define DMA.BYTE_TRANS 0
访问位宽， 0:byte
- #define DMA.HALFWORD_TRANS 1
访问位宽， 1:half-word
- #define DMA.WORD_TRANS 2
访问位宽， 2:word
- #define DMA.REQ_SW DMA.REQ_SW
SW 触发
- #define DMA.REQ_GPIO DMA.REQ_GPIO
GPIO 触发
- #define DMA.REQ_CMP DMA.REQ_CMP
CMP 触发
- #define DMA.REQ_SIF DMA.REQ_SIF
SIF 触发
- #define DMA.REQ_HALL DMA.REQ_HALL
HALL0 触发
- #define DMA.REQ_UART1_TX DMA.REQ_UART1_TX

- `UART1_TX` 触发
- `#define DMA_REQ_UART1_RX DMA_REQ_UART1_RX`
`UART1_RX` 触发
- `#define DMA_REQ_UART0_TX DMA_REQ_UART0_TX`
`UART0_TX` 触发
- `#define DMA_REQ_UART0_RX DMA_REQ_UART0_RX`
`UART0_RX` 触发
- `#define DMA_REQ_I2C0_TX DMA_REQ_I2C0_TX`
`I2C0_TX` 触发
- `#define DMA_REQ_I2C0_RX DMA_REQ_I2C0_RX`
`I2C0_RX` 触发
- `#define DMA_REQ_MCPWM1 DMA_REQ_MCPWM1`
`MCPWM1` 触发
- `#define DMA_REQ_MCPWM0 DMA_REQ_MCPWM0`
`MCPWM0` 触发
- `#define DMA_REQ_SPI_TX DMA_REQ_SPI_TX`
`SPI_TX` 触发
- `#define DMA_REQ_SPI_RX DMA_REQ_SPI_RX`
`SPI_RX` 触发
- `#define DMA_REQ_TIMER3 DMA_REQ_TIMER3`
`TIMER3` 触发
- `#define DMA_REQ_TIMER2 DMA_REQ_TIMER2`
`TIMER2` 触发
- `#define DMA_REQ_TIMER1 DMA_REQ_TIMER1`
`TIMER1` 触发
- `#define DMA_REQ_TIMER0 DMA_REQ_TIMER0`
`TIMER0` 触发
- `#define DMA_REQ_DSP DMA_REQ_DSP`
`DSP` 触发
- `#define DMA_REQ_ADC1 DMA_REQ_ADC1`
`ADC1` 触发
- `#define DMA_REQ_ADC0 DMA_REQ_ADC0`
`ADC0` 触发
- `#define DMA_IF_CH0 BIT0`
`DMA`中断标志位`MASK`
- `#define DMA_IF_CH1 BIT1`
- `#define DMA_IF_CH2 BIT2`
- `#define DMA_IF_CH3 BIT3`
- `#define DMA_IE_CH0 BIT0`
`DMA`中断使能位`MASK`
- `#define DMA_IE_CH1 BIT1`
- `#define DMA_IE_CH2 BIT2`
- `#define DMA_IE_CH3 BIT3`

函数

- void **DMA_Init** (**DMA_TypeDef** *DMAx, **DMA_InitTypeDef** *DMAInitStruct)
初始化DMA模块
- void **DMA_StructInit** (**DMA_InitTypeDef** *DMAInitStruct)
将DMA初始化结构体重置为默认值
- uint32_t **DMA_GetIRQFlag** (uint32_t DMA_IF_x)
获取DMA中断标志位状态
- void **DMA_ClearIRQFlag** (uint32_t DMA_IF_x)
清除DMA中断标志位
- uint32_t **DMA_GetFlag** (uint32_t DMA_IF_x)
获取DMA标志位状态
- void **DMA_Enable** (void)
使能DMA模块
- void **DMA_Disable** (void)
禁用DMA模块
- void **DMA_Reset** (void)
复位DMA模块
- void **DMA_SetAddr** (**DMA_TypeDef** *DMA_CHx, uint32_t addr1, uint32_t addr2)
设置DMA通道源地址和目的地址
- void **DMA_SetLength** (**DMA_TypeDef** *DMA_CHx, uint32_t len)
设置DMA通道传输长度
- void **DMA_Trigger** (**DMA_TypeDef** *DMA_CHx)
触发DMA通道传输
- void **DMA_SetRequest** (**DMA_TypeDef** *DMA_CHx, uint32_t DMA_REQ_x)
设置DMA通道请求源
- void **DMA_CHx_EN** (**DMA_TypeDef** *DMAx, uint32_t Channel_EN)
使能或禁用DMA通道
- void **DMA_Memcpy** (**DMA_TypeDef** *DMAx, void *dest, void *src, uint32_t n)
使用dma实现的内存拷贝
- void **DMA_Mem2Reg** (**DMA_TypeDef** *DMAx, void *dest, void *src, uint32_t n, uint8_t DMA_TRANS_x)
使用dma向特定寄存器写入多个数据，可用于crc计算等需要连续对单个寄存器写入数据的情况

3.11.1 宏定义说明

3.11.1.1 DMA_BYTE_TRANS

```
#define DMA_BYTE_TRANS 0
```

访问位宽， 0:byte

在文件 [lks32mc09x_dma.h](#) 第 59 行定义.

3.11.1.2 DMA_HALFWORD_TRANS

```
#define DMA_HALFWORD_TRANS 1
```

访问位宽， 1:half-word

在文件 [lks32mc09x_dma.h](#) 第 60 行定义.

3.11.1.3 DMA_IE_CH0

```
#define DMA_IE_CH0 BIT0
```

DMA中断使能位MASK

在文件 [lks32mc09x_dma.h](#) 第 98 行定义.

3.11.1.4 DMA_IE_CH1

```
#define DMA_IE_CH1 BIT1
```

在文件 [lks32mc09x_dma.h](#) 第 99 行定义.

3.11.1.5 DMA_IE_CH2

```
#define DMA_IE_CH2 BIT2
```

在文件 [lks32mc09x_dma.h](#) 第 100 行定义.

3.11.1.6 DMA_IE_CH3

```
#define DMA_IE_CH3 BIT3
```

在文件 [lks32mc09x_dma.h](#) 第 101 行定义.

3.11.1.7 DMA_IF_CH0

```
#define DMA_IF_CH0 BIT0
```

DMA中断标志位MASK

在文件 [lks32mc09x_dma.h](#) 第 90 行定义.

3.11.1.8 DMA_IF_CH1

```
#define DMA_IF_CH1 BIT1
```

在文件 [lks32mc09x_dma.h](#) 第 91 行定义.

3.11.1.9 DMA_IF_CH2

```
#define DMA_IF_CH2 BIT2
```

在文件 [lks32mc09x_dma.h](#) 第 92 行定义.

3.11.1.10 DMA_IF_CH3

```
#define DMA_IF_CH3 BIT3
```

在文件 [lks32mc09x_dma.h](#) 第 93 行定义.

3.11.1.11 DMA_MEMORY_2_PERI

```
#define DMA_MEMORY_2_PERI 1
```

内存至外设

在文件 [lks32mc09x_dma.h](#) 第 57 行定义.

3.11.1.12 DMA_PERI_2_MEMORY

```
#define DMA_PERI_2_MEMORY 0
```

外设至内存

在文件 [lks32mc09x_dma.h](#) 第 56 行定义.

3.11.1.13 DMA_REN_ADC0

```
#define DMA_REN_ADC0 BIT0
```

DMA搬运使能源定义

在文件 [lks32mc09x_dma.h](#) 第 9 行定义.

3.11.1.14 DMA_REN_ADC1

```
#define DMA_REN_ADC1 BIT1
```

在文件 [lks32mc09x_dma.h](#) 第 10 行定义.

3.11.1.15 DMA_REN_CMP

```
#define DMA_REN_CMP BIT26
```

在文件 [lks32mc09x_dma.h](#) 第 30 行定义.

3.11.1.16 DMA_REN_DSP

```
#define DMA_REN_DSP BIT3
```

在文件 [lks32mc09x_dma.h](#) 第 11 行定义.

3.11.1.17 DMA_REN_GPIO

```
#define DMA_REN_GPIO BIT29
```

在文件 [lks32mc09x_dma.h](#) 第 31 行定义.

3.11.1.18 DMA_REN_HALL0

```
#define DMA_REN_HALL0 BIT24
```

在文件 [lks32mc09x_dma.h](#) 第 28 行定义.

3.11.1.19 DMA_REN_I2C0_RX

```
#define DMA_REN_I2C0_RX BIT14
```

在文件 [lks32mc09x_dma.h](#) 第 20 行定义.

3.11.1.20 DMA_REN_I2C0_TX

```
#define DMA_REN_I2C0_TX BIT15
```

在文件 [lks32mc09x_dma.h](#) 第 21 行定义.

3.11.1.21 DMA_REN_MCPWM_RE0

```
#define DMA_REN_MCPWM_RE0 BIT12
```

在文件 [lks32mc09x_dma.h](#) 第 18 行定义.

3.11.1.22 DMA_REN_MCPWM_RE1

```
#define DMA_REN_MCPWM_RE1 BIT13
```

在文件 [lks32mc09x_dma.h](#) 第 19 行定义.

3.11.1.23 DMA_REN_SIF

```
#define DMA_REN_SIF BIT25
```

在文件 [lks32mc09x_dma.h](#) 第 29 行定义.

3.11.1.24 DMA_REN_SPI_RX

```
#define DMA_REN_SPI_RX BIT10
```

在文件 [lks32mc09x_dma.h](#) 第 16 行定义.

3.11.1.25 DMA_REN_SPI_TX

```
#define DMA_REN_SPI_TX BIT11
```

在文件 [lks32mc09x_dma.h](#) 第 17 行定义.

3.11.1.26 DMA_REN_SW

```
#define DMA_REN_SW BIT31
```

在文件 [lks32mc09x_dma.h](#) 第 32 行定义.

3.11.1.27 DMA_REN_TIMER0

```
#define DMA_REN_TIMER0 BIT4
```

在文件 [lks32mc09x_dma.h](#) 第 12 行定义.

3.11.1.28 DMA_REN_TIMER1

```
#define DMA_REN_TIMER1 BIT5
```

在文件 [lks32mc09x_dma.h](#) 第 13 行定义.

3.11.1.29 DMA_REN_TIMER2

```
#define DMA_REN_TIMER2 BIT6
```

在文件 [lks32mc09x_dma.h](#) 第 14 行定义.

3.11.1.30 DMA_REN_TIMER3

```
#define DMA_REN_TIMER3 BIT7
```

在文件 [lks32mc09x_dma.h](#) 第 15 行定义.

3.11.1.31 DMA_REN_UART0_RX

```
#define DMA_REN_UART0_RX BIT16
```

在文件 [lks32mc09x_dma.h](#) 第 22 行定义.

3.11.1.32 DMA_REN_UART0_TX

```
#define DMA_REN_UART0_TX BIT17
```

在文件 [lks32mc09x_dma.h](#) 第 23 行定义.

3.11.1.33 DMA_REN_UART1_RX

```
#define DMA_REN_UART1_RX BIT18
```

在文件 [lks32mc09x_dma.h](#) 第 24 行定义.

3.11.1.34 DMA_REN_UART1_TX

```
#define DMA_REN_UART1_TX BIT19
```

在文件 [lks32mc09x_dma.h](#) 第 25 行定义.

3.11.1.35 DMA_REN_UART2_RX

```
#define DMA_REN_UART2_RX BIT20
```

在文件 [lks32mc09x_dma.h](#) 第 26 行定义.

3.11.1.36 DMA_REN_UART2_TX

```
#define DMA_REN_UART2_TX BIT21
```

在文件 [lks32mc09x_dma.h](#) 第 27 行定义.

3.11.1.37 DMA_REQ_ADC0

```
#define DMA_REQ_ADC0 DMA_REQ_ADC0
```

ADC0 触发

在文件 [lks32mc09x_dma.h](#) 第 85 行定义.

3.11.1.38 DMA_REQ_ADC1

```
#define DMA_REQ_ADC1 DMA_REQ_ADC1
```

ADC1 触发

在文件 [lks32mc09x_dma.h](#) 第 84 行定义.

3.11.1.39 DMA_REQ_CMP

```
#define DMA_REQ_CMP DMA_REQ_CMP
```

CMP 触发

在文件 [lks32mc09x_dma.h](#) 第 66 行定义.

3.11.1.40 DMA_REQ_DSP

```
#define DMA_REQ_DSP DMA_REQ_DSP
```

DSP 触发

在文件 [lks32mc09x_dma.h](#) 第 83 行定义.

3.11.1.41 DMA_REQ_GPIO

```
#define DMA_REQ_GPIO DMA_REQ_GPIO
```

GPIO 触发

在文件 [lks32mc09x_dma.h](#) 第 65 行定义.

3.11.1.42 DMA_REQ_HALL

```
#define DMA_REQ_HALL DMA_REQ_HALL
```

HALL0 触发

在文件 [lks32mc09x_dma.h](#) 第 68 行定义.

3.11.1.43 DMA_REQ_I2C0_RX

```
#define DMA_REQ_I2C0_RX DMA_REQ_I2C0_RX
```

I2C0_RX 触发

在文件 [lks32mc09x_dma.h](#) 第 74 行定义.

3.11.1.44 DMA_REQ_I2C0_TX

```
#define DMA_REQ_I2C0_TX DMA_REQ_I2C0_TX
```

I2C0_TX 触发

在文件 [lks32mc09x_dma.h](#) 第 73 行定义.

3.11.1.45 DMA_REQ_MCPWM0

```
#define DMA_REQ_MCPWM0 DMA_REQ_MCPWM0
```

MCPWM0 触发

在文件 [lks32mc09x_dma.h](#) 第 76 行定义.

3.11.1.46 DMA_REQ_MCPWM1

```
#define DMA_REQ_MCPWM1 DMA_REQ_MCPWM1
```

MCPWM1 触发

在文件 [lks32mc09x_dma.h](#) 第 75 行定义.

3.11.1.47 DMA_REQ_SIF

```
#define DMA_REQ_SIF DMA_REQ_SIF
```

SIF 触发

在文件 [lks32mc09x_dma.h](#) 第 67 行定义.

3.11.1.48 DMA_REQ_SPI_RX

```
#define DMA_REQ_SPI_RX DMA_REQ_SPI_RX
```

SPI_RX 触发

在文件 [lks32mc09x_dma.h](#) 第 78 行定义.

3.11.1.49 DMA_REQ_SPI_TX

```
#define DMA_REQ_SPI_TX DMA_REQ_SPI_TX
```

SPI_TX 触发

在文件 [lks32mc09x_dma.h](#) 第 77 行定义.

3.11.1.50 DMA_REQ_SW

```
#define DMA_REQ_SW DMA_REQ_SW
```

SW 触发

在文件 [lks32mc09x_dma.h](#) 第 64 行定义.

3.11.1.51 DMA_REQ_TIMER0

```
#define DMA_REQ_TIMER0 DMA_REQ_TIMER0
```

TIMER0 触发

在文件 [lks32mc09x_dma.h](#) 第 82 行定义.

3.11.1.52 DMA_REQ_TIMER1

```
#define DMA_REQ_TIMER1 DMA_REQ_TIMER1
```

TIMER1 触发

在文件 [lks32mc09x_dma.h](#) 第 81 行定义.

3.11.1.53 DMA_REQ_TIMER2

```
#define DMA_REQ_TIMER2 DMA_REQ_TIMER2
```

TIMER2 触发

在文件 [lks32mc09x_dma.h](#) 第 80 行定义.

3.11.1.54 DMA_REQ_TIMER3

```
#define DMA_REQ_TIMER3 DMA_REQ_TIMER3
```

TIMER3 触发

在文件 [lks32mc09x_dma.h](#) 第 79 行定义.

3.11.1.55 DMA_REQ_UART0_RX

```
#define DMA_REQ_UART0_RX DMA_REQ_UART0_RX
```

UART0_RX 触发

在文件 [lks32mc09x_dma.h](#) 第 72 行定义.

3.11.1.56 DMA_REQ_UART0_TX

```
#define DMA_REQ_UART0_TX DMA_REQ_UART0_TX
```

UART0_TX 触发

在文件 [lks32mc09x_dma.h](#) 第 71 行定义.

3.11.1.57 DMA_REQ_UART1_RX

```
#define DMA_REQ_UART1_RX DMA_REQ_UART1_RX
```

UART1_RX 触发

在文件 [lks32mc09x_dma.h](#) 第 70 行定义.

3.11.1.58 DMA_REQ_UART1_TX

```
#define DMA_REQ_UART1_TX DMA_REQ_UART1_TX
```

UART1_TX 触发

在文件 [lks32mc09x_dma.h](#) 第 69 行定义.

3.11.1.59 DMA_TCIE

```
#define DMA_TCIE BIT0
```

传输完成中断使能，高有效

在文件 [lks32mc09x_dma.h](#) 第 53 行定义.

3.11.1.60 DMA_TEIE

```
#define DMA_TEIE BIT1
```

错误中断使能，高有效

在文件 [lks32mc09x_dma.h](#) 第 54 行定义.

3.11.1.61 DMA_WORD_TRANS

```
#define DMA_WORD_TRANS 2
```

访问位宽， 2:word

在文件 [lks32mc09x_dma.h](#) 第 61 行定义.

3.11.2 函数说明

3.11.2.1 DMA_CHx.EN()

```
void DMA_CHx_EN (
    DMA_TypeDef * DMAx,
    uint32_t Channel_EN )
```

使能或禁用DMA通道

参数

DMAx	DMA寄存器基地址
Channel_EN	通道使能MASK

在文件 [lks32mc09x_dma.c](#) 第 218 行定义.

3.11.2.2 DMA_ClearIRQFlag()

```
void DMA_ClearIRQFlag (
    uint32_t DMA_IF_x )
```

清除DMA中断标志位

参数

DMA_↔ IF_x	中断标志位MASK

在文件 [lks32mc09x_dma.c](#) 第 106 行定义.

3.11.2.3 DMA_Disable()

```
void DMA_Disable (
    void )
```

禁用DMA模块

在文件 [lks32mc09x_dma.c](#) 第 134 行定义.

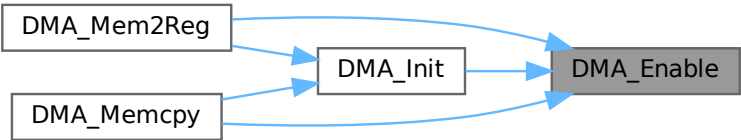
3.11.2.4 DMA_Enable()

```
void DMA_Enable (
    void )
```

使能DMA模块

在文件 [lks32mc09x_dma.c](#) 第 126 行定义.

这是这个函数的调用关系图:



3.11.2.5 DMA_GetFlag()

```
uint32_t DMA_GetFlag (
    uint32_t DMA_IF_x )
```

获取DMA标志位状态

参数

<i>DMA_↔ IF_x</i>	标志位MASK

返回

标志位状态

在文件 [lks32mc09x_dma.c](#) 第 117 行定义.

函数调用图:



3.11.2.6 DMA_GetIRQFlag()

```
uint32_t DMA_GetIRQFlag (
    uint32_t DMA_IF_x )
```

获取DMA中断标志位状态

参数

<i>DMA_↔ IF_x</i>	中断标志位MASK

返回

中断标志位状态

在文件 [lks32mc09x_dma.c](#) 第 96 行定义.

这是这个函数的调用关系图:



3.11.2.7 DMA_Init()

```
void DMA_Init (
    DMA_TypeDef * DMAx,
    DMA_InitTypeDef * DMAInitStruct )
```

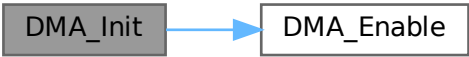
初始化DMA模块

参数

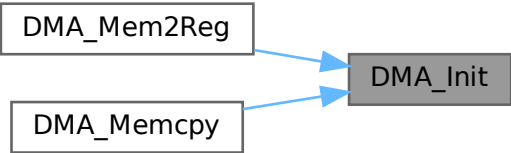
DMAx	DMA寄存器基地址
DMAInitStruct	初始化结构体指针

在文件 [lks32mc09x_dma.c](#) 第 11 行定义.

函数调用图:



这是这个函数的调用关系图:



3.11.2.8 DMA_Mem2Reg()

```
void DMA_Mem2Reg (
    DMA_TypeDef * DMAx,
    void * dest,
    void * src,
    uint32_t n,
    uint8_t DMA_TRANS_x )
```

使用dma向特定寄存器写入多个数据，可用于crc计算等需要连续对单个寄存器写入数据的情况

参数

<i>DMAx</i>	DMA寄存器基地址
<i>dest</i>	目标内存指针
<i>src</i>	源内存指针
<i>n</i>	要复制的字节数，如果使用dma1或dma2，字节数应小于0xff，如果使用dma0或dma3，字节数小于0xffff
<i>DMA_↔TRANS_x</i>	位宽，输入输出位宽一致

< 源地址访问位宽 0: Byte 1: Halfword 2: Word

< 目的地址访问位宽 0: Byte 1: Halfword 2: Word

< 源地址递增使能

< 目的地址递增使能

< 循环模式使能

< 多轮传输使能，多个DMA 请求完成传输

< 通道使能

< DMA 通道 x 数据搬运次数。

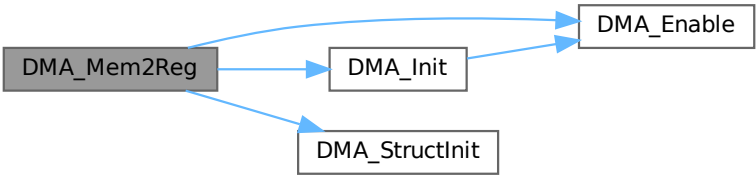
< DMA 通道 x 源地址

< DMA 通道 x 目的地址

< DMA 请求使能

在文件 [lks32mc09x_dma.c](#) 第 305 行定义.

函数调用图:



3.11.2.9 DMA_Memcpy()

```
void DMA_Memcpy (
    DMA_TypeDef * DMAx,
    void * dest,
    void * src,
    uint32_t n )
```

使用dma实现的内存拷贝

参数

<i>DMAx</i>	DMA寄存器基地址
<i>dest</i>	目标内存指针
<i>src</i>	源内存指针
<i>n</i>	要复制的字节数，如果使用dma1或dma2，字节数应小于0xff，如果使用dma0或dma3，字节数小于0xffff

< 源地址访问位宽 0: Byte 1: Halfword 2: Word

< 目的地址访问位宽 0: Byte 1: Halfword 2: Word

< 源地址递增使能

< 目的地址递增使能

< 循环模式使能

< 多轮传输使能，多个DMA 请求完成传输

< 通道使能

< DMA 通道 x 数据搬运次数。

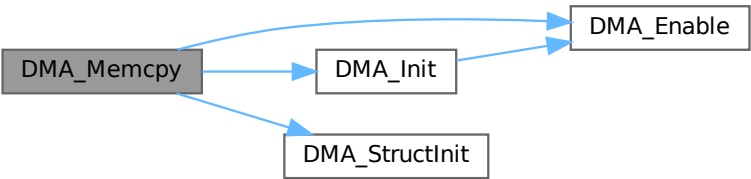
< DMA 通道 x 源地址

< DMA 通道 x 目的地址

< DMA 请求使能

在文件 [lks32mc09x_dma.c](#) 第 237 行定义.

函数调用图:



3.11.2.10 DMA_Reset()

```
void DMA_Reset (
    void )
```

复位DMA模块

在文件 [lks32mc09x_dma.c](#) 第 142 行定义.

函数调用图:



3.11.2.11 DMA_SetAddr()

```
void DMA_SetAddr (
    DMA_TypeDef * DMAx,
    uint32_t addr1,
    uint32_t addr2 )
```

设置DMA通道源地址和目的地址

参数

<i>DMA_CHx</i>	DMA通道寄存器基地址
<i>addr1</i>	源地址
<i>addr2</i>	目的地址
<i>DMAx</i>	DMA通道寄存器基地址
<i>addr1</i>	源地址
<i>addr2</i>	目的地址

在文件 [lks32mc09x_dma.c](#) 第 154 行定义.

3.11.2.12 DMA_SetLength()

```
void DMA_SetLength (
    DMA_TypeDef * DMAx,
    uint32_t len )
```

设置DMA通道传输长度

参数

<i>DMA_CHx</i>	DMA通道寄存器基地址
<i>len</i>	传输长度
<i>DMAx</i>	DMA通道寄存器基地址
<i>len</i>	传输长度

在文件 [lks32mc09x_dma.c](#) 第 166 行定义.

3.11.2.13 DMA_SetRequest()

```
void DMA_SetRequest (
    DMA_TypeDef * DMAx,
    uint32_t DMA_REQ_x )
```

设置DMA通道请求源

参数

<i>DMA_CHx</i>	DMA通道寄存器基地址
<i>DMA_↔REQ_x</i>	请求源MASK
<i>DMAx</i>	DMA通道寄存器基地址
<i>DMA_↔REQ_x</i>	请求源MASK

在文件 [lks32mc09x_dma.c](#) 第 207 行定义.

3.11.2.14 DMA_StructInit()

```
void DMA_StructInit (
    DMA_InitTypeDef * DMAInitStruct )
```

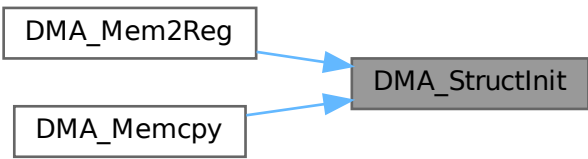
将DMA初始化结构体重置为默认值

参数

<i>DMAInitStruct</i>	初始化结构体指针

在文件 [lks32mc09x_dma.c](#) 第 81 行定义.

这是这个函数的调用关系图:



3.11.2.15 DMA_Trigger()

```
void DMA_Trigger (
    DMA_TypeDef * DMAx )
```

触发DMA通道传输

参数

DMA_CHx	DMA通道寄存器基地址
DMAx	DMA通道寄存器基地址

在文件 [lks32mc09x_dma.c](#) 第 196 行定义.

3.12 lks32mc09x_dma.h

[浏览该文件的文档.](#)

```
00001 #ifndef __LKS32MC09X_DMA_H__
00002 #define __LKS32MC09X_DMA_H__
00003
00004 #include "lks32mc09x_reg.h"
00005
00009 #define DMA_REN_ADC0      BIT0
00010 #define DMA_REN_ADC1      BIT1
00011 #define DMA_REN_DSP       BIT3
00012 #define DMA_REN_TIMER0    BIT4
00013 #define DMA_REN_TIMER1    BIT5
00014 #define DMA_REN_TIMER2    BIT6
00015 #define DMA_REN_TIMER3    BIT7
00016 #define DMA_REN_SPI_RX    BIT10
00017 #define DMA_REN_SPI_TX    BIT11
00018 #define DMA_REN_MCPWM_RE0 BIT12
00019 #define DMA_REN_MCPWM_RE1 BIT13
00020 #define DMA_REN_I2C0_RX   BIT14
00021 #define DMA_REN_I2C0_TX   BIT15
00022 #define DMA_REN_UART0_RX  BIT16
00023 #define DMA_REN_UART0_TX  BIT17
00024 #define DMA_REN_UART1_RX  BIT18
00025 #define DMA_REN_UART1_TX  BIT19
00026 #define DMA_REN_UART2_RX  BIT20
00027 #define DMA_REN_UART2_TX  BIT21
00028 #define DMA_REN_HALL0     BIT24
00029 #define DMA_REN_SIF       BIT25
00030 #define DMA_REN_CMP       BIT26
00031 #define DMA_REN_GPIO      BIT29
```



```

00032 #define DMA.REN_SW          BIT31
00033
00037 typedef struct
00038 {
00039     uint8_t  IRQ_EN;
00040     uint8_t  SBTW;
00041     uint8_t  DBTW;
00042     uint8_t  SINC;
00043     uint8_t  DINC;
00044     uint8_t  CIRC;
00045     uint8_t  RMODE;
00046     uint8_t  EN;
00047     uint16_t TMS;
00048     uint32_t SADR;
00049     uint32_t DADR;
00050     uint32_t REN;
00051 } DMA_InitTypeDef;
00052
00053 #define DMA.TCIE          BIT0
00054 #define DMA.TEIE          BIT1
00055
00056 #define DMA.PERI_2_MEMORY 0
00057 #define DMA.MEMORY_2_PERI 1
00058
00059 #define DMA.BYTE_TRANS     0
00060 #define DMA.HALFWORD_TRANS 1
00061 #define DMA.WORD_TRANS     2
00062
00063 // 兼容写法
00064 #define DMA.REQ_SW          DMA.REQ_SW
00065 #define DMA.REQ_GPIO        DMA.REQ_GPIO
00066 #define DMA.REQ_CMP          DMA.REQ_CMP
00067 #define DMA.REQ_SIF          DMA.REQ_SIF
00068 #define DMA.REQ_HALL          DMA.REQ_HALL
00069 #define DMA.REQ_UART1_TX     DMA.REQ_UART1_TX
00070 #define DMA.REQ_UART1_RX     DMA.REQ_UART1_RX
00071 #define DMA.REQ_UART0_TX     DMA.REQ_UART0_TX
00072 #define DMA.REQ_UART0_RX     DMA.REQ_UART0_RX
00073 #define DMA.REQ_I2C0_TX      DMA.REQ_I2C0_TX
00074 #define DMA.REQ_I2C0_RX      DMA.REQ_I2C0_RX
00075 #define DMA.REQ_MCPWM1       DMA.REQ_MCPWM1
00076 #define DMA.REQ_MCPWM0       DMA.REQ_MCPWM0
00077 #define DMA.REQ_SPI_TX       DMA.REQ_SPI_TX
00078 #define DMA.REQ_SPI_RX       DMA.REQ_SPI_RX
00079 #define DMA.REQ_TIMER3        DMA.REQ_TIMER3
00080 #define DMA.REQ_TIMER2        DMA.REQ_TIMER2
00081 #define DMA.REQ_TIMER1        DMA.REQ_TIMER1
00082 #define DMA.REQ_TIMER0        DMA.REQ_TIMER0
00083 #define DMA.REQ_DSP           DMA.REQ_DSP
00084 #define DMA.REQ_ADC1          DMA.REQ_ADC1
00085 #define DMA.REQ_ADC0          DMA.REQ_ADC0
00086
00090 #define DMA.IF.CH0 BIT0
00091 #define DMA.IF.CH1 BIT1
00092 #define DMA.IF.CH2 BIT2
00093 #define DMA.IF.CH3 BIT3
00094
00098 #define DMA.IE.CH0 BIT0
00099 #define DMA.IE.CH1 BIT1
00100 #define DMA.IE.CH2 BIT2
00101 #define DMA.IE.CH3 BIT3
00102
00109 void DMA_Init(DMA_TypeDef *DMAx, DMA_InitTypeDef *DMAInitStruct);
00110
00116 void DMA_StructInit(DMA_InitTypeDef *DMAInitStruct);
00117
00124 uint32_t DMA_GetIRQFlag(uint32_t DMA_IF_x);
00125
00131 void DMA_ClearIRQFlag(uint32_t DMA_IF_x);
00132
00139 uint32_t DMA_GetFlag(uint32_t DMA_IF_x);
00140
00144 void DMA_Enable(void);
00145
00149 void DMA_Disable(void);
00150
00154 void DMA_Reset(void);
00155
00163 void DMA_SetAddr(DMA_TypeDef *DMA_CHx, uint32_t addr1, uint32_t addr2);
00164
00171 void DMA_SetLength(DMA_TypeDef *DMA_CHx, uint32_t len);
00172
00178 void DMA_Trigger(DMA_TypeDef *DMA_CHx);
00179
00186 void DMA_SetRequest(DMA_TypeDef *DMA_CHx, uint32_t DMA_REQ_x);
00187
00194 void DMA_CHx_EN(DMA_TypeDef *DMAx, uint32_t Channel_EN);

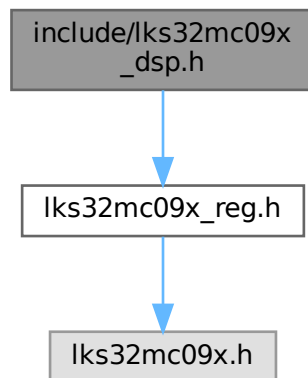
```

```
00195
00203 void DMA_Memcpy(DMA_TypeDef *DMAx, void *dest, void *src, uint32_t n);
00212 void DMA_Mem2Reg(DMA_TypeDef *DMAx, void *dest, void *src, uint32_t n, uint8_t DMA_TRANS_x);
00213 #endif
```

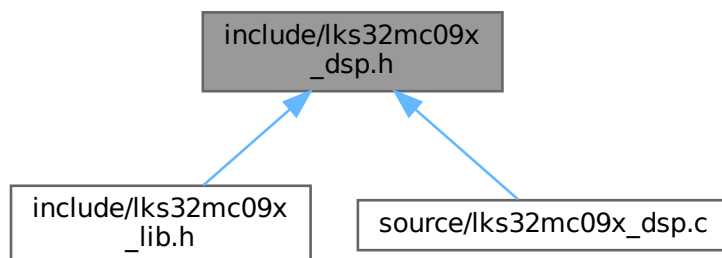
3.13 include/lks32mc09x_dsp.h 文件参考

#include "lks32mc09x_reg.h"

lks32mc09x_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:



函数

- void **DSP_Init** (void)
初始化DSP模块
- void **DSP_Enable** (void)

- 使能DSP模块
- void **DSP_Disable** (void)

禁用DSP模块
 - void **DSP_Reset** (void)

复位DSP模块
 - s32 **DSP_Div** (s32 a, s32 b)

执行整数除法
 - uint16_t **DSP_Rms** (s16 a, s16 b)

计算均方根值
 - s32 **DSP_Mod** (s32 x, s32 y)

执行取模运算
 - uint16_t **DSP_Sqrt** (uint32_t val)

计算平方根

3.13.1 函数说明

3.13.1.1 DSP_Disable()

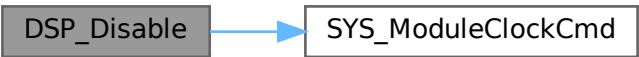
```
void DSP_Disable (  
    void )
```

禁用DSP模块

该函数用于禁用DSP模块，停止处理数据。

在文件 [lks32mc09x_dsp.c](#) 第 30 行定义.

函数调用图:



3.13.1.2 DSP_Div()

```
s32 DSP_Div (  
    s32 a,  
    s32 b )
```

执行整数除法

参数

<i>a</i>	被除数
<i>b</i>	除数

返回

â†

在文件 [lks32mc09x_dsp.c](#) 第 52 行定义.

3.13.1.3 DSP.Enable()

```
void DSP_Enable (
    void )
```

使能DSP模块

该函数用于使能DSP模块，开始处理数据。

在文件 [lks32mc09x_dsp.c](#) 第 20 行定义.

函数调用图:



3.13.1.4 DSP.Init()

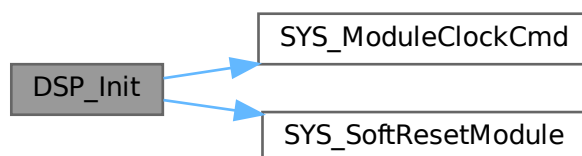
```
void DSP_Init (
    void )
```

初始化DSP模块

该函数用于初始化DSP模块，使其处于可用状态。

在文件 [lks32mc09x_dsp.c](#) 第 9 行定义.

函数调用图:



3.13.1.5 DSP_Mod()

```
s32 DSP_Mod (
    s32 a,
    s32 b )
```

执行取模运算

参数

<i>x</i>	被除数
<i>y</i>	除数

返回
余数

参数

<i>a</i>	被除数
<i>b</i>	除数

返回
余数

在文件 [lks32mc09x_dsp.c](#) 第 83 行定义.

3.13.1.6 DSP_Reset()

```
void DSP_Reset (
    void )
```

复位DSP模块

该函数用于复位DSP模块，使其恢复到初始状态。

在文件 [lks32mc09x_dsp.c](#) 第 40 行定义.

函数调用图:



3.13.1.7 DSP_Rms()

```
uint16_t DSP_Rms (
    s16 a,
    s16 b )
```

计算均方根值

参数

<i>a</i>	输入值1
<i>b</i>	输入值2

返回

均方根值

在文件 [lks32mc09x_dsp.c](#) 第 67 行定义.

3.13.1.8 DSP_Sqrt()

```
uint16_t DSP_Sqrt (
    uint32_t val )
```

计算平方根

参数

<i>val</i>	输入值

返回

平方根值

在文件 [lks32mc09x_dsp.c](#) 第 96 行定义.

3.14 lks32mc09x_dsp.h

[浏览该文件的文档.](#)

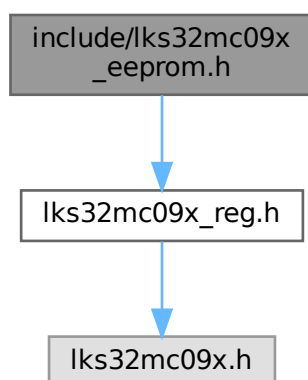
```
00001 #ifndef __LKS32MC09X_DSP_H__
00002 #define __LKS32MC09X_DSP_H__
00003
00004 #include "lks32mc09x_reg.h"
00005
00011 void DSP_Init(void);
00012
00018 void DSP_Enable(void);
00019
00025 void DSP_Disable(void);
00026
00032 void DSP_Reset(void);
00033
```

```
00041 s32 DSP_Div(s32 a, s32 b);
00042
00050 uint16_t DSP_Rms(s16 a, s16 b);
00051
00059 s32 DSP_Mod(s32 x, s32 y);
00060
00067 uint16_t DSP_Sqrt(uint32_t val);
00068
00069 #endif // __LKS32MC09X_DSP_H__
```

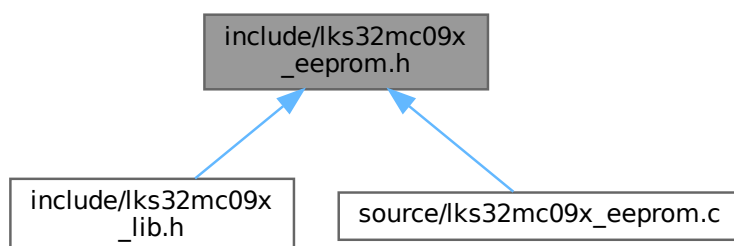
3.15 include/lks32mc09x_eeprom.h 文件参考

```
#include "lks32mc09x_reg.h"
```

lks32mc09x_eeprom.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 [EEPROM_InitTypeDef](#)
EEPROM初始化结构体定义

宏定义

- `#define EEPROM_BAUD_100K 0`
EEPROM波特率100K
- `#define EEPROM_BAUD_400K 1`
EEPROM波特率400K
- `#define EEPROM_BAUD_1M 2`
EEPROM波特率1M
- `#define EEPROM_BAUD_1M2 3`
EEPROM波特率1M2
- `#define EEPROM_ADDR_LEN_8 0`
数据地址长度为8位 影响 `EEPROM_Read` `EEPROM_Write`
- `#define EEPROM_ADDR_LEN_16 1`
数据地址长度位16位 影响 `EEPROM_Read` `EEPROM_Write`

函数

- `void EEPROM_Init (EEPROM_InitTypeDef *EEPROM_InitStruct)`
初始化
- `void EEPROM_StructInit (EEPROM_InitTypeDef *EEPROM_InitStruct)`
初始化`EEPROM_InitStruct`结构体为默认值
- `void EEPROM_InitDefault (void)`
初始化为默认配置，IO不交换，波特率100k，地址数据长度16位，读写时响应中断
- `void EEPROM_Enable (void)`
EEPROM使能
- `void EEPROM_Disable (void)`
EEPROM失能
- `void EEPROM_Reset (void)`
EEPROM复位
- `uint8_t EEPROM_Read (uint16_t eeprom_addr)`
读取eeprom内数据
- `void EEPROM_Write (uint16_t eeprom_addr, uint8_t data)`
向eeprom写入数据
- `void EEPROM_WriteData (uint8_t eepromaddr, u16 taddr, u8 *data, uint32_t len)`
向eeprom写入数据(多字节)，09上eeprom本身不支持多字节模式，这里用n个单次读写实现，执行时间为len*5mS（典型值）
- `void EEPROM_ReadData (uint8_t eepromaddr, u16 taddr, u8 *data, uint32_t len)`
读取eeprom内数据，09上eeprom本身不支持多字节模式，这里用n个单次读写实现

3.15.1 宏定义说明

3.15.1.1 EEPROM_ADDR_LEN_16

```
#define EEPROM_ADDR_LEN_16 1
```

数据地址长度位16位 影响 `EEPROM_Read` `EEPROM_Write`

在文件 `lks32mc09x_eeprom.h` 第 11 行定义.

3.15.1.2 EEPROM_ADDR_LEN_8

```
#define EEPROM_ADDR_LEN_8 0
```

数据地址长度为8位 影响 EEPROM_Read EEPROM_Write

在文件 [lks32mc09x_eeprom.h](#) 第 10 行定义.

3.15.1.3 EEPROM_BAUD_100K

```
#define EEPROM_BAUD_100K 0
```

EEPROM波特率100K

在文件 [lks32mc09x_eeprom.h](#) 第 5 行定义.

3.15.1.4 EEPROM_BAUD_1M

```
#define EEPROM_BAUD_1M 2
```

EEPROM波特率1M

在文件 [lks32mc09x_eeprom.h](#) 第 7 行定义.

3.15.1.5 EEPROM_BAUD_1M2

```
#define EEPROM_BAUD_1M2 3
```

EEPROM波特率1M2

在文件 [lks32mc09x_eeprom.h](#) 第 8 行定义.

3.15.1.6 EEPROM_BAUD_400K

```
#define EEPROM_BAUD_400K 1
```

EEPROM波特率400K

在文件 [lks32mc09x_eeprom.h](#) 第 6 行定义.

3.15.2 函数说明

3.15.2.1 EEPROM_Disable()

```
void EEPROM_Disable (
    void )
```

EEPROM失能

在文件 [lks32mc09x_eeeprom.c](#) 第 52 行定义.

函数调用图:



3.15.2.2 EEPROM_Enable()

```
void EEPROM_Enable (
    void )
```

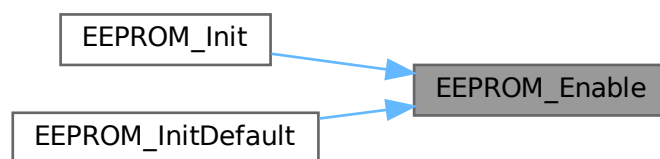
EEPROM使能

在文件 [lks32mc09x_eeeprom.c](#) 第 45 行定义.

函数调用图:



这是这个函数的调用关系图:



3.15.2.3 EEPROM_Init()

```
void EEPROM_Init (
    EEPROM_InitTypeDef * EEPROM_InitStruct )
```

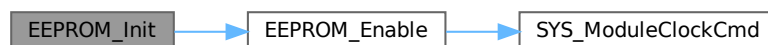
初始化

参数

<i>EEPROM_InitStruct</i>	EEPROM初始化结构体指针

在文件 [lks32mc09x_eeprom.c](#) 第 13 行定义.

函数调用图:



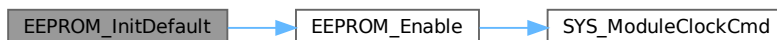
3.15.2.4 EEPROM_InitDefault()

```
void EEPROM_InitDefault (
    void )
```

初始化为默认配置，IO不交换，波特率100k，地址数据长度16位，读写时响应中断

在文件 [lks32mc09x_eeprom.c](#) 第 37 行定义.

函数调用图:



3.15.2.5 EEPROM_Read()

```
uint8_t EEPROM_Read (
    uint16_t eeprom_addr )
```

读取eeprom内数据

参数

<i>eeeprom_addr</i>	eeeprom数据地址

返回

数据

在文件 [lks32mc09x_eeeprom.c](#) 第 69 行定义.

这是这个函数的调用关系图:



3.15.2.6 EEPROM_ReadData()

```
void EEPROM_ReadData (
    uint8_t eeepromaddr,
    u16 taddr,
    u8 * data,
    uint32_t len )
```

读取eeeprom内数据，09上eeeprom本身不支持多字节模式，这里用n个单次读写实现

参数

<i>eeepromaddr</i>	eeeprom模块设备地址，例如0xa0
<i>taddr</i>	eeeprom数据的起始地址
<i>data</i>	存放数据的地址
<i>len</i>	数据长度

在文件 [lks32mc09x_eeeprom.c](#) 第 145 行定义.

函数调用图:



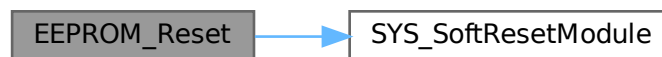
3.15.2.7 EEPROM_Reset()

```
void EEPROM_Reset (
    void )
```

EEPROM复位

在文件 [lks32mc09x_eeprom.c](#) 第 59 行定义.

函数调用图:



3.15.2.8 EEPROM_StructInit()

```
void EEPROM_StructInit (
    EEPROM_InitTypeDef * EEPROM_InitStruct )
```

初始化EEPROM_InitStruct结构体为默认值

在文件 [lks32mc09x_eeprom.c](#) 第 26 行定义.

3.15.2.9 EEPROM_Write()

```
void EEPROM_Write (
    uint16_t eeprom_addr,
    uint8_t data )
```

向eeprom写入数据

参数

<i>eeeprom_addr</i>	eeeprom数据地址
<i>data</i>	要写入的数据

在文件 [lks32mc09x_eeeprom.c](#) 第 94 行定义.

这是这个函数的调用关系图:



3.15.2.10 EEPROM_WriteData()

```
void EEPROM_WriteData (
    uint8_t eeepromaddr,
    u16 taddr,
    u8 * data,
    uint32_t len )
```

向eeeprom写入数据(多字节), 09上eeeprom本身不支持多字节模式, 这里用n个单次读写实现, 执行时间为len*5mS (典型值)

参数

<i>eeepromaddr</i>	eeeprom模块设备地址, 例如0xa0
<i>taddr</i>	eeeprom数据的起始地址
<i>data</i>	要写入的数据地址
<i>len</i>	要写入的长度

在文件 [lks32mc09x_eeeprom.c](#) 第 128 行定义.

函数调用图:



3.16 lks32mc09x_eeprom.h

[浏览该文件的文档.](#)

```

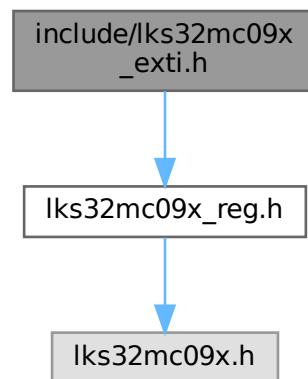
00001 #ifndef __LKS32MC09X_EEPROM_H__
00002 #define __LKS32MC09X_EEPROM_H__
00003
00004 #include "lks32mc09x_reg.h"
00005 #define EEPROM_BAUD_100K 0
00006 #define EEPROM_BAUD_400K 1
00007 #define EEPROM_BAUD_1M 2
00008 #define EEPROM_BAUD_1M2 3
00009
00010 #define EEPROM_ADDR_LEN_8 0
00011 #define EEPROM_ADDR_LEN_16 1
00012
00016 typedef struct
00017 {
00018     uint32_t addr;
00019     uint32_t EEPROM_BAUD_x;
00020     uint32_t RDY_RLS;
00021     uint32_t IO_SWITCH;
00022     uint32_t ADDR_LEN_x;
00023 } EEPROM_InitTypeDef;
00028 void EEPROM_Init(EEPROM_InitTypeDef *EEPROM_InitStruct);
00032 void EEPROM_StructInit(EEPROM_InitTypeDef *EEPROM_InitStruct);
00036 void EEPROM_InitDefault(void);
00040 void EEPROM_Enable(void);
00044 void EEPROM_Disable(void);
00048 void EEPROM_Reset(void);
00049
00055 uint8_t EEPROM_Read(uint16_t eeprom_addr);
00061 void EEPROM_Write(uint16_t eeprom_addr, uint8_t data);
00069 void EEPROM_WriteData(uint8_t eeprom_addr, u16_t taddr, u8 *data, uint32_t len);
00077 void EEPROM_ReadData(uint8_t eeprom_addr, u16_t taddr, u8 *data, uint32_t len);
00078
00079 #endif

```

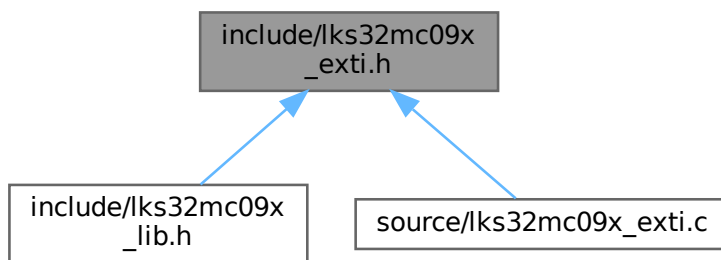
3.17 include/lks32mc09x_exti.h 文件参考

#include "lks32mc09x_reg.h"

lks32mc09x_exti.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 EXTI_CR_P0_0 0
- #define EXTI_CR_P0_2 1
- #define EXTI_CR_P0_11 2
- #define EXTI_CR_P0_12 3
- #define EXTI_CR_P0_13 4
- #define EXTI_CR_P0_14 5
- #define EXTI_CR_P0_15 6
- #define EXTI_CR_P1_0 7
- #define EXTI_CR_P2_0 8
- #define EXTI_CR_P2_3 9
- #define EXTI_CR_P2_4 10
- #define EXTI_CR_P2_5 11
- #define EXTI_CR_P2_6 12
- #define EXTI_CR_P2_7 13
- #define EXTI_CR_P2_9 14
- #define EXTI_CR_P2_13 15
- #define EXTI_IF_P2_0 BIT15
- #define EXTI_IF_P2_13 BIT14
- #define EXTI_IF_P2_6 BIT13
- #define EXTI_IF_P2_5 BIT12
- #define EXTI_IF_P2_4 BIT11
- #define EXTI_IF_P2_3 BIT10
- #define EXTI_IF_P2_9 BIT9
- #define EXTI_IF_P2_7 BIT8
- #define EXTI_IF_P1_0 BIT7
- #define EXTI_IF_P0_15 BIT6
- #define EXTI_IF_P0_14 BIT5
- #define EXTI_IF_P0_13 BIT4
- #define EXTI_IF_P0_12 BIT3
- #define EXTI_IF_P0_11 BIT2
- #define EXTI_IF_P0_2 BIT1
- #define EXTI_IF_P0_0 BIT0
- #define EXTI_CLKO_HSE BIT4
- #define EXTI_CLKO_ADC BIT3
- #define EXTI_CLKO_PLL BIT2
- #define EXTI_CLKO_HSI BIT1
- #define EXTI_CLKO_LSI BIT0
- #define EXTI_CLKO_0 0

枚举

- enum `EXTI_Mode_TypeDef` { `EXTI_Trigger_None` = 0x00 , `EXTI_Trigger_Negedge` = 0x01 , `EXTI_Trigger_Posedge` = 0x02 , `EXTI_Trigger_Edge` = 0x03 }

*EXT*触发信号定义

函数

- void `EXTI_Reset` (`EXTI_TypeDef` *`EXTIx`)
复位*EXT*模块
- void `EXTI_TriggerConfig` (`EXTI_TypeDef` *`EXTIx`, uint8_t `EXTI_CR_x`, `EXTI_Mode_TypeDef` `EXTI_Trigger`)
配置*GPIO*引脚的中断触发模式
- void `EXTI_EnableIRQFlag` (`EXTI_TypeDef` *`EXTIx`, uint32_t `EXTI_IF_x`)
使能*GPIO*引脚的中断标志
- void `EXTI_DisableIRQFlag` (`EXTI_TypeDef` *`EXTIx`, uint32_t `EXTI_IF_x`)
失能*GPIO*引脚的中断标志
- void `EXTI_ClearIRQFlag` (`EXTI_TypeDef` *`EXTIx`, uint32_t `EXTI_IF_x`)
清除*GPIO*引脚的中断标志
- uint8_t `EXTI_GetIRQFlag` (`EXTI_TypeDef` *`EXTIx`, uint32_t `EXTI_IF_x`)
获取*GPIO*引脚的中断标志状态
- void `EXTI_SetClko` (`EXTI_TypeDef` *`EXTIx`, uint32_t `EXTI_CLKO_x`)
输出时钟选择

3.17.1 宏定义说明

3.17.1.1 EXTI_CLKO_0

```
#define EXTI_CLKO_0 0
```

在文件 `lks32mc09x_exti.h` 第 44 行定义.

3.17.1.2 EXTI_CLKO_ADC

```
#define EXTI_CLKO_ADC BIT3
```

在文件 `lks32mc09x_exti.h` 第 40 行定义.

3.17.1.3 EXTI_CLKO_HSE

```
#define EXTI_CLKO_HSE BIT4
```

在文件 `lks32mc09x_exti.h` 第 39 行定义.

3.17.1.4 EXTI_CLKO_HSI

```
#define EXTI_CLKO_HSI BIT1
```

在文件 `lks32mc09x_exti.h` 第 42 行定义.

3.17.1.5 EXTI_CLKO_LSI

```
#define EXTI_CLKO_LSI BIT0
```

在文件 [lks32mc09x_exti.h](#) 第 43 行定义.

3.17.1.6 EXTI_CLKO_PLL

```
#define EXTI_CLKO_PLL BIT2
```

在文件 [lks32mc09x_exti.h](#) 第 41 行定义.

3.17.1.7 EXTI_CR_P0_0

```
#define EXTI_CR_P0_0 0
```

在文件 [lks32mc09x_exti.h](#) 第 5 行定义.

3.17.1.8 EXTI_CR_P0_11

```
#define EXTI_CR_P0_11 2
```

在文件 [lks32mc09x_exti.h](#) 第 7 行定义.

3.17.1.9 EXTI_CR_P0_12

```
#define EXTI_CR_P0_12 3
```

在文件 [lks32mc09x_exti.h](#) 第 8 行定义.

3.17.1.10 EXTI_CR_P0_13

```
#define EXTI_CR_P0_13 4
```

在文件 [lks32mc09x_exti.h](#) 第 9 行定义.

3.17.1.11 EXTI_CR_P0_14

```
#define EXTI_CR_P0_14 5
```

在文件 [lks32mc09x_exti.h](#) 第 10 行定义.

3.17.1.12 EXTI_CR_P0_15

```
#define EXTI_CR_P0_15 6
```

在文件 [lks32mc09x_exti.h](#) 第 11 行定义.

3.17.1.13 EXTI_CR_P0_2

```
#define EXTI_CR_P0_2 1
```

在文件 [lks32mc09x_exti.h](#) 第 6 行定义.

3.17.1.14 EXTI_CR_P1_0

```
#define EXTI_CR_P1_0 7
```

在文件 [lks32mc09x_exti.h](#) 第 12 行定义.

3.17.1.15 EXTI_CR_P2_0

```
#define EXTI_CR_P2_0 8
```

在文件 [lks32mc09x_exti.h](#) 第 13 行定义.

3.17.1.16 EXTI_CR_P2_13

```
#define EXTI_CR_P2_13 15
```

在文件 [lks32mc09x_exti.h](#) 第 20 行定义.

3.17.1.17 EXTI_CR_P2_3

```
#define EXTI_CR_P2_3 9
```

在文件 [lks32mc09x_exti.h](#) 第 14 行定义.

3.17.1.18 EXTI_CR_P2_4

```
#define EXTI_CR_P2_4 10
```

在文件 [lks32mc09x_exti.h](#) 第 15 行定义.

3.17.1.19 EXTI_CR_P2_5

```
#define EXTI_CR_P2_5 11
```

在文件 [lks32mc09x_exti.h](#) 第 16 行定义.

3.17.1.20 EXTI_CR_P2_6

```
#define EXTI_CR_P2_6 12
```

在文件 [lks32mc09x_exti.h](#) 第 17 行定义.

3.17.1.21 EXTI_CR_P2_7

```
#define EXTI_CR_P2_7 13
```

在文件 [lks32mc09x_exti.h](#) 第 18 行定义.

3.17.1.22 EXTI_CR_P2_9

```
#define EXTI_CR_P2_9 14
```

在文件 [lks32mc09x_exti.h](#) 第 19 行定义.

3.17.1.23 EXTI_IF_P0_0

```
#define EXTI_IF_P0_0 BIT0
```

在文件 [lks32mc09x_exti.h](#) 第 37 行定义.

3.17.1.24 EXTI_IF_P0_11

```
#define EXTI_IF_P0_11 BIT2
```

在文件 [lks32mc09x_exti.h](#) 第 35 行定义.

3.17.1.25 EXTI_IF_P0_12

```
#define EXTI_IF_P0_12 BIT3
```

在文件 [lks32mc09x_exti.h](#) 第 34 行定义.

3.17.1.26 EXTI_IF_P0_13

```
#define EXTI_IF_P0_13 BIT4
```

在文件 [lks32mc09x_exti.h](#) 第 33 行定义.

3.17.1.27 EXTI_IF_P0_14

```
#define EXTI_IF_P0_14 BIT5
```

在文件 [lks32mc09x_exti.h](#) 第 32 行定义.

3.17.1.28 EXTI_IF_P0_15

```
#define EXTI_IF_P0_15 BIT6
```

在文件 [lks32mc09x_exti.h](#) 第 31 行定义.

3.17.1.29 EXTI_IF_P0_2

```
#define EXTI_IF_P0_2 BIT1
```

在文件 [lks32mc09x_exti.h](#) 第 36 行定义.

3.17.1.30 EXTI_IF_P1_0

```
#define EXTI_IF_P1_0 BIT7
```

在文件 [lks32mc09x_exti.h](#) 第 30 行定义.

3.17.1.31 EXTI_IF_P2_0

```
#define EXTI_IF_P2_0 BIT15
```

在文件 [lks32mc09x_exti.h](#) 第 22 行定义.

3.17.1.32 EXTI_IF_P2_13

```
#define EXTI_IF_P2_13 BIT14
```

在文件 [lks32mc09x_exti.h](#) 第 23 行定义.

3.17.1.33 EXTI_IF_P2_3

```
#define EXTI_IF_P2_3 BIT10
```

在文件 [lks32mc09x_exti.h](#) 第 27 行定义.

3.17.1.34 EXTI_IF_P2_4

```
#define EXTI_IF_P2_4 BIT11
```

在文件 [lks32mc09x_exti.h](#) 第 26 行定义.

3.17.1.35 EXTI_IF_P2_5

```
#define EXTI_IF_P2_5 BIT12
```

在文件 [lks32mc09x_exti.h](#) 第 25 行定义.

3.17.1.36 EXTI_IF_P2_6

```
#define EXTI_IF_P2_6 BIT13
```

在文件 [lks32mc09x_exti.h](#) 第 24 行定义.

3.17.1.37 EXTI_IF_P2_7

#define EXTI_IF_P2_7 BIT8

在文件 lks32mc09x_exti.h 第 29 行定义.

3.17.1.38 EXTI_IF_P2_9

#define EXTI_IF_P2_9 BIT9

在文件 lks32mc09x_exti.h 第 28 行定义.

3.17.2 枚举类型说明

3.17.2.1 EXTI_Mode_TypeDef

enum EXTI_Mode_TypeDef

EXTI触发信号定义

枚举值

EXTI_Trigger_None	不触发
EXTI_Trigger_Negedge	下降沿触发
EXTI_Trigger_Posedge	上升沿触发
EXTI_Trigger_Edge	上升沿、下降沿均触发

在文件 lks32mc09x_exti.h 第 48 行定义.

3.17.3 函数说明

3.17.3.1 EXTI_ClearIRQFlag()

```
void EXTI_ClearIRQFlag (
    EXTI_TypeDef * EXTIx,
    uint32_t EXTI_IF_x )
```

清除GPIO引脚的中断标志

参数

GPIOx	GPIO模块指针
GPIO.PinSource	GPIO引脚源

在文件 lks32mc09x_exti.c 第 115 行定义.

3.17.3.2 EXTI_DisableIRQFlag()

```
void EXTI_DisableIRQFlag (
    EXTI_TypeDef * EXTIx,
    uint32_t EXTI_IF_x )
```

失能GPIO引脚的中断标志

参数

<i>GPIOx</i>	GPIO模块指针
<i>GPIO.PinSource</i>	GPIO引脚源

失能GPIO引脚的中断标志

参数

<i>EXTIx</i>	EXTI模块指针
<i>EXTI_↔IF_x</i>	外部中断源

在文件 [lks32mc09x_exti.c](#) 第 80 行定义.

3.17.3.3 EXTI_EnableIRQFlag()

```
void EXTI_EnableIRQFlag (
    EXTI_TypeDef * EXTIx,
    uint32_t EXTI_IF_x )
```

使能GPIO引脚的中断标志

参数

<i>GPIOx</i>	GPIO模块指针
<i>GPIO.PinSource</i>	GPIO引脚源

使能GPIO引脚的中断标志

参数

<i>EXTIx</i>	EXTI模块指针
<i>EXTI_↔IF_x</i>	外部中断源

在文件 [lks32mc09x_exti.c](#) 第 45 行定义.

3.17.3.4 EXTI_GetIRQFlag()

```
uint8_t EXTI_GetIRQFlag (
    EXTI_TypeDef * EXTIx,
    uint32_t EXTI_IF_x )
```

获取GPIO引脚的中断标志状态

参数

<i>GPIOx</i>	GPIO模块指针
<i>GPIO.PinSource</i>	GPIO引脚源

返回
中断标志状态

返回值

0	无中断
1	有中断

在文件 [lks32mc09x_exti.c](#) 第 130 行定义.

3.17.3.5 EXTI_Reset()

```
void EXTI_Reset (
    EXTI_TypeDef * EXTIx )
```

复位EXTI模块

参数

<i>EXTIx</i>	EXTI模块指针

在文件 [lks32mc09x_exti.c](#) 第 8 行定义.

3.17.3.6 EXTI_SetClko()

```
void EXTI_SetClko (
    EXTI_TypeDef * EXTIx,
    uint32_t EXTI_CLKO_x )
```

输出时钟选择

参数

<i>EXTIx</i>	EXTI模块指针
 南京凌鸥创芯电子有限公司 LINKO SEMICONDUCTOR CO.,LTD.	

在文件 [lks32mc09x_exti.c](#) 第 141 行定义.

3.17.3.7 EXTI_TriggerConfig()

```
void EXTI_TriggerConfig (
    EXTI_TypeDef * EXTIx,
    uint8_t EXTI_CR_x,
    EXTI_Mode_TypeDef EXTI_Trigger )
```

配置GPIO引脚的中断触发模式

参数

<i>EXTIx</i>	EXTI模块指针
<i>GPIO_PinSource</i>	GPIO引脚源
<i>EXTI_Trigger</i>	触发模式
<i>GPIOx</i>	GPIO模块指针
<i>GPIO_PinSource</i>	GPIO引脚源
<i>EXTI_Trigger</i>	触发模式

在文件 [lks32mc09x_exti.c](#) 第 24 行定义.

3.18 lks32mc09x_exti.h

[浏览该文件的文档.](#)

```
00001 #ifndef __LKS32MC09X_EXTI_H__
00002 #define __LKS32MC09X_EXTI_H__
00003
00004 #include "lks32mc09x_reg.h"
00005 #define EXTI_CR_P0_0 0 // GPIO 0[0]/P0[0]外部触发类型选择
00006 #define EXTI_CR_P0_2 1 // GPIO 0[2]/P0[2]外部触发类型选择
00007 #define EXTI_CR_P0_11 2 // GPIO 0[11]/P0[11]外部触发类型选择
00008 #define EXTI_CR_P0_12 3 // GPIO 0[12]/P0[12]外部触发类型选择
00009 #define EXTI_CR_P0_13 4 // GPIO 0[13]/P0[13]外部触发类型选择
00010 #define EXTI_CR_P0_14 5 // GPIO 0[14]/P0[14]外部触发类型选择
00011 #define EXTI_CR_P0_15 6 // GPIO 0[15]/P0[15]外部触发类型选择
00012 #define EXTI_CR_P1_0 7 // GPIO 1[0]/P1[0]外部触发类型选择
00013 #define EXTI_CR_P2_0 8 // GPIO 2[0]/P2[0]外部触发类型选择
00014 #define EXTI_CR_P2_3 9 // GPIO 2[3]/P2[3]外部触发类型选择
00015 #define EXTI_CR_P2_4 10 // GPIO 2[4]/P2[4]外部触发类型选择
00016 #define EXTI_CR_P2_5 11 // GPIO 2[5]/P2[5]外部触发类型选择
00017 #define EXTI_CR_P2_6 12 // GPIO 2[6]/P2[6]外部触发类型选择
00018 #define EXTI_CR_P2_7 13 // GPIO 2[7]/P2[7]外部触发类型选择
00019 #define EXTI_CR_P2_9 14 // GPIO 2[9]/P2[9]外部触发类型选择
00020 #define EXTI_CR_P2_13 15 // GPIO 2[13]/P2[13]外部触发类型选择
00021
00022 #define EXTI_IF_P2_0 BIT15 // 外部中断标志. 中断标志高有效, 写 1 清零
00023 #define EXTI_IF_P2_13 BIT14 // 外部中断标志. 中断标志高有效, 写 1 清零
00024 #define EXTI_IF_P2_6 BIT13 // 外部中断标志. 中断标志高有效, 写 1 清零
00025 #define EXTI_IF_P2_5 BIT12 // 外部中断标志. 中断标志高有效, 写 1 清零
00026 #define EXTI_IF_P2_4 BIT11 // 外部中断标志. 中断标志高有效, 写 1 清零
00027 #define EXTI_IF_P2_3 BIT10 // 外部中断标志. 中断标志高有效, 写 1 清零
00028 #define EXTI_IF_P2_9 BIT9 // 外部中断标志. 中断标志高有效, 写 1 清零
00029 #define EXTI_IF_P2_7 BIT8 // 外部中断标志. 中断标志高有效, 写 1 清零
00030 #define EXTI_IF_P1_0 BIT7 // 外部中断标志. 中断标志高有效, 写 1 清零
00031 #define EXTI_IF_P0_15 BIT6 // 外部中断标志. 中断标志高有效, 写 1 清零
00032 #define EXTI_IF_P0_14 BIT5 // 外部中断标志. 中断标志高有效, 写 1 清零
00033 #define EXTI_IF_P0_13 BIT4 // 外部中断标志. 中断标志高有效, 写 1 清零
00034 #define EXTI_IF_P0_12 BIT3 // 外部中断标志. 中断标志高有效, 写 1 清零
00035 #define EXTI_IF_P0_11 BIT2 // 外部中断标志. 中断标志高有效, 写 1 清零
00036 #define EXTI_IF_P0_2 BIT1 // 外部中断标志. 中断标志高有效, 写 1 清零
00037 #define EXTI_IF_P0_0 BIT0 // 外部中断标志. 中断标志高有效, 写 1 清零
00038
```

```

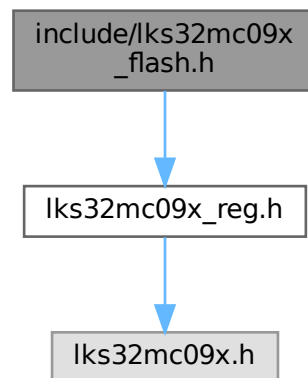
00039 #define EXTI_CLKO_HSE BIT4 // 晶振时钟输出
00040 #define EXTI_CLKO_ADC BIT3 // ADC 时钟输出
00041 #define EXTI_CLKO_PLL BIT2 // PLL 输出 注意: 由于GPIO本身输出能力的限制, 选择PLL输出到gpio上时, 用示波器只能
    在gpio上看到频率为96Mhz的一个微弱的信号
00042 #define EXTI_CLKO_HSI BIT1 // HRC 输出
00043 #define EXTI_CLKO_LSI BIT0 // LRC 输出
00044 #define EXTI_CLKO_0 0 // 不输出任何时钟
00048 typedef enum
00049 {
00050     EXTI_Trigger_None = 0x00,
00051     EXTI_Trigger_Negedge = 0x01,
00052     EXTI_Trigger_Posedge = 0x02,
00053     EXTI_Trigger_Edge = 0x03
00054 } EXTI_Mode_TypeDef;
00055
00061 void EXTI_Reset(EXTI_TypeDef *EXTIx);
00062
00070 void EXTI_TriggerConfig(EXTI_TypeDef *EXTIx, uint8_t EXTI_CR_x, EXTI_Mode_TypeDef EXTI_Trigger);
00071
00078 void EXTI_EnableIRQFlag(EXTI_TypeDef *EXTIx, uint32_t EXTI_IF_x);
00079
00086 void EXTI_DisableIRQFlag(EXTI_TypeDef *EXTIx, uint32_t EXTI_IF_x);
00087
00094 void EXTI_ClearIRQFlag(EXTI_TypeDef *EXTIx, uint32_t EXTI_IF_x);
00095
00105 uint8_t EXTI_GetIRQFlag(EXTI_TypeDef *EXTIx, uint32_t EXTI_IF_x);
00106
00113 void EXTI_SetClko(EXTI_TypeDef *EXTIx, uint32_t EXTI_CLKO_x);
00114 #endif

```

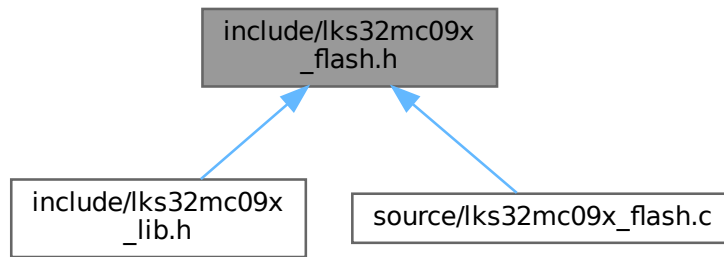
3.19 include/lks32mc09x_flash.h 文件参考

#include "lks32mc09x_reg.h"

lks32mc09x_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：



宏定义

- #define FLASH_SECTOR(x) (x * 0x200)
- #define FLASH_SECTOR_0 FLASH_SECTOR(0)
- #define FLASH_SECTOR_1 FLASH_SECTOR(1)
- #define FLASH_SECTOR_2 FLASH_SECTOR(2)
- #define FLASH_SECTOR_3 FLASH_SECTOR(3)
- #define FLASH_SECTOR_4 FLASH_SECTOR(4)
- #define FLASH_SECTOR_5 FLASH_SECTOR(5)
- #define FLASH_SECTOR_6 FLASH_SECTOR(6)
- #define FLASH_SECTOR_7 FLASH_SECTOR(7)
- #define FLASH_SECTOR_8 FLASH_SECTOR(8)
- #define FLASH_SECTOR_9 FLASH_SECTOR(9)
- #define FLASH_SECTOR_10 FLASH_SECTOR(10)
- #define FLASH_SECTOR_11 FLASH_SECTOR(11)
- #define FLASH_SECTOR_12 FLASH_SECTOR(12)
- #define FLASH_SECTOR_13 FLASH_SECTOR(13)
- #define FLASH_SECTOR_14 FLASH_SECTOR(14)
- #define FLASH_SECTOR_15 FLASH_SECTOR(15)
- #define FLASH_SECTOR_16 FLASH_SECTOR(16)
- #define FLASH_SECTOR_17 FLASH_SECTOR(17)
- #define FLASH_SECTOR_18 FLASH_SECTOR(18)
- #define FLASH_SECTOR_19 FLASH_SECTOR(19)
- #define FLASH_SECTOR_20 FLASH_SECTOR(20)
- #define FLASH_SECTOR_21 FLASH_SECTOR(21)
- #define FLASH_SECTOR_22 FLASH_SECTOR(22)
- #define FLASH_SECTOR_23 FLASH_SECTOR(23)
- #define FLASH_SECTOR_24 FLASH_SECTOR(24)
- #define FLASH_SECTOR_25 FLASH_SECTOR(25)
- #define FLASH_SECTOR_26 FLASH_SECTOR(26)
- #define FLASH_SECTOR_27 FLASH_SECTOR(27)
- #define FLASH_SECTOR_28 FLASH_SECTOR(28)
- #define FLASH_SECTOR_29 FLASH_SECTOR(29)
- #define FLASH_SECTOR_30 FLASH_SECTOR(30)
- #define FLASH_SECTOR_31 FLASH_SECTOR(31)
- #define FLASH_SECTOR_32 FLASH_SECTOR(32)

- #define FLASH_SECTOR_33 FLASH_SECTOR(33)
- #define FLASH_SECTOR_34 FLASH_SECTOR(34)
- #define FLASH_SECTOR_35 FLASH_SECTOR(35)
- #define FLASH_SECTOR_36 FLASH_SECTOR(36)
- #define FLASH_SECTOR_37 FLASH_SECTOR(37)
- #define FLASH_SECTOR_38 FLASH_SECTOR(38)
- #define FLASH_SECTOR_39 FLASH_SECTOR(39)
- #define FLASH_SECTOR_40 FLASH_SECTOR(40)
- #define FLASH_SECTOR_41 FLASH_SECTOR(41)
- #define FLASH_SECTOR_42 FLASH_SECTOR(42)
- #define FLASH_SECTOR_43 FLASH_SECTOR(43)
- #define FLASH_SECTOR_44 FLASH_SECTOR(44)
- #define FLASH_SECTOR_45 FLASH_SECTOR(45)
- #define FLASH_SECTOR_46 FLASH_SECTOR(46)
- #define FLASH_SECTOR_47 FLASH_SECTOR(47)
- #define FLASH_SECTOR_48 FLASH_SECTOR(48)
- #define FLASH_SECTOR_49 FLASH_SECTOR(49)
- #define FLASH_SECTOR_50 FLASH_SECTOR(50)
- #define FLASH_SECTOR_51 FLASH_SECTOR(51)
- #define FLASH_SECTOR_52 FLASH_SECTOR(52)
- #define FLASH_SECTOR_53 FLASH_SECTOR(53)
- #define FLASH_SECTOR_54 FLASH_SECTOR(54)
- #define FLASH_SECTOR_55 FLASH_SECTOR(55)
- #define FLASH_SECTOR_56 FLASH_SECTOR(56)
- #define FLASH_SECTOR_57 FLASH_SECTOR(57)
- #define FLASH_SECTOR_58 FLASH_SECTOR(58)
- #define FLASH_SECTOR_59 FLASH_SECTOR(59)
- #define FLASH_SECTOR_60 FLASH_SECTOR(60)
- #define FLASH_SECTOR_61 FLASH_SECTOR(61)
- #define FLASH_SECTOR_62 FLASH_SECTOR(62)
- #define FLASH_SECTOR_63 FLASH_SECTOR(63)
- #define FLASH_SECTOR_64 FLASH_SECTOR(64)
- #define FLASH_SECTOR_65 FLASH_SECTOR(65)
- #define FLASH_SECTOR_66 FLASH_SECTOR(66)
- #define FLASH_SECTOR_67 FLASH_SECTOR(67)
- #define FLASH_SECTOR_68 FLASH_SECTOR(68)
- #define FLASH_SECTOR_69 FLASH_SECTOR(69)
- #define FLASH_SECTOR_70 FLASH_SECTOR(70)
- #define FLASH_SECTOR_71 FLASH_SECTOR(71)
- #define FLASH_SECTOR_72 FLASH_SECTOR(72)
- #define FLASH_SECTOR_73 FLASH_SECTOR(73)
- #define FLASH_SECTOR_74 FLASH_SECTOR(74)
- #define FLASH_SECTOR_75 FLASH_SECTOR(75)
- #define FLASH_SECTOR_76 FLASH_SECTOR(76)
- #define FLASH_SECTOR_77 FLASH_SECTOR(77)
- #define FLASH_SECTOR_78 FLASH_SECTOR(78)
- #define FLASH_SECTOR_79 FLASH_SECTOR(79)
- #define FLASH_SECTOR_80 FLASH_SECTOR(80)
- #define FLASH_SECTOR_81 FLASH_SECTOR(81)
- #define FLASH_SECTOR_82 FLASH_SECTOR(82)
- #define FLASH_SECTOR_83 FLASH_SECTOR(83)
- #define FLASH_SECTOR_84 FLASH_SECTOR(84)
- #define FLASH_SECTOR_85 FLASH_SECTOR(85)
- #define FLASH_SECTOR_86 FLASH_SECTOR(86)
- #define FLASH_SECTOR_87 FLASH_SECTOR(87)

- #define FLASH_SECTOR_88 FLASH_SECTOR(88)
- #define FLASH_SECTOR_89 FLASH_SECTOR(89)
- #define FLASH_SECTOR_90 FLASH_SECTOR(90)
- #define FLASH_SECTOR_91 FLASH_SECTOR(91)
- #define FLASH_SECTOR_92 FLASH_SECTOR(92)
- #define FLASH_SECTOR_93 FLASH_SECTOR(93)
- #define FLASH_SECTOR_94 FLASH_SECTOR(94)
- #define FLASH_SECTOR_95 FLASH_SECTOR(95)
- #define FLASH_SECTOR_96 FLASH_SECTOR(96)
- #define FLASH_SECTOR_97 FLASH_SECTOR(97)
- #define FLASH_SECTOR_98 FLASH_SECTOR(98)
- #define FLASH_SECTOR_99 FLASH_SECTOR(99)
- #define FLASH_SECTOR_100 FLASH_SECTOR(100)
- #define FLASH_SECTOR_101 FLASH_SECTOR(101)
- #define FLASH_SECTOR_102 FLASH_SECTOR(102)
- #define FLASH_SECTOR_103 FLASH_SECTOR(103)
- #define FLASH_SECTOR_104 FLASH_SECTOR(104)
- #define FLASH_SECTOR_105 FLASH_SECTOR(105)
- #define FLASH_SECTOR_106 FLASH_SECTOR(106)
- #define FLASH_SECTOR_107 FLASH_SECTOR(107)
- #define FLASH_SECTOR_108 FLASH_SECTOR(108)
- #define FLASH_SECTOR_109 FLASH_SECTOR(109)
- #define FLASH_SECTOR_110 FLASH_SECTOR(110)
- #define FLASH_SECTOR_111 FLASH_SECTOR(111)
- #define FLASH_SECTOR_112 FLASH_SECTOR(112)
- #define FLASH_SECTOR_113 FLASH_SECTOR(113)
- #define FLASH_SECTOR_114 FLASH_SECTOR(114)
- #define FLASH_SECTOR_115 FLASH_SECTOR(115)
- #define FLASH_SECTOR_116 FLASH_SECTOR(116)
- #define FLASH_SECTOR_117 FLASH_SECTOR(117)
- #define FLASH_SECTOR_118 FLASH_SECTOR(118)
- #define FLASH_SECTOR_119 FLASH_SECTOR(119)
- #define FLASH_SECTOR_120 FLASH_SECTOR(120)
- #define FLASH_SECTOR_121 FLASH_SECTOR(121)
- #define FLASH_SECTOR_122 FLASH_SECTOR(122)
- #define FLASH_SECTOR_123 FLASH_SECTOR(123)
- #define FLASH_SECTOR_124 FLASH_SECTOR(124)
- #define FLASH_SECTOR_125 FLASH_SECTOR(125)
- #define FLASH_SECTOR_126 FLASH_SECTOR(126)
- #define FLASH_SECTOR_127 FLASH_SECTOR(127)
- #define FLASH_SECTOR_128 FLASH_SECTOR(128)
- #define FLASH_SECTOR_129 FLASH_SECTOR(129)
- #define FLASH_SECTOR_130 FLASH_SECTOR(130)
- #define FLASH_SECTOR_131 FLASH_SECTOR(131)
- #define FLASH_SECTOR_132 FLASH_SECTOR(132)
- #define FLASH_SECTOR_133 FLASH_SECTOR(133)
- #define FLASH_SECTOR_134 FLASH_SECTOR(134)
- #define FLASH_SECTOR_135 FLASH_SECTOR(135)
- #define FLASH_SECTOR_136 FLASH_SECTOR(136)
- #define FLASH_SECTOR_137 FLASH_SECTOR(137)
- #define FLASH_SECTOR_138 FLASH_SECTOR(138)
- #define FLASH_SECTOR_139 FLASH_SECTOR(139)
- #define FLASH_SECTOR_140 FLASH_SECTOR(140)
- #define FLASH_SECTOR_141 FLASH_SECTOR(141)
- #define FLASH_SECTOR_142 FLASH_SECTOR(142)

- #define FLASH_SECTOR_143 FLASH_SECTOR(143)
- #define FLASH_SECTOR_144 FLASH_SECTOR(144)
- #define FLASH_SECTOR_145 FLASH_SECTOR(145)
- #define FLASH_SECTOR_146 FLASH_SECTOR(146)
- #define FLASH_SECTOR_147 FLASH_SECTOR(147)
- #define FLASH_SECTOR_148 FLASH_SECTOR(148)
- #define FLASH_SECTOR_149 FLASH_SECTOR(149)
- #define FLASH_SECTOR_150 FLASH_SECTOR(150)
- #define FLASH_SECTOR_151 FLASH_SECTOR(151)
- #define FLASH_SECTOR_152 FLASH_SECTOR(152)
- #define FLASH_SECTOR_153 FLASH_SECTOR(153)
- #define FLASH_SECTOR_154 FLASH_SECTOR(154)
- #define FLASH_SECTOR_155 FLASH_SECTOR(155)
- #define FLASH_SECTOR_156 FLASH_SECTOR(156)
- #define FLASH_SECTOR_157 FLASH_SECTOR(157)
- #define FLASH_SECTOR_158 FLASH_SECTOR(158)
- #define FLASH_SECTOR_159 FLASH_SECTOR(159)
- #define FLASH_SECTOR_160 FLASH_SECTOR(160)
- #define FLASH_SECTOR_161 FLASH_SECTOR(161)
- #define FLASH_SECTOR_162 FLASH_SECTOR(162)
- #define FLASH_SECTOR_163 FLASH_SECTOR(163)
- #define FLASH_SECTOR_164 FLASH_SECTOR(164)
- #define FLASH_SECTOR_165 FLASH_SECTOR(165)
- #define FLASH_SECTOR_166 FLASH_SECTOR(166)
- #define FLASH_SECTOR_167 FLASH_SECTOR(167)
- #define FLASH_SECTOR_168 FLASH_SECTOR(168)
- #define FLASH_SECTOR_169 FLASH_SECTOR(169)
- #define FLASH_SECTOR_170 FLASH_SECTOR(170)
- #define FLASH_SECTOR_171 FLASH_SECTOR(171)
- #define FLASH_SECTOR_172 FLASH_SECTOR(172)
- #define FLASH_SECTOR_173 FLASH_SECTOR(173)
- #define FLASH_SECTOR_174 FLASH_SECTOR(174)
- #define FLASH_SECTOR_175 FLASH_SECTOR(175)
- #define FLASH_SECTOR_176 FLASH_SECTOR(176)
- #define FLASH_SECTOR_177 FLASH_SECTOR(177)
- #define FLASH_SECTOR_178 FLASH_SECTOR(178)
- #define FLASH_SECTOR_179 FLASH_SECTOR(179)
- #define FLASH_SECTOR_180 FLASH_SECTOR(180)
- #define FLASH_SECTOR_181 FLASH_SECTOR(181)
- #define FLASH_SECTOR_182 FLASH_SECTOR(182)
- #define FLASH_SECTOR_183 FLASH_SECTOR(183)
- #define FLASH_SECTOR_184 FLASH_SECTOR(184)
- #define FLASH_SECTOR_185 FLASH_SECTOR(185)
- #define FLASH_SECTOR_186 FLASH_SECTOR(186)
- #define FLASH_SECTOR_187 FLASH_SECTOR(187)
- #define FLASH_SECTOR_188 FLASH_SECTOR(188)
- #define FLASH_SECTOR_189 FLASH_SECTOR(189)
- #define FLASH_SECTOR_190 FLASH_SECTOR(190)
- #define FLASH_SECTOR_191 FLASH_SECTOR(191)
- #define FLASH_SECTOR_192 FLASH_SECTOR(192)
- #define FLASH_SECTOR_193 FLASH_SECTOR(193)
- #define FLASH_SECTOR_194 FLASH_SECTOR(194)
- #define FLASH_SECTOR_195 FLASH_SECTOR(195)
- #define FLASH_SECTOR_196 FLASH_SECTOR(196)
- #define FLASH_SECTOR_197 FLASH_SECTOR(197)

- #define FLASH_SECTOR_198 FLASH_SECTOR(198)
- #define FLASH_SECTOR_199 FLASH_SECTOR(199)
- #define FLASH_SECTOR_200 FLASH_SECTOR(200)
- #define FLASH_SECTOR_201 FLASH_SECTOR(201)
- #define FLASH_SECTOR_202 FLASH_SECTOR(202)
- #define FLASH_SECTOR_203 FLASH_SECTOR(203)
- #define FLASH_SECTOR_204 FLASH_SECTOR(204)
- #define FLASH_SECTOR_205 FLASH_SECTOR(205)
- #define FLASH_SECTOR_206 FLASH_SECTOR(206)
- #define FLASH_SECTOR_207 FLASH_SECTOR(207)
- #define FLASH_SECTOR_208 FLASH_SECTOR(208)
- #define FLASH_SECTOR_209 FLASH_SECTOR(209)
- #define FLASH_SECTOR_210 FLASH_SECTOR(210)
- #define FLASH_SECTOR_211 FLASH_SECTOR(211)
- #define FLASH_SECTOR_212 FLASH_SECTOR(212)
- #define FLASH_SECTOR_213 FLASH_SECTOR(213)
- #define FLASH_SECTOR_214 FLASH_SECTOR(214)
- #define FLASH_SECTOR_215 FLASH_SECTOR(215)
- #define FLASH_SECTOR_216 FLASH_SECTOR(216)
- #define FLASH_SECTOR_217 FLASH_SECTOR(217)
- #define FLASH_SECTOR_218 FLASH_SECTOR(218)
- #define FLASH_SECTOR_219 FLASH_SECTOR(219)
- #define FLASH_SECTOR_220 FLASH_SECTOR(220)
- #define FLASH_SECTOR_221 FLASH_SECTOR(221)
- #define FLASH_SECTOR_222 FLASH_SECTOR(222)
- #define FLASH_SECTOR_223 FLASH_SECTOR(223)
- #define FLASH_SECTOR_224 FLASH_SECTOR(224)
- #define FLASH_SECTOR_225 FLASH_SECTOR(225)
- #define FLASH_SECTOR_226 FLASH_SECTOR(226)
- #define FLASH_SECTOR_227 FLASH_SECTOR(227)
- #define FLASH_SECTOR_228 FLASH_SECTOR(228)
- #define FLASH_SECTOR_229 FLASH_SECTOR(229)
- #define FLASH_SECTOR_230 FLASH_SECTOR(230)
- #define FLASH_SECTOR_231 FLASH_SECTOR(231)
- #define FLASH_SECTOR_232 FLASH_SECTOR(232)
- #define FLASH_SECTOR_233 FLASH_SECTOR(233)
- #define FLASH_SECTOR_234 FLASH_SECTOR(234)
- #define FLASH_SECTOR_235 FLASH_SECTOR(235)
- #define FLASH_SECTOR_236 FLASH_SECTOR(236)
- #define FLASH_SECTOR_237 FLASH_SECTOR(237)
- #define FLASH_SECTOR_238 FLASH_SECTOR(238)
- #define FLASH_SECTOR_239 FLASH_SECTOR(239)
- #define FLASH_SECTOR_240 FLASH_SECTOR(240)
- #define FLASH_SECTOR_241 FLASH_SECTOR(241)
- #define FLASH_SECTOR_242 FLASH_SECTOR(242)
- #define FLASH_SECTOR_243 FLASH_SECTOR(243)
- #define FLASH_SECTOR_244 FLASH_SECTOR(244)
- #define FLASH_SECTOR_245 FLASH_SECTOR(245)
- #define FLASH_SECTOR_246 FLASH_SECTOR(246)
- #define FLASH_SECTOR_247 FLASH_SECTOR(247)
- #define FLASH_SECTOR_248 FLASH_SECTOR(248)
- #define FLASH_SECTOR_249 FLASH_SECTOR(249)
- #define FLASH_SECTOR_250 FLASH_SECTOR(250)
- #define FLASH_SECTOR_251 FLASH_SECTOR(251)
- #define FLASH_SECTOR_252 FLASH_SECTOR(252)

- `#define FLASH_SECTOR_253 FLASH_SECTOR(253)`
- `#define FLASH_SECTOR_254 FLASH_SECTOR(254)`
- `#define FLASH_SECTOR_255 FLASH_SECTOR(255)`
- `#define FLASH_SECTOR_END_128K FLASH_SECTOR_255`
- `#define FLASH_SECTOR_END_64K FLASH_SECTOR_127`
- `#define FLASH_SECTOR_END_32K FLASH_SECTOR_63`

函数

- `void FLASH_Write (uint32_t adr, uint8_t *data, uint32_t len)`
向指定地址写入数据
- `void FLASH_Read (uint32_t adr, uint8_t *data, uint32_t len)`
从指定地址读取数据
- `void FLASH_Erase (uint32_t adr)`
擦除指定扇区
- `void FLASH_WriteNvr (uint32_t adr, uint8_t *data, uint32_t len)`
向指定地址写入数据(NVR区域)
- `void FLASH_EraseNvr (uint32_t adr)`
擦除指定扇区(NVR区域)
- `void FLASH_ReadNvr (uint32_t adr, uint8_t *data, uint32_t len)`
从指定地址读取数据(NVR区域)

3.19.1 宏定义说明

3.19.1.1 FLASH_SECTOR

```
#define FLASH_SECTOR(  
    x ) (x * 0x200)
```

在文件 [lks32mc09x_flash.h](#) 第 6 行定义。

3.19.1.2 FLASH_SECTOR_0

```
#define FLASH_SECTOR_0 FLASH_SECTOR(0)
```

在文件 [lks32mc09x_flash.h](#) 第 7 行定义。

3.19.1.3 FLASH_SECTOR_1

```
#define FLASH_SECTOR_1 FLASH_SECTOR(1)
```

在文件 [lks32mc09x_flash.h](#) 第 8 行定义。

3.19.1.4 FLASH_SECTOR_10

```
#define FLASH_SECTOR_10 FLASH_SECTOR(10)
```

在文件 [lks32mc09x_flash.h](#) 第 17 行定义。

3.19.1.5 FLASH_SECTOR_100

```
#define FLASH_SECTOR_100 FLASH_SECTOR(100)
```

在文件 [lks32mc09x_flash.h](#) 第 107 行定义.

3.19.1.6 FLASH_SECTOR_101

```
#define FLASH_SECTOR_101 FLASH_SECTOR(101)
```

在文件 [lks32mc09x_flash.h](#) 第 108 行定义.

3.19.1.7 FLASH_SECTOR_102

```
#define FLASH_SECTOR_102 FLASH_SECTOR(102)
```

在文件 [lks32mc09x_flash.h](#) 第 109 行定义.

3.19.1.8 FLASH_SECTOR_103

```
#define FLASH_SECTOR_103 FLASH_SECTOR(103)
```

在文件 [lks32mc09x_flash.h](#) 第 110 行定义.

3.19.1.9 FLASH_SECTOR_104

```
#define FLASH_SECTOR_104 FLASH_SECTOR(104)
```

在文件 [lks32mc09x_flash.h](#) 第 111 行定义.

3.19.1.10 FLASH_SECTOR_105

```
#define FLASH_SECTOR_105 FLASH_SECTOR(105)
```

在文件 [lks32mc09x_flash.h](#) 第 112 行定义.

3.19.1.11 FLASH_SECTOR_106

```
#define FLASH_SECTOR_106 FLASH_SECTOR(106)
```

在文件 [lks32mc09x_flash.h](#) 第 113 行定义.

3.19.1.12 FLASH_SECTOR_107

```
#define FLASH_SECTOR_107 FLASH_SECTOR(107)
```

在文件 [lks32mc09x_flash.h](#) 第 114 行定义.

3.19.1.13 FLASH_SECTOR_108

```
#define FLASH_SECTOR_108 FLASH_SECTOR(108)
```

在文件 [lks32mc09x_flash.h](#) 第 115 行定义.

3.19.1.14 FLASH_SECTOR_109

```
#define FLASH_SECTOR_109 FLASH_SECTOR(109)
```

在文件 [lks32mc09x_flash.h](#) 第 116 行定义.

3.19.1.15 FLASH_SECTOR_11

```
#define FLASH_SECTOR_11 FLASH_SECTOR(11)
```

在文件 [lks32mc09x_flash.h](#) 第 18 行定义.

3.19.1.16 FLASH_SECTOR_110

```
#define FLASH_SECTOR_110 FLASH_SECTOR(110)
```

在文件 [lks32mc09x_flash.h](#) 第 117 行定义.

3.19.1.17 FLASH_SECTOR_111

```
#define FLASH_SECTOR_111 FLASH_SECTOR(111)
```

在文件 [lks32mc09x_flash.h](#) 第 118 行定义.

3.19.1.18 FLASH_SECTOR_112

```
#define FLASH_SECTOR_112 FLASH_SECTOR(112)
```

在文件 [lks32mc09x_flash.h](#) 第 119 行定义.

3.19.1.19 FLASH_SECTOR_113

```
#define FLASH_SECTOR_113 FLASH_SECTOR(113)
```

在文件 [lks32mc09x_flash.h](#) 第 120 行定义.

3.19.1.20 FLASH_SECTOR_114

```
#define FLASH_SECTOR_114 FLASH_SECTOR(114)
```

在文件 [lks32mc09x_flash.h](#) 第 121 行定义.

3.19.1.21 FLASH_SECTOR_115

```
#define FLASH_SECTOR_115 FLASH_SECTOR(115)
```

在文件 [lks32mc09x_flash.h](#) 第 122 行定义.

3.19.1.22 FLASH_SECTOR_116

```
#define FLASH_SECTOR_116 FLASH_SECTOR(116)
```

在文件 [lks32mc09x_flash.h](#) 第 123 行定义.

3.19.1.23 FLASH_SECTOR_117

```
#define FLASH_SECTOR_117 FLASH_SECTOR(117)
```

在文件 [lks32mc09x_flash.h](#) 第 124 行定义.

3.19.1.24 FLASH_SECTOR_118

```
#define FLASH_SECTOR_118 FLASH_SECTOR(118)
```

在文件 [lks32mc09x_flash.h](#) 第 125 行定义.

3.19.1.25 FLASH_SECTOR_119

```
#define FLASH_SECTOR_119 FLASH_SECTOR(119)
```

在文件 [lks32mc09x_flash.h](#) 第 126 行定义.

3.19.1.26 FLASH_SECTOR_12

```
#define FLASH_SECTOR_12 FLASH_SECTOR(12)
```

在文件 [lks32mc09x_flash.h](#) 第 19 行定义.

3.19.1.27 FLASH_SECTOR_120

```
#define FLASH_SECTOR_120 FLASH_SECTOR(120)
```

在文件 [lks32mc09x_flash.h](#) 第 127 行定义.

3.19.1.28 FLASH_SECTOR_121

```
#define FLASH_SECTOR_121 FLASH_SECTOR(121)
```

在文件 [lks32mc09x_flash.h](#) 第 128 行定义.

3.19.1.29 FLASH_SECTOR_122

```
#define FLASH_SECTOR_122 FLASH_SECTOR(122)
```

在文件 [lks32mc09x_flash.h](#) 第 129 行定义.

3.19.1.30 FLASH_SECTOR_123

```
#define FLASH_SECTOR_123 FLASH_SECTOR(123)
```

在文件 [lks32mc09x_flash.h](#) 第 130 行定义.

3.19.1.31 FLASH_SECTOR_124

```
#define FLASH_SECTOR_124 FLASH_SECTOR(124)
```

在文件 [lks32mc09x_flash.h](#) 第 131 行定义.

3.19.1.32 FLASH_SECTOR_125

```
#define FLASH_SECTOR_125 FLASH_SECTOR(125)
```

在文件 [lks32mc09x_flash.h](#) 第 132 行定义.

3.19.1.33 FLASH_SECTOR_126

```
#define FLASH_SECTOR_126 FLASH_SECTOR(126)
```

在文件 [lks32mc09x_flash.h](#) 第 133 行定义.

3.19.1.34 FLASH_SECTOR_127

```
#define FLASH_SECTOR_127 FLASH_SECTOR(127)
```

在文件 [lks32mc09x_flash.h](#) 第 134 行定义.

3.19.1.35 FLASH_SECTOR_128

```
#define FLASH_SECTOR_128 FLASH_SECTOR(128)
```

在文件 [lks32mc09x_flash.h](#) 第 135 行定义.

3.19.1.36 FLASH_SECTOR_129

```
#define FLASH_SECTOR_129 FLASH_SECTOR(129)
```

在文件 [lks32mc09x_flash.h](#) 第 136 行定义.

3.19.1.37 FLASH_SECTOR_13

```
#define FLASH_SECTOR_13 FLASH_SECTOR(13)
```

在文件 [lks32mc09x_flash.h](#) 第 20 行定义.

3.19.1.38 FLASH_SECTOR_130

```
#define FLASH_SECTOR_130 FLASH_SECTOR(130)
```

在文件 [lks32mc09x_flash.h](#) 第 137 行定义.

3.19.1.39 FLASH_SECTOR_131

```
#define FLASH_SECTOR_131 FLASH_SECTOR(131)
```

在文件 [lks32mc09x_flash.h](#) 第 138 行定义.

3.19.1.40 FLASH_SECTOR_132

```
#define FLASH_SECTOR_132 FLASH_SECTOR(132)
```

在文件 [lks32mc09x_flash.h](#) 第 139 行定义.

3.19.1.41 FLASH_SECTOR_133

```
#define FLASH_SECTOR_133 FLASH_SECTOR(133)
```

在文件 [lks32mc09x_flash.h](#) 第 140 行定义.

3.19.1.42 FLASH_SECTOR_134

```
#define FLASH_SECTOR_134 FLASH_SECTOR(134)
```

在文件 [lks32mc09x_flash.h](#) 第 141 行定义.

3.19.1.43 FLASH_SECTOR_135

```
#define FLASH_SECTOR_135 FLASH_SECTOR(135)
```

在文件 [lks32mc09x_flash.h](#) 第 142 行定义.

3.19.1.44 FLASH_SECTOR_136

```
#define FLASH_SECTOR_136 FLASH_SECTOR(136)
```

在文件 [lks32mc09x_flash.h](#) 第 143 行定义.

3.19.1.45 FLASH_SECTOR_137

```
#define FLASH_SECTOR_137 FLASH_SECTOR(137)
```

在文件 [lks32mc09x_flash.h](#) 第 144 行定义.

3.19.1.46 FLASH_SECTOR_138

```
#define FLASH_SECTOR_138 FLASH_SECTOR(138)
```

在文件 [lks32mc09x_flash.h](#) 第 145 行定义.

3.19.1.47 FLASH_SECTOR_139

```
#define FLASH_SECTOR_139 FLASH_SECTOR(139)
```

在文件 [lks32mc09x_flash.h](#) 第 146 行定义.

3.19.1.48 FLASH_SECTOR_14

```
#define FLASH_SECTOR_14 FLASH_SECTOR(14)
```

在文件 [lks32mc09x_flash.h](#) 第 21 行定义.

3.19.1.49 FLASH_SECTOR_140

```
#define FLASH_SECTOR_140 FLASH_SECTOR(140)
```

在文件 [lks32mc09x_flash.h](#) 第 147 行定义.

3.19.1.50 FLASH_SECTOR_141

```
#define FLASH_SECTOR_141 FLASH_SECTOR(141)
```

在文件 [lks32mc09x_flash.h](#) 第 148 行定义.

3.19.1.51 FLASH_SECTOR_142

```
#define FLASH_SECTOR_142 FLASH_SECTOR(142)
```

在文件 [lks32mc09x_flash.h](#) 第 149 行定义.

3.19.1.52 FLASH_SECTOR_143

```
#define FLASH_SECTOR_143 FLASH_SECTOR(143)
```

在文件 [lks32mc09x_flash.h](#) 第 150 行定义.

3.19.1.53 FLASH_SECTOR_144

```
#define FLASH_SECTOR_144 FLASH_SECTOR(144)
```

在文件 [lks32mc09x_flash.h](#) 第 151 行定义.

3.19.1.54 FLASH_SECTOR_145

```
#define FLASH_SECTOR_145 FLASH_SECTOR(145)
```

在文件 [lks32mc09x_flash.h](#) 第 152 行定义.

3.19.1.55 FLASH_SECTOR_146

```
#define FLASH_SECTOR_146 FLASH_SECTOR(146)
```

在文件 [lks32mc09x_flash.h](#) 第 153 行定义.

3.19.1.56 FLASH_SECTOR_147

```
#define FLASH_SECTOR_147 FLASH_SECTOR(147)
```

在文件 [lks32mc09x_flash.h](#) 第 154 行定义.

3.19.1.57 FLASH_SECTOR_148

```
#define FLASH_SECTOR_148 FLASH_SECTOR(148)
```

在文件 [lks32mc09x_flash.h](#) 第 155 行定义.

3.19.1.58 FLASH_SECTOR_149

```
#define FLASH_SECTOR_149 FLASH_SECTOR(149)
```

在文件 [lks32mc09x_flash.h](#) 第 156 行定义.

3.19.1.59 FLASH_SECTOR_15

```
#define FLASH_SECTOR_15 FLASH_SECTOR(15)
```

在文件 [lks32mc09x_flash.h](#) 第 22 行定义.

3.19.1.60 FLASH_SECTOR_150

```
#define FLASH_SECTOR_150 FLASH_SECTOR(150)
```

在文件 [lks32mc09x_flash.h](#) 第 157 行定义.

3.19.1.61 FLASH_SECTOR_151

```
#define FLASH_SECTOR_151 FLASH_SECTOR(151)
```

在文件 [lks32mc09x_flash.h](#) 第 158 行定义.

3.19.1.62 FLASH_SECTOR_152

```
#define FLASH_SECTOR_152 FLASH_SECTOR(152)
```

在文件 [lks32mc09x_flash.h](#) 第 159 行定义.

3.19.1.63 FLASH_SECTOR_153

```
#define FLASH_SECTOR_153 FLASH_SECTOR(153)
```

在文件 [lks32mc09x_flash.h](#) 第 160 行定义.

3.19.1.64 FLASH_SECTOR_154

```
#define FLASH_SECTOR_154 FLASH_SECTOR(154)
```

在文件 [lks32mc09x_flash.h](#) 第 161 行定义.

3.19.1.65 FLASH_SECTOR_155

```
#define FLASH_SECTOR_155 FLASH_SECTOR(155)
```

在文件 [lks32mc09x_flash.h](#) 第 162 行定义.

3.19.1.66 FLASH_SECTOR_156

```
#define FLASH_SECTOR_156 FLASH_SECTOR(156)
```

在文件 [lks32mc09x_flash.h](#) 第 163 行定义.

3.19.1.67 FLASH_SECTOR_157

```
#define FLASH_SECTOR_157 FLASH_SECTOR(157)
```

在文件 [lks32mc09x_flash.h](#) 第 164 行定义.

3.19.1.68 FLASH_SECTOR_158

```
#define FLASH_SECTOR_158 FLASH_SECTOR(158)
```

在文件 [lks32mc09x_flash.h](#) 第 165 行定义.

3.19.1.69 FLASH_SECTOR_159

```
#define FLASH_SECTOR_159 FLASH_SECTOR(159)
```

在文件 [lks32mc09x_flash.h](#) 第 166 行定义.

3.19.1.70 FLASH_SECTOR_16

```
#define FLASH_SECTOR_16 FLASH_SECTOR(16)
```

在文件 [lks32mc09x_flash.h](#) 第 23 行定义.

3.19.1.71 FLASH_SECTOR_160

```
#define FLASH_SECTOR_160 FLASH_SECTOR(160)
```

在文件 [lks32mc09x_flash.h](#) 第 167 行定义.

3.19.1.72 FLASH_SECTOR_161

```
#define FLASH_SECTOR_161 FLASH_SECTOR(161)
```

在文件 [lks32mc09x_flash.h](#) 第 168 行定义.

3.19.1.73 FLASH_SECTOR_162

```
#define FLASH_SECTOR_162 FLASH_SECTOR(162)
```

在文件 [lks32mc09x_flash.h](#) 第 169 行定义.

3.19.1.74 FLASH_SECTOR_163

```
#define FLASH_SECTOR_163 FLASH_SECTOR(163)
```

在文件 [lks32mc09x_flash.h](#) 第 170 行定义.

3.19.1.75 FLASH_SECTOR_164

```
#define FLASH_SECTOR_164 FLASH_SECTOR(164)
```

在文件 [lks32mc09x_flash.h](#) 第 171 行定义.

3.19.1.76 FLASH_SECTOR_165

```
#define FLASH_SECTOR_165 FLASH_SECTOR(165)
```

在文件 [lks32mc09x_flash.h](#) 第 172 行定义.

3.19.1.77 FLASH_SECTOR_166

```
#define FLASH_SECTOR_166 FLASH_SECTOR(166)
```

在文件 [lks32mc09x_flash.h](#) 第 173 行定义.

3.19.1.78 FLASH_SECTOR_167

```
#define FLASH_SECTOR_167 FLASH_SECTOR(167)
```

在文件 [lks32mc09x_flash.h](#) 第 174 行定义.

3.19.1.79 FLASH_SECTOR_168

```
#define FLASH_SECTOR_168 FLASH_SECTOR(168)
```

在文件 [lks32mc09x_flash.h](#) 第 175 行定义.

3.19.1.80 FLASH_SECTOR_169

```
#define FLASH_SECTOR_169 FLASH_SECTOR(169)
```

在文件 [lks32mc09x_flash.h](#) 第 176 行定义.

3.19.1.81 FLASH_SECTOR_17

```
#define FLASH_SECTOR_17 FLASH_SECTOR(17)
```

在文件 [lks32mc09x_flash.h](#) 第 24 行定义.

3.19.1.82 FLASH_SECTOR_170

```
#define FLASH_SECTOR_170 FLASH_SECTOR(170)
```

在文件 [lks32mc09x_flash.h](#) 第 177 行定义.

3.19.1.83 FLASH_SECTOR_171

```
#define FLASH_SECTOR_171 FLASH_SECTOR(171)
```

在文件 [lks32mc09x_flash.h](#) 第 178 行定义.

3.19.1.84 FLASH_SECTOR_172

```
#define FLASH_SECTOR_172 FLASH_SECTOR(172)
```

在文件 [lks32mc09x_flash.h](#) 第 179 行定义.

3.19.1.85 FLASH_SECTOR_173

```
#define FLASH_SECTOR_173 FLASH_SECTOR(173)
```

在文件 [lks32mc09x_flash.h](#) 第 180 行定义.

3.19.1.86 FLASH_SECTOR_174

```
#define FLASH_SECTOR_174 FLASH_SECTOR(174)
```

在文件 [lks32mc09x_flash.h](#) 第 181 行定义.

3.19.1.87 FLASH_SECTOR_175

```
#define FLASH_SECTOR_175 FLASH_SECTOR(175)
```

在文件 [lks32mc09x_flash.h](#) 第 182 行定义.

3.19.1.88 FLASH_SECTOR_176

```
#define FLASH_SECTOR_176 FLASH_SECTOR(176)
```

在文件 [lks32mc09x_flash.h](#) 第 183 行定义.

3.19.1.89 FLASH_SECTOR_177

```
#define FLASH_SECTOR_177 FLASH_SECTOR(177)
```

在文件 [lks32mc09x_flash.h](#) 第 184 行定义.

3.19.1.90 FLASH_SECTOR_178

```
#define FLASH_SECTOR_178 FLASH_SECTOR(178)
```

在文件 [lks32mc09x_flash.h](#) 第 185 行定义.

3.19.1.91 FLASH_SECTOR_179

```
#define FLASH_SECTOR_179 FLASH_SECTOR(179)
```

在文件 [lks32mc09x_flash.h](#) 第 186 行定义.

3.19.1.92 FLASH_SECTOR_18

```
#define FLASH_SECTOR_18 FLASH_SECTOR(18)
```

在文件 [lks32mc09x_flash.h](#) 第 25 行定义.

3.19.1.93 FLASH_SECTOR_180

```
#define FLASH_SECTOR_180 FLASH_SECTOR(180)
```

在文件 [lks32mc09x_flash.h](#) 第 187 行定义.

3.19.1.94 FLASH_SECTOR_181

```
#define FLASH_SECTOR_181 FLASH_SECTOR(181)
```

在文件 [lks32mc09x_flash.h](#) 第 188 行定义.

3.19.1.95 FLASH_SECTOR_182

```
#define FLASH_SECTOR_182 FLASH_SECTOR(182)
```

在文件 [lks32mc09x_flash.h](#) 第 189 行定义.

3.19.1.96 FLASH_SECTOR_183

```
#define FLASH_SECTOR_183 FLASH_SECTOR(183)
```

在文件 [lks32mc09x_flash.h](#) 第 190 行定义.

3.19.1.97 FLASH_SECTOR_184

```
#define FLASH_SECTOR_184 FLASH_SECTOR(184)
```

在文件 [lks32mc09x_flash.h](#) 第 191 行定义.

3.19.1.98 FLASH_SECTOR_185

```
#define FLASH_SECTOR_185 FLASH_SECTOR(185)
```

在文件 [lks32mc09x_flash.h](#) 第 192 行定义.

3.19.1.99 FLASH_SECTOR_186

```
#define FLASH_SECTOR_186 FLASH_SECTOR(186)
```

在文件 [lks32mc09x_flash.h](#) 第 193 行定义.

3.19.1.100 FLASH_SECTOR_187

```
#define FLASH_SECTOR_187 FLASH_SECTOR(187)
```

在文件 [lks32mc09x_flash.h](#) 第 194 行定义.

3.19.1.101 FLASH_SECTOR_188

```
#define FLASH_SECTOR_188 FLASH_SECTOR(188)
```

在文件 [lks32mc09x_flash.h](#) 第 195 行定义.

3.19.1.102 FLASH_SECTOR_189

```
#define FLASH_SECTOR_189 FLASH_SECTOR(189)
```

在文件 [lks32mc09x_flash.h](#) 第 196 行定义.

3.19.1.103 FLASH_SECTOR_19

```
#define FLASH_SECTOR_19 FLASH_SECTOR(19)
```

在文件 [lks32mc09x_flash.h](#) 第 26 行定义.

3.19.1.104 FLASH_SECTOR_190

```
#define FLASH_SECTOR_190 FLASH_SECTOR(190)
```

在文件 [lks32mc09x_flash.h](#) 第 197 行定义.

3.19.1.105 FLASH_SECTOR_191

```
#define FLASH_SECTOR_191 FLASH_SECTOR(191)
```

在文件 [lks32mc09x_flash.h](#) 第 198 行定义.

3.19.1.106 FLASH_SECTOR_192

```
#define FLASH_SECTOR_192 FLASH_SECTOR(192)
```

在文件 [lks32mc09x_flash.h](#) 第 199 行定义.

3.19.1.107 FLASH_SECTOR_193

```
#define FLASH_SECTOR_193 FLASH_SECTOR(193)
```

在文件 [lks32mc09x_flash.h](#) 第 200 行定义.

3.19.1.108 FLASH_SECTOR_194

```
#define FLASH_SECTOR_194 FLASH_SECTOR(194)
```

在文件 [lks32mc09x_flash.h](#) 第 201 行定义.

3.19.1.109 FLASH_SECTOR_195

```
#define FLASH_SECTOR_195 FLASH_SECTOR(195)
```

在文件 [lks32mc09x_flash.h](#) 第 202 行定义.

3.19.1.110 FLASH_SECTOR_196

```
#define FLASH_SECTOR_196 FLASH_SECTOR(196)
```

在文件 [lks32mc09x_flash.h](#) 第 203 行定义.

3.19.1.111 FLASH_SECTOR_197

```
#define FLASH_SECTOR_197 FLASH_SECTOR(197)
```

在文件 [lks32mc09x_flash.h](#) 第 204 行定义.

3.19.1.112 FLASH_SECTOR_198

```
#define FLASH_SECTOR_198 FLASH_SECTOR(198)
```

在文件 [lks32mc09x_flash.h](#) 第 205 行定义.

3.19.1.113 FLASH_SECTOR_199

```
#define FLASH_SECTOR_199 FLASH_SECTOR(199)
```

在文件 [lks32mc09x_flash.h](#) 第 206 行定义.

3.19.1.114 FLASH_SECTOR_2

```
#define FLASH_SECTOR_2 FLASH_SECTOR(2)
```

在文件 [lks32mc09x_flash.h](#) 第 9 行定义.

3.19.1.115 FLASH_SECTOR_20

```
#define FLASH_SECTOR_20 FLASH_SECTOR(20)
```

在文件 [lks32mc09x_flash.h](#) 第 27 行定义.

3.19.1.116 FLASH_SECTOR_200

```
#define FLASH_SECTOR_200 FLASH_SECTOR(200)
```

在文件 [lks32mc09x_flash.h](#) 第 207 行定义.

3.19.1.117 FLASH_SECTOR_201

```
#define FLASH_SECTOR_201 FLASH_SECTOR(201)
```

在文件 [lks32mc09x_flash.h](#) 第 208 行定义.

3.19.1.118 FLASH_SECTOR_202

```
#define FLASH_SECTOR_202 FLASH_SECTOR(202)
```

在文件 [lks32mc09x_flash.h](#) 第 209 行定义.

3.19.1.119 FLASH_SECTOR_203

```
#define FLASH_SECTOR_203 FLASH_SECTOR(203)
```

在文件 [lks32mc09x_flash.h](#) 第 210 行定义.

3.19.1.120 FLASH_SECTOR_204

```
#define FLASH_SECTOR_204 FLASH_SECTOR(204)
```

在文件 [lks32mc09x_flash.h](#) 第 211 行定义.

3.19.1.121 FLASH_SECTOR_205

```
#define FLASH_SECTOR_205 FLASH_SECTOR(205)
```

在文件 [lks32mc09x_flash.h](#) 第 212 行定义.

3.19.1.122 FLASH_SECTOR_206

```
#define FLASH_SECTOR_206 FLASH_SECTOR(206)
```

在文件 [lks32mc09x_flash.h](#) 第 213 行定义.

3.19.1.123 FLASH_SECTOR_207

```
#define FLASH_SECTOR_207 FLASH_SECTOR(207)
```

在文件 [lks32mc09x_flash.h](#) 第 214 行定义.

3.19.1.124 FLASH_SECTOR_208

```
#define FLASH_SECTOR_208 FLASH_SECTOR(208)
```

在文件 [lks32mc09x_flash.h](#) 第 215 行定义.

3.19.1.125 FLASH_SECTOR_209

```
#define FLASH_SECTOR_209 FLASH_SECTOR(209)
```

在文件 [lks32mc09x_flash.h](#) 第 216 行定义.

3.19.1.126 FLASH_SECTOR_21

```
#define FLASH_SECTOR_21 FLASH_SECTOR(21)
```

在文件 [lks32mc09x_flash.h](#) 第 28 行定义.

3.19.1.127 FLASH_SECTOR_210

```
#define FLASH_SECTOR_210 FLASH_SECTOR(210)
```

在文件 [lks32mc09x_flash.h](#) 第 217 行定义.

3.19.1.128 FLASH_SECTOR_211

```
#define FLASH_SECTOR_211 FLASH_SECTOR(211)
```

在文件 [lks32mc09x_flash.h](#) 第 218 行定义.

3.19.1.129 FLASH_SECTOR_212

```
#define FLASH_SECTOR_212 FLASH_SECTOR(212)
```

在文件 [lks32mc09x_flash.h](#) 第 219 行定义.

3.19.1.130 FLASH_SECTOR_213

```
#define FLASH_SECTOR_213 FLASH_SECTOR(213)
```

在文件 [lks32mc09x_flash.h](#) 第 220 行定义.

3.19.1.131 FLASH_SECTOR_214

```
#define FLASH_SECTOR_214 FLASH_SECTOR(214)
```

在文件 [lks32mc09x_flash.h](#) 第 221 行定义.

3.19.1.132 FLASH_SECTOR_215

```
#define FLASH_SECTOR_215 FLASH_SECTOR(215)
```

在文件 [lks32mc09x_flash.h](#) 第 222 行定义.

3.19.1.133 FLASH_SECTOR_216

```
#define FLASH_SECTOR_216 FLASH_SECTOR(216)
```

在文件 [lks32mc09x_flash.h](#) 第 223 行定义.

3.19.1.134 FLASH_SECTOR_217

```
#define FLASH_SECTOR_217 FLASH_SECTOR(217)
```

在文件 [lks32mc09x_flash.h](#) 第 224 行定义.

3.19.1.135 FLASH_SECTOR_218

```
#define FLASH_SECTOR_218 FLASH_SECTOR(218)
```

在文件 [lks32mc09x_flash.h](#) 第 225 行定义.

3.19.1.136 FLASH_SECTOR_219

```
#define FLASH_SECTOR_219 FLASH_SECTOR(219)
```

在文件 [lks32mc09x_flash.h](#) 第 226 行定义.

3.19.1.137 FLASH_SECTOR_22

```
#define FLASH_SECTOR_22 FLASH_SECTOR(22)
```

在文件 [lks32mc09x_flash.h](#) 第 29 行定义.

3.19.1.138 FLASH_SECTOR_220

```
#define FLASH_SECTOR_220 FLASH_SECTOR(220)
```

在文件 [lks32mc09x_flash.h](#) 第 227 行定义.

3.19.1.139 FLASH_SECTOR_221

```
#define FLASH_SECTOR_221 FLASH_SECTOR(221)
```

在文件 [lks32mc09x_flash.h](#) 第 228 行定义.

3.19.1.140 FLASH_SECTOR_222

```
#define FLASH_SECTOR_222 FLASH_SECTOR(222)
```

在文件 [lks32mc09x_flash.h](#) 第 229 行定义.

3.19.1.141 FLASH_SECTOR_223

```
#define FLASH_SECTOR_223 FLASH_SECTOR(223)
```

在文件 [lks32mc09x_flash.h](#) 第 230 行定义.

3.19.1.142 FLASH_SECTOR_224

```
#define FLASH_SECTOR_224 FLASH_SECTOR(224)
```

在文件 [lks32mc09x_flash.h](#) 第 231 行定义.

3.19.1.143 FLASH_SECTOR_225

```
#define FLASH_SECTOR_225 FLASH_SECTOR(225)
```

在文件 [lks32mc09x_flash.h](#) 第 232 行定义.

3.19.1.144 FLASH_SECTOR_226

```
#define FLASH_SECTOR_226 FLASH_SECTOR(226)
```

在文件 [lks32mc09x_flash.h](#) 第 233 行定义.

3.19.1.145 FLASH_SECTOR_227

```
#define FLASH_SECTOR_227 FLASH_SECTOR(227)
```

在文件 [lks32mc09x_flash.h](#) 第 234 行定义.

3.19.1.146 FLASH_SECTOR_228

```
#define FLASH_SECTOR_228 FLASH_SECTOR(228)
```

在文件 [lks32mc09x_flash.h](#) 第 235 行定义.

3.19.1.147 FLASH_SECTOR_229

```
#define FLASH_SECTOR_229 FLASH_SECTOR(229)
```

在文件 [lks32mc09x_flash.h](#) 第 236 行定义.

3.19.1.148 FLASH_SECTOR_23

```
#define FLASH_SECTOR_23 FLASH_SECTOR(23)
```

在文件 [lks32mc09x_flash.h](#) 第 30 行定义.

3.19.1.149 FLASH_SECTOR_230

```
#define FLASH_SECTOR_230 FLASH_SECTOR(230)
```

在文件 [lks32mc09x_flash.h](#) 第 237 行定义.

3.19.1.150 FLASH_SECTOR_231

```
#define FLASH_SECTOR_231 FLASH_SECTOR(231)
```

在文件 [lks32mc09x_flash.h](#) 第 238 行定义.

3.19.1.151 FLASH_SECTOR_232

```
#define FLASH_SECTOR_232 FLASH_SECTOR(232)
```

在文件 [lks32mc09x_flash.h](#) 第 239 行定义.

3.19.1.152 FLASH_SECTOR_233

```
#define FLASH_SECTOR_233 FLASH_SECTOR(233)
```

在文件 [lks32mc09x_flash.h](#) 第 240 行定义.

3.19.1.153 FLASH_SECTOR_234

```
#define FLASH_SECTOR_234 FLASH_SECTOR(234)
```

在文件 [lks32mc09x_flash.h](#) 第 241 行定义.

3.19.1.154 FLASH_SECTOR_235

```
#define FLASH_SECTOR_235 FLASH_SECTOR(235)
```

在文件 [lks32mc09x_flash.h](#) 第 242 行定义.

3.19.1.155 FLASH_SECTOR_236

```
#define FLASH_SECTOR_236 FLASH_SECTOR(236)
```

在文件 [lks32mc09x_flash.h](#) 第 243 行定义.

3.19.1.156 FLASH_SECTOR_237

```
#define FLASH_SECTOR_237 FLASH_SECTOR(237)
```

在文件 [lks32mc09x_flash.h](#) 第 244 行定义.

3.19.1.157 FLASH_SECTOR_238

```
#define FLASH_SECTOR_238 FLASH_SECTOR(238)
```

在文件 [lks32mc09x_flash.h](#) 第 245 行定义.

3.19.1.158 FLASH_SECTOR_239

```
#define FLASH_SECTOR_239 FLASH_SECTOR(239)
```

在文件 [lks32mc09x_flash.h](#) 第 246 行定义.

3.19.1.159 FLASH_SECTOR_24

```
#define FLASH_SECTOR_24 FLASH_SECTOR(24)
```

在文件 [lks32mc09x_flash.h](#) 第 31 行定义.

3.19.1.160 FLASH_SECTOR_240

```
#define FLASH_SECTOR_240 FLASH_SECTOR(240)
```

在文件 [lks32mc09x_flash.h](#) 第 247 行定义.

3.19.1.161 FLASH_SECTOR_241

```
#define FLASH_SECTOR_241 FLASH_SECTOR(241)
```

在文件 [lks32mc09x_flash.h](#) 第 248 行定义.

3.19.1.162 FLASH_SECTOR_242

```
#define FLASH_SECTOR_242 FLASH_SECTOR(242)
```

在文件 [lks32mc09x_flash.h](#) 第 249 行定义.

3.19.1.163 FLASH_SECTOR_243

```
#define FLASH_SECTOR_243 FLASH_SECTOR(243)
```

在文件 [lks32mc09x_flash.h](#) 第 250 行定义.

3.19.1.164 FLASH_SECTOR_244

```
#define FLASH_SECTOR_244 FLASH_SECTOR(244)
```

在文件 [lks32mc09x_flash.h](#) 第 251 行定义.

3.19.1.165 FLASH_SECTOR_245

```
#define FLASH_SECTOR_245 FLASH_SECTOR(245)
```

在文件 [lks32mc09x_flash.h](#) 第 252 行定义.

3.19.1.166 FLASH_SECTOR_246

```
#define FLASH_SECTOR_246 FLASH_SECTOR(246)
```

在文件 [lks32mc09x_flash.h](#) 第 253 行定义.

3.19.1.167 FLASH_SECTOR_247

```
#define FLASH_SECTOR_247 FLASH_SECTOR(247)
```

在文件 [lks32mc09x_flash.h](#) 第 254 行定义.

3.19.1.168 FLASH_SECTOR_248

```
#define FLASH_SECTOR_248 FLASH_SECTOR(248)
```

在文件 [lks32mc09x_flash.h](#) 第 255 行定义.

3.19.1.169 FLASH_SECTOR_249

```
#define FLASH_SECTOR_249 FLASH_SECTOR(249)
```

在文件 [lks32mc09x_flash.h](#) 第 256 行定义.

3.19.1.170 FLASH_SECTOR_25

```
#define FLASH_SECTOR_25 FLASH_SECTOR(25)
```

在文件 [lks32mc09x_flash.h](#) 第 32 行定义.

3.19.1.171 FLASH_SECTOR_250

```
#define FLASH_SECTOR_250 FLASH_SECTOR(250)
```

在文件 [lks32mc09x_flash.h](#) 第 257 行定义.

3.19.1.172 FLASH_SECTOR_251

```
#define FLASH_SECTOR_251 FLASH_SECTOR(251)
```

在文件 [lks32mc09x_flash.h](#) 第 258 行定义.

3.19.1.173 FLASH_SECTOR_252

```
#define FLASH_SECTOR_252 FLASH_SECTOR(252)
```

在文件 [lks32mc09x_flash.h](#) 第 259 行定义.

3.19.1.174 FLASH_SECTOR_253

```
#define FLASH_SECTOR_253 FLASH_SECTOR(253)
```

在文件 [lks32mc09x_flash.h](#) 第 260 行定义.

3.19.1.175 FLASH_SECTOR_254

```
#define FLASH_SECTOR_254 FLASH_SECTOR(254)
```

在文件 [lks32mc09x_flash.h](#) 第 261 行定义.

3.19.1.176 FLASH_SECTOR_255

```
#define FLASH_SECTOR_255 FLASH_SECTOR(255)
```

在文件 [lks32mc09x_flash.h](#) 第 262 行定义.

3.19.1.177 FLASH_SECTOR_26

```
#define FLASH_SECTOR_26 FLASH_SECTOR(26)
```

在文件 [lks32mc09x_flash.h](#) 第 33 行定义.

3.19.1.178 FLASH_SECTOR_27

```
#define FLASH_SECTOR_27 FLASH_SECTOR(27)
```

在文件 [lks32mc09x_flash.h](#) 第 34 行定义.

3.19.1.179 FLASH_SECTOR_28

```
#define FLASH_SECTOR_28 FLASH_SECTOR(28)
```

在文件 [lks32mc09x_flash.h](#) 第 35 行定义.

3.19.1.180 FLASH_SECTOR_29

```
#define FLASH_SECTOR_29 FLASH_SECTOR(29)
```

在文件 [lks32mc09x_flash.h](#) 第 36 行定义.

3.19.1.181 FLASH_SECTOR_3

```
#define FLASH_SECTOR_3 FLASH_SECTOR(3)
```

在文件 [lks32mc09x_flash.h](#) 第 10 行定义.

3.19.1.182 FLASH_SECTOR_30

```
#define FLASH_SECTOR_30 FLASH_SECTOR(30)
```

在文件 [lks32mc09x_flash.h](#) 第 37 行定义.

3.19.1.183 FLASH_SECTOR_31

```
#define FLASH_SECTOR_31 FLASH_SECTOR(31)
```

在文件 [lks32mc09x_flash.h](#) 第 38 行定义.

3.19.1.184 FLASH_SECTOR_32

```
#define FLASH_SECTOR_32 FLASH_SECTOR(32)
```

在文件 [lks32mc09x_flash.h](#) 第 39 行定义.

3.19.1.185 FLASH_SECTOR_33

```
#define FLASH_SECTOR_33 FLASH_SECTOR(33)
```

在文件 [lks32mc09x_flash.h](#) 第 40 行定义.

3.19.1.186 FLASH_SECTOR_34

```
#define FLASH_SECTOR_34 FLASH_SECTOR(34)
```

在文件 [lks32mc09x_flash.h](#) 第 41 行定义.

3.19.1.187 FLASH_SECTOR_35

```
#define FLASH_SECTOR_35 FLASH_SECTOR(35)
```

在文件 [lks32mc09x_flash.h](#) 第 42 行定义.

3.19.1.188 FLASH_SECTOR_36

```
#define FLASH_SECTOR_36 FLASH_SECTOR(36)
```

在文件 [lks32mc09x_flash.h](#) 第 43 行定义.

3.19.1.189 FLASH_SECTOR_37

```
#define FLASH_SECTOR_37 FLASH_SECTOR(37)
```

在文件 [lks32mc09x_flash.h](#) 第 44 行定义.

3.19.1.190 FLASH_SECTOR_38

```
#define FLASH_SECTOR_38 FLASH_SECTOR(38)
```

在文件 [lks32mc09x_flash.h](#) 第 45 行定义.

3.19.1.191 FLASH_SECTOR_39

```
#define FLASH_SECTOR_39 FLASH_SECTOR(39)
```

在文件 [lks32mc09x_flash.h](#) 第 46 行定义.

3.19.1.192 FLASH_SECTOR_4

```
#define FLASH_SECTOR_4 FLASH_SECTOR(4)
```

在文件 [lks32mc09x_flash.h](#) 第 11 行定义.

3.19.1.193 FLASH_SECTOR_40

```
#define FLASH_SECTOR_40 FLASH_SECTOR(40)
```

在文件 [lks32mc09x_flash.h](#) 第 47 行定义.

3.19.1.194 FLASH_SECTOR_41

```
#define FLASH_SECTOR_41 FLASH_SECTOR(41)
```

在文件 [lks32mc09x_flash.h](#) 第 48 行定义.

3.19.1.195 FLASH_SECTOR_42

```
#define FLASH_SECTOR_42 FLASH_SECTOR(42)
```

在文件 [lks32mc09x_flash.h](#) 第 49 行定义.

3.19.1.196 FLASH_SECTOR_43

```
#define FLASH_SECTOR_43 FLASH_SECTOR(43)
```

在文件 [lks32mc09x_flash.h](#) 第 50 行定义.

3.19.1.197 FLASH_SECTOR_44

```
#define FLASH_SECTOR_44 FLASH_SECTOR(44)
```

在文件 [lks32mc09x_flash.h](#) 第 51 行定义.

3.19.1.198 FLASH_SECTOR_45

```
#define FLASH_SECTOR_45 FLASH_SECTOR(45)
```

在文件 [lks32mc09x_flash.h](#) 第 52 行定义.

3.19.1.199 FLASH_SECTOR_46

```
#define FLASH_SECTOR_46 FLASH_SECTOR(46)
```

在文件 [lks32mc09x_flash.h](#) 第 53 行定义.

3.19.1.200 FLASH_SECTOR_47

```
#define FLASH_SECTOR_47 FLASH_SECTOR(47)
```

在文件 [lks32mc09x_flash.h](#) 第 54 行定义.

3.19.1.201 FLASH_SECTOR_48

```
#define FLASH_SECTOR_48 FLASH_SECTOR(48)
```

在文件 [lks32mc09x_flash.h](#) 第 55 行定义.

3.19.1.202 FLASH_SECTOR_49

```
#define FLASH_SECTOR_49 FLASH_SECTOR(49)
```

在文件 [lks32mc09x_flash.h](#) 第 56 行定义.

3.19.1.203 FLASH_SECTOR_5

```
#define FLASH_SECTOR_5 FLASH_SECTOR(5)
```

在文件 [lks32mc09x_flash.h](#) 第 12 行定义.

3.19.1.204 FLASH_SECTOR_50

```
#define FLASH_SECTOR_50 FLASH_SECTOR(50)
```

在文件 [lks32mc09x_flash.h](#) 第 57 行定义.

3.19.1.205 FLASH_SECTOR_51

```
#define FLASH_SECTOR_51 FLASH_SECTOR(51)
```

在文件 [lks32mc09x_flash.h](#) 第 58 行定义.

3.19.1.206 FLASH_SECTOR_52

```
#define FLASH_SECTOR_52 FLASH_SECTOR(52)
```

在文件 [lks32mc09x_flash.h](#) 第 59 行定义.

3.19.1.207 FLASH_SECTOR_53

```
#define FLASH_SECTOR_53 FLASH_SECTOR(53)
```

在文件 [lks32mc09x_flash.h](#) 第 60 行定义.

3.19.1.208 FLASH_SECTOR_54

```
#define FLASH_SECTOR_54 FLASH_SECTOR(54)
```

在文件 [lks32mc09x_flash.h](#) 第 61 行定义.

3.19.1.209 FLASH_SECTOR_55

```
#define FLASH_SECTOR_55 FLASH_SECTOR(55)
```

在文件 [lks32mc09x_flash.h](#) 第 62 行定义.

3.19.1.210 FLASH_SECTOR_56

```
#define FLASH_SECTOR_56 FLASH_SECTOR(56)
```

在文件 [lks32mc09x_flash.h](#) 第 63 行定义.

3.19.1.211 FLASH_SECTOR_57

```
#define FLASH_SECTOR_57 FLASH_SECTOR(57)
```

在文件 [lks32mc09x_flash.h](#) 第 64 行定义.

3.19.1.212 FLASH_SECTOR_58

```
#define FLASH_SECTOR_58 FLASH_SECTOR(58)
```

在文件 [lks32mc09x_flash.h](#) 第 65 行定义.

3.19.1.213 FLASH_SECTOR_59

```
#define FLASH_SECTOR_59 FLASH_SECTOR(59)
```

在文件 [lks32mc09x_flash.h](#) 第 66 行定义.

3.19.1.214 FLASH_SECTOR_6

```
#define FLASH_SECTOR_6 FLASH_SECTOR(6)
```

在文件 [lks32mc09x_flash.h](#) 第 13 行定义.

3.19.1.215 FLASH_SECTOR_60

```
#define FLASH_SECTOR_60 FLASH_SECTOR(60)
```

在文件 [lks32mc09x_flash.h](#) 第 67 行定义.

3.19.1.216 FLASH_SECTOR_61

```
#define FLASH_SECTOR_61 FLASH_SECTOR(61)
```

在文件 [lks32mc09x_flash.h](#) 第 68 行定义.

3.19.1.217 FLASH_SECTOR_62

```
#define FLASH_SECTOR_62 FLASH_SECTOR(62)
```

在文件 [lks32mc09x_flash.h](#) 第 69 行定义.

3.19.1.218 FLASH_SECTOR_63

```
#define FLASH_SECTOR_63 FLASH_SECTOR(63)
```

在文件 [lks32mc09x_flash.h](#) 第 70 行定义.

3.19.1.219 FLASH_SECTOR_64

```
#define FLASH_SECTOR_64 FLASH_SECTOR(64)
```

在文件 [lks32mc09x_flash.h](#) 第 71 行定义.

3.19.1.220 FLASH_SECTOR_65

```
#define FLASH_SECTOR_65 FLASH_SECTOR(65)
```

在文件 [lks32mc09x_flash.h](#) 第 72 行定义.

3.19.1.221 FLASH_SECTOR_66

```
#define FLASH_SECTOR_66 FLASH_SECTOR(66)
```

在文件 [lks32mc09x_flash.h](#) 第 73 行定义.

3.19.1.222 FLASH_SECTOR_67

```
#define FLASH_SECTOR_67 FLASH_SECTOR(67)
```

在文件 [lks32mc09x_flash.h](#) 第 74 行定义.

3.19.1.223 FLASH_SECTOR_68

```
#define FLASH_SECTOR_68 FLASH_SECTOR(68)
```

在文件 [lks32mc09x_flash.h](#) 第 75 行定义.

3.19.1.224 FLASH_SECTOR_69

```
#define FLASH_SECTOR_69 FLASH_SECTOR(69)
```

在文件 [lks32mc09x_flash.h](#) 第 76 行定义.

3.19.1.225 FLASH_SECTOR_7

```
#define FLASH_SECTOR_7 FLASH_SECTOR(7)
```

在文件 [lks32mc09x_flash.h](#) 第 14 行定义.

3.19.1.226 FLASH_SECTOR_70

```
#define FLASH_SECTOR_70 FLASH_SECTOR(70)
```

在文件 [lks32mc09x_flash.h](#) 第 77 行定义.

3.19.1.227 FLASH_SECTOR_71

```
#define FLASH_SECTOR_71 FLASH_SECTOR(71)
```

在文件 [lks32mc09x_flash.h](#) 第 78 行定义.

3.19.1.228 FLASH_SECTOR_72

```
#define FLASH_SECTOR_72 FLASH_SECTOR(72)
```

在文件 [lks32mc09x_flash.h](#) 第 79 行定义.

3.19.1.229 FLASH_SECTOR_73

```
#define FLASH_SECTOR_73 FLASH_SECTOR(73)
```

在文件 [lks32mc09x_flash.h](#) 第 80 行定义.

3.19.1.230 FLASH_SECTOR_74

```
#define FLASH_SECTOR_74 FLASH_SECTOR(74)
```

在文件 [lks32mc09x_flash.h](#) 第 81 行定义.

3.19.1.231 FLASH_SECTOR_75

```
#define FLASH_SECTOR_75 FLASH_SECTOR(75)
```

在文件 [lks32mc09x_flash.h](#) 第 82 行定义.

3.19.1.232 FLASH_SECTOR_76

```
#define FLASH_SECTOR_76 FLASH_SECTOR(76)
```

在文件 [lks32mc09x_flash.h](#) 第 83 行定义.

3.19.1.233 FLASH_SECTOR_77

```
#define FLASH_SECTOR_77 FLASH_SECTOR(77)
```

在文件 [lks32mc09x_flash.h](#) 第 84 行定义.

3.19.1.234 FLASH_SECTOR_78

```
#define FLASH_SECTOR_78 FLASH_SECTOR(78)
```

在文件 [lks32mc09x_flash.h](#) 第 85 行定义.

3.19.1.235 FLASH_SECTOR_79

```
#define FLASH_SECTOR_79 FLASH_SECTOR(79)
```

在文件 [lks32mc09x_flash.h](#) 第 86 行定义.

3.19.1.236 FLASH_SECTOR_8

```
#define FLASH_SECTOR_8 FLASH_SECTOR(8)
```

在文件 [lks32mc09x_flash.h](#) 第 15 行定义.

3.19.1.237 FLASH_SECTOR_80

```
#define FLASH_SECTOR_80 FLASH_SECTOR(80)
```

在文件 [lks32mc09x_flash.h](#) 第 87 行定义.

3.19.1.238 FLASH_SECTOR_81

```
#define FLASH_SECTOR_81 FLASH_SECTOR(81)
```

在文件 [lks32mc09x_flash.h](#) 第 88 行定义.

3.19.1.239 FLASH_SECTOR_82

```
#define FLASH_SECTOR_82 FLASH_SECTOR(82)
```

在文件 [lks32mc09x_flash.h](#) 第 89 行定义.

3.19.1.240 FLASH_SECTOR_83

```
#define FLASH_SECTOR_83 FLASH_SECTOR(83)
```

在文件 [lks32mc09x_flash.h](#) 第 90 行定义.

3.19.1.241 FLASH_SECTOR_84

```
#define FLASH_SECTOR_84 FLASH_SECTOR(84)
```

在文件 [lks32mc09x_flash.h](#) 第 91 行定义.

3.19.1.242 FLASH_SECTOR_85

```
#define FLASH_SECTOR_85 FLASH_SECTOR(85)
```

在文件 [lks32mc09x_flash.h](#) 第 92 行定义.

3.19.1.243 FLASH_SECTOR_86

```
#define FLASH_SECTOR_86 FLASH_SECTOR(86)
```

在文件 [lks32mc09x_flash.h](#) 第 93 行定义.

3.19.1.244 FLASH_SECTOR_87

```
#define FLASH_SECTOR_87 FLASH_SECTOR(87)
```

在文件 [lks32mc09x_flash.h](#) 第 94 行定义.

3.19.1.245 FLASH_SECTOR_88

```
#define FLASH_SECTOR_88 FLASH_SECTOR(88)
```

在文件 [lks32mc09x_flash.h](#) 第 95 行定义.

3.19.1.246 FLASH_SECTOR_89

```
#define FLASH_SECTOR_89 FLASH_SECTOR(89)
```

在文件 [lks32mc09x_flash.h](#) 第 96 行定义.

3.19.1.247 FLASH_SECTOR_9

```
#define FLASH_SECTOR_9 FLASH_SECTOR(9)
```

在文件 [lks32mc09x_flash.h](#) 第 16 行定义.

3.19.1.248 FLASH_SECTOR_90

```
#define FLASH_SECTOR_90 FLASH_SECTOR(90)
```

在文件 [lks32mc09x_flash.h](#) 第 97 行定义.

3.19.1.249 FLASH_SECTOR_91

```
#define FLASH_SECTOR_91 FLASH_SECTOR(91)
```

在文件 [lks32mc09x_flash.h](#) 第 98 行定义.

3.19.1.250 FLASH_SECTOR_92

```
#define FLASH_SECTOR_92 FLASH_SECTOR(92)
```

在文件 [lks32mc09x_flash.h](#) 第 99 行定义.

3.19.1.251 FLASH_SECTOR_93

```
#define FLASH_SECTOR_93 FLASH_SECTOR(93)
```

在文件 [lks32mc09x_flash.h](#) 第 100 行定义.

3.19.1.252 FLASH_SECTOR_94

```
#define FLASH_SECTOR_94 FLASH_SECTOR(94)
```

在文件 [lks32mc09x_flash.h](#) 第 101 行定义.

3.19.1.253 FLASH_SECTOR_95

```
#define FLASH_SECTOR_95 FLASH_SECTOR(95)
```

在文件 [lks32mc09x_flash.h](#) 第 102 行定义.

3.19.1.254 FLASH_SECTOR_96

```
#define FLASH_SECTOR_96 FLASH_SECTOR(96)
```

在文件 [lks32mc09x_flash.h](#) 第 103 行定义.

3.19.1.255 FLASH_SECTOR_97

```
#define FLASH_SECTOR_97 FLASH_SECTOR(97)
```

在文件 [lks32mc09x_flash.h](#) 第 104 行定义.

3.19.1.256 FLASH_SECTOR_98

```
#define FLASH_SECTOR_98 FLASH_SECTOR(98)
```

在文件 [lks32mc09x_flash.h](#) 第 105 行定义.

3.19.1.257 FLASH_SECTOR_99

```
#define FLASH_SECTOR_99 FLASH_SECTOR(99)
```

在文件 [lks32mc09x_flash.h](#) 第 106 行定义.

3.19.1.258 FLASH_SECTOR_END_128K

```
#define FLASH_SECTOR_END_128K FLASH_SECTOR_255
```

在文件 [lks32mc09x_flash.h](#) 第 264 行定义.

3.19.1.259 FLASH_SECTOR_END_32K

```
#define FLASH_SECTOR_END_32K FLASH_SECTOR_63
```

在文件 [lks32mc09x_flash.h](#) 第 266 行定义.

3.19.1.260 FLASH_SECTOR_END_64K

```
#define FLASH_SECTOR_END_64K FLASH_SECTOR_127
```

在文件 [lks32mc09x_flash.h](#) 第 265 行定义.

3.19.2 函数说明

3.19.2.1 FLASH_Erase()

```
void FLASH_Erase (
    uint32_t adr )
```

擦除指定扇区

参数

<i>adr</i>	目标地址
<i>adr</i>	目标地址

注解

触发掉电检测后，对flash的操作无效，掉电检测的配置参考SYS_SetPowerFailureDetection

在文件 [lks32mc09x_flash.c](#) 第 53 行定义.

3.19.2.2 FLASH.EraseNvr()

```
void FLASH_EraseNvr (
    uint32_t adr )
```

擦除指定扇区(NVR区域)

参数

<i>adr</i>	目标地址

注解

触发掉电检测后，对flash的操作无效，掉电检测的配置参考SYS_SetPowerFailureDetection

在文件 [lks32mc09x_flash.c](#) 第 102 行定义.

3.19.2.3 FLASH.Read()

```
void FLASH_Read (
    uint32_t adr,
    uint8_t * data,
    uint32_t len )
```

从指定地址读取数据

参数

<i>adr</i>	目标地址
<i>buf</i>	数据缓冲区指针
<i>len</i>	数据长度

在文件 [lks32mc09x_flash.c](#) 第 38 行定义.

3.19.2.4 FLASH.ReadNvr()

```
void FLASH_ReadNvr (
    uint32_t adr,
    uint8_t * data,
    uint32_t len )
```

从指定地址读取数据(NVR区域)

参数

<i>adr</i>	目标地址
<i>buf</i>	数据缓冲区指针
<i>len</i>	数据长度

在文件 [lks32mc09x_flash.c](#) 第 123 行定义.

3.19.2.5 FLASH.Write()

```
void FLASH_Write (
    uint32_t adr,
    uint8_t * data,
    uint32_t len )
```

向指定地址写入数据

参数

<i>adr</i>	目标地址
<i>data</i>	数据缓冲区指针
<i>len</i>	数据长度
<i>adr</i>	目标地址
<i>data</i>	数据缓冲区指针
<i>len</i>	数据长度

注解

触发掉电检测后，对flash的操作无效，掉电检测的配置参考SYS.SetPowerFailureDetection

在文件 [lks32mc09x_flash.c](#) 第 10 行定义.

3.19.2.6 FLASH.WriteNvr()

```
void FLASH_WriteNvr (
    uint32_t adr,
    uint8_t * data,
    uint32_t len )
```

向指定地址写入数据(NVR区域)

参数

<i>adr</i>	目标地址
<i>data</i>	数据缓冲区指针
<i>len</i>	数据长度

注解

触发掉电检测后，对flash的操作无效，掉电检测的配置参考SYS_SetPowerFailureDetection

在文件 [lks32mc09x_flash.c](#) 第 75 行定义.

3.20 lks32mc09x_flash.h

[浏览该文件的文档.](#)

```

00001 #ifndef __LKS32MC09X_FLASH_H__
00002 #define __LKS32MC09X_FLASH_H__
00003
00004 #include "lks32mc09x_reg.h"
00005 // 各扇区的定义
00006 #define FLASH_SECTOR(x)          (x * 0x200)          // 0.5KB per sector (0x200 = 512 bytes)
00007 #define FLASH_SECTOR_0           FLASH_SECTOR(0)      // 0x00000
00008 #define FLASH_SECTOR_1           FLASH_SECTOR(1)      // 0x00200
00009 #define FLASH_SECTOR_2           FLASH_SECTOR(2)      // 0x00400
00010 #define FLASH_SECTOR_3           FLASH_SECTOR(3)      // 0x00600
00011 #define FLASH_SECTOR_4           FLASH_SECTOR(4)      // 0x00800
00012 #define FLASH_SECTOR_5           FLASH_SECTOR(5)      // 0x00A00
00013 #define FLASH_SECTOR_6           FLASH_SECTOR(6)      // 0x00C00
00014 #define FLASH_SECTOR_7           FLASH_SECTOR(7)      // 0x00E00
00015 #define FLASH_SECTOR_8           FLASH_SECTOR(8)      // 0x01000
00016 #define FLASH_SECTOR_9           FLASH_SECTOR(9)      // 0x01200
00017 #define FLASH_SECTOR_10          FLASH_SECTOR(10)     // 0x01400
00018 #define FLASH_SECTOR_11          FLASH_SECTOR(11)     // 0x01600
00019 #define FLASH_SECTOR_12          FLASH_SECTOR(12)     // 0x01800
00020 #define FLASH_SECTOR_13          FLASH_SECTOR(13)     // 0x01A00
00021 #define FLASH_SECTOR_14          FLASH_SECTOR(14)     // 0x01C00
00022 #define FLASH_SECTOR_15          FLASH_SECTOR(15)     // 0x01E00
00023 #define FLASH_SECTOR_16          FLASH_SECTOR(16)     // 0x02000
00024 #define FLASH_SECTOR_17          FLASH_SECTOR(17)     // 0x02200
00025 #define FLASH_SECTOR_18          FLASH_SECTOR(18)     // 0x02400
00026 #define FLASH_SECTOR_19          FLASH_SECTOR(19)     // 0x02600
00027 #define FLASH_SECTOR_20          FLASH_SECTOR(20)     // 0x02800
00028 #define FLASH_SECTOR_21          FLASH_SECTOR(21)     // 0x02A00
00029 #define FLASH_SECTOR_22          FLASH_SECTOR(22)     // 0x02C00
00030 #define FLASH_SECTOR_23          FLASH_SECTOR(23)     // 0x02E00
00031 #define FLASH_SECTOR_24          FLASH_SECTOR(24)     // 0x03000
00032 #define FLASH_SECTOR_25          FLASH_SECTOR(25)     // 0x03200
00033 #define FLASH_SECTOR_26          FLASH_SECTOR(26)     // 0x03400
00034 #define FLASH_SECTOR_27          FLASH_SECTOR(27)     // 0x03600
00035 #define FLASH_SECTOR_28          FLASH_SECTOR(28)     // 0x03800
00036 #define FLASH_SECTOR_29          FLASH_SECTOR(29)     // 0x03A00
00037 #define FLASH_SECTOR_30          FLASH_SECTOR(30)     // 0x03C00
00038 #define FLASH_SECTOR_31          FLASH_SECTOR(31)     // 0x03E00
00039 #define FLASH_SECTOR_32          FLASH_SECTOR(32)     // 0x04000
00040 #define FLASH_SECTOR_33          FLASH_SECTOR(33)     // 0x04200
00041 #define FLASH_SECTOR_34          FLASH_SECTOR(34)     // 0x04400
00042 #define FLASH_SECTOR_35          FLASH_SECTOR(35)     // 0x04600
00043 #define FLASH_SECTOR_36          FLASH_SECTOR(36)     // 0x04800
00044 #define FLASH_SECTOR_37          FLASH_SECTOR(37)     // 0x04A00
00045 #define FLASH_SECTOR_38          FLASH_SECTOR(38)     // 0x04C00
00046 #define FLASH_SECTOR_39          FLASH_SECTOR(39)     // 0x04E00
00047 #define FLASH_SECTOR_40          FLASH_SECTOR(40)     // 0x05000
00048 #define FLASH_SECTOR_41          FLASH_SECTOR(41)     // 0x05200
00049 #define FLASH_SECTOR_42          FLASH_SECTOR(42)     // 0x05400
00050 #define FLASH_SECTOR_43          FLASH_SECTOR(43)     // 0x05600
00051 #define FLASH_SECTOR_44          FLASH_SECTOR(44)     // 0x05800
00052 #define FLASH_SECTOR_45          FLASH_SECTOR(45)     // 0x05A00
00053 #define FLASH_SECTOR_46          FLASH_SECTOR(46)     // 0x05C00
00054 #define FLASH_SECTOR_47          FLASH_SECTOR(47)     // 0x05E00
00055 #define FLASH_SECTOR_48          FLASH_SECTOR(48)     // 0x06000
00056 #define FLASH_SECTOR_49          FLASH_SECTOR(49)     // 0x06200
00057 #define FLASH_SECTOR_50          FLASH_SECTOR(50)     // 0x06400

```

```
00058 #define FLASH_SECTOR_51    FLASH_SECTOR(51) // 0x06600
00059 #define FLASH_SECTOR_52    FLASH_SECTOR(52) // 0x06800
00060 #define FLASH_SECTOR_53    FLASH_SECTOR(53) // 0x06A00
00061 #define FLASH_SECTOR_54    FLASH_SECTOR(54) // 0x06C00
00062 #define FLASH_SECTOR_55    FLASH_SECTOR(55) // 0x06E00
00063 #define FLASH_SECTOR_56    FLASH_SECTOR(56) // 0x07000
00064 #define FLASH_SECTOR_57    FLASH_SECTOR(57) // 0x07200
00065 #define FLASH_SECTOR_58    FLASH_SECTOR(58) // 0x07400
00066 #define FLASH_SECTOR_59    FLASH_SECTOR(59) // 0x07600
00067 #define FLASH_SECTOR_60    FLASH_SECTOR(60) // 0x07800
00068 #define FLASH_SECTOR_61    FLASH_SECTOR(61) // 0x07A00
00069 #define FLASH_SECTOR_62    FLASH_SECTOR(62) // 0x07C00
00070 #define FLASH_SECTOR_63    FLASH_SECTOR(63) // 0x07E00
00071 #define FLASH_SECTOR_64    FLASH_SECTOR(64) // 0x08000
00072 #define FLASH_SECTOR_65    FLASH_SECTOR(65) // 0x08200
00073 #define FLASH_SECTOR_66    FLASH_SECTOR(66) // 0x08400
00074 #define FLASH_SECTOR_67    FLASH_SECTOR(67) // 0x08600
00075 #define FLASH_SECTOR_68    FLASH_SECTOR(68) // 0x08800
00076 #define FLASH_SECTOR_69    FLASH_SECTOR(69) // 0x08A00
00077 #define FLASH_SECTOR_70    FLASH_SECTOR(70) // 0x08C00
00078 #define FLASH_SECTOR_71    FLASH_SECTOR(71) // 0x08E00
00079 #define FLASH_SECTOR_72    FLASH_SECTOR(72) // 0x09000
00080 #define FLASH_SECTOR_73    FLASH_SECTOR(73) // 0x09200
00081 #define FLASH_SECTOR_74    FLASH_SECTOR(74) // 0x09400
00082 #define FLASH_SECTOR_75    FLASH_SECTOR(75) // 0x09600
00083 #define FLASH_SECTOR_76    FLASH_SECTOR(76) // 0x09800
00084 #define FLASH_SECTOR_77    FLASH_SECTOR(77) // 0x09A00
00085 #define FLASH_SECTOR_78    FLASH_SECTOR(78) // 0x09C00
00086 #define FLASH_SECTOR_79    FLASH_SECTOR(79) // 0x09E00
00087 #define FLASH_SECTOR_80    FLASH_SECTOR(80) // 0x0A000
00088 #define FLASH_SECTOR_81    FLASH_SECTOR(81) // 0x0A200
00089 #define FLASH_SECTOR_82    FLASH_SECTOR(82) // 0x0A400
00090 #define FLASH_SECTOR_83    FLASH_SECTOR(83) // 0x0A600
00091 #define FLASH_SECTOR_84    FLASH_SECTOR(84) // 0x0A800
00092 #define FLASH_SECTOR_85    FLASH_SECTOR(85) // 0x0AA00
00093 #define FLASH_SECTOR_86    FLASH_SECTOR(86) // 0x0AC00
00094 #define FLASH_SECTOR_87    FLASH_SECTOR(87) // 0x0AE00
00095 #define FLASH_SECTOR_88    FLASH_SECTOR(88) // 0x0B000
00096 #define FLASH_SECTOR_89    FLASH_SECTOR(89) // 0x0B200
00097 #define FLASH_SECTOR_90    FLASH_SECTOR(90) // 0x0B400
00098 #define FLASH_SECTOR_91    FLASH_SECTOR(91) // 0x0B600
00099 #define FLASH_SECTOR_92    FLASH_SECTOR(92) // 0x0B800
00100 #define FLASH_SECTOR_93    FLASH_SECTOR(93) // 0x0BA00
00101 #define FLASH_SECTOR_94    FLASH_SECTOR(94) // 0x0BC00
00102 #define FLASH_SECTOR_95    FLASH_SECTOR(95) // 0x0BE00
00103 #define FLASH_SECTOR_96    FLASH_SECTOR(96) // 0x0C000
00104 #define FLASH_SECTOR_97    FLASH_SECTOR(97) // 0x0C200
00105 #define FLASH_SECTOR_98    FLASH_SECTOR(98) // 0x0C400
00106 #define FLASH_SECTOR_99    FLASH_SECTOR(99) // 0x0C600
00107 #define FLASH_SECTOR_100    FLASH_SECTOR(100) // 0x0C800
00108 #define FLASH_SECTOR_101    FLASH_SECTOR(101) // 0x0CA00
00109 #define FLASH_SECTOR_102    FLASH_SECTOR(102) // 0x0CC00
00110 #define FLASH_SECTOR_103    FLASH_SECTOR(103) // 0x0CE00
00111 #define FLASH_SECTOR_104    FLASH_SECTOR(104) // 0x0D000
00112 #define FLASH_SECTOR_105    FLASH_SECTOR(105) // 0x0D200
00113 #define FLASH_SECTOR_106    FLASH_SECTOR(106) // 0x0D400
00114 #define FLASH_SECTOR_107    FLASH_SECTOR(107) // 0x0D600
00115 #define FLASH_SECTOR_108    FLASH_SECTOR(108) // 0x0D800
00116 #define FLASH_SECTOR_109    FLASH_SECTOR(109) // 0x0DA00
00117 #define FLASH_SECTOR_110    FLASH_SECTOR(110) // 0x0DC00
00118 #define FLASH_SECTOR_111    FLASH_SECTOR(111) // 0x0DE00
00119 #define FLASH_SECTOR_112    FLASH_SECTOR(112) // 0x0E000
00120 #define FLASH_SECTOR_113    FLASH_SECTOR(113) // 0x0E200
00121 #define FLASH_SECTOR_114    FLASH_SECTOR(114) // 0x0E400
00122 #define FLASH_SECTOR_115    FLASH_SECTOR(115) // 0x0E600
00123 #define FLASH_SECTOR_116    FLASH_SECTOR(116) // 0x0E800
00124 #define FLASH_SECTOR_117    FLASH_SECTOR(117) // 0x0EA00
00125 #define FLASH_SECTOR_118    FLASH_SECTOR(118) // 0x0EC00
00126 #define FLASH_SECTOR_119    FLASH_SECTOR(119) // 0x0EE00
00127 #define FLASH_SECTOR_120    FLASH_SECTOR(120) // 0x0F000
00128 #define FLASH_SECTOR_121    FLASH_SECTOR(121) // 0x0F200
00129 #define FLASH_SECTOR_122    FLASH_SECTOR(122) // 0x0F400
00130 #define FLASH_SECTOR_123    FLASH_SECTOR(123) // 0x0F600
00131 #define FLASH_SECTOR_124    FLASH_SECTOR(124) // 0x0F800
00132 #define FLASH_SECTOR_125    FLASH_SECTOR(125) // 0x0FA00
00133 #define FLASH_SECTOR_126    FLASH_SECTOR(126) // 0x0FC00
00134 #define FLASH_SECTOR_127    FLASH_SECTOR(127) // 0x0FE00
00135 #define FLASH_SECTOR_128    FLASH_SECTOR(128) // 0x10000
00136 #define FLASH_SECTOR_129    FLASH_SECTOR(129) // 0x10200
00137 #define FLASH_SECTOR_130    FLASH_SECTOR(130) // 0x10400
00138 #define FLASH_SECTOR_131    FLASH_SECTOR(131) // 0x10600
00139 #define FLASH_SECTOR_132    FLASH_SECTOR(132) // 0x10800
00140 #define FLASH_SECTOR_133    FLASH_SECTOR(133) // 0x10A00
00141 #define FLASH_SECTOR_134    FLASH_SECTOR(134) // 0x10C00
00142 #define FLASH_SECTOR_135    FLASH_SECTOR(135) // 0x10E00
00143 #define FLASH_SECTOR_136    FLASH_SECTOR(136) // 0x11000
00144 #define FLASH_SECTOR_137    FLASH_SECTOR(137) // 0x11200
```

```
00145 #define FLASH_SECTOR_138 FLASH_SECTOR(138) // 0x11400
00146 #define FLASH_SECTOR_139 FLASH_SECTOR(139) // 0x11600
00147 #define FLASH_SECTOR_140 FLASH_SECTOR(140) // 0x11800
00148 #define FLASH_SECTOR_141 FLASH_SECTOR(141) // 0x11A00
00149 #define FLASH_SECTOR_142 FLASH_SECTOR(142) // 0x11C00
00150 #define FLASH_SECTOR_143 FLASH_SECTOR(143) // 0x11E00
00151 #define FLASH_SECTOR_144 FLASH_SECTOR(144) // 0x12000
00152 #define FLASH_SECTOR_145 FLASH_SECTOR(145) // 0x12200
00153 #define FLASH_SECTOR_146 FLASH_SECTOR(146) // 0x12400
00154 #define FLASH_SECTOR_147 FLASH_SECTOR(147) // 0x12600
00155 #define FLASH_SECTOR_148 FLASH_SECTOR(148) // 0x12800
00156 #define FLASH_SECTOR_149 FLASH_SECTOR(149) // 0x12A00
00157 #define FLASH_SECTOR_150 FLASH_SECTOR(150) // 0x12C00
00158 #define FLASH_SECTOR_151 FLASH_SECTOR(151) // 0x12E00
00159 #define FLASH_SECTOR_152 FLASH_SECTOR(152) // 0x13000
00160 #define FLASH_SECTOR_153 FLASH_SECTOR(153) // 0x13200
00161 #define FLASH_SECTOR_154 FLASH_SECTOR(154) // 0x13400
00162 #define FLASH_SECTOR_155 FLASH_SECTOR(155) // 0x13600
00163 #define FLASH_SECTOR_156 FLASH_SECTOR(156) // 0x13800
00164 #define FLASH_SECTOR_157 FLASH_SECTOR(157) // 0x13A00
00165 #define FLASH_SECTOR_158 FLASH_SECTOR(158) // 0x13C00
00166 #define FLASH_SECTOR_159 FLASH_SECTOR(159) // 0x13E00
00167 #define FLASH_SECTOR_160 FLASH_SECTOR(160) // 0x14000
00168 #define FLASH_SECTOR_161 FLASH_SECTOR(161) // 0x14200
00169 #define FLASH_SECTOR_162 FLASH_SECTOR(162) // 0x14400
00170 #define FLASH_SECTOR_163 FLASH_SECTOR(163) // 0x14600
00171 #define FLASH_SECTOR_164 FLASH_SECTOR(164) // 0x14800
00172 #define FLASH_SECTOR_165 FLASH_SECTOR(165) // 0x14A00
00173 #define FLASH_SECTOR_166 FLASH_SECTOR(166) // 0x14C00
00174 #define FLASH_SECTOR_167 FLASH_SECTOR(167) // 0x14E00
00175 #define FLASH_SECTOR_168 FLASH_SECTOR(168) // 0x15000
00176 #define FLASH_SECTOR_169 FLASH_SECTOR(169) // 0x15200
00177 #define FLASH_SECTOR_170 FLASH_SECTOR(170) // 0x15400
00178 #define FLASH_SECTOR_171 FLASH_SECTOR(171) // 0x15600
00179 #define FLASH_SECTOR_172 FLASH_SECTOR(172) // 0x15800
00180 #define FLASH_SECTOR_173 FLASH_SECTOR(173) // 0x15A00
00181 #define FLASH_SECTOR_174 FLASH_SECTOR(174) // 0x15C00
00182 #define FLASH_SECTOR_175 FLASH_SECTOR(175) // 0x15E00
00183 #define FLASH_SECTOR_176 FLASH_SECTOR(176) // 0x16000
00184 #define FLASH_SECTOR_177 FLASH_SECTOR(177) // 0x16200
00185 #define FLASH_SECTOR_178 FLASH_SECTOR(178) // 0x16400
00186 #define FLASH_SECTOR_179 FLASH_SECTOR(179) // 0x16600
00187 #define FLASH_SECTOR_180 FLASH_SECTOR(180) // 0x16800
00188 #define FLASH_SECTOR_181 FLASH_SECTOR(181) // 0x16A00
00189 #define FLASH_SECTOR_182 FLASH_SECTOR(182) // 0x16C00
00190 #define FLASH_SECTOR_183 FLASH_SECTOR(183) // 0x16E00
00191 #define FLASH_SECTOR_184 FLASH_SECTOR(184) // 0x17000
00192 #define FLASH_SECTOR_185 FLASH_SECTOR(185) // 0x17200
00193 #define FLASH_SECTOR_186 FLASH_SECTOR(186) // 0x17400
00194 #define FLASH_SECTOR_187 FLASH_SECTOR(187) // 0x17600
00195 #define FLASH_SECTOR_188 FLASH_SECTOR(188) // 0x17800
00196 #define FLASH_SECTOR_189 FLASH_SECTOR(189) // 0x17A00
00197 #define FLASH_SECTOR_190 FLASH_SECTOR(190) // 0x17C00
00198 #define FLASH_SECTOR_191 FLASH_SECTOR(191) // 0x17E00
00199 #define FLASH_SECTOR_192 FLASH_SECTOR(192) // 0x18000
00200 #define FLASH_SECTOR_193 FLASH_SECTOR(193) // 0x18200
00201 #define FLASH_SECTOR_194 FLASH_SECTOR(194) // 0x18400
00202 #define FLASH_SECTOR_195 FLASH_SECTOR(195) // 0x18600
00203 #define FLASH_SECTOR_196 FLASH_SECTOR(196) // 0x18800
00204 #define FLASH_SECTOR_197 FLASH_SECTOR(197) // 0x18A00
00205 #define FLASH_SECTOR_198 FLASH_SECTOR(198) // 0x18C00
00206 #define FLASH_SECTOR_199 FLASH_SECTOR(199) // 0x18E00
00207 #define FLASH_SECTOR_200 FLASH_SECTOR(200) // 0x19000
00208 #define FLASH_SECTOR_201 FLASH_SECTOR(201) // 0x19200
00209 #define FLASH_SECTOR_202 FLASH_SECTOR(202) // 0x19400
00210 #define FLASH_SECTOR_203 FLASH_SECTOR(203) // 0x19600
00211 #define FLASH_SECTOR_204 FLASH_SECTOR(204) // 0x19800
00212 #define FLASH_SECTOR_205 FLASH_SECTOR(205) // 0x19A00
00213 #define FLASH_SECTOR_206 FLASH_SECTOR(206) // 0x19C00
00214 #define FLASH_SECTOR_207 FLASH_SECTOR(207) // 0x19E00
00215 #define FLASH_SECTOR_208 FLASH_SECTOR(208) // 0x1A000
00216 #define FLASH_SECTOR_209 FLASH_SECTOR(209) // 0x1A200
00217 #define FLASH_SECTOR_210 FLASH_SECTOR(210) // 0x1A400
00218 #define FLASH_SECTOR_211 FLASH_SECTOR(211) // 0x1A600
00219 #define FLASH_SECTOR_212 FLASH_SECTOR(212) // 0x1A800
00220 #define FLASH_SECTOR_213 FLASH_SECTOR(213) // 0x1AA00
00221 #define FLASH_SECTOR_214 FLASH_SECTOR(214) // 0x1AC00
00222 #define FLASH_SECTOR_215 FLASH_SECTOR(215) // 0x1AE00
00223 #define FLASH_SECTOR_216 FLASH_SECTOR(216) // 0x1B000
00224 #define FLASH_SECTOR_217 FLASH_SECTOR(217) // 0x1B200
00225 #define FLASH_SECTOR_218 FLASH_SECTOR(218) // 0x1B400
00226 #define FLASH_SECTOR_219 FLASH_SECTOR(219) // 0x1B600
00227 #define FLASH_SECTOR_220 FLASH_SECTOR(220) // 0x1B800
00228 #define FLASH_SECTOR_221 FLASH_SECTOR(221) // 0x1BA00
00229 #define FLASH_SECTOR_222 FLASH_SECTOR(222) // 0x1BC00
00230 #define FLASH_SECTOR_223 FLASH_SECTOR(223) // 0x1BE00
00231 #define FLASH_SECTOR_224 FLASH_SECTOR(224) // 0x1C000
```

```

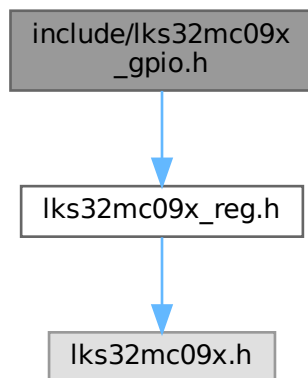
00232 #define FLASH_SECTOR_225    FLASH_SECTOR(225) // 0x1C200
00233 #define FLASH_SECTOR_226    FLASH_SECTOR(226) // 0x1C400
00234 #define FLASH_SECTOR_227    FLASH_SECTOR(227) // 0x1C600
00235 #define FLASH_SECTOR_228    FLASH_SECTOR(228) // 0x1C800
00236 #define FLASH_SECTOR_229    FLASH_SECTOR(229) // 0x1CA00
00237 #define FLASH_SECTOR_230    FLASH_SECTOR(230) // 0x1CC00
00238 #define FLASH_SECTOR_231    FLASH_SECTOR(231) // 0x1CE00
00239 #define FLASH_SECTOR_232    FLASH_SECTOR(232) // 0x1D000
00240 #define FLASH_SECTOR_233    FLASH_SECTOR(233) // 0x1D200
00241 #define FLASH_SECTOR_234    FLASH_SECTOR(234) // 0x1D400
00242 #define FLASH_SECTOR_235    FLASH_SECTOR(235) // 0x1D600
00243 #define FLASH_SECTOR_236    FLASH_SECTOR(236) // 0x1D800
00244 #define FLASH_SECTOR_237    FLASH_SECTOR(237) // 0x1DA00
00245 #define FLASH_SECTOR_238    FLASH_SECTOR(238) // 0x1DC00
00246 #define FLASH_SECTOR_239    FLASH_SECTOR(239) // 0x1DE00
00247 #define FLASH_SECTOR_240    FLASH_SECTOR(240) // 0x1E000
00248 #define FLASH_SECTOR_241    FLASH_SECTOR(241) // 0x1E200
00249 #define FLASH_SECTOR_242    FLASH_SECTOR(242) // 0x1E400
00250 #define FLASH_SECTOR_243    FLASH_SECTOR(243) // 0x1E600
00251 #define FLASH_SECTOR_244    FLASH_SECTOR(244) // 0x1E800
00252 #define FLASH_SECTOR_245    FLASH_SECTOR(245) // 0x1EA00
00253 #define FLASH_SECTOR_246    FLASH_SECTOR(246) // 0x1EC00
00254 #define FLASH_SECTOR_247    FLASH_SECTOR(247) // 0x1EE00
00255 #define FLASH_SECTOR_248    FLASH_SECTOR(248) // 0x1F000
00256 #define FLASH_SECTOR_249    FLASH_SECTOR(249) // 0x1F200
00257 #define FLASH_SECTOR_250    FLASH_SECTOR(250) // 0x1F400
00258 #define FLASH_SECTOR_251    FLASH_SECTOR(251) // 0x1F600
00259 #define FLASH_SECTOR_252    FLASH_SECTOR(252) // 0x1F800
00260 #define FLASH_SECTOR_253    FLASH_SECTOR(253) // 0x1FA00
00261 #define FLASH_SECTOR_254    FLASH_SECTOR(254) // 0x1FC00
00262 #define FLASH_SECTOR_255    FLASH_SECTOR(255) // 0x1FE00
00263
00264 #define FLASH_SECTOR_END_128K FLASH_SECTOR_255 // 芯片容量为128k的最后一个扇区
00265 #define FLASH_SECTOR_END_64K  FLASH_SECTOR_127 // 芯片容量为64k的最后一个扇区
00266 #define FLASH_SECTOR_END_32K  FLASH_SECTOR_63  // 芯片容量为32k的最后一个扇区
00267
00274 void FLASH_Write(uint32_t adr, uint8_t *data, uint32_t len);
00275
00283 void FLASH_Read(uint32_t adr, uint8_t *data, uint32_t len);
00284
00290 void FLASH_Erase(uint32_t adr);
00291
00299 void FLASH_WriteNvr(uint32_t adr, uint8_t *data, uint32_t len);
00300
00307 void FLASH_EraseNvr(uint32_t adr);
00315 void FLASH_ReadNvr(uint32_t adr, uint8_t *data, uint32_t len);
00316
00317 #endif

```

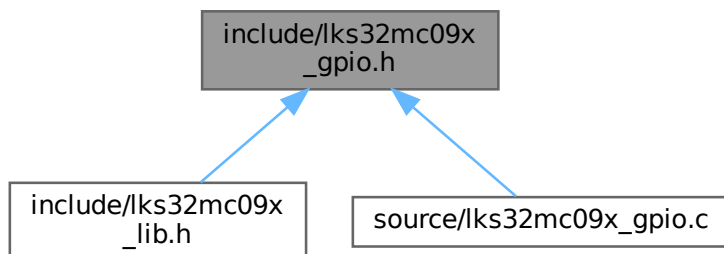
3.21 include/lks32mc09x_gpio.h 文件参考

```
#include "lks32mc09x_reg.h"
```

lks32mc09x_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:



结构体

- struct [GPIO_InitTypeDef](#)
GPIO功能配置结构体句柄

宏定义

- #define [GPIO_AF_AFE](#) 0
GPIO复用 模拟
- #define [GPIO_AF_GPIO](#) 0

- GPIO复用 GPIO
- #define GPIO_AF_CMP_OUT 1
 - GPIO复用 比较器输出
- #define GPIO_AF_CLKO 1
 - GPIO复用 时钟输出
- #define GPIO_AF_HALL 2
 - GPIO复用 霍尔信号输入
- #define GPIO_AF_MCPWM 3
 - GPIO复用 MCPWM输出
- #define GPIO_AF_UART 4
 - GPIO复用 UART
- #define GPIO_AF_SPI 5
 - GPIO复用 SPI
- #define GPIO_AF_I2C 6
 - GPIO复用 I2C
- #define GPIO_AF_TIMER0 7
 - GPIO复用 TIMER0
- #define GPIO_AF_TIMER1 7
 - GPIO复用 TIMER1
- #define GPIO_AF_TIMER2 8
 - GPIO复用 TIMER2
- #define GPIO_AF_QEP0 8
 - GPIO复用 TIMER2
- #define GPIO_AF_ADC0_TRIGGER 9
 - GPIO复用 ADC0触发信号
- #define GPIO_AF_ADC1_TRIGGER 9
 - GPIO复用 ADC1触发信号
- #define GPIO_AF_CAN 10
 - GPIO复用 CAN
- #define GPIO_AF_SIF 11
 - GPIO复用 SIF
- #define GPIO_AF_CL 12
 - GPIO复用 CL CL0可配置逻辑
- #define GPIO_AF_CLU 12
 - GPIO复用 CLU CL0可配置逻辑
- #define AF0_GPIO 0
 - GPIO复用 GPIO
- #define AF1_SYSAF 1
 - GPIO复用 SYS_AF、CMP、CLOCK等数字信号输出功能
- #define AF2_HALL 2
 - GPIO复用 HALL功能
- #define AF3_MCPWM 3
 - GPIO复用 MCPWM功能
- #define AF4_UART 4
 - GPIO复用 UART功能
- #define AF5_SPI 5
 - GPIO复用 SPI功能
- #define AF6_I2C 6
 - GPIO复用 I2C功能
- #define AF7_TIMER01 7
 - GPIO复用 Timer0、Timer1功能

- #define AF8_TIMER23 8
GPIO复用 Timer2、Timer3、QEPO、QEP1功能
- #define AF9_DEBUG 9
GPIO复用 ADC trigger debug功能
- #define AF10_CAN 10
GPIO复用 CAN功能
- #define AF11_SIF 11
GPIO复用 SIF功能
- #define AF12_CL 12
GPIO复用 CL功能
- #define GPIO_Pin_NONE 0
GPIO pin脚定义，无pin脚
- #define GPIO_Pin_0 BIT0
GPIO pin脚定义，pin_0脚
- #define GPIO_Pin_1 BIT1
GPIO pin脚定义，pin_1脚
- #define GPIO_Pin_2 BIT2
GPIO pin脚定义，pin_2脚
- #define GPIO_Pin_3 BIT3
GPIO pin脚定义，pin_3脚
- #define GPIO_Pin_4 BIT4
GPIO pin脚定义，pin_4脚
- #define GPIO_Pin_5 BIT5
GPIO pin脚定义，pin_5脚
- #define GPIO_Pin_6 BIT6
GPIO pin脚定义，pin_6脚
- #define GPIO_Pin_7 BIT7
GPIO pin脚定义，pin_7脚
- #define GPIO_Pin_8 BIT8
GPIO pin脚定义，pin_8脚
- #define GPIO_Pin_9 BIT9
GPIO pin脚定义，pin_9脚
- #define GPIO_Pin_10 BIT10
GPIO pin脚定义，pin_10脚
- #define GPIO_Pin_11 BIT11
GPIO pin脚定义，pin_11脚
- #define GPIO_Pin_12 BIT12
GPIO pin脚定义，pin_12脚
- #define GPIO_Pin_13 BIT13
GPIO pin脚定义，pin_13脚
- #define GPIO_Pin_14 BIT14
GPIO pin脚定义，pin_14脚
- #define GPIO_Pin_15 BIT15
GPIO pin脚定义，pin_15脚
- #define GPIO_Pin_ALL (uint32_t)0xFFFF
GPIO pin脚定义，所有pin脚
- #define GPIO_PinSource_0 0
GPIO pin脚定义，pin_0脚
- #define GPIO_PinSource_1 1
GPIO pin脚定义，pin_1脚
- #define GPIO_PinSource_2 2

- GPIO pin脚定义, pin.2脚
- #define GPIO_PinSource_3 3
GPIO pin脚定义, pin.3脚
- #define GPIO_PinSource_4 4
GPIO pin脚定义, pin.4脚
- #define GPIO_PinSource_5 5
GPIO pin脚定义, pin.5脚
- #define GPIO_PinSource_6 6
GPIO pin脚定义, pin.6脚
- #define GPIO_PinSource_7 7
GPIO pin脚定义, pin.7脚
- #define GPIO_PinSource_8 8
GPIO pin脚定义, pin.8脚
- #define GPIO_PinSource_9 9
GPIO pin脚定义, pin.9脚
- #define GPIO_PinSource_10 10
GPIO pin脚定义, pin.10脚
- #define GPIO_PinSource_11 11
GPIO pin脚定义, pin.11脚
- #define GPIO_PinSource_12 12
GPIO pin脚定义, pin.12脚
- #define GPIO_PinSource_13 13
GPIO pin脚定义, pin.13脚
- #define GPIO_PinSource_14 14
GPIO pin脚定义, pin.14脚
- #define GPIO_PinSource_15 15
GPIO pin脚定义, pin.15脚

枚举

- enum GPIO_Mode_TypeDef { GPIO_Mode_IN = 0x00 , GPIO_Mode_OUT = 0x01 , GPIO_Mode_ANA = 0x02 , GPIO_Mode_IO = 0x03 }
GPIO模式选择枚举
- enum GPIO_PuPd_TypeDef { GPIO_PuPd_NOPULL = 0x00 , GPIO_PuPd_UP = 0x01 }
GPIO输出模式选择枚举
- enum GPIO_Bit_Action { GPIO_Bit_RESET = 0 , GPIO_Bit_SET }
GPIO寄存器写入量枚举

函数

- void GPIO_Enable (void)
gpio使能
- void GPIO_Disable (void)
gpio失能
- void GPIO_Reset (void)
gpio模块复位
- void GPIO_Init (GPIO_TypeDef *GPIOx, GPIO_InitTypeDef *GPIO_InitStruct)
初始化GPIO
- void GPIO_StructInit (GPIO_InitTypeDef *GPIO_InitStruct)
初始化GPIO结构体为默认值

- void [GPIO_LockRegister](#) (GPIO_TypeDef *GPIOx)
锁定GPIO寄存器
- void [GPIO_UnlockRegister](#) (GPIO_TypeDef *GPIOx)
解锁GPIO寄存器
- uint8_t [GPIO_ReadInputDataBit](#) (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
读取指定GPIO引脚的输入数据位
- uint32_t [GPIO_ReadInputData](#) (GPIO_TypeDef *GPIOx)
读取GPIO端口的输入数据
- uint8_t [GPIO_ReadOutputDataBit](#) (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
读取指定GPIO引脚的输出数据位
- uint32_t [GPIO_ReadOutputData](#) (GPIO_TypeDef *GPIOx)
读取GPIO端口的输出数据
- void [GPIO_SetBits](#) (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
设置指定的GPIO引脚位
- void [GPIO_ResetBits](#) (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
复位指定的GPIO引脚位
- void [GPIO_Write](#) (GPIO_TypeDef *GPIOx, uint32_t Val)
写入数据到指定的GPIO端口
- void [GPIO_WriteBit](#) (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin, [GPIO_Bit_Action](#) BitVal)
设置或复位指定的GPIO引脚位
- void [GPIO_ToggleBits](#) (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
翻转指定的GPIO引脚位
- void [GPIO_PinAFConfig](#) (GPIO_TypeDef *GPIOx, uint32_t GPIO_PinSource, uint32_t GPIO_AF)
配置GPIO引脚复用功能
- void [GPIO_Config](#) (GPIO_TypeDef *GPIOx, uint32_t GPIO_PinSource, [GPIO_Mode_TypeDef](#) mode, uint32_t GPIO_AF_x)
GPIO配置函数

3.21.1 宏定义说明

3.21.1.1 AF0.GPIO

```
#define AF0_GPIO 0
```

GPIO复用 GPIO

在文件 [lks32mc09x_gpio.h](#) 第 68 行定义.

3.21.1.2 AF10.CAN

```
#define AF10_CAN 10
```

GPIO复用 CAN功能

在文件 [lks32mc09x_gpio.h](#) 第 78 行定义.

3.21.1.3 AF11_SIF

```
#define AF11_SIF 11
```

GPIO复用 SIF功能

在文件 [lks32mc09x_gpio.h](#) 第 79 行定义.

3.21.1.4 AF12_CL

```
#define AF12_CL 12
```

GPIO复用 CL功能

在文件 [lks32mc09x_gpio.h](#) 第 80 行定义.

3.21.1.5 AF1_SYSAF

```
#define AF1_SYSAF 1
```

GPIO复用 SYS_AF、CMP、CLOCK等数字信号输出功能

在文件 [lks32mc09x_gpio.h](#) 第 69 行定义.

3.21.1.6 AF2_HALL

```
#define AF2_HALL 2
```

GPIO复用 HALL功能

在文件 [lks32mc09x_gpio.h](#) 第 70 行定义.

3.21.1.7 AF3_MCPWM

```
#define AF3_MCPWM 3
```

GPIO复用 MCPWM功能

在文件 [lks32mc09x_gpio.h](#) 第 71 行定义.

3.21.1.8 AF4_UART

```
#define AF4_UART 4
```

GPIO复用 UART功能

在文件 [lks32mc09x_gpio.h](#) 第 72 行定义.

3.21.1.9 AF5_SPI

```
#define AF5_SPI 5
```

GPIO复用 SPI功能

在文件 [lks32mc09x_gpio.h](#) 第 73 行定义.

3.21.1.10 AF6_I2C

```
#define AF6_I2C 6
```

GPIO复用 I2C功能

在文件 [lks32mc09x_gpio.h](#) 第 74 行定义.

3.21.1.11 AF7_TIMER01

```
#define AF7_TIMER01 7
```

GPIO复用 Timer0、Timer1功能

在文件 [lks32mc09x_gpio.h](#) 第 75 行定义.

3.21.1.12 AF8_TIMER23

```
#define AF8_TIMER23 8
```

GPIO复用 Timer2、Timer3、QEP0、QEP1功能

在文件 [lks32mc09x_gpio.h](#) 第 76 行定义.

3.21.1.13 AF9_DEBUG

```
#define AF9_DEBUG 9
```

GPIO复用 ADC trigger debug功能

在文件 [lks32mc09x_gpio.h](#) 第 77 行定义.

3.21.1.14 GPIO_AF_ADC0_TRIGGER

```
#define GPIO_AF_ADC0_TRIGGER 9
```

GPIO复用 ADC0触发信号

在文件 [lks32mc09x_gpio.h](#) 第 61 行定义.

3.21.1.15 GPIO_AF_ADC1_TRIGGER

```
#define GPIO_AF_ADC1_TRIGGER 9
```

GPIO复用 ADC1触发信号

在文件 [lks32mc09x_gpio.h](#) 第 62 行定义.

3.21.1.16 GPIO_AF_AFE

```
#define GPIO_AF_AFE 0
```

GPIO复用 模拟

在文件 [lks32mc09x_gpio.h](#) 第 48 行定义.

3.21.1.17 GPIO_AF_CAN

```
#define GPIO_AF_CAN 10
```

GPIO复用 CAN

在文件 [lks32mc09x_gpio.h](#) 第 63 行定义.

3.21.1.18 GPIO_AF_CL

```
#define GPIO_AF_CL 12
```

GPIO复用 CL CL0可配置逻辑

在文件 [lks32mc09x_gpio.h](#) 第 65 行定义.

3.21.1.19 GPIO_AF_CLKO

```
#define GPIO_AF_CLKO 1
```

GPIO复用 时钟输出

在文件 [lks32mc09x_gpio.h](#) 第 51 行定义.

3.21.1.20 GPIO_AF_CLU

```
#define GPIO_AF_CLU 12
```

GPIO复用 CLU CL0可配置逻辑

在文件 [lks32mc09x_gpio.h](#) 第 66 行定义.

3.21.1.21 GPIO_AF_CMP_OUT

```
#define GPIO_AF_CMP_OUT 1
```

GPIO复用 比较器输出

在文件 [lks32mc09x_gpio.h](#) 第 50 行定义.

3.21.1.22 GPIO_AF_GPIO

```
#define GPIO_AF_GPIO 0
```

GPIO复用 GPIO

在文件 [lks32mc09x_gpio.h](#) 第 49 行定义.

3.21.1.23 GPIO_AF_HALL

```
#define GPIO_AF_HALL 2
```

GPIO复用 霍尔信号输入

在文件 [lks32mc09x_gpio.h](#) 第 52 行定义.

3.21.1.24 GPIO_AF_I2C

```
#define GPIO_AF_I2C 6
```

GPIO复用 I2C

在文件 [lks32mc09x_gpio.h](#) 第 56 行定义.

3.21.1.25 GPIO_AF_MCPWM

```
#define GPIO_AF_MCPWM 3
```

GPIO复用 MCPWM输出

在文件 [lks32mc09x_gpio.h](#) 第 53 行定义.

3.21.1.26 GPIO_AF_QEP0

```
#define GPIO_AF_QEP0 8
```

GPIO复用 TIMER2

在文件 [lks32mc09x_gpio.h](#) 第 60 行定义.

3.21.1.27 GPIO_AF_SIF

```
#define GPIO_AF_SIF 11
```

GPIO复用 SIF

在文件 [lks32mc09x_gpio.h](#) 第 64 行定义.

3.21.1.28 GPIO_AF_SPI

```
#define GPIO_AF_SPI 5
```

GPIO复用 SPI

在文件 [lks32mc09x_gpio.h](#) 第 55 行定义.

3.21.1.29 GPIO_AF_TIMER0

```
#define GPIO_AF_TIMER0 7
```

GPIO复用 TIMER0

在文件 [lks32mc09x_gpio.h](#) 第 57 行定义.

3.21.1.30 GPIO_AF_TIMER1

```
#define GPIO_AF_TIMER1 7
```

GPIO复用 TIMER1

在文件 [lks32mc09x_gpio.h](#) 第 58 行定义.

3.21.1.31 GPIO_AF_TIMER2

```
#define GPIO_AF_TIMER2 8
```

GPIO复用 TIMER2

在文件 [lks32mc09x_gpio.h](#) 第 59 行定义.

3.21.1.32 GPIO_AF_UART

```
#define GPIO_AF_UART 4
```

GPIO复用 UART

在文件 [lks32mc09x_gpio.h](#) 第 54 行定义.

3.21.1.33 GPIO_Pin_0

```
#define GPIO_Pin_0 BIT0
```

GPIO pin脚定义，pin_0脚

在文件 [lks32mc09x_gpio.h](#) 第 84 行定义.

3.21.1.34 GPIO_Pin_1

```
#define GPIO_Pin_1 BIT1
```

GPIO pin脚定义，pin_1脚

在文件 [lks32mc09x_gpio.h](#) 第 85 行定义.

3.21.1.35 GPIO_Pin_10

```
#define GPIO_Pin_10 BIT10
```

GPIO pin脚定义，pin_10脚

在文件 [lks32mc09x_gpio.h](#) 第 94 行定义.

3.21.1.36 GPIO_Pin_11

```
#define GPIO_Pin_11 BIT11
```

GPIO pin脚定义，pin_11脚

在文件 [lks32mc09x_gpio.h](#) 第 95 行定义.

3.21.1.37 GPIO_Pin_12

```
#define GPIO_Pin_12 BIT12
```

GPIO pin脚定义，pin_12脚

在文件 [lks32mc09x_gpio.h](#) 第 96 行定义.

3.21.1.38 GPIO_Pin_13

```
#define GPIO_Pin_13 BIT13
```

GPIO pin脚定义，pin_13脚

在文件 [lks32mc09x_gpio.h](#) 第 97 行定义.

3.21.1.39 GPIO_Pin_14

```
#define GPIO_Pin_14 BIT14
```

GPIO pin脚定义，pin.14脚

在文件 [lks32mc09x_gpio.h](#) 第 98 行定义.

3.21.1.40 GPIO_Pin_15

```
#define GPIO_Pin_15 BIT15
```

GPIO pin脚定义，pin.15脚

在文件 [lks32mc09x_gpio.h](#) 第 99 行定义.

3.21.1.41 GPIO_Pin_2

```
#define GPIO_Pin_2 BIT2
```

GPIO pin脚定义，pin.2脚

在文件 [lks32mc09x_gpio.h](#) 第 86 行定义.

3.21.1.42 GPIO_Pin_3

```
#define GPIO_Pin_3 BIT3
```

GPIO pin脚定义，pin.3脚

在文件 [lks32mc09x_gpio.h](#) 第 87 行定义.

3.21.1.43 GPIO_Pin_4

```
#define GPIO_Pin_4 BIT4
```

GPIO pin脚定义，pin.4脚

在文件 [lks32mc09x_gpio.h](#) 第 88 行定义.

3.21.1.44 GPIO_Pin_5

```
#define GPIO_Pin_5 BIT5
```

GPIO pin脚定义，pin.5脚

在文件 [lks32mc09x_gpio.h](#) 第 89 行定义.

3.21.1.45 GPIO_Pin_6

```
#define GPIO_Pin_6 BIT6
```

GPIO pin脚定义，pin.6脚

在文件 [lks32mc09x_gpio.h](#) 第 90 行定义.

3.21.1.46 GPIO_Pin_7

```
#define GPIO_Pin_7 BIT7
```

GPIO pin脚定义，pin.7脚

在文件 [lks32mc09x_gpio.h](#) 第 91 行定义.

3.21.1.47 GPIO_Pin_8

```
#define GPIO_Pin_8 BIT8
```

GPIO pin脚定义，pin.8脚

在文件 [lks32mc09x_gpio.h](#) 第 92 行定义.

3.21.1.48 GPIO_Pin_9

```
#define GPIO_Pin_9 BIT9
```

GPIO pin脚定义，pin.9脚

在文件 [lks32mc09x_gpio.h](#) 第 93 行定义.

3.21.1.49 GPIO_Pin_ALL

```
#define GPIO_Pin_ALL (uint32_t)0xFFFF
```

GPIO pin脚定义，所有pin脚

在文件 [lks32mc09x_gpio.h](#) 第 100 行定义.

3.21.1.50 GPIO_Pin_NONE

```
#define GPIO_Pin_NONE 0
```

GPIO pin脚定义，无pin脚

在文件 [lks32mc09x_gpio.h](#) 第 83 行定义.

3.21.1.51 GPIO_PinSource_0

```
#define GPIO_PinSource_0 0
```

GPIO pin脚定义，pin_0脚

在文件 [lks32mc09x_gpio.h](#) 第 103 行定义.

3.21.1.52 GPIO_PinSource_1

```
#define GPIO_PinSource_1 1
```

GPIO pin脚定义，pin_1脚

在文件 [lks32mc09x_gpio.h](#) 第 104 行定义.

3.21.1.53 GPIO_PinSource_10

```
#define GPIO_PinSource_10 10
```

GPIO pin脚定义，pin_10脚

在文件 [lks32mc09x_gpio.h](#) 第 113 行定义.

3.21.1.54 GPIO_PinSource_11

```
#define GPIO_PinSource_11 11
```

GPIO pin脚定义，pin_11脚

在文件 [lks32mc09x_gpio.h](#) 第 114 行定义.

3.21.1.55 GPIO_PinSource_12

```
#define GPIO_PinSource_12 12
```

GPIO pin脚定义，pin_12脚

在文件 [lks32mc09x_gpio.h](#) 第 115 行定义.

3.21.1.56 GPIO_PinSource_13

```
#define GPIO_PinSource_13 13
```

GPIO pin脚定义，pin_13脚

在文件 [lks32mc09x_gpio.h](#) 第 116 行定义.

3.21.1.57 GPIO_PinSource_14

```
#define GPIO_PinSource_14 14
```

GPIO pin脚定义，pin.14脚

在文件 [lks32mc09x_gpio.h](#) 第 117 行定义.

3.21.1.58 GPIO_PinSource_15

```
#define GPIO_PinSource_15 15
```

GPIO pin脚定义，pin.15脚

在文件 [lks32mc09x_gpio.h](#) 第 118 行定义.

3.21.1.59 GPIO_PinSource_2

```
#define GPIO_PinSource_2 2
```

GPIO pin脚定义，pin.2脚

在文件 [lks32mc09x_gpio.h](#) 第 105 行定义.

3.21.1.60 GPIO_PinSource_3

```
#define GPIO_PinSource_3 3
```

GPIO pin脚定义，pin.3脚

在文件 [lks32mc09x_gpio.h](#) 第 106 行定义.

3.21.1.61 GPIO_PinSource_4

```
#define GPIO_PinSource_4 4
```

GPIO pin脚定义，pin.4脚

在文件 [lks32mc09x_gpio.h](#) 第 107 行定义.

3.21.1.62 GPIO_PinSource_5

```
#define GPIO_PinSource_5 5
```

GPIO pin脚定义，pin.5脚

在文件 [lks32mc09x_gpio.h](#) 第 108 行定义.

3.21.1.63 GPIO_PinSource_6

```
#define GPIO_PinSource_6 6
```

GPIO pin脚定义，pin.6脚

在文件 [lks32mc09x_gpio.h](#) 第 109 行定义.

3.21.1.64 GPIO_PinSource_7

```
#define GPIO_PinSource_7 7
```

GPIO pin脚定义，pin.7脚

在文件 [lks32mc09x_gpio.h](#) 第 110 行定义.

3.21.1.65 GPIO_PinSource_8

```
#define GPIO_PinSource_8 8
```

GPIO pin脚定义，pin.8脚

在文件 [lks32mc09x_gpio.h](#) 第 111 行定义.

3.21.1.66 GPIO_PinSource_9

```
#define GPIO_PinSource_9 9
```

GPIO pin脚定义，pin.9脚

在文件 [lks32mc09x_gpio.h](#) 第 112 行定义.

3.21.2 枚举类型说明

3.21.2.1 GPIO_Bit_Action

```
enum GPIO_Bit_Action
```

GPIO寄存器写入量枚举

枚举值

GPIO_Bit_RESET	
GPIO_Bit_SET	

在文件 [lks32mc09x_gpio.h](#) 第 29 行定义.

3.21.2.2 GPIO_Mode_TypeDef

enum GPIO_Mode_TypeDef

GPIO模式选择枚举

枚举值

GPIO_Mode_IN	数字输入模式
GPIO_Mode_OUT	数字输出模式
GPIO_Mode_ANA	模拟输入
GPIO_Mode_IO	输入输出模式，通常会同时打开开漏从而作为线与逻辑使用

在文件 [lks32mc09x_gpio.h](#) 第 9 行定义.

3.21.2.3 GPIO_PuPd_TypeDef

enum GPIO_PuPd_TypeDef

GPIO输出模式选择枚举

枚举值

GPIO_PuPd_NOPULL	高阻态
GPIO_PuPd_UP	上拉

在文件 [lks32mc09x_gpio.h](#) 第 20 行定义.

3.21.3 函数说明

3.21.3.1 GPIO_Config()

```
void GPIO_Config (
    GPIO_TypeDef * GPIOx,
    uint32_t GPIO_PinSource,
    GPIO_Mode_TypeDef mode,
    uint32_t GPIO_AF_x )
```

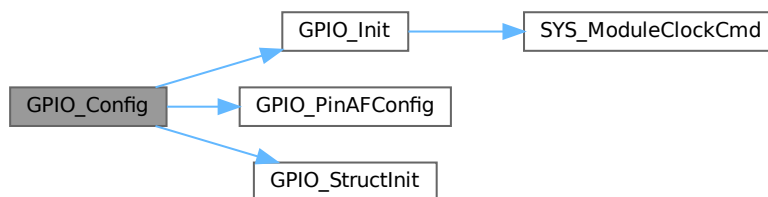
GPIO配置函数

参数

GPIOx	GPIO端口
GPIO_PinSource	GPIO引脚
mode	GPIO模式
GPIO_AF_x	GPIO复用功能选择

在文件 [lks32mc09x_gpio.c](#) 第 264 行定义.

函数调用图:



3.21.3.2 GPIO_Disable()

```
void GPIO_Disable (  
    void )
```

gpio失能

在文件 [lks32mc09x_gpio.c](#) 第 14 行定义.

函数调用图:



3.21.3.3 GPIO_Enable()

```
void GPIO_Enable (  
    void )
```

gpio使能

在文件 [lks32mc09x_gpio.c](#) 第 7 行定义.

函数调用图:



3.21.3.4 GPIO_Init()

```
void GPIO_Init (
    GPIO_TypeDef * GPIOx,
    GPIO_InitTypeDef * GPIO_InitStruct )
```

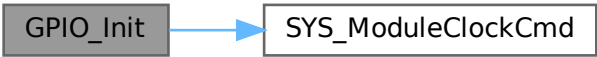
初始化GPIO

参数

GPIOx	GPIO模块指针
GPIO_InitStruct	指向包含初始化参数的GPIO_InitTypeDef结构体

在文件 [lks32mc09x_gpio.c](#) 第 43 行定义.

函数调用图:



这是这个函数的调用关系图:



3.21.3.5 GPIO_LockRegister()

```
void GPIO_LockRegister (
    GPIO_TypeDef * GPIOx )
```

锁定GPIO寄存器

参数

GPIOx	GPIO模块指针

在文件 [lks32mc09x_gpio.c](#) 第 105 行定义.

3.21.3.6 GPIO_PinAFConfig()

```
void GPIO_PinAFConfig (
    GPIO_TypeDef * GPIOx,
    uint32_t GPIO_PinSource,
    uint32_t GPIO_AF )
```

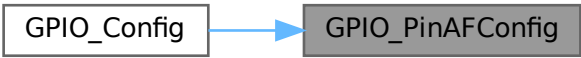
配置GPIO引脚复用功能

参数

<i>GPIOx</i>	GPIO模块指针
<i>GPIO_PinSource</i>	引脚源
<i>GPIO_AF</i>	复用功能选择

在文件 [lks32mc09x_gpio.c](#) 第 223 行定义.

这是这个函数的调用关系图:



3.21.3.7 GPIO_ReadInputData()

```
uint32_t GPIO_ReadInputData (
    GPIO_TypeDef * GPIOx )
```

读取GPIO端口的输入数据

参数

<i>GPIOx</i>	GPIO模块指针

返回

GPIO端口的输入数据

在文件 [lks32mc09x_gpio.c](#) 第 143 行定义.

3.21.3.8 GPIO_ReadInputDataBit()

```
uint8_t GPIO_ReadInputDataBit (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

读取指定GPIO引脚的输入数据位

参数

GPIOx	GPIO模块指针
GPIO_Pin	要读取的引脚

返回

引脚的输入数据位状态

在文件 [lks32mc09x_gpio.c](#) 第 133 行定义.

3.21.3.9 GPIO_ReadOutputData()

```
uint32_t GPIO_ReadOutputData (
    GPIO_TypeDef * GPIOx )
```

读取GPIO端口的输出数据

参数

GPIOx	GPIO模块指针

返回

GPIO端口的输出数据

在文件 [lks32mc09x_gpio.c](#) 第 164 行定义.

3.21.3.10 GPIO_ReadOutputDataBit()

```
uint8_t GPIO_ReadOutputDataBit (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

读取指定GPIO引脚的输出数据位

参数

GPIOx	GPIO模块指针
GPIO_Pin	要读取的引脚

返回

引脚的输出数据位状态

在文件 [lks32mc09x_gpio.c](#) 第 154 行定义.

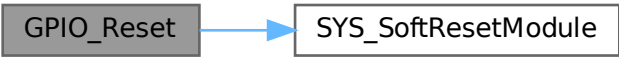
3.21.3.11 GPIO_Reset()

```
void GPIO_Reset (
    void )
```

gpio模块复位

在文件 [lks32mc09x_gpio.c](#) 第 21 行定义.

函数调用图:



3.21.3.12 GPIO_ResetBits()

```
void GPIO_ResetBits (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

复位指定的GPIO引脚位

参数

<i>GPIOx</i>	GPIO模块指针
<i>GPIO_Pin</i>	要复位的引脚

在文件 [lks32mc09x_gpio.c](#) 第 184 行定义.

3.21.3.13 GPIO_SetBits()

```
void GPIO_SetBits (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

设置指定的GPIO引脚位

参数

<i>GPIOx</i>	GPIO模块指针
<i>GPIO.Pin</i>	要设置的引脚

在文件 [lks32mc09x_gpio.c](#) 第 174 行定义.

3.21.3.14 GPIO_StructInit()

```
void GPIO_StructInit (
    GPIO_InitTypeDef * GPIO_InitStruct )
```

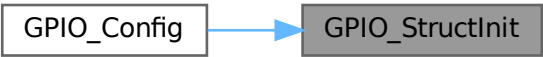
初始化GPIO结构体为默认值

参数

<i>GPIO_InitStruct</i>	指向要初始化的GPIO_InitTypeDef结构体

在文件 [lks32mc09x_gpio.c](#) 第 29 行定义.

这是这个函数的调用关系图:



3.21.3.15 GPIO_ToggleBits()

```
void GPIO_ToggleBits (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

翻转指定的GPIO引脚位

参数

<i>GPIOx</i>	GPIO模块指针
<i>GPIO.Pin</i>	要翻转的引脚

在文件 [lks32mc09x_gpio.c](#) 第 252 行定义.

3.21.3.16 GPIO_UnlockRegister()

```
void GPIO_UnlockRegister (
    GPIO_TypeDef * GPIOx )
```

解锁GPIO寄存器

参数

<i>GPIOx</i>	GPIO模块指针

在文件 [lks32mc09x_gpio.c](#) 第 118 行定义.

3.21.3.17 GPIO_Write()

```
void GPIO_Write (
    GPIO_TypeDef * GPIOx,
    uint32_t Val )
```

写入数据到指定的GPIO端口

参数

<i>GPIOx</i>	GPIO模块指针
<i>Val</i>	要写入的数据

在文件 [lks32mc09x_gpio.c](#) 第 194 行定义.

3.21.3.18 GPIO_WriteBit()

```
void GPIO_WriteBit (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin,
    GPIO_Bit_Action BitVal )
```

设置或复位指定的GPIO引脚位

参数

<i>GPIOx</i>	GPIO模块指针
<i>GPIO_Pin</i>	要操作的引脚
<i>BitVal</i>	要执行的操作: GPIO_Bit_SET 或 GPIO_Bit_RESET

在文件 [lks32mc09x_gpio.c](#) 第 205 行定义.

3.22 lks32mc09x_gpio.h

浏览该文件的文档.

```

00001 #ifndef __LKS32MC09X_GPIO_H__
00002 #define __LKS32MC09X_GPIO_H__
00003
00004 #include "lks32mc09x_reg.h"
00005
00009 typedef enum
00010 {
00011     GPIO_Mode_IN   = 0x00,
00012     GPIO_Mode_OUT  = 0x01,
00013     GPIO_Mode_ANA  = 0x02,
00014     GPIO_Mode_IO   = 0x03,
00015 } GPIO_Mode_TypeDef;
00016
00020 typedef enum
00021 {
00022     GPIO_PuPd_NOPULL = 0x00,
00023     GPIO_PuPd_UP     = 0x01,
00024 } GPIO_PuPd_TypeDef;
00025
00029 typedef enum
00030 {
00031     GPIO_Bit_RESET = 0,
00032     GPIO_Bit_SET   = 1,
00033 } GPIO_Bit_Action;
00034
00038 typedef struct
00039 {
00040     uint32_t GPIO_Pin;
00041     GPIO_Mode_TypeDef GPIO_Mode;
00042     GPIO_PuPd_TypeDef GPIO_PuPd;
00043     uint32_t GPIO_PODENA;
00044     uint32_t GPIO_PFLT;
00045 } GPIO_InitTypeDef;
00046
00047 // GPIO复用功能选择
00048 #define GPIO_AF_AFE          0
00049 #define GPIO_AF_GPIO        0
00050 #define GPIO_AF_CMP_OUT     1
00051 #define GPIO_AF_CLKO        1
00052 #define GPIO_AF_HALL        2
00053 #define GPIO_AF_MCPWM       3
00054 #define GPIO_AF_UART        4
00055 #define GPIO_AF_SPI         5
00056 #define GPIO_AF_I2C         6
00057 #define GPIO_AF_TIMER0      7
00058 #define GPIO_AF_TIMER1      7
00059 #define GPIO_AF_TIMER2      8
00060 #define GPIO_AF_QEP0        8
00061 #define GPIO_AF_ADC0_TRIGGER 9
00062 #define GPIO_AF_ADC1_TRIGGER 9
00063 #define GPIO_AF_CAN         10
00064 #define GPIO_AF_SIF         11
00065 #define GPIO_AF_CL          12
00066 #define GPIO_AF_CLU         12
00067 // 兼容之前的写法
00068 #define AF0_GPIO            0
00069 #define AF1_SYSAF          1
00070 #define AF2_HALL           2
00071 #define AF3_MCPWM          3
00072 #define AF4_UART           4
00073 #define AF5_SPI            5
00074 #define AF6_I2C            6
00075 #define AF7_TIMER0         7
00076 #define AF8_TIMER23        8
00077 #define AF9_DEBUG          9
00078 #define AF10_CAN           10
00079 #define AF11_SIF           11
00080 #define AF12_CL            12
00081
00082 /* GPIO引脚定义 */
00083 #define GPIO_Pin_NONE      0
00084 #define GPIO_Pin_0         BIT0
00085 #define GPIO_Pin_1         BIT1
00086 #define GPIO_Pin_2         BIT2
00087 #define GPIO_Pin_3         BIT3
00088 #define GPIO_Pin_4         BIT4
00089 #define GPIO_Pin_5         BIT5
00090 #define GPIO_Pin_6         BIT6
00091 #define GPIO_Pin_7         BIT7
00092 #define GPIO_Pin_8         BIT8
00093 #define GPIO_Pin_9         BIT9
00094 #define GPIO_Pin_10        BIT10

```

```

00095 #define GPIO_Pin_11    BIT11
00096 #define GPIO_Pin_12    BIT12
00097 #define GPIO_Pin_13    BIT13
00098 #define GPIO_Pin_14    BIT14
00099 #define GPIO_Pin_15    BIT15
00100 #define GPIO_Pin_ALL    (uint32_t)0xFFFF
00101
00102 /* GPIO引脚源定义 */
00103 #define GPIO_PinSource_0  0
00104 #define GPIO_PinSource_1  1
00105 #define GPIO_PinSource_2  2
00106 #define GPIO_PinSource_3  3
00107 #define GPIO_PinSource_4  4
00108 #define GPIO_PinSource_5  5
00109 #define GPIO_PinSource_6  6
00110 #define GPIO_PinSource_7  7
00111 #define GPIO_PinSource_8  8
00112 #define GPIO_PinSource_9  9
00113 #define GPIO_PinSource_10 10
00114 #define GPIO_PinSource_11 11
00115 #define GPIO_PinSource_12 12
00116 #define GPIO_PinSource_13 13
00117 #define GPIO_PinSource_14 14
00118 #define GPIO_PinSource_15 15
00119
00120 void GPIO_Enable(void);
00121 void GPIO_Disable(void);
00122 void GPIO_Reset(void);
00123 /* 初始化、配置函数 */
00124 void GPIO_Init(GPIO_TypeDef *GPIOx, GPIO_InitTypeDef *GPIO_InitStruct);
00125 void GPIO_StructInit(GPIO_InitTypeDef *GPIO_InitStruct);
00126 void GPIO_LockRegister(GPIO_TypeDef *GPIOx);
00127 void GPIO_UnlockRegister(GPIO_TypeDef *GPIOx);
00128
00129 /* GPIO读写函数 */
00130 uint8_t GPIO_ReadInputDataBit(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin);
00131 uint32_t GPIO_ReadInputData(GPIO_TypeDef *GPIOx);
00132 uint8_t GPIO_ReadOutputDataBit(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin);
00133 uint32_t GPIO_ReadOutputData(GPIO_TypeDef *GPIOx);
00134 void GPIO_SetBits(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin);
00135 void GPIO_ResetBits(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin);
00136 void GPIO_Write(GPIO_TypeDef *GPIOx, uint32_t Val);
00137 void GPIO_WriteBit(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin, GPIO_BitAction BitVal);
00138 void GPIO_ToggleBits(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin);
00139 /* GPIO功能选择函数 */
00140 void GPIO_PinAFConfig(GPIO_TypeDef *GPIOx, uint32_t GPIO_PinSource, uint32_t GPIO_AF);
00148 void GPIO_Config(GPIO_TypeDef *GPIOx, uint32_t GPIO_PinSource, GPIO_ModeTypeDef mode, uint32_t
    GPIO_AF_x);
00149
00150 #endif

```

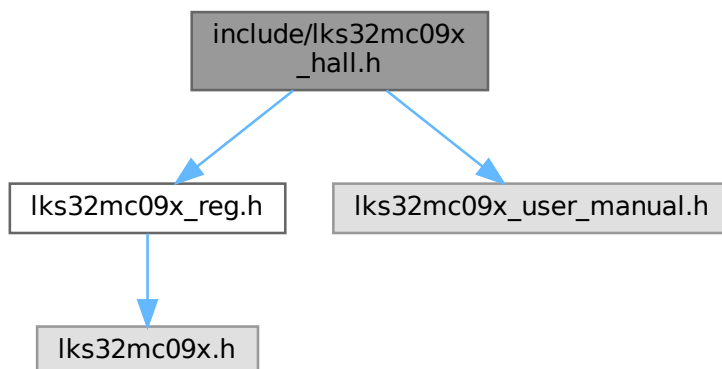
3.23 include/lks32mc09x_hall.h 文件参考

```

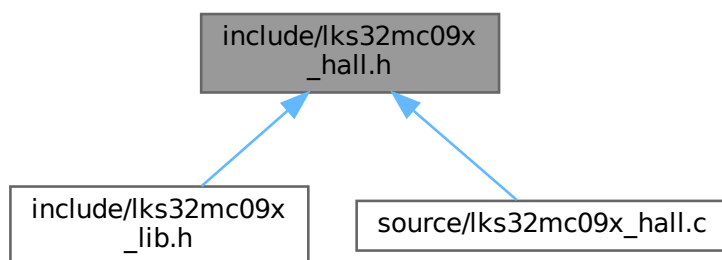
#include "lks32mc09x_reg.h"
#include "lks32mc09x_user_manual.h"

```


lks32mc09x_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:



结构体

- struct [HALL_InitTypeDef](#)
HALL初始化结构体句柄，初始化时请定义该句柄，并用其它地址来传参

宏定义

- #define [HALL_CLK_DIV1](#) ((uint32_t)0x00)
HALL时钟为系统主时钟1分频
- #define [HALL_CLK_DIV2](#) ((uint32_t)0x01)
HALL时钟为系统主时钟2分频
- #define [HALL_CLK_DIV4](#) ((uint32_t)0x02)
HALL时钟为系统主时钟4分频

- `#define HALL_CLK_DIV8 ((uint32_t)0x03)`
HALL时钟为系统主时钟8分频
- `#define HALL_CAPTURE_EVENT ((uint32_t)HALL0_CFG_CHG_IE_BIT)`
HALL中断使能和标志定义，HALL信号变化中断
- `#define HALL_OVERFLOW_EVENT ((uint32_t)HALL0_CFG_OV_IE_BIT)`
HALL中断使能和标志定义，HALL计数溢出中断

函数

- `void HALL_StructInit (HALL_InitTypeDef *HALL_InitStruct)`
初始化HALL结构体
- `void HALL_Init (HALL_InitTypeDef *HALL_InitStruct)`
初始化HALL模块
- `void HALL_Cmd (uint8_t state)`
使能或失能HALL模块
- `uint32_t HALL_GetFilterValue (void)`
读取滤波后的HALL值
- `uint32_t HALL_GetCaptureValue (void)`
读取未滤波的HALL值
- `uint32_t HALL_GetCount (void)`
读取HALL计数值
- `uint32_t HALL_IsCaptureEvent (void)`
判断是否发生捕获事件
- `uint32_t HALL_IsOverflowEvent (void)`
判断是否发生溢出事件
- `uint32_t HALL_GetFlag (uint32_t tempFlag)`
获取HALL事件标志
- `void HALL_Clear_IRQ (uint32_t tempFlag)`
清除HALL中断标志位
- `uint32_t HALL_GetIRQFlag (uint32_t tempFlag)`
读取HALL中断标志位
- `void HALL_SoftTrigger (void)`
软件触发HALL信号变化中断

3.23.1 宏定义说明

3.23.1.1 HALL_CAPTURE_EVENT

```
#define HALL_CAPTURE_EVENT ((uint32_t)HALL0_CFG_CHG_IE_BIT)
```

HALL中断使能和标志定义，HALL信号变化中断

在文件 [lks32mc09x_hall.h](#) 第 27 行定义。

3.23.1.2 HALL_CLK_DIV1

```
#define HALL_CLK_DIV1 ((uint32_t)0x00)
```

HALL时钟为系统主时钟1分频

在文件 [lks32mc09x_hall.h](#) 第 22 行定义。

3.23.1.3 HALL_CLK_DIV2

```
#define HALL_CLK_DIV2 ((uint32_t)0x01)
```

HALL时钟为系统主时钟2分频

在文件 [lks32mc09x_hall.h](#) 第 23 行定义.

3.23.1.4 HALL_CLK_DIV4

```
#define HALL_CLK_DIV4 ((uint32_t)0x02)
```

HALL时钟为系统主时钟4分频

在文件 [lks32mc09x_hall.h](#) 第 24 行定义.

3.23.1.5 HALL_CLK_DIV8

```
#define HALL_CLK_DIV8 ((uint32_t)0x03)
```

HALL时钟为系统主时钟8分频

在文件 [lks32mc09x_hall.h](#) 第 25 行定义.

3.23.1.6 HALL_OVERFLOW_EVENT

```
#define HALL_OVERFLOW_EVENT ((uint32_t)HALL0_CFG_OV_IE_BIT)
```

HALL中断使能和标志定义, HALL计数溢出中断

在文件 [lks32mc09x_hall.h](#) 第 28 行定义.

3.23.2 函数说明

3.23.2.1 HALL_Clear_IRQ()

```
void HALL_Clear_IRQ (
    uint32_t tempFlag )
```

清除HALL中断标志位

参数

<i>tempFlag</i>	要清除的标志位

在文件 [lks32mc09x_hall.c](#) 第 120 行定义.

3.23.2.2 HALL_Cmd()

```
void HALL_Cmd (
    uint8_t state )
```

使能或失能HALL模块

参数

state	功能状态，ENABLE或DISABLE

在文件 [lks32mc09x_hall.c](#) 第 49 行定义.

3.23.2.3 HALL_GetCaptureValue()

```
uint32_t HALL_GetCaptureValue (
    void )
```

读取未滤波的HALL值

返回

未滤波的HALL值

在文件 [lks32mc09x_hall.c](#) 第 74 行定义.

3.23.2.4 HALL_GetCount()

```
uint32_t HALL_GetCount (
    void )
```

读取HALL计数值

返回

HALL计数值

在文件 [lks32mc09x_hall.c](#) 第 83 行定义.

3.23.2.5 HALL_GetFilterValue()

```
uint32_t HALL_GetFilterValue (
    void )
```

读取滤波后的HALL值

返回

滤波后的HALL值

在文件 [lks32mc09x_hall.c](#) 第 65 行定义.

3.23.2.6 HALL_GetFlag()

```
uint32_t HALL_GetFlag (
    uint32_t tempFlag )
```

获取HALL事件标志

参数

<i>tempFlag</i>	要获取的标志位

返回

对应标志位的状态

在文件 [lks32mc09x_hall.c](#) 第 111 行定义.

3.23.2.7 HALL_GetIRQFlag()

```
uint32_t HALL_GetIRQFlag (  
    uint32_t tempFlag )
```

读取HALL中断标志位

参数

<i>tempFlag</i>	要读取的标志位

返回

对应标志位的状态

在文件 [lks32mc09x_hall.c](#) 第 130 行定义.

3.23.2.8 HALL_Init()

```
void HALL_Init (  
    HALL_InitTypeDef * HALL_InitStruct )
```

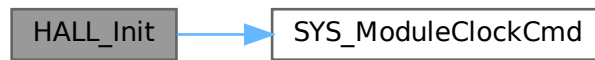
初始化HALL模块

参数

<i>HALL_InitStruct</i>	指向HALL_InitTypeDef结构体的指针

在文件 [lks32mc09x_hall.c](#) 第 21 行定义.

函数调用图:



3.23.2.9 HALL_IsCaptureEvent()

```
uint32_t HALL_IsCaptureEvent (
    void )
```

判断是否发生捕获事件

返回

发生捕获事件返回非零值（True），否则返回0（False）

在文件 [lks32mc09x_hall.c](#) 第 92 行定义.

3.23.2.10 HALL_IsOverFlowEvent()

```
uint32_t HALL_IsOverFlowEvent (
    void )
```

判断是否发生溢出事件

返回

发生溢出事件返回非零值（True），否则返回0（False）

在文件 [lks32mc09x_hall.c](#) 第 101 行定义.

3.23.2.11 HALL_SoftTrigger()

```
void HALL_SoftTrigger (
    void )
```

软件触发HALL信号变化中断

在文件 [lks32mc09x_hall.c](#) 第 138 行定义.

3.23.2.12 HALL_StructInit()

```
void HALL_StructInit (
    HALL_InitTypeDef * HALL_InitStruct )
```

初始化HALL结构体

参数

<code>HALL_InitStruct</code>	指向HALL_InitTypeDef结构体的指针

在文件 `lks32mc09x_hall.c` 第 8 行定义.

3.24 lks32mc09x_hall.h

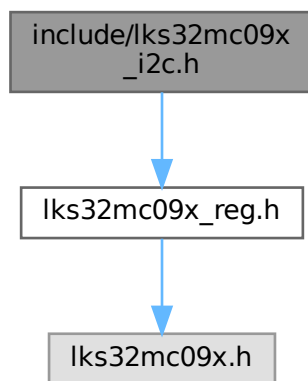
浏览该文件的文档.

```
00001 #ifndef __LKS32MC09X_HALL_H__
00002 #define __LKS32MC09X_HALL_H__
00003
00004 #include "lks32mc09x_reg.h"
00005 #include "lks32mc09x_user_manual.h"
00006
00010 typedef struct
00011 {
00012     uint32_t CountTH;
00013     uint16_t FilterLen;
00014     uint8_t ClockDivision;
00015     uint8_t Filter75_Ena;
00016     uint8_t HALL_Ena;
00017     uint8_t Capture_IRQ_Ena;
00018     uint8_t OverFlow_IRQ_Ena;
00019     uint8_t softIE;
00020 } HALL_InitTypeDef;
00021
00022 #define HALL_CLK_DIV1      ((uint32_t)0x00)
00023 #define HALL_CLK_DIV2      ((uint32_t)0x01)
00024 #define HALL_CLK_DIV4      ((uint32_t)0x02)
00025 #define HALL_CLK_DIV8      ((uint32_t)0x03)
00026
00027 #define HALL_CAPTURE_EVENT ((uint32_t)HALL0_CFG_CHG_IE_BIT)
00028 #define HALL_OVERFLOW_EVENT ((uint32_t)HALL0_CFG_OV_IE_BIT)
00029
00034 void HALL_StructInit(HALL_InitTypeDef *HALL_InitStruct);
00035
00040 void HALL_Init(HALL_InitTypeDef *HALL_InitStruct);
00041
00046 void HALL_Cmd(uint8_t state);
00047
00052 uint32_t HALL_GetFilterValue(void);
00053
00058 uint32_t HALL_GetCaptureValue(void);
00059
00064 uint32_t HALL_GetCount(void);
00065
00070 uint32_t HALL_IsCaptureEvent(void);
00071
00076 uint32_t HALL_IsOverflowEvent(void);
00077
00083 uint32_t HALL_GetFlag(uint32_t tempFlag);
00084
00089 void HALL_Clear_IRQ(uint32_t tempFlag);
00090
00096 uint32_t HALL_GetIRQFlag(uint32_t tempFlag);
00097
00101 void HALL_SoftTrigger(void);
00102
00103 #endif
```

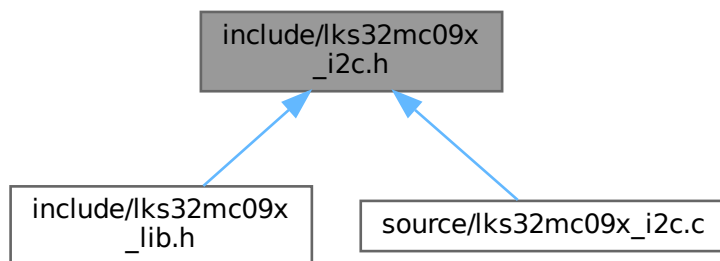
3.25 include/lks32mc09x_i2c.h 文件参考

```
#include "lks32mc09x_reg.h"
```

lks32mc09x_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:



结构体

- struct [I2C_InitTypeDef](#)

宏定义

- #define [I2C_BUS_STATE_STT_ERR](#) BIT7
总线错误
- #define [I2C_BUS_STATE_LOST_ARB](#) BIT6

- 总线仲裁丢失
- #define I2C_BUS_STATE_STOP_EVT BIT5
STOP事件记录
- #define I2C_BUS_STATE_ADDR_DATA BIT3
正在传输地址数据
- #define I2C_BUS_STATE_RX_ACK BIT1
接收到ACK响应
- #define I2C_BUS_STATE_Done BIT0
传输完成
- #define I2C_IF_TC 0
传输完成中断
- #define I2C_IF_BUS_ERR 1
总线错误中断
- #define I2C_IF_STOP 2
STOP事件中断
- #define I2C_IF_NACK 3
NACK事件中断
- #define I2C_IF_ADDR_CMP 4
硬件地址匹配中断

函数

- void I2C_Init (I2C_TypeDef *I2Cx, I2C_InitTypeDef *I2C_InitStruct)
I2C初始化
- void I2C_StructInit (I2C_InitTypeDef *I2C_InitStruct)
I2C配置结构体初始化
- void I2C_ClearIRQFlag (I2C_TypeDef *I2Cx, uint16_t I2C_IF_x)
清除I2C中断标志
- uint8_t I2C_GetIRQFlag (I2C_TypeDef *I2Cx, uint16_t I2C_IF_x)
获取I2C中断标志状态
- uint8_t I2C_GetFlag (I2C_TypeDef *I2Cx, uint16_t I2C_IF_x)
获取I2C状态标志
- void I2C_Enable (I2C_TypeDef *I2Cx)
使能I2C外设
- void I2C_Disable (I2C_TypeDef *I2Cx)
禁用I2C外设
- void I2C_SetBaudRate (I2C_TypeDef *I2Cx, uint32_t BaudRate)
设置I2C波特率
- uint8_t I2C_ReadBusState (I2C_TypeDef *I2Cx, uint16_t state)
读取I2C总线状态
- void I2C_ClearBusState (I2C_TypeDef *I2Cx, uint16_t state)
清除I2C总线状态
- void I2C_SendData (I2C_TypeDef *I2Cx, uint8_t n)
发送数据到I2C总线
- uint8_t I2C_ReadData (I2C_TypeDef *I2Cx)
从I2C总线读取数据
- int I2C_TableSend (I2C_TypeDef *I2Cx, u8 taddr, u8 *data, u8 len)
向从机发送数据（需要在中断里执行I2C_TableIrq）
- int I2C_TableRead (I2C_TypeDef *I2Cx, u8 taddr, u8 *data, u8 len)
向从机读取数据（需要在中断里执行I2C_TableIrq）

- int [I2C_TableState](#) ([I2C_TypeDef](#) *I2Cx)
获取 *I2C.Table* 发送状态
- int [I2C_TableAck](#) ([I2C_TypeDef](#) *I2Cx)
获取 *I2C.Table* 发送后的 *ack* 电平
- void [I2C_TableIrq](#) ([I2C_TypeDef](#) *I2Cx)
I2C.SendTable 和 *I2C.ReadTable* 对应的中断处理函数

3.25.1 宏定义说明

3.25.1.1 I2C_BUS_STATE_ADDR_DATA

```
#define I2C_BUS_STATE_ADDR_DATA BIT3
```

正在传输地址数据

在文件 [lks32mc09x_i2c.h](#) 第 10 行定义.

3.25.1.2 I2C_BUS_STATE_Done

```
#define I2C_BUS_STATE_Done BIT0
```

传输完成

在文件 [lks32mc09x_i2c.h](#) 第 12 行定义.

3.25.1.3 I2C_BUS_STATE_LOST_ARB

```
#define I2C_BUS_STATE_LOST_ARB BIT6
```

总线仲裁丢失

在文件 [lks32mc09x_i2c.h](#) 第 8 行定义.

3.25.1.4 I2C_BUS_STATE_RX_ACK

```
#define I2C_BUS_STATE_RX_ACK BIT1
```

接收到ACK响应

在文件 [lks32mc09x_i2c.h](#) 第 11 行定义.

3.25.1.5 I2C_BUS_STATE_STOP_EVT

```
#define I2C_BUS_STATE_STOP_EVT BIT5
```

STOP事件记录

在文件 [lks32mc09x_i2c.h](#) 第 9 行定义.

3.25.1.6 I2C_BUS_STATE_STT_ERR

```
#define I2C_BUS_STATE_STT_ERR BIT7
```

总线错误

在文件 [lks32mc09x_i2c.h](#) 第 7 行定义.

3.25.1.7 I2C_IF_ADDR_CMP

```
#define I2C_IF_ADDR_CMP 4
```

硬件地址匹配中断

在文件 [lks32mc09x_i2c.h](#) 第 19 行定义.

3.25.1.8 I2C_IF_BUS_ERR

```
#define I2C_IF_BUS_ERR 1
```

总线错误中断

在文件 [lks32mc09x_i2c.h](#) 第 16 行定义.

3.25.1.9 I2C_IF_NACK

```
#define I2C_IF_NACK 3
```

NACK事件中断

在文件 [lks32mc09x_i2c.h](#) 第 18 行定义.

3.25.1.10 I2C_IF_STOP

```
#define I2C_IF_STOP 2
```

STOP事件中断

在文件 [lks32mc09x_i2c.h](#) 第 17 行定义.

3.25.1.11 I2C_IF_TC

```
#define I2C_IF_TC 0
```

传输完成中断

在文件 [lks32mc09x_i2c.h](#) 第 15 行定义.

3.25.2 函数说明

3.25.2.1 I2c_ClearBusState()

```
void I2c_ClearBusState (
    I2C_TypeDef * I2Cx,
    uint16_t state )
```

清除I2C总线状态

参数

<i>I2Cx</i>	I2C外设指针
<i>state</i>	要清除的状态位

在文件 [lks32mc09x_i2c.c](#) 第 207 行定义.

3.25.2.2 I2C_ClearIRQFlag()

```
void I2C_ClearIRQFlag (
    I2C_TypeDef * I2Cx,
    uint16_t I2C_IF_x )
```

清除I2C中断标志

参数

<i>I2Cx</i>	I2C外设指针
<i>I2C_↔IF_x</i>	要清除的中断标志

在文件 [lks32mc09x_i2c.c](#) 第 43 行定义.

3.25.2.3 I2C_Disable()

```
void I2C_Disable (
    I2C_TypeDef * I2Cx )
```

禁用I2C外设

参数

<i>I2Cx</i>	I2C外设指针

在文件 [lks32mc09x_i2c.c](#) 第 161 行定义.

函数调用图:



3.25.2.4 I2C_Enable()

```
void I2C_Enable (
    I2C.TypeDef * I2Cx )
```

使能I2C外设

参数

I2Cx	I2C外设指针

在文件 lks32mc09x_i2c.c 第 149 行定义.

函数调用图:



3.25.2.5 I2C_GetFlag()

```
uint8_t I2C_GetFlag (
    I2C.TypeDef * I2Cx,
    uint16_t I2C_IF_x )
```

获取I2C状态标志

参数

I2Cx	I2C外设指针
I2C_↔IF_x	要获取的状态标志

返回

状态标志状态

在文件 lks32mc09x_i2c.c 第 84 行定义.

这是这个函数的调用关系图:



3.25.2.6 I2C_GetIRQFlag()

```
uint8_t I2C_GetIRQFlag (
    I2C_TypeDef * I2Cx,
    uint16_t I2C_IF_x )
```

获取I2C中断标志状态

参数

I2Cx	I2C外设指针
I2C_↔IF_x	要获取的中断标志

返回

中断标志状态

在文件 [lks32mc09x_i2c.c](#) 第 73 行定义.

函数调用图:



3.25.2.7 I2C_Init()

```
void I2C_Init (
    I2C_TypeDef * I2Cx,
    I2C_InitTypeDef * I2C_InitStruct )
```

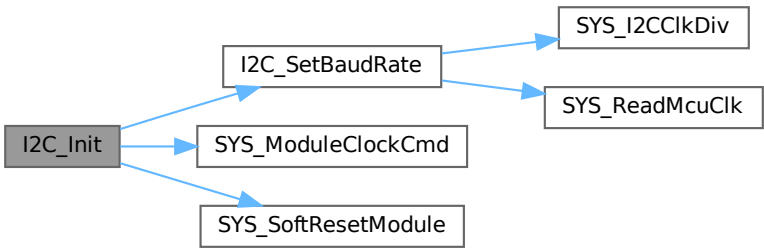
I2C初始化

参数

<i>I2Cx</i>	I2C外设指针
<i>I2C_InitStruct</i>	I2C初始化结构体指针

在文件 [lks32mc09x_i2c.c](#) 第 9 行定义.

函数调用图:



3.25.2.8 I2c_ReadBusState()

```
uint8_t I2c_ReadBusState (
    I2C.TypeDef * I2Cx,
    uint16_t state )
```

读取I2C总线状态

参数

<i>I2Cx</i>	I2C外设指针
<i>state</i>	要读取的状态位

返回

状态位值

在文件 [lks32mc09x_i2c.c](#) 第 190 行定义.

3.25.2.9 I2C_ReadData()

```
uint8_t I2C_ReadData (
    I2C.TypeDef * I2Cx )
```

从I2C总线读取数据

参数

<i>I2Cx</i>	I2C外设指针

返回

读取到的数据

在文件 [lks32mc09x_i2c.c](#) 第 227 行定义.

3.25.2.10 I2C_SendData()

```
void I2C_SendData (
    I2C_TypeDef * I2Cx,
    uint8_t n )
```

发送数据到I2C总线

参数

<i>I2Cx</i>	I2C外设指针
<i>n</i>	要发送的数据

在文件 [lks32mc09x_i2c.c](#) 第 217 行定义.

3.25.2.11 I2C_SetBaudRate()

```
void I2C_SetBaudRate (
    I2C_TypeDef * I2Cx,
    uint32_t BaudRate )
```

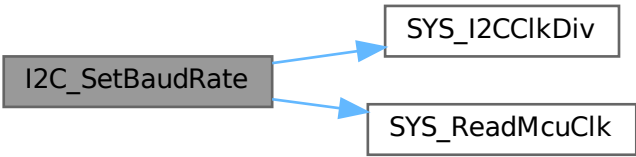
设置I2C波特率

参数

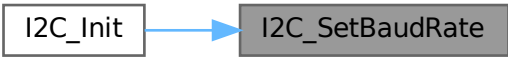
<i>BaudRate</i>	波特率值

在文件 [lks32mc09x_i2c.c](#) 第 173 行定义.

函数调用图:



这是这个函数的调用关系图:



3.25.2.12 I2C_StructInit()

```
void I2C_StructInit (  
    I2C_InitTypeDef * I2C_InitStruct )
```

I2C配置结构体初始化

参数

I2C_InitStruct	I2C初始化结构体指针

在文件 [lks32mc09x_i2c.c](#) 第 29 行定义.

3.25.2.13 I2C_TableAck()

```
int I2C_TableAck (  
    I2C_TypeDef * I2Cx )
```

获取I2C_Table发送后的ack电平

返回

收到的ACK电平

在文件 [lks32mc09x_i2c.c](#) 第 291 行定义.

3.25.2.14 I2C_TableIrq()

```
void I2C_TableIrq (
    I2C_TypeDef * I2Cx )
```

I2C_SendTable和I2C_ReadTable对应的中断处理函数

在文件 [lks32mc09x_i2c.c](#) 第 320 行定义.

3.25.2.15 I2C_TableRead()

```
int I2C_TableRead (
    I2C_TypeDef * I2Cx,
    u8 taddr,
    u8 * data,
    u8 len )
```

向从机读取数据（需要在中断里执行I2C_TableIrq）

参数

<i>I2Cx</i>	I2C外设指针
<i>taddr</i>	从机地址
<i>data</i>	数据指针
<i>len</i>	数据长度

返回

状态，0成功 1失败（i2c总线非空闲，或上一次数据尚未发送完毕）

在文件 [lks32mc09x_i2c.c](#) 第 272 行定义.

3.25.2.16 I2C_TableSend()

```
int I2C_TableSend (
    I2C_TypeDef * I2Cx,
    u8 taddr,
    u8 * data,
    u8 len )
```

向从机发送数据（需要在中断里执行I2C_TableIrq）

参数

<i>I2Cx</i>	I2C外设指针
<i>taddr</i>	从机地址
<i>data</i>	数据指针
<i>len</i>	数据长度

返回

状态，0成功 1失败（i2c总线非空闲，或上一次数据尚未发送完毕）

在文件 [lks32mc09x_i2c.c](#) 第 251 行定义.

3.25.2.17 I2C_TableState()

```
int I2C_TableState (
    I2C_TypeDef * I2Cx )
```

获取I2C_Table发送状态

返回

状态，0空闲 1正在发送或接收数据

在文件 [lks32mc09x_i2c.c](#) 第 306 行定义.

3.26 lks32mc09x_i2c.h

[浏览该文件的文档.](#)

```
00001 #ifndef __LKS32MC09X_I2C_H__
00002 #define __LKS32MC09X_I2C_H__
00003
00004 #include "lks32mc09x_reg.h"
00005
00006 // I2C总线状态宏定义
00007 #define I2C_BUS_STATE_STT_ERR BIT7
00008 #define I2C_BUS_STATE_LOST_ARB BIT6
00009 #define I2C_BUS_STATE_STOP_EVT BIT5
00010 #define I2C_BUS_STATE_ADDR_DATA BIT3
00011 #define I2C_BUS_STATE_RX_ACK BIT1
00012 #define I2C_BUS_STATE_Done BIT0
00013
00014 // I2C中断标志宏定义
00015 #define I2C_IF_TC 0
00016 #define I2C_IF_BUS_ERR 1
00017 #define I2C_IF_STOP 2
00018 #define I2C_IF_NACK 3
00019 #define I2C_IF_ADDR_CMP 4
00020
00021 // I2C初始化结构体定义
00022 typedef struct
00023 {
00024     uint32_t ADRCMP;
00025     uint32_t ADDR;
00026     uint32_t MST_MODE;
00027     uint32_t SLV_MODE;
00028     uint32_t BUSRT_EN;
00029     uint32_t BaudRate;
00030     uint32_t IE;
00031     uint32_t TC_IE;
00032     uint32_t BUS_ERR_IE;
00033     uint32_t STOP_IE;
00034     uint32_t BURST_NACK;
00035     uint32_t BURST_ADDR_CMP;
00036 } I2C_InitTypeDef;
00037
00038 // 函数声明
00039
00045 void I2C_Init(I2C_TypeDef *I2Cx, I2C_InitTypeDef *I2C_InitStruct);
00046
00051 void I2C_StructInit(I2C_InitTypeDef *I2C_InitStruct);
00052
00058 void I2C_ClearIRQFlag(I2C_TypeDef *I2Cx, uint16_t I2C_IF_x);
00059
00066 uint8_t I2C_GetIRQFlag(I2C_TypeDef *I2Cx, uint16_t I2C_IF_x);
00067
```

```

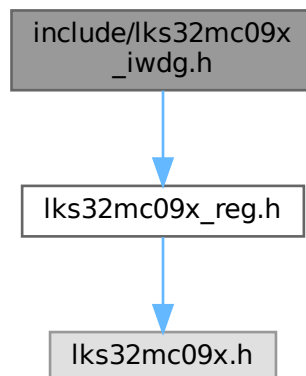
00074 uint8_t I2C_GetFlag(I2C_TypeDef *I2Cx, uint16_t I2C_IF_x);
00075
00080 void I2C_Enable(I2C_TypeDef *I2Cx);
00081
00086 void I2C_Disable(I2C_TypeDef *I2Cx);
00087
00092 void I2C_SetBaudRate(I2C_TypeDef *I2Cx, uint32_t BaudRate);
00093
00100 uint8_t I2C_ReadBusState(I2C_TypeDef *I2Cx, uint16_t state);
00101
00107 void I2C_ClearBusState(I2C_TypeDef *I2Cx, uint16_t state);
00108
00114 void I2C_SendData(I2C_TypeDef *I2Cx, uint8_t n);
00115
00121 uint8_t I2C_ReadData(I2C_TypeDef *I2Cx);
00122
00131 int I2C_TableSend(I2C_TypeDef *I2Cx, u8 taddr, u8 *data, u8 len);
00132
00141 int I2C_TableRead(I2C_TypeDef *I2Cx, u8 taddr, u8 *data, u8 len);
00146 int I2C_TableState(I2C_TypeDef *I2Cx);
00151 int I2C_TableAck(I2C_TypeDef *I2Cx);
00155 void I2C_TableIrq(I2C_TypeDef *I2Cx);
00156 #endif // __LKS32MC09X_I2C_H__

```

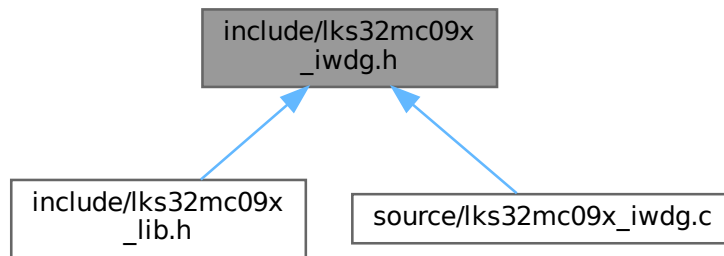
3.27 include/lks32mc09x_iwdg.h 文件参考

```
#include "lks32mc09x_reg.h"
```

lks32mc09x_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：



结构体

- struct **IWDG_InitTypeDef**
IWDG初始化结构体句柄

宏定义

- #define **IWDG_SECOND2IWDGCNT_SAT**(x, min, max) ((x) < (min) ? (min) : ((x) > (max) ? (max) : (x)))
将秒数转换为IWDG计数器值
- #define **IWDG_SECOND2IWDGCNT**(x) **IWDG_SECOND2IWDGCNT_SAT**((int32_t)(32000 * (float)x + 0x40), 0x100, 0x1fff00)

函数

- void **IWDG_Init** (IWDG_InitTypeDef *IWDG_InitStruct)
IWDG初始化
- void **IWDG_StrutInit** (IWDG_InitTypeDef *IWDG_InitStruct)
IWDG配置结构体初始化
- void **IWDG_DISABLE** (void)
关闭看门狗
- void **IWDG_ENABLE** (void)
使能看门狗
- void **IWDG_Feed** (void)
喂狗，重置看门狗计数器
- void **IWDG_Update** (uint32_t rth)
设置超时复位时间

3.27.1 宏定义说明

3.27.1.1 IWDG_SECOND2IWDGCNT

```
#define IWDG_SECOND2IWDGCNT(
    x ) IWDG_SECOND2IWDGCNT_SAT((int32_t)(32000 * (float)x + 0x40), 0x100, 0x1fff00)
```

在文件 **lks32mc09x_iwdg.h** 第 23 行定义。

3.27.1.2 IWDG_SECOND2IWDGCNT_SAT

```
#define IWDG_SECOND2IWDGCNT_SAT(  
    x,  
    min,  
    max ) ((x) < (min) ? (min) : ((x) > (max) ? (max) : (x)))
```

将秒数转换为IWDG计数器值

参数

x	秒数

返回

IWDG计数器值，向上取整，例如设置10ms复位，最小复位门限是8mS，则实际复位时间是16mS

在文件 [lks32mc09x_iwdg.h](#) 第 22 行定义.

3.27.2 函数说明

3.27.2.1 IWDG_DISABLE()

```
void IWDG_DISABLE (  
    void )
```

关闭看门狗

在文件 [lks32mc09x_iwdg.c](#) 第 42 行定义.

3.27.2.2 IWDG_ENABLE()

```
void IWDG_ENABLE (  
    void )
```

使能看门狗

在文件 [lks32mc09x_iwdg.c](#) 第 50 行定义.

3.27.2.3 IWDG_Feed()

```
void IWDG_Feed (  
    void )
```

喂狗，重置看门狗计数器

在文件 [lks32mc09x_iwdg.c](#) 第 58 行定义.

3.27.2.4 IWDG_Init()

```
void IWDG_Init (  
    IWDG_InitTypeDef * IWDG_InitStruct )
```

IWDG初始化

参数

<i>IWDG_InitStruct</i>	指向IWDG初始化结构体的指针

在文件 [lks32mc09x_iwdg.c](#) 第 7 行定义.

3.27.2.5 IWDG_StrutInit()

```
void IWDG_StrutInit (
    IWDG_InitTypeDef * IWDG_InitStruct )
```

IWDG配置结构体初始化

参数

<i>IWDG_InitStruct</i>	指向IWDG初始化结构体的指针

在文件 [lks32mc09x_iwdg.c](#) 第 30 行定义.

3.27.2.6 IWDG_Update()

```
void IWDG_Update (
    uint32_t rth )
```

设置超时复位时间

参数

<i>rth</i>	超时复位时间，单位1/64k 秒，建议用 SECOND2IWDGCNT 宏将秒转换

在文件 [lks32mc09x_iwdg.c](#) 第 67 行定义.

3.28 lks32mc09x_iwdg.h

[浏览该文件的文档.](#)

```
00001 #ifndef __LKS32MC09X_IWDG_H__
00002 #define __LKS32MC09X_IWDG_H__
00003
00004 #include "lks32mc09x_reg.h"
00005
00009 typedef struct
00010 {
00011     uint32_t DWK_EN;
00012     uint32_t WDG_EN;
00013     uint32_t WTH;
00014     uint32_t RTH;
00015 } IWDG_InitTypeDef;
00016
00022 #define IWDG_SECOND2IWDGCNT(x, min, max) ((x) < (min) ? (min) : ((x) > (max) ? (max) : (x)))
00023 #define IWDG_SECOND2IWDGCNT(x) IWDG_SECOND2IWDGCNT_SAT((int32_t)(32000 * (float)x +
00024     0x40), 0x100, 0xffff00)
00024
00029 void IWDG_Init(IWDG_InitTypeDef *IWDG_InitStruct);
```

```

00030
00035 void IWDG_StrutInit(IWDG_InitTypeDef *IWDG_InitStruct);
00039 void IWDG_DISABLE(void);
00040
00044 void IWDG_ENABLE(void);
00045
00049 void IWDG_Feed(void);
00050
00055 void IWDG_Update(uint32_t rth);
00056 #endif

```

3.29 include/lks32mc09x_lib.h 文件参考

```

#include "lks32mc09x_reg.h"
#include "lks32mc09x_user_manual.h"
#include "lks32mc09x_adc.h"
#include "lks32mc09x_clu.h"
#include "lks32mc09x_cmp.h"
#include "lks32mc09x_crc.h"
#include "lks32mc09x_dac.h"
#include "lks32mc09x_dma.h"
#include "lks32mc09x_dsp.h"
#include "lks32mc09x_eeeprom.h"
#include "lks32mc09x_exti.h"
#include "lks32mc09x_flash.h"
#include "lks32mc09x_gpio.h"
#include "lks32mc09x_hall.h"
#include "lks32mc09x_i2c.h"
#include "lks32mc09x_iwdg.h"
#include "lks32mc09x_mcpwm.h"
#include "lks32mc09x_opa.h"
#include "lks32mc09x_qep.h"
#include "lks32mc09x_sif.h"
#include "lks32mc09x_spi.h"
#include "lks32mc09x_sys.h"
#include "lks32mc09x_timer.h"
#include "lks32mc09x_tmp.h"
#include "lks32mc09x_trim.h"
#include "lks32mc09x_uart.h"
#include "lks32mc09x_wake.h"
lks32mc09x_lib.h 的引用(Include)关系图:

```



3.30 lks32mc09x_lib.h

[浏览该文件的文档.](#)

```

00001 #ifndef __lks32mc09x_LIB_H
00002 #define __lks32mc09x_LIB_H
00003
00004 #include "lks32mc09x_reg.h"
00005 #include "lks32mc09x_user_manual.h"
00006
00007 #include "lks32mc09x_adc.h"
00008 #include "lks32mc09x_clu.h"
00009 #include "lks32mc09x_cmp.h"
00010 #include "lks32mc09x_crc.h"

```

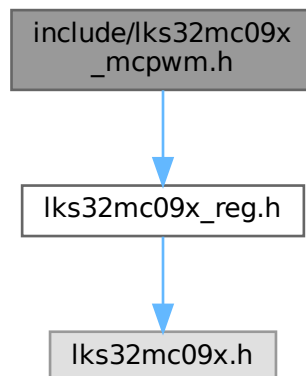


```
00011 #include "lks32mc09x_dac.h"
00012 #include "lks32mc09x_dma.h"
00013 #include "lks32mc09x_dsp.h"
00014 #include "lks32mc09x_eeprom.h"
00015 #include "lks32mc09x_exti.h"
00016 #include "lks32mc09x_flash.h"
00017 #include "lks32mc09x_gpio.h"
00018 #include "lks32mc09x_hall.h"
00019 #include "lks32mc09x_i2c.h"
00020 #include "lks32mc09x_iwdg.h"
00021 #include "lks32mc09x_mcpwm.h"
00022 #include "lks32mc09x_opa.h"
00023 #include "lks32mc09x_qep.h"
00024 #include "lks32mc09x_sif.h"
00025 #include "lks32mc09x_spi.h"
00026 #include "lks32mc09x_sys.h"
00027 #include "lks32mc09x_timer.h"
00028 #include "lks32mc09x_tmp.h"
00029 #include "lks32mc09x_trim.h"
00030 #include "lks32mc09x_uart.h"
00031 #include "lks32mc09x_wake.h"
00032
00033 #endif
```

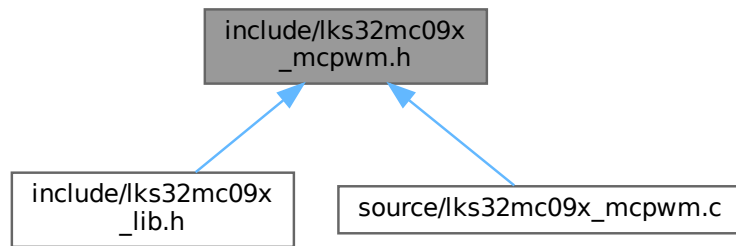
3.31 include/lks32mc09x_mcpwm.h 文件参考

```
#include "lks32mc09x_reg.h"
```

lks32mc09x_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：



结构体

- struct [MCPWM_InitTypeDef](#)

宏定义

- #define [MCPWM_IF_UP](#) BIT14
- #define [MCPWM_IF_TMR3](#) BIT13
- #define [MCPWM_IF_TMR2](#) BIT12
- #define [MCPWM_IF_TMR1](#) BIT11
- #define [MCPWM_IF_TMR0](#) BIT10
- #define [MCPWM_IF_TH21](#) BIT7
- #define [MCPWM_IF_TH20](#) BIT6
- #define [MCPWM_IF_TH11](#) BIT5
- #define [MCPWM_IF_TH10](#) BIT4
- #define [MCPWM_IF_TH01](#) BIT3
- #define [MCPWM_IF_TH00](#) BIT2
- #define [MCPWM_IF_T1](#) BIT1
- #define [MCPWM_IF_T0](#) BIT0
- #define [MCPWM_IE_UP](#) [MCPWM_IF_UP](#)
- #define [MCPWM_IE_TMR3](#) [MCPWM_IF_TMR3](#)
- #define [MCPWM_IE_TMR2](#) [MCPWM_IF_TMR2](#)
- #define [MCPWM_IE_TMR1](#) [MCPWM_IF_TMR1](#)
- #define [MCPWM_IE_TMR0](#) [MCPWM_IF_TMR0](#)
- #define [MCPWM_IE_TH21](#) [MCPWM_IF_TH21](#)
- #define [MCPWM_IE_TH20](#) [MCPWM_IF_TH20](#)
- #define [MCPWM_IE_TH11](#) [MCPWM_IF_TH11](#)
- #define [MCPWM_IE_TH10](#) [MCPWM_IF_TH10](#)
- #define [MCPWM_IE_TH01](#) [MCPWM_IF_TH01](#)
- #define [MCPWM_IE_TH00](#) [MCPWM_IF_TH00](#)
- #define [MCPWM_IE_T1](#) [MCPWM_IF_T1](#)
- #define [MCPWM_IE_T0](#) [MCPWM_IF_T0](#)
- #define [MCPWM_EIF_FAIL0](#) BIT4
- #define [MCPWM_EIF_FAIL1](#) BIT5
- #define [MCPWM_EIE_FAIL0](#) [MCPWM_EIF_FAIL0](#)
- #define [MCPWM_EIE_FAIL1](#) [MCPWM_EIF_FAIL1](#)

- #define MCPWM_AUEN_IO3 BIT28
- #define MCPWM_AUEN_TMR3 BIT25
- #define MCPWM_AUEN_TMR2 BIT24
- #define MCPWM_AUEN_TH31 BIT17
- #define MCPWM_AUEN_TH30 BIT16
- #define MCPWM_AUEN_IO012 BIT12
- #define MCPWM_AUEN_CNT0 BIT11
- #define MCPWM_AUEN_TH0 BIT10
- #define MCPWM_AUEN_TMR1 BIT9
- #define MCPWM_AUEN_TMR0 BIT8
- #define MCPWM_AUEN_TH21 BIT5
- #define MCPWM_AUEN_TH20 BIT4
- #define MCPWM_AUEN_TH11 BIT3
- #define MCPWM_AUEN_TH10 BIT2
- #define MCPWM_AUEN_TH01 BIT1
- #define MCPWM_AUEN_TH00 BIT0
- #define MCPWM_AUEN_IO (MCPWM_AUEN_IO012 | MCPWM_AUEN_IO3)
- #define MCPWM_AUEN_ALL (MCPWM_AUEN_IO3 | MCPWM_AUEN_TMR3 | MCPWM_AUEN_TMR2 | MCPWM_AUEN_TH31 | MCPWM_AUEN_TH30 | MCPWM_AUEN_IO012 | MCPWM_AUEN_CNT0 | MCPWM_AUEN_TH0 | MCPWM_AUEN_TMR1 | MCPWM_AUEN_TMR0 | MCPWM_AUEN_TH21 | MCPWM_AUEN_TH20 | MCPWM_AUEN_TH11 | MCPWM_AUEN_TH10 | MCPWM_AUEN_TH01 | MCPWM_AUEN_TH00)
- #define MCPWM_AUEN_DEFAULT (MCPWM_AUEN_ALL ^ MCPWM_AUEN_CNT0)
- #define MCPWM_IO_WM BIT7
- #define MCPWM_IO_PN_SW BIT6
- #define MCPWM_IO_SCTRLP BIT5
- #define MCPWM_IO_SCTRLN BIT4
- #define MCPWM_IO_PS BIT3
- #define MCPWM_IO_NS BIT2
- #define MCPWM_IO_PP BIT1
- #define MCPWM_IO_NP BIT0
- #define MCPWM_IO_DISABLE 0
- #define MCPWM_IO_HH_PWM 0
- #define MCPWM_IO_HH_LOW (MCPWM_IO_SCTRLN | MCPWM_IO_PS | MCPWM_IO_NS)
- #define MCPWM_IO_HH_HIGH (MCPWM_IO_SCTRLP | MCPWM_IO_PS | MCPWM_IO_NS)
- #define MCPWM_IO_HH_HPWM (MCPWM_IO_NS)
- #define MCPWM_IO_HH_LPWM (MCPWM_IO_PS)
- #define MCPWM_IO_HH_OFF (MCPWM_IO_NS | MCPWM_IO_PS)
- #define MCPWM_IO_HHS_PWM (MCPWM_IO_HH_PWM | MCPWM_IO_PN_SW)
- #define MCPWM_IO_HHS_LOW (MCPWM_IO_HH_LOW | MCPWM_IO_PN_SW)
- #define MCPWM_IO_HHS_HIGH (MCPWM_IO_HH_HIGH | MCPWM_IO_PN_SW)
- #define MCPWM_IO_HHS_HPWM (MCPWM_IO_HH_HPWM | MCPWM_IO_PN_SW)
- #define MCPWM_IO_HHS_LPWM (MCPWM_IO_HH_LPWM | MCPWM_IO_PN_SW)
- #define MCPWM_IO_HHS_OFF (MCPWM_IO_HH_OFF | MCPWM_IO_PN_SW)
- #define MCPWM_IO_HL_PWM (MCPWM_IO_NP)
- #define MCPWM_IO_HL_LOW (MCPWM_IO_NP | MCPWM_IO_SCTRLN | MCPWM_IO_PS | MCPWM_IO_NS)
- #define MCPWM_IO_HL_HIGH (MCPWM_IO_NP | MCPWM_IO_SCTRLP | MCPWM_IO_PS | MCPWM_IO_NS)
- #define MCPWM_IO_HL_HPWM (MCPWM_IO_NP | MCPWM_IO_NS)
- #define MCPWM_IO_HL_LPWM (MCPWM_IO_NP | MCPWM_IO_PS)
- #define MCPWM_IO_HL_OFF (MCPWM_IO_NP | MCPWM_IO_NS | MCPWM_IO_PS)
- #define MCPWM_IO_HLS_PWM (MCPWM_IO_HL_PWM | MCPWM_IO_PN_SW)
- #define MCPWM_IO_HLS_LOW (MCPWM_IO_HL_LOW | MCPWM_IO_PN_SW)
- #define MCPWM_IO_HLS_HIGH (MCPWM_IO_HL_HIGH | MCPWM_IO_PN_SW)
- #define MCPWM_IO_HLS_HPWM (MCPWM_IO_HL_HPWM | MCPWM_IO_PN_SW)
- #define MCPWM_IO_HLS_LPWM (MCPWM_IO_HL_LPWM | MCPWM_IO_PN_SW)

- #define MCPWM_IO_HLS_OFF (MCPWM_IO_HL_OFF | MCPWM_IO_PN_SW)
- #define MCPWM_IO_LH_PWM (MCPWM_IO_PP)
- #define MCPWM_IO_LH_LOW (MCPWM_IO_PP | MCPWM_IO_SCTRLN | MCPWM_IO_PS | MCPWM_IO_NS)
- #define MCPWM_IO_LH_HIGH (MCPWM_IO_PP | MCPWM_IO_SCTRLP | MCPWM_IO_PS | MCPWM_IO_NS)
- #define MCPWM_IO_LH_HPWM (MCPWM_IO_PP | MCPWM_IO_NS)
- #define MCPWM_IO_LH_LPWM (MCPWM_IO_PP | MCPWM_IO_PS)
- #define MCPWM_IO_LH_OFF (MCPWM_IO_PP | MCPWM_IO_NS | MCPWM_IO_PS)
- #define MCPWM_IO_LHS_PWM (MCPWM_IO_LH_PWM | MCPWM_IO_PN_SW)
- #define MCPWM_IO_LHS_LOW (MCPWM_IO_LH_LOW | MCPWM_IO_PN_SW)
- #define MCPWM_IO_LHS_HIGH (MCPWM_IO_LH_HIGH | MCPWM_IO_PN_SW)
- #define MCPWM_IO_LHS_HPWM (MCPWM_IO_LH_HPWM | MCPWM_IO_PN_SW)
- #define MCPWM_IO_LHS_LPWM (MCPWM_IO_LH_LPWM | MCPWM_IO_PN_SW)
- #define MCPWM_IO_LHS_OFF (MCPWM_IO_LH_OFF | MCPWM_IO_PN_SW)
- #define MCPWM_IO_LL_PWM (MCPWM_IO_PP | MCPWM_IO_NP)
- #define MCPWM_IO_LL_LOW (MCPWM_IO_PP | MCPWM_IO_NP | MCPWM_IO_SCTRLN | MCPWM_IO_PS | MCPWM_IO_NS)
- #define MCPWM_IO_LL_HIGH (MCPWM_IO_PP | MCPWM_IO_NP | MCPWM_IO_SCTRLP | MCPWM_IO_PS | MCPWM_IO_NS)
- #define MCPWM_IO_LL_HPWM (MCPWM_IO_PP | MCPWM_IO_NP | MCPWM_IO_NS)
- #define MCPWM_IO_LL_LPWM (MCPWM_IO_PP | MCPWM_IO_NP | MCPWM_IO_PS)
- #define MCPWM_IO_LL_OFF (MCPWM_IO_PP | MCPWM_IO_NP | MCPWM_IO_NS | MCPWM_IO_PS)
- #define MCPWM_IO_LLS_PWM (MCPWM_IO_LL_PWM | MCPWM_IO_PN_SW)
- #define MCPWM_IO_LLS_LOW (MCPWM_IO_LL_LOW | MCPWM_IO_PN_SW)
- #define MCPWM_IO_LLS_HIGH (MCPWM_IO_LL_HIGH | MCPWM_IO_PN_SW)
- #define MCPWM_IO_LLS_HPWM (MCPWM_IO_LL_HPWM | MCPWM_IO_PN_SW)
- #define MCPWM_IO_LLS_LPWM (MCPWM_IO_LL_LPWM | MCPWM_IO_PN_SW)
- #define MCPWM_IO_LLS_OFF (MCPWM_IO_LL_OFF | MCPWM_IO_PN_SW)
- #define MCPWM_COUNT_OFF 0
- #define MCPWM_COUNT_START BIT31
- #define MCPWM_COUNT_EVT_TIMER0_CMP0 BIT0
- #define MCPWM_COUNT_EVT_TIMER0_CMP1 BIT1
- #define MCPWM_COUNT_EVT_TIMER1_CMP0 BIT2
- #define MCPWM_COUNT_EVT_TIMER1_CMP1 BIT3
- #define MCPWM_COUNT_EVT_TIMER2_CMP0 BIT4
- #define MCPWM_COUNT_EVT_TIMER2_CMP1 BIT5
- #define MCPWM_CLK_DIV_1 0
- #define MCPWM_CLK_DIV_2 1
- #define MCPWM_CLK_DIV_4 2
- #define MCPWM_CLK_DIV_8 3
- #define MCPWM_CLK_DIV_16 4
- #define MCPWM_CLK_DIV_32 5
- #define MCPWM_CLK_DIV_64 6
- #define MCPWM_CLK_DIV_128 7
- #define MCPWM_TR_T0 BIT4
- #define MCPWM_TR_T1 BIT5
- #define MCPWM_FAIL_IO_HIGH 0xaa
- #define MCPWM_FAIL_IO_LOW 0x55
- #define MCPWM_FAIL_IO_OFF 0
- #define MCPWM_FAIL0_OFF (0)
- #define MCPWM_FAIL0_CMP0_H (BIT8 | BIT4)
- #define MCPWM_FAIL0_CMP0_L (BIT8 | BIT4 | BIT2)
- #define MCPWM_FAIL0_BKIN0_H (BIT4)
- #define MCPWM_FAIL0_BKIN0_L (BIT4 | BIT2)
- #define MCPWM_FAIL0_CLU_OUT0_H (BIT9 | BIT4)
- #define MCPWM_FAIL0_CLU_OUT0_L (BIT9 | BIT4 | BIT2)

- #define MCPWM_FAIL0_CLU_OUT1_H (BIT9 | BIT8 | BIT4)
- #define MCPWM_FAIL0_CLU_OUT1_L (BIT9 | BIT8 | BIT4 | BIT2)
- #define MCPWM_FAIL1_OFF (0)
- #define MCPWM_FAIL1_CMP1_H (BIT10 | BIT5)
- #define MCPWM_FAIL1_CMP1_L (BIT10 | BIT5 | BIT3)
- #define MCPWM_FAIL1_BKIN1_H (BIT5)
- #define MCPWM_FAIL1_BKIN1_L (BIT5 | BIT3)
- #define MCPWM_FAIL1_CLU_OUT2_H (BIT11 | BIT5)
- #define MCPWM_FAIL1_CLU_OUT2_L (BIT11 | BIT5 | BIT3)
- #define MCPWM_FAIL1_CLU_OUT3_H (BIT11 | BIT10 | BIT5)
- #define MCPWM_FAIL1_CLU_OUT3_L (BIT11 | BIT10 | BIT5 | BIT3)
- #define MCPWM_IRQ_UP(MCPWMx) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_UP)
- #define MCPWM_IRQ_TMR3(MCPWMx) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_TMR3)
- #define MCPWM_IRQ_TMR2(MCPWMx) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_TMR2)
- #define MCPWM_IRQ_TMR1(MCPWMx) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_TMR1)
- #define MCPWM_IRQ_TMR0(MCPWMx) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_TMR0)
- #define MCPWM_IRQ_TH21(MCPWMx) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_TH21)
- #define MCPWM_IRQ_TH20(MCPWMx) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_TH20)
- #define MCPWM_IRQ_TH11(MCPWMx) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_TH11)
- #define MCPWM_IRQ_TH10(MCPWMx) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_TH10)
- #define MCPWM_IRQ_TH01(MCPWMx) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_TH01)
- #define MCPWM_IRQ_TH00(MCPWMx) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_TH00)
- #define MCPWM_IRQ_T1(MCPWMx) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_T1)
- #define MCPWM_IRQ_T0(MCPWMx) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_T0)
- #define MCPWM_IRQ_FAIL0(MCPWMx) (MCPWMx->EIF & MCPWM->EIE & MCPWM_EIF_FAIL0)
- #define MCPWM_IRQ_FAIL1(MCPWMx) (MCPWMx->EIF & MCPWM->EIE & MCPWM_EIF_FAIL1)
- #define MCPWM_GET_IRQ_UP(MCPWMx) (MCPWMx->IF0 & MCPWM_IF_UP)
- #define MCPWM_GET_IRQ_TMR3(MCPWMx) (MCPWMx->IF0 & MCPWM_IF_TMR3)
- #define MCPWM_GET_IRQ_TMR2(MCPWMx) (MCPWMx->IF0 & MCPWM_IF_TMR2)
- #define MCPWM_GET_IRQ_TMR1(MCPWMx) (MCPWMx->IF0 & MCPWM_IF_TMR1)
- #define MCPWM_GET_IRQ_TMR0(MCPWMx) (MCPWMx->IF0 & MCPWM_IF_TMR0)
- #define MCPWM_GET_IRQ_TH21(MCPWMx) (MCPWMx->IF0 & MCPWM_IF_TH21)
- #define MCPWM_GET_IRQ_TH20(MCPWMx) (MCPWMx->IF0 & MCPWM_IF_TH20)
- #define MCPWM_GET_IRQ_TH11(MCPWMx) (MCPWMx->IF0 & MCPWM_IF_TH11)
- #define MCPWM_GET_IRQ_TH10(MCPWMx) (MCPWMx->IF0 & MCPWM_IF_TH10)
- #define MCPWM_GET_IRQ_TH01(MCPWMx) (MCPWMx->IF0 & MCPWM_IF_TH01)
- #define MCPWM_GET_IRQ_TH00(MCPWMx) (MCPWMx->IF0 & MCPWM_IF_TH00)
- #define MCPWM_GET_IRQ_T1(MCPWMx) (MCPWMx->IF0 & MCPWM_IF_T1)
- #define MCPWM_GET_IRQ_T0(MCPWMx) (MCPWMx->IF0 & MCPWM_IF_T0)
- #define MCPWM_GET_IRQ_FAIL0(MCPWMx) (MCPWMx->EIF & MCPWM_EIF_FAIL0)
- #define MCPWM_GET_IRQ_FAIL1(MCPWMx) (MCPWMx->EIF & MCPWM_EIF_FAIL1)
- #define MCPWM_IRQ_CLEAR_UP(MCPWMx) MCPWMx->IF0 = MCPWM_IF_UP
- #define MCPWM_IRQ_CLEAR_TMR3(MCPWMx) MCPWMx->IF0 = MCPWM_IF_TMR3
- #define MCPWM_IRQ_CLEAR_TMR2(MCPWMx) MCPWMx->IF0 = MCPWM_IF_TMR2
- #define MCPWM_IRQ_CLEAR_TMR1(MCPWMx) MCPWMx->IF0 = MCPWM_IF_TMR1
- #define MCPWM_IRQ_CLEAR_TMR0(MCPWMx) MCPWMx->IF0 = MCPWM_IF_TMR0
- #define MCPWM_IRQ_CLEAR_TH21(MCPWMx) MCPWMx->IF0 = MCPWM_IF_TH21
- #define MCPWM_IRQ_CLEAR_TH20(MCPWMx) MCPWMx->IF0 = MCPWM_IF_TH20
- #define MCPWM_IRQ_CLEAR_TH11(MCPWMx) MCPWMx->IF0 = MCPWM_IF_TH11
- #define MCPWM_IRQ_CLEAR_TH10(MCPWMx) MCPWMx->IF0 = MCPWM_IF_TH10
- #define MCPWM_IRQ_CLEAR_TH01(MCPWMx) MCPWMx->IF0 = MCPWM_IF_TH01
- #define MCPWM_IRQ_CLEAR_TH00(MCPWMx) MCPWMx->IF0 = MCPWM_IF_TH00
- #define MCPWM_IRQ_CLEAR_T1(MCPWMx) MCPWMx->IF0 = MCPWM_IF_T1
- #define MCPWM_IRQ_CLEAR_T0(MCPWMx) MCPWMx->IF0 = MCPWM_IF_T0
- #define MCPWM_IRQ_CLEAR_FAIL0(MCPWMx) MCPWMx->EIF = MCPWM_EIF_FAIL0
- #define MCPWM_IRQ_CLEAR_FAIL1(MCPWMx) MCPWMx->EIF = MCPWM_EIF_FAIL1

函数

- void **MCPWM_Enable** (MCPWM_TypeDef *MCPWMx)
模块使能
- void **MCPWM_Disable** (MCPWM_TypeDef *MCPWMx)
模块失能
- void **MCPWM_Reset** (MCPWM_TypeDef *MCPWMx)
模块复位
- void **MCPWM_Init** (MCPWM_TypeDef *MCPWMx, MCPWM_InitTypeDef *MCPWM_InitStruct)
模块初始化
- void **MCPWM_StructInit** (MCPWM_InitTypeDef *MCPWM_InitStruct)
*mcpwm*配置结构体初始化
- void **MCPWM_SetOutputState** (MCPWM_TypeDef *MCPWMx, uint8_t state)
设置*MCPWM*输出状态
- void **MCPWM_SetOutputVal** (MCPWM_TypeDef *MCPWMx, uint8_t chn, int16_t p, int16_t n)
设置*MCPWM*输出值
- void **MCPWM_SetTMR** (MCPWM_TypeDef *MCPWMx, int16_t tmr0, int16_t tmr1, int16_t tmr2, int16_t tmr3)
设置*MCPWM*触发*ADC*事件时间
- void **MCPWM_SetIOConfig** (MCPWM_TypeDef *MCPWMx, uint16_t io0, uint16_t io1, uint16_t io2, uint16_t io3)
设置*IO*配置
- void **MCPWM_StartCount** (MCPWM_TypeDef *MCPWMx)
开始计数
- void **MCPWM_StopCount** (MCPWM_TypeDef *MCPWMx)
停止计数
- uint16_t **MCPWM_GetIRQFlag** (MCPWM_TypeDef *MCPWMx, uint32_t MCPWM_IF_x)
获取*MCPWM*中断标志
- uint16_t **MCPWM_GetIfFlag** (MCPWM_TypeDef *MCPWMx, uint32_t MCPWM_IF_x)
获取*MCPWM*事件
- uint16_t **MCPWM_GetEIFFlag** (MCPWM_TypeDef *MCPWMx, uint32_t MCPWM_IF_x)
获取*MCPWM*错误中断标志
- uint16_t **MCPWM_GetErrorIRQFlag** (MCPWM_TypeDef *MCPWMx, uint32_t MCPWM{EIF_x})
获取*MCPWM*错误中断标志
- void **MCPWM_ClearIRQFlag** (MCPWM_TypeDef *MCPWMx, uint32_t MCPWM_IF_x)
清除*MCPWM*中断标志
- void **MCPWM_ClearErrorIRQFlag** (MCPWM_TypeDef *MCPWMx, uint32_t MCPWM{EIF_x})
清除*MCPWM*错误中断标志
- int16_t **MCPWM_GetCountTh** (MCPWM_TypeDef *MCPWMx)
获取当前计数器门限值
- int16_t **MCPWM_GetCountVal** (MCPWM_TypeDef *MCPWMx)
获取当前计数器值

3.31.1 宏定义说明

3.31.1.1 MCPWM_AUEN_ALL

```
#define MCPWM_AUEN_ALL (MCPWM_AUEN_IO3 | MCPWM_AUEN_TMR3 | MCPWM_AUEN_TMR2 | MCPWM_AUEN_TH31 | MCPWM_AUEN_TH30 |
MCPWM_AUEN_IO012 | MCPWM_AUEN_CNT0 | MCPWM_AUEN_TH0 | MCPWM_AUEN_TMR1 | MCPWM_AUEN_TMR0 | MCPWM_AUEN_TH21 |
MCPWM_AUEN_TH20 | MCPWM_AUEN_TH11 | MCPWM_AUEN_TH10 | MCPWM_AUEN_TH01 | MCPWM_AUEN_TH00)
```

在文件 **lks32mc09x_mcpwm.h** 第 86 行定义。

3.31.1.2 MCPWM_AUEN_CNT0

```
#define MCPWM_AUEN_CNT0 BIT11
```

在文件 [lks32mc09x_mcpwm.h](#) 第 73 行定义.

3.31.1.3 MCPWM_AUEN_DEFAULT

```
#define MCPWM_AUEN_DEFAULT (MCPWM_AUEN_ALL ^ MCPWM_AUEN_CNT0)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 88 行定义.

3.31.1.4 MCPWM_AUEN_IO

```
#define MCPWM_AUEN_IO (MCPWM_AUEN_IO012 | MCPWM_AUEN_IO3)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 84 行定义.

3.31.1.5 MCPWM_AUEN_IO012

```
#define MCPWM_AUEN_IO012 BIT12
```

在文件 [lks32mc09x_mcpwm.h](#) 第 72 行定义.

3.31.1.6 MCPWM_AUEN_IO3

```
#define MCPWM_AUEN_IO3 BIT28
```

在文件 [lks32mc09x_mcpwm.h](#) 第 67 行定义.

3.31.1.7 MCPWM_AUEN_TH0

```
#define MCPWM_AUEN_TH0 BIT10
```

在文件 [lks32mc09x_mcpwm.h](#) 第 74 行定义.

3.31.1.8 MCPWM_AUEN_TH00

```
#define MCPWM_AUEN_TH00 BIT0
```

在文件 [lks32mc09x_mcpwm.h](#) 第 82 行定义.

3.31.1.9 MCPWM_AUEN_TH01

```
#define MCPWM_AUEN_TH01 BIT1
```

在文件 [lks32mc09x_mcpwm.h](#) 第 81 行定义.

3.31.1.10 MCPWM_AUEN_TH10

```
#define MCPWM_AUEN_TH10 BIT2
```

在文件 [lks32mc09x_mcpwm.h](#) 第 80 行定义.

3.31.1.11 MCPWM_AUEN_TH11

```
#define MCPWM_AUEN_TH11 BIT3
```

在文件 [lks32mc09x_mcpwm.h](#) 第 79 行定义.

3.31.1.12 MCPWM_AUEN_TH20

```
#define MCPWM_AUEN_TH20 BIT4
```

在文件 [lks32mc09x_mcpwm.h](#) 第 78 行定义.

3.31.1.13 MCPWM_AUEN_TH21

```
#define MCPWM_AUEN_TH21 BIT5
```

在文件 [lks32mc09x_mcpwm.h](#) 第 77 行定义.

3.31.1.14 MCPWM_AUEN_TH30

```
#define MCPWM_AUEN_TH30 BIT16
```

在文件 [lks32mc09x_mcpwm.h](#) 第 71 行定义.

3.31.1.15 MCPWM_AUEN_TH31

```
#define MCPWM_AUEN_TH31 BIT17
```

在文件 [lks32mc09x_mcpwm.h](#) 第 70 行定义.

3.31.1.16 MCPWM_AUEN_TMR0

```
#define MCPWM_AUEN_TMR0 BIT8
```

在文件 [lks32mc09x_mcpwm.h](#) 第 76 行定义.

3.31.1.17 MCPWM_AUEN_TMR1

```
#define MCPWM_AUEN_TMR1 BIT9
```

在文件 [lks32mc09x_mcpwm.h](#) 第 75 行定义.

3.31.1.18 MCPWM_AUEN_TMR2

```
#define MCPWM_AUEN_TMR2 BIT24
```

在文件 [lks32mc09x_mcpwm.h](#) 第 69 行定义.

3.31.1.19 MCPWM_AUEN_TMR3

```
#define MCPWM_AUEN_TMR3 BIT25
```

在文件 [lks32mc09x_mcpwm.h](#) 第 68 行定义.

3.31.1.20 MCPWM_CLK_DIV_1

```
#define MCPWM_CLK_DIV_1 0
```

在文件 [lks32mc09x_mcpwm.h](#) 第 167 行定义.

3.31.1.21 MCPWM_CLK_DIV_128

```
#define MCPWM_CLK_DIV_128 7
```

在文件 [lks32mc09x_mcpwm.h](#) 第 174 行定义.

3.31.1.22 MCPWM_CLK_DIV_16

```
#define MCPWM_CLK_DIV_16 4
```

在文件 [lks32mc09x_mcpwm.h](#) 第 171 行定义.

3.31.1.23 MCPWM_CLK_DIV_2

```
#define MCPWM_CLK_DIV_2 1
```

在文件 [lks32mc09x_mcpwm.h](#) 第 168 行定义.

3.31.1.24 MCPWM_CLK_DIV_32

```
#define MCPWM_CLK_DIV_32 5
```

在文件 [lks32mc09x_mcpwm.h](#) 第 172 行定义.

3.31.1.25 MCPWM_CLK_DIV_4

```
#define MCPWM_CLK_DIV_4 2
```

在文件 [lks32mc09x_mcpwm.h](#) 第 169 行定义.

3.31.1.26 MCPWM_CLK_DIV_64

```
#define MCPWM_CLK_DIV_64 6
```

在文件 [lks32mc09x_mcpwm.h](#) 第 173 行定义.

3.31.1.27 MCPWM_CLK_DIV_8

```
#define MCPWM_CLK_DIV_8 3
```

在文件 [lks32mc09x_mcpwm.h](#) 第 170 行定义.

3.31.1.28 MCPWM_COUNT_EVT_TIMER0_CMP0

```
#define MCPWM_COUNT_EVT_TIMER0_CMP0 BIT0
```

在文件 [lks32mc09x_mcpwm.h](#) 第 160 行定义.

3.31.1.29 MCPWM_COUNT_EVT_TIMER0_CMP1

```
#define MCPWM_COUNT_EVT_TIMER0_CMP1 BIT1
```

在文件 [lks32mc09x_mcpwm.h](#) 第 161 行定义.

3.31.1.30 MCPWM_COUNT_EVT_TIMER1_CMP0

```
#define MCPWM_COUNT_EVT_TIMER1_CMP0 BIT2
```

在文件 [lks32mc09x_mcpwm.h](#) 第 162 行定义.

3.31.1.31 MCPWM_COUNT_EVT_TIMER1_CMP1

```
#define MCPWM_COUNT_EVT_TIMER1_CMP1 BIT3
```

在文件 [lks32mc09x_mcpwm.h](#) 第 163 行定义.

3.31.1.32 MCPWM_COUNT_EVT_TIMER2_CMP0

```
#define MCPWM_COUNT_EVT_TIMER2_CMP0 BIT4
```

在文件 [lks32mc09x_mcpwm.h](#) 第 164 行定义.

3.31.1.33 MCPWM_COUNT_EVT_TIMER2_CMP1

```
#define MCPWM_COUNT_EVT_TIMER2_CMP1 BIT5
```

在文件 [lks32mc09x_mcpwm.h](#) 第 165 行定义.

3.31.1.34 MCPWM_COUNT_OFF

```
#define MCPWM_COUNT_OFF 0
```

在文件 [lks32mc09x_mcpwm.h](#) 第 158 行定义.

3.31.1.35 MCPWM_COUNT_START

```
#define MCPWM_COUNT_START BIT31
```

在文件 [lks32mc09x_mcpwm.h](#) 第 159 行定义.

3.31.1.36 MCPWM_EIE_FAIL0

```
#define MCPWM_EIE_FAIL0 MCPWM_EIF_FAIL0
```

在文件 [lks32mc09x_mcpwm.h](#) 第 64 行定义.

3.31.1.37 MCPWM_EIE_FAIL1

```
#define MCPWM_EIE_FAIL1 MCPWM_EIF_FAIL1
```

在文件 [lks32mc09x_mcpwm.h](#) 第 65 行定义.

3.31.1.38 MCPWM_EIF_FAIL0

```
#define MCPWM_EIF_FAIL0 BIT4
```

在文件 [lks32mc09x_mcpwm.h](#) 第 62 行定义.

3.31.1.39 MCPWM_EIF_FAIL1

```
#define MCPWM_EIF_FAIL1 BIT5
```

在文件 [lks32mc09x_mcpwm.h](#) 第 63 行定义.

3.31.1.40 MCPWM_FAIL0_BKIN0_H

```
#define MCPWM_FAIL0_BKIN0_H (BIT4)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 188 行定义.

3.31.1.41 MCPWM_FAIL0_BKIN0_L

```
#define MCPWM_FAIL0_BKIN0_L (BIT4 | BIT2)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 189 行定义.

3.31.1.42 MCPWM_FAIL0_CLU_OUT0_H

```
#define MCPWM_FAIL0_CLU_OUT0_H (BIT9 | BIT4)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 190 行定义.

3.31.1.43 MCPWM_FAIL0_CLU_OUT0_L

```
#define MCPWM_FAIL0_CLU_OUT0_L (BIT9 | BIT4 | BIT2)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 191 行定义.

3.31.1.44 MCPWM_FAIL0_CLU_OUT1_H

```
#define MCPWM_FAIL0_CLU_OUT1_H (BIT9 | BIT8 | BIT4)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 192 行定义.

3.31.1.45 MCPWM_FAIL0_CLU_OUT1_L

```
#define MCPWM_FAIL0_CLU_OUT1_L (BIT9 | BIT8 | BIT4 | BIT2)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 193 行定义.

3.31.1.46 MCPWM_FAIL0_CMP0_H

```
#define MCPWM_FAIL0_CMP0_H (BIT8 | BIT4)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 186 行定义.

3.31.1.47 MCPWM_FAIL0_CMP0_L

```
#define MCPWM_FAIL0_CMP0_L (BIT8 | BIT4 | BIT2)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 187 行定义.

3.31.1.48 MCPWM_FAIL0_OFF

```
#define MCPWM_FAIL0_OFF (0)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 185 行定义.

3.31.1.49 MCPWM_FAIL1_BKIN1_H

```
#define MCPWM_FAIL1_BKIN1_H (BIT5)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 199 行定义.

3.31.1.50 MCPWM_FAIL1_BKIN1_L

```
#define MCPWM_FAIL1_BKIN1_L (BIT5 | BIT3)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 200 行定义.

3.31.1.51 MCPWM_FAIL1_CLU_OUT2_H

```
#define MCPWM_FAIL1_CLU_OUT2_H (BIT11 | BIT5)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 201 行定义.

3.31.1.52 MCPWM_FAIL1_CLU_OUT2_L

```
#define MCPWM_FAIL1_CLU_OUT2_L (BIT11 | BIT5 | BIT3)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 202 行定义.

3.31.1.53 MCPWM_FAIL1_CLU_OUT3_H

```
#define MCPWM_FAIL1_CLU_OUT3_H (BIT11 | BIT10 | BIT5)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 203 行定义.

3.31.1.54 MCPWM_FAIL1_CLU_OUT3_L

```
#define MCPWM_FAIL1_CLU_OUT3_L (BIT11 | BIT10 | BIT5 | BIT3)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 204 行定义.

3.31.1.55 MCPWM_FAIL1_CMP1_H

```
#define MCPWM_FAIL1_CMP1_H (BIT10 | BIT5)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 197 行定义.

3.31.1.56 MCPWM_FAIL1_CMP1_L

```
#define MCPWM_FAIL1_CMP1_L (BIT10 | BIT5 | BIT3)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 198 行定义.

3.31.1.57 MCPWM_FAIL1_OFF

```
#define MCPWM_FAIL1_OFF (0)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 196 行定义.

3.31.1.58 MCPWM_FAIL_IO_HIGH

```
#define MCPWM_FAIL_IO_HIGH 0xaa
```

在文件 [lks32mc09x_mcpwm.h](#) 第 180 行定义.

3.31.1.59 MCPWM_FAIL_IO_LOW

```
#define MCPWM_FAIL_IO_LOW 0x55
```

在文件 [lks32mc09x_mcpwm.h](#) 第 181 行定义.

3.31.1.60 MCPWM_FAIL_IO_OFF

```
#define MCPWM_FAIL_IO_OFF 0
```

在文件 [lks32mc09x_mcpwm.h](#) 第 182 行定义.

3.31.1.61 MCPWM_GET_IRQ_FAIL0

```
#define MCPWM_GET_IRQ_FAIL0(  
    MCPWMx ) (MCPWMx->EIF & MCPWM{EIF_FAIL0})
```

在文件 [lks32mc09x_mcpwm.h](#) 第 359 行定义.

3.31.1.62 MCPWM_GET_IRQ_FAIL1

```
#define MCPWM_GET_IRQ_FAIL1(  
    MCPWMx ) (MCPWMx->EIF & MCPWM{EIF_FAIL1})
```

在文件 [lks32mc09x_mcpwm.h](#) 第 360 行定义.

3.31.1.63 MCPWM_GET_IRQ_T0

```
#define MCPWM_GET_IRQ_T0(  
    MCPWMx ) (MCPWMx->IF0 & MCPWM{IF,T0})
```

在文件 [lks32mc09x_mcpwm.h](#) 第 358 行定义.

3.31.1.64 MCPWM_GET_IRQ_T1

```
#define MCPWM_GET_IRQ_T1(  
    MCPWMx ) (MCPWMx->IF0 & MCPWM{IF,T1})
```

在文件 [lks32mc09x_mcpwm.h](#) 第 357 行定义.

3.31.1.65 MCPWM_GET_IRQ_TH00

```
#define MCPWM_GET_IRQ_TH00(  
    MCPWMx ) (MCPWMx->IF0 & MCPWM_IF_TH00)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 356 行定义.

3.31.1.66 MCPWM_GET_IRQ_TH01

```
#define MCPWM_GET_IRQ_TH01(  
    MCPWMx ) (MCPWMx->IF0 & MCPWM_IF_TH01)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 355 行定义.

3.31.1.67 MCPWM_GET_IRQ_TH10

```
#define MCPWM_GET_IRQ_TH10(  
    MCPWMx ) (MCPWMx->IF0 & MCPWM_IF_TH10)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 354 行定义.

3.31.1.68 MCPWM_GET_IRQ_TH11

```
#define MCPWM_GET_IRQ_TH11(  
    MCPWMx ) (MCPWMx->IF0 & MCPWM_IF_TH11)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 353 行定义.

3.31.1.69 MCPWM_GET_IRQ_TH20

```
#define MCPWM_GET_IRQ_TH20(  
    MCPWMx ) (MCPWMx->IF0 & MCPWM_IF_TH20)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 352 行定义.

3.31.1.70 MCPWM_GET_IRQ_TH21

```
#define MCPWM_GET_IRQ_TH21(  
    MCPWMx ) (MCPWMx->IF0 & MCPWM_IF_TH21)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 351 行定义.

3.31.1.71 MCPWM_GET_IRQ_TMR0

```
#define MCPWM_GET_IRQ_TMR0(  
    MCPWMx ) (MCPWMx->IF0 & MCPWM_IF_TMR0)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 350 行定义.

3.31.1.72 MCPWM_GET_IRQ_TMR1

```
#define MCPWM_GET_IRQ_TMR1 (  
    MCPWMx ) (MCPWMx->IF0 & MCPWM_IF_TMR1)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 349 行定义.

3.31.1.73 MCPWM_GET_IRQ_TMR2

```
#define MCPWM_GET_IRQ_TMR2 (  
    MCPWMx ) (MCPWMx->IF0 & MCPWM_IF_TMR2)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 348 行定义.

3.31.1.74 MCPWM_GET_IRQ_TMR3

```
#define MCPWM_GET_IRQ_TMR3 (  
    MCPWMx ) (MCPWMx->IF0 & MCPWM_IF_TMR3)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 347 行定义.

3.31.1.75 MCPWM_GET_IRQ_UP

```
#define MCPWM_GET_IRQ_UP (  
    MCPWMx ) (MCPWMx->IF0 & MCPWM_IF_UP)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 346 行定义.

3.31.1.76 MCPWM_IE_T0

```
#define MCPWM_IE_T0 MCPWM_IF_T0
```

在文件 [lks32mc09x_mcpwm.h](#) 第 60 行定义.

3.31.1.77 MCPWM_IE_T1

```
#define MCPWM_IE_T1 MCPWM_IF_T1
```

在文件 [lks32mc09x_mcpwm.h](#) 第 59 行定义.

3.31.1.78 MCPWM_IE_TH00

```
#define MCPWM_IE_TH00 MCPWM_IF_TH00
```

在文件 [lks32mc09x_mcpwm.h](#) 第 58 行定义.

3.31.1.79 MCPWM_IE_TH01

```
#define MCPWM_IE_TH01 MCPWM_IF_TH01
```

在文件 [lks32mc09x_mcpwm.h](#) 第 57 行定义.

3.31.1.80 MCPWM_IE_TH10

```
#define MCPWM_IE_TH10 MCPWM_IF_TH10
```

在文件 [lks32mc09x_mcpwm.h](#) 第 56 行定义.

3.31.1.81 MCPWM_IE_TH11

```
#define MCPWM_IE_TH11 MCPWM_IF_TH11
```

在文件 [lks32mc09x_mcpwm.h](#) 第 55 行定义.

3.31.1.82 MCPWM_IE_TH20

```
#define MCPWM_IE_TH20 MCPWM_IF_TH20
```

在文件 [lks32mc09x_mcpwm.h](#) 第 54 行定义.

3.31.1.83 MCPWM_IE_TH21

```
#define MCPWM_IE_TH21 MCPWM_IF_TH21
```

在文件 [lks32mc09x_mcpwm.h](#) 第 53 行定义.

3.31.1.84 MCPWM_IE_TMR0

```
#define MCPWM_IE_TMR0 MCPWM_IF_TMR0
```

在文件 [lks32mc09x_mcpwm.h](#) 第 52 行定义.

3.31.1.85 MCPWM_IE_TMR1

```
#define MCPWM_IE_TMR1 MCPWM_IF_TMR1
```

在文件 [lks32mc09x_mcpwm.h](#) 第 51 行定义.

3.31.1.86 MCPWM_IE_TMR2

```
#define MCPWM_IE_TMR2 MCPWM_IF_TMR2
```

在文件 [lks32mc09x_mcpwm.h](#) 第 50 行定义.

3.31.1.87 MCPWM_IE_TMR3

```
#define MCPWM_IE_TMR3 MCPWM_IF_TMR3
```

在文件 [lks32mc09x_mcpwm.h](#) 第 49 行定义.

3.31.1.88 MCPWM_IE_UP

```
#define MCPWM_IE_UP MCPWM_IF_UP
```

在文件 [lks32mc09x_mcpwm.h](#) 第 48 行定义.

3.31.1.89 MCPWM_IF_T0

```
#define MCPWM_IF_T0 BIT0
```

在文件 [lks32mc09x_mcpwm.h](#) 第 46 行定义.

3.31.1.90 MCPWM_IF_T1

```
#define MCPWM_IF_T1 BIT1
```

在文件 [lks32mc09x_mcpwm.h](#) 第 45 行定义.

3.31.1.91 MCPWM_IF_TH00

```
#define MCPWM_IF_TH00 BIT2
```

在文件 [lks32mc09x_mcpwm.h](#) 第 44 行定义.

3.31.1.92 MCPWM_IF_TH01

```
#define MCPWM_IF_TH01 BIT3
```

在文件 [lks32mc09x_mcpwm.h](#) 第 43 行定义.

3.31.1.93 MCPWM_IF_TH10

```
#define MCPWM_IF_TH10 BIT4
```

在文件 [lks32mc09x_mcpwm.h](#) 第 42 行定义.

3.31.1.94 MCPWM_IF_TH11

```
#define MCPWM_IF_TH11 BIT5
```

在文件 [lks32mc09x_mcpwm.h](#) 第 41 行定义.

3.31.1.95 MCPWM_IF_TH20

```
#define MCPWM_IF_TH20 BIT6
```

在文件 [lks32mc09x_mcpwm.h](#) 第 40 行定义.

3.31.1.96 MCPWM_IF_TH21

```
#define MCPWM_IF_TH21 BIT7
```

在文件 [lks32mc09x_mcpwm.h](#) 第 39 行定义.

3.31.1.97 MCPWM_IF_TMR0

```
#define MCPWM_IF_TMR0 BIT10
```

在文件 [lks32mc09x_mcpwm.h](#) 第 38 行定义.

3.31.1.98 MCPWM_IF_TMR1

```
#define MCPWM_IF_TMR1 BIT11
```

在文件 [lks32mc09x_mcpwm.h](#) 第 37 行定义.

3.31.1.99 MCPWM_IF_TMR2

```
#define MCPWM_IF_TMR2 BIT12
```

在文件 [lks32mc09x_mcpwm.h](#) 第 36 行定义.

3.31.1.100 MCPWM_IF_TMR3

```
#define MCPWM_IF_TMR3 BIT13
```

在文件 [lks32mc09x_mcpwm.h](#) 第 35 行定义.

3.31.1.101 MCPWM_IF_UP

```
#define MCPWM_IF_UP BIT14
```

在文件 [lks32mc09x_mcpwm.h](#) 第 34 行定义.

3.31.1.102 MCPWM_IO_DISABLE

```
#define MCPWM_IO_DISABLE 0
```

在文件 [lks32mc09x_mcpwm.h](#) 第 100 行定义.

3.31.1.103 MCPWM_IO_HH_HIGH

```
#define MCPWM_IO_HH_HIGH (MCPWM_IO_SCTRLP | MCPWM_IO_PS | MCPWM_IO_NS)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 103 行定义.

3.31.1.104 MCPWM_IO_HH_HPWM

```
#define MCPWM_IO_HH_HPWM (MCPWM_IO_NS)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 104 行定义.

3.31.1.105 MCPWM_IO_HH_LOW

```
#define MCPWM_IO_HH_LOW (MCPWM_IO_SCTRLN | MCPWM_IO_PS | MCPWM_IO_NS)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 102 行定义.

3.31.1.106 MCPWM_IO_HH_LPWM

```
#define MCPWM_IO_HH_LPWM (MCPWM_IO_PS)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 105 行定义.

3.31.1.107 MCPWM_IO_HH_OFF

```
#define MCPWM_IO_HH_OFF (MCPWM_IO_NS | MCPWM_IO_PS)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 106 行定义.

3.31.1.108 MCPWM_IO_HH_PWM

```
#define MCPWM_IO_HH_PWM 0
```

在文件 [lks32mc09x_mcpwm.h](#) 第 101 行定义.

3.31.1.109 MCPWM_IO_HHS_HIGH

```
#define MCPWM_IO_HHS_HIGH (MCPWM_IO_HH_HIGH | MCPWM_IO_PN_SW)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 110 行定义.

3.31.1.110 MCPWM_IO_HHS_HPWM

```
#define MCPWM_IO_HHS_HPWM (MCPWM_IO_HH_HPWM | MCPWM_IO_PN_SW)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 111 行定义.

3.31.1.111 MCPWM_IO_HHS_LOW

```
#define MCPWM_IO_HHS_LOW (MCPWM_IO_HH_LOW | MCPWM_IO_PN_SW)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 109 行定义.

3.31.1.112 MCPWM_IO_HHS_LPWM

```
#define MCPWM_IO_HHS_LPWM (MCPWM_IO_HH_LPWM | MCPWM_IO_PN_SW)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 112 行定义.

3.31.1.113 MCPWM_IO_HHS_OFF

```
#define MCPWM_IO_HHS_OFF (MCPWM_IO_HH_OFF | MCPWM_IO_PN_SW)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 113 行定义.

3.31.1.114 MCPWM_IO_HHS_PWM

```
#define MCPWM_IO_HHS_PWM (MCPWM_IO_HH_PWM | MCPWM_IO_PN_SW)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 108 行定义.

3.31.1.115 MCPWM_IO_HL_HIGH

```
#define MCPWM_IO_HL_HIGH (MCPWM_IO_NP | MCPWM_IO_SCTRLP | MCPWM_IO_PS | MCPWM_IO_NS)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 117 行定义.

3.31.1.116 MCPWM_IO_HL_HPWM

```
#define MCPWM_IO_HL_HPWM (MCPWM_IO_NP | MCPWM_IO_NS)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 118 行定义.

3.31.1.117 MCPWM_IO_HL_LOW

```
#define MCPWM_IO_HL_LOW (MCPWM_IO_NP | MCPWM_IO_SCTRLN | MCPWM_IO_PS | MCPWM_IO_NS)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 116 行定义.

3.31.1.118 MCPWM_IO_HL_LPWM

```
#define MCPWM_IO_HL_LPWM (MCPWM_IO_NP | MCPWM_IO_PS)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 119 行定义.

3.31.1.119 MCPWM_IO_HL_OFF

```
#define MCPWM_IO_HL_OFF (MCPWM_IO_NP | MCPWM_IO_NS | MCPWM_IO_PS)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 120 行定义。

3.31.1.120 MCPWM_IO_HL_PWM

```
#define MCPWM_IO_HL_PWM (MCPWM_IO_NP)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 115 行定义。

3.31.1.121 MCPWM_IO_HLS_HIGH

```
#define MCPWM_IO_HLS_HIGH (MCPWM_IO_HL_HIGH | MCPWM_IO_PN_SW)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 124 行定义。

3.31.1.122 MCPWM_IO_HLS_HPWM

```
#define MCPWM_IO_HLS_HPWM (MCPWM_IO_HL_HPWM | MCPWM_IO_PN_SW)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 125 行定义。

3.31.1.123 MCPWM_IO_HLS_LOW

```
#define MCPWM_IO_HLS_LOW (MCPWM_IO_HL_LOW | MCPWM_IO_PN_SW)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 123 行定义。

3.31.1.124 MCPWM_IO_HLS_LPWM

```
#define MCPWM_IO_HLS_LPWM (MCPWM_IO_HL_LPWM | MCPWM_IO_PN_SW)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 126 行定义。

3.31.1.125 MCPWM_IO_HLS_OFF

```
#define MCPWM_IO_HLS_OFF (MCPWM_IO_HL_OFF | MCPWM_IO_PN_SW)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 127 行定义。

3.31.1.126 MCPWM_IO_HLS_PWM

```
#define MCPWM_IO_HLS_PWM (MCPWM_IO_HL_PWM | MCPWM_IO_PN_SW)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 122 行定义。

3.31.1.127 MCPWM_IO_LH_HIGH

```
#define MCPWM_IO_LH_HIGH (MCPWM_IO_PP | MCPWM_IO_SCTRLP | MCPWM_IO_PS | MCPWM_IO_NS)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 131 行定义.

3.31.1.128 MCPWM_IO_LH_HPWM

```
#define MCPWM_IO_LH_HPWM (MCPWM_IO_PP | MCPWM_IO_NS)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 132 行定义.

3.31.1.129 MCPWM_IO_LH_LOW

```
#define MCPWM_IO_LH_LOW (MCPWM_IO_PP | MCPWM_IO_SCTRLN | MCPWM_IO_PS | MCPWM_IO_NS)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 130 行定义.

3.31.1.130 MCPWM_IO_LH_LPWM

```
#define MCPWM_IO_LH_LPWM (MCPWM_IO_PP | MCPWM_IO_PS)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 133 行定义.

3.31.1.131 MCPWM_IO_LH_OFF

```
#define MCPWM_IO_LH_OFF (MCPWM_IO_PP | MCPWM_IO_NS | MCPWM_IO_PS)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 134 行定义.

3.31.1.132 MCPWM_IO_LH_PWM

```
#define MCPWM_IO_LH_PWM (MCPWM_IO_PP)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 129 行定义.

3.31.1.133 MCPWM_IO_LHS_HIGH

```
#define MCPWM_IO_LHS_HIGH (MCPWM_IO_LH_HIGH | MCPWM_IO_PN_SW)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 138 行定义.

3.31.1.134 MCPWM_IO_LHS_HPWM

```
#define MCPWM_IO_LHS_HPWM (MCPWM_IO_LH_HPWM | MCPWM_IO_PN_SW)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 139 行定义.

3.31.1.135 MCPWM_IO_LHS_LOW

```
#define MCPWM_IO_LHS_LOW (MCPWM_IO_LH_LOW | MCPWM_IO_PN_SW)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 137 行定义.

3.31.1.136 MCPWM_IO_LHS_LPWM

```
#define MCPWM_IO_LHS_LPWM (MCPWM_IO_LH_LPWM | MCPWM_IO_PN_SW)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 140 行定义.

3.31.1.137 MCPWM_IO_LHS_OFF

```
#define MCPWM_IO_LHS_OFF (MCPWM_IO_LH_OFF | MCPWM_IO_PN_SW)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 141 行定义.

3.31.1.138 MCPWM_IO_LHS_PWM

```
#define MCPWM_IO_LHS_PWM (MCPWM_IO_LH_PWM | MCPWM_IO_PN_SW)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 136 行定义.

3.31.1.139 MCPWM_IO_LL_HIGH

```
#define MCPWM_IO_LL_HIGH (MCPWM_IO_PP | MCPWM_IO_NP | MCPWM_IO_SCTRLP | MCPWM_IO_PS | MCPWM_IO_NS)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 145 行定义.

3.31.1.140 MCPWM_IO_LL_HPWM

```
#define MCPWM_IO_LL_HPWM (MCPWM_IO_PP | MCPWM_IO_NP | MCPWM_IO_NS)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 146 行定义.

3.31.1.141 MCPWM_IO_LL_LOW

```
#define MCPWM_IO_LL_LOW (MCPWM_IO_PP | MCPWM_IO_NP | MCPWM_IO_SCTRLN | MCPWM_IO_PS | MCPWM_IO_NS)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 144 行定义.

3.31.1.142 MCPWM_IO_LL_LPWM

```
#define MCPWM_IO_LL_LPWM (MCPWM_IO_PP | MCPWM_IO_NP | MCPWM_IO_PS)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 147 行定义.

3.31.1.143 MCPWM_IO_LL_OFF

```
#define MCPWM_IO_LL_OFF (MCPWM_IO_PP | MCPWM_IO_NP | MCPWM_IO_NS | MCPWM_IO_PS)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 148 行定义。

3.31.1.144 MCPWM_IO_LL_PWM

```
#define MCPWM_IO_LL_PWM (MCPWM_IO_PP | MCPWM_IO_NP)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 143 行定义。

3.31.1.145 MCPWM_IO_LLS_HIGH

```
#define MCPWM_IO_LLS_HIGH (MCPWM_IO_LL_HIGH | MCPWM_IO_PN_SW)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 152 行定义。

3.31.1.146 MCPWM_IO_LLS_HPWM

```
#define MCPWM_IO_LLS_HPWM (MCPWM_IO_LL_HPWM | MCPWM_IO_PN_SW)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 153 行定义。

3.31.1.147 MCPWM_IO_LLS_LOW

```
#define MCPWM_IO_LLS_LOW (MCPWM_IO_LL_LOW | MCPWM_IO_PN_SW)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 151 行定义。

3.31.1.148 MCPWM_IO_LLS_LPWM

```
#define MCPWM_IO_LLS_LPWM (MCPWM_IO_LL_LPWM | MCPWM_IO_PN_SW)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 154 行定义。

3.31.1.149 MCPWM_IO_LLS_OFF

```
#define MCPWM_IO_LLS_OFF (MCPWM_IO_LL_OFF | MCPWM_IO_PN_SW)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 155 行定义。

3.31.1.150 MCPWM_IO_LLS_PWM

```
#define MCPWM_IO_LLS_PWM (MCPWM_IO_LL_PWM | MCPWM_IO_PN_SW)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 150 行定义。

3.31.1.151 MCPWM_IO_NP

```
#define MCPWM_IO_NP BIT0
```

在文件 [lks32mc09x_mcpwm.h](#) 第 98 行定义.

3.31.1.152 MCPWM_IO_NS

```
#define MCPWM_IO_NS BIT2
```

在文件 [lks32mc09x_mcpwm.h](#) 第 96 行定义.

3.31.1.153 MCPWM_IO_PN_SW

```
#define MCPWM_IO_PN_SW BIT6
```

在文件 [lks32mc09x_mcpwm.h](#) 第 92 行定义.

3.31.1.154 MCPWM_IO_PP

```
#define MCPWM_IO_PP BIT1
```

在文件 [lks32mc09x_mcpwm.h](#) 第 97 行定义.

3.31.1.155 MCPWM_IO_PS

```
#define MCPWM_IO_PS BIT3
```

在文件 [lks32mc09x_mcpwm.h](#) 第 95 行定义.

3.31.1.156 MCPWM_IO_SCTRLN

```
#define MCPWM_IO_SCTRLN BIT4
```

在文件 [lks32mc09x_mcpwm.h](#) 第 94 行定义.

3.31.1.157 MCPWM_IO_SCTRLP

```
#define MCPWM_IO_SCTRLP BIT5
```

在文件 [lks32mc09x_mcpwm.h](#) 第 93 行定义.

3.31.1.158 MCPWM_IO_WM

```
#define MCPWM_IO_WM BIT7
```

在文件 [lks32mc09x_mcpwm.h](#) 第 91 行定义.

3.31.1.159 MCPWM_IRQ_CLEAR_FAIL0

```
#define MCPWM_IRQ_CLEAR_FAIL0(  
    MCPWMx ) MCPWMx->EIF = MCPWM_EIF_FAIL0
```

在文件 [lks32mc09x_mcpwm.h](#) 第 376 行定义.

3.31.1.160 MCPWM_IRQ_CLEAR_FAIL1

```
#define MCPWM_IRQ_CLEAR_FAIL1(  
    MCPWMx ) MCPWMx->EIF = MCPWM_EIF_FAIL1
```

在文件 [lks32mc09x_mcpwm.h](#) 第 377 行定义.

3.31.1.161 MCPWM_IRQ_CLEAR_T0

```
#define MCPWM_IRQ_CLEAR_T0(  
    MCPWMx ) MCPWMx->IF0 = MCPWM_IF_T0
```

在文件 [lks32mc09x_mcpwm.h](#) 第 375 行定义.

3.31.1.162 MCPWM_IRQ_CLEAR_T1

```
#define MCPWM_IRQ_CLEAR_T1(  
    MCPWMx ) MCPWMx->IF0 = MCPWM_IF_T1
```

在文件 [lks32mc09x_mcpwm.h](#) 第 374 行定义.

3.31.1.163 MCPWM_IRQ_CLEAR_TH00

```
#define MCPWM_IRQ_CLEAR_TH00(  
    MCPWMx ) MCPWMx->IF0 = MCPWM_IF_TH00
```

在文件 [lks32mc09x_mcpwm.h](#) 第 373 行定义.

3.31.1.164 MCPWM_IRQ_CLEAR_TH01

```
#define MCPWM_IRQ_CLEAR_TH01(  
    MCPWMx ) MCPWMx->IF0 = MCPWM_IF_TH01
```

在文件 [lks32mc09x_mcpwm.h](#) 第 372 行定义.

3.31.1.165 MCPWM_IRQ_CLEAR_TH10

```
#define MCPWM_IRQ_CLEAR_TH10(  
    MCPWMx ) MCPWMx->IF0 = MCPWM_IF_TH10
```

在文件 [lks32mc09x_mcpwm.h](#) 第 371 行定义.

3.31.1.166 MCPWM_IRQ_CLEAR_TH11

```
#define MCPWM_IRQ_CLEAR_TH11(  
    MCPWMx ) MCPWMx->IF0 = MCPWM_IF_TH11
```

在文件 [lks32mc09x_mcpwm.h](#) 第 370 行定义.

3.31.1.167 MCPWM_IRQ_CLEAR_TH20

```
#define MCPWM_IRQ_CLEAR_TH20(  
    MCPWMx ) MCPWMx->IF0 = MCPWM_IF_TH20
```

在文件 [lks32mc09x_mcpwm.h](#) 第 369 行定义.

3.31.1.168 MCPWM_IRQ_CLEAR_TH21

```
#define MCPWM_IRQ_CLEAR_TH21(  
    MCPWMx ) MCPWMx->IF0 = MCPWM_IF_TH21
```

在文件 [lks32mc09x_mcpwm.h](#) 第 368 行定义.

3.31.1.169 MCPWM_IRQ_CLEAR_TMR0

```
#define MCPWM_IRQ_CLEAR_TMR0(  
    MCPWMx ) MCPWMx->IF0 = MCPWM_IF_TMR0
```

在文件 [lks32mc09x_mcpwm.h](#) 第 367 行定义.

3.31.1.170 MCPWM_IRQ_CLEAR_TMR1

```
#define MCPWM_IRQ_CLEAR_TMR1(  
    MCPWMx ) MCPWMx->IF0 = MCPWM_IF_TMR1
```

在文件 [lks32mc09x_mcpwm.h](#) 第 366 行定义.

3.31.1.171 MCPWM_IRQ_CLEAR_TMR2

```
#define MCPWM_IRQ_CLEAR_TMR2(  
    MCPWMx ) MCPWMx->IF0 = MCPWM_IF_TMR2
```

在文件 [lks32mc09x_mcpwm.h](#) 第 365 行定义.

3.31.1.172 MCPWM_IRQ_CLEAR_TMR3

```
#define MCPWM_IRQ_CLEAR_TMR3(  
    MCPWMx ) MCPWMx->IF0 = MCPWM_IF_TMR3
```

在文件 [lks32mc09x_mcpwm.h](#) 第 364 行定义.

3.31.1.173 MCPWM_IRQ_CLEAR_UP

```
#define MCPWM_IRQ_CLEAR_UP(  
    MCPWMx ) MCPWMx->IF0 = MCPWM_IF_UP
```

在文件 [lks32mc09x_mcpwm.h](#) 第 363 行定义.

3.31.1.174 MCPWM_IRQ_FAIL0

```
#define MCPWM_IRQ_FAIL0(  
    MCPWMx ) (MCPWMx->EIF & MCPWM->EIE & MCPWM{EIF_FAIL0})
```

在文件 [lks32mc09x_mcpwm.h](#) 第 342 行定义.

3.31.1.175 MCPWM_IRQ_FAIL1

```
#define MCPWM_IRQ_FAIL1(  
    MCPWMx ) (MCPWMx->EIF & MCPWM->EIE & MCPWM{EIF_FAIL1})
```

在文件 [lks32mc09x_mcpwm.h](#) 第 343 行定义.

3.31.1.176 MCPWM_IRQ_T0

```
#define MCPWM_IRQ_T0(  
    MCPWMx ) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_T0)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 341 行定义.

3.31.1.177 MCPWM_IRQ_T1

```
#define MCPWM_IRQ_T1(  
    MCPWMx ) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_T1)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 340 行定义.

3.31.1.178 MCPWM_IRQ_TH00

```
#define MCPWM_IRQ_TH00(  
    MCPWMx ) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_TH00)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 339 行定义.

3.31.1.179 MCPWM_IRQ_TH01

```
#define MCPWM_IRQ_TH01(  
    MCPWMx ) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_TH01)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 338 行定义.

3.31.1.180 MCPWM_IRQ_TH10

```
#define MCPWM_IRQ_TH10(  
    MCPWMx ) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_TH10)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 337 行定义.

3.31.1.181 MCPWM_IRQ_TH11

```
#define MCPWM_IRQ_TH11(  
    MCPWMx ) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_TH11)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 336 行定义.

3.31.1.182 MCPWM_IRQ_TH20

```
#define MCPWM_IRQ_TH20(  
    MCPWMx ) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_TH20)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 335 行定义.

3.31.1.183 MCPWM_IRQ_TH21

```
#define MCPWM_IRQ_TH21(  
    MCPWMx ) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_TH21)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 334 行定义.

3.31.1.184 MCPWM_IRQ_TMR0

```
#define MCPWM_IRQ_TMR0(  
    MCPWMx ) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_TMR0)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 333 行定义.

3.31.1.185 MCPWM_IRQ_TMR1

```
#define MCPWM_IRQ_TMR1(  
    MCPWMx ) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_TMR1)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 332 行定义.

3.31.1.186 MCPWM_IRQ_TMR2

```
#define MCPWM_IRQ_TMR2(  
    MCPWMx ) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_TMR2)
```

在文件 [lks32mc09x_mcpwm.h](#) 第 331 行定义.

3.31.1.187 MCPWM_IRQ_TMR3

```
#define MCPWM_IRQ_TMR3 (
    MCPWMx ) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_TMR3)
```

在文件 lks32mc09x_mcpwm.h 第 330 行定义.

3.31.1.188 MCPWM_IRQ_UP

```
#define MCPWM_IRQ_UP (
    MCPWMx ) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_UP)
```

在文件 lks32mc09x_mcpwm.h 第 329 行定义.

3.31.1.189 MCPWM_TR_T0

```
#define MCPWM_TR_T0 BIT4
```

在文件 lks32mc09x_mcpwm.h 第 176 行定义.

3.31.1.190 MCPWM_TR_T1

```
#define MCPWM_TR_T1 BIT5
```

在文件 lks32mc09x_mcpwm.h 第 177 行定义.

3.31.2 函数说明

3.31.2.1 MCPWM_ClearErrorIRQFlag()

```
void MCPWM_ClearErrorIRQFlag (
    MCPWM_TypeDef * MCPWMx,
    uint32_t MCPWM{EIF_x} )
```

清除MCPWM错误中断标志

参数

MCPWMx	MCPWM模块
MCPWMEIF_x	错误中断标志位

在文件 lks32mc09x_mcpwm.c 第 284 行定义.

3.31.2.2 MCPWM_ClearIRQFlag()

```
void MCPWM_ClearIRQFlag (
```

```
MCPWM_TypeDef * MCPWMx,  
uint32_t MCPWM_IF_x )
```

清除MCPWM中断标志

参数

MCPWMx	MCPWM模块
MCPWM_↔IF_x	中断标志位

在文件 lks32mc09x_mcpwm.c 第 275 行定义.

3.31.2.3 MCPWM_Disable()

```
void MCPWM_Disable (  
MCPWM_TypeDef * MCPWMx )
```

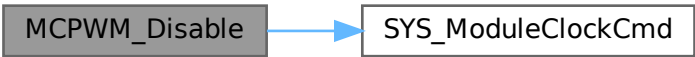
模块失能

参数

MCPWMx	MCPWM模块

在文件 lks32mc09x_mcpwm.c 第 22 行定义.

函数调用图:



这是这个函数的调用关系图:



3.31.2.4 MCPWM_Enable()

```
void MCPWM_Enable (
    MCPWM_TypeDef * MCPWMx )
```

模块使能

参数

<i>MCPWMx</i>	MCPWM模块

在文件 [lks32mc09x-mcpwm.c](#) 第 11 行定义.

函数调用图:



这是这个函数的调用关系图:



3.31.2.5 MCPWM_GetCountTh()

```
int16_t MCPWM_GetCountTh (
    MCPWM_TypeDef * MCPWMx )
```

获取当前计数器门限值

参数

<i>MCPWMx</i>	MCPWM模块

返回

当前计数器门限值

在文件 [lks32mc09x_mcpwm.c](#) 第 294 行定义.

3.31.2.6 MCPWM_GetCountVal()

```
int16_t MCPWM_GetCountVal (
    MCPWM_TypeDef * MCPWMx )
```

获取当前计数器值

参数

MCPWMx	MCPWM模块

返回

当前计数器值

在文件 [lks32mc09x_mcpwm.c](#) 第 303 行定义.

3.31.2.7 MCPWM_GetEifFlag()

```
uint16_t MCPWM_GetEifFlag (
    MCPWM_TypeDef * MCPWMx,
    uint32_t MCPWM_IF_x )
```

获取MCPWM错误中断标志

参数

MCPWMx	MCPWM模块
MCPWM_↔ EIF_x	错误中断标志位

返回

1事件发生 0事件未发生

在文件 [lks32mc09x_mcpwm.c](#) 第 256 行定义.

3.31.2.8 MCPWM_GetErrorIRQFlag()

```
uint16_t MCPWM_GetErrorIRQFlag (
    MCPWM_TypeDef * MCPWMx,
    uint32_t MCPWM{EIF_x )
```

获取CMPWM错误中断标志

参数

<i>MCPWM_x</i>	MCPWM模块
<i>MCPWM_↔ EIF_x</i>	错误中断标志位

返回

1中断发生 0中断未发生

在文件 [lks32mc09x_mcpwm.c](#) 第 266 行定义.

3.31.2.9 MCPWM.GetIfFlag()

```
uint16_t MCPWM.GetIfFlag (
    MCPWM_TypeDef * MCPWMx,
    uint32_t MCPWM_IF_x )
```

获取MCPWM事件

参数

<i>MCPWM_x</i>	MCPWM模块
<i>MCPWM_↔ IF_x</i>	中断标志位

返回

1事件发生 0事件未发生

注解

事件和中断的区别在于事件只是一个标志位，用于在中断函数外部判断事件是否发生

在文件 [lks32mc09x_mcpwm.c](#) 第 246 行定义.

3.31.2.10 MCPWM.GetIRQFlag()

```
uint16_t MCPWM.GetIRQFlag (
    MCPWM_TypeDef * MCPWMx,
    uint32_t MCPWM_IF_x )
```

获取MCPWM中断标志

参数

<i>MCPWM_x</i>	MCPWM模块
<i>MCPWM_↔</i>	中断标志位

返回

1中断发生 0中断未发生

在文件 [lks32mc09x_mcpwm.c](#) 第 235 行定义.

3.31.2.11 MCPWM.Init()

```
void MCPWM_Init (
    MCPWM_TypeDef * MCPWMx,
    MCPWM_InitTypeDef * MCPWM_InitStruct )
```

模块初始化

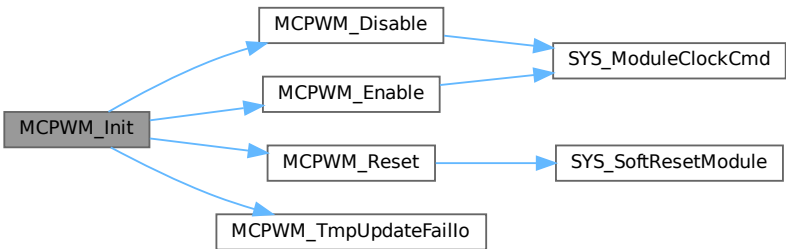
模块初始化

参数

<i>MCPWMx</i>	MCPWM模块指针
<i>MCPWM_InitStruct</i>	指向包含初始化参数的MCPWM_InitTypeDef结构体

在文件 [lks32mc09x_mcpwm.c](#) 第 45 行定义.

函数调用图:



3.31.2.12 MCPWM.Reset()

```
void MCPWM_Reset (
    MCPWM_TypeDef * MCPWMx )
```

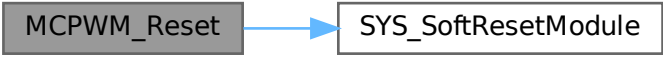
模块复位

参数

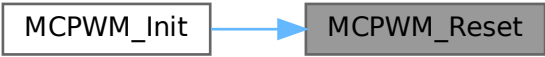
<i>MCPWMx</i>	MCPWM模块

在文件 [lks32mc09x_mcpwm.c](#) 第 33 行定义.

函数调用图:



这是这个函数的调用关系图:



3.31.2.13 MCPWM.SetIOConfig()

```
void MCPWM_SetIOConfig (
    MCPWM_TypeDef * MCPWMx,
    uint16_t io0,
    uint16_t io1,
    uint16_t io2,
    uint16_t io3 )
```

设置IO配置

参数

<i>MCPWMx</i>	MCPWM模块
<i>io0</i>	IO0配置
<i>io1</i>	IO1配置
<i>io2</i>	IO2配置
<i>io3</i>	IO3配置

注解

由于mcpwm的io0和io1是一个寄存器，io2和io3是一个寄存器，所以需要同时配置

在文件 [lks32mc09x_mcpwm.c](#) 第 200 行定义.

3.31.2.14 MCPWM.SetOutputState()

```
void MCPWM_SetOutputState (
    MCPWM_TypeDef * MCPWMx,
    uint8_t state )
```

设置MCPWM输出状态

参数

MCPWMx	MCPWM模块
state	输出状态 1打开输出 0关闭输出

在文件 [lks32mc09x_mcpwm.c](#) 第 131 行定义.

3.31.2.15 MCPWM.SetOutputVal()

```
void MCPWM_SetOutputVal (
    MCPWM_TypeDef * MCPWMx,
    uint8_t chn,
    int16_t p,
    int16_t n )
```

设置MCPWM输出值

参数

MCPWMx	MCPWM模块
chn	通道号 0-3
p	上管打开时间，取值范围-th到th
n	下管打开时间，取值范围-th到th
MCPWMx	MCPWM模块
chn	通道号 0-3
p	上管打开时间
n	下管打开时间

在文件 [lks32mc09x_mcpwm.c](#) 第 152 行定义.

3.31.2.16 MCPWM.SetTMR()

```
void MCPWM_SetTMR (
    MCPWM_TypeDef * MCPWMx,
    int16_t tmr0,
    int16_t tmr1,
    int16_t tmr2,
    int16_t tmr3 )
```

设置MCPWM触发ADC事件时间

参数

<i>MCPWMx</i>	MCPWM模块
<i>tmr0</i>	事件0时间
<i>tmr1</i>	事件1时间
<i>tmr2</i>	事件2时间
<i>tmr3</i>	事件3时间

在文件 [lks32mc09x_mcpwm.c](#) 第 184 行定义.

3.31.2.17 MCPWM.StartCount()

```
void MCPWM_StartCount (
    MCPWM_TypeDef * MCPWMx )
```

开始计数

参数

<i>MCPWMx</i>	MCPWM模块
<i>MCPWMx</i>	MCPWM模块
<i>count</i>	计数器初始值

在文件 [lks32mc09x_mcpwm.c](#) 第 212 行定义.

3.31.2.18 MCPWM.StopCount()

```
void MCPWM_StopCount (
    MCPWM_TypeDef * MCPWMx )
```

停止计数

参数

<i>MCPWMx</i>	MCPWM模块

在文件 [lks32mc09x_mcpwm.c](#) 第 223 行定义.

3.31.2.19 MCPWM.StructInit()

```
void MCPWM_StructInit (
    MCPWM_InitTypeDef * MCPWM_InitStruct )
```

mcpwm配置结构体初始化

mcpwm配置结构体初始化

参数

<code>MCPWM_InitStruct</code>	指向要初始化的MCPWM_InitTypeDef结构体

在文件 `lks32mc09x_mcpwm.c` 第 118 行定义。

3.32 lks32mc09x_mcpwm.h

浏览该文件的文档。

```

00001 #ifndef __LKS32MC09X_MCPWM_H__
00002 #define __LKS32MC09X_MCPWM_H__
00003 #include "lks32mc09x_reg.h"
00004
00005 typedef struct
00006 {
00007     // 基本配置项
00008     uint16_t EN; // 模块使能, 1使能, 0不使能, 不使能时MCPWM模块会被复位, 下面的配置项无意义
00009     uint32_t COUNT; // 计数器配置, 参考MCPWM_COUNT_x, 例如MCPWM_COUNT_START
00010     uint16_t CLK_DIV; // 时钟分频配置, 参考MCPWM_CLK_DIV_x, 例如MCPWM_CLK_DIV_1
00011     uint16_t TH; // 计数器门限值 (PWM周期), 0-32767, 设置为0后cnt恒为0, 无意义
00012     uint16_t TR; // 自动更新时刻, 参考MCPWM_TR_x, 例如MCPWM_TR_T0
00013     int16_t TMR0; // PWM触发ADC事件0时间点, -32768到32767
00014     int16_t TMR1; // PWM触发ADC事件1时间点, -32768到32767
00015     int16_t TMR2; // PWM触发ADC事件2时间点, -32768到32767
00016     int16_t TMR3; // PWM触发ADC事件3时间点, -32768到32767
00017     uint16_t DTHP; // 上管死区时间设置, 0-1023
00018     uint16_t DTHN; // 下管死区时间设置, 0-1023
00019     uint16_t FLT_DIV; // fail信号滤波时钟分频, 取值范围 0-255 对应1分频到256分频
00020     uint16_t IO0; // 通道0输出IO配置, 参考MCPWM_IO_x, 例如MCPWM_IO_HH_PWM
00021     uint16_t IO1; // 通道1输出IO配置, 参考MCPWM_IO_x, 例如MCPWM_IO_HH_PWM
00022     uint16_t IO2; // 通道2输出IO配置, 参考MCPWM_IO_x, 例如MCPWM_IO_HH_PWM
00023     uint16_t IO3; // 通道3输出IO配置, 参考MCPWM_IO_x, 例如MCPWM_IO_HH_PWM
00024     uint32_t AUEN; // 自动更新使能, 参考MCPWM_AUEN_x, 例如MCPWM_AUEN_ALL
00025     uint32_t IE; // 常规中断, 参考MCPWM_IE_x, 例如MCPWM_IF_T0
00026     uint32_t EIE; // 错误中断 fail信号产生的中断, 参考MCPWM_EIE_x, 例如MCPWM_EIE_FAIL0
00027     uint32_t RE; // DMA请求信号配置
00028     uint32_t FAIL_IO; // fail和关闭输出后的IO状态配置, 参考MCPWM_FAIL_IO_x, 例如MCPWM_FAIL_IO_HIGH
00029     uint32_t FAIL0; // FAIL0信号来源配置, 参考MCPWM_FAIL0_x, 例如MCPWM_FAIL0_x
00030     uint32_t FAIL1; // FAIL1信号来源配置, 参考MCPWM_FAIL1_x, 例如MCPWM_FAIL1_x
00031 } MCPWM_InitTypeDef;
00032
00033 // 常规中断配置
00034 #define MCPWM_IF_UP BIT14 // 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。
00035 #define MCPWM_IF_TMR3 BIT13 // MCPWM0_CNT0 等于 MCPWM0_TMR3 中断事件。
00036 #define MCPWM_IF_TMR2 BIT12 // MCPWM0_CNT0 等于 MCPWM0_TMR2 中断事件。
00037 #define MCPWM_IF_TMR1 BIT11 // MCPWM0_CNT0 等于 MCPWM0_TMR1 中断事件。
00038 #define MCPWM_IF_TMR0 BIT10 // MCPWM0_CNT0 等于 MCPWM0_TMR0 中断事件。
00039 #define MCPWM_IF_TH21 BIT7 // MCPWM0_CNT0 等于 MCPWM0_TH21 中断事件。
00040 #define MCPWM_IF_TH20 BIT6 // MCPWM0_CNT0 等于 MCPWM0_TH20 中断事件。
00041 #define MCPWM_IF_TH11 BIT5 // MCPWM0_CNT0 等于 MCPWM0_TH11 中断事件。
00042 #define MCPWM_IF_TH10 BIT4 // MCPWM0_CNT0 等于 MCPWM0_TH10 中断事件。
00043 #define MCPWM_IF_TH01 BIT3 // MCPWM0_CNT0 等于 MCPWM0_TH01 中断事件。
00044 #define MCPWM_IF_TH00 BIT2 // MCPWM0_CNT0 等于 MCPWM0_TH00 中断事件。
00045 #define MCPWM_IF_T1 BIT1 // t1 事件, MCPWM0_CNT0 等于 0 中断事件。
00046 #define MCPWM_IF_T0 BIT0 // t0 事件, MCPWM0_CNT0 等于 MCPWM0_TH 中断事件。
00047 // IE和IF寄存器对应位一致
00048 #define MCPWM_IE_UP MCPWM_IF_UP
00049 #define MCPWM_IE_TMR3 MCPWM_IF_TMR3
00050 #define MCPWM_IE_TMR2 MCPWM_IF_TMR2
00051 #define MCPWM_IE_TMR1 MCPWM_IF_TMR1
00052 #define MCPWM_IE_TMR0 MCPWM_IF_TMR0
00053 #define MCPWM_IE_TH21 MCPWM_IF_TH21
00054 #define MCPWM_IE_TH20 MCPWM_IF_TH20
00055 #define MCPWM_IE_TH11 MCPWM_IF_TH11
00056 #define MCPWM_IE_TH10 MCPWM_IF_TH10
00057 #define MCPWM_IE_TH01 MCPWM_IF_TH01
00058 #define MCPWM_IE_TH00 MCPWM_IF_TH00
00059 #define MCPWM_IE_T1 MCPWM_IF_T1
00060 #define MCPWM_IE_T0 MCPWM_IF_T0
00061 // 故障中断配置
00062 #define MCPWM_EIF_FAIL0 BIT4 // FAIL1 中断源事件。
00063 #define MCPWM_EIF_FAIL1 BIT5 // FAIL0 中断源事件。
00064 #define MCPWM_EIE_FAIL0 MCPWM_EIF_FAIL0
00065 #define MCPWM_EIE_FAIL1 MCPWM_EIF_FAIL1
00066
00067 #define MCPWM_AUEN_IO3 BIT28 // MCPWM0_IO3 自动加载使能

```



```

00068 #define MCPWM_AUEN.TMR3      BIT25 // MCPWM0.TMR3 自动加载使能
00069 #define MCPWM_AUEN.TMR2      BIT24 // MCPWM0.TMR2 自动加载使能
00070 #define MCPWM_AUEN.TH31     BIT17 // MCPWM0.TH31 自动加载使能
00071 #define MCPWM_AUEN.TH30     BIT16 // MCPWM0.TH30 自动加载使能
00072 #define MCPWM_AUEN.IO012    BIT12 // MCPWM0.IO012 自动加载使能
00073 #define MCPWM_AUEN.CNT0     BIT11 // MCPWM0.CNT0 自动加载使能
00074 #define MCPWM_AUEN.TH0      BIT10 // MCPWM0.TH0 自动加载使能
00075 #define MCPWM_AUEN.TMR1     BIT9  // MCPWM0.TMR1 自动加载使能
00076 #define MCPWM_AUEN.TMR0     BIT8  // MCPWM0.TMR0 自动加载使能
00077 #define MCPWM_AUEN.TH21     BIT5  // MCPWM0.TH21 自动加载使能
00078 #define MCPWM_AUEN.TH20     BIT4  // MCPWM0.TH20 自动加载使能
00079 #define MCPWM_AUEN.TH11     BIT3  // MCPWM0.TH11 自动加载使能
00080 #define MCPWM_AUEN.TH10     BIT2  // MCPWM0.TH10 自动加载使能
00081 #define MCPWM_AUEN.TH01     BIT1  // MCPWM0.TH01 自动加载使能
00082 #define MCPWM_AUEN.TH00     BIT0  // MCPWM0.TH00 自动加载使能
00083
00084 #define MCPWM_AUEN.IO        (MCPWM_AUEN.IO012 | MCPWM_AUEN.IO3) // MCPWM0.IO 自动加载使能
00085
00086 #define MCPWM_AUEN_ALL      (MCPWM_AUEN.IO3 | MCPWM_AUEN.TMR3 | MCPWM_AUEN.TMR2 | MCPWM_AUEN.TH31 |
MCPWM_AUEN.TH30 | MCPWM_AUEN.IO012 | MCPWM_AUEN.CNT0 | MCPWM_AUEN.TH0 | MCPWM_AUEN.TMR1 | MCPWM_AUEN.TMR0 |
MCPWM_AUEN.TH21 | MCPWM_AUEN.TH20 | MCPWM_AUEN.TH11 | MCPWM_AUEN.TH10 | MCPWM_AUEN.TH01 | MCPWM_AUEN.TH00)
// 打开所有自动加载使能

00087
00088 #define MCPWM_AUEN.DEFAULT (MCPWM_AUEN_ALL ^ MCPWM_AUEN.CNT0) // 通常不会打开cnt的自动更新使能
00089
00090 // mcpwmIo状态配置位定义
00091 #define MCPWM_IO_WM          BIT7 // CH0 工作模式选择。1:边沿模式; 0:互补模式
00092 #define MCPWM_IO_PN_SW       BIT6 // CH0 的 P 和 N 通道输出互换选择。即 P 通道信号最后从 N 通道输出, N 通道的信号最后从
P 通道输出。1:互换; 0:不互换。
00093 #define MCPWM_IO_SCTRLP     BIT5 // 当 CH0_PS =1 时, 输出到 CH0 P 通道的值。1:上管打开 0:上管关闭
00094 #define MCPWM_IO_SCTRLN     BIT4 // 当 CH0_NS =1 时, 输出到 CH0 N 通道的值。1:下管打开 0:下管关闭
00095 #define MCPWM_IO_PS         BIT3 // P 来源。1:来自 CH0_SCTRLP; 0:MCPWM 实际运行系统中计数器产生。
00096 #define MCPWM_IO_NS         BIT2 // N 来源。1:来自 CH0_SCTRLN; 0:MCPWM 实际运行系统中计数器产生。
00097 #define MCPWM_IO_PP         BIT1 // P 极性选择。1 上管低有效 0 上管高有效
00098 #define MCPWM_IO_NP         BIT0 // N 极性选择。1 下管低有效 0 下管高有效
00099 // 互补模式下io配置项
00100 #define MCPWM_IO_DISABLE    0 // 不使用该通道(默认配置: 上管高有
效, 下管高有效, 正常输出PWM)
00101 #define MCPWM_IO_HH_PWM     0 // 上管高有效, 下管高有效, 正常输
出PWM
00102 #define MCPWM_IO_HH_LOW     (MCPWM_IO_SCTRLN | MCPWM_IO_PS | MCPWM_IO_NS) // 上管高有效, 下管高有效, 下管恒开 此
时pwm配置无效, 但仍然会正常计数&触发中断
00103 #define MCPWM_IO_HH_HIGH    (MCPWM_IO_SCTRLP | MCPWM_IO_PS | MCPWM_IO_NS) // 上管高有效, 下管高有效, 上管恒开 此
时pwm配置无效, 但仍然会正常计数&触发中断
00104 #define MCPWM_IO_HH_HPWM    (MCPWM_IO_NS) // 上管高有效, 下管高有效, 上管斩波
00105 #define MCPWM_IO_HH_LPWM    (MCPWM_IO_PS) // 上管高有效, 下管高有效, 下管斩波
00106 #define MCPWM_IO_HH_OFF     (MCPWM_IO_NS | MCPWM_IO_PS) // 上管高有效, 下管高有效, 上下管全关,
和MOE不同, 通过Io可以关闭任意一路pwm
00107
00108 #define MCPWM_IO_HHS_PWM    (MCPWM_IO_HH_PWM | MCPWM_IO_PN_SW) // 上管高有效, 下管高有效, 通道交换打开, 正常输
出PWM
00109 #define MCPWM_IO_HHS_LOW    (MCPWM_IO_HH_LOW | MCPWM_IO_PN_SW) // 上管高有效, 下管高有效, 通道交换打开, 下管恒开
此时pwm配置无效, 但仍然会正常计数&触发中断
00110 #define MCPWM_IO_HHS_HIGH   (MCPWM_IO_HH_HIGH | MCPWM_IO_PN_SW) // 上管高有效, 下管高有效, 通道交换打开, 上管恒开
此时pwm配置无效, 但仍然会正常计数&触发中断
00111 #define MCPWM_IO_HHS_HPWM   (MCPWM_IO_HH_HPWM | MCPWM_IO_PN_SW) // 上管高有效, 下管高有效, 通道交换打开, 上管斩波
00112 #define MCPWM_IO_HHS_LPWM   (MCPWM_IO_HH_LPWM | MCPWM_IO_PN_SW) // 上管高有效, 下管高有效, 通道交换打开, 下管斩波
00113 #define MCPWM_IO_HHS_OFF    (MCPWM_IO_HH_OFF | MCPWM_IO_PN_SW) // 上管高有效, 下管高有效, 通道交换打开, 上下管全
关, 和MOE不同, 通过Io可以关闭任意一路pwm
00114
00115 #define MCPWM_IO_HL_PWM     (MCPWM_IO_NP) // 上管高有效, 下管低
有效, 正常输出PWM
00116 #define MCPWM_IO_HL_LOW     (MCPWM_IO_NP | MCPWM_IO_SCTRLN | MCPWM_IO_PS | MCPWM_IO_NS) // 上管高有效, 下管低有
效, 下管恒开 此时pwm配置无效, 但仍然会正常计数&触发中断
00117 #define MCPWM_IO_HL_HIGH    (MCPWM_IO_NP | MCPWM_IO_SCTRLP | MCPWM_IO_PS | MCPWM_IO_NS) // 上管高有效, 下管低有
效, 上管恒开 此时pwm配置无效, 但仍然会正常计数&触发中断
00118 #define MCPWM_IO_HL_HPWM    (MCPWM_IO_NP | MCPWM_IO_NS) // 上管高有效, 下管低
有效, 上管斩波
00119 #define MCPWM_IO_HL_LPWM    (MCPWM_IO_NP | MCPWM_IO_PS) // 上管高有效, 下管低
有效, 下管斩波
00120 #define MCPWM_IO_HL_OFF     (MCPWM_IO_NP | MCPWM_IO_NS | MCPWM_IO_PS) // 上管高有效, 下管低有
效, 上下管全关, 和MOE不同, 通过Io可以关闭任意一路pwm
00121
00122 #define MCPWM_IO_HLS_PWM    (MCPWM_IO_HL_PWM | MCPWM_IO_PN_SW) // 上管高有效, 下管低有效, 通道交换打开, 正常输
出PWM
00123 #define MCPWM_IO_HLS_LOW    (MCPWM_IO_HL_LOW | MCPWM_IO_PN_SW) // 上管高有效, 下管低有效, 通道交换打开, 下管恒开
此时pwm配置无效, 但仍然会正常计数&触发中断
00124 #define MCPWM_IO_HLS_HIGH   (MCPWM_IO_HL_HIGH | MCPWM_IO_PN_SW) // 上管高有效, 下管低有效, 通道交换打开, 上管恒开
此时pwm配置无效, 但仍然会正常计数&触发中断
00125 #define MCPWM_IO_HLS_HPWM   (MCPWM_IO_HL_HPWM | MCPWM_IO_PN_SW) // 上管高有效, 下管低有效, 通道交换打开, 上管斩波
00126 #define MCPWM_IO_HLS_LPWM   (MCPWM_IO_HL_LPWM | MCPWM_IO_PN_SW) // 上管高有效, 下管低有效, 通道交换打开, 下管斩波
00127 #define MCPWM_IO_HLS_OFF    (MCPWM_IO_HL_OFF | MCPWM_IO_PN_SW) // 上管高有效, 下管低有效, 通道交换打开, 上下管全
关, 和MOE不同, 通过Io可以关闭任意一路pwm
00128
00129 #define MCPWM_IO_LH_PWM     (MCPWM_IO_PP) // 上管低有效, 下管高
有效, 正常输出PWM
00130 #define MCPWM_IO_LH_LOW     (MCPWM_IO_PP | MCPWM_IO_SCTRLN | MCPWM_IO_PS | MCPWM_IO_NS) // 上管低有效, 下管高有

```

```

效, 下管恒开 此时pwm配置无效, 但仍然会正常计数&触发中断
00131 #define MCPWM_IO_LH_HIGH (MCPWM_IO_PP | MCPWM_IO_SCTRLP | MCPWM_IO_PS | MCPWM_IO_NS) // 上管低有效, 下管高有
效, 上管恒开 此时pwm配置无效, 但仍然会正常计数&触发中断
00132 #define MCPWM_IO_LH_HPWM (MCPWM_IO_PP | MCPWM_IO_NS) // 上管低有效, 下管高
有效, 上管斩波
00133 #define MCPWM_IO_LH_LPWM (MCPWM_IO_PP | MCPWM_IO_PS) // 上管低有效, 下管高
有效, 下管斩波
00134 #define MCPWM_IO_LH_OFF (MCPWM_IO_PP | MCPWM_IO_NS | MCPWM_IO_PS) // 上管低有效, 下管高有
效, 上下管全关, 和MOE不同, 通过IO可以关闭任意一路pwm
00135
00136 #define MCPWM_IO_LHS_PWM (MCPWM_IO_LH_PWM | MCPWM_IO_PN_SW) // 上管低有效, 下管高有效, 通道交换打开, 正常输
出PWM
00137 #define MCPWM_IO_LHS_LOW (MCPWM_IO_LH_LOW | MCPWM_IO_PN_SW) // 上管低有效, 下管高有效, 通道交换打开, 下管恒开
此时pwm配置无效, 但仍然会正常计数&触发中断
00138 #define MCPWM_IO_LHS_HIGH (MCPWM_IO_LH_HIGH | MCPWM_IO_PN_SW) // 上管低有效, 下管高有效, 通道交换打开, 上管恒开
此时pwm配置无效, 但仍然会正常计数&触发中断
00139 #define MCPWM_IO_LHS_HPWM (MCPWM_IO_LH_HPWM | MCPWM_IO_PN_SW) // 上管低有效, 下管高有效, 通道交换打开, 上管斩波
00140 #define MCPWM_IO_LHS_LPWM (MCPWM_IO_LH_LPWM | MCPWM_IO_PN_SW) // 上管低有效, 下管高有效, 通道交换打开, 下管斩波
00141 #define MCPWM_IO_LHS_OFF (MCPWM_IO_LH_OFF | MCPWM_IO_PN_SW) // 上管低有效, 下管高有效, 通道交换打开, 上下管全
关, 和MOE不同, 通过IO可以关闭任意一路pwm
00142
00143 #define MCPWM_IO_LL_PWM (MCPWM_IO_PP | MCPWM_IO_NP) // 上
管低有效, 下管低有效, 正常输出PWM
00144 #define MCPWM_IO_LL_LOW (MCPWM_IO_PP | MCPWM_IO_NP | MCPWM_IO_SCTRLN | MCPWM_IO_PS | MCPWM_IO_NS) // 上管
低有效, 下管低有效, 下管恒开 此时pwm配置无效, 但仍然会正常计数&触发中断
00145 #define MCPWM_IO_LL_HIGH (MCPWM_IO_PP | MCPWM_IO_NP | MCPWM_IO_SCTRLP | MCPWM_IO_PS | MCPWM_IO_NS) // 上管
低有效, 下管低有效, 上管恒开 此时pwm配置无效, 但仍然会正常计数&触发中断
00146 #define MCPWM_IO_LL_HPWM (MCPWM_IO_PP | MCPWM_IO_NP | MCPWM_IO_NS) // 上
管低有效, 下管低有效, 上管斩波
00147 #define MCPWM_IO_LL_LPWM (MCPWM_IO_PP | MCPWM_IO_NP | MCPWM_IO_PS) // 上
管低有效, 下管低有效, 下管斩波
00148 #define MCPWM_IO_LL_OFF (MCPWM_IO_PP | MCPWM_IO_NP | MCPWM_IO_NS | MCPWM_IO_PS) // 上管
低有效, 下管低有效, 上下管全关, 和MOE不同, 通过IO可以关闭任意一路pwm
00149
00150 #define MCPWM_IO_LLS_PWM (MCPWM_IO_LL_PWM | MCPWM_IO_PN_SW) // 上管低有效, 下管低有效, 通道交换打开, 正常输
出PWM
00151 #define MCPWM_IO_LLS_LOW (MCPWM_IO_LL_LOW | MCPWM_IO_PN_SW) // 上管低有效, 下管低有效, 通道交换打开, 下管恒开
此时pwm配置无效, 但仍然会正常计数&触发中断
00152 #define MCPWM_IO_LLS_HIGH (MCPWM_IO_LL_HIGH | MCPWM_IO_PN_SW) // 上管低有效, 下管低有效, 通道交换打开, 上管恒开
此时pwm配置无效, 但仍然会正常计数&触发中断
00153 #define MCPWM_IO_LLS_HPWM (MCPWM_IO_LL_HPWM | MCPWM_IO_PN_SW) // 上管低有效, 下管低有效, 通道交换打开, 上管斩波
00154 #define MCPWM_IO_LLS_LPWM (MCPWM_IO_LL_LPWM | MCPWM_IO_PN_SW) // 上管低有效, 下管低有效, 通道交换打开, 下管斩波
00155 #define MCPWM_IO_LLS_OFF (MCPWM_IO_LL_OFF | MCPWM_IO_PN_SW) // 上管低有效, 下管低有效, 通道交换打开, 上下管全
关, 和MOE不同, 通过IO可以关闭任意一路pwm
00156
00157 // 计数器配置
00158 #define MCPWM_COUNT_OFF 0 // 计数器停止计数
00159 #define MCPWM_COUNT_START BIT31 // 计数器立即开始计数
00160 #define MCPWM_COUNT_EVT_TIMER0_CMP0 BIT0 // timer0的通道0事件触发MCPWM开始计数
00161 #define MCPWM_COUNT_EVT_TIMER0_CMP1 BIT1 // timer0的通道1事件触发MCPWM开始计数
00162 #define MCPWM_COUNT_EVT_TIMER1_CMP0 BIT2 // timer1的通道0事件触发MCPWM开始计数
00163 #define MCPWM_COUNT_EVT_TIMER1_CMP1 BIT3 // timer1的通道1事件触发MCPWM开始计数
00164 #define MCPWM_COUNT_EVT_TIMER2_CMP0 BIT4 // timer2的通道0事件触发MCPWM开始计数
00165 #define MCPWM_COUNT_EVT_TIMER2_CMP1 BIT5 // timer2的通道1事件触发MCPWM开始计数
00166
00167 #define MCPWM_CLK_DIV_1 0 // 时钟1分频
00168 #define MCPWM_CLK_DIV_2 1 // 时钟2分频
00169 #define MCPWM_CLK_DIV_4 2 // 时钟4分频
00170 #define MCPWM_CLK_DIV_8 3 // 时钟8分频
00171 #define MCPWM_CLK_DIV_16 4 // 时钟16分频
00172 #define MCPWM_CLK_DIV_32 5 // 时钟32分频
00173 #define MCPWM_CLK_DIV_64 6 // 时钟64分频
00174 #define MCPWM_CLK_DIV_128 7 // 时钟128分频
00175 // 自动更新配置, 通常配置为溢出时刻进行更新
00176 #define MCPWM_TR_T0 BIT4 // cnt溢出时刻触发自动更新 (CNT起点)
00177 #define MCPWM_TR_T1 BIT5 // cnt过零时刻触发自动更新
00178
00179 // 注意, 这里的fail信号配置会参考IO里配置的通道交换和极性配置进行进一步交换
00180 #define MCPWM_FAIL_IO_HIGH 0xaa // 触发fail信号后上管打开, 下管关闭
00181 #define MCPWM_FAIL_IO_LOW 0x55 // 触发fail信号后上管关闭, 下管打开
00182 #define MCPWM_FAIL_IO_OFF 0 // 触发fail信号后上下管全关
00183
00184 // fail信号来源配置
00185 #define MCPWM_FAIL0_OFF (0) // 关闭fail0
00186 #define MCPWM_FAIL0_CMP0_H (BIT8 | BIT4) // 使用cmp0的高电平作为fail0的输入
00187 #define MCPWM_FAIL0_CMP0_L (BIT8 | BIT4 | BIT2) // 使用cmp0的低电平作为fail0的输入
00188 #define MCPWM_FAIL0_BKIN0_H (BIT4) // 使用BKIN0的高电平作为fail0的输入
00189 #define MCPWM_FAIL0_BKIN0_L (BIT4 | BIT2) // 使用BKIN0的低电平作为fail0的输入
00190 #define MCPWM_FAIL0_CLU_OUT0_H (BIT9 | BIT4) // 使用CLU_OUT0的高电平作为fail0的输入
00191 #define MCPWM_FAIL0_CLU_OUT0_L (BIT9 | BIT4 | BIT2) // 使用CLU_OUT0的低电平作为fail0的输入
00192 #define MCPWM_FAIL0_CLU_OUT1_H (BIT9 | BIT8 | BIT4) // 使用CLU_OUT1的高电平作为fail0的输入
00193 #define MCPWM_FAIL0_CLU_OUT1_L (BIT9 | BIT8 | BIT4 | BIT2) // 使用CLU_OUT1的低电平作为fail0的输入
00194
00195 // fail信号来源配置
00196 #define MCPWM_FAIL1_OFF (0) // 关闭fail1
00197 #define MCPWM_FAIL1_CMP1_H (BIT10 | BIT5) // 使用cmp1的高电平作为fail1的输入
00198 #define MCPWM_FAIL1_CMP1_L (BIT10 | BIT5 | BIT3) // 使用cmp1的低电平作为fail1的输入

```

```

00199 #define MCPWM_FAIL1_BKIN1_L (BIT5) // 使用BKIN1的高电平作为fail1的输入
00200 #define MCPWM_FAIL1_BKIN1_L (BIT5 | BIT3) // 使用BKIN1的低电平作为fail1的输入
00201 #define MCPWM_FAIL1_CLU_OUT2_H (BIT11 | BIT5) // 使用CLU_OUT2的高电平作为fail1的输入
00202 #define MCPWM_FAIL1_CLU_OUT2_L (BIT11 | BIT5 | BIT3) // 使用CLU_OUT2的低电平作为fail1的输入
00203 #define MCPWM_FAIL1_CLU_OUT3_H (BIT11 | BIT10 | BIT5) // 使用CLU_OUT3的高电平作为fail1的输入
00204 #define MCPWM_FAIL1_CLU_OUT3_L (BIT11 | BIT10 | BIT5 | BIT3) // 使用CLU_OUT3的低电平作为fail1的输入
00205
00210 void MCPWM_Enable(MCPWM_TypeDef *MCPWMx);
00215 void MCPWM_Disable(MCPWM_TypeDef *MCPWMx);
00220 void MCPWM_Reset(MCPWM_TypeDef *MCPWMx);
00224 void MCPWM_Init(MCPWM_TypeDef *MCPWMx, MCPWM_InitTypeDef *MCPWM_InitStruct);
00228 void MCPWM_StructInit(MCPWM_TypeDef *MCPWMx, MCPWM_InitTypeDef *MCPWM_InitStruct);
00234 void MCPWM_SetOutputState(MCPWM_TypeDef *MCPWMx, uint8_t state);
00242 void MCPWM_SetOutputVal(MCPWM_TypeDef *MCPWMx, uint8_t chn, int16_t p, int16_t n);
00251 void MCPWM_SetTMR(MCPWM_TypeDef *MCPWMx, int16_t tmr0, int16_t tmr1, int16_t tmr2, int16_t tmr3);
00261 void MCPWM_SetIOConfig(MCPWM_TypeDef *MCPWMx, uint16_t io0, uint16_t io1, uint16_t io2, uint16_t io3);
00266 void MCPWM_StartCount(MCPWM_TypeDef *MCPWMx);
00271 void MCPWM_StopCount(MCPWM_TypeDef *MCPWMx);
00278 uint16_t MCPWM_GetIRQFlag(MCPWM_TypeDef *MCPWMx, uint32_t MCPWM_IF_x);
00286 uint16_t MCPWM_GetIfFlag(MCPWM_TypeDef *MCPWMx, uint32_t MCPWM_IF_x);
00293 uint16_t MCPWM_GetEIFFlag(MCPWM_TypeDef *MCPWMx, uint32_t MCPWM_IF_x);
00300 uint16_t MCPWM_GetErrorIRQFlag(MCPWM_TypeDef *MCPWMx, uint32_t MCPWM{EIF_x);
00306 void MCPWM_ClearIRQFlag(MCPWM_TypeDef *MCPWMx, uint32_t MCPWM_IF_x);
00312 void MCPWM_ClearErrorIRQFlag(MCPWM_TypeDef *MCPWMx, uint32_t MCPWM{EIF_x);
00313
00319 int16_t MCPWM_GetCountTh(MCPWM_TypeDef *MCPWMx);
00325 int16_t MCPWM_GetCountVal(MCPWM_TypeDef *MCPWMx);
00326
00327 // 通过宏定义简化中断执行时间
00328 // 中断标志位判断
00329 #define MCPWM_IRQ_UP(MCPWMx) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_UP) // 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。
00330 #define MCPWM_IRQ_TMR3(MCPWMx) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_TMR3) // MCPWM0_CNT0 等于
MCPWM0_TMR3 中断事件。
00331 #define MCPWM_IRQ_TMR2(MCPWMx) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_TMR2) // MCPWM0_CNT0 等于
MCPWM0_TMR2 中断事件。
00332 #define MCPWM_IRQ_TMR1(MCPWMx) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_TMR1) // MCPWM0_CNT0 等于
MCPWM0_TMR1 中断事件。
00333 #define MCPWM_IRQ_TMR0(MCPWMx) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_TMR0) // MCPWM0_CNT0 等于
MCPWM0_TMR0 中断事件。
00334 #define MCPWM_IRQ_TH21(MCPWMx) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_TH21) // MCPWM0_CNT0 等于
MCPWM0_TH21 中断事件。
00335 #define MCPWM_IRQ_TH20(MCPWMx) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_TH20) // MCPWM0_CNT0 等于
MCPWM0_TH20 中断事件。
00336 #define MCPWM_IRQ_TH11(MCPWMx) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_TH11) // MCPWM0_CNT0 等于
MCPWM0_TH11 中断事件。
00337 #define MCPWM_IRQ_TH10(MCPWMx) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_TH10) // MCPWM0_CNT0 等于
MCPWM0_TH10 中断事件。
00338 #define MCPWM_IRQ_TH01(MCPWMx) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_TH01) // MCPWM0_CNT0 等于
MCPWM0_TH01 中断事件。
00339 #define MCPWM_IRQ_TH00(MCPWMx) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_TH00) // MCPWM0_CNT0 等于
MCPWM0_TH00 中断事件。
00340 #define MCPWM_IRQ_T1(MCPWMx) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_T1) // t1 事件, MCPWM0_CNT0 等于
0 中断事件。
00341 #define MCPWM_IRQ_T0(MCPWMx) (MCPWMx->IF0 & MCPWM->IE0 & MCPWM_IF_T0) // t0 事件, MCPWM0_CNT0 等于
MCPWM0_TH 中断事件。
00342 #define MCPWM_IRQ_FAIL0(MCPWMx) (MCPWMx->EIF & MCPWM->EIE & MCPWM{EIF_FAIL0) // FAIL1 中断源事件。
00343 #define MCPWM_IRQ_FAIL1(MCPWMx) (MCPWMx->EIF & MCPWM->EIE & MCPWM{EIF_FAIL1) // FAIL0 中断源事件。
00344
00345 // 中断标志位获取, 和MCPWM_IRQ_x的区别是不进行中断使能位的判断
00346 #define MCPWM_GET_IRQ_UP(MCPWMx) (MCPWMx->IF0 & MCPWM_IF_UP) // 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。
00347 #define MCPWM_GET_IRQ_TMR3(MCPWMx) (MCPWMx->IF0 & MCPWM_IF_TMR3) // MCPWM0_CNT0 等于 MCPWM0_TMR3 中断事
件。
00348 #define MCPWM_GET_IRQ_TMR2(MCPWMx) (MCPWMx->IF0 & MCPWM_IF_TMR2) // MCPWM0_CNT0 等于 MCPWM0_TMR2 中断事
件。
00349 #define MCPWM_GET_IRQ_TMR1(MCPWMx) (MCPWMx->IF0 & MCPWM_IF_TMR1) // MCPWM0_CNT0 等于 MCPWM0_TMR1 中断事
件。
00350 #define MCPWM_GET_IRQ_TMR0(MCPWMx) (MCPWMx->IF0 & MCPWM_IF_TMR0) // MCPWM0_CNT0 等于 MCPWM0_TMR0 中断事
件。
00351 #define MCPWM_GET_IRQ_TH21(MCPWMx) (MCPWMx->IF0 & MCPWM_IF_TH21) // MCPWM0_CNT0 等于 MCPWM0_TH21 中断事
件。
00352 #define MCPWM_GET_IRQ_TH20(MCPWMx) (MCPWMx->IF0 & MCPWM_IF_TH20) // MCPWM0_CNT0 等于 MCPWM0_TH20 中断事
件。
00353 #define MCPWM_GET_IRQ_TH11(MCPWMx) (MCPWMx->IF0 & MCPWM_IF_TH11) // MCPWM0_CNT0 等于 MCPWM0_TH11 中断事
件。
00354 #define MCPWM_GET_IRQ_TH10(MCPWMx) (MCPWMx->IF0 & MCPWM_IF_TH10) // MCPWM0_CNT0 等于 MCPWM0_TH10 中断事
件。
00355 #define MCPWM_GET_IRQ_TH01(MCPWMx) (MCPWMx->IF0 & MCPWM_IF_TH01) // MCPWM0_CNT0 等于 MCPWM0_TH01 中断事
件。
00356 #define MCPWM_GET_IRQ_TH00(MCPWMx) (MCPWMx->IF0 & MCPWM_IF_TH00) // MCPWM0_CNT0 等于 MCPWM0_TH00 中断事
件。
00357 #define MCPWM_GET_IRQ_T1(MCPWMx) (MCPWMx->IF0 & MCPWM_IF_T1) // t1 事件, MCPWM0_CNT0 等于 0 中断事
件。
00358 #define MCPWM_GET_IRQ_T0(MCPWMx) (MCPWMx->IF0 & MCPWM_IF_T0) // t0 事件, MCPWM0_CNT0 等于 MCPWM0_TH
中断事件。
00359 #define MCPWM_GET_IRQ_FAIL0(MCPWMx) (MCPWMx->EIF & MCPWM{EIF_FAIL0) // FAIL1 中断源事件。

```

```

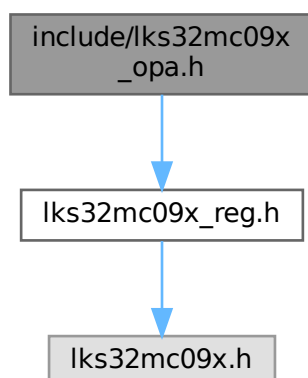
00360 #define MCPWM_GET_IRQ_FAIL1(MCPWMx) (MCPWMx->EIF & MCPWM{EIF_FAIL1}) // FAIL0 中断源事件。
00361
00362 // 中断标志位清除
00363 #define MCPWM_IRQ_CLEAR_UP(MCPWMx)      MCPWMx->IF0 = MCPWM{EIF_UP} // 时基 0 更新事件
00364 // MCPWM0.TH0/MCPWM0.TH00~MCPWM0.TH21/MCPWM0.TMR 等寄存器更新到 MCPWM 实际运行系统的中断事件。
00364 #define MCPWM_IRQ_CLEAR_TMR3(MCPWMx)    MCPWMx->IF0 = MCPWM{EIF_TMR3} // MCPWM0.CNT0 等于 MCPWM0.TMR3 中断事
00365 件。
00365 #define MCPWM_IRQ_CLEAR_TMR2(MCPWMx)    MCPWMx->IF0 = MCPWM{EIF_TMR2} // MCPWM0.CNT0 等于 MCPWM0.TMR2 中断事
00366 件。
00366 #define MCPWM_IRQ_CLEAR_TMR1(MCPWMx)    MCPWMx->IF0 = MCPWM{EIF_TMR1} // MCPWM0.CNT0 等于 MCPWM0.TMR1 中断事
00367 件。
00367 #define MCPWM_IRQ_CLEAR_TMR0(MCPWMx)    MCPWMx->IF0 = MCPWM{EIF_TMR0} // MCPWM0.CNT0 等于 MCPWM0.TMR0 中断事
00368 件。
00368 #define MCPWM_IRQ_CLEAR_TH21(MCPWMx)    MCPWMx->IF0 = MCPWM{EIF_TH21} // MCPWM0.CNT0 等于 MCPWM0.TH21 中断事
00369 件。
00369 #define MCPWM_IRQ_CLEAR_TH20(MCPWMx)    MCPWMx->IF0 = MCPWM{EIF_TH20} // MCPWM0.CNT0 等于 MCPWM0.TH20 中断事
00370 件。
00370 #define MCPWM_IRQ_CLEAR_TH11(MCPWMx)    MCPWMx->IF0 = MCPWM{EIF_TH11} // MCPWM0.CNT0 等于 MCPWM0.TH11 中断事
00371 件。
00371 #define MCPWM_IRQ_CLEAR_TH10(MCPWMx)    MCPWMx->IF0 = MCPWM{EIF_TH10} // MCPWM0.CNT0 等于 MCPWM0.TH10 中断事
00372 件。
00372 #define MCPWM_IRQ_CLEAR_TH01(MCPWMx)    MCPWMx->IF0 = MCPWM{EIF_TH01} // MCPWM0.CNT0 等于 MCPWM0.TH01 中断事
00373 件。
00373 #define MCPWM_IRQ_CLEAR_TH00(MCPWMx)    MCPWMx->IF0 = MCPWM{EIF_TH00} // MCPWM0.CNT0 等于 MCPWM0.TH00 中断事
00374 件。
00374 #define MCPWM_IRQ_CLEAR_T1(MCPWMx)      MCPWMx->IF0 = MCPWM{EIF_T1} // t1 事件, MCPWM0.CNT0 等于 0 中断事
00375 件。
00375 #define MCPWM_IRQ_CLEAR_T0(MCPWMx)      MCPWMx->IF0 = MCPWM{EIF_T0} // t0 事件, MCPWM0.CNT0 等于 MCPWM0.TH
00376 #define MCPWM_IRQ_CLEAR_FAIL0(MCPWMx)  MCPWMx->EIF = MCPWM{EIF_FAIL0} // FAIL1 中断源事件。
00377 #define MCPWM_IRQ_CLEAR_FAIL1(MCPWMx)  MCPWMx->EIF = MCPWM{EIF_FAIL1} // FAIL0 中断源事件。
00378 #endif

```

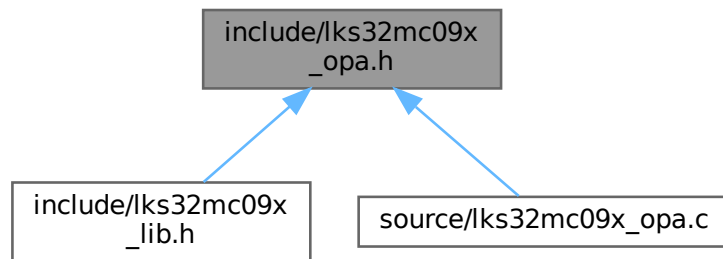
3.33 include/lks32mc09x_opa.h 文件参考

#include "lks32mc09x_reg.h"

lks32mc09x_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：



结构体

- struct **OPA_InitTypeDef**
OPA初始化结构体句柄，初始化时请定义该句柄，并用其它地址来传参

宏定义

- #define **OPA_CHANNEL_0** 0
运放0
- #define **OPA_CHANNEL_1** 1
运放1
- #define **OPA_CHANNEL_2** 2
运放2
- #define **OPA_CHANNEL_3** 3
运放3
- #define **OPA_GAIN_33** 0x00
OPA 增益(4个运放可以独立设置)
- #define **OPA_GAIN_16** 0x01
反馈电阻640k:40k
- #define **OPA_GAIN_8** 0x02
反馈电阻320k:40k
- #define **OPA_GAIN_4** 0x03
反馈电阻160k:40k
- #define **OPA_IT_1_0** 0x00
OPA 偏置电流(4个运放共用同一个偏置电流)
- #define **OPA_IT_1_2** 0x01
OPA 偏置电流×1.2
- #define **OPA_IT_1_5** 0x02
OPA 偏置电流×1.5
- #define **OPA_IT_2_0** 0x03
OPA 偏置电流×2
- #define **OPA_OUT_CMP_OUT0P** 0x00
OPA_OUT_CMP信号来源配置

- `#define OPA_OUT_CMP_OUT1P 0x01`
将OPA1输出正端输出到OPA_OUT_CMP
- `#define OPA_OUT_CMP_OUT2P 0x02`
将OPA2输出正端输出到OPA_OUT_CMP
- `#define OPA_OUT_CMP_OUT3P 0x03`
将OPA3输出正端输出到OPA_OUT_CMP
- `#define OPA_OUT_CMP_OUT0N 0x04`
将OPA0输出负端输出到OPA_OUT_CMP
- `#define OPA_OUT_CMP_OUT1N 0x05`
将OPA1输出负端输出到OPA_OUT_CMP
- `#define OPA_OUT_CMP_OUT2N 0x06`
将OPA2输出负端输出到OPA_OUT_CMP
- `#define OPA_OUT_CMP_OUT3N 0x07`
将OPA3输出负端输出到OPA_OUT_CMP

函数

- `void OPA_Init (uint32_t OPA_CHANNEL_x, OPA_InitTypeDef *OPA_InitStruct)`
OPA 初始化
- `void OPA_StructInit (OPA_InitTypeDef *OPA_InitStruct)`
OPA初始化结构体初始化
- `void OPA_Enable (uint32_t OPA_CHANNEL_x)`
OPA 使能
- `void OPA_Disable (uint32_t OPA_CHANNEL_x)`
OPA 失能
- `void OPA_SetGain (uint32_t OPA_CHANNEL_x, uint32_t OPA_GAIN_x)`
OPA 增益设置
- `void OPA_SetItt (uint32_t OPA_IT_x)`
OPA 偏置电流设置
- `void OPA_OutCmpSignal (uint32_t OPA_OUT_CMP_x, uint32_t Out2Gpio)`
OPA 输出信号选择（使能输出到GPIO）

3.33.1 宏定义说明

3.33.1.1 OPA_CHANNEL_0

```
#define OPA_CHANNEL_0 0
```

运放0

在文件 [lks32mc09x_opa.h](#) 第 6 行定义.

3.33.1.2 OPA_CHANNEL_1

```
#define OPA_CHANNEL_1 1
```

运放1

在文件 [lks32mc09x_opa.h](#) 第 7 行定义.

3.33.1.3 OPA_CHANNEL_2

```
#define OPA_CHANNEL_2 2
```

运放2

在文件 [lks32mc09x_opa.h](#) 第 8 行定义.

3.33.1.4 OPA_CHANNEL_3

```
#define OPA_CHANNEL_3 3
```

运放3

在文件 [lks32mc09x_opa.h](#) 第 9 行定义.

3.33.1.5 OPA_GAIN_16

```
#define OPA_GAIN_16 0x01
```

反馈电阻640k:40k

在文件 [lks32mc09x_opa.h](#) 第 24 行定义.

3.33.1.6 OPA_GAIN_33

```
#define OPA_GAIN_33 0x00
```

OPA 增益(4个运放可以独立设置)

反馈电阻660k:20k

在文件 [lks32mc09x_opa.h](#) 第 23 行定义.

3.33.1.7 OPA_GAIN_4

```
#define OPA_GAIN_4 0x03
```

反馈电阻160k:40k

在文件 [lks32mc09x_opa.h](#) 第 26 行定义.

3.33.1.8 OPA_GAIN_8

```
#define OPA_GAIN_8 0x02
```

反馈电阻320k:40k

在文件 [lks32mc09x_opa.h](#) 第 25 行定义.

3.33.1.9 OPA_IT_1_0

```
#define OPA_IT_1_0 0x00
```

OPA 偏置电流(4个运放共用同一个偏置电流)

OPA 偏置电流×1

在文件 [lks32mc09x_opa.h](#) 第 31 行定义.

3.33.1.10 OPA_IT_1_2

```
#define OPA_IT_1_2 0x01
```

OPA 偏置电流×1.2

在文件 [lks32mc09x_opa.h](#) 第 32 行定义.

3.33.1.11 OPA_IT_1_5

```
#define OPA_IT_1_5 0x02
```

OPA 偏置电流×1.5

在文件 [lks32mc09x_opa.h](#) 第 33 行定义.

3.33.1.12 OPA_IT_2_0

```
#define OPA_IT_2_0 0x03
```

OPA 偏置电流×2

在文件 [lks32mc09x_opa.h](#) 第 34 行定义.

3.33.1.13 OPA_OUT_CMP_OUT0N

```
#define OPA_OUT_CMP_OUT0N 0x04
```

将OPA0输出负端输出到OPA_OUT_CMP

在文件 [lks32mc09x_opa.h](#) 第 43 行定义.

3.33.1.14 OPA_OUT_CMP_OUT0P

```
#define OPA_OUT_CMP_OUT0P 0x00
```

OPA_OUT_CMP信号来源配置

将OPA0输出正端输出到OPA_OUT_CMP

在文件 [lks32mc09x_opa.h](#) 第 39 行定义.

3.33.1.15 OPA_OUT_CMP_OUT1N

```
#define OPA_OUT_CMP_OUT1N 0x05
```

将OPA1输出负端输出到OPA_OUT_CMP
在文件 [lks32mc09x_opa.h](#) 第 44 行定义.

3.33.1.16 OPA_OUT_CMP_OUT1P

```
#define OPA_OUT_CMP_OUT1P 0x01
```

将OPA1输出正端输出到OPA_OUT_CMP
在文件 [lks32mc09x_opa.h](#) 第 40 行定义.

3.33.1.17 OPA_OUT_CMP_OUT2N

```
#define OPA_OUT_CMP_OUT2N 0x06
```

将OPA2输出负端输出到OPA_OUT_CMP
在文件 [lks32mc09x_opa.h](#) 第 45 行定义.

3.33.1.18 OPA_OUT_CMP_OUT2P

```
#define OPA_OUT_CMP_OUT2P 0x02
```

将OPA2输出正端输出到OPA_OUT_CMP
在文件 [lks32mc09x_opa.h](#) 第 41 行定义.

3.33.1.19 OPA_OUT_CMP_OUT3N

```
#define OPA_OUT_CMP_OUT3N 0x07
```

将OPA3输出负端输出到OPA_OUT_CMP
在文件 [lks32mc09x_opa.h](#) 第 46 行定义.

3.33.1.20 OPA_OUT_CMP_OUT3P

```
#define OPA_OUT_CMP_OUT3P 0x03
```

将OPA3输出正端输出到OPA_OUT_CMP
在文件 [lks32mc09x_opa.h](#) 第 42 行定义.

3.33.2 函数说明

3.33.2.1 OPA_Disable()

```
void OPA_Disable (
    uint32_t OPA_CHANNEL_x )
```

OPA 失能

参数

<i>OPA</i> ↔ <i>CHANNEL_x</i>	OPA通道

在文件 [lks32mc09x_opa.c](#) 第 61 行定义.

这是这个函数的调用关系图:



3.33.2.2 OPA.Enable()

```
void OPA_Enable (
    uint32_t OPA_CHANNEL_x )
```

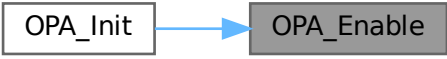
OPA 使能

参数

<i>OPA</i> ↔ <i>CHANNEL_x</i>	OPA通道

在文件 [lks32mc09x_opa.c](#) 第 37 行定义.

这是这个函数的调用关系图:



3.33.2.3 OPA.Init()

```
void OPA_Init (
    uint32_t OPA_CHANNEL_x,
    OPA_InitTypeDef * OPA_InitStruct )
```

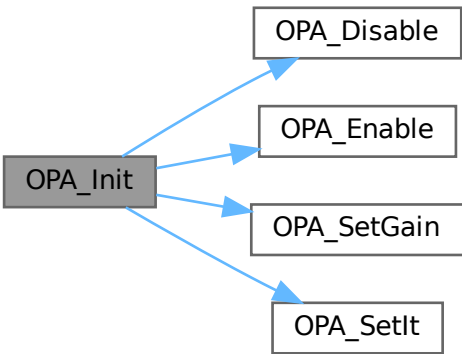
OPA 初始化

参数

<i>OPA_↔CHANNEL_x</i>	OPA通道
<i>OPA_InitStruct</i>	OPA初始化结构体指针

在文件 [lks32mc09x_opa.c](#) 第 8 行定义.

函数调用图:



3.33.2.4 OPA_OutCmpSignal()

```
void OPA_OutCmpSignal (
    uint32_t OPA_OUT_CMP_x,
    uint32_t Out2Gpio )
```

OPA 输出信号选择（使能输出到GPIO）

参数

<i>OPA_OUT_↔CMP_x</i>	OPA_OUT_CMP信号来源配置
<i>Out2Gpio</i>	OPA_OUT_CMP输出到IO使能，P2.7

在文件 [lks32mc09x_opa.c](#) 第 121 行定义.

3.33.2.5 OPA_SetGain()

```
void OPA_SetGain (
    uint32_t OPA_CHANNEL_x,
    uint32_t OPA_GAIN_x )
```

OPA 增益设置

参数

<i>OPA</i> ↔ <i>CHANNEL_x</i>	OPA通道
<i>OPA_GAIN_x</i>	OPA增益

在文件 [lks32mc09x_opa.c](#) 第 86 行定义.

这是这个函数的调用关系图:



3.33.2.6 OPA_SetIt()

```
void OPA_SetIt (
    uint32_t OPA_IT_x )
```

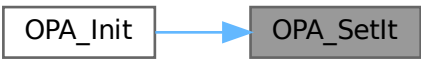
OPA 偏置电流设置

参数

<i>OPA</i> ↔ <i>CHANNEL_x</i>	OPA通道
<i>OPA_IT_x</i>	OPA偏置电流
<i>OPA_IT_x</i>	OPA偏置电流

在文件 [lks32mc09x_opa.c](#) 第 110 行定义.

这是这个函数的调用关系图:



3.33.2.7 OPA_StructInit()

```
void OPA_StructInit (
    OPA_InitTypeDef * OPA_InitStruct )
```

OPA初始化结构体初始化

在文件 `lks32mc09x_opa.c` 第 25 行定义.

3.34 lks32mc09x_opa.h

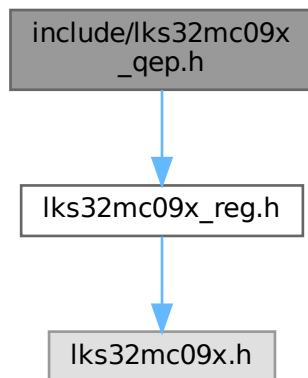
浏览该文件的文档.

```
00001 #ifndef __LKS32MC09X_OPA_H__
00002 #define __LKS32MC09X_OPA_H__
00003
00004 #include "lks32mc09x_reg.h"
00005
00006 #define OPA_CHANNEL_0 0
00007 #define OPA_CHANNEL_1 1
00008 #define OPA_CHANNEL_2 2
00009 #define OPA_CHANNEL_3 3
00013 typedef struct
00014 {
00015     uint32_t OPA_IT;
00016     uint32_t OPA_Gain;
00017     uint32_t OPA_CLEna;
00018 } OPA_InitTypeDef;
00019
00023 #define OPA_GAIN_33 0x00
00024 #define OPA_GAIN_16 0x01
00025 #define OPA_GAIN_8 0x02
00026 #define OPA_GAIN_4 0x03
00027
00031 #define OPA_IT_1_0 0x00
00032 #define OPA_IT_1_2 0x01
00033 #define OPA_IT_1_5 0x02
00034 #define OPA_IT_2_0 0x03
00035
00039 #define OPA_OUT_CMP_OUT0P 0x00
00040 #define OPA_OUT_CMP_OUT1P 0x01
00041 #define OPA_OUT_CMP_OUT2P 0x02
00042 #define OPA_OUT_CMP_OUT3P 0x03
00043 #define OPA_OUT_CMP_OUT0N 0x04
00044 #define OPA_OUT_CMP_OUT1N 0x05
00045 #define OPA_OUT_CMP_OUT2N 0x06
00046 #define OPA_OUT_CMP_OUT3N 0x07
00052 void OPA_Init(uint32_t OPA_CHANNEL_x, OPA_InitTypeDef *OPA_InitStruct);
00053
00057 void OPA_StructInit(OPA_InitTypeDef *OPA_InitStruct);
00062 void OPA_Enable(uint32_t OPA_CHANNEL_x);
00067 void OPA_Disable(uint32_t OPA_CHANNEL_x);
00073 void OPA_SetGain(uint32_t OPA_CHANNEL_x, uint32_t OPA_GAIN_x);
00079 void OPA_SetIt(uint32_t OPA_IT_x);
00085 void OPA_OutCmpSignal(uint32_t OPA_OUT_CMP_x, uint32_t Out2Gpio);
00086 #endif
```

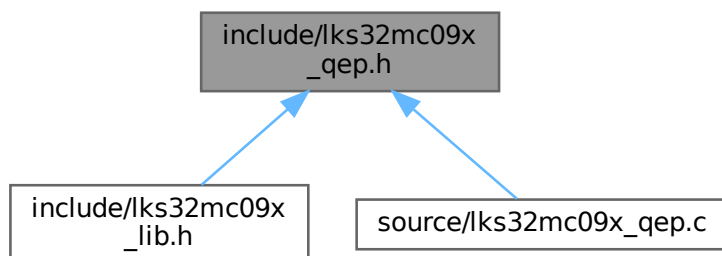
3.35 include/lks32mc09x_qep.h 文件参考

```
#include "lks32mc09x_reg.h"
```

lks32mc09x_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:



结构体

- struct [QEP_InitTypeDef](#)
QEP初始化结构体定义

宏定义

- #define [QEP_CLK_DIV1](#) 0x00
QEP时钟分频宏定义
- #define [QEP_CLK_DIV2](#) 0x01

- QEP计数器频率2分频
- #define QEP_CLK_DIV4 0x02
 - QEP计数器频率4分频
- #define QEP_CLK_DIV8 0x03
 - QEP计数器频率8分频
- #define QEP_CLK_DIV16 0x04
 - QEP计数器频率16分频
- #define QEP_CLK_DIV32 0x05
 - QEP计数器频率32分频
- #define QEP_CLK_DIV64 0x06
 - QEP计数器频率64分频
- #define QEP_CLK_DIV128 0x07
 - QEP计数器频率128分频
- #define QEP_IRQEna_None 0
 - QEP中断使能宏定义
- #define QEP_IRQEna_UnderFlow BIT0
 - 使能下溢中断
- #define QEP_IRQEna_OverFlow BIT1
 - 使能上溢中断
- #define QEP_IF_UNDERFLOW BIT0
 - 编码器中断标志定义
- #define QEP_IF_OVERFLOW BIT1
 - Encoder上溢中断
- #define QEP_Mode_T1 ((uint32_t)0x00)
 - QEP模式选择宏定义
- #define QEP_Mode_T1_T2 ((uint32_t)0x01)
 - counting on T1&T2
- #define QEP_Mode_CCWSIGN_UpDown ((uint32_t)0x02)
 - CCW+SIGN 符号加脉冲信号计数模式, 若需要计数信号的上下沿, 则QEP_FE_CNT_EN置1
- #define QEP_Mode_CCWCW_UpDown ((uint32_t)0x03)
 - CCW+CW 双向计数模式, a和b分别为递增和递减脉冲 a信号变化则cnt递增, b信号变化则cnt递减
- #define QEP_ZC_NONE (0x0)
 - Z信号配置选择
- #define QEP_ZC_EDEG_P (0x5)
 - 上升沿清零
- #define QEP_ZC_EDEG_N (0x9)
 - 下降沿清零
- #define QEP_ZC_EDEG_ALL (0xd)
 - 上升沿&下降沿清零
- #define QEP_ZC_LEVEL_H (0xa)
 - 高电平清零
- #define QEP_ZC_LEVEL_L (0x6)
 - 低电平清零

函数

- void **QEP_Init** (**QEP_TypeDef** *QEPx, **QEP_InitTypeDef** *TIM_QEPInitStruct)
QEP初始化函数（初始化完成后会自动使能，并使能timer2）
- void **QEP_StructInit** (**QEP_InitTypeDef** *TIM_QEPInitStruct)
QEP结构体初始化函数
- void **QEP_Cmd** (**QEP_TypeDef** *QEPx, uint8_t state)
QEP使能/失能函数
- uint16_t **QEP_GetCount** (**QEP_TypeDef** *QEPx)
获取QEP计数器当前值
- void **QEP_ClearIRQFlag** (**QEP_TypeDef** *QEPx, uint32_t tempFlag)
清除QEP中断标志
- uint16_t **QEP_GetIRQFlag** (**QEP_TypeDef** *QEPx, uint32_t QEP_IF_x)
获取QEP中断标志状态
- uint16_t **QEP_GetFlag** (**QEP_TypeDef** *QEPx, uint32_t QEP_IF_x)
获取QEP状态标志
- void **QEP_SetMode** (**QEP_TypeDef** *QEPx, uint32_t QEP_MODE_x)
获取QEP状态标志

3.35.1 宏定义说明

3.35.1.1 QEP_CLK_DIV1

```
#define QEP_CLK_DIV1 0x00
```

QEP时钟分频宏定义

QEP计数器频率1分频

在文件 [lks32mc09x-qep.h](#) 第 23 行定义.

3.35.1.2 QEP_CLK_DIV128

```
#define QEP_CLK_DIV128 0x07
```

QEP计数器频率128分频

在文件 [lks32mc09x-qep.h](#) 第 30 行定义.

3.35.1.3 QEP_CLK_DIV16

```
#define QEP_CLK_DIV16 0x04
```

QEP计数器频率16分频

在文件 [lks32mc09x-qep.h](#) 第 27 行定义.

3.35.1.4 QEP_CLK_DIV2

```
#define QEP_CLK_DIV2 0x01
```

QEP计数器频率2分频

在文件 [lks32mc09x_qep.h](#) 第 24 行定义.

3.35.1.5 QEP_CLK_DIV32

```
#define QEP_CLK_DIV32 0x05
```

QEP计数器频率32分频

在文件 [lks32mc09x_qep.h](#) 第 28 行定义.

3.35.1.6 QEP_CLK_DIV4

```
#define QEP_CLK_DIV4 0x02
```

QEP计数器频率4分频

在文件 [lks32mc09x_qep.h](#) 第 25 行定义.

3.35.1.7 QEP_CLK_DIV64

```
#define QEP_CLK_DIV64 0x06
```

QEP计数器频率64分频

在文件 [lks32mc09x_qep.h](#) 第 29 行定义.

3.35.1.8 QEP_CLK_DIV8

```
#define QEP_CLK_DIV8 0x03
```

QEP计数器频率8分频

在文件 [lks32mc09x_qep.h](#) 第 26 行定义.

3.35.1.9 QEP_IF_OVERFLOW

```
#define QEP_IF_OVERFLOW BIT1
```

Encoder上溢中断

在文件 [lks32mc09x_qep.h](#) 第 43 行定义.

3.35.1.10 QEP_IF_UNDERFLOW

```
#define QEP_IF_UNDERFLOW BIT0
```

编码器中断标志定义

Encoder下溢中断

在文件 [lks32mc09x_qep.h](#) 第 42 行定义.

3.35.1.11 QEP_IRQEna_None

```
#define QEP_IRQEna_None 0
```

QEP中断使能宏定义

无使能

在文件 [lks32mc09x_qep.h](#) 第 35 行定义.

3.35.1.12 QEP_IRQEna_OverFlow

```
#define QEP_IRQEna_OverFlow BIT1
```

使能上溢中断

在文件 [lks32mc09x_qep.h](#) 第 37 行定义.

3.35.1.13 QEP_IRQEna_UnderFlow

```
#define QEP_IRQEna_UnderFlow BIT0
```

使能下溢中断

在文件 [lks32mc09x_qep.h](#) 第 36 行定义.

3.35.1.14 QEP_Mode_CCWCW_UpDown

```
#define QEP_Mode_CCWCW_UpDown ((uint32_t)0x03)
```

CCW+CW 双向计数模式，a和b分别为递增和递减脉冲 a信号变化则cnt递增，b信号变化则cnt递减

在文件 [lks32mc09x_qep.h](#) 第 51 行定义.

3.35.1.15 QEP_Mode_CCWSIGN_UpDown

```
#define QEP_Mode_CCWSIGN_UpDown ((uint32_t)0x02)
```

CCW+SIGN 符号加脉冲信号计数模式, 若需要计数信号的上下沿, 则QEP_FE_CNT.EN置1

在文件 [lks32mc09x_qep.h](#) 第 50 行定义.

3.35.1.16 QEP_Mode_T1

```
#define QEP_Mode_T1 ((uint32_t)0x00)
```

QEP模式选择宏定义

counting on T1

在文件 [lks32mc09x_qep.h](#) 第 48 行定义.

3.35.1.17 QEP_Mode_T1_T2

```
#define QEP_Mode_T1_T2 ((uint32_t)0x01)
```

counting on T1&T2

在文件 [lks32mc09x_qep.h](#) 第 49 行定义.

3.35.1.18 QEP_ZC_EDEG_ALL

```
#define QEP_ZC_EDEG_ALL (0xd)
```

上升沿&下降沿清零

在文件 [lks32mc09x_qep.h](#) 第 59 行定义.

3.35.1.19 QEP_ZC_EDEG_N

```
#define QEP_ZC_EDEG_N (0x9)
```

下降沿清零

在文件 [lks32mc09x_qep.h](#) 第 58 行定义.

3.35.1.20 QEP_ZC_EDEG_P

```
#define QEP_ZC_EDEG_P (0x5)
```

上升沿清零

在文件 [lks32mc09x_qep.h](#) 第 57 行定义.

3.35.1.21 QEP_ZC_LEVEL_H

```
#define QEP_ZC_LEVEL_H (0xa)
```

高电平清零

在文件 [lks32mc09x_qep.h](#) 第 60 行定义.

3.35.1.22 QEP_ZC_LEVEL_L

```
#define QEP_ZC_LEVEL_L (0x6)
```

低电平清零

在文件 [lks32mc09x-qep.h](#) 第 61 行定义.

3.35.1.23 QEP_ZC_NONE

```
#define QEP_ZC_NONE (0x0)
```

Z信号配置选择

不使用

在文件 [lks32mc09x-qep.h](#) 第 56 行定义.

3.35.2 函数说明

3.35.2.1 QEP_ClearIRQFlag()

```
void QEP_ClearIRQFlag (
    QEP_TypeDef * QEPx,
    uint32_t tempFlag )
```

清除QEP中断标志

参数

<i>QEPx</i>	QEP模块指针
<i>tempFlag</i>	要清除的中断标志

在文件 [lks32mc09x-qep.c](#) 第 82 行定义.

3.35.2.2 QEP_Cmd()

```
void QEP_Cmd (
    QEP_TypeDef * QEPx,
    uint8_t state )
```

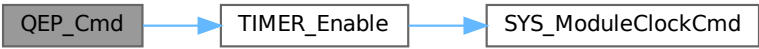
QEP使能/失能函数

参数

<i>QEPx</i>	QEP模块指针
<i>state</i>	使能/失能状态

在文件 [lks32mc09x-qep.c](#) 第 54 行定义.

函数调用图:



3.35.2.3 QEP_GetCount()

```
uint16_t QEP_GetCount (
    QEP_TypeDef * QEPx )
```

获取QEP计数器当前值

参数

QEPx	QEP模块指针

返回
计数器当前值

在文件 [lks32mc09x-qep.c](#) 第 72 行定义.

3.35.2.4 QEP_GetFlag()

```
uint16_t QEP_GetFlag (
    QEP_TypeDef * QEPx,
    uint32_t QEP_IFx )
```

获取QEP状态标志

参数

QEPx	QEP模块指针
QEP_↔IF_x	状态标志类型

返回
状态标志

在文件 [lks32mc09x-qep.c](#) 第 104 行定义.

3.35.2.5 QEP_GetIRQFlag()

```
uint16_t QEP_GetIRQFlag (
    QEP_TypeDef * QEPx,
    uint32_t QEP_IF_x )
```

获取QEP中断标志状态

参数

QEPx	QEP模块指针
QEP_IF_x	中断标志类型

返回
中断标志状态

在文件 lks32mc09x-qep.c 第 93 行定义.

3.35.2.6 QEP_Init()

```
void QEP_Init (
    QEP_TypeDef * QEPx,
    QEP_InitTypeDef * TIM_QEPInitStruct )
```

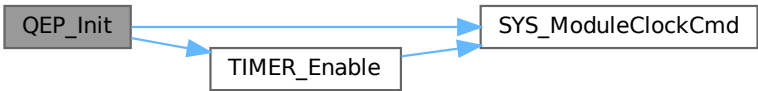
QEP初始化函数（初始化完成后会自动使能，并使能timer2）

参数

QEPx	QEP模块指针
TIM_QEPInitStruct	初始化结构体指针

在文件 lks32mc09x-qep.c 第 10 行定义.

函数调用图:



3.35.2.7 QEP_SetMode()

```
void QEP_SetMode (
```

```
QEP_TypeDef * QEPx,  
uint32_t QEP_MODE_x )
```

获取QEP状态标志

参数

QEPx	QEP模块指针
QEP_↔ MODE_x	编码器工作模式

在文件 [lks32mc09x_qep.c](#) 第 114 行定义.

3.35.2.8 QEP_StructInit()

```
void QEP_StructInit (  
    QEP_InitTypeDef * TIM_QEPInitStruct )
```

QEP结构体初始化函数

参数

TIM_QEPInitStruct	初始化结构体指针

在文件 [lks32mc09x_qep.c](#) 第 40 行定义.

3.36 lks32mc09x_qep.h

[浏览该文件的文档.](#)

```
00001 #ifndef __LKS32MC09X_QEP_H__  
00002 #define __LKS32MC09X_QEP_H__  
00003  
00004 #include "lks32mc09x_reg.h"  
00005  
00009 typedef struct  
00010 {  
00011     uint32_t Mode;  
00012     uint32_t FE_CNT_EN;  
00013     uint16_t TH;  
00014     uint32_t ClockDiv;  
00015     uint32_t Filter;  
00016     uint32_t IRQEna;  
00017     uint32_t ZC;  
00018 } QEP_InitTypeDef;  
00019  
00023 #define QEP_CLK_DIV1    0x00  
00024 #define QEP_CLK_DIV2    0x01  
00025 #define QEP_CLK_DIV4    0x02  
00026 #define QEP_CLK_DIV8    0x03  
00027 #define QEP_CLK_DIV16   0x04  
00028 #define QEP_CLK_DIV32   0x05  
00029 #define QEP_CLK_DIV64   0x06  
00030 #define QEP_CLK_DIV128  0x07  
00031  
00035 #define QEP_IRQEna_None    0  
00036 #define QEP_IRQEna_UnderFlow BIT0  
00037 #define QEP_IRQEna_OverFlow BIT1  
00038  
00042 #define QEP_IF_UNDERFLOW BIT0  
00043 #define QEP_IF_OVERFLOW  BIT1  
00044
```



```

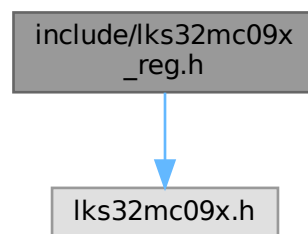
00048 #define QEP_Mode_T1          ((uint32_t)0x00)
00049 #define QEP_Mode_T1_T2        ((uint32_t)0x01)
00050 #define QEP_Mode_CCWSIGN_UpDown ((uint32_t)0x02)
00051 #define QEP_Mode_CCWCW_UpDown  ((uint32_t)0x03)
00052
00056 #define QEP_ZC_NONE           (0x0)
00057 #define QEP_ZC_EDEG_P         (0x5)
00058 #define QEP_ZC_EDEG_N         (0x9)
00059 #define QEP_ZC_EDEG_ALL       (0xd)
00060 #define QEP_ZC_LEVEL_H         (0xa)
00061 #define QEP_ZC_LEVEL_L         (0x6)
00062
00068 void QEP_Init(QEP_TypeDef *QEPx, QEP_InitTypeDef *TIM_QEPInitStruct);
00069
00074 void QEP_StructInit(QEP_InitTypeDef *TIM_QEPInitStruct);
00075
00081 void QEP_Cmd(QEP_TypeDef *QEPx, uint8_t state);
00082
00088 uint16_t QEP_GetCount(QEP_TypeDef *QEPx);
00089
00095 void QEP_ClearIRQFlag(QEP_TypeDef *QEPx, uint32_t tempFlag);
00096
00103 uint16_t QEP_GetIRQFlag(QEP_TypeDef *QEPx, uint32_t QEP_IF_x);
00104
00111 uint16_t QEP_GetFlag(QEP_TypeDef *QEPx, uint32_t QEP_IF_x);
00112
00118 void QEP_SetMode(QEP_TypeDef *QEPx, uint32_t QEP_MODE_x);
00119
00120 #endif

```

3.37 include/lks32mc09x_reg.h 文件参考

#include "lks32mc09x.h"

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 **ADC_TypeDef**
ADC0寄存器 (ADC0)
- struct **CMP_TypeDef**
比较器寄存器 (CMP)

- struct [CLU_TypeDef](#)
CL0寄存器
- struct [CRC_TypeDef](#)
CRC0寄存器 (CRC0)
- struct [DMA_TypeDef](#)
DMA寄存器
- struct [DMAMODULE_TypeDef](#)
DMA控制状态寄存器对象定义
- struct [DSP_TypeDef](#)
DSP0寄存器结构体 (DSP0)
- struct [EXTI_TypeDef](#)
EXTI寄存器 (EXTI)
- struct [FLASH_TypeDef](#)
FLASH寄存器 (FLASH)
- struct [GPIO_TypeDef](#)
GPIO寄存器
- struct [HALL_TypeDef](#)
HALL0寄存器 (HALL0)
- struct [I2C_TypeDef](#)
I2C0寄存器 (I2C0)
- struct [MCPWM_TypeDef](#)
MCPWM0寄存器 (MCPWM0)
- struct [QEP_TypeDef](#)
QEP0寄存器 (QEP0)
- struct [SIF_TypeDef](#)
SIF0寄存器 (SIF0)
- struct [SPI_TypeDef](#)
SPI0寄存器 (SPI0)
- struct [SYS_TypeDef](#)
SYS寄存器 (SYS)
- struct [TIMER_TypeDef](#)
TIMER0寄存器 (TIMER0)
- struct [UART_TypeDef](#)
UART寄存器
- struct [IWDG_TypeDef](#)
IWDG寄存器结构体句柄
- struct [AON_TypeDef](#)
AON寄存器结构体句柄

宏定义

- #define [ADC0](#) (([ADC_TypeDef](#) *)ADC0_BASE)
- #define [ADC1](#) (([ADC_TypeDef](#) *)ADC1_BASE)
- #define [CMP0](#) (([CMP_TypeDef](#) *)CMP_BASE)
- #define [CLU0](#) (([CLU_TypeDef](#) *)CL0_BASE)
- #define [CRC0](#) (([CRC_TypeDef](#) *)CRC0_BASE)
- #define [DMA_CHN0](#) (([DMA_TypeDef](#) *)DMA0_BASE)
- #define [DMA_CHN1](#) (([DMA_TypeDef](#) *) (DMA0_BASE + 0x20))
- #define [DMA_CHN2](#) (([DMA_TypeDef](#) *) (DMA0_BASE + 0x40))
- #define [DMA_CHN3](#) (([DMA_TypeDef](#) *) (DMA0_BASE + 0x60))
- #define [DMA_TOP](#) (([DMAMODULE_TypeDef](#) *) (DMA0_BASE + 0x80))

- #define DSP0 ((DSP_TypeDef*)(DSP0_BASE))
- #define EXTI0 ((EXTI_TypeDef*)(EXTI_BASE))
- #define FLASH0 ((FLASH_TypeDef*)(FLASH_CR_BASE))
- #define GPIO0 ((GPIO_TypeDef*)(GPIO0_BASE))
- #define GPIO1 ((GPIO_TypeDef*)(GPIO1_BASE))
- #define GPIO2 ((GPIO_TypeDef*)(GPIO2_BASE))
- #define HALL0 ((HALL_TypeDef*)(HALL0_BASE))
- #define I2C0 ((I2C_TypeDef*)(I2C0_BASE))
- #define MCPWM0 ((MCPWM_TypeDef*)(MCPWM0_BASE))
- #define QEP0 ((QEP_TypeDef*)(QEP0_BASE))
- #define SIF0 ((SIF_TypeDef*)(SIF0_BASE))
- #define SPI0 ((SPI_TypeDef*)(SPI0_BASE))
- #define SYS0 ((SYS_TypeDef*)(SYS_BASE))
- #define TIMER0 ((TIMER_TypeDef*)(TIMER0_BASE))
- #define TIMER1 ((TIMER_TypeDef*)(TIMER1_BASE))
- #define TIMER2 ((TIMER_TypeDef*)(TIMER2_BASE))
- #define UART0 ((UART_TypeDef*)(UART0_BASE))
- #define UART1 ((UART_TypeDef*)(UART1_BASE))
- #define UART2 ((UART_TypeDef*)(UART2_BASE))
- #define IWDG0 ((IWDG_TypeDef*)(IWDG_BASE))
- #define AON0 ((AON_TypeDef*)(AON_BASE))
- #define DSP DSP0
- #define DSP_REG DSP0
- #define ADC ADC0
- #define AON AON0
- #define CMP CMP0
- #define CLU CLU0
- #define CL0 CLU0
- #define DMA0 DMA_CHN0
- #define DMA1 DMA_CHN1
- #define DMA2 DMA_CHN2
- #define DMA3 DMA_CHN3
- #define DMAREG DMA_TOP
- #define EXTI EXTI0
- #define FLASH FLASH0
- #define HALL HALL0
- #define I2C I2C0
- #define MCPWM MCPWM0
- #define QEP QEP0
- #define SIF SIF0
- #define SPI SPI0
- #define SYS SYS0
- #define UTIMER0 TIMER0
- #define UTIMER1 TIMER1
- #define UTIMER2 TIMER2
- #define UART UART0
- #define TIMER16_0 UTIMER0
- #define TIMER16_1 UTIMER1
- #define TIMER32_0 UTIMER2
- #define UTIMER16_0 UTIMER0
- #define UTIMER16_1 UTIMER1
- #define UTIMER32_0 UTIMER2
- #define IWDG IWDG0
- #define REG_WRITE(reg, mask) reg = (mask)
- #define REG_READ(reg) (reg)

- #define `REG.WRITE_BIT`(reg, mask, val) reg = ((reg) & ~(mask)) | (val)
- #define `REG.READ_BIT`(reg, mask) ((reg) & (mask))
- #define `REG.READ_BIT_POS_MASK`(reg, mask) (((reg) & (mask##_MASK)) >> mask##_POS)
- #define `REG.WRITE_BIT_POS_MASK`(reg, name, val) reg = ((reg) & ~(name##_MASK)) | name(val)
- #define `REG.SET`(reg, mask) reg |= (mask)
- #define `REG.RESET`(reg, mask) reg &= ~(mask)
- #define `REG.IF_WRITE`(reg, mask, condition) reg = (condition) ? ((reg) | (mask)) : ((reg) & ~(mask))

3.37.1 宏定义说明

3.37.1.1 ADC

```
#define ADC ADC0
```

在文件 `lks32mc09x_reg.h` 第 425 行定义。

3.37.1.2 ADC0

```
#define ADC0 ((ADC_TypeDef *)ADC0_BASE)
```

在文件 `lks32mc09x_reg.h` 第 390 行定义。

3.37.1.3 ADC1

```
#define ADC1 ((ADC_TypeDef *)ADC1_BASE)
```

在文件 `lks32mc09x_reg.h` 第 391 行定义。

3.37.1.4 AON

```
#define AON AON0
```

在文件 `lks32mc09x_reg.h` 第 426 行定义。

3.37.1.5 AON0

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

在文件 `lks32mc09x_reg.h` 第 420 行定义。

3.37.1.6 CL0

```
#define CL0 CLU0
```

在文件 `lks32mc09x_reg.h` 第 429 行定义。

3.37.1.7 CLU

```
#define CLU CLU0
```

在文件 [lks32mc09x_reg.h](#) 第 428 行定义.

3.37.1.8 CLU0

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

在文件 [lks32mc09x_reg.h](#) 第 393 行定义.

3.37.1.9 CMP

```
#define CMP CMP0
```

在文件 [lks32mc09x_reg.h](#) 第 427 行定义.

3.37.1.10 CMP0

```
#define CMP0 ((CMP_TypeDef *)CMP_BASE)
```

在文件 [lks32mc09x_reg.h](#) 第 392 行定义.

3.37.1.11 CRC0

```
#define CRC0 ((CRC_TypeDef *)CRC0_BASE)
```

在文件 [lks32mc09x_reg.h](#) 第 394 行定义.

3.37.1.12 DMA0

```
#define DMA0 DMA_CHN0
```

在文件 [lks32mc09x_reg.h](#) 第 430 行定义.

3.37.1.13 DMA1

```
#define DMA1 DMA_CHN1
```

在文件 [lks32mc09x_reg.h](#) 第 431 行定义.

3.37.1.14 DMA2

```
#define DMA2 DMA_CHN2
```

在文件 [lks32mc09x_reg.h](#) 第 432 行定义.

3.37.1.15 DMA3

```
#define DMA3 DMA_CHN3
```

在文件 [lks32mc09x_reg.h](#) 第 433 行定义.

3.37.1.16 DMA_CHN0

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

在文件 [lks32mc09x_reg.h](#) 第 395 行定义.

3.37.1.17 DMA_CHN1

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

在文件 [lks32mc09x_reg.h](#) 第 396 行定义.

3.37.1.18 DMA_CHN2

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

在文件 [lks32mc09x_reg.h](#) 第 397 行定义.

3.37.1.19 DMA_CHN3

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

在文件 [lks32mc09x_reg.h](#) 第 398 行定义.

3.37.1.20 DMA_TOP

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

在文件 [lks32mc09x_reg.h](#) 第 399 行定义.

3.37.1.21 DMAREG

```
#define DMAREG DMA_TOP
```

在文件 [lks32mc09x_reg.h](#) 第 434 行定义.

3.37.1.22 DSP

```
#define DSP DSP0
```

在文件 [lks32mc09x_reg.h](#) 第 423 行定义.

3.37.1.23 DSP0

```
#define DSP0 ((DSP_TypeDef *) (DSP0_BASE))
```

在文件 [lks32mc09x_reg.h](#) 第 400 行定义.

3.37.1.24 DSP_REG

```
#define DSP_REG DSP0
```

在文件 [lks32mc09x_reg.h](#) 第 424 行定义.

3.37.1.25 EXTI

```
#define EXTI EXTI0
```

在文件 [lks32mc09x_reg.h](#) 第 435 行定义.

3.37.1.26 EXTI0

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

在文件 [lks32mc09x_reg.h](#) 第 401 行定义.

3.37.1.27 FLASH

```
#define FLASH FLASH0
```

在文件 [lks32mc09x_reg.h](#) 第 436 行定义.

3.37.1.28 FLASH0

```
#define FLASH0 ((FLASH_TypeDef *) FLASH_CR_BASE)
```

在文件 [lks32mc09x_reg.h](#) 第 402 行定义.

3.37.1.29 GPIO0

```
#define GPIO0 ((GPIO_TypeDef *) GPIO0_BASE)
```

在文件 [lks32mc09x_reg.h](#) 第 403 行定义.

3.37.1.30 GPIO1

```
#define GPIO1 ((GPIO_TypeDef *) GPIO1_BASE)
```

在文件 [lks32mc09x_reg.h](#) 第 404 行定义.

3.37.1.31 GPIO2

```
#define GPIO2 ((GPIO_TypeDef *)GPIO2_BASE)
```

在文件 [lks32mc09x_reg.h](#) 第 405 行定义.

3.37.1.32 HALL

```
#define HALL HALL0
```

在文件 [lks32mc09x_reg.h](#) 第 437 行定义.

3.37.1.33 HALL0

```
#define HALL0 ((HALL_TypeDef *)HALL0_BASE)
```

在文件 [lks32mc09x_reg.h](#) 第 406 行定义.

3.37.1.34 I2C

```
#define I2C I2C0
```

在文件 [lks32mc09x_reg.h](#) 第 438 行定义.

3.37.1.35 I2C0

```
#define I2C0 ((I2C_TypeDef *)I2C0_BASE)
```

在文件 [lks32mc09x_reg.h](#) 第 407 行定义.

3.37.1.36 IWDG

```
#define IWDG IWDG0
```

在文件 [lks32mc09x_reg.h](#) 第 454 行定义.

3.37.1.37 IWDG0

```
#define IWDG0 ((IWDG_TypeDef *)IWDG_BASE)
```

在文件 [lks32mc09x_reg.h](#) 第 419 行定义.

3.37.1.38 MCPWM

```
#define MCPWM MCPWM0
```

在文件 [lks32mc09x_reg.h](#) 第 439 行定义.

3.37.1.39 MCPWM0

```
#define MCPWM0 ((MCPWM_TypeDef *)MCPWM0_BASE)
```

在文件 [lks32mc09x_reg.h](#) 第 408 行定义.

3.37.1.40 QEP

```
#define QEP QEP0
```

在文件 [lks32mc09x_reg.h](#) 第 440 行定义.

3.37.1.41 QEP0

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

在文件 [lks32mc09x_reg.h](#) 第 409 行定义.

3.37.1.42 REG_IF_WRITE

```
#define REG_IF_WRITE(  
    reg,  
    mask,  
    condition ) reg = (condition) ? ((reg) | (mask)) : ((reg) & ~(mask))
```

在文件 [lks32mc09x_reg.h](#) 第 464 行定义.

3.37.1.43 REG_READ

```
#define REG_READ(  
    reg ) (reg)
```

在文件 [lks32mc09x_reg.h](#) 第 457 行定义.

3.37.1.44 REG_READ_BIT

```
#define REG_READ_BIT(  
    reg,  
    mask ) ((reg) & (mask))
```

在文件 [lks32mc09x_reg.h](#) 第 459 行定义.

3.37.1.45 REG_READ_BIT_POS_MASK

```
#define REG_READ_BIT_POS_MASK(  
    reg,  
    mask ) (((reg) & (mask##_MASK)) >> mask##_POS)
```

在文件 [lks32mc09x_reg.h](#) 第 460 行定义.

3.37.1.46 REG_RESET

```
#define REG_RESET(  
    reg,  
    mask ) reg &= ~(mask)
```

在文件 [lks32mc09x_reg.h](#) 第 463 行定义.

3.37.1.47 REG_SET

```
#define REG_SET(  
    reg,  
    mask ) reg |= (mask)
```

在文件 [lks32mc09x_reg.h](#) 第 462 行定义.

3.37.1.48 REG_WRITE

```
#define REG_WRITE(  
    reg,  
    mask ) reg = (mask)
```

在文件 [lks32mc09x_reg.h](#) 第 456 行定义.

3.37.1.49 REG_WRITE_BIT

```
#define REG_WRITE_BIT(  
    reg,  
    mask,  
    val ) reg = ((reg) & ~(mask)) | (val)
```

在文件 [lks32mc09x_reg.h](#) 第 458 行定义.

3.37.1.50 REG_WRITE_BIT_POS_MASK

```
#define REG_WRITE_BIT_POS_MASK(  
    reg,  
    name,  
    val ) reg = ((reg) & ~(name##_MASK)) | name(val)
```

在文件 [lks32mc09x_reg.h](#) 第 461 行定义.

3.37.1.51 SIF

```
#define SIF SIF0
```

在文件 [lks32mc09x_reg.h](#) 第 441 行定义.

3.37.1.52 SIF0

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

在文件 [lks32mc09x_reg.h](#) 第 410 行定义.

3.37.1.53 SPI

```
#define SPI SPI0
```

在文件 [lks32mc09x_reg.h](#) 第 442 行定义.

3.37.1.54 SPI0

```
#define SPI0 ((SPI_TypeDef *) SPI0_BASE)
```

在文件 [lks32mc09x_reg.h](#) 第 411 行定义.

3.37.1.55 SYS

```
#define SYS SYS0
```

在文件 [lks32mc09x_reg.h](#) 第 443 行定义.

3.37.1.56 SYS0

```
#define SYS0 ((SYS_TypeDef *) (SYS_BASE))
```

在文件 [lks32mc09x_reg.h](#) 第 412 行定义.

3.37.1.57 TIMER0

```
#define TIMER0 ((TIMER_TypeDef *) TIMER0_BASE)
```

在文件 [lks32mc09x_reg.h](#) 第 413 行定义.

3.37.1.58 TIMER1

```
#define TIMER1 ((TIMER_TypeDef *) TIMER1_BASE)
```

在文件 [lks32mc09x_reg.h](#) 第 414 行定义.

3.37.1.59 TIMER16_0

```
#define TIMER16_0 UTIMER0
```

在文件 [lks32mc09x_reg.h](#) 第 448 行定义.

3.37.1.60 TIMER16_1

```
#define TIMER16_1 UTIMER1
```

在文件 [lks32mc09x_reg.h](#) 第 449 行定义.

3.37.1.61 TIMER2

```
#define TIMER2 ((TIMER_TypeDef *)TIMER2_BASE)
```

在文件 [lks32mc09x_reg.h](#) 第 415 行定义.

3.37.1.62 TIMER32_0

```
#define TIMER32_0 UTIMER2
```

在文件 [lks32mc09x_reg.h](#) 第 450 行定义.

3.37.1.63 UART

```
#define UART UART0
```

在文件 [lks32mc09x_reg.h](#) 第 447 行定义.

3.37.1.64 UART0

```
#define UART0 ((UART_TypeDef *)UART0_BASE)
```

在文件 [lks32mc09x_reg.h](#) 第 416 行定义.

3.37.1.65 UART1

```
#define UART1 ((UART_TypeDef *)UART1_BASE)
```

在文件 [lks32mc09x_reg.h](#) 第 417 行定义.

3.37.1.66 UART2

```
#define UART2 ((UART_TypeDef *)UART2_BASE)
```

在文件 [lks32mc09x_reg.h](#) 第 418 行定义.

3.37.1.67 UTIMER0

```
#define UTIMER0 TIMER0
```

在文件 [lks32mc09x_reg.h](#) 第 444 行定义.

3.37.1.68 UTIMER1

```
#define UTIMER1 TIMER1
```

在文件 [lks32mc09x_reg.h](#) 第 445 行定义.

3.37.1.69 UTIMER16_0

```
#define UTIMER16_0 UTIMER0
```

在文件 [lks32mc09x_reg.h](#) 第 451 行定义.

3.37.1.70 UTIMER16_1

```
#define UTIMER16_1 UTIMER1
```

在文件 [lks32mc09x_reg.h](#) 第 452 行定义.

3.37.1.71 UTIMER2

```
#define UTIMER2 TIMER2
```

在文件 [lks32mc09x_reg.h](#) 第 446 行定义.

3.37.1.72 UTIMER32_0

```
#define UTIMER32_0 UTIMER2
```

在文件 [lks32mc09x_reg.h](#) 第 453 行定义.

3.38 lks32mc09x_reg.h

[浏览该文件的文档.](#)

```
00001 #ifndef __LKS32MC09X_REG_H__
00002 #define __LKS32MC09X_REG_H__
00003
00004 #include "lks32mc09x.h"
00008 typedef struct
00009 {
00010     __IOM uint32_t DAT0;
00011     __IOM uint32_t DAT1;
00012     __IOM uint32_t DAT2;
00013     __IOM uint32_t DAT3;
00014     __IOM uint32_t DAT4;
00015     __IOM uint32_t DAT5;
00016     __IOM uint32_t DAT6;
00017     __IOM uint32_t DAT7;
00018     __IOM uint32_t DAT8;
00019     __IOM uint32_t DAT9;
00020     __IOM uint32_t DAT10;
00021     __IOM uint32_t DAT11;
00022     __IOM uint32_t RESERVED[8]; //
00023     __IOM uint32_t CHN0;
00024     __IOM uint32_t CHN1;
00025     __IOM uint32_t CHN2;
00026     __IOM uint32_t RESERVED1; //
```

```

00027     __IOM uint32_t  CHNT;
00028     __IOM uint32_t  GAIN;
00029     __IM uint32_t  RESERVED2[3];  //
00030     __IOM uint32_t  CFG;
00031     __IOM uint32_t  TRIG;
00032     __IOM uint32_t  SWT;
00033     __IOM uint32_t  DC0;
00034     __IOM uint32_t  AMC0;
00035     __IOM uint32_t  DC1;
00036     __IOM uint32_t  AMC1;
00037     __IOM uint32_t  IE;
00038     __IOM uint32_t  IF;
00039     __IM uint32_t  RESERVED3[11];  //
00040     __IOM uint32_t  LTH;
00041     __IOM uint32_t  HTH;
00042     __IOM uint32_t  GEN;
00043 } ADC_TypeDef;
00047 typedef struct
00048 {
00049     __IOM uint32_t  IE;
00050     __IOM uint32_t  IF;
00051     __IOM uint32_t  TCLK;
00052     __IOM uint32_t  CFG;
00053     __IOM uint32_t  BLCWIN;
00054     __IOM uint32_t  DATA;
00055 } CMP_TypeDef;
00059 typedef struct
00060 {
00061     __IOM uint32_t  EN;
00062     __IOM uint32_t  IE;
00063     __IOM uint32_t  IF;
00064     __IOM uint32_t  OUT;
00065     __IOM uint32_t  MX;
00066     __IOM uint32_t  FN;
00067     __IOM uint32_t  CF;
00068 } CLU_TypeDef;
00072 typedef struct
00073 {
00074     __IOM uint32_t  DR;
00075     __IOM uint32_t  CR;
00076     __IOM uint32_t  INIT;
00077     __IOM uint32_t  POL;
00078 } CRC_TypeDef;
00082 typedef struct
00083 {
00084     __IOM uint32_t  CCR;
00085     __IOM uint32_t  REN;
00086     __IOM uint32_t  CTMS;
00087     __IOM uint32_t  SADR;
00088     __IOM uint32_t  DADR;
00089 } DMA_TypeDef;
00093 typedef struct
00094 {
00095     __IOM uint32_t  CTRL;
00096     __IOM uint32_t  IE;
00097     __IOM uint32_t  IF;
00098 } DMAMODULE_TypeDef;
00102 typedef struct
00103 {
00104     __IM uint32_t  RESERVED[1032];
00105     __IOM uint32_t  DID;
00106     __IOM uint32_t  DIS;
00107     __IM uint32_t  QUO;
00108     __IM uint32_t  REM;
00109     __IOM uint32_t  DID1;
00110     __IOM uint32_t  DIS1;
00111     __IM uint32_t  QUO1;
00112     __IM uint32_t  RESERVED1;
00113     __IOM uint32_t  RAD;
00114     __IM uint16_t  SQRT;
00115     __IM uint16_t  RESERVED2;
00116 } DSP_Typedef;
00120 typedef struct
00121 {
00122     __IOM uint32_t  CR0;
00123     __IOM uint32_t  CR1;
00124     __IOM uint32_t  IE;
00125     __IOM uint32_t  IF;
00126     __IOM uint32_t  CLKO;
00127     __IOM uint32_t  SWAP;
00128 } EXTI_TypeDef;
00132 typedef struct
00133 {
00134     __IOM uint32_t  CFG;
00135     __IOM uint32_t  ADDR;
00136     __IOM uint32_t  WDATA;
00137     __IM uint32_t  RDATA;

```

```

00138     __OM uint32_t ERASE;
00139     __IM uint32_t PROTECT;
00140     __IM uint32_t READY;
00141     __IM uint32_t SIZE;
00142     __IOM uint32_t NCFG;
00143     __IOM uint32_t NADDR;
00144     __OM uint32_t NWDATA;
00145     __IM uint32_t NRDATA;
00146     __OM uint32_t NERASE;
00147     __OM uint32_t NKEY;
00148     __IOM uint32_t EPSM;
00149     __IM uint32_t RESERVED; //
00150     __IOM uint32_t EPSN;
00151     __IOM uint32_t LIB_ADDR;
00152 } FLASH_TypeDef;
00156 typedef struct
00157 {
00158     __IOM uint32_t PIE;
00159     __IOM uint32_t POE;
00160     __IOM uint32_t PDI;
00161     __IOM uint32_t PDO;
00162     __IOM uint32_t PUE;
00163     __IM uint32_t RESERVED; //
00164     __IOM uint32_t PODE;
00165     __IOM uint32_t PFLT;
00166     __IOM uint32_t F3210;
00167     __IOM uint32_t F7654;
00168     __IOM uint32_t FBA98;
00169     __IOM uint32_t FFEDC;
00170     __OM uint32_t BSRR;
00171     __OM uint32_t BRR;
00172 } GPIO_TypeDef;
00176 typedef struct
00177 {
00178     __IOM uint32_t CFG;
00179     __IOM uint32_t INFO;
00180     __IM uint32_t WIDTH;
00181     __IOM uint32_t TH;
00182     __IOM uint32_t CNT;
00183 } HALL_TypeDef;
00187 typedef struct
00188 {
00189     __IOM uint32_t ADDR;
00190     __IOM uint32_t CFG;
00191     __IOM uint32_t SCR;
00192     __IOM uint32_t DATA;
00193     __IOM uint32_t MSCR;
00194     __IOM uint32_t BCR;
00195     __IOM uint32_t BSIZ;
00196 } I2C_TypeDef;
00200 typedef struct
00201 {
00202     __IOM uint32_t TH00;
00203     __IOM uint32_t TH01;
00204     __IOM uint32_t TH10;
00205     __IOM uint32_t TH11;
00206     __IOM uint32_t TH20;
00207     __IOM uint32_t TH21;
00208     __IOM uint32_t TH30;
00209     __IOM uint32_t TH31;
00210     __IM uint32_t RESERVED[4]; //
00211     __IOM uint32_t TMR0;
00212     __IOM uint32_t TMR1;
00213     __IOM uint32_t TMR2;
00214     __IOM uint32_t TMR3;
00215     __IOM uint32_t TH0;
00216     __IM uint32_t RESERVED1; //
00217     __IOM uint32_t CNT0;
00218     __IM uint32_t RESERVED2; //
00219     __IOM uint32_t UPDATE;
00220     __IOM uint32_t FCNT;
00221     __IOM uint32_t EVT0;
00222     __IM uint32_t RESERVED3; //
00223     __IOM uint32_t DTH00;
00224     __IOM uint32_t DTH01;
00225     __IM uint32_t RESERVED4[2]; //
00226     __IOM uint32_t FLT;
00227     __IOM uint32_t SDCFG;
00228     __IOM uint32_t AUVEN;
00229     __IOM uint32_t TCLK;
00230     __IOM uint32_t IE0;
00231     __IOM uint32_t IF0;
00232     __IM uint32_t RESERVED5[2]; //
00233     __IOM uint32_t EIE;
00234     __IOM uint32_t EIF;
00235     __IOM uint32_t RE;
00236     __IOM uint32_t PP;

```

```

00237     __IOM uint32_t IO01;
00238     __IOM uint32_t IO23;
00239     __IM uint32_t RESERVED6[2]; //
00240     __IOM uint32_t CH_FAIL;
00241     __IM uint32_t RESERVED7; //
00242     __IOM uint32_t CH_DEF;
00243     __IOM uint32_t CH_MSK;
00244     __IOM uint32_t PRT;
00245     __IM uint32_t RESERVED8[3]; //
00246     __IOM uint32_t SW_FAIL;
00247 } MCPWM_TypeDef;
00251 typedef struct
00252 {
00253     __IOM uint32_t CFG;
00254     __IOM uint32_t TH;
00255     __IOM uint32_t CNT;
00256     __IOM uint32_t IE;
00257     __IOM uint32_t IF;
00258 } QEP_TypeDef;
00262 typedef struct
00263 {
00264     __IOM uint32_t CFG;
00265     __IOM uint32_t TOSC;
00266     __IOM uint32_t TSTH1;
00267     __IOM uint32_t TDTH1;
00268     __IOM uint32_t IRQ;
00269     __IOM uint32_t WDATA;
00270 } SIF_TypeDef;
00274 typedef struct
00275 {
00276     __IOM uint32_t CFG;
00277     __IOM uint32_t IE;
00278     __IOM uint32_t DIV;
00279     __IOM uint32_t TX_DATA;
00280     __IOM uint32_t RX_DATA;
00281     __IOM uint32_t SIZE;
00282 } SPI_TypeDef;
00286 typedef struct
00287 {
00288     __IM uint32_t AFE_ADC;
00289     __IM uint32_t AFE_INFO;
00290     __IM uint32_t AFE_DBG;
00291     __IOM uint32_t OPA_SEL;
00292     __IOM uint32_t AFE_REG0;
00293     __IOM uint32_t AFE_REG1;
00294     __IOM uint32_t AFE_REG2;
00295     __IOM uint32_t AFE_REG3;
00296     __IOM uint32_t AFE_REG4;
00297     __IOM uint32_t AFE_REG5;
00298     __IM uint32_t RESERVED; //
00299     __IOM uint32_t AFE_REG7;
00300     __IM uint32_t RESERVED1[11]; //
00301     __IOM uint32_t AFE_DAC_CTRL;
00302     __IOM uint32_t AFE_DAC0;
00303     __IOM uint32_t AFE_DAC1;
00304     __IOM uint32_t AFE_DAC0_AMC;
00305     __IOM uint32_t AFE_DAC0_DC;
00306     __IOM uint32_t AFE_DAC1_AMC;
00307     __IOM uint32_t AFE_DAC1_DC;
00308     __IM uint32_t RESERVED2[2]; //
00309     __IOM uint32_t CLK_CFG;
00310     __IOM uint32_t IO_CFG;
00311     __IOM uint32_t DBG_CFG;
00312     __IM uint32_t RESERVED3; //
00313     __IOM uint32_t CLK_DIV0;
00314     __IOM uint32_t CLK_DIV1; //
00315     __IOM uint32_t CLK_DIV2;
00316     __IOM uint32_t CLK_FEN;
00317     __IM uint32_t RESERVED41; //
00318     __IOM uint32_t SFT_RST;
00319     __IOM uint32_t PROTECT;
00320     __IM uint32_t RESERVED5; //
00321     __IOM uint32_t NMI_TH;
00322     __IOM uint32_t IF;
00323     __IOM uint32_t RAM_LOCK;
00324     __IM uint32_t RESERVED6; //
00325     __IM uint32_t RESERVED7[4]; //
00326     __IOM uint32_t FLSE;
00327     __IOM uint32_t FLSP;
00328 } SYS_TypeDef;
00332 typedef struct
00333 {
00334     __IOM uint32_t CFG0;
00335     __IOM uint32_t CFG1;
00336     __IOM uint32_t CFG2;
00337     __IM uint32_t RESERVED; //
00338     __IOM uint32_t TH;

```



```

00339     __IOM uint32_t CNT;
00340     __IOM uint32_t CHN0;
00341     __IOM uint32_t CHN1;
00342     __IOM uint32_t EVT;
00343     __IOM uint32_t FLT;
00344     __IOM uint32_t IE;
00345     __IOM uint32_t IF;
00346     __IOM uint32_t IO;
00347 } TIMER_TypeDef;
00351 typedef struct
00352 {
00353     __IOM uint32_t CTRL;
00354     __IOM uint32_t DIVH;
00355     __IOM uint32_t DIVL;
00356     __IOM uint32_t BUFF;
00357     __IOM uint32_t ADR;
00358     __IOM uint32_t STT;
00359     __IOM uint32_t RE;
00360     __IOM uint32_t IE;
00361     __IOM uint32_t IF;
00362     __IOM uint32_t IOC;
00363 } UART_TypeDef;
00367 typedef struct
00368 {
00369     __IO uint32_t PSW;
00370     __IO uint32_t CFG;
00371     __IO uint32_t CLR;
00372     __IO uint32_t WTH;
00373     __IO uint32_t RTH;
00374     __IO uint32_t CNT;
00375 } IWDG_TypeDef;
00376
00380 typedef struct
00381 {
00382     __IOM uint32_t PWR_CFG;
00383     __IOM uint32_t EVT_RCD;
00384     __IOM uint32_t IO_WAKE_POL;
00385     __IOM uint32_t IO_WAKE_EN;
00386     __IOM uint32_t BKP_REG0;
00387     __IOM uint32_t BKP_REG1;
00388 } AON_TypeDef;
00389
00390 #define ADC0      ((ADC_TypeDef *)ADC0_BASE)
00391 #define ADC1      ((ADC_TypeDef *)ADC1_BASE)
00392 #define CMP0      ((CMP_TypeDef *)CMP_BASE)
00393 #define CLU0      ((CLU_TypeDef *)CLU_BASE)
00394 #define CRC0      ((CRC_TypeDef *)CRC0_BASE)
00395 #define DMA_CHN0  ((DMA_TypeDef *)DMA0_BASE)
00396 #define DMA_CHN1  ((DMA_TypeDef *) (DMA0_BASE + 0x20))
00397 #define DMA_CHN2  ((DMA_TypeDef *) (DMA0_BASE + 0x40))
00398 #define DMA_CHN3  ((DMA_TypeDef *) (DMA0_BASE + 0x60))
00399 #define DMA_TOP   ((DMAMODULE_TypeDef *) (DMA0_BASE + 0x80))
00400 #define DSP0      ((DSP_TypeDef *) (DSP0_BASE))
00401 #define EXTI0     ((EXTI_TypeDef *)EXTI_BASE)
00402 #define FLASH0    ((FLASH_TypeDef *)FLS_CR_BASE)
00403 #define GPIO0     ((GPIO_TypeDef *)GPIO0_BASE)
00404 #define GPIO1     ((GPIO_TypeDef *)GPIO1_BASE)
00405 #define GPIO2     ((GPIO_TypeDef *)GPIO2_BASE)
00406 #define HALL0     ((HALL_TypeDef *)HALL0_BASE)
00407 #define I2C0      ((I2C_TypeDef *)I2C0_BASE)
00408 #define MCPWM0    ((MCPWM_TypeDef *)MCPWM0_BASE)
00409 #define QEP0      ((QEP_TypeDef *)QEP0_BASE)
00410 #define SIF0      ((SIF_TypeDef *) (SIF0_BASE))
00411 #define SPI0      ((SPI_TypeDef *)SPI0_BASE)
00412 #define SYS0      ((SYS_TypeDef *) (SYS_BASE))
00413 #define TIMER0    ((TIMER_TypeDef *)TIMER0_BASE)
00414 #define TIMER1    ((TIMER_TypeDef *)TIMER1_BASE)
00415 #define TIMER2    ((TIMER_TypeDef *)TIMER2_BASE)
00416 #define UART0     ((UART_TypeDef *)UART0_BASE)
00417 #define UART1     ((UART_TypeDef *)UART1_BASE)
00418 #define UART2     ((UART_TypeDef *)UART2_BASE)
00419 #define IWDG0     ((IWDG_TypeDef *)IWDG_BASE)
00420 #define AON0      ((AON_TypeDef *)AON_BASE)
00421
00422 // 兼容一些之前的写法
00423 #define DSP      DSP0
00424 #define DSP_REG  DSP0
00425 #define ADC      ADC0
00426 #define AON      AON0
00427 #define CMP      CMP0
00428 #define CLU      CLU0
00429 #define CL0      CLU0
00430 #define DMA0     DMA_CHN0
00431 #define DMA1     DMA_CHN1
00432 #define DMA2     DMA_CHN2
00433 #define DMA3     DMA_CHN3
00434 #define DMAREG   DMA_TOP

```

```

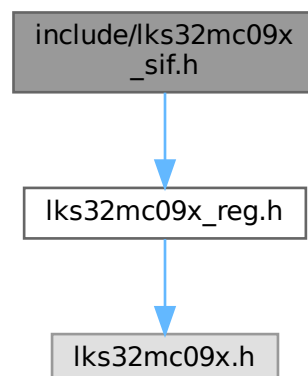
00435 #define EXTI                                EXTI0
00436 #define FLASH                                FLASH0
00437 #define HALL                                  HALL0
00438 #define I2C                                  I2C0
00439 #define MCPWM                                MCPWM0
00440 #define QEP                                  QEP0
00441 #define SIF                                  SIF0
00442 #define SPI                                  SPI0
00443 #define SYS                                  SYS0
00444 #define UTIMER0                              TIMER0
00445 #define UTIMER1                              TIMER1
00446 #define UTIMER2                              TIMER2
00447 #define UART                                  UART0
00448 #define TIMER16_0                            UTIMER0
00449 #define TIMER16_1                            UTIMER1
00450 #define TIMER32_0                            UTIMER2
00451 #define UTIMER16_0                            UTIMER0
00452 #define UTIMER16_1                            UTIMER1
00453 #define UTIMER32_0                            UTIMER2
00454 #define IWDG                                  IWDG0
00455
00456 #define REG.WRITE(reg, mask)                  reg = (mask) // 写寄存器
00457 #define REG.READ(reg)                        (reg)
00458 #define REG.WRITE.BIT(reg, mask, val)        reg = ((reg) & ~(mask)) | (val)
00459 #define REG.READ.BIT(reg, mask)              ((reg) & (mask))
00460 #define REG.READ.BIT.POS.MASK(reg, mask)      (((reg) & (mask##_MASK)) >> mask##_POS)
00461 #define REG.WRITE.BIT.POS.MASK(reg, name, val) reg = ((reg) & ~(name##_MASK)) | name(val)
00462 #define REG.SET(reg, mask)                   reg |= (mask)
00463 #define REG.RESET(reg, mask)                 reg &= ~(mask)
00464 #define REG_IF.WRITE(reg, mask, condition)   reg = (condition) ? ((reg) | (mask)) : ((reg) & ~(mask))
00465
00466 #endif

```

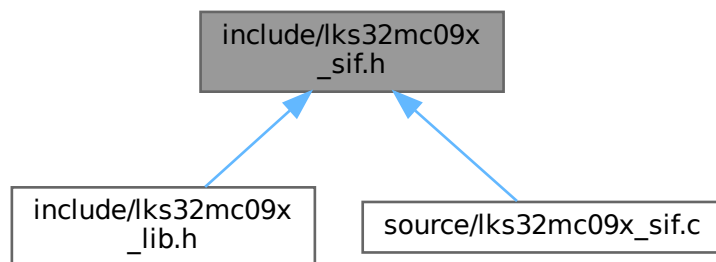
3.39 include/lks32mc09x_sif.h 文件参考

```
#include "lks32mc09x_reg.h"
```

lks32mc09x_sif.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 **SIF_InitTypeDef**
SIF初始化结构体定义

宏定义

- #define **SIF_IE.Enable** BIT0
SIF中断使能定义
- #define **SIF_DMA.Enable** BIT1
SIF DMA请求使能开关。默认值为0
- #define **SIF_IF.CLEAR** BIT4
SIF中断标志定义

函数

- void **SIF_StructInit** (**SIF_InitTypeDef** *SIFInitStruct)
初始化SIF结构体
- void **SIF_Init** (**SIF_TypeDef** *SIFx, **SIF_InitTypeDef** *SIFInitStruct)
初始化SIF模块
- void **SIF_Timebase.set** (**SIF_TypeDef** *SIFx, uint32_t tosc)
设置SIF时基
- void **SIF_TH.set** (**SIF_TypeDef** *SIFx, uint32_t tsh, uint32_t tdh)
设置SIF同步和结束信号时长
- void **SIF_Senddata** (**SIF_TypeDef** *SIFx, uint8_t Data)
发送数据
- void **SIF_DMA.IsEnable** (**SIF_TypeDef** *SIFx, uint8_t state)
设置SIF DMA使能状态
- void **SIF_Irq.IsEnable** (**SIF_TypeDef** *SIFx, uint8_t state)
设置SIF中断使能状态
- void **SIF_Clear_Irq** (**SIF_TypeDef** *SIFx)
清除SIF中断标志

3.39.1 宏定义说明

3.39.1.1 SIF_DMA_Enable

```
#define SIF_DMA_Enable BIT1
```

SIF DMA请求使能开关。默认值为0

在文件 [lks32mc09x_sif.h](#) 第 31 行定义.

3.39.1.2 SIF_IE_Enable

```
#define SIF_IE_Enable BIT0
```

SIF中断使能定义

SIF中断使能开关。默认值为0

在文件 [lks32mc09x_sif.h](#) 第 30 行定义.

3.39.1.3 SIF_IF_CLEAR

```
#define SIF_IF_CLEAR BIT4
```

SIF中断标志定义

SIF传输，完成事件。高电平有效，写1清除。

在文件 [lks32mc09x_sif.h](#) 第 36 行定义.

3.39.2 函数说明

3.39.2.1 SIF_Clear_Irq()

```
void SIF_Clear_Irq (
    SIF_TypeDef * SIFx )
```

清除SIF中断标志

在文件 [lks32mc09x_sif.c](#) 第 109 行定义.

3.39.2.2 SIF_DMA_IsEnable()

```
void SIF_DMA_IsEnable (
    SIF_TypeDef * SIFx,
    uint8_t state )
```

设置SIF DMA使能状态

参数

<i>state</i>	使能状态，1表示使能，0表示禁用

在文件 [lks32mc09x_sif.c](#) 第 78 行定义.

3.39.2.3 SIF_Init()

```
void SIF_Init (
    SIF_TypeDef * SIFx,
    SIF_InitTypeDef * SIFInitStruct )
```

初始化SIF模块

参数

<i>SIFx</i>	指向SIF_TypeDef结构体的指针，表示具体的SIF实例
<i>SIFInitStruct</i>	指向SIF_InitTypeDef结构体的指针，包含初始化参数

在文件 [lks32mc09x_sif.c](#) 第 22 行定义.

函数调用图:



3.39.2.4 SIF_Irq_IsEnable()

```
void SIF_Irq_IsEnable (
    SIF_TypeDef * SIFx,
    uint8_t state )
```

设置SIF中断使能状态

参数

<i>state</i>	使能状态，1表示使能，0表示禁用

在文件 [lks32mc09x_sif.c](#) 第 94 行定义.

3.39.2.5 SIF_Senddata()

```
void SIF_Senddata (
    SIF_TypeDef * SIFx,
    uint8_t Data )
```

发送数据

参数

<i>Data</i>	要发送的数据

在文件 [lks32mc09x_sif.c](#) 第 69 行定义.

3.39.2.6 SIF_StructInit()

```
void SIF_StructInit (
    SIF_InitTypeDef * SIFInitStruct )
```

初始化SIF结构体

参数

<i>SIFInitStruct</i>	指向SIF_InitTypeDef结构体的指针

在文件 [lks32mc09x_sif.c](#) 第 8 行定义.

3.39.2.7 SIF_TH.set()

```
void SIF_TH.set (
    SIF_TypeDef * SIFx,
    uint32_t tsh,
    uint32_t tdh )
```

设置SIF同步和结束信号时长

参数

<i>tsh</i>	同步信号时长
<i>tdh</i>	结束信号时长

在文件 [lks32mc09x_sif.c](#) 第 59 行定义.

3.39.2.8 SIF_Timebase_set()

```
void SIF_Timebase_set (
    SIF_TypeDef * SIFx,
    uint32_t tosc )
```

设置SIF时基

参数

<i>tosc</i>	时基值

在文件 [lks32mc09x_sif.c](#) 第 49 行定义.

3.40 lks32mc09x_sif.h

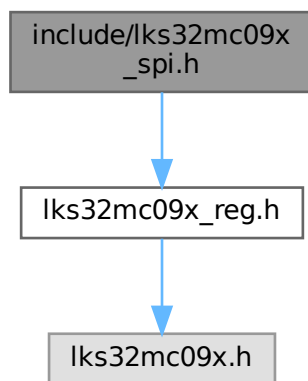
[浏览该文件的文档.](#)

```
00001 #ifndef __LKS32MC09X_SIF_H__
00002 #define __LKS32MC09X_SIF_H__
00003
00004 #include "lks32mc09x_reg.h"
00005
00009 typedef struct
00010 {
00011     uint8_t SIF_DONE;
00012     uint8_t SIF_SYNC;
00013     uint8_t SIF_SYNC_PULSE;
00014     uint8_t SIF_DONE_VLD;
00015     uint8_t SIF_SYNC_VLD;
00016     uint8_t SIF_RATIO;
00017     uint8_t SIF_MSB;
00018     uint8_t SIF_EN;
00019     uint16_t SIF_TOSC;
00020     uint16_t SIF_TSTH1;
00021     uint8_t SIF_TDTH1;
00022     uint8_t SIF_IRQ_IF;
00023     uint8_t SIF_DMA_EN;
00024     uint8_t SIF_IRQ_EN;
00025 } SIF_InitTypeDef;
00026
00030 #define SIF_IE_Enable BIT0
00031 #define SIF_DMA_Enable BIT1
00032
00036 #define SIF_IF_CLEAR BIT4
00037
00042 void SIF_StructInit(SIF_InitTypeDef *SIFInitStruct);
00043
00049 void SIF_Init(SIF_TypeDef *SIFx, SIF_InitTypeDef *SIFInitStruct);
00050
00055 void SIF_Timebase_set(SIF_TypeDef *SIFx, uint32_t tosc);
00056
00062 void SIF_TH_set(SIF_TypeDef *SIFx, uint32_t tsh, uint32_t tdh);
00063
00068 void SIF_Senddata(SIF_TypeDef *SIFx, uint8_t Data);
00069
00074 void SIF_DMA_IsEnable(SIF_TypeDef *SIFx, uint8_t state);
00075
00080 void SIF_Irq_IsEnable(SIF_TypeDef *SIFx, uint8_t state);
00081
00085 void SIF_Clear_Irq(SIF_TypeDef *SIFx);
00086
00087 #endif // __LKS32MC09X_SIF_H__
```

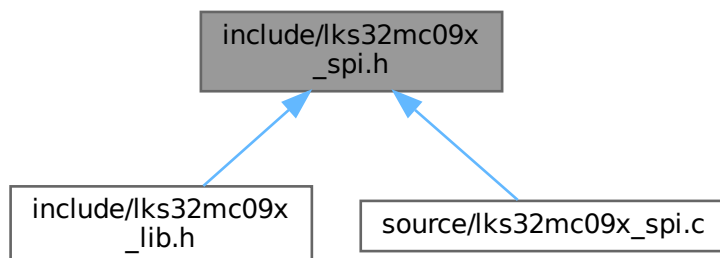
3.41 include/lks32mc09x_spi.h 文件参考

```
#include "lks32mc09x_reg.h"
```

lks32mc09x_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:



结构体

- struct **SPI_InitTypeDef**
SPI模块初始化结构体

宏定义

- #define **SPI_FIRSTSEND_LSB** 1
SPI数据传输顺序宏定义

- `#define SPI_FIRSTSEND_MSB 0`
高位先传
- `#define SPI_DMA_ENABLE 0`
*SPI*数据搬移方式宏定义
- `#define SPI_DMA_DISABLE 1`
禁止*DMA*搬运
- `#define SPI_IRQEna.Enable BIT7`
*SPI*中断使能位定义
- `#define SPI_IRQEna.TranDone BIT6`
传输完成中断使能
- `#define SPI_IRQEna.SSErr BIT5`
传输异常事件中断使能
- `#define SPI_IRQEna.DataOver BIT4`
传输溢出事件中断使能
- `#define SPI_IF_TranDone BIT2`
*SPI*中断标志位定义
- `#define SPI_IF_SSErr BIT1`
传输异常中断标志位
- `#define SPI_IF_DataOver BIT0`
传输溢出中断标志位

枚举

- `enum SPI_Mode { SPI_Master = 0x01 , SPI_Slave = 0x00 }`
*SPI*模块主从模式枚举
- `enum SPI_Duplex { SPI_Full = 0x00 , SPI_Half_Tx = 0x02 , SPI_Half_Rx = 0x03 }`
*SPI*模块全双工/半双工模式枚举

函数

- `void SPI_Init (SPI_TypeDef *SPIx, SPI_InitTypeDef *SPI_InitStruct)`
*SPI*模块初始化函数
- `void SPI_StructInit (SPI_InitTypeDef *SPI_InitStruct)`
*SPI*初始化结构体默认配置函数
- `void SPI_SendData (SPI_TypeDef *SPIx, uint8_t n)`
*SPI*发送数据函数
- `uint8_t SPI_ReadData (SPI_TypeDef *SPIx)`
*SPI*读取数据函数
- `uint8_t SPI_GetIRQFlag (SPI_TypeDef *SPIx, uint8_t IRQ_Flag)`
获取*SPI*中断标志位状态函数
- `void SPI_ClearIRQFlag (SPI_TypeDef *SPIx, uint8_t IRQ_Flag)`
清除*SPI*中断标志位函数

3.41.1 宏定义说明

3.41.1.1 SPI_DMA_DISABLE

```
#define SPI_DMA_DISABLE 1
```

禁止*DMA*搬运

在文件 [lks32mc09x_spi.h](#) 第 54 行定义。

3.41.1.2 SPI_DMA_ENABLE

```
#define SPI_DMA_ENABLE 0
```

SPI数据搬移方式宏定义

使能DMA搬运

在文件 [lks32mc09x_spi.h](#) 第 53 行定义.

3.41.1.3 SPI_FIRSTSEND_LSB

```
#define SPI_FIRSTSEND_LSB 1
```

SPI数据传输顺序宏定义

低位先传

在文件 [lks32mc09x_spi.h](#) 第 47 行定义.

3.41.1.4 SPI_FIRSTSEND_MSB

```
#define SPI_FIRSTSEND_MSB 0
```

高位先传

在文件 [lks32mc09x_spi.h](#) 第 48 行定义.

3.41.1.5 SPI_IF_DataOver

```
#define SPI_IF_DataOver BIT0
```

传输溢出中断标志位

在文件 [lks32mc09x_spi.h](#) 第 69 行定义.

3.41.1.6 SPI_IF_SSErr

```
#define SPI_IF_SSErr BIT1
```

传输异常中断标志位

在文件 [lks32mc09x_spi.h](#) 第 68 行定义.

3.41.1.7 SPI_IF_TranDone

```
#define SPI_IF_TranDone BIT2
```

SPI中断标志位定义

传输完成中断标志位

在文件 [lks32mc09x_spi.h](#) 第 67 行定义.

3.41.1.8 SPI_IRQEna_DataOver

#define SPI_IRQEna_DataOver BIT4

传输溢出事件中断使能

在文件 lks32mc09x_spi.h 第 62 行定义.

3.41.1.9 SPI_IRQEna_Enable

#define SPI_IRQEna_Enable BIT7

SPI中断使能位定义

SPI中断使能

在文件 lks32mc09x_spi.h 第 59 行定义.

3.41.1.10 SPI_IRQEna_SSErr

#define SPI_IRQEna_SSErr BIT5

传输异常事件中断使能

在文件 lks32mc09x_spi.h 第 61 行定义.

3.41.1.11 SPI_IRQEna-TranDone

#define SPI_IRQEna-TranDone BIT6

传输完成中断使能

在文件 lks32mc09x_spi.h 第 60 行定义.

3.41.2 枚举类型说明

3.41.2.1 SPI_Duplex

enum SPI_Duplex

SPI模块全双工/半双工模式枚举

枚举值

SPI_Full	全双工模式
SPI_Half_Tx	半双工模式，仅发送
SPI_Half_Rx	半双工模式，仅接收

在文件 [lks32mc09x_spi.h](#) 第 18 行定义.

3.41.2.2 SPI_Mode

enum [SPI_Mode](#)

SPI模块主从模式枚举

枚举值

<code>SPI_Master</code>	主模式
<code>SPI_Slave</code>	从模式

在文件 [lks32mc09x_spi.h](#) 第 9 行定义.

3.41.3 函数说明

3.41.3.1 SPI_ClearIRQFlag()

```
void SPI_ClearIRQFlag (
    SPI\_TypeDef * SPIx,
    uint8_t IRQ_Flag )
```

清除SPI中断标志位函数

参数

<code>SPIx</code>	SPI模块指针
<code>IRQ_Flag</code>	要清除的中断标志位

在文件 [lks32mc09x_spi.c](#) 第 84 行定义.

3.41.3.2 SPI_GetIRQFlag()

```
uint8_t SPI_GetIRQFlag (
    SPI\_TypeDef * SPIx,
    uint8_t IRQ_Flag )
```

获取SPI中断标志位状态函数

参数

<code>SPIx</code>	SPI模块指针
<code>IRQ_Flag</code>	中断标志位

返回

中断标志位状态

在文件 [lks32mc09x_spi.c](#) 第 74 行定义.

3.41.3.3 SPI_Init()

```
void SPI_Init (
    SPI_TypeDef * SPIx,
    SPI_InitTypeDef * SPI_InitStruct )
```

SPI模块初始化函数

参数

<i>SPIx</i>	SPI模块指针
<i>SPI_InitStruct</i>	指向SPI初始化结构体的指针

在文件 [lks32mc09x_spi.c](#) 第 9 行定义.

函数调用图:



3.41.3.4 SPI_ReadData()

```
uint8_t SPI_ReadData (
    SPI_TypeDef * SPIx )
```

SPI读取数据函数

参数

<i>SPIx</i>	SPI模块指针

返回

读取到的数据

在文件 [lks32mc09x_spi.c](#) 第 63 行定义.

3.41.3.5 SPI_SendData()

```
void SPI_SendData (
    SPI_TypeDef * SPIx,
    uint8_t n )
```

SPI发送数据函数

参数

<i>SPIx</i>	SPI模块指针
<i>n</i>	要发送的数据

在文件 [lks32mc09x_spi.c](#) 第 53 行定义.

3.41.3.6 SPI_StructInit()

```
void SPI_StructInit (
    SPI_InitTypeDef * SPI_InitStruct )
```

SPI初始化结构体默认配置函数

参数

<i>SPI_InitStruct</i>	指向SPI初始化结构体的指针

在文件 [lks32mc09x_spi.c](#) 第 39 行定义.

3.42 lks32mc09x_spi.h

浏览该文件的文档.

```
00001 #ifndef __LKS32MC09X_SPI_H__
00002 #define __LKS32MC09X_SPI_H__
00003
00004 #include "lks32mc09x_reg.h"
00005
00009 typedef enum
00010 {
00011     SPI_Master = 0x01,
00012     SPI_Slave = 0x00
00013 } SPI_Mode;
00014
00018 typedef enum
00019 {
00020     SPI_Full = 0x00,
00021     SPI_Half_Tx = 0x02,
00022     SPI_Half_Rx = 0x03
00023 } SPI_Duplex;
00024
00028 typedef struct
00029 {
00030     SPI_Duplex Duplex;
00031     uint8_t CS;
00032     SPI_Mode Mode;
00033     uint8_t CPHA;
00034     uint8_t CPOL;
00035     uint8_t ENDIAN;
00036     uint8_t EN;
00037     uint8_t IRQEna;
```

```

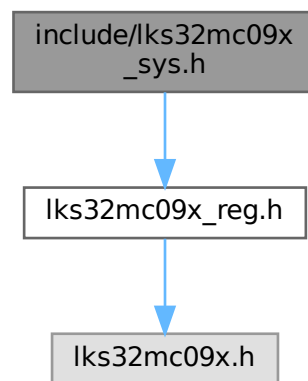
00038     uint8_t Trig;
00039     uint8_t TRANS_MODE;
00040     uint16_t BaudRate;
00041     uint8_t ByteLength;
00042 } SPI_InitTypeDef;
00043
00047 #define SPI_FIRSTSEND_LSB 1
00048 #define SPI_FIRSTSEND_MSB 0
00049
00053 #define SPI_DMA_ENABLE 0
00054 #define SPI_DMA_DISABLE 1
00055
00059 #define SPI_IRQEna_Enable BIT7
00060 #define SPI_IRQEna-TranDone BIT6
00061 #define SPI_IRQEna_SSErr BIT5
00062 #define SPI_IRQEna-DataOver BIT4
00063
00067 #define SPI_IF-TranDone BIT2
00068 #define SPI_IF_SSErr BIT1
00069 #define SPI_IF-DataOver BIT0
00070
00076 void SPI_Init(SPI_TypeDef *SPIx, SPI_InitTypeDef *SPI_InitStruct);
00077
00082 void SPI_StructInit(SPI_InitTypeDef *SPI_InitStruct);
00083
00089 void SPI_SendData(SPI_TypeDef *SPIx, uint8_t n);
00090
00096 uint8_t SPI_ReadData(SPI_TypeDef *SPIx);
00097
00104 uint8_t SPI_GetIRQFlag(SPI_TypeDef *SPIx, uint8_t IRQ_Flag);
00105
00111 void SPI_ClearIRQFlag(SPI_TypeDef *SPIx, uint8_t IRQ_Flag);
00112
00113 #endif // __LKS32MC09X_SPI_H__

```

3.43 include/lks32mc09x_sys.h 文件参考

#include "lks32mc09x_reg.h"

lks32mc09x_sys.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 [SYS_MODULE_EEPROM](#) BIT26
- #define [SYS_MODULE_ADC1](#) BIT23
- #define [SYS_MODULE_ADC0](#) BIT22
- #define [SYS_MODULE_CL0](#) BIT21
- #define [SYS_MODULE_SIF0](#) BIT20
- #define [SYS_MODULE_DMA0](#) BIT18
- #define [SYS_MODULE_DSP0](#) BIT17
- #define [SYS_MODULE_CRC0](#) BIT16
- #define [SYS_MODULE_UART2](#) BIT14
- #define [SYS_MODULE_UART1](#) BIT13
- #define [SYS_MODULE_UART0](#) BIT12
- #define [SYS_MODULE_GPIO](#) BIT11
- #define [SYS_MODULE_MCPWM0](#) BIT10
- #define [SYS_MODULE_QEP0](#) BIT8
- #define [SYS_MODULE_TIMER2](#) BIT6
- #define [SYS_MODULE_TIMER1](#) BIT5
- #define [SYS_MODULE_TIMER0](#) BIT4
- #define [SYS_MODULE_HALL0](#) BIT3
- #define [SYS_MODULE_CMP](#) BIT2
- #define [SYS_MODULE_I2C0](#) BIT1
- #define [SYS_MODULE_SPI0](#) BIT0
- #define [SYS_MCLK_RC_96M](#) 0x1ff
- #define [SYS_MCLK_RC_48M](#) 0x155
- #define [SYS_MCLK_RC_24M](#) 0x111
- #define [SYS_MCLK_RC_12M](#) 0x101
- #define [SYS_MCLK_RC_8M](#) 0x000
- #define [SYS_MCLK_XTAL_96M](#) 0x81ff
- #define [SYS_MCLK_XTAL_48M](#) 0x8155
- #define [SYS_MCLK_XTAL_24M](#) 0x8111
- #define [SYS_MCLK_XTAL_12M](#) 0x8101
- #define [SYS_PVDSEL_4_0V](#) 0
- #define [SYS_PVDSEL_3_74V](#) BIT13
- #define [SYS_PVDSEL_3_49V](#) BIT14
- #define [SYS_PVDSEL_2_8V](#) BIT13 | BIT14
- #define [SYS_PVDSEL_DAC](#) BIT12
- #define [SYS_PVDSEL_OFF](#) BIT11
- #define [SYS_PVDSEL_3_75V](#) [SYS_PVDSEL_3_74V](#)
- #define [SYS_PVDSEL_3_5V](#) [SYS_PVDSEL_3_49V](#)

函数

- void [SYS_ModuleClockCmd](#) (uint32_t SYS_MODULE_x, uint8_t state)
数字模块时钟使能
- void [SYS_SoftResetModule](#) (uint32_t nModule)
软复位模块
- void [SYS_SoftReset_all](#) (void)
软复位所有模块
- void [SYS_MclkChoice](#) (uint32_t SYS_MCLK_x)
选择系统时钟
- void [SYS_UartClkDiv](#) (uint16_t div)
*UART*时钟分频

- void **SYS_I2CCLKDiv** (uint16_t div)
I2C时钟分频
- void **SYS_SPICLKDiv** (uint16_t div)
SP时钟分频
- uint32_t **SYS_ReadMcuCik** (void)
获取当前MCU时钟
- void **SYS_SetPowerFailureDetection** (uint32_t SYS_PVDSEL_x)
掉电检测配置
- void **SYS_WaitPowerReady** (uint32_t SYS_PVDSEL_x)
等待电源就绪
- void **SYS_ResetPeripheral** (void)
复位所有外设到默认状态（芯片上电后的状态）
- void **SYS_ResetPeripheralInternal** (void)
复位所有内部外设为默认值(除io的所有模块)
- void **SYS_ResetIO** (void)
复位所有IO为默认值（gpio以及部分和io有关的sys寄存器）

3.43.1 宏定义说明

3.43.1.1 SYS_MCLK_RC_12M

```
#define SYS_MCLK_RC_12M 0x101
```

在文件 [lks32mc09x_sys.h](#) 第 32 行定义.

3.43.1.2 SYS_MCLK_RC_24M

```
#define SYS_MCLK_RC_24M 0x111
```

在文件 [lks32mc09x_sys.h](#) 第 31 行定义.

3.43.1.3 SYS_MCLK_RC_48M

```
#define SYS_MCLK_RC_48M 0x155
```

在文件 [lks32mc09x_sys.h](#) 第 30 行定义.

3.43.1.4 SYS_MCLK_RC_8M

```
#define SYS_MCLK_RC_8M 0x000
```

在文件 [lks32mc09x_sys.h](#) 第 33 行定义.

3.43.1.5 SYS_MCLK_RC_96M

```
#define SYS_MCLK_RC_96M 0x1ff
```

在文件 [lks32mc09x_sys.h](#) 第 29 行定义.

3.43.1.6 SYS_MCLK_XTAL_12M

```
#define SYS_MCLK_XTAL_12M 0x8101
```

在文件 [lks32mc09x.sys.h](#) 第 38 行定义.

3.43.1.7 SYS_MCLK_XTAL_24M

```
#define SYS_MCLK_XTAL_24M 0x8111
```

在文件 [lks32mc09x.sys.h](#) 第 37 行定义.

3.43.1.8 SYS_MCLK_XTAL_48M

```
#define SYS_MCLK_XTAL_48M 0x8155
```

在文件 [lks32mc09x.sys.h](#) 第 36 行定义.

3.43.1.9 SYS_MCLK_XTAL_96M

```
#define SYS_MCLK_XTAL_96M 0x81ff
```

在文件 [lks32mc09x.sys.h](#) 第 35 行定义.

3.43.1.10 SYS_MODULE_ADC0

```
#define SYS_MODULE_ADC0 BIT22
```

在文件 [lks32mc09x.sys.h](#) 第 9 行定义.

3.43.1.11 SYS_MODULE_ADC1

```
#define SYS_MODULE_ADC1 BIT23
```

在文件 [lks32mc09x.sys.h](#) 第 8 行定义.

3.43.1.12 SYS_MODULE_CL0

```
#define SYS_MODULE_CL0 BIT21
```

在文件 [lks32mc09x.sys.h](#) 第 10 行定义.

3.43.1.13 SYS_MODULE_CMP

```
#define SYS_MODULE_CMP BIT2
```

在文件 [lks32mc09x.sys.h](#) 第 25 行定义.

3.43.1.14 SYS_MODULE_CRC0

```
#define SYS_MODULE_CRC0 BIT16
```

在文件 [lks32mc09x.sys.h](#) 第 14 行定义.

3.43.1.15 SYS_MODULE_DMA0

```
#define SYS_MODULE_DMA0 BIT18
```

在文件 [lks32mc09x.sys.h](#) 第 12 行定义.

3.43.1.16 SYS_MODULE_DSP0

```
#define SYS_MODULE_DSP0 BIT17
```

在文件 [lks32mc09x.sys.h](#) 第 13 行定义.

3.43.1.17 SYS_MODULE_EEPROM

```
#define SYS_MODULE_EEPROM BIT26
```

在文件 [lks32mc09x.sys.h](#) 第 7 行定义.

3.43.1.18 SYS_MODULE_GPIO

```
#define SYS_MODULE_GPIO BIT11
```

在文件 [lks32mc09x.sys.h](#) 第 18 行定义.

3.43.1.19 SYS_MODULE_HALL0

```
#define SYS_MODULE_HALL0 BIT3
```

在文件 [lks32mc09x.sys.h](#) 第 24 行定义.

3.43.1.20 SYS_MODULE_I2C0

```
#define SYS_MODULE_I2C0 BIT1
```

在文件 [lks32mc09x.sys.h](#) 第 26 行定义.

3.43.1.21 SYS_MODULE_MCPWM0

```
#define SYS_MODULE_MCPWM0 BIT10
```

在文件 [lks32mc09x.sys.h](#) 第 19 行定义.

3.43.1.22 SYS_MODULE_QEP0

```
#define SYS_MODULE_QEP0 BIT8
```

在文件 [lks32mc09x.sys.h](#) 第 20 行定义.

3.43.1.23 SYS_MODULE_SIF0

```
#define SYS_MODULE_SIF0 BIT20
```

在文件 [lks32mc09x.sys.h](#) 第 11 行定义.

3.43.1.24 SYS_MODULE_SPI0

```
#define SYS_MODULE_SPI0 BIT0
```

在文件 [lks32mc09x.sys.h](#) 第 27 行定义.

3.43.1.25 SYS_MODULE_TIMER0

```
#define SYS_MODULE_TIMER0 BIT4
```

在文件 [lks32mc09x.sys.h](#) 第 23 行定义.

3.43.1.26 SYS_MODULE_TIMER1

```
#define SYS_MODULE_TIMER1 BIT5
```

在文件 [lks32mc09x.sys.h](#) 第 22 行定义.

3.43.1.27 SYS_MODULE_TIMER2

```
#define SYS_MODULE_TIMER2 BIT6
```

在文件 [lks32mc09x.sys.h](#) 第 21 行定义.

3.43.1.28 SYS_MODULE_UART0

```
#define SYS_MODULE_UART0 BIT12
```

在文件 [lks32mc09x.sys.h](#) 第 17 行定义.

3.43.1.29 SYS_MODULE_UART1

```
#define SYS_MODULE_UART1 BIT13
```

在文件 [lks32mc09x.sys.h](#) 第 16 行定义.

3.43.1.30 SYS_MODULE_UART2

```
#define SYS_MODULE_UART2 BIT14
```

在文件 [lks32mc09x.sys.h](#) 第 15 行定义.

3.43.1.31 SYS_PVDSEL_2.8V

```
#define SYS_PVDSEL_2.8V BIT13 | BIT14
```

在文件 [lks32mc09x.sys.h](#) 第 43 行定义.

3.43.1.32 SYS_PVDSEL_3.49V

```
#define SYS_PVDSEL_3.49V BIT14
```

在文件 [lks32mc09x.sys.h](#) 第 42 行定义.

3.43.1.33 SYS_PVDSEL_3.5V

```
#define SYS_PVDSEL_3.5V SYS\_PVDSEL\_3.49V
```

在文件 [lks32mc09x.sys.h](#) 第 48 行定义.

3.43.1.34 SYS_PVDSEL_3.74V

```
#define SYS_PVDSEL_3.74V BIT13
```

在文件 [lks32mc09x.sys.h](#) 第 41 行定义.

3.43.1.35 SYS_PVDSEL_3.75V

```
#define SYS_PVDSEL_3.75V SYS\_PVDSEL\_3.74V
```

在文件 [lks32mc09x.sys.h](#) 第 47 行定义.

3.43.1.36 SYS_PVDSEL_4.0V

```
#define SYS_PVDSEL_4.0V 0
```

在文件 [lks32mc09x.sys.h](#) 第 40 行定义.

3.43.1.37 SYS_PVDSEL_DAC

```
#define SYS_PVDSEL_DAC BIT12
```

在文件 [lks32mc09x.sys.h](#) 第 44 行定义.

3.43.1.38 SYS_PVDSEL_OFF

```
#define SYS_PVDSEL_OFF BIT11
```

在文件 [lks32mc09x.sys.h](#) 第 45 行定义.

3.43.2 函数说明

3.43.2.1 SYS_I2CClkDiv()

```
void SYS_I2CClkDiv (
    uint16_t div )
```

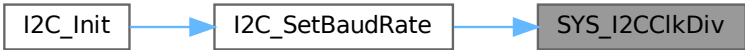
I2C时钟分频

参数

<i>div</i>	分频系数

在文件 [lks32mc09x.sys.c](#) 第 91 行定义.

这是这个函数的调用关系图:



3.43.2.2 SYS_MclkChoice()

```
void SYS_MclkChoice (
    uint32_t SYS_MCLK_x )
```

选择系统时钟

参数

<i>SYS_MCLK_x</i>	时钟源选择
<i>SYS_MCLK_x</i>	时钟源选择

参见

使用外部晶振时，除了晶振使能外，还需要配置 `SYS_AFE_REG1_XTAL_FLT_BIT` [9] 晶振低通滤波关闭 1:关闭 0:打开 `SYS_AFE_REG1_XTAL_SCHT_BIT` [4] 晶振选择反相器或者施密特 1:反相器 0:施密特 `SYS_AFE_REG2_XTAL_RSEL_BIT` [8] 晶体起振电路电阻调节 0:P端电阻维持原值 1: P端电阻增加一倍

在文件 `lks32mc09x_sys.c` 第 54 行定义.

3.43.2.3 `SYS_ModuleClockCmd()`

```
void SYS_ModuleClockCmd (
    uint32_t nModule,
    uint8_t state )
```

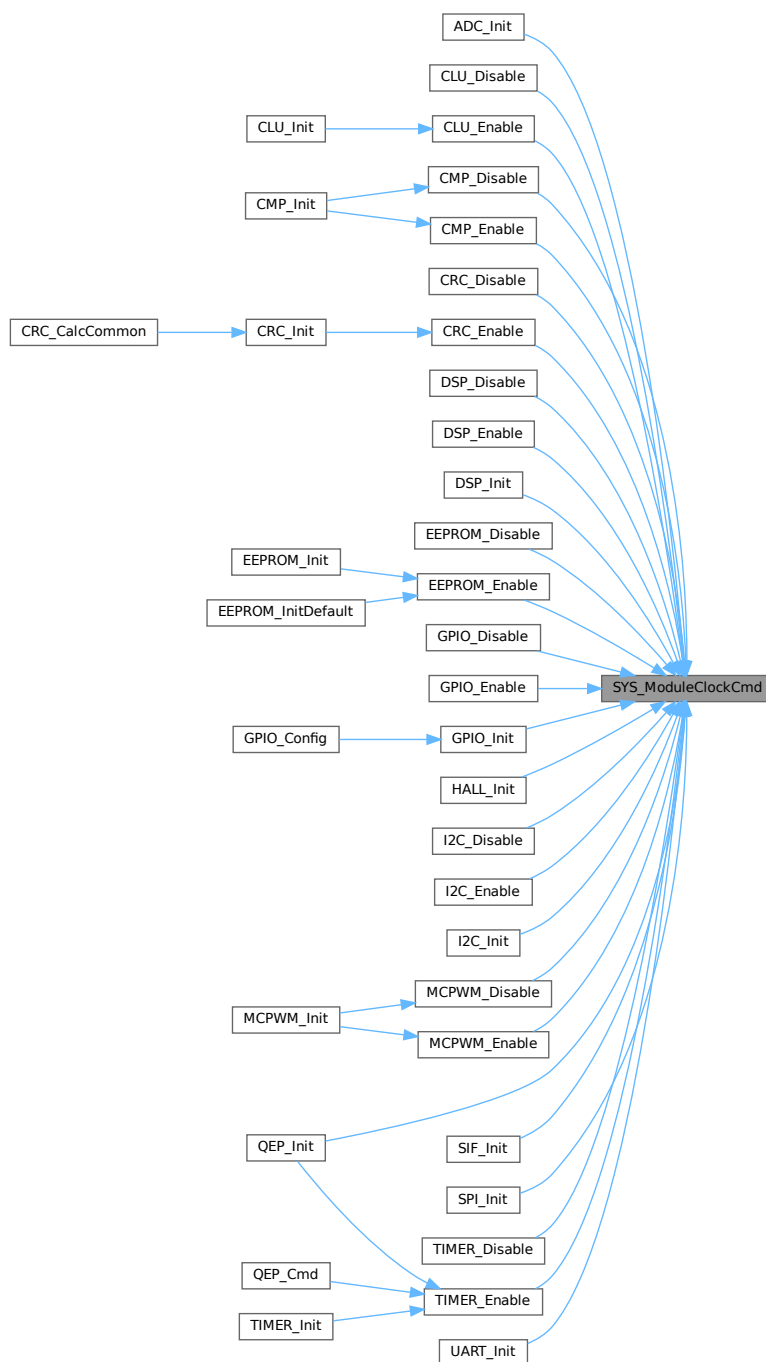
数字模块时钟使能

参数

<i>nModule</i>	模块编号
<i>state</i>	使能或禁用状态

在文件 `lks32mc09x_sys.c` 第 9 行定义.

这是这个函数的调用关系图:



3.43.2.4 SYS_ReadMcuClk()

```
uint32_t SYS_ReadMcuClk (
    void )
```

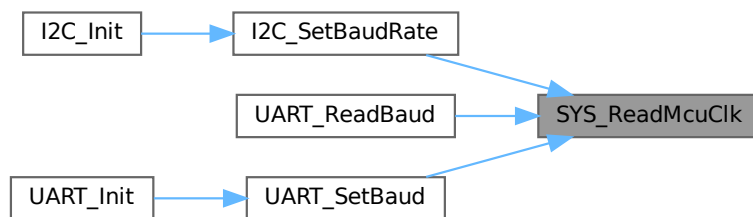
获取当前MCU时钟

返回

当前MCU时钟频率

在文件 [lks32mc09x-sys.c](#) 第 115 行定义.

这是这个函数的调用关系图:



3.43.2.5 SYS_ResetIO()

```
void SYS_ResetIO (
    void )
```

复位所有IO为默认值（gpio以及部分和io有关的sys寄存器）

在文件 [lks32mc09x-sys.c](#) 第 308 行定义.

这是这个函数的调用关系图:



3.43.2.6 SYS_ResetPeripheral()

```
void SYS_ResetPeripheral (
    void )
```

复位所有外设到默认状态（芯片上电后的状态）

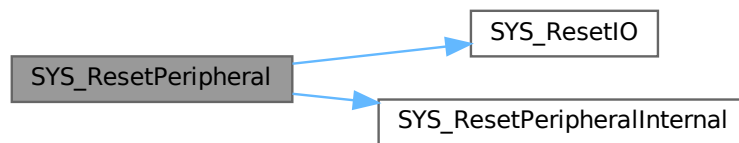
警告

不复位的寄存器有：复位事件记录寄存器 后备存储器 ram 校准寄存器（仅限于上电会自动加载的校准寄存器）所有内核的寄存器

不复位的寄存器有：复位事件记录寄存器 后备存储器 ram 校准寄存器（仅限于上电会自动加载的校准寄存器）只读的寄存器 部分flash寄存器（如FLASH->WDATA等，写入操作无意义）所有内核的寄存器

在文件 [lks32mc09x_sys.c](#) 第 238 行定义.

函数调用图:



3.43.2.7 SYS_ResetPeripheralInternal()

```
void SYS_ResetPeripheralInternal (
    void )
```

复位所有内部外设为默认值(除io的所有模块)

警告

不复位的寄存器有：复位事件记录寄存器 后备存储器 ram 校准寄存器（仅限于上电会自动加载的校准寄存器）所有内核的寄存器

不复位的寄存器有：复位事件记录寄存器 后备存储器 ram 校准寄存器（仅限于上电会自动加载的校准寄存器）只读的寄存器 部分flash寄存器（如FLASH->WDATA等，写入操作无意义）所有内核的寄存器

在文件 [lks32mc09x_sys.c](#) 第 254 行定义.

这是这个函数的调用关系图:



3.43.2.8 SYS_SetPowerFailureDetection()

```
void SYS_SetPowerFailureDetection (
    uint32_t SYS_PVDSEL_x )
```

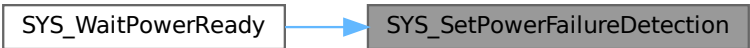
掉电检测配置

参数

SYS↔ PVDSEL_x	掉电检测配置值，可选： • SYS.PVDSEL_4.0V 4.0V检测阈值 • SYS.PVDSEL_3.74V 3.74V检测阈值 • SYS.PVDSEL_3.49V 3.49V检测阈值 • SYS.PVDSEL_2.8V 2.8V检测阈值 • SYS.PVDSEL_DAC DAC (这里使用的是DAC0) • SYS.PVDSEL_OFF OFF

在文件 [lks32mc09x_sys.c](#) 第 178 行定义.

这是这个函数的调用关系图:



3.43.2.9 SYS.SoftReset_all()

```
void SYS_SoftReset_all (
    void )
```

软复位所有模块

在文件 [lks32mc09x_sys.c](#) 第 38 行定义.

3.43.2.10 SYS.SoftResetModule()

```
void SYS_SoftResetModule (
    uint32_t nModule )
```

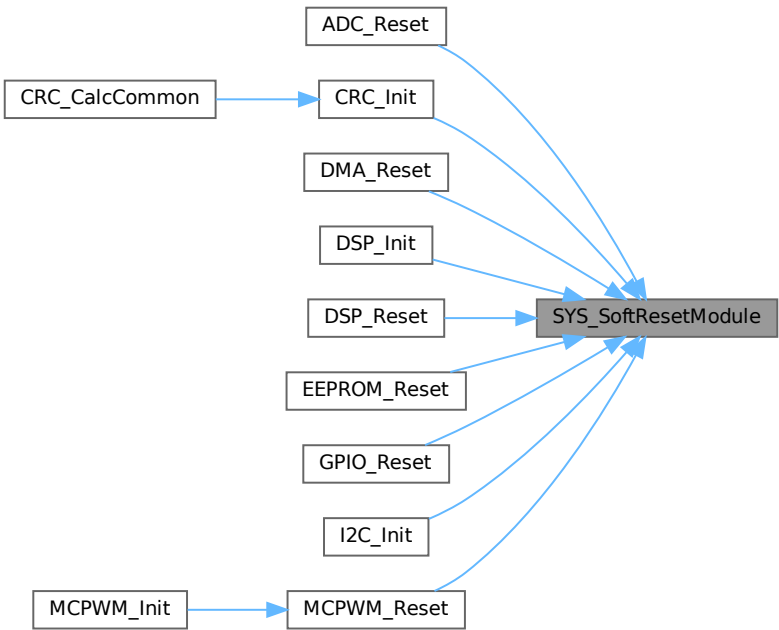
软复位模块

参数

<i>nModule</i>	模块编号

在文件 [lks32mc09x.sys.c](#) 第 27 行定义.

这是这个函数的调用关系图:



3.43.2.11 SYS.SPIClkDiv()

```
void SYS_SPIClkDiv (
    uint16_t div )
```

SPI时钟分频

参数

<i>div</i>	分频系数

在文件 [lks32mc09x.sys.c](#) 第 102 行定义.

3.43.2.12 SYS.UartClkDiv()

```
void SYS_UartClkDiv (
    uint16_t div )
```

UART时钟分频

参数

<i>div</i>	分频系数

在文件 [lks32mc09x_sys.c](#) 第 80 行定义.

这是这个函数的调用关系图:



3.43.2.13 SYS.WaitPowerReady()

```
void SYS.WaitPowerReady (
    uint32_t SYS_PVDSEL_x )
```

等待电源就绪

参数

SYS_↔ PVDSEL_x	电源电压检测阈值配置，可选： • SYS.PVDSEL_4.0V 4.0V检测阈值 • SYS.PVDSEL_3.74V 3.74V检测阈值 • SYS.PVDSEL_3.49V 3.49V检测阈值 • SYS.PVDSEL_2.8V 2.8V检测阈值 注意：SYS.PVDSEL_DAC暂不支持

注解

```
使用前确保已配置PWRDN中断（如需要）
void SystemInit()
{
    SYS.WaitPowerReady(SYS_PVDSEL_4.0V); // 等待电源就绪
    NVIC_SetPriority(PWRDN_IRQn, 0);      // 掉电检测中断优先级配置
    NVIC_EnableIRQ(PWRDN_IRQn);          // 使能掉电检测中断
    // 其他初始化代码
}
void PWRDN_IRQHandler(void) {
    __disable_irq();
    SYS.ResetPeripheral();
    NVIC_SystemReset();
}
```

参数

SYS_↔ PVDSEL_x	电源电压检测阈值配置，可选： • SYS_PVDSEL_4.0V 4.0V • SYS_PVDSEL_3.74V 3.74V • SYS_PVDSEL_3.49V 3.49V • SYS_PVDSEL_2.8V 2.8V 注意：SYS_PVDSEL_DAC暂不支持

注解

```
使用前确保已配置PWRDN中断（如需要）
void SystemInit()
{
    SYS_WaitPowerReady(SYS_PVDSEL_4.0V); // 等待电源就绪
    NVIC_SetPriority(PWRDN_IRQn, 0);      // 掉电检测中断优先级配置
    NVIC_EnableIRQ(PWRDN_IRQn);          // 使能掉电检测中断
    // 其他初始化代码
}
void PWRDN_IRQHandler(void) {
    __disable_irq();
    SYS_ResetPeripheral();
    NVIC_SystemReset();
}
```

在文件 lks32mc09x_sys.c 第 209 行定义.

函数调用图:



3.44 lks32mc09x_sys.h

浏览该文件的文档.

```
00001 #ifndef __LKS32MC09X_SYS_H__
00002 #define __LKS32MC09X_SYS_H__
00003
00004 #include "lks32mc09x_reg.h"
00005
00006 // 定义各个模块标志
00007 #define SYS_MODULE_EEPROM BIT26 // EEPROM模块
00008 #define SYS_MODULE_ADC1 BIT23 // ADC1模块
00009 #define SYS_MODULE_ADC0 BIT22 // ADC0模块
00010 #define SYS_MODULE_CL0 BIT21 // CL0模块
00011 #define SYS_MODULE_SIF0 BIT20 // SIF0模块
00012 #define SYS_MODULE_DMA0 BIT18 // DMA0模块
00013 #define SYS_MODULE_DSP0 BIT17 // DSP0模块
00014 #define SYS_MODULE_CRC0 BIT16 // CRC0模块
00015 #define SYS_MODULE_UART2 BIT14 // UART2模块
00016 #define SYS_MODULE_UART1 BIT13 // UART1模块
00017 #define SYS_MODULE_UART0 BIT12 // UART0模块
00018 #define SYS_MODULE_GPIO BIT11 // GPIO模块
00019 #define SYS_MODULE_MCPWM0 BIT10 // MCPWM0模块
00020 #define SYS_MODULE_QEP0 BIT8 // QEP0模块
00021 #define SYS_MODULE_TIMER2 BIT6 // TIMER2模块
```

```

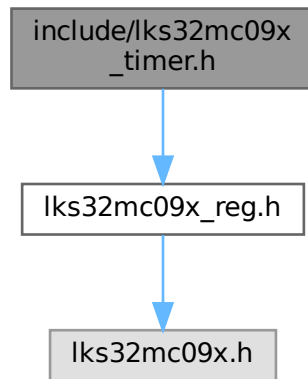
00022 #define SYS_MODULE_TIMER1 BIT5 // TIMER1模块
00023 #define SYS_MODULE_TIMER0 BIT4 // TIMER0模块
00024 #define SYS_MODULE_HALL0 BIT3 // HALL0模块
00025 #define SYS_MODULE_CMP BIT2 // CMP模块
00026 #define SYS_MODULE_I2C0 BIT1 // I2C0模块
00027 #define SYS_MODULE_SPI0 BIT0 // SPI0模块
00028
00029 #define SYS_MCLK_RC_96M 0x1ff // 96MHz RC时钟 默认值, 芯片上电后默认状态
00030 #define SYS_MCLK_RC_48M 0x155 // 48MHz RC时钟
00031 #define SYS_MCLK_RC_24M 0x111 // 24MHz RC时钟
00032 #define SYS_MCLK_RC_12M 0x101 // 12MHz RC时钟
00033 #define SYS_MCLK_RC_8M 0x000 // 8MHz RC时钟
00034 // 这里的0x8000用来区分RC和XTAL
00035 #define SYS_MCLK_XTAL_96M 0x81ff // 96MHz 外部晶体时钟
00036 #define SYS_MCLK_XTAL_48M 0x8155 // 48MHz 外部晶体时钟
00037 #define SYS_MCLK_XTAL_24M 0x8111 // 24MHz 外部晶体时钟
00038 #define SYS_MCLK_XTAL_12M 0x8101 // 12MHz 外部晶体时钟
00039
00040 #define SYS_PVDSEL_4_0V 0 // 4.0V
00041 #define SYS_PVDSEL_3_74V BIT13 // 3.75V
00042 #define SYS_PVDSEL_3_49V BIT14 // 3.5V
00043 #define SYS_PVDSEL_2_8V BIT13 | BIT14 // 2.8V
00044 #define SYS_PVDSEL_DAC BIT12 // DAC
00045 #define SYS_PVDSEL_OFF BIT11 // OFF
00046 // 兼容旧版本
00047 #define SYS_PVDSEL_3_75V SYS_PVDSEL_3_74V
00048 #define SYS_PVDSEL_3_5V SYS_PVDSEL_3_49V
00054 void SYS_ModuleClockCmd(uint32_t SYS_MODULE_x, uint8_t state);
00055
00060 void SYS_SoftResetModule(uint32_t nModule);
00061
00065 void SYS_SoftReset_all(void);
00066
00071 void SYS_MclkChoice(uint32_t SYS_MCLK_x);
00072
00077 void SYS_UartClkDiv(uint16_t div);
00078
00083 void SYS_I2CClkDiv(uint16_t div);
00084
00089 void SYS_SPIClkDiv(uint16_t div);
00090
00095 uint32_t SYS_ReadMcuClk(void);
00096
00107 void SYS_SetPowerFailureDetection(uint32_t SYS_PVDSEL_x);
00132 void SYS_WaitPowerReady(uint32_t SYS_PVDSEL_x);
00142 void SYS_ResetPeripheral(void);
00152 void SYS_ResetPeripheralInternal(void);
00156 void SYS_ResetIO(void);
00157 #endif // __LKS32MC09X_SYS_H__

```

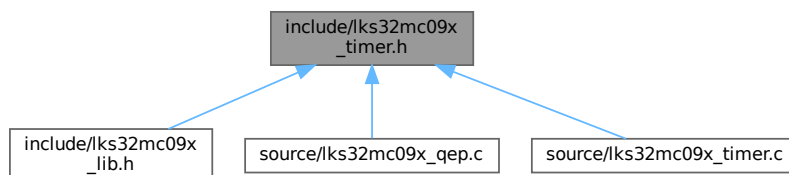

3.45 include/lks32mc09x_timer.h 文件参考

```
#include "lks32mc09x_reg.h"
```

lks32mc09x_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:



结构体

- struct **TIMER_TimerInitTypeDef**
*Timer*初始化结构体

宏定义

- #define **TIMER_IF_FAIL** BIT3
- #define **TIMER_IF_ZC** BIT2
- #define **TIMER_IF_CH1** BIT1
- #define **TIMER_IF_CH0** BIT0
- #define **TIMER_CH.MODE_CMP** 0
- #define **TIMER_CH.MODE_CAP** 1
- #define **TIMER_CH.MODE_Comparison** 0

- `#define TIMER_CH.MODE_Capture 1`
- `#define TIMER_StatisticsStart(TIMERx) TIMERx->CNT = 0`
*timer*时间统计, 开始(复位*timer*计数值)
- `#define TIMER_StatisticsStop(TIMERx) (TIMERx->CNT)`
*timer*时间统计, 结束

函数

- `void TIMER_Init (TIMER_TypeDef *TIMERx, TIMER_TimerInitTypeDef *TIMER_InitStruct)`
*Timer*初始化
- `void TIMER_StructInit (TIMER_TimerInitTypeDef *TIMER_InitStruct)`
*Timer*结构体初始化
- `void TIMER_TimerCmd (TIMER_TypeDef *TIMERx, uint8_t state)`
*Timer*使能/失能
- `void TIMER_CMP0 (TIMER_TypeDef *TIMERx, uint32_t temp)`
设置*Timer*通道0比较值
- `void TIMER_CMP1 (TIMER_TypeDef *TIMERx, uint32_t temp)`
设置*Timer*通道1比较值
- `uint32_t TIMER_Timer_GetCMPT0 (TIMER_TypeDef *TIMERx)`
获取*Timer*通道0比较值
- `uint32_t TIMER_Timer_GetCMPT1 (TIMER_TypeDef *TIMERx)`
获取*Timer*通道1比较值
- `void TIMER_ClearIRQFlag (TIMER_TypeDef *TIMERx, uint32_t tempFlag)`
清除*Timer*中断标志
- `uint16_t TIMER_GetIRQFlag (TIMER_TypeDef *TIMERx, uint32_t TIMER_IF_x)`
获取*Timer*中断标志状态
- `uint16_t TIMER_GetFlag (TIMER_TypeDef *TIMERx, uint32_t TIMER_IF_x)`
获取*Timer*状态标志
- `uint32_t TIMER_Timer_GetCount (TIMER_TypeDef *TIMERx)`
获取*Timer*当前计数值
- `void TIMER_Enable (TIMER_TypeDef *TIMERx)`
使能*Timer*
- `void TIMER_Disable (TIMER_TypeDef *TIMERx)`
失能*Timer*
- `void TIMER_SetOutputState (TIMER_TypeDef *TIMERx, uint8_t state)`
设置输出状态

3.45.1 宏定义说明

3.45.1.1 TIMER_CH.MODE_CAP

```
#define TIMER_CH.MODE_CAP 1
```

在文件 `lks32mc09x_timer.h` 第 58 行定义.

3.45.1.2 TIMER_CH.MODE_Capture

```
#define TIMER_CH.MODE_Capture 1
```

在文件 `lks32mc09x_timer.h` 第 60 行定义.

3.45.1.3 TIMER_CH_MODE_CMP

```
#define TIMER_CH_MODE_CMP 0
```

在文件 [lks32mc09x_timer.h](#) 第 57 行定义.

3.45.1.4 TIMER_CH_MODE_Comparison

```
#define TIMER_CH_MODE_Comparison 0
```

在文件 [lks32mc09x_timer.h](#) 第 59 行定义.

3.45.1.5 TIMER_IF_CH0

```
#define TIMER_IF_CH0 BIT0
```

在文件 [lks32mc09x_timer.h](#) 第 55 行定义.

3.45.1.6 TIMER_IF_CH1

```
#define TIMER_IF_CH1 BIT1
```

在文件 [lks32mc09x_timer.h](#) 第 54 行定义.

3.45.1.7 TIMER_IF_FAIL

```
#define TIMER_IF_FAIL BIT3
```

在文件 [lks32mc09x_timer.h](#) 第 52 行定义.

3.45.1.8 TIMER_IF_ZC

```
#define TIMER_IF_ZC BIT2
```

在文件 [lks32mc09x_timer.h](#) 第 53 行定义.

3.45.1.9 TIMER_StatisticsStart

```
#define TIMER_StatisticsStart(  
    TIMERx ) TIMERx->CNT = 0
```

timer时间统计，开始(复位timer计数值)

在文件 [lks32mc09x_timer.h](#) 第 64 行定义.

3.45.1.10 TIMER_StatisticsStop

```
#define TIMER_StatisticsStop(  
    TIMERx ) (TIMERx->CNT)
```

timer时间统计，结束

返回
时间统计值

在文件 [lks32mc09x_timer.h](#) 第 69 行定义.

3.45.2 函数说明

3.45.2.1 TIMER_ClearIRQFlag()

```
void TIMER_ClearIRQFlag (  
    TIMER_TypeDef * TIMERx,  
    uint32_t tempFlag )
```

清除Timer中断标志

参数

<i>TIMERx</i>	Timer实例指针
<i>tempFlag</i>	要清除的中断标志

在文件 [lks32mc09x_timer.c](#) 第 150 行定义.

3.45.2.2 TIMER_CMP0()

```
void TIMER_CMP0 (  
    TIMER_TypeDef * TIMERx,  
    uint32_t temp )
```

设置Timer通道0比较值

参数

<i>TIMERx</i>	Timer实例指针
<i>temp</i>	比较值

在文件 [lks32mc09x_timer.c](#) 第 110 行定义.

3.45.2.3 TIMER_CMP1()

```
void TIMER_CMP1 (  

```

```
TIMER_TypeDef * TIMEx,
uint32_t temp )
```

设置Timer通道1比较值

参数

<i>TIMEx</i>	Timer实例指针
<i>temp</i>	比较值

在文件 [lks32mc09x_timer.c](#) 第 120 行定义.

3.45.2.4 TIMER_Disable()

```
void TIMER_Disable (
    TIMER_TypeDef * TIMEx )
```

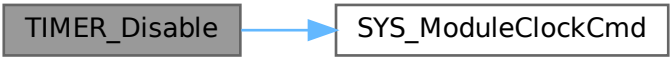
失能Timer

参数

<i>TIMEx</i>	Timer实例指针

在文件 [lks32mc09x_timer.c](#) 第 215 行定义.

函数调用图:



3.45.2.5 TIMER_Enable()

```
void TIMER_Enable (
    TIMER_TypeDef * TIMEx )
```

使能Timer

参数

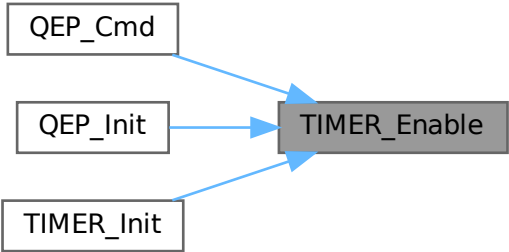
<i>TIMEx</i>	Timer实例指针

在文件 [lks32mc09x_timer.c](#) 第 191 行定义.

函数调用图:



这是这个函数的调用关系图:



3.45.2.6 **TIMER_GetFlag()**

```
uint16_t TIMER_GetFlag (
    TIMER_TypeDef * TIMERx,
    uint32_t TIMER_IF_x )
```

获取Timer状态标志

参数

<i>TIMERx</i>	Timer实例指针
<i>TIMER_↔IF_x</i>	状态标志位

返回
状态标志

在文件 [lks32mc09x_timer.c](#) 第 172 行定义.

3.45.2.7 TIMER_GetIRQFlag()

```
uint16_t TIMER_GetIRQFlag (
    TIMER_TypeDef * TIMEx,
    uint32_t TIMER_IF_x )
```

获取Timer中断标志状态

参数

<i>TIMERx</i>	Timer实例指针
<i>TIMER_IF_x</i>	中断标志位

返回
中断标志状态

在文件 [lks32mc09x_timer.c](#) 第 161 行定义.

3.45.2.8 TIMER_Init()

```
void TIMER_Init (
    TIMER_TypeDef * TIMEx,
    TIMER_TimerInitTypeDef * TIMER_InitStruct )
```

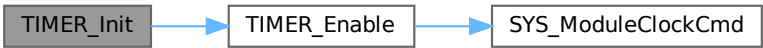
Timer初始化

参数

<i>TIMERx</i>	Timer实例指针
<i>TIMER_InitStruct</i>	Timer初始化结构体指针

在文件 [lks32mc09x_timer.c](#) 第 9 行定义.

函数调用图:



3.45.2.9 TIMER_SetOutputState()

```
void TIMER_SetOutputState (
    TIMER_TypeDef * TIMEx,
    uint8_t state )
```

设置输出状态

参数

<i>TIMERx</i>	Timer实例指针
<i>state</i>	输出状态，0关闭pwm输出，其他值，打开pwm输出

在文件 [lks32mc09x_timer.c](#) 第 240 行定义.

3.45.2.10 **TIMER_StructInit()**

```
void TIMER_StructInit (
    TIMER_TimerInitTypeDef * TIMER_InitStruct )
```

Timer结构体初始化

参数

<i>TIMER_InitStruct</i>	Timer初始化结构体指针

在文件 [lks32mc09x_timer.c](#) 第 79 行定义.

3.45.2.11 **TIMER_Timer_GetCMPT0()**

```
uint32_t TIMER_Timer_GetCMPT0 (
    TIMER_TypeDef * TIMERx )
```

获取Timer通道0比较值

参数

<i>TIMERx</i>	Timer实例指针

返回

比较值

在文件 [lks32mc09x_timer.c](#) 第 130 行定义.

3.45.2.12 **TIMER_Timer_GetCMPT1()**

```
uint32_t TIMER_Timer_GetCMPT1 (
    TIMER_TypeDef * TIMERx )
```

获取Timer通道1比较值

参数

<i>TIMERx</i>	Timer实例指针

返回
比较值

在文件 [lks32mc09x_timer.c](#) 第 140 行定义.

3.45.2.13 **TIMER_Timer_GetCount()**

```
uint32_t TIMER_Timer_GetCount (
    TIMER_TypeDef * TIMERx )
```

获取Timer当前计数值

参数

<i>TIMERx</i>	Timer实例指针

返回
当前计数值

在文件 [lks32mc09x_timer.c](#) 第 182 行定义.

3.45.2.14 **TIMER_TimerCmd()**

```
void TIMER_TimerCmd (
    TIMER_TypeDef * TIMERx,
    uint8_t state )
```

Timer使能/失能

参数

<i>TIMERx</i>	Timer实例指针
<i>state</i>	使能或失能状态

在文件 [lks32mc09x_timer.c](#) 第 93 行定义.

3.46 **lks32mc09x_timer.h**

[浏览该文件的文档.](#)

```

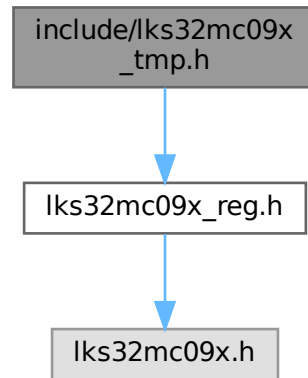
00001 #ifndef __LKS32MC09X_TIMER_H__
00002 #define __LKS32MC09X_TIMER_H__
00003
00004 #include "lks32mc09x_reg.h"
00005
00009 typedef struct
00010 {
00011     uint8_t EN;
00012     uint32_t CAP1_CLR_EN;
00013     uint32_t CAP0_CLR_EN;
00014     uint32_t ONE_TRIG;
00015     uint32_t CENTER;
00016     uint32_t DIR;
00017     uint32_t ClockDiv;
00018     uint32_t ETON;
00019     uint32_t GATE_EN;
00020     uint32_t RL_EN;
00021     uint32_t XCLK_EN;
00022     uint32_t SRC1;
00023     uint32_t CH1Output;
00024     uint32_t CH1_WorkMode;
00025     uint32_t CH1_FE_CAP_EN;
00026     uint32_t CH1_RE_CAP_EN;
00027     uint32_t SRC0;
00028     uint32_t CH0Output;
00029     uint32_t CH0_WorkMode;
00030     uint32_t CH0_FE_CAP_EN;
00031     uint32_t CH0_RE_CAP_EN;
00032     uint32_t TH;
00033     uint32_t CNT;
00034     uint32_t CHN0;
00035     uint32_t CHN1;
00036     uint32_t EVT;
00037     uint32_t FLT;
00038     uint32_t IE;
00039     // 注意: 以下配置项仅timer0和timer1有效
00040     uint32_t CMP0_TRIGGER_MODE;
00041     uint32_t CMP1_TRIGGER_MODE;
00042     uint32_t SHADOW;
00043     uint32_t CH1_DEFAULT;
00044     uint32_t CH0_DEFAULT;
00045     uint32_t HALT_PRT;
00046     uint32_t FAIL_SEL;
00047     uint32_t FAIL_POL;
00048     uint32_t FAIL_EN;
00049     uint32_t MOE;
00050 } TIMER_TimerInitTypeDef;
00051
00052 #define TIMER_IF_FAIL          BIT3
00053 #define TIMER_IF_ZC            BIT2
00054 #define TIMER_IF_CH1          BIT1
00055 #define TIMER_IF_CH0          BIT0
00056
00057 #define TIMER_CH_MODE_CMP      0 // 比较模式
00058 #define TIMER_CH_MODE_CAP      1 // 捕获模式
00059 #define TIMER_CH_MODE_Comparison 0 // 比较模式
00060 #define TIMER_CH_MODE_Capture 1 // 捕获模式
00061 #define TIMER_StatisticsStart(TIMERx) (TIMERx->CNT = 0)
00062 #define TIMER_StatisticsStop(TIMERx) (TIMERx->CNT)
00063 void TIMER_Init(TIMER_TypeDef *TIMERx, TIMER_TimerInitTypeDef *TIMER_InitStruct);
00064
00065 void TIMER_StructInit(TIMER_TimerInitTypeDef *TIMER_InitStruct);
00066
00067 void TIMER_TimerCmd(TIMER_TypeDef *TIMERx, uint8_t state);
00068
00069 void TIMER_CMP0(TIMER_TypeDef *TIMERx, uint32_t temp);
00070
00071 void TIMER_CMP1(TIMER_TypeDef *TIMERx, uint32_t temp);
00072
00073 uint32_t TIMER_Timer_GetCMPT0(TIMER_TypeDef *TIMERx);
00074
00075 uint32_t TIMER_Timer_GetCMPT1(TIMER_TypeDef *TIMERx);
00076
00077 void TIMER_ClearIRQFlag(TIMER_TypeDef *TIMERx, uint32_t tempFlag);
00078
00079 uint16_t TIMER_GetIRQFlag(TIMER_TypeDef *TIMERx, uint32_t TIMER_IF_x);
00080
00081 uint16_t TIMER_GetFlag(TIMER_TypeDef *TIMERx, uint32_t TIMER_IF_x);
00082
00083 uint32_t TIMER_Timer_GetCount(TIMER_TypeDef *TIMERx);
00084
00085 void TIMER_Enable(TIMER_TypeDef *TIMERx);
00086
00087 void TIMER_Disable(TIMER_TypeDef *TIMERx);
00088
00089 void TIMER_SetOutputState(TIMER_TypeDef *TIMERx, uint8_t state);
00090 #endif

```

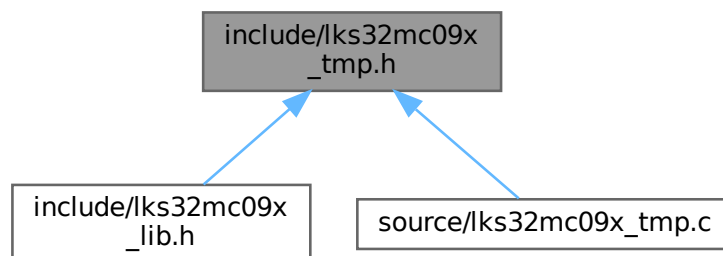
3.47 include/lks32mc09x_tmp.h 文件参考

```
#include "lks32mc09x_reg.h"
```

lks32mc09x_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:



函数

- void **TMP_Init** (void)
初始化温度传感器(读取温度传感器校准参数&使能温度传感器)
- void **TMP_Enable** (void)
使能温度传感器
- void **TMP_Disable** (void)
禁用温度传感器
- void **TMP_Reset** (void)
复位温度传感器(禁用温度传感器)
- int16_t **TMP_GetCurrentTemperature** (int16_t ADC_value)
获取当前温度

3.47.1 函数说明

3.47.1.1 TMP_Disable()

```
void TMP_Disable (
    void )
```

禁用温度传感器

在文件 [lks32mc09x.tmp.c](#) 第 29 行定义.

这是这个函数的调用关系图:



3.47.1.2 TMP_Enable()

```
void TMP_Enable (
    void )
```

使能温度传感器

在文件 [lks32mc09x.tmp.c](#) 第 19 行定义.

这是这个函数的调用关系图:



3.47.1.3 TMP_GetCurrentTemperature()

```
int16_t TMP_GetCurrentTemperature (
    int16_t ADC_value )
```

获取当前温度

参数

<i>ADC_value</i>	ADC采样值

返回

当前温度值，单位0.1摄氏度

3.47.1.4 TMP_Init()

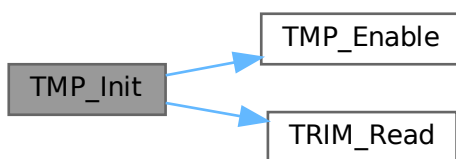
```
void TMP_Init (
    void )
```

初始化温度传感器(读取温度传感器校准参数&使能温度传感器)

初始化温度传感器(读取温度传感器校准参数&使能温度传感器)

在文件 [lks32mc09x.tmp.c](#) 第 9 行定义.

函数调用图:



3.47.1.5 TMP_Reset()

```
void TMP_Reset (
    void )
```

复位温度传感器(禁用温度传感器)

复位温度传感器(禁用温度传感器)

在文件 [lks32mc09x.tmp.c](#) 第 39 行定义.

函数调用图:



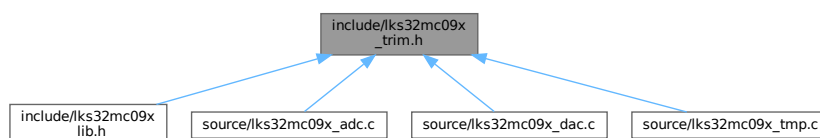
3.48 lks32mc09x_tmp.h

[浏览该文件的文档.](#)

```
00001 #ifndef __LKS32MC09X_TMP_H__
00002 #define __LKS32MC09X_TMP_H__
00003
00004 #include "lks32mc09x_reg.h"
00005
00009 void TMP_Init(void);
00010
00014 void TMP_Enable(void);
00015
00019 void TMP_Disable(void);
00020
00024 void TMP_Reset(void);
00025
00031 int16_t TMP_GetCurrentTemperature(int16_t ADC_value);
00032
00033 #endif
```

3.49 include/lks32mc09x_trim.h 文件参考

此图展示该文件直接或间接的被哪些文件引用了:



宏定义

- #define NVR_ADDR_AFE_REG8 0x00001400
- #define NVR_ADDR_AFE_REGA 0x00001404
- #define NVR_ADDR_AFE_REGB 0x00001408
- #define NVR_ADDR_AFE_REGC 0x0000140C
- #define NVR_ADDR_AFE_REG9 0x00001410
- #define NVR_ADDR_ADC_0_DC0 0x00001420
- #define NVR_ADDR_ADC_0_AMC0 0x00001424
- #define NVR_ADDR_ADC_0_DC1 0x00001428
- #define NVR_ADDR_ADC_0_AMC1 0x0000142C
- #define NVR_ADDR_ADC_1_DC0 0x00001430
- #define NVR_ADDR_ADC_1_AMC0 0x00001434
- #define NVR_ADDR_ADC_1_DC1 0x00001438
- #define NVR_ADDR_ADC_1_AMC1 0x0000143C
- #define NVR_ADDR_DAC_0_AMC_4.8V 0x00001450
- #define NVR_ADDR_DAC_0_DC_4.8V 0x00001454
- #define NVR_ADDR_DAC_0_AMC_1.20V 0x00001458
- #define NVR_ADDR_DAC_0_DC_1.20V 0x0000145C
- #define NVR_ADDR_DAC_1_AMC_3.0V 0x00001460
- #define NVR_ADDR_DAC_1_DC_3.0V 0x00001464
- #define NVR_ADDR_DAC_1_AMC_1.20V 0x00001468
- #define NVR_ADDR_DAC_1_DC_1.20V 0x0000146C
- #define NVR_ADDR_OPA_0 0x000014C0

- #define NVR_ADDR_OPA_1 0x000014C4
- #define NVR_ADDR_OPA_2 0x000014C8
- #define NVR_ADDR_OPA_3 0x000014CC
- #define NVR_ADDR_OPA_0_660K_20K_GAIN 0x00001480
- #define NVR_ADDR_OPA_0_640K_40K_GAIN 0x00001484
- #define NVR_ADDR_OPA_0_320K_40K_GAIN 0x00001488
- #define NVR_ADDR_OPA_0_160K_40K_GAIN 0x0000148C
- #define NVR_ADDR_OPA_1_660K_20K_GAIN 0x00001490
- #define NVR_ADDR_OPA_1_640K_40K_GAIN 0x00001494
- #define NVR_ADDR_OPA_1_320K_40K_GAIN 0x00001498
- #define NVR_ADDR_OPA_1_160K_40K_GAIN 0x0000149C
- #define NVR_ADDR_OPA_2_660K_20K_GAIN 0x000014A0
- #define NVR_ADDR_OPA_2_640K_40K_GAIN 0x000014A4
- #define NVR_ADDR_OPA_2_320K_40K_GAIN 0x000014A8
- #define NVR_ADDR_OPA_2_160K_40K_GAIN 0x000014AC
- #define NVR_ADDR_OPA_3_660K_20K_GAIN 0x000014B0
- #define NVR_ADDR_OPA_3_640K_40K_GAIN 0x000014B4
- #define NVR_ADDR_OPA_3_320K_40K_GAIN 0x000014B8
- #define NVR_ADDR_OPA_3_160K_40K_GAIN 0x000014BC
- #define NVR_ADDR_OPA_0_660K_20K_RES 0x00001500
- #define NVR_ADDR_OPA_0_640K_40K_RES 0x00001504
- #define NVR_ADDR_OPA_0_320K_40K_RES 0x00001508
- #define NVR_ADDR_OPA_0_160K_40K_RES 0x0000150C
- #define NVR_ADDR_OPA_1_660K_20K_RES 0x00001510
- #define NVR_ADDR_OPA_1_640K_40K_RES 0x00001514
- #define NVR_ADDR_OPA_1_320K_40K_RES 0x00001518
- #define NVR_ADDR_OPA_1_160K_40K_RES 0x0000151C
- #define NVR_ADDR_OPA_2_660K_20K_RES 0x00001520
- #define NVR_ADDR_OPA_2_640K_40K_RES 0x00001524
- #define NVR_ADDR_OPA_2_320K_40K_RES 0x00001528
- #define NVR_ADDR_OPA_2_160K_40K_RES 0x0000152C
- #define NVR_ADDR_OPA_3_660K_20K_RES 0x00001530
- #define NVR_ADDR_OPA_3_640K_40K_RES 0x00001534
- #define NVR_ADDR_OPA_3_320K_40K_RES 0x00001538
- #define NVR_ADDR_OPA_3_160K_40K_RES 0x0000153C
- #define NVR_ADDR_OPA_0_OFFSET_H 0x00001560
- #define NVR_ADDR_OPA_0_OFFSET_L 0x00001564
- #define NVR_ADDR_OPA_1_OFFSET_H 0x00001568
- #define NVR_ADDR_OPA_1_OFFSET_L 0x0000156C
- #define NVR_ADDR_OPA_2_OFFSET_H 0x00001570
- #define NVR_ADDR_OPA_2_OFFSET_L 0x00001574
- #define NVR_ADDR_OPA_3_OFFSET_H 0x00001578
- #define NVR_ADDR_OPA_3_OFFSET_L 0x0000157C
- #define NVR_ADDR_TMP_K 0x000014D0
- #define NVR_ADDR_TMP_B 0x000014D4
- #define NVR_ADDR_AMTE_TMP 0x000014D8
- #define NVR_ADDR_AMTE_CNT 0x000014E0
- #define NVR_ADDR_AMTE_VERSION 0x000014E4
- #define NVR_ADDR_CHIP_MODEL 0x000014E8
- #define NVR_ADDR_CHIP_BOOT_VERSION 0x000014EC
- #define NVR_ADDR_CHIP_FLASH_SIZE 0x000014F0
- #define NVR_ADDR_CHIP_ID0 0x1e84
- #define NVR_ADDR_CHIP_ID1 0x1e88
- #define NVR_ADDR_CHIP_ID2 0x1e8c
- #define NVR_ADDR_CHIP_ID3 0x1e90
- #define NVR_ADDR_CHIP_ID4 0x1e94

函数

- uint32_t [TRIM_Read](#) (uint32_t adr)

3.49.1 宏定义说明

3.49.1.1 NVR_ADDR_ADC_0_AMC0

```
#define NVR_ADDR_ADC_0_AMC0 0x00001424
```

在文件 [lks32mc09x.trim.h](#) 第 13 行定义.

3.49.1.2 NVR_ADDR_ADC_0_AMC1

```
#define NVR_ADDR_ADC_0_AMC1 0x0000142C
```

在文件 [lks32mc09x.trim.h](#) 第 15 行定义.

3.49.1.3 NVR_ADDR_ADC_0_DC0

```
#define NVR_ADDR_ADC_0_DC0 0x00001420
```

在文件 [lks32mc09x.trim.h](#) 第 12 行定义.

3.49.1.4 NVR_ADDR_ADC_0_DC1

```
#define NVR_ADDR_ADC_0_DC1 0x00001428
```

在文件 [lks32mc09x.trim.h](#) 第 14 行定义.

3.49.1.5 NVR_ADDR_ADC_1_AMC0

```
#define NVR_ADDR_ADC_1_AMC0 0x00001434
```

在文件 [lks32mc09x.trim.h](#) 第 17 行定义.

3.49.1.6 NVR_ADDR_ADC_1_AMC1

```
#define NVR_ADDR_ADC_1_AMC1 0x0000143C
```

在文件 [lks32mc09x.trim.h](#) 第 19 行定义.

3.49.1.7 NVR_ADDR_ADC_1_DC0

```
#define NVR_ADDR_ADC_1_DC0 0x00001430
```

在文件 [lks32mc09x.trim.h](#) 第 16 行定义.

3.49.1.8 NVR_ADDR_ADC_1_DC1

```
#define NVR_ADDR_ADC_1_DC1 0x00001438
```

在文件 [lks32mc09x.trim.h](#) 第 18 行定义.

3.49.1.9 NVR_ADDR_AFE_REG8

```
#define NVR_ADDR_AFE_REG8 0x00001400
```

在文件 [lks32mc09x.trim.h](#) 第 6 行定义.

3.49.1.10 NVR_ADDR_AFE_REG9

```
#define NVR_ADDR_AFE_REG9 0x00001410
```

在文件 [lks32mc09x.trim.h](#) 第 10 行定义.

3.49.1.11 NVR_ADDR_AFE_REGA

```
#define NVR_ADDR_AFE_REGA 0x00001404
```

在文件 [lks32mc09x.trim.h](#) 第 7 行定义.

3.49.1.12 NVR_ADDR_AFE_REGB

```
#define NVR_ADDR_AFE_REGB 0x00001408
```

在文件 [lks32mc09x.trim.h](#) 第 8 行定义.

3.49.1.13 NVR_ADDR_AFE_REGC

```
#define NVR_ADDR_AFE_REGC 0x0000140C
```

在文件 [lks32mc09x.trim.h](#) 第 9 行定义.

3.49.1.14 NVR_ADDR_AMTE_CNT

```
#define NVR_ADDR_AMTE_CNT 0x000014E0
```

在文件 [lks32mc09x.trim.h](#) 第 79 行定义.

3.49.1.15 NVR_ADDR_AMTE_TMP

```
#define NVR_ADDR_AMTE_TMP 0x000014D8
```

在文件 [lks32mc09x.trim.h](#) 第 78 行定义.

3.49.1.16 NVR_ADDR_AMTE_VERSION

```
#define NVR_ADDR_AMTE_VERSION 0x000014E4
```

在文件 [lks32mc09x.trim.h](#) 第 80 行定义.

3.49.1.17 NVR_ADDR_CHIP_BOOT_VERSION

```
#define NVR_ADDR_CHIP_BOOT_VERSION 0x000014EC
```

在文件 [lks32mc09x.trim.h](#) 第 83 行定义.

3.49.1.18 NVR_ADDR_CHIP_FLASH_SIZE

```
#define NVR_ADDR_CHIP_FLASH_SIZE 0x000014F0
```

在文件 [lks32mc09x.trim.h](#) 第 84 行定义.

3.49.1.19 NVR_ADDR_CHIP_ID0

```
#define NVR_ADDR_CHIP_ID0 0x1e84
```

在文件 [lks32mc09x.trim.h](#) 第 86 行定义.

3.49.1.20 NVR_ADDR_CHIP_ID1

```
#define NVR_ADDR_CHIP_ID1 0x1e88
```

在文件 [lks32mc09x.trim.h](#) 第 87 行定义.

3.49.1.21 NVR_ADDR_CHIP_ID2

```
#define NVR_ADDR_CHIP_ID2 0x1e8c
```

在文件 [lks32mc09x.trim.h](#) 第 88 行定义.

3.49.1.22 NVR_ADDR_CHIP_ID3

```
#define NVR_ADDR_CHIP_ID3 0x1e90
```

在文件 [lks32mc09x.trim.h](#) 第 89 行定义.

3.49.1.23 NVR_ADDR_CHIP_ID4

```
#define NVR_ADDR_CHIP_ID4 0x1e94
```

在文件 [lks32mc09x.trim.h](#) 第 90 行定义.

3.49.1.24 NVR_ADDR_CHIP_MODEL

```
#define NVR_ADDR_CHIP_MODEL 0x000014E8
```

在文件 [lks32mc09x.trim.h](#) 第 82 行定义.

3.49.1.25 NVR_ADDR_DAC_0_AMC_1_20V

```
#define NVR_ADDR_DAC_0_AMC_1_20V 0x00001458
```

在文件 [lks32mc09x.trim.h](#) 第 23 行定义.

3.49.1.26 NVR_ADDR_DAC_0_AMC_4_8V

```
#define NVR_ADDR_DAC_0_AMC_4_8V 0x00001450
```

在文件 [lks32mc09x.trim.h](#) 第 21 行定义.

3.49.1.27 NVR_ADDR_DAC_0_DC_1_20V

```
#define NVR_ADDR_DAC_0_DC_1_20V 0x0000145C
```

在文件 [lks32mc09x.trim.h](#) 第 24 行定义.

3.49.1.28 NVR_ADDR_DAC_0_DC_4_8V

```
#define NVR_ADDR_DAC_0_DC_4_8V 0x00001454
```

在文件 [lks32mc09x.trim.h](#) 第 22 行定义.

3.49.1.29 NVR_ADDR_DAC_1_AMC_1_20V

```
#define NVR_ADDR_DAC_1_AMC_1_20V 0x00001468
```

在文件 [lks32mc09x.trim.h](#) 第 27 行定义.

3.49.1.30 NVR_ADDR_DAC_1_AMC_3_0V

```
#define NVR_ADDR_DAC_1_AMC_3_0V 0x00001460
```

在文件 [lks32mc09x.trim.h](#) 第 25 行定义.

3.49.1.31 NVR_ADDR_DAC_1_DC_1_20V

```
#define NVR_ADDR_DAC_1_DC_1_20V 0x0000146C
```

在文件 [lks32mc09x.trim.h](#) 第 28 行定义.

3.49.1.32 NVR_ADDR_DAC_1_DC_3_0V

```
#define NVR_ADDR_DAC_1_DC_3_0V 0x00001464
```

在文件 [lks32mc09x.trim.h](#) 第 26 行定义.

3.49.1.33 NVR_ADDR_OPA_0

```
#define NVR_ADDR_OPA_0 0x000014C0
```

在文件 [lks32mc09x.trim.h](#) 第 30 行定义.

3.49.1.34 NVR_ADDR_OPA_0_160K_40K_GAIN

```
#define NVR_ADDR_OPA_0_160K_40K_GAIN 0x0000148C
```

在文件 [lks32mc09x.trim.h](#) 第 37 行定义.

3.49.1.35 NVR_ADDR_OPA_0_160K_40K_RES

```
#define NVR_ADDR_OPA_0_160K_40K_RES 0x0000150C
```

在文件 [lks32mc09x.trim.h](#) 第 53 行定义.

3.49.1.36 NVR_ADDR_OPA_0_320K_40K_GAIN

```
#define NVR_ADDR_OPA_0_320K_40K_GAIN 0x00001488
```

在文件 [lks32mc09x.trim.h](#) 第 36 行定义.

3.49.1.37 NVR_ADDR_OPA_0_320K_40K_RES

```
#define NVR_ADDR_OPA_0_320K_40K_RES 0x00001508
```

在文件 [lks32mc09x.trim.h](#) 第 52 行定义.

3.49.1.38 NVR_ADDR_OPA_0_640K_40K_GAIN

```
#define NVR_ADDR_OPA_0_640K_40K_GAIN 0x00001484
```

在文件 [lks32mc09x.trim.h](#) 第 35 行定义.

3.49.1.39 NVR_ADDR_OPA_0_640K_40K_RES

```
#define NVR_ADDR_OPA_0_640K_40K_RES 0x00001504
```

在文件 [lks32mc09x.trim.h](#) 第 51 行定义.

3.49.1.40 NVR_ADDR_OPA_0_660K_20K_GAIN

```
#define NVR_ADDR_OPA_0_660K_20K_GAIN 0x00001480
```

在文件 [lks32mc09x.trim.h](#) 第 34 行定义.

3.49.1.41 NVR_ADDR_OPA_0_660K_20K_RES

```
#define NVR_ADDR_OPA_0_660K_20K_RES 0x00001500
```

在文件 [lks32mc09x.trim.h](#) 第 50 行定义.

3.49.1.42 NVR_ADDR_OPA_0_OFFSET_H

```
#define NVR_ADDR_OPA_0_OFFSET_H 0x00001560
```

在文件 [lks32mc09x.trim.h](#) 第 66 行定义.

3.49.1.43 NVR_ADDR_OPA_0_OFFSET_L

```
#define NVR_ADDR_OPA_0_OFFSET_L 0x00001564
```

在文件 [lks32mc09x.trim.h](#) 第 67 行定义.

3.49.1.44 NVR_ADDR_OPA_1

```
#define NVR_ADDR_OPA_1 0x000014C4
```

在文件 [lks32mc09x.trim.h](#) 第 31 行定义.

3.49.1.45 NVR_ADDR_OPA_1_160K_40K_GAIN

```
#define NVR_ADDR_OPA_1_160K_40K_GAIN 0x0000149C
```

在文件 [lks32mc09x.trim.h](#) 第 41 行定义.

3.49.1.46 NVR_ADDR_OPA_1_160K_40K_RES

```
#define NVR_ADDR_OPA_1_160K_40K_RES 0x0000151C
```

在文件 [lks32mc09x.trim.h](#) 第 57 行定义.

3.49.1.47 NVR_ADDR_OPA_1_320K_40K_GAIN

```
#define NVR_ADDR_OPA_1_320K_40K_GAIN 0x00001498
```

在文件 [lks32mc09x.trim.h](#) 第 40 行定义.

3.49.1.48 NVR_ADDR_OPA_1_320K_40K_RES

```
#define NVR_ADDR_OPA_1_320K_40K_RES 0x00001518
```

在文件 [lks32mc09x.trim.h](#) 第 56 行定义.

3.49.1.49 NVR_ADDR_OPA_1_640K_40K_GAIN

```
#define NVR_ADDR_OPA_1_640K_40K_GAIN 0x00001494
```

在文件 [lks32mc09x.trim.h](#) 第 39 行定义.

3.49.1.50 NVR_ADDR_OPA_1_640K_40K_RES

```
#define NVR_ADDR_OPA_1_640K_40K_RES 0x00001514
```

在文件 [lks32mc09x.trim.h](#) 第 55 行定义.

3.49.1.51 NVR_ADDR_OPA_1_660K_20K_GAIN

```
#define NVR_ADDR_OPA_1_660K_20K_GAIN 0x00001490
```

在文件 [lks32mc09x.trim.h](#) 第 38 行定义.

3.49.1.52 NVR_ADDR_OPA_1_660K_20K_RES

```
#define NVR_ADDR_OPA_1_660K_20K_RES 0x00001510
```

在文件 [lks32mc09x.trim.h](#) 第 54 行定义.

3.49.1.53 NVR_ADDR_OPA_1_OFFSET_H

```
#define NVR_ADDR_OPA_1_OFFSET_H 0x00001568
```

在文件 [lks32mc09x.trim.h](#) 第 68 行定义.

3.49.1.54 NVR_ADDR_OPA_1_OFFSET_L

```
#define NVR_ADDR_OPA_1_OFFSET_L 0x0000156C
```

在文件 [lks32mc09x.trim.h](#) 第 69 行定义.

3.49.1.55 NVR_ADDR_OPA_2

```
#define NVR_ADDR_OPA_2 0x000014C8
```

在文件 [lks32mc09x.trim.h](#) 第 32 行定义.

3.49.1.56 NVR_ADDR_OPA_2_160K_40K_GAIN

```
#define NVR_ADDR_OPA_2_160K_40K_GAIN 0x000014AC
```

在文件 [lks32mc09x.trim.h](#) 第 45 行定义.

3.49.1.57 NVR_ADDR_OPA_2_160K_40K_RES

```
#define NVR_ADDR_OPA_2_160K_40K_RES 0x0000152C
```

在文件 [lks32mc09x.trim.h](#) 第 61 行定义.

3.49.1.58 NVR_ADDR_OPA_2_320K_40K_GAIN

```
#define NVR_ADDR_OPA_2_320K_40K_GAIN 0x000014A8
```

在文件 [lks32mc09x.trim.h](#) 第 44 行定义.

3.49.1.59 NVR_ADDR_OPA_2_320K_40K_RES

```
#define NVR_ADDR_OPA_2_320K_40K_RES 0x00001528
```

在文件 [lks32mc09x.trim.h](#) 第 60 行定义.

3.49.1.60 NVR_ADDR_OPA_2_640K_40K_GAIN

```
#define NVR_ADDR_OPA_2_640K_40K_GAIN 0x000014A4
```

在文件 [lks32mc09x.trim.h](#) 第 43 行定义.

3.49.1.61 NVR_ADDR_OPA_2_640K_40K_RES

```
#define NVR_ADDR_OPA_2_640K_40K_RES 0x00001524
```

在文件 [lks32mc09x.trim.h](#) 第 59 行定义.

3.49.1.62 NVR_ADDR_OPA_2_660K_20K_GAIN

```
#define NVR_ADDR_OPA_2_660K_20K_GAIN 0x000014A0
```

在文件 [lks32mc09x.trim.h](#) 第 42 行定义.

3.49.1.63 NVR_ADDR_OPA_2_660K_20K_RES

```
#define NVR_ADDR_OPA_2_660K_20K_RES 0x00001520
```

在文件 [lks32mc09x.trim.h](#) 第 58 行定义.

3.49.1.64 NVR_ADDR_OPA_2_OFFSET_H

```
#define NVR_ADDR_OPA_2_OFFSET_H 0x00001570
```

在文件 [lks32mc09x.trim.h](#) 第 70 行定义.

3.49.1.65 NVR_ADDR_OPA_2_OFFSET_L

```
#define NVR_ADDR_OPA_2_OFFSET_L 0x00001574
```

在文件 [lks32mc09x.trim.h](#) 第 71 行定义.

3.49.1.66 NVR_ADDR_OPA_3

```
#define NVR_ADDR_OPA_3 0x000014CC
```

在文件 [lks32mc09x.trim.h](#) 第 33 行定义.

3.49.1.67 NVR_ADDR_OPA_3_160K_40K_GAIN

```
#define NVR_ADDR_OPA_3_160K_40K_GAIN 0x000014BC
```

在文件 [lks32mc09x.trim.h](#) 第 49 行定义.

3.49.1.68 NVR_ADDR_OPA_3_160K_40K_RES

```
#define NVR_ADDR_OPA_3_160K_40K_RES 0x0000153C
```

在文件 [lks32mc09x.trim.h](#) 第 65 行定义.

3.49.1.69 NVR_ADDR_OPA_3_320K_40K_GAIN

```
#define NVR_ADDR_OPA_3_320K_40K_GAIN 0x000014B8
```

在文件 [lks32mc09x.trim.h](#) 第 48 行定义.

3.49.1.70 NVR_ADDR_OPA_3_320K_40K_RES

```
#define NVR_ADDR_OPA_3_320K_40K_RES 0x00001538
```

在文件 [lks32mc09x.trim.h](#) 第 64 行定义.

3.49.1.71 NVR_ADDR_OPA_3_640K_40K_GAIN

```
#define NVR_ADDR_OPA_3_640K_40K_GAIN 0x000014B4
```

在文件 [lks32mc09x.trim.h](#) 第 47 行定义.

3.49.1.72 NVR_ADDR_OPA_3_640K_40K_RES

```
#define NVR_ADDR_OPA_3_640K_40K_RES 0x00001534
```

在文件 [lks32mc09x.trim.h](#) 第 63 行定义.

3.49.1.73 NVR_ADDR_OPA_3_660K_20K_GAIN

```
#define NVR_ADDR_OPA_3_660K_20K_GAIN 0x000014B0
```

在文件 [lks32mc09x.trim.h](#) 第 46 行定义.

3.49.1.74 NVR_ADDR_OPA_3_660K_20K_RES

```
#define NVR_ADDR_OPA_3_660K_20K_RES 0x00001530
```

在文件 [lks32mc09x.trim.h](#) 第 62 行定义.

3.49.1.75 NVR_ADDR_OPA_3_OFFSET_H

```
#define NVR_ADDR_OPA_3_OFFSET_H 0x00001578
```

在文件 [lks32mc09x.trim.h](#) 第 72 行定义.

3.49.1.76 NVR_ADDR_OPA_3_OFFSET_L

```
#define NVR_ADDR_OPA_3_OFFSET_L 0x0000157C
```

在文件 [lks32mc09x.trim.h](#) 第 73 行定义.

3.49.1.77 NVR_ADDR_TMP_B

```
#define NVR_ADDR_TMP_B 0x000014D4
```

在文件 [lks32mc09x.trim.h](#) 第 76 行定义.

3.49.1.78 NVR_ADDR_TMP_K

```
#define NVR_ADDR_TMP_K 0x000014D0
```

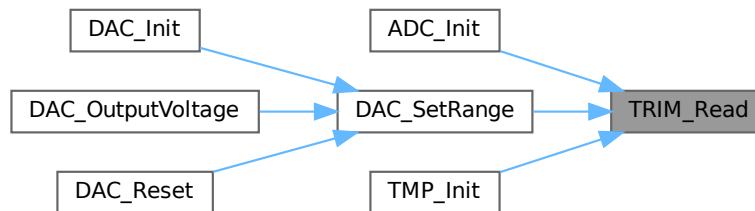
在文件 [lks32mc09x.trim.h](#) 第 75 行定义.

3.49.2 函数说明

3.49.2.1 TRIM_Read()

```
uint32_t TRIM_Read (
    uint32_t adr )
```

这是这个函数的调用关系图:



3.50 lks32mc09x_trim.h

[浏览该文件的文档.](#)

```

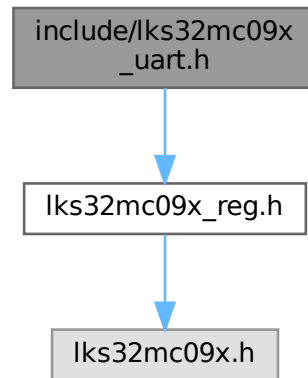
00001 #ifndef __LKS32MC09X_TRIM_H__
00002 #define __LKS32MC09X_TRIM_H__
00003
00004 uint32_t TRIM_Read(uint32_t adr);
00005 // AFE
00006 #define NVR_ADDR_AFE_REG8 0x00001400
00007 #define NVR_ADDR_AFE_REGA 0x00001404
00008 #define NVR_ADDR_AFE_REGB 0x00001408
00009 #define NVR_ADDR_AFE_REGC 0x0000140C
00010 #define NVR_ADDR_AFE_REG9 0x00001410
00011 // ADC
00012 #define NVR_ADDR_ADC_0_DC0 0x00001420
00013 #define NVR_ADDR_ADC_0_AMC0 0x00001424
00014 #define NVR_ADDR_ADC_0_DC1 0x00001428
00015 #define NVR_ADDR_ADC_0_AMC1 0x0000142C
00016 #define NVR_ADDR_ADC_1_DC0 0x00001430
00017 #define NVR_ADDR_ADC_1_AMC0 0x00001434
00018 #define NVR_ADDR_ADC_1_DC1 0x00001438
00019 #define NVR_ADDR_ADC_1_AMC1 0x0000143C
00020 // DAC
00021 #define NVR_ADDR_DAC_0_AMC_4.8V 0x00001450
00022 #define NVR_ADDR_DAC_0_DC_4.8V 0x00001454
00023 #define NVR_ADDR_DAC_0_AMC_1.20V 0x00001458
00024 #define NVR_ADDR_DAC_0_DC_1.20V 0x0000145C
00025 #define NVR_ADDR_DAC_1_AMC_3.0V 0x00001460
00026 #define NVR_ADDR_DAC_1_DC_3.0V 0x00001464
00027 #define NVR_ADDR_DAC_1_AMC_1.20V 0x00001468
00028 #define NVR_ADDR_DAC_1_DC_1.20V 0x0000146C
00029 // OPA
00030 #define NVR_ADDR_OPA_0 0x000014C0
00031 #define NVR_ADDR_OPA_1 0x000014C4
00032 #define NVR_ADDR_OPA_2 0x000014C8
00033 #define NVR_ADDR_OPA_3 0x000014CC
00034 #define NVR_ADDR_OPA_0_660K_20K_GAIN 0x00001480
00035 #define NVR_ADDR_OPA_0_640K_40K_GAIN 0x00001484
00036 #define NVR_ADDR_OPA_0_320K_40K_GAIN 0x00001488
00037 #define NVR_ADDR_OPA_0_160K_40K_GAIN 0x0000148C
00038 #define NVR_ADDR_OPA_1_660K_20K_GAIN 0x00001490
00039 #define NVR_ADDR_OPA_1_640K_40K_GAIN 0x00001494
00040 #define NVR_ADDR_OPA_1_320K_40K_GAIN 0x00001498
00041 #define NVR_ADDR_OPA_1_160K_40K_GAIN 0x0000149C
00042 #define NVR_ADDR_OPA_2_660K_20K_GAIN 0x000014A0
00043 #define NVR_ADDR_OPA_2_640K_40K_GAIN 0x000014A4
```

```
00044 #define NVR_ADDR_OPA_2_320K_40K_GAIN 0x000014A8
00045 #define NVR_ADDR_OPA_2_160K_40K_GAIN 0x000014AC
00046 #define NVR_ADDR_OPA_3_660K_20K_GAIN 0x000014B0
00047 #define NVR_ADDR_OPA_3_640K_40K_GAIN 0x000014B4
00048 #define NVR_ADDR_OPA_3_320K_40K_GAIN 0x000014B8
00049 #define NVR_ADDR_OPA_3_160K_40K_GAIN 0x000014BC
00050 #define NVR_ADDR_OPA_0_660K_20K_RES 0x00001500
00051 #define NVR_ADDR_OPA_0_640K_40K_RES 0x00001504
00052 #define NVR_ADDR_OPA_0_320K_40K_RES 0x00001508
00053 #define NVR_ADDR_OPA_0_160K_40K_RES 0x0000150C
00054 #define NVR_ADDR_OPA_1_660K_20K_RES 0x00001510
00055 #define NVR_ADDR_OPA_1_640K_40K_RES 0x00001514
00056 #define NVR_ADDR_OPA_1_320K_40K_RES 0x00001518
00057 #define NVR_ADDR_OPA_1_160K_40K_RES 0x0000151C
00058 #define NVR_ADDR_OPA_2_660K_20K_RES 0x00001520
00059 #define NVR_ADDR_OPA_2_640K_40K_RES 0x00001524
00060 #define NVR_ADDR_OPA_2_320K_40K_RES 0x00001528
00061 #define NVR_ADDR_OPA_2_160K_40K_RES 0x0000152C
00062 #define NVR_ADDR_OPA_3_660K_20K_RES 0x00001530
00063 #define NVR_ADDR_OPA_3_640K_40K_RES 0x00001534
00064 #define NVR_ADDR_OPA_3_320K_40K_RES 0x00001538
00065 #define NVR_ADDR_OPA_3_160K_40K_RES 0x0000153C
00066 #define NVR_ADDR_OPA_0_OFFSET_H 0x00001560
00067 #define NVR_ADDR_OPA_0_OFFSET_L 0x00001564
00068 #define NVR_ADDR_OPA_1_OFFSET_H 0x00001568
00069 #define NVR_ADDR_OPA_1_OFFSET_L 0x0000156C
00070 #define NVR_ADDR_OPA_2_OFFSET_H 0x00001570
00071 #define NVR_ADDR_OPA_2_OFFSET_L 0x00001574
00072 #define NVR_ADDR_OPA_3_OFFSET_H 0x00001578
00073 #define NVR_ADDR_OPA_3_OFFSET_L 0x0000157C
00074 // TMP
00075 #define NVR_ADDR_TMP_K 0x000014D0
00076 #define NVR_ADDR_TMP_B 0x000014D4
00077 // TMP
00078 #define NVR_ADDR_AMTE_TMP 0x000014D8
00079 #define NVR_ADDR_AMTE_CNT 0x000014E0
00080 #define NVR_ADDR_AMTE_VERSION 0x000014E4
00081 // CHIP
00082 #define NVR_ADDR_CHIP_MODEL 0x000014E8
00083 #define NVR_ADDR_CHIP_BOOT_VERSION 0x000014EC
00084 #define NVR_ADDR_CHIP_FLASH_SIZE 0x000014F0
00085
00086 #define NVR_ADDR_CHIP_ID0 0x1e84
00087 #define NVR_ADDR_CHIP_ID1 0x1e88
00088 #define NVR_ADDR_CHIP_ID2 0x1e8c
00089 #define NVR_ADDR_CHIP_ID3 0x1e90
00090 #define NVR_ADDR_CHIP_ID4 0x1e94
00091
00092 #endif
```

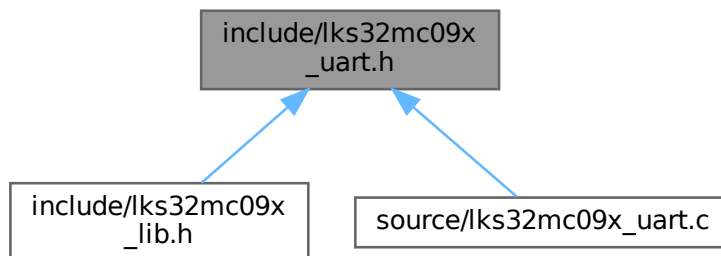
3.51 include/lks32mc09x_uart.h 文件参考

```
#include "lks32mc09x_reg.h"
```

lks32mc09x_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:



结构体

- struct [UART_InitTypeDef](#)
UART 初始化结构体定义

宏定义

- [#define UART_WORDLENGTH_8b 0](#)
数据长度定义
- [#define UART_WORDLENGTH_9b 1](#)

- 9位数据位
- `#define UART_STOPBITS_1b 0`
停止位长度定义
- `#define UART_STOPBITS_2b 1`
1位停止位
- `#define UART_FIRSTSEND_LSB 0`
发送数据大小端设置
- `#define UART_FIRSTSEND_MSB 1`
高端先发送
- `#define UART_IRQEna_SendOver BIT0`
中断使能定义
- `#define UART_IRQEna_RcvOver BIT1`
使能接收完成中断
- `#define UART_IRQEna_SendBuffEmpty BIT2`
使能发送缓冲区空中断
- `#define UART_IRQEna_StopError BIT3`
使能停止位错误中断
- `#define UART_IRQEna_CheckError BIT4`
使能校验错误中断
- `#define UART_IRQEna_TX_OV BIT5`
发送缓冲区溢出中断使能
- `#define UART_IRQEna_RX_OV BIT6`
接收缓冲区溢出中断使能
- `#define UART_IRQEna_LBD BIT7`
*LIN*同步间隔场检测中断使能
- `#define UART_IRQEna_IDLE BIT8`
空闲帧中断使能 在串口接收完成后再等待10个*baud*的时间后触发
- `#define UART_IRQEna_ABD BIT9`
波特率自适应使能中断
- `#define TX_DMA_RE BIT0`
DMA 请求使能定义
- `#define RX_DMA_RE BIT1`
接收完成*DMA*请求使能
- `#define TX_BUF_DMA_RE BIT2`
发送缓冲区空*DMA*请求使能
- `#define UART_IF_SendOver BIT0`
中断标志定义
- `#define UART_IF_RcvOver BIT1`
接收完成中断标志
- `#define UART_IF_SendBufEmpty BIT2`
发送缓冲区空中断标志
- `#define UART_IF_StopError BIT3`
停止位错误中断标志
- `#define UART_IF_CheckError BIT4`
校验错误中断标志
- `#define UART_IF_TX_OV BIT5`
发送缓冲区溢出中断标志
- `#define UART_IF_RX_OV BIT6`
接收缓冲区溢出中断标志
- `#define UART_IF_LBD BIT7`
LIN break character 检测中断标志

- `#define UART_IF_IDLE BIT8`
空闲帧中断标志
- `#define UART_IF_ABD BIT9`
波特率自适应使能中断标志

枚举

- `enum UART_ParityMode { UART_Parity_NO = 0x00 , UART_Parity_EVEN = 0x01 , UART_Parity_ODD = 0x02 }`
UART 奇偶校验配置枚举成员

函数

- `void UART_Init (UART_TypeDef *UARTx, UART_InitTypeDef *UART_InitStruct)`
函数声明
- `void UART_StructInit (UART_InitTypeDef *UART_InitStruct)`
初始化*UART*结构体为默认值
- `void UART_SendData (UART_TypeDef *UARTx, uint32_t n)`
发送数据
- `uint32_t UART_ReadData (UART_TypeDef *UARTx)`
读取接收到的数据
- `void UART_SendnData (UART_TypeDef *UARTx, char *sendData, uint32_t len)`
发送指定长度的数据
- `void UART_RecvnData (UART_TypeDef *UARTx, char *recvData, uint32_t maxlen)`
接收指定长度的数据
- `void UART_ClearIRQFlag (UART_TypeDef *UARTx, uint32_t tempFlag)`
清除指定的中断标志
- `uint32_t UART_GetIRQFlag (UART_TypeDef *UARTx, uint32_t tempFlag)`
获取指定的中断标志状态
- `uint8_t UART_SendAddr_485 (UART_TypeDef *UARTx, uint8_t n)`
在*485*模式下发送地址
- `uint8_t UART_SendData_485 (UART_TypeDef *UARTx, uint8_t n)`
在*485*模式下发送数据
- `uint32_t UART_SetBaud (UART_TypeDef *UARTx, uint32_t Baud)`
设置波特率
- `void UART_SetClkDiv (uint32_t div)`
设置*uart*时钟分频

3.51.1 宏定义说明

3.51.1.1 RX_DMA_RE

```
#define RX_DMA_RE BIT1
```

接收完成DMA请求使能

在文件 `lks32mc09x_uart.h` 第 62 行定义.

3.51.1.2 TX_BUF_DMA_RE

```
#define TX_BUF_DMA_RE BIT2
```

发送缓冲区空DMA请求使能

在文件 [lks32mc09x_uart.h](#) 第 63 行定义.

3.51.1.3 TX_DMA_RE

```
#define TX_DMA_RE BIT0
```

DMA 请求使能定义

发送完成DMA请求使能

在文件 [lks32mc09x_uart.h](#) 第 61 行定义.

3.51.1.4 UART_FIRSTSEND_LSB

```
#define UART_FIRSTSEND_LSB 0
```

发送数据大小端设置

低端先发送

在文件 [lks32mc09x_uart.h](#) 第 45 行定义.

3.51.1.5 UART_FIRSTSEND_MSB

```
#define UART_FIRSTSEND_MSB 1
```

高端先发送

在文件 [lks32mc09x_uart.h](#) 第 46 行定义.

3.51.1.6 UART_IF_ABD

```
#define UART_IF_ABD BIT9
```

波特率自适应使能中断标志

在文件 [lks32mc09x_uart.h](#) 第 75 行定义.

3.51.1.7 UART_IF_CheckError

```
#define UART_IF_CheckError BIT4
```

校验错误中断标志

在文件 [lks32mc09x_uart.h](#) 第 70 行定义.

3.51.1.8 UART_IF_IDLE

```
#define UART_IF_IDLE BIT8
```

空闲帧中断标志

在文件 [lks32mc09x_uart.h](#) 第 74 行定义.

3.51.1.9 UART_IF_LBD

```
#define UART_IF_LBD BIT7
```

LIN break character 检测中断标志

在文件 [lks32mc09x_uart.h](#) 第 73 行定义.

3.51.1.10 UART_IF_RcvOver

```
#define UART_IF_RcvOver BIT1
```

接收完成中断标志

在文件 [lks32mc09x_uart.h](#) 第 67 行定义.

3.51.1.11 UART_IF_RX_OV

```
#define UART_IF_RX_OV BIT6
```

接收缓冲区溢出中断标志

在文件 [lks32mc09x_uart.h](#) 第 72 行定义.

3.51.1.12 UART_IF_SendBufEmpty

```
#define UART_IF_SendBufEmpty BIT2
```

发送缓冲区空中断标志

在文件 [lks32mc09x_uart.h](#) 第 68 行定义.

3.51.1.13 UART_IF_SendOver

```
#define UART_IF_SendOver BIT0
```

中断标志定义

发送完成中断标志

在文件 [lks32mc09x_uart.h](#) 第 66 行定义.

3.51.1.14 UART_IF_StopError

```
#define UART_IF_StopError BIT3
```

停止位错误中断标志

在文件 [lks32mc09x_uart.h](#) 第 69 行定义.

3.51.1.15 UART_IF_TX_OV

```
#define UART_IF_TX_OV BIT5
```

发送缓冲区溢出中断标志

在文件 [lks32mc09x_uart.h](#) 第 71 行定义.

3.51.1.16 UART_IRQEna_ABD

```
#define UART_IRQEna_ABD BIT9
```

波特率自适应使能中断

在文件 [lks32mc09x_uart.h](#) 第 58 行定义.

3.51.1.17 UART_IRQEna_CheckError

```
#define UART_IRQEna_CheckError BIT4
```

使能校验错误中断

在文件 [lks32mc09x_uart.h](#) 第 53 行定义.

3.51.1.18 UART_IRQEna_IDLE

```
#define UART_IRQEna_IDLE BIT8
```

空闲帧中断使能 在串口接收完成后再等待10个baud的时间后触发

在文件 [lks32mc09x_uart.h](#) 第 57 行定义.

3.51.1.19 UART_IRQEna_LBD

```
#define UART_IRQEna_LBD BIT7
```

LIN同步间隔场检测中断使能

在文件 [lks32mc09x_uart.h](#) 第 56 行定义.

3.51.1.20 UART_IRQEna_RcvOver

```
#define UART_IRQEna_RcvOver BIT1
```

使能接收完成中断

在文件 [lks32mc09x_uart.h](#) 第 50 行定义.

3.51.1.21 UART_IRQEna_RX_OV

```
#define UART_IRQEna_RX_OV BIT6
```

接收缓冲区溢出中断使能

在文件 [lks32mc09x_uart.h](#) 第 55 行定义.

3.51.1.22 UART_IRQEna_SendBuffEmpty

```
#define UART_IRQEna_SendBuffEmpty BIT2
```

使能发送缓冲区空中断

在文件 [lks32mc09x_uart.h](#) 第 51 行定义.

3.51.1.23 UART_IRQEna_SendOver

```
#define UART_IRQEna_SendOver BIT0
```

中断使能定义

使能发送完成中断

在文件 [lks32mc09x_uart.h](#) 第 49 行定义.

3.51.1.24 UART_IRQEna_StopError

```
#define UART_IRQEna_StopError BIT3
```

使能停止位错误中断

在文件 [lks32mc09x_uart.h](#) 第 52 行定义.

3.51.1.25 UART_IRQEna_TX_OV

```
#define UART_IRQEna_TX_OV BIT5
```

发送缓冲区溢出中断使能

在文件 [lks32mc09x_uart.h](#) 第 54 行定义.

3.51.1.26 UART_STOPBITS_1b

```
#define UART_STOPBITS_1b 0
```

停止位长度定义

无停止位

在文件 [lks32mc09x_uart.h](#) 第 41 行定义.

3.51.1.27 UART_STOPBITS_2b

```
#define UART_STOPBITS_2b 1
```

1位停止位

在文件 [lks32mc09x_uart.h](#) 第 42 行定义.

3.51.1.28 UART_WORDLENGTH_8b

```
#define UART_WORDLENGTH_8b 0
```

数据长度定义

8位数据位

在文件 [lks32mc09x_uart.h](#) 第 37 行定义.

3.51.1.29 UART_WORDLENGTH_9b

```
#define UART_WORDLENGTH_9b 1
```

9位数据位

在文件 [lks32mc09x_uart.h](#) 第 38 行定义.

3.51.2 枚举类型说明

3.51.2.1 UART_ParityMode

```
enum UART_ParityMode
```

UART 奇偶校验配置枚举成员

枚举值

UART_Parity_NO	无校验
UART_Parity_EVEN	偶校验
UART_Parity_ODD	奇校验

在文件 `lks32mc09x_uart.h` 第 7 行定义.

3.51.3 函数说明

3.51.3.1 UART_ClearIRQFlag()

```
void UART_ClearIRQFlag (
    UART_TypeDef * UARTx,
    uint32_t tempFlag )
```

清除指定的中断标志

参数

<i>UARTx</i>	UART模块指针
<i>tempFlag</i>	要清除的中断标志

在文件 `lks32mc09x_uart.c` 第 134 行定义.

3.51.3.2 UART_GetIRQFlag()

```
uint32_t UART_GetIRQFlag (
    UART_TypeDef * UARTx,
    uint32_t tempFlag )
```

获取指定的中断标志状态

参数

<i>UARTx</i>	UART模块指针
<i>tempFlag</i>	要获取的中断标志

返回
中断标志状态

在文件 `lks32mc09x_uart.c` 第 145 行定义.

3.51.3.3 UART_Init()

```
void UART_Init (
    UART_TypeDef * UARTx,
    UART_InitTypeDef * UART_InitStruct )
```

函数声明

初始化UART模块

参数

UARTx	UART模块指针
UART_InitStruct	初始化结构体指针

函数声明

参数

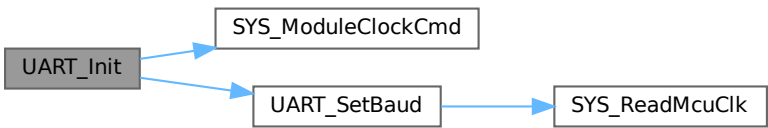
UARTx	UART模块指针
UART_InitStruct	初始化结构体指针

< LIN同步段检测长度 0:10bits 1:11bits

< LIN波特率自适应使能

在文件 [lks32mc09x_uart.c](#) 第 9 行定义.

函数调用图:



3.51.3.4 UART_ReadData()

```
uint32_t UART_ReadData (
    UART_TypeDef * UARTx )
```

读取接收到的数据

参数

UARTx	UART模块指针

返回

接收到的数据

在文件 [lks32mc09x_uart.c](#) 第 92 行定义.

这是这个函数的调用关系图:



3.51.3.5 UART_RecvnData()

```
void UART_RecvnData (
    UART_TypeDef * UARTx,
    char * recvData,
    uint32_t maxlen )
```

接收指定长度的数据

参数

UARTx	UART模块指针
recvData	接收数据缓冲区指针
maxlen	最大接收长度

在文件 [lks32mc09x_uart.c](#) 第 119 行定义.

函数调用图:



3.51.3.6 UART_SendAddr_485()

```
uint8_t UART_SendAddr_485 (
    UART_TypeDef * UARTx,
    uint8_t n )
```

在485模式下发送地址

参数

<i>UARTx</i>	UART模块指针
<i>n</i>	要发送的地址

返回
发送状态

参数

<i>UARTx</i>	UART模块指针
<i>n</i>	要发送的地址

返回
发送状态（简单实现，总是返回成功）

在文件 [lks32mc09x_uart.c](#) 第 156 行定义.

函数调用图:



3.51.3.7 UART_SendData()

```
void UART_SendData (
    UART_TypeDef * UARTx,
    uint32_t n )
```

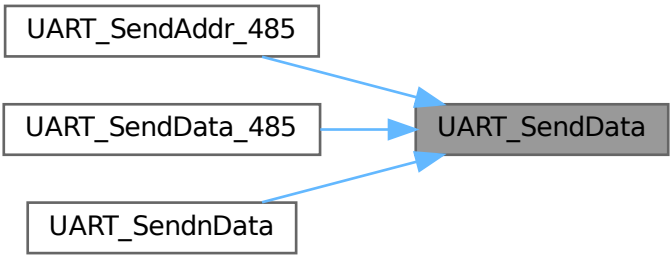
发送数据

参数

<i>UARTx</i>	UART模块指针
<i>n</i>	要发送的数据

在文件 [lks32mc09x_uart.c](#) 第 78 行定义.

这是这个函数的调用关系图:



3.51.3.8 UART_SendData_485()

```
uint8_t UART_SendData_485 (
    UART_TypeDef * UARTx,
    uint8_t n )
```

在485模式下发送数据

参数

<i>UARTx</i>	UART模块指针
<i>n</i>	要发送的数据

返回

发送状态

参数

<i>UARTx</i>	UART模块指针
<i>n</i>	要发送的数据

返回

发送状态（简单实现，总是返回成功）

在文件 [lks32mc09x_uart.c](#) 第 168 行定义.

函数调用图:



3.51.3.9 UART_SendnData()

```
void UART_SendnData (
    UART_TypeDef * UARTx,
    char * sendData,
    uint32_t len )
```

发送指定长度的数据

参数

<i>UARTx</i>	UART模块指针
<i>sendData</i>	要发送的数据指针
<i>len</i>	数据长度

在文件 [lks32mc09x_uart.c](#) 第 103 行定义.

函数调用图:



3.51.3.10 UART_SetBaud()

```
uint32_t UART_SetBaud (
    UART_TypeDef * UARTx,
    uint32_t Baud )
```

设置波特率

参数

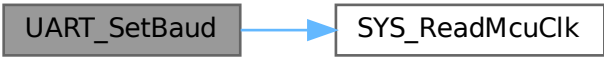
<i>UARTx</i>	UART模块
<i>Baud</i>	波特率范围

返回

实际波特率（不考虑时钟误差）

在文件 [lks32mc09x_uart.c](#) 第 180 行定义.

函数调用图:



这是这个函数的调用关系图:



3.51.3.11 UART.SetClkDiv()

```
void UART_SetClkDiv (
    uint32_t div )
```

设置uart时钟分频

参数

<i>div</i>	分频系数，0对应不分频，1对应2分频

在文件 [lks32mc09x_uart.c](#) 第 209 行定义.

函数调用图:



3.51.3.12 UART_StructInit()

```
void UART_StructInit (
    UART_InitTypeDef * UART_InitStruct )
```

初始化UART结构体为默认值

参数

UART_InitStruct	初始化结构体指针

在文件 [lks32mc09x_uart.c](#) 第 56 行定义.

3.52 lks32mc09x_uart.h

浏览该文件的文档.

```
00001 #ifndef __LKS32MC09X_UART_H__
00002 #define __LKS32MC09X_UART_H__
00003
00004 #include "lks32mc09x_reg.h"
00005
00007 typedef enum
00008 {
00009     UART_Parity_NO    = 0x00,
00010     UART_Parity_EVEN = 0x01,
00011     UART_Parity_ODD  = 0x02
00012 } UART_ParityMode;
00013
00015 typedef struct
00016 {
00017     uint32_t BaudRate;
00018     uint8_t WordLength;
00019     uint8_t StopBits;
00020     uint8_t FirstSend;
00021     UART_ParityMode ParityMode;
00022     uint8_t MultiDropEna;
00023     uint8_t Bit9Value;
00024     uint8_t DUPLEX;
00025     uint16_t Match485Addr;
00026     uint16_t IRQEna;
00027     uint8_t DMARE;
00028     uint8_t RXD_INV;
00029     uint8_t TXD_INV;
00030     uint8_t ABD_EN;
00031     uint8_t LBDL;
00032     uint8_t LIN_EN;
00033     uint8_t CK_EN;
00034 } UART_InitTypeDef;
00035
00037 #define UART_WORDLENGTH_8b 0
```

```

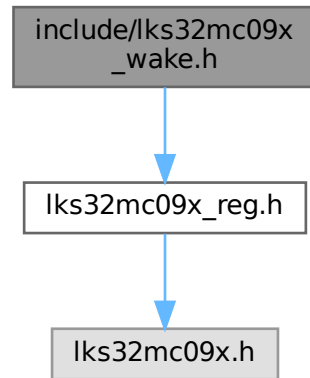
00038 #define UART_WORDLENGTH_9b 1
00039
00041 #define UART_STOPBITS_1b 0
00042 #define UART_STOPBITS_2b 1
00043
00045 #define UART_FIRSTSEND_LSB 0
00046 #define UART_FIRSTSEND_MSB 1
00047
00049 #define UART_IRQEna_SendOver      BIT0
00050 #define UART_IRQEna_RcvOver       BIT1
00051 #define UART_IRQEna_SendBuffEmpty BIT2
00052 #define UART_IRQEna_StopError     BIT3
00053 #define UART_IRQEna_CheckError    BIT4
00054 #define UART_IRQEna_TX_OV         BIT5
00055 #define UART_IRQEna_RX_OV         BIT6
00056 #define UART_IRQEna_LBD           BIT7
00057 #define UART_IRQEna_IDLE          BIT8
00058 #define UART_IRQEna_ABD           BIT9
00059
00061 #define TX_DMA_RE      BIT0
00062 #define RX_DMA_RE      BIT1
00063 #define TX_BUF_DMA_RE BIT2
00064
00066 #define UART_IF_SendOver      BIT0
00067 #define UART_IF_RcvOver       BIT1
00068 #define UART_IF_SendBufEmpty BIT2
00069 #define UART_IF_StopError     BIT3
00070 #define UART_IF_CheckError    BIT4
00071 #define UART_IF_TX_OV         BIT5
00072 #define UART_IF_RX_OV         BIT6
00073 #define UART_IF_LBD           BIT7
00074 #define UART_IF_IDLE          BIT8
00075 #define UART_IF_ABD           BIT9
00076
00078
00084 void UART_Init(UART_TypeDef *UARTx, UART_InitTypeDef *UART_InitStruct);
00085
00090 void UART_StructInit(UART_InitTypeDef *UART_InitStruct);
00091
00097 void UART_SendData(UART_TypeDef *UARTx, uint32_t n);
00098
00104 uint32_t UART_ReadData(UART_TypeDef *UARTx);
00105
00112 void UART_SendnData(UART_TypeDef *UARTx, char *sendData, uint32_t len);
00113
00120 void UART_RecvnData(UART_TypeDef *UARTx, char *recvData, uint32_t maxlen);
00121
00127 void UART_ClearIRQFlag(UART_TypeDef *UARTx, uint32_t tempFlag);
00128
00135 uint32_t UART_GetIRQFlag(UART_TypeDef *UARTx, uint32_t tempFlag);
00136
00143 uint8_t UART_SendAddr_485(UART_TypeDef *UARTx, uint8_t n);
00144
00151 uint8_t UART_SendData_485(UART_TypeDef *UARTx, uint8_t n);
00152
00159 uint32_t UART_SetBaud(UART_TypeDef *UARTx, uint32_t Baud);
00160
00165 void UART_SetClkDiv(uint32_t div);
00166
00167 #endif // __LKS32MC09X_UART_H__

```

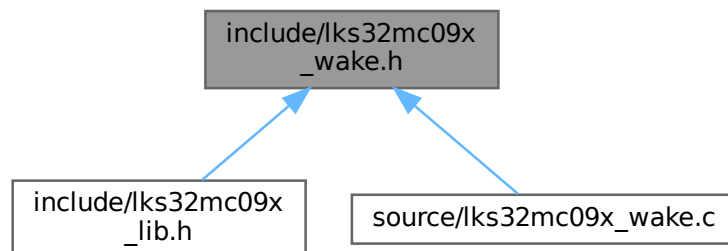
3.53 include/lks32mc09x_wake.h 文件参考

```
#include "lks32mc09x_reg.h"
```

lks32mc09x_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:



宏定义

- `#define SYS_WakeInterval_025s 0x00`
休眠唤醒间隔时间 $0.25s$
- `#define SYS_WakeInterval_05s 0x01`
休眠唤醒间隔时间 $0.5s$
- `#define SYS_WakeInterval_1s 0x02`
休眠唤醒间隔时间 $1s$
- `#define SYS_WakeInterval_2s 0x03`
休眠唤醒间隔时间 $2s$

- `#define SYS_WakeInterval_4s` 0x04
休眠唤醒间隔时间4s
- `#define SYS_WakeInterval_8s` 0x05
休眠唤醒间隔时间8s
- `#define SYS_WakeInterval_16s` 0x06
休眠唤醒间隔时间16s
- `#define SYS_WakeInterval_32s` 0x07
休眠唤醒间隔时间32s
- `#define WAKE_FLT_EN` 0x02
使能IO滤波
- `#define WAKE_FLT_DI` 0x00
关闭IO滤波
- `#define WAKE_IO_P0_0` BIT0
P0.0 作为唤醒IO
- `#define WAKE_IO_P0_2` BIT1
P0.2 作为唤醒IO
- `#define WAKE_IO_P0_12` BIT2
P0.12作为唤醒IO
- `#define WAKE_IO_P0_13` BIT3
P0.13作为唤醒IO
- `#define WAKE_IO_P0_14` BIT4
P0.14作为唤醒IO
- `#define WAKE_IO_P2_7` BIT5
P2.7 作为唤醒IO
- `#define WAKE_IO_P2_4` BIT6
P2.4 作为唤醒IO
- `#define WAKE_IO_P2_0` BIT7
P2.0 作为唤醒IO
- `#define WAKE_IO_CLUOUT0` BIT8
CLUOUT0 作为唤醒使能
- `#define WAKE_IO_CLUOUT1` BIT9
CLUOUT1 作为唤醒使能
- `#define WAKE_IO_CLUOUT2` BIT10
CLUOUT2 作为唤醒使能
- `#define WAKE_IO_CLUOUT3` BIT11
CLUOUT3 作为唤醒使能

枚举

- `enum WAKE_TRIG_x` { `WAKE_TRIG_LOW` , `WAKE_TRIG_HIGH` }
低功耗IO唤醒电平枚举成员

函数

- `void WAKE_EVTCLRSleep` (void)
清除休眠事件记录
- `void WAKE_FallSleep` (void)
进入低功耗模式
- `void WAKE_SetWakeInterval` (uint32_t rth, uint32_t with, uint8_t ena)
设置唤醒间隔时间
- `void WAKE_SetWakeIO` (uint32_t WAKE_IO_x, `WAKE_TRIG_x` trigLevel, uint32_t flt, uint8_t ena)
设置唤醒IO

3.53.1 宏定义说明

3.53.1.1 SYS_WakeInterval_025s

```
#define SYS_WakeInterval_025s 0x00
```

休眠唤醒间隔时间0.25s

在文件 [lks32mc09x_wake.h](#) 第 16 行定义.

3.53.1.2 SYS_WakeInterval_05s

```
#define SYS_WakeInterval_05s 0x01
```

休眠唤醒间隔时间0.5s

在文件 [lks32mc09x_wake.h](#) 第 17 行定义.

3.53.1.3 SYS_WakeInterval_16s

```
#define SYS_WakeInterval_16s 0x06
```

休眠唤醒间隔时间16s

在文件 [lks32mc09x_wake.h](#) 第 22 行定义.

3.53.1.4 SYS_WakeInterval_1s

```
#define SYS_WakeInterval_1s 0x02
```

休眠唤醒间隔时间1s

在文件 [lks32mc09x_wake.h](#) 第 18 行定义.

3.53.1.5 SYS_WakeInterval_2s

```
#define SYS_WakeInterval_2s 0x03
```

休眠唤醒间隔时间2s

在文件 [lks32mc09x_wake.h](#) 第 19 行定义.

3.53.1.6 SYS_WakeInterval_32s

```
#define SYS_WakeInterval_32s 0x07
```

休眠唤醒间隔时间32s

低功耗唤醒IO滤波

在文件 [lks32mc09x_wake.h](#) 第 25 行定义.

3.53.1.7 SYS_WakeInterval_4s

```
#define SYS_WakeInterval_4s 0x04
```

休眠唤醒间隔时间4s

在文件 [lks32mc09x_wake.h](#) 第 20 行定义.

3.53.1.8 SYS_WakeInterval_8s

```
#define SYS_WakeInterval_8s 0x05
```

休眠唤醒间隔时间8s

在文件 [lks32mc09x_wake.h](#) 第 21 行定义.

3.53.1.9 WAKE_FLT_DI

```
#define WAKE_FLT_DI 0x00
```

关闭IO滤波

低功耗唤醒IO选择

在文件 [lks32mc09x_wake.h](#) 第 29 行定义.

3.53.1.10 WAKE_FLT_EN

```
#define WAKE_FLT_EN 0x02
```

使能IO滤波

在文件 [lks32mc09x_wake.h](#) 第 26 行定义.

3.53.1.11 WAKE_IO_CLUOUT0

```
#define WAKE_IO_CLUOUT0 BIT8
```

CLUOUT0 作为唤醒使能

在文件 [lks32mc09x_wake.h](#) 第 38 行定义.

3.53.1.12 WAKE_IO_CLUOUT1

```
#define WAKE_IO_CLUOUT1 BIT9
```

CLUOUT1 作为唤醒使能

在文件 [lks32mc09x_wake.h](#) 第 39 行定义.

3.53.1.13 WAKE_IO_CLUOUT2

```
#define WAKE_IO_CLUOUT2 BIT10
```

CLUOUT2 作为唤醒使能

在文件 [lks32mc09x_wake.h](#) 第 40 行定义.

3.53.1.14 WAKE_IO_CLUOUT3

```
#define WAKE_IO_CLUOUT3 BIT11
```

CLUOUT3 作为唤醒使能

在文件 [lks32mc09x_wake.h](#) 第 41 行定义.

3.53.1.15 WAKE_IO_P0.0

```
#define WAKE_IO_P0.0 BIT0
```

P0.0 作为唤醒IO

在文件 [lks32mc09x_wake.h](#) 第 30 行定义.

3.53.1.16 WAKE_IO_P0.12

```
#define WAKE_IO_P0.12 BIT2
```

P0.12作为唤醒IO

在文件 [lks32mc09x_wake.h](#) 第 32 行定义.

3.53.1.17 WAKE_IO_P0.13

```
#define WAKE_IO_P0.13 BIT3
```

P0.13作为唤醒IO

在文件 [lks32mc09x_wake.h](#) 第 33 行定义.

3.53.1.18 WAKE_IO_P0.14

```
#define WAKE_IO_P0.14 BIT4
```

P0.14作为唤醒IO

在文件 [lks32mc09x_wake.h](#) 第 34 行定义.

3.53.1.19 WAKE_IO_P0.2

```
#define WAKE_IO_P0.2 BIT1
```

P0.2 作为唤醒IO

在文件 [lks32mc09x_wake.h](#) 第 31 行定义.

3.53.1.20 WAKE_IO_P2.0

```
#define WAKE_IO_P2.0 BIT7
```

P2.0 作为唤醒IO

在文件 [lks32mc09x_wake.h](#) 第 37 行定义.

3.53.1.21 WAKE_IO_P2.4

```
#define WAKE_IO_P2.4 BIT6
```

P2.4 作为唤醒IO

在文件 [lks32mc09x_wake.h](#) 第 36 行定义.

3.53.1.22 WAKE_IO_P2.7

```
#define WAKE_IO_P2.7 BIT5
```

P2.7 作为唤醒IO

在文件 [lks32mc09x_wake.h](#) 第 35 行定义.

3.53.2 枚举类型说明

3.53.2.1 WAKE_TRIG_x

```
enum WAKE_TRIG_x
```

低功耗IO唤醒电平枚举成员

枚举值

WAKE_TRIG_LOW	输入IO低电平唤醒
WAKE_TRIG_HIGH	输入IO高电平唤醒

在文件 [lks32mc09x_wake.h](#) 第 9 行定义.

3.53.3 函数说明

3.53.3.1 WAKE_EVTCLRSleep()

```
void WAKE_EVTCLRSleep (
    void )
```

清除休眠事件记录

在文件 [lks32mc09x_wake.c](#) 第 6 行定义.

3.53.3.2 WAKE_FallSleep()

```
void WAKE_FallSleep (
    void )
```

进入低功耗模式

进入低功耗模式

在文件 [lks32mc09x_wake.c](#) 第 15 行定义.

3.53.3.3 WAKE_SetWakeInterval()

```
void WAKE_SetWakeInterval (
    uint32_t rth,
    uint32_t wth,
    uint8_t ena )
```

设置唤醒间隔时间

参数

<i>rth</i>	唤醒复位阈值，由于计数器计数阈值是从复位阈值开始计数的，所有唤醒阈值必须大于复位阈值
<i>wth</i>	唤醒阈值
<i>ena</i>	使能标志

在文件 [lks32mc09x_wake.c](#) 第 106 行定义.

3.53.3.4 WAKE_SetWakeIO()

```
void WAKE_SetWakeIO (
    uint32_t WAKE_IO_x,
    WAKE_TRIG_x trigLevel,
    uint32_t flt,
    uint8_t ena )
```

设置唤醒IO

参数

<i>WAKE_IO_x</i>	唤醒IO选择
<i>trigLevel</i>	触发电平
<i>flt</i>	滤波使能
<i>ena</i>	使能标志
<i>wakeIO</i>	唤醒IO选择，必须打开对应的gpio输入使能（GPIO设置为输入模式），可以复用成任何模式
<i>trigLevel</i>	触发电平
<i>flt</i>	滤波使能
<i>ena</i>	使能标志

在文件 [lks32mc09x_wake.c](#) 第 138 行定义.

3.54 lks32mc09x_wake.h

浏览该文件的文档.

```

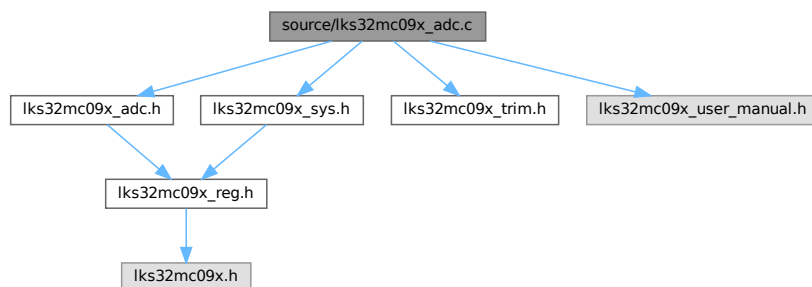
00001 #ifndef __LKS32MC09X_WAKE_H__
00002 #define __LKS32MC09X_WAKE_H__
00003
00004 #include "lks32mc09x_reg.h"
00005
00009 typedef enum
00010 {
00011     WAKE_TRIG_LOW,
00012     WAKE_TRIG_HIGH,
00013 } WAKE_TRIG_x;
00014
00016 #define SYS_WakeInterval_025s 0x00
00017 #define SYS_WakeInterval_05s 0x01
00018 #define SYS_WakeInterval_1s 0x02
00019 #define SYS_WakeInterval_2s 0x03
00020 #define SYS_WakeInterval_4s 0x04
00021 #define SYS_WakeInterval_8s 0x05
00022 #define SYS_WakeInterval_16s 0x06
00023 #define SYS_WakeInterval_32s 0x07
00024
00026 #define WAKE_FLT_EN 0x02
00027 #define WAKE_FLT_DI 0x00
00028
00030 #define WAKE_IO_P0_0 BIT0
00031 #define WAKE_IO_P0_2 BIT1
00032 #define WAKE_IO_P0_12 BIT2
00033 #define WAKE_IO_P0_13 BIT3
00034 #define WAKE_IO_P0_14 BIT4
00035 #define WAKE_IO_P2_7 BIT5
00036 #define WAKE_IO_P2_4 BIT6
00037 #define WAKE_IO_P2_0 BIT7
00038 #define WAKE_IO_CLUOUT0 BIT8
00039 #define WAKE_IO_CLUOUT1 BIT9
00040 #define WAKE_IO_CLUOUT2 BIT10
00041 #define WAKE_IO_CLUOUT3 BIT11
00042
00046 void WAKE_EVTCLRSleep(void);
00047
00051 void WAKE_FallSleep(void);
00052
00059 void WAKE_SetWakeInterval(uint32_t rth, uint32_t wth, uint8_t ena);
00060
00068 void WAKE_SetWakeIO(uint32_t WAKE_IO_x, WAKE_TRIG_x trigLevel, uint32_t flt, uint8_t ena);
00069
00070 #endif

```

3.55 source/lks32mc09x_adc.c 文件参考

```
#include "lks32mc09x_adc.h"
#include "lks32mc09x_sys.h"
#include "lks32mc09x_trim.h"
#include "lks32mc09x_user_manual.h"
```

lks32mc09x_adc.c 的引用(Include)关系图:



函数

- void **ADC_Enable** (ADC_TypeDef *ADCx)
使能ADC模块
- void **ADC_Disable** (ADC_TypeDef *ADCx)
禁用ADC模块
- void **ADC_Reset** (ADC_TypeDef *ADCx)
复位ADC模块
- void **ADC_Init** (ADC_TypeDef *ADCx, ADC_InitTypeDef *ADC_InitStruct)
初始化ADC模块
- void **ADC_StructInit** (ADC_InitTypeDef *ADC_InitStruct)
将ADC初始化结构体置为默认值
- uint16_t **ADC_GetIRQFlag** (ADC_TypeDef *ADCx, uint16_t INT_flag)
获取ADC中断标志位状态
- void **ADC_ClearIRQFlag** (ADC_TypeDef *ADCx, uint16_t INT_flag)
清除ADC中断标志位
- void **ADC_SetGain** (ADC_TypeDef *ADCx, ADC_DAT_x DAT_x, ADC_GAIN_x gainx)
设置ADC通道的增益
- s16 **ADC_GetConversionValue** (ADC_TypeDef *ADCx, ADC_DAT_x DATNum)
获取ADC通道的转换值
- int32_t **ADC_GetConversionValueS32** (ADC_TypeDef *ADCx, ADC_DAT_x DATNum)
获取ADC通道的转换值(按int32返回, 采样通道4、5、6、7时, 返回值大于32767)
- void **ADC_SoftTrig** (ADC_TypeDef *ADCx)
执行软件触发ADC采样, 采样次数取决于ADC_InitTypeDef里的S1的配置
- void **ADC_SoftIdleTrig** (ADC_TypeDef *ADCx)
执行软件触发ADC空闲采样
- void **ADC_SetChn** (ADC_TypeDef *ADCx, ADC_DAT_x datx, ADC_CHN_x chnx)
设置第每次的采样通道, 例如ADC_SetChn(ADC0, ADC_DAT_0, ADC_CHN_0), 把adc第一次采样的通道设置为通道0
- void **ADC_SetRange** (ADC_TypeDef *ADCx, ADC_RANGE_x rangex)
设置ADC基准电压

3.55.1 函数说明

3.55.1.1 ADC_ClearIRQFlag()

```
void ADC_ClearIRQFlag (
    ADC_TypeDef * ADCx,
    uint16_t INT_flag )
```

清除ADC中断标志位

参数

<i>ADCx</i>	ADC模块指针，指向ADC0或ADC1
<i>INT_flag</i>	中断标志位，可以是ADC_IF_ISF等

在文件 [lks32mc09x_adc.c](#) 第 151 行定义.

3.55.1.2 ADC_Disable()

```
void ADC_Disable (
    ADC_TypeDef * ADCx )
```

禁用ADC模块

参数

<i>ADCx</i>	ADC模块指针，指向ADC0或ADC1

在文件 [lks32mc09x_adc.c](#) 第 25 行定义.

3.55.1.3 ADC_Enable()

```
void ADC_Enable (
    ADC_TypeDef * ADCx )
```

使能ADC模块

参数

<i>ADCx</i>	ADC模块指针，指向ADC0或ADC1

在文件 [lks32mc09x_adc.c](#) 第 10 行定义.

这是这个函数的调用关系图:



3.55.1.4 ADC_GetConversionValue()

```
s16 ADC_GetConversionValue (
    ADC_TypeDef * ADCx,
    ADC_DAT_x DATNum )
```

获取ADC通道的转换值

参数

ADCx	ADC模块指针，指向ADC0或ADC1
DATNum	数据寄存器序号，如ADC_DAT_0

返回

转换值

在文件 [lks32mc09x_adc.c](#) 第 173 行定义.

函数调用图:



3.55.1.5 ADC_GetConversionValueS32()

```
int32_t ADC_GetConversionValueS32 (
    ADC_TypeDef * ADCx,
    ADC_DAT_x DATNum )
```

获取ADC通道的转换值(按int32返回，采样通道4、5、6、7时，返回值大于32767)

获取ADC通道的转换值(按int32返回，用来采样adc通道4、5、6、7)

参数

<i>ADCx</i>	ADC模块指针，指向ADC0或ADC1
<i>DATNum</i>	数据寄存器序号，如ADC_DAT_0

返回

转换值

在文件 [lks32mc09x_adc.c](#) 第 190 行定义.

这是这个函数的调用关系图:



3.55.1.6 ADC_GetIRQFlag()

```
uint16_t ADC_GetIRQFlag (
    ADC_TypeDef * ADCx,
    uint16_t INT_flag )
```

获取ADC中断标志位状态

参数

<i>ADCx</i>	ADC模块指针，指向ADC0或ADC1
<i>INT_flag</i>	中断标志位，可以是ADC_IF_ISF等

返回

中断标志位状态

在文件 [lks32mc09x_adc.c](#) 第 141 行定义.

3.55.1.7 ADC_Init()

```
void ADC_Init (
    ADC_TypeDef * ADCx,
    ADC_InitTypeDef * ADC_InitStruct )
```

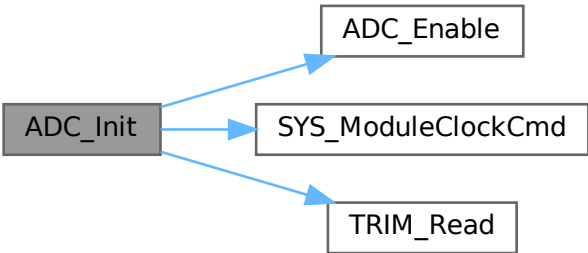
初始化ADC模块

参数

<i>ADCx</i>	ADC模块指针，指向ADC0或ADC1
<i>ADC_InitStruct</i>	指向ADC初始化结构体的指针

在文件 [lks32mc09x_adc.c](#) 第 61 行定义.

函数调用图:



3.55.1.8 ADC_Reset()

```
void ADC_Reset (
    ADC_TypeDef * ADCx )
```

复位ADC模块

参数

<i>ADCx</i>	ADC模块指针，指向ADC0或ADC1

在文件 [lks32mc09x_adc.c](#) 第 40 行定义.

函数调用图:



3.55.1.9 ADC_SetChn()

```
void ADC_SetChn (
    ADC_TypeDef * ADCx,
    ADC_DAT_x datx,
    ADC_CHN_x chnx )
```

设置第每次的采样通道，例如ADC_SetChn(ADC0,ADC_DAT_0,ADC_CHN_0),把adc第一次采样的通道设置为通道0

参数

<i>ADCx</i>	ADC模块指针，指向ADC0或ADC1
<i>datx</i>	数据寄存器序号，如ADC_DAT_0
<i>chnx</i>	ADC采样通道序号枚举定义，如ADC_CHN_0

在文件 [lks32mc09x_adc.c](#) 第 299 行定义.

3.55.1.10 ADC_SetGain()

```
void ADC_SetGain (
    ADC_TypeDef * ADCx,
    ADC_DAT_x DAT_x,
    ADC_GAIN_x gainx )
```

设置ADC通道的增益

参数

<i>ADCx</i>	ADC模块指针，指向ADC0或ADC1
<i>DAT↔ _x</i>	数据寄存器序号，如ADC_DAT_0表示第0次采样结果
<i>gainx</i>	增益选择，如ADC_GAIN_3V6

在文件 [lks32mc09x_adc.c](#) 第 162 行定义.

3.55.1.11 ADC_SetRange()

```
void ADC_SetRange (
    ADC_TypeDef * ADCx,
    ADC_RANGE_x rangex )
```

设置ADC基准电压

参数

<i>ADCx</i>	ADC模块指针,注意：adc0和adc1共用同一个基准电压
<i>rangex</i>	基准选择

- < 3.3V基准
- < 外部基准
- < 3.6V基准

在文件 [lks32mc09x_adc.c](#) 第 348 行定义.

3.55.1.12 ADC_SoftIdleTrig()

```
void ADC_SoftIdleTrig (  
    ADC_TypeDef * ADCx )
```

执行软件触发ADC空闲采样

参数

ADCx	ADC模块指针，指向ADC0

在文件 [lks32mc09x_adc.c](#) 第 287 行定义.

3.55.1.13 ADC_SoftTrig()

```
void ADC_SoftTrig (  
    ADC_TypeDef * ADCx )
```

执行软件触发ADC采样，采样次数取决于ADC.InitTypeDef里的S1的配置

参数

ADCx	ADC模块指针，指向ADC0或ADC1

在文件 [lks32mc09x_adc.c](#) 第 277 行定义.

3.55.1.14 ADC_StructInit()

```
void ADC_StructInit (  
    ADC_InitTypeDef * ADC_InitStruct )
```

将ADC初始化结构体置为默认值

参数

ADC_InitStruct	指向ADC初始化结构体的指针

在文件 [lks32mc09x_adc.c](#) 第 126 行定义.

3.56 lks32mc09x_adc.c

[浏览该文件的文档。](#)

```

00001 #include "lks32mc09x_adc.h"
00002 #include "lks32mc09x_sys.h"
00003 #include "lks32mc09x_trim.h"
00004 #include "lks32mc09x_user_manual.h"
00005
00010 void ADC_Enable(ADC_TypeDef *ADCx)
00011 {
00012     REG_WRITE(SYS0->PROTECT, 0x7a83);
00013     // 使能ADC模块时钟
00014     if ((ADCx == ADC0) || (ADCx == ADC1))
00015     {
00016         REG_SET(SYS0->AFE_REG7, SYS_AFE_REG7_SHAPDN_BIT | SYS_AFE_REG7_ADCPDN_BIT);
00017     }
00018     REG_WRITE(SYS0->PROTECT, 0);
00019 }
00020
00025 void ADC_Disable(ADC_TypeDef *ADCx)
00026 {
00027     REG_WRITE(SYS0->PROTECT, 0x7a83);
00028     // 关闭ADC模块时钟
00029     if ((ADCx == ADC0) || (ADCx == ADC1))
00030     {
00031         REG_RESET(SYS0->AFE_REG7, SYS_AFE_REG7_SHAPDN_BIT | SYS_AFE_REG7_ADCPDN_BIT);
00032     }
00033     REG_WRITE(SYS0->PROTECT, 0);
00034 }
00035
00040 void ADC_Reset(ADC_TypeDef *ADCx)
00041 {
00042     if (ADCx == ADC0)
00043     {
00044         SYS_SoftResetModule(SYS_MODULE_ADC0);
00045     }
00046     else if (ADCx == ADC1)
00047     {
00048         SYS_SoftResetModule(SYS_MODULE_ADC1);
00049     }
00050     else
00051     {
00052         _NOP();
00053     }
00054 }
00055
00061 void ADC_Init(ADC_TypeDef *ADCx, ADC_InitTypeDef *ADC_InitStruct)
00062 {
00063     ADC_Enable(ADCx);
00064     SYS_ModuleClockCmd(SYS_MODULE_ADC0, 1);
00065     SYS_ModuleClockCmd(SYS_MODULE_ADC1, 1);
00066
00067     if (ADC_InitStruct->S2 == 0)
00068     {
00069         ADC_InitStruct->NSMP = DISABLE; // 当第二段采样次数为0时, 不允许配置第二段使能
00070     }
00071
00072     // 解锁SYS寄存器
00073     REG_WRITE(SYS0->PROTECT, 0x7a83);
00074
00075     // 配置ADC初始化结构体中的各个成员
00076     REG_WRITE(ADCx->IE, ADC_InitStruct->IE | ADC_InitStruct->RE);
00077
00078     // 计算并一次性配置CFG寄存器
00079     REG_WRITE(ADCx->CFG, ((ADC_InitStruct->NSMP << 12) |
00080                          (1 << 11) | // 初始化时复位adc状态机
00081                          (ADC_InitStruct->DATA_ALIGN << 10) |
00082                          (ADC_InitStruct->IDLE_PRI << 9) |
00083                          (ADC_InitStruct->CSMP << 8) |
00084                          (ADC_InitStruct->TCNT << 4) |
00085                          (ADC_InitStruct->TROVS << 3) |
00086                          (ADC_InitStruct->OVSR << 0)));
00087
00088     REG_WRITE(ADCx->TRIG, ADC_InitStruct->TRIG);
00089
00090     // 配置CHNT寄存器
00091     REG_WRITE(ADCx->CHNT, (REG_READ(ADCx->CHNT) & ~((0x0F << 0) | (0x0F << 4) | (0x0F << 8))) |
00092                  ((ADC_InitStruct->S1 << 0) |
00093                   (ADC_InitStruct->S2 << 4) |
00094                   (ADC_InitStruct->IS1 << 8)));
00095
00096     // 配置GAIN寄存器
00097     REG_WRITE_BIT(ADCx->GAIN, 1 << 0, ADC_InitStruct->GAIN << 0);
00098
00099     REG_WRITE(ADCx->LTH, ADC_InitStruct->LTH);

```

```

00100     REG_WRITE(ADCx->HTH, ADC_InitStruct->HTH);
00101     REG_WRITE(ADCx->GEN, ADC_InitStruct->GEN);
00102     if (ADCx == ADC0)
00103     {
00104         REG_WRITE(ADCx->AMC0, TRIM_Read(NVR_ADDR_ADC_0_AMC0));
00105         REG_WRITE(ADCx->DC0, TRIM_Read(NVR_ADDR_ADC_0_DC0));
00106         REG_WRITE(ADCx->AMC1, TRIM_Read(NVR_ADDR_ADC_0_AMC1));
00107         REG_WRITE(ADCx->DC1, TRIM_Read(NVR_ADDR_ADC_0_DC1));
00108     }
00109     else if (ADCx == ADC1)
00110     {
00111         REG_WRITE(ADCx->AMC0, TRIM_Read(NVR_ADDR_ADC_1_AMC0));
00112         REG_WRITE(ADCx->DC0, TRIM_Read(NVR_ADDR_ADC_1_DC0));
00113         REG_WRITE(ADCx->AMC1, TRIM_Read(NVR_ADDR_ADC_1_AMC1));
00114         REG_WRITE(ADCx->DC1, TRIM_Read(NVR_ADDR_ADC_1_DC1));
00115     }
00116     else
00117     {
00118         __NOP();
00119     }
00120 }
00121
00126 void ADC_StructInit(ADC_InitTypeDef *ADC_InitStruct)
00127 {
00128     uint32_t i;
00129     for (i = 0; i < sizeof(ADC_InitTypeDef); i++)
00130     {
00131         ((uint8_t *)ADC_InitStruct)[i] = 0;
00132     }
00133 }
00134
00141 uint16_t ADC_GetIRQFlag(ADC_TypeDef *ADCx, uint16_t INT_flag)
00142 {
00143     return REG_READ_BIT(ADCx->IF, INT_flag) ? 1 : 0;
00144 }
00145
00151 void ADC_ClearIRQFlag(ADC_TypeDef *ADCx, uint16_t INT_flag)
00152 {
00153     REG_WRITE(ADCx->IF, INT_flag);
00154 }
00155
00162 void ADC_SetGain(ADC_TypeDef *ADCx, ADC_DAT_x DAT_x, ADC_GAIN_x gainx)
00163 {
00164     REG_WRITE_BIT(ADCx->GAIN, 1 << DAT_x, gainx << DAT_x);
00165 }
00166
00173 s16 ADC_GetConversionValue(ADC_TypeDef *ADCx, ADC_DAT_x DATNum)
00174 {
00175     int32_t adc_dat = ADC_GetConversionValueS32(ADCx, DATNum);
00176     // adc_dat不会小于-32767, 这里不做判断
00177     if (adc_dat > 32767)
00178     {
00179         adc_dat = 32767;
00180     }
00181     return adc_dat;
00182 }
00183
00190 int32_t ADC_GetConversionValueS32(ADC_TypeDef *ADCx, ADC_DAT_x DATNum)
00191 {
00192     int32_t adc_dat = 0;
00193     uint16_t adc_chn = 0;
00194     switch (DATNum)
00195     {
00196     {
00197         case ADC_DAT_0:
00198             adc_dat = REG_READ(ADCx->DAT0);
00199             adc_chn = REG_READ_BIT_POS_MASK(ADCx->CHN0, ADC0_CHN0_PDS0);
00200             break;
00201         case ADC_DAT_1:
00202             adc_dat = REG_READ(ADCx->DAT1);
00203             adc_chn = REG_READ_BIT_POS_MASK(ADCx->CHN0, ADC0_CHN0_PDS1);
00204             break;
00205         case ADC_DAT_2:
00206             adc_dat = REG_READ(ADCx->DAT2);
00207             adc_chn = REG_READ_BIT_POS_MASK(ADCx->CHN0, ADC0_CHN0_PDS2);
00208             break;
00209         case ADC_DAT_3:
00210             adc_dat = REG_READ(ADCx->DAT3);
00211             adc_chn = REG_READ_BIT_POS_MASK(ADCx->CHN0, ADC0_CHN0_PDS3);
00212             break;
00213         case ADC_DAT_4:
00214             adc_dat = REG_READ(ADCx->DAT4);
00215             adc_chn = REG_READ_BIT_POS_MASK(ADCx->CHN1, ADC0_CHN1_PDS4);
00216             break;
00217         case ADC_DAT_5:
00218             adc_dat = REG_READ(ADCx->DAT5);
00219             adc_chn = REG_READ_BIT_POS_MASK(ADCx->CHN1, ADC0_CHN1_PDS5);

```

```

00220         break;
00221     case ADC_DAT_6:
00222         adc_dat = REG_READ(ADCx->DAT6);
00223         adc_chn = REG_READ_BIT_POS_MASK(ADCx->CHN1, ADC0.CHN1.PDS6);
00224         break;
00225     case ADC_DAT_7:
00226         adc_dat = REG_READ(ADCx->DAT7);
00227         adc_chn = REG_READ_BIT_POS_MASK(ADCx->CHN1, ADC0.CHN1.PDS7);
00228         break;
00229     case ADC_DAT_8:
00230         adc_dat = REG_READ(ADCx->DAT8);
00231         adc_chn = REG_READ_BIT_POS_MASK(ADCx->CHN2, ADC0.CHN2.PDS8);
00232         break;
00233     case ADC_DAT_9:
00234         adc_dat = REG_READ(ADCx->DAT9);
00235         adc_chn = REG_READ_BIT_POS_MASK(ADCx->CHN2, ADC0.CHN2.PDS9);
00236         break;
00237     case ADC_DAT_10:
00238         adc_dat = REG_READ(ADCx->DAT10);
00239         adc_chn = REG_READ_BIT_POS_MASK(ADCx->CHN2, ADC0.CHN2.PDS10);
00240         break;
00241     case ADC_DAT_11:
00242         adc_dat = REG_READ(ADCx->DAT11);
00243         adc_chn = REG_READ_BIT_POS_MASK(ADCx->CHN2, ADC0.CHN2.PDS11);
00244         break;
00245     default:
00246         adc_dat = 0;
00247         break;
00248 }
00249 // 先按照int16处理一下
00250 adc_dat = (int16_t)(adc_dat & 0xffff);
00251 // ADC0/1采样公共通道4/5/6/7时, 负端电压为2.4V, 实际允许的输入电压范围是
00252 // 仅内部基准有效
00253 if ((adc_chn == 4) || (adc_chn == 5) || (adc_chn == 6) || (adc_chn == 7))
00254 {
00255     int32_t dat = adc_dat;
00256     // 使用外部基准时需要手动处理
00257     if (REG_READ_BIT(SYS_AFE_REG4, SYS_AFE_REG4.REF2VDD_BIT) == 0)
00258     {
00259         if (REG_READ_BIT(ADC0.CFG, ADC0.CFG.DATA_ALIGN_BIT) == 0)
00260         { // 左对齐
00261             dat = dat + 0x5550;
00262             adc_dat = dat;
00263         }
00264         else
00265         { // 右对齐
00266             dat = dat + 0x555;
00267             adc_dat = dat;
00268         }
00269     }
00270 }
00271 return adc_dat;
00272 }
00273 void ADC_SoftTrig(ADC_TypeDef *ADCx)
00274 {
00275     // 写入软件触发值
00276     REG_WRITE(ADCx->SWT, 0x5AA5);
00277 }
00278 void ADC_SoftIdleTrig(ADC_TypeDef *ADCx)
00279 {
00280     // 写入软件触发值
00281     REG_WRITE(ADCx->SWT, 0xf00f);
00282 }
00283 void ADC_SetChn(ADC_TypeDef *ADCx, ADC_DAT_x datx, ADC_CHN_x chnx)
00284 {
00285     uint8_t n = (datx & 0x3) * 4;
00286     switch (datx)
00287     {
00288     case ADC_DAT_0:
00289         REG_WRITE(ADCx->CHN0, (REG_READ(ADCx->CHN0) & ~(0xf << n)) | (chnx << n));
00290         break;
00291     case ADC_DAT_1:
00292         REG_WRITE(ADCx->CHN0, (REG_READ(ADCx->CHN0) & ~(0xf << n)) | (chnx << n));
00293         break;
00294     case ADC_DAT_2:
00295         REG_WRITE(ADCx->CHN0, (REG_READ(ADCx->CHN0) & ~(0xf << n)) | (chnx << n));
00296         break;
00297     case ADC_DAT_3:
00298         REG_WRITE(ADCx->CHN0, (REG_READ(ADCx->CHN0) & ~(0xf << n)) | (chnx << n));
00299         break;
00300     case ADC_DAT_4:
00301         REG_WRITE(ADCx->CHN1, (REG_READ(ADCx->CHN1) & ~(0xf << n)) | (chnx << n));
00302         break;
00303     case ADC_DAT_5:
00304         REG_WRITE(ADCx->CHN1, (REG_READ(ADCx->CHN1) & ~(0xf << n)) | (chnx << n));
00305     }
00306 }

```

```

00321         break;
00322     case ADC_DAT_6:
00323         REG_WRITE(ADCx->CHN1, (REG_READ(ADCx->CHN1) & ~(0xf << n)) | (chnx << n));
00324         break;
00325     case ADC_DAT_7:
00326         REG_WRITE(ADCx->CHN1, (REG_READ(ADCx->CHN1) & ~(0xf << n)) | (chnx << n));
00327         break;
00328     case ADC_DAT_8:
00329         REG_WRITE(ADCx->CHN2, (REG_READ(ADCx->CHN2) & ~(0xf << n)) | (chnx << n));
00330         break;
00331     case ADC_DAT_9:
00332         REG_WRITE(ADCx->CHN2, (REG_READ(ADCx->CHN2) & ~(0xf << n)) | (chnx << n));
00333         break;
00334     case ADC_DAT_10:
00335         REG_WRITE(ADCx->CHN2, (REG_READ(ADCx->CHN2) & ~(0xf << n)) | (chnx << n));
00336         break;
00337     case ADC_DAT_11:
00338         REG_WRITE(ADCx->CHN2, (REG_READ(ADCx->CHN2) & ~(0xf << n)) | (chnx << n));
00339         break;
00340     }
00341 }
00342
00348 void ADC_SetRange(ADC_TypeDef *ADCx, ADC_RANGE_x rangex)
00349 {
00350     if ((ADCx == ADC0) || (ADCx == ADC1))
00351     {
00352         REG_WRITE(SYS_WR_PROTECT, 0x7a83);
00353         switch (rangex)
00354         {
00355             case ADC_RANGE_3_3:
00356                 REG_SET(SYS_AFE_REG0, BIT15);
00357                 REG_RESET(SYS_AFE_REG4, BIT4);
00358                 break;
00359             case ADC_RANGE_EXTERNAL:
00360                 REG_RESET(SYS_AFE_REG0, BIT15);
00361                 REG_SET(SYS_AFE_REG4, BIT4);
00362                 break;
00363             default: // 默认配置为3.6V基准
00364                 case ADC_RANGE_3_6:
00365                     REG_RESET(SYS_AFE_REG0, BIT15);
00366                     REG_RESET(SYS_AFE_REG4, BIT4);
00367                     break;
00368         }
00369         REG_WRITE(SYS_WR_PROTECT, 0);
00370     }
00371 }

```

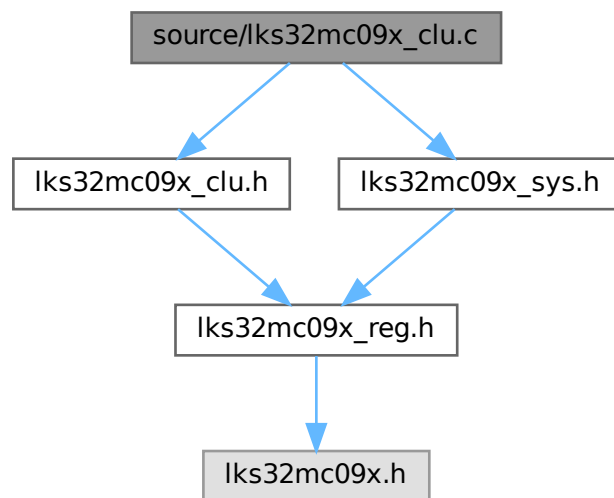
3.57 source/lks32mc09x_clu.c 文件参考

```

#include "lks32mc09x_clu.h"
#include "lks32mc09x_sys.h"

```

lks32mc09x_clu.c 的引用(Include)关系图:



函数

- void **CLU_Init** (**CLU_TypeDef** *CLUx, uint8_t CLU_MODEL_CLUx, **CLU_InitTypeDef** *CLU_InitStruct)
CLU计算初始化
- void **CLU_StructInit** (**CLU_InitTypeDef** *CLU_InitStruct)
CLU初始化结构体初始化
- void **CLU_Enable** (**CLU_TypeDef** *CLUx)
CLU模块时钟使能
- void **CLU_Disable** (**CLU_TypeDef** *CLUx)
CLU模块时钟关闭
- uint8_t **CLU_ReadOutValue** (**CLU_TypeDef** *CLUx, uint8_t CLU_MODEL_CLUx)
获取CLU的输出值

3.57.1 函数说明

3.57.1.1 CLU_Disable()

```
void CLU_Disable (
    CLU_TypeDef * CLUx )
```

CLU模块时钟关闭

在文件 **lks32mc09x_clu.c** 第 79 行定义.

函数调用图:



3.57.1.2 CLU_Enable()

```
void CLU_Enable (
    CLU_TypeDef * CLUx )
```

CLU模块时钟使能

在文件 [lks32mc09x_clu.c](#) 第 68 行定义.

函数调用图:



这是这个函数的调用关系图:



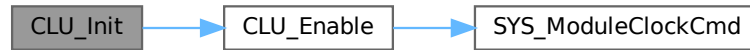
3.57.1.3 CLU_Init()

```
void CLU_Init (
    CLU_TypeDef * CLUx,
    uint8_t CLU_MODEL_CLUx,
    CLU_InitTypeDef * CLU_InitStruct )
```

CLU计算初始化

在文件 [lks32mc09x_clu.c](#) 第 8 行定义.

函数调用图:



3.57.1.4 CLU_ReadOutValue()

```
uint8_t CLU_ReadOutValue (
    CLU_TypeDef * CLUx,
    uint8_t CLU_MODEL_CLUx )
```

获取CLU的输出值

在文件 [lks32mc09x_clu.c](#) 第 90 行定义.

3.57.1.5 CLU_StructInit()

```
void CLU_StructInit (
    CLU_InitTypeDef * CLU_InitStruct )
```

CLU初始化结构体初始化

在文件 [lks32mc09x_clu.c](#) 第 56 行定义.

3.58 lks32mc09x_clu.c

浏览该文件的文档.

```

00001 #include "lks32mc09x_clu.h"
00002 #include "lks32mc09x_sys.h"
00003
00008 void CLU_Init(CLU_TypeDef *CLUx, uint8_t CLU_MODEL_CLUx, CLU_InitTypeDef *CLU_InitStruct)
00009 {
00010     CLU_Enable(CLUx);
00011     if ((CLU_MODEL_CLUx & CLU_MODEL_CLU0) != 0)
00012     {
00013         REG_WRITE_BIT(CLUx->EN, BIT0, CLU_InitStruct->EN ? BIT0 : 0);
00014         REG_WRITE_BIT(CLUx->IE, BIT0, CLU_InitStruct->FIE ? BIT0 : 0);
00015         REG_WRITE_BIT(CLUx->IE, BIT1, CLU_InitStruct->RIE ? BIT1 : 0);
00016         REG_WRITE_BIT(CLUx->MX, 0x000f, (CLU_InitStruct->MXA << 4) | (CLU_InitStruct->MXB));
00017         REG_WRITE_BIT(CLUx->FN, 0x000f, CLU_InitStruct->FN);
00018         REG_WRITE_BIT(CLUx->CF, 0x000f, (CLU_InitStruct->OUTSEL << 7) | (CLU_InitStruct->OEN << 6) |
(CLU_InitStruct->CLKINV << 2) | (CLU_InitStruct->CLKSEL));
00019         REG_SET(CLUx->CF, BIT3);
00020     }
00021     if ((CLU_MODEL_CLUx & CLU_MODEL_CLU1) != 0)
00022     {
00023         REG_WRITE_BIT(CLUx->EN, BIT1, CLU_InitStruct->EN ? BIT1 : 0);
00024         REG_WRITE_BIT(CLUx->IE, BIT2, CLU_InitStruct->FIE ? BIT2 : 0);
00025         REG_WRITE_BIT(CLUx->IE, BIT3, CLU_InitStruct->RIE ? BIT3 : 0);
  
```

```

00026     REG_WRITE_BIT(CLUx->MX, 0x00f0, (CLU_InitStruct->MXA << 12) | (CLU_InitStruct->MXB << 8));
00027     REG_WRITE_BIT(CLUx->FN, 0x00f0, CLU_InitStruct->FN << 8);
00028     REG_WRITE_BIT(CLUx->CF, 0x00f0, (CLU_InitStruct->OUTSEL << 15) | (CLU_InitStruct->OEN << 14) |
(CLU_InitStruct->CLKINV << 10) | (CLU_InitStruct->CLKSEL << 8));
00029     REG_SET(CLUx->CF, BIT11);
00030 }
00031 if ((CLU_MODEL_CLUx & CLU_MODEL_CLU2) != 0)
00032 {
00033     REG_WRITE_BIT(CLUx->EN, BIT2, CLU_InitStruct->EN ? BIT2 : 0);
00034     REG_WRITE_BIT(CLUx->IE, BIT4, CLU_InitStruct->FIE ? BIT4 : 0);
00035     REG_WRITE_BIT(CLUx->IE, BIT5, CLU_InitStruct->RIE ? BIT5 : 0);
00036     REG_WRITE_BIT(CLUx->MX, 0x0f00, (CLU_InitStruct->MXA << 20) | (CLU_InitStruct->MXB << 16));
00037     REG_WRITE_BIT(CLUx->FN, 0x0f00, CLU_InitStruct->FN << 16);
00038     REG_WRITE_BIT(CLUx->CF, 0x0f00, (CLU_InitStruct->OUTSEL << 23) | (CLU_InitStruct->OEN << 22) |
(CLU_InitStruct->CLKINV << 18) | (CLU_InitStruct->CLKSEL << 16));
00039     REG_SET(CLUx->CF, BIT19);
00040 }
00041 if ((CLU_MODEL_CLUx & CLU_MODEL_CLU3) != 0)
00042 {
00043     REG_WRITE_BIT(CLUx->EN, BIT3, CLU_InitStruct->EN ? BIT3 : 0);
00044     REG_WRITE_BIT(CLUx->IE, BIT6, CLU_InitStruct->FIE ? BIT6 : 0);
00045     REG_WRITE_BIT(CLUx->IE, BIT7, CLU_InitStruct->RIE ? BIT7 : 0);
00046     REG_WRITE_BIT(CLUx->MX, 0xf000, (CLU_InitStruct->MXA << 28) | (CLU_InitStruct->MXB << 24));
00047     REG_WRITE_BIT(CLUx->FN, 0xf000, CLU_InitStruct->FN << 24);
00048     REG_WRITE_BIT(CLUx->CF, 0xf000, (CLU_InitStruct->OUTSEL << 31) | (CLU_InitStruct->OEN << 30) |
(CLU_InitStruct->CLKINV << 26) | (CLU_InitStruct->CLKSEL << 24));
00049     REG_SET(CLUx->CF, BIT27);
00050 }
00051 }
00052
00056 void CLU_StructInit(CLU_InitTypeDef *CLU_InitStruct)
00057 {
00058     uint32_t i;
00059     for (i = 0; i < sizeof(CLU_InitTypeDef); i++)
00060     {
00061         ((uint8_t *)CLU_InitStruct)[i] = 0;
00062     }
00063 }
00064
00068 void CLU_Enable(CLU_TypeDef *CLUx)
00069 {
00070     if (CLUx == CLU0)
00071     {
00072         SYS_ModuleClockCmd(SYS_MODULE_CL0, ENABLE);
00073     }
00074 }
00075
00079 void CLU_Disable(CLU_TypeDef *CLUx)
00080 {
00081     if (CLUx == CLU0)
00082     {
00083         SYS_ModuleClockCmd(SYS_MODULE_CL0, DISABLE);
00084     }
00085 }
00086
00090 uint8_t CLU_ReadOutV alue(CLU_TypeDef *CLUx, uint8_t CLU_MODEL_CLUx)
00091 {
00092     return (REG_READ_BIT(CLUx->OUT, CLU_MODEL_CLUx) != 0) ? 1 : 0;
00093 }

```

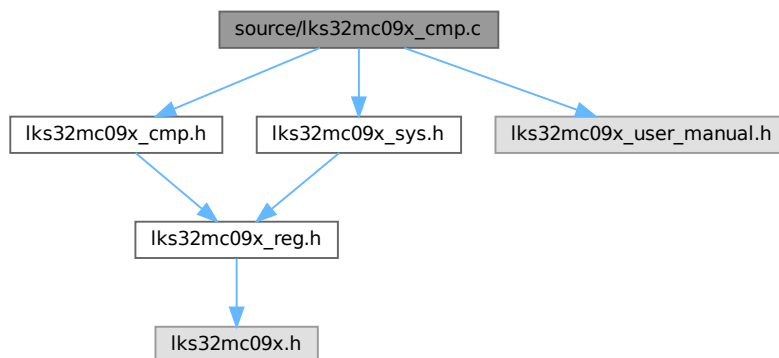
3.59 source/lks32mc09x_cmp.c 文件参考

```

#include "lks32mc09x_cmp.h"
#include "lks32mc09x_sys.h"
#include "lks32mc09x_user_manual.h"

```

lks32mc09x_cmp.c 的引用(Include)关系图:



函数

- void **CMP_Enable** (uint8_t CMP_CHN_x)
使能比较器
- void **CMP_Disable** (uint8_t CMP_CHN_x)
失能比较器
- void **CMP_SetSignal** (uint8_t CMP_CHN_x, uint8_t CMP_SELN_x_x, uint8_t CMP_SELP_x_x)
设置比较器信号来源
- void **CMP_SetHysteresis** (uint8_t CMP_HYS_x)
设置比较器回差(两个比较器共用同一个回差)
- void **CMP_SetWindowSignal** (uint8_t CMP_CHN_x, uint8_t CMP_WIN_x, uint8_t CMP_WIN_LEVEL_x)
设置比较器开窗信号
- void **CMP_SetFilterWidth** (uint8_t CMP_CLK_DIV_x, uint16_t CMP0_FIL_x, uint16_t CMP1_FIL_x)
设置比较器的滤波宽度
- void **CMP_SetIRQSignal** (uint8_t CMP_CHN_x, uint8_t CMP_SIGNAL_x)
设置比较器中断信号
- void **CMP_ClearIRQFlag** (uint8_t CMP_CHN_x)
清除CMP中断标志
- uint8_t **CMP_GetIRQFlag** (uint8_t CMP_CHN_x)
读取CMP中断标志
- uint8_t **CMP_GetFilterValue** (uint8_t CMP_CHN_x)
读取滤波后的比较器值
- uint8_t **CMP_GetValue** (uint8_t CMP_CHN_x)
读取未滤波的比较器值
- void **CMP_StructInit** (CMP_InitTypeDef *CMP_InitStruct)
CMP结构体初始化
- void **CMP_Init** (CMP_InitTypeDef *CMP_InitStruct)
CMP初始化

3.59.1 函数说明

3.59.1.1 CMP_ClearIRQFlag()

```
void CMP_ClearIRQFlag (
    uint8_t CMP_CHN_x )
```

清除CMP中断标志

参数

<i>CMP_↔ CHN_x</i>	比较器通道选择, CMP_CHN_0, CMP_CHN_1

在文件 [lks32mc09x_cmp.c](#) 第 236 行定义.

3.59.1.2 CMP_Disable()

```
void CMP_Disable (
    uint8_t CMP_CHN_x )
```

失能比较器

参数

<i>CMP_↔ CHN_x</i>	比较器通道选择, CMP_CHN_0, CMP_CHN_1

在文件 [lks32mc09x_cmp.c](#) 第 38 行定义.

函数调用图:



这是这个函数的调用关系图:



3.59.1.3 CMP_Enable()

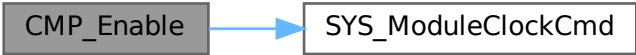
```
void CMP_Enable (
    uint8_t CMP_CHN_x )
```

使能比较器

参数

CMP_↔ CHN_x	比较器通道选择，CMP_CHN_0, CMP_CHN_1

在文件 [lks32mc09x_cmp.c](#) 第 10 行定义.
函数调用图:



这是这个函数的调用关系图:



3.59.1.4 CMP_GetFilterValue()

```
uint8_t CMP_GetFilterValue (
    uint8_t CMP_CHN_x )
```

读取滤波后的比较器值

返回

: 滤波后的比较器值 0或1

在文件 [lks32mc09x_cmp.c](#) 第 281 行定义.

3.59.1.5 CMP_GetIRQFlag()

```
uint8_t CMP_GetIRQFlag (
    uint8_t CMP_CHN_x )
```

读取CMP中断标志

参数

<i>CMP_↔ CHN_x</i>	比较器通道选择, CMP_CHN_0, CMP_CHN_1

返回

: 中断标志状态 0或1

在文件 [lks32mc09x_cmp.c](#) 第 261 行定义.

3.59.1.6 CMP_GetValue()

```
uint8_t CMP_GetValue (
    uint8_t CMP_CHN_x )
```

读取未滤波的比较器值

返回

: 未滤波的比较器值 0或1

在文件 [lks32mc09x_cmp.c](#) 第 301 行定义.

3.59.1.7 CMP_Init()

```
void CMP_Init (
    CMP_InitTypeDef * CMP_InitStruct )
```

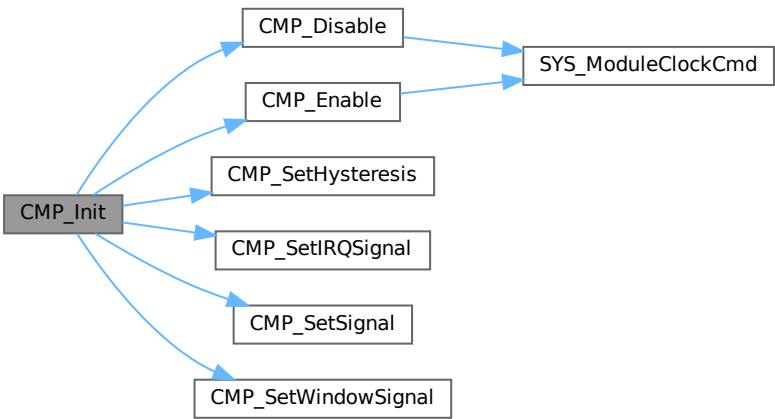
CMP初始化

参数

<i>CMP_InitStruct</i>	指向CMP初始化结构体的指针

在文件 [lks32mc09x_cmp.c](#) 第 332 行定义.

函数调用图:



3.59.1.8 CMP_SetFilterWidth()

```
void CMP_SetFilterWidth (
    uint8_t CMP_CLK_DIV_x,
    uint16_t CMP0_FIL_x,
    uint16_t CMP1_FIL_x )
```

设置比较器的滤波宽度

参数

<i>CMP_CLK_↔ DIV_x</i>	比较器时钟分频系数，CMP_CLK_DIV_0, CMP_CLK_DIV_1, CMP_CLK_DIV_2, CMP_CLK_DIV_3, CMP_CLK_DIV_4
<i>CMP0_FIL_x</i>	比较器0滤波宽度，取值范围0-2047，0表示关闭信号的输入
<i>CMP1_FIL_x</i>	比较器1滤波宽度，取值范围0-2047，0表示关闭信号的输入

在文件 [lks32mc09x_cmp.c](#) 第 149 行定义.

3.59.1.9 CMP_SetHysteresis()

```
void CMP_SetHysteresis (
    uint8_t CMP_HYS_x )
```

设置比较器回差(两个比较器共用同一个回差)

参数

<i>CMP_↔ HYS_x</i>	回差选择，CMP_HYS_20mV, CMP_HYS_0mV

在文件 [lks32mc09x_cmp.c](#) 第 104 行定义.

这是这个函数的调用关系图:



3.59.1.10 CMP_SetIRQSignal()

```
void CMP_SetIRQSignal (
    uint8_t CMP_CHN_x,
    uint8_t CMP_SIGNAL_x )
```

设置比较器中断信号

参数

<i>CMP_CHN_x</i>	比较器通道选择, CMP_CHN_0, CMP_CHN_1
<i>CMP_↔ SIGNAL_x</i>	中断信号选择, CMP_SIGNAL_NONE, CMP_SIGNAL_EDGE_P, CMP_SIGNAL_EDGE_N, CMP_SIGNAL_LEVEL_H, CMP_SIGNAL_LEVEL_L

在文件 [lks32mc09x_cmp.c](#) 第 195 行定义.

这是这个函数的调用关系图:



3.59.1.11 CMP_SetSignal()

```
void CMP_SetSignal (
    uint8_t CMP_CHN_x,
    uint8_t CMP_SELN_x_x,
    uint8_t CMP_SELN_x_x )
```

设置比较器信号来源

参数

<i>CMP_CHN_x</i>	比较器通道选择, CMP_CHN_0, CMP_CHN_1
<i>CMP_SELN_x↔x</i>	信号负端选择, 注意, 由于比较器的各个通道不完全一样, 所以需要指明使用的比较器
<i>CMP_SELP_x↔x</i>	信号正端选择, 注意, 由于比较器的各个通道不完全一样, 所以需要指明使用的比较器

在文件 [lks32mc09x_cmp.c](#) 第 72 行定义.

这是这个函数的调用关系图:



3.59.1.12 CMP_SetWindowSignal()

```
void CMP_SetWindowSignal (
    uint8_t CMP_CHN_x,
    uint8_t CMP_WIN_x,
    uint8_t CMP_WIN_LEVEL_x )
```

设置比较器开窗信号

参数

<i>CMP_CHN_x</i>	比较器通道选择, CMP_CHN_0, CMP_CHN_1
<i>CMP_WIN_x</i>	开窗信号选择, CMP_WIN_SIGNAL_NONE, CMP_WIN_SIGNAL_MCPWM0_0
<i>CMP_WIN_LEVEL_x↔LEVEL_x</i>	开窗电平选择, CMP_WIN_LEVEL_H, CMP_WIN_LEVEL_L

在文件 [lks32mc09x_cmp.c](#) 第 118 行定义.

这是这个函数的调用关系图:



3.59.1.13 CMP_StructInit()

```
void CMP_StructInit (
    CMP_InitTypeDef * CMP_InitStruct )
```

CMP结构体初始化

参数

<i>CMP_InitStruct</i>	指向CMP初始化结构体的指针

在文件 [lks32mc09x_cmp.c](#) 第 320 行定义.

3.60 lks32mc09x_cmp.c

[浏览该文件的文档.](#)

```
00001 #include "lks32mc09x_cmp.h"
00002 #include "lks32mc09x_sys.h"
00003 #include "lks32mc09x_user_manual.h"
00004
00005 // 模拟部分
00010 void CMP_Enable(uint8_t CMP_CHN_x)
00011 {
00012     SYS_ModuleClockCmd(SYS_MODULE_CMP, ENABLE);
00013     if (CMP_CHN_x == CMP_CHN_0)
00014     {
00015         SYS0->PROTECT = 0x7a83;
00016         SYS0->AFE_REG5 |= SYS_AFE_REG5_CMP0PDN_BIT;
00017         SYS0->PROTECT = 0;
00018     }
00019     else
00020     {
00021         __NOP();
00022     }
00023     if (CMP_CHN_x == CMP_CHN_1)
00024     {
00025         SYS0->PROTECT = 0x7a83;
00026         SYS0->AFE_REG5 |= SYS_AFE_REG5_CMP1PDN_BIT;
00027         SYS0->PROTECT = 0;
00028     }
00029     else
00030     {
00031         __NOP();
00032     }
00033 }
00038 void CMP_Disable(uint8_t CMP_CHN_x)
00039 {
00040     if (CMP_CHN_x == CMP_CHN_0)
00041     {
00042         SYS0->PROTECT = 0x7a83;
00043         SYS0->AFE_REG5 &= ~BIT6;
00044         SYS0->PROTECT = 0;
00045     }
00046     else
00047     {
00048         __NOP();
00049     }
00050     if (CMP_CHN_x == CMP_CHN_1)
00051     {
00052         SYS0->PROTECT = 0x7a83;
00053         SYS0->AFE_REG5 &= ~BIT7;
00054         SYS0->PROTECT = 0;
00055     }
00056     else
00057     {
00058         __NOP();
00059     }
00060     // 当比较器0和比较器1同时关闭时, 关闭总时钟
00061     if (REG_READ_BIT(SYS0->AFE_REG5, SYS_AFE_REG5_CMP0PDN_BIT | SYS_AFE_REG5_CMP1PDN_BIT) == 0)
00062     {
00063         SYS_ModuleClockCmd(SYS_MODULE_CMP, DISABLE);
00064     }
```

```

00065 }
00072 void CMP_SetSignal(uint8_t CMP_CHN_x, uint8_t CMP_SELN_x, uint8_t CMP_SEL_P_x)
00073 {
00074     if (CMP_CHN_x == CMP_CHN_0)
00075     {
00076         SYS0->PROTECT = 0x7a83;
00077         SYS0->AFE_REG3 = (SYS0->AFE_REG3 & ~(0x0707)) |
00078             (CMP_SEL_P_x << 8) |
00079             (CMP_SELN_x << 4);
00080         SYS0->PROTECT = 0;
00081     }
00082     else
00083     {
00084         __NOP();
00085     }
00086     if (CMP_CHN_x == CMP_CHN_1)
00087     {
00088         SYS0->PROTECT = 0x7a83;
00089         SYS0->AFE_REG3 = (SYS0->AFE_REG3 & ~(0x7070)) |
00090             (CMP_SEL_P_x << 12) |
00091             (CMP_SELN_x << 4);
00092         SYS0->PROTECT = 0;
00093     }
00094     else
00095     {
00096         __NOP();
00097     }
00098 }
00099
00104 void CMP_SetHysteresis(uint8_t CMP_HYS_x)
00105 {
00106     SYS0->PROTECT = 0x7a83;
00107     SYS0->AFE_REG3 = (SYS0->AFE_REG3 & ~(BIT7)) |
00108         (CMP_HYS_x << 7);
00109     SYS0->PROTECT = 0;
00110 }
00111 // 数字部分
00118 void CMP_SetWindowSignal(uint8_t CMP_CHN_x, uint8_t CMP_WIN_x, uint8_t CMP_WIN_LEVEL_x)
00119 {
00120     if (CMP_CHN_x == CMP_CHN_0)
00121     {
00122         CMP->BLCWIN = (CMP->BLCWIN & ~(0xf)) |
00123             (CMP_WIN_x);
00124         CMP->CFG = (CMP->CFG & ~(BIT3)) |
00125             (CMP_WIN_LEVEL_x << 3);
00126     }
00127     else
00128     {
00129         __NOP();
00130     }
00131     if (CMP_CHN_x == CMP_CHN_1)
00132     {
00133         CMP->BLCWIN = (CMP->BLCWIN & ~(0xf0)) |
00134             (CMP_WIN_x << 4);
00135         CMP->CFG = (CMP->CFG & ~(BIT3)) |
00136             (CMP_WIN_LEVEL_x << 7);
00137     }
00138     else
00139     {
00140         __NOP();
00141     }
00142 }
00149 void CMP_SetFilterWidth(uint8_t CMP_CLK_DIV_x, uint16_t CMP0_FIL_x, uint16_t CMP1_FIL_x)
00150 {
00151     uint16_t div2_0 = 0;
00152     uint16_t div16_0 = 0;
00153     uint16_t div2_1 = 0;
00154     uint16_t div16_1 = 0;
00155     uint16_t flten_0 = 0;
00156     uint16_t flten_1 = 0;
00157     if (CMP0_FIL_x > 2047)
00158     {
00159         CMP0_FIL_x = 2047;
00160     }
00161     if (CMP1_FIL_x > 2047)
00162     {
00163         CMP1_FIL_x = 2047;
00164     }
00165     for (; CMP0_FIL_x > 15;)
00166     {
00167         div2_0++;
00168         CMP0_FIL_x >>= 1;
00169     }
00170     div16_0 = CMP0_FIL_x;
00171     flten_0 = CMP0_FIL_x ? 1 : 0;
00172
00173     for (; CMP1_FIL_x > 15;)

```

```

00174     {
00175         div2_1++;
00176         CMP1_FIL_x >= 1;
00177     }
00178     div16_1 = CMP1_FIL_x;
00179     flten_1 = CMP1_FIL_x ? 1 : 0;
00180
00181     CMP->TCLK = (CMP_CLK_DIV_x << 24) |
00182                 (div16_1 << 12) |
00183                 (flten_1 << 11) |
00184                 (div2_1 << 8) |
00185                 (div16_0 << 4) |
00186                 (flten_0 << 3) |
00187                 (div2_0);
00188 }
00189
00195 void CMP_SetIRQSignal(uint8_t CMP_CHN_x, uint8_t CMP_SIGNAL_x)
00196 {
00197     if (CMP_CHN_x == CMP_CHN_0)
00198     {
00199         if (CMP_SIGNAL_x == CMP_SIGNAL_NONE)
00200         {
00201             CMP0->IE &= ~BIT0;
00202         }
00203         else
00204         {
00205             CMP0->IE |= BIT0;
00206         }
00207         CMP0->CFG = (CMP0->CFG & ~(0x7)) |
00208                     (CMP_SIGNAL_x);
00209     }
00210     else
00211     {
00212         __NOP();
00213     }
00214     if (CMP_CHN_x == CMP_CHN_1)
00215     {
00216         if (CMP_SIGNAL_x == CMP_SIGNAL_NONE)
00217         {
00218             CMP0->IE &= ~BIT1;
00219         }
00220         else
00221         {
00222             CMP0->IE |= BIT1;
00223         }
00224         CMP0->CFG = (CMP0->CFG & ~(0x70)) |
00225                     (CMP_SIGNAL_x << 4);
00226     }
00227     else
00228     {
00229         __NOP();
00230     }
00231 }
00236 void CMP_ClearIRQFlag(uint8_t CMP_CHN_x)
00237 {
00238     if (CMP_CHN_x == CMP_CHN_0)
00239     {
00240         CMP0->IF = BIT0;
00241     }
00242     else
00243     {
00244         __NOP();
00245     }
00246     if (CMP_CHN_x == CMP_CHN_1)
00247     {
00248         CMP0->IF = BIT1;
00249     }
00250     else
00251     {
00252         __NOP();
00253     }
00254 }
00255
00261 uint8_t CMP_GetIRQFlag(uint8_t CMP_CHN_x)
00262 {
00263     if (CMP_CHN_x == CMP_CHN_0)
00264     {
00265         return CMP0->IF & BIT0;
00266     }
00267     else if (CMP_CHN_x == CMP_CHN_1)
00268     {
00269         return (CMP0->IF & BIT1) >> 1;
00270     }
00271     else
00272     {
00273         return 0;
00274     }

```

```

00275 }
00276
00281 uint8_t CMP_GetFilterValue(uint8_t CMP_CHN_x)
00282 {
00283     if (CMP_CHN_x == CMP_CHN_0)
00284     {
00285         return (CMP0->DATA & BIT8) >> 8;
00286     }
00287     else if (CMP_CHN_x == CMP_CHN_1)
00288     {
00289         return (CMP0->DATA & BIT9) >> 9;
00290     }
00291     else
00292     {
00293         return 0;
00294     }
00295 }
00296
00301 uint8_t CMP_GetValue(uint8_t CMP_CHN_x)
00302 {
00303     if (CMP_CHN_x == CMP_CHN_0)
00304     {
00305         return CMP0->DATA & BIT0;
00306     }
00307     else if (CMP_CHN_x == CMP_CHN_1)
00308     {
00309         return (CMP0->DATA & BIT1) >> 1;
00310     }
00311     else
00312     {
00313         return 0;
00314     }
00315 }
00320 void CMP_StructInit(CMP_InitTypeDef *CMP_InitStruct)
00321 {
00322     uint32_t i = 0;
00323     for (i = 0; i < sizeof(CMP_InitTypeDef); i++)
00324     {
00325         ((uint8_t *)CMP_InitStruct)[i] = 0;
00326     }
00327 }
00332 void CMP_Init(CMP_InitTypeDef *CMP_InitStruct)
00333 {
00334     if (CMP_InitStruct->CMP0_CFG.EN)
00335     {
00336         CMP_Enable(CMP_CHN_0);
00337     }
00338     else
00339     {
00340         CMP_Disable(CMP_CHN_0);
00341     }
00342     if (CMP_InitStruct->CMP1_CFG.EN)
00343     {
00344         CMP_Enable(CMP_CHN_1);
00345     }
00346     else
00347     {
00348         CMP_Disable(CMP_CHN_1);
00349     }
00350     CMP_SetHysteresis(CMP_InitStruct->HYS);
00351     {
00352         REG_WRITE_BIT(CMP0->TCLK, CMP_TCLK.COMM.DIV_MASK, CMP_TCLK.COMM.DIV(0));
00353         REG_SET(CMP0->TCLK, CMP_TCLK.CLK0.EN.BIT | CMP_TCLK.CLK1.EN.BIT);
00354         REG_WRITE_BIT(CMP0->TCLK, CMP_TCLK.FIL0.CLK.DIV16_MASK,
00355             CMP_TCLK.FIL0.CLK.DIV16(CMP_InitStruct->CMP0_CFG.FIL.CLK.DIV16));
00356         REG_WRITE_BIT(CMP0->TCLK, CMP_TCLK.FIL0.CLK.DIV2_MASK,
00357             CMP_TCLK.FIL0.CLK.DIV2(CMP_InitStruct->CMP0_CFG.FIL.CLK.DIV2));
00358         REG_WRITE_BIT(CMP0->TCLK, CMP_TCLK.FIL1.CLK.DIV16_MASK,
00359             CMP_TCLK.FIL1.CLK.DIV16(CMP_InitStruct->CMP1_CFG.FIL.CLK.DIV16));
00360         REG_WRITE_BIT(CMP0->TCLK, CMP_TCLK.FIL1.CLK.DIV2_MASK,
00361             CMP_TCLK.FIL1.CLK.DIV2(CMP_InitStruct->CMP1_CFG.FIL.CLK.DIV2));
00362     }
00363     {
00364         uint32_t val =
00365             (CMP_InitStruct->CMP0_CFG.CHN0P.PWM.EN << 0) |
00366             (CMP_InitStruct->CMP0_CFG.CHN1P.PWM.EN << 1) |
00367             (CMP_InitStruct->CMP0_CFG.CHN2P.PWM.EN << 2) |
00368             (CMP_InitStruct->CMP0_CFG.CHN3P.PWM.EN << 3);
00369         CMP_SetWindowSignal(CMP_CHN_0, val, CMP_InitStruct->CMP0_CFG.W.PWM.POL);
00370     }
00371     {
00372         uint32_t val =
00373             (CMP_InitStruct->CMP1_CFG.CHN0P.PWM.EN << 0) |

```

```

00374         (CMP_InitStruct->CMP1_CFG.CHN1P_PWM_EN << 1) |
00375         (CMP_InitStruct->CMP1_CFG.CHN2P_PWM_EN << 2) |
00376         (CMP_InitStruct->CMP1_CFG.CHN3P_PWM_EN << 3);
00377     CMP_SetWindowSignal(CMP_CHN1, val, CMP_InitStruct->CMP1_CFG.W_PWM_POL);
00378 }
00379
00380 {
00381     uint16_t irqSignal = 0;
00382     irqSignal |= (CMP_InitStruct->CMP0_CFG.POL) ? BIT0 : 0; // 极性位
00383     irqSignal |= (CMP_InitStruct->CMP0_CFG.IE) ? BIT1 : 0; // 中断使能位
00384     irqSignal |= (CMP_InitStruct->CMP0_CFG.IRQ_TRIG) ? BIT2 : 0; // 触发方式位
00385
00386     CMP_SetIRQSignal(CMP_CHN0, irqSignal);
00387 }
00388 {
00389     uint16_t irqSignal = 0;
00390     irqSignal |= (CMP_InitStruct->CMP1_CFG.POL) ? BIT0 : 0; // 极性位
00391     irqSignal |= (CMP_InitStruct->CMP1_CFG.IE) ? BIT1 : 0; // 中断使能位
00392     irqSignal |= (CMP_InitStruct->CMP1_CFG.IRQ_TRIG) ? BIT2 : 0; // 触发方式位
00393
00394     CMP_SetIRQSignal(CMP_CHN1, irqSignal);
00395 }
00396
00397 if (CMP_InitStruct->CMP0_CFG.RE)
00398 {
00399     CMP0->IE |= BIT8;
00400 }
00401 if (CMP_InitStruct->CMP1_CFG.RE)
00402 {
00403     CMP0->IE |= BIT9;
00404 }
00405 }

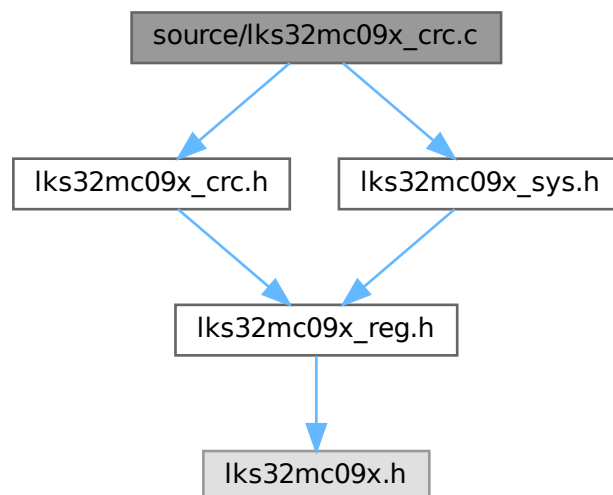
```

3.61 source/lks32mc09x_crc.c 文件参考

```

#include "lks32mc09x_crc.h"
#include "lks32mc09x_sys.h"
lks32mc09x_crc.c 的引用(Include)关系图:

```



函数

- void **CRC_Init** (CRC_TypeDef *CRCx, CRC_InitTypeDef *CRC_InitStruct)

- CRC计算初始化
- void `CRC_StructInit` (`CRC_InitTypeDef` *`CRC_InitStructure`)
初始化CRC结构体为默认值
 - uint32_t `CRC_Calc` (`CRC_TypeDef` *`CRCx`, uint8_t *`data`, uint32_t `len`)
计算CRC值
 - uint32_t `CRC_CalcCommon` (`CRC_TypeDef` *`CRCx`, uint8_t *`data`, uint32_t `len`, uint16_t `CRC_MODEL_x`)
使用常见模型计算CRC值
 - void `CRC_Enable` (`CRC_TypeDef` *`CRCx`)
使能CRC模块
 - void `CRC_Disable` (`CRC_TypeDef` *`CRCx`)
禁用CRC模块

变量

- static uint32_t `CRC_XOROUT` = 0
初始化CRC模块

3.61.1 函数说明

3.61.1.1 CRC_Calc()

```
uint32_t CRC_Calc (  
    CRC_TypeDef * CRCx,  
    uint8_t * data,  
    uint32_t len )
```

计算CRC值

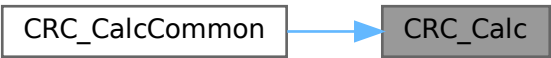
参数

<i>CRCx</i>	CRC模块指针
<i>data</i>	数据指针
<i>len</i>	数据长度

返回

计算得到的CRC值

在文件 `lks32mc09x_crc.c` 第 79 行定义.
这是这个函数的调用关系图:



3.61.1.2 CRC_CalcCommon()

```
uint32_t CRC_CalcCommon (
    CRC_TypeDef * CRCx,
    uint8_t * data,
    uint32_t len,
    uint16_t CRC_MODEL_x )
```

使用常见模型计算CRC值

参数

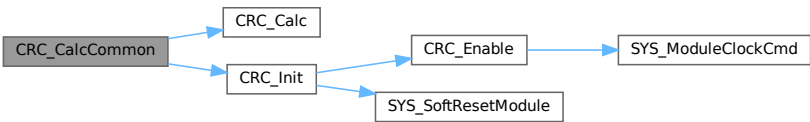
CRCx	CRC模块指针
data	数据指针
len	数据长度
CRC_MODEL_x	CRC模型

返回

计算得到的CRC值

在文件 lks32mc09x_crc.c 第 106 行定义.

函数调用图:



3.61.1.3 CRC_Disable()

```
void CRC_Disable (
    CRC_TypeDef * CRCx )
```

禁用CRC模块

参数

CRCx	CRC模块指针

在文件 lks32mc09x_crc.c 第 304 行定义.

函数调用图:



3.61.1.4 CRC_Enable()

```
void CRC_Enable (  
    CRC_TypeDef * CRCx )
```

使能CRC模块

参数

CRCx	CRC模块指针

在文件 [lks32mc09x_crc.c](#) 第 288 行定义.

函数调用图:



这是这个函数的调用关系图:



3.61.1.5 CRC_Init()

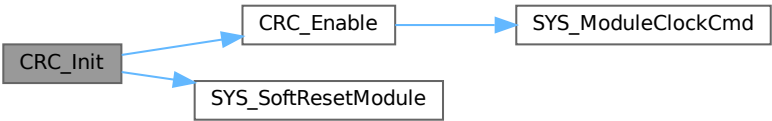
```
void CRC_Init (
    CRC_TypeDef * CRCx,
    CRC_InitTypeDef * CRC_InitStruct )
```

CRC计算初始化

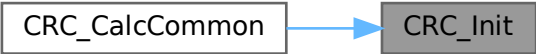
初始化CRC模块

在文件 [lks32mc09x_crc.c](#) 第 14 行定义.

函数调用图:



这是这个函数的调用关系图:



3.61.1.6 CRC_StructInit()

```
void CRC_StructInit (
    CRC_InitTypeDef * CRC_InitStruct )
```

初始化CRC结构体为默认值

参数

<i>CRC_InitStruct</i>	CRC初始化结构体指针

在文件 [lks32mc09x_crc.c](#) 第 61 行定义.

3.61.2 变量说明

3.61.2.1 CRC_XOROUT

```
uint32_t CRC_XOROUT = 0 [static]
```

初始化CRC模块

参数

<i>CRCx</i>	CRC模块指针
<i>CRC_InitStruct</i>	CRC初始化结构体指针

在文件 [lks32mc09x_crc.c](#) 第 9 行定义.

3.62 lks32mc09x_crc.c

[浏览该文件的文档.](#)

```
00001 #include "lks32mc09x_crc.h"
00002 #include "lks32mc09x_sys.h"
00003
00009 static uint32_t CRC_XOROUT = 0;
00010
00014 void CRC_Init(CRC_TypeDef *CRCx, CRC_InitTypeDef *CRC_InitStruct)
00015 {
00016     if (CRC_InitStruct->WIDTH > 3)
00017     {
00018         switch (CRC_InitStruct->WIDTH)
00019         {
00020             case 7:
00021                 CRC_InitStruct->WIDTH = CRC_WIDTH_7;
00022                 break;
00023             case 8:
00024                 CRC_InitStruct->WIDTH = CRC_WIDTH_8;
00025                 break;
00026             case 16:
00027                 CRC_InitStruct->WIDTH = CRC_WIDTH_16;
00028                 break;
00029             case 32:
00030                 CRC_InitStruct->WIDTH = CRC_WIDTH_32;
00031                 break;
00032             default:
00033                 CRC_InitStruct->WIDTH = CRC_WIDTH_8;
00034                 break;
00035         }
00036     }
00037     else
00038     {
00039         __NOP();
00040     }
00041     CRC_Enable(CRCx);
00042     SYS_SoftResetModule(SYS_MODULE_CRC0);
00043     CRC0_CR = (u32)(CRC_InitStruct->REFOUT << 12) | (u32)(CRC_InitStruct->REFIN << 8) |
00044     (u32)(CRC_InitStruct->WIDTH << 4);
00045     CRC0_CR |= BIT0; // 复位
00046     CRC0_POL = CRC_InitStruct->POLY;
00047     CRC0_INIT = CRC_InitStruct->INIT;
00048     if (CRC_InitStruct->WIDTH == CRC_WIDTH_16)
00049     {
00050         CRC_XOROUT = CRC_InitStruct->XOROUT << 16;
00051     }
00052     else
00053     {
00054         CRC_XOROUT = CRC_InitStruct->XOROUT;
00055     }
00056 }
00061 void CRC_StructInit(CRC_InitTypeDef *CRC_InitStruct)
00062 {
00063     uint32_t i;
```

```

00064     for (i = 0; i < sizeof(CRC_InitTypeDef); i++)
00065     {
00066         ((uint8_t *)CRC_InitStruct)[i] = 0;
00067     }
00068     CRC_InitStruct->POLY = 7; // 多项式
00069     CRC_InitStruct->WIDTH = CRC_WIDTH_8; // 渝藉害
00070 }
00071
00079 uint32_t CRC_Calc(CRC_TypeDef *CRCx, uint8_t *data, uint32_t len)
00080 {
00081     u32 val;
00082     if (len == 0)
00083     {
00084         return 0;
00085     }
00086     else
00087     {
00088         while ((len--) != 0)
00089         {
00090             REG8(&CRCx->DR) = *data;
00091             data++;
00092         }
00093     }
00094     val = (REG32(&CRCx->DR)) ^ CRC_XOROUT;
00095     return val;
00096 }
00097
00106 uint32_t CRC_CalcCommon(CRC_TypeDef *CRCx, uint8_t *data, uint32_t len, uint16_t CRC_MODEL_x)
00107 {
00108     u32 val;
00109     CRC_InitTypeDef CRC_InitStruct;
00110     switch (CRC_MODEL_x)
00111     {
00112         case CRC_MODEL_CRC_8: // 8 07 00 00 false false
00113             CRC_InitStruct.WIDTH = CRC_WIDTH_8; // 渝藉害
00114             CRC_InitStruct.POLY = 0x07; // 多项式
00115             CRC_InitStruct.INIT = 0x00; // 初始值
00116             CRC_InitStruct.XOROUT = 0x00; // 结果异或值
00117             CRC_InitStruct.REFIN = DISABLE; // 输入数据反转
00118             CRC_InitStruct.REFOUT = DISABLE; // 输出数据反转
00119             CRC_Init(CRCx, &CRC_InitStruct);
00120             break;
00121         case CRC_MODEL_CRC_8_ITU: // 8 07 00 55 false
00122             CRC_InitStruct.WIDTH = CRC_WIDTH_8; // 渝藉害
00123             CRC_InitStruct.POLY = 0x07; // 多项式
00124             CRC_InitStruct.INIT = 0x00; // 初始值
00125             CRC_InitStruct.XOROUT = 0x55; // 结果异或值
00126             CRC_InitStruct.REFIN = DISABLE; // 输入数据反转
00127             CRC_InitStruct.REFOUT = DISABLE; // 输出数据反转
00128             CRC_Init(CRCx, &CRC_InitStruct);
00129             break;
00130         case CRC_MODEL_CRC_8_ROHC: // 8 07 FF 00 true true
00131             CRC_InitStruct.WIDTH = CRC_WIDTH_8; // 渝藉害
00132             CRC_InitStruct.POLY = 0x07; // 多项式
00133             CRC_InitStruct.INIT = 0xff; // 初始值
00134             CRC_InitStruct.XOROUT = 0x00; // 结果异或值
00135             CRC_InitStruct.REFIN = ENABLE; // 输入数据反转
00136             CRC_InitStruct.REFOUT = ENABLE; // 输出数据反转
00137             CRC_Init(CRCx, &CRC_InitStruct);
00138             break;
00139         case CRC_MODEL_CRC_8_MAXIMx: // 8 31 00 00 true true
00140             CRC_InitStruct.WIDTH = CRC_WIDTH_8; // 渝藉害
00141             CRC_InitStruct.POLY = 0x31; // 多项式
00142             CRC_InitStruct.INIT = 0x00; // 初始值
00143             CRC_InitStruct.XOROUT = 0x00; // 结果异或值
00144             CRC_InitStruct.REFIN = ENABLE; // 输入数据反转
00145             CRC_InitStruct.REFOUT = ENABLE; // 输出数据反转
00146             CRC_Init(CRCx, &CRC_InitStruct);
00147             break;
00148         case CRC_MODEL_CRC_16_IBMx1: // 16 8005 0000 0000 true true
00149             CRC_InitStruct.WIDTH = CRC_WIDTH_16; // 渝藉害
00150             CRC_InitStruct.POLY = 0x8005; // 多项式
00151             CRC_InitStruct.INIT = 0x0000; // 初始值
00152             CRC_InitStruct.XOROUT = 0x0000; // 结果异或值
00153             CRC_InitStruct.REFIN = ENABLE; // 输入数据反转
00154             CRC_InitStruct.REFOUT = ENABLE; // 输出数据反转
00155             CRC_Init(CRCx, &CRC_InitStruct);
00156             break;
00157         case CRC_MODEL_CRC_16_MAXIM: // 16 8005 0000 FFFF true true
00158             CRC_InitStruct.WIDTH = CRC_WIDTH_16; // 渝藉害
00159             CRC_InitStruct.POLY = 0x8005; // 多项式

```

```

00165         CRC_InitStruct.INIT = 0x0000; // 初始值
00166         CRC_InitStruct.XOROUT = 0xffff; // 结果异或值
00167         CRC_InitStruct.REFIN = ENABLE; // 输入数据反转
00168         CRC_InitStruct.REFOUT = ENABLE; // 输出数据反转
00169         CRC_Init(CRCx, &CRC_InitStruct);
00170         break;
00171     case CRC_MODEL_CRC_16_USBx1: // 16 8005 FFFF FFFF true true
00172
00173         CRC_InitStruct.WIDTH = CRC_WIDTH_16; // 瀚稽害
00174         CRC_InitStruct.POLY = 0x8005; // 多项式
00175         CRC_InitStruct.INIT = 0xffff; // 初始值
00176         CRC_InitStruct.XOROUT = 0xffff; // 结果异或值
00177         CRC_InitStruct.REFIN = ENABLE; // 输入数据反转
00178         CRC_InitStruct.REFOUT = ENABLE; // 输出数据反转
00179         CRC_Init(CRCx, &CRC_InitStruct);
00180         break;
00181     case CRC_MODEL_CRC_16_MODBU: // 16 8005 FFFF 0000 true true
00182
00183         CRC_InitStruct.WIDTH = CRC_WIDTH_16; // 瀚稽害
00184         CRC_InitStruct.POLY = 0x8005; // 多项式
00185         CRC_InitStruct.INIT = 0xffff; // 初始值
00186         CRC_InitStruct.XOROUT = 0x0000; // 结果异或值
00187         CRC_InitStruct.REFIN = ENABLE; // 输入数据反转
00188         CRC_InitStruct.REFOUT = ENABLE; // 输出数据反转
00189         CRC_Init(CRCx, &CRC_InitStruct);
00190         break;
00191     case CRC_MODEL_CRC_16_CCITT: // 16 1021 0000 0000 true true
00192
00193         CRC_InitStruct.WIDTH = CRC_WIDTH_16; // 瀚稽害
00194         CRC_InitStruct.POLY = 0x1021; // 多项式
00195         CRC_InitStruct.INIT = 0x0000; // 初始值
00196         CRC_InitStruct.XOROUT = 0x0000; // 结果异或值
00197         CRC_InitStruct.REFIN = ENABLE; // 输入数据反转
00198         CRC_InitStruct.REFOUT = ENABLE; // 输出数据反转
00199         CRC_Init(CRCx, &CRC_InitStruct);
00200         break;
00201     case CRC_MODEL_CRC_16_CCITT_FALSE: // 16 1021 FFFF 0000 false false
00202
00203         CRC_InitStruct.WIDTH = CRC_WIDTH_16; // 瀚稽害
00204         CRC_InitStruct.POLY = 0x1021; // 多项式
00205         CRC_InitStruct.INIT = 0xffff; // 初始值
00206         CRC_InitStruct.XOROUT = 0x0000; // 结果异或值
00207         CRC_InitStruct.REFIN = DISABLE; // 输入数据反转
00208         CRC_InitStruct.REFOUT = DISABLE; // 输出数据反转
00209         CRC_Init(CRCx, &CRC_InitStruct);
00210         break;
00211     case CRC_MODEL_CRC_16_X25: // 16 1021 FFFF FFFF true true
00212
00213         CRC_InitStruct.WIDTH = CRC_WIDTH_16; // 瀚稽害
00214         CRC_InitStruct.POLY = 0x1021; // 多项式
00215         CRC_InitStruct.INIT = 0xffff; // 初始值
00216         CRC_InitStruct.XOROUT = 0xffff; // 结果异或值
00217         CRC_InitStruct.REFIN = ENABLE; // 输入数据反转
00218         CRC_InitStruct.REFOUT = ENABLE; // 输出数据反转
00219         CRC_Init(CRCx, &CRC_InitStruct);
00220         break;
00221     case CRC_MODEL_CRC_16_XMODEM: // 16 1021 0000 0000 false false
00222
00223         CRC_InitStruct.WIDTH = CRC_WIDTH_16; // 瀚稽害
00224         CRC_InitStruct.POLY = 0x1021; // 多项式
00225         CRC_InitStruct.INIT = 0x0000; // 初始值
00226         CRC_InitStruct.XOROUT = 0x0000; // 结果异或值
00227         CRC_InitStruct.REFIN = DISABLE; // 输入数据反转
00228         CRC_InitStruct.REFOUT = DISABLE; // 输出数据反转
00229         CRC_Init(CRCx, &CRC_InitStruct);
00230         break;
00231     case CRC_MODEL_CRC_16_DNP: // 16 3D65 0000 FFFF true true
00232
00233         CRC_InitStruct.WIDTH = CRC_WIDTH_16; // 瀚稽害
00234         CRC_InitStruct.POLY = 0x3d65; // 多项式
00235         CRC_InitStruct.INIT = 0x0000; // 初始值
00236         CRC_InitStruct.XOROUT = 0xffff; // 结果异或值
00237         CRC_InitStruct.REFIN = ENABLE; // 输入数据反转
00238         CRC_InitStruct.REFOUT = ENABLE; // 输出数据反转
00239         CRC_Init(CRCx, &CRC_InitStruct);
00240         break;
00241     case CRC_MODEL_CRC_32: // 32 04C11DB7 FFFFFFFF FFFFFFFF true true
00242
00243         CRC_InitStruct.WIDTH = CRC_WIDTH_32; // 瀚稽害
00244         CRC_InitStruct.POLY = 0x04C11DB7; // 多项式
00245         CRC_InitStruct.INIT = 0xFFFFFFFF; // 初始值
00246         CRC_InitStruct.XOROUT = 0xFFFFFFFF; // 结果异或值
00247         CRC_InitStruct.REFIN = ENABLE; // 输入数据反转
00248         CRC_InitStruct.REFOUT = ENABLE; // 输出数据反转
00249         CRC_Init(CRCx, &CRC_InitStruct);
00250         break;
00251     case CRC_MODEL_CRC_32_MPEG2: // 32 04C11DB7 FFFFFFFF 00000000 false false

```

```

00252
00253     CRC_InitStruct.WIDTH = CRC_WIDTH_32; // 渝藉害
00254     CRC_InitStruct.POLY  = 0x04C11DB7; // 多项式
00255     CRC_InitStruct.INIT  = 0xFFFFFFFF; // 初始值
00256     CRC_InitStruct.XOROUT = 0x00000000; // 结果异或值
00257     CRC_InitStruct.REFIN  = DISABLE; // 输入数据反转
00258     CRC_InitStruct.REFOUT = DISABLE; // 输出数据反转
00259     CRC_Init(CRCx, &CRC_InitStruct);
00260     break;
00261 }
00262 val = CRC_Calc(CRCx, data, len);
00263 if (CRC_InitStruct.REFOUT)
00264 {
00265     switch (CRC_InitStruct.WIDTH)
00266     {
00267         case CRC_WIDTH_8:
00268             val >>= 24;
00269             break;
00270         case CRC_WIDTH_16:
00271             val >>= 16;
00272             break;
00273         default:
00274             break;
00275     }
00276 }
00277 else
00278 {
00279     __NOP();
00280 }
00281 return val;
00282 }
00283
00288 void CRC_Enable(CRC_TypeDef *CRCx)
00289 {
00290     if (CRCx == CRC0)
00291     {
00292         SYS_ModuleClockCmd(SYS_MODULE_CRC0, ENABLE);
00293     }
00294     else
00295     {
00296         __NOP();
00297     }
00298 }
00299
00304 void CRC_Disable(CRC_TypeDef *CRCx)
00305 {
00306     if (CRCx == CRC0)
00307     {
00308         SYS_ModuleClockCmd(SYS_MODULE_CRC0, DISABLE);
00309     }
00310     else
00311     {
00312         __NOP();
00313     }
00314 }

```

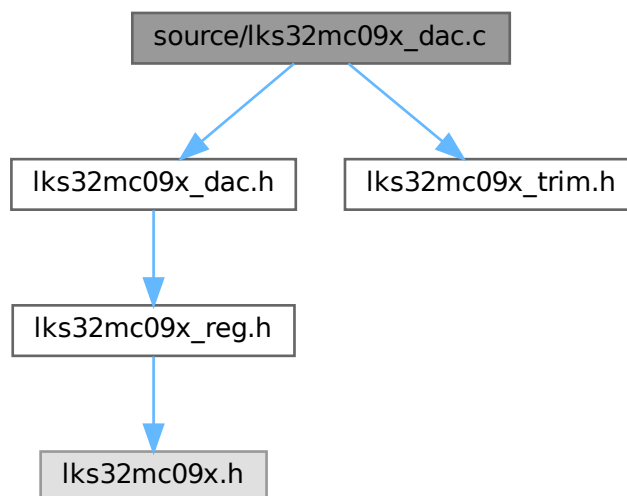
3.63 source/lks32mc09x_dac.c 文件参考

```

#include "lks32mc09x_dac.h"
#include "lks32mc09x_trim.h"

```

lks32mc09x_dac.c 的引用(Include)关系图:



函数

- void **DAC_Init** (uint8_t DAC_Channel, **DAC_InitTypeDef** *DAC_InitStruct)
DAC初始化
- void **DAC_StructInit** (**DAC_InitTypeDef** *DAC_InitStruct)
DAC初始化结构体初始化
- void **DAC_OutputValue** (uint8_t DAC_Channel, uint32_t DACValue)
DAC输出数值设置 (数字量)
- void **DAC_SetRange** (uint8_t DAC_Channel, uint32_t DAC_RANGE_x)
设置DAC量程
- void **DAC_OutputVoltage** (uint8_t DAC_Channel, uint16_t DACVoltage)
DAC输出模拟量, 自动配置量程&同时自动使能对应的模块
- void **DAC_Cmd** (uint8_t DAC_Channel, uint8_t state)
DAC通道时钟使能
- void **DAC_Enable** (uint8_t DAC_Channel)
DAC使能
- void **DAC_Disable** (uint8_t DAC_Channel)
DAC失能
- void **DAC_Reset** (uint8_t DAC_Channel)
DAC复位
- void **DAC_Output** (uint8_t DAC_Channel, uint8_t state)
DAC输出至IO口使能配置
- uint32_t **DAC.Value** (uint8_t DAC_Channel)
获取当前DAC输出值
- uint16_t **DAC.Voltage** (uint8_t DAC_Channel)
获取当前DAC输出电压
- uint8_t **DAC.Range** (uint8_t DAC_Channel)
获取当前DAC量程
- void **DAC.Step** (uint8_t DAC_Channel, uint8_t DAC_STEP_x, uint8_t DAC_STEP_TRG_x)
DAC0步进设置

3.63.1 函数说明

3.63.1.1 DAC_Cmd()

```
void DAC_Cmd (
    uint8_t DAC_Channel,
    uint8_t state )
```

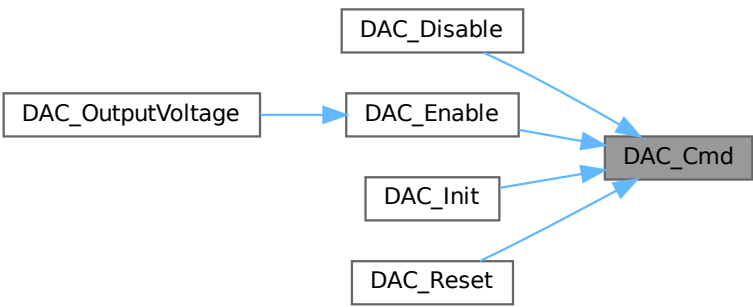
DAC通道时钟使能

参数

DAC_Channel	DAC通道选择
state	使能状态，1模块使能 0 模块关闭

在文件 [lks32mc09x_dac.c](#) 第 151 行定义.

这是这个函数的调用关系图:



3.63.1.2 DAC_Disable()

```
void DAC_Disable (
    uint8_t DAC_Channel )
```

DAC失能

参数

DAC_Channel	DAC通道选择

在文件 [lks32mc09x_dac.c](#) 第 199 行定义.

函数调用图:



3.63.1.3 DAC_Enable()

```
void DAC_Enable (
    uint8_t DAC_Channel )
```

DAC使能

参数

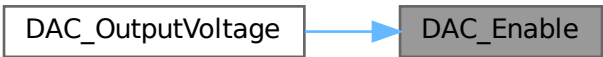
DAC_Channel	DAC通道选择

在文件 [lks32mc09x-dac.c](#) 第 191 行定义.

函数调用图:



这是这个函数的调用关系图:



3.63.1.4 DAC_Init()

```
void DAC_Init (
    uint8_t DAC_Channel,
    DAC_InitTypeDef * DAC_InitStruct )
```

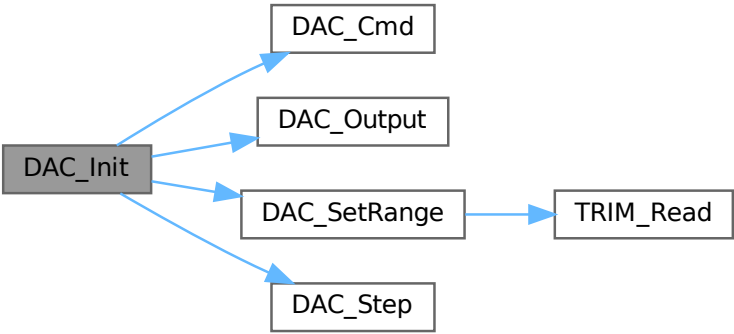
DAC初始化

参数

DAC_Channel	DAC通道选择
DAC_InitStruct	DAC初始化结构体

在文件 [lks32mc09x_dac.c](#) 第 9 行定义.

函数调用图:



3.63.1.5 DAC_Output()

```
void DAC_Output (
    uint8_t DAC_Channel,
    uint8_t state )
```

DAC输出至IO口使能配置

参数

DAC_Channel	DAC通道选择
state	1Dac电压p0.0口 0停止输出到P0.0

在文件 [lks32mc09x_dac.c](#) 第 218 行定义.

这是这个函数的调用关系图:



3.63.1.6 DAC_OutputValue()

```
void DAC_OutputValue (
    uint8_t DAC_Channel,
    uint32_t DACValue )
```

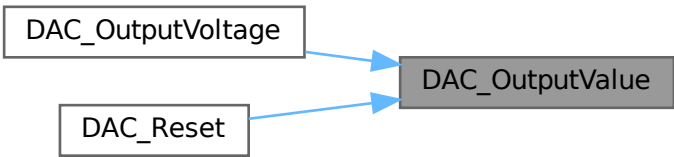
DAC输出数值设置（数字量）

参数

DAC_Channel	DAC通道选择
DACValue	DAC输出数字量

在文件 [lks32mc09x_dac.c](#) 第 33 行定义.

这是这个函数的调用关系图:



3.63.1.7 DAC_OutputVoltage()

```
void DAC_OutputVoltage (
    uint8_t DAC_Channel,
    uint16_t DACVoltage )
```

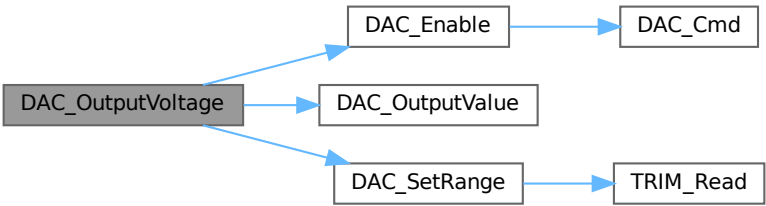
DAC输出模拟量，自动配置量程&同时自动使能对应的模块

参数

<i>DAC_Channel</i>	DAC通道选择
<i>DACVoltage</i>	DAC输出模拟量 单位mV

在文件 [lks32mc09x_dac.c](#) 第 106 行定义.

函数调用图:



3.63.1.8 DAC.Range()

```
uint8_t DACRange (
    uint8_t DAC_Channel )
```

获取当前DAC量程

参数

<i>DAC_Channel</i>	DAC通道选择

返回

DAC量程 dac0 0:4.85V 1:1.2V dac1 0:3V 1:1.2V

在文件 [lks32mc09x_dac.c](#) 第 310 行定义.

这是这个函数的调用关系图:



3.63.1.9 DAC_Reset()

```
void DAC_Reset (
    uint8_t DAC_Channel )
```

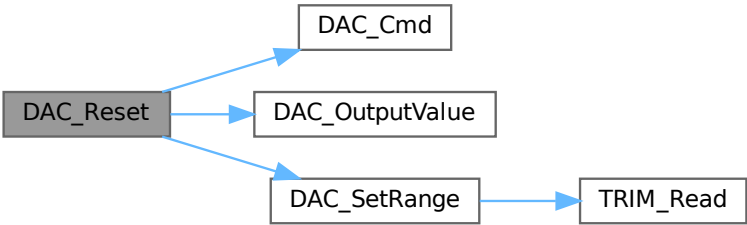
DAC复位

参数

DAC_Channel	DAC通道选择

在文件 [lks32mc09x_dac.c](#) 第 207 行定义.

函数调用图:



3.63.1.10 DAC.SetRange()

```
void DAC_SetRange (
    uint8_t DAC_Channel,
    uint32_t DAC_RANGE_x )
```

设置DAC量程

参数

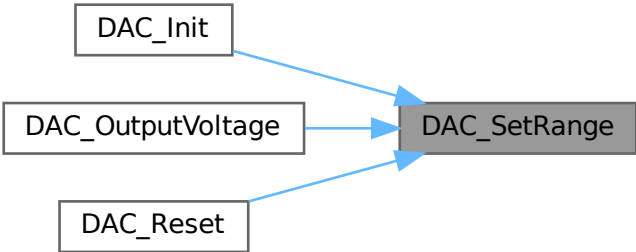
DAC_Channel	DAC通道选择
DAC↔ RANGE_x	DAC输出量程

在文件 [lks32mc09x_dac.c](#) 第 61 行定义.

函数调用图:



这是这个函数的调用关系图:



3.63.1.11 DAC.Step()

```
void DAC_Step (
    uint8_t DAC_Channel,
    uint8_t DAC_STEP_x,
    uint8_t DAC_STEP_TRG_x )
```

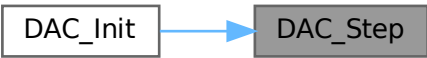
DAC0步进设置

参数

DAC_Channel	DAC通道选择
DAC_STEP_x	DAC0步进值
DAC_STEP_↔ TRG_x	DAC0步进触发源

在文件 [lks32mc09x_dac.c](#) 第 346 行定义.

这是这个函数的调用关系图:



3.63.1.12 DAC_StructInit()

```
void DAC_StructInit (
    DAC_InitTypeDef * DAC_InitStruct )
```

DAC初始化结构体初始化

参数

DAC_InitStruct	DAC初始化结构体

在文件 [lks32mc09x-dac.c](#) 第 20 行定义.

3.63.1.13 DAC_Value()

```
uint32_t DAC_Value (
    uint8_t DAC_Channel )
```

获取当前DAC输出值

参数

DAC_Channel	DAC通道选择

返回

DAC输出值

在文件 [lks32mc09x_dac.c](#) 第 254 行定义.

这是这个函数的调用关系图:



3.63.1.14 DAC.Voltage()

```
uint16_t DAC.Voltage (
    uint8_t DAC_Channel )
```

获取当前DAC输出电压

参数

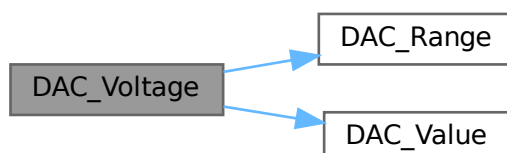
<i>DAC_Channel</i>	DAC通道选择

返回

DAC输出电压 单位mV

在文件 [lks32mc09x_dac.c](#) 第 270 行定义.

函数调用图:



3.64 lks32mc09x_dac.c

[浏览该文件的文档.](#)

```

00001 #include "lks32mc09x_dac.h"
00002 #include "lks32mc09x_trim.h"
00003
00009 void DAC_Init(uint8_t DAC_Channel, DAC_InitTypeDef *DAC_InitStruct)
00010 {
00011     DAC_Cmd(DAC_Channel, ENABLE);
00012     DAC_SetRange(DAC_Channel, DAC_InitStruct->DAC_RANGE);
00013     DAC_Step(DAC_Channel, DAC_InitStruct->DAC_STEP, DAC_InitStruct->DAC_STEP_TRG_x);
00014     DAC_Output(DAC_Channel, DAC_InitStruct->DACOUT_EN);
00015 }
00020 void DAC_StructInit(DAC_InitTypeDef *DAC_InitStruct)
00021 {
00022     uint32_t i;
00023     for (i = 0; i < sizeof(DAC_InitTypeDef); i++)
00024     {
00025         ((uint8_t *)DAC_InitStruct)[i] = 0;
00026     }
00027 }
00033 void DAC_OutputValue(uint8_t DAC_Channel, uint32_t DACValue)
00034 {
00035     if (DACValue > 4095)
00036     {
00037         DACValue = 4095;
00038     }
00039     else
00040     {
00041         __NOP();
00042     }
00043     if (DAC_Channel == DAC_CHANNEL_0)
00044     {
00045         SYS0->AFE_DAC0 = DACValue;
00046     }
00047     else if (DAC_Channel == DAC_CHANNEL_1)
00048     {
00049         SYS0->AFE_DAC1 = DACValue >> 4; // DAC1本身是8位, DACValue是12位, 这里右移4位, 以方便使用
00050     }
00051     else
00052     {
00053         __NOP();
00054     }
00055 }
00061 void DAC_SetRange(uint8_t DAC_Channel, uint32_t DAC_RANGE_x)
00062 {
00063     SYS0->PROTECT = 0x7a83;
00064     if (DAC_Channel == DAC_CHANNEL_0)
00065     {
00066         if (DAC_RANGE_x == DAC_RANGE_0_1V2)
00067         {
00068             SYS0->AFE_REG1 |= BIT6;
00069             SYS0->AFE_DAC0_AMC = TRIM_Read(NVR_ADDR_DAC_0_AMC_1_20V);
00070             SYS0->AFE_DAC0_DC = TRIM_Read(NVR_ADDR_DAC_0_DC_1_20V);
00071         }
00072         else
00073         {
00074             SYS0->AFE_REG1 &= ~BIT6;
00075             SYS0->AFE_DAC0_AMC = TRIM_Read(NVR_ADDR_DAC_0_AMC_4_8V);
00076             SYS0->AFE_DAC0_DC = TRIM_Read(NVR_ADDR_DAC_0_DC_4_8V);
00077         }
00078     }
00079     else if (DAC_Channel == DAC_CHANNEL_1)
00080     {
00081         if (DAC_RANGE_x == DAC_RANGE_1_1V2)
00082         {
00083             SYS0->AFE_REG1 |= BIT7;
00084             SYS0->AFE_DAC1_AMC = TRIM_Read(NVR_ADDR_DAC_1_AMC_1_20V);
00085             SYS0->AFE_DAC1_DC = TRIM_Read(NVR_ADDR_DAC_1_DC_1_20V);
00086         }
00087         else
00088         {
00089             SYS0->AFE_REG1 &= ~BIT7;
00090             SYS0->AFE_DAC1_AMC = TRIM_Read(NVR_ADDR_DAC_1_AMC_3_0V);
00091             SYS0->AFE_DAC1_DC = TRIM_Read(NVR_ADDR_DAC_1_DC_3_0V);
00092         }
00093     }
00094     else
00095     {
00096         __NOP();
00097     }
00098     SYS0->PROTECT = 0;
00099 }
00100
00106 void DAC_OutputVoltage(uint8_t DAC_Channel, uint16_t DACVoltage)

```

```

00107 {
00108     uint32_t val;
00109     DAC_Enable(DAC_Channel);
00110     if (DAC_Channel == DAC_CHANNEL_0)
00111     {
00112         if (DACVoltage < 1000)
00113         {
00114             DAC_SetRange(DAC_Channel, DAC_RANGE_0_1V2);
00115             val = 0.001f * DACVoltage / 1.2f * 4095;
00116             DAC_OutputValue(DAC_Channel, val);
00117         }
00118         else
00119         {
00120             DAC_SetRange(DAC_Channel, DAC_RANGE_0_4V76);
00121             val = DACVoltage * 0.001f / 4.76f * 4096;
00122             DAC_OutputValue(DAC_Channel, val);
00123         }
00124     }
00125     else if (DAC_Channel == DAC_CHANNEL_1)
00126     {
00127         if (DACVoltage < 1000)
00128         {
00129             DAC_SetRange(DAC_Channel, DAC_RANGE_1_1V2);
00130             val = 0.001f * DACVoltage / 1.2f * 4095;
00131             DAC_OutputValue(DAC_Channel, val);
00132         }
00133         else
00134         {
00135             DAC_SetRange(DAC_Channel, DAC_RANGE_1_3V);
00136             val = 0.001f * DACVoltage / 3.0f * 4095;
00137             DAC_OutputValue(DAC_Channel, val);
00138         }
00139     }
00140     else
00141     {
00142         __NOP();
00143     }
00144 }
00145
00151 void DAC_Cmd(uint8_t DAC_Channel, uint8_t state)
00152 {
00153     SYS0->PROTECT = 0x7a83;
00154     if (DAC_Channel == DAC_CHANNEL_0)
00155     {
00156         if (state)
00157         {
00158             SYS0->AFE_REG5 |= BIT1;
00159         }
00160         else
00161         {
00162             SYS0->AFE_REG5 &= ~BIT1;
00163         }
00164     }
00165     else
00166     {
00167         __NOP();
00168     }
00169     if (DAC_Channel == DAC_CHANNEL_1)
00170     {
00171         if (state)
00172         {
00173             SYS0->AFE_REG5 |= BIT2;
00174         }
00175         else
00176         {
00177             SYS0->AFE_REG5 &= ~BIT2;
00178         }
00179     }
00180     else
00181     {
00182         __NOP();
00183     }
00184     SYS0->PROTECT = 0;
00185 }
00186
00191 void DAC_Enable(uint8_t DAC_Channel)
00192 {
00193     DAC_Cmd(DAC_Channel, ENABLE);
00194 }
00199 void DAC_Disable(uint8_t DAC_Channel)
00200 {
00201     DAC_Cmd(DAC_Channel, DISABLE);
00202 }
00207 void DAC_Reset(uint8_t DAC_Channel)
00208 {
00209     DAC_SetRange(DAC_Channel, DAC_RANGE_0_4V76);
00210     DAC_OutputValue(DAC_Channel, 0);

```

```

00211     DAC_Cmd(DAC_Channel, DISABLE);
00212 }
00218 void DAC_Output(uint8_t DAC_Channel, uint8_t state)
00219 {
00220     SYS0->PROTECT = 0x7a83;
00221     if (DAC_Channel == DAC_CHANNEL_0)
00222     {
00223         if (state)
00224         {
00225             SYS0->AFE_REG2 |= BIT6;
00226         }
00227         else
00228         {
00229             SYS0->AFE_REG2 &= ~BIT6;
00230         }
00231     }
00232     else if (DAC_Channel == DAC_CHANNEL_1)
00233     {
00234         if (state)
00235         {
00236             SYS0->AFE_REG2 |= BIT7;
00237         }
00238         else
00239         {
00240             SYS0->AFE_REG2 &= ~BIT7;
00241         }
00242     }
00243     else
00244     {
00245         __NOP();
00246     }
00247     SYS0->PROTECT = 0;
00248 }
00254 uint32_t DAC_Value(uint8_t DAC_Channel)
00255 {
00256     if (DAC_Channel == DAC_CHANNEL_0)
00257     {
00258         return (SYS0->AFE_DAC0 & 0xFFF);
00259     }
00260     else
00261     {
00262         return (SYS0->AFE_DAC1 & 0xFF) << 4; // DAC1本身是8位，这里左移4位，以确保DAC0和DAC1用法一致
00263     }
00264 }
00270 uint16_t DAC_Voltage(uint8_t DAC_Channel)
00271 {
00272     uint32_t val = 0;
00273     if (DAC_Channel == DAC_CHANNEL_0)
00274     {
00275         if (DAC_Range(DAC_Channel)) // 1.2V量程
00276         {
00277             val = DAC_Value(DAC_Channel);
00278             val = val * 1.2f * 1000 / 4095;
00279         }
00280         else
00281         {
00282             val = DAC_Value(DAC_Channel);
00283             val = val * 4.85f * 1000 / 4095;
00284         }
00285     }
00286     else if (DAC_Channel == DAC_CHANNEL_1)
00287     {
00288         if (DAC_Range(DAC_Channel)) // 1.2V量程
00289         {
00290             val = DAC_Value(DAC_Channel);
00291             val = val * 1.2f * 1000 / 4095;
00292         }
00293         else
00294         {
00295             val = DAC_Value(DAC_Channel);
00296             val = val * 3.0f * 1000 / 4095;
00297         }
00298     }
00299     else
00300     {
00301         __NOP();
00302     }
00303     return val;
00304 }
00310 uint8_t DAC_Range(uint8_t DAC_Channel)
00311 {
00312     if (DAC_Channel == DAC_CHANNEL_0)
00313     {
00314         if (SYS0->AFE_REG1 & BIT6)
00315         {
00316             return DAC_RANGE_0.4V76;
00317         }

```

```

00318         else
00319         {
00320             return DAC_RANGE_0.1V2;
00321         }
00322     }
00323     else if (DAC_Channel == DAC_CHANNEL_1)
00324     {
00325         if (SYS0->AFE_REG1 & BIT7)
00326         {
00327             return DAC_RANGE_1.3V;
00328         }
00329         else
00330         {
00331             return DAC_RANGE_1.1V2;
00332         }
00333     }
00334     else
00335     {
00336         __NOP();
00337     }
00338     return 0;
00339 }
00340 void DAC_Step(uint8_t DAC_Channel, uint8_t DAC_STEP_x, uint8_t DAC_STEP_TRG_x)
00341 {
00342     if (DAC_Channel == DAC_CHANNEL_0)
00343     {
00344         SYS0->PROTECT = 0x7a83;
00345         SYS0->AFE_DAC_CTRL = (DAC_STEP_x << 4) | (DAC_STEP_TRG_x);
00346         SYS0->PROTECT = 0;
00347     }
00348     else
00349     {
00350         // dac1不支持步进
00351     }
00352 }
00353 }

```

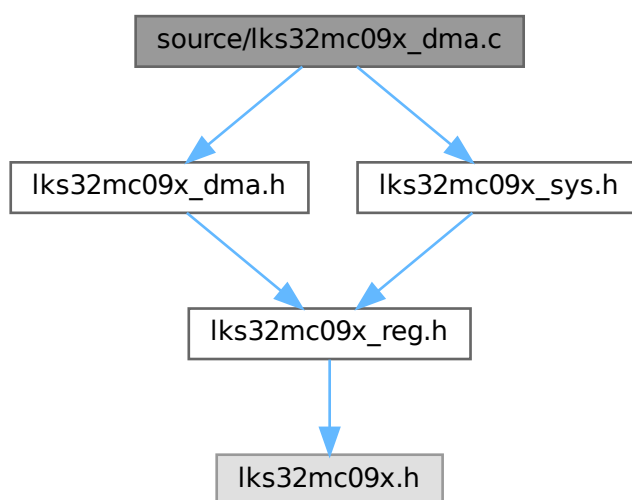
3.65 source/lks32mc09x_dma.c 文件参考

```

#include "lks32mc09x_dma.h"
#include "lks32mc09x_sys.h"

```

lks32mc09x_dma.c 的引用(Include)关系图:



函数

- void **DMA_Init** (**DMA_TypeDef** *DMAx, **DMA_InitTypeDef** *DMAInitStruct)
初始化DMA模块
- void **DMA_StructInit** (**DMA_InitTypeDef** *DMAInitStruct)
将DMA初始化结构体重置为默认值
- uint32_t **DMA_GetIRQFlag** (uint32_t DMA_IF_x)
获取DMA中断标志位状态
- void **DMA_ClearIRQFlag** (uint32_t DMA_IF_x)
清除DMA中断标志位
- uint32_t **DMA_GetFlag** (uint32_t DMA_IF_x)
获取DMA标志位状态
- void **DMA_Enable** (void)
使能DMA模块
- void **DMA_Disable** (void)
禁用DMA模块
- void **DMA_Reset** (void)
复位DMA模块
- void **DMA_SetAddr** (**DMA_TypeDef** *DMAx, uint32_t addr1, uint32_t addr2)
设置DMA通道源地址和目的地址
- void **DMA_SetLength** (**DMA_TypeDef** *DMAx, uint32_t len)
设置DMA通道传输长度
- void **DMA_Trigger** (**DMA_TypeDef** *DMAx)
触发DMA通道传输
- void **DMA_SetRequest** (**DMA_TypeDef** *DMAx, uint32_t DMA_REQ_x)
设置DMA通道请求源
- void **DMA_CHx_EN** (**DMA_TypeDef** *DMAx, uint32_t Channel_EN)
使能或禁用DMA通道
- void **DMA_Memcpy** (**DMA_TypeDef** *DMAx, void *dest, void *src, uint32_t n)
使用dma实现的内存拷贝
- void **DMA_Mem2Reg** (**DMA_TypeDef** *DMAx, void *dest, void *src, uint32_t n, uint8_t DMA_TRANS_x)
使用dma向特定寄存器写入多个数据，可用于crc计算等需要连续对单个寄存器写入数据的情况

3.65.1 函数说明

3.65.1.1 DMA_CHx.EN()

```
void DMA_CHx_EN (  
    DMA_TypeDef * DMAx,  
    uint32_t Channel_EN )
```

使能或禁用DMA通道

参数

DMAx	DMA寄存器基地址
Channel_EN	通道使能MASK

在文件 [lks32mc09x_dma.c](#) 第 218 行定义.

3.65.1.2 DMA_ClearIRQFlag()

```
void DMA_ClearIRQFlag (
    uint32_t DMA_IF_x )
```

清除DMA中断标志位

参数

DMA_↔ IF_x	中断标志位MASK

在文件 [lks32mc09x_dma.c](#) 第 106 行定义.

3.65.1.3 DMA_Disable()

```
void DMA_Disable (
    void )
```

禁用DMA模块

在文件 [lks32mc09x_dma.c](#) 第 134 行定义.

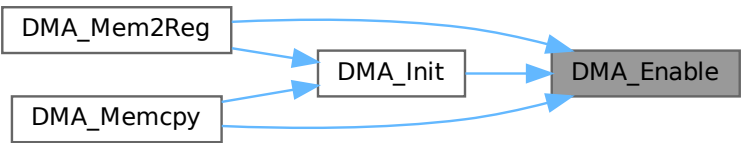
3.65.1.4 DMA_Enable()

```
void DMA_Enable (
    void )
```

使能DMA模块

在文件 [lks32mc09x_dma.c](#) 第 126 行定义.

这是这个函数的调用关系图:



3.65.1.5 DMA_GetFlag()

```
uint32_t DMA_GetFlag (
    uint32_t DMA_IF_x )
```

获取DMA标志位状态

参数

<i>DMA_↔ IF_x</i>	标志位MASK

返回

标志位状态

在文件 [lks32mc09x_dma.c](#) 第 117 行定义.

函数调用图:



3.65.1.6 DMA_GetIRQFlag()

```
uint32_t DMA_GetIRQFlag (
    uint32_t DMA_IF_x )
```

获取DMA中断标志位状态

参数

<i>DMA_↔ IF_x</i>	中断标志位MASK

返回

中断标志位状态

在文件 [lks32mc09x_dma.c](#) 第 96 行定义.

这是这个函数的调用关系图:



3.65.1.7 DMA_Init()

```
void DMA_Init (
    DMA_TypeDef * DMAx,
    DMA_InitTypeDef * DMAInitStruct )
```

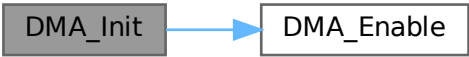
初始化DMA模块

参数

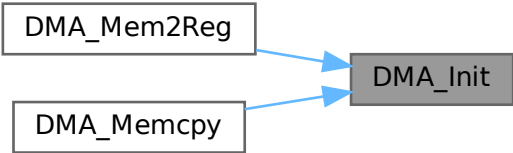
DMAx	DMA寄存器基地址
DMAInitStruct	初始化结构体指针

在文件 [lks32mc09x_dma.c](#) 第 11 行定义.

函数调用图:



这是这个函数的调用关系图:



3.65.1.8 DMA_Mem2Reg()

```
void DMA_Mem2Reg (
    DMA_TypeDef * DMAx,
    void * dest,
    void * src,
    uint32_t n,
    uint8_t DMA_TRANS_x )
```

使用dma向特定寄存器写入多个数据，可用于crc计算等需要连续对单个寄存器写入数据的情况

参数

<i>DMAx</i>	DMA寄存器基地址
<i>dest</i>	目标内存指针
<i>src</i>	源内存指针
<i>n</i>	要复制的字节数，如果使用dma1或dma2，字节数应小于0xff，如果使用dma0或dma3，字节数小于0xffff
<i>DMA_↔TRANS_x</i>	位宽，输入输出位宽一致

< 源地址访问位宽 0: Byte 1: Halfword 2: Word

< 目的地址访问位宽 0: Byte 1: Halfword 2: Word

< 源地址递增使能

< 目的地址递增使能

< 循环模式使能

< 多轮传输使能，多个DMA 请求完成传输

< 通道使能

< DMA 通道 x 数据搬运次数。

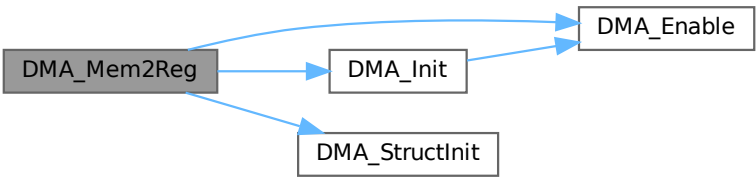
< DMA 通道 x 源地址

< DMA 通道 x 目的地址

< DMA 请求使能

在文件 [lks32mc09x_dma.c](#) 第 305 行定义.

函数调用图:



3.65.1.9 DMA_Memcpy()

```
void DMA_Memcpy (
    DMA_TypeDef * DMAx,
    void * dest,
    void * src,
    uint32_t n )
```

使用dma实现的内存拷贝

参数

<i>DMAx</i>	DMA寄存器基地址
<i>dest</i>	目标内存指针
<i>src</i>	源内存指针
<i>n</i>	要复制的字节数，如果使用dma1或dma2，字节数应小于0xff，如果使用dma0或dma3，字节数小于0xffff

< 源地址访问位宽 0: Byte 1: Halfword 2: Word

< 目的地址访问位宽 0: Byte 1: Halfword 2: Word

< 源地址递增使能

< 目的地址递增使能

< 循环模式使能

< 多轮传输使能，多个DMA 请求完成传输

< 通道使能

< DMA 通道 x 数据搬运次数。

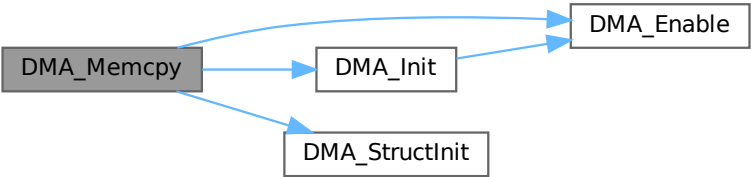
< DMA 通道 x 源地址

< DMA 通道 x 目的地址

< DMA 请求使能

在文件 [lks32mc09x_dma.c](#) 第 237 行定义.

函数调用图:



3.65.1.10 DMA_Reset()

```
void DMA_Reset (
    void )
```

复位DMA模块

在文件 [lks32mc09x_dma.c](#) 第 142 行定义.

函数调用图:



3.65.1.11 DMA_SetAddr()

```
void DMA_SetAddr (
    DMA_TypeDef * DMAx,
    uint32_t addr1,
    uint32_t addr2 )
```

设置DMA通道源地址和目的地址

参数

<i>DMAx</i>	DMA通道寄存器基地址
<i>addr1</i>	源地址
<i>addr2</i>	目的地址

在文件 [lks32mc09x_dma.c](#) 第 154 行定义.

3.65.1.12 DMA_SetLength()

```
void DMA_SetLength (
    DMA_TypeDef * DMAx,
    uint32_t len )
```

设置DMA通道传输长度

参数

<i>DMAx</i>	DMA通道寄存器基地址
<i>len</i>	传输长度

在文件 [lks32mc09x_dma.c](#) 第 166 行定义.

3.65.1.13 DMA_SetRequest()

```
void DMA_SetRequest (
    DMA_TypeDef * DMAx,
    uint32_t DMA_REQ_x )
```

设置DMA通道请求源

参数

DMAx	DMA通道寄存器基地址
DMA_↔ REQ_x	请求源MASK

在文件 [lks32mc09x_dma.c](#) 第 207 行定义.

3.65.1.14 DMA_StructInit()

```
void DMA_StructInit (
    DMA_InitTypeDef * DMAInitStruct )
```

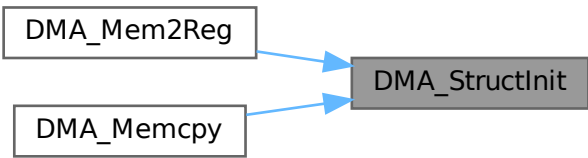
将DMA初始化结构体重置为默认值

参数

DMAInitStruct	初始化结构体指针

在文件 [lks32mc09x_dma.c](#) 第 81 行定义.

这是这个函数的调用关系图:



3.65.1.15 DMA_Trigger()

```
void DMA_Trigger (
    DMA_TypeDef * DMAx )
```

触发DMA通道传输

参数

<i>DMAx</i>	DMA通道寄存器基地址

在文件 [lks32mc09x_dma.c](#) 第 196 行定义.

3.66 lks32mc09x_dma.c

浏览该文件的文档.

```

00001 #include "lks32mc09x_dma.h"
00002 #include "lks32mc09x_sys.h"
00003
00011 void DMA_Init(DMA_TypeDef *DMAx, DMA_InitTypeDef *DMAInitStruct)
00012 {
00013     DMA_Enable();
00014     DMAx->CCR = 0;
00015     DMAx->REN = DMAInitStruct->REN;
00016     DMAx->SADR = DMAInitStruct->SADR;
00017     DMAx->DADR = DMAInitStruct->DADR;
00018
00019     if (DMAInitStruct->TMS == 0)
00020     {
00021         DMAInitStruct->TMS = 1;
00022     }
00023     else
00024     {
00025         __NOP();
00026     }
00027     if ((DMAx == DMA1) || (DMAx == DMA2))
00028     {
00029         if (DMAInitStruct->TMS >= 0x100)
00030         {
00031             DMAInitStruct->TMS = 0xff;
00032         }
00033     }
00034     else if (DMAInitStruct->TMS >= 0x1000)
00035     {
00036         DMAInitStruct->TMS = 0xfff;
00037     }
00038     else
00039     {
00040         __NOP();
00041     }
00042     do
00043     {
00044         DMAx->CTMS = DMAInitStruct->TMS;
00045     } while (DMAx->CTMS != DMAInitStruct->TMS);
00046
00047     DMAx->CCR = (DMAInitStruct->SBTW << 10) |
00048                 (DMAInitStruct->DBTW << 8) |
00049                 (DMAInitStruct->SINC << 6) |
00050                 (DMAInitStruct->DINC << 4) |
00051                 (DMAInitStruct->CIRC << 3) |
00052                 (DMAInitStruct->RMODE << 1) |
00053                 DMAInitStruct->EN;
00054
00055     if (DMAInitStruct->IRQ_EN)
00056     {
00057         if (DMAx == DMA0)
00058         {
00059             REG_WRITE_BIT(DMA0_IE, BIT0, DMAInitStruct->IRQ_EN << 0);
00060         }
00061         else if (DMAx == DMA1)
00062         {
00063             REG_WRITE_BIT(DMA0_IE, BIT1, DMAInitStruct->IRQ_EN << 1);
00064         }
00065         else if (DMAx == DMA2)
00066         {
00067             REG_WRITE_BIT(DMA0_IE, BIT2, DMAInitStruct->IRQ_EN << 2);
00068         }
00069         else if (DMAx == DMA3)
00070         {
00071             REG_WRITE_BIT(DMA0_IE, BIT3, DMAInitStruct->IRQ_EN << 3);
00072         }
00073     }

```

```

00073     }
00074 }
00075
00081 void DMA_StructInit(DMA_InitTypeDef *DMAInitStruct)
00082 {
00083     uint32_t i;
00084     for (i = 0; i < sizeof(DMA_InitTypeDef); i++)
00085     {
00086         ((uint8_t *)DMAInitStruct)[i] = 0;
00087     }
00088 }
00089
00096 uint32_t DMA_GetIRQFlag(uint32_t DMA_IF_x)
00097 {
00098     return (DMA_TOP->IF & DMA_IF_x) ? 1 : 0;
00099 }
00100
00106 void DMA_ClearIRQFlag(uint32_t DMA_IF_x)
00107 {
00108     DMA_TOP->IF = DMA_IF_x;
00109 }
00110
00117 uint32_t DMA_GetFlag(uint32_t DMA_IF_x)
00118 {
00119     // DMA标志位和中断标志位共用同一个寄存器
00120     return DMA_GetIRQFlag(DMA_IF_x);
00121 }
00122
00126 void DMA_Enable(void)
00127 {
00128     DMA_TOP->CTRL |= 0x01;
00129 }
00130
00134 void DMA_Disable(void)
00135 {
00136     DMA_TOP->CTRL &= ~0x01;
00137 }
00138
00142 void DMA_Reset(void)
00143 {
00144     SYS_SoftResetModule(SYS_MODULE_DMA0);
00145 }
00146
00154 void DMA_SetAddr(DMA_TypeDef *DMAx, uint32_t addr1, uint32_t addr2)
00155 {
00156     DMAx->SADDR = addr1;
00157     DMAx->DADDR = addr2;
00158 }
00159
00166 void DMA_SetLength(DMA_TypeDef *DMAx, uint32_t len)
00167 {
00168     if ((DMAx == DMA1) || (DMAx == DMA2))
00169     {
00170         if (len >= 0x100)
00171         {
00172             len = 0xff;
00173         }
00174     }
00175     else if (len >= 0x1000)
00176     {
00177         len = 0xffff;
00178     }
00179     else
00180     {
00181         __NOP();
00182     }
00183     DMAx->CCR = DMAx->CCR & ~BIT0;
00184     do
00185     {
00186         DMAx->CTMS = len;
00187     } while (DMAx->CTMS != len);
00188     DMAx->CCR = DMAx->CCR | BIT0;
00189 }
00190
00196 void DMA_Trigger(DMA_TypeDef *DMAx)
00197 {
00198     REG_SET(DMAx->REN, DMA_REN_SW_TRIG);
00199 }
00200
00207 void DMA_SetRequest(DMA_TypeDef *DMAx, uint32_t DMA_REQ_x)
00208 {
00209     REG_SET(DMAx->REN, DMA_REQ_x);
00210 }
00211
00218 void DMA_CHx_EN(DMA_TypeDef *DMAx, uint32_t Channel_EN)
00219 {
00220     if (Channel_EN)

```

```

00221     {
00222         DMAx->CCR |= 0x01; // 使能通道
00223     }
00224     else
00225     {
00226         DMAx->CCR &= ~0x01; // 禁用通道
00227     }
00228 }
00229
00230 void DMA_Memcpy(DMA_TypeDef *DMAx, void *dest, void *src, uint32_t n)
00231 {
00232     uint32_t ifbit = 0;
00233     uint32_t maxlen = 0;
00234     if (DMAx == DMA0)
00235     {
00236         ifbit = BIT0;
00237         maxlen = 65535;
00238     }
00239     else if (DMAx == DMA1)
00240     {
00241         ifbit = BIT1;
00242         maxlen = 255;
00243     }
00244     else if (DMAx == DMA2)
00245     {
00246         ifbit = BIT2;
00247         maxlen = 255;
00248     }
00249     else if (DMAx == DMA3)
00250     {
00251         ifbit = BIT3;
00252         maxlen = 65535;
00253     }
00254     else
00255     {
00256         __NOP();
00257     }
00258     if (n > maxlen)
00259     {
00260         n = maxlen;
00261     }
00262     else
00263     {
00264         __NOP();
00265     }
00266     DMA_Enable();
00267     DMA_InitTypeDef dmaconfig;
00268     DMA_StructInit(&dmaconfig);
00269
00270     dmaconfig.IRQ_EN = 0;
00271     dmaconfig.SBTW = 0;
00272     dmaconfig.DBTW = 0;
00273     dmaconfig.SINC = 1;
00274     dmaconfig.DINC = 1;
00275     dmaconfig.CIRC = 0;
00276     dmaconfig.RMODE = 0;
00277     dmaconfig.EN = 1;
00278     dmaconfig.TMS = n;
00279     dmaconfig.SADR = (uint32_t)(intptr_t)src;
00280     dmaconfig.DADR = (uint32_t)(intptr_t)dest;
00281     dmaconfig.REN = 0;
00282     DMA_Init(DMAx, &dmaconfig);
00283     DMA0_IF = ifbit;
00284     DMAx->REN = DMA_REN_SW_TRIG;
00285     while ((DMA0_IF & ifbit) == 0)
00286     {
00287         __NOP();
00288     }
00289 }
00290 void DMA_Mem2Reg(DMA_TypeDef *DMAx, void *dest, void *src, uint32_t n, uint8_t DMA_TRANS_x)
00291 {
00292     uint32_t ifbit = 0;
00293     uint32_t maxlen = 0;
00294     if (DMAx == DMA0)
00295     {
00296         ifbit = BIT0;
00297         maxlen = 65535;
00298     }
00299     else if (DMAx == DMA1)
00300     {
00301         ifbit = BIT1;
00302         maxlen = 255;
00303     }
00304     else if (DMAx == DMA2)
00305     {
00306         ifbit = BIT2;
00307         maxlen = 255;
00308     }

```



```

00323     }
00324     else if (DMAx == DMA3)
00325     {
00326         ifbit = BIT3;
00327         maxlen = 65535;
00328     }
00329     else
00330     {
00331         __NOP();
00332     }
00333     if (n > maxlen)
00334     {
00335         n = maxlen;
00336     }
00337     else
00338     {
00339         __NOP();
00340     }
00341     DMA_Enable();
00342     DMA_InitTypeDef dmaconfig;
00343     DMA_StructInit(&dmaconfig);
00344
00345     dmaconfig.IRQ_EN = 0;
00346     dmaconfig.SBTW = DMA_TRANS_x;
00347     dmaconfig.DBTW = DMA_TRANS_x;
00348     dmaconfig.SINC = 1;
00349     dmaconfig.DINC = 0;
00350     dmaconfig.CIRC = 0;
00351     dmaconfig.RMODE = 0;
00352     dmaconfig.EN = 1;
00353     dmaconfig.TMS = n;
00354     dmaconfig.SADDR = (uint32_t)(intptr_t)src;
00355     dmaconfig.DADDR = (uint32_t)(intptr_t)dest;
00356     dmaconfig.REN = 0;
00357     DMA_Init(DMAx, &dmaconfig);
00358     DMA0_IF = ifbit;
00359     DMAx->REN = DMA_REN_SW_TRIG;
00360     while ((DMA0_IF & ifbit) == 0)
00361     {
00362         __NOP();
00363     }
00364 }

```

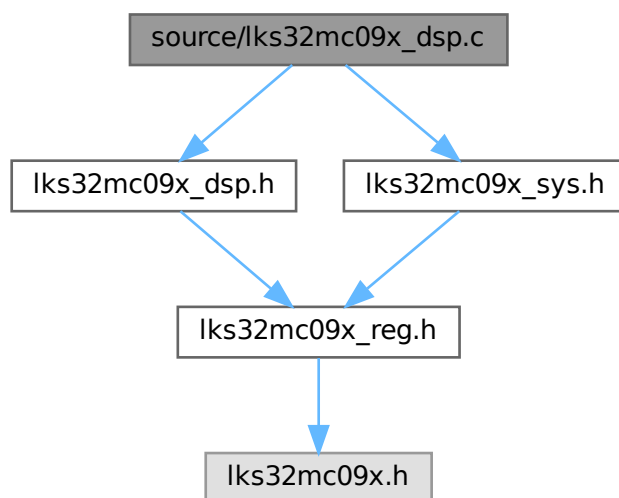
3.67 source/lks32mc09x_dsp.c 文件参考

```

#include "lks32mc09x_dsp.h"
#include "lks32mc09x_sys.h"

```

lks32mc09x_dsp.c 的引用(Include)关系图:



函数

- void **DSP_Init** (void)
初始化DSP模块
- void **DSP_Enable** (void)
使能DSP模块
- void **DSP_Disable** (void)
禁用DSP模块
- void **DSP_Reset** (void)
复位DSP模块
- s32 **DSP_Div** (s32 a, s32 b)
执行整数除法
- uint16_t **DSP_Rms** (s16 a, s16 b)
计算均方根值
- s32 **DSP_Mod** (s32 a, s32 b)
执行取模运算
- uint16_t **DSP_Sqrt** (uint32_t val)
计算平方根

3.67.1 函数说明

3.67.1.1 DSP_Disable()

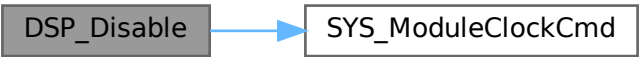
```
void DSP_Disable (
    void )
```

禁用DSP模块

该函数用于禁用DSP模块，停止处理数据。

在文件 [lks32mc09x_dsp.c](#) 第 30 行定义.

函数调用图:



3.67.1.2 DSP.Div()

```
s32 DSP_Div (
    s32 a,
    s32 b )
```

执行整数除法

参数

<i>a</i>	被除数
<i>b</i>	除数

返回

$a \div b$

在文件 [lks32mc09x_dsp.c](#) 第 52 行定义.

3.67.1.3 DSP.Enable()

```
void DSP_Enable (
    void )
```

使能DSP模块

该函数用于使能DSP模块，开始处理数据。

在文件 [lks32mc09x_dsp.c](#) 第 20 行定义.

函数调用图:



3.67.1.4 DSP_Init()

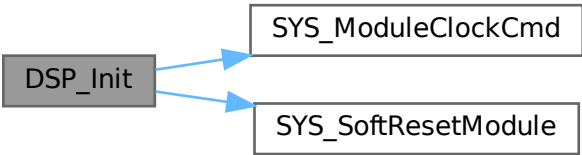
```
void DSP_Init (
    void )
```

初始化DSP模块

该函数用于初始化DSP模块，使其处于可用状态。

在文件 [lks32mc09x_dsp.c](#) 第 9 行定义.

函数调用图:



3.67.1.5 DSP_Mod()

```
s32 DSP_Mod (
    s32 a,
    s32 b )
```

执行取模运算

参数

<i>a</i>	被除数
<i>b</i>	除数

返回

余数

在文件 [lks32mc09x_dsp.c](#) 第 83 行定义.

3.67.1.6 DSP_Reset()

```
void DSP_Reset (
    void )
```

复位DSP模块

该函数用于复位DSP模块，使其恢复到初始状态。

在文件 [lks32mc09x_dsp.c](#) 第 40 行定义.

函数调用图:



3.67.1.7 DSP_Rms()

```
uint16_t DSP_Rms (
    s16 a,
    s16 b )
```

计算均方根值

参数

<i>a</i>	输入值1
<i>b</i>	输入值2

返回

均方根值

在文件 [lks32mc09x_dsp.c](#) 第 67 行定义.

3.67.1.8 DSP_Sqrt()

```
uint16_t DSP_Sqrt (
    uint32_t val )
```

计算平方根

参数

<i>val</i>	输入值

返回

平方根值

在文件 [lks32mc09x_dsp.c](#) 第 96 行定义.

3.68 lks32mc09x_dsp.c

[浏览该文件的文档.](#)

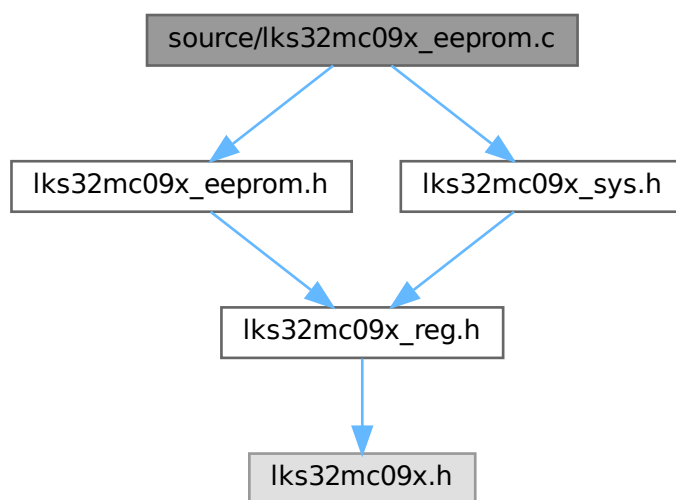
```
00001 #include "lks32mc09x_dsp.h"
00002 #include "lks32mc09x_sys.h"
00003
00009 void DSP_Init(void)
00010 {
00011     SYS_ModuleClockCmd(SYS_MODULE_DSP0, ENABLE);
00012     SYS_SoftResetModule(SYS_MODULE_DSP0);
00013 }
00014
00020 void DSP_Enable(void)
00021 {
00022     SYS_ModuleClockCmd(SYS_MODULE_DSP0, ENABLE);
00023 }
00024
00030 void DSP_Disable(void)
00031 {
00032     SYS_ModuleClockCmd(SYS_MODULE_DSP0, DISABLE);
00033 }
00034
00040 void DSP_Reset(void)
00041 {
00042     SYS_SoftResetModule(SYS_MODULE_DSP0);
00043 }
00044
00052 s32 DSP_Div(s32 a, s32 b)
00053 {
00054     DSP0_DID = a;
00055     DSP0_DIS = b;
00056     // 执行整数除法, 这里直接返回除法结果
00057     return DSP0_QUO;
00058 }
00059
00067 uint16_t DSP_Rms(s16 a, s16 b)
00068 {
00069     int32_t a32, b32;
00070     a32 = a;
00071     b32 = b;
00072     DSP0_RAD = a32 * a32 + b32 * b32;
00073     return DSP0_SQRT;
00074 }
00075
00083 s32 DSP_Mod(s32 a, s32 b)
00084 {
00085     DSP0_DID = a;
00086     DSP0_DIS = b;
00087     return DSP0_REM;
00088 }
00089
00096 uint16_t DSP_Sqrt(uint32_t val)
00097 {
00098     DSP0_RAD = val;
00099     return DSP0_SQRT;
00100 }
```

3.69 source/lks32mc09x_eeprom.c 文件参考

```
#include "lks32mc09x_eeprom.h"
```

```
#include "lks32mc09x_sys.h"
```

lks32mc09x_eeprom.c 的引用(Include)关系图:



宏定义

- #define `EEPROM_CFG` REG32(0x40048000)
- #define `EEPROM_RDATA` REG32(0x40048004)

函数

- void `EEPROM_Init` (`EEPROM_InitTypeDef` *EEPROM_InitStruct)
初始化
- void `EEPROM_StructInit` (`EEPROM_InitTypeDef` *EEPROM_InitStruct)
初始化EEPROM_InitStruct结构体为默认值
- void `EEPROM_InitDefault` (void)
初始化为默认配置, IO不交换, 波特率100k, 地址数据长度16位, 读写时响应中断
- void `EEPROM_Enable` (void)
EEPROM使能
- void `EEPROM_Disable` (void)
EEPROM失能
- void `EEPROM_Reset` (void)
EEPROM复位
- uint8_t `EEPROM_Read` (uint16_t eeprom_addr)
读取eeprom内数据
- void `EEPROM_Write` (uint16_t eeprom_addr, uint8_t data)

向 *eeeprom* 写入数据

- void **EEPROM.WriteData** (uint8_t eeepromaddr, u16 taddr, u8 *data, uint32_t len)

向 *eeeprom* 写入数据(多字节), 09上 *eeeprom* 本身不支持多字节模式, 这里用 *n* 个单次读写实现, 执行时间为 *len*5mS* (典型值)

- void **EEPROM.ReadData** (uint8_t eeepromaddr, u16 taddr, u8 *data, uint32_t len)

读取 *eeeprom* 内数据, 09上 *eeeprom* 本身不支持多字节模式, 这里用 *n* 个单次读写实现

3.69.1 宏定义说明

3.69.1.1 EEPROM_CFG

```
#define EEPROM_CFG REG32(0x40048000)
```

在文件 [lks32mc09x_eeeprom.c](#) 第 4 行定义.

3.69.1.2 EEPROM_RDATA

```
#define EEPROM_RDATA REG32(0x40048004)
```

在文件 [lks32mc09x_eeeprom.c](#) 第 7 行定义.

3.69.2 函数说明

3.69.2.1 EEPROM_Disable()

```
void EEPROM_Disable (
    void )
```

EEPROM失能

在文件 [lks32mc09x_eeeprom.c](#) 第 52 行定义.

函数调用图:



3.69.2.2 EEPROM_Enable()

```
void EEPROM_Enable (
    void )
```

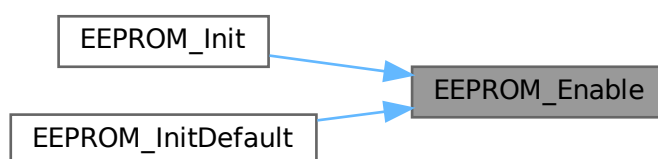
EEPROM使能

在文件 [lks32mc09x_eeprom.c](#) 第 45 行定义.

函数调用图:



这是这个函数的调用关系图:



3.69.2.3 EEPROM_Init()

```
void EEPROM_Init (
    EEPROM_InitTypeDef * EEPROM_InitStruct )
```

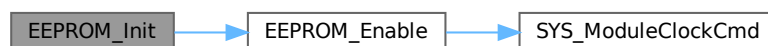
初始化

参数

<code>EEPROM_InitStruct</code>	EEPROM初始化结构体指针

在文件 [lks32mc09x_eeprom.c](#) 第 13 行定义.

函数调用图:



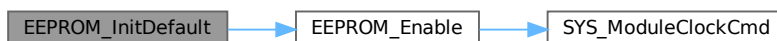
3.69.2.4 EEPROM_InitDefault()

```
void EEPROM_InitDefault (
    void )
```

初始化为默认配置，IO不交换，波特率100k，地址数据长度16位，读写时响应中断

在文件 [lks32mc09x_eeprom.c](#) 第 37 行定义.

函数调用图:



3.69.2.5 EEPROM_Read()

```
uint8_t EEPROM_Read (
    uint16_t eeprom_addr )
```

读取eeprom内数据

参数

<i>eeprom_addr</i>	eeprom数据地址

返回

数据

在文件 [lks32mc09x_eeprom.c](#) 第 69 行定义.

这是这个函数的调用关系图:



3.69.2.6 EEPROM_ReadData()

```
void EEPROM_ReadData (
    uint8_t eepromaddr,
    u16 taddr,
    u8 * data,
    uint32_t len )
```

读取eeprom内数据，09上eeprom本身不支持多字节模式，这里用n个单次读写实现

参数

<i>eepromaddr</i>	eeprom模块设备地址，例如0xa0
<i>taddr</i>	eeprom数据的起始地址
<i>data</i>	存放数据的地址
<i>len</i>	数据长度

在文件 [lks32mc09x_eeprom.c](#) 第 145 行定义.

函数调用图:



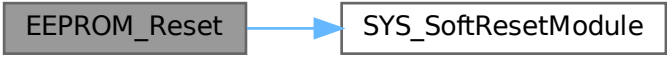
3.69.2.7 EEPROM_Reset()

```
void EEPROM_Reset (
    void )
```

EEPROM复位

在文件 [lks32mc09x_eeprom.c](#) 第 59 行定义.

函数调用图:



3.69.2.8 EEPROM_StructInit()

```
void EEPROM_StructInit (
    EEPROM_InitTypeDef * EEPROM_InitStruct )
```

初始化EEPROM_InitStruct结构体为默认值

在文件 [lks32mc09x_eeprom.c](#) 第 26 行定义.

3.69.2.9 EEPROM_Write()

```
void EEPROM_Write (
    uint16_t eeprom_addr,
    uint8_t data )
```

向eeprom写入数据

参数

<i>eeprom_addr</i>	eeprom数据地址
<i>data</i>	要写入的数据

在文件 [lks32mc09x_eeprom.c](#) 第 94 行定义.

这是这个函数的调用关系图:



3.69.2.10 EEPROM_WriteData()

```
void EEPROM_WriteData (
    uint8_t eepromaddr,
    u16 taddr,
    u8 * data,
    uint32_t len )
```

向eeprom写入数据(多字节), 09上eeprom本身不支持多字节模式, 这里用n个单次读写实现, 执行时间为len*5mS (典型值)

参数

<i>eepromaddr</i>	eeprom模块设备地址, 例如0xa0
<i>taddr</i>	eeprom数据的起始地址
<i>data</i>	要写入的数据地址
<i>len</i>	要写入的长度

在文件 [lks32mc09x_eeprom.c](#) 第 128 行定义.

函数调用图:



3.70 lks32mc09x_eeprom.c

[浏览该文件的文档.](#)

```
00001 #include "lks32mc09x_eeprom.h"
00002 #include "lks32mc09x_sys.h"
00003 #ifndef EEPROM_CFG
00004 #define EEPROM_CFG REG32(0x40048000)
00005 #endif
00006 #ifndef EEPROM_RDATA
00007 #define EEPROM_RDATA REG32(0x40048004)
00008 #endif
00013 void EEPROM_Init(EEPROM_InitTypeDef *EEPROM_InitStruct)
00014 {
00015     EEPROM_Enable();
00016     EEPROM_InitStruct->addr >>= 1;
00017     EEPROM_CFG = (EEPROM_InitStruct->addr << 2) |
00018                 (EEPROM_InitStruct->EEPROM_BAUD_x << 0) |
00019                 (EEPROM_InitStruct->RDY_RLS << 16) |
00020                 (EEPROM_InitStruct->IO_SWITCH << 10) |
00021                 (EEPROM_InitStruct->ADDR_LEN_x << 9);
00022 }
00026 void EEPROM_StructInit(EEPROM_InitTypeDef *EEPROM_InitStruct)
00027 {
00028     uint32_t i;
00029     for (i = 0; i < sizeof(EEPROM_InitTypeDef); i++)
00030     {
00031         ((uint8_t *)EEPROM_InitStruct)[i] = 0;
00032     }
00033 }
```

```

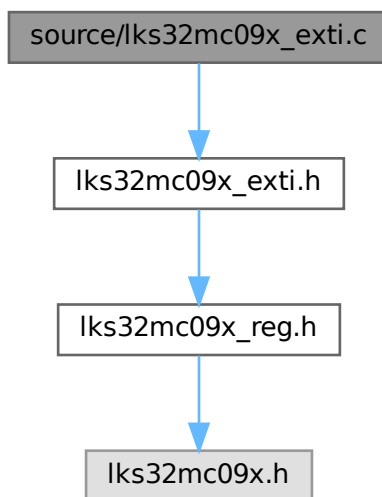
00037 void EEPROM_InitDefault(void)
00038 {
00039     EEPROM_Enable();
00040     EEPROM_CFG = BIT16 | BIT9 | (0xa0 << 1) | 0;
00041 }
00045 void EEPROM_Enable(void)
00046 {
00047     SYS_ModuleClockCmd(SYS_MODULE_EEPROM, ENABLE);
00048 }
00052 void EEPROM_Disable(void)
00053 {
00054     SYS_ModuleClockCmd(SYS_MODULE_EEPROM, DISABLE);
00055 }
00059 void EEPROM_Reset(void)
00060 {
00061     SYS_SoftResetModule(SYS_MODULE_EEPROM);
00062 }
00063
00069 uint8_t EEPROM_Read(uint16_t eeprom_addr)
00070 {
00071     uint8_t val;
00072     if (eeprom_addr > 32767)
00073     {
00074         eeprom_addr = 32767;
00075     }
00076     else
00077     {
00078         __NOP();
00079     }
00080     while (EEPROM_CFG & BIT17); // 等待之前的操作结束 (如果有的话)
00081     do
00082     {
00083         val = REG8(eeprom_addr | 0x40040000);
00084         while (EEPROM_CFG & BIT17);
00085     } while (EEPROM_CFG & BIT11);
00086     val = EEPROM_RDATA;
00087     return val;
00088 }
00094 void EEPROM_Write(uint16_t eeprom_addr, uint8_t data)
00095 {
00096     if (eeprom_addr > 32767)
00097     {
00098         eeprom_addr = 32767;
00099     }
00100     else
00101     {
00102         __NOP();
00103     }
00104     if (EEPROM_CFG & BIT16)
00105     {
00106         do
00107         {
00108             REG8(eeprom_addr + 0x40040000) = data;
00109             while (EEPROM_CFG & BIT17);
00110         } while (EEPROM_CFG & BIT11); // 收到NACK则自动重发
00111     }
00112     else
00113     {
00114         REG8(eeprom_addr + 0x40040000) = data; // 先触发一次写操作
00115         do
00116         {
00117             REG8(eeprom_addr + 0x40040000) = data; // 这里的写操作时,mcu会暂停运行,等待上一个写操作完成
00118         } while (EEPROM_CFG & BIT11); // 这里判断的是上一次的写操作是否收到nack
00119     }
00120 }
00128 void EEPROM_WriteData(uint8_t eepromaddr, u16 taddr, u8 *data, uint32_t len)
00129 {
00130     uint16_t i;
00131     eepromaddr &= 0xfe;
00132     EEPROM_CFG = (EEPROM_CFG & (~0x1fc)) | (eepromaddr << 1);
00133     for (i = 0; i < len; i++)
00134     {
00135         EEPROM_Write(taddr + i, data[i]);
00136     }
00137 }
00145 void EEPROM_ReadData(uint8_t eepromaddr, u16 taddr, u8 *data, uint32_t len)
00146 {
00147     uint16_t i;
00148     eepromaddr &= 0xfe;
00149     EEPROM_CFG = (EEPROM_CFG & (~0x1fc)) | (eepromaddr << 1);
00150     for (i = 0; i < len; i++)
00151     {
00152         data[i] = EEPROM_Read(taddr + i);
00153     }
00154 }

```

3.71 source/lks32mc09x_exti.c 文件参考

```
#include "lks32mc09x_exti.h"
```

lks32mc09x_exti.c 的引用(Include)关系图:



函数

- void `EXTI_Reset` (`EXTI_TypeDef` *EXTIx)
复位EXT模块
- void `EXTI_TriggerConfig` (`EXTI_TypeDef` *EXTIx, uint8_t EXTI_CR_x, `EXTI_Mode_TypeDef` EXTI_Trigger)
配置GPIO引脚的中断触发模式
- void `EXTI_EnableIRQFlag` (`EXTI_TypeDef` *EXTIx, uint32_t EXTI_IF_x)
使能外部中断标志
- void `EXTI_DisableIRQFlag` (`EXTI_TypeDef` *EXTIx, uint32_t EXTI_IF_x)
失能外部中断标志
- void `EXTI_ClearIRQFlag` (`EXTI_TypeDef` *EXTIx, uint32_t EXTI_IF_x)
清除GPIO引脚的中断标志
- uint8_t `EXTI_GetIRQFlag` (`EXTI_TypeDef` *EXTIx, uint32_t EXTI_IF_x)
获取GPIO引脚的中断标志状态
- void `EXTI_SetClko` (`EXTI_TypeDef` *EXTIx, uint32_t EXTI_CLKO_x)
输出时钟选择

变量

- static uint16_t `exti_ie_flg` = 0

3.71.1 函数说明

3.71.1.1 EXTI_ClearIRQFlag()

```
void EXTI_ClearIRQFlag (
    EXTI_TypeDef * EXTIx,
    uint32_t EXTI_IF_x )
```

清除GPIO引脚的中断标志

参数

<i>GPIOx</i>	GPIO模块指针
<i>GPIO.PinSource</i>	GPIO引脚源

在文件 [lks32mc09x_exti.c](#) 第 115 行定义.

3.71.1.2 EXTI_DisableIRQFlag()

```
void EXTI_DisableIRQFlag (
    EXTI_TypeDef * EXTIx,
    uint32_t EXTI_IF_x )
```

失能外部中断标志

失能GPIO引脚的中断标志

参数

<i>EXTIx</i>	EXTI模块指针
<i>EXTI_↔IF_x</i>	外部中断源

在文件 [lks32mc09x_exti.c](#) 第 80 行定义.

3.71.1.3 EXTI_EnableIRQFlag()

```
void EXTI_EnableIRQFlag (
    EXTI_TypeDef * EXTIx,
    uint32_t EXTI_IF_x )
```

使能外部中断标志

使能GPIO引脚的中断标志

参数

<i>EXTIx</i>	EXTI模块指针
<i>EXTI_↔IF_x</i>	外部中断源

在文件 [lks32mc09x_exti.c](#) 第 45 行定义.

3.71.1.4 EXTI_GetIRQFlag()

```
uint8_t EXTI_GetIRQFlag (
    EXTI_TypeDef * EXTIx,
    uint32_t EXTI_IF_x )
```

获取GPIO引脚的中断标志状态

参数

<i>GPIOx</i>	GPIO模块指针
<i>GPIO_PinSource</i>	GPIO引脚源

返回
中断标志状态

返回值

0	无中断
1	有中断

在文件 [lks32mc09x_exti.c](#) 第 130 行定义.

3.71.1.5 EXTI_Reset()

```
void EXTI_Reset (
    EXTI_TypeDef * EXTIx )
```

复位EXTI模块

参数

<i>EXTIx</i>	EXTI模块指针

在文件 [lks32mc09x_exti.c](#) 第 8 行定义.

3.71.1.6 EXTI_SetClko()

```
void EXTI_SetClko (
    EXTI_TypeDef * EXTIx,
    uint32_t EXTI_CLKO_x )
```

输出时钟选择

参数

<i>EXTIx</i>	EXTI模块指针
<i>EXTI_↔ CLKO_x</i>	输出的时钟信号

在文件 [lks32mc09x_exti.c](#) 第 141 行定义.

3.71.1.7 EXTI_TriggerConfig()

```
void EXTI_TriggerConfig (  
    EXTI_TypeDef * EXTIX,  
    uint8_t EXTI_CR_x,  
    EXTI_Mode_TypeDef EXTI_Trigger )
```

配置GPIO引脚的中断触发模式

参数

<i>GPIOx</i>	GPIO模块指针
<i>GPIO_PinSource</i>	GPIO引脚源
<i>EXTI_Trigger</i>	触发模式

在文件 [lks32mc09x_exti.c](#) 第 24 行定义.

3.71.2 变量说明

3.71.2.1 exti_ie_flg

```
uint16_t exti_ie_flg = 0 [static]
```

在文件 [lks32mc09x_exti.c](#) 第 2 行定义.

3.72 lks32mc09x_exti.c

[浏览该文件的文档.](#)

```
00001 #include "lks32mc09x_exti.h"  
00002 static uint16_t exti_ie_flg = 0;  
00008 void EXTI_Reset(EXTI_TypeDef *EXTIX)  
00009 {  
00010     // 复位EXTI模块寄存器  
00011     EXTIX->CR0 = 0;  
00012     EXTIX->CR1 = 0;  
00013     EXTIX->IE = 0;  
00014     EXTIX->IF = 0;  
00015 }  
00016  
00024 void EXTI_TriggerConfig(EXTI_TypeDef *EXTIX, uint8_t EXTI_CR_x, EXTI_Mode_TypeDef EXTI_Trigger)  
00025 {  
00026     // 根据GPIO引脚源和触发模式配置EXTI触发寄存器  
00027     if (EXTI_CR_x < 8) // cr0  
00028     {  
00029         EXTIX->CR0 &= ~(0x03 << (EXTI_CR_x * 2));
```

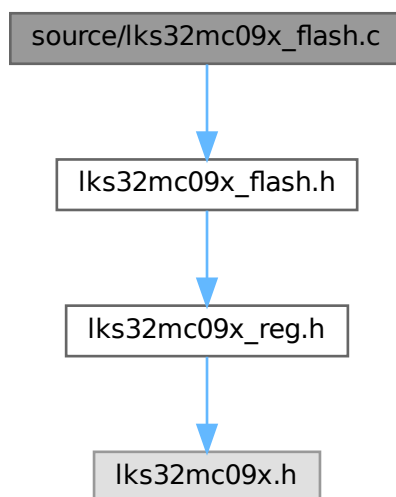
```

00030     EXTI->CR0 |= (EXTI_Trigger << (EXTI_CR_x * 2));
00031 }
00032 else // cr1
00033 {
00034     EXTI->CR1 &= ~(0x03 << ((EXTI_CR_x - 8) * 2));
00035     EXTI->CR1 |= (EXTI_Trigger << ((EXTI_CR_x - 8) * 2));
00036 }
00037 }
00038
00045 void EXTI_EnableIRQFlag(EXTI_TypeDef *EXTI, uint32_t EXTI_IF_x)
00046 {
00047     exti_ie_flg |= EXTI_IF_x;
00048     if (exti_ie_flg & 0x7f)
00049     {
00050         EXTI->IE |= BIT0;
00051     }
00052     else
00053     {
00054         EXTI->IE &= ~BIT0;
00055     }
00056     if (exti_ie_flg & 0x80)
00057     {
00058         EXTI->IE |= BIT1;
00059     }
00060     else
00061     {
00062         EXTI->IE &= ~BIT1;
00063     }
00064     if (exti_ie_flg & 0xff00)
00065     {
00066         EXTI->IE |= BIT2;
00067     }
00068     else
00069     {
00070         EXTI->IE &= ~BIT2;
00071     }
00072 }
00073
00080 void EXTI_DisableIRQFlag(EXTI_TypeDef *EXTI, uint32_t EXTI_IF_x)
00081 {
00082     exti_ie_flg &= ~EXTI_IF_x;
00083     if (exti_ie_flg & 0x3f)
00084     {
00085         EXTI->IE |= BIT0;
00086     }
00087     else
00088     {
00089         EXTI->IE &= ~BIT0;
00090     }
00091     if (exti_ie_flg & 0x80)
00092     {
00093         EXTI->IE |= BIT1;
00094     }
00095     else
00096     {
00097         EXTI->IE &= ~BIT1;
00098     }
00099     if (exti_ie_flg & 0xff00)
00100     {
00101         EXTI->IE |= BIT2;
00102     }
00103     else
00104     {
00105         EXTI->IE &= ~BIT2;
00106     }
00107 }
00108
00115 void EXTI_ClearIRQFlag(EXTI_TypeDef *EXTI, uint32_t EXTI_IF_x)
00116 {
00117     // 清除对应GPIO引脚的中断标志
00118     EXTI->IF = EXTI_IF_x;
00119 }
00120
00130 uint8_t EXTI_GetIRQFlag(EXTI_TypeDef *EXTI, uint32_t EXTI_IF_x)
00131 {
00132     // 获取对应GPIO引脚的中断标志状态
00133     return EXTI->IF & EXTI_IF_x;
00134 }
00141 void EXTI_SetClko(EXTI_TypeDef *EXTI, uint32_t EXTI_CLKO_x)
00142 {
00143     EXTI->CLKO = EXTI_CLKO_x;
00144 }

```

3.73 source/lks32mc09x_flash.c 文件参考

#include "lks32mc09x_flash.h"
lks32mc09x_flash.c 的引用(Include)关系图:



函数

- void **FLASH.Write** (uint32_t adr, uint8_t *data, uint32_t len)
向指定地址写入数据
- void **FLASH.Read** (uint32_t adr, uint8_t *data, uint32_t len)
从指定地址读取数据
- void **FLASH.Erase** (uint32_t adr)
擦除指定扇区
- void **FLASH.WriteNvr** (uint32_t adr, uint8_t *data, uint32_t len)
向指定地址写入数据(NVR区域)
- void **FLASH.EraseNvr** (uint32_t adr)
擦除指定扇区(NVR区域)
- void **FLASH.ReadNvr** (uint32_t adr, uint8_t *data, uint32_t len)
从指定地址读取数据(NVR区域)

3.73.1 函数说明

3.73.1.1 FLASH.Erase()

```
void FLASH_Erase (
    uint32_t adr )
```

擦除指定扇区

参数

<i>adr</i>	目标地址

注解

触发掉电检测后，对flash的操作无效，掉电检测的配置参考SYS.SetPowerFailureDetection

在文件 [lks32mc09x.flash.c](#) 第 53 行定义.

3.73.1.2 FLASH.EraseNvr()

```
void FLASH_EraseNvr (
    uint32_t adr )
```

擦除指定扇区(NVR区域)

参数

<i>adr</i>	目标地址

注解

触发掉电检测后，对flash的操作无效，掉电检测的配置参考SYS.SetPowerFailureDetection

在文件 [lks32mc09x.flash.c](#) 第 102 行定义.

3.73.1.3 FLASH.Read()

```
void FLASH_Read (
    uint32_t adr,
    uint8_t * data,
    uint32_t len )
```

从指定地址读取数据

参数

<i>adr</i>	目标地址
<i>buf</i>	数据缓冲区指针
<i>len</i>	数据长度

在文件 [lks32mc09x.flash.c](#) 第 38 行定义.

3.73.1.4 FLASH.ReadNvr()

```
void FLASH_ReadNvr (
```

```
uint32_t adr,
uint8_t * data,
uint32_t len )
```

从指定地址读取数据(NVR区域)

参数

<i>adr</i>	目标地址
<i>buf</i>	数据缓冲区指针
<i>len</i>	数据长度

在文件 [lks32mc09x_flash.c](#) 第 123 行定义.

3.73.1.5 FLASH.Write()

```
void FLASH.Write (
    uint32_t adr,
    uint8_t * data,
    uint32_t len )
```

向指定地址写入数据

参数

<i>adr</i>	目标地址
<i>data</i>	数据缓冲区指针
<i>len</i>	数据长度

注解

触发掉电检测后，对flash的操作无效，掉电检测的配置参考SYS_SetPowerFailureDetection

在文件 [lks32mc09x_flash.c](#) 第 10 行定义.

3.73.1.6 FLASH.WriteNvr()

```
void FLASH.WriteNvr (
    uint32_t adr,
    uint8_t * data,
    uint32_t len )
```

向指定地址写入数据(NVR区域)

参数

<i>adr</i>	目标地址
<i>data</i>	数据缓冲区指针
<i>len</i>	数据长度

注解

触发掉电检测后，对flash的操作无效，掉电检测的配置参考SYS_SetPowerFailureDetection

在文件 lks32mc09x_flash.c 第 75 行定义.

3.74 lks32mc09x_flash.c

[浏览该文件的文档.](#)

```
00001 #include "lks32mc09x_flash.h"
00002
00010 void FLASH_Write(uint32_t adr, uint8_t *data, uint32_t len)
00011 {
00012     uint8_t *p = data;
00013     uint32_t flg = __get_PRIMASK();
00014     __disable_irq();
00015     SYS_WR_PROTECT = 0x7a83;
00016     SYS_FLSP = 0x8f35;
00017     FLASH->CFG |= BIT27 | BIT23;
00018     FLASH->ADDR = adr;
00019     len &= ~3; // 低两位值忽略
00020     for (; len > 0; len -= 4)
00021     {
00022         FLASH->WDATA = p[0] | (p[1] << 8) | (p[2] << 16) | (p[3] << 24);
00023         p += 4;
00024     }
00025     SYS_FLSP = 0;
00026     FLASH->CFG &= ~(BIT27 | BIT23);
00027     SYS_WR_PROTECT = 0;
00028     __set_PRIMASK(flg);
00029 }
00030
00038 void FLASH_Read(uint32_t adr, uint8_t *data, uint32_t len)
00039 {
00040     uint32_t i;
00041     for (i = 0; i < len; i++)
00042     {
00043         data[i] = REG8(adr + i);
00044     }
00045 }
00046
00053 void FLASH_Erase(uint32_t adr)
00054 {
00055     uint32_t flg = __get_PRIMASK();
00056     __disable_irq();
00057     SYS_WR_PROTECT = 0x7a83;
00058     SYS_FLSE = 0x8fca;
00059     FLASH->CFG |= BIT31;
00060     FLASH->CFG &= ~BIT15;
00061     FLASH->ADDR = adr;
00062     FLASH->ERASE = 0x7654DCBA;
00063     SYS_FLSE = 0;
00064     FLASH->CFG &= ~BIT31;
00065     __set_PRIMASK(flg);
00066 }
00067
00075 void FLASH_WriteNvr(uint32_t adr, uint8_t *data, uint32_t len)
00076 {
00077     uint8_t *p = data;
00078     uint32_t flg = __get_PRIMASK();
00079     __disable_irq();
00080     SYS_WR_PROTECT = 0x7a83;
00081     SYS_FLSP = 0x8f35;
00082     FLASH->CFG |= BIT27 | BIT23 | BIT11;
00083     FLASH->ADDR = adr;
00084     len &= ~3; // 低两位值忽略
00085     for (; len > 0; len -= 4)
00086     {
00087         FLASH->WDATA = p[0] | (p[1] << 8) | (p[2] << 16) | (p[3] << 24);
00088         p += 4;
00089     }
00090     SYS_FLSP = 0;
00091     FLASH->CFG &= ~(BIT27 | BIT23 | BIT11);
00092     SYS_WR_PROTECT = 0;
00093     __set_PRIMASK(flg);
00094 }
00095
00102 void FLASH_EraseNvr(uint32_t adr)
00103 {
```

```

00104     uint32_t flg = __get_PRIMASK();
00105     __disable_irq();
00106     SYS_WR_PROTECT = 0x7a83;
00107     SYS_FLSE      = 0x8fca;
00108     FLASH->CFG |= BIT31 | BIT11;
00109     FLASH->CFG &= ~BIT15;
00110     FLASH->ADDR = adr;
00111     FLASH->ERASE = 0x7654DCBA;
00112     SYS_FLSE      = 0;
00113     FLASH->CFG &= ~(BIT31 | BIT11);
00114     __set_PRIMASK(flg);
00115 }
00123 void FLASH_ReadNvr(uint32_t adr, uint8_t *data, uint32_t len)
00124 {
00125     uint32_t i;
00126     uint32_t flg = __get_PRIMASK();
00127     __disable_irq();
00128     FLASH->CFG |= BIT11;
00129     FLASH->ADDR = adr;
00130     for (i = 0; i < len; i += 4)
00131     {
00132         uint32_t val = FLASH_RDATA;
00133         data[i]      = val;
00134         data[i + 1]  = val >> 8;
00135         data[i + 2]  = val >> 16;
00136         data[i + 3]  = val >> 24;
00137     }
00138     FLASH->CFG &= ~BIT11;
00139     __set_PRIMASK(flg);
00140 }

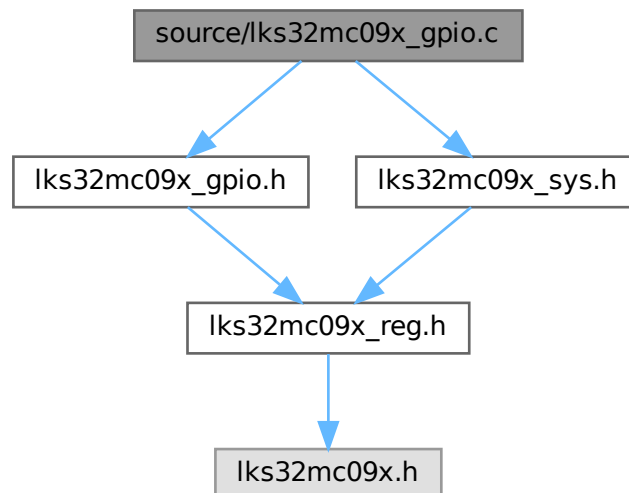
```

3.75 source/lks32mc09x_gpio.c 文件参考

```

#include "lks32mc09x_gpio.h"
#include "lks32mc09x_sys.h"
lks32mc09x_gpio.c 的引用(Include)关系图:

```



函数

- void [GPIO.Enable](#) (void)

- gpio*使能
 - void **GPIO_Disable** (void)
- gpio*失能
 - void **GPIO_Reset** (void)
- gpio*模块复位
 - void **GPIO_StructInit** (GPIO_InitTypeDef *GPIO_InitStruct)
 - 初始化GPIO结构体为默认值
 - void **GPIO_Init** (GPIO_TypeDef *GPIOx, GPIO_InitTypeDef *GPIO_InitStruct)
 - 初始化GPIO
 - void **GPIO_LockRegister** (GPIO_TypeDef *GPIOx)
 - 锁定GPIO寄存器
 - void **GPIO_UnlockRegister** (GPIO_TypeDef *GPIOx)
 - 解锁GPIO寄存器
 - uint8_t **GPIO_ReadInputDataBit** (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
 - 读取指定GPIO引脚的输入数据位
 - uint32_t **GPIO_ReadInputData** (GPIO_TypeDef *GPIOx)
 - 读取GPIO端口的输入数据
 - uint8_t **GPIO_ReadOutputDataBit** (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
 - 读取指定GPIO引脚的输出数据位
 - uint32_t **GPIO_ReadOutputData** (GPIO_TypeDef *GPIOx)
 - 读取GPIO端口的输出数据
 - void **GPIO_SetBits** (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
 - 设置指定的GPIO引脚位
 - void **GPIO_ResetBits** (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
 - 复位指定的GPIO引脚位
 - void **GPIO_Write** (GPIO_TypeDef *GPIOx, uint32_t Val)
 - 写入数据到指定的GPIO端口
 - void **GPIO_WriteBit** (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin, GPIO_Bit_Action BitVal)
 - 设置或复位指定的GPIO引脚位
 - void **GPIO_PinAFConfig** (GPIO_TypeDef *GPIOx, uint32_t GPIO_PinSource, uint32_t GPIO_AF)
 - 配置GPIO引脚复用功能
 - void **GPIO_ToggleBits** (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
 - 翻转指定的GPIO引脚位
 - void **GPIO_Config** (GPIO_TypeDef *GPIOx, uint32_t GPIO_PinSource, GPIO_Mode_TypeDef mode, uint32_t GPIO_AF_x)
 - GPIO配置函数

3.75.1 函数说明

3.75.1.1 GPIO_Config()

```
void GPIO_Config (
    GPIO_TypeDef * GPIOx,
    uint32_t GPIO_PinSource,
    GPIO_Mode_TypeDef mode,
    uint32_t GPIO_AF_x )
```

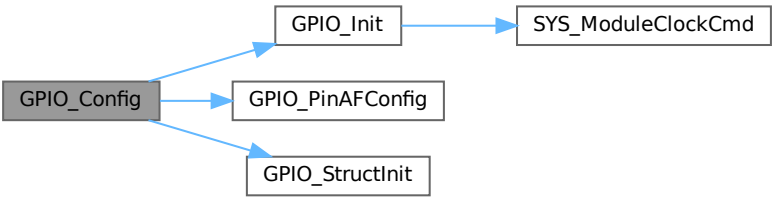
GPIO配置函数

参数

<i>GPIOx</i>	GPIO端口
<i>GPIO.PinSource</i>	GPIO引脚
<i>mode</i>	GPIO模式
<i>GPIO.AF_x</i>	GPIO复用功能选择

在文件 [lks32mc09x_gpio.c](#) 第 264 行定义.

函数调用图:



3.75.1.2 GPIO_Disable()

```
void GPIO_Disable (
    void )
```

gpio失能

在文件 [lks32mc09x_gpio.c](#) 第 14 行定义.

函数调用图:



3.75.1.3 GPIO_Enable()

```
void GPIO_Enable (
    void )
```

gpio使能

在文件 [lks32mc09x_gpio.c](#) 第 7 行定义.

函数调用图:



3.75.1.4 GPIO_Init()

```
void GPIO_Init (
    GPIO_TypeDef * GPIOx,
    GPIO_InitTypeDef * GPIO_InitStruct )
```

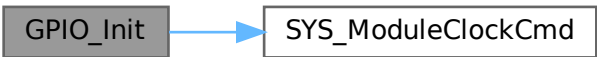
初始化GPIO

参数

<i>GPIOx</i>	GPIO模块指针
<i>GPIO_InitStruct</i>	指向包含初始化参数的GPIO_InitTypeDef结构体

在文件 [lks32mc09x_gpio.c](#) 第 43 行定义.

函数调用图:



这是这个函数的调用关系图:



3.75.1.5 GPIO_LockRegister()

```
void GPIO_LockRegister (
    GPIO_TypeDef * GPIOx )
```

锁定GPIO寄存器

参数

GPIOx	GPIO模块指针

在文件 [lks32mc09x_gpio.c](#) 第 105 行定义.

3.75.1.6 GPIO_PinAFConfig()

```
void GPIO_PinAFConfig (
    GPIO_TypeDef * GPIOx,
    uint32_t GPIO_PinSource,
    uint32_t GPIO_AF )
```

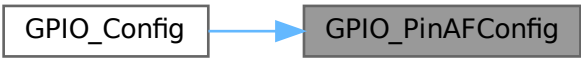
配置GPIO引脚复用功能

参数

GPIOx	GPIO模块指针
GPIO_PinSource	引脚源
GPIO_AF	复用功能选择

在文件 [lks32mc09x_gpio.c](#) 第 223 行定义.

这是这个函数的调用关系图:



3.75.1.7 GPIO_ReadInputData()

```
uint32_t GPIO_ReadInputData (
    GPIO_TypeDef * GPIOx )
```

读取GPIO端口的输入数据

参数

GPIOx	GPIO模块指针

返回

GPIO端口的输入数据

在文件 [lks32mc09x_gpio.c](#) 第 143 行定义.

3.75.1.8 GPIO_ReadInputDataBit()

```
uint8_t GPIO_ReadInputDataBit (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

读取指定GPIO引脚的输入数据位

参数

GPIOx	GPIO模块指针
GPIO_Pin	要读取的引脚

返回

引脚的输入数据位状态

在文件 [lks32mc09x_gpio.c](#) 第 133 行定义.

3.75.1.9 GPIO_ReadOutputData()

```
uint32_t GPIO_ReadOutputData (
    GPIO_TypeDef * GPIOx )
```

读取GPIO端口的输出数据

参数

GPIOx	GPIO模块指针

返回

GPIO端口的输出数据

在文件 [lks32mc09x_gpio.c](#) 第 164 行定义.

3.75.1.10 GPIO_ReadOutputDataBit()

```
uint8_t GPIO_ReadOutputDataBit (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

读取指定GPIO引脚的输出数据位

参数

GPIOx	GPIO模块指针
GPIO_Pin	要读取的引脚

返回

引脚的输出数据位状态

在文件 [lks32mc09x_gpio.c](#) 第 154 行定义.

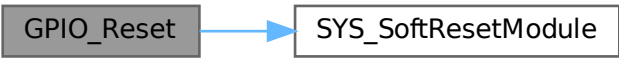
3.75.1.11 GPIO_Reset()

```
void GPIO_Reset (
    void )
```

gpio模块复位

在文件 [lks32mc09x_gpio.c](#) 第 21 行定义.

函数调用图:



3.75.1.12 GPIO_ResetBits()

```
void GPIO_ResetBits (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

复位指定的GPIO引脚位

参数

GPIOx	GPIO模块指针
GPIO_Pin	要复位的引脚

在文件 [lks32mc09x_gpio.c](#) 第 184 行定义.

3.75.1.13 GPIO_SetBits()

```
void GPIO_SetBits (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

设置指定的GPIO引脚位

参数

GPIOx	GPIO模块指针
GPIO_Pin	要设置的引脚

在文件 [lks32mc09x_gpio.c](#) 第 174 行定义.

3.75.1.14 GPIO_StructInit()

```
void GPIO_StructInit (
    GPIO_InitTypeDef * GPIO_InitStruct )
```

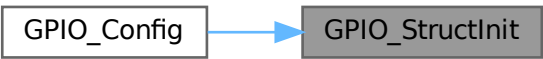
初始化GPIO结构体为默认值

参数

<i>GPIO_InitStruct</i>	指向要初始化的GPIO_InitTypeDef结构体

在文件 [lks32mc09x_gpio.c](#) 第 29 行定义.

这是这个函数的调用关系图:



3.75.1.15 GPIO_ToggleBits()

```
void GPIO_ToggleBits (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

翻转指定的GPIO引脚位

参数

<i>GPIOx</i>	GPIO模块指针
<i>GPIO_Pin</i>	要翻转的引脚

在文件 [lks32mc09x_gpio.c](#) 第 252 行定义.

3.75.1.16 GPIO_UnlockRegister()

```
void GPIO_UnlockRegister (
    GPIO_TypeDef * GPIOx )
```

解锁GPIO寄存器

参数

<i>GPIOx</i>	GPIO模块指针

在文件 [lks32mc09x_gpio.c](#) 第 118 行定义.

3.75.1.17 GPIO_Write()

```
void GPIO_Write (
    GPIO_TypeDef * GPIOx,
    uint32_t Val )
```

写入数据到指定的GPIO端口

参数

<i>GPIOx</i>	GPIO模块指针
<i>Val</i>	要写入的数据

在文件 [lks32mc09x_gpio.c](#) 第 194 行定义.

3.75.1.18 GPIO_WriteBit()

```
void GPIO_WriteBit (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin,
    GPIO_Bit_Action BitVal )
```

设置或复位指定的GPIO引脚位

参数

<i>GPIOx</i>	GPIO模块指针
<i>GPIO_Pin</i>	要操作的引脚
<i>BitVal</i>	要执行的操作: GPIO_Bit_SET 或 GPIO_Bit_RESET

在文件 [lks32mc09x_gpio.c](#) 第 205 行定义.

3.76 lks32mc09x_gpio.c

[浏览该文件的文档.](#)

```
00001 #include "lks32mc09x_gpio.h"
00002 #include "lks32mc09x_sys.h"
00003
00007 void GPIO_Enable(void)
00008 {
00009     SYS_ModuleClockCmd(SYS_MODULE_GPIO, 1);
00010 }
00014 void GPIO_Disable(void)
00015 {
00016     SYS_ModuleClockCmd(SYS_MODULE_GPIO, 0);
00017 }
00021 void GPIO_Reset(void)
00022 {
00023     SYS_SoftResetModule(SYS_MODULE_GPIO);
00024 }
00029 void GPIO_StructInit(GPIO_InitTypeDef *GPIO_InitStruct)
00030 {
00031     uint32_t i;
00032     for (i = 0; i < sizeof(GPIO_InitTypeDef); i++)
00033     {
00034         ((uint8_t *)GPIO_InitStruct)[i] = 0;
```

```

00035     }
00036 }
00037
00043 void GPIO_Init(GPIO_TypeDef *GPIOx, GPIO_InitTypeDef *GPIO_InitStruct)
00044 {
00045     // 使能GPIO时钟
00046     SYS_ModuleClockCmd(SYS_MODULE_GPIO, ENABLE);
00047
00048     // 配置引脚模式
00049     if (GPIO_InitStruct->GPIO_Mode == GPIO_Mode_IN)
00050     {
00051         GPIOx->PIE |= GPIO_InitStruct->GPIO_Pin; // 打开输入使能
00052         GPIOx->POE &= ~GPIO_InitStruct->GPIO_Pin; // 关闭输出使能
00053         GPIOx->PUE &= ~GPIO_InitStruct->GPIO_Pin; // 关闭上拉使能
00054     }
00055     else if (GPIO_InitStruct->GPIO_Mode == GPIO_Mode_OUT)
00056     {
00057         GPIOx->PIE &= ~GPIO_InitStruct->GPIO_Pin; // 关闭输入使能
00058         GPIOx->POE |= GPIO_InitStruct->GPIO_Pin; // 使能输出
00059     }
00060     else if (GPIO_InitStruct->GPIO_Mode == GPIO_Mode_IO)
00061     {
00062         GPIOx->PIE |= GPIO_InitStruct->GPIO_Pin; // 使能输入
00063         GPIOx->POE |= GPIO_InitStruct->GPIO_Pin; // 使能输出
00064     }
00065     else // GPIO_Mode_ANA
00066     {
00067         GPIOx->PIE &= ~GPIO_InitStruct->GPIO_Pin; // 关闭输入使能
00068         GPIOx->POE &= ~GPIO_InitStruct->GPIO_Pin; // 关闭输出使能
00069         GPIOx->PUE &= ~GPIO_InitStruct->GPIO_Pin; // 关闭上拉使能
00070     }
00071
00072     if (GPIO_InitStruct->GPIO_PuPd == GPIO_PuPd_UP)
00073     {
00074         GPIOx->PUE |= GPIO_InitStruct->GPIO_Pin; // 使能上拉
00075     }
00076     else
00077     {
00078         GPIOx->PUE &= ~GPIO_InitStruct->GPIO_Pin; // 关闭上拉
00079     }
00080     // 配置开漏使能
00081     if (GPIO_InitStruct->GPIO_PODENa)
00082     {
00083         GPIOx->PODE |= GPIO_InitStruct->GPIO_Pin;
00084     }
00085     else
00086     {
00087         GPIOx->PODE &= ~GPIO_InitStruct->GPIO_Pin;
00088     }
00089
00090     // 配置滤波使能
00091     if (GPIO_InitStruct->GPIO_PFLT)
00092     {
00093         GPIOx->PFLT |= GPIO_InitStruct->GPIO_Pin;
00094     }
00095     else
00096     {
00097         GPIOx->PFLT &= ~GPIO_InitStruct->GPIO_Pin;
00098     }
00099 }
00100
00105 void GPIO_LockRegister(GPIO_TypeDef *GPIOx)
00106 {
00107     // 09x 无此功能
00108     if (GPIOx == GPIO0)
00109     {
00110         __NOP();
00111     }
00112 }
00113
00118 void GPIO_UnlockRegister(GPIO_TypeDef *GPIOx)
00119 {
00120     // 09x 无此功能
00121     if (GPIOx == GPIO0)
00122     {
00123         __NOP();
00124     }
00125 }
00126
00133 uint8_t GPIO_ReadInputDataBit(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
00134 {
00135     return (GPIOx->PDI & GPIO_Pin) ? 1 : 0;
00136 }
00137
00143 uint32_t GPIO_ReadInputData(GPIO_TypeDef *GPIOx)
00144 {
00145     return GPIOx->PDI;

```

```

00146 }
00147
00154 uint8_t GPIO_ReadOutputDataBit(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
00155 {
00156     return (GPIOx->PDO & GPIO_Pin) ? 1 : 0;
00157 }
00158
00164 uint32_t GPIO_ReadOutputData(GPIO_TypeDef *GPIOx)
00165 {
00166     return GPIOx->PDO;
00167 }
00168
00174 void GPIO_SetBits(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
00175 {
00176     GPIOx->PDO |= GPIO_Pin;
00177 }
00178
00184 void GPIO_ResetBits(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
00185 {
00186     GPIOx->PDO &= ~GPIO_Pin;
00187 }
00188
00194 void GPIO_Write(GPIO_TypeDef *GPIOx, uint32_t Val)
00195 {
00196     GPIOx->PDO = Val;
00197 }
00198
00205 void GPIO_WriteBit(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin, GPIO_BitAction BitVal)
00206 {
00207     if (BitVal == GPIO_Bit_SET)
00208     {
00209         GPIOx->PDO |= GPIO_Pin;
00210     }
00211     else
00212     {
00213         GPIOx->PDO &= ~GPIO_Pin;
00214     }
00215 }
00216
00223 void GPIO_PinAFConfig(GPIO_TypeDef *GPIOx, uint32_t GPIO_PinSource, uint32_t GPIO_AF)
00224 {
00225     uint8_t offset;
00226     uint8_t pins = GPIO_PinSource >> 2;
00227     offset = ((GPIO_PinSource & 0x3) * 4);
00228     switch (pins)
00229     {
00230         case 0:
00231             GPIOx->F3210 = (GPIOx->F3210 & ~(0xf << offset)) | (GPIO_AF << offset);
00232             break;
00233         case 1:
00234             GPIOx->F7654 = (GPIOx->F7654 & ~(0xf << offset)) | (GPIO_AF << offset);
00235             break;
00236         case 2:
00237             GPIOx->FBA98 = (GPIOx->FBA98 & ~(0xf << offset)) | (GPIO_AF << offset);
00238             break;
00239         case 3:
00240             GPIOx->FFEDC = (GPIOx->FFEDC & ~(0xf << offset)) | (GPIO_AF << offset);
00241             break;
00242         default:
00243             break;
00244     }
00245 }
00246
00252 void GPIO_ToggleBits(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
00253 {
00254     GPIOx->PDO = GPIOx->PDO ^ GPIO_Pin;
00255 }
00256
00264 void GPIO_Config(GPIO_TypeDef *GPIOx, uint32_t GPIO_PinSource, GPIO_Mode_TypeDef mode, uint32_t
GPIO_AF_x)
00265 {
00266     GPIO_PinAFConfig(GPIOx, GPIO_PinSource, GPIO_AF_x);
00267     GPIO_InitTypeDef GPIO_InitStructure;
00268     GPIO_StructInit(&GPIO_InitStructure);
00269     GPIO_InitStructure.GPIO_Pin = 1 << GPIO_PinSource;
00270     GPIO_InitStructure.GPIO_Mode = mode;
00271     if (GPIO_AF_x == GPIO_AF_I2C)
00272     {
00273         GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
00274     }
00275     else
00276     {
00277         __NOP();
00278     }
00279     GPIO_Init(GPIOx, &GPIO_InitStructure);
00280 }

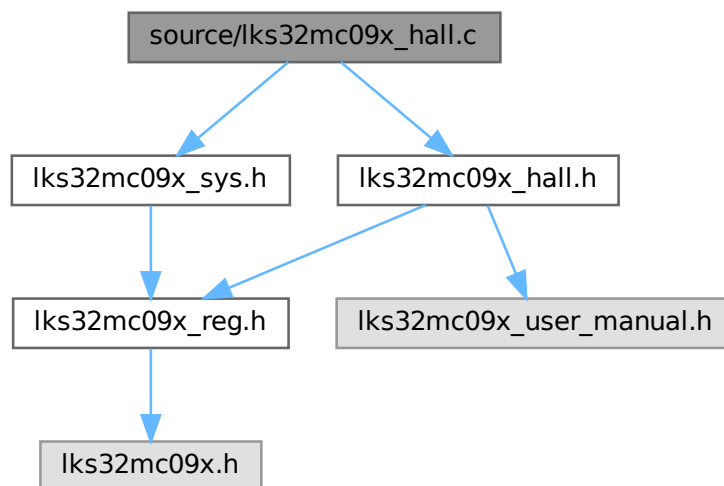
```

3.77 source/lks32mc09x_hall.c 文件参考

```
#include "lks32mc09x_hall.h"
```

```
#include "lks32mc09x_sys.h"
```

lks32mc09x_hall.c 的引用(Include)关系图:



函数

- void **HALL_StructInit** (HALL_InitTypeDef *HALL_InitStruct)
初始化HALL结构体
- void **HALL_Init** (HALL_InitTypeDef *HALL_InitStruct)
初始化HALL模块
- void **HALL_Cmd** (uint8_t state)
使能或失能HALL模块
- uint32_t **HALL_GetFilterValue** (void)
读取滤波后的HALL值
- uint32_t **HALL_GetCaptureValue** (void)
读取未滤波的HALL值
- uint32_t **HALL_GetCount** (void)
读取HALL计数值
- uint32_t **HALL_IsCaptureEvent** (void)
判断是否发生捕获事件
- uint32_t **HALL_IsOverflowEvent** (void)
判断是否发生溢出事件
- uint32_t **HALL_GetFlag** (uint32_t tempFlag)
获取HALL事件标志
- void **HALL_Clear_IRQ** (uint32_t tempFlag)
清除HALL中断标志位
- uint32_t **HALL_GetIRQFlag** (uint32_t tempFlag)
读取HALL中断标志位
- void **HALL_SoftTrigger** (void)
软件触发HALL信号变化中断

3.77.1 函数说明

3.77.1.1 HALL_Clear_IRQ()

```
void HALL_Clear_IRQ (
    uint32_t tempFlag )
```

清除HALL中断标志位

参数

<i>tempFlag</i>	要清除的标志位

在文件 [lks32mc09x_hall.c](#) 第 120 行定义.

3.77.1.2 HALL_Cmd()

```
void HALL_Cmd (
    uint8_t state )
```

使能或失能HALL模块

参数

<i>state</i>	功能状态，ENABLE或DISABLE

在文件 [lks32mc09x_hall.c](#) 第 49 行定义.

3.77.1.3 HALL_GetCaptureValue()

```
uint32_t HALL_GetCaptureValue (
    void )
```

读取未滤波的HALL值

返回

未滤波的HALL值

在文件 [lks32mc09x_hall.c](#) 第 74 行定义.

3.77.1.4 HALL_GetCount()

```
uint32_t HALL_GetCount (
    void )
```

读取HALL计数值

返回

HALL计数值

在文件 [lks32mc09x_hall.c](#) 第 83 行定义.

3.77.1.5 HALL_GetFilterValue()

```
uint32_t HALL_GetFilterValue (
    void )
```

读取滤波后的HALL值

返回

滤波后的HALL值

在文件 [lks32mc09x_hall.c](#) 第 65 行定义.

3.77.1.6 HALL_GetFlag()

```
uint32_t HALL_GetFlag (
    uint32_t tempFlag )
```

获取HALL事件标志

参数

<i>tempFlag</i>	要获取的标志位

返回

对应标志位的状态

在文件 [lks32mc09x_hall.c](#) 第 111 行定义.

3.77.1.7 HALL_GetIRQFlag()

```
uint32_t HALL_GetIRQFlag (
    uint32_t tempFlag )
```

读取HALL中断标志位

参数

<i>tempFlag</i>	要读取的标志位

返回

对应标志位的状态

在文件 [lks32mc09x_hall.c](#) 第 130 行定义.

3.77.1.8 HALL_Init()

```
void HALL_Init (
    HALL_InitTypeDef * HALL_InitStruct )
```

初始化HALL模块

参数

<i>HALL_InitStruct</i>	指向HALL_InitTypeDef结构体的指针

在文件 [lks32mc09x_hall.c](#) 第 21 行定义.

函数调用图:



3.77.1.9 HALL_IsCaptureEvent()

```
uint32_t HALL_IsCaptureEvent (
    void )
```

判断是否发生捕获事件

返回

发生捕获事件返回非零值（True），否则返回0（False）

在文件 [lks32mc09x_hall.c](#) 第 92 行定义.

3.77.1.10 HALL_IsOverFlowEvent()

```
uint32_t HALL_IsOverFlowEvent (
    void )
```

判断是否发生溢出事件

返回

发生溢出事件返回非零值（True），否则返回0（False）

在文件 [lks32mc09x_hall.c](#) 第 101 行定义.

3.77.1.11 HALL_SoftTrigger()

```
void HALL_SoftTrigger (
    void )
```

软件触发HALL信号变化中断

在文件 [lks32mc09x_hall.c](#) 第 138 行定义.

3.77.1.12 HALL_StructInit()

```
void HALL_StructInit (
    HALL_InitTypeDef * HALL_InitStruct )
```

初始化HALL结构体

参数

<i>HALL_InitStruct</i>	指向HALL_InitTypeDef结构体的指针

在文件 [lks32mc09x_hall.c](#) 第 8 行定义.

3.78 lks32mc09x_hall.c

[浏览该文件的文档.](#)

```
00001 #include "lks32mc09x_hall.h"
00002 #include "lks32mc09x_sys.h"
00003
00008 void HALL_StructInit(HALL_InitTypeDef *HALL_InitStruct)
00009 {
00010     uint32_t i;
00011     for (i = 0; i < sizeof(HALL_InitTypeDef); i++)
00012     {
00013         ((uint8_t *)HALL_InitStruct)[i] = 0;
00014     }
00015 }
00016
00021 void HALL_Init(HALL_InitTypeDef *HALL_InitStruct)
00022 {
00023     if (HALL_InitStruct != 0)
00024     {
00025         // 使能HALL模块时钟
00026         SYS_ModuleClockCmd(SYS_MODULE_HALL0, HALL_InitStruct->HALL_Ena);
00027
00028         // 霍尔配置寄存器
00029         REG_IF_WRITE(HALL0->CFG, HALL0_CFG_FIL75_BIT, HALL_InitStruct->Filter75_Ena);
00030         REG_WRITE_BIT_POS_MASK(HALL0->CFG, HALL0_CFG_CLK_DIV, HALL_InitStruct->ClockDivision);
00031         REG_WRITE_BIT_POS_MASK(HALL0->CFG, HALL0_CFG_FIL_LEN, HALL_InitStruct->FilterLen);
00032         REG_IF_WRITE(HALL0->CFG, HALL0_CFG_CHG_IE_BIT, HALL_InitStruct->Capture_IRQ_Ena);
00033         REG_IF_WRITE(HALL0->CFG, HALL0_CFG_OV_IE_BIT, HALL_InitStruct->OverFlow_IRQ_Ena);
00034         REG_IF_WRITE(HALL0->CFG, HALL0_CFG_SW_IE_BIT, HALL_InitStruct->softIE);
00035         REG_IF_WRITE(HALL0->CFG, HALL0_CFG_HALL_EN_BIT, HALL_InitStruct->HALL_Ena);
00036
00037         REG_WRITE(HALL0->TH, HALL_InitStruct->CountTH);
00038     }
00039     else
00040     {
00041         __NOP();
00042     }
00043 }
00044
00049 void HALL_Cmd(uint8_t state)
00050 {
00051     if (state == ENABLE)
```



```

00052     {
00053         HALLO->CFG |= (1 << 24); // 使能HALL模块
00054     }
00055     else
00056     {
00057         HALLO->CFG &= ~(1 << 24); // 失能HALL模块
00058     }
00059 }
00060
00065 uint32_t HALL_GetFilterValue(void)
00066 {
00067     return (HALLO->INFO & 0x07);
00068 }
00069
00074 uint32_t HALL_GetCaptureValue(void)
00075 {
00076     return (HALLO->INFO >> 8) & 0x07;
00077 }
00078
00083 uint32_t HALL_GetCount(void)
00084 {
00085     return HALLO->CNT;
00086 }
00087
00092 uint32_t HALL_IsCaptureEvent(void)
00093 {
00094     return (HALLO->INFO & HALL_CAPTURE_EVENT) ? 1 : 0;
00095 }
00096
00101 uint32_t HALL_IsOverFlowEvent(void)
00102 {
00103     return (HALLO->INFO & HALL_OVERFLOW_EVENT) ? 1 : 0;
00104 }
00105
00111 uint32_t HALL_GetFlag(uint32_t tempFlag)
00112 {
00113     return (HALLO->INFO & tempFlag) ? 1 : 0;
00114 }
00115
00120 void HALL_Clear_IRQ(uint32_t tempFlag)
00121 {
00122     HALLO->INFO &= ~tempFlag;
00123 }
00124
00130 uint32_t HALL_GetIRQFlag(uint32_t tempFlag)
00131 {
00132     return (HALLO->INFO & tempFlag) ? 1 : 0;
00133 }
00134
00138 void HALL_SoftTrigger(void)
00139 {
00140     HALLO->INFO |= (1 << 18); // 触发HALL信号变化中断
00141 }

```

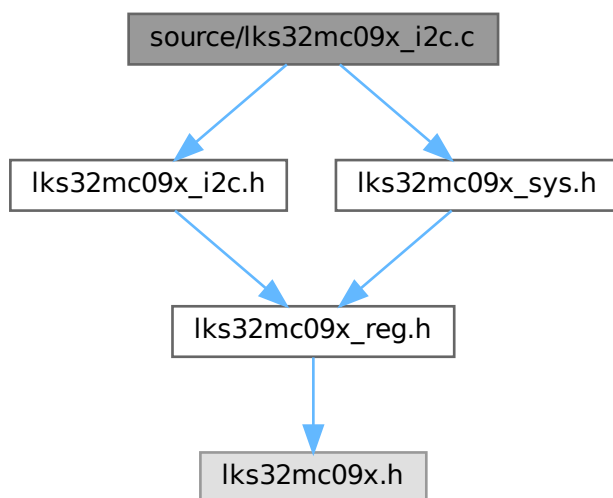
3.79 source/lks32mc09x_i2c.c 文件参考

```

#include "lks32mc09x_i2c.h"
#include "lks32mc09x_sys.h"

```

lks32mc09x_i2c.c 的引用(Include)关系图:



结构体

- struct [iictmpval](#)

函数

- void [I2C_Init](#) ([I2C_TypeDef](#) *I2Cx, [I2C_InitTypeDef](#) *I2C_InitStruct)
I2C初始化
- void [I2C_StructInit](#) ([I2C_InitTypeDef](#) *I2C_InitStruct)
I2C配置结构体初始化
- void [I2C_ClearIRQFlag](#) ([I2C_TypeDef](#) *I2Cx, uint16_t I2C_IF_x)
清除I2C中断标志
- uint8_t [I2C_GetIRQFlag](#) ([I2C_TypeDef](#) *I2Cx, uint16_t I2C_IF_x)
获取I2C中断标志状态
- uint8_t [I2C_GetFlag](#) ([I2C_TypeDef](#) *I2Cx, uint16_t I2C_IF_x)
获取I2C状态标志
- void [I2C_Enable](#) ([I2C_TypeDef](#) *I2Cx)
使能I2C外设
- void [I2C_Disable](#) ([I2C_TypeDef](#) *I2Cx)
禁用I2C外设
- void [I2C_SetBaudRate](#) ([I2C_TypeDef](#) *I2Cx, uint32_t BaudRate)
设置I2C波特率
- uint8_t [I2C_ReadBusState](#) ([I2C_TypeDef](#) *I2Cx, uint16_t state)
读取I2C总线状态
- void [I2C_ClearBusState](#) ([I2C_TypeDef](#) *I2Cx, uint16_t state)
清除I2C总线状态
- void [I2C_SendData](#) ([I2C_TypeDef](#) *I2Cx, uint8_t n)

- 发送数据到I2C总线
- `uint8_t I2C_ReadData (I2C_TypeDef *I2Cx)`
从I2C总线读取数据
 - `int I2C_TableSend (I2C_TypeDef *I2Cx, u8 taddr, u8 *data, u8 len)`
向从机发送数据 （需要在中断里执行I2C_TableIrq）
 - `int I2C_TableRead (I2C_TypeDef *I2Cx, u8 taddr, u8 *data, u8 len)`
向从机读取数据 （需要在中断里执行I2C_TableIrq）
 - `int I2C_TableAck (I2C_TypeDef *I2Cx)`
获取I2C_Table发送后的ack电平
 - `int I2C_TableState (I2C_TypeDef *I2Cx)`
获取I2C_Table发送状态
 - `void I2C_TableIrq (I2C_TypeDef *I2Cx)`
I2C_SendTable和I2C_ReadTable对应的中断处理函数

变量

- `static volatile struct iictmpval I2C_TableVariable [1] = {0}`

3.79.1 函数说明

3.79.1.1 I2c_ClearBusState()

```
void I2c_ClearBusState (
    I2C_TypeDef * I2Cx,
    uint16_t state )
```

清除I2C总线状态

参数

I2Cx	I2C外设指针
state	要清除的状态位

在文件 `lks32mc09x_i2c.c` 第 207 行定义.

3.79.1.2 I2C_ClearIRQFlag()

```
void I2C_ClearIRQFlag (
    I2C_TypeDef * I2Cx,
    uint16_t I2C_IF_x )
```

清除I2C中断标志

参数

I2Cx	I2C外设指针
I2C_↔IF_x	要清除的中断标志

在文件 `lks32mc09x_i2c.c` 第 43 行定义.

3.79.1.3 I2C_Disable()

```
void I2C_Disable (
    I2C_TypeDef * I2Cx )
```

禁用I2C外设

参数

I2Cx	I2C外设指针

在文件 `lks32mc09x_i2c.c` 第 161 行定义.

函数调用图:



3.79.1.4 I2C_Enable()

```
void I2C_Enable (
    I2C_TypeDef * I2Cx )
```

使能I2C外设

参数

I2Cx	I2C外设指针

在文件 `lks32mc09x_i2c.c` 第 149 行定义.

函数调用图:



3.79.1.5 I2C_GetFlag()

```
uint8_t I2C_GetFlag (
    I2C_TypeDef * I2Cx,
    uint16_t I2C_IF_x )
```

获取I2C状态标志

参数

I2Cx	I2C外设指针
I2C_↔IF_x	要获取的状态标志

返回

状态标志状态

在文件 [lks32mc09x_i2c.c](#) 第 84 行定义.

这是这个函数的调用关系图:



3.79.1.6 I2C_GetIRQFlag()

```
uint8_t I2C_GetIRQFlag (
    I2C_TypeDef * I2Cx,
    uint16_t I2C_IF_x )
```

获取I2C中断标志状态

参数

I2Cx	I2C外设指针
I2C_↔IF_x	要获取的中断标志

返回

中断标志状态

在文件 `lks32mc09x_i2c.c` 第 73 行定义.

函数调用图:



3.79.1.7 I2C_Init()

```
void I2C_Init (
    I2C_TypeDef * I2Cx,
    I2C_InitTypeDef * I2C_InitStruct )
```

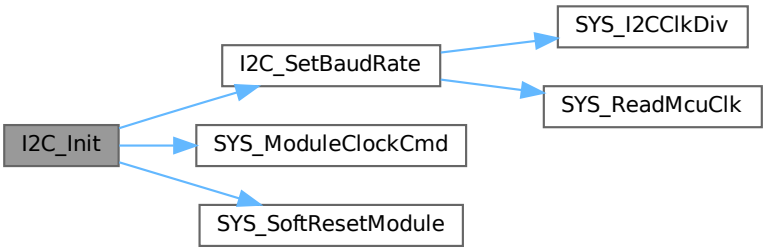
I2C初始化

参数

<i>I2Cx</i>	I2C外设指针
<i>I2C_InitStruct</i>	I2C初始化结构体指针

在文件 `lks32mc09x_i2c.c` 第 9 行定义.

函数调用图:



3.79.1.8 I2c_ReadBusState()

```
uint8_t I2c_ReadBusState (
    I2C_TypeDef * I2Cx,
    uint16_t state )
```

读取I2C总线状态

参数

<i>I2Cx</i>	I2C外设指针
<i>state</i>	要读取的状态位

返回
状态位值

在文件 [lks32mc09x_i2c.c](#) 第 190 行定义.

3.79.1.9 I2C_ReadData()

```
uint8_t I2C_ReadData (
    I2C_TypeDef * I2Cx )
```

从I2C总线读取数据

参数

<i>I2Cx</i>	I2C外设指针

返回
读取到的数据

在文件 [lks32mc09x_i2c.c](#) 第 227 行定义.

3.79.1.10 I2C_SendData()

```
void I2C_SendData (
    I2C_TypeDef * I2Cx,
    uint8_t n )
```

发送数据到I2C总线

参数

<i>I2Cx</i>	I2C外设指针
<i>n</i>	要发送的数据

在文件 [lks32mc09x_i2c.c](#) 第 217 行定义.

3.79.1.11 I2C_SetBaudRate()

```
void I2C_SetBaudRate (
```

```
I2C_TypeDef * I2Cx,  
uint32_t BaudRate )
```

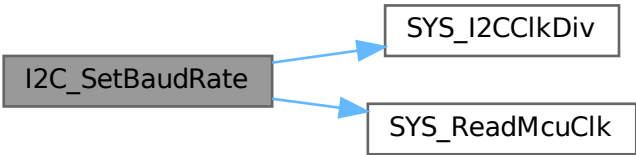
设置I2C波特率

参数

<i>BaudRate</i>	波特率值

在文件 [lks32mc09x_i2c.c](#) 第 173 行定义.

函数调用图:



这是这个函数的调用关系图:



3.79.1.12 I2C_StructInit()

```
void I2C_StructInit (  
    I2C_InitTypeDef * I2C_InitStruct )
```

I2C配置结构体初始化

参数

<i>I2C_InitStruct</i>	I2C初始化结构体指针

在文件 [lks32mc09x_i2c.c](#) 第 29 行定义.

3.79.1.13 I2C_TableAck()

```
int I2C_TableAck (
    I2C_TypeDef * I2Cx )
```

获取I2C_Table发送后的ack电平

返回

收到的ACK电平

在文件 [lks32mc09x_i2c.c](#) 第 291 行定义.

3.79.1.14 I2C_TableIrq()

```
void I2C_TableIrq (
    I2C_TypeDef * I2Cx )
```

I2C_SendTable和I2C_ReadTable对应的中断处理函数

在文件 [lks32mc09x_i2c.c](#) 第 320 行定义.

3.79.1.15 I2C_TableRead()

```
int I2C_TableRead (
    I2C_TypeDef * I2Cx,
    u8 taddr,
    u8 * data,
    u8 len )
```

向从机读取数据（需要在中断里执行I2C_TableIrq）

参数

<i>I2Cx</i>	I2C外设指针
<i>taddr</i>	从机地址
<i>data</i>	数据指针
<i>len</i>	数据长度

返回

状态，0成功 1失败（i2c总线非空闲，或上一次数据尚未发送完毕）

在文件 [lks32mc09x_i2c.c](#) 第 272 行定义.

3.79.1.16 I2C_TableSend()

```
int I2C_TableSend (
    I2C_TypeDef * I2Cx,
```

```
u8 taddr,
u8 * data,
u8 len )
```

向从机发送数据（需要在中断里执行I2C_TableIrq）

参数

<i>I2Cx</i>	I2C外设指针
<i>taddr</i>	从机地址
<i>data</i>	数据指针
<i>len</i>	数据长度

返回

状态，0成功 1失败（i2c总线非空闲，或上一次数据尚未发送完毕）

在文件 [lks32mc09x_i2c.c](#) 第 251 行定义.

3.79.1.17 I2C_TableState()

```
int I2C_TableState (
    I2C_TypeDef * I2Cx )
```

获取I2C_Table发送状态

返回

状态，0空闲 1正在发送或接收数据

在文件 [lks32mc09x_i2c.c](#) 第 306 行定义.

3.79.2 变量说明

3.79.2.1 I2C_TableVariable

```
volatile struct iictmpval I2C_TableVariable[1] = {0} [static]
```

在文件 [lks32mc09x_i2c.c](#) 第 241 行定义.

3.80 lks32mc09x_i2c.c

[浏览该文件的文档。](#)

```

00001 #include "lks32mc09x_i2c.h"
00002 #include "lks32mc09x_sys.h"
00003
00009 void I2C_Init(I2C_TypeDef *I2Cx, I2C_InitTypeDef *I2C_InitStruct)
00010 {
00011     SYS_ModuleClockCmd(SYS_MODULE_I2C0, ENABLE);
00012     SYS_SoftResetModule(SYS_MODULE_I2C0);
00013     I2C_SetBaudRate(I2Cx, I2C_InitStruct->BaudRate);
00014     I2Cx->ADDR = (I2C_InitStruct->ADRCMP << 7);
00015     I2Cx->CFG = (I2C_InitStruct->IE << 7) | (I2C_InitStruct->TC_IE << 6) |
00016         (I2C_InitStruct->BUS_ERR_IE << 5) | (I2C_InitStruct->STOP_IE << 4) |
00017         (I2C_InitStruct->MST_MODE << 1) | (I2C_InitStruct->SLV_MODE);
00018     I2Cx->SCR = 0;
00019     I2Cx->DATA = 0;
00020     I2Cx->MSCR = 0;
00021     I2Cx->BCR = (I2C_InitStruct->BURST_NACK << 7) | (I2C_InitStruct->BURST_ADDR_CMP << 6) |
00022         (I2C_InitStruct->BUSRT_EN << 5);
00023 }
00024
00029 void I2C_StructInit(I2C_InitTypeDef *I2C_InitStruct)
00030 {
00031     uint32_t i;
00032     for (i = 0; i < sizeof(I2C_InitTypeDef); i++)
00033     {
00034         ((uint8_t *)I2C_InitStruct)[i] = 0;
00035     }
00036 }
00037
00043 void I2C_ClearIRQFlag(I2C_TypeDef *I2Cx, uint16_t I2C_IF_x)
00044 {
00045     switch (I2C_IF_x)
00046     {
00047         case I2C_IF_TC: // 传输完成中断
00048             I2Cx->SCR &= ~BIT0;
00049             break;
00050         case I2C_IF_BUS_ERR: // 总线错误中断
00051             I2Cx->SCR &= ~BIT7;
00052             break;
00053         case I2C_IF_STOP: // STOP事件中断
00054             I2Cx->SCR &= ~BIT5;
00055             break;
00056         case I2C_IF_NACK: // NACK事件中断
00057             I2Cx->SCR &= ~BIT1;
00058             break;
00059         case I2C_IF_ADDR_CMP: // 硬件地址比较中断
00060             break;
00061         default:
00062             break;
00063     }
00064 }
00065
00073 uint8_t I2C_GetIRQFlag(I2C_TypeDef *I2Cx, uint16_t I2C_IF_x)
00074 {
00075     return I2C_GetFlag(I2Cx, I2C_IF_x);
00076 }
00077
00084 uint8_t I2C_GetFlag(I2C_TypeDef *I2Cx, uint16_t I2C_IF_x)
00085 {
00086     switch (I2C_IF_x)
00087     {
00088         case I2C_IF_TC: // 传输完成中断
00089             if (I2Cx->SCR & BIT0)
00090             {
00091                 return 1;
00092             }
00093             else
00094             {
00095                 return 0;
00096             }
00097         case I2C_IF_BUS_ERR: // 总线错误中断
00098             if (I2Cx->SCR & BIT7)
00099             {
00100                 return 1;
00101             }
00102             else
00103             {
00104                 return 0;
00105             }
00106         case I2C_IF_STOP: // STOP事件中断
00107             if (I2Cx->SCR & BIT5)
00108             {

```

```

00109         return 1;
00110     }
00111     else
00112     {
00113         return 0;
00114     }
00115     case I2C_IF_NACK: // NACK事件中断
00116     {
00117         if (I2Cx->SCR & BIT1)
00118         {
00119             return 1;
00120         }
00121         else
00122         {
00123             return 0;
00124         }
00125     }
00126     case I2C_IF_ADDR_CMP: // 硬件地址匹配中断
00127     {
00128         if (I2Cx->SCR & BIT3)
00129         {
00130             if ((u32)(I2Cx->DATA >> 1) == (I2Cx->ADDR & 0x7f))
00131             {
00132                 return 1;
00133             }
00134             else
00135             {
00136                 return 0;
00137             }
00138         }
00139         else
00140         {
00141             return 0;
00142         }
00143     }
00144     default:
00145     {
00146         return 0;
00147     }
00148 }
00149 void I2C_Enable(I2C_TypeDef *I2Cx)
00150 {
00151     if (I2Cx == I2C0)
00152     {
00153         SYS_ModuleClockCmd(SYS_MODULE_I2C0, ENABLE);
00154     }
00155 }
00156 void I2C_Disable(I2C_TypeDef *I2Cx)
00157 {
00158     if (I2Cx == I2C0)
00159     {
00160         SYS_ModuleClockCmd(SYS_MODULE_I2C0, DISABLE);
00161     }
00162 }
00163 void I2C_SetBaudRate(I2C_TypeDef *I2Cx, uint32_t BaudRate)
00164 {
00165     if (I2Cx == I2C0)
00166     {
00167         // 根据系统时钟和波特率计算分频值
00168         u32 div;
00169         div = ((u32)SYS_ReadMcUClk() + ((BaudRate * 17) >> 1)) / (BaudRate * 17) - 1;
00170         SYS_I2CClkDiv(div);
00171     }
00172 }
00173 uint8_t I2c_ReadBusState(I2C_TypeDef *I2Cx, uint16_t state)
00174 {
00175     if (I2Cx->SCR & state)
00176     {
00177         return ENABLE;
00178     }
00179     else
00180     {
00181         return DISABLE;
00182     }
00183 }
00184 void I2c_ClearBusState(I2C_TypeDef *I2Cx, uint16_t state)
00185 {
00186     I2Cx->SCR &= ~state;
00187 }
00188 void I2C_SendData(I2C_TypeDef *I2Cx, uint8_t n)
00189 {
00190     I2Cx->DATA = n;
00191 }
00192 uint8_t I2C_ReadData(I2C_TypeDef *I2Cx)
00193 {

```

```

00229     return I2Cx->DATA;
00230 }
00231 // 仅用于Table收发的变量
00232 struct iictmpval
00233 {
00234     volatile uint8_t *txdata;
00235     volatile uint8_t txdatap;
00236     volatile uint8_t *rxdata;
00237     volatile uint8_t rxdatap;
00238     volatile uint8_t state; // 0 无动作 1接收 2发送
00239     volatile uint8_t nackflg; // 0 通信正常 1 发生nack事件
00240 };
00241 static volatile struct iictmpval I2C_TableVariable[1] = {0}; // 只有一个tab
00242
00251 int I2C_TableSend(I2C_TypeDef *I2Cx, u8 taddr, u8 *data, u8 len)
00252 {
00253     I2C_TableVariable[0].nackflg = 0;
00254     while (I2C_TableVariable[0].state != 0);
00255     I2Cx->SCR = 0;
00256     I2C_TableVariable[0].txdata = data;
00257     I2C_TableVariable[0].txdatap = len;
00258     I2C_TableVariable[0].state = 2;
00259     I2Cx->DATA = taddr;
00260     I2Cx->MSCR = 1; // 触发I2C发送数据
00261     return 0;
00262 }
00263
00272 int I2C_TableRead(I2C_TypeDef *I2Cx, u8 taddr, u8 *data, u8 len)
00273 {
00274     if (I2Cx == I2C0)
00275     {
00276         I2C_TableVariable[0].nackflg = 0;
00277         while (I2C_TableVariable[0].state != 0);
00278         I2Cx->SCR = 0;
00279         I2C_TableVariable[0].rxdata = data;
00280         I2C_TableVariable[0].rxdatap = len;
00281         I2C_TableVariable[0].state = 1;
00282         I2Cx->DATA = taddr | 1;
00283         I2Cx->MSCR = 1; // 触发I2C发送数据
00284     }
00285     return 0;
00286 }
00291 int I2C_TableAck(I2C_TypeDef *I2Cx)
00292 {
00293     if (I2Cx == I2C0)
00294     {
00295         return I2C_TableVariable[0].nackflg;
00296     }
00297     else
00298     {
00299         return 0;
00300     }
00301 }
00306 int I2C_TableState(I2C_TypeDef *I2Cx)
00307 {
00308     if (I2Cx == I2C0)
00309     {
00310         return I2C_TableVariable[0].state;
00311     }
00312     else
00313     {
00314         return 0;
00315     }
00316 }
00320 void I2C_TableIrq(I2C_TypeDef *I2Cx)
00321 {
00322     if (I2Cx->SCR & BIT1)
00323     {
00324         I2C_TableVariable[0].nackflg = 1;
00325     }
00326     else
00327     {
00328         __NOP();
00329     }
00330     if (I2Cx->SCR & BIT5)
00331     {
00332         I2C_TableVariable[0].state = 0;
00333         I2Cx->SCR = 0;
00334     }
00335     else if (I2C_TableVariable[0].state == 1)
00336     {
00337         if (I2Cx->SCR & BIT3)
00338         {
00339             I2Cx->SCR = 0;
00340         }
00341         else
00342         {

```

```

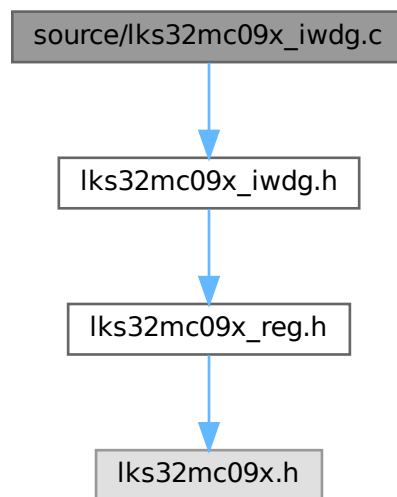
00343         I2Cx->DATA = 0xff;
00344         *I2C_TableVariable[0].rxdata = I2Cx->DATA;
00345         I2C_TableVariable[0].rxdatap--;
00346         if (I2C_TableVariable[0].rxdatap != 0)
00347         {
00348             I2C_TableVariable[0].rxdata++;
00349             I2Cx->SCR = BIT4;
00350         }
00351         else
00352         {
00353             I2Cx->SCR = 0;
00354         }
00355     }
00356 }
00357 else if (I2C_TableVariable[0].state == 2)
00358 {
00359     if (I2C_TableVariable[0].txdatap != 0)
00360     {
00361         I2Cx->DATA = *I2C_TableVariable[0].txdata;
00362         I2C_TableVariable[0].txdata++;
00363         I2C_TableVariable[0].txdatap--;
00364         I2Cx->SCR = BIT2;
00365     }
00366     else
00367     {
00368         I2Cx->SCR = 0;
00369     }
00370 }
00371 else
00372 {
00373     I2Cx->SCR = 0;
00374 }
00375 }

```

3.81 source/lks32mc09x_iwdg.c 文件参考

```
#include "lks32mc09x_iwdg.h"
```

lks32mc09x_iwdg.c 的引用(Include)关系图:



函数

- void IWDG_Init (IWDG_InitTypeDef *IWDG_InitStruct)

- IWDG*初始化
 - void [IWDG_Init](#) ([IWDG_InitTypeDef](#) *[IWDG_InitStruct](#))
*IWDG*配置结构体初始化
 - void [IWDG_DISABLE](#) (void)
关闭看门狗
 - void [IWDG_ENABLE](#) (void)
使能看门狗
 - void [IWDG_Feed](#) (void)
喂狗, 重置看门狗计数器
 - void [IWDG_Update](#) (uint32_t rth)
设置超时复位时间

3.81.1 函数说明

3.81.1.1 IWDG_DISABLE()

```
void IWDG_DISABLE (  
    void )
```

关闭看门狗

在文件 [lks32mc09x_iwdg.c](#) 第 42 行定义.

3.81.1.2 IWDG_ENABLE()

```
void IWDG_ENABLE (  
    void )
```

使能看门狗

在文件 [lks32mc09x_iwdg.c](#) 第 50 行定义.

3.81.1.3 IWDG_Feed()

```
void IWDG_Feed (  
    void )
```

喂狗, 重置看门狗计数器

在文件 [lks32mc09x_iwdg.c](#) 第 58 行定义.

3.81.1.4 IWDG_Init()

```
void IWDG_Init (  
    IWDG\_InitTypeDef * IWDG\_InitStruct )
```

*IWDG*初始化

参数

<i>IWDG_InitStruct</i>	指向IWDG初始化结构体的指针

在文件 [lks32mc09x_iwdg.c](#) 第 7 行定义.

3.81.1.5 IWDG_StrutInit()

```
void IWDG_StrutInit (
    IWDG_InitTypeDef * IWDG_InitStruct )
```

IWDG配置结构体初始化

参数

<i>IWDG_InitStruct</i>	指向IWDG初始化结构体的指针

在文件 [lks32mc09x_iwdg.c](#) 第 30 行定义.

3.81.1.6 IWDG_Update()

```
void IWDG_Update (
    uint32_t rth )
```

设置超时复位时间

参数

<i>rth</i>	超时复位时间，单位1/64k 秒，建议用 SECOND2IWDGCNT 宏将秒转换

在文件 [lks32mc09x_iwdg.c](#) 第 67 行定义.

3.82 lks32mc09x_iwdg.c

[浏览该文件的文档.](#)

```
00001 #include "lks32mc09x_iwdg.h"
00002
00007 void IWDG_Init(IWDG_InitTypeDef *IWDG_InitStruct)
00008 {
00009     IWDG_InitStruct->RTH = (IWDG_InitStruct->RTH > BIT21) ? BIT21 - 1 : IWDG_InitStruct->RTH;
00010     IWDG_InitStruct->WTH = (IWDG_InitStruct->WTH > BIT21) ? BIT21 - 1 : IWDG_InitStruct->WTH;
00011     IWDG->CFG = (IWDG_InitStruct->DWK_EN << 4) | (IWDG_InitStruct->WDG_EN);
00012     IWDG->PSW = PSW_IWDG_PRE;
00013     IWDG->RTH = IWDG_InitStruct->RTH;
00014     if (IWDG_InitStruct->RTH > IWDG_InitStruct->WTH)
00015     {
00016         IWDG->WTH = IWDG_InitStruct->RTH - IWDG_InitStruct->WTH;
00017     }
00018     else
00019     {
00020         IWDG->WTH = 0x001000;
00021     }
00022     IWDG->PSW = PSW_IWDG_PRE;
```



```

00023     IWDG->CLR = PSW_IWDG_CLR;
00024 }
00025
00030 void IWDG_InitInit(IWDG_InitTypeDef *IWDG_InitStruct)
00031 {
00032     IWDG_InitStruct->DWK_EN = DISABLE;           // 深度休眠定时唤醒使能
00033     IWDG_InitStruct->WDG_EN = DISABLE;           // 独立看门狗使能
00034     IWDG_InitStruct->WTH = IWDG_SECOND2IWDGCNT(1); // 看门狗定时唤醒时间, 大于复位时间后无效
00035     IWDG_InitStruct->RTH = IWDG_SECOND2IWDGCNT(2); // 看门狗超时复位时间 (21位计数器, 但低12恒位0)
00036                                           // SECOND2IWDGCNT输入参数, 秒
00037 }
00038
00042 void IWDG_DISABLE(void)
00043 {
00044     IWDG->CFG = 0x3c00;
00045 }
00046
00050 void IWDG_ENABLE(void)
00051 {
00052     IWDG->CFG = 0x1;
00053 }
00054
00058 void IWDG_Feed(void)
00059 {
00060     IWDG->PSW = PSW_IWDG_PRE;
00061     IWDG->CLR = PSW_IWDG_CLR;
00062 }
00067 void IWDG_Update(uint32_t rth)
00068 {
00069     IWDG->PSW = PSW_IWDG_PRE;
00070     IWDG->RTH = rth;
00071     IWDG->CLR = PSW_IWDG_CLR;
00072 }

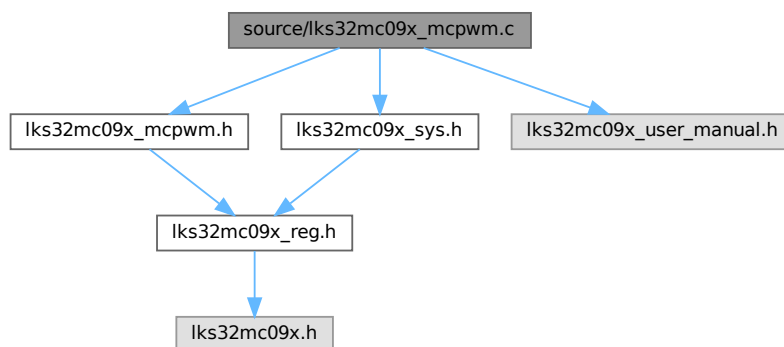
```

3.83 source/lks32mc09x_mcpwm.c 文件参考

```

#include "lks32mc09x_mcpwm.h"
#include "lks32mc09x_sys.h"
#include "lks32mc09x_user_manual.h"
lks32mc09x_mcpwm.c 的引用(Include)关系图:

```



函数

- static void **MCPWM_TmpUpdateFaillo** (MCPWM_TypeDef *MCPWMx)
更新faillo状态(仅库函数内部使用)
- void **MCPWM_Enable** (MCPWM_TypeDef *MCPWMx)
模块使能

- void **MCPWM_Disable** (MCPWM_TypeDef *MCPWMx)
模块失能
- void **MCPWM_Reset** (MCPWM_TypeDef *MCPWMx)
模块复位
- void **MCPWM_Init** (MCPWM_TypeDef *MCPWMx, MCPWM_InitTypeDef *MCPWM_InitStruct)
初始化MCPWM模块
- void **MCPWM_StructInit** (MCPWM_InitTypeDef *MCPWM_InitStruct)
将MCPWM_InitTypeDef结构体初始化为默认值
- void **MCPWM_SetOutputState** (MCPWM_TypeDef *MCPWMx, uint8_t state)
设置MCPWM输出状态
- void **MCPWM_SetOutputVal** (MCPWM_TypeDef *MCPWMx, uint8_t chn, int16_t p, int16_t n)
设置MCPWM输出值
- void **MCPWM_SetTMR** (MCPWM_TypeDef *MCPWMx, int16_t tmr0, int16_t tmr1, int16_t tmr2, int16_t tmr3)
设置MCPWM触发ADC事件时间
- void **MCPWM_SetIOConfig** (MCPWM_TypeDef *MCPWMx, uint16_t io0, uint16_t io1, uint16_t io2, uint16_t io3)
设置IO配置
- void **MCPWM_StartCount** (MCPWM_TypeDef *MCPWMx)
开始计数
- void **MCPWM_StopCount** (MCPWM_TypeDef *MCPWMx)
停止计数
- uint16_t **MCPWM_GetIRQFlag** (MCPWM_TypeDef *MCPWMx, uint32_t MCPWM_IF_x)
获取MCPWM中断标志
- uint16_t **MCPWM_GetIfFlag** (MCPWM_TypeDef *MCPWMx, uint32_t MCPWM_IF_x)
获取MCPWM事件
- uint16_t **MCPWM_GetEifFlag** (MCPWM_TypeDef *MCPWMx, uint32_t MCPWM_IF_x)
获取MCPWM错误中断标志
- uint16_t **MCPWM_GetErrorIRQFlag** (MCPWM_TypeDef *MCPWMx, uint32_t MCPWM{EIF_x)
获取MCPWM错误中断标志
- void **MCPWM_ClearIRQFlag** (MCPWM_TypeDef *MCPWMx, uint32_t MCPWM_IF_x)
清除MCPWM中断标志
- void **MCPWM_ClearErrorIRQFlag** (MCPWM_TypeDef *MCPWMx, uint32_t MCPWM{EIF_x)
清除MCPWM错误中断标志
- int16_t **MCPWM_GetCountTh** (MCPWM_TypeDef *MCPWMx)
获取当前计数器门限值
- int16_t **MCPWM_GetCountVal** (MCPWM_TypeDef *MCPWMx)
获取当前计数器值

变量

- static uint32_t **mcpwm_default_io**

3.83.1 函数说明

3.83.1.1 MCPWM_ClearErrorIRQFlag()

```
void MCPWM_ClearErrorIRQFlag (
    MCPWM_TypeDef * MCPWMx,
    uint32_t MCPWM{EIF_x} )
```

清除MCPWM错误中断标志

参数

<i>MCPWM_x</i>	MCPWM模块
<i>MCPWM_↔ EIF_x</i>	错误中断标志位

在文件 [lks32mc09x_mcpwm.c](#) 第 284 行定义.

3.83.1.2 MCPWM_ClearIRQFlag()

```
void MCPWM_ClearIRQFlag (
    MCPWM_TypeDef * MCPWMx,
    uint32_t MCPWM_IF_x )
```

清除MCPWM中断标志

参数

<i>MCPWM_x</i>	MCPWM模块
<i>MCPWM_↔ IF_x</i>	中断标志位

在文件 [lks32mc09x_mcpwm.c](#) 第 275 行定义.

3.83.1.3 MCPWM_Disable()

```
void MCPWM_Disable (
    MCPWM_TypeDef * MCPWMx )
```

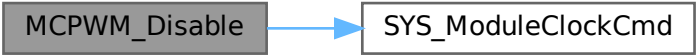
模块失能

参数

<i>MCPWM_x</i>	MCPWM模块

在文件 [lks32mc09x_mcpwm.c](#) 第 22 行定义.

函数调用图:



这是这个函数的调用关系图:



3.83.1.4 MCPWM.Enable()

```
void MCPWM_Enable (
    MCPWM_TypeDef * MCPWMx )
```

模块使能

参数

MCPWMx	MCPWM模块

在文件 [lks32mc09x-mcpwm.c](#) 第 11 行定义.

函数调用图:



这是这个函数的调用关系图:



3.83.1.5 MCPWM_GetCountTh()

```
int16_t MCPWM_GetCountTh (
    MCPWM_TypeDef * MCPWMx )
```

获取当前计数器门限值

参数

MCPWMx	MCPWM模块

返回

当前计数器门限值

在文件 [lks32mc09x_mcpwm.c](#) 第 294 行定义.

3.83.1.6 MCPWM_GetCountVal()

```
int16_t MCPWM_GetCountVal (
    MCPWM_TypeDef * MCPWMx )
```

获取当前计数器值

参数

MCPWMx	MCPWM模块

返回

当前计数器值

在文件 [lks32mc09x_mcpwm.c](#) 第 303 行定义.

3.83.1.7 MCPWM_GetEifFlag()

```
uint16_t MCPWM_GetEifFlag (
    MCPWM_TypeDef * MCPWMx,
    uint32_t MCPWM_IF_x )
```

获取MCPWM错误中断标志

参数

MCPWMx	MCPWM模块
MCPWM_↔ EIF_x	错误中断标志位

返回

1事件发生 0事件未发生

在文件 [lks32mc09x_mcpwm.c](#) 第 256 行定义.

3.83.1.8 MCPWM_GetErrorIRQFlag()

```
uint16_t MCPWM_GetErrorIRQFlag (
    MCPWM_TypeDef * MCPWMx,
    uint32_t MCPWM{EIF_x} )
```

获取CMPWM错误中断标志

参数

<i>MCPWMx</i>	MCPWM模块
<i>MCPWM_{←}EIF_x</i>	错误中断标志位

返回

1中断发生 0中断未发生

在文件 [lks32mc09x_mcpwm.c](#) 第 266 行定义.

3.83.1.9 MCPWM_GetIfFlag()

```
uint16_t MCPWM_GetIfFlag (
    MCPWM_TypeDef * MCPWMx,
    uint32_t MCPWM_{IF}_x )
```

获取MCPWM事件

参数

<i>MCPWMx</i>	MCPWM模块
<i>MCPWM_{←}IF_x</i>	中断标志位

返回

1事件发生 0事件未发生

注解

事件和中断的区别在于事件只是一个标志位，用于在中断函数外部判断事件是否发生

在文件 [lks32mc09x_mcpwm.c](#) 第 246 行定义.

3.83.1.10 MCPWM_GetIRQFlag()

```
uint16_t MCPWM_GetIRQFlag (
    MCPWM_TypeDef * MCPWMx,
    uint32_t MCPWM_IF_x )
```

获取MCPWM中断标志

参数

MCPWMx	MCPWM模块
MCPWM_↔IF_x	中断标志位

返回

1中断发生 0中断未发生

在文件 lks32mc09x_mcpwm.c 第 235 行定义.

3.83.1.11 MCPWM_Init()

```
void MCPWM_Init (
    MCPWM_TypeDef * MCPWMx,
    MCPWM_InitTypeDef * MCPWM_InitStruct )
```

初始化MCPWM模块

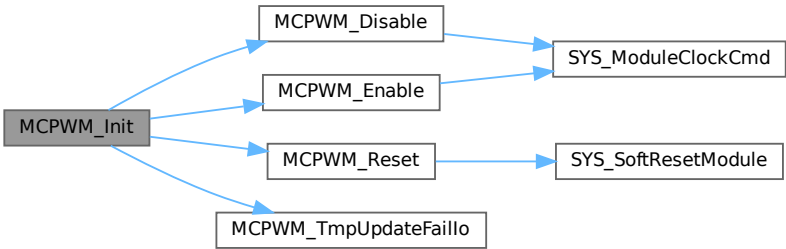
模块初始化

参数

MCPWMx	MCPWM模块指针
MCPWM_InitStruct	指向包含初始化参数的MCPWM_InitTypeDef结构体

在文件 lks32mc09x_mcpwm.c 第 45 行定义.

函数调用图:



3.83.1.12 MCPWM.Reset()

```
void MCPWM_Reset (
    MCPWM_TypeDef * MCPWMx )
```

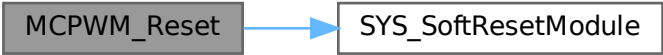
模块复位

参数

MCPWMx	MCPWM模块

在文件 [lks32mc09x-mcpwm.c](#) 第 33 行定义.

函数调用图:



这是这个函数的调用关系图:



3.83.1.13 MCPWM.SetIOConfig()

```
void MCPWM_SetIOConfig (
    MCPWM_TypeDef * MCPWMx,
    uint16_t io0,
    uint16_t io1,
    uint16_t io2,
    uint16_t io3 )
```

设置IO配置

参数

MCPWMx	MCPWM模块
io0	IO0配置
io1	IO1配置
io2	IO2配置
io3	IO3配置

注解

由于mcpwm的io0和io1是一个寄存器，io2和io3是一个寄存器，所以需要同时配置

在文件 [lks32mc09x_mcpwm.c](#) 第 200 行定义.

3.83.1.14 MCPWM_SetOutputState()

```
void MCPWM_SetOutputState (
    MCPWM_TypeDef * MCPWMx,
    uint8_t state )
```

设置MCPWM输出状态

参数

<i>MCPWMx</i>	MCPWM模块
<i>state</i>	输出状态 1打开输出 0关闭输出

在文件 [lks32mc09x_mcpwm.c](#) 第 131 行定义.

3.83.1.15 MCPWM_SetOutputVal()

```
void MCPWM_SetOutputVal (
    MCPWM_TypeDef * MCPWMx,
    uint8_t chn,
    int16_t p,
    int16_t n )
```

设置MCPWM输出值

参数

<i>MCPWMx</i>	MCPWM模块
<i>chn</i>	通道号 0-3
<i>p</i>	上管打开时间
<i>n</i>	下管打开时间

在文件 [lks32mc09x_mcpwm.c](#) 第 152 行定义.

3.83.1.16 MCPWM_SetTMR()

```
void MCPWM_SetTMR (
    MCPWM_TypeDef * MCPWMx,
    int16_t tmr0,
    int16_t tmr1,
    int16_t tmr2,
    int16_t tmr3 )
```

设置MCPWM触发ADC事件时间

参数

<i>MCPWMx</i>	MCPWM模块
<i>tmr0</i>	事件0时间
<i>tmr1</i>	事件1时间
<i>tmr2</i>	事件2时间
<i>tmr3</i>	事件3时间

在文件 [lks32mc09x_mcpwm.c](#) 第 184 行定义.

3.83.1.17 MCPWM_StartCount()

```
void MCPWM_StartCount (
    MCPWM_TypeDef * MCPWMx )
```

开始计数

参数

<i>MCPWMx</i>	MCPWM模块
<i>count</i>	计数器初始值

在文件 [lks32mc09x_mcpwm.c](#) 第 212 行定义.

3.83.1.18 MCPWM_StopCount()

```
void MCPWM_StopCount (
    MCPWM_TypeDef * MCPWMx )
```

停止计数

参数

<i>MCPWMx</i>	MCPWM模块

在文件 [lks32mc09x_mcpwm.c](#) 第 223 行定义.

3.83.1.19 MCPWM_StructInit()

```
void MCPWM_StructInit (
    MCPWM_InitTypeDef * MCPWM_InitStruct )
```

将MCPWM_InitTypeDef结构体初始化为默认值

mcpwm配置结构体初始化

参数

<code>MCPWM_InitStruct</code>	指向要初始化的MCPWM_InitTypeDef结构体

在文件 [lks32mc09x_mcpwm.c](#) 第 118 行定义.

3.83.1.20 MCPWM_TmpUpdateFailIo()

```
static void MCPWM_TmpUpdateFailIo (  
    MCPWM_TypeDef * MCPWMx ) [static]
```

更新failIO状态(仅库函数内部使用)

在文件 [lks32mc09x_mcpwm.c](#) 第 310 行定义.

这是这个函数的调用关系图:



3.83.2 变量说明

3.83.2.1 mcpwm_default_io

```
uint32_t mcpwm_default_io [static]
```

在文件 [lks32mc09x_mcpwm.c](#) 第 4 行定义.

3.84 lks32mc09x_mcpwm.c

[浏览该文件的文档.](#)

```
00001 #include "lks32mc09x_mcpwm.h"  
00002 #include "lks32mc09x_sys.h"  
00003 #include "lks32mc09x_user_manual.h"  
00004 static uint32_t mcpwm_default_io;  
00005 // 仅内部使用  
00006 static void MCPWM_TmpUpdateFailIo(MCPWM_TypeDef *MCPWMx);  
00011 void MCPWM_Enable(MCPWM_TypeDef *MCPWMx)  
00012 {  
00013     SYS_ModuleClockCmd(SYS_MODULE_MCPWM0, ENABLE);  
00014     MCPWMx->PRT = 0xdead;  
00015     MCPWMx->TCLK |= BIT2;  
00016     MCPWMx->PRT = 0;  
00017 }  
00022 void MCPWM_Disable(MCPWM_TypeDef *MCPWMx)  
00023 {  
00024     MCPWMx->PRT = 0xdead;  
00025     MCPWMx->TCLK &= ~BIT2;  
00026     MCPWMx->PRT = 0;
```

```

00027     SYS_ModuleClockCmd(SYS_MODULE_MCPWM0, DISABLE);
00028 }
00033 void MCPWM_Reset (MCPWM_TypeDef *MCPWMx)
00034 {
00035     if (MCPWMx == MCPWM0)
00036     {
00037         SYS_SoftResetModule(SYS_MODULE_MCPWM0);
00038     }
00039 }
00045 void MCPWM_Init (MCPWM_TypeDef *MCPWMx, MCPWM_InitTypeDef *MCPWM_InitStruct)
00046 {
00047     // 复位MCPWM模块
00048     MCPWM_Reset (MCPWMx);
00049     if (MCPWM_InitStruct->EN)
00050     {
00051         // 使能MCPWM模块
00052         MCPWM_Enable (MCPWMx);
00053         // 初始化MCPWM模块
00054         MCPWMx->PRT = 0xdead; // mcpwm模块解锁
00055         if (MCPWM_InitStruct->COUNT & MCPWM_COUNT_START)
00056         {
00057             MCPWMx->TCLK = BIT6 | (MCPWM_InitStruct->CLK_DIV << 12) | BIT2;
00058         }
00059         else
00060         {
00061             if (MCPWM_InitStruct->COUNT != 0)
00062             {
00063                 MCPWMx->TCLK = ((MCPWMx->TCLK & (~BIT6)) | BIT8) | (MCPWM_InitStruct->CLK_DIV << 12) |
00064                 BIT2; // 关闭计数器，并打开外部触发
00065                 MCPWMx->EVT0 = MCPWM_InitStruct->COUNT;
00066                 // 设置外部触发信号
00067             }
00068             else
00069             {
00070                 MCPWMx->TCLK = (MCPWM_InitStruct->CLK_DIV << 12) | BIT2;
00071             }
00072             MCPWMx->TH00 = 0;
00073             MCPWMx->TH01 = 0;
00074             MCPWMx->TH10 = 0;
00075             MCPWMx->TH11 = 0;
00076             MCPWMx->TH20 = 0;
00077             MCPWMx->TH21 = 0;
00078             MCPWMx->TH30 = 0;
00079             MCPWMx->TH31 = 0;
00080             MCPWMx->CNT0 = -MCPWM_InitStruct->TH;
00081             MCPWMx->TH0 = MCPWM_InitStruct->TH;
00082             MCPWMx->TMR0 = MCPWM_InitStruct->TMR0;
00083             MCPWMx->TMR1 = MCPWM_InitStruct->TMR1;
00084             MCPWMx->TMR2 = MCPWM_InitStruct->TMR2;
00085             MCPWMx->TMR3 = MCPWM_InitStruct->TMR3;
00086             MCPWMx->DTH00 = MCPWM_InitStruct->DTHP;
00087             MCPWMx->DTH01 = MCPWM_InitStruct->DTHN;
00088             MCPWMx->FLT = MCPWM_InitStruct->FLT_DIV;
00089             MCPWMx->IO01 = MCPWM_InitStruct->IO0 | (MCPWM_InitStruct->IO1 << 8);
00090             MCPWMx->IO23 = MCPWM_InitStruct->IO2 | (MCPWM_InitStruct->IO3 << 8);
00091             MCPWMx->CH_FAIL = MCPWM_InitStruct->FAIL0 | MCPWM_InitStruct->FAIL1;
00092             MCPWMx->AUEN = MCPWM_InitStruct->AUEN;
00093             MCPWMx->IE0 = MCPWM_InitStruct->IE;
00094             MCPWMx->EIE = MCPWM_InitStruct->EIE;
00095             MCPWMx->RE = MCPWM_InitStruct->RE;
00096             MCPWMx->SDCFG = MCPWM_InitStruct->TR;
00097             // 更新所有存在影子寄存器的寄存器
00098             MCPWMx->UPDATE = 0xffffffff;
00099             {
00100                 // IO01 IO23更新完成后再设置默认电平，会根据FAIL_IO的设置，自动处理极性和通道交换
00101                 mcpwm_default_io = MCPWM_InitStruct->FAIL_IO;
00102                 MCPWM_TmpUpdateFailIo (MCPWMx);
00103             }
00104             MCPWMx->IF0 = 0xffff;
00105             MCPWMx->EIF = 0xffff;
00106             MCPWMx->PRT = 0x0000; // mcpwm模块上锁
00107         }
00108     }
00109     // 失能MCPWM模块
00110     MCPWM_Disable (MCPWMx);
00111 }
00112 }
00113
00118 void MCPWM_StructInit (MCPWM_InitTypeDef *MCPWM_InitStruct)
00119 {
00120     uint32_t i;
00121     for (i = 0; i < sizeof(MCPWM_InitTypeDef); i++)
00122     {
00123         ((uint8_t *)MCPWM_InitStruct)[i] = 0;
00124     }

```

```

00125 }
00131 void MCPWM_SetOutputState(MCPWM_TypeDef *MCPWMx, uint8_t state)
00132 {
00133     MCPWMx->PRT = 0xdead; // mcpwm模块解锁
00134     if (state)
00135     {
00136         MCPWMx->EIF = MCPWMx->EIF;
00137         MCPWMx->CH_FAIL |= BIT6;
00138     }
00139     else
00140     {
00141         MCPWMx->CH_FAIL &= ~BIT6;
00142     }
00143     MCPWMx->PRT = 0x0000; // mcpwm模块上锁
00144 }
00152 void MCPWM_SetOutputVal(MCPWM_TypeDef *MCPWMx, uint8_t chn, int16_t p, int16_t n)
00153 {
00154     switch (chn)
00155     {
00156         case 0:
00157             MCPWMx->TH00 = p;
00158             MCPWMx->TH01 = n;
00159             break;
00160         case 1:
00161             MCPWMx->TH10 = p;
00162             MCPWMx->TH11 = n;
00163             break;
00164         case 2:
00165             MCPWMx->TH20 = p;
00166             MCPWMx->TH21 = n;
00167             break;
00168         case 3:
00169             MCPWMx->TH30 = p;
00170             MCPWMx->TH31 = n;
00171             break;
00172         default:
00173             break;
00174     }
00175 }
00184 void MCPWM_SetTMR(MCPWM_TypeDef *MCPWMx, int16_t tmr0, int16_t tmr1, int16_t tmr2, int16_t tmr3)
00185 {
00186     MCPWMx->TMR0 = tmr0;
00187     MCPWMx->TMR1 = tmr1;
00188     MCPWMx->TMR2 = tmr2;
00189     MCPWMx->TMR3 = tmr3;
00190 }
00200 void MCPWM_SetIOConfig(MCPWM_TypeDef *MCPWMx, uint16_t io0, uint16_t io1, uint16_t io2, uint16_t io3)
00201 {
00202     MCPWMx->PRT = 0xdead; // mcpwm模块解锁
00203     MCPWMx->IO01 = io0 | (io1 << 8);
00204     MCPWMx->IO23 = io2 | (io3 << 8);
00205     MCPWMx->PRT = 0x0000; // mcpwm模块上锁
00206 }
00212 void MCPWM_StartCount(MCPWM_TypeDef *MCPWMx)
00213 {
00214     MCPWMx->PRT = 0xdead; // mcpwm模块解锁
00215     // MCPWMx->UPDATE = BIT11;
00216     MCPWMx->TCLK |= BIT6;
00217     MCPWMx->PRT = 0x0000; // mcpwm模块上锁
00218 }
00223 void MCPWM_StopCount(MCPWM_TypeDef *MCPWMx)
00224 {
00225     MCPWMx->PRT = 0xdead; // mcpwm模块解锁
00226     MCPWMx->TCLK &= ~BIT6;
00227     MCPWMx->PRT = 0x0000; // mcpwm模块上锁
00228 }
00235 uint16_t MCPWM_GetIRQFlag(MCPWM_TypeDef *MCPWMx, uint32_t MCPWM_IF_x)
00236 {
00237     return (MCPWMx->IF0 & MCPWMx->IE0 & MCPWM_IF_x) ? 1 : 0;
00238 }
00246 uint16_t MCPWM_GetIfFlag(MCPWM_TypeDef *MCPWMx, uint32_t MCPWM_IF_x)
00247 {
00248     return (MCPWMx->IF0 & MCPWM_IF_x) ? 1 : 0;
00249 }
00256 uint16_t MCPWM_GetEifFlag(MCPWM_TypeDef *MCPWMx, uint32_t MCPWM_IF_x)
00257 {
00258     return (MCPWMx->EIF & MCPWMx->EIE & MCPWM_IF_x) ? 1 : 0;
00259 }
00266 uint16_t MCPWM_GetErrorIRQFlag(MCPWM_TypeDef *MCPWMx, uint32_t MCPWM{EIF_x)
00267 {
00268     return (MCPWMx->EIF & MCPWM{EIF_x) ? 1 : 0;
00269 }
00275 void MCPWM_ClearIRQFlag(MCPWM_TypeDef *MCPWMx, uint32_t MCPWM_IF_x)
00276 {
00277     MCPWMx->IF0 = MCPWM_IF_x;
00278 }
00284 void MCPWM_ClearErrorIRQFlag(MCPWM_TypeDef *MCPWMx, uint32_t MCPWM{EIF_x)

```

```

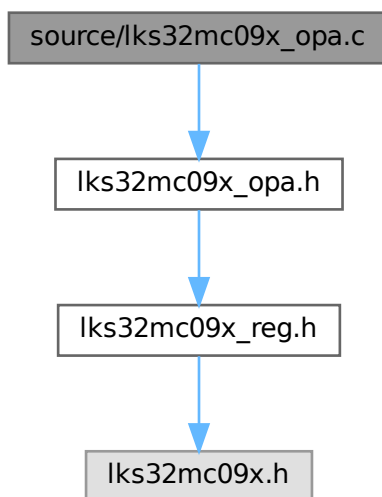
00285 {
00286     MCPWMx->EIF = MCPWM_EIF_x;
00287 }
00288
00294 int16_t MCPWM_GetCountTh(MCPWM_TypeDef *MCPWMx)
00295 {
00296     return MCPWMx->TH0;
00297 }
00303 int16_t MCPWM_GetCountVal(MCPWM_TypeDef *MCPWMx)
00304 {
00305     return MCPWMx->CNT0;
00306 }
00310 static void MCPWM_ImpUpdateFailIo(MCPWM_TypeDef *MCPWMx)
00311 {
00312     uint32_t chdef = mcpwm_default_io;
00313     if (MCPWMx->IO01 & MCPWM0_IO01_CH0_NP_BIT)
00314     {
00315         chdef ^= BIT0;
00316     }
00317     if (MCPWMx->IO01 & MCPWM0_IO01_CH0_PP_BIT)
00318     {
00319         chdef ^= BIT1;
00320     }
00321     if (MCPWMx->IO01 & MCPWM0_IO01_CH1_NP_BIT)
00322     {
00323         chdef ^= BIT2;
00324     }
00325     if (MCPWMx->IO01 & MCPWM0_IO01_CH1_PP_BIT)
00326     {
00327         chdef ^= BIT3;
00328     }
00329     if (MCPWMx->IO23 & MCPWM0_IO23_CH2_NP_BIT)
00330     {
00331         chdef ^= BIT4;
00332     }
00333     if (MCPWMx->IO23 & MCPWM0_IO23_CH2_PP_BIT)
00334     {
00335         chdef ^= BIT5;
00336     }
00337     if (MCPWMx->IO23 & MCPWM0_IO23_CH3_NP_BIT)
00338     {
00339         chdef ^= BIT6;
00340     }
00341     if (MCPWMx->IO23 & MCPWM0_IO23_CH3_PP_BIT)
00342     {
00343         chdef ^= BIT7;
00344     }
00345     if (MCPWMx->IO01 & MCPWM0_IO01_CH0_PN_SW_BIT)
00346     {
00347         // 交换chdef的bit0和bit1
00348         chdef = (chdef & (~ (BIT0 | BIT1))) | ((chdef & BIT0) << 1) | ((chdef & BIT1) >> 1);
00349     }
00350     if (MCPWMx->IO01 & MCPWM0_IO01_CH1_PN_SW_BIT)
00351     {
00352         // 交换chdef的bit2和bit3
00353         chdef = (chdef & (~ (BIT2 | BIT3))) | ((chdef & BIT2) << 1) | ((chdef & BIT3) >> 1);
00354     }
00355     if (MCPWMx->IO23 & MCPWM0_IO23_CH2_PN_SW_BIT)
00356     {
00357         // 交换chdef的bit4和bit5
00358         chdef = (chdef & (~ (BIT4 | BIT5))) | ((chdef & BIT4) << 1) | ((chdef & BIT5) >> 1);
00359     }
00360     if (MCPWMx->IO23 & MCPWM0_IO23_CH3_PN_SW_BIT)
00361     {
00362         // 交换chdef的bit6和bit7
00363         chdef = (chdef & (~ (BIT6 | BIT7))) | ((chdef & BIT6) << 1) | ((chdef & BIT7) >> 1);
00364     }
00365
00366     MCPWMx->PRT = 0xdead;
00367     MCPWMx->CH_DEF = chdef;
00368 }

```

3.85 source/lks32mc09x_opa.c 文件参考

```
#include "lks32mc09x_opa.h"
```

lks32mc09x_opa.c 的引用(Include)关系图:



函数

- void **OPA_Init** (uint32_t OPA_CHANNEL_x, **OPA_InitTypeDef** *OPA_InitStruct)
OPA 初始化
- void **OPA_StructInit** (**OPA_InitTypeDef** *OPA_InitStruct)
OPA初始化结构体初始化
- void **OPA_Enable** (uint32_t OPA_CHANNEL_x)
OPA 使能
- void **OPA_Disable** (uint32_t OPA_CHANNEL_x)
OPA 失能
- void **OPA_SetGain** (uint32_t OPA_CHANNEL_x, uint32_t OPA_GAIN_x)
OPA 增益设置
- void **OPA_SetItt** (uint32_t OPA_IT_x)
OPA 偏置电流设置
- void **OPA_OutCmpSignal** (uint32_t OPA_OUT_CMP_x, uint32_t Out2Gpio)
OPA 输出信号选择 (使能输出到GPIO)

3.85.1 函数说明

3.85.1.1 OPA_Disable()

```
void OPA_Disable (
    uint32_t OPA_CHANNEL_x )
```

OPA 失能

参数

$OPA_{\leftarrow}$ <i>CHANNEL_x</i>	OPA通道

在文件 [lks32mc09x_opa.c](#) 第 61 行定义.

这是这个函数的调用关系图:



3.85.1.2 OPA.Enable()

```
void OPA_Enable (
    uint32_t OPA_CHANNEL_x )
```

OPA 使能

参数

$OPA_{\leftarrow}$ <i>CHANNEL_x</i>	OPA通道

在文件 [lks32mc09x_opa.c](#) 第 37 行定义.

这是这个函数的调用关系图:



3.85.1.3 OPA.Init()

```
void OPA_Init (
    uint32_t OPA_CHANNEL_x,
    OPA_InitTypeDef * OPA_InitStruct )
```

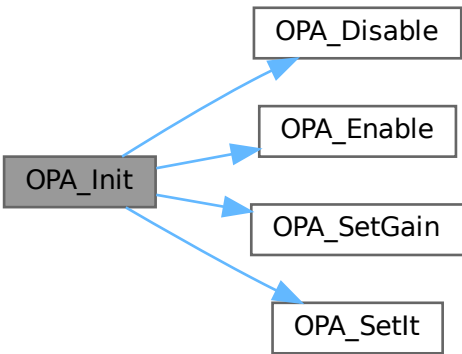
OPA 初始化

参数

<i>OPA_↔CHANNEL_x</i>	OPA通道
<i>OPA_InitStruct</i>	OPA初始化结构体指针

在文件 [lks32mc09x_opa.c](#) 第 8 行定义.

函数调用图:



3.85.1.4 OPA_OutCmpSignal()

```
void OPA_OutCmpSignal (
    uint32_t OPA_OUT_CMP_x,
    uint32_t Out2Gpio )
```

OPA 输出信号选择（使能输出到GPIO）

参数

<i>OPA_OUT_↔CMP_x</i>	OPA_OUT_CMP信号来源配置
<i>Out2Gpio</i>	OPA_OUT_CMP输出到IO使能，P2.7

在文件 [lks32mc09x_opa.c](#) 第 121 行定义.

3.85.1.5 OPA_SetGain()

```
void OPA_SetGain (
    uint32_t OPA_CHANNEL_x,
    uint32_t OPA_GAIN_x )
```

OPA 增益设置

参数

$OPA_{\leftrightarrow} CHANNEL\_x$	OPA通道
OPA_GAIN_x	OPA增益

在文件 [lks32mc09x_opa.c](#) 第 86 行定义.

这是这个函数的调用关系图:



3.85.1.6 OPA_SetIt()

```
void OPA_SetIt (
    uint32_t OPA_IT_x )
```

OPA 偏置电流设置

参数

$OPA_{\leftrightarrow} IT\_x$	OPA偏置电流

在文件 [lks32mc09x_opa.c](#) 第 110 行定义.

这是这个函数的调用关系图:



3.85.1.7 OPA_StructInit()

```
void OPA_StructInit (
    OPA_InitTypeDef * OPA_InitStruct )
```

OPA初始化结构体初始化

在文件 `lks32mc09x_opa.c` 第 25 行定义.

3.86 lks32mc09x_opa.c

[浏览该文件的文档.](#)

```

00001 #include "lks32mc09x_opa.h"
00002
00008 void OPA_Init(uint32_t OPA_CHANNEL_x, OPA_InitTypeDef *OPA_InitStruct)
00009 {
00010     if (OPA_InitStruct->OPA_CLEna)
00011     {
00012         OPA_Enable(OPA_CHANNEL_x);
00013     }
00014     else
00015     {
00016         OPA_Disable(OPA_CHANNEL_x);
00017     }
00018     OPA_Enable(OPA_CHANNEL_x);
00019     OPA_SetIt(OPA_InitStruct->OPA_IT);
00020     OPA_SetGain(OPA_CHANNEL_x, OPA_InitStruct->OPA_Gain);
00021 }
00025 void OPA_StructInit(OPA_InitTypeDef *OPA_InitStruct)
00026 {
00027     uint32_t i;
00028     for (i = 0; i < sizeof(OPA_InitTypeDef); i++)
00029     {
00030         ((uint8_t *)OPA_InitStruct)[i] = 0;
00031     }
00032 }
00037 void OPA_Enable(uint32_t OPA_CHANNEL_x)
00038 {
00039     SYS0->PROTECT = 0x7a83;
00040     switch (OPA_CHANNEL_x)
00041     {
00042         case OPA_CHANNEL_0:
00043             SYS0->AFE_REG1 |= BIT12;
00044             break;
00045         case OPA_CHANNEL_1:
00046             SYS0->AFE_REG1 |= BIT13;
00047             break;
00048         case OPA_CHANNEL_2:
00049             SYS0->AFE_REG1 |= BIT14;
00050             break;
00051         case OPA_CHANNEL_3:
00052             SYS0->AFE_REG1 |= BIT15;
00053             break;
00054     }
00055     SYS0->PROTECT = 0;
00056 }
00061 void OPA_Disable(uint32_t OPA_CHANNEL_x)
00062 {
00063     SYS0->PROTECT = 0x7a83;
00064     switch (OPA_CHANNEL_x)
00065     {
00066         case OPA_CHANNEL_0:
00067             SYS0->AFE_REG1 &= ~BIT12;
00068             break;
00069         case OPA_CHANNEL_1:
00070             SYS0->AFE_REG1 &= ~BIT13;
00071             break;
00072         case OPA_CHANNEL_2:
00073             SYS0->AFE_REG1 &= ~BIT14;
00074             break;
00075         case OPA_CHANNEL_3:
00076             SYS0->AFE_REG1 &= ~BIT15;
00077             break;
00078     }
00079     SYS0->PROTECT = 0;
00080 }
00086 void OPA_SetGain(uint32_t OPA_CHANNEL_x, uint32_t OPA_GAIN_x)
00087 {
00088     SYS0->PROTECT = 0x7a83;
00089     switch (OPA_CHANNEL_x)
00090     {
00091         case OPA_CHANNEL_0:
00092             SYS0->AFE_REG0 = (SYS0->AFE_REG0 & ~(3)) | OPA_GAIN_x;
00093             break;
00094         case OPA_CHANNEL_1:

```

```

00095         SYS0->AFE_REG0 = (SYS0->AFE_REG0 & ~(3 << 2)) | (OPA_GAIN_x << 2);
00096         break;
00097     case OPA_CHANNEL_2:
00098         SYS0->AFE_REG0 = (SYS0->AFE_REG0 & ~(3 << 4)) | (OPA_GAIN_x << 4);
00099         break;
00100     case OPA_CHANNEL_3:
00101         SYS0->AFE_REG0 = (SYS0->AFE_REG0 & ~(3 << 6)) | (OPA_GAIN_x << 6);
00102         break;
00103     }
00104     SYS0->PROTECT = 0;
00105 }
00110 void OPA_SetIt(uint32_t OPA_IT_x)
00111 {
00112     SYS0->PROTECT = 0x7a83;
00113     SYS0->AFE_REG0 = (SYS0->AFE_REG0 & ~(3 << 8)) | (OPA_IT_x << 8);
00114     SYS0->PROTECT = 0;
00115 }
00121 void OPA_OutCmpSignal(uint32_t OPA_OUT_CMP_x, uint32_t Out2Gpio)
00122 {
00123     SYS0->PROTECT = 0x7a83;
00124     SYS0->AFE_REG2 = (SYS0->AFE_REG2 & ~(0xf)) |
00125                     (OPA_OUT_CMP_x |
00126                     (Out2Gpio << 3));
00127     SYS0->PROTECT = 0;
00128 }

```

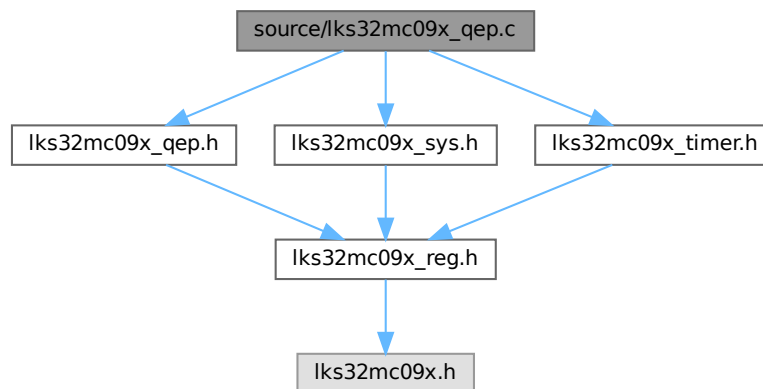
3.87 source/lks32mc09x_qep.c 文件参考

```

#include "lks32mc09x_qep.h"
#include "lks32mc09x_sys.h"
#include "lks32mc09x_timer.h"

```

lks32mc09x_qep.c 的引用(Include)关系图:



函数

- void **QEP_Init** (**QEP_TypeDef** *QEPx, **QEP_InitTypeDef** *TIM_QEPInitStruct)
QEP初始化函数 (初始化完成后会自动使能, 并使能timer2)
- void **QEP_StructInit** (**QEP_InitTypeDef** *TIM_QEPInitStruct)
QEP结构体初始化函数
- void **QEP_Cmd** (**QEP_TypeDef** *QEPx, uint8_t state)
QEP使能/失能函数
- uint16_t **QEP_GetCount** (**QEP_TypeDef** *QEPx)

- 获取QEP计数器当前值
- void QEP_ClearIRQFlag (QEP_TypeDef *QEPx, uint32_t tempFlag)
清除QEP中断标志
- uint16_t QEP_GetIRQFlag (QEP_TypeDef *QEPx, uint32_t QEP_IF_x)
获取QEP中断标志状态
- uint16_t QEP_GetFlag (QEP_TypeDef *QEPx, uint32_t QEP_IF_x)
获取QEP状态标志
- void QEP_SetMode (QEP_TypeDef *QEPx, uint32_t QEP_MODE_x)
获取QEP状态标志

3.87.1 函数说明

3.87.1.1 QEP_ClearIRQFlag()

```
void QEP_ClearIRQFlag (  
    QEP_TypeDef * QEPx,  
    uint32_t tempFlag )
```

清除QEP中断标志

参数

QEPx	QEP模块指针
tempFlag	要清除的中断标志

在文件 lks32mc09x_qep.c 第 82 行定义.

3.87.1.2 QEP_Cmd()

```
void QEP_Cmd (  
    QEP_TypeDef * QEPx,  
    uint8_t state )
```

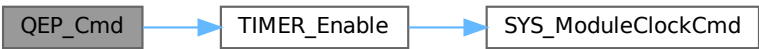
QEP使能/失能函数

参数

QEPx	QEP模块指针
state	使能/失能状态

在文件 lks32mc09x_qep.c 第 54 行定义.

函数调用图:



3.87.1.3 QEP_GetCount()

```
uint16_t QEP_GetCount (
    QEP_TypeDef * QEPx )
```

获取QEP计数器当前值

参数

QEPx	QEP模块指针

返回
计数器当前值

在文件 [lks32mc09x_qep.c](#) 第 72 行定义.

3.87.1.4 QEP_GetFlag()

```
uint16_t QEP_GetFlag (
    QEP_TypeDef * QEPx,
    uint32_t QEP_IFx )
```

获取QEP状态标志

参数

QEPx	QEP模块指针
QEP_↔IF_x	状态标志类型

返回
状态标志

在文件 [lks32mc09x_qep.c](#) 第 104 行定义.

3.87.1.5 QEP_GetIRQFlag()

```
uint16_t QEP_GetIRQFlag (
    QEP_TypeDef * QEPx,
    uint32_t QEP_IF_x )
```

获取QEP中断标志状态

参数

QEPx	QEP模块指针
QEP_↔IF_x	中断标志类型

返回

中断标志状态

在文件 lks32mc09x-qep.c 第 93 行定义.

3.87.1.6 QEP_Init()

```
void QEP_Init (
    QEP_TypeDef * QEPx,
    QEP_InitTypeDef * TIM_QEPInitStruct )
```

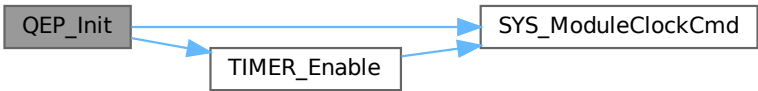
QEP初始化函数（初始化完成后会自动使能，并使能timer2）

参数

QEPx	QEP模块指针
TIM_QEPInitStruct	初始化结构体指针

在文件 lks32mc09x-qep.c 第 10 行定义.

函数调用图:



3.87.1.7 QEP_SetMode()

```
void QEP_SetMode (
```



```
QEP_TypeDef * QEPx,
uint32_t QEP_MODE_x )
```

获取QEP状态标志

参数

<i>QEPx</i>	QEP模块指针
<i>QEP_↔ MODE_x</i>	编码器工作模式

在文件 [lks32mc09x_qep.c](#) 第 114 行定义.

3.87.1.8 QEP_StructInit()

```
void QEP_StructInit (
    QEP_InitTypeDef * TIM_QEPInitStruct )
```

QEP结构体初始化函数

参数

<i>TIM_QEPInitStruct</i>	初始化结构体指针

在文件 [lks32mc09x_qep.c](#) 第 40 行定义.

3.88 lks32mc09x_qep.c

[浏览该文件的文档.](#)

```
00001 #include "lks32mc09x_qep.h"
00002 #include "lks32mc09x_sys.h"
00003 #include "lks32mc09x_timer.h"
00004
00010 void QEP_Init(QEP_TypeDef *QEPx, QEP_InitTypeDef *TIM_QEPInitStruct)
00011 {
00012     // 使能QEP模块时钟
00013     if (QEPx == QEP0)
00014     {
00015         SYS_ModuleClockCmd(SYS_MODULE_QEP0, ENABLE);
00016         TIMER_Enable(TIMER2);
00017     }
00018     else
00019     {
00020         __NOP();
00021     }
00022     // QEP配置寄存器
00023     QEPx->CFG = BIT15 |
00024         (TIM_QEPInitStruct->Filter << 16) |
00025         (TIM_QEPInitStruct->FE_CNT_EN << 10) |
00026         (TIM_QEPInitStruct->Mode << 8) |
00027         (TIM_QEPInitStruct->ZC);
00028
00029     // 设置计数门限
00030     QEPx->TH = TIM_QEPInitStruct->TH;
00031
00032     // 设置中断使能
00033     QEPx->IE = TIM_QEPInitStruct->IRQEna;
00034 }
00035
00040 void QEP_StructInit(QEP_InitTypeDef *TIM_QEPInitStruct)
00041 {
```

```

00042     uint32_t i;
00043     for (i = 0; i < sizeof(QEP_InitTypeDef); i++)
00044     {
00045         ((uint8_t *)TIM_QEPInitStruct)[i] = 0;
00046     }
00047 }
00048
00054 void QEP_Cmd(QEP_TypeDef *QEPx, uint8_t state)
00055 {
00056     if (state == ENABLE)
00057     {
00058         QEPx->CFG |= (1 << 15); // 使能QEP模块
00059         TIMER_Enable(TIMER2); // 使用QEP时会自动使能timer2
00060     }
00061     else
00062     {
00063         QEPx->CFG &= ~(1 << 15); // 失能QEP模块
00064     }
00065 }
00066
00072 uint16_t QEP_GetCount(QEP_TypeDef *QEPx)
00073 {
00074     return QEPx->CNT;
00075 }
00076
00082 void QEP_ClearIRQFlag(QEP_TypeDef *QEPx, uint32_t tempFlag)
00083 {
00084     QEPx->IF = tempFlag;
00085 }
00086
00093 uint16_t QEP_GetIRQFlag(QEP_TypeDef *QEPx, uint32_t QEP_IF_x)
00094 {
00095     return (QEPx->IF & QEP_IF_x) ? 1 : 0;
00096 }
00097
00104 uint16_t QEP_GetFlag(QEP_TypeDef *QEPx, uint32_t QEP_IF_x)
00105 {
00106     return (QEPx->IF & QEP_IF_x) ? 1 : 0;
00107 }
00108
00114 void QEP_SetMode(QEP_TypeDef *QEPx, uint32_t QEP_MODE_x)
00115 {
00116     QEPx->CFG = (QEPx->CFG & ~(0x3 << 8)) | (QEP_MODE_x << 8);
00117 }

```

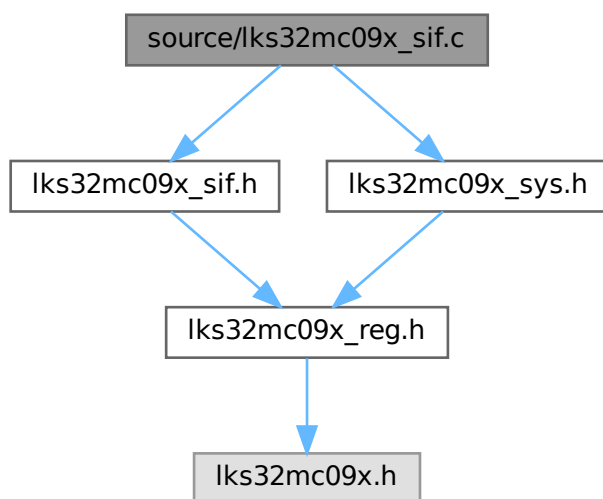
3.89 source/lks32mc09x_sif.c 文件参考

```

#include "lks32mc09x_sif.h"
#include "lks32mc09x_sys.h"

```

lks32mc09x_sif.c 的引用(Include)关系图:



函数

- void **SIF_StructInit** (SIF_InitTypeDef *SIFInitStruct)
初始化SIF结构体
- void **SIF_Init** (SIF_TypeDef *SIFx, SIF_InitTypeDef *SIFInitStruct)
初始化SIF模块
- void **SIF_Timebase_set** (SIF_TypeDef *SIFx, uint32_t tosc)
设置SIF时基
- void **SIF_TH_set** (SIF_TypeDef *SIFx, uint32_t tsh, uint32_t tdh)
设置SIF同步和结束信号时长
- void **SIF_Senddata** (SIF_TypeDef *SIFx, uint8_t Data)
发送数据
- void **SIF_DMA_IsEnable** (SIF_TypeDef *SIFx, uint8_t state)
设置SIF DMA使能状态
- void **SIF_Irq_IsEnable** (SIF_TypeDef *SIFx, uint8_t state)
设置SIF中断使能状态
- void **SIF_Clear_Irq** (SIF_TypeDef *SIFx)
清除SIF中断标志

3.89.1 函数说明

3.89.1.1 SIF_Clear_Irq()

```
void SIF_Clear_Irq (
    SIF_TypeDef * SIFx )
```

清除SIF中断标志

在文件 **lks32mc09x_sif.c** 第 109 行定义.

3.89.1.2 SIF_DMA.IsEnable()

```
void SIF_DMA_IsEnable (
    SIF_TypeDef * SIFx,
    uint8_t state )
```

设置SIF DMA使能状态

参数

state	使能状态，1表示使能，0表示禁用

在文件 lks32mc09x_sif.c 第 78 行定义.

3.89.1.3 SIF_Init()

```
void SIF_Init (
    SIF_TypeDef * SIFx,
    SIF_InitTypeDef * SIFInitStruct )
```

初始化SIF模块

参数

SIFx	指向SIF_TypeDef结构体的指针，表示具体的SIF实例
SIFInitStruct	指向SIF_InitTypeDef结构体的指针，包含初始化参数

在文件 lks32mc09x_sif.c 第 22 行定义.

函数调用图:



3.89.1.4 SIF_Irq.IsEnable()

```
void SIF_Irq_IsEnable (
    SIF_TypeDef * SIFx,
    uint8_t state )
```

设置SIF中断使能状态

参数

<i>state</i>	使能状态，1表示使能，0表示禁用

在文件 [lks32mc09x_sif.c](#) 第 94 行定义.

3.89.1.5 SIF_Senddata()

```
void SIF_Senddata (
    SIF_TypeDef * SIFx,
    uint8_t Data )
```

发送数据

参数

<i>Data</i>	要发送的数据

在文件 [lks32mc09x_sif.c](#) 第 69 行定义.

3.89.1.6 SIF_StructInit()

```
void SIF_StructInit (
    SIF_InitTypeDef * SIFInitStruct )
```

初始化SIF结构体

参数

<i>SIFInitStruct</i>	指向SIF_InitTypeDef结构体的指针

在文件 [lks32mc09x_sif.c](#) 第 8 行定义.

3.89.1.7 SIF_TH.set()

```
void SIF_TH_set (
    SIF_TypeDef * SIFx,
    uint32_t tsh,
    uint32_t tdh )
```

设置SIF同步和结束信号时长

参数

<i>tsh</i>	同步信号时长
<i>tdh</i>	结束信号时长

在文件 `lks32mc09x_sif.c` 第 59 行定义.

3.89.1.8 SIF_Timebase_set()

```
void SIF_Timebase_set (
    SIF_TypeDef * SIFx,
    uint32_t tosc )
```

设置SIF时基

参数

<code>tosc</code>	时基值

在文件 `lks32mc09x_sif.c` 第 49 行定义.

3.90 lks32mc09x_sif.c

[浏览该文件的文档.](#)

```
00001 #include "lks32mc09x_sif.h"
00002 #include "lks32mc09x_sys.h"
00003
00008 void SIF_StructInit(SIF_InitTypeDef *SIFInitStruct)
00009 {
00010     uint32_t i;
00011     for (i = 0; i < sizeof(SIF_InitTypeDef); i++)
00012     {
00013         ((uint8_t *)SIFInitStruct)[i] = 0;
00014     }
00015 }
00016
00022 void SIF_Init(SIF_TypeDef *SIFx, SIF_InitTypeDef *SIFInitStruct)
00023 {
00024     // 使能SIF模块时钟
00025     SYS_ModuleClockCmd(SYS_MODULE_SIF0, ENABLE);
00026
00027     // 配置SIF寄存器
00028     SIFx->CFG = (SIFInitStruct->SIF_DONE << 7) |
00029                 (SIFInitStruct->SIF_SYNC << 6) |
00030                 (SIFInitStruct->SIF_SYNC_PULSE << 5) |
00031                 (SIFInitStruct->SIF_DONE_VLD << 4) |
00032                 (SIFInitStruct->SIF_SYNC_VLD << 3) |
00033                 (SIFInitStruct->SIF_RATIO << 2) |
00034                 (SIFInitStruct->SIF_MSB << 1) |
00035                 SIFInitStruct->SIF_EN;
00036
00037     SIFx->TOSC = SIFInitStruct->SIF_TOSC;
00038     SIFx->TSTH1 = SIFInitStruct->SIF_TSTH1;
00039     SIFx->TDTH1 = SIFInitStruct->SIF_TDTH1;
00040     SIFx->IRQ = (SIFInitStruct->SIF_IRQ_IF << 4) |
00041                (SIFInitStruct->SIF_DMA_EN << 1) |
00042                SIFInitStruct->SIF_IRQ_EN;
00043 }
00044
00049 void SIF_Timebase_set(SIF_TypeDef *SIFx, uint32_t tosc)
00050 {
00051     SIFx->TOSC = tosc;
00052 }
00053
00059 void SIF_TH_set(SIF_TypeDef *SIFx, uint32_t tsh, uint32_t tdh)
00060 {
00061     SIFx->TSTH1 = tsh;
00062     SIFx->TDTH1 = tdh;
00063 }
00064
00069 void SIF_Senddata(SIF_TypeDef *SIFx, uint8_t Data)
00070 {
00071     SIFx->WDATA = Data;
00072 }
```

```

00073
00078 void SIF_DMA_IsEnable(SIF_TypeDef *SIFx, uint8_t state)
00079 {
00080     if (state)
00081     {
00082         SIFx->IRQ |= SIF_DMA_Enable;
00083     }
00084     else
00085     {
00086         SIFx->IRQ &= ~SIF_DMA_Enable;
00087     }
00088 }
00089
00094 void SIF_Irq_IsEnable(SIF_TypeDef *SIFx, uint8_t state)
00095 {
00096     if (state)
00097     {
00098         SIFx->IRQ |= SIF_IE_Enable;
00099     }
00100     else
00101     {
00102         SIFx->IRQ &= ~SIF_IE_Enable;
00103     }
00104 }
00105
00109 void SIF_Clear_Irq(SIF_TypeDef *SIFx)
00110 {
00111     SIFx->IRQ |= SIF_IF_CLEAR;
00112 }

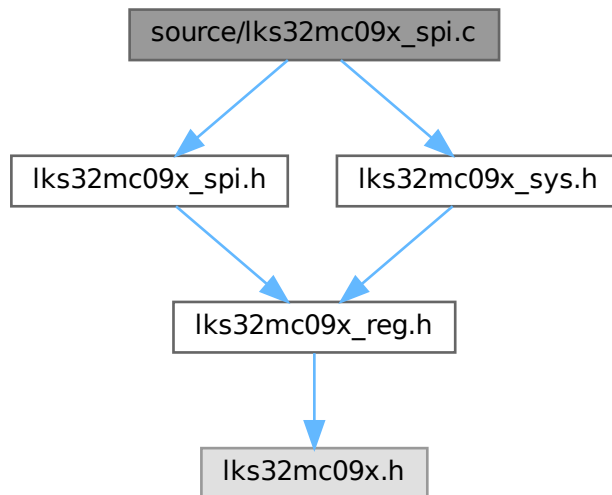
```

3.91 source/lks32mc09x_spi.c 文件参考

```
#include "lks32mc09x_spi.h"
```

```
#include "lks32mc09x_sys.h"
```

lks32mc09x_spi.c 的引用(Include)关系图:



函数

- void **SPI_Init** (**SPI_TypeDef** *SPIx, **SPI_InitTypeDef** *SPI_InitStruct)

- SPI*模块初始化函数
- void **SPI_StructInit** (**SPI_InitTypeDef** *SPI_InitStruct)
*SPI*初始化结构体默认配置函数
 - void **SPI_SendData** (**SPI_TypeDef** *SPIx, uint8_t n)
*SPI*发送数据函数
 - uint8_t **SPI_ReadData** (**SPI_TypeDef** *SPIx)
*SPI*读取数据函数
 - uint8_t **SPI_GetIRQFlag** (**SPI_TypeDef** *SPIx, uint8_t IRQ_Flag)
获取*SPI*中断标志位状态函数
 - void **SPI_ClearIRQFlag** (**SPI_TypeDef** *SPIx, uint8_t IRQ_Flag)
清除*SPI*中断标志位函数

3.91.1 函数说明

3.91.1.1 SPI_ClearIRQFlag()

```
void SPI_ClearIRQFlag (  
    SPI_TypeDef * SPIx,  
    uint8_t IRQ_Flag )
```

清除SPI中断标志位函数

参数

<i>SPIx</i>	SPI模块指针
<i>IRQ_Flag</i>	要清除的中断标志位

在文件 [lks32mc09x_spi.c](#) 第 84 行定义.

3.91.1.2 SPI_GetIRQFlag()

```
uint8_t SPI_GetIRQFlag (  
    SPI_TypeDef * SPIx,  
    uint8_t IRQ_Flag )
```

获取SPI中断标志位状态函数

参数

<i>SPIx</i>	SPI模块指针
<i>IRQ_Flag</i>	中断标志位

返回
中断标志位状态

在文件 [lks32mc09x_spi.c](#) 第 74 行定义.

3.91.1.3 SPI_Init()

```
void SPI_Init (
    SPI_TypeDef * SPIx,
    SPI_InitTypeDef * SPI_InitStruct )
```

SPI模块初始化函数

参数

<i>SPIx</i>	SPI模块指针
<i>SPI_InitStruct</i>	指向SPI初始化结构体的指针

在文件 [lks32mc09x_spi.c](#) 第 9 行定义.

函数调用图:



3.91.1.4 SPI_ReadData()

```
uint8_t SPI_ReadData (
    SPI_TypeDef * SPIx )
```

SPI读取数据函数

参数

<i>SPIx</i>	SPI模块指针

返回
读取到的数据

在文件 [lks32mc09x_spi.c](#) 第 63 行定义.

3.91.1.5 SPI_SendData()

```
void SPI_SendData (
    SPI_TypeDef * SPIx,
    uint8_t n )
```

SPI发送数据函数

参数

<i>SPIx</i>	SPI模块指针
<i>n</i>	要发送的数据

在文件 [lks32mc09x_spi.c](#) 第 53 行定义.

3.91.1.6 SPI_StructInit()

```
void SPI_StructInit (  
    SPI_InitTypeDef * SPI_InitStruct )
```

SPI初始化结构体默认配置函数

参数

<i>SPI_InitStruct</i>	指向SPI初始化结构体的指针

在文件 [lks32mc09x_spi.c](#) 第 39 行定义.

3.92 lks32mc09x_spi.c

[浏览该文件的文档.](#)

```
00001 #include "lks32mc09x_spi.h"  
00002 #include "lks32mc09x_sys.h"  
00003  
00009 void SPI_Init(SPI_TypeDef *SPIx, SPI_InitTypeDef *SPI_InitStruct)  
00010 {  
00011     // 使能SPI模块时钟  
00012     SYS_ModuleClockCmd(SYS_MODULE_SPI0, ENABLE);  
00013  
00014     // 配置SPI控制寄存器  
00015     SPIx->CFG = (SPI_InitStruct->Duplex << 6) |  
00016                 (SPI_InitStruct->CS << 5) |  
00017                 (SPI_InitStruct->Mode << 4) |  
00018                 (SPI_InitStruct->CPHA << 3) |  
00019                 (SPI_InitStruct->CPOL << 2) |  
00020                 (SPI_InitStruct->ENDIAN << 1) |  
00021     SPI_InitStruct->EN;  
00022  
00023     // 配置SPI中断使能寄存器  
00024     SPIx->IE = (SPI_InitStruct->IRQEna << 4) |  
00025     (SPI_InitStruct->Trig << 3);  
00026  
00027     // 配置SPI波特率  
00028     SPIx->DIV = (SPI_InitStruct->TRANS_MODE << 15) |  
00029     (SPI_InitStruct->BaudRate);  
00030  
00031     // 配置SPI数据长度  
00032     SPIx->SIZE = SPI_InitStruct->ByteLength;  
00033 }  
00034  
00039 void SPI_StructInit(SPI_InitTypeDef *SPI_InitStruct)  
00040 {  
00041     uint32_t i;  
00042     for (i = 0; i < sizeof(SPI_InitTypeDef); i++)  
00043     {  
00044         ((uint8_t *)SPI_InitStruct)[i] = 0;  
00045     }  
00046 }  
00047  
00053 void SPI_SendData(SPI_TypeDef *SPIx, uint8_t n)  
00054 {  
00055     SPIx->TX_DATA = n;
```

```

00056 }
00057
00063 uint8_t SPI_ReadData(SPI_TypeDef *SPIx)
00064 {
00065     return (uint8_t)SPIx->RX_DATA;
00066 }
00067
00074 uint8_t SPI_GetIRQFlag(SPI_TypeDef *SPIx, uint8_t IRQ_Flag)
00075 {
00076     return (SPIx->IE & IRQ_Flag) ? 1 : 0;
00077 }
00078
00084 void SPI_ClearIRQFlag(SPI_TypeDef *SPIx, uint8_t IRQ_Flag)
00085 {
00086     SPIx->IE = (SPIx->IE & 0xf8) | IRQ_Flag;
00087 }

```

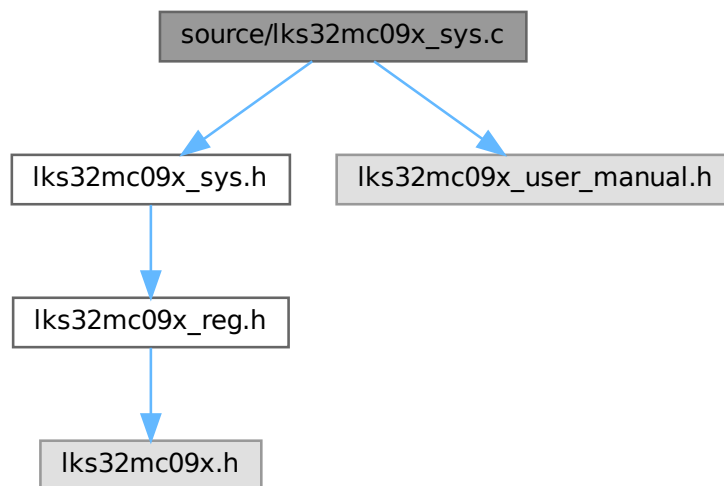
3.93 source/lks32mc09x_sys.c 文件参考

```

#include "lks32mc09x_sys.h"
#include "lks32mc09x_user_manual.h"

```

lks32mc09x_sys.c 的引用(Include)关系图:



函数

- void **SYS_ModuleClockCmd** (uint32_t nModule, uint8_t state)
数字模块时钟使能
- void **SYS_SoftResetModule** (uint32_t nModule)
软复位模块
- void **SYS_SoftReset_all** (void)
软复位所有模块
- void **SYS_MclkChoice** (uint32_t SYS_MCLK_x)
选择系统时钟
- void **SYS_UartClkDiv** (uint16_t div)

- UART时钟分频
- void [SYS_I2CClkDiv](#) (uint16_t div)
- I2C时钟分频
- void [SYS_SPIClkDiv](#) (uint16_t div)
- SP时钟分频
- uint32_t [SYS_ReadMcuClk](#) (void)
- 获取当前MCU时钟
- void [SYS_SetPowerFailureDetection](#) (uint32_t SYS_PVDSEL_x)
- 掉电检测配置
- void [SYS_WaitPowerReady](#) (uint32_t SYS_PVDSEL_x)
- 等待电源就绪
- void [SYS_ResetPeripheral](#) (void)
- 复位所有外设到默认状态（芯片上电后的状态）
- void [SYS_ResetPeripheralInternal](#) (void)
- 复位所有内部外设为默认值(除io的所有模块)
- void [SYS_ResetIO](#) (void)
- 复位所有IO为默认值（gpio以及部分和io有关的sys寄存器）

3.93.1 函数说明

3.93.1.1 SYS_I2CClkDiv()

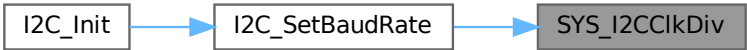
```
void SYS_I2CClkDiv (  
    uint16_t div )
```

I2C时钟分频

参数

<i>div</i>	分频系数

在文件 [lks32mc09x_sys.c](#) 第 91 行定义.
这是这个函数的调用关系图:



3.93.1.2 SYS_MclkChoice()

```
void SYS_MclkChoice (  
    uint32_t SYS_MCLK_x )
```

选择系统时钟

参数

<i>SYS_↔ MCLK_x</i>	时钟源选择

参见

使用外部晶振时，除了晶振使能外，还需要配置 `SYS_AFE_REG1_XTAL_FLT_BIT` [9] 晶振低通滤波关闭 1:关闭 0:打开 `SYS_AFE_REG1_XTAL_SCHT_BIT` [4] 晶振选择反相器或者施密特 1:反相器 0:施密特 `SYS_AFE_REG2_XTAL_RSEL_BIT` [8] 晶体起振电路电阻调节 0:P端电阻维持原值 1: P端电阻增加一倍

在文件 `lks32mc09x_sys.c` 第 54 行定义.

3.93.1.3 `SYS_ModuleClockCmd()`

```
void SYS_ModuleClockCmd (
    uint32_t nModule,
    uint8_t state )
```

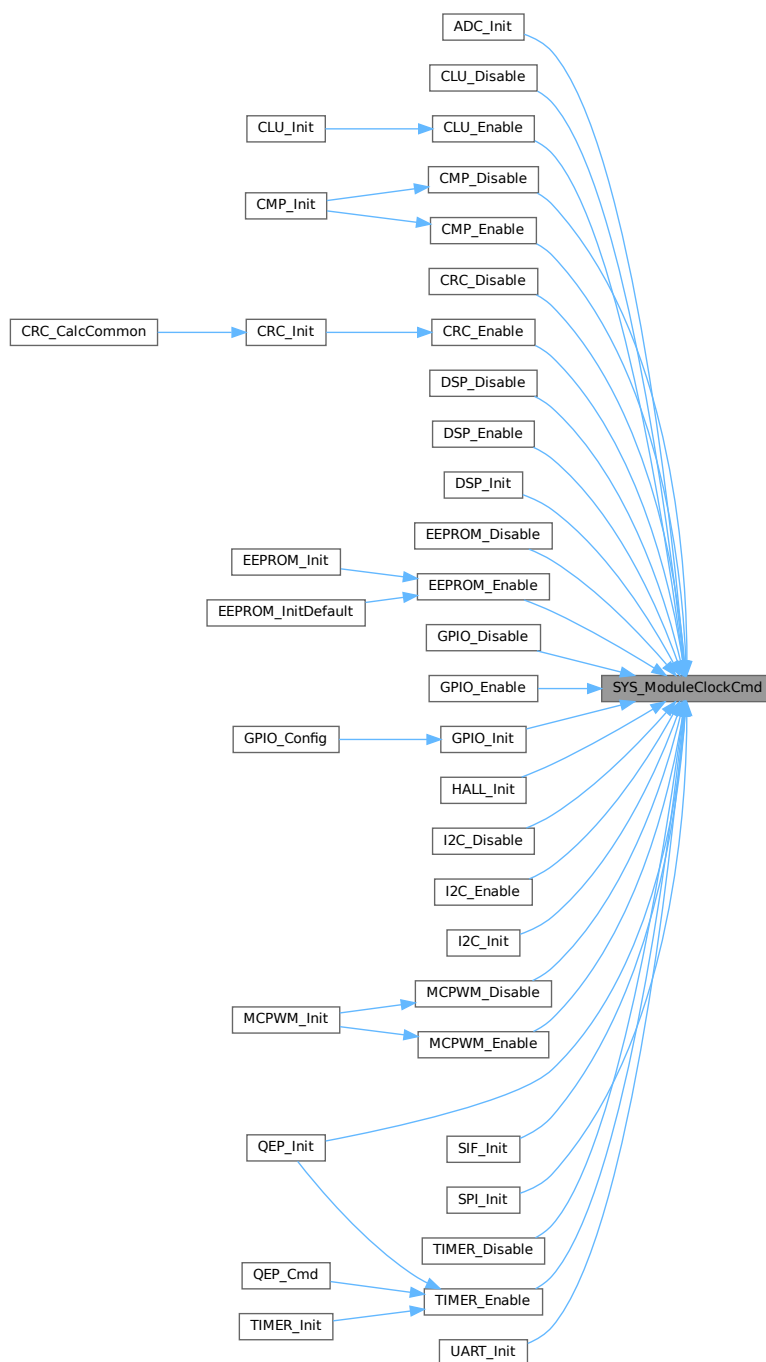
数字模块时钟使能

参数

<i>nModule</i>	模块编号
<i>state</i>	使能或禁用状态

在文件 `lks32mc09x_sys.c` 第 9 行定义.

这是这个函数的调用关系图:



3.93.1.4 SYS_ReadMcuClk()

```
uint32_t SYS_ReadMcuClk (
    void )
```

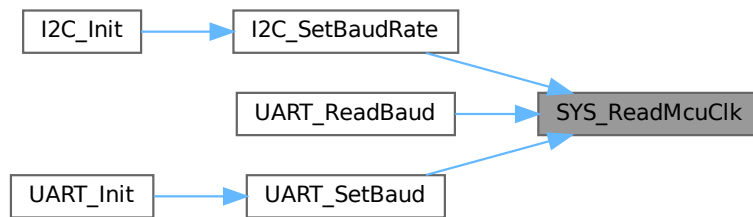
获取当前MCU时钟

返回

当前MCU时钟频率

在文件 [lks32mc09x_sys.c](#) 第 115 行定义.

这是这个函数的调用关系图:



3.93.1.5 SYS_ResetIO()

```
void SYS_ResetIO (
    void )
```

复位所有IO为默认值（gpio以及部分和io有关的sys寄存器）

在文件 [lks32mc09x_sys.c](#) 第 308 行定义.

这是这个函数的调用关系图:



3.93.1.6 SYS_ResetPeripheral()

```
void SYS_ResetPeripheral (
    void )
```

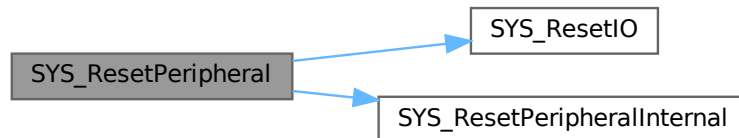
复位所有外设到默认状态（芯片上电后的状态）

警告

不复位的寄存器有：复位事件记录寄存器 后备存储器 ram 校准寄存器（仅限于上电会自动加载的校准寄存器）只读的寄存器 部分flash寄存器（如FLASH->WDATA等，写入操作无意义）所有内核的寄存器

在文件 [lks32mc09x.sys.c](#) 第 238 行定义.

函数调用图:

**3.93.1.7 SYS_ResetPeripheralInternal()**

```
void SYS_ResetPeripheralInternal (
    void )
```

复位所有内部外设为默认值(除io的所有模块)

警告

不复位的寄存器有：复位事件记录寄存器 后备存储器 ram 校准寄存器（仅限于上电会自动加载的校准寄存器）只读的寄存器 部分flash寄存器（如FLASH->WDATA等，写入操作无意义）所有内核的寄存器

在文件 [lks32mc09x.sys.c](#) 第 254 行定义.

这是这个函数的调用关系图:

**3.93.1.8 SYS_SetPowerFailureDetection()**

```
void SYS_SetPowerFailureDetection (
    uint32_t SYS_PVDSEL_x )
```

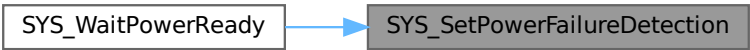
掉电检测配置

参数

SYS_↔ PVDSEL_x	掉电检测配置值，可选： • SYS_PVDSEL_4_0V 4.0V • SYS_PVDSEL_3_74V 3.74V • SYS_PVDSEL_3_49V 3.49V • SYS_PVDSEL_2_8V 2.8V • SYS_PVDSEL_DAC DAC (这里使用的是DAC0) • SYS_PVDSEL_OFF OFF

在文件 [lks32mc09x-sys.c](#) 第 178 行定义.

这是这个函数的调用关系图:



3.93.1.9 SYS_SoftReset_all()

```
void SYS_SoftReset_all (
    void )
```

软复位所有模块

在文件 [lks32mc09x-sys.c](#) 第 38 行定义.

3.93.1.10 SYS_SoftResetModule()

```
void SYS_SoftResetModule (
    uint32_t nModule )
```

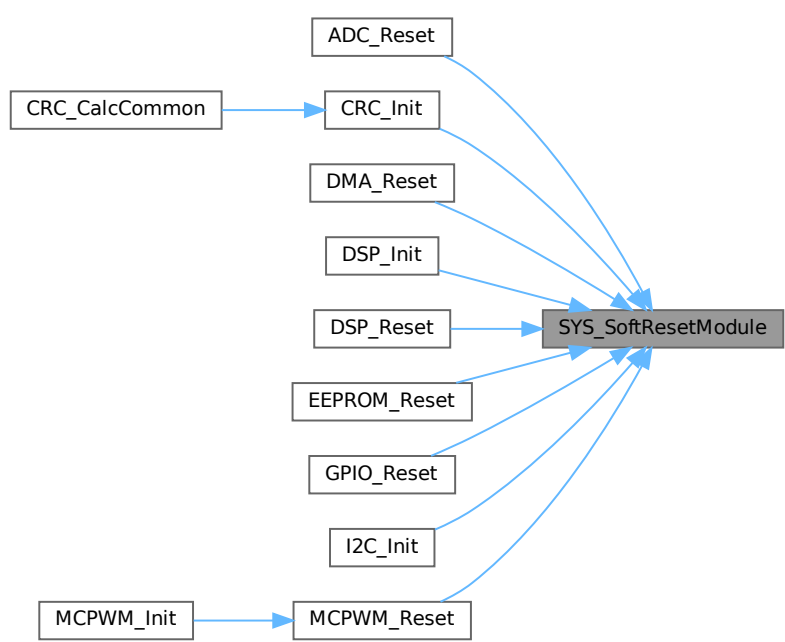
软复位模块

参数

<i>nModule</i>	模块编号

在文件 [lks32mc09x-sys.c](#) 第 27 行定义.

这是这个函数的调用关系图:



3.93.1.11 SYS.SPIClkDiv()

```
void SYS_SPIClkDiv (
    uint16_t div )
```

SPI时钟分频

参数

<i>div</i>	分频系数

在文件 [lks32mc09x_sys.c](#) 第 102 行定义.

3.93.1.12 SYS.UartClkDiv()

```
void SYS_UartClkDiv (
    uint16_t div )
```

UART时钟分频

参数

<i>div</i>	分频系数

在文件 `lks32mc09x_sys.c` 第 80 行定义.

这是这个函数的调用关系图:



3.93.1.13 SYS.WaitPowerReady()

```
void SYS.WaitPowerReady (
    uint32_t SYS_PVDSEL_x )
```

等待电源就绪

参数

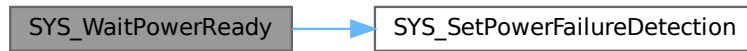
<i>SYS_↔</i> <i>PVDSEL_x</i>	电源电压检测阈值配置，可选： • SYS_PVDSEL_4.0V 4.0V • SYS_PVDSEL_3.74V 3.74V • SYS_PVDSEL_3.49V 3.49V • SYS_PVDSEL_2.8V 2.8V 注意：SYS_PVDSEL_DAC暂不支持

注解

```
使用前确保已配置PWRDN中断（如需要）
void SystemInit()
{
    SYS.WaitPowerReady(SYS_PVDSEL_4.0V); // 等待电源就绪
    NVIC.SetPriority(PWRDN_IRQn, 0);      // 掉电检测中断优先级配置
    NVIC.EnableIRQ(PWRDN_IRQn);          // 使能掉电检测中断
    // 其他初始化代码
}
void PWRDN_IRQHandler(void) {
    __disable_irq();
    SYS.ResetPeripheral();
    NVIC.SystemReset();
}
```

在文件 `lks32mc09x_sys.c` 第 209 行定义.

函数调用图:



3.94 lks32mc09x_sys.c

[浏览该文件的文档.](#)

```

00001 #include "lks32mc09x_sys.h"
00002 #include "lks32mc09x_user_manual.h"
00003
00009 void SYS.ModuleClockCmd(uint32_t nModule, uint8_t state)
00010 {
00011     REG_WRITE(SYS0->PROTECT, 0x7a83);
00012     if (state)
00013     {
00014         REG_SET(SYS0->CLK_FEN, nModule);
00015     }
00016     else
00017     {
00018         REG_RESET(SYS0->CLK_FEN, nModule);
00019     }
00020     REG_WRITE(SYS0->PROTECT, 0);
00021 }
00022
00027 void SYS.SoftResetModule(uint32_t nModule)
00028 {
00029     REG_WRITE(SYS0->PROTECT, 0x7a83);
00030     REG_SET(SYS0->SFT_RST, nModule);
00031     REG_RESET(SYS0->SFT_RST, nModule); // 触发复位后需清除复位位
00032     REG_WRITE(SYS0->PROTECT, 0);
00033 }
00034
00038 void SYS.SoftReset_all(void)
00039 {
00040     REG_WRITE(SYS0->PROTECT, 0x7a83);
00041     REG_WRITE(SYS0->SFT_RST, 0xFFFFFFFF);
00042     REG_WRITE(SYS0->SFT_RST, 0x00000000); // 触发复位后需清除所有复位位
00043     REG_WRITE(SYS0->PROTECT, 0);
00044 }
00045
00054 void SYS.MclkChoice(uint32_t SYS_MCLK_x)
00055 {
00056     REG_WRITE(SYS0->PROTECT, 0x7a83);
00057     if (SYS_MCLK_x & 0x8000) // 使用外部晶振
00058     {
00059         REG_SET(SYS_AFE_REG5, BIT4);
00060     }
00061     else
00062     {
00063         REG_RESET(SYS_AFE_REG5, BIT4);
00064     }
00065     REG_WRITE_BIT(SYS0->CLK_CFG, 0x3ff, SYS_MCLK_x);
00066     {
00067         int cnt;
00068         for (cnt = 0; cnt < 4000; cnt++)
00069         {
00070             __NOP();
00071         }
00072     }
00073     REG_WRITE(SYS0->PROTECT, 0);
00074 }
00075
00080 void SYS.UartClkDiv(uint16_t div)
00081 {
00082     REG_WRITE(SYS0->PROTECT, 0x7a83);
00083     REG_WRITE(SYS0->CLK_DIV2, div);
00084     REG_WRITE(SYS0->PROTECT, 0);
00085 }
00086
  
```

```

00091 void SYS_I2CClkDiv(uint16_t div)
00092 {
00093     REG_WRITE(SYS0->PROTECT, 0x7a83);
00094     REG_WRITE(SYS0->CLK_DIV0, div);
00095     REG_WRITE(SYS0->PROTECT, 0);
00096 }
00097
00102 void SYS_SPIClkDiv(uint16_t div)
00103 {
00104     // 保留接口, 09x的SPI时钟只需要配置模块本身的分频即可
00105     if (div != 0)
00106     {
00107         __NOP();
00108     }
00109 }
00110
00115 uint32_t SYS_ReadMcuClk(void)
00116 {
00117     uint32_t clk = 96000000;
00118     uint32_t clkdiv = (SYS_CLK_CFG & SYS_CLK_CFG_CLK_DIV_MASK) >> SYS_CLK_CFG_CLK_DIV_POS;
00119     uint32_t clkssel = (SYS_CLK_CFG & SYS_CLK_CFG_CLK_SEL_MASK) >> SYS_CLK_CFG_CLK_SEL_POS;
00120     switch (clkssel) // 0: HRC 1: PLL 2: LRC
00121     {
00122         case 0: // HRC
00123             clk = 8000000;
00124             break;
00125         case 1: // PLL
00126             switch (clkdiv)
00127             {
00128                 case 0xff: // 96M
00129                     clk = 96000000;
00130                     break;
00131                 case 0x55: // 48M
00132                     clk = 48000000;
00133                     break;
00134                 case 0x11: // 24M
00135                     clk = 24000000;
00136                     break;
00137                 case 0x01: // 12M
00138                     clk = 12000000;
00139                     break;
00140                 case 0x00: // 12M
00141                     clk = 12000000;
00142                     break;
00143                 default: // 逐位计算
00144                     clk = 0;
00145                     for (int i = 0; i < 8; i++)
00146                     {
00147                         if (clkdiv & (1 << i))
00148                         {
00149                             clk += 12000000;
00150                         }
00151                         else
00152                         {
00153                             __NOP();
00154                         }
00155                     }
00156                     break;
00157             }
00158             break;
00159         case 2: // LRC
00160             clk = 32000;
00161             break;
00162         default:
00163             break;
00164     }
00165     return clk;
00166 }
00167
00178 void SYS_SetPowerFailureDetection(uint32_t SYS_PVDSEL_x)
00179 {
00180     REG_WRITE(SYS0->PROTECT, 0x7a83);
00181     REG_WRITE_BIT(SYS0->AFE_REG5, SYS_AFE_REG5_RSTSEL_MASK, SYS_AFE_REG5_RSTSEL(1));
00182     REG_WRITE_BIT(SYS0->AFE_REG0, (SYS_AFE_REG0_PVDSEL_MASK | SYS_AFE_REG0_PD_PDT_BIT |
SYS_AFE_REG0_VSR_PDT_BIT), SYS_PVDSEL_x);
00183     REG_WRITE(SYS0->PROTECT, 0);
00184 }
00209 void SYS_WaitPowerReady(uint32_t SYS_PVDSEL_x)
00210 {
00211     // 通常SYS_WaitPowerReady执行时全局中断处于关闭状态。
00212     // 为了确保兼容性, 这里使用原子锁进行处理。
00213     uint32_t flg = __get_PRIMASK();
00214     __disable_irq();
00215     // 设置掉电检测电压
00216     SYS_SetPowerFailureDetection(SYS_PVDSEL_x);
00217     // 等待电源就绪
00218     while (REG_READ_BIT(SYS0->AFE_DBG, SYS_AFE_DBG_PWR_WEAK_BIT))

```

```

00219     {
00220         __NOP();
00221     }
00222     // 电源就绪后清除中断标志位, 上电过程中一定会触发一次掉电中断
00223     NVIC_ClearPendingIRQ(PWRDN_IRQn);
00224     __set_PRIMASK(flag); // 还原中断状态
00225 }
00226
00238 void SYS_ResetPeripheral(void)
00239 {
00240     SYS_ResetIO();
00241     SYS_ResetPeripheralInternal();
00242 }
00254 void SYS_ResetPeripheralInternal(void)
00255 {
00256     uint32_t flg = __get_PRIMASK();
00257     __disable_irq();
00258     SYS->PROTECT = 0x7a83;
00259     SYS->SFT_RST = (0xffffffff & ~BIT11); // 复位除GPIO之外的所有模块
00260     SYS->SFT_RST = 0;
00261     SYS->AFE_REG0 = 0;
00262     SYS->AFE_REG1 = 0;
00263     SYS->AFE_REG2 = 0;
00264     SYS->AFE_REG3 = 0;
00265     SYS->AFE_REG4 = 0;
00266     SYS->AFE_REG5 = 0;
00267     SYS->AFE_REG7 = 0;
00268     SYS->AFE_DAC_CTRL = 0;
00269     SYS->AFE_DAC0 = 0;
00270     SYS->AFE_DAC1 = 0;
00271
00272     SYS->CLK_CFG = 0;
00273     {
00274         int cnt;
00275         for (cnt = 0; cnt < 4000; cnt++)
00276         {
00277             __NOP();
00278         }
00279     }
00280     SYS->IO_CFG = 0;
00281     SYS->DBG_CFG = 0;
00282     SYS->CLK_DIV0 = 0;
00283     SYS->CLK_DIV1 = 0;
00284     SYS->CLK_DIV2 = 0;
00285     SYS->CLK_FEN = 0;
00286     SYS->RAM_LOCK = 0;
00287     SYS->SFT_RST = 0;
00288     SYS->FLSE = 0;
00289     SYS->FLSP = 0;
00290     SYS->PROTECT = 0;
00291
00292     FLASH->CFG = 0xe0;
00293     FLASH->ADDR = 0;
00294     FLASH->NADDR = 0;
00295     FLASH->NKEY = 0;
00296     {
00297         uint32_t reg = FLASH->PROTECT; // 更新flash状态
00298         if (reg != 0)
00299         {
00300             __NOP();
00301         }
00302     }
00303     __set_PRIMASK(flag);
00304 }
00308 void SYS_ResetIO(void)
00309 {
00310     uint32_t flg = __get_PRIMASK();
00311     __disable_irq();
00312     SYS_WR_PROTECT = 0x7a83;
00313     SYS_SFT_RST = BIT11; // 复位GPIO, GPIO0 GPIO1 GPIO2三个模块的所有寄存器
00314     SYS_SFT_RST = 0;
00315     SYS_AFE_REG1 &= ~BIT8; // 关闭LDO输出到P0.0
00316     SYS_AFE_REG2 &= ~(BIT15 | BIT14 | BIT13 | BIT11 | BIT10 | 0xff); // 关闭dac、vref、vbgp、opa输出
到P0.0
00317     SYS_IO_CFG = 0; // 恢复SWD&RST为默认状态
00318     SYS_WR_PROTECT = 0;
00319     __set_PRIMASK(flag);
00320 }

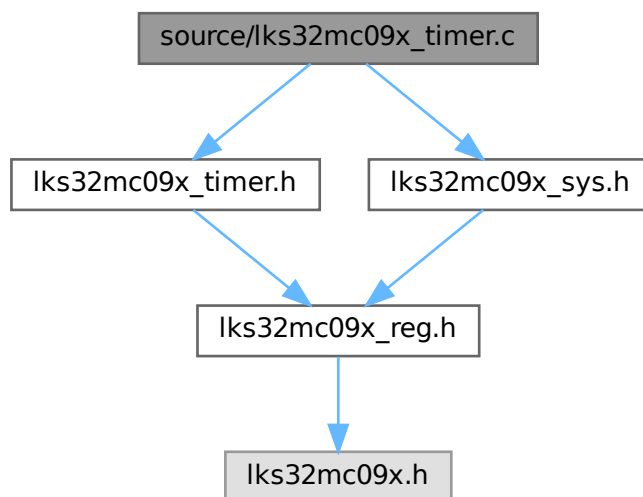
```

3.95 source/lks32mc09x_timer.c 文件参考

```
#include "lks32mc09x_timer.h"
```

```
#include "lks32mc09x_sys.h"
```

lks32mc09x_timer.c 的引用(Include)关系图:



函数

- void **TIMER_Init** (TIMER_TypeDef *TIMERx, TIMER_TimerInitTypeDef *TIMER_InitStruct)
Timer初始化
- void **TIMER_StructInit** (TIMER_TimerInitTypeDef *TIMER_InitStruct)
Timer结构体初始化
- void **TIMER_TimerCmd** (TIMER_TypeDef *TIMERx, uint8_t state)
Timer使能/失能
- void **TIMER_CMP0** (TIMER_TypeDef *TIMERx, uint32_t temp)
设置Timer通道0比较值
- void **TIMER_CMP1** (TIMER_TypeDef *TIMERx, uint32_t temp)
设置Timer通道1比较值
- uint32_t **TIMER_Timer_GetCMPT0** (TIMER_TypeDef *TIMERx)
获取Timer通道0比较值
- uint32_t **TIMER_Timer_GetCMPT1** (TIMER_TypeDef *TIMERx)
获取Timer通道1比较值
- void **TIMER_ClearIRQFlag** (TIMER_TypeDef *TIMERx, uint32_t tempFlag)
清除Timer中断标志
- uint16_t **TIMER_GetIRQFlag** (TIMER_TypeDef *TIMERx, uint32_t TIMER_IF_x)
获取Timer中断标志状态
- uint16_t **TIMER_GetFlag** (TIMER_TypeDef *TIMERx, uint32_t TIMER_IF_x)
获取Timer状态标志
- uint32_t **TIMER_Timer_GetCount** (TIMER_TypeDef *TIMERx)

- 获取*Timer*当前计数值
- void **TIMER_Enable** (**TIMER_TypeDef** *TIMERx)
使能*Timer*
 - void **TIMER_Disable** (**TIMER_TypeDef** *TIMERx)
失能*Timer*
 - void **TIMER_SetOutputState** (**TIMER_TypeDef** *TIMERx, uint8_t state)
设置输出状态

3.95.1 函数说明

3.95.1.1 TIMER_ClearIRQFlag()

```
void TIMER_ClearIRQFlag (  
    TIMER_TypeDef * TIMERx,  
    uint32_t tempFlag )
```

清除Timer中断标志

参数

<i>TIMERx</i>	Timer实例指针
<i>tempFlag</i>	要清除的中断标志

在文件 [lks32mc09x_timer.c](#) 第 150 行定义.

3.95.1.2 TIMER_CMP0()

```
void TIMER_CMP0 (  
    TIMER_TypeDef * TIMERx,  
    uint32_t temp )
```

设置Timer通道0比较值

参数

<i>TIMERx</i>	Timer实例指针
<i>temp</i>	比较值

在文件 [lks32mc09x_timer.c](#) 第 110 行定义.

3.95.1.3 TIMER_CMP1()

```
void TIMER_CMP1 (  
    TIMER_TypeDef * TIMERx,  
    uint32_t temp )
```

设置Timer通道1比较值

参数

<i>TIMERx</i>	Timer实例指针
<i>temp</i>	比较值

在文件 [lks32mc09x_timer.c](#) 第 120 行定义.

3.95.1.4 **TIMER_Disable()**

```
void TIMER_Disable (
    TIMER_TypeDef * TIMERx )
```

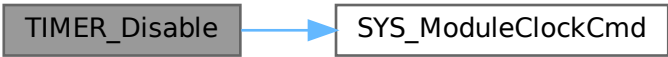
失能Timer

参数

<i>TIMERx</i>	Timer实例指针

在文件 [lks32mc09x_timer.c](#) 第 215 行定义.

函数调用图:



3.95.1.5 **TIMER_Enable()**

```
void TIMER_Enable (
    TIMER_TypeDef * TIMERx )
```

使能Timer

参数

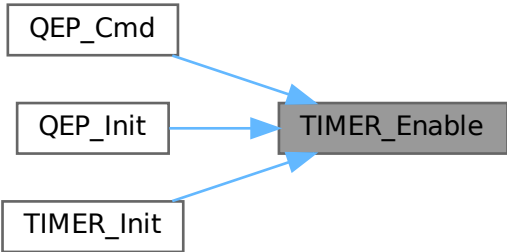
<i>TIMERx</i>	Timer实例指针

在文件 [lks32mc09x_timer.c](#) 第 191 行定义.

函数调用图:



这是这个函数的调用关系图:



3.95.1.6 TIMER_GetFlag()

```
uint16_t TIMER_GetFlag (
    TIMER_TypeDef * TIMERx,
    uint32_t TIMER_IF_x )
```

获取Timer状态标志

参数

<i>TIMERx</i>	Timer实例指针
<i>TIMER_↔IF_x</i>	状态标志位

返回

状态标志

在文件 [lks32mc09x.timer.c](#) 第 172 行定义.

3.95.1.7 TIMER_GetIRQFlag()

```
uint16_t TIMER_GetIRQFlag (
    TIMER_TypeDef * TIMERx,
    uint32_t TIMER_IF_x )
```

获取Timer中断标志状态

参数

<i>TIMERx</i>	Timer实例指针
<i>TIMER_IF_x</i>	中断标志位

返回
中断标志状态

在文件 [lks32mc09x_timer.c](#) 第 161 行定义.

3.95.1.8 TIMER_Init()

```
void TIMER_Init (
    TIMER_TypeDef * TIMERx,
    TIMER_TimerInitTypeDef * TIMER_InitStruct )
```

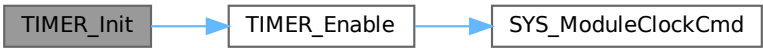
Timer初始化

参数

<i>TIMERx</i>	Timer实例指针
<i>TIMER_InitStruct</i>	Timer初始化结构体指针

在文件 [lks32mc09x_timer.c](#) 第 9 行定义.

函数调用图:



3.95.1.9 TIMER_SetOutputState()

```
void TIMER_SetOutputState (
    TIMER_TypeDef * TIMERx,
    uint8_t state )
```

设置输出状态

参数

<i>TIMERx</i>	Timer实例指针
<i>state</i>	输出状态，0关闭pwm输出，其他值，打开pwm输出

在文件 [lks32mc09x_timer.c](#) 第 240 行定义.

3.95.1.10 **TIMER_StructInit()**

```
void TIMER_StructInit (
    TIMER_TimerInitTypeDef * TIMER_InitStruct )
```

Timer结构体初始化

参数

<i>TIMER_InitStruct</i>	Timer初始化结构体指针

在文件 [lks32mc09x_timer.c](#) 第 79 行定义.

3.95.1.11 **TIMER_Timer_GetCMPT0()**

```
uint32_t TIMER_Timer_GetCMPT0 (
    TIMER_TypeDef * TIMERx )
```

获取Timer通道0比较值

参数

<i>TIMERx</i>	Timer实例指针

返回
比较值

在文件 [lks32mc09x_timer.c](#) 第 130 行定义.

3.95.1.12 **TIMER_Timer_GetCMPT1()**

```
uint32_t TIMER_Timer_GetCMPT1 (
    TIMER_TypeDef * TIMERx )
```

获取Timer通道1比较值

参数

<i>TIMERx</i>	Timer实例指针

返回
比较值

在文件 [lks32mc09x_timer.c](#) 第 140 行定义.

3.95.1.13 **TIMER_Timer_GetCount()**

```
uint32_t TIMER_Timer_GetCount (
    TIMER_TypeDef * TIMERx )
```

获取Timer当前计数值

参数

<i>TIMERx</i>	Timer实例指针

返回
当前计数值

在文件 [lks32mc09x_timer.c](#) 第 182 行定义.

3.95.1.14 **TIMER_TimerCmd()**

```
void TIMER_TimerCmd (
    TIMER_TypeDef * TIMERx,
    uint8_t state )
```

Timer使能/失能

参数

<i>TIMERx</i>	Timer实例指针
<i>state</i>	使能或失能状态

在文件 [lks32mc09x_timer.c](#) 第 93 行定义.

3.96 **lks32mc09x_timer.c**

[浏览该文件的文档.](#)

```

00001 #include "lks32mc09x.timer.h"
00002 #include "lks32mc09x.sys.h"
00003
00009 void TIMER_Init(TIMER_TypeDef *TIMERx, TIMER_TimerInitTypeDef *TIMER_InitStruct)
00010 {
00011     uint32_t th = TIMER_InitStruct->TH;
00012     TIMER_Enable(TIMERx);
00013     if (TIMERx != TIMER2) // 只有timer2是32位定时器
00014     {
00015         if (th > 0xffff)
00016         {
00017             th = 0xffff;
00018         }
00019         else
00020         {
00021             __NOP();
00022         }
00023     }
00024     else
00025     {
00026         __NOP();
00027     }
00028     // 配置Timer寄存器
00029     TIMERx->CFG2 = (TIMER_InitStruct->EN << 15) |
00030                   (TIMER_InitStruct->ONE.TRIG << 9) |
00031                   (TIMER_InitStruct->CENTER << 8) |
00032                   (TIMER_InitStruct->DIR << 6) |
00033                   (TIMER_InitStruct->ClockDiv << 4) |
00034                   (TIMER_InitStruct->ETON << 3) |
00035                   (TIMER_InitStruct->GATE_EN << 2) |
00036                   (TIMER_InitStruct->RL_EN << 1) |
00037                   TIMER_InitStruct->XCLK_EN;
00038
00039     TIMERx->CFG0 = (TIMER_InitStruct->CMP0.TRIGGER_MODE << 9) |
00040                   (TIMER_InitStruct->CAP0.CLR_EN << 8) |
00041                   (TIMER_InitStruct->SRC0 << 4) |
00042                   (TIMER_InitStruct->CH0Output << 3) |
00043                   (TIMER_InitStruct->CH0.WorkMode << 2) |
00044                   (TIMER_InitStruct->CH0.FE.CAP_EN << 1) |
00045                   (TIMER_InitStruct->CH0.RE.CAP_EN << 0);
00046
00047     TIMERx->CFG1 = (TIMER_InitStruct->CMP1.TRIGGER_MODE << 9) |
00048                   (TIMER_InitStruct->CAP1.CLR_EN << 8) |
00049                   (TIMER_InitStruct->SRC1 << 4) |
00050                   (TIMER_InitStruct->CH1Output << 3) |
00051                   (TIMER_InitStruct->CH1.WorkMode << 2) |
00052                   (TIMER_InitStruct->CH1.FE.CAP_EN << 1) |
00053                   (TIMER_InitStruct->CH1.RE.CAP_EN << 0);
00054
00055     TIMERx->CHN0 = TIMER_InitStruct->CHN0;
00056     TIMERx->CHN1 = TIMER_InitStruct->CHN1;
00057     TIMERx->TH = th;
00058     TIMERx->CNT = TIMER_InitStruct->CNT;
00059     TIMERx->EVT = TIMER_InitStruct->EVT;
00060     TIMERx->FLT = TIMER_InitStruct->FLT;
00061     TIMERx->IE = TIMER_InitStruct->IE;
00062
00063     TIMERx->IO = (TIMER_InitStruct->HALT_PRT << 7) |
00064                 (TIMER_InitStruct->CH0.DEFAULT << 8) |
00065                 (TIMER_InitStruct->CH1.DEFAULT << 9) |
00066                 (TIMER_InitStruct->MOE << 6) |
00067                 (TIMER_InitStruct->FAIL_SEL << 2) |
00068                 (TIMER_InitStruct->FAIL_POL << 1) |
00069                 (TIMER_InitStruct->FAIL_EN << 0);
00070
00071     // 影子寄存器配置
00072     TIMERx->CFG2 |= (TIMER_InitStruct->SHADOW << 10);
00073 }
00074
00079 void TIMER_StructInit(TIMER_TimerInitTypeDef *TIMER_InitStruct)
00080 {
00081     uint32_t i;
00082     for (i = 0; i < sizeof(TIMER_TimerInitTypeDef); i++)
00083     {
00084         ((uint8_t *)TIMER_InitStruct)[i] = 0;
00085     }
00086 }
00087
00093 void TIMER_TimerCmd(TIMER_TypeDef *TIMERx, uint8_t state)
00094 {
00095     if (state == ENABLE)
00096     {
00097         TIMERx->CFG2 |= (1 << 15);
00098     }
00099     else
00100     {
00101         TIMERx->CFG2 &= ~(1 << 15);

```

```

00102     }
00103 }
00104
00110 void TIMER_CMP0(TIMER_TypeDef *TIMEx, uint32_t temp)
00111 {
00112     TIMEx->CHN0 = temp;
00113 }
00114
00120 void TIMER_CMP1(TIMER_TypeDef *TIMEx, uint32_t temp)
00121 {
00122     TIMEx->CHN1 = temp;
00123 }
00124
00130 uint32_t TIMER_Timer_GetCMPT0(TIMER_TypeDef *TIMEx)
00131 {
00132     return TIMEx->CHN0;
00133 }
00134
00140 uint32_t TIMER_Timer_GetCMPT1(TIMER_TypeDef *TIMEx)
00141 {
00142     return TIMEx->CHN1;
00143 }
00144
00150 void TIMER_ClearIRQFlag(TIMER_TypeDef *TIMEx, uint32_t tempFlag)
00151 {
00152     TIMEx->IF = tempFlag;
00153 }
00154
00161 uint16_t TIMER_GetIRQFlag(TIMER_TypeDef *TIMEx, uint32_t TIMER_IF_x)
00162 {
00163     return (TIMEx->IF & TIMER_IF_x) ? 1 : 0;
00164 }
00165
00172 uint16_t TIMER_GetFlag(TIMER_TypeDef *TIMEx, uint32_t TIMER_IF_x)
00173 {
00174     return (TIMEx->IF & TIMER_IF_x) ? 1 : 0;
00175 }
00176
00182 uint32_t TIMER_Timer_GetCount(TIMER_TypeDef *TIMEx)
00183 {
00184     return TIMEx->CNT;
00185 }
00186
00191 void TIMER_Enable(TIMER_TypeDef *TIMEx)
00192 {
00193     if (TIMEx == TIMER0)
00194     {
00195         SYS.ModuleClockCmd(SYS_MODULE_TIMER0, ENABLE);
00196     }
00197     else if (TIMEx == TIMER1)
00198     {
00199         SYS.ModuleClockCmd(SYS_MODULE_TIMER1, ENABLE);
00200     }
00201     else if (TIMEx == TIMER2)
00202     {
00203         SYS.ModuleClockCmd(SYS_MODULE_TIMER2, ENABLE);
00204     }
00205     else
00206     {
00207         __NOP();
00208     }
00209 }
00210
00215 void TIMER_Disable(TIMER_TypeDef *TIMEx)
00216 {
00217     if (TIMEx == TIMER0)
00218     {
00219         SYS.ModuleClockCmd(SYS_MODULE_TIMER0, DISABLE);
00220     }
00221     else if (TIMEx == TIMER1)
00222     {
00223         SYS.ModuleClockCmd(SYS_MODULE_TIMER1, DISABLE);
00224     }
00225     else if (TIMEx == TIMER2)
00226     {
00227         SYS.ModuleClockCmd(SYS_MODULE_TIMER2, DISABLE);
00228     }
00229     else
00230     {
00231         __NOP();
00232     }
00233 }
00234
00240 void TIMER_SetOutputState(TIMER_TypeDef *TIMEx, uint8_t state)
00241 {
00242     if (state == 0)
00243     {

```

```

00244     TIMERx->IO &= ~BIT6;
00245 }
00246 else
00247 {
00248     TIMERx->IO |= BIT6;
00249 }
00250 }

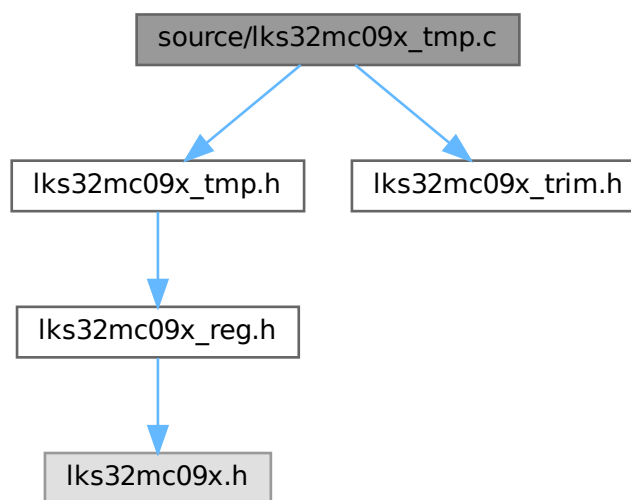
```

3.97 source/lks32mc09x_tmp.c 文件参考

```

#include "lks32mc09x_tmp.h"
#include "lks32mc09x_trim.h"
lks32mc09x_tmp.c 的引用(Include)关系图:

```



函数

- void **TMP_Init** (void)
初始化温度传感器
- void **TMP_Enable** (void)
使能温度传感器
- void **TMP_Disable** (void)
禁用温度传感器
- void **TMP_Reset** (void)
复位温度传感器
- s16 **TMP_GetCurrentTemperature** (s16 ADC_value)
获取当前温度

变量

- static int16_t **tmp_k** = 0
- static int16_t **tmp_b** = 0

3.97.1 函数说明

3.97.1.1 TMP_Disable()

```
void TMP_Disable (
    void )
```

禁用温度传感器

在文件 [lks32mc09x.tmp.c](#) 第 29 行定义.

这是这个函数的调用关系图:



3.97.1.2 TMP_Enable()

```
void TMP_Enable (
    void )
```

使能温度传感器

在文件 [lks32mc09x.tmp.c](#) 第 19 行定义.

这是这个函数的调用关系图:



3.97.1.3 TMP_GetCurrentTemperature()

```
s16 TMP_GetCurrentTemperature (
    s16 ADC_value )
```

获取当前温度

参数

<i>ADC_value</i>	ADC采样值

返回

当前温度值，单位为摄氏度

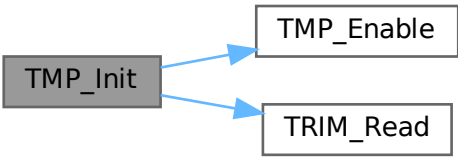
在文件 [lks32mc09x.tmp.c](#) 第 49 行定义.

3.97.1.4 TMP_Init()

```
void TMP_Init (
    void )
```

初始化温度传感器
初始化温度传感器(读取温度传感器校准参数&使能温度传感器)
在文件 [lks32mc09x.tmp.c](#) 第 9 行定义.

函数调用图:

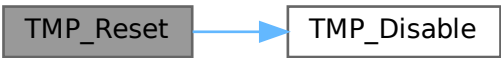


3.97.1.5 TMP_Reset()

```
void TMP_Reset (
    void )
```

复位温度传感器
复位温度传感器(禁用温度传感器)
在文件 [lks32mc09x.tmp.c](#) 第 39 行定义.

函数调用图:



3.97.2 变量说明

3.97.2.1 tmp_b

```
int16_t tmp_b = 0 [static]
```

在文件 [lks32mc09x_tmp.c](#) 第 5 行定义.

3.97.2.2 tmp_k

```
int16_t tmp_k = 0 [static]
```

在文件 [lks32mc09x_tmp.c](#) 第 4 行定义.

3.98 lks32mc09x_tmp.c

[浏览该文件的文档.](#)

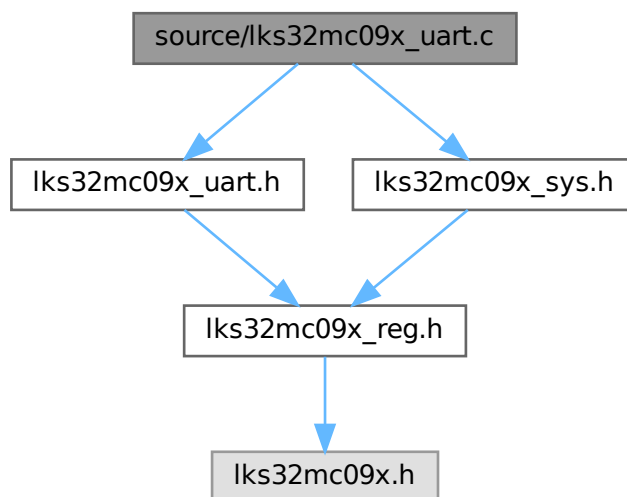
```
00001 #include "lks32mc09x_tmp.h"
00002 #include "lks32mc09x_trim.h"
00003
00004 static int16_t tmp_k = 0;
00005 static int16_t tmp_b = 0;
00009 void TMP_Init(void)
00010 {
00011     tmp_k = TRIM_Read(NVR_ADDR_TMP_K);
00012     tmp_b = TRIM_Read(NVR_ADDR_TMP_B);
00013     TMP_Enable();
00014 }
00015
00019 void TMP_Enable(void)
00020 {
00021     SYS_WR_PROTECT = 0x7a83;
00022     SYS_AFE_REG5 |= BIT3;
00023     SYS_WR_PROTECT = 0;
00024 }
00025
00029 void TMP_Disable(void)
00030 {
00031     SYS_WR_PROTECT = 0x7a83;
00032     SYS_AFE_REG5 &= ~BIT3;
00033     SYS_WR_PROTECT = 0;
00034 }
00035
00039 void TMP_Reset(void)
00040 {
00041     TMP_Disable();
00042 }
00043
00049 s16 TMP_GetCurrentTemperature(s16 ADC_value)
00050 {
00051     s16 t_Temperature;
00052     if (ADC1->CFG & BIT10)
00053     {
00054         t_Temperature = (tmp_b - ((s32)tmp_k * ADC_value) / 1000);
00055     }
00056     else
00057     {
00058         t_Temperature = (tmp_b - ((s32)tmp_k * (ADC_value >> 4)) / 1000);
00059     }
00060
00061     return t_Temperature;
00062 }
```

3.99 source/lks32mc09x_uart.c 文件参考

```
#include "lks32mc09x_uart.h"
```

```
#include "lks32mc09x_sys.h"
```

lks32mc09x_uart.c 的引用(Include)关系图:



函数

- void **UART_Init** (UART_TypeDef *UARTx, UART_InitTypeDef *UART_InitStruct)
初始化UART模块
- void **UART_StructInit** (UART_InitTypeDef *UART_InitStruct)
初始化UART结构体为默认值
- void **UART_SendData** (UART_TypeDef *UARTx, uint32_t n)
发送数据
- uint32_t **UART_ReadData** (UART_TypeDef *UARTx)
读取接收到的数据
- void **UART_SendnData** (UART_TypeDef *UARTx, char *sendData, uint32_t len)
发送指定长度的数据
- void **UART_RecvnData** (UART_TypeDef *UARTx, char *recvData, uint32_t maxlen)
接收指定长度的数据
- void **UART_ClearIRQFlag** (UART_TypeDef *UARTx, uint32_t tempFlag)
清除指定的中断标志
- uint32_t **UART_GetIRQFlag** (UART_TypeDef *UARTx, uint32_t tempFlag)
获取指定的中断标志状态
- uint8_t **UART_SendAddr_485** (UART_TypeDef *UARTx, uint8_t n)
在485模式下发送地址
- uint8_t **UART_SendData_485** (UART_TypeDef *UARTx, uint8_t n)
在485模式下发送数据
- uint32_t **UART_SetBaud** (UART_TypeDef *UARTx, uint32_t Baud)

- 设置波特率
- `uint32_t UART_ReadBaud (UART_TypeDef *UARTx)`
 - `void UART_SetClkDiv (uint32_t div)`
- 设置uart时钟分频

3.99.1 函数说明

3.99.1.1 UART_ClearIRQFlag()

```
void UART_ClearIRQFlag (  
    UART_TypeDef * UARTx,  
    uint32_t tempFlag )
```

清除指定的中断标志

参数

<i>UARTx</i>	UART模块指针
<i>tempFlag</i>	要清除的中断标志

在文件 `lks32mc09x_uart.c` 第 134 行定义.

3.99.1.2 UART_GetIRQFlag()

```
uint32_t UART_GetIRQFlag (  
    UART_TypeDef * UARTx,  
    uint32_t tempFlag )
```

获取指定的中断标志状态

参数

<i>UARTx</i>	UART模块指针
<i>tempFlag</i>	要获取的中断标志

返回

中断标志状态

在文件 `lks32mc09x_uart.c` 第 145 行定义.

3.99.1.3 UART_Init()

```
void UART_Init (  
    UART_TypeDef * UARTx,  
    UART_InitTypeDef * UART_InitStruct )
```

初始化UART模块

函数声明

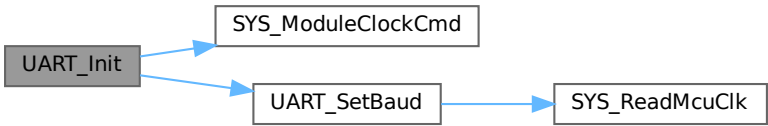
参数

<i>UARTx</i>	UART模块指针
<i>UART_InitStruct</i>	初始化结构体指针

- < LIN同步段检测长度 0:10bits 1:11bits
- < LIN波特率自适应使能

在文件 lks32mc09x_uart.c 第 9 行定义.

函数调用图:

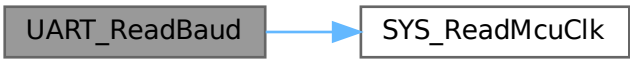


3.99.1.4 UART_ReadBaud()

```
uint32_t UART_ReadBaud (
    UART_TypeDef * UARTx )
```

在文件 lks32mc09x_uart.c 第 198 行定义.

函数调用图:



3.99.1.5 UART_ReadData()

```
uint32_t UART_ReadData (
    UART_TypeDef * UARTx )
```

读取接收到的数据

参数

<i>UARTx</i>	UART模块指针

返回

接收到的数据

在文件 [lks32mc09x_uart.c](#) 第 92 行定义.

这是这个函数的调用关系图:



3.99.1.6 UART_RecvnData()

```
void UART_RecvnData (
    UART_TypeDef * UARTx,
    char * recvData,
    uint32_t maxlen )
```

接收指定长度的数据

参数

<i>UARTx</i>	UART模块指针
<i>recvData</i>	接收数据缓冲区指针
<i>maxlen</i>	最大接收长度

在文件 [lks32mc09x_uart.c](#) 第 119 行定义.

函数调用图:



3.99.1.7 UART_SendAddr_485()

```
uint8_t UART_SendAddr_485 (
    UART_TypeDef * UARTx,
    uint8_t n )
```

在485模式下发送地址

参数

UARTx	UART模块指针
n	要发送的地址

返回

发送状态（简单实现，总是返回成功）

在文件 lks32mc09x_uart.c 第 156 行定义.

函数调用图:



3.99.1.8 UART_SendData()

```
void UART_SendData (
    UART_TypeDef * UARTx,
    uint32_t n )
```

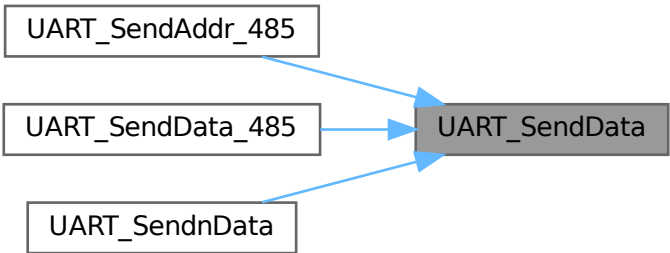
发送数据

参数

UARTx	UART模块指针
n	要发送的数据

在文件 lks32mc09x_uart.c 第 78 行定义.

这是这个函数的调用关系图:



3.99.1.9 UART_SendData_485()

```
uint8_t UART_SendData_485 (
    UART_TypeDef * UARTx,
    uint8_t n )
```

在485模式下发送数据

参数

<i>UARTx</i>	UART模块指针
<i>n</i>	要发送的数据

返回

发送状态（简单实现，总是返回成功）

在文件 [lks32mc09x_uart.c](#) 第 168 行定义.

函数调用图:



3.99.1.10 UART_SendnData()

```
void UART_SendnData (
    UART_TypeDef * UARTx,
    char * sendData,
    uint32_t len )
```

发送指定长度的数据

参数

<i>UARTx</i>	UART模块指针
<i>sendData</i>	要发送的数据指针
<i>len</i>	数据长度

在文件 [lks32mc09x_uart.c](#) 第 103 行定义.

函数调用图:



3.99.1.11 UART_SetBaud()

```
uint32_t UART_SetBaud (
    UART_TypeDef * UARTx,
    uint32_t Baud )
```

设置波特率

参数

<i>UARTx</i>	UART模块
<i>Baud</i>	波特率范围

返回

实际波特率（不考虑时钟误差）

在文件 [lks32mc09x_uart.c](#) 第 180 行定义.

函数调用图:



这是这个函数的调用关系图:



3.99.1.12 UART_SetClkDiv()

```
void UART_SetClkDiv (
    uint32_t div )
```

设置uart时钟分频

参数

<i>div</i>	分频系数，0对应不分频，1对应2分频

在文件 [lks32mc09x_uart.c](#) 第 209 行定义.

函数调用图:



3.99.1.13 UART_StructInit()

```
void UART_StructInit (
    UART_InitTypeDef * UART_InitStruct )
```

初始化UART结构体为默认值

参数

<i>UART_InitStruct</i>	初始化结构体指针

在文件 [lks32mc09x_uart.c](#) 第 56 行定义.

3.100 lks32mc09x_uart.c

[浏览该文件的文档.](#)

```
00001 #include "lks32mc09x_uart.h"
00002 #include "lks32mc09x_sys.h"
00003
00009 void UART_Init(UART_TypeDef *UARTx, UART_InitTypeDef *UART_InitStruct)
00010 {
00011     // 使能UART模块时钟
00012     if (UARTx == UART0)
00013     {
00014         SYS.ModuleClockCmd(SYS_MODULE_UART0, ENABLE);
00015     }
00016     else if (UARTx == UART1)
00017     {
00018         SYS.ModuleClockCmd(SYS_MODULE_UART1, ENABLE);
00019     }
00020     else if (UARTx == UART2)
00021     {
00022         SYS.ModuleClockCmd(SYS_MODULE_UART2, ENABLE);
00023     }
00024
00025     UART_SetBaud(UARTx, UART_InitStruct->BaudRate);
00026     // 配置UART控制寄存器
00027     UARTx->CTRL = (UART_InitStruct->DUPLEX << 7) |
00028         (UART_InitStruct->LIN_EN << 6) |
00029         (UART_InitStruct->MultiDropEna << 5) |
00030         (UART_InitStruct->CK_EN << 4) |
00031         (UART_InitStruct->ParityMode << 3) |
00032         (UART_InitStruct->FirstSend << 2) |
00033         (UART_InitStruct->StopBits << 1) |
00034         (UART_InitStruct->WordLength);
00035
00036     // 配置中断使能寄存器
00037     UARTx->IE = UART_InitStruct->IRQEna;
00038
00039     // 配置DMA请求使能寄存器
00040     UARTx->RE = UART_InitStruct->DMARE;
00041
00042     // 配置IO极性
00043     UARTx->IOC =
00044         (UART_InitStruct->LBDL << 6) |
00045         (UART_InitStruct->ABD_EN << 4) |
00046         (UART_InitStruct->TXD_INV << 1) |
00047         (UART_InitStruct->RXD_INV) |
00048         0;
00049     UARTx->IF = 0x3ff;
00050 }
00051
00056 void UART_StructInit(UART_InitTypeDef *UART_InitStruct)
00057 {
00058     uint32_t i;
00059     for (i = 0; i < sizeof(UART_InitTypeDef); i++)
00060     {
00061         ((uint8_t *)UART_InitStruct)[i] = 0;
00062     }
00063
00064     // 默认配置
00065     UART_InitStruct->WordLength = UART_WORDLENGTH_8b;
```

```

00066     UART_InitStructure->StopBits      = UART_STOPBITS_1b;
00067     UART_InitStructure->FirstSend       = UART_FIRSTSEND_LSB;
00068     UART_InitStructure->ParityMode      = UART_Parity_NO;
00069     UART_InitStructure->MultiDropEna    = DISABLE;
00070     UART_InitStructure->DUPLEX          = 0; // 全双工
00071 }
00072
00078 void UART_SendData(UART_TypeDef *UARTx, uint32_t n)
00079 {
00080     UARTx->BUFF = n & 0xFF;
00081     while ((UARTx->STT & BIT0) == 0)
00082     {
00083         // 等待发送完成
00084     }
00085 }
00086
00092 uint32_t UART_ReadData(UART_TypeDef *UARTx)
00093 {
00094     return UARTx->BUFF;
00095 }
00096
00103 void UART_SendnData(UART_TypeDef *UARTx, char *sendData, uint32_t len)
00104 {
00105     for (uint32_t i = 0; i < len; i++)
00106     {
00107         UART_SendData(UARTx, sendData[i]);
00108         // 等待发送完成 (简单实现, 实际可能需要更复杂的逻辑)
00109         while (!(UARTx->STT & 0x01));
00110     }
00111 }
00112
00119 void UART_RecvnData(UART_TypeDef *UARTx, char *recvData, uint32_t maxlen)
00120 {
00121     for (uint32_t i = 0; i < maxlen; i++)
00122     {
00123         // 等待接收完成 (简单实现, 实际可能需要更复杂的逻辑)
00124         while (!(UARTx->STT & 0x02));
00125         recvData[i] = UART_ReadData(UARTx);
00126     }
00127 }
00128
00134 void UART_ClearIRQFlag(UART_TypeDef *UARTx, uint32_t tempFlag)
00135 {
00136     UARTx->IF = tempFlag;
00137 }
00138
00145 uint32_t UART_GetIRQFlag(UART_TypeDef *UARTx, uint32_t tempFlag)
00146 {
00147     return UARTx->IF & tempFlag;
00148 }
00149
00156 uint8_t UART_SendAddr_485(UART_TypeDef *UARTx, uint8_t n)
00157 {
00158     UART_SendData(UARTx, n);
00159     return 1; // 简单实现, 总是返回成功
00160 }
00161
00168 uint8_t UART_SendData_485(UART_TypeDef *UARTx, uint8_t n)
00169 {
00170     UART_SendData(UARTx, n);
00171     return 1; // 简单实现, 总是返回成功
00172 }
00173
00180 uint32_t UART_SetBaud(UART_TypeDef *UARTx, uint32_t Baud)
00181 {
00182     uint32_t uartclk = SYS_ReadMcuClk() / (SYS->CLK_DIV2 + 1); // 串口的时钟频率
00183     uint32_t div     = uartclk / Baud - 1;
00184     if (div > 0xffff)
00185     {
00186         div = 0xffff;
00187     }
00188     else
00189     {
00190         __NOP();
00191     }
00192     UARTx->DIVL = div & 0xFF;
00193     UARTx->DIVH = (div & 0xFF00) >> 8;
00194     Baud        = uartclk / (div + 1);
00195     return Baud;
00196 }
00197
00198 uint32_t UART_ReadBaud(UART_TypeDef *UARTx)
00199 {
00200     uint32_t baud;
00201     baud = SYS_ReadMcuClk() / (SYS->CLK_DIV2 + 1) / (1 + 256 * UARTx->DIVH + UARTx->DIVL);
00202     return baud;
00203 }

```

```

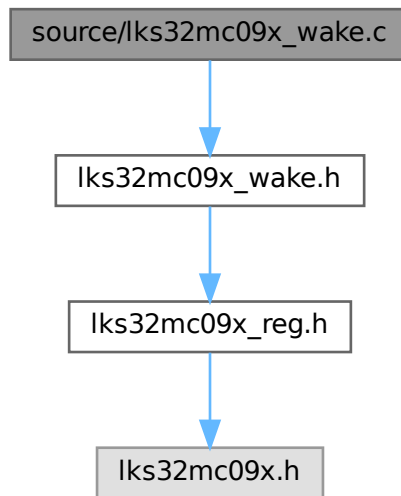
00204
00209 void UART_SetClkDiv(uint32_t div)
00210 {
00211     SYS_UartClkDiv(div);
00212 }

```

3.101 source/lks32mc09x_wake.c 文件参考

```
#include "lks32mc09x_wake.h"
```

lks32mc09x_wake.c 的引用(Include)关系图:



函数

- void [WAKE_EVTCLRSleep](#) (void)
清除休眠事件记录
- void [WAKE_FallSleep](#) (void)
进入低功耗模式，必须定义 `WAKE_IRQHandler` 函数，`WAKE_IRQHandler` 内可以不做任何操作
- void [WAKE_SetWakeInterval](#) (uint32_t rth, uint32_t wth, uint8_t ena)
设置唤醒间隔时间
- void [WAKE_SetWakeIO](#) (uint32_t WAKE_IO_x, [WAKE_TRIG_x](#) trigLevel, uint32_t flt, uint8_t ena)
设置唤醒IO

3.101.1 函数说明

3.101.1.1 WAKE_EVTCLRSleep()

```
void WAKE_EVTCLRSleep (
    void )
```

清除休眠事件记录

在文件 `lks32mc09x_wake.c` 第 6 行定义.

3.101.1.2 WAKE_FallSleep()

```
void WAKE_FallSleep (
    void )
```

进入低功耗模式，必须定义WAKE_IRQHandler函数，WAKE_IRQHandler内可以不做任何操作

进入低功耗模式

在文件 [lks32mc09x_wake.c](#) 第 15 行定义.

3.101.1.3 WAKE_SetWakeInterval()

```
void WAKE_SetWakeInterval (
    uint32_t rth,
    uint32_t wth,
    uint8_t ena )
```

设置唤醒间隔时间

参数

<i>rth</i>	唤醒复位阈值，由于计数器计数阈值是从复位阈值开始计数的，所有唤醒阈值必须大于复位阈值
<i>wth</i>	唤醒阈值
<i>ena</i>	使能标志

在文件 [lks32mc09x_wake.c](#) 第 106 行定义.

3.101.1.4 WAKE_SetWakeIO()

```
void WAKE_SetWakeIO (
    uint32_t WAKE_IO_x,
    WAKE_TRIG_x trigLevel,
    uint32_t flt,
    uint8_t ena )
```

设置唤醒IO

参数

<i>wakeIO</i>	唤醒IO选择，必须打开对应的gpio输入使能（GPIO设置为输入模式），可以复用成任何模式
<i>trigLevel</i>	触发电平
<i>flt</i>	滤波使能
<i>ena</i>	使能标志

在文件 [lks32mc09x_wake.c](#) 第 138 行定义.

3.102 lks32mc09x_wake.c

浏览该文件的文档。

```

00001 #include "lks32mc09x_wake.h"
00002
00006 void WAKE_EVTCLRSleep(void)
00007 {
00008     // 清除深度休眠记录
00009     AON0->EVT_RCD &= ~BIT13;
00010 }
00011
00015 void WAKE_FallSleep(void)
00016 {
00017     int irqstate;
00018     uint16_t sys_reg_0;
00019     uint16_t sys_reg_1;
00020     uint16_t sys_reg_2;
00021     uint16_t sys_reg_3;
00022     uint16_t sys_reg_4;
00023     uint16_t sys_reg_5;
00024     uint16_t sys_reg_7;
00025     uint32_t sys_clk_fen;
00026     uint32_t sys_clk_cfg;
00027     uint32_t nvic_state; // 各个使能的状态
00028     irqstate = _get_PRIMASK(); // 保存全局中断状态
00029     _disable_irq(); // 关闭全局中断
00030
00031     SYS_WR_PROTECT = 0x7a83; /*开启寄存器写使能*/
00032     sys_reg_0 = SYS_AFE_REG0;
00033     sys_reg_1 = SYS_AFE_REG1;
00034     sys_reg_2 = SYS_AFE_REG2;
00035     sys_reg_3 = SYS_AFE_REG3;
00036     sys_reg_4 = SYS_AFE_REG4;
00037     sys_reg_5 = SYS_AFE_REG5;
00038     sys_reg_7 = SYS_AFE_REG7;
00039     sys_clk_fen = SYS_CLK_FEN;
00040     sys_clk_cfg = SYS_CLK_CFG;
00041     SYS_AFE_REG0 = 0;
00042     SYS_AFE_REG1 = BIT11 | BIT10; // 关闭BGP和RCH
00043     SYS_AFE_REG2 = 0;
00044     SYS_AFE_REG3 = 0;
00045     SYS_AFE_REG4 = 0;
00046     SYS_AFE_REG5 = (sys_reg_5 & (BIT8 | BIT9)); // 保持复位电压阈值不变
00047     SYS_AFE_REG7 = 0;
00048     SYS_CLK_FEN = 0;
00049     SYS_CLK_CFG = 0x0;
00050     {
00051         int cnt;
00052         for (cnt = 0; cnt < 4000; cnt++)
00053         {
00054             __NOP();
00055         }
00056     }
00057     SYS_WR_PROTECT = 0x0;
00058
00059     nvic_state = NVIC->ICER[0];
00060     NVIC->ICER[0] = 0xffffffff; // 关闭所有中断
00061     NVIC_EnableIRQ(WAKE_IRQn); /* 使能WAKE中断 */
00062     NVIC_SetPriority(WAKE_IRQn, 0); /* 配置WAKE中断优先级 */
00063     IWDG_PSW = 0xA6B4;
00064     IWDG_CLR = 0x798D;
00065     SCB->SCR |= (1UL << 2);
00066     __enable_irq();
00067     __WFI(); // 使用WFI指令进入休眠模式
00068
00069     // 休眠后程序会停在这里，唤醒后会执行WAKE_IRQHandler中断，随后从这里继续执行
00070     _disable_irq();
00071     SYS_WR_PROTECT = 0x7a83; /*开启寄存器写使能*/
00072     SYS_AFE_REG0 = sys_reg_0;
00073     SYS_AFE_REG1 = sys_reg_1;
00074     SYS_AFE_REG2 = sys_reg_2;
00075     SYS_AFE_REG3 = sys_reg_3;
00076     SYS_AFE_REG4 = sys_reg_4;
00077     SYS_AFE_REG5 = sys_reg_5;
00078     SYS_AFE_REG7 = sys_reg_7;
00079     SYS_CLK_FEN = sys_clk_fen;
00080     {
00081         int cnt;
00082         for (cnt = 0; cnt < 4000; cnt++)
00083         {
00084             __NOP();
00085         }
00086     }
00087     SYS_CLK_CFG = sys_clk_cfg;
00088     {

```



```

00089         int cnt;
00090         for (cnt = 0; cnt < 4000; cnt++)
00091         {
00092             __NOP();
00093         }
00094     }
00095     SYS_WR_PROTECT = 0x0;
00096     __set_PRIMASK(irqstate); // 还原中断状态
00097     NVIC->ISER[0] = nvic_state;
00098 }
00099
00106 void WAKE_SetWakeInterval(uint32_t rth, uint32_t wth, uint8_t ena)
00107 {
00108     if (ena)
00109     {
00110         int32_t val;
00111         IWDG0->CFG |= BIT4;
00112         IWDG0->PSW = PSW_IWDG_PRE;
00113         IWDG0->RTH = rth; // 设置复位阈值
00114         val = rth - wth;
00115         if (val < 0x100)
00116         {
00117             val = 0x100;
00118         }
00119         else
00120         {
00121             __NOP();
00122         }
00123         IWDG0->WTH = val; // 设置唤醒阈值
00124     }
00125     else
00126     {
00127         IWDG0->CFG &= ~BIT4;
00128     }
00129 }
00130
00138 void WAKE_SetWakeIO(uint32_t WAKE_IO_x, WAKE_TRIG_x trigLevel, uint32_t flt, uint8_t ena)
00139 {
00140     if (ena)
00141     {
00142         AON_IO_WAKE_EN |= WAKE_IO_x;
00143         // 使能IO唤醒
00144         AON0->IO_WAKE_EN |= WAKE_IO_x;
00145
00146         // 设置触发电平
00147         if (trigLevel == WAKE_TRIG_HIGH)
00148         {
00149             AON0->IO_WAKE_POL |= WAKE_IO_x;
00150         }
00151         else
00152         {
00153             AON0->IO_WAKE_POL &= ~WAKE_IO_x;
00154         }
00155
00156         // 设置滤波使能
00157         if (flt == WAKE_FLT_EN)
00158         {
00159             AON0->PWR_CFG |= BIT1;
00160         }
00161         else
00162         {
00163             AON0->PWR_CFG &= ~BIT1;
00164         }
00165     }
00166     else
00167     {
00168         // 禁用IO唤醒
00169         AON0->IO_WAKE_EN &= ~WAKE_IO_x;
00170     }
00171 }

```


Index

ABD_EN
 UART_InitTypeDef, 120
ADC
 lks32mc09x_reg.h, 436
ADC0
 lks32mc09x_reg.h, 436
ADC0_CHN_ADC01_CH10
 lks32mc09x_adc.h, 139
ADC0_CHN_ADC01_CH11
 lks32mc09x_adc.h, 139
ADC0_CHN_ADC01_CH4
 lks32mc09x_adc.h, 139
ADC0_CHN_ADC01_CH5
 lks32mc09x_adc.h, 139
ADC0_CHN_ADC01_CH6
 lks32mc09x_adc.h, 139
ADC0_CHN_ADC01_CH7
 lks32mc09x_adc.h, 139
ADC0_CHN_ADC01_CH12
 lks32mc09x_adc.h, 139
ADC0_CHN_ADC01_CH13
 lks32mc09x_adc.h, 139
ADC0_CHN_ADC01_CH8
 lks32mc09x_adc.h, 139
ADC0_CHN_AVSS
 lks32mc09x_adc.h, 139
ADC0_CHN_DAC
 lks32mc09x_adc.h, 139
ADC0_CHN_OPA0
 lks32mc09x_adc.h, 139
ADC0_CHN_OPA1
 lks32mc09x_adc.h, 139
ADC0_CHN_OPA2
 lks32mc09x_adc.h, 139
ADC0_CHN_OPA3
 lks32mc09x_adc.h, 139
ADC0_CHN_REF24
 lks32mc09x_adc.h, 139
ADC1
 lks32mc09x_reg.h, 436
ADC1_CHN_ADC01_CH10
 lks32mc09x_adc.h, 139
ADC1_CHN_ADC01_CH11
 lks32mc09x_adc.h, 139
ADC1_CHN_ADC01_CH4
 lks32mc09x_adc.h, 139
ADC1_CHN_ADC01_CH5
 lks32mc09x_adc.h, 139
ADC1_CHN_ADC01_CH6
 lks32mc09x_adc.h, 139
ADC1_CHN_ADC01_CH7
 lks32mc09x_adc.h, 139
ADC1_CHN_ADC01_CH12
 lks32mc09x_adc.h, 140
ADC1_CHN_ADC01_CH13
 lks32mc09x_adc.h, 140
ADC1_CHN_ADC01_CH8
 lks32mc09x_adc.h, 139
ADC1_CHN_ADC01_CH9
 lks32mc09x_adc.h, 139
ADC1_CHN_AVDD
 lks32mc09x_adc.h, 140
ADC1_CHN_OPA0
 lks32mc09x_adc.h, 139
ADC1_CHN_OPA1
 lks32mc09x_adc.h, 139
ADC1_CHN_OPA2
 lks32mc09x_adc.h, 139
ADC1_CHN_OPA3
 lks32mc09x_adc.h, 139
ADC1_CHN_TMP
 lks32mc09x_adc.h, 140
ADC_CHN_0
 lks32mc09x_adc.h, 139
ADC_CHN_1
 lks32mc09x_adc.h, 139
ADC_CHN_10
 lks32mc09x_adc.h, 139
ADC_CHN_11
 lks32mc09x_adc.h, 139
ADC_CHN_12
 lks32mc09x_adc.h, 139
ADC_CHN_13
 lks32mc09x_adc.h, 139
ADC_CHN_14
 lks32mc09x_adc.h, 139
ADC_CHN_15
 lks32mc09x_adc.h, 139
ADC_CHN_2
 lks32mc09x_adc.h, 139
ADC_CHN_3
 lks32mc09x_adc.h, 139
ADC_CHN_4
 lks32mc09x_adc.h, 139
ADC_CHN_5
 lks32mc09x_adc.h, 139
ADC_CHN_6
 lks32mc09x_adc.h, 139

ADC_CHN_7
 lks32mc09x_adc.h, 139
 ADC_CHN_8
 lks32mc09x_adc.h, 139
 ADC_CHN_9
 lks32mc09x_adc.h, 139
 ADC_CHN_x
 lks32mc09x_adc.h, 138
 ADC_ClearIRQFlag
 lks32mc09x_adc.c, 534
 lks32mc09x_adc.h, 141
 ADC_DAT_0
 lks32mc09x_adc.h, 140
 ADC_DAT_1
 lks32mc09x_adc.h, 140
 ADC_DAT_10
 lks32mc09x_adc.h, 140
 ADC_DAT_11
 lks32mc09x_adc.h, 140
 ADC_DAT_2
 lks32mc09x_adc.h, 140
 ADC_DAT_3
 lks32mc09x_adc.h, 140
 ADC_DAT_4
 lks32mc09x_adc.h, 140
 ADC_DAT_5
 lks32mc09x_adc.h, 140
 ADC_DAT_6
 lks32mc09x_adc.h, 140
 ADC_DAT_7
 lks32mc09x_adc.h, 140
 ADC_DAT_8
 lks32mc09x_adc.h, 140
 ADC_DAT_9
 lks32mc09x_adc.h, 140
 ADC_DAT_x
 lks32mc09x_adc.h, 140
 ADC_Disable
 lks32mc09x_adc.c, 534
 lks32mc09x_adc.h, 141
 ADC_Enable
 lks32mc09x_adc.c, 534
 lks32mc09x_adc.h, 141
 ADC_GAIN_3V6
 lks32mc09x_adc.h, 140
 ADC_GAIN_7V2
 lks32mc09x_adc.h, 140
 ADC_GAIN_x
 lks32mc09x_adc.h, 140
 ADC_GetConversionValue
 lks32mc09x_adc.c, 535
 lks32mc09x_adc.h, 142
 ADC_GetConversionValueS32
 lks32mc09x_adc.c, 535
 lks32mc09x_adc.h, 142
 ADC_GetIRQFlag
 lks32mc09x_adc.c, 536
 lks32mc09x_adc.h, 143
 ADC_IE_AWD0
 lks32mc09x_adc.h, 131
 ADC_IE_HERR
 lks32mc09x_adc.h, 131
 ADC_IE_ISF
 lks32mc09x_adc.h, 131
 ADC_IE_SERR
 lks32mc09x_adc.h, 131
 ADC_IE_SF1
 lks32mc09x_adc.h, 132
 ADC_IE_SF2
 lks32mc09x_adc.h, 132
 ADC_IF_ALL
 lks32mc09x_adc.h, 132
 ADC_IF_AWD0
 lks32mc09x_adc.h, 132
 ADC_IF_HERR
 lks32mc09x_adc.h, 132
 ADC_IF_ISF
 lks32mc09x_adc.h, 132
 ADC_IF_SERR
 lks32mc09x_adc.h, 133
 ADC_IF_SF1
 lks32mc09x_adc.h, 133
 ADC_IF_SF2
 lks32mc09x_adc.h, 133
 ADC_Init
 lks32mc09x_adc.c, 536
 lks32mc09x_adc.h, 144
 ADC_InitTypeDef, 5
 CSMP, 6
 DATA_ALIGN, 6
 GAIN, 6
 GEN, 6
 HTH, 6
 IDLE_PRI, 7
 IE, 7
 IS1, 7
 LTH, 7
 NSMP, 7
 OVSr, 7
 RE, 8
 S1, 8
 S2, 8
 TCNT, 8
 TRIG, 8
 TROVS, 8
 ADC_RANGE_3_3
 lks32mc09x_adc.h, 141
 ADC_RANGE_3_6
 lks32mc09x_adc.h, 141
 ADC_RANGE_EXTERNAL
 lks32mc09x_adc.h, 141
 ADC_RANGE_x
 lks32mc09x_adc.h, 140
 ADC_RE_AWD0
 lks32mc09x_adc.h, 133
 ADC_RE_HERR

lks32mc09x_adc.h, 133

ADC_RE_ISF
lks32mc09x_adc.h, 133

ADC_RE_SERR
lks32mc09x_adc.h, 134

ADC_RE_SF1
lks32mc09x_adc.h, 134

ADC_RE_SF2
lks32mc09x_adc.h, 134

ADC_Reset
lks32mc09x_adc.c, 537
lks32mc09x_adc.h, 144

ADC_SetChn
lks32mc09x_adc.c, 537
lks32mc09x_adc.h, 145

ADC_SetGain
lks32mc09x_adc.c, 538
lks32mc09x_adc.h, 145

ADC_SetRange
lks32mc09x_adc.c, 538
lks32mc09x_adc.h, 146

ADC_SoftIdleTrig
lks32mc09x_adc.c, 539
lks32mc09x_adc.h, 146

ADC_SoftTrig
lks32mc09x_adc.c, 539
lks32mc09x_adc.h, 146

ADC_StructInit
lks32mc09x_adc.c, 539
lks32mc09x_adc.h, 147

ADC_TRIG_CL_IT0
lks32mc09x_adc.h, 134

ADC_TRIG_CL_IT1
lks32mc09x_adc.h, 134

ADC_TRIG_CL_IT2
lks32mc09x_adc.h, 134

ADC_TRIG_CL_IT3
lks32mc09x_adc.h, 135

ADC_TRIG_CL_NT0
lks32mc09x_adc.h, 135

ADC_TRIG_CL_NT1
lks32mc09x_adc.h, 135

ADC_TRIG_CL_NT2
lks32mc09x_adc.h, 135

ADC_TRIG_CL_NT3
lks32mc09x_adc.h, 135

ADC_TRIG_MCPWM0_T0
lks32mc09x_adc.h, 135

ADC_TRIG_MCPWM0_T0_IEN
lks32mc09x_adc.h, 136

ADC_TRIG_MCPWM0_T1
lks32mc09x_adc.h, 136

ADC_TRIG_MCPWM0_T1_IEN
lks32mc09x_adc.h, 136

ADC_TRIG_MCPWM0_T2
lks32mc09x_adc.h, 136

ADC_TRIG_MCPWM0_T2_IEN
lks32mc09x_adc.h, 136

ADC_TRIG_MCPWM0_T3
lks32mc09x_adc.h, 136

ADC_TRIG_MCPWM0_T3_IEN
lks32mc09x_adc.h, 137

ADC_TRIG_SOFT
lks32mc09x_adc.h, 137

ADC_TRIG_TIMER0_CMP0
lks32mc09x_adc.h, 137

ADC_TRIG_TIMER0_CMP0_IEN
lks32mc09x_adc.h, 137

ADC_TRIG_TIMER0_CMP1
lks32mc09x_adc.h, 137

ADC_TRIG_TIMER0_CMP1_IEN
lks32mc09x_adc.h, 137

ADC_TRIG_TIMER1_CMP0
lks32mc09x_adc.h, 138

ADC_TRIG_TIMER1_CMP1
lks32mc09x_adc.h, 138

ADC_TRIG_TIMER2_CMP0_IEN
lks32mc09x_adc.h, 138

ADC_TRIG_TIMER2_CMP1_IEN
lks32mc09x_adc.h, 138

ADC_TypeDef, 9

AMC0, 11

AMC1, 11

CFG, 11

CHN0, 11

CHN1, 11

CHN2, 11

CHNT, 12

DAT0, 12

DAT1, 12

DAT10, 12

DAT11, 12

DAT2, 12

DAT3, 13

DAT4, 13

DAT5, 13

DAT6, 13

DAT7, 13

DAT8, 13

DAT9, 14

DC0, 14

DC1, 14

GAIN, 14

GEN, 14

HTH, 14

IE, 15

IF, 15

LTH, 15

RESERVED, 15

RESERVED1, 15

RESERVED2, 15

RESERVED3, 16

SWT, 16

TRIG, 16

ADDR

FLASH_TypeDef, 47

I2C_InitTypeDef, 61
 I2C_TypeDef, 64
 addr
 EEPROM_InitTypeDef, 43
 ADDR_LEN_x
 EEPROM_InitTypeDef, 43
 ADR
 UART_TypeDef, 124
 ADRCMP
 I2C_InitTypeDef, 61
 AF0_GPIO
 lks32mc09x_gpio.h, 323
 AF10_CAN
 lks32mc09x_gpio.h, 323
 AF11_SIF
 lks32mc09x_gpio.h, 323
 AF12_CL
 lks32mc09x_gpio.h, 324
 AF1_SYSAF
 lks32mc09x_gpio.h, 324
 AF2_HALL
 lks32mc09x_gpio.h, 324
 AF3_MCPWM
 lks32mc09x_gpio.h, 324
 AF4_UART
 lks32mc09x_gpio.h, 324
 AF5_SPI
 lks32mc09x_gpio.h, 324
 AF6_I2C
 lks32mc09x_gpio.h, 325
 AF7_TIMER01
 lks32mc09x_gpio.h, 325
 AF8_TIMER23
 lks32mc09x_gpio.h, 325
 AF9_DEBUG
 lks32mc09x_gpio.h, 325
 AFE_ADC
 SYS_TypeDef, 101
 AFE_DAC0
 SYS_TypeDef, 101
 AFE_DAC0_AMC
 SYS_TypeDef, 101
 AFE_DAC0_DC
 SYS_TypeDef, 101
 AFE_DAC1
 SYS_TypeDef, 101
 AFE_DAC1_AMC
 SYS_TypeDef, 101
 AFE_DAC1_DC
 SYS_TypeDef, 101
 AFE_DAC_CTRL
 SYS_TypeDef, 102
 AFE_DBG
 SYS_TypeDef, 102
 AFE_INFO
 SYS_TypeDef, 102
 AFE_REG0
 SYS_TypeDef, 102
 AFE_REG1
 SYS_TypeDef, 102
 AFE_REG2
 SYS_TypeDef, 102
 AFE_REG3
 SYS_TypeDef, 103
 AFE_REG4
 SYS_TypeDef, 103
 AFE_REG5
 SYS_TypeDef, 103
 AFE_REG7
 SYS_TypeDef, 103
 AMC0
 ADC_TypeDef, 11
 AMC1
 ADC_TypeDef, 11
 AON
 lks32mc09x_reg.h, 436
 AON0
 lks32mc09x_reg.h, 436
 AON_TypeDef, 16
 BKP_REG0, 17
 BKP_REG1, 17
 EVT_RCD, 17
 IO_WAKE_EN, 18
 IO_WAKE_POL, 18
 PWR_CFG, 18
 AUEN
 MCPWM_InitTypeDef, 70
 MCPWM_TypeDef, 76
 BaudRate
 I2C_InitTypeDef, 62
 SPI_InitTypeDef, 95
 UART_InitTypeDef, 120
 BCR
 I2C_TypeDef, 64
 Bit9Value
 UART_InitTypeDef, 120
 BKP_REG0
 AON_TypeDef, 17
 BKP_REG1
 AON_TypeDef, 17
 BLCWIN
 CMP_TypeDef, 29
 BRR
 GPIO_TypeDef, 54
 BSIZE
 I2C_TypeDef, 64
 BSRR
 GPIO_TypeDef, 54
 BUFF
 UART_TypeDef, 124
 BURST_ADDR_CMP
 I2C_InitTypeDef, 62
 BURST_NACK
 I2C_InitTypeDef, 62
 BUS_ERR_IE
 I2C_InitTypeDef, 62

BUSRT_EN
 I2C_InitTypeDef, 62
 ByteLength
 SPI_InitTypeDef, 95

 CAP0_CLR_EN
 TIMER_TimerInitTypeDef, 109
 CAP1_CLR_EN
 TIMER_TimerInitTypeDef, 109
 Capture_IRQ_Ena
 HALL_InitTypeDef, 58
 CCR
 DMA_TypeDef, 38
 CENTER
 TIMER_TimerInitTypeDef, 110
 CF
 CLU_TypeDef, 22
 CFG
 ADC_TypeDef, 11
 CMP_TypeDef, 29
 FLASH_TypeDef, 47
 HALL_TypeDef, 60
 I2C_TypeDef, 65
 IWDG_TypeDef, 68
 QEP_TypeDef, 88
 SIF_TypeDef, 93
 SPI_TypeDef, 97
 CFG0
 TIMER_TypeDef, 117
 CFG1
 TIMER_TypeDef, 117
 CFG2
 TIMER_TypeDef, 117
 CH0_DEFAULT
 TIMER_TimerInitTypeDef, 110
 CH0_FE_CAP_EN
 TIMER_TimerInitTypeDef, 110
 CH0_RE_CAP_EN
 TIMER_TimerInitTypeDef, 110
 CH0_WorkMode
 TIMER_TimerInitTypeDef, 110
 CH0Output
 TIMER_TimerInitTypeDef, 110
 CH1_DEFAULT
 TIMER_TimerInitTypeDef, 111
 CH1_FE_CAP_EN
 TIMER_TimerInitTypeDef, 111
 CH1_RE_CAP_EN
 TIMER_TimerInitTypeDef, 111
 CH1_WorkMode
 TIMER_TimerInitTypeDef, 111
 CH1Output
 TIMER_TimerInitTypeDef, 111
 CH_DEF
 MCPWM_TypeDef, 76
 CH_FAIL
 MCPWM_TypeDef, 76
 CH_MSK
 MCPWM_TypeDef, 76

 CHN0
 ADC_TypeDef, 11
 TIMER_TimerInitTypeDef, 111
 TIMER_TypeDef, 117
 CHN0P_PWM_EN
 CMP_SingleTypeDef, 26
 CHN1
 ADC_TypeDef, 11
 TIMER_TimerInitTypeDef, 112
 TIMER_TypeDef, 117
 CHN1P_PWM_EN
 CMP_SingleTypeDef, 26
 CHN2
 ADC_TypeDef, 11
 CHN2P_PWM_EN
 CMP_SingleTypeDef, 26
 CHN3P_PWM_EN
 CMP_SingleTypeDef, 26
 CHNT
 ADC_TypeDef, 12
 CIRC
 DMA_InitTypeDef, 35
 CK_EN
 UART_InitTypeDef, 120
 CL0
 lks32mc09x_reg.h, 436
 CLK_CFG
 SYS_TypeDef, 103
 CLK_COM_DIV
 CMP_InitTypeDef, 24
 CLK_DIV
 MCPWM_InitTypeDef, 70
 CLK_DIV0
 SYS_TypeDef, 104
 CLK_DIV1
 SYS_TypeDef, 104
 CLK_DIV2
 SYS_TypeDef, 104
 CLK_FEN
 SYS_TypeDef, 104
 CLKINV
 CLU_InitTypeDef, 19
 CLKO
 EXTI_TypeDef, 45
 CLKSEL
 CLU_InitTypeDef, 19
 ClockDiv
 QEP_InitTypeDef, 86
 TIMER_TimerInitTypeDef, 112
 ClockDivision
 HALL_InitTypeDef, 58
 CLR
 IWDG_TypeDef, 68
 CLU
 lks32mc09x_reg.h, 436
 CLU0
 lks32mc09x_reg.h, 437
 CLU_Disable

lks32mc09x_clu.c, 544
 lks32mc09x_clu.h, 176
 CLU_Enable
 lks32mc09x_clu.c, 545
 lks32mc09x_clu.h, 176
 CLU_FN_0
 lks32mc09x_clu.h, 154
 CLU_FN_1
 lks32mc09x_clu.h, 154
 CLU_FN_A
 lks32mc09x_clu.h, 155
 CLU_FN_A_B
 lks32mc09x_clu.h, 155
 CLU_FN_A_B_C
 lks32mc09x_clu.h, 155
 CLU_FN_A_B_NOTC
 lks32mc09x_clu.h, 155
 CLU_FN_A_C
 lks32mc09x_clu.h, 155
 CLU_FN_A_NOTB
 lks32mc09x_clu.h, 155
 CLU_FN_A_NOTB_C
 lks32mc09x_clu.h, 156
 CLU_FN_A_NOTB_NOTC
 lks32mc09x_clu.h, 156
 CLU_FN_A_NOTC
 lks32mc09x_clu.h, 156
 CLU_FN_B
 lks32mc09x_clu.h, 156
 CLU_FN_B_C
 lks32mc09x_clu.h, 156
 CLU_FN_B_NOTC
 lks32mc09x_clu.h, 156
 CLU_FN_C
 lks32mc09x_clu.h, 157
 CLU_FN_NOTA
 lks32mc09x_clu.h, 157
 CLU_FN_NOTA_B
 lks32mc09x_clu.h, 157
 CLU_FN_NOTA_B_C
 lks32mc09x_clu.h, 157
 CLU_FN_NOTA_B_NOTC
 lks32mc09x_clu.h, 157
 CLU_FN_NOTA_C
 lks32mc09x_clu.h, 157
 CLU_FN_NOTA_NOTB
 lks32mc09x_clu.h, 158
 CLU_FN_NOTA_NOTB_C
 lks32mc09x_clu.h, 158
 CLU_FN_NOTA_NOTB_NOTC
 lks32mc09x_clu.h, 158
 CLU_FN_NOTA_NOTC
 lks32mc09x_clu.h, 158
 CLU_FN_NOTB
 lks32mc09x_clu.h, 158
 CLU_FN_NOTB_C
 lks32mc09x_clu.h, 158
 CLU_FN_NOTB_NOTC
 lks32mc09x_clu.h, 159
 CLU_FN_NOTC
 lks32mc09x_clu.h, 159
 CLU_Init
 lks32mc09x_clu.c, 545
 lks32mc09x_clu.h, 177
 CLU_InitTypeDef, 18
 CLKINV, 19
 CLKSEL, 19
 EN, 19
 FIE, 20
 FN, 20
 MXA, 20
 MXB, 20
 OEN, 20
 OUTSEL, 20
 RIE, 21
 CLU_MODEL_CLU0
 lks32mc09x_clu.h, 159
 CLU_MODEL_CLU1
 lks32mc09x_clu.h, 159
 CLU_MODEL_CLU2
 lks32mc09x_clu.h, 159
 CLU_MODEL_CLU3
 lks32mc09x_clu.h, 159
 CLU_MXA_CLU0_CLU0OUT
 lks32mc09x_clu.h, 160
 CLU_MXA_CLU0_CLU1OUT
 lks32mc09x_clu.h, 160
 CLU_MXA_CLU0_CLU2OUT
 lks32mc09x_clu.h, 160
 CLU_MXA_CLU0_CLU3OUT
 lks32mc09x_clu.h, 160
 CLU_MXA_CLU0_P0_0
 lks32mc09x_clu.h, 160
 CLU_MXA_CLU0_P0_12
 lks32mc09x_clu.h, 160
 CLU_MXA_CLU0_P0_14
 lks32mc09x_clu.h, 160
 CLU_MXA_CLU0_P0_3
 lks32mc09x_clu.h, 161
 CLU_MXA_CLU0_P0_5
 lks32mc09x_clu.h, 161
 CLU_MXA_CLU0_P0_7
 lks32mc09x_clu.h, 161
 CLU_MXA_CLU0_P1_0
 lks32mc09x_clu.h, 161
 CLU_MXA_CLU0_P2_12
 lks32mc09x_clu.h, 161
 CLU_MXA_CLU0_PWM0
 lks32mc09x_clu.h, 161
 CLU_MXA_CLU0_PWM1
 lks32mc09x_clu.h, 161
 CLU_MXA_CLU0_TIMER0ch0ovfl
 lks32mc09x_clu.h, 161
 CLU_MXA_CLU0_TIMER1ch0ovfl
 lks32mc09x_clu.h, 162
 CLU_MXA_CLU1_CLU0OUT

lks32mc09x_clu.h, [162](#)
 CLU_MXA_CLU1_CLU1OUT
 lks32mc09x_clu.h, [162](#)
 CLU_MXA_CLU1_CLU2OUT
 lks32mc09x_clu.h, [162](#)
 CLU_MXA_CLU1_CLU3OUT
 lks32mc09x_clu.h, [162](#)
 CLU_MXA_CLU1_P0_0
 lks32mc09x_clu.h, [162](#)
 CLU_MXA_CLU1_P0_11
 lks32mc09x_clu.h, [162](#)
 CLU_MXA_CLU1_P0_14
 lks32mc09x_clu.h, [162](#)
 CLU_MXA_CLU1_P0_15
 lks32mc09x_clu.h, [163](#)
 CLU_MXA_CLU1_P0_2
 lks32mc09x_clu.h, [163](#)
 CLU_MXA_CLU1_P0_5
 lks32mc09x_clu.h, [163](#)
 CLU_MXA_CLU1_P0_6
 lks32mc09x_clu.h, [163](#)
 CLU_MXA_CLU1_P2_12
 lks32mc09x_clu.h, [163](#)
 CLU_MXA_CLU1_PWM2
 lks32mc09x_clu.h, [163](#)
 CLU_MXA_CLU1_PWM3
 lks32mc09x_clu.h, [163](#)
 CLU_MXA_CLU1_TIMER0ch1ovfl
 lks32mc09x_clu.h, [163](#)
 CLU_MXA_CLU1_TIMER1ch1ovfl
 lks32mc09x_clu.h, [164](#)
 CLU_MXA_CLU2_CLU0OUT
 lks32mc09x_clu.h, [164](#)
 CLU_MXA_CLU2_CLU1OUT
 lks32mc09x_clu.h, [164](#)
 CLU_MXA_CLU2_CLU2OUT
 lks32mc09x_clu.h, [164](#)
 CLU_MXA_CLU2_CLU3OUT
 lks32mc09x_clu.h, [164](#)
 CLU_MXA_CLU2_P1_10
 lks32mc09x_clu.h, [164](#)
 CLU_MXA_CLU2_P1_13
 lks32mc09x_clu.h, [164](#)
 CLU_MXA_CLU2_P1_15
 lks32mc09x_clu.h, [164](#)
 CLU_MXA_CLU2_P2_1
 lks32mc09x_clu.h, [165](#)
 CLU_MXA_CLU2_P2_14
 lks32mc09x_clu.h, [165](#)
 CLU_MXA_CLU2_P2_4
 lks32mc09x_clu.h, [165](#)
 CLU_MXA_CLU2_P2_6
 lks32mc09x_clu.h, [165](#)
 CLU_MXA_CLU2_P2_8
 lks32mc09x_clu.h, [165](#)
 CLU_MXA_CLU2_PWM0
 lks32mc09x_clu.h, [165](#)
 CLU_MXA_CLU2_PWM2
 lks32mc09x_clu.h, [165](#)
 CLU_MXA_CLU2_TIMER2ch0ovfl
 lks32mc09x_clu.h, [165](#)
 CLU_MXA_CLU2_TIMER3ch0ovfl
 lks32mc09x_clu.h, [166](#)
 CLU_MXA_CLU3_CLU0OUT
 lks32mc09x_clu.h, [166](#)
 CLU_MXA_CLU3_CLU1OUT
 lks32mc09x_clu.h, [166](#)
 CLU_MXA_CLU3_CLU2OUT
 lks32mc09x_clu.h, [166](#)
 CLU_MXA_CLU3_CLU3OUT
 lks32mc09x_clu.h, [166](#)
 CLU_MXA_CLU3_P1_15
 lks32mc09x_clu.h, [166](#)
 CLU_MXA_CLU3_P2_0
 lks32mc09x_clu.h, [166](#)
 CLU_MXA_CLU3_P2_1
 lks32mc09x_clu.h, [166](#)
 CLU_MXA_CLU3_P2_13
 lks32mc09x_clu.h, [167](#)
 CLU_MXA_CLU3_P2_3
 lks32mc09x_clu.h, [167](#)
 CLU_MXA_CLU3_P2_6
 lks32mc09x_clu.h, [167](#)
 CLU_MXA_CLU3_P2_7
 lks32mc09x_clu.h, [167](#)
 CLU_MXA_CLU3_P2_8
 lks32mc09x_clu.h, [167](#)
 CLU_MXA_CLU3_PWM1
 lks32mc09x_clu.h, [167](#)
 CLU_MXA_CLU3_PWM3
 lks32mc09x_clu.h, [167](#)
 CLU_MXA_CLU3_TIMER2ch1ovfl
 lks32mc09x_clu.h, [168](#)
 CLU_MXA_CLU3_TIMER3ch1ovfl
 lks32mc09x_clu.h, [168](#)
 CLU_MXB_CLU0_ADBUSY0
 lks32mc09x_clu.h, [168](#)
 CLU_MXB_CLU0_CLU0OUT
 lks32mc09x_clu.h, [168](#)
 CLU_MXB_CLU0_CLU1OUT
 lks32mc09x_clu.h, [168](#)
 CLU_MXB_CLU0_CLU2OUT
 lks32mc09x_clu.h, [168](#)
 CLU_MXB_CLU0_CLU3OUT
 lks32mc09x_clu.h, [168](#)
 CLU_MXB_CLU0_CMP0_OUT
 lks32mc09x_clu.h, [168](#)
 CLU_MXB_CLU0_P0_11
 lks32mc09x_clu.h, [169](#)
 CLU_MXB_CLU0_P0_13
 lks32mc09x_clu.h, [169](#)
 CLU_MXB_CLU0_P0_15
 lks32mc09x_clu.h, [169](#)
 CLU_MXB_CLU0_P0_2
 lks32mc09x_clu.h, [169](#)
 CLU_MXB_CLU0_P0_4

lks32mc09x.clu.h, 169
 CLU_MXB_CLU0_P0_6
 lks32mc09x.clu.h, 169
 CLU_MXB_CLU0_P2_11
 lks32mc09x.clu.h, 169
 CLU_MXB_CLU0_P2_7
 lks32mc09x.clu.h, 169
 CLU_MXB_CLU0_PWM3
 lks32mc09x.clu.h, 170
 CLU_MXB_CLU0_SPIMOSI
 lks32mc09x.clu.h, 170
 CLU_MXB_CLU1_ADBUSY0
 lks32mc09x.clu.h, 170
 CLU_MXB_CLU1_CLU0OUT
 lks32mc09x.clu.h, 170
 CLU_MXB_CLU1_CLU1OUT
 lks32mc09x.clu.h, 170
 CLU_MXB_CLU1_CLU2OUT
 lks32mc09x.clu.h, 170
 CLU_MXB_CLU1_CLU3OUT
 lks32mc09x.clu.h, 170
 CLU_MXB_CLU1_CMP1_OUT
 lks32mc09x.clu.h, 170
 CLU_MXB_CLU1_P0_12
 lks32mc09x.clu.h, 171
 CLU_MXB_CLU1_P0_13
 lks32mc09x.clu.h, 171
 CLU_MXB_CLU1_P0_3
 lks32mc09x.clu.h, 171
 CLU_MXB_CLU1_P0_4
 lks32mc09x.clu.h, 171
 CLU_MXB_CLU1_P0_7
 lks32mc09x.clu.h, 171
 CLU_MXB_CLU1_P1_0
 lks32mc09x.clu.h, 171
 CLU_MXB_CLU1_P2_11
 lks32mc09x.clu.h, 171
 CLU_MXB_CLU1_P2_7
 lks32mc09x.clu.h, 171
 CLU_MXB_CLU1_PWM1
 lks32mc09x.clu.h, 172
 CLU_MXB_CLU1_SPIMISO
 lks32mc09x.clu.h, 172
 CLU_MXB_CLU2_CLU0OUT
 lks32mc09x.clu.h, 172
 CLU_MXB_CLU2_CLU1OUT
 lks32mc09x.clu.h, 172
 CLU_MXB_CLU2_CLU2OUT
 lks32mc09x.clu.h, 172
 CLU_MXB_CLU2_CLU3OUT
 lks32mc09x.clu.h, 172
 CLU_MXB_CLU2_I2CSDA
 lks32mc09x.clu.h, 172
 CLU_MXB_CLU2_P1_11
 lks32mc09x.clu.h, 172
 CLU_MXB_CLU2_P1_14
 lks32mc09x.clu.h, 173
 CLU_MXB_CLU2_P2_0

lks32mc09x.clu.h, 173
 CLU_MXB_CLU2_P2_13
 lks32mc09x.clu.h, 173
 CLU_MXB_CLU2_P2_15
 lks32mc09x.clu.h, 173
 CLU_MXB_CLU2_P2_3
 lks32mc09x.clu.h, 173
 CLU_MXB_CLU2_P2_5
 lks32mc09x.clu.h, 173
 CLU_MXB_CLU2_P3_9
 lks32mc09x.clu.h, 173
 CLU_MXB_CLU2_PWM1
 lks32mc09x.clu.h, 173
 CLU_MXB_CLU2_PWM3
 lks32mc09x.clu.h, 174
 CLU_MXB_CLU2_SPICLK
 lks32mc09x.clu.h, 174
 CLU_MXB_CLU3_CLU0OUT
 lks32mc09x.clu.h, 174
 CLU_MXB_CLU3_CLU1OUT
 lks32mc09x.clu.h, 174
 CLU_MXB_CLU3_CLU2OUT
 lks32mc09x.clu.h, 174
 CLU_MXB_CLU3_CLU3OUT
 lks32mc09x.clu.h, 174
 CLU_MXB_CLU3_I2CSCL
 lks32mc09x.clu.h, 174
 CLU_MXB_CLU3_P1_10
 lks32mc09x.clu.h, 174
 CLU_MXB_CLU3_P1_11
 lks32mc09x.clu.h, 175
 CLU_MXB_CLU3_P1_13
 lks32mc09x.clu.h, 175
 CLU_MXB_CLU3_P1_14
 lks32mc09x.clu.h, 175
 CLU_MXB_CLU3_P2_14
 lks32mc09x.clu.h, 175
 CLU_MXB_CLU3_P2_15
 lks32mc09x.clu.h, 175
 CLU_MXB_CLU3_P2_4
 lks32mc09x.clu.h, 175
 CLU_MXB_CLU3_P2_5
 lks32mc09x.clu.h, 175
 CLU_MXB_CLU3_PWM0
 lks32mc09x.clu.h, 175
 CLU_MXB_CLU3_PWM2
 lks32mc09x.clu.h, 176
 CLU_MXB_CLU3_SPICSN
 lks32mc09x.clu.h, 176
 CLU_ReadOutVole
 lks32mc09x.clu.c, 546
 lks32mc09x.clu.h, 177
 CLU_StructInit
 lks32mc09x.clu.c, 546
 lks32mc09x.clu.h, 178
 CLU_TypeDef, 21
 CF, 22
 EN, 22

FN, 22
 IE, 22
 IF, 22
 MX, 23
 OUT, 23
 CMP
 lks32mc09x_reg.h, 437
 CMP0
 lks32mc09x_reg.h, 437
 CMP0_CFG
 CMP_InitTypeDef, 24
 CMP0_TRIGGER_MODE
 TIMER_TimerInitTypeDef, 112
 CMP1_CFG
 CMP_InitTypeDef, 24
 CMP1_TRIGGER_MODE
 TIMER_TimerInitTypeDef, 112
 CMP_CHN_0
 lks32mc09x_cmp.h, 184
 CMP_CHN_1
 lks32mc09x_cmp.h, 184
 CMP_ClearIRQFlag
 lks32mc09x_cmp.c, 549
 lks32mc09x_cmp.h, 194
 CMP_CLK_DIV_0
 lks32mc09x_cmp.h, 184
 CMP_CLK_DIV_1
 lks32mc09x_cmp.h, 184
 CMP_CLK_DIV_2
 lks32mc09x_cmp.h, 185
 CMP_CLK_DIV_3
 lks32mc09x_cmp.h, 185
 CMP_CLK_DIV_4
 lks32mc09x_cmp.h, 185
 CMP_Disable
 lks32mc09x_cmp.c, 549
 lks32mc09x_cmp.h, 194
 CMP_Enable
 lks32mc09x_cmp.c, 550
 lks32mc09x_cmp.h, 195
 CMP_GetFilterValue
 lks32mc09x_cmp.c, 550
 lks32mc09x_cmp.h, 196
 CMP_GetIRQFlag
 lks32mc09x_cmp.c, 550
 lks32mc09x_cmp.h, 196
 CMP_GetValue
 lks32mc09x_cmp.c, 551
 lks32mc09x_cmp.h, 197
 CMP_HYS_0mV
 lks32mc09x_cmp.h, 185
 CMP_HYS_20mV
 lks32mc09x_cmp.h, 185
 CMP_IE_CMP0
 lks32mc09x_cmp.h, 185
 CMP_IE_CMP1
 lks32mc09x_cmp.h, 185
 CMP_IF_CMP0
 lks32mc09x_cmp.h, 186
 CMP_IF_CMP1
 lks32mc09x_cmp.h, 186
 CMP_Init
 lks32mc09x_cmp.c, 551
 lks32mc09x_cmp.h, 197
 CMP_InitTypeDef, 23
 CLK_COM_DIV, 24
 CMP0_CFG, 24
 CMP1_CFG, 24
 HYS, 24
 CMP_RE_CMP0
 lks32mc09x_cmp.h, 186
 CMP_RE_CMP1
 lks32mc09x_cmp.h, 186
 CMP_SELN_0_DAC0
 lks32mc09x_cmp.h, 186
 CMP_SELN_0_DAC1
 lks32mc09x_cmp.h, 186
 CMP_SELN_0_HS_MID0
 lks32mc09x_cmp.h, 186
 CMP_SELN_0_IN_CMP0
 lks32mc09x_cmp.h, 187
 CMP_SELN_0_VBGP
 lks32mc09x_cmp.h, 187
 CMP_SELN_1_DAC0
 lks32mc09x_cmp.h, 187
 CMP_SELN_1_DAC1
 lks32mc09x_cmp.h, 187
 CMP_SELN_1_HS_MID1
 lks32mc09x_cmp.h, 187
 CMP_SELN_1_IN_CMP1
 lks32mc09x_cmp.h, 187
 CMP_SELN_1_VBGP
 lks32mc09x_cmp.h, 188
 CMP_SEL0_CMP0_IP0
 lks32mc09x_cmp.h, 188
 CMP_SEL0_CMP0_IP1
 lks32mc09x_cmp.h, 188
 CMP_SEL0_CMP0_IP2
 lks32mc09x_cmp.h, 188
 CMP_SEL0_CMP0_IP3
 lks32mc09x_cmp.h, 188
 CMP_SEL0_CMP0_IP4
 lks32mc09x_cmp.h, 188
 CMP_SEL0_CMP1_IP1
 lks32mc09x_cmp.h, 189
 CMP_SEL0_CMP1_IP2
 lks32mc09x_cmp.h, 189
 CMP_SEL0_CMP1_IP3
 lks32mc09x_cmp.h, 189
 CMP_SEL0_DAC0_OUT
 lks32mc09x_cmp.h, 189
 CMP_SEL0_OPA0_IP
 lks32mc09x_cmp.h, 189
 CMP_SEL0_OPA0_OUTP
 lks32mc09x_cmp.h, 189
 CMP_SEL0_OPA1_IP

lks32mc09x_cmp.h, 190
 CMP_SEL_P_0_OPA1_OUTP
 lks32mc09x_cmp.h, 190
 CMP_SEL_P_0_OUT_OPA_CMP
 lks32mc09x_cmp.h, 190
 CMP_SEL_P_1_CMP0_IP1
 lks32mc09x_cmp.h, 190
 CMP_SEL_P_1_CMP0_IP2
 lks32mc09x_cmp.h, 190
 CMP_SEL_P_1_CMP0_IP3
 lks32mc09x_cmp.h, 190
 CMP_SEL_P_1_CMP1_IP0
 lks32mc09x_cmp.h, 191
 CMP_SEL_P_1_CMP1_IP1
 lks32mc09x_cmp.h, 191
 CMP_SEL_P_1_CMP1_IP2
 lks32mc09x_cmp.h, 191
 CMP_SEL_P_1_CMP1_IP3
 lks32mc09x_cmp.h, 191
 CMP_SEL_P_1_DAC1_OUT
 lks32mc09x_cmp.h, 191
 CMP_SEL_P_1_OPA2_IP
 lks32mc09x_cmp.h, 191
 CMP_SEL_P_1_OPA2_OUTP
 lks32mc09x_cmp.h, 192
 CMP_SEL_P_1_OPA3_IP
 lks32mc09x_cmp.h, 192
 CMP_SEL_P_1_OPA3_OUTP
 lks32mc09x_cmp.h, 192
 CMP_SEL_P_1_OPA_OUT_CMP
 lks32mc09x_cmp.h, 192
 CMP_SetFilterWidth
 lks32mc09x_cmp.c, 552
 lks32mc09x_cmp.h, 198
 CMP_SetHysteresis
 lks32mc09x_cmp.c, 552
 lks32mc09x_cmp.h, 198
 CMP_SetIRQSignal
 lks32mc09x_cmp.c, 553
 lks32mc09x_cmp.h, 199
 CMP_SetSignal
 lks32mc09x_cmp.c, 553
 lks32mc09x_cmp.h, 199
 CMP_SetWindowSignal
 lks32mc09x_cmp.c, 554
 lks32mc09x_cmp.h, 200
 CMP_SIGNAL_EDGE_N
 lks32mc09x_cmp.h, 192
 CMP_SIGNAL_EDGE_P
 lks32mc09x_cmp.h, 192
 CMP_SIGNAL_LEVEL_H
 lks32mc09x_cmp.h, 193
 CMP_SIGNAL_LEVEL_L
 lks32mc09x_cmp.h, 193
 CMP_SIGNAL_NONE
 lks32mc09x_cmp.h, 193
 CMP_SingleTypeDef, 25
 CHNOP_PWM_EN, 26
 CHN1P_PWM_EN, 26
 CHN2P_PWM_EN, 26
 CHN3P_PWM_EN, 26
 EN, 26
 FIL_CLK_DIV16, 26
 FIL_CLK_DIV2, 26
 IE, 27
 IRQ_TRIG, 27
 POL, 27
 RE, 27
 SELN, 27
 SELP, 27
 W_PWM_POL, 28
 CMP_StructInit
 lks32mc09x_cmp.c, 555
 lks32mc09x_cmp.h, 201
 CMP_TypeDef, 28
 BLCWIN, 29
 CFG, 29
 DATA, 29
 IE, 29
 IF, 29
 TCLK, 30
 CMP_WIN_LEVEL_H
 lks32mc09x_cmp.h, 193
 CMP_WIN_LEVEL_L
 lks32mc09x_cmp.h, 193
 CMP_WIN_SIGNAL_MCPWM0_0
 lks32mc09x_cmp.h, 193
 CMP_WIN_SIGNAL_MCPWM0_1
 lks32mc09x_cmp.h, 193
 CMP_WIN_SIGNAL_MCPWM0_2
 lks32mc09x_cmp.h, 193
 CMP_WIN_SIGNAL_MCPWM0_3
 lks32mc09x_cmp.h, 194
 CMP_WIN_SIGNAL_NONE
 lks32mc09x_cmp.h, 194
 CNT
 HALL_TypeDef, 60
 IWDG_TypeDef, 69
 QEP_TypeDef, 88
 TIMER_TimerInitTypeDef, 112
 TIMER_TypeDef, 118
 CNT0
 MCPWM_TypeDef, 77
 COUNT
 MCPWM_InitTypeDef, 70
 CountTH
 HALL_InitTypeDef, 58
 CPHA
 SPI_InitTypeDef, 95
 CPOL
 SPI_InitTypeDef, 95
 CR
 CRC_TypeDef, 32
 CR0
 EXTI_TypeDef, 45
 CR1

EXTI_TypeDef, 45

CRC0

lks32mc09x_reg.h, 437

CRC_Calc

lks32mc09x_crc.c, 560

lks32mc09x_crc.h, 208

CRC_CalcCommon

lks32mc09x_crc.c, 560

lks32mc09x_crc.h, 208

CRC_Disable

lks32mc09x_crc.c, 561

lks32mc09x_crc.h, 209

CRC_Enable

lks32mc09x_crc.c, 562

lks32mc09x_crc.h, 209

CRC_Init

lks32mc09x_crc.c, 562

lks32mc09x_crc.h, 210

CRC_InitTypeDef, 30

INIT, 31

null, 31

POLY, 31

REFIN, 31

REFOUT, 31

WIDTH, 31

XOROUT, 31

CRC_MODEL_CRC_16_CCITT

lks32mc09x_crc.h, 205

CRC_MODEL_CRC_16_CCITT_FALSE

lks32mc09x_crc.h, 205

CRC_MODEL_CRC_16_DNP

lks32mc09x_crc.h, 205

CRC_MODEL_CRC_16_IBMx1

lks32mc09x_crc.h, 205

CRC_MODEL_CRC_16_MAXIM

lks32mc09x_crc.h, 205

CRC_MODEL_CRC_16_MODBU

lks32mc09x_crc.h, 205

CRC_MODEL_CRC_16_USBx1

lks32mc09x_crc.h, 205

CRC_MODEL_CRC_16_X25

lks32mc09x_crc.h, 206

CRC_MODEL_CRC_16_XMODEM

lks32mc09x_crc.h, 206

CRC_MODEL_CRC_32

lks32mc09x_crc.h, 206

CRC_MODEL_CRC_32_MPEG_2

lks32mc09x_crc.h, 206

CRC_MODEL_CRC_8

lks32mc09x_crc.h, 206

CRC_MODEL_CRC_8_ITU

lks32mc09x_crc.h, 206

CRC_MODEL_CRC_8_MAXIMx

lks32mc09x_crc.h, 207

CRC_MODEL_CRC_8_ROHC

lks32mc09x_crc.h, 207

CRC_StructInit

lks32mc09x_crc.c, 563

lks32mc09x_crc.h, 211

CRC_TypeDef, 32

CR, 32

DR, 32

INIT, 33

POL, 33

CRC_WIDTH_16

lks32mc09x_crc.h, 207

CRC_WIDTH_32

lks32mc09x_crc.h, 207

CRC_WIDTH_7

lks32mc09x_crc.h, 207

CRC_WIDTH_8

lks32mc09x_crc.h, 207

CRC_XOROUT

lks32mc09x_crc.c, 564

CS

SPI_InitTypeDef, 95

CSMP

ADC_InitTypeDef, 6

CTMS

DMA_TypeDef, 38

CTRL

DMAMODULE_TypeDef, 39

UART_TypeDef, 124

DAC_CHANNEL_0

lks32mc09x_dac.h, 214

DAC_CHANNEL_1

lks32mc09x_dac.h, 214

DAC_Cmd

lks32mc09x_dac.c, 569

lks32mc09x_dac.h, 217

DAC_Disable

lks32mc09x_dac.c, 569

lks32mc09x_dac.h, 218

DAC_Enable

lks32mc09x_dac.c, 570

lks32mc09x_dac.h, 218

DAC_Init

lks32mc09x_dac.c, 570

lks32mc09x_dac.h, 219

DAC_InitTypeDef, 33

DAC_RANGE, 34

DAC_STEP, 34

DAC_STEP_TRG_x, 34

DACOUT_EN, 34

DAC_Output

lks32mc09x_dac.c, 571

lks32mc09x_dac.h, 220

DAC_OutputValue

lks32mc09x_dac.c, 572

lks32mc09x_dac.h, 220

DAC_OutputVoltage

lks32mc09x_dac.c, 572

lks32mc09x_dac.h, 221

DAC_RANGE

DAC_InitTypeDef, 34

DAC.Range

lks32mc09x_dac.c, [573](#)
 lks32mc09x_dac.h, [222](#)
 DAC_RANGE_0_1V2
 lks32mc09x_dac.h, [215](#)
 DAC_RANGE_0_4V76
 lks32mc09x_dac.h, [215](#)
 DAC_RANGE_1_1V2
 lks32mc09x_dac.h, [215](#)
 DAC_RANGE_1_3V
 lks32mc09x_dac.h, [215](#)
 DAC_Reset
 lks32mc09x_dac.c, [573](#)
 lks32mc09x_dac.h, [222](#)
 DAC_SetRange
 lks32mc09x_dac.c, [574](#)
 lks32mc09x_dac.h, [223](#)
 DAC_STEP
 DAC_InitTypeDef, [34](#)
 DAC_Step
 lks32mc09x_dac.c, [575](#)
 lks32mc09x_dac.h, [224](#)
 DAC_STEP_1
 lks32mc09x_dac.h, [215](#)
 DAC_STEP_2
 lks32mc09x_dac.h, [215](#)
 DAC_STEP_3
 lks32mc09x_dac.h, [216](#)
 DAC_STEP_0
 lks32mc09x_dac.h, [216](#)
 DAC_STEP_1
 lks32mc09x_dac.h, [216](#)
 DAC_STEP_2
 lks32mc09x_dac.h, [216](#)
 DAC_STEP_3
 lks32mc09x_dac.h, [216](#)
 DAC_STEP_4
 lks32mc09x_dac.h, [216](#)
 DAC_STEP_TRG_NONE
 lks32mc09x_dac.h, [217](#)
 DAC_STEP_TRG_TIMER0
 lks32mc09x_dac.h, [217](#)
 DAC_STEP_TRG_TIMER1
 lks32mc09x_dac.h, [217](#)
 DAC_STEP_TRG_TIMER2
 lks32mc09x_dac.h, [217](#)
 DAC_STEP_TRG_TIMER3
 lks32mc09x_dac.h, [217](#)
 DAC_STEP_TRG_x
 DAC_InitTypeDef, [34](#)
 DAC_StructInit
 lks32mc09x_dac.c, [576](#)
 lks32mc09x_dac.h, [224](#)
 DAC_Value
 lks32mc09x_dac.c, [576](#)
 lks32mc09x_dac.h, [226](#)
 DAC_Voltage
 lks32mc09x_dac.c, [577](#)
 lks32mc09x_dac.h, [226](#)
 DACOUT_EN
 DAC_InitTypeDef, [34](#)
 DADR
 DMA_InitTypeDef, [35](#)
 DMA_TypeDef, [38](#)
 DAT0
 ADC_TypeDef, [12](#)
 DAT1
 ADC_TypeDef, [12](#)
 DAT10
 ADC_TypeDef, [12](#)
 DAT11
 ADC_TypeDef, [12](#)
 DAT2
 ADC_TypeDef, [12](#)
 DAT3
 ADC_TypeDef, [13](#)
 DAT4
 ADC_TypeDef, [13](#)
 DAT5
 ADC_TypeDef, [13](#)
 DAT6
 ADC_TypeDef, [13](#)
 DAT7
 ADC_TypeDef, [13](#)
 DAT8
 ADC_TypeDef, [13](#)
 DAT9
 ADC_TypeDef, [14](#)
 DATA
 CMP_TypeDef, [29](#)
 I2C_TypeDef, [65](#)
 DATA_ALIGN
 ADC_InitTypeDef, [6](#)
 DBG_CFG
 SYS_TypeDef, [104](#)
 DBTW
 DMA_InitTypeDef, [35](#)
 DC0
 ADC_TypeDef, [14](#)
 DC1
 ADC_TypeDef, [14](#)
 DID
 DSP_Typedef, [40](#)
 DID1
 DSP_Typedef, [40](#)
 DINC
 DMA_InitTypeDef, [36](#)
 DIR
 TIMER_TimerInitTypeDef, [112](#)
 DIS
 DSP_Typedef, [41](#)
 DIS1
 DSP_Typedef, [41](#)
 DIV
 SPI_TypeDef, [97](#)
 DIVH
 UART_TypeDef, [124](#)

DIVL
 UART_TypeDef, 124

DMA0
 lks32mc09x_reg.h, 437

DMA1
 lks32mc09x_reg.h, 437

DMA2
 lks32mc09x_reg.h, 437

DMA3
 lks32mc09x_reg.h, 437

DMA_BYTE_TRANS
 lks32mc09x_dma.h, 231

DMA_CHN0
 lks32mc09x_reg.h, 438

DMA_CHN1
 lks32mc09x_reg.h, 438

DMA_CHN2
 lks32mc09x_reg.h, 438

DMA_CHN3
 lks32mc09x_reg.h, 438

DMA_CHx_EN
 lks32mc09x_dma.c, 582
 lks32mc09x_dma.h, 240

DMA_ClearIRQFlag
 lks32mc09x_dma.c, 582
 lks32mc09x_dma.h, 241

DMA_Disable
 lks32mc09x_dma.c, 583
 lks32mc09x_dma.h, 241

DMA_Enable
 lks32mc09x_dma.c, 583
 lks32mc09x_dma.h, 241

DMA_GetFlag
 lks32mc09x_dma.c, 583
 lks32mc09x_dma.h, 241

DMA_GetIRQFlag
 lks32mc09x_dma.c, 584
 lks32mc09x_dma.h, 242

DMA_HALFWORD_TRANS
 lks32mc09x_dma.h, 231

DMA_IE_CH0
 lks32mc09x_dma.h, 231

DMA_IE_CH1
 lks32mc09x_dma.h, 232

DMA_IE_CH2
 lks32mc09x_dma.h, 232

DMA_IE_CH3
 lks32mc09x_dma.h, 232

DMA_IF_CH0
 lks32mc09x_dma.h, 232

DMA_IF_CH1
 lks32mc09x_dma.h, 232

DMA_IF_CH2
 lks32mc09x_dma.h, 232

DMA_IF_CH3
 lks32mc09x_dma.h, 232

DMA_Init
 lks32mc09x_dma.c, 584
 lks32mc09x_dma.h, 242

DMA_InitTypeDef, 34

CIRC, 35

DADR, 35

DBTW, 35

DINC, 36

EN, 36

IRQ_EN, 36

REN, 36

RMODE, 36

SADR, 36

SBTW, 37

SINC, 37

TMS, 37

DMA_Mem2Reg
 lks32mc09x_dma.c, 585
 lks32mc09x_dma.h, 243

DMA_Memcpy
 lks32mc09x_dma.c, 586
 lks32mc09x_dma.h, 244

DMA_MEMORY_2_PERI
 lks32mc09x_dma.h, 233

DMA_PERI_2_MEMORY
 lks32mc09x_dma.h, 233

DMA_REN_ADC0
 lks32mc09x_dma.h, 233

DMA_REN_ADC1
 lks32mc09x_dma.h, 233

DMA_REN_CMP
 lks32mc09x_dma.h, 233

DMA_REN_DSP
 lks32mc09x_dma.h, 233

DMA_REN_GPIO
 lks32mc09x_dma.h, 233

DMA_REN_HALL0
 lks32mc09x_dma.h, 234

DMA_REN_I2C0_RX
 lks32mc09x_dma.h, 234

DMA_REN_I2C0_TX
 lks32mc09x_dma.h, 234

DMA_REN_MCPWM_RE0
 lks32mc09x_dma.h, 234

DMA_REN_MCPWM_RE1
 lks32mc09x_dma.h, 234

DMA_REN_SIF
 lks32mc09x_dma.h, 234

DMA_REN_SPI_RX
 lks32mc09x_dma.h, 234

DMA_REN_SPI_TX
 lks32mc09x_dma.h, 234

DMA_REN_SW
 lks32mc09x_dma.h, 235

DMA_REN_TIMER0
 lks32mc09x_dma.h, 235

DMA_REN_TIMER1
 lks32mc09x_dma.h, 235

DMA_REN_TIMER2
 lks32mc09x_dma.h, 235

- DMA_REN_TIMER3
 lks32mc09x_dma.h, 235
 DMA_REN_UART0_RX
 lks32mc09x_dma.h, 235
 DMA_REN_UART0_TX
 lks32mc09x_dma.h, 235
 DMA_REN_UART1_RX
 lks32mc09x_dma.h, 235
 DMA_REN_UART1_TX
 lks32mc09x_dma.h, 236
 DMA_REN_UART2_RX
 lks32mc09x_dma.h, 236
 DMA_REN_UART2_TX
 lks32mc09x_dma.h, 236
 DMA_REQ_ADC0
 lks32mc09x_dma.h, 236
 DMA_REQ_ADC1
 lks32mc09x_dma.h, 236
 DMA_REQ_CMP
 lks32mc09x_dma.h, 236
 DMA_REQ_DSP
 lks32mc09x_dma.h, 236
 DMA_REQ_GPIO
 lks32mc09x_dma.h, 237
 DMA_REQ_HALL
 lks32mc09x_dma.h, 237
 DMA_REQ_I2C0_RX
 lks32mc09x_dma.h, 237
 DMA_REQ_I2C0_TX
 lks32mc09x_dma.h, 237
 DMA_REQ_MCPWM0
 lks32mc09x_dma.h, 237
 DMA_REQ_MCPWM1
 lks32mc09x_dma.h, 237
 DMA_REQ_SIF
 lks32mc09x_dma.h, 238
 DMA_REQ_SPL_RX
 lks32mc09x_dma.h, 238
 DMA_REQ_SPL_TX
 lks32mc09x_dma.h, 238
 DMA_REQ_SW
 lks32mc09x_dma.h, 238
 DMA_REQ_TIMER0
 lks32mc09x_dma.h, 238
 DMA_REQ_TIMER1
 lks32mc09x_dma.h, 238
 DMA_REQ_TIMER2
 lks32mc09x_dma.h, 239
 DMA_REQ_TIMER3
 lks32mc09x_dma.h, 239
 DMA_REQ_UART0_RX
 lks32mc09x_dma.h, 239
 DMA_REQ_UART0_TX
 lks32mc09x_dma.h, 239
 DMA_REQ_UART1_RX
 lks32mc09x_dma.h, 239
 DMA_REQ_UART1_TX
 lks32mc09x_dma.h, 239
 DMA_Reset
 lks32mc09x_dma.c, 587
 lks32mc09x_dma.h, 245
 DMA_SetAddr
 lks32mc09x_dma.c, 588
 lks32mc09x_dma.h, 246
 DMA_SetLength
 lks32mc09x_dma.c, 588
 lks32mc09x_dma.h, 246
 DMA_SetRequest
 lks32mc09x_dma.c, 589
 lks32mc09x_dma.h, 247
 DMA_StructInit
 lks32mc09x_dma.c, 589
 lks32mc09x_dma.h, 247
 DMA_TCIE
 lks32mc09x_dma.h, 240
 DMA_TEIE
 lks32mc09x_dma.h, 240
 DMA_TOP
 lks32mc09x_reg.h, 438
 DMA_Trigger
 lks32mc09x_dma.c, 589
 lks32mc09x_dma.h, 248
 DMA_TypeDef, 37
 CCR, 38
 CTMS, 38
 DADR, 38
 REN, 38
 SADR, 38
 DMA_WORD_TRANS
 lks32mc09x_dma.h, 240
 DMAMODULE_TypeDef, 39
 CTRL, 39
 IE, 39
 IF, 39
 DMARE
 UART_InitTypeDef, 121
 DMAREG
 lks32mc09x_reg.h, 438
 DR
 CRC_TypeDef, 32
 DSP
 lks32mc09x_reg.h, 438
 DSP0
 lks32mc09x_reg.h, 438
 DSP_Disable
 lks32mc09x_dsp.c, 594
 lks32mc09x_dsp.h, 251
 DSP_Div
 lks32mc09x_dsp.c, 595
 lks32mc09x_dsp.h, 251
 DSP_Enable
 lks32mc09x_dsp.c, 595
 lks32mc09x_dsp.h, 252
 DSP_Init
 lks32mc09x_dsp.c, 596
 lks32mc09x_dsp.h, 252

DSP_Mod
 lks32mc09x_dsp.c, 596
 lks32mc09x_dsp.h, 252
 DSP_REG
 lks32mc09x_reg.h, 439
 DSP_Reset
 lks32mc09x_dsp.c, 597
 lks32mc09x_dsp.h, 253
 DSP_Rms
 lks32mc09x_dsp.c, 597
 lks32mc09x_dsp.h, 253
 DSP_Sqrt
 lks32mc09x_dsp.c, 597
 lks32mc09x_dsp.h, 254
 DSP_Typedef, 40
 DID, 40
 DID1, 40
 DIS, 41
 DIS1, 41
 QUO, 41
 QUO1, 41
 RAD, 41
 REM, 41
 RESERVED, 42
 RESERVED1, 42
 RESERVED2, 42
 SQRT, 42
 DTH00
 MCPWM_TypeDef, 77
 DTH01
 MCPWM_TypeDef, 77
 DTHN
 MCPWM_InitTypeDef, 70
 DTHP
 MCPWM_InitTypeDef, 71
 DUPLEX
 UART_InitTypeDef, 121
 Duplex
 SPI_InitTypeDef, 95
 DWK_EN
 IWDG_InitTypeDef, 67
 EEPROM_ADDR_LEN_16
 lks32mc09x_eeprom.h, 256
 EEPROM_ADDR_LEN_8
 lks32mc09x_eeprom.h, 256
 EEPROM_BAUD_100K
 lks32mc09x_eeprom.h, 257
 EEPROM_BAUD_1M
 lks32mc09x_eeprom.h, 257
 EEPROM_BAUD_1M2
 lks32mc09x_eeprom.h, 257
 EEPROM_BAUD_400K
 lks32mc09x_eeprom.h, 257
 EEPROM_BAUD_x
 EEPROM_InitTypeDef, 43
 EEPROM_CFG
 lks32mc09x_eeprom.c, 600
 EEPROM_Disable
 lks32mc09x_eeprom.c, 600
 lks32mc09x_eeprom.h, 258
 EEPROM_Enable
 lks32mc09x_eeprom.c, 600
 lks32mc09x_eeprom.h, 258
 EEPROM_Init
 lks32mc09x_eeprom.c, 601
 lks32mc09x_eeprom.h, 258
 EEPROM_InitDefault
 lks32mc09x_eeprom.c, 602
 lks32mc09x_eeprom.h, 259
 EEPROM_InitTypeDef, 42
 addr, 43
 ADDR_LEN_x, 43
 EEPROM_BAUD_x, 43
 IO_SWITCH, 43
 RDY_RLS, 44
 EEPROM_RDATA
 lks32mc09x_eeprom.c, 600
 EEPROM_Read
 lks32mc09x_eeprom.c, 602
 lks32mc09x_eeprom.h, 259
 EEPROM_ReadData
 lks32mc09x_eeprom.c, 603
 lks32mc09x_eeprom.h, 260
 EEPROM_Reset
 lks32mc09x_eeprom.c, 603
 lks32mc09x_eeprom.h, 261
 EEPROM_StructInit
 lks32mc09x_eeprom.c, 604
 lks32mc09x_eeprom.h, 261
 EEPROM_Write
 lks32mc09x_eeprom.c, 604
 lks32mc09x_eeprom.h, 261
 EEPROM_WriteData
 lks32mc09x_eeprom.c, 604
 lks32mc09x_eeprom.h, 262
 EIE
 MCPWM_InitTypeDef, 71
 MCPWM_TypeDef, 77
 EIF
 MCPWM_TypeDef, 77
 EN
 CLU_InitTypeDef, 19
 CLU_TypeDef, 22
 CMP_SingleTypeDef, 26
 DMA_InitTypeDef, 36
 MCPWM_InitTypeDef, 71
 SPI_InitTypeDef, 95
 TIMER_TimerInitTypeDef, 113
 ENDIAN
 SPI_InitTypeDef, 96
 EPSM
 FLASH_TypeDef, 48
 EPSN
 FLASH_TypeDef, 48
 ERASE
 FLASH_TypeDef, 48

ETON
 TIMER_TimerInitTypeDef, 113
 EVT
 TIMER_TimerInitTypeDef, 113
 TIMER_TypeDef, 118
 EVT0
 MCPWM_TypeDef, 77
 EVT_RCD
 AON_TypeDef, 17
 EXTI
 lks32mc09x_reg.h, 439
 EXTI0
 lks32mc09x_reg.h, 439
 EXTI_ClearIRQFlag
 lks32mc09x_exti.c, 608
 lks32mc09x_exti.h, 270
 EXTI_CLKO_0
 lks32mc09x_exti.h, 265
 EXTI_CLKO_ADC
 lks32mc09x_exti.h, 265
 EXTI_CLKO_HSE
 lks32mc09x_exti.h, 265
 EXTI_CLKO_HSI
 lks32mc09x_exti.h, 265
 EXTI_CLKO_LSI
 lks32mc09x_exti.h, 265
 EXTI_CLKO_PLL
 lks32mc09x_exti.h, 266
 EXTI_CR_P0_0
 lks32mc09x_exti.h, 266
 EXTI_CR_P0_11
 lks32mc09x_exti.h, 266
 EXTI_CR_P0_12
 lks32mc09x_exti.h, 266
 EXTI_CR_P0_13
 lks32mc09x_exti.h, 266
 EXTI_CR_P0_14
 lks32mc09x_exti.h, 266
 EXTI_CR_P0_15
 lks32mc09x_exti.h, 266
 EXTI_CR_P0_2
 lks32mc09x_exti.h, 266
 EXTI_CR_P1_0
 lks32mc09x_exti.h, 267
 EXTI_CR_P2_0
 lks32mc09x_exti.h, 267
 EXTI_CR_P2_13
 lks32mc09x_exti.h, 267
 EXTI_CR_P2_3
 lks32mc09x_exti.h, 267
 EXTI_CR_P2_4
 lks32mc09x_exti.h, 267
 EXTI_CR_P2_5
 lks32mc09x_exti.h, 267
 EXTI_CR_P2_6
 lks32mc09x_exti.h, 267
 EXTI_CR_P2_7
 lks32mc09x_exti.h, 267
 EXTI_CR_P2_9
 lks32mc09x_exti.h, 268
 EXTI_DisableIRQFlag
 lks32mc09x_exti.c, 608
 lks32mc09x_exti.h, 270
 EXTI_EnableIRQFlag
 lks32mc09x_exti.c, 608
 lks32mc09x_exti.h, 271
 EXTI_GetIRQFlag
 lks32mc09x_exti.c, 609
 lks32mc09x_exti.h, 271
 exti_ie_flg
 lks32mc09x_exti.c, 610
 EXTI_IF_P0_0
 lks32mc09x_exti.h, 268
 EXTI_IF_P0_11
 lks32mc09x_exti.h, 268
 EXTI_IF_P0_12
 lks32mc09x_exti.h, 268
 EXTI_IF_P0_13
 lks32mc09x_exti.h, 268
 EXTI_IF_P0_14
 lks32mc09x_exti.h, 268
 EXTI_IF_P0_15
 lks32mc09x_exti.h, 268
 EXTI_IF_P0_2
 lks32mc09x_exti.h, 268
 EXTI_IF_P1_0
 lks32mc09x_exti.h, 269
 EXTI_IF_P2_0
 lks32mc09x_exti.h, 269
 EXTI_IF_P2_13
 lks32mc09x_exti.h, 269
 EXTI_IF_P2_3
 lks32mc09x_exti.h, 269
 EXTI_IF_P2_4
 lks32mc09x_exti.h, 269
 EXTI_IF_P2_5
 lks32mc09x_exti.h, 269
 EXTI_IF_P2_6
 lks32mc09x_exti.h, 269
 EXTI_IF_P2_7
 lks32mc09x_exti.h, 269
 EXTI_IF_P2_9
 lks32mc09x_exti.h, 270
 EXTI_Mode_TypeDef
 lks32mc09x_exti.h, 270
 EXTI_Reset
 lks32mc09x_exti.c, 609
 lks32mc09x_exti.h, 272
 EXTI_SetClko
 lks32mc09x_exti.c, 609
 lks32mc09x_exti.h, 272
 EXTI_Trigger_Edge
 lks32mc09x_exti.h, 270
 EXTI_Trigger_Negedge
 lks32mc09x_exti.h, 270
 EXTI_Trigger_None

lks32mc09x_exti.h, 270
 EXTI_Trigger_Posedge
 lks32mc09x_exti.h, 270
 EXTI_TriggerConfig
 lks32mc09x_exti.c, 610
 lks32mc09x_exti.h, 273
 EXTI_TypeDef, 44
 CLKO, 45
 CR0, 45
 CR1, 45
 IE, 45
 IF, 45
 SWAP, 46
 F3210
 GPIO_TypeDef, 54
 F7654
 GPIO_TypeDef, 55
 FAIL0
 MCPWM_InitTypeDef, 71
 FAIL1
 MCPWM_InitTypeDef, 71
 FAIL_EN
 TIMER_TimrInitTypeDef, 113
 FAIL_IO
 MCPWM_InitTypeDef, 71
 FAIL_POL
 TIMER_TimrInitTypeDef, 113
 FAIL_SEL
 TIMER_TimrInitTypeDef, 113
 FBA98
 GPIO_TypeDef, 55
 FCNT
 MCPWM_TypeDef, 78
 FE_CNT_EN
 QEP_InitTypeDef, 86
 FFEDC
 GPIO_TypeDef, 55
 FIE
 CLU_InitTypeDef, 20
 FIL_CLK_DIV16
 CMP_SingleTypeDef, 26
 FIL_CLK_DIV2
 CMP_SingleTypeDef, 26
 Filter
 QEP_InitTypeDef, 86
 Filter75_Ena
 HALL_InitTypeDef, 58
 FilterLen
 HALL_InitTypeDef, 58
 FirstSend
 UART_InitTypeDef, 121
 FLASH
 lks32mc09x_reg.h, 439
 FLASH0
 lks32mc09x_reg.h, 439
 FLASH_Erase
 lks32mc09x_flash.c, 612
 lks32mc09x_flash.h, 312
 FLASH_EraseNvr
 lks32mc09x_flash.c, 613
 lks32mc09x_flash.h, 313
 FLASH_Read
 lks32mc09x_flash.c, 613
 lks32mc09x_flash.h, 313
 FLASH_ReadNvr
 lks32mc09x_flash.c, 613
 lks32mc09x_flash.h, 313
 FLASH_SECTOR
 lks32mc09x_flash.h, 280
 FLASH_SECTOR_0
 lks32mc09x_flash.h, 280
 FLASH_SECTOR_1
 lks32mc09x_flash.h, 280
 FLASH_SECTOR_10
 lks32mc09x_flash.h, 280
 FLASH_SECTOR_100
 lks32mc09x_flash.h, 280
 FLASH_SECTOR_101
 lks32mc09x_flash.h, 281
 FLASH_SECTOR_102
 lks32mc09x_flash.h, 281
 FLASH_SECTOR_103
 lks32mc09x_flash.h, 281
 FLASH_SECTOR_104
 lks32mc09x_flash.h, 281
 FLASH_SECTOR_105
 lks32mc09x_flash.h, 281
 FLASH_SECTOR_106
 lks32mc09x_flash.h, 281
 FLASH_SECTOR_107
 lks32mc09x_flash.h, 281
 FLASH_SECTOR_108
 lks32mc09x_flash.h, 281
 FLASH_SECTOR_109
 lks32mc09x_flash.h, 282
 FLASH_SECTOR_11
 lks32mc09x_flash.h, 282
 FLASH_SECTOR_110
 lks32mc09x_flash.h, 282
 FLASH_SECTOR_111
 lks32mc09x_flash.h, 282
 FLASH_SECTOR_112
 lks32mc09x_flash.h, 282
 FLASH_SECTOR_113
 lks32mc09x_flash.h, 282
 FLASH_SECTOR_114
 lks32mc09x_flash.h, 282
 FLASH_SECTOR_115
 lks32mc09x_flash.h, 282
 FLASH_SECTOR_116
 lks32mc09x_flash.h, 283
 FLASH_SECTOR_117
 lks32mc09x_flash.h, 283
 FLASH_SECTOR_118
 lks32mc09x_flash.h, 283
 FLASH_SECTOR_119

lks32mc09x_flash.h, 283
FLASH_SECTOR_12
lks32mc09x_flash.h, 283
FLASH_SECTOR_120
lks32mc09x_flash.h, 283
FLASH_SECTOR_121
lks32mc09x_flash.h, 283
FLASH_SECTOR_122
lks32mc09x_flash.h, 283
FLASH_SECTOR_123
lks32mc09x_flash.h, 284
FLASH_SECTOR_124
lks32mc09x_flash.h, 284
FLASH_SECTOR_125
lks32mc09x_flash.h, 284
FLASH_SECTOR_126
lks32mc09x_flash.h, 284
FLASH_SECTOR_127
lks32mc09x_flash.h, 284
FLASH_SECTOR_128
lks32mc09x_flash.h, 284
FLASH_SECTOR_129
lks32mc09x_flash.h, 284
FLASH_SECTOR_13
lks32mc09x_flash.h, 284
FLASH_SECTOR_130
lks32mc09x_flash.h, 285
FLASH_SECTOR_131
lks32mc09x_flash.h, 285
FLASH_SECTOR_132
lks32mc09x_flash.h, 285
FLASH_SECTOR_133
lks32mc09x_flash.h, 285
FLASH_SECTOR_134
lks32mc09x_flash.h, 285
FLASH_SECTOR_135
lks32mc09x_flash.h, 285
FLASH_SECTOR_136
lks32mc09x_flash.h, 285
FLASH_SECTOR_137
lks32mc09x_flash.h, 285
FLASH_SECTOR_138
lks32mc09x_flash.h, 286
FLASH_SECTOR_139
lks32mc09x_flash.h, 286
FLASH_SECTOR_14
lks32mc09x_flash.h, 286
FLASH_SECTOR_140
lks32mc09x_flash.h, 286
FLASH_SECTOR_141
lks32mc09x_flash.h, 286
FLASH_SECTOR_142
lks32mc09x_flash.h, 286
FLASH_SECTOR_143
lks32mc09x_flash.h, 286
FLASH_SECTOR_144
lks32mc09x_flash.h, 286
FLASH_SECTOR_145

lks32mc09x_flash.h, 287
FLASH_SECTOR_146
lks32mc09x_flash.h, 287
FLASH_SECTOR_147
lks32mc09x_flash.h, 287
FLASH_SECTOR_148
lks32mc09x_flash.h, 287
FLASH_SECTOR_149
lks32mc09x_flash.h, 287
FLASH_SECTOR_15
lks32mc09x_flash.h, 287
FLASH_SECTOR_150
lks32mc09x_flash.h, 287
FLASH_SECTOR_151
lks32mc09x_flash.h, 287
FLASH_SECTOR_152
lks32mc09x_flash.h, 288
FLASH_SECTOR_153
lks32mc09x_flash.h, 288
FLASH_SECTOR_154
lks32mc09x_flash.h, 288
FLASH_SECTOR_155
lks32mc09x_flash.h, 288
FLASH_SECTOR_156
lks32mc09x_flash.h, 288
FLASH_SECTOR_157
lks32mc09x_flash.h, 288
FLASH_SECTOR_158
lks32mc09x_flash.h, 288
FLASH_SECTOR_159
lks32mc09x_flash.h, 288
FLASH_SECTOR_16
lks32mc09x_flash.h, 289
FLASH_SECTOR_160
lks32mc09x_flash.h, 289
FLASH_SECTOR_161
lks32mc09x_flash.h, 289
FLASH_SECTOR_162
lks32mc09x_flash.h, 289
FLASH_SECTOR_163
lks32mc09x_flash.h, 289
FLASH_SECTOR_164
lks32mc09x_flash.h, 289
FLASH_SECTOR_165
lks32mc09x_flash.h, 289
FLASH_SECTOR_166
lks32mc09x_flash.h, 289
FLASH_SECTOR_167
lks32mc09x_flash.h, 290
FLASH_SECTOR_168
lks32mc09x_flash.h, 290
FLASH_SECTOR_169
lks32mc09x_flash.h, 290
FLASH_SECTOR_17
lks32mc09x_flash.h, 290
FLASH_SECTOR_170
lks32mc09x_flash.h, 290
FLASH_SECTOR_171

lks32mc09x_flash.h, [290](#)
FLASH_SECTOR_172
lks32mc09x_flash.h, [290](#)
FLASH_SECTOR_173
lks32mc09x_flash.h, [290](#)
FLASH_SECTOR_174
lks32mc09x_flash.h, [291](#)
FLASH_SECTOR_175
lks32mc09x_flash.h, [291](#)
FLASH_SECTOR_176
lks32mc09x_flash.h, [291](#)
FLASH_SECTOR_177
lks32mc09x_flash.h, [291](#)
FLASH_SECTOR_178
lks32mc09x_flash.h, [291](#)
FLASH_SECTOR_179
lks32mc09x_flash.h, [291](#)
FLASH_SECTOR_18
lks32mc09x_flash.h, [291](#)
FLASH_SECTOR_180
lks32mc09x_flash.h, [291](#)
FLASH_SECTOR_181
lks32mc09x_flash.h, [292](#)
FLASH_SECTOR_182
lks32mc09x_flash.h, [292](#)
FLASH_SECTOR_183
lks32mc09x_flash.h, [292](#)
FLASH_SECTOR_184
lks32mc09x_flash.h, [292](#)
FLASH_SECTOR_185
lks32mc09x_flash.h, [292](#)
FLASH_SECTOR_186
lks32mc09x_flash.h, [292](#)
FLASH_SECTOR_187
lks32mc09x_flash.h, [292](#)
FLASH_SECTOR_188
lks32mc09x_flash.h, [292](#)
FLASH_SECTOR_189
lks32mc09x_flash.h, [293](#)
FLASH_SECTOR_19
lks32mc09x_flash.h, [293](#)
FLASH_SECTOR_190
lks32mc09x_flash.h, [293](#)
FLASH_SECTOR_191
lks32mc09x_flash.h, [293](#)
FLASH_SECTOR_192
lks32mc09x_flash.h, [293](#)
FLASH_SECTOR_193
lks32mc09x_flash.h, [293](#)
FLASH_SECTOR_194
lks32mc09x_flash.h, [293](#)
FLASH_SECTOR_195
lks32mc09x_flash.h, [293](#)
FLASH_SECTOR_196
lks32mc09x_flash.h, [294](#)
FLASH_SECTOR_197
lks32mc09x_flash.h, [294](#)
FLASH_SECTOR_198
lks32mc09x_flash.h, [294](#)
FLASH_SECTOR_199
lks32mc09x_flash.h, [294](#)
FLASH_SECTOR_2
lks32mc09x_flash.h, [294](#)
FLASH_SECTOR_20
lks32mc09x_flash.h, [294](#)
FLASH_SECTOR_200
lks32mc09x_flash.h, [294](#)
FLASH_SECTOR_201
lks32mc09x_flash.h, [294](#)
FLASH_SECTOR_202
lks32mc09x_flash.h, [295](#)
FLASH_SECTOR_203
lks32mc09x_flash.h, [295](#)
FLASH_SECTOR_204
lks32mc09x_flash.h, [295](#)
FLASH_SECTOR_205
lks32mc09x_flash.h, [295](#)
FLASH_SECTOR_206
lks32mc09x_flash.h, [295](#)
FLASH_SECTOR_207
lks32mc09x_flash.h, [295](#)
FLASH_SECTOR_208
lks32mc09x_flash.h, [295](#)
FLASH_SECTOR_209
lks32mc09x_flash.h, [295](#)
FLASH_SECTOR_21
lks32mc09x_flash.h, [296](#)
FLASH_SECTOR_210
lks32mc09x_flash.h, [296](#)
FLASH_SECTOR_211
lks32mc09x_flash.h, [296](#)
FLASH_SECTOR_212
lks32mc09x_flash.h, [296](#)
FLASH_SECTOR_213
lks32mc09x_flash.h, [296](#)
FLASH_SECTOR_214
lks32mc09x_flash.h, [296](#)
FLASH_SECTOR_215
lks32mc09x_flash.h, [296](#)
FLASH_SECTOR_216
lks32mc09x_flash.h, [296](#)
FLASH_SECTOR_217
lks32mc09x_flash.h, [297](#)
FLASH_SECTOR_218
lks32mc09x_flash.h, [297](#)
FLASH_SECTOR_219
lks32mc09x_flash.h, [297](#)
FLASH_SECTOR_22
lks32mc09x_flash.h, [297](#)
FLASH_SECTOR_220
lks32mc09x_flash.h, [297](#)
FLASH_SECTOR_221
lks32mc09x_flash.h, [297](#)
FLASH_SECTOR_222
lks32mc09x_flash.h, [297](#)
FLASH_SECTOR_223

lks32mc09x_flash.h, 297
FLASH_SECTOR_224
lks32mc09x_flash.h, 298
FLASH_SECTOR_225
lks32mc09x_flash.h, 298
FLASH_SECTOR_226
lks32mc09x_flash.h, 298
FLASH_SECTOR_227
lks32mc09x_flash.h, 298
FLASH_SECTOR_228
lks32mc09x_flash.h, 298
FLASH_SECTOR_229
lks32mc09x_flash.h, 298
FLASH_SECTOR_23
lks32mc09x_flash.h, 298
FLASH_SECTOR_230
lks32mc09x_flash.h, 298
FLASH_SECTOR_231
lks32mc09x_flash.h, 299
FLASH_SECTOR_232
lks32mc09x_flash.h, 299
FLASH_SECTOR_233
lks32mc09x_flash.h, 299
FLASH_SECTOR_234
lks32mc09x_flash.h, 299
FLASH_SECTOR_235
lks32mc09x_flash.h, 299
FLASH_SECTOR_236
lks32mc09x_flash.h, 299
FLASH_SECTOR_237
lks32mc09x_flash.h, 299
FLASH_SECTOR_238
lks32mc09x_flash.h, 299
FLASH_SECTOR_239
lks32mc09x_flash.h, 300
FLASH_SECTOR_24
lks32mc09x_flash.h, 300
FLASH_SECTOR_240
lks32mc09x_flash.h, 300
FLASH_SECTOR_241
lks32mc09x_flash.h, 300
FLASH_SECTOR_242
lks32mc09x_flash.h, 300
FLASH_SECTOR_243
lks32mc09x_flash.h, 300
FLASH_SECTOR_244
lks32mc09x_flash.h, 300
FLASH_SECTOR_245
lks32mc09x_flash.h, 300
FLASH_SECTOR_246
lks32mc09x_flash.h, 301
FLASH_SECTOR_247
lks32mc09x_flash.h, 301
FLASH_SECTOR_248
lks32mc09x_flash.h, 301
FLASH_SECTOR_249
lks32mc09x_flash.h, 301
FLASH_SECTOR_25
lks32mc09x_flash.h, 301
FLASH_SECTOR_250
lks32mc09x_flash.h, 301
FLASH_SECTOR_251
lks32mc09x_flash.h, 301
FLASH_SECTOR_252
lks32mc09x_flash.h, 301
FLASH_SECTOR_253
lks32mc09x_flash.h, 302
FLASH_SECTOR_254
lks32mc09x_flash.h, 302
FLASH_SECTOR_255
lks32mc09x_flash.h, 302
FLASH_SECTOR_26
lks32mc09x_flash.h, 302
FLASH_SECTOR_27
lks32mc09x_flash.h, 302
FLASH_SECTOR_28
lks32mc09x_flash.h, 302
FLASH_SECTOR_29
lks32mc09x_flash.h, 302
FLASH_SECTOR_3
lks32mc09x_flash.h, 302
FLASH_SECTOR_30
lks32mc09x_flash.h, 303
FLASH_SECTOR_31
lks32mc09x_flash.h, 303
FLASH_SECTOR_32
lks32mc09x_flash.h, 303
FLASH_SECTOR_33
lks32mc09x_flash.h, 303
FLASH_SECTOR_34
lks32mc09x_flash.h, 303
FLASH_SECTOR_35
lks32mc09x_flash.h, 303
FLASH_SECTOR_36
lks32mc09x_flash.h, 303
FLASH_SECTOR_37
lks32mc09x_flash.h, 303
FLASH_SECTOR_38
lks32mc09x_flash.h, 304
FLASH_SECTOR_39
lks32mc09x_flash.h, 304
FLASH_SECTOR_4
lks32mc09x_flash.h, 304
FLASH_SECTOR_40
lks32mc09x_flash.h, 304
FLASH_SECTOR_41
lks32mc09x_flash.h, 304
FLASH_SECTOR_42
lks32mc09x_flash.h, 304
FLASH_SECTOR_43
lks32mc09x_flash.h, 304
FLASH_SECTOR_44
lks32mc09x_flash.h, 304
FLASH_SECTOR_45
lks32mc09x_flash.h, 305
FLASH_SECTOR_46

lks32mc09x_flash.h, 305
FLASH_SECTOR_47
lks32mc09x_flash.h, 305
FLASH_SECTOR_48
lks32mc09x_flash.h, 305
FLASH_SECTOR_49
lks32mc09x_flash.h, 305
FLASH_SECTOR_5
lks32mc09x_flash.h, 305
FLASH_SECTOR_50
lks32mc09x_flash.h, 305
FLASH_SECTOR_51
lks32mc09x_flash.h, 305
FLASH_SECTOR_52
lks32mc09x_flash.h, 306
FLASH_SECTOR_53
lks32mc09x_flash.h, 306
FLASH_SECTOR_54
lks32mc09x_flash.h, 306
FLASH_SECTOR_55
lks32mc09x_flash.h, 306
FLASH_SECTOR_56
lks32mc09x_flash.h, 306
FLASH_SECTOR_57
lks32mc09x_flash.h, 306
FLASH_SECTOR_58
lks32mc09x_flash.h, 306
FLASH_SECTOR_59
lks32mc09x_flash.h, 306
FLASH_SECTOR_6
lks32mc09x_flash.h, 307
FLASH_SECTOR_60
lks32mc09x_flash.h, 307
FLASH_SECTOR_61
lks32mc09x_flash.h, 307
FLASH_SECTOR_62
lks32mc09x_flash.h, 307
FLASH_SECTOR_63
lks32mc09x_flash.h, 307
FLASH_SECTOR_64
lks32mc09x_flash.h, 307
FLASH_SECTOR_65
lks32mc09x_flash.h, 307
FLASH_SECTOR_66
lks32mc09x_flash.h, 307
FLASH_SECTOR_67
lks32mc09x_flash.h, 308
FLASH_SECTOR_68
lks32mc09x_flash.h, 308
FLASH_SECTOR_69
lks32mc09x_flash.h, 308
FLASH_SECTOR_7
lks32mc09x_flash.h, 308
FLASH_SECTOR_70
lks32mc09x_flash.h, 308
FLASH_SECTOR_71
lks32mc09x_flash.h, 308
FLASH_SECTOR_72
lks32mc09x_flash.h, 308
FLASH_SECTOR_73
lks32mc09x_flash.h, 308
FLASH_SECTOR_74
lks32mc09x_flash.h, 309
FLASH_SECTOR_75
lks32mc09x_flash.h, 309
FLASH_SECTOR_76
lks32mc09x_flash.h, 309
FLASH_SECTOR_77
lks32mc09x_flash.h, 309
FLASH_SECTOR_78
lks32mc09x_flash.h, 309
FLASH_SECTOR_79
lks32mc09x_flash.h, 309
FLASH_SECTOR_8
lks32mc09x_flash.h, 309
FLASH_SECTOR_80
lks32mc09x_flash.h, 309
FLASH_SECTOR_81
lks32mc09x_flash.h, 310
FLASH_SECTOR_82
lks32mc09x_flash.h, 310
FLASH_SECTOR_83
lks32mc09x_flash.h, 310
FLASH_SECTOR_84
lks32mc09x_flash.h, 310
FLASH_SECTOR_85
lks32mc09x_flash.h, 310
FLASH_SECTOR_86
lks32mc09x_flash.h, 310
FLASH_SECTOR_87
lks32mc09x_flash.h, 310
FLASH_SECTOR_88
lks32mc09x_flash.h, 310
FLASH_SECTOR_89
lks32mc09x_flash.h, 311
FLASH_SECTOR_9
lks32mc09x_flash.h, 311
FLASH_SECTOR_90
lks32mc09x_flash.h, 311
FLASH_SECTOR_91
lks32mc09x_flash.h, 311
FLASH_SECTOR_92
lks32mc09x_flash.h, 311
FLASH_SECTOR_93
lks32mc09x_flash.h, 311
FLASH_SECTOR_94
lks32mc09x_flash.h, 311
FLASH_SECTOR_95
lks32mc09x_flash.h, 311
FLASH_SECTOR_96
lks32mc09x_flash.h, 312
FLASH_SECTOR_97
lks32mc09x_flash.h, 312
FLASH_SECTOR_98
lks32mc09x_flash.h, 312
FLASH_SECTOR_99

lks32mc09x_flash.h, 312
 FLASH_SECTOR_END_128K
 lks32mc09x_flash.h, 312
 FLASH_SECTOR_END_32K
 lks32mc09x_flash.h, 312
 FLASH_SECTOR_END_64K
 lks32mc09x_flash.h, 312
 FLASH_TypeDef, 46
 ADDR, 47
 CFG, 47
 EPSM, 48
 EPSN, 48
 ERASE, 48
 LIB_ADDR, 48
 NADDR, 49
 NCFG, 49
 NERASE, 49
 NKEY, 49
 NRDATA, 49
 NWDATA, 50
 PROTECT, 50
 RDATA, 50
 READY, 50
 RESERVED, 50
 SIZE, 50
 WDATA, 51
 FLASH_Write
 lks32mc09x_flash.c, 614
 lks32mc09x_flash.h, 314
 FLASH_WriteNvr
 lks32mc09x_flash.c, 614
 lks32mc09x_flash.h, 314
 FLSE
 SYS_TypeDef, 104
 FLSP
 SYS_TypeDef, 105
 FLT
 MCPWM_TypeDef, 78
 TIMER_TimerInitTypeDef, 114
 TIMER_TypeDef, 118
 FLT_DIV
 MCPWM_InitTypeDef, 71
 FN
 CLU_InitTypeDef, 20
 CLU_TypeDef, 22
 GAIN
 ADC_InitTypeDef, 6
 ADC_TypeDef, 14
 GATE_EN
 TIMER_TimerInitTypeDef, 114
 GEN
 ADC_InitTypeDef, 6
 ADC_TypeDef, 14
 GPIO0
 lks32mc09x_reg.h, 439
 GPIO1
 lks32mc09x_reg.h, 439
 GPIO2
 lks32mc09x_reg.h, 439
 GPIO_AF_ADC0_TRIGGER
 lks32mc09x_gpio.h, 325
 GPIO_AF_ADC1_TRIGGER
 lks32mc09x_gpio.h, 325
 GPIO_AF_AFE
 lks32mc09x_gpio.h, 326
 GPIO_AF_CAN
 lks32mc09x_gpio.h, 326
 GPIO_AF_CL
 lks32mc09x_gpio.h, 326
 GPIO_AF_CLKO
 lks32mc09x_gpio.h, 326
 GPIO_AF_CLU
 lks32mc09x_gpio.h, 326
 GPIO_AF_CMP_OUT
 lks32mc09x_gpio.h, 326
 GPIO_AF_GPIO
 lks32mc09x_gpio.h, 327
 GPIO_AF_HALL
 lks32mc09x_gpio.h, 327
 GPIO_AF_I2C
 lks32mc09x_gpio.h, 327
 GPIO_AF_MCPWM
 lks32mc09x_gpio.h, 327
 GPIO_AF_QEP0
 lks32mc09x_gpio.h, 327
 GPIO_AF_SIF
 lks32mc09x_gpio.h, 327
 GPIO_AF_SPI
 lks32mc09x_gpio.h, 328
 GPIO_AF_TIMER0
 lks32mc09x_gpio.h, 328
 GPIO_AF_TIMER1
 lks32mc09x_gpio.h, 328
 GPIO_AF_TIMER2
 lks32mc09x_gpio.h, 328
 GPIO_AF_UART
 lks32mc09x_gpio.h, 328
 GPIO_Bit_Action
 lks32mc09x_gpio.h, 334
 GPIO_Bit_RESET
 lks32mc09x_gpio.h, 334
 GPIO_Bit_SET
 lks32mc09x_gpio.h, 334
 GPIO_Config
 lks32mc09x_gpio.c, 617
 lks32mc09x_gpio.h, 335
 GPIO_Disable
 lks32mc09x_gpio.c, 618
 lks32mc09x_gpio.h, 336
 GPIO_Enable
 lks32mc09x_gpio.c, 618
 lks32mc09x_gpio.h, 336
 GPIO_Init
 lks32mc09x_gpio.c, 619
 lks32mc09x_gpio.h, 336
 GPIO_InitTypeDef, 51

- GPIO_Mode, 51
- GPIO_PFLT, 51
- GPIO_Pin, 52
- GPIO_PODena, 52
- GPIO_PuPd, 52
- GPIO_LockRegister
 - lks32mc09x_gpio.c, 620
 - lks32mc09x_gpio.h, 337
- GPIO_Mode
 - GPIO_InitTypeDef, 51
- GPIO_Mode_ANA
 - lks32mc09x_gpio.h, 335
- GPIO_Mode_IN
 - lks32mc09x_gpio.h, 335
- GPIO_Mode_IO
 - lks32mc09x_gpio.h, 335
- GPIO_Mode_OUT
 - lks32mc09x_gpio.h, 335
- GPIO_Mode_TypeDef
 - lks32mc09x_gpio.h, 334
- GPIO_PFLT
 - GPIO_InitTypeDef, 51
- GPIO_Pin
 - GPIO_InitTypeDef, 52
- GPIO_Pin_0
 - lks32mc09x_gpio.h, 328
- GPIO_Pin_1
 - lks32mc09x_gpio.h, 329
- GPIO_Pin_10
 - lks32mc09x_gpio.h, 329
- GPIO_Pin_11
 - lks32mc09x_gpio.h, 329
- GPIO_Pin_12
 - lks32mc09x_gpio.h, 329
- GPIO_Pin_13
 - lks32mc09x_gpio.h, 329
- GPIO_Pin_14
 - lks32mc09x_gpio.h, 329
- GPIO_Pin_15
 - lks32mc09x_gpio.h, 330
- GPIO_Pin_2
 - lks32mc09x_gpio.h, 330
- GPIO_Pin_3
 - lks32mc09x_gpio.h, 330
- GPIO_Pin_4
 - lks32mc09x_gpio.h, 330
- GPIO_Pin_5
 - lks32mc09x_gpio.h, 330
- GPIO_Pin_6
 - lks32mc09x_gpio.h, 330
- GPIO_Pin_7
 - lks32mc09x_gpio.h, 331
- GPIO_Pin_8
 - lks32mc09x_gpio.h, 331
- GPIO_Pin_9
 - lks32mc09x_gpio.h, 331
- GPIO_Pin_ALL
 - lks32mc09x_gpio.h, 331
- GPIO_Pin_NONE
 - lks32mc09x_gpio.h, 331
- GPIO_PinAFConfig
 - lks32mc09x_gpio.c, 620
 - lks32mc09x_gpio.h, 338
- GPIO_PinSource_0
 - lks32mc09x_gpio.h, 331
- GPIO_PinSource_1
 - lks32mc09x_gpio.h, 332
- GPIO_PinSource_10
 - lks32mc09x_gpio.h, 332
- GPIO_PinSource_11
 - lks32mc09x_gpio.h, 332
- GPIO_PinSource_12
 - lks32mc09x_gpio.h, 332
- GPIO_PinSource_13
 - lks32mc09x_gpio.h, 332
- GPIO_PinSource_14
 - lks32mc09x_gpio.h, 332
- GPIO_PinSource_15
 - lks32mc09x_gpio.h, 333
- GPIO_PinSource_2
 - lks32mc09x_gpio.h, 333
- GPIO_PinSource_3
 - lks32mc09x_gpio.h, 333
- GPIO_PinSource_4
 - lks32mc09x_gpio.h, 333
- GPIO_PinSource_5
 - lks32mc09x_gpio.h, 333
- GPIO_PinSource_6
 - lks32mc09x_gpio.h, 333
- GPIO_PinSource_7
 - lks32mc09x_gpio.h, 334
- GPIO_PinSource_8
 - lks32mc09x_gpio.h, 334
- GPIO_PinSource_9
 - lks32mc09x_gpio.h, 334
- GPIO_PODena
 - GPIO_InitTypeDef, 52
- GPIO_PuPd
 - GPIO_InitTypeDef, 52
- GPIO_PuPd_NOPULL
 - lks32mc09x_gpio.h, 335
- GPIO_PuPd_TypeDef
 - lks32mc09x_gpio.h, 335
- GPIO_PuPd_UP
 - lks32mc09x_gpio.h, 335
- GPIO_ReadInputData
 - lks32mc09x_gpio.c, 621
 - lks32mc09x_gpio.h, 338
- GPIO_ReadInputDataBit
 - lks32mc09x_gpio.c, 621
 - lks32mc09x_gpio.h, 338
- GPIO_ReadOutputData
 - lks32mc09x_gpio.c, 621
 - lks32mc09x_gpio.h, 339
- GPIO_ReadOutputDataBit
 - lks32mc09x_gpio.c, 622

- lks32mc09x_gpio.h, 339
- GPIO_Reset
 - lks32mc09x_gpio.c, 622
 - lks32mc09x_gpio.h, 340
- GPIO_ResetBits
 - lks32mc09x_gpio.c, 623
 - lks32mc09x_gpio.h, 340
- GPIO_SetBits
 - lks32mc09x_gpio.c, 623
 - lks32mc09x_gpio.h, 340
- GPIO_StructInit
 - lks32mc09x_gpio.c, 623
 - lks32mc09x_gpio.h, 341
- GPIO_ToggleBits
 - lks32mc09x_gpio.c, 624
 - lks32mc09x_gpio.h, 341
- GPIO_TypeDef, 52
 - BRR, 54
 - BSRR, 54
 - F3210, 54
 - F7654, 55
 - FBA98, 55
 - FFEDC, 55
 - PDI, 55
 - PDO, 55
 - PFLT, 55
 - PIE, 56
 - PODE, 56
 - POE, 56
 - PUE, 56
 - RESERVED, 57
- GPIO_UnlockRegister
 - lks32mc09x_gpio.c, 624
 - lks32mc09x_gpio.h, 341
- GPIO_Write
 - lks32mc09x_gpio.c, 624
 - lks32mc09x_gpio.h, 342
- GPIO_WriteBit
 - lks32mc09x_gpio.c, 625
 - lks32mc09x_gpio.h, 342
- HALL
 - lks32mc09x_reg.h, 440
- HALL0
 - lks32mc09x_reg.h, 440
- HALL_CAPTURE_EVENT
 - lks32mc09x_hall.h, 346
- HALL_Clear_IRQ
 - lks32mc09x_hall.c, 629
 - lks32mc09x_hall.h, 347
- HALL_CLK_DIV1
 - lks32mc09x_hall.h, 346
- HALL_CLK_DIV2
 - lks32mc09x_hall.h, 346
- HALL_CLK_DIV4
 - lks32mc09x_hall.h, 347
- HALL_CLK_DIV8
 - lks32mc09x_hall.h, 347
- HALL_Cmd
 - lks32mc09x_hall.c, 629
 - lks32mc09x_hall.h, 347
- HALL_Ena
 - HALL_InitTypeDef, 58
- HALL_GetCaptureValue
 - lks32mc09x_hall.c, 629
 - lks32mc09x_hall.h, 348
- HALL_GetCount
 - lks32mc09x_hall.c, 629
 - lks32mc09x_hall.h, 348
- HALL_GetFilterValue
 - lks32mc09x_hall.c, 629
 - lks32mc09x_hall.h, 348
- HALL_GetFlag
 - lks32mc09x_hall.c, 630
 - lks32mc09x_hall.h, 348
- HALL_GetIRQFlag
 - lks32mc09x_hall.c, 630
 - lks32mc09x_hall.h, 349
- HALL_Init
 - lks32mc09x_hall.c, 630
 - lks32mc09x_hall.h, 349
- HALL_InitTypeDef, 57
 - Capture_IRQ_Ena, 58
 - ClockDivision, 58
 - CountTH, 58
 - Filter75_Ena, 58
 - FilterLen, 58
 - HALL_Ena, 58
 - OverFlow_IRQ_Ena, 58
 - softIE, 59
- HALL_IsCaptureEvent
 - lks32mc09x_hall.c, 631
 - lks32mc09x_hall.h, 350
- HALL_IsOverFlowEvent
 - lks32mc09x_hall.c, 631
 - lks32mc09x_hall.h, 350
- HALL_OVERFLOW_EVENT
 - lks32mc09x_hall.h, 347
- HALL_SoftTrigger
 - lks32mc09x_hall.c, 631
 - lks32mc09x_hall.h, 350
- HALL_StructInit
 - lks32mc09x_hall.c, 632
 - lks32mc09x_hall.h, 350
- HALL_TypeDef, 59
 - CFG, 60
 - CNT, 60
 - INFO, 60
 - TH, 60
 - WIDTH, 60
- HALT_PRT
 - TIMER_TimerInitTypeDef, 114
- HTH
 - ADC_InitTypeDef, 6
 - ADC_TypeDef, 14
- HYS
 - CMP_InitTypeDef, 24

- I2C
 - lks32mc09x_reg.h, 440
- I2C0
 - lks32mc09x_reg.h, 440
- I2C_BUS_STATE_ADDR_DATA
 - lks32mc09x_i2c.h, 354
- I2C_BUS_STATE_Done
 - lks32mc09x_i2c.h, 354
- I2C_BUS_STATE_LOST_ARB
 - lks32mc09x_i2c.h, 354
- I2C_BUS_STATE_RX_ACK
 - lks32mc09x_i2c.h, 354
- I2C_BUS_STATE_STOP_EVT
 - lks32mc09x_i2c.h, 354
- I2C_BUS_STATE_STT_ERR
 - lks32mc09x_i2c.h, 354
- I2c_ClearBusState
 - lks32mc09x_i2c.c, 635
 - lks32mc09x_i2c.h, 355
- I2C_ClearIRQFlag
 - lks32mc09x_i2c.c, 635
 - lks32mc09x_i2c.h, 356
- I2C_Disable
 - lks32mc09x_i2c.c, 636
 - lks32mc09x_i2c.h, 356
- I2C_Enable
 - lks32mc09x_i2c.c, 636
 - lks32mc09x_i2c.h, 356
- I2C_GetFlag
 - lks32mc09x_i2c.c, 637
 - lks32mc09x_i2c.h, 357
- I2C_GetIRQFlag
 - lks32mc09x_i2c.c, 637
 - lks32mc09x_i2c.h, 358
- I2C_IF_ADDR_CMP
 - lks32mc09x_i2c.h, 355
- I2C_IF_BUS_ERR
 - lks32mc09x_i2c.h, 355
- I2C_IF_NACK
 - lks32mc09x_i2c.h, 355
- I2C_IF_STOP
 - lks32mc09x_i2c.h, 355
- I2C_IF_TC
 - lks32mc09x_i2c.h, 355
- I2C_Init
 - lks32mc09x_i2c.c, 638
 - lks32mc09x_i2c.h, 358
- I2C_InitTypeDef, 61
 - ADDR, 61
 - ADRCMP, 61
 - BaudRate, 62
 - BURST_ADDR_CMP, 62
 - BURST_NACK, 62
 - BUS_ERR_IE, 62
 - BUSRT_EN, 62
 - IE, 62
 - MST_MODE, 63
 - SLV_MODE, 63
 - STOP_IE, 63
 - TC_IE, 63
- I2c_ReadBusState
 - lks32mc09x_i2c.c, 638
 - lks32mc09x_i2c.h, 359
- I2C_ReadData
 - lks32mc09x_i2c.c, 639
 - lks32mc09x_i2c.h, 359
- I2C_SendData
 - lks32mc09x_i2c.c, 639
 - lks32mc09x_i2c.h, 360
- I2C_SetBaudRate
 - lks32mc09x_i2c.c, 639
 - lks32mc09x_i2c.h, 360
- I2C_StructInit
 - lks32mc09x_i2c.c, 640
 - lks32mc09x_i2c.h, 361
- I2C_TableAck
 - lks32mc09x_i2c.c, 640
 - lks32mc09x_i2c.h, 361
- I2C_TableIrq
 - lks32mc09x_i2c.c, 641
 - lks32mc09x_i2c.h, 361
- I2C_TableRead
 - lks32mc09x_i2c.c, 641
 - lks32mc09x_i2c.h, 362
- I2C_TableSend
 - lks32mc09x_i2c.c, 641
 - lks32mc09x_i2c.h, 362
- I2C_TableState
 - lks32mc09x_i2c.c, 642
 - lks32mc09x_i2c.h, 363
- I2C_TableVariable
 - lks32mc09x_i2c.c, 642
- I2C_TypeDef, 63
 - ADDR, 64
 - BCR, 64
 - BSIZE, 64
 - CFG, 65
 - DATA, 65
 - MSCR, 65
 - SCR, 65
- IDLE_PRI
 - ADC_InitTypeDef, 7
- IE
 - ADC_InitTypeDef, 7
 - ADC_TypeDef, 15
 - CLU_TypeDef, 22
 - CMP_SingleTypeDef, 27
 - CMP_TypeDef, 29
 - DMAMODULE_TypeDef, 39
 - EXTI_TypeDef, 45
 - I2C_InitTypeDef, 62
 - MCPWM_InitTypeDef, 71
 - QEP_TypeDef, 88
 - SPI_TypeDef, 97
 - TIMER_TimerInitTypeDef, 114
 - TIMER_TypeDef, 118

UART_TypeDef, 124
 IE0
 MCPWM_TypeDef, 78
 IF
 ADC_TypeDef, 15
 CLU_TypeDef, 22
 CMP_TypeDef, 29
 DMAMODULE_TypeDef, 39
 EXTI_TypeDef, 45
 QEP_TypeDef, 88
 SYS_TypeDef, 105
 TIMER_TypeDef, 118
 UART_TypeDef, 125
 IF0
 MCPWM_TypeDef, 78
 iicmpval, 65
 nackflg, 66
 rxdata, 66
 rxdatap, 66
 state, 66
 txdata, 66
 txdatap, 66
 include/lks32mc09x_adc.h, 127, 147
 include/lks32mc09x_clu.h, 150, 178
 include/lks32mc09x_cmp.h, 181, 201
 include/lks32mc09x_crc.h, 203, 211
 include/lks32mc09x_dac.h, 212, 227
 include/lks32mc09x_dma.h, 228, 248
 include/lks32mc09x_dsp.h, 250, 254
 include/lks32mc09x_eeprom.h, 255, 263
 include/lks32mc09x_exti.h, 263, 273
 include/lks32mc09x_flash.h, 274, 315
 include/lks32mc09x_gpio.h, 319, 343
 include/lks32mc09x_hall.h, 344, 351
 include/lks32mc09x_i2c.h, 352, 363
 include/lks32mc09x_iwdg.h, 364, 367
 include/lks32mc09x_lib.h, 368
 include/lks32mc09x_mcpwm.h, 369, 408
 include/lks32mc09x_opa.h, 412, 422
 include/lks32mc09x_qep.h, 423, 432
 include/lks32mc09x_reg.h, 433, 445
 include/lks32mc09x_sif.h, 450, 455
 include/lks32mc09x_spi.h, 456, 462
 include/lks32mc09x_sys.h, 463, 479
 include/lks32mc09x_timer.h, 481, 489
 include/lks32mc09x_tmp.h, 491, 494
 include/lks32mc09x_trim.h, 494, 506
 include/lks32mc09x_uart.h, 508, 523
 include/lks32mc09x_wake.h, 525, 532
 INFO
 HALL_TypeDef, 60
 INIT
 CRC_InitTypeDef, 31
 CRC_TypeDef, 33
 IO
 TIMER_TypeDef, 118
 IO0
 MCPWM_InitTypeDef, 72
 IO01
 MCPWM_TypeDef, 79
 IO1
 MCPWM_InitTypeDef, 72
 IO2
 MCPWM_InitTypeDef, 72
 IO23
 MCPWM_TypeDef, 79
 IO3
 MCPWM_InitTypeDef, 72
 IO_CFG
 SYS_TypeDef, 105
 IO_SWITCH
 EEPROM_InitTypeDef, 43
 IO_WAKE_EN
 AON_TypeDef, 18
 IO_WAKE_POL
 AON_TypeDef, 18
 IOC
 UART_TypeDef, 125
 IRQ
 SIF_TypeDef, 93
 IRQ_EN
 DMA_InitTypeDef, 36
 IRQ_TRIG
 CMP_SingleTypeDef, 27
 IRQEna
 QEP_InitTypeDef, 86
 SPI_InitTypeDef, 96
 UART_InitTypeDef, 121
 IS1
 ADC_InitTypeDef, 7
 IWDG
 lks32mc09x_reg.h, 440
 IWDG0
 lks32mc09x_reg.h, 440
 IWDG_DISABLE
 lks32mc09x_iwdg.c, 647
 lks32mc09x_iwdg.h, 366
 IWDG_ENABLE
 lks32mc09x_iwdg.c, 647
 lks32mc09x_iwdg.h, 366
 IWDG_Feed
 lks32mc09x_iwdg.c, 647
 lks32mc09x_iwdg.h, 366
 IWDG_Init
 lks32mc09x_iwdg.c, 647
 lks32mc09x_iwdg.h, 366
 IWDG_InitTypeDef, 67
 DWK_EN, 67
 RTH, 67
 WDG_EN, 67
 WTH, 67
 IWDG_SECOND2IWDGCNT
 lks32mc09x_iwdg.h, 365
 IWDG_SECOND2IWDGCNT_SAT
 lks32mc09x_iwdg.h, 365
 IWDG_StrutInit

- lks32mc09x_iwdg.c, 648
- lks32mc09x_iwdg.h, 367
- IWDG_TypeDef, 68
 - CFG, 68
 - CLR, 68
 - CNT, 69
 - PSW, 69
 - RTH, 69
 - WTH, 69
- IWDG_Update
 - lks32mc09x_iwdg.c, 648
 - lks32mc09x_iwdg.h, 367
- LBDL
 - UART_InitTypeDef, 121
- LIB_ADDR
 - FLASH_TypeDef, 48
- LIN_EN
 - UART_InitTypeDef, 121
- lks32mc09x_adc.c
 - ADC_ClearIRQFlag, 534
 - ADC_Disable, 534
 - ADC_Enable, 534
 - ADC_GetConversionValue, 535
 - ADC_GetConversionValueS32, 535
 - ADC_GetIRQFlag, 536
 - ADC_Init, 536
 - ADC_Reset, 537
 - ADC_SetChn, 537
 - ADC_SetGain, 538
 - ADC_SetRange, 538
 - ADC_SoftIdleTrig, 539
 - ADC_SoftTrig, 539
 - ADC_StructInit, 539
- lks32mc09x_adc.h
 - ADC0_CHN_ADC01_CH10, 139
 - ADC0_CHN_ADC01_CH11, 139
 - ADC0_CHN_ADC01_CH4, 139
 - ADC0_CHN_ADC01_CH5, 139
 - ADC0_CHN_ADC01_CH6, 139
 - ADC0_CHN_ADC01_CH7, 139
 - ADC0_CHN_ADC01_CH12, 139
 - ADC0_CHN_ADC01_CH13, 139
 - ADC0_CHN_ADC01_CH8, 139
 - ADC0_CHN_AVSS, 139
 - ADC0_CHN_DAC, 139
 - ADC0_CHN_OPA0, 139
 - ADC0_CHN_OPA1, 139
 - ADC0_CHN_OPA2, 139
 - ADC0_CHN_OPA3, 139
 - ADC0_CHN_REF24, 139
 - ADC1_CHN_ADC01_CH10, 139
 - ADC1_CHN_ADC01_CH11, 139
 - ADC1_CHN_ADC01_CH4, 139
 - ADC1_CHN_ADC01_CH5, 139
 - ADC1_CHN_ADC01_CH6, 139
 - ADC1_CHN_ADC01_CH7, 139
 - ADC1_CHN_ADC01_CH12, 140
 - ADC1_CHN_ADC01_CH13, 140
 - ADC1_CHN_ADC1_CH8, 139
 - ADC1_CHN_ADC1_CH9, 139
 - ADC1_CHN_AVDD, 140
 - ADC1_CHN_OPA0, 139
 - ADC1_CHN_OPA1, 139
 - ADC1_CHN_OPA2, 139
 - ADC1_CHN_OPA3, 139
 - ADC1_CHN_TMP, 140
 - ADC_CHN_0, 139
 - ADC_CHN_1, 139
 - ADC_CHN_10, 139
 - ADC_CHN_11, 139
 - ADC_CHN_12, 139
 - ADC_CHN_13, 139
 - ADC_CHN_14, 139
 - ADC_CHN_15, 139
 - ADC_CHN_2, 139
 - ADC_CHN_3, 139
 - ADC_CHN_4, 139
 - ADC_CHN_5, 139
 - ADC_CHN_6, 139
 - ADC_CHN_7, 139
 - ADC_CHN_8, 139
 - ADC_CHN_9, 139
 - ADC_CHN_x, 138
 - ADC_ClearIRQFlag, 141
 - ADC_DAT_0, 140
 - ADC_DAT_1, 140
 - ADC_DAT_10, 140
 - ADC_DAT_11, 140
 - ADC_DAT_2, 140
 - ADC_DAT_3, 140
 - ADC_DAT_4, 140
 - ADC_DAT_5, 140
 - ADC_DAT_6, 140
 - ADC_DAT_7, 140
 - ADC_DAT_8, 140
 - ADC_DAT_9, 140
 - ADC_DAT_x, 140
 - ADC_Disable, 141
 - ADC_Enable, 141
 - ADC_GAIN_3V6, 140
 - ADC_GAIN_7V2, 140
 - ADC_GAIN_x, 140
 - ADC_GetConversionValue, 142
 - ADC_GetConversionValueS32, 142
 - ADC_GetIRQFlag, 143
 - ADC_IE_AWD0, 131
 - ADC_IE_HERR, 131
 - ADC_IE_ISF, 131
 - ADC_IE_SERR, 131
 - ADC_IE_SF1, 132
 - ADC_IE_SF2, 132
 - ADC_IF_ALL, 132
 - ADC_IF_AWD0, 132
 - ADC_IF_HERR, 132
 - ADC_IF_ISF, 132
 - ADC_IF_SERR, 133

- ADC_IF_SF1, 133
- ADC_IF_SF2, 133
- ADC_Init, 144
- ADC_RANGE_3.3, 141
- ADC_RANGE_3.6, 141
- ADC_RANGE_EXTERNAL, 141
- ADC_RANGE_x, 140
- ADC_RE_AWD0, 133
- ADC_RE_HERR, 133
- ADC_RE_ISF, 133
- ADC_RE_SERR, 134
- ADC_RE_SF1, 134
- ADC_RE_SF2, 134
- ADC_Reset, 144
- ADC_SetChn, 145
- ADC_SetGain, 145
- ADC_SetRange, 146
- ADC_SoftIdleTrig, 146
- ADC_SoftTrig, 146
- ADC_StructInit, 147
- ADC_TRIG_CL_IT0, 134
- ADC_TRIG_CL_IT1, 134
- ADC_TRIG_CL_IT2, 134
- ADC_TRIG_CL_IT3, 135
- ADC_TRIG_CL_NT0, 135
- ADC_TRIG_CL_NT1, 135
- ADC_TRIG_CL_NT2, 135
- ADC_TRIG_CL_NT3, 135
- ADC_TRIG_MCPWM0_T0, 135
- ADC_TRIG_MCPWM0_T0_IEN, 136
- ADC_TRIG_MCPWM0_T1, 136
- ADC_TRIG_MCPWM0_T1_IEN, 136
- ADC_TRIG_MCPWM0_T2, 136
- ADC_TRIG_MCPWM0_T2_IEN, 136
- ADC_TRIG_MCPWM0_T3, 136
- ADC_TRIG_MCPWM0_T3_IEN, 137
- ADC_TRIG_SOFT, 137
- ADC_TRIG_TIMER0_CMP0, 137
- ADC_TRIG_TIMER0_CMP0_IEN, 137
- ADC_TRIG_TIMER0_CMP1, 137
- ADC_TRIG_TIMER0_CMP1_IEN, 137
- ADC_TRIG_TIMER1_CMP0, 138
- ADC_TRIG_TIMER1_CMP1, 138
- ADC_TRIG_TIMER2_CMP0_IEN, 138
- ADC_TRIG_TIMER2_CMP1_IEN, 138
- lks32mc09x_clu.c
 - CLU_Disable, 544
 - CLU_Enable, 545
 - CLU_Init, 545
 - CLU_ReadOutValue, 546
 - CLU_StructInit, 546
- lks32mc09x_clu.h
 - CLU_Disable, 176
 - CLU_Enable, 176
 - CLU_FN_0, 154
 - CLU_FN_1, 154
 - CLU_FN_A, 155
 - CLU_FN_A_B, 155
 - CLU_FN_A_B_C, 155
 - CLU_FN_A_B_NOTC, 155
 - CLU_FN_A_C, 155
 - CLU_FN_A_NOTB, 155
 - CLU_FN_A_NOTB_C, 156
 - CLU_FN_A_NOTB_NOTC, 156
 - CLU_FN_A_NOTC, 156
 - CLU_FN_B, 156
 - CLU_FN_B_C, 156
 - CLU_FN_B_NOTC, 156
 - CLU_FN_C, 157
 - CLU_FN_NOTA, 157
 - CLU_FN_NOTA_B, 157
 - CLU_FN_NOTA_B_C, 157
 - CLU_FN_NOTA_B_NOTC, 157
 - CLU_FN_NOTA_C, 157
 - CLU_FN_NOTA_NOTB, 158
 - CLU_FN_NOTA_NOTB_C, 158
 - CLU_FN_NOTA_NOTB_NOTC, 158
 - CLU_FN_NOTA_NOTC, 158
 - CLU_FN_NOTB, 158
 - CLU_FN_NOTB_C, 158
 - CLU_FN_NOTB_NOTC, 159
 - CLU_FN_NOTC, 159
 - CLU_Init, 177
 - CLU_MODEL_CLU0, 159
 - CLU_MODEL_CLU1, 159
 - CLU_MODEL_CLU2, 159
 - CLU_MODEL_CLU3, 159
 - CLU_MXA_CLU0_CLU0OUT, 160
 - CLU_MXA_CLU0_CLU1OUT, 160
 - CLU_MXA_CLU0_CLU2OUT, 160
 - CLU_MXA_CLU0_CLU3OUT, 160
 - CLU_MXA_CLU0_P0.0, 160
 - CLU_MXA_CLU0_P0.12, 160
 - CLU_MXA_CLU0_P0.14, 160
 - CLU_MXA_CLU0_P0.3, 161
 - CLU_MXA_CLU0_P0.5, 161
 - CLU_MXA_CLU0_P0.7, 161
 - CLU_MXA_CLU0_P1.0, 161
 - CLU_MXA_CLU0_P2.12, 161
 - CLU_MXA_CLU0_PWM0, 161
 - CLU_MXA_CLU0_PWM1, 161
 - CLU_MXA_CLU0_TIMER0ch0ovfl, 161
 - CLU_MXA_CLU0_TIMER1ch0ovfl, 162
 - CLU_MXA_CLU1_CLU0OUT, 162
 - CLU_MXA_CLU1_CLU1OUT, 162
 - CLU_MXA_CLU1_CLU2OUT, 162
 - CLU_MXA_CLU1_CLU3OUT, 162
 - CLU_MXA_CLU1_P0.0, 162
 - CLU_MXA_CLU1_P0.11, 162
 - CLU_MXA_CLU1_P0.14, 162
 - CLU_MXA_CLU1_P0.15, 163
 - CLU_MXA_CLU1_P0.2, 163
 - CLU_MXA_CLU1_P0.5, 163
 - CLU_MXA_CLU1_P0.6, 163
 - CLU_MXA_CLU1_P2.12, 163
 - CLU_MXA_CLU1_PWM2, 163

CLU_MXA_CLU1_PWM3, 163
 CLU_MXA_CLU1_TIMER0ch1ovfl, 163
 CLU_MXA_CLU1_TIMER1ch1ovfl, 164
 CLU_MXA_CLU2_CLU0OUT, 164
 CLU_MXA_CLU2_CLU1OUT, 164
 CLU_MXA_CLU2_CLU2OUT, 164
 CLU_MXA_CLU2_CLU3OUT, 164
 CLU_MXA_CLU2_P1.10, 164
 CLU_MXA_CLU2_P1.13, 164
 CLU_MXA_CLU2_P1.15, 164
 CLU_MXA_CLU2_P2.1, 165
 CLU_MXA_CLU2_P2.14, 165
 CLU_MXA_CLU2_P2.4, 165
 CLU_MXA_CLU2_P2.6, 165
 CLU_MXA_CLU2_P2.8, 165
 CLU_MXA_CLU2_PWM0, 165
 CLU_MXA_CLU2_PWM2, 165
 CLU_MXA_CLU2_TIMER2ch0ovfl, 165
 CLU_MXA_CLU2_TIMER3ch0ovfl, 166
 CLU_MXA_CLU3_CLU0OUT, 166
 CLU_MXA_CLU3_CLU1OUT, 166
 CLU_MXA_CLU3_CLU2OUT, 166
 CLU_MXA_CLU3_CLU3OUT, 166
 CLU_MXA_CLU3_P1.15, 166
 CLU_MXA_CLU3_P2.0, 166
 CLU_MXA_CLU3_P2.1, 166
 CLU_MXA_CLU3_P2.13, 167
 CLU_MXA_CLU3_P2.3, 167
 CLU_MXA_CLU3_P2.6, 167
 CLU_MXA_CLU3_P2.7, 167
 CLU_MXA_CLU3_P2.8, 167
 CLU_MXA_CLU3_PWM1, 167
 CLU_MXA_CLU3_PWM3, 167
 CLU_MXA_CLU3_TIMER2ch1ovfl, 168
 CLU_MXA_CLU3_TIMER3ch1ovfl, 168
 CLU_MXB_CLU0_ADBUSY0, 168
 CLU_MXB_CLU0_CLU0OUT, 168
 CLU_MXB_CLU0_CLU1OUT, 168
 CLU_MXB_CLU0_CLU2OUT, 168
 CLU_MXB_CLU0_CLU3OUT, 168
 CLU_MXB_CLU0_CMP0_OUT, 168
 CLU_MXB_CLU0_P0.11, 169
 CLU_MXB_CLU0_P0.13, 169
 CLU_MXB_CLU0_P0.15, 169
 CLU_MXB_CLU0_P0.2, 169
 CLU_MXB_CLU0_P0.4, 169
 CLU_MXB_CLU0_P0.6, 169
 CLU_MXB_CLU0_P2.11, 169
 CLU_MXB_CLU0_P2.7, 169
 CLU_MXB_CLU0_PWM3, 170
 CLU_MXB_CLU0_SPIMOSI, 170
 CLU_MXB_CLU1_ADBUSY0, 170
 CLU_MXB_CLU1_CLU0OUT, 170
 CLU_MXB_CLU1_CLU1OUT, 170
 CLU_MXB_CLU1_CLU2OUT, 170
 CLU_MXB_CLU1_CLU3OUT, 170
 CLU_MXB_CLU1_CMP1_OUT, 170
 CLU_MXB_CLU1_P0.12, 171
 CLU_MXB_CLU1_P0.13, 171
 CLU_MXB_CLU1_P0.3, 171
 CLU_MXB_CLU1_P0.4, 171
 CLU_MXB_CLU1_P0.7, 171
 CLU_MXB_CLU1_P1.0, 171
 CLU_MXB_CLU1_P2.11, 171
 CLU_MXB_CLU1_P2.7, 171
 CLU_MXB_CLU1_PWM1, 172
 CLU_MXB_CLU1_SPIMISO, 172
 CLU_MXB_CLU2_CLU0OUT, 172
 CLU_MXB_CLU2_CLU1OUT, 172
 CLU_MXB_CLU2_CLU2OUT, 172
 CLU_MXB_CLU2_CLU3OUT, 172
 CLU_MXB_CLU2_I2CSDA, 172
 CLU_MXB_CLU2_P1.11, 172
 CLU_MXB_CLU2_P1.14, 173
 CLU_MXB_CLU2_P2.0, 173
 CLU_MXB_CLU2_P2.13, 173
 CLU_MXB_CLU2_P2.15, 173
 CLU_MXB_CLU2_P2.3, 173
 CLU_MXB_CLU2_P2.5, 173
 CLU_MXB_CLU2_P3.9, 173
 CLU_MXB_CLU2_PWM1, 173
 CLU_MXB_CLU2_PWM3, 174
 CLU_MXB_CLU2_SPICLK, 174
 CLU_MXB_CLU3_CLU0OUT, 174
 CLU_MXB_CLU3_CLU1OUT, 174
 CLU_MXB_CLU3_CLU2OUT, 174
 CLU_MXB_CLU3_CLU3OUT, 174
 CLU_MXB_CLU3_I2CSCL, 174
 CLU_MXB_CLU3_P1.10, 174
 CLU_MXB_CLU3_P1.11, 175
 CLU_MXB_CLU3_P1.13, 175
 CLU_MXB_CLU3_P1.14, 175
 CLU_MXB_CLU3_P2.14, 175
 CLU_MXB_CLU3_P2.15, 175
 CLU_MXB_CLU3_P2.4, 175
 CLU_MXB_CLU3_P2.5, 175
 CLU_MXB_CLU3_PWM0, 175
 CLU_MXB_CLU3_PWM2, 176
 CLU_MXB_CLU3_SPICSN, 176
 CLU_ReadOutValue, 177
 CLU_StructInit, 178
 lks32mc09x_cmp.c
 CMP_ClearIRQFlag, 549
 CMP_Disable, 549
 CMP_Enable, 550
 CMP_GetFilterValue, 550
 CMP_GetIRQFlag, 550
 CMP_GetValue, 551
 CMP_Init, 551
 CMP_SetFilterWidth, 552
 CMP_SetHysteresis, 552
 CMP_SetIRQSignal, 553
 CMP_SetSignal, 553
 CMP_SetWindowSignal, 554
 CMP_StructInit, 555
 lks32mc09x_cmp.h

- CMP_CHN_0, [184](#)
- CMP_CHN_1, [184](#)
- CMP_ClearIRQFlag, [194](#)
- CMP_CLK_DIV_0, [184](#)
- CMP_CLK_DIV_1, [184](#)
- CMP_CLK_DIV_2, [185](#)
- CMP_CLK_DIV_3, [185](#)
- CMP_CLK_DIV_4, [185](#)
- CMP_Disable, [194](#)
- CMP_Enable, [195](#)
- CMP_GetFilterValue, [196](#)
- CMP_GetIRQFlag, [196](#)
- CMP_GetValue, [197](#)
- CMP_HYS_0mV, [185](#)
- CMP_HYS_20mV, [185](#)
- CMP_IE_CMP0, [185](#)
- CMP_IE_CMP1, [185](#)
- CMP_IF_CMP0, [186](#)
- CMP_IF_CMP1, [186](#)
- CMP_Init, [197](#)
- CMP_RE_CMP0, [186](#)
- CMP_RE_CMP1, [186](#)
- CMP_SELN_0_DAC0, [186](#)
- CMP_SELN_0_DAC1, [186](#)
- CMP_SELN_0_HS_MID0, [186](#)
- CMP_SELN_0_IN_CMP0, [187](#)
- CMP_SELN_0_VBGP, [187](#)
- CMP_SELN_1_DAC0, [187](#)
- CMP_SELN_1_DAC1, [187](#)
- CMP_SELN_1_HS_MID1, [187](#)
- CMP_SELN_1_IN_CMP1, [187](#)
- CMP_SELN_1_VBGP, [188](#)
- CMP_SEL_P_0_CMP0_IP0, [188](#)
- CMP_SEL_P_0_CMP0_IP1, [188](#)
- CMP_SEL_P_0_CMP0_IP2, [188](#)
- CMP_SEL_P_0_CMP0_IP3, [188](#)
- CMP_SEL_P_0_CMP0_IP4, [188](#)
- CMP_SEL_P_0_CMP1_IP1, [189](#)
- CMP_SEL_P_0_CMP1_IP2, [189](#)
- CMP_SEL_P_0_CMP1_IP3, [189](#)
- CMP_SEL_P_0_DAC0_OUT, [189](#)
- CMP_SEL_P_0_OPA0_IP, [189](#)
- CMP_SEL_P_0_OPA0_OUTP, [189](#)
- CMP_SEL_P_0_OPA1_IP, [190](#)
- CMP_SEL_P_0_OPA1_OUTP, [190](#)
- CMP_SEL_P_0_OUT_OPA_CMP, [190](#)
- CMP_SEL_P_1_CMP0_IP1, [190](#)
- CMP_SEL_P_1_CMP0_IP2, [190](#)
- CMP_SEL_P_1_CMP0_IP3, [190](#)
- CMP_SEL_P_1_CMP1_IP0, [191](#)
- CMP_SEL_P_1_CMP1_IP1, [191](#)
- CMP_SEL_P_1_CMP1_IP2, [191](#)
- CMP_SEL_P_1_CMP1_IP3, [191](#)
- CMP_SEL_P_1_DAC1_OUT, [191](#)
- CMP_SEL_P_1_OPA2_IP, [191](#)
- CMP_SEL_P_1_OPA2_OUTP, [192](#)
- CMP_SEL_P_1_OPA3_IP, [192](#)
- CMP_SEL_P_1_OPA3_OUTP, [192](#)
- CMP_SEL_P_1_OPA_OUT_CMP, [192](#)
- CMP_SetFilterWidth, [198](#)
- CMP_SetHysteresis, [198](#)
- CMP_SetIRQSignal, [199](#)
- CMP_SetSignal, [199](#)
- CMP_SetWindowSignal, [200](#)
- CMP_SIGNAL_EDGE_N, [192](#)
- CMP_SIGNAL_EDGE_P, [192](#)
- CMP_SIGNAL_LEVEL_H, [193](#)
- CMP_SIGNAL_LEVEL_L, [193](#)
- CMP_SIGNAL_NONE, [193](#)
- CMP_StructInit, [201](#)
- CMP_WIN_LEVEL_H, [193](#)
- CMP_WIN_LEVEL_L, [193](#)
- CMP_WIN_SIGNAL_MCPWM0_0, [193](#)
- CMP_WIN_SIGNAL_MCPWM0_1, [193](#)
- CMP_WIN_SIGNAL_MCPWM0_2, [193](#)
- CMP_WIN_SIGNAL_MCPWM0_3, [194](#)
- CMP_WIN_SIGNAL_NONE, [194](#)
- lks32mc09x_crc.c
 - CRC_Calc, [560](#)
 - CRC_CalcCommon, [560](#)
 - CRC_Disable, [561](#)
 - CRC_Enable, [562](#)
 - CRC_Init, [562](#)
 - CRC_StructInit, [563](#)
 - CRC_XOROUT, [564](#)
- lks32mc09x_crc.h
 - CRC_Calc, [208](#)
 - CRC_CalcCommon, [208](#)
 - CRC_Disable, [209](#)
 - CRC_Enable, [209](#)
 - CRC_Init, [210](#)
 - CRC_MODEL_CRC_16_CCITT, [205](#)
 - CRC_MODEL_CRC_16_CCITT_FALSE, [205](#)
 - CRC_MODEL_CRC_16_DNP, [205](#)
 - CRC_MODEL_CRC_16_IBMx1, [205](#)
 - CRC_MODEL_CRC_16_MAXIM, [205](#)
 - CRC_MODEL_CRC_16_MODEBU, [205](#)
 - CRC_MODEL_CRC_16_USBx1, [205](#)
 - CRC_MODEL_CRC_16_X25, [206](#)
 - CRC_MODEL_CRC_16_XMODEM, [206](#)
 - CRC_MODEL_CRC_32, [206](#)
 - CRC_MODEL_CRC_32_MPEG_2, [206](#)
 - CRC_MODEL_CRC_8, [206](#)
 - CRC_MODEL_CRC_8_ITU, [206](#)
 - CRC_MODEL_CRC_8_MAXIMx, [207](#)
 - CRC_MODEL_CRC_8_ROHC, [207](#)
 - CRC_StructInit, [211](#)
 - CRC_WIDTH_16, [207](#)
 - CRC_WIDTH_32, [207](#)
 - CRC_WIDTH_7, [207](#)
 - CRC_WIDTH_8, [207](#)
- lks32mc09x_dac.c
 - DAC_Cmd, [569](#)
 - DAC_Disable, [569](#)
 - DAC_Enable, [570](#)
 - DAC_Init, [570](#)

- DAC_Output, 571
- DAC_OutputValue, 572
- DAC_OutputVoltage, 572
- DAC_Range, 573
- DAC_Reset, 573
- DAC_SetRange, 574
- DAC_Step, 575
- DAC_StructInit, 576
- DAC_Value, 576
- DAC_Voltage, 577
- lks32mc09x_dac.h
 - DAC_CHANNEL_0, 214
 - DAC_CHANNEL_1, 214
 - DAC_Cmd, 217
 - DAC_Disable, 218
 - DAC_Enable, 218
 - DAC_Init, 219
 - DAC_Output, 220
 - DAC_OutputValue, 220
 - DAC_OutputVoltage, 221
 - DAC_Range, 222
 - DAC_RANGE_0_1V2, 215
 - DAC_RANGE_0_4V76, 215
 - DAC_RANGE_1_1V2, 215
 - DAC_RANGE_1_3V, 215
 - DAC_Reset, 222
 - DAC_SetRange, 223
 - DAC_Step, 224
 - DAC_STEP_1, 215
 - DAC_STEP_2, 215
 - DAC_STEP_3, 216
 - DAC_STEP_0, 216
 - DAC_STEP_1, 216
 - DAC_STEP_2, 216
 - DAC_STEP_3, 216
 - DAC_STEP_4, 216
 - DAC_STEP_TRG_NONE, 217
 - DAC_STEP_TRG_TIMER0, 217
 - DAC_STEP_TRG_TIMER1, 217
 - DAC_STEP_TRG_TIMER2, 217
 - DAC_STEP_TRG_TIMER3, 217
 - DAC_StructInit, 224
 - DAC_Value, 226
 - DAC_Voltage, 226
- lks32mc09x_dma.c
 - DMA_CHx_EN, 582
 - DMA_ClearIRQFlag, 582
 - DMA_Disable, 583
 - DMA_Enable, 583
 - DMA_GetFlag, 583
 - DMA_GetIRQFlag, 584
 - DMA_Init, 584
 - DMA_Mem2Reg, 585
 - DMA_Memcpy, 586
 - DMA_Reset, 587
 - DMA_SetAddr, 588
 - DMA_SetLength, 588
 - DMA_SetRequest, 589
 - DMA_StructInit, 589
 - DMA_Trigger, 589
- lks32mc09x_dma.h
 - DMA_BYTE_TRANS, 231
 - DMA_CHx_EN, 240
 - DMA_ClearIRQFlag, 241
 - DMA_Disable, 241
 - DMA_Enable, 241
 - DMA_GetFlag, 241
 - DMA_GetIRQFlag, 242
 - DMA_HALFWORD_TRANS, 231
 - DMA_IE_CH0, 231
 - DMA_IE_CH1, 232
 - DMA_IE_CH2, 232
 - DMA_IE_CH3, 232
 - DMA_IF_CH0, 232
 - DMA_IF_CH1, 232
 - DMA_IF_CH2, 232
 - DMA_IF_CH3, 232
 - DMA_Init, 242
 - DMA_Mem2Reg, 243
 - DMA_Memcpy, 244
 - DMA_MEMORY_2_PERI, 233
 - DMA_PERI_2_MEMORY, 233
 - DMA_REN_ADC0, 233
 - DMA_REN_ADC1, 233
 - DMA_REN_CMP, 233
 - DMA_REN_DSP, 233
 - DMA_REN_GPIO, 233
 - DMA_REN_HALL0, 234
 - DMA_REN_I2C0_RX, 234
 - DMA_REN_I2C0_TX, 234
 - DMA_REN_MCPWM_RE0, 234
 - DMA_REN_MCPWM_RE1, 234
 - DMA_REN_SIF, 234
 - DMA_REN_SPI_RX, 234
 - DMA_REN_SPI_TX, 234
 - DMA_REN_SW, 235
 - DMA_REN_TIMER0, 235
 - DMA_REN_TIMER1, 235
 - DMA_REN_TIMER2, 235
 - DMA_REN_TIMER3, 235
 - DMA_REN_UART0_RX, 235
 - DMA_REN_UART0_TX, 235
 - DMA_REN_UART1_RX, 235
 - DMA_REN_UART1_TX, 236
 - DMA_REN_UART2_RX, 236
 - DMA_REN_UART2_TX, 236
 - DMA_REQ_ADC0, 236
 - DMA_REQ_ADC1, 236
 - DMA_REQ_CMP, 236
 - DMA_REQ_DSP, 236
 - DMA_REQ_GPIO, 237
 - DMA_REQ_HALL, 237
 - DMA_REQ_I2C0_RX, 237
 - DMA_REQ_I2C0_TX, 237
 - DMA_REQ_MCPWM0, 237
 - DMA_REQ_MCPWM1, 237

- DMA_REQ_SIF, 238
- DMA_REQ_SPI_RX, 238
- DMA_REQ_SPI_TX, 238
- DMA_REQ_SW, 238
- DMA_REQ_TIMER0, 238
- DMA_REQ_TIMER1, 238
- DMA_REQ_TIMER2, 239
- DMA_REQ_TIMER3, 239
- DMA_REQ_UART0_RX, 239
- DMA_REQ_UART0_TX, 239
- DMA_REQ_UART1_RX, 239
- DMA_REQ_UART1_TX, 239
- DMA_Reset, 245
- DMA_SetAddr, 246
- DMA_SetLength, 246
- DMA_SetRequest, 247
- DMA_StructInit, 247
- DMA_TCIE, 240
- DMA_TEIE, 240
- DMA_Trigger, 248
- DMA_WORD_TRANS, 240
- lks32mc09x_dsp.c
 - DSP_Disable, 594
 - DSP_Div, 595
 - DSP_Enable, 595
 - DSP_Init, 596
 - DSP_Mod, 596
 - DSP_Reset, 597
 - DSP_Rms, 597
 - DSP_Sqrt, 597
- lks32mc09x_dsp.h
 - DSP_Disable, 251
 - DSP_Div, 251
 - DSP_Enable, 252
 - DSP_Init, 252
 - DSP_Mod, 252
 - DSP_Reset, 253
 - DSP_Rms, 253
 - DSP_Sqrt, 254
- lks32mc09x_eeprom.c
 - EEPROM_CFG, 600
 - EEPROM_Disable, 600
 - EEPROM_Enable, 600
 - EEPROM_Init, 601
 - EEPROM_InitDefault, 602
 - EEPROM_RDATA, 600
 - EEPROM_Read, 602
 - EEPROM_ReadData, 603
 - EEPROM_Reset, 603
 - EEPROM_StructInit, 604
 - EEPROM_Write, 604
 - EEPROM_WriteData, 604
- lks32mc09x_eeprom.h
 - EEPROM_ADDR_LEN_16, 256
 - EEPROM_ADDR_LEN_8, 256
 - EEPROM_BAUD_100K, 257
 - EEPROM_BAUD_1M, 257
 - EEPROM_BAUD_1M2, 257
 - EEPROM_BAUD_400K, 257
 - EEPROM_Disable, 258
 - EEPROM_Enable, 258
 - EEPROM_Init, 258
 - EEPROM_InitDefault, 259
 - EEPROM_Read, 259
 - EEPROM_ReadData, 260
 - EEPROM_Reset, 261
 - EEPROM_StructInit, 261
 - EEPROM_Write, 261
 - EEPROM_WriteData, 262
- lks32mc09x_exti.c
 - EXTI_ClearIRQFlag, 608
 - EXTI_DisableIRQFlag, 608
 - EXTI_EnableIRQFlag, 608
 - EXTI_GetIRQFlag, 609
 - exti_ie_flg, 610
 - EXTI_Reset, 609
 - EXTI_SetCkco, 609
 - EXTI_TriggerConfig, 610
- lks32mc09x_exti.h
 - EXTI_ClearIRQFlag, 270
 - EXTI_CLKO_0, 265
 - EXTI_CLKO_ADC, 265
 - EXTI_CLKO_HSE, 265
 - EXTI_CLKO_HSI, 265
 - EXTI_CLKO_LSI, 265
 - EXTI_CLKO_PLL, 266
 - EXTI_CR_P0_0, 266
 - EXTI_CR_P0_11, 266
 - EXTI_CR_P0_12, 266
 - EXTI_CR_P0_13, 266
 - EXTI_CR_P0_14, 266
 - EXTI_CR_P0_15, 266
 - EXTI_CR_P0_2, 266
 - EXTI_CR_P1_0, 267
 - EXTI_CR_P2_0, 267
 - EXTI_CR_P2_13, 267
 - EXTI_CR_P2_3, 267
 - EXTI_CR_P2_4, 267
 - EXTI_CR_P2_5, 267
 - EXTI_CR_P2_6, 267
 - EXTI_CR_P2_7, 267
 - EXTI_CR_P2_9, 268
 - EXTI_DisableIRQFlag, 270
 - EXTI_EnableIRQFlag, 271
 - EXTI_GetIRQFlag, 271
 - EXTI_IF_P0_0, 268
 - EXTI_IF_P0_11, 268
 - EXTI_IF_P0_12, 268
 - EXTI_IF_P0_13, 268
 - EXTI_IF_P0_14, 268
 - EXTI_IF_P0_15, 268
 - EXTI_IF_P0_2, 268
 - EXTI_IF_P1_0, 269
 - EXTI_IF_P2_0, 269
 - EXTI_IF_P2_13, 269
 - EXTI_IF_P2_3, 269

- EXTI_IF_P2_4, [269](#)
- EXTI_IF_P2_5, [269](#)
- EXTI_IF_P2_6, [269](#)
- EXTI_IF_P2_7, [269](#)
- EXTI_IF_P2_9, [270](#)
- EXTI_Mode_TypeDef, [270](#)
- EXTI_Reset, [272](#)
- EXTI_SetClko, [272](#)
- EXTI_Trigger_Edge, [270](#)
- EXTI_Trigger_Negedge, [270](#)
- EXTI_Trigger_None, [270](#)
- EXTI_Trigger_Posedge, [270](#)
- EXTI_TriggerConfig, [273](#)
- lks32mc09x_flash.c
 - FLASH_Erase, [612](#)
 - FLASH_EraseNvr, [613](#)
 - FLASH_Read, [613](#)
 - FLASH_ReadNvr, [613](#)
 - FLASH_Write, [614](#)
 - FLASH_WriteNvr, [614](#)
- lks32mc09x_flash.h
 - FLASH_Erase, [312](#)
 - FLASH_EraseNvr, [313](#)
 - FLASH_Read, [313](#)
 - FLASH_ReadNvr, [313](#)
 - FLASH_SECTOR, [280](#)
 - FLASH_SECTOR_0, [280](#)
 - FLASH_SECTOR_1, [280](#)
 - FLASH_SECTOR_10, [280](#)
 - FLASH_SECTOR_100, [280](#)
 - FLASH_SECTOR_101, [281](#)
 - FLASH_SECTOR_102, [281](#)
 - FLASH_SECTOR_103, [281](#)
 - FLASH_SECTOR_104, [281](#)
 - FLASH_SECTOR_105, [281](#)
 - FLASH_SECTOR_106, [281](#)
 - FLASH_SECTOR_107, [281](#)
 - FLASH_SECTOR_108, [281](#)
 - FLASH_SECTOR_109, [282](#)
 - FLASH_SECTOR_11, [282](#)
 - FLASH_SECTOR_110, [282](#)
 - FLASH_SECTOR_111, [282](#)
 - FLASH_SECTOR_112, [282](#)
 - FLASH_SECTOR_113, [282](#)
 - FLASH_SECTOR_114, [282](#)
 - FLASH_SECTOR_115, [282](#)
 - FLASH_SECTOR_116, [283](#)
 - FLASH_SECTOR_117, [283](#)
 - FLASH_SECTOR_118, [283](#)
 - FLASH_SECTOR_119, [283](#)
 - FLASH_SECTOR_12, [283](#)
 - FLASH_SECTOR_120, [283](#)
 - FLASH_SECTOR_121, [283](#)
 - FLASH_SECTOR_122, [283](#)
 - FLASH_SECTOR_123, [284](#)
 - FLASH_SECTOR_124, [284](#)
 - FLASH_SECTOR_125, [284](#)
 - FLASH_SECTOR_126, [284](#)
 - FLASH_SECTOR_127, [284](#)
 - FLASH_SECTOR_128, [284](#)
 - FLASH_SECTOR_129, [284](#)
 - FLASH_SECTOR_13, [284](#)
 - FLASH_SECTOR_130, [285](#)
 - FLASH_SECTOR_131, [285](#)
 - FLASH_SECTOR_132, [285](#)
 - FLASH_SECTOR_133, [285](#)
 - FLASH_SECTOR_134, [285](#)
 - FLASH_SECTOR_135, [285](#)
 - FLASH_SECTOR_136, [285](#)
 - FLASH_SECTOR_137, [285](#)
 - FLASH_SECTOR_138, [286](#)
 - FLASH_SECTOR_139, [286](#)
 - FLASH_SECTOR_14, [286](#)
 - FLASH_SECTOR_140, [286](#)
 - FLASH_SECTOR_141, [286](#)
 - FLASH_SECTOR_142, [286](#)
 - FLASH_SECTOR_143, [286](#)
 - FLASH_SECTOR_144, [286](#)
 - FLASH_SECTOR_145, [287](#)
 - FLASH_SECTOR_146, [287](#)
 - FLASH_SECTOR_147, [287](#)
 - FLASH_SECTOR_148, [287](#)
 - FLASH_SECTOR_149, [287](#)
 - FLASH_SECTOR_15, [287](#)
 - FLASH_SECTOR_150, [287](#)
 - FLASH_SECTOR_151, [287](#)
 - FLASH_SECTOR_152, [288](#)
 - FLASH_SECTOR_153, [288](#)
 - FLASH_SECTOR_154, [288](#)
 - FLASH_SECTOR_155, [288](#)
 - FLASH_SECTOR_156, [288](#)
 - FLASH_SECTOR_157, [288](#)
 - FLASH_SECTOR_158, [288](#)
 - FLASH_SECTOR_159, [288](#)
 - FLASH_SECTOR_16, [289](#)
 - FLASH_SECTOR_160, [289](#)
 - FLASH_SECTOR_161, [289](#)
 - FLASH_SECTOR_162, [289](#)
 - FLASH_SECTOR_163, [289](#)
 - FLASH_SECTOR_164, [289](#)
 - FLASH_SECTOR_165, [289](#)
 - FLASH_SECTOR_166, [289](#)
 - FLASH_SECTOR_167, [290](#)
 - FLASH_SECTOR_168, [290](#)
 - FLASH_SECTOR_169, [290](#)
 - FLASH_SECTOR_17, [290](#)
 - FLASH_SECTOR_170, [290](#)
 - FLASH_SECTOR_171, [290](#)
 - FLASH_SECTOR_172, [290](#)
 - FLASH_SECTOR_173, [290](#)
 - FLASH_SECTOR_174, [291](#)
 - FLASH_SECTOR_175, [291](#)
 - FLASH_SECTOR_176, [291](#)
 - FLASH_SECTOR_177, [291](#)
 - FLASH_SECTOR_178, [291](#)
 - FLASH_SECTOR_179, [291](#)

FLASH_SECTOR.18, [291](#)
FLASH_SECTOR.180, [291](#)
FLASH_SECTOR.181, [292](#)
FLASH_SECTOR.182, [292](#)
FLASH_SECTOR.183, [292](#)
FLASH_SECTOR.184, [292](#)
FLASH_SECTOR.185, [292](#)
FLASH_SECTOR.186, [292](#)
FLASH_SECTOR.187, [292](#)
FLASH_SECTOR.188, [292](#)
FLASH_SECTOR.189, [293](#)
FLASH_SECTOR.19, [293](#)
FLASH_SECTOR.190, [293](#)
FLASH_SECTOR.191, [293](#)
FLASH_SECTOR.192, [293](#)
FLASH_SECTOR.193, [293](#)
FLASH_SECTOR.194, [293](#)
FLASH_SECTOR.195, [293](#)
FLASH_SECTOR.196, [294](#)
FLASH_SECTOR.197, [294](#)
FLASH_SECTOR.198, [294](#)
FLASH_SECTOR.199, [294](#)
FLASH_SECTOR.2, [294](#)
FLASH_SECTOR.20, [294](#)
FLASH_SECTOR.200, [294](#)
FLASH_SECTOR.201, [294](#)
FLASH_SECTOR.202, [295](#)
FLASH_SECTOR.203, [295](#)
FLASH_SECTOR.204, [295](#)
FLASH_SECTOR.205, [295](#)
FLASH_SECTOR.206, [295](#)
FLASH_SECTOR.207, [295](#)
FLASH_SECTOR.208, [295](#)
FLASH_SECTOR.209, [295](#)
FLASH_SECTOR.21, [296](#)
FLASH_SECTOR.210, [296](#)
FLASH_SECTOR.211, [296](#)
FLASH_SECTOR.212, [296](#)
FLASH_SECTOR.213, [296](#)
FLASH_SECTOR.214, [296](#)
FLASH_SECTOR.215, [296](#)
FLASH_SECTOR.216, [296](#)
FLASH_SECTOR.217, [297](#)
FLASH_SECTOR.218, [297](#)
FLASH_SECTOR.219, [297](#)
FLASH_SECTOR.22, [297](#)
FLASH_SECTOR.220, [297](#)
FLASH_SECTOR.221, [297](#)
FLASH_SECTOR.222, [297](#)
FLASH_SECTOR.223, [297](#)
FLASH_SECTOR.224, [298](#)
FLASH_SECTOR.225, [298](#)
FLASH_SECTOR.226, [298](#)
FLASH_SECTOR.227, [298](#)
FLASH_SECTOR.228, [298](#)
FLASH_SECTOR.229, [298](#)
FLASH_SECTOR.23, [298](#)
FLASH_SECTOR.230, [298](#)
FLASH_SECTOR.231, [299](#)
FLASH_SECTOR.232, [299](#)
FLASH_SECTOR.233, [299](#)
FLASH_SECTOR.234, [299](#)
FLASH_SECTOR.235, [299](#)
FLASH_SECTOR.236, [299](#)
FLASH_SECTOR.237, [299](#)
FLASH_SECTOR.238, [299](#)
FLASH_SECTOR.239, [300](#)
FLASH_SECTOR.24, [300](#)
FLASH_SECTOR.240, [300](#)
FLASH_SECTOR.241, [300](#)
FLASH_SECTOR.242, [300](#)
FLASH_SECTOR.243, [300](#)
FLASH_SECTOR.244, [300](#)
FLASH_SECTOR.245, [300](#)
FLASH_SECTOR.246, [301](#)
FLASH_SECTOR.247, [301](#)
FLASH_SECTOR.248, [301](#)
FLASH_SECTOR.249, [301](#)
FLASH_SECTOR.25, [301](#)
FLASH_SECTOR.250, [301](#)
FLASH_SECTOR.251, [301](#)
FLASH_SECTOR.252, [301](#)
FLASH_SECTOR.253, [302](#)
FLASH_SECTOR.254, [302](#)
FLASH_SECTOR.255, [302](#)
FLASH_SECTOR.26, [302](#)
FLASH_SECTOR.27, [302](#)
FLASH_SECTOR.28, [302](#)
FLASH_SECTOR.29, [302](#)
FLASH_SECTOR.3, [302](#)
FLASH_SECTOR.30, [303](#)
FLASH_SECTOR.31, [303](#)
FLASH_SECTOR.32, [303](#)
FLASH_SECTOR.33, [303](#)
FLASH_SECTOR.34, [303](#)
FLASH_SECTOR.35, [303](#)
FLASH_SECTOR.36, [303](#)
FLASH_SECTOR.37, [303](#)
FLASH_SECTOR.38, [304](#)
FLASH_SECTOR.39, [304](#)
FLASH_SECTOR.4, [304](#)
FLASH_SECTOR.40, [304](#)
FLASH_SECTOR.41, [304](#)
FLASH_SECTOR.42, [304](#)
FLASH_SECTOR.43, [304](#)
FLASH_SECTOR.44, [304](#)
FLASH_SECTOR.45, [305](#)
FLASH_SECTOR.46, [305](#)
FLASH_SECTOR.47, [305](#)
FLASH_SECTOR.48, [305](#)
FLASH_SECTOR.49, [305](#)
FLASH_SECTOR.5, [305](#)
FLASH_SECTOR.50, [305](#)
FLASH_SECTOR.51, [305](#)
FLASH_SECTOR.52, [306](#)
FLASH_SECTOR.53, [306](#)

- FLASH_SECTOR.54, [306](#)
- FLASH_SECTOR.55, [306](#)
- FLASH_SECTOR.56, [306](#)
- FLASH_SECTOR.57, [306](#)
- FLASH_SECTOR.58, [306](#)
- FLASH_SECTOR.59, [306](#)
- FLASH_SECTOR.6, [307](#)
- FLASH_SECTOR.60, [307](#)
- FLASH_SECTOR.61, [307](#)
- FLASH_SECTOR.62, [307](#)
- FLASH_SECTOR.63, [307](#)
- FLASH_SECTOR.64, [307](#)
- FLASH_SECTOR.65, [307](#)
- FLASH_SECTOR.66, [307](#)
- FLASH_SECTOR.67, [308](#)
- FLASH_SECTOR.68, [308](#)
- FLASH_SECTOR.69, [308](#)
- FLASH_SECTOR.7, [308](#)
- FLASH_SECTOR.70, [308](#)
- FLASH_SECTOR.71, [308](#)
- FLASH_SECTOR.72, [308](#)
- FLASH_SECTOR.73, [308](#)
- FLASH_SECTOR.74, [309](#)
- FLASH_SECTOR.75, [309](#)
- FLASH_SECTOR.76, [309](#)
- FLASH_SECTOR.77, [309](#)
- FLASH_SECTOR.78, [309](#)
- FLASH_SECTOR.79, [309](#)
- FLASH_SECTOR.8, [309](#)
- FLASH_SECTOR.80, [309](#)
- FLASH_SECTOR.81, [310](#)
- FLASH_SECTOR.82, [310](#)
- FLASH_SECTOR.83, [310](#)
- FLASH_SECTOR.84, [310](#)
- FLASH_SECTOR.85, [310](#)
- FLASH_SECTOR.86, [310](#)
- FLASH_SECTOR.87, [310](#)
- FLASH_SECTOR.88, [310](#)
- FLASH_SECTOR.89, [311](#)
- FLASH_SECTOR.9, [311](#)
- FLASH_SECTOR.90, [311](#)
- FLASH_SECTOR.91, [311](#)
- FLASH_SECTOR.92, [311](#)
- FLASH_SECTOR.93, [311](#)
- FLASH_SECTOR.94, [311](#)
- FLASH_SECTOR.95, [311](#)
- FLASH_SECTOR.96, [312](#)
- FLASH_SECTOR.97, [312](#)
- FLASH_SECTOR.98, [312](#)
- FLASH_SECTOR.99, [312](#)
- FLASH_SECTOR.END_128K, [312](#)
- FLASH_SECTOR.END_32K, [312](#)
- FLASH_SECTOR.END_64K, [312](#)
- FLASH.Write, [314](#)
- FLASH.WriteNvr, [314](#)
- lks32mc09x_gpio.c
 - GPIO.Config, [617](#)
 - GPIO.Disable, [618](#)
- GPIO.Enable, [618](#)
- GPIO.Init, [619](#)
- GPIO.LockRegister, [620](#)
- GPIO.PinAFConfig, [620](#)
- GPIO.ReadInputData, [621](#)
- GPIO.ReadInputDataBit, [621](#)
- GPIO.ReadOutputData, [621](#)
- GPIO.ReadOutputDataBit, [622](#)
- GPIO.Reset, [622](#)
- GPIO.ResetBits, [623](#)
- GPIO.SetBits, [623](#)
- GPIO.StructInit, [623](#)
- GPIO.ToggleBits, [624](#)
- GPIO.UnlockRegister, [624](#)
- GPIO.Write, [624](#)
- GPIO.WriteBit, [625](#)
- lks32mc09x_gpio.h
 - AF0.GPIO, [323](#)
 - AF10.CAN, [323](#)
 - AF11.SIF, [323](#)
 - AF12.CL, [324](#)
 - AF1.SYSAF, [324](#)
 - AF2.HALL, [324](#)
 - AF3.MCPWM, [324](#)
 - AF4.UART, [324](#)
 - AF5.SPI, [324](#)
 - AF6.I2C, [325](#)
 - AF7.TIMER01, [325](#)
 - AF8.TIMER23, [325](#)
 - AF9.DEBUG, [325](#)
 - GPIO.AF_ADC0.TRIGGER, [325](#)
 - GPIO.AF_ADC1.TRIGGER, [325](#)
 - GPIO.AF_AFE, [326](#)
 - GPIO.AF_CAN, [326](#)
 - GPIO.AF_CL, [326](#)
 - GPIO.AF_CLKO, [326](#)
 - GPIO.AF_CLU, [326](#)
 - GPIO.AF_CMP_OUT, [326](#)
 - GPIO.AF_GPIO, [327](#)
 - GPIO.AF_HALL, [327](#)
 - GPIO.AF_I2C, [327](#)
 - GPIO.AF_MCPWM, [327](#)
 - GPIO.AF_QEP0, [327](#)
 - GPIO.AF_SIF, [327](#)
 - GPIO.AF_SPI, [328](#)
 - GPIO.AF_TIMER0, [328](#)
 - GPIO.AF_TIMER1, [328](#)
 - GPIO.AF_TIMER2, [328](#)
 - GPIO.AF_UART, [328](#)
 - GPIO.Bit.Action, [334](#)
 - GPIO.Bit.RESET, [334](#)
 - GPIO.Bit.SET, [334](#)
 - GPIO.Config, [335](#)
 - GPIO.Disable, [336](#)
 - GPIO.Enable, [336](#)
 - GPIO.Init, [336](#)
 - GPIO.LockRegister, [337](#)
 - GPIO.Mode_ANA, [335](#)

- GPIO_Mode_IN, 335
- GPIO_Mode_IO, 335
- GPIO_Mode_OUT, 335
- GPIO_Mode_TypeDef, 334
- GPIO_Pin_0, 328
- GPIO_Pin_1, 329
- GPIO_Pin_10, 329
- GPIO_Pin_11, 329
- GPIO_Pin_12, 329
- GPIO_Pin_13, 329
- GPIO_Pin_14, 329
- GPIO_Pin_15, 330
- GPIO_Pin_2, 330
- GPIO_Pin_3, 330
- GPIO_Pin_4, 330
- GPIO_Pin_5, 330
- GPIO_Pin_6, 330
- GPIO_Pin_7, 331
- GPIO_Pin_8, 331
- GPIO_Pin_9, 331
- GPIO_Pin_ALL, 331
- GPIO_Pin_NONE, 331
- GPIO_PinAFConfig, 338
- GPIO_PinSource_0, 331
- GPIO_PinSource_1, 332
- GPIO_PinSource_10, 332
- GPIO_PinSource_11, 332
- GPIO_PinSource_12, 332
- GPIO_PinSource_13, 332
- GPIO_PinSource_14, 332
- GPIO_PinSource_15, 333
- GPIO_PinSource_2, 333
- GPIO_PinSource_3, 333
- GPIO_PinSource_4, 333
- GPIO_PinSource_5, 333
- GPIO_PinSource_6, 333
- GPIO_PinSource_7, 334
- GPIO_PinSource_8, 334
- GPIO_PinSource_9, 334
- GPIO_PuPd_NOPULL, 335
- GPIO_PuPd_TypeDef, 335
- GPIO_PuPd_UP, 335
- GPIO_ReadInputData, 338
- GPIO_ReadInputDataBit, 338
- GPIO_ReadOutputData, 339
- GPIO_ReadOutputDataBit, 339
- GPIO_Reset, 340
- GPIO_ResetBits, 340
- GPIO_SetBits, 340
- GPIO_StructInit, 341
- GPIO_ToggleBits, 341
- GPIO_UnlockRegister, 341
- GPIO_Write, 342
- GPIO_WriteBit, 342
- lks32mc09x_hall.c
 - HALL_Clear_IRQ, 629
 - HALL_Cmd, 629
 - HALL_GetCaptureValue, 629
 - HALL_GetCount, 629
 - HALL_GetFilterValue, 629
 - HALL_GetFlag, 630
 - HALL_GetIRQFlag, 630
 - HALL_Init, 630
 - HALL_IsCaptureEvent, 631
 - HALL_IsOverFlowEvent, 631
 - HALL_SoftTrigger, 631
 - HALL_StructInit, 632
- lks32mc09x_hall.h
 - HALL_CAPTURE_EVENT, 346
 - HALL_Clear_IRQ, 347
 - HALL_CLK_DIV1, 346
 - HALL_CLK_DIV2, 346
 - HALL_CLK_DIV4, 347
 - HALL_CLK_DIV8, 347
 - HALL_Cmd, 347
 - HALL_GetCaptureValue, 348
 - HALL_GetCount, 348
 - HALL_GetFilterValue, 348
 - HALL_GetFlag, 348
 - HALL_GetIRQFlag, 349
 - HALL_Init, 349
 - HALL_IsCaptureEvent, 350
 - HALL_IsOverFlowEvent, 350
 - HALL_OVERFLOW_EVENT, 347
 - HALL_SoftTrigger, 350
 - HALL_StructInit, 350
- lks32mc09x_i2c.c
 - I2C_ClearBusState, 635
 - I2C_ClearIRQFlag, 635
 - I2C_Disable, 636
 - I2C_Enable, 636
 - I2C_GetFlag, 637
 - I2C_GetIRQFlag, 637
 - I2C_Init, 638
 - I2C_ReadBusState, 638
 - I2C_ReadData, 639
 - I2C_SendData, 639
 - I2C_SetBaudRate, 639
 - I2C_StructInit, 640
 - I2C_TableAck, 640
 - I2C_TableIrq, 641
 - I2C_TableRead, 641
 - I2C_TableSend, 641
 - I2C_TableState, 642
 - I2C_TableVariable, 642
- lks32mc09x_i2c.h
 - I2C_BUS_STATE_ADDR_DATA, 354
 - I2C_BUS_STATE_Done, 354
 - I2C_BUS_STATE_LOST_ARB, 354
 - I2C_BUS_STATE_RX_ACK, 354
 - I2C_BUS_STATE_STOP_EVT, 354
 - I2C_BUS_STATE_STT_ERR, 354
 - I2C_ClearBusState, 355
 - I2C_ClearIRQFlag, 356
 - I2C_Disable, 356
 - I2C_Enable, 356

- I2C_GetFlag, 357
- I2C_GetIRQFlag, 358
- I2C_IF_ADDR_CMP, 355
- I2C_IF_BUS_ERR, 355
- I2C_IF_NACK, 355
- I2C_IF_STOP, 355
- I2C_IF_TC, 355
- I2C_Init, 358
- I2c_ReadBusState, 359
- I2C_ReadData, 359
- I2C_SendData, 360
- I2C_SetBaudRate, 360
- I2C_StructInit, 361
- I2C_TableAck, 361
- I2C_TableIrq, 361
- I2C_TableRead, 362
- I2C_TableSend, 362
- I2C_TableState, 363
- lks32mc09x_iwdg.c
 - IWDG_DISABLE, 647
 - IWDG_ENABLE, 647
 - IWDG_Feed, 647
 - IWDG_Init, 647
 - IWDG_StrutInit, 648
 - IWDG_Update, 648
- lks32mc09x_iwdg.h
 - IWDG_DISABLE, 366
 - IWDG_ENABLE, 366
 - IWDG_Feed, 366
 - IWDG_Init, 366
 - IWDG_SECOND2IWDGCNT, 365
 - IWDG_SECOND2IWDGCNT_SAT, 365
 - IWDG_StrutInit, 367
 - IWDG_Update, 367
- lks32mc09x_mcpwm.c
 - MCPWM_ClearErrorIRQFlag, 650
 - MCPWM_ClearIRQFlag, 651
 - mcpwm_default_io, 659
 - MCPWM_Disable, 651
 - MCPWM_Enable, 652
 - MCPWM_GetCountTh, 652
 - MCPWM_GetCountVal, 653
 - MCPWM_GetEifFlag, 653
 - MCPWM_GetErrorIRQFlag, 654
 - MCPWM_GetIfFlag, 654
 - MCPWM_GetIRQFlag, 654
 - MCPWM_Init, 655
 - MCPWM_Reset, 656
 - MCPWM_SetIOConfig, 656
 - MCPWM_SetOutputState, 657
 - MCPWM_SetOutputVal, 657
 - MCPWM_SetTMR, 657
 - MCPWM_StartCount, 658
 - MCPWM_StopCount, 658
 - MCPWM_StructInit, 658
 - MCPWM_TmpUpdateFaillo, 659
- lks32mc09x_mcpwm.h
 - MCPWM_AUEN_ALL, 374
 - MCPWM_AUEN_CNT0, 374
 - MCPWM_AUEN_DEFAULT, 375
 - MCPWM_AUEN_IO, 375
 - MCPWM_AUEN_IO012, 375
 - MCPWM_AUEN_IO3, 375
 - MCPWM_AUEN_TH0, 375
 - MCPWM_AUEN_TH00, 375
 - MCPWM_AUEN_TH01, 375
 - MCPWM_AUEN_TH10, 375
 - MCPWM_AUEN_TH11, 376
 - MCPWM_AUEN_TH20, 376
 - MCPWM_AUEN_TH21, 376
 - MCPWM_AUEN_TH30, 376
 - MCPWM_AUEN_TH31, 376
 - MCPWM_AUEN_TMR0, 376
 - MCPWM_AUEN_TMR1, 376
 - MCPWM_AUEN_TMR2, 376
 - MCPWM_AUEN_TMR3, 377
 - MCPWM_ClearErrorIRQFlag, 399
 - MCPWM_ClearIRQFlag, 399
 - MCPWM_CLK_DIV_1, 377
 - MCPWM_CLK_DIV_128, 377
 - MCPWM_CLK_DIV_16, 377
 - MCPWM_CLK_DIV_2, 377
 - MCPWM_CLK_DIV_32, 377
 - MCPWM_CLK_DIV_4, 377
 - MCPWM_CLK_DIV_64, 377
 - MCPWM_CLK_DIV_8, 378
 - MCPWM_COUNT_EVT_TIMER0_CMP0, 378
 - MCPWM_COUNT_EVT_TIMER0_CMP1, 378
 - MCPWM_COUNT_EVT_TIMER1_CMP0, 378
 - MCPWM_COUNT_EVT_TIMER1_CMP1, 378
 - MCPWM_COUNT_EVT_TIMER2_CMP0, 378
 - MCPWM_COUNT_EVT_TIMER2_CMP1, 378
 - MCPWM_COUNT_OFF, 378
 - MCPWM_COUNT_START, 379
 - MCPWM_Disable, 400
 - MCPWM_EIE_FAIL0, 379
 - MCPWM_EIE_FAIL1, 379
 - MCPWM_EIF_FAIL0, 379
 - MCPWM_EIF_FAIL1, 379
 - MCPWM_Enable, 400
 - MCPWM_FAIL0_BKIN0_H, 379
 - MCPWM_FAIL0_BKIN0_L, 379
 - MCPWM_FAIL0_CLU_OUT0_H, 379
 - MCPWM_FAIL0_CLU_OUT0_L, 380
 - MCPWM_FAIL0_CLU_OUT1_H, 380
 - MCPWM_FAIL0_CLU_OUT1_L, 380
 - MCPWM_FAIL0_CMP0_H, 380
 - MCPWM_FAIL0_CMP0_L, 380
 - MCPWM_FAIL0_OFF, 380
 - MCPWM_FAIL1_BKIN1_H, 380
 - MCPWM_FAIL1_BKIN1_L, 380
 - MCPWM_FAIL1_CLU_OUT2_H, 381
 - MCPWM_FAIL1_CLU_OUT2_L, 381
 - MCPWM_FAIL1_CLU_OUT3_H, 381
 - MCPWM_FAIL1_CLU_OUT3_L, 381
 - MCPWM_FAIL1_CMP1_H, 381

MCPWM_FAIL1_CMP1_L, 381
MCPWM_FAIL1_OFF, 381
MCPWM_FAIL_IO_HIGH, 381
MCPWM_FAIL_IO_LOW, 382
MCPWM_FAIL_IO_OFF, 382
MCPWM_GET_IRQ_FAIL0, 382
MCPWM_GET_IRQ_FAIL1, 382
MCPWM_GET_IRQ_T0, 382
MCPWM_GET_IRQ_T1, 382
MCPWM_GET_IRQ_TH00, 382
MCPWM_GET_IRQ_TH01, 383
MCPWM_GET_IRQ_TH10, 383
MCPWM_GET_IRQ_TH11, 383
MCPWM_GET_IRQ_TH20, 383
MCPWM_GET_IRQ_TH21, 383
MCPWM_GET_IRQ_TMR0, 383
MCPWM_GET_IRQ_TMR1, 383
MCPWM_GET_IRQ_TMR2, 384
MCPWM_GET_IRQ_TMR3, 384
MCPWM_GET_IRQ_UP, 384
MCPWM_GetCountTh, 401
MCPWM_GetCountVal, 402
MCPWM_GetEifFlag, 402
MCPWM_GetErrorIRQFlag, 402
MCPWM_GetIfFlag, 403
MCPWM_GetIRQFlag, 403
MCPWM_IE_T0, 384
MCPWM_IE_T1, 384
MCPWM_IE_TH00, 384
MCPWM_IE_TH01, 384
MCPWM_IE_TH10, 385
MCPWM_IE_TH11, 385
MCPWM_IE_TH20, 385
MCPWM_IE_TH21, 385
MCPWM_IE_TMR0, 385
MCPWM_IE_TMR1, 385
MCPWM_IE_TMR2, 385
MCPWM_IE_TMR3, 385
MCPWM_IE_UP, 386
MCPWM_IF_T0, 386
MCPWM_IF_T1, 386
MCPWM_IF_TH00, 386
MCPWM_IF_TH01, 386
MCPWM_IF_TH10, 386
MCPWM_IF_TH11, 386
MCPWM_IF_TH20, 386
MCPWM_IF_TH21, 387
MCPWM_IF_TMR0, 387
MCPWM_IF_TMR1, 387
MCPWM_IF_TMR2, 387
MCPWM_IF_TMR3, 387
MCPWM_IF_UP, 387
MCPWM_Init, 404
MCPWM_IO_DISABLE, 387
MCPWM_IO_HH_HIGH, 387
MCPWM_IO_HH_HPWM, 388
MCPWM_IO_HH_LOW, 388
MCPWM_IO_HH_LPWM, 388
MCPWM_IO_HH_OFF, 388
MCPWM_IO_HH_PWM, 388
MCPWM_IO_HHS_HIGH, 388
MCPWM_IO_HHS_HPWM, 388
MCPWM_IO_HHS_LOW, 388
MCPWM_IO_HHS_LPWM, 389
MCPWM_IO_HHS_OFF, 389
MCPWM_IO_HHS_PWM, 389
MCPWM_IO_HL_HIGH, 389
MCPWM_IO_HL_HPWM, 389
MCPWM_IO_HL_LOW, 389
MCPWM_IO_HL_LPWM, 389
MCPWM_IO_HL_OFF, 389
MCPWM_IO_HL_PWM, 390
MCPWM_IO_HLS_HIGH, 390
MCPWM_IO_HLS_HPWM, 390
MCPWM_IO_HLS_LOW, 390
MCPWM_IO_HLS_LPWM, 390
MCPWM_IO_HLS_OFF, 390
MCPWM_IO_HLS_PWM, 390
MCPWM_IO_LH_HIGH, 390
MCPWM_IO_LH_HPWM, 391
MCPWM_IO_LH_LOW, 391
MCPWM_IO_LH_LPWM, 391
MCPWM_IO_LH_OFF, 391
MCPWM_IO_LH_PWM, 391
MCPWM_IO_LHS_HIGH, 391
MCPWM_IO_LHS_HPWM, 391
MCPWM_IO_LHS_LOW, 391
MCPWM_IO_LHS_LPWM, 392
MCPWM_IO_LHS_OFF, 392
MCPWM_IO_LHS_PWM, 392
MCPWM_IO_LL_HIGH, 392
MCPWM_IO_LL_HPWM, 392
MCPWM_IO_LL_LOW, 392
MCPWM_IO_LL_LPWM, 392
MCPWM_IO_LL_OFF, 392
MCPWM_IO_LL_PWM, 393
MCPWM_IO_LLS_HIGH, 393
MCPWM_IO_LLS_HPWM, 393
MCPWM_IO_LLS_LOW, 393
MCPWM_IO_LLS_LPWM, 393
MCPWM_IO_LLS_OFF, 393
MCPWM_IO_LLS_PWM, 393
MCPWM_IO_NP, 393
MCPWM_IO_NS, 394
MCPWM_IO_PN_SW, 394
MCPWM_IO_PP, 394
MCPWM_IO_PS, 394
MCPWM_IO_SCTRLN, 394
MCPWM_IO_SCTRLP, 394
MCPWM_IO_WM, 394
MCPWM_IRQ_CLEAR_FAIL0, 394
MCPWM_IRQ_CLEAR_FAIL1, 395
MCPWM_IRQ_CLEAR_T0, 395
MCPWM_IRQ_CLEAR_T1, 395
MCPWM_IRQ_CLEAR_TH00, 395
MCPWM_IRQ_CLEAR_TH01, 395

- MCPWM_IRQ_CLEAR_TH10, 395
- MCPWM_IRQ_CLEAR_TH11, 395
- MCPWM_IRQ_CLEAR_TH20, 396
- MCPWM_IRQ_CLEAR_TH21, 396
- MCPWM_IRQ_CLEAR_TMR0, 396
- MCPWM_IRQ_CLEAR_TMR1, 396
- MCPWM_IRQ_CLEAR_TMR2, 396
- MCPWM_IRQ_CLEAR_TMR3, 396
- MCPWM_IRQ_CLEAR_UP, 396
- MCPWM_IRQ_FAIL0, 397
- MCPWM_IRQ_FAIL1, 397
- MCPWM_IRQ_T0, 397
- MCPWM_IRQ_T1, 397
- MCPWM_IRQ_TH00, 397
- MCPWM_IRQ_TH01, 397
- MCPWM_IRQ_TH10, 397
- MCPWM_IRQ_TH11, 398
- MCPWM_IRQ_TH20, 398
- MCPWM_IRQ_TH21, 398
- MCPWM_IRQ_TMR0, 398
- MCPWM_IRQ_TMR1, 398
- MCPWM_IRQ_TMR2, 398
- MCPWM_IRQ_TMR3, 398
- MCPWM_IRQ_UP, 399
- MCPWM_Reset, 404
- MCPWM_SetIOConfig, 405
- MCPWM_SetOutputState, 405
- MCPWM_SetOutputVal, 406
- MCPWM_SetTMR, 406
- MCPWM_StartCount, 407
- MCPWM_StopCount, 407
- MCPWM_StructInit, 407
- MCPWM_TR_T0, 399
- MCPWM_TR_T1, 399
- lks32mc09x_opa.c
 - OPA_Disable, 663
 - OPA_Enable, 664
 - OPA_Init, 664
 - OPA_OutCmpSignal, 665
 - OPA_SetGain, 665
 - OPA_SetIt, 667
 - OPA_StructInit, 667
- lks32mc09x_opa.h
 - OPA_CHANNEL_0, 414
 - OPA_CHANNEL_1, 414
 - OPA_CHANNEL_2, 414
 - OPA_CHANNEL_3, 415
 - OPA_Disable, 417
 - OPA_Enable, 418
 - OPA_GAIN_16, 415
 - OPA_GAIN_33, 415
 - OPA_GAIN_4, 415
 - OPA_GAIN_8, 415
 - OPA_Init, 418
 - OPA_IT_1_0, 415
 - OPA_IT_1_2, 416
 - OPA_IT_1_5, 416
 - OPA_IT_2_0, 416
 - OPA_OUT_CMP_OUT0N, 416
 - OPA_OUT_CMP_OUT0P, 416
 - OPA_OUT_CMP_OUT1N, 416
 - OPA_OUT_CMP_OUT1P, 417
 - OPA_OUT_CMP_OUT2N, 417
 - OPA_OUT_CMP_OUT2P, 417
 - OPA_OUT_CMP_OUT3N, 417
 - OPA_OUT_CMP_OUT3P, 417
 - OPA_OutCmpSignal, 419
 - OPA_SetGain, 419
 - OPA_SetIt, 421
 - OPA_StructInit, 421
- lks32mc09x_qep.c
 - QEP_ClearIRQFlag, 670
 - QEP_Cmd, 670
 - QEP_GetCount, 671
 - QEP_GetFlag, 671
 - QEP_GetIRQFlag, 671
 - QEP_Init, 672
 - QEP_SetMode, 672
 - QEP_StructInit, 673
- lks32mc09x_qep.h
 - QEP_ClearIRQFlag, 429
 - QEP_CLK_DIV1, 425
 - QEP_CLK_DIV128, 425
 - QEP_CLK_DIV16, 425
 - QEP_CLK_DIV2, 425
 - QEP_CLK_DIV32, 426
 - QEP_CLK_DIV4, 426
 - QEP_CLK_DIV64, 426
 - QEP_CLK_DIV8, 426
 - QEP_Cmd, 429
 - QEP_GetCount, 430
 - QEP_GetFlag, 430
 - QEP_GetIRQFlag, 430
 - QEP_IF_OVERFLOW, 426
 - QEP_IF_UNDERFLOW, 426
 - QEP_Init, 431
 - QEP_IRQEna_None, 427
 - QEP_IRQEna_OverFlow, 427
 - QEP_IRQEna_UnderFlow, 427
 - QEP_Mode_CCWCW_UpDown, 427
 - QEP_Mode_CCWSIGN_UpDown, 427
 - QEP_Mode_T1, 427
 - QEP_Mode_T1_T2, 428
 - QEP_SetMode, 431
 - QEP_StructInit, 432
 - QEP_ZC_EDEG_ALL, 428
 - QEP_ZC_EDEG_N, 428
 - QEP_ZC_EDEG_P, 428
 - QEP_ZC_LEVEL_H, 428
 - QEP_ZC_LEVEL_L, 428
 - QEP_ZC_NONE, 429
- lks32mc09x_reg.h
 - ADC, 436
 - ADC0, 436
 - ADC1, 436
 - AON, 436

AON0, 436
CL0, 436
CLU, 436
CLU0, 437
CMP, 437
CMP0, 437
CRC0, 437
DMA0, 437
DMA1, 437
DMA2, 437
DMA3, 437
DMA_CHN0, 438
DMA_CHN1, 438
DMA_CHN2, 438
DMA_CHN3, 438
DMA_TOP, 438
DMAREG, 438
DSP, 438
DSP0, 438
DSP_REG, 439
EXTI, 439
EXTI0, 439
FLASH, 439
FLASH0, 439
GPIO0, 439
GPIO1, 439
GPIO2, 439
HALL, 440
HALL0, 440
I2C, 440
I2C0, 440
IWDG, 440
IWDG0, 440
MCPWM, 440
MCPWM0, 440
QEP, 441
QEP0, 441
REG_IF_WRITE, 441
REG_READ, 441
REG_READ_BIT, 441
REG_READ_BIT_POS_MASK, 441
REG_RESET, 441
REG_SET, 442
REG_WRITE, 442
REG_WRITE_BIT, 442
REG_WRITE_BIT_POS_MASK, 442
SIF, 442
SIF0, 442
SPI, 443
SPI0, 443
SYS, 443
SYS0, 443
TIMER0, 443
TIMER1, 443
TIMER16_0, 443
TIMER16_1, 443
TIMER2, 444
TIMER32_0, 444
UART, 444
UART0, 444
UART1, 444
UART2, 444
UTIMER0, 444
UTIMER1, 444
UTIMER16_0, 445
UTIMER16_1, 445
UTIMER2, 445
UTIMER32_0, 445
lks32mc09x_sif.c
SIF_Clear_Irq, 675
SIF_DMA_IsEnable, 675
SIF_Init, 676
SIF_Irq_IsEnable, 676
SIF_Senddata, 677
SIF_StructInit, 677
SIF_TH_set, 677
SIF_Timebase_set, 678
lks32mc09x_sif.h
SIF_Clear_Irq, 452
SIF_DMA_Enable, 452
SIF_DMA_IsEnable, 452
SIF_IE_Enable, 452
SIF_IF_CLEAR, 452
SIF_Init, 453
SIF_Irq_IsEnable, 453
SIF_Senddata, 453
SIF_StructInit, 454
SIF_TH_set, 454
SIF_Timebase_set, 454
lks32mc09x_spi.c
SPI_ClearIRQFlag, 680
SPI_GetIRQFlag, 680
SPI_Init, 680
SPI_ReadData, 681
SPI_SendData, 681
SPI_StructInit, 682
lks32mc09x_spi.h
SPI_ClearIRQFlag, 460
SPI_DMA_DISABLE, 457
SPI_DMA_ENABLE, 457
SPI_Duplex, 459
SPI_FIRSTSEND_LSB, 458
SPI_FIRSTSEND_MSB, 458
SPI_Full, 459
SPI_GetIRQFlag, 460
SPI_Half_Rx, 459
SPI_Half_Tx, 459
SPI_IF_DataOver, 458
SPI_IF_SSErr, 458
SPI_IF_TranDone, 458
SPI_Init, 461
SPI_IRQEna_DataOver, 458
SPI_IRQEna_Enable, 459
SPI_IRQEna_SSErr, 459
SPI_IRQEna_TranDone, 459
SPI_Master, 460

- SPI_Mode, 460
- SPI_ReadData, 461
- SPI_SendData, 461
- SPI_Slave, 460
- SPI_StructInit, 462
- lks32mc09x_sys.c
 - SYS_I2CClkDiv, 684
 - SYS_MclkChoice, 684
 - SYS_ModuleClockCmd, 685
 - SYS_ReadMcuClk, 686
 - SYS_ResetIO, 687
 - SYS_ResetPeripheral, 687
 - SYS_ResetPeripheralInternal, 688
 - SYS_SetPowerFailureDetection, 688
 - SYS_SoftReset_all, 689
 - SYS_SoftResetModule, 689
 - SYS_SPIClkDiv, 690
 - SYS_UartClkDiv, 690
 - SYS_WaitPowerReady, 691
- lks32mc09x_sys.h
 - SYS_I2CClkDiv, 470
 - SYS_MCLK_RC_12M, 465
 - SYS_MCLK_RC_24M, 465
 - SYS_MCLK_RC_48M, 465
 - SYS_MCLK_RC_8M, 465
 - SYS_MCLK_RC_96M, 465
 - SYS_MCLK_XTAL_12M, 465
 - SYS_MCLK_XTAL_24M, 466
 - SYS_MCLK_XTAL_48M, 466
 - SYS_MCLK_XTAL_96M, 466
 - SYS_MclkChoice, 470
 - SYS_MODULE_ADC0, 466
 - SYS_MODULE_ADC1, 466
 - SYS_MODULE_CL0, 466
 - SYS_MODULE_CMP, 466
 - SYS_MODULE_CRC0, 466
 - SYS_MODULE_DMA0, 467
 - SYS_MODULE_DSP0, 467
 - SYS_MODULE_EEPROM, 467
 - SYS_MODULE_GPIO, 467
 - SYS_MODULE_HALL0, 467
 - SYS_MODULE_I2C0, 467
 - SYS_MODULE_MCPWM0, 467
 - SYS_MODULE_QEP0, 467
 - SYS_MODULE_SIF0, 468
 - SYS_MODULE_SPI0, 468
 - SYS_MODULE_TIMER0, 468
 - SYS_MODULE_TIMER1, 468
 - SYS_MODULE_TIMER2, 468
 - SYS_MODULE_UART0, 468
 - SYS_MODULE_UART1, 468
 - SYS_MODULE_UART2, 468
 - SYS_ModuleClockCmd, 471
 - SYS_PVDSEL_2.8V, 469
 - SYS_PVDSEL_3.49V, 469
 - SYS_PVDSEL_3.5V, 469
 - SYS_PVDSEL_3.74V, 469
 - SYS_PVDSEL_3.75V, 469
- SYS_PVDSEL_4.0V, 469
- SYS_PVDSEL_DAC, 469
- SYS_PVDSEL_OFF, 469
- SYS_ReadMcuClk, 472
- SYS_ResetIO, 473
- SYS_ResetPeripheral, 473
- SYS_ResetPeripheralInternal, 474
- SYS_SetPowerFailureDetection, 474
- SYS_SoftReset_all, 475
- SYS_SoftResetModule, 475
- SYS_SPIClkDiv, 476
- SYS_UartClkDiv, 476
- SYS_WaitPowerReady, 478
- lks32mc09x_timer.c
 - TIMER_ClearIRQFlag, 696
 - TIMER_CMP0, 696
 - TIMER_CMP1, 696
 - TIMER_Disable, 697
 - TIMER_Enable, 697
 - TIMER_GetFlag, 698
 - TIMER_GetIRQFlag, 698
 - TIMER_Init, 699
 - TIMER_SetOutputState, 699
 - TIMER_StructInit, 700
 - TIMER_Timer_GetCMPT0, 700
 - TIMER_Timer_GetCMPT1, 700
 - TIMER_Timer_GetCount, 701
 - TIMER_TimerCmd, 701
- lks32mc09x_timer.h
 - TIMER_CH_MODE_CAP, 482
 - TIMER_CH_MODE_Capture, 482
 - TIMER_CH_MODE_CMP, 482
 - TIMER_CH_MODE_Comparison, 483
 - TIMER_ClearIRQFlag, 484
 - TIMER_CMP0, 484
 - TIMER_CMP1, 484
 - TIMER_Disable, 485
 - TIMER_Enable, 485
 - TIMER_GetFlag, 486
 - TIMER_GetIRQFlag, 486
 - TIMER_IF_CH0, 483
 - TIMER_IF_CH1, 483
 - TIMER_IF_FAIL, 483
 - TIMER_IF_ZC, 483
 - TIMER_Init, 487
 - TIMER_SetOutputState, 487
 - TIMER_StatisticsStart, 483
 - TIMER_StatisticsStop, 483
 - TIMER_StructInit, 488
 - TIMER_Timer_GetCMPT0, 488
 - TIMER_Timer_GetCMPT1, 488
 - TIMER_Timer_GetCount, 489
 - TIMER_TimerCmd, 489
- lks32mc09x_tmp.c
 - tmp_b, 707
 - TMP_Disable, 705
 - TMP_Enable, 705
 - TMP_GetCurrentTemperature, 705

- TMP_Init, 706
- tmp_k, 707
- TMP_Reset, 706
- lks32mc09x_tmp.h
 - TMP_Disable, 492
 - TMP_Enable, 492
 - TMP_GetCurrentTemperature, 492
 - TMP_Init, 493
 - TMP_Reset, 493
- lks32mc09x_trim.h
 - NVR_ADDR_ADC_0_AMC0, 496
 - NVR_ADDR_ADC_0_AMC1, 496
 - NVR_ADDR_ADC_0_DC0, 496
 - NVR_ADDR_ADC_0_DC1, 496
 - NVR_ADDR_ADC_1_AMC0, 496
 - NVR_ADDR_ADC_1_AMC1, 496
 - NVR_ADDR_ADC_1_DC0, 496
 - NVR_ADDR_ADC_1_DC1, 496
 - NVR_ADDR_AFE_REG8, 497
 - NVR_ADDR_AFE_REG9, 497
 - NVR_ADDR_AFE_REGA, 497
 - NVR_ADDR_AFE_REGB, 497
 - NVR_ADDR_AFE_REGC, 497
 - NVR_ADDR_AMTE_CNT, 497
 - NVR_ADDR_AMTE_TMP, 497
 - NVR_ADDR_AMTE_VERSION, 497
 - NVR_ADDR_CHIP_BOOT_VERSION, 498
 - NVR_ADDR_CHIP_FLASH_SIZE, 498
 - NVR_ADDR_CHIP_ID0, 498
 - NVR_ADDR_CHIP_ID1, 498
 - NVR_ADDR_CHIP_ID2, 498
 - NVR_ADDR_CHIP_ID3, 498
 - NVR_ADDR_CHIP_ID4, 498
 - NVR_ADDR_CHIP_MODEL, 498
 - NVR_ADDR_DAC_0_AMC_1_20V, 499
 - NVR_ADDR_DAC_0_AMC_4_8V, 499
 - NVR_ADDR_DAC_0_DC_1_20V, 499
 - NVR_ADDR_DAC_0_DC_4_8V, 499
 - NVR_ADDR_DAC_1_AMC_1_20V, 499
 - NVR_ADDR_DAC_1_AMC_3_0V, 499
 - NVR_ADDR_DAC_1_DC_1_20V, 499
 - NVR_ADDR_DAC_1_DC_3_0V, 499
 - NVR_ADDR_OPA_0, 500
 - NVR_ADDR_OPA_0_160K_40K_GAIN, 500
 - NVR_ADDR_OPA_0_160K_40K_RES, 500
 - NVR_ADDR_OPA_0_320K_40K_GAIN, 500
 - NVR_ADDR_OPA_0_320K_40K_RES, 500
 - NVR_ADDR_OPA_0_640K_40K_GAIN, 500
 - NVR_ADDR_OPA_0_640K_40K_RES, 500
 - NVR_ADDR_OPA_0_660K_20K_GAIN, 500
 - NVR_ADDR_OPA_0_660K_20K_RES, 501
 - NVR_ADDR_OPA_0_OFFSET_H, 501
 - NVR_ADDR_OPA_0_OFFSET_L, 501
 - NVR_ADDR_OPA_1, 501
 - NVR_ADDR_OPA_1_160K_40K_GAIN, 501
 - NVR_ADDR_OPA_1_160K_40K_RES, 501
 - NVR_ADDR_OPA_1_320K_40K_GAIN, 501
 - NVR_ADDR_OPA_1_320K_40K_RES, 501
 - NVR_ADDR_OPA_1_640K_40K_GAIN, 502
 - NVR_ADDR_OPA_1_640K_40K_RES, 502
 - NVR_ADDR_OPA_1_660K_20K_GAIN, 502
 - NVR_ADDR_OPA_1_660K_20K_RES, 502
 - NVR_ADDR_OPA_1_OFFSET_H, 502
 - NVR_ADDR_OPA_1_OFFSET_L, 502
 - NVR_ADDR_OPA_2, 502
 - NVR_ADDR_OPA_2_160K_40K_GAIN, 502
 - NVR_ADDR_OPA_2_160K_40K_RES, 503
 - NVR_ADDR_OPA_2_320K_40K_GAIN, 503
 - NVR_ADDR_OPA_2_320K_40K_RES, 503
 - NVR_ADDR_OPA_2_640K_40K_GAIN, 503
 - NVR_ADDR_OPA_2_640K_40K_RES, 503
 - NVR_ADDR_OPA_2_660K_20K_GAIN, 503
 - NVR_ADDR_OPA_2_660K_20K_RES, 503
 - NVR_ADDR_OPA_2_OFFSET_H, 503
 - NVR_ADDR_OPA_2_OFFSET_L, 504
 - NVR_ADDR_OPA_3, 504
 - NVR_ADDR_OPA_3_160K_40K_GAIN, 504
 - NVR_ADDR_OPA_3_160K_40K_RES, 504
 - NVR_ADDR_OPA_3_320K_40K_GAIN, 504
 - NVR_ADDR_OPA_3_320K_40K_RES, 504
 - NVR_ADDR_OPA_3_640K_40K_GAIN, 504
 - NVR_ADDR_OPA_3_640K_40K_RES, 504
 - NVR_ADDR_OPA_3_660K_20K_GAIN, 505
 - NVR_ADDR_OPA_3_660K_20K_RES, 505
 - NVR_ADDR_OPA_3_OFFSET_H, 505
 - NVR_ADDR_OPA_3_OFFSET_L, 505
 - NVR_ADDR_TMP_B, 505
 - NVR_ADDR_TMP_K, 505
 - TRIM_Read, 506
- lks32mc09x_uart.c
 - UART_ClearIRQFlag, 709
 - UART_GetIRQFlag, 709
 - UART_Init, 709
 - UART_ReadBaud, 710
 - UART_ReadData, 710
 - UART_RecvnData, 711
 - UART_SendAddr_485, 711
 - UART_SendData, 712
 - UART_SendData_485, 713
 - UART_SendnData, 713
 - UART_SetBaud, 714
 - UART_SetClkDiv, 715
 - UART_StructInit, 715
- lks32mc09x_uart.h
 - RX_DMA_RE, 510
 - TX_BUF_DMA_RE, 510
 - TX_DMA_RE, 511
 - UART_ClearIRQFlag, 516
 - UART_FIRSTSEND_LSB, 511
 - UART_FIRSTSEND_MSB, 511
 - UART_GetIRQFlag, 516
 - UART_IF_ABD, 511
 - UART_IF_CheckError, 511
 - UART_IF_IDLE, 511
 - UART_IF_LBD, 512
 - UART_IF_RcvOver, 512

- UART_IF_RX_OV, 512
- UART_IF_SendBufEmpty, 512
- UART_IF_SendOver, 512
- UART_IF_StopError, 512
- UART_IF_TX_OV, 513
- UART_Init, 516
- UART_IRQEna_ABD, 513
- UART_IRQEna_CheckError, 513
- UART_IRQEna_IDLE, 513
- UART_IRQEna_LBD, 513
- UART_IRQEna_RcvOver, 513
- UART_IRQEna_RX_OV, 514
- UART_IRQEna_SendBuffEmpty, 514
- UART_IRQEna_SendOver, 514
- UART_IRQEna_StopError, 514
- UART_IRQEna_TX_OV, 514
- UART_Parity_EVEN, 515
- UART_Parity_NO, 515
- UART_Parity_ODD, 515
- UART_ParityMode, 515
- UART_ReadData, 517
- UART_RecvnData, 518
- UART_SendAddr_485, 518
- UART_SendData, 519
- UART_SendData_485, 520
- UART_SendnData, 521
- UART_SetBaud, 521
- UART_SetClkDiv, 522
- UART_STOPBITS_1b, 514
- UART_STOPBITS_2b, 515
- UART_StructInit, 523
- UART_WORDLENGTH_8b, 515
- UART_WORDLENGTH_9b, 515
- lks32mc09x_wake.c
 - WAKE_EVTCLRSleep, 718
 - WAKE_FallSleep, 718
 - WAKE_SetWakeInterval, 719
 - WAKE_SetWakeIO, 719
- lks32mc09x_wake.h
 - SYS_WakeInterval_025s, 527
 - SYS_WakeInterval_05s, 527
 - SYS_WakeInterval_16s, 527
 - SYS_WakeInterval_1s, 527
 - SYS_WakeInterval_2s, 527
 - SYS_WakeInterval_32s, 527
 - SYS_WakeInterval_4s, 527
 - SYS_WakeInterval_8s, 528
 - WAKE_EVTCLRSleep, 531
 - WAKE_FallSleep, 531
 - WAKE_FLT_DI, 528
 - WAKE_FLT_EN, 528
 - WAKE_IO_CLUOUT0, 528
 - WAKE_IO_CLUOUT1, 528
 - WAKE_IO_CLUOUT2, 528
 - WAKE_IO_CLUOUT3, 529
 - WAKE_IO_P0_0, 529
 - WAKE_IO_P0_12, 529
 - WAKE_IO_P0_13, 529
 - WAKE_IO_P0_14, 529
 - WAKE_IO_P0_2, 529
 - WAKE_IO_P2_0, 530
 - WAKE_IO_P2_4, 530
 - WAKE_IO_P2_7, 530
 - WAKE_SetWakeInterval, 531
 - WAKE_SetWakeIO, 531
 - WAKE_TRIG_HIGH, 530
 - WAKE_TRIG_LOW, 530
 - WAKE_TRIG_x, 530
- LTH
 - ADC_InitTypeDef, 7
 - ADC_TypeDef, 15
- Match485Addr
 - UART_InitTypeDef, 122
- MCPWM
 - lks32mc09x_reg.h, 440
- MCPWM0
 - lks32mc09x_reg.h, 440
- MCPWM_AUEN_ALL
 - lks32mc09x_mcpwm.h, 374
- MCPWM_AUEN_CNT0
 - lks32mc09x_mcpwm.h, 374
- MCPWM_AUEN_DEFAULT
 - lks32mc09x_mcpwm.h, 375
- MCPWM_AUEN_IO
 - lks32mc09x_mcpwm.h, 375
- MCPWM_AUEN_IO012
 - lks32mc09x_mcpwm.h, 375
- MCPWM_AUEN_IO3
 - lks32mc09x_mcpwm.h, 375
- MCPWM_AUEN_TH0
 - lks32mc09x_mcpwm.h, 375
- MCPWM_AUEN_TH00
 - lks32mc09x_mcpwm.h, 375
- MCPWM_AUEN_TH01
 - lks32mc09x_mcpwm.h, 375
- MCPWM_AUEN_TH10
 - lks32mc09x_mcpwm.h, 375
- MCPWM_AUEN_TH11
 - lks32mc09x_mcpwm.h, 376
- MCPWM_AUEN_TH20
 - lks32mc09x_mcpwm.h, 376
- MCPWM_AUEN_TH21
 - lks32mc09x_mcpwm.h, 376
- MCPWM_AUEN_TH30
 - lks32mc09x_mcpwm.h, 376
- MCPWM_AUEN_TH31
 - lks32mc09x_mcpwm.h, 376
- MCPWM_AUEN_TMR0
 - lks32mc09x_mcpwm.h, 376
- MCPWM_AUEN_TMR1
 - lks32mc09x_mcpwm.h, 376
- MCPWM_AUEN_TMR2
 - lks32mc09x_mcpwm.h, 376
- MCPWM_AUEN_TMR3
 - lks32mc09x_mcpwm.h, 377
- MCPWM_ClearErrorIRQFlag

lks32mc09x_mcpwm.c, 650
 lks32mc09x_mcpwm.h, 399
 MCPWM_ClearIRQFlag
 lks32mc09x_mcpwm.c, 651
 lks32mc09x_mcpwm.h, 399
 MCPWM_CLK_DIV_1
 lks32mc09x_mcpwm.h, 377
 MCPWM_CLK_DIV_128
 lks32mc09x_mcpwm.h, 377
 MCPWM_CLK_DIV_16
 lks32mc09x_mcpwm.h, 377
 MCPWM_CLK_DIV_2
 lks32mc09x_mcpwm.h, 377
 MCPWM_CLK_DIV_32
 lks32mc09x_mcpwm.h, 377
 MCPWM_CLK_DIV_4
 lks32mc09x_mcpwm.h, 377
 MCPWM_CLK_DIV_64
 lks32mc09x_mcpwm.h, 377
 MCPWM_CLK_DIV_8
 lks32mc09x_mcpwm.h, 378
 MCPWM_COUNT_EVT_TIMER0_CMP0
 lks32mc09x_mcpwm.h, 378
 MCPWM_COUNT_EVT_TIMER0_CMP1
 lks32mc09x_mcpwm.h, 378
 MCPWM_COUNT_EVT_TIMER1_CMP0
 lks32mc09x_mcpwm.h, 378
 MCPWM_COUNT_EVT_TIMER1_CMP1
 lks32mc09x_mcpwm.h, 378
 MCPWM_COUNT_EVT_TIMER2_CMP0
 lks32mc09x_mcpwm.h, 378
 MCPWM_COUNT_EVT_TIMER2_CMP1
 lks32mc09x_mcpwm.h, 378
 MCPWM_COUNT_OFF
 lks32mc09x_mcpwm.h, 378
 MCPWM_COUNT_START
 lks32mc09x_mcpwm.h, 379
 mcpwm_default_io
 lks32mc09x_mcpwm.c, 659
 MCPWM_Disable
 lks32mc09x_mcpwm.c, 651
 lks32mc09x_mcpwm.h, 400
 MCPWM_EIE_FAIL0
 lks32mc09x_mcpwm.h, 379
 MCPWM_EIE_FAIL1
 lks32mc09x_mcpwm.h, 379
 MCPWM_EIF_FAIL0
 lks32mc09x_mcpwm.h, 379
 MCPWM_EIF_FAIL1
 lks32mc09x_mcpwm.h, 379
 MCPWM_Enable
 lks32mc09x_mcpwm.c, 652
 lks32mc09x_mcpwm.h, 400
 MCPWM_FAIL0_BKIN0_H
 lks32mc09x_mcpwm.h, 379
 MCPWM_FAIL0_BKIN0_L
 lks32mc09x_mcpwm.h, 379
 MCPWM_FAIL0_CLU_OUT0_H
 lks32mc09x_mcpwm.h, 379
 MCPWM_FAIL0_CLU_OUT0_L
 lks32mc09x_mcpwm.h, 380
 MCPWM_FAIL0_CLU_OUT1_H
 lks32mc09x_mcpwm.h, 380
 MCPWM_FAIL0_CLU_OUT1_L
 lks32mc09x_mcpwm.h, 380
 MCPWM_FAIL0_CMP0_H
 lks32mc09x_mcpwm.h, 380
 MCPWM_FAIL0_CMP0_L
 lks32mc09x_mcpwm.h, 380
 MCPWM_FAIL0_OFF
 lks32mc09x_mcpwm.h, 380
 MCPWM_FAIL1_BKIN1_H
 lks32mc09x_mcpwm.h, 380
 MCPWM_FAIL1_BKIN1_L
 lks32mc09x_mcpwm.h, 380
 MCPWM_FAIL1_CLU_OUT2_H
 lks32mc09x_mcpwm.h, 381
 MCPWM_FAIL1_CLU_OUT2_L
 lks32mc09x_mcpwm.h, 381
 MCPWM_FAIL1_CLU_OUT3_H
 lks32mc09x_mcpwm.h, 381
 MCPWM_FAIL1_CLU_OUT3_L
 lks32mc09x_mcpwm.h, 381
 MCPWM_FAIL1_CMP1_H
 lks32mc09x_mcpwm.h, 381
 MCPWM_FAIL1_CMP1_L
 lks32mc09x_mcpwm.h, 381
 MCPWM_FAIL1_OFF
 lks32mc09x_mcpwm.h, 381
 MCPWM_FAIL_IO_HIGH
 lks32mc09x_mcpwm.h, 381
 MCPWM_FAIL_IO_LOW
 lks32mc09x_mcpwm.h, 382
 MCPWM_FAIL_IO_OFF
 lks32mc09x_mcpwm.h, 382
 MCPWM_GET_IRQ_FAIL0
 lks32mc09x_mcpwm.h, 382
 MCPWM_GET_IRQ_FAIL1
 lks32mc09x_mcpwm.h, 382
 MCPWM_GET_IRQ_T0
 lks32mc09x_mcpwm.h, 382
 MCPWM_GET_IRQ_T1
 lks32mc09x_mcpwm.h, 382
 MCPWM_GET_IRQ_TH00
 lks32mc09x_mcpwm.h, 382
 MCPWM_GET_IRQ_TH01
 lks32mc09x_mcpwm.h, 383
 MCPWM_GET_IRQ_TH10
 lks32mc09x_mcpwm.h, 383
 MCPWM_GET_IRQ_TH11
 lks32mc09x_mcpwm.h, 383
 MCPWM_GET_IRQ_TH20
 lks32mc09x_mcpwm.h, 383
 MCPWM_GET_IRQ_TH21
 lks32mc09x_mcpwm.h, 383
 MCPWM_GET_IRQ_TMR0

lks32mc09x_mcpwm.h, 383
 MCPWM_GET_IRQ.TMR1
 lks32mc09x_mcpwm.h, 383
 MCPWM_GET_IRQ.TMR2
 lks32mc09x_mcpwm.h, 384
 MCPWM_GET_IRQ.TMR3
 lks32mc09x_mcpwm.h, 384
 MCPWM_GET_IRQ.UP
 lks32mc09x_mcpwm.h, 384
 MCPWM_GetCountTh
 lks32mc09x_mcpwm.c, 652
 lks32mc09x_mcpwm.h, 401
 MCPWM_GetCountVal
 lks32mc09x_mcpwm.c, 653
 lks32mc09x_mcpwm.h, 402
 MCPWM_GetEifFlag
 lks32mc09x_mcpwm.c, 653
 lks32mc09x_mcpwm.h, 402
 MCPWM_GetErrorIRQFlag
 lks32mc09x_mcpwm.c, 654
 lks32mc09x_mcpwm.h, 402
 MCPWM_GetIfFlag
 lks32mc09x_mcpwm.c, 654
 lks32mc09x_mcpwm.h, 403
 MCPWM_GetIRQFlag
 lks32mc09x_mcpwm.c, 654
 lks32mc09x_mcpwm.h, 403
 MCPWM_IE.T0
 lks32mc09x_mcpwm.h, 384
 MCPWM_IE.T1
 lks32mc09x_mcpwm.h, 384
 MCPWM_IE.TH00
 lks32mc09x_mcpwm.h, 384
 MCPWM_IE.TH01
 lks32mc09x_mcpwm.h, 384
 MCPWM_IE.TH10
 lks32mc09x_mcpwm.h, 385
 MCPWM_IE.TH11
 lks32mc09x_mcpwm.h, 385
 MCPWM_IE.TH20
 lks32mc09x_mcpwm.h, 385
 MCPWM_IE.TH21
 lks32mc09x_mcpwm.h, 385
 MCPWM_IE.TMR0
 lks32mc09x_mcpwm.h, 385
 MCPWM_IE.TMR1
 lks32mc09x_mcpwm.h, 385
 MCPWM_IE.TMR2
 lks32mc09x_mcpwm.h, 385
 MCPWM_IE.TMR3
 lks32mc09x_mcpwm.h, 385
 MCPWM_IE.UP
 lks32mc09x_mcpwm.h, 386
 MCPWM_IF.T0
 lks32mc09x_mcpwm.h, 386
 MCPWM_IF.T1
 lks32mc09x_mcpwm.h, 386
 MCPWM_IF.TH00
 lks32mc09x_mcpwm.h, 386
 MCPWM_IF.TH01
 lks32mc09x_mcpwm.h, 386
 MCPWM_IF.TH10
 lks32mc09x_mcpwm.h, 386
 MCPWM_IF.TH11
 lks32mc09x_mcpwm.h, 386
 MCPWM_IF.TH20
 lks32mc09x_mcpwm.h, 386
 MCPWM_IF.TH21
 lks32mc09x_mcpwm.h, 387
 MCPWM_IF.TMR0
 lks32mc09x_mcpwm.h, 387
 MCPWM_IF.TMR1
 lks32mc09x_mcpwm.h, 387
 MCPWM_IF.TMR2
 lks32mc09x_mcpwm.h, 387
 MCPWM_IF.TMR3
 lks32mc09x_mcpwm.h, 387
 MCPWM_IF.UP
 lks32mc09x_mcpwm.h, 387
 MCPWM_Init
 lks32mc09x_mcpwm.c, 655
 lks32mc09x_mcpwm.h, 404
 MCPWM_InitTypeDef, 69
 AUEN, 70
 CLK_DIV, 70
 COUNT, 70
 DTHN, 70
 DTHP, 71
 EIE, 71
 EN, 71
 FAIL0, 71
 FAIL1, 71
 FAIL_IO, 71
 FLT_DIV, 71
 IE, 71
 IO0, 72
 IO1, 72
 IO2, 72
 IO3, 72
 RE, 72
 TH, 72
 TMR0, 72
 TMR1, 72
 TMR2, 73
 TMR3, 73
 TR, 73
 MCPWM_IO_DISABLE
 lks32mc09x_mcpwm.h, 387
 MCPWM_IO_HH_HIGH
 lks32mc09x_mcpwm.h, 387
 MCPWM_IO_HH_HPWM
 lks32mc09x_mcpwm.h, 388
 MCPWM_IO_HH_LOW
 lks32mc09x_mcpwm.h, 388
 MCPWM_IO_HH_LPWM
 lks32mc09x_mcpwm.h, 388

MCPWM_IO_HH_OFF
lks32mc09x_mcpwm.h, 388

MCPWM_IO_HH_PWM
lks32mc09x_mcpwm.h, 388

MCPWM_IO_HHS_HIGH
lks32mc09x_mcpwm.h, 388

MCPWM_IO_HHS_HPWM
lks32mc09x_mcpwm.h, 388

MCPWM_IO_HHS_LOW
lks32mc09x_mcpwm.h, 388

MCPWM_IO_HHS_LPWM
lks32mc09x_mcpwm.h, 389

MCPWM_IO_HHS_OFF
lks32mc09x_mcpwm.h, 389

MCPWM_IO_HHS_PWM
lks32mc09x_mcpwm.h, 389

MCPWM_IO_HL_HIGH
lks32mc09x_mcpwm.h, 389

MCPWM_IO_HL_HPWM
lks32mc09x_mcpwm.h, 389

MCPWM_IO_HL_LOW
lks32mc09x_mcpwm.h, 389

MCPWM_IO_HL_LPWM
lks32mc09x_mcpwm.h, 389

MCPWM_IO_HL_OFF
lks32mc09x_mcpwm.h, 389

MCPWM_IO_HL_PWM
lks32mc09x_mcpwm.h, 390

MCPWM_IO_HLS_HIGH
lks32mc09x_mcpwm.h, 390

MCPWM_IO_HLS_HPWM
lks32mc09x_mcpwm.h, 390

MCPWM_IO_HLS_LOW
lks32mc09x_mcpwm.h, 390

MCPWM_IO_HLS_LPWM
lks32mc09x_mcpwm.h, 390

MCPWM_IO_HLS_OFF
lks32mc09x_mcpwm.h, 390

MCPWM_IO_HLS_PWM
lks32mc09x_mcpwm.h, 390

MCPWM_IO_LH_HIGH
lks32mc09x_mcpwm.h, 390

MCPWM_IO_LH_HPWM
lks32mc09x_mcpwm.h, 391

MCPWM_IO_LH_LOW
lks32mc09x_mcpwm.h, 391

MCPWM_IO_LH_LPWM
lks32mc09x_mcpwm.h, 391

MCPWM_IO_LH_OFF
lks32mc09x_mcpwm.h, 391

MCPWM_IO_LH_PWM
lks32mc09x_mcpwm.h, 391

MCPWM_IO_LHS_HIGH
lks32mc09x_mcpwm.h, 391

MCPWM_IO_LHS_HPWM
lks32mc09x_mcpwm.h, 391

MCPWM_IO_LHS_LOW
lks32mc09x_mcpwm.h, 391

MCPWM_IO_LHS_LPWM
lks32mc09x_mcpwm.h, 392

MCPWM_IO_LHS_OFF
lks32mc09x_mcpwm.h, 392

MCPWM_IO_LHS_PWM
lks32mc09x_mcpwm.h, 392

MCPWM_IO_LL_HIGH
lks32mc09x_mcpwm.h, 392

MCPWM_IO_LL_HPWM
lks32mc09x_mcpwm.h, 392

MCPWM_IO_LL_LOW
lks32mc09x_mcpwm.h, 392

MCPWM_IO_LL_LPWM
lks32mc09x_mcpwm.h, 392

MCPWM_IO_LL_OFF
lks32mc09x_mcpwm.h, 392

MCPWM_IO_LL_PWM
lks32mc09x_mcpwm.h, 393

MCPWM_IO_LLS_HIGH
lks32mc09x_mcpwm.h, 393

MCPWM_IO_LLS_HPWM
lks32mc09x_mcpwm.h, 393

MCPWM_IO_LLS_LOW
lks32mc09x_mcpwm.h, 393

MCPWM_IO_LLS_LPWM
lks32mc09x_mcpwm.h, 393

MCPWM_IO_LLS_OFF
lks32mc09x_mcpwm.h, 393

MCPWM_IO_LLS_PWM
lks32mc09x_mcpwm.h, 393

MCPWM_IO_NP
lks32mc09x_mcpwm.h, 393

MCPWM_IO_NS
lks32mc09x_mcpwm.h, 394

MCPWM_IO_PN_SW
lks32mc09x_mcpwm.h, 394

MCPWM_IO_PP
lks32mc09x_mcpwm.h, 394

MCPWM_IO_PS
lks32mc09x_mcpwm.h, 394

MCPWM_IO_SCTRLN
lks32mc09x_mcpwm.h, 394

MCPWM_IO_SCTRLP
lks32mc09x_mcpwm.h, 394

MCPWM_IO_WM
lks32mc09x_mcpwm.h, 394

MCPWM_IRQ_CLEAR_FAIL0
lks32mc09x_mcpwm.h, 394

MCPWM_IRQ_CLEAR_FAIL1
lks32mc09x_mcpwm.h, 395

MCPWM_IRQ_CLEAR_T0
lks32mc09x_mcpwm.h, 395

MCPWM_IRQ_CLEAR_T1
lks32mc09x_mcpwm.h, 395

MCPWM_IRQ_CLEAR_TH00
lks32mc09x_mcpwm.h, 395

MCPWM_IRQ_CLEAR_TH01
lks32mc09x_mcpwm.h, 395

MCPWM_IRQ_CLEAR_TH10
 lks32mc09x_mcpwm.h, 395
 MCPWM_IRQ_CLEAR_TH11
 lks32mc09x_mcpwm.h, 395
 MCPWM_IRQ_CLEAR_TH20
 lks32mc09x_mcpwm.h, 396
 MCPWM_IRQ_CLEAR_TH21
 lks32mc09x_mcpwm.h, 396
 MCPWM_IRQ_CLEAR_TMR0
 lks32mc09x_mcpwm.h, 396
 MCPWM_IRQ_CLEAR_TMR1
 lks32mc09x_mcpwm.h, 396
 MCPWM_IRQ_CLEAR_TMR2
 lks32mc09x_mcpwm.h, 396
 MCPWM_IRQ_CLEAR_TMR3
 lks32mc09x_mcpwm.h, 396
 MCPWM_IRQ_CLEAR_UP
 lks32mc09x_mcpwm.h, 396
 MCPWM_IRQ_FAIL0
 lks32mc09x_mcpwm.h, 397
 MCPWM_IRQ_FAIL1
 lks32mc09x_mcpwm.h, 397
 MCPWM_IRQ_T0
 lks32mc09x_mcpwm.h, 397
 MCPWM_IRQ_T1
 lks32mc09x_mcpwm.h, 397
 MCPWM_IRQ_TH00
 lks32mc09x_mcpwm.h, 397
 MCPWM_IRQ_TH01
 lks32mc09x_mcpwm.h, 397
 MCPWM_IRQ_TH10
 lks32mc09x_mcpwm.h, 397
 MCPWM_IRQ_TH11
 lks32mc09x_mcpwm.h, 398
 MCPWM_IRQ_TH20
 lks32mc09x_mcpwm.h, 398
 MCPWM_IRQ_TH21
 lks32mc09x_mcpwm.h, 398
 MCPWM_IRQ_TMR0
 lks32mc09x_mcpwm.h, 398
 MCPWM_IRQ_TMR1
 lks32mc09x_mcpwm.h, 398
 MCPWM_IRQ_TMR2
 lks32mc09x_mcpwm.h, 398
 MCPWM_IRQ_TMR3
 lks32mc09x_mcpwm.h, 398
 MCPWM_IRQ_UP
 lks32mc09x_mcpwm.h, 399
 MCPWM_Reset
 lks32mc09x_mcpwm.c, 656
 lks32mc09x_mcpwm.h, 404
 MCPWM_SetIOConfig
 lks32mc09x_mcpwm.c, 656
 lks32mc09x_mcpwm.h, 405
 MCPWM_SetOutputState
 lks32mc09x_mcpwm.c, 657
 lks32mc09x_mcpwm.h, 405
 MCPWM_SetOutputVal
 lks32mc09x_mcpwm.c, 657
 lks32mc09x_mcpwm.h, 406
 MCPWM_SetTMR
 lks32mc09x_mcpwm.c, 657
 lks32mc09x_mcpwm.h, 406
 MCPWM_StartCount
 lks32mc09x_mcpwm.c, 658
 lks32mc09x_mcpwm.h, 407
 MCPWM_StopCount
 lks32mc09x_mcpwm.c, 658
 lks32mc09x_mcpwm.h, 407
 MCPWM_StructInit
 lks32mc09x_mcpwm.c, 658
 lks32mc09x_mcpwm.h, 407
 MCPWM_TmpUpdateFaillo
 lks32mc09x_mcpwm.c, 659
 MCPWM_TR_T0
 lks32mc09x_mcpwm.h, 399
 MCPWM_TR_T1
 lks32mc09x_mcpwm.h, 399
 MCPWM_TypeDef, 73
 AUEN, 76
 CH_DEF, 76
 CH_FAIL, 76
 CH_MSK, 76
 CNT0, 77
 DTH00, 77
 DTH01, 77
 EIE, 77
 EIF, 77
 EVT0, 77
 FCNT, 78
 FLT, 78
 IE0, 78
 IF0, 78
 IO01, 79
 IO23, 79
 PP, 79
 PRT, 79
 RE, 80
 RESERVED, 80
 RESERVED1, 80
 RESERVED2, 80
 RESERVED3, 80
 RESERVED4, 80
 RESERVED5, 81
 RESERVED6, 81
 RESERVED7, 81
 RESERVED8, 81
 SDCFG, 81
 SW_FAIL, 81
 TCLK, 81
 TH0, 82
 TH00, 82
 TH01, 82
 TH10, 82
 TH11, 82
 TH20, 82

TH21, [83](#)
 TH30, [83](#)
 TH31, [83](#)
 TMR0, [83](#)
 TMR1, [83](#)
 TMR2, [83](#)
 TMR3, [84](#)
 UPDATE, [84](#)
 Mode
 QEP_InitTypeDef, [87](#)
 SPI_InitTypeDef, [96](#)
 MOE
 TIMER_TimerInitTypeDef, [114](#)
 MSCR
 I2C_TypeDef, [65](#)
 MST_MODE
 I2C_InitTypeDef, [63](#)
 MultiDropEna
 UART_InitTypeDef, [122](#)
 MX
 CLU_TypeDef, [23](#)
 MXA
 CLU_InitTypeDef, [20](#)
 MXB
 CLU_InitTypeDef, [20](#)

 nackflg
 iictmpval, [66](#)
 NADDR
 FLASH_TypeDef, [49](#)
 NCFG
 FLASH_TypeDef, [49](#)
 NERASE
 FLASH_TypeDef, [49](#)
 NKEY
 FLASH_TypeDef, [49](#)
 NMI_TH
 SYS_TypeDef, [105](#)
 NRDATA
 FLASH_TypeDef, [49](#)
 NSMP
 ADC_InitTypeDef, [7](#)
 null
 CRC_InitTypeDef, [31](#)
 NVR_ADDR_ADC_0_AMC0
 lks32mc09x_trim.h, [496](#)
 NVR_ADDR_ADC_0_AMC1
 lks32mc09x_trim.h, [496](#)
 NVR_ADDR_ADC_0_DC0
 lks32mc09x_trim.h, [496](#)
 NVR_ADDR_ADC_0_DC1
 lks32mc09x_trim.h, [496](#)
 NVR_ADDR_ADC_1_AMC0
 lks32mc09x_trim.h, [496](#)
 NVR_ADDR_ADC_1_AMC1
 lks32mc09x_trim.h, [496](#)
 NVR_ADDR_ADC_1_DC0
 lks32mc09x_trim.h, [496](#)
 NVR_ADDR_ADC_1_DC1
 lks32mc09x_trim.h, [496](#)
 NVR_ADDR_AFE_REG8
 lks32mc09x_trim.h, [497](#)
 NVR_ADDR_AFE_REG9
 lks32mc09x_trim.h, [497](#)
 NVR_ADDR_AFE_REGA
 lks32mc09x_trim.h, [497](#)
 NVR_ADDR_AFE_REGB
 lks32mc09x_trim.h, [497](#)
 NVR_ADDR_AFE_REGC
 lks32mc09x_trim.h, [497](#)
 NVR_ADDR_AMTE_CNT
 lks32mc09x_trim.h, [497](#)
 NVR_ADDR_AMTE_TMP
 lks32mc09x_trim.h, [497](#)
 NVR_ADDR_AMTE_VERSION
 lks32mc09x_trim.h, [497](#)
 NVR_ADDR_CHIP_BOOT_VERSION
 lks32mc09x_trim.h, [498](#)
 NVR_ADDR_CHIP_FLASH_SIZE
 lks32mc09x_trim.h, [498](#)
 NVR_ADDR_CHIP_ID0
 lks32mc09x_trim.h, [498](#)
 NVR_ADDR_CHIP_ID1
 lks32mc09x_trim.h, [498](#)
 NVR_ADDR_CHIP_ID2
 lks32mc09x_trim.h, [498](#)
 NVR_ADDR_CHIP_ID3
 lks32mc09x_trim.h, [498](#)
 NVR_ADDR_CHIP_ID4
 lks32mc09x_trim.h, [498](#)
 NVR_ADDR_CHIP_MODEL
 lks32mc09x_trim.h, [498](#)
 NVR_ADDR_DAC_0_AMC_1_20V
 lks32mc09x_trim.h, [499](#)
 NVR_ADDR_DAC_0_AMC_4_8V
 lks32mc09x_trim.h, [499](#)
 NVR_ADDR_DAC_0_DC_1_20V
 lks32mc09x_trim.h, [499](#)
 NVR_ADDR_DAC_0_DC_4_8V
 lks32mc09x_trim.h, [499](#)
 NVR_ADDR_DAC_1_AMC_1_20V
 lks32mc09x_trim.h, [499](#)
 NVR_ADDR_DAC_1_AMC_3_0V
 lks32mc09x_trim.h, [499](#)
 NVR_ADDR_DAC_1_DC_1_20V
 lks32mc09x_trim.h, [499](#)
 NVR_ADDR_DAC_1_DC_3_0V
 lks32mc09x_trim.h, [499](#)
 NVR_ADDR_OPA_0
 lks32mc09x_trim.h, [500](#)
 NVR_ADDR_OPA_0_160K_40K_GAIN
 lks32mc09x_trim.h, [500](#)
 NVR_ADDR_OPA_0_160K_40K_RES
 lks32mc09x_trim.h, [500](#)
 NVR_ADDR_OPA_0_320K_40K_GAIN
 lks32mc09x_trim.h, [500](#)
 NVR_ADDR_OPA_0_320K_40K_RES
 lks32mc09x_trim.h, [500](#)

lks32mc09x_trim.h, [500](#)
 NVR_ADDR_OPA_0_640K_40K_GAIN
 lks32mc09x_trim.h, [500](#)
 NVR_ADDR_OPA_0_640K_40K_RES
 lks32mc09x_trim.h, [500](#)
 NVR_ADDR_OPA_0_660K_20K_GAIN
 lks32mc09x_trim.h, [500](#)
 NVR_ADDR_OPA_0_660K_20K_RES
 lks32mc09x_trim.h, [501](#)
 NVR_ADDR_OPA_0_OFFSET_H
 lks32mc09x_trim.h, [501](#)
 NVR_ADDR_OPA_0_OFFSET_L
 lks32mc09x_trim.h, [501](#)
 NVR_ADDR_OPA_1
 lks32mc09x_trim.h, [501](#)
 NVR_ADDR_OPA_1_160K_40K_GAIN
 lks32mc09x_trim.h, [501](#)
 NVR_ADDR_OPA_1_160K_40K_RES
 lks32mc09x_trim.h, [501](#)
 NVR_ADDR_OPA_1_320K_40K_GAIN
 lks32mc09x_trim.h, [501](#)
 NVR_ADDR_OPA_1_320K_40K_RES
 lks32mc09x_trim.h, [501](#)
 NVR_ADDR_OPA_1_640K_40K_GAIN
 lks32mc09x_trim.h, [502](#)
 NVR_ADDR_OPA_1_640K_40K_RES
 lks32mc09x_trim.h, [502](#)
 NVR_ADDR_OPA_1_660K_20K_GAIN
 lks32mc09x_trim.h, [502](#)
 NVR_ADDR_OPA_1_660K_20K_RES
 lks32mc09x_trim.h, [502](#)
 NVR_ADDR_OPA_1_OFFSET_H
 lks32mc09x_trim.h, [502](#)
 NVR_ADDR_OPA_1_OFFSET_L
 lks32mc09x_trim.h, [502](#)
 NVR_ADDR_OPA_2
 lks32mc09x_trim.h, [502](#)
 NVR_ADDR_OPA_2_160K_40K_GAIN
 lks32mc09x_trim.h, [502](#)
 NVR_ADDR_OPA_2_160K_40K_RES
 lks32mc09x_trim.h, [503](#)
 NVR_ADDR_OPA_2_320K_40K_GAIN
 lks32mc09x_trim.h, [503](#)
 NVR_ADDR_OPA_2_320K_40K_RES
 lks32mc09x_trim.h, [503](#)
 NVR_ADDR_OPA_2_640K_40K_GAIN
 lks32mc09x_trim.h, [503](#)
 NVR_ADDR_OPA_2_640K_40K_RES
 lks32mc09x_trim.h, [503](#)
 NVR_ADDR_OPA_2_660K_20K_GAIN
 lks32mc09x_trim.h, [503](#)
 NVR_ADDR_OPA_2_660K_20K_RES
 lks32mc09x_trim.h, [503](#)
 NVR_ADDR_OPA_2_OFFSET_H
 lks32mc09x_trim.h, [503](#)
 NVR_ADDR_OPA_2_OFFSET_L
 lks32mc09x_trim.h, [504](#)
 NVR_ADDR_OPA_3
 lks32mc09x_trim.h, [504](#)
 NVR_ADDR_OPA_3_160K_40K_GAIN
 lks32mc09x_trim.h, [504](#)
 NVR_ADDR_OPA_3_160K_40K_RES
 lks32mc09x_trim.h, [504](#)
 NVR_ADDR_OPA_3_320K_40K_GAIN
 lks32mc09x_trim.h, [504](#)
 NVR_ADDR_OPA_3_320K_40K_RES
 lks32mc09x_trim.h, [504](#)
 NVR_ADDR_OPA_3_640K_40K_GAIN
 lks32mc09x_trim.h, [504](#)
 NVR_ADDR_OPA_3_640K_40K_RES
 lks32mc09x_trim.h, [504](#)
 NVR_ADDR_OPA_3_660K_20K_GAIN
 lks32mc09x_trim.h, [505](#)
 NVR_ADDR_OPA_3_660K_20K_RES
 lks32mc09x_trim.h, [505](#)
 NVR_ADDR_OPA_3_OFFSET_H
 lks32mc09x_trim.h, [505](#)
 NVR_ADDR_OPA_3_OFFSET_L
 lks32mc09x_trim.h, [505](#)
 NVR_ADDR_TMP_B
 lks32mc09x_trim.h, [505](#)
 NVR_ADDR_TMP_K
 lks32mc09x_trim.h, [505](#)
 NWDATA
 FLASH_TypeDef, [50](#)
 OEN
 CLU_InitTypeDef, [20](#)
 ONE_TRIG
 TIMER_TimerInitTypeDef, [114](#)
 OPA_CHANNEL_0
 lks32mc09x_opa.h, [414](#)
 OPA_CHANNEL_1
 lks32mc09x_opa.h, [414](#)
 OPA_CHANNEL_2
 lks32mc09x_opa.h, [414](#)
 OPA_CHANNEL_3
 lks32mc09x_opa.h, [415](#)
 OPA_CLEna
 OPA_InitTypeDef, [85](#)
 OPA_Disable
 lks32mc09x_opa.c, [663](#)
 lks32mc09x_opa.h, [417](#)
 OPA_Enable
 lks32mc09x_opa.c, [664](#)
 lks32mc09x_opa.h, [418](#)
 OPA_Gain
 OPA_InitTypeDef, [85](#)
 OPA_GAIN_16
 lks32mc09x_opa.h, [415](#)
 OPA_GAIN_33
 lks32mc09x_opa.h, [415](#)
 OPA_GAIN_4
 lks32mc09x_opa.h, [415](#)
 OPA_GAIN_8
 lks32mc09x_opa.h, [415](#)
 OPA_Init

lks32mc09x_opa.c, 664
 lks32mc09x_opa.h, 418
 OPA_InitTypeDef, 84
 OPA_CLEna, 85
 OPA_Gain, 85
 OPA_IT, 85
 OPA_IT
 OPA_InitTypeDef, 85
 OPA_IT_1_0
 lks32mc09x_opa.h, 415
 OPA_IT_1_2
 lks32mc09x_opa.h, 416
 OPA_IT_1_5
 lks32mc09x_opa.h, 416
 OPA_IT_2_0
 lks32mc09x_opa.h, 416
 OPA_OUT_CMP_OUT0N
 lks32mc09x_opa.h, 416
 OPA_OUT_CMP_OUT0P
 lks32mc09x_opa.h, 416
 OPA_OUT_CMP_OUT1N
 lks32mc09x_opa.h, 416
 OPA_OUT_CMP_OUT1P
 lks32mc09x_opa.h, 417
 OPA_OUT_CMP_OUT2N
 lks32mc09x_opa.h, 417
 OPA_OUT_CMP_OUT2P
 lks32mc09x_opa.h, 417
 OPA_OUT_CMP_OUT3N
 lks32mc09x_opa.h, 417
 OPA_OUT_CMP_OUT3P
 lks32mc09x_opa.h, 417
 OPA_OutCmpSignal
 lks32mc09x_opa.c, 665
 lks32mc09x_opa.h, 419
 OPA_SEL
 SYS_TypeDef, 105
 OPA_SetGain
 lks32mc09x_opa.c, 665
 lks32mc09x_opa.h, 419
 OPA_SetIt
 lks32mc09x_opa.c, 667
 lks32mc09x_opa.h, 421
 OPA_StructInit
 lks32mc09x_opa.c, 667
 lks32mc09x_opa.h, 421
 OUT
 CLU_TypeDef, 23
 OUTSEL
 CLU_InitTypeDef, 20
 OverFlow_IRQ_Ena
 HALL_InitTypeDef, 58
 OVSR
 ADC_InitTypeDef, 7
 ParityMode
 UART_InitTypeDef, 122
 PDI
 GPIO_TypeDef, 55
 PDO
 GPIO_TypeDef, 55
 PFLT
 GPIO_TypeDef, 55
 PIE
 GPIO_TypeDef, 56
 PODE
 GPIO_TypeDef, 56
 POE
 GPIO_TypeDef, 56
 POL
 CMP_SingleTypeDef, 27
 CRC_TypeDef, 33
 POLY
 CRC_InitTypeDef, 31
 PP
 MCPWM_TypeDef, 79
 PROTECT
 FLASH_TypeDef, 50
 SYS_TypeDef, 105
 PRT
 MCPWM_TypeDef, 79
 PSW
 IWDG_TypeDef, 69
 PUE
 GPIO_TypeDef, 56
 PWR_CFG
 AON_TypeDef, 18
 QEP
 lks32mc09x_reg.h, 441
 QEPO
 lks32mc09x_reg.h, 441
 QEP_ClearIRQFlag
 lks32mc09x_qep.c, 670
 lks32mc09x_qep.h, 429
 QEP_CLK_DIV1
 lks32mc09x_qep.h, 425
 QEP_CLK_DIV128
 lks32mc09x_qep.h, 425
 QEP_CLK_DIV16
 lks32mc09x_qep.h, 425
 QEP_CLK_DIV2
 lks32mc09x_qep.h, 425
 QEP_CLK_DIV32
 lks32mc09x_qep.h, 426
 QEP_CLK_DIV4
 lks32mc09x_qep.h, 426
 QEP_CLK_DIV64
 lks32mc09x_qep.h, 426
 QEP_CLK_DIV8
 lks32mc09x_qep.h, 426
 QEP_Cmd
 lks32mc09x_qep.c, 670
 lks32mc09x_qep.h, 429
 QEP_GetCount
 lks32mc09x_qep.c, 671
 lks32mc09x_qep.h, 430
 QEP_GetFlag

lks32mc09x_qep.c, 671
 lks32mc09x_qep.h, 430
 QEP_GetIRQFlag
 lks32mc09x_qep.c, 671
 lks32mc09x_qep.h, 430
 QEP_IF_OVERFLOW
 lks32mc09x_qep.h, 426
 QEP_IF_UNDERFLOW
 lks32mc09x_qep.h, 426
 QEP_Init
 lks32mc09x_qep.c, 672
 lks32mc09x_qep.h, 431
 QEP_InitTypeDef, 85
 ClockDiv, 86
 FE_CNT_EN, 86
 Filter, 86
 IRQEna, 86
 Mode, 87
 TH, 87
 ZC, 87
 QEP_IRQEna_None
 lks32mc09x_qep.h, 427
 QEP_IRQEna_OverFlow
 lks32mc09x_qep.h, 427
 QEP_IRQEna_UnderFlow
 lks32mc09x_qep.h, 427
 QEP_Mode_CCWCW_UpDown
 lks32mc09x_qep.h, 427
 QEP_Mode_CCWSIGN_UpDown
 lks32mc09x_qep.h, 427
 QEP_Mode_T1
 lks32mc09x_qep.h, 427
 QEP_Mode_T1_T2
 lks32mc09x_qep.h, 428
 QEP_SetMode
 lks32mc09x_qep.c, 672
 lks32mc09x_qep.h, 431
 QEP_StructInit
 lks32mc09x_qep.c, 673
 lks32mc09x_qep.h, 432
 QEP_TypeDef, 87
 CFG, 88
 CNT, 88
 IE, 88
 IF, 88
 TH, 88
 QEP_ZC_EDEG_ALL
 lks32mc09x_qep.h, 428
 QEP_ZC_EDEG_N
 lks32mc09x_qep.h, 428
 QEP_ZC_EDEG_P
 lks32mc09x_qep.h, 428
 QEP_ZC_LEVEL_H
 lks32mc09x_qep.h, 428
 QEP_ZC_LEVEL_L
 lks32mc09x_qep.h, 428
 QEP_ZC_NONE
 lks32mc09x_qep.h, 429
 QUO
 DSP_TypeDef, 41
 QUO1
 DSP_TypeDef, 41
 RAD
 DSP_TypeDef, 41
 RAM_LOCK
 SYS_TypeDef, 106
 RDATA
 FLASH_TypeDef, 50
 RDY_RLS
 EEPROM_InitTypeDef, 44
 RE
 ADC_InitTypeDef, 8
 CMP_SingleTypeDef, 27
 MCPWM_InitTypeDef, 72
 MCPWM_TypeDef, 80
 UART_TypeDef, 125
 READY
 FLASH_TypeDef, 50
 REFIN
 CRC_InitTypeDef, 31
 REFOUT
 CRC_InitTypeDef, 31
 REG_IF_WRITE
 lks32mc09x_reg.h, 441
 REG_READ
 lks32mc09x_reg.h, 441
 REG_READ_BIT
 lks32mc09x_reg.h, 441
 REG_READ_BIT_POS_MASK
 lks32mc09x_reg.h, 441
 REG_RESET
 lks32mc09x_reg.h, 441
 REG_SET
 lks32mc09x_reg.h, 442
 REG_WRITE
 lks32mc09x_reg.h, 442
 REG_WRITE_BIT
 lks32mc09x_reg.h, 442
 REG_WRITE_BIT_POS_MASK
 lks32mc09x_reg.h, 442
 REM
 DSP_TypeDef, 41
 REN
 DMA_InitTypeDef, 36
 DMA_TypeDef, 38
 RESERVED
 ADC_TypeDef, 15
 DSP_TypeDef, 42
 FLASH_TypeDef, 50
 GPIO_TypeDef, 57
 MCPWM_TypeDef, 80
 SYS_TypeDef, 106
 TIMER_TypeDef, 119
 RESERVED1
 ADC_TypeDef, 15
 DSP_TypeDef, 42

MCPWM_TypeDef, 80
 SYS_TypeDef, 106
 RESERVED2
 ADC_TypeDef, 15
 DSP_TypeDef, 42
 MCPWM_TypeDef, 80
 SYS_TypeDef, 106
 RESERVED3
 ADC_TypeDef, 16
 MCPWM_TypeDef, 80
 SYS_TypeDef, 106
 RESERVED4
 MCPWM_TypeDef, 80
 RESERVED41
 SYS_TypeDef, 106
 RESERVED5
 MCPWM_TypeDef, 81
 SYS_TypeDef, 107
 RESERVED6
 MCPWM_TypeDef, 81
 SYS_TypeDef, 107
 RESERVED7
 MCPWM_TypeDef, 81
 SYS_TypeDef, 107
 RESERVED8
 MCPWM_TypeDef, 81
 RIE
 CLU_InitTypeDef, 21
 RL_EN
 TIMER_TimerInitTypeDef, 115
 RMODE
 DMA_InitTypeDef, 36
 RTH
 IWDG_InitTypeDef, 67
 IWDG_TypeDef, 69
 RX_DATA
 SPI_TypeDef, 98
 RX_DMA_RE
 lks32mc09x_uart.h, 510
 RXD_INV
 UART_InitTypeDef, 122
 rxdata
 iictmpval, 66
 rxdatap
 iictmpval, 66
 S1
 ADC_InitTypeDef, 8
 S2
 ADC_InitTypeDef, 8
 SADR
 DMA_InitTypeDef, 36
 DMA_TypeDef, 38
 SBTW
 DMA_InitTypeDef, 37
 SCR
 I2C_TypeDef, 65
 SDCFG
 MCPWM_TypeDef, 81
 SELN
 CMP_SingleTypeDef, 27
 SELP
 CMP_SingleTypeDef, 27
 SFT_RST
 SYS_TypeDef, 107
 SHADOW
 TIMER_TimerInitTypeDef, 115
 SIF
 lks32mc09x_reg.h, 442
 SIF0
 lks32mc09x_reg.h, 442
 SIF_Clear_Irq
 lks32mc09x_sif.c, 675
 lks32mc09x_sif.h, 452
 SIF_DMA_EN
 SIF_InitTypeDef, 90
 SIF_DMA_Enable
 lks32mc09x_sif.h, 452
 SIF_DMA_IsEnable
 lks32mc09x_sif.c, 675
 lks32mc09x_sif.h, 452
 SIF_DONE
 SIF_InitTypeDef, 90
 SIF_DONE_VLD
 SIF_InitTypeDef, 90
 SIF_EN
 SIF_InitTypeDef, 90
 SIF_IE_Enable
 lks32mc09x_sif.h, 452
 SIF_IF_CLEAR
 lks32mc09x_sif.h, 452
 SIF_Init
 lks32mc09x_sif.c, 676
 lks32mc09x_sif.h, 453
 SIF_InitTypeDef, 89
 SIF_DMA_EN, 90
 SIF_DONE, 90
 SIF_DONE_VLD, 90
 SIF_EN, 90
 SIF_IRQ_EN, 90
 SIF_IRQ_IF, 90
 SIF_MSB, 91
 SIF_RATIO, 91
 SIF_SYNC, 91
 SIF_SYNC_PULSE, 91
 SIF_SYNC_VLD, 91
 SIF_TDTH1, 91
 SIF_TOSC, 92
 SIF_TSTH1, 92
 SIF_IRQ_EN
 SIF_InitTypeDef, 90
 SIF_IRQ_IF
 SIF_InitTypeDef, 90
 SIF_Irq_IsEnable
 lks32mc09x_sif.c, 676
 lks32mc09x_sif.h, 453
 SIF_MSB

- SIF_InitTypeDef, 91
- SIF_RATIO
 - SIF_InitTypeDef, 91
- SIF_Senddata
 - lks32mc09x_sif.c, 677
 - lks32mc09x_sif.h, 453
- SIF_StructInit
 - lks32mc09x_sif.c, 677
 - lks32mc09x_sif.h, 454
- SIF_SYNC
 - SIF_InitTypeDef, 91
- SIF_SYNC_PULSE
 - SIF_InitTypeDef, 91
- SIF_SYNC_VLD
 - SIF_InitTypeDef, 91
- SIF_TDTH1
 - SIF_InitTypeDef, 91
- SIF_TH_set
 - lks32mc09x_sif.c, 677
 - lks32mc09x_sif.h, 454
- SIF_Timebase_set
 - lks32mc09x_sif.c, 678
 - lks32mc09x_sif.h, 454
- SIF_TOSC
 - SIF_InitTypeDef, 92
- SIF_TSTH1
 - SIF_InitTypeDef, 92
- SIF_TypeDef, 92
 - CFG, 93
 - IRQ, 93
 - TDTH1, 93
 - TOSC, 93
 - TSTH1, 93
 - WDATA, 93
- SINC
 - DMA_InitTypeDef, 37
- SIZE
 - FLASH_TypeDef, 50
 - SPI_TypeDef, 98
- SLV_MODE
 - I2C_InitTypeDef, 63
- softIE
 - HALL_InitTypeDef, 59
- source/lks32mc09x_adc.c, 533, 540
- source/lks32mc09x_clu.c, 543, 546
- source/lks32mc09x_cmp.c, 547, 555
- source/lks32mc09x_crc.c, 559, 564
- source/lks32mc09x_dac.c, 567, 578
- source/lks32mc09x_dma.c, 581, 590
- source/lks32mc09x_dsp.c, 593, 598
- source/lks32mc09x_eeprom.c, 599, 605
- source/lks32mc09x_exti.c, 607, 610
- source/lks32mc09x_flash.c, 612, 615
- source/lks32mc09x_gpio.c, 616, 625
- source/lks32mc09x_hall.c, 628, 632
- source/lks32mc09x_i2c.c, 633, 643
- source/lks32mc09x_iwdg.c, 646, 648
- source/lks32mc09x_mcpwm.c, 649, 659
- source/lks32mc09x_opa.c, 663, 668
- source/lks32mc09x_qep.c, 669, 673
- source/lks32mc09x_sif.c, 674, 678
- source/lks32mc09x_spi.c, 679, 682
- source/lks32mc09x_sys.c, 683, 692
- source/lks32mc09x_timer.c, 695, 701
- source/lks32mc09x_tmp.c, 704, 707
- source/lks32mc09x_uart.c, 708, 716
- source/lks32mc09x_wake.c, 718, 720
- SPI
 - lks32mc09x_reg.h, 443
- SPI0
 - lks32mc09x_reg.h, 443
- SPI_ClearIRQFlag
 - lks32mc09x_spi.c, 680
 - lks32mc09x_spi.h, 460
- SPI_DMA_DISABLE
 - lks32mc09x_spi.h, 457
- SPI_DMA_ENABLE
 - lks32mc09x_spi.h, 457
- SPI_Duplex
 - lks32mc09x_spi.h, 459
- SPI_FIRSTSEND_LSB
 - lks32mc09x_spi.h, 458
- SPI_FIRSTSEND_MSB
 - lks32mc09x_spi.h, 458
- SPI_Full
 - lks32mc09x_spi.h, 459
- SPI_GetIRQFlag
 - lks32mc09x_spi.c, 680
 - lks32mc09x_spi.h, 460
- SPI_Half_Rx
 - lks32mc09x_spi.h, 459
- SPI_Half_Tx
 - lks32mc09x_spi.h, 459
- SPI_IF_DataOver
 - lks32mc09x_spi.h, 458
- SPI_IF_SSErr
 - lks32mc09x_spi.h, 458
- SPI_IF_TranDone
 - lks32mc09x_spi.h, 458
- SPI_Init
 - lks32mc09x_spi.c, 680
 - lks32mc09x_spi.h, 461
- SPI_InitTypeDef, 94
 - BaudRate, 95
 - ByteLength, 95
 - CPHA, 95
 - CPOL, 95
 - CS, 95
 - Duplex, 95
 - EN, 95
 - ENDIAN, 96
 - IRQEna, 96
 - Mode, 96
 - TRANS_MODE, 96
 - Trig, 96
- SPI_IRQEna_DataOver

lks32mc09x_spi.h, 458
 SPI_IRQEna_Enable
 lks32mc09x_spi.h, 459
 SPI_IRQEna_SSErr
 lks32mc09x_spi.h, 459
 SPI_IRQEna_TranDone
 lks32mc09x_spi.h, 459
 SPI_Master
 lks32mc09x_spi.h, 460
 SPI_Mode
 lks32mc09x_spi.h, 460
 SPI_ReadData
 lks32mc09x_spi.c, 681
 lks32mc09x_spi.h, 461
 SPI_SendData
 lks32mc09x_spi.c, 681
 lks32mc09x_spi.h, 461
 SPI_Slave
 lks32mc09x_spi.h, 460
 SPI_StructInit
 lks32mc09x_spi.c, 682
 lks32mc09x_spi.h, 462
 SPI_TypeDef, 97
 CFG, 97
 DIV, 97
 IE, 97
 RX_DATA, 98
 SIZE, 98
 TX_DATA, 98
 SQRT
 DSP_TypeDef, 42
 SRC0
 TIMER_TimerInitTypeDef, 115
 SRC1
 TIMER_TimerInitTypeDef, 115
 state
 iictmpval, 66
 STOP_IE
 I2C_InitTypeDef, 63
 StopBits
 UART_InitTypeDef, 122
 STT
 UART_TypeDef, 125
 SW_FAIL
 MCPWM_TypeDef, 81
 SWAP
 EXTI_TypeDef, 46
 SWT
 ADC_TypeDef, 16
 SYS
 lks32mc09x_reg.h, 443
 SYS0
 lks32mc09x_reg.h, 443
 SYS_I2CClkDiv
 lks32mc09x_sys.c, 684
 lks32mc09x_sys.h, 470
 SYS_MCLK_RC_12M
 lks32mc09x_sys.h, 465
 SYS_MCLK_RC_24M
 lks32mc09x_sys.h, 465
 SYS_MCLK_RC_48M
 lks32mc09x_sys.h, 465
 SYS_MCLK_RC_8M
 lks32mc09x_sys.h, 465
 SYS_MCLK_RC_96M
 lks32mc09x_sys.h, 465
 SYS_MCLK_XTAL_12M
 lks32mc09x_sys.h, 465
 SYS_MCLK_XTAL_24M
 lks32mc09x_sys.h, 466
 SYS_MCLK_XTAL_48M
 lks32mc09x_sys.h, 466
 SYS_MCLK_XTAL_96M
 lks32mc09x_sys.h, 466
 SYS_MclkChoice
 lks32mc09x_sys.c, 684
 lks32mc09x_sys.h, 470
 SYS_MODULE_ADC0
 lks32mc09x_sys.h, 466
 SYS_MODULE_ADC1
 lks32mc09x_sys.h, 466
 SYS_MODULE_CL0
 lks32mc09x_sys.h, 466
 SYS_MODULE_CMP
 lks32mc09x_sys.h, 466
 SYS_MODULE_CRC0
 lks32mc09x_sys.h, 466
 SYS_MODULE_DMA0
 lks32mc09x_sys.h, 467
 SYS_MODULE_DSP0
 lks32mc09x_sys.h, 467
 SYS_MODULE_EEPROM
 lks32mc09x_sys.h, 467
 SYS_MODULE_GPIO
 lks32mc09x_sys.h, 467
 SYS_MODULE_HALL0
 lks32mc09x_sys.h, 467
 SYS_MODULE_I2C0
 lks32mc09x_sys.h, 467
 SYS_MODULE_MCPWM0
 lks32mc09x_sys.h, 467
 SYS_MODULE_QEP0
 lks32mc09x_sys.h, 467
 SYS_MODULE_SIF0
 lks32mc09x_sys.h, 468
 SYS_MODULE_SPI0
 lks32mc09x_sys.h, 468
 SYS_MODULE_TIMER0
 lks32mc09x_sys.h, 468
 SYS_MODULE_TIMER1
 lks32mc09x_sys.h, 468
 SYS_MODULE_TIMER2
 lks32mc09x_sys.h, 468
 SYS_MODULE_UART0
 lks32mc09x_sys.h, 468
 SYS_MODULE_UART1

- lks32mc09x_sys.h, 468
- SYS_MODULE_UART2
 - lks32mc09x_sys.h, 468
- SYS_ModuleClockCmd
 - lks32mc09x_sys.c, 685
 - lks32mc09x_sys.h, 471
- SYS_PVDSEL_2_8V
 - lks32mc09x_sys.h, 469
- SYS_PVDSEL_3_49V
 - lks32mc09x_sys.h, 469
- SYS_PVDSEL_3_5V
 - lks32mc09x_sys.h, 469
- SYS_PVDSEL_3_74V
 - lks32mc09x_sys.h, 469
- SYS_PVDSEL_3_75V
 - lks32mc09x_sys.h, 469
- SYS_PVDSEL_4_0V
 - lks32mc09x_sys.h, 469
- SYS_PVDSEL_DAC
 - lks32mc09x_sys.h, 469
- SYS_PVDSEL_OFF
 - lks32mc09x_sys.h, 469
- SYS_ReadMcuClk
 - lks32mc09x_sys.c, 686
 - lks32mc09x_sys.h, 472
- SYS_ResetIO
 - lks32mc09x_sys.c, 687
 - lks32mc09x_sys.h, 473
- SYS_ResetPeripheral
 - lks32mc09x_sys.c, 687
 - lks32mc09x_sys.h, 473
- SYS_ResetPeripheralInternal
 - lks32mc09x_sys.c, 688
 - lks32mc09x_sys.h, 474
- SYS_SetPowerFailureDetection
 - lks32mc09x_sys.c, 688
 - lks32mc09x_sys.h, 474
- SYS_SoftReset_all
 - lks32mc09x_sys.c, 689
 - lks32mc09x_sys.h, 475
- SYS_SoftResetModule
 - lks32mc09x_sys.c, 689
 - lks32mc09x_sys.h, 475
- SYS_SPIClkDiv
 - lks32mc09x_sys.c, 690
 - lks32mc09x_sys.h, 476
- SYS_TypeDef, 98
 - AFE_ADC, 101
 - AFE_DAC0, 101
 - AFE_DAC0_AMC, 101
 - AFE_DAC0_DC, 101
 - AFE_DAC1, 101
 - AFE_DAC1_AMC, 101
 - AFE_DAC1_DC, 101
 - AFE_DAC_CTRL, 102
 - AFE_DBG, 102
 - AFE_INFO, 102
 - AFE_REG0, 102
 - AFE_REG1, 102
 - AFE_REG2, 102
 - AFE_REG3, 103
 - AFE_REG4, 103
 - AFE_REG5, 103
 - AFE_REG7, 103
 - CLK_CFG, 103
 - CLK_DIV0, 104
 - CLK_DIV1, 104
 - CLK_DIV2, 104
 - CLK_FEN, 104
 - DBG_CFG, 104
 - FLSE, 104
 - FLSP, 105
 - IF, 105
 - IO_CFG, 105
 - NMI_TH, 105
 - OPA_SEL, 105
 - PROTECT, 105
 - RAM_LOCK, 106
 - RESERVED, 106
 - RESERVED1, 106
 - RESERVED2, 106
 - RESERVED3, 106
 - RESERVED41, 106
 - RESERVED5, 107
 - RESERVED6, 107
 - RESERVED7, 107
 - SFT_RST, 107
- SYS_UartClkDiv
 - lks32mc09x_sys.c, 690
 - lks32mc09x_sys.h, 476
- SYS_WaitPowerReady
 - lks32mc09x_sys.c, 691
 - lks32mc09x_sys.h, 478
- SYS_WakeInterval_025s
 - lks32mc09x_wake.h, 527
- SYS_WakeInterval_05s
 - lks32mc09x_wake.h, 527
- SYS_WakeInterval_16s
 - lks32mc09x_wake.h, 527
- SYS_WakeInterval_1s
 - lks32mc09x_wake.h, 527
- SYS_WakeInterval_2s
 - lks32mc09x_wake.h, 527
- SYS_WakeInterval_32s
 - lks32mc09x_wake.h, 527
- SYS_WakeInterval_4s
 - lks32mc09x_wake.h, 527
- SYS_WakeInterval_8s
 - lks32mc09x_wake.h, 528
- TC_IE
 - I2C_InitTypeDef, 63
- TCLK
 - CMP_TypeDef, 30
 - MCPWM_TypeDef, 81
- TCNT
 - ADC_InitTypeDef, 8

TDTH1
 SIF_TypeDef, 93
 TH
 HALL_TypeDef, 60
 MCPWM_InitTypeDef, 72
 QEP_InitTypeDef, 87
 QEP_TypeDef, 88
 TIMER_TimerInitTypeDef, 115
 TIMER_TypeDef, 119
 TH0
 MCPWM_TypeDef, 82
 TH00
 MCPWM_TypeDef, 82
 TH01
 MCPWM_TypeDef, 82
 TH10
 MCPWM_TypeDef, 82
 TH11
 MCPWM_TypeDef, 82
 TH20
 MCPWM_TypeDef, 82
 TH21
 MCPWM_TypeDef, 83
 TH30
 MCPWM_TypeDef, 83
 TH31
 MCPWM_TypeDef, 83
 TIMERO
 lks32mc09x_reg.h, 443
 TIMER1
 lks32mc09x_reg.h, 443
 TIMER16_0
 lks32mc09x_reg.h, 443
 TIMER16_1
 lks32mc09x_reg.h, 443
 TIMER2
 lks32mc09x_reg.h, 444
 TIMER32_0
 lks32mc09x_reg.h, 444
 TIMER_CH.MODE_CAP
 lks32mc09x_timer.h, 482
 TIMER_CH.MODE_Capture
 lks32mc09x_timer.h, 482
 TIMER_CH.MODE_CMP
 lks32mc09x_timer.h, 482
 TIMER_CH.MODE_Comparison
 lks32mc09x_timer.h, 483
 TIMER_ClearIRQFlag
 lks32mc09x_timer.c, 696
 lks32mc09x_timer.h, 484
 TIMER_CMP0
 lks32mc09x_timer.c, 696
 lks32mc09x_timer.h, 484
 TIMER_CMP1
 lks32mc09x_timer.c, 696
 lks32mc09x_timer.h, 484
 TIMER_Disable
 lks32mc09x_timer.c, 697
 lks32mc09x_timer.h, 485
 TIMER_Enable
 lks32mc09x_timer.c, 697
 lks32mc09x_timer.h, 485
 TIMER_GetFlag
 lks32mc09x_timer.c, 698
 lks32mc09x_timer.h, 486
 TIMER_GetIRQFlag
 lks32mc09x_timer.c, 698
 lks32mc09x_timer.h, 486
 TIMER_IF_CH0
 lks32mc09x_timer.h, 483
 TIMER_IF_CH1
 lks32mc09x_timer.h, 483
 TIMER_IF_FAIL
 lks32mc09x_timer.h, 483
 TIMER_IF_ZC
 lks32mc09x_timer.h, 483
 TIMER_Init
 lks32mc09x_timer.c, 699
 lks32mc09x_timer.h, 487
 TIMER_SetOutputState
 lks32mc09x_timer.c, 699
 lks32mc09x_timer.h, 487
 TIMER_StatisticsStart
 lks32mc09x_timer.h, 483
 TIMER_StatisticsStop
 lks32mc09x_timer.h, 483
 TIMER_StructInit
 lks32mc09x_timer.c, 700
 lks32mc09x_timer.h, 488
 TIMER_Timer_GetCMPT0
 lks32mc09x_timer.c, 700
 lks32mc09x_timer.h, 488
 TIMER_Timer_GetCMPT1
 lks32mc09x_timer.c, 700
 lks32mc09x_timer.h, 488
 TIMER_Timer_GetCount
 lks32mc09x_timer.c, 701
 lks32mc09x_timer.h, 489
 TIMER_TimerCmd
 lks32mc09x_timer.c, 701
 lks32mc09x_timer.h, 489
 TIMER_TimerInitTypeDef, 107
 CAP0_CLR_EN, 109
 CAP1_CLR_EN, 109
 CENTER, 110
 CH0_DEFAULT, 110
 CH0_FE_CAP_EN, 110
 CH0_RE_CAP_EN, 110
 CH0_WorkMode, 110
 CH0Output, 110
 CH1_DEFAULT, 111
 CH1_FE_CAP_EN, 111
 CH1_RE_CAP_EN, 111
 CH1_WorkMode, 111
 CH1Output, 111
 CHN0, 111

CHN1, [112](#)
 ClockDiv, [112](#)
 CMP0_TRIGGER_MODE, [112](#)
 CMP1_TRIGGER_MODE, [112](#)
 CNT, [112](#)
 DIR, [112](#)
 EN, [113](#)
 ETON, [113](#)
 EVT, [113](#)
 FAIL_EN, [113](#)
 FAIL_POL, [113](#)
 FAIL_SEL, [113](#)
 FLT, [114](#)
 GATE_EN, [114](#)
 HALT_PRT, [114](#)
 IE, [114](#)
 MOE, [114](#)
 ONE_TRIG, [114](#)
 RL_EN, [115](#)
 SHADOW, [115](#)
 SRC0, [115](#)
 SRC1, [115](#)
 TH, [115](#)
 XCLK_EN, [115](#)
 TIMER_TypeDef, [116](#)
 CFG0, [117](#)
 CFG1, [117](#)
 CFG2, [117](#)
 CHN0, [117](#)
 CHN1, [117](#)
 CNT, [118](#)
 EVT, [118](#)
 FLT, [118](#)
 IE, [118](#)
 IF, [118](#)
 IO, [118](#)
 RESERVED, [119](#)
 TH, [119](#)
 tmp_b
 lks32mc09x_tmp.c, [707](#)
 TMP_Disable
 lks32mc09x_tmp.c, [705](#)
 lks32mc09x_tmp.h, [492](#)
 TMP_Enable
 lks32mc09x_tmp.c, [705](#)
 lks32mc09x_tmp.h, [492](#)
 TMP_GetCurrentTemperature
 lks32mc09x_tmp.c, [705](#)
 lks32mc09x_tmp.h, [492](#)
 TMP_Init
 lks32mc09x_tmp.c, [706](#)
 lks32mc09x_tmp.h, [493](#)
 tmp_k
 lks32mc09x_tmp.c, [707](#)
 TMP_Reset
 lks32mc09x_tmp.c, [706](#)
 lks32mc09x_tmp.h, [493](#)
 TMR0
 MCPWM_InitTypeDef, [72](#)
 MCPWM_TypeDef, [83](#)
 TMR1
 MCPWM_InitTypeDef, [72](#)
 MCPWM_TypeDef, [83](#)
 TMR2
 MCPWM_InitTypeDef, [73](#)
 MCPWM_TypeDef, [83](#)
 TMR3
 MCPWM_InitTypeDef, [73](#)
 MCPWM_TypeDef, [84](#)
 TMS
 DMA_InitTypeDef, [37](#)
 TOSC
 SIF_TypeDef, [93](#)
 TR
 MCPWM_InitTypeDef, [73](#)
 TRANS_MODE
 SPI_InitTypeDef, [96](#)
 TRIG
 ADC_InitTypeDef, [8](#)
 ADC_TypeDef, [16](#)
 Trig
 SPI_InitTypeDef, [96](#)
 TRIM_Read
 lks32mc09x_trim.h, [506](#)
 TROVS
 ADC_InitTypeDef, [8](#)
 TSTH1
 SIF_TypeDef, [93](#)
 TX_BUF_DMA_RE
 lks32mc09x_uart.h, [510](#)
 TX_DATA
 SPI_TypeDef, [98](#)
 TX_DMA_RE
 lks32mc09x_uart.h, [511](#)
 TXD_INV
 UART_InitTypeDef, [122](#)
 txdata
 iictmpval, [66](#)
 txdatap
 iictmpval, [66](#)
 UART
 lks32mc09x_reg.h, [444](#)
 UART0
 lks32mc09x_reg.h, [444](#)
 UART1
 lks32mc09x_reg.h, [444](#)
 UART2
 lks32mc09x_reg.h, [444](#)
 UART_ClearIRQFlag
 lks32mc09x_uart.c, [709](#)
 lks32mc09x_uart.h, [516](#)
 UART_FIRSTSEND_LSB
 lks32mc09x_uart.h, [511](#)
 UART_FIRSTSEND_MSB
 lks32mc09x_uart.h, [511](#)
 UART_GetIRQFlag

lks32mc09x_uart.c, 709
 lks32mc09x_uart.h, 516
 UART_IF_ABD
 lks32mc09x_uart.h, 511
 UART_IF_CheckError
 lks32mc09x_uart.h, 511
 UART_IF_IDLE
 lks32mc09x_uart.h, 511
 UART_IF_LBD
 lks32mc09x_uart.h, 512
 UART_IF_RcvOver
 lks32mc09x_uart.h, 512
 UART_IF_RX_OV
 lks32mc09x_uart.h, 512
 UART_IF_SendBufEmpty
 lks32mc09x_uart.h, 512
 UART_IF_SendOver
 lks32mc09x_uart.h, 512
 UART_IF_StopError
 lks32mc09x_uart.h, 512
 UART_IF_TX_OV
 lks32mc09x_uart.h, 513
 UART_Init
 lks32mc09x_uart.c, 709
 lks32mc09x_uart.h, 516
 UART_InitTypeDef, 119
 ABD_EN, 120
 BaudRate, 120
 Bit9Value, 120
 CK_EN, 120
 DMARE, 121
 DUPLEX, 121
 FirstSend, 121
 IRQEna, 121
 LBDL, 121
 LIN_EN, 121
 Match485Addr, 122
 MultiDropEna, 122
 ParityMode, 122
 RXD_INV, 122
 StopBits, 122
 TXD_INV, 122
 WordLength, 123
 UART_IRQEna_ABD
 lks32mc09x_uart.h, 513
 UART_IRQEna_CheckError
 lks32mc09x_uart.h, 513
 UART_IRQEna_IDLE
 lks32mc09x_uart.h, 513
 UART_IRQEna_LBD
 lks32mc09x_uart.h, 513
 UART_IRQEna_RcvOver
 lks32mc09x_uart.h, 513
 UART_IRQEna_RX_OV
 lks32mc09x_uart.h, 514
 UART_IRQEna_SendBuffEmpty
 lks32mc09x_uart.h, 514
 UART_IRQEna_SendOver
 lks32mc09x_uart.h, 514
 UART_IRQEna_StopError
 lks32mc09x_uart.h, 514
 UART_IRQEna_TX_OV
 lks32mc09x_uart.h, 514
 UART_Parity_EVEN
 lks32mc09x_uart.h, 515
 UART_Parity_NO
 lks32mc09x_uart.h, 515
 UART_Parity_ODD
 lks32mc09x_uart.h, 515
 UART_ParityMode
 lks32mc09x_uart.h, 515
 UART_ReadBaud
 lks32mc09x_uart.c, 710
 UART_ReadData
 lks32mc09x_uart.c, 710
 lks32mc09x_uart.h, 517
 UART_RecvnData
 lks32mc09x_uart.c, 711
 lks32mc09x_uart.h, 518
 UART_SendAddr_485
 lks32mc09x_uart.c, 711
 lks32mc09x_uart.h, 518
 UART_SendData
 lks32mc09x_uart.c, 712
 lks32mc09x_uart.h, 519
 UART_SendData_485
 lks32mc09x_uart.c, 713
 lks32mc09x_uart.h, 520
 UART_SendnData
 lks32mc09x_uart.c, 713
 lks32mc09x_uart.h, 521
 UART_SetBaud
 lks32mc09x_uart.c, 714
 lks32mc09x_uart.h, 521
 UART_SetClkDiv
 lks32mc09x_uart.c, 715
 lks32mc09x_uart.h, 522
 UART_STOPBITS_1b
 lks32mc09x_uart.h, 514
 UART_STOPBITS_2b
 lks32mc09x_uart.h, 515
 UART_StructInit
 lks32mc09x_uart.c, 715
 lks32mc09x_uart.h, 523
 UART_TypeDef, 123
 ADR, 124
 BUFF, 124
 CTRL, 124
 DIVH, 124
 DIVL, 124
 IE, 124
 IF, 125
 IOC, 125
 RE, 125
 STT, 125
 UART_WORDLENGTH_8b

lks32mc09x_uart.h, 515
 UART_WORDLENGTH_9b
 lks32mc09x_uart.h, 515
 UPDATE
 MCPWM_TypeDef, 84
 UTIMER0
 lks32mc09x_reg.h, 444
 UTIMER1
 lks32mc09x_reg.h, 444
 UTIMER16_0
 lks32mc09x_reg.h, 445
 UTIMER16_1
 lks32mc09x_reg.h, 445
 UTIMER2
 lks32mc09x_reg.h, 445
 UTIMER32_0
 lks32mc09x_reg.h, 445

 W_PWM_POL
 CMP_SingleTypeDef, 28
 WAKE_EVTCLRSleep
 lks32mc09x_wake.c, 718
 lks32mc09x_wake.h, 531
 WAKE_FallSleep
 lks32mc09x_wake.c, 718
 lks32mc09x_wake.h, 531
 WAKE_FLT_DI
 lks32mc09x_wake.h, 528
 WAKE_FLT_EN
 lks32mc09x_wake.h, 528
 WAKE_IO_CLUOUT0
 lks32mc09x_wake.h, 528
 WAKE_IO_CLUOUT1
 lks32mc09x_wake.h, 528
 WAKE_IO_CLUOUT2
 lks32mc09x_wake.h, 528
 WAKE_IO_CLUOUT3
 lks32mc09x_wake.h, 529
 WAKE_IO_P0_0
 lks32mc09x_wake.h, 529
 WAKE_IO_P0_12
 lks32mc09x_wake.h, 529
 WAKE_IO_P0_13
 lks32mc09x_wake.h, 529
 WAKE_IO_P0_14
 lks32mc09x_wake.h, 529
 WAKE_IO_P0_2
 lks32mc09x_wake.h, 529
 WAKE_IO_P2_0
 lks32mc09x_wake.h, 530
 WAKE_IO_P2_4
 lks32mc09x_wake.h, 530
 WAKE_IO_P2_7
 lks32mc09x_wake.h, 530
 WAKE_SetWakeInterval
 lks32mc09x_wake.c, 719
 lks32mc09x_wake.h, 531
 WAKE_SetWakeIO
 lks32mc09x_wake.c, 719
 lks32mc09x_wake.h, 531
 WAKE_TRIG_HIGH
 lks32mc09x_wake.h, 530
 WAKE_TRIG_LOW
 lks32mc09x_wake.h, 530
 WAKE_TRIG_x
 lks32mc09x_wake.h, 530
 WDATA
 FLASH_TypeDef, 51
 SIF_TypeDef, 93
 WDG_EN
 IWDG_InitTypeDef, 67
 WIDTH
 CRC_InitTypeDef, 31
 HALL_TypeDef, 60
 WordLength
 UART_InitTypeDef, 123
 WTH
 IWDG_InitTypeDef, 67
 IWDG_TypeDef, 69

 XCLK_EN
 TIMER_TimerInitTypeDef, 115
 XOROUT
 CRC_InitTypeDef, 31

 ZC
 QEP_InitTypeDef, 87