

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	9
2.1 ADC_InitTypeDef结构体 参考	9
2.1.1 详细描述	10
2.1.2 结构体成员变量说明	10
2.1.2.1 ADC_CLK_DIV	10
2.1.2.2 Align	11
2.1.2.3 Con_Sample	11
2.1.2.4 DAT0_TH	11
2.1.2.5 DAT0_TH.Type	11
2.1.2.6 FirSeg_Ch	11
2.1.2.7 FouSeg_Ch	12
2.1.2.8 Gain0	12
2.1.2.9 Gain1	12
2.1.2.10 IE	12
2.1.2.11 MCPWM_Trigger_En	13
2.1.2.12 RefVol	13
2.1.2.13 SecSeg_Ch	13
2.1.2.14 ThrSeg_Ch	13
2.1.2.15 Trigger_Cnt	14
2.1.2.16 Trigger_Mode	14
2.1.2.17 UTIMER_Trigger_En	14
2.2 ADC_TypeDef结构体 参考	15
2.2.1 详细描述	17
2.2.2 结构体成员变量说明	17
2.2.2.1 AMC_A0	17
2.2.2.2 AMC_A1	17
2.2.2.3 AMC_B0	17
2.2.2.4 AMC_B1	17
2.2.2.5 CFG	18
2.2.2.6 CHN0	18
2.2.2.7 CHN1	18
2.2.2.8 CHN2	18
2.2.2.9 CHN3	18
2.2.2.10 CHN4	18
2.2.2.11 CHN5	19
2.2.2.12 CHN6	19
2.2.2.13 CHN7	19
2.2.2.14 CHN8	19

2.2.2.15 CHN9	19
2.2.2.16 CHNT0	19
2.2.2.17 CHNT1	20
2.2.2.18 DAT0	20
2.2.2.19 DAT0_TH	20
2.2.2.20 DAT1	20
2.2.2.21 DAT10	20
2.2.2.22 DAT11	20
2.2.2.23 DAT12	21
2.2.2.24 DAT13	21
2.2.2.25 DAT14	21
2.2.2.26 DAT15	21
2.2.2.27 DAT16	21
2.2.2.28 DAT17	21
2.2.2.29 DAT18	22
2.2.2.30 DAT19	22
2.2.2.31 DAT2	22
2.2.2.32 DAT3	22
2.2.2.33 DAT4	22
2.2.2.34 DAT5	22
2.2.2.35 DAT6	23
2.2.2.36 DAT7	23
2.2.2.37 DAT8	23
2.2.2.38 DAT9	23
2.2.2.39 DC_A0	23
2.2.2.40 DC_A1	23
2.2.2.41 DC_B0	24
2.2.2.42 DC_B1	24
2.2.2.43 GAIN_0	24
2.2.2.44 GAIN_1	24
2.2.2.45 IE	24
2.2.2.46 IF	24
2.2.2.47 SD0	25
2.2.2.48 SD1	25
2.2.2.49 SWT	25
2.2.2.50 TRG	25
2.3 CAN_InitTypeDef结构体 参考	25
2.3.1 详细描述	26
2.3.2 结构体成员变量说明	26
2.3.2.1 CAN_Btr0	26
2.3.2.2 CAN_Btr1	26
2.3.2.3 CAN_DMAEn	26

2.3.2.4 CAN.ErrWarThre	26
2.3.2.5 CAN.WorkMode	26
2.3.2.6 IRQEna	27
2.4 CAN_TypeDef结构体 参考	27
2.4.1 详细描述	28
2.4.2 结构体成员变量说明	28
2.4.2.1 ALC	28
2.4.2.2 BTR0	28
2.4.2.3 BTR1	29
2.4.2.4 CMR	29
2.4.2.5 ECC	29
2.4.2.6 EIR	29
2.4.2.7 EWLR	29
2.4.2.8 IR	29
2.4.2.9 MOD	30
2.4.2.10 null0	30
2.4.2.11 null1	30
2.4.2.12 null2	30
2.4.2.13 null3	30
2.4.2.14 RBSA	30
2.4.2.15 RMC	30
2.4.2.16 RXERR	31
2.4.2.17 SR	31
2.4.2.18 TXERR	31
2.4.2.19 TXRX0	31
2.4.2.20 TXRX1	31
2.4.2.21 TXRX2	31
2.4.2.22 TXRX3	32
2.4.2.23 TXRX4	32
2.4.2.24 TXRX5	32
2.4.2.25 TXRX6	32
2.4.2.26 TXRX7	32
2.4.2.27 TXRX8	32
2.4.2.28 TXRX9	33
2.4.2.29 TXRXA	33
2.4.2.30 TXRXB	33
2.4.2.31 TXRXC	33
2.5 CMP_InitTypeDef结构体 参考	33
2.5.1 详细描述	35
2.5.2 结构体成员变量说明	35
2.5.2.1 CLK10_EN	35
2.5.2.2 CMP0.CHN0P_WIN_EN	36

2.5.2.3 CMP0.CHN1P_WIN_EN	36
2.5.2.4 CMP0.CHN2P_WIN_EN	36
2.5.2.5 CMP0.CHN3P_WIN_EN	36
2.5.2.6 CMP0.EN	36
2.5.2.7 CMP0.IE	36
2.5.2.8 CMP0.IN_EN	37
2.5.2.9 CMP0.IRQ_TRIG	37
2.5.2.10 CMP0.POL	37
2.5.2.11 CMP0.SELN	37
2.5.2.12 CMP0.SELP	38
2.5.2.13 CMP0.W_PWM.POL	38
2.5.2.14 CMP1.CHN0P_WIN_EN	38
2.5.2.15 CMP1.CHN1P_WIN_EN	38
2.5.2.16 CMP1.CHN2P_WIN_EN	39
2.5.2.17 CMP1.CHN3P_WIN_EN	39
2.5.2.18 CMP1.EN	39
2.5.2.19 CMP1.IE	39
2.5.2.20 CMP1.IN_EN	39
2.5.2.21 CMP1.IRQ_TRIG	39
2.5.2.22 CMP1.POL	40
2.5.2.23 CMP1.SELN	40
2.5.2.24 CMP1.SELP	40
2.5.2.25 CMP1.W_PWM.POL	40
2.5.2.26 CMP_FT	41
2.5.2.27 CMP_HYS	41
2.5.2.28 FIL_CLK10_DIV16	41
2.5.2.29 FIL_CLK10_DIV2	41
2.6 CMP_TypeDef结构体 参考	42
2.6.1 详细描述	42
2.6.2 结构体成员变量说明	42
2.6.2.1 BLCWIN	42
2.6.2.2 CFG	42
2.6.2.3 DATA	43
2.6.2.4 IE	43
2.6.2.5 IF	43
2.6.2.6 TCLK	43
2.7 DAC_InitTypeDef结构体 参考	43
2.7.1 详细描述	43
2.7.2 结构体成员变量说明	44
2.7.2.1 DAC_GAIN	44
2.7.2.2 DACOUT_EN	44
2.8 DMA_InitTypeDef结构体 参考	44

2.8.1 详细描述	45
2.8.2 结构体成员变量说明	45
2.8.2.1 DMA_CIRC	45
2.8.2.2 DMA_CMAR	45
2.8.2.3 DMA_CPAR	45
2.8.2.4 DMA_DIR	45
2.8.2.5 DMA_IRQ_EN	46
2.8.2.6 DMA_MBTW	46
2.8.2.7 DMA_MINC	46
2.8.2.8 DMA_PBTW	46
2.8.2.9 DMA_PINC	47
2.8.2.10 DMA_REQ_EN	47
2.8.2.11 DMA_ROUND	47
2.8.2.12 DMA_TIMES	47
2.9 DMA_RegTypeDef结构体 参考	48
2.9.1 详细描述	48
2.9.2 结构体成员变量说明	48
2.9.2.1 DMA_CCR	48
2.9.2.2 DMA_CMAR	48
2.9.2.3 DMA_CPAR	49
2.9.2.4 DMA_CTMS	49
2.10 EXTI_TypeDef结构体 参考	49
2.10.1 详细描述	49
2.10.2 结构体成员变量说明	49
2.10.2.1 CR0	49
2.10.2.2 CR1	50
2.10.2.3 EN	50
2.10.2.4 IF	50
2.10.2.5 POL	50
2.11 GPIO_InitTypeDef结构体 参考	50
2.11.1 详细描述	50
2.11.2 结构体成员变量说明	51
2.11.2.1 GPIO_Mode	51
2.11.2.2 GPIO_Pin	51
2.11.2.3 GPIO_PODEna	51
2.11.2.4 GPIO_PuPd	51
2.12 GPIO_TypeDef结构体 参考	51
2.12.1 详细描述	52
2.12.2 结构体成员变量说明	52
2.12.2.1 F3210	52
2.12.2.2 F7654	52
2.12.2.3 FBA98	52

2.12.2.4 FFEDC	52
2.12.2.5 LCKR	52
2.12.2.6 PDE	53
2.12.2.7 PDI	53
2.12.2.8 PDO	53
2.12.2.9 PIE	53
2.12.2.10 PODE	53
2.12.2.11 POE	53
2.12.2.12 PUE	54
2.13 HALL_InitTypeDef结构体 参考	54
2.13.1 详细描述	54
2.13.2 结构体成员变量说明	54
2.13.2.1 Capture_IRQ_Ena	54
2.13.2.2 ClockDivision	55
2.13.2.3 CountTH	55
2.13.2.4 Filter75_Ena	55
2.13.2.5 FilterLen	55
2.13.2.6 HALL_Ena	55
2.13.2.7 OverFlow_IRQ_Ena	55
2.13.2.8 softIE	56
2.14 HALL_TypeDef结构体 参考	56
2.14.1 详细描述	56
2.14.2 结构体成员变量说明	56
2.14.2.1 CFG	56
2.14.2.2 CNT	57
2.14.2.3 INFO	57
2.14.2.4 TH	57
2.14.2.5 WIDTH	57
2.15 I2C_InitTypeDef结构体 参考	57
2.15.1 详细描述	58
2.15.2 结构体成员变量说明	58
2.15.2.1 ADRCMP	58
2.15.2.2 BaudRate	58
2.15.2.3 BURST_ADDR_CMP	59
2.15.2.4 BURST_NACK	59
2.15.2.5 BUS_ERR_IE	59
2.15.2.6 DMA	59
2.15.2.7 IE	59
2.15.2.8 MST_MODE	59
2.15.2.9 SLV_MODE	60
2.15.2.10 STOP_IE	60
2.15.2.11 TC_IE	60

2.16 I2C_TypeDef结构体 参考	60
2.16.1 详细描述	60
2.16.2 结构体成员变量说明	61
2.16.2.1 ADDR	61
2.16.2.2 BCR	61
2.16.2.3 CFG	61
2.16.2.4 DATA	61
2.16.2.5 MSCR	61
2.16.2.6 SCR	61
2.17 IWDG_InitTypeDef结构体 参考	61
2.17.1 详细描述	62
2.17.2 结构体成员变量说明	62
2.17.2.1 RTH	62
2.17.2.2 WDG_EN	62
2.18 MCPWM_InitTypeDef结构体 参考	62
2.18.1 详细描述	67
2.18.2 结构体成员变量说明	67
2.18.2.1 CH0_NS	67
2.18.2.2 CH0_PS	67
2.18.2.3 CH0N_default_output	67
2.18.2.4 CH0N_Polarity_INV	68
2.18.2.5 CH0N_SCTRLN	68
2.18.2.6 CH0P_default_output	68
2.18.2.7 CH0P_Polarity_INV	68
2.18.2.8 CH0P_SCTRLP	68
2.18.2.9 CH1_NS	69
2.18.2.10 CH1_PS	69
2.18.2.11 CH1N_default_output	69
2.18.2.12 CH1N_Polarity_INV	69
2.18.2.13 CH1N_SCTRLN	69
2.18.2.14 CH1P_default_output	70
2.18.2.15 CH1P_Polarity_INV	70
2.18.2.16 CH1P_SCTRLP	70
2.18.2.17 CH2_NS	70
2.18.2.18 CH2_PS	70
2.18.2.19 CH2N_default_output	71
2.18.2.20 CH2N_Polarity_INV	71
2.18.2.21 CH2N_SCTRLN	71
2.18.2.22 CH2P_default_output	71
2.18.2.23 CH2P_Polarity_INV	71
2.18.2.24 CH2P_SCTRLP	72
2.18.2.25 CH3_NS	72

2.18.2.26 CH3_PS	72
2.18.2.27 CH3N_default_output	72
2.18.2.28 CH3N_Polarity_INV	72
2.18.2.29 CH3N_SCTRLN	73
2.18.2.30 CH3P_default_output	73
2.18.2.31 CH3P_Polarity_INV	73
2.18.2.32 CH3P_SCTRLP	73
2.18.2.33 CLK_DIV	73
2.18.2.34 CMP_BKIN_Filter	74
2.18.2.35 DeadTimeCH0N	74
2.18.2.36 DeadTimeCH0P	74
2.18.2.37 DeadTimeCH1N	74
2.18.2.38 DeadTimeCH1P	74
2.18.2.39 DeadTimeCH2N	75
2.18.2.40 DeadTimeCH2P	75
2.18.2.41 DeadTimeCH3N	75
2.18.2.42 DeadTimeCH3P	75
2.18.2.43 DebugMode_PWM_out	76
2.18.2.44 FAIL0_INPUT_EN	76
2.18.2.45 FAIL0_INT_EN	76
2.18.2.46 FAIL0_Polarity	76
2.18.2.47 FAIL0_Signal_Sel	76
2.18.2.48 FAIL1_INPUT_EN	77
2.18.2.49 FAIL1_INT_EN	77
2.18.2.50 FAIL1_Polarity	77
2.18.2.51 FAIL1_Signal_Sel	77
2.18.2.52 GPIO_BKIN_Filter	77
2.18.2.53 IO0_PPE	78
2.18.2.54 IO1_PPE	78
2.18.2.55 IO2_PPE	78
2.18.2.56 IO3_PPE	78
2.18.2.57 MCLK_EN	78
2.18.2.58 MCPWM_Auto_ERR_EN	78
2.18.2.59 MCPWM_Cnt_EN	79
2.18.2.60 MCPWM_PERIOD	79
2.18.2.61 MCPWM_REG_UPDATE_INT_EN	79
2.18.2.62 MCPWM_T0_UpdateEN	79
2.18.2.63 MCPWM_T1_UpdateEN	79
2.18.2.64 MCPWM_UpdateInterval	79
2.18.2.65 MCPWM_WorkModeCH0	80
2.18.2.66 MCPWM_WorkModeCH1	80
2.18.2.67 MCPWM_WorkModeCH2	80

2.18.2.68 MCPWM_WorkModeCH3	80
2.18.2.69 Switch_CH0N_CH0P	81
2.18.2.70 Switch_CH1N_CH1P	81
2.18.2.71 Switch_CH2N_CH2P	81
2.18.2.72 Switch_CH3N_CH3P	81
2.18.2.73 T0_Update_INT_EN	81
2.18.2.74 T1_Update_INT_EN	81
2.18.2.75 TH00_Match_INT_EN	82
2.18.2.76 TH01_Match_INT_EN	82
2.18.2.77 TH10_Match_INT_EN	82
2.18.2.78 TH11_Match_INT_EN	82
2.18.2.79 TH20_Match_INT_EN	82
2.18.2.80 TH21_Match_INT_EN	82
2.18.2.81 TH30_Match_INT_EN	83
2.18.2.82 TH31_Match_INT_EN	83
2.18.2.83 TMR0_Match_INT_EN	83
2.18.2.84 TMR1_Match_INT_EN	83
2.18.2.85 TMR2_Match_INT_EN	83
2.18.2.86 TMR3_Match_INT_EN	83
2.18.2.87 TriggerPoint0	84
2.18.2.88 TriggerPoint1	84
2.18.2.89 TriggerPoint2	84
2.18.2.90 TriggerPoint3	84
2.19 OPA_InitTypeDef结构体 参考	84
2.19.1 详细描述	85
2.19.2 结构体成员变量说明	85
2.19.2.1 OPA_CLEna	85
2.19.2.2 OPA_Gain	85
2.20 SPI_InitTypeDef结构体 参考	85
2.20.1 详细描述	86
2.20.2 结构体成员变量说明	86
2.20.2.1 BaudRate	86
2.20.2.2 ByteLength	87
2.20.2.3 CPHA	87
2.20.2.4 CPOL	87
2.20.2.5 DataOrder	87
2.20.2.6 DMA	87
2.20.2.7 Duplex	87
2.20.2.8 EN	88
2.20.2.9 IRQEna	88
2.20.2.10 Mode	88
2.20.2.11 SS	88

2.20.2.12 Trig	89
2.21 SPI_TypeDef结构体 参考	89
2.21.1 详细描述	89
2.21.2 结构体成员变量说明	89
2.21.2.1 CFG	89
2.21.2.2 DIV	90
2.21.2.3 IE	90
2.21.2.4 RX_DATA	90
2.21.2.5 SIZE	90
2.21.2.6 TX_DATA	90
2.22 stru_CordicComponents结构体 参考	90
2.22.1 详细描述	91
2.22.2 结构体成员变量说明	91
2.22.2.1 ArctanValue	91
2.22.2.2 ArctanX	91
2.22.2.3 ArctanY	91
2.22.2.4 CosValue	92
2.22.2.5 SinValue	92
2.22.2.6 Theta	92
2.23 stru_DivComponents结构体 参考	92
2.23.1 详细描述	93
2.23.2 结构体成员变量说明	93
2.23.2.1 Dividend	93
2.23.2.2 Divisor	93
2.23.2.3 Quotient	93
2.23.2.4 Remainder	93
2.24 Stru_TempertureCof_Def结构体 参考	93
2.24.1 详细描述	94
2.24.2 结构体成员变量说明	94
2.24.2.1 nCofA	94
2.24.2.2 nOffsetB	94
2.25 SYS_InitTypeDef结构体 参考	94
2.25.1 详细描述	95
2.25.2 结构体成员变量说明	95
2.25.2.1 Clk_DivSPI	95
2.25.2.2 Clk_DivUART	95
2.25.2.3 Clk_FEN	96
2.25.2.4 Clk_Sel	96
2.25.2.5 PLL_DivSel	96
2.25.2.6 PLL_ReDiv	96
2.25.2.7 PLL_SrcSel	96
2.25.2.8 PORFilter_Ena	97

2.25.2.9 WDT_Ena	97
2.26 TIM_ECDInitTypeDef结构体 参考	97
2.26.1 详细描述	97
2.26.2 结构体成员变量说明	98
2.26.2.1 ECD_ClockDiv	98
2.26.2.2 ECD_Filter0	98
2.26.2.3 ECD_Filter1	98
2.26.2.4 ECD_IRQEna	98
2.26.2.5 ECD_Mode	99
2.26.2.6 ECD_TH	99
2.27 TIM_ECDTypeDef结构体 参考	99
2.27.1 详细描述	100
2.27.2 结构体成员变量说明	100
2.27.2.1 CFG	100
2.27.2.2 CNT	100
2.27.2.3 TH	100
2.28 TIM_TimerInitTypeDef结构体 参考	100
2.28.1 详细描述	101
2.28.2 结构体成员变量说明	101
2.28.2.1 EN	101
2.28.2.2 Timer_CH0_CapMode	102
2.28.2.3 Timer_CH0_WorkMode	102
2.28.2.4 Timer_CH0Output	102
2.28.2.5 Timer_CH1_CapMode	103
2.28.2.6 Timer_CH1_WorkMode	103
2.28.2.7 Timer_CH1Output	103
2.28.2.8 Timer_ClockDiv	104
2.28.2.9 Timer_CMP0	104
2.28.2.10 Timer_CMP1	104
2.28.2.11 Timer_Filter0	104
2.28.2.12 Timer_Filter1	104
2.28.2.13 Timer_IRQEna	105
2.28.2.14 Timer_TH	105
2.29 TIM_TimerTypeDef结构体 参考	105
2.29.1 详细描述	105
2.29.2 结构体成员变量说明	106
2.29.2.1 CFG	106
2.29.2.2 CMPT0	106
2.29.2.3 CMPT1	106
2.29.2.4 CNT	106
2.29.2.5 EVT	106
2.29.2.6 TH	106

2.30 UART_InitTypeDef结构体 参考	107
2.30.1 详细描述	107
2.30.2 结构体成员变量说明	107
2.30.2.1 BaudRate	107
2.30.2.2 Bit9Value	108
2.30.2.3 FirstSend	108
2.30.2.4 IRQEna	108
2.30.2.5 Match485Addr	108
2.30.2.6 MultiDropEna	109
2.30.2.7 ParityMode	109
2.30.2.8 RXD_INV	109
2.30.2.9 StopBits	109
2.30.2.10 TXD_INV	109
2.30.2.11 WordLength	110
2.31 UART_TypeDef结构体 参考	110
2.31.1 详细描述	110
2.31.2 结构体成员变量说明	111
2.31.2.1 ADR	111
2.31.2.2 BUFF	111
2.31.2.3 CTRL	111
2.31.2.4 DIVH	111
2.31.2.5 DIVL	111
2.31.2.6 IE	111
2.31.2.7 IF	112
2.31.2.8 INV	112
2.31.2.9 STT	112
2.32 WAKE_InitTypeDef结构体 参考	112
2.32.1 详细描述	112
2.32.2 结构体成员变量说明	112
2.32.2.1 AFE_REG5_RECORD	112
2.32.2.2 AFE_REG6_RECORD	112
2.32.2.3 CLK_CFG_RECORD	112
3 文件说明	113
3.1 include/lks32mc08x_adc.h 文件参考	113
3.1.1 详细描述	117
3.1.2 宏定义说明	118
3.1.2.1 ADC0	118
3.1.2.2 ADC_10_TIMES_SAMPLE	118
3.1.2.3 ADC_11_TIMES_SAMPLE	118
3.1.2.4 ADC_12_TIMES_SAMPLE	118
3.1.2.5 ADC_13_TIMES_SAMPLE	118

3.1.2.6 ADC_14_TIMES_SAMPLE	119
3.1.2.7 ADC_15_TIMES_SAMPLE	119
3.1.2.8 ADC_16_TIMES_SAMPLE	119
3.1.2.9 ADC_17_TIMES_SAMPLE	119
3.1.2.10 ADC_18_TIMES_SAMPLE	119
3.1.2.11 ADC_19_TIMES_SAMPLE	119
3.1.2.12 ADC_1_TIMES_SAMPLE	120
3.1.2.13 ADC_1SEG_TRG	120
3.1.2.14 ADC_20_TIMES_SAMPLE	120
3.1.2.15 ADC_2_TIMES_SAMPLE	120
3.1.2.16 ADC_2SEG_TRG	120
3.1.2.17 ADC_3_TIMES_SAMPLE	120
3.1.2.18 ADC_4_TIMES_SAMPLE	121
3.1.2.19 ADC_4SEG_TRG	121
3.1.2.20 ADC_5_TIMES_SAMPLE	121
3.1.2.21 ADC_6_TIMES_SAMPLE	121
3.1.2.22 ADC_7_TIMES_SAMPLE	121
3.1.2.23 ADC_8_TIMES_SAMPLE	121
3.1.2.24 ADC_9_TIMES_SAMPLE	122
3.1.2.25 ADC_ALL_IRQ_IF	122
3.1.2.26 ADC_CHANNEL_0	122
3.1.2.27 ADC_CHANNEL_1	122
3.1.2.28 ADC_CHANNEL_10	122
3.1.2.29 ADC_CHANNEL_11	122
3.1.2.30 ADC_CHANNEL_12	123
3.1.2.31 ADC_CHANNEL_13	123
3.1.2.32 ADC_CHANNEL_14	123
3.1.2.33 ADC_CHANNEL_15	123
3.1.2.34 ADC_CHANNEL_16	123
3.1.2.35 ADC_CHANNEL_17	123
3.1.2.36 ADC_CHANNEL_18	124
3.1.2.37 ADC_CHANNEL_19	124
3.1.2.38 ADC_CHANNEL_2	124
3.1.2.39 ADC_CHANNEL_3	124
3.1.2.40 ADC_CHANNEL_4	124
3.1.2.41 ADC_CHANNEL_5	124
3.1.2.42 ADC_CHANNEL_6	125
3.1.2.43 ADC_CHANNEL_7	125
3.1.2.44 ADC_CHANNEL_8	125
3.1.2.45 ADC_CHANNEL_9	125
3.1.2.46 ADC_Clock_12M	125
3.1.2.47 ADC_Clock_24M	125

3.1.2.48 ADC_Clock_48M	125
3.1.2.49 ADC_DAT0_HTH	126
3.1.2.50 ADC_DAT0_LTH	126
3.1.2.51 ADC_DAT0_OV_IRQ_EN	126
3.1.2.52 ADC_DAT0_OV_IRQ_IF	126
3.1.2.53 ADC_EOS0_IRQ_EN	126
3.1.2.54 ADC_EOS0_IRQ_IF	126
3.1.2.55 ADC_EOS1_IRQ_EN	127
3.1.2.56 ADC_EOS1_IRQ_IF	127
3.1.2.57 ADC_EOS2_IRQ_EN	127
3.1.2.58 ADC_EOS2_IRQ_IF	127
3.1.2.59 ADC_EOS3_IRQ_EN	127
3.1.2.60 ADC_EOS3_IRQ_IF	127
3.1.2.61 ADC_H_CONFLICT_IRQ_EN	128
3.1.2.62 ADC_H_CONFLICT_IRQ_IF	128
3.1.2.63 ADC_HARDWARE_T0_TRG	128
3.1.2.64 ADC_HARDWARE_T1_TRG	128
3.1.2.65 ADC_HARDWARE_T2_TRG	128
3.1.2.66 ADC_HARDWARE_T3_TRG	128
3.1.2.67 ADC_LEFT_ALIGN	129
3.1.2.68 ADC_MCPWM_T0_TRG	129
3.1.2.69 ADC_MCPWM_T1_TRG	129
3.1.2.70 ADC_MCPWM_T2_TRG	129
3.1.2.71 ADC_MCPWM_T3_TRG	129
3.1.2.72 ADC_RIGHT_ALIGN	129
3.1.2.73 ADC_S_CONFLICT_IRQ_EN	130
3.1.2.74 ADC_S_CONFLICT_IRQ_IF	130
3.1.2.75 ADC_UTIMER_T0_TRG	130
3.1.2.76 ADC_UTIMER_T1_TRG	130
3.1.2.77 ADC_UTIMER_T2_TRG	130
3.1.2.78 ADC_UTIMER_T3_TRG	130
3.1.2.79 RefVol_1V2	131
3.1.2.80 RefVol_2V4	131
3.1.3 枚举类型说明	131
3.1.3.1 CHNx	131
3.1.3.2 DATx	132
3.1.4 函数说明	132
3.1.4.1 ADC_ChannelConfig()	132
3.1.4.2 ADC_ClearIRQFlag()	133
3.1.4.3 ADC_GetConversionValue()	135
3.1.4.4 ADC_GetIRQFlag()	135
3.1.4.5 ADC_Init()	137

3.1.4.6 ADC_SoftTrgEN()	138
3.1.4.7 ADC_StructInit()	139
3.2 lks32mc08x_adc.h	140
3.3 include/lks32mc08x_can.h 文件参考	143
3.3.1 详细描述	146
3.3.2 宏定义说明	146
3.3.2.1 CAN_ALLSR	146
3.3.2.2 CAN_DATAAVAIL	146
3.3.2.3 CAN_ERROV	146
3.3.2.4 CAN_IE_BUSERR	147
3.3.2.5 CAN_IE_LOSTARB	147
3.3.2.6 CAN_IE_PASSIVEERR	147
3.3.2.7 CAN_IE_RFIFONEMPTY	147
3.3.2.8 CAN_IE_RFIFOOV	147
3.3.2.9 CAN_IE_TXDONE	147
3.3.2.10 CAN_IE_WAKE	148
3.3.2.11 CAN_IE_WERR	148
3.3.2.12 CAN_IF_BUSERR	148
3.3.2.13 CAN_IF_LOSTARB	148
3.3.2.14 CAN_IF_PASSIVEERR	148
3.3.2.15 CAN_IF_RFIFONEMPTY	148
3.3.2.16 CAN_IF_RFIFOOV	149
3.3.2.17 CAN_IF_TXDONE	149
3.3.2.18 CAN_IF_WAKE	149
3.3.2.19 CAN_IF_WERR	149
3.3.2.20 CAN_MONITOR_MODE	149
3.3.2.21 CAN_NORMAL_MODE	149
3.3.2.22 CAN_ONBUS	150
3.3.2.23 CAN_RFIFOEMPTY	150
3.3.2.24 CAN_RXING	150
3.3.2.25 CAN_SELFTEST_MODE	150
3.3.2.26 CAN_TFIFOEMPTY	150
3.3.2.27 CAN_TRNASDONE	151
3.3.2.28 CAN_TXING	151
3.3.3 枚举类型说明	151
3.3.3.1 enumErrortime	151
3.3.3.2 enumErrortype	151
3.3.3.3 enumErrposition	152
3.3.3.4 enumLossposition	152
3.3.4 函数说明	153
3.3.4.1 CAN_GetIRQFlag()	153
3.3.4.2 CAN_Init()	154

3.3.4.3 CAN_Manual_Awake()	155
3.3.4.4 CAN_ReadState()	156
3.3.4.5 CAN_Receive_Msg()	158
3.3.4.6 CAN_Sleep()	158
3.3.4.7 CAN_StructInit()	159
3.3.4.8 EFF_ID_Filter()	160
3.3.4.9 ID1_Filter_Dual()	161
3.3.4.10 ID2_Filter_Dual()	162
3.3.4.11 My_CAN_Send_Msg()	164
3.3.4.12 SFF_ID_Filter()	165
3.4 lks32mc08x.can.h	166
3.5 include/lks32mc08x_cmp.h 文件参考	168
3.5.1 详细描述	171
3.5.2 宏定义说明	171
3.5.2.1 CMP0	171
3.5.2.2 CMP0_SEL_P_IP0	172
3.5.2.3 CMP0_SEL_P_IP1	172
3.5.2.4 CMP0_SEL_P_IP2	172
3.5.2.5 CMP0_SEL_P_IP3	172
3.5.2.6 CMP0_SEL_P_IP4	172
3.5.2.7 CMP0_SEL_P_OPA0_IP	172
3.5.2.8 CMP0_SEL_P_OPA0_OUT	173
3.5.2.9 CMP0_SEL_P_OPA1_OUT	173
3.5.2.10 CMP1	173
3.5.2.11 CMP1_SEL_P_AVSS	173
3.5.2.12 CMP1_SEL_P_IP0	173
3.5.2.13 CMP1_SEL_P_IP1	173
3.5.2.14 CMP1_SEL_P_IP2	174
3.5.2.15 CMP1_SEL_P_IP3	174
3.5.2.16 CMP1_SEL_P_OPA2_OUT	174
3.5.2.17 CMP1_SEL_P_OPA3_IP	174
3.5.2.18 CMP1_SEL_P_OPA3_OUT	174
3.5.2.19 CMP_HYS_0mV	174
3.5.2.20 CMP_HYS_20mV	175
3.5.2.21 CMP_SEL_N_DAC	175
3.5.2.22 CMP_SEL_N_HALL_MID	175
3.5.2.23 CMP_SEL_N_IN	175
3.5.2.24 CMP_SEL_N_REF	175
3.5.2.25 SEL_N_CMP_IN	175
3.5.2.26 SEL_N_DAC	176
3.5.2.27 SEL_N_HALL_MID	176
3.5.2.28 SEL_N_REF	176

3.5.2.29 SELP_CMP_IP0	176
3.5.2.30 SELP_CMP_IP1	176
3.5.2.31 SELP_CMP_IP2	176
3.5.2.32 SELP_CMP_IP3	177
3.5.2.33 SELP_CMP_IP4	177
3.5.2.34 SELP_CMP_OPA0IP	177
3.5.2.35 SELP_CMP_OPA0OUT	177
3.5.2.36 SELP_CMP_OPA1OUT	177
3.5.2.37 SELP_CMP_OPA2OUT	177
3.5.2.38 SELP_CMP_OPA3IP	178
3.5.2.39 SELP_CMP_OPA3OUT	178
3.5.3 函数说明	178
3.5.3.1 CMP_ClearIRQFlag()	178
3.5.3.2 CMP_GetIRQFlag()	179
3.5.3.3 CMP_Init()	180
3.5.3.4 CMP_StructInit()	181
3.6 lks32mc08x_cmp.h	182
3.7 include/lks32mc08x_dac.h 文件参考	184
3.7.1 详细描述	185
3.7.2 宏定义说明	185
3.7.2.1 DAC_RANGE_1V2	185
3.7.2.2 DAC_RANGE_3V0	185
3.7.2.3 DAC_RANGE_4V85	185
3.7.3 函数说明	186
3.7.3.1 DAC_Init()	186
3.7.3.2 DAC_Output()	187
3.7.3.3 DAC_OutputValue()	187
3.7.3.4 DAC_OutputVoltage()	188
3.7.3.5 DAC_StructInit()	189
3.8 lks32mc08x_dac.h	190
3.9 include/lks32mc08x_dma.h 文件参考	190
3.9.1 详细描述	192
3.9.2 宏定义说明	193
3.9.2.1 DMA_BYTE_TRANS	193
3.9.2.2 DMA_CH0_ADC0_REQ_EN	193
3.9.2.3 DMA_CH0{EIF	193
3.9.2.4 DMA_CH0_FIF	193
3.9.2.5 DMA_CH0_TIMER0_REQ_EN	193
3.9.2.6 DMA_CH0_TIMER1_REQ_EN	193
3.9.2.7 DMA_CH1{EIF	194
3.9.2.8 DMA_CH1_FIF	194
3.9.2.9 DMA_CH1_MCPWM_REQ_EN	194

3.9.2.10 DMA.CH1_SPI_RX_REQ_EN	194
3.9.2.11 DMA.CH1_TIMER2_REQ_EN	194
3.9.2.12 DMA.CH2{EIF	194
3.9.2.13 DMA.CH2_FIF	195
3.9.2.14 DMA.CH2_SPI_TX_REQ_EN	195
3.9.2.15 DMA.CH2_TIMER3_REQ_EN	195
3.9.2.16 DMA.CH2_UART0_REQ_EN	195
3.9.2.17 DMA.CH3_CAN_REQ_EN	195
3.9.2.18 DMA.CH3{EIF	195
3.9.2.19 DMA.CH3_FIF	196
3.9.2.20 DMA.CH3_I2C_REQ_EN	196
3.9.2.21 DMA.CH3_UART1_REQ_EN	196
3.9.2.22 DMA_HALFWORD_TRANS	196
3.9.2.23 DMA_TCIE	196
3.9.2.24 DMA_TEIE	196
3.9.2.25 DMA_WORD_TRANS	197
3.9.2.26 MEMORY2PERI	197
3.9.2.27 PERI2MEMORY	197
3.9.3 函数说明	197
3.9.3.1 DMA_CHx_EN()	197
3.9.3.2 DMA_ClearIRQFlag()	198
3.9.3.3 DMA_GetIRQFlag()	199
3.9.3.4 DMA_Init()	201
3.9.3.5 DMA_StructInit()	202
3.10 lks32mc08x_dma.h	203
3.11 include/lks32mc08x_dsp.h 文件参考	204
3.11.1 详细描述	206
3.11.2 宏定义说明	206
3.11.2.1 DSP_CALC_ARCTAN	206
3.11.2.2 DSP_CALC_COS	206
3.11.2.3 DSP_CALC_DIV	207
3.11.2.4 DSP_CALC_MOD	207
3.11.2.5 DSP_CALC_RMS	207
3.11.2.6 DSP_CALC_SIN	208
3.11.2.7 DSP_CALC_SQRT	208
3.11.3 枚举类型说明	208
3.11.3.1 Arc.or.SinCFuction	208
3.11.4 函数说明	208
3.11.4.1 DSP_arctan()	208
3.11.4.2 DSP_CalcCordic()	209
3.11.4.3 DSP_CalcDivision()	210
3.11.4.4 DSP_CalcSprt()	211

3.11.4.5 DSP_Cmd()	212
3.11.4.6 DSP_cos()	213
3.11.4.7 DSP_div()	214
3.11.4.8 DSP_GetSqrt()	214
3.11.4.9 DSP_Init()	215
3.11.4.10 DSP_mod()	215
3.11.4.11 DSP_rms()	215
3.11.4.12 DSP_sin()	216
3.11.4.13 DSP_sqrt()	216
3.12 lks32mc08x_dsp.h	216
3.13 include/lks32mc08x_exti.h 文件参考	217
3.13.1 详细描述	219
3.13.2 宏定义说明	219
3.13.2.1 EXTI	219
3.13.3 枚举类型说明	219
3.13.3.1 EXTIMode_TypeDef	219
3.13.4 函数说明	220
3.13.4.1 EXTI_ClearIRQFlag()	220
3.13.4.2 EXTI_GetIRQFlag()	221
3.13.4.3 EXTI_Trigger_Config()	222
3.14 lks32mc08x_exti.h	223
3.15 include/lks32mc08x_flash.h 文件参考	223
3.15.1 详细描述	224
3.15.2 宏定义说明	225
3.15.2.1 Flash_MAIN	225
3.15.2.2 Flash_NVR	225
3.15.3 函数说明	225
3.15.3.1 EraseSector()	225
3.15.3.2 ProgramPage()	226
3.15.3.3 Read_More_Flash()	228
3.15.4 变量说明	230
3.15.4.1 erase_flag	230
3.15.4.2 progm_flag	230
3.16 lks32mc08x_flash.h	230
3.17 include/lks32mc08x_gpio.h 文件参考	231
3.17.1 详细描述	234
3.17.2 宏定义说明	234
3.17.2.1 AF0_GPIO	234
3.17.2.2 AF10_CAN	234
3.17.2.3 AF11_SIF	234
3.17.2.4 AF1_SYSAF	235
3.17.2.5 AF2_HALL	235

3.17.2.6 AF3_MCPWM	235
3.17.2.7 AF4_UART	235
3.17.2.8 AF5_SPI	235
3.17.2.9 AF6_I2C	235
3.17.2.10 AF7_TIMER01	236
3.17.2.11 AF8_TIMER23	236
3.17.2.12 AF9_DEBUG	236
3.17.2.13 GPIO0	236
3.17.2.14 GPIO1	236
3.17.2.15 GPIO2	236
3.17.2.16 GPIO3	237
3.17.2.17 GPIO_AF_CAN	237
3.17.2.18 GPIO_AF_DEBUG	237
3.17.2.19 GPIO_AF_GPIO	237
3.17.2.20 GPIO_AF_HALL	237
3.17.2.21 GPIO_AF_I2C	237
3.17.2.22 GPIO_AF_MCPWM	237
3.17.2.23 GPIO_AF_SIF	238
3.17.2.24 GPIO_AF_SPI	238
3.17.2.25 GPIO_AF_SYSAF	238
3.17.2.26 GPIO_AF_TIMER0	238
3.17.2.27 GPIO_AF_TIMER1	238
3.17.2.28 GPIO_AF_TIMER2	238
3.17.2.29 GPIO_AF_TIMER3	238
3.17.2.30 GPIO_AF_UART	238
3.17.2.31 GPIO_Pin_0	239
3.17.2.32 GPIO_Pin_1	239
3.17.2.33 GPIO_Pin_10	239
3.17.2.34 GPIO_Pin_11	239
3.17.2.35 GPIO_Pin_12	239
3.17.2.36 GPIO_Pin_13	239
3.17.2.37 GPIO_Pin_14	240
3.17.2.38 GPIO_Pin_15	240
3.17.2.39 GPIO_Pin_2	240
3.17.2.40 GPIO_Pin_3	240
3.17.2.41 GPIO_Pin_4	240
3.17.2.42 GPIO_Pin_5	240
3.17.2.43 GPIO_Pin_6	241
3.17.2.44 GPIO_Pin_7	241
3.17.2.45 GPIO_Pin_8	241
3.17.2.46 GPIO_Pin_9	241
3.17.2.47 GPIO_Pin_ALL	241

3.17.2.48 GPIO_Pin_NONE	241
3.17.2.49 GPIO_PinSource_0	242
3.17.2.50 GPIO_PinSource_1	242
3.17.2.51 GPIO_PinSource_10	242
3.17.2.52 GPIO_PinSource_11	242
3.17.2.53 GPIO_PinSource_12	242
3.17.2.54 GPIO_PinSource_13	242
3.17.2.55 GPIO_PinSource_14	243
3.17.2.56 GPIO_PinSource_15	243
3.17.2.57 GPIO_PinSource_2	243
3.17.2.58 GPIO_PinSource_3	243
3.17.2.59 GPIO_PinSource_4	243
3.17.2.60 GPIO_PinSource_5	243
3.17.2.61 GPIO_PinSource_6	244
3.17.2.62 GPIO_PinSource_7	244
3.17.2.63 GPIO_PinSource_8	244
3.17.2.64 GPIO_PinSource_9	244
3.17.3 枚举类型说明	244
3.17.3.1 BitAction	244
3.17.3.2 GPIOMode_TypeDef	245
3.17.3.3 GPIOPuPd_TypeDef	245
3.17.4 函数说明	245
3.17.4.1 GPIO_Config()	245
3.17.4.2 GPIO_Init()	246
3.17.4.3 GPIO_LockRegister()	247
3.17.4.4 GPIO_PinAFConfig()	248
3.17.4.5 GPIO_ReadInputData()	250
3.17.4.6 GPIO_ReadInputDataBit()	251
3.17.4.7 GPIO_ReadOutputData()	252
3.17.4.8 GPIO_ReadOutputDataBit()	253
3.17.4.9 GPIO_ReadPinBit()	254
3.17.4.10 GPIO_ResetBits()	254
3.17.4.11 GPIO_SetBits()	255
3.17.4.12 GPIO_StructInit()	256
3.17.4.13 GPIO_UnlockRegister()	258
3.17.4.14 GPIO_Write()	258
3.17.4.15 GPIO_WriteBit()	259
3.17.4.16 GPIO_WritePinBit()	261
3.18 lks32mc08x_gpio.h	261
3.19 include/lks32mc08x_hall.h 文件参考	263
3.19.1 详细描述	264
3.19.2 宏定义说明	265

3.19.2.1 HALL_CAPTURE_EVENT	265
3.19.2.2 HALL_CLK_DIV1	265
3.19.2.3 HALL_CLK_DIV2	265
3.19.2.4 HALL_CLK_DIV4	265
3.19.2.5 HALL_CLK_DIV8	265
3.19.2.6 HALL_OVERFLOW_EVENT	266
3.19.3 函数说明	266
3.19.3.1 HALL_ClearIRQFlag()	266
3.19.3.2 HALL_GetCaptureValue()	267
3.19.3.3 HALL_GetCount()	268
3.19.3.4 HALL_GetFilterValue()	269
3.19.3.5 HALL_GetIRQFlag()	270
3.19.3.6 HALL_Init()	271
3.19.3.7 HALL_StructInit()	272
3.19.3.8 HALL_WIDCount()	273
3.20 lks32mc08x_hall.h	274
3.21 include/lks32mc08x_i2c.h 文件参考	274
3.21.1 详细描述	275
3.21.2 函数说明	276
3.21.2.1 i2c_dma_init()	276
3.21.2.2 i2c_dma_rx()	277
3.21.2.3 i2c_dma_state_over()	279
3.21.2.4 i2c_dma_tx()	279
3.21.2.5 I2C_Init()	281
3.21.2.6 I2C_ReadData()	282
3.21.2.7 I2C_SendData()	283
3.21.2.8 I2C_StructInit()	284
3.22 lks32mc08x_i2c.h	285
3.23 include/lks32mc08x_iwdg.h 文件参考	286
3.23.1 详细描述	287
3.23.2 宏定义说明	287
3.23.2.1 WDT_RTH_2S	287
3.23.2.2 WDT_RTH_4S	287
3.23.2.3 WDT_RTH_64S	288
3.23.2.4 WDT_RTH_8S	288
3.23.3 函数说明	288
3.23.3.1 IWDG_DISABLE()	288
3.23.3.2 IWDG_ENABLE()	289
3.23.3.3 IWDG_Feed()	290
3.23.3.4 IWDG_Init()	291
3.23.3.5 IWDG_StructInit()	292
3.24 lks32mc08x_iwdg.h	293

3.25 include/lks32mc08x.lib.h 文件参考	293
3.25.1 详细描述	294
3.26 lks32mc08x.lib.h	294
3.27 include/lks32mc08x_mcpwm.h 文件参考	295
3.27.1 详细描述	299
3.27.2 宏定义说明	299
3.27.2.1 CENTRAL_PWM_MODE	299
3.27.2.2 EDGE_PWM_MODE	300
3.27.2.3 FAIL_SEL_CMP	300
3.27.2.4 FAIL_SEL_IO	300
3.27.2.5 HIGH_LEVEL	300
3.27.2.6 HIGH_LEVEL_ACTIVE	300
3.27.2.7 LOW_LEVEL	300
3.27.2.8 LOW_LEVEL_ACTIVE	301
3.27.2.9 MCPWM_CH0_SHORT_IRQ_EN	301
3.27.2.10 MCPWM_CH0_SHORT_IRQ_IF	301
3.27.2.11 MCPWM_CH1_SHORT_IRQ_EN	301
3.27.2.12 MCPWM_CH1_SHORT_IRQ_IF	301
3.27.2.13 MCPWM_CH2_SHORT_IRQ_EN	301
3.27.2.14 MCPWM_CH2_SHORT_IRQ_IF	302
3.27.2.15 MCPWM_CH3_SHORT_IRQ_EN	302
3.27.2.16 MCPWM_CH3_SHORT_IRQ_IF	302
3.27.2.17 MCPWM{EIF_CH0_SHORT	302
3.27.2.18 MCPWM{EIF_CH1_SHORT	302
3.27.2.19 MCPWM{EIF_CH2_SHORT	302
3.27.2.20 MCPWM{EIF_CH3_SHORT	303
3.27.2.21 MCPWM{EIF_FAIL0	303
3.27.2.22 MCPWM{EIF_FAIL1	303
3.27.2.23 MCPWM_FAIL0_IRQ_EN	303
3.27.2.24 MCPWM_FAIL0_IRQ_IF	303
3.27.2.25 MCPWM_FAIL1_IRQ_EN	303
3.27.2.26 MCPWM_FAIL1_IRQ_IF	304
3.27.2.27 MCPWM_IF_SHADE	304
3.27.2.28 MCPWM_IF_T0	304
3.27.2.29 MCPWM_IF_T1	304
3.27.2.30 MCPWM_IF_TH00	304
3.27.2.31 MCPWM_IF_TH01	304
3.27.2.32 MCPWM_IF_TH10	305
3.27.2.33 MCPWM_IF_TH11	305
3.27.2.34 MCPWM_IF_TH20	305
3.27.2.35 MCPWM_IF_TH21	305
3.27.2.36 MCPWM_IF_TH30	305

3.27.2.37 MCPWM_IF_TH31	305
3.27.2.38 MCPWM_IF_TMR0	306
3.27.2.39 MCPWM_IF_TMR1	306
3.27.2.40 MCPWM_IF_TMR2	306
3.27.2.41 MCPWM_IF_TMR3	306
3.27.2.42 MCPWM_MOE_DISABLE_MASK	306
3.27.2.43 MCPWM_MOE_ENABLE_MASK	306
3.27.2.44 MCPWM_SHADE_IRQ_EN	307
3.27.2.45 MCPWM_SHADE_IRQ_IF	307
3.27.2.46 MCPWM_T0_IRQ_EN	307
3.27.2.47 MCPWM_T0_IRQ_IF	307
3.27.2.48 MCPWM_T1_IRQ_EN	307
3.27.2.49 MCPWM_T1_IRQ_IF	307
3.27.2.50 MCPWM_TH00_IRQ_EN	308
3.27.2.51 MCPWM_TH00_IRQ_IF	308
3.27.2.52 MCPWM_TH01_IRQ_EN	308
3.27.2.53 MCPWM_TH01_IRQ_IF	308
3.27.2.54 MCPWM_TH10_IRQ_EN	308
3.27.2.55 MCPWM_TH10_IRQ_IF	308
3.27.2.56 MCPWM_TH11_IRQ_EN	309
3.27.2.57 MCPWM_TH11_IRQ_IF	309
3.27.2.58 MCPWM_TH20_IRQ_EN	309
3.27.2.59 MCPWM_TH20_IRQ_IF	309
3.27.2.60 MCPWM_TH21_IRQ_EN	309
3.27.2.61 MCPWM_TH21_IRQ_IF	309
3.27.2.62 MCPWM_TH30_IRQ_EN	310
3.27.2.63 MCPWM_TH30_IRQ_IF	310
3.27.2.64 MCPWM_TH31_IRQ_EN	310
3.27.2.65 MCPWM_TH31_IRQ_IF	310
3.27.2.66 MCPWM_TMR0_IRQ_EN	310
3.27.2.67 MCPWM_TMR0_IRQ_IF	310
3.27.2.68 MCPWM_TMR1_IRQ_EN	311
3.27.2.69 MCPWM_TMR1_IRQ_IF	311
3.27.2.70 MCPWM_TMR2_IRQ_EN	311
3.27.2.71 MCPWM_TMR2_IRQ_IF	311
3.27.2.72 MCPWM_TMR3_IRQ_EN	311
3.27.2.73 MCPWM_TMR3_IRQ_IF	311
3.27.2.74 MCPWM_UPDATE_REG	312
3.27.3 函数说明	312
3.27.3.1 MCPWM_ClearIRQFlag()	312
3.27.3.2 MCPWM_GetIRQFlag()	314
3.27.3.3 MCPWM_Init()	316

3.27.3.4 MCPWM_StructInit()	317
3.27.3.5 PWMOutputs()	318
3.28 lks32mc08x_mcpwm.h	319
3.29 include/lks32mc08x_nvr.h 文件参考	321
3.29.1 详细描述	322
3.29.2 函数说明	323
3.29.2.1 Read_NVR()	323
3.29.2.2 Read_Trim()	323
3.30 lks32mc08x_nvr.h	324
3.31 include/lks32mc08x_opa.h 文件参考	324
3.31.1 详细描述	325
3.31.2 宏定义说明	326
3.31.2.1 PGA_GAIN_20	326
3.31.2.2 PGA_GAIN_4P25	326
3.31.2.3 PGA_GAIN_6	326
3.31.2.4 PGA_GAIN_9P5	326
3.31.3 枚举类型说明	326
3.31.3.1 enumOPA	326
3.31.4 函数说明	327
3.31.4.1 OPA_Init()	327
3.31.4.2 OPA_OUT()	328
3.31.4.3 OPA_StructInit()	329
3.32 lks32mc08x_opa.h	330
3.33 include/lks32mc08x_spi.h 文件参考	330
3.33.1 详细描述	332
3.33.2 宏定义说明	332
3.33.2.1 SPI_DMA_DISABLE	332
3.33.2.2 SPI_DMA_ENABLE	332
3.33.2.3 SPI_FIRSTSEND_LSB	332
3.33.2.4 SPI_FIRSTSEND_MSB	333
3.33.2.5 SPI_IF_DataOver	333
3.33.2.6 SPI_IF_SSErr	333
3.33.2.7 SPI_IF_TranDone	333
3.33.2.8 SPI_IRQEna_DataOver	333
3.33.2.9 SPI_IRQEna_Enable	333
3.33.2.10 SPI_IRQEna_SSErr	334
3.33.2.11 SPI_IRQEna_TranDone	334
3.33.3 枚举类型说明	334
3.33.3.1 SPI_Duplex	334
3.33.3.2 SPI_Mode	334
3.33.4 函数说明	335
3.33.4.1 SPI_ClearIRQFlag()	335

3.33.4.2 SPI_GetIRQFlag()	336
3.33.4.3 SPI_Init()	337
3.33.4.4 SPI_ReadData()	338
3.33.4.5 SPI_SendData()	339
3.33.4.6 SPI_StructInit()	340
3.34 lks32mc08x_spi.h	341
3.35 include/lks32mc08x_sys.h 文件参考	342
3.35.1 详细描述	346
3.35.2 宏定义说明	346
3.35.2.1 MCU_POWER_3v3	346
3.35.2.2 MCU_POWER_5v0	346
3.35.2.3 POWER_MODE	346
3.35.2.4 SYS_AnalogModule_ADC	346
3.35.2.5 SYS_AnalogModule_BGP	347
3.35.2.6 SYS_AnalogModule_CMP0	347
3.35.2.7 SYS_AnalogModule_CMP1	347
3.35.2.8 SYS_AnalogModule_DAC	347
3.35.2.9 SYS_AnalogModule_OPA0	347
3.35.2.10 SYS_AnalogModule_OPA1	347
3.35.2.11 SYS_AnalogModule_OPA2	348
3.35.2.12 SYS_AnalogModule_OPA3	348
3.35.2.13 SYS_AnalogModule_PLL	348
3.35.2.14 SYS_AnalogModule_RCH	348
3.35.2.15 SYS_AnalogModule_TMP	348
3.35.2.16 SYS_AnalogModule_XTAL	348
3.35.2.17 SYS_CLK_SEL_CLK_HS	349
3.35.2.18 SYS_CLK_SEL_PLL	349
3.35.2.19 SYS_Clk_SPIDiv1	349
3.35.2.20 SYS_Clk_SPIDiv2	349
3.35.2.21 SYS_Clk_SPIDiv4	349
3.35.2.22 SYS_Clk_SPIDiv8	349
3.35.2.23 SYS_Clk_UARTDiv1	350
3.35.2.24 SYS_Clk_UARTDiv2	350
3.35.2.25 SYS_Clk_UARTDiv4	350
3.35.2.26 SYS_Clk_UARTDiv8	350
3.35.2.27 SYS_Module_CAN	350
3.35.2.28 SYS_Module_DSP	350
3.35.2.29 SYS_Module_HALL	351
3.35.2.30 SYS_Module_I2C	351
3.35.2.31 SYS_Module_MCPWM	351
3.35.2.32 SYS_Module_SPI	351
3.35.2.33 SYS_Module_TIMER	351

3.35.2.34 SYS_Module_UART0	351
3.35.2.35 SYS_Module_UART1	352
3.35.2.36 SYS_PLLREDIV_1	352
3.35.2.37 SYS_PLLREDIV_4	352
3.35.2.38 SYS_PLLSRSEL_CRYSTAL	352
3.35.2.39 SYS_PLLSRSEL_RCH	352
3.35.2.40 SYS_RstSrc_HPOR	352
3.35.2.41 SYS_RstSrc_KEYBOARD	353
3.35.2.42 SYS_RstSrc_LPOR	353
3.35.2.43 SYS_RstSrc_WDT	353
3.35.2.44 SYS_WakeInterval_025s	353
3.35.2.45 SYS_WakeInterval_05s	353
3.35.2.46 SYS_WakeInterval_16s	353
3.35.2.47 SYS_WakeInterval_1s	354
3.35.2.48 SYS_WakeInterval_2s	354
3.35.2.49 SYS_WakeInterval_32s	354
3.35.2.50 SYS_WakeInterval_4s	354
3.35.2.51 SYS_WakeInterval_8s	354
3.35.2.52 SYS_WD_TimeOut2s	354
3.35.2.53 SYS_WD_TimeOut4s	355
3.35.2.54 SYS_WD_TimeOut64s	355
3.35.2.55 SYS_WD_TimeOut8s	355
3.35.2.56 WAKEIO_P0_0	355
3.35.2.57 WAKEIO_P0_1	355
3.35.2.58 WAKEIO_P1_0	355
3.35.2.59 WAKEIO_P1_1	355
3.35.3 枚举类型说明	355
3.35.3.1 enumWakeTrigLevel	355
3.35.4 函数说明	356
3.35.4.1 SetWakeIO()	356
3.35.4.2 Switch2HRC()	357
3.35.4.3 Switch2PLL()	358
3.35.4.4 SYS_AnalogModuleClockCmd()	359
3.35.4.5 SYS_ClearRst()	361
3.35.4.6 SYS_FallSleep()	362
3.35.4.7 SYS_FeedDog()	363
3.35.4.8 SYS_GetRstSource()	364
3.35.4.9 SYS_Init()	365
3.35.4.10 SYS_ModuleClockCmd()	366
3.35.4.11 SYS_ResetIO()	368
3.35.4.12 SYS_ResetPeripheral()	368
3.35.4.13 SYS_ResetPeripheralInternal()	368

3.35.4.14 SYS_SetWakeInterval()	368
3.35.4.15 SYS_SoftResetModule()	369
3.35.4.16 SYS_StructInit()	371
3.35.4.17 SYS_VolSelModule()	372
3.35.4.18 SYS_VolSelModuleEnableIRQ()	372
3.35.4.19 SYS_VolSelModuleIRQ()	372
3.35.4.20 SYS_WatchDogCmd()	372
3.36 lks32mc08x_sys.h	373
3.37 include/lks32mc08x_tim.h 文件参考	375
3.37.1 详细描述	378
3.37.2 宏定义说明	378
3.37.2.1 ECD_Clk_Div1	378
3.37.2.2 ECD_Clk_Div2	378
3.37.2.3 ECD_Clk_Div4	379
3.37.2.4 ECD_Clk_Div8	379
3.37.2.5 ECD_IF_OVERFLOW	379
3.37.2.6 ECD_IF_UNDERFLOW	379
3.37.2.7 ECD_IRQEna_None	379
3.37.2.8 ECD_IRQEna_OverFlow	379
3.37.2.9 ECD_IRQEna_UnderFlow	380
3.37.2.10 ECD_Mode_CCWCW	380
3.37.2.11 ECD_Mode_CCWCW_UpDown	380
3.37.2.12 ECD_Mode_CCWSIGN	380
3.37.2.13 ECD_Mode_CCWSIGN_UpDown	380
3.37.2.14 ECD_Mode_T1	380
3.37.2.15 ECD_Mode_T1_T2	381
3.37.2.16 TIM_Clk_Div1	381
3.37.2.17 TIM_Clk_Div2	381
3.37.2.18 TIM_Clk_Div4	381
3.37.2.19 TIM_Clk_Div8	381
3.37.2.20 TIMER_CapMode_Fall	381
3.37.2.21 TIMER_CapMode_None	382
3.37.2.22 TIMER_CapMode_Rise	382
3.37.2.23 TIMER_CapMode_RiseFall	382
3.37.2.24 TIMER_IF_CH0	382
3.37.2.25 TIMER_IF_CH1	382
3.37.2.26 TIMER_IF_ZERO	382
3.37.2.27 TIMER_OPMODE_CAP	383
3.37.2.28 TIMER_OPMODE_CMP	383
3.37.3 枚举类型说明	383
3.37.3.1 Timer_IRQEnaDef	383
3.37.4 函数说明	383

3.37.4.1 ECD_ClearIRQFlag()	383
3.37.4.2 ECD_GetIRQFlag()	384
3.37.4.3 TIM_ClearIRQFlag()	385
3.37.4.4 TIM_ECD_GetCount()	386
3.37.4.5 TIM_ECDCmd()	387
3.37.4.6 TIM_ECDInit()	388
3.37.4.7 TIM_ECDStructInit()	390
3.37.4.8 TIM_GetIRQFlag()	391
3.37.4.9 TIM_Timer_GetCMPT0()	392
3.37.4.10 TIM_Timer_GetCMPT1()	393
3.37.4.11 TIM_Timer_GetCount()	394
3.37.4.12 TIM_TimerCmd()	395
3.37.4.13 TIM_TimerInit()	396
3.37.4.14 TIM_TimerStrutInit()	397
3.38 lks32mc08x_tim.h	398
3.39 include/lks32mc08x_tmp.h 文件参考	400
3.39.1 详细描述	401
3.39.2 函数说明	401
3.39.2.1 GetCurrentTemperature()	401
3.39.2.2 TempSensor_Init()	402
3.40 lks32mc08x_tmp.h	403
3.41 include/lks32mc08x_uart.h 文件参考	404
3.41.1 详细描述	406
3.41.2 宏定义说明	406
3.41.2.1 UART_FIRSTSEND_LSB	406
3.41.2.2 UART_FIRSTSEND_MSB	406
3.41.2.3 UART_IF_CheckError	406
3.41.2.4 UART_IF_RcvOver	407
3.41.2.5 UART_IF_SendBufEmpty	407
3.41.2.6 UART_IF_SendOver	407
3.41.2.7 UART_IF_StopError	407
3.41.2.8 UART_IRQEna_CheckError	407
3.41.2.9 UART_IRQEna_RcvOver	407
3.41.2.10 UART_IRQEna_RX_DMA_RE	408
3.41.2.11 UART_IRQEna_SendBuffEmpty	408
3.41.2.12 UART_IRQEna_SendOver	408
3.41.2.13 UART_IRQEna_StopError	408
3.41.2.14 UART_IRQEna_TX_BUF_DMA_RE	408
3.41.2.15 UART_IRQEna_TX_DMA_RE	408
3.41.2.16 UART_STOPBITS_1b	409
3.41.2.17 UART_STOPBITS_2b	409
3.41.2.18 UART_WORDLENGTH_7b	409

3.41.2.19 UART_WORDLENGTH_8b	409
3.41.3 枚举类型说明	409
3.41.3.1 UART_ParityMode	409
3.41.4 函数说明	410
3.41.4.1 UART_ClearIRQFlag()	410
3.41.4.2 UART_GetIRQFlag()	411
3.41.4.3 UART_Init()	412
3.41.4.4 UART_ReadData()	413
3.41.4.5 UART_SendAddr_485()	414
3.41.4.6 UART_SendData()	415
3.41.4.7 UART_SendData_485()	416
3.41.4.8 UART_StructInit()	417
3.42 lks32mc08x_uart.h	418
3.43 source/lks32mc08x_adc.c 文件参考	419
3.43.1 详细描述	420
3.43.2 函数说明	420
3.43.2.1 ADC_ChannelConfig()	420
3.43.2.2 ADC_ClearIRQFlag()	421
3.43.2.3 ADC_GetConversionValue()	423
3.43.2.4 ADC_GetIRQFlag()	424
3.43.2.5 ADC_Init()	425
3.43.2.6 ADC_SoftTrgEN()	426
3.43.2.7 ADC_StructInit()	427
3.44 lks32mc08x_adc.c	428
3.45 source/lks32mc08x_can.c 文件参考	431
3.45.1 详细描述	432
3.45.2 函数说明	432
3.45.2.1 CAN_GetIRQFlag()	432
3.45.2.2 CAN_Init()	433
3.45.2.3 CAN_Manual_Awake()	435
3.45.2.4 CAN_ReadALC()	435
3.45.2.5 CAN_ReadErrorCode()	436
3.45.2.6 CAN_ReadRXERRCNT()	437
3.45.2.7 CAN_ReadState()	438
3.45.2.8 CAN_ReadTXERRCNT()	440
3.45.2.9 CAN_ReceFramInfor()	441
3.45.2.10 CAN_Receive_Msg()	442
3.45.2.11 CAN_Sleep()	443
3.45.2.12 CAN_StructInit()	444
3.45.2.13 EFF_ID_Filter()	445
3.45.2.14 ID1_Filter_Dual()	446
3.45.2.15 ID2_Filter_Dual()	447

3.45.2.16 My_CAN_Send_Msg()	449
3.45.2.17 SFF_ID_Filter()	450
3.46 lks32mc08x_can.c	451
3.47 source/lks32mc08x_cmp.c 文件参考	455
3.47.1 详细描述	456
3.47.2 函数说明	456
3.47.2.1 CMP_ClearIRQFlag()	456
3.47.2.2 CMP_GetIRQFlag()	457
3.47.2.3 CMP_Init()	458
3.47.2.4 CMP_StructInit()	460
3.48 lks32mc08x_cmp.c	461
3.49 source/lks32mc08x_dac.c 文件参考	462
3.49.1 详细描述	463
3.49.2 函数说明	463
3.49.2.1 DAC_Init()	463
3.49.2.2 DAC_Output()	464
3.49.2.3 DAC_OutputValue()	465
3.49.2.4 DAC_OutputVoltage()	465
3.49.2.5 DAC_StructInit()	466
3.50 lks32mc08x_dac.c	467
3.51 source/lks32mc08x_dma.c 文件参考	468
3.51.1 详细描述	469
3.51.2 函数说明	470
3.51.2.1 DMA_CHx_EN()	470
3.51.2.2 DMA_ClearIRQFlag()	471
3.51.2.3 DMA_GetIRQFlag()	472
3.51.2.4 DMA_Init()	473
3.51.2.5 DMA_StructInit()	475
3.52 lks32mc08x_dma.c	476
3.53 source/lks32mc08x_dsp.c 文件参考	477
3.53.1 详细描述	478
3.53.2 函数说明	478
3.53.2.1 DSP_arctan()	478
3.53.2.2 DSP_CalcCordic()	478
3.53.2.3 DSP_CalcDivision()	480
3.53.2.4 DSP_CalcSqrt()	481
3.53.2.5 DSP_Cmd()	482
3.53.2.6 DSP_cos()	483
3.53.2.7 DSP_div()	483
3.53.2.8 DSP_GetSqrt()	484
3.53.2.9 DSP_Init()	485
3.53.2.10 DSP_mod()	485

3.53.2.11 DSP_rms()	485
3.53.2.12 DSP_sin()	485
3.53.2.13 DSP_sqrt()	486
3.54 lks32mc08x_dsp.c	486
3.55 source/lks32mc08x_exti.c 文件参考	488
3.55.1 详细描述	488
3.55.2 函数说明	489
3.55.2.1 EXTI_ClearIRQFlag()	489
3.55.2.2 EXTI_GetIRQFlag()	490
3.55.2.3 EXTI_Trigger_Config()	491
3.56 lks32mc08x_exti.c	492
3.57 source/lks32mc08x_flash.c 文件参考	493
3.57.1 详细描述	494
3.57.2 函数说明	494
3.57.2.1 EraseSector()	494
3.57.2.2 ProgramPage()	495
3.57.2.3 Read_Flash()	496
3.57.2.4 Read_More_Flash()	498
3.57.3 变量说明	499
3.57.3.1 erase_flag	499
3.57.3.2 progmem_flag	499
3.58 lks32mc08x_flash.c	500
3.59 source/lks32mc08x_gpio.c 文件参考	501
3.59.1 详细描述	503
3.59.2 函数说明	503
3.59.2.1 GPIO_Config()	503
3.59.2.2 GPIO_Init()	504
3.59.2.3 GPIO_LockRegister()	505
3.59.2.4 GPIO_PinAFConfig()	506
3.59.2.5 GPIO_ReadInputData()	508
3.59.2.6 GPIO_ReadInputDataBit()	509
3.59.2.7 GPIO_ReadOutputData()	510
3.59.2.8 GPIO_ReadOutputDataBit()	511
3.59.2.9 GPIO_ReadPinBit()	512
3.59.2.10 GPIO_ResetBits()	512
3.59.2.11 GPIO_SetBits()	513
3.59.2.12 GPIO_StructInit()	514
3.59.2.13 GPIO_UnlockRegister()	516
3.59.2.14 GPIO_Write()	516
3.59.2.15 GPIO_WriteBit()	517
3.59.2.16 GPIO_WritePinBit()	519
3.60 lks32mc08x_gpio.c	519

3.61 source/lks32mc08x_hall.c 文件参考	523
3.61.1 详细描述	524
3.61.2 函数说明	524
3.61.2.1 HALL_ClearIRQFlag()	524
3.61.2.2 HALL_GetCaptureValue()	525
3.61.2.3 HALL_GetCount()	526
3.61.2.4 HALL_GetFilterValue()	527
3.61.2.5 HALL_GetIRQFlag()	528
3.61.2.6 HALL_Init()	529
3.61.2.7 HALL_StructInit()	530
3.61.2.8 HALL_WIDCount()	531
3.62 lks32mc08x_hall.c	532
3.63 source/lks32mc08x_i2c.c 文件参考	533
3.63.1 详细描述	534
3.63.2 函数说明	534
3.63.2.1 i2c_dma_delay_over()	534
3.63.2.2 i2c_dma_init()	535
3.63.2.3 i2c_dma_rx()	537
3.63.2.4 i2c_dma_state_over()	538
3.63.2.5 i2c_dma_state_start()	539
3.63.2.6 i2c_dma_tx()	540
3.63.2.7 I2C_Init()	541
3.63.2.8 I2C_ReadData()	543
3.63.2.9 I2C_SendData()	544
3.63.2.10 I2C_StructInit()	545
3.63.3 变量说明	546
3.63.3.1 i2c_dma_state	546
3.64 lks32mc08x_i2c.c	546
3.65 source/lks32mc08x_iwdg.c 文件参考	548
3.65.1 详细描述	548
3.65.2 函数说明	549
3.65.2.1 IWDG_DISABLE()	549
3.65.2.2 IWDG_ENABLE()	550
3.65.2.3 IWDG_Feed()	551
3.65.2.4 IWDG_Init()	552
3.65.2.5 IWDG_StructInit()	553
3.66 lks32mc08x_iwdg.c	554
3.67 source/lks32mc08x_mcpwm.c 文件参考	554
3.67.1 详细描述	555
3.67.2 函数说明	556
3.67.2.1 MCPWM_ClearIRQFlag()	556
3.67.2.2 MCPWM_GetIRQFlag()	558

3.67.2.3 MCPWM_Init()	560
3.67.2.4 MCPWM_StructInit()	561
3.67.2.5 PWMOutputs()	562
3.68 lks32mc08x_mcpwm.c	563
3.69 source/lks32mc08x_opa.c 文件参考	565
3.69.1 详细描述	565
3.69.2 函数说明	566
3.69.2.1 OPA_Init()	566
3.69.2.2 OPA_OUT()	567
3.69.2.3 OPA_StructInit()	568
3.70 lks32mc08x_opa.c	569
3.71 source/lks32mc08x_spi.c 文件参考	570
3.71.1 详细描述	571
3.71.2 函数说明	571
3.71.2.1 SPI_ClearIRQFlag()	571
3.71.2.2 SPI_GetIRQFlag()	572
3.71.2.3 SPI_Init()	573
3.71.2.4 SPI_ReadData()	575
3.71.2.5 SPI_SendData()	575
3.71.2.6 SPI_StructInit()	576
3.72 lks32mc08x_spi.c	577
3.73 source/lks32mc08x_sys.c 文件参考	578
3.73.1 详细描述	579
3.73.2 函数说明	580
3.73.2.1 SetWakeIO()	580
3.73.2.2 SoftDelay()	581
3.73.2.3 Switch2HRC()	582
3.73.2.4 Switch2PLL()	583
3.73.2.5 SYS_AnalogModuleClockCmd()	584
3.73.2.6 SYS_ClearRst()	586
3.73.2.7 SYS_EnablePORFilter()	587
3.73.2.8 SYS_FallSleep()	588
3.73.2.9 SYS_FeedDog()	589
3.73.2.10 SYS_GetRstSource()	590
3.73.2.11 SYS_Init()	591
3.73.2.12 SYS_ModuleClockCmd()	592
3.73.2.13 SYS_SetWakeInterval()	594
3.73.2.14 SYS_SoftResetModule()	595
3.73.2.15 SYS_StructInit()	596
3.73.2.16 SYS_WatchDogCmd()	597
3.73.3 变量说明	598
3.73.3.1 WAKE_InitStruct	598

3.74 lks32mc08x_sys.c	598
3.75 source/lks32mc08x_tim.c 文件参考	603
3.75.1 详细描述	604
3.75.2 函数说明	605
3.75.2.1 ECD_ClearIRQFlag()	605
3.75.2.2 ECD_GetIRQFlag()	606
3.75.2.3 TIM_ClearIRQFlag()	607
3.75.2.4 TIM_ECD_GetCount()	608
3.75.2.5 TIM_ECDCmd()	609
3.75.2.6 TIM_ECDInit()	610
3.75.2.7 TIM_ECDStructInit()	611
3.75.2.8 TIM_GetIRQFlag()	612
3.75.2.9 TIM_Timer_GetCMPT0()	613
3.75.2.10 TIM_Timer_GetCMPT1()	614
3.75.2.11 TIM_Timer_GetCount()	615
3.75.2.12 TIM_TimerCmd()	616
3.75.2.13 TIM_TimerInit()	617
3.75.2.14 TIM_TimerStrutInit()	619
3.76 lks32mc08x_tim.c	620
3.77 source/lks32mc08x_tmp.c 文件参考	624
3.77.1 详细描述	625
3.77.2 函数说明	625
3.77.2.1 GetCurrentTemperature()	625
3.77.2.2 TempSensor_Init()	626
3.77.3 变量说明	627
3.77.3.1 m_TempertureCof	627
3.78 lks32mc08x_tmp.c	628
3.79 source/lks32mc08x_uart.c 文件参考	628
3.79.1 详细描述	629
3.79.2 函数说明	629
3.79.2.1 UART_ClearIRQFlag()	629
3.79.2.2 UART_GetIRQFlag()	630
3.79.2.3 UART_Init()	631
3.79.2.4 UART_ReadData()	633
3.79.2.5 UART_SendAddr_485()	634
3.79.2.6 UART_SendData()	635
3.79.2.7 UART_SendData_485()	636
3.79.2.8 UART_StructInit()	637
3.80 lks32mc08x_uart.c	638

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初始化结构体句柄，初始化时请定义该句柄，并用其它地址来传参	9
ADC_TypeDef	ADC寄存器结构体句柄	15
CAN_InitTypeDef	CAN结构体基地址定义	25
CAN_TypeDef	CAN寄存器结构体句柄	27
CMP_InitTypeDef	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	33
CMP_TypeDef	CMP寄存器结构体句柄	42
DAC_InitTypeDef		43
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	44
DMA_RegTypeDef	DAM通道0结构体基地址定义	48
EXTI_TypeDef	EXTI寄存器结构体句柄	49
GPIO_InitTypeDef	GPIO功能配置结构体句柄	50
GPIO_TypeDef	GPIO寄存器结构体句柄	51
HALL_InitTypeDef	HALL结构体基地址定义	54
HALL_TypeDef	HALL寄存器结构体句柄	56
I2C_InitTypeDef	I2C结构体基地址定义	57
I2C_TypeDef	I2C寄存器结构体句柄	60
IWDG_InitTypeDef	IWDG寄存器结构体句柄	61
MCPWM_InitTypeDef	MCPWM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	62
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
SPI_InitTypeDef	SPI结构体基地址定义	85
SPI_TypeDef	SPI寄存器结构体句柄	89
stru_CordicComponents	三角函数运算结构体句柄	90
stru_DivComponents	除法运算结构体句柄	92
Stru_TemperatureCof_Def	芯片内置温度传感器初始化结构体句柄，初始化时请定义该句柄，并用其它地址来传参	93
SYS_InitTypeDef	系统初始化结构体句柄，初始化时请定义该句柄，并用其它地址来传参	94
TIM_ECDInitTypeDef		97
TIM_ECDTypeDef	编码器寄存器结构体句柄	99
TIM_TimerInitTypeDef		100

TIM_TimerTypeDef	
定时器寄存器结构体句柄	105
UART_InitTypeDef	
UART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	107
UART_TypeDef	
UART寄存器结构体句柄	110
WAKE_InitTypeDef	
低功耗时钟配置中间变量	112

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/lks32mc08x_adc.h`

文件名称: `lks32mc08x_adc.h`

文件标识: 无

内容摘要: ADC驱动头文件

其它说明: 无

113

`include/lks32mc08x_can.h`

文件名称: `lks32mc08x_can.h`

文件标识: 无

内容摘要: CAN外设驱动程序头文件

其它说明: 无

143

`include/lks32mc08x_cmp.h`

文件名称: `lks32mc08x_cmp.h`

文件标识: 无

内容摘要: CMP外设驱动程序头文件

其它说明: 无

168

`include/lks32mc08x_dac.h`

文件名称: `lks32mc08x_dac.h`

文件标识: 无

内容摘要: DAC外设驱动程序头文件

其它说明: 无

184

`include/lks32mc08x_dma.h`

文件名称: `lks32mc08x_dma.h`

文件标识: 无

内容摘要: DMA外设驱动程序头文件

其它说明: 无

190

`include/lks32mc08x_dsp.h`

文件名称: `lks32mc08x_dsp.h`

文件标识: 无

内容摘要: DSP驱动头文件

其它说明: 无

204

include/lks32mc08x_exti.h

文件名称: [lks32mc08x_exti.h](#)
文件标识: 无
内容摘要: EXTI外部中断驱动头文件
其它说明: 无

[217](#)**include/lks32mc08x_flash.h**

文件名称: [LKS32MC08x_flash.h](#)
文件标识: 无
内容摘要: FLASH外设驱动程序
其它说明: 无

[223](#)**include/lks32mc08x_gpio.h**

文件名称: [lks32mc08x_gpio.h](#)
文件标识: 无
内容摘要: GPIO外设驱动程序头文件
其它说明: 无

[231](#)**include/lks32mc08x_hall.h**

文件名称: [lks32mc08x_hall.h](#)
文件标识: 无
内容摘要: HALL驱动头文件
其它说明: 无

[263](#)**include/lks32mc08x_i2c.h**

文件名称: [lks32mc08x_i2c.h](#)
文件标识: 无
内容摘要: I2C驱动头文件
其它说明: 无

[274](#)**include/lks32mc08x_iwdg.h**

文件名称: [lks32mc08x_IWDG.h](#)
文件标识: 无
内容摘要: 看门狗驱动头文件
其它说明: 无

[286](#)**include/lks32mc08x_lib.h**

文件名称: [LKS32MC08x_lib.h](#)
文件标识: 无
内容摘要: LIB库函数头文件
其它说明: 无

[293](#)**include/lks32mc08x_mcpwm.h**

文件名称: [lks32mc08x_mcpwm.h](#)
文件标识: 无
内容摘要: mcpwm驱动头文件
其它说明: 无

[295](#)**include/lks32mc08x_nvr.h**

文件名称: [lks32mc08x_nvr.h](#)
文件标识: 无
内容摘要: NVR驱动头文件
其它说明: 无

[321](#)

- [include/lks32mc08x_opa.h](#)
文件名称: [lks32mc08x_opa.h](#)
文件标识: 无
内容摘要: OPA驱动头文件
其它说明: 无
[324](#)
- [include/lks32mc08x_spi.h](#)
文件名称: [lks32mc08x_spi.h](#)
文件标识: 无
内容摘要: SPI驱动头文件
其它说明: 无
[330](#)
- [include/lks32mc08x_sys.h](#)
文件名称: [lks32mc08x_sys.h](#)
文件标识: 无
内容摘要: sys驱动头文件
其它说明: 无
[342](#)
- [include/lks32mc08x_tim.h](#)
文件名称: [lks32mc08x_tim.h](#)
文件标识: 无
内容摘要: TIM驱动头文件
其它说明: 无
[375](#)
- [include/lks32mc08x_tmp.h](#)
文件名称: [lks32mc08x_tmp.h](#)
文件标识: 无
内容摘要: 芯片内置温度传感器驱动头文件
其它说明: 无
[400](#)
- [include/lks32mc08x_uart.h](#)
文件名称: [lks32mc08x_uart.h](#)
文件标识: 无
内容摘要: UART驱动头文件
其它说明: 无
[404](#)
- [source/lks32mc08x_adc.c](#)
文件名称: [lks32mc08x_adc.c](#)
文件标识: 无
内容摘要: ADC外设驱动程序
其它说明: 无
[419](#)
- [source/lks32mc08x_can.c](#)
文件名称: [lks32mc08x_can.c](#)
文件标识: 无
内容摘要: CAN外设驱动程序
其它说明: 无
[431](#)
- [source/lks32mc08x_cmp.c](#)
文件名称: [lks32mc08x_cmp.c](#)
文件标识: 无
内容摘要: cmp外设驱动程序
其它说明: 无
[455](#)

[source/lks32mc08x_dac.c](#)

文件名称: LKS32MC08x_dac.c
文件标识: 无
内容摘要: DAC外设驱动程序
其它说明: 无

[462](#)[source/lks32mc08x_dma.c](#)

文件名称: LKS32MC08x_dma.c
文件标识: 无
内容摘要: DMA外设驱动程序
其它说明: 无

[468](#)[source/lks32mc08x_dsp.c](#)

文件名称: [lks32mc08x_dsp.c](#)
文件标识: 无
内容摘要: DSP驱动程序
其它说明: 无

[477](#)[source/lks32mc08x_exti.c](#)

文件名称: [lks32mc08x_exti.c](#)
文件标识: 无
内容摘要: EXTI外设驱动程序
其它说明: 无

[488](#)[source/lks32mc08x_flash.c](#)

文件名称: [lks32mc08x_flash.c](#)
文件标识: 无
内容摘要: FLASH外设驱动程序
其它说明: 无

[493](#)[source/lks32mc08x_gpio.c](#)

文件名称: [lks32mc08x_gpio.c](#)
文件标识: 无
内容摘要: GPIO外设驱动程序
其它说明: 无

[501](#)[source/lks32mc08x_hall.c](#)

文件名称: [lks32mc08x_hall.c](#)
文件标识: 无
内容摘要: HALL外设驱动程序
其它说明: 无

[523](#)[source/lks32mc08x_i2c.c](#)

文件名称: [lks32mc08x_i2c.c](#)
文件标识: 无
内容摘要: I2C外设驱动程序
其它说明: 无

[533](#)[source/lks32mc08x_iwdg.c](#)

文件名称: [lks32mc08x_IWDG.c](#)
文件标识: 无
内容摘要: IWDG外设驱动程序
其它说明: 无

[548](#)

source/lks32mc08x_mcpwm.c

文件名称: [lks32mc08x_mcpwm.c](#)
文件标识: 无
内容摘要: MCPWM外设驱动程序
其它说明: 无

[554](#)**source/lks32mc08x_opa.c**

文件名称: [lks32mc08x_opa.c](#)
文件标识: 无
内容摘要: OPA运放外设驱动程序
其它说明: 无

[565](#)**source/lks32mc08x_spi.c**

文件名称: [lks32mc08x_spi.c](#)
文件标识: 无
内容摘要: SPI外设驱动程序
其它说明: 无

[570](#)**source/lks32mc08x_sys.c**

文件名称: [lks32mc08x_sys.c](#)
文件标识: 无
内容摘要: SYS外设驱动程序
其它说明: 无

[578](#)**source/lks32mc08x_tim.c**

文件名称: [lks32mc08x_tim.c](#)
文件标识: 无
内容摘要: 定时器外设驱动程序
其它说明: 无

[603](#)**source/lks32mc08x_tmp.c**

文件名称: [lks32mc08x_tmp.c](#)
文件标识: 无
内容摘要: 芯片内置温度传感器外设驱动程序
其它说明: 无

[624](#)**source/lks32mc08x_uart.c**

文件名称: [lks32mc08x_uart.c](#)
文件标识: 无
内容摘要: UART外设驱动程序
其它说明: 无

[628](#)

Chapter 2

结构体说明

2.1 ADC_InitTypeDef结构体 参考

ADC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 <lks32mc08x_adc.h>
```

成员变量

- uint16_t **IE**
ADC中断使能
- uint16_t **Align**
采样数据对齐方式
- uint16_t **ADC_CLK_DIV**
ADC时钟分频
- uint16_t **Con_Sample**
连续采样模式 *DISABLE*:关闭, *ENABLE*:使能
- uint16_t **Trigger_Cnt**
*Trigger_Cnt*设置1, 即需要触发ADC两次采进行一次ADC采样转换, 设置范围0~15, 表示需要1~16次触发事件
- uint16_t **FirSeg_Ch**
使用该段采样对应段的采样个数不可设置为0, 如果未用该段采样, 采样个数可设置为0。
- uint16_t **SecSeg_Ch**
使用该段采样对应段的采样个数不可设置为0, 如果未用该段采样, 采样个数可设置为0。
- uint16_t **ThrSeg_Ch**
使用该段采样对应段的采样个数不可设置为0, 如果未用该段采样, 采样个数可设置为0。
- uint16_t **FouSeg_Ch**
使用该段采样对应段的采样个数不可设置为0, 如果未用该段采样, 采样个数可设置为0。
- uint16_t **Trigger_Mode**
ADC 触发模式 单段、双段或四段
- uint16_t **MCPWM_Trigger_En**

*MCPWM*触发采样使能

- uint16_t [UTIMER_Trigger_En](#)

*UTIMER*触发采样使能

- uint16_t [DAT0_TH_Type](#)

设置*DAT0_TH*作为: 1上阈值, 0下阈值

- uint16_t [DAT0_TH](#)

通道0阈值设置

- uint16_t [Gain0](#)

*ADC_DAT0-ADC_DAT9*采样保持电路增益 *BIT0~BIT9*可以或*BIT0|BIT1*

- uint16_t [Gain1](#)

*ADC_DAT10-ADC_DAT19*采样保持电路增益 *BIT0~BIT9*可以或*BIT0|BIT1*

- uint16_t [RefVol](#)

基准电压选择

2.1.1 详细描述

ADC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 [lks32mc08x_adc.h](#) 第 89 行定义.

2.1.2 结构体成员变量说明

2.1.2.1 ADC_CLK_DIV

uint16_t ADC_CLK_DIV

ADC时钟分频

参见

[ADC_Clock_48M](#)

[ADC_Clock_24M](#)

[ADC_Clock_12M](#)

在文件 [lks32mc08x_adc.h](#) 第 114 行定义.

2.1.2.2 Align

uint16_t Align

采样数据对齐方式

参见

[ADC_LEFT_ALIGN](#)

[ADC_RIGHT_ALIGN](#)

在文件 [lks32mc08x_adc.h](#) 第 107 行定义.

2.1.2.3 Con_Sample

uint16_t Con_Sample

连续采样模式 DISABLE:关闭, ENABLE:使能

在文件 [lks32mc08x_adc.h](#) 第 115 行定义.

2.1.2.4 DAT0.TH

uint16_t DAT0.TH

通道0阈值设置

在文件 [lks32mc08x_adc.h](#) 第 161 行定义.

2.1.2.5 DAT0.TH_Type

uint16_t DAT0.TH_Type

设置DAT0.TH作为: 1上阈值, 0下阈值

在文件 [lks32mc08x_adc.h](#) 第 160 行定义.

2.1.2.6 FirSeg.Ch

uint16_t FirSeg.Ch

使用该段采样对应段的采样个数不可设置为0, 如果未用该段采样, 采样个数可设置为0。

第一段采样的总通道数

在文件 [lks32mc08x_adc.h](#) 第 124 行定义.

2.1.2.7 FouSeg_Ch

uint16_t FouSeg_Ch

使用该段采样对应段的采样个数不可设置为0，如果未用该段采样，采样个数可设置为0。

第四段采样的总通道数

在文件 [lks32mc08x_adc.h](#) 第 136 行定义.

2.1.2.8 Gain0

uint16_t Gain0

ADC_DAT0-ADC_DAT9采样保持电路增益 BIT0~BIT9可以或BIT0|BIT1

在文件 [lks32mc08x_adc.h](#) 第 162 行定义.

2.1.2.9 Gain1

uint16_t Gain1

ADC_DAT10-ADC_DAT19采样保持电路增益 BIT0~BIT9可以或BIT0|BIT1

在文件 [lks32mc08x_adc.h](#) 第 163 行定义.

2.1.2.10 IE

uint16_t IE

ADC中断使能

参见

[ADC_DAT0_OV_IRQ_EN](#)

[ADC_H_CONFLICT_IRQ_EN](#)

[ADC_S_CONFLICT_IRQ_EN](#)

[ADC_EOS3_IRQ_EN](#)

[ADC_EOS2_IRQ_EN](#)

[ADC_EOS1_IRQ_EN](#)

[ADC_EOS0_IRQ_EN](#)

在文件 [lks32mc08x_adc.h](#) 第 101 行定义.

2.1.2.11 MCPWM_Trigger_En

uint16_t MCPWM_Trigger_En

MCPWM触发采样使能

参见

[ADC_MCPWM_T0_TRG](#)

[ADC_MCPWM_T1_TRG](#)

[ADC_MCPWM_T2_TRG](#)

[ADC_MCPWM_T3_TRG](#)

在文件 [lks32mc08x_adc.h](#) 第 151 行定义.

2.1.2.12 RefVol

uint16_t RefVol

基准电压选择

参见

[RefVol_1V2](#)

[RefVol_2V4](#)

在文件 [lks32mc08x_adc.h](#) 第 169 行定义.

2.1.2.13 SecSeg_Ch

uint16_t SecSeg_Ch

使用该段采样对应段的采样个数不可设置为0，如果未用该段采样，采样个数可设置为0。

第二段采样的总通道数

在文件 [lks32mc08x_adc.h](#) 第 128 行定义.

2.1.2.14 ThrSeg_Ch

uint16_t ThrSeg_Ch

使用该段采样对应段的采样个数不可设置为0，如果未用该段采样，采样个数可设置为0。

第三段采样的总通道数

在文件 [lks32mc08x_adc.h](#) 第 132 行定义.

2.1.2.15 Trigger_Cnt

uint16_t Trigger_Cnt

Trigger_Cnt设置1，即需要触发ADC两次采进行一次ADC采样转换，设置范围0~15，表示需要1~16次触发事件

单段模式下触发下一次采样所需的触发事件数

在文件 [lks32mc08x_adc.h](#) 第 119 行定义.

2.1.2.16 Trigger_Mode

uint16_t Trigger_Mode

ADC 触发模式 单段、双段或四段

参见

[ADC_1SEG_TRG](#)

[ADC_2SEG_TRG](#)

[ADC_4SEG_TRG](#)

在文件 [lks32mc08x_adc.h](#) 第 143 行定义.

2.1.2.17 UTIMER_Trigger_En

uint16_t UTIMER_Trigger_En

UTIMER触发采样使能

参见

[ADC_UTIMER_T0_TRG](#)

[ADC_UTIMER_T1_TRG](#)

[ADC_UTIMER_T2_TRG](#)

[ADC_UTIMER_T3_TRG](#)

在文件 [lks32mc08x_adc.h](#) 第 159 行定义.

2.2 ADC_TypeDef结构体 参考

ADC寄存器结构体句柄

```
#include <lks32mc08x_adc.h>
```

成员变量

- `__IO uint32_t DAT0`
ADC数据寄存器0
- `__IO uint32_t DAT1`
ADC数据寄存器1
- `__IO uint32_t DAT2`
ADC数据寄存器2
- `__IO uint32_t DAT3`
ADC数据寄存器3
- `__IO uint32_t DAT4`
ADC数据寄存器4
- `__IO uint32_t DAT5`
ADC数据寄存器5
- `__IO uint32_t DAT6`
ADC数据寄存器6
- `__IO uint32_t DAT7`
ADC数据寄存器7
- `__IO uint32_t DAT8`
ADC数据寄存器8
- `__IO uint32_t DAT9`
ADC数据寄存器9
- `__IO uint32_t DAT10`
ADC数据寄存器10
- `__IO uint32_t DAT11`
ADC数据寄存器11
- `__IO uint32_t DAT12`
ADC数据寄存器12
- `__IO uint32_t DAT13`
ADC数据寄存器13
- `__IO uint32_t DAT14`
ADC数据寄存器14
- `__IO uint32_t DAT15`
ADC数据寄存器15
- `__IO uint32_t DAT16`
ADC数据寄存器16
- `__IO uint32_t DAT17`
ADC数据寄存器17
- `__IO uint32_t DAT18`
ADC数据寄存器18
- `__IO uint32_t DAT19`
ADC数据寄存器19
- `__IO uint32_t CHN0`
ADC采样信号来源寄存器0

- `__IO uint32_t CHN1`
ADC采样信号来源寄存器1
- `__IO uint32_t CHN2`
ADC采样信号来源寄存器2
- `__IO uint32_t CHN3`
ADC采样信号来源寄存器3
- `__IO uint32_t CHN4`
ADC采样信号来源寄存器4
- `__IO uint32_t CHN5`
ADC采样信号来源寄存器5
- `__IO uint32_t CHN6`
ADC采样信号来源寄存器6
- `__IO uint32_t CHN7`
ADC采样信号来源寄存器7
- `__IO uint32_t CHN8`
ADC采样信号来源寄存器8
- `__IO uint32_t CHN9`
ADC采样信号来源寄存器9
- `__IO uint32_t CHNT0`
ADC分段通道数寄存器0 设置每段采样数量
- `__IO uint32_t CHNT1`
ADC分段通道数寄存器1 设置每段采样数量
- `__IO uint32_t SD0`
保留
- `__IO uint32_t SD1`
保留
- `__IO uint32_t GAIN_0`
ADC增益选择寄存器0
- `__IO uint32_t GAIN_1`
ADC增益选择寄存器1
- `__IO uint32_t DC_A0`
ADC直流偏置寄存器0 采样保持电路A对应2/3倍增益通道偏置
- `__IO uint32_t DC_A1`
ADC直流偏置寄存器1 采样保持电路A对应1倍增益通道偏置
- `__IO uint32_t AMC_A0`
ADC增益校正寄存器0 采样保持电路A对应2/3倍增益通道增益校正
- `__IO uint32_t AMC_A1`
ADC增益校正寄存器0 采样保持电路A对应1倍增益通道增益校正
- `__IO uint32_t DC_B0`
ADC直流偏置寄存器0 采样保持电路B对应2/3倍增益通道偏置
- `__IO uint32_t DC_B1`
ADC直流偏置寄存器1 采样保持电路B对应1倍增益通道偏置
- `__IO uint32_t AMC_B0`
ADC增益校正寄存器0 采样保持电路B对应2/3倍增益通道增益校正
- `__IO uint32_t AMC_B1`
ADC增益校正寄存器0 采样保持电路B对应1倍增益通道增益校正
- `__IO uint32_t IE`
ADC中断寄存器
- `__IO uint32_t IF`
ADC中断标志位寄存器
- `__IO uint32_t CFG`

- ADC模式配置寄存器
 - `__IO uint32_t TRG`
ADC触发控制寄存器
 - `__IO uint32_t SWT`
ADC软件触发寄存器
 - `__IO uint32_t DAT0_TH`
ADC通道0阈值寄存器

2.2.1 详细描述

ADC寄存器结构体句柄

在文件 [lks32mc08x_adc.h](#) 第 21 行定义.

2.2.2 结构体成员变量说明

2.2.2.1 AMC_A0

```
__IO uint32_t AMC_A0
```

ADC增益校正寄存器0 采样保持电路A对应2/3倍增益通道增益校正

在文件 [lks32mc08x_adc.h](#) 第 64 行定义.

2.2.2.2 AMC_A1

```
__IO uint32_t AMC_A1
```

ADC增益校正寄存器0 采样保持电路A对应 1 倍增益通道增益校正

在文件 [lks32mc08x_adc.h](#) 第 65 行定义.

2.2.2.3 AMC_B0

```
__IO uint32_t AMC_B0
```

ADC增益校正寄存器0 采样保持电路B对应2/3倍增益通道增益校正

在文件 [lks32mc08x_adc.h](#) 第 69 行定义.

2.2.2.4 AMC_B1

```
__IO uint32_t AMC_B1
```

ADC增益校正寄存器0 采样保持电路B对应 1 倍增益通道增益校正

在文件 [lks32mc08x_adc.h](#) 第 70 行定义.

2.2.2.5 CFG

```
--IO uint32_t CFG
```

ADC模式配置寄存器

在文件 [lks32mc08x_adc.h](#) 第 74 行定义.

2.2.2.6 CHN0

```
--IO uint32_t CHN0
```

ADC采样信号来源寄存器0

在文件 [lks32mc08x_adc.h](#) 第 44 行定义.

2.2.2.7 CHN1

```
--IO uint32_t CHN1
```

ADC采样信号来源寄存器1

在文件 [lks32mc08x_adc.h](#) 第 45 行定义.

2.2.2.8 CHN2

```
--IO uint32_t CHN2
```

ADC采样信号来源寄存器2

在文件 [lks32mc08x_adc.h](#) 第 46 行定义.

2.2.2.9 CHN3

```
--IO uint32_t CHN3
```

ADC采样信号来源寄存器3

在文件 [lks32mc08x_adc.h](#) 第 47 行定义.

2.2.2.10 CHN4

```
--IO uint32_t CHN4
```

ADC采样信号来源寄存器4

在文件 [lks32mc08x_adc.h](#) 第 48 行定义.

2.2.2.11 CHN5

```
--IO uint32_t CHN5
```

ADC采样信号来源寄存器5

在文件 [lks32mc08x_adc.h](#) 第 49 行定义.

2.2.2.12 CHN6

```
--IO uint32_t CHN6
```

ADC采样信号来源寄存器6

在文件 [lks32mc08x_adc.h](#) 第 50 行定义.

2.2.2.13 CHN7

```
--IO uint32_t CHN7
```

ADC采样信号来源寄存器7

在文件 [lks32mc08x_adc.h](#) 第 51 行定义.

2.2.2.14 CHN8

```
--IO uint32_t CHN8
```

ADC采样信号来源寄存器8

在文件 [lks32mc08x_adc.h](#) 第 52 行定义.

2.2.2.15 CHN9

```
--IO uint32_t CHN9
```

ADC采样信号来源寄存器9

在文件 [lks32mc08x_adc.h](#) 第 53 行定义.

2.2.2.16 CHNT0

```
--IO uint32_t CHNT0
```

ADC分段通道数寄存器0 设置每段采样数量

在文件 [lks32mc08x_adc.h](#) 第 55 行定义.

2.2.2.17 CHNT1

```
--IO uint32_t CHNT1
```

ADC分段通道数寄存器1 设置每段采样数量

在文件 [lks32mc08x_adc.h](#) 第 56 行定义.

2.2.2.18 DAT0

```
--IO uint32_t DAT0
```

ADC数据寄存器0

在文件 [lks32mc08x_adc.h](#) 第 23 行定义.

2.2.2.19 DAT0.TH

```
--IO uint32_t DAT0.TH
```

ADC通道0阈值寄存器

在文件 [lks32mc08x_adc.h](#) 第 77 行定义.

2.2.2.20 DAT1

```
--IO uint32_t DAT1
```

ADC数据寄存器1

在文件 [lks32mc08x_adc.h](#) 第 24 行定义.

2.2.2.21 DAT10

```
--IO uint32_t DAT10
```

ADC数据寄存器10

在文件 [lks32mc08x_adc.h](#) 第 33 行定义.

2.2.2.22 DAT11

```
--IO uint32_t DAT11
```

ADC数据寄存器11

在文件 [lks32mc08x_adc.h](#) 第 34 行定义.

2.2.2.23 DAT12

```
--IO uint32_t DAT12
```

ADC数据寄存器12

在文件 [lks32mc08x_adc.h](#) 第 35 行定义.

2.2.2.24 DAT13

```
--IO uint32_t DAT13
```

ADC数据寄存器13

在文件 [lks32mc08x_adc.h](#) 第 36 行定义.

2.2.2.25 DAT14

```
--IO uint32_t DAT14
```

ADC数据寄存器14

在文件 [lks32mc08x_adc.h](#) 第 37 行定义.

2.2.2.26 DAT15

```
--IO uint32_t DAT15
```

ADC数据寄存器15

在文件 [lks32mc08x_adc.h](#) 第 38 行定义.

2.2.2.27 DAT16

```
--IO uint32_t DAT16
```

ADC数据寄存器16

在文件 [lks32mc08x_adc.h](#) 第 39 行定义.

2.2.2.28 DAT17

```
--IO uint32_t DAT17
```

ADC数据寄存器17

在文件 [lks32mc08x_adc.h](#) 第 40 行定义.

2.2.2.29 DAT18

```
--IO uint32_t DAT18
```

ADC数据寄存器18

在文件 [lks32mc08x_adc.h](#) 第 41 行定义.

2.2.2.30 DAT19

```
--IO uint32_t DAT19
```

ADC数据寄存器19

在文件 [lks32mc08x_adc.h](#) 第 42 行定义.

2.2.2.31 DAT2

```
--IO uint32_t DAT2
```

ADC数据寄存器2

在文件 [lks32mc08x_adc.h](#) 第 25 行定义.

2.2.2.32 DAT3

```
--IO uint32_t DAT3
```

ADC数据寄存器3

在文件 [lks32mc08x_adc.h](#) 第 26 行定义.

2.2.2.33 DAT4

```
--IO uint32_t DAT4
```

ADC数据寄存器4

在文件 [lks32mc08x_adc.h](#) 第 27 行定义.

2.2.2.34 DAT5

```
--IO uint32_t DAT5
```

ADC数据寄存器5

在文件 [lks32mc08x_adc.h](#) 第 28 行定义.

2.2.2.35 DAT6

```
--IO uint32_t DAT6
```

ADC数据寄存器6

在文件 [lks32mc08x_adc.h](#) 第 29 行定义.

2.2.2.36 DAT7

```
--IO uint32_t DAT7
```

ADC数据寄存器7

在文件 [lks32mc08x_adc.h](#) 第 30 行定义.

2.2.2.37 DAT8

```
--IO uint32_t DAT8
```

ADC数据寄存器8

在文件 [lks32mc08x_adc.h](#) 第 31 行定义.

2.2.2.38 DAT9

```
--IO uint32_t DAT9
```

ADC数据寄存器9

在文件 [lks32mc08x_adc.h](#) 第 32 行定义.

2.2.2.39 DC_A0

```
--IO uint32_t DC_A0
```

ADC直流偏置寄存器0 采样保持电路A对应2/3倍增益通道偏置

在文件 [lks32mc08x_adc.h](#) 第 62 行定义.

2.2.2.40 DC_A1

```
--IO uint32_t DC_A1
```

ADC直流偏置寄存器1 采样保持电路A对应 1 倍增益通道偏置

在文件 [lks32mc08x_adc.h](#) 第 63 行定义.

2.2.2.41 DC_B0

```
--IO uint32_t DC_B0
```

ADC直流偏置寄存器0 采样保持电路B对应2/3倍增益通道偏置

在文件 [lks32mc08x_adc.h](#) 第 67 行定义.

2.2.2.42 DC_B1

```
--IO uint32_t DC_B1
```

ADC直流偏置寄存器1 采样保持电路B对应1倍增益通道偏置

在文件 [lks32mc08x_adc.h](#) 第 68 行定义.

2.2.2.43 GAIN_0

```
--IO uint32_t GAIN_0
```

ADC增益选择寄存器0

在文件 [lks32mc08x_adc.h](#) 第 59 行定义.

2.2.2.44 GAIN_1

```
--IO uint32_t GAIN_1
```

ADC增益选择寄存器1

在文件 [lks32mc08x_adc.h](#) 第 60 行定义.

2.2.2.45 IE

```
--IO uint32_t IE
```

ADC中断寄存器

在文件 [lks32mc08x_adc.h](#) 第 72 行定义.

2.2.2.46 IF

```
--IO uint32_t IF
```

ADC中断标志位寄存器

在文件 [lks32mc08x_adc.h](#) 第 73 行定义.

2.2.2.47 SD0

```
--IO uint32_t SD0
```

保留

在文件 [lks32mc08x_adc.h](#) 第 57 行定义.

2.2.2.48 SD1

```
--IO uint32_t SD1
```

保留

在文件 [lks32mc08x_adc.h](#) 第 58 行定义.

2.2.2.49 SWT

```
--IO uint32_t SWT
```

ADC软件触发寄存器

在文件 [lks32mc08x_adc.h](#) 第 76 行定义.

2.2.2.50 TRG

```
--IO uint32_t TRG
```

ADC触发控制寄存器

在文件 [lks32mc08x_adc.h](#) 第 75 行定义.

2.3 CAN.InitTypeDef结构体 参考

CAN结构体基地址定义

```
#include <lks32mc08x_can.h>
```

成员变量

- uint32_t [CAN.WorkMode](#)
CAN工作模式选择 1: 自测模式 2: 监听模式 3:正常收发模式
- uint32_t [CAN.DMAEn](#)
使能CAN的DMA功能
- uint32_t [CAN.ErrWarThre](#)
错误与警告门限值
- uint32_t [CAN.Btr0](#)
波特率 0
- uint32_t [CAN.Btr1](#)
波特率 1
- uint32_t [IRQEna](#)
中断使能

2.3.1 详细描述

CAN结构体基地址定义

CAN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 [lks32mc08x.can.h](#) 第 74 行定义.

2.3.2 结构体成员变量说明

2.3.2.1 CAN_Btr0

```
uint32_t CAN_Btr0
```

波特率 0

在文件 [lks32mc08x.can.h](#) 第 79 行定义.

2.3.2.2 CAN_Btr1

```
uint32_t CAN_Btr1
```

波特率 1

在文件 [lks32mc08x.can.h](#) 第 80 行定义.

2.3.2.3 CAN_DMAEn

```
uint32_t CAN_DMAEn
```

使能CAN的DMA功能

在文件 [lks32mc08x.can.h](#) 第 77 行定义.

2.3.2.4 CAN_ErrWarThre

```
uint32_t CAN_ErrWarThre
```

错误与警告门限值

在文件 [lks32mc08x.can.h](#) 第 78 行定义.

2.3.2.5 CAN_WorkMode

```
uint32_t CAN_WorkMode
```

CAN工作模式选择 1: 自测模式 2: 监听模式 3:正常收发模式

在文件 [lks32mc08x.can.h](#) 第 76 行定义.

2.3.2.6 IRQEna

uint32_t IRQEna

中断使能

在文件 [lks32mc08x_can.h](#) 第 81 行定义.

2.4 CAN_TypeDef结构体 参考

CAN寄存器结构体句柄

```
#include <lks32mc08x_can.h>
```

成员变量

- __IO uint32_t [MOD](#)
CAN 模式寄存器
- __IO uint32_t [CMR](#)
CAN 命令寄存器
- __IO uint32_t [SR](#)
CAN 状态寄存器
- __IO uint32_t [IR](#)
CAN 中断状态寄存器
- __IO uint32_t [EIR](#)
CAN 中断控制寄存器
- __IO uint32_t [null0](#)
- __IO uint32_t [BTR0](#)
CAN 总线时序控制寄存器 0
- __IO uint32_t [BTR1](#)
CAN 总线时序控制寄存器 1
- __IO uint32_t [null1](#)
- __IO uint32_t [null2](#)
- __IO uint32_t [null3](#)
- __IO uint32_t [ALC](#)
CAN 仲裁丢失捕捉寄存器
- __IO uint32_t [ECC](#)
CAN 错误码捕捉寄存器
- __IO uint32_t [EWLR](#)
CAN 错误&警告门限值设置寄存器
- __IO uint32_t [RXERR](#)
CAN 接收错误计数器
- __IO uint32_t [TXERR](#)
CAN 发送错误计数器
- __IO uint32_t [TXRX0](#)
正常工作模式下, CAN 发送帧格式寄存器0/CAN 接收帧格式寄存器0
- __IO uint32_t [TXRX1](#)
正常工作模式下, CAN 发送数据寄存器0/CAN 接收数据寄存器1
- __IO uint32_t [TXRX2](#)

- 正常工作模式下, CAN 发送数据寄存器2/CAN 接收数据寄存器2
- `__IO uint32_t TXRX3`
- 正常工作模式下, CAN 发送数据寄存器3/CAN 接收数据寄存器3
- `__IO uint32_t TXRX4`
- 正常工作模式下, CAN 发送数据寄存器4/CAN 接收数据寄存器4
- `__IO uint32_t TXRX5`
- 正常工作模式下, CAN 发送数据寄存器5/CAN 接收数据寄存器5
- `__IO uint32_t TXRX6`
- 正常工作模式下, CAN 发送数据寄存器6/CAN 接收数据寄存器6
- `__IO uint32_t TXRX7`
- 正常工作模式下, CAN 发送数据寄存器7/CAN 接收数据寄存器7
- `__IO uint32_t TXRX8`
- 正常工作模式下, CAN 发送数据寄存器8/CAN 接收数据寄存器8
- `__IO uint32_t TXRX9`
- 正常工作模式下, CAN 发送数据寄存器9/CAN 接收数据寄存器9
- `__IO uint32_t TXRXA`
- 正常工作模式下, CAN 发送数据寄存器10/CAN 接收数据寄存器10
- `__IO uint32_t TXRXB`
- 正常工作模式下, CAN 发送数据寄存器11/CAN 接收数据寄存器11
- `__IO uint32_t TXRXC`
- 正常工作模式下, CAN 发送数据寄存器 12/CAN 接收数据寄存器 12
- `__IO uint32_t RMC`
- CAN FIFO 有效接收信息计数器
- `__IO uint32_t RBSA`
- CAN 第一条有效接收信息在 FIFO 中的地址寄存器

2.4.1 详细描述

CAN寄存器结构体句柄

在文件 [lks32mc08x_can.h](#) 第 22 行定义.

2.4.2 结构体成员变量说明

2.4.2.1 ALC

```
__IO uint32_t ALC
```

CAN 仲裁丢失捕捉寄存器

在文件 [lks32mc08x_can.h](#) 第 37 行定义.

2.4.2.2 BTR0

```
__IO uint32_t BTR0
```

CAN 总线时序控制寄存器 0

在文件 [lks32mc08x_can.h](#) 第 31 行定义.

2.4.2.3 BTR1

```
--IO uint32_t BTR1
```

CAN 总线时序控制寄存器 1

在文件 [lks32mc08x.can.h](#) 第 32 行定义.

2.4.2.4 CMR

```
--IO uint32_t CMR
```

CAN 命令寄存器

在文件 [lks32mc08x.can.h](#) 第 25 行定义.

2.4.2.5 ECC

```
--IO uint32_t ECC
```

CAN 错误码捕捉寄存器

在文件 [lks32mc08x.can.h](#) 第 39 行定义.

2.4.2.6 EIR

```
--IO uint32_t EIR
```

CAN 中断控制寄存器

在文件 [lks32mc08x.can.h](#) 第 29 行定义.

2.4.2.7 EWLR

```
--IO uint32_t EWLR
```

CAN 错误&警告门限值设置寄存器

在文件 [lks32mc08x.can.h](#) 第 40 行定义.

2.4.2.8 IR

```
--IO uint32_t IR
```

CAN 中断状态寄存器

在文件 [lks32mc08x.can.h](#) 第 27 行定义.

2.4.2.9 MOD

```
--IO uint32_t MOD
```

CAN 模式寄存器

在文件 [lks32mc08x.can.h](#) 第 24 行定义.

2.4.2.10 null0

```
--IO uint32_t null0
```

在文件 [lks32mc08x.can.h](#) 第 30 行定义.

2.4.2.11 null1

```
--IO uint32_t null1
```

在文件 [lks32mc08x.can.h](#) 第 34 行定义.

2.4.2.12 null2

```
--IO uint32_t null2
```

在文件 [lks32mc08x.can.h](#) 第 35 行定义.

2.4.2.13 null3

```
--IO uint32_t null3
```

在文件 [lks32mc08x.can.h](#) 第 36 行定义.

2.4.2.14 RBSA

```
--IO uint32_t RBSA
```

CAN 第一条有效接收信息在 FIFO 中的地址寄存器

在文件 [lks32mc08x.can.h](#) 第 61 行定义.

2.4.2.15 RMC

```
--IO uint32_t RMC
```

CAN FIFO 有效接收信息计数器

在文件 [lks32mc08x.can.h](#) 第 60 行定义.

2.4.2.16 RXERR

```
--IO uint32_t RXERR
```

CAN 接收错误计数器

在文件 [lks32mc08x.can.h](#) 第 41 行定义.

2.4.2.17 SR

```
--IO uint32_t SR
```

CAN 状态寄存器

在文件 [lks32mc08x.can.h](#) 第 26 行定义.

2.4.2.18 TXERR

```
--IO uint32_t TXERR
```

CAN 发送错误计数器

在文件 [lks32mc08x.can.h](#) 第 42 行定义.

2.4.2.19 TXRX0

```
--IO uint32_t TXRX0
```

正常工作模式下, CAN 发送帧格式寄存器0/CAN 接收帧格式寄存器0

在文件 [lks32mc08x.can.h](#) 第 44 行定义.

2.4.2.20 TXRX1

```
--IO uint32_t TXRX1
```

正常工作模式下, CAN 发送数据寄存器0/CAN 接收数据寄存器1

在文件 [lks32mc08x.can.h](#) 第 45 行定义.

2.4.2.21 TXRX2

```
--IO uint32_t TXRX2
```

正常工作模式下, CAN 发送数据寄存器2/CAN 接收数据寄存器2

在文件 [lks32mc08x.can.h](#) 第 46 行定义.

2.4.2.22 TXRX3

```
--IO uint32_t TXRX3
```

正常工作模式下, CAN 发送数据寄存器3/CAN 接收数据寄存器3

在文件 [lks32mc08x.can.h](#) 第 47 行定义.

2.4.2.23 TXRX4

```
--IO uint32_t TXRX4
```

正常工作模式下, CAN 发送数据寄存器4/CAN 接收数据寄存器4

在文件 [lks32mc08x.can.h](#) 第 49 行定义.

2.4.2.24 TXRX5

```
--IO uint32_t TXRX5
```

正常工作模式下, CAN 发送数据寄存器5/CAN 接收数据寄存器5

在文件 [lks32mc08x.can.h](#) 第 50 行定义.

2.4.2.25 TXRX6

```
--IO uint32_t TXRX6
```

正常工作模式下, CAN 发送数据寄存器6/CAN 接收数据寄存器6

在文件 [lks32mc08x.can.h](#) 第 51 行定义.

2.4.2.26 TXRX7

```
--IO uint32_t TXRX7
```

正常工作模式下, CAN 发送数据寄存器7/CAN 接收数据寄存器7

在文件 [lks32mc08x.can.h](#) 第 52 行定义.

2.4.2.27 TXRX8

```
--IO uint32_t TXRX8
```

正常工作模式下, CAN 发送数据寄存器8/CAN 接收数据寄存器8

在文件 [lks32mc08x.can.h](#) 第 54 行定义.

2.4.2.28 TXRX9

```
--IO uint32_t TXRX9
```

正常工作模式下，CAN 发送数据寄存器9/CAN 接收数据寄存器9

在文件 [lks32mc08x_can.h](#) 第 55 行定义.

2.4.2.29 TXRXA

```
--IO uint32_t TXRXA
```

正常工作模式下，CAN 发送数据寄存器10/CAN 接收数据寄存器10

在文件 [lks32mc08x_can.h](#) 第 56 行定义.

2.4.2.30 TXRXB

```
--IO uint32_t TXRXB
```

正常工作模式下，CAN 发送数据寄存器11/CAN 接收数据寄存器11

在文件 [lks32mc08x_can.h](#) 第 57 行定义.

2.4.2.31 TXRXC

```
--IO uint32_t TXRXC
```

正常工作模式下，CAN 发送数据寄存器 12/CAN 接收数据寄存器 12

在文件 [lks32mc08x_can.h](#) 第 59 行定义.

2.5 CMP_InitTypeDef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 <lks32mc08x_cmp.h>
```

成员变量

- u32 **FIL_CLK10_DIV16**
CMP 滤波时钟频率 $\text{Freq}(\text{CMP\_Filter}) = (\text{Freq}(\text{MCLK})/2^{\text{FIL\_CLK10\_DIV2}})/(\text{FIL\_CLK10\_DIV16}+1)$ ，其中 MCLK 为系统的主时钟，通常为 96MHz 全速时钟。需要注意的是，产生 CMP 滤波时钟需要使能 CLK10.EN。
- FuncState **CLK10_EN**
比较器 1/0 滤波时钟使能
DISABLE:失能，ENABLE: 使能
- u32 **FIL_CLK10_DIV2**
CMP 滤波时钟频率 $\text{Freq}(\text{CMP\_Filter}) = (\text{Freq}(\text{MCLK})/2^{\text{FIL\_CLK10\_DIV2}})/(\text{FIL\_CLK10\_DIV16}+1)$ ，其中 MCLK 为系统的主时钟，通常为 96MHz 全速时钟。需要注意的是，产生 CMP 滤波时钟需要使能 CLK10.EN。
- FuncState **CMP_FT**
比较器快速比较使能
DISABLE:失能，ENABLE: 使能
- u32 **CMP_HYS**
比较器回差选择
- FuncState **CMP0_EN**
比较器 0 使能
DISABLE:失能，ENABLE: 使能
- u32 **CMP0_SELP**
比较器 0 信号正端选择
- u32 **CMP0_SELN**
比较器 0 信号负端选择
- FuncState **CMP0_IE**
比较器 0 中断使能
DISABLE:失能，ENABLE: 使能
- u32 **CMP0_W_PWM_POL**
比较器 0 开窗 PWM 信号极性选择
0:高电平有效；1:低电平有效
- FuncState **CMP0_IRQ_TRIG**
比较器 0 边沿触发使能
DISABLE:失能，ENABLE: 使能
- FuncState **CMP0_IN_EN**
比较器 0 信号输入使能
DISABLE:失能，ENABLE: 使能
- u32 **CMP0_POL**
比较器 0 极性选择
0:高电平有效；1:低电平有效
- FuncState **CMP0_CHN3P_WIN_EN**
MCPWM 模块 CHN3.P 通道使能比较器 0 开窗 DISABLE:失能，ENABLE: 使能
- FuncState **CMP0_CHN2P_WIN_EN**
MCPWM 模块 CHN2.P 通道使能比较器 0 开窗 DISABLE:失能，ENABLE: 使能
- FuncState **CMP0_CHN1P_WIN_EN**
MCPWM 模块 CHN1.P 通道使能比较器 0 开窗 DISABLE:失能，ENABLE: 使能
- FuncState **CMP0_CHN0P_WIN_EN**
MCPWM 模块 CHN0.P 通道使能比较器 0 开窗 DISABLE:失能，ENABLE: 使能
- FuncState **CMP1_EN**
比较器 1 使能
使能 DISABLE:失能，ENABLE: 使能

- u32 **CMP1_SEL_P**
比较器 0 信号正端选择
- u32 **CMP1_SEL_N**
比较器 0 信号负端选择
- FuncState **CMP1_IE**
比较器 1 中断使能
DISABLE:失能, *ENABLE*: 使能
- u32 **CMP1_W_PWM_POL**
比较器 1 开窗 *PWM* 信号极性选择
0:高电平有效; 1:低电平有效
- FuncState **CMP1_IRQ_TRIG**
比较器 1 边沿触发使能
DISABLE:失能, *ENABLE*: 使能
- FuncState **CMP1_IN_EN**
比较器 1 信号输入使能
DISABLE:失能, *ENABLE*: 使能
- u32 **CMP1_POL**
比较器 1 极性选择 0:高电平有效; 1:低电平有效
- FuncState **CMP1_CHN3P_WIN_EN**
MCPWM 模块 *CHN3_P* 通道使能比较器 1 开窗
0:高电平有效; 1:低电平有效
- FuncState **CMP1_CHN2P_WIN_EN**
MCPWM 模块 *CHN2_P* 通道使能比较器 1 开窗
0:高电平有效; 1:低电平有效
- FuncState **CMP1_CHN1P_WIN_EN**
MCPWM 模块 *CHN1_P* 通道使能比较器 1 开窗
0:高电平有效; 1:低电平有效
- FuncState **CMP1_CHN0P_WIN_EN**
MCPWM 模块 *CHN0_P* 通道使能比较器 1 开窗
0:高电平有效; 1:低电平有效

2.5.1 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 [lks32mc08x_cmp.h](#) 第 102 行定义.

2.5.2 结构体成员变量说明

2.5.2.1 CLK10.EN

FuncState CLK10.EN

比较器 1/0 滤波时钟使能
DISABLE:失能, *ENABLE*: 使能

在文件 [lks32mc08x_cmp.h](#) 第 109 行定义.

2.5.2.2 CMP0_CHN0P_WIN_EN

FuncState CMP0_CHN0P_WIN_EN

MCPWM 模块 CHN0_P 通道使能比较器
0 开窗 DISABLE:失能, ENABLE: 使能

在文件 [lks32mc08x_cmp.h](#) 第 146 行定义.

2.5.2.3 CMP0_CHN1P_WIN_EN

FuncState CMP0_CHN1P_WIN_EN

MCPWM 模块 CHN1_P 通道使能比较器
0 开窗 DISABLE:失能, ENABLE: 使能

在文件 [lks32mc08x_cmp.h](#) 第 145 行定义.

2.5.2.4 CMP0_CHN2P_WIN_EN

FuncState CMP0_CHN2P_WIN_EN

MCPWM 模块 CHN2_P 通道使能比较器
0 开窗 DISABLE:失能, ENABLE: 使能

在文件 [lks32mc08x_cmp.h](#) 第 144 行定义.

2.5.2.5 CMP0_CHN3P_WIN_EN

FuncState CMP0_CHN3P_WIN_EN

MCPWM 模块 CHN3_P 通道使能比较器
0 开窗 DISABLE:失能, ENABLE: 使能

在文件 [lks32mc08x_cmp.h](#) 第 143 行定义.

2.5.2.6 CMP0_EN

FuncState CMP0_EN

比较器 0 使能
DISABLE:失能, ENABLE: 使能

在文件 [lks32mc08x_cmp.h](#) 第 118 行定义.

2.5.2.7 CMP0_IE

FuncState CMP0_IE

比较器 0 中断使能
DISABLE:失能, ENABLE: 使能

在文件 [lks32mc08x_cmp.h](#) 第 138 行定义.

2.5.2.8 CMP0_IN_EN

FuncState CMP0_IN_EN

比较器 0 信号输入使能

DISABLE:失能, ENABLE: 使能

在文件 [lks32mc08x_cmp.h](#) 第 141 行定义.

2.5.2.9 CMP0_IRQ_TRIG

FuncState CMP0_IRQ_TRIG

比较器 0 边沿触发使能

DISABLE:失能, ENABLE: 使能

在文件 [lks32mc08x_cmp.h](#) 第 140 行定义.

2.5.2.10 CMP0_POL

u32 CMP0_POL

比较器 0 极性选择

0:高电平有效; 1:低电平有效

在文件 [lks32mc08x_cmp.h](#) 第 142 行定义.

2.5.2.11 CMP0_SELN

u32 CMP0_SELN

比较器 0 信号负端选择

参见

[CMP_SELN_IN](#)

[CMP_SELN_REF](#)

[CMP_SELN_DAC](#)

[CMP_SELN_HALL_MID](#)

在文件 [lks32mc08x_cmp.h](#) 第 137 行定义.

2.5.2.12 CMP0_SELP

u32 CMP0_SELP

比较器 0 信号正端选择

参见

CMP_SELP_IP0
CMP_SELP_OPA_IP
CMP_SELP_OPA_OUT
CMP_SELP_IP1
CMP_SELP_IP2
CMP_SELP_IP3
CMP_SELP_IP4

在文件 [lks32mc08x_cmp.h](#) 第 129 行定义.

2.5.2.13 CMP0_W_PWM_POL

u32 CMP0_W_PWM_POL

比较器 0 开窗 PWM 信号极性选择
0:高电平有效; 1:低电平有效

在文件 [lks32mc08x_cmp.h](#) 第 139 行定义.

2.5.2.14 CMP1_CHN0P_WIN_EN

FuncState CMP1_CHN0P_WIN_EN

MCPWM 模块 CHN0_P 通道使能比较器 1 开窗
0:高电平有效; 1:低电平有效

在文件 [lks32mc08x_cmp.h](#) 第 176 行定义.

2.5.2.15 CMP1_CHN1P_WIN_EN

FuncState CMP1_CHN1P_WIN_EN

MCPWM 模块 CHN1_P 通道使能比较器 1 开窗
0:高电平有效; 1:低电平有效

在文件 [lks32mc08x_cmp.h](#) 第 175 行定义.

2.5.2.16 CMP1_CHN2P_WIN_EN

FuncState CMP1_CHN2P_WIN_EN

MCPWM 模块 CHN2_P 通道使能比较器 1 开窗
0:高电平有效; 1:低电平有效

在文件 [lks32mc08x_cmp.h](#) 第 174 行定义.

2.5.2.17 CMP1_CHN3P_WIN_EN

FuncState CMP1_CHN3P_WIN_EN

MCPWM 模块 CHN3_P 通道使能比较器 1 开窗
0:高电平有效; 1:低电平有效

在文件 [lks32mc08x_cmp.h](#) 第 173 行定义.

2.5.2.18 CMP1_EN

FuncState CMP1_EN

比较器 1 使能
使能 DISABLE:失能, ENABLE: 使能

在文件 [lks32mc08x_cmp.h](#) 第 148 行定义.

2.5.2.19 CMP1_IE

FuncState CMP1_IE

比较器 1 中断使能
DISABLE:失能, ENABLE: 使能

在文件 [lks32mc08x_cmp.h](#) 第 168 行定义.

2.5.2.20 CMP1_IN_EN

FuncState CMP1_IN_EN

比较器 1 信号输入使能
DISABLE:失能, ENABLE: 使能

在文件 [lks32mc08x_cmp.h](#) 第 171 行定义.

2.5.2.21 CMP1_IRQ_TRIG

FuncState CMP1_IRQ_TRIG

比较器 1 边沿触发使能
DISABLE:失能, ENABLE: 使能

在文件 [lks32mc08x_cmp.h](#) 第 170 行定义.

2.5.2.22 CMP1_POL

u32 CMP1_POL

比较器 1 极性选择 0:高电平有效; 1:低电平有效

在文件 [lks32mc08x_cmp.h](#) 第 172 行定义.

2.5.2.23 CMP1_SELN

u32 CMP1_SELN

比较器 0 信号负端选择

参见

[CMP_SELN_IN](#)

[CMP_SELN_REF](#)

[CMP_SELN_DAC](#)

[CMP_SELN_HALL_MID](#)

在文件 [lks32mc08x_cmp.h](#) 第 167 行定义.

2.5.2.24 CMP1_SELP

u32 CMP1_SELP

比较器 0 信号正端选择

参见

[CMP_SELP_IP0](#)

[CMP_SELP_OPA_IP](#)

[CMP_SELP_OPA_OUT](#)

[CMP_SELP_IP1](#)

[CMP_SELP_IP2](#)

[CMP_SELP_IP3](#)

[CMP_SELP_IP4](#)

在文件 [lks32mc08x_cmp.h](#) 第 159 行定义.

2.5.2.25 CMP1_W_PWM_POL

u32 CMP1_W_PWM_POL

比较器 1 开窗 PWM 信号极性选择
0:高电平有效; 1:低电平有效

在文件 [lks32mc08x_cmp.h](#) 第 169 行定义.

2.5.2.26 CMP_FT

FuncState CMP_FT

比较器快速比较使能

DISABLE:失能, ENABLE: 使能

在文件 [lks32mc08x_cmp.h](#) 第 115 行定义.

2.5.2.27 CMP_HYS

u32 CMP_HYS

比较器回差选择

参见

[CMP_HYS_0mV](#)

[CMP_HYS_20mV](#)

在文件 [lks32mc08x_cmp.h](#) 第 116 行定义.

2.5.2.28 FIL_CLK10_DIV16

u32 FIL_CLK10_DIV16

CMP 滤波时钟频率 $\text{Freq}(\text{CMP\_Filter}) = (\text{Freq}(\text{MCLK})/2^{\text{FIL\_CLK10\_DIV2}})/(\text{FIL\_CLK10\_DIV16}+1)$, 其中MCLK为系统的主时钟, 通常为 96MHz 全速时钟。需要注意的是, 产生 CMP 滤波时钟需要使能CLK10_EN。

比较器 1/0 滤波时钟分频

赋值范围0-0xF

在文件 [lks32mc08x_cmp.h](#) 第 108 行定义.

2.5.2.29 FIL_CLK10_DIV2

u32 FIL_CLK10_DIV2

CMP 滤波时钟频率 $\text{Freq}(\text{CMP\_Filter}) = (\text{Freq}(\text{MCLK})/2^{\text{FIL\_CLK10\_DIV2}})/(\text{FIL\_CLK10\_DIV16}+1)$, 其中MCLK为系统的主时钟, 通常为 96MHz 全速时钟。需要注意的是, 产生 CMP 滤波时钟需要使能CLK10_EN。

比较器 1/0 滤波时钟分频

0x0:1分频, 0x1:2分频,0x2:4分频,0x3:8分频

在文件 [lks32mc08x_cmp.h](#) 第 114 行定义.

2.6 CMP_TypeDef结构体 参考

CMP寄存器结构体句柄

```
#include <lks32mc08x_cmp.h>
```

成员变量

- `__IO uint32_t IE`
比较器中断使能寄存器
- `__IO uint32_t IF`
比较器中断标志寄存器
- `__IO uint32_t TCLK`
比较器分频时钟控制寄存器
- `__IO uint32_t CFG`
比较器控制寄存器
- `__IO uint32_t BLCWIN`
比较器开窗控制寄存器
- `__IO uint32_t DATA`
比较器输出数值寄存器

2.6.1 详细描述

CMP寄存器结构体句柄

在文件 [lks32mc08x_cmp.h](#) 第 75 行定义.

2.6.2 结构体成员变量说明

2.6.2.1 BLCWIN

```
__IO uint32_t BLCWIN
```

比较器开窗控制寄存器

在文件 [lks32mc08x_cmp.h](#) 第 81 行定义.

2.6.2.2 CFG

```
__IO uint32_t CFG
```

比较器控制寄存器

在文件 [lks32mc08x_cmp.h](#) 第 80 行定义.

2.6.2.3 DATA

```
--IO uint32_t DATA
```

比较器输出数值寄存器

在文件 [lks32mc08x_cmp.h](#) 第 82 行定义.

2.6.2.4 IE

```
--IO uint32_t IE
```

比较器中断使能寄存器

在文件 [lks32mc08x_cmp.h](#) 第 77 行定义.

2.6.2.5 IF

```
--IO uint32_t IF
```

比较器中断标志寄存器

在文件 [lks32mc08x_cmp.h](#) 第 78 行定义.

2.6.2.6 TCLK

```
--IO uint32_t TCLK
```

比较器分频时钟控制寄存器

在文件 [lks32mc08x_cmp.h](#) 第 79 行定义.

2.7 DAC_InitTypeDef结构体 参考

```
#include <lks32mc08x_dac.h>
```

成员变量

- `uint8_t DAC_GAIN`
DAC输出档位: 0, 3V; 1, 1.2V; 2, 4.85V
- `uint8_t DACOUT_EN`
DAC输出使能: *DISABLE*, 不使能; *ENABLE*, 使能。

2.7.1 详细描述

在文件 [lks32mc08x_dac.h](#) 第 19 行定义.

2.7.2 结构体成员变量说明

2.7.2.1 DAC_GAIN

uint8_t DAC_GAIN

DAC输出档位: 0, 3V; 1, 1.2V; 2, 4.85V

参见

[DAC_RANGE_3V0](#)

[DAC_RANGE_1V2](#)

[DAC_RANGE_4V85](#)

在文件 [lks32mc08x-dac.h](#) 第 27 行定义.

2.7.2.2 DACOUT_EN

uint8_t DACOUT_EN

DAC输出使能: DISABLE, 不使能; ENABLE, 使能。

在文件 [lks32mc08x-dac.h](#) 第 28 行定义.

2.8 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 <lks32mc08x-dma.h>
```

成员变量

- uint8_t [DMA_IRQ_EN](#)
DMA 中断使能
- uint8_t [DMA_DIR](#)
DMA传输方向
- uint8_t [DMA_CIRC](#)
DMA传输模式: 循环模式, *ENABLE*:使能, *DISABLE*:失能
- uint8_t [DMA_PINC](#)
外设地址每轮内是否递增,*ENABLE*:使能, *DISABLE*:失能
- uint8_t [DMA_MINC](#)
内存地址第二轮是否在第一轮地址的基础上递增, *ENABLE*:使能, *DISABLE*:失能
- uint8_t [DMA_PBTW](#)
外设访问位宽
- uint8_t [DMA_MBTW](#)
内存访问位宽
- uint8_t [DMA_REQ_EN](#)
通道 x 三个硬件 DMA 请求使能
- uint16_t [DMA_TIMES](#)
DMA 通道 x 每轮数据搬运次数 1~511
- uint8_t [DMA_ROUND](#)
DMA 通道 x 采样轮数 1~255
- uint32_t [DMA_CPAR](#)
DMA 通道 x 外设地址
- uint32_t [DMA_CMAR](#)
DMA 通道 x 内存地址

2.8.1 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 [lks32mc08x_dma.h](#) 第 22 行定义.

2.8.2 结构体成员变量说明

2.8.2.1 DMA_CIRC

```
uint8_t DMA_CIRC
```

DMA传输模式：循环模式， ENABLE:使能， DISABLE:失能

在文件 [lks32mc08x_dma.h](#) 第 36 行定义.

2.8.2.2 DMA_CMAR

```
uint32_t DMA_CMAR
```

DMA 通道 x 内存地址

在文件 [lks32mc08x_dma.h](#) 第 72 行定义.

2.8.2.3 DMA_CPAR

```
uint32_t DMA_CPAR
```

DMA 通道 x 外设地址

在文件 [lks32mc08x_dma.h](#) 第 71 行定义.

2.8.2.4 DMA_DIR

```
uint8_t DMA_DIR
```

DMA传输方向

参见

[PERI2MEMORY](#) 外设至内存搬运

[MEMORY2PERI](#) 内存至外设搬运

在文件 [lks32mc08x_dma.h](#) 第 35 行定义.

2.8.2.5 DMA_IRQ_EN

uint8_t DMA_IRQ_EN

DMA 中断使能

参见

[DMA_TCIE](#)

[DMA_TEIE](#) DMA 中断使能

在文件 [lks32mc08x_dma.h](#) 第 29 行定义.

2.8.2.6 DMA_MBTW

uint8_t DMA_MBTW

内存访问位宽

参见

[DMA_BYTE_TRANS](#)

[DMA_HALFWORD_TRANS](#)

[DMA_WORD_TRANS](#)

在文件 [lks32mc08x_dma.h](#) 第 52 行定义.

2.8.2.7 DMA_MINC

uint8_t DMA_MINC

内存地址第二轮是否在第一轮地址的基础上递增, ENABLE:使能, DISABLE:失能

在文件 [lks32mc08x_dma.h](#) 第 38 行定义.

2.8.2.8 DMA_PBTW

uint8_t DMA_PBTW

外设访问位宽

参见

[DMA_BYTE_TRANS](#)

[DMA_HALFWORD_TRANS](#)

[DMA_WORD_TRANS](#)

在文件 [lks32mc08x_dma.h](#) 第 45 行定义.

2.8.2.9 DMA_PINC

uint8_t DMA_PINC

外设地址每轮内是否递增,ENABLE:使能, DISABLE:失能

在文件 [lks32mc08x_dma.h](#) 第 37 行定义.

2.8.2.10 DMA_REQ_EN

uint8_t DMA_REQ_EN

通道 x 三个硬件 DMA 请求使能

参见

[DMA_CH0_ADC0_REQ_EN](#) (DMA通道0使用)
[DMA_CH0_TIMER0_REQ_EN](#) (DMA通道0使用)
[DMA_CH0_TIMER1_REQ_EN](#) (DMA通道0使用)
[DMA_CH1_SPI_RX_REQ_EN](#) (DMA通道1使用)
[DMA_CH1_MCPWM_REQ_EN](#) (DMA通道1使用)
[DMA_CH1_TIMER2_REQ_EN](#) (DMA通道1使用)
[DMA_CH2_UART0_REQ_EN](#) (DMA通道2使用)
[DMA_CH2_SPI_TX_REQ_EN](#) (DMA通道2使用)
[DMA_CH2_TIMER3_REQ_EN](#) (DMA通道2使用)
[DMA_CH3_UART1_REQ_EN](#) (DMA通道3使用)
[DMA_CH3_CAN_REQ_EN](#) (DMA通道3使用)
[DMA_CH3_I2C_REQ_EN](#) (DMA通道3使用)

在文件 [lks32mc08x_dma.h](#) 第 68 行定义.

2.8.2.11 DMA_ROUND

uint8_t DMA_ROUND

DMA 通道 x 采样轮数 1~255

在文件 [lks32mc08x_dma.h](#) 第 70 行定义.

2.8.2.12 DMA_TIMES

uint16_t DMA_TIMES

DMA 通道 x 每轮数据搬运次数 1~511

在文件 [lks32mc08x_dma.h](#) 第 69 行定义.

2.9 DMA_RegTypeDef结构体 参考

DAM通道0结构体基地址定义

```
#include <lks32mc08x_dma.h>
```

成员变量

- __IO uint32_t [DMA_CCR](#)
DMA 通道配置寄存器
- __IO uint32_t [DMA_CTMS](#)
DMA 控制寄存器
- __IO uint32_t [DMA_CPAR](#)
DMA 外设地址寄存器
- __IO uint32_t [DMA_CMAR](#)
DMA 内存地址寄存器

2.9.1 详细描述

DAM通道0结构体基地址定义

DAM通道1结构体基地址定义 DAM通道2结构体基地址定义 DAM通道3结构体基地址定义

DMA寄存器结构体句柄

在文件 [lks32mc08x_dma.h](#) 第 104 行定义.

2.9.2 结构体成员变量说明

2.9.2.1 DMA_CCR

```
__IO uint32_t DMA_CCR
```

DMA 通道配置寄存器

在文件 [lks32mc08x_dma.h](#) 第 106 行定义.

2.9.2.2 DMA_CMAR

```
__IO uint32_t DMA_CMAR
```

DMA 内存地址寄存器

在文件 [lks32mc08x_dma.h](#) 第 109 行定义.

2.9.2.3 DMA_CPAR

```
--IO uint32_t DMA_CPAR
```

DMA 外设地址寄存器

在文件 [lks32mc08x_dma.h](#) 第 108 行定义.

2.9.2.4 DMA_CTMS

```
--IO uint32_t DMA_CTMS
```

DMA 控制寄存器

在文件 [lks32mc08x_dma.h](#) 第 107 行定义.

2.10 EXTI_TypeDef结构体 参考

EXTI寄存器结构体句柄

```
#include <lks32mc08x_exti.h>
```

成员变量

- `--IO uint32_t CR0`
- `--IO uint32_t CR1`
- `--IO uint32_t IF`
- `--IO uint32_t POL`
- `--IO uint32_t EN`

2.10.1 详细描述

EXTI寄存器结构体句柄

在文件 [lks32mc08x_exti.h](#) 第 26 行定义.

2.10.2 结构体成员变量说明

2.10.2.1 CR0

```
--IO uint32_t CR0
```

在文件 [lks32mc08x_exti.h](#) 第 28 行定义.

2.10.2.2 CR1

```
--IO uint32_t CR1
```

在文件 [lks32mc08x_exti.h](#) 第 29 行定义.

2.10.2.3 EN

```
--IO uint32_t EN
```

在文件 [lks32mc08x_exti.h](#) 第 32 行定义.

2.10.2.4 IF

```
--IO uint32_t IF
```

在文件 [lks32mc08x_exti.h](#) 第 30 行定义.

2.10.2.5 POL

```
--IO uint32_t POL
```

在文件 [lks32mc08x_exti.h](#) 第 31 行定义.

2.11 GPIO_InitTypeDef结构体 参考

GPIO功能配置结构体句柄

```
#include <lks32mc08x_gpio.h>
```

成员变量

- [uint32_t GPIO_Pin](#)
- [GPIO_Mode_TypeDef GPIO_Mode](#)
- [GPIO_PuPd_TypeDef GPIO_PuPd](#)
- [uint32_t GPIO_PODEna](#)

2.11.1 详细描述

GPIO功能配置结构体句柄

在文件 [lks32mc08x_gpio.h](#) 第 97 行定义.

2.11.2 结构体成员变量说明

2.11.2.1 GPIO_Mode

`GPIO_Mode_TypeDef` `GPIO_Mode`

GPIO模式:输入、输出、模拟

在文件 `lks32mc08x-gpio.h` 第 100 行定义.

2.11.2.2 GPIO_Pin

`uint32_t` `GPIO_Pin`

要配置的Pad

在文件 `lks32mc08x-gpio.h` 第 99 行定义.

2.11.2.3 GPIO_PODena

`uint32_t` `GPIO_PODena`

开漏

在文件 `lks32mc08x-gpio.h` 第 102 行定义.

2.11.2.4 GPIO_PuPd

`GPIO_PuPd_TypeDef` `GPIO_PuPd`

默认/上拉

在文件 `lks32mc08x-gpio.h` 第 101 行定义.

2.12 GPIO_TypeDef结构体 参考

GPIO寄存器结构体句柄

```
#include <lks32mc08x-gpio.h>
```

成员变量

- `__IO uint32_t` `PIE`
- `__IO uint32_t` `POE`
- `__IO uint32_t` `PDI`
- `__IO uint32_t` `PDO`
- `__IO uint32_t` `PUE`
- `__IO uint32_t` `PDE`
- `__IO uint32_t` `PODE`
- `__IO uint32_t` `LCKR`
- `__IO uint32_t` `F3210`
- `__IO uint32_t` `F7654`
- `__IO uint32_t` `FBA98`
- `__IO uint32_t` `FFEDC`

2.12.1 详细描述

GPIO寄存器结构体句柄

在文件 [lks32mc08x_gpio.h](#) 第 24 行定义.

2.12.2 结构体成员变量说明

2.12.2.1 F3210

```
--IO uint32_t F3210
```

功能选择 GPIO.Pin_0-3

在文件 [lks32mc08x_gpio.h](#) 第 34 行定义.

2.12.2.2 F7654

```
--IO uint32_t F7654
```

功能选择 GPIO.Pin_4-7

在文件 [lks32mc08x_gpio.h](#) 第 35 行定义.

2.12.2.3 FBA98

```
--IO uint32_t FBA98
```

功能选择 GPIO.Pin_8-11

在文件 [lks32mc08x_gpio.h](#) 第 36 行定义.

2.12.2.4 FFEDC

```
--IO uint32_t FFEDC
```

功能选择 GPIO.Pin_12-15

在文件 [lks32mc08x_gpio.h](#) 第 37 行定义.

2.12.2.5 LCKR

```
--IO uint32_t LCKR
```

配置锁存

在文件 [lks32mc08x_gpio.h](#) 第 33 行定义.

2.12.2.6 PDE

```
--IO uint32_t PDE
```

空闲

在文件 [lks32mc08x_gpio.h](#) 第 31 行定义.

2.12.2.7 PDI

```
--IO uint32_t PDI
```

输入数据

在文件 [lks32mc08x_gpio.h](#) 第 28 行定义.

2.12.2.8 PDO

```
--IO uint32_t PDO
```

输出数据

在文件 [lks32mc08x_gpio.h](#) 第 29 行定义.

2.12.2.9 PIE

```
--IO uint32_t PIE
```

输入使能

在文件 [lks32mc08x_gpio.h](#) 第 26 行定义.

2.12.2.10 PODE

```
--IO uint32_t PODE
```

开漏使能

在文件 [lks32mc08x_gpio.h](#) 第 32 行定义.

2.12.2.11 POE

```
--IO uint32_t POE
```

输出使能

在文件 [lks32mc08x_gpio.h](#) 第 27 行定义.

2.12.2.12 PUE

--IO uint32_t PUE

上拉使能

在文件 [lks32mc08x_gpio.h](#) 第 30 行定义.

2.13 HALL_InitTypeDef结构体 参考

HALL结构体基地址定义

```
#include <lks32mc08x_hall.h>
```

成员变量

- uint16_t [FilterLen](#)
滤波长度,0对应长度1到32767对应长度32768滤波长度, 滤波时钟为系统时钟
- uint8_t [ClockDivision](#)
分频 0~3:/1 /2 /4 /8
- uint8_t [Filter75_Ena](#)
使能第一级7/5滤波,ENABLE: 使能 DISABLE: 失能
- uint8_t [HALL_Ena](#)
使能HALL,ENABLE: 使能 DISABLE: 失能
- uint8_t [Capture_IRQ_Ena](#)
HALL信号变化中断使能,ENABLE: 使能 DISABLE: 失能
- uint8_t [OverFlow_IRQ_Ena](#)
HALL计数器溢出中断使能,ENABLE: 使能 DISABLE: 失能
- uint32_t [CountTH](#)
HALL计数器门限值,有效位为24位
- uint8_t [softIE](#)
软件中断使能 ENABLE: 使能 DISABLE: 失能

2.13.1 详细描述

HALL结构体基地址定义

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 [lks32mc08x_hall.h](#) 第 42 行定义.

2.13.2 结构体成员变量说明

2.13.2.1 Capture_IRQ_Ena

uint8_t Capture_IRQ_Ena

HALL信号变化中断使能,ENABLE: 使能 DISABLE: 失能

在文件 [lks32mc08x_hall.h](#) 第 48 行定义.

2.13.2.2 ClockDivision

uint8_t ClockDivision

分频 0~3:1 /2 /4 /8

在文件 [lks32mc08x_hall.h](#) 第 45 行定义.

2.13.2.3 CountTH

uint32_t CountTH

HALL计数器门限值, 有效位为24位

在文件 [lks32mc08x_hall.h](#) 第 50 行定义.

2.13.2.4 Filter75_Ena

uint8_t Filter75_Ena

使能第一级7/5滤波,ENABLE: 使能 DISABLE: 失能

在文件 [lks32mc08x_hall.h](#) 第 46 行定义.

2.13.2.5 FilterLen

uint16_t FilterLen

滤波长度,0对应长度1到32767对应长度32768滤波长度, 滤波时钟为系统时钟

在文件 [lks32mc08x_hall.h](#) 第 44 行定义.

2.13.2.6 HALL_Ena

uint8_t HALL_Ena

使能HALL,ENABLE: 使能 DISABLE: 失能

在文件 [lks32mc08x_hall.h](#) 第 47 行定义.

2.13.2.7 OverFlow_IRQ_Ena

uint8_t OverFlow_IRQ_Ena

HALL计数器溢出中断使能,ENABLE: 使能 DISABLE: 失能

在文件 [lks32mc08x_hall.h](#) 第 49 行定义.

2.13.2.8 softIE

uint8_t softIE

软件中断使能 ENABLE: 使能 DISABLE: 失能

在文件 [lks32mc08x_hall.h](#) 第 51 行定义.

2.14 HALL_TypeDef结构体 参考

HALL寄存器结构体句柄

```
#include <lks32mc08x_hall.h>
```

成员变量

- __IO uint32_t [CFG](#)
HALL 模块配置寄存器
- __IO uint32_t [INFO](#)
HALL 模块信息寄存器
- __IO uint32_t [WIDTH](#)
HALL 宽度计数值寄存器
- __IO uint32_t [TH](#)
HALL 模块计数器门限值寄存器
- __IO uint32_t [CNT](#)
HALL 计数寄存器

2.14.1 详细描述

HALL寄存器结构体句柄

在文件 [lks32mc08x_hall.h](#) 第 23 行定义.

2.14.2 结构体成员变量说明

2.14.2.1 CFG

__IO uint32_t CFG

HALL 模块配置寄存器

在文件 [lks32mc08x_hall.h](#) 第 25 行定义.

2.14.2.2 CNT

```
--IO uint32_t CNT
```

HALL 计数寄存器

在文件 [lks32mc08x_hall.h](#) 第 29 行定义.

2.14.2.3 INFO

```
--IO uint32_t INFO
```

HALL 模块信息寄存器

在文件 [lks32mc08x_hall.h](#) 第 26 行定义.

2.14.2.4 TH

```
--IO uint32_t TH
```

HALL 模块计数器门限值寄存器

在文件 [lks32mc08x_hall.h](#) 第 28 行定义.

2.14.2.5 WIDTH

```
--IO uint32_t WIDTH
```

HALL 宽度计数值寄存器

在文件 [lks32mc08x_hall.h](#) 第 27 行定义.

2.15 I2C_InitTypeDef结构体 参考

I2C结构体基地址定义

```
#include <lks32mc08x_i2c.h>
```

成员变量

- u8 **ADRCMP**
I2C 硬件地址比较使能开关，只有在 DMA 模式下开启才有效
- u8 **MST_MODE**
I2C 主模式使能；ENABLE:使能，DISABLE:失能
- u8 **SLV_MODE**
I2C 从模式使能；ENABLE:使能，DISABLE:失能
- u8 **DMA**
I2C DMA传输使能；ENABLE:使能，DISABLE:失能
- u32 **BaudRate**
I2C 波特率
- u8 **IE**
I2C 中断使能；ENABLE:使能，DISABLE:失能
- u8 **TC_IE**
I2C 数据传输完成中断使能；ENABLE:使能，DISABLE:失能
- u8 **BUS_ERR_IE**
I2C 总线错误事件中断使能；ENABLE:使能，DISABLE:失能
- u8 **STOP_IE**
I2C STOP 事件中断使能；ENABLE:使能，DISABLE:失能
- u8 **BURST_NACK**
I2C 传输，NACK 事件中断使能；ENABLE:使能，DISABLE:失能
- u8 **BURST_ADDR_CMP**
I2C 传输，硬件地址匹配中断使能；ENABLE:使能，DISABLE:失能

2.15.1 详细描述

I2C结构体基地址定义

I2C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 [lks32mc08x_i2c.h](#) 第 43 行定义。

2.15.2 结构体成员变量说明

2.15.2.1 ADRCMP

u8 ADRCMP

I2C 硬件地址比较使能开关，只有在 DMA 模式下开启才有效

在文件 [lks32mc08x_i2c.h](#) 第 45 行定义。

2.15.2.2 BaudRate

u32 BaudRate

I2C 波特率

在文件 [lks32mc08x_i2c.h](#) 第 49 行定义。

2.15.2.3 BURST_ADDR_CMP

u8 BURST_ADDR_CMP

I2C 传输，硬件地址匹配中断使能；ENABLE:使能，DISABLE:失能

在文件 [lks32mc08x_i2c.h](#) 第 55 行定义.

2.15.2.4 BURST_NACK

u8 BURST_NACK

I2C 传输，NACK 事件中断使能；ENABLE:使能，DISABLE:失能

在文件 [lks32mc08x_i2c.h](#) 第 54 行定义.

2.15.2.5 BUS_ERR_IE

u8 BUS_ERR_IE

I2C 总线错误事件中断使能；ENABLE:使能，DISABLE:失能

在文件 [lks32mc08x_i2c.h](#) 第 52 行定义.

2.15.2.6 DMA

u8 DMA

I2C DMA传输使能；ENABLE:使能，DISABLE:失能

在文件 [lks32mc08x_i2c.h](#) 第 48 行定义.

2.15.2.7 IE

u8 IE

I2C 中断使能；ENABLE:使能，DISABLE:失能

在文件 [lks32mc08x_i2c.h](#) 第 50 行定义.

2.15.2.8 MST_MODE

u8 MST_MODE

I2C 主模式使能；ENABLE:使能，DISABLE:失能

在文件 [lks32mc08x_i2c.h](#) 第 46 行定义.

2.15.2.9 SLV_MODE

u8 SLV_MODE

I2C 从模式使能; ENABLE:使能, DISABLE:失能

在文件 [lks32mc08x_i2c.h](#) 第 47 行定义.

2.15.2.10 STOP_IE

u8 STOP_IE

I2C STOP 事件中断使能; ENABLE:使能, DISABLE:失能

在文件 [lks32mc08x_i2c.h](#) 第 53 行定义.

2.15.2.11 TC_IE

u8 TC_IE

I2C 数据传输完成中断使能; ENABLE:使能, DISABLE:失能

在文件 [lks32mc08x_i2c.h](#) 第 51 行定义.

2.16 I2C_TypeDef结构体 参考

I2C寄存器结构体句柄

```
#include <lks32mc08x_i2c.h>
```

成员变量

- __IO uint32_t ADDR
- __IO uint32_t CFG
- __IO uint32_t SCR
- __IO uint32_t DATA
- __IO uint32_t MSCR
- __IO uint32_t BCR

2.16.1 详细描述

I2C寄存器结构体句柄

在文件 [lks32mc08x_i2c.h](#) 第 23 行定义.

2.16.2 结构体成员变量说明

2.16.2.1 ADDR

```
--IO uint32_t ADDR
```

在文件 [lks32mc08x_i2c.h](#) 第 25 行定义.

2.16.2.2 BCR

```
--IO uint32_t BCR
```

在文件 [lks32mc08x_i2c.h](#) 第 30 行定义.

2.16.2.3 CFG

```
--IO uint32_t CFG
```

在文件 [lks32mc08x_i2c.h](#) 第 26 行定义.

2.16.2.4 DATA

```
--IO uint32_t DATA
```

在文件 [lks32mc08x_i2c.h](#) 第 28 行定义.

2.16.2.5 MSCR

```
--IO uint32_t MSCR
```

在文件 [lks32mc08x_i2c.h](#) 第 29 行定义.

2.16.2.6 SCR

```
--IO uint32_t SCR
```

在文件 [lks32mc08x_i2c.h](#) 第 27 行定义.

2.17 IWDG_InitTypeDef结构体 参考

IWDG寄存器结构体句柄

```
#include <lks32mc08x-iwdg.h>
```

成员变量

- u32 [WDG_EN](#)
独立看门狗使能
- u32 [RTH](#)
看门狗超时复位时间

2.17.1 详细描述

IWDG寄存器结构体句柄

在文件 [lks32mc08x_iwdg.h](#) 第 23 行定义.

2.17.2 结构体成员变量说明

2.17.2.1 RTH

u32 RTH

看门狗超时复位时间

在文件 [lks32mc08x_iwdg.h](#) 第 26 行定义.

2.17.2.2 WDG_EN

u32 WDG_EN

独立看门狗使能

在文件 [lks32mc08x_iwdg.h](#) 第 25 行定义.

2.18 MCPWM_InitTypeDef结构体 参考

MCPWM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 <lks32mc08x_mcpwm.h>
```

成员变量

- uint16_t **MCPWM_PERIOD**
MCPWM输出周期
#define PWM_PERIOD ((u16) (PWM_MCLK / (u32)(2 * PWM_FREQ * (PWM_PRSC+1))))
PWM_PERIOD为MCPWM_PERIOD赋值, PWM_MCLK为PWM模块时钟频率如96000000HZ
PWM_FREQ为PWM斩波频率如16000Hz即16KHz, PWM_PRSC为PWM分频系数为0/1/3/7
- uint8_t **CLK_DIV**
MCPWM 分频系数 为0:1分频, 1:2分频, 2:4分频, 3:8分频
- uint8_t **MCLK_EN**
MCPWM 时钟使能开关: ENABLE使能, DISABLE关闭
- uint8_t **MCPWM_Cnt_EN**
MCPWM 主计数器使能开关: ENABLE使能, DISABLE关闭
- uint8_t **GPIO_BKIN_Filter**
GPIO输入滤波时钟设置1-16
- uint8_t **CMP_BKIN_Filter**
比较器CMP输入滤波时钟设置1-16
- uint16_t **MCPWM_WorkModeCH0**
MCPWM CH0工作模式: 边沿对齐/中心对齐
- uint16_t **MCPWM_WorkModeCH1**
MCPWM CH1工作模式: 边沿对齐/中心对齐
- uint16_t **MCPWM_WorkModeCH2**
MCPWM CH2工作模式: 边沿对齐/中心对齐
- uint16_t **MCPWM_WorkModeCH3**
MCPWM CH3工作模式: 边沿对齐/中心对齐
- uint16_t **TriggerPoint0**
PWM触发ADC事件0, 时间点设置, 取值范围 (-MCPWM_PERIOD~MCPWM_PERIOD)
- uint16_t **TriggerPoint1**
PWM触发ADC事件1, 时间点设置, 取值范围 (-MCPWM_PERIOD~MCPWM_PERIOD)
- uint16_t **TriggerPoint2**
PWM触发ADC事件2, 时间点设置, 取值范围 (-MCPWM_PERIOD~MCPWM_PERIOD)
- uint16_t **TriggerPoint3**
PWM触发ADC事件3, 时间点设置, 取值范围 (-MCPWM_PERIOD~MCPWM_PERIOD)
- uint16_t **DeadTimeCH0N**
CH0N死区时间设置
#define DEADTIME (u16)((unsigned long long)PWM_MCLK * (unsigned long long)DEADTIME_NS/1000000000uL)
DEADTIME为DeadTimeCH0N~DeadTimeCH3P值, PWM_MCLK主频如96000000HZ, DEADTIME_NS死区时间如1200单位纳秒
- uint16_t **DeadTimeCH0P**
CH0P死区时间设置
#define DEADTIME (u16)((unsigned long long)PWM_MCLK * (unsigned long long)DEADTIME_NS/1000000000uL)
DEADTIME为DeadTimeCH0N~DeadTimeCH3P值, PWM_MCLK主频如96000000HZ, DEADTIME_NS死区时间如1200单位纳秒
- uint16_t **DeadTimeCH1N**
CH1N死区时间设置
#define DEADTIME (u16)((unsigned long long)PWM_MCLK * (unsigned long long)DEADTIME_NS/1000000000uL)
DEADTIME为DeadTimeCH0N~DeadTimeCH3P值, PWM_MCLK主频如96000000HZ, DEADTIME_NS死区时间如1200单位纳秒
- uint16_t **DeadTimeCH1P**
CH1P死区时间设置
#define DEADTIME (u16)((unsigned long long)PWM_MCLK * (unsigned long long)DEADTIME_NS/1000000000uL)
DEADTIME为DeadTimeCH0N~DeadTimeCH3P值, PWM_MCLK主频如96000000HZ, DEADTIME_NS死区时间如1200单位纳秒
- uint16_t **DeadTimeCH2N**

CH2N死区时间设置

#define DEADTIME (u16)(((unsigned long long)PWM_MCLK * (unsigned long long)DEADTIME_NS)/1000000000uL)
DEADTIME为DeadTimeCH0N~DeadTimeCH3P值, PWM_MCLK主频如96000000HZ, DEADTIME_NS死区时间如1200单位纳秒

- uint16_t [DeadTimeCH2P](#)

CH2P死区时间设置

#define DEADTIME (u16)(((unsigned long long)PWM_MCLK * (unsigned long long)DEADTIME_NS)/1000000000uL)
DEADTIME为DeadTimeCH0N~DeadTimeCH3P值, PWM_MCLK主频如96000000HZ, DEADTIME_NS死区时间如1200单位纳秒

- uint16_t [DeadTimeCH3N](#)

CH3N死区时间设置

#define DEADTIME (u16)(((unsigned long long)PWM_MCLK * (unsigned long long)DEADTIME_NS)/1000000000uL)
DEADTIME为DeadTimeCH0N~DeadTimeCH3P值, PWM_MCLK主频如96000000HZ, DEADTIME_NS死区时间如1200单位纳秒

- uint16_t [DeadTimeCH3P](#)

CH3P死区时间设置

#define DEADTIME (u16)(((unsigned long long)PWM_MCLK * (unsigned long long)DEADTIME_NS)/1000000000uL)
DEADTIME为DeadTimeCH0N~DeadTimeCH3P值, PWM_MCLK主频如96000000HZ, DEADTIME_NS死区时间如1200单位纳秒

- uint8_t [CH0N.Polarity_INV](#)

CH0N输出极性取反, 0:正常输出; 1:取反输出

- uint8_t [CH0P.Polarity_INV](#)

CH0P输出极性取反, 0:正常输出; 1:取反输出

- uint8_t [CH1N.Polarity_INV](#)

CH1N输出极性取反, 0:正常输出; 1:取反输出

- uint8_t [CH1P.Polarity_INV](#)

CH1P输出极性取反, 0:正常输出; 1:取反输出

- uint8_t [CH2N.Polarity_INV](#)

CH2N输出极性取反, 0:正常输出; 1:取反输出

- uint8_t [CH2P.Polarity_INV](#)

CH2P输出极性取反, 0:正常输出; 1:取反输出

- uint8_t [CH3N.Polarity_INV](#)

CH3N输出极性取反, 0:正常输出; 1:取反输出

- uint8_t [CH3P.Polarity_INV](#)

CH3P输出极性取反, 0:正常输出; 1:取反输出

- uint8_t [CH0P.SCTRLP](#)

当CH0.PS = 1时, 输出到CH0.P通道的值高电平(前提极性没有取反)

- uint8_t [CH0N.SCTRLN](#)

当CH0.NS = 1时, 输出到CH0.N通道的值高电平(前提极性没有取反)

- uint8_t [CH1P.SCTRLP](#)

当CH1.PS = 1时, 输出到CH1.P通道的值高电平(前提极性没有取反)

- uint8_t [CH1N.SCTRLN](#)

当CH1.NS = 1时, 输出到CH1.N通道的值高电平(前提极性没有取反)

- uint8_t [CH2P.SCTRLP](#)

当CH2.PS = 1时, 输出到CH2.P通道的值高电平(前提极性没有取反)

- uint8_t [CH2N.SCTRLN](#)

当CH2.NS = 1时, 输出到CH2.N通道的值高电平(前提极性没有取反)

- uint8_t [CH3P.SCTRLP](#)

当CH3.PS = 1时, 输出到CH3.P通道的值高电平(前提极性没有取反)

- uint8_t [CH3N.SCTRLN](#)

当CH3.NS = 1时, 输出到CH3.PN通道的值高电平(前提极性没有取反)

- uint8_t [CH0.PS](#)

- CH0_P*信号来源, 1: 来自*CH0_SCTRLP* (软件控制*PWM*输出); 0: 来自*MCPWM*实际运行计数器 (硬件计数器控制*PWM*输出)
- uint8_t [CH0_NS](#)
 - CH0_N*信号来源, 1: 来自*CH0_SCTRLN* (软件控制*PWM*输出); 0: 来自*MCPWM*实际运行计数器 (硬件计数器控制*PWM*输出)
- uint8_t [CH1_PS](#)
 - CH1_P*信号来源, 1: 来自*CH1_SCTRLP* (软件控制*PWM*输出); 0: 来自*MCPWM*实际运行计数器 (硬件计数器控制*PWM*输出)
- uint8_t [CH1_NS](#)
 - CH1_N*信号来源, 1: 来自*CH1_SCTRLN* (软件控制*PWM*输出); 0: 来自*MCPWM*实际运行计数器 (硬件计数器控制*PWM*输出)
- uint8_t [CH2_PS](#)
 - CH2_P*信号来源, 1: 来自*CH2_SCTRLP* (软件控制*PWM*输出); 0: 来自*MCPWM*实际运行计数器 (硬件计数器控制*PWM*输出)
- uint8_t [CH2_NS](#)
 - CH2_N*信号来源, 1: 来自*CH2_SCTRLN* (软件控制*PWM*输出); 0: 来自*MCPWM*实际运行计数器 (硬件计数器控制*PWM*输出)
- uint8_t [CH3_PS](#)
 - CH3_P*信号来源, 1: 来自*CH3_SCTRLP* (软件控制*PWM*输出); 0: 来自*MCPWM*实际运行计数器 (硬件计数器控制*PWM*输出)
- uint8_t [CH3_NS](#)
 - CH3_N*信号来源, 1: 来自*CH3_SCTRLN* (软件控制*PWM*输出); 0: 来自*MCPWM*实际运行计数器 (硬件计数器控制*PWM*输出)
- uint8_t [Switch_CH0N.CH0P](#)
 - 交换*CH0N*, *CH0P*信号输出使能开关: *ENABLE*使能, *DISABLE*关闭
- uint8_t [Switch_CH1N.CH1P](#)
 - 交换*CH1N*, *CH1P*信号输出使能开关: *ENABLE*使能, *DISABLE*关闭
- uint8_t [Switch_CH2N.CH2P](#)
 - 交换*CH2N*, *CH2P*信号输出使能开关: *ENABLE*使能, *DISABLE*关闭
- uint8_t [Switch_CH3N.CH3P](#)
 - 交换*CH3N*, *CH3P*信号输出使能开关: *ENABLE*使能, *DISABLE*关闭
- uint8_t [MCPWM.UpdateInterval](#)
 - MCPWM T0/T1*事件更新间隔, 0~15对应发送更新事件0~15次才更新一次
- uint8_t [MCPWM.T0.UpdateEN](#)
 - MCPWM T0*事件更新使能: *ENABLE*使能, *DISABLE*关闭
- uint8_t [MCPWM.T1.UpdateEN](#)
 - MCPWM T1*事件更新使能: *ENABLE*使能, *DISABLE*关闭
- uint8_t [MCPWM.Auto_ERR_EN](#)
 - 当发生*FAIL*事件关闭*MCPWM*时 (即*MOE*关闭), *MCPWM*更新事件是否自动打开*MOE*, 使能开关
- uint8_t [FAIL0.INPUT_EN](#)
 - FAIL0* 输入功能使能: *ENABLE*使能, *DISABLE*关闭
- uint8_t [FAIL1.INPUT_EN](#)
 - FAIL1* 输入功能使能: *ENABLE*使能, *DISABLE*关闭
- uint8_t [FAIL0.Signal.Sel](#)
 - FAIL0* 信号选择, 比较器0 或 *GPIO*(*MCPWM.BKIN0*)
- uint8_t [FAIL1.Signal.Sel](#)
 - FAIL1* 信号选择, 比较器1 或 *GPIO*(*MCPWM.BKIN1*)
- uint8_t [FAIL0.Polarity](#)
 - FAIL0* 信号极性设置, 高有效或低有效
- uint8_t [FAIL1.Polarity](#)
 - FAIL1* 信号极性设置, 高有效或低有效
- uint8_t [DebugMode_PWM_out](#)

- Debug*时, *MCU*进入*Halt*, *MCPWM*信号是否正常输出
- uint8_t [CH0P.default.output](#)
CH0P MOE为0时或发生*FAIL*事件时, 默认电平输出,
 - uint8_t [CH0N.default.output](#)
CH0N MOE为0时或发生*FAIL*事件时, 默认电平输出,
 - uint8_t [CH1P.default.output](#)
CH1P MOE为0时或发生*FAIL*事件时, 默认电平输出,
 - uint8_t [CH1N.default.output](#)
CH1N MOE为0时或发生*FAIL*事件时, 默认电平输出,
 - uint8_t [CH2P.default.output](#)
CH2P MOE为0时或发生*FAIL*事件时, 默认电平输出,
 - uint8_t [CH2N.default.output](#)
CH2N MOE为0时或发生*FAIL*事件时, 默认电平输出,
 - uint8_t [CH3P.default.output](#)
CH3P MOE为0时或发生*FAIL*事件时, 默认电平输出,
 - uint8_t [CH3N.default.output](#)
CH3N MOE为0时或发生*FAIL*事件时, 默认电平输出,
 - uint8_t [T0.Update.INT.EN](#)
*T0*更新事件中断使能: *ENABLE*使能, *DISABLE*关闭
 - uint8_t [T1.Update.INT.EN](#)
*T1*更新事件中断使能: *ENABLE*使能, *DISABLE*关闭
 - uint8_t [TH00.Match.INT.EN](#)
*TH00*与计数器值匹配事件中断使能: *ENABLE*使能, *DISABLE*关闭
 - uint8_t [TH01.Match.INT.EN](#)
*TH01*与计数器值匹配事件中断使能: *ENABLE*使能, *DISABLE*关闭
 - uint8_t [TH10.Match.INT.EN](#)
*TH10*与计数器值匹配事件中断使能: *ENABLE*使能, *DISABLE*关闭
 - uint8_t [TH11.Match.INT.EN](#)
*TH11*与计数器值匹配事件中断使能: *ENABLE*使能, *DISABLE*关闭
 - uint8_t [TH20.Match.INT.EN](#)
*TH20*与计数器值匹配事件中断使能: *ENABLE*使能, *DISABLE*关闭
 - uint8_t [TH21.Match.INT.EN](#)
*TH21*与计数器值匹配事件中断使能: *ENABLE*使能, *DISABLE*关闭
 - uint8_t [TH30.Match.INT.EN](#)
*TH30*与计数器值匹配事件中断使能: *ENABLE*使能, *DISABLE*关闭
 - uint8_t [TH31.Match.INT.EN](#)
*TH31*与计数器值匹配事件中断使能: *ENABLE*使能, *DISABLE*关闭
 - uint8_t [TMR0.Match.INT.EN](#)
*TMR0*与计数器值匹配事件中断使能: *ENABLE*使能, *DISABLE*关闭
 - uint8_t [TMR1.Match.INT.EN](#)
*TMR1*与计数器值匹配事件中断使能: *ENABLE*使能, *DISABLE*关闭
 - uint8_t [TMR2.Match.INT.EN](#)
*TMR2*与计数器值匹配事件中断使能: *ENABLE*使能, *DISABLE*关闭
 - uint8_t [TMR3.Match.INT.EN](#)
*TMR3*与计数器值匹配事件中断使能: *ENABLE*使能, *DISABLE*关闭
 - uint8_t [MCPWM.REG.UPDATE.INT.EN](#)
*MCPWM*寄存器更新事件中断使能: *ENABLE*使能, *DISABLE*关闭
 - uint8_t [FAIL0.INT.EN](#)
*FAIL0*事件中断使能: *ENABLE*使能, *DISABLE*关闭
 - uint8_t [FAIL1.INT.EN](#)
*FAIL1*事件中断使能: *ENABLE*使能, *DISABLE*关闭

- `uint8_t IO0_PPE`
IO0推挽模式使能: *ENABLE*使能, *DISABLE*关闭
- `uint8_t IO1_PPE`
IO1推挽模式使能: *ENABLE*使能, *DISABLE*关闭
- `uint8_t IO2_PPE`
IO2推挽模式使能: *ENABLE*使能, *DISABLE*关闭
- `uint8_t IO3_PPE`
IO3推挽模式使能: *ENABLE*使能, *DISABLE*关闭

2.18.1 详细描述

MCPWM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 [lks32mc08x_mcpwm.h](#) 第 25 行定义.

2.18.2 结构体成员变量说明

2.18.2.1 CH0_NS

`uint8_t CH0_NS`

CH0_N信号来源, 1: 来自CH0_SCTRLN (软件控制PWM输出); 0: 来自MCPWM实际运行计数器 (硬件计数器控制PWM输出)

在文件 [lks32mc08x_mcpwm.h](#) 第 117 行定义.

2.18.2.2 CH0_PS

`uint8_t CH0_PS`

CH0_P信号来源, 1: 来自CH0_SCTRLP (软件控制PWM输出); 0: 来自MCPWM实际运行计数器 (硬件计数器控制PWM输出)

在文件 [lks32mc08x_mcpwm.h](#) 第 116 行定义.

2.18.2.3 CH0N_default_output

`uint8_t CH0N_default_output`

CH0N MOE为0时或发生FAIL事件时, 默认电平输出,

参见

[HIGH_LEVEL](#)

[LOW_LEVEL](#)

在文件 [lks32mc08x_mcpwm.h](#) 第 144 行定义.

2.18.2.4 CH0N.Polarity_INV

uint8_t CH0N.Polarity_INV

CH0N输出极性取反，0:正常输出；1:取反输出

在文件 [lks32mc08x_mcpwm.h](#) 第 98 行定义.

2.18.2.5 CH0N.SCTRLN

uint8_t CH0N.SCTRLN

当CH0_NS = 1时，输出到CH0.N通道的值高电平(前提极性没有取反)

在文件 [lks32mc08x_mcpwm.h](#) 第 108 行定义.

2.18.2.6 CH0P.default_output

uint8_t CH0P.default_output

CH0P MOE为0时或发生FAIL事件时，默认电平输出，

参见

[HIGH.LEVEL](#)

[LOW.LEVEL](#)

在文件 [lks32mc08x_mcpwm.h](#) 第 143 行定义.

2.18.2.7 CH0P.Polarity_INV

uint8_t CH0P.Polarity_INV

CH0P输出极性取反，0:正常输出；1:取反输出

在文件 [lks32mc08x_mcpwm.h](#) 第 99 行定义.

2.18.2.8 CH0P.SCTRLP

uint8_t CH0P.SCTRLP

当CH0_PS = 1时，输出到CH0.P通道的值高电平(前提极性没有取反)

在文件 [lks32mc08x_mcpwm.h](#) 第 107 行定义.

2.18.2.9 CH1_NS

uint8_t CH1_NS

CH1_N信号来源，1：来自CH1_SCTRLN（软件控制PWM输出）；0：来自MCPWM实际运行计数器（硬件计数器控制PWM输出）

在文件 [lks32mc08x_mcpwm.h](#) 第 119 行定义.

2.18.2.10 CH1_PS

uint8_t CH1_PS

CH1_P信号来源，1：来自CH1_SCTRLP（软件控制PWM输出）；0：来自MCPWM实际运行计数器（硬件计数器控制PWM输出）

在文件 [lks32mc08x_mcpwm.h](#) 第 118 行定义.

2.18.2.11 CH1N.default_output

uint8_t CH1N_default_output

CH1N MOE为0时或发生FAIL事件时，默认电平输出，

参见

[HIGH_LEVEL](#)

[LOW_LEVEL](#)

在文件 [lks32mc08x_mcpwm.h](#) 第 146 行定义.

2.18.2.12 CH1N.Polarity_INV

uint8_t CH1N_Polarity_INV

CH1N输出极性取反，0:正常输出；1:取反输出

在文件 [lks32mc08x_mcpwm.h](#) 第 100 行定义.

2.18.2.13 CH1N.SCTRLN

uint8_t CH1N_SCTRLN

当CH1_NS = 1时，输出到CH1_N通道的值高电平(前提极性没有取反)

在文件 [lks32mc08x_mcpwm.h](#) 第 110 行定义.

2.18.2.14 CH1P_default_output

uint8_t CH1P_default_output

CH1P MOE为0时或发生FAIL事件时，默认电平输出，

参见

[HIGH_LEVEL](#)

[LOW_LEVEL](#)

在文件 [lks32mc08x_mcpwm.h](#) 第 145 行定义.

2.18.2.15 CH1P.Polarity_INV

uint8_t CH1P.Polarity_INV

CH1P输出极性取反，0:正常输出；1:取反输出

在文件 [lks32mc08x_mcpwm.h](#) 第 101 行定义.

2.18.2.16 CH1P_SCTRLP

uint8_t CH1P_SCTRLP

当CH1_PS = 1时，输出到CH1.P通道的值高电平(前提极性没有取反)

在文件 [lks32mc08x_mcpwm.h](#) 第 109 行定义.

2.18.2.17 CH2_NS

uint8_t CH2_NS

CH2_N信号来源，1：来自CH2_SCTRLN（软件控制PWM输出）；0：来自MCPWM实际运行计数器（硬件计数器控制PWM输出）

在文件 [lks32mc08x_mcpwm.h](#) 第 121 行定义.

2.18.2.18 CH2_PS

uint8_t CH2_PS

CH2_P信号来源，1：来自CH2_SCTRLP（软件控制PWM输出）；0：来自MCPWM实际运行计数器（硬件计数器控制PWM输出）

在文件 [lks32mc08x_mcpwm.h](#) 第 120 行定义.

2.18.2.19 CH2N.default_output

uint8_t CH2N.default_output

CH2N MOE为0时或发生FAIL事件时，默认电平输出，

参见

[HIGH_LEVEL](#)

[LOW_LEVEL](#)

在文件 [lks32mc08x_mcpwm.h](#) 第 148 行定义.

2.18.2.20 CH2N.Polarity_INV

uint8_t CH2N.Polarity_INV

CH2N输出极性取反，0:正常输出；1:取反输出

在文件 [lks32mc08x_mcpwm.h](#) 第 102 行定义.

2.18.2.21 CH2N.SCTRLN

uint8_t CH2N.SCTRLN

当CH2_NS = 1时，输出到CH2.N通道的值高电平(前提极性没有取反)

在文件 [lks32mc08x_mcpwm.h](#) 第 112 行定义.

2.18.2.22 CH2P.default_output

uint8_t CH2P.default_output

CH2P MOE为0时或发生FAIL事件时，默认电平输出，

参见

[HIGH_LEVEL](#)

[LOW_LEVEL](#)

在文件 [lks32mc08x_mcpwm.h](#) 第 147 行定义.

2.18.2.23 CH2P.Polarity_INV

uint8_t CH2P.Polarity_INV

CH2P输出极性取反，0:正常输出；1:取反输出

在文件 [lks32mc08x_mcpwm.h](#) 第 103 行定义.

2.18.2.24 CH2P_SCTRLP

uint8_t CH2P_SCTRLP

当CH2.PS = 1时，输出到CH2.P通道的值高电平(前提极性没有取反)

在文件 [lks32mc08x_mcpwm.h](#) 第 111 行定义.

2.18.2.25 CH3_NS

uint8_t CH3_NS

CH3_N信号来源，1: 来自CH3.SCTRLN（软件控制PWM输出）；0: 来自MCPWM实际运行计数器（硬件计数器控制PWM输出）

在文件 [lks32mc08x_mcpwm.h](#) 第 123 行定义.

2.18.2.26 CH3_PS

uint8_t CH3_PS

CH3_P信号来源，1: 来自CH3.SCTRLP（软件控制PWM输出）；0: 来自MCPWM实际运行计数器（硬件计数器控制PWM输出）

在文件 [lks32mc08x_mcpwm.h](#) 第 122 行定义.

2.18.2.27 CH3N.default_output

uint8_t CH3N_default_output

CH3N MOE为0时或发生FAIL事件时，默认电平输出，

参见

[HIGH_LEVEL](#)

[LOW_LEVEL](#)

在文件 [lks32mc08x_mcpwm.h](#) 第 150 行定义.

2.18.2.28 CH3N.Polarity_INV

uint8_t CH3N_Polarity_INV

CH3N输出极性取反，0:正常输出；1:取反输出

在文件 [lks32mc08x_mcpwm.h](#) 第 104 行定义.

2.18.2.29 CH3N_SCTRLN

uint8_t CH3N_SCTRLN

当CH3_NS = 1时，输出到CH3_PN通道的值高电平(前提极性没有取反)

在文件 [lks32mc08x_mcpwm.h](#) 第 114 行定义.

2.18.2.30 CH3P_default_output

uint8_t CH3P_default_output

CH3P MOE为0时或发生FAIL事件时，默认电平输出，

参见

[HIGH_LEVEL](#)

[LOW_LEVEL](#)

在文件 [lks32mc08x_mcpwm.h](#) 第 149 行定义.

2.18.2.31 CH3P_Polarity_INV

uint8_t CH3P_Polarity_INV

CH3P输出极性取反，0:正常输出；1:取反输出

在文件 [lks32mc08x_mcpwm.h](#) 第 105 行定义.

2.18.2.32 CH3P_SCTRLP

uint8_t CH3P_SCTRLP

当CH3_PS = 1时，输出到CH3_P通道的值高电平(前提极性没有取反)

在文件 [lks32mc08x_mcpwm.h](#) 第 113 行定义.

2.18.2.33 CLK_DIV

uint8_t CLK_DIV

MCPWM 分频系数 为0:1分频，1:2分频，2:4分频，3:8分频

在文件 [lks32mc08x_mcpwm.h](#) 第 34 行定义.

2.18.2.34 CMP_BKIN_Filter

uint8_t CMP_BKIN_Filter

比较器CMP输入滤波时钟设置1-16

在文件 [lks32mc08x_mcpwm.h](#) 第 38 行定义.

2.18.2.35 DeadTimeCH0N

uint16_t DeadTimeCH0N

CH0N死区时间设置

#define DEADTIME (u16)((((unsigned long long)PWM_MCLK * (unsigned long long)DEADTIME_NS)/1000000000uL)

DEADTIME为DeadTimeCH0N~DeadTimeCH3P值，PWM_MCLK主频如96000000HZ，DEADTIME_NS死区时间如1200单位纳秒

在文件 [lks32mc08x_mcpwm.h](#) 第 54 行定义.

2.18.2.36 DeadTimeCH0P

uint16_t DeadTimeCH0P

CH0P死区时间设置

#define DEADTIME (u16)((((unsigned long long)PWM_MCLK * (unsigned long long)DEADTIME_NS)/1000000000uL)

DEADTIME为DeadTimeCH0N~DeadTimeCH3P值，PWM_MCLK主频如96000000HZ，DEADTIME_NS死区时间如1200单位纳秒

在文件 [lks32mc08x_mcpwm.h](#) 第 60 行定义.

2.18.2.37 DeadTimeCH1N

uint16_t DeadTimeCH1N

CH1N死区时间设置

#define DEADTIME (u16)((((unsigned long long)PWM_MCLK * (unsigned long long)DEADTIME_NS)/1000000000uL)

DEADTIME为DeadTimeCH0N~DeadTimeCH3P值，PWM_MCLK主频如96000000HZ，DEADTIME_NS死区时间如1200单位纳秒

在文件 [lks32mc08x_mcpwm.h](#) 第 66 行定义.

2.18.2.38 DeadTimeCH1P

uint16_t DeadTimeCH1P

CH1P死区时间设置

#define DEADTIME (u16)((((unsigned long long)PWM_MCLK * (unsigned long long)DEADTIME_NS)/1000000000uL)

DEADTIME为DeadTimeCH0N~DeadTimeCH3P值，PWM_MCLK主频如96000000HZ，DEADTIME_NS死区时间如1200单位纳秒

在文件 [lks32mc08x_mcpwm.h](#) 第 72 行定义.

2.18.2.39 DeadTimeCH2N

uint16_t DeadTimeCH2N

CH2N死区时间设置

#define DEADTIME (u16)((((unsigned long long)PWM_MCLK * (unsigned long long)DEADTIME_NS)/1000000000uL)

DEADTIME为DeadTimeCH0N~DeadTimeCH3P值，PWM_MCLK主频如96000000HZ，DEADTIME_NS死区时间如1200单位纳秒

在文件 [lks32mc08x_mcpwm.h](#) 第 78 行定义.

2.18.2.40 DeadTimeCH2P

uint16_t DeadTimeCH2P

CH2P死区时间设置

#define DEADTIME (u16)((((unsigned long long)PWM_MCLK * (unsigned long long)DEADTIME_NS)/1000000000uL)

DEADTIME为DeadTimeCH0N~DeadTimeCH3P值，PWM_MCLK主频如96000000HZ，DEADTIME_NS死区时间如1200单位纳秒

在文件 [lks32mc08x_mcpwm.h](#) 第 84 行定义.

2.18.2.41 DeadTimeCH3N

uint16_t DeadTimeCH3N

CH3N死区时间设置

#define DEADTIME (u16)((((unsigned long long)PWM_MCLK * (unsigned long long)DEADTIME_NS)/1000000000uL)

DEADTIME为DeadTimeCH0N~DeadTimeCH3P值，PWM_MCLK主频如96000000HZ，DEADTIME_NS死区时间如1200单位纳秒

在文件 [lks32mc08x_mcpwm.h](#) 第 90 行定义.

2.18.2.42 DeadTimeCH3P

uint16_t DeadTimeCH3P

CH3P死区时间设置

#define DEADTIME (u16)((((unsigned long long)PWM_MCLK * (unsigned long long)DEADTIME_NS)/1000000000uL)

DEADTIME为DeadTimeCH0N~DeadTimeCH3P值，PWM_MCLK主频如96000000HZ，DEADTIME_NS死区时间如1200单位纳秒

在文件 [lks32mc08x_mcpwm.h](#) 第 96 行定义.

2.18.2.43 DebugMode_PWM_out

```
uint8_t DebugMode_PWM_out
```

Debug时，MCU进入Halt, MCPWM信号是否正常输出

在文件 [lks32mc08x_mcpwm.h](#) 第 141 行定义.

2.18.2.44 FAIL0_INPUT_EN

```
uint8_t FAIL0_INPUT_EN
```

FAIL0 输入功能使能: ENABLE使能, DISABLE关闭

在文件 [lks32mc08x_mcpwm.h](#) 第 135 行定义.

2.18.2.45 FAIL0_INT_EN

```
uint8_t FAIL0_INT_EN
```

FAIL0事件中中断使能: ENABLE使能, DISABLE关闭

在文件 [lks32mc08x_mcpwm.h](#) 第 171 行定义.

2.18.2.46 FAIL0.Polarity

```
uint8_t FAIL0.Polarity
```

FAIL0 信号极性设置, 高有效或低有效

参见

[HIGH_LEVEL_ACTIVE](#)

[LOW_LEVEL_ACTIVE](#)

在文件 [lks32mc08x_mcpwm.h](#) 第 139 行定义.

2.18.2.47 FAIL0.Signal_Sel

```
uint8_t FAIL0.Signal_Sel
```

FAIL0 信号选择, 比较器0 或 GPIO(MCPWM_BKIN0)

参见

[FAIL_SEL_CMP](#)

[FAIL_SEL_IO](#)

在文件 [lks32mc08x_mcpwm.h](#) 第 137 行定义.

2.18.2.48 FAIL1.INPUT_EN

uint8_t FAIL1.INPUT_EN

FAIL1 输入功能使能: ENABLE使能, DISABLE关闭

在文件 [lks32mc08x_mcpwm.h](#) 第 136 行定义.

2.18.2.49 FAIL1.INT_EN

uint8_t FAIL1.INT_EN

FAIL1事件中断使能: ENABLE使能, DISABLE关闭

在文件 [lks32mc08x_mcpwm.h](#) 第 172 行定义.

2.18.2.50 FAIL1.Polarity

uint8_t FAIL1.Polarity

FAIL1 信号极性设置, 高有效或低有效

参见

[HIGH_LEVEL_ACTIVE](#)

[LOW_LEVEL_ACTIVE](#)

在文件 [lks32mc08x_mcpwm.h](#) 第 140 行定义.

2.18.2.51 FAIL1.Signal_Sel

uint8_t FAIL1.Signal_Sel

FAIL1 信号选择, 比较器1 或 GPIO(MCPWM_BKIN1)

参见

[FAIL_SEL_CMP](#)

[FAIL_SEL_IO](#)

在文件 [lks32mc08x_mcpwm.h](#) 第 138 行定义.

2.18.2.52 GPIO_BKIN_Filter

uint8_t GPIO_BKIN_Filter

GPIO输入滤波时钟设置1-16

在文件 [lks32mc08x_mcpwm.h](#) 第 37 行定义.

2.18.2.53 IO0_PPE

uint8_t IO0_PPE

IO0推挽模式使能: ENABLE使能, DISABLE关闭

在文件 [lks32mc08x_mcpwm.h](#) 第 174 行定义.

2.18.2.54 IO1_PPE

uint8_t IO1_PPE

IO1推挽模式使能: ENABLE使能, DISABLE关闭

在文件 [lks32mc08x_mcpwm.h](#) 第 175 行定义.

2.18.2.55 IO2_PPE

uint8_t IO2_PPE

IO2推挽模式使能: ENABLE使能, DISABLE关闭

在文件 [lks32mc08x_mcpwm.h](#) 第 176 行定义.

2.18.2.56 IO3_PPE

uint8_t IO3_PPE

IO3推挽模式使能: ENABLE使能, DISABLE关闭

在文件 [lks32mc08x_mcpwm.h](#) 第 177 行定义.

2.18.2.57 MCLK_EN

uint8_t MCLK_EN

MCPWM 时钟使能开关: ENABLE使能, DISABLE关闭

在文件 [lks32mc08x_mcpwm.h](#) 第 35 行定义.

2.18.2.58 MCPWM_Auto_ERR_EN

uint8_t MCPWM_Auto_ERR_EN

当发生FAIL事件关闭MCPWM时(即MOE关闭), MCPWM更新事件是否自动打开MOE, 使能开关

在文件 [lks32mc08x_mcpwm.h](#) 第 133 行定义.

2.18.2.59 MCPWM_Cnt_EN

```
uint8_t MCPWM_Cnt_EN
```

MCPWM 主计数器使能开关: ENABLE使能, DISABLE关闭

在文件 [lks32mc08x_mcpwm.h](#) 第 36 行定义.

2.18.2.60 MCPWM_PERIOD

```
uint16_t MCPWM_PERIOD
```

MCPWM输出周期

```
#define PWM_PERIOD ((u16) (PWM_MCLK / (u32)(2 * PWM_FREQ *(PWM_PRSC+1))))
```

PWM_PERIOD为MCPWM_PERIOD赋值, PWM_MCLK为PWM模块时钟频率如96000000HZ

PWM_FREQ为PWM斩波频率如16000Hz即16KHz,PWM_PRSC为PWM分频系数为0/1/3/7

在文件 [lks32mc08x_mcpwm.h](#) 第 33 行定义.

2.18.2.61 MCPWM_REG_UPDATE_INT_EN

```
uint8_t MCPWM_REG_UPDATE_INT_EN
```

MCPWM寄存器更新事件中断使能: ENABLE使能, DISABLE关闭

在文件 [lks32mc08x_mcpwm.h](#) 第 169 行定义.

2.18.2.62 MCPWM_T0_UpdateEN

```
uint8_t MCPWM_T0_UpdateEN
```

MCPWM T0事件更新使能: ENABLE使能, DISABLE关闭

在文件 [lks32mc08x_mcpwm.h](#) 第 131 行定义.

2.18.2.63 MCPWM_T1_UpdateEN

```
uint8_t MCPWM_T1_UpdateEN
```

MCPWM T1事件更新使能: ENABLE使能, DISABLE关闭

在文件 [lks32mc08x_mcpwm.h](#) 第 132 行定义.

2.18.2.64 MCPWM_UpdateInterval

```
uint8_t MCPWM_UpdateInterval
```

MCPWM T0/T1事件更新间隔, 0~15对应发送更新事件0~15次才更新一次

在文件 [lks32mc08x_mcpwm.h](#) 第 130 行定义.

2.18.2.65 MCPWM_WorkModeCH0

uint16_t MCPWM_WorkModeCH0

MCPWM CH0工作模式：边沿对齐/中心对齐

参见

[EDGE_PWM_MODE](#)

[CENTRAL_PWM_MODE](#)

在文件 [lks32mc08x_mcpwm.h](#) 第 40 行定义.

2.18.2.66 MCPWM_WorkModeCH1

uint16_t MCPWM_WorkModeCH1

MCPWM CH1工作模式：边沿对齐/中心对齐

参见

[EDGE_PWM_MODE](#)

[CENTRAL_PWM_MODE](#)

在文件 [lks32mc08x_mcpwm.h](#) 第 41 行定义.

2.18.2.67 MCPWM_WorkModeCH2

uint16_t MCPWM_WorkModeCH2

MCPWM CH2工作模式：边沿对齐/中心对齐

参见

[EDGE_PWM_MODE](#)

[CENTRAL_PWM_MODE](#)

在文件 [lks32mc08x_mcpwm.h](#) 第 42 行定义.

2.18.2.68 MCPWM_WorkModeCH3

uint16_t MCPWM_WorkModeCH3

MCPWM CH3工作模式：边沿对齐/中心对齐

参见

[EDGE_PWM_MODE](#)

[CENTRAL_PWM_MODE](#)

在文件 [lks32mc08x_mcpwm.h](#) 第 43 行定义.

2.18.2.69 Switch_CH0N_CH0P

```
uint8_t Switch_CH0N_CH0P
```

交换CH0N, CH0P信号输出使能开关: ENABLE使能, DISABLE关闭

在文件 [lks32mc08x_mcpwm.h](#) 第 125 行定义.

2.18.2.70 Switch_CH1N_CH1P

```
uint8_t Switch_CH1N_CH1P
```

交换CH1N, CH1P信号输出使能开关: ENABLE使能, DISABLE关闭

在文件 [lks32mc08x_mcpwm.h](#) 第 126 行定义.

2.18.2.71 Switch_CH2N_CH2P

```
uint8_t Switch_CH2N_CH2P
```

交换CH2N, CH2P信号输出使能开关: ENABLE使能, DISABLE关闭

在文件 [lks32mc08x_mcpwm.h](#) 第 127 行定义.

2.18.2.72 Switch_CH3N_CH3P

```
uint8_t Switch_CH3N_CH3P
```

交换CH3N, CH3P信号输出使能开关: ENABLE使能, DISABLE关闭

在文件 [lks32mc08x_mcpwm.h](#) 第 128 行定义.

2.18.2.73 T0_Update_INT_EN

```
uint8_t T0_Update_INT_EN
```

T0更新事件中断使能: ENABLE使能, DISABLE关闭

在文件 [lks32mc08x_mcpwm.h](#) 第 152 行定义.

2.18.2.74 T1_Update_INT_EN

```
uint8_t T1_Update_INT_EN
```

T1更新事件中断使能: ENABLE使能, DISABLE关闭

在文件 [lks32mc08x_mcpwm.h](#) 第 153 行定义.

2.18.2.75 TH00_Match_INT_EN

```
uint8_t TH00_Match_INT_EN
```

TH00与计数器值匹配事件中断使能：ENABLE使能，DISABLE关闭

在文件 [lks32mc08x_mcpwm.h](#) 第 155 行定义.

2.18.2.76 TH01_Match_INT_EN

```
uint8_t TH01_Match_INT_EN
```

TH01与计数器值匹配事件中断使能：ENABLE使能，DISABLE关闭

在文件 [lks32mc08x_mcpwm.h](#) 第 156 行定义.

2.18.2.77 TH10_Match_INT_EN

```
uint8_t TH10_Match_INT_EN
```

TH10与计数器值匹配事件中断使能：ENABLE使能，DISABLE关闭

在文件 [lks32mc08x_mcpwm.h](#) 第 157 行定义.

2.18.2.78 TH11_Match_INT_EN

```
uint8_t TH11_Match_INT_EN
```

TH11与计数器值匹配事件中断使能：ENABLE使能，DISABLE关闭

在文件 [lks32mc08x_mcpwm.h](#) 第 158 行定义.

2.18.2.79 TH20_Match_INT_EN

```
uint8_t TH20_Match_INT_EN
```

TH20与计数器值匹配事件中断使能：ENABLE使能，DISABLE关闭

在文件 [lks32mc08x_mcpwm.h](#) 第 159 行定义.

2.18.2.80 TH21_Match_INT_EN

```
uint8_t TH21_Match_INT_EN
```

TH21与计数器值匹配事件中断使能：ENABLE使能，DISABLE关闭

在文件 [lks32mc08x_mcpwm.h](#) 第 160 行定义.

2.18.2.81 TH30_Match_INT_EN

```
uint8_t TH30_Match_INT_EN
```

TH30与计数器值匹配事件中断使能：ENABLE使能，DISABLE关闭

在文件 [lks32mc08x_mcpwm.h](#) 第 161 行定义.

2.18.2.82 TH31_Match_INT_EN

```
uint8_t TH31_Match_INT_EN
```

TH31与计数器值匹配事件中断使能：ENABLE使能，DISABLE关闭

在文件 [lks32mc08x_mcpwm.h](#) 第 162 行定义.

2.18.2.83 TMR0_Match_INT_EN

```
uint8_t TMR0_Match_INT_EN
```

TMR0与计数器值匹配事件中断使能：ENABLE使能，DISABLE关闭

在文件 [lks32mc08x_mcpwm.h](#) 第 164 行定义.

2.18.2.84 TMR1_Match_INT_EN

```
uint8_t TMR1_Match_INT_EN
```

TMR1与计数器值匹配事件中断使能：ENABLE使能，DISABLE关闭

在文件 [lks32mc08x_mcpwm.h](#) 第 165 行定义.

2.18.2.85 TMR2_Match_INT_EN

```
uint8_t TMR2_Match_INT_EN
```

TMR2与计数器值匹配事件中断使能：ENABLE使能，DISABLE关闭

在文件 [lks32mc08x_mcpwm.h](#) 第 166 行定义.

2.18.2.86 TMR3_Match_INT_EN

```
uint8_t TMR3_Match_INT_EN
```

TMR3与计数器值匹配事件中断使能：ENABLE使能，DISABLE关闭

在文件 [lks32mc08x_mcpwm.h](#) 第 167 行定义.

2.18.2.87 TriggerPoint0

uint16_t TriggerPoint0

PWM触发ADC事件0，时间点设置，取值范围（-MCPWM.PERIOD~MCPWM.PERIOD）

在文件 [lks32mc08x_mcpwm.h](#) 第 45 行定义.

2.18.2.88 TriggerPoint1

uint16_t TriggerPoint1

PWM触发ADC事件1，时间点设置，取值范围（-MCPWM.PERIOD~MCPWM.PERIOD）

在文件 [lks32mc08x_mcpwm.h](#) 第 46 行定义.

2.18.2.89 TriggerPoint2

uint16_t TriggerPoint2

PWM触发ADC事件2，时间点设置，取值范围（-MCPWM.PERIOD~MCPWM.PERIOD）

在文件 [lks32mc08x_mcpwm.h](#) 第 47 行定义.

2.18.2.90 TriggerPoint3

uint16_t TriggerPoint3

PWM触发ADC事件3，时间点设置，取值范围（-MCPWM.PERIOD~MCPWM.PERIOD）

在文件 [lks32mc08x_mcpwm.h](#) 第 48 行定义.

2.19 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 <lks32mc08x_opa.h>
```

成员变量

- uint32_t [OPA_Gain](#)
运放闭环增益
- uint32_t [OPA_CLEna](#)
运放闭环使能:*Enable,Disable*

2.19.1 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 [lks32mc08x_opa.h](#) 第 31 行定义.

2.19.2 结构体成员变量说明

2.19.2.1 OPA_CLEna

```
uint32_t OPA_CLEna
```

运放闭环使能:Enable,Disable

在文件 [lks32mc08x_opa.h](#) 第 41 行定义.

2.19.2.2 OPA_Gain

```
uint32_t OPA_Gain
```

运放闭环增益

参见

[PGA_GAIN_20](#)

[PGA_GAIN_9P5](#)

[PGA_GAIN_6](#)

[PGA_GAIN_4P25](#)

在文件 [lks32mc08x_opa.h](#) 第 40 行定义.

2.20 SPI_InitTypeDef结构体 参考

SPI结构体基地址定义

```
#include <lks32mc08x_spi.h>
```

成员变量

- [SPI_Duplex Duplex](#)
全双工、半双工工作模式选择: 0, 全双工; 2, 半双工仅发送; 3, 半双工仅接收
- [uint8_t SS](#)
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: 0, 从模式; 1, 主模式
- [uint8_t CPHA](#)
相位选择: 0对应0; 1对应1
- [uint8_t CPOL](#)
极性选择: 0对应0; 1对应1
- [uint8_t DataOrder](#)
传输顺序: 0, 高位先传; 1, 低位先传
- [uint8_t EN](#)
SPI模块使能: 0, 关闭; 1, 开启
- [uint8_t IRQEna](#)
SPI中断使能: *DISABLE*: 关闭所以中断
- [uint8_t Trig](#)
传输触发选择: 0, 内部自动执行 (仅主模式有效); 1, 外部触发
- [uint8_t DMA](#)
SPI数据搬运方式: 0, DMA搬运; 1, MCU搬运
- [uint8_t BaudRate](#)
波特率设置
- [uint8_t ByteLength](#)
SPI传输数据长度, CPU搬运只能写1

2.20.1 详细描述

SPI结构体基地址定义

SPI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 [lks32mc08x_spi.h](#) 第 62 行定义.

2.20.2 结构体成员变量说明

2.20.2.1 BaudRate

`uint8_t BaudRate`

波特率设置

在文件 [lks32mc08x_spi.h](#) 第 90 行定义.

2.20.2.2 ByteLength

uint8_t ByteLength

SPI传输数据长度，CPU搬运只能写1

在文件 [lks32mc08x_spi.h](#) 第 92 行定义.

2.20.2.3 CPHA

uint8_t CPHA

相位选择: 0对应0; 1对应1

在文件 [lks32mc08x_spi.h](#) 第 75 行定义.

2.20.2.4 CPOL

uint8_t CPOL

极性选择: 0对应0; 1对应1

在文件 [lks32mc08x_spi.h](#) 第 76 行定义.

2.20.2.5 DataOrder

uint8_t DataOrder

传输顺序: 0, 高位先传; 1, 低位先传

在文件 [lks32mc08x_spi.h](#) 第 77 行定义.

2.20.2.6 DMA

uint8_t DMA

SPI数据搬运方式: 0, DMA搬运; 1, MCU搬运

在文件 [lks32mc08x_spi.h](#) 第 89 行定义.

2.20.2.7 Duplex

[SPI_Duplex](#) Duplex

全双工、半双工工作模式选择: 0, 全双工; 2, 半双工仅发送; 3, 半双工仅接收

参见

[SPI_Duplex](#)

在文件 [lks32mc08x_spi.h](#) 第 68 行定义.

2.20.2.8 EN

uint8_t EN

SPI模块使能：0，关闭；1，开启

在文件 [lks32mc08x_spi.h](#) 第 78 行定义.

2.20.2.9 IRQEna

uint8_t IRQEna

SPI中断使能：DISABLE：关闭所以中断

参见

[SPI_IRQEna_Enable](#)

[SPI_IRQEna_TranDone](#)

[SPI_IRQEna_SSErr](#)

[SPI_IRQEna_DataOver](#)

在文件 [lks32mc08x_spi.h](#) 第 86 行定义.

2.20.2.10 Mode

[SPI_Mode](#) Mode

主从模式选择：0，从模式；1，主模式

参见

[SPI_Mode](#)

在文件 [lks32mc08x_spi.h](#) 第 74 行定义.

2.20.2.11 SS

uint8_t SS

从设备下片选信号来源：0，恒为0；1，来源于主设备

在文件 [lks32mc08x_spi.h](#) 第 69 行定义.

2.20.2.12 Trig

uint8_t Trig

传输触发选择：0，内部自动执行（仅主模式有效）；1，外部触发

在文件 [lks32mc08x_spi.h](#) 第 87 行定义.

2.21 SPI_TypeDef结构体 参考

SPI寄存器结构体句柄

```
#include <lks32mc08x_spi.h>
```

成员变量

- __IO uint32_t [CFG](#)
SPI控制寄存器
- __IO uint32_t [IE](#)
SPI中断寄存器
- __IO uint32_t [DIV](#)
SPI波特率控制寄存器
- __IO uint32_t [TX_DATA](#)
SPI发送数据寄存器
- __IO uint32_t [RX_DATA](#)
SPI接收数据寄存器
- __IO uint32_t [SIZE](#)
SPI传输数据长度寄存器

2.21.1 详细描述

SPI寄存器结构体句柄

在文件 [lks32mc08x_spi.h](#) 第 42 行定义.

2.21.2 结构体成员变量说明

2.21.2.1 CFG

__IO uint32_t CFG

SPI控制寄存器

在文件 [lks32mc08x_spi.h](#) 第 44 行定义.

2.21.2.2 DIV

```
--IO uint32_t DIV
```

SPI波特率控制寄存器

在文件 [lks32mc08x_spi.h](#) 第 46 行定义.

2.21.2.3 IE

```
--IO uint32_t IE
```

SPI中断寄存器

在文件 [lks32mc08x_spi.h](#) 第 45 行定义.

2.21.2.4 RX_DATA

```
--IO uint32_t RX_DATA
```

SPI接收数据寄存器

在文件 [lks32mc08x_spi.h](#) 第 48 行定义.

2.21.2.5 SIZE

```
--IO uint32_t SIZE
```

SPI传输数据长度寄存器

在文件 [lks32mc08x_spi.h](#) 第 49 行定义.

2.21.2.6 TX_DATA

```
--IO uint32_t TX_DATA
```

SPI发送数据寄存器

在文件 [lks32mc08x_spi.h](#) 第 47 行定义.

2.22 **stru_CordicComponents**结构体 参考

三角函数运算结构体句柄

```
#include <lks32mc08x_dsp.h>
```


成员变量

- s16 **Theta**
角度-32768~32767对应 $-(\pi) \sim \pi$
- s16 **SinValue**
计算得到sin值
- s16 **CosValue**
计算得到cos值
- s16 **ArctanX**
Arctan计算对应X值 $\arctan = Y/X$
- s16 **ArctanY**
Arctan计算对应Y值 $\arctan = Y/X$
- s16 **ArctanValue**
计算得到arctan值

2.22.1 详细描述

三角函数运算结构体句柄

在文件 [lks32mc08x_dsp.h](#) 第 35 行定义.

2.22.2 结构体成员变量说明

2.22.2.1 ArctanValue

s16 ArctanValue

计算得到arctan值

在文件 [lks32mc08x_dsp.h](#) 第 42 行定义.

2.22.2.2 ArctanX

s16 ArctanX

Arctan计算对应X值 $\arctan = Y/X$

在文件 [lks32mc08x_dsp.h](#) 第 40 行定义.

2.22.2.3 ArctanY

s16 ArctanY

Arctan计算对应Y值 $\arctan = Y/X$

在文件 [lks32mc08x_dsp.h](#) 第 41 行定义.

2.22.2.4 CosValue

s16 CosValue

计算得到cos值

在文件 [lks32mc08x_dsp.h](#) 第 39 行定义.

2.22.2.5 SinValue

s16 SinValue

计算得到sin值

在文件 [lks32mc08x_dsp.h](#) 第 38 行定义.

2.22.2.6 Theta

s16 Theta

角度-32768~32767对应 $-(\pi) \sim \pi$

在文件 [lks32mc08x_dsp.h](#) 第 37 行定义.

2.23 **stru_DiviComponents**结构体 参考

除法运算结构体句柄

```
#include <lks32mc08x_dsp.h>
```

成员变量

- s32 [Dividend](#)
被除数
- s16 [Divisor](#)
除数
- s32 [Quotient](#)
商
- s16 [Remainder](#)
余数

2.23.1 详细描述

除法运算结构体句柄

在文件 [lks32mc08x_dsp.h](#) 第 24 行定义.

2.23.2 结构体成员变量说明

2.23.2.1 Dividend

s32 Dividend

被除数

在文件 [lks32mc08x_dsp.h](#) 第 26 行定义.

2.23.2.2 Divisor

s16 Divisor

除数

在文件 [lks32mc08x_dsp.h](#) 第 27 行定义.

2.23.2.3 Quotient

s32 Quotient

商

在文件 [lks32mc08x_dsp.h](#) 第 28 行定义.

2.23.2.4 Remainder

s16 Remainder

余数

在文件 [lks32mc08x_dsp.h](#) 第 29 行定义.

2.24 Stru_TempertureCof_Def结构体 参考

芯片内置温度传感器初始化结构体句柄，初始化时请定义该句柄，并用其它地址来传参

```
#include <lks32mc08x_tmp.h>
```

成员变量

- **u16 nCofA**
温度系数A，来源`Read.Trim(0x00000398)`输出值
- **u16 nOffsetB**
温度系数偏置，来源`Read.Trim(0x0000039C)`输出值

2.24.1 详细描述

芯片内置温度传感器初始化结构体句柄，初始化时请定义该句柄，并用其它地址来传参

在文件 `lks32mc08x_tmp.h` 第 22 行定义.

2.24.2 结构体成员变量说明

2.24.2.1 nCofA

`u16 nCofA`

温度系数A，来源`Read.Trim(0x00000398)`输出值

在文件 `lks32mc08x_tmp.h` 第 24 行定义.

2.24.2.2 nOffsetB

`u16 nOffsetB`

温度系数偏置，来源`Read.Trim(0x0000039C)`输出值

在文件 `lks32mc08x_tmp.h` 第 25 行定义.

2.25 SYS_InitTypeDef结构体 参考

系统初始化结构体句柄，初始化时请定义该句柄，并用其它地址来传参

```
#include <lks32mc08x-sys.h>
```

成员变量

- uint32_t [PLL_SrcSel](#)
PLL信号输入源选择, RC时钟或外部晶体
- uint32_t [PLL_DivSel](#)
选择8个时钟周期中, 哪个周期输出时钟
- uint32_t [Clk_Sel](#)
系统时钟选择, 内部RC时钟或外部晶体时钟
- uint32_t [Clk_DivSPI](#)
I2C时钟分频
- uint32_t [Clk_DivUART](#)
UART时钟分频
- uint32_t [Clk_FEN](#)
模块时钟使能
- uint32_t [WDT_Ena](#)
使能看门狗
- uint32_t [PORFilter_Ena](#)
使能POR滤波
- uint32_t [PLL_ReDiv](#)
以弃用

2.25.1 详细描述

系统初始化结构体句柄, 初始化时请定义该句柄, 并用其它地址来传参
在文件 [lks32mc08x_sys.h](#) 第 41 行定义.

2.25.2 结构体成员变量说明

2.25.2.1 Clk_DivSPI

uint32_t Clk_DivSPI

I2C时钟分频

在文件 [lks32mc08x_sys.h](#) 第 48 行定义.

2.25.2.2 Clk_DivUART

uint32_t Clk_DivUART

UART时钟分频

在文件 [lks32mc08x_sys.h](#) 第 49 行定义.

2.25.2.3 Clk_FEN

```
uint32_t Clk_FEN
```

模块时钟使能

在文件 [lks32mc08x_sys.h](#) 第 50 行定义.

2.25.2.4 Clk_Sel

```
uint32_t Clk_Sel
```

系统时钟选择, 内部RC时钟或外部晶体时钟

在文件 [lks32mc08x_sys.h](#) 第 46 行定义.

2.25.2.5 PLL_DivSel

```
uint32_t PLL_DivSel
```

选择8个时钟周期中, 哪个周期输出时钟

在文件 [lks32mc08x_sys.h](#) 第 45 行定义.

2.25.2.6 PLL_ReDiv

```
uint32_t PLL_ReDiv
```

以弃用

在文件 [lks32mc08x_sys.h](#) 第 55 行定义.

2.25.2.7 PLL_SrcSel

```
uint32_t PLL_SrcSel
```

PLL信号输入源选择, RC时钟或外部晶体

参见

[SYS.PLLSRSEL.RCH](#)

[SYS.PLLSRSEL.CRYSTAL](#)

在文件 [lks32mc08x_sys.h](#) 第 43 行定义.

2.25.2.8 PORFilter_Ena

```
uint32_t PORFilter_Ena
```

使能POR滤波

在文件 [lks32mc08x_sys.h](#) 第 53 行定义.

2.25.2.9 WDT_Ena

```
uint32_t WDT_Ena
```

使能看门狗

在文件 [lks32mc08x_sys.h](#) 第 52 行定义.

2.26 TIM_ECDInitTypeDef结构体 参考

```
#include <lks32mc08x_tim.h>
```

成员变量

- [uint32_t ECD.Mode](#)
编码器模式选择
- [uint32_t ECD.TH](#)
编码器计数门限值
- [uint32_t ECD.ClockDiv](#)
编码器对应UTIMER时钟分频系数
- [uint32_t ECD.Filter0](#)
编码器0滤波分频0:不滤波 N:滤波采用的时钟是系统时钟N分频
- [uint32_t ECD.Filter1](#)
编码器1滤波分频0:不滤波 N:滤波采用的时钟是系统时钟N分频
- [uint32_t ECD.IRQEna](#)
编码器中断使能

2.26.1 详细描述

在文件 [lks32mc08x_tim.h](#) 第 184 行定义.

2.26.2 结构体成员变量说明

2.26.2.1 ECD_ClockDiv

uint32_t ECD_ClockDiv

编码器对应UTIMER时钟分频系数

参见

[ECD_Clk_Div1](#)

[ECD_Clk_Div2](#)

[ECD_Clk_Div4](#)

[ECD_Clk_Div8](#)

在文件 [lks32mc08x_tim.h](#) 第 204 行定义.

2.26.2.2 ECD_Filter0

uint32_t ECD_Filter0

编码器0滤波分频0:不滤波 N:滤波采用的时钟是系统时钟N分频

在文件 [lks32mc08x_tim.h](#) 第 206 行定义.

2.26.2.3 ECD_Filter1

uint32_t ECD_Filter1

编码器1滤波分频0:不滤波 N:滤波采用的时钟是系统时钟N分频

在文件 [lks32mc08x_tim.h](#) 第 207 行定义.

2.26.2.4 ECD_IRQEna

uint32_t ECD_IRQEna

编码器中断使能

参见

[ECD_IRQEna.None](#)

[ECD_IRQEna.UnderFlow](#)

[ECD_IRQEna.Overflow](#)

在文件 [lks32mc08x_tim.h](#) 第 214 行定义.

2.26.2.5 ECD_Mode

uint32_t ECD_Mode

编码器模式选择

参见

[ECD_Mode_T1](#)

[ECD_Mode_T1_T2](#)

[ECD_Mode_CCWSIGN](#)

[ECD_Mode_CCWCW](#)

[ECD_Mode_CCWSIGN_UpDown](#)

[ECD_Mode_CCWCW_UpDown](#)

在文件 [lks32mc08x_tim.h](#) 第 195 行定义.

2.26.2.6 ECD_TH

uint32_t ECD_TH

编码器计数门限值

在文件 [lks32mc08x_tim.h](#) 第 196 行定义.

2.27 TIM_ECDTypeDef结构体 参考

编码器寄存器结构体句柄

```
#include <lks32mc08x_tim.h>
```

成员变量

- __IO uint32_t [CFG](#)
编码器配置寄存器
- __IO uint32_t [TH](#)
编码器计数门限寄存器
- __IO uint32_t [CNT](#)
编码器计数值寄存器

2.27.1 详细描述

编码器寄存器结构体句柄

在文件 [lks32mc08x_tim.h](#) 第 36 行定义.

2.27.2 结构体成员变量说明

2.27.2.1 CFG

```
--IO uint32_t CFG
```

编码器配置寄存器

在文件 [lks32mc08x_tim.h](#) 第 38 行定义.

2.27.2.2 CNT

```
--IO uint32_t CNT
```

编码器计数值寄存器

在文件 [lks32mc08x_tim.h](#) 第 40 行定义.

2.27.2.3 TH

```
--IO uint32_t TH
```

编码器计数门限寄存器

在文件 [lks32mc08x_tim.h](#) 第 39 行定义.

2.28 TIM_TimerInitTypeDef结构体 参考

```
#include <lks32mc08x_tim.h>
```

成员变量

- FuncState EN
- uint32_t Timer_CH0_WorkMode
channel0工作模式:捕获模式, 比较模式
- uint32_t Timer_CH0_CapMode
channel0捕获模式:上升沿、下降沿、双沿
- uint32_t Timer_CH0Output
计数器回零时, 比较模式IO输出 1: 高电平 0: 低电平
- uint32_t Timer_CH1_WorkMode
channel1工作模式:捕获模式, 比较模式
- uint32_t Timer_CH1_CapMode
channel1捕获模式:上升沿、下降沿、双沿
- uint32_t Timer_CH1Output
计数器回零时, 比较模式IO输出 1: 高电平 0: 低电平
- uint32_t Timer_TH
定时器计数门限
- uint32_t Timer_CMP0
CH0比较计数器值
- uint32_t Timer_CMP1
CH1比较计数器值
- uint32_t Timer_Filter0
CH0滤波分频, 0:不滤波 N:滤波采用的时钟是系统时钟N分频
- uint32_t Timer_Filter1
CH1滤波分频 0:不滤波 N:滤波采用的时钟是系统时钟N分频
- uint32_t Timer_ClockDiv
定时器分频
- uint32_t Timer_IRQEna
定时器中断使能

2.28.1 详细描述

在文件 [lks32mc08x.tim.h](#) 第 128 行定义.

2.28.2 结构体成员变量说明

2.28.2.1 EN

FuncState EN

在文件 [lks32mc08x.tim.h](#) 第 130 行定义.

2.28.2.2 Timer_CH0_CapMode

uint32_t Timer_CH0_CapMode

channel0捕获模式:上升沿、下降沿、双沿

参见

[TIMER_CapMode_None](#)

[TIMER_CapMode_Rise](#)

[TIMER_CapMode_Fall](#)

[TIMER_CapMode_RiseFall](#)

在文件 [lks32mc08x_tim.h](#) 第 144 行定义.

2.28.2.3 Timer_CH0_WorkMode

uint32_t Timer_CH0_WorkMode

channel0工作模式:捕获模式, 比较模式

参见

[TIMER_OPMODE_CAP](#)

[TIMER_OPMODE_CMP](#)

在文件 [lks32mc08x_tim.h](#) 第 136 行定义.

2.28.2.4 Timer_CH0Output

uint32_t Timer_CH0Output

计数器回零时, 比较模式IO输出 1: 高电平 0: 低电平

在文件 [lks32mc08x_tim.h](#) 第 145 行定义.

2.28.2.5 Timer_CH1_CapMode

uint32_t Timer_CH1_CapMode

channel0捕获模式:上升沿、下降沿、双沿

参见

[TIMER_CapMode_None](#)

[TIMER_CapMode_Rise](#)

[TIMER_CapMode_Fall](#)

[TIMER_CapMode_RiseFall](#)

在文件 [lks32mc08x_tim.h](#) 第 159 行定义.

2.28.2.6 Timer_CH1_WorkMode

uint32_t Timer_CH1_WorkMode

channel0工作模式:捕获模式, 比较模式

参见

[TIMER_OPMODE_CAP](#)

[TIMER_OPMODE_CMP](#)

在文件 [lks32mc08x_tim.h](#) 第 151 行定义.

2.28.2.7 Timer_CH1Output

uint32_t Timer_CH1Output

计数器回零时, 比较模式IO输出 1: 高电平 0: 低电平

在文件 [lks32mc08x_tim.h](#) 第 160 行定义.

2.28.2.8 Timer_ClockDiv

uint32_t Timer_ClockDiv

定时器分频

参见

[TIM_Clk_Div1](#)

[TIM_Clk_Div2](#)

[TIM_Clk_Div4](#)

[TIM_Clk_Div8](#)

在文件 [lks32mc08x_tim.h](#) 第 176 行定义.

2.28.2.9 Timer_CMP0

uint32_t Timer_CMP0

CH0比较计数器值

在文件 [lks32mc08x_tim.h](#) 第 163 行定义.

2.28.2.10 Timer_CMP1

uint32_t Timer_CMP1

CH1比较计数器值

在文件 [lks32mc08x_tim.h](#) 第 164 行定义.

2.28.2.11 Timer_Filter0

uint32_t Timer_Filter0

CH0滤波分频, 0:不滤波 N:滤波采用的时钟是系统时钟N分频

在文件 [lks32mc08x_tim.h](#) 第 166 行定义.

2.28.2.12 Timer_Filter1

uint32_t Timer_Filter1

CH1滤波分频 0:不滤波 N:滤波采用的时钟是系统时钟N分频

在文件 [lks32mc08x_tim.h](#) 第 167 行定义.

2.28.2.13 Timer_IRQEna

uint32_t Timer_IRQEna

定时器中断使能

参见

[Timer_IRQEnaDef](#)

在文件 [lks32mc08x_tim.h](#) 第 181 行定义.

2.28.2.14 Timer_TH

uint32_t Timer_TH

定时器计数门限

在文件 [lks32mc08x_tim.h](#) 第 162 行定义.

2.29 TIM_TimerTypeDef结构体 参考

定时器寄存器结构体句柄

```
#include <lks32mc08x_tim.h>
```

成员变量

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

2.29.1 详细描述

定时器寄存器结构体句柄

在文件 [lks32mc08x_tim.h](#) 第 23 行定义.

2.29.2 结构体成员变量说明

2.29.2.1 CFG

```
--IO uint32_t CFG
```

定时器配置寄存器

在文件 [lks32mc08x_tim.h](#) 第 25 行定义.

2.29.2.2 CMPT0

```
--IO uint32_t CMPT0
```

定时器比较/捕获寄存器 0

在文件 [lks32mc08x_tim.h](#) 第 28 行定义.

2.29.2.3 CMPT1

```
--IO uint32_t CMPT1
```

定时器比较/捕获寄存器 1

在文件 [lks32mc08x_tim.h](#) 第 29 行定义.

2.29.2.4 CNT

```
--IO uint32_t CNT
```

定时器计数值寄存器

在文件 [lks32mc08x_tim.h](#) 第 27 行定义.

2.29.2.5 EVT

```
--IO uint32_t EVT
```

定时器外部事件选择寄存器

在文件 [lks32mc08x_tim.h](#) 第 30 行定义.

2.29.2.6 TH

```
--IO uint32_t TH
```

定时器计数门限寄存器

在文件 [lks32mc08x_tim.h](#) 第 26 行定义.

2.30 UART_InitTypeDef结构体 参考

UART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 <lks32mc08x-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 改为一般不在初始化内使用
- uint16_t **Match485Addr**
用作485通信时的匹配地址
- uint8_t **IRQEna**
中断使能寄存器
- uint8_t **RXD_INV**
接收电平取反 ENABLE取反, DISABLE正常
- uint8_t **TXD_INV**
发送电平取反 ENABLE取反, DISABLE正常

2.30.1 详细描述

UART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 [lks32mc08x-uart.h](#) 第 47 行定义.

2.30.2 结构体成员变量说明

2.30.2.1 BaudRate

```
uint32_t BaudRate
```

波特率

在文件 [lks32mc08x-uart.h](#) 第 49 行定义.

2.30.2.2 Bit9Value

uint8_t Bit9Value

Multi-drop Master模式时,第9个数据位值 改为一般不在初始化内使用

在文件 [lks32mc08x_uart.h](#) 第 56 行定义.

2.30.2.3 FirstSend

uint8_t FirstSend

发送数据大小端设置

参见

[UART_FIRSTSEND_LSB](#)

[UART_FIRSTSEND_MSB](#)

在文件 [lks32mc08x_uart.h](#) 第 52 行定义.

2.30.2.4 IRQEna

uint8_t IRQEna

中断使能寄存器

参见

[UART_IRQEna_SendOver](#)

[UART_IRQEna_RcvOver](#)

[UART_IRQEna_SendBuffEmpty](#)

[UART_IRQEna_StopError](#)

[UART_IRQEna_CheckError](#)

在文件 [lks32mc08x_uart.h](#) 第 66 行定义.

2.30.2.5 Match485Addr

uint16_t Match485Addr

用作485通信时的匹配地址

在文件 [lks32mc08x_uart.h](#) 第 58 行定义.

2.30.2.6 MultiDropEna

uint8_t MultiDropEna

使能Multi-drop,0:Disable 1:Enable

在文件 [lks32mc08x_uart.h](#) 第 55 行定义.

2.30.2.7 ParityMode

UART_ParityMode ParityMode

奇偶校验,

参见

[UART_ParityMode](#)

在文件 [lks32mc08x_uart.h](#) 第 53 行定义.

2.30.2.8 RXD_INV

uint8_t RXD_INV

接收电平取反 ENABLE取反, DISABLE正常

在文件 [lks32mc08x_uart.h](#) 第 67 行定义.

2.30.2.9 StopBits

uint8_t StopBits

停止位长度

参见

[UART_STOPBITS_1b](#)

[UART_STOPBITS_2b](#)

在文件 [lks32mc08x_uart.h](#) 第 51 行定义.

2.30.2.10 TXD_INV

uint8_t TXD_INV

发送电平取反 ENABLE取反, DISABLE正常

在文件 [lks32mc08x_uart.h](#) 第 68 行定义.

2.30.2.11 WordLength

uint8_t WordLength

数据长度

参见

[UART_WORDLENGTH_8b](#)

[UART_WORDLENGTH_7b](#)

在文件 [lks32mc08x_uart.h](#) 第 50 行定义.

2.31 UART_TypeDef结构体 参考

UART寄存器结构体句柄

```
#include <lks32mc08x_uart.h>
```

成员变量

- __IO uint32_t [CTRL](#)
UART控制寄存器
- __IO uint32_t [DIVH](#)
UART波特率设置高字节寄存器
- __IO uint32_t [DIVL](#)
UART波特率设置低字节寄存器
- __IO uint32_t [BUFF](#)
UART收发缓冲寄存器
- __IO uint32_t [ADR](#)
485通信地址匹配寄存器
- __IO uint32_t [STT](#)
UART状态寄存器
- __IO uint32_t [IE](#)
UART中断使能寄存器
- __IO uint32_t [IF](#)
UART中断标志寄存器
- __IO uint32_t [INV](#)
UART IO翻转使能

2.31.1 详细描述

UART寄存器结构体句柄

在文件 [lks32mc08x_uart.h](#) 第 31 行定义.

2.31.2 结构体成员变量说明

2.31.2.1 ADR

```
--IO uint32_t ADR
```

485通信地址匹配寄存器

在文件 [lks32mc08x_uart.h](#) 第 37 行定义.

2.31.2.2 BUFF

```
--IO uint32_t BUFF
```

UART收发缓冲寄存器

在文件 [lks32mc08x_uart.h](#) 第 36 行定义.

2.31.2.3 CTRL

```
--IO uint32_t CTRL
```

UART控制寄存器

在文件 [lks32mc08x_uart.h](#) 第 33 行定义.

2.31.2.4 DIVH

```
--IO uint32_t DIVH
```

UART波特率设置高字节寄存器

在文件 [lks32mc08x_uart.h](#) 第 34 行定义.

2.31.2.5 DIVL

```
--IO uint32_t DIVL
```

UART波特率设置低字节寄存器

在文件 [lks32mc08x_uart.h](#) 第 35 行定义.

2.31.2.6 IE

```
--IO uint32_t IE
```

UART中断使能寄存器

在文件 [lks32mc08x_uart.h](#) 第 39 行定义.

2.31.2.7 IF

`__IO uint32_t IF`

UART中断标志寄存器

在文件 [lks32mc08x_uart.h](#) 第 40 行定义.

2.31.2.8 INV

`__IO uint32_t INV`

UART IO翻转使能

在文件 [lks32mc08x_uart.h](#) 第 41 行定义.

2.31.2.9 STT

`__IO uint32_t STT`

UART状态寄存器

在文件 [lks32mc08x_uart.h](#) 第 38 行定义.

2.32 WAKE_InitTypeDef结构体 参考

低功耗时钟配置中间变量

`#include <lks32mc08x_sys.h>`

成员变量

- `uint32_t AFE_REG5_RECORD`
- `uint32_t AFE_REG6_RECORD`
- `uint32_t CLK_CFG_RECORD`

2.32.1 详细描述

低功耗时钟配置中间变量

在文件 [lks32mc08x_sys.h](#) 第 31 行定义.

2.32.2 结构体成员变量说明

2.32.2.1 AFE_REG5_RECORD

`uint32_t AFE_REG5_RECORD`

在文件 [lks32mc08x_sys.h](#) 第 33 行定义.

2.32.2.2 AFE_REG6_RECORD

`uint32_t AFE_REG6_RECORD`

在文件 [lks32mc08x_sys.h](#) 第 34 行定义.

2.32.2.3 CLK_CFG_RECORD

`uint32_t CLK_CFG_RECORD`

在文件 [lks32mc08x_sys.h](#) 第 35 行定义.

文件说明

文件名称: lks32mc08x_adc.h
文件标识: 无
内容摘要: ADC驱动头文件
其它说明: 无

The diagram illustrates a hierarchical tree structure for sample classification. At the top is the root node "All Samples". It splits into two main categories: "Non-Infected" and "Infected (COVID-19)". Under "Non-Infected", there are two sub-categories: "Healthy" and "Infected (Other Diseases)". Under "Infected (COVID-19)", there are two sub-categories: "Mild" and "Severe". Each of these four categories further branches down into numerous individual species names, such as "Homo sapiens", "Mus musculus", "Rattus norvegicus", etc., representing the detailed classification of the 1000 samples.

- struct [ADC_TypeDef](#)
ADC寄存器结构体句柄
- struct [ADC_InitTypeDef](#)
ADC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 **ADC0** ((ADC_TypeDef *)ADC0_BASE)
ADC结构体基地址定义
- #define **ADC_Clock_48M** (0x00) /*48MHz*/
- #define **ADC_Clock_24M** (0x03) /*24MHz*/
- #define **ADC_Clock_12M** (0x02) /*12MHz*/
- #define **RefVol_2V4** ((uint16_t)0x00) /*ADC基准电压2.4V*/
- #define **RefVol_1V2** ((uint16_t)0x01) /*ADC基准电压1.2V*/
- #define **ADC_1_TIMES_SAMPLE** ((u8)1)
ADC采样通道数1次采样
- #define **ADC_2_TIMES_SAMPLE** ((u8)2)
ADC采样通道数2次采样
- #define **ADC_3_TIMES_SAMPLE** ((u8)3)
ADC采样通道数3次采样
- #define **ADC_4_TIMES_SAMPLE** ((u8)4)
ADC采样通道数4次采样
- #define **ADC_5_TIMES_SAMPLE** ((u8)5)
ADC采样通道数5次采样
- #define **ADC_6_TIMES_SAMPLE** ((u8)6)
ADC采样通道数6次采样
- #define **ADC_7_TIMES_SAMPLE** ((u8)7)
ADC采样通道数7次采样
- #define **ADC_8_TIMES_SAMPLE** ((u8)8)
ADC采样通道数8次采样
- #define **ADC_9_TIMES_SAMPLE** ((u8)9)
ADC采样通道数9次采样
- #define **ADC_10_TIMES_SAMPLE** ((u8)10)
ADC采样通道数10次采样
- #define **ADC_11_TIMES_SAMPLE** ((u8)11)
ADC采样通道数11次采样
- #define **ADC_12_TIMES_SAMPLE** ((u8)12)
ADC采样通道数12次采样
- #define **ADC_13_TIMES_SAMPLE** ((u8)13)
ADC采样通道数13次采样
- #define **ADC_14_TIMES_SAMPLE** ((u8)14)
ADC采样通道数14次采样
- #define **ADC_15_TIMES_SAMPLE** ((u8)15)
ADC采样通道数15次采样
- #define **ADC_16_TIMES_SAMPLE** ((u8)16)
ADC采样通道数16次采样
- #define **ADC_17_TIMES_SAMPLE** ((u8)17)
ADC采样通道数17次采样
- #define **ADC_18_TIMES_SAMPLE** ((u8)18)
ADC采样通道数18次采样
- #define **ADC_19_TIMES_SAMPLE** ((u8)19)
ADC采样通道数19次采样
- #define **ADC_20_TIMES_SAMPLE** ((u8)20)
ADC采样通道数20次采样
- #define **ADC_DAT0_OV_IRQ_EN** ((uint16_t)0x40)
ADC0.DAT0超阈值中断

- **#define ADC_H_CONFLICT_IRQ_EN** ((uint16_t)0x20)
硬件触发冲突
- **#define ADC_S_CONFLICT_IRQ_EN** ((uint16_t)0x10)
软件触发冲突
- **#define ADC_EOS3_IRQ_EN** ((uint16_t)0x08)
第四段扫描结束中断
- **#define ADC_EOS2_IRQ_EN** ((uint16_t)0x04)
第三段扫描结束中断
- **#define ADC_EOS1_IRQ_EN** ((uint16_t)0x02)
第二段扫描结束中断
- **#define ADC_EOS0_IRQ_EN** ((uint16_t)0x01)
第一段扫描结束中断
- **#define ADC_DAT0_OV_IRQ_IF** ((uint16_t)0x40)
ADC_DAT0超阈值中断标志位
- **#define ADC_H_CONFLICT_IRQ_IF** ((uint16_t)0x20)
硬件触发冲突标志
- **#define ADC_S_CONFLICT_IRQ_IF** ((uint16_t)0x10)
软件触发冲突标志
- **#define ADC_EOS3_IRQ_IF** ((uint16_t)0x08)
第四段扫描结束中断标志
- **#define ADC_EOS2_IRQ_IF** ((uint16_t)0x04)
第三段扫描结束中断标志
- **#define ADC_EOS1_IRQ_IF** ((uint16_t)0x02)
第二段扫描结束中断标志
- **#define ADC_EOS0_IRQ_IF** ((uint16_t)0x01)
第一段扫描结束中断标志
- **#define ADC_ALL_IRQ_IF** ((uint16_t)0x7f)
ADC全部中断标志位——可用于初始化
- **#define ADC_LEFT_ALIGN** ((uint16_t)0x0000)
ADC数据输出左对齐
- **#define ADC_RIGHT_ALIGN** ((uint16_t)0x0001)
ADC数据输出右对齐
- **#define ADC_DAT0_HTH** ((uint16_t)0x0001)
ADC_DAT0.TH作为上阈值
- **#define ADC_DAT0_LTH** ((uint16_t)0x0000)
ADC_DAT0.TH作为下阈值
- **#define ADC_MCPWM_T0_TRG** ((uint16_t)0x01)
ADC采用MCPWM T0事件触发
- **#define ADC_MCPWM_T1_TRG** ((uint16_t)0x02)
ADC采用MCPWM T1事件触发
- **#define ADC_MCPWM_T2_TRG** ((uint16_t)0x04)
ADC采用MCPWM T2事件触发
- **#define ADC_MCPWM_T3_TRG** ((uint16_t)0x08)
ADC采用MCPWM T3事件触发
- **#define ADC_UTIMER_T0_TRG** ((uint16_t)0x01)
ADC采用UTIMER T0事件触发
- **#define ADC_UTIMER_T1_TRG** ((uint16_t)0x02)
ADC采用UTIMER T1事件触发
- **#define ADC_UTIMER_T2_TRG** ((uint16_t)0x04)
ADC采用UTIMER T2事件触发
- **#define ADC_UTIMER_T3_TRG** ((uint16_t)0x08)

- ADC采用*UTIMER T3*事件触发
- #define **ADC_1SEG_TRG** ((uint16_t)0x00)
ADC采用单段模式
- #define **ADC_2SEG_TRG** ((uint16_t)0x01)
ADC采用2段模式
- #define **ADC_4SEG_TRG** ((uint16_t)0x03)
ADC采用4段模式
- #define **ADC_CHANNEL_0** ((uint16_t)0x00)
ADC通道0-OPA0_OUT
- #define **ADC_CHANNEL_1** ((uint16_t)0x01)
ADC通道1-OPA1_OUT
- #define **ADC_CHANNEL_2** ((uint16_t)0x02)
ADC通道2-OPA2_OUT
- #define **ADC_CHANNEL_3** ((uint16_t)0x03)
ADC通道3-OPA3_OUT
- #define **ADC_CHANNEL_4** ((uint16_t)0x04)
ADC通道4
- #define **ADC_CHANNEL_5** ((uint16_t)0x05)
ADC通道5
- #define **ADC_CHANNEL_6** ((uint16_t)0x06)
ADC通道6
- #define **ADC_CHANNEL_7** ((uint16_t)0x07)
ADC通道7
- #define **ADC_CHANNEL_8** ((uint16_t)0x08)
ADC通道8
- #define **ADC_CHANNEL_9** ((uint16_t)0x09)
ADC通道9
- #define **ADC_CHANNEL_10** ((uint16_t)0x0a)
ADC通道10
- #define **ADC_CHANNEL_11** ((uint16_t)0x0b)
ADC通道11
- #define **ADC_CHANNEL_12** ((uint16_t)0x0c)
ADC通道12
- #define **ADC_CHANNEL_13** ((uint16_t)0x0d)
ADC通道13
- #define **ADC_CHANNEL_14** ((uint16_t)0x0e)
ADC通道14
- #define **ADC_CHANNEL_15** ((uint16_t)0x0f)
ADC通道15
- #define **ADC_CHANNEL_16** ((uint16_t)0x10)
ADC通道16
- #define **ADC_CHANNEL_17** ((uint16_t)0x11)
ADC通道17
- #define **ADC_CHANNEL_18** ((uint16_t)0x12)
ADC通道18-Temp
- #define **ADC_CHANNEL_19** ((uint16_t)0x13)
ADC通道19-VSS
- #define **ADC_HARDWARE_T0_TRG** ((uint16_t)0x01)
ADC采用硬件*T0*事件触发
- #define **ADC_HARDWARE_T1_TRG** ((uint16_t)0x02)
ADC采用硬件*T1*事件触发

- `#define ADC_HARDWARE_T2_TRG ((uint16_t)0x04)`
ADC采用硬件T2事件触发
- `#define ADC_HARDWARE_T3_TRG ((uint16_t)0x08)`
ADC采用硬件T3事件触发

枚举

- `enum CHNx {`
 `CHN0 = 0, CHN1, CHN2, CHN3,`
 `CHN4, CHN5, CHN6, CHN7,`
 `CHN8, CHN9, CHN10, CHN11,`
 `CHN12, CHN13, CHN14, CHN15,`
 `CHN16, CHN17, CHN18, CHN19 }`
ADC采样信号来源寄存器序号
- `enum DATx {`
 `DAT0 = 0, DAT1, DAT2, DAT3,`
 `DAT4, DAT5, DAT6, DAT7,`
 `DAT8, DAT9, DAT10, DAT11,`
 `DAT12, DAT13, DAT14, DAT15,`
 `DAT16, DAT17, DAT18, DAT19 }`
ADC数据寄存器序号

函数

- `void ADC_Init (ADC_TypeDef *ADCx, ADC_InitTypeDef *ADC_InitStruct)`
函数名称: `void ADC_Init(ADC_TypeDef* ADCx, ADC_InitTypeDef* ADC_InitStruct)`
- `void ADC_StructInit (ADC_InitTypeDef *ADC_InitStruct)`
函数名称: `void ADC_StructInit(ADC_InitTypeDef* ADC_InitStruct)`
- `void ADC_ClearIRQFlag (ADC_TypeDef *ADCx, uint16_t INT_flag)`
函数名称: `void ADC_ClearIRQFlag(ADC_TypeDef* ADCx, uint16_t INT_flag)`
- `void ADC_ChannelConfig (ADC_TypeDef *ADCx, CHNx CHNum, uint16_t nChannel)`
函数名称: `void ADC_ChannelConfig(ADC_TypeDef* ADCx, CHNx CHNum, uint16_t Channel_0, uint16_t Channel_1)`
- `uint16_t ADC_GetIRQFlag (ADC_TypeDef *ADCx, uint16_t INT_flag)`
函数名称: `uint16_t ADC_GetIRQFlag(ADC_TypeDef* ADCx, uint16_t INT_flag)`
- `s16 ADC_GetConversionValue (DATx DATNum)`
函数名称: `s16 ADC_GetConversionValue(DATx DATNum)`
- `void ADC_SoftTrgEN (ADC_TypeDef *ADCx, FuncState state)`
函数名称: `void ADC_SoftTrgEN(ADC_TypeDef* ADCx, FuncState state)`

3.1.1 详细描述

文件名称: `lks32mc08x_adc.h`
 文件标识: 无
 内容摘要: ADC驱动头文件
 其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2018年07月25日	1.0	Howlet	创建

修改日志:

在文件 [lks32mc08x_adc.h](#) 中定义.

3.1.2 宏定义说明

3.1.2.1 ADC0

```
#define ADC0 ((ADC_TypeDef *)ADC0_BASE)
```

ADC结构体基地址定义

在文件 [lks32mc08x_adc.h](#) 第 84 行定义.

3.1.2.2 ADC_10_TIMES_SAMPLE

```
#define ADC_10_TIMES_SAMPLE ((u8)10)
```

ADC采样通道数10次采样

在文件 [lks32mc08x_adc.h](#) 第 188 行定义.

3.1.2.3 ADC_11_TIMES_SAMPLE

```
#define ADC_11_TIMES_SAMPLE ((u8)11)
```

ADC采样通道数11次采样

在文件 [lks32mc08x_adc.h](#) 第 189 行定义.

3.1.2.4 ADC_12_TIMES_SAMPLE

```
#define ADC_12_TIMES_SAMPLE ((u8)12)
```

ADC采样通道数12次采样

在文件 [lks32mc08x_adc.h](#) 第 190 行定义.

3.1.2.5 ADC_13_TIMES_SAMPLE

```
#define ADC_13_TIMES_SAMPLE ((u8)13)
```

ADC采样通道数13次采样

在文件 [lks32mc08x_adc.h](#) 第 191 行定义.

3.1.2.6 ADC_14_TIMES_SAMPLE

```
#define ADC_14_TIMES_SAMPLE ((u8)14)
```

ADC采样通道数14次采样

在文件 [lks32mc08x_adc.h](#) 第 192 行定义.

3.1.2.7 ADC_15_TIMES_SAMPLE

```
#define ADC_15_TIMES_SAMPLE ((u8)15)
```

ADC采样通道数15次采样

在文件 [lks32mc08x_adc.h](#) 第 193 行定义.

3.1.2.8 ADC_16_TIMES_SAMPLE

```
#define ADC_16_TIMES_SAMPLE ((u8)16)
```

ADC采样通道数16次采样

在文件 [lks32mc08x_adc.h](#) 第 194 行定义.

3.1.2.9 ADC_17_TIMES_SAMPLE

```
#define ADC_17_TIMES_SAMPLE ((u8)17)
```

ADC采样通道数17次采样

在文件 [lks32mc08x_adc.h](#) 第 195 行定义.

3.1.2.10 ADC_18_TIMES_SAMPLE

```
#define ADC_18_TIMES_SAMPLE ((u8)18)
```

ADC采样通道数18次采样

在文件 [lks32mc08x_adc.h](#) 第 196 行定义.

3.1.2.11 ADC_19_TIMES_SAMPLE

```
#define ADC_19_TIMES_SAMPLE ((u8)19)
```

ADC采样通道数19次采样

在文件 [lks32mc08x_adc.h](#) 第 197 行定义.

3.1.2.12 ADC_1_TIMES_SAMPLE

```
#define ADC_1_TIMES_SAMPLE ((u8)1)
```

ADC采样通道数1次采样

在文件 [lks32mc08x_adc.h](#) 第 179 行定义.

3.1.2.13 ADC_1SEG_TRG

```
#define ADC_1SEG_TRG ((uint16_t)0x00)
```

ADC采用单段模式

在文件 [lks32mc08x_adc.h](#) 第 232 行定义.

3.1.2.14 ADC_20_TIMES_SAMPLE

```
#define ADC_20_TIMES_SAMPLE ((u8)20)
```

ADC采样通道数20次采样

在文件 [lks32mc08x_adc.h](#) 第 198 行定义.

3.1.2.15 ADC_2_TIMES_SAMPLE

```
#define ADC_2_TIMES_SAMPLE ((u8)2)
```

ADC采样通道数2次采样

在文件 [lks32mc08x_adc.h](#) 第 180 行定义.

3.1.2.16 ADC_2SEG_TRG

```
#define ADC_2SEG_TRG ((uint16_t)0x01)
```

ADC采用2段模式

在文件 [lks32mc08x_adc.h](#) 第 233 行定义.

3.1.2.17 ADC_3_TIMES_SAMPLE

```
#define ADC_3_TIMES_SAMPLE ((u8)3)
```

ADC采样通道数3次采样

在文件 [lks32mc08x_adc.h](#) 第 181 行定义.

3.1.2.18 ADC_4_TIMES_SAMPLE

```
#define ADC_4_TIMES_SAMPLE ((u8)4)
```

ADC采样通道数4次采样

在文件 [lks32mc08x_adc.h](#) 第 182 行定义.

3.1.2.19 ADC_4SEG_TRG

```
#define ADC_4SEG_TRG ((uint16_t)0x03)
```

ADC采用4段模式

在文件 [lks32mc08x_adc.h](#) 第 234 行定义.

3.1.2.20 ADC_5_TIMES_SAMPLE

```
#define ADC_5_TIMES_SAMPLE ((u8)5)
```

ADC采样通道数5次采样

在文件 [lks32mc08x_adc.h](#) 第 183 行定义.

3.1.2.21 ADC_6_TIMES_SAMPLE

```
#define ADC_6_TIMES_SAMPLE ((u8)6)
```

ADC采样通道数6次采样

在文件 [lks32mc08x_adc.h](#) 第 184 行定义.

3.1.2.22 ADC_7_TIMES_SAMPLE

```
#define ADC_7_TIMES_SAMPLE ((u8)7)
```

ADC采样通道数7次采样

在文件 [lks32mc08x_adc.h](#) 第 185 行定义.

3.1.2.23 ADC_8_TIMES_SAMPLE

```
#define ADC_8_TIMES_SAMPLE ((u8)8)
```

ADC采样通道数8次采样

在文件 [lks32mc08x_adc.h](#) 第 186 行定义.

3.1.2.24 ADC_9_TIMES_SAMPLE

```
#define ADC_9_TIMES_SAMPLE ((u8)9)
```

ADC采样通道数9次采样

在文件 [lks32mc08x_adc.h](#) 第 187 行定义.

3.1.2.25 ADC_ALL_IRQ_IF

```
#define ADC_ALL_IRQ_IF ((uint16_t)0x7f)
```

ADC全部中断标志位——可用于初始化

在文件 [lks32mc08x_adc.h](#) 第 215 行定义.

3.1.2.26 ADC_CHANNEL_0

```
#define ADC_CHANNEL_0 ((uint16_t)0x00)
```

ADC通道0-OPA0_OUT

在文件 [lks32mc08x_adc.h](#) 第 236 行定义.

3.1.2.27 ADC_CHANNEL_1

```
#define ADC_CHANNEL_1 ((uint16_t)0x01)
```

ADC通道1-OPA1_OUT

在文件 [lks32mc08x_adc.h](#) 第 237 行定义.

3.1.2.28 ADC_CHANNEL_10

```
#define ADC_CHANNEL_10 ((uint16_t)0x0a)
```

ADC通道10

在文件 [lks32mc08x_adc.h](#) 第 246 行定义.

3.1.2.29 ADC_CHANNEL_11

```
#define ADC_CHANNEL_11 ((uint16_t)0x0b)
```

ADC通道11

在文件 [lks32mc08x_adc.h](#) 第 247 行定义.



3.1.2.30 ADC_CHANNEL_12

```
#define ADC_CHANNEL_12 ((uint16_t)0x0c)
```

ADC通道12

在文件 [lks32mc08x_adc.h](#) 第 248 行定义.

3.1.2.31 ADC_CHANNEL_13

```
#define ADC_CHANNEL_13 ((uint16_t)0x0d)
```

ADC通道13

在文件 [lks32mc08x_adc.h](#) 第 249 行定义.

3.1.2.32 ADC_CHANNEL_14

```
#define ADC_CHANNEL_14 ((uint16_t)0x0e)
```

ADC通道14

在文件 [lks32mc08x_adc.h](#) 第 250 行定义.

3.1.2.33 ADC_CHANNEL_15

```
#define ADC_CHANNEL_15 ((uint16_t)0x0f)
```

ADC通道15

在文件 [lks32mc08x_adc.h](#) 第 251 行定义.

3.1.2.34 ADC_CHANNEL_16

```
#define ADC_CHANNEL_16 ((uint16_t)0x10)
```

ADC通道16

在文件 [lks32mc08x_adc.h](#) 第 252 行定义.

3.1.2.35 ADC_CHANNEL_17

```
#define ADC_CHANNEL_17 ((uint16_t)0x11)
```

ADC通道17

在文件 [lks32mc08x_adc.h](#) 第 253 行定义.

3.1.2.36 ADC_CHANNEL_18

```
#define ADC_CHANNEL_18 ((uint16_t)0x12)
```

ADC通道18-Temp

在文件 [lks32mc08x_adc.h](#) 第 254 行定义.

3.1.2.37 ADC_CHANNEL_19

```
#define ADC_CHANNEL_19 ((uint16_t)0x13)
```

ADC通道19-VSS

在文件 [lks32mc08x_adc.h](#) 第 255 行定义.

3.1.2.38 ADC_CHANNEL_2

```
#define ADC_CHANNEL_2 ((uint16_t)0x02)
```

ADC通道2-OPA2_OUT

在文件 [lks32mc08x_adc.h](#) 第 238 行定义.

3.1.2.39 ADC_CHANNEL_3

```
#define ADC_CHANNEL_3 ((uint16_t)0x03)
```

ADC通道3-OPA3_OUT

在文件 [lks32mc08x_adc.h](#) 第 239 行定义.

3.1.2.40 ADC_CHANNEL_4

```
#define ADC_CHANNEL_4 ((uint16_t)0x04)
```

ADC通道4

在文件 [lks32mc08x_adc.h](#) 第 240 行定义.

3.1.2.41 ADC_CHANNEL_5

```
#define ADC_CHANNEL_5 ((uint16_t)0x05)
```

ADC通道5

在文件 [lks32mc08x_adc.h](#) 第 241 行定义.

3.1.2.42 ADC_CHANNEL_6

```
#define ADC_CHANNEL_6 ((uint16_t)0x06)
```

ADC通道6

在文件 [lks32mc08x_adc.h](#) 第 242 行定义.

3.1.2.43 ADC_CHANNEL_7

```
#define ADC_CHANNEL_7 ((uint16_t)0x07)
```

ADC通道7

在文件 [lks32mc08x_adc.h](#) 第 243 行定义.

3.1.2.44 ADC_CHANNEL_8

```
#define ADC_CHANNEL_8 ((uint16_t)0x08)
```

ADC通道8

在文件 [lks32mc08x_adc.h](#) 第 244 行定义.

3.1.2.45 ADC_CHANNEL_9

```
#define ADC_CHANNEL_9 ((uint16_t)0x09)
```

ADC通道9

在文件 [lks32mc08x_adc.h](#) 第 245 行定义.

3.1.2.46 ADC_Clock_12M

```
#define ADC_Clock_12M (0x02) /*12MHz*/
```

在文件 [lks32mc08x_adc.h](#) 第 174 行定义.

3.1.2.47 ADC_Clock_24M

```
#define ADC_Clock_24M (0x03) /*24MHz*/
```

在文件 [lks32mc08x_adc.h](#) 第 173 行定义.

3.1.2.48 ADC_Clock_48M

```
#define ADC_Clock_48M (0x00) /*48MHz*/
```

在文件 [lks32mc08x_adc.h](#) 第 172 行定义.

3.1.2.49 ADC_DAT0_HTH

```
#define ADC_DAT0_HTH ((uint16_t)0x0001)
```

ADC_DAT0_TH作为上阈值

在文件 [lks32mc08x_adc.h](#) 第 220 行定义.

3.1.2.50 ADC_DAT0_LTH

```
#define ADC_DAT0_LTH ((uint16_t)0x0000)
```

ADC_DAT0_TH作为下阈值

在文件 [lks32mc08x_adc.h](#) 第 221 行定义.

3.1.2.51 ADC_DAT0_OV_IRQ_EN

```
#define ADC_DAT0_OV_IRQ_EN ((uint16_t)0x40)
```

ADC0_DAT0超阈值中断

在文件 [lks32mc08x_adc.h](#) 第 200 行定义.

3.1.2.52 ADC_DAT0_OV_IRQ_IF

```
#define ADC_DAT0_OV_IRQ_IF ((uint16_t)0x40)
```

ADC_DAT0超阈值中断标志位

在文件 [lks32mc08x_adc.h](#) 第 208 行定义.

3.1.2.53 ADC_EOS0_IRQ_EN

```
#define ADC_EOS0_IRQ_EN ((uint16_t)0x01)
```

第一段扫描结束中断

在文件 [lks32mc08x_adc.h](#) 第 206 行定义.

3.1.2.54 ADC_EOS0_IRQ_IF

```
#define ADC_EOS0_IRQ_IF ((uint16_t)0x01)
```

第一段扫描结束中断标志

在文件 [lks32mc08x_adc.h](#) 第 214 行定义.

3.1.2.55 ADC_EOS1_IRQ_EN

```
#define ADC_EOS1_IRQ_EN ((uint16_t)0x02)
```

第二段扫描结束中断

在文件 [lks32mc08x_adc.h](#) 第 205 行定义.

3.1.2.56 ADC_EOS1_IRQ_IF

```
#define ADC_EOS1_IRQ_IF ((uint16_t)0x02)
```

第二段扫描结束中断标志

在文件 [lks32mc08x_adc.h](#) 第 213 行定义.

3.1.2.57 ADC_EOS2_IRQ_EN

```
#define ADC_EOS2_IRQ_EN ((uint16_t)0x04)
```

第三段扫描结束中断

在文件 [lks32mc08x_adc.h](#) 第 204 行定义.

3.1.2.58 ADC_EOS2_IRQ_IF

```
#define ADC_EOS2_IRQ_IF ((uint16_t)0x04)
```

第三段扫描结束中断标志

在文件 [lks32mc08x_adc.h](#) 第 212 行定义.

3.1.2.59 ADC_EOS3_IRQ_EN

```
#define ADC_EOS3_IRQ_EN ((uint16_t)0x08)
```

第四段扫描结束中断

在文件 [lks32mc08x_adc.h](#) 第 203 行定义.

3.1.2.60 ADC_EOS3_IRQ_IF

```
#define ADC_EOS3_IRQ_IF ((uint16_t)0x08)
```

第四段扫描结束中断标志

在文件 [lks32mc08x_adc.h](#) 第 211 行定义.

3.1.2.61 ADC_H.CONFLICT_IRQ_EN

```
#define ADC_H.CONFLICT_IRQ_EN ((uint16_t)0x20)
```

硬件触发冲突

在文件 [lks32mc08x_adc.h](#) 第 201 行定义.

3.1.2.62 ADC_H.CONFLICT_IRQ_IF

```
#define ADC_H.CONFLICT_IRQ_IF ((uint16_t)0x20)
```

硬件触发冲突标志

在文件 [lks32mc08x_adc.h](#) 第 209 行定义.

3.1.2.63 ADC_HARDWARE_T0_TRG

```
#define ADC_HARDWARE_T0_TRG ((uint16_t)0x01)
```

ADC采用硬件T0事件触发

在文件 [lks32mc08x_adc.h](#) 第 257 行定义.

3.1.2.64 ADC_HARDWARE_T1_TRG

```
#define ADC_HARDWARE_T1_TRG ((uint16_t)0x02)
```

ADC采用硬件T1事件触发

在文件 [lks32mc08x_adc.h](#) 第 258 行定义.

3.1.2.65 ADC_HARDWARE_T2_TRG

```
#define ADC_HARDWARE_T2_TRG ((uint16_t)0x04)
```

ADC采用硬件T2事件触发

在文件 [lks32mc08x_adc.h](#) 第 259 行定义.

3.1.2.66 ADC_HARDWARE_T3_TRG

```
#define ADC_HARDWARE_T3_TRG ((uint16_t)0x08)
```

ADC采用硬件T3事件触发

在文件 [lks32mc08x_adc.h](#) 第 260 行定义.

3.1.2.67 ADC_LEFT_ALIGN

```
#define ADC_LEFT_ALIGN ((uint16_t)0x0000)
```

ADC数据输出左对齐

在文件 [lks32mc08x_adc.h](#) 第 217 行定义.

3.1.2.68 ADC_MCPWM_T0_TRG

```
#define ADC_MCPWM_T0_TRG ((uint16_t)0x01)
```

ADC采用MCPWM T0事件触发

在文件 [lks32mc08x_adc.h](#) 第 223 行定义.

3.1.2.69 ADC_MCPWM_T1_TRG

```
#define ADC_MCPWM_T1_TRG ((uint16_t)0x02)
```

ADC采用MCPWM T1事件触发

在文件 [lks32mc08x_adc.h](#) 第 224 行定义.

3.1.2.70 ADC_MCPWM_T2_TRG

```
#define ADC_MCPWM_T2_TRG ((uint16_t)0x04)
```

ADC采用MCPWM T2事件触发

在文件 [lks32mc08x_adc.h](#) 第 225 行定义.

3.1.2.71 ADC_MCPWM_T3_TRG

```
#define ADC_MCPWM_T3_TRG ((uint16_t)0x08)
```

ADC采用MCPWM T3事件触发

在文件 [lks32mc08x_adc.h](#) 第 226 行定义.

3.1.2.72 ADC_RIGHT_ALIGN

```
#define ADC_RIGHT_ALIGN ((uint16_t)0x0001)
```

ADC数据输出右对齐

在文件 [lks32mc08x_adc.h](#) 第 218 行定义.

3.1.2.73 ADC_S_CONFLICT_IRQ_EN

```
#define ADC_S_CONFLICT_IRQ_EN ((uint16_t)0x10)
```

软件触发冲突

在文件 [lks32mc08x_adc.h](#) 第 202 行定义.

3.1.2.74 ADC_S_CONFLICT_IRQ_IF

```
#define ADC_S_CONFLICT_IRQ_IF ((uint16_t)0x10)
```

软件触发冲突标志

在文件 [lks32mc08x_adc.h](#) 第 210 行定义.

3.1.2.75 ADC_UTIMER_T0_TRG

```
#define ADC_UTIMER_T0_TRG ((uint16_t)0x01)
```

ADC采用UTIMER T0事件触发

在文件 [lks32mc08x_adc.h](#) 第 227 行定义.

3.1.2.76 ADC_UTIMER_T1_TRG

```
#define ADC_UTIMER_T1_TRG ((uint16_t)0x02)
```

ADC采用UTIMER T1事件触发

在文件 [lks32mc08x_adc.h](#) 第 228 行定义.

3.1.2.77 ADC_UTIMER_T2_TRG

```
#define ADC_UTIMER_T2_TRG ((uint16_t)0x04)
```

ADC采用UTIMER T2事件触发

在文件 [lks32mc08x_adc.h](#) 第 229 行定义.

3.1.2.78 ADC_UTIMER_T3_TRG

```
#define ADC_UTIMER_T3_TRG ((uint16_t)0x08)
```

ADC采用UTIMER T3事件触发

在文件 [lks32mc08x_adc.h](#) 第 230 行定义.

3.1.2.79 RefVol_1V2

```
#define RefVol_1V2 ((uint16_t)0x01) /*ADC基准电压1.2V*/
```

在文件 [lks32mc08x_adc.h](#) 第 177 行定义。

3.1.2.80 RefVol_2V4

```
#define RefVol_2V4 ((uint16_t)0x00) /*ADC基准电压2.4V*/
```

在文件 [lks32mc08x_adc.h](#) 第 176 行定义。

3.1.3 枚举类型说明

3.1.3.1 CHNx

```
enum CHNx
```

ADC采样信号来源寄存器序号

枚举值

CHN0	第0次采样信号对应信号来源寄存器序号
CHN1	第1次采样信号对应信号来源寄存器序号
CHN2	第2次采样信号对应信号来源寄存器序号
CHN3	第3次采样信号对应信号来源寄存器序号
CHN4	第4次采样信号对应信号来源寄存器序号
CHN5	第5次采样信号对应信号来源寄存器序号
CHN6	第6次采样信号对应信号来源寄存器序号
CHN7	第7次采样信号对应信号来源寄存器序号
CHN8	第8次采样信号对应信号来源寄存器序号
CHN9	第9次采样信号对应信号来源寄存器序号
CHN10	第10次采样信号对应信号来源寄存器序号
CHN11	第11次采样信号对应信号来源寄存器序号
CHN12	第12次采样信号对应信号来源寄存器序号
CHN13	第13次采样信号对应信号来源寄存器序号
CHN14	第14次采样信号对应信号来源寄存器序号
CHN15	第15次采样信号对应信号来源寄存器序号
CHN16	第16次采样信号对应信号来源寄存器序号
CHN17	第17次采样信号对应信号来源寄存器序号
CHN18	第18次采样信号对应信号来源寄存器序号
CHN19	第19次采样信号对应信号来源寄存器序号

在文件 [lks32mc08x_adc.h](#) 第 263 行定义。

3.1.3.2 DATx

enum DATx

ADC数据寄存器序号

枚举值

DAT0	第0次采样信号对应结果寄存器序号
DAT1	第1次采样信号对应结果寄存器序号
DAT2	第2次采样信号对应结果寄存器序号
DAT3	第3次采样信号对应结果寄存器序号
DAT4	第4次采样信号对应结果寄存器序号
DAT5	第5次采样信号对应结果寄存器序号
DAT6	第6次采样信号对应结果寄存器序号
DAT7	第7次采样信号对应结果寄存器序号
DAT8	第8次采样信号对应结果寄存器序号
DAT9	第9次采样信号对应结果寄存器序号
DAT10	第10次采样信号对应结果寄存器序号
DAT11	第11次采样信号对应结果寄存器序号
DAT12	第12次采样信号对应结果寄存器序号
DAT13	第13次采样信号对应结果寄存器序号
DAT14	第14次采样信号对应结果寄存器序号
DAT15	第15次采样信号对应结果寄存器序号
DAT16	第16次采样信号对应结果寄存器序号
DAT17	第17次采样信号对应结果寄存器序号
DAT18	第18次采样信号对应结果寄存器序号
DAT19	第19次采样信号对应结果寄存器序号

在文件 [lks32mc08x_adc.h](#) 第 288 行定义.

3.1.4 函数说明

3.1.4.1 ADC_ChannelConfig()

```
void ADC_ChannelConfig (
    ADC_TypeDef * ADCx,
    CHNx CHNum,
    uint16_t nChannel )
```

函数名称: void ADC_ChannelConfig(ADC_TypeDef* ADCx,CHNx CHNum, uint16_t Channel_0 ,uint16_t Channel_1)

功能描述: ADC采样通道设置

参见

被引用内容: [CHNx](#)

参数

输入参数:	ADCx, 08系列固定选择ADC0 CHNum为枚举类型, 对应0-19; Channel.0为采样通道序号对应0-19 nChannel: ADC_CHANNEL_0 ~ ADC_CHANNEL_19
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
ADC_ChannelConfig(ADC0, CHN0, ADC_CHANNEL_3); //配置ADC第0次采样ADC通道3的信号
ADC_ChannelConfig(ADC0, CHN1, ADC_CHANNEL_0); //配置ADC第1次采样ADC通道0的信号
```

Date	Version	Author	Description
2022年04月20日	1.0	Zhu Jie	创建

修改日志:

在文件 [lks32mc08x_adc.c](#) 第 264 行定义.

3.1.4.2 ADC_ClearIRQFlag()

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

函数名称: void [ADC_ClearIRQFlag\(ADC_TypeDef* ADCx, uint16_t INT_flag\)](#)

功能描述: ADC中断标志位清零

参见

被引用内容: 无

参数

输入参数:	ADCx:	ADC0
	INT_flag:	ADC中断标志
	ADC_DAT0_OV_IRQ_IF	ADC_DAT0超阈值中断标志位
	ADC_H_CONFLICT_IRQ_IF	硬件触发冲突标志
	ADC_S_CONFLICT_IRQ_IF	软件触发冲突标志
	ADC_EOS3_IRQ_IF	第四段扫描结束中断标志
	ADC_EOS2_IRQ_IF	第三段扫描结束中断标志
	ADC_EOS1_IRQ_IF	第二段扫描结束中断标志
	ADC_EOS0_IRQ_IF	第一段扫描结束中断标志
	ADC_ALL_IRQ_IF	ADC全部中断标志位——可用于初始化
输出参数:	无	

返回

返回 值: 无

注解

其它说明: 无

警告

无

示例代码:

```
if (ADC_GetIRQFlag (ADC0, ADC_EOS0_IRQ_IF) ) // 获取第一段扫描结束中断标志
{
    ADC_ClearIRQFlag (ADC0, ADC_EOS0_IRQ_IF) // 第一段扫描结束中断标志
}
```

Date	Version	Author	Description
2022年04月20日	1.0	Zhu Jie	创建

修改日志:

在文件 [lks32mc08x_adc.c](#) 第 189 行定义.

3.1.4.3 ADC_GetConversionValue()

```
s16 ADC_GetConversionValue (
    DATx DATNum )
```

函数名称: s16 ADC_GetConversionValue(DATNx DATNum)

功能描述: 读ADC采样值

参见

被引用内容: DATx

参数

输入参数:	DATx: 读取第几次采样结果值
输出参数:	无

返回

返回值: 对应x次采样结果值

注解

其它说明: 无

警告

无

示例代码:

```
s16 ADC_samp0 = 0;
ADC_samp0 = ADC_GetConversionValue (DAT0); //读取ADC第0次采样结果
```

Date	Version	Author	Description
2022年04月20日	1.0	Zhu Jie	创建

修改日志:

在文件 [lks32mc08x_adc.c](#) 第 295 行定义.

3.1.4.4 ADC_GetIRQFlag()

```
u16 ADC_GetIRQFlag (
    ADC_TypeDef * ADCx,
    uint16_t INT_flag )
```

函数名称: `uint16_t ADC_GetIRQFlag(ADC_TypeDef* ADCx, uint16_t INT_flag)`

功能描述: 获取ADC中断标志

参见
被引用内容: 无

参数

输入参数:	ADCx:	ADC0
	INT_flag:	ADC中断标志
	ADC_DAT0_OV_IRQ_IF	ADC_DAT0超阈值中断标志位
	ADC_H_CONFLICT_IRQ_IF	硬件触发冲突标志
	ADC_S_CONFLICT_IRQ_IF	软件触发冲突标志
	ADC_EOS3_IRQ_IF	第四段扫描结束中断标志
	ADC_EOS2_IRQ_IF	第三段扫描结束中断标志
	ADC_EOS1_IRQ_IF	第二段扫描结束中断标志
	ADC_EOS0_IRQ_IF	第一段扫描结束中断标志
	ADC_ALL_IRQ_IF	ADC全部中断标志位——可用于初始化
输出参数:	无	

返回
返回 值: 读对应标志位是否为1, 1位置位, 0为未置位

注解
其它说明: 无

警告
只有对应中断使能后, 改为才能读取, 如果对应中断未使能, 读取结果一直为0

示例代码:

```
if(ADC_GetIRQFlag(ADC0,ADC_EOS0_IRQ_IF))//获取第一段扫描结束中断标志
{
    ADC_ClearIRQFlag(ADC0,ADC_EOS0_IRQ_IF)//第一段扫描结束中断标志
}
```

Date	Version	Author	Description
2022年04月20日	1.0	Zhu Jie	创建

修改日志:

在文件 `lks32mc08x_adc.c` 第 227 行定义.

3.1.4.5 ADC_Init()

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

函数名称: `void ADC_Init(ADC_TypeDef* ADCx, ADC_InitTypeDef* ADC_InitStruct)`

功能描述: ADC初始化函数

参见

被引用内容: `SYS_AnalogModuleClockCmd()`

参数

输入参数:	ADCx: ADC0 , <code>ADC_InitTypeDef *ADC_InitTypeDef</code>
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
ADC_InitTypeDef ADC_InitStructure;
ADC_StructInit (&ADC_InitStructure);
ADC_InitStructure.ADC_CLK_DIV = ADC_Clock_48M;
ADC_InitStructure.Align = ADC_LEFT_ALIGN;
ADC_InitStructure.Trigger_Mode = ADC_2SEG_TRG;
ADC_InitStructure.FirSeg_Ch = ADC_2_TIMES_SAMPLE;
ADC_InitStructure.SecSeg_Ch = ADC_2_TIMES_SAMPLE;
ADC_InitStructure.ThrSeg_Ch = 0;
ADC_InitStructure.FouSeg_Ch = 0;
ADC_InitStructure.RefVol = RefVol_2V4;
```

```
// 初始化结构体
// ADC时钟选择48MHz
// ADC数据输出左对齐
// 设置ADC转换模式为双段式采样
// 第一段共采样2个通道
// 第二段共采样2个通道
// 第三段共采样0个通道
// 第四段共采样0个通道
// ADC基准电压2.4V
```

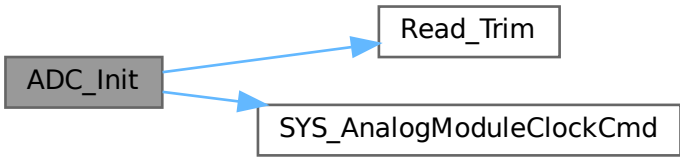
```
ADC.InitStructure.Gain0 = 0x0; // ADC_DAT0-ADC_DAT9 (BIT0~BIT9) 采样保持电
路增益设置2/3
ADC.InitStructure.Gain1 = 0x0; // ADC_DAT10-ADC_DAT19 (BIT10~BIT19) 采样保
持电路增益设置2/3
ADC.InitStructure.Con.Sample = DISABLE; // 关闭连续采样
// 打开 UTIMER T0 T1 硬件触发采样
ADC.InitStructure.MCPWM.Trigger.En = ADC_MCPWM_T0_TRG|ADC_MCPWM_T1_TRG; // 开启UTIMER硬件第一段和第二段触发采样
ADC.InitStructure.UTIMER.Trigger.En = DISABLE; // 关闭UTIMER硬件触发采样
ADC.InitStructure.Trigger.Cnt = 0; // 发生 (Trigger.Cnt+1) 次触发事件执行一段转换
动作
ADC.InitStructure.IE = ADC_EOS0_IRQ_EN | ADC_EOS1_IRQ_EN; // ADC第一、二段采样结束中断使能
ADC_Init (ADC0, &ADC.InitStructure);
```

Date	Version	Author	Description
2015年11月26日	1.0	Howlet Li	创建

修改日志:

在文件 [lks32mc08x_adc.c](#) 第 59 行定义.

函数调用图:



3.1.4.6 ADC_SoftTrgEN()

```
void ADC_SoftTrgEN (
    ADC_TypeDef * ADCx,
    FuncState state )
```

函数名称: void [ADC_SoftTrgEN\(ADC_TypeDef* ADCx, FuncState state\)](#)

功能描述: ADC采样软件触发使能

参见

被引用内容: FuncState

参数

输入参数:	ADCx, 08系列固定选择ADC0 state: ENABLE触发, DISABLE不触发
输出参数:	无

返回

返回值: 无

注解

其它说明: ADC0_SWT写0x5AA5仅触发一次, 完成后自动清零, 等待下次软件触发

警告

无

示例代码:

```
ADC_SoftTrgEN(ADC0, ENABLE); //软件触发一次ADC采样
```

Date	Version	Author	Description
2022年04月20日	1.0	Zhu Jie	创建

修改日志:

在文件 [lks32mc08x_adc.c](#) 第 399 行定义.

3.1.4.7 ADC_StructInit()

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

函数名称: void [ADC_StructInit\(ADC_InitTypeDef* ADC_InitStruct\)](#)

功能描述: ADC结构体初始化

参见

被引用内容: 无

参数

输入参数:	ADC_InitTypeDef *ADC_InitStruct
输出参数:	无

返回

返回值: 无

注解

其它说明： 无

警告

无

示例代码:

```
ADC_InitTypeDef ADC_InitStructure;  
ADC_StructInit (&ADC_InitStructure); //初始化结构体
```

Date	Version	Author	Description
2015年11月26日	1.0	Howlet	修改

修改日志:

在文件 [lks32mc08x_adc.c](#) 第 136 行定义.

3.2 lks32mc08x_adc.h

[浏览该文件的文档.](#)

```
00001  
00014 #ifndef _LKS32MC08x_ADC_H_  
00015 #define _LKS32MC08x_ADC_H_  
00016  
00020 #include "lks32mc08x_lib.h"  
00021 typedef struct  
00022 {  
00023     __IO uint32_t DAT0;  
00024     __IO uint32_t DAT1;  
00025     __IO uint32_t DAT2;  
00026     __IO uint32_t DAT3;  
00027     __IO uint32_t DAT4;  
00028     __IO uint32_t DAT5;  
00029     __IO uint32_t DAT6;  
00030     __IO uint32_t DAT7;  
00031     __IO uint32_t DAT8;  
00032     __IO uint32_t DAT9;  
00033     __IO uint32_t DAT10;  
00034     __IO uint32_t DAT11;  
00035     __IO uint32_t DAT12;  
00036     __IO uint32_t DAT13;  
00037     __IO uint32_t DAT14;  
00038     __IO uint32_t DAT15;  
00039     __IO uint32_t DAT16;  
00040     __IO uint32_t DAT17;  
00041     __IO uint32_t DAT18;  
00042     __IO uint32_t DAT19;  
00044     __IO uint32_t CHN0;  
00045     __IO uint32_t CHN1;  
00046     __IO uint32_t CHN2;  
00047     __IO uint32_t CHN3;  
00048     __IO uint32_t CHN4;  
00049     __IO uint32_t CHN5;  
00050     __IO uint32_t CHN6;  
00051     __IO uint32_t CHN7;  
00052     __IO uint32_t CHN8;  
00053     __IO uint32_t CHN9;  
00055     __IO uint32_t CHNT0;  
00056     __IO uint32_t CHNT1;
```

```

00057     __IO uint32_t SD0;
00058     __IO uint32_t SD1;
00059     __IO uint32_t GAIN_0;
00060     __IO uint32_t GAIN_1;
00061                                     /* 采样保持电路A对应通道范围是第0、2、4等偶数字号通道 */
00062     __IO uint32_t DC_A0;
00063     __IO uint32_t DC_A1;
00064     __IO uint32_t AMC_A0;
00065     __IO uint32_t AMC_A1;
00066                                     /* 采样保持电路B对应通道范围是第1、3、5等奇数字号通道 */
00067     __IO uint32_t DC_B0;
00068     __IO uint32_t DC_B1;
00069     __IO uint32_t AMC_B0;
00070     __IO uint32_t AMC_B1;
00072     __IO uint32_t IE;
00073     __IO uint32_t IF;
00074     __IO uint32_t CFG;
00075     __IO uint32_t TRG;
00076     __IO uint32_t SWT;
00077     __IO uint32_t DAT0_TH;
00078 } ADC_TypeDef;
00079
00083 #ifndef ADC0
00084 #define ADC0 ((ADC_TypeDef *)ADC0_BASE)
00085 #endif
00089 typedef struct
00090 {
00101     uint16_t IE;
00107     uint16_t Align;
00114     uint16_t ADC_CLK_DIV;
00115     uint16_t Con_Sample;
00119     uint16_t Trigger_Cnt;
00124     uint16_t FirSeq_Ch;
00128     uint16_t SecSeq_Ch;
00132     uint16_t ThrSeq_Ch;
00136     uint16_t FouSeq_Ch;
00143     uint16_t Trigger_Mode;
00151     uint16_t MCPWM_Trigger_En;
00159     uint16_t UTIMER_Trigger_En;
00160     uint16_t DAT0_TH_Type;
00161     uint16_t DAT0_TH;
00162     uint16_t Gain0;
00163     uint16_t Gain1;
00169     uint16_t RefVol;
00170 } ADC_InitTypeDef;
00171
00172 #define ADC_Clock_48M      (0x00) /*48MHz*/
00173 #define ADC_Clock_24M      (0x03) /*24MHz*/
00174 #define ADC_Clock_12M      (0x02) /*12MHz*/
00175
00176 #define RefVol_2V4         ((uint16_t)0x00) /*ADC基准电压2.4V*/
00177 #define RefVol_1V2         ((uint16_t)0x01) /*ADC基准电压1.2V*/
00178
00179 #define ADC_1_TIMES_SAMPLE ((u8)1)
00180 #define ADC_2_TIMES_SAMPLE ((u8)2)
00181 #define ADC_3_TIMES_SAMPLE ((u8)3)
00182 #define ADC_4_TIMES_SAMPLE ((u8)4)
00183 #define ADC_5_TIMES_SAMPLE ((u8)5)
00184 #define ADC_6_TIMES_SAMPLE ((u8)6)
00185 #define ADC_7_TIMES_SAMPLE ((u8)7)
00186 #define ADC_8_TIMES_SAMPLE ((u8)8)
00187 #define ADC_9_TIMES_SAMPLE ((u8)9)
00188 #define ADC_10_TIMES_SAMPLE ((u8)10)
00189 #define ADC_11_TIMES_SAMPLE ((u8)11)
00190 #define ADC_12_TIMES_SAMPLE ((u8)12)
00191 #define ADC_13_TIMES_SAMPLE ((u8)13)
00192 #define ADC_14_TIMES_SAMPLE ((u8)14)
00193 #define ADC_15_TIMES_SAMPLE ((u8)15)
00194 #define ADC_16_TIMES_SAMPLE ((u8)16)
00195 #define ADC_17_TIMES_SAMPLE ((u8)17)
00196 #define ADC_18_TIMES_SAMPLE ((u8)18)
00197 #define ADC_19_TIMES_SAMPLE ((u8)19)
00198 #define ADC_20_TIMES_SAMPLE ((u8)20)
00200 #define ADC_DAT0_OV_IRQ_EN ((uint16_t)0x40)
00201 #define ADC_H_CONFLICT_IRQ_EN ((uint16_t)0x20)
00202 #define ADC_S_CONFLICT_IRQ_EN ((uint16_t)0x10)
00203 #define ADC_EOS3_IRQ_EN ((uint16_t)0x08)
00204 #define ADC_EOS2_IRQ_EN ((uint16_t)0x04)
00205 #define ADC_EOS1_IRQ_EN ((uint16_t)0x02)
00206 #define ADC_EOS0_IRQ_EN ((uint16_t)0x01)
00208 #define ADC_DAT0_OV_IRQ_IF ((uint16_t)0x40)
00209 #define ADC_H_CONFLICT_IRQ_IF ((uint16_t)0x20)
00210 #define ADC_S_CONFLICT_IRQ_IF ((uint16_t)0x10)
00211 #define ADC_EOS3_IRQ_IF ((uint16_t)0x08)
00212 #define ADC_EOS2_IRQ_IF ((uint16_t)0x04)
00213 #define ADC_EOS1_IRQ_IF ((uint16_t)0x02)
00214 #define ADC_EOS0_IRQ_IF ((uint16_t)0x01)

```

```

00215 #define ADC_ALL_IRQ_IF      ((uint16_t)0x7f)
00217 #define ADC_LEFT_ALIGN      ((uint16_t)0x0000)
00218 #define ADC_RIGHT_ALIGN     ((uint16_t)0x0001)
00220 #define ADC_DAT0_HTH        ((uint16_t)0x0001)
00221 #define ADC_DAT0_LTH        ((uint16_t)0x0000)
00223 #define ADC_MCPWM_T0_TRG     ((uint16_t)0x01)
00224 #define ADC_MCPWM_T1_TRG     ((uint16_t)0x02)
00225 #define ADC_MCPWM_T2_TRG     ((uint16_t)0x04)
00226 #define ADC_MCPWM_T3_TRG     ((uint16_t)0x08)
00227 #define ADC_UTIMER_T0_TRG    ((uint16_t)0x01)
00228 #define ADC_UTIMER_T1_TRG    ((uint16_t)0x02)
00229 #define ADC_UTIMER_T2_TRG    ((uint16_t)0x04)
00230 #define ADC_UTIMER_T3_TRG    ((uint16_t)0x08)
00232 #define ADC_1SEG_TRG         ((uint16_t)0x00)
00233 #define ADC_2SEG_TRG         ((uint16_t)0x01)
00234 #define ADC_4SEG_TRG         ((uint16_t)0x03)
00236 #define ADC_CHANNEL_0        ((uint16_t)0x00)
00237 #define ADC_CHANNEL_1        ((uint16_t)0x01)
00238 #define ADC_CHANNEL_2        ((uint16_t)0x02)
00239 #define ADC_CHANNEL_3        ((uint16_t)0x03)
00240 #define ADC_CHANNEL_4        ((uint16_t)0x04)
00241 #define ADC_CHANNEL_5        ((uint16_t)0x05)
00242 #define ADC_CHANNEL_6        ((uint16_t)0x06)
00243 #define ADC_CHANNEL_7        ((uint16_t)0x07)
00244 #define ADC_CHANNEL_8        ((uint16_t)0x08)
00245 #define ADC_CHANNEL_9        ((uint16_t)0x09)
00246 #define ADC_CHANNEL_10       ((uint16_t)0x0a)
00247 #define ADC_CHANNEL_11       ((uint16_t)0x0b)
00248 #define ADC_CHANNEL_12       ((uint16_t)0x0c)
00249 #define ADC_CHANNEL_13       ((uint16_t)0x0d)
00250 #define ADC_CHANNEL_14       ((uint16_t)0x0e)
00251 #define ADC_CHANNEL_15       ((uint16_t)0x0f)
00252 #define ADC_CHANNEL_16       ((uint16_t)0x10)
00253 #define ADC_CHANNEL_17       ((uint16_t)0x11)
00254 #define ADC_CHANNEL_18       ((uint16_t)0x12)
00255 #define ADC_CHANNEL_19       ((uint16_t)0x13)
00257 #define ADC_HARDWARE_T0_TRG  ((uint16_t)0x01)
00258 #define ADC_HARDWARE_T1_TRG  ((uint16_t)0x02)
00259 #define ADC_HARDWARE_T2_TRG  ((uint16_t)0x04)
00260 #define ADC_HARDWARE_T3_TRG  ((uint16_t)0x08)
00263 typedef enum
00264 {
00265     CHN0 = 0,
00266     CHN1,
00267     CHN2,
00268     CHN3,
00269     CHN4,
00270     CHN5,
00271     CHN6,
00272     CHN7,
00273     CHN8,
00274     CHN9,
00275     CHN10,
00276     CHN11,
00277     CHN12,
00278     CHN13,
00279     CHN14,
00280     CHN15,
00281     CHN16,
00282     CHN17,
00283     CHN18,
00284     CHN19,
00285 } CHNx;
00286
00288 typedef enum
00289 {
00290     DAT0 = 0,
00291     DAT1,
00292     DAT2,
00293     DAT3,
00294     DAT4,
00295     DAT5,
00296     DAT6,
00297     DAT7,
00298     DAT8,
00299     DAT9,
00300     DAT10,
00301     DAT11,
00302     DAT12,
00303     DAT13,
00304     DAT14,
00305     DAT15,
00306     DAT16,
00307     DAT17,
00308     DAT18,
00309     DAT19,
00310 } DATx;

```

```

00311
00312 void ADC_Init(ADC_TypeDef *ADCx, ADC_InitTypeDef *ADC_InitStruct);
00313 void ADC_StructInit(ADC_InitTypeDef *ADC_InitStruct);
00314 void ADC_ClearIRQFlag(ADC_TypeDef *ADCx, uint16_t INT_flag);
00315 void ADC_ChannelConfig(ADC_TypeDef *ADCx, CHNx ChNum, uint16_t nChannel);
00316 uint16_t ADC_GetIRQFlag(ADC_TypeDef *ADCx, uint16_t INT_flag);
00317 uint16_t ADC_GetConversionValue(DATx DATNum); /* 取对应序号ADC数据寄存器的值 */
00318 void ADC_SoftTrgEN(ADC_TypeDef *ADCx, FuncState state);
00319 #endif /*_LKS32MC08x_ADC_H_*/
00320
00321 /***** (C) COPYRIGHT LINKO SEMICONDUCTOR *****/

```

3.3 include/lks32mc08x_can.h 文件参考

文件名称: lks32mc08x_can.h

文件标识: 无

内容摘要: CAN外设驱动程序头文件

其它说明: 无

```
#include "lks32mc08x_lib.h"
```

lks32mc08x_can.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



结构体

- struct **CAN_TypeDef**
CAN寄存器结构体句柄
- struct **CAN_InitTypeDef**
CAN结构体基地址定义

宏定义

- #define CAN_SELFTEST_MODE BIT2
CAN工作模式选择,自测模式
- #define CAN_MONITOR_MODE BIT1
CAN工作模式选择,监听模式
- #define CAN_NORMAL_MODE 0x0
CAN工作模式选择,正常收发模式
- #define CAN_IE_BUSERR BIT7
中断使能定义,总线错误中断使能
- #define CAN_IE_LOSTARB BIT6

- 中断使能定义,丢失仲裁中断使能
- #define CAN_IE_PASSIVEERR BIT5
中断使能定义,被动错误错误中断使能
- #define CAN_IE_WAKE BIT4
中断使能定义,CAN模块从休眠中唤醒中断使能
- #define CAN_IE_RFIFOOV BIT3
中断使能定义,RFIFO数据发送溢出中断使能
- #define CAN_IE_WERR BIT2
中断使能定义,错误报警中断使能
- #define CAN_IE_TXDONE BIT1
中断使能定义,发送完毕当前帧中断使能
- #define CAN_IE_RFIFONEMPTY BIT0
中断使能定义,RFIFO有新的数据被接收到中断使能
- #define CAN_IF_BUSERR BIT7
总线错误中断标志
- #define CAN_IF_LOSTARB BIT6
丢失仲裁中断标志
- #define CAN_IF_PASSIVEERR BIT5
被动错误错误中断标志
- #define CAN_IF_WAKE BIT4
CAN模块从休眠中唤醒中断标志
- #define CAN_IF_RFIFOOV BIT3
RFIFO数据发送溢出中断标志
- #define CAN_IF_WERR BIT2
错误报警中断标志
- #define CAN_IF_TXDONE BIT1
发送完毕当前帧中断标志
- #define CAN_IF_RFIFONEMPTY BIT0
RFIFO有新的数据被接收到中断标志
- #define CAN_ONBUS BIT7
1: CAN 模块, 处于 BUS OFF 状态, 无数据发送接收动作 \ 0: CAN 模块, 可以发送数据到 CAN 总线, 或接收 CAN 总线数据
- #define CAN_ERROV BIT6
1: CAN 传输产生的错误总数达到或超过 CAN.EWL 规定值 \ 0: CAN 传输产生的错误总数低于 CAN.EWL 规定值
- #define CAN_TXING BIT5
1: CAN 模块正在发送一帧数据 \ 0: CAN 模块没有发送数据的动作
- #define CAN_RXING BIT4
1: CAN 模块正在接收一帧数据 \ 0: CAN 模块没有接收数据的动作
- #define CAN_TRNADONE BIT3
1: 最近一次传输已完成 \ 0: 最近一次传输未完成
- #define CAN_TFIFOEMPTY BIT2
1: TFIFO 空, 可以写入发送数据 \ 0: TFIFO 非空, 内部数据没有发送完毕
- #define CAN_RFIFOEMPTY BIT1
1: RFIFO 存入的帧太多, 已满, 导致数据丢失 \ 0: RFIFO 未满
- #define CAN_DATAAVAIL BIT0
1: RFIFO 存有一帧或多帧数据, 可以通过 RFIFO 寄存器读取 \ 0: RFIFO 没有有效帧数据
- #define CAN_ALLSR 0xff
读取全部状态

枚举

- enum `enumLossposition` {
`CAN_LOST_ID0` , `CAN_LOST_ID1` , `CAN_LOST_ID2` , `CAN_LOST_ID3` ,
`CAN_LOST_ID4` , `CAN_LOST_ID5` , `CAN_LOST_ID6` , `CAN_LOST_ID7` ,
`CAN_LOST_ID8` , `CAN_LOST_ID9` , `CAN_LOST_IDA` , `CAN_LOST_SRTR` ,
`CAN_LOST_IDE` }
- enum `enumErrortype` { `CAN_BITERROR` , `CAN_FORMATERROR` , `CAN_FILLERROR` , `CAN_OTHERERROR` }
- enum `enumErrortime` { `CAN_RECE_PROCESS` , `CAN_SEND_PROCESS` }
- enum `enumErrposition` {
`null0` , `null1` , `CAN_ID28_ID21` , `CAN_START` ,
`CAN_SRTR` , `CAN_IDE` , `CAN_ID20_ID18` , `CAN_ID17_ID13` ,
`CAN_CRC` , `CAN_KEEP0` , `CAN_DATA_FIELD` , `CAN_DATALENGTHCODE` ,
`CAN_RTR` , `CAN_KEEP1` , `CAN_ID4_ID0` , `CAN_ID15_ID12` ,
`null2` , `CAN_ACTIVE_ERROR_FLAG` , `CAN_INTERMISSION` , `CAN_TOLERATE_DOMINANT` ,
`null3` , `null4` , `CAN_PASSIVE_ERROR` , `CAN_ERROR_DELIMITER` ,
`CAN_CRC_DELIMITER` , `CAN_CONNFIRM` , `CAN_FRAMEEND` , `CAN_CONNFIRM_DELIMITER` ,
`CAN_OVERLOAD_FLAG` }

函数

- void `CAN_StructInit` (`CAN_InitTypeDef` *`CAN_InitStruct`)
函数名称: `void CAN_StructInit(CAN_InitTypeDef *CAN_InitStruct)`
- void `CAN_Init` (`CAN_TypeDef` *`CANx`, `CAN_InitTypeDef` *`CAN_InitStruct`)
函数名称: `void CAN_Init(CAN_TypeDef* CANx, CAN_InitTypeDef* CAN_InitStruct)`
- void `SFF_ID_Filter` (`u32` `acr`, `u32` `amr`, `u8` `rtr_acr`, `u8` `rtr_amr`, `u8` `Byte1_acr`, `u8` `Byte1_amr`, `u8` `Byte2_acr`, `u8` `Byte2_amr`)
函数名称: `void SFF_ID_Filter(u32 acr,u32 amr,u8 rtr_acr,u8 rtr_amr, u8 Byte1_acr,u8 Byte1_amr,u8 Byte2_acr,u8 Byte2_amr)`
- void `EFF_ID_Filter` (`u32` `acr`, `u32` `amr`, `u8` `rtr_acr`, `u8` `rtr_amr`)
函数名称: `void EFF_ID_Filter(u32 acr,u32 amr,u8 rtr_acr,u8 rtr_amr)`
- void `ID1_Filter_Dual` (`u32` `acr1`, `u32` `amr1`, `u8` `rtr_acr1`, `u8` `rtr_amr1`, `u8` `Byte_acr1`, `u8` `Byte_amr1`, `u8` `ide`)
函数名称: `void ID1_Filter_Dual(u32 acr1 ,u32 amr1,u8 rtr_acr1, u8 rtr_amr1,u8 Byte_acr1,u8 Byte_amr1,u8 ide)`
- void `ID2_Filter_Dual` (`u32` `acr2`, `u32` `amr2`, `u8` `rtr_acr2`, `u8` `rtr_amr2`, `u8` `ide`)
函数名称: `void ID2_Filter_Dual(u32 acr2 ,u32 amr2,u8 rtr_acr2,u8 rtr_amr2 ,u8 ide)`
- `u8` `My_CAN_Send_Msg` (`u32` `id`, `u8` `ide`, `u8` `rtr`, `u8` *`msg`, `u8` `len`)
函数名称: `u8 My_CAN_Send_Msg(u32 id,u8 ide,u8 rtr,const u8 *msg,u8 len)`
- `u8` `CAN_Receive_Msg` (`u32` *`id`, `u8` *`ide`, `u8` *`rtr`, `u8` *`buf`)
- `uint8_t` `CAN_ReadState` (`CAN_TypeDef` *`CANx`, `uint32_t` `sr`)
函数名称: `uint8_t CAN_ReadState(CAN_TypeDef* CANx , uint32_t sr)`
- `uint32_t` `CAN_GetIRQFlag` (`void`)
函数名称: `uint32_t CAN_GetIRQFlag(void)`
- void `CAN_Sleep` (`u32` `Baud`)
函数名称: `void CAN.Sleep(u32 Baud)`
- void `CAN_Manual_Awake` (`void`)
函数名称: `void CAN.Manual_Awake(void)`

3.3.1 详细描述

文件名称: [lks32mc08x.can.h](#)
文件标识: 无
内容摘要: CAN外设驱动程序头文件
其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2022年06月1日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x.can.h](#) 中定义.

3.3.2 宏定义说明

3.3.2.1 CAN_ALLSR

```
#define CAN_ALLSR 0xff
```

读取全部状态

记录总线仲裁丢失的具体位置定义

在文件 [lks32mc08x.can.h](#) 第 136 行定义.

3.3.2.2 CAN_DATAAVAIL

```
#define CAN_DATAAVAIL BIT0
```

1: RFIFO 存有一帧或多帧数据, 可以通过 RFIFO 寄存器读取 \ 0: RFIFO 没有有效帧数据

选择读取CAN.SR状态定义

在文件 [lks32mc08x.can.h](#) 第 133 行定义.

3.3.2.3 CAN_ERROV

```
#define CAN_ERROV BIT6
```

1: CAN 传输产生的错误总数达到或超过 CAN_EWL 规定值 \ 0: CAN 传输产生的错误总数低于 CAN_EWL 规定值

选择读取CAN.SR状态定义

在文件 [lks32mc08x.can.h](#) 第 115 行定义.

3.3.2.4 CAN_IE_BUSERR

```
#define CAN_IE_BUSERR BIT7
```

中断使能定义,总线错误中断使能

在文件 [lks32mc08x_can.h](#) 第 88 行定义.

3.3.2.5 CAN_IE_LOSTARB

```
#define CAN_IE_LOSTARB BIT6
```

中断使能定义,丢失仲裁中断使能

在文件 [lks32mc08x_can.h](#) 第 89 行定义.

3.3.2.6 CAN_IE_PASSIVEERR

```
#define CAN_IE_PASSIVEERR BIT5
```

中断使能定义,被动错误错误中断使能

在文件 [lks32mc08x_can.h](#) 第 90 行定义.

3.3.2.7 CAN_IE_RFIFONEMPTY

```
#define CAN_IE_RFIFONEMPTY BIT0
```

中断使能定义,RFIFI0有新的数据被接收到中断使能

在文件 [lks32mc08x_can.h](#) 第 95 行定义.

3.3.2.8 CAN_IE_RFIFOOV

```
#define CAN_IE_RFIFOOV BIT3
```

中断使能定义,RFIFO数据发送溢出中断使能

在文件 [lks32mc08x_can.h](#) 第 92 行定义.

3.3.2.9 CAN_IE_TXDONE

```
#define CAN_IE_TXDONE BIT1
```

中断使能定义,发送完毕当前帧中断使能

在文件 [lks32mc08x_can.h](#) 第 94 行定义.

3.3.2.10 CAN_IE_WAKE

```
#define CAN_IE_WAKE BIT4
```

中断使能定义,CAN模块从休眠中唤醒中断使能

在文件 [lks32mc08x_can.h](#) 第 91 行定义.

3.3.2.11 CAN_IE_WERR

```
#define CAN_IE_WERR BIT2
```

中断使能定义,错误报警中断使能

在文件 [lks32mc08x_can.h](#) 第 93 行定义.

3.3.2.12 CAN_IF_BUSERR

```
#define CAN_IF_BUSERR BIT7
```

总线错误中断标志

在文件 [lks32mc08x_can.h](#) 第 97 行定义.

3.3.2.13 CAN_IF_LOSTARB

```
#define CAN_IF_LOSTARB BIT6
```

丢失仲裁中断标志

在文件 [lks32mc08x_can.h](#) 第 98 行定义.

3.3.2.14 CAN_IF_PASSIVEERR

```
#define CAN_IF_PASSIVEERR BIT5
```

被动错误错误中断标志

在文件 [lks32mc08x_can.h](#) 第 99 行定义.

3.3.2.15 CAN_IF_RFIFONEMPTY

```
#define CAN_IF_RFIFONEMPTY BIT0
```

RFIFIO有新的数据被接收到中断标志

选择读取CAN_SR状态定义

在文件 [lks32mc08x_can.h](#) 第 109 行定义.

3.3.2.16 CAN_IF_RFIFOOV

```
#define CAN_IF_RFIFOOV BIT3
```

RFIFO数据发送溢出中断标志

在文件 [lks32mc08x.can.h](#) 第 101 行定义.

3.3.2.17 CAN_IF_TXDONE

```
#define CAN_IF_TXDONE BIT1
```

发送完毕当前帧中断标志

中断标志定义,注意CAN_IR为读清除寄存器,只有 BIT0–RFIFO.N_EMPTY_IF 无法读清除, 此位只能通过对CAN_CMR BIT2 写 1 清除, 若 RX FIFO 有多帧接收的数据, CAN_CMR BIT2 写 1 也无法清除, 除非RX FIFO 全部读完, 即读一帧数据写 CAN_CMR BIT2, 反复迭代。

在文件 [lks32mc08x.can.h](#) 第 106 行定义.

3.3.2.18 CAN_IF_WAKE

```
#define CAN_IF_WAKE BIT4
```

CAN模块从休眠中唤醒中断标志

在文件 [lks32mc08x.can.h](#) 第 100 行定义.

3.3.2.19 CAN_IF_WERR

```
#define CAN_IF_WERR BIT2
```

错误报警中断标志

在文件 [lks32mc08x.can.h](#) 第 102 行定义.

3.3.2.20 CAN_MONITOR_MODE

```
#define CAN_MONITOR_MODE BIT1
```

CAN工作模式选择,监听模式

在文件 [lks32mc08x.can.h](#) 第 85 行定义.

3.3.2.21 CAN_NORMAL_MODE

```
#define CAN_NORMAL_MODE 0x0
```

CAN工作模式选择,正常收发模式

在文件 [lks32mc08x.can.h](#) 第 86 行定义.

3.3.2.22 CAN_ONBUS

```
#define CAN_ONBUS BIT7
```

1: CAN 模块, 处于 BUS OFF 状态, 无数据发送接收动作 \ 0: CAN 模块, 可以发送数据到 CAN 总线, 或接收 CAN 总线数据

选择读取CAN.SR状态定义

在文件 [lks32mc08x_can.h](#) 第 112 行定义.

3.3.2.23 CAN_RFIFOEMPTY

```
#define CAN_RFIFOEMPTY BIT1
```

1: RFIFO 存入的帧太多, 已满, 导致数据丢失 \ 0: RFIFO 未滿

选择读取CAN.SR状态定义

在文件 [lks32mc08x_can.h](#) 第 130 行定义.

3.3.2.24 CAN_RXING

```
#define CAN_RXING BIT4
```

1: CAN 模块正在接收一帧数据 \ 0: CAN 模块没有接收数据的动作

选择读取CAN.SR状态定义

在文件 [lks32mc08x_can.h](#) 第 121 行定义.

3.3.2.25 CAN_SELFTEST_MODE

```
#define CAN_SELFTEST_MODE BIT2
```

CAN工作模式选择, 自测模式

在文件 [lks32mc08x_can.h](#) 第 84 行定义.

3.3.2.26 CAN_TFIFOEMPTY

```
#define CAN_TFIFOEMPTY BIT2
```

1: TFIFO 空, 可以写入发送数据 \ 0: TFIFO 非空, 内部数据没有发送完毕

选择读取CAN.SR状态定义

在文件 [lks32mc08x_can.h](#) 第 127 行定义.

3.3.2.27 CAN_TRNASDONE

```
#define CAN_TRNASDONE BIT3
```

1: 最近一次传输已完成 \ 0: 最近一次传输未完成

选择读取CAN_SR状态定义

在文件 [lks32mc08x_can.h](#) 第 124 行定义.

3.3.2.28 CAN_TXING

```
#define CAN_TXING BIT5
```

1: CAN 模块正在发送一帧数据 \ 0: CAN 模块没有发送数据的动作

选择读取CAN_SR状态定义

在文件 [lks32mc08x_can.h](#) 第 118 行定义.

3.3.3 枚举类型说明

3.3.3.1 enumErrortime

```
enum enumErrortime
```

枚举值

CAN_RECE_PROCESS	总线错误发生在发送过程中
CAN_SEND_PROCESS	总线错误发生在接收过程中

在文件 [lks32mc08x_can.h](#) 第 164 行定义.

3.3.3.2 enumErrortype

```
enum enumErrortype
```

枚举值

CAN_BITERROR	bit错误
CAN_FORMATERROR	格式错误
CAN_FILLERROR	填充错误
CAN_OTHERERROR	其它类型错误

在文件 [lks32mc08x_can.h](#) 第 155 行定义.

3.3.3.3 enumErrposition

enum `enumErrposition`

枚举值

<code>NULL0</code>	
<code>NULL1</code>	
<code>CAN_ID28_ID21</code>	ID.28 到 ID.21
<code>CAN_START</code>	起始帧
<code>CAN_SRTR</code>	SRTR 位
<code>CAN_IDE</code>	IDE 位
<code>CAN_ID20_ID18</code>	ID.20 到 ID.18
<code>CAN_ID17_ID13</code>	ID.17 到 ID.13
<code>CAN_CRC</code>	CRC 数据段
<code>CAN_KEEP0</code>	保留位, 固定为 0
<code>CAN_DATA_FIELD</code>	Data Field
<code>CAN_DATALENGTHCODE</code>	Data Length Code
<code>CAN_RTR</code>	RTR 位
<code>CAN_KEEP1</code>	保留位, 固定为 1
<code>CAN_ID4_ID0</code>	ID.4 到 ID.0
<code>CAN_ID15_ID12</code>	ID.12 到 ID.5
<code>NULL2</code>	
<code>CAN_ACTIVE_ERROR_FLAG</code>	active error flag
<code>CAN_INTERMISSION</code>	Intermission
<code>CAN_TOLERATE_DOMINANT</code>	Tolerate dominant位
<code>NULL3</code>	
<code>NULL4</code>	
<code>CAN_PASSIVE_ERROR</code>	passive error flag
<code>CAN_ERROR_DELIMITER</code>	Error delimiter
<code>CAN_CRC_DELIMITER</code>	CRC delimiter
<code>CAN_CONNFIRM</code>	确认位
<code>CAN_FRAMEEND</code>	帧结束
<code>CAN_CONNFIRM_DELIMITER</code>	确认 delimiter
<code>CAN_OVERLOAD_FLAG</code>	Overload flag

在文件 `lks32mc08x_can.h` 第 171 行定义。

3.3.3.4 enumLossposition

enum `enumLossposition`

枚举值

<code>CAN_LOST_ID0</code>	ID的第一位
<code>CAN_LOST_ID1</code>	ID的第二位
<code>CAN_LOST_ID2</code>	ID的第三位

枚举值

CAN_LOST_ID3	ID的第四位
CAN_LOST_ID4	ID的第五位
CAN_LOST_ID5	ID的第六位
CAN_LOST_ID6	ID的第七位
CAN_LOST_ID7	ID的第八位
CAN_LOST_ID8	ID的第九位
CAN_LOST_ID9	ID的第十位
CAN_LOST_IDA	ID的第十一位
CAN_LOST_SRTR	ID的SRTR位
CAN_LOST_IDE	ID的IDE位

在文件 [lks32mc08x.can.h](#) 第 137 行定义.

3.3.4 函数说明

3.3.4.1 CAN_GetIRQFlag()

```
uint32_t CAN_GetIRQFlag (
    void )
```

函数名称: `uint32_t CAN_GetIRQFlag(void)`

功能描述: 获取中断标志与清除中断标志位

参见

被应用枚举: [enumErrortype](#) , [enumErrortime](#) , [enumErrposition](#)

参数

输入参数:	无
输出参数:	无

返回

返回 值: TIM中断标志

注解

其它说明: CAN中断标志位为读清除

警告

无

示例代码:

```
u8 CAN_SR 0;  
CAN_SR = CAN_GetIRQFlag(); //获取CAN中断标志位且清除CAN中断标志位
```

Date	Version	Author	Description
2022年04月29日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x.can.c](#) 第 828 行定义.

3.3.4.2 CAN_Init()

```
void CAN_Init (  
    CAN_TypeDef * CANx,  
    CAN_InitTypeDef * CAN_InitStruct )
```

函数名称: void [CAN_Init\(CAN_TypeDef* CANx, CAN_InitTypeDef* CAN_InitStruct\)](#)

功能描述: ADC初始化函数

参见

被调用函数: [SYS_ModuleClockCmd\(\)](#)

参数

输入参数:	CANx: CAN , CAN_InitTypeDef
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
CAN_InitTypeDef CAN_InitStruct;  
CAN_StructInit(&CAN_InitStruct);  
//Can波特率:500Kbps  
CAN_InitStruct.CAN_Btr0 = 0x05;  
CAN_InitStruct.CAN_Btr1 = 0x1C;  
CAN_InitStruct.CAN_DMAEn = DISABLE;           //无DMA搬运数据  
CAN_InitStruct.CAN_WorkMode = CAN_NORMAL_MODE; //CAN正常收发工作模式  
CAN_InitStruct.CAN_ErrWarThre = 50;           //错误与警告门限值为50  
//开启接收中断使能,发送完毕当前帧中断使能,错误报警中断使能  
CAN_InitStruct.IRQEna = CAN_IE_RFIFOEMPTY | CAN_IE_TXDONE | CAN_IE_WERR;  
CAN_Init(CAN, &CAN_InitStruct);
```

Date	Version	Author	Description
2022年01月16日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_can.c](#) 第 46 行定义.

函数调用图:



3.3.4.3 CAN_Manual_Awake()

```
void CAN_Manual_Awake (  
    void )
```

函数名称: void [CAN_Manual_Awake\(void\)](#)

功能描述: CAN手动休眠唤醒函数

参见

被调用函数: 无

参数

输入参数:	无
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
CAN_Manual_Awake(); //唤醒CAN休眠
```

Date	Version	Author	Description
2022年01月16日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_can.c](#) 第 152 行定义.

3.3.4.4 CAN_ReadState()

```
uint8_t CAN_ReadState (
    CAN_TypeDef * CANx,
    uint32_t sr )
```

函数名称: `uint8_t CAN_ReadState(CAN_TypeDef* CANx , uint32_t sr)`

功能描述: 获取CAN状态

参见

被调用函数: 无

参数

输入参数:	sr:读取指定状态或者所有状态结果 <table> <tr> <td>CAN_ONBUS</td><td>1: CAN 模块, 处于 BUS OFF 状态, 无数据发送接收动作 0: CAN 模块, 可以发送数据到 CAN 总线, 或接收 CAN 总线数据</td></tr> <tr> <td>CAN_ERROV</td><td>1: CAN 传输产生的错误总数达到或超过 CAN_EWL 规定值 0: CAN 传输产生的错误总数低于 CAN_EWL 规定值</td></tr> <tr> <td>CAN_TXING</td><td>1: CAN 模块正在发送一帧数据 0: CAN 模块没有发送数据的动作</td></tr> <tr> <td>CAN_RXING</td><td>1: CAN 模块正在接收一帧数据 0: CAN 模块没有接收数据的动作</td></tr> <tr> <td>CAN_TRNASDONE</td><td>1: 最近一次传输已完成 0: 最近一次传输未完成</td></tr> <tr> <td>CAN_TFIFOEMPTY</td><td>1: TFIFO 空, 可以写入发送数据 0: TFIFO 非空, 内部数据没有发送完毕</td></tr> <tr> <td>CAN_RFIFOEMPTY</td><td>1: RFIFO 存入的帧太多, 已满, 导致数据丢失 0: RFIFO 未滿</td></tr> <tr> <td>CAN_DATAAVAIL</td><td>1: RFIFO 存有一帧或多帧数据, 可以通过 RFIFO 寄存器读取 0: RFIFO 没有有效帧数据</td></tr> <tr> <td>CAN_ALLSR</td><td>读取全部状态</td></tr> </table>	CAN_ONBUS	1: CAN 模块, 处于 BUS OFF 状态, 无数据发送接收动作 0: CAN 模块, 可以发送数据到 CAN 总线, 或接收 CAN 总线数据	CAN_ERROV	1: CAN 传输产生的错误总数达到或超过 CAN_EWL 规定值 0: CAN 传输产生的错误总数低于 CAN_EWL 规定值	CAN_TXING	1: CAN 模块正在发送一帧数据 0: CAN 模块没有发送数据的动作	CAN_RXING	1: CAN 模块正在接收一帧数据 0: CAN 模块没有接收数据的动作	CAN_TRNASDONE	1: 最近一次传输已完成 0: 最近一次传输未完成	CAN_TFIFOEMPTY	1: TFIFO 空, 可以写入发送数据 0: TFIFO 非空, 内部数据没有发送完毕	CAN_RFIFOEMPTY	1: RFIFO 存入的帧太多, 已满, 导致数据丢失 0: RFIFO 未滿	CAN_DATAAVAIL	1: RFIFO 存有一帧或多帧数据, 可以通过 RFIFO 寄存器读取 0: RFIFO 没有有效帧数据	CAN_ALLSR	读取全部状态
CAN_ONBUS	1: CAN 模块, 处于 BUS OFF 状态, 无数据发送接收动作 0: CAN 模块, 可以发送数据到 CAN 总线, 或接收 CAN 总线数据																		
CAN_ERROV	1: CAN 传输产生的错误总数达到或超过 CAN_EWL 规定值 0: CAN 传输产生的错误总数低于 CAN_EWL 规定值																		
CAN_TXING	1: CAN 模块正在发送一帧数据 0: CAN 模块没有发送数据的动作																		
CAN_RXING	1: CAN 模块正在接收一帧数据 0: CAN 模块没有接收数据的动作																		
CAN_TRNASDONE	1: 最近一次传输已完成 0: 最近一次传输未完成																		
CAN_TFIFOEMPTY	1: TFIFO 空, 可以写入发送数据 0: TFIFO 非空, 内部数据没有发送完毕																		
CAN_RFIFOEMPTY	1: RFIFO 存入的帧太多, 已满, 导致数据丢失 0: RFIFO 未滿																		
CAN_DATAAVAIL	1: RFIFO 存有一帧或多帧数据, 可以通过 RFIFO 寄存器读取 0: RFIFO 没有有效帧数据																		
CAN_ALLSR	读取全部状态																		
输出参数:	无																		

返回

返回 值: 返回指定状态结果或者所有状态结果

CAN_ONBUS	1: CAN 模块, 处于 BUS OFF 状态, 无数据发送接收动作 0: CAN 模块, 可以发送数据到 CAN 总线, 或接收 CAN 总线数据
CAN_ERROV	1: CAN 传输产生的错误总数达到或超过 CAN_EWL 规定值 0: CAN 传输产生的错误总数低于 CAN_EWL 规定值
CAN_TXING	1: CAN 模块正在发送一帧数据 0: CAN 模块没有发送数据的动作
CAN_RXING	1: CAN 模块正在接收一帧数据 0: CAN 模块没有接收数据的动作
CAN_TRNASDONE	1: 最近一次传输已完成 0: 最近一次传输未完成
CAN_TFIFOEMPTY	1: TFIFO 空, 可以写入发送数据 0: TFIFO 非空, 内部数据没有发送完毕
CAN_RFIFOEMPTY	1: RFIFO 存入的帧太多, 已满, 导致数据丢失 0: RFIFO 未滿
CAN_DATAAVAIL	1: RFIFO 存有一帧或多帧数据, 可以通过 RFIFO 寄存器读取 0: RFIFO 没有有效帧数据
CAN_ALLSR	读取全部状态

注解

其它说明： 无

警告

无

示例代码:

```
//判断CAN的RFIFO是否存在一帧或多帧数据，此时是否可以通过RFIFO寄存器读取
if(CAN_ReadState(CAN , CAN_DATAAVAIL) == CAN_DATAAVAIL)
{
}
```

Date	Version	Author	Description
2022年04月29日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_can.c](#) 第 669 行定义.

3.3.4.5 CAN_Receive_Msg()

```
u8 CAN_Receive_Msg (
    u32 * id,
    u8 * ide,
    u8 * rtr,
    u8 * buf )
```

3.3.4.6 CAN_Sleep()

```
void CAN_Sleep (
    u32 Baud )
```

函数名称: void [CAN_Sleep\(u32 Baud\)](#)

功能描述: CAN开启休眠函数

参见

被调用函数： 无

参数

输入参数:	Baud: 波特率, 单位Kbps
输出参数:	无

返回

返回值: 无

注解

其它说明: 休眠唤醒有两种: 1、软件清零CAN_MOD[4], 即自动退出休眠模式。 , 2、一旦检测到CAN总线的活动, 即自动退出休眠模式。

警告

无

示例代码:

```
CAN_Sleep(500); //开启CAN休眠, CAN比特率500kbps
```

Date	Version	Author	Description
2022年01月16日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_can.c](#) 第 115 行定义.

3.3.4.7 CAN_StructInit()

```
void CAN_StructInit (
    CAN_InitTypeDef * CAN_InitStruct )
```

函数名称: void [CAN_StructInit\(CAN_InitTypeDef *CAN_InitStruct\)](#)

功能描述: CAN结构体初始化

参见

被调用函数: 无

参数

输入参数:	CAN_InitTypeDef
输出参数:	无

返回

返回值: 无

注解

其它说明： 无

警告

无

示例代码:

```
CAN_InitTypeDef CAN_InitStructure;  
CAN_StructInit (&CAN_InitStructure); //初始化结构体
```

Date	Version	Author	Description
2022年01月16日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_can.c](#) 第 84 行定义.

3.3.4.8 EFF_ID_Filter()

```
void EFF_ID_Filter (  
    u32 acr,  
    u32 amr,  
    u8 rtr_acr,  
    u8 rtr_amr )
```

函数名称: void [EFF_ID_Filter\(u32 acr,u32 amr,u8 rtr_acr,u8 rtr_amr\)](#)

功能描述: CAN扩展帧单ID过滤,采用2.0B协议

参见

被调用函数： 无

参数

输入参数:	acr: ID值。 amr: 屏蔽位 rtr_acr: 接收（1远程/0数据帧）帧类型选择 rtr_amr: 远程/数据帧滤波屏蔽位，0表示只接收rtr_acr类型帧，为1表示远程/0标志帧均接收
输出参数:	无

返回

返回值: 无

注解

其它说明: 1、扩展帧 29位ID长度
2、amr: 屏蔽位对应位0表示此为对应ID必须匹配才会接收

警告

无

示例代码:

示例1:

```
//扩展帧接收滤波: ID:0X10005A5A ,数据帧
EFF_ID_Filter(0X10005A5A, 0x00, 0x0, 0x0);
//接收的扩展帧只有满足ID地址为0X10005A5A且为数据帧才会接收
```

示例2:

```
//扩展帧接收滤波: ID:0X10005A5A ,数据帧和远程帧 (因为rtr.amr为1, 所以接收帧类型不需要匹配也可以接收)
EFF_ID_Filter(0X10005A5A, 0x00, 0x0, 0x1);
//接收的扩展帧只有满足ID地址为0X10005A5A的数据帧和远程帧才会接收
```

Date	Version	Author	Description
2022年01月16日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_can.c](#) 第 252 行定义.

3.3.4.9 ID1_Filter_Dual()

```
void ID1_Filter_Dual (
    u32 acr1,
    u32 amr1,
    u8 rtr_acr1,
    u8 rtr_amr1,
    u8 Byte_acr1,
    u8 Byte_amr1,
    u8 ide )
```

函数名称: void ID1_Filter_Dual(u32 acr1 ,u32 amr1,u8 rtr_acr1, u8 rtr_amr1,u8 Byte_acr1,u8 Byte_amr1,u8 ide)

功能描述: CAN双ID过滤中ID1滤波设置 采用2.0B协议

参见

被调用函数: 无

参数

输入参数:	acr1: ID1值（标志帧ID11位，扩展帧ID范围1D28~ID13即ID高16位有效） amr1: ID1屏蔽位 rtr_acr1: 接收（1远程/0数据帧）帧类型选择（只支持SFF模式，扩展帧配置无效） rtr_amr1: 远程/标志帧滤波屏蔽位，0表示只接收rtr_acr类型帧，为1表示远程/0标志帧均接收（只支持SFF模式，扩展帧配置无效） Byte_acr1: 数据帧第1个Byte低4位滤波选择（只支持SFF模式，扩展帧配置无效） Byte_amr1: 数据帧第1个Byte低4位屏蔽位（只支持SFF模式，扩展帧配置无效） ide: 0标准(SFF模式)/1扩展帧
输出参数:	无

返回

返回 值: 无

注解

其它说明: 1、SFF模式为标准帧模式
2、扩展帧时，只对ID的bit13- bit28即ID高16位进行筛选过滤
3、因为双滤波ID1函数兼容标志准和扩展帧滤波（标准帧和扩展帧的双滤波有区别），所以该函数的形参部分只支持标准帧模式配置使用，扩展帧模式配置无效。

警告

无

示例代码:

```
示例1:
//标准帧ID1接收滤波: ID:0X75A ,数据帧
ID1_Filter_Dual(0X75A, 0x00, 0x0, 0x0, 0x0F, 0x00, 0);
//接收的标准帧只有满足ID地址为0X75A且为数据帧且接收数据（不是帧的第一个数据，而是数据区的第一个数据）的第1个
Byte低4位为0XF才会接收。
示例2:
//扩展帧ID1接收滤波: ID:0XFECB的扩展帧
ID1_Filter_Dual(0XFECB, 0x00, 0x0, 0x0, 0x00, 0x00,1);
//接收的扩展帧只有满足ID地址为0XFECB才会接收
```

Date	Version	Author	Description
2022年01月16日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_can.c](#) 第 319 行定义.

3.3.4.10 ID2_Filter_Dual()

```
void ID2_Filter_Dual (
    u32 acr2,
```



```
u32 amr2,
u8 rtr_acr2,
u8 rtr_amr2,
u8 ide )
```

函数名称: void ID2.Filter_Dual(u32 acr2 ,u32 amr2,u8 rtr_acr2,u8 rtr_amr2 ,u8 ide)

功能描述: CAN双ID过滤中ID2滤波设置 采用2.0B协议

参见
被调用函数: 无

参数

输入参数:	acr2: ID2值（范围1D28~ID13即ID高16位有效） amr2: ID2屏蔽位 rtr_acr2: 接收（1远程/0数据帧）帧类型选择（只支持SFF标准帧模式，扩展帧配置无效） rtr_amr2: 远程/标志帧滤波屏蔽位，0表示只接收rtr_acr类型帧，为1表示远程/0标志 帧均接收（只支持SFF标准帧模式，扩展帧配置无效）
输出参数:	无

返回
返回值: 无

注解
其它说明: 1、SFF模式为标准帧模式
2、扩展帧时，只对ID的bit13- bit28即ID高16位进行筛选过滤
3、因为双滤波ID1函数兼容标志准和扩展帧滤波（标准帧和扩展帧的双滤波有区别），所以该函数的形参部分只支持标准帧模式配置使用，扩展帧模式配置无效。

警告
无

示例代码:

```
示例1:
//标准帧ID2接收滤波: ID:0X75A ,数据帧
ID2.Filter_Dual(0X75A, 0x00, 0x0, 0x0, 0);
//接收的标准帧只有满足ID地址为0X75A且为数据帧才会接收。

示例2:
//扩展帧ID2接收滤波: ID:0XFECB的扩展帧
ID2.Filter_Dual(0XFECB, 0x00, 0x0, 0x0, 1);
//接收的扩展帧只有满足ID地址为0XFECB才会接收
```

Date	Version	Author	Description
2022年01月16日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_can.c](#) 第 386 行定义.

3.3.4.11 My_CAN_Send_Msg()

```
u8 My_CAN_Send_Msg (
    u32 id,
    u8 ide,
    u8 rtr,
    u8 * msg,
    u8 len )
```

函数名称: u8 My_CAN_Send_Msg(u32 id,u8 ide,u8 rtr,const u8 *msg,u8 len)

功能描述: CAN轮询发送数据程序 采用2.0B协议

参见
被调用函数: 无

参数

输入参数:	id: 发送id ide: 标准/扩展帧 rtr: 远程帧/数据帧 *msg: 发送的数据 len:发送字节长度
输出参数:	无

返回

返回 值: 0: 发送成功 !0: 发送失败

注解

其它说明: 1、id:标准ID(11位)/扩展ID(11位+18位)
2、ide:0,标准帧;1,扩展帧
3、rtr:0,数据帧;1,远程帧。

警告
无

示例代码:

```
示例1:
u8 Can_TX[8] = {0x81, 0x23, 0x33, 0x44, 0x15, 0x26, 0x37, 0x48};
// 发送数据 ID:0x03,标准帧,数据帧,数据,数据长度8字节
My_CAN_Send_Msg(0x5A, 0, 0, Can_TX, 8);

示例2:
u8 Can_TX[8] = {0x81, 0x23, 0x33, 0x44, 0x15, 0x26, 0x37, 0x48};
// 发送数据 ID:0x03,扩展帧,数据帧,数据,数据长度8字节
My_CAN_Send_Msg(0X10005A5A, 1, 0, Can_TX, 8);
```

Date	Version	Author	Description
2018年08月10日	1.0	ZKW	创建

修改日志:

Date	Version	Author	Description
2022年01月16日	1.0	HuangMG	创建

修改日志:增加自测模式发送数据功能

在文件 [lks32mc08x_can.c](#) 第 455 行定义.

3.3.4.12 SFF_ID.Filter()

```
void SFF_ID_Filter (
    u32 acr,
    u32 amr,
    u8 rtr_acr,
    u8 rtr_amr,
    u8 Byte1_acr,
    u8 Byte1_amr,
    u8 Byte2_acr,
    u8 Byte2_amr )
```

函数名称: void [SFF_ID_Filter\(u32 acr,u32 amr,u8 rtr_acr,u8 rtr_amr, u8 Byte1_acr,u8 Byte1_amr,u8 Byte2_acr,u8 Byte2_amr\)](#)

功能描述: CAN标准帧单ID过滤,采用2.0B协议

参见

被调用函数: 无

参数

输入参数:	acr: ID值。 amr: 屏蔽位 rtr_acr: 接收（1远程/0数据帧）帧类型选择 rtr_amr: 远程/数据帧滤波屏蔽位，0表示只接收rtr_acr类型帧，为1表示远程/0标志帧均接收 Byte1_acr: 数据第一个字节 Byte1_amr: 第一个字节屏蔽位 Byte2_acr: 数据第二个字节 Byte2_amr: 第二个字节屏蔽位
输出参数:	无

返回

返回值: 无

注解

其它说明: 1、标准帧 11位ID长度
2、amr: 屏蔽位对应位0表示此为对应ID必须匹配才会接收

警告

无

示例代码:

示例1:

```
//标准帧接收单滤波: ID:0x5A ,数据帧, BYTE1:0x81, BYTE:0x23
SFF_ID_Filter(0x5A, 0x00, 0x0, 0x0, 0x81, 0x00, 0x23, 0x00);
//接收的标准帧只有满足ID地址为0x5A的数据帧且第一个数据字节为0x81, 第二个字节为0x23才会接收
```

示例2:

```
//标准帧接收单滤波: ID:0x5A和0x5B (因为amr第一位为1, 所以第一位可以为1或0, 无需匹配即可接收) ,数据帧, BYTE1:0x81, BYTE:0x23
SFF_ID_Filter(0x5A, 0x01, 0x0, 0x0, 0x81, 0x00, 0x23, 0x00);
//接收的标准帧只有满足ID地址为0x5A和0x5B的数据帧且第一个数据字节为0x81, 第二个字节为0x23才会接收
```

Date	Version	Author	Description
2022年01月16日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_can.c](#) 第 196 行定义.

3.4 lks32mc08x_can.h

[浏览该文件的文档.](#)

```
00001
00014 #ifndef __LKS32MC08x_CAN_H_
00015 #define __LKS32MC08x_CAN_H_
00016 /* Includes -----*/
00017 #include "lks32mc08x_lib.h"
00018
00022 typedef struct
00023 {
00024     __IO uint32_t MOD;
00025     __IO uint32_t CMR;
00026     __IO uint32_t SR;
00027     __IO uint32_t IR;
00029     __IO uint32_t EIR;
00030     __IO uint32_t null0;
00031     __IO uint32_t BTR0;
00032     __IO uint32_t BTR1;
00034     __IO uint32_t null1;
00035     __IO uint32_t null2;
00036     __IO uint32_t null3;
00037     __IO uint32_t ALC;
00039     __IO uint32_t ECC;
00040     __IO uint32_t EWLR;
```

```

00041     __IO uint32_t RXERR;
00042     __IO uint32_t TXERR;
00044     __IO uint32_t TXRX0;
00045     __IO uint32_t TXRX1;
00046     __IO uint32_t TXRX2;
00047     __IO uint32_t TXRX3;
00049     __IO uint32_t TXRX4;
00050     __IO uint32_t TXRX5;
00051     __IO uint32_t TXRX6;
00052     __IO uint32_t TXRX7;
00054     __IO uint32_t TXRX8;
00055     __IO uint32_t TXRX9;
00056     __IO uint32_t TXRXA;
00057     __IO uint32_t TXRXB;
00059     __IO uint32_t TXRXC;
00060     __IO uint32_t RMC;
00061     __IO uint32_t RBSA;
00062 } CAN_TypeDef;
00063
00067 #ifndef CAN
00068 #define CAN ((CAN_TypeDef *)CAN_BASE)
00069 #endif
00070
00074 typedef struct
00075 {
00076     uint32_t CAN_WorkMode;
00077     uint32_t CAN_DMAEn;
00078     uint32_t CAN_ErrWarThre;
00079     uint32_t CAN_Btr0;
00080     uint32_t CAN_Btr1;
00081     uint32_t IRQEna;
00082 } CAN_InitTypeDef;
00083
00084 #define CAN_SELFTEST_MODE    BIT2
00085 #define CAN_MONITOR_MODE    BIT1
00086 #define CAN_NORMAL_MODE    0x0
00088 #define CAN_IE_BUSERR    BIT7
00089 #define CAN_IE_LOSTARB    BIT6
00090 #define CAN_IE_PASSIVEERR    BIT5
00091 #define CAN_IE_WAKE    BIT4
00092 #define CAN_IE_RFIFOOV    BIT3
00093 #define CAN_IE_WERR    BIT2
00094 #define CAN_IE_TXDONE    BIT1
00095 #define CAN_IE_RFIFONEMPTY    BIT0
00097 #define CAN_IF_BUSERR    BIT7
00098 #define CAN_IF_LOSTARB    BIT6
00099 #define CAN_IF_PASSIVEERR    BIT5
00100 #define CAN_IF_WAKE    BIT4
00101 #define CAN_IF_RFIFOOV    BIT3
00102 #define CAN_IF_WERR    BIT2
00103 #define CAN_IF_TXDONE    BIT1
00107 #define CAN_IF_RFIFONEMPTY    BIT0
00110 #define CAN_ONBUS    BIT7
00113 #define CAN_ERROV    BIT6
00116 #define CAN_TXING    BIT5
00119 #define CAN_RXING    BIT4
00122 #define CAN_TRNASDONE    BIT3
00125 #define CAN_TFIFOEMPTY    BIT2
00128 #define CAN_RFIFOEMPTY    BIT1
00131 #define CAN_DATAAVAIL    BIT0
00134 #define CAN_ALLSR    0xff
00137 typedef enum
00138 {
00139     CAN_LOST_ID0,
00140     CAN_LOST_ID1,
00141     CAN_LOST_ID2,
00142     CAN_LOST_ID3,
00143     CAN_LOST_ID4,
00144     CAN_LOST_ID5,
00145     CAN_LOST_ID6,
00146     CAN_LOST_ID7,
00147     CAN_LOST_ID8,
00148     CAN_LOST_ID9,
00149     CAN_LOST_IDA,
00150     CAN_LOST_SRTR,
00151     CAN_LOST_IDE,
00152 } enumLossposition;
00153
00155 typedef enum
00156 {
00157     CAN_BITERROR,
00158     CAN_FORMATERROR,
00159     CAN_FILERROR,
00160     CAN_OTHERERROR,
00161 } enumErrortype;
00162
00164 typedef enum

```

```

00165 {
00166     CAN_RECE_PROCESS,
00167     CAN_SEND_PROCESS,
00168 } enumErrortime;
00169
00171 typedef enum
00172 {
00173     null0,
00174     null1,
00175     CAN_ID28_ID21,
00176     CAN_START,
00177     CAN_SRTR,
00178     CAN_IDE,
00179     CAN_ID20_ID18,
00180     CAN_ID17_ID13,
00181     CAN_CRC,
00182     CAN_KEEPO,
00183     CAN_DATA_FIELD,
00184     CAN_DATALENGTHCODE,
00185     CAN_RTR,
00186     CAN_KEEPI,
00187     CAN_ID4_ID0,
00188     CAN_ID15_ID12,
00189     null2,
00190     CAN_ACTIVE_ERROR_FLAG,
00191     CAN_INTERMISSION,
00192     CAN_TOLERATE_DOMINANT,
00193     null3,
00194     null4,
00195     CAN_PASSIVE_ERROR,
00196     CAN_ERROR_DELIMITER,
00197     CAN_CRC_DELIMITER,
00198     CAN_CONNFIRM,
00199     CAN_FRAMEEND,
00200     CAN_CONNFIRM_DELIMITER,
00201     CAN_OVERLOAD_FLAG,
00202 } enumErrposition;
00203
00204 void CAN_StructInit(CAN_InitTypeDef *CAN_InitStruct);
00205 void CAN_Init(CAN_TypeDef *CANx, CAN_InitTypeDef *CAN_InitStruct);
00206 void SFF_ID_Filter(u32 acr, u32 amr, u8 rtr_acr, u8 rtr_amr, u8 Byte1_acr, u8 Byte1_amr, u8 Byte2_acr, u8
    Byte2_amr);
00207 void EFF_ID_Filter(u32 acr, u32 amr, u8 rtr_acr, u8 rtr_amr);
00208 void ID1_Filter_Dual(u32 acr1, u32 amr1, u8 rtr_acr1, u8 rtr_amr1, u8 Byte_acr1, u8 Byte_amr1, u8 ide);
00209 void ID2_Filter_Dual(u32 acr2, u32 amr2, u8 rtr_acr2, u8 rtr_amr2, u8 ide);
00210 u8 My_CAN_Send_Msg(u32 id, u8 ide, u8 rtr, u8 *msg, u8 len);
00211 u8 CAN_Receive_Msg(u32 *id, u8 *ide, u8 *rtr, u8 *buf);
00212 uint8_t CAN_ReadState(CAN_TypeDef *CANx, uint32_t sr);
00213 uint32_t CAN_Get_IRQ_Flag(void);
00214 void CAN_Sleep(u32 Baud);
00215 void CAN_Manual_Awake(void);
00216
00217 #endif

```

3.5 include/lks32mc08x_cmp.h 文件参考

文件名称: [lks32mc08x_cmp.h](#)

文件标识: 无

内容摘要: CMP外设驱动程序头文件

其它说明: 无

#include "lks32mc08x_lib.h"

lks32mc08x_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_TypeDef**
*CMP*寄存器结构体句柄
- struct **CMP_InitTypeDef**
*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

宏定义

- #define **CMP_HYS_20mV** 0
 < 比较器回差选择
- #define **CMP_HYS_0mV** 1
 0mV
- #define **CMP_SELN_IN** 0
 比较器*N*端，*N*端输入
- #define **CMP_SELN_REF** 1
 比较器*N*端，*N*端内部参考*REF*输入
- #define **CMP_SELN_DAC** 2
 比较器*N*端，*N*端内部*DAC*输入
- #define **CMP_SELN_HALL_MID** 3
 比较器*N*端，*N*端反电势过零中点输入
- #define **CMP0_SELP_IP0** 0
 *P*端*IP0*输入
- #define **CMP0_SELP_OPA0_IP** 1
 *P*端*OPA0 IP*输入
- #define **CMP0_SELP_OPA0_OUT** 2
 *P*端*OPA0 OUT*
- #define **CMP0_SELP_OPA1_OUT** 3
 *P*端*OPA1 OUT*
- #define **CMP0_SELP_IP1** 4
 *P*端*IP1*输入
- #define **CMP0_SELP_IP2** 5
 *P*端*IP2*输入
- #define **CMP0_SELP_IP3** 6
 *P*端*IP3*输入
- #define **CMP0_SELP_IP4** 7
 *P*端*IP4*输入
- #define **CMP1_SELP_IP0** 0
 *P*端*IP0*输入
- #define **CMP1_SELP_OPA3_IP** 1
 *P*端*OPA3 IP*输入
- #define **CMP1_SELP_OPA2_OUT** 2
 *P*端*OPA2 OUT*

- `#define CMP1_SELPA3_OUT 3`
P端OPA3 OUT
- `#define CMP1_SELPIP1 4`
P端IP1输入
- `#define CMP1_SELPIP2 5`
P端IP2输入
- `#define CMP1_SELPIP3 6`
P端IP3输入
- `#define CMP1_SELPAVSS 7`
芯片地
- `#define SELN_CMP_IN 0`
N端输入（为保证库兼容性，保留修改的宏定义）
- `#define SELN_REF 1`
N端内部参考REF输入（为保证库兼容性，保留修改的宏定义）
- `#define SELN_DAC 2`
N端内部DAC输入（为保证库兼容性，保留修改的宏定义）
- `#define SELN_HALL_MID 3`
N端反电势过零中点输入（为保证库兼容性，保留修改的宏定义）
- `#define SELP_CMP_OPA0IP 1`
P端OPA0_IP输入（为保证库兼容性，保留修改的宏定义）
- `#define SELP_CMP_OPA0OUT 2`
P端OPA0_OUT输入（为保证库兼容性，保留修改的宏定义）
- `#define SELP_CMP_OPA1OUT 3`
P端OPA1_OUT输入（为保证库兼容性，保留修改的宏定义）
- `#define SELP_CMP_OPA3IP 1`
P端OPA3_IP输入（为保证库兼容性，保留修改的宏定义）
- `#define SELP_CMP_OPA2OUT 2`
P端OPA2_OUT输入（为保证库兼容性，保留修改的宏定义）
- `#define SELP_CMP_OPA3OUT 3`
P端OPA3_OUT输入（为保证库兼容性，保留修改的宏定义）
- `#define SELP_CMP_IP0 0`
P端IP0输入（为保证库兼容性，保留修改的宏定义）
- `#define SELP_CMP_IP1 4`
P端IP1输入（为保证库兼容性，保留修改的宏定义）
- `#define SELP_CMP_IP2 5`
P端IP2输入（为保证库兼容性，保留修改的宏定义）
- `#define SELP_CMP_IP3 6`
P端IP3输入（为保证库兼容性，保留修改的宏定义）
- `#define SELP_CMP_IP4 7`
P端IP4输入（为保证库兼容性，保留修改的宏定义）
- `#define CMP0 0`
CMP结构体基地址定义
- `#define CMP1 1`
比较器1

函数

- void **CMP_Init** (CMP_InitTypeDef *)
函数名称: *CMP_Init(CMP_InitTypeDef* this)*
- void **CMP_StructInit** (CMP_InitTypeDef *)
函数名称: *void CMP_StructInit(CMP_InitTypeDef* CMP_InitStruct)*
- void **CMP_ClearIRQFlag** (u8 CMPx)
函数名称: *void CMP_ClearIRQFlag(u8 CMPx)*
- u8 **CMP_GetIRQFlag** (u8 CMPx)
函数名称: *u8 CMP_GetIRQFlag(u8 CMPx)*

3.5.1 详细描述

文件名称: [lks32mc08x_cmp.h](#)
 文件标识: 无
 内容摘要: CMP外设驱动程序头文件
 其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2018年07月25日	1.0	William Zhang	创建

修改日志:

Date	Version	Author	Description
2022年04月26日	2.0	YangZJ	创建

修改日志:增加开窗配置、增加比较器中断相关函数

在文件 [lks32mc08x_cmp.h](#) 中定义.

3.5.2 宏定义说明

3.5.2.1 CMP0

```
#define CMP0 0
```

CMP结构体基地址定义

比较器0

在文件 [lks32mc08x_cmp.h](#) 第 93 行定义.

3.5.2.2 CMP0_SELP_IP0

```
#define CMP0_SELP_IP0 0
```

P端IP0输入

在文件 [lks32mc08x_cmp.h](#) 第 35 行定义.

3.5.2.3 CMP0_SELP_IP1

```
#define CMP0_SELP_IP1 4
```

P端IP1输入

在文件 [lks32mc08x_cmp.h](#) 第 39 行定义.

3.5.2.4 CMP0_SELP_IP2

```
#define CMP0_SELP_IP2 5
```

P端IP2输入

在文件 [lks32mc08x_cmp.h](#) 第 40 行定义.

3.5.2.5 CMP0_SELP_IP3

```
#define CMP0_SELP_IP3 6
```

P端IP3输入

在文件 [lks32mc08x_cmp.h](#) 第 41 行定义.

3.5.2.6 CMP0_SELP_IP4

```
#define CMP0_SELP_IP4 7
```

P端IP4输入

在文件 [lks32mc08x_cmp.h](#) 第 42 行定义.

3.5.2.7 CMP0_SELP_OPA0_IP

```
#define CMP0_SELP_OPA0_IP 1
```

P端OPA0 IP输入

在文件 [lks32mc08x_cmp.h](#) 第 36 行定义.

3.5.2.8 CMP0_SELP_OPA0_OUT

```
#define CMP0_SELP_OPA0_OUT 2
```

P端OPA0 OUT

在文件 [lks32mc08x_cmp.h](#) 第 37 行定义.

3.5.2.9 CMP0_SELP_OPA1_OUT

```
#define CMP0_SELP_OPA1_OUT 3
```

P端OPA1 OUT

在文件 [lks32mc08x_cmp.h](#) 第 38 行定义.

3.5.2.10 CMP1

```
#define CMP1 1
```

比较器1

在文件 [lks32mc08x_cmp.h](#) 第 97 行定义.

3.5.2.11 CMP1_SELP_AVSS

```
#define CMP1_SELP_AVSS 7
```

芯片地

在文件 [lks32mc08x_cmp.h](#) 第 52 行定义.

3.5.2.12 CMP1_SELP_IP0

```
#define CMP1_SELP_IP0 0
```

P端IP0输入

在文件 [lks32mc08x_cmp.h](#) 第 45 行定义.

3.5.2.13 CMP1_SELP_IP1

```
#define CMP1_SELP_IP1 4
```

P端IP1输入

在文件 [lks32mc08x_cmp.h](#) 第 49 行定义.

3.5.2.14 CMP1_SELP_IP2

```
#define CMP1_SELP_IP2 5
```

P端IP2输入

在文件 [lks32mc08x_cmp.h](#) 第 50 行定义.

3.5.2.15 CMP1_SELP_IP3

```
#define CMP1_SELP_IP3 6
```

P端IP3输入

在文件 [lks32mc08x_cmp.h](#) 第 51 行定义.

3.5.2.16 CMP1_SELP_OPA2_OUT

```
#define CMP1_SELP_OPA2_OUT 2
```

P端OPA2 OUT

在文件 [lks32mc08x_cmp.h](#) 第 47 行定义.

3.5.2.17 CMP1_SELP_OPA3_IP

```
#define CMP1_SELP_OPA3_IP 1
```

P端OPA3 IP输入

在文件 [lks32mc08x_cmp.h](#) 第 46 行定义.

3.5.2.18 CMP1_SELP_OPA3_OUT

```
#define CMP1_SELP_OPA3_OUT 3
```

P端OPA3 OUT

在文件 [lks32mc08x_cmp.h](#) 第 48 行定义.

3.5.2.19 CMP_HYS_0mV

```
#define CMP_HYS_0mV 1
```

0mV

在文件 [lks32mc08x_cmp.h](#) 第 26 行定义.

3.5.2.20 CMP_HYS_20mV

```
#define CMP_HYS_20mV 0
```

< 比较器回差选择

20mV

在文件 [lks32mc08x_cmp.h](#) 第 25 行定义.

3.5.2.21 CMP_SELN_DAC

```
#define CMP_SELN_DAC 2
```

比较器N端，N端内部DAC输入

在文件 [lks32mc08x_cmp.h](#) 第 31 行定义.

3.5.2.22 CMP_SELN_HALL_MID

```
#define CMP_SELN_HALL_MID 3
```

比较器N端，N端反电势过零中点输入

在文件 [lks32mc08x_cmp.h](#) 第 32 行定义.

3.5.2.23 CMP_SELN_IN

```
#define CMP_SELN_IN 0
```

比较器N端，N端输入

在文件 [lks32mc08x_cmp.h](#) 第 29 行定义.

3.5.2.24 CMP_SELN_REF

```
#define CMP_SELN_REF 1
```

比较器N端，N端内部参考REF输入

在文件 [lks32mc08x_cmp.h](#) 第 30 行定义.

3.5.2.25 SELN_CMP_IN

```
#define SELN_CMP_IN 0
```

N端输入（为保证库兼容性，保留修改的宏定义）

在文件 [lks32mc08x_cmp.h](#) 第 55 行定义.

3.5.2.26 SELN_DAC

```
#define SELN_DAC 2
```

N端内部DAC输入（为保证库兼容性，保留修改的宏定义）

在文件 [lks32mc08x_cmp.h](#) 第 57 行定义.

3.5.2.27 SELN_HALL_MID

```
#define SELN_HALL_MID 3
```

N端反电势过零中点输入（为保证库兼容性，保留修改的宏定义）

在文件 [lks32mc08x_cmp.h](#) 第 58 行定义.

3.5.2.28 SELN_REF

```
#define SELN_REF 1
```

N端内部参考REF输入（为保证库兼容性，保留修改的宏定义）

在文件 [lks32mc08x_cmp.h](#) 第 56 行定义.

3.5.2.29 SELP_CMP_IP0

```
#define SELP_CMP_IP0 0
```

P端IP0输入（为保证库兼容性，保留修改的宏定义）

在文件 [lks32mc08x_cmp.h](#) 第 66 行定义.

3.5.2.30 SELP_CMP_IP1

```
#define SELP_CMP_IP1 4
```

P端IP1输入（为保证库兼容性，保留修改的宏定义）

在文件 [lks32mc08x_cmp.h](#) 第 67 行定义.

3.5.2.31 SELP_CMP_IP2

```
#define SELP_CMP_IP2 5
```

P端IP2输入（为保证库兼容性，保留修改的宏定义）

在文件 [lks32mc08x_cmp.h](#) 第 68 行定义.

3.5.2.32 SELP_CMP_IP3

```
#define SELP_CMP_IP3 6
```

P端IP3输入（为保证库兼容性，保留修改的宏定义）

在文件 [lks32mc08x_cmp.h](#) 第 69 行定义.

3.5.2.33 SELP_CMP_IP4

```
#define SELP_CMP_IP4 7
```

P端IP4输入（为保证库兼容性，保留修改的宏定义）

在文件 [lks32mc08x_cmp.h](#) 第 70 行定义.

3.5.2.34 SELP_CMP_OPA0IP

```
#define SELP_CMP_OPA0IP 1
```

P端OPA0_IP输入（为保证库兼容性，保留修改的宏定义）

在文件 [lks32mc08x_cmp.h](#) 第 60 行定义.

3.5.2.35 SELP_CMP_OPA0OUT

```
#define SELP_CMP_OPA0OUT 2
```

P端OPA0_OUT输入（为保证库兼容性，保留修改的宏定义）

在文件 [lks32mc08x_cmp.h](#) 第 61 行定义.

3.5.2.36 SELP_CMP_OPA1OUT

```
#define SELP_CMP_OPA1OUT 3
```

P端OPA1_OUT输入（为保证库兼容性，保留修改的宏定义）

在文件 [lks32mc08x_cmp.h](#) 第 62 行定义.

3.5.2.37 SELP_CMP_OPA2OUT

```
#define SELP_CMP_OPA2OUT 2
```

P端OPA2_OUT输入（为保证库兼容性，保留修改的宏定义）

在文件 [lks32mc08x_cmp.h](#) 第 64 行定义.

3.5.2.38 SELP_CMP_OPA3IP

#define SELP_CMP_OPA3IP 1

P端OPA3_IP输入（为保证库兼容性，保留修改的宏定义）

在文件 [lks32mc08x_cmp.h](#) 第 63 行定义.

3.5.2.39 SELP_CMP_OPA3OUT

#define SELP_CMP_OPA3OUT 3

P端OPA3_OUT输入（为保证库兼容性，保留修改的宏定义）

在文件 [lks32mc08x_cmp.h](#) 第 65 行定义.

3.5.3 函数说明

3.5.3.1 CMP_ClearIRQFlag()

void CMP_ClearIRQFlag (
 u8 CMPx)

函数名称: void [CMP_ClearIRQFlag\(u8 CMPx\)](#)

功能描述: 清除TIM中断标志

参见

被调用函数: 无

参数

输入参数:	CMP0, CMP1即获取CMP0和CMP1中断标志位
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
if(CMP_GetIRQFlag(CMP1))//获取CMP1中断标志位
{
    CMP_ClearIRQFlag(CMP1); //清除CMP1中断标志位
}
```

Date	Version	Author	Description
2022年4月11日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_cmp.c](#) 第 197 行定义.

3.5.3.2 CMP_GetIRQFlag()

```
u8 CMP_GetIRQFlag (
    u8 CMPx )
```

函数名称: u8 [CMP_GetIRQFlag\(u8 CMPx\)](#)

功能描述: 读取比较器中断

参见

被调用函数: 无

参数

输入参数:	CMP0, CMP1即获取CMP0和CMP1中断标志位
输出参数:	无

返回

返回值: TIM中断标志0/1

注解

其它说明: 无

警告

无

示例代码:

```
if(CMP_GetIRQFlag(CMP1))//获取CMP1中断标志位
{
    CMP_ClearIRQFlag(CMP1); //清除CMP1中断标志位
}
```

Date	Version	Author	Description
2022年4月21日	1.0	Yangzj	创建

修改日志:

在文件 [lks32mc08x_cmp.c](#) 第 148 行定义.

3.5.3.3 CMP_Init()

```
void CMP_Init (
    CMP_InitTypeDef * this )
```

函数名称: [CMP_Init\(CMP_InitTypeDef* this\)](#)

功能描述: CMP初始化

参见

被调用函数: [SYS_AnalogModuleClockCmd\(\)](#)

参数

输入参数:	CMP_InitTypeDef
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
CMP_InitTypeDef CMP_InitStructure;
CMP_StructInit (&CMP_InitStructure);
CMP_InitStructure.CLK10_EN           = ENABLE           ; // 比较器 1/0 滤波时钟使能
CMP_InitStructure.FIL_CLK10_DIV16    = 15               ; // 比较器 1/0 滤波系数      0~0xF
CMP_InitStructure.FIL_CLK10_DIV2     = 0                ; // 比较器 1/0 滤波移位系数    0~0x3    最终分频系数为
(DIV16+1) << DIV2
CMP_InitStructure.CMP_FT              = DISABLE         ; // 比较器快速比较使能
CMP_InitStructure.CMP_HYS            = 0                ; // 比较器回差选择          0: 回差20mV 1: 回差0mV

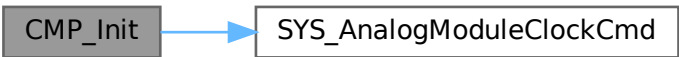
CMP_InitStructure.CMP1_EN            = ENABLE           ; // 比较器 1 使能
CMP_InitStructure.CMP1_IN_EN         = ENABLE           ; // 比较器 1 信号输入使能 影响比较器数字模块信号, 模拟部
分不受影响
CMP_InitStructure.CMP1_SELFP         = CMP1_SELFP_IP2   ; // 比较器 1 信号正端选择 见 CMP1_SELFP_CMP1_IP0 比较器
定义
CMP_InitStructure.CMP1_SELNP         = CMP1_SELNP_DAC   ; // 比较器 1 信号负端选择
CMP_InitStructure.CMP1_IE            = ENABLE           ; // 比较器 1 中断使能
CMP_InitStructure.CMP1_POL           = 0                ; // 比较器 1 极性选择      0: 高电平有效 1: 低电平有效
只影响比较器中断
CMP_InitStructure.CMP1_RE            = DISABLE          ; // 比较器 1 DMA 请求使能
CMP_InitStructure.CMP1_IRQ_TRIG      = DISABLE          ; // 比较器 1 边沿触发使能
CMP_InitStructure.CMP1_CHN3P_WIN_EN  = DISABLE          ; // MCPWM 模块 CHN3_P 通道使能比较器 1 开窗
CMP_InitStructure.CMP1_CHN2P_WIN_EN  = DISABLE          ; // MCPWM 模块 CHN2_P 通道使能比较器 1 开窗
CMP_InitStructure.CMP1_CHN1P_WIN_EN  = DISABLE          ; // MCPWM 模块 CHN1_P 通道使能比较器 1 开窗
CMP_InitStructure.CMP1_CHN0P_WIN_EN  = DISABLE          ; // MCPWM 模块 CHN0_P 通道使能比较器 1 开窗
CMP_InitStructure.CMP1_W_PWM_POL     = DISABLE          ; // 比较器 1 开窗 PWM 信号极性选择 0:高电平有效; 1:低
电平有效
CMP_Init (&CMP_InitStructure);
```

Date	Version	Author	Description
2018年11月26日	1.0	Howlet Li	创建

修改日志:

在文件 [lks32mc08x_cmp.c](#) 第 56 行定义.

函数调用图:



3.5.3.4 CMP_StructInit()

```
void CMP_StructInit (
    CMP_InitTypeDef * CMP_InitStructure )
```

函数名称: void [CMP_StructInit\(CMP_InitTypeDef* CMP_InitStructure\)](#)

功能描述: ADC结构体初始化

参见

被调用函数: 无

参数

输入参数:	CMP.InitTypeDef
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
CMP_StructInit CMP_InitStructure;  
CMP_StructInit (&CMP_InitStructure); //初始化结构体
```

Date	Version	Author	Description
2018年11月26日	1.0	Howlet Li	创建

修改日志:

在文件 [lks32mc08x_cmp.c](#) 第 121 行定义.

3.6 lks32mc08x_cmp.h

浏览该文件的文档.

```
00001  
00019 #ifndef __LKS32MC08x_CMP_H  
00020 #define __LKS32MC08x_CMP_H  
00021 /* Includes -----*/  
00022 #include "lks32mc08x.lib.h"  
00023  
00025 #define CMP_HYS_20mV 0  
00026 #define CMP_HYS_0mV 1  
00028 // 比较器N端  
00029 #define CMP_SELN_IN 0  
00030 #define CMP_SELN_REF 1  
00031 #define CMP_SELN_DAC 2  
00032 #define CMP_SELN_HALL_MID 3  
00034 // 比较器O P端  
00035 #define CMP0_SELP_IP0 0  
00036 #define CMP0_SELP_OPA0_IP 1  
00037 #define CMP0_SELP_OPA0_OUT 2  
00038 #define CMP0_SELP_OPA1_OUT 3  
00039 #define CMP0_SELP_IP1 4  
00040 #define CMP0_SELP_IP2 5
```

```

00041 #define CMP0_SELFP_IP3      6
00042 #define CMP0_SELFP_IP4      7
00044 // 比较器1 P端
00045 #define CMP1_SELFP_IP0      0
00046 #define CMP1_SELFP_OPA3_IP  1
00047 #define CMP1_SELFP_OPA2_OUT 2
00048 #define CMP1_SELFP_OPA3_OUT 3
00049 #define CMP1_SELFP_IP1      4
00050 #define CMP1_SELFP_IP2      5
00051 #define CMP1_SELFP_IP3      6
00052 #define CMP1_SELFP_AVSS     7
00054 // 为保证库兼容性, 保留修改的宏定义
00055 #define SELN_CMP_IN         0
00056 #define SELN_REF            1
00057 #define SELN_DAC            2
00058 #define SELN_HALL_MID       3
00060 #define SELP_CMP_OPA0_IP    1
00061 #define SELP_CMP_OPA0_OUT   2
00062 #define SELP_CMP_OPA1_OUT   3
00063 #define SELP_CMP_OPA3_IP    1
00064 #define SELP_CMP_OPA2_OUT   2
00065 #define SELP_CMP_OPA3_OUT   3
00066 #define SELP_CMP_IP0        0
00067 #define SELP_CMP_IP1        4
00068 #define SELP_CMP_IP2        5
00069 #define SELP_CMP_IP3        6
00070 #define SELP_CMP_IP4        7
00075 typedef struct
00076 {
00077     __IO uint32_t IE;
00078     __IO uint32_t IF;
00079     __IO uint32_t TCLK;
00080     __IO uint32_t CFG;
00081     __IO uint32_t BLCWIN;
00082     __IO uint32_t DATA;
00083 } CMP_TypeDef;
00087 #ifndef CMP
00088 #define CMP ((CMP_TypeDef *)CMP_BASE)
00089 #endif
00093 #define CMP0 0
00097 #define CMP1 1
00098
00102 typedef struct
00103 {
00108     u32 FIL_CLK10_DIV16;
00109     FuncState CLK10_EN;
00114     u32 FIL_CLK10_DIV2;
00115     FuncState CMP_FT;
00116     u32 CMP_HYS;
00118     FuncState CMP0_EN;
00129     u32 CMP0_SELFP;
00137     u32 CMP0_SELN;
00138     FuncState CMP0_IE;
00139     u32 CMP0_W_PWM_POL;
00140     FuncState CMP0_IRQ_TRIG;
00141     FuncState CMP0_IN_EN;
00142     u32 CMP0_POL;
00143     FuncState CMP0_CHN3P_WIN_EN;
00144     FuncState CMP0_CHN2P_WIN_EN;
00145     FuncState CMP0_CHN1P_WIN_EN;
00146     FuncState CMP0_CHNOP_WIN_EN;
00148     FuncState CMP1_EN;
00159     u32 CMP1_SELFP;
00167     u32 CMP1_SELN;
00168     FuncState CMP1_IE;
00169     u32 CMP1_W_PWM_POL;
00170     FuncState CMP1_IRQ_TRIG;
00171     FuncState CMP1_IN_EN;
00172     u32 CMP1_POL;
00173     FuncState CMP1_CHN3P_WIN_EN;
00174     FuncState CMP1_CHN2P_WIN_EN;
00175     FuncState CMP1_CHN1P_WIN_EN;
00176     FuncState CMP1_CHNOP_WIN_EN;
00177 } CMP_InitTypeDef;
00178
00179 void CMP_Init(CMP_InitTypeDef *); // 比较器初始化
00180 void CMP_StructInit(CMP_InitTypeDef *); // 比较器配置结构体初始化
00181 void CMP_ClearIRQFlag(u8 CMPx); // 清CMP中断
00182 u8 CMP_GetIRQFlag(u8 CMPx); // 读CMP中断
00183
00184 #endif /*__LKS32MC05x_CMP_H */
00185 /***** (C) COPYRIGHT LINKO SEMICONDUCTOR *****/

```

3.7 include/lks32mc08x_dac.h 文件参考

文件名称: [lks32mc08x_dac.h](#)

文件标识: 无

内容摘要: DAC外设驱动程序头文件

其它说明: 无

```
#include "lks32mc08x_lib.h"
```

[lks32mc08x_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_RANGE_3V0](#) 0
DAC 3.0V量程
- #define [DAC_RANGE_1V2](#) 1
DAC 1.2V量程
- #define [DAC_RANGE_4V85](#) 2
DAC 4.85V量程

函数

- void [DAC_StructInit](#) ([DAC_InitTypeDef](#) *DAC_InitStruct)
函数名称: void [DAC_StructInit](#)([DAC_InitTypeDef](#)* [DAC_InitStruct](#))
- void [DAC_Init](#) ([DAC_InitTypeDef](#) *DAC_InitStruct)
函数名称: void [DAC_init](#)([DAC_InitTypeDef](#)* [DAC_InitStruct](#))
- void [DAC_OutputValue](#) (uint32_t DACValue)
函数名称: void [DAC_OutputValue](#)(uint32_t [DACValue](#))
- void [DAC_OutputVoltage](#) (uint16_t DACVoltage)
函数名称: void [DAC_OutputVoltage](#)(uint32_t [DACVoltage](#))
- void [DAC_Output](#) ([DAC_InitTypeDef](#) *DAC_InitStruct)

3.7.1 详细描述

文件名称: [lks32mc08x.dac.h](#)
文件标识: 无
内容摘要: DAC外设驱动程序头文件
其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2022年04月18日	1.0	Zhu Jie	创建

修改日志:

在文件 [lks32mc08x.dac.h](#) 中定义.

3.7.2 宏定义说明

3.7.2.1 DAC_RANGE_1V2

```
#define DAC_RANGE_1V2 1
```

DAC 1.2V量程

在文件 [lks32mc08x.dac.h](#) 第 33 行定义.

3.7.2.2 DAC_RANGE_3V0

```
#define DAC_RANGE_3V0 0
```

DAC 3.0V量程

在文件 [lks32mc08x.dac.h](#) 第 32 行定义.

3.7.2.3 DAC_RANGE_4V85

```
#define DAC_RANGE_4V85 2
```

DAC 4.85V量程

在文件 [lks32mc08x.dac.h](#) 第 34 行定义.

3.7.3 函数说明

3.7.3.1 DAC_Init()

```
void DAC_Init (
    DAC_InitTypeDef * DAC_InitStruct )
```

函数名称: void DAC.init(DAC_InitTypeDef* DAC_InitStruct)

功能描述: DAC初始化函数

参见
被调用函数: [SYS_AnalogModuleClockCmd\(\)](#)

参数

输入参数:	DAC_InitTypeDef
输出参数:	无

返回
返回值: 无

注解
其它说明: 无

警告
无

示例代码:

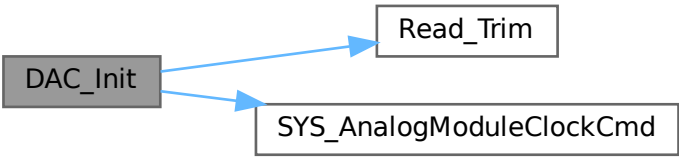
```
DAC_InitTypeDef DAC_InitStructure;
DAC_StructInit (&DAC_InitStructure);
DAC_InitStructure.DAC_GAIN = DAC_RANGE_1V2; //DAC量程为1.2V
DAC_InitStructure.DACOUT_EN = DISABLE ;    //DAC通过IO P0[0]输出
DAC_Init (&DAC_InitStructure);            // DAC初始化
```

Date	Version	Author	Description
2022年4月18日	1.0	Zhu Jie	创建

修改日志:

在文件 [lks32mc08x_dac.c](#) 第 64 行定义.

函数调用图:



3.7.3.2 DAC_Output()

```
void DAC_Output (
    DAC_InitTypeDef * DAC_InitStruct )
```

在文件 [lks32mc08x-dac.c](#) 第 165 行定义.

3.7.3.3 DAC_OutputValue()

```
void DAC_OutputValue (
    uint32_t DACValue )
```

函数名称: void [DAC.OutputValue\(uint32_t DACValue\)](#)

功能描述: DAC输出数字量数值设置

参见
被调用函数: 无

参数

输入参数:	DACValue:DAC输出电压对应数字量
输出参数:	无

返回
返回 值: 无

注解
其它说明: 无

警告

无

示例代码:

```
DAC.OutputValue(2048); //DAC输出 $2048 \times 1.2 / 4096 = 0.6V$ 
```

Date	Version	Author	Description
2022年4月18日	1.0	Zhu Jie	创建

修改日志:

在文件 [lks32mc08x-dac.c](#) 第 111 行定义.

3.7.3.4 DAC_OutputVoltage()

```
void DAC_OutputVoltage (
    uint16_t DACVoltage )
```

函数名称: void DAC.OutputVoltage(uint32_t DACVoltage)

功能描述: DAC输出模拟量数值设置

参见

被调用函数: 无

参数

输入参数:	DACVoltage: DAC输出电压对应数字量
输出参数:	无

返回

返回值: 无

注解

其它说明: 输入变量DACVoltage为Q12格式, 即输出1V对应DACVoltage = $1 \ll 12 = 4096$

警告

无

示例代码:

```
DAC.OutputVoltage(BIT12 * 0.6); //DAC输出0.6V
```

Date	Version	Author	Description
2022年4月18日	1.0	Zhu Jie	创建

修改日志:

在文件 [lks32mc08x-dac.c](#) 第 139 行定义.

3.7.3.5 DAC_StructInit()

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

函数名称: void [DAC_StructInit\(DAC_InitTypeDef* DAC_InitStruct\)](#)

功能描述: DAC结构体初始化

参见

被调用函数: 无

参数

输入参数:	ADC_InitTypeDef
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
DAC_StructInit DAC_InitStructure;
DAC_StructInit (&DAC_InitStructure); //初始化结构体
```

Date	Version	Author	Description
2022年4月18日	1.0	Zhu Jie	创建

修改日志:

在文件 [lks32mc08x_dac.c](#) 第 37 行定义.

3.8 lks32mc08x_dac.h

[浏览该文件的文档.](#)

```
00001
00014 #ifndef __LKS32MC08x_DAC_H
00015 #define __LKS32MC08x_DAC_H
00016
00017 /* Includes -----*/
00018 #include "lks32mc08x_lib.h"
00019 typedef struct
00020 {
00027     uint8_t DAC_GAIN;
00028     uint8_t DACOUT_EN;
00029 } DAC_InitTypeDef;
00030
00031 /* -----DAC操作相关定义 ----- */
00032 #define DAC_RANGE_3V0 0
00033 #define DAC_RANGE_1V2 1
00034 #define DAC_RANGE_4V85 2
00036 void DAC_StructInit(DAC_InitTypeDef *DAC_InitStruct); /* DAC结构体初始化 */
00037 void DAC_Init(DAC_InitTypeDef *DAC_InitStruct); /* DAC初始化 */
00038 void DAC_OutputValue(uint32_t DACValue); /* DAC输出数值设置--数字量 */
00039 void DAC_OutputVoltage(uint16_t DACVoltage); /* DAC输出模拟量数值设置--模拟量 */
00040 void DAC_Output(DAC_InitTypeDef *DAC_InitStruct); /* DAC输出至IO口使能配置 */
00041
00042 #endif
```

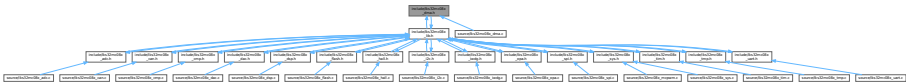
3.9 include/lks32mc08x_dma.h 文件参考

文件名称: [lks32mc08x_dma.h](#)
文件标识: 无
内容摘要: DMA外设驱动程序头文件
其它说明: 无

#include "lks32mc08x_lib.h"
lks32mc08x_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
- struct [DMA_RegTypeDef](#)
DMA通道0结构体基地址定义

宏定义

- #define [DMA_TCIE](#) BIT0
DMA中断使能定义，传输完成中断使能，高有效
- #define [DMA_TEIE](#) BIT1
DMA中断使能定义，错误中断使能，高有效
- #define [PERI2MEMORY](#) 0
DMA搬运方向定义，外设至内存
- #define [MEMORY2PERI](#) 1
DMA搬运方向定义，内存至外设
- #define [DMA_BYTE_TRANS](#) 0
DMA搬运数据位宽定义，访问位宽，0:byte
- #define [DMA_HALFWORD_TRANS](#) 1
DMA搬运数据位宽定义，访问位宽，1:half-word
- #define [DMA_WORD_TRANS](#) 2
DMA搬运数据位宽定义，访问位宽，2:word
- #define [DMA_CH0_ADC0_REQ_EN](#) BIT0
触发DMA搬运源定义，Channel0 ADC0 DMA请求使能
- #define [DMA_CH0_TIMER0_REQ_EN](#) BIT1
触发DMA搬运源定义，Channel0 TIMER0 DMA请求使能
- #define [DMA_CH0_TIMER1_REQ_EN](#) BIT2
触发DMA搬运源定义，Channel0 TIMER1 DMA请求使能
- #define [DMA_CH1_SPI_RX_REQ_EN](#) BIT0
触发DMA搬运源定义，Channel1 SPI_RX DMA请求使能
- #define [DMA_CH1_MCPWM_REQ_EN](#) BIT1
触发DMA搬运源定义，Channel1 MCPWM DMA请求使能
- #define [DMA_CH1_TIMER2_REQ_EN](#) BIT2
触发DMA搬运源定义，Channel1 TIMER2 DMA请求使能
- #define [DMA_CH2_UART0_REQ_EN](#) BIT0
触发DMA搬运源定义，Channel2 UART0 DMA请求使能
- #define [DMA_CH2_SPI_TX_REQ_EN](#) BIT1
触发DMA搬运源定义，Channel2 SPI_TX DMA请求使能
- #define [DMA_CH2_TIMER3_REQ_EN](#) BIT2
触发DMA搬运源定义，Channel2 TIMER3 DMA请求使能
- #define [DMA_CH3_UART1_REQ_EN](#) BIT0
触发DMA搬运源定义，Channel3 UART1 DMA请求使能
- #define [DMA_CH3_CAN_REQ_EN](#) BIT1
触发DMA搬运源定义，Channel3 SPI.CAN DMA请求使能
- #define [DMA_CH3_I2C_REQ_EN](#) BIT2
触发DMA搬运源定义，Channel3 I2X DMA请求使能
- #define [DMA_CH0_FIF](#) BIT0
DMA中断标志定义，通道0完成中断标志
- #define [DMA_CH1_FIF](#) BIT1

- DMA*中断标志定义, 通道 1 完成中断标志
- `#define DMA_CH2_FIF BIT2`
*DMA*中断标志定义, 通道 2 完成中断标志
- `#define DMA_CH3_FIF BIT3`
*DMA*中断标志定义, 通道 3 完成中断标志
- `#define DMA_CH0{EIF BIT4`
*DMA*中断标志定义, 通道 0 错误中断标志
- `#define DMA_CH1{EIF BIT5`
*DMA*中断标志定义, 通道 1 错误中断标志
- `#define DMA_CH2{EIF BIT6`
*DMA*中断标志定义, 通道 2 错误中断标志
- `#define DMA_CH3{EIF BIT7`
*DMA*中断标志定义, 通道 3 错误中断标志

函数

- `void DMA_Init (DMA_RegTypeDef *DMAx, DMA_InitTypeDef *DMAInitStruct)`
函数名称: `void DMA_Init(DMA_RegTypeDef *DMAx, DMA_InitTypeDef *DMAInitStruct)`
- `void DMA_StructInit (DMA_InitTypeDef *DMAInitStruct)`
函数名称: `void DMA_StructInit(DMA_InitTypeDef *DMAInitStruct)`
- `uint32_t DMA_GetIRQFlag (DMA_RegTypeDef *DMAx, u32 timer_if)`
函数名称: `uint32_t DMA_GetIRQFlag(DMA_RegTypeDef *DMAx , u32 timer_if)`
- `void DMA_ClearIRQFlag (DMA_RegTypeDef *DMAx, uint32_t tempFlag)`
函数名称: `void DMA_ClearIRQFlag(DMA_RegTypeDef *DMAx , uint32_t tempFlag)`
- `void DMA_CHx_EN (DMA_RegTypeDef *DMAx, u32 Channel_EN)`
函数名称: `void DMA_CHx_EN(DMA_RegTypeDef *DMAx,u32 Channel_EN)`

3.9.1 详细描述

文件名称: `lks32mc08x_dma.h`
 文件标识: 无
 内容摘要: *DMA*外设驱动程序头文件
 其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2020年08月1日	1.0	Howlet	创建

修改日志:

在文件 `lks32mc08x_dma.h` 中定义.

3.9.2 宏定义说明

3.9.2.1 DMA_BYTE_TRANS

```
#define DMA_BYTE_TRANS 0
```

DMA搬运数据位宽定义，访问位宽， 0:byte

在文件 [lks32mc08x_dma.h](#) 第 118 行定义.

3.9.2.2 DMA_CH0_ADC0_REQ_EN

```
#define DMA_CH0_ADC0_REQ_EN BIT0
```

触发DMA搬运源定义， Channel.0 ADC0 DMA请求使能

在文件 [lks32mc08x_dma.h](#) 第 122 行定义.

3.9.2.3 DMA_CH0_EIF

```
#define DMA_CH0_EIF BIT4
```

DMA中断标志定义，通道 0 错误中断标志

在文件 [lks32mc08x_dma.h](#) 第 139 行定义.

3.9.2.4 DMA_CH0_FIF

```
#define DMA_CH0_FIF BIT0
```

DMA中断标志定义，通道 0 完成中断标志

在文件 [lks32mc08x_dma.h](#) 第 135 行定义.

3.9.2.5 DMA_CH0_TIMER0_REQ_EN

```
#define DMA_CH0_TIMER0_REQ_EN BIT1
```

触发DMA搬运源定义， Channel.0 TIMER0 DMA请求使能

在文件 [lks32mc08x_dma.h](#) 第 123 行定义.

3.9.2.6 DMA_CH0_TIMER1_REQ_EN

```
#define DMA_CH0_TIMER1_REQ_EN BIT2
```

触发DMA搬运源定义， Channel.0 TIMER1 DMA请求使能

在文件 [lks32mc08x_dma.h](#) 第 124 行定义.

3.9.2.7 DMA_CH1_EIF

```
#define DMA_CH1_EIF BIT5
```

DMA中断标志定义，通道 1 错误中断标志

在文件 [lks32mc08x_dma.h](#) 第 140 行定义.

3.9.2.8 DMA_CH1_FIF

```
#define DMA_CH1_FIF BIT1
```

DMA中断标志定义，通道 1 完成中断标志

在文件 [lks32mc08x_dma.h](#) 第 136 行定义.

3.9.2.9 DMA_CH1_MCPWM_REQ_EN

```
#define DMA_CH1_MCPWM_REQ_EN BIT1
```

触发DMA搬运源定义， Channel_1 MCPWM DMA请求使能

在文件 [lks32mc08x_dma.h](#) 第 126 行定义.

3.9.2.10 DMA_CH1_SPI_RX_REQ_EN

```
#define DMA_CH1_SPI_RX_REQ_EN BIT0
```

触发DMA搬运源定义， Channel_1 SPI_RX DMA请求使能

在文件 [lks32mc08x_dma.h](#) 第 125 行定义.

3.9.2.11 DMA_CH1_TIMER2_REQ_EN

```
#define DMA_CH1_TIMER2_REQ_EN BIT2
```

触发DMA搬运源定义， Channel_1 TIMER2 DMA请求使能

在文件 [lks32mc08x_dma.h](#) 第 127 行定义.

3.9.2.12 DMA_CH2_EIF

```
#define DMA_CH2_EIF BIT6
```

DMA中断标志定义，通道 2 错误中断标志

在文件 [lks32mc08x_dma.h](#) 第 141 行定义.

3.9.2.13 DMA_CH2_FIF

```
#define DMA_CH2_FIF BIT2
```

DMA中断标志定义，通道 2 完成中断标志

在文件 [lks32mc08x_dma.h](#) 第 137 行定义.

3.9.2.14 DMA_CH2_SPI_TX_REQ_EN

```
#define DMA_CH2_SPI_TX_REQ_EN BIT1
```

触发DMA搬运源定义， Channel.2 SPI.TX DMA请求使能

在文件 [lks32mc08x_dma.h](#) 第 129 行定义.

3.9.2.15 DMA_CH2_TIMER3_REQ_EN

```
#define DMA_CH2_TIMER3_REQ_EN BIT2
```

触发DMA搬运源定义， Channel.2 TIMER3 DMA请求使能

在文件 [lks32mc08x_dma.h](#) 第 130 行定义.

3.9.2.16 DMA_CH2_UART0_REQ_EN

```
#define DMA_CH2_UART0_REQ_EN BIT0
```

触发DMA搬运源定义， Channel.2 UART0 DMA请求使能

在文件 [lks32mc08x_dma.h](#) 第 128 行定义.

3.9.2.17 DMA_CH3_CAN_REQ_EN

```
#define DMA_CH3_CAN_REQ_EN BIT1
```

触发DMA搬运源定义， Channel.3 SPI.CAN DMA请求使能

在文件 [lks32mc08x_dma.h](#) 第 132 行定义.

3.9.2.18 DMA_CH3{EIF

```
#define DMA_CH3{EIF BIT7
```

DMA中断标志定义，通道 3 错误中断标志

在文件 [lks32mc08x_dma.h](#) 第 142 行定义.

3.9.2.19 DMA_CH3_FIF

```
#define DMA_CH3_FIF BIT3
```

DMA中断标志定义，通道 3 完成中断标志

在文件 [lks32mc08x_dma.h](#) 第 138 行定义.

3.9.2.20 DMA_CH3_I2C_REQ_EN

```
#define DMA_CH3_I2C_REQ_EN BIT2
```

触发DMA搬运源定义， Channel.3 I2X DMA请求使能

在文件 [lks32mc08x_dma.h](#) 第 133 行定义.

3.9.2.21 DMA_CH3_UART1_REQ_EN

```
#define DMA_CH3_UART1_REQ_EN BIT0
```

触发DMA搬运源定义， Channel.3 UART1 DMA请求使能

在文件 [lks32mc08x_dma.h](#) 第 131 行定义.

3.9.2.22 DMA_HALFWORD_TRANS

```
#define DMA_HALFWORD_TRANS 1
```

DMA搬运数据位宽定义，访问位宽， 1:half-word

在文件 [lks32mc08x_dma.h](#) 第 119 行定义.

3.9.2.23 DMA_TCIE

```
#define DMA_TCIE BIT0
```

DMA中断使能定义，传输完成中断使能，高有效

在文件 [lks32mc08x_dma.h](#) 第 112 行定义.

3.9.2.24 DMA_TEIE

```
#define DMA_TEIE BIT1
```

DMA中断使能定义，错误中断使能，高有效

在文件 [lks32mc08x_dma.h](#) 第 113 行定义.

3.9.2.25 DMA_WORD_TRANS

```
#define DMA_WORD_TRANS 2
```

DMA搬运数据位宽定义，访问位宽， 2:word

在文件 [lks32mc08x_dma.h](#) 第 120 行定义.

3.9.2.26 MEMORY2PERI

```
#define MEMORY2PERI 1
```

DMA搬运方向定义，内存至外设

在文件 [lks32mc08x_dma.h](#) 第 116 行定义.

3.9.2.27 PERI2MEMORY

```
#define PERI2MEMORY 0
```

DMA搬运方向定义，外设至内存

在文件 [lks32mc08x_dma.h](#) 第 115 行定义.

3.9.3 函数说明

3.9.3.1 DMA_CHx_EN()

```
void DMA_CHx_EN (
    DMA_RegTypeDef *DMAx,
    u32 Channel_EN )
```

函数名称: void [DMA_CHx_EN\(DMA_RegTypeDef *DMAx,u32 Channel_EN\)](#)

功能描述: 使能DMA通道

参见

被引用内容: DMAx可选: DMA_CH0 , DMA_CH1 , DMA_CH2 , DMA_CH3

参数

输入参数:	DMAx: DMA通道选择 Channel_EN: ENABLE: 使能DMA通道, DISABLE: 关闭DMA通道使能
输出参数:	无

返回

返回 值: 无

注解

其它说明: 无

警告

无

示例代码:

```
DMA_CHx_EN (DMA_CH0, ENABLE); //使能DMA通道0
```

Date	Version	Author	Description
2022年4月13日	1.1	HuangMG	修改

修改日志:

在文件 [lks32mc08x_dma.c](#) 第 140 行定义.

3.9.3.2 DMA_ClearIRQFlag()

```
void DMA_ClearIRQFlag (
    DMA_RegTypeDef * DMAx,
    uint32_t tempFlag )
```

函数名称: void [DMA_ClearIRQFlag\(DMA_RegTypeDef *DMAx , uint32_t tempFlag\)](#)

功能描述: 清除DMA中断标志

参见

被引用内容: DMAx可选: DMA_CH0 , DMA_CH1 , DMA_CH2 , DMA_CH3

参数

输入参数:	DMAx: DMA通道选择	
	timer_if参数可选:	
	宏定义	说明
	DMA_CH0_FIF	通道 0 完成中断标志
	DMA_CH1_FIF	通道 1 完成中断标志
	DMA_CH2_FIF	通道 2 完成中断标志
	DMA_CH3_FIF	通道 3 完成中断标志
	DMA_CH0{EIF	通道 0 错误中断标志
	DMA_CH1{EIF	通道 1 错误中断标志
	DMA_CH2{EIF	通道 2 错误中断标志
	DMA_CH3{EIF	通道 3 错误中断标志
输出参数:	无	

返回

返回 值: 无

注解

其它说明: 无

警告

无

示例代码:

```
if (DMA_GetIRQFlag (DMA_CH0, DMA_CH0_FIF) ) //获取DMA通道0完成中断标志
{
    DMA_ClearIRQFlag (DMA_CH0, DMA_CH0_FIF) //清除DMA通道0完成中断标志
}
```

Date	Version	Author	Description
2022年4月13日	1.1	HuangMG	修改

修改日志:

在文件 [lks32mc08x-dma.c](#) 第 241 行定义.

3.9.3.3 DMA_GetIRQFlag()

```
uint32_t DMA_GetIRQFlag (
    DMA_RegTypeDef * DMAx,
    u32 timer_if )
```

函数名称: `uint32_t DMA_GetIRQFlag(DMA_RegTypeDef *DMAx , u32 timer_if)`

功能描述: 获取DMA中断标志

参见

被引用内容: DMAx可选: DMA_CH0 , DMA_CH1 , DMA_CH2 , DMA_CH3

参数

输入参数:	DMAx: DMA通道选择	
	timer_if参数可选:	
	宏定义	说明
	DMA_CH0_FIF	通道 0 完成中断标志
	DMA_CH1_FIF	通道 1 完成中断标志
	DMA_CH2_FIF	通道 2 完成中断标志
	DMA_CH3_FIF	通道 3 完成中断标志
	DMA_CH0 EIF	通道 0 错误中断标志
	DMA_CH1 EIF	通道 1 错误中断标志
	DMA_CH2 EIF	通道 2 错误中断标志
	DMA_CH3 EIF	通道 3 错误中断标志

参见

参数

输出参数:	无

返回

返回 值: 无

注解

其它说明: 无

警告

只有对应中断使能后, 改为才能读取, 如果对应中断未使能, 读取结果一直为0

示例代码:

```
if (DMA_GetIRQFlag (DMA_CH0, DMA_CH0_FIF) ) //获取DMA通道0完成中断标志
{
}
```

Date	Version	Author	Description
2022年4月13日	1.1	HuangMG	修改

修改日志:

在文件 [lks32mc08x_dma.c](#) 第 185 行定义.

3.9.3.4 DMA_Init()

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

函数名称: void [DMA_Init\(DMA_RegTypeDef *DMAx, DMA_InitTypeDef *DMAInitStruct\)](#)

功能描述: DMA初始化函数

参见
被引用内容: 无

参数

输入参数:	ADCx: ADC0 , ADC_InitTypeDef
输出参数:	无

返回
返回 值: 无

注解
其它说明: 无

警告
无

示例代码:

```
u8 memaddr[25] = {0x11, 0x12, 0x13}; // 发送数据缓冲区

DMA_InitTypeDef DMA_InitStruct;
DMA_StructInit (&DMA_InitStruct);
DMA_InitStruct.DMA_IRQ_EN = DMA_TCIE; // DMA 传输完成中断使能
DMA_InitStruct.DMA_DIR = MEMORY2PERI; // 内存至外设
DMA_InitStruct.DMA_CIRC = DISABLE; // 关闭循环传输模式
DMA_InitStruct.DMA_PINC = ENABLE; // 外设地址每轮内是否递增, 高有效
DMA_InitStruct.DMA_MINC = ENABLE; // 内存地址第二轮是否在第一轮地址的基础上递增 (轮内地址一定递增), 高有效
DMA_InitStruct.DMA_PBTW = DMA_BYTE_TRANS; // 外设访问位宽, 0:byte, 1:half-word, 2:word
```

```
DMA_InitStructure.DMA_MBTW = DMA_BYTE_TRANS; // 内存访问位宽, 0:byte, 1:half-word, 2:word
DMA_InitStructure.DMA_REQ_EN = DMA_CH3_CAN_REQ_EN; // CAN DMA请求使能, 高有效
DMA_InitStructure.DMA_TIMES = 5; // DMA 通道 x 每轮数据搬运次数 1~511
DMA_InitStructure.DMA_ROUND = 5; // DMA 通道 x 采样轮数 1~255
DMA_InitStructure.DMA_CPAR = (u32)&CAN_TXRX0; // DMA 通道 x 外设地址
DMA_InitStructure.DMA_CMAR = (u32)memaddr; // DMA 通道 x 内存地址
DMA_Init(DMA_CH3, &DMA_InitStructure);
DMA_CHx_EN(DMA_CH3, ENABLE); //使能DMA_CH3通道
```

Date	Version	Author	Description
2015年11月26日	1.0	Howlet Li	创建

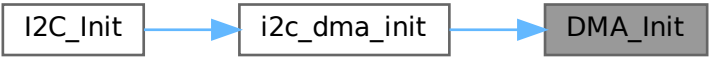
修改日志:

Date	Version	Author	Description
2022年4月13日	1.1	HuangMG	修改

修改日志:去除初始化使能DMA通道，改为DMA_CHx_EN（）函数使能DMA通道

在文件 lks32mc08x_dma.c 第 101 行定义.

这是这个函数的调用关系图:



3.9.3.5 DMA_StructInit()

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

函数名称: void DMA_StructInit(DMA_InitTypeDef *DMAInitStruct)

功能描述: DMA结构体初始化

参见

被引用内容: 无

参数

输入参数:	DMA.InitTypeDef
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
DMA_InitTypeDef DMA_InitStructure;  
DMA_StructInit (&DMA_InitStructure); //初始化结构体
```

Date	Version	Author	Description
2015年11月26日	1.0	Howlet	修改

修改日志:

在文件 [lks32mc08x_dma.c](#) 第 55 行定义.

3.10 lks32mc08x_dma.h

浏览该文件的文档.

```
00001  
00014 #ifndef __LKS32MC08x_DMA_H  
00015 #define __LKS32MC08x_DMA_H  
00016  
00017 #include "lks32mc08x_lib.h"  
00018  
00022 typedef struct  
00023 {  
00029     uint8_t DMA_IRQ_EN;  
00035     uint8_t DMA_DIR;  
00036     uint8_t DMA_CIRC;  
00037     uint8_t DMA_PINC;  
00038     uint8_t DMA_MINC;  
00045     uint8_t DMA_PBTW; /* 外设访问位宽, 0:byte, 1:half-word, 2:word */  
00052     uint8_t DMA_MBTW; /* 内存访问位宽, 0:byte, 1:half-word, 2:word */  
00068     uint8_t DMA_REQ_EN;  
00069     uint16_t DMA_TIMES;  
00070     uint8_t DMA_ROUND;  
00071     uint32_t DMA_CPAR;  
00072     uint32_t DMA_CMAR;
```

```

00074 } DMA_InitTypeDef;
00075
00079 #ifndef DMA_CH0
00080 #define DMA_CH0 ((DMA_RegTypeDef *)DMA_BASE)
00081 #endif
00085 #ifndef DMA_CH1
00086 #define DMA_CH1 ((DMA_RegTypeDef *) (DMA_BASE + 0x10))
00087 #endif
00091 #ifndef DMA_CH2
00092 #define DMA_CH2 ((DMA_RegTypeDef *) (DMA_BASE + 0x20))
00093 #endif
00097 #ifndef DMA_CH3
00098 #define DMA_CH3 ((DMA_RegTypeDef *) (DMA_BASE + 0x30))
00099 #endif
00100
00104 typedef struct
00105 {
00106     __IO uint32_t DMA_CCR;
00107     __IO uint32_t DMA_CTMS;
00108     __IO uint32_t DMA_CPAR;
00109     __IO uint32_t DMA_CMAR;
00110 } DMA_RegTypeDef;
00111
00112 #define DMA_TCIE BIT0
00113 #define DMA_TEIE BIT1
00115 #define PERI2MEMORY 0
00116 #define MEMORY2PERI 1
00118 #define DMA_BYTE_TRANS 0
00119 #define DMA_HALFWORD_TRANS 1
00120 #define DMA_WORD_TRANS 2
00122 #define DMA_CH0_ADC0_REQ_EN BIT0
00123 #define DMA_CH0_TIMER0_REQ_EN BIT1
00124 #define DMA_CH0_TIMER1_REQ_EN BIT2
00125 #define DMA_CH1_SPI_RX_REQ_EN BIT0
00126 #define DMA_CH1_MCPWM_REQ_EN BIT1
00127 #define DMA_CH1_TIMER2_REQ_EN BIT2
00128 #define DMA_CH2_UART0_REQ_EN BIT0
00129 #define DMA_CH2_SPI_TX_REQ_EN BIT1
00130 #define DMA_CH2_TIMER3_REQ_EN BIT2
00131 #define DMA_CH3_UART1_REQ_EN BIT0
00132 #define DMA_CH3_CAN_REQ_EN BIT1
00133 #define DMA_CH3_I2C_REQ_EN BIT2
00135 #define DMA_CH0_FIF BIT0
00136 #define DMA_CH1_FIF BIT1
00137 #define DMA_CH2_FIF BIT2
00138 #define DMA_CH3_FIF BIT3
00139 #define DMA_CH0_EIF BIT4
00140 #define DMA_CH1_EIF BIT5
00141 #define DMA_CH2_EIF BIT6
00142 #define DMA_CH3_EIF BIT7
00144 void DMA_Init(DMA_RegTypeDef *DMAx, DMA_InitTypeDef *DMAInitStruct);
00145 void DMA_StructInit(DMA_InitTypeDef *DMAInitStruct);
00146 uint32_t DMA_GetIRQFlag(DMA_RegTypeDef *DMAx, uint32_t timer_if);
00147 void DMA_ClearIRQFlag(DMA_RegTypeDef *DMAx, uint32_t tempFlag);
00148 void DMA_CHx_EN(DMA_RegTypeDef *DMAx, uint32_t Channel_EN);
00149 #endif

```

3.11 include/lks32mc08x_dsp.h 文件参考

文件名称: lks32mc08x_dsp.h

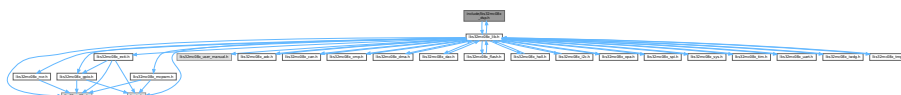
文件标识: 无

内容摘要: DSP驱动头文件

其它说明: 无

```
#include "lks32mc08x_lib.h"
```

lks32mc08x_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：



结构体

- struct [stru_DiviComponents](#)
除法运算结构体句柄
- struct [stru_CordicComponents](#)
三角函数运算结构体句柄

宏定义

- #define [DSP_CALC_SIN](#)(sin, angle)
- #define [DSP_CALC_COS](#)(cos, angle)
- #define [DSP_CALC_RMS](#)(rms, a, b)
- #define [DSP_CALC_ARCTAN](#)(arctan, x, y)
- #define [DSP_CALC_DIV](#)(c, a, b)
- #define [DSP_CALC_MOD](#)(mod, x, y)
- #define [DSP_CALC_SQRT](#)(sqrt, val)

枚举

- enum [Arc_or_SinCFuction](#) { [ARCTAN](#) = 0 , [SINCOS](#) = 1 }

函数

- void [DSP_Cmd](#) (FuncState state)
函数名称: *void DSP_Cmd(FuncState state)*
- void [DSP_CalcDivision](#) ([stru_DiviComponents](#) *stru_Divi)
函数名称: *void DSP_CalcDivision(stru_DiviComponents * stru_Divi)*
- void [DSP_CalcCordic](#) ([stru_CordicComponents](#) *Cordic, [Arc_or_SinCFuction](#) choise)
函数名称: *void DSP_CalcCordic(stru_CordicComponents *Cordic, Arc_or_SinCFuction choise)*
- uint32_t [DSP_GetSqrt](#) (uint32_t Data)
函数名称: *uint32_t DSP_GetSqrt(uint32_t Data)*
- s16 [DSP_CalcSprt](#) (s16 X, s16 Y)
函数名称: *s16 DSP_CalcSprt(s16 X,s16 Y)*
- void [DSP_Init](#) (void)
- s16 [DSP_arctan](#) (s16 x, s16 y)
- s32 [DSP_div](#) (s32 a, s32 b)
- s16 [DSP_sin](#) (u16 angle)
- s16 [DSP_cos](#) (u16 angle)
- u16 [DSP_rms](#) (s16 a, s16 b)
- s16 [DSP_mod](#) (s32 x, s16 y)
- u16 [DSP_sqrt](#) (u32 val)

3.11.1 详细描述

文件名称: lks32mc08x.dsp.h
文件标识: 无
内容摘要: DSP驱动头文件
其它说明: 无

版权所有
(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2018年07月25日	1.0	Howlet	创建

修改日志:

在文件 lks32mc08x.dsp.h 中定义.

3.11.2 宏定义说明

3.11.2.1 DSP_CALC_ARCTAN

```
#define DSP_CALC_ARCTAN(  
    arctan,  
    x,  
    y )
```

值:

```
{  
    int32_t flg = __get_PRIMASK(); \br/>    __disable_irq(); \br/>    DSP_SC = BIT1; \br/>    DSP_Y = y; \br/>    arctan = DSP_ARCTAN; \br/>    __set_PRIMASK(flgs); \br/>}
```

在文件 lks32mc08x.dsp.h 第 95 行定义.

3.11.2.2 DSP_CALC_COS

```
#define DSP_CALC_COS(  
    cos,  
    angle )
```

值:

```
{  
    int32_t flg = __get_PRIMASK(); \br/>    __disable_irq(); \br/>    DSP_SC = BIT2 | BIT1; \br/>    DSP_THETA = (u16)angle; \br/>    cos = DSP_COS; \br/>    __set_PRIMASK(flgs); \br/>}
```

在文件 lks32mc08x.dsp.h 第 76 行定义.

3.11.2.3 DSP_CALC_DIV

```
#define DSP_CALC_DIV(
```

```
    c,  
    a,  
    b )
```

值:

```
{  
    int32_t flg = __get_PRIMASK(); \\  
    __disable_irq(); \\  
    DSP_DID = (a); \\  
    DSP_DIS = (b); \\  
    c      = DSP_QUO; \\  
    __set_PRIMASK(flg); \\  
}
```

在文件 [lks32mc08x_dsp.h](#) 第 104 行定义.

3.11.2.4 DSP_CALC_MOD

```
#define DSP_CALC_MOD(
```

```
    mod,  
    x,  
    y )
```

值:

```
{  
    int32_t flg = __get_PRIMASK(); \\  
    __disable_irq(); \\  
    DSP_SC = BIT1; \\  
    DSP_X = x; \\  
    DSP_Y = y; \\  
    mod   = DSP_MOD; \\  
    __set_PRIMASK(flg); \\  
}
```

在文件 [lks32mc08x_dsp.h](#) 第 113 行定义.

3.11.2.5 DSP_CALC_RMS

```
#define DSP_CALC_RMS(
```

```
    rms,  
    a,  
    b )
```

值:

```
{  
    int32_t flg = __get_PRIMASK(); \\  
    __disable_irq(); \\  
    DSP_DID = a; \\  
    DSP_DIS = b; \\  
    rms     = DSP_REM; \\  
    __set_PRIMASK(flg); \\  
}
```

在文件 [lks32mc08x_dsp.h](#) 第 85 行定义.

3.11.2.6 DSP_CALC_SIN

```
#define DSP_CALC_SIN(  
    sin,  
    angle )  
  
值:  
{  
    int32_t flg = __get_PRIMASK(); \\  
    __disable_irq(); \\  
    DSP_SC      = BIT2 | BIT1; \\  
    DSP_THETA   = (u16)angle; \\  
    sin         = DSP_SIN; \\  
    __set_PRIMASK(flg); \\  
}
```

在文件 [lks32mc08x_dsp.h](#) 第 67 行定义.

3.11.2.7 DSP_CALC_SQRT

```
#define DSP_CALC_SQRT(  
    sqrt,  
    val )  
  
值:  
{  
    int32_t flg = __get_PRIMASK(); \\  
    __disable_irq(); \\  
    DSP_RAD = val; \\  
    sqrt    = DSP_SQRT; \\  
    __set_PRIMASK(flg); \\  
}
```

在文件 [lks32mc08x_dsp.h](#) 第 123 行定义.

3.11.3 枚举类型说明

3.11.3.1 Arc_or_SinCFuction

```
enum Arc_or_SinCFuction
```

枚举值

ARCTAN	0计算arctan
SINCOS	1计算sin cos

在文件 [lks32mc08x_dsp.h](#) 第 45 行定义.

3.11.4 函数说明

3.11.4.1 DSP_arctan()

```
s16 DSP_arctan (  
    s16 x,  
    s16 y )
```

在文件 [lks32mc08x_dsp.c](#) 第 242 行定义.

3.11.4.2 DSP_CalcCordic()

```
void DSP_CalcCordic (
    stru_CordicComponents * Cordic,
    Arc_or_SinCFuction choose )
```

函数名称: void DSP_CalcCordic(stru_CordicComponents *Cordic, Arc_or_SinCFuction choose)

功能描述: DSP三角运算

参见

被引用内容: stru_CordicComponents , Arc_or_SinCFuction , SYS_SoftResetModule()

参数

输入参数:	Cordic: 角度或坐标X和Y, choose: 计算模式选择 0 Arctan;1 sin/cos
输出参数:	无

返回

返回 值: 无

注解

- 其它说明:
- 1、开方8个总线周期完成
 - 2、Theta(角度)为16位有符号定点数，表示范围 $-(32768) \sim 32767$ 对应 $-(\pi) \sim \pi$
 - 3、SinValue值为16位有符号定点数，其中 1bit 符号位，15bit小数位；表示范围 $-(1) \sim 1$
 - 4、CosValue值为16位有符号定点数，其中 1bit 符号位，15bit小数位；表示范围 $-(1) \sim 1$
 - 5、ArctanX值为16位有符号定点数，Q15 格式，其中 1bit 符号位，15bit整数位，表示范围 $-(32768) \sim 32767$
 - 6、ArctanY值为16位有符号定点数，Q15 格式，其中 1bit 符号位，15bit整数位，表示范围 $-(32768) \sim 32767$
 - 7、ArctanValue值为16位有符号定点数，表示范围 $-(32768) \sim 32767$ 对应 $-(\pi) \sim \pi$

警告

无

示例代码:

```
stru_CordicComponents CordicData; // 定义三角函数计算结构体变量
示例1:
CordicData.ArctanX = 100; //坐标x赋值 取值范围  $-(32768) \sim 32767$ 
CordicData.ArctanY = 10;  //坐标y赋值 取值范围  $-(32768) \sim 32767$ 
    DSP_CalcDivision(&CordicData,ARCTAN);
arctan储存位置: CordicData.ArctanValue (输出范围  $-(32768) \sim 32767$ 对应  $-(\pi) \sim \pi$ )
示例2:
CordicData.Theta = 0;      //角度赋值 取值范围  $-(32768) \sim 32767$  对应  $-(\pi) \sim \pi$ 
    DSP_CalcDivision(&CordicData,SINCOS);
sin结果储存位置: CordicData.SinValue (输出范围  $-(32768) \sim 32767$ 对应  $-(1) \sim 1$ )
cos结果储存位置: CordicData.CosValue (输出范围  $-(32768) \sim 32767$ 对应  $-(1) \sim 1$ )
```

Date	Version	Author	Description
2016年5月25日	1.0	cfwu	创建

修改日志:

Date	Version	Author	Description
2022年4月18日	2.0	Zhu Jie	修改

修改日志:使用结构体变量，增加防止DSP计算被打断逻辑语句

在文件 [lks32mc08x_dsp.c](#) 第 136 行定义.

3.11.4.3 DSP_CalcDivision()

```
void DSP_CalcDivision (
    stru\_DiviComponents * stru_Divi )
```

函数名称: void [DSP_CalcDivision\(stru_DiviComponents * stru_Divi\)](#)

功能描述: DSP使能和关闭

参见

被引用内容: [stru_DiviComponents](#) , [SYS_SoftResetModule\(\)](#)

参数

输入参数:	.nDID,被除数 .nDIS除数
输出参数:	无

返回

返回值: 无

注解

- 其它说明:
- 1、除法10个总线周期完成
 - 2、被除数Dividend为32位大小，即范围是 $-(2^{31} - 1) \sim (2^{31} - 1)$
 - 3、除数Divisor为16位大小，即范围是 $-(2^{15} - 1) \sim (2^{15} - 1)$
 - 4、被除数不支持赋值为 -2^{31} ，除数不支持赋值为 -2^{15}
 - 5、商Quotient为32位大小
 - 6、余数Remainder为16位大小

警告

无

示例代码:

```

stru_DiviComponents DiviData; //定义除法函数计算结构体变量
DiviData.Dividend = 100; //被除数赋值
DiviData.Divisor = 10; //除数赋值
DSP_CalcDivision(&DiviData);
除法的商储存位置: DiviData.Quotient
除法的余数储存位置: DiviData.Remainder

```

Date	Version	Author	Description
2016年5月25日	1.0	cfwu	创建

修改日志:

Date	Version	Author	Description
2022年4月18日	2.0	Zhu Jie	修改

修改日志:使用结构体变量, 增加防止DSP计算被打断逻辑语句

在文件 [lks32mc08x_dsp.c](#) 第 82 行定义.

3.11.4.4 DSP_CalcSqrt()

```

s16 DSP_CalcSqrt (
    s16 X,
    s16 Y )

```

函数名称: **s16 DSP_CalcSqrt(s16 X,s16 Y)**

功能描述: DSP两个数平方后开方运算

参见

被引用内容: 无

参数

输入参数:	X和Y
输出参数:	无

返回

返回 值: $\text{sqrt}(X^2+Y^2)$ 结果值

注解

其它说明: 1、开方8个总线周期完成 2、X值为16位有符号定点数, Q15 格式, 其中 1bit 符号位, 15bit整数位, 表示范围 $-(32768)\sim 32767$
3、Y值为16位有符号定点数, Q15 格式, 其中 1bit 符号位, 15bit整数位, 表示范围 $-(32768)\sim 32767$
4、输出结果 $\text{sqrt}(X^2+Y^2)$: 为16位有符号定点数, 其中1bit符号位, 15bit整数位, 表示范围 $-(32768)\sim 32767$

警告

无

示例代码:

```
s16 DSP_Value = 0;  
DSP_Value = DSP_CalcSprt(50,100); //sqrt(50^2+100^2) 50的平方加100的平方后开根号
```

Date	Version	Author	Description
2016年5月25日	1.0	cfwu	创建

修改日志:

Date	Version	Author	Description
2022年4月18日	2.0	Zhu Jie	修改

修改日志:使用结构体变量, 增加防止DSP计算被打断逻辑语句

在文件 [lks32mc08x_dsp.c](#) 第 186 行定义.

3.11.4.5 DSP_Cmd()

```
void DSP_Cmd (  
    FuncState state )
```

函数名称: void [DSP_Cmd\(FuncState state\)](#)

功能描述: DSP使能和关闭

参见

被引用内容: [SYS.ModuleClockCmd\(\)](#)

参数

输入参数:	state: ENABLE: 使能 , DISABLE:失能
输出参数:	无

返回

返回 值: 无

注解

其它说明: 无

警告

无

示例代码:

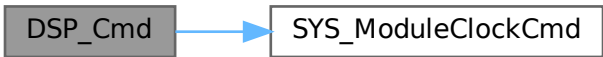
```
DSP_Cmd (ENABLE) ; // 使能DSP时钟
```

Date	Version	Author	Description
2022年4月18日	1.0	Zhu Jie	创建

修改日志:

在文件 lks32mc08x_dsp.c 第 41 行定义.

函数调用图:



3.11.4.6 DSP_cos()

```
s16 DSP_cos (
    u16 angle )
```

在文件 lks32mc08x_dsp.c 第 276 行定义.

3.11.4.7 DSP_div()

```
s32 DSP_div (
    s32 a,
    s32 b )
```

在文件 [lks32mc08x_dsp.c](#) 第 254 行定义.

3.11.4.8 DSP_GetSqrt()

```
uint32_t DSP_GetSqrt (
    uint32_t Data )
```

函数名称: `uint32_t DSP_GetSqrt(uint32_t Data)`

功能描述: DSP开平方函数

参见
被引用内容: 无

参数

输入参数:	被开方数
输出参数:	无

返回
返回 值: 开方根

注解
其它说明: 1、开方8个总线周期完成
2、被开方数Data为32位无符号数，即范围是(0 ~ (2^31))
3、开方根sqrt(Data)为16位无符号数，即范围是(0 ~ (2^15))

警告
无

示例代码:

```
u16 GetSqrt_Value = 0;
GetSqrt_Value = DSP_GetSqrt(100); //sqrt(100) 100开根号结果为10
```

Date	Version	Author	Description
2016年5月25日	1.0	cfwu	创建

修改日志:

Date	Version	Author	Description
2022年4月18日	2.0	Zhu Jie	修改

修改日志:使用结构体变量，增加防止DSP计算被打断逻辑语句

在文件 [lks32mc08x_dsp.c](#) 第 227 行定义.

3.11.4.9 DSP_Init()

```
void DSP_Init (
    void )
```

在文件 [lks32mc08x_dsp.c](#) 第 238 行定义.

函数调用图:



3.11.4.10 DSP_mod()

```
s16 DSP_mod (
    s32 x,
    s16 y )
```

在文件 [lks32mc08x_dsp.c](#) 第 298 行定义.

3.11.4.11 DSP_rms()

```
u16 DSP_rms (
    s16 a,
    s16 b )
```

在文件 [lks32mc08x_dsp.c](#) 第 287 行定义.

3.11.4.12 DSP_sin()

```
s16 DSP_sin (
    u16 angle )
```

在文件 [lks32mc08x_dsp.c](#) 第 265 行定义.

3.11.4.13 DSP_sqrt()

```
u16 DSP_sqrt (
    u32 val )
```

在文件 [lks32mc08x_dsp.c](#) 第 310 行定义.

3.12 lks32mc08x_dsp.h

[浏览该文件的文档.](#)

```
00001
00015 #ifndef __LKS32MC08x_DSP_H
00016 #define __LKS32MC08x_DSP_H
00017
00018 /* Includes -----*/
00019 #include "lks32mc08x.lib.h"
00020
00024 typedef struct
00025 {
00026     s32 Dividend;
00027     s16 Divisor;
00028     s32 Quotient;
00029     s16 Remainder;
00030 } stru_DiviComponents;
00031
00035 typedef struct
00036 {
00037     s16 Theta;
00038     s16 SinValue;
00039     s16 CosValue;
00040     s16 ArctanX;
00041     s16 ArctanY;
00042     s16 ArctanValue;
00043 } stru_CordicComponents;
00044
00045 typedef enum
00046 {
00047     ARCTAN = 0,
00048     SINCOS = 1
00049 } Arc_or_SinCFuction;
00050
00051 void DSP_Cmd(FuncState state); /* DSP使能和关闭 */
00052 void DSP_CalcDivision(stru_DiviComponents *stru_Divi); /* DSP除法运算 */
00053 void DSP_CalcCordic(stru_CordicComponents *Cordic, Arc_or_SinCFuction choose); /* DSP三角函数运算 */
00054 uint32_t DSP_GetSqrt(uint32_t Data); /* DSP取得开方根 */
00055 s16 DSP_CalcSprt(s16 X, s16 Y); /* DSP计算开方根 */
00056
00057 void DSP_Init(void);
00058 s16 DSP_arctan(s16 x, s16 y);
00059 s32 DSP_div(s32 a, s32 b);
00060 s16 DSP_sin(u16 angle);
00061 s16 DSP_cos(u16 angle);
00062 u16 DSP_rms(s16 a, s16 b);
00063 s16 DSP_mod(s32 x, s16 y);
00064 u16 DSP_sqrt(u32 val);
00065
00066 // 增加DSP_CALC系列的接口, 以降低参数传递带来的时间损耗
00067 #define DSP_CALC_SIN(sin, angle) \
00068 { \
00069     int32_t flg = __get_PRIMASK(); \
00070     __disable_irq(); \
00071     DSP_SC = BIT2 | BIT1; \
00072     DSP_THETA = (u16)angle; \
00073     sin = DSP_SIN; \
00074     __set_PRIMASK(flg); \
}
```

```

00075     }
00076 #define DSP_CALC_COS(cos, angle) //
00077 {
00078     int32_t flg = __get_PRIMASK(); //
00079     __disable_irq(); //
00080     DSP_SC = BIT2 | BIT1; //
00081     DSP_THETA = (u16)angle; //
00082     cos = DSP_COS; //
00083     __set_PRIMASK(flg); //
00084 }
00085 #define DSP_CALC_RMS(rms, a, b) //
00086 {
00087     int32_t flg = __get_PRIMASK(); //
00088     __disable_irq(); //
00089     DSP_DID = a; //
00090     DSP_DIS = b; //
00091     rms = DSP_REM; //
00092     __set_PRIMASK(flg); //
00093 }
00094
00095 #define DSP_CALC_ARCTAN(arctan, x, y) //
00096 {
00097     int32_t flg = __get_PRIMASK(); //
00098     __disable_irq(); //
00099     DSP_SC = BIT1; //
00100     DSP_Y = y; //
00101     arctan = DSP_ARCTAN; //
00102     __set_PRIMASK(flg); //
00103 }
00104 #define DSP_CALC_DIV(c, a, b) //
00105 {
00106     int32_t flg = __get_PRIMASK(); //
00107     __disable_irq(); //
00108     DSP_DID = (a); //
00109     DSP_DIS = (b); //
00110     c = DSP_QUO; //
00111     __set_PRIMASK(flg); //
00112 }
00113 #define DSP_CALC_MOD(mod, x, y) //
00114 {
00115     int32_t flg = __get_PRIMASK(); //
00116     __disable_irq(); //
00117     DSP_SC = BIT1; //
00118     DSP_X = x; //
00119     DSP_Y = y; //
00120     mod = DSP_MOD; //
00121     __set_PRIMASK(flg); //
00122 }
00123 #define DSP_CALC_SQRT(sqrt, val) //
00124 {
00125     int32_t flg = __get_PRIMASK(); //
00126     __disable_irq(); //
00127     DSP_RAD = val; //
00128     sqrt = DSP_SQRT; //
00129     __set_PRIMASK(flg); //
00130 }
00131
00132 #endif
00133
00134 /***** (C) COPYRIGHT LINKO SEMICONDUCTOR *****/

```

3.13 include/lks32mc08x_exti.h 文件参考

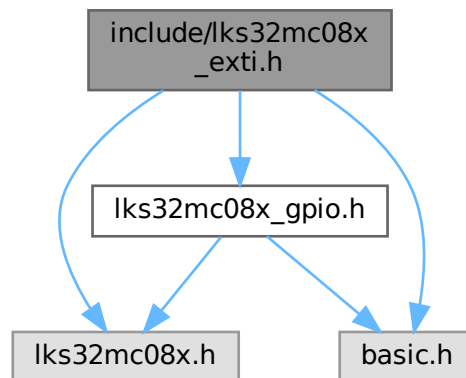
文件名称: [lks32mc08x_exti.h](#)
 文件标识: 无
 内容摘要: EXTI外部中断驱动头文件
 其它说明: 无

```

#include "lks32mc08x.h"
#include "lks32mc08x_gpio.h"
#include "basic.h"

```

lks32mc08x_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:



结构体

- struct **EXTI_TypeDef**
*EXTI*寄存器结构体句柄

宏定义

- #define **EXTI** ((**EXTI_TypeDef** *)EXTI_BASE)
*EXTI*结构体基地址定义

枚举

- enum **EXTIMode_TypeDef** { **EXTI_Trigger_None** = 0x00 , **EXTI_Trigger_Negedge** = 0x01 , **EXTI_Trigger_Posedge** = 0x02 , **EXTI_Trigger_Edge** = 0x03 }
- EXTI*触发信号定义

函数

- void **EXTI_Trigger_Config** (**GPIO_TypeDef** *GPIOx, uint32_t GPIO_PinSource, uint16_t EXTI_Trigger)
函数名称: void **EXTI_Trigger_Config**(**GPIO_TypeDef** *GPIOx, uint32_t GPIO_PinSource, uint16_t EXTI_Trigger)
- void **EXTI_ClearIRQFlag** (**GPIO_TypeDef** *GPIOx, uint32_t GPIO_PinSource)
函数名称: void **EXTI_ClearIRQFlag**(**GPIO_TypeDef** *GPIOx, uint32_t GPIO_PinSource)
- u8 **EXTI_GetIRQFlag** (**GPIO_TypeDef** *GPIOx, uint32_t GPIO_PinSource)
函数名称: u8 **EXTI_GetIRQFlag**(**GPIO_TypeDef** *GPIOx, uint32_t GPIO_PinSource)

3.13.1 详细描述

文件名称: [lks32mc08x_exti.h](#)
文件标识: 无
内容摘要: EXTI外部中断驱动头文件
其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2021年11月10日	1.0	YangZJ	创建

修改日志:

在文件 [lks32mc08x_exti.h](#) 中定义.

3.13.2 宏定义说明

3.13.2.1 EXTI

```
#define EXTI ((EXTI_TypeDef *)EXTI_BASE)
```

EXTI结构体基地址定义

在文件 [lks32mc08x_exti.h](#) 第 39 行定义.

3.13.3 枚举类型说明

3.13.3.1 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 [lks32mc08x_exti.h](#) 第 45 行定义.

3.13.4 函数说明

3.13.4.1 EXTI_ClearIRQFlag()

```
void EXTI_ClearIRQFlag (
    GPIO_TypeDef * GPIOx,
    uint32_t GPIO_PinSource )
```

函数名称: void [EXTI_ClearIRQFlag\(GPIO_TypeDef *GPIOx, uint32_t GPIO_PinSource\)](#)

功能描述: 清除外部中断标志位

参见

被引用内容: 无

参数

输入参数:	GPIOx:GPIO对象可选(GPIO0~GPIO3) GPIO_PinSource:指定的PinSource可选(GPIO_PinSource_0~GPIO_PinSource_15)
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
if (EXTI_GetIRQFlag (GPIO0, GPIO_PinSource_0)) //获取P0.0外部中断标志位
{
    EXTI_ClearIRQFlag (GPIO0, GPIO_PinSource_0); //清除P0.0外部中断标志位
}
```

Date	Version	Author	Description
2021年11月10日	1.0	YangZJ	创建

修改日志:

在文件 [lks32mc08x_exti.c](#) 第 124 行定义.

3.13.4.2 EXTI_GetIRQFlag()

```
u8 EXTI_GetIRQFlag (
    GPIO_TypeDef * GPIOx,
    uint32_t GPIO_PinSource )
```

函数名称: **u8 EXTI_GetIRQFlag(GPIO_TypeDef *GPIOx, uint32_t GPIO_PinSource)**

功能描述: 读取外部中断标志位

参见

被引用内容: 无

参数

输入参数:	GPIOx:GPIO对象可选(GPIO0~GPIO3) GPIO_PinSource:指定的PinSource可选(GPIO_PinSource_0~GPIO_PinSource_15)
输出参数:	无

返回

返回值: 中断标志, 高有效

注解

其它说明: 无

警告

无

示例代码:

```
if (EXTI_GetIRQFlag (GPIO0, GPIO_PinSource_0)) //获取P0.0外部中断标志位
{
}
```

Date	Version	Author	Description
2021年11月10日	1.0	YangZJ	创建

修改日志:

在文件 [lks32mc08x_exti.c](#) 第 80 行定义.

3.13.4.3 EXTI_Trigger_Config()

```
void EXTI_Trigger_Config (
    GPIO_TypeDef * GPIOx,
    uint32_t GPIO_PinSource,
    uint16_t EXTI_Trigger )
```

函数名称: void [EXTI_Trigger_Config\(GPIO_TypeDef *GPIOx, uint32_t GPIO_PinSource, uint16_t EXTI_Trigger\)](#)

功能描述: GPIO外部触发极性选择

参见

被引用内容: [GPIO_TypeDef](#)

参数

输入参数:	GPIOx:GPIO对象可选(GPIO0) GPIO.PinSource:指定的PadSource可选(GPIO_PinSource_0~GPIO_PinSource_15) EXTI.Trigger:指定的触发极性

参见

EXTI_Trigger参数来源: [EXTIMode_TypeDef](#)

参数

输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
EXTI_Trigger_Config(GPIO0,GPIO_PinSource_0,EXTI_Trigger_Negedge); //P0.0信号下降沿触发外部中断事件
```

Date	Version	Author	Description
2021年11月10日	1.0	YangZJ	创建

修改日志:

在文件 `lks32mc08x_exti.c` 第 38 行定义.

3.14 lks32mc08x_exti.h

[浏览该文件的文档.](#)

```
00001
00015 #ifndef __LKS32MC08x_EXTI_H
00016 #define __LKS32MC08x_EXTI_H
00017
00018 /* Includes -----*/
00019 #include "lks32mc08x.h"
00020 #include "lks32mc08x_gpio.h"
00021 #include "basic.h"
00022
00026 typedef struct
00027 {
00028     __IO uint32_t CR0;
00029     __IO uint32_t CR1;
00030     __IO uint32_t IF;
00031     __IO uint32_t POL;
00032     __IO uint32_t EN;
00033 } EXTI_TypeDef;
00034
00038 #ifndef EXTI
00039 #define EXTI ((EXTI_TypeDef *)EXTI_BASE)
00040 #endif
00041
00045 typedef enum
00046 {
00047     EXTI_Trigger_None    = 0x00,
00048     EXTI_Trigger_Negedge = 0x01,
00049     EXTI_Trigger_Posedge = 0x02,
00050     EXTI_Trigger_Edge    = 0x03,
00051 } EXTIMode_TypeDef;
00052
00053 void EXTI_Trigger_Config(GPIO_TypeDef *GPIOx, uint32_t GPIO_PinSource, uint16_t EXTI_Trigger);
00054 void EXTI_ClearIRQFlag(GPIO_TypeDef *GPIOx, uint32_t GPIO_PinSource);
00055 u8 EXTI_GetIRQFlag(GPIO_TypeDef *GPIOx, uint32_t GPIO_PinSource);
00056
00057 #endif /*__lks32mc08x_EXTI_H */
00058
00059 /***** (C) COPYRIGHT LINKO SEMICONDUCTOR *****/
```

3.15 include/lks32mc08x_flash.h 文件参考

文件名称: LKS32MC08x_flash.h
 文件标识: 无
 内容摘要: FLASH外设驱动程序
 其它说明: 无

```
#include "lks32mc08x_lib.h"
```

lks32mc08x_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_MAIN 0x000**
Flash的mian区域定义
- **#define Flash_NVR 0x800**
Flash的nvr区域定义

函数

- **int ProgramPage** (u32 adr, u32 sz, u8 *buf, u16 nvr)
函数名称: *int ProgramPage(u32 adr, u32 sz, u8 *buf, u16 nvr)*
- **void EraseSector** (u32 adr, u16 nvr)
函数名称: *void EraseSector(u32 adr, u16 nvr)*
- **void Read.More.Flash** (u32 adr, u32 sz, u32 *buf, u16 nvr)
函数名称: *void Read.More.Flash(u32 adr, u32 sz, u32 *buf, u16 nvr)*

变量

- **volatile u32 erase_flag**
flash擦除操作启动变量
- **volatile u32 prog_m_flag**
flash编程操作启动变量

3.15.1 详细描述

文件名称: LKS32MC08x_flash.h
文件标识: 无
内容摘要: FLASH外设驱动程序
其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2018年07月25日	1.0	Howlet	创建

修改日志:

Date	Version	Author	Description
2020年12月15日	1.1	DengT	修改

修改日志:增加程序跑飞时, 对Flash的保护

在文件 [lks32mc08x_flash.h](#) 中定义.

3.15.2 宏定义说明

3.15.2.1 Flash.MAIN

```
#define Flash_MAIN 0x000
```

Flash的mian区域定义

在文件 [lks32mc08x_flash.h](#) 第 25 行定义.

3.15.2.2 Flash.NVR

```
#define Flash_NVR 0x800
```

Flash的nvr区域定义

在文件 [lks32mc08x_flash.h](#) 第 26 行定义.

3.15.3 函数说明

3.15.3.1 EraseSector()

```
void EraseSector (
    u32 adr,
    u16 nvr ) [extern]
```

函数名称: void [EraseSector\(u32 adr, u16 nvr\)](#)

功能描述: Flash按扇区擦除操作

参见

被引用内容: 无

参数

输入参数:	adr: 扇区地址 (一个扇区512字节) nvr: 为0x800的时候擦除NVR, 为0时擦除主Flash erase_flag: 调用函数前必须赋值0x9A0D361F, 否则不执行擦除, 用来防程序跑飞
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

事例1:

```
erase_flag = 0x9A0D361F; //擦除解锁
EraseSector(0x00000000, Flash.MAIN); //擦除flash的main区域即存储程序区域
erase_flag = 0x00; //擦除上锁
```

事例2:

```
erase_flag = 0x9A0D361F; //擦除解锁
EraseSector(0x00000000, Flash.NVR); //擦除flash的NVR区域即独立的1KB用户存储区
erase_flag = 0x00; //擦除上锁
```

Date	Version	Author	Description
2020年12月15日	1.1	DengT	创建

修改日志:

在文件 [lks32mc08x_flash.c](#) 第 55 行定义.

3.15.3.2 ProgramPage()

```
int ProgramPage (
    u32 adr,
    u32 sz,
    u8 * buf,
    u16 nvr )
```

函数名称: `int ProgramPage(u32 adr, u32 sz, u8 *buf, u16 nvr)`

功能描述: Flash编程操作

参见

被引用内容： 无

参数

输入参数:	adr: 扇区地址 sz : 编程字节数量(0~512字节) buf: 要编程的数据数组指针 nvr: 为0x800的时候编程NVR, 为0时编程主Flash progm_flag: 调用函数前必须赋值0x9AFDA40C, 否则不执行编程, 用来防程序跑飞
输出参数:	无

返回

返回 值: 1: 编程成功; 0: 编程失败

注解

其它说明: 无

警告

无

示例代码:

```
u8 Flash_Buff[2] = {0x12,0x23};
事例1:
erase_flag = 0x9A0D361F; //擦除解锁
EraseSector(0x00000000 , Flash_MAIN); //擦除flash的main区域即存储程序区域
erase_flag = 0x00; //擦除上锁

progm_flag = 0x9AFDA40C; //编程解锁
ProgramPage(0x00000000 ,2,Flash_Buff, Flash_MAIN); //编程flash的main区域即存储程序区域
erase_flag = 0x00; //编程上锁
事例2:
erase_flag = 0x9A0D361F; //擦除解锁
EraseSector(0x00000000 , Flash_NVR); //擦除flash的NVR区域即独立的1KB用户存储区
erase_flag = 0x00; //擦除上锁

erase_flag = 0x9A0D361F; //编程解锁
EraseSector(0x00000000,2,Flash_Buff,Flash_NVR); //编程flash的NVR区域即独立的1KB用户存储区
erase_flag = 0x00; //编程上锁
```

Date	Version	Author	Description
2020年12月15日	1.1	DengT	创建

修改日志:

在文件 [lks32mc08x_flash.c](#) 第 134 行定义.

3.15.3.3 Read_More_Flash()

```
void Read_More_Flash (
    u32 adr,
```

```
u32 sz,  
u32 * buf,  
u16 nvr )
```

函数名称: void [Read_More_Flash\(u32 adr, u32 sz, u32 *buf, u16 nvr\)](#)

功能描述: 读取FLASH多地址数据函数

参见
被引用内容: 无

参数

输入参数:	adr: 扇区地址 (一个扇区512字节) nvr: 为0x800, 读取MAIN数据, 为0x0, 读取NVR数据 buf: 读取数据存储的地址 sz : 读取sz个字
输出参数:	无

返回
返 回 值: 读取的数据值

注解
其它说明: 无

警告
无

示例代码:

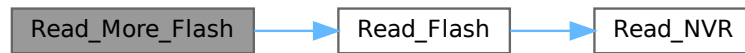
```
u32 Flash_Read_Buff[4] = {0};  
事例1: Read\_More\_Flash (0x00000000 ,4,Flash_Read_Buff,Flash.MAIN); //获取flash的main区域0x00000000地址4个word数据  
事例2: Read\_More\_Flash (0x00000000 ,4,Flash_Read_Buff,Flash.NVR); //获取flash的NVR区域0x00000000地址4个word数据
```

Date	Version	Author	Description
2020年12月15日	1.1	DengT	创建

修改日志:

在文件 [lks32mc08x_flash.c](#) 第 [272](#) 行定义.

函数调用图:



3.15.4 变量说明

3.15.4.1 erase_flag

```
volatile u32 erase_flag [extern]
```

flash擦除操作启动变量

在文件 [lks32mc08x_flash.c](#) 第 20 行定义.

3.15.4.2 prog_m_flag

```
volatile u32 prog_m_flag [extern]
```

flash编程操作启动变量

在文件 [lks32mc08x_flash.c](#) 第 24 行定义.

3.16 lks32mc08x_flash.h

[浏览该文件的文档.](#)

```
00001
00020 #ifndef __LKS32MC08X_FLASH__
00021 #define __LKS32MC08X_FLASH__
00022
00023 #include "lks32mc08x.lib.h"
00024
00025 #define Flash_MAIN 0x000
00026 #define Flash_NVR 0x800
00028 extern volatile u32 erase_flag;
00029 extern volatile u32 prog_m_flag;
00030 int ProgramPage(u32 adr, u32 sz, u8 *buf, u16 nvr);
00031 extern void EraseSector(u32 adr, u16 nvr);
00032 void ReadMoreFlash(u32 adr, u32 sz, u32 *buf, u16 nvr);
00033
00034 #endif
```

3.17 include/lks32mc08x_gpio.h 文件参考

文件名称: [lks32mc08x_gpio.h](#)

文件标识: 无

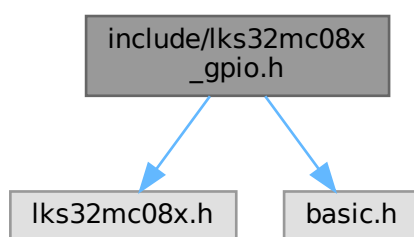
内容摘要: GPIO外设驱动程序头文件

其它说明: 无

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

```
#include "basic.h"
```

lks32mc08x_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_TypeDef](#)
GPIO寄存器结构体句柄
- struct [GPIO_InitTypeDef](#)
GPIO功能配置结构体句柄

宏定义

- #define [GPIO0](#) (([GPIO_TypeDef](#) *)GPIO0_BASE)
GPIO0结构体基地址定义
- #define [GPIO1](#) (([GPIO_TypeDef](#) *)GPIO1_BASE)
GPIO1结构体基地址定义
- #define [GPIO2](#) (([GPIO_TypeDef](#) *)GPIO2_BASE)
GPIO2结构体基地址定义
- #define [GPIO3](#) (([GPIO_TypeDef](#) *)GPIO3_BASE)
GPIO3结构体基地址定义

- #define AF0_GPIO 0
- #define AF1_SYSAF 1
- #define AF2_HALL 2
- #define AF3_MCPWM 3
- #define AF4_UART 4
- #define AF5_SPI 5
- #define AF6_I2C 6
- #define AF7_TIMER0 7
- #define AF8_TIMER23 8
- #define AF9_DEBUG 9
- #define AF10_CAN 10
- #define AF11_SIF 11
- #define GPIO_AF_GPIO 0x00
- #define GPIO_AF_SYSAF 0x01
- #define GPIO_AF_HALL 0x02
- #define GPIO_AF_MCPWM 0x03
- #define GPIO_AF_UART 0x04
- #define GPIO_AF_SPI 0x05
- #define GPIO_AF_I2C 0x06
- #define GPIO_AF_TIMER0 0x07
- #define GPIO_AF_TIMER1 0x07
- #define GPIO_AF_TIMER2 0x08
- #define GPIO_AF_TIMER3 0x08
- #define GPIO_AF_DEBUG 0x09
- #define GPIO_AF_CAN 0x0A
- #define GPIO_AF_SIF 0x0B
- #define GPIO_Pin_NONE 0
- #define GPIO_Pin_0 BIT0
- #define GPIO_Pin_1 BIT1
- #define GPIO_Pin_2 BIT2
- #define GPIO_Pin_3 BIT3
- #define GPIO_Pin_4 BIT4
- #define GPIO_Pin_5 BIT5
- #define GPIO_Pin_6 BIT6
- #define GPIO_Pin_7 BIT7
- #define GPIO_Pin_8 BIT8
- #define GPIO_Pin_9 BIT9
- #define GPIO_Pin_10 BIT10
- #define GPIO_Pin_11 BIT11
- #define GPIO_Pin_12 BIT12
- #define GPIO_Pin_13 BIT13
- #define GPIO_Pin_14 BIT14
- #define GPIO_Pin_15 BIT15
- #define GPIO_Pin_ALL (uint32_t)0xFFFF
- #define GPIO_PinSource_0 0
- #define GPIO_PinSource_1 1
- #define GPIO_PinSource_2 2
- #define GPIO_PinSource_3 3
- #define GPIO_PinSource_4 4
- #define GPIO_PinSource_5 5
- #define GPIO_PinSource_6 6
- #define GPIO_PinSource_7 7
- #define GPIO_PinSource_8 8
- #define GPIO_PinSource_9 9
- #define GPIO_PinSource_10 10

- #define GPIO_PinSource_11 11
- #define GPIO_PinSource_12 12
- #define GPIO_PinSource_13 13
- #define GPIO_PinSource_14 14
- #define GPIO_PinSource_15 15

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 **BitAction** { **Bit_RESET** = 0 , **Bit_SET** }
- GPIO寄存器写入量枚举

函数

- void **GPIO_Init** (**GPIO_TypeDef** *GPIOx, **GPIO_InitTypeDef** *GPIO_InitStruct)
- 函数名称: void **GPIO_Init**(**GPIO_TypeDef*** GPIOx, **GPIO_InitTypeDef*** GPIO_InitStruct)
- void **GPIO_StructInit** (**GPIO_InitTypeDef** *GPIO_InitStruct)
- 函数名称: void **GPIO_StructInit**(**GPIO_InitTypeDef*** GPIO_InitStruct)
- void **GPIO_LockRegister** (**GPIO_TypeDef** *GPIOx, uint16_t GPIO_Pin)
- 函数名称: void **GPIO_LockRegister**(**GPIO_TypeDef*** GPIOx, uint16_t GPIO_Pin)
- void **GPIO_UnlockRegister** (**GPIO_TypeDef** *GPIOx, uint16_t GPIO_Pin)
- 函数名称: void **GPIO_UnlockRegister**(**GPIO_TypeDef*** GPIOx, uint16_t GPIO_Pin)
- uint8_t **GPIO_ReadInputDataBit** (**GPIO_TypeDef** *GPIOx, uint16_t GPIO_Pin)
- 函数名称: uint8_t **GPIO_ReadInputDataBit**(**GPIO_TypeDef*** GPIOx, uint16_t GPIO_Pin)
- uint32_t **GPIO_ReadInputData** (**GPIO_TypeDef** *GPIOx)
- 函数名称: uint32_t **GPIO_ReadInputData**(**GPIO_TypeDef*** GPIOx)
- uint8_t **GPIO_ReadOutputDataBit** (**GPIO_TypeDef** *GPIOx, uint16_t GPIO_Pin)
- 函数名称: uint8_t **GPIO_ReadOutputDataBit**(**GPIO_TypeDef*** GPIOx, uint16_t GPIO_Pin)
- uint32_t **GPIO_ReadOutputData** (**GPIO_TypeDef** *GPIOx)
- 函数名称: uint32_t **GPIO_ReadOutputData**(**GPIO_TypeDef*** GPIOx)
- void **GPIO_SetBits** (**GPIO_TypeDef** *GPIOx, uint16_t GPIO_Pin)
- 函数名称: void **GPIO_SetBits**(**GPIO_TypeDef*** GPIOx, uint16_t GPIO_Pin)
- void **GPIO_ResetBits** (**GPIO_TypeDef** *GPIOx, uint16_t GPIO_Pin)
- 函数名称: void **GPIO_ResetBits**(**GPIO_TypeDef*** GPIOx, uint16_t GPIO_Pin)
- void **GPIO_Write** (**GPIO_TypeDef** *GPIOx, uint32_t Val)
- 函数名称: void **GPIO_Write**(**GPIO_TypeDef*** GPIOx, uint32_t Val)
- void **GPIO_WriteBit** (**GPIO_TypeDef** *GPIOx, uint16_t GPIO_Pin, **BitAction** BitVal)
- 函数名称: void **GPIO_WriteBit**(**GPIO_TypeDef*** GPIOx, uint16_t GPIO_Pin, **BitAction** BitVal)
- void **GPIO_PinAFConfig** (**GPIO_TypeDef** *GPIOx, uint32_t GPIO_PinSource, uint32_t GPIO_AF)
- 函数名称: void **GPIO_PinAFConfig**(**GPIO_TypeDef*** GPIOx, uint32_t GPIO_PinSource, uint32_t GPIO_AF)
- void **GPIO_Config** (**GPIO_TypeDef** *GPIOx, u32 GPIO_PinSource, u8 GPIO_Mode, u8 GPIO_AF)
- GPIO初始化
- void **GPIO_WritePinBit** (**GPIO_TypeDef** *GPIOx, u32 GPIO_PinSource, u8 Val)
- GPIO写一个bit
- u8 **GPIO_ReadPinBit** (**GPIO_TypeDef** *GPIOx, u32 GPIO_PinSource)
- GPIO读一个bit

3.17.1 详细描述

文件名称: [lks32mc08x_gpio.h](#)
文件标识: 无
内容摘要: GPIO外设驱动程序头文件
其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2016年6月19日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_gpio.h](#) 中定义.

3.17.2 宏定义说明

3.17.2.1 AF0.GPIO

```
#define AF0_GPIO 0
```

GPIO复用功能0, 模拟功能

在文件 [lks32mc08x_gpio.h](#) 第 105 行定义.

3.17.2.2 AF10.CAN

```
#define AF10_CAN 10
```

GPIO复用功能10, CAN功能

在文件 [lks32mc08x_gpio.h](#) 第 115 行定义.

3.17.2.3 AF11.SIF

```
#define AF11_SIF 11
```

GPIO复用功能11, SIF功能

在文件 [lks32mc08x_gpio.h](#) 第 116 行定义.

3.17.2.4 AF1.SYSAF

```
#define AF1_SYSAF 1
```

GPIO复用功能1, SYS.AF、CMP、CLOCK等数字信号输出功能

在文件 [lks32mc08x_gpio.h](#) 第 106 行定义.

3.17.2.5 AF2.HALL

```
#define AF2_HALL 2
```

GPIO复用功能2, HALL功能

在文件 [lks32mc08x_gpio.h](#) 第 107 行定义.

3.17.2.6 AF3.MCPWM

```
#define AF3_MCPWM 3
```

GPIO复用功能3, MCPWM功能

在文件 [lks32mc08x_gpio.h](#) 第 108 行定义.

3.17.2.7 AF4.UART

```
#define AF4_UART 4
```

GPIO复用功能4, UART功能

在文件 [lks32mc08x_gpio.h](#) 第 109 行定义.

3.17.2.8 AF5.SPI

```
#define AF5_SPI 5
```

GPIO复用功能5, SPI功能

在文件 [lks32mc08x_gpio.h](#) 第 110 行定义.

3.17.2.9 AF6.I2C

```
#define AF6_I2C 6
```

GPIO复用功能6, I2C功能

在文件 [lks32mc08x_gpio.h](#) 第 111 行定义.

3.17.2.10 AF7_TIMER01

```
#define AF7_TIMER01 7
```

GPIO复用功能7, Timer0、Timer1功能

在文件 [lks32mc08x_gpio.h](#) 第 112 行定义.

3.17.2.11 AF8_TIMER23

```
#define AF8_TIMER23 8
```

GPIO复用功能8, Timer2、Timer3、QEP0、QEP1功能

在文件 [lks32mc08x_gpio.h](#) 第 113 行定义.

3.17.2.12 AF9_DEBUG

```
#define AF9_DEBUG 9
```

GPIO复用功能9, ADC trigger debug功能

在文件 [lks32mc08x_gpio.h](#) 第 114 行定义.

3.17.2.13 GPIO0

```
#define GPIO0 ((GPIO_TypeDef *)GPIO0_BASE)
```

GPIO0结构体基地址定义

在文件 [lks32mc08x_gpio.h](#) 第 44 行定义.

3.17.2.14 GPIO1

```
#define GPIO1 ((GPIO_TypeDef *)GPIO1_BASE)
```

GPIO1结构体基地址定义

在文件 [lks32mc08x_gpio.h](#) 第 50 行定义.

3.17.2.15 GPIO2

```
#define GPIO2 ((GPIO_TypeDef *)GPIO2_BASE)
```

GPIO2结构体基地址定义

在文件 [lks32mc08x_gpio.h](#) 第 56 行定义.

3.17.2.16 GPIO3

```
#define GPIO3 ((GPIO_TypeDef *) GPIO3_BASE)
```

GPIO3结构体基地址定义

在文件 [lks32mc08x_gpio.h](#) 第 62 行定义.

3.17.2.17 GPIO_AF_CAN

```
#define GPIO_AF_CAN 0x0A
```

在文件 [lks32mc08x_gpio.h](#) 第 130 行定义.

3.17.2.18 GPIO_AF_DEBUG

```
#define GPIO_AF_DEBUG 0x09
```

在文件 [lks32mc08x_gpio.h](#) 第 129 行定义.

3.17.2.19 GPIO_AF_GPIO

```
#define GPIO_AF_GPIO 0x00
```

在文件 [lks32mc08x_gpio.h](#) 第 118 行定义.

3.17.2.20 GPIO_AF_HALL

```
#define GPIO_AF_HALL 0x02
```

在文件 [lks32mc08x_gpio.h](#) 第 120 行定义.

3.17.2.21 GPIO_AF_I2C

```
#define GPIO_AF_I2C 0x06
```

在文件 [lks32mc08x_gpio.h](#) 第 124 行定义.

3.17.2.22 GPIO_AF_MCPWM

```
#define GPIO_AF_MCPWM 0x03
```

在文件 [lks32mc08x_gpio.h](#) 第 121 行定义.



3.17.2.23 GPIO_AF_SIF

```
#define GPIO_AF_SIF 0x0B
```

在文件 [lks32mc08x_gpio.h](#) 第 131 行定义.

3.17.2.24 GPIO_AF_SPI

```
#define GPIO_AF_SPI 0x05
```

在文件 [lks32mc08x_gpio.h](#) 第 123 行定义.

3.17.2.25 GPIO_AF_SYSAF

```
#define GPIO_AF_SYSAF 0x01
```

在文件 [lks32mc08x_gpio.h](#) 第 119 行定义.

3.17.2.26 GPIO_AF_TIMER0

```
#define GPIO_AF_TIMER0 0x07
```

在文件 [lks32mc08x_gpio.h](#) 第 125 行定义.

3.17.2.27 GPIO_AF_TIMER1

```
#define GPIO_AF_TIMER1 0x07
```

在文件 [lks32mc08x_gpio.h](#) 第 126 行定义.

3.17.2.28 GPIO_AF_TIMER2

```
#define GPIO_AF_TIMER2 0x08
```

在文件 [lks32mc08x_gpio.h](#) 第 127 行定义.

3.17.2.29 GPIO_AF_TIMER3

```
#define GPIO_AF_TIMER3 0x08
```

在文件 [lks32mc08x_gpio.h](#) 第 128 行定义.

3.17.2.30 GPIO_AF_UART

```
#define GPIO_AF_UART 0x04
```

在文件 [lks32mc08x_gpio.h](#) 第 122 行定义.

3.17.2.31 GPIO_Pin_0

```
#define GPIO_Pin_0 BIT0
```

<GPIO pin脚定义, pin_0脚

在文件 [lks32mc08x_gpio.h](#) 第 135 行定义.

3.17.2.32 GPIO_Pin_1

```
#define GPIO_Pin_1 BIT1
```

GPIO pin脚定义, pin_1脚

在文件 [lks32mc08x_gpio.h](#) 第 136 行定义.

3.17.2.33 GPIO_Pin_10

```
#define GPIO_Pin_10 BIT10
```

GPIO pin脚定义, pin_10脚

在文件 [lks32mc08x_gpio.h](#) 第 145 行定义.

3.17.2.34 GPIO_Pin_11

```
#define GPIO_Pin_11 BIT11
```

GPIO pin脚定义, pin_11脚

在文件 [lks32mc08x_gpio.h](#) 第 146 行定义.

3.17.2.35 GPIO_Pin_12

```
#define GPIO_Pin_12 BIT12
```

GPIO pin脚定义, pin_12脚

在文件 [lks32mc08x_gpio.h](#) 第 147 行定义.

3.17.2.36 GPIO_Pin_13

```
#define GPIO_Pin_13 BIT13
```

GPIO pin脚定义, pin_13脚

在文件 [lks32mc08x_gpio.h](#) 第 148 行定义.

3.17.2.37 GPIO_Pin_14

```
#define GPIO_Pin_14 BIT14
```

GPIO pin脚定义，pin.14脚

在文件 [lks32mc08x_gpio.h](#) 第 149 行定义.

3.17.2.38 GPIO_Pin_15

```
#define GPIO_Pin_15 BIT15
```

GPIO pin脚定义，pin.15脚

在文件 [lks32mc08x_gpio.h](#) 第 150 行定义.

3.17.2.39 GPIO_Pin_2

```
#define GPIO_Pin_2 BIT2
```

GPIO pin脚定义，pin.2脚

在文件 [lks32mc08x_gpio.h](#) 第 137 行定义.

3.17.2.40 GPIO_Pin_3

```
#define GPIO_Pin_3 BIT3
```

GPIO pin脚定义，pin.3脚

在文件 [lks32mc08x_gpio.h](#) 第 138 行定义.

3.17.2.41 GPIO_Pin_4

```
#define GPIO_Pin_4 BIT4
```

GPIO pin脚定义，pin.4脚

在文件 [lks32mc08x_gpio.h](#) 第 139 行定义.

3.17.2.42 GPIO_Pin_5

```
#define GPIO_Pin_5 BIT5
```

GPIO pin脚定义，pin.5脚

在文件 [lks32mc08x_gpio.h](#) 第 140 行定义.

3.17.2.43 GPIO_Pin_6

```
#define GPIO_Pin_6 BIT6
```

GPIO pin脚定义，pin.6脚

在文件 [lks32mc08x_gpio.h](#) 第 141 行定义.

3.17.2.44 GPIO_Pin_7

```
#define GPIO_Pin_7 BIT7
```

GPIO pin脚定义，pin.7脚

在文件 [lks32mc08x_gpio.h](#) 第 142 行定义.

3.17.2.45 GPIO_Pin_8

```
#define GPIO_Pin_8 BIT8
```

GPIO pin脚定义，pin.8脚

在文件 [lks32mc08x_gpio.h](#) 第 143 行定义.

3.17.2.46 GPIO_Pin_9

```
#define GPIO_Pin_9 BIT9
```

GPIO pin脚定义，pin.9脚

在文件 [lks32mc08x_gpio.h](#) 第 144 行定义.

3.17.2.47 GPIO_Pin_ALL

```
#define GPIO_Pin_ALL (uint32_t)0xFFFF
```

GPIO pin脚定义，所有pin脚

在文件 [lks32mc08x_gpio.h](#) 第 151 行定义.

3.17.2.48 GPIO_Pin_NONE

```
#define GPIO_Pin_NONE 0
```

GPIO pin脚定义，无pin脚

在文件 [lks32mc08x_gpio.h](#) 第 134 行定义.

3.17.2.49 GPIO_PinSource_0

```
#define GPIO_PinSource_0 0
```

GPIO pin脚定义，pin_0脚

在文件 [lks32mc08x_gpio.h](#) 第 154 行定义.

3.17.2.50 GPIO_PinSource_1

```
#define GPIO_PinSource_1 1
```

GPIO pin脚定义，pin_1脚

在文件 [lks32mc08x_gpio.h](#) 第 155 行定义.

3.17.2.51 GPIO_PinSource_10

```
#define GPIO_PinSource_10 10
```

GPIO pin脚定义，pin_10脚

在文件 [lks32mc08x_gpio.h](#) 第 164 行定义.

3.17.2.52 GPIO_PinSource_11

```
#define GPIO_PinSource_11 11
```

GPIO pin脚定义，pin_11脚

在文件 [lks32mc08x_gpio.h](#) 第 165 行定义.

3.17.2.53 GPIO_PinSource_12

```
#define GPIO_PinSource_12 12
```

GPIO pin脚定义，pin_12脚

在文件 [lks32mc08x_gpio.h](#) 第 166 行定义.

3.17.2.54 GPIO_PinSource_13

```
#define GPIO_PinSource_13 13
```

GPIO pin脚定义，pin_13脚

在文件 [lks32mc08x_gpio.h](#) 第 167 行定义.

3.17.2.55 GPIO_PinSource_14

```
#define GPIO_PinSource_14 14
```

GPIO pin脚定义，pin.14脚

在文件 [lks32mc08x_gpio.h](#) 第 168 行定义.

3.17.2.56 GPIO_PinSource_15

```
#define GPIO_PinSource_15 15
```

GPIO pin脚定义，pin.15脚

在文件 [lks32mc08x_gpio.h](#) 第 169 行定义.

3.17.2.57 GPIO_PinSource_2

```
#define GPIO_PinSource_2 2
```

GPIO pin脚定义，pin.2脚

在文件 [lks32mc08x_gpio.h](#) 第 156 行定义.

3.17.2.58 GPIO_PinSource_3

```
#define GPIO_PinSource_3 3
```

GPIO pin脚定义，pin.3脚

在文件 [lks32mc08x_gpio.h](#) 第 157 行定义.

3.17.2.59 GPIO_PinSource_4

```
#define GPIO_PinSource_4 4
```

GPIO pin脚定义，pin.4脚

在文件 [lks32mc08x_gpio.h](#) 第 158 行定义.

3.17.2.60 GPIO_PinSource_5

```
#define GPIO_PinSource_5 5
```

GPIO pin脚定义，pin.5脚

在文件 [lks32mc08x_gpio.h](#) 第 159 行定义.

3.17.2.61 GPIO_PinSource_6

```
#define GPIO_PinSource_6 6
```

GPIO pin脚定义，pin.6脚

在文件 [lks32mc08x_gpio.h](#) 第 160 行定义.

3.17.2.62 GPIO_PinSource_7

```
#define GPIO_PinSource_7 7
```

GPIO pin脚定义，pin.7脚

在文件 [lks32mc08x_gpio.h](#) 第 161 行定义.

3.17.2.63 GPIO_PinSource_8

```
#define GPIO_PinSource_8 8
```

GPIO pin脚定义，pin.8脚

在文件 [lks32mc08x_gpio.h](#) 第 162 行定义.

3.17.2.64 GPIO_PinSource_9

```
#define GPIO_PinSource_9 9
```

GPIO pin脚定义，pin.9脚

在文件 [lks32mc08x_gpio.h](#) 第 163 行定义.

3.17.3 枚举类型说明

3.17.3.1 BitAction

```
enum BitAction
```

GPIO寄存器写入量枚举

枚举值

Bit_RESET	
Bit_SET	

在文件 [lks32mc08x_gpio.h](#) 第 88 行定义.

3.17.3.2 GPIO_Mode_TypeDef

```
enum GPIO_Mode_TypeDef
```

GPIO模式选择枚举

枚举值

GPIO_Mode_IN	GPIO Input Mode
GPIO_Mode_OUT	GPIO Output Mode
GPIO_Mode_ANA	GPIO Analog In/Out Mode,用作ANA时需禁止输入输出
GPIO_Mode_IO	GPIO 输入&输出，通常和开漏配合使用，打开内部上拉可选，可以实现线与电平

在文件 [lks32mc08x_gpio.h](#) 第 68 行定义.

3.17.3.3 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 [lks32mc08x_gpio.h](#) 第 79 行定义.

3.17.4 函数说明

3.17.4.1 GPIO_Config()

```
void GPIO_Config (
    GPIO_TypeDef * GPIOx,
    u32 GPIO_PinSource,
    u8 GPIO_Mode,
    u8 GPIO_AF )
```

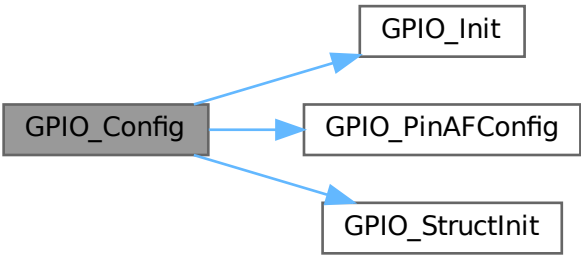
GPIO初始化

参数

GPIOx	GPIO对象,参数可选 (GPIO0~GPIO3)
GPIO_PinSource	指定的PinSource (GPIO_PinSource_0~GPIO_PinSource_15)
GPIO_Mode	GPIO模式
GPIO_AF	GPIO复用功能

在文件 [lks32mc08x_gpio.c](#) 第 618 行定义.

函数调用图:



3.17.4.2 GPIO_Init()

```
void GPIO_Init (
    GPIO_TypeDef * GPIOx,
    GPIO_InitTypeDef * GPIO_InitStruct )
```

函数名称: void [GPIO_Init\(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct\)](#)

功能描述: GPIO初始化函数

参见
被引用内容: 无

参数

输入参数:	GPIOx:GPIO对象,参数可选 (GPIO0~GPIO3) GPIO_InitStruct: GPIO初始化结构体
输出参数:	无

返回
返回值: 无

注解
其它说明: 无

警告

无

示例代码:

```
GPIO_InitTypeDef GPIO_InitStructure;
//P2.11输入模式
GPIO_StructInit(&GPIO_InitStructure); //初始化结构体
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN; //GPIO输入模式
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_11;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init(GPIO2, &GPIO_InitStructure);
//P0.2输出模式
GPIO_StructInit(&GPIO_InitStructure); //初始化结构体
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT; //GPIO输出模式
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init(GPIO0, &GPIO_InitStructure);
//P3.2开漏输出
GPIO_StructInit(&GPIO_InitStructure); //初始化结构体
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT; //GPIO输出模式
GPIO_InitStructure.GPIO_ODEna = ENABLE; //使能开漏模式
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL; //关闭上拉, 配置为高阻态
GPIO_Init(GPIO3, &GPIO_InitStructure);
```

Date	Version	Author	Description
2016年4月18日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_gpio.c](#) 第 57 行定义.

这是这个函数的调用关系图:



3.17.4.3 GPIO_LockRegister()

```
void GPIO_LockRegister (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

函数名称: void [GPIO_LockRegister\(GPIO_TypeDef* GPIOx,uint16_t GPIO_Pin\)](#)

功能描述: GPIO寄存器保护锁定

参见

被引用内容: 无

参数

输入参数:	GPIOx:GPIO对象,参数可选 (GPIO0~GPIO3) GPIO_Pin:指定的Pin (GPIO_Pin_0~GPIO_Pin_15)
输出参数:	无

返回

返回值: 无

注解

其它说明: 上锁后GPIO输入/输出/上下拉/开漏/功能选择不能被修改

警告

无

示例代码:

```
GPIO_LockRegister(GPIO0,GPIO_Pin_0); //P0.0上锁
```

Date	Version	Author	Description
2016年4月18日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_gpio.c](#) 第 424 行定义.

3.17.4.4 GPIO_PinAFConfig()

```
void GPIO_PinAFConfig (
    GPIO_TypeDef * GPIOx,
    uint32_t GPIO_PinSource,
    uint32_t GPIO_AF )
```

函数名称: void [GPIO_PinAFConfig\(GPIO_TypeDef* GPIOx, uint32_t GPIO_PinSource, uint32_t GPIO_AF\)](#)

功能描述: GPIO复用功能选择

参见

被引用内容: 无

参数

输入参数:	GPIOx:GPIO对象,参数可选 (GPIO0~GPIO3)	
	GPIO_PadSource:指定的PadSource (GPIO_PinSource_0~GPIO_PinSource_15)	
	GPIO_AF:指定复用功能	
	复用功能名称	说明
	AF0.GPIO	模拟功能
	AF1.SYSAF	SYS_AF、CMP、CLOCK等数字信号输出功能
	AF2.HALL	HALL功能
	AF3.MCPWM	MCPWM功能
	AF4.UART	UART功能
	AF5.SPI	SPI功能
	AF6.I2C	I2C功能
	AF7.TIMER01	Timer0、Timer1功能
	AF8.TIMER23	Timer2、Timer3、QEP0、QEP1功能
	AF9.DEBUG	ADC trigger debug功能
输出参数:	AF10.CAN	CAN功能
	AF11.SIF	SIF功能
无		

返回

返回 值: 无

注解

其它说明: 无

警告

无

示例代码:

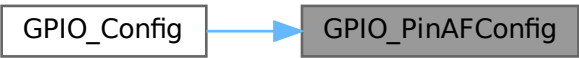
```
GPIO_PinAFConfig(GPIO1,GPIO_Pin_0, AF4_UART); //P1.0复用为串口功能
```

Date	Version	Author	Description
2016年4月18日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_gpio.c](#) 第 501 行定义.

这是这个函数的调用关系图:



3.17.4.5 GPIO_ReadInputData()

```
uint32_t GPIO_ReadInputData (
    GPIO_TypeDef * GPIOx )
```

函数名称: uint32_t [GPIO_ReadInputData\(GPIO_TypeDef* GPIOx\)](#)

功能描述: 读取GPIO的输入数据

参见

被引用内容: 无

参数

输入参数:	GPIOx:GPIO对象,参数可选 (GPIO0~GPIO3)
输出参数:	无

返回

返回值: 输入数据

注解

其它说明: 无

警告

无

示例代码:

```
u8 GPIO_IN_VALUE = 0;
GPIO_IN_VALUE = GPIO\_ReadInputData\(GPIO0\); //读取P0.0~P0.15输入电平
```


Date	Version	Author	Description
2016年4月18日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_gpio.c](#) 第 224 行定义.

3.17.4.6 GPIO_ReadInputDataBit()

```
uint8_t GPIO_ReadInputDataBit (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

函数名称: `uint8_t GPIO_ReadInputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)`

功能描述: 读取GPIO的指定Pin的输入值

参见

被引用内容: 无

参数

输入参数:	GPIOx:GPIO对象,参数可选 (GPIO0~GPIO3) GPIO_Pin:指定的Pin (GPIO_Pin_0~GPIO_Pin_15)
输出参数:	无

返回

返回 值: 指定Pin输入数据

注解

其它说明: 无

警告

无

示例代码:

```
u8 GPIO_IN_VALUE = 0;
GPIO_IN_VALUE = GPIO_ReadInputDataBit (GPIO0,GPIO_Pin_0) ;//读取P0.0输入电平 1为高电平, 0为低电平
```

Date	Version	Author	Description
2016年4月18日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_gpio.c](#) 第 189 行定义.

3.17.4.7 GPIO_ReadOutputData()

```
uint32_t GPIO_ReadOutputData (
    GPIO_TypeDef * GPIOx )
```

函数名称: `uint32_t GPIO_ReadOutputData(GPIO_TypeDef* GPIOx)`

功能描述: 读取GPIO的输出数据

参见

被引用内容: 无

参数

输入参数:	GPIOx:GPIO对象,参数可选 (GPIO0~GPIO3)
输出参数:	无

返回

返回值: 输出数据

注解

其它说明: 无

警告

无

示例代码:

```
u8 GPIO_OUT_VALUE = 0;
GPIO_OUT_VALUE = GPIO\_ReadOutputData(GPIO0); //读取P0.0~P0.15输出电平
```

Date	Version	Author	Description
2016年4月18日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_gpio.c](#) 第 286 行定义.

3.17.4.8 GPIO_ReadOutputDataBit()

```
uint8_t GPIO_ReadOutputDataBit (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

函数名称: `uint8_t GPIO_ReadOutputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)`

功能描述: 读取GPIO的指定Pin的输出值

参见
被引用内容: 无

参数

输入参数:	GPIOx:GPIO对象,参数可选 (GPIO0~GPIO3) GPIO_Pin:指定的Pin (GPIO_Pin_0~GPIO_Pin_15)
输出参数:	无

返回
返回值: 指定IO输出数据

注解
其它说明: 无

警告
无

示例代码:

```
u8 GPIO_OUT_VALUE = 0;
GPIO_OUT_VALUE = GPIO_ReadOutputDataBit (GPIO0,GPIO_Pin_0); //读取P0.0输出电平 1为高电平，0为低电平
```

Date	Version	Author	Description
2016年4月18日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_gpio.c](#) 第 251 行定义.

3.17.4.9 GPIO_ReadPinBit()

```
u8 GPIO_ReadPinBit (
    GPIO_TypeDef * GPIOx,
    u32 GPIO_PinSource )
```

GPIO读一个bit

参数

GPIOx	GPIO对象,参数可选 (GPIO0~GPIO3)
GPIO_PinSource	指定的PinSource (GPIO_PinSource_0~GPIO_PinSource_15)

返回

0/1

在文件 [lks32mc08x_gpio.c](#) 第 655 行定义.

3.17.4.10 GPIO_ResetBits()

```
void GPIO_ResetBits (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

函数名称: void [GPIO_ResetBits\(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin\)](#)

功能描述: GPIO指定Pin置1

参见

被引用内容: 无

参数

输入参数:	GPIOx:GPIO对象,参数可选 (GPIO0~GPIO3) GPIO_Pin:指定的Pin (GPIO_Pin_0~GPIO_Pin_15)
输出参数:	无

返回

返回值: 无

注解

其它说明： 无

警告

无

示例代码:

```
GPIO_ResetBits (GPIO0,GPIO_Pin_0); //P0.0输出低电平
```

Date	Version	Author	Description
2016年4月18日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_gpio.c](#) 第 338 行定义.

函数调用图:



3.17.4.11 GPIO_SetBits()

```
void GPIO_SetBits (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

函数名称: void [GPIO_SetBits\(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin\)](#)

功能描述: GPIO指定Pin置1

参见

被引用内容： 无

参数

输入参数:	GPIOx:GPIO对象,参数可选 (GPIO0~GPIO3) GPIO_Pin:指定的Pin (GPIO_Pin_0~GPIO_Pin_15)
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

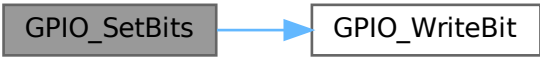
示例代码:

```
GPIO_SetBits(GPIO0,GPIO_Pin_0); //P0.0输出高电平
```

Date	Version	Author	Description
2016年4月18日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_gpio.c](#) 第 [312](#) 行定义.
函数调用图:



3.17.4.12 GPIO_StructInit()

```
void GPIO_StructInit (
    GPIO_InitTypeDef * GPIO_InitStruct )
```

函数名称: void [GPIO_StructInit\(GPIO_InitTypeDef* GPIO_InitStruct\)](#)
功能描述: GPIO结构体初始化

参见

被引用内容： 无

参数

输入参数:	GPIO_InitStruct: GPIO初始化结构体
输出参数:	无

返回

返回 值: 无

注解

其它说明： 无

警告

无

示例代码:

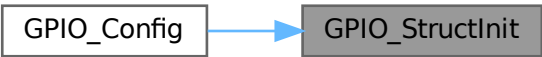
```
GPIO_InitTypeDef GPIO_InitStructure;
GPIO_StructInit(&GPIO_InitStructure); //初始化结构体
```

Date	Version	Author	Description
2016年4月18日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_gpio.c](#) 第 159 行定义.

这是这个函数的调用关系图:



3.17.4.13 GPIO_UnlockRegister()

```
void GPIO_UnlockRegister (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

函数名称: void **GPIO_UnlockRegister**(GPIO_TypeDef* GPIOx,uint16_t GPIO_Pin)

功能描述: GPIO寄存器保护解锁

参见

被引用内容: 无

参数

输入参数:	GPIOx:GPIO对象,参数可选 (GPIO0~GPIO3) GPIO_Pin:指定的Pin (GPIO_Pin_0~GPIO_Pin_15)
输出参数:	无

返回

返回值: 无

注解

其它说明: 解锁后GPIO输入/输出/上下拉/开漏/功能选择可以被修改

警告

无

示例代码:

```
GPIO_UnlockRegister(GPIO0,GPIO_Pin_0); //P0.0解锁
```

Date	Version	Author	Description
2016年4月18日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_gpio.c](#) 第 454 行定义.

3.17.4.14 GPIO_Write()

```
void GPIO_Write (
```



```
GPIO_TypeDef * GPIOx,
uint32_t Val )
```

函数名称: void GPIO_Write(GPIO_TypeDef* GPIOx, uint32_t Val)

功能描述: 向GPIO写入数据

参见

被引用内容: 无

参数

输入参数:	GPIOx:GPIO对象,参数可选 (GPIO0~GPIO3) BitVal:写入的Bit值 (0x00~0xffff)
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
GPIO_WriteBit(GPIO0, 0x01); //P0.0置1, 输出高电平, P0.1~P0.15值不变保持上一时刻状态
```

Date	Version	Author	Description
2016年4月18日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x-gpio.c](#) 第 398 行定义.

3.17.4.15 GPIO_WriteBit()

```
void GPIO_WriteBit (
    GPIO_TypeDef * GPIOx,
```

```
uint16_t GPIO_Pin,  
BitAction BitVal )
```

函数名称: void **GPIO_WriteBit**(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, BitAction BitVal)

功能描述: 向GPIO指定的Pin写入数据

参见

被引用内容: 无

参数

输入参数:	GPIOx:GPIO对象,参数可选 (GPIO0~GPIO3) GPIO_Pin:指定的Pin (GPIO_Pin_0~GPIO_Pin_15) BitVal:写入的Bit值 (Bit_RESET 或 Bit_SET)
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

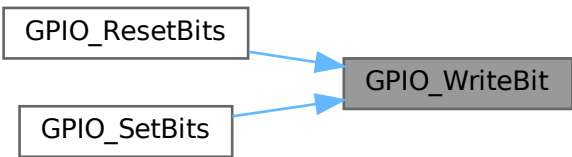
```
GPIO_WriteBit(GPIO0,GPIO_Pin_0,Bit_RESET); //P0.0输出低电平
```

Date	Version	Author	Description
2016年4月18日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_gpio.c](#) 第 365 行定义.

这是这个函数的调用关系图:



3.17.4.16 GPIO_WritePinBit()

```
void GPIO_WritePinBit (
    GPIO_TypeDef * GPIOx,
    u32 GPIO_PinSource,
    u8 Val )
```

GPIO写一个bit

参数

<i>GPIOx</i>	GPIO对象,参数可选 (GPIO0~GPIO3)
<i>GPIO_PinSource</i>	指定的PinSource (GPIO_PinSource_0~GPIO_PinSource_15)
<i>Val</i>	0/1

在文件 [lks32mc08x_gpio.c](#) 第 633 行定义.

3.18 lks32mc08x_gpio.h

[浏览该文件的文档.](#)

```
00001
00014 #ifndef __LKS32MC08x_GPIO_H
00015 #define __LKS32MC08x_GPIO_H
00016
00017 /* Includes -----*/
00018 #include "lks32mc08x.h"
00019 #include "basic.h"
00020
00024 typedef struct
00025 {
00026     __IO uint32_t PIE;
00027     __IO uint32_t POE;
00028     __IO uint32_t PDI;
00029     __IO uint32_t PDO;
00030     __IO uint32_t PUE;
00031     __IO uint32_t PDE;
00032     __IO uint32_t PODE;
00033     __IO uint32_t LCKR;
00034     __IO uint32_t F3210;
00035     __IO uint32_t F7654;
00036     __IO uint32_t FBA98;
00037     __IO uint32_t FFEDC;
00038 } GPIO_TypeDef;
```

```

00039
00043 #ifndef GPIO0
00044 #define GPIO0 ((GPIO_TypeDef *)GPIO0_BASE)
00045 #endif
00049 #ifndef GPIO1
00050 #define GPIO1 ((GPIO_TypeDef *)GPIO1_BASE)
00051 #endif
00055 #ifndef GPIO2
00056 #define GPIO2 ((GPIO_TypeDef *)GPIO2_BASE)
00057 #endif
00061 #ifndef GPIO3
00062 #define GPIO3 ((GPIO_TypeDef *)GPIO3_BASE)
00063 #endif
00064
00068 typedef enum
00069 {
00070     GPIO_Mode_IN   = 0x00,
00071     GPIO_Mode_OUT  = 0x01,
00072     GPIO_Mode_ANA  = 0x02,
00073     GPIO_Mode_IO   = 0x03,
00074 } GPIO_Mode_TypeDef;
00075
00079 typedef enum
00080 {
00081     GPIO_PuPd_NOPULL = 0x00,
00082     GPIO_PuPd_UP     = 0x01,
00083 } GPIO_PuPd_TypeDef;
00084
00088 typedef enum
00089 {
00090     Bit_RESET = 0, // 写0
00091     Bit_SET   = 1, // 写1
00092 } BitAction;
00093
00097 typedef struct
00098 {
00099     uint32_t GPIO_Pin;
00100     GPIO_Mode_TypeDef GPIO_Mode;
00101     GPIO_PuPd_TypeDef GPIO_PuPd;
00102     uint32_t GPIO_PODena;
00103 } GPIO_InitTypeDef;
00104
00105 #define AF0_GPIO      0
00106 #define AF1_SYSAF     1
00107 #define AF2_HALL      2
00108 #define AF3_MCPWM     3
00109 #define AF4_UART      4
00110 #define AF5_SPI       5
00111 #define AF6_I2C       6
00112 #define AF7_TIMER01   7
00113 #define AF8_TIMER23   8
00114 #define AF9_DEBUG     9
00115 #define AF10_CAN      10
00116 #define AF11_SIF      11
00118 #define GPIO_AF_GPIO  0x00
00119 #define GPIO_AF_SYSAF 0x01
00120 #define GPIO_AF_HALL  0x02
00121 #define GPIO_AF_MCPWM 0x03
00122 #define GPIO_AF_UART  0x04
00123 #define GPIO_AF_SPI   0x05
00124 #define GPIO_AF_I2C   0x06
00125 #define GPIO_AF_TIMER0 0x07
00126 #define GPIO_AF_TIMER1 0x07
00127 #define GPIO_AF_TIMER2 0x08
00128 #define GPIO_AF_TIMER3 0x08
00129 #define GPIO_AF_DEBUG 0x09
00130 #define GPIO_AF_CAN   0x0A
00131 #define GPIO_AF_SIF   0x0B
00132
00133 /*GPIO pin脚定义*/
00134 #define GPIO_Pin_NONE 0
00135 #define GPIO_Pin_0    BIT0
00136 #define GPIO_Pin_1    BIT1
00137 #define GPIO_Pin_2    BIT2
00138 #define GPIO_Pin_3    BIT3
00139 #define GPIO_Pin_4    BIT4
00140 #define GPIO_Pin_5    BIT5
00141 #define GPIO_Pin_6    BIT6
00142 #define GPIO_Pin_7    BIT7
00143 #define GPIO_Pin_8    BIT8
00144 #define GPIO_Pin_9    BIT9
00145 #define GPIO_Pin_10   BIT10
00146 #define GPIO_Pin_11   BIT11
00147 #define GPIO_Pin_12   BIT12
00148 #define GPIO_Pin_13   BIT13
00149 #define GPIO_Pin_14   BIT14
00150 #define GPIO_Pin_15   BIT15

```

```

00151 #define GPIO_Pin_ALL (uint32_t)0xFFFF
00153 /*GPIO复用功能pin脚定义*/
00154 #define GPIO_PinSource_0 0
00155 #define GPIO_PinSource_1 1
00156 #define GPIO_PinSource_2 2
00157 #define GPIO_PinSource_3 3
00158 #define GPIO_PinSource_4 4
00159 #define GPIO_PinSource_5 5
00160 #define GPIO_PinSource_6 6
00161 #define GPIO_PinSource_7 7
00162 #define GPIO_PinSource_8 8
00163 #define GPIO_PinSource_9 9
00164 #define GPIO_PinSource_10 10
00165 #define GPIO_PinSource_11 11
00166 #define GPIO_PinSource_12 12
00167 #define GPIO_PinSource_13 13
00168 #define GPIO_PinSource_14 14
00169 #define GPIO_PinSource_15 15
00171 /* 初始化、配置函数 *****/
00172 void GPIO_Init(GPIO_TypeDef *GPIOx, GPIO_InitTypeDef *GPIO_InitStruct);
00173 void GPIO_StructInit(GPIO_InitTypeDef *GPIO_InitStruct);
00174
00175 void GPIO_LockRegister(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin);
00176 void GPIO_UnlockRegister(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin);
00177
00178 /* GPIO读写函数 *****/
00179 uint8_t GPIO_ReadInputDataBit(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin);
00180 uint32_t GPIO_ReadInputData(GPIO_TypeDef *GPIOx);
00181 uint8_t GPIO_ReadOutputDataBit(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin);
00182 uint32_t GPIO_ReadOutputData(GPIO_TypeDef *GPIOx);
00183
00184 void GPIO_SetBits(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin);
00185 void GPIO_ResetBits(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin);
00186 void GPIO_Write(GPIO_TypeDef *GPIOx, uint32_t Val);
00187 void GPIO_WriteBit(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin, BitAction BitVal);
00188
00189 /* GPIO功能选择函数 *****/
00190 void GPIO_PinAFConfig(GPIO_TypeDef *GPIOx, uint32_t GPIO_PinSource, uint32_t GPIO_AF);
00191
00192 void GPIO_Config(GPIO_TypeDef *GPIOx, u32 GPIO_PinSource, u8 GPIO_Mode, u8 GPIO_AF);
00193 void GPIO_WritePinBit(GPIO_TypeDef *GPIOx, u32 GPIO_PinSource, u8 Val);
00194 u8 GPIO_ReadPinBit(GPIO_TypeDef *GPIOx, u32 GPIO_PinSource);
00195 #endif /*_lks32mc08x_GPIO_H */
00196
00197 /***** (C) COPYRIGHT LINKO SEMICONDUCTOR *****END OF FILE*****/

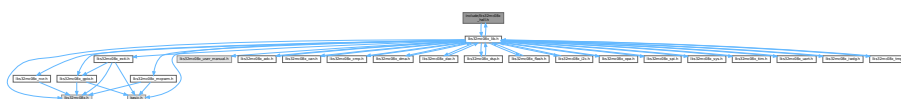
```

3.19 include/lks32mc08x_hall.h 文件参考

文件名称: lks32mc08x_hall.h
文件标识: 无
内容摘要: HALL驱动头文件
其它说明: 无

```
#include "lks32mc08x_lib.h"
```

lks32mc08x_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_TypeDef](#)
HALL寄存器结构体句柄
- struct [HALL_InitTypeDef](#)
HALL结构体基地址定义

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)0x00010000)
HALL中断使能和标志定义, HALL信号变化中断
- #define [HALL_OVERFLOW_EVENT](#) ((uint32_t)0x00020000)
HALL中断使能和标志定义, HALL计数溢出中断

函数

- void [HALL_Init](#) ([HALL_InitTypeDef](#) *HALL_InitStruct)
函数名称: [void HALL_Init\(HALL_InitTypeDef* HALL_InitStruct\)](#)
- void [HALL_StructInit](#) ([HALL_InitTypeDef](#) *HALL_InitStruct)
函数名称: [void HALL_StructInit\(HALL_InitTypeDef* HALL_InitStruct\)](#)
- uint32_t [HALL_GetFilterValue](#) (void)
函数名称: [uint32_t HALL_GetFilterValue\(void\)](#)
- uint32_t [HALL_GetCaptureValue](#) (void)
函数名称: [uint32_t HALL_GetCaptureValue\(void\)](#)
- uint32_t [HALL_WIDCount](#) (void)
函数名称: [uint32_t HALL_WIDCount\(void\)](#)
- uint32_t [HALL_GetCount](#) (void)
函数名称: [uint32_t HALL_GetCount\(void\)](#)
- uint32_t [HALL_GetIRQFlag](#) (uint32_t tempFlag)
函数名称: [uint32_t HALL_GetIRQFlag\(uint32_t tempFlag\)](#)
- void [HALL_ClearIRQFlag](#) (uint32_t tempFlag)
函数名称: [uint32_t HALL_ClearIRQFlag\(uint32_t tempFlag\)](#)

3.19.1 详细描述

文件名称: [lks32mc08x_hall.h](#)
 文件标识: 无
 内容摘要: HALL驱动头文件
 其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2016年04月19日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_hall.h](#) 中定义.

3.19.2 宏定义说明

3.19.2.1 HALL_CAPTURE_EVENT

```
#define HALL_CAPTURE_EVENT ((uint32_t)0x00010000)
```

HALL中断使能和标志定义, HALL信号变化中断

在文件 [lks32mc08x_hall.h](#) 第 59 行定义.

3.19.2.2 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 [lks32mc08x_hall.h](#) 第 54 行定义.

3.19.2.3 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 [lks32mc08x_hall.h](#) 第 55 行定义.

3.19.2.4 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 [lks32mc08x_hall.h](#) 第 56 行定义.

3.19.2.5 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 [lks32mc08x_hall.h](#) 第 57 行定义.

3.19.2.6 HALL_OVERFLOW_EVENT

```
#define HALL_OVERFLOW_EVENT ((uint32_t)0x00020000)
```

HALL中断使能和标志定义，HALL计数溢出中断

在文件 [lks32mc08x_hall.h](#) 第 60 行定义.

3.19.3 函数说明

3.19.3.1 HALL_ClearIRQFlag()

```
void HALL_ClearIRQFlag (
    uint32_t tempFlag )
```

函数名称: `uint32_t HALL_ClearIRQFlag(uint32_t tempFlag)`

功能描述: 清除HALL中断标志位

参见
被引用内容: 无

参数

输入参数:	tempFlag:	
	HALL_CAPTURE_EVENT	HALL信号变化中断
	HALL_OVERFLOW_EVENT	HALL计数溢出中断
输出参数:	无	

返回
返回值: 无

注解
其它说明: 无

警告
无

示例代码:

```
if (HALL_GetIRQFlag (HALL_CAPTURE_EVENT)) //读取HALL信号变化中断标志位
{
    HALL_ClearIRQFlag (HALL_CAPTURE_EVENT); //清除HALL信号变化中断标志位
}
```


Date	Version	Author	Description
2022年5月20日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_hall.c](#) 第 242 行定义.

3.19.3.2 HALL_GetCaptureValue()

```
uint32_t HALL_GetCaptureValue (
    void )
```

函数名称: uint32_t [HALL_GetCaptureValue\(void\)](#)

功能描述: 取得HALL值, 未滤波结果, 即采样的原始HALL状态值

参见

被引用内容: 无

参数

输入参数:	无
输出参数:	无

返回

返回 值: 未滤波的HALL信号结果值, 第三位有效, 其它位恒为0

注解

其它说明: 无

警告

无

示例代码:

```
uint32_t HALL_Value = 0;
HALL_Value = HALL\_GetCaptureValue\(\); //读取未滤波的HALL结果值
```

Date	Version	Author	Description
2015年11月26日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_hall.c](#) 第 131 行定义.

3.19.3.3 HALL_GetCount()

```
uint32_t HALL_GetCount (
    void )
```

函数名称: `uint32_t HALL_GetCount(void)`

功能描述: 读取实时HALL计数器值

参见

被引用内容: 无

参数

输入参数:	无
输出参数:	无

返回

返回 值: 实时HALL计数器值

注解

其它说明: 无

警告

无

示例代码:

```
uint32_t HALL_Value = 0;
HALL_Value = HALL\_GetCount\(\); //读取实时HALL计数器值
```

Date	Version	Author	Description
2015年11月26日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_hall.c](#) 第 181 行定义.

3.19.3.4 HALL.GetFilterValue()

```
uint32_t HALL.GetFilterValue (  
    void )
```

函数名称: [uint32_t HALL.GetFilterValue\(void\)](#)

功能描述: 取得HALL值, 滤波结果

参见

被引用内容: 无

参数

输入参数:	无
输出参数:	无

返回

返回 值: 经过滤波的HALL信号结果值, 第三位有效, 其它位恒为0

注解

其它说明: 无

警告

无

示例代码:

```
uint32_t HALL.Value = 0;  
HALL.Value = HALL.GetFilterValue\(\) ; //读取经过滤波的HALL结果值
```

Date	Version	Author	Description
2015年11月26日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_hall.c](#) 第 106 行定义.

3.19.3.5 HALL_GetIRQFlag()

```
uint32_t HALL_GetIRQFlag (  
    uint32_t tempFlag )
```

函数名称: `uint32_t HALL_GetIRQFlag(uint32_t tempFlag)`

功能描述: 读取HALL中断标志

参见

被引用内容: 无

参数

输入参数:	tempFlag:	
	HALL_CAPTURE_EVENT	HALL信号变化中断
	HALL_OVERFLOW_EVENT	HALL计数溢出中断
输出参数:	无	

返回

返回值: HALL中断标志 为1即对应标志位置位, 为0即对应标志位未置位

注解

其它说明: 无

警告

无

示例代码:

```
if (HALL_GetIRQFlag (HALL_CAPTURE_EVENT)) //读取HALL信号变化中断标志位  
{  
}
```

Date	Version	Author	Description
2022年5月20日	1.0	HuangMG	创建

修改日志:

在文件 `lks32mc08x_hall.c` 第 211 行定义.

3.19.3.6 HALL_Init()

```
void HALL_Init (
    HALL_InitTypeDef * HALL_InitStruct )
```

函数名称: void HALL_Init(HALL_InitTypeDef* HALL_InitStruct)

功能描述: HALL初始化函数

参见

被引用内容: SYS_ModuleClockCmd()

参数

输入参数:	HALL_InitTypeDef HALL_InitStruct
输出参数:	无

返回

返回 值: 无

注解

其它说明: 无

警告

无

示例代码:

```
HALL_InitTypeDef HALL_InitStruct;
HALL_StructInit(&HALL_InitStruct);           // 初始化结构体
HALL_InitStruct.HALL_Ena = ENABLE;           // 模块使能
HALL_InitStruct.ClockDivision = HALL_CLK_DIV1; // 设置Hall模块时钟分频系数
HALL_InitStruct.CountTH = 9600000;           // Hall模块计数模值, 计数超过模值会产生超时中断
HALL_InitStruct.FilterLen = 512;             // Hall信号数字滤波长度 512个时钟周期
HALL_InitStruct.Filter75_Ena = ENABLE;       // Hall信号滤波方式, 7判5模式或者全1有效模式
HALL_InitStruct.Capture_IRQ_Ena = ENABLE;    // 捕捉中断使能
HALL_InitStruct.Overflow_IRQ_Ena = ENABLE;    // 超时中断使能
HALL_InitStruct.softIE = DISABLE;            // 软件中断使能
HALL_Init (&HALL_InitStruct);
```

Date	Version	Author	Description
2015年11月26日	1.0	cfwu	创建

修改日志:

在文件 lks32mc08x_hall.c 第 45 行定义.

函数调用图:



3.19.3.7 HALL_StructInit()

```
void HALL_StructInit (
    HALL_InitTypeDef * HALL_InitStruct )
```

函数名称: void HALL_StructInit(HALL_InitTypeDef* HALL_InitStruct)

功能描述: HALL结构体初始化

参见
被引用内容: 无

参数

输入参数:	HALL_InitTypeDef HALL_InitStruct
输出参数:	无

返回
返回值: 无

注解
其它说明: 无

警告
无

示例代码:

```
HALL_InitTypeDef HALL_InitStruct;
HALL_StructInit(&HALL_InitStruct); //初始化结构体
```

Date	Version	Author	Description
2015年11月26日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_hall.c](#) 第 74 行定义.

3.19.3.8 HALL_WIDCount()

```
uint32_t HALL_WIDCount (  
    void )
```

函数名称: `uint32_t HALL_WIDCount(void)`

功能描述: 读取HALL信号变化事件时刻HALL计数值

参见

被引用内容: 无

参数

输入参数:	无
输出参数:	无

返回

返回值: 获取三颗HALL信号任何一个信号发生变化事件时刻的HALL计数器的计数值

注解

其它说明: 无

警告

无

示例代码:

```
uint32_t HALL_Value = 0;  
HALL_Value = HALL\_WIDCount\(\); //读取HALL信号变化事件时HALL计数值
```

Date	Version	Author	Description
2015年11月26日	1.0	cfwu	创建

修改日志:

在文件 `lks32mc08x_hall.c` 第 156 行定义.

3.20 lks32mc08x_hall.h

[浏览该文件的文档.](#)

```

00001
00014 #ifndef __LKS32MC08x_HALL_H
00015 #define __LKS32MC08x_HALL_H
00016
00017 /* Includes -----*/
00018 #include "lks32mc08x.lib.h"
00019
00023 typedef struct
00024 {
00025     __IO uint32_t CFG;
00026     __IO uint32_t INFO;
00027     __IO uint32_t WIDTH;
00028     __IO uint32_t TH;
00029     __IO uint32_t CNT;
00030 } HALL_TypeDef;
00031
00035 #ifndef HALL
00036 #define HALL ((HALL_TypeDef *)HALL_BASE)
00037 #endif
00038
00042 typedef struct
00043 {
00044     uint16_t FilterLen;
00045     uint8_t ClockDivision;
00046     uint8_t Filter75_Ena;
00047     uint8_t HALL_Ena;
00048     uint8_t Capture_IRQ_Ena;
00049     uint8_t Overflow_IRQ_Ena;
00050     uint32_t CountTH;
00051     uint8_t softIE;
00052 } HALL_InitTypeDef;
00053
00054 #define HALL_CLK_DIV1      ((uint32_t)0x00)
00055 #define HALL_CLK_DIV2      ((uint32_t)0x01)
00056 #define HALL_CLK_DIV4      ((uint32_t)0x02)
00057 #define HALL_CLK_DIV8      ((uint32_t)0x03)
00059 #define HALL_CAPTURE_EVENT ((uint32_t)0x00010000)
00060 #define HALL_OVERFLOW_EVENT ((uint32_t)0x00020000)
00062 void HALL_Init(HALL_InitTypeDef *HALL_InitStruct);
00063 void HALL_StructInit(HALL_InitTypeDef *HALL_InitStruct);
00064
00065 uint32_t HALL_GetFilterValue(void);
00066 uint32_t HALL_GetCaptureValue(void);
00067 uint32_t HALL_WIDCount(void);
00068 uint32_t HALL_GetCount(void);
00069
00070 uint32_t HALL_GetIRQFlag(uint32_t tempFlag);
00071 void HALL_ClearIRQFlag(uint32_t tempFlag);
00072
00073 #endif /*__lks32mc08x_HALL_H */
00074
00075 /***** (C) COPYRIGHT LINKO SEMICONDUCTOR *****END OF FILE*****/

```

3.21 include/lks32mc08x_i2c.h 文件参考

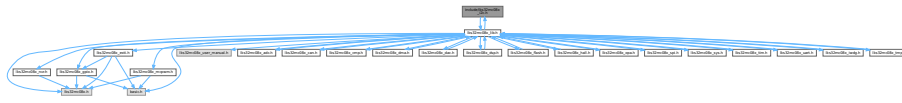
文件名称: `lks32mc08x_i2c.h`

文件标识: 无

内容摘要: I2C驱动头文件

其它说明: 无

#include "lks32mc08x_lib.h"
lks32mc08x_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_TypeDef**
I2C寄存器结构体句柄
- struct **I2C_InitTypeDef**
I2C结构体基地址定义

函数

- void **I2C_Init** (I2C_TypeDef *I2Cx, I2C_InitTypeDef *I2C_InitStruct)
函数名称: void I2C_Init(I2C_TypeDef* I2Cx, I2C_InitTypeDef* I2C_InitStruct)
- void **I2C_StructInit** (I2C_InitTypeDef *I2C_InitStruct)
函数名称: void I2C_StructInit(I2C_InitTypeDef* I2C_InitStruct)
- void **I2C_SendData** (I2C_TypeDef *I2Cx, uint8_t n)
函数名称: void I2C_SendData(I2C_TypeDef *I2Cx, uint8_t n)
- uint8_t **I2C_ReadData** (I2C_TypeDef *I2Cx)
函数名称: uint8_t I2C_ReadData(I2C_TypeDef *I2Cx)
- void **i2c_dma_rx** (u8 addr, u8 *data, u8 len)
函数名称: void i2c_dma_rx(u8 addr, u8 *data, u8 len)
- void **i2c_dma_tx** (u8 addr, u8 *data, u8 len)
函数名称: void i2c_dma_tx(u8 addr, u8 *data, u8 len)
- void **i2c_dma_init** (void)
函数名称: void i2c_dma_init(void)
- void **i2c_dma_state_over** (void)
函数名称: void i2c_dma_state_over()

3.21.1 详细描述

文件名称: **lks32mc08x_i2c.h**
文件标识: 无
内容摘要: I2C驱动头文件
其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2022年05月21日	1.0	YangZJ	创建

修改日志:

在文件 [lks32mc08x_i2c.h](#) 中定义.

3.21.2 函数说明

3.21.2.1 i2c_dma_init()

```
void i2c_dma_init (  
    void )
```

函数名称: [void i2c_dma_init\(void\)](#)

功能描述: DMA硬件初始化

参见

被引用内容: 无

参数

输入参数:	无
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
i2c_dma_init(); // I2C的相关DMA模块硬件初始化
```

Date	Version	Author	Description
2022年05月21日	1.0	YangZJ	创建

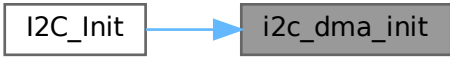
修改日志:

在文件 [lks32mc08x_i2c.c](#) 第 357 行定义.

函数调用图:



这是这个函数的调用关系图:



3.21.2.2 i2c_dma_rx()

```
void i2c_dma_rx (
    u8 addr,
    u8 * data,
    u8 len )
```

函数名称: void [i2c_dma_rx\(u8 addr, u8 *data, u8 len\)](#)

功能描述: I2C数据接收函数

参见

被引用内容: 无

参数

输入参数:	addr: I2C发送地址 data: 发送数据缓冲器 len: 发送数据长度
输出参数:	无

返回

返回 值: 无

注解

其它说明: 无

警告

无

示例代码:

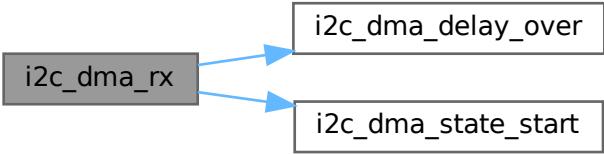
```
u8 i2c_Buff[8] = {0x00};
i2c_dma_rx(0x23 , i2c_Buff , 8); //将接收0x23地址的从机8个字节存储在i2c_Buff数组内
```

Date	Version	Author	Description
2022年05月21日	1.0	YangZJ	创建

修改日志:

在文件 [lks32mc08x_i2c.c](#) 第 320 行定义.

函数调用图:



3.21.2.3 i2c_dma_state_over()

```
void i2c_dma_state_over (
    void )
```

函数名称: `void i2c_dma_state_over()`

功能描述: I2C发送完成函数

参见

被引用内容: 无

参数

输入参数:	无
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
void I2C0_IRQHandler    (void)
{
    I2C->SCR = 0;
    i2c_dma_state_over(); // iic传输完成
}
```

Date	Version	Author	Description
2022年05月21日	1.0	YangZJ	创建

修改日志:

在文件 [lks32mc08x_i2c.c](#) 第 227 行定义.

3.21.2.4 i2c_dma_tx()

```
void i2c_dma_tx (
```

```
u8 addr,  
u8 * data,  
u8 len )
```

函数名称: `void i2c_dma_tx(u8 addr, u8 *data, u8 len)`

功能描述: I2C数据发送函数

参见

被引用内容: 无

参数

输入参数:	addr: I2C发送地址 data: 发送数据缓冲器 len: 发送数据长度
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

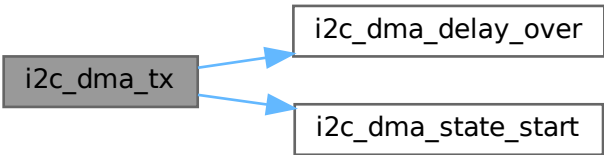
```
u8 i2c_Buff[8] = {0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88};  
i2c_dma_tx(0x23 , i2c_Buff , 8); //发生给0x23地址的从机8个字节i2c_Buff数据
```

Date	Version	Author	Description
2022年05月21日	1.0	YangZJ	创建

修改日志:

在文件 `lks32mc08x_i2c.c` 第 279 行定义.

函数调用图:



3.21.2.5 I2C_Init()

```
void I2C_Init (
    I2C_TypeDef * I2Cx,
    I2C_InitTypeDef * I2C_InitStruct )
```

函数名称: void I2C_Init(I2C_TypeDef* I2Cx, I2C_InitTypeDef* I2C_InitStruct)

功能描述: I2C初始化

参见

被引用内容: [SYS_ModuleClockCmd\(\)](#) [SYS_SoftResetModule\(\)](#);

参数

输入参数:	I2Cx: I2C , I2C_InitTypeDef *I2C_InitStruct
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

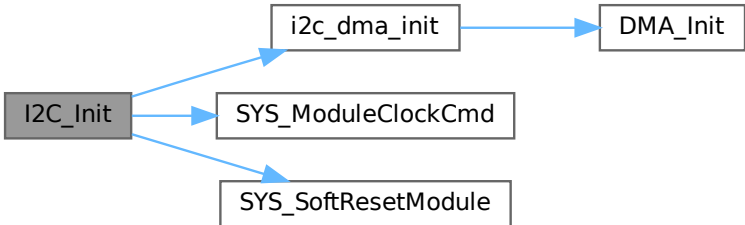
```
I2C_InitTypeDef I2C_InitStructure;  
I2C_StructInit(&I2C_InitStructure);  
I2C_InitStructure.ADRCMP      = DISABLE ; // I2C 硬件地址比较使能开关，只有在 DMA 模式下开启才有效。  
I2C_InitStructure.MST_MODE    = ENABLE   ; // I2C 主模式使能  
I2C_InitStructure.SLV_MODE    = DISABLE  ; // I2C 从模式使能  
I2C_InitStructure.DMA         = ENABLE   ; // I2C DMA传输使能  
I2C_InitStructure.BaudRate    = 100000  ; // I2C 波特率  
I2C_InitStructure.IE          = ENABLE   ; // I2C 中断使能  
I2C_InitStructure.TC_IE       = ENABLE   ; // I2C 数据传输完成中断使能  
I2C_InitStructure.BUS_ERR_IE  = DISABLE  ; // I2C 总线错误事件中断使能  
I2C_InitStructure.STOP_IE     = DISABLE  ; // I2C STOP 事件中断使能  
I2C_InitStructure.BURST_NACK  = ENABLE   ; // I2C 传输, NACK 事件中断使能  
I2C_InitStructure.BURST_ADDR_CMP = DISABLE ; // I2C 传输, 硬件地址匹配中断使能  
I2C_Init(I2C, &I2C_InitStructure);
```

Date	Version	Author	Description
2022年05月21日	1.0	YangZJ	创建

修改日志:

在文件 [lks32mc08x_i2c.c](#) 第 50 行定义.

函数调用图:



3.21.2.6 I2C_ReadData()

```
uint8_t I2C_ReadData (  
    I2C_TypeDef * I2Cx )
```

函数名称: `uint8_t I2C_ReadData(I2C_TypeDef *I2Cx)`

功能描述: I2C读缓冲区数据

参见

被引用内容: 无

参数

输入参数:	I2Cx:I2C
输出参数:	无

返回

返回 值: I2C接收的一字节数据

注解

其它说明: 无

警告

无

示例代码:

```
u8 I2C_Value = 0;
I2C_Value = I2C_ReadData(I2C); //I2C接收的一字节数据
```

Date	Version	Author	Description
2022年05月21日	1.0	YangZJ	创建

修改日志:

在文件 [lks32mc08x_i2c.c](#) 第 158 行定义.

3.21.2.7 I2C_SendData()

```
void I2C_SendData (
    I2C_TypeDef * I2Cx,
    uint8_t n )
```

函数名称: void [I2C_SendData\(I2C_TypeDef *I2Cx, uint8_t n\)](#)

功能描述: I2C发送数据

参见

被引用内容: 无

参数

输入参数:	I2Cx:I2C, n:要发生的一字节数据
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
I2C_SendData(I2C, 0x12); //发生0x12数据
```

Date	Version	Author	Description
2022年05月21日	1.0	YangZJ	创建

修改日志:

在文件 [lks32mc08x_i2c.c](#) 第 133 行定义.

3.21.2.8 I2C_StructInit()

```
void I2C_StructInit (  
    I2C_InitTypeDef * I2C_InitStruct )
```

函数名称: void [I2C_StructInit\(I2C_InitTypeDef* I2C_InitStruct\)](#)

功能描述: I2C结构体初始化

参见

被引用内容: 无

参数

输入参数:	I2C_InitTypeDef *I2C_InitStruct
输出参数:	无



返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
I2C_InitTypeDef I2C_InitStruct;
I2C_StructInit(&I2C_InitStruct); //初始化结构体
```

Date	Version	Author	Description
2022年05月21日	1.0	YangZJ	创建

修改日志:

在文件 [lks32mc08x_i2c.c](#) 第 99 行定义.

3.22 lks32mc08x_i2c.h

[浏览该文件的文档.](#)

```
00001
00015 #ifndef __LKS32MC08x_I2C_H
00016 #define __LKS32MC08x_I2C_H
00017
00018 #include "lks32mc08x.lib.h"
00019
00023 typedef struct
00024 {
00025     __IO uint32_t ADDR;
00026     __IO uint32_t CFG;
00027     __IO uint32_t SCR;
00028     __IO uint32_t DATA;
00029     __IO uint32_t MSCR;
00030     __IO uint32_t BCR;
00031 } I2C_TypeDef;
00032
00036 #ifndef I2C
00037 #define I2C ((I2C_TypeDef *)I2C0_BASE)
00038 #endif
00039
00043 typedef struct
00044 {
00045     u8  ADRCMP;
00046     u8  MST_MODE;
00047     u8  SLV_MODE;
00048     u8  DMA;
00049     u32 BaudRate;
00050     u8  IE;
00051     u8  TC_IE;
00052     u8  BUS_ERR_IE;
00053     u8  STOP_IE;
```

```

00054     u8 BURST_NACK;
00055     u8 BURST_ADDR_CMP;
00056 } I2C_InitTypeDef;
00057
00058 void I2C_Init(I2C_TypeDef *I2Cx, I2C_InitTypeDef *I2C_InitStruct);
00059 void I2C_StructInit(I2C_InitTypeDef *I2C_InitStruct);
00060
00061 void I2C_SendData(I2C_TypeDef *I2Cx, uint8_t n);
00062 uint8_t I2C_ReadData(I2C_TypeDef *I2Cx);
00063
00064 // i2c&dma发送&接收
00065 void i2c_dma_rx(u8 addr, u8 *data, u8 len);
00066 void i2c_dma_tx(u8 addr, u8 *data, u8 len);
00067 void i2c_dma_init(void);
00068 void i2c_dma_state_over(void);
00069
00070 #endif /*__LKS32MC08X_I2C_H */
00071
00072 /***** (C) COPYRIGHT LINKO SEMICONDUCTOR *****END OF FILE*****/

```

3.23 include/lks32mc08x_iwdg.h 文件参考

文件名称: lks32mc08x_IWDG.h

文件标识: 无

内容摘要: 看门狗驱动头文件

其它说明: 无

```
#include "lks32mc08x_lib.h"
```

lks32mc08x_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寄存器结构体句柄

宏定义

- #define **WDT_RTH_2S** 3
看门狗超时复位时间1秒
- #define **WDT_RTH_4S** 2
看门狗超时复位时间4秒
- #define **WDT_RTH_8S** 1
看门狗超时复位时间8秒
- #define **WDT_RTH_64S** 0
看门狗超时复位时间64秒

函数

- void **IWDG_Init** (IWDG_InitTypeDef *)
函数名称: void *IWDG_Init(IWDG_InitTypeDef *this)*
- void **IWDG_StrutInit** (IWDG_InitTypeDef *)
函数名称: void *IWDG_StrutInit(IWDG_InitTypeDef *this)*
- void **IWDG_DISABLE** (void)
函数名称: void *IWDG_DISABLE(void)*
- void **IWDG_ENABLE** (void)
函数名称: void *IWDG_ENABLE(void)*
- void **IWDG_Feed** (uint32_t timeOut)
函数名称: void *IWDG_Feed(uint32_t timeOut)*

3.23.1 详细描述

文件名称: lks32mc08x_IWDG.h
文件标识: 无
内容摘要: 看门狗驱动头文件
其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2021年11月09日	1.0	YangZJ	创建

修改日志:

在文件 [lks32mc08x_iwdg.h](#) 中定义.

3.23.2 宏定义说明

3.23.2.1 WDT_RTH_2S

#define WDT_RTH_2S 3

看门狗超时复位时间1秒

在文件 [lks32mc08x_iwdg.h](#) 第 29 行定义.

3.23.2.2 WDT_RTH_4S

#define WDT_RTH_4S 2

看门狗超时复位时间4秒

在文件 [lks32mc08x_iwdg.h](#) 第 30 行定义.

3.23.2.3 WDT_RTH_64S

```
#define WDT_RTH_64S 0
```

看门狗超时复位时间64秒

在文件 [lks32mc08x_iwdg.h](#) 第 32 行定义.

3.23.2.4 WDT_RTH_8S

```
#define WDT_RTH_8S 1
```

看门狗超时复位时间8秒

在文件 [lks32mc08x_iwdg.h](#) 第 31 行定义.

3.23.3 函数说明

3.23.3.1 IWDG_DISABLE()

```
void IWDG_DISABLE (  
    void )
```

函数名称: [void IWDG_DISABLE\(void\)](#)

功能描述: 关闭看门狗模块

参见

被引用内容: 无

参数

输入参数:	无
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
IWDG_DISABLE(); //关闭看门狗模块
```

Date	Version	Author	Description
2021年11月09日	1.0	YangZJ	创建

修改日志:

在文件 [lks32mc08x_iwdg.c](#) 第 104 行定义.

3.23.3.2 IWDG_ENABLE()

```
void IWDG_ENABLE (
    void )
```

函数名称: void [IWDG_ENABLE\(void\)](#)

功能描述: 打开看门狗模块

参见

被引用内容: 无

参数

输入参数:	无
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
IWDG_ENABLE(); //打开看门狗模块
```

Date	Version	Author	Description
2021年11月09日	1.0	YangZJ	创建

修改日志:

在文件 [lks32mc08x_iwdg.c](#) 第 128 行定义.

3.23.3.3 IWDG_Feed()

```
void IWDG_Feed (
    uint32_t timeOut )
```

函数名称: `void IWDG_Feed(uint32_t timeOut)`

功能描述: 看门狗喂狗

参见

被引用内容: 无

参数

输入参数:	timeOut: 看门狗复位时间 WDT_RTH_2S 3s WDT_RTH_4S 2s WDT_RTH_8S 1s WDT_RTH_64S 0s
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
IWDG_Feed(WDT_RTH_2S); //看门狗喂狗, 且设置下一次复位时间为2s
```


Date	Version	Author	Description
2021年11月09日	1.0	YangZJ	创建

修改日志:

在文件 [lks32mc08x_iwdg.c](#) 第 157 行定义.

3.23.3.4 IWDG_Init()

```
void IWDG_Init (
    IWDG_InitTypeDef * this )
```

函数名称: `void IWDG_Init(IWDG_InitTypeDef *this)`

功能描述: 看门狗初始化

参见

被引用内容: 无

参数

输入参数:	IWDG_InitTypeDef *this 看门狗配置结构体
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
IWDG_InitTypeDef IWDG_InitStructure;
IWDG_StructInit (&IWDG_InitStructure); //初始化结构体
IWDG_InitStructure.WDG_EN = ENABLE; //使能看门狗
IWDG_InitStructure.RTH = WDT_RTH_2S; //设置看门狗2s后复位芯片
IWDG_Init (&IWDG_InitStructure);
```

Date	Version	Author	Description
2021年11月09日	1.0	YangZJ	创建

修改日志:

在文件 [lks32mc08x_iwdg.c](#) 第 40 行定义.

3.23.3.5 IWDG_StrutInit()

```
void IWDG_StrutInit (
    IWDG_InitTypeDef * this )
```

函数名称: void [IWDG_StrutInit\(IWDG_InitTypeDef *this\)](#)

功能描述: 看门狗配置结构体初始化

参见

被引用内容: 无

参数

输入参数:	IWDG_InitTypeDef *this 看门狗配置结构体
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
IWDG_InitTypeDef IWDG_InitStruct;
IWDG_StrutInit (&IWDG_InitStruct); //初始化结构体
```

Date	Version	Author	Description
2021年11月09日	1.0	YangZJ	创建

修改日志:

在文件 `lks32mc08x_iwdg.c` 第 76 行定义.

3.24 lks32mc08x_iwdg.h

[浏览该文件的文档.](#)

```
00001
00014 #ifndef __LKS32MC08x_IWDG_H
00015 #define __LKS32MC08x_IWDG_H
00016
00017 /* Includes -----*/
00018 #include "lks32mc08x_lib.h"
00019
00023 typedef struct
00024 {
00025     u32 WDG_EN;
00026     u32 RTH;
00027 } IWDG_InitTypeDef;
00028
00029 #define WDT_RTH_2S 3
00030 #define WDT_RTH_4S 2
00031 #define WDT_RTH_8S 1
00032 #define WDT_RTH_64S 0
00034 void IWDG_Init(IWDG_InitTypeDef *); // 看门狗初始化
00035 void IWDG_StrutInit(IWDG_InitTypeDef *); // 看门狗配置结构体初始化
00036 void IWDG_DISABLE(void); // 关闭看门狗
00037 void IWDG_ENABLE(void); // 使能看门狗
00038 void IWDG_Feed(uint32_t timeOut); // 看门狗喂狗
00039 #endif // __lks32mc08x_IWDG_H */
00040 /***** (C) COPYRIGHT LINKO SEMICONDUCTOR *****END OF FILE*****/
```

3.25 include/lks32mc08x_lib.h 文件参考

文件名称: LKS32MC08x_lib.h

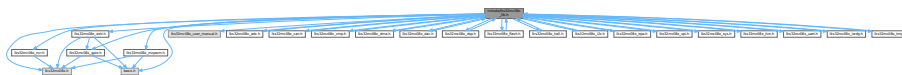
文件标识: 无

内容摘要: LIB库函数头文件

其它说明: 无

```
#include "lks32mc08x.h"
#include "basic.h"
#include "lks32mc08x_user_manual.h"
#include "lks32mc08x_adc.h"
#include "lks32mc08x_can.h"
#include "lks32mc08x_cmp.h"
#include "lks32mc08x_dma.h"
#include "lks32mc08x_dac.h"
#include "lks32mc08x_dsp.h"
#include "lks32mc08x_exti.h"
#include "lks32mc08x_flash.h"
#include "lks32mc08x_gpio.h"
#include "lks32mc08x_hall.h"
#include "lks32mc08x_i2c.h"
#include "lks32mc08x_mcpwm.h"
#include "lks32mc08x_nvr.h"
#include "lks32mc08x_opa.h"
#include "lks32mc08x_spi.h"
#include "lks32mc08x_sys.h"
```

```
#include "lks32mc08x-tim.h"
#include "lks32mc08x-uart.h"
#include "lks32mc08x-iwdg.h"
#include "lks32mc08x-tmp.h"
lks32mc08x_lib.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:



3.25.1 详细描述

文件名称: LKS32MC08x_lib.h
文件标识: 无
内容摘要: LIB库函数头文件
其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2021年11月11日	1.0	YangZJ	创建

修改日志:

在文件 `lks32mc08x_lib.h` 中定义.

3.26 lks32mc08x_lib.h

[浏览该文件的文档.](#)

```
00001
00015 #ifndef __LKS32MC08X_LIB_H
00016 #define __LKS32MC08X_LIB_H
00017
00018 #include "lks32mc08x.h"
00019 #include "basic.h"
00020
00021 #include "lks32mc08x_user_manual.h"
00022 #include "lks32mc08x_adc.h"
00023 #include "lks32mc08x_can.h"
00024 #include "lks32mc08x_cmp.h"
```

```

00025 #include "lks32mc08x_dma.h"
00026 #include "lks32mc08x_dac.h"
00027 #include "lks32mc08x_dsp.h"
00028 #include "lks32mc08x_exti.h"
00029 #include "lks32mc08x_flash.h"
00030 #include "lks32mc08x_gpio.h"
00031 #include "lks32mc08x_hall.h"
00032 #include "lks32mc08x_i2c.h"
00033 #include "lks32mc08x_mcpwm.h"
00034 #include "lks32mc08x_nvr.h"
00035 #include "lks32mc08x_opa.h"
00036 #include "lks32mc08x_spi.h"
00037 #include "lks32mc08x_sys.h"
00038 #include "lks32mc08x_tim.h"
00039 #include "lks32mc08x_uart.h"
00040 #include "lks32mc08x_spi.h"
00041 #include "lks32mc08x_iwdg.h"
00042 #include "lks32mc08x_tmp.h"
00043
00044 #endif /*_LKS32MC08X_LIB.H */
00045
00046 /***** (C) COPYRIGHT LINKO SEMICONDUCTOR *****/

```

3.27 include/lks32mc08x_mcpwm.h 文件参考

文件名称: [lks32mc08x_mcpwm.h](#)

文件标识: 无

内容摘要: mcpwm驱动头文件

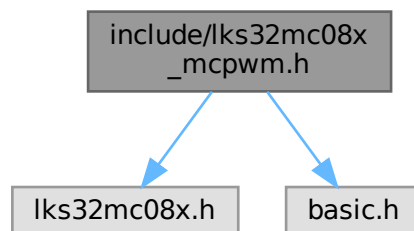
其它说明: 无

```

#include "lks32mc08x.h"
#include "basic.h"

```

lks32mc08x_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](#)

*MCPWM*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 MCPWM_MOE_ENABLE_MASK ((uint16_t)0x0040)`
打开MOE位
- `#define MCPWM_MOE_DISABLE_MASK ((uint16_t)~0x0040)`
关MOE位
- `#define CENTRAL_PWM_MODE 0`
中心对齐PWM模式
- `#define EDGE_PWM_MODE 1`
边沿对齐PWM模式
- `#define HIGH_LEVEL 1`
高电平
- `#define LOW_LEVEL 0`
低电平
- `#define HIGH_LEVEL_ACTIVE 0`
高电平有效
- `#define LOW_LEVEL_ACTIVE 1`
低电平有效
- `#define FAIL_SEL_CMP 1`
Fail事件来源比较器
- `#define FAIL_SEL_IO 0`
Fail事件来源比较器
- `#define MCPWM_UPDATE_REG()`
手动更新占空比寄存器值到实际运行寄存器中
- `#define MCPWM_T0_IRQ_EN BIT0`
计数器计数值回到MCPWM.TH中断使能
- `#define MCPWM_T1_IRQ_EN BIT1`
计数器计数值回到0中断使能
- `#define MCPWM_TH00_IRQ_EN BIT2`
计数器计数值回到MCPWM.TH00中断使能
- `#define MCPWM_TH01_IRQ_EN BIT3`
计数器计数值回到MCPWM.TH01中断使能
- `#define MCPWM_TH10_IRQ_EN BIT4`
计数器计数值回到MCPWM.TH10中断使能
- `#define MCPWM_TH11_IRQ_EN BIT5`
计数器计数值回到MCPWM.TH11中断使能
- `#define MCPWM_TH20_IRQ_EN BIT6`
计数器计数值回到MCPWM.TH20中断使能
- `#define MCPWM_TH21_IRQ_EN BIT7`
计数器计数值回到MCPWM.TH21中断使能
- `#define MCPWM_TH30_IRQ_EN BIT8`
计数器计数值回到MCPWM.TH30中断使能
- `#define MCPWM_TH31_IRQ_EN BIT9`
计数器计数值回到MCPWM.TH31中断使能
- `#define MCPWM_TMR0_IRQ_EN BIT10`
计数器计数值回到MCPWM.TMR0中断使能
- `#define MCPWM_TMR1_IRQ_EN BIT11`
计数器计数值回到MCPWM.TMR1中断使能
- `#define MCPWM_TMR2_IRQ_EN BIT12`
计数器计数值回到MCPWM.TMR0中断使能
- `#define MCPWM_TMR3_IRQ_EN BIT13`

- 计数器计数值回到MCPWM.TMR1中断使能
- #define MCPWM_SHADE_IRQ_EN BIT14
MCPWM.TH/MCPWM.TH00-MCPWM.TH31/\ MCPWM.TR0-MCPWM.TR3等寄存器更新到MCPWM实际运行系统中断使能
- #define MCPWM_CH0_SHORT_IRQ_EN BIT0
MCPWMCH0.P和CH0N同时有效中断使能
- #define MCPWM_CH1_SHORT_IRQ_EN BIT1
MCPWMCH1.P和CH1N同时有效中断使能
- #define MCPWM_CH2_SHORT_IRQ_EN BIT2
MCPWMCH2.P和CH2N同时有效中断使能
- #define MCPWM_CH3_SHORT_IRQ_EN BIT3
MCPWMCH3.P和CH3N同时有效中断使能
- #define MCPWM_FAIL0_IRQ_EN BIT4
MCPWM.FAIL0中断使能
- #define MCPWM_FAIL1_IRQ_EN BIT5
MCPWM.FAIL1中断使能
- #define MCPWM_T0_IRQ_IF BIT0
计数器计数值回到MCPWM.TH中断标志位
- #define MCPWM_T1_IRQ_IF BIT1
计数器计数值回到0中断标志位
- #define MCPWM_TH00_IRQ_IF BIT2
计数器计数值回到MCPWM.TH00中断标志位
- #define MCPWM_TH01_IRQ_IF BIT3
计数器计数值回到MCPWM.TH01中断标志位
- #define MCPWM_TH10_IRQ_IF BIT4
计数器计数值回到MCPWM.TH10中断标志位
- #define MCPWM_TH11_IRQ_IF BIT5
计数器计数值回到MCPWM.TH11中断标志位
- #define MCPWM_TH20_IRQ_IF BIT6
计数器计数值回到MCPWM.TH20中断标志位
- #define MCPWM_TH21_IRQ_IF BIT7
计数器计数值回到MCPWM.TH21中断标志位
- #define MCPWM_TH30_IRQ_IF BIT8
计数器计数值回到MCPWM.TH30中断标志位
- #define MCPWM_TH31_IRQ_IF BIT9
计数器计数值回到MCPWM.TH31中断标志位
- #define MCPWM_TMR0_IRQ_IF BIT10
计数器计数值回到MCPWM.TMR0中断标志位
- #define MCPWM_TMR1_IRQ_IF BIT11
计数器计数值回到MCPWM.TMR1中断标志位
- #define MCPWM_TMR2_IRQ_IF BIT12
计数器计数值回到MCPWM.TMR0中断标志位
- #define MCPWM_TMR3_IRQ_IF BIT13
计数器计数值回到MCPWM.TMR1中断标志位
- #define MCPWM_SHADE_IRQ_IF BIT14
MCPWM.TH/MCPWM.TH00-MCPWM.TH31/\ MCPWM.TR0-MCPWM.TR3等寄存器更新到MCPWM实际运行系统中断标志位
- #define MCPWM_CH0_SHORT_IRQ_IF BIT15
MCPWMCH0.P和CH0N同时有效中断标志位
- #define MCPWM_CH1_SHORT_IRQ_IF BIT16
MCPWMCH1.P和CH1N同时有效中断标志位

- **#define MCPWM_CH2_SHORT_IRQ_IF** BIT17
MCPWMCH2.P和CH2N同时有效中断标志位
- **#define MCPWM_CH3_SHORT_IRQ_IF** BIT18
MCPWMCH3.P和CH3N同时有效中断标志位
- **#define MCPWM_FAIL0_IRQ_IF** BIT19
MCPWM.FAIL0中断标志位
- **#define MCPWM_FAIL1_IRQ_IF** BIT20
MCPWM.FAIL1中断标志位
- **#define MCPWM_IF_SHADE** BIT14
MCPWM.TH/MCPWM.TH00-MCPWM.TH31/MCPWM.TMR0-MCPWM.TMR3等寄存器更新到MCPWM实际运行系统中中断标志位。
- **#define MCPWM_IF_TMR3** BIT13
MCPWM实际运行系统中计数器的计数值等于MCPWM.TMR3中断源事件。1，发生；0，没发生。写1清零。
- **#define MCPWM_IF_TMR2** BIT12
MCPWM实际运行系统中计数器的计数值等于MCPWM.TMR2中断源事件。1，发生；0，没发生。写1清零。
- **#define MCPWM_IF_TMR1** BIT11
MCPWM实际运行系统中计数器的计数值等于MCPWM.TMR1中断源事件。1，发生；0，没发生。写1清零。
- **#define MCPWM_IF_TMR0** BIT10
MCPWM实际运行系统中计数器的计数值等于MCPWM.TMR0中断源事件。1，发生；0，没发生。写1清零。
- **#define MCPWM_IF_TH31** BIT9
MCPWM实际运行系统中计数器的计数值等于MCPWM.TH31中断源事件。1，发生；0，没发生。写1清零。
- **#define MCPWM_IF_TH30** BIT8
MCPWM实际运行系统中计数器的计数值等于MCPWM.TH30中断源事件。1，发生；0，没发生。写1清零。
- **#define MCPWM_IF_TH21** BIT7
MCPWM实际运行系统中计数器的计数值等于MCPWM.TH21中断源事件。1，发生；0，没发生。写1清零。
- **#define MCPWM_IF_TH20** BIT6
MCPWM实际运行系统中计数器的计数值等于MCPWM.TH20中断源事件。1，发生；0，没发生。写1清零。
- **#define MCPWM_IF_TH11** BIT5
MCPWM实际运行系统中计数器的计数值等于MCPWM.TH11中断源事件。1，发生；0，没发生。写1清零。
- **#define MCPWM_IF_TH10** BIT4
MCPWM实际运行系统中计数器的计数值等于MCPWM.TH10中断源事件。1，发生；0，没发生。写1清零。
- **#define MCPWM_IF_TH01** BIT3
MCPWM实际运行系统中计数器的计数值等于MCPWM.TH01中断源事件。1，发生；0，没发生。写1清零。
- **#define MCPWM_IF_TH00** BIT2
MCPWM实际运行系统中计数器的计数值等于MCPWM.TH00中断源事件。1，发生；0，没发生。写1清零。
- **#define MCPWM_IF_T1** BIT1
t1事件，计数器的计数值到达0中断源事件。1，发生；0，没发生。写1清零。
- **#define MCPWM_IF_T0** BIT0
t0事件，计数器的计数值回到MCPWM.TH中断源事件。1，发生；0，没发生。写1清零。
- **#define MCPWM{EIF_FAIL1}** BIT5
FAIL1中断源使能。1，使能；0，关闭。
- **#define MCPWM{EIF_FAIL0}** BIT4
FAIL0中断源使能。1，使能；0，关闭。
- **#define MCPWM{EIF_CH3_SHORT}** BIT3
MCPWM CH3.P 和 CH3.N 同时有效，中断源使能。1，使能；0，关闭。
- **#define MCPWM{EIF_CH2_SHORT}** BIT2
MCPWM CH2.P 和 CH2.N 同时有效，中断源使能。1，使能；0，关闭。
- **#define MCPWM{EIF_CH1_SHORT}** BIT1

- MCPWM CH1_P 和 CH1_N 同时有效，中断源使能。1，使能；0，关闭。

• #define MCPWM_EIF_CH0_SHORT BIT0

MCPWM CH0_P 和 CH0_N 同时有效，中断源使能。1，使能；0，关闭。

函数

- void PWMOutputs (FuncState t.state)

函数名称: void PWMOutputs(FuncState t.state)
- void MCPWM_Init (MCPWM_InitTypeDef *MCPWM_InitStruct)

函数名称: void MCPWM_Init(MCPWM_InitTypeDef *MCPWM_InitStruct)
- void MCPWM_StructInit (MCPWM_InitTypeDef *MCPWM_InitStruct)

函数名称: void MCPWM_StructInit(MCPWM_InitTypeDef* MCPWM_InitStruct)
- u16 MCPWM_GetIRQFlag (uint32_t INT_flag)

函数名称: u16 MCPWM_GetIRQFlag(uint32_t INT_flag)
- void MCPWM_ClearIRQFlag (uint32_t INT_flag)

函数名称: void MCPWM_ClearIRQFlag(uint32_t INT_flag)

3.27.1 详细描述

文件名称: lks32mc08x_mcpwm.h

文件标识: 无

内容摘要: mcpwm驱动头文件

其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2019年05月21日	1.0	Howlet	创建

修改日志:

在文件 lks32mc08x_mcpwm.h 中定义.

3.27.2 宏定义说明

3.27.2.1 CENTRAL_PWM_MODE

#define CENTRAL_PWM_MODE 0

中心对齐PWM模式

在文件 lks32mc08x_mcpwm.h 第 184 行定义.

3.27.2.2 EDGE_PWM_MODE

```
#define EDGE_PWM_MODE 1
```

边沿对齐PWM模式

在文件 [lks32mc08x_mcpwm.h](#) 第 185 行定义.

3.27.2.3 FAIL_SEL_CMP

```
#define FAIL_SEL_CMP 1
```

Fail事件来源比较器

在文件 [lks32mc08x_mcpwm.h](#) 第 193 行定义.

3.27.2.4 FAIL_SEL_IO

```
#define FAIL_SEL_IO 0
```

Fail事件来源比较器

在文件 [lks32mc08x_mcpwm.h](#) 第 194 行定义.

3.27.2.5 HIGH_LEVEL

```
#define HIGH_LEVEL 1
```

高电平

在文件 [lks32mc08x_mcpwm.h](#) 第 187 行定义.

3.27.2.6 HIGH_LEVEL_ACTIVE

```
#define HIGH_LEVEL_ACTIVE 0
```

高电平有效

在文件 [lks32mc08x_mcpwm.h](#) 第 190 行定义.

3.27.2.7 LOW_LEVEL

```
#define LOW_LEVEL 0
```

低电平

在文件 [lks32mc08x_mcpwm.h](#) 第 188 行定义.

3.27.2.8 LOW_LEVEL_ACTIVE

```
#define LOW_LEVEL_ACTIVE 1
```

低电平有效

在文件 [lks32mc08x_mcpwm.h](#) 第 191 行定义.

3.27.2.9 MCPWM_CH0_SHORT_IRQ_EN

```
#define MCPWM_CH0_SHORT_IRQ_EN BIT0
```

MCPWMCH0_P和CH0N同时有效中断使能

在文件 [lks32mc08x_mcpwm.h](#) 第 221 行定义.

3.27.2.10 MCPWM_CH0_SHORT_IRQ_IF

```
#define MCPWM_CH0_SHORT_IRQ_IF BIT15
```

MCPWMCH0_P和CH0N同时有效中断标志位

在文件 [lks32mc08x_mcpwm.h](#) 第 245 行定义.

3.27.2.11 MCPWM_CH1_SHORT_IRQ_EN

```
#define MCPWM_CH1_SHORT_IRQ_EN BIT1
```

MCPWMCH1_P和CH1N同时有效中断使能

在文件 [lks32mc08x_mcpwm.h](#) 第 222 行定义.

3.27.2.12 MCPWM_CH1_SHORT_IRQ_IF

```
#define MCPWM_CH1_SHORT_IRQ_IF BIT16
```

MCPWMCH1_P和CH1N同时有效中断标志位

在文件 [lks32mc08x_mcpwm.h](#) 第 246 行定义.

3.27.2.13 MCPWM_CH2_SHORT_IRQ_EN

```
#define MCPWM_CH2_SHORT_IRQ_EN BIT2
```

MCPWMCH2_P和CH2N同时有效中断使能

在文件 [lks32mc08x_mcpwm.h](#) 第 223 行定义.

3.27.2.14 MCPWM_CH2_SHORT_IRQ_IF

```
#define MCPWM_CH2_SHORT_IRQ_IF BIT17
```

MCPWMCH2_P和CH2N同时有效中断标志位

在文件 [lks32mc08x_mcpwm.h](#) 第 247 行定义.

3.27.2.15 MCPWM_CH3_SHORT_IRQ_EN

```
#define MCPWM_CH3_SHORT_IRQ_EN BIT3
```

MCPWMCH3_P和CH3N同时有效中断使能

在文件 [lks32mc08x_mcpwm.h](#) 第 224 行定义.

3.27.2.16 MCPWM_CH3_SHORT_IRQ_IF

```
#define MCPWM_CH3_SHORT_IRQ_IF BIT18
```

MCPWMCH3_P和CH3N同时有效中断标志位

在文件 [lks32mc08x_mcpwm.h](#) 第 248 行定义.

3.27.2.17 MCPWM{EIF_CH0_SHORT

```
#define MCPWM{EIF_CH0_SHORT BIT0
```

MCPWM CH0_P 和 CH0_N 同时有效，中断源使能。1，使能；0，关闭。

在文件 [lks32mc08x_mcpwm.h](#) 第 273 行定义.

3.27.2.18 MCPWM{EIF_CH1_SHORT

```
#define MCPWM{EIF_CH1_SHORT BIT1
```

MCPWM CH1_P 和 CH1_N 同时有效，中断源使能。1，使能；0，关闭。

在文件 [lks32mc08x_mcpwm.h](#) 第 272 行定义.

3.27.2.19 MCPWM{EIF_CH2_SHORT

```
#define MCPWM{EIF_CH2_SHORT BIT2
```

MCPWM CH2_P 和 CH2_N 同时有效，中断源使能。1，使能；0，关闭。

在文件 [lks32mc08x_mcpwm.h](#) 第 271 行定义.

3.27.2.20 MCPWM_EIF_CH3_SHORT

```
#define MCPWM_EIF_CH3_SHORT BIT3
```

MCPWM CH3_P 和 CH3_N 同时有效，中断源使能。1，使能；0，关闭。

在文件 [lks32mc08x_mcpwm.h](#) 第 270 行定义。

3.27.2.21 MCPWM_EIF_FAIL0

```
#define MCPWM_EIF_FAIL0 BIT4
```

FAIL0 中断源使能。1，使能；0，关闭。

在文件 [lks32mc08x_mcpwm.h](#) 第 269 行定义。

3.27.2.22 MCPWM_EIF_FAIL1

```
#define MCPWM_EIF_FAIL1 BIT5
```

FAIL1 中断源使能。1，使能；0，关闭。

在文件 [lks32mc08x_mcpwm.h](#) 第 268 行定义。

3.27.2.23 MCPWM_FAIL0_IRQ_EN

```
#define MCPWM_FAIL0_IRQ_EN BIT4
```

MCPWM_FAIL0中断使能

在文件 [lks32mc08x_mcpwm.h](#) 第 225 行定义。

3.27.2.24 MCPWM_FAIL0_IRQ_IF

```
#define MCPWM_FAIL0_IRQ_IF BIT19
```

MCPWM_FAIL0中断标志位

在文件 [lks32mc08x_mcpwm.h](#) 第 249 行定义。

3.27.2.25 MCPWM_FAIL1_IRQ_EN

```
#define MCPWM_FAIL1_IRQ_EN BIT5
```

MCPWM_FAIL1中断使能

在文件 [lks32mc08x_mcpwm.h](#) 第 226 行定义。

3.27.2.26 MCPWM_FAIL1_IRQ_IF

```
#define MCPWM_FAIL1_IRQ_IF BIT20
```

MCPWM_FAIL1中断标志位

在文件 [lks32mc08x_mcpwm.h](#) 第 250 行定义。

3.27.2.27 MCPWM_IF_SHADE

```
#define MCPWM_IF_SHADE BIT14
```

MCPWM_TH/MCPWM_TH00-MCPWM_TH31/MCPWM_TMR0-MCPWM_TMR3等寄存器更新到MCPWM实际运行系统中中断标志位。

在文件 [lks32mc08x_mcpwm.h](#) 第 253 行定义。

3.27.2.28 MCPWM_IF_T0

```
#define MCPWM_IF_T0 BIT0
```

t0事件，计数器的计数值回到MCPWM_TH中断源事件。1，发生；0，没发生。写1清零。

在文件 [lks32mc08x_mcpwm.h](#) 第 267 行定义。

3.27.2.29 MCPWM_IF_T1

```
#define MCPWM_IF_T1 BIT1
```

t1事件，计数器的计数值到达0中断源事件。1，发生；0，没发生。写1清零。

在文件 [lks32mc08x_mcpwm.h](#) 第 266 行定义。

3.27.2.30 MCPWM_IF_TH00

```
#define MCPWM_IF_TH00 BIT2
```

MCPWM实际运行系统中计数器的计数值等于MCPWM_TH00中断源事件。1，发生；0，没发生。写1清零。

在文件 [lks32mc08x_mcpwm.h](#) 第 265 行定义。

3.27.2.31 MCPWM_IF_TH01

```
#define MCPWM_IF_TH01 BIT3
```

MCPWM实际运行系统中计数器的计数值等于MCPWM_TH01中断源事件。1，发生；0，没发生。写1清零。

在文件 [lks32mc08x_mcpwm.h](#) 第 264 行定义。

3.27.2.32 MCPWM_IF_TH10

```
#define MCPWM_IF_TH10 BIT4
```

MCPWM实际运行系统中计数器的计数值等于MCPWM.TH10中断源事件。1，发生；0，没发生。写1清零。

在文件 [lks32mc08x_mcpwm.h](#) 第 263 行定义。

3.27.2.33 MCPWM_IF_TH11

```
#define MCPWM_IF_TH11 BIT5
```

MCPWM实际运行系统中计数器的计数值等于MCPWM.TH11中断源事件。1，发生；0，没发生。写1清零。

在文件 [lks32mc08x_mcpwm.h](#) 第 262 行定义。

3.27.2.34 MCPWM_IF_TH20

```
#define MCPWM_IF_TH20 BIT6
```

MCPWM实际运行系统中计数器的计数值等于MCPWM.TH20中断源事件。1，发生；0，没发生。写1清零。

在文件 [lks32mc08x_mcpwm.h](#) 第 261 行定义。

3.27.2.35 MCPWM_IF_TH21

```
#define MCPWM_IF_TH21 BIT7
```

MCPWM实际运行系统中计数器的计数值等于MCPWM.TH21中断源事件。1，发生；0，没发生。写1清零。

在文件 [lks32mc08x_mcpwm.h](#) 第 260 行定义。

3.27.2.36 MCPWM_IF_TH30

```
#define MCPWM_IF_TH30 BIT8
```

MCPWM实际运行系统中计数器的计数值等于MCPWM.TH30中断源事件。1，发生；0，没发生。写1清零。

在文件 [lks32mc08x_mcpwm.h](#) 第 259 行定义。

3.27.2.37 MCPWM_IF_TH31

```
#define MCPWM_IF_TH31 BIT9
```

MCPWM实际运行系统中计数器的计数值等于MCPWM.TH31中断源事件。1，发生；0，没发生。写1清零。

在文件 [lks32mc08x_mcpwm.h](#) 第 258 行定义。

3.27.2.38 MCPWM_IF_TMR0

```
#define MCPWM_IF_TMR0 BIT10
```

MCPWM实际运行系统中计数器的计数值等于MCPWM_TMR0中断源事件。1，发生；0，没发生。写1清零。

在文件 [lks32mc08x_mcpwm.h](#) 第 257 行定义。

3.27.2.39 MCPWM_IF_TMR1

```
#define MCPWM_IF_TMR1 BIT11
```

MCPWM实际运行系统中计数器的计数值等于MCPWM_TMR1中断源事件。1，发生；0，没发生。写1清零。

在文件 [lks32mc08x_mcpwm.h](#) 第 256 行定义。

3.27.2.40 MCPWM_IF_TMR2

```
#define MCPWM_IF_TMR2 BIT12
```

MCPWM实际运行系统中计数器的计数值等于MCPWM_TMR2中断源事件。1，发生；0，没发生。写1清零。

在文件 [lks32mc08x_mcpwm.h](#) 第 255 行定义。

3.27.2.41 MCPWM_IF_TMR3

```
#define MCPWM_IF_TMR3 BIT13
```

MCPWM实际运行系统中计数器的计数值等于MCPWM_TMR3中断源事件。1，发生；0，没发生。写1清零。

在文件 [lks32mc08x_mcpwm.h](#) 第 254 行定义。

3.27.2.42 MCPWM_MOE_DISABLE_MASK

```
#define MCPWM_MOE_DISABLE_MASK ((uint16_t)~0x0040)
```

关MOE位

在文件 [lks32mc08x_mcpwm.h](#) 第 182 行定义。

3.27.2.43 MCPWM_MOE_ENABLE_MASK

```
#define MCPWM_MOE_ENABLE_MASK ((uint16_t)0x0040)
```

打开MOE位

在文件 [lks32mc08x_mcpwm.h](#) 第 181 行定义。

3.27.2.44 MCPWM_SHADE_IRQ_EN

```
#define MCPWM_SHADE_IRQ_EN BIT14
```

MCPWM_TH/MCPWM_TH00-MCPWM_TH31/ \ MCPWM_TR0-MCPWM_TR3等寄存器更新到MCPWM实际运行系统中断使能

在文件 [lks32mc08x_mcpwm.h](#) 第 219 行定义.

3.27.2.45 MCPWM_SHADE_IRQ_IF

```
#define MCPWM_SHADE_IRQ_IF BIT14
```

MCPWM_TH/MCPWM_TH00-MCPWM_TH31/ \ MCPWM_TR0-MCPWM_TR3等寄存器更新到MCPWM实际运行系统中断标志位

在文件 [lks32mc08x_mcpwm.h](#) 第 243 行定义.

3.27.2.46 MCPWM_T0_IRQ_EN

```
#define MCPWM_T0_IRQ_EN BIT0
```

计数器计数值回到MCPWM_TH中断使能

在文件 [lks32mc08x_mcpwm.h](#) 第 204 行定义.

3.27.2.47 MCPWM_T0_IRQ_IF

```
#define MCPWM_T0_IRQ_IF BIT0
```

计数器计数值回到MCPWM_TH中断标志位

在文件 [lks32mc08x_mcpwm.h](#) 第 228 行定义.

3.27.2.48 MCPWM_T1_IRQ_EN

```
#define MCPWM_T1_IRQ_EN BIT1
```

计数器计数值回到0中断使能

在文件 [lks32mc08x_mcpwm.h](#) 第 205 行定义.

3.27.2.49 MCPWM_T1_IRQ_IF

```
#define MCPWM_T1_IRQ_IF BIT1
```

计数器计数值回到0中断标志位

在文件 [lks32mc08x_mcpwm.h](#) 第 229 行定义.

3.27.2.50 MCPWM_TH00_IRQ_EN

```
#define MCPWM_TH00_IRQ_EN BIT2
```

计数器计数值回到MCPWM_TH00中断使能

在文件 [lks32mc08x_mcpwm.h](#) 第 206 行定义.

3.27.2.51 MCPWM_TH00_IRQ_IF

```
#define MCPWM_TH00_IRQ_IF BIT2
```

计数器计数值回到MCPWM_TH00中断标志位

在文件 [lks32mc08x_mcpwm.h](#) 第 230 行定义.

3.27.2.52 MCPWM_TH01_IRQ_EN

```
#define MCPWM_TH01_IRQ_EN BIT3
```

计数器计数值回到MCPWM_TH01中断使能

在文件 [lks32mc08x_mcpwm.h](#) 第 207 行定义.

3.27.2.53 MCPWM_TH01_IRQ_IF

```
#define MCPWM_TH01_IRQ_IF BIT3
```

计数器计数值回到MCPWM_TH01中断标志位

在文件 [lks32mc08x_mcpwm.h](#) 第 231 行定义.

3.27.2.54 MCPWM_TH10_IRQ_EN

```
#define MCPWM_TH10_IRQ_EN BIT4
```

计数器计数值回到MCPWM_TH10中断使能

在文件 [lks32mc08x_mcpwm.h](#) 第 208 行定义.

3.27.2.55 MCPWM_TH10_IRQ_IF

```
#define MCPWM_TH10_IRQ_IF BIT4
```

计数器计数值回到MCPWM_TH10中断标志位

在文件 [lks32mc08x_mcpwm.h](#) 第 232 行定义.

3.27.2.56 MCPWM_TH11_IRQ_EN

```
#define MCPWM_TH11_IRQ_EN BIT5
```

计数器计数值回到MCPWM_TH11中断使能

在文件 [lks32mc08x_mcpwm.h](#) 第 209 行定义.

3.27.2.57 MCPWM_TH11_IRQ_IF

```
#define MCPWM_TH11_IRQ_IF BIT5
```

计数器计数值回到MCPWM_TH11中断标志位

在文件 [lks32mc08x_mcpwm.h](#) 第 233 行定义.

3.27.2.58 MCPWM_TH20_IRQ_EN

```
#define MCPWM_TH20_IRQ_EN BIT6
```

计数器计数值回到MCPWM_TH20中断使能

在文件 [lks32mc08x_mcpwm.h](#) 第 210 行定义.

3.27.2.59 MCPWM_TH20_IRQ_IF

```
#define MCPWM_TH20_IRQ_IF BIT6
```

计数器计数值回到MCPWM_TH20中断标志位

在文件 [lks32mc08x_mcpwm.h](#) 第 234 行定义.

3.27.2.60 MCPWM_TH21_IRQ_EN

```
#define MCPWM_TH21_IRQ_EN BIT7
```

计数器计数值回到MCPWM_TH21中断使能

在文件 [lks32mc08x_mcpwm.h](#) 第 211 行定义.

3.27.2.61 MCPWM_TH21_IRQ_IF

```
#define MCPWM_TH21_IRQ_IF BIT7
```

计数器计数值回到MCPWM_TH21中断标志位

在文件 [lks32mc08x_mcpwm.h](#) 第 235 行定义.

3.27.2.62 MCPWM_TH30_IRQ_EN

```
#define MCPWM_TH30_IRQ_EN BIT8
```

计数器计数值回到MCPWM_TH30中断使能

在文件 [lks32mc08x_mcpwm.h](#) 第 212 行定义.

3.27.2.63 MCPWM_TH30_IRQ_IF

```
#define MCPWM_TH30_IRQ_IF BIT8
```

计数器计数值回到MCPWM_TH30中断标志位

在文件 [lks32mc08x_mcpwm.h](#) 第 236 行定义.

3.27.2.64 MCPWM_TH31_IRQ_EN

```
#define MCPWM_TH31_IRQ_EN BIT9
```

计数器计数值回到MCPWM_TH31中断使能

在文件 [lks32mc08x_mcpwm.h](#) 第 213 行定义.

3.27.2.65 MCPWM_TH31_IRQ_IF

```
#define MCPWM_TH31_IRQ_IF BIT9
```

计数器计数值回到MCPWM_TH31中断标志位

在文件 [lks32mc08x_mcpwm.h](#) 第 237 行定义.

3.27.2.66 MCPWM_TMR0_IRQ_EN

```
#define MCPWM_TMR0_IRQ_EN BIT10
```

计数器计数值回到MCPWM_TMR0中断使能

在文件 [lks32mc08x_mcpwm.h](#) 第 214 行定义.

3.27.2.67 MCPWM_TMR0_IRQ_IF

```
#define MCPWM_TMR0_IRQ_IF BIT10
```

计数器计数值回到MCPWM_TMR0中断标志位

在文件 [lks32mc08x_mcpwm.h](#) 第 238 行定义.

3.27.2.68 MCPWM_TMR1_IRQ_EN

```
#define MCPWM_TMR1_IRQ_EN BIT11
```

计数器计数值回到MCPWM_TMR1中断使能

在文件 [lks32mc08x_mcpwm.h](#) 第 215 行定义.

3.27.2.69 MCPWM_TMR1_IRQ_IF

```
#define MCPWM_TMR1_IRQ_IF BIT11
```

计数器计数值回到MCPWM_TMR1中断标志位

在文件 [lks32mc08x_mcpwm.h](#) 第 239 行定义.

3.27.2.70 MCPWM_TMR2_IRQ_EN

```
#define MCPWM_TMR2_IRQ_EN BIT12
```

计数器计数值回到MCPWM_TMR0中断使能

在文件 [lks32mc08x_mcpwm.h](#) 第 216 行定义.

3.27.2.71 MCPWM_TMR2_IRQ_IF

```
#define MCPWM_TMR2_IRQ_IF BIT12
```

计数器计数值回到MCPWM_TMR0中断标志位

在文件 [lks32mc08x_mcpwm.h](#) 第 240 行定义.

3.27.2.72 MCPWM_TMR3_IRQ_EN

```
#define MCPWM_TMR3_IRQ_EN BIT13
```

计数器计数值回到MCPWM_TMR1中断使能

在文件 [lks32mc08x_mcpwm.h](#) 第 217 行定义.

3.27.2.73 MCPWM_TMR3_IRQ_IF

```
#define MCPWM_TMR3_IRQ_IF BIT13
```

计数器计数值回到MCPWM_TMR1中断标志位

在文件 [lks32mc08x_mcpwm.h](#) 第 241 行定义.

3.27.2.74 MCPWM_UPDATE_REG

```
#define MCPWM_UPDATE_REG ( )
```

值:

```
{  
    MCPWM_UPDATE = 0x00ff; \br/>}
```

手动更新占空比寄存器值到实际运行寄存器中

在文件 [lks32mc08x_mcpwm.h](#) 第 202 行定义.

3.27.3 函数说明

3.27.3.1 MCPWM_ClearIRQFlag()

```
void MCPWM_ClearIRQFlag (  
    uint32_t INT_flag )
```

函数名称: void [MCPWM_ClearIRQFlag\(uint32_t INT_flag\)](#)

功能描述: 清除MCPWM中断标志

参见

被引用内容: 无

参数

输入参数:	MCPWM_T0_IRQ_IF	BIT0	计数器计数值回到MCPWM_TH中断标志位
	MCPWM_T1_IRQ_IF	BIT1	计数器计数值回到0中断标志位
	MCPWM_TH00_IRQ_IF	BIT2	计数器计数值回到MCPWM_TH00中断标志位
	MCPWM_TH01_IRQ_IF	BIT3	计数器计数值回到MCPWM_TH01中断标志位
	MCPWM_TH10_IRQ_IF	BIT4	计数器计数值回到MCPWM_TH10中断标志位
	MCPWM_TH11_IRQ_IF	BIT5	计数器计数值回到MCPWM_TH11中断标志位
	MCPWM_TH20_IRQ_IF	BIT6	计数器计数值回到MCPWM_TH20中断标志位
	MCPWM_TH21_IRQ_IF	BIT7	计数器计数值回到MCPWM_TH21中断标志位
	MCPWM_TH30_IRQ_IF	BIT8	计数器计数值回到MCPWM_TH30中断标志位
	MCPWM_TH31_IRQ_IF	BIT9	计数器计数值回到MCPWM_TH31中断标志位
	MCPWM_TMR0_IRQ_IF	BIT10	计数器计数值回到MCPWM_TMR0中断标志位
	MCPWM_TMR1_IRQ_IF	BIT11	计数器计数值回到MCPWM_TMR1中断标志位
	MCPWM_TMR2_IRQ_IF	BIT12	计数器计数值回到MCPWM_TMR0中断标志位
	MCPWM_TMR3_IRQ_IF	BIT13	计数器计数值回到MCPWM_TMR1中断标志位
	MCPWM_SHADE_IRQ_IF	BIT14	MCPWM_TH/MCPWM_↔TH00-MCPWM_TH31/↔MCPWM_TR0-MCPWM_↔TR3等寄存器更新到MCPWM实际运行系统中断标志位
输出参数:	无		

返回

返回值: 无

注解

其它说明： 无

警告

无

示例代码:

```
if (MCPWM_GetIRQFlag (MCPWM_T0_IRQ_EN) ) // 获取MCPWM的计数器计数值回到MCPWM_TH中断标志
{
    MCPWM_ClearIRQFlag (MCPWM_T0_IRQ_EN) ; // 清除MCPWM的计数器计数值回到MCPWM_TH中断标志
}
```

Date	Version	Author	Description
2022年04月20日	1.0	Zhu Jie	创建

修改日志:

在文件 [lks32mc08x_mcpwm.c](#) 第 [257](#) 行定义.

3.27.3.2 MCPWM_GetIRQFlag()

```
u16 MCPWM_GetIRQFlag (
    uint32_t INT_flag )
```

函数名称: **u16 MCPWM_GetIRQFlag(uint32_t INT_flag)**

功能描述: 获取MCPWM中断标志

参见

被引用内容： 无

参数

输入参数:	MCPWM_T0_IRQ_IF	BIT0	计数器计数值回到MCPWM_TH中断标志位
	MCPWM_T1_IRQ_IF	BIT1	计数器计数值回到0中断标志位
	MCPWM_TH00_IRQ_IF	BIT2	计数器计数值回到MCPWM_TH00中断标志位
	MCPWM_TH01_IRQ_IF	BIT3	计数器计数值回到MCPWM_TH01中断标志位
	MCPWM_TH10_IRQ_IF	BIT4	计数器计数值回到MCPWM_TH10中断标志位
	MCPWM_TH11_IRQ_IF	BIT5	计数器计数值回到MCPWM_TH11中断标志位
	MCPWM_TH20_IRQ_IF	BIT6	计数器计数值回到MCPWM_TH20中断标志位
	MCPWM_TH21_IRQ_IF	BIT7	计数器计数值回到MCPWM_TH21中断标志位
	MCPWM_TH30_IRQ_IF	BIT8	计数器计数值回到MCPWM_TH30中断标志位
	MCPWM_TH31_IRQ_IF	BIT9	计数器计数值回到MCPWM_TH31中断标志位
	MCPWM_TMR0_IRQ_IF	BIT10	计数器计数值回到MCPWM_TMR0中断标志位
	MCPWM_TMR1_IRQ_IF	BIT11	计数器计数值回到MCPWM_TMR1中断标志位
	MCPWM_TMR2_IRQ_IF	BIT12	计数器计数值回到MCPWM_TMR0中断标志位
	MCPWM_TMR3_IRQ_IF	BIT13	计数器计数值回到MCPWM_TMR1中断标志位
	MCPWM_SHADE_IRQ_IF	BIT14	MCPWM_TH/MCPWM_↔TH00-MCPWM_TH31/↔MCPWM_TR0-MCPWM_↔TR3等寄存器更新到MCPWM实际运行系统中断标志位
输出参数:	无		

返回

返回 值: 读对应标志位是否为1, 1位置位, 0为未置位

注解

其它说明： 无

警告

只有对应中断使能后， 改为才能读取， 如果对应中断未使能， 读取结果一直为0

示例代码:

```
if (MCPWM_GetIRQFlag (MCPWM_T0_IRQ_EN) ) //获取MCPWM的计数器计数值回到MCPWM_TH中断标志
{
}
```

Date	Version	Author	Description
2022年04月20日	1.0	Zhu Jie	创建

修改日志:

在文件 [lks32mc08x_mcpwm.c](#) 第 191 行定义.

3.27.3.3 MCPWM_Init()

```
void MCPWM_Init (
    MCPWM_InitTypeDef * MCPWM_InitStruct )
```

函数名称: void [MCPWM_Init\(MCPWM_InitTypeDef *MCPWM_InitStruct\)](#)

功能描述: MCPWM初始化函数

参见

被引用内容: [SYS_ModuleClockCmd\(\)](#)

参数

输入参数:	MCPWM_InitTypeDef *MCPWM_InitStruct
输出参数:	无

返回

返回值: 无

注解

其它说明： 无

警告

无

示例代码:

```
MCPWM_InitTypeDef MCPWM_InitStructure;  
MCPWM_StructInit (&MCPWM_InitStructure); //初始化MCPWM相关结构体  
MCPWM_InitStructure.CLK_DIV = PWM_PRSC; // MCPWM时钟分频设置  
MCPWM_InitStructure.MCLK_EN = ENABLE; // 模块时钟开启  
MCPWM_InitStructure.MCPWM_Cnt_EN = ENABLE; // 主计数器开始计数使能开关  
MCPWM_InitStructure.MCPWM_WorkModeCH0 = CENTRAL_PWM_MODE; // MCPWM CH0工作模式: 中心对齐PWM模式  
MCPWM_InitStructure.MCPWM_WorkModeCH1 = CENTRAL_PWM_MODE; // 通道工作模式设置, 中心对齐或边沿对齐  
MCPWM_InitStructure.MCPWM_WorkModeCH2 = CENTRAL_PWM_MODE;  
MCPWM_InitStructure.MCPWM_WorkModeCH3 = CENTRAL_PWM_MODE;  
MCPWM_Init (&MCPWM_InitStructure);
```

Date	Version	Author	Description
2019年05月21日	1.0	Howlet	创建

修改日志:

在文件 [lks32mc08x_mcpwm.c](#) 第 107 行定义.

函数调用图:



3.27.3.4 MCPWM.StructInit()

```
void MCPWM_StructInit (  
    MCPWM_InitTypeDef * MCPWM_InitStructure )
```

函数名称: void [MCPWM_StructInit\(MCPWM_InitTypeDef* MCPWM_InitStructure\)](#)

功能描述: MCPWM结构体初始化

参见

被引用内容: 无

参数

输入参数:	MCPWM_InitTypeDef *MCPWM_InitStruct
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
MCPWM_InitTypeDef MCPWM_InitStructure;
MCPWM_StructInit(&MCPWM_InitStructure); //初始化MCPWM相关结构体
```

Date	Version	Author	Description
2019年05月21日	1.0	Howlet	创建

修改日志:

在文件 [lks32mc08x-mcpwm.c](#) 第 73 行定义.

3.27.3.5 PWMOutputs()

```
void PWMOutputs (
    FuncState t_state )
```

函数名称: void [PWMOutputs\(FuncState t_state\)](#)

功能描述: MCPWM波形输出使能与失能函数

参见

被引用内容: 无

参数

输入参数:	FuncState t_state ENABLE: 使能, DISABLE:使能
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
PWMOutputs (ENABLE) //使能MCPWM输出
```

Date	Version	Author	Description
2019年05月21日	1.0	Howlet	创建

修改日志:

在文件 [lks32mc08x_mcpwm.c](#) 第 38 行定义.

3.28 lks32mc08x_mcpwm.h

[浏览该文件的文档.](#)

```
00001
00015 #ifndef _LKS08X.PWM.H
00016 #define _LKS08X.PWM.H
00017
00018 /* Includes ----- */
00019 #include "lks32mc08x.h"
00020 #include "basic.h"
00021
00025 typedef struct
00026 {
00033     uint16_t MCPWM_PERIOD;
00034     uint8_t CLK_DIV;
00035     uint8_t MCLK_EN;
00036     uint8_t MCPWM_Cnt_EN;
00037     uint8_t GPIO_BKIN_Filter;
00038     uint8_t CMP_BKIN_Filter;
00040     uint16_t MCPWM_WorkModeCH0;
00041     uint16_t MCPWM_WorkModeCH1;
00042     uint16_t MCPWM_WorkModeCH2;
00043     uint16_t MCPWM_WorkModeCH3;
00045     uint16_t TriggerPoint0;
00046     uint16_t TriggerPoint1;
00047     uint16_t TriggerPoint2;
00048     uint16_t TriggerPoint3;
00054     uint16_t DeadTimeCH0N;
00060     uint16_t DeadTimeCH0P;
00066     uint16_t DeadTimeCH1N;
00072     uint16_t DeadTimeCH1P;
00078     uint16_t DeadTimeCH2N;
00084     uint16_t DeadTimeCH2P;
00090     uint16_t DeadTimeCH3N;
00096     uint16_t DeadTimeCH3P;
00097
```

```

00098     uint8_t  CH0N.Polarity.INV;
00099     uint8_t  CH0P.Polarity.INV;
00100     uint8_t  CH1N.Polarity.INV;
00101     uint8_t  CH1P.Polarity.INV;
00102     uint8_t  CH2N.Polarity.INV;
00103     uint8_t  CH2P.Polarity.INV;
00104     uint8_t  CH3N.Polarity.INV;
00105     uint8_t  CH3P.Polarity.INV;
00107     uint8_t  CH0P.SCTRLP;
00108     uint8_t  CH0N.SCTRLN;
00109     uint8_t  CH1P.SCTRLP;
00110     uint8_t  CH1N.SCTRLN;
00111     uint8_t  CH2P.SCTRLP;
00112     uint8_t  CH2N.SCTRLN;
00113     uint8_t  CH3P.SCTRLP;
00114     uint8_t  CH3N.SCTRLN;
00116     uint8_t  CH0_PS;
00117     uint8_t  CH0_NS;
00118     uint8_t  CH1_PS;
00119     uint8_t  CH1_NS;
00120     uint8_t  CH2_PS;
00121     uint8_t  CH2_NS;
00122     uint8_t  CH3_PS;
00123     uint8_t  CH3_NS;
00125     uint8_t  Switch.CH0N.CH0P;
00126     uint8_t  Switch.CH1N.CH1P;
00127     uint8_t  Switch.CH2N.CH2P;
00128     uint8_t  Switch.CH3N.CH3P;
00130     uint8_t  MCPWM.UpdateInterval;
00131     uint8_t  MCPWM.T0.UpdateEN;
00132     uint8_t  MCPWM.T1.UpdateEN;
00133     uint8_t  MCPWM.Auto.ERR_EN;
00135     uint8_t  FAIL0.INPUT_EN;
00136     uint8_t  FAIL1.INPUT_EN;
00137     uint8_t  FAIL0.Signal_Sel;
00138     uint8_t  FAIL1.Signal_Sel;
00139     uint8_t  FAIL0.Polarity;
00140     uint8_t  FAIL1.Polarity;
00141     uint8_t  DebugMode.PWM.out;
00143     uint8_t  CH0P.default.output;
00144     uint8_t  CH0N.default.output;
00145     uint8_t  CH1P.default.output;
00146     uint8_t  CH1N.default.output;
00147     uint8_t  CH2P.default.output;
00148     uint8_t  CH2N.default.output;
00149     uint8_t  CH3P.default.output;
00150     uint8_t  CH3N.default.output;
00152     uint8_t  T0.Update.INT_EN;
00153     uint8_t  T1.Update.INT_EN;
00155     uint8_t  TH00.Match.INT_EN;
00156     uint8_t  TH01.Match.INT_EN;
00157     uint8_t  TH10.Match.INT_EN;
00158     uint8_t  TH11.Match.INT_EN;
00159     uint8_t  TH20.Match.INT_EN;
00160     uint8_t  TH21.Match.INT_EN;
00161     uint8_t  TH30.Match.INT_EN;
00162     uint8_t  TH31.Match.INT_EN;
00164     uint8_t  TMR0.Match.INT_EN;
00165     uint8_t  TMR1.Match.INT_EN;
00166     uint8_t  TMR2.Match.INT_EN;
00167     uint8_t  TMR3.Match.INT_EN;
00169     uint8_t  MCPWM.REG.UPDATE.INT_EN;
00171     uint8_t  FAIL0.INT_EN;
00172     uint8_t  FAIL1.INT_EN;
00174     uint8_t  IO0_PPE;
00175     uint8_t  IO1_PPE;
00176     uint8_t  IO2_PPE;
00177     uint8_t  IO3_PPE;
00179 } MCPWM.InitTypeDef;
00180
00181 #define MCPWM.MOE_ENABLE_MASK ((uint16_t)0x0040)
00182 #define MCPWM.MOE_DISABLE_MASK ((uint16_t)~0x0040)
00184 #define CENTRAL_PWM_MODE      0
00185 #define EDGE_PWM_MODE        1
00187 #define HIGH_LEVEL            1
00188 #define LOW_LEVEL             0
00190 #define HIGH_LEVEL_ACTIVE     0
00191 #define LOW_LEVEL_ACTIVE      1
00193 #define FAIL_SEL_CMP          1
00194 #define FAIL_SEL_IO           0
00196 #define MCPWM.UPDATE_REG()    \
00197     {                          \
00198         MCPWM.UPDATE = 0x00ff; \
00199     }
00201 #define MCPWM.T0_IRQ_EN      BIT0
00202 #define MCPWM.T1_IRQ_EN      BIT1
00203 #define MCPWM.TH00_IRQ_EN    BIT2

```

```

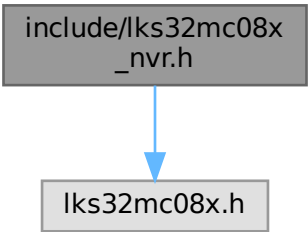
00204 #define MCPWM.TH01.IRQ_EN BIT3
00205 #define MCPWM.TH10.IRQ_EN BIT4
00206 #define MCPWM.TH11.IRQ_EN BIT5
00207 #define MCPWM.TH20.IRQ_EN BIT6
00208 #define MCPWM.TH21.IRQ_EN BIT7
00209 #define MCPWM.TH30.IRQ_EN BIT8
00210 #define MCPWM.TH31.IRQ_EN BIT9
00211 #define MCPWM.TMR0.IRQ_EN BIT10
00212 #define MCPWM.TMR1.IRQ_EN BIT11
00213 #define MCPWM.TMR2.IRQ_EN BIT12
00214 #define MCPWM.TMR3.IRQ_EN BIT13
00215 #define MCPWM.SHADE.IRQ_EN BIT14
00218 #define MCPWM.CH0.SHORT.IRQ_EN BIT0
00219 #define MCPWM.CH1.SHORT.IRQ_EN BIT1
00220 #define MCPWM.CH2.SHORT.IRQ_EN BIT2
00221 #define MCPWM.CH3.SHORT.IRQ_EN BIT3
00222 #define MCPWM.FAIL0.IRQ_EN BIT4
00223 #define MCPWM.FAIL1.IRQ_EN BIT5
00225 #define MCPWM.T0.IRQ_IF BIT0
00226 #define MCPWM.T1.IRQ_IF BIT1
00227 #define MCPWM.TH00.IRQ_IF BIT2
00228 #define MCPWM.TH01.IRQ_IF BIT3
00229 #define MCPWM.TH10.IRQ_IF BIT4
00230 #define MCPWM.TH11.IRQ_IF BIT5
00231 #define MCPWM.TH20.IRQ_IF BIT6
00232 #define MCPWM.TH21.IRQ_IF BIT7
00233 #define MCPWM.TH30.IRQ_IF BIT8
00234 #define MCPWM.TH31.IRQ_IF BIT9
00235 #define MCPWM.TMR0.IRQ_IF BIT10
00236 #define MCPWM.TMR1.IRQ_IF BIT11
00237 #define MCPWM.TMR2.IRQ_IF BIT12
00238 #define MCPWM.TMR3.IRQ_IF BIT13
00239 #define MCPWM.SHADE.IRQ_IF BIT14
00242 #define MCPWM.CH0.SHORT.IRQ_IF BIT15
00243 #define MCPWM.CH1.SHORT.IRQ_IF BIT16
00244 #define MCPWM.CH2.SHORT.IRQ_IF BIT17
00245 #define MCPWM.CH3.SHORT.IRQ_IF BIT18
00246 #define MCPWM.FAIL0.IRQ_IF BIT19
00247 #define MCPWM.FAIL1.IRQ_IF BIT20
00249 // mcpwm寄存器中断标志位
00250 #define MCPWM.IF.SHADE BIT14
00251 #define MCPWM.IF.TMR3 BIT13
00252 #define MCPWM.IF.TMR2 BIT12
00253 #define MCPWM.IF.TMR1 BIT11
00254 #define MCPWM.IF.TMR0 BIT10
00255 #define MCPWM.IF.TH31 BIT9
00256 #define MCPWM.IF.TH30 BIT8
00257 #define MCPWM.IF.TH21 BIT7
00258 #define MCPWM.IF.TH20 BIT6
00259 #define MCPWM.IF.TH11 BIT5
00260 #define MCPWM.IF.TH10 BIT4
00261 #define MCPWM.IF.TH01 BIT3
00262 #define MCPWM.IF.TH00 BIT2
00263 #define MCPWM.IF.T1 BIT1
00264 #define MCPWM.IF.T0 BIT0
00265 #define MCPWM.EIF.FAIL1 BIT5
00266 #define MCPWM.EIF.FAIL0 BIT4
00267 #define MCPWM.EIF.CH3.SHORT BIT3
00268 #define MCPWM.EIF.CH2.SHORT BIT2
00269 #define MCPWM.EIF.CH1.SHORT BIT1
00270 #define MCPWM.EIF.CH0.SHORT BIT0
00271
00272 void PWMOutputs(FuncState t.state);
00273 void MCPWM_Init(MCPWM_InitTypeDef *MCPWM_InitStruct);
00274 void MCPWM_StructInit(MCPWM_InitTypeDef *MCPWM_InitStruct);
00275 uint16_t MCPWM_GetIRQFlag(uint32_t INT_flag);
00276 void MCPWM_ClearIRQFlag(uint32_t INT_flag);
00277 #endif /*__CHANHOM_PWM_H */
00278
00279 /***** (C) COPYRIGHT LINKO SEMICONDUCTOR *****/

```

3.29 include/lks32mc08x_nvr.h 文件参考

文件名称: [lks32mc08x_nvr.h](#)
 文件标识: 无
 内容摘要: NVR驱动头文件
 其它说明: 无

#include "lks32mc08x.h"
lks32mc08x_nvr.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



函数

- uint32_t [Read_NVR](#) (uint32_t adr)
 读取NVR 1KB大小的用户区函数
- uint32_t [Read_Trim](#) (uint32_t adr)
 读取芯片模块校正值函数

3.29.1 详细描述

文件名称: [lks32mc08x_nvr.h](#)
文件标识: 无
内容摘要: NVR驱动头文件
其它说明: 无

版权所有
(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2018年07月25日	1.0	DengT	创建

修改日志:

在文件 [lks32mc08x_nvr.h](#) 中定义.

3.29.2 函数说明

3.29.2.1 Read_NVR()

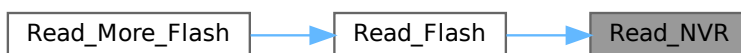
```
uint32_t Read_NVR (  
    uint32_t adr )
```

读取NVR 1KB大小的用户区函数

示例代码：读取NVR用户地址0的数据

```
uint32_t LKS.NVRValue = 0;  
LKS.NVRValue = Read_NVR(0x00); //范围0x000~0x400 1KB空间
```

这是这个函数的调用关系图:



3.29.2.2 Read_Trim()

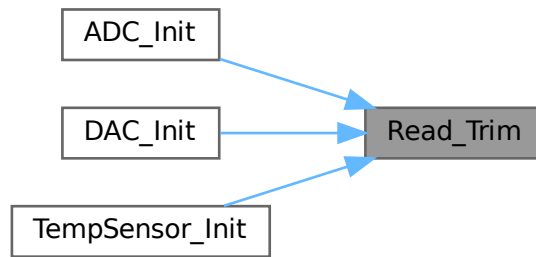
```
uint32_t Read_Trim (  
    uint32_t adr )
```

读取芯片模块校正值函数

示例代码：读取芯片ID

```
uint32_t LKS.ID1;  
uint32_t LKS.ID2;  
uint32_t LKS.ID3;  
uint32_t LKS.ID4;  
LKS.ID1 = Read_Trim(0x0000084C);  
LKS.ID2 = Read_Trim(0x00000850);  
LKS.ID3 = Read_Trim(0x00000854);  
LKS.ID4 = Read_Trim(0x00000858);
```

这是这个函数的调用关系图:



3.30 lks32mc08x_nvr.h

[浏览该文件的文档.](#)

```

00001
00014 #ifndef _LKS32MC08X_NVR_H_
00015 #define _LKS32MC08X_NVR_H_
00016
00017 #include "lks32mc08x.h"
00018
00027 uint32_t Read_NVR(uint32_t adr);
00028
00043 uint32_t Read_Trim(uint32_t adr);
00044
00045 #endif
00046 /***** (C) COPYRIGHT LINKO SEMICONDUCTOR *****/
00047 /* -----END OF FILE----- */
  
```

3.31 include/lks32mc08x_opa.h 文件参考

文件名称: [lks32mc08x_opa.h](#)
 文件标识: 无
 内容摘要: OPA驱动头文件
 其它说明: 无

#include "lks32mc08x_lib.h"

lks32mc08x_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 [PGA_GAIN_20](#) 0x00
反馈电阻200:10.2
- #define [PGA_GAIN_9P5](#) 0x01
反馈电阻190:20.2
- #define [PGA_GAIN_6](#) 0x02
反馈电阻180:30.4
- #define [PGA_GAIN_4P25](#) 0x03
反馈电阻170:40.4

枚举

- enum [enumOPA](#) { [OPA0](#) = 0 , [OPA1](#) = 1 , [OPA2](#) = 2 , [OPA3](#) = 3 }

函数

- void [OPA_Init](#) ([enumOPA](#) opa, [OPA_InitTypeDef](#) *OPA_InitStruct)
函数名称: void [OPA_Init\(enumOPA opa, OPA_InitTypeDef* OPA_InitStruct\)](#)
- void [OPA_StructInit](#) ([OPA_InitTypeDef](#) *OPA_InitStruct)
函数名称: void [OPA_StructInit\(OPA_InitTypeDef* OPA_InitStruct\)](#)
- void [OPA_OUT](#) ([enumOPA](#) opa, [FuncState](#) state)
函数名称: void [OPA_OUT\(enumOPA opa, FuncState state\)](#)

3.31.1 详细描述

文件名称: [lks32mc08x_opa.h](#)
文件标识: 无
内容摘要: OPA驱动头文件
其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2016年06月02日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_opa.h](#) 中定义.

3.31.2 宏定义说明

3.31.2.1 PGA_GAIN_20

```
#define PGA_GAIN_20 0x00
```

反馈电阻200:10.2

在文件 [lks32mc08x_opa.h](#) 第 45 行定义.

3.31.2.2 PGA_GAIN_4P25

```
#define PGA_GAIN_4P25 0x03
```

反馈电阻170:40.4

在文件 [lks32mc08x_opa.h](#) 第 48 行定义.

3.31.2.3 PGA_GAIN_6

```
#define PGA_GAIN_6 0x02
```

反馈电阻180:30.4

在文件 [lks32mc08x_opa.h](#) 第 47 行定义.

3.31.2.4 PGA_GAIN_9P5

```
#define PGA_GAIN_9P5 0x01
```

反馈电阻190:20.2

在文件 [lks32mc08x_opa.h](#) 第 46 行定义.

3.31.3 枚举类型说明

3.31.3.1 enumOPA

```
enum enumOPA
```

枚举值

OPA0	OPA0模块
OPA1	OPA1模块
OPA2	OPA2模块

在文件 [lks32mc08x_opa.h](#) 第 20 行定义.

3.31.4 函数说明

3.31.4.1 OPA_Init()

```
void OPA_Init (
    enumOPA opa,
    OPA_InitTypeDef * OPA_InitStruct )
```

函数名称: void [OPA_Init](#)(enumOPA opa, [OPA_InitTypeDef](#)* [OPA_InitStruct](#))

功能描述: OPA初始化函数

参见
被引用内容: 无

参数

输入参数:	enumOPA opa , OPA_InitTypeDef *OPA_InitStruct
输出参数:	无

返回
返回值: 无

注解
其它说明: 无

警告
无

示例代码:

```
OPA_InitTypeDef OPA_InitStruct;
OPA_StructInit (&OPA_InitStruct);           //初始化结构体
OPA_InitStruct.OPA_CLEna = ENABLE;           //使能OPA
OPA_InitStruct.OPA_Gain = PGA_GAIN_20;       //OPA内部增益电阻选择200: 10.2
OPA_Init (OPA0, &OPA_InitStruct);
```

Date	Version	Author	Description
2016年06月02日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_opa.c](#) 第 40 行定义.

3.31.4.2 OPA_OUT()

```
void OPA_OUT (
    enumOPA opa,
    FuncState state )
```

函数名称: void [OPA_OUT\(enumOPA opa, FuncState state\)](#)

功能描述: OPA外部输出，输出到外部P2.7引脚

参见

被引用内容: 无

参数

输入参数:	enumOPA opa, FuncState state
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
OPA\_OUT (OPA0, ENABLE); //将OPA0输出通过外部P2.7引脚送出
```

Date	Version	Author	Description
2022年04月20日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_opa.c](#) 第 130 行定义.

3.31.4.3 OPA_StructInit()

```
void OPA_StructInit (
    OPA_InitTypeDef * OPA_InitStruct )
```

函数名称: void [OPA_StructInit\(OPA_InitTypeDef* OPA_InitStruct\)](#)

功能描述: OPA结构体初始化

参见
被引用内容: 无

参数

输入参数:	OPA_InitTypeDef *OPA_InitStruct
输出参数:	无

返回
返回 值: 无

注解
其它说明: 无

警告
无

示例代码:

```
OPA\_InitTypeDef OPA_InitStruct;
OPA_StructInit (&OPA_InitStruct);           //初始化结构体
```

Date	Version	Author	Description
2016年06月02日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_opa.c](#) 第 105 行定义.

3.32 lks32mc08x_opa.h

浏览该文件的文档.

```

00001
00014 #ifndef __LKS32MC08x_OPA_H
00015 #define __LKS32MC08x_OPA_H
00016
00017 /* Includes ----- */
00018 #include "lks32mc08x_lib.h"
00019
00020 typedef enum
00021 {
00022     OPA0 = 0,
00023     OPA1 = 1,
00024     OPA2 = 2,
00025     OPA3 = 3
00026 } enumOPA;
00027
00031 typedef struct
00032 {
00040     uint32_t OPA_Gain;
00041     uint32_t OPA_CLEna;
00042 } OPA_InitTypeDef;
00043
00044 /* -----PGA操作相关定义 ----- */
00045 #define PGA_GAIN_20    0x00
00046 #define PGA_GAIN_9P5   0x01
00047 #define PGA_GAIN_6     0x02
00048 #define PGA_GAIN_4P25  0x03
00050 void OPA_Init(enumOPA opa, OPA_InitTypeDef *OPA_InitStruct);
00051 void OPA_StructInit(OPA_InitTypeDef *OPA_InitStruct);
00052 void OPA_OUT(enumOPA opa, FuncState state);
00053
00054 #endif /* __lks32mc08x_OPA_H */
00055
00056 /***** (C) COPYRIGHT LINKO SEMICONDUCTOR *****/

```

3.33 include/lks32mc08x_spi.h 文件参考

文件名称: [lks32mc08x_spi.h](#)

文件标识: 无

内容摘要: SPI驱动头文件

其它说明: 无

#include "lks32mc08x_lib.h"

lks32mc08x_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_TypeDef](#)
SPI寄存器结构体句柄
- struct [SPI_InitTypeDef](#)
SPI结构体基址定义

宏定义

- `#define SPI_FIRSTSEND_LSB 1`
低位先传
- `#define SPI_FIRSTSEND_MSB 0`
高位先传
- `#define SPI_DMA_ENABLE 0`
使能DMA搬运
- `#define SPI_DMA_DISABLE 1`
禁止DMA搬运
- `#define SPI_IRQEna.Enable BIT7`
中断使能定义, SPI中断使能
- `#define SPI_IRQEna.TranDone BIT6`
中断使能定义, 传输完成中断
- `#define SPI_IRQEna.SSErr BIT5`
中断使能定义, 传输异常事件中断
- `#define SPI_IRQEna.DataOver BIT4`
中断使能定义, 传输溢出事件中断
- `#define SPI_IF_TranDone BIT2`
中断标志定义, 传输完成中断标志位
- `#define SPI_IF_SSErr BIT1`
中断标志定义, 传输异常中断标志位
- `#define SPI_IF_DataOver BIT0`
中断标志定义, 传输溢出中断标志位

枚举

- `enum SPI_Mode { SPI_Master = 0x01, SPI_Slave = 0x00 }`
SPI模式配置枚举
- `enum SPI_Duplex { SPI_Full = 0x0, SPI_Half_Tx = 0x2, SPI_Half_Rx = 0x3 }`
SPI工作模式配置枚举

函数

- `void SPI_Init (SPI_TypeDef *SPIx, SPI_InitTypeDef *SPI_InitStruct)`
函数名称: `void SPI_Init(SPI_TypeDef* SPIx, SPI_InitTypeDef* SPI_InitStruct)`
- `void SPI_StructInit (SPI_InitTypeDef *SPI_InitStruct)`
函数名称: `void SPI_StructInit(SPI_InitTypeDef* SPI_InitStruct)`
- `uint8_t SPI_GetIRQFlag (SPI_TypeDef *SPIx, u8 IRQ_Flag)`
函数名称: `uint8_t SPI_GetIRQFlag(SPI_TypeDef *SPIx, u8 IRQ_Flag)`
- `void SPI_ClearIRQFlag (SPI_TypeDef *SPIx, u8 IRQ_Flag)`
函数名称: `void SPI_ClearIRQFlag(SPI_TypeDef *SPIx, u8 IRQ_Flag)`
- `void SPI_SendData (SPI_TypeDef *SPIx, uint8_t n)`
函数名称: `void SPI_SendData(SPI_TypeDef *SPIx, uint8_t n)`
- `uint8_t SPI_ReadData (SPI_TypeDef *SPIx)`
函数名称: `uint8_t SPI_ReadData(SPI_TypeDef *SPIx)`

3.33.1 详细描述

文件名称: [lks32mc08x_spi.h](#)
文件标识: 无
内容摘要: SPI驱动头文件
其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2016年05月21日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_spi.h](#) 中定义.

3.33.2 宏定义说明

3.33.2.1 SPI_DMA_DISABLE

```
#define SPI_DMA_DISABLE 1
```

禁止DMA搬运

在文件 [lks32mc08x_spi.h](#) 第 99 行定义.

3.33.2.2 SPI_DMA_ENABLE

```
#define SPI_DMA_ENABLE 0
```

使能DMA搬运

在文件 [lks32mc08x_spi.h](#) 第 98 行定义.

3.33.2.3 SPI_FIRSTSEND_LSB

```
#define SPI_FIRSTSEND_LSB 1
```

低位先传

在文件 [lks32mc08x_spi.h](#) 第 95 行定义.

3.33.2.4 SPI_FIRSTSEND_MSB

```
#define SPI_FIRSTSEND_MSB 0
```

高位先传

在文件 [lks32mc08x_spi.h](#) 第 96 行定义.

3.33.2.5 SPI_IF_DataOver

```
#define SPI_IF_DataOver BIT0
```

中断标志定义，传输溢出中断标志位

在文件 [lks32mc08x_spi.h](#) 第 108 行定义.

3.33.2.6 SPI_IF_SSErr

```
#define SPI_IF_SSErr BIT1
```

中断标志定义，传输异常中断标志位

在文件 [lks32mc08x_spi.h](#) 第 107 行定义.

3.33.2.7 SPI_IF_TranDone

```
#define SPI_IF_TranDone BIT2
```

中断标志定义，传输完成中断标志位

在文件 [lks32mc08x_spi.h](#) 第 106 行定义.

3.33.2.8 SPI_IRQEna_DataOver

```
#define SPI_IRQEna_DataOver BIT4
```

中断使能定义，传输溢出事件中断

在文件 [lks32mc08x_spi.h](#) 第 104 行定义.

3.33.2.9 SPI_IRQEna_Enable

```
#define SPI_IRQEna_Enable BIT7
```

中断使能定义，SPI中断使能

在文件 [lks32mc08x_spi.h](#) 第 101 行定义.

3.33.2.10 SPI_IRQEna_SSErr

```
#define SPI_IRQEna_SSErr BIT5
```

中断使能定义，传输异常事件中断

在文件 [lks32mc08x_spi.h](#) 第 103 行定义.

3.33.2.11 SPI_IRQEna-TranDone

```
#define SPI_IRQEna-TranDone BIT6
```

中断使能定义，传输完成中断

在文件 [lks32mc08x_spi.h](#) 第 102 行定义.

3.33.3 枚举类型说明

3.33.3.1 SPI_Duplex

```
enum SPI_Duplex
```

SPI工作模式配置枚举

枚举值

SPI_Full	SPI全双工
SPI_Half_Tx	SPI半双工仅发送
SPI_Half_Rx	SPI半双工仅接收

在文件 [lks32mc08x_spi.h](#) 第 32 行定义.

3.33.3.2 SPI_Mode

```
enum SPI_Mode
```

SPI模式配置枚举

枚举值

SPI_Master	SPI主模式
SPI_Slave	SPI从模式

在文件 [lks32mc08x_spi.h](#) 第 23 行定义.

3.33.4 函数说明

3.33.4.1 SPI_ClearIRQFlag()

```
void SPI_ClearIRQFlag (
    SPI_TypeDef * SPIx,
    u8 IRQ_Flag )
```

函数名称: void SPI_ClearIRQFlag(SPI_TypeDef *SPIx,u8 IRQ_Flag)

功能描述: 清除SPI中断标志位

参见
被引用内容: 无

参数

输入参数:	SPIx: SPI_IRQ_Flag:		
	SPI_IF_TranDone		传输完成中断标志位
	SPI_IF_SSErr		传输异常中断标志位
	SPI_IF_DataOver		传输溢出中断标志位
输出参数:	无		

返回
返回值: 无

注解
其它说明: 无

警告
无

示例代码:

```
if(SPI_GetIRQFlag(SPI,SPI_IF_TranDone))//获取传输完成中断标志位
{
    SPI_ClearIRQFlag(SPI,SPI_IF_TranDone); //清除传输完成中断标志位
}
```

Date	Version	Author	Description
2022年04月20日	1.0	Zhu Jie	创建

修改日志:

在文件 [lks32mc08x_spi.c](#) 第 222 行定义.

3.33.4.2 SPI_GetIRQFlag()

```
uint8_t SPI_GetIRQFlag (
    SPI_TypeDef * SPIx,
    u8 IRQ_Flag )
```

函数名称: `uint8_t SPI_GetIRQFlag(SPI_TypeDef *SPIx,u8 IRQ_Flag)`

功能描述: 获取SPI中断标志位

参见
被引用内容: 无

参数

输入参数:	SPIx: SPI_IRQ_Flag:	
	SPI_IF_TranDone	传输完成中断标志位
	SPI_IF_SSErr	传输异常中断标志位
	SPI_IF_DataOver	传输溢出中断标志位
输出参数:	无	

返回
返回值: 无

注解
其它说明: 无

警告
只有对应中断使能后, 改为才能读取, 如果对应中断未使能, 读取结果一直为0

示例代码:

```
if (SPI_GetIRQFlag (SPI, SPI_IF_TranDone)) //获取传输完成中断标志位
{
}
```

Date	Version	Author	Description
2022年04月20日	1.0	Zhu Jie	创建

修改日志:

在文件 `lks32mc08x_spi.c` 第 181 行定义.

3.33.4.3 SPI_Init()

```
void SPI_Init (
    SPI_TypeDef * SPIx,
    SPI_InitTypeDef * SPI_InitStruct )
```

函数名称: `void SPI_Init(SPI_TypeDef* SPIx, SPI_InitTypeDef* SPI_InitStruct)`

功能描述: SPI初始化函数

参见

被引用内容: `SYS_ModuleClockCmd()`

参数

输入参数:	SPIx: SPI , <code>SPI_InitTypeDef</code> * <code>SPI_InitStruct</code>
输出参数:	无

返回

返回 值: 无

注解

其它说明: 无

警告

无

示例代码:

```
SPI_InitTypeDef SPI_InitStruct;
SPI_StructInit (&SPI_InitStruct); //SPI结构体初始化
SPI_InitStruct.Duplex = SPI_Full; //双工模式设置
SPI_InitStruct.Mode = SPI_Master; //master主机模式
SPI_InitStruct.EN = ENABLE; //使能SPI模块
SPI_InitStruct.DMA = SPI_DMA_DISABLE; // 选择SPI搬移方式，由MCU搬运数据到SPI
SPI_InitStruct.Trig = 0; //内部自动触发传输
SPI_InitStruct.DataOrder = SPI_FIRSTSEND_MSB; //该字节先发送 MSB
SPI_InitStruct.CPHA = 0; //第一个沿为发送数据时刻
SPI_InitStruct.CPOL = 0; //CLK默认高电平
SPI_InitStruct.BaudRate = 30; //传输速度96MHZ/(2*(30+1))= 1.548MHZ
SPI_InitStruct.ByteLength = 0; //发送零个字节
SPI_InitStruct.IRQEna = DISABLE; //关闭SPI各中断
SPI_Init (SPI, &SPI_InitStruct); //SPI初始化程序
```

Date	Version	Author	Description
2016年05月21日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_spi.c](#) 第 49 行定义.

函数调用图:



3.33.4.4 SPI_ReadData()

```
uint8_t SPI_ReadData (
    SPI_TypeDef * SPIx )
```

函数名称: uint8_t [SPI_ReadData\(SPI_TypeDef *SPIx\)](#)

功能描述: SPI读缓冲区数据

参见

被引用内容: 无

参数

输入参数:	SPIx: SPI
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
u8 SPI_RXValue = 0;
SPI_RXValue = SPI_ReadData(SPI); //SPI接收一字节数据
```

Date	Version	Author	Description
2016年05月21日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_spi.c](#) 第 150 行定义.

3.33.4.5 SPI_SendData()

```
void SPI_SendData (
    SPI_TypeDef * SPIx,
    uint8_t n )
```

函数名称: void [SPI_SendData\(SPI_TypeDef *SPIx, uint8_t n\)](#)

功能描述: SPI发送数据

参见

被引用内容: 无

参数

输入参数:	SPIx: SPI , n: 发送一字节数据
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
SPI_SendData(SPI,0x12); //SPI发送0x12一字节数据
```

Date	Version	Author	Description
2016年05月21日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_spi.c](#) 第 125 行定义.

3.33.4.6 SPI_StructInit()

```
void SPI_StructInit (
    SPI_InitTypeDef * SPI_InitStruct )
```

函数名称: void [SPI_StructInit\(SPI_InitTypeDef* SPI_InitStruct\)](#)

功能描述: SPI结构体初始化

参见

被引用内容: 无

参数

输入参数:	SPI_InitTypeDef *SPI_InitStruct
输出参数:	无

返回

返回 值: 无

注解

其它说明: 无

警告

无

示例代码:

```
SPI\_InitTypeDef SPI_InitStruct;
SPI_StructInit (&SPI_InitStruct); //SPI结构体初始化
```

Date	Version	Author	Description
2016年05月21日	1.0	cfwu	创建

修改日志:

在文件 `lks32mc08x_spi.c` 第 86 行定义.

3.34 lks32mc08x_spi.h

[浏览该文件的文档.](#)

```
00001
00014 #ifndef __LKS32MC08x_SPI_H
00015 #define __LKS32MC08x_SPI_H
00016
00017 /* Includes -----*/
00018 #include "lks32mc08x_lib.h"
00019
00023 typedef enum
00024 {
00025     SPI_Master = 0x01,
00026     SPI_Slave = 0x00
00027 } SPI_Mode;
00028
00032 typedef enum
00033 {
00034     SPI_Full = 0x0,
00035     SPI_Half_Tx = 0x2,
00036     SPI_Half_Rx = 0x3
00037 } SPI_Duplex;
00038
00042 typedef struct
00043 {
00044     __IO uint32_t CFG;
00045     __IO uint32_t IE;
00046     __IO uint32_t DIV;
00047     __IO uint32_t TX_DATA;
00048     __IO uint32_t RX_DATA;
00049     __IO uint32_t SIZE;
00050 } SPI_TypeDef;
00051
00055 #ifndef SPI
00056 #define SPI ((SPI_TypeDef *)SPI0_BASE)
00057 #endif
00058
00062 typedef struct
00063 {
00068     SPI_Duplex Duplex;
00069     uint8_t SS;
00074     SPI_Mode Mode;
00075     uint8_t CPHA;
00076     uint8_t CPOL;
00077     uint8_t DataOrder;
00078     uint8_t EN;
00086     uint8_t IRQEna;
00087     uint8_t Trig;
00089     uint8_t DMA;
00090     uint8_t BaudRate;
00092     uint8_t ByteLength;
00093 } SPI_InitTypeDef;
00094
00095 #define SPI_FIRSTSEND_LSB 1
00096 #define SPI_FIRSTSEND_MSB 0
00098 #define SPI_DMA_ENABLE 0
00099 #define SPI_DMA_DISABLE 1
00101 #define SPI_IRQEna_Enable BIT7
00102 #define SPI_IRQEna_TranDone BIT6
00103 #define SPI_IRQEna_SSErr BIT5
00104 #define SPI_IRQEna_DataOver BIT4
00106 #define SPI_IF_TranDone BIT2
00107 #define SPI_IF_SSErr BIT1
00108 #define SPI_IF_DataOver BIT0
00110 void SPI_Init(SPI_TypeDef *SPIx, SPI_InitTypeDef *SPI_InitStruct);
00111 void SPI_StructInit(SPI_InitTypeDef *SPI_InitStruct);
00112
```

```

00113 uint8_t SPI_GetIRQFlag(SPI_TypeDef *SPIx, u8 IRQ_Flag);
00114 void SPI_ClearIRQFlag(SPI_TypeDef *SPIx, u8 IRQ_Flag);
00115
00116 void SPI_SendData(SPI_TypeDef *SPIx, uint8_t n);
00117 uint8_t SPI_ReadData(SPI_TypeDef *SPIx);
00118 #endif /*_lks32mc08x.SPI_H */
00119
00120 /***** (C) COPYRIGHT LINKO SEMICONDUCTOR *****/

```

3.35 include/lks32mc08x_sys.h 文件参考

文件名称: lks32mc08x_sys.h

文件标识: 无

内容摘要: sys驱动头文件

其它说明: 无

```
#include "lks32mc08x_lib.h"
```

lks32mc08x_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:



结构体

- struct **WAKE_InitTypeDef**
低功耗时钟配置中间变量
- struct **SYS_InitTypeDef**
系统初始化结构体句柄，初始化时请定义该句柄，并用其它地址来传参

宏定义

- #define **SYS_PLLSRSEL_RCH** 0x00
- #define **SYS_PLLSRSEL_CRYSTAL** 0x01
- #define **SYS_CLK_SEL_PLL** 0x01
系统时钟选择，内部 PLL 时钟
- #define **SYS_CLK_SEL_CLK_HS** 0x00
系统时钟选择，内部 RC 时钟
- #define **SYS_PLLREDIV_4** 0x00
在 PLL 分频基础上再/4
- #define **SYS_PLLREDIV_1** 0x01
在 PLL 分频基础上不再分频
- #define **SYS_Clk_SPIDiv1** 0

- SP*时钟分频, 占时未使用
- #define **SYS_Clk_SPIDiv2** 1
*SP*时钟分频, 占时未使用
- #define **SYS_Clk_SPIDiv4** 2
*SP*时钟分频, 占时未使用
- #define **SYS_Clk_SPIDiv8** 3
*SP*时钟分频, 占时未使用
- #define **SYS_Clk_UARTDiv1** 0
*UART*时钟分频, 占时未使用
- #define **SYS_Clk_UARTDiv2** 1
*UART*时钟分频, 占时未使用
- #define **SYS_Clk_UARTDiv4** 2
*UART*时钟分频, 占时未使用
- #define **SYS_Clk_UARTDiv8** 3
*UART*时钟分频, 占时未使用
- #define **SYS_Module_I2C** BIT0
*I2C*模块
- #define **SYS_Module_HALL** BIT1
*HALL*模块
- #define **SYS_Module_TIMER** BIT2
*TIMER*模块
- #define **SYS_Module_MCPWM** BIT3
*MCPWM*模块
- #define **SYS_Module_UART0** BIT4
*UART0*模块
- #define **SYS_Module_UART1** BIT5
*UART1*模块
- #define **SYS_Module_DSP** BIT6
*DSP*模块
- #define **SYS_Module_CAN** BIT7
*CAN*模块
- #define **SYS_Module_SPI** BIT8
*SPI*模块
- #define **SYS_AnalogModule_ADC** BIT0
*ADC*模块
- #define **SYS_AnalogModule_OPA0** BIT2
*OPA0*模块
- #define **SYS_AnalogModule_OPA1** BIT3
*OPA1*模块
- #define **SYS_AnalogModule_OPA2** BIT4
*OPA2*模块
- #define **SYS_AnalogModule_OPA3** BIT5
*OPA3*模块
- #define **SYS_AnalogModule_CMP0** BIT6
*CMP0*模块
- #define **SYS_AnalogModule_CMP1** BIT7
*CMP1*模块
- #define **SYS_AnalogModule_BGP** BIT8
*BGP*模块
- #define **SYS_AnalogModule_RCH** BIT10
*RCH*模块

- #define SYS_AnalogModule_DAC BIT12
DAC模块
- #define SYS_AnalogModule_TMP BIT13
温度传感器模块
- #define SYS_AnalogModule_XTAL BIT14
晶振模块
- #define SYS_AnalogModule_PLL BIT15
PLL模块
- #define SYS_WD_TimeOut2s 0
看门狗超时时间2s复位
- #define SYS_WD_TimeOut4s 1
看门狗超时时间4s复位
- #define SYS_WD_TimeOut8s 2
看门狗超时时间8s复位
- #define SYS_WD_TimeOut64s 3
看门狗超时时间64s复位
- #define SYS_RstSrc_LPOR 0x01
复位信号源: 内部1.5V电源管理复位
- #define SYS_RstSrc_HPOR 0x02
复位信号源: 内部3.3V电源管理
- #define SYS_RstSrc_KEYBOARD 0x04
复位信号源: 按键复位
- #define SYS_RstSrc_WDT 0x08
复位信号源: WDT复位
- #define SYS_WakeInterval_0.25s 0x00
休眠唤醒间隔时间0.25s
- #define SYS_WakeInterval_0.5s 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 WAKEIO_P0_0 BIT0
系统低功耗时使用P0.0作为唤醒IO
- #define WAKEIO_P0_1 BIT1
系统低功耗时使用P0.1作为唤醒IO
- #define WAKEIO_P1_0 BIT2
系统低功耗时使用P1.0作为唤醒IO
- #define WAKEIO_P1_1 BIT3
系统低功耗时使用P1.1作为唤醒IO
- #define MCU_POWER_3v3 0
- #define MCU_POWER_5v0 1
- #define POWER_MODE MCU_POWER_5v0

枚举

- enum `enumWakeTrigLevel` { `WAKETRIG_LOW` , `WAKETRIG_HIGH` }
低功耗IO唤醒电平枚举成员

函数

- void `SYS_Init` (`SYS_InitTypeDef` *`SYS_InitStruct`)
函数名称: `void SYS_Init(SYS_InitTypeDef* SYS_InitStruct)`
- void `SYS_StructInit` (`SYS_InitTypeDef` *`SYS_InitStruct`)
函数名称: `void SYS_StructInit(SYS_InitTypeDef* SYS_InitStruct)`
- uint32_t `SYS_GetRstSource` (void)
函数名称: `uint32_t SYS_GetRstSource(void)`
- void `SYS_ClearRst` (void)
函数名称: `void SYS_ClearRst(void)`
- void `SYS_WatchDogCmd` (`FuncState` state)
函数名称: `void SYS_WatchDogCmd(FuncState state)`
- void `SYS_FeedDog` (uint32_t timeOut)
函数名称: `void SYS_FeedDog(uint32_t timeOut)`
- void `SYS_FallSleep` (void)
函数名称: `void SYS_FallSleep(void)`
- void `SYS_SetWakeInterval` (uint32_t nInterval)
函数名称: `void SYS_SetWakeInterval(uint32_t nInterval)`
- void `Switch2PLL` (void)
函数名称: `void Switch2PLL(void)`
- void `Switch2HRC` (void)
函数名称: `void Switch2HRC(void)`
- void `SetWakeIO` (uint32_t wakeIO, `enumWakeTrigLevel` trigLevel, uint8_t ena)
函数名称: `void SetWakeIO(uint32_t wakeIO, enumWakeTrigLevel trigLevel, uint8_t ena)`
- void `SYS_ModuleClockCmd` (uint32_t nModule, `FuncState` state)
函数名称: `void SYS_ModuleClockCmd(uint32_t nModule, FuncState state)`
- void `SYS_AnalogModuleClockCmd` (uint32_t nModule, `FuncState` state)
函数名称: `void SYS_AnalogModuleClockCmd(uint32_t nModule, FuncState state)`
- void `SYS_SoftResetModule` (uint32_t nModule)
函数名称: `void SYS_SoftResetModule(uint32_t nModule)`
- void `SYS_VolSelModule` (uint32_t Vol)
- void `SYS_VolSelModuleIRQ` (void)
- void `SYS_VolSelModuleEnableIRQ` (uint32_t Vol)
- 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)

3.35.1 详细描述

文件名称: [lks32mc08x.sys.h](#)
文件标识: 无
内容摘要: sys驱动头文件
其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2016年05月21日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x.sys.h](#) 中定义.

3.35.2 宏定义说明

3.35.2.1 MCU_POWER_3v3

```
#define MCU_POWER_3v3 0
```

在文件 [lks32mc08x.sys.h](#) 第 135 行定义.

3.35.2.2 MCU_POWER_5v0

```
#define MCU_POWER_5v0 1
```

在文件 [lks32mc08x.sys.h](#) 第 136 行定义.

3.35.2.3 POWER_MODE

```
#define POWER_MODE MCU_POWER_5v0
```

在文件 [lks32mc08x.sys.h](#) 第 138 行定义.

3.35.2.4 SYS_AnalogModule_ADC

```
#define SYS_AnalogModule_ADC BIT0
```

ADC模块

在文件 [lks32mc08x.sys.h](#) 第 93 行定义.

3.35.2.5 SYS_AnalogModule.BGP

```
#define SYS_AnalogModule.BGP BIT8
```

BGP模块

在文件 [lks32mc08x_sys.h](#) 第 100 行定义.

3.35.2.6 SYS_AnalogModule.CMP0

```
#define SYS_AnalogModule.CMP0 BIT6
```

CMP0模块

在文件 [lks32mc08x_sys.h](#) 第 98 行定义.

3.35.2.7 SYS_AnalogModule.CMP1

```
#define SYS_AnalogModule.CMP1 BIT7
```

CMP1模块

在文件 [lks32mc08x_sys.h](#) 第 99 行定义.

3.35.2.8 SYS_AnalogModule.DAC

```
#define SYS_AnalogModule.DAC BIT12
```

DAC模块

在文件 [lks32mc08x_sys.h](#) 第 102 行定义.

3.35.2.9 SYS_AnalogModule.OPA0

```
#define SYS_AnalogModule.OPA0 BIT2
```

OPA0模块

在文件 [lks32mc08x_sys.h](#) 第 94 行定义.

3.35.2.10 SYS_AnalogModule.OPA1

```
#define SYS_AnalogModule.OPA1 BIT3
```

OPA1模块

在文件 [lks32mc08x_sys.h](#) 第 95 行定义.

3.35.2.11 SYS_AnalogModule_OPA2

```
#define SYS_AnalogModule_OPA2 BIT4
```

OPA2模块

在文件 [lks32mc08x_sys.h](#) 第 96 行定义.

3.35.2.12 SYS_AnalogModule_OPA3

```
#define SYS_AnalogModule_OPA3 BIT5
```

OPA3模块

在文件 [lks32mc08x_sys.h](#) 第 97 行定义.

3.35.2.13 SYS_AnalogModule_PLL

```
#define SYS_AnalogModule_PLL BIT15
```

PLL模块

在文件 [lks32mc08x_sys.h](#) 第 105 行定义.

3.35.2.14 SYS_AnalogModule_RCH

```
#define SYS_AnalogModule_RCH BIT10
```

RCH模块

在文件 [lks32mc08x_sys.h](#) 第 101 行定义.

3.35.2.15 SYS_AnalogModule_TMP

```
#define SYS_AnalogModule_TMP BIT13
```

温度传感器模块

在文件 [lks32mc08x_sys.h](#) 第 103 行定义.

3.35.2.16 SYS_AnalogModule_XTAL

```
#define SYS_AnalogModule_XTAL BIT14
```

晶振模块

在文件 [lks32mc08x_sys.h](#) 第 104 行定义.

3.35.2.17 SYS_CLK_SEL_CLK_HS

```
#define SYS_CLK_SEL_CLK_HS 0x00
```

系统时钟选择，内部RC时钟

在文件 [lks32mc08x_sys.h](#) 第 64 行定义.

3.35.2.18 SYS_CLK_SEL_PLL

```
#define SYS_CLK_SEL_PLL 0x01
```

系统时钟选择，内部PLL时钟

在文件 [lks32mc08x_sys.h](#) 第 63 行定义.

3.35.2.19 SYS_Clk_SPIDiv1

```
#define SYS_Clk_SPIDiv1 0
```

SPI时钟分频，占时未使用

在文件 [lks32mc08x_sys.h](#) 第 70 行定义.

3.35.2.20 SYS_Clk_SPIDiv2

```
#define SYS_Clk_SPIDiv2 1
```

SPI时钟分频，占时未使用

在文件 [lks32mc08x_sys.h](#) 第 71 行定义.

3.35.2.21 SYS_Clk_SPIDiv4

```
#define SYS_Clk_SPIDiv4 2
```

SPI时钟分频，占时未使用

在文件 [lks32mc08x_sys.h](#) 第 72 行定义.

3.35.2.22 SYS_Clk_SPIDiv8

```
#define SYS_Clk_SPIDiv8 3
```

SPI时钟分频，占时未使用

在文件 [lks32mc08x_sys.h](#) 第 73 行定义.

3.35.2.23 SYS_Clk_UARTDiv1

```
#define SYS_Clk_UARTDiv1 0
```

UART时钟分频，占时未使用

在文件 [lks32mc08x_sys.h](#) 第 76 行定义.

3.35.2.24 SYS_Clk_UARTDiv2

```
#define SYS_Clk_UARTDiv2 1
```

UART时钟分频，占时未使用

在文件 [lks32mc08x_sys.h](#) 第 77 行定义.

3.35.2.25 SYS_Clk_UARTDiv4

```
#define SYS_Clk_UARTDiv4 2
```

UART时钟分频，占时未使用

在文件 [lks32mc08x_sys.h](#) 第 78 行定义.

3.35.2.26 SYS_Clk_UARTDiv8

```
#define SYS_Clk_UARTDiv8 3
```

UART时钟分频，占时未使用

在文件 [lks32mc08x_sys.h](#) 第 79 行定义.

3.35.2.27 SYS_Module_CAN

```
#define SYS_Module_CAN BIT7
```

CAN模块

在文件 [lks32mc08x_sys.h](#) 第 89 行定义.

3.35.2.28 SYS_Module_DSP

```
#define SYS_Module_DSP BIT6
```

DSP模块

在文件 [lks32mc08x_sys.h](#) 第 88 行定义.

3.35.2.29 SYS.Module_HALL

```
#define SYS.Module_HALL BIT1
```

HALL模块

在文件 [lks32mc08x_sys.h](#) 第 83 行定义.

3.35.2.30 SYS.Module_I2C

```
#define SYS.Module_I2C BIT0
```

I2C模块

在文件 [lks32mc08x_sys.h](#) 第 82 行定义.

3.35.2.31 SYS.Module_MCPWM

```
#define SYS.Module_MCPWM BIT3
```

MCPWM模块

在文件 [lks32mc08x_sys.h](#) 第 85 行定义.

3.35.2.32 SYS.Module_SPI

```
#define SYS.Module_SPI BIT8
```

SPI模块

在文件 [lks32mc08x_sys.h](#) 第 90 行定义.

3.35.2.33 SYS.Module_TIMER

```
#define SYS.Module_TIMER BIT2
```

TIMER模块

在文件 [lks32mc08x_sys.h](#) 第 84 行定义.

3.35.2.34 SYS.Module_UART0

```
#define SYS.Module_UART0 BIT4
```

UART0模块

在文件 [lks32mc08x_sys.h](#) 第 86 行定义.

3.35.2.35 SYS.Module_UART1

```
#define SYS.Module_UART1 BIT5
```

UART1模块

在文件 [lks32mc08x_sys.h](#) 第 87 行定义.

3.35.2.36 SYS.PLLREDIV_1

```
#define SYS.PLLREDIV_1 0x01
```

在PLL分频基础上不再分频

在文件 [lks32mc08x_sys.h](#) 第 67 行定义.

3.35.2.37 SYS.PLLREDIV_4

```
#define SYS.PLLREDIV_4 0x00
```

在PLL分频基础上再/4

在文件 [lks32mc08x_sys.h](#) 第 66 行定义.

3.35.2.38 SYS.PLLSRSEL_CRYSTAL

```
#define SYS.PLLSRSEL_CRYSTAL 0x01
```

在文件 [lks32mc08x_sys.h](#) 第 60 行定义.

3.35.2.39 SYS.PLLSRSEL_RCH

```
#define SYS.PLLSRSEL_RCH 0x00
```

在文件 [lks32mc08x_sys.h](#) 第 59 行定义.

3.35.2.40 SYS.RstSrc_HPOR

```
#define SYS.RstSrc_HPOR 0x02
```

复位信号源: 内部3.3V电源管理

在文件 [lks32mc08x_sys.h](#) 第 115 行定义.

3.35.2.41 SYS.RstSrc_KEYBOARD

```
#define SYS.RstSrc_KEYBOARD 0x04
```

复位信号源: 按键复位

在文件 [lks32mc08x_sys.h](#) 第 116 行定义.

3.35.2.42 SYS.RstSrc_LPOR

```
#define SYS.RstSrc_LPOR 0x01
```

复位信号源: 内部1.5V电源管理复位

在文件 [lks32mc08x_sys.h](#) 第 114 行定义.

3.35.2.43 SYS.RstSrc_WDT

```
#define SYS.RstSrc_WDT 0x08
```

复位信号源: WDT复位

在文件 [lks32mc08x_sys.h](#) 第 117 行定义.

3.35.2.44 SYS.WakeInterval_025s

```
#define SYS.WakeInterval_025s 0x00
```

休眠唤醒间隔时间0.25s

在文件 [lks32mc08x_sys.h](#) 第 120 行定义.

3.35.2.45 SYS.WakeInterval_05s

```
#define SYS.WakeInterval_05s 0x01
```

休眠唤醒间隔时间0.5s

在文件 [lks32mc08x_sys.h](#) 第 121 行定义.

3.35.2.46 SYS.WakeInterval_16s

```
#define SYS.WakeInterval_16s 0x06
```

休眠唤醒间隔时间16s

在文件 [lks32mc08x_sys.h](#) 第 126 行定义.

3.35.2.47 SYS.WakeInterval_1s

```
#define SYS.WakeInterval_1s 0x02
```

休眠唤醒间隔时间1s

在文件 [lks32mc08x_sys.h](#) 第 122 行定义.

3.35.2.48 SYS.WakeInterval_2s

```
#define SYS.WakeInterval_2s 0x03
```

休眠唤醒间隔时间2s

在文件 [lks32mc08x_sys.h](#) 第 123 行定义.

3.35.2.49 SYS.WakeInterval_32s

```
#define SYS.WakeInterval_32s 0x07
```

休眠唤醒间隔时间32s

在文件 [lks32mc08x_sys.h](#) 第 127 行定义.

3.35.2.50 SYS.WakeInterval_4s

```
#define SYS.WakeInterval_4s 0x04
```

休眠唤醒间隔时间4s

在文件 [lks32mc08x_sys.h](#) 第 124 行定义.

3.35.2.51 SYS.WakeInterval_8s

```
#define SYS.WakeInterval_8s 0x05
```

休眠唤醒间隔时间8s

在文件 [lks32mc08x_sys.h](#) 第 125 行定义.

3.35.2.52 SYS.WD_TimeOut2s

```
#define SYS.WD_TimeOut2s 0
```

看门狗超时时间2s复位

在文件 [lks32mc08x_sys.h](#) 第 108 行定义.

3.35.2.53 SYS.WD.TimeOut4s

```
#define SYS.WD.TimeOut4s 1
```

看门狗超时时间4s复位

在文件 [lks32mc08x_sys.h](#) 第 109 行定义.

3.35.2.54 SYS.WD.TimeOut64s

```
#define SYS.WD.TimeOut64s 3
```

看门狗超时时间64s复位

在文件 [lks32mc08x_sys.h](#) 第 111 行定义.

3.35.2.55 SYS.WD.TimeOut8s

```
#define SYS.WD.TimeOut8s 2
```

看门狗超时时间8s复位

在文件 [lks32mc08x_sys.h](#) 第 110 行定义.

3.35.2.56 WAKEIO.P0_0

```
#define WAKEIO.P0_0 BIT0
```

系统低功耗时使用P0.0作为唤醒IO

在文件 [lks32mc08x_sys.h](#) 第 130 行定义.

3.35.2.57 WAKEIO.P0_1

```
#define WAKEIO.P0_1 BIT1
```

系统低功耗时使用P0.1作为唤醒IO

在文件 [lks32mc08x_sys.h](#) 第 131 行定义.

3.35.2.58 WAKEIO.P1_0

```
#define WAKEIO.P1_0 BIT2
```

系统低功耗时使用P1.0作为唤醒IO

在文件 [lks32mc08x_sys.h](#) 第 132 行定义.

3.35.2.59 WAKEIO.P1_1

```
#define WAKEIO.P1_1 BIT3
```

系统低功耗时使用P1.1作为唤醒IO

在文件 [lks32mc08x_sys.h](#) 第 133 行定义.

3.35.3 枚举类型说明

3.35.3.1 enumWakeTrigLevel

```
enum enumWakeTrigLevel
```

低功耗IO唤醒电平枚举成员

枚举值

WAKETRIG_LOW	
WAKETRIG_HIGH	输入IO低电平唤醒

在文件 [lks32mc08x_sys.h](#) 第 22 行定义.

3.35.4 函数说明

3.35.4.1 SetWakeIO()

```
void SetWakeIO (
    uint32_t wakeIO,
    enumWakeTrigLevel trigLevel,
    uint8_t ena )
```

函数名称: void [SetWakeIO\(uint32_t wakeIO, enumWakeTrigLevel trigLevel, uint8_t ena\)](#)

功能描述: 设置休眠唤醒IO

参见

唤醒电平选择枚举: [enumWakeTrigLevel](#)

参数

输入参数:	wakeIO = WAKEIO_P1_1: P1.1 外部唤醒使能。1: 使能; 0: 禁用 wakeIO = WAKEIO_P1_0: P1.0 外部唤醒使能。1: 使能; 0: 禁用 wakeIO = WAKEIO_P0_1: P0.1 外部唤醒使能。1: 使能; 0: 禁用 wakeIO = WAKEIO_P0_0: P0.0 外部唤醒使能。1: 使能; 0: 禁用 enumWakeTrigLevel : 唤醒电平选择: 0:低电平唤醒 1:高电平唤醒 ena: ENABLE: 开启IO唤醒 DISABLE: 关闭IO唤醒
输出参数:	无

返回

返回 值: 无

注解

其它说明: 无

警告

无

示例代码:

```
SetWakeIO(WAKEIO_P0_0,WAKETRIG_LOW,ENABLE); //系统休眠时,使能P0.0作为低功耗唤醒IO.  
低电平唤醒
```

Date	Version	Author	Description
2022年04月14日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_sys.c](#) 第 297 行定义.

3.35.4.2 Switch2HRC()

```
void Switch2HRC (  
    void )
```

函数名称: void [Switch2HRC\(void\)](#)

功能描述: 休眠唤醒前一刻, 关闭高速时钟与BGP和掉电检测电路函数

参见

被调用函数: 无

参数

输入参数:	无
输出参数:	无

返回

返回值: 无

注解

其它说明: 系统低功耗休眠使用

警告

无

示例代码:

```
Switch2HRC();
```

Date	Version	Author	Description
2022年04月14日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_sys.c](#) 第 395 行定义.

3.35.4.3 Switch2PLL()

```
void Switch2PLL (  
    void )
```

函数名称: void [Switch2PLL\(void\)](#)

功能描述: 休眠唤醒后, 系统时钟切换到PLL时钟函数

参见

被调用函数: 无

参数

输入参数:	无
输出参数:	无

返回

返回 值: 无

注解

其它说明: 系统低功耗唤醒使用

警告

无

示例代码:

```
Switch2PLL();
```

Date	Version	Author	Description
2022年04月14日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_sys.c](#) 第 364 行定义.

函数调用图:



3.35.4.4 SYS_AnalogModuleClockCmd()

```
void SYS_AnalogModuleClockCmd (
    uint32_t nModule,
    FuncState state )
```

函数名称: void [SYS_AnalogModuleClockCmd\(uint32_t nModule, FuncState state\)](#)

功能描述: 模拟模块使能和停止

参见

被调用枚举: FuncState

参数

输入参数:	state: ENABLE使能, DISABLE关闭 nModule: 参数可选:		
	宏定义	说明	
	SYS_AnalogModule_ADC	BIT0	ADC模块
	SYS_AnalogModule_OPA0	BIT2	OPA0模块
	SYS_AnalogModule_OPA1	BIT3	OPA1模块
	SYS_AnalogModule_OPA2	BIT4	OPA2模块
	SYS_AnalogModule_OPA3	BIT5	OPA3模块
	SYS_AnalogModule_CMP0	BIT6	CMP0模块
	SYS_AnalogModule_CMP1	BIT7	CMP1模块
	SYS_AnalogModule_BGP	BIT8	BGP模块
	SYS_AnalogModule_RCH	BIT10	RCH模块
	SYS_AnalogModule_DAC	BIT12	DAC模块
	SYS_AnalogModule_TMP	BIT13	温度传感器
	SYS_AnalogModule_XTAL	BIT14	晶振
	SYS_AnalogModule_PLL	BIT15	PLL模块
输出参数:	无		

返回

返回 值: 无

注解

其它说明: 无

警告

无

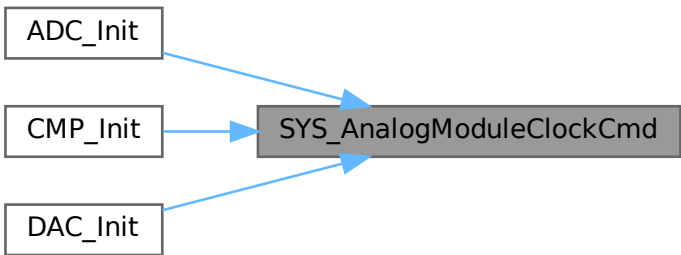
示例代码:

```
SYS_AnalogModuleClockCmd(SYS_AnalogModule_PLL,ENABLE); //使能PLL模块
```

修改日志:

在文件 [lks32mc08x_sys.c](#) 第 605 行定义.

这是这个函数的调用关系图:



3.35.4.5 SYS.ClearRst()

```
void SYS_ClearRst (
    void )
```

函数名称: void **SYS.ClearRst(void)**

功能描述: **SYS**清除复位标志记录

参见
被调用函数: 无

参数

输入参数:	无
输出参数:	无

返回
返回 值: 无

注解
其它说明: 请注意由于复位记录工作于低速时钟域，清除执行完成需要一定时间，不应清 除后立
即读记录状态

警告
无

示例代码:

```
u32 RSET_VALUE = 0;
SYS_ClearRst(); //清除复位标志记录
delay_ms(1);
RSET_VALUE = SYS_GetRstSource(); //读取复位源记录值
```

Date	Version	Author	Description
2016年07月02日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_sys.c](#) 第 139 行定义.

3.35.4.6 SYS_FallSleep()

```
void SYS_FallSleep (
    void )
```

函数名称: void [SYS_FallSleep\(void\)](#)

功能描述: 触发系统进入休眠状态

参见

被调用函数: 无

参数

输入参数:	无
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
SYS\_FallSleep\(\); //触发系统进入休眠状态
```


Date	Version	Author	Description
2016年07月02日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_sys.c](#) 第 197 行定义.

3.35.4.7 SYS_FeedDog()

```
void SYS_FeedDog (
    uint32_t timeOut )
```

函数名称: void [SYS_FeedDog\(uint32_t timeOut\)](#)

功能描述: 喂狗

参见

被调用函数: 无

参数

输入参数:	timeOut: 看门狗超时时间		
	SYS_WD_TimeOut2s	0	2s复位
	SYS_WD_TimeOut4s	1	4s复位
	SYS_WD_TimeOut8s	2	8s复位
	SYS_WD_TimeOut64s	3	64s复位
输出参数:	无		

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
SYS\_FeedDog (SYS\_WD\_TimeOut2s) ; //看门狗喂狗且设置下一次如果2s不喂狗则芯片复位
```

Date	Version	Author	Description
2016年06月28日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_sys.c](#) 第 495 行定义.

3.35.4.8 SYS_GetRstSource()

```
uint32_t SYS_GetRstSource (
    void )
```

函数名称: uint32_t [SYS_GetRstSource\(void\)](#)

功能描述: 获得SYS复位源信号

参见

被调用函数: 无

参数

输入参数:	无
输出参数:	无

返回

返回 值: 复位源记录值

注解

其它说明: 返回值位介绍

警告

无

BIT3	看门狗复位发生标志	高有效
BIT2	按键复位发生标志	高有效
BIT1	HPOR 复位发生标志	高有效
BIT0	LPOR 复位发生标志	高有效

示例代码:

```
u32 RSET_VALUE = 0;
RSET_VALUE = SYS\_GetRstSource\(\); // 读取复位源记录值
如 RSET_VALUE = 0x01; 则LPOR发生复位
```

Date	Version	Author	Description
2016年07月02日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_sys.c](#) 第 173 行定义.

3.35.4.9 SYS_Init()

```
void SYS_Init (
    SYS_InitTypeDef * SYS_InitStruct )
```

函数名称: void [SYS_Init\(SYS_InitTypeDef* SYS_InitStruct\)](#)

功能描述: **SYS**模块初始化函数

参见

被调用函数: 无

参数

输入参数:	SYS_InitTypeDef* SYS_InitStruct
输出参数:	无

返回

返回值: 无

注解

其它说明: 一般不使用该函数, 因为涉及模块较多, 比较杂乱, 对应里面的配置在各模块 初始化单独配置。

警告

无

示例代码:

```
SYS_InitTypeDef SYS_InitStruct;
SYS_StructInit (&SYS_InitStruct);
SYS_InitStruct.PLL_SrcSel = SYS_PLLSRSEL_RCH; // 使用内部4MHz晶振作为时钟输入源
SYS_Init (&SYS_InitStruct);
```

Date	Version	Author	Description
2016年07月02日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_sys.c](#) 第 45 行定义.

3.35.4.10 SYS.ModuleClockCmd()

```
void SYS_ModuleClockCmd (
    uint32_t nModule,
    FuncState state )
```

函数名称: void [SYS.ModuleClockCmd\(uint32_t nModule, FuncState state\)](#)

功能描述: 数字模块时钟使能和停止

参见

被调用枚举: FuncState

参数

输入参数:	state: ENABLE使能 , DISABLE关闭 nModule: 参数可选:		
	宏定义	原始值	说明
	SYS_Module_I2C	BIT0	I2C模块
	SYS_Module_HALL	BIT1	HALL模块
	SYS_Module_TIMER	BIT2	UTIMER模块
	SYS_Module_MCPWM	BIT3	MCPWM模块
	SYS_Module_UART0	BIT4	UART0模块
	SYS_Module_UART1	BIT5	UART1模块
	SYS_Module_DSP	BIT6	DSP模块
	SYS_Module_CAN	BIT7	CAN模块
	SYS_Module_SPI	BIT8	SPI模块
输出参数:	无		

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

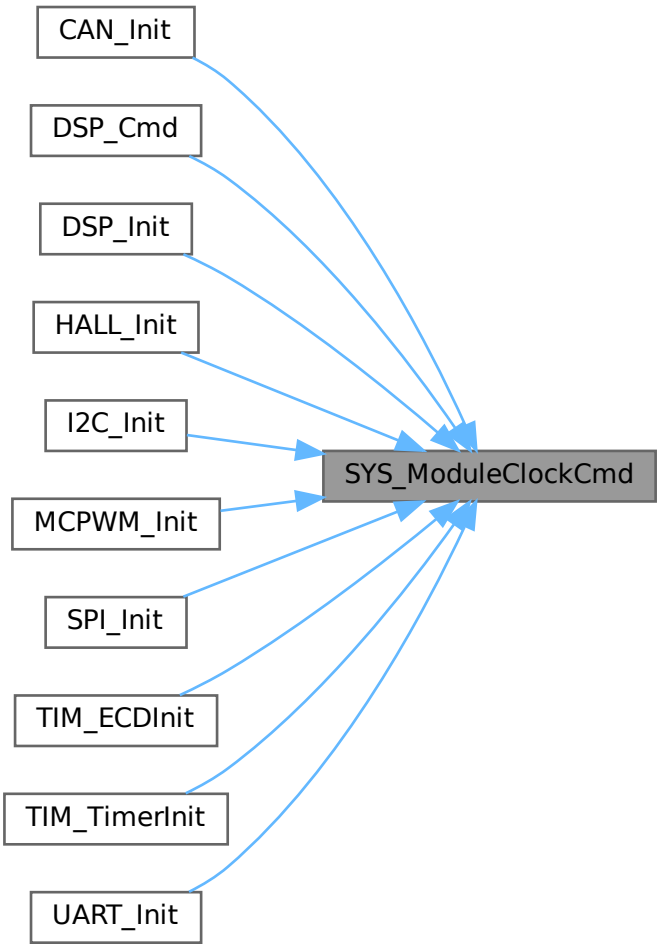
```
SYS_ModuleClockCmd(SYS_Module_SPI, ENABLE); //使能SPI时钟
```

Date	Version	Author	Description
2016年06月28日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x-sys.c](#) 第 552 行定义.

这是这个函数的调用关系图:



3.35.4.11 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）

3.35.4.12 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

3.35.4.13 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

3.35.4.14 SYS.SetWakeInterval()

```
void SYS_SetWakeInterval (
    uint32_t nInterval )
```

函数名称: void [SYS.SetWakeInterval\(uint32_t nInterval\)](#)

功能描述: 设置系统休眠唤醒周期

参见

被调用函数: 无

参数

输入参数:	nInterval: 唤醒时间		
	宏定义	原始值	说明
	SYS_WakeInterval_025s	0x00	0.25s
	SYS_WakeInterval_05s	0x01	0.5s
	SYS_WakeInterval_1s	0x02	1s
	SYS_WakeInterval_2s	0x03	2s
	SYS_WakeInterval_4s	0x04	4s
	SYS_WakeInterval_8s	0x05	8s
	SYS_WakeInterval_16s	0x06	16s
	SYS_WakeInterval_32s	0x07	32s
输出参数:	无		

返回

返回 值: 无

注解

其它说明: 无

警告

无

示例代码:

```
SYS_SetWakeInterval(SYS_WakeInterval_1s); //系统休眠时,设置系统在1s后唤醒
```

Date	Version	Author	Description
2016年07月02日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_sys.c](#) 第 256 行定义.

3.35.4.15 SYS_SoftResetModule()

```
void SYS_SoftResetModule (
    uint32_t nModule )
```

函数名称: void [SYS_SoftResetModule\(uint32_t nModule\)](#)

功能描述: 数字模块软复位

参见

被调用枚举： 无

参数

输入参数:	state: ENABLE使能 , DISABLE关闭 nModule: 参数可选:		
	宏定义	原始值	说明
	SYS_Module_I2C	BIT0	I2C模块
	SYS_Module_HALL	BIT1	HALL模块
	SYS_Module_TIMER	BIT2	UTIMER模块
	SYS_Module_MCPWM	BIT3	MCPWM模块
	SYS_Module_UART0	BIT4	UART0模块
	SYS_Module_UART1	BIT5	UART1模块
	SYS_Module_DSP	BIT6	DSP模块
	SYS_Module_CAN	BIT7	CAN模块
	SYS_Module_SPI	BIT8	SPI模块
输出参数:	无		

返回

返回 值: 无

注解

其它说明： 无

警告

无

示例代码:

```
//软复位SPI模块，清除SPI寄存器配置，PC指针复位
SYS_SoftResetModule(SYS_Module_SPI);
```

Date	Version	Author	Description
2016年06月28日	1.0	cfwu	创建

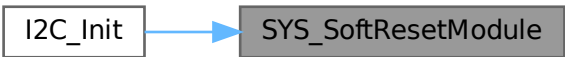
修改日志:

Date	Version	Author	Description
2022年04月14日	1.1	HuangMG	修改

修改日志:增加延时函数，等待寄存器复位完毕与释放复位操作。

在文件 [lks32mc08x_sys.c](#) 第 659 行定义.

这是这个函数的调用关系图:



3.35.4.16 SYS_StructInit()

```
void SYS_StructInit (
    SYS_InitTypeDef * SYS_InitStruct )
```

函数名称: void [SYS_StructInit\(SYS_InitTypeDef* SYS_InitStruct\)](#)

功能描述: SYS结构体初始化

参见

被调用函数: 无

参数

输入参数:	SYS_InitTypeDef* SYS_InitStruct
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
SYS_InitTypeDef SYS_InitStruct;  
SYS_StructInit(&SYS_InitStruct);
```

Date	Version	Author	Description
2016年07月02日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_sys.c](#) 第 101 行定义.

3.35.4.17 SYS_VolSelModule()

```
void SYS_VolSelModule (  
    uint32_t Vol )
```

3.35.4.18 SYS_VolSelModuleEnableIRQ()

```
void SYS_VolSelModuleEnableIRQ (  
    uint32_t Vol )
```

3.35.4.19 SYS_VolSelModuleIRQ()

```
void SYS_VolSelModuleIRQ (  
    void )
```

3.35.4.20 SYS_WatchDogCmd()

```
void SYS_WatchDogCmd (  
    FuncState state )
```

函数名称: void [SYS.WatchDogCmd\(FuncState state\)](#)

功能描述: 看门狗模块使能函数

参见

被调用函数: 无

参数

输入参数:	state: ENABLE使能看门狗/DISABLE关闭开门狗
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
SYS_WatchDogCmd(ENABLE); //使能看门狗模块
```

Date	Version	Author	Description
2016年06月28日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_sys.c](#) 第 427 行定义.

3.36 lks32mc08x_sys.h

[浏览该文件的文档.](#)

```
00001
00015 #ifndef __LKS32MC08x_SYS_H
00016 #define __LKS32MC08x_SYS_H
00017
00018 /* Includes ----- */
00019 #include "lks32mc08x_lib.h"
00020
00022 typedef enum
00023 {
00024     WAKETRIG_LOW,
00025     WAKETRIG_HIGH,
00026 } enumWakeTrigLevel;
00027
00031 typedef struct
00032 {
00033     uint32_t AFE_REG5_RECORD;
00034     uint32_t AFE_REG6_RECORD;
00035     uint32_t CLK_CFG_RECORD;
00036 } WAKE_InitTypeDef;
00037
00041 typedef struct
00042 {
00043     uint32_t PLL_SrcSel;
00045     uint32_t PLL_DivSel;
00046     uint32_t Clk_Sel;
00048     uint32_t Clk_DivSPI;
00049     uint32_t Clk_DivUART;
00050     uint32_t Clk_FEN;
00052     uint32_t WDT_Ena;
00053     uint32_t PORFilter_Ena;
00055     uint32_t PLL_ReDiv;
00056 } SYS_InitTypeDef;
00057
```

```
00058 /*PLL信号输入源选择*/
00059 #define SYS.PLLSRSEL_RCH 0x00
00060 #define SYS.PLLSRSEL_CRYSTAL 0x01
00061
00062 /*系统时钟选择*/
00063 #define SYS.CLK_SEL_PLL 0x01
00064 #define SYS.CLK_SEL_CLK_HS 0x00
00065 /*以弃用*/
00066 #define SYS.PLLREDIV_4 0x00
00067 #define SYS.PLLREDIV_1 0x01
00069 /*SPI时钟分频*/
00070 #define SYS.Clk_SPIDiv1 0
00071 #define SYS.Clk_SPIDiv2 1
00072 #define SYS.Clk_SPIDiv4 2
00073 #define SYS.Clk_SPIDiv8 3
00075 /*UART时钟分频*/
00076 #define SYS.Clk_UARTDiv1 0
00077 #define SYS.Clk_UARTDiv2 1
00078 #define SYS.Clk_UARTDiv4 2
00079 #define SYS.Clk_UARTDiv8 3
00081 /*数字模块位定义*/
00082 #define SYS.Module_I2C BIT0
00083 #define SYS.Module_HALL BIT1
00084 #define SYS.Module_TIMER BIT2
00085 #define SYS.Module_MCPWM BIT3
00086 #define SYS.Module_UART0 BIT4
00087 #define SYS.Module_UART1 BIT5
00088 #define SYS.Module_DSP BIT6
00089 #define SYS.Module_CAN BIT7
00090 #define SYS.Module_SPI BIT8
00092 /*模拟模块定义*/
00093 #define SYS.AnalogModule_ADC BIT0
00094 #define SYS.AnalogModule_OPA0 BIT2
00095 #define SYS.AnalogModule_OPA1 BIT3
00096 #define SYS.AnalogModule_OPA2 BIT4
00097 #define SYS.AnalogModule_OPA3 BIT5
00098 #define SYS.AnalogModule_CMP0 BIT6
00099 #define SYS.AnalogModule_CMP1 BIT7
00100 #define SYS.AnalogModule_BGP BIT8
00101 #define SYS.AnalogModule_RCH BIT10
00102 #define SYS.AnalogModule_DAC BIT12
00103 #define SYS.AnalogModule_TMP BIT13
00104 #define SYS.AnalogModule_XTAL BIT14
00105 #define SYS.AnalogModule_PLL BIT15
00107 /*看门狗超时时间*/
00108 #define SYS.WD.TimeOut2s 0
00109 #define SYS.WD.TimeOut4s 1
00110 #define SYS.WD.TimeOut8s 2
00111 #define SYS.WD.TimeOut64s 3
00113 /*复位信号源定义*/
00114 #define SYS.RstSrc_LPOR 0x01
00115 #define SYS.RstSrc_HPOR 0x02
00116 #define SYS.RstSrc_KEYBOARD 0x04
00117 #define SYS.RstSrc_WDT 0x08
00119 /*休眠唤醒间隔时间*/
00120 #define SYS.WakeInterval_025s 0x00
00121 #define SYS.WakeInterval_05s 0x01
00122 #define SYS.WakeInterval_1s 0x02
00123 #define SYS.WakeInterval_2s 0x03
00124 #define SYS.WakeInterval_4s 0x04
00125 #define SYS.WakeInterval_8s 0x05
00126 #define SYS.WakeInterval_16s 0x06
00127 #define SYS.WakeInterval_32s 0x07
00129 /*低功耗唤醒IO选择*/
00130 #define WAKEIO_P0_0 BIT0
00131 #define WAKEIO_P0_1 BIT1
00132 #define WAKEIO_P1_0 BIT2
00133 #define WAKEIO_P1_1 BIT3
00135 #define MCU.POWER_3v3 0 // MCU 3.3V供电
00136 #define MCU.POWER_5v0 1 // MCU 5V供电
00137
00138 #define POWER_MODE MCU_POWER_5v0
00139
00140 #ifndef POWER_4V5
00141 #define POWER_4V5 0x00 /*电源掉电监测阈值为4.5V*/
00142 #endif
00143 #ifndef POWER_4V2
00144 #define POWER_4V2 0x01 /*电源掉电监测阈值为4.2V*/
00145 #endif
00146 #ifndef POWER_3V9
00147 #define POWER_3V9 0x02 /*电源掉电监测阈值为3.9V*/
00148 #endif
00149 #ifndef POWER_3V6
00150 #define POWER_3V6 0x03 /*电源掉电监测阈值为3.6V*/
00151 #endif
00152
00153 /*系统设置*/
```

```

00154 void SYS_InitTypeDef (SYS_InitTypeDef *SYS_InitStruct);
00155 void SYS_StructInit (SYS_InitTypeDef *SYS_InitStruct);
00156
00157 /*复位源*/
00158 uint32_t SYS_GetRstSource (void);
00159 void SYS_ClearRst (void);
00160
00161 /*看门狗*/
00162 void SYS_WatchDogCmd (FuncState state);
00163 void SYS_FeedDog (uint32_t timeOut);
00164
00165 /*POR滤波*/
00166 // void SYS_PORFilterCmd (FuncState state);
00167
00168 /*休眠、唤醒、时钟切换*/
00169 void SYS_FallSleep (void);
00170 void SYS_SetWakeInterval (uint32_t nInterval);
00171 void Switch2PLL (void);
00172 void Switch2HRC (void);
00173 void SetWakeIO (uint32_t wakeIO, enumWakeTrigLevel trigLevel, uint8_t ena);
00174
00175 /*控制8路时钟使能*/
00176 void SYS_ModuleClockCmd (uint32_t nModule, FuncState state);
00177
00178 /*控制模拟模块时钟使能*/
00179 void SYS_AnalogModuleClockCmd (uint32_t nModule, FuncState state);
00180
00181 /*软复位各模块*/
00182 void SYS_SoftResetModule (uint32_t nModule);
00183
00184 /*IAP_MODE*/
00185 // 允许进入IAP_MODE, 在软复位以后
00186
00187 /* MCU供电电压稳定判断函数*/
00188 void SYS_VolSelModule (uint32_t Vol);
00189
00190 /* MCU供电电压稳定判断中断函数*/
00191 void SYS_VolSelModuleIRQ (void);
00192
00193 /* MCU供电电压稳定判断中断使能函数*/
00194 void SYS_VolSelModuleEnableIRQ (uint32_t Vol);
00204 void SYS_ResetPeripheral (void);
00214 void SYS_ResetPeripheralInternal (void);
00218 void SYS_ResetIO (void);
00219 #endif /*__lks32mc08x.SYS_H */
00220
00221 /***** (C) COPYRIGHT LINKO SEMICONDUCTOR *****END OF FILE*****/

```

3.37 include/lks32mc08x_tim.h 文件参考

文件名称: lks32mc08x_tim.h
文件标识: 无
内容摘要: TIM驱动头文件
其它说明: 无

#include "lks32mc08x-lib.h"
lks32mc08x_tim.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 [TIM_TimerTypeDef](#)
定时器寄存器结构体句柄
- struct [TIM_ECDTypeDef](#)
编码器寄存器结构体句柄
- struct [TIM_TimerInitTypeDef](#)
- struct [TIM_ECDInitTypeDef](#)

宏定义

- #define [TIM_Clk_Div1](#) 0x00
*TIMER0*结构体基地址定义
- #define [TIM_Clk_Div2](#) 0x01
*Timer*工作频率48M
- #define [TIM_Clk_Div4](#) 0x02
*Timer*工作频率24M
- #define [TIM_Clk_Div8](#) 0x03
*Timer*工作频率12M
- #define [ECD_Clk_Div1](#) 0x00
*ECD*工作频率96M
- #define [ECD_Clk_Div2](#) 0x01
*ECD*工作频率48M
- #define [ECD_Clk_Div4](#) 0x02
*ECD*工作频率24M
- #define [ECD_Clk_Div8](#) 0x03
*ECD*工作频率12M
- #define [TIMER_IF_CH1](#) BIT2
中断标志定义,*Timer CH1*中断, 比较或捕获中断
- #define [TIMER_IF_CH0](#) BIT1
中断标志定义,*Timer CH0*中断, 比较或捕获中断
- #define [TIMER_IF_ZERO](#) BIT0
中断标志定义,*Timer* 过零中断
- #define [ECD_IRQEna_None](#) 0
编码器中断使能,配置定义无使能
- #define [ECD_IRQEna_UnderFlow](#) BIT0
编码器中断使能,使能下溢中断
- #define [ECD_IRQEna_OverFlow](#) BIT1
编码器中断使能,使能上溢中断
- #define [ECD_IF_UNDERFLOW](#) BIT0
编码器中断标志定义,*Encoder*下溢中断
- #define [ECD_IF_OVERFLOW](#) BIT1
编码器中断标志定义,*Encoder*上溢中断
- #define [TIMER_CapMode_None](#) ((uint32_t)0x00000000)
捕获模式定义,无捕获
- #define [TIMER_CapMode_Rise](#) ((uint32_t)0x00000001)
捕获模式定义,上升沿捕获
- #define [TIMER_CapMode_Fall](#) ((uint32_t)0x00000002)
捕获模式定义,下降沿捕获
- #define [TIMER_CapMode_RiseFall](#) ((uint32_t)0x00000003)

- 捕获模式定义,双沿捕获
- #define **TIMER_OPMode_CAP** ((uint32_t)0x0000001)
工作模式,捕获模式
- #define **TIMER_OPMode_CMP** ((uint32_t)0x0000000)
工作模式,比较模式
- #define **ECD_Mode_T1** ((uint32_t)0x00)
counting on T1
- #define **ECD_Mode_T1_T2** ((uint32_t)0x01)
counting on T1&T2
- #define **ECD_Mode_CCWSIGN** ((uint32_t)0x02)
CCW+SIGN, 仅计数信号上升沿
- #define **ECD_Mode_CCWCW** ((uint32_t)0x03)
CCW+CW, 仅计数信号上升沿
- #define **ECD_Mode_CCWSIGN_UpDown** ((uint32_t)0x06)
符号加脉冲信号计数模式, 信号的上下沿均计数
- #define **ECD_Mode_CCWCW_UpDown** ((uint32_t)0x07)
CCW+CW 双脉冲信号计数模式, 信号的上下沿均计数

枚举

- enum **Timer_IRQEnaDef** {
 Timer_IRQEna_None = 0 , **Timer_IRQEna_Zero** = BIT0 , **Timer_IRQEna_CH0** = BIT1 , **Timer_IRQEna_CH1** = BIT2 ,
 Timer_IRQEna_All = 0x07 }
中断使能配置定义

函数

- void **TIM_TimerInit** (**TIM_TimerTypeDef** *TIMERx, **TIM_TimerInitTypeDef** *TIM_TimerInitStruct)
函数名称: void **TIM_TimerInit**(**TIM_TimerTypeDef** *TIMERx, **TIM_TimerInitTypeDef** *TIM_TimerInitStruct)
- void **TIM_TimerStrutInit** (**TIM_TimerInitTypeDef** *TIM_TimerInitStruct)
函数名称: void **TIM_TimerStrutInit**(**TIM_TimerInitTypeDef** *TIM_TimerInitStruct)
- void **TIM_TimerCmd** (**TIM_TimerTypeDef** *TIMERx, FuncState state)
函数名称: void **TIM_TimerCmd**(**TIM_TimerTypeDef** *TIMERx, **FuncState** state)
- uint32_t **TIM_GetIRQFlag** (**TIM_TimerTypeDef** *TIMERx, u32 timer_if)
函数名称: uint32_t **TIM_GetIRQFlag**(**TIM_TimerTypeDef** *TIMERx, u32 timer_if)
- void **TIM_ClearIRQFlag** (**TIM_TimerTypeDef** *TIMERx, uint32_t tempFlag)
函数名称: void **TIM_ClearIRQFlag**(**TIM_TimerTypeDef** *TIMERx, uint32_t tempFlag)
- uint32_t **TIM_Timer_GetCount** (**TIM_TimerTypeDef** *TIM_TIMERx)
函数名称: uint32_t **TIM_Timer_GetCount**(**TIM_TimerTypeDef** *TIM_TIMERx)
- uint32_t **TIM_Timer_GetCMPT0** (**TIM_TimerTypeDef** *TIM_TIMERx)
函数名称: uint32_t **TIM_Timer_GetCMPT0**(**TIM_TimerTypeDef** *TIM_TIMERx)
- uint32_t **TIM_Timer_GetCMPT1** (**TIM_TimerTypeDef** *TIM_TIMERx)
函数名称: uint32_t **TIM_Timer_GetCMPT1**(**TIM_TimerTypeDef** *TIM_TIMERx)
- void **TIM_ECDInit** (**TIM_ECDTypeDef** *ECDx, **TIM_ECDInitTypeDef** *TIM_ECDInitStruct)
函数名称: void **TIM_ECDInit**(**TIM_ECDTypeDef** *ECDx, **TIM_ECDInitTypeDef** *TIM_ECDInitStruct)
- void **TIM_ECDStructInit** (**TIM_ECDInitTypeDef** *TIM_ECDInitStruct)
函数名称: void **TIM_ECDInit**(**TIM_ECDTypeDef** *ECDx, **TIM_ECDInitTypeDef** *TIM_ECDInitStruct)
- void **TIM_ECDCmd** (**TIM_ECDTypeDef** *ECDx, FuncState state)
函数名称: void **TIM_ECDCmd**(**TIM_ECDTypeDef** *ECDx, **FuncState** state)

- `uint32_t TIM_ECD_GetCount (TIM_ECDTypeDef *TIM_ECDx)`
函数名称: `uint32_t TIM_ECD_GetCount(TIM_ECDTypeDef *TIM_ECDx)`
- `uint32_t ECD_GetIRQFlag (TIM_ECDTypeDef *ECDx, u32 timer_if)`
函数名称: `uint32_t ECD_GetIRQFlag(TIM_ECDTypeDef *ECDx , u32 timer_if)`
- `void ECD_ClearIRQFlag (TIM_ECDTypeDef *ECDx, uint32_t tempFlag)`
函数名称: `void ECD_ClearIRQFlag(TIM_ECDTypeDef *ECDx , uint32_t tempFlag)`

3.37.1 详细描述

文件名称: `lks32mc08x_tim.h`
文件标识: 无
内容摘要: TIM驱动头文件
其它说明: 无

版权所有
(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2016年06月06日	1.0	cfwu	创建

修改日志:

在文件 `lks32mc08x_tim.h` 中定义.

3.37.2 宏定义说明

3.37.2.1 ECD_Clk_Div1

`#define ECD_Clk_Div1 0x00`

ECD工作频率96M

在文件 `lks32mc08x_tim.h` 第 85 行定义.

3.37.2.2 ECD_Clk_Div2

`#define ECD_Clk_Div2 0x01`

ECD工作频率48M

在文件 `lks32mc08x_tim.h` 第 86 行定义.

3.37.2.3 ECD_Clk_Div4

```
#define ECD_Clk_Div4 0x02
```

ECD工作频率24M

在文件 [lks32mc08x_tim.h](#) 第 87 行定义.

3.37.2.4 ECD_Clk_Div8

```
#define ECD_Clk_Div8 0x03
```

ECD工作频率12M

在文件 [lks32mc08x_tim.h](#) 第 88 行定义.

3.37.2.5 ECD_IF_OVERFLOW

```
#define ECD_IF_OVERFLOW BIT1
```

编码器中断标志定义,Encoder上溢中断

在文件 [lks32mc08x_tim.h](#) 第 111 行定义.

3.37.2.6 ECD_IF_UNDERFLOW

```
#define ECD_IF_UNDERFLOW BIT0
```

编码器中断标志定义,Encoder下溢中断

在文件 [lks32mc08x_tim.h](#) 第 110 行定义.

3.37.2.7 ECD_IRQEna_None

```
#define ECD_IRQEna_None 0
```

编码器中断使能,配置定义无使能

在文件 [lks32mc08x_tim.h](#) 第 106 行定义.

3.37.2.8 ECD_IRQEna_OverFlow

```
#define ECD_IRQEna_OverFlow BIT1
```

编码器中断使能,使能上溢中断

在文件 [lks32mc08x_tim.h](#) 第 108 行定义.

3.37.2.9 ECD_IRQEna_UnderFlow

```
#define ECD_IRQEna_UnderFlow BIT0
```

编码器中断使能,使能下溢中断

在文件 [lks32mc08x_tim.h](#) 第 107 行定义.

3.37.2.10 ECD_Mode_CCWCW

```
#define ECD_Mode_CCWCW ((uint32_t)0x03)
```

CCW+CW, 仅计数信号上升沿

在文件 [lks32mc08x_tim.h](#) 第 124 行定义.

3.37.2.11 ECD_Mode_CCWCW_UpDown

```
#define ECD_Mode_CCWCW_UpDown ((uint32_t)0x07)
```

CCW+CW 双脉冲信号计数模式, 信号的上下沿均计数

在文件 [lks32mc08x_tim.h](#) 第 126 行定义.

3.37.2.12 ECD_Mode_CCWSIGN

```
#define ECD_Mode_CCWSIGN ((uint32_t)0x02)
```

CCW+SIGN, 仅计数信号上升沿

在文件 [lks32mc08x_tim.h](#) 第 123 行定义.

3.37.2.13 ECD_Mode_CCWSIGN_UpDown

```
#define ECD_Mode_CCWSIGN_UpDown ((uint32_t)0x06)
```

符号加脉冲信号计数模式, 信号的上下沿均计数

在文件 [lks32mc08x_tim.h](#) 第 125 行定义.

3.37.2.14 ECD_Mode_T1

```
#define ECD_Mode_T1 ((uint32_t)0x00)
```

counting on T1

在文件 [lks32mc08x_tim.h](#) 第 121 行定义.

3.37.2.15 ECD_Mode_T1_T2

```
#define ECD_Mode_T1_T2 ((uint32_t)0x01)
```

counting on T1&T2

在文件 [lks32mc08x.tim.h](#) 第 122 行定义.

3.37.2.16 TIM_Clk_Div1

```
#define TIM_Clk_Div1 0x00
```

TIMER0结构体基地址定义

TIMER1结构体基地址定义 TIMER2结构体基地址定义 TIMER3结构体基地址定义 ECD0结构体基地址定义 ECD1结构体基地址定义 Timer工作频率96M

在文件 [lks32mc08x.tim.h](#) 第 80 行定义.

3.37.2.17 TIM_Clk_Div2

```
#define TIM_Clk_Div2 0x01
```

Timer工作频率48M

在文件 [lks32mc08x.tim.h](#) 第 81 行定义.

3.37.2.18 TIM_Clk_Div4

```
#define TIM_Clk_Div4 0x02
```

Timer工作频率24M

在文件 [lks32mc08x.tim.h](#) 第 82 行定义.

3.37.2.19 TIM_Clk_Div8

```
#define TIM_Clk_Div8 0x03
```

Timer工作频率12M

在文件 [lks32mc08x.tim.h](#) 第 83 行定义.

3.37.2.20 TIMER_CapMode_Fall

```
#define TIMER_CapMode_Fall ((uint32_t)0x00000002)
```

捕获模式定义,下降沿捕获

在文件 [lks32mc08x.tim.h](#) 第 115 行定义.

3.37.2.21 TIMER_CapMode_None

```
#define TIMER_CapMode_None ((uint32_t)0x00000000)
```

捕获模式定义,无捕获

在文件 [lks32mc08x.tim.h](#) 第 113 行定义.

3.37.2.22 TIMER_CapMode_Rise

```
#define TIMER_CapMode_Rise ((uint32_t)0x00000001)
```

捕获模式定义,上升沿捕获

在文件 [lks32mc08x.tim.h](#) 第 114 行定义.

3.37.2.23 TIMER_CapMode_RiseFall

```
#define TIMER_CapMode_RiseFall ((uint32_t)0x00000003)
```

捕获模式定义,双沿捕获

在文件 [lks32mc08x.tim.h](#) 第 116 行定义.

3.37.2.24 TIMER_IF_CH0

```
#define TIMER_IF_CH0 BIT1
```

中断标志定义,Timer CH0中断, 比较或捕获中断

在文件 [lks32mc08x.tim.h](#) 第 103 行定义.

3.37.2.25 TIMER_IF_CH1

```
#define TIMER_IF_CH1 BIT2
```

中断标志定义,Timer CH1中断, 比较或捕获中断

在文件 [lks32mc08x.tim.h](#) 第 102 行定义.

3.37.2.26 TIMER_IF_ZERO

```
#define TIMER_IF_ZERO BIT0
```

中断标志定义,Timer 过零中断

在文件 [lks32mc08x.tim.h](#) 第 104 行定义.

3.37.2.27 TIMER_OPMODE_CAP

```
#define TIMER_OPMODE_CAP ((uint32_t)0x00000001)
```

工作模式,捕获模式

在文件 [lks32mc08x.tim.h](#) 第 118 行定义.

3.37.2.28 TIMER_OPMODE_CMP

```
#define TIMER_OPMODE_CMP ((uint32_t)0x00000000)
```

工作模式,比较模式

在文件 [lks32mc08x.tim.h](#) 第 119 行定义.

3.37.3 枚举类型说明

3.37.3.1 Timer_IRQEnaDef

```
enum Timer_IRQEnaDef
```

中断使能配置定义

枚举值

Timer_IRQEna_None	
Timer_IRQEna_Zero	使能过零中断
Timer_IRQEna_CH0	使能CH0中断，含比较、捕获中断
Timer_IRQEna_CH1	使能CH1中断，含比较、捕获中断
Timer_IRQEna_All	使能Timer全部中断

在文件 [lks32mc08x.tim.h](#) 第 93 行定义.

3.37.4 函数说明

3.37.4.1 ECD_ClearIRQFlag()

```
void ECD_ClearIRQFlag (
    TIM_ECDTypeDef * ECDx,
    uint32_t tempFlag )
```

函数名称: void [ECD_ClearIRQFlag\(TIM_ECDTypeDef *ECDx , uint32_t tempFlag\)](#)

功能描述: 清除ECD中断标志

参见

被引用内容: 无

参数

输入参数:	ECDx: ECD0/ECD1	
	timer_if:	
	ECD_IF_UNDERFLOW ECD_IF_OVERFLOW	Encoder下溢中断 Encoder上溢中断
输出参数:	无	

返回

返回 值: 无

注解

其它说明: 无

警告

无

示例代码:

```
if (ECD_GetIRQFlag (ECD0, ECD_IF_OVERFLOW)) //获取编码器上溢中断标志
{
    ECD_ClearIRQFlag (ECD0, ECD_IF_OVERFLOW); //清除编码器上溢中断标志
}
```

Date	Version	Author	Description
2022年04月13日	1.0	HuangMG	修改

修改日志:

在文件 [lks32mc08x_tim.c](#) 第 711 行定义.

3.37.4.2 ECD_GetIRQFlag()

```
uint32_t ECD_GetIRQFlag (
    TIM_ECDTypeDef * ECDx,
    u32 timer_if )
```

函数名称: `uint32_t ECD_GetIRQFlag(TIM_ECDTypeDef *ECDx , u32 timer_if)`

功能描述: 获取ECD中断标志

参见

被引用内容: 无

参数

输入参数:	ECDx: ECD0/ECD1		
	timer_if:		
	ECD_IF_UNDERFLOW	Encoder下溢中断	
	ECD_IF_OVERFLOW	Encoder上溢中断	
输出参数:	无		

返回

返回 值: 无

注解

其它说明: 无

警告

只有对应中断使能后，改为才能读取，如果对应中断未使能，读取结果一直为0

示例代码:

```
if (ECD_GetIRQFlag(ECD0, ECD_IF_OVERFLOW)) //获取上溢中断标志
{
}
```

Date	Version	Author	Description
2022年04月13日	1.0	HuangMG	修改

修改日志:

在文件 [lks32mc08x_tim.c](#) 第 645 行定义.

3.37.4.3 TIM_ClearIRQFlag()

```
void TIM_ClearIRQFlag (
    TIM_TimerTypeDef * TIMERx,
    uint32_t tempFlag )
```

函数名称: void [TIM_ClearIRQFlag\(TIM_TimerTypeDef *TIMERx ,uint32_t tempFlag\)](#)

功能描述: 清除TIM中断标志

参见

被引用内容: 无

参数

输入参数:	TIMERx: TIMER0/TIMER1/TIMER2/TIMER3	
	timer_if:	
	TIMER_IF_CH1	Timer CH1中断, 比较或捕获中断
	TIMER_IF_CH0	Timer CH0中断, 比较或捕获中断
	TIMER_IF_ZERO	Timer 过零中断
输出参数:	无	

返回

返回 值: 无

注解

其它说明: 无

警告

无

示例代码:

```
if(TIM_GetIRQFlag(TIMER0,TIMER_IF_CH0)) //判断UTimer0的CH0是否发生比较中断
{
    TIM_ClearIRQFlag(TIMER0,TIMER_IF_CH0); //清除UTimer0通道0比较中断标志位
}
```

Date	Version	Author	Description
2022年04月13日	1.0	HuangMG	修改

修改日志:

在文件 [lks32mc08x_tim.c](#) 第 334 行定义.

3.37.4.4 TIM_ECD_GetCount()

```
uint32_t TIM_ECD_GetCount (
    TIM_ECDTypeDef * TIM_ECDx )
```

函数名称: uint32_t [TIM_ECD_GetCount\(TIM_ECDTypeDef *TIM_ECDx\)](#)

功能描述: 编码器计数值

参见

被引用内容: 无

参数

输入参数:	ECDx: ECD0/ECD1
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
uint32_t ECD0_Value = 0;  
ECD0_Value = TIM_ECD_GetCount(ECD0); //获取编码器0计数值
```

Date	Version	Author	Description
2016年06月06日	1.0	cfwu	修改

修改日志:

在文件 [lks32mc08x_tim.c](#) 第 615 行定义.

3.37.4.5 TIM_ECDCmd()

```
void TIM_ECDCmd (  
    TIM_ECDTypeDef * ECDx,  
    FuncState state )
```

函数名称: void [TIM_ECDCmd\(TIM_ECDTypeDef *ECDx, FuncState state\)](#)

功能描述: 编码器ECDx使能和停止

参见

被引用内容: 无

参数

输入参数:	ECDx: ECD0/ECD1 FuncState state: 使能与失能
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
TIM_ECDCmd(ECD0, ENABLE); // ECD0模块使能
```

Date	Version	Author	Description
2022年04月13日	1.0	HuangMG	修改

修改日志:

在文件 [lks32mc08x_tim.c](#) 第 565 行定义.

3.37.4.6 TIM_ECDInit()

```
void TIM_ECDInit (
    TIM_ECDTypeDef * ECDx,
    TIM_ECDInitTypeDef * TIM_ECDInitStruct )
```

函数名称: void [TIM_ECDInit\(TIM_ECDTypeDef *ECDx, TIM_ECDInitTypeDef *TIM_ECDInitStruct\)](#)

功能描述: 编码器初始化函数

参见

被引用内容: [SYS_ModuleClockCmd\(\)](#);

参数

输入参数:	ECDx: ECD0/ECD1 TIM_ECDInitTypeDef *TIM_ECDInitStruct
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
TIM_ECDInitTypeDef TIM_ECDInitStruct;  
TIM_ECDStructInit(&TIM_ECDInitStruct);  
TIM_ECDInitStruct.ECD_ClockDiv = ECD_Clk_Div1; //96M  
TIM_ECDInitStruct.ECD_Filter0 = 10; //CH0滤波, 滤波宽度为Timer.Filter0*8  
//个时钟周期 ECD.Filter0~15  
TIM_ECDInitStruct.ECD_Filter1 = 10; //CH1滤波  
TIM_ECDInitStruct.ECD_Mode = ECD_Mode_CCWSIGN; //CCW+SIGN, 仅计数信号上升沿  
TIM_ECDInitStruct.ECD_TH = 4096; //计数门限  
TIM_ECDInitStruct.ECD_IRQEna = ECD_IRQEna_Overflow; //使能上溢中断  
TIM_ECDInit(ECD0, &TIM_ECDInitStruct);  
TIM_ECDCmd(ECD0, ENABLE); //使能ECD0模块
```

Date	Version	Author	Description
2016年06月06日	1.0	cfwu	修改

修改日志:

Date	Version	Author	Description
2022年04月13日	1.0	HuangMG	修改

修改日志:增加定时器时钟使能函数

在文件 lks32mc08x_tim.c 第 464 行定义.

函数调用图:



3.37.4.7 TIM_ECDStructInit()

```
void TIM_ECDStructInit (
    TIM_ECDInitTypeDef * TIM_ECDInitStruct )
```

函数名称: void TIM_ECDInit(TIM_ECDTypeDef *ECDx, TIM_ECDInitTypeDef *TIM_ECDInitStruct)

功能描述: 编码器结构体初始化

参见

被引用内容: 无

参数

输入参数:	TIM_ECDInitTypeDef *TIM_ECDInitStruct
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
TIM_ECDInitTypeDef TIM_ECDInitStruct;
TIM_ECDStructInit (&TIM_ECDInitStruct); //编码器结构体初始化
```

Date	Version	Author	Description
2016年06月06日	1.0	cfwu	修改

修改日志:

在文件 [lks32mc08x.tim.c](#) 第 535 行定义.

3.37.4.8 TIM_GetIRQFlag()

```
uint32_t TIM_GetIRQFlag (
    TIM_TimerTypeDef * TIMERx,
    u32 timer_if )
```

函数名称: `uint32_t TIM_GetIRQFlag(TIM_TimerTypeDef *TIMERx , u32 timer_if)`

功能描述: 获取TIM中断标志

参见

被引用内容: 无

参数

输入参数:	TIMERx: TIMER0/TIMER1/TIMER2/TIMER3		
	timer_if:		
	TIMER_IF_CH1	Timer CH1中断, 比较或捕获中断	
	TIMER_IF_CH0	Timer CH0中断, 比较或捕获中断	
	TIMER_IF_ZERO	Timer 过零中断	
输出参数:	无		

返回

返回 值: 0或1, 对应中断标志置位返回1, 未置位返回0

注解

其它说明: 无

警告

只有对应中断使能后, 改为才能读取, 如果对应中断未使能, 读取结果一直为0

示例代码:

```
if (TIM_GetIRQFlag (TIMER0, TIMER_IF_CH0)) //判断UTimer0的CH0是否发生比较中断
{
}
```

Date	Version	Author	Description
2022年04月13日	1.0	HuangMG	修改

修改日志:

在文件 [lks32mc08x_tim.c](#) 第 255 行定义.

3.37.4.9 TIM_Timer_GetCMPT0()

```
uint32_t TIM_Timer_GetCMPT0 (
    TIM_TimerTypeDef * TIM_TIMERx )
```

函数名称: `uint32_t TIM_Timer_GetCMPT0(TIM_TimerTypeDef *TIM_TIMERx)`

功能描述: 获取定时器通道0捕获值

参见

被引用内容: 无

参数

输入参数:	TIM_TIMERx: TIMER0/TIMER1/TIMER2/TIMER3
输出参数:	无

返回

返回值: 定时器通道0捕获值

注解

其它说明: 当定时器发生捕获事件时, 将捕获时刻的cnt值存储到该寄存器中

警告

无

示例代码:

```
uint32_t TIMER0_CAPValue = 0;
TIMER0_CAPValue = TIM_Timer_GetCMPT0(TIMER0); //获取定时器0捕获值
```

Date	Version	Author	Description
2016年06月06日	1.0	cfwu	修改

修改日志:

在文件 [lks32mc08x_tim.c](#) 第 399 行定义.

3.37.4.10 TIM_Timer_GetCMPT1()

```
uint32_t TIM_Timer_GetCMPT1 (
    TIM_TimerTypeDef * TIM_TIMERx )
```

函数名称: uint32_t [TIM_Timer_GetCMPT1](#)(TIM_TimerTypeDef *TIM_TIMERx)

功能描述: 获取定时器通道1捕获值

参见
被引用内容: 无

参数

输入参数:	TIM_TIMERx: TIMER0/TIMER1/TIMER2/TIMER3
输出参数:	无

返回
返回值: 定时器通道1捕获值

注解
其它说明: 当定时器发生捕获事件时, 将捕获时刻的cnt值存储到该寄存器中

警告
无

示例代码:

```
uint32_t TIMER0_CAPValue = 0;
TIMER0_CAPValue = TIM\_Timer\_GetCMPT1(TIMER0); //获取定时器0捕获值
```

Date	Version	Author	Description
2016年06月06日	1.0	cfwu	修改

修改日志:

在文件 [lks32mc08x_tim.c](#) 第 424 行定义.

3.37.4.11 TIM_Timer_GetCount()

```
uint32_t TIM_Timer_GetCount (
    TIM_TimerTypeDef * TIMERx )
```

函数名称: `uint32_t TIM_Timer_GetCount(TIM_TimerTypeDef *TIMERx)`

功能描述: 获取Timer计数值

参见
被引用内容: 无

参数

输入参数:	TIMERx: TIMER0/TIMER1/TIMER2/TIMER3
输出参数:	无

返回
返回值: 无

注解
其它说明: 无

警告
无

示例代码:

```
uint32_t TIMER0.Value = 0;
TIMER0.Value = TIM_Timer_GetCount(TIMER0); //获取定时器0计数值
```

Date	Version	Author	Description
2016年06月06日	1.0	cfwu	修改

修改日志:

在文件 [lks32mc08x_tim.c](#) 第 374 行定义.

3.37.4.12 TIM_TimerCmd()

```
void TIM_TimerCmd (
    TIM_TimerTypeDef * TIMeRx,
    FuncState state )
```

函数名称: void TIM_TimerCmd(TIM_TimerTypeDef *TIMeRx, FuncState state)

功能描述: 定时器TIMeRx使能和停止

参见

被引用内容: 无

参数

输入参数:	TIMeRx: TIMER0/TIMER1/TIMER2/TIMER3 FuncState state: 使能与失能
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

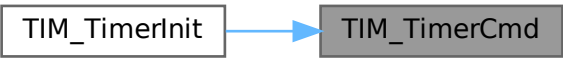
```
TIM_TimerCmd (TIMER0, ENABLE); // Timer0模块使能
```

Date	Version	Author	Description
2022年04月13日	1.0	HuangMG	修改

修改日志:

在文件 lks32mc08x_tim.c 第 187 行定义.

这是这个函数的调用关系图:



3.37.4.13 TIM_TimerInit()

```
void TIM_TimerInit (
    TIM_TimerTypeDef * TIMERx,
    TIM_TimerInitTypeDef * TIM_TimerInitStruct )
```

函数名称: void TIM_TimerInit(TIM_TimerTypeDef *TIMERx, TIM_TimerInitTypeDef *TIM_TimerInitStruct)

功能描述: 定时器初始化函数

参见

被引用内容: SYS_ModuleClockCmd()

参数

输入参数:	TIMERx: TIMER0/TIMER1/TIMER2/TIMER3 TIM_TimerInitTypeDef *TIM_TimerInitStruct
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
TIM_TimerInitTypeDef TIM_InitStructure;
TIM_TimerStrutInit (&TIM_InitStructure); // Timer结构体初始化
TIM_InitStructure.EN = ENABLE; // 定时器模块使能
TIM_InitStructure.Timer_ClockDiv = TIM_Clk_Div1; // 设置Timer模块时钟1分频系数
TIM_InitStructure.Timer_CH0_WorkMode = TIMER_OPMODE_CMP; // 设置Timer CH0 为比较模式
TIM_InitStructure.Timer_CH0Output = 0; // 计数器回零时, 比较模式输出极性控制
TIM_InitStructure.Timer_CH1_WorkMode = TIMER_OPMODE_CMP; // 设置Timer CH1 为比较模式
TIM_InitStructure.Timer_CH1Output = 0; // 计数器回零时, 比较模式输出极性控制
TIM_InitStructure.Timer_TH = 48000; // 定时器计数门限初始值500us
TIM_InitStructure.Timer_CMP0 = 36000; // 设置比较模式的CH0比较初始值25%转空比
TIM_InitStructure.Timer_CMP1 = 24000; // 设置比较模式的CH1比较初始值24000
TIM_InitStructure.Timer_IRQEna = Timer_IRQEna_CH0; // 开启Timer模块比较中断
TIM_TimerInit(TIM0, &TIM_InitStructure);
```

Date	Version	Author	Description
2016年06月06日	1.0	cfwu	创建

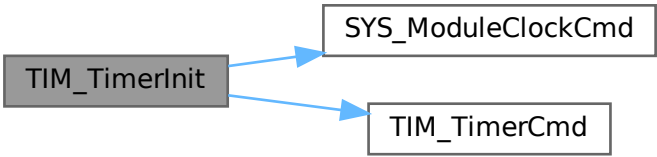
修改日志:

Date	Version	Author	Description
2022年04月13日	1.0	HuangMG	修改

修改日志: 去除定时器使能接口

在文件 [lks32mc08x_tim.c](#) 第 54 行定义.

函数调用图:



3.37.4.14 TIM_TimerStrutInit()

```
void TIM_TimerStrutInit (
    TIM_TimerInitTypeDef * TIM_TimerInitStruct )
```

函数名称: void [TIM_TimerStrutInit\(TIM_TimerInitTypeDef *TIM_TimerInitStruct\)](#)

功能描述: 定时器结构体初始化

参见

被引用内容： 无

参数

输入参数:	TIM_TimerInitTypeDef *TIM.TimerInitStruct
输出参数:	无

返回

返回 值: 无

注解

其它说明： 无

警告

无

示例代码:

```
TIM_TimerInitTypeDef TIM_InitStruct;
TIM_TimerStrutInit (&TIM_InitStruct); // Timer结构体初始化
```

Date	Version	Author	Description
2016年06月06日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_tim.c](#) 第 147 行定义.

3.38 lks32mc08x_tim.h

[浏览该文件的文档.](#)

```
00001
00014 #ifndef __LKS32MC08x_TIM_H
00015 #define __LKS32MC08x_TIM_H
00016
00017 /* Includes -----*/
00018 #include "lks32mc08x.lib.h"
00019
00023 typedef struct
00024 {
00025     __IO uint32_t CFG;
00026     __IO uint32_t TH;
00027     __IO uint32_t CNT;
00028     __IO uint32_t CMPT0;
00029     __IO uint32_t CMPT1;
```

```

00030     __IO uint32_t EVT;
00031 } TIM_TimeTypeDef;
00032
00036 typedef struct
00037 {
00038     __IO uint32_t CFG;
00039     __IO uint32_t TH;
00040     __IO uint32_t CNT;
00041 } TIM_ECDTypeDef;
00042
00046 #ifndef TIMER0
00047 #define TIMER0 ((TIM_TimeTypeDef *)UTIMER_BASE)
00048 #endif
00052 #ifndef TIMER1
00053 #define TIMER1 ((TIM_TimeTypeDef *) (UTIMER_BASE + 0x20))
00054 #endif
00058 #ifndef TIMER2
00059 #define TIMER2 ((TIM_TimeTypeDef *) (UTIMER_BASE + 0x40))
00060 #endif
00064 #ifndef TIMER3
00065 #define TIMER3 ((TIM_TimeTypeDef *) (UTIMER_BASE + 0x60))
00066 #endif
00070 #ifndef ECD0
00071 #define ECD0 ((TIM_ECDTypeDef *) (UTIMER_BASE + 0x80))
00072 #endif
00076 #ifndef ECD1
00077 #define ECD1 ((TIM_ECDTypeDef *) (UTIMER_BASE + 0x90))
00078 #endif
00079
00080 #define TIM_Clk_Div1 0x00
00081 #define TIM_Clk_Div2 0x01
00082 #define TIM_Clk_Div4 0x02
00083 #define TIM_Clk_Div8 0x03
00085 #define ECD_Clk_Div1 0x00
00086 #define ECD_Clk_Div2 0x01
00087 #define ECD_Clk_Div4 0x02
00088 #define ECD_Clk_Div8 0x03
00093 typedef enum
00094 {
00095     Timer_IRQEna_None = 0,
00096     Timer_IRQEna_Zero = BIT0,
00097     Timer_IRQEna_CH0 = BIT1,
00098     Timer_IRQEna_CH1 = BIT2,
00099     Timer_IRQEna_All = 0x07
00100 } Timer_IRQEnaDef;
00101
00102 #define TIMER_IF_CH1 BIT2
00103 #define TIMER_IF_CH0 BIT1
00104 #define TIMER_IF_ZERO BIT0
00106 #define ECD_IRQEna_None 0
00107 #define ECD_IRQEna_UnderFlow BIT0
00108 #define ECD_IRQEna_Overflow BIT1
00110 #define ECD_IF_UNDERFLOW BIT0
00111 #define ECD_IF_OVERFLOW BIT1
00113 #define TIMER_CapMode_None ((uint32_t)0x00000000)
00114 #define TIMER_CapMode_Rise ((uint32_t)0x00000001)
00115 #define TIMER_CapMode_Fall ((uint32_t)0x00000002)
00116 #define TIMER_CapMode_RiseFall ((uint32_t)0x00000003)
00118 #define TIMER_OPMode_CAP ((uint32_t)0x00000001)
00119 #define TIMER_OPMode_CMP ((uint32_t)0x00000000)
00121 #define ECD_Mode_T1 ((uint32_t)0x00)
00122 #define ECD_Mode_T1_T2 ((uint32_t)0x01)
00123 #define ECD_Mode_CCWSIGN ((uint32_t)0x02)
00124 #define ECD_Mode_CCWCW ((uint32_t)0x03)
00125 #define ECD_Mode_CCWSIGN_UpDown ((uint32_t)0x06)
00126 #define ECD_Mode_CCWCW_UpDown ((uint32_t)0x07)
00128 typedef struct
00129 {
00130     FuncState EN; /*定时器模块使能*/
00136     uint32_t Timer_CH0_WorkMode;
00144     uint32_t Timer_CH0_CapMode;
00145     uint32_t Timer_CH0_Output;
00151     uint32_t Timer_CH1_WorkMode;
00159     uint32_t Timer_CH1_CapMode;
00160     uint32_t Timer_CH1_Output;
00162     uint32_t Timer_TH;
00163     uint32_t Timer_CMP0;
00164     uint32_t Timer_CMP1;
00166     uint32_t Timer_Filter0;
00167     uint32_t Timer_Filter1;
00176     uint32_t Timer_ClockDiv;
00181     uint32_t Timer_IRQEna;
00182 } TIM_TimerInitTypeDef;
00183
00184 typedef struct
00185 {
00195     uint32_t ECD_Mode;

```

```

00196     uint32_t ECD_TH;
00204     uint32_t ECD_ClockDiv;
00205
00206     uint32_t ECD_Filter0;
00207     uint32_t ECD_Filter1;
00214     uint32_t ECD_IRQEna;
00215 } TIM_ECDInitTypeDef;
00216
00217 /*Timer初始化*/
00218 void TIM_TimerInit(TIM_TimerTypeDef *TIMERx, TIM_TimerInitTypeDef *TIM_TimerInitStruct);
00219 void TIM_TimerStrutInit(TIM_TimerInitTypeDef *TIM_TimerInitStruct);
00220 // void TIM_StartTimer(TIM_TimerTypeDef *TIMERx);
00221 // void TIM_StopTimer(TIM_TimerTypeDef *TIMERx);
00222 void TIM_TimerCmd(TIM_TimerTypeDef *TIMERx, FuncState state);
00223
00224 /*TIM获取中断标志*/
00225 uint32_t TIM_GetIRQFlag(TIM_TimerTypeDef *TIMERx, u32 timer_if);
00226 /*TIM中断清除*/
00227 void TIM_ClearIRQFlag(TIM_TimerTypeDef *TIMERx, uint32_t tempFlag);
00228
00229 uint32_t TIM_Timer_GetCount(TIM_TimerTypeDef *TIM_TIMERx);
00230 uint32_t TIM_Timer_GetCMPT0(TIM_TimerTypeDef *TIM_TIMERx);
00231 uint32_t TIM_Timer_GetCMPT1(TIM_TimerTypeDef *TIM_TIMERx);
00232
00233 /*ECD初始化*/
00234 void TIM_ECDInit(TIM_ECDTypeDef *ECDx, TIM_ECDInitTypeDef *TIM_ECDInitStruct);
00235 void TIM_ECDStructInit(TIM_ECDInitTypeDef *TIM_ECDInitStruct);
00236 void TIM_ECDCmd(TIM_ECDTypeDef *ECDx, FuncState state);
00237
00238 uint32_t TIM_ECD_GetCount(TIM_ECDTypeDef *TIM_ECDx);
00239 /*ECD获取中断标志*/
00240 uint32_t ECD_GetIRQFlag(TIM_ECDTypeDef *ECDx, u32 timer_if);
00241 /*ECD中断清除*/
00242 void ECD_ClearIRQFlag(TIM_ECDTypeDef *ECDx, uint32_t tempFlag);
00243 #endif /*_lks32mc08x_TIM_H */
00244
00245 /***** (C) COPYRIGHT LINKO SEMICONDUCTOR *****/

```

3.39 include/lks32mc08x_tmp.h 文件参考

文件名称: [lks32mc08x_tmp.h](#)

文件标识: 无

内容摘要: 芯片内置温度传感器驱动头文件

其它说明: 无

#include "lks32mc08x_lib.h"

lks32mc08x.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:



结构体

- struct **Stru_TemperatureCof_Def**

芯片内置温度传感器初始化结构体句柄, 初始化时请定义该句柄, 并用其它地址来传参

函数

- void **TempSensor_Init** (void)
函数名称: void **TempSensor_Init**(void)
- s16 **GetCurrentTemperature** (s16 ADC_value)
函数名称: s16 **GetCurrentTemperature**(s16 ADC_value)

3.39.1 详细描述

文件名称: lks32mc08x_tmp.h
文件标识: 无
内容摘要: 芯片内置温度传感器驱动头文件
其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2015年11月5日	1.0	Howlet	创建

修改日志:

在文件 **lks32mc08x_tmp.h** 中定义.

3.39.2 函数说明

3.39.2.1 GetCurrentTemperature()

```
s16 GetCurrentTemperature (  
    s16 ADC_value )
```

函数名称: s16 **GetCurrentTemperature**(s16 ADC_value)

功能描述: 获取当前温度值

参见

被引用内容: 无

参数

输入参数:	ADC_value: ADC通道14为温度传感器, ADC采样结果值
输出参数:	无

返回

返回 值: `t.Temperture`: 当前温度值, 单位: 1个Lsb代表0.1度

注解

其它说明: `ADC.value`输入的ADC值不区分左右对齐, 函数内已经进行ADC结果左右对齐判断

警告

无

示例代码:

```
s16 temp_value = 0;
temp_value = GetCurrentTemperature(ADC0.DATA0); //获取当前温度值,单位0.1度
```

Date	Version	Author	Description
2015年11月5日	1.0	Howlet	创建

修改日志:

在文件 [lks32mc08x_tmp.c](#) 第 67 行定义.

3.39.2.2 TempSensor.Init()

```
void TempSensor.Init (
    void )
```

函数名称: `void TempSensor.Init(void)`

功能描述: 温度传感器初始化

参见

被引用内容: 无

参数

输入参数:	无
输出参数:	无

返回

返回 值: 无

注解

其它说明： 无

警告

无

示例代码:

```
TempSensor.Init(); //温度传感器初始化
```

Date	Version	Author	Description
2015年11月5日	1.0	Howlet	创建

修改日志:

在文件 [lks32mc08x_tmp.c](#) 第 38 行定义.

函数调用图:



3.40 lks32mc08x_tmp.h

[浏览该文件的文档.](#)

```
00001
00014 #ifndef _LKS32MC08x_TMP_H_
00015 #define _LKS32MC08x_TMP_H_
00016
00017 #include "lks32mc08x.lib.h"
00018
00022 typedef struct
00023 {
00024     u16 nCofA;
00025     u16 nOffsetB;
00026 } Stru_TempertureCof_Def;
00027
00028 void TempSensor.Init(void);
00029 s16 GetCurrentTemperature(s16 ADC.value);
00030
00031 #endif /*_CONNIE_TMP_H_ */
00032
00033 /***** (C) COPYRIGHT LINKO SEMICONDUCTOR *****/
```

3.41 include/lks32mc08x_uart.h 文件参考

文件名称: [lks32mc08x_uart.h](#)

文件标识: 无

内容摘要: UART驱动头文件

其它说明: 无

```
#include "lks32mc08x_lib.h"
```

[lks32mc08x_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_TypeDef](#)
UART寄存器结构体句柄
- [struct UART_InitTypeDef](#)
UART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 UART_WORDLENGTH_8b 0](#)
串口0结构体基地址定义
- [#define UART_WORDLENGTH_7b 1](#)
7bit数据位
- [#define UART_STOPBITS_1b 0](#)
无停止位
- [#define UART_STOPBITS_2b 1](#)
1bit停止位
- [#define UART_FIRSTSEND_LSB 0](#)
低端先发送
- [#define UART_FIRSTSEND_MSB 1](#)
高端先发送
- [#define UART_IRQEna_SendOver BIT0](#)
使能发送完成中断
- [#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
- `#define UART_IRQEna_CheckError` BIT4
使能校验错误
- `#define UART_IRQEna_TX_DMA_RE` BIT5
发送完成DMA请求使能
- `#define UART_IRQEna_RX_DMA_RE` BIT6
接收完成DMA请求使能
- `#define UART_IRQEna_TX_BUF_DMA_RE` BIT7
发送缓冲区空DMA请求使能
- `#define UART_IF_SendOver` BIT0
发送完成中断
- `#define UART_IF_RcvOver` BIT1
接收完成中断
- `#define UART_IF_SendBufEmpty` BIT2
发送缓冲区空中断
- `#define UART_IF_StopError` BIT3
停止位错误
- `#define UART_IF_CheckError` BIT4
校验错误

枚举

- `enum UART_ParityMode { UART_Parity_NO = 0x00 , UART_Parity_EVEN = 0x01 , UART_Parity_ODD = 0x02 }`
串口奇偶校验配置枚举成员

函数

- `void UART_Init (UART_TypeDef *UARTx, UART_InitTypeDef *UART_InitStruct)`
函数名称: `void UART_Init(UART_TypeDef* UARTx, UART_InitTypeDef* UART_InitStruct)`
- `void UART_StructInit (UART_InitTypeDef *UART_InitStruct)`
函数名称: `void UART_StructInit(UART_InitTypeDef* UART_InitStruct)`
- `void UART_SendData (UART_TypeDef *UARTx, uint32_t n)`
函数名称: `void UART_SENDDATA(UART_TypeDef *UARTx, uint32_t n)`
- `uint32_t UART_ReadData (UART_TypeDef *UARTx)`
函数名称: `uint32_t UART_ReadData(UART_TypeDef *UARTx)`
- `uint8_t UART_SendAddr_485 (UART_TypeDef *UARTx, uint8_t n)`
函数名称: `uint8_t UART_SendAddr_485(UART_TypeDef *UARTx, uint8_t n)`
- `uint8_t UART_SendData_485 (UART_TypeDef *UARTx, uint8_t n)`
函数名称: `uint8_t UART_SendData_485(UART_TypeDef *UARTx, uint8_t n)`
- `uint32_t UART_GetIRQFlag (UART_TypeDef *UARTx, uint32_t tempFlag)`
函数名称: `uint32_t UART_GetIRQFlag(UART_TypeDef *UARTx, uint32_t tempFlag)`
- `void UART_ClearIRQFlag (UART_TypeDef *UARTx, uint32_t tempFlag)`
函数名称: `void UART_ClearIRQFlag(UART_TypeDef *UARTx, uint32_t tempFlag)`

3.41.1 详细描述

文件名称: [lks32mc08x_uart.h](#)

文件标识: 无

内容摘要: UART驱动头文件

其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2016年05月21日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_uart.h](#) 中定义.

3.41.2 宏定义说明

3.41.2.1 UART_FIRSTSEND_LSB

```
#define UART_FIRSTSEND_LSB 0
```

低端先发送

在文件 [lks32mc08x_uart.h](#) 第 96 行定义.

3.41.2.2 UART_FIRSTSEND_MSB

```
#define UART_FIRSTSEND_MSB 1
```

高端先发送

在文件 [lks32mc08x_uart.h](#) 第 97 行定义.

3.41.2.3 UART_IF_CheckError

```
#define UART_IF_CheckError BIT4
```

校验错误

在文件 [lks32mc08x_uart.h](#) 第 116 行定义.

3.41.2.4 UART_IF_RcvOver

```
#define UART_IF_RcvOver BIT1
```

接收完成中断

在文件 [lks32mc08x_uart.h](#) 第 113 行定义.

3.41.2.5 UART_IF_SendBufEmpty

```
#define UART_IF_SendBufEmpty BIT2
```

发送缓冲区空中断

在文件 [lks32mc08x_uart.h](#) 第 114 行定义.

3.41.2.6 UART_IF_SendOver

```
#define UART_IF_SendOver BIT0
```

发送完成中断

在文件 [lks32mc08x_uart.h](#) 第 112 行定义.

3.41.2.7 UART_IF_StopError

```
#define UART_IF_StopError BIT3
```

停止位错误

在文件 [lks32mc08x_uart.h](#) 第 115 行定义.

3.41.2.8 UART_IRQEna_CheckError

```
#define UART_IRQEna_CheckError BIT4
```

使能校验错误

在文件 [lks32mc08x_uart.h](#) 第 104 行定义.

3.41.2.9 UART_IRQEna_RcvOver

```
#define UART_IRQEna_RcvOver BIT1
```

使能接收完成中断

在文件 [lks32mc08x_uart.h](#) 第 101 行定义.

3.41.2.10 UART_IRQEna_RX_DMA_RE

```
#define UART_IRQEna_RX_DMA_RE BIT6
```

接收完成DMA请求使能

在文件 [lks32mc08x_uart.h](#) 第 108 行定义.

3.41.2.11 UART_IRQEna_SendBuffEmpty

```
#define UART_IRQEna_SendBuffEmpty BIT2
```

使能发送缓冲区空中断

在文件 [lks32mc08x_uart.h](#) 第 102 行定义.

3.41.2.12 UART_IRQEna_SendOver

```
#define UART_IRQEna_SendOver BIT0
```

使能发送完成中断

在文件 [lks32mc08x_uart.h](#) 第 100 行定义.

3.41.2.13 UART_IRQEna_StopError

```
#define UART_IRQEna_StopError BIT3
```

使能停止位错误

在文件 [lks32mc08x_uart.h](#) 第 103 行定义.

3.41.2.14 UART_IRQEna_TX_BUF_DMA_RE

```
#define UART_IRQEna_TX_BUF_DMA_RE BIT7
```

发送缓冲区空DMA请求使能

在文件 [lks32mc08x_uart.h](#) 第 109 行定义.

3.41.2.15 UART_IRQEna_TX_DMA_RE

```
#define UART_IRQEna_TX_DMA_RE BIT5
```

发送完成DMA请求使能

在文件 [lks32mc08x_uart.h](#) 第 107 行定义.

3.41.2.16 UART_STOPBITS_1b

```
#define UART_STOPBITS_1b 0
```

无停止位

在文件 [lks32mc08x_uart.h](#) 第 93 行定义.

3.41.2.17 UART_STOPBITS_2b

```
#define UART_STOPBITS_2b 1
```

1bit停止位

在文件 [lks32mc08x_uart.h](#) 第 94 行定义.

3.41.2.18 UART_WORDLENGTH_7b

```
#define UART_WORDLENGTH_7b 1
```

7bit数据位

在文件 [lks32mc08x_uart.h](#) 第 91 行定义.

3.41.2.19 UART_WORDLENGTH_8b

```
#define UART_WORDLENGTH_8b 0
```

串口0结构体基地址定义

串口1结构体基地址定义 系统时钟（UART使用系统时钟分频） 8bit数据位

在文件 [lks32mc08x_uart.h](#) 第 90 行定义.

3.41.3 枚举类型说明

3.41.3.1 UART_ParityMode

```
enum UART_ParityMode
```

串口奇偶校验配置枚举成员

枚举值

UART_Parity_NO	无奇偶校验
UART_Parity_EVEN	偶校验
UART_Parity_ODD	奇校验

在文件 [lks32mc08x_uart.h](#) 第 21 行定义.

3.41.4 函数说明

3.41.4.1 UART_ClearIRQFlag()

```
void UART_ClearIRQFlag (
    UART_TypeDef * UARTx,
    uint32_t tempFlag )
```

函数名称: void [UART_ClearIRQFlag\(UART_TypeDef *UARTx, uint32_t tempFlag\)](#)

功能描述: 清除UART中断标志

参见

被引用内容: UART0 , UART1

参数

输入参数:	UART_TypeDef: UART0/UART1	
	tempFlag参数可选:	
	宏定义	说明
	UART_IF_SendOver	发送完成中断
	UART_IF_RcvOver	接收完成中断
	UART_IF_SendBufEmpty	发送缓冲区空中断
	UART_IF_StopError	停止位错误
	UART_IF_CheckError	校验错误
输出参数:	无	

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
if (UART_GetIRQFlag (UART0, UART_IF_SendOver)) //获取发送完成中断标志位
{
    UART_ClearIRQFlag (UART0, UART_IF_SendOver) //清除发送完成中断标志位
}
```


Date	Version	Author	Description
2012年04月21日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_uart.c](#) 第 332 行定义.

3.41.4.2 UART_GetIRQFlag()

```
uint32_t UART_GetIRQFlag (  
    UART_TypeDef * UARTx,  
    uint32_t tempFlag )
```

函数名称: `uint32_t UART_GetIRQFlag(UART_TypeDef *UARTx , uint32_t tempFlag)`

功能描述: 获取UART中断标志

参见

被引用内容: UART0 , UART1

参数

输入参数:	UART_TypeDef: UART0/UART1	
	tempFlag参数可选:	
	宏定义	说明
	UART_IF_SendOver	发送完成中断
	UART_IF_RcvOver	接收完成中断
	UART_IF_SendBufEmpty	发送缓冲区空中断
	UART_IF_StopError	停止位错误
输出参数:	UART_IF_CheckError	校验错误
	无	

返回

返回值: 无

注解

其它说明: 无

警告

只有对应中断使能后, 改为才能读取, 如果对应中断未使能, 读取结果一直为0

示例代码:

```
if(UART_GetIRQFlag(UART0,UART_IF_SendOver))//获取发送完成中断标志位
{
    UART_ClearIRQFlag(UART0,UART_IF_SendOver)//清除发送完成中断标志位
}
```

Date	Version	Author	Description
2012年04月21日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_uart.c](#) 第 293 行定义.

3.41.4.3 UART_Init()

```
void UART_Init (
    UART_TypeDef * UARTx,
    UART_InitTypeDef * UART_InitStruct )
```

函数名称: void UART_Init(UART_TypeDef* UARTx, EUART_InitTypeDef* UART_InitStruct)

功能描述: UART初始化函数

参见

被引用内容: [SYS_ModuleClockCmd\(\)](#)

参数

输入参数:	UART_TypeDef: UART0/UART1 , UART_InitTypeDef
输出参数:	无

返回

返回值: 无

注解

其它说明: 串口波特率计算使用UART.MCU.MCLK系统时钟进行计算，使用前需要确认系统时钟频率

警告

无

示例代码:

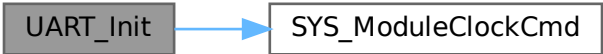
```
UART_InitTypeDef UART_InitStructure;  
UART_StructInit(&UART_InitStructure);  
UART_InitStructure.BaudRate = 38400; // 设置波特率38400  
UART_InitStructure.WordLength = UART_WORDLENGTH_8b; // 发送数据长度8位  
UART_InitStructure.StopBits = UART_STOPBITS_1b; // 停止位1位  
UART_InitStructure.FirstSend = UART_FIRSTSEND_LSB; // 先发送LSB  
UART_InitStructure.ParityMode = UART_Parity_NO; // 无奇偶校验  
UART_InitStructure.IRQEna = UART_IRQEna_SendOver | UART_IRQEna_RcvOver; // 串口中断使能  
UART_Init(UART0, &UART_InitStructure);
```

Date	Version	Author	Description
2016年05月21日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_uart.c](#) 第 44 行定义.

函数调用图:



3.41.4.4 UART_ReadData()

```
uint32_t UART_ReadData (  
    UART_TypeDef * UARTx )
```

函数名称: `uint32_t UART_ReadData(UART_TypeDef *UARTx)`

功能描述: UART读缓冲区数据

参见

被引用内容: UART0 , UART1

参数

输入参数:	UART_TypeDef: UART0/UART1
输出参数:	无

返回

返回 值: 返回串口接收缓冲区的1Byte数据

注解

其它说明: 无

警告

无

示例代码:

```
u8 UART_BUFF = 0;
UART_BUFF = UART_ReadData (UART0); //读取串口0接收一字节数据
```

Date	Version	Author	Description
2016年05月21日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_uart.c](#) 第 174 行定义.

3.41.4.5 UART_SendAddr_485()

```
uint8_t UART_SendAddr_485 (
    UART_TypeDef * UARTx,
    uint8_t n )
```

函数名称: `uint8_t UART_SendAddr_485(UART_TypeDef *UARTx, uint8_t n)`

功能描述: 发送485通讯中需要接收数据的从机地址

参见

被引用内容: UART0 , UART1

参数

输入参数:	UART_TypeDef: UART0/UART1 , n: 要发生1Byrt地址
输出参数:	无

返回

返回 值： 1： 发送失败， 0： 发送成功

注解

其它说明： cnt作用是延时防止程序卡死。在发送数据后立即发送地址的情况下,会 判断等待发送缓冲区空在进行地址发送，如果长时间还未发送完成（cnt计数时间）， 则跳出while循环。如果串口波特率较低cnt计数已满， 会出现数据还未发送完成进行 下一个数据发送，为了避免此类问题发送可以增加cnt值。

警告

无

示例代码:

```
UART_SendAddr_485 (UART0,0x55); //串口0发送485从机接收地址为0x55
```

Date	Version	Author	Description
2012年04月21日	1.0	HuangMG	创建

修改日志:

在文件 lks32mc08x_uart.c 第 201 行定义.

3.41.4.6 UART_SendData()

```
void UART_SendData (
    UART_TypeDef * UARTx,
    uint32_t n )
```

函数名称: void UART_SENDDATA(UART_TypeDef *UARTx, uint32_t n)

功能描述: UART发送数据

参见

被引用内容： UART0 ， UART1

参数

输入参数:	UART_TypeDef: UART0/UART1 , n: 要发生1Byte数据
输出参数:	无

返回

返回 值： 无

注解

其它说明： 无

警告

无

示例代码:

```
UART_SendData (UART0, 0x12); //串口0发送0x12一字节数据
```

Date	Version	Author	Description
2016年05月21日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_uart.c](#) 第 148 行定义.

3.41.4.7 UART_SendData_485()

```
uint8_t UART_SendData_485 (
    UART_TypeDef * UARTx,
    uint8_t n )
```

函数名称: `uint8_t UART_SendData_485(UART_TypeDef *UARTx, uint8_t n)`

功能描述: 发送485通讯中的数据信息

参见

被引用内容: UART0 , UART1

参数

输入参数:	UART_TypeDef: UART0/UART1 , n: 要发生1Byrt数据
输出参数:	无

返回

返回 值： 1: 发送失败, 0: 发送成功

注解

其它说明： cnt作用是延时防止程序卡死。在发送数据后立即发送地址的情况下,会 判断等待发送缓冲区空在进行地址发送，如果长时间还未发送完成（cnt计数时间）， 则跳出while循环。如果串口波特率较低cnt计数已满， 会出现数据还未发送完成进行 下一个数据发送， 为了避免此类问题发送可以增加cnt值。

警告

无

示例代码:

```
UART_SendData_485 (UART0,0x12); //串口0发送485数据为0x12
```

Date	Version	Author	Description
2012年04月21日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_uart.c](#) 第 243 行定义.

3.41.4.8 UART_StructInit()

```
void UART_StructInit (
    UART_InitTypeDef * UART_InitStruct )
```

函数名称: void [UART_StructInit\(UART_InitTypeDef* UART_InitStruct\)](#)

功能描述: UART结构体初始化函数

参见

被引用内容: 无

参数

输入参数:	UART_InitTypeDef
输出参数:	无

返回

返回 值: 无

注解

其它说明： 无

警告

无

示例代码:

```
UART_InitTypeDef UART_InitStructure;  
UART_StructInit(&UART_InitStructure); //初始化UART相关配置结构体变量
```

Date	Version	Author	Description
2016年05月21日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_uart.c](#) 第 112 行定义.

3.42 lks32mc08x_uart.h

[浏览该文件的文档.](#)

```
00001  
00015 #ifndef __LKS32MC08x_UART_H  
00016 #define __LKS32MC08x_UART_H  
00017  
00018 #include "lks32mc08x_lib.h"  
00019  
00021 typedef enum  
00022 {  
00023     UART_Parity_NO    = 0x00,  
00024     UART_Parity_EVEN  = 0x01,  
00025     UART_Parity_ODD   = 0x02  
00026 } UART_ParityMode;  
00027  
00031 typedef struct  
00032 {  
00033     __IO uint32_t CTRL;  
00034     __IO uint32_t DIVH;  
00035     __IO uint32_t DIVL;  
00036     __IO uint32_t BUFF;  
00037     __IO uint32_t ADR;  
00038     __IO uint32_t STT;  
00039     __IO uint32_t IE;  
00040     __IO uint32_t IF;  
00041     __IO uint32_t INV;  
00042 } UART_TypeDef;  
00043  
00047 typedef struct  
00048 {  
00049     uint32_t BaudRate;  
00050     uint8_t WordLength;  
00051     uint8_t StopBits;  
00052     uint8_t FirstSend;  
00053     UART_ParityMode ParityMode;  
00055     uint8_t MultiDropEna;  
00056     uint8_t Bit9Value;  
00058     uint16_t Match485Addr;  
00066     uint8_t IRQEna;  
00067     uint8_t RXD_INV;  
00068     uint8_t TXD_INV;
```



```

00069 } UART_InitTypeDef;
00073 #ifndef UART0
00074 #define UART0 ((UART_TypeDef *)UART0_BASE)
00075 #endif
00079 #ifndef UART1
00080 #define UART1 ((UART_TypeDef *)UART1_BASE)
00081 #endif
00082
00086 #ifndef UART_MCU_MCLK
00087 #define UART_MCU_MCLK (96000000LL)
00088 #endif
00089
00090 #define UART_WORDLENGTH_8b 0
00091 #define UART_WORDLENGTH_7b 1
00093 #define UART_STOPBITS_1b 0
00094 #define UART_STOPBITS_2b 1
00096 #define UART_FIRSTSEND_LSB 0
00097 #define UART_FIRSTSEND_MSB 1
00099 /*中断使能定义*/
00100 #define UART_IRQEna_SendOver BIT0
00101 #define UART_IRQEna_RcvOver BIT1
00102 #define UART_IRQEna_SendBuffEmpty BIT2
00103 #define UART_IRQEna_StopError BIT3
00104 #define UART_IRQEna_CheckError BIT4
00106 /*DMA请求使能定义*/
00107 #define UART_IRQEna_TX_DMA_RE BIT5
00108 #define UART_IRQEna_RX_DMA_RE BIT6
00109 #define UART_IRQEna_TX_BUF_DMA_RE BIT7
00111 /* 中断标志定义*/
00112 #define UART_IF_SendOver BIT0
00113 #define UART_IF_RcvOver BIT1
00114 #define UART_IF_SendBufEmpty BIT2
00115 #define UART_IF_StopError BIT3
00116 #define UART_IF_CheckError BIT4
00118 void UART_Init(UART_TypeDef *UARTx, UART_InitTypeDef *UART_InitTypeDef);
00119 void UART_StructInit(UART_InitTypeDef *UART_InitStruct);
00120
00121 void UART_SendData(UART_TypeDef *UARTx, uint32_t n);
00122 uint32_t UART_ReadData(UART_TypeDef *UARTx);
00123
00124 uint8_t UART_SendAddr_485(UART_TypeDef *UARTx, uint8_t n);
00125 uint8_t UART_SendData_485(UART_TypeDef *UARTx, uint8_t n);
00126
00127 uint32_t UART_GetIRQFlag(UART_TypeDef *UARTx, uint32_t tempFlag);
00128 void UART_ClearIRQFlag(UART_TypeDef *UARTx, uint32_t tempFlag);
00129
00130 #endif /*_lks32mc08x_UART_H */
00131
00132 /***** (C) COPYRIGHT LINKO SEMICONDUCTOR *****/

```

3.43 source/lks32mc08x_adc.c 文件参考

文件名称: lks32mc08x_adc.c

文件标识: 无

内容摘要: ADC外设驱动程序

其它说明: 无

```
#include "lks32mc08x_adc.h"
```

lks32mc08x_adc.c 的引用(Include)关系图:



函数

- void [ADC_Init](#) (ADC_TypeDef *ADCx, [ADC_InitTypeDef](#) *ADC_InitStruct)

- 函数名称: `void ADC_Init(ADC_TypeDef* ADCx, ADC_InitTypeDef* ADC_InitStruct)`
- `void ADC_StructInit (ADC_InitTypeDef *ADC_InitStruct)`
 - 函数名称: `void ADC_StructInit(ADC_InitTypeDef* ADC_InitStruct)`
- `void ADC_ClearIRQFlag (ADC_TypeDef *ADCx, uint16_t INT_flag)`
 - 函数名称: `void ADC_ClearIRQFlag(ADC_TypeDef* ADCx, uint16_t INT_flag)`
- `uint16_t ADC_GetIRQFlag (ADC_TypeDef *ADCx, uint16_t INT_flag)`
 - 函数名称: `uint16_t ADC_GetIRQFlag(ADC_TypeDef* ADCx, uint16_t INT_flag)`
- `void ADC_ChannelConfig (ADC_TypeDef *ADCx, CHN_t CHNum, uint16_t nChannel)`
 - 函数名称: `void ADC_ChannelConfig(ADC_TypeDef* ADCx, CHN_t CHNum, uint16_t Channel_0, uint16_t Channel_1)`
- `s16 ADC_GetConversionValue (DAT_t DATNum)`
 - 函数名称: `s16 ADC_GetConversionValue(DAT_t DATNum)`
- `void ADC_SoftTrgEN (ADC_TypeDef *ADCx, FuncState state)`
 - 函数名称: `void ADC_SoftTrgEN(ADC_TypeDef* ADCx, FuncState state)`

3.43.1 详细描述

文件名称: [lks32mc08x_adc.c](#)
 文件标识: 无
 内容摘要: ADC外设驱动程序
 其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2015年11月5日	1.0	Howlet	创建

修改日志:

Date	Version	Author	Description
2020年12月15日	1.1	Howlet	修改

修改日志:调用NVR.Trim访问ADC加载值

在文件 [lks32mc08x_adc.c](#) 中定义.

3.43.2 函数说明

3.43.2.1 ADC_ChannelConfig()

```
void ADC_ChannelConfig (
    ADC_TypeDef * ADCx,
```

```
CHNx CHNum,
uint16_t nChannel )
```

函数名称: void ADC_ChannelConfig(ADC_TypeDef* ADCx,CHNx CHNum, uint16_t Channel_0 ,uint16_t Channel_1)

功能描述: ADC采样通道设置

参见

被引用内容: CHNx

参数

输入参数:	ADCx, 08系列固定选择ADC0 CHNum为枚举类型，对应0-19； Channel_0为采样通道序号对应0-19 nChannel: ADC_CHANNEL_0 ~ ADC_CHANNEL_19
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
ADC_ChannelConfig (ADC0, CHN0, ADC_CHANNEL_3) ;//配置ADC第0次采样ADC通道3的信号
ADC_ChannelConfig (ADC0, CHN1, ADC_CHANNEL_0) ;//配置ADC第1次采样ADC通道0的信号
```

Date	Version	Author	Description
2022年04月20日	1.0	Zhu Jie	创建

修改日志:

在文件 lks32mc08x_adc.c 第 264 行定义.

3.43.2.2 ADC_ClearIRQFlag()

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

函数名称: void **ADC_ClearIRQFlag**(ADC_TypeDef* ADCx, uint16_t INT_flag)

功能描述: ADC中断标志位清零

参见
被引用内容: 无

参数

输入参数:	ADCx:	ADC0
	INT_flag:	ADC中断标志
	ADC_DAT0_OV_IRQ_IF	ADC_DAT0超阈值中断标志位
	ADC_H_CONFLICT_IRQ_IF	硬件触发冲突标志
	ADC_S_CONFLICT_IRQ_IF	软件触发冲突标志
	ADC_EOS3_IRQ_IF	第四段扫描结束中断标志
	ADC_EOS2_IRQ_IF	第三段扫描结束中断标志
	ADC_EOS1_IRQ_IF	第二段扫描结束中断标志
	ADC_EOS0_IRQ_IF	第一段扫描结束中断标志
	ADC_ALL_IRQ_IF	ADC全部中断标志位——可用于初始化
输出参数:	无	

返回
返回值: 无

注解
其它说明: 无

警告
无

示例代码:

```
if(ADC_GetIRQFlag(ADC0,ADC_EOS0_IRQ_IF))//获取第一段扫描结束中断标志
{
    ADC_ClearIRQFlag(ADC0,ADC_EOS0_IRQ_IF)//第一段扫描结束中断标志
}
```

Date	Version	Author	Description
2022年04月20日	1.0	Zhu Jie	创建

修改日志:

在文件 [lks32mc08x_adc.c](#) 第 189 行定义.

3.43.2.3 ADC_GetConversionValue()

s16 ADC_GetConversionValue (
 [DATx](#) DATNum)

函数名称: s16 ADC_GetConversionValue(DATNx DATNum)

功能描述: 读ADC采样值

参见

被引用内容: [DATx](#)

参数

输入参数:	DATx: 读取第几次采样结果值
输出参数:	无

返回

返回 值: 对应x次采样结果值

注解

其它说明: 无

警告

无

示例代码:

```
s16 ADC_samp0 = 0;  
ADC_samp0 = ADC_GetConversionValue(DAT0); //读取ADC第0次采样结果
```

Date	Version	Author	Description
2022年04月20日	1.0	Zhu Jie	创建

修改日志:

在文件 [lks32mc08x_adc.c](#) 第 295 行定义.

3.43.2.4 ADC_GetIRQFlag()

```
u16 ADC_GetIRQFlag (
    ADC_TypeDef * ADCx,
    uint16_t INT_flag )
```

函数名称: `uint16_t ADC_GetIRQFlag(ADC_TypeDef* ADCx, uint16_t INT_flag)`

功能描述: 获取ADC中断标志

参见
被引用内容: 无

参数

输入参数:	ADCx:	ADC0
	INT_flag:	ADC中断标志
	ADC_DAT0_OV_IRQ_IF	ADC_DAT0超阈值中断标志位
	ADC_H_CONFLICT_IRQ_IF	硬件触发冲突标志
	ADC_S_CONFLICT_IRQ_IF	软件触发冲突标志
	ADC_EOS3_IRQ_IF	第四段扫描结束中断标志
	ADC_EOS2_IRQ_IF	第三段扫描结束中断标志
	ADC_EOS1_IRQ_IF	第二段扫描结束中断标志
	ADC_EOS0_IRQ_IF	第一段扫描结束中断标志
	ADC_ALL_IRQ_IF	ADC全部中断标志位——可用于初始化
输出参数:	无	

返回
返回值: 读对应标志位是否为1, 1位置位, 0为未置位

注解
其它说明: 无

警告

只有对应中断使能后，改为才能读取，如果对应中断未使能，读取结果一直为0

示例代码:

```
if (ADC_GetIRQFlag(ADC0, ADC_EOS0_IRQ_IF)) // 获取第一段扫描结束中断标志
{
    ADC_ClearIRQFlag(ADC0, ADC_EOS0_IRQ_IF) // 第一段扫描结束中断标志
}
```

Date	Version	Author	Description
2022年04月20日	1.0	Zhu Jie	创建

修改日志:

在文件 [lks32mc08x_adc.c](#) 第 227 行定义.

3.43.2.5 ADC_Init()

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

函数名称: void [ADC_Init\(ADC_TypeDef* ADCx, ADC_InitTypeDef* ADC_InitStruct\)](#)

功能描述: ADC初始化函数

参见

被引用内容: [SYS_AnalogModuleClockCmd\(\)](#)

参数

输入参数:	ADCx: ADC0 , ADC_InitTypeDef *ADC_InitTypeDef
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

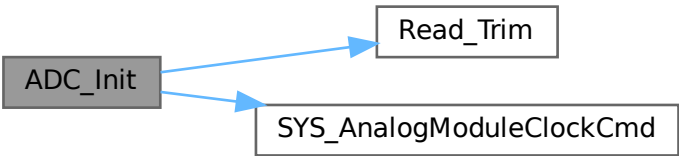
```
ADC_InitTypeDef ADC_InitStructure;
ADC_StructInit (&ADC_InitStructure); // 初始化结构体
ADC_InitStructure.ADC_CLK_DIV = ADC_Clock_48M; // ADC时钟选择48MHz
ADC_InitStructure.Align = ADC_LEFT_ALIGN; // ADC数据输出左对齐
ADC_InitStructure.Trigger_Mode = ADC_2SEG_TRG; // 设置ADC转换模式为双段式采样
ADC_InitStructure.FirSeg_Ch = ADC_2_TIMES_SAMPLE; // 第一段共采样2个通道
ADC_InitStructure.SecSeg_Ch = ADC_2_TIMES_SAMPLE; // 第二段共采样2个通道
ADC_InitStructure.ThrSeg_Ch = 0; // 第三段共采样0个通道
ADC_InitStructure.FouSeg_Ch = 0; // 第四段共采样0个通道
ADC_InitStructure.RefVol = RefVol_2V4; // ADC基准电压2.4V
ADC_InitStructure.Gain0 = 0x0; // ADC.DAT0-ADC.DAT9 (BIT0~BIT9) 采样保持电
路增益设置2/3
ADC_InitStructure.Gain1 = 0x0; // ADC.DAT10-ADC.DAT19 (BIT10~BIT19) 采样保
持电路增益设置2/3
ADC_InitStructure.Con_Sample = DISABLE; // 关闭连续采样
// 打开 UTIMER T0 T1 硬件触发采样
ADC_InitStructure.MCPWM_Trigger_En = ADC_MCPWM_T0_TRG|ADC_MCPWM_T1_TRG; // 开启UTIMER硬件第一段和第二段触发采样
ADC_InitStructure.UTIMER_Trigger_En = DISABLE; // 关闭UTIMER硬件触发采样
ADC_InitStructure.Trigger_Cnt = 0; // 发生(Trigger_Cnt+1)次触发事件执行一段转换
动作
ADC_InitStructure.IE = ADC_EOS0_IRQ_EN | ADC_EOS1_IRQ_EN; // ADC第一、二段采样结束中断使能
ADC_Init (ADC0, &ADC_InitStructure);
```

Date	Version	Author	Description
2015年11月26日	1.0	Howlet Li	创建

修改日志:

在文件 [lks32mc08x_adc.c](#) 第 59 行定义.

函数调用图:



3.43.2.6 ADC_SoftTrgEN()

```
void ADC_SoftTrgEN (
    ADC_TypeDef * ADCx,
    FuncState state )
```

函数名称: void [ADC_SoftTrgEN\(ADC_TypeDef* ADCx, FuncState state\)](#)

功能描述: ADC采样软件触发使能

参见

被引用内容: FuncState

参数

输入参数:	ADCx, 08系列固定选择ADC0 state: ENABLE触发, DISABLE不触发
输出参数:	无

返回

返回 值: 无

注解

其它说明: ADC0_SWT写0x5AA5仅触发一次, 完成后自动清零, 等待下次软件触发

警告

无

示例代码:

```
ADC_SoftTrgEN(ADC0, ENABLE); //软件触发一次ADC采样
```

Date	Version	Author	Description
2022年04月20日	1.0	Zhu Jie	创建

修改日志:

在文件 lks32mc08x_adc.c 第 399 行定义.

3.43.2.7 ADC_StructInit()

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

函数名称: void ADC_StructInit(ADC_InitTypeDef* ADC_InitStruct)

功能描述: ADC结构体初始化

参见

被引用内容: 无

参数

输入参数:	ADC_InitTypeDef *ADC_InitStruct
输出参数:	无

返回

返回 值: 无

注解

其它说明: 无

警告

无

示例代码:

```
ADC\_InitTypeDef ADC_InitStructure;  
ADC\_StructInit (&ADC_InitStructure); //初始化结构体
```

Date	Version	Author	Description
2015年11月26日	1.0	Howlet	修改

修改日志:

在文件 [lks32mc08x_adc.c](#) 第 136 行定义.

3.44 lks32mc08x_adc.c

[浏览该文件的文档.](#)

```
00001  
00020 #include "lks32mc08x_adc.h"  
00021  
00059 void ADC\_Init (ADC\_TypeDef *ADCx, ADC\_InitTypeDef *ADC_InitStruct)  
00060 {  
00061     uint16_t t_reg;  
00062  
00063     SYS\_AnalogModuleClockCmd (SYS\_AnalogModule\_ADC, ENABLE); // ADC模块使能  
00064  
00065     SYS\_WR\_PROTECT = 0x7A83; // 解除系统寄存器写保护  
00066     SYS\_AFE\_REG1 |= ADC\_InitStruct->RefVol << 3; // ADC基准电压选择  
00067     SYS\_AFE\_REG7 = ADC\_InitStruct->ADC\_CLK\_DIV << 4; // ADC时钟选择  
00068     SYS\_WR\_PROTECT = 0x0; /*关闭系统寄存器写操作*/  
00069  
00070     ADCx->IE = ADC\_InitStruct->IE;  
00071  
00072     t_reg = (ADC\_InitStruct->Align) | (ADC\_InitStruct->DAT0\_TH\_Type << 1);  
00073     ADCx->CFG = t_reg;  
00074  
00075     t_reg = (ADC\_InitStruct->MCPWM\_Trigger\_En) | (ADC\_InitStruct->UTIMER\_Trigger\_En << 4) |
```

```

00076         (ADC.InitStruct->Trigger.Cnt << 8) | (ADC.InitStruct->Trigger.Mode << 12) |
00077         (ADC.InitStruct->Con.Sample << 14);
00078
00079     ADCx->TRG = t_reg; // TRIG
00080
00081     ADCx->CHNT0 = ADC.InitStruct->FirSeg.Ch | (ADC.InitStruct->SecSeg.Ch << 8);
00082     ADCx->CHNT1 = ADC.InitStruct->ThrSeg.Ch | (ADC.InitStruct->FouSeg.Ch << 8);
00083
00084     ADCx->DAT0.TH = ADC.InitStruct->DAT0.TH;
00085
00086     ADCx->GAIN_0 = ADC.InitStruct->Gain0;
00087
00088     ADCx->GAIN_1 = ADC.InitStruct->Gain1;
00089
00090     if (ADC.InitStruct->Align == 0)
00091     {
00092         ADCx->DC_A0 = Read.Trim(0x00000310) << 4;
00093         ADCx->DC_A1 = Read.Trim(0x00000314) << 4;
00094         ADCx->AMC_A0 = Read.Trim(0x00000318);
00095         ADCx->AMC_A1 = Read.Trim(0x0000031C);
00096
00097         ADCx->DC_B0 = Read.Trim(0x00000320) << 4;
00098         ADCx->DC_B1 = Read.Trim(0x00000324) << 4;
00099         ADCx->AMC_B0 = Read.Trim(0x00000328);
00100         ADCx->AMC_B1 = Read.Trim(0x0000032C);
00101     }
00102     else
00103     {
00104         ADCx->DC_A0 = Read.Trim(0x00000310);
00105         ADCx->DC_A1 = Read.Trim(0x00000314);
00106         ADCx->AMC_A0 = Read.Trim(0x00000318);
00107         ADCx->AMC_A1 = Read.Trim(0x0000031C);
00108
00109         ADCx->DC_B0 = Read.Trim(0x00000320);
00110         ADCx->DC_B1 = Read.Trim(0x00000324);
00111         ADCx->AMC_B0 = Read.Trim(0x00000328);
00112         ADCx->AMC_B1 = Read.Trim(0x0000032C);
00113     }
00114 }
00115
00136 void ADC.StructInit(ADC.InitTypeDef *ADC.InitStruct)
00137 {
00138     ADC.InitStruct->IE = 0;
00139     ADC.InitStruct->Align = 0;
00140     ADC.InitStruct->Con.Sample = 0;
00141     ADC.InitStruct->Trigger.Cnt = 0;
00142     ADC.InitStruct->FirSeg.Ch = 0;
00143     ADC.InitStruct->SecSeg.Ch = 0;
00144     ADC.InitStruct->ThrSeg.Ch = 0;
00145     ADC.InitStruct->FouSeg.Ch = 0;
00146     ADC.InitStruct->Trigger.Mode = 0;
00147     ADC.InitStruct->MCPWM.Trigger.En = 0;
00148     ADC.InitStruct->UTIMER.Trigger.En = 0;
00149     ADC.InitStruct->DAT0.TH.Type = 0;
00150     ADC.InitStruct->Gain0 = 0;
00151     ADC.InitStruct->Gain1 = 0;
00152     ADC.InitStruct->RefVol = 0;
00153     ADC.InitStruct->ADC.CLK.DIV = 0;
00154 }
00155
00189 void ADC.ClearIRQFlag(ADC.TypeDef *ADCx, uint16_t INT.flag)
00190 {
00191     ADCx->IF = INT.flag;
00192 }
00193
00227 u16 ADC.GetIRQFlag(ADC.TypeDef *ADCx, uint16_t INT.flag)
00228 {
00229     u8 ADCIRQFlag = 0;
00230     if (ADCx->IF & INT.flag & ADCx->IE)
00231     {
00232         ADCIRQFlag = 1;
00233     }
00234     else
00235     {
00236         ADCIRQFlag = 0;
00237     }
00238     return ADCIRQFlag;
00239 }
00240
00264 void ADC.ChannelConfig(ADC.TypeDef *ADCx, CHNx CHNum, uint16_t nChannel)
00265 {
00266     u32 *p;
00267     u32 reg;
00268     p = (u32 *)&ADCx->CHN0;
00269     reg = p[CHNum >> 1];
00270     reg &= ~(0xff << (8 * (CHNum & 1)));
00271     reg |= nChannel << (8 * (CHNum & 1));

```

```
00272     p[CHNum >> 1] = reg;
00273 }
00274
00295 s16 ADC_GetConversionValue(DATx DATNum)
00296 {
00297     if (DATNum == DAT0)
00298     {
00299         return ADC0_DAT0;
00300     }
00301     else if (DATNum == DAT1)
00302     {
00303         return ADC0_DAT1;
00304     }
00305     else if (DATNum == DAT2)
00306     {
00307         return ADC0_DAT2;
00308     }
00309     else if (DATNum == DAT3)
00310     {
00311         return ADC0_DAT3;
00312     }
00313     else if (DATNum == DAT4)
00314     {
00315         return ADC0_DAT4;
00316     }
00317     else if (DATNum == DAT5)
00318     {
00319         return ADC0_DAT5;
00320     }
00321     else if (DATNum == DAT6)
00322     {
00323         return ADC0_DAT6;
00324     }
00325     else if (DATNum == DAT7)
00326     {
00327         return ADC0_DAT7;
00328     }
00329     else if (DATNum == DAT8)
00330     {
00331         return ADC0_DAT8;
00332     }
00333     else if (DATNum == DAT9)
00334     {
00335         return ADC0_DAT9;
00336     }
00337     else if (DATNum == DAT10)
00338     {
00339         return ADC0_DAT10;
00340     }
00341     else if (DATNum == DAT11)
00342     {
00343         return ADC0_DAT11;
00344     }
00345     else if (DATNum == DAT12)
00346     {
00347         return ADC0_DAT12;
00348     }
00349     else if (DATNum == DAT13)
00350     {
00351         return ADC0_DAT13;
00352     }
00353     else if (DATNum == DAT14)
00354     {
00355         return ADC0_DAT14;
00356     }
00357     else if (DATNum == DAT15)
00358     {
00359         return ADC0_DAT15;
00360     }
00361     else if (DATNum == DAT16)
00362     {
00363         return ADC0_DAT16;
00364     }
00365     else if (DATNum == DAT17)
00366     {
00367         return ADC0_DAT17;
00368     }
00369     else if (DATNum == DAT18)
00370     {
00371         return ADC0_DAT18;
00372     }
00373     else //(DATNum == DAT19)
00374     {
00375         return ADC0_DAT19;
00376     }
00377 }
00378
```

```

00399 void ADC_SoftTrgEN(ADC_TypeDef *ADCx, FuncState state)
00400 {
00401     if (state == ENABLE)
00402     {
00403         ADCx->SWT = 0x5AA5;
00404     }
00405 }
00406
00407 /***** (C) COPYRIGHT LINKO SEMICONDUCTOR *****END OF FILE*****/

```

3.45 source/lks32mc08x_can.c 文件参考

文件名称: [lks32mc08x_can.c](#)

文件标识: 无

内容摘要: CAN外设驱动程序

其它说明: 无

#include "lks32mc08x_can.h"

lks32mc08x_can.c 的引用(Include)关系图:



函数

- void [CAN_Init](#) (CAN_TypeDef *CANx, CAN_InitTypeDef *CAN_InitStruct)
函数名称: void [CAN_Init\(CAN_TypeDef* CANx, CAN_InitTypeDef* CAN_InitStruct\)](#)
- void [CAN_StructInit](#) (CAN_InitTypeDef *CAN_InitStruct)
函数名称: void [CAN_StructInit\(CAN_InitTypeDef *CAN_InitStruct\)](#)
- void [CAN_Sleep](#) (u32 Baud)
函数名称: void [CAN_Sleep\(u32 Baud\)](#)
- void [CAN_Manual_Awake](#) (void)
函数名称: void [CAN_Manual_Awake\(void\)](#)
- void [SFF_ID_Filter](#) (u32 acr, u32 amr, u8 rtr_acr, u8 rtr_amr, u8 Byte1_acr, u8 Byte1_amr, u8 Byte2_acr, u8 Byte2_amr)
函数名称: void [SFF_ID_Filter\(u32 acr,u32 amr,u8 rtr_acr,u8 rtr_amr, u8 Byte1_acr,u8 Byte1_amr,u8 Byte2_acr,u8 Byte2_amr\)](#)
- void [EFF_ID_Filter](#) (u32 acr, u32 amr, u8 rtr_acr, u8 rtr_amr)
函数名称: void [EFF_ID_Filter\(u32 acr,u32 amr,u8 rtr_acr,u8 rtr_amr\)](#)
- void [ID1_Filter_Dual](#) (u32 acr1, u32 amr1, u8 rtr_acr1, u8 rtr_amr1, u8 Byte_acr1, u8 Byte_amr1, u8 ide)
函数名称: void [ID1_Filter_Dual\(u32 acr1 ,u32 amr1,u8 rtr_acr1, u8 rtr_amr1,u8 Byte_acr1,u8 Byte_amr1,u8 ide\)](#)
- void [ID2_Filter_Dual](#) (u32 acr2, u32 amr2, u8 rtr_acr2, u8 rtr_amr2, u8 ide)
函数名称: void [ID2_Filter_Dual\(u32 acr2 ,u32 amr2,u8 rtr_acr2,u8 rtr_amr2 ,u8 ide\)](#)
- u8 [My_CAN_Send_Msg](#) (u32 id, u8 ide, u8 rtr, u8 *msg, u8 len)
函数名称: u8 [My_CAN_Send_Msg\(u32 id,u8 ide,u8 rtr,const u8 *msg,u8 len\)](#)
- u8 [CAN_Receive_Msg](#) (uint32_t *id, u8 *ide, u8 *rtr, u8 *buf)
函数名称: u8 [CAN_Receive_Msg\(uint32_t* id,u8* ide,u8* rtr,u8 *buf\)](#)
- uint8_t [CAN_ReadState](#) (CAN_TypeDef *CANx, uint32_t sr)
函数名称: uint8_t [CAN_ReadState\(CAN_TypeDef* CANx , uint32_t sr\)](#)
- enumLossposition [CAN_ReadALC](#) (void)

- 函数名称: `enumLossposition CAN_ReadALC(void)`
- `uint8_t CAN_ReadRXERRCNT (void)`
函数名称: `uint8_t CAN_ReadRXERRCNT(void)`
- `uint8_t CAN_ReadTXERRCNT (void)`
函数名称: `uint8_t CAN_ReadTXERRCNT(void)`
- `void CAN_ReceFramInfor (uint8_t *cnt, uint8_t *rbsa)`
函数名称: `void CAN_ReceFramInfor(uint8_t* cnt , uint8_t* rbsa )`
- `void CAN_ReadErrorCode (enumErrortype *errtype, enumErrortime *errtim, enumErrposition *errpos)`
函数名称: `void CAN_ReadErrorCode(enumErrortype* errtype , enumErrortime* errtim , enumErrposition* errpos)`
- `uint32_t CAN_GetIRQFlag (void)`
函数名称: `uint32_t CAN_GetIRQFlag(void)`

3.45.1 详细描述

文件名称: `lks32mc08x.can.c`
文件标识: 无
内容摘要: CAN外设驱动程序
其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2022年4月29日	1.0	HuangMG	创建

修改日志:

在文件 `lks32mc08x.can.c` 中定义.

3.45.2 函数说明

3.45.2.1 CAN_GetIRQFlag()

```
uint32_t CAN_GetIRQFlag (
    void )
```

函数名称: `uint32_t CAN_GetIRQFlag(void)`

功能描述: 获取中断标志与清除中断标志位

参见

被应用枚举: `enumErrortype` , `enumErrortime` , `enumErrposition`

参数

输入参数:	无
输出参数:	无

返回

返回 值: TIM中断标志

注解

其它说明: CAN中断标志位为读清除

警告

无

示例代码:

```
u8 CAN_SR 0;  
CAN_SR = CAN_GetIRQFlag(); //获取CAN中断标志位且清除CAN中断标志位
```

Date	Version	Author	Description
2022年04月29日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x-can.c](#) 第 828 行定义.

3.45.2.2 CAN_Init()

```
void CAN_Init (  
    CAN_TypeDef * CANx,  
    CAN_InitTypeDef * CAN_InitStruct )
```

函数名称: void [CAN_Init\(CAN_TypeDef* CANx, CAN_InitTypeDef* CAN_InitStruct\)](#)

功能描述: ADC初始化函数

参见

被调用函数: [SYS_ModuleClockCmd\(\)](#)

参数

输入参数:	CANx: CAN , CAN_InitTypeDef
输出参数:	无

返回

返回 值: 无

注解

其它说明: 无

警告

无

示例代码:

```
CAN_InitTypeDef CAN_InitStructure;  
CAN_StructInit(&CAN_InitStructure);  
//Can波特率:500Kbps  
CAN_InitStructure.CAN_Btr0 = 0x05;  
CAN_InitStructure.CAN_Btr1 = 0x1C;  
CAN_InitStructure.CAN_DMAEn = DISABLE;           //无DMA搬运数据  
CAN_InitStructure.CAN_WorkMode = CAN_NORMAL_MODE; //CAN正常收发工作模式  
CAN_InitStructure.CAN_ErrWarThre = 50;           //错误与警告门限值为50  
//开启接收中断使能,发送完毕当前帧中断使能,错误报警中断使能  
CAN_InitStructure.IRQEna = CAN_IE_RFIFONEMPTY | CAN_IE_TXDONE | CAN_IE_WERR;  
CAN_Init(CAN, &CAN_InitStructure);
```

Date	Version	Author	Description
2022年01月16日	1.0	HuangMG	创建

修改日志:

在文件 lks32mc08x_can.c 第 46 行定义.

函数调用图:



3.45.2.3 CAN_Manual_Awake()

```
void CAN_Manual_Awake (  
    void )
```

函数名称: void [CAN_Manual_Awake\(void\)](#)

功能描述: CAN手动休眠唤醒函数

参见

被调用函数: 无

参数

输入参数:	无
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
CAN\_Manual\_Awake\(\); //唤醒CAN休眠
```

Date	Version	Author	Description
2022年01月16日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x-can.c](#) 第 152 行定义.

3.45.2.4 CAN_ReadALC()

```
enumLossposition CAN_ReadALC (  
    void )
```

函数名称: `enumLossposition CAN_ReadALC(void)`

功能描述: 获取仲裁丢失的具体位置

参见

被应用枚举: `enumLossposition`

参数

输入参数:	无
输出参数:	无

返回

返回值: 返回仲裁丢失具体位置 `enumLossposition`

注解

其它说明: 无

警告

无

示例代码:

```
if (CAN_ReadALC() == CAN_LOST_ID0) //总线仲裁丢失在ID的第一位
{
}
```

Date	Version	Author	Description
2022年04月29日	1.0	HuangMG	创建

修改日志:

在文件 `lks32mc08x_can.c` 第 695 行定义.

3.45.2.5 CAN_ReadErrorCode()

```
void CAN_ReadErrorCode (
    enumErrortype * errtype,
    enumErrortime * errtim,
    enumErrposition * errpos )
```

函数名称: `void CAN_ReadErrorCode(enumErrortype* errtype , enumErrortime* errtim , enumErrposition* errpos)`

功能描述: 获取总线错误类型和错误位置

参见

被应用枚举: [enumErrortype](#) , [enumErrortime](#) , [enumErrposition](#)

参数

输入参数:	errtype: 总线错误类型 errtim: 总线错误发生时刻 errpos: 错误发生的位置
输出参数:	无

返回

返回 值: 无

注解

其它说明: 无

警告

无

示例代码:

```
u8 err_type = 0 , err_time = 0 , err_location = 0; //存储变量申请
CAN_ReadErrorCode(err_type,err_time,err_location);
```

Date	Version	Author	Description
2022年04月29日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_can.c](#) 第 801 行定义.

3.45.2.6 CAN_ReadRXERRCNT()

```
uint8_t CAN_ReadRXERRCNT (
    void )
```

函数名称: `uint8_t CAN_ReadRXERRCNT(void)`

功能描述: 获取接收错误计数值

参见

被应用枚举: [enumLossposition](#)

参数

输入参数:	无
输出参数:	无

返回

返回 值: 返回接收错误计数值

注解

其它说明: 无

警告

无

示例代码:

```
u8 err_value = 0;
err_value = CAN_ReadRXERRCNT(); //获取错误计数值个数
```

Date	Version	Author	Description
2022年04月29日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x-can.c](#) 第 720 行定义.

3.45.2.7 CAN_ReadState()

```
uint8_t CAN_ReadState (
    CAN_TypeDef * CANx,
    uint32_t sr )
```

函数名称: `uint8_t CAN_ReadState(CAN_TypeDef* CANx , uint32_t sr)`

功能描述: 获取CAN状态

参见

被调用函数: 无

参数

输入参数:	sr: 读取指定状态或者所有状态结果 <table border="1"> <tr> <td>CAN_ONBUS</td><td>1: CAN 模块, 处于 BUS OFF 状态, 无数据发送接收动作 0: CAN 模块, 可以发送数据到 CAN 总线, 或接收 CAN 总线数据</td></tr> <tr> <td>CAN_ERROV</td><td>1: CAN 传输产生的错误总数达到或超过 CAN_EWL 规定值 0: CAN 传输产生的错误总数低于 CAN_EWL 规定值</td></tr> <tr> <td>CAN_TXING</td><td>1: CAN 模块正在发送一帧数据 0: CAN 模块没有发送数据的动作</td></tr> <tr> <td>CAN_RXING</td><td>1: CAN 模块正在接收一帧数据 0: CAN 模块没有接收数据的动作</td></tr> <tr> <td>CAN_TRNASDONE</td><td>1: 最近一次传输已完成 0: 最近一次传输未完成</td></tr> <tr> <td>CAN_TFIFOEMPTY</td><td>1: TFIFO 空, 可以写入发送数据 0: TFIFO 非空, 内部数据没有发送完毕</td></tr> <tr> <td>CAN_RFIFOEMPTY</td><td>1: RFIFO 存入的帧太多, 已满, 导致数据丢失 0: RFIFO 未滿</td></tr> <tr> <td>CAN_DATAAVAIL</td><td>1: RFIFO 存有一帧或多帧数据, 可以通过 RFIFO 寄存器读取 0: RFIFO 没有有效帧数据</td></tr> <tr> <td>CAN_ALLSR</td><td>读取全部状态</td></tr> </table>	CAN_ONBUS	1: CAN 模块, 处于 BUS OFF 状态, 无数据发送接收动作 0: CAN 模块, 可以发送数据到 CAN 总线, 或接收 CAN 总线数据	CAN_ERROV	1: CAN 传输产生的错误总数达到或超过 CAN_EWL 规定值 0: CAN 传输产生的错误总数低于 CAN_EWL 规定值	CAN_TXING	1: CAN 模块正在发送一帧数据 0: CAN 模块没有发送数据的动作	CAN_RXING	1: CAN 模块正在接收一帧数据 0: CAN 模块没有接收数据的动作	CAN_TRNASDONE	1: 最近一次传输已完成 0: 最近一次传输未完成	CAN_TFIFOEMPTY	1: TFIFO 空, 可以写入发送数据 0: TFIFO 非空, 内部数据没有发送完毕	CAN_RFIFOEMPTY	1: RFIFO 存入的帧太多, 已满, 导致数据丢失 0: RFIFO 未滿	CAN_DATAAVAIL	1: RFIFO 存有一帧或多帧数据, 可以通过 RFIFO 寄存器读取 0: RFIFO 没有有效帧数据	CAN_ALLSR	读取全部状态
CAN_ONBUS	1: CAN 模块, 处于 BUS OFF 状态, 无数据发送接收动作 0: CAN 模块, 可以发送数据到 CAN 总线, 或接收 CAN 总线数据																		
CAN_ERROV	1: CAN 传输产生的错误总数达到或超过 CAN_EWL 规定值 0: CAN 传输产生的错误总数低于 CAN_EWL 规定值																		
CAN_TXING	1: CAN 模块正在发送一帧数据 0: CAN 模块没有发送数据的动作																		
CAN_RXING	1: CAN 模块正在接收一帧数据 0: CAN 模块没有接收数据的动作																		
CAN_TRNASDONE	1: 最近一次传输已完成 0: 最近一次传输未完成																		
CAN_TFIFOEMPTY	1: TFIFO 空, 可以写入发送数据 0: TFIFO 非空, 内部数据没有发送完毕																		
CAN_RFIFOEMPTY	1: RFIFO 存入的帧太多, 已满, 导致数据丢失 0: RFIFO 未滿																		
CAN_DATAAVAIL	1: RFIFO 存有一帧或多帧数据, 可以通过 RFIFO 寄存器读取 0: RFIFO 没有有效帧数据																		
CAN_ALLSR	读取全部状态																		
输出参数:	无																		

返回

返回 值: 返回指定状态结果或者所有状态结果

CAN_ONBUS	1: CAN 模块, 处于 BUS OFF 状态, 无数据发送接收动作 0: CAN 模块, 可以发送数据到 CAN 总线, 或接收 CAN 总线数据
CAN_ERROV	1: CAN 传输产生的错误总数达到或超过 CAN_EWL 规定值 0: CAN 传输产生的错误总数低于 CAN_EWL 规定值
CAN_TXING	1: CAN 模块正在发送一帧数据 0: CAN 模块没有发送数据的动作
CAN_RXING	1: CAN 模块正在接收一帧数据 0: CAN 模块没有接收数据的动作
CAN_TRNASDONE	1: 最近一次传输已完成 0: 最近一次传输未完成
CAN_TFIFOEMPTY	1: TFIFO 空, 可以写入发送数据 0: TFIFO 非空, 内部数据没有发送完毕
CAN_RFIFOEMPTY	1: RFIFO 存入的帧太多, 已满, 导致数据丢失 0: RFIFO 未滿
CAN_DATAAVAIL	1: RFIFO 存有一帧或多帧数据, 可以通过 RFIFO 寄存器读取 0: RFIFO 没有有效帧数据
CAN_ALLSR	读取全部状态

注解

其它说明： 无

警告

无

示例代码:

```
//判断CAN的RFIFO是否存在一帧或多帧数据，此时是否可以通过RFIFO寄存器读取
if(CAN_ReadState(CAN , CAN_DATAAVAIL) == CAN_DATAAVAIL)
{
}
```

Date	Version	Author	Description
2022年04月29日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_can.c](#) 第 669 行定义.

3.45.2.8 CAN_ReadTXERRCNT()

```
uint8_t CAN_ReadTXERRCNT (
    void )
```

函数名称: [uint8_t CAN_ReadTXERRCNT\(void\)](#)

功能描述: 获取发送错误计数值

参见

被应用枚举: [enumLossposition](#)

参数

输入参数:	无
输出参数:	无

返回

返回 值: 返回发送错误计数值

注解

其它说明： 无

警告

无

示例代码:

```
u8 err_value = 0;
err_value = CAN_ReadTXERRCNT(); //获取错误计数值个数
```

Date	Version	Author	Description
2022年04月29日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x.can.c](#) 第 745 行定义.

3.45.2.9 CAN_ReceFramInfor()

```
void CAN_ReceFramInfor (
    uint8_t * cnt,
    uint8_t * rbsa )
```

函数名称: void [CAN_ReceFramInfor\(uint8_t* cnt , uint8_t* rbsa \)](#)

功能描述: 获取接收缓冲区有效接收帧数与第一帧有效接收数据在 **RFIFO**的位置

参见

被应用枚举: [enumLossposition](#)

参数

输入参数:	无
输出参数:	cnt: 有效帧个数 rbsa: RFIFO 的大小为 32 字节，rbsa存储第一帧有效接收数据在 RFIFO 的位置0-31

返回

返回值: 无

注解

其它说明： 无

警告

无

示例代码:

```
u8 frames_number = 0 ,  RFIFO_location = 0;//存储变量申请
CAN_ReceFramInfor(frames_number, RFIFO_location);
//frames_number接收缓冲区存储帧个数, RFIFO_location缓冲区最早读取的帧位置
```

Date	Version	Author	Description
2022年04月29日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_can.c](#) 第 772 行定义.

3.45.2.10 CAN_Receive_Msg()

```
u8 CAN_Receive_Msg (
    uint32_t * id,
    u8 * ide,
    u8 * rtr,
    u8 * buf )
```

函数名称: u8 [CAN_Receive_Msg\(uint32_t* id,u8* ide,u8* rtr,u8 *buf\)](#)

功能描述: CAN轮询接收数据程序

参见

被调用函数： 无

参数

输入参数:	无
输出参数:	id: 接收帧id ide: 接收帧的类型，标准/扩展帧 rtr: 接收帧的类型，远程帧/数据帧 *buf: 接收帧的数据存储缓存区

返回

返 回 值： 0: 接收完成 !0: 接收失败

注解

其它说明： 1、id:标准ID(11位)/扩展ID(11位+18位)
2、ide:0,标准帧;1,扩展帧
3、rtr:0,数据帧;1,远程帧。

警告

无

示例代码:

```
//can_par.id存储接收帧ID, can_par.ide存储接收帧类型标准/扩展帧,  
can_par.rtr存储接收帧类型远程帧/数据帧, can_par.RX存储接收帧数据  
CAN_Receive_Msg (&( can_par.id), &(can_par.ide), &(can_par.rtr), can_par.RX) ;
```

Date	Version	Author	Description
2018年08月10日	1.0	ZKW	创建

修改日志:

在文件 [lks32mc08x_can.c](#) 第 549 行定义.

3.45.2.11 CAN.Sleep()

```
void CAN_Sleep (  
    u32 Baud )
```

函数名称: void [CAN.Sleep\(u32 Baud\)](#)

功能描述: CAN开启休眠函数

参见

被调用函数: 无

参数

输入参数:	Baud: 波特率, 单位Kbps
输出参数:	无

返回

返回 值: 无

注解

其它说明：休眠唤醒有两种：1、软件清零CAN_MOD[4]，即自动退出休眠模式。，2、一旦检测到CAN总线的活动，即自动退出休眠模式。

警告

无

示例代码:

```
CAN_Sleep(500); //开启CAN休眠, CAN比特率500kbps
```

Date	Version	Author	Description
2022年01月16日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_can.c](#) 第 115 行定义.

3.45.2.12 CAN_StructInit()

```
void CAN_StructInit (  
    CAN_InitTypeDef * CAN_InitStruct )
```

函数名称: void [CAN_StructInit\(CAN_InitTypeDef *CAN_InitStruct\)](#)

功能描述: CAN结构体初始化

参见

被调用函数: 无

参数

输入参数:	CAN_InitTypeDef
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
CAN_InitTypeDef CAN_InitStructure;  
CAN_StructInit(&CAN_InitStructure); //初始化结构体
```

Date	Version	Author	Description
2022年01月16日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_can.c](#) 第 84 行定义.

3.45.2.13 EFF_ID_Filter()

```
void EFF_ID_Filter (  
    u32 acr,  
    u32 amr,  
    u8 rtr_acr,  
    u8 rtr_amr )
```

函数名称: void [EFF_ID_Filter\(u32 acr,u32 amr,u8 rtr_acr,u8 rtr_amr\)](#)

功能描述: CAN扩展帧单ID过滤,采用2.0B协议

参见

被调用函数: 无

参数

输入参数:	acr: ID值。 amr: 屏蔽位 rtr_acr: 接收（1远程/0数据帧）帧类型选择 rtr_amr: 远程/数据帧滤波屏蔽位，0表示只接收rtr_acr类型帧，为1表示远程/0标志帧均接收
输出参数:	无

返回

返回 值: 无

注解

其它说明： 1、扩展帧 29位ID长度
2、amr：屏蔽位对应位0表示此为对应ID必须匹配才会接收

警告

无

示例代码:

```
示例1:
//扩展帧接收滤波: ID:0X10005A5A ,数据帧
EFF_ID_Filter(0X10005A5A, 0x00, 0x0, 0x0);
//接收的扩展帧只有满足ID地址为0X10005A5A且为数据帧才会接收

示例2:
//扩展帧接收滤波: ID:0X10005A5A ,数据帧和远程帧 (因为rtr.amr为1, 所以接收帧类型不需要匹配也可以接收)
EFF_ID_Filter(0X10005A5A, 0x00, 0x0, 0x1);
//接收的扩展帧只有满足ID地址为0X10005A5A的数据帧和远程帧才会接收
```

Date	Version	Author	Description
2022年01月16日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_can.c](#) 第 252 行定义.

3.45.2.14 ID1_Filter_Dual()

```
void ID1_Filter_Dual (
    u32 acr1,
    u32 amr1,
    u8 rtr_acr1,
    u8 rtr_amr1,
    u8 Byte_acr1,
    u8 Byte_amr1,
    u8 ide )
```

函数名称: void ID1_Filter_Dual(u32 acr1 ,u32 amr1,u8 rtr_acr1, u8 rtr_amr1,u8 Byte_acr1,u8 Byte_amr1,u8 ide)

功能描述: CAN双ID过滤中ID1滤波设置 采用2.0B协议

参见

被调用函数: 无

参数

输入参数:	acr1: ID1值 (标志帧ID11位, 扩展帧ID范围1D28~ID13即ID高16位有效) amr1: ID1屏蔽位 rtr_acr1: 接收 (1远程/0数据帧) 帧类型选择 (只支持SFF模式, 扩展帧配置无效) rtr_amr1: 远程/标志帧滤波屏蔽位, 0表示只接收rtr_acr类型帧, 为1表示远程/0标志帧均接收 (只支持SFF模式, 扩展帧配置无效) Byte_acr1: 数据帧第1个Byte低4位滤波选择 (只支持SFF模式, 扩展帧配置无效) Byte_amr1: 数据帧第1个Byte低4位屏蔽位 (只支持SFF模式, 扩展帧配置无效) ide: 0标准(SFF模式)/1扩展帧
输出参数:	无

返回

返回 值: 无

注解

其它说明: 1、SFF模式为标准帧模式
2、扩展帧时, 只对ID的bit13- bit28即ID高16位进行筛选过滤
3、因为双滤波ID1函数兼容标志准和扩展帧滤波 (标准帧和扩展帧的双滤波有区别), 所以该函数的形参部分只支持标准帧模式配置使用, 扩展帧模式配置无效。

警告

无

示例代码:

```
示例1:
//标准帧ID1接收滤波: ID:0X75A ,数据帧
ID1_Filter_Dual(0X75A, 0x00, 0x0, 0x0, 0x0F, 0x00, 0);
//接收的标准帧只有满足ID地址为0X75A且为数据帧且接收数据 (不是帧的第一个数据, 而是数据区的第一个数据) 的第1个
Byte低4位为0XF才会接收。
示例2:
//扩展帧ID1接收滤波: ID:0XFECB的扩展帧
ID1_Filter_Dual(0XFECB, 0x00, 0x0, 0x0, 0x00, 0x00, 1);
//接收的扩展帧只有满足ID地址为0XFECB才会接收
```

Date	Version	Author	Description
2022年01月16日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_can.c](#) 第 319 行定义.

3.45.2.15 ID2.Filter_Dual()

```
void ID2.Filter_Dual (
    u32 acr2,
```

```
u32 amr2,
u8 rtr_acr2,
u8 rtr_amr2,
u8 ide )
```

函数名称: void ID2.Filter_Dual(u32 acr2 ,u32 amr2,u8 rtr_acr2,u8 rtr_amr2 ,u8 ide)

功能描述: CAN双ID过滤中ID2滤波设置 采用2.0B协议

参见
被调用函数: 无

参数

输入参数:	acr2: ID2值（范围1D28~ID13即ID高16位有效） amr2: ID2屏蔽位 rtr_acr2: 接收（1远程/0数据帧）帧类型选择（只支持SFF标准帧模式，扩展帧配置无效） rtr_amr2: 远程/标志帧滤波屏蔽位，0表示只接收rtr_acr类型帧，为1表示远程/0标志 帧均接收（只支持SFF标准帧模式，扩展帧配置无效）
输出参数:	无

返回
返回 值: 无

注解
其它说明: 1、SFF模式为标准帧模式
2、扩展帧时，只对ID的bit13- bit28即ID高16位进行筛选过滤
3、因为双滤波ID1函数兼容标志准和扩展帧滤波（标准帧和扩展帧的双滤波有区别），所以该函数的形参部分只支持标准帧模式配置使用，扩展帧模式配置无效。

警告
无

示例代码:

```
示例1:
//标准帧ID2接收滤波: ID:0X75A ,数据帧
ID2.Filter_Dual(0X75A, 0x00, 0x0, 0x0, 0);
//接收的标准帧只有满足ID地址为0X75A且为数据帧才会接收。

示例2:
//扩展帧ID2接收滤波: ID:0XFECB的扩展帧
ID2.Filter_Dual(0XFECB, 0x00, 0x0, 0x0, 1);
//接收的扩展帧只有满足ID地址为0XFECB才会接收
```

Date	Version	Author	Description
2022年01月16日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_can.c](#) 第 386 行定义.

3.45.2.16 My_CAN_Send_Msg()

```
u8 My_CAN_Send_Msg (
    u32 id,
    u8 ide,
    u8 rtr,
    u8 * msg,
    u8 len )
```

函数名称: u8 My_CAN_Send_Msg(u32 id,u8 ide,u8 rtr,const u8 *msg,u8 len)

功能描述: CAN轮询发送数据程序 采用2.0B协议

参见
被调用函数: 无

参数

输入参数:	id: 发送id ide: 标准/扩展帧 rtr: 远程帧/数据帧 *msg: 发送的数据 len:发送字节长度
输出参数:	无

返回

返回 值: 0: 发送成功 !0: 发送失败

注解

其它说明: 1、id:标准ID(11位)/扩展ID(11位+18位)
2、ide:0,标准帧;1,扩展帧
3、rtr:0,数据帧;1,远程帧。

警告
无

示例代码:

```
示例1:
u8 Can_TX[8] = {0x81, 0x23, 0x33, 0x44, 0x15, 0x26, 0x37, 0x48};
// 发送数据 ID:0x03,标准帧,数据帧,数据,数据长度8字节
My_CAN_Send_Msg(0x5A, 0, 0, Can_TX, 8);

示例2:
u8 Can_TX[8] = {0x81, 0x23, 0x33, 0x44, 0x15, 0x26, 0x37, 0x48};
// 发送数据 ID:0x03,扩展帧,数据帧,数据,数据长度8字节
My_CAN_Send_Msg(0X10005A5A, 1, 0, Can_TX, 8);
```

Date	Version	Author	Description
2018年08月10日	1.0	ZKW	创建

修改日志:

Date	Version	Author	Description
2022年01月16日	1.0	HuangMG	创建

修改日志:增加自测模式发送数据功能

在文件 [lks32mc08x_can.c](#) 第 455 行定义.

3.45.2.17 SFF_ID_Filter()

```
void SFF_ID_Filter (
    u32 acr,
    u32 amr,
    u8 rtr_acr,
    u8 rtr_amr,
    u8 Byte1_acr,
    u8 Byte1_amr,
    u8 Byte2_acr,
    u8 Byte2_amr )
```

函数名称: void [SFF_ID_Filter\(u32 acr,u32 amr,u8 rtr_acr,u8 rtr_amr, u8 Byte1_acr,u8 Byte1_amr,u8 Byte2_acr,u8 Byte2_amr\)](#)

功能描述: CAN标准帧单ID过滤,采用2.0B协议

参见

被调用函数: 无

参数

输入参数:	acr: ID值。 amr: 屏蔽位 rtr_acr: 接收（1远程/0数据帧）帧类型选择 rtr_amr: 远程/数据帧滤波屏蔽位，0表示只接收rtr_acr类型帧，为1表示远程/0标志帧均接收 Byte1_acr: 数据第一个字节 Byte1_amr: 第一个字节屏蔽位 Byte2_acr: 数据第二个字节 Byte2_amr: 第二个字节屏蔽位
输出参数:	无

返回

返回 值: 无

注解

其它说明: 1、标准帧 11位ID长度
2、amr: 屏蔽位对应位0表示此为对应ID必须匹配才会接收

警告

无

示例代码:

示例1:

```
//标准帧接收单滤波: ID:0x5A ,数据帧, BYTE1:0x81, BYTE:0x23
SFF_ID_Filter(0x5A, 0x00, 0x0, 0x0, 0x81, 0x00, 0x23, 0x00);
//接收的标准帧只有满足ID地址为0x5A的数据帧且第一个数据字节为0x81, 第二个字节为0x23才会接收
```

示例2:

```
//标准帧接收单滤波: ID:0x5A和0x5B (因为amr第一位为1, 所以第一位可以为1或0, 无需匹配即可接收) ,数据帧, BYTE1:0x81, BYTE:0x23
SFF_ID_Filter(0x5A, 0x01, 0x0, 0x0, 0x81, 0x00, 0x23, 0x00);
//接收的标准帧只有满足ID地址为0x5A和0x5B的数据帧且第一个数据字节为0x81, 第二个字节为0x23才会接收
```

Date	Version	Author	Description
2022年01月16日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_can.c](#) 第 196 行定义.

3.46 lks32mc08x_can.c

[浏览该文件的文档.](#)

```
00001
00015 #include "lks32mc08x_can.h"
00016
00046 void CAN_Init(CAN_TypeDef *CANx, CAN_InitTypeDef *CAN_InitStruct)
00047 {
00048     SYS_ModuleClockCmd(SYS_Module_CAN, ENABLE);
00049     CAN->MOD |= 1 << 0; // 复位模式
00050     if (CAN_InitStruct->CAN_DMAEn)
00051     {
00052         CANx->CMR |= BIT5;
00053     }
00054     else
00055     {
00056         CANx->CMR &= ~BIT5;
00057     }
00058     CANx->EWLR = CAN_InitStruct->CAN_ErrWarThre;
00059     CANx->BTR0 = CAN_InitStruct->CAN_Btr0;
00060     CANx->BTR1 = CAN_InitStruct->CAN_Btr1;
00061     CANx->MOD = CAN_InitStruct->CAN_WorkMode;
00062     CANx->EIR = CAN_InitStruct->IRQEna;
00063 }
00084 void CAN_StructInit(CAN_InitTypeDef *CAN_InitStruct)
```

```

00085 {
00086     CAN_InitStruct->CAN_DMAEn = DISABLE;
00087     CAN_InitStruct->CAN_WorkMode = CAN_NORMAL_MODE;
00088     CAN_InitStruct->CAN_ErrWarThre = 120;
00089     /*波特率选择500K*/
00090     CAN_InitStruct->CAN_Btr0 = 0x05;
00091     CAN_InitStruct->CAN_Btr1 = 0x1C;
00092     CAN_InitStruct->IRQEna = 0x0;
00093 }
00094
00115 void CAN_Sleep(u32 Baud)
00116 {
00117     u32 rate = 0, t_cnt = 0;
00118     CAN_MODE &= ~BIT0; // 正常输出模式
00119     rate = 1152000 / Baud; // 此值为主频96MHz进行计算, 主频48MHz写入的Baud适当加大
00120     for (t_cnt = 0; t_cnt < rate; t_cnt++) // 等待11个波特率周期, 进入正常工作模式
00121     {
00122         __NOP();
00123     }
00124     CAN_MODE |= BIT4;
00125     rate = 1632000 / Baud; // 此值为主频96MHz进行计算, 主频48MHz写入的Baud适当加大
00126     // 等待16个波特率周期, 正式进入休眠等待16个波特率周期, 正式进入休眠
00127     for (t_cnt = 0; t_cnt < rate; t_cnt++)
00128     {
00129         __NOP();
00130     }
00131 }
00132
00152 void CAN_Manual_Awake(void)
00153 {
00154     CAN_MODE &= ~BIT0;
00155     CAN_MODE &= ~BIT4;
00156 }
00157
00196 void SFF_ID_Filter(u32 acr, u32 amr, u8 rtr_acr, u8 rtr_amr, u8 Byte1_acr, u8 Byte1_amr, u8 Byte2_acr, u8
Byte2_amr)
00197 {
00198     CAN->MOD |= 1 << 0; // 复位模式
00199     CAN->MOD |= 1 << 3; // 单滤波ID, 一个 32 位长的滤波ID
00200
00201     acr &= 0x7ff;
00202     amr &= 0x7ff;
00203
00204     rtr_acr &= 0x1;
00205     rtr_amr &= 0x1;
00206
00207     CAN_ACR0 = (u8)(acr >> 3);
00208     CAN_ACR1 = (u8)(acr << 5) | (rtr_acr << 4);
00209     CAN_ACR2 = Byte1_acr;
00210     CAN_ACR3 = Byte2_acr;
00211
00212     CAN_AMR0 = (u8)(amr >> 3);
00213     CAN_AMR1 = (u8)(amr << 5) | (rtr_amr << 4) | 0x0f;
00214     CAN_AMR2 = Byte1_amr;
00215     CAN_AMR3 = Byte2_amr;
00216
00217     CAN->MOD &= ~(1 << 0); // 正常输出模式
00218 }
00219
00252 void EFF_ID_Filter(u32 acr, u32 amr, u8 rtr_acr, u8 rtr_amr)
00253 {
00254     CAN->MOD |= 1 << 0; // 复位模式
00255     CAN->MOD |= 1 << 3; // 单滤波ID, 一个 32 位长的滤波ID
00256
00257     acr &= 0x1fffffff;
00258     amr &= 0x1fffffff;
00259     rtr_acr &= 0x1;
00260     rtr_amr &= 0x1;
00261
00262     /*可接收的ID寄存器, 当输入帧的 ID 同此匹配, 则被 CAN 模块接受
    配合 AMR 寄存器, 可同一类 ID 匹配。*/
00263     CAN_ACR3 = (u8)(acr << 3) | (rtr_acr << 2);
00264     CAN_ACR2 = (u8)(acr >> 5);
00265     CAN_ACR1 = (u8)(acr >> 13);
00266     CAN_ACR0 = (u8)(acr >> 21);
00267     /*配合 ACR 寄存器, 同输入 ID 进行匹配。AMR 寄存器某一位为 0, 标识
    ID 对应位需要同 ACR 匹配; 为 1, 标识 ID 对应位不需要同 ACR 匹配*/
00268     CAN_AMR3 = (u8)((amr << 3) | (rtr_amr << 2) | 0x03);
00269     CAN_AMR2 = (u8)(amr >> 5);
00270     CAN_AMR1 = (u8)(amr >> 13);
00271     CAN_AMR0 = (u8)(amr >> 21);
00272
00273     CAN->MOD &= ~(1 << 0); // 正常输出模式
00274 }
00275
00319 void ID1_Filter_Dual(u32 acr1, u32 amr1, u8 rtr_acr1, u8 rtr_amr1, u8 Byte_acr1, u8 Byte_amr1, u8 ide)
00320 {

```

```

00321 CAN->MOD |= 1 << 0; // RESET MODE
00322 CAN->MOD &= ~(1 << 3); // 双滤波ID
00323 rtr_acr1 &= 0x1;
00324 rtr_amr1 &= 0x1;
00325
00326 if (ide) /*扩展帧*/
00327 {
00328     acr1 &= 0x1fffffff;
00329     CAN_ACR1 = (u8)(acr1 >> 13);
00330     CAN_ACR0 = (u8)(acr1 >> 21);
00331
00332     CAN_AMR1 = (u8)(amr1 >> 13);
00333     CAN_AMR0 = (u8)(amr1 >> 21);
00334 }
00335 else /*标准帧*/
00336 {
00337     acr1 &= 0x7ff;
00338     CAN_ACR0 = (u8)(acr1 >> 3);
00339     CAN_ACR1 = (u8)(acr1 << 5) | (rtr_acr1 << 4) | (Byte_acr1 >> 4);
00340     CAN_ACR3 &= 0xF0;
00341     CAN_ACR3 |= (u8)(Byte_acr1 & 0x0F);
00342
00343     CAN_AMR0 = (u8)(amr1 >> 3);
00344     CAN_AMR1 = (u8)(amr1 << 5) | (rtr_amr1 << 4) | (Byte_amr1 >> 4);
00345     CAN_AMR3 &= 0xF0;
00346     CAN_AMR3 |= (u8)(Byte_amr1 & 0x0F);
00347 }
00348 CAN->MOD &= ~(1 << 0); // 正常输出模式
00349 }
00350
00386 void ID2.FilterDual(u32 acr2, u32 amr2, u8 rtr_acr2, u8 rtr_amr2, u8 ide)
00387 {
00388     CAN->MOD |= 1 << 0; // RESET MODE
00389     CAN->MOD &= ~(1 << 3); // 双滤波ID
00390     rtr_acr2 &= 0x1;
00391     rtr_amr2 &= 0x1;
00392     if (ide) // externed
00393     {
00394         acr2 &= 0x1fffffff;
00395         CAN_ACR3 = (u8)(acr2 >> 13);
00396         CAN_ACR2 = (u8)(acr2 >> 21);
00397
00398         CAN_AMR3 = (u8)(amr2 >> 13);
00399         CAN_AMR2 = (u8)(amr2 >> 21);
00400     }
00401     else
00402     {
00403         acr2 &= 0x7ff;
00404         CAN_ACR3 &= 0x0f;
00405         CAN_ACR2 = (u8)(acr2 >> 3);
00406         CAN_ACR3 |= (u8)(acr2 << 5) | (rtr_acr2 << 4);
00407
00408         CAN_AMR3 &= 0x0f;
00409         CAN_AMR2 = (u8)(amr2 >> 3);
00410         CAN_AMR3 |= (u8)(amr2 << 5) | (rtr_amr2 >> 4);
00411     }
00412     CAN->MOD &= ~(1 << 0); // 正常输出模式
00413 }
00414
00455 u8 My_CAN_Send_Msg(u32 id, u8 ide, u8 rtr, u8 *msg, u8 len)
00456 {
00457     u8 frame_inf = 0;
00458     u16 i = 0;
00459
00460     if (CAN->SR & 0X04) // TFIFO 空
00461     {
00462         CAN->MOD &= ~(1 << 0); // 正常输出模式
00463         frame_inf |= ide << 7; // 0标准/1扩展帧
00464         frame_inf |= rtr << 6; // 0,数据帧;1,遥控帧
00465         frame_inf |= len << 0; // 发送数据长度
00466         CAN->TXRX0 = frame_inf; // 发送TX帧信息
00467         if (0 == ide) // 标准帧
00468         {
00469             id &= 0x7ff;
00470             // 发送接收寄存器1
00471             CAN->TXRX1 = id >> 3; // TX ID0
00472             CAN->TXRX2 = (id & 0X07) << 5; // TX ID1
00473             CAN->TXRX3 = msg[0]; // TX DATA0
00474             CAN->TXRX4 = msg[1]; // TX DATA1
00475             CAN->TXRX5 = msg[2]; // TX DATA2
00476             CAN->TXRX6 = msg[3]; // TX DATA3
00477             CAN->TXRX7 = msg[4]; // TX DATA4
00478             CAN->TXRX8 = msg[5]; // TX DATA5
00479             CAN->TXRX9 = msg[6]; // TX DATA6
00480             CAN->TXRXA = msg[7]; // TX DATA7
00481         }
00482         else // 扩展帧

```

```

00483     {
00484         id &= 0x1FFFFFFF;
00485         CAN->TXRX1 = (u8)(id >> 21); // TX ID0
00486         CAN->TXRX2 = (u8)((id >> 13) & 0xFF); // TX ID1
00487         CAN->TXRX3 = (u8)((id >> 5) & 0xFF); // TX ID2
00488         CAN->TXRX4 = (u8)(id & 0xFF) << 3; // TX ID3
00489         CAN->TXRX5 = msg[0]; // TX DATA0
00490         CAN->TXRX6 = msg[1]; // TX DATA1
00491         CAN->TXRX7 = msg[2]; // TX DATA2
00492         CAN->TXRX8 = msg[3]; // TX DATA3
00493         CAN->TXRX9 = msg[4]; // TX DATA4
00494         CAN->TXRXA = msg[5]; // TX DATA5
00495         CAN->TXRXB = msg[6]; // TX DATA6
00496         CAN->TXRXC = msg[7]; // TX DATA7
00497     }
00498     if ((CAN->MOD) & CAN_SELFTEST_MODE) /*CAN工作在自测模式*/
00499     {
00500         CAN->CMR |= 1 << 4; // CAN发送数据的同时也将数据接收回来
00501     }
00502     else
00503     {
00504         CAN->CMR |= 1 << 0; // CAN发送传输请求
00505     }
00506     while (!(CAN->SR & 0x08)) // 最近一次是否传输完成
00507     {
00508         if ((i++) >= 0xFF)
00509         {
00510             return 0xFF; // 发送失败
00511         }
00512     };
00513     return 0; // 发送完成
00514 }
00515 else // TFIFO非空
00516 {
00517     return 0xFF; // 发送失败
00518 }
00519 }
00520
00549 u8 CAN_Receive_Msg(uint32_t *id, u8 *ide, u8 *rtr, u8 *buf)
00550 {
00551     u8 reg_inf;
00552
00553     if (CAN->SR & 0x01) // RFIFO 存有一帧或多帧数据, 可以通过 RFIFO 寄存器读取
00554     {
00555         CAN->MOD &= ~(1 << 0); // 正常输出模式
00556         reg_inf = CAN->TXRX0; // 读取的一帧或多帧数据
00557         if (reg_inf & 0x80) // EFF (扩展) 帧
00558         {
00559             *ide = 1;
00560             *id = (CAN->TXRX1 << 21) | (CAN->TXRX2 << 13) | (CAN->TXRX3 << 5) | (CAN->TXRX4 >> 3); //
00561             获取ID
00562             if (reg_inf & 0x40)
00563             {
00564                 *rtr = 1; // 遥控帧
00565             }
00566             else
00567             {
00568                 *rtr = 0; // 数据帧
00569             }
00570             buf[0] = CAN->TXRX5; // 获取数据
00571             buf[1] = CAN->TXRX6;
00572             buf[2] = CAN->TXRX7;
00573             buf[3] = CAN->TXRX8;
00574             buf[4] = CAN->TXRX9;
00575             buf[5] = CAN->TXRXA;
00576             buf[6] = CAN->TXRXB;
00577             buf[7] = CAN->TXRXC;
00578         }
00579         else // SFF (标准) 帧
00580         {
00581             *ide = 0;
00582             if (reg_inf & 0x40)
00583             {
00584                 *rtr = 1;
00585             }
00586             else
00587             {
00588                 *rtr = 0;
00589             }
00590             *id = ((CAN->TXRX1 << 3) | (CAN->TXRX2 >> 5));
00591             buf[0] = CAN->TXRX3;
00592             buf[1] = CAN->TXRX4;
00593             buf[2] = CAN->TXRX5;
00594             buf[3] = CAN->TXRX6;
00595             buf[4] = CAN->TXRX7;
00596             buf[5] = CAN->TXRX8;

```

```

00597         buf[6] = CAN->TXRX9;
00598         buf[7] = CAN->TXRXA;
00599     }
00600
00601     CAN->CMR |= 0x04; // 释放接收缓冲区
00602
00603     return 0; // 接收完成
00604 }
00605 else // fifo is empty
00606 {
00607     return 0xff; // 接收异常
00608 }
00609 }
00610
00669 uint8_t CAN_ReadState(CAN_TypeDef *CANx, uint32_t sr)
00670 {
00671     return (uint8_t)(CANx->SR & sr);
00672 }
00673
00695 enumLossposition CAN_ReadALC(void)
00696 {
00697     return (enumLossposition) CAN_ALC;
00698 }
00699
00720 uint8_t CAN_ReadRXERRCNT(void)
00721 {
00722     return CAN_RXERR;
00723 }
00724
00745 uint8_t CAN_ReadTXERRCNT(void)
00746 {
00747     return CAN_TXERR;
00748 }
00749
00772 void CAN_ReceFramInfor(uint8_t *cnt, uint8_t *rbsa)
00773 {
00774     *cnt = CAN_RMC;
00775     *rbsa = CAN_RBSA;
00776 }
00777
00801 void CAN_ReadErrorCode(enumErrortype *errtype, enumErrortime *errtim, enumErrposition *errpos)
00802 {
00803     *errtype = (enumErrortype)((CAN_ECC & 0xC0) >> 6);
00804     *errtim = (enumErrortime)((CAN_ECC & 0x20) >> 5);
00805     *errpos = (enumErrposition)(CAN_ECC & 0x1f);
00806 }
00807
00828 uint32_t CAN_GetIRQFlag(void)
00829 {
00830     return CAN_INTR;
00831 }

```

3.47 source/lks32mc08x_cmp.c 文件参考

文件名称: [lks32mc08x_cmp.c](#)

文件标识: 无

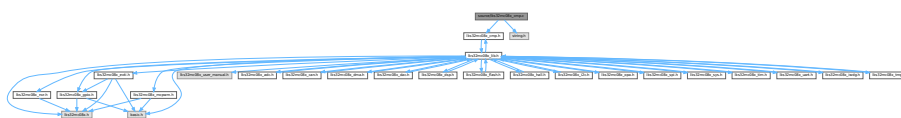
内容摘要: cmp外设驱动程序

其它说明: 无

```
#include "lks32mc08x_cmp.h"
```

```
#include <string.h>
```

lks32mc08x_cmp.c 的引用(Include)关系图:



函数

- void **CMP_Init** (CMP_InitTypeDef *this)
函数名称: *CMP_Init(CMP_InitTypeDef* this)*
- void **CMP_StructInit** (CMP_InitTypeDef *CMP_InitStruct)
函数名称: *void CMP_StructInit(CMP_InitTypeDef* CMP_InitStruct)*
- u8 **CMP_GetIRQFlag** (u8 CMPx)
函数名称: *u8 CMP_GetIRQFlag(u8 CMPx)*
- void **CMP_ClearIRQFlag** (u8 CMPx)
函数名称: *void CMP_ClearIRQFlag(u8 CMPx)*

3.47.1 详细描述

文件名称: [lks32mc08x_cmp.c](#)
 文件标识: 无
 内容摘要: cmp外设驱动程序
 其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2019年3月5日	1.0	Howlet	创建

修改日志:

在文件 [lks32mc08x_cmp.c](#) 中定义.

3.47.2 函数说明

3.47.2.1 CMP_ClearIRQFlag()

```
void CMP_ClearIRQFlag (
    u8 CMPx )
```

函数名称: void **CMP_ClearIRQFlag**(u8 CMPx)

功能描述: 清除TIM中断标志

参见

被调用函数: 无

参数

输入参数:	CMP0, CMP1即获取CMP0和CMP1中断标志位
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
if (CMP_GetIRQFlag (CMP1)) //获取CMP1中断标志位
{
    CMP_ClearIRQFlag (CMP1); //清除CMP1中断标志位
}
```

Date	Version	Author	Description
2022年4月11日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_cmp.c](#) 第 197 行定义.

3.47.2.2 CMP_GetIRQFlag()

```
u8 CMP_GetIRQFlag (
    u8 CMPx )
```

函数名称: [u8 CMP_GetIRQFlag\(u8 CMPx\)](#)

功能描述: 读取比较器中断

参见

被调用函数: 无

参数

输入参数:	CMP0, CMP1即获取CMP0和CMP1中断标志位
输出参数:	无

返回

返回 值: TIM中断标志0/1

注解

其它说明: 无

警告

无

示例代码:

```
if (CMP_GetIRQFlag (CMP1)) //获取CMP1中断标志位
{
    CMP_ClearIRQFlag (CMP1); //清除CMP1中断标志位
}
```

Date	Version	Author	Description
2022年4月21日	1.0	Yangzj	创建

修改日志:

在文件 [lks32mc08x_cmp.c](#) 第 148 行定义.

3.47.2.3 CMP_Init()

```
void CMP_Init (
    CMP_InitTypeDef * this )
```

函数名称: [CMP_Init\(CMP_InitTypeDef* this\)](#)

功能描述: CMP初始化

参见

被调用函数: [SYS_AnalogModuleClockCmd\(\)](#)

参数

输入参数:	CMP.InitTypeDef
输出参数:	无

返回

返回 值: 无

注解

其它说明: 无

警告

无

示例代码:

```
CMP_InitTypeDef CMP_InitStructure;  
CMP_StructInit (&CMP_InitStructure);  
CMP_InitStructure.CLK10_EN      = ENABLE      ; // 比较器 1/0 滤波时钟使能  
CMP_InitStructure.FIL_CLK10_DIV16 = 15         ; // 比较器 1/0 滤波系数      0~0xF  
CMP_InitStructure.FIL_CLK10_DIV2  = 0          ; // 比较器 1/0 滤波移位系数  0~0x3    最终分频系数为  
                                   (DIV16+1) << DIV2  
CMP_InitStructure.CMP_FT         = DISABLE     ; // 比较器快速比较使能  
CMP_InitStructure.CMP_HYS        = 0           ; // 比较器回差选择      0: 回差20mV 1: 回差0mV  
  
CMP_InitStructure.CMP1_EN        = ENABLE      ; // 比较器 1 使能  
CMP_InitStructure.CMP1_IN_EN     = ENABLE      ; // 比较器 1 信号输入使能 影响比较器数字模块信号, 模拟部  
分不受影响  
CMP_InitStructure.CMP1_SELNP     = CMP1_SELNP_IP2 ; // 比较器 1 信号正端选择 见 CMP1_SELNP_CMP1_IP0 比较器  
定义  
CMP_InitStructure.CMP1_SELNM     = CMP1_SELNM_DAC ; // 比较器 1 信号负端选择  
CMP_InitStructure.CMP1_IE        = ENABLE      ; // 比较器 1 中断使能  
CMP_InitStructure.CMP1_POL       = 0           ; // 比较器 1 极性选择      0: 高电平有效 1: 低电平有效  
只影响比较器中断  
CMP_InitStructure.CMP1_RE        = DISABLE     ; // 比较器 1 DMA 请求使能  
CMP_InitStructure.CMP1_IRQ_TRIG  = DISABLE     ; // 比较器 1 边沿触发使能  
CMP_InitStructure.CMP1_CHN3P_WIN_EN = DISABLE  ; // MCPWM 模块 CHN3_P 通道使能比较器 1 开窗  
CMP_InitStructure.CMP1_CHN2P_WIN_EN = DISABLE  ; // MCPWM 模块 CHN2_P 通道使能比较器 1 开窗  
CMP_InitStructure.CMP1_CHN1P_WIN_EN = DISABLE  ; // MCPWM 模块 CHN1_P 通道使能比较器 1 开窗  
CMP_InitStructure.CMP1_CHN0P_WIN_EN = DISABLE  ; // MCPWM 模块 CHN0_P 通道使能比较器 1 开窗  
CMP_InitStructure.CMP1_W_PWM_POL = DISABLE     ; // 比较器 1 开窗 PWM 信号极性选择 0: 高电平有效; 1: 低  
电平有效  
CMP_Init (&CMP_InitStructure);
```

Date	Version	Author	Description
2018年11月26日	1.0	Howlet Li	创建

修改日志:

在文件 lks32mc08x_cmp.c 第 56 行定义.

函数调用图:



3.47.2.4 CMP_StructInit()

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

函数名称: void CMP_StructInit(CMP_InitTypeDef* CMP_InitStruct)

功能描述: ADC结构体初始化

参见

被调用函数: 无

参数

输入参数:	CMP_InitTypeDef
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
CMP_StructInit CMP_InitStructure;  
CMP_StructInit (&CMP_InitStructure); //初始化结构体
```

Date	Version	Author	Description
2018年11月26日	1.0	Howlet Li	创建

修改日志:

在文件 `lks32mc08x_cmp.c` 第 121 行定义.

3.48 lks32mc08x_cmp.c

[浏览该文件的文档.](#)

```

00001
00014 #include "lks32mc08x_cmp.h"
00015 #include <string.h>
00056 void CMP_Init(CMP_InitTypeDef *this)
00057 {
00058     // Comparator初始化
00059     SYS_AnalogModuleClockCmd(SYS_AnalogModule_CMP1, this->CMP1.EN);
00060     SYS_AnalogModuleClockCmd(SYS_AnalogModule_CMP0, this->CMP0.EN);
00061     SYS_WR_PROTECT = 0x7A83;
00062
00063     SYS_AFE_REG3 |= (this->CMP0_SELN << 2) | (this->CMP1_SELN << 4) | (this->CMP0_SELP << 8) |
00064                    (this->CMP1_SELP << 12) | (this->CMP_HYS << 7);
00065
00066     /* CMP_HYS 比较器滞回功能, 默认20mV, 写0关闭 */
00067
00068     SYS_AFE_REG1 |= this->CMP_FT; /* 快速比较, Enable后 30nS */
00069
00070     SYS_AFE_REG5 |= (this->CMP0_EN << 6) | (this->CMP1_EN << 7);
00071
00072     SYS_WR_PROTECT = 0;
00073
00074     CMP->IE = (this->CMP1_IE << 1) | (this->CMP0_IE);
00075     if (this->CLK10_EN == DISABLE)
00076     {
00077         if (this->CMP1_IN_EN || this->CMP0_IN_EN)
00078         {
00079             CMP->TCLK = BIT3;
00080         }
00081         else
00082         {
00083             CMP->TCLK = 0;
00084         }
00085     }
00086     else
00087     {
00088         CMP->TCLK = (this->FIL_CLK10_DIV16 << 4) | (this->CLK10_EN << 3) |
00089                    (this->FIL_CLK10_DIV2);
00090     }
00091     CMP->CFG = (this->CMP1_W_PWM_POL << 7) | (this->CMP1_IRQ_TRIG << 6) |
00092                (this->CMP1_IN_EN << 5) | (this->CMP1_POL << 4) |
00093                (this->CMP0_W_PWM_POL << 3) | (this->CMP0_IRQ_TRIG << 2) |
00094                (this->CMP0_IN_EN << 1) | (this->CMP0_POL);
00095     CMP->BLCWIN = (this->CMP1_CHN3P_WIN_EN << 7) | (this->CMP1_CHN2P_WIN_EN << 6) |
00096                  (this->CMP1_CHN1P_WIN_EN << 5) | (this->CMP1_CHN0P_WIN_EN << 4) |
00097                  (this->CMP0_CHN3P_WIN_EN << 3) | (this->CMP0_CHN2P_WIN_EN << 2) |
00098                  (this->CMP0_CHN1P_WIN_EN << 1) | (this->CMP0_CHN0P_WIN_EN);
00099     CMP->IF = 3;
00100 }
00121 void CMP_StructInit(CMP_InitTypeDef *CMP_InitStruct)
00122 {
00123     memset(CMP_InitStruct, 0, sizeof(CMP_InitTypeDef));
00124 }
00125
00148 u8 CMP_GetIRQFlag(u8 CMPx)
00149 {
00150     if (CMPx == CMP0)
00151     {
00152         if (CMP_IF & BIT0)
00153         {
00154             return 1;
00155         }
00156         else
00157         {
00158             return 0;

```

```

00159     }
00160   }
00161   else if (CMPx == CMP1)
00162   {
00163       if (CMP_IF & BIT1)
00164       {
00165           return 1;
00166       }
00167       else
00168       {
00169           return 0;
00170       }
00171   }
00172   return 0;
00173 }
00174
00197 void CMP_ClearIRQFlag(u8 CMPx)
00198 {
00199     if (CMPx == CMP0)
00200     {
00201         CMP_IF = BIT0;
00202     }
00203     else if (CMPx == CMP1)
00204     {
00205         CMP_IF = BIT1;
00206     }
00207 }

```

3.49 source/lks32mc08x_dac.c 文件参考

文件名称: LKS32MC08x_dac.c

文件标识: 无

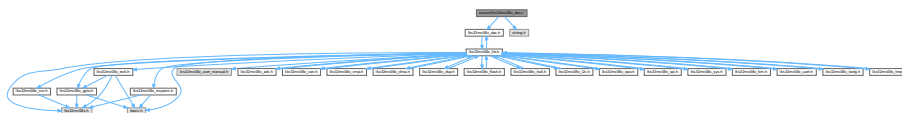
内容摘要: DAC外设驱动程序

其它说明: 无

```
#include "lks32mc08x_dac.h"
```

```
#include "string.h"
```

lks32mc08x_dac.c 的引用(Include)关系图:



函数

- void **DAC_StructInit** (DAC_InitTypeDef *DAC_InitStruct)
函数名称: void **DAC_StructInit**(DAC_InitTypeDef* DAC_InitStruct)
- void **DAC_Init** (DAC_InitTypeDef *DAC_InitStruct)
函数名称: void **DAC_init**(DAC_InitTypeDef* DAC_InitStruct)
- void **DAC_OutputValue** (uint32_t DACValue)
函数名称: void **DAC_OutputValue**(uint32_t DACValue)
- void **DAC_OutputVoltage** (uint16_t DACVoltage)
函数名称: void **DAC_OutputVoltage**(uint32_t DACVoltage)
- void **DAC_Output** (DAC_InitTypeDef *DAC_InitStruct)

3.49.1 详细描述

文件名称: LKS32MC08x_dac.c
文件标识: 无
内容摘要: DAC外设驱动程序
其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2022年4月18日	1.0	Zhu Jie	创建

修改日志:

在文件 [lks32mc08x_dac.c](#) 中定义.

3.49.2 函数说明

3.49.2.1 DAC_Init()

```
void DAC_Init (
    DAC_InitTypeDef * DAC_InitStruct )
```

函数名称: void DAC_init(DAC_InitTypeDef* DAC_InitStruct)

功能描述: DAC初始化函数

参见

被调用函数: [SYS_AnalogModuleClockCmd\(\)](#)

参数

输入参数:	DAC_InitTypeDef
输出参数:	无

返回

返回值: 无

注解

其它说明： 无

警告

无

示例代码:

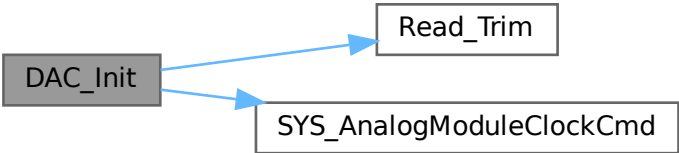
```
DAC_InitTypeDef DAC_InitStructure;  
DAC_StructInit (&DAC_InitStructure);  
DAC_InitStructure.DAC_GAIN = DAC_RANGE_1V2; //DAC量程为1.2V  
DAC_InitStructure.DACOUT_EN = DISABLE ; //DAC通过IO P0[0]输出  
DAC_Init (&DAC_InitStructure); // DAC初始化
```

Date	Version	Author	Description
2022年4月18日	1.0	Zhu Jie	创建

修改日志:

在文件 [lks32mc08x-dac.c](#) 第 64 行定义.

函数调用图:



3.49.2.2 DAC_Output()

```
void DAC_Output (  
DAC_InitTypeDef * DAC_InitStruct )
```

在文件 [lks32mc08x-dac.c](#) 第 165 行定义.

3.49.2.3 DAC_OutputValue()

```
void DAC_OutputValue (
    uint32_t DACValue )
```

函数名称: void [DAC_OutputValue\(uint32_t DACValue\)](#)

功能描述: DAC输出数字量数值设置

参见

被调用函数: 无

参数

输入参数:	DACValue: DAC输出电压对应数字量
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
DAC\_OutputValue (2048); //DAC输出 $2048 \times 1.2 / 4096 = 0.6V$ 
```

Date	Version	Author	Description
2022年4月18日	1.0	Zhu Jie	创建

修改日志:

在文件 [lks32mc08x-dac.c](#) 第 111 行定义.

3.49.2.4 DAC_OutputVoltage()

```
void DAC_OutputVoltage (
    uint16_t DACVoltage )
```

函数名称: void DAC_OutputVoltage(uint32_t DACVoltage)

功能描述: DAC输出模拟量数值设置

参见

被调用函数: 无

参数

输入参数:	DACVoltage: DAC输出电压对应数字量
输出参数:	无

返回

返回值: 无

注解

其它说明: 输入变量DACVoltage为Q12格式, 即输出1V对应DACVoltage = $1 \ll 12 = 4096$

警告

无

示例代码:

```
DAC_OutputVoltage(BIT12 * 0.6); //DAC输出0.6V
```

Date	Version	Author	Description
2022年4月18日	1.0	Zhu Jie	创建

修改日志:

在文件 [lks32mc08x-dac.c](#) 第 139 行定义.

3.49.2.5 DAC_StructInit()

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

函数名称: void [DAC_StructInit\(DAC_InitTypeDef* DAC_InitStruct\)](#)

功能描述: DAC结构体初始化

参见

被调用函数: 无

参数

输入参数:	ADC_InitTypeDef
输出参数:	无

返回

返回 值: 无

注解

其它说明: 无

警告

无

示例代码:

```
DAC_StructInit DAC_InitStructure;  
DAC_StructInit (&DAC_InitStructure); //初始化结构体
```

Date	Version	Author	Description
2022年4月18日	1.0	Zhu Jie	创建

修改日志:

在文件 [lks32mc08x_dac.c](#) 第 37 行定义.

3.50 lks32mc08x_dac.c

浏览该文件的文档.

```
00001  
00014 #include "lks32mc08x_dac.h"  
00015 #include "string.h"  
00016  
00037 void DAC_StructInit (DAC_InitTypeDef *DAC_InitStructure)  
00038 {  
00039     memset(DAC_InitStructure, 0, sizeof(DAC_InitTypeDef));  
00040 }  
00064 void DAC_Init (DAC_InitTypeDef *DAC_InitStructure)  
00065 {  
00066     SYS_AnalogModuleClockCmd(SYS_AnalogModule_DAC, ENABLE); /* DAC 时钟使能 */  
00067  
00068     SYS_WR_PROTECT = 0x7a83; /* 解锁寄存器写保护 */  
00069  
00070     SYS_AFE_REG1 &= ~( (u32)0x03 << 6); /* DAC增益寄存器清零 */  
00071     SYS_AFE_REG1 |= DAC_InitStructure->DAC_GAIN << 6; /* 设置DAC满量程为1.2V */  
00072  
00073     if ( ((SYS_AFE_REG1 >> 6) & 0x03) == DAC_RANGE_1V2)  
00074     { /* 加载DAC 1.2V量程校正值 */  
00075         SYS_AFE_DAC_DC = ReadTrim(0x00000344);
```

```

00076     SYS_AFE_DAC_AMC = Read.Trim(0x00000334);
00077 }
00078 else if (((SYS_AFE_REG1 >> 6) & 0x03) == DAC_RANGE_3V0)
00079 { /* 加载DAC 3.0V量程校正值 */
00080     SYS_AFE_DAC_DC = Read.Trim(0x00000340);
00081     SYS_AFE_DAC_AMC = Read.Trim(0x00000330);
00082 }
00083 else if (((SYS_AFE_REG1 >> 6) & 0x03) == DAC_RANGE_4V85)
00084 { /* 加载DAC 4.85V量程校正值 */
00085     SYS_AFE_DAC_DC = Read.Trim(0x00000348);
00086     SYS_AFE_DAC_AMC = Read.Trim(0x00000338);
00087 }
00088
00089 SYS_WR_PROTECT = 0xffff; /* 锁定寄存器写保护 */
00090 }
00091
00111 void DAC.OutputValue(uint32_t DACValue)
00112 {
00113     if (DACValue >= 4095)
00114     {
00115         DACValue = 4095;
00116     }
00117     SYS_AFE_DAC = DACValue;
00118 }
00119
00139 void DAC.OutputVoltage(uint16_t DACVoltage)
00140 {
00141     u32 temp = 0;
00142     u32 range = 0;
00143
00144     if (((SYS_AFE_REG1 >> 6) & 0x03) == DAC_RANGE_1V2)
00145     { /* 加载DAC 1.2V量程校正值 */
00146         range = (uint16_t)((1.0 / 1.2) * BIT12);
00147     }
00148     else if (((SYS_AFE_REG1 >> 6) & 0x03) == DAC_RANGE_3V0)
00149     { /* 加载DAC 3.0V量程校正值 */
00150         range = (uint16_t)((1.0 / 3.0) * BIT12);
00151     }
00152     else if (((SYS_AFE_REG1 >> 6) & 0x03) == DAC_RANGE_4V85)
00153     { /* 加载DAC 4.85V量程校正值 */
00154         range = (uint16_t)((1.0 / 4.85) * BIT12);
00155     }
00156     temp = (DACVoltage * range + BIT11) >> 12;
00157
00158     if (temp >= 4095)
00159     {
00160         temp = 4095;
00161     }
00162     SYS_AFE_DAC = temp;
00163 }
00164
00165 void DAC.Output(DAC_InitTypeDef *DAC_InitStruct)
00166 {
00167     SYS_WR_PROTECT = 0x7a83; /* 解锁寄存器写保护 */
00168
00169     SYS_AFE_REG3 |= DAC_InitStruct->DACOUT_EN << 11; /* DAC输出配置 */
00170
00171     SYS_WR_PROTECT = 0xffff; /* 锁定寄存器写保护 */
00172 }
00173 /***** (C) COPYRIGHT LINKO SEMICONDUCTOR *****/

```

3.51 source/lks32mc08x_dma.c 文件参考

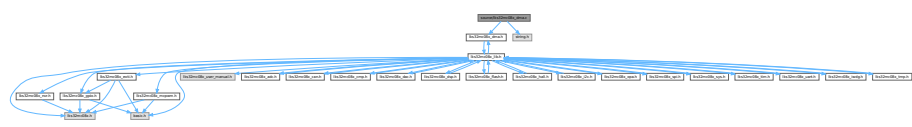
文件名称: LKS32MC08x_dma.c
 文件标识: 无
 内容摘要: DMA外设驱动程序
 其它说明: 无

```

#include "lks32mc08x_dma.h"
#include "string.h"

```

lks32mc08x_dma.c 的引用(Include)关系图:



函数

- void DMA_StructInit (DMA_InitTypeDef *DMAInitStruct)
函数名称: void DMA_StructInit(DMA_InitTypeDef *DMAInitStruct)
- void DMA_Init (DMA_RegTypeDef *DMAx, DMA_InitTypeDef *DMAInitStruct)
函数名称: void DMA_Init(DMA_RegTypeDef *DMAx, DMA_InitTypeDef *DMAInitStruct)
- void DMA_CHx_EN (DMA_RegTypeDef *DMAx, u32 Channel_EN)
函数名称: void DMA_CHx_EN(DMA_RegTypeDef *DMAx,u32 Channel_EN)
- uint32_t DMA_GetIRQFlag (DMA_RegTypeDef *DMAx, u32 timer_if)
函数名称: uint32_t DMA_GetIRQFlag(DMA_RegTypeDef *DMAx , u32 timer_if)
- void DMA_ClearIRQFlag (DMA_RegTypeDef *DMAx, uint32_t tempFlag)
函数名称: void DMA_ClearIRQFlag(DMA_RegTypeDef *DMAx , uint32_t tempFlag)

3.51.1 详细描述

文件名称: LKS32MC08x_dma.c
文件标识: 无
内容摘要: DMA外设驱动程序
其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2020年8月1日	1.0	Howlet	创建

修改日志:

Date	Version	Author	Description
2022年4月13日	1.1	HuangMG	修改

修改日志:增加DMA通道使能函数,读取和清除DAM中断标志位函数

在文件 lks32mc08x_dma.c 中定义.

3.51.2 函数说明

3.51.2.1 DMA_CHx.EN()

```
void DMA_CHx_EN (
    DMA_RegTypeDef * DMAx,
    u32 Channel_EN )
```

函数名称: void DMA_CHx_EN(DMA_RegTypeDef *DMAx,u32 Channel_EN)

功能描述: 使能DMA通道

参见

被引用内容: DMAx可选: DMA_CH0 , DMA_CH1 , DMA_CH2 , DMA_CH3

参数

输入参数:	DMAx: DMA通道选择 Channel_EN: ENABLE: 使能DMA通道, DISABLE: 关闭DMA通道使能
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
DMA_CHx_EN (DMA_CH0, ENABLE) ; // 使能DMA通道0
```

Date	Version	Author	Description
2022年4月13日	1.1	HuangMG	修改

修改日志:

在文件 lks32mc08x_dma.c 第 140 行定义.

3.51.2.2 DMA_ClearIRQFlag()

```
void DMA_ClearIRQFlag (
    DMA_RegTypeDef * DMAx,
    uint32_t tempFlag )
```

函数名称: void DMA_ClearIRQFlag(DMA_RegTypeDef *DMAx , uint32_t tempFlag)

功能描述: 清除DMA中断标志

参见

被引用内容: DMAx可选: DMA_CH0 , DMA_CH1 , DMA_CH2 , DMA_CH3

参数

输入参数:	DMAx: DMA通道选择	
	timer_if参数可选:	
	宏定义	说明
	DMA_CH0_FIF	通道 0 完成中断标志
	DMA_CH1_FIF	通道 1 完成中断标志
	DMA_CH2_FIF	通道 2 完成中断标志
	DMA_CH3_FIF	通道 3 完成中断标志
	DMA_CH0{EIF	通道 0 错误中断标志
	DMA_CH1{EIF	通道 1 错误中断标志
输出参数:	无	

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
if (DMA_GetIRQFlag (DMA_CH0, DMA_CH0_FIF) ) //获取DMA通道0完成中断标志
{
    DMA_ClearIRQFlag (DMA_CH0, DMA_CH0_FIF) //清除DMA通道0完成中断标志
}
```

Date	Version	Author	Description
2022年4月13日	1.1	HuangMG	修改

修改日志:

在文件 [lks32mc08x_dma.c](#) 第 241 行定义.

3.51.2.3 DMA_GetIRQFlag()

```
uint32_t DMA_GetIRQFlag (
    DMA_RegTypeDef * DMAx,
    u32 timer_if )
```

函数名称: `uint32_t DMA_GetIRQFlag(DMA_RegTypeDef *DMAx , u32 timer_if)`

功能描述: 获取DMA中断标志

参见

被引用内容: DMAx可选: DMA_CH0 , DMA_CH1 , DMA_CH2 , DMA_CH3

参数

输入参数:	DMAx: DMA通道选择	
	timer_if参数可选:	
	宏定义	说明
	DMA_CH0_FIF	通道 0 完成中断标志
	DMA_CH1_FIF	通道 1 完成中断标志
	DMA_CH2_FIF	通道 2 完成中断标志
	DMA_CH3_FIF	通道 3 完成中断标志
	DMA_CH0{EIF	通道 0 错误中断标志
	DMA_CH1{EIF	通道 1 错误中断标志
	DMA_CH2{EIF	通道 2 错误中断标志
	DMA_CH3{EIF	通道 3 错误中断标志

参见

参数

输出参数:	无

返回

返回 值： 无

注解

其它说明： 无

警告

只有对应中断使能后，改为才能读取，如果对应中断未使能，读取结果一直为0

示例代码:

```
if (DMA_GetIRQFlag (DMA_CH0, DMA_CH0_FIF) ) //获取DMA通道0完成中断标志
{
}
```

Date	Version	Author	Description
2022年4月13日	1.1	HuangMG	修改

修改日志:

在文件 [lks32mc08x_dma.c](#) 第 185 行定义.

3.51.2.4 DMA_Init()

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

函数名称: void [DMA_Init\(DMA_RegTypeDef *DMAx, DMA_InitTypeDef *DMAInitStruct\)](#)

功能描述: DMA初始化函数

参见

被引用内容： 无

参数

输入参数:	ADCx: ADC0 , ADC_InitTypeDef
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
u8 memaddr[25] = {0x11, 0x12, 0x13}; // 发送数据缓冲区

DMA_InitTypeDef DMA_InitStructure;
DMA_StructInit(&DMA_InitStructure);
DMA_InitStructure.DMA_IRQ_EN = DMA_TCIE; // DMA 传输完成中断使能
DMA_InitStructure.DMA_DIR = MEMORY2PERI; // 内存至外设
DMA_InitStructure.DMA_CIRC = DISABLE; // 关闭循环传输模式
DMA_InitStructure.DMA_PINC = ENABLE; // 外设地址每轮内是否递增, 高有效
DMA_InitStructure.DMA_MINC = ENABLE; // 内存地址第二轮是否在第一轮地址的基础上递增 (轮内地址一定递增), 高有效
DMA_InitStructure.DMA_PBTW = DMA_BYTE_TRANS; // 外设访问位宽. 0:byte, 1:half-word, 2:word
DMA_InitStructure.DMA_MBTW = DMA_BYTE_TRANS; // 内存访问位宽. 0:byte, 1:half-word, 2:word
DMA_InitStructure.DMA_REQ_EN = DMA_CH3_CAN_REQ_EN; // CAN DMA请求使能, 高有效
DMA_InitStructure.DMA_TIMES = 5; // DMA 通道 x 每轮数据搬运次数 1~511
DMA_InitStructure.DMA_ROUND = 5; // DMA 通道 x 采样轮数 1~255
DMA_InitStructure.DMA_CPAR = (u32)&CAN_TXRX0; // DMA 通道 x 外设地址
DMA_InitStructure.DMA_CMAR = (u32)memaddr; // DMA 通道 x 内存地址
DMA_Init(DMA_CH3, &DMA_InitStructure);
DMA_CHx_EN(DMA_CH3, ENABLE); // 使能DMA_CH3通道
```

Date	Version	Author	Description
2015年11月26日	1.0	Howlet Li	创建

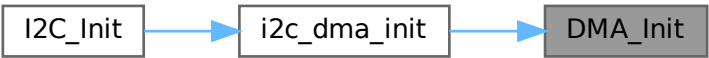
修改日志:

Date	Version	Author	Description
2022年4月13日	1.1	HuangMG	修改

修改日志:去除初始化使能DMA通道, 改为DMA_CHx_EN () 函数使能DMA通道

在文件 lks32mc08x_dma.c 第 101 行定义.

这是这个函数的调用关系图:



3.51.2.5 DMA_StructInit()

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

函数名称: void DMA_StructInit(DMA_InitTypeDef *DMAInitStruct)

功能描述: DMA结构体初始化

参见
被引用内容: 无

参数

输入参数:	DMA_InitTypeDef
输出参数:	无

返回
返回 值: 无

注解
其它说明: 无

警告
无

示例代码:

```
DMA_InitTypeDef DMA_InitStructure;
DMA_StructInit (&DMA_InitStructure); //初始化结构体
```

Date	Version	Author	Description
2015年11月26日	1.0	Howlet	修改

修改日志:

在文件 `lks32mc08x_dma.c` 第 55 行定义.

3.52 lks32mc08x_dma.c

[浏览该文件的文档.](#)

```

00001
00020 #include "lks32mc08x_dma.h"
00021 #include "string.h"
00022
00023 /*****
00024 函数名称: void DMA_StructInit(DMA_InitTypeDef *DMAInitStruct)
00025 功能描述: DMA结构体初始化
00026 操作的表: 无
00027 输入参数: 无
00028 输出参数: 无
00029 返回值: 无
00030 其它说明:
00031 修改日期 版本号 修改人 修改内容
00032 -----
00033 2020.08.1 V1.0 howlet 创建
00034 *****/
00055 void DMA_StructInit(DMA_InitTypeDef *DMAInitStruct)
00056 {
00057     memset(DMAInitStruct, 0, sizeof(DMA_InitTypeDef));
00058 }
00059
00101 void DMA_Init(DMA_RegTypeDef *DMAx, DMA_InitTypeDef *DMAInitStruct)
00102 {
00103     uint32_t ctms;
00104     /* 通道配置寄存器 赋值 */
00105     DMAx->DMA_CCR = 0;
00106     ctms = DMAInitStruct->DMA_TIMES | (DMAInitStruct->DMA_ROUND << 16);
00107     do {
00108         DMAx->DMA_CTMS = ctms;
00109     } while (DMAx->DMA_CTMS != ctms);
00110
00111     DMAx->DMA_CPAR = DMAInitStruct->DMA_CPAR; /* 外设地址寄存器 */
00112     DMAx->DMA_CMAR = DMAInitStruct->DMA_CMAR; /* 内存地址寄存器 */
00113     DMAx->DMA_CCR = (DMAInitStruct->DMA_IRQ_EN << 1) | (DMAInitStruct->DMA_DIR << 4) |
00114         (DMAInitStruct->DMA_CIRC << 5) | (DMAInitStruct->DMA_PINC << 6) |
00115         (DMAInitStruct->DMA_MINC << 7) |
00116         (DMAInitStruct->DMA_PBTW << 8) | (DMAInitStruct->DMA_MBTW << 10) |
00117         (DMAInitStruct->DMA_REQ_EN << 12);
00118     DMA_CTRL = 0x0001; /*enable dma, mcu has higher priorit */
00119 }
00140 void DMA_CHx.EN(DMA_RegTypeDef *DMAx, u32 Channel.EN)
00141 {
00142     if (Channel.EN)
00143     {
00144         DMAx->DMA_CCR |= BIT0; /*使能DMA搬运*/
00145     }
00146     else
00147     {
00148         DMAx->DMA_CCR &= ~BIT0; /*关闭通道使能*/
00149     }
00150 }
00151
00185 uint32_t DMA_GetIRQFlag(DMA_RegTypeDef *DMAx, u32 timer.if)
00186 {
00187     u8 reg = 0;
00188     if ((DMAx->DMA_CCR) & BIT1) /*传输完成中断使能*/
00189     {
00190         reg = timer.if & 0x0f;
00191         if (DMA_IF & reg)
00192         {
00193             return 1;
00194         }
00195     }

```

```

00195     }
00196     if ((DMAx->DMA_CCR) & BIT2) /*错误中断使能*/
00197     {
00198         reg = timer_if & 0xf0;
00199         if (DMA_IF & reg)
00200         {
00201             return 1;
00202         }
00203     }
00204     return 0;
00205 }
00206
00241 void DMA_ClearIRQFlag(DMA_RegTypeDef *DMAx, uint32_t tempFlag)
00242 {
00243     (void)DMAx;
00244     DMA_IF = tempFlag;
00245 }

```

3.53 source/lks32mc08x_dsp.c 文件参考

文件名称: [lks32mc08x_dsp.c](#)

文件标识: 无

内容摘要: DSP驱动程序

其它说明: 无

#include "lks32mc08x_dsp.h"

lks32mc08x_dsp.c 的引用(Include)关系图:



函数

- void [DSP_Cmd](#) (FuncState state)
函数名称: [void DSP_Cmd\(FuncState state\)](#)
- void [DSP_CalcDivision](#) (stru_DiviComponents *stru_Divi)
函数名称: [void DSP_CalcDivision\(stru_DiviComponents * stru_Divi\)](#)
- void [DSP_CalcCordic](#) (stru_CordicComponents *Cordic, Arc.or.SinCFuction choise)
函数名称: [void DSP_CalcCordic\(stru_CordicComponents *Cordic, Arc.or.SinCFuction choise\)](#)
- s16 [DSP_CalcSprt](#) (s16 X, s16 Y)
函数名称: [s16 DSP_CalcSprt\(s16 X,s16 Y\)](#)
- uint32_t [DSP_GetSqrt](#) (uint32_t Data)
函数名称: [uint32_t DSP_GetSqrt\(uint32_t Data\)](#)
- void [DSP_Init](#) (void)
- s16 [DSP_arctan](#) (s16 x, s16 y)
- s32 [DSP_div](#) (s32 a, s32 b)
- s16 [DSP_sin](#) (u16 angle)
- s16 [DSP_cos](#) (u16 angle)
- u16 [DSP_rms](#) (s16 a, s16 b)
- s16 [DSP_mod](#) (s32 x, s16 y)
- u16 [DSP_sqrt](#) (u32 val)

3.53.1 详细描述

文件名称: lks32mc08x.dsp.c
文件标识: 无
内容摘要: DSP驱动程序
其它说明: 无

版权所有
(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2015年11月5日	1.0	Howlet	创建

修改日志:

Date	Version	Author	Description
2022年4月18日	2.0	Zhu Jie	修改

修改日志:DSP调用函数内增加中断打断重新复位DSP功能

在文件 lks32mc08x.dsp.c 中定义.

3.53.2 函数说明

3.53.2.1 DSP_arctan()

```
s16 DSP_arctan (
    s16 x,
    s16 y )
```

在文件 lks32mc08x.dsp.c 第 242 行定义.

3.53.2.2 DSP_CalcCordic()

```
void DSP_CalcCordic (
    stru_CordicComponents * Cordic,
    Arc_or_SinCFuction chose )
```

函数名称: void DSP_CalcCordic(stru_CordicComponents *Cordic, Arc_or_SinCFuction chose)

功能描述: DSP三角运算

参见

被引用内容: stru_CordicComponents , Arc_or_SinCFuction , SYS_SoftResetModule()

参数

输入参数:	Cordic: 角度或坐标X和Y, choose: 计算模式选择 0 Arctan;1 sin/cos
输出参数:	无

返回

返回 值: 无

注解

其它说明: 1、开方8个总线周期完成
2、Theta(角度)为16位有符号定点数, 表示范围 $-(32768) \sim 32767$) 对应 $-(\pi) \sim \pi$
3、SinValue值为16位有符号定点数, 其中 1bit 符号位, 15bit小数位; 表示范围 $-(1) \sim 1$)
4、CosValue值为16位有符号定点数, 其中 1bit 符号位, 15bit小数位; 表示范围 $-(1) \sim 1$)
5、ArctanX值为16位有符号定点数, Q15 格式, 其中 1bit 符号位, 15bit整数位, 表示范围 $-(32768) \sim 32767$)
6、ArctanY值为16位有符号定点数, Q15 格式, 其中 1bit 符号位, 15bit整数位, 表示范围 $-(32768) \sim 32767$)
7、ArctanValue值为16位有符号定点数, 表示范围 $-(32768) \sim 32767$)对应 $-(\pi) \sim \pi$)

警告

无

示例代码:

```
stru_CordicComponents CordicData; // 定义三角函数计算结构体变量
示例1:
CordicData.ArctanX = 100; //坐标x赋值 取值范围  $-(32768) \sim 32767$ )
CordicData.ArctanY = 10; //坐标y赋值 取值范围  $-(32768) \sim 32767$ )
    DSP_CalcDivision(&CordicData, ARCTAN);
arctan储存位置: CordicData.ArctanValue (输出范围 $-(32768) \sim 32767$ )对应 $-(\pi) \sim \pi$ )
示例2:
CordicData.Theta = 0; //角度赋值 取值范围  $-(32768) \sim 32767$ ) 对应 $-(\pi) \sim \pi$ )
    DSP_CalcDivision(&CordicData, SINCOS);
sin结果储存位置: CordicData.SinValue (输出范围 $-(32768) \sim 32767$ )对应 $-(1) \sim 1$ )
cos结果储存位置: CordicData.CosValue (输出范围 $-(32768) \sim 32767$ )对应 $-(1) \sim 1$ )
```

Date	Version	Author	Description
2016年5月25日	1.0	cfwu	创建

修改日志:

Date	Version	Author	Description
2022年4月18日	2.0	Zhu Jie	修改

修改日志:使用结构体变量，增加防止DSP计算被打断逻辑语句

在文件 [lks32mc08x_dsp.c](#) 第 136 行定义.

3.53.2.3 DSP_CalcDivision()

```
void DSP_CalcDivision (
    stru_DiviComponents * stru_Divi )
```

函数名称: void [DSP_CalcDivision\(stru_DiviComponents * stru_Divi\)](#)

功能描述: DSP使能和关闭

参见
被引用内容: [stru_DiviComponents](#) , [SYS_SoftResetModule\(\)](#)

参数

输入参数:	.nDID,被除数 .nDIS除数
输出参数:	无

返回

返回 值: 无

注解

- 其它说明:
- 1、除法10个总线周期完成
 - 2、被除数Dividend为32位大小，即范围是 $-(2^{31} - 1) \sim (2^{31} - 1)$
 - 3、除数Divisor为16位大小，即范围是 $-(2^{15} - 1) \sim (2^{15} - 1)$
 - 4、被除数不支持赋值为 -2^{31} ，除数不支持赋值为 -2^{15}
 - 5、商Quotient为32位大小
 - 6、余数Remainder为16位大小

警告

无

示例代码:

```
stru_DiviComponents DiviData; //定义除法函数计算结构体变量
DiviData.Dividend = 100; //被除数赋值
DiviData.Divisor = 10; //除数赋值
DSP_CalcDivision(&DiviData);
除法的商储存位置: DiviData.Quotient
除法的余数储存位置: DiviData.Remainder
```

Date	Version	Author	Description
2016年5月25日	1.0	cfwu	创建

修改日志:

Date	Version	Author	Description
2022年4月18日	2.0	Zhu Jie	修改

修改日志:使用结构体变量，增加防止DSP计算被打断逻辑语句

在文件 [lks32mc08x_dsp.c](#) 第 82 行定义.

3.53.2.4 DSP_CalcSprt()

```
s16 DSP_CalcSprt (
    s16 X,
    s16 Y )
```

函数名称: s16 [DSP_CalcSprt\(s16 X,s16 Y\)](#)

功能描述: DSP两个数平方后开方运算

参见

被引用内容: 无

参数

输入参数:	X和Y
输出参数:	无

返回

返回 值: $\sqrt{X^2+Y^2}$ 结果值

注解

其它说明: 1、开方8个总线周期完成 2、X值为16位有符号定点数，Q15 格式，其中 1bit 符号位，15bit整数位，表示范围(-(32768)~32767)
3、Y值为16位有符号定点数，Q15 格式，其中 1bit 符号位，15bit整数位，表示范围(-(32768)~32767)
4、输出结果 $\sqrt{X^2+Y^2}$: 为16位有符号定点数，其中1bit符号位，15bit整数位，表示范围(-(32768) ~ 32767)

警告
无

示例代码:

```
s16 DSP_Value = 0;  
DSP_Value = DSP_CalcSqrt(50,100); //sqrt(50^2+100^2) 50的平方加100的平方后开根号
```

Date	Version	Author	Description
2016年5月25日	1.0	cfwu	创建

修改日志:

Date	Version	Author	Description
2022年4月18日	2.0	Zhu Jie	修改

修改日志:使用结构体变量，增加防止DSP计算被打断逻辑语句

在文件 [lks32mc08x_dsp.c](#) 第 186 行定义.

3.53.2.5 DSP_Cmd()

```
void DSP_Cmd (  
    FuncState state )
```

函数名称: void [DSP_Cmd\(FuncState state\)](#)

功能描述: DSP使能和关闭

参见

被引用内容: [SYS_ModuleClockCmd\(\)](#)

参数

输入参数:	state: ENABLE: 使能， DISABLE:失能
输出参数:	无

返回

返回值: 无

注解

其它说明： 无

警告

无

示例代码:

```
DSP_Cmd (ENABLE); // 使能DSP时钟
```

Date	Version	Author	Description
2022年4月18日	1.0	Zhu Jie	创建

修改日志:

在文件 [lks32mc08x_dsp.c](#) 第 41 行定义.

函数调用图:



3.53.2.6 DSP_cos()

```
s16 DSP_cos (  
    u16 angle )
```

在文件 [lks32mc08x_dsp.c](#) 第 276 行定义.

3.53.2.7 DSP_div()

```
s32 DSP_div (  
    s32 a,  
    s32 b )
```

在文件 [lks32mc08x_dsp.c](#) 第 254 行定义.

3.53.2.8 DSP_GetSqrt()

```
uint32_t DSP_GetSqrt (
    uint32_t Data )
```

函数名称: `uint32_t DSP_GetSqrt(uint32_t Data)`

功能描述: DSP开平方函数

参见

被引用内容: 无

参数

输入参数:	被开方数
输出参数:	无

返回

返回值: 开方根

注解

其它说明: 1、开方8个总线周期完成
2、被开方数Data为32位无符号数, 即范围是(0 ~ (2³¹))
3、开方根sqrt(Data)为16位无符号数, 即范围是(0 ~ (2¹⁵))

警告

无

示例代码:

```
u16 GetSqrt_Value = 0;
GetSqrt_Value = DSP_GetSqrt(100); //sqrt(100) 100开根号结果为10
```

Date	Version	Author	Description
2016年5月25日	1.0	cfwu	创建

修改日志:

Date	Version	Author	Description
2022年4月18日	2.0	Zhu Jie	修改

修改日志:使用结构体变量, 增加防止DSP计算被打断逻辑语句

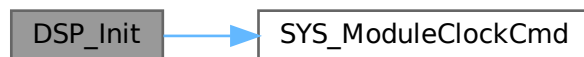
在文件 [lks32mc08x_dsp.c](#) 第 227 行定义.

3.53.2.9 DSP_Init()

```
void DSP_Init (
    void )
```

在文件 [lks32mc08x_dsp.c](#) 第 238 行定义.

函数调用图:



3.53.2.10 DSP_mod()

```
s16 DSP_mod (
    s32 x,
    s16 y )
```

在文件 [lks32mc08x_dsp.c](#) 第 298 行定义.

3.53.2.11 DSP_rms()

```
u16 DSP_rms (
    s16 a,
    s16 b )
```

在文件 [lks32mc08x_dsp.c](#) 第 287 行定义.

3.53.2.12 DSP_sin()

```
s16 DSP_sin (
    u16 angle )
```

在文件 [lks32mc08x_dsp.c](#) 第 265 行定义.

3.53.2.13 DSP_sqrt()

```
ul6 DSP_sqrt (
    u32 val )
```

在文件 [lks32mc08x_dsp.c](#) 第 310 行定义.

3.54 lks32mc08x_dsp.c

[浏览该文件的文档.](#)

```
00001
00020 #include "lks32mc08x_dsp.h"
00021
00041 void DSP_Cmd(FuncState state)
00042 {
00043     SYS_ModuleClockCmd(SYS_Module_DSP, state); /* DSP时钟使能 */
00044 }
00045
00082 void DSP_CalcDivision(stru_DiviComponents *stru_Divi)
00083 {
00084     int32_t flg = __get_PRIMASK();
00085     __disable_irq();
00086     DSP_DID = stru_Divi->Dividend; /* 被除数寄存器赋值 */
00087     DSP_DIS = stru_Divi->Divisor; /* 除数寄存器赋值 */
00088
00089     stru_Divi->Quotient = DSP_QUO; /* 取出商寄存器值 */
00090     stru_Divi->Remainder = DSP_REM; /* 取出余数寄存器值 */
00091     __set_PRIMASK(flgs);
00092 }
00093
00136 void DSP_CalcCordic(stru_CordicComponents *Cordic, Arc_or_SinCFuction choise)
00137 {
00138     int32_t flg = __get_PRIMASK();
00139     __disable_irq();
00140     if (choise) /* Sin Cos 值计算*/
00141     {
00142         DSP_SC = BIT2 | BIT1; /* 计算模式选择 Sin和Cos */
00143         DSP_CORDIC_THETA = Cordic->Theta; /* 角度寄存器赋值 */
00144         Cordic->SinValue = DSP_CORDIC_SIN; /* 取出sin寄存器值 */
00145         Cordic->CosValue = DSP_CORDIC_COS; /* 取出cos寄存器值 */
00146     }
00147     else /* Arctan 值计算*/
00148     {
00149         DSP_SC = BIT1; /* 计算模式选择 Arctan */
00150         DSP_CORDIC_X = Cordic->ArctanX; /* 坐标x寄存器赋值 */
00151         DSP_CORDIC_Y = Cordic->ArctanY; /* 坐标y寄存器赋值 */
00152         Cordic->ArctanValue = DSP_CORDIC_ARCTAN; /* 取出arctan寄存器值 */
00153     }
00154     __set_PRIMASK(flgs);
00155 }
00156
00186 s16 DSP_CalcSprt(s16 X, s16 Y)
00187 {
00188     s16 dsp_mod;
00189     int32_t flg = __get_PRIMASK();
00190     __disable_irq();
00191     DSP_SC = BIT1;
00192     DSP_CORDIC_X = X;
00193     DSP_CORDIC_Y = Y;
00194     dsp_mod = DSP_CORDIC_MOD;
00195     __set_PRIMASK(flgs);
00196     return dsp_mod;
00197 }
00198
00227 uint32_t DSP_GetSqrt(uint32_t Data)
00228 {
00229     uint32_t tdsp_sqrt;
00230     int32_t flg = __get_PRIMASK();
00231     __disable_irq();
00232     DSP_RAD = Data; /*被开方寄存器赋值*/
00233     tdsp_sqrt = DSP_SQRT;
00234     __set_PRIMASK(flgs);
00235     return tdsp_sqrt;
00236 }
00237
00238 void DSP_Init(void)
00239 {
00240     SYS_ModuleClockCmd(SYS_Module_DSP, ENABLE); /* DSP时钟使能*/
```

```

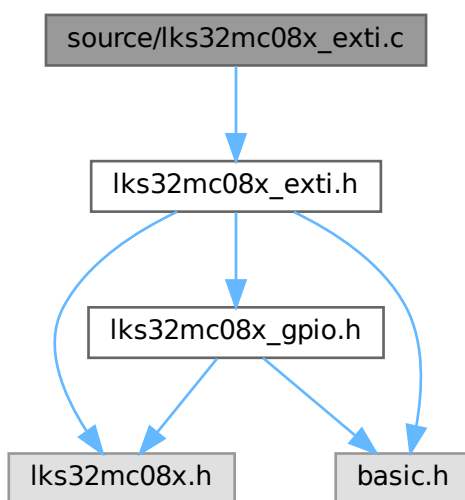
00241 }
00242 s16 DSP_arctan(s16 x, s16 y)
00243 {
00244     s16 answer;
00245     int32_t flg = __get_PRIMASK();
00246     __disable_irq();
00247     DSP_SC = BIT1; /* 计算模式选择 Arctan */
00248     DSP_CORDIC_X = x; /* 坐标x寄存器赋值 */
00249     DSP_CORDIC_Y = y; /* 坐标y寄存器赋值 */
00250     answer = DSP_CORDIC_ARCTAN; /* 取出arctan寄存器值 */
00251     __set_PRIMASK(flg);
00252     return answer;
00253 }
00254 s32 DSP_div(s32 a, s32 b)
00255 {
00256     s32 answer;
00257     int32_t flg = __get_PRIMASK();
00258     __disable_irq();
00259     DSP_DID = a;
00260     DSP_DIS = b;
00261     answer = DSP_QUO;
00262     __set_PRIMASK(flg);
00263     return answer;
00264 }
00265 s16 DSP_sin(u16 angle)
00266 {
00267     s16 answer;
00268     int32_t flg = __get_PRIMASK();
00269     __disable_irq();
00270     DSP_SC = BIT2 | BIT1;
00271     DSP_THETA = angle;
00272     answer = DSP_SIN;
00273     __set_PRIMASK(flg);
00274     return answer;
00275 }
00276 s16 DSP_cos(u16 angle)
00277 {
00278     s16 answer;
00279     int32_t flg = __get_PRIMASK();
00280     __disable_irq();
00281     DSP_SC = BIT2 | BIT1;
00282     DSP_THETA = angle;
00283     answer = DSP_COS;
00284     __set_PRIMASK(flg);
00285     return answer;
00286 }
00287 u16 DSP_rms(s16 a, s16 b)
00288 {
00289     u16 answer;
00290     int32_t flg = __get_PRIMASK();
00291     __disable_irq();
00292     DSP_DID = a;
00293     DSP_DIS = b;
00294     answer = DSP_REM;
00295     __set_PRIMASK(flg);
00296     return answer;
00297 }
00298 s16 DSP_mod(s32 x, s16 y)
00299 {
00300     s16 answer;
00301     int32_t flg = __get_PRIMASK();
00302     __disable_irq();
00303     DSP_SC = BIT1; /* 计算模式选择 Arctan */
00304     DSP_X = x;
00305     DSP_Y = y;
00306     answer = DSP_MOD;
00307     __set_PRIMASK(flg);
00308     return answer;
00309 }
00310 u16 DSP_sqrt(u32 val)
00311 {
00312     u16 answer;
00313     int32_t flg = __get_PRIMASK();
00314     __disable_irq();
00315     DSP_RAD = val;
00316     answer = DSP_SQRT;
00317     __set_PRIMASK(flg);
00318     return answer;
00319 }
00320 /***** (C) COPYRIGHT LINKO SEMICONDUCTOR *****/

```

3.55 source/lks32mc08x_exti.c 文件参考

文件名称: [lks32mc08x_exti.c](#)
 文件标识: 无
 内容摘要: EXTI外设驱动程序
 其它说明: 无

#include "lks32mc08x_exti.h"
 lks32mc08x_exti.c 的引用(Include)关系图:



函数

- void [EXTI_Trigger_Config](#) (GPIO_TypeDef *GPIOx, uint32_t GPIO_PinSource, uint16_t EXTI_Trigger)
 函数名称: void [EXTI_Trigger_Config\(GPIO_TypeDef *GPIOx, uint32_t GPIO_PinSource, uint16_t EXTI_Trigger\)](#)
- u8 [EXTI_GetIRQFlag](#) (GPIO_TypeDef *GPIOx, uint32_t GPIO_PinSource)
 函数名称: u8 [EXTI_GetIRQFlag\(GPIO_TypeDef *GPIOx, uint32_t GPIO_PinSource\)](#)
- void [EXTI_ClearIRQFlag](#) (GPIO_TypeDef *GPIOx, uint32_t GPIO_PinSource)
 函数名称: void [EXTI_ClearIRQFlag\(GPIO_TypeDef *GPIOx, uint32_t GPIO_PinSource\)](#)

3.55.1 详细描述

文件名称: [lks32mc08x_exti.c](#)
 文件标识: 无
 内容摘要: EXTI外设驱动程序
 其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2021年11月10日	1.0	YangZJ	创建

在文件 [lks32mc08x_exti.c](#) 中定义.

3.55.2 函数说明

3.55.2.1 EXTI_ClearIRQFlag()

```
void EXTI_ClearIRQFlag (
    GPIO_TypeDef * GPIOx,
    uint32_t GPIO_PinSource )
```

函数名称: void [EXTI_ClearIRQFlag\(GPIO_TypeDef *GPIOx, uint32_t GPIO_PinSource\)](#)

功能描述: 清除外部中断标志位

参见

被引用内容: 无

参数

输入参数:	GPIOx:GPIO对象可选(GPIO0~GPIO3) GPIO_PinSource:指定的PinSource可选(GPIO_PinSource_0~GPIO_PinSource_15)
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
if(EXTI_GetIRQFlag(GPIO0,GPIO_PinSource_0))//获取P0.0外部中断标志位
{
    EXTI_ClearIRQFlag(GPIO0,GPIO_PinSource_0);//清除P0.0外部中断标志位
}
```

Date	Version	Author	Description
2021年11月10日	1.0	YangZJ	创建

修改日志:

在文件 [lks32mc08x_exti.c](#) 第 124 行定义.

3.55.2.2 EXTI_GetIRQFlag()

```
u8 EXTI_GetIRQFlag (
    GPIO_TypeDef * GPIOx,
    uint32_t GPIO_PinSource )
```

函数名称: u8 EXTI_GetIRQFlag(GPIO_TypeDef *GPIOx, uint32_t GPIO_PinSource)

功能描述: 读取外部中断标志位

参见

被引用内容: 无

参数

输入参数:	GPIOx:GPIO对象可选(GPIO0~GPIO3) GPIO_PinSource:指定的PinSource可选(GPIO_PinSource_0~GPIO_PinSource_15)
输出参数:	无

返回

返回值: 中断标志, 高有效

注解

其它说明: 无

警告

无

示例代码:

```
if(EXTI_GetIRQFlag(GPIO0,GPIO_PinSource_0))//获取P0.0外部中断标志位
{
}
```

Date	Version	Author	Description
2021年11月10日	1.0	YangZJ	创建

修改日志:

在文件 [lks32mc08x_exti.c](#) 第 80 行定义.

3.55.2.3 EXTI_Trigger.Config()

```
void EXTI_Trigger_Config (
    GPIO_TypeDef * GPIOx,
    uint32_t GPIO_PinSource,
    uint16_t EXTI_Trigger )
```

函数名称: void [EXTI_Trigger_Config\(GPIO_TypeDef *GPIOx, uint32_t GPIO_PinSource, uint16_t EXTI_Trigger\)](#)

功能描述: GPIO外部触发极性选择

参见

被引用内容: [GPIO_TypeDef](#)

参数

输入参数:	GPIOx:GPIO对象可选(GPIO0) GPIO_PinSource:指定的PinSource可选(GPIO_PinSource_0~GPIO_PinSource_15) EXTI_Trigger:指定的触发极性

参见

EXTI_Trigger参数来源: [EXTIMode_TypeDef](#)

参数

输出参数:	无

返回

返回值: 无

注解

其它说明： 无

警告

无

示例代码:

```
EXTI_Trigger_Config(GPIO0,GPIO_PinSource_0,EXTI_Trigger_Negedge); //P0.0信号下降沿触发外部中断事件
```

Date	Version	Author	Description
2021年11月10日	1.0	YangZJ	创建

修改日志:

在文件 [lks32mc08x_exti.c](#) 第 38 行定义.

3.56 lks32mc08x_exti.c

[浏览该文件的文档.](#)

```
00001
00014 #include "lks32mc08x_exti.h"
00015
00038 void EXTI_Trigger_Config(GPIO_TypeDef *GPIOx, uint32_t GPIO_PinSource, uint16_t EXTI_Trigger)
00039 {
00040     if (GPIOx == GPIO0)
00041     {
00042         if (GPIO_PinSource > 16)
00043         {
00044             return;
00045         }
00046         if (GPIO_PinSource < 8)
00047         {
00048             EXTI->CR0 = (EXTI->CR0 & 0xffffc) | (EXTI_Trigger << (2 * GPIO_PinSource));
00049         }
00050         else
00051         {
00052             EXTI->CR1 = (EXTI->CR1 & 0xffffc) | (EXTI_Trigger << (2 * (GPIO_PinSource - 8)));
00053         }
00054     }
00055 }
00056
00080 u8 EXTI_GetIRQFlag(GPIO_TypeDef *GPIOx, uint32_t GPIO_PinSource)
00081 {
00082     if (GPIO_PinSource >= 16)
00083     {
00084         return 0;
00085     }
00086     if (GPIOx == GPIO0)
00087     {
00088         if (EXTI_IF & (1 << GPIO_PinSource))
00089         {
00090             return 1;
00091         }
00092         else
00093         {
00094             return 0;
00095         }
00096     }
00097     return 0;
```

```

00098 }
00099
00124 void EXTI_ClearIRQFlag(GPIO_TypeDef *GPIOx, uint32_t GPIO_PinSource)
00125 {
00126     if (GPIOx == GPIO0)
00127     {
00128         EXTI_IF = (1 << GPIO_PinSource);
00129     }
00130 }
00131
00132 /***** (C) COPYRIGHT LINKO SEMICONDUCTOR *****END OF FILE*****/

```

3.57 source/lks32mc08x_flash.c 文件参考

文件名称: lks32mc08x_flash.c
文件标识: 无
内容摘要: FLASH外设驱动程序
其它说明: 无

#include "lks32mc08x_flash.h"
lks32mc08x_flash.c 的引用(Include)关系图:



函数

- void **EraseSector** (u32 adr, u16 nvr)
函数名称: void *EraseSector(u32 adr, u16 nvr)*
- int **ProgramPage** (u32 adr, u32 sz, u8 *buf, u16 nvr)
函数名称: int *ProgramPage(u32 adr, u32 sz, u8 *buf, u16 nvr)*
- u32 **Read_Flash** (uint32_t adr, u16 nvr)
函数名称: u32 *Read_Flash(uint32_t adr, u16 nvr)*
- void **Read_More_Flash** (u32 adr, u32 sz, u32 *buf, u16 nvr)
函数名称: void *Read_More_Flash(u32 adr, u32 sz, u32 *buf, u16 nvr)*

变量

- volatile u32 **erase_flag** = 0
*flash*擦除操作启动变量
- volatile u32 **progm_flag** = 0
*flash*编程操作启动变量

3.57.1 详细描述

文件名称: [lks32mc08x_flash.c](#)
文件标识: 无
内容摘要: FLASH外设驱动程序
其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2020年12月15日	1.1	DengT	创建

修改日志:

在文件 [lks32mc08x_flash.c](#) 中定义.

3.57.2 函数说明

3.57.2.1 EraseSector()

```
void EraseSector (
    u32 adr,
    u16 nvr )
```

函数名称: void [EraseSector\(u32 adr, u16 nvr\)](#)

功能描述: Flash按扇区擦除操作

参见

被引用内容: 无

参数

输入参数:	adr: 扇区地址 (一个扇区512字节) nvr: 为0x800的时候擦除NVR, 为0时擦除主Flash erase_flag: 调用函数前必须赋值0x9A0D361F, 否则不执行擦除, 用来防程序跑飞
输出参数:	无

返回

返回值: 无

注解

其它说明： 无

警告

无

示例代码:

事例1:

```
erase_flag = 0x9A0D361F; //擦除解锁
EraseSector(0x00000000 , Flash.MAIN); //擦除flash的main区域即存储程序区域
erase_flag = 0x00; //擦除上锁
```

事例2:

```
erase_flag = 0x9A0D361F; //擦除解锁
EraseSector(0x00000000 , Flash.NVR); //擦除flash的NVR区域即独立的1KB用户存储区
erase_flag = 0x00; //擦除上锁
```

Date	Version	Author	Description
2020年12月15日	1.1	DengT	创建

修改日志:

在文件 [lks32mc08x_flash.c](#) 第 55 行定义.

3.57.2.2 ProgramPage()

```
int ProgramPage (
    u32 adr,
    u32 sz,
    u8 * buf,
    u16 nvr )
```

函数名称: **int ProgramPage(u32 adr, u32 sz, u8 *buf, u16 nvr)**

功能描述: **Flash编程操作**

参见

被引用内容： 无

参数

输入参数:	adr: 扇区地址 sz : 编程字节数量(0~512字节) buf: 要编程的数据数组指针 nvr: 为0x800的时候编程NVR, 为0时编程主Flash prog_m_flag: 调用函数前必须赋值0x9AFDA40C, 否则不执行编程, 用来防程序跑飞
输出参数:	无

返回

返回 值： 1： 编程成功； 0: 编程失败

注解

其它说明： 无

警告

无

示例代码:

```
u8 Flash_Buff[2] = {0x12,0x23};
事例1:
    erase_flag = 0x9A0D361F; //擦除解锁
    EraseSector(0x00000000 , Flash.MAIN); //擦除flash的main区域即存储程序区域
    erase_flag = 0x00; //擦除上锁

    prog_m_flag = 0x9AFDA40C; //编程解锁
    ProgramPage(0x00000000 , 2,Flash_Buff, Flash.MAIN); //编程flash的main区域即存储程序区域
    erase_flag = 0x00; //编程上锁
事例2:
    erase_flag = 0x9A0D361F; //擦除解锁
    EraseSector(0x00000000 , Flash.NVR); //擦除flash的NVR区域即独立的1KB用户存储区
    erase_flag = 0x00; //擦除上锁

    erase_flag = 0x9A0D361F; //编程解锁
    EraseSector(0x00000000,2,Flash_Buff,Flash.NVR); //编程flash的NVR区域即独立的1KB用户存储区
    erase_flag = 0x00; //编程上锁
```

Date	Version	Author	Description
2020年12月15日	1.1	DengT	创建

修改日志:

在文件 [lks32mc08x_flash.c](#) 第 134 行定义.

3.57.2.3 Read_Flash()

```
u32 Read_Flash (
    uint32_t adr,
    u16 nvr )
```

函数名称: u32 [Read_Flash\(uint32_t adr , u16 nvr\)](#)

功能描述: 读取FLASH数据函数

参见

被引用内容： 无

参数

输入参数:	adr: 扇区地址 (一个扇区512字节) nvr: 为0x800的时候擦除NVR, 为0时擦除主Flash
输出参数:	无

返回

返回 值: 读取的数据值

注解

其它说明: 无

警告

无

示例代码:

```
事例1:
u32 Flash_main = 0;
Flash_main = Read_Flash(0x00000000 , Flash_MAIN); //获取flash的main区域0x00000000地址一个word数据
事例2:
u32 Flash_nvr = 0;
Flash_nvr = Read_Flash(0x00000000 , Flash_NVR); //获取flash的NVR区域0x00000000地址一个word数据
```

Date	Version	Author	Description
2020年12月15日	1.1	DengT	创建

修改日志:

在文件 [lks32mc08x_flash.c](#) 第 [228](#) 行定义.

函数调用图:



这是这个函数的调用关系图:



3.57.2.4 Read_More_Flash()

```
void ReadMore.Flash (
    u32 adr,
    u32 sz,
    u32 * buf,
    u16 nvr )
```

函数名称: void [Read.More.Flash\(u32 adr, u32 sz, u32 *buf, u16 nvr\)](#)

功能描述: 读取FLASH多地址数据函数

参见
被引用内容: 无

参数

输入参数:	adr: 扇区地址 (一个扇区512字节) nvr: 为0x800, 读取MAIN数据, 为0x0, 读取NVR数据 buf:读取数据存储的地址 sz :读取sz个字
输出参数:	无

返回
返 回 值: 读取的数据值

注解
其它说明: 无

警告
无

示例代码:

```
u32 Flash_Read.Buff[4] = {0};
事例1: Read_More_Flash(0x00000000,4,Flash_Read.Buff,Flash.MAIN); //获取flash的main区域0x00000000地址4个word数据
事例2: Read_More_Flash(0x00000000,4,Flash_Read.Buff,Flash.NVR); //获取flash的NVR区域0x00000000地址4个word数据
```

Date	Version	Author	Description
2020年12月15日	1.1	DengT	创建

修改日志:

在文件 lks32mc08x_flash.c 第 272 行定义.

函数调用图:



3.57.3 变量说明

3.57.3.1 erase_flag

```
volatile u32 erase_flag = 0
```

flash擦除操作启动变量

在文件 lks32mc08x_flash.c 第 20 行定义.

3.57.3.2 prog_flag

```
volatile u32 prog_flag = 0
```

flash编程操作启动变量

在文件 lks32mc08x_flash.c 第 24 行定义.

3.58 lks32mc08x_flash.c

[浏览该文件的文档.](#)

```

00001
00015 #include "lks32mc08x_flash.h"
00016
00020 volatile u32 erase_flag = 0;
00024 volatile u32 prog_m_flag = 0;
00025
00055 void EraseSector(u32 adr, u16 nvr)
00056 {
00057     u32 key1, key2, key3;
00058
00059     key1 = 0x0;
00060     key2 = 0x0;
00061     key3 = 0x0;
00062     if (erase_flag == 0x9A0D361F)
00063     { // get flash_cfg addr
00064         key1 = 0xB00C060A;
00065         key2 = 0x6A013415;
00066         key3 = erase_flag ^ key1 ^ key2;
00067         REG32(key3) &= ~(BIT31 | BIT27 | BIT23 | BIT15 | BIT11);
00068         FLASH_ADDR = adr;
00069         REG32(key3) |= (0x80000000 | nvr);
00070         key1 = 0x0;
00071         key2 = 0x0;
00072         key3 = 0x0;
00073     }
00074
00075     if (erase_flag == 0x9A0D361F)
00076     { // get flash_erase addr
00077         key1 = 0xB001341A;
00078         key2 = 0x6A0C0615;
00079         key3 = erase_flag ^ key1 ^ key2;
00080         REG32(key3) = 0x7654DCBA; // trig sector erase, FLASH_ERASE
00081         FLASH_CFG &= ~0x80000000;
00082         erase_flag = 0x00000000;
00083         key1 = 0x0;
00084         key2 = 0x0;
00085         key3 = 0x0;
00086     }
00087     FLASH_CFG &= ~(BIT31 | BIT27 | BIT23 | BIT15 | BIT11);
00088     erase_flag = 0x00000000;
00089     key1 = 0x0;
00090     key2 = 0x0;
00091     key3 = 0x0;
00092 }
00093
00134 int ProgramPage(u32 adr, u32 sz, u8 *buf, u16 nvr)
00135 {
00136
00137     /* Add your Code */
00138     volatile u32 Addr, size;
00139     volatile u8 *p;
00140     volatile u8 t_rlt = 1;
00141
00142     u32 key1, key2, key3;
00143
00144     key1 = 0x0;
00145     key2 = 0x0;
00146     key3 = 0x0;
00147
00148     if (prog_m_flag == 0x9AFDA40C)
00149     {
00150         // get flash_cfg addr
00151         key1 = 0x6A5C040F;
00152         key2 = 0xB0A1A403;
00153
00154         key3 = prog_m_flag ^ key1 ^ key2;
00155         REG32(key3) &= ~(BIT31 | BIT27 | BIT23 | BIT15 | BIT11);
00156         FLASH_ADDR = adr;
00157         REG32(key3) |= (0x08000000 | nvr);
00158         prog_m_flag = 0;
00159         key1 = 0x0;
00160         key2 = 0x0;
00161         key3 = 0x0;
00162
00163         p = buf;
00164         size = (sz + 3) & ~3; // align the word
00165         Addr = adr;
00166         while (size)
00167         {
00168             FLASH_ADDR = (Addr & ~0x00000003); // address
00169             FLASH_WDATA = *p + ((*p + 1) << 8) + ((*p + 2) << 16) + ((*p + 3) << 24); // write data to
00170             flash

```

```

00170
00171     p += 4;
00172     size -= 4; // Go to next word
00173     Addr += 4;
00174 }
00175
00176     p = buf;
00177     size = (sz + 3) & ~3; // align the word
00178     Addr = adr;
00179     while (size)
00180     {
00181         u32 t_RData;
00182         FLASH_ADDR = (Addr & ~0x00000003); // address alignment
00183         t_RData = *p + (*p + 1) << 8 + (*p + 2) << 16 + (*p + 3) << 24;
00184         if (t_RData != FLASH_RDATA)
00185         {
00186             t_rlt = 0;
00187         }
00188
00189         p += 4;
00190         size -= 4;
00191         Addr += 4;
00192     }
00193 }
00194 FLASH_CFG &= ~(BIT31 | BIT27 | BIT23 | BIT15 | BIT11);
00195 key1 = 0x0;
00196 key2 = 0x0;
00197 key3 = 0x0;
00198 prog_flag = 0;
00199 return (t_rlt);
00200 }
00201
00228 u32 Read_Flash(uint32_t adr, u16 nvr)
00229 {
00230     uint32_t data;
00231     if (nvr == FLASH_MAIN) // 读取MIAN区域
00232     {
00233         FLASH_CFG &= ~BIT11;
00234         FLASH_ADDR = adr;
00235         data = FLASH_RDATA;
00236         FLASH_CFG &= ~BIT11; // 恢复MIAN操作
00237     }
00238     else
00239     {
00240         data = Read_NVR(adr);
00241     }
00242     return data;
00243 }
00244
00272 void Read_More_Flash(u32 adr, u32 sz, u32 *buf, u16 nvr)
00273 {
00274     volatile u32 Addr = 0;
00275     volatile u32 size = 0;
00276     volatile u32 *p;
00277     size = sz;
00278     Addr = adr;
00279     p = buf;
00280     while (size)
00281     {
00282         *p = Read_Flash(Addr, nvr);
00283         while (!(FLASH_READY & BIT0))
00284         {
00285         };
00286         size -= 1;
00287         Addr += 4;
00288         p += 1;
00289     }
00290 }

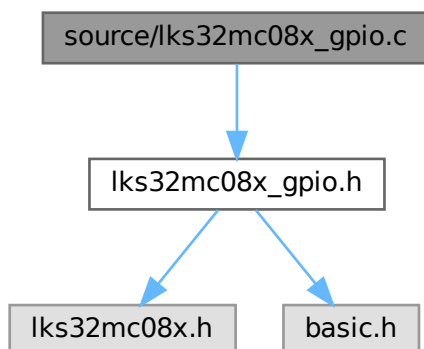
```

3.59 source/lks32mc08x_gpio.c 文件参考

文件名称: [lks32mc08x_gpio.c](#)
 文件标识: 无
 内容摘要: GPIO外设驱动程序
 其它说明: 无

```
#include "lks32mc08x_gpio.h"
```

lks32mc08x_gpio.c 的引用(Include)关系图:



函数

- void **GPIO_Init** (GPIO_TypeDef *GPIOx, GPIO_InitTypeDef *GPIO_InitStruct)
函数名称: void *GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct)*
- void **GPIO_StructInit** (GPIO_InitTypeDef *GPIO_InitStruct)
函数名称: void *GPIO_StructInit(GPIO_InitTypeDef* GPIO_InitStruct)*
- uint8_t **GPIO_ReadInputDataBit** (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
函数名称: uint8_t *GPIO_ReadInputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)*
- uint32_t **GPIO_ReadInputData** (GPIO_TypeDef *GPIOx)
函数名称: uint32_t *GPIO_ReadInputData(GPIO_TypeDef* GPIOx)*
- uint8_t **GPIO_ReadOutputDataBit** (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
函数名称: uint8_t *GPIO_ReadOutputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)*
- uint32_t **GPIO_ReadOutputData** (GPIO_TypeDef *GPIOx)
函数名称: uint32_t *GPIO_ReadOutputData(GPIO_TypeDef* GPIOx)*
- void **GPIO_SetBits** (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
函数名称: void *GPIO_SetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)*
- void **GPIO_ResetBits** (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
函数名称: void *GPIO_ResetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)*
- void **GPIO_WriteBit** (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin, BitAction BitVal)
函数名称: void *GPIO_WriteBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, BitAction BitVal)*
- void **GPIO_Write** (GPIO_TypeDef *GPIOx, uint32_t Val)
函数名称: void *GPIO_Write(GPIO_TypeDef* GPIOx, uint32_t Val)*
- void **GPIO_LockRegister** (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
函数名称: void *GPIO_LockRegister(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)*
- void **GPIO_UnlockRegister** (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
函数名称: void *GPIO_UnlockRegister(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)*
- void **GPIO_PinAFConfig** (GPIO_TypeDef *GPIOx, uint32_t GPIO_PinSource, uint32_t GPIO_AF)
函数名称: void *GPIO_PinAFConfig(GPIO_TypeDef* GPIOx, uint32_t GPIO_PinSource, uint32_t GPIO_AF)*
- void **GPIO_Config** (GPIO_TypeDef *GPIOx, u32 GPIO_PinSource, u8 GPIO_Modex, u8 GPIO_AF)
GPIO初始化

- void **GPIO_WritePinBit** (**GPIO_TypeDef** *GPIOx, u32 GPIO_PinSource, u8 Val)
GPIO写一个bit
- u8 **GPIO_ReadPinBit** (**GPIO_TypeDef** *GPIOx, u32 GPIO_PinSource)
GPIO读一个bit

3.59.1 详细描述

文件名称: [lks32mc08x_gpio.c](#)
 文件标识: 无
 内容摘要: GPIO外设驱动程序
 其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2016年6月19日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_gpio.c](#) 中定义.

3.59.2 函数说明

3.59.2.1 GPIO_Config()

```
void GPIO_Config (
    GPIO_TypeDef * GPIOx,
    u32 GPIO_PinSource,
    u8 GPIO_ModeX,
    u8 GPIO_AF )
```

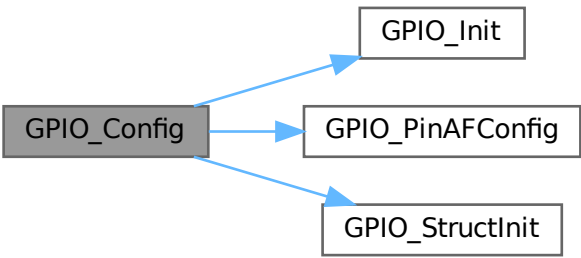
GPIO初始化

参数

<i>GPIOx</i>	GPIO对象,参数可选 (GPIO0~GPIO3)
<i>GPIO_PinSource</i>	指定的PinSource (GPIO_PinSource_0~GPIO_PinSource_15)
<i>GPIO_ModeX</i>	GPIO模式
<i>GPIO_AF</i>	GPIO复用功能

在文件 [lks32mc08x_gpio.c](#) 第 618 行定义.

函数调用图:



3.59.2.2 GPIO_Init()

```
void GPIO_Init (
    GPIO_TypeDef * GPIOx,
    GPIO_InitTypeDef * GPIO_InitStruct )
```

函数名称: void **GPIO_Init**(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct)

功能描述: GPIO初始化函数

参见
被引用内容: 无

参数

输入参数:	GPIOx:GPIO对象,参数可选 (GPIO0~GPIO3) GPIO_InitStruct: GPIO初始化结构体
输出参数:	无

返回
返回值: 无

注解
其它说明: 无

警告
无

示例代码:

```
GPIO_InitTypeDef GPIO_InitStructure;
//P2.11输入模式
GPIO_StructInit(&GPIO_InitStructure); //初始化结构体
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN; //GPIO输入模式
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_11;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init(GPIO2, &GPIO_InitStructure);
//P0.2输出模式
GPIO_StructInit(&GPIO_InitStructure); //初始化结构体
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT; //GPIO输出模式
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init(GPIO0, &GPIO_InitStructure);
//P3.2开漏输出
GPIO_StructInit(&GPIO_InitStructure); //初始化结构体
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT; //GPIO输出模式
GPIO_InitStructure.GPIO_ODENa = ENABLE; //使能开漏模式
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL; //关闭上拉, 配置为高阻态
GPIO_Init(GPIO3, &GPIO_InitStructure);
```

Date	Version	Author	Description
2016年4月18日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_gpio.c](#) 第 57 行定义.

这是这个函数的调用关系图:



3.59.2.3 GPIO_LockRegister()

```
void GPIO_LockRegister (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

函数名称: void [GPIO_LockRegister\(GPIO_TypeDef* GPIOx,uint16_t GPIO_Pin\)](#)

功能描述: GPIO寄存器保护锁定

参见

被引用内容: 无

参数

输入参数:	GPIOx:GPIO对象,参数可选 (GPIO0~GPIO3) GPIO_Pin:指定的Pin (GPIO_Pin_0~GPIO_Pin_15)
输出参数:	无

返回

返回值: 无

注解

其它说明: 上锁后GPIO输入/输出/上下拉/开漏/功能选择不能被修改

警告

无

示例代码:

```
GPIO_LockRegister(GPIO0,GPIO_Pin_0); //P0.0上锁
```

Date	Version	Author	Description
2016年4月18日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_gpio.c](#) 第 424 行定义.

3.59.2.4 GPIO_PinAFConfig()

```
void GPIO_PinAFConfig (
    GPIO_TypeDef * GPIOx,
    uint32_t GPIO_PinSource,
    uint32_t GPIO_AF )
```

函数名称: void [GPIO_PinAFConfig\(GPIO_TypeDef* GPIOx, uint32_t GPIO_PinSource, uint32_t GPIO_AF\)](#)

功能描述: GPIO复用功能选择

参见

被引用内容: 无

参数

输入参数:	GPIOx:GPIO对象,参数可选 (GPIO0~GPIO3)	
	GPIO_PadSource:指定的PadSource (GPIO_PinSource_0~GPIO_PinSource_15)	
	GPIO_AF:指定复用功能	
	复用功能名称	说明
	AF0.GPIO	模拟功能
	AF1.SYSAF	SYS_AF、CMP、CLOCK等数字信号输出功能
	AF2.HALL	HALL功能
	AF3.MCPWM	MCPWM功能
	AF4.UART	UART功能
	AF5.SPI	SPI功能
	AF6.I2C	I2C功能
	AF7.TIMER01	Timer0、Timer1功能
	AF8.TIMER23	Timer2、Timer3、QEP0、QEP1功能
	AF9.DEBUG	ADC trigger debug功能
输出参数:	AF10.CAN	CAN功能
	AF11.SIF	SIF功能
无		

返回

返回 值: 无

注解

其它说明: 无

警告

无

示例代码:

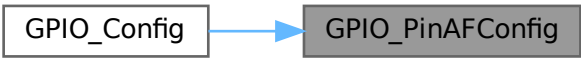
```
GPIO_PinAFConfig(GPIO1,GPIO_Pin_0, AF4_UART); //P1.0复用为串口功能
```

Date	Version	Author	Description
2016年4月18日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_gpio.c](#) 第 501 行定义.

这是这个函数的调用关系图:



3.59.2.5 GPIO_ReadInputData()

```
uint32_t GPIO_ReadInputData (
    GPIO_TypeDef * GPIOx )
```

函数名称: uint32_t [GPIO_ReadInputData\(GPIO_TypeDef* GPIOx\)](#)

功能描述: 读取GPIO的输入数据

参见
被引用内容: 无

参数

输入参数:	GPIOx:GPIO对象,参数可选 (GPIO0~GPIO3)
输出参数:	无

返回
返回值: 输入数据

注解
其它说明: 无

警告
无

示例代码:

```
u8 GPIO_IN_VALUE = 0;
GPIO_IN_VALUE = GPIO\_ReadInputData\(GPIO0\); //读取P0.0~P0.15输入电平
```

Date	Version	Author	Description
2016年4月18日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_gpio.c](#) 第 224 行定义.

3.59.2.6 GPIO_ReadInputDataBit()

```
uint8_t GPIO_ReadInputDataBit (  
    GPIO_TypeDef * GPIOx,  
    uint16_t GPIO_Pin )
```

函数名称: `uint8_t GPIO_ReadInputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)`

功能描述: 读取GPIO的指定Pin的输入值

参见

被引用内容: 无

参数

输入参数:	GPIOx:GPIO对象,参数可选 (GPIO0~GPIO3) GPIO_Pin:指定的Pin (GPIO_Pin_0~GPIO_Pin_15)
输出参数:	无

返回

返回值: 指定Pin输入数据

注解

其它说明: 无

警告

无

示例代码:

```
u8 GPIO_IN_VALUE = 0;  
GPIO_IN_VALUE = GPIO_ReadInputDataBit (GPIO0,GPIO_Pin_0); //读取P0.0输入电平 1为高电平, 0为低电平
```

Date	Version	Author	Description
2016年4月18日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_gpio.c](#) 第 189 行定义.

3.59.2.7 GPIO_ReadOutputData()

```
uint32_t GPIO_ReadOutputData (
    GPIO_TypeDef * GPIOx )
```

函数名称: `uint32_t GPIO_ReadOutputData(GPIO_TypeDef* GPIOx)`

功能描述: 读取GPIO的输出数据

参见

被引用内容: 无

参数

输入参数:	GPIOx:GPIO对象,参数可选 (GPIO0~GPIO3)
输出参数:	无

返回

返回值: 输出数据

注解

其它说明: 无

警告

无

示例代码:

```
u8 GPIO_OUT_VALUE = 0;
GPIO_OUT_VALUE = GPIO_ReadOutputData(GPIO0); //读取P0.0~P0.15输出电平
```

Date	Version	Author	Description
2016年4月18日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_gpio.c](#) 第 286 行定义.

3.59.2.8 GPIO_ReadOutputDataBit()

```
uint8_t GPIO_ReadOutputDataBit (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

函数名称: `uint8_t GPIO_ReadOutputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)`

功能描述: 读取GPIO的指定Pin的输出值

参见
被引用内容: 无

参数

输入参数:	GPIOx:GPIO对象,参数可选 (GPIO0~GPIO3) GPIO_Pin:指定的Pin (GPIO_Pin_0~GPIO_Pin_15)
输出参数:	无

返回
返回值: 指定IO输出数据

注解
其它说明: 无

警告
无

示例代码:

```
u8 GPIO_OUT_VALUE = 0;
GPIO_OUT_VALUE = GPIO_ReadOutputDataBit (GPIO0,GPIO_Pin_0); //读取P0.0输出电平 1为高电平，0为低电平
```

Date	Version	Author	Description
2016年4月18日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_gpio.c](#) 第 251 行定义.

3.59.2.9 GPIO_ReadPinBit()

```
u8 GPIO_ReadPinBit (
    GPIO_TypeDef * GPIOx,
    u32 GPIO_PinSource )
```

GPIO读一个bit

参数

GPIOx	GPIO对象,参数可选 (GPIO0~GPIO3)
GPIO_PinSource	指定的PinSource (GPIO_PinSource_0~GPIO_PinSource_15)

返回

0/1

在文件 [lks32mc08x_gpio.c](#) 第 655 行定义.

3.59.2.10 GPIO_ResetBits()

```
void GPIO_ResetBits (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

函数名称: void [GPIO_ResetBits\(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin\)](#)

功能描述: GPIO指定Pin置1

参见

被引用内容: 无

参数

输入参数:	GPIOx:GPIO对象,参数可选 (GPIO0~GPIO3) GPIO_Pin:指定的Pin (GPIO_Pin_0~GPIO_Pin_15)
输出参数:	无

返回

返回值: 无

注解

其它说明： 无

警告

无

示例代码:

```
GPIO_ResetBits (GPIO0,GPIO_Pin_0); //P0.0输出低电平
```

Date	Version	Author	Description
2016年4月18日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_gpio.c](#) 第 338 行定义.

函数调用图:



3.59.2.11 GPIO_SetBits()

```
void GPIO_SetBits (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

函数名称: void [GPIO_SetBits\(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin\)](#)

功能描述: GPIO指定Pin置1

参见

被引用内容： 无

参数

输入参数:	GPIOx:GPIO对象,参数可选 (GPIO0~GPIO3) GPIO_Pin:指定的Pin (GPIO_Pin_0~GPIO_Pin_15)
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

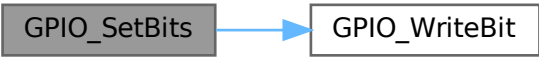
示例代码:

```
GPIO_SetBits(GPIO0,GPIO_Pin_0); //P0.0输出高电平
```

Date	Version	Author	Description
2016年4月18日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_gpio.c](#) 第 [312](#) 行定义.
函数调用图:



3.59.2.12 GPIO_StructInit()

```
void GPIO_StructInit (
    GPIO_InitTypeDef * GPIO_InitStruct )
```

函数名称: void [GPIO_StructInit\(GPIO_InitTypeDef* GPIO_InitStruct\)](#)
功能描述: GPIO结构体初始化

参见

被引用内容： 无

参数

输入参数:	GPIO_InitStruct: GPIO初始化结构体
输出参数:	无

返回

返回 值: 无

注解

其它说明： 无

警告

无

示例代码:

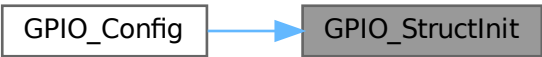
```
GPIO_InitTypeDef GPIO_InitStructure;  
GPIO_StructInit(&GPIO_InitStructure); //初始化结构体
```

Date	Version	Author	Description
2016年4月18日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_gpio.c](#) 第 159 行定义.

这是这个函数的调用关系图:



3.59.2.13 GPIO_UnlockRegister()

```
void GPIO_UnlockRegister (
    GPIO_TypeDef * GPIOx,
    uint16_t GPIO_Pin )
```

函数名称: void **GPIO_UnlockRegister**(GPIO_TypeDef* GPIOx,uint16_t GPIO_Pin)

功能描述: GPIO寄存器保护解锁

参见

被引用内容: 无

参数

输入参数:	GPIOx:GPIO对象,参数可选 (GPIO0~GPIO3) GPIO_Pin:指定的Pin (GPIO_Pin_0~GPIO_Pin_15)
输出参数:	无

返回

返回值: 无

注解

其它说明: 解锁后GPIO输入/输出/上下拉/开漏/功能选择可以被修改

警告

无

示例代码:

```
GPIO_UnlockRegister(GPIO0,GPIO_Pin_0); //P0.0解锁
```

Date	Version	Author	Description
2016年4月18日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_gpio.c](#) 第 454 行定义.

3.59.2.14 GPIO_Write()

```
void GPIO_Write (
```

```
GPIO_TypeDef * GPIOx,  
uint32_t Val )
```

函数名称: void **GPIO_Write**(GPIO_TypeDef* GPIOx, uint32_t Val)

功能描述: 向GPIO写入数据

参见

被引用内容: 无

参数

输入参数:	GPIOx:GPIO对象,参数可选 (GPIO0~GPIO3) BitVal:写入的Bit值 (0x00~0xffff)
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
GPIO_WriteBit (GPIO0, 0x01); //P0.0置1, 输出高电平, P0.1~P0.15值不变保持上一时刻状态
```

Date	Version	Author	Description
2016年4月18日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x-gpio.c](#) 第 398 行定义.

3.59.2.15 GPIO_WriteBit()

```
void GPIO_WriteBit (  
    GPIO_TypeDef * GPIOx,
```

```
uint16_t GPIO_Pin,
BitAction BitVal )
```

函数名称: void GPIO_WriteBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, BitAction BitVal)

功能描述: 向GPIO指定的Pin写入数据

参见
被引用内容: 无

参数

输入参数:	GPIOx:GPIO对象,参数可选 (GPIO0~GPIO3) GPIO_Pin:指定的Pin (GPIO_Pin_0~GPIO_Pin_15) BitVal:写入的Bit值 (Bit_RESET 或 Bit_SET)
输出参数:	无

返回
返回值: 无

注解
其它说明: 无

警告
无

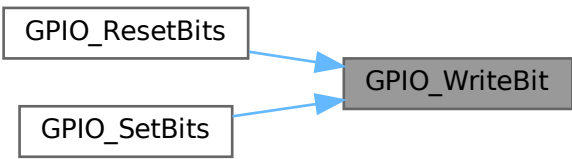
示例代码:
`GPIO_WriteBit(GPIO0,GPIO_Pin_0,Bit_RESET); //P0.0输出低电平`

Date	Version	Author	Description
2016年4月18日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_gpio.c](#) 第 365 行定义.

这是这个函数的调用关系图:



3.59.2.16 GPIO_WritePinBit()

```
void GPIO_WritePinBit (
    GPIO_TypeDef * GPIOx,
    u32 GPIO_PinSource,
    u8 Val )
```

GPIO写一个bit

参数

GPIOx	GPIO对象,参数可选 (GPIO0~GPIO3)
GPIO_PinSource	指定的PinSource (GPIO_PinSource_0~GPIO_PinSource_15)
Val	0/1

在文件 [lks32mc08x_gpio.c](#) 第 633 行定义.

3.60 lks32mc08x_gpio.c

[浏览该文件的文档.](#)

```
00001
00015 #include "lks32mc08x_gpio.h"
00016
00057 void GPIO_Init(GPIO_TypeDef *GPIOx, GPIO_InitTypeDef *GPIO_InitStruct)
00058 {
00059     uint32_t pinpos = 0x00, pos = 0x00, currentpin = 0x00, tempreg = 0x00;
00060
00061     /*----- Configure the port pins -----*/
00062     for (pinpos = 0x00; pinpos < 0x10; pinpos++)
00063     {
00064         pos = ((uint32_t)0x01) << pinpos;
00065
00066         /* Get the port pins position */
00067         currentpin = (GPIO_InitStruct->GPIO_Pin) & pos;
00068         if (currentpin == pos)
00069         {
00070             if (GPIO_InitStruct->GPIO_Mode == GPIO_Mode_IN)
00071             {
00072                 tempreg = GPIOx->PIE; /*使能输入*/
00073                 tempreg |= (BIT0 << pinpos);
00074                 GPIOx->PIE = tempreg;
00075
00076                 tempreg = GPIOx->POE; /*禁止输出*/
```

```

00077         tempreg &= ~(BIT0 << pinpos);
00078         GPIOx->POE = tempreg;
00079     }
00080     else if (GPIO.InitStruct->GPIO_Mode == GPIO_Mode_OUT)
00081     {
00082         tempreg = GPIOx->PIE; /*禁止输入*/
00083         tempreg &= ~(BIT0 << pinpos);
00084         GPIOx->PIE = tempreg;
00085
00086         tempreg = GPIOx->POE; /*使能输出*/
00087         tempreg |= (BIT0 << pinpos);
00088         GPIOx->POE = tempreg;
00089     }
00090     else if (GPIO.InitStruct->GPIO_Mode == GPIO_Mode_ANA)
00091     {
00092         tempreg = GPIOx->PIE; /*禁止输入*/
00093         tempreg &= ~(BIT0 << pinpos);
00094         GPIOx->PIE = tempreg;
00095
00096         tempreg = GPIOx->POE; /*禁止输出*/
00097         tempreg &= ~(BIT0 << pinpos);
00098         GPIOx->POE = tempreg;
00099     }
00100     else if (GPIO.InitStruct->GPIO_Mode == GPIO_Mode_IO)
00101     {
00102         tempreg = GPIOx->PIE; /*使能输入*/
00103         tempreg |= (BIT0 << pinpos);
00104         GPIOx->PIE = tempreg;
00105
00106         tempreg = GPIOx->POE; /*使能输出*/
00107         tempreg |= (BIT0 << pinpos);
00108         GPIOx->POE = tempreg;
00109     }
00110
00111     if (GPIO.InitStruct->GPIO_PuPd == GPIO_PuPd_UP)
00112     {
00113         tempreg = GPIOx->PUE;
00114         tempreg |= (BIT0 << pinpos);
00115         GPIOx->PUE = tempreg;
00116     }
00117     else if (GPIO.InitStruct->GPIO_PuPd == GPIO_PuPd_NOPULL)
00118     {
00119         tempreg = GPIOx->PUE;
00120         tempreg &= ~(BIT0 << pinpos);
00121         GPIOx->PUE = tempreg;
00122     }
00123
00124     tempreg = GPIOx->PODE;
00125     if (GPIO.InitStruct->GPIO_PODENA == (uint32_t)ENABLE)
00126     {
00127         tempreg |= (BIT0 << pinpos);
00128     }
00129     else
00130     {
00131         tempreg &= ~(BIT0 << pinpos);
00132     }
00133     GPIOx->PODE = tempreg;
00134 }
00135 }
00136 }
00137
00159 void GPIO_StructInit(GPIO_InitTypeDef *GPIO_InitStruct)
00160 {
00161     GPIO_InitStruct->GPIO_Pin = GPIO_Pin_NONE;
00162     GPIO_InitStruct->GPIO_Mode = GPIO_Mode_IN;
00163     GPIO_InitStruct->GPIO_PuPd = GPIO_PuPd_NOPULL;
00164     GPIO_InitStruct->GPIO_PODENA = DISABLE;
00165 }
00166
00189 uint8_t GPIO_ReadInputDataBit(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
00190 {
00191     uint8_t bitstatus = 0x00;
00192     if ((GPIOx->PDI & GPIO_Pin) != 0)
00193     {
00194         bitstatus = (uint8_t)Bit_SET;
00195     }
00196     else
00197     {
00198         bitstatus = (uint8_t)Bit_RESET;
00199     }
00200     return bitstatus;
00201 }
00202
00224 uint32_t GPIO_ReadInputData(GPIO_TypeDef *GPIOx)
00225 {
00226     return GPIOx->PDI;
00227 }

```

```

00228
00251 uint8_t GPIO_ReadOutputDataBit(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
00252 {
00253     uint8_t bitstatus = 0x00;
00254     if ((GPIOx->PDO & GPIO_Pin) != 0)
00255     {
00256         bitstatus = (uint8_t)Bit_SET;
00257     }
00258     else
00259     {
00260         bitstatus = (uint8_t)Bit_RESET;
00261     }
00262     return bitstatus;
00263 }
00264
00286 uint32_t GPIO_ReadOutputData(GPIO_TypeDef *GPIOx)
00287 {
00288     return GPIOx->PDO;
00289 }
00290
00312 void GPIO_SetBits(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
00313 {
00314     GPIO_WriteBit(GPIOx, GPIO_Pin, Bit_SET);
00315 }
00316
00338 void GPIO_ResetBits(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
00339 {
00340     GPIO_WriteBit(GPIOx, GPIO_Pin, Bit_RESET);
00341 }
00342
00365 void GPIO_WriteBit(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin, BitAction BitVal)
00366 {
00367     if (BitVal != Bit_RESET)
00368     {
00369         GPIOx->PDO |= GPIO_Pin;
00370     }
00371     else
00372     {
00373         GPIOx->PDO &= ~GPIO_Pin;
00374     }
00375 }
00376
00398 void GPIO_Write(GPIO_TypeDef *GPIOx, uint32_t Val)
00399 {
00400     GPIOx->PDO |= Val;
00401 }
00402
00424 void GPIO_LockRegister(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
00425 {
00426     LCKR_PRT = 0x5AC4; /* 关闭写保护 */
00427
00428     GPIOx->LCKR |= GPIO_Pin; /* 锁定GPIOx.Pin配置 */
00429
00430     LCKR_PRT = 0xFFFF; /* 开启写保护 */
00431 }
00432
00454 void GPIO_UnlockRegister(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
00455 {
00456     LCKR_PRT = 0x5AC4; /* 关闭写保护 */
00457
00458     GPIOx->LCKR &= (~GPIO_Pin); /* 解锁GPIOx.Pin配置 */
00459
00460     LCKR_PRT = 0xFFFF; /* 开启写保护 */
00461 }
00462
00501 void GPIO_PinAFConfig(GPIO_TypeDef *GPIOx, uint32_t GPIO_PinSource, uint32_t GPIO_AF)
00502 {
00503     uint32_t temp;
00504     switch (GPIO_PinSource)
00505     {
00506         /*F3210*/
00507         case 0:
00508         {
00509             temp = GPIOx->F3210 & 0xFFF0; /*get F321*/
00510             GPIOx->F3210 = temp + GPIO_AF; /*F321 + F0*/
00511             break;
00512         }
00513         case 1:
00514         {
00515             temp = GPIOx->F3210 & 0xFF0F;
00516             GPIOx->F3210 = temp + (GPIO_AF << 4);
00517             break;
00518         }
00519         case 2:
00520         { /* 2020.8.13 Repair bug HL */
00521             temp = GPIOx->F3210 & 0xF0FF;
00522             GPIOx->F3210 = temp + (GPIO_AF << 8);

```

```

00523         break;
00524     }
00525     case 3:
00526     {
00527         temp = GPIOx->F3210 & 0x0FFF;
00528         GPIOx->F3210 = temp + (GPIO_AF << 12);
00529         break;
00530     }
00531     /*F7654*/
00532     case 4:
00533     {
00534         temp = GPIOx->F7654 & 0xFFF0;
00535         GPIOx->F7654 = temp + GPIO_AF;
00536         break;
00537     }
00538     case 5:
00539     {
00540         temp = GPIOx->F7654 & 0xFF0F;
00541         GPIOx->F7654 = temp + (GPIO_AF << 4);
00542         break;
00543     }
00544     case 6:
00545     {
00546         temp = GPIOx->F7654 & 0xF0FF;
00547         GPIOx->F7654 = temp + (GPIO_AF << 8);
00548         break;
00549     }
00550     case 7:
00551     {
00552         temp = GPIOx->F7654 & 0x0FFF;
00553         GPIOx->F7654 = temp + (GPIO_AF << 12);
00554         break;
00555     }
00556     /*FBA98*/
00557     case 8:
00558     {
00559         temp = GPIOx->FBA98 & 0xFFF0;
00560         GPIOx->FBA98 = temp + GPIO_AF;
00561         break;
00562     }
00563     case 9:
00564     {
00565         temp = GPIOx->FBA98 & 0xFF0F;
00566         GPIOx->FBA98 = temp + (GPIO_AF << 4);
00567         break;
00568     }
00569     case 10:
00570     {
00571         temp = GPIOx->FBA98 & 0xF0FF;
00572         GPIOx->FBA98 = temp + (GPIO_AF << 8);
00573         break;
00574     }
00575     case 11:
00576     {
00577         temp = GPIOx->FBA98 & 0x0FFF;
00578         GPIOx->FBA98 = temp + (GPIO_AF << 12);
00579         break;
00580     }
00581     /*FFEDC*/
00582     case 12:
00583     {
00584         temp = GPIOx->FFEDC & 0xFFF0;
00585         GPIOx->FFEDC = temp + GPIO_AF;
00586         break;
00587     }
00588     case 13:
00589     {
00590         temp = GPIOx->FFEDC & 0xFF0F;
00591         GPIOx->FFEDC = temp + (GPIO_AF << 4);
00592         break;
00593     }
00594     case 14:
00595     {
00596         temp = GPIOx->FFEDC & 0xF0FF;
00597         GPIOx->FFEDC = temp + (GPIO_AF << 8);
00598         break;
00599     }
00600     case 15:
00601     {
00602         temp = GPIOx->FFEDC & 0x0FFF;
00603         GPIOx->FFEDC = temp + (GPIO_AF << 12);
00604         break;
00605     }
00606     default:
00607         break;
00608 }
00609 }

```



```

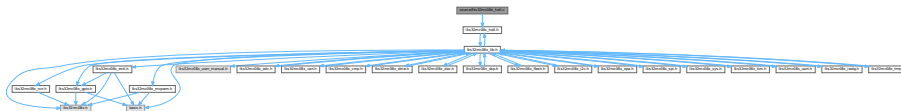
00610
00618 void GPIO_Config(GPIO_TypeDef *GPIOx, u32 GPIO_PinSource, u8 GPIO_Mode, u8 GPIO_AF)
00619 {
00620     GPIO_InitTypeDef GPIO_InitStructure;
00621     GPIO_StructInit(&GPIO_InitStructure);
00622     GPIO_InitStructure.GPIO_Mode = (GPIO_Mode_TypeDef)GPIO_Mode;
00623     GPIO_InitStructure.GPIO_Pin = 1 << GPIO_PinSource;
00624     GPIO_Init(GPIOx, &GPIO_InitStructure);
00625     GPIO_PinAFConfig(GPIOx, GPIO_PinSource, GPIO_AF);
00626 }
00633 void GPIO_WritePinBit(GPIO_TypeDef *GPIOx, u32 GPIO_PinSource, u8 Val)
00634 {
00635     u32 reg, pdo;
00636     pdo = 1 << GPIO_PinSource;
00637     reg = __get_PRIMASK();
00638     __disable_irq();
00639     if (Val == 0)
00640     {
00641         GPIOx->PDR &= ~pdo;
00642     }
00643     else
00644     {
00645         GPIOx->PDR |= pdo;
00646     }
00647     __set_PRIMASK(reg);
00648 }
00655 u8 GPIO_ReadPinBit(GPIO_TypeDef *GPIOx, u32 GPIO_PinSource)
00656 {
00657     u32 pdr;
00658     pdr = 1 << GPIO_PinSource;
00659     if (GPIOx->PDR & pdr)
00660     {
00661         return 1;
00662     }
00663     else
00664     {
00665         return 0;
00666     }
00667 }
00668 /***** (C) COPYRIGHT LINKO SEMICONDUCTOR *****/

```

3.61 source/lks32mc08x_hall.c 文件参考

文件名称: [lks32mc08x_hall.c](#)
 文件标识: 无
 内容摘要: HALL外设驱动程序
 其它说明: 无

#include "lks32mc08x_hall.h"
 lks32mc08x_hall.c 的引用(Include)关系图:



函数

- void [HALL_Init](#) (HALL_InitTypeDef *HALL_InitStruct)
函数名称: void [HALL_Init](#)(HALL_InitTypeDef* HALL_InitStruct)
- void [HALL_StructInit](#) (HALL_InitTypeDef *HALL_InitStruct)
函数名称: void [HALL_StructInit](#)(HALL_InitTypeDef* HALL_InitStruct)
- uint32_t [HALL_GetFilterValue](#) (void)
函数名称: uint32_t [HALL_GetFilterValue](#)(void)

- uint32_t [HALL_GetCaptureValue](#) (void)
函数名称: [uint32_t HALL_GetCaptureValue\(void\)](#)
- uint32_t [HALL_WIDCount](#) (void)
函数名称: [uint32_t HALL_WIDCount\(void\)](#)
- uint32_t [HALL_GetCount](#) (void)
函数名称: [uint32_t HALL_GetCount\(void\)](#)
- uint32_t [HALL_GetIRQFlag](#) (uint32_t tempFlag)
函数名称: [uint32_t HALL_GetIRQFlag\(uint32_t tempFlag\)](#)
- void [HALL_ClearIRQFlag](#) (uint32_t tempFlag)
函数名称: [uint32_t HALL_ClearIRQFlag\(uint32_t tempFlag\)](#)

3.61.1 详细描述

文件名称: [lks32mc08x_hall.c](#)
文件标识: 无
内容摘要: HALL外设驱动程序
其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2016年04月19日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_hall.c](#) 中定义.

3.61.2 函数说明

3.61.2.1 HALL_ClearIRQFlag()

```
void HALL_ClearIRQFlag (  
    uint32_t tempFlag )
```

函数名称: [uint32_t HALL_ClearIRQFlag\(uint32_t tempFlag\)](#)

功能描述: 清除HALL中断标志位

参见

被引用内容: 无

参数

输入参数:	tempFlag:	
	HALL_CAPTURE_EVENT	HALL信号变化中断
	HALL_OVERFLOW_EVENT	HALL计数溢出中断
输出参数:	无	

返回

返回 值: 无

注解

其它说明: 无

警告

无

示例代码:

```
if (HALL_GetIRQFlag (HALL_CAPTURE_EVENT)) //读取HALL信号变化中断标志位
{
    HALL_ClearIRQFlag (HALL_CAPTURE_EVENT); //清除HALL信号变化中断标志位
}
```

Date	Version	Author	Description
2022年5月20日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_hall.c](#) 第 242 行定义.

3.61.2.2 HALL_GetCaptureValue()

```
uint32_t HALL_GetCaptureValue (
    void )
```

函数名称: uint32_t [HALL_GetCaptureValue\(void\)](#)

功能描述: 取得HALL值，未滤波结果，即采样的原始HALL状态值

参见

被引用内容: 无

参数

输入参数:	无
输出参数:	无

返回

返回 值: 未滤波的HALL信号结果值, 第三位有效, 其它位恒为0

注解

其它说明: 无

警告

无

示例代码:

```
uint32_t HALL.Value = 0;  
HALL.Value = HALL.GetCaptureValue(); //读取未滤波的HALL结果值
```

Date	Version	Author	Description
2015年11月26日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_hall.c](#) 第 131 行定义.

3.61.2.3 HALL.GetCount()

```
uint32_t HALL.GetCount (  
    void )
```

函数名称: `uint32_t HALL.GetCount(void)`

功能描述: 读取实时HALL计数器值

参见

被引用内容: 无

参数

输入参数:	无
输出参数:	无

返回

返回 值: 实时HALL计数器值

注解

其它说明: 无

警告

无

示例代码:

```
uint32_t HALLValue = 0;  
HALLValue = HALL_GetCount(); //读取实时HALL计数器值
```

Date	Version	Author	Description
2015年11月26日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_hall.c](#) 第 181 行定义.

3.61.2.4 HALL_GetFilterValue()

```
uint32_t HALL_GetFilterValue (  
    void )
```

函数名称: uint32_t [HALL_GetFilterValue\(void\)](#)

功能描述: 取得HALL值, 滤波结果

参见

被引用内容: 无

参数

输入参数:	无
输出参数:	无

返回

返回 值: 经过滤波的HALL信号结果值, 第三位有效, 其它位恒为0

注解

其它说明： 无

警告

无

示例代码:

```
uint32_t HALLValue = 0;
HALLValue = HALL.GetFilterValue(); //读取经过滤波的HALL结果值
```

Date	Version	Author	Description
2015年11月26日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_hall.c](#) 第 106 行定义.

3.61.2.5 HALL.GetIRQFlag()

```
uint32_t HALL.GetIRQFlag (
    uint32_t tempFlag )
```

函数名称: uint32_t [HALL.GetIRQFlag\(uint32_t tempFlag\)](#)

功能描述: 读取HALL中断标志

参见

被引用内容： 无

参数

输入参数:	tempFlag:	
	HALL_CAPTURE_EVENT	HALL信号变化中断
	HALL_OVERFLOW_EVENT	HALL计数溢出中断
输出参数:	无	

返回

返 回 值: HALL中断标志 为1即对应标志位置位，为0即对应标志位未置位

注解

其它说明： 无

警告

无

示例代码:

```
if (HALL_GetIRQFlag (HALL_CAPTURE_EVENT)) //读取HALL信号变化中断标志位
{
}
```

Date	Version	Author	Description
2022年5月20日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_hall.c](#) 第 211 行定义.

3.61.2.6 HALL_Init()

```
void HALL_Init (
    HALL_InitTypeDef * HALL_InitStruct )
```

函数名称: void [HALL_Init\(HALL_InitTypeDef* HALL_InitStruct\)](#)

功能描述: HALL初始化函数

参见

被引用内容: [SYS_ModuleClockCmd\(\)](#)

参数

输入参数:	HALL_InitTypeDef HALL_InitStruct
输出参数:	无

返回

返回值: 无

注解

其它说明： 无

警告

无

示例代码:

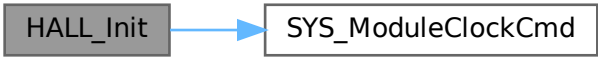
```
HALL_InitTypeDef HALL_InitStruct;
HALL_StructInit(&HALL_InitStruct);           // 初始化结构体
HALL_InitStruct.HALL_Ena = ENABLE;           // 模块使能
HALL_InitStruct.ClockDivision = HALL_CLK_DIV1; // 设置Hall模块时钟分频系数
HALL_InitStruct.CountTH = 9600000;           // Hall模块计数模值, 计数超过模值会产生超时中断
HALL_InitStruct.FilterLen = 512;             // Hall信号数字滤波长度 512个时钟周期
HALL_InitStruct.Filter75_Ena = ENABLE;       // Hall信号滤波方式, 7判5模式或者全1有效模式
HALL_InitStruct.Capture_IRQ_Ena = ENABLE;    // 捕捉中断使能
HALL_InitStruct.Overflow_IRQ_Ena = ENABLE;    // 超时中断使能
HALL_InitStruct.softIE = DISABLE;            // 软件中断使能
HALL_Init(&HALL_InitStruct);
```

Date	Version	Author	Description
2015年11月26日	1.0	cfwu	创建

修改日志:

在文件 `lks32mc08x_hall.c` 第 45 行定义.

函数调用图:



3.61.2.7 HALL_StructInit()

```
void HALL_StructInit (
    HALL_InitTypeDef * HALL_InitStruct )
```

函数名称: void HALL_StructInit(HALL_InitTypeDef* HALL_InitStruct)

功能描述: HALL结构体初始化

参见

被引用内容: 无

参数

输入参数:	HALL_InitTypeDef HALL_InitStruct
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
HALL_InitTypeDef HALL_InitStruct;  
HALL_StructInit(&HALL_InitStruct); //初始化结构体
```

Date	Version	Author	Description
2015年11月26日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_hall.c](#) 第 74 行定义.

3.61.2.8 HALL_WIDCount()

```
uint32_t HALL_WIDCount (  
    void )
```

函数名称: `uint32_t HALL_WIDCount(void)`

功能描述: 读取HALL信号变化事件时刻HALL计数值

参见

被引用内容: 无

参数

输入参数:	无
输出参数:	无

返回

返回 值： 获取三颗HALL信号任何一个信号发生变化事件时刻的HALL计数器的计数值

注解

其它说明： 无

警告

无

示例代码:

```
uint32_t HALL_Value = 0;
HALL_Value = HALL_WIDCount(); //读取HALL信号变化事件时HALL计数值
```

Date	Version	Author	Description
2015年11月26日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_hall.c](#) 第 156 行定义.

3.62 lks32mc08x_hall.c

[浏览该文件的文档.](#)

```
00001
00014 #include "lks32mc08x_hall.h"
00015
00045 void HALL_Init(HALL_InitTypeDef *HALL_InitStruct)
00046 {
00047     SYS_ModuleClockCmd(SYS_Module.HALL, ENABLE); // HALL时钟使能
00048
00049     HALL->CFG = (HALL_InitStruct->FilterLen) | (HALL_InitStruct->ClockDivision << 16) |
(HALL_InitStruct->Filter75.Ena << 20) | (HALL_InitStruct->HALL.Ena << 24) |
(HALL_InitStruct->Capture_IRQ.Ena << 28) | (HALL_InitStruct->OverFlow_IRQ.Ena << 29) |
(HALL_InitStruct->softIE << 30);
00050     HALL->TH = HALL_InitStruct->CountTH;
00051     HALL->INFO = 0;
00052 }
00053
00074 void HALL_StructInit(HALL_InitTypeDef *HALL_InitStruct)
00075 {
00076     HALL_InitStruct->FilterLen = 1023;
00077     HALL_InitStruct->ClockDivision = HALL_CLK_DIV1;
00078     HALL_InitStruct->Filter75.Ena = ENABLE;
00079     HALL_InitStruct->HALL.Ena = ENABLE;
00080     HALL_InitStruct->Capture_IRQ.Ena = ENABLE;
00081     HALL_InitStruct->OverFlow_IRQ.Ena = DISABLE;
00082     HALL_InitStruct->CountTH = 1000;
00083     HALL_InitStruct->softIE = DISABLE;
00084 }
00085
00106 uint32_t HALL_GetFilterValue(void)
00107 {
00108     return (HALL->INFO & 0x07);
00109 }
00110
```

```

00131 uint32_t HALL.GetCaptureValue(void)
00132 {
00133     return (HALL->INFO >> 8) & 0x07;
00134 }
00135
00156 uint32_t HALL.WIDCount(void)
00157 {
00158     return HALL->WIDTH;
00159 }
00160
00181 uint32_t HALL.GetCount(void)
00182 {
00183     return HALL->CNT;
00184 }
00185
00211 uint32_t HALL.GetIRQFlag(uint32_t tempFlag)
00212 {
00213     return (HALL->INFO & tempFlag);
00214 }
00215
00242 void HALL.ClearIRQFlag(uint32_t tempFlag)
00243 {
00244     HALL->INFO = tempFlag;
00245 }
00246
00247 /***** (C) COPYRIGHT LINKO SEMICONDUCTOR *****END OF FILE*****/

```

3.63 source/lks32mc08x_i2c.c 文件参考

文件名称: lks32mc08x_i2c.c

文件标识: 无

内容摘要: I2C外设驱动程序

其它说明: 无

```
#include "lks32mc08x_i2c.h"
```

lks32mc08x_i2c.c 的引用(Include)关系图:



函数

- void **I2C_Init** (**I2C_TypeDef** *I2Cx, **I2C_InitTypeDef** *I2C_InitStruct)
函数名称: void *I2C_Init(I2C_TypeDef* I2Cx, I2C_InitTypeDef* I2C_InitStruct)*
- void **I2C_StructInit** (**I2C_InitTypeDef** *I2C_InitStruct)
函数名称: void *I2C_StructInit(I2C_InitTypeDef* I2C_InitStruct)*
- void **I2C_SendData** (**I2C_TypeDef** *I2Cx, uint8_t n)
函数名称: void *I2C_SendData(I2C_TypeDef *I2Cx, uint8_t n)*
- uint8_t **I2C_ReadData** (**I2C_TypeDef** *I2Cx)
函数名称: uint8_t *I2C_ReadData(I2C_TypeDef *I2Cx)*
- static void **i2c_dma_delay_over** ()
函数名称: static void *i2c_dma_delay_over()*
- void **i2c_dma_state_over** ()
函数名称: void *i2c_dma_state_over()*
- static void **i2c_dma_state_start** ()
函数名称: static void *i2c_dma_state_start()*

- void **i2c_dma_tx** (u8 addr, u8 *data, u8 len)
函数名称: void **i2c_dma_tx**(u8 addr, u8 *data, u8 len)
- void **i2c_dma_rx** (u8 addr, u8 *data, u8 len)
函数名称: void **i2c_dma_rx**(u8 addr, u8 *data, u8 len)
- void **i2c_dma_init** (void)
函数名称: void **i2c_dma_init**(void)

变量

- static volatile u8 **i2c_dma_state** = 0
I2C发送状态

3.63.1 详细描述

文件名称: **lks32mc08x_i2c.c**
文件标识: 无
内容摘要: I2C外设驱动程序
其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2022年05月21日	1.0	YangZJ	创建

修改日志:

在文件 **lks32mc08x_i2c.c** 中定义.

3.63.2 函数说明

3.63.2.1 i2c_dma.delay_over()

```
static void i2c_dma_delay_over ( ) [static]
```

函数名称: static void **i2c_dma_delay_over**()

功能描述: I2C发送接收等待函数

参见

被引用内容: 无

参数

输入参数:	无
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

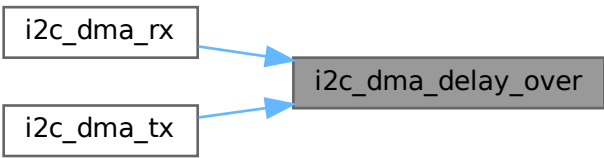
示例代码:

```
i2c_dma_delay_over();
```

Date	Version	Author	Description
2022年05月21日	1.0	YangZJ	创建

修改日志:

在文件 [lks32mc08x_i2c.c](#) 第 182 行定义.
这是这个函数的调用关系图:



3.63.2.2 i2c_dma_init()

```
void i2c_dma_init (
    void )
```

函数名称: void i2c_dma_init(void)

功能描述: DMA硬件初始化

参见
被引用内容: 无

参数

输入参数:	无
输出参数:	无

返回
返回值: 无

注解
其它说明: 无

警告
无

示例代码:
`i2c_dma_init();` // I2C的相关DMA模块硬件初始化

Date	Version	Author	Description
2022年05月21日	1.0	YangZJ	创建

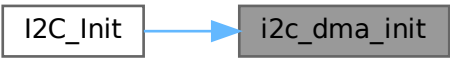
修改日志:

在文件 lks32mc08x_i2c.c 第 357 行定义.

函数调用图:



这是这个函数的调用关系图:



3.63.2.3 i2c_dma_rx()

```
void i2c_dma_rx (
    u8 addr,
    u8 * data,
    u8 len )
```

函数名称: void i2c_dma_rx(u8 addr, u8 *data, u8 len)

功能描述: I2C数据接收函数

参见
被引用内容: 无

参数

输入参数:	addr: I2C发送地址 data: 发送数据缓冲器 len: 发送数据长度
输出参数:	无

返回
返回值: 无

注解
其它说明: 无

警告
无

示例代码:

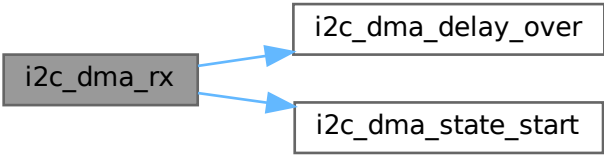
```
u8 i2c_Buff[8] = {0x00};
i2c_dma_rx(0x23 , i2c_Buff , 8); //将接收0x23地址的从机8个字节存储在i2c_Buff数组内
```

Date	Version	Author	Description
2022年05月21日	1.0	YangZJ	创建

修改日志:

在文件 [lks32mc08x_i2c.c](#) 第 320 行定义.

函数调用图:



3.63.2.4 i2c_dma_state_over()

```
void i2c_dma_state_over (
    void )
```

函数名称: [void i2c_dma_state_over\(\)](#)

功能描述: I2C发送完成函数

参见

被引用内容: 无

参数

输入参数:	无
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
void I2C0_IRQHandler      (void)
{
    I2C->SCR = 0;
    i2c_dma.state_over(); // iic传输完成
}
```

Date	Version	Author	Description
2022年05月21日	1.0	YangZJ	创建

修改日志:

在文件 [lks32mc08x_i2c.c](#) 第 227 行定义.

3.63.2.5 i2c_dma_state_start()

```
static void i2c_dma_state_start ( ) [static]
```

函数名称: static void [i2c_dma_state_start\(\)](#)

功能描述: I2C开始发送

参见

被引用内容: 无

参数

输入参数:	无
输出参数:	无

返回

返回 值: 无

注解

其它说明: 无

警告

无

示例代码:

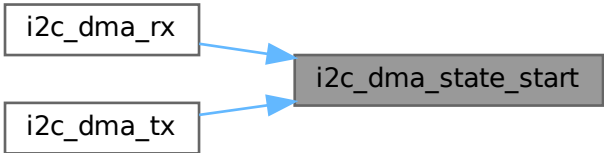
```
i2c_dma_state_start();
```

Date	Version	Author	Description
2022年05月21日	1.0	YangZJ	创建

修改日志:

在文件 [lks32mc08x_i2c.c](#) 第 251 行定义.

这是这个函数的调用关系图:



3.63.2.6 i2c_dma.tx()

```
void i2c_dma_tx (
    u8 addr,
    u8 * data,
    u8 len )
```

函数名称: `void i2c_dma.tx(u8 addr, u8 *data, u8 len)`

功能描述: I2C数据发送函数

参见

被引用内容: 无

参数

输入参数:	addr: I2C发送地址 data: 发送数据缓冲器 len: 发送数据长度
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

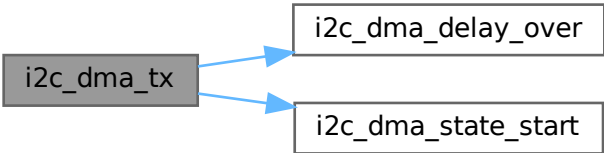
```
u8 i2c_Buff[8] = {0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88};
i2c_dma_tx(0x23 , i2c_Buff , 8); //发生给0x23地址的从机8个字节i2c_Buff数据
```

Date	Version	Author	Description
2022年05月21日	1.0	YangZJ	创建

修改日志:

在文件 [lks32mc08x_i2c.c](#) 第 279 行定义.

函数调用图:



3.63.2.7 I2C_Init()

```
void I2C_Init (
    I2C_TypeDef * I2Cx,
    I2C_InitTypeDef * I2C_InitStruct )
```

函数名称: void [I2C_Init\(I2C_TypeDef* I2Cx, I2C_InitTypeDef* I2C_InitStruct\)](#)

功能描述: I2C初始化

参见

被引用内容: [SYS_ModuleClockCmd\(\)](#) [SYS_SoftResetModule\(\)](#);

参数

输入参数:	I2Cx: I2C , I2C_InitTypeDef *I2C_InitStruct
输出参数:	无

返回

返回 值: 无

注解

其它说明: 无

警告

无

示例代码:

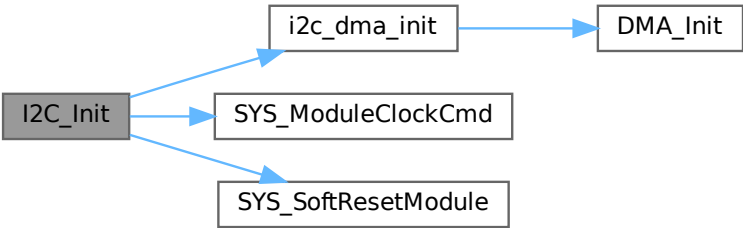
```
I2C_InitTypeDef I2C_InitStruct;  
I2C_StructInit(&I2C_InitStruct);  
I2C_InitStruct.ADRCMP      = DISABLE ; // I2C 硬件地址比较使能开关，只有在 DMA 模式下开启才有效。  
I2C_InitStruct.MST_MODE    = ENABLE  ; // I2C 主模式使能  
I2C_InitStruct.SLV_MODE    = DISABLE ; // I2C 从模式使能  
I2C_InitStruct.DMA         = ENABLE  ; // I2C DMA传输使能  
I2C_InitStruct.BaudRate    = 100000 ; // I2C 波特率  
I2C_InitStruct.IE          = ENABLE  ; // I2C 中断使能  
I2C_InitStruct.TC_IE       = ENABLE  ; // I2C 数据传输完成中断使能  
I2C_InitStruct.BUS_ERR_IE  = DISABLE ; // I2C 总线错误事件中断使能  
I2C_InitStruct.STOP_IE     = DISABLE ; // I2C STOP 事件中断使能  
I2C_InitStruct.BURST_NACK  = ENABLE  ; // I2C 传输，NACK 事件中断使能  
I2C_InitStruct.BURST_ADDR_CMP = DISABLE ; // I2C 传输，硬件地址匹配中断使能  
I2C_Init(I2C, &I2C_InitStruct);
```

Date	Version	Author	Description
2022年05月21日	1.0	YangZJ	创建

修改日志:

在文件 [lks32mc08x_i2c.c](#) 第 50 行定义.

函数调用图:



3.63.2.8 I2C_ReadData()

```
uint8_t I2C_ReadData (
    I2C_TypeDef * I2Cx )
```

函数名称: uint8_t I2C_ReadData(I2C_TypeDef *I2Cx)

功能描述: I2C读缓冲区数据

参见
被引用内容: 无

参数

输入参数:	I2Cx:I2C
输出参数:	无

返回
返回值: I2C接收的一字节数据

注解
其它说明: 无

警告
无

示例代码:

```
u8 I2C_Value = 0;
I2C_Value = I2C_ReadData (I2C); //I2C接收的一字节数据
```

Date	Version	Author	Description
2022年05月21日	1.0	YangZJ	创建

修改日志:

在文件 [lks32mc08x_i2c.c](#) 第 158 行定义.

3.63.2.9 I2C_SendData()

```
void I2C_SendData (
    I2C_TypeDef * I2Cx,
    uint8_t n )
```

函数名称: void [I2C_SendData\(I2C_TypeDef *I2Cx, uint8_t n\)](#)

功能描述: I2C发送数据

参见
被引用内容: 无

参数

输入参数:	I2Cx:I2C , n:要发生的一字节数据
输出参数:	无

返回
返回值: 无

注解
其它说明: 无

警告
无

示例代码:
[I2C_SendData](#) (I2C, 0x12); //发生0x12数据

Date	Version	Author	Description
2022年05月21日	1.0	YangZJ	创建

修改日志:

在文件 [lks32mc08x_i2c.c](#) 第 133 行定义.

3.63.2.10 I2C_StructInit()

```
void I2C_StructInit (
    I2C_InitTypeDef * I2C_InitStruct )
```

函数名称: `void I2C_StructInit(I2C_InitTypeDef* I2C_InitStruct)`

功能描述: I2C结构体初始化

参见

被引用内容: 无

参数

输入参数:	I2C_InitTypeDef *I2C_InitStruct
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
I2C\_InitTypeDef I2C_InitStruct;
I2C_StructInit (&I2C_InitStruct); //初始化结构体
```

Date	Version	Author	Description
2022年05月21日	1.0	YangZJ	创建

修改日志:

在文件 `lks32mc08x_i2c.c` 第 99 行定义。

3.63.3 变量说明

3.63.3.1 i2c_dma_state

```
volatile u8 i2c_dma_state = 0 [static]
```

I2C发送状态

在文件 `lks32mc08x_i2c.c` 第 16 行定义。

3.64 lks32mc08x_i2c.c

[浏览该文件的文档。](#)

```
00001
00014 #include "lks32mc08x_i2c.h"
00015
00016 static volatile u8 i2c_dma_state = 0;
00050 void I2C_Init(I2C_TypeDef *I2Cx, I2C_InitTypeDef *I2C_InitStruct)
00051 {
00052
00053     SYS_ModuleClockCmd(SYS_Module_I2C, ENABLE);
00054     SYS_SoftResetModule(SYS_Module_I2C);
00055
00056     {
00057         u32 BaudRate;
00058         BaudRate = ((u64)96000000 + ((I2C_InitStruct->BaudRate * 17) >> 1)) /
(I2C_InitStruct->BaudRate * 17) - 1;
00059         SYS_WR_PROTECT = 0x7A83; // 解除系统寄存器写保护
00060         SYS_CLK_DIV0 = BaudRate;
00061         SYS_WR_PROTECT = 0; // 解除系统寄存器写保护
00062     }
00063
00064     I2Cx->ADDR = I2C_InitStruct->ADRCMP << 7;
00065
00066     I2Cx->CFG = (I2C_InitStruct->IE << 7) | (I2C_InitStruct->TC_IE << 6) | (I2C_InitStruct->BUS_ERR_IE
<< 5) | (I2C_InitStruct->STOP_IE << 4) | (I2C_InitStruct->MST_MODE << 1) | (I2C_InitStruct->SLV_MODE <<
0);
00067
00068     I2Cx->SCR = 0;
00069
00070     I2Cx->MSCR = 0;
00071
00072     I2Cx->BCR = I2C_InitStruct->BURST_NACK << 7 | (I2C_InitStruct->BURST_ADDR_CMP << 6) |
(I2C_InitStruct->DMA << 5);
00073     if (I2C_InitStruct->DMA)
00074     {
00075         i2c_dma_init();
00076     }
00077 }
00078
00099 void I2C_StructInit(I2C_InitTypeDef *I2C_InitStruct)
00100 {
00101     I2C_InitStruct->ADRCMP = DISABLE; // I2C 硬件地址比较使能开关, 只有在 DMA 模式下开启才有效。
00102     I2C_InitStruct->MST_MODE = DISABLE; // I2C 主模式使能
00103     I2C_InitStruct->SLV_MODE = DISABLE; // I2C 从模式使能
00104     I2C_InitStruct->DMA = DISABLE; // I2C DMA传输使能
00105     I2C_InitStruct->BaudRate = 100000; // I2C 波特率
00106     I2C_InitStruct->IE = DISABLE; // I2C 中断使能
00107     I2C_InitStruct->TC_IE = DISABLE; // I2C 数据传输完成中断使能
00108     I2C_InitStruct->BUS_ERR_IE = DISABLE; // I2C 总线错误事件中断使能
00109     I2C_InitStruct->STOP_IE = DISABLE; // I2C STOP 事件中断使能
00110     I2C_InitStruct->BURST_NACK = DISABLE; // I2C 传输, NACK 事件中断使能
00111     I2C_InitStruct->BURST_ADDR_CMP = DISABLE; // I2C 传输, 硬件地址匹配中断使能
00112 }
00113
00133 void I2C_SendData(I2C_TypeDef *I2Cx, uint8_t n)
00134 {
00135     I2Cx->DATA = n;
00136 }
00137
00158 uint8_t I2C_ReadData(I2C_TypeDef *I2Cx)
```



```

00159 {
00160     return I2Cx->DATA;
00161 }
00162
00182 static void i2c.dma.delay_over()
00183 {
00184     u32 t = 100000000;
00185     while (i2c.dma.state) // 等待I2C发送完成
00186     {
00187         t--;
00188         if (t == 0)
00189         { // 避免程序卡死
00190             return;
00191         }
00192     }
00193     i2c.dma.state = 0;
00194     while (I2C0_MSCR & BIT3) // 等待I2C总线空闲
00195     {
00196         t--;
00197         if (t == 0)
00198         { // 避免程序卡死
00199             return;
00200         }
00201     }
00202 }
00203
00227 void i2c.dma.state_over()
00228 {
00229     i2c.dma.state = 0;
00230 }
00231
00251 static void i2c.dma.state.start()
00252 {
00253     i2c.dma.state = 1;
00254 }
00255
00279 void i2c.dma.tx(u8 addr, u8 *data, u8 len)
00280 {
00281     len = (len) ? len : 1;
00282     len = (len > 16) ? 16 : len; // 输入限制, 1-16
00283     i2c.dma.delay_over(); // 等待I2C完成
00284     I2C->ADDR = addr & 0x7f; // 从设备地址
00285     I2C->BCR = (I2C->BCR & 0xf0) | ((len - 1) & 0xf); // 设定数据长度
00286     I2C->SCR |= BIT2; // I2C传输方向
00287     I2C->DATA = 0;
00288     DMA_CH3->DMA_CCR &= ~BIT0;
00289     DMA_CH3->DMA_CMAR = (u32)data; // DMA发送数据地址
00290     DMA_CH3->DMA_CTMS = ((len + 1) << 16) | 1; // DMA搬运的数据长度
00291     DMA_CH3->DMA_CCR |= BIT4;
00292     DMA_CH3->DMA_CCR |= BIT0; // 配置DMA传输方向并使能DMA
00293     i2c.dma.state.start();
00294     I2C->MSCR = 1; // 触发I2C发送数据
00295 }
00296
00320 void i2c.dma.rx(u8 addr, u8 *data, u8 len)
00321 {
00322     len = (len) ? len : 1;
00323     len = (len > 16) ? 16 : len; // 输入限制, 1-16
00324     i2c.dma.delay_over(); // 等待I2C完成
00325     DMA_CH3->DMA_CCR &= ~BIT0;
00326     DMA_CH3->DMA_CMAR = (u32)data; // DMA发送数据地址
00327     DMA_CH3->DMA_CTMS = ((len) << 16) | 1; // DMA搬运的数据长度
00328     DMA_CH3->DMA_CCR &= ~BIT4; // DMA传输方向
00329     DMA_CH3->DMA_CCR |= BIT0; // 使能DMA
00330     I2C->ADDR = addr & 0x7f; // 从设备地址
00331     I2C->BCR = (I2C->BCR & 0xf0) | ((len - 1) & 0xf); // 设定数据长度
00332     I2C->SCR |= BIT4; // 收到数据后进行ACK回应
00333     I2C->SCR &= ~BIT2; // I2C传输方向
00334     i2c.dma.state.start();
00335     I2C->MSCR = 1; // 触发I2C发送数据
00336 }
00337
00357 void i2c.dma.init(void)
00358 {
00359     DMA_InitTypeDef DMA_InitStructure;
00360
00361     DMA_InitStructure.DMA_IRQ_EN = DISABLE; // DMA 中断使能 */
00362     DMA_InitStructure.DMA_DIR = 1; // DMA传输方向 0:外设至内存, 1:内存至外设 */
00363     DMA_InitStructure.DMA_CIRC = DISABLE; // DMA传输模式: 循环模式, 高有效 */
00364     DMA_InitStructure.DMA_PINC = DISABLE; // 外设地址每轮内是否递增, 高有效 */
00365     DMA_InitStructure.DMA_MINC = ENABLE; // 内存地址第二轮是否在第一轮地址的基础上递增, 高有效 */
00366     DMA_InitStructure.DMA_PBTW = 0; // 外设访问位宽, 0:byte, 1:half-word, 2:word */
00367     DMA_InitStructure.DMA_MBTW = 0; // 内存访问位宽, 0:byte, 1:half-word, 2:word */
00368     DMA_InitStructure.DMA_REQ_EN = DMA_CH3_I2C_REQ_EN; // 通道 x 三个硬件 DMA 请求使能, 高有效 */
00369     DMA_InitStructure.DMA_TIMES = 1; // DMA 通道 x 每轮数据搬运次数 1~511 */
00370     DMA_InitStructure.DMA_ROUND = 0; // DMA 通道 x 采样轮数 1~255 */
00371     DMA_InitStructure.DMA_CPAR = (u32)&I2C0_DATA; // DMA 通道 x 外设地址 */

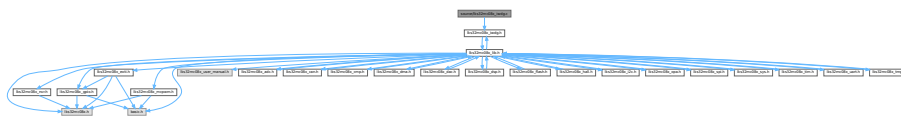
```

```
00372     DMA_InitStruct.DMA_CMAR    = 0;                               /* DMA 通道 x 内存地址 */
00373     DMA_Init(DMA_CH3, &DMA_InitStruct);
00374     DMA_IF = 0xff;
00375 }
```

3.65 source/lks32mc08x_iwdg.c 文件参考

文件名称: lks32mc08x_IWDG.c
文件标识: 无
内容摘要: IWDG外设驱动程序
其它说明: 无

#include "lks32mc08x_iwdg.h"
lks32mc08x_iwdg.c 的引用(Include)关系图:



函数

- void **IWDG_Init**(IWDG_InitTypeDef *this)
函数名称: void *IWDG_Init(IWDG_InitTypeDef *this)*
- void **IWDG_StrutInit**(IWDG_InitTypeDef *this)
函数名称: void *IWDG_StrutInit(IWDG_InitTypeDef *this)*
- void **IWDG_DISABLE**(void)
函数名称: void *IWDG_DISABLE(void)*
- void **IWDG_ENABLE**(void)
函数名称: void *IWDG_ENABLE(void)*
- void **IWDG_Feed**(uint32_t timeOut)
函数名称: void *IWDG_Feed(uint32_t timeOut)*

3.65.1 详细描述

文件名称: lks32mc08x_IWDG.c
文件标识: 无
内容摘要: IWDG外设驱动程序
其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2021年11月09日	1.0	YangZJ	创建

修改日志:

在文件 [lks32mc08x_iwdg.c](#) 中定义.

3.65.2 函数说明

3.65.2.1 IWDG_DISABLE()

```
void IWDG_DISABLE (
    void )
```

函数名称: [void IWDG_DISABLE\(void\)](#)

功能描述: 关闭看门狗模块

参见

被引用内容: 无

参数

输入参数:	无
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
IWDG\_DISABLE\(\);    //关闭看门狗模块
```

Date	Version	Author	Description
2021年11月09日	1.0	YangZJ	创建

修改日志:

在文件 [lks32mc08x_iwdg.c](#) 第 104 行定义.

3.65.2.2 IWDG_ENABLE()

```
void IWDG_ENABLE (
    void )
```

函数名称: void [IWDG_ENABLE\(void\)](#)

功能描述: 打开看门狗模块

参见

被引用内容: 无

参数

输入参数:	无
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
IWDG\_ENABLE\(\);    //打开看门狗模块
```

Date	Version	Author	Description
2021年11月09日	1.0	YangZJ	创建

修改日志:

在文件 [lks32mc08x_iwdg.c](#) 第 128 行定义.

3.65.2.3 IWDG_Feed()

```
void IWDG_Feed (
    uint32_t timeOut )
```

函数名称: `void IWDG_Feed(uint32_t timeOut)`

功能描述: 看门狗喂狗

参见

被引用内容: 无

参数

输入参数:	timeOut: 看门狗复位时间 WDT_RTH_2S 3s WDT_RTH_4S 2s WDT_RTH_8S 1s WDT_RTH_64S 0s
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
IWDG_Feed(WDT_RTH_2S); //看门狗喂狗，且设置下一次复位时间为2s
```

Date	Version	Author	Description
2021年11月09日	1.0	YangZJ	创建

修改日志:

在文件 [lks32mc08x_iwdg.c](#) 第 157 行定义.

3.65.2.4 IWDG_Init()

```
void IWDG_Init (
    IWDG_InitTypeDef * this )
```

函数名称: `void IWDG_Init(IWDG_InitTypeDef *this)`

功能描述: 看门狗初始化

参见

被引用内容: 无

参数

输入参数:	IWDG_InitTypeDef *this 看门狗配置结构体
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
IWDG_InitTypeDef IWDG_InitStructure;
IWDG_StructInit(&IWDG_InitStructure); //初始化结构体
IWDG_InitStructure.WDG_EN = ENABLE; //使能看门狗
IWDG_InitStructure.RTH = WDT_RTH_2S; //设置看门狗2s后复位芯片
IWDG_Init(&IWDG_InitStructure);
```

Date	Version	Author	Description
2021年11月09日	1.0	YangZJ	创建

修改日志:

在文件 [lks32mc08x_iwdg.c](#) 第 40 行定义.

3.65.2.5 IWDG_StrutInit()

```
void IWDG_StrutInit (
    IWDG_InitTypeDef * this )
```

函数名称: `void IWDG_StrutInit(IWDG_InitTypeDef *this)`

功能描述: 看门狗配置结构体初始化

参见

被引用内容: 无

参数

输入参数:	IWDG_InitTypeDef *this 看门狗配置结构体
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
IWDG_InitTypeDef IWDG_InitStruct;
IWDG_StrutInit (&IWDG_InitStruct); //初始化结构体
```

Date	Version	Author	Description
2021年11月09日	1.0	YangZJ	创建

修改日志:

在文件 `lks32mc08x_iwdg.c` 第 76 行定义.

3.66 lks32mc08x_iwdg.c

[浏览该文件的文档.](#)

```

00001
00015 #include "lks32mc08x_iwdg.h"
00016
00040 void IWDG_Init(IWDG_InitTypeDef *this)
00041 {
00042     SYS_WR_PROTECT = 0xCAFE;
00043     SYS_WDT_CLR     = 0x7988 | (this->RTH << 1) | (1);
00044     SYS_WR_PROTECT = 0x7a83;
00045     if (this->WDG_EN)
00046     {
00047         SYS_RST_CFG |= BIT0;
00048     }
00049     else
00050     {
00051         SYS_RST_CFG &= ~BIT0;
00052     }
00053     SYS_WR_PROTECT = 0;
00054 }
00055
00076 void IWDG_StructInit(IWDG_InitTypeDef *this)
00077 {
00078     this->WDG_EN = DISABLE;    // 独立看门狗使能
00079     this->RTH     = WDT_RTH_8S; // 看门狗超时复位时间 (21位计数器, 但低12恒位0)
00080                                     // SECOND2IWDGCNT输入参数, 秒
00081     // this->DWK_EN = DISABLE;    // 深度休眠定时唤醒使能
00082     // this->WTH = SECOND2IWDGCNT(1); // 看门狗定时唤醒时间, 大于复位时间后无效
00083 }
00084
00104 void IWDG_DISABLE(void)
00105 {
00106     SYS_RST_CFG &= ~BIT0;
00107 }
00108
00128 void IWDG_ENABLE(void)
00129 {
00130     SYS_RST_CFG |= BIT0;
00131 }
00132
00157 void IWDG_Feed(uint32_t timeOut)
00158 {
00159     SYS_WR_PROTECT = 0xCAFE; // 0xCAFE;
00160
00161     if (timeOut == (uint32_t)WDT_RTH_64S)
00162     {
00163         SYS_WDT_CLR = 0x7989; // 1 00 1
00164     }
00165     else if (timeOut == (uint32_t)WDT_RTH_8S)
00166     {
00167         SYS_WDT_CLR = 0x798B; // 1 01 1
00168     }
00169     else if (timeOut == (uint32_t)WDT_RTH_4S)
00170     {
00171         SYS_WDT_CLR = 0x798D; // 1 10 1
00172     }
00173     else // WDT_RTH_2S
00174     {
00175         SYS_WDT_CLR = 0x798F; // 1 11 1
00176     }
00177 }
00178
00179     SYS_WR_PROTECT = 0;
00180 }
00181 /***** (C) COPYRIGHT LINKO SEMICONDUCTOR *****/

```

3.67 source/lks32mc08x_mcpwm.c 文件参考

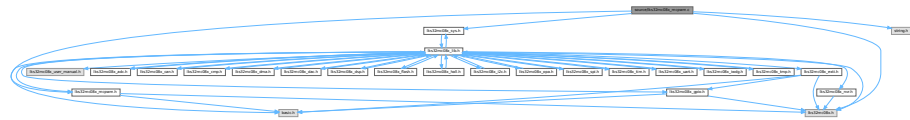
文件名称: `lks32mc08x_mcpwm.c`

文件标识: 无

内容摘要: MCPWM外设驱动程序

其它说明： 无

```
#include "lks32mc08x_mcpwm.h"
#include "lks32mc08x.h"
#include "lks32mc08x_sys.h"
#include "string.h"
lks32mc08x_mcpwm.c 的引用(Include)关系图:
```



函数

- void PWMOutputs (FuncState t.state)
函数名称: void PWMOutputs(FuncState t.state)
- void MCPWM_StructInit (MCPWM_InitTypeDef *MCPWM_InitStruct)
函数名称: void MCPWM_StructInit(MCPWM_InitTypeDef* MCPWM_InitStruct)
- void MCPWM_Init (MCPWM_InitTypeDef *MCPWM_InitStruct)
函数名称: void MCPWM_Init(MCPWM_InitTypeDef *MCPWM_InitStruct)
- u16 MCPWM_GetIRQFlag (uint32_t INT_flag)
函数名称: u16 MCPWM_GetIRQFlag(uint32_t INT_flag)
- void MCPWM_ClearIRQFlag (uint32_t INT_flag)
函数名称: void MCPWM_ClearIRQFlag(uint32_t INT_flag)

3.67.1 详细描述

文件名称: lks32mc08x_mcpwm.c
文件标识: 无
内容摘要: MCPWM外设驱动程序
其它说明: 无

版权所有
(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2019年05月21日	1.0	Howlet	创建

修改日志:

在文件 lks32mc08x_mcpwm.c 中定义.

3.67.2 函数说明

3.67.2.1 MCPWM_ClearIRQFlag()

```
void MCPWM_ClearIRQFlag (
    uint32_t INT_flag )
```

函数名称: void [MCPWM_ClearIRQFlag\(uint32_t INT_flag\)](#)

功能描述: 清除MCPWM中断标志

参见

被引用内容: 无

参数

输入参数:	MCPWM_T0_IRQ_IF	BIT0	计数器计数值回到MCPWM_TH中断标志位
	MCPWM_T1_IRQ_IF	BIT1	计数器计数值回到0中断标志位
	MCPWM_TH00_IRQ_IF	BIT2	计数器计数值回到MCPWM_TH00中断标志位
	MCPWM_TH01_IRQ_IF	BIT3	计数器计数值回到MCPWM_TH01中断标志位
	MCPWM_TH10_IRQ_IF	BIT4	计数器计数值回到MCPWM_TH10中断标志位
	MCPWM_TH11_IRQ_IF	BIT5	计数器计数值回到MCPWM_TH11中断标志位
	MCPWM_TH20_IRQ_IF	BIT6	计数器计数值回到MCPWM_TH20中断标志位
	MCPWM_TH21_IRQ_IF	BIT7	计数器计数值回到MCPWM_TH21中断标志位
	MCPWM_TH30_IRQ_IF	BIT8	计数器计数值回到MCPWM_TH30中断标志位
	MCPWM_TH31_IRQ_IF	BIT9	计数器计数值回到MCPWM_TH31中断标志位
	MCPWM_TMR0_IRQ_IF	BIT10	计数器计数值回到MCPWM_TMR0中断标志位
	MCPWM_TMR1_IRQ_IF	BIT11	计数器计数值回到MCPWM_TMR1中断标志位
	MCPWM_TMR2_IRQ_IF	BIT12	计数器计数值回到MCPWM_TMR0中断标志位
	MCPWM_TMR3_IRQ_IF	BIT13	计数器计数值回到MCPWM_TMR1中断标志位
	MCPWM_SHADE_IRQ_IF	BIT14	MCPWM_TH/MCPWM_↔TH00-MCPWM_TH31/↔MCPWM_TR0-MCPWM_↔TR3等寄存器更新到MCPWM实际运行系统中断标志位
输出参数:	无		

返回

返回值: 无

注解

其它说明： 无

警告

无

示例代码:

```
if (MCPWM_GetIRQFlag (MCPWM_T0_IRQ_EN) ) // 获取MCPWM的计数器计数值回到MCPWM_TH中断标志
{
    MCPWM_ClearIRQFlag (MCPWM_T0_IRQ_EN) ; // 清除MCPWM的计数器计数值回到MCPWM_TH中断标志
}
```

Date	Version	Author	Description
2022年04月20日	1.0	Zhu Jie	创建

修改日志:

在文件 [lks32mc08x_mcpwm.c](#) 第 [257](#) 行定义.

3.67.2.2 MCPWM_GetIRQFlag()

```
u16 MCPWM_GetIRQFlag (
    uint32_t INT_flag )
```

函数名称: [u16 MCPWM_GetIRQFlag\(uint32_t INT_flag\)](#)

功能描述: 获取MCPWM中断标志

参见

被引用内容： 无

参数

输入参数:	MCPWM_T0_IRQ_IF	BIT0	计数器计数值回到MCPWM_TH中断标志位
	MCPWM_T1_IRQ_IF	BIT1	计数器计数值回到0中断标志位
	MCPWM_TH00_IRQ_IF	BIT2	计数器计数值回到MCPWM_TH00中断标志位
	MCPWM_TH01_IRQ_IF	BIT3	计数器计数值回到MCPWM_TH01中断标志位
	MCPWM_TH10_IRQ_IF	BIT4	计数器计数值回到MCPWM_TH10中断标志位
	MCPWM_TH11_IRQ_IF	BIT5	计数器计数值回到MCPWM_TH11中断标志位
	MCPWM_TH20_IRQ_IF	BIT6	计数器计数值回到MCPWM_TH20中断标志位
	MCPWM_TH21_IRQ_IF	BIT7	计数器计数值回到MCPWM_TH21中断标志位
	MCPWM_TH30_IRQ_IF	BIT8	计数器计数值回到MCPWM_TH30中断标志位
	MCPWM_TH31_IRQ_IF	BIT9	计数器计数值回到MCPWM_TH31中断标志位
	MCPWM_TMR0_IRQ_IF	BIT10	计数器计数值回到MCPWM_TMR0中断标志位
	MCPWM_TMR1_IRQ_IF	BIT11	计数器计数值回到MCPWM_TMR1中断标志位
	MCPWM_TMR2_IRQ_IF	BIT12	计数器计数值回到MCPWM_TMR0中断标志位
	MCPWM_TMR3_IRQ_IF	BIT13	计数器计数值回到MCPWM_TMR1中断标志位
	MCPWM_SHADE_IRQ_IF	BIT14	MCPWM_TH/MCPWM_↔TH00-MCPWM_TH31/↔MCPWM_TR0-MCPWM_↔TR3等寄存器更新到MCPWM实际运行系统中断标志位
输出参数:	无		

返回

返回 值: 读对应标志位是否为1, 1位置位, 0为未置位

注解

其它说明： 无

警告

只有对应中断使能后， 改为才能读取， 如果对应中断未使能， 读取结果一直为0

示例代码:

```
if (MCPWM_GetIRQFlag (MCPWM_T0_IRQ_EN) ) // 获取MCPWM的计数器计数值回到MCPWM_TH中断标志
{
}
```

Date	Version	Author	Description
2022年04月20日	1.0	Zhu Jie	创建

修改日志:

在文件 [lks32mc08x_mcpwm.c](#) 第 191 行定义.

3.67.2.3 MCPWM_Init()

```
void MCPWM_Init (
    MCPWM_InitTypeDef * MCPWM_InitStruct )
```

函数名称: void [MCPWM_Init\(MCPWM_InitTypeDef *MCPWM_InitStruct\)](#)

功能描述: MCPWM初始化函数

参见

被引用内容: [SYS_ModuleClockCmd\(\)](#)

参数

输入参数:	MCPWM_InitTypeDef *MCPWM_InitStruct
输出参数:	无

返回

返回值: 无

注解

其它说明： 无

警告

无

示例代码:

```
MCPWM_InitTypeDef MCPWM_InitStructure;  
MCPWM_StructInit (&MCPWM_InitStructure); //初始化MCPWM相关结构体  
MCPWM_InitStructure.CLK_DIV = PWM_PRSC; // MCPWM时钟分频设置  
MCPWM_InitStructure.MCLK_EN = ENABLE; // 模块时钟开启  
MCPWM_InitStructure.MCPWM_Cnt_EN = ENABLE; // 主计数器开始计数使能开关  
MCPWM_InitStructure.MCPWM_WorkModeCH0 = CENTRAL_PWM_MODE; // MCPWM CH0工作模式: 中心对齐PWM模式  
MCPWM_InitStructure.MCPWM_WorkModeCH1 = CENTRAL_PWM_MODE; // 通道工作模式设置, 中心对齐或边沿对齐  
MCPWM_InitStructure.MCPWM_WorkModeCH2 = CENTRAL_PWM_MODE;  
MCPWM_InitStructure.MCPWM_WorkModeCH3 = CENTRAL_PWM_MODE;  
MCPWM_Init (&MCPWM_InitStructure);
```

Date	Version	Author	Description
2019年05月21日	1.0	Howlet	创建

修改日志:

在文件 [lks32mc08x_mcpwm.c](#) 第 107 行定义.

函数调用图:



3.67.2.4 MCPWM.StructInit()

```
void MCPWM_StructInit (  
    MCPWM_InitTypeDef * MCPWM_InitStructure )
```

函数名称: void [MCPWM_StructInit\(MCPWM_InitTypeDef* MCPWM_InitStructure\)](#)

功能描述: MCPWM结构体初始化

参见

被引用内容: 无

参数

输入参数:	MCPWM_InitTypeDef *MCPWM_InitStruct
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
MCPWM_InitTypeDef MCPWM_InitStructure;  
MCPWM_StructInit(&MCPWM_InitStructure); //初始化MCPWM相关结构体
```

Date	Version	Author	Description
2019年05月21日	1.0	Howlet	创建

修改日志:

在文件 [lks32mc08x_mcpwm.c](#) 第 73 行定义.

3.67.2.5 PWMOutputs()

```
void PWMOutputs (  
    FuncState t_state )
```

函数名称: void [PWMOutputs\(FuncState t_state\)](#)

功能描述: MCPWM波形输出使能与失能函数

参见

被引用内容: 无

参数

输入参数:	FuncState t_state ENABLE: 使能, DISABLE:使能
输出参数:	无



返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
PWMOutputs (ENABLE) //使能MCPWM输出
```

Date	Version	Author	Description
2019年05月21日	1.0	Howlet	创建

修改日志:

在文件 `lks32mc08x_mcpwm.c` 第 38 行定义.

3.68 lks32mc08x_mcpwm.c

[浏览该文件的文档.](#)

```
00001
00014 #include "lks32mc08x_mcpwm.h"
00015 #include "lks32mc08x.h"
00016 #include "lks32mc08x-sys.h"
00017 #include "string.h"
00018
00038 void PWMOutputs(FuncState t_state)
00039 {
00040     MCPWM_PRT = 0x0000DEAD;
00041
00042     if (t_state == ENABLE)
00043     {
00044         MCPWM_FAIL |= MCPWM_MOE_ENABLE_MASK;
00045     }
00046     else
00047     {
00048         MCPWM_FAIL &= MCPWM_MOE_DISABLE_MASK;
00049     }
00050     MCPWM_PRT = 0x0000CAFE;
00051 }
00052
00073 void MCPWM_StructInit(MCPWM_InitTypeDef *MCPWM_InitStruct)
00074 {
00075
00076     memset(MCPWM_InitStruct, 0, sizeof(MCPWM_InitTypeDef));
00077 }
00078
00107 void MCPWM_Init(MCPWM_InitTypeDef *MCPWM_InitStruct)
00108 {
00109     SYS_ModuleClockCmd(SYS_Module_MCPWM, ENABLE); // MCPWM时钟使能
00110
00111     MCPWM_PRT = 0x0000DEAD; /*enter password to unlock write protection */
```

```

00112     MCPWM.TCLK = MCPWM.InitStruct->CLK_DIV | (MCPWM.InitStruct->MCLK_EN << 2) |
(MCPWM.InitStruct->MCPWM.Cnt.EN << 3) | (MCPWM.InitStruct->GPIO.BKIN.Filter << 8) |
(MCPWM.InitStruct->CMP.BKIN.Filter << 12);
00113
00114     MCPWM.TH = MCPWM.InitStruct->MCPWM.PERIOD;
00115
00116     MCPWM.TMR0 = MCPWM.InitStruct->TriggerPoint0;
00117     MCPWM.TMR1 = MCPWM.InitStruct->TriggerPoint1;
00118     MCPWM.TMR2 = MCPWM.InitStruct->TriggerPoint2;
00119     MCPWM.TMR3 = MCPWM.InitStruct->TriggerPoint3;
00120
00121     MCPWM.IO01 = MCPWM.InitStruct->CH0N.Polarity.INV | (MCPWM.InitStruct->CH0P.Polarity.INV << 1) |
(MCPWM.InitStruct->CH0.NS << 2) | (MCPWM.InitStruct->CH0.PS << 3) | (MCPWM.InitStruct->CH0N.SCTRLN << 4)
| (MCPWM.InitStruct->CH0P.SCTRLP << 5) | (MCPWM.InitStruct->Switch.CH0N.CH0P << 6) |
(MCPWM.InitStruct->MCPWM.WorkModeCH0 << 7) | (MCPWM.InitStruct->CH1N.Polarity.INV << 8) |
(MCPWM.InitStruct->CH1P.Polarity.INV << 9) | (MCPWM.InitStruct->CH1.NS << 10) | (MCPWM.InitStruct->CH1.PS
<< 11) | (MCPWM.InitStruct->CH1N.SCTRLN << 12) | (MCPWM.InitStruct->CH1P.SCTRLP << 13) |
(MCPWM.InitStruct->Switch.CH1N.CH1P << 14) | (MCPWM.InitStruct->MCPWM.WorkModeCH1 << 15);
00122
00123     MCPWM.IO23 = MCPWM.InitStruct->CH2N.Polarity.INV | (MCPWM.InitStruct->CH2P.Polarity.INV << 1) |
(MCPWM.InitStruct->CH2.NS << 2) | (MCPWM.InitStruct->CH2.PS << 3) | (MCPWM.InitStruct->CH2N.SCTRLN << 4)
| (MCPWM.InitStruct->CH2P.SCTRLP << 5) | (MCPWM.InitStruct->Switch.CH2N.CH2P << 6) |
(MCPWM.InitStruct->MCPWM.WorkModeCH2 << 7) | (MCPWM.InitStruct->CH3N.Polarity.INV << 8) |
(MCPWM.InitStruct->CH3P.Polarity.INV << 9) | (MCPWM.InitStruct->CH3.NS << 10) | (MCPWM.InitStruct->CH3.PS
<< 11) | (MCPWM.InitStruct->CH3N.SCTRLN << 12) | (MCPWM.InitStruct->CH3P.SCTRLP << 13) |
(MCPWM.InitStruct->Switch.CH3N.CH3P << 14) | (MCPWM.InitStruct->MCPWM.WorkModeCH3 << 15);
00124
00125     MCPWM.PP = MCPWM.InitStruct->IO0.PPE | (MCPWM.InitStruct->IO1.PPE << 1) | (MCPWM.InitStruct->IO2.PPE
<< 2) | (MCPWM.InitStruct->IO3.PPE << 3);
00126
00127     MCPWM.FAIL = MCPWM.InitStruct->FAIL0.Signal.Sel | (MCPWM.InitStruct->FAIL1.Signal.Sel << 1) |
(MCPWM.InitStruct->FAIL0.Polarity << 2) | (MCPWM.InitStruct->FAIL1.Polarity << 3) |
(MCPWM.InitStruct->FAIL0.INPUT.EN << 4) | (MCPWM.InitStruct->FAIL1.INPUT.EN << 5)
00128
00129     | (MCPWM.InitStruct->DebugMode.PWM.out << 7) | (MCPWM.InitStruct->CH0P.default.output
<< 8) | (MCPWM.InitStruct->CH0N.default.output << 9) | (MCPWM.InitStruct->CH1P.default.output << 10) |
(MCPWM.InitStruct->CH1N.default.output << 11) | (MCPWM.InitStruct->CH2P.default.output << 12) |
(MCPWM.InitStruct->CH2N.default.output << 13) | (MCPWM.InitStruct->CH3P.default.output << 14) |
(MCPWM.InitStruct->CH3N.default.output << 15);
00130
00131     MCPWM.SDCFG = MCPWM.InitStruct->MCPWM.UpdateInterval | (MCPWM.InitStruct->MCPWM.T0.UpdateEN << 4) |
(MCPWM.InitStruct->MCPWM.T1.UpdateEN << 5) | (MCPWM.InitStruct->MCPWM.Auto.ERR.EN << 6);
00132
00133     MCPWM.DTH00 = MCPWM.InitStruct->DeadTimeCH0N;
00134     MCPWM.DTH01 = MCPWM.InitStruct->DeadTimeCH0P;
00135     MCPWM.DTH10 = MCPWM.InitStruct->DeadTimeCH1N;
00136     MCPWM.DTH11 = MCPWM.InitStruct->DeadTimeCH1P;
00137     MCPWM.DTH20 = MCPWM.InitStruct->DeadTimeCH2N;
00138     MCPWM.DTH21 = MCPWM.InitStruct->DeadTimeCH2P;
00139     MCPWM.DTH30 = MCPWM.InitStruct->DeadTimeCH3N;
00140     MCPWM.DTH31 = MCPWM.InitStruct->DeadTimeCH3P;
00141
00142     MCPWM.EIF = 0xffff;
00143     MCPWM.IE = MCPWM.InitStruct->T0.Update.INT.EN | (MCPWM.InitStruct->T1.Update.INT.EN << 1) |
(MCPWM.InitStruct->TH00.Match.INT.EN << 2) | (MCPWM.InitStruct->TH01.Match.INT.EN << 3) |
(MCPWM.InitStruct->TH10.Match.INT.EN << 4) | (MCPWM.InitStruct->TH11.Match.INT.EN << 5) |
(MCPWM.InitStruct->TH20.Match.INT.EN << 6) | (MCPWM.InitStruct->TH21.Match.INT.EN << 7) |
(MCPWM.InitStruct->TH30.Match.INT.EN << 8) | (MCPWM.InitStruct->TH31.Match.INT.EN << 9) |
(MCPWM.InitStruct->TMR0.Match.INT.EN << 10) | (MCPWM.InitStruct->TMR1.Match.INT.EN << 11) |
(MCPWM.InitStruct->TMR2.Match.INT.EN << 12) | (MCPWM.InitStruct->TMR3.Match.INT.EN << 13) |
(MCPWM.InitStruct->MCPWM.REG.UPDATE.INT.EN << 14);
00144
00145     MCPWM.EIE = (MCPWM.InitStruct->FAIL0.INT.EN << 4) | (MCPWM.InitStruct->FAIL1.INT.EN << 5);
00146
00147     MCPWM.UPDATE = 0xffff; /* write whatever value to trigger update */
00148
00149     MCPWM.UPDATE = 0x00;
00150
00151     MCPWM.PRT = 0x0000CAFE; /* write any value other than 0xDEAD to enable write protection */
00152 }
00153
00191 u16 MCPWM.GetIRQFlag(uint32_t INT_flag)
00192 {
00193     u8 MCPWMIRQFlag = 0;
00194     if (INT_flag <= BIT14)
00195     {
00196         if (MCPWM.IF & INT_flag & MCPWM.IE)
00197         {
00198             MCPWMIRQFlag = 1;
00199         }
00200     }
00201     else
00202     {
00203         MCPWMIRQFlag = 0;
00204     }
00205 }
00206

```

```

00207         if (MCPWM(EIF & (INT_flag >> 15) & MCPWM(EIE)
00208         {
00209             MCPWMIRQFlag = 1;
00210         }
00211         else
00212         {
00213             MCPWMIRQFlag = 0;
00214         }
00215     }
00216     return MCPWMIRQFlag;
00217 }
00218
00257 void MCPWM_ClearIRQFlag(uint32_t INT_flag)
00258 {
00259     if (INT_flag <= BIT14)
00260     {
00261         MCPWM(IF = INT_flag;
00262     }
00263     else
00264     {
00265         MCPWM(EIF = (INT_flag >> 15);
00266     }
00267 }
00268
00269 /***** (C) COPYRIGHT LINKO SEMICONDUCTOR *****/

```

3.69 source/lks32mc08x_opa.c 文件参考

文件名称: lks32mc08x_opa.c
文件标识: 无
内容摘要: OPA运放外设驱动程序
其它说明: 无

#include "lks32mc08x_opa.h"
lks32mc08x_opa.c 的引用(Include)关系图:



函数

- void **OPA_Init** (enumOPA opa, OPA_InitTypeDef *OPA_InitStruct)
函数名称: void *OPA_Init(enumOPA opa, OPA_InitTypeDef* OPA_InitStruct)*
- void **OPA_StructInit** (OPA_InitTypeDef *OPA_InitStruct)
函数名称: void *OPA_StructInit(OPA_InitTypeDef* OPA_InitStruct)*
- void **OPA_OUT** (enumOPA opa, FuncState state)
函数名称: void *OPA_OUT(enumOPA opa, FuncState state)*

3.69.1 详细描述

文件名称: lks32mc08x_opa.c
文件标识: 无
内容摘要: OPA运放外设驱动程序
其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2016年06月02日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_opa.c](#) 中定义.

3.69.2 函数说明

3.69.2.1 OPA_Init()

```
void OPA_Init (
    enumOPA opa,
    OPA_InitTypeDef * OPA_InitStruct )
```

函数名称: void [OPA_Init\(enumOPA opa, OPA_InitTypeDef* OPA_InitStruct\)](#)

功能描述: OPA初始化函数

参见

被引用内容: 无

参数

输入参数:	enumOPA opa , OPA_InitTypeDef *OPA_InitStruct
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
OPA_InitTypeDef OPA_InitStructure;  
OPA_StructInit(&OPA_InitStructure);           //初始化结构体  
OPA_InitStructure.OPA_CLEna = ENABLE;         //使能OPA  
OPA_InitStructure.OPA_Gain = PGA_GAIN_20;     //OPA内部增益电阻选择200: 10.2  
OPA_Init(OPA0, &OPA_InitStructure);
```

Date	Version	Author	Description
2016年06月02日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_opa.c](#) 第 40 行定义.

3.69.2.2 OPA_OUT()

```
void OPA_OUT (  
    enumOPA opa,  
    FuncState state )
```

函数名称: void [OPA_OUT\(enumOPA opa, FuncState state\)](#)

功能描述: OPA外部输出，输出到外部P2.7引脚

参见

被引用内容: 无

参数

输入参数:	enumOPA opa, FuncState state
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
OPA_OUT(OPA0, ENABLE); //将OPA0输出通过外部P2.7引脚送出
```

Date	Version	Author	Description
2022年04月20日	1.0	HuangMG	创建

修改日志:

在文件 lks32mc08x_opa.c 第 130 行定义.

3.69.2.3 OPA_StructInit()

```
void OPA_StructInit (
    OPA_InitTypeDef * OPA_InitStruct )
```

函数名称: void OPA_StructInit(OPA_InitTypeDef* OPA_InitStruct)

功能描述: OPA结构体初始化

参见

被引用内容: 无

参数

输入参数:	OPA_InitTypeDef *OPA_InitStruct
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
OPA_InitTypeDef OPA_InitStruct;
OPA_StructInit (&OPA_InitStruct); //初始化结构体
```

Date	Version	Author	Description
2016年06月02日	1.0	cfwu	创建

修改日志:

在文件 `lks32mc08x_opa.c` 第 105 行定义.

3.70 lks32mc08x_opa.c

[浏览该文件的文档.](#)

```

00001
00015 #include "lks32mc08x_opa.h"
00016
00040 void OPA_Init(enumOPA opa, OPA_InitTypeDef *OPA_InitStruct)
00041 {
00042     uint32_t tmp1, tmp2;
00043     SYS_WR_PROTECT = 0x7a83; /* 解除系统寄存器写保护 */
00044     tmp1           = SYS_AFE_REG0;
00045     tmp2           = SYS_AFE_REG5;
00046
00047     if (opa == OPA0)
00048     {
00049         tmp1 &= ~(BIT0 | BIT1);
00050         tmp1 |= OPA_InitStruct->OPA_Gain;
00051
00052         tmp2 &= ~BIT2;
00053         tmp2 |= OPA_InitStruct->OPA_CLEna << 2;
00054     }
00055     else if (opa == OPA1)
00056     {
00057         tmp1 &= ~(BIT2 | BIT3);
00058         tmp1 |= (OPA_InitStruct->OPA_Gain << 2);
00059
00060         tmp2 &= ~BIT3;
00061         tmp2 |= (OPA_InitStruct->OPA_CLEna << 3);
00062     }
00063     else if (opa == OPA2)
00064     {
00065         tmp1 &= ~(BIT4 | BIT5);
00066         tmp1 |= (OPA_InitStruct->OPA_Gain << 4);
00067
00068         tmp2 &= ~BIT4;
00069         tmp2 |= (OPA_InitStruct->OPA_CLEna << 4);
00070     }
00071     else if (opa == OPA3)
00072     {
00073         tmp1 &= ~(BIT6 | BIT7);
00074         tmp1 |= (OPA_InitStruct->OPA_Gain << 6);
00075
00076         tmp2 &= ~BIT5;
00077         tmp2 |= (OPA_InitStruct->OPA_CLEna << 5);
00078     }
00079
00080     SYS_AFE_REG0 = tmp1;
00081     SYS_AFE_REG5 = tmp2;
00082     SYS_WR_PROTECT = 0x0; /* 开启系统寄存器写保护 */
00083 }
00084
00105 void OPA_StructInit(OPA_InitTypeDef *OPA_InitStruct)
00106 {
00107     OPA_InitStruct->OPA_Gain = PGA_GAIN_4P25;
00108     OPA_InitStruct->OPA_CLEna = DISABLE;
00109 }
00110
00130 void OPA_OUT(enumOPA opa, FuncState state)
00131 {
00132     SYS_WR_PROTECT = 0x7a83; /* 解除系统寄存器写保护 */
00133     if (state != DISABLE)
00134     {
00135         if (opa == OPA0)
00136         {
00137             SYS_AFE_REG2 = 0x01;
00138         }
00139         else if (opa == OPA1)

```

```

00140     {
00141         SYS_AFE_REG2 = 0x02;
00142     }
00143     else if (opa == OPA2)
00144     {
00145         SYS_AFE_REG2 = 0x03;
00146     }
00147     else if (opa == OPA3)
00148     {
00149         SYS_AFE_REG2 = 0x04;
00150     }
00151 }
00152 else
00153 {
00154     SYS_AFE_REG2 = 0x00;
00155 }
00156 SYS_WR_PROTECT = 0x0; /* 开启系统寄存器写保护 */
00157 }
00158
00159 /***** (C) COPYRIGHT LINKO SEMICONDUCTOR *****/

```

3.71 source/lks32mc08x_spi.c 文件参考

文件名称: [lks32mc08x_spi.c](#)
 文件标识: 无
 内容摘要: SPI外设驱动程序
 其它说明: 无

#include "lks32mc08x_spi.h"
 lks32mc08x_spi.c 的引用(Include)关系图:



函数

- void **SPI_Init** (SPI_TypeDef *SPIx, SPI_InitTypeDef *SPI_InitStruct)
函数名称: void *SPI_Init(SPI_TypeDef* SPIx, SPI_InitTypeDef* SPI_InitStruct)*
- void **SPI_StructInit** (SPI_InitTypeDef *SPI_InitStruct)
函数名称: void *SPI_StructInit(SPI_InitTypeDef* SPI_InitStruct)*
- void **SPI_SendData** (SPI_TypeDef *SPIx, uint8_t n)
函数名称: void *SPI_SendData(SPI_TypeDef* SPIx, uint8_t n)*
- uint8_t **SPI_ReadData** (SPI_TypeDef *SPIx)
函数名称: uint8_t *SPI_ReadData(SPI_TypeDef* SPIx)*
- uint8_t **SPI_GetIRQFlag** (SPI_TypeDef *SPIx, u8 IRQ_Flag)
函数名称: uint8_t *SPI_GetIRQFlag(SPI_TypeDef* SPIx, u8 IRQ_Flag)*
- void **SPI_ClearIRQFlag** (SPI_TypeDef *SPIx, u8 IRQ_Flag)
函数名称: void *SPI_ClearIRQFlag(SPI_TypeDef* SPIx, u8 IRQ_Flag)*

3.71.1 详细描述

文件名称: lks32mc08x_spi.c
文件标识: 无
内容摘要: SPI外设驱动程序
其它说明: 无

版权所有
(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2016年05月21日	1.0	cfwu	创建

修改日志:

在文件 lks32mc08x_spi.c 中定义.

3.71.2 函数说明

3.71.2.1 SPI_ClearIRQFlag()

```
void SPI_ClearIRQFlag (
    SPI_TypeDef * SPIx,
    u8 IRQ_Flag )
```

函数名称: void SPI_ClearIRQFlag(SPI_TypeDef *SPIx,u8 IRQ_Flag)

功能描述: 清除SPI中断标志位

参见
被引用内容: 无

参数

输入参数:	SPIx: SPI_IRQ_Flag:	
	SPI_IF_TranDone	传输完成中断标志位
	SPI_IF_SSErr	传输异常中断标志位
	SPI_IF_DataOver	传输溢出中断标志位
输出参数:	无	

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
if(SPI_GetIRQFlag(SPI,SPI_IF_TranDone))//获取传输完成中断标志位
{
    SPI_ClearIRQFlag(SPI,SPI_IF_TranDone); //清除传输完成中断标志位
}
```

Date	Version	Author	Description
2022年04月20日	1.0	Zhu Jie	创建

修改日志:

在文件 [lks32mc08x_spi.c](#) 第 222 行定义.

3.71.2.2 SPI_GetIRQFlag()

```
uint8_t SPI_GetIRQFlag (
    SPI_TypeDef * SPIx,
    u8 IRQ_Flag )
```

函数名称: `uint8_t SPI_GetIRQFlag(SPI_TypeDef *SPIx,u8 IRQ_Flag)`

功能描述: 获取SPI中断标志位

参见

被引用内容: 无

参数

输入参数:	SPIx: SPI_IRQ_Flag:	
	SPI_IF_TranDone	传输完成中断标志位
	SPI_IF_SSErr	传输异常中断标志位
	SPI_IF_DataOver	传输溢出中断标志位
输出参数:	无	

返回

返回 值: 无

注解

其它说明: 无

警告

只有对应中断使能后，改为才能读取，如果对应中断未使能，读取结果一直为0

示例代码:

```
if(SPI_GetIRQFlag(SPI,SPI_IF_TranDone))//获取传输完成中断标志位
{
}
```

Date	Version	Author	Description
2022年04月20日	1.0	Zhu Jie	创建

修改日志:

在文件 [lks32mc08x_spi.c](#) 第 181 行定义.

3.71.2.3 SPI_Init()

void SPI_Init (

 SPI_TypeDef * SPIx,

 SPI_InitTypeDef * SPI_InitStruct)

函数名称: void [SPI_Init\(SPI_TypeDef* SPIx, SPI_InitTypeDef* SPI_InitStruct\)](#)

功能描述: SPI初始化函数

参见

被引用内容: [SYS_ModuleClockCmd\(\)](#)

参数

输入参数:	SPIx: SPI , SPI_InitTypeDef *SPI_InitStruct
输出参数:	无

返回

返回 值: 无

注解

其它说明: 无

警告

无

示例代码:

```
SPI_InitTypeDef SPI_InitStruct;
SPI_StructInit(&SPI_InitStruct);           //SPI结构体初始化
SPI_InitStruct.Duplex = SPI_Full;          //双工模式设置
SPI_InitStruct.Mode = SPI_Master;         //master主机模式
SPI_InitStruct.EN = ENABLE;               //使能SPI模块
SPI_InitStruct.DMA = SPI_DMA_DISABLE;     // 选择SPI搬移方式，由MCU搬运数据到SPI
SPI_InitStruct.Trig = 0;                  //内部自动触发传输
SPI_InitStruct.DataOrder = SPI_FIRSTSEND_MSB; //该字节先发送 MSB
SPI_InitStruct.CPHA = 0;                  //第一个沿为发送数据时刻
SPI_InitStruct.CPOL = 0;                  //CLK默认高电平
SPI_InitStruct.BaudRate = 30;             //传输速度96MHZ/(2*(30+1))= 1.548MHZ
SPI_InitStruct.ByteLength = 0;            //发送零个字节
SPI_InitStruct.IRQEna = DISABLE;          //关闭SPI各中断
SPI_Init(SPI, &SPI_InitStruct);          //SPI初始化程序
```

Date	Version	Author	Description
2016年05月21日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_spi.c](#) 第 49 行定义.

函数调用图:



3.71.2.4 SPI_ReadData()

```
uint8_t SPI_ReadData (
    SPI_TypeDef * SPIx )
```

函数名称: uint8_t SPI_ReadData(SPI_TypeDef *SPIx)

功能描述: SPI读缓冲区数据

参见
被引用内容: 无

参数

输入参数:	SPIx: SPI
输出参数:	无

返回
返回值: 无

注解
其它说明: 无

警告
无

示例代码:

```
u8 SPI_RXValue = 0;
SPI_RXValue = SPI_ReadData(SPI); //SPI接收一字节数据
```

Date	Version	Author	Description
2016年05月21日	1.0	cfwu	创建

修改日志:

在文件 lks32mc08x_spi.c 第 150 行定义.

3.71.2.5 SPI_SendData()

```
void SPI_SendData (
    SPI_TypeDef * SPIx,
    uint8_t n )
```

函数名称: void [SPI_SendData\(SPI_TypeDef *SPIx, uint8_t n\)](#)

功能描述: SPI发送数据

参见

被引用内容: 无

参数

输入参数:	SPIx: SPI, n: 发送一字节数据
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
SPI\_SendData(SPI,0x12); //SPI发送0x12一字节数据
```

Date	Version	Author	Description
2016年05月21日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_spi.c](#) 第 125 行定义.

3.71.2.6 SPI_StructInit()

```
void SPI_StructInit (  
    SPI\_InitTypeDef * SPI_InitStruct )
```

函数名称: void [SPI_StructInit\(SPI_InitTypeDef* SPI_InitStruct\)](#)

功能描述: SPI结构体初始化

参见

被引用内容: 无

参数

输入参数:	<code>SPI_InitTypeDef *SPI_InitStruct</code>
输出参数:	无

返回

返回 值: 无

注解

其它说明: 无

警告

无

示例代码:

```
SPI_InitTypeDef SPI_InitStruct;  
SPI_StructInit(&SPI_InitStruct);           //SPI结构体初始化
```

Date	Version	Author	Description
2016年05月21日	1.0	cfwu	创建

修改日志:

在文件 `lks32mc08x_spi.c` 第 86 行定义.

3.72 lks32mc08x_spi.c

浏览该文件的文档.

```
00001  
00015 #include "lks32mc08x_spi.h"  
00016  
00049 void SPI_Init(SPI_TypeDef *SPIx, SPI_InitTypeDef *SPI_InitStruct)  
00050 {  
00051     SYS_WR_PROTECT = 0x7a83; /* 使能系统寄存器写操作*/  
00052     SYS_SFT_RST |= BIT8;      /* 复位SPI  
00053     SYS_SFT_RST &= ~BIT8;     /* 释放SPI  
00054     SYS_WR_PROTECT = 0x0;     /* 关闭系统寄存器写操作*/  
00055     SYS_ModuleClockCmd(SYS_Module_SPI, ENABLE);  
00056  
00057     SPIx->IE = SPI_InitStruct->Trig << 3 | (SPI_InitStruct->IRQEna);  
00058  
00059     SPIx->DIV = SPI_InitStruct->BaudRate | (SPI_InitStruct->DMA << 7);  
00060  
00061     SPIx->SIZE = SPI_InitStruct->ByteLength;  
00062  
00063     SPIx->CFG = SPI_InitStruct->EN | (SPI_InitStruct->DataOrder << 1) | (SPI_InitStruct->CPOL << 2) |  
        (SPI_InitStruct->CPHA << 3) | (SPI_InitStruct->Mode << 4) | (SPI_InitStruct->SS << 5) |  
        (SPI_InitStruct->Duplex << 6);
```

```

00064 }
00065
00086 void SPI_StructInit (SPI_InitTypeDef *SPI_InitStruct)
00087 {
00088     SPI_InitStruct->Duplex      = SPI_Full;
00089     SPI_InitStruct->SS          = 0;
00090     SPI_InitStruct->Mode        = SPI_Slave;
00091     SPI_InitStruct->CPOL        = 0;
00092     SPI_InitStruct->CPHA        = 0;
00093     SPI_InitStruct->DataOrder    = DISABLE;
00094     SPI_InitStruct->EN          = 0;
00095
00096     SPI_InitStruct->IRQEna      = DISABLE;
00097     SPI_InitStruct->Trig        = 0;
00098
00099     SPI_InitStruct->DMA          = 0;
00100     SPI_InitStruct->BaudRate     = 3;
00101
00102     SPI_InitStruct->ByteLength   = 0;
00103 }
00104
00125 void SPI_SendData (SPI_TypeDef *SPIx, uint8_t n)
00126 {
00127     SPIx->TX_DATA = n;
00128 }
00129
00150 uint8_t SPI_ReadData (SPI_TypeDef *SPIx)
00151 {
00152     return SPIx->RX_DATA;
00153 }
00154
00181 uint8_t SPI_GetIRQFlag (SPI_TypeDef *SPIx, u8 IRQ_Flag)
00182 {
00183     u8 SPI_IRQFlag = 0;
00184     if (SPIx->IE & IRQ_Flag)
00185     {
00186         SPI_IRQFlag = 1;
00187     }
00188     else
00189     {
00190         SPI_IRQFlag = 0;
00191     }
00192     return SPI_IRQFlag;
00193 }
00194
00222 void SPI_ClearIRQFlag (SPI_TypeDef *SPIx, u8 IRQ_Flag)
00223 {
00224     SPIx->IE |= IRQ_Flag;
00225 }

```

3.73 source/lks32mc08x_sys.c 文件参考

文件名称: [lks32mc08x_sys.c](#)

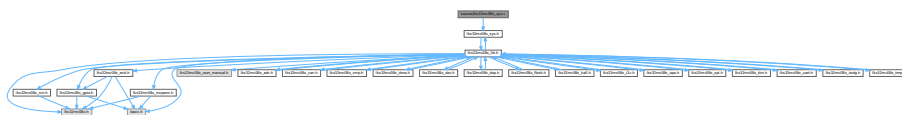
文件标识: 无

内容摘要: SYS外设驱动程序

其它说明: 无

#include "lks32mc08x_sys.h"

lks32mc08x_sys.c 的引用(Include)关系图:



函数

- void [SYS_Init](#) (SYS_InitTypeDef *SYS_InitStruct)

- 函数名称: *void SYS_Init(SYS_InitTypeDef* SYS_InitStruct)*
- void **SYS_StructInit** (SYS_InitTypeDef *SYS_InitStruct)
 - 函数名称: *void SYS_StructInit(SYS_InitTypeDef* SYS_InitStruct)*
- void **SYS_ClearRst** (void)
 - 函数名称: *void SYS_ClearRst(void)*
- uint32_t **SYS_GetRstSource** (void)
 - 函数名称: *uint32_t SYS_GetRstSource(void)*
- void **SYS_FallSleep** (void)
 - 函数名称: *void SYS_FallSleep(void)*
- void **SYS_SetWakeInterval** (uint32_t nInterval)
 - 函数名称: *void SYS_SetWakeInterval(uint32_t nInterval)*
- void **SetWakeIO** (uint32_t wakeIO, enumWakeTrigLevel trigLevel, uint8_t ena)
 - 函数名称: *void SetWakeIO(uint32_t wakeIO, enumWakeTrigLevel trigLevel, uint8_t ena)*
- static void **SoftDelay** (u32 cnt)
 - 函数名称: *static void SoftDelay(u32 cnt)*
- void **Switch2PLL** (void)
 - 函数名称: *void Switch2PLL(void)*
- void **Switch2HRC** (void)
 - 函数名称: *void Switch2HRC(void)*
- void **SYS_WatchDogCmd** (FuncState state)
 - 函数名称: *void SYS_WatchDogCmd(FuncState state)*
- void **SYS_EnablePORFilter** (void)
 - 函数名称: *void SYS_EnablePORFilter(void)*
- void **SYS_FeedDog** (uint32_t timeOut)
 - 函数名称: *void SYS_FeedDog(uint32_t timeOut)*
- void **SYS_ModuleClockCmd** (uint32_t nModule, FuncState state)
 - 函数名称: *void SYS_ModuleClockCmd(uint32_t nModule, FuncState state)*
- void **SYS_AnalogModuleClockCmd** (uint32_t nModule, FuncState state)
 - 函数名称: *void SYS_AnalogModuleClockCmd(uint32_t nModule, FuncState state)*
- void **SYS_SoftResetModule** (uint32_t nModule)
 - 函数名称: *void SYS_SoftResetModule(uint32_t nModule)*

变量

- **WAKE_InitTypeDef WAKE_InitStruct**
 - 低功耗初始化结构体定义

3.73.1 详细描述

文件名称: **lks32mc08x_sys.c**
 文件标识: 无
 内容摘要: **SYS**外设驱动程序
 其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2016年05月21日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_sys.c](#) 中定义.

3.73.2 函数说明

3.73.2.1 SetWakeIO()

```
void SetWakeIO (
    uint32_t wakeIO,
    enumWakeTrigLevel trigLevel,
    uint8_t ena )
```

函数名称: void [SetWakeIO\(uint32_t wakeIO, enumWakeTrigLevel trigLevel, uint8_t ena\)](#)

功能描述: 设置休眠唤醒IO

参见

唤醒电平选择枚举: [enumWakeTrigLevel](#)

参数

输入参数:	wakeIO = WAKEIO_P1_1: P1.1 外部唤醒使能。1: 使能; 0: 禁用
	wakeIO = WAKEIO_P1_0: P1.0 外部唤醒使能。1: 使能; 0: 禁用
	wakeIO = WAKEIO_P0_1: P0.1 外部唤醒使能。1: 使能; 0: 禁用
	wakeIO = WAKEIO_P0_0: P0.0 外部唤醒使能。1: 使能; 0: 禁用
输入参数:	enumWakeTrigLevel :
	唤醒电平选择:
	0:低电平唤醒 1:高电平唤醒
输入参数:	ena:
	ENABLE: 开启IO唤醒 DISABLE: 关闭IO唤醒
输出参数:	无

返回

返回 值: 无

注解

其它说明: 无

警告

无

示例代码:

```
SetWakeIO(WAKEIO_P0_0, WAKETRIG_LOW, ENABLE); // 系统休眠时,使能P0.0作为低功耗唤醒IO.  
低电平唤醒
```

Date	Version	Author	Description
2022年04月14日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_sys.c](#) 第 297 行定义.

3.73.2.2 SoftDelay()

```
static void SoftDelay (  
    u32 cnt ) [static]
```

函数名称: static void [SoftDelay\(u32 cnt\)](#)

功能描述: 延时函数

参见

被调用函数: 无

参数

输入参数:	cnt: 延时cnt的空指令NOP()
输出参数:	无

返回

返回值: 无

注解

其它说明: 改函数只在此文件使用, 主要用在Switch2PLL函数内

警告

无

示例代码:

```
SoftDelay(0xff); //延时255个_NOP() 时间1个_NOP大约按一个系统时钟周期计算
```

Date	Version	Author	Description
2022年04月14日	1.0	HuangMG	创建

修改日志:

在文件 lks32mc08x_sys.c 第 336 行定义.

这是这个函数的调用关系图:



3.73.2.3 Switch2HRC()

```
void Switch2HRC (
    void )
```

函数名称: void Switch2HRC(void)

功能描述: 休眠唤醒前一刻，关闭高速时钟与BGP和掉电检测电路函数

参见

被调用函数: 无

参数

输入参数:	无
输出参数:	无

返回

返回值: 无

注解

其它说明： 系统低功耗休眠使用

警告

无

示例代码:

```
Switch2HRC();
```

Date	Version	Author	Description
2022年04月14日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x-sys.c](#) 第 395 行定义.

3.73.2.4 Switch2PLL()

```
void Switch2PLL (
    void )
```

函数名称: void [Switch2PLL\(void\)](#)

功能描述: 休眠唤醒后, 系统时钟切换到PLL时钟函数

参见

被调用函数: 无

参数

输入参数:	无
输出参数:	无

返回

返回 值: 无

注解

其它说明： 系统低功耗唤醒使用

警告
无

示例代码:

```
Switch2PLL();
```

Date	Version	Author	Description
2022年04月14日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_sys.c](#) 第 364 行定义.

函数调用图:



3.73.2.5 SYS_AnalogModuleClockCmd()

```
void SYS_AnalogModuleClockCmd (
    uint32_t nModule,
    FuncState state )
```

函数名称: void [SYS_AnalogModuleClockCmd\(uint32_t nModule, FuncState state\)](#)

功能描述: 模拟模块使能和停止

参见

被调用枚举: FuncState

参数

输入参数:	state: ENABLE使能, DISABLE关闭 nModule: 参数可选:		
	宏定义	说明	
	SYS_AnalogModule_ADC	BIT0	ADC模块
	SYS_AnalogModule_OPA0	BIT2	OPA0模块
	SYS_AnalogModule_OPA1	BIT3	OPA1模块
	SYS_AnalogModule_OPA2	BIT4	OPA2模块
	SYS_AnalogModule_OPA3	BIT5	OPA3模块
	SYS_AnalogModule_CMP0	BIT6	CMP0模块
	SYS_AnalogModule_CMP1	BIT7	CMP1模块
	SYS_AnalogModule_BGP	BIT8	BGP模块
	SYS_AnalogModule_RCH	BIT10	RCH模块
	SYS_AnalogModule_DAC	BIT12	DAC模块
	SYS_AnalogModule_TMP	BIT13	温度传感器
	SYS_AnalogModule_XTAL	BIT14	晶振
	SYS_AnalogModule_PLL	BIT15	PLL模块
输出参数:	无		

返回

返回 值: 无

注解

其它说明: 无

警告

无

示例代码:

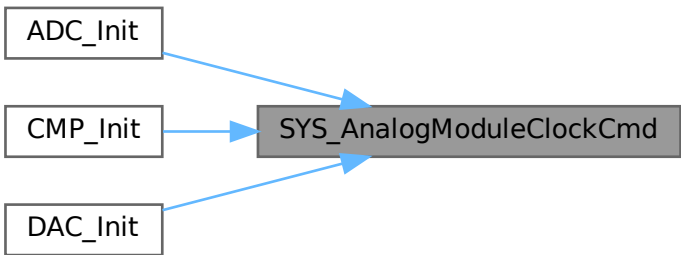
```
SYS_AnalogModuleClockCmd(SYS_AnalogModule_PLL,ENABLE); //使能PLL模块
```

Date	Version	Author	Description
2016年06月28日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_sys.c](#) 第 605 行定义.

这是这个函数的调用关系图:



3.73.2.6 SYS.ClearRst()

```
void SYS_ClearRst (
    void )
```

函数名称: void **SYS.ClearRst(void)**

功能描述: **SYS**清除复位标志记录

参见
被调用函数: 无

参数

输入参数:	无
输出参数:	无

返回
返回 值: 无

注解
其它说明: 请注意由于复位记录工作于低速时钟域，清除执行完成需要一定时间，不应清 除后立
即读记录状态

警告
无

示例代码:

```
u32 RSET_VALUE = 0;
SYS_ClearRst(); //清除复位标志记录
delay_ms(1);
RSET_VALUE = SYS_GetRstSource(); //读取复位源记录值
```

Date	Version	Author	Description
2016年07月02日	1.0	cfwu	创建

修改日志:

在文件 lks32mc08x_sys.c 第 139 行定义.

3.73.2.7 SYS.EnablePORFilter()

```
void SYS_EnablePORFilter (
    void )
```

函数名称: void SYS.EnablePORFilter(void)

功能描述: 允许对POR进行滤波处理

参见

被调用函数: 无

参数

输入参数:	无
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

该函数目前不开放, 请不要使用

示例代码:

```
SYS.EnablePORFilter();
```

Date	Version	Author	Description
2016年06月28日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_sys.c](#) 第 461 行定义.

3.73.2.8 SYS_FallSleep()

```
void SYS_FallSleep (
    void )
```

函数名称: [void SYS_FallSleep\(void\)](#)

功能描述: 触发系统进入休眠状态

参见

被调用函数: 无

参数

输入参数:	无
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
SYS\_FallSleep\(\) ; // 触发系统进入休眠状态
```

Date	Version	Author	Description
2016年07月02日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_sys.c](#) 第 197 行定义.

3.73.2.9 SYS_FeedDog()

```
void SYS_FeedDog (
    uint32_t timeOut )
```

函数名称: void [SYS_FeedDog\(uint32_t timeOut\)](#)

功能描述: 喂狗

参见
被调用函数: 无

参数

输入参数:	timeOut: 看门狗超时时间		
	SYS_WD_TimeOut2s	0	2s复位
	SYS_WD_TimeOut4s	1	4s复位
	SYS_WD_TimeOut8s	2	8s复位
	SYS_WD_TimeOut64s	3	64s复位
输出参数:	无		

返回
返 回 值: 无

注解
其它说明: 无

警告
无

示例代码:
[SYS_FeedDog\(SYS_WD_TimeOut2s\)](#) ; //看门狗喂狗且设置下一次如果2s不喂狗则芯片复位

Date	Version	Author	Description
2016年06月28日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_sys.c](#) 第 495 行定义.

3.73.2.10 SYS_GetRstSource()

```
uint32_t SYS_GetRstSource (
    void )
```

函数名称: uint32_t [SYS_GetRstSource\(void\)](#)

功能描述: 获得SYS复位源信号

参见
被调用函数: 无

参数

输入参数:	无
输出参数:	无

返回
返 回 值: 复位源记录值

注解
其它说明: 返回值位介绍

警告
无

BIT3	看门狗复位发生标志	高有效
BIT2	按键复位发生标志	高有效
BIT1	HPOR 复位发生标志	高有效
BIT0	LPOR 复位发生标志	高有效

示例代码:

```
u32 RSET_VALUE = 0;
RSET_VALUE = SYS\_GetRstSource\(\); // 读取复位源记录值
如 RSET_VALUE = 0x01; 则LPOR发生复位
```

Date	Version	Author	Description
2016年07月02日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_sys.c](#) 第 173 行定义.

3.73.2.11 SYS.Init()

```
void SYS_Init (
    SYS_InitTypeDef * SYS_InitStruct )
```

函数名称: void [SYS.Init\(SYS.InitTypeDef* SYS.InitStruct\)](#)

功能描述: **SYS**模块初始化函数

参见

被调用函数: 无

参数

输入参数:	SYS_InitTypeDef* SYS_InitStruct
输出参数:	无

返回

返回值: 无

注解

其它说明: 一般不使用该函数, 因为涉及模块较多, 比较杂乱, 对应里面的配置在各模块 初始化单独配置。

警告

无

示例代码:

```
SYS_InitTypeDef SYS_InitStruct;
SYS_StructInit(&SYS_InitStruct);
SYS_InitStruct.PLL_SrcSel = SYS_PLLSRSEL_RCH; // 使用内部4MHz晶振作为时钟输入源
SYS_Init(&SYS_InitStruct);
```

Date	Version	Author	Description
2016年07月02日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_sys.c](#) 第 45 行定义.

3.73.2.12 SYS.ModuleClockCmd()

```
void SYS.ModuleClockCmd (
    uint32_t nModule,
    FuncState state )
```

函数名称: void [SYS.ModuleClockCmd\(uint32_t nModule, FuncState state\)](#)

功能描述: 数字模块时钟使能和停止

参见

被调用枚举: FuncState

参数

输入参数:	state: ENABLE使能 , DISABLE关闭 nModule: 参数可选:		
	宏定义	原始值	说明
	SYS_Module_I2C	BIT0	I2C模块
	SYS_Module_HALL	BIT1	HALL模块
	SYS_Module_TIMER	BIT2	UTIMER模块
	SYS_Module_MCPWM	BIT3	MCPWM模块
	SYS_Module_UART0	BIT4	UART0模块
	SYS_Module_UART1	BIT5	UART1模块
	SYS_Module_DSP	BIT6	DSP模块
	SYS_Module_CAN	BIT7	CAN模块
	SYS_Module_SPI	BIT8	SPI模块
输出参数:	无		

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

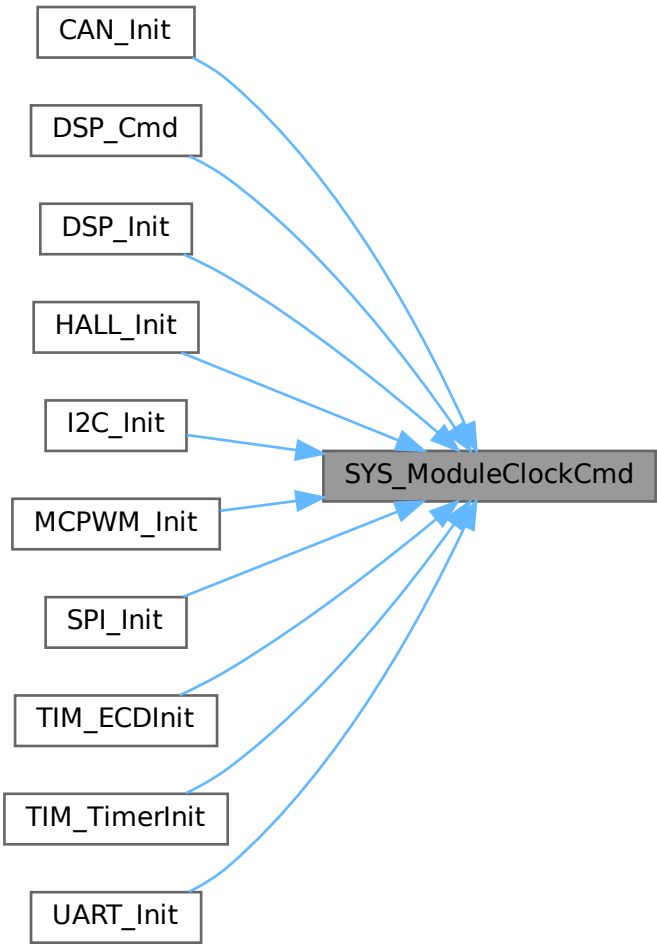
```
SYS_ModuleClockCmd(SYS_Module_SPI, ENABLE); //使能SPI时钟
```

Date	Version	Author	Description
2016年06月28日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x-sys.c](#) 第 552 行定义.

这是这个函数的调用关系图:



3.73.2.13 SYS.SetWakeInterval()

```
void SYS_SetWakeInterval (
    uint32_t nInterval )
```

函数名称: void **SYS.SetWakeInterval**(uint32_t nInterval)

功能描述: 设置系统休眠唤醒周期

参见

被调用函数: 无

参数

输入参数:	nInterval: 唤醒时间		
	宏定义	原始值	说明
	SYS_WakeInterval_025s	0x00	0.25s
	SYS_WakeInterval_05s	0x01	0.5s
	SYS_WakeInterval_1s	0x02	1s
	SYS_WakeInterval_2s	0x03	2s
	SYS_WakeInterval_4s	0x04	4s
	SYS_WakeInterval_8s	0x05	8s
	SYS_WakeInterval_16s	0x06	16s
	SYS_WakeInterval_32s	0x07	32s
输出参数:	无		

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
SYS_SetWakeInterval (SYS_WakeInterval_1s); // 系统休眠时, 设置系统在1s后唤醒
```

Date	Version	Author	Description
2016年07月02日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_sys.c](#) 第 256 行定义.

3.73.2.14 SYS.SoftResetModule()

```
void SYS_SoftResetModule (
    uint32_t nModule )
```

函数名称: void [SYS.SoftResetModule\(uint32_t nModule\)](#)

功能描述: 数字模块软复位

参见
被调用枚举: 无

参数

输入参数:	state: ENABLE使能 , DISABLE关闭 nModule: 参数可选:		
	宏定义	原始值	说明
	SYS_Module_I2C	BIT0	I2C模块
	SYS_Module_HALL	BIT1	HALL模块
	SYS_Module_TIMER	BIT2	UTIMER模块
	SYS_Module_MCPWM	BIT3	MCPWM模块
	SYS_Module_UART0	BIT4	UART0模块
	SYS_Module_UART1	BIT5	UART1模块
	SYS_Module_DSP	BIT6	DSP模块
	SYS_Module_CAN	BIT7	CAN模块
	SYS_Module_SPI	BIT8	SPI模块
输出参数:	无		

返回
返回 值: 无

注解
其它说明: 无

警告
无

示例代码:

```
//软复位SPI模块，清除SPI寄存器配置，PC指针复位
SYS_SoftResetModule(SYS_Module_SPI);
```

Date	Version	Author	Description
2016年06月28日	1.0	cfwu	创建

修改日志:

Date	Version	Author	Description
2022年04月14日	1.1	HuangMG	修改

修改日志:增加延时函数，等待寄存器复位完毕与释放复位操作。

在文件 `lks32mc08x_sys.c` 第 659 行定义.

这是这个函数的调用关系图:



3.73.2.15 SYS_StructInit()

```
void SYS_StructInit (
    SYS_InitTypeDef * SYS_InitStruct )
```

函数名称: void `SYS_StructInit(SYS_InitTypeDef* SYS_InitStruct)`

功能描述: **SYS**结构体初始化

参见

被调用函数: 无

参数

输入参数:	SYS_InitTypeDef* SYS_InitStruct
输出参数:	无

返回

返回 值： 无

注解

其它说明： 无

警告

无

示例代码:

```
SYS_InitTypeDef SYS_InitStruct;  
SYS_StructInit (&SYS_InitStruct);
```

Date	Version	Author	Description
2016年07月02日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_sys.c](#) 第 101 行定义.

3.73.2.16 SYS.WatchDogCmd()

```
void SYS_WatchDogCmd (  
    FuncState state )
```

函数名称: void [SYS.WatchDogCmd\(FuncState state\)](#)

功能描述: 看门狗模块使能函数

参见

被调用函数： 无

参数

输入参数:	state: ENABLE使能看门狗/DISABLE关闭开门狗
输出参数:	无

返回

返回 值： 无

注解

其它说明： 无

警告

无

示例代码:

```
SYS_WatchDogCmd(ENABLE); //使能看门狗模块
```

Date	Version	Author	Description
2016年06月28日	1.0	cfwu	创建

修改日志:

在文件 lks32mc08x_sys.c 第 427 行定义.

3.73.3 变量说明

3.73.3.1 WAKE_InitStruct

```
WAKE_InitTypeDef WAKE_InitStruct
```

低功耗初始化结构体定义

在文件 lks32mc08x_sys.c 第 21 行定义.

3.74 lks32mc08x_sys.c

浏览该文件的文档.

```
00001
00015 #include "lks32mc08x_sys.h"
00016
00021 WAKE_InitTypeDef WAKE_InitStruct;
00045 void SYS_Init(SYS_InitTypeDef *SYS_InitStruct)
00046 {
00047     uint32_t RCHPd, XTALPdn;
00048     uint32_t tmp1, tmp2;
00049     SYS_WR_PROTECT = 0x7a83; /*开启寄存器写使能*/
00050     tmp1 = SYS_AFE_REG6;
00051     tmp1 &= 0x7FFF;
00052     tmp1 |= (SYS_InitStruct->PLL_SrcSel << 15);
00053     SYS_AFE_REG6 = tmp1;
00054
00055     if (SYS_InitStruct->PLL_SrcSel == SYS_PLLSRSEL_CRYSTAL)
00056     {
00057         RCHPd = ENABLE; /*RCH(4MHz)时钟关闭*/
00058         XTALPdn = ENABLE; /*晶体起振电路开启*/
00059     }
00060     else
00061     {
00062         RCHPd = DISABLE; /*RCH(4MHz)时钟不关闭*/
```

```

00063     XTALPdn = DISABLE; /*晶体起振电路不开启*/
00064 }
00065
00066 tmp2 = SYS_AFE_REG5;
00067 tmp2 &= 0xBBFF;
00068 tmp2 |= (RCHPd << 10);
00069 tmp2 |= (XTALPdn << 14);
00070 SYS_AFE_REG5 = tmp2;
00071
00072 SYS_CLK_CFG = SYS_InitStruct->PLL_DivSel + (SYS_InitStruct->Clk_Sel << 8);
00073 SYS_RST_CFG = SYS_InitStruct->WDT_Ena | (SYS_InitStruct->PORFilter_Ena << 1);
00074
00075 SYS_CLK_DIV0 = SYS_InitStruct->Clk_DivSPI;
00076 SYS_CLK_DIV2 = SYS_InitStruct->Clk_DivUART;
00077 SYS_CLK_FEN = SYS_InitStruct->Clk_FEN;
00078 SYS_WR_PROTECT = 0x0;
00079 }
00080
00101 void SYS_StructInit(SYS_InitTypeDef *SYS_InitStruct)
00102 {
00103     SYS_InitStruct->PLL_SrcSel = SYS_PLLSRSEL_RCH;
00104     SYS_InitStruct->PLL_DivSel = 0xFF;
00105     SYS_InitStruct->Clk_Sel = SYS_CLK_SEL_PLL;
00106     SYS_InitStruct->PLL_ReDiv = 0;
00107
00108     SYS_InitStruct->Clk_DivSPI = SYS_Clk_SPIDiv1;
00109     SYS_InitStruct->Clk_DivUART = SYS_Clk_UARTDiv1;
00110     SYS_InitStruct->Clk_FEN = 0;
00111
00112     SYS_InitStruct->WDT_Ena = DISABLE;
00113     SYS_InitStruct->PORFilter_Ena = DISABLE;
00114 }
00115
00139 void SYS_ClearRst(void)
00140 {
00141     SYS_WR_PROTECT = 0x7a83; /*开启寄存器写使能*/
00142     SYS_CLR_RST = 0xDEAD;
00143     SYS_WR_PROTECT = 0x0;
00144 }
00145
00173 uint32_t SYS_GetRstSource(void)
00174 {
00175     return SYS_RST_SRC;
00176 }
00177
00197 void SYS_Fallsleep(void)
00198 {
00199     NVIC_DisableIRQ(TIMER0_IRQn); /*关闭TIMER0中断*/
00200     NVIC_DisableIRQ(TIMER1_IRQn); /*关闭TIMER1中断*/
00201     NVIC_DisableIRQ(TIMER2_IRQn); /*关闭TIMER2中断*/
00202     NVIC_DisableIRQ(TIMER3_IRQn); /*关闭TIMER3中断*/
00203     NVIC_DisableIRQ(ENCODER0_IRQn); /*关闭ENCODER0_IRQn中断*/
00204     NVIC_DisableIRQ(ENCODER1_IRQn); /*关闭ENCODER1_IRQn中断*/
00205     NVIC_DisableIRQ(I2C0_IRQn); /*关闭I2C0中断*/
00206     NVIC_DisableIRQ(SPI0_IRQn); /*关闭SPI0中断*/
00207     NVIC_DisableIRQ(GPIO_IRQn); /*关闭GPIO中断*/
00208     NVIC_DisableIRQ(HALL_IRQn); /*关闭HALL中断*/
00209     NVIC_DisableIRQ(UART0_IRQn); /*关闭UART0中断*/
00210     NVIC_DisableIRQ(UART1_IRQn); /*关闭UART1中断*/
00211     NVIC_DisableIRQ(DSP_IRQn); /*关闭DSP中断*/
00212     NVIC_DisableIRQ(CMP_IRQn); /*关闭CMP中断*/
00213     NVIC_DisableIRQ(ADC0_IRQn); /*关闭ADC0中断*/
00214     NVIC_DisableIRQ(MCPWM_IRQn); /*关闭MCPWM中断*/
00215     NVIC_DisableIRQ(DMA_IRQn); /*关闭DMA中断*/
00216     NVIC_DisableIRQ(CAN_IRQn); /*关闭CAN中断*/
00217     NVIC_DisableIRQ(SIF_IRQn); /*关闭SIF中断*/
00218     NVIC_DisableIRQ(PWRDN_IRQn); /*关闭PWRDN中断*/
00219
00220     SYS_WR_PROTECT = 0x7a83; /*开启寄存器写使能*/
00221     SYS_CLK_SLP = 0xDEAD;
00222     SYS_WR_PROTECT = 0x0;
00223     _WFI(); /*PC指针停止运行，等待中断*/
00224 }
00225
00256 void SYS_SetWakeInterval(uint32_t nInterval)
00257 {
00258     uint32_t tempReg = SYS_RST_CFG;
00259     SYS_WR_PROTECT = 0x7a83; /*开启寄存器写使能*/
00260     tempReg |= (nInterval << 2);
00261     SYS_RST_CFG = tempReg;
00262     SYS_WR_PROTECT = 0x0;
00263 }
00264
00297 void SetWakeIO(uint32_t wakeIO, enumWakeTrigLevel trigLevel, uint8_t ena)
00298 {
00299     if (ena == ENABLE)
00300     {

```

```

00301     GPIO_WAKE_EN |= wakeIO;
00302 }
00303 else
00304 {
00305     GPIO_WAKE_EN &= ~wakeIO;
00306 }
00307
00308 if (trigLevel == WAKETRIG_HIGH)
00309 {
00310     GPIO_WAKE_POL |= wakeIO;
00311 }
00312 else
00313 {
00314     GPIO_WAKE_POL &= ~wakeIO;
00315 }
00316 }
00336 static void SoftDelay(u32 cnt)
00337 {
00338     volatile u32 t_cnt;
00339     for (t_cnt = 0; t_cnt < cnt; t_cnt++)
00340     {
00341         __NOP();
00342     }
00343 }
00344
00364 void Switch2PLL(void)
00365 {
00366     SYS_WR_PROTECT = 0x7a83; /*开启寄存器写使能*/
00367     SYS_AFE_REG5 = WAKE_InitStruct.AFE_REG5_RECORD; /*开启 PLL, HRC, BGP*/
00368     SoftDelay(200);
00369     SYS_AFE_REG6 = WAKE_InitStruct.AFE_REG6_RECORD; /*开启掉电检测电路*/
00370     NVIC_ClearPendingIRQ(PWRDN_IRQn); /*清除电源检测中断标志*/
00371     NVIC_EnableIRQ(PWRDN_IRQn); /*打开电源检测中断*/
00372     SYS_CLK_CFG = WAKE_InitStruct.CLK_CFG_RECORD; /*将主时钟切换到PLL时钟*/
00373     SYS_WR_PROTECT = 0x0;
00374 }
00375
00395 void Switch2HRC(void)
00396 {
00397     SYS_WR_PROTECT = 0x7a83; /*开启寄存器写使能*/
00398     WAKE_InitStruct.AFE_REG5_RECORD = SYS_AFE_REG5;
00399     WAKE_InitStruct.AFE_REG6_RECORD = SYS_AFE_REG6;
00400     WAKE_InitStruct.CLK_CFG_RECORD = SYS_CLK_CFG;
00401     SYS_CLK_CFG = 0x101; /*使用PLL 1/8分频时钟*/
00402     SYS_AFE_REG5 = 0x0500; /*准备关闭PLL, HRC, BGP*/
00403     NVIC_DisableIRQ(PWRDN_IRQn); /*关闭电源检测中断*/
00404     SYS_AFE_REG6 |= 1; /*关闭掉电检测电路*/
00405     SYS_WR_PROTECT = 0x0;
00406 }
00407
00427 void SYS_WatchDogCmd(FuncState state)
00428 {
00429     SYS_WR_PROTECT = 0x7a83; //解除写保护
00430
00431     if (state == ENABLE)
00432     {
00433         SYS_RST_CFG |= BIT0; //使能看门狗
00434     }
00435     else
00436     {
00437         SYS_RST_CFG &= (~BIT0); //关闭看门狗
00438     }
00439     SYS_WR_PROTECT = 0x0;
00440 }
00441
00461 void SYS_EnablePORFilter(void)
00462 {
00463     uint32_t tempReg = SYS_RST_CFG;
00464     SYS_WR_PROTECT = 0x7a83; //解除系统寄存器写保护
00465     tempReg |= BIT1;
00466     SYS_RST_CFG = tempReg;
00467     SYS_WR_PROTECT = 0x0; /*关闭系统寄存器写操作*/
00468 }
00469
00495 void SYS_FeedDog(uint32_t timeOut)
00496 {
00497     SYS_WR_PROTECT = 0xcafe; //0xcafe;
00498
00499     if (timeOut == (uint32_t)SYS_WD_TimeOut64s)
00500     {
00501         SYS_WDT_CLR = 0x7989; //1 00 1
00502     }
00503     else if (timeOut == (uint32_t)SYS_WD_TimeOut8s)
00504     {
00505         SYS_WDT_CLR = 0x798b; //1 01 1
00506     }
00507     else if (timeOut == (uint32_t)SYS_WD_TimeOut4s)

```

```

00508     {
00509         SYS_WDT_CLR = 0x798D; // 1 10 1
00510     }
00511     else // SYS_WD_TimeOut2s
00512     {
00513         SYS_WDT_CLR = 0x798F; // 1 11 1
00514     }
00515
00516     SYS_WR_PROTECT = 0;
00517 }
00518
00552 void SYS_ModuleClockCmd(uint32_t nModule, FuncState state)
00553 {
00554     SYS_WR_PROTECT = 0x7A83; // 解除系统寄存器写保护
00555
00556     if (state != DISABLE)
00557     {
00558         SYS_CLK_FEN |= nModule;
00559     }
00560     else
00561     {
00562         SYS_CLK_FEN &= ~nModule;
00563     }
00564
00565     SYS_WR_PROTECT = 0x0; /*关闭系统寄存器写操作*/
00566 }
00567
00605 void SYS_AnalogModuleClockCmd(uint32_t nModule, FuncState state)
00606 {
00607     SYS_WR_PROTECT = 0x7A83; // 解除系统寄存器写保护
00608
00609     if (state != DISABLE)
00610     {
00611         SYS_AFE_REG5 |= nModule;
00612     }
00613     else
00614     {
00615         SYS_AFE_REG5 &= ~nModule;
00616     }
00617     SYS_WR_PROTECT = 0x0; /*关闭系统寄存器写操作*/
00618 }
00619
00659 void SYS_SoftResetModule(uint32_t nModule)
00660 {
00661     SYS_WR_PROTECT = 0x7A83;
00662     SYS_SFT_RST = nModule; /*模块复位*/
00663     SYS_SFT_RST &= ~nModule; /*释放复位操作*/
00664     SYS_WR_PROTECT = 0x0; /*关闭系统寄存器写操作*/
00665 }
00666
00685 void SYS_VolSelModule(uint32_t Vol)
00686 {
00687     if (Vol == MCU_POWER_5v0) // 5.0 Voltage
00688     {
00689         SYS_WR_PROTECT = 0x7A83; /* 解除系统寄存器写保护 */
00690         SYS_AFE_REG6 &= ~(BIT8 | BIT9); // main init set to 3.6v, this clause reset to 4.5v
00691         SoftDelay(100); /* 延时100us*/
00692         SYS_AFE_REG6 |= POWER_4V2 << 8; // 配置为4.2v监控, 可根据实际配置其它档位: 3.6V/3.9V/4.2V/4.5V
00693         SYS_AFE_REG6 &= ~1; /* 打开掉电检测电路*/
00694         SoftDelay(100); /* 延时100us, 等待电源检测配置完毕*/
00695         // while (SYS_AFE_CMP & BIT13) // 等待外部5VOK
00696         // {
00697         // }
00698         SYS_AFE_REG6 &= ~(BIT8 | BIT9); // reset to 4.5V
00699         SoftDelay(100); /* 延时100us*/
00700         SYS_AFE_REG6 |= POWER_3V6 << 8; // 配置为3.6v, 防止电源和欠压比较的抖动
00701         SoftDelay(100); /* 延时100us*/
00702         NVIC_ClearPendingIRQ(PWRDN_IRQn);
00703         SYS_WR_PROTECT = 0x0; /*关闭系统寄存器写操作*/
00704     }
00705 }
00706
00725 void SYS_VolSelModuleIRQ(void)
00726 {
00727     NVIC_DisableIRQ(PWRDN_IRQn); /* 关闭电源检测中断 */
00728     NVIC_ClearPendingIRQ(PWRDN_IRQn);
00729     // turn off pwm
00730     MCPWM_PRT = 0xDEAD;
00731     MCPWM_FAIL &= ~BIT6;
00732     MCPWM_TCLK &= ~0x0000000C;
00733     MCPWM_PRT = 0x0000;
00734     SYS_WR_PROTECT = 0x7A83; /* 解除系统寄存器写保护 */
00735     SYS_SFT_RST = 0xff; /* 所有外设复位 */
00736     SYS_CLK_CFG &= ~BIT8; /* 系统时钟选择RCH */
00737     SYS_AFE_REG5 &= ~BIT15; /* 关闭PLL */
00738     SYS_SFT_RST = 0x0; /* 释放外设复位 */
00739     SYS_WR_PROTECT = 0x0; /*关闭系统寄存器写操作*/

```

```

00740     NVIC_SystemReset();          // 软复位
00741 }
00742
00761 void SYS_VolSelModuleEnableIRQ(uint32_t Vol)
00762 {
00763     if (Vol == MCU_POWER_5v0)
00764     {
00765         NVIC_SetPriority(PWRDN_IRQn, 0); /*PWRDN_IRQn中断优先级设置为0*/
00766         NVIC_EnableIRQ(PWRDN_IRQn);      /* 打开电源检测中断 */
00767     }
00768     else
00769     {
00770
00771         NVIC_DisableIRQ(PWRDN_IRQn); /* 关闭电源检测中断 */
00772     }
00773 }
00774
00784 void SYS_ResetPeripheral(void)
00785 {
00786     SYS_ResetIO();
00787     SYS_ResetPeripheralInternal();
00788 }
00798 void SYS_ResetPeripheralInternal(void)
00799 {
00800     uint32_t flg = __get_PRIMASK();
00801     __disable_irq();
00802
00803     SYS_WR_PROTECT = 0x7a83;
00804     SYS_SFT_RST = 0x1fff;
00805     SYS_SFT_RST = 0;
00806
00807     SYS_AFE_REG0 = 0;
00808     SYS_AFE_REG1 = 0;
00809     SYS_AFE_REG2 = 0;
00810     SYS_AFE_REG3 = 0;
00811     SYS_AFE_REG4 = 0;
00812     SYS_AFE_REG5 = 0;
00813     SYS_AFE_REG6 = 0;
00814     SYS_AFE_REG7 = 0;
00815     SYS_AFE_DAC = 0;
00816     SYS_CLK_CFG = 0;
00817     SYS_RST_CFG = 0;
00818     // SYS_RST_SRC 是只读寄存器
00819     // SYS_RST_SRC 记录了复位来源, 这里不做配置
00820     SYS_CLK_DIV0 = 0;
00821     SYS_CLK_DIV2 = 0;
00822     SYS_CLK_FEN = 0;
00823     // SYS_RST_SRC 写入后会进入休眠状态, 这里不做配置
00824     SYS_WR_PROTECT = 0;
00825
00826     FLASH_CFG = 0x60; // FLASH配置寄存器, 默认值0x60
00827     FLASH_ADDR = 0;   // 地址寄存器
00828     // FLASH_WDATA 写数据寄存器 为了避免对flash的误操作, 这里不对FLASH_WDATA做任何配置
00829     // FLASH_RDATA 读数据寄存器 只读
00830     // FLASH_ERASE 擦除使能寄存器 写入后触发擦除动作, 这里不做配置
00831     // FLASH_READY FLASH闲忙状态寄存器 只读
00832     DMA_CCR0 = 0;     // DMA通道0通道配置寄存器
00833     DMA_CTMS0 = 0;    // DMA通道0传输次数寄存器
00834     DMA_CPAR0 = 0;    // DMA通道0外设地址寄存器
00835     DMA_CMAR0 = 0;    // DMA通道0内存地址寄存器
00836     DMA_CCR1 = 0;     // DMA通道1通道配置寄存器
00837     DMA_CTMS1 = 0;    // DMA通道1传输次数寄存器
00838     DMA_CPAR1 = 0;    // DMA通道1外设地址寄存器
00839     DMA_CMAR1 = 0;    // DMA通道1内存地址寄存器
00840     DMA_CCR2 = 0;     // DMA通道2通道配置寄存器
00841     DMA_CTMS2 = 0;    // DMA通道2传输次数寄存器
00842     DMA_CPAR2 = 0;    // DMA通道2外设地址寄存器
00843     DMA_CMAR2 = 0;    // DMA通道2内存地址寄存器
00844     DMA_CCR3 = 0;     // DMA通道3通道配置寄存器
00845     DMA_CTMS3 = 0;    // DMA通道3传输次数寄存器
00846     DMA_CPAR3 = 0;    // DMA通道3外设地址寄存器
00847     DMA_CMAR3 = 0;    // DMA通道3内存地址寄存器
00848     DMA_CTRL = 0;     // DMA控制寄存器
00849     DMA_IF = 0xffff;  // DMA中断标志寄存器 写1清零
00850
00851     CRC_DR = 0;
00852     CRC_CR = 0;
00853     CRC_INIT = 0;
00854     CRC_POL = 0;
00855
00856     ADC0_TRIG = 0;    // 先关闭adc硬件触发使能后再进行复位
00857     ADC0_CFG = BIT2;  // adc状态机复位
00858     // adc的寄存器比较多, 这里用for配置部分寄存器
00859
00860     {
00861         uint32_t addr;
00862         for (addr = (uint32_t)(&ADC0_DAT0); addr <= (uint32_t)(&ADC0_CHNT1); addr += 4)

```



```

00863     {
00864         REG32(addr) = 0;
00865     }
00866 }
00867 ADC0_GAIN0 = 0;
00868 ADC0_GAIN1 = 0;
00869 // dc amc是校准参数, 这里不做配置
00870 ADC0_IE = 0;
00871 ADC0_IF = 0x7f; // 写1清零
00872 ADC0_DAT0_TH = 0;
00873 CMP_IE = 0;
00874 CMP_IF = 3;
00875 CMP_TCLK = 0;
00876 CMP_CFG = 0;
00877 CMP_BLCWIN = 0;
00878
00879 SIF_CFG = 0;
00880 SIF_FREQ = 0;
00881 SIF_IRQ = 0;
00882 // SIF_WDATA 写入操作将直接触发sif发送数据, 这里不做配置
00883
00884 SYS_WR_PROTECT = 0xCAFE;
00885 SYS_WDT_CLR = 0;
00886 SYS_WR_PROTECT = 0;
00887 __set_PRIMASK(flq);
00888 }
00892 void SYS_ResetIO(void)
00893 {
00894     uint32_t flq = __get_PRIMASK();
00895     __disable_irq();
00896     // 先关闭上拉, 再关闭开漏使能, 最后关闭输出使能
00897     LCKR_PRT = 0x5ac4;
00898     GPIO0->LCKR = 0;
00899     GPIO0->PUE = 0;
00900     GPIO0->PODE = 0;
00901     GPIO0->POE = 0;
00902     GPIO1->LCKR = 0;
00903     GPIO1->PUE = 0;
00904     GPIO1->PODE = 0;
00905     GPIO1->POE = 0;
00906     GPIO2->LCKR = 0;
00907     GPIO2->PUE = 0;
00908     GPIO2->PODE = 0;
00909     GPIO2->POE = 0;
00910     GPIO3->PUE = 0;
00911     GPIO3->PODE = 0;
00912     GPIO3->POE = 0;
00913     {
00914         uint32_t i;
00915         for (i = 0x40012000; i < 0x40012100; i += 4)
00916         {
00917             REG32(i) = 0;
00918         }
00919     }
00920     EXTI->CR0 = 0;
00921     EXTI->CR1 = 0;
00922     EXTI->IF = 0xfffff;
00923
00924     WAKE_EN = 0;
00925     WAKE_POL = 0;
00926
00927     SYS_WR_PROTECT = 0x7a83;
00928     SYS_RST_CFG &= ~BITS;
00929     SYS_WR_PROTECT = 0;
00930     __set_PRIMASK(flq);
00931 }
00932 /***** (C) COPYRIGHT LINKO SEMICONDUCTOR *****END OF FILE*****/

```

3.75 source/lks32mc08x_tim.c 文件参考

文件名称: [lks32mc08x_tim.c](#)

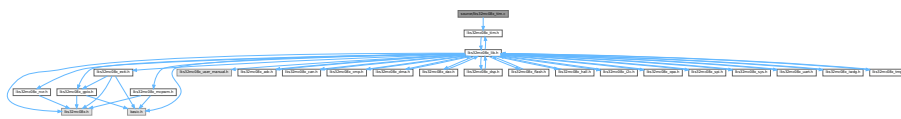
文件标识: 无

内容摘要: 定时器外设驱动程序

其它说明: 无

```
#include "lks32mc08x_tim.h"
```

lks32mc08x_tim.c 的引用(Include)关系图:



函数

- void **TIM_TimerInit** (TIM_TimerTypeDef *TIMERx, TIM_TimerInitTypeDef *TIM_TimerInitStruct)
函数名称: void *TIM_TimerInit(TIM_TimerTypeDef *TIMERx, TIM_TimerInitTypeDef *TIM_TimerInitStruct)*
- void **TIM_TimerStrutInit** (TIM_TimerInitTypeDef *TIM_TimerInitStruct)
函数名称: void *TIM_TimerStrutInit(TIM_TimerInitTypeDef *TIM_TimerInitStruct)*
- void **TIM_TimerCmd** (TIM_TimerTypeDef *TIMERx, FuncState state)
函数名称: void *TIM_TimerCmd(TIM_TimerTypeDef *TIMERx, FuncState state)*
- uint32_t **TIM_GetIRQFlag** (TIM_TimerTypeDef *TIMERx, u32 timer_if)
函数名称: uint32_t *TIM_GetIRQFlag(TIM_TimerTypeDef *TIMERx, u32 timer_if)*
- void **TIM_ClearIRQFlag** (TIM_TimerTypeDef *TIMERx, uint32_t tempFlag)
函数名称: void *TIM_ClearIRQFlag(TIM_TimerTypeDef *TIMERx, uint32_t tempFlag)*
- uint32_t **TIM_Timer_GetCount** (TIM_TimerTypeDef *TIMERx)
函数名称: uint32_t *TIM_Timer_GetCount(TIM_TimerTypeDef *TIMERx)*
- uint32_t **TIM_Timer_GetCMPT0** (TIM_TimerTypeDef *TIM_TIMERx)
函数名称: uint32_t *TIM_Timer_GetCMPT0(TIM_TimerTypeDef *TIM_TIMERx)*
- uint32_t **TIM_Timer_GetCMPT1** (TIM_TimerTypeDef *TIM_TIMERx)
函数名称: uint32_t *TIM_Timer_GetCMPT1(TIM_TimerTypeDef *TIM_TIMERx)*
- void **TIM_ECDInit** (TIM_ECDDTypeDef *ECDx, TIM_ECDInitTypeDef *TIM_ECDInitStruct)
函数名称: void *TIM_ECDInit(TIM_ECDDTypeDef *ECDx, TIM_ECDInitTypeDef *TIM_ECDInitStruct)*
- void **TIM_ECDStructInit** (TIM_ECDInitTypeDef *TIM_ECDInitStruct)
函数名称: void *TIM_ECDInit(TIM_ECDDTypeDef *ECDx, TIM_ECDInitTypeDef *TIM_ECDInitStruct)*
- void **TIM_ECDCmd** (TIM_ECDDTypeDef *ECDx, FuncState state)
函数名称: void *TIM_ECDCmd(TIM_ECDDTypeDef *ECDx, FuncState state)*
- uint32_t **TIM_ECD_GetCount** (TIM_ECDDTypeDef *TIM_ECDx)
函数名称: uint32_t *TIM_ECD_GetCount(TIM_ECDDTypeDef *TIM_ECDx)*
- uint32_t **ECD_GetIRQFlag** (TIM_ECDDTypeDef *ECDx, u32 timer_if)
函数名称: uint32_t *ECD_GetIRQFlag(TIM_ECDDTypeDef *ECDx, u32 timer_if)*
- void **ECD_ClearIRQFlag** (TIM_ECDDTypeDef *ECDx, uint32_t tempFlag)
函数名称: void *ECD_ClearIRQFlag(TIM_ECDDTypeDef *ECDx, uint32_t tempFlag)*

3.75.1 详细描述

文件名称: lks32mc08x_tim.c

文件标识: 无

内容摘要: 定时器外设驱动程序

其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2016年06月06日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x.tim.c](#) 中定义.

3.75.2 函数说明

3.75.2.1 ECD_ClearIRQFlag()

```
void ECD_ClearIRQFlag (
    TIM_ECDTypeDef * ECDx,
    uint32_t tempFlag )
```

函数名称: void [ECD_ClearIRQFlag\(TIM_ECDTypeDef *ECDx , uint32_t tempFlag\)](#)

功能描述: 清除ECD中断标志

参见

被引用内容: 无

参数

输入参数:	ECDx: ECD0/ECD1	
	timer_if:	
	ECD_IF_UNDERFLOW ECD_IF_OVERFLOW	Encoder下溢中断 Encoder上溢中断
输出参数:	无	

返回

返回 值: 无

注解

其它说明: 无

警告

无

示例代码:

```
if(ECD_GetIRQFlag(ECD0,ECD_IF_OVERFLOW)) //获取编码器上溢中断标志
{
    ECD_ClearIRQFlag(ECD0,ECD_IF_OVERFLOW); //清除编码器上溢中断标志
}
```

Date	Version	Author	Description
2022年04月13日	1.0	HuangMG	修改

修改日志:

在文件 lks32mc08x_tim.c 第 711 行定义.

3.75.2.2 ECD_GetIRQFlag()

```
uint32_t ECD_GetIRQFlag (
    TIM_ECDTypeDef * ECDx,
    u32 timer_if )
```

函数名称: uint32_t ECD_GetIRQFlag(TIM_ECDTypeDef *ECDx , u32 timer_if)

功能描述: 获取ECD中断标志

参见
被引用内容: 无

参数

输入参数:	ECDx: ECD0/ECD1	
	timer_if:	
	ECD_IF_UNDERFLOW ECD_IF_OVERFLOW	Encoder下溢中断 Encoder上溢中断
输出参数:	无	

返回
返回 值: 无

注解
其它说明: 无

警告
只有对应中断使能后，改为才能读取，如果对应中断未使能，读取结果一直为0

示例代码:

```
if(ECD.GetIRQFlag(ECD0,ECD_IF_OVERFLOW)) //获取上溢中断标志
{
}
```

Date	Version	Author	Description
2022年04月13日	1.0	HuangMG	修改

修改日志:

在文件 [lks32mc08x_tim.c](#) 第 645 行定义.

3.75.2.3 TIM_ClearIRQFlag()

```
void TIM_ClearIRQFlag (
    TIM_TimerTypeDef * TIMeRx,
    uint32_t tempFlag )
```

函数名称: void [TIM_ClearIRQFlag\(TIM_TimerTypeDef *TIMeRx ,uint32_t tempFlag\)](#)

功能描述: 清除TIM中断标志

参见

被引用内容: 无

参数

输入参数:	TIMERx: TIMER0/TIMER1/TIMER2/TIMER3		
	timer_if:		
	TIMER_IF_CH1	Timer CH1中断, 比较或捕获中断	
	TIMER_IF_CH0	Timer CH0中断, 比较或捕获中断	
	TIMER_IF_ZERO	Timer 过零中断	
输出参数:	无		

返回

返回值: 无

注解

其它说明: 无

警告
无

示例代码:

```
if(TIM_GetIRQFlag(TIMERO,TIMER_IF_CH0)) //判断UTimer0的CH0是否发生比较中断
{
    TIM_ClearIRQFlag(TIMERO,TIMER_IF_CH0); //清除UTimer0通道0比较中断标志位
}
```

Date	Version	Author	Description
2022年04月13日	1.0	HuangMG	修改

修改日志:

在文件 [lks32mc08x_tim.c](#) 第 334 行定义.

3.75.2.4 TIM_ECD_GetCount()

```
uint32_t TIM_ECD_GetCount (
    TIM_ECDTypeDef * TIM_ECDx )
```

函数名称: `uint32_t TIM_ECD_GetCount(TIM_ECDTypeDef *TIM_ECDx)`

功能描述: 编码器计数值

参见
被引用内容: 无

参数

输入参数:	ECDx: ECD0/ECD1
输出参数:	无

返回
返回值: 无

注解
其它说明: 无

警告

无

示例代码:

```
uint32_t ECD0_Value = 0;
ECD0_Value = TIM_ECD_GetCount (ECD0); //获取编码器0计数值
```

Date	Version	Author	Description
2016年06月06日	1.0	cfwu	修改

修改日志:

在文件 [lks32mc08x_tim.c](#) 第 615 行定义.

3.75.2.5 TIM_ECDCmd()

```
void TIM_ECDCmd (
    TIM_ECDTypeDef * ECDx,
    FuncState state )
```

函数名称: void [TIM_ECDCmd\(TIM_ECDTypeDef *ECDx, FuncState state\)](#)

功能描述: 编码器ECDx使能和停止

参见

被引用内容: 无

参数

输入参数:	ECDx: ECD0/ECD1 FuncState state: 使能与失能
输出参数:	无

返回

返回 值: 无

注解

其它说明: 无

警告

无

示例代码:

```
TIM_ECDCmd(ECD0, ENABLE); // ECD0模块使能
```

Date	Version	Author	Description
2022年04月13日	1.0	HuangMG	修改

修改日志:

在文件 [lks32mc08x_tim.c](#) 第 565 行定义.

3.75.2.6 TIM_ECDInit()

```
void TIM_ECDInit (
    TIM_ECDTypeDef * ECDx,
    TIM_ECDInitTypeDef * TIM_ECDInitStruct )
```

函数名称: void [TIM_ECDInit\(TIM_ECDTypeDef *ECDx, TIM_ECDInitTypeDef *TIM_ECDInitStruct\)](#)

功能描述: 编码器初始化函数

参见

被引用内容: [SYS_ModuleClockCmd\(\)](#);

参数

输入参数:	ECDx: ECD0/ECD1 TIM_ECDInitTypeDef *TIM_ECDInitStruct
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
TIM_ECDInitTypeDef TIM_ECDInitStruct;  
TIM_ECDStructInit(&TIM_ECDInitStruct);  
TIM_ECDInitStruct.ECD_ClockDiv = ECD_Clk_Div1;           //96M  
TIM_ECDInitStruct.ECD_Filter0 = 10;                      //CH0滤波, 滤波宽度为Timer_Filter0*8  
                                                         个时钟周期 ECD_Filter0~15  
  
TIM_ECDInitStruct.ECD_Filter1 = 10;                      //CH1滤波  
TIM_ECDInitStruct.ECD_Mode = ECD_Mode_CCWSIGN;          //CCW+SIGN, 仅计数信号上升沿  
TIM_ECDInitStruct.ECD_TH = 4096;                        //计数门限  
TIM_ECDInitStruct.ECD_IRQEna = ECD_IRQEna_OverFlow;    //使能上溢中断  
TIM_ECDInit(ECD0, &TIM_ECDInitStruct);  
TIM_ECDCmd(ECD0, ENABLE);                               //使能ECD0模块
```

Date	Version	Author	Description
2016年06月06日	1.0	cfwu	修改

修改日志:

Date	Version	Author	Description
2022年04月13日	1.0	HuangMG	修改

修改日志:增加定时器时钟使能函数

在文件 [lks32mc08x_tim.c](#) 第 464 行定义.

函数调用图:



3.75.2.7 TIM_ECDStructInit()

```
void TIM_ECDStructInit (   
    TIM_ECDInitTypeDef * TIM_ECDInitStruct )
```

函数名称: void [TIM_ECDInit](#)([TIM_ECDTypeDef](#) *[ECDx](#), [TIM_ECDInitTypeDef](#) *[TIM_ECDInitStruct](#))

功能描述: 编码器结构体初始化

参见

被引用内容: 无

参数

输入参数:	TIM_ECDInitTypeDef *TIM.ECDInitStruct
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
TIM\_ECDInitTypeDef TIM_ECDInitStruct;  
TIM\_ECDStructInit (&TIM_ECDInitStruct); //编码器结构体初始化
```

Date	Version	Author	Description
2016年06月06日	1.0	cfwu	修改

修改日志:

在文件 [lks32mc08x_tim.c](#) 第 535 行定义.

3.75.2.8 TIM_GetIRQFlag()

```
uint32_t TIM_GetIRQFlag (  
    TIM\_TimerTypeDef * TIMERx,  
    u32 timer_if )
```

函数名称: [uint32_t TIM_GetIRQFlag\(TIM_TimerTypeDef *TIMERx , u32 timer_if\)](#)

功能描述: 获取TIM中断标志

参见

被引用内容: 无

参数

输入参数:	TIMERx: TIMER0/TIMER1/TIMER2/TIMER3		
	timer_if:		
	TIMER_IF_CH1	Timer CH1中断, 比较或捕获中断	
	TIMER_IF_CH0	Timer CH0中断, 比较或捕获中断	
	TIMER_IF_ZERO	Timer 过零中断	
输出参数:	无		

返回

返回 值: 0或1, 对应中断标志置位返回1, 未置位返回0

注解

其它说明: 无

警告

只有对应中断使能后, 改为才能读取, 如果对应中断未使能, 读取结果一直为0

示例代码:

```
if(TIM_GetIRQFlag(TIMER0,TIMER_IF_CH0)) //判断UTimer0的CH0是否发生比较中断
{
}
```

Date	Version	Author	Description
2022年04月13日	1.0	HuangMG	修改

修改日志:

在文件 [lks32mc08x_tim.c](#) 第 255 行定义.

3.75.2.9 TIM_Timer_GetCMPT0()

```
uint32_t TIM_Timer_GetCMPT0 (
    TIM_TimerTypeDef * TIM_TIMERx )
```

函数名称: uint32_t [TIM_Timer_GetCMPT0\(TIM_TimerTypeDef *TIM_TIMERx\)](#)

功能描述: 获取定时器通道0捕获值

参见

被引用内容: 无

参数

输入参数:	TIM_TIMERx: TIMER0/TIMER1/TIMER2/TIMER3
输出参数:	无

返回

返回 值: 定时器通道0捕获值

注解

其它说明: 当定时器发生捕获事件时, 将捕获时刻的cnt值存储到该寄存器中

警告

无

示例代码:

```
uint32_t TIMER0_CAPValue = 0;
TIMER0_CAPValue = TIM_Timr_GetCMPT0(TIMER0); //获取定时器0捕获值
```

Date	Version	Author	Description
2016年06月06日	1.0	cfwu	修改

修改日志:

在文件 [lks32mc08x_tim.c](#) 第 399 行定义.

3.75.2.10 TIM_Timr_GetCMPT1()

```
uint32_t TIM_Timr_GetCMPT1 (
    TIM_TimrTypeDef * TIM_TIMERx )
```

函数名称: uint32_t [TIM_Timr_GetCMPT1\(TIM_TimrTypeDef *TIM_TIMERx\)](#)

功能描述: 获取定时器通道1捕获值

参见

被引用内容: 无

参数

输入参数:	TIM_TIMERx: TIMER0/TIMER1/TIMER2/TIMER3
输出参数:	无

返回

返回 值: 定时器通道1捕获值

注解

其它说明: 当定时器发生捕获事件时, 将捕获时刻的cnt值存储到该寄存器中

警告

无

示例代码:

```
uint32_t TIMeR0_CaPValue = 0;
TIMeR0_CaPValue = TIM.Timer_GetCMPt1(TIMeR0); //获取定时器0捕获值
```

Date	Version	Author	Description
2016年06月06日	1.0	cfwu	修改

修改日志:

在文件 [lks32mc08x.tim.c](#) 第 424 行定义.

3.75.2.11 TIM_Timer_GetCount()

```
uint32_t TIM_Timer_GetCount (
    TIM_TimerTypeDef * TIMeRx )
```

函数名称: uint32_t [TIM_Timer_GetCount\(TIM_TimerTypeDef *TIMeRx\)](#)

功能描述: 获取Timer计数值

参见

被引用内容: 无

参数

输入参数:	TIMeRx: TIMeR0/TIMeR1/TIMeR2/TIMeR3
输出参数:	无

返回

返回 值: 无

注解

其它说明： 无

警告

无

示例代码:

```
uint32_t TIMER0.Value = 0;
TIMER0.Value = TIM_Timer_GetCount(TIMER0); //获取定时器0计数值
```

Date	Version	Author	Description
2016年06月06日	1.0	cfwu	修改

修改日志:

在文件 [lks32mc08x_tim.c](#) 第 374 行定义.

3.75.2.12 TIM_TimerCmd()

```
void TIM_TimerCmd (
    TIM_TimerTypeDef * TIMERx,
    FuncState state )
```

函数名称: void [TIM_TimerCmd\(TIM_TimerTypeDef *TIMERx, FuncState state\)](#)

功能描述: 定时器TIMERx使能和停止

参见

被引用内容： 无

参数

输入参数:	TIMERx: TIMER0/TIMER1/TIMER2/TIMER3 FuncState state: 使能与失能
输出参数:	无

返回

返回 值: 无

注解

其它说明： 无

警告

无

示例代码:

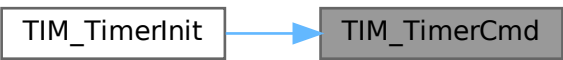
```
TIM.TimerCmd(TIMER0, ENABLE); // Timer0模块使能
```

Date	Version	Author	Description
2022年04月13日	1.0	HuangMG	修改

修改日志:

在文件 [lks32mc08x_tim.c](#) 第 187 行定义.

这是这个函数的调用关系图:



3.75.2.13 TIM_TimerInit()

```
void TIM_TimerInit (
    TIM.TimerTypeDef * TIMERx,
    TIM.TimerInitTypeDef * TIM_TimerInitStruct )
```

函数名称: void [TIM_TimerInit\(TIM.TimerTypeDef *TIMERx, TIM.TimerInitTypeDef *TIM_TimerInitStruct\)](#)

功能描述: 定时器初始化函数

参见

被引用内容: [SYS_ModuleClockCmd\(\)](#)

参数

输入参数:	TIMERx: TIMER0/TIMER1/TIMER2/TIMER3 TIM_TimerInitTypeDef *TIM_TimerInitStruct
输出参数:	无

返回

返回 值: 无

注解

其它说明: 无

警告

无

示例代码:

```
TIM_TimerInitTypeDef TIM_InitStruct;
TIM_TimerStrutInit (&TIM_InitStruct); // Timer结构体初始化
TIM_InitStruct.EN = ENABLE; // 定时器模块使能
TIM_InitStruct.Timer_ClockDiv = TIM_Clk_Div1; // 设置Timer模块时钟1分频系数
TIM_InitStruct.Timer_CH0.WorkMode = TIMER_OPMODE_CMP; // 设置Timer CH0 为比较模式
TIM_InitStruct.Timer_CH0Output = 0; // 计数器回零时, 比较模式输出极性控制
TIM_InitStruct.Timer_CH1.WorkMode = TIMER_OPMODE_CMP; // 设置Timer CH1 为比较模式
TIM_InitStruct.Timer_CH1Output = 0; // 计数器回零时, 比较模式输出极性控制
TIM_InitStruct.Timer_TH = 48000; // 定时器计数门限初始值500us
TIM_InitStruct.Timer_CMP0 = 36000; // 设置比较模式的CH0比较初始值25%转空比
TIM_InitStruct.Timer_CMP1 = 24000; // 设置比较模式的CH1比较初始值24000
TIM_InitStruct.Timer_IRQEna = Timer_IRQEna_CH0; // 开启Timer模块比较中断
TIM_TimerInit (TIMER0, &TIM_InitStruct);
```

Date	Version	Author	Description
2016年06月06日	1.0	cfwu	创建

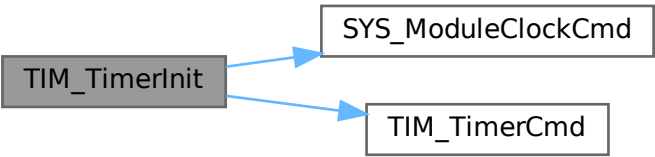
修改日志:

Date	Version	Author	Description
2022年04月13日	1.0	HuangMG	修改

修改日志: 去除定时器使能接口

在文件 [lks32mc08x_tim.c](#) 第 54 行定义.

函数调用图:



3.75.2.14 TIM_TimerStrutInit()

```
void TIM_TimerStrutInit (
    TIM_TimerInitTypeDef * TIM_TimerInitStruct )
```

函数名称: void TIM_TimerStrutInit(TIM_TimerInitTypeDef *TIM_TimerInitStruct)

功能描述: 定时器结构体初始化

参见

被引用内容: 无

参数

输入参数:	TIM_TimerInitTypeDef *TIM_TimerInitStruct
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
TIM_TimerInitTypeDef TIM_InitStruct;
TIM_TimerStrutInit (&TIM_InitStruct); // Timer结构体初始化
```

Date	Version	Author	Description
2016年06月06日	1.0	cfwu	创建

修改日志:

在文件 `lks32mc08x_tim.c` 第 147 行定义.

3.76 lks32mc08x_tim.c

[浏览该文件的文档.](#)

```

00001
00015 #include "lks32mc08x_tim.h"
00016
00054 void TIM_TimerInit(TIM_TimerTypeDef *TIMERx, TIM_TimerInitTypeDef *TIM_TimerInitStruct)
00055 {
00056     uint32_t tempIE;
00057     uint32_t tempVal;
00058     uint32_t tempFilter;
00059     uint32_t filter0;
00060     uint32_t filter1;
00061
00062     SYS_ModuleClockCmd(SYS_Module.TIMER, ENABLE); // 打开Timer时钟
00063
00064     TIM_TimerCmd(TIMERx, TIM_TimerInitStruct->EN);
00065
00066     TIMERx->CFG = TIM_TimerInitStruct->Timer_CH0.CapMode | (TIM_TimerInitStruct->Timer_CH0.WorkMode <<
2) | (TIM_TimerInitStruct->Timer_CH0.Output << 3) | (TIM_TimerInitStruct->Timer_CH1.CapMode << 4) |
(TIM_TimerInitStruct->Timer_CH1.WorkMode << 6) | (TIM_TimerInitStruct->Timer_CH1.Output << 7) |
(TIM_TimerInitStruct->Timer.ClockDiv << 8);
00067     TIMERx->TH = TIM_TimerInitStruct->Timer.TH;
00068     TIMERx->CMP0 = TIM_TimerInitStruct->Timer.CMP0;
00069     TIMERx->CMP1 = TIM_TimerInitStruct->Timer.CMP1;
00070
00071     /*使能定时器中断*/
00072     /*滤波参数*/
00073     tempIE = UTIMER_IE;
00074     filter0 = TIM_TimerInitStruct->Timer.Filter0 & 0x000F; /*滤波参数只使用低四位*/
00075     filter1 = TIM_TimerInitStruct->Timer.Filter1 & 0x000F;
00076     if (TIMERx == TIMER0)
00077     {
00078         tempVal = 0x0007;
00079         tempIE &= ~tempVal;
00080         tempIE |= (TIM_TimerInitStruct->Timer.IRQEna);
00081
00082         tempFilter = UTIMER_FLT_TH01;
00083         tempFilter &= 0xFF00;
00084         tempFilter |= filter0;
00085         tempFilter |= (filter1 << 4);
00086         UTIMER_FLT_TH01 = tempFilter;
00087     }
00088     else if (TIMERx == TIMER1)
00089     {
00090         tempVal = 0x0038;
00091         tempIE &= ~tempVal;
00092         tempIE |= (TIM_TimerInitStruct->Timer.IRQEna << 3);
00093
00094         tempFilter = UTIMER_FLT_TH01;
00095         tempFilter &= 0x00FF;
00096         tempFilter |= (filter0 << 8);
00097         tempFilter |= (filter1 << 12);
00098         UTIMER_FLT_TH01 = tempFilter;
00099     }
00100     else if (TIMERx == TIMER2)
00101     {
00102         tempVal = 0x01C0;
00103         tempIE &= ~tempVal;
00104         tempIE |= (TIM_TimerInitStruct->Timer.IRQEna << 6);
00105
00106         tempFilter = UTIMER_FLT_TH23;
00107         tempFilter &= 0xFF00;
00108         tempFilter |= filter0;
00109         tempFilter |= (filter1 << 4);
00110         UTIMER_FLT_TH23 = tempFilter;
00111     }

```

```

00112     else if (TIMERx == TIMER3)
00113     {
00114         tempVal = 0x0E00;
00115         tempIE &= ~tempVal;
00116         tempIE |= (TIM.TimerInitStruct->Timer_IRQEna << 9);
00117
00118         tempFilter = UTIMER.FLT.TH23;
00119         tempFilter &= 0x00FF;
00120         tempFilter |= (filter0 << 8);
00121         tempFilter |= (filter1 << 12);
00122         UTIMER.FLT.TH23 = tempFilter;
00123     }
00124     UTIMER.IE = tempIE;
00125 }
00126
00147 void TIM.TimerStrutInit(TIM.TimerInitTypeDef *TIM.TimerInitStruct)
00148 {
00149     TIM.TimerInitStruct->Timer_CH0_CapMode = TIMER_CapMode_None;
00150     TIM.TimerInitStruct->Timer_CH0_WorkMode = TIMER_OPMODE_CMP;
00151     TIM.TimerInitStruct->Timer_CH0_Output = 0;
00152
00153     TIM.TimerInitStruct->Timer_CH1_CapMode = TIMER_CapMode_None;
00154     TIM.TimerInitStruct->Timer_CH1_WorkMode = TIMER_OPMODE_CMP;
00155     TIM.TimerInitStruct->Timer_CH1_Output = 0;
00156
00157     TIM.TimerInitStruct->Timer_TH = 1000;
00158     TIM.TimerInitStruct->Timer_CMP0 = 500;
00159     TIM.TimerInitStruct->Timer_CMP1 = 500;
00160     TIM.TimerInitStruct->Timer_Filter0 = 0;
00161     TIM.TimerInitStruct->Timer_Filter1 = 0;
00162     TIM.TimerInitStruct->Timer_ClockDiv = TIM_Clk_Div1;
00163     TIM.TimerInitStruct->Timer_IRQEna = Timer_IRQEna_None;
00164     TIM.TimerInitStruct->EN = DISABLE;
00165 }
00166
00187 void TIM.TimerCmd(TIM.TimerTypeDef *TIMERx, FuncState state)
00188 {
00189     if (state != DISABLE)
00190     {
00191         if (TIMERx == TIMER0)
00192         {
00193             UTIMER.CFG |= BIT4;
00194         }
00195         else if (TIMERx == TIMER1)
00196         {
00197             UTIMER.CFG |= BIT5;
00198         }
00199         else if (TIMERx == TIMER2)
00200         {
00201             UTIMER.CFG |= BIT6;
00202         }
00203         else if (TIMERx == TIMER3)
00204         {
00205             UTIMER.CFG |= BIT7;
00206         }
00207     }
00208     else
00209     {
00210         if (TIMERx == TIMER0)
00211         {
00212             UTIMER.CFG &= ~BIT4;
00213         }
00214         else if (TIMERx == TIMER1)
00215         {
00216             UTIMER.CFG &= ~BIT5;
00217         }
00218         else if (TIMERx == TIMER2)
00219         {
00220             UTIMER.CFG &= ~BIT6;
00221         }
00222         else if (TIMERx == TIMER3)
00223         {
00224             UTIMER.CFG &= ~BIT7;
00225         }
00226     }
00227 }
00228
00255 uint32_t TIM_GetIRQFlag(TIM.TimerTypeDef *TIMERx, u32 timer_if)
00256 {
00257     if (TIMERx == TIMER0)
00258     {
00259         if ((UTIMER_IF & timer_if & UTIMER_IE) == 0)
00260         {
00261             return 0;
00262         }
00263     }
00264     else
00265     {

```

```

00265         return 1;
00266     }
00267 }
00268
00269 if (TIMERx == TIMER1)
00270 {
00271     if ((UTIMER_IF >> 3) & timer_if & (UTIMER_IE >> 3)) == 0)
00272     {
00273         return 0;
00274     }
00275     else
00276     {
00277         return 1;
00278     }
00279 }
00280
00281 if (TIMERx == TIMER2)
00282 {
00283     if ((UTIMER_IF >> 6) & timer_if & (UTIMER_IE >> 6)) == 0)
00284     {
00285         return 0;
00286     }
00287     else
00288     {
00289         return 1;
00290     }
00291 }
00292
00293 if (TIMERx == TIMER3)
00294 {
00295     if ((UTIMER_IF >> 9) & timer_if & (UTIMER_IE >> 9)) == 0)
00296     {
00297         return 0;
00298     }
00299     else
00300     {
00301         return 1;
00302     }
00303 }
00304 return 0;
00305 }
00306
00334 void TIM_ClearIRQFlag(TIM_TimerTypeDef *TIMERx, uint32_t tempFlag)
00335 {
00336     if (TIMERx == TIMER0)
00337     {
00338         UTIMER_IF = tempFlag;
00339     }
00340     if (TIMERx == TIMER1)
00341     {
00342         UTIMER_IF = tempFlag << 3;
00343     }
00344     if (TIMERx == TIMER2)
00345     {
00346         UTIMER_IF = tempFlag << 6;
00347     }
00348     if (TIMERx == TIMER3)
00349     {
00350         UTIMER_IF = tempFlag << 9;
00351     }
00352 }
00353
00374 uint32_t TIM_Timer_GetCount(TIM_TimerTypeDef *TIMERx)
00375 {
00376     return TIMERx->CNT;
00377 }
00378
00399 uint32_t TIM_Timer_GetCMPT0(TIM_TimerTypeDef *TIM_TIMERx)
00400 {
00401     return TIM_TIMERx->CMPT0;
00402 }
00403
00424 uint32_t TIM_Timer_GetCMPT1(TIM_TimerTypeDef *TIM_TIMERx)
00425 {
00426     return TIM_TIMERx->CMPT1;
00427 }
00428
00464 void TIM_ECDInit(TIM_ECDTypeDef *ECDx, TIM_ECDInitTypeDef *TIM_ECDInitStruct)
00465 {
00466     uint32_t tempIE;
00467     uint32_t tempFilter;
00468     uint32_t filter0;
00469     uint32_t filter1;
00470     uint32_t tempTimerCfg;
00471
00472     SYS_ModuleClockCmd(SYS_Module.TIMER, ENABLE); /*打开Timer时钟*/
00473

```

```

00474     ECDx->CFG = TIM_ECDInitStruct->ECD_Mode << 8;
00475     ECDx->TH = TIM_ECDInitStruct->ECD_TH;
00476
00477     filter0 = TIM_ECDInitStruct->ECD_Filter0 & 0x000F; /*滤波参数只使用低四位*/
00478     filter1 = TIM_ECDInitStruct->ECD_Filter1 & 0x000F;
00479     tempIE = UTIMER_IE;
00480     if (ECDx == ECD0)
00481     {
00482         tempFilter = UTIMER_FLT_TH23;
00483         tempFilter &= 0xFF00;
00484         tempFilter |= filter0;
00485         tempFilter |= (filter1 << 4);
00486         UTIMER_FLT_TH23 = tempFilter;
00487
00488         tempIE |= (TIM_ECDInitStruct->ECD_IRQEna << 12);
00489
00490         tempTimerCfg = TIMER2->CFG;
00491         tempTimerCfg &= ~(0x0300);
00492         tempTimerCfg |= (TIM_ECDInitStruct->ECD_ClockDiv << 8);
00493         TIMER2->CFG = tempTimerCfg;
00494         UTIMER_CFG |= BIT8;
00495     }
00496     else if (ECDx == ECD1)
00497     {
00498         tempFilter = UTIMER_FLT_TH23;
00499         tempFilter &= 0x00FF;
00500         tempFilter |= (filter0 << 8);
00501         tempFilter |= (filter1 << 12);
00502         UTIMER_FLT_TH23 = tempFilter;
00503
00504         tempIE |= (TIM_ECDInitStruct->ECD_IRQEna << 14);
00505
00506         tempTimerCfg = TIMER3->CFG;
00507         tempTimerCfg &= ~(0x0300);
00508         tempTimerCfg |= (TIM_ECDInitStruct->ECD_ClockDiv << 8);
00509         TIMER3->CFG = tempTimerCfg;
00510         UTIMER_CFG |= BIT9;
00511     }
00512     UTIMER_IE = tempIE;
00513 }
00514
00535 void TIM_ECDStructInit(TIM_ECDInitTypeDef *TIM_ECDInitStruct)
00536 {
00537     TIM_ECDInitStruct->ECD_Mode = ECD_Mode_T1;
00538     TIM_ECDInitStruct->ECD_TH = 1000;
00539     TIM_ECDInitStruct->ECD_ClockDiv = TIM_Clk_Div1;
00540     TIM_ECDInitStruct->ECD_Filter0 = 0;
00541     TIM_ECDInitStruct->ECD_Filter1 = 0;
00542     TIM_ECDInitStruct->ECD_IRQEna = ECD_IRQEna_UnderFlow | ECD_IRQEna_OverFlow;
00543 }
00544
00565 void TIM_ECDCmd(TIM_ECDTypeDef *ECDx, FuncState state)
00566 {
00567     uint32_t tempEna;
00568     tempEna = UTIMER_CFG;
00569
00570     if (state != DISABLE)
00571     {
00572         if (ECDx == ECD0)
00573         {
00574             tempEna |= BIT8;
00575         }
00576         else if (ECDx == ECD1)
00577         {
00578             tempEna |= BIT9;
00579         }
00580     }
00581     else
00582     {
00583         if (ECDx == ECD0)
00584         {
00585             tempEna &= ~BIT8;
00586         }
00587         else if (ECDx == ECD1)
00588         {
00589             tempEna &= ~BIT9;
00590         }
00591     }
00592     UTIMER_CFG = tempEna;
00593 }
00594
00615 uint32_t TIM_ECD_GetCount(TIM_ECDTypeDef *TIM_ECDx)
00616 {
00617     return TIM_ECDx->CNT;
00618 }
00619
00645 uint32_t ECD_GetIRQFlag(TIM_ECDTypeDef *ECDx, u32 timer_if)

```

```

00646 {
00647     if (ECDx == ECD0)
00648     {
00649         if (((UTIMER_IF >> 12) & timer_if & (UTIMER_IE >> 12)) == 0)
00650         {
00651             return 0;
00652         }
00653         else
00654         {
00655             return 1;
00656         }
00657     }
00658
00659     if (ECDx == ECD1)
00660     {
00661         if (((UTIMER_IF >> 14) & timer_if & (UTIMER_IE >> 14)) == 0)
00662         {
00663             return 0;
00664         }
00665         else
00666         {
00667             return 1;
00668         }
00669     }
00670     return 0;
00671 }
00672
00673 /*****
00674 函数名称:      void ECD_ClearIRQFlag(TIM_ECDTypeDef *ECDx , uint32_t tempFlag)
00675 功能描述:      清除ECD中断标志
00676 操作的表:      无
00677 输入参数:      ECD中断标志
00678 输出参数:      无
00679 返回值:        无
00680 其它说明:
00681 修改日期      版本号      修改人      修改内容
00682 -----
00683 2022/4/13      V1.0      HuangMG      创建
00684 *****/
00711 void ECD_ClearIRQFlag(TIM_ECDTypeDef *ECDx, uint32_t tempFlag)
00712 {
00713     if (ECDx == ECD0)
00714     {
00715         UTIMER_IF = tempFlag << 12;
00716     }
00717     if (ECDx == ECD1)
00718     {
00719         UTIMER_IF = tempFlag << 14;
00720     }
00721 }
00722 /***** (C) COPYRIGHT LINKO SEMICONDUCTOR *****/

```

3.77 source/lks32mc08x_tmp.c 文件参考

文件名称: lks32mc08x_tmp.c

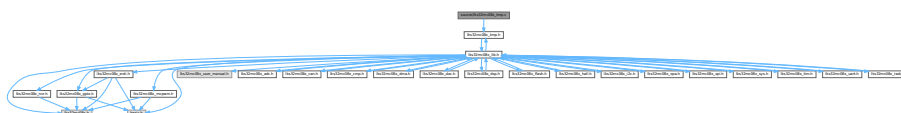
文件标识: 无

内容摘要: 芯片内置温度传感器外设驱动程序

其它说明: 无

```
#include "lks32mc08x_tmp.h"
```

lks32mc08x_tmp.c 的引用(Include)关系图:



函数

- void TempSensor_Init (void)

函数名称: `void TempSensor_Init(void)`

- `s16 GetCurrentTemperature (s16 ADC_value)`

函数名称: `s16 GetCurrentTemperature(s16 ADC_value)`

变量

- `Stru_TempertureCof_Def m_TempertureCof`

温度传感器系数

3.77.1 详细描述

文件名称: `lks32mc08x_tmp.c`

文件标识: 无

内容摘要: 芯片内置温度传感器外设驱动程序

其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2015年11月5日	1.0	Howlet	创建

修改日志:

在文件 `lks32mc08x_tmp.c` 中定义.

3.77.2 函数说明

3.77.2.1 GetCurrentTemperature()

```
s16 GetCurrentTemperature (
    s16 ADC_value )
```

函数名称: `s16 GetCurrentTemperature(s16 ADC_value)`

功能描述: 获取当前温度值

参见

被引用内容: 无

参数

输入参数:	ADC_value: ADC通道14为温度传感器, ADC采样结果值
输出参数:	无

返回

返回 值: t.Temperture: 当前温度值, 单位: 1个Lsb代表0.1度

注解

其它说明: ADC_value输入的ADC值不区分左右对齐, 函数内已经进行ADC结果左右对齐判断

警告

无

示例代码:

```
s16 temp_value = 0;
temp_value = GetCurrentTemperature(ADC0_DATA0); //获取当前温度值, 单位0.1度
```

Date	Version	Author	Description
2015年11月5日	1.0	Howlet	创建

修改日志:

在文件 [lks32mc08x_tmp.c](#) 第 67 行定义.

3.77.2.2 TempSensor_Init()

```
void TempSensor_Init (
    void )
```

函数名称: void [TempSensor_Init\(void\)](#)

功能描述: 温度传感器初始化

参见

被引用内容: 无

参数

输入参数:	无
输出参数:	无

返回

返回值：无

注解

其它说明：无

警告

无

示例代码:

```
TempSensor.Init(); //温度传感器初始化
```

Date	Version	Author	Description
2015年11月5日	1.0	Howlet	创建

修改日志:

在文件 [lks32mc08x.tmp.c](#) 第 38 行定义.

函数调用图:



3.77.3 变量说明

3.77.3.1 m_TempertureCof

```
Stru.TempertureCof_Def m_TempertureCof
```

温度传感器系数

在文件 [lks32mc08x.tmp.c](#) 第 17 行定义.

3.78 lks32mc08x_tmp.c

[浏览该文件的文档.](#)

```

00001
00015 #include "lks32mc08x.tmp.h"
00016
00017 Stru_TempertureCof_Def m_TempertureCof;
00038 void TempSensor_Init(void)
00039 {
00040     SYS_WR_PROTECT = 0x7a83; /* 解除系统寄存器写保护 */
00041     SYS_AFE_REG5 |= BIT13; /* 打开温度传感器开关 */
00042
00043     m_TempertureCof.nCofA = Read_Trim(0x00000398);
00044     m_TempertureCof.nOffsetB = Read_Trim(0x0000039C);
00045 }
00046
00067 s16 GetCurrentTemperature(s16 ADC_value)
00068 {
00069     s16 t_Temperture;
00070     if (ADC0->CFG & BIT0)
00071     {
00072         t_Temperture = (m_TempertureCof.nOffsetB - ((s32)m_TempertureCof.nCofA * ADC_value) / 1000);
00073     }
00074     else
00075     {
00076         t_Temperture = (m_TempertureCof.nOffsetB - ((s32)m_TempertureCof.nCofA * (ADC_value >> 4)) /
1000);
00077     }
00078
00079     return t_Temperture;
00080 }
00081 /***** (C) COPYRIGHT LINKO SEMICONDUCTOR *****END OF FILE*****/

```

3.79 source/lks32mc08x_uart.c 文件参考

文件名称: lks32mc08x_uart.c
文件标识: 无
内容摘要: UART外设驱动程序
其它说明: 无

#include "lks32mc08x_uart.h"
lks32mc08x_uart.c 的引用(Include)关系图:



函数

- void **UART_Init** (**UART_TypeDef** *UARTx, **UART_InitTypeDef** *UART_InitStruct)
函数名称: void *UART_Init(UART_TypeDef* UARTx, UART_InitTypeDef* UART_InitStruct)*
- void **UART_StructInit** (**UART_InitTypeDef** *UART_InitStruct)
函数名称: void *UART_StructInit(UART_InitTypeDef* UART_InitStruct)*
- void **UART_SendData** (**UART_TypeDef** *UARTx, uint32_t n)
函数名称: void *UART_SENDDATA(UART_TypeDef* UARTx, uint32_t n)*
- uint32_t **UART_ReadData** (**UART_TypeDef** *UARTx)
函数名称: uint32_t *UART_ReadData(UART_TypeDef* UARTx)*
- uint8_t **UART_SendAddr_485** (**UART_TypeDef** *UARTx, uint8_t n)
函数名称: uint8_t *UART_SendAddr_485(UART_TypeDef* UARTx, uint8_t n)*

- `uint8_t UART_SendData_485 (UART_TypeDef *UARTx, uint8_t n)`
函数名称: `uint8_t UART_SendData_485(UART_TypeDef *UARTx, uint8_t n)`
- `uint32_t UART_GetIRQFlag (UART_TypeDef *UARTx, uint32_t tempFlag)`
函数名称: `uint32_t UART_GetIRQFlag(UART_TypeDef *UARTx, uint32_t tempFlag)`
- `void UART_ClearIRQFlag (UART_TypeDef *UARTx, uint32_t tempFlag)`
函数名称: `void UART_ClearIRQFlag(UART_TypeDef *UARTx, uint32_t tempFlag)`

3.79.1 详细描述

文件名称: `lks32mc08x_uart.c`
文件标识: 无
内容摘要: UART外设驱动程序
其它说明: 无

版权所有

(C)2015, LINKO SEMICONDUCTOR Co.ltd

Date	Version	Author	Description
2016年05月21日	1.0	cfwu	创建

修改日志:

在文件 `lks32mc08x_uart.c` 中定义.

3.79.2 函数说明

3.79.2.1 UART_ClearIRQFlag()

```
void UART_ClearIRQFlag (  
    UART_TypeDef * UARTx,  
    uint32_t tempFlag )
```

函数名称: `void UART_ClearIRQFlag(UART_TypeDef *UARTx, uint32_t tempFlag)`

功能描述: 清除UART中断标志

参见

被引用内容: UART0 , UART1

参数

输入参数:	UART_TypeDef: UART0/UART1	
	tempFlag参数可选:	
	宏定义	说明
	UART_IF_SendOver	发送完成中断
	UART_IF_RcvOver	接收完成中断
	UART_IF_SendBufEmpty	发送缓冲区空中断
	UART_IF_StopError	停止位错误
	UART_IF_CheckError	校验错误
输出参数:	无	

返回

返回 值: 无

注解

其它说明: 无

警告

无

示例代码:

```
if(UART_GetIRQFlag(UART0,UART_IF_SendOver))//获取发送完成中断标志位
{
    UART_ClearIRQFlag(UART0,UART_IF_SendOver)//清除发送完成中断标志位
}
```

Date	Version	Author	Description
2012年04月21日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_uart.c](#) 第 332 行定义.

3.79.2.2 UART_GetIRQFlag()

```
uint32_t UART_GetIRQFlag (
    UART_TypeDef * UARTx,
    uint32_t tempFlag )
```

函数名称: uint32_t [UART_GetIRQFlag\(UART_TypeDef *UARTx , uint32_t tempFlag\)](#)

功能描述: 获取UART中断标志

参见

被引用内容： UART0 ， UART1

参数

输入参数:	UART_TypeDef: UART0/UART1	
	tempFlag参数可选:	
	宏定义	说明
	UART_IF_SendOver	发送完成中断
	UART_IF_RcvOver	接收完成中断
	UART_IF_SendBufEmpty	发送缓冲区空中断
	UART_IF_StopError	停止位错误
	UART_IF_CheckError	校验错误
输出参数:	无	

返回

返回值： 无

注解

其它说明： 无

警告

只有对应中断使能后， 改为才能读取， 如果对应中断未使能， 读取结果一直为0

示例代码:

```
if (UART_GetIRQFlag (UART0, UART_IF_SendOver)) //获取发送完成中断标志位
{
    UART_ClearIRQFlag (UART0, UART_IF_SendOver) //清除发送完成中断标志位
}
```

Date	Version	Author	Description
2012年04月21日	1.0	HuangMG	创建

修改日志:

在文件 lks32mc08x_uart.c 第 293 行定义.

3.79.2.3 UART_Init()

```
void UART_Init (
    UART_TypeDef * UARTx,
    UART_InitTypeDef * UART_InitStruct )
```

函数名称: void UART_Init(UART_TypeDef* UARTx, EUART_InitTypeDef* UART_InitStruct)

功能描述: UART初始化函数

参见

被引用内容: [SYS_ModuleClockCmd\(\)](#)

参数

输入参数:	UART_TypeDef: UART0/UART1 , UART_InitTypeDef
输出参数:	无

返回

返回值: 无

注解

其它说明: 串口波特率计算使用UART_MCU_MCLK系统时钟进行计算，使用前需要确认系统时钟频率

警告

无

示例代码:

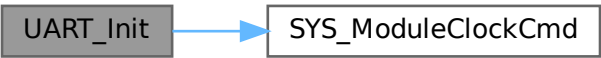
```
UART_InitTypeDef UART_InitStruct;
UART_StructInit(&UART_InitStruct);
UART_InitStruct.BaudRate = 38400; // 设置波特率38400
UART_InitStruct.WordLength = UART_WORDLENGTH_8b; // 发送数据长度8位
UART_InitStruct.StopBits = UART_STOPBITS_1b; // 停止位1位
UART_InitStruct.FirstSend = UART_FIRSTSEND_LSB; // 先发送LSB
UART_InitStruct.ParityMode = UART_Parity_NO; // 无奇偶校验
UART_InitStruct.IRQEna = UART_IRQEna_SendOver | UART_IRQEna_RcvOver; // 串口中断使能
UART_Init(UART0, &UART_InitStruct);
```

Date	Version	Author	Description
2016年05月21日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_uart.c](#) 第 44 行定义.

函数调用图:



3.79.2.4 UART_ReadData()

```
uint32_t UART_ReadData (
    UART_TypeDef * UARTx )
```

函数名称: uint32_t **UART_ReadData**(UART_TypeDef *UARTx)

功能描述: UART读缓冲区数据

参见

被引用内容: UART0 , UART1

参数

输入参数:	UART_TypeDef: UART0/UART1
输出参数:	无

返回

返回 值: 返回串口接收缓冲区的1Byte数据

注解

其它说明: 无

警告

无

示例代码:

```
u8 UART_BUFF = 0;
UART_BUFF = UART_ReadData (UART0); //读取串口0接收一字节数据
```

Date	Version	Author	Description
2016年05月21日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_uart.c](#) 第 174 行定义.

3.79.2.5 UART_SendAddr_485()

```
uint8_t UART_SendAddr_485 (
    UART_TypeDef * UARTx,
    uint8_t n )
```

函数名称: `uint8_t UART_SendAddr_485(UART_TypeDef *UARTx, uint8_t n)`

功能描述: 发送485通讯中需要接收数据的从机地址

参见

被引用内容: UART0 , UART1

参数

输入参数:	UART_TypeDef: UART0/UART1 , n: 要发生1Byrt地址
输出参数:	无

返回

返回值: 1: 发送失败, 0: 发送成功

注解

其它说明: cnt作用是延时防止程序卡死。在发送数据后立即发送地址的情况下,会判断等待发送缓冲区空在进行地址发送,如果长时间还未发送完成(cnt计数时间),则跳出while循环。如果串口波特率较低cnt计数已满,会出现数据还未发送完成进行下一个数据发送,为了避免此类问题发送可以增加cnt值。

警告

无

示例代码:

```
UART_SendAddr_485 (UART0, 0x55); //串口0发送485从机接收地址为0x55
```


Date	Version	Author	Description
2012年04月21日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_uart.c](#) 第 201 行定义.

3.79.2.6 UART_SendData()

```
void UART_SendData (
    UART_TypeDef * UARTx,
    uint32_t n )
```

函数名称: void UART_SENDDATA(UART_TypeDef *UARTx, uint32_t n)

功能描述: UART发送数据

参见

被引用内容: UART0 , UART1

参数

输入参数:	UART_TypeDef: UART0/UART1 , n: 要发生1Byte数据
输出参数:	无

返回

返回值: 无

注解

其它说明: 无

警告

无

示例代码:

```
UART_SendData (UART0, 0x12); //串口0发送0x12一字节数据
```

Date	Version	Author	Description
2016年05月21日	1.0	cfwu	创建

修改日志:

在文件 [lks32mc08x_uart.c](#) 第 148 行定义.

3.79.2.7 UART_SendData_485()

```
uint8_t UART_SendData_485 (
    UART_TypeDef * UARTx,
    uint8_t n )
```

函数名称: `uint8_t UART_SendData_485(UART_TypeDef *UARTx, uint8_t n)`

功能描述: 发送485通讯中的数据信息

参见

被引用内容: UART0 , UART1

参数

输入参数:	UART_TypeDef: UART0/UART1 , n: 要发生1Byrt数据
输出参数:	无

返回

返回值: 1: 发送失败, 0: 发送成功

注解

其它说明: cnt作用是延时防止程序卡死。在发送数据后立即发送地址的情况下,会判断等待发送缓冲区空在进行地址发送,如果长时间还未发送完成(cnt计数时间),则跳出while循环。如果串口波特率较低cnt计数已满,会出现数据还未发送完成进行下一个数据发送,为了避免此类问题发送可以增加cnt值。

警告

无

示例代码:

```
UART_SendData_485 (UART0, 0x12); //串口0发送485数据为0x12
```

Date	Version	Author	Description
2012年04月21日	1.0	HuangMG	创建

修改日志:

在文件 [lks32mc08x_uart.c](#) 第 243 行定义.

3.79.2.8 UART_StructInit()

```
void UART_StructInit (
    UART_InitTypeDef * UART_InitStruct )
```

函数名称: void [UART_StructInit\(UART_InitTypeDef* UART_InitStruct\)](#)

功能描述: UART结构体初始化函数

参见

被引用内容: 无

参数

输入参数:	UART_InitTypeDef
输出参数:	无

返回

返回 值: 无

注解

其它说明: 无

警告

无

示例代码:

```
UART_InitTypeDef UART_InitStruct;
UART_StructInit(&UART_InitStruct); //初始化UART相关配置结构体变量
```

Date	Version	Author	Description
2016年05月21日	1.0	cfwu	创建

修改日志:

在文件 `lks32mc08x_uart.c` 第 112 行定义.

3.80 lks32mc08x_uart.c

[浏览该文件的文档.](#)

```

00001
00015 #include "lks32mc08x_uart.h"
00016
00044 void UART_Init(UART_TypeDef *UARTx, UART_InitTypeDef *UART_InitStruct)
00045 {
00046     uint32_t DIVHDATA;
00047     uint32_t DIVLDATA;
00048     uint32_t parity;
00049     uint32_t parityEna;
00050
00051     if (UARTx == UART0)
00052     {
00053         SYS_ModuleClockCmd(SYS_Module_UART0, ENABLE);
00054     }
00055     else if (UARTx == UART1)
00056     {
00057         SYS_ModuleClockCmd(SYS_Module_UART1, ENABLE);
00058     }
00059
00060     DIVHDATA = (u32)((UART_MCU_MCLK / 4) / UART_InitStruct->BaudRate) / 256;
00061     DIVLDATA = (u32)((UART_MCU_MCLK / 4) / UART_InitStruct->BaudRate) - (DIVHDATA * 256) - 1;
00062
00063     SYS_WR_PROTECT = 0x7A83; // 解除系统寄存器写保护
00064     SYS_CLK_DIV2 = 3;
00065     SYS_WR_PROTECT = 0;
00066     UARTx->DIVL = DIVLDATA;
00067     UARTx->DIVH = DIVHDATA;
00068
00069     if (UART_InitStruct->ParityMode == UART_Parity_EVEN)
00070     {
00071         parityEna = ENABLE;
00072         parity = 0;
00073     }
00074     else if (UART_InitStruct->ParityMode == UART_Parity_ODD)
00075     {
00076         parityEna = ENABLE;
00077         parity = 1;
00078     }
00079     else
00080     {
00081         parityEna = DISABLE;
00082         parity = 0;
00083     }
00084     UARTx->CTRL = UART_InitStruct->WordLength | (UART_InitStruct->StopBits << 1) |
    (UART_InitStruct->FirstSend << 2) | (parity << 3) | (parityEna << 4) | (UART_InitStruct->MultiDropEna
    << 5) | (UART_InitStruct->Bit9Value << 6);
00085
00086     UARTx->INV = (UART_InitStruct->TXD_INV << 1) | UART_InitStruct->RXD_INV;
00087
00088     UARTx->ADR = UART_InitStruct->Match485Addr;
00089     UARTx->IE = UART_InitStruct->IRQEna;
00090 }
00091
00112 void UART_StructInit(UART_InitTypeDef *UART_InitStruct)
00113 {
00114     UART_InitStruct->BaudRate = 9600;
00115     UART_InitStruct->WordLength = UART_WORDLENGTH_8b;
00116     UART_InitStruct->StopBits = UART_STOPBITS_1b;
00117     UART_InitStruct->FirstSend = UART_FIRSTSEND_LSB;
00118     UART_InitStruct->ParityMode = UART_Parity_NO;
00119
00120     UART_InitStruct->MultiDropEna = DISABLE;
00121     UART_InitStruct->Bit9Value = 0;
00122
00123     UART_InitStruct->Match485Addr = 0;
00124     UART_InitStruct->IRQEna = 0;
00125     UART_InitStruct->RXD_INV = DISABLE;
00126     UART_InitStruct->TXD_INV = DISABLE;
00127 }
00128
00148 void UART_SendData(UART_TypeDef *UARTx, uint32_t n)
00149 {
00150     UARTx->BUFF = n;
00151 }
00152
00174 uint32_t UART_ReadData(UART_TypeDef *UARTx)

```

```

00175 {
00176     return UARTx->BUFF;
00177 }
00178
00201 uint8_t UART_SendAddr_485(UART_TypeDef *UARTx, uint8_t n)
00202 {
00203     u32 cnt = 0xffffffff;
00204     UARTx->CTRL |= BIT6;
00205     UARTx->BUFF = n;
00206     while (!(UARTx->STT & BIT0)) /*发送缓冲区非空*/
00207     {
00208         if (cnt > 0)
00209         {
00210             cnt--;
00211             __NOP();
00212         }
00213         else
00214         {
00215             return 1; /*发送失败*/
00216         }
00217     }
00218     return 0; /*发送成功*/
00219 }
00220
00243 uint8_t UART_SendData_485(UART_TypeDef *UARTx, uint8_t n)
00244 {
00245     u32 cnt = 0xffffffff;
00246     UARTx->CTRL &= ~BIT6;
00247     UARTx->BUFF = n;
00248     while (!(UARTx->STT & BIT0)) /*发送缓冲区非空*/
00249     {
00250         if (cnt > 0)
00251         {
00252             cnt--;
00253             __NOP();
00254         }
00255         else
00256         {
00257             return 1; /*发送失败*/
00258         }
00259     }
00260     return 0; /*发送成功*/
00261 }
00262
00293 uint32_t UART_GetIRQFlag(UART_TypeDef *UARTx, uint32_t tempFlag)
00294 {
00295     if ((UARTx->IF) & tempFlag & (UARTx->IE))
00296     {
00297         return 1;
00298     }
00299     return 0;
00300 }
00301
00332 void UART_ClearIRQFlag(UART_TypeDef *UARTx, uint32_t tempFlag)
00333 {
00334     UARTx->IF = tempFlag;
00335 }
00336
00337 /***** (C) COPYRIGHT LINKO SEMICONDUCTOR *****END OF FILE*****/

```


Index

ADC0
lks32mc08x_adc.h, 118

ADC_10_TIMES_SAMPLE
lks32mc08x_adc.h, 118

ADC_11_TIMES_SAMPLE
lks32mc08x_adc.h, 118

ADC_12_TIMES_SAMPLE
lks32mc08x_adc.h, 118

ADC_13_TIMES_SAMPLE
lks32mc08x_adc.h, 118

ADC_14_TIMES_SAMPLE
lks32mc08x_adc.h, 118

ADC_15_TIMES_SAMPLE
lks32mc08x_adc.h, 119

ADC_16_TIMES_SAMPLE
lks32mc08x_adc.h, 119

ADC_17_TIMES_SAMPLE
lks32mc08x_adc.h, 119

ADC_18_TIMES_SAMPLE
lks32mc08x_adc.h, 119

ADC_19_TIMES_SAMPLE
lks32mc08x_adc.h, 119

ADC_1_TIMES_SAMPLE
lks32mc08x_adc.h, 119

ADC_1SEG_TRG
lks32mc08x_adc.h, 120

ADC_20_TIMES_SAMPLE
lks32mc08x_adc.h, 120

ADC_2_TIMES_SAMPLE
lks32mc08x_adc.h, 120

ADC_2SEG_TRG
lks32mc08x_adc.h, 120

ADC_3_TIMES_SAMPLE
lks32mc08x_adc.h, 120

ADC_4_TIMES_SAMPLE
lks32mc08x_adc.h, 120

ADC_4SEG_TRG
lks32mc08x_adc.h, 121

ADC_5_TIMES_SAMPLE
lks32mc08x_adc.h, 121

ADC_6_TIMES_SAMPLE
lks32mc08x_adc.h, 121

ADC_7_TIMES_SAMPLE
lks32mc08x_adc.h, 121

ADC_8_TIMES_SAMPLE
lks32mc08x_adc.h, 121

ADC_9_TIMES_SAMPLE
lks32mc08x_adc.h, 121

ADC_ALL_IRQ_IF
lks32mc08x_adc.h, 122

ADC_CHANNEL_0
lks32mc08x_adc.h, 122

ADC_CHANNEL_1
lks32mc08x_adc.h, 122

ADC_CHANNEL_10
lks32mc08x_adc.h, 122

ADC_CHANNEL_11
lks32mc08x_adc.h, 122

ADC_CHANNEL_12
lks32mc08x_adc.h, 122

ADC_CHANNEL_13
lks32mc08x_adc.h, 123

ADC_CHANNEL_14
lks32mc08x_adc.h, 123

ADC_CHANNEL_15
lks32mc08x_adc.h, 123

ADC_CHANNEL_16
lks32mc08x_adc.h, 123

ADC_CHANNEL_17
lks32mc08x_adc.h, 123

ADC_CHANNEL_18
lks32mc08x_adc.h, 123

ADC_CHANNEL_19
lks32mc08x_adc.h, 124

ADC_CHANNEL_2
lks32mc08x_adc.h, 124

ADC_CHANNEL_3
lks32mc08x_adc.h, 124

ADC_CHANNEL_4
lks32mc08x_adc.h, 124

ADC_CHANNEL_5
lks32mc08x_adc.h, 124

ADC_CHANNEL_6
lks32mc08x_adc.h, 124

ADC_CHANNEL_7
lks32mc08x_adc.h, 125

ADC_CHANNEL_8
lks32mc08x_adc.h, 125

ADC_CHANNEL_9
lks32mc08x_adc.h, 125

ADC_ChannelConfig
lks32mc08x_adc.c, 420
lks32mc08x_adc.h, 132

ADC_ClearIRQFlag
lks32mc08x_adc.c, 421
lks32mc08x_adc.h, 133

ADC_CLK_DIV
ADC_InitTypeDef, 10

ADC_Clock_12M
lks32mc08x_adc.h, 125

ADC_Clock_24M
lks32mc08x_adc.h, 125

ADC_Clock_48M
lks32mc08x_adc.h, 125

ADC_DAT0_HTH
lks32mc08x_adc.h, 125

ADC_DAT0_LTH
lks32mc08x_adc.h, 126

ADC_DAT0_OV_IRQ_EN
lks32mc08x_adc.h, 126

ADC_DAT0_OV_IRQ_IF
lks32mc08x_adc.h, 126

ADC_EOS0_IRQ_EN
lks32mc08x_adc.h, 126

ADC_EOS0_IRQ_IF
lks32mc08x_adc.h, 126

ADC_EOS1_IRQ_EN
lks32mc08x_adc.h, 126

ADC_EOS1_IRQ_IF
lks32mc08x_adc.h, 127

ADC_EOS2_IRQ_EN
lks32mc08x_adc.h, 127

ADC_EOS2_IRQ_IF
lks32mc08x_adc.h, 127

ADC_EOS3_IRQ_EN
lks32mc08x_adc.h, 127

ADC_EOS3_IRQ_IF
lks32mc08x_adc.h, 127

ADC_GetConversionValue
lks32mc08x_adc.c, 423
lks32mc08x_adc.h, 134

ADC_GetIRQFlag
lks32mc08x_adc.c, 424
lks32mc08x_adc.h, 135

ADC_H_CONFLICT_IRQ_EN
lks32mc08x_adc.h, 127

ADC_H_CONFLICT_IRQ_IF
lks32mc08x_adc.h, 128

ADC_HARDWARE_T0_TRG
lks32mc08x_adc.h, 128

ADC_HARDWARE_T1_TRG
lks32mc08x_adc.h, 128

ADC_HARDWARE_T2_TRG
lks32mc08x_adc.h, 128

ADC_HARDWARE_T3_TRG
lks32mc08x_adc.h, 128

ADC_Init
lks32mc08x_adc.c, 425
lks32mc08x_adc.h, 137

ADC_InitTypeDef, 9

ADC_CLK_DIV, 10

Align, 10

Con_Sample, 11

DAT0_TH, 11

DAT0_TH_Type, 11

FirSeg_Ch, 11

FouSeg_Ch, 11

Gain0, 12

Gain1, 12

IE, 12

MCPWM_Trigger_En, 12

RefVol, 13

SecSeg_Ch, 13

ThrSeg_Ch, 13

Trigger_Cnt, 13

Trigger_Mode, 14

UTIMER_Trigger_En, 14

ADC_LEFT_ALIGN
lks32mc08x_adc.h, 128

ADC_MCPWM_T0_TRG
lks32mc08x_adc.h, 129

ADC_MCPWM_T1_TRG
lks32mc08x_adc.h, 129

ADC_MCPWM_T2_TRG
lks32mc08x_adc.h, 129

ADC_MCPWM_T3_TRG
lks32mc08x_adc.h, 129

ADC_RIGHT_ALIGN
lks32mc08x_adc.h, 129

ADC_S_CONFLICT_IRQ_EN
lks32mc08x_adc.h, 129

ADC_S_CONFLICT_IRQ_IF
lks32mc08x_adc.h, 130

ADC_SoftTrgEN
lks32mc08x_adc.c, 426
lks32mc08x_adc.h, 138

ADC_StructInit
lks32mc08x_adc.c, 427
lks32mc08x_adc.h, 139

ADC_TypeDef, 15

AMC_A0, 17

AMC_A1, 17

AMC_B0, 17

AMC_B1, 17

CFG, 17

CHN0, 18

CHN1, 18

CHN2, 18

CHN3, 18

CHN4, 18

CHN5, 18

CHN6, 19

CHN7, 19

CHN8, 19

CHN9, 19

CHNT0, 19

CHNT1, 19

DAT0, 20

DAT0_TH, 20

DAT1, 20

DAT10, 20

DAT11, 20

DAT12, 20

DAT13, 21

DAT14, [21](#)
 DAT15, [21](#)
 DAT16, [21](#)
 DAT17, [21](#)
 DAT18, [21](#)
 DAT19, [22](#)
 DAT2, [22](#)
 DAT3, [22](#)
 DAT4, [22](#)
 DAT5, [22](#)
 DAT6, [22](#)
 DAT7, [23](#)
 DAT8, [23](#)
 DAT9, [23](#)
 DC_A0, [23](#)
 DC_A1, [23](#)
 DC_B0, [23](#)
 DC_B1, [24](#)
 GAIN_0, [24](#)
 GAIN_1, [24](#)
 IE, [24](#)
 IF, [24](#)
 SD0, [24](#)
 SD1, [25](#)
 SWT, [25](#)
 TRG, [25](#)
 ADC_UTIMER_T0_TRG
 lks32mc08x_adc.h, [130](#)
 ADC_UTIMER_T1_TRG
 lks32mc08x_adc.h, [130](#)
 ADC_UTIMER_T2_TRG
 lks32mc08x_adc.h, [130](#)
 ADC_UTIMER_T3_TRG
 lks32mc08x_adc.h, [130](#)
 ADDR
 I2C_TypeDef, [61](#)
 ADR
 UART_TypeDef, [111](#)
 ADRCMP
 I2C_InitTypeDef, [58](#)
 AF0_GPIO
 lks32mc08x_gpio.h, [234](#)
 AF10_CAN
 lks32mc08x_gpio.h, [234](#)
 AF11_SIF
 lks32mc08x_gpio.h, [234](#)
 AF1_SYSAF
 lks32mc08x_gpio.h, [234](#)
 AF2_HALL
 lks32mc08x_gpio.h, [235](#)
 AF3_MCPWM
 lks32mc08x_gpio.h, [235](#)
 AF4_UART
 lks32mc08x_gpio.h, [235](#)
 AF5_SPI
 lks32mc08x_gpio.h, [235](#)
 AF6_I2C
 lks32mc08x_gpio.h, [235](#)
 AF7_TIMER01
 lks32mc08x_gpio.h, [235](#)
 AF8_TIMER23
 lks32mc08x_gpio.h, [236](#)
 AF9_DEBUG
 lks32mc08x_gpio.h, [236](#)
 AFE_REG5_RECORD
 WAKE_InitTypeDef, [112](#)
 AFE_REG6_RECORD
 WAKE_InitTypeDef, [112](#)
 ALC
 CAN_TypeDef, [28](#)
 Align
 ADC_InitTypeDef, [10](#)
 AMC_A0
 ADC_TypeDef, [17](#)
 AMC_A1
 ADC_TypeDef, [17](#)
 AMC_B0
 ADC_TypeDef, [17](#)
 AMC_B1
 ADC_TypeDef, [17](#)
 Arc_or_SinCFuction
 lks32mc08x_dsp.h, [208](#)
 ARCTAN
 lks32mc08x_dsp.h, [208](#)
 ArctanValue
 stru.CordicComponents, [91](#)
 ArctanX
 stru.CordicComponents, [91](#)
 ArctanY
 stru.CordicComponents, [91](#)
 BaudRate
 I2C_InitTypeDef, [58](#)
 SPI_InitTypeDef, [86](#)
 UART_InitTypeDef, [107](#)
 BCR
 I2C_TypeDef, [61](#)
 Bit9Value
 UART_InitTypeDef, [107](#)
 Bit_RESET
 lks32mc08x_gpio.h, [244](#)
 Bit_SET
 lks32mc08x_gpio.h, [244](#)
 BitAction
 lks32mc08x_gpio.h, [244](#)
 BLCWIN
 CMP_TypeDef, [42](#)
 BTR0
 CAN_TypeDef, [28](#)
 BTR1
 CAN_TypeDef, [28](#)
 BUFF
 UART_TypeDef, [111](#)
 BURST_ADDR_CMP
 I2C_InitTypeDef, [58](#)
 BURST_NACK
 I2C_InitTypeDef, [59](#)

BUS_ERR_IE
 I2C_InitTypeDef, 59
 ByteLength
 SPI_InitTypeDef, 86

 CAN_ACTIVE_ERROR_FLAG
 lks32mc08x_can.h, 152
 CAN_ALLSR
 lks32mc08x_can.h, 146
 CAN_BITERROR
 lks32mc08x_can.h, 151
 CAN_Btr0
 CAN_InitTypeDef, 26
 CAN_Btr1
 CAN_InitTypeDef, 26
 CAN_CONNFIRM
 lks32mc08x_can.h, 152
 CAN_CONNFIRM_DELIMITER
 lks32mc08x_can.h, 152
 CAN_CRC
 lks32mc08x_can.h, 152
 CAN_CRC_DELIMITER
 lks32mc08x_can.h, 152
 CAN_DATA_FIELD
 lks32mc08x_can.h, 152
 CAN_DATAAVAIL
 lks32mc08x_can.h, 146
 CAN_DATALENGTHCODE
 lks32mc08x_can.h, 152
 CAN_DMAEn
 CAN_InitTypeDef, 26
 CAN_ERROR_DELIMITER
 lks32mc08x_can.h, 152
 CAN_ERROV
 lks32mc08x_can.h, 146
 CAN_ErrWarThre
 CAN_InitTypeDef, 26
 CAN_FILLERROR
 lks32mc08x_can.h, 151
 CAN_FORMATERROR
 lks32mc08x_can.h, 151
 CAN_FRAMEEND
 lks32mc08x_can.h, 152
 CAN_GetIRQFlag
 lks32mc08x_can.c, 432
 lks32mc08x_can.h, 153
 CAN_ID15_ID12
 lks32mc08x_can.h, 152
 CAN_ID17_ID13
 lks32mc08x_can.h, 152
 CAN_ID20_ID18
 lks32mc08x_can.h, 152
 CAN_ID28_ID21
 lks32mc08x_can.h, 152
 CAN_ID4_ID0
 lks32mc08x_can.h, 152
 CAN_IDE
 lks32mc08x_can.h, 152
 CAN_IE_BUSERR
 lks32mc08x_can.h, 146
 CAN_IE_LOSTARB
 lks32mc08x_can.h, 147
 CAN_IE_PASSIVEERR
 lks32mc08x_can.h, 147
 CAN_IE_RFIFONEMPTY
 lks32mc08x_can.h, 147
 CAN_IE_RFIFOOV
 lks32mc08x_can.h, 147
 CAN_IE_TXDONE
 lks32mc08x_can.h, 147
 CAN_IE_WAKE
 lks32mc08x_can.h, 147
 CAN_IE_WERR
 lks32mc08x_can.h, 148
 CAN_IF_BUSERR
 lks32mc08x_can.h, 148
 CAN_IF_LOSTARB
 lks32mc08x_can.h, 148
 CAN_IF_PASSIVEERR
 lks32mc08x_can.h, 148
 CAN_IF_RFIFONEMPTY
 lks32mc08x_can.h, 148
 CAN_IF_RFIFOOV
 lks32mc08x_can.h, 148
 CAN_IF_TXDONE
 lks32mc08x_can.h, 149
 CAN_IF_WAKE
 lks32mc08x_can.h, 149
 CAN_IF_WERR
 lks32mc08x_can.h, 149
 CAN_Init
 lks32mc08x_can.c, 433
 lks32mc08x_can.h, 154
 CAN_InitTypeDef, 25
 CAN_Btr0, 26
 CAN_Btr1, 26
 CAN_DMAEn, 26
 CAN_ErrWarThre, 26
 CAN_WorkMode, 26
 IRQEna, 26
 CAN_INTERMISSION
 lks32mc08x_can.h, 152
 CAN_KEEPO
 lks32mc08x_can.h, 152
 CAN_KEEP1
 lks32mc08x_can.h, 152
 CAN_LOST_ID0
 lks32mc08x_can.h, 152
 CAN_LOST_ID1
 lks32mc08x_can.h, 152
 CAN_LOST_ID2
 lks32mc08x_can.h, 152
 CAN_LOST_ID3
 lks32mc08x_can.h, 153
 CAN_LOST_ID4
 lks32mc08x_can.h, 153
 CAN_LOST_ID5

- lks32mc08x_can.h, 153
- CAN_LOST_ID6
 - lks32mc08x_can.h, 153
- CAN_LOST_ID7
 - lks32mc08x_can.h, 153
- CAN_LOST_ID8
 - lks32mc08x_can.h, 153
- CAN_LOST_ID9
 - lks32mc08x_can.h, 153
- CAN_LOST_IDA
 - lks32mc08x_can.h, 153
- CAN_LOST_IDE
 - lks32mc08x_can.h, 153
- CAN_LOST_SRTR
 - lks32mc08x_can.h, 153
- CAN_Manual_Awake
 - lks32mc08x_can.c, 434
 - lks32mc08x_can.h, 155
- CAN_MONITOR_MODE
 - lks32mc08x_can.h, 149
- CAN_NORMAL_MODE
 - lks32mc08x_can.h, 149
- CAN_ONBUS
 - lks32mc08x_can.h, 149
- CAN_OTHERERROR
 - lks32mc08x_can.h, 151
- CAN_OVERLOAD_FLAG
 - lks32mc08x_can.h, 152
- CAN_PASSIVE_ERROR
 - lks32mc08x_can.h, 152
- CAN_ReadALC
 - lks32mc08x_can.c, 435
- CAN_ReadErrorCode
 - lks32mc08x_can.c, 436
- CAN_ReadRXERRCNT
 - lks32mc08x_can.c, 437
- CAN_ReadState
 - lks32mc08x_can.c, 438
 - lks32mc08x_can.h, 156
- CAN_ReadTXERRCNT
 - lks32mc08x_can.c, 440
- CAN_RECE_PROCESS
 - lks32mc08x_can.h, 151
- CAN_ReceFramInfor
 - lks32mc08x_can.c, 441
- CAN_Receive_Msg
 - lks32mc08x_can.c, 442
 - lks32mc08x_can.h, 158
- CAN_RFIFOEMPTY
 - lks32mc08x_can.h, 150
- CAN_RTR
 - lks32mc08x_can.h, 152
- CAN_RXING
 - lks32mc08x_can.h, 150
- CAN_SELFTEST_MODE
 - lks32mc08x_can.h, 150
- CAN_SEND_PROCESS
 - lks32mc08x_can.h, 151
- CAN_Sleep
 - lks32mc08x_can.c, 443
 - lks32mc08x_can.h, 158
- CAN_SRTR
 - lks32mc08x_can.h, 152
- CAN_START
 - lks32mc08x_can.h, 152
- CAN_StructInit
 - lks32mc08x_can.c, 444
 - lks32mc08x_can.h, 159
- CAN_TFIFOEMPTY
 - lks32mc08x_can.h, 150
- CAN_TOLERATE_DOMINANT
 - lks32mc08x_can.h, 152
- CAN_TRNASDONE
 - lks32mc08x_can.h, 150
- CAN_TXING
 - lks32mc08x_can.h, 151
- CAN_TypeDef, 27
 - ALC, 28
 - BTR0, 28
 - BTR1, 28
 - CMR, 29
 - ECC, 29
 - EIR, 29
 - EWLR, 29
 - IR, 29
 - MOD, 29
 - null0, 30
 - null1, 30
 - null2, 30
 - null3, 30
 - RBSA, 30
 - RMC, 30
 - RXERR, 30
 - SR, 31
 - TXERR, 31
 - TXRX0, 31
 - TXRX1, 31
 - TXRX2, 31
 - TXRX3, 31
 - TXRX4, 32
 - TXRX5, 32
 - TXRX6, 32
 - TXRX7, 32
 - TXRX8, 32
 - TXRX9, 32
 - TXRXA, 33
 - TXRXB, 33
 - TXRXC, 33
- CAN_WorkMode
 - CAN_InitTypeDef, 26
- Capture_IRQ_Ena
 - HALL_InitTypeDef, 54
- CENTRAL_PWM_MODE
 - lks32mc08x_mcpwm.h, 299
- CFG
 - ADC_TypeDef, 17

CMP_TypeDef, 42
 HALL_TypeDef, 56
 I2C_TypeDef, 61
 SPI_TypeDef, 89
 TIM_ECDTypeDef, 100
 TIM_TimerTypeDef, 106
 CH0_NS
 MCPWM_InitTypeDef, 67
 CH0_PS
 MCPWM_InitTypeDef, 67
 CH0N_default_output
 MCPWM_InitTypeDef, 67
 CH0N_Polarity_INV
 MCPWM_InitTypeDef, 67
 CH0N_SCTRLN
 MCPWM_InitTypeDef, 68
 CH0P_default_output
 MCPWM_InitTypeDef, 68
 CH0P_Polarity_INV
 MCPWM_InitTypeDef, 68
 CH0P_SCTRLP
 MCPWM_InitTypeDef, 68
 CH1_NS
 MCPWM_InitTypeDef, 68
 CH1_PS
 MCPWM_InitTypeDef, 69
 CH1N_default_output
 MCPWM_InitTypeDef, 69
 CH1N_Polarity_INV
 MCPWM_InitTypeDef, 69
 CH1N_SCTRLN
 MCPWM_InitTypeDef, 69
 CH1P_default_output
 MCPWM_InitTypeDef, 69
 CH1P_Polarity_INV
 MCPWM_InitTypeDef, 70
 CH1P_SCTRLP
 MCPWM_InitTypeDef, 70
 CH2_NS
 MCPWM_InitTypeDef, 70
 CH2_PS
 MCPWM_InitTypeDef, 70
 CH2N_default_output
 MCPWM_InitTypeDef, 70
 CH2N_Polarity_INV
 MCPWM_InitTypeDef, 71
 CH2N_SCTRLN
 MCPWM_InitTypeDef, 71
 CH2P_default_output
 MCPWM_InitTypeDef, 71
 CH2P_Polarity_INV
 MCPWM_InitTypeDef, 71
 CH2P_SCTRLP
 MCPWM_InitTypeDef, 71
 CH3_NS
 MCPWM_InitTypeDef, 72
 CH3_PS
 MCPWM_InitTypeDef, 72
 CH3N_default_output
 MCPWM_InitTypeDef, 72
 CH3N_Polarity_INV
 MCPWM_InitTypeDef, 72
 CH3N_SCTRLN
 MCPWM_InitTypeDef, 72
 CH3P_default_output
 MCPWM_InitTypeDef, 73
 CH3P_Polarity_INV
 MCPWM_InitTypeDef, 73
 CH3P_SCTRLP
 MCPWM_InitTypeDef, 73
 CHN0
 ADC_TypeDef, 18
 lks32mc08x_adc.h, 131
 CHN1
 ADC_TypeDef, 18
 lks32mc08x_adc.h, 131
 CHN10
 lks32mc08x_adc.h, 131
 CHN11
 lks32mc08x_adc.h, 131
 CHN12
 lks32mc08x_adc.h, 131
 CHN13
 lks32mc08x_adc.h, 131
 CHN14
 lks32mc08x_adc.h, 131
 CHN15
 lks32mc08x_adc.h, 131
 CHN16
 lks32mc08x_adc.h, 131
 CHN17
 lks32mc08x_adc.h, 131
 CHN18
 lks32mc08x_adc.h, 131
 CHN19
 lks32mc08x_adc.h, 131
 CHN2
 ADC_TypeDef, 18
 lks32mc08x_adc.h, 131
 CHN3
 ADC_TypeDef, 18
 lks32mc08x_adc.h, 131
 CHN4
 ADC_TypeDef, 18
 lks32mc08x_adc.h, 131
 CHN5
 ADC_TypeDef, 18
 lks32mc08x_adc.h, 131
 CHN6
 ADC_TypeDef, 19
 lks32mc08x_adc.h, 131
 CHN7
 ADC_TypeDef, 19
 lks32mc08x_adc.h, 131
 CHN8
 ADC_TypeDef, 19

lks32mc08x_adc.h, 131
 CHN9
 ADC_TypeDef, 19
 lks32mc08x_adc.h, 131
 CHNT0
 ADC_TypeDef, 19
 CHNT1
 ADC_TypeDef, 19
 CHNx
 lks32mc08x_adc.h, 131
 CLK10_EN
 CMP_InitTypeDef, 35
 CLK_CFG_RECORD
 WAKE_InitTypeDef, 112
 CLK_DIV
 MCPWM_InitTypeDef, 73
 Clk_DivSPI
 SYS_InitTypeDef, 95
 Clk_DivUART
 SYS_InitTypeDef, 95
 Clk_FEN
 SYS_InitTypeDef, 95
 Clk_Sel
 SYS_InitTypeDef, 96
 ClockDivision
 HALL_InitTypeDef, 54
 CMP0
 lks32mc08x_cmp.h, 171
 CMP0_CHN0P_WIN_EN
 CMP_InitTypeDef, 35
 CMP0_CHN1P_WIN_EN
 CMP_InitTypeDef, 36
 CMP0_CHN2P_WIN_EN
 CMP_InitTypeDef, 36
 CMP0_CHN3P_WIN_EN
 CMP_InitTypeDef, 36
 CMP0_EN
 CMP_InitTypeDef, 36
 CMP0_IE
 CMP_InitTypeDef, 36
 CMP0_IN_EN
 CMP_InitTypeDef, 36
 CMP0_IRQ_TRIG
 CMP_InitTypeDef, 37
 CMP0_POL
 CMP_InitTypeDef, 37
 CMP0_SELN
 CMP_InitTypeDef, 37
 CMP0_SEL_P
 CMP_InitTypeDef, 37
 CMP0_SEL_P_IP0
 lks32mc08x_cmp.h, 171
 CMP0_SEL_P_IP1
 lks32mc08x_cmp.h, 172
 CMP0_SEL_P_IP2
 lks32mc08x_cmp.h, 172
 CMP0_SEL_P_IP3
 lks32mc08x_cmp.h, 172
 CMP0_SEL_P_IP4
 lks32mc08x_cmp.h, 172
 CMP0_SEL_P_OPA0_IP
 lks32mc08x_cmp.h, 172
 CMP0_SEL_P_OPA0_OUT
 lks32mc08x_cmp.h, 172
 CMP0_SEL_P_OPA1_OUT
 lks32mc08x_cmp.h, 173
 CMP0_W_PWM_POL
 CMP_InitTypeDef, 38
 CMP1
 lks32mc08x_cmp.h, 173
 CMP1_CHN0P_WIN_EN
 CMP_InitTypeDef, 38
 CMP1_CHN1P_WIN_EN
 CMP_InitTypeDef, 38
 CMP1_CHN2P_WIN_EN
 CMP_InitTypeDef, 38
 CMP1_CHN3P_WIN_EN
 CMP_InitTypeDef, 39
 CMP1_EN
 CMP_InitTypeDef, 39
 CMP1_IE
 CMP_InitTypeDef, 39
 CMP1_IN_EN
 CMP_InitTypeDef, 39
 CMP1_IRQ_TRIG
 CMP_InitTypeDef, 39
 CMP1_POL
 CMP_InitTypeDef, 39
 CMP1_SELN
 CMP_InitTypeDef, 40
 CMP1_SEL_P
 CMP_InitTypeDef, 40
 CMP1_SEL_P_AVSS
 lks32mc08x_cmp.h, 173
 CMP1_SEL_P_IP0
 lks32mc08x_cmp.h, 173
 CMP1_SEL_P_IP1
 lks32mc08x_cmp.h, 173
 CMP1_SEL_P_IP2
 lks32mc08x_cmp.h, 173
 CMP1_SEL_P_IP3
 lks32mc08x_cmp.h, 174
 CMP1_SEL_P_OPA2_OUT
 lks32mc08x_cmp.h, 174
 CMP1_SEL_P_OPA3_IP
 lks32mc08x_cmp.h, 174
 CMP1_SEL_P_OPA3_OUT
 lks32mc08x_cmp.h, 174
 CMP1_W_PWM_POL
 CMP_InitTypeDef, 40
 CMP_BKIN_Filter
 MCPWM_InitTypeDef, 73
 CMP_ClearIRQFlag
 lks32mc08x_cmp.c, 456
 lks32mc08x_cmp.h, 178
 CMP_FT

CMP_InitTypeDef, 40
 CMP_GetIRQFlag
 lks32mc08x_cmp.c, 457
 lks32mc08x_cmp.h, 179
 CMP_HYS
 CMP_InitTypeDef, 41
 CMP_HYS_0mV
 lks32mc08x_cmp.h, 174
 CMP_HYS_20mV
 lks32mc08x_cmp.h, 174
 CMP_Init
 lks32mc08x_cmp.c, 458
 lks32mc08x_cmp.h, 180
 CMP_InitTypeDef, 33
 CLK10_EN, 35
 CMP0_CHN0P_WIN_EN, 35
 CMP0_CHN1P_WIN_EN, 36
 CMP0_CHN2P_WIN_EN, 36
 CMP0_CHN3P_WIN_EN, 36
 CMP0_EN, 36
 CMP0_IE, 36
 CMP0_IN_EN, 36
 CMP0_IRQ_TRIG, 37
 CMP0_POL, 37
 CMP0_SELN, 37
 CMP0_SELP, 37
 CMP0_W_PWM_POL, 38
 CMP1_CHN0P_WIN_EN, 38
 CMP1_CHN1P_WIN_EN, 38
 CMP1_CHN2P_WIN_EN, 38
 CMP1_CHN3P_WIN_EN, 39
 CMP1_EN, 39
 CMP1_IE, 39
 CMP1_IN_EN, 39
 CMP1_IRQ_TRIG, 39
 CMP1_POL, 39
 CMP1_SELN, 40
 CMP1_SELP, 40
 CMP1_W_PWM_POL, 40
 CMP_FT, 40
 CMP_HYS, 41
 FIL_CLK10_DIV16, 41
 FIL_CLK10_DIV2, 41
 CMP_SELN_DAC
 lks32mc08x_cmp.h, 175
 CMP_SELN_HALL_MID
 lks32mc08x_cmp.h, 175
 CMP_SELN_IN
 lks32mc08x_cmp.h, 175
 CMP_SELN_REF
 lks32mc08x_cmp.h, 175
 CMP_StructInit
 lks32mc08x_cmp.c, 460
 lks32mc08x_cmp.h, 181
 CMP_TypeDef, 42
 BLCWIN, 42
 CFG, 42
 DATA, 42
 IE, 43
 IF, 43
 TCLK, 43
 CMPT0
 TIM_TimerTypeDef, 106
 CMPT1
 TIM_TimerTypeDef, 106
 CMR
 CAN_TypeDef, 29
 CNT
 HALL_TypeDef, 56
 TIM_ECDTypeDef, 100
 TIM_TimerTypeDef, 106
 Con_Sample
 ADC_InitTypeDef, 11
 CosValue
 stru_CordicComponents, 91
 CountTH
 HALL_InitTypeDef, 55
 CPHA
 SPI_InitTypeDef, 87
 CPOL
 SPI_InitTypeDef, 87
 CR0
 EXTI_TypeDef, 49
 CR1
 EXTI_TypeDef, 49
 CTRL
 UART_TypeDef, 111
 DAC_GAIN
 DAC_InitTypeDef, 44
 DAC_Init
 lks32mc08x_dac.c, 463
 lks32mc08x_dac.h, 186
 DAC_InitTypeDef, 43
 DAC_GAIN, 44
 DACOUT_EN, 44
 DAC_Output
 lks32mc08x_dac.c, 464
 lks32mc08x_dac.h, 187
 DAC_OutputValue
 lks32mc08x_dac.c, 464
 lks32mc08x_dac.h, 187
 DAC_OutputVoltage
 lks32mc08x_dac.c, 465
 lks32mc08x_dac.h, 188
 DAC_RANGE_1V2
 lks32mc08x_dac.h, 185
 DAC_RANGE_3V0
 lks32mc08x_dac.h, 185
 DAC_RANGE_4V85
 lks32mc08x_dac.h, 185
 DAC_StructInit
 lks32mc08x_dac.c, 466
 lks32mc08x_dac.h, 189
 DACOUT_EN
 DAC_InitTypeDef, 44
 DAT0

ADC_TypeDef, 20
 lks32mc08x_adc.h, 132
 DAT0_TH
 ADC_InitTypeDef, 11
 ADC_TypeDef, 20
 DAT0_TH_Type
 ADC_InitTypeDef, 11
 DAT1
 ADC_TypeDef, 20
 lks32mc08x_adc.h, 132
 DAT10
 ADC_TypeDef, 20
 lks32mc08x_adc.h, 132
 DAT11
 ADC_TypeDef, 20
 lks32mc08x_adc.h, 132
 DAT12
 ADC_TypeDef, 20
 lks32mc08x_adc.h, 132
 DAT13
 ADC_TypeDef, 21
 lks32mc08x_adc.h, 132
 DAT14
 ADC_TypeDef, 21
 lks32mc08x_adc.h, 132
 DAT15
 ADC_TypeDef, 21
 lks32mc08x_adc.h, 132
 DAT16
 ADC_TypeDef, 21
 lks32mc08x_adc.h, 132
 DAT17
 ADC_TypeDef, 21
 lks32mc08x_adc.h, 132
 DAT18
 ADC_TypeDef, 21
 lks32mc08x_adc.h, 132
 DAT19
 ADC_TypeDef, 22
 lks32mc08x_adc.h, 132
 DAT2
 ADC_TypeDef, 22
 lks32mc08x_adc.h, 132
 DAT3
 ADC_TypeDef, 22
 lks32mc08x_adc.h, 132
 DAT4
 ADC_TypeDef, 22
 lks32mc08x_adc.h, 132
 DAT5
 ADC_TypeDef, 22
 lks32mc08x_adc.h, 132
 DAT6
 ADC_TypeDef, 22
 lks32mc08x_adc.h, 132
 DAT7
 ADC_TypeDef, 23
 lks32mc08x_adc.h, 132
 DAT8
 ADC_TypeDef, 23
 lks32mc08x_adc.h, 132
 DAT9
 ADC_TypeDef, 23
 lks32mc08x_adc.h, 132
 DATA
 CMP_TypeDef, 42
 I2C_TypeDef, 61
 DataOrder
 SPI_InitTypeDef, 87
 DATx
 lks32mc08x_adc.h, 131
 DC_A0
 ADC_TypeDef, 23
 DC_A1
 ADC_TypeDef, 23
 DC_B0
 ADC_TypeDef, 23
 DC_B1
 ADC_TypeDef, 24
 DeadTimeCH0N
 MCPWM_InitTypeDef, 74
 DeadTimeCH0P
 MCPWM_InitTypeDef, 74
 DeadTimeCH1N
 MCPWM_InitTypeDef, 74
 DeadTimeCH1P
 MCPWM_InitTypeDef, 74
 DeadTimeCH2N
 MCPWM_InitTypeDef, 74
 DeadTimeCH2P
 MCPWM_InitTypeDef, 75
 DeadTimeCH3N
 MCPWM_InitTypeDef, 75
 DeadTimeCH3P
 MCPWM_InitTypeDef, 75
 DebugMode_PWM_out
 MCPWM_InitTypeDef, 75
 DIV
 SPI_TypeDef, 89
 DIVH
 UART_TypeDef, 111
 Dividend
 stru_DiviComponents, 93
 Divisor
 stru_DiviComponents, 93
 DIVL
 UART_TypeDef, 111
 DMA
 I2C_InitTypeDef, 59
 SPI_InitTypeDef, 87
 DMA_BYTE_TRANS
 lks32mc08x_dma.h, 193
 DMA_CCR
 DMA_RegTypeDef, 48
 DMA_CH0_ADC0_REQ_EN
 lks32mc08x_dma.h, 193

DMA_CH0_EIF
 lks32mc08x_dma.h, 193
 DMA_CH0_FIF
 lks32mc08x_dma.h, 193
 DMA_CH0_TIMER0_REQ_EN
 lks32mc08x_dma.h, 193
 DMA_CH0_TIMER1_REQ_EN
 lks32mc08x_dma.h, 193
 DMA_CH1_EIF
 lks32mc08x_dma.h, 193
 DMA_CH1_FIF
 lks32mc08x_dma.h, 194
 DMA_CH1_MCPWM_REQ_EN
 lks32mc08x_dma.h, 194
 DMA_CH1_SPI_RX_REQ_EN
 lks32mc08x_dma.h, 194
 DMA_CH1_TIMER2_REQ_EN
 lks32mc08x_dma.h, 194
 DMA_CH2_EIF
 lks32mc08x_dma.h, 194
 DMA_CH2_FIF
 lks32mc08x_dma.h, 194
 DMA_CH2_SPI_TX_REQ_EN
 lks32mc08x_dma.h, 195
 DMA_CH2_TIMER3_REQ_EN
 lks32mc08x_dma.h, 195
 DMA_CH2_UART0_REQ_EN
 lks32mc08x_dma.h, 195
 DMA_CH3_CAN_REQ_EN
 lks32mc08x_dma.h, 195
 DMA_CH3_EIF
 lks32mc08x_dma.h, 195
 DMA_CH3_FIF
 lks32mc08x_dma.h, 195
 DMA_CH3_I2C_REQ_EN
 lks32mc08x_dma.h, 196
 DMA_CH3_UART1_REQ_EN
 lks32mc08x_dma.h, 196
 DMA_CHx_EN
 lks32mc08x_dma.c, 470
 lks32mc08x_dma.h, 197
 DMA_CIRC
 DMA_InitTypeDef, 45
 DMA_ClearIRQFlag
 lks32mc08x_dma.c, 470
 lks32mc08x_dma.h, 198
 DMA_CMAR
 DMA_InitTypeDef, 45
 DMA_RegTypeDef, 48
 DMA_CPAR
 DMA_InitTypeDef, 45
 DMA_RegTypeDef, 48
 DMA_CTMS
 DMA_RegTypeDef, 49
 DMA_DIR
 DMA_InitTypeDef, 45
 DMA_GetIRQFlag
 lks32mc08x_dma.c, 472
 lks32mc08x_dma.h, 199
 DMA_HALFWORD_TRANS
 lks32mc08x_dma.h, 196
 DMA_Init
 lks32mc08x_dma.c, 473
 lks32mc08x_dma.h, 201
 DMA_InitTypeDef, 44
 DMA_CIRC, 45
 DMA_CMAR, 45
 DMA_CPAR, 45
 DMA_DIR, 45
 DMA_IRQ_EN, 45
 DMA_MBTW, 46
 DMA_MINC, 46
 DMA_PBTW, 46
 DMA_PINC, 46
 DMA_REQ_EN, 47
 DMA_ROUND, 47
 DMA_TIMES, 47
 DMA_IRQ_EN
 DMA_InitTypeDef, 45
 DMA_MBTW
 DMA_InitTypeDef, 46
 DMA_MINC
 DMA_InitTypeDef, 46
 DMA_PBTW
 DMA_InitTypeDef, 46
 DMA_PINC
 DMA_InitTypeDef, 46
 DMA_RegTypeDef, 48
 DMA_CCR, 48
 DMA_CMAR, 48
 DMA_CPAR, 48
 DMA_CTMS, 49
 DMA_REQ_EN
 DMA_InitTypeDef, 47
 DMA_ROUND
 DMA_InitTypeDef, 47
 DMA_StructInit
 lks32mc08x_dma.c, 475
 lks32mc08x_dma.h, 202
 DMA_TCIE
 lks32mc08x_dma.h, 196
 DMA_TEIE
 lks32mc08x_dma.h, 196
 DMA_TIMES
 DMA_InitTypeDef, 47
 DMA_WORD_TRANS
 lks32mc08x_dma.h, 196
 DSP_arctan
 lks32mc08x_dsp.c, 478
 lks32mc08x_dsp.h, 208
 DSP_CALC_ARCTAN
 lks32mc08x_dsp.h, 206
 DSP_CALC_COS
 lks32mc08x_dsp.h, 206
 DSP_CALC_DIV
 lks32mc08x_dsp.h, 206

DSP_CALC_MOD
 lks32mc08x_dsp.h, 207
 DSP_CALC_RMS
 lks32mc08x_dsp.h, 207
 DSP_CALC_SIN
 lks32mc08x_dsp.h, 207
 DSP_CALC_SQRT
 lks32mc08x_dsp.h, 208
 DSP_CalcCordic
 lks32mc08x_dsp.c, 478
 lks32mc08x_dsp.h, 208
 DSP_CalcDivision
 lks32mc08x_dsp.c, 480
 lks32mc08x_dsp.h, 210
 DSP_CalcSqrt
 lks32mc08x_dsp.c, 481
 lks32mc08x_dsp.h, 211
 DSP_Cmd
 lks32mc08x_dsp.c, 482
 lks32mc08x_dsp.h, 212
 DSP_cos
 lks32mc08x_dsp.c, 483
 lks32mc08x_dsp.h, 213
 DSP_div
 lks32mc08x_dsp.c, 483
 lks32mc08x_dsp.h, 213
 DSP_GetSqrt
 lks32mc08x_dsp.c, 483
 lks32mc08x_dsp.h, 214
 DSP_Init
 lks32mc08x_dsp.c, 485
 lks32mc08x_dsp.h, 215
 DSP_mod
 lks32mc08x_dsp.c, 485
 lks32mc08x_dsp.h, 215
 DSP_rms
 lks32mc08x_dsp.c, 485
 lks32mc08x_dsp.h, 215
 DSP_sin
 lks32mc08x_dsp.c, 485
 lks32mc08x_dsp.h, 215
 DSP_sqrt
 lks32mc08x_dsp.c, 485
 lks32mc08x_dsp.h, 216
 Duplex
 SPI_InitTypeDef, 87
 ECC
 CAN_TypeDef, 29
 ECD_ClearIRQFlag
 lks32mc08x_tim.c, 605
 lks32mc08x_tim.h, 383
 ECD_Clk_Div1
 lks32mc08x_tim.h, 378
 ECD_Clk_Div2
 lks32mc08x_tim.h, 378
 ECD_Clk_Div4
 lks32mc08x_tim.h, 378
 ECD_Clk_Div8
 lks32mc08x_tim.h, 379
 ECD_ClockDiv
 TIM_ECDInitTypeDef, 98
 ECD_Filter0
 TIM_ECDInitTypeDef, 98
 ECD_Filter1
 TIM_ECDInitTypeDef, 98
 ECD_GetIRQFlag
 lks32mc08x_tim.c, 606
 lks32mc08x_tim.h, 384
 ECD_IF_OVERFLOW
 lks32mc08x_tim.h, 379
 ECD_IF_UNDERFLOW
 lks32mc08x_tim.h, 379
 ECD_IRQEna
 TIM_ECDInitTypeDef, 98
 ECD_IRQEna_None
 lks32mc08x_tim.h, 379
 ECD_IRQEna_OverFlow
 lks32mc08x_tim.h, 379
 ECD_IRQEna_UnderFlow
 lks32mc08x_tim.h, 379
 ECD_Mode
 TIM_ECDInitTypeDef, 98
 ECD_Mode_CCWCW
 lks32mc08x_tim.h, 380
 ECD_Mode_CCWCW_UpDown
 lks32mc08x_tim.h, 380
 ECD_Mode_CCWSIGN
 lks32mc08x_tim.h, 380
 ECD_Mode_CCWSIGN_UpDown
 lks32mc08x_tim.h, 380
 ECD_Mode_T1
 lks32mc08x_tim.h, 380
 ECD_Mode_T1_T2
 lks32mc08x_tim.h, 380
 ECD_TH
 TIM_ECDInitTypeDef, 99
 EDGE_PWM_MODE
 lks32mc08x_mcpwm.h, 299
 EFF_ID_Filter
 lks32mc08x_can.c, 445
 lks32mc08x_can.h, 160
 EIR
 CAN_TypeDef, 29
 EN
 EXTI_TypeDef, 50
 SPI_InitTypeDef, 87
 TIM_TimerInitTypeDef, 101
 enumErrortime
 lks32mc08x_can.h, 151
 enumErrortype
 lks32mc08x_can.h, 151
 enumErrposition
 lks32mc08x_can.h, 151
 enumLossposition
 lks32mc08x_can.h, 152
 enumOPA

lks32mc08x_opa.h, 326
 enumWakeTrigLevel
 lks32mc08x_sys.h, 355
 erase_flag
 lks32mc08x_flash.c, 499
 lks32mc08x_flash.h, 230
 EraseSector
 lks32mc08x_flash.c, 494
 lks32mc08x_flash.h, 225
 EVT
 TIM_TimerTypeDef, 106
 EWLR
 CAN_TypeDef, 29
 EXTI
 lks32mc08x_exti.h, 219
 EXTI_ClearIRQFlag
 lks32mc08x_exti.c, 489
 lks32mc08x_exti.h, 220
 EXTI_GetIRQFlag
 lks32mc08x_exti.c, 490
 lks32mc08x_exti.h, 221
 EXTI_Trigger_Config
 lks32mc08x_exti.c, 491
 lks32mc08x_exti.h, 222
 EXTI_Trigger_Edge
 lks32mc08x_exti.h, 219
 EXTI_Trigger_Negedge
 lks32mc08x_exti.h, 219
 EXTI_Trigger_None
 lks32mc08x_exti.h, 219
 EXTI_Trigger_Posedge
 lks32mc08x_exti.h, 219
 EXTI_TypeDef, 49
 CR0, 49
 CR1, 49
 EN, 50
 IF, 50
 POL, 50
 EXTI_Mode_TypeDef
 lks32mc08x_exti.h, 219
 F3210
 GPIO_TypeDef, 52
 F7654
 GPIO_TypeDef, 52
 FAIL0_INPUT_EN
 MCPWM_InitTypeDef, 76
 FAIL0_INT_EN
 MCPWM_InitTypeDef, 76
 FAIL0_Polarity
 MCPWM_InitTypeDef, 76
 FAIL0_Signal_Sel
 MCPWM_InitTypeDef, 76
 FAIL1_INPUT_EN
 MCPWM_InitTypeDef, 76
 FAIL1_INT_EN
 MCPWM_InitTypeDef, 77
 FAIL1_Polarity
 MCPWM_InitTypeDef, 77
 FAIL1_Signal_Sel
 MCPWM_InitTypeDef, 77
 FAIL_SEL_CMP
 lks32mc08x_mcpwm.h, 300
 FAIL_SEL_IO
 lks32mc08x_mcpwm.h, 300
 FBA98
 GPIO_TypeDef, 52
 FFEDC
 GPIO_TypeDef, 52
 FIL_CLK10_DIV16
 CMP_InitTypeDef, 41
 FIL_CLK10_DIV2
 CMP_InitTypeDef, 41
 Filter75_Ena
 HALL_InitTypeDef, 55
 FilterLen
 HALL_InitTypeDef, 55
 FirSeg_Ch
 ADC_InitTypeDef, 11
 FirstSend
 UART_InitTypeDef, 108
 Flash_MAIN
 lks32mc08x_flash.h, 225
 Flash_NVR
 lks32mc08x_flash.h, 225
 FouSeg_Ch
 ADC_InitTypeDef, 11
 Gain0
 ADC_InitTypeDef, 12
 Gain1
 ADC_InitTypeDef, 12
 GAIN_0
 ADC_TypeDef, 24
 GAIN_1
 ADC_TypeDef, 24
 GetCurrentTemperature
 lks32mc08x_tmp.c, 625
 lks32mc08x_tmp.h, 401
 GPIO0
 lks32mc08x_gpio.h, 236
 GPIO1
 lks32mc08x_gpio.h, 236
 GPIO2
 lks32mc08x_gpio.h, 236
 GPIO3
 lks32mc08x_gpio.h, 236
 GPIO_AF_CAN
 lks32mc08x_gpio.h, 237
 GPIO_AF_DEBUG
 lks32mc08x_gpio.h, 237
 GPIO_AF_GPIO
 lks32mc08x_gpio.h, 237
 GPIO_AF_HALL
 lks32mc08x_gpio.h, 237
 GPIO_AF_I2C
 lks32mc08x_gpio.h, 237
 GPIO_AF_MCPWM

lks32mc08x_gpio.h, 237

GPIO_AF_SIF
lks32mc08x_gpio.h, 237

GPIO_AF_SPI
lks32mc08x_gpio.h, 238

GPIO_AF_SYSAF
lks32mc08x_gpio.h, 238

GPIO_AF_TIMER0
lks32mc08x_gpio.h, 238

GPIO_AF_TIMER1
lks32mc08x_gpio.h, 238

GPIO_AF_TIMER2
lks32mc08x_gpio.h, 238

GPIO_AF_TIMER3
lks32mc08x_gpio.h, 238

GPIO_AF_UART
lks32mc08x_gpio.h, 238

GPIO_BKIN_Filter
MCPWM_InitTypeDef, 77

GPIO_Config
lks32mc08x_gpio.c, 503
lks32mc08x_gpio.h, 245

GPIO_Init
lks32mc08x_gpio.c, 504
lks32mc08x_gpio.h, 246

GPIO_InitTypeDef, 50
GPIO_Mode, 51
GPIO_Pin, 51
GPIO_PODena, 51
GPIO_PuPd, 51

GPIO_LockRegister
lks32mc08x_gpio.c, 505
lks32mc08x_gpio.h, 247

GPIO_Mode
GPIO_InitTypeDef, 51

GPIO_Mode_ANA
lks32mc08x_gpio.h, 245

GPIO_Mode_IN
lks32mc08x_gpio.h, 245

GPIO_Mode_IO
lks32mc08x_gpio.h, 245

GPIO_Mode_OUT
lks32mc08x_gpio.h, 245

GPIO_Pin
GPIO_InitTypeDef, 51

GPIO_Pin_0
lks32mc08x_gpio.h, 238

GPIO_Pin_1
lks32mc08x_gpio.h, 239

GPIO_Pin_10
lks32mc08x_gpio.h, 239

GPIO_Pin_11
lks32mc08x_gpio.h, 239

GPIO_Pin_12
lks32mc08x_gpio.h, 239

GPIO_Pin_13
lks32mc08x_gpio.h, 239

GPIO_Pin_14
lks32mc08x_gpio.h, 239

GPIO_Pin_15
lks32mc08x_gpio.h, 240

GPIO_Pin_2
lks32mc08x_gpio.h, 240

GPIO_Pin_3
lks32mc08x_gpio.h, 240

GPIO_Pin_4
lks32mc08x_gpio.h, 240

GPIO_Pin_5
lks32mc08x_gpio.h, 240

GPIO_Pin_6
lks32mc08x_gpio.h, 240

GPIO_Pin_7
lks32mc08x_gpio.h, 241

GPIO_Pin_8
lks32mc08x_gpio.h, 241

GPIO_Pin_9
lks32mc08x_gpio.h, 241

GPIO_Pin_ALL
lks32mc08x_gpio.h, 241

GPIO_Pin_NONE
lks32mc08x_gpio.h, 241

GPIO_PinAFConfig
lks32mc08x_gpio.c, 506
lks32mc08x_gpio.h, 248

GPIO_PinSource_0
lks32mc08x_gpio.h, 241

GPIO_PinSource_1
lks32mc08x_gpio.h, 242

GPIO_PinSource_10
lks32mc08x_gpio.h, 242

GPIO_PinSource_11
lks32mc08x_gpio.h, 242

GPIO_PinSource_12
lks32mc08x_gpio.h, 242

GPIO_PinSource_13
lks32mc08x_gpio.h, 242

GPIO_PinSource_14
lks32mc08x_gpio.h, 242

GPIO_PinSource_15
lks32mc08x_gpio.h, 243

GPIO_PinSource_2
lks32mc08x_gpio.h, 243

GPIO_PinSource_3
lks32mc08x_gpio.h, 243

GPIO_PinSource_4
lks32mc08x_gpio.h, 243

GPIO_PinSource_5
lks32mc08x_gpio.h, 243

GPIO_PinSource_6
lks32mc08x_gpio.h, 243

GPIO_PinSource_7
lks32mc08x_gpio.h, 244

GPIO_PinSource_8
lks32mc08x_gpio.h, 244

GPIO_PinSource_9
lks32mc08x_gpio.h, 244

- GPIO_PODEna
 - GPIO_InitTypeDef, 51
- GPIO_PuPd
 - GPIO_InitTypeDef, 51
- GPIO_PuPd_NOPULL
 - lks32mc08x_gpio.h, 245
- GPIO_PuPd_UP
 - lks32mc08x_gpio.h, 245
- GPIO_ReadInputData
 - lks32mc08x_gpio.c, 508
 - lks32mc08x_gpio.h, 250
- GPIO_ReadInputDataBit
 - lks32mc08x_gpio.c, 509
 - lks32mc08x_gpio.h, 251
- GPIO_ReadOutputData
 - lks32mc08x_gpio.c, 510
 - lks32mc08x_gpio.h, 252
- GPIO_ReadOutputDataBit
 - lks32mc08x_gpio.c, 511
 - lks32mc08x_gpio.h, 253
- GPIO_ReadPinBit
 - lks32mc08x_gpio.c, 512
 - lks32mc08x_gpio.h, 254
- GPIO_ResetBits
 - lks32mc08x_gpio.c, 512
 - lks32mc08x_gpio.h, 254
- GPIO_SetBits
 - lks32mc08x_gpio.c, 513
 - lks32mc08x_gpio.h, 255
- GPIO_StructInit
 - lks32mc08x_gpio.c, 514
 - lks32mc08x_gpio.h, 256
- GPIO_TypeDef, 51
 - F3210, 52
 - F7654, 52
 - FBA98, 52
 - FFEDC, 52
 - LCKR, 52
 - PDE, 52
 - PDI, 53
 - PDO, 53
 - PIE, 53
 - PODE, 53
 - POE, 53
 - PUE, 53
- GPIO_UnlockRegister
 - lks32mc08x_gpio.c, 515
 - lks32mc08x_gpio.h, 257
- GPIO_Write
 - lks32mc08x_gpio.c, 516
 - lks32mc08x_gpio.h, 258
- GPIO_WriteBit
 - lks32mc08x_gpio.c, 517
 - lks32mc08x_gpio.h, 259
- GPIO_WritePinBit
 - lks32mc08x_gpio.c, 519
 - lks32mc08x_gpio.h, 261
- GPIO_Mode_TypeDef
 - lks32mc08x_gpio.h, 244
- GPIO_PuPd_TypeDef
 - lks32mc08x_gpio.h, 245
- HALL_CAPTURE_EVENT
 - lks32mc08x_hall.h, 265
- HALL_ClearIRQFlag
 - lks32mc08x_hall.c, 524
 - lks32mc08x_hall.h, 266
- HALL_CLK_DIV1
 - lks32mc08x_hall.h, 265
- HALL_CLK_DIV2
 - lks32mc08x_hall.h, 265
- HALL_CLK_DIV4
 - lks32mc08x_hall.h, 265
- HALL_CLK_DIV8
 - lks32mc08x_hall.h, 265
- HALL_Ena
 - HALL_InitTypeDef, 55
- HALL_GetCaptureValue
 - lks32mc08x_hall.c, 525
 - lks32mc08x_hall.h, 267
- HALL_GetCount
 - lks32mc08x_hall.c, 526
 - lks32mc08x_hall.h, 268
- HALL_GetFilterValue
 - lks32mc08x_hall.c, 527
 - lks32mc08x_hall.h, 269
- HALL_GetIRQFlag
 - lks32mc08x_hall.c, 528
 - lks32mc08x_hall.h, 269
- HALL_Init
 - lks32mc08x_hall.c, 529
 - lks32mc08x_hall.h, 270
- HALL_InitTypeDef, 54
 - Capture_IRQ_Ena, 54
 - ClockDivision, 54
 - CountTH, 55
 - Filter75_Ena, 55
 - FilterLen, 55
 - HALL_Ena, 55
 - Overflow_IRQ_Ena, 55
 - softIE, 55
- HALL_OVERFLOW_EVENT
 - lks32mc08x_hall.h, 265
- HALL_StructInit
 - lks32mc08x_hall.c, 530
 - lks32mc08x_hall.h, 272
- HALL_TypeDef, 56
 - CFG, 56
 - CNT, 56
 - INFO, 57
 - TH, 57
 - WIDTH, 57
- HALL_WIDCount
 - lks32mc08x_hall.c, 531
 - lks32mc08x_hall.h, 273
- HIGH_LEVEL
 - lks32mc08x_mcpwm.h, 300

- HIGH_LEVEL_ACTIVE
 - lks32mc08x_mcpwm.h, 300
- i2c_dma_delay_over
 - lks32mc08x_i2c.c, 534
- i2c_dma_init
 - lks32mc08x_i2c.c, 535
 - lks32mc08x_i2c.h, 276
- i2c_dma_rx
 - lks32mc08x_i2c.c, 537
 - lks32mc08x_i2c.h, 277
- i2c_dma_state
 - lks32mc08x_i2c.c, 546
- i2c_dma_state_over
 - lks32mc08x_i2c.c, 538
 - lks32mc08x_i2c.h, 278
- i2c_dma_state_start
 - lks32mc08x_i2c.c, 539
- i2c_dma_tx
 - lks32mc08x_i2c.c, 540
 - lks32mc08x_i2c.h, 279
- I2C_Init
 - lks32mc08x_i2c.c, 541
 - lks32mc08x_i2c.h, 281
- I2C_InitTypeDef, 57
 - ADRCMP, 58
 - BaudRate, 58
 - BURST_ADDR_CMP, 58
 - BURST_NACK, 59
 - BUS_ERR_IE, 59
 - DMA, 59
 - IE, 59
 - MST_MODE, 59
 - SLV_MODE, 59
 - STOP_IE, 60
 - TC_IE, 60
- I2C_ReadData
 - lks32mc08x_i2c.c, 543
 - lks32mc08x_i2c.h, 282
- I2C_SendData
 - lks32mc08x_i2c.c, 544
 - lks32mc08x_i2c.h, 283
- I2C_StructInit
 - lks32mc08x_i2c.c, 545
 - lks32mc08x_i2c.h, 284
- I2C_TypeDef, 60
 - ADDR, 61
 - BCR, 61
 - CFG, 61
 - DATA, 61
 - MSCR, 61
 - SCR, 61
- ID1_Filter_Dual
 - lks32mc08x_can.c, 446
 - lks32mc08x_can.h, 161
- ID2_Filter_Dual
 - lks32mc08x_can.c, 447
 - lks32mc08x_can.h, 162
- IE
 - ADC_InitTypeDef, 12
 - ADC_TypeDef, 24
 - CMP_TypeDef, 43
 - I2C_InitTypeDef, 59
 - SPI_TypeDef, 90
 - UART_TypeDef, 111
- IF
 - ADC_TypeDef, 24
 - CMP_TypeDef, 43
 - EXTI_TypeDef, 50
 - UART_TypeDef, 111
- include/lks32mc08x_adc.h, 113, 140
- include/lks32mc08x_can.h, 143, 166
- include/lks32mc08x_cmp.h, 168, 182
- include/lks32mc08x_dac.h, 184, 190
- include/lks32mc08x_dma.h, 190, 203
- include/lks32mc08x_dsp.h, 204, 216
- include/lks32mc08x_exti.h, 217, 223
- include/lks32mc08x_flash.h, 223, 230
- include/lks32mc08x_gpio.h, 231, 261
- include/lks32mc08x_hall.h, 263, 274
- include/lks32mc08x_i2c.h, 274, 285
- include/lks32mc08x_iwdg.h, 286, 293
- include/lks32mc08x_lib.h, 293, 294
- include/lks32mc08x_mcpwm.h, 295, 319
- include/lks32mc08x_nvr.h, 321, 324
- include/lks32mc08x_opa.h, 324, 330
- include/lks32mc08x_spi.h, 330, 341
- include/lks32mc08x_sys.h, 342, 373
- include/lks32mc08x_tim.h, 375, 398
- include/lks32mc08x_tmp.h, 400, 403
- include/lks32mc08x_uart.h, 404, 418
- INFO
 - HALL_TypeDef, 57
- INV
 - UART_TypeDef, 112
- IO0_PPE
 - MCPWM_InitTypeDef, 77
- IO1_PPE
 - MCPWM_InitTypeDef, 78
- IO2_PPE
 - MCPWM_InitTypeDef, 78
- IO3_PPE
 - MCPWM_InitTypeDef, 78
- IR
 - CAN_TypeDef, 29
- IRQEna
 - CAN_InitTypeDef, 26
 - SPI_InitTypeDef, 88
 - UART_InitTypeDef, 108
- IWDG_DISABLE
 - lks32mc08x_iwdg.c, 549
 - lks32mc08x_iwdg.h, 288
- IWDG_ENABLE
 - lks32mc08x_iwdg.c, 550
 - lks32mc08x_iwdg.h, 289
- IWDG_Feed
 - lks32mc08x_iwdg.c, 551

- lks32mc08x_iwdg.h, 290
- IWDG_Init
 - lks32mc08x_iwdg.c, 552
 - lks32mc08x_iwdg.h, 291
- IWDG_InitTypeDef, 61
 - RTH, 62
 - WDG_EN, 62
- IWDG_StructInit
 - lks32mc08x_iwdg.c, 553
 - lks32mc08x_iwdg.h, 292
- LCKR
 - GPIO_TypeDef, 52
- lks32mc08x_adc.c
 - ADC_ChannelConfig, 420
 - ADC_ClearIRQFlag, 421
 - ADC_GetConversionValue, 423
 - ADC_GetIRQFlag, 424
 - ADC_Init, 425
 - ADC_SoftTrgEN, 426
 - ADC_StructInit, 427
- lks32mc08x_adc.h
 - ADC0, 118
 - ADC_10_TIMES_SAMPLE, 118
 - ADC_11_TIMES_SAMPLE, 118
 - ADC_12_TIMES_SAMPLE, 118
 - ADC_13_TIMES_SAMPLE, 118
 - ADC_14_TIMES_SAMPLE, 118
 - ADC_15_TIMES_SAMPLE, 119
 - ADC_16_TIMES_SAMPLE, 119
 - ADC_17_TIMES_SAMPLE, 119
 - ADC_18_TIMES_SAMPLE, 119
 - ADC_19_TIMES_SAMPLE, 119
 - ADC_1_TIMES_SAMPLE, 119
 - ADC_1SEG_TRG, 120
 - ADC_20_TIMES_SAMPLE, 120
 - ADC_2_TIMES_SAMPLE, 120
 - ADC_2SEG_TRG, 120
 - ADC_3_TIMES_SAMPLE, 120
 - ADC_4_TIMES_SAMPLE, 120
 - ADC_4SEG_TRG, 121
 - ADC_5_TIMES_SAMPLE, 121
 - ADC_6_TIMES_SAMPLE, 121
 - ADC_7_TIMES_SAMPLE, 121
 - ADC_8_TIMES_SAMPLE, 121
 - ADC_9_TIMES_SAMPLE, 121
 - ADC_ALL_IRQ_IF, 122
 - ADC_CHANNEL_0, 122
 - ADC_CHANNEL_1, 122
 - ADC_CHANNEL_10, 122
 - ADC_CHANNEL_11, 122
 - ADC_CHANNEL_12, 122
 - ADC_CHANNEL_13, 123
 - ADC_CHANNEL_14, 123
 - ADC_CHANNEL_15, 123
 - ADC_CHANNEL_16, 123
 - ADC_CHANNEL_17, 123
 - ADC_CHANNEL_18, 123
 - ADC_CHANNEL_19, 124
 - ADC_CHANNEL_2, 124
 - ADC_CHANNEL_3, 124
 - ADC_CHANNEL_4, 124
 - ADC_CHANNEL_5, 124
 - ADC_CHANNEL_6, 124
 - ADC_CHANNEL_7, 125
 - ADC_CHANNEL_8, 125
 - ADC_CHANNEL_9, 125
 - ADC_ChannelConfig, 132
 - ADC_ClearIRQFlag, 133
 - ADC_Clock_12M, 125
 - ADC_Clock_24M, 125
 - ADC_Clock_48M, 125
 - ADC_DAT0_HTH, 125
 - ADC_DAT0_LTH, 126
 - ADC_DAT0_OV_IRQ_EN, 126
 - ADC_DAT0_OV_IRQ_IF, 126
 - ADC_EOS0_IRQ_EN, 126
 - ADC_EOS0_IRQ_IF, 126
 - ADC_EOS1_IRQ_EN, 126
 - ADC_EOS1_IRQ_IF, 127
 - ADC_EOS2_IRQ_EN, 127
 - ADC_EOS2_IRQ_IF, 127
 - ADC_EOS3_IRQ_EN, 127
 - ADC_EOS3_IRQ_IF, 127
 - ADC_GetConversionValue, 134
 - ADC_GetIRQFlag, 135
 - ADC_H_CONFLICT_IRQ_EN, 127
 - ADC_H_CONFLICT_IRQ_IF, 128
 - ADC_HARDWARE_T0_TRG, 128
 - ADC_HARDWARE_T1_TRG, 128
 - ADC_HARDWARE_T2_TRG, 128
 - ADC_HARDWARE_T3_TRG, 128
 - ADC_Init, 137
 - ADC_LEFT_ALIGN, 128
 - ADC_MCPWM_T0_TRG, 129
 - ADC_MCPWM_T1_TRG, 129
 - ADC_MCPWM_T2_TRG, 129
 - ADC_MCPWM_T3_TRG, 129
 - ADC_RIGHT_ALIGN, 129
 - ADC_S_CONFLICT_IRQ_EN, 129
 - ADC_S_CONFLICT_IRQ_IF, 130
 - ADC_SoftTrgEN, 138
 - ADC_StructInit, 139
 - ADC_UTIMER_T0_TRG, 130
 - ADC_UTIMER_T1_TRG, 130
 - ADC_UTIMER_T2_TRG, 130
 - ADC_UTIMER_T3_TRG, 130
 - CHN0, 131
 - CHN1, 131
 - CHN10, 131
 - CHN11, 131
 - CHN12, 131
 - CHN13, 131
 - CHN14, 131
 - CHN15, 131
 - CHN16, 131
 - CHN17, 131

- CHN18, [131](#)
- CHN19, [131](#)
- CHN2, [131](#)
- CHN3, [131](#)
- CHN4, [131](#)
- CHN5, [131](#)
- CHN6, [131](#)
- CHN7, [131](#)
- CHN8, [131](#)
- CHN9, [131](#)
- CHNx, [131](#)
- DAT0, [132](#)
- DAT1, [132](#)
- DAT10, [132](#)
- DAT11, [132](#)
- DAT12, [132](#)
- DAT13, [132](#)
- DAT14, [132](#)
- DAT15, [132](#)
- DAT16, [132](#)
- DAT17, [132](#)
- DAT18, [132](#)
- DAT19, [132](#)
- DAT2, [132](#)
- DAT3, [132](#)
- DAT4, [132](#)
- DAT5, [132](#)
- DAT6, [132](#)
- DAT7, [132](#)
- DAT8, [132](#)
- DAT9, [132](#)
- DATx, [131](#)
- RefVol_1V2, [130](#)
- RefVol_2V4, [131](#)
- lks32mc08x.can.c
 - CAN_GetIRQFlag, [432](#)
 - CAN_Init, [433](#)
 - CAN_Manual_Awake, [434](#)
 - CAN_ReadALC, [435](#)
 - CAN_ReadErrorCode, [436](#)
 - CAN_ReadRXERRCNT, [437](#)
 - CAN_ReadState, [438](#)
 - CAN_ReadTXERRCNT, [440](#)
 - CAN_ReceFramInfor, [441](#)
 - CAN_Receive_Msg, [442](#)
 - CAN_Sleep, [443](#)
 - CAN_StructInit, [444](#)
 - EFF_ID_Filter, [445](#)
 - ID1_Filter_Dual, [446](#)
 - ID2_Filter_Dual, [447](#)
 - My_CAN_Send_Msg, [449](#)
 - SFF_ID_Filter, [450](#)
- lks32mc08x.can.h
 - CAN_ACTIVE_ERROR_FLAG, [152](#)
 - CAN_ALLSR, [146](#)
 - CAN_BITERROR, [151](#)
 - CAN_CONNFIRM, [152](#)
 - CAN_CONNFIRM_DELIMITER, [152](#)
 - CAN_CRC, [152](#)
 - CAN_CRC_DELIMITER, [152](#)
 - CAN_DATA_FIELD, [152](#)
 - CAN_DATAAVAIL, [146](#)
 - CAN_DATALENGTHCODE, [152](#)
 - CAN_ERROR_DELIMITER, [152](#)
 - CAN_ERROV, [146](#)
 - CAN_FILLERROR, [151](#)
 - CAN_FORMATERROR, [151](#)
 - CAN_FRAMEEND, [152](#)
 - CAN_GetIRQFlag, [153](#)
 - CAN_ID15_ID12, [152](#)
 - CAN_ID17_ID13, [152](#)
 - CAN_ID20_ID18, [152](#)
 - CAN_ID28_ID21, [152](#)
 - CAN_ID4_ID0, [152](#)
 - CAN_IDE, [152](#)
 - CAN_IE_BUSERR, [146](#)
 - CAN_IE_LOSTARB, [147](#)
 - CAN_IE_PASSIVEERR, [147](#)
 - CAN_IE_RFIFONEMPTY, [147](#)
 - CAN_IE_RFIFOV, [147](#)
 - CAN_IE_TXDONE, [147](#)
 - CAN_IE_WAKE, [147](#)
 - CAN_IE_WERR, [148](#)
 - CAN_IF_BUSERR, [148](#)
 - CAN_IF_LOSTARB, [148](#)
 - CAN_IF_PASSIVEERR, [148](#)
 - CAN_IF_RFIFONEMPTY, [148](#)
 - CAN_IF_RFIFOV, [148](#)
 - CAN_IF_TXDONE, [149](#)
 - CAN_IF_WAKE, [149](#)
 - CAN_IF_WERR, [149](#)
 - CAN_Init, [154](#)
 - CAN_INTERMISSION, [152](#)
 - CAN_KEEPO, [152](#)
 - CAN_KEEPI, [152](#)
 - CAN_LOST_ID0, [152](#)
 - CAN_LOST_ID1, [152](#)
 - CAN_LOST_ID2, [152](#)
 - CAN_LOST_ID3, [153](#)
 - CAN_LOST_ID4, [153](#)
 - CAN_LOST_ID5, [153](#)
 - CAN_LOST_ID6, [153](#)
 - CAN_LOST_ID7, [153](#)
 - CAN_LOST_ID8, [153](#)
 - CAN_LOST_ID9, [153](#)
 - CAN_LOST_IDA, [153](#)
 - CAN_LOST_IDE, [153](#)
 - CAN_LOST_SRTR, [153](#)
 - CAN_Manual_Awake, [155](#)
 - CAN_MONITOR_MODE, [149](#)
 - CAN_NORMAL_MODE, [149](#)
 - CAN_ONBUS, [149](#)
 - CAN_OTHERERROR, [151](#)
 - CAN_OVERLOAD_FLAG, [152](#)
 - CAN_PASSIVE_ERROR, [152](#)
 - CAN_ReadState, [156](#)

- CAN_RECE_PROCESS, 151
- CAN_Receive_Msg, 158
- CAN_RFIFOEMPTY, 150
- CAN_RTR, 152
- CAN_RXING, 150
- CAN_SELFTEST_MODE, 150
- CAN_SEND_PROCESS, 151
- CAN_Sleep, 158
- CAN_SRTR, 152
- CAN_START, 152
- CAN_StructInit, 159
- CAN_TFIFOEMPTY, 150
- CAN_TOLERATE_DOMINANT, 152
- CAN_TRNASDONE, 150
- CAN_TXING, 151
- EFF_ID_Filter, 160
- enumErrortime, 151
- enumErrortype, 151
- enumErrposition, 151
- enumLossposition, 152
- ID1_Filter_Dual, 161
- ID2_Filter_Dual, 162
- My_CAN_Send_Msg, 164
- null0, 152
- null1, 152
- null2, 152
- null3, 152
- null4, 152
- SFF_ID_Filter, 165
- lks32mc08x_cmp.c
 - CMP_ClearIRQFlag, 456
 - CMP_GetIRQFlag, 457
 - CMP_Init, 458
 - CMP_StructInit, 460
- lks32mc08x_cmp.h
 - CMP0, 171
 - CMP0_SEL_P_IP0, 171
 - CMP0_SEL_P_IP1, 172
 - CMP0_SEL_P_IP2, 172
 - CMP0_SEL_P_IP3, 172
 - CMP0_SEL_P_IP4, 172
 - CMP0_SEL_P_OPA0_IP, 172
 - CMP0_SEL_P_OPA0_OUT, 172
 - CMP0_SEL_P_OPA1_OUT, 173
 - CMP1, 173
 - CMP1_SEL_P_AVSS, 173
 - CMP1_SEL_P_IP0, 173
 - CMP1_SEL_P_IP1, 173
 - CMP1_SEL_P_IP2, 173
 - CMP1_SEL_P_IP3, 174
 - CMP1_SEL_P_OPA2_OUT, 174
 - CMP1_SEL_P_OPA3_IP, 174
 - CMP1_SEL_P_OPA3_OUT, 174
 - CMP_ClearIRQFlag, 178
 - CMP_GetIRQFlag, 179
 - CMP_HYS_0mV, 174
 - CMP_HYS_20mV, 174
 - CMP_Init, 180
- CMP_SELN_DAC, 175
- CMP_SELN_HALL_MID, 175
- CMP_SELN_IN, 175
- CMP_SELN_REF, 175
- CMP_StructInit, 181
- SELN_CMP_IN, 175
- SELN_DAC, 175
- SELN_HALL_MID, 176
- SELN_REF, 176
- SELP_CMP_IP0, 176
- SELP_CMP_IP1, 176
- SELP_CMP_IP2, 176
- SELP_CMP_IP3, 176
- SELP_CMP_IP4, 177
- SELP_CMP_OPA0IP, 177
- SELP_CMP_OPA0OUT, 177
- SELP_CMP_OPA1OUT, 177
- SELP_CMP_OPA2OUT, 177
- SELP_CMP_OPA3IP, 177
- SELP_CMP_OPA3OUT, 178
- lks32mc08x_dac.c
 - DAC_Init, 463
 - DAC_Output, 464
 - DAC_OutputValue, 464
 - DAC_OutputVoltage, 465
 - DAC_StructInit, 466
- lks32mc08x_dac.h
 - DAC_Init, 186
 - DAC_Output, 187
 - DAC_OutputValue, 187
 - DAC_OutputVoltage, 188
 - DAC_RANGE_1V2, 185
 - DAC_RANGE_3V0, 185
 - DAC_RANGE_4V85, 185
 - DAC_StructInit, 189
- lks32mc08x_dma.c
 - DMA_CHx_EN, 470
 - DMA_ClearIRQFlag, 470
 - DMA_GetIRQFlag, 472
 - DMA_Init, 473
 - DMA_StructInit, 475
- lks32mc08x_dma.h
 - DMA_BYTE_TRANS, 193
 - DMA_CH0_ADC0_REQ_EN, 193
 - DMA_CH0_EIF, 193
 - DMA_CH0_FIF, 193
 - DMA_CH0_TIMER0_REQ_EN, 193
 - DMA_CH0_TIMER1_REQ_EN, 193
 - DMA_CH1_EIF, 193
 - DMA_CH1_FIF, 194
 - DMA_CH1_MCPWM_REQ_EN, 194
 - DMA_CH1_SPI_RX_REQ_EN, 194
 - DMA_CH1_TIMER2_REQ_EN, 194
 - DMA_CH2_EIF, 194
 - DMA_CH2_FIF, 194
 - DMA_CH2_SPI_TX_REQ_EN, 195
 - DMA_CH2_TIMER3_REQ_EN, 195
 - DMA_CH2_UART0_REQ_EN, 195

- DMA_CH3_CAN_REQ_EN, 195
- DMA_CH3_EIF, 195
- DMA_CH3_FIF, 195
- DMA_CH3_I2C_REQ_EN, 196
- DMA_CH3_UART1_REQ_EN, 196
- DMA_CHx_EN, 197
- DMA_ClearIRQFlag, 198
- DMA_GetIRQFlag, 199
- DMA_HALFWORD_TRANS, 196
- DMA_Init, 201
- DMA_StructInit, 202
- DMA_TCIE, 196
- DMA_TEIE, 196
- DMA_WORD_TRANS, 196
- MEMORY2PERI, 197
- PERI2MEMORY, 197
- lks32mc08x_dsp.c
 - DSP_arctan, 478
 - DSP_CalcCordic, 478
 - DSP_CalcDivision, 480
 - DSP_CalcSprt, 481
 - DSP_Cmd, 482
 - DSP_cos, 483
 - DSP_div, 483
 - DSP_GetSqrt, 483
 - DSP_Init, 485
 - DSP_mod, 485
 - DSP_rms, 485
 - DSP_sin, 485
 - DSP_sqrt, 485
- lks32mc08x_dsp.h
 - Arc_or_SinCFuction, 208
 - ARCTAN, 208
 - DSP_arctan, 208
 - DSP_CALC_ARCTAN, 206
 - DSP_CALC_COS, 206
 - DSP_CALC_DIV, 206
 - DSP_CALC_MOD, 207
 - DSP_CALC_RMS, 207
 - DSP_CALC_SIN, 207
 - DSP_CALC_SQRT, 208
 - DSP_CalcCordic, 208
 - DSP_CalcDivision, 210
 - DSP_CalcSprt, 211
 - DSP_Cmd, 212
 - DSP_cos, 213
 - DSP_div, 213
 - DSP_GetSqrt, 214
 - DSP_Init, 215
 - DSP_mod, 215
 - DSP_rms, 215
 - DSP_sin, 215
 - DSP_sqrt, 216
 - SINCOS, 208
- lks32mc08x_exti.c
 - EXTI_ClearIRQFlag, 489
 - EXTI_GetIRQFlag, 490
 - EXTI_Trigger_Config, 491
- lks32mc08x_exti.h
 - EXTI, 219
 - EXTI_ClearIRQFlag, 220
 - EXTI_GetIRQFlag, 221
 - EXTI_Trigger_Config, 222
 - EXTI_Trigger_Edge, 219
 - EXTI_Trigger_Negedge, 219
 - EXTI_Trigger_None, 219
 - EXTI_Trigger_Posedge, 219
 - EXTIMode_TypeDef, 219
- lks32mc08x_flash.c
 - erase_flag, 499
 - EraseSector, 494
 - progmem_flag, 499
 - ProgramPage, 495
 - Read_Flash, 496
 - Read_More_Flash, 498
- lks32mc08x_flash.h
 - erase_flag, 230
 - EraseSector, 225
 - Flash_MAIN, 225
 - Flash_NVR, 225
 - progmem_flag, 230
 - ProgramPage, 226
 - Read_More_Flash, 228
- lks32mc08x_gpio.c
 - GPIO_Config, 503
 - GPIO_Init, 504
 - GPIO_LockRegister, 505
 - GPIO_PinAFConfig, 506
 - GPIO_ReadInputData, 508
 - GPIO_ReadInputDataBit, 509
 - GPIO_ReadOutputData, 510
 - GPIO_ReadOutputDataBit, 511
 - GPIO_ReadPinBit, 512
 - GPIO_ResetBits, 512
 - GPIO_SetBits, 513
 - GPIO_StructInit, 514
 - GPIO_UnlockRegister, 515
 - GPIO_Write, 516
 - GPIO_WriteBit, 517
 - GPIO_WritePinBit, 519
- lks32mc08x_gpio.h
 - AF0_GPIO, 234
 - AF10_CAN, 234
 - AF11_SIF, 234
 - AF1_SYSAF, 234
 - AF2_HALL, 235
 - AF3_MCPWM, 235
 - AF4_UART, 235
 - AF5_SPI, 235
 - AF6_I2C, 235
 - AF7_TIMER01, 235
 - AF8_TIMER23, 236
 - AF9_DEBUG, 236
 - Bit_RESET, 244
 - Bit_SET, 244
 - BitAction, 244

- GPIO0, [236](#)
- GPIO1, [236](#)
- GPIO2, [236](#)
- GPIO3, [236](#)
- GPIO_AF_CAN, [237](#)
- GPIO_AF_DEBUG, [237](#)
- GPIO_AF_GPIO, [237](#)
- GPIO_AF_HALL, [237](#)
- GPIO_AF_I2C, [237](#)
- GPIO_AF_MCPWM, [237](#)
- GPIO_AF_SIF, [237](#)
- GPIO_AF_SPI, [238](#)
- GPIO_AF_SYSAF, [238](#)
- GPIO_AF_TIMER0, [238](#)
- GPIO_AF_TIMER1, [238](#)
- GPIO_AF_TIMER2, [238](#)
- GPIO_AF_TIMER3, [238](#)
- GPIO_AF_UART, [238](#)
- GPIO_Config, [245](#)
- GPIO_Init, [246](#)
- GPIO_LockRegister, [247](#)
- GPIO_Mode_ANA, [245](#)
- GPIO_Mode_IN, [245](#)
- GPIO_Mode_IO, [245](#)
- GPIO_Mode_OUT, [245](#)
- GPIO_Pin_0, [238](#)
- GPIO_Pin_1, [239](#)
- GPIO_Pin_10, [239](#)
- GPIO_Pin_11, [239](#)
- GPIO_Pin_12, [239](#)
- GPIO_Pin_13, [239](#)
- GPIO_Pin_14, [239](#)
- GPIO_Pin_15, [240](#)
- GPIO_Pin_2, [240](#)
- GPIO_Pin_3, [240](#)
- GPIO_Pin_4, [240](#)
- GPIO_Pin_5, [240](#)
- GPIO_Pin_6, [240](#)
- GPIO_Pin_7, [241](#)
- GPIO_Pin_8, [241](#)
- GPIO_Pin_9, [241](#)
- GPIO_Pin_ALL, [241](#)
- GPIO_Pin_NONE, [241](#)
- GPIO_PinAFConfig, [248](#)
- GPIO_PinSource_0, [241](#)
- GPIO_PinSource_1, [242](#)
- GPIO_PinSource_10, [242](#)
- GPIO_PinSource_11, [242](#)
- GPIO_PinSource_12, [242](#)
- GPIO_PinSource_13, [242](#)
- GPIO_PinSource_14, [242](#)
- GPIO_PinSource_15, [243](#)
- GPIO_PinSource_2, [243](#)
- GPIO_PinSource_3, [243](#)
- GPIO_PinSource_4, [243](#)
- GPIO_PinSource_5, [243](#)
- GPIO_PinSource_6, [243](#)
- GPIO_PinSource_7, [244](#)
- GPIO_PinSource_8, [244](#)
- GPIO_PinSource_9, [244](#)
- GPIO_PuPd_NOPULL, [245](#)
- GPIO_PuPd_UP, [245](#)
- GPIO_ReadInputData, [250](#)
- GPIO_ReadInputDataBit, [251](#)
- GPIO_ReadOutputData, [252](#)
- GPIO_ReadOutputDataBit, [253](#)
- GPIO_ReadPinBit, [254](#)
- GPIO_ResetBits, [254](#)
- GPIO_SetBits, [255](#)
- GPIO_StructInit, [256](#)
- GPIO_UnlockRegister, [257](#)
- GPIO_Write, [258](#)
- GPIO_WriteBit, [259](#)
- GPIO_WritePinBit, [261](#)
- GPIO_Mode_TypeDef, [244](#)
- GPIO_PuPd_TypeDef, [245](#)
- lks32mc08x.hall.c
 - HALL_ClearIRQFlag, [524](#)
 - HALL_GetCaptureValue, [525](#)
 - HALL_GetCount, [526](#)
 - HALL_GetFilterValue, [527](#)
 - HALL_GetIRQFlag, [528](#)
 - HALL_Init, [529](#)
 - HALL_StructInit, [530](#)
 - HALL_WIDCount, [531](#)
- lks32mc08x.hall.h
 - HALL_CAPTURE_EVENT, [265](#)
 - HALL_ClearIRQFlag, [266](#)
 - HALL_CLK_DIV1, [265](#)
 - HALL_CLK_DIV2, [265](#)
 - HALL_CLK_DIV4, [265](#)
 - HALL_CLK_DIV8, [265](#)
 - HALL_GetCaptureValue, [267](#)
 - HALL_GetCount, [268](#)
 - HALL_GetFilterValue, [269](#)
 - HALL_GetIRQFlag, [269](#)
 - HALL_Init, [270](#)
 - HALL_OVERFLOW_EVENT, [265](#)
 - HALL_StructInit, [272](#)
 - HALL_WIDCount, [273](#)
- lks32mc08x.i2c.c
 - i2c_dma_delay_over, [534](#)
 - i2c_dma_init, [535](#)
 - i2c_dma_rx, [537](#)
 - i2c_dma_state, [546](#)
 - i2c_dma_state_over, [538](#)
 - i2c_dma_state_start, [539](#)
 - i2c_dma_tx, [540](#)
 - I2C_Init, [541](#)
 - I2C_ReadData, [543](#)
 - I2C_SendData, [544](#)
 - I2C_StructInit, [545](#)
- lks32mc08x.i2c.h
 - i2c_dma_init, [276](#)
 - i2c_dma_rx, [277](#)
 - i2c_dma_state_over, [278](#)

- i2c_dma_tx, 279
- I2C_Init, 281
- I2C_ReadData, 282
- I2C_SendData, 283
- I2C_StructInit, 284
- lks32mc08x_iwdg.c
 - IWDG_DISABLE, 549
 - IWDG_ENABLE, 550
 - IWDG_Feed, 551
 - IWDG_Init, 552
 - IWDG_StructInit, 553
- lks32mc08x_iwdg.h
 - IWDG_DISABLE, 288
 - IWDG_ENABLE, 289
 - IWDG_Feed, 290
 - IWDG_Init, 291
 - IWDG_StructInit, 292
 - WDT_RTH_2S, 287
 - WDT_RTH_4S, 287
 - WDT_RTH_64S, 287
 - WDT_RTH_8S, 288
- lks32mc08x_mcpwm.c
 - MCPWM_ClearIRQFlag, 556
 - MCPWM_GetIRQFlag, 558
 - MCPWM_Init, 560
 - MCPWM_StructInit, 561
 - PWMOutputs, 562
- lks32mc08x_mcpwm.h
 - CENTRAL_PWM_MODE, 299
 - EDGE_PWM_MODE, 299
 - FAIL_SEL_CMP, 300
 - FAIL_SEL_IO, 300
 - HIGH_LEVEL, 300
 - HIGH_LEVEL_ACTIVE, 300
 - LOW_LEVEL, 300
 - LOW_LEVEL_ACTIVE, 300
 - MCPWM_CH0.SHORT_IRQ_EN, 301
 - MCPWM_CH0.SHORT_IRQ_IF, 301
 - MCPWM_CH1.SHORT_IRQ_EN, 301
 - MCPWM_CH1.SHORT_IRQ_IF, 301
 - MCPWM_CH2.SHORT_IRQ_EN, 301
 - MCPWM_CH2.SHORT_IRQ_IF, 301
 - MCPWM_CH3.SHORT_IRQ_EN, 302
 - MCPWM_CH3.SHORT_IRQ_IF, 302
 - MCPWM_ClearIRQFlag, 312
 - MCPWM_EIF_CH0.SHORT, 302
 - MCPWM_EIF_CH1.SHORT, 302
 - MCPWM_EIF_CH2.SHORT, 302
 - MCPWM_EIF_CH3.SHORT, 302
 - MCPWM_EIF_FAIL0, 303
 - MCPWM_EIF_FAIL1, 303
 - MCPWM_FAIL0_IRQ_EN, 303
 - MCPWM_FAIL0_IRQ_IF, 303
 - MCPWM_FAIL1_IRQ_EN, 303
 - MCPWM_FAIL1_IRQ_IF, 303
 - MCPWM_GetIRQFlag, 314
 - MCPWM_IF_SHADE, 304
 - MCPWM_IF_T0, 304
 - MCPWM_IF_T1, 304
 - MCPWM_IF_TH00, 304
 - MCPWM_IF_TH01, 304
 - MCPWM_IF_TH10, 304
 - MCPWM_IF_TH11, 305
 - MCPWM_IF_TH20, 305
 - MCPWM_IF_TH21, 305
 - MCPWM_IF_TH30, 305
 - MCPWM_IF_TH31, 305
 - MCPWM_IF_TMR0, 305
 - MCPWM_IF_TMR1, 306
 - MCPWM_IF_TMR2, 306
 - MCPWM_IF_TMR3, 306
 - MCPWM_Init, 316
 - MCPWM_MOE_DISABLE_MASK, 306
 - MCPWM_MOE_ENABLE_MASK, 306
 - MCPWM_SHADE_IRQ_EN, 306
 - MCPWM_SHADE_IRQ_IF, 307
 - MCPWM_StructInit, 317
 - MCPWM_T0_IRQ_EN, 307
 - MCPWM_T0_IRQ_IF, 307
 - MCPWM_T1_IRQ_EN, 307
 - MCPWM_T1_IRQ_IF, 307
 - MCPWM_TH00_IRQ_EN, 307
 - MCPWM_TH00_IRQ_IF, 308
 - MCPWM_TH01_IRQ_EN, 308
 - MCPWM_TH01_IRQ_IF, 308
 - MCPWM_TH10_IRQ_EN, 308
 - MCPWM_TH10_IRQ_IF, 308
 - MCPWM_TH11_IRQ_EN, 308
 - MCPWM_TH11_IRQ_IF, 309
 - MCPWM_TH20_IRQ_EN, 309
 - MCPWM_TH20_IRQ_IF, 309
 - MCPWM_TH21_IRQ_EN, 309
 - MCPWM_TH21_IRQ_IF, 309
 - MCPWM_TH30_IRQ_EN, 309
 - MCPWM_TH30_IRQ_IF, 310
 - MCPWM_TH31_IRQ_EN, 310
 - MCPWM_TH31_IRQ_IF, 310
 - MCPWM_TMR0_IRQ_EN, 310
 - MCPWM_TMR0_IRQ_IF, 310
 - MCPWM_TMR1_IRQ_EN, 310
 - MCPWM_TMR1_IRQ_IF, 311
 - MCPWM_TMR2_IRQ_EN, 311
 - MCPWM_TMR2_IRQ_IF, 311
 - MCPWM_TMR3_IRQ_EN, 311
 - MCPWM_TMR3_IRQ_IF, 311
 - MCPWM_UPDATE_REG, 311
 - PWMOutputs, 318
- lks32mc08x_nvr.h
 - Read_NVR, 323
 - Read_Trim, 323
- lks32mc08x_opa.c
 - OPA_Init, 566
 - OPA_OUT, 567
 - OPA_StructInit, 568
- lks32mc08x_opa.h
 - enumOPA, 326

- OPA0, [326](#)
- OPA1, [326](#)
- OPA2, [326](#)
- OPA3, [326](#)
- OPA_Init, [327](#)
- OPA_OUT, [328](#)
- OPA_StructInit, [329](#)
- PGA_GAIN_20, [326](#)
- PGA_GAIN_4P25, [326](#)
- PGA_GAIN_6, [326](#)
- PGA_GAIN_9P5, [326](#)
- lks32mc08x_spi.c
 - SPI_ClearIRQFlag, [571](#)
 - SPI_GetIRQFlag, [572](#)
 - SPI_Init, [573](#)
 - SPI_ReadData, [574](#)
 - SPI_SendData, [575](#)
 - SPI_StructInit, [576](#)
- lks32mc08x_spi.h
 - SPI_ClearIRQFlag, [335](#)
 - SPI_DMA_DISABLE, [332](#)
 - SPI_DMA_ENABLE, [332](#)
 - SPI_Duplex, [334](#)
 - SPI_FIRSTSEND_LSB, [332](#)
 - SPI_FIRSTSEND_MSB, [332](#)
 - SPI_Full, [334](#)
 - SPI_GetIRQFlag, [336](#)
 - SPI_Half_Rx, [334](#)
 - SPI_Half_Tx, [334](#)
 - SPI_IF_DataOver, [333](#)
 - SPI_IF_SSErr, [333](#)
 - SPI_IF_TranDone, [333](#)
 - SPI_Init, [337](#)
 - SPI_IRQEna_DataOver, [333](#)
 - SPI_IRQEna_Enable, [333](#)
 - SPI_IRQEna_SSErr, [333](#)
 - SPI_IRQEna_TranDone, [334](#)
 - SPI_Master, [334](#)
 - SPI_Mode, [334](#)
 - SPI_ReadData, [338](#)
 - SPI_SendData, [339](#)
 - SPI_Slave, [334](#)
 - SPI_StructInit, [340](#)
- lks32mc08x_sys.c
 - SetWakeIO, [580](#)
 - SoftDelay, [581](#)
 - Switch2HRC, [582](#)
 - Switch2PLL, [583](#)
 - SYS_AnalogModuleClockCmd, [584](#)
 - SYS_ClearRst, [586](#)
 - SYS_EnablePORFilter, [587](#)
 - SYS_FallSleep, [588](#)
 - SYS_FeedDog, [589](#)
 - SYS_GetRstSource, [590](#)
 - SYS_Init, [591](#)
 - SYS_ModuleClockCmd, [592](#)
 - SYS_SetWakeInterval, [593](#)
 - SYS_SoftResetModule, [595](#)
 - SYS_StructInit, [596](#)
 - SYS_WatchDogCmd, [597](#)
 - WAKE_InitStruct, [598](#)
- lks32mc08x_sys.h
 - enumWakeTrigLevel, [355](#)
 - MCU_POWER_3v3, [346](#)
 - MCU_POWER_5v0, [346](#)
 - POWER_MODE, [346](#)
 - SetWakeIO, [356](#)
 - Switch2HRC, [357](#)
 - Switch2PLL, [358](#)
 - SYS_AnalogModule_ADC, [346](#)
 - SYS_AnalogModule_BGP, [346](#)
 - SYS_AnalogModule_CMP0, [347](#)
 - SYS_AnalogModule_CMP1, [347](#)
 - SYS_AnalogModule_DAC, [347](#)
 - SYS_AnalogModule_OPA0, [347](#)
 - SYS_AnalogModule_OPA1, [347](#)
 - SYS_AnalogModule_OPA2, [347](#)
 - SYS_AnalogModule_OPA3, [348](#)
 - SYS_AnalogModule_PLL, [348](#)
 - SYS_AnalogModule_RCH, [348](#)
 - SYS_AnalogModule_TMP, [348](#)
 - SYS_AnalogModule_XTAL, [348](#)
 - SYS_AnalogModuleClockCmd, [359](#)
 - SYS_ClearRst, [361](#)
 - SYS_CLK_SEL_CLK_HS, [348](#)
 - SYS_CLK_SEL_PLL, [349](#)
 - SYS_Clk.SPIDiv1, [349](#)
 - SYS_Clk.SPIDiv2, [349](#)
 - SYS_Clk.SPIDiv4, [349](#)
 - SYS_Clk.SPIDiv8, [349](#)
 - SYS_Clk_UARTDiv1, [349](#)
 - SYS_Clk_UARTDiv2, [350](#)
 - SYS_Clk_UARTDiv4, [350](#)
 - SYS_Clk_UARTDiv8, [350](#)
 - SYS_FallSleep, [362](#)
 - SYS_FeedDog, [363](#)
 - SYS_GetRstSource, [364](#)
 - SYS_Init, [365](#)
 - SYS_Module_CAN, [350](#)
 - SYS_Module_DSP, [350](#)
 - SYS_Module_HALL, [350](#)
 - SYS_Module_I2C, [351](#)
 - SYS_Module_MCPWM, [351](#)
 - SYS_Module_SPI, [351](#)
 - SYS_Module_TIMER, [351](#)
 - SYS_Module_UART0, [351](#)
 - SYS_Module_UART1, [351](#)
 - SYS_ModuleClockCmd, [366](#)
 - SYS_PLLREDIV_1, [352](#)
 - SYS_PLLREDIV_4, [352](#)
 - SYS_PLLSRSEL_CRYSTAL, [352](#)
 - SYS_PLLSRSEL_RCH, [352](#)
 - SYS_ResetIO, [367](#)
 - SYS_ResetPeripheral, [368](#)
 - SYS_ResetPeripheralInternal, [368](#)
 - SYS_RstSrc_HPOR, [352](#)

- SYS_RstSrc_KEYBOARD, 352
- SYS_RstSrc_LPOR, 353
- SYS_RstSrc_WDT, 353
- SYS_SetWakeInterval, 368
- SYS_SoftResetModule, 369
- SYS_StructInit, 371
- SYS_VolSelModule, 372
- SYS_VolSelModuleEnableIRQ, 372
- SYS_VolSelModuleIRQ, 372
- SYS_WakeInterval_025s, 353
- SYS_WakeInterval_05s, 353
- SYS_WakeInterval_16s, 353
- SYS_WakeInterval_1s, 353
- SYS_WakeInterval_2s, 354
- SYS_WakeInterval_32s, 354
- SYS_WakeInterval_4s, 354
- SYS_WakeInterval_8s, 354
- SYS_WatchDogCmd, 372
- SYS_WD_TimeOut2s, 354
- SYS_WD_TimeOut4s, 354
- SYS_WD_TimeOut64s, 355
- SYS_WD_TimeOut8s, 355
- WAKEIO_P0_0, 355
- WAKEIO_P0_1, 355
- WAKEIO_P1_0, 355
- WAKEIO_P1_1, 355
- WAKETRIG_HIGH, 356
- WAKETRIG_LOW, 356
- lks32mc08x.tim.c
 - ECD_ClearIRQFlag, 605
 - ECD_GetIRQFlag, 606
 - TIM_ClearIRQFlag, 607
 - TIM_ECD_GetCount, 608
 - TIM_ECDCmd, 609
 - TIM_ECDInit, 610
 - TIM_ECDStructInit, 611
 - TIM_GetIRQFlag, 612
 - TIM_Timer_GetCMPT0, 613
 - TIM_Timer_GetCMPT1, 614
 - TIM_Timer_GetCount, 615
 - TIM_TimerCmd, 616
 - TIM_TimerInit, 617
 - TIM_TimerStrutInit, 619
- lks32mc08x.tim.h
 - ECD_ClearIRQFlag, 383
 - ECD_Clk_Div1, 378
 - ECD_Clk_Div2, 378
 - ECD_Clk_Div4, 378
 - ECD_Clk_Div8, 379
 - ECD_GetIRQFlag, 384
 - ECD_IF_OVERFLOW, 379
 - ECD_IF_UNDERFLOW, 379
 - ECD_IRQEna_None, 379
 - ECD_IRQEna_OverFlow, 379
 - ECD_IRQEna_UnderFlow, 379
 - ECD_Mode_CCWCW, 380
 - ECD_Mode_CCWCW_UpDown, 380
 - ECD_Mode_CCWSIGN, 380
 - ECD_Mode_CCWSIGN_UpDown, 380
 - ECD_Mode_T1, 380
 - ECD_Mode_T1_T2, 380
 - TIM_ClearIRQFlag, 385
 - TIM_Clk_Div1, 381
 - TIM_Clk_Div2, 381
 - TIM_Clk_Div4, 381
 - TIM_Clk_Div8, 381
 - TIM_ECD_GetCount, 386
 - TIM_ECDCmd, 387
 - TIM_ECDInit, 388
 - TIM_ECDStructInit, 390
 - TIM_GetIRQFlag, 391
 - TIM_Timer_GetCMPT0, 392
 - TIM_Timer_GetCMPT1, 393
 - TIM_Timer_GetCount, 394
 - TIM_TimerCmd, 394
 - TIM_TimerInit, 396
 - TIM_TimerStrutInit, 397
 - TIMER_CapMode_Fall, 381
 - TIMER_CapMode_None, 381
 - TIMER_CapMode_Rise, 382
 - TIMER_CapMode_RiseFall, 382
 - TIMER_IF_CH0, 382
 - TIMER_IF_CH1, 382
 - TIMER_IF_ZERO, 382
 - Timer_IRQEna_All, 383
 - Timer_IRQEna_CH0, 383
 - Timer_IRQEna_CH1, 383
 - Timer_IRQEna_None, 383
 - Timer_IRQEna_Zero, 383
 - Timer_IRQEnaDef, 383
 - TIMER_OPMODE_CAP, 382
 - TIMER_OPMODE_CMP, 383
- lks32mc08x.tmp.c
 - GetCurrentTemperature, 625
 - m_TempertureCof, 627
 - TempSensor_Init, 626
- lks32mc08x.tmp.h
 - GetCurrentTemperature, 401
 - TempSensor_Init, 402
- lks32mc08x.uart.c
 - UART_ClearIRQFlag, 629
 - UART_GetIRQFlag, 630
 - UART_Init, 631
 - UART_ReadData, 633
 - UART_SendAddr_485, 634
 - UART_SendData, 635
 - UART_SendData_485, 636
 - UART_StructInit, 637
- lks32mc08x.uart.h
 - UART_ClearIRQFlag, 410
 - UART_FIRSTSEND_LSB, 406
 - UART_FIRSTSEND_MSB, 406
 - UART_GetIRQFlag, 411
 - UART_IF_CheckError, 406
 - UART_IF_RcvOver, 406
 - UART_IF_SendBufEmpty, 407

UART_IF_SendOver, 407
 UART_IF_StopError, 407
 UART_Init, 412
 UART_IRQEna_CheckError, 407
 UART_IRQEna_RcvOver, 407
 UART_IRQEna_RX_DMA_RE, 407
 UART_IRQEna_SendBuffEmpty, 408
 UART_IRQEna_SendOver, 408
 UART_IRQEna_StopError, 408
 UART_IRQEna_TX_BUF_DMA_RE, 408
 UART_IRQEna_TX_DMA_RE, 408
 UART_Parity_EVEN, 409
 UART_Parity_NO, 409
 UART_Parity_ODD, 409
 UART_ParityMode, 409
 UART_ReadData, 413
 UART_SendAddr_485, 414
 UART_SendData, 415
 UART_SendData_485, 416
 UART_STOPBITS_1b, 408
 UART_STOPBITS_2b, 409
 UART_StructInit, 417
 UART_WORDLENGTH_7b, 409
 UART_WORDLENGTH_8b, 409
 LOW_LEVEL
 lks32mc08x_mcpwm.h, 300
 LOW_LEVEL_ACTIVE
 lks32mc08x_mcpwm.h, 300
 m_TemperatureCof
 lks32mc08x_tmp.c, 627
 Match485Addr
 UART_InitTypeDef, 108
 MCLK_EN
 MCPWM_InitTypeDef, 78
 MCPWM_Auto_ERR_EN
 MCPWM_InitTypeDef, 78
 MCPWM_CH0_SHORT_IRQ_EN
 lks32mc08x_mcpwm.h, 301
 MCPWM_CH0_SHORT_IRQ_IF
 lks32mc08x_mcpwm.h, 301
 MCPWM_CH1_SHORT_IRQ_EN
 lks32mc08x_mcpwm.h, 301
 MCPWM_CH1_SHORT_IRQ_IF
 lks32mc08x_mcpwm.h, 301
 MCPWM_CH2_SHORT_IRQ_EN
 lks32mc08x_mcpwm.h, 301
 MCPWM_CH2_SHORT_IRQ_IF
 lks32mc08x_mcpwm.h, 301
 MCPWM_CH3_SHORT_IRQ_EN
 lks32mc08x_mcpwm.h, 302
 MCPWM_CH3_SHORT_IRQ_IF
 lks32mc08x_mcpwm.h, 302
 MCPWM_ClearIRQFlag
 lks32mc08x_mcpwm.c, 556
 lks32mc08x_mcpwm.h, 312
 MCPWM_Cnt_EN
 MCPWM_InitTypeDef, 78
 MCPWM{EIF_CH0_SHORT
 lks32mc08x_mcpwm.h, 302
 MCPWM{EIF_CH1_SHORT
 lks32mc08x_mcpwm.h, 302
 MCPWM{EIF_CH2_SHORT
 lks32mc08x_mcpwm.h, 302
 MCPWM{EIF_CH3_SHORT
 lks32mc08x_mcpwm.h, 302
 MCPWM{EIF_FAIL0
 lks32mc08x_mcpwm.h, 303
 MCPWM{EIF_FAIL1
 lks32mc08x_mcpwm.h, 303
 MCPWM_FAIL0_IRQ_EN
 lks32mc08x_mcpwm.h, 303
 MCPWM_FAIL0_IRQ_IF
 lks32mc08x_mcpwm.h, 303
 MCPWM_FAIL1_IRQ_EN
 lks32mc08x_mcpwm.h, 303
 MCPWM_FAIL1_IRQ_IF
 lks32mc08x_mcpwm.h, 303
 MCPWM_GetIRQFlag
 lks32mc08x_mcpwm.c, 558
 lks32mc08x_mcpwm.h, 314
 MCPWM_IF_SHADE
 lks32mc08x_mcpwm.h, 304
 MCPWM_IF_T0
 lks32mc08x_mcpwm.h, 304
 MCPWM_IF_T1
 lks32mc08x_mcpwm.h, 304
 MCPWM_IF_TH00
 lks32mc08x_mcpwm.h, 304
 MCPWM_IF_TH01
 lks32mc08x_mcpwm.h, 304
 MCPWM_IF_TH10
 lks32mc08x_mcpwm.h, 304
 MCPWM_IF_TH11
 lks32mc08x_mcpwm.h, 305
 MCPWM_IF_TH20
 lks32mc08x_mcpwm.h, 305
 MCPWM_IF_TH21
 lks32mc08x_mcpwm.h, 305
 MCPWM_IF_TH30
 lks32mc08x_mcpwm.h, 305
 MCPWM_IF_TH31
 lks32mc08x_mcpwm.h, 305
 MCPWM_IF_TMR0
 lks32mc08x_mcpwm.h, 305
 MCPWM_IF_TMR1
 lks32mc08x_mcpwm.h, 306
 MCPWM_IF_TMR2
 lks32mc08x_mcpwm.h, 306
 MCPWM_IF_TMR3
 lks32mc08x_mcpwm.h, 306
 MCPWM_Init
 lks32mc08x_mcpwm.c, 560
 lks32mc08x_mcpwm.h, 316
 MCPWM_InitTypeDef, 62
 CH0_NS, 67
 CH0_PS, 67

CH0N_default_output, 67
 CH0N_Polarity_INV, 67
 CH0N_SCTRLN, 68
 CH0P_default_output, 68
 CH0P_Polarity_INV, 68
 CH0P_SCTRLP, 68
 CH1_NS, 68
 CH1_PS, 69
 CH1N_default_output, 69
 CH1N_Polarity_INV, 69
 CH1N_SCTRLN, 69
 CH1P_default_output, 69
 CH1P_Polarity_INV, 70
 CH1P_SCTRLP, 70
 CH2_NS, 70
 CH2_PS, 70
 CH2N_default_output, 70
 CH2N_Polarity_INV, 71
 CH2N_SCTRLN, 71
 CH2P_default_output, 71
 CH2P_Polarity_INV, 71
 CH2P_SCTRLP, 71
 CH3_NS, 72
 CH3_PS, 72
 CH3N_default_output, 72
 CH3N_Polarity_INV, 72
 CH3N_SCTRLN, 72
 CH3P_default_output, 73
 CH3P_Polarity_INV, 73
 CH3P_SCTRLP, 73
 CLK_DIV, 73
 CMP_BKIN_Filter, 73
 DeadTimeCH0N, 74
 DeadTimeCH0P, 74
 DeadTimeCH1N, 74
 DeadTimeCH1P, 74
 DeadTimeCH2N, 74
 DeadTimeCH2P, 75
 DeadTimeCH3N, 75
 DeadTimeCH3P, 75
 DebugMode_PWM_out, 75
 FAIL0_INPUT_EN, 76
 FAIL0_INT_EN, 76
 FAIL0_Polarity, 76
 FAIL0_Signal_Sel, 76
 FAIL1_INPUT_EN, 76
 FAIL1_INT_EN, 77
 FAIL1_Polarity, 77
 FAIL1_Signal_Sel, 77
 GPIO_BKIN_Filter, 77
 IO0_PPE, 77
 IO1_PPE, 78
 IO2_PPE, 78
 IO3_PPE, 78
 MCLK_EN, 78
 MCPWM_Auto_ERR_EN, 78
 MCPWM_Cnt_EN, 78
 MCPWM_PERIOD, 79
 MCPWM_REG_UPDATE_INT_EN, 79
 MCPWM_T0_UpdateEN, 79
 MCPWM_T1_UpdateEN, 79
 MCPWM_UpdateInterval, 79
 MCPWM_WorkModeCH0, 79
 MCPWM_WorkModeCH1, 80
 MCPWM_WorkModeCH2, 80
 MCPWM_WorkModeCH3, 80
 Switch_CH0N_CH0P, 80
 Switch_CH1N_CH1P, 81
 Switch_CH2N_CH2P, 81
 Switch_CH3N_CH3P, 81
 T0_Update_INT_EN, 81
 T1_Update_INT_EN, 81
 TH00_Match_INT_EN, 81
 TH01_Match_INT_EN, 82
 TH10_Match_INT_EN, 82
 TH11_Match_INT_EN, 82
 TH20_Match_INT_EN, 82
 TH21_Match_INT_EN, 82
 TH30_Match_INT_EN, 82
 TH31_Match_INT_EN, 83
 TMR0_Match_INT_EN, 83
 TMR1_Match_INT_EN, 83
 TMR2_Match_INT_EN, 83
 TMR3_Match_INT_EN, 83
 TriggerPoint0, 83
 TriggerPoint1, 84
 TriggerPoint2, 84
 TriggerPoint3, 84
 MCPWM_MOE_DISABLE_MASK
 lks32mc08x_mcpwm.h, 306
 MCPWM_MOE_ENABLE_MASK
 lks32mc08x_mcpwm.h, 306
 MCPWM_PERIOD
 MCPWM_InitTypeDef, 79
 MCPWM_REG_UPDATE_INT_EN
 MCPWM_InitTypeDef, 79
 MCPWM_SHADE_IRQ_EN
 lks32mc08x_mcpwm.h, 306
 MCPWM_SHADE_IRQ_IF
 lks32mc08x_mcpwm.h, 307
 MCPWM_StructInit
 lks32mc08x_mcpwm.c, 561
 lks32mc08x_mcpwm.h, 317
 MCPWM_T0_IRQ_EN
 lks32mc08x_mcpwm.h, 307
 MCPWM_T0_IRQ_IF
 lks32mc08x_mcpwm.h, 307
 MCPWM_T0_UpdateEN
 MCPWM_InitTypeDef, 79
 MCPWM_T1_IRQ_EN
 lks32mc08x_mcpwm.h, 307
 MCPWM_T1_IRQ_IF
 lks32mc08x_mcpwm.h, 307
 MCPWM_T1_UpdateEN
 MCPWM_InitTypeDef, 79
 MCPWM_TH00_IRQ_EN

- lks32mc08x_mcpwm.h, 307
- MCPWM.TH00_IRQ_IF
 - lks32mc08x_mcpwm.h, 308
- MCPWM.TH01_IRQ_EN
 - lks32mc08x_mcpwm.h, 308
- MCPWM.TH01_IRQ_IF
 - lks32mc08x_mcpwm.h, 308
- MCPWM.TH10_IRQ_EN
 - lks32mc08x_mcpwm.h, 308
- MCPWM.TH10_IRQ_IF
 - lks32mc08x_mcpwm.h, 308
- MCPWM.TH11_IRQ_EN
 - lks32mc08x_mcpwm.h, 308
- MCPWM.TH11_IRQ_IF
 - lks32mc08x_mcpwm.h, 309
- MCPWM.TH20_IRQ_EN
 - lks32mc08x_mcpwm.h, 309
- MCPWM.TH20_IRQ_IF
 - lks32mc08x_mcpwm.h, 309
- MCPWM.TH21_IRQ_EN
 - lks32mc08x_mcpwm.h, 309
- MCPWM.TH21_IRQ_IF
 - lks32mc08x_mcpwm.h, 309
- MCPWM.TH30_IRQ_EN
 - lks32mc08x_mcpwm.h, 309
- MCPWM.TH30_IRQ_IF
 - lks32mc08x_mcpwm.h, 310
- MCPWM.TH31_IRQ_EN
 - lks32mc08x_mcpwm.h, 310
- MCPWM.TH31_IRQ_IF
 - lks32mc08x_mcpwm.h, 310
- MCPWM.TMR0_IRQ_EN
 - lks32mc08x_mcpwm.h, 310
- MCPWM.TMR0_IRQ_IF
 - lks32mc08x_mcpwm.h, 310
- MCPWM.TMR1_IRQ_EN
 - lks32mc08x_mcpwm.h, 310
- MCPWM.TMR1_IRQ_IF
 - lks32mc08x_mcpwm.h, 311
- MCPWM.TMR2_IRQ_EN
 - lks32mc08x_mcpwm.h, 311
- MCPWM.TMR2_IRQ_IF
 - lks32mc08x_mcpwm.h, 311
- MCPWM.TMR3_IRQ_EN
 - lks32mc08x_mcpwm.h, 311
- MCPWM.TMR3_IRQ_IF
 - lks32mc08x_mcpwm.h, 311
- MCPWM.Trigger_En
 - ADC_InitTypeDef, 12
- MCPWM.UPDATE_REG
 - lks32mc08x_mcpwm.h, 311
- MCPWM.UpdateInterval
 - MCPWM_InitTypeDef, 79
- MCPWM.WorkModeCH0
 - MCPWM_InitTypeDef, 79
- MCPWM.WorkModeCH1
 - MCPWM_InitTypeDef, 80
- MCPWM.WorkModeCH2
 - MCPWM_InitTypeDef, 80
- MCPWM.WorkModeCH3
 - MCPWM_InitTypeDef, 80
- MCU_POWER_3v3
 - lks32mc08x_sys.h, 346
- MCU_POWER_5v0
 - lks32mc08x_sys.h, 346
- MEMORY2PERI
 - lks32mc08x_dma.h, 197
- MOD
 - CAN_TypeDef, 29
- Mode
 - SPI_InitTypeDef, 88
- MSCR
 - I2C_TypeDef, 61
- MST_MODE
 - I2C_InitTypeDef, 59
- MultiDropEna
 - UART_InitTypeDef, 108
- My_CAN_Send_Msg
 - lks32mc08x_can.c, 449
 - lks32mc08x_can.h, 164
- nCofA
 - Stru_TemperatureCof_Def, 94
- nOffsetB
 - Stru_TemperatureCof_Def, 94
- null0
 - CAN_TypeDef, 30
 - lks32mc08x_can.h, 152
- null1
 - CAN_TypeDef, 30
 - lks32mc08x_can.h, 152
- null2
 - CAN_TypeDef, 30
 - lks32mc08x_can.h, 152
- null3
 - CAN_TypeDef, 30
 - lks32mc08x_can.h, 152
- null4
 - lks32mc08x_can.h, 152
- OPA0
 - lks32mc08x_opa.h, 326
- OPA1
 - lks32mc08x_opa.h, 326
- OPA2
 - lks32mc08x_opa.h, 326
- OPA3
 - lks32mc08x_opa.h, 326
- OPA.CLEna
 - OPA_InitTypeDef, 85
- OPA.Gain
 - OPA_InitTypeDef, 85
- OPA.Init
 - lks32mc08x_opa.c, 566
 - lks32mc08x_opa.h, 327
- OPA_InitTypeDef, 84
 - OPA.CLEna, 85

- OPA_Gain, [85](#)
- OPA_OUT
 - lks32mc08x_opa.c, [567](#)
 - lks32mc08x_opa.h, [328](#)
- OPA_StructInit
 - lks32mc08x_opa.c, [568](#)
 - lks32mc08x_opa.h, [329](#)
- OverFlow_IRQ_Ena
 - HALL_InitTypeDef, [55](#)
- ParityMode
 - UART_InitTypeDef, [109](#)
- PDE
 - GPIO_TypeDef, [52](#)
- PDI
 - GPIO_TypeDef, [53](#)
- PDO
 - GPIO_TypeDef, [53](#)
- PERI2MEMORY
 - lks32mc08x_dma.h, [197](#)
- PGA_GAIN_20
 - lks32mc08x_opa.h, [326](#)
- PGA_GAIN_4P25
 - lks32mc08x_opa.h, [326](#)
- PGA_GAIN_6
 - lks32mc08x_opa.h, [326](#)
- PGA_GAIN_9P5
 - lks32mc08x_opa.h, [326](#)
- PIE
 - GPIO_TypeDef, [53](#)
- PLL_DivSel
 - SYS_InitTypeDef, [96](#)
- PLL_ReDiv
 - SYS_InitTypeDef, [96](#)
- PLL_SrcSel
 - SYS_InitTypeDef, [96](#)
- PODE
 - GPIO_TypeDef, [53](#)
- POE
 - GPIO_TypeDef, [53](#)
- POL
 - EXTI_TypeDef, [50](#)
- PORFilter_Ena
 - SYS_InitTypeDef, [96](#)
- POWER_MODE
 - lks32mc08x_sys.h, [346](#)
- prog_m_flag
 - lks32mc08x_flash.c, [499](#)
 - lks32mc08x_flash.h, [230](#)
- ProgramPage
 - lks32mc08x_flash.c, [495](#)
 - lks32mc08x_flash.h, [226](#)
- PUE
 - GPIO_TypeDef, [53](#)
- PWMOutputs
 - lks32mc08x_mcpwm.c, [562](#)
 - lks32mc08x_mcpwm.h, [318](#)
- Quotient
 - stru_DivComponents, [93](#)
- RBSA
 - CAN_TypeDef, [30](#)
- Read_Flash
 - lks32mc08x_flash.c, [496](#)
- Read_More_Flash
 - lks32mc08x_flash.c, [498](#)
 - lks32mc08x_flash.h, [228](#)
- Read_NVR
 - lks32mc08x_nvr.h, [323](#)
- Read_Trim
 - lks32mc08x_nvr.h, [323](#)
- RefVol
 - ADC_InitTypeDef, [13](#)
- RefVol_1V2
 - lks32mc08x_adc.h, [130](#)
- RefVol_2V4
 - lks32mc08x_adc.h, [131](#)
- Remainder
 - stru_DivComponents, [93](#)
- RMC
 - CAN_TypeDef, [30](#)
- RTH
 - IWDG_InitTypeDef, [62](#)
- RX_DATA
 - SPI_TypeDef, [90](#)
- RXD_INV
 - UART_InitTypeDef, [109](#)
- RXERR
 - CAN_TypeDef, [30](#)
- SCR
 - I2C_TypeDef, [61](#)
- SD0
 - ADC_TypeDef, [24](#)
- SD1
 - ADC_TypeDef, [25](#)
- SecSeg_Ch
 - ADC_InitTypeDef, [13](#)
- SELN_CMP_IN
 - lks32mc08x_cmp.h, [175](#)
- SELN_DAC
 - lks32mc08x_cmp.h, [175](#)
- SELN_HALL_MID
 - lks32mc08x_cmp.h, [176](#)
- SELN_REF
 - lks32mc08x_cmp.h, [176](#)
- SELP_CMP_IP0
 - lks32mc08x_cmp.h, [176](#)
- SELP_CMP_IP1
 - lks32mc08x_cmp.h, [176](#)
- SELP_CMP_IP2
 - lks32mc08x_cmp.h, [176](#)
- SELP_CMP_IP3
 - lks32mc08x_cmp.h, [176](#)
- SELP_CMP_IP4
 - lks32mc08x_cmp.h, [177](#)
- SELP_CMP_OPA0IP

- lks32mc08x_cmp.h, 177
- SELP_CMP_OPA0OUT
 - lks32mc08x_cmp.h, 177
- SELP_CMP_OPA1OUT
 - lks32mc08x_cmp.h, 177
- SELP_CMP_OPA2OUT
 - lks32mc08x_cmp.h, 177
- SELP_CMP_OPA3IP
 - lks32mc08x_cmp.h, 177
- SELP_CMP_OPA3OUT
 - lks32mc08x_cmp.h, 178
- SetWakeIO
 - lks32mc08x_sys.c, 580
 - lks32mc08x_sys.h, 356
- SFF_ID_Filter
 - lks32mc08x_can.c, 450
 - lks32mc08x_can.h, 165
- SINCOS
 - lks32mc08x_dsp.h, 208
- SinValue
 - stru_CordicComponents, 92
- SIZE
 - SPI_TypeDef, 90
- SLV_MODE
 - I2C_InitTypeDef, 59
- SoftDelay
 - lks32mc08x_sys.c, 581
- softIE
 - HALL_InitTypeDef, 55
- source/lks32mc08x_adc.c, 419, 428
- source/lks32mc08x_can.c, 431, 451
- source/lks32mc08x_cmp.c, 455, 461
- source/lks32mc08x_dac.c, 462, 467
- source/lks32mc08x_dma.c, 468, 476
- source/lks32mc08x_dsp.c, 477, 486
- source/lks32mc08x_exti.c, 488, 492
- source/lks32mc08x_flash.c, 493, 500
- source/lks32mc08x_gpio.c, 501, 519
- source/lks32mc08x_hall.c, 523, 532
- source/lks32mc08x_i2c.c, 533, 546
- source/lks32mc08x_iwdg.c, 548, 554
- source/lks32mc08x_mcpwm.c, 554, 563
- source/lks32mc08x_opa.c, 565, 569
- source/lks32mc08x_spi.c, 570, 577
- source/lks32mc08x_sys.c, 578, 598
- source/lks32mc08x_tim.c, 603, 620
- source/lks32mc08x_tmp.c, 624, 628
- source/lks32mc08x_uart.c, 628, 638
- SPI_ClearIRQFlag
 - lks32mc08x_spi.c, 571
 - lks32mc08x_spi.h, 335
- SPI_DMA_DISABLE
 - lks32mc08x_spi.h, 332
- SPI_DMA_ENABLE
 - lks32mc08x_spi.h, 332
- SPI_Duplex
 - lks32mc08x_spi.h, 334
- SPI_FIRSTSEND_LSB
 - lks32mc08x_spi.h, 332
- SPI_FIRSTSEND_MSB
 - lks32mc08x_spi.h, 332
- SPI_Full
 - lks32mc08x_spi.h, 334
- SPI_GetIRQFlag
 - lks32mc08x_spi.c, 572
 - lks32mc08x_spi.h, 336
- SPI_Half_Rx
 - lks32mc08x_spi.h, 334
- SPI_Half_Tx
 - lks32mc08x_spi.h, 334
- SPI_IF_DataOver
 - lks32mc08x_spi.h, 333
- SPI_IF_SSErr
 - lks32mc08x_spi.h, 333
- SPI_IF_TranDone
 - lks32mc08x_spi.h, 333
- SPI_Init
 - lks32mc08x_spi.c, 573
 - lks32mc08x_spi.h, 337
- SPI_InitTypeDef, 85
 - BaudRate, 86
 - ByteLength, 86
 - CPHA, 87
 - CPOL, 87
 - DataOrder, 87
 - DMA, 87
 - Duplex, 87
 - EN, 87
 - IRQEna, 88
 - Mode, 88
 - SS, 88
 - Trig, 88
- SPI_IRQEna_DataOver
 - lks32mc08x_spi.h, 333
- SPI_IRQEna_Enable
 - lks32mc08x_spi.h, 333
- SPI_IRQEna_SSErr
 - lks32mc08x_spi.h, 333
- SPI_IRQEna_TranDone
 - lks32mc08x_spi.h, 334
- SPI_Master
 - lks32mc08x_spi.h, 334
- SPI_Mode
 - lks32mc08x_spi.h, 334
- SPI_ReadData
 - lks32mc08x_spi.c, 574
 - lks32mc08x_spi.h, 338
- SPI_SendData
 - lks32mc08x_spi.c, 575
 - lks32mc08x_spi.h, 339
- SPI_Slave
 - lks32mc08x_spi.h, 334
- SPI_StructInit
 - lks32mc08x_spi.c, 576
 - lks32mc08x_spi.h, 340
- SPI_TypeDef, 89

CFG, 89
 DIV, 89
 IE, 90
 RX_DATA, 90
 SIZE, 90
 TX_DATA, 90
 SR
 CAN_TypeDef, 31
 SS
 SPI_InitTypeDef, 88
 STOP_IE
 I2C_InitTypeDef, 60
 StopBits
 UART_InitTypeDef, 109
 stru_CordicComponents, 90
 ArctanValue, 91
 ArctanX, 91
 ArctanY, 91
 CosValue, 91
 SinValue, 92
 Theta, 92
 stru_DivComponents, 92
 Dividend, 93
 Divisor, 93
 Quotient, 93
 Remainder, 93
 Stru_TemperatureCof_Def, 93
 nCofA, 94
 nOffsetB, 94
 STT
 UART_TypeDef, 112
 Switch2HRC
 lks32mc08x_sys.c, 582
 lks32mc08x_sys.h, 357
 Switch2PLL
 lks32mc08x_sys.c, 583
 lks32mc08x_sys.h, 358
 Switch_CH0N_CH0P
 MCPWM_InitTypeDef, 80
 Switch_CH1N_CH1P
 MCPWM_InitTypeDef, 81
 Switch_CH2N_CH2P
 MCPWM_InitTypeDef, 81
 Switch_CH3N_CH3P
 MCPWM_InitTypeDef, 81
 SWT
 ADC_TypeDef, 25
 SYS_AnalogModule_ADC
 lks32mc08x_sys.h, 346
 SYS_AnalogModule_BGP
 lks32mc08x_sys.h, 346
 SYS_AnalogModule_CMP0
 lks32mc08x_sys.h, 347
 SYS_AnalogModule_CMP1
 lks32mc08x_sys.h, 347
 SYS_AnalogModule_DAC
 lks32mc08x_sys.h, 347
 SYS_AnalogModule_OPA0
 lks32mc08x_sys.h, 347
 SYS_AnalogModule_OPA1
 lks32mc08x_sys.h, 347
 SYS_AnalogModule_OPA2
 lks32mc08x_sys.h, 347
 SYS_AnalogModule_OPA3
 lks32mc08x_sys.h, 348
 SYS_AnalogModule_PLL
 lks32mc08x_sys.h, 348
 SYS_AnalogModule_RCH
 lks32mc08x_sys.h, 348
 SYS_AnalogModule_TMP
 lks32mc08x_sys.h, 348
 SYS_AnalogModule_XTAL
 lks32mc08x_sys.h, 348
 SYS_AnalogModuleClockCmd
 lks32mc08x_sys.c, 584
 lks32mc08x_sys.h, 359
 SYS_ClearRst
 lks32mc08x_sys.c, 586
 lks32mc08x_sys.h, 361
 SYS_CLK_SEL_CLK_HS
 lks32mc08x_sys.h, 348
 SYS_CLK_SEL_PLL
 lks32mc08x_sys.h, 349
 SYS_Clk_SPIDiv1
 lks32mc08x_sys.h, 349
 SYS_Clk_SPIDiv2
 lks32mc08x_sys.h, 349
 SYS_Clk_SPIDiv4
 lks32mc08x_sys.h, 349
 SYS_Clk_SPIDiv8
 lks32mc08x_sys.h, 349
 SYS_Clk_UARTDiv1
 lks32mc08x_sys.h, 349
 SYS_Clk_UARTDiv2
 lks32mc08x_sys.h, 350
 SYS_Clk_UARTDiv4
 lks32mc08x_sys.h, 350
 SYS_Clk_UARTDiv8
 lks32mc08x_sys.h, 350
 SYS_EnablePORFilter
 lks32mc08x_sys.c, 587
 SYS_FallSleep
 lks32mc08x_sys.c, 588
 lks32mc08x_sys.h, 362
 SYS_FeedDog
 lks32mc08x_sys.c, 589
 lks32mc08x_sys.h, 363
 SYS_GetRstSource
 lks32mc08x_sys.c, 590
 lks32mc08x_sys.h, 364
 SYS_Init
 lks32mc08x_sys.c, 591
 lks32mc08x_sys.h, 365
 SYS_InitTypeDef, 94
 Clk_DivSPI, 95
 Clk_DivUART, 95

Clk_FEN, 95
 Clk_Sel, 96
 PLL_DivSel, 96
 PLL_ReDiv, 96
 PLL_SrcSel, 96
 PORFilter_Ena, 96
 WDT_Ena, 97
 SYS_Module_CAN
 lks32mc08x_sys.h, 350
 SYS_Module_DSP
 lks32mc08x_sys.h, 350
 SYS_Module_HALL
 lks32mc08x_sys.h, 350
 SYS_Module_I2C
 lks32mc08x_sys.h, 351
 SYS_Module_MCPWM
 lks32mc08x_sys.h, 351
 SYS_Module_SPI
 lks32mc08x_sys.h, 351
 SYS_Module_TIMER
 lks32mc08x_sys.h, 351
 SYS_Module_UART0
 lks32mc08x_sys.h, 351
 SYS_Module_UART1
 lks32mc08x_sys.h, 351
 SYS_ModuleClockCmd
 lks32mc08x_sys.c, 592
 lks32mc08x_sys.h, 366
 SYS_PLLREDIV_1
 lks32mc08x_sys.h, 352
 SYS_PLLREDIV_4
 lks32mc08x_sys.h, 352
 SYS_PLLSRSEL_CRYSTAL
 lks32mc08x_sys.h, 352
 SYS_PLLSRSEL_RCH
 lks32mc08x_sys.h, 352
 SYS_ResetIO
 lks32mc08x_sys.h, 367
 SYS_ResetPeripheral
 lks32mc08x_sys.h, 368
 SYS_ResetPeripheralInternal
 lks32mc08x_sys.h, 368
 SYS_RstSrc_HPOR
 lks32mc08x_sys.h, 352
 SYS_RstSrc_KEYBOARD
 lks32mc08x_sys.h, 352
 SYS_RstSrc_LPOR
 lks32mc08x_sys.h, 353
 SYS_RstSrc_WDT
 lks32mc08x_sys.h, 353
 SYS_SetWakeInterval
 lks32mc08x_sys.c, 593
 lks32mc08x_sys.h, 368
 SYS_SoftResetModule
 lks32mc08x_sys.c, 595
 lks32mc08x_sys.h, 369
 SYS_StructInit
 lks32mc08x_sys.c, 596
 lks32mc08x_sys.h, 371
 SYS_VolSelModule
 lks32mc08x_sys.h, 372
 SYS_VolSelModuleEnableIRQ
 lks32mc08x_sys.h, 372
 SYS_VolSelModuleIRQ
 lks32mc08x_sys.h, 372
 SYS_WakeInterval_025s
 lks32mc08x_sys.h, 353
 SYS_WakeInterval_05s
 lks32mc08x_sys.h, 353
 SYS_WakeInterval_16s
 lks32mc08x_sys.h, 353
 SYS_WakeInterval_1s
 lks32mc08x_sys.h, 353
 SYS_WakeInterval_2s
 lks32mc08x_sys.h, 354
 SYS_WakeInterval_32s
 lks32mc08x_sys.h, 354
 SYS_WakeInterval_4s
 lks32mc08x_sys.h, 354
 SYS_WakeInterval_8s
 lks32mc08x_sys.h, 354
 SYS_WatchDogCmd
 lks32mc08x_sys.c, 597
 lks32mc08x_sys.h, 372
 SYS_WD_TimeOut2s
 lks32mc08x_sys.h, 354
 SYS_WD_TimeOut4s
 lks32mc08x_sys.h, 354
 SYS_WD_TimeOut64s
 lks32mc08x_sys.h, 355
 SYS_WD_TimeOut8s
 lks32mc08x_sys.h, 355
 T0_Update_INT_EN
 MCPWM_InitTypeDef, 81
 T1_Update_INT_EN
 MCPWM_InitTypeDef, 81
 TC_IE
 I2C_InitTypeDef, 60
 TCLK
 CMP_TypeDef, 43
 TempSensor_Init
 lks32mc08x_tmp.c, 626
 lks32mc08x_tmp.h, 402
 TH
 HALL_TypeDef, 57
 TIM_ECDTypeDef, 100
 TIM_TimerTypeDef, 106
 TH00_Match_INT_EN
 MCPWM_InitTypeDef, 81
 TH01_Match_INT_EN
 MCPWM_InitTypeDef, 82
 TH10_Match_INT_EN
 MCPWM_InitTypeDef, 82
 TH11_Match_INT_EN
 MCPWM_InitTypeDef, 82
 TH20_Match_INT_EN

MCPWM_InitTypeDef, 82
 TH21_Match_INT_EN
 MCPWM_InitTypeDef, 82
 TH30_Match_INT_EN
 MCPWM_InitTypeDef, 82
 TH31_Match_INT_EN
 MCPWM_InitTypeDef, 83
 Theta
 stru_CordicComponents, 92
 ThrSeg_Ch
 ADC_InitTypeDef, 13
 TIM_ClearIRQFlag
 lks32mc08x_tim.c, 607
 lks32mc08x_tim.h, 385
 TIM_Clk_Div1
 lks32mc08x_tim.h, 381
 TIM_Clk_Div2
 lks32mc08x_tim.h, 381
 TIM_Clk_Div4
 lks32mc08x_tim.h, 381
 TIM_Clk_Div8
 lks32mc08x_tim.h, 381
 TIM_ECD_GetCount
 lks32mc08x_tim.c, 608
 lks32mc08x_tim.h, 386
 TIM_ECDCmd
 lks32mc08x_tim.c, 609
 lks32mc08x_tim.h, 387
 TIM_ECDInit
 lks32mc08x_tim.c, 610
 lks32mc08x_tim.h, 388
 TIM_ECDInitTypeDef, 97
 ECD_ClockDiv, 98
 ECD_Filter0, 98
 ECD_Filter1, 98
 ECD_IRQEna, 98
 ECD_Mode, 98
 ECD_TH, 99
 TIM_ECDStructInit
 lks32mc08x_tim.c, 611
 lks32mc08x_tim.h, 390
 TIM_ECDTypeDef, 99
 CFG, 100
 CNT, 100
 TH, 100
 TIM_GetIRQFlag
 lks32mc08x_tim.c, 612
 lks32mc08x_tim.h, 391
 TIM_Timer_GetCMPT0
 lks32mc08x_tim.c, 613
 lks32mc08x_tim.h, 392
 TIM_Timer_GetCMPT1
 lks32mc08x_tim.c, 614
 lks32mc08x_tim.h, 393
 TIM_Timer_GetCount
 lks32mc08x_tim.c, 615
 lks32mc08x_tim.h, 394
 TIM_TimerCmd
 lks32mc08x_tim.c, 616
 lks32mc08x_tim.h, 394
 TIM_TimerInit
 lks32mc08x_tim.c, 617
 lks32mc08x_tim.h, 396
 TIM_TimerInitTypeDef, 100
 EN, 101
 Timer_CH0_CapMode, 101
 Timer_CH0_WorkMode, 102
 Timer_CH0Output, 102
 Timer_CH1_CapMode, 102
 Timer_CH1_WorkMode, 103
 Timer_CH1Output, 103
 Timer_ClockDiv, 103
 Timer_CMP0, 104
 Timer_CMP1, 104
 Timer_Filter0, 104
 Timer_Filter1, 104
 Timer_IRQEna, 104
 Timer_TH, 105
 TIM_TimerStrutInit
 lks32mc08x_tim.c, 619
 lks32mc08x_tim.h, 397
 TIM_TimerTypeDef, 105
 CFG, 106
 CMPT0, 106
 CMPT1, 106
 CNT, 106
 EVT, 106
 TH, 106
 TIMER_CapMode_Fall
 lks32mc08x_tim.h, 381
 TIMER_CapMode_None
 lks32mc08x_tim.h, 381
 TIMER_CapMode_Rise
 lks32mc08x_tim.h, 382
 TIMER_CapMode_RiseFall
 lks32mc08x_tim.h, 382
 Timer_CH0_CapMode
 TIM_TimerInitTypeDef, 101
 Timer_CH0_WorkMode
 TIM_TimerInitTypeDef, 102
 Timer_CH0Output
 TIM_TimerInitTypeDef, 102
 Timer_CH1_CapMode
 TIM_TimerInitTypeDef, 102
 Timer_CH1_WorkMode
 TIM_TimerInitTypeDef, 103
 Timer_CH1Output
 TIM_TimerInitTypeDef, 103
 Timer_ClockDiv
 TIM_TimerInitTypeDef, 103
 Timer_CMP0
 TIM_TimerInitTypeDef, 104
 Timer_CMP1
 TIM_TimerInitTypeDef, 104
 Timer_Filter0
 TIM_TimerInitTypeDef, 104

Timer_Filter1
 TIM_TimerInitTypeDef, 104
 TIMER_IF_CH0
 lks32mc08x_tim.h, 382
 TIMER_IF_CH1
 lks32mc08x_tim.h, 382
 TIMER_IF_ZERO
 lks32mc08x_tim.h, 382
 Timer_IRQEna
 TIM_TimerInitTypeDef, 104
 Timer_IRQEna_All
 lks32mc08x_tim.h, 383
 Timer_IRQEna_CH0
 lks32mc08x_tim.h, 383
 Timer_IRQEna_CH1
 lks32mc08x_tim.h, 383
 Timer_IRQEna_None
 lks32mc08x_tim.h, 383
 Timer_IRQEna_Zero
 lks32mc08x_tim.h, 383
 Timer_IRQEnaDef
 lks32mc08x_tim.h, 383
 TIMER_OPMode_CAP
 lks32mc08x_tim.h, 382
 TIMER_OPMode_CMP
 lks32mc08x_tim.h, 383
 Timer_TH
 TIM_TimerInitTypeDef, 105
 TMR0_Match_INT_EN
 MCPWM_InitTypeDef, 83
 TMR1_Match_INT_EN
 MCPWM_InitTypeDef, 83
 TMR2_Match_INT_EN
 MCPWM_InitTypeDef, 83
 TMR3_Match_INT_EN
 MCPWM_InitTypeDef, 83
 TRG
 ADC_TypeDef, 25
 Trig
 SPI_InitTypeDef, 88
 Trigger_Cnt
 ADC_InitTypeDef, 13
 Trigger_Mode
 ADC_InitTypeDef, 14
 TriggerPoint0
 MCPWM_InitTypeDef, 83
 TriggerPoint1
 MCPWM_InitTypeDef, 84
 TriggerPoint2
 MCPWM_InitTypeDef, 84
 TriggerPoint3
 MCPWM_InitTypeDef, 84
 TX_DATA
 SPI_TypeDef, 90
 TXD_INV
 UART_InitTypeDef, 109
 TXERR
 CAN_TypeDef, 31
 TXRX0
 CAN_TypeDef, 31
 TXRX1
 CAN_TypeDef, 31
 TXRX2
 CAN_TypeDef, 31
 TXRX3
 CAN_TypeDef, 31
 TXRX4
 CAN_TypeDef, 32
 TXRX5
 CAN_TypeDef, 32
 TXRX6
 CAN_TypeDef, 32
 TXRX7
 CAN_TypeDef, 32
 TXRX8
 CAN_TypeDef, 32
 TXRX9
 CAN_TypeDef, 32
 TXRXA
 CAN_TypeDef, 33
 TXRXB
 CAN_TypeDef, 33
 TXRXC
 CAN_TypeDef, 33
 UART_ClearIRQFlag
 lks32mc08x_uart.c, 629
 lks32mc08x_uart.h, 410
 UART_FIRSTSEND_LSB
 lks32mc08x_uart.h, 406
 UART_FIRSTSEND_MSB
 lks32mc08x_uart.h, 406
 UART_GetIRQFlag
 lks32mc08x_uart.c, 630
 lks32mc08x_uart.h, 411
 UART_IF_CheckError
 lks32mc08x_uart.h, 406
 UART_IF_RcvOver
 lks32mc08x_uart.h, 406
 UART_IF_SendBufEmpty
 lks32mc08x_uart.h, 407
 UART_IF_SendOver
 lks32mc08x_uart.h, 407
 UART_IF_StopError
 lks32mc08x_uart.h, 407
 UART_Init
 lks32mc08x_uart.c, 631
 lks32mc08x_uart.h, 412
 UART_InitTypeDef, 107
 BaudRate, 107
 Bit9Value, 107
 FirstSend, 108
 IRQEna, 108
 Match485Addr, 108
 MultiDropEna, 108
 ParityMode, 109
 RXD_INV, 109

StopBits, 109
TXD_INV, 109
WordLength, 109
UART_IRQEna.CheckError
lks32mc08x_uart.h, 407
UART_IRQEna.RcvOver
lks32mc08x_uart.h, 407
UART_IRQEna.RX_DMA_RE
lks32mc08x_uart.h, 407
UART_IRQEna.SendBuffEmpty
lks32mc08x_uart.h, 408
UART_IRQEna.SendOver
lks32mc08x_uart.h, 408
UART_IRQEna.StopError
lks32mc08x_uart.h, 408
UART_IRQEna.TX_BUF_DMA_RE
lks32mc08x_uart.h, 408
UART_IRQEna.TX_DMA_RE
lks32mc08x_uart.h, 408
UART_Parity_EVEN
lks32mc08x_uart.h, 409
UART_Parity_NO
lks32mc08x_uart.h, 409
UART_Parity_ODD
lks32mc08x_uart.h, 409
UART_ParityMode
lks32mc08x_uart.h, 409
UART_ReadData
lks32mc08x_uart.c, 633
lks32mc08x_uart.h, 413
UART_SendAddr_485
lks32mc08x_uart.c, 634
lks32mc08x_uart.h, 414
UART_SendData
lks32mc08x_uart.c, 635
lks32mc08x_uart.h, 415
UART_SendData_485
lks32mc08x_uart.c, 636
lks32mc08x_uart.h, 416
UART_STOPBITS_1b
lks32mc08x_uart.h, 408
UART_STOPBITS_2b
lks32mc08x_uart.h, 409
UART_StructInit
lks32mc08x_uart.c, 637
lks32mc08x_uart.h, 417
UART_TypeDef, 110
ADR, 111
BUFF, 111
CTRL, 111
DIVH, 111
DIVL, 111
IE, 111
IF, 111
INV, 112
STT, 112
UART_WORDLENGTH_7b
lks32mc08x_uart.h, 409
UART_WORDLENGTH_8b
lks32mc08x_uart.h, 409
UTIMER_Trigger_En
ADC_InitTypeDef, 14
WAKE_InitStruct
lks32mc08x_sys.c, 598
WAKE_InitTypeDef, 112
AFE_REG5_RECORD, 112
AFE_REG6_RECORD, 112
CLK_CFG_RECORD, 112
WAKEIO_P0_0
lks32mc08x_sys.h, 355
WAKEIO_P0_1
lks32mc08x_sys.h, 355
WAKEIO_P1_0
lks32mc08x_sys.h, 355
WAKEIO_P1_1
lks32mc08x_sys.h, 355
WAKETRIG_HIGH
lks32mc08x_sys.h, 356
WAKETRIG_LOW
lks32mc08x_sys.h, 356
WDG_EN
IWDG_InitTypeDef, 62
WDT_Ena
SYS_InitTypeDef, 97
WDT_RTH_2S
lks32mc08x_iwdg.h, 287
WDT_RTH_4S
lks32mc08x_iwdg.h, 287
WDT_RTH_64S
lks32mc08x_iwdg.h, 287
WDT_RTH_8S
lks32mc08x_iwdg.h, 288
WIDTH
HALL_TypeDef, 57
WordLength
UART_InitTypeDef, 109