

RTL8762D 外设参考手册

v1.0

2020/5/22

修订历史(Revision History)

日期	版本	修改
2020/5/20	V1.0	First Release version

目录

修订历史(Revision History)	2
目录.....	3
图示列表	16
表格列表	17
1 外设固件概述	18
2 模数转换器(ADC).....	19
2.1 ADC 寄存器结构	19
2.2 ADC 库函数	20
2.2.1 函数 ADC_DeInit.....	21
2.2.2 函数 ADC_Init	21
2.2.3 函数 ADC_StructInit	24
2.2.4 函数 ADC_INTConfig	25
2.2.5 函数 ADC_Cmd	25
2.2.6 函数 ADC_BypassCmd.....	26
2.2.7 函数 ADC_ReadRawData.....	26
2.2.8 函数 ADC_GetFIFOData.....	27
2.2.9 函数 ADC_GetAvgData	27
2.2.10 函数 ADC_SchTableConfig	27
2.2.11 函数 ADC_PowerSupplyConfig	28
2.2.12 函数 ADC_FastPowerSupplyConfig.....	28
2.2.13 函数 ADC_ClearINTPendingBit	29
2.2.14 函数 ADC_GetINTStatus	29
2.2.15 函数 ADC_GetAllINTStatus	29
2.2.16 函数 ADC_BitMapConfig.....	30
2.2.17 函数 ADC_ReadFIFOData.....	30
2.2.18 函数 ADC_GetFIFODataLen	31
2.2.19 函数 ADC_ClearFIFO	31
2.2.20 函数 ADC_ManualPowerOnCmd.....	31

2.2.21	函数 ADC_WriteFIFOCmd	32
2.2.22	函数 ADC_DelayUs.....	32
2.2.23	函数 ADC_CalibrationInit.....	32
2.2.24	函数 ADC_GetVoltage.....	33
2.2.25	函数 ADC_GetResistance.....	34
3	编解码器(CODEC)	35
3.1	CODEC 寄存器架构	35
3.2	CODEC 库函数	36
3.2.1	函数 CODEC_DeInit	37
3.2.2	函数 CODEC_Init.....	37
3.2.3	函数 CODEC_StructInit.....	40
3.2.4	函数 CODEC_EQInit.....	41
3.2.5	函数 CODEC_EQStructInit	42
3.2.6	函数 CODEC_MICBIASCmd.....	43
3.2.7	函数 CODEC_Reset.....	44
3.2.8	函数 CODEC_SetMICBIAS.....	44
4	直接内存存取控制器(GDMA).....	45
4.1	GDMA 寄存器架构	45
4.2	GDMA 库函数	49
4.2.1	函数 GDMA_DeInit	50
4.2.2	函数 GDMA_Init	50
4.2.3	函数 GDMA_StructInit.....	55
4.2.4	函数 GDMA_Cmd	55
4.2.5	函数 GDMA_INTConfig.....	56
4.2.6	函数 GDMA_ClearINTPendingBit	56
4.2.7	函数 GDMA_GetChannelStatus	57
4.2.8	函数 GDMA_GetTransferINTStatus.....	57
4.2.9	函数 GDMA_ClearAllTypeINT.....	57
4.2.10	函数 GDMA_SetSourceAddress.....	58
4.2.11	函数 GDMA_SetDestinationAddress	58

4.2.12	函数 GDMA_SetBufferSize	59
4.2.13	函数 GDMA_SuspendCmd.....	59
4.2.14	函数 GDMA_GetFIFOStatus.....	59
4.2.15	函数 GDMA_GetTransferLen.....	60
4.2.16	函数 GDMA_SetLLPAddress	60
4.2.17	函数 GDMA_GetSuspendChannelStatus	61
4.2.18	函数 GDMA_GetSuspendCmdStatus.....	61
5	通用输入/输出(GPIO).....	62
5.1	GPIO 寄存器结构	62
5.2	GPIO 库函数	63
5.2.1	函数 GPIO_Delnit	63
5.2.2	函数 GPIO_GetPin	64
5.2.3	函数 GPIO_Init.....	64
5.2.4	函数 GPIO_StructInit	66
5.2.5	函数 GPIO_INTConfig	67
5.2.6	函数 GPIO_ClearINTPendingBit.....	68
5.2.7	函数 GPIO_MaskINTConfig	68
5.2.8	函数 GPIO_ReadInputDataBit	69
5.2.9	函数 GPIO_ReadInputData.....	69
5.2.10	函数 GPIO_ReadOutputDataBit.....	70
5.2.11	函数 GPIO_ReadOutputData	70
5.2.12	函数 GPIO_SetBits	70
5.2.13	函数 GPIO_ResetBits	71
5.2.14	函数 GPIO_WriteBit.....	71
5.2.15	函数 GPIO_Write	72
5.2.16	函数 GPIO_GetINTStatus	72
5.2.17	函数 GPIO_GetNum.....	72
5.2.18	函数 GPIO_Debounce_Time	73
5.2.19	函数 GPIO_DBClkCmd.....	73
6	内部集成电路(I2C)	75

6.1	I2C 寄存器架构	75
6.2	I2C 库函数	77
6.2.1	函数 I2C_DeInit	77
6.2.2	函数 I2C_Init.....	78
6.2.3	函数 I2C_StructInit.....	79
6.2.4	函数 I2C_Cmd.....	79
6.2.5	函数 I2C_MasterWrite	80
6.2.6	函数 I2C_MasterRead	80
6.2.7	函数 I2C_RepeatRead	80
6.2.8	函数 I2C_INTConfig	81
6.2.9	函数 I2C_ClearINTPendingBit.....	82
6.2.10	函数 I2C_SetSlaveAddress	82
6.2.11	函数 I2C_SendCmd	83
6.2.12	函数 I2C_ReceiveData	83
6.2.13	函数 I2C_GetRxFIFOLen.....	84
6.2.14	函数 I2C_GetTxFIFOLen.....	84
6.2.15	函数 I2C_ClearAllINT	84
6.2.16	函数 I2C_GetFlagState.....	85
6.2.17	函数 I2C_CheckEvent.....	85
6.2.18	函数 I2C_GetINTStatus	86
6.2.19	函数 I2C_GDMACmd	87
7	正交解调(QDEC)	88
7.1	QDEC 寄存器架构	88
7.2	QDEC 库函数	88
7.2.1	函数 QDEC_Init.....	89
7.2.2	函数 QDEC_DeInit	92
7.2.3	函数 QDEC_StructInit	92
7.2.4	函数 QDEC_INTConfig	92
7.2.5	函数 QDEC_GetFlagState	93
7.2.6	函数 QDEC_INTMask.....	94

7.2.7 函数 QDEC_Cmd.....	94
7.2.8 函数 QDEC_ClearINTPendingBit.....	95
7.2.9 函数 QDEC_GetAxisDirection.....	96
7.2.10 函数 QDEC_GetAxisCount	96
7.2.11 函数 QDEC_CounterPauseCmd	97
8 管脚分配定义(PAD).....	98
8.1 PAD 库函数	98
8.1.1 函数 Pad_Config.....	98
8.1.2 函数 Pad_OutputControlValue	100
8.1.3 函数 Pad_OutputEnableValue	101
8.1.4 函数 Pad_PullEnableValue	101
8.1.5 函数 Pad_PullUpOrDownValue.....	102
8.1.6 函数 Pad_PullConfigValue.....	102
8.1.7 函数 Pad_WakeupEnableValue.....	103
8.1.8 函数 Pad_WakeupPolarityValue	103
8.1.9 函数 Pad_PowerOrShutDownValue.....	103
8.1.10 函数 Pad_ControlSelectValue.....	104
8.1.11 函数 Pad_WKDebounceConfig	104
8.1.12 函数 Pad_WakeupInterruptValue	105
8.1.13 函数 Pad_ClearWakeupINTPendingBit.....	105
8.1.14 函数 System_WakeUpInterruptValue	106
8.1.15 函数 System_WakeUpDebounceTime	106
8.1.16 函数 System_WakeUpPinEnable	106
8.1.17 函数 System_WakeUpPinDisable	107
9 引脚复用(PINMUX).....	107
9.1 PINMUX 库函数	107
9.1.1 函数 Pinmux_Config.....	108
9.1.2 函数 Pinmux_Deinit	111
9.1.3 函数 Pinmux_Reset	112
10 低功耗比较器(LPC).....	113

10.1	LPC 寄存器结构	113
10.2	LPC 库函数	113
10.2.1	函数 LPC_Init	114
10.2.2	函数 LPC_StructInit.....	116
10.2.3	函数 LPC_Cmd	117
10.2.4	函数 LPC_CounterCmd	117
10.2.5	函数 LPC_ResetCounter.....	118
10.2.6	函数 LPC_SetCompValue.....	118
10.2.7	函数 LPC_GetCompValue	118
10.2.8	函数 LPC_GetCounter.....	119
10.2.9	函数 LPC_INTConfig.....	119
10.2.10	函数 LPC_ClearINTPendingBit	120
10.2.11	函数 LPC_GetINTStatus	120
11	实时时钟(RTC)	122
11.1	RTC 寄存器结构	122
11.2	RTC 库函数	122
11.2.1	函数 RTC_DeInit.....	123
11.2.2	函数 RTC_SetPrescaler	123
11.2.3	函数 RTC_SetPreCompValue	124
11.2.4	函数 RTC_SetCompValue.....	124
11.2.5	函数 RTC_Cmd	125
11.2.6	函数 RTC_MaskINTConfig.....	125
11.2.7	函数 RTC_INTConfig.....	126
11.2.8	函数 RTC_NvCmd.....	127
11.2.9	函数 RTC_ClearINTPendingBit	127
11.2.10	函数 RTC_GetINTStatus	128
11.2.11	函数 RTC_SystemWakeupConfig	128
11.2.12	函数 RTC_ResetCounter	128
11.2.13	函数 RTC_ResetPrescalerCounter	129
11.2.14	函数 RTC_GetCounter	129

11.2.15	函数 RTC_GetPreCounter	130
11.2.16	函数 RTC_GetCompValue	130
11.2.17	函数 RTC_GetPreCompValue	131
11.2.18	函数 RTC_ClearCompINT	131
11.2.19	函数 RTC_ClearOverFlowINT	131
11.2.20	函数 RTC_ClearTickINT	132
12	串型外设接口(SPI)	133
12.1	SPI 寄存器架构	133
12.2	SPI 库函数	134
12.2.1	函数 SPI_DelInit	135
12.2.2	函数 SPI_Init	135
12.2.3	函数 SPI_StructInit	139
12.2.4	函数 SPI_Cmd	140
12.2.5	函数 SPI_SendBuffer	140
12.2.6	函数 SPI_SendWord	141
12.2.7	函数 SPI_SendHalfWord	141
12.2.8	函数 SPI_INTConfig	141
12.2.9	函数 SPI_ClearINTPendingBit	142
12.2.10	函数 SPI_SendData	143
12.2.11	函数 SPI_ReceiveData	143
12.2.12	函数 SPI_GetRxFIFOLen	143
12.2.13	函数 SPI_GetTxFIFOLen	144
12.2.14	函数 SPI_ChangeDirection	144
12.2.15	函数 SPI_SetReadLen	145
12.2.16	函数 SPI_SetCSNumber	145
12.2.17	函数 SPI_GetFlagState	145
12.2.18	函数 SPI_GetINTStatus	146
12.2.19	函数 SPI_GDMACmd	147
13	三线 SPI(SPI3WIRE)	148
13.1	SPI3WIRE 寄存器结构	148

13.2	SPI3WIRE 库函数.....	148
13.2.1	函数 SPI3WIRE_DelInit	149
13.2.2	函数 SPI3WIRE_Init.....	149
13.2.3	函数 SPI3WIRE_StructInit.....	151
13.2.4	函数 SPI3WIRE_Cmd.....	151
13.2.5	函数 SPI3WIRE_SetResyncTime.....	152
13.2.6	函数 SPI3WIRE_ResyncSignalCmd.....	152
13.2.7	函数 SPI3WIRE_INTConfig	152
13.2.8	函数 SPI3WIRE_GetFlagStatus.....	153
13.2.9	函数 SPI3WIRE_ClearINTPendingBit.....	153
13.2.10	函数 SPI3WIRE_GetRxDataLen.....	154
13.2.11	函数 SPI3WIRE_ClearRxFIFO.....	154
13.2.12	函数 SPI3WIRE_ClearRxDataLen	154
13.2.13	函数 SPI3WIRE_StartWrite	155
13.2.14	函数 SPI3WIRE_StartRead	155
13.2.15	函数 SPI3WIRE_ReadBuf	156
14	定时器和脉冲宽度调制(TIM&PWM).....	157
14.1	TIM 寄存器结构	157
14.2	TIM 库函数	157
14.2.1	函数 TIM_DelInit.....	158
14.2.2	函数 TIM_TimeBaseInit	158
14.2.3	函数 TIM_StructInit	160
14.2.4	函数 TIM_Cmd.....	161
14.2.5	函数 TIM_ChangePeriod.....	161
14.2.6	函数 TIM_INTConfig	162
14.2.7	函数 TIM_GetCurrentValue.....	162
14.2.8	函数 TIM_GetINTStatus.....	162
14.2.9	函数 TIM_ClearINT	163
14.2.10	函数 TIM_PWMChangeFreqAndDuty.....	163
14.2.11	函数 PWM_Deadzone_EMStop.....	164

15 通用异步接收发送端(UART)	165
15.1 UART 寄存器架构	165
15.2 UART 库函数	165
15.2.1 函数 UART_DeInit	166
15.2.2 函数 UART_Init	166
15.2.3 函数 UART_StructInit	170
15.2.4 函数 UART_ReceiveData	170
15.2.5 函数 UART_SendData	171
15.2.6 函数 UART_INTConfig	171
15.2.7 函数 UART_GetFlagState	172
15.2.8 函数 UART_ClearTxFifo	172
15.2.9 函数 UART_LoopBackCmd	173
15.2.10 函数 UART_ClearRxFifo	173
15.2.11 函数 UART_GetTxFIFOLen	174
15.2.12 函数 UART_GetRxFIFOLen	174
15.2.13 函数 UART_ReceiveByte	174
15.2.14 函数 UART_SendByte	175
15.2.15 函数 UART_GetID	175
16 按键扫描(KEYSCAN)	177
16.1 KEYCAN 寄存器架构	177
16.2 KEYSCAN 库函数	177
16.2.1 函数 KeyScan_Init	178
16.2.2 函数 KeyScan_DeInit	180
16.2.3 函数 KeyScan_StructInit	180
16.2.4 函数 KeyScan_INTConfig	181
16.2.5 函数 KeyScan_INTMask	181
16.2.6 函数 KeyScan_Read	182
16.2.7 函数 KeyScan_Cmd	182
16.2.8 函数 KeyScan_GetFifoDataNum	183
16.2.9 函数 KeyScan_ClearINTPendingBit	183

16.2.10	函数 KeyScan_ClearFlags	183
16.2.11	函数 KeyScan_GetFlagState	184
16.2.12	函数 KeyScan_FilterDataConfig	184
16.2.13	函数 KeyScan_debounceConfig	185
16.2.14	函数 KeyScan_ReadFifoData	185
17	红外(IR)	187
17.1	IR 寄存器结构	187
17.2	IR 库函数	188
17.2.1	函数 IR_DeInit	188
17.2.2	函数 IR_Init	189
17.2.3	函数 IR_StructInit	192
17.2.4	函数 IR_Cmd	192
17.2.5	函数 IR_StartManualRxTrigger	193
17.2.6	函数 IR_SetRxCounterThreshold	193
17.2.7	函数 IR_SendBuf	194
17.2.8	函数 IR_ReceiveBuf	194
17.2.9	函数 IR_INTConfig	195
17.2.10	函数 IR_MaskINTConfig	195
17.2.11	函数 IR_GetINTStatus	196
17.2.12	函数 IR_ClearINTPendingBit	196
17.2.13	函数 IR_GetTxFIFOFreeLen	197
17.2.14	函数 IR_GetRxDataLen	197
17.2.15	函数 IR_SendData	198
17.2.16	函数 IR_ReceiveData	198
17.2.17	函数 IR_SetTxThreshold	199
17.2.18	函数 IR_SetRxThreshold	199
17.2.19	函数 IR_ClearTxFIFO	199
17.2.20	函数 IR_ClearRxFIFO	200
17.2.21	函数 IR_GetFlagStatus	200
17.2.22	函数 IR_SetTxInverse	201

17.2.23	函数 IR_TxOutputInverse	201
17.2.24	函数 IR_SendCompenBuf	202
18	集成电路内置音频总线(I2S)	203
18.1	I2S 寄存器结构	203
18.2	I2S 库函数	203
18.2.1	函数 I2S_DeInit	204
18.2.2	函数 I2S_Init	204
18.2.3	函数 I2S_StructInit	208
18.2.4	函数 I2S_Cmd	208
18.2.5	函数 I2S_INTConfig	209
18.2.6	函数 I2S_GetINTStatus	209
18.2.7	函数 I2S_SendData	210
18.2.8	函数 I2S_ReceiveData	210
18.2.9	函数 I2S_GetTxFIFOFreeLen	211
18.2.10	函数 I2S_GetRxFIFOLen	211
18.2.11	函数 I2S_GetTxErrCnt	211
18.2.12	函数 I2S_GetRxErcnt	212
18.2.13	函数 I2S_SwapBytesForSend	212
18.2.14	函数 I2S_SwapBytesForRead	213
18.2.15	函数 I2S_SwapLRChDataForSend	213
18.2.16	函数 I2S_SwapLRChDataForRead	214
19	8080 并行接口控制器(IF8080)	215
19.1	IF8080 寄存器结构	215
19.2	IF8080 库函数	216
19.2.1	函数 IF8080_DeInit	217
19.2.2	函数 IF8080_PinGroupConfig	217
19.2.3	函数 IF8080_Init	217
19.2.4	函数 IF8080_StructInit	221
19.2.5	函数 IF8080_AutoModeCmd	221
19.2.6	函数 IF8080_SendCommand	222

19.2.7	函数 IF8080_SendData	222
19.2.8	函数 IF8080_ReceiveData.....	222
19.2.9	函数 IF8080_Write.....	223
19.2.10	函数 IF8080_Read.....	223
19.2.11	函数 IF8080_SetCmdSequence	224
19.2.12	函数 IF8080_MaskINTConfig	224
19.2.13	函数 IF8080_GetINTStatus	225
19.2.14	函数 IF8080_GetFlagStatus	226
19.2.15	函数 IF8080_SwitchMode	226
19.2.16	函数 IF8080_GDMALLIConfig	227
19.2.17	函数 IF8080_GDMACmd.....	227
19.2.18	函数 IF8080_SetCS.....	228
19.2.19	函数 IF8080_ResetCS.....	228
19.2.20	函数 IF8080_ClearINTPendingBit	228
19.2.21	函数 IF8080_SetTxDataLen.....	229
19.2.22	函数 IF8080_GetTxDataLen.....	229
19.2.23	函数 IF8080_GetTxCounter	230
19.2.24	函数 IF8080_SetRxDataLen	230
19.2.25	函数 IF8080_GetRxDataLen.....	231
19.2.26	函数 IF8080_GetRxCounter	231
19.2.27	函数 IF8080_ClearTxCounter.....	231
19.2.28	函数 IF8080_WriteFIFO	232
19.2.29	函数 IF8080_ReadFIFO	232
19.2.30	函数 IF8080_ClearFIFO	233
19.2.31	函数 IF8080_VsyncCmd.....	233
20	外设使用流程指南	233
20.1	外设初始化流程	233
20.1.1	PAD 配置	234
20.1.2	PinMux 配置	234
20.1.3	时钟配置.....	234

20.1.4	中断配置.....	235
20.1.5	通用输入/输出(GPIO)初始化.....	235
20.2	管脚分配定义(PAD) 与引脚复用(PINMUX)使用流程	236
20.2.1	PAD 和 PINMUX 简介	236
20.2.2	PAD 唤醒设定	237
20.2.3	进出 DLPS 时 PAD 切换流程	238
20.2.4	DLPS 时管脚输出电平设定流程	239
20.2.5	PAD 防漏电设定	239

图示列表

图 20-1 初始化流程图	234
图 20-2 GPIO 初始化流程图	235
图 20-3 外设和 PAD 电路示意图	237

表格列表

Realtek Confidential

1 外设固件概述

表 1-1 外设缩写定义

缩写	外设/单元
ADC	模数转换器
CODEC	编解码器
GDMA	直接内存存取控制器
GPIO	通用输入输出
I2C	内部集成电路
QDEC	正交解调
PAD	管脚分配定义
PINMUX	引脚复用
LPC	低功耗比较器
RTC	实时时钟
SPI	串型外设接口
SPI3WIRE	三线 SPI
TIM(PWM)	通用定时器(脉宽调试)
UART	通用异步接收发送端
KEYSCAN	按键扫描
IR	红外
I2S	集成电路内置音频总线
IF8080	8080 并行接口控制器

函数的描述按如下格式：

表 1-2 函数描述格式

函数名	固件函数的名称
函数原型	函数原型声明
功能描述	简要描述函数的执行功能
输入参数	输入参数描述
输出参数	输出参数描述
返回值	函数的返回值
先决条件	调用该函数的前提条件
被调用函数	其他被该函数调用的固件库函数

2 模数转换器(ADC)

2.1 ADC 寄存器结构

```
typedef struct
{
    __O  uint32_t FIFO;
    __IO uint32_t CR;
    __IO uint32_t SCHCR;
    __IO uint32_t INTCR;
    __IO uint32_t SCHTAB0;
    __IO uint32_t SCHTAB1;
    __IO uint32_t SCHTAB2;
    __IO uint32_t SCHTAB3;
    __IO uint32_t SCHTAB4;
    __IO uint32_t SCHTAB5;
    __IO uint32_t SCHTAB6;
    __IO uint32_t SCHTAB7;
    __IO uint32_t SCHD0;
    __IO uint32_t SCHD1;
    __IO uint32_t SCHD2;
    __IO uint32_t SCHD3;
    __IO uint32_t SCHD4;
    __IO uint32_t SCHD5;
    __IO uint32_t SCHD6;
    __IO uint32_t SCHD7;
    __IO uint32_t PWRDLY;
    __IO uint32_t DATCLK;
    __IO uint32_t ANACTL;
} ADC_TypeDef;
```

表 2-1 例举 ADC 所有寄存器

表 2-1 ADC 寄存器

寄存器	描述
FIFO	ADC 数据 FIFO
CR	ADC 控制寄存器
SCHCR	ADC Schedule Table Mapping 控制寄存器
INTCR	ADC 中断控制/状态寄存器
SCHTAB0	ADC Schedule Table 工作方式 0 & 1 寄存器
SCHTAB1	ADC Schedule Table 工作方式 2 & 3 寄存器
SCHTAB2	ADC Schedule Table 工作方式 4 & 5 寄存器
SCHTAB3	ADC Schedule Table 工作方式 6 & 7 寄存器

SCHTAB4	ADC Schedule Table 工作方式 8 & 9 寄存器
SCHTAB5	ADC Schedule Table 工作方式 10 & 11 寄存器
SCHTAB6	ADC Schedule Table 工作方式 12 & 13 寄存器
SCHTAB7	ADC Schedule Table 工作方式 14 & 15 寄存器
SCHD0	ADC Schedule Table 转换数据 0 & 1 寄存器
SCHD1	ADC Schedule Table 转换数据 2 & 3 寄存器
SCHD2	ADC Schedule Table 转换数据 4 & 5 寄存器
SCHD3	ADC Schedule Table 转换数据 6 & 7 寄存器
SCHD4	ADC Schedule Table 转换数据 8 & 9 寄存器
SCHD5	ADC Schedule Table 转换数据 10 & 11 寄存器
SCHD6	ADC Schedule Table 转换数据 12 & 13 寄存器
SCHD7	ADC Schedule Table 转换数据 14 & 15 寄存器
DATCLK	ADC 数据&时钟配置寄存器
ANACTL	ADC 模拟控制寄存器

2.2 ADC 库函数

表 2-2 例举 ADC 的所有库函数

表 2-2 ADC 库函数

函数名	描述
ADC_DeInit	关闭 ADC 时钟源&重置 ADC 寄存器
ADC_Init	初始化 ADC 模块
ADC_StructInit	ADC_InitStruct 中的每一个参数写入缺省值
ADC_INTConfig	使能或失能 ADC 外部中断
ADC_Cmd	使能或失能 ADC 外设模块
ADC_BypassCmd	使能或失能 ADC 通道旁路模式
ADC_ReadByScheduleIndex	按照表索引读取相应通道的转换结果
ADC_GetFIFOData	连续读取 ADC FIFO 数据
ADC_GetAvgData	获取 ADC 数据平均值
ADC_SchTableConfig	设置 ADC 特定通道到 ADC Schedule table
ADC_PowerSupplyConfig	ADC 供电配置
ADC_FastPowerSupplyConfig	ADC 快速上电配置
ADC_ClearINTPendingBit	清除 ADC 挂起中断
ADC_GetINTStatus	获取 ADC 指定中断状态
ADC_GetAllINTStatus	获取 ADC 所有中断状态
ADC_SchTableSet	设置 ADC Schedule Table Map
ADC_ReadFIFOData	读取 ADC 一个 FIFO 数据
ADC_GetFIFODataLen	获取 ADC FIFO 数据长度
ADC_ClearFifo	清空 ADC FIFO
ADC_ManualPowerOnCmd	使能或失能 ADC 供电常开并上电
ADC_WriteFIFO Cmd	使能或失能 ADC FIFO 数据写入

ADC_DelayUs	延时 us 函数（内部函数）
ADC_CalibrationInit	ADC 电压换算系数初始化函数（Lib 函数）
ADC_GetVoltage	获取相应模式的 ADC 电压值（Lib 函数）
ADC_GetResistance	获取 ADC 内部电阻值（Lib 函数）

2.2.1 函数 ADC_DeInit

表 2-3 函数 ADC_DeInit

函数名	ADC_DeInit
函数原型	void ADC_DeInit(ADC_TypeDef *ADCx)
功能描述	关闭 ADC 时钟源
输入参数	ADCx: 指向选择的 ADC 模块，ADCx 的 Base Address
输出参数	无
返回值	无
先决条件	无
被调用函数	RCC_PeriphClockCmd 函数

例：

```
/* Reset ADC */
ADC_DeInit(ADC);
```

2.2.2 函数 ADC_Init

表 2-4 函数 ADC_Init

函数名	ADC_Init
函数原型	void ADC_Init(ADC_TypeDef* ADCx, ADC_InitTypeDef* ADC_InitStruct)
功能描述	根据 ADC_InitStruct 中指定参数初始化 ADC 寄存器
输入参数 1	ADCx: 指向选择的 ADC 模块，ADCx 的 Base Address
输入参数 2	ADC_InitStruct: 指向 ADC_InitTypeDef 的指针，ADC_InitStruct 中包含相关配置信息
输出参数	无
返回值	无
先决条件	无
被调用函数	无

```
typedef struct
{
    uint32_t ADC_SampleTime;
    uint32_t ADC_ConvertTime;
    uint32_t ADC_DataWriteToFifo;
    uint32_t ADC_FifoThdLevel;
    uint32_t ADC_WaterLevel;
    uint32_t ADC_FifoOverWriteEn;
    uint32_t ADC_DataLatchEdge;
```

```
uint16_t ADC_SchIndex[16];
uint16_t ADC_Bitmap;
uint32_t ADC_TimerTriggerEn;
uint32_t ADC_DataAlign;
uint32_t ADC_DataMinusEn;
uint32_t ADC_DataMinusOffset;
uint32_t ADC_DataAvgSel;
uint32_t ADC_DataAvgEn;
uint32_t ADC_FifoWriteStop;
uint32_t ADC_PowerOnMode;
uint32_t ADC_PowerAlwaysOnEn;
uint32_t ADC_DataLatchDly;
uint32_t ADC_RG2X0Dly;
uint32_t ADC_RG0X1Dly;
uint32_t ADC_RG0X0Dly;
}ADC_InitTypeDef;
```

ADC_SampleTime: 该参数选择需要设置的 ADC 连续采样的采样周期参数，实际 ADC 单通道的电压采集时间为(ADC_SampleTime +1)/10M + ADC_ConvertTime(0.5us)。该参数可取的值为 19~255 或 2048~14591。例如，当 ADC_SampleTime 设置为 19，ADC 电压采集时间 $T = 20/10M + 0.5\mu s = 2.5\mu s$ ，即 ADC 的采样率为 400K。

ADC_DataWriteToFifo: ADC 单次模式转换值是否同步写入 fifo。错误!未找到引用源。给出了该参数可取的值。

表 2-5 DataWriteToFifo 值

DataWriteToFifo	描述
ADC_DATA_WRITE_TO_FIFO_ENABLE	ADC 转换值同步写入 FIFO
ADC_DATA_WRITE_TO_FIFO_DISABLE	ADC 转换值不写入 FIFO

ADC_FifoThdLevel: 设置 FIFO 阈值，触发中断操作。当 FIFO 中的数据量达到该值，触发中断。需要预先设置 FIFO 中断使能。

ADC_WaterLevel: 触发 GDMA 操作的 ADC 的 FIFO 阈值。当 FIFO 中的数据量达到该值时，触发 GDMA 传输数据。该参数只在 ADC 采用 GDMA 方式传输时有效。

ADC_FifoOverWriteEn: 控制 FIFO Overflow 时是否继续写 ADC FIFO。

ADC_DataLatchEdge: 配置 ADC 数字电路获取数据方式。

ADC_SchIndex: ADC 单次转换表，用于选择 ADC 转换的通道值。表 2-6 给出了该参数可取的值

表 2-6 schedule index 值

schedule index	描述
EXT_SINGLE_ENDED(0)	ADC 单端模式(ADC 外部通道 0)
EXT_SINGLE_ENDED(1)	ADC 单端模式(ADC 外部通道 1)

EXT_SINGLE_ENDED(2)	ADC 单端模式(ADC 外部通道 2)
EXT_SINGLE_ENDED(3)	ADC 单端模式(ADC 外部通道 3)
EXT_SINGLE_ENDED(4)	ADC 单端模式(ADC 外部通道 4)
EXT_SINGLE_ENDED(5)	ADC 单端模式(ADC 外部通道 5)
EXT_SINGLE_ENDED(6)	ADC 单端模式(ADC 外部通道 6)
EXT_SINGLE_ENDED(7)	ADC 单端模式(ADC 外部通道 7)
EXT_DIFFERENTIAL(0)	ADC 差分模式(通道 P: 0 通道 N: 1)
EXT_DIFFERENTIAL(1)	ADC 差分模式(通道 P: 1 通道 N: 0)
EXT_DIFFERENTIAL(2)	ADC 差分模式(通道 P: 2 通道 N: 3)
EXT_DIFFERENTIAL(3)	ADC 差分模式(通道 P: 3 通道 N: 2)
EXT_DIFFERENTIAL(4)	ADC 差分模式(通道 P: 4 通道 N: 5)
EXT_DIFFERENTIAL(5)	ADC 差分模式(通道 P: 5 通道 N: 4)
EXT_DIFFERENTIAL(6)	ADC 差分模式(通道 P: 6 通道 N: 7)
EXT_DIFFERENTIAL(7)	ADC 差分模式(通道 P: 7 通道 N: 6)
INTERNAL_VBAT_MODE	ADC 电池电压检测通道

ADC_Bitmap: 该参数控制 ADC 转换表实际转换内容,

例: 0x7 代表转换表 0, 1, 2 将会在转换中执行。

ADC_TimerTriggerEn: 是否使用硬件 TIM7 控制 ADC 转换。错误!未找到引用源。给出了该参数可取的值。

表 2-7 TimerTriggerEn 值

TimerTriggerEn	描述
ADC_TIMER_TRIGGER_ENABLE	ADC 在每次 TIM7 到期时转换一次
ADC_TIMER_TRIGGER_DISABLE	ADC 不会被 TIM7 控制转换

ADC_DataAlign: ADC 数据对齐方式。错误!未找到引用源。给出了该参数可取的值。

表 2-8 DataAlign 值

DataAlign	描述
ADC_DATA_ALIGN_LSB	ADC 转换值低位在前
ADC_DATA_ALIGN_MSB	ADC 转换值高位在前

ADC_DataMinusEn: ADC 转换值自动减去一个特定值。错误!未找到引用源。给出了该参数可取的值。

表 2-9 DataMinusEn 值

DataMinusEn	描述
ADC_DATA_MINUS_DISABLE	ADC 转换值自动减去一个特定值
ADC_DATA_MINUS_ENABLE	ADC 转换值不自动减去一个特定值

ADC_DataMinusOffset: ADC 转换值自动消除值

ADC_PowerOnMode: ADC 上电模式。错误!未找到引用源。给出了该参数可取的值。

表 2-10 PowerOnMode 值

PowerOnMode	描述
ADC_POWER_ON_MANUAL	ADC 转换手动上电
ADC_POWER_ON_AUTO	ADC 转换自动上电

ADC_PowerAlwaysOnEn: 设置 ADC 模拟 power 是否常开。错误!未找到引用源。给出了该参数可取的值。

表 2-11 PowerAlwaysOnEn 值

PowerAlwaysOnEn	描述
ADC_POWER_ALWAYS_ON_ENABLE	模拟 power 常开
ADC_POWER_ALWAYS_ON_DISABLE	One shot 模式下,当 ADC 转换结束后关闭模拟 power

ADC_DataLatchDly: 数据锁存延时时间, 时间单位为 ADC 采样时钟。该参数可取范围为 1~7, 默认设置为 1, 一般不需要设置。

ADC_RG2X0Dly / ADC_RG0X1Dly / ADC_RG0X0Dly: 控制 ADC 模拟电路上电时间。采用默认值, 一般不需要设置。

例:

```
ADC_InitTypeDef adcInitStruct;
ADC_StructInit(&adcInitStruct);
ADC_InitStruct.ADC_DataWriteToFifo = ADC_DATA_WRITE_TO_FIFO_ENABLE;
ADC_InitStruct.ADC_WaterLevel = 0x4;
ADC_InitStruct.ADC_SchIndex[0] = EXT_SINGLE_ENDED(0);
ADC_InitStruct.ADC_SchIndex[1] = EXT_SINGLE_ENDED(1);
ADC_InitStruct.ADC_Bitmap = 0x03;
ADC_InitStruct.ADC_TimerTriggerEn = ADC_TIMER_TRIGGER_ENABLE;
ADC_InitStruct.ADC_PowerAlwaysOnEn = ADC_POWER_ALWAYS_ON_ENABLE;
ADC_Init(ADC, &adcInitStruct);
```

2.2.3 函数 ADC_StructInit

表 2-12 函数 ADC_StructInit

函数名	ADC_StructInit
函数原型	void ADC_StructInit(ADC_InitTypeDef* ADC_InitStruct)
功能描述	把 ADC_InitStruct 中的每一个参数按缺省值填入
输入参数	ADC_InitStruct: 指向结构 ADC_InitTypeDef 的指针, 包含了外设 ADC 的配置信息
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Configure the ADC init structure parameter */
ADC_InitTypeDef ADC_InitStruct;
```

```
ADC_StructInit (&ADC_InitStruct);
```

2.2.4 函数 ADC_INTConfig

表 2.1 函数 ADC_INTConfig

函数名	ADC_INTConfig
函数原型	ADC_INTConfig(ADC_TypeDef* ADCx, uint32_t ADC_IT, FunctionalState newState)
功能描述	使能或失能 ADC 外部中断
输入参数 1	ADCx: 指向选择的 ADC 模块, ADCx 的 Base Address
输入参数 2	ADC_IT: 指定使能中断类型
输入参数 3	NewState: 使能或者关闭参数 2 指定的中断 可选参数: ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

ADC_IT: 允许的 ADC 中断类型如错误!未找到引用源。所示, 使用操作符“|”可以一次选中多个中断作为该参数的值。

表 2-13 ADC_IT 值

ADC_IT	描述
ADC_INT_FIFO_RD_REQ	ADC 读请求中断
ADC_INT_FIFO_RD_ERR	ADC 误读中断
ADC_INT_FIFO_TH	ADC FIFO 数目超过给定值中断
ADC_INT_FIFO_FULL	ADC FIFO 满中断
ADC_INT_ONE_SHOT_DONE	ADC 单次转换完成中断

例:

```
/* Enable ADC one shot done interrupt */
ADC_INTConfig(ADC, ADC_INT_ONE_SHOT_DONE, ENABLE);
```

2.2.5 函数 ADC_Cmd

表 2-14 函数 ADC_Cmd

函数名	ADC_Cmd
函数原型	void ADC_Cmd(ADC_TypeDef* ADCx, uint8_t adcMode, FunctionalState NewState)
功能描述	使能或失能 ADC 外设模块
输入参数 1	ADCx: 指向选择的 ADC 模块, ADCx 的 Base Address
输入参数 2	adcMode: 选择 ADC 模式 ADC_One_Shot_Mode: one shot mode. ADC_Continuous_Mode: Continuous mode
输入参数 3	NewState: 外设 ADC 的新状态 这个参数可以取: ENABLE 或者 DISABLE

输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable ADC */
ADC_Cmd(ADC, ADC_One_Shot_Mode, ENABLE);
```

2.2.6 函数 ADC_BypassCmd

表 2-15 函数 ADC_HighBypassCmd

函数名	ADC_BypassCmd
函数原型	void ADC_BypassCmd(uint8_t channelNum, FunctionalState NewState)
功能描述	配置 ADC 特定外部通道高阻态模式
输入参数 1	channelNum: 外部通道值
输入参数 2	NewState: ENABLE: 配置为高阻态模式, 电压转换范围为 0~0.9V DISABLE: 配置为普通模式, 电压转换范围为 0~3.3V
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Get ADC external channel 3 bypass mode*/
ADC_BypassCmd(3,ENABLE);
```

2.2.7 函数 ADC_ReadRawData

表 2-16 函数 ADC_ReadRawData

函数名	ADC_ReadRawData
函数原型	uint16_t ADC_ReadRawData (ADC_TypeDef *ADCx, uint8_t ScheduleIndex)
功能描述	按照表索引读取相应通道的转换结果
输入参数 1	ADCx: 指向选择的 ADC 模块, ADCx 的 Base Address
输入参数 2	ScheduleIndex: 被设置的转换表值, 取值范围为 0~15
输出参数	无
返回值	The 12-bit converted ADC data
先决条件	无
被调用函数	无

例：

```
/* Read data by schedule index */
```

```
uint16_t value = 0;
value = ADC_ ReadRawData (ADC, 0);
```

2.2.8 函数 ADC_GetFIFOData

表 2-17 函数 ADC_GetFIFOData

函数名	ADC_GetFIFOData
函数原型	void ADC_GetFIFOData(ADC_TypeDef *ADCx, uint16_t *outBuf, uint16_t Num)
功能描述	从 ADC FIFO 获取特定数量的数据
输入参数 1	ADCx: 指向选择的 ADC 模块, ADCx 的 Base Address
输入参数 2	Num: 获取的 ADC 转换数据长度
输出参数	outBuf: 存入 ADC 数据的地址
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Get ADC data */
ADC_GetFIFOData (ADC,data,5);
```

2.2.9 函数 ADC_GetAvgData

表 2-18 函数 ADC_GetAvgData

函数名	ADC_GetFIFOData
函数原型	uint16_t ADC_GetAvgData(ADC_TypeDef *ADCx)
功能描述	从 ADC FIFO 获取一个数据
输入参数 1	ADCx: 指向选择的 ADC 模块, ADCx 的 Base Address
返回值	ADC RawData
先决条件	无
被调用函数	无

例:

```
/* Get ADC data */
ADC_GetAvgData (ADC);
```

2.2.10 函数 ADC_SchTableConfig

表 2-19 函数 ADC_SchTableConfig

函数名	ADC_SchTableConfig
函数原型	void ADC_SchTableConfig(ADC_TypeDef *ADCx, uint16_t Index, uint8_t adcMode)
功能描述	配置特定的 ADC 通道到 ADC 转换表

输入参数 1	ADCx: 指向选择的 ADC 模块, ADCx 的 Base Address
输入参数 2	Index: ADC 转换表索引 可选参数: 0 - 15
输入参数 3	adcMode: 指定的 ADC 通道, 具体参阅 2.2.1 节 channelMap 介绍部分
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Enable ADC specific channel */
ADC_SchTableConfig (ADC,0, EXT_SINGLE_ENDED(0));
```

2.2.11 函数 ADC_PowerSupplyConfig

表 2-20 函数 ADC_PowerSupplyConfig

函数名	ADC_SchTableConfig
函数原型	Void ADC_PowerSupplyConfig(FunctionalState NewState)
功能描述	设置 ADC power supply
输入参数 1	NewState: ENABLE or DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Enable ADC 1.8V */
ADC_PowerSupplyConfig (ENABLE);
```

2.2.12 函数 ADC_FastPowerSupplyConfig

表 2-21 函数 ADC_FastPowerSupplyConfig

函数名	ADC_FastPowerSupplyConfig
函数原型	void ADC_FastPowerSupplyConfig(FunctionalState NewState)
功能描述	设置 ADC fast power supply
输入参数 1	NewState: ENABLE or DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Enable ADC fast power supply */
ADC_FastPowerSupplyConfig (ENABLE);
```

2.2.13 函数 ADC_ClearINTPendingBit

表 2-22 函数 ADC_ClearINTPendingBit

函数名	ADC_ClearINTPendingBit
函数原型	void ADC_ClearINTPendingBit(ADC_TypeDef* ADCx, uint32_t ADC_IT)
功能描述	清除挂起的 ADC 中断
输入参数 1	ADCx: 指向选择的 ADC 模块, ADCx 的 Base Address
输入参数 2	ADC_IT: 指定的 ADC 中断, 具体参阅 2.2.4 节 ADC_IT 介绍部分
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Clear ADC interrupt */
ADC_ClearINTPendingBit(ADC, ADC_INT_ONE_SHOT_DONE);
```

2.2.14 函数 ADC_GetINTStatus

表 2-23 函数 ADC_GetINTStatus

函数名	ADC_GetINTStatus
函数原型	FlagStatus ADC_GetINTStatus (ADC_TypeDef* ADCx, uint32_t ADC_INT_FLAG)
功能描述	检查指定的 ADC 中断标志是否被置位
输入参数 1	ADCx: 指向选择的 ADC 模块, ADCx 的 Base Address
输入参数 2	ADC_INT_FLAG: 指定的 ADC 中断标志位, 具体参阅 2.2.4 节 ADC_IT 介绍部分
输出参数	无
返回值	状态标志
先决条件	无
被调用函数	无

例:

```
/*wait for adc sample ready*/
while (ADC_GetINTStatus (ADC, ADC_INT_ONE_SHOT_DONE) != SET);
```

2.2.15 函数 ADC_GetAllINTStatus

表 2-24 函数 ADC_GetAllINTStatus

函数名	ADC_GetAllINTStatus
函数原型	uint8_t ADC_GetAllINTStatus (ADC_TypeDef *ADCx)
功能描述	获取 ADC 中断状态
输入参数 1	ADCx: 指向选择的 ADC 模块, ADCx 的 Base Address
输出参数	无

返回值	ADC 中断状态
先决条件	无
被调用函数	无

例：

```
/* Get ADC all interrupt status */
uint8_t int_status = ADC_GetAllINTStatus (ADC);
```

2.2.16 函数 ADC_BitMapConfig

表 2-25 函数 ADC_BitMapConfig

函数名	ADC_BitMapConfig
函数原型	ADC_BitMapConfig (ADC_TypeDef *ADCx, uint16_t channelMap, FunctionalState NewState)
功能描述	设置 ADC 的 Schedule Table Map
输入参数 1	ADCx: 指向选择的 ADC 模块, ADCx 的 Base Address
输入参数 2	channelMap: Schedule Table Map 的设定值
输入参数 3	NewState: ENABLE or DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Set adc schedule table */
ADC_BitMapConfig (ADC, 0x01,ENABLE);
```

2.2.17 函数 ADC_ReadFIFOData

表 2-26 函数 ADC_ReadFIFOData

函数名	ADC_ReadFIFOData
函数原型	uint16_t ADC_ReadFIFOData (ADC_TypeDef *ADCx)
功能描述	通过外设 ADC 读取一个数据
输入参数 1	ADCx: 指向选择的 ADC 模块, ADCx 的 Base Address
输出参数	无
返回值	通过 ADC 转换的值
先决条件	无
被调用函数	无

例：

```
/* Receive one half word*/
Uint16_t data = ADC_ReadFIFOData (ADC);
```

2.2.18 函数 ADC_GetFIFODataLen

表 2-27 函数 ADC_GetFIFODataLen

函数名	ADC_GetFIFODataLen
函数原型	uint8_t ADC_GetFIFODataLen (ADC_TypeDef *ADCx)
功能描述	获取 ADC FIFO 中数目长度
输入参数 1	ADCx: 指向选择的 ADC 模块, ADCx 的 Base Address
输出参数	无
返回值	ADC FIFO 中数目长度
先决条件	无
被调用函数	无

例:

```
/* Get ADC fifo number */
Uint8_t length = ADC_GetFIFODataLen (ADC);
```

2.2.19 函数 ADC_ClearFIFO

表 2-28 函数 ADC_ClearFifo

函数名	ADC_ClearFifo
函数原型	uint8_t ADC_ClearFifo (ADC_TypeDef *ADCx)
功能描述	清空 ADC FIFO
输入参数 1	ADCx: 指向选择的 ADC 模块, ADCx 的 Base Address
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Clear ADC fifo */
ADC_ClearFifo (ADC);
```

2.2.20 函数 ADC_ManualPowerOnCmd

表 2-29 函数 ADC_ManualPowerOnCmd

函数名	ADC_ManualPowerOnCmd
函数原型	void ADC_ManualPowerOnCmd (ADC_TypeDef *ADCx, FunctionalState NewState)
功能描述	使能或失能 ADC 供电常开功能
输入参数 1	ADCx: 指向选择的 ADC 模块, ADCx 的 Base Address
输入参数 2	NewState: ENBALE or DISABLE
输出参数	无
返回值	无

先决条件	无
被调用函数	无

例：

```
/* Enable or disable ADC Power always on */
ADC_ManualPowerOnCmd (ADC,ENBALE);
```

2.2.21 函数 ADC_WriteFIFOcmd

表 2-30 函数 ADC_WriteFIFOcmd

函数名	ADC_WriteFIFOcmd
函数原型	void ADC_WriteFIFOcmd (ADC_TypeDef *ADCx, FunctionalState NewState)
功能描述	使能或失能 ADC 供电常开功能
输入参数 1	ADCx: 指向选择的 ADC 模块, ADCx 的 Base Address
输入参数 2	NewState: ENBALE or DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable or disable ADC Power always on */
ADC_WriteFIFOcmd (ADC,ENBALE);
```

2.2.22 函数 ADC_DelayUs

表 2-31 函数 ADC_DelayUs

函数名	ADC_DelayUs
函数原型	uint8_t ADC_DelayUs (uint32_t t)
功能描述	延时 us 函数
输入参数 1	t: 延时时间
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* ADC delay */
ADC_DelayUs (1000);
```

2.2.23 函数 ADC_CalibrationInit

表 2-32 函数 ADC_CalibrationInit

函数名	ADC_CalibrationInit
函数原型	bool ADC_CalibrationInit(void)
功能描述	ADC 电压换算系数初始化函数
输入参数	无
输出参数	无
返回值	初始化是否成功 true or false
先决条件	无
被调用函数	无

例：

```
/*Get ADC efuse k value*/
bool status = ADC_CalibrationInit();
```

2.2.24 函数 ADC_GetVoltage

表 2-33 函数 ADC_GetVoltage

函数名	ADC_GetVoltage
函数原型	float ADC_GetVoltage(const ADC_SampleMode vSampleMode, int32_t vSampleData, ADC_ErrorStatus *pErrorStatus)
功能描述	获取相应模式的 ADC 电压值
输入参数 1	ADC 采样模式，取值见表 2-34
输入参数 2	ADC 采样原始数据
输入参数 3	ADC 电压转换后参数错误状态返回值，定义见表 2-35
输出参数	无
返回值	转换后的电压值
先决条件	无
被调用函数	无

表 2-34 ADC 采样模式

模式	描述
DIVIDE_SINGLE_MODE	ADC single divide 模式
BYPASS_SINGLE_MODE	ADC single bypass 模式
DIVIDE_DIFFERENTIAL_MODE	ADC 差分 divide 模式
BYPASS_DIFFERENTIAL_MODE	ADC 差分 bypass 模式

表 2-35 ADC 电压换算错误状态

参数取值	描述
NO_ERROR	无错误
PARAMETER_ERROR	参数错误
RAM_DATA_ERROR	RAM 数据错误
NO_CALIBRATION	未校准
VERSION_ERROR	版本错误

例：

```
/*Get ADC voltage*/
ADC_ErrorStatus error_status = NO_ERROR;
float voltage = ADC_GetVoltage (DIVIDE_SINGLE_MODE,2048, &error_status);
```

2.2.25 函数 ADC_GetResistance

表 2-36 函数 ADC_GetResistance

函数名	ADC_GetResistance
函数原型	uint16_t ADC_GetResistance(void);
功能描述	获取 ADC 内部电阻值
输入参数	无
输出参数	无
返回值	内部电阻值
先决条件	无
被调用函数	无

例：

```
/*Get ADC resistance value*/
uint16_t value = ADC_GetResistance();
```

3 编解码器(CODEC)

3.1 CODEC 寄存器架构

```
typedef struct
{
    __IO uint32_t ANA_CR0;          /*!< 0x00 */
    __IO uint32_t ANA_CR1;          /*!< 0x04 */
    __IO uint32_t ANA_CR2;          /*!< 0x08 */
} CODEC_AnalogTypeDef;
```

错误!未找到引用源。例举 CODEC Analog 寄存器

表 3-1 CODEC Analog 寄存器

寄存器	描述
ANA_CR0	CODEC 模拟控制寄存器 0
ANA_CR1	CODEC 模拟控制寄存器 1
ANA_CR2	CODEC 模拟控制寄存器 2

```
typedef struct
{
    __IO uint32_t CR0;
    __IO uint32_t ANA_CR1;
    __IO uint32_t CLK_CR1;
    __IO uint32_t CLK_CR2;
    __IO uint32_t CLK_CR3;
    __IO uint32_t ASRC_CR0;
    __IO uint32_t ASRC_CR1;
    __IO uint32_t I2S_CTRL;
    __IO uint32_t ADC0_CTRL0;
    __IO uint32_t ADC0_CTRL1;
    __IO uint32_t ADC1_CTRL0;
    __IO uint32_t ADC1_CTRL1;
    __IO uint32_t DAC_CTRL0;
    __IO uint32_t DAC_CTRL1;
} CODEC_TypeDef;
```

错误!未找到引用源。例举 CODEC 所有寄存器

表 3-2 CODE 寄存器

寄存器	描述
CR0	CODEC 控制寄存器 0
ANA_CR1	CODEC 模拟控制寄存器 1
CLK_CR1	CODEC 时钟控制寄存器 1
CLK_CR2	CODEC 时钟控制寄存器 2

CLK_CR3	CODEC 时钟控制寄存器 3
ASRC_CRO	ASRC 控制寄存器
ASRC_CR1	ASRC 控制寄存器
I2S_CTRL	I2S 控制寄存器
ADCO_CTRL0	ADCO 控制寄存器 0
ADCO_CTRL1	ADCO 控制寄存器 1
ADC1_CTRL0	ADC1 控制寄存器 0
ADC1_CTRL1	ADC1 控制寄存器 1
DAC_CTRL0	DAC 控制寄存器 0
DAC_CTRL1	DAC 控制寄存器 1

```

typedef struct
{
    __IO uint32_t EQ_H0;          /*!< 0x40 */
    __IO uint32_t EQ_B1;          /*!< 0x44 */
    __IO uint32_t EQ_B2;          /*!< 0x48 */
    __IO uint32_t EQ_A1;          /*!< 0x4C */
    __IO uint32_t EQ_A2;          /*!< 0x50 */
}CODEC_EQTypeDef;

```

错误!未找到引用源。例举 EQ 所有寄存器

表 3-3 CODEC EQ 寄存器

寄存器	描述
EQ_H0	EQ 控制寄存器 H0
EQ_B1	EQ 控制寄存器 B1
EQ_B2	EQ 控制寄存器 B2
EQ_A1	EQ 控制寄存器 A1
EQ_A2	EQ 控制寄存器 A2

3.2 CODEC 库函数

表 3-4 例举 CODEC 的所有库函数

函数名	描述
CODEC_DelInit	关闭 CODEC 的时钟
CODEC_Init	根据 CODEC_InitStruct 中指定参数初始化 CODEC 寄存器
CODEC_StructInit	把 CODEC_InitStruct 中的每一个参数按缺省值填入
CODEC_EQInit	根据 CODEC_EQInitStruct 中指定参数初始化 CODEC EQ 寄存器
CODEC_EQStructInit	把 CODEC_EQInitStruct 中的每一个参数按缺省值填入
CODEC_MICBIASCmd	使能或者失能 MICBIAS
CODEC_Reset	复位 CODEC 外设
CODEC_SetMICBIAS	设置 MICBIAS 电压范围

3.2.1 函数 CODEC_DeInit

表 3-5 函数 CODEC_DeInit

函数名	CODEC_DeInit
函数原型	void CODEC_DeInit(CODEC_TypeDef* CODECx)
功能描述	关闭 CODEC 的时钟
输入参数 1	CODECx: 指向选择的 CODEC 模块, CODEC 的 Base Address
输出参数	无
返回值	无
先决条件	无
被调用函数	RCC_PeriphClockCmd 函数

例:

```
/* Close codec */
CODEC_DeInit(CODEC);
```

3.2.2 函数 CODEC_Init

表 3-6 函数 CODEC_Init

函数名	CODEC_Init
函数原型	void CODEC_Init(CODEC_TypeDef* CODECx, CODEC_InitTypeDef* CODEC_InitStruct)
功能描述	根据 CODEC_InitStruct 中指定参数初始化 CODEC 寄存器
输入参数 1	CODECx: 指向选择的 CODEC 模块, CODEC 的 Base Address
输入参数 2	CODEC_InitStruct: 指向 CODEC_InitTypeDef 的指针, CODEC_InitStruct 中包含相关配置信息
输出参数	无
返回值	无
先决条件	无
被调用函数	无

```
typedef struct
{
    uint32_t CODEC_MicType;           /*!< Specifies the mic type, which can be dmic or amic */
    uint32_t CODEC_SampleRate;         /*!< Specifies the sample rate */
    /* I2S parameters section */
    uint32_t CODEC_I2SFormat;          /*!< Specifies the I2S format of codec port */
    uint32_t CODEC_I2SDataWidth;       /*!< Specifies the I2S data width of codec port */
    /* Input: ADC parametes section */
    uint32_t CODEC_MicBIAS;           /*!< Specifies the MICBIAS voltage */
    uint32_t CODEC_AdGain;             /*!< Specifies the ADC digital volume */
    uint32_t CODEC_BoostGain;          /*!< Specifies the boost gain */
    uint32_t CODEC_AdZeroDetTimeout;   /*!< Specifies the mono ADC zero detection timeout control */
    uint32_t CODEC_MICBstGain;         /*!< Specifies the MICBst gain */
```

```

uint32_t CODEC_MICBstMode;           /*!< Specifies the MICBST mode */
/* Input: Dmic parametes section */
uint32_t CODEC_DmicClock;           /*!< Specifies the dmic clock */
uint32_t CODEC_DmicDataLatch;       /*!< Specifies the dmic data latch type */
/* Output: DAC control */
uint32_t CODEC_DaMute;              /*!< Specifies the DAC mute status */
uint32_t CODEC_DaGain;              /*!< Specifies the DAC gain control */
uint32_t CODEC_DacZeroDetTimeout;   /*!< Specifies the mono DAC zero detection timeout control */
}CODEC_InitTypeDef;

```

CODEC_MicType: 麦克风类型定义，表明是模拟麦克风还是数字麦克风，此值可以设置为 CODEC_AMIC 或者 CODEC_DMIC。

CODEC_SampleRate: 设置采样频率。错误!未找到引用源。给出了该参数可取的值

表 3-7 CODEC_SampleRate 值

CODEC_SampleRate	描述
SAMPLE_RATE_8KHz	采样频率为 8KHz
SAMPLE_RATE_16Khz	采样频率为 16KHz

CODEC_I2SFormat: 设置数据格式。错误!未找到引用源。给出了该参数可取的值。

表 3-8 CODEC_I2SFormat 值

CODEC_I2SFormat	描述
CODEC_I2S_DataFormat_I2S	I2S 格式
CODEC_I2S_DataFormat_LeftJustified	左对齐格式
CODEC_I2S_DataFormat_PCM_A	PCM A 格式
CODEC_I2S_DataFormat_PCM_B	PCM B 格式

CODEC_I2SDataWidth: 设置数据宽度。错误!未找到引用源。给出了该参数可取的值。

表 3-9 CODEC_I2SDataWidth 值

CODEC_I2SDataWidth	描述
CODEC_I2S_DataWidth_16Bits	数据宽度为 16bit
CODEC_I2S_DataWidth_24Bits	数据宽度为 24bit
CODEC_I2S_DataWidth_8Bits	数据宽度为 8bit

CODEC_MicBIAS: 设置 MIC BIAS 电压值。错误!未找到引用源。给出了该参数可取的值。

表 3-10 CODEC_MicBIAS 值

CODEC_MicBIAS	描述
MICBIAS_VOLTAGE_1_507	1.507v
MICBIAS_VOLTAGE_1_62	1.62v
MICBIAS_VOLTAGE_1_705	1.705v
MICBIAS_VOLTAGE_1_8	1.8v
MICBIAS_VOLTAGE_1_906	1.906v
MICBIAS_VOLTAGE_2_025	2.025v

MICBIAS_VOLTAGE_2_16	2.16v
MICBIAS_VOLTAGE_2_314	2.314v

CODEC_AdGain: 设置麦克风音量。该参数设置范围为 0x00 到 0x7f。其中，0x00 为 -17.625dB，0x2f 为 0dB，0x7f 为 30dB。

CODEC_BoostGain: 设置 Boost 增益。错误!未找到引用源。给出了该参数可取的值。

表 3-11 CODEC_BoostGain 值

CODEC_BoostGain	描述
Boost_Gain_0dB	0dB
Boost_Gain_12dB	12dB
Boost_Gain_24dB	24dB
Boost_Gain_36dB	36dB

CODEC_AdZeroDetTimeout: 设置 ADC 的零点探测超时时间。错误!未找到引用源。给出了该参数可取的值。

表 3-12 CODEC_AdZeroDetTimeout 值

CODEC_AdZeroDetTimeout	描述
ADC_Zero_DetTimeout_1024_16_Sample	1024*16 采样次数
ADC_Zero_DetTimeout_1024_32_Sample	1024*32 采样次数
ADC_Zero_DetTimeout_1024_64_Sample	1024*64 采样次数
ADC_Zero_DetTimeout_64_Sample	64 采样次数

CODEC_MICBstGain: 设置 MICBST 增益。错误!未找到引用源。给出了该参数可取的值。

表 3-13 CODEC_MICBstGain 值

CODEC_MICBstGain	描述
MICBst_Gain_0dB	0dB
MICBst_Gain_20dB	20dB
MICBst_Gain_30dB	30dB
MICBst_Gain_40dB	40dB

CODEC_DmicClock: 设置 DMIC 的时钟频率。表 3.1 给出了该参数可取的值。

表 3.1 CODEC_DmicClock 值

CODEC_DmicClock	描述
DMIC_Clock_4MHz	设置 DMIC 的时钟频率为 4M
DMIC_Clock_2MHz	设置 DMIC 的时钟频率为 2M
DMIC_Clock_1MHz	设置 DMIC 的时钟频率为 1M
DMIC_Clock_500KHz	设置 DMIC 的时钟频率为 500K

CODEC_DmicDataLatch: 设置数字麦克风的数据锁存方式。错误!未找到引用源。给出了该参数可取的值。

表 3-14 CODEC_DmicDataLatch 值

CODEC_DmicDataLatch	描述
DMIC_Rising_Latch	上升沿锁存

DMIC_Falling_Latch	下降沿锁存
--------------------	-------

CODEC_DaMute: 使能或者失能 DAC 输出。错误!未找到引用源。给出了该参数可取的值。

表 3-15 CODEC_DaMute 值

CODEC_DaMute	描述
DAC_UnMute	使能 DAC 输出
DAC_Mute	失能 DAC 输出

CODEC_DaGain: 设置 DAC 音量。该参数设置范围为 0x00 到 0xff。其中，0x00 为 -65.625dB，0xaf 为 0dB，0.375dB/step。

CODEC_DacZeroDetTimeout: 设置 DAC 的零点探测超时时间。错误!未找到引用源。给出了该参数可取的值。

表 3-16 CODEC_DacZeroDetTimeout 值

CODEC_DacZeroDetTimeout	描述
DAC_Zero_DetTimeout_1024_16_Sample	1024*16 采样次数
DAC_Zero_DetTimeout_1024_32_Sample	1024*32 采样次数
DAC_Zero_DetTimeout_1024_64_Sample	1024*64 采样次数
DAC_Zero_DetTimeout_256_Sample	256 采样次数

例：

```
/* Initialize codec */
CODEC_InitTypeDef CODEC_InitStruct;
CODEC_StructInit(&CODEC_InitStruct);
CODEC_InitStruct.CODEC_MicType      = CODEC_DMIC;
CODEC_InitStruct.CODEC_DmicClock   = DMIC_Clock_2MHz;
CODEC_InitStruct.CODEC_DmicDataLatch = DMIC_Rising_Latch;
CODEC_InitStruct.CODEC_SampleRate    = SAMPLE_RATE_16KHz;
CODEC_InitStruct.CODEC_I2SFormat     = CODEC_I2S_DataFormat_I2S;
CODEC_InitStruct.CODEC_I2SDataWidth   = CODEC_I2S_DataWidth_16Bits;
CODEC_Init(CODEC, &CODEC_InitStruct);
```

3.2.3 函数 CODEC_StructInit

表 3-17 函数 CODEC_StructInit

函数名	CODEC_StructInit
函数原型	void CODEC_StructInit(CODEC_InitTypeDef* CODEC_InitStruct)
功能描述	把 CODEC_InitStruct 中的每一个参数按缺省值填入
输入参数	CODEC_InitStruct: 指向结构 CODEC_InitTypeDef 的指针，包含了外设 CODEC 的配置信息
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Initialize codec */
CODEC_InitTypeDef CODEC_InitStruct;
CODEC_StructInit(&CODEC_InitStruct);
```

3.2.4 函数 CODEC_EQInit

表 3-18 函数 CODEC_EQInit

函数名	CODEC_EQInit
函数原型	void CODEC_EQInit(CODEC_EQTypeDef* CODEC_EQx, CODEC_EQInitTypeDef* CODEC_EQInitStruct)
功能描述	根据 CODEC_EQInitStruct 中指定参数初始化 CODEC EQ 寄存器
输入参数 1	CODEC_EQx：指向选择的 CODEC EQ 模块，CODEC EQ 的 Base Address
输入参数 2	CODEC_EQInitStruct：指向 CODEC_EQInitTypeDef 的指针，CODEC_EQInitStruct 中包含相关配置信息
输出参数	无
返回值	无
先决条件	无
被调用函数	无

```
typedef struct
{
    uint32_t CODEC_EQChCmd;           /*!< Specifies the EQ channel status */
    uint32_t CODEC_EQCoefH0;          /*!< Specifies the EQ coef.h0. This value can be 0 to 0xFFFF,
                                         whose physical meaning represents a range of -8 to 7.99 */
    uint32_t CODEC_EQCoefB1;          /*!< Specifies the EQ coef.b1. This value can be 0 to 0xFFFF,
                                         whose physical meaning represents a range of -8 to 7.99 */
    uint32_t CODEC_EQCoefB2;          /*!< Specifies the EQ coef.b2. This value can be 0 to 0xFFFF,
                                         whose physical meaning represents a range of -8 to 7.99 */
    uint32_t CODEC_EQCoefA1;          /*!< Specifies the EQ coef.a1. This value can be 0 to 0xFFFF,
                                         whose physical meaning represents a range of -8 to 7.99 */
    uint32_t CODEC_EQCoefA2;          /*!< Specifies the EQ coef.a2. This value can be 0 to 0xFFFF,
                                         whose physical meaning represents a range of -8 to 7.99 */
}CODEC_EQInitTypeDef;
```

CODEC_EQChCmd: 使能或者失能 EQ 相应通道功能。错误!未找到引用源。给出了该参数可取的值

表 3-19 CODEC_EQChCmd 值

CODEC_EQChCmd	描述
EQ_CH_Cmd_ENABLE	使能 EQ 功能
EQ_CH_Cmd_DISABLE	失能 EQ 功能

CODEC_EQCoefH0: 设置 EQ H0。该参数设置范围为 0x00 到 0x7FFF。其中，0x00 为-8，0x2f 为 0dB，0x7FFF 为 7.99。

CODEC_EQCoefB1: 设置 EQ B1。该参数设置范围为 0x00 到 0x7FFF。其中，0x00 为-8，0x2f 为 0dB，0x7FFF 为 7.99。

CODEC_EQCoefB2: 设置 EQ B2。该参数设置范围为 0x00 到 0x7FFF。其中，0x00 为-8，0x2f 为 0dB，0x7FFF 为 7.99。

CODEC_EQCoefA1: 设置 EQ A1。该参数设置范围为 0x00 到 0x7FFF。其中，0x00 为-8，0x2f 为 0dB，0x7FFF 为 7.99。

CODEC_EQCoefA2: 设置 EQ A2。该参数设置范围为 0x00 到 0x7FFF。其中，0x00 为-8，0x2f 为 0dB，0x7FFF 为 7.99。

例：

```
/* Initialize codec EQ1 */
CODEC_EQInitTypeDef CODEC_EQInitStruct;
CODEC_StructInit(&CODEC_EQInitStruct);
CODEC_EQInitStruct.CODEC_EQChCmd = EQ_CH_Cmd_ENABLE;
CODEC_EQInitStruct.CODEC_EQCoefH0 = 0xFF;
CODEC_EQInitStruct.CODEC_EQCoefB1 = 0xFF;
CODEC_EQInitStruct.CODEC_EQCoefB2 = 0xFF;
CODEC_EQInitStruct.CODEC_EQCoefA1 = 0xFF;
CODEC_EQInitStruct.CODEC_EQCoefA2 = 0xFF;
CODEC_EQInit(CODEC_EQ1, CODEC_EQInitStruct);
```

3.2.5 函数 CODEC_EQStructInit

表 3-20 函数 CODEC_EQStructInit

函数名	CODEC_EQStructInit
函数原型	void CODEC_EQStructInit(CODEC_EQInitTypeDef* CODEC_EQInitStruct)
功能描述	把 CODEC_EQInitStruct 中的每一个参数按缺省值填入
输入参数	CODEC_EQInitStruct：指向结构 CODEC_EQInitTypeDef 的指针，包含了外设 CODEC EQ 的配置信息
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Initialize EQ */
CODEC_EQInitTypeDef CODEC_EQInitStruct;
CODEC_EQStructInit(&CODEC_EQInitStruct);
```

3.2.6 函数 CODEC_MICBIASCmd

表 3-21 函数 CODEC_MICBIASCmd

函数名	CODEC_MICBIASCmd
函数原型	void CODEC_MICBIASCmd(CODEC_TypeDef* CODECx, FunctionalState NewState)
功能描述	使能或者失能 MICBIAS
输入参数 1	CODECx: 指向选择的 CODEC 模块, CODEC 的 Base Address
输入参数 2	NewState: MICBIAS 的新状态 可选参数: ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Enable MICBIAS */  
CODEC_MICBIASCmd (CODEC, ENABLE);
```

3.2.7 函数 CODEC_Reset

表 3-22 函数 CODEC_Reset

函数名	CODEC_Reset
函数原型	void CODEC_Reset(CODEC_TypeDef* CODECx)
功能描述	复位 CODEC 外设
输入参数	CODECx: 指向选择的 CODEC 模块, CODEC 的 Base Address
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Reset codec */
CODEC_Reset(CODEC);
```

3.2.8 函数 CODEC_SetMICBIAS

表 3-23 函数 CODEC_SetMICBIAS

函数名	CODEC_SetMICBIAS
函数原型	void CODEC_SetMICBIAS(CODEC_TypeDef* CODECx, uint16_t data)
功能描述	设置 MICBIAS 电压大小
输入参数 1	CODECx: 指向选择的 CODEC 模块, CODEC 的 Base Address
输入参数 2	<p>data: MICBIAS 电压大小, 可选参数如下:</p> <p>MICBIAS_VOLTAGE_1_507: Vref voltage is 1.507V.</p> <p>MICBIAS_VOLTAGE_1_62: Vref voltage is 1.62V.</p> <p>MICBIAS_VOLTAGE_1_705: Vref voltage is 1.705V.</p> <p>MICBIAS_VOLTAGE_1_8: Vref voltage is 1.8V.</p> <p>MICBIAS_VOLTAGE_1_906: Vref voltage is 1.906V.</p> <p>MICBIAS_VOLTAGE_2_025: Vref voltage is 2.025V.</p> <p>MICBIAS_VOLTAGE_2_16: Vref voltage is 2.16V.</p> <p>MICBIAS_VOLTAGE_2_314: Vref voltage is 2.314V.</p>
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Set MICBIAS */
CODEC_SetMICBIAS (CODEC, MICBIAS_VOLTAGE_1_8);
```

4 直接内存存取控制器(GDMA)

4.1 GDMA 寄存器架构

```
typedef struct
{
    __I uint32_t  RAW_TFR;
    uint32_t  RSVDO;
    __I uint32_t  RAW_BLOCK;
    uint32_t  RSVD1;
    __I uint32_t  RAW_SRC_TRAN;
    uint32_t  RSVD2;
    __I uint32_t  RAW_DST_TRAN;
    uint32_t  RSVD3;
    __I uint32_t  RAW_ERR;
    uint32_t  RSVD4;

    __I uint32_t  STATUS_TFR;
    uint32_t  RSVD5;
    __I uint32_t  STATUS_BLOCK;
    uint32_t  RSVD6;
    __I uint32_t  STATUS_SRC_TRAN;
    uint32_t  RSVD7;
    __I uint32_t  STATUS_DST_TRAN;
    uint32_t  RSVD8;
    __I uint32_t  STATUS_ERR;
    uint32_t  RSVD9;

    __IO uint32_t  MASK_TFR;
    uint32_t  RSVD10;
    __IO uint32_t  MASK_BLOCK;
    uint32_t  RSVD11;
    __IO uint32_t  MASK_SRC_TRAN;
    uint32_t  RSVD12;
    __IO uint32_t  MASK_DST_TRAN;
    uint32_t  RSVD13;
    __IO uint32_t  MASK_ERR;
    uint32_t  RSVD14;

    __O uint32_t  CLEAR_TFR;
    uint32_t  RSVD15;
```

```
__O uint32_t  CLEAR_BLOCK;
uint32_t  RSVD16;
__O uint32_t  CLEAR_SRC_TRAN;
uint32_t  RSVD17;
__O uint32_t  CLEAR_DST_TRAN;
uint32_t  RSVD18;
__O uint32_t  CLEAR_ERR;
uint32_t  RSVD19;
__O uint32_t  StatusInt;
uint32_t  RSVD191;

__IO uint32_t  ReqSrcReg;
uint32_t  RSVD20;
__IO uint32_t  ReqDstReg;
uint32_t  RSVD21;
__IO uint32_t  SglReqSrcReg;
uint32_t  RSVD22;
__IO uint32_t  SglReqDstReg;
uint32_t  RSVD23;
__IO uint32_t  LstSrcReg;
uint32_t  RSVD24;
__IO uint32_t  LstDstReg;
uint32_t  RSVD25;

__IO uint32_t  DmaCfgReg;
uint32_t  RSVD26;
__IO uint32_t  ChEnReg;
uint32_t  RSVD27;
__I uint32_t  DmaldReg;
uint32_t  RSVD28;
__IO uint32_t  DmaTestReg;
uint32_t  RSVD29;

} GDMA_TypeDef;
```

错误!未找到引用源。例举 GDMA 所有寄存器。

表 4-1 GDMA 寄存器

寄存器	描述
RAW_TFR	屏蔽前传输完成中断状态寄存器
RSVD0	保留
RAW_BLOCK	屏蔽前 Block 传输完成中断状态寄存器
RSVD1	保留
RAW_SRC_TRAN	屏蔽前 Source 端传输完成中断状态寄存器
RSVD2	保留
RAW_DST_TRAN	屏蔽前 Destination 端传输完成中断状态寄存器

RSVD3	保留
RAW_ERR	屏蔽前传输错误中断状态寄存器
RSVD4	保留
STATUS_TFR	传输完成中断状态寄存器
RSVD5	保留
STATUS_BLOCK	block 传输完成中断状态寄存器
RSVD6	保留
STATUS_SRC_TRAN	源端传输完成中断状态寄存器
RSVD7	保留
STATUS_DST_TRAN	目的端传输完成中断状态寄存器
RSVD8	保留
STATUS_ERR	传输错误中断状态寄存器
RSVD9	保留
MASK_TFR	传输完成中断屏蔽寄存器
RSVD10	保留
MASK_BLOCK	block 传输完成中断屏蔽寄存器
RSVD11	保留
MASK_SRC_TRAN	源端传输完成中断屏蔽寄存器
RSVD12	保留
MASK_DST_TRAN	目的端传输完成中断屏蔽寄存器
RSVD13	保留
MASK_ERR	传输错误中断屏蔽寄存器
RSVD14	保留
CLEAR_TFR	传输完成中断清除寄存器
RSVD15	保留
CLEAR_BLOCK	block 传输完成中断清除寄存器
RSVD16	保留
CLEAR_SRC_TRAN	源端传输完成中断清除寄存器
RSVD17	保留
CLEAR_DST_TRAN	目的端传输完成中断清除寄存器
RSVD18	保留
CLEAR_ERR	传输错误中断清除寄存器
RSVD19	保留
StatusInt	所有中断类型状态寄存器
RSVD191	保留
ReqSrcReg	Source 端传输请求寄存器
RSVD20	保留
ReqDstReg	Destination 端传输请求寄存器
RSVD21	保留
SglReqSrcReg	Single source 端传输请求寄存器
RSVD22	保留
SglReqDstReg	Single destination 端传输请求寄存器

RSVD23	保留
LstSrcReg	Last source 端传输请求寄存器
RSVD24	保留
LstDstReg	Last destination 端传输请求寄存器
RSVD25	保留
DmaCfgReg	GDMA 配置寄存器
RSVD26	保留
ChEnReg	通道使能寄存器
RSVD27	保留
DmaldReg	GDMA ID 寄存器
RSVD28	保留
DmaTestReg	GDMA Test 寄存器
RSVD29	保留

```

typedef struct
{
    __IO uint32_t SAR;
    uint32_t RSVD0;
    __IO uint32_t DAR;
    uint32_t RSVD1;
    __IO uint32_t LLP;
    uint32_t RSVD2;
    __IO uint32_t CTL_LOW;
    __IO uint32_t CTL_HIGH;
    __IO uint32_t SSTAT;
    uint32_t RSVD4;
    __IO uint32_t DSTAT;
    uint32_t RSVD5;
    __IO uint32_t SSTATAR;
    uint32_t RSVD6;
    __IO uint32_t DSTATAR;
    uint32_t RSVD7;
    __IO uint32_t CFG_LOW;
    __IO uint32_t CFG_HIGH;
    __IO uint32_t SGR;
    uint32_t RSVD9;
    __IO uint32_t DSR;
    uint32_t RSVD10;
} GDMA_ChannelTypeDef;

```

错误!未找到引用源。例举 GDMA_Channel 所有寄存器。

表 4-2 GDMA_Channel 寄存器

寄存器	描述
SAR	GDMA 通道源地址寄存器

RSVDO	保留
DAR	GDMA 通道目的地址寄存器
RSVD1	保留
LLP	GDMA 通道链表指针寄存器
RSVD2	保留
CTL_LOW	GDMA 通道控制寄存器(L)
CTL_HIGH	GDMA 通道控制寄存器(H)
SSTAT	GDMA 通道源状态寄存器
RSVD4	保留
DSTAT	GDMA 通道目的状态寄存器
RSVD5	保留
SSTATAR	GDMA 通道源状态地址寄存器
RSVD6	保留
DSTATAR	GDMA 通道目的状态地址寄存器
RSVD7	保留
CFG_LOW	GDMA 通道配置寄存器(L)
CFG_HIGH	GDMA 通道配置寄存器(H)
SGR	GDMA 通道源收集寄存器
RSVD9	保留
DSR	GDMA 通道目的分散寄存器
RSVD10	保留

4.2 GDMA 库函数

表 4-3 例举 GDMA 的所有库函数

函数名	描述
GDMA_DelInit	关闭 GDMA 时钟源
GDMA_Init	根据 GDMA_InitStruct 中指定参数初始化 GDMA 寄存器
GDMA_StructInit	把 GDMA_InitStruct 中的每一个参数按缺省值填入
GDMA_Cmd	使能或失能指定的 GDMA 通道
GDMA_INTConfig	使能或失能指定的 GDMA 通道中断
GDMA_ClearINTPendingBit	清除挂起的指定类型中断
GDMA_GetChannelStatus	检测指定的 GDMA 通道状态是否处于使用状态
GDMA_GetTransferINTStatus	检测指定的 GDMA 通道的总传输完成中断状态
GDMA_ClearAllTypeINT	清除指定通道的所有 GDMA 中断状态
GDMA_SetSourceAddress	设置指定 GDMA 通道的源地址
GDMA_SetDestinationAddress	设置指定 GDMA 通道的目标地址
GDMA_SetBufferSize	设置指定 GDMA 通道的传输数据长度
GDMA_SuspendCmd	暂停 GDMA 通道数据传输
GDMA_GetFIFOStatus	检测 GDMA 通道 FIFO 状态
GDMA_GetTransferLen	获取 GDMA 通道已传输数据长度

GDMA_SetLLPAddress	设置 GDMA 通道链表指针地址
GDMA_GetSuspendChannelStatus	检测指定的 GDMA 通道暂停状态
GDMA_GetSuspendCmdStatus	检测指定的 GDMA 通道暂停指令状态

4.2.1 函数 GDMA_DeInit

表 4-4 函数 GDMA_DeInit

函数名	GDMA_DeInit
函数原型	void GDMA_DeInit(void)
功能描述	关闭 GDMA 时钟源
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	RCC_PeriphClockCmd 函数

例：

```
/* Close GDMA peripheral clock */
GDMA_DeInit();
```

4.2.2 函数 GDMA_Init

表 4-5 函数 GDMA_Init

函数名	GDMA_Init
函数原型	void GDMA_Init(GDMA_ChannelTypeDef* GDMA_Channelx, GDMA_InitTypeDef* GDMA_InitStruct)
功能描述	根据 GDMA_InitStruct 中指定参数初始化 GDMA 寄存器
输入参数 1	GDMA_Channelx: x 可以被设置 0 到 5 GDMA 通道
输入参数 2	GDMA_InitStruct: 指向 GDMA_InitTypeDef 的指针, GDMA_InitStruct 中包含相关配置信息
输出参数	无
返回值	无
先决条件	无
被调用函数	无

```
typedef struct
{
    uint8_t GDMA_ChannelNum;
    uint8_t GDMA_DIR;
    uint32_t GDMA_BufferSize;
    uint8_t GDMA_SourceInc;
    uint8_t GDMA_DestinationInc;
    uint32_t GDMA_SourceDataSize;
```

```
uint32_t GDMA_DestinationDataSize;  
uint32_t GDMA_SourceMsize;  
uint32_t GDMA_DestinationMsize;  
uint32_t GDMA_SourceAddr;  
uint32_t GDMA_DestinationAddr;  
uint32_t GDMA_ChannelPriority;  
uint32_t GDMA_Multi_Block_Struct;  
uint8_t  GDMA_Multi_Block_En;  
uint8_t  GDMA_Scatter_En;  
uint8_t  GDMA_Gather_En;  
uint32_t GDMA_GatherCount;  
uint32_t GDMA_GatherInterval;  
uint32_t GDMA_ScatterCount;  
uint32_t GDMA_ScatterInterval;  
uint32_t GDMA_Multi_Block_Mode;  
uint8_t  GDMA_SourceHandshake;  
uint8_t  GDMA_DestHandshake;  
}GDMA_InitTypeDef;
```

ChannelNum: 指定的 GDMA 外设通道号，可选 0~5，总共 6 个通道。

GDMA_DIR: GDMA_DIR 规定 GDMA 的传输方向。错误!未找到引用源。给出了该参数的取值范围。

表 4-6 GDMA_DIR 值

GDMA_DIR	描述
GDMA_DIR_MemoryToMemory	内存到内存传输
GDMA_DIR_MemoryToPeripheral	内存到外设传输
GDMA_DIR_PeripheralToMemory	外设到内存传输
GDMA_DIR_PeripheralToPeripheral	外设到外设传输

GDMA_BufferSize: 规定完成一次 GDMA 总传输的数据大小。该传输分为多个 GDMA burst 传输。

GDMA_SourceInc: 规定 GDMA 操作的源地址是递增的还是递减的，错误!未找到引用源。给出了该参数的取值范围。

表 4.1 GDMA_SourceInc 值

GDMA_SourceInc	描述
DMA_SourceInc_Inc	源端地址递增
DMA_SourceInc_Dec	源端地址递减
DMA_SourceInc_Fix	源端地址固定

GDMA_DestinationInc: 规定 GDMA 操作的目的地址是递增的还是递减的，错误!未找到引用源。给出了该参数的取值范围。

表 4-7 GDMA_DestinationInc 值

GDMA_DestinationInc	描述
DMA_DestinationInc_Inc	目的端地址递增

DMA_DestinationInc_Dec	目的端地址递减
DMA_DestinationInc_Fix	目的端地址固定

GDMA_SourceDataSize: 规定了源数据宽度，**错误!未找到引用源。** 给出了该参数的取值范围。

表 4-8 GDMA_SourceDataSize 值

GDMA_SourceDataSize	描述
GDMA_DataSize_Byte	源端的数据宽度为 8 位
GDMA_DataSize_HalfWord	源端的数据宽度为 16 位
GDMA_DataSize_Word	源端的数据宽度为 32 位

GDMA_DestinationDataSize: 规定了目的数据宽度，**错误!未找到引用源。** 给出了该参数的取值范围。

表 4-9 GDMA_DestinationDataSize 值

GDMA_DestinationDataSize	描述
GDMA_DataSize_Byte	目的端的数据宽度为 8 位
GDMA_DataSize_HalfWord	目的端的数据宽度为 16 位
GDMA_DataSize_Word	目的端的数据宽度为 32 位

GDMA_SourceMsize: 规定了每次 burst 传输中，源端的数据长度。其中，数据的最小单位为

GDMA_SourceDataSize 参数设置的数据宽度。**错误!未找到引用源。** 给出了该参数的取值范围。

表 4-10 GDMA_SourceMsize 值

GDMA_SourceMsize	描述
GDMA_Msize_1	单次 burst 传输的数据个数为 1
GDMA_Msize_4	单次 burst 传输的数据个数为 4
GDMA_Msize_8	单次 burst 传输的数据个数为 8
GDMA_Msize_16	单次 burst 传输的数据个数为 16
GDMA_Msize_32	单次 burst 传输的数据个数为 32
GDMA_Msize_64	单次 burst 传输的数据个数为 64
GDMA_Msize_128	单次 burst 传输的数据个数为 128
GDMA_Msize_256	单次 burst 传输的数据个数为 256

GDMA_DestinationMsize: 规定了每次 burst 传输中，目的端的数据长度。其中，数据的最小单位为

GDMA_DestinationDataSize 参数设置的数据宽度。**错误!未找到引用源。** 给出了该参数的取值范围。

表 4-11 GDMA_DestinationMsize 值

GDMA_DestinationMsize	描述
GDMA_Msize_1	单次 burst 传输的数据个数为 1
GDMA_Msize_4	单次 burst 传输的数据个数为 4
GDMA_Msize_8	单次 burst 传输的数据个数为 8
GDMA_Msize_16	单次 burst 传输的数据个数为 16
GDMA_Msize_32	单次 burst 传输的数据个数为 32
GDMA_Msize_64	单次 burst 传输的数据个数为 64
GDMA_Msize_128	单次 burst 传输的数据个数为 128
GDMA_Msize_256	单次 burst 传输的数据个数为 256

注意：GDMA_SourceDataSize、GDMA_SourceMsize、GDMA_DestinationDataSize、GDMA_DestinationMsize
参数配置需要满足以下公式：

$\text{GDMA_SourceDataSize} * \text{GDMA_SourceMsize} = \text{GDMA_DestinationDataSize} * \text{GDMA_DestinationMsize}$ 。

GDMA_SourceAddr: 设置 GDMA 通道数据传输的源端的存取数据地址。

GDMA_DestinationAddr: 设置 GDMA 通道数据传输的目的端的存取数据地址。

GDMA_ChannelPriority: 指定 GDMA 传输通道的软件优先级。该参数取值范围为 0~5。其中通道 0 固定为 0，且优先级最高，通道 5 固定为 5，且优先级最低。

GDMA_Multi_Block_Struct: 在 Multi-block 传输中，设置传输 LLI 类型结构体首地址。

GDMA_Multi_Block_En: 使能或者失能 Multi-block 传输。

GDMA_Scatter_En: 使能或者失能 scatter 传输。

GDMA_Gather_En: 使能或者失能 gather 传输。

GDMA_GatherCount: 设置 GDMA 在 gather 传输中连续取数据的长度。

GDMA_GatherInterval: 设置 GDMA 在 gather 传输中的地址间隔长度。

GDMA_ScatterCount: 设置 GDMA 在 scatter 传输中连续存取数据的长度。

GDMA_ScatterInterval: 设置 GDMA 在 scatter 传输中的地址间隔长度。

GDMA_Multi_Block_Mode: 设置 GDMA multi-block 传输类型。每次 block 传输结束后可以选择自动加载初始值或者加载 LLI 结构体中相应值或者加载相应寄存器当前值。**错误!未找到引用源。** 给出了该参数的取值范围。

表 4-12 GDMA_Multi_Block_Mode 值

GDMA_Multi_Block_Mode	描述
AUTO_RELOAD_WITH_CONTIGUOUS_SAR	每次 block 传输后自动加载源地址初始值 每次 block 传输后自动加载目的地址寄存器当前值
AUTO_RELOAD_WITH_CONTIGUOUS_DAR	每次 block 传输后自动加载目的地址初始值 每次 block 传输后自动加载源地址寄存器当前值
AUTO_RELOAD_TRANSFER	每次 block 传输后自动加载源地址与目的地址初始值
LLI_WITH_CONTIGUOUS_SAR	每次 block 传输后自动加载源地址寄存器当前值 每次 block 传输后自动加载 LLI 结构体中目的地址值
LLI_WITH_AUTO_RELOAD_SAR	每次 block 传输后自动加载源地址初始值 每次 block 传输后自动加载 LLI 结构体中目的地址值
LLI_WITH_CONTIGUOUS_DAR	每次 block 传输后自动加载目的地址寄存器当前值 每次 block 传输后自动加载 LLI 结构体中源地址值
LLI_WITH_AUTO_RELOAD_DAR	每次 block 传输后自动加载 LLI 结构体中源地址值 每次 block 传输后自动加载目的地址初始值
LLI_TRANSFER	每次 block 传输后自动加载 LLI 结构体中源地址与目的地址值

GDMA_SourceHandshake: 设置 GDMA 源地址端 handshake 索引值。**错误!未找到引用源。** 给出了该参数的取值范围。

表 4-13 GDMA_SourceHandshake 值

GDMA_SourceHandshake	描述
GDMA_Handshake_UART0_RX	UART0 接收
GDMA_Handshake_UART2_RX	UART2 接收
GDMA_Handshake_SPI0_RX	SPI0 接收
GDMA_Handshake_SPI1_RX	SPI1 接收

GDMA_Handshake_I2C0_RX	I2C0 接收
GDMA_Handshake_I2C1_RX	I2C1 接收
GDMA_Handshake_ADC	ADC 数据读取
GDMA_Handshake_AES_RX	AES 接收
GDMA_Handshake_SPORT0_RX	I2S0 接收
GDMA_Handshake_SPORT1_RX	I2S1 接收
GDMA_Handshake_SPIC_RX	SPIC 接收
GDMA_Handshake_UART1_RX	UART 接收

GDMA_DestHandshake: 设置 GDMA 目的地址端 handshake 索引值。错误!未找到引用源。给出了该参数的取值范围。

表 4-14 GDMA_DestHandshake 值

GDMA_DestHandshake	描述
GDMA_Handshake_UART0_TX	UART0 发送
GDMA_Handshake_UART2_TX	UART2 发送
GDMA_Handshake_SPI0_TX	SPI0 发送
GDMA_Handshake_SPI1_TX	SPI1 发送
GDMA_Handshake_I2C0_TX	I2C0 发送
GDMA_Handshake_I2C1_TX	I2C1 发送
GDMA_Handshake_AES_TX	AES 发送
GDMA_Handshake_UART1_TX	UART1 发送
GDMA_Handshake_SPORT0_TX	I2S0 发送
GDMA_Handshake_SPORT1_TX	I2S1 发送
GDMA_Handshake_SPDIF_TX	SPDIF 发送
GDMA_Handshake_IF8080	IF8080 IF8080 接口
GDMA_Handshake_TIM0	TIM0
GDMA_Handshake_TIM1	TIM1
GDMA_Handshake_TIM2	TIM2
GDMA_Handshake_TIM3	TIM3
GDMA_Handshake_TIM4	TIM4
GDMA_Handshake_TIM5	TIM5
GDMA_Handshake_TIM6	TIM6
GDMA_Handshake_TIM7	TIM7

例：

```
/* Initialize GDMA */
GDMA_InitTypeDef GDMA_InitStruct;
GDMA_StructInit(&GDMA_InitStruct);
GDMA_InitStruct.GDMA_ChannelNum          = AudioTrans_GDMA_Channel_NUM;
GDMA_InitStruct.GDMA_DIR                 = GDMA_DIR_MemoryToPeripheral;
GDMA_InitStruct.GDMA_BufferSize          = 0;
GDMA_InitStruct.GDMA_SourceInc           = DMA_SourceInc_Inc;
GDMA_InitStruct.GDMA_DestinationInc     = DMA_DestinationInc_Fix;
GDMA_InitStruct.GDMA_SourceDataSize      = GDMA_DataSize_Word;
```

```

GDMA_InitStruct.GDMA_DestinationDataSize = GDMA_DataSize_Word;
GDMA_InitStruct.GDMA_SourceMsize = GDMA_Msize_16;
GDMA_InitStruct.GDMA_DestinationMsize = GDMA_Msize_16;
GDMA_InitStruct.GDMA_SourceAddr = 0;
GDMA_InitStruct.GDMA_DestinationAddr = (uint32_t)(&(I2S_USR->TX_DR));
GDMA_InitStruct.GDMA_DestHandshake = GDMA_Handshake_I2S_USR_TX;
GDMA_Init(AudioTrans_GDMA_Channel, &GDMA_InitStruct);

```

4.2.3 函数 **GDMA_StructInit**

表 4-15 函数 **GDMA_StructInit**

函数名	GDMA_StructInit
函数原型	void GDMA_StructInit(GDMA_InitTypeDef* GDMA_InitStruct)
功能描述	把 GDMA_InitStruct 中的每一个参数按缺省值填入
输入参数	GDMA_InitStruct: 指向结构 GDMA_InitTypeDef 的指针, 包含了外设 GDMA 的配置信息
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```

/* Configure the GDMA Init Structure parameter */
GDMA_InitTypeDef GDMA_InitStruct;
GDMA_StructInit(&GDMA_InitStruct);

```

4.2.4 函数 **GDMA_Cmd**

表 4-16 函数 **GDMA_Cmd**

函数名	GDMA_Cmd
函数原型	void GDMA_Cmd(uint8_t GDMA_ChannelNum, FunctionalState NewState)
功能描述	使能或失能指定的 GDMA 通道
输入参数 1	GDMA_ChannelNum: GDMA 通道, GDMA_ChannelNum 可以设置为 0 到 5
输入参数 2	NewState: 使能或者失能 GDMA 通道
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```

/* Enable specify GDMA channel */
GDMA_Cmd(1, ENABLE);

```

4.2.5 函数 GDMA_INTConfig

表 4-17 函数 GDMA_INTConfig

函数名	GDMA_INTConfig
函数原型	void GDMA_INTConfig(uint8_t GDMA_ChannelNum, uint32_t GDMA_IT, FunctionalState NewState)
功能描述	使能或失能指定的 GDMA 通道中断
输入参数 1	GDMA_ChannelNum:GDMA 通道, GDMA_ChannelNum 可以设置为 0 到 5
输入参数 2	GDMA_IT: 指定使能中断类型, 具体如错误!未找到引用源。所示
输入参数 3	NewState: 使能或者失能参数 2 指定类型的中断 可选参数: ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

GDMA_IT: 允许的 GDMA 中断源如错误!未找到引用源。所示, 使用操作符“|”可以一次选中多个中断作为该参数的值。

表 4-18 GDMA_IT 值

GDMA_IT	描述
GDMA_INT_Transfer	GDMA 总传输完成中断
GDMA_INT_Block	GDMA block 传输完成中断(等同于 GDMA 总传输完成中断)
GDMA_INT_SrcTransfer	源地址端传输完成中断
GDMA_INT_DstTransfer	目的地址端传输完成中断
GDMA_INT_Error	传输错误中断

例:

```
/* Enable specify GDMA interrupt */
GDMA_INTConfig (1, GDMA_INT_Transfer,ENABLE);
```

4.2.6 函数 GDMA_ClearINTPendingBit

表 4-19 函数 GDMA_ClearINTPendingBit

函数名	GDMA_ClearINTPendingBit
函数原型	void GDMA_ClearINTPendingBit(uint8_t GDMA_ChannelNum, uint32_t GDMA_IT)
功能描述	清除挂起的指定类型中断
输入参数 1	GDMA_ChannelNum:GDMA 通道, GDMA_ChannelNum 可以设置为 0 到 5
输入参数 2	GDMA_IT: 指定使能中断类型, 具体如错误!未找到引用源。所示
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Clear specify GDMA interrupt */
GDMA_ClearINTPendingBit(1, GDMA_INT_Transfer);
```

4.2.7 函数 GDMA_GetChannelStatus

表 4-20 函数 GDMA_GetChannelStatus

函数名	GDMA_GetChannelStatus
函数原型	FlagStatus GDMA_GetChannelStatus(uint8_t GDMA_Channel_Num)
功能描述	检测指定的 GDMA 通道状态是否处于使用状态
输入参数 1	GDMA_ChannelNum:GDMA 通道, GDMA_ChannelNum 可以设置为 0 到 5
输出参数	无
返回值	状态标志
先决条件	无
被调用函数	无

例:

```
/* Check GDMA channel status before use it */
BOOL useGDMAChannel = FALSE;
useGDMAChannel = GDMA_GetChannelStatus (1);
```

4.2.8 函数 GDMA_GetTransferINTStatus

表 4-21 函数 GDMA_GetTransferINTStatus

函数名	GDMA_GetTransferINTStatus
函数原型	ITStatus GDMA_GetTransferINTStatus(uint8_t GDMA_Channel_Num)
功能描述	检测指定的 GDMA 通道的总传输完成中断状态
输入参数 1	GDMA_ChannelNum:GDMA 通道, GDMA_ChannelNum 可以设置为 0 到 5
输出参数	无
返回值	状态标志
先决条件	无
被调用函数	无

例:

```
/* Check GDMA transfer finish or not */
BOOL isTransfer= FALSE;
isTransfer = GDMA_GetTransferINTStatus (1);
```

4.2.9 函数 GDMA_ClearAllTypeINT

表 4-22 函数 GDMA_ClearAllTypeINT

函数名	GDMA_ClearAllTypeINT
函数原型	void GDMA_ClearAllTypeINT(uint8_t GDMA_Channel_Num)

功能描述	清除指定通道的所有 GDMA 中断状态
输入参数 1	GDMA_ChannelNum:GDMA 通道, GDMA_ChannelNum 可以设置为 0 到 5
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Clear all interrupt */
GDMA_ClearAllTypeINT (1);
```

4.2.10 函数 **GDMA_SetSourceAddress**

表 4-23 函数 **GDMA_SetSourceAddress**

函数名	GDMA_SetSourceAddress
函数原型	void GDMA_SetSourceAddress(GDMA_ChannelTypeDef* GDMA_Channelx, uint32_t Address)
功能描述	设置指定 GDMA 通道的源地址
输入参数 1	GDMA_Channelx: GDMA 通道, x 可以被设置 0 到 5
输入参数 2	设置指定 GDMA 通道的源地址
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Configure source address */
uint8_t transferBuffer[256];
GDMA_SetSourceAddress(GDMA_Channel1, transferBuffer);
```

4.2.11 函数 **GDMA_SetDestinationAddress**

表 4-24 函数 **GDMA_SetDestinationAddress**

函数名	GDMA_SetDestinationAddress
函数原型	void GDMA_SetDestinationAddress(GDMA_ChannelTypeDef* GDMA_Channelx, uint32_t Address)
功能描述	设置指定 GDMA 通道的目标地址
输入参数 1	GDMA_Channelx: x 可以被设置 0 到 5 GDMA 通道
输入参数 2	设置指定 GDMA 通道的目标地址
输出参数	无
返回值	无
先决条件	无
被调用函数	无

```
/* Configure destination address */  
uint8_t transferBuffer[256];  
GDMA_SetDestinationAddress (GDMA_Channel1, transferBuffer);
```

4.2.12 函数 GDMA_SetBufferSize

表 4-25 函数 GDMA_SetBufferSize

函数名	GDMA_SetBufferSize
函数原型	void GDMA_SetBufferSize(GDMA_ChannelTypeDef* GDMA_Channelx, uint32_t buffer_size)
功能描述	设置一次 GDMA 总传输的数据大小，该传输单位为 source 端的数据宽度
输入参数 1	GDMA_Channelx: GDMA 通道, x 可以被设置 0 到 5
输入参数 2	设置指定 GDMA 通道的传输数据长度
输出参数	无
返回值	无
先决条件	无
被调用函数	无

```
/* Configure total number of transfer */  
GDMA_SetDestinationAddress (GDMA_Channel1, 256);
```

4.2.13 函数 GDMA_SuspendCmd

表 4-26 函数 GDMA_SuspendCmd

函数名	GDMA_SuspendCmd
函数原型	void GDMA_SuspendCmd(GDMA_ChannelTypeDef *GDMA_Channelx, FunctionalState NewState)
功能描述	暂停 GDMA 通道数据传输
输入参数 1	GDMA_Channelx: x 可以被设置 0 到 5 GDMA 通道
输入参数 2	NewState: 使能或者失能暂停参数 1 指定的 GDMA 通道数据传输 可选参数: ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

```
/* Abort GDMA channel transfer */  
GDMA_SuspendCmd(GDMA_Channel0, ENABLE);
```

4.2.14 函数 GDMA_GetFIFOStatus

表 4-27 函数 GDMA_GetFIFOStatus

函数名	GDMA_GetFIFOStatus
-----	--------------------

函数原型	FlagStatus GDMA_GetFIFOStatus(GDMA_ChannelTypeDef *GDMA_Channelx)
功能描述	检测 GDMA 通道 FIFO 状态
输入参数 1	GDMA_Channelx: x 可以被设置 0 到 5 GDMA 通道
输出参数	无
返回值	GDMA FIFO 状态: SET: empty, RESET:not empty.
先决条件	无
被调用函数	无

```
/* Check GDMA channel FIFO status*/
BOOL GDMAChannelFIFOStatus= FALSE;
GDMAChannelFIFOStatus = GDMA_GetFIFOStatus (GDMA_Channel0);
```

4.2.15 函数 GDMA_GetTransferLen

表 4-28 函数 GDMA_GetTransferLen

函数名	GDMA_GetTransferLen
函数原型	uint16_t GDMA_GetTransferLen (GDMA_ChannelTypeDef *GDMA_Channelx)
功能描述	获取 GDMA 通道已传输数据长度
输入参数 1	GDMA_Channelx: x 可以被设置 0 到 5 GDMA 通道
输出参数	无
返回值	GDMA 通道传输数据长度。
先决条件	无
被调用函数	无

```
/* Get GDMA channel transfer data length*/
Uint16_t  dataLength= 0;
dataLength = GDMA_GetTransferLen (GDMA_Channel0);
```

4.2.16 函数 GDMA_SetLLPAddress

表 4-29 函数 GDMA_SetLLPAddress

函数名	GDMA_SetLLPAddress
函数原型	void GDMA_SetLLPAddress(GDMA_ChannelTypeDef *GDMA_Channelx, uint32_t Address)
功能描述	设置 GDMA 通道链表指针地址
输入参数 1	GDMA_Channelx: x 可以被设置 0 到 5 GDMA 通道
输入参数 2	设置指定 GDMA 通道的链表指针地址
输出参数	无
返回值	无
先决条件	无
被调用函数	无

```
/* Set GDMA channel LLP address*/
GDMA_LLIDef  GDMA_LLIStruct[16];
GDMA_SetLLPAddress (GDMA_Channel0, GDMA_LLIStruct);
```

4.2.17 函数 GDMA_GetSuspendChannelStatus

表 4-30 函数 GDMA_GetSuspendChannelStatus

函数名	GDMA_GetSuspendChannelStatus
函数原型	FlagStatus GDMA_GetSuspendChannelStatus(GDMA_ChannelTypeDef *GDMA_Channelx)
功能描述	检测指定 GDMA 通道暂停状态
输入参数	GDMA_Channelx: GDMA 通道, x 可以被设置 0 到 5
输出参数	无
返回值	GDMA 通道暂停状态: SET:inactive, RESET:active.
先决条件	无
被调用函数	无

```
/* Check GDMA suspend channel status*/  
FlagStatus GDMA_SuspendChannelStatus = RESET ;  
GDMA_SuspendChannelStatus = GDMA_GetSuspendChannelStatus (GDMA_Channel0);
```

4.2.18 函数 GDMA_GetSuspendCmdStatus

表 4-31 函数 GDMA_GetSuspendCmdStatus

函数名	GDMA_GetSuspendCmdStatus
函数原型	FlagStatus GDMA_GetSuspendCmdStatus (GDMA_ChannelTypeDef *GDMA_Channelx)
功能描述	检测指定 GDMA 通道暂停指令状态
输入参数	GDMA_Channelx: GDMA 通道, x 可以被设置 0 到 5
输出参数	无
返回值	GDMA 暂停通道状态: SET: suspend, RESET: not suspend.
先决条件	无
被调用函数	无

```
/* Check GDMA channel suspend cmd status*/  
FlagStatus GDMA_SuspendCmdStatus =RESET ;  
GDMA_SuspendCmdStatus = GDMA_GetSuspendCmdStatus (GDMA_Channel0);
```

5 通用输入/输出(GPIO)

5.1 GPIO 寄存器结构

```
typedef struct
{
    __IO uint32_t  DATAOUT;
    __IO uint32_t  DATADIR;
    __IO uint32_t  DATASRC;
    uint32_t      RSVDO[9];
    __IO uint32_t  INTEN;
    __IO uint32_t  INTMASK;
    __IO uint32_t  INTTYPE;
    __IO uint32_t  INTPOLARITY;
    __IO uint32_t  INTSTATUS;
    __IO uint32_t  RAWINTSTATUS;
    __IO uint32_t  DEBOUNCE;
    __O  uint32_t   INTCLR;
    __I  uint32_t   DATAIN;
    uint32_t      RSVD1[3];
    __IO uint32_t  LSSYNC;
    __I  uint32_t   IDCODE;
    __IO uint32_t  INTBOTHEdge;
} GPIO_TypeDef;
```

错误!未找到引用源。例举 GPIO 所有寄存器

表 5-1 GPIO 寄存器

寄存器	描述
DATAOUT	数据输出寄存器
DATADIR	数据方向寄存器
DATASRC	数据源寄存器
RSVD0[9]	保留
INTEN	中断控制寄存器
INTMASK	中断屏蔽控制寄存器
INTTYPE	中断类型控制寄存器
INTPOLARITY	中断极性控制寄存器
INTSTATUS	中断状态寄存器
RAWINTSTATUS	原始状态寄存器(不受中断屏蔽寄存器屏蔽)
DEBOUNCE	去抖动参数设置寄存器
INTCLR	中断清除寄存器
DATAIN	外部电平输入寄存器

RSVD1[3]	保留
LSSYNC	在电平触发方式下中断信号同步使能寄存器
IDCODE	ID 寄存器
INTBOTHEDGE	双边沿触发中断寄存器

5.2 GPIO 库函数

错误!未找到引用源。例举 GPIO 的所有库函数

表 5-2 GPIO 库函数

函数名	描述
GPIO_DeInit	关闭 GPIO 时钟源
GPIO_GetPin	获取 GPIO 管脚值
GPIO_Init	根据 GPIO_InitStruct 中指定参数初始化 GPIO 寄存器
GPIO_StructInit	把 GPIO_InitStruct 中的每一个参数按缺省值填入
GPIO_INTConfig	使能或失能 GPIO 外部中断
GPIO_ClearINTPendingBit	清除 GPIO 中断标志
GPIO_MaskINTConfig	屏蔽或者不屏蔽 GPIO 外部中断
GPIO_ReadInputDataBit	读取指定管脚的输入
GPIO_ReadInputData	读取所有 GPIO 端口输入
GPIO_ReadOutputDataBit	读取指定管脚的输出
GPIO_ReadOutputData	读取所有 GPIO 端口输出
GPIO_SetBits	设置指定管脚为高电平
GPIO_ResetBits	设置指定管脚为低电平
GPIO_WriteBit	设置或者清除指定管脚的电平值
GPIO_Write	设置或者清除所有 GPIO 端口的电平值
GPIO_GetINTStatus	读取 GPIO 指定管脚的中断标志位
GPIO_GetNum	获取 pin 对应的 GPIO 序号值
GPIO_Debounce_Time	设置 GPIO 中断 Debounce 时间
GPIO_DBClkCmd	使能或失能 GPIO Debounce Clock

5.2.1 函数 GPIO_DeInit

表 5-3 函数 GPIO_DeInit

函数名	GPIO_DeInit
函数原型	void GPIO_DeInit(void)
功能描述	关闭 GPIO 时钟源
输入参数	无
输出参数	无
返回值	无
先决条件	无

被调用函数	RCC_PeriphClockCmd()
-------	----------------------

例：

```
/* Close GPIO clock */
GPIO_DeInit();
```

5.2.2 函数 GPIO_GetPin

表 5-4 函数 GPIO_GetPin

函数名	GPIO_GetPin
函数原型	uint32_t GPIO_GetPin(uint8_t Pin_num)
功能描述	获取初始化参数 GPIO_Pin 的值
输入参数	Pin_num：外部管脚值，该参数可选择：P0_0 至 P3_6 或 P4_0 至 P4_3 或 H0 至 H2。 当输入参数 Pin_num 值为 P0_0 至 P3_6，该函数的返回值依次为 GPIO_Pin_0 至 GPIO_Pin_30。当输入参数 Pin_num 值为 P4_0 至 P4_3，该函数的返回值依次为 GPIO_Pin_28 至 GPIO_Pin_31。当输入参数 Pin_num 值为 H0 至 H2，该函数的返回值依次为 GPIO_Pin_25 至 GPIO_Pin_27。注意不能将多根管脚配置成相同 GPIO 索引号上。例如，当 P3_4 已经配置成 GPIO_Pin_28 时，P4_0 不能配置成 GPIO_Pin_28。
输出参数	无
返回值	返回参数 GPIO_Pin 值
先决条件	无
被调用函数	无

例：

```
/* Get specified GPIO Pin value*/
uint32_t GPIO_UsePin = GPIO_GetPin(P0_2);
```

5.2.3 函数 GPIO_Init

表 5-5 函数 GPIO_Init

函数名	GPIO_Init
函数原型	void GPIO_Init(GPIO_InitTypeDef* GPIO_InitStruct)
功能描述	根据 GPIO_InitStruct 中指定参数初始化 GPIO 寄存器
输入参数	GPIO_InitStruct：指向结构 GPIO_InitTypeDef 的指针，包含了外设 GPIO 的配置信息
输出参数	无
返回值	无
先决条件	无
被调用函数	无

```
GPIO_InitTypeDef structure
typedef struct
{
    uint32_t          GPIO_Pin;
    GPIO_MODE_TypeDef GPIO_Mode;
```

```

FunctionalState      GPIO_ITCmd;
GPIOIT_LevelType    GPIO_ITTrigger;
GPIOIT_PolarityType GPIO_ITPolarity;
GPIOIT_DebounceType GPIO_ITDebounce;
GPIOControlMode_Typedef GPIO_ControlMode;
uint32_t             GPIO_DebounceTime;

}GPIO_InitTypeDef;

```

GPIO_Pin: 该参数选择需要设置的 GPIO 管脚，使用操作符 “|” 可以一次选中多个管脚。通过 `GPIO_GetPin` 函数获取该参数，参阅 `GPIO_GetPin` 函数描述。

GPIO_Mode: 该参数用以设置选中管脚的工作状态。表 5.6 给出了该参数可取的值

表 5-6 GPIO_Mode 值

GPIO_Mode	描述
GPIO_Mode_IN	输入模式
GPIO_Mode_OUT	输出模式

GPIO_ITCmd: 该参数用以设置选中管脚的工作状态。表 5.7 给出了该参数可取的值

表 5.1 GPIO_ITCmd 值

GPIO_ITCmd	描述
DISABLE	不配置 GPIO 中断模式
ENABLE	配置 GPIO 中断模式

GPIO_ITTrigger: 该参数用以设置选中管脚在中断模式下的工作状态。表 5.8 给出了该参数可取的值

表 5-7 GPIO_ITTrigger 值

GPIO_ITTrigger	描述
GPIO_INT_Trigger_LEVEL	电平触发中断
GPIO_INT_Trigger_EDGE	边沿触发中断
GPIO_INT_BOTH_EDGE	双边沿触发中断

GPIO_ITPolarity: 该参数用以设置选中管脚在中断模式下的工作状态。表 5.9 给出了该参数可取的值

表 5-8 GPIO_ITPolarity 值

GPIO_ITPolarity	描述
GPIO_INT_POLARITY_ACTIVE_LOW	低电平触发或者下降沿触发
GPIO_INT_POLARITY_ACTIVE_HIGH	高电平触发或者上升沿触发

GPIO_ITDebounce: 该参数用以设置选中管脚在中断模式下的工作状态。表 5.10 给出了该参数可取的值

表 5-9 GPIO_ITDebounce 值

GPIO_ITPolarity	描述
GPIO_INT_DEBOUNCE_DISABLE	失能中断去抖动
GPIO_INT_DEBOUNCE_ENABLE	使能中断去抖动

GPIO_ControlMode: 该参数用以设置 GPIO 工作模式。错误!未找到引用源。给出了该参数可取的值

表 5-10 GPIO_ControlMode 值

GPIO_ControlMode	描述
GPIO_SOFTWARE_MODE	软件模式，一般操作使用该模式
GPIO_HARDWARE_MODE	硬件模式，用于 GDMA 定时控制 GPIO 管脚

GPIO_DebounceTime: 该参数用以设置 GPIO 去抖动时间，单位是 ms。该参数取值范围为 1 至 64。

例：

```
GPIO_InitTypeDef GPIO_InitStruct;
GPIO_StructInit(&GPIO_InitStruct);
GPIO_InitStruct.GPIO_Pin = GPIO_GetPin(P0_0);
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN;
GPIO_InitStruct.GPIO_ITCmd = ENABLE;
GPIO_InitStruct.GPIO_ITTrigger = GPIO_INT_Trigger_EDGE;
GPIO_InitStruct.GPIO_ITPolarity = GPIO_INT_POLARITY_ACTIVE_LOW;
GPIO_InitStruct.GPIO_ITDebounce = GPIO_INT_DEBOUNCE_ENABLE;
GPIO_InitStruct.GPIO_DebounceTime = 30;
GPIO_Init(&GPIO_InitStruct);
```

5.2.4 函数 GPIO_StructInit

表 5-11 函数 GPIO_StructInit

函数名	GPIO_StructInit
函数原型	void GPIO_StructInit(GPIO_InitTypeDef* GPIO_InitStruct)
功能描述	把 GPIO_InitStruct 中的每一个参数按缺省值填入
输入参数	GPIO_InitStruct: 指向结构 GPIO_InitTypeDef 的指针，包含了外设 GPIO 的配置信息
输出参数	无
返回值	无
先决条件	无
被调用函数	无

表 5-12 GPIO_InitStruct 缺省值

成员	缺省值
GPIO_Pin	GPIO_Pin_All
GPIO_Mode	GPIO_Mode_IN
GPIO_ITCmd	DISABLE
GPIO_ITTrigger	GPIO_INT_Trigger_LEVEL
GPIO_ITPolarity	GPIO_INT_POLARITY_ACTIVE_LOW
GPIO_ITDebounce	GPIO_INT_DEBOUNCE_DISABLE

例：

```
/* Configure the GPIO Init Structure parameter */
GPIO_InitTypeDef GPIO_InitStruct;
GPIO_StructInit(&GPIO_InitStruct);
```

5.2.5 函数 GPIO_INTConfig

表 5-13 函数 GPIO_INTConfig

函数名	GPIO_INTConfig
函数原型	void GPIO_INTConfig(uint32_t GPIO_Pin, FunctionalState NewState)
功能描述	使能或失能 GPIO 外部中断，管脚与 GPIO 中断函数对应如错误!未找到引用源。
输入参数 1	GPIO_Pin: 通过 GPIO_GetPin 函数返回值获取
输入参数 2	NewState: GPIO_Pin 中断的新状态 可选参数: ENABLE(使能)或者 DISABLE(失能)
输出参数	无
返回值	无
先决条件	无
被调用函数	无

表 5-14 管脚与 GPIO 中断函数对应表

Pin_Num	GPIO 中断函数
P0_0	GPIO0_Handler
P0_1	GPIO1_Handler
P0_2	GPIO2_Handler
P0_3	GPIO3_Handler
P0_4	GPIO4_Handler
P0_5	GPIO5_Handler
P0_6	GPIO6_Handler
P0_7	GPIO7_Handler
P1_0	GPIO8_Handler
P1_1	GPIO9_Handler
P1_2	GPIO10_Handler
P1_3	GPIO11_Handler
P1_4	GPIO12_Handler
P1_5	GPIO13_Handler
P1_6	GPIO14_Handler
P1_7	GPIO15_Handler
P2_0	GPIO16_Handler
P2_1	GPIO17_Handler
P2_2	GPIO18_Handler
P2_3	GPIO19_Handler
P2_4	GPIO20_Handler
P2_5	GPIO21_Handler
P2_6	GPIO22_Handler
P2_7	GPIO23_Handler
P3_0	GPIO24_Handler
P3_1	GPIO25_Handler

P3_2	GPIO26_Handler
P3_3	GPIO27_Handler
P3_4	GPIO28_Handler
P3_5	GPIO29_Handler
P3_6	GPIO30_Handler
P4_0	GPIO28_Handler
P4_1	GPIO29_Handler
P4_2	GPIO30_Handler
P4_3	GPIO31_Handler
H_0	GPIO25_Handler
H_1	GPIO26_Handler
H_2	GPIO27_Handler

例：

```
/* Enable P0_2 interrupt */
GPIO_INTConfig(GPIO_GetPin(P0_2), ENABLE);
```

5.2.6 函数 GPIO_ClearINTPendingBit

表 5-15 函数 GPIO_ClearINTPendingBit

函数名	GPIO_ClearINTPendingBit
函数原型	void GPIO_ClearINTPendingBit(uint32_t GPIO_Pin)
功能描述	清除 GPIO_Pin 的中断标志位
输入参数 1	GPIO_Pin: 通过 GPIO_GetPin 函数返回值获取
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear P0_2 interrupt */
GPIO_ClearINTPendingBit (GPIO_GetPin(P0_2));
```

5.2.7 函数 GPIO_MaskINTConfig

表 5-16 函数 GPIO_MaskINTConfig

函数名	GPIO_MaskINTConfig
函数原型	void GPIO_MaskINTConfig(uint32_t GPIO_Pin, FunctionalState NewState)
功能描述	屏蔽或不屏蔽 GPIO 外部中断
输入参数 1	GPIO_Pin: 通过 GPIO_GetPin 函数返回值获取

输入参数 2	NewState: GPIO_Pin 的新状态 可选参数: ENABLE(屏蔽)或者 DISABLE(不屏蔽)
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* mask P0_2 interrupt */
GPIO_MaskINTConfig (GPIO_GetPin(P0_2), ENABLE);
```

5.2.8 函数 GPIO_ReadInputDataBit

表 5-17 函数 GPIO_ReadInputDataBit

函数名	GPIO_ReadInputDataBit
函数原型	uint8_t GPIO_ReadInputDataBit(uint32_t GPIO_Pin)
功能描述	读取指定管脚的输入
输入参数	GPIO_Pin: 通过 GPIO_GetPin 函数返回值获取
输出参数	无
返回值	输入端口管脚值: 0 为低电平, 1 为高电平
先决条件	无
被调用函数	无

例:

```
/* Get specified GPIO Pin value */
uint8_t read_value = GPIO_ReadInputDataBit (GPIO_GetPin(P0_2));
```

5.2.9 函数 GPIO_ReadInputData

表 5-18 函数 GPIO_ReadInputData

函数名	GPIO_ReadInputData
函数原型	uint32_t GPIO_ReadInputData(void)
功能描述	读取所有 GPIO 端口输入
输入参数	无
输出参数	无
返回值	所有 GPIO 输入管脚值
先决条件	无
被调用函数	无

例:

```
/* Get all GPIO Pin value */
uint32_t read_value = GPIO_ReadInputData();
```

5.2.10 函数 GPIO_ReadOutputDataBit

表 5-19 函数 GPIO_ReadOutputDataBit

函数名	GPIO_ReadOutputDataBit
函数原型	uint8_t GPIO_ReadOutputDataBit(uint32_t GPIO_Pin)
功能描述	读取指定管脚的输出
输入参数	GPIO_Pin: 通过 GPIO_GetPin 函数返回值获取
输出参数	无
返回值	指定 GPIO 管脚的输出端口值
先决条件	无
被调用函数	无

例：

```
/* Get specified GPIO Pin value */
uint8_t read_value = GPIO_ReadOutputDataBit (GPIO_GetPin(P0_2));
```

5.2.11 函数 GPIO_ReadOutputData

表 5-20 函数 GPIO_ReadOutputData

函数名	GPIO_ReadOutputData
函数原型	uint32_t GPIO_ReadOutputData (void)
功能描述	读取所有 GPIO 端口输出值
输入参数	无
输出参数	无
返回值	所有 GPIO 输出管脚值
先决条件	无
被调用函数	无

例：

```
/* Get all GPIO Pin value */
uint32_t read_value = GPIO_ReadOutputData ();
```

5.2.12 函数 GPIO_SetBits

表 5-21 函数 GPIO_SetBits

函数名	GPIO_SetBits
函数原型	void GPIO_SetBits (uint32_t GPIO_Pin)
功能描述	设置指定管脚为高电平
输入参数	GPIO_Pin: 通过 GPIO_GetPin 函数返回值获取
输出参数	无
返回值	无
先决条件	无

被调用函数	无
-------	---

例：

```
/* Configure specified GPIO Pin output high level */
GPIO_SetBits(GPIO_GetPin(P0_2));
```

5.2.13 函数 GPIO_ResetBits

表 5-22 函数 GPIO_ResetBits

函数名	GPIO_ResetBits
函数原型	void GPIO_ResetBits (uint32_t GPIO_Pin)
功能描述	设置指定管脚为低电平
输入参数	GPIO_Pin: 通过 GPIO_GetPin 函数返回值获取
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Configure specified GPIO Pin output low level */
GPIO_ResetBits(GPIO_GetPin(P0_2));
```

5.2.14 函数 GPIO_WriteBit

表 5-23 函数 GPIO_WriteBit

函数名	GPIO_WriteBit
函数原型	void GPIO_WriteBit(uint32_t GPIO_Pin, BitAction BitVal)
功能描述	设置或者清除指定管脚的电平值
输入参数 1	GPIO_Pin: 通过 GPIO_GetPin 函数返回值获取
输入参数 2	BitVal: 该参数指定了待写入的值，该参数必须取枚举 BitAction 的其中一个值。 Bit_RESET: 清除数据端口位 Bit_SET: 设置数据端口位
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/*Set specified GPIO Pin */
GPIO_WriteBit(GPIO_GetPin(P0_2), Bit_SET);
```

5.2.15 函数 GPIO_Write

表 5-24 函数 GPIO_Write

函数名	GPIO_Write
函数原型	void GPIO_Write(uint32_t PortVal)
功能描述	设置或者清除指定管脚的电平值
输入参数	PortVal: 待写入端口数据寄存器的值
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Get all GPIO Pin value */
uint32_t read_value = GPIO_ReadOutputData();
/*Configure GPIO_Pin_2 output high level */
GPIO_Write (read_value | BIT(2));
```

5.2.16 函数 GPIO_GetINTStatus

表 5-25 函数 GPIO_GetINTStatus

函数名	GPIO_GetINTStatus
函数原型	ITStatus GPIO_GetINTStatus(uint32_t GPIO_Pin)
功能描述	检查指定管脚的中断发生与否
输入参数	GPIO_Pin: 通过 GPIO_GetPin 函数返回值获取
输出参数	无
返回值	指定管脚的中断标志位的值，该值为枚举 ITStatus 的其中一个值。 RESET: 未产生中断 SET: 产生中断
先决条件	无
被调用函数	无

例：

```
/* Check if the specified pin interrupt has occurred or not */
ITStatus Status;
Status = GPIO_GetINTStatus (GPIO_GetPin(P0_2));
```

5.2.17 函数 GPIO_GetNum

表 5-26 函数 GPIO_GetNum

函数名	GPIO_GetNum
函数原型	uint8_t GPIO_GetNum(uint8_t Pin_num)

功能描述	获取 pin 对应的 GPIO 序号值
输入参数	Pin_num: 外部管脚值, 该参数可选择: P0_0 到 P3_6 或者 P4_0 到 P4_3 或者 H_0 到 H_2
输出参数	无
返回值	GPIO 序列号
先决条件	无
被调用函数	无

例:

```
/* Get specified GPIO Num value*/
uint8_t GPIO_Num = GPIO_GetNum(P0_2);
```

5.2.18 函数 GPIO_Debounce_Time

表 5-27 函数 GPIO_Debounce_Time

函数名	GPIO_Debounce_Time
函数原型	void GPIO_Debounce_Time(uint32_t DebounceTime)
功能描述	设置 GPIO 中断 Debounce 时间
输入参数	DebounceTime: 时间, 单位 ms, 1ms-64ms
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Set GPIO debounce value*/
GPIO_Debounce_Time(10);
```

5.2.19 函数 GPIO_DBClkCmd

表 5-28 函数 GPIO_DBClkCmd

函数名	GPIO_DBClkCmd
函数原型	void GPIO_DBClkCmd(FunctionalState NewState)
功能描述	使能或失能 GPIO Debounce Clock
输入参数	NewState: 使能或失能 GPIO Debounce Clock 可选参数: ENABLE(使能)或者 DISABLE(失能)
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* EnableGPIO debounce clock*/
```

```
GPIO_DBClkCmd(ENABLE);
```

6 内部集成电路(I2C)

6.1 I2C 寄存器架构

```
typedef struct
{
    __IO uint32_t  IC_CON;
    __IO uint32_t  IC_TAR;
    __IO uint32_t  IC_SAR;
    __IO uint32_t  IC_HS_MADDR;
    __IO uint32_t  IC_DATA_CMD;
    __IO uint32_t  IC_SS_SCL_HCNT;
    __IO uint32_t  IC_SS_SCL_LCNT;
    __IO uint32_t  IC_FS_SCL_HCNT;
    __IO uint32_t  IC_FS_SCL_LCNT;
    __IO uint32_t  IC_HS_SCL_HCNT;
    __IO uint32_t  IC_HS_SCL_LCNT;
    __I uint32_t   IC_INTR_STAT;
    __IO uint32_t  IC_INTR_MASK;
    __I uint32_t   IC_RAW_INTR_STAT;
    __IO uint32_t  IC_RX_TL;
    __IO uint32_t  IC_TX_TL;
    __I uint32_t   IC_CLR_INTR;
    __I uint32_t   IC_CLR_RX_UNDER;
    __I uint32_t   IC_CLR_RX_OVER;
    __I uint32_t   IC_CLR_TX_OVER;
    __I uint32_t   IC_CLR_RD_REQ;
    __I uint32_t   IC_CLR_TX_ABRT;
    __I uint32_t   IC_CLR_RX_DONE;
    __I uint32_t   IC_CLR_ACTIVITY;
    __I uint32_t   IC_CLR_STOP_DET;
    __I uint32_t   IC_CLR_START_DET;
    __I uint32_t   IC_CLR_GEN_CALL;
    __IO uint32_t  IC_ENABLE;
    __I uint32_t   IC_STATUS;
    __I uint32_t   IC_TXFLR;
    __I uint32_t   IC_RXFLR;
    __IO uint32_t  IC_SDA_HOLD;
    __I uint32_t   IC_TX_ABRT_SOURCE;
    __IO uint32_t  IC_SLV_DATA_NACK_ONLY;
    __IO uint32_t  IC_DMA_CR;
```

```

__IO uint32_t IC_DMA_TDRL;
__IO uint32_t IC_DMA_RDLR;
__IO uint32_t IC_SDA_SETUP;
__IO uint32_t IC_ACK_GENERAL_CALL;
__IO uint32_t IC_ENABLE_STATUS;
}
```

错误!未找到引用源。例举 I2C 所有寄存器

表 6-1 I2C 寄存器

寄存器	描述
IC_CON	I2C 控制寄存器
IC_TAR	I2C 作为主设备目标从设备地址寄存器
IC_SAR	I2C 作为从设备地址寄存器
IC_HS_MADDR	I2C 高速主模式下编码地址
IC_DATA_CMD	I2C 接收或者发送数据/命令
IC_SS_SCL_HCNT	I2C 标准模式下时钟(H)
IC_SS_SCL_LCNT	I2C 标准模式下时钟(L)
IC_FS_SCL_HCNT	I2C 快速模式下时钟(H)
IC_FS_SCL_LCNT	I2C 快速模式下时钟(L)
IC_HS_SCL_HCNT	I2C 高速模式下时钟(H)
IC_HS_SCL_LCNT	I2C 高速模式下时钟(L)
IC_INTR_STAT	I2C 中断状态寄存器
IC_INTR_MASK	I2C 中断屏蔽寄存器
IC_RAW_INTR_STAT	I2C 原始中断状态(不受中断屏蔽寄存器屏蔽)
IC_RX_TL	I2C 接收 FIFO 阈值
IC_TX_TL	I2C 发送 FIFO 阈值
IC_CLR_INTR	清除中断标志寄存器
IC_CLR_RX_UNDER	清除接收中断(UNDER)
IC_CLR_RX_OVER	清除接收中断(OVER)
IC_CLR_TX_OVER	清除发送中断(OVER)
IC_CLR_RD_REQ	清除读请求中断
IC_CLR_TX_ABRT	清除发送中断(ABRT)
IC_CLR_RX_DONE	清除接收中断(DONE)
IC_CLR_ACTIVITY	清除活动中断
IC_CLR_STOP_DET	清除检测到 STOP 中断
IC_CLR_START_DET	清除检测到 START 中断
IC_CLR_GEN_CALL	清除产生调用中断
IC_ENABLE	I2C 使能
IC_STATUS	I2C 状态
IC_TXFLR	表明在 I2C 发送 FIFO 中有效数据
IC_RXFLR	表明在 I2C 接收 FIFO 中有效数据
IC_SDA_HOLD	I2C SDA 保持时间
IC_TX_ABRT_SOURCE	I2C 发送停止原因寄存器

IC_SLV_DATA_NACK_ONLY	产生不应答寄存器(作为从设备)
IC_DMA_CR	I2C DMA 控制寄存器
IC_DMA_TDLR	DMA 传输数据级别寄存器
IC_DMA_RDLR	DMA 读取数据级别寄存器
IC_SDA_SETUP	I2C SDA 建立寄存器(SLAVE)
IC_ACK_GENERAL_CALL	I2C 应答使能
IC_ENABLE_STATUS	I2C 使能状态寄存器

6.2 I2C 库函数

表 6-2 例举 I2C 的所有库函数

函数名	描述
I2C_DeInit	关闭 I2C 时钟源
I2C_Init	初始化 I2C 模块
I2C_StructInit	把 I2C_InitStruct 中的每一个参数按缺省值填入
I2C_Cmd	使能或不使能指定的 I2C 通道
I2C_MasterWrite	在主机模式下通过 I2C 外设发送数据
I2C_MasterRead	在主机模式下通过 I2C 外设读数据
I2C_RepeatRead	在主机模式下通过 I2C 外设连续发送和接收数据
I2C_INTConfig	使能或者失能 I2C 中断
I2C_ClearINTPendingBit	清除 I2C 中断挂起位
I2C_SetSlaveAddress	设置传输的从设备地址
I2C_SendCmd	主设备模式下通过外设 I2C 发送或者读取命令
I2C_ReceiveData	通过外设 I2C 接收一个数据
I2C_GetRxFIFOLen	从 I2C 接收 FIFO 中读取接收数据的长度
I2C_GetTxFIFOLen	从 I2C 发送 FIFO 中读取发送数据的长度
I2C_GetFlagState	检查 I2C 指定的标志位是否被设置
I2C_CheckEvent	检查 I2C 指定的传输失败事件状态
I2C_GetINTStatus	获得指定的 I2C 中断状态
I2C_ClearAllINT	清除 I2C 所有挂起的中断
I2C_GDMACmd	使能或者失能 GDMA 指定类型的数据传输功能

6.2.1 函数 I2C_DeInit

表 6-3 函数 I2C_DeInit

函数名	I2C_DeInit
函数原型	void I2C_DeInit(I2C_TypeDef* I2Cx)
功能描述	关闭指定的 I2C 时钟
输入参数 1	I2Cx: x 可以设置成 0 或者 1, 选择指定的 I2C 外设
输出参数	无

返回值	无
先决条件	无
被调用函数	RCC_PeriphClockCmd()

6.2.2 函数 I2C_Init

表 6-4 函数 I2C_Init

函数名	I2C_Init
函数原型	void I2C_Init(I2C_TypeDef* I2Cx, I2C_InitTypeDef* I2C_InitStruct)
功能描述	根据 I2C_InitStruct 中指定参数初始化 I2C 寄存器
输入参数 1	I2Cx: x 可以设置成 0 或者 1, 选择指定的 I2C 外设
输入参数 2	I2C_InitStruct: 指向 I2C_InitTypeDef 的指针, I2C_InitStruct 中包含相关配置信息
输出参数	无
返回值	无
先决条件	无
被调用函数	无

```
typedef struct
{
    uint32_t I2C_ClockSpeed;
    uint16_t I2C_DeviveMode;
    uint16_t I2C_AddressMode;
    uint16_t I2C_SlaveAddress;
    uint16_t I2C_Ack;
}I2C_InitTypeDef;
```

I2C_ClockSpeed: 设置 I2C 总线上的时钟速率。

I2C_DeviveMode: 设置 I2C 外设当前的运行模式, 错误!未找到引用源。给出了该参数可取的值。

表 6-5 I2C_DeviveMode 值

I2C_DeviveMode	描述
I2C_DeviveMode_Master	主设备
I2C_DeviveMode_Slave	从设备

I2C_AddressMode: 设置 I2C 外设的地址模式, 错误!未找到引用源。给出了该参数可取的值。

表 6-6 I2C_AddressMode 值

I2C_AddressMode	描述
I2C_AddressMode_7BIT	7 位地址模式
I2C_AddressMode_10BIT	10 位地址模式

I2C_SlaveAddress: 指定的 I2C 从机相应的地址, 可以选择设置为 7 位地址还是 10 位地址。

I2C_Ack: 当收到 General all 时, 是否使能或者失能应答信号, 这个值可以设置为 I2C_Ack_Enable 或者 I2C_Ack_Disable。

例:

```
/* Initialize I2C0 */  
I2C_InitTypeDef I2C_InitStructure;  
I2C_InitStructure.I2C_ClockSpeed = 100000;  
I2C_InitStructure.I2C_DeviveMode = I2C_DeviveMode_Master;  
I2C_InitStructure.I2C_AddressMode = I2C_AddressMode_7BIT;  
I2C_InitStructure.I2C_SlaveAddress = 0x50;  
I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;  
I2C_Init(I2C0, &I2C_InitStructure);
```

6.2.3 函数 I2C_StructInit

表 6-7 函数 I2C_StructInit

函数名	I2C_StructInit
函数原型	void I2C_StructInit(I2C_InitTypeDef* I2C_InitStruct)
功能描述	把 I2C_InitStruct 中的每一个参数按缺省值填入
输入参数	I2C_InitStruct: 指向结构 I2C_InitTypeDef 的指针, 包含了外设 I2C 的配置信息
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Initialize I2C */  
I2C_InitTypeDef I2C_InitStruct;  
I2C_StructInit(&I2C_InitStruct);
```

6.2.4 函数 I2C_Cmd

表 6-8 函数 I2C_Cmd

函数名	I2C_Cmd
函数原型	void I2C_Cmd(I2C_TypeDef* I2Cx, FunctionalState NewState)
功能描述	使能或不使能指定的 I2C 通道
输入参数 1	I2Cx: x 可以设置成 0 或者 1, 选择指定的 I2C 外设
输入参数 2	NewState: 外设 I2Cx 的新状态 这个参数可以取: ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Enable I2C0 */  
I2C_Cmd(I2C0, ENABLE);
```

6.2.5 函数 I2C_MasterWrite

表 6.1 函数 I2C_MasterWrite

函数名	I2C_MasterWrite
函数原型	void I2C_MasterWrite(I2C_TypeDef* I2Cx, uint8_t* pBuf, uint8_t len)
功能描述	在主机模式下通过 I2C 外设发送数据
输入参数 1	I2Cx: x 可以设置成 0 或者 1, 选择指定的 I2C 外设
输入参数 2	pBuf: 指向将要发送数据的指针
输入参数 3	len:发送数据的长度
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Send data */
uint8_t writeBuf[16] = {0x00,0x01,0x02,0x03};
I2C_MasterWrite (I2C0, writeBuf, 4);
```

6.2.6 函数 I2C_MasterRead

表 6-9 函数 I2C_MasterRead

函数名	I2C_MasterRead
函数原型	void I2C_MasterRead(I2C_TypeDef* I2Cx, uint8_t* pBuf, uint8_t len)
功能描述	在主机模式下通过 I2C 外设读数据
输入参数 1	I2Cx: x 可以设置成 0 或者 1, 选择指定的 I2C 外设
输入参数 2	pBuf: 指向将要接收数据的指针
输入参数 3	len:接收数据的长度
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Receive data */
uint8_t receiveBuf[16] = {0};
I2C_MasterRead (I2C0, receiveBuf, 4);
```

6.2.7 函数 I2C_RepeatRead

表 6-10 函数 I2C_RepeatRead

函数名	I2C_RepeatRead
-----	----------------

函数原型	<code>uint8_t I2C_RepeatRead(I2C_TypeDef* I2Cx, uint8_t* pWriteBuf, uint8_t Writelen, uint8_t* pReadBuf, uint8_t Readlen)</code>
功能描述	在主机模式下通过 I2C 外设连续发送和接收数据
输入参数 1	I2Cx: x 可以设置成 0 或者 1, 选择指定的 I2C 外设
输入参数 2	pWriteBuf: 指向将要发送数据的指针
输入参数 3	Writelen: 发送数据的长度
输入参数 4	pReadBuf: 指向将要接收数据的指针
输入参数 5	Readlen: 接收数据的长度
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Repeat read data */
uint8_t I2C_WriteBuf[2] = {0x00,0x01};
uint8_t I2C_ReadBuf[4] = {0, 0 , 0, 0};
I2C_RepeatRead(I2C0, I2C_WriteBuf, 2, I2C_ReadBuf, 4);
```

6.2.8 函数 I2C_INTConfig

表 6-11 函数 I2C_INTConfig

函数名	I2C_INTConfig
函数原型	<code>void I2C_INTConfig(I2C_TypeDef* I2Cx, uint16_t I2C_IT, FunctionalState NewState)</code>
功能描述	使能或者失能 I2C 中断
输入参数 1	I2Cx: x 可以设置成 0 或者 1, 选择指定的 I2C 外设
输入参数 2	I2C_IT: 待使能或者失能的 I2C 中断源, 具体参阅 I2C_IT 介绍部分
输入参数 3	NewState:I2C 中断的新状态 这个参数可以取 ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

I2C_IT: 允许的 SPI 中断类型如错误!未找到引用源。所示, 使用操作符 “|” 可以一次选中多个中断作为该参数的值。

表 6-12 I2C_IT 值

I2C_IT	描述
I2C_INT_GEN_CALL	接收到一般呼叫地址并且进行应答时会产生该中断
I2C_INT_START_DET	检测到启动或者重新启动信号时会产生该中断
I2C_INT_STOP_DET	检测到停止信号时会产生该中断
I2C_INT_ACTIVITY	检测到总线处于通信状态时会产生该中断
I2C_INT_RX_DONE	当设备处于从设备模式并且正在发送数据时, 检测到数据传输完成会产生该中断

I2C_INT_TX_ABRT	检测到总线通信异常时会产生该中断
I2C_INT_RD_REQ	当设备处于从设备模式时，检测到主设备发来读取数据请求会产生该中断
I2C_INT_TX_EMPTY	检测到发送缓冲区的数据个数小于设置的发送阈值时会产生该中断
I2C_INT_TX_OVER	检测到发送缓冲区溢出会产生该中断
I2C_INT_RX_FULL	检测到接收缓冲区的数据个数大于设置的接收阈值时会产生该中断
I2C_INT_RX_OVER	检测到接收缓冲区溢出会产生该中断
I2C_INT_RX_UNDER	当接收缓冲区已经为空时，读取接收缓冲区会产生该中断

例：

```
/* Detect stop signal */
I2C_INTConfig(I2C0, I2C_INT_STOP_DET, ENABLE);
```

6.2.9 函数 I2C_ClearINTPendingBit

表 6-13 函数 I2C_ClearINTPendingBit

函数名	I2C_ClearINTPendingBit
函数原型	void I2C_ClearINTPendingBit(I2C_TypeDef* I2Cx, uint16_t I2C_IT)
功能描述	清除 I2C 中断挂起位
输入参数 1	I2Cx: x 可以设置成 0 或者 1，选择指定的 I2C 外设
输入参数 2	I2C_IT: 待检查 I2C 外设中断源，具体参阅错误!未找到引用源。 I2C_IT 部分的介绍
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear stop signal interrupt */
I2C_ClearINTPendingBit(I2C0, I2C_INT_STOP_DET);
```

6.2.10 函数 I2C_SetSlaveAddress

表 6-14 函数 I2C_SetSlaveAddress

函数名	I2C_SetSlaveAddress
函数原型	void I2C_SetSlaveAddress(I2C_TypeDef* I2Cx, uint16_t Address)
功能描述	设置传输的从设备地址
输入参数 1	I2Cx: x 可以设置成 0 或者 1，选择指定的 I2C 外设
输入参数 2	Address: 从设备地址
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Configure new slave address which the communication is require */
```

```
I2C_SetSlaveAddress(I2C0, 0x66);
```

6.2.11 函数 I2C_SendCmd

表 6-15 函数 I2C_SendCmd

函数名	I2C_SendCmd
函数原型	<code>void I2C_SendCmd(I2C_TypeDef *I2Cx, uint16_t command, uint8_t data, uint16_t StopState)</code>
功能描述	主设备模式下通过外设 I2C 发送或者读取命令
输入参数 1	I2Cx: x 可以设置成 0 或者 1, 选择指定的 I2C 外设
输入参数 2	command: 命令类型, 该参数可取值为: I2C_READ_CMD: 数据读取命令 I2C_WRITE_CMD: 数据发送命令
输入参数 3	data: 待发送的数据
输入参数 4	StopState: 是否产生停止信号 可选参数: ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Send one byte and generate a stop signal */  
I2C_SendCmd (I2C0, I2C_WRITE_CMD, 0x66, ENABLE);
```

6.2.12 函数 I2C_ReceiveData

表 6-16 函数 I2C_ReceiveData

函数名	I2C_ReceiveData
函数原型	<code>uint8_t I2C_ReceiveData(I2C_TypeDef *I2Cx)</code>
功能描述	通过外设 I2C 接收一个数据
输入参数 1	I2Cx: x 可以设置成 0 或者 1, 选择指定的 I2C 外设
输出参数	无
返回值	作为从设备接收到 I2C 总线上的数据
先决条件	无
被调用函数	无

例:

```
/* Receive one byte */  
uint8_t data = I2C_ReceiveData (I2C0);
```

6.2.13 函数 I2C_GetRxFIFOLen

表 6-17 函数 I2C_GetRxFIFOLen

函数名	I2C_GetRxFIFOLen
函数原型	uint8_t I2C_GetRxFIFOLen(I2C_TypeDef *I2Cx)
功能描述	从 I2C 接收 FIFO 中读取接收数据的长度
输入参数	I2Cx: x 可以设置成 0 或者 1, 选择指定的 I2C 外设
输出参数	无
返回值	数据长度
先决条件	无
被调用函数	无

例：

```
/* Get data number in RX FIFO */  
uint8_t  number = 0;  
number = I2C_GetRxFIFOLen (I2C0);
```

6.2.14 函数 I2C_GetTxFIFOLen

表 6-18 函数 I2C_GetTxFIFOLen

函数名	I2C_GetTxFIFOLen
函数原型	uint8_t I2C_GetTxFIFOLen(I2C_TypeDef *I2Cx)
功能描述	从 I2C 发送 FIFO 中读取发送数据的长度
输入参数 1	I2Cx: x 可以设置成 0 或者 1, 选择指定的 I2C 外设
输出参数	无
返回值	数据长度
先决条件	无
被调用函数	无

例：

```
/* Get data number in Tx FIFO */  
uint8_t  number = 0;  
number = I2C_GetTxFIFOLen (I2C0);
```

6.2.15 函数 I2C_ClearAllINT

表 6-19 函数 I2C_ClearAllINT

函数名	I2C_ClearAllINT
函数原型	void I2C_ClearAllINT(I2C_TypeDef* I2Cx)
功能描述	清除 I2C 所有挂起的中断
输入参数 1	I2Cx: x 可以设置成 0 或者 1, 选择指定的 I2C 外设
输出参数	无

返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear all interrupt state */
I2C_ClearAllINT(I2C0);
```

6.2.16 函数 I2C_GetFlagState

表 6-20 函数 I2C_GetFlagState

函数名	I2C_GetFlagState
函数原型	FlagStatus I2C_GetFlagState(I2C_TypeDef* I2Cx, uint32_t I2C_FLAG)
功能描述	检查 I2C 指定的标志位是否被设置
输入参数 1	I2Cx: x 可以设置成 0 或者 1, 选择指定的 I2C 外设
输入参数 2	I2C_FLAG: 待检测的标志位
输出参数	无
返回值	I2C 指定标志的新状态 (SET 或者 RESET)
先决条件	无
被调用函数	无

I2C_FLAG: 允许的 I2C 状态标志如表 6. 2 所示。

表 6. 2 I2C_FLAG 值

I2C_FLAG	描述
I2C_FLAG_SLV_ACTIVITY	当设备处于从设备模式并且总线处于通信状态
I2C_FLAG_MST_ACTIVITY	当设备处于主设备模式并且总线处于通信状态
I2C_FLAG_RFF	接收缓冲区已满
I2C_FLAG_RFNE	接收缓冲区不为空
I2C_FLAG_TFE	发送缓冲区为空
I2C_FLAG_TFNF	发送缓冲区没有满
I2C_FLAG_ACTIVITY	总线状态

例：

```
/* Wait communication end as a master . If SET, I2C0 have not finish transmission */
while(I2C_GetFlagState(I2C0, I2C_FLAG_MST_ACTIVITY) == 1);
```

6.2.17 函数 I2C_CheckEvent

表 6-21 函数 I2C_CheckEvent

函数名	I2C_CheckEvent
函数原型	FlagStatus I2C_CheckEvent(I2C_TypeDef* I2Cx, uint32_t I2C_EVENT)
功能描述	检查 I2C 指定的传输失败事件状态
输入参数 1	I2Cx: x 可以设置成 0 或者 1, 选择指定的 I2C 外设

输入参数 2	I2C_EVENT: 待检测的 I2C 传输事件标志, 具体参阅错误!未找到引用源。I2C_EVENT 介绍部分
输出参数	无
返回值	I2C 指定异常事件的新状态(SET 或者 RESET)
先决条件	无
被调用函数	无

I2C_EVENT: 允许的 I2C 事件标志如错误!未找到引用源。所示。

表 6-22 I2C_EVENT 值

I2C_EVENT	描述
ABRT_SLRD_INTX	当设备处于从设备模式且收到主设备的读取请求时, 发送读取命令会触发该异常
ABRT_SLV_ARBLOST	当设备处于从设备模式且正在发送数据时, 总线触发该异常
ABRT_SLVFLUSH_TXFIFO	当设备处于从设备模式且收到主设备的读取请求时, 如果发送缓冲区有数据会导致原数据被覆盖, 从而触发该异常
ARB_LOST	设备失去仲裁权会触发该异常
ABRT_MASTER_DIS	当设备的主设备模式被禁用时, 启动主设备模式操作会触发该异常
ABRT_10B_RD_NORSTR	当设备处于主设备模式并且重启动功能被禁用时, 以 10bit 寻址模式发送读取命令时会触发该异常
ABRT_SBYTE_NORSTR	当设备处于主设备模式且重启动功能被禁用时, 发送启动命令会触发该异常
ABRT_HS_NORSTR	当设备处于主设备模式且重启动功能被禁用时, 在高速模式(总线速率大于 400K) 传输数据会触发该异常
ABRT_SBYTE_ACKDET	当设备处于主设备模式时, 发送启动命令后收到应答信号会触发该异常
ABRT_HS_ACKDET	当设备处于主设备模式且为高速模式(总线速率)时, 在高速模式(总线速率大于 400K) 传输数据会触发该异常
ABRT_GCALL_READ	当设备处于主设备模式且发送一般呼叫命令后, 发送读取命令会触发该异常
ABRT_GCALL_NOACK	当设备处于主设备模式且发送一般呼叫命令后, 未收到应答信号会触发该异常
ABRT_TXDATA_NOACK	当设备处于主设备模式时, 发送数据后未收到应答信号会触发该异常
ABRT_10ADDR2_NOACK	当设备处于主设备模式且为 10 bit 寻址时, 发送第二个地址数据后未收到应答信号会触发该异常
ABRT_10ADDR1_NOACK	当设备处于主设备模式且为 10 bit 寻址时, 发送第一个地址数据后未收到应答信号会触发该异常
ABRT_7B_ADDR_NOACK	当设备处于主设备模式且为 7 bit 寻址时, 发送地址数据后未收到应答信号会触发该异常

例:

```
/* Check I2C receive acknowledgement or not after 7-bit address command to be sent */
bool flag_state = I2C_CheckEvent(I2C0, ABRT_7B_ADDR_NOACK);
```

6.2.18 函数 I2C_GetINTStatus

表 6-23 函数 I2C_GetINTStatus

函数名	I2C_GetINTStatus
函数原型	ITStatus I2C_GetINTStatus(I2C_TypeDef* I2Cx, uint32_t I2C_IT)

功能描述	获得指定的 I2C 中断状态
输入参数 1	I2Cx: x 可以设置成 0 或者 1, 选择指定的 I2C 外设
输入参数 2	I2C_IT: 待获得状态的中断源, 具体参阅错误!未找到引用源。 I2C_IT 介绍部分
输出参数	无
返回值	I2C 指定中断的新状态 (SET 或者 RESET)
先决条件	无
被调用函数	无

例:

```
/* Check stop singal interrupt state */
Bool int_state = I2C_GetINTStatus(I2C0, I2C_INT_STOP_DET);
```

6.2.19 函数 I2C_GDMACmd

表 6-24 函数 I2C_GDMACmd

函数名	I2C_GDMACmd
函数原型	void I2C_GDMACmd(I2C_TypeDef *I2Cx, uint16_t I2C_GDMAReq, FunctionalState NewState)
功能描述	使能或者失能 GDMA 指定类型的数据传输功能
输入参数 1	I2Cx: x 可以设置成 0 或者 1, 选择指定的 I2C 外设
输入参数 2	GDMAReq: 数据传输类型, 该参数可取: I2C_GDMAReq_Tx: GDMA 数据发送功能 I2C_GDMAReq_Rx: GDMA 数据接收功能
输入参数 3	NewState:I2C GDMA 数据传输的新状态 这个参数可以取 ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Enable I2C GDMA data transmission function */
I2C_GDMACmd (I2C0, I2C_GDMAReq_Tx, ENABLE);
```

7 正交解调(QDEC)

7.1 QDEC 寄存器架构

```
typedef struct {
    __IO uint32_t    REG_DIV;
    __IO uint32_t    REG_CR_X;
    __IO uint32_t    REG_SR_X;
    __IO uint32_t    REG_CR_Y;
    __IO uint32_t    REG_SR_Y;
    __IO uint32_t    REG_CR_Z;
    __IO uint32_t    REG_SR_Z;
    __IO uint32_t    INT_MASK;
    __IO uint32_t    INT_SR;
    __IO uint32_t    INT_CLR;
    __IO uint32_t    REG_DBG;
    __IO uint32_t    REG_VERSION;
} QDEC_TypeDef;
```

错误!未找到引用源。例举 QDEC 所有寄存器

表 7-1 QDEC 寄存器

寄存器	描述
REG_DIV	分频寄存器
REG_CR_X	X 轴控制寄存器
REG_SR_X	X 轴状态寄存器
REG_CR_Y	Y 轴控制寄存器
REG_SR_Y	Y 轴状态寄存器
REG_CR_Z	Z 轴控制寄存器
REG_SR_Z	Z 轴状态寄存器
INT_MASK	中断屏蔽寄存器
INT_SR	中断状态寄存器
INT_CLR	中断清楚寄存器
REG_DBG	调试寄存器
REG_VERSION	版本寄存器

7.2 QDEC 库函数

表 7-2 例举 QDEC 的所有库函数

函数名	描述

QDEC_Init	根据 QDEC_InitStruct 中指定参数初始化 QDEC 寄存器
QDEC_DelInit	关闭 QDEC 外设时钟
QDEC_StructInit	把 QDEC_InitStruct 中的每一个值按照缺省值填入
QDEC_INTConfig	使能或者失能 QDEC 指定的中断源
QDEC_GetFlagState	检查指定的 QDEC 标志是否被置位
QDEC_INTMask	屏蔽指定的 QDEC 中断源
QDEC_Cmd	使能或失能 QDEC 指定通道
QDEC_ClearINTPendingBit	清除 QDEC 外设挂起的中断标志位
QDEC.GetAxisDirection	获得指定轴的运动方向
QDEC.GetAxisCount	获取指定 x 或 y 或 z 轴的加速值
QDEC_CounterPauseCmd	暂停指定 x 或 y 或 z 轴

7.2.1 函数 QDEC_Init

表 7-3 函数 QDEC_Init

函数名	QDEC_Init
函数原型	void QDEC_Init(QDEC_TypeDef* QDECx, QDEC_InitTypeDef* QDEC_InitStruct)
功能描述	根据 QDEC_InitStruct 中指定参数初始化 QDEC 寄存器
输入参数 1	QDEC: 指向选择的 QDEC 模块
输入参数 2	QDEC_InitStruct: 指向 QDEC_InitTypeDef 的指针, QDEC_InitStruct 中包含相关配置信息
输出参数	无
返回值	无
先决条件	无
被调用函数	无

```
typedef struct
{
    uint16_t scanClockDiv;
    uint16_t debounceClockDiv;
    uint8_t axisConfigX;
    uint8_t axisConfigY;
    uint8_t axisConfigZ;
    uint8_t autoLoadInitPhase;
    uint16_t counterScaleX;
    uint16_t debounceEnableX;
    uint16_t debounceTimeX;
    uint16_t initPhaseX;
    uint16_t counterScaleY;
    uint16_t debounceEnableY;
    uint16_t debounceTimeY;
    uint16_t initPhaseY;
    uint16_t counterScaleZ;
```

```

    uint16_t debounceEnableZ;
    uint16_t debounceTimeZ;
    uint16_t initPhaseZ;
} QDEC_InitTypeDef;

```

scanClockDiv: 设置 QDEC 外设的扫描时钟分频系数，该参数的取值范围为 0 到 4095。

扫描时钟=外设时钟/(扫描分频系数+1)，其中外设时钟为 20MHz。

debounceClockDiv: 设置 QDEC 外设的去抖动时钟分频系数，该参数的取值范围为 0 到 15。

去抖动时钟=扫描时钟/(分频系数+1)。

axisConfigX: 使能或者失能 X 轴的功能。该参数可选择 ENABLE 或者 DISABLE。

debounceTimeX: 设置 QDEC 外设 X 轴的消抖时间，该参数的取值范围为 0 到 255。

counterScaleX: 设置 X 轴的计数器计数模式。错误!未找到引用源。给出了该参数可取的值。

表 7-4 counterScaleX 值

counterScaleX	描述
CounterScale_1_Phase	每发生一个相位变化，计数器进行更新
CounterScale_2_Phase	每发生一个相位变化，计数器进行更新

debounceEnableX: 使能或者失能去抖动功能。错误!未找到引用源。给出了该参数可取的值。

表 7-5 debounceEnableX 值

debounceEnableX	描述
Debounce_Disable	失能去抖动功能
Debounce_Enable	使能去抖动功能

initPhaseX: 设置 X 轴的初始相位值。错误!未找到引用源。给出了该参数可取的值。

表 7-6 initPhaseX 值

initPhaseX	描述
phaseMode0	初始相位为 0
phaseMode1	初始相位为 1
phaseMode2	初始相位为 2
phaseMode3	初始相位为 3

axisConfigY: 使能或者失能 Y 轴的功能。该参数可选择 ENABLE 或者 DISABLE。

debounceTimeY: 设置 QDEC 外设 Y 轴的消抖时间，该参数的取值范围为 0 到 255。

counterScaleY: 设置 Y 轴的计数器计数模式。错误!未找到引用源。给出了该参数可取的值。

表 7-7 counterScaleY 值

counterScaleY	描述
CounterScale_1_Phase	每发生一个相位变化，计数器进行更新
CounterScale_2_Phase	每发生一个相位变化，计数器进行更新

debounceEnableY: 使能或者失能去抖动功能。错误!未找到引用源。给出了该参数可取的值。

表 7-8 debounceEnableY 值

debounceEnableY	描述
Debounce_Disable	失能去抖动功能

Debounce_Enable	使能去抖动功能
-----------------	---------

initPhaseY: 设置 Y 轴的初始相位值。错误!未找到引用源。给出了该参数可取的值。

表 7-9 initPhaseY 值

initPhaseY	描述
phaseMode0	初始相位为 0
phaseMode1	初始相位为 1
phaseMode2	初始相位为 2
phaseMode3	初始相位为 3

axisConfigZ: 使能或者失能 Z 轴的功能。该参数可选择 ENABLE 或者 DISABLE。

debounceTimeZ: 设置 QDEC 外设 Z 轴的消抖时间，该参数的取值范围为 0 到 255。

counterScaleZ: 设置 Z 轴的计数器计数模式。错误!未找到引用源。给出了该参数可取的值。

表 7-10 counterScaleZ 值

counterScaleZ	描述
CounterScale_1_Phase	每发生一个相位变化，计数器进行更新
CounterScale_2_Phase	每发生一个相位变化，计数器进行更新

debounceEnableZ: 使能或者失能去抖动功能。错误!未找到引用源。给出了该参数可取的值。

表 7-11 debounceEnableZ 值

debounceEnableZ	描述
Debounce_Disable	失能去抖动功能
Debounce_Enable	使能去抖动功能

initPhaseZ: 设置 Z 轴的初始相位值。错误!未找到引用源。给出了该参数可取的值。

表 7-12 initPhaseZ 值

initPhaseZ	描述
phaseMode0	初始相位为 0
phaseMode1	初始相位为 1
phaseMode2	初始相位为 2
phaseMode3	初始相位为 3

例：

```
/*Initialize qdecoder */
QDEC_InitTypeDef qdecInitStruct;
qdecInitStruct.axisConfigY      = ENABLE;
qdecInitStruct.scanClockDiv    = 38;
qdecInitStruct.debounceClockDiv = 0xF;
qdecInitStruct.debounceTimeY   = 80;
qdecInitStruct.counterScaleY  = counterScaleDisable;
qdecInitStruct.debounceEnableY = Debounce_Enable;
qdecInitStruct.initPhaseY     = phase;
QDEC_Init(QDEC, &qdecInitStruct);
```

7.2.2 函数 QDEC_DeInit

表 7-13 函数 QDEC_DeInit

函数名	QDEC_DeInit
函数原型	void QDEC_DeInit(QDEC_TypeDef* QDECx)
功能描述	关闭 QDEC 外设时钟
输入参数	QDEC: 指向选择的 QDEC 模块
输出参数	无
返回值	无
先决条件	无
被调用函数	RCC_PeriphClockCmd()

例：

```
/* Close qdecoder clock */
QDEC_DeInit(QDEC);
```

7.2.3 函数 QDEC_StructInit

表 7-14 函数 QDEC_StructInit

函数名	QDEC_StructInit
函数原型	void QDEC_StructInit(QDEC_InitTypeDef* QDEC_InitStruct)
功能描述	把 QDEC_InitStruct 中的每一个值按照缺省值填入
输入参数 1	QDEC_InitStruct: 指向结构体 QDEC_InitTypeDef 的指针, 待初始化
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Initialize qdecoder by default parameter */
QDEC_InitTypeDef  QDEC_InitStruct;
QDEC_StructInit(&QDEC_InitStruct);
```

7.2.4 函数 QDEC_INTConfig

表 7-15 函数 QDEC_INTConfig

函数名	QDEC_INTConfig
函数原型	void QDEC_INTConfig(QDEC_TypeDef* QDECx, uint32_t QDEC_IT, FunctionalState newState)
功能描述	使能或者失能 QDEC 指定的中断源
输入参数 1	QDECx: 指向选择的 QDEC 模块
输入参数 2	QDEC_IT:待使能或者失能的 QDEC 中断源, 具体参阅 QDEC_IT 介绍部分

输入参数 3	NewState:中断的新状态 这个参数可以去 ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

QDEC_IT: 允许的 QDEC 中断类型如错误!未找到引用源。所示, 使用操作符 “|” 可以一次选中多个中断作为该参数的值。

表 7-16QDEC_IT 值

QDEC_IT	描述
QDEC_X_INT_NEW_DATA	X 轴有新的数据产生就发起中断
QDEC_X_INT_ILLEGAL	X 轴相位发生异常时发起中断
QDEC_Y_INT_NEW_DATA	Y 轴有新的数据产生就发起中断
QDEC_Y_INT_ILLEGAL	Y 轴相位发生异常时发起中断
QDEC_Z_INT_NEW_DATA	Z 轴有新的数据产生就发起中断
QDEC_Z_INT_ILLEGAL	Z 轴相位发生异常时发起中断

例:

```
/* Enable qdecoder specify interrupt */
QDEC_INTConfig(QDEC, QDEC_X_INT_NEW_DATA, ENABLE);
```

7.2.5 函数 QDEC_GetFlagState

表 7-17 函数 QDEC_GetFlagState

函数名	QDEC_GetFlagState
函数原型	FlagStatus QDEC_GetFlagState(QDEC_TypeDef* QDECx, uint32_t QDEC_FLAG)
功能描述	检查指定的 QDEC 标志是否被置位
输入参数 1	QDECx: 指向选择的 QDEC 模块
输入参数 2	QDEC_FLAG:待检测的状态标志
输出参数	无
返回值	指定 QDEC 的状态, SET 或 RESET
先决条件	无
被调用函数	无

QDEC_FLAG: 允许的 QDEC 状态类型如错误!未找到引用源。所示。

表 7-18 QDEC_FLAG 值

QDEC_IT	描述
QDEC_FLAG_ILLEGAL_STATUS_X	X 轴出现相位变化异常时, 该状态置位
QDEC_FLAG_ILLEGAL_STATUS_Y	Y 轴出现相位变化异常时, 该状态置位
QDEC_FLAG_ILLEGAL_STATUS_Z	Z 轴出现相位变化异常时, 该状态置位
QDEC_FLAG_NEW_STATUS_X	X 轴有新的数据时, 该状态置位
QDEC_FLAG_NEW_STATUS_Y	Y 轴有新的数据时, 该状态置位
QDEC_FLAG_NEW_STATUS_Z	Z 轴有新的数据时, 该状态置位

例：

```
/* Check specify QDEC flag */
BOOL state = FALSE;
state = QDEC_GetFlagState(QDEC, QDEC_FLAG_NEW_STATUS_X);
```

7.2.6 函数 QDEC_INTMask

表 7-19 函数 QDEC_INTMask

函数名	QDEC_INTMask
函数原型	void QDEC_INTMask(QDEC_TypeDef* QDECx, uint32_t QDEC_AXIS, FunctionalState newState)
功能描述	屏蔽指定的 QDEC 中断源
输入参数 1	QDECx: 指向选择的 QDEC 模块
输入参数 2	QDEC_AXIS: 待屏蔽的中断源
输入参数 3	NewState: 外设 QDECx 指定通道的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

QDEC_AXIS: 允许屏蔽的 QDEC 中断类型如错误!未找到引用源。所示。

表 7-20 QDEC_AXIS 值

QDEC_AXIS	描述
QDEC_X_CT_INT_MASK	X 轴产生新数据中断
QDEC_Y_CT_INT_MASK	Y 轴产生新数据中断
QDEC_Z_CT_INT_MASK	Z 轴产生新数据中断
QDEC_X_ILLEGAL_INT_MASK	X 轴出现相位变化异常中断
QDEC_Y_ILLEGAL_INT_MASK	Y 轴出现相位变化异常中断
QDEC_Z_ILLEGAL_INT_MASK	Z 轴出现相位变化异常中断

例：

```
/* Mask X axis of qdecoder */
QDEC_INTMask (QDEC, QDEC_X_CT_INT_MASK, ENABLE);
```

7.2.7 函数 QDEC_Cmd

表 7-21 函数 QDEC_Cmd

函数名	QDEC_Cmd
函数原型	void QDEC_Cmd(QDEC_TypeDef *QDECx, uint32_t QDEC_AXIS, FunctionalState newState)
功能描述	使能或失能 QDEC 指定通道
输入参数 1	QDECx: x 可以设置成 0 或者 1, 选择指定的 QDEC 外设

输入参数 2	QDEC_AXIS: QDEC 指定通道, 该参数可取: QDEC_AXIS_X: X 轴通道 QDEC_AXIS_Y: Y 轴通道 QDEC_AXIS_Z: Z 轴通道
输入参数 3	NewState: 外设 QDECx 指定通道的新状态 这个参数可以取: ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Enable QDEC */
QDEC_Cmd(QDEC, QDEC_AXIS_X, ENABLE);
```

7.2.8 函数 QDEC_ClearINTPendingBit

表 7-22 函数 QDEC_ClearINTPendingBit

函数名	QDEC_ClearINTPendingBit
函数原型	void QDEC_ClearINTPendingBit(QDEC_TypeDef* QDECx, uint32_t QDEC_CLR_INT);
功能描述	清除 QDEC 外设挂起的中断标志位
输入参数 1	QDECx: 指向选择的 QDEC 模块
输入参数 2	QDEC_CLR_INT: 待清除的中断
输出参数	无
返回值	无
先决条件	无
被调用函数	无

QDEC_INT_FLAG: 允许清除的 QDEC 中断类型如错误!未找到引用源。所示。

表 7-23 QDEC_INT_FLAG 值

QDEC_INT_FLAG	描述
QDEC_CLR_ILLEGAL_CT_X	清除 X 轴相位非法计数
QDEC_CLR_ILLEGAL_CT_Y	清除 Y 轴相位非法计数
QDEC_CLR_ILLEGAL_CT_Z	清除 Z 轴相位非法计数
QDEC_CLR_ACC_CT_X	清除 X 轴相位累加计数
QDEC_CLR_ACC_CT_Y	清除 Y 轴相位累加计数
QDEC_CLR_ACC_CT_Z	清除 Z 轴相位累加计数
QDEC_CLR_ILLEGAL_INT_X	清除 X 轴相位变化异常计数中断
QDEC_CLR_ILLEGAL_INT_Y	清除 Y 轴相位变化异常计数中断
QDEC_CLR_ILLEGAL_INT_Z	清除 Z 轴相位变化异常计数中断
QDEC_CLR_UNDERFLOW_X	清除 X 轴计数器下溢中断
QDEC_CLR_UNDERFLOW_Y	清除 Y 轴计数器下溢中断

QDEC_CLR_UNDERFLOW_Z	清除 Z 轴计数器下溢中断
QDEC_CLR_OVERFLOW_X	清除 X 轴计数器溢出中断
QDEC_CLR_OVERFLOW_Y	清除 Y 轴计数器溢出中断
QDEC_CLR_OVERFLOW_Z	清除 Z 轴计数器溢出中断
QDEC_CLR_NEW_CT_X	清除 X 轴计数器产生新数据中断
QDEC_CLR_NEW_CT_Y	清除 Y 轴计数器产生新数据中断
QDEC_CLR_NEW_CT_Z	清除 Z 轴计数器产生新数据中断

例：

```
/* Clear specify QDEC interrupt */
QDEC_ClearINTPendingBit(QDEC, QDEC_CLR_NEW_CT_X);
```

7.2.9 函数 QDEC_GetAxisDirection

表 7-24 函数 QDEC_GetAxisDirection

函数名	QDEC_GetAxisDirection
函数原型	uint16_t QDEC_GetAxisDirection(QDEC_TypeDef* QDECx, uint32_t QDEC_AXIS)
功能描述	获得指定轴的运动方向
输入参数 1	QDECx: 指向选择的 QDEC 模块
输入参数 2	QDEC_AXIS: 指定的 x 或 y 或 z 轴, 具体参阅 7.2.4 节 QDEC_AXIS 介绍部分
输出参数	无
返回值	运动方向: 返回值如下 QDEC_AXIS_DIR_UP: 向上滚动 QDEC_AXIS_DIR_DOWN: 向下滚动
先决条件	无
被调用函数	无

例：

```
/* Wait for moving upward */
while(QDEC_AXIS_DIR_UP == QDEC_GetAxisDirection(QDEC, QDEC_AXIS_X));
```

7.2.10 函数 QDEC_GetAxisCount

表 7-25 函数 QDEC_GetAxisCount

函数名	QDEC_GetAxisCount
函数原型	uint16_t QDEC_GetAxisCount (QDEC_TypeDef* QDECx, uint32_t QDEC_AXIS)
功能描述	获取指定 x 或 y 或 z 轴的加速值
输入参数 1	QDECx: 指向选择的 QDEC 模块
输入参数 2	QDEC_AXIS: 指定的 x 或 y 或 z 轴, 具体参阅 7.2.4 节 QDEC_AXIS 介绍部分
输出参数	无
返回值	指定轴的计数值
先决条件	无

被调用函数	无
-------	---

例：

```
/* Get direction data of X axis */  
uitn16_t value = 0;  
value = QDEC_GetAxisCount (QDEC, QDEC_AXIS_X);
```

7.2.11 函数 QDEC_CounterPauseCmd

表 7-26 函数 QDEC_CounterPauseCmd

函数名	QDEC_CounterPauseCmd
函数原型	void QDEC_CounterPauseCmd(QDEC_TypeDef *QDECx, uint32_t QDEC_AXIS, FunctionalState newState)
功能描述	暂停指定 x 或 y 或 z 轴
输入参数 1	QDECx: 指向选择的 QDEC 模块
输入参数 2	QDEC_AXIS:指定的 x 或 y 或 z 轴 取值范围: QDEC_AXIS_X QDEC_AXIS_Y QDEC_AXIS_Z
输入参数 3	NewState: 外设 QDECx 指定通道的新状态 取值范围: ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Pause specify axis */  
QDEC_CounterPauseCmd (QDEC, QDEC_AXIS_X, ENABLE);
```

8 管脚分配定义(PAD)

8.1 PAD 库函数

表 8-1 例举 PAD 所有库函数

函数名	描述
Pad_Config	配置指定管脚的相关运行模式、外围电路以及在软件模式下的输出电平值
Pad_OutputControlValue	配置 PAD 输出电平
Pad_OutputEnableValue	使能或失能 PAD 输出模式
Pad_PullEnableValue	使能或失能 PAD 上拉/下拉电阻功能
Pad_PullUpOrDownValue	配置 PAD 上拉或下拉电阻
Pad_PullConfigValue	配置 PAD 上下拉电阻强弱
Pad_WakeupEnableValue	使能或失能 PAD 唤醒功能
Pad_WakeupPolarityValue	配置 PAD 唤醒极性
Pad_PowerOrShutDownValue	配置 PAD 供电模式
Pad_ControlSelectValue	配置 PAD 模式
Pad_WKDebounceConfig	使能或失能 PAD 唤醒延时功能
Pad_ClearWakeupINTPendingBit	清除 PAD 唤醒中断标志位
Pad_WakeupInterruptValue	检测 PAD 唤醒中断状态
System_WakeUpInterruptValue	检测系统唤醒中断状态
System_WakeUpDebounceTime	配置指定管脚的唤醒延时时间
System_WakeUpPinEnable	使能指定管脚唤醒系统的功能
System_WakeUpPinDisable	失能指定管脚唤醒系统的功能

8.1.1 函数 Pad_Config

表 8-2 函数 Pad_Config

函数名	Pad_Config
函数原型	void Pad_Config(uint8_t Pin_Num, BOOL AON_PAD_Mode, BOOL AON_PAD_PwrOn, BOOL AON_PAD_Pull, BOOL AON_PAD_E, BOOL AON_PAD_O)
功能描述	配置指定管脚的相关运行模式、外围电路以及在软件模式下的输出电平值
输入参数 1	Pin_Num: 所需配置的管脚值, 具体参阅 Pin_Num 介绍部分
输入参数 2	AON_PAD_Mode: 该参数设置管脚的模式, 该值为枚举 PAD_Mode 的其中一个值 PAD_SW_MODE: 软件模式 PAD_PINMUX_MODE: pinmux 模式
输入参数 3	AON_PAD_PwrOn: 该参数设置管脚的电源模式, 该值为枚举 PAD_PWR_Mode 的其中一个值 PAD_NOT_PWRON: 关闭电源 PAD_IS_PWRON: 打开电源

输入参数 4	AON_PAD_Pull: 该参数设置管脚的外围电路, 该值为枚举 PAD_Pull_Mode 的其中一个值 PAD_PULL_NONE: 没有上拉或者下拉电阻 PAD_PULL_UP: 内部连接上拉电阻 PAD_PULL_DOWN: 内部连接下拉电阻
输入参数 5	AON_PAD_E: 该参数设置管脚在软件模式下是否输出电平, 该值为枚举 PAD_OUTPUT_ENABLE_Mode 的其中一个值 PAD_OUT_DISABLE: 失能输出 PAD_OUT_ENABLE: 使能输出
输入参数 6	AON_PAD_O: 该参数设置管脚在软件模式下输出的电平值, 该值为枚举 PAD_OUTPUT_VAL 的其中一个值 PAD_OUT_LOW: 输出低电平 PAD_OUT_HIGH: 输出高电平
输出参数	无
返回值	无
先决条件	无
被调用函数	无

Pin_Num: 所有可选管脚值如错误!未找到引用源。所示, 具体可选管脚请参阅具体芯片型号手册。

表 8-3Pin_Num 值

Pin_Num	描述
P0_0	选中管脚 0
P0_1	选中管脚 1
P0_2	选中管脚 2
P0_3	选中管脚 3
P0_4	选中管脚 4
P0_5	选中管脚 5
P0_6	选中管脚 6
P0_7	选中管脚 7
P1_0	选中管脚 8
P1_1	选中管脚 9
P1_2	选中管脚 10
P1_3	选中管脚 11
P1_4	选中管脚 12
P1_5	选中管脚 13
P1_6	选中管脚 14
P1_7	选中管脚 15
P2_0	选中管脚 16
P2_1	选中管脚 17
P2_2	选中管脚 18
P2_3	选中管脚 19
P2_4	选中管脚 20
P2_5	选中管脚 21

P2_6	选中管脚 22(MIC_N)
P2_7	选中管脚 23(MIC_P)
P3_0	选中管脚 24
P3_1	选中管脚 25
P3_2	选中管脚 26
P3_3	选中管脚 27
P3_4	选中管脚 28
P3_5	选中管脚 29
P3_6	选中管脚 30
P4_0	选中管脚 32
P4_1	选中管脚 33
P4_2	选中管脚 34
P4_3	选中管脚 35
H_0	选中管脚 36(MICBIAS)
H_1	选中管脚 37(32K_XI)
H_2	选中管脚 38(32K_XO)

例：

```
/* Configure P0_5 for specified mode */
Pad_Config(P0_5,PAD_PINMUX_MODE,PAD_IS_PWRON,PAD_PULL_NONE, PAD_OUT_ENABLE, PAD_OUT_HIGH);
```

8.1.2 函数 Pad_OutputControlValue

表 8-4 函数 Pad_OutputControlValue

函数名	Pad_OutputControlValue
函数原型	void Pad_OutputControlValue (uint8_t Pin_Num, uint8_t value)
功能描述	配置 PAD 输出电平
输入参数 1	Pin_Num：所需配置的管脚值，具体参阅 Pin_Num 介绍部分
输入参数 2	AON_PAD_O：该参数设置管脚在软件模式下输出的电平值，该值为枚举 PAD_OUTPUT_VAL 的其中一个值 PAD_OUT_LOW：输出低电平 PAD_OUT_HIGH：输出高电平
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Configure P0_5 output value */
Pad_OutputControlValue(P0_5, PAD_OUT_HIGH);
```

8.1.3 函数 Pad_OutputEnableValue

表 8-5 函数 Pad_OutputEnableValue

函数名	Pad_OutputEnableValue
函数原型	void Pad_OutputEnableValue(uint8_t Pin_Num, uint8_t value)
功能描述	使能或失能 PAD 输出模式
输入参数 1	Pin_Num: 所需配置的管脚值, 具体参阅 Pin_Num 介绍部分
输入参数 2	AON_PAD_E: 该参数设置管脚在软件模式下是否输出电平, 该值为枚举 PAD_OUTPUT_ENABLE_Mode 的其中一个值 PAD_OUT_DISABLE: 失能输出 PAD_OUT_ENABLE: 使能输出
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/*Enable P0_5 output */
Pad_OutputEnableValue (P0_5, PAD_OUT_ENABLE);
```

8.1.4 函数 Pad_PullEnableValue

表 8-6 函数 Pad_PullEnableValue

函数名	Pad_PullEnableValue
函数原型	void Pad_PullEnableValue (uint8_t Pin_Num, uint8_t value)
功能描述	使能或失能 PAD 上拉/下拉电阻功能
输入参数 1	Pin_Num: 所需配置的管脚值, 具体参阅 Pin_Num 介绍部分
输入参数 2	value: 该参数设置 PAD 上拉/下拉电阻功能, 该值可选参数为 ENABLE: 使能 PAD 上拉/下拉功能 DISABLE: 失能 PAD 上拉/下拉功能
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/*Enable P0_5 pull function*/
Pad_PullEnableValue(P0_5, ENABLE);
```

8.1.5 函数 Pad_PullUpOrDownValue

表 8-7 函数 Pad_PullUpOrDownValue

函数名	Pad_PullUpOrDownValue
函数原型	void Pad_PullUpOrDownValue (uint8_t Pin_Num, uint8_t value)
功能描述	PAD 上拉/下拉电阻功能选择
输入参数 1	Pin_Num: 所需配置的管脚值, 具体参阅 Pin_Num 介绍部分
输入参数 2	value: 该参数设置 PAD 上拉/下拉电阻功能, 该值可选参数为 true: 配置 PAD 上拉功能 false: 配置 PAD 下拉功能
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/*Config P0_5 pull up or down*/
Pad_PullUpOrDownValue(P0_5, true);
```

8.1.6 函数 Pad_PullConfigValue

表 8-8 函数 Pad_PullConfigValue

函数名	Pad_PullConfigValue
函数原型	void Pad_PullConfigValue (uint8_t Pin_Num, uint8_t value)
功能描述	配置 PAD 上下拉电阻强弱
输入参数 1	Pin_Num: 所需配置的管脚值, 具体参阅 Pin_Num 介绍部分
输入参数 2	value: 该参数设置管脚的模式, 该值为枚举 PAD_PULL_VAL 的其中一个值。如果调用该接口进行配置, 默认为电阻弱拉。 PAD_WEAK_PULL: 电阻弱拉 PAD_STRONG_PULL: 电阻强拉
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/*Config P0_2 pull weakly*/
Pad_WakeupInterruptValue (P0_2, PAD_WEAK_PULL);
```

8.1.7 函数 Pad_WakeupEnableValue

表 8-9 函数 Pad_WakeupEnableValue

函数名	Pad_WakeupEnableValue
函数原型	void Pad_WakeupEnableValue(uint8_t Pin_Num, uint8_t value)
功能描述	使能参数设置管脚的唤醒功能
输入参数 1	Pin_Num: 所需配置的管脚值, 具体参阅 Pin_Num 介绍部分
输入参数 2	value: 使能或失能 1: 使能 0: 失能
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Enable P0_2 wake up function*/
Pad_WakeupEnableValue (P0_2, 1);
```

8.1.8 函数 Pad_WakeupPolarityValue

表 8-10 函数 Pad_WakeupPolarityValue

函数名	Pad_WakeupPolarityValue
函数原型	void Pad_WakeupPolarityValue(uint8_t Pin_Num, uint8_t value)
功能描述	配置该管脚的唤醒极性
输入参数 1	Pin_Num: 所需配置的管脚值, 具体参阅 Pin_Num 介绍部分。
输入参数 2	value: 该参数设置管脚的唤醒极性, 该值为枚举_PAD_WAKEUP_POL_VAL 的其中一个值 PAD_WAKEUP_POL_HIGH: 高电平唤醒 PAD_WAKEUP_POL_LOW: 低电平唤醒
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Config P0_2 wake up polarity*/
Pad_WakeupPolarityValue(P0_2, PAD_WAKEUP_POL_HIGH);
```

8.1.9 函数 Pad_PowerOrShutdownValue

表 8-11 函数 Pad_PowerOrShutdownValue

函数名	Pad_PowerOrShutDownValue
函数原型	void Pad_PowerOrShutDownValue (uint8_t Pin_Num, uint8_t value)
功能描述	该参数设置管脚的模式
输入参数 1	Pin_Num: 所需配置的管脚值, 具体参阅 Pin_Num 介绍部分
输入参数 2	value: 该参数设置管脚的供电模式, 该值为枚举_PAD_PWR_Mode 的其中一个值 PAD_NOT_PWRON: 供电关闭 PAD_IS_PWRON: 供电打开
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Enable P0_2 power on function*/
Pad_PowerOrShutDownValue (P0_2, PAD_IS_PWRON);
```

8.1.10 函数 Pad_ControlSelectValue

表 8-12 函数 Pad_ControlSelectValue

函数名	Pad_ControlSelectValue
函数原型	void Pad_ControlSelectValue (uint8_t Pin_Num, uint8_t value)
功能描述	该参数设置管脚的模式
输入参数 1	Pin_Num: 所需配置的管脚值, 具体参阅 Pin_Num 介绍部分。
输入参数 2	value: 该参数设置管脚的模式, 该值为枚举 PAD_Mode 的其中一个值 PAD_SW_MODE: 软件模式 PAD_PINMUX_MODE: pinmux 模式
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Config P0_2 sw mode */
Pad_ControlSelectValue (P0_2, PAD_SW_MODE);
```

8.1.11 函数 Pad_WKDebounceConfig

表 8-13 函数 Pad_WKDebounceConfig

函数名	Pad_WKDebounceConfig
函数原型	void Pad_WKDebounceConfig(uint8_t Pin_Num, uint8_t value)
功能描述	使能参数设置管脚的 Debounce 功能
输入参数 1	Pin_Num: 所需配置的管脚值, 具体参阅 Pin_Num 介绍部分

输入参数 2	value: 使能或失能, 该值为枚举_PAD_WAKEUP_DEBOUNCE_EN 的其中一个值 PAD_WK_DEBOUNCE_ENABLE: 使能 PAD_WK_DEBOUNCE_DISABLE: 失能
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Enable P0_2 debounce function */
Pad_WKDebounceConfig(P0_2, PAD_WK_DEBOUNCE_ENABLE);
```

8.1.12 函数 Pad_WakeupInterruptValue

表 8-14 函数 Pad_WakeupInterruptValue

函数名	Pad_WakeupInterruptValue
函数原型	uint8_t Pad_WakeupInterruptValue(uint8_t Pin_Num)
功能描述	读取管脚的中断状态
输入参数 1	Pin_Num: 所需配置的管脚值, 具体参阅 Pin_Num 介绍部分
输出参数	无
返回值	中断状态: 1:该管脚唤醒系统, 0: 该管脚未唤醒系统
先决条件	无
被调用函数	无

例:

```
/* Read P0_2 interrupt value*/
uint8_t intvalue= Pad_WakeupInterruptValue (P0_2);
```

8.1.13 函数 Pad_ClearWakeupINTPendingBit

表 8-15 函数 Pad_ClearWakeupINTPendingBit

函数名	Pad_ClearWakeupINTPendingBit
函数原型	void Pad_ClearWakeupINTPendingBit(uint8_t Pin_Num)
功能描述	清除参数设置管脚的唤醒标志位
输入参数 1	Pin_Num: 所需配置的管脚值, 具体参阅 Pin_Num 介绍部分
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Clear P0_2 wake up interrupt flag bit*/
Pad_ClearWakeupINTPendingBit (P0_2);
```

8.1.14 函数 System_WakeUpInterruptValue

表 8-16 函数 System_WakeUpInterruptValue

函数名	System_WakeUpInterruptValue
函数原型	void System_WakeUpInterruptValue (uint8_t Pin_Num)
功能描述	读取管脚的中断状态
输入参数 1	Pin_Num: 所需配置的管脚值, 具体参阅 Pin_Num 介绍部分
输出参数	无
返回值	中断状态: 1:该管脚唤醒系统, 0: 该管脚未唤醒系统
先决条件	无
被调用函数	无

例:

```
/* Read P0_2 interrupt value*/
uint8_t intvalue= System_WakeUpInterruptValue(P0_2);
```

8.1.15 函数 System_WakeUpDebounceTime

表 8-17 函数 System_WakeUpDebounceTime

函数名	System_WakeUpDebounceTime
函数原型	void System_WakeUpDebounceTime (uint8_t time)
功能描述	配置管脚唤醒的延时时间
输入参数	time:唤醒时间 1-64ms
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Configure all pin wake up debounce time */
System_WakeUpDebounceTime(10);//10ms
```

8.1.16 函数 System_WakeUpPinEnable

表 8-18 函数 System_WakeUpPinEnable

函数名	System_WakeUpPinEnable
函数原型	void System_WakeUpPinEnable(uint8_t Pin_Num, uint8_t Polarity, uint8_t DebounceEn)
功能描述	使能指定管脚唤醒系统的功能
输入参数 1	Pin_Num: 所需配置的管脚值, 具体参阅 Pin_Num 介绍部分
输入参数 2	Polarity: 唤醒系统的电平极性, 该值为 0 或者 1 0: 高电平唤醒

	1: 低电平唤醒
输入参数 3	DebounceEn: 使能延时功能 1: 使能 0: 失能
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Configure P0_2 for low voltage wake up */
System_WakeUpPinEnable(P0_2, 1, 1);
```

8.1.17 函数 System_WakeUpPinDisable

表 8-19 函数 System_WakeUpPinDisable

函数名	System_WakeUpPinDisable
函数原型	void System_WakeUpPinDisable (uint8_t Pin_Num)
功能描述	失能指定管脚唤醒系统的功能
输入参数	Pin_Num: 所需配置的管脚值, 具体参阅 Pin_Num 介绍部分
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Disable P0_2 for wake up system */
System_WakeUpPinDisable (P0_2);
```

9 引脚复用(PINMUX)

9.1 PINMUX 库函数

错误!未找到引用源。例举 PINMUX 的所有库函数

表 9-1 PINMUX 库函数

函数名	描述
Pinmux_Config	配置指定管脚的使用功能
Pinmux_Deinit	将指定管脚配置成空闲模式(IDLE_MODE)
Pinmux_Reset	将所有管脚配置成空闲模式(IDLE_MODE)

9.1.1 函数 Pinmux_Config

表 9-2 函数 Pinmux_Config

函数名	Pinmux_Config
函数原型	void Pinmux_Config(uint8_t Pin_Num, uint8_t Pin_Func)
功能描述	配置指定管脚的使用功能
输入参数 1	Pin_Num: 所需配置的管脚值, 具体参阅 Pin_Num 介绍部分
输入参数 2	Pin_Func: 该参数设置管脚的具体功能
输出参数	无
返回值	无
先决条件	无
被调用函数	无

Pin_Func: 所有可选功能如错误!未找到引用源。所示。

表 9-3 Pin_Func 值

Pin_Func	描述
IDLE	空闲模式
HCI_UART_TX	HCI UART 发送
HCI_UART_RX	HCI UART 接收
HCI_UART_CTS	HCI UART 发送允许
HCI_UART_RTS	HCI UART 发送请求
I2C0_CLK	I2C0 的时钟线
I2C0_DAT	I2C0 的数据线
I2C1_CLK	I2C1 的时钟线
I2C1_DAT	I2C1 的数据线
PWM2_P	PWM2 互补输出 P 通道
PWM2_N	PWM2 互补输出 N 通道
PWM3_P	PWM3 互补输出 P 通道
PWM3_N	PWM3 互补输出 N 通道
PWM0	PWM 通道 0 输出
PWM1	PWM 通道 1 输出
PWM2	PWM 通道 2 输出
PWM3	PWM 通道 3 输出
PWM4	PWM 通道 0 输出
PWM5	PWM 通道 1 输出
PWM6	PWM 通道 2 输出
PWM7	PWM 通道 3 输出
qdec_phase_a_x	QDEC x 轴 a 通道
qdec_phase_b_x	QDEC x 轴 b 通道
qdec_phase_a_y	QDEC y 轴 a 通道
qdec_phase_b_y	QDEC y 轴 b 通道

qdec_phase_a_z	QDEC z 轴 a 通道
qdec_phase_b_z	QDEC z 轴 b 通道
UART2_TX	UART2 发送
UART2_RX	UART2 接收
UART1_TX	UART1 发送
UART1_RX	UART1 接收
UART1_CTS	UART1 发送允许
UART1_RTS	UART1 发送请求
IRDA_TX	红外接收
IRDA_RX	红外发送
UART0_TX	UART0 发送
UART0_RX	UART0 接收
UART0_CTS	UART0 发送允许
UART0_RTS	UART0 发送请求
SPI1_SS_N_0 (master only)	SPI1 在主机模式下的片选信号通道 0 线
SPI1_SS_N_1 (master only)	SPI1 在主机模式下的片选信号通道 1 线
SPI1_SS_N_2 (master only)	SPI1 在主机模式下的片选信号通道 2 线
SPI1_CLK (master only)	SPI1 在主机模式下的时钟线
SPI1_MO (master only)	SPI1 在主机模式下的主机输出从机输入线
SPI1_MI (master only)	SPI1 在主机模式下的主机输入从机输出线
SPI0_SS_N_0 (slave)	SPI0 在从机模式下的片选信号线
SPI0_CLK (slave)	SPI0 在从机模式下的时钟信号线
SPI0_SO (slave)	SPI0 在从机模式下的主机输入从机输出线
SPI0_SI (slave)	SPI0 在从机模式下的主机输出从机输入线
SPI0_SS_N_0 (master only)	SPI0 在主机模式下的片选信号线
SPI0_CLK (master only)	SPI0 在主机模式下的时钟信号线
SPI0_MO (master only)	SPI0 在主机模式下的主机输出从机输入线
SPI0_MI (master only)	SPI0 在主机模式下的主机输入从机输出线
SPI2W_DATA (master only)	两线/三线 SPI 在主机模式下的数据线
SPI2W_CLK (master only)	两线/三线 SPI 在主机模式下的时钟线
SPI2W_CS (master only)	两线/三线 SPI 在主机模式下的片选线
SWD_CLK	SWD 的时钟线
SWD_DIO	SWD 的数据线
KEY_COL_0	Keyscan 的第 0 列
KEY_COL_1	Keyscan 的第 1 列
KEY_COL_2	Keyscan 的第 2 列
KEY_COL_3	Keyscan 的第 3 列
KEY_COL_4	Keyscan 的第 4 列
KEY_COL_5	Keyscan 的第 5 列
KEY_COL_6	Keyscan 的第 6 列
KEY_COL_7	Keyscan 的第 7 列

KEY_COL_8	Keyscan 的第 8 列
KEY_COL_9	Keyscan 的第 9 列
KEY_COL_10	Keyscan 的第 10 列
KEY_COL_11	Keyscan 的第 11 列
KEY_COL_12	Keyscan 的第 12 列
KEY_COL_13	Keyscan 的第 13 列
KEY_COL_14	Keyscan 的第 14 列
KEY_COL_15	Keyscan 的第 15 列
KEY_COL_16	Keyscan 的第 16 列
KEY_COL_17	Keyscan 的第 17 列
KEY_COL_18	Keyscan 的第 18 列
KEY_COL_19	Keyscan 的第 19 列
KEY_ROW_0	Keyscan 的第 0 行
KEY_ROW_1	Keyscan 的第 1 行
KEY_ROW_2	Keyscan 的第 2 行
KEY_ROW_3	Keyscan 的第 3 行
KEY_ROW_4	Keyscan 的第 4 行
KEY_ROW_5	Keyscan 的第 5 行
KEY_ROW_6	Keyscan 的第 6 行
KEY_ROW_7	Keyscan 的第 7 行
KEY_ROW_8	Keyscan 的第 8 行
KEY_ROW_9	Keyscan 的第 9 行
KEY_ROW_10	Keyscan 的第 10 行
KEY_ROW_11	Keyscan 的第 11 行
DWGPI0	GPIO 功能
LRC_SPORT1	I2S1 LRCK 左右通道选择线
BCLK_SPORT1	I2S1 BIT 时钟线
ADCDAT_SPORT1	I2S1 输入线
DACDAT_SPORT1	I2S1 输出线
DMIC1_CLK	DMIC 的数据线
DMIC1_DAT	DMIC 的时钟线
LRC_I_CODEC_SLAVE	I2S Slave 左右通道选择线
BCLK_I_CODEC_SLAVE	I2S BIT 时钟线
SDI_CODEC_SLAVE	CODEC I2S 输入线
SDO_CODEC_SLAVE	CODEC I2S 输出线
LRC_I_PCM	PCM LR 左右通道选择线
BCLK_I_PCM	PCM BIT 时钟线
UART2_CTS	UART2 发送允许
UART2_RTS	UART2 发送请求
BT_COEX_I_0	保留
BT_COEX_I_1	保留

BT_COEX_I_2	保留
BT_COEX_I_3	保留
BT_COEX_O_0	保留
BT_COEX_O_1	保留
BT_COEX_O_2	保留
BT_COEX_O_3	保留
PTA_I2C_CLK_SLAVE	保留
PTA_I2C_DAT_SLAVE	保留
PTA_I2C_INT_OUT	保留
EN_EXPA	保留
EN_EXLNA	保留
ANT_SW0	保留
ANT_SW1	保留
ANT_SW2	保留
ANT_SW3	保留
LRC_SPORT0	I2S0 LRCK 左右通道选择线
BCLK_SPORT0	I2S0 BIT 时钟线
ADCDAT_SPORT0	I2S0 输入线
DACDAT_SPORT0	I2S0 输出线
MCLK	I2S MCLK 输出线

例：

```
/* Configure P0_2 for GPIO mode */
Pinmux_Config(P0_2, DWGPIO);
```

9.1.2 函数 Pinmux_Deinit

表 9-4 函数 Pinmux_Deinit

函数名	Pinmux_Deinit
函数原型	void Pinmux_Deinit(uint8_t Pin_Num)
功能描述	将指定管脚配置成空闲模式(IDLE_MODE)
输入参数	Pin_Num：所需配置的管脚值，具体参阅 Pin_Num 介绍部分
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Configure P0_2 for idle mode */
Pinmux_Deinit (P0_2);
```

9.1.3 函数 Pinmux_Reset

表 9-5 函数 Pinmux_Reset

函数名	Pinmux_Reset
函数原型	void Pinmux_Reset(void)
功能描述	将所有管脚配置成空闲模式(IDLE_MODE)
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Configure all pin for idle mode */  
Pinmux_Reset();
```

10 低功耗比较器(LPC)

10.1 LPC 寄存器结构

```
typedef struct
{
    __IO uint32_t LPC_CRO;
    __IO uint32_t LPC_SR;
    __IO uint32_t LPC_CMP;
    __IO uint32_t LPC_CNT;
} LPC_TypeDef;
```

错误!未找到引用源。例举 LPC 所有寄存器

表 10-1 LPC 寄存器

寄存器	描述
LPC_CRO	控制寄存器 0
LPC_SR	状态寄存器
LPC_CMP	比较值加载寄存器
LPC_CNT	比较计数器

10.2 LPC 库函数

错误!未找到引用源。例举 LPC 的所有库函数

表 10-2 LPC 库函数

函数名	描述
LPC_Init	根据 LPC_InitStruct 中指定参数初始化 LPC 寄存器
LPC_StructInit	把 LPC_InitStruct 中的每一个参数按缺省值填入
LPC_Cmd	使能或者失能 LPC
LPC_CounterCmd	使能或者失能 LPC 计数器
LPC_CounterReset	复位 LPC 计数器
LPC_CompValueSet	设置 LPC 比较器值
LPC_INTConfig	使能或者失能 LPC 指定的中断源
LPC_ClearINTPendingBit	清除 LPC 某一中断源中断挂起位
LPC_GetINTStatus	获得 LPC 指定的中断源状态
LPC_GetComp	读取 LPC 比较器值
LPC_GetCounter	读取 LPC 计数器值

10.2.1 函数 LPC_Init

表 10-3 函数 LPC_Init

函数名	LPC_Init
函数原型	void LPC_Init(LPC_InitTypeDef *LPC_InitStruct)
功能描述	根据 LPC_InitStruct 中指定参数初始化 LPC 寄存器
输入参数	LPC_InitStruct: 指向 LPC_InitTypeDef 的指针, LPC_InitStruct 中包含相关配置信息
输出参数	无
返回值	无
先决条件	无
被调用函数	无

```
typedef struct
{
    uint16_t LPC_Channel;           /*!< Specifies the input pin. */
    uint32_t LPC_Edge;             /*!< Specifies the comparator output edge */
    uint32_t LPC_Threshold;         /*!< Specifies the threshold value of comparator voltage. */
} LPC_InitTypeDef;
```

LPC_Channel: 设置 LPC 通道。错误!未找到引用源。给出了该参数可取的值。

表 10-4 LPC_Channel 值

LPC_Channel	描述
LPC_CHANNEL_P2_0	P2_0 管脚
LPC_CHANNEL_P2_1	P2_1 管脚
LPC_CHANNEL_P2_2	P2_2 管脚
LPC_CHANNEL_P2_3	P2_3 管脚
LPC_CHANNEL_P2_4	P2_4 管脚
LPC_CHANNEL_P2_5	P2_5 管脚
LPC_CHANNEL_P2_6	P2_6 管脚
LPC_CHANNEL_P2_7	P2_7 管脚
LPC_CHANNEL_VBAT	Vbat 管脚

LPC_Edge: 设置 LPC 比较器输出极性。错误!未找到引用源。给出了该参数可取的值。

表 10-5 LPC_Edge 值

LPC_Edge	描述
LPC_Vin_Below_Vth	低于设置的电压阈值
LPC_Vin_Over_Vth	高于设置的电压阈值

LPC_Threshold: 设置 LPC 比较器的电压阈值。错误!未找到引用源。给出了该参数可取的值。

表 10-6 LPC_Threshold 值

LPC_Threshold	描述
LPC_80_mV	电压阈值为 80mv

LPC_160_mV	电压阈值为 160mv
LPC_240_mV	电压阈值为 240mv
LPC_320_mV	电压阈值为 320mv
LPC_400_mV	电压阈值为 400mv
LPC_480_mV	电压阈值为 480mv
LPC_560_mV	电压阈值为 560mv
LPC_640_mV	电压阈值为 640mv
LPC_680_mV	电压阈值为 680mv
LPC_720_mV	电压阈值为 720mv
LPC_760_mV	电压阈值为 760mv
LPC_800_mV	电压阈值为 800mv
LPC_840_mV	电压阈值为 840mv
LPC_880_mV	电压阈值为 880mv
LPC_920_mV	电压阈值为 920mv
LPC_960_mV	电压阈值为 960mv
LPC_1000_mV	电压阈值为 100mv
LPC_1040_mV	电压阈值为 1040mv
LPC_1080_mV	电压阈值为 1080mv
LPC_1120_mV	电压阈值为 1120mv
LPC_1160_mV	电压阈值为 1160mv
LPC_1200_mV	电压阈值为 1200mv
LPC_1240_mV	电压阈值为 1240mv
LPC_1280_mV	电压阈值为 1280mv
LPC_1320_mV	电压阈值为 1320mv
LPC_1360_mV	电压阈值为 1360mv
LPC_1400_mV	电压阈值为 1400mv
LPC_1440_mV	电压阈值为 1440mv
LPC_1480_mV	电压阈值为 1480mv
LPC_1520_mV	电压阈值为 1520mv
LPC_1560_mV	电压阈值为 1560mv
LPC_1600_mV	电压阈值为 1600mv
LPC_1640_mV	电压阈值为 1640mv
LPC_1680_mV	电压阈值为 1680mv
LPC_1720_mV	电压阈值为 1720mv
LPC_1760_mV	电压阈值为 1760mv
LPC_1800_mV	电压阈值为 1800mv
LPC_1840_mV	电压阈值为 1840mv
LPC_1880_mV	电压阈值为 1880mv
LPC_1920_mV	电压阈值为 1920mv
LPC_1960_mV	电压阈值为 1960mv
LPC_2000_mV	电压阈值为 2000mv

LPC_2040_mV	电压阈值为 2040mv
LPC_2080_mV	电压阈值为 2080mv
LPC_2120_mV	电压阈值为 2120mv
LPC_2160_mV	电压阈值为 2160mv
LPC_2200_mV	电压阈值为 2200mv
LPC_2240_mV	电压阈值为 2240mv
LPC_2280_mV	电压阈值为 2280mv
LPC_2320_mV	电压阈值为 2320mv
LPC_2360_mV	电压阈值为 2360mv
LPC_2400_mV	电压阈值为 2400mv
LPC_2440_mV	电压阈值为 2440mv
LPC_2480_mV	电压阈值为 2480mv
LPC_2520_mV	电压阈值为 2520mv
LPC_2560_mV	电压阈值为 2560mv
LPC_2640_mV	电压阈值为 2640mv
LPC_2720_mV	电压阈值为 2720mv
LPC_2800_mV	电压阈值为 2800mv
LPC_2880_mV	电压阈值为 2880mv
LPC_2960_mV	电压阈值为 2960mv
LPC_3040_mV	电压阈值为 3040mv
LPC_3120_mV	电压阈值为 3120mv
LPC_3200_mV	电压阈值为 3200mv

例：

```
/* Initialize LPC */
LPC_InitTypeDef LPC_InitStruct;
LPC_InitStruct.LPC_Channel    = LPC_CAPTURE_PIN;
LPC_InitStruct.LPC_Edge       = LPC_Vin_Below_Vth;
LPC_InitStruct.LPC_Threshold  = LPC_1600_mV;
LPC_Init(&LPC_InitStruct);
```

10.2.2 函数 **LPC_StructInit**

表 10-7 函数 **LPC_StructInit**

函数名	LPC_StructInit
函数原型	LPC_StructInit(LPC_InitTypeDef* LPC_InitStruct)
功能描述	把 LPC_InitStruct 中的每一个值按照缺省值填入
输入参数	LPC_InitStruct : 指向结构体 LPC_InitTypeDef 的指针, 包含了外设 LPC 的配置参数
输出参数	无
返回值	无
先决条件	无

被调用函数	无
-------	---

例：

```
/* Initialize LPC */  
LPC_InitTypeDef  LPC_InitStruct;  
LPC_StructInit(&LPC_InitStruct);
```

10.2.3 函数 **LPC_Cmd**

表 10-8 函数 **LPC_Cmd**

函数名	LPC_Cmd
函数原型	void LPC_Cmd(FunctionalState NewState)
功能描述	使能或者失能 LPC
输入参数	NewState: LPC 的新状态 这个参数可以取 ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable LPC */  
LPC_Cmd(ENABLE);
```

10.2.4 函数 **LPC_CounterCmd**

表 10-9 函数 **LPC_CounterCmd**

函数名	LPC_CounterCmd
函数原型	void LPC_CounterCmd (FunctionalState NewState)
功能描述	使能或者失能 LPC 计数器
输入参数	NewState: LPC 的新状态 这个参数可以取 ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Start LPC counter */  
LPC_CounterCmd(ENABLE);
```

10.2.5 函数 **LPC_ResetCounter**

表 10-10 函数 **LPC_CounterReset**

函数名	LPC_ResetCounter
函数原型	void LPC_ResetCounter (void)
功能描述	复位 LPC 计数器
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Reset LPC counter */
LPC_CounterReset();
```

10.2.6 函数 **LPC_SetCompValue**

表 10-11 函数 **LPC_SetCompValue**

函数名	LPC_SetCompValue
函数原型	void LPC_SetCompValue (uint32_t value)
功能描述	设置 LPC 比较器值
输入参数	value: LPC 新的比较器值，该参数的取值范围为 0x0 至 0xFFFF
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/*Configure LPC comparator value */
LPC_SetCompValue (0x200);
```

10.2.7 函数 **LPC_GetCompValue**

表 10-12 函数 **LPC_GetCompValue**

函数名	LPC_GetCompValue
函数原型	uint16_t LPC_GetCompValue (void)
功能描述	读取 LPC 比较器值
输入参数	无
输出参数	无
返回值	LPC 比较器值，该数据的取值范围为 0x0 至 0xFFFF

先决条件	无
被调用函数	无

例：

```
uint16_t value = 0;
/* Get LPC comparator value */
value = LPC_GetCompValue();
```

10.2.8 函数 **LPC_GetCounter**

表 10-13 函数 **LPC_GetCounter**

函数名	LPC_GetCounter
函数原型	<code>uint16_t LPC_GetCounter (void)</code>
功能描述	读取 LPC 计数器值
输入参数	无
输出参数	无
返回值	LPC 计数值，该数据的取值范围为 0x0 至 0xFFFF
先决条件	无
被调用函数	无

例：

```
uint16_t value = 0;
/* Get LPC counter value */
value = LPC_GetCounter();
```

10.2.9 函数 **LPC_INTConfig**

表 10-14 函数 **LPC_INTConfig**

函数名	LPC_INTConfig
函数原型	<code>void LPC_INTConfig(uint32_t LPC_INT, FunctionalState NewState)</code>
功能描述	使能或者失能 LPC 指定的中断源
输入参数 1	LPC_INT : 待使能或者失能的 LPC 中断源，具体参阅错误!未找到引用源。关于 LPC_INT 介绍部分
输入参数 2	NewState : 中断的新状态 这个参数可以取 ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

LPC_INT: 允许的 LPC 中断类型如错误!未找到引用源。所示，使用操作符“|”可以一次选中多个中断作为该参数的值。

表 10-15 **LPC_INT** 值

LPC_INT	描述
LPC_INT_VOLTAGE_COMP	检测电压符合 LPC 比较器输出极性时触发该中断
LPC_INT_COUNT_COMP	计数器数据与比较器数据相等时，触发该中断

例：

```
/* Enable voltage detection interrupt.If Vin<Vth, cause this interrupt */
LPC_INTConfig(LPC_INT_VOLTAGE_COMP, ENABLE);
```

10.2.10 函数 **LPC_ClearINTPendingBit**

表 10-16 函数 **LPC_ClearINTPendingBit**

函数名	LPC_ClearINTPendingBit
函数原型	void LPC_ClearINTPendingBit(uint32_t LPC_INT)
功能描述	清除 LPC 某一中断源中断挂起位
输入参数	LPC_INT : LPC 中断清除类型，具体参阅 错误!未找到引用源 。关于 LPC_INT 介绍部分
输出参数	无
返回值	无
先决条件	无
被调用函数	无

LPC_INT: 允许的 LPC 中断清除类型如**错误!未找到引用源** 所示。

表 10-17 **LPC_INT** 值

LPC_INT	描述
LPC_INT_COUNT_COMP	清除当计数器数据与比较器数据相等时触发的中断

例：

```
/* Clear comparator interrupt */
LPC_ClearINTPendingBit(LPC_INT_COUNT_COMP);
```

10.2.11 函数 **LPC_GetINTStatus**

表 10-18 函数 **LPC_GetINTStatus**

函数名	LPC_GetINTStatus
函数原型	ITStatus LPC_GetINTStatus(uint32_t LPC_INT)
功能描述	获得 LPC 指定的中断源状态
输入参数	LPC_INT : 指定的中断源类型，具体参阅 错误!未找到引用源 。关于 LPC_INT 介绍部分
输出参数	无
返回值	LPC 指定中断的新状态(SET 或者 RESET)
先决条件	无
被调用函数	无

例：

```
/* Get LPC the specified interrupt status */
ITStatus LPC_Status = RESET;
```

```
LPC_Status = LPC_GetINTStatus(LPC_INT_COUNT_COMP);
```

11 实时时钟(RTC)

11.1 RTC 寄存器结构

```
typedef struct
{
    __IO uint32_t CR0;
    __IO uint32_t INT_CLR;
    __IO uint32_t INT_SR;
    __IO uint32_t PRESCALER;
    __IO uint32_t COMPO;
    __IO uint32_t COMP1;
    __IO uint32_t COMP2;
    __IO uint32_t COMP3;
    __I  uint32_t CNT;
    __IO uint32_t PRE_CNT;
    __IO uint32_t PRE_COMP;
} RTC_TypeDef;
```

错误!未找到引用源。例举 RTC 所有寄存器

表 11-1 RTC 寄存器

寄存器	描述
CR0	控制寄存器 0
INT_CLR	中断清楚除寄存器
INT_SR	中断状态寄存器
PRESCALER	预分频寄存器
COMPO	比较器 0
COMP1	比较器 1
COMP2	比较器 2
COMP3	比较器 3
CNT	计数器
PRE_CNT	预分频计数器
PRE_COMP	预分频比较器

11.2 RTC 库函数

错误!未找到引用源。例举 RTC 的所有库函数

表 11-2 RTC 库函数

函数名	描述
RTC_DeInit	复位 RTC 外设

RTC_PrescalerSet	设置 RTC 的预分频系数
RTC_PreCompValueSet	设置 RTC 预分频比较值
RTC_CompValueSet	设置 RTC 比较器通道和比较值
RTC_RunCmd	启动或者停止 RTC 外设
RTC_MaskINTConfig	屏蔽或者不屏蔽 RTC 指定中断源
RTC_INTConfig	使能或失能 RTC 中断
RTC_NvCmd	使能或失能 RTC 中断输出信号到 MCU
RTC_ClearINTPendingBit	清除 RTC 中断挂起标志位
RTC_GetINTStatus	获取 RTC 指定中断源状态
RTC_SystemWakeupConfig	使能或者失能 RTC 唤醒系统功能
RTC_GetCounter	获取 RTC 计数值
RTC_GePreCounter	获取 RTC 预分频计数值
RTC_CounterReset	复位 RTC 计数器
RTC_PrescalerCounterReset	复位 RTC 预分频计数器
RTC_GetComp	获取 RTC 指定通道的比较值
RTC_GetPreComp	获取 RTC 预分频的比较值
RTC_ClearCompINT	清除 RTC 指定通道的比较中断
RTC_ClearOverFlowINT	清除 RTC 溢出中断
RTC_ClearTickINT	清除 RTC 滴答中断
RTC_BackupRegWrite	写 RTC BackUp 寄存器
RTC_BackupRegRead	读 RTC BackUp 寄存器

11.2.1 函数 RTC_DeInit

表 11-3 函数 RTC_DeInit

函数名	RTC_DeInit
函数原型	void RTC_DeInit(void)
功能描述	复位 RTC 外设
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Reset RTC */
RTC_DeInit();
```

11.2.2 函数 RTC_SetPrescaler

表 11-4 函数 RTC_SetPrescaler

函数名	<code>RTC_SetPrescaler</code>
函数原型	<code>void RTC_SetPrescaler (uint32_t value)</code>
功能描述	设置 RTC 的预分频系数
输入参数	<code>value</code> : 预分频系数值。该值取值范围为 0x00 至 0xffff 计数器频率(Hz): RTC 时钟/(value +1)
输出参数	无
返回值	无
先决条件	必须在 RTC 停止的情况下设置分频系数
被调用函数	无
函数名	无

例:

```
/* Configure RTC prescaler */
RTC_SetPrescaler (0);
```

11.2.3 函数 `RTC_SetPreCompValue`

表 11-5 函数 `RTC_SetPreCompValue`

函数名	<code>RTC_SetPreCompValue</code>
函数原型	<code>void RTC_SetPreCompValue (uint32_t value)</code>
功能描述	设置 RTC 的预分频比较值
输入参数	<code>value</code> : 预分频比较值。该值取值范围为 0x00 至 0xffff
输出参数	无
返回值	无
先决条件	必须在 RTC 停止的情况下设置分频系数
被调用函数	无
函数名	无

例:

```
/* Configure RTC prescaler */
RTC_SetPreCompValue (0);
```

11.2.4 函数 `RTC_SetCompValue`

表 11-6 函数 `RTC_SetCompValue`

函数名	<code>RTC_SetCompValue</code>
函数原型	<code>void RTC_SetCompValue (uint8_t index, uint32_t value)</code>
功能描述	设置比较器通道和比较值
输入参数 1	<code>index</code> : 比较器通道的索引号, 该值可配置范围为 0 到 3。 0: 比较器 0

	1: 比较器 1 2: 比较器 2 3: 比较器 3
输入参数 2	value: 指定比较器的比较值。该值取值范围为 0x00 至 0xffff
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Configure RTC comparator 0 */
RTC_SetCompValue (0, 32768);
```

11.2.5 函数 RTC_Cmd

表 11-7 函数 RTC_Cmd

函数名	RTC_Cmd
函数原型	void RTC_Cmd (FunctionalState NewState)
功能描述	启动或者停止 RTC 外设
输入参数	NewState: 外设 RTC 的新状态 这个参数可以取：ENABLE 或者 DISABLE ENABLE: 启动 RTC DISABLE: 停止 RTC
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Start RTC */
RTC_Cmd (ENABLE);
```

11.2.6 函数 RTC_MaskINTConfig

表 11-8 函数 RTC_MaskINTConfig

函数名	RTC_MaskINTConfig
函数原型	void RTC_MaskINTConfig(uint32_t RTC_INT, FunctionalState NewState)
功能描述	屏蔽或者不屏蔽 RTC 指定中断源
输入参数 1	RTC_INT: 待屏蔽或者不屏蔽的 RTC 中断类型，具体参阅错误!未找到引用源。 RTC_INT 介绍部分
输入参数 2	NewState: 中断的新状态 可选参数：ENABLE 或者 DISABLE

输出参数	无
返回值	无
先决条件	无
被调用函数	无

RTC_INT: 允许的 RTC 中断类型如错误!未找到引用源。所示，使用操作符“|”可以一次选中多个中断作为该参数的值。

表 11-9 RTC_INT 值

RTC_INT	描述
RTC_INT_TICK	滴答中断
RTC_INT_OVF	RTC 计数器溢出中断
RTC_INT_COMPO	比较器通道 0 计数中断
RTC_INT_COMP1	比较器通道 1 计数中断
RTC_INT_COMP2	比较器通道 2 计数中断
RTC_INT_COMP3	比较器通道 3 计数中断
RTC_INT_PRE_COMP	预分频比较中断
RTC_INT_PRE_COMP3	预分频比较和比较器通道三比较中断

例：

```
/* Unmask RTC overflow interrupt */
RTC_MaskINTConfig(RTC_INT_OVF, DISABLE);
```

11.2.7 函数 RTC_INTConfig

表 11-10 函数 RTC_INTConfig

函数名	RTC_INTConfig
函数原型	void RTC_INTConfig (uint32_t RTC_INT, FunctionalState NewState)
功能描述	使能或失能 RTC 指定通道的比较中断
输入参数 1	RTC_INT: 待使能或者失能的 RTC 中断，具体参阅错误!未找到引用源。 RTC_INT 介绍部分
输入参数 2	NewState: 中断的新状态 可选参数: ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

RTC_INT: 允许的 RTC 中断类型如错误!未找到引用源。所示，使用操作符“|”可以一次选中多个中断作为该参数的值。

表 11-11 RTC_INT 值

RTC_INT	描述
RTC_INT_TICK	RTC Tick 中断
RTC_INT_COMPO	比较器通道 0 计数中断
RTC_INT_COMP1	比较器通道 1 计数中断

RTC_INT_COMP2	比较器通道 2 计数中断
RTC_INT_COMP3	比较器通道 3 计数中断
RTC_INT_PRE_COMP	预分频比较中断
RTC_INT_PRE_COMP3	预分频比较和比较器通道三比较中断

例：

```
/* Enable RTC comparator 1 interrupt */
RTC_CompINTConfig(RTC_INT_CMP_1, ENABLE);
```

11.2.8 函数 RTC_NvCmd

表 11-12 函数 RTC_NvCmd

函数名	RTC_NvCmd
函数原型	void RTC_NvCmd (FunctionalState NewState)
功能描述	使能或失能 RTC 中断信号到 MCU
输入参数	NewState: 中断的新状态 可选参数: ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable RTC interrupt signal to MCU */
RTC_NvCmd (ENABLE);
```

11.2.9 函数 RTC_ClearINTPendingBit

表 11-13 函数 RTC_ClearINTPendingBit

函数名	RTC_ClearINTPendingBit
函数原型	void RTC_ClearINTPendingBit (uint32_t RTC_INT)
输入参数	RTC_INT: 清除中断标志位, 具体参阅错误!未找到引用源。 RTC_INT 介绍部分
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear interrupt of RTC */
RTC_ClearINTPendingBit (RTC_INT_COMP1);
```

11.2.10 函数 RTC_GetINTStatus

表 11-14 函数 RTC_GetINTStatus

函数名	RTC_GetINTStatus
函数原型	ITStatus RTC_GetINTStatus(uint32_t RTC_INT)
输入参数	RTC_INT: 待屏蔽或者不屏蔽的 RTC 中断类型, 具体参阅错误!未找到引用源。 RTC_INT 介绍部分
输出参数	无
返回值	RTC 指定中断的新状态(SET 或者 RESET)
先决条件	无
被调用函数	无

例:

```
/* Get RTC comparator 1 interrupt status */
ITStatus status = RESET;
status = RTC_GetINTStatus(RTC_INT_COMP1);
```

11.2.11 函数 RTC_SystemWakeupConfig

表 11-15 函数 RTC_SystemWakeupConfig

函数名	RTC_SystemWakeupConfig
函数原型	void RTC_SystemWakeupConfig(FunctionalState NewState)
功能描述	使能或者失能 RTC 唤醒系统功能
输入参数	NewState: 中断的新状态 可选参数: ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Enable RTC system wake up function */
RTC_SystemWakeupConfig(ENABLE);
```

11.2.12 函数 RTC_ResetCounter

表 11-16 函数 RTC_ResetCounter

函数名	RTC_ResetCounter
函数原型	void RTC_ResetCounter (void)
功能描述	复位 RTC 计数器

输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear RTC counter value */
RTC_ResetCounter();
```

11.2.13 函数 RTC_ResetPrescalerCounter

表 11-17 函数 RTC_ResetPrescalerCounter

函数名	RTC_ResetPrescalerCounter
函数原型	void RTC_ResetPrescalerCounter (void)
功能描述	复位 RTC prescaler 计数器
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear RTC counter value */
RTC_ResetPrescalerCounter();
```

11.2.14 函数 RTC_GetCounter

表 11-18 函数 RTC_GetCounter

函数名	RTC_GetCounter
函数原型	uint32_t RTC_GetCounter(void)
功能描述	获取 RTC 计数值
输入参数	无
输出参数	无
返回值	当前 RTC 计数器值
先决条件	无
被调用函数	无

例：

```
/* Get RTC counter value */
uint32_t value = 0;
```

```
vuae = RTC_GetCounter();
```

11.2.15 函数 RTC_GetPreCounter

表 11-19 函数 RTC_GetPreCounter

函数名	RTC_GetPreCounter
函数原型	uint32_t RTC_GetPreCounter (void)
功能描述	获取 RTC prescaler 计数值
输入参数	无
输出参数	无
返回值	当前 RTC prescaler 计数器值
先决条件	无
被调用函数	无

例：

```
/* Get RTC counter value */  
uint32_t value = 0;  
vuae = RTC_GetCounter();
```

11.2.16 函数 RTC_GetCompValue

表 11-20 函数 RTC_GetCompValue

函数名	RTC_GetCompValue
函数原型	uint32_t RTC_GetCompValue (uint8_t index)
功能描述	获取 RTC 指定通道的比较值
输入参数	index: 比较器通道的索引号, 该值可配置范围为 0 到 3。 0: 比较器 0 1: 比较器 1 2: 比较器 2 3: 比较器 3
输出参数	无
返回值	指定 RTC 比较器值
先决条件	无
被调用函数	无

例：

```
/* Get RTC comparator 1 value */  
uint32_t value = 0;  
vuae = RTC_GetCompValue (1);
```

11.2.17 函数 RTC_GetPreCompValue

表 11-21 函数 RTC_GetPreCompValue

函数名	RTC_GetPreCompValue
函数原型	uint32_t RTC_GetPreCompValue (void)
功能描述	获取 RTC prescaler 比较值
输入参数	无
输出参数	无
返回值	返回 RTC prescaler 比较器值
先决条件	无
被调用函数	无

例：

```
/* Get RTC prescaler comparator value */  
uint32_t value = 0;  
value = RTC_GetPreCompValue (void);
```

11.2.18 函数 RTC_ClearCompINT

表 11-22 函数 RTC_ClearCompINT

函数名	RTC_ClearCompINT
函数原型	void RTC_ClearCompINT(uint8_t index)
功能描述	清除 RTC 指定通道的比较中断
输入参数	index: 比较器通道的索引号, 该值可配置范围为 0 到 3。 0: 比较器 0 1: 比较器 1 2: 比较器 2 3: 比较器 3
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear RTC comparator 1 interrupt */  
RTC_ClearCompINT(1);
```

11.2.19 函数 RTC_ClearOverflowINT

表 11-23 函数 RTC_ClearOverflowINT r

函数名	RTC_ClearOverflowINT
函数原型	void RTC_ClearOverflowINT(void)
功能描述	清除 RTC 溢出中断
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear RTC overflow interrupt */  
RTC_ClearOverflowINT();
```

11.2.20 函数 RTC_ClearTickINT

表 11-24 函数 RTC_ClearTickINT

函数名	RTC_ClearTickINT
函数原型	void RTC_ClearTickINT(void)
功能描述	清除 RTC 滴答中断
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear RTC tick interrupt */  
RTC_ClearTickINT();
```

12 串型外设接口(SPI)

12.1 SPI 寄存器架构

```
typedef struct
{
    __IO uint32_t CTRLR0;
    __IO uint32_t CTRLR1;
    __IO uint32_t SSIENR;
    __IO uint32_t RSVD_0C;
    __IO uint32_t SER;
    __IO uint32_t BAUDR;
    __IO uint32_t TXFTLR;
    __IO uint32_t RXFTLR;
    __I uint32_t TXFLR;
    __I uint32_t RXFLR;
    __I uint32_t SR;
    __IO uint32_t IMR;
    __I uint32_t ISR;
    __I uint32_t RISR;
    __I uint32_t TXOICR;
    __I uint32_t RXOICR;
    __I uint32_t RXUICR;
    __I uint32_t RSVD_44;
    __I uint32_t ICR;
    __IO uint32_t DMACR;
    __IO uint32_t DMATDLR;
    __IO uint32_t DMARDLR;
    __I uint32_t IDR;
    __I uint32_t SSI_COMP_VERSION;
    __IO uint32_t DR[36];
    __IO uint32_t RX_SAMPLE_DLY;
}SPI_TypeDef;
```

错误!未找到引用源。例举 SPI 所有寄存器

表 12-1 SPI 寄存器

寄存器	描述
CTRLR0	SPI 控制寄存器 0
CTRLR1	SPI 控制寄存器 1
SSIENR	SPI 使能寄存器
RSVD_0C	保留

SER	从设备使能寄存器
BAUDR	波特率选择寄存器
TXFTLR	发送 FIFO 阈值寄存器
RXFTLR	接收 FIFO 阈值寄存器
TXFLR	发送 FIFO 深度寄存器
RXFLR	接收 FIFO 深度寄存器
SR	状态寄存器
IMR	中断屏蔽寄存器
ISR	中断状态寄存器
RISR	原始中断状态寄存器(RAW)
TXOICR	发送 FIFO 溢出中断清除寄存器
RXOICR	接收 FIFO 溢出中断清除寄存器
RXUICR	接收 FIFO 中断清除寄存器(underflow)
RSVD_44	保留
ICR	中断清除寄存器
DMACR	DMA 控制寄存器
DMATDLR	DMA 传输数据深度寄存器
DMARDLR	DMA 接收数据深度寄存器
IDR	ID 寄存器
SSI_COMP_VERSION	SPI IP Core Version 寄存器
DR[36]	数据寄存器
RX_SAMPLE_DLY	RXD Sample Delay Register

12.2 SPI 库函数

表 12-2 例举 SPI 的所有库函数

函数名	描述
SPI_DelInit	关闭指定的 SPI 时钟
SPI_Init	根据 SPI_InitStruct 中指定参数初始化 SPI 寄存器
SPI_StructInit	根据 SPI_InitStruct 中指定参数初始化 SPI 寄存器
SPI_Cmd	使能或者失能指定的 SPI 外设模块
SPI_SendBuffer	通过 SPI 外设发送数据
SPI_SendWord	通过 SPI 外设发送四字节数据
SPI_SendHalfWord	通过 SPI 外设发送二字节数据
SPI_INTConfig	使能或者失能 SPI 指定的中断源
SPI_ClearINTPendingBit	清除挂起的 SPI 中断标志位
SPI_SendData	通过 SPI 外设传输数据
SPI_ReceiveData	SPI 外设接收到的数据
SPI_GetRxFIFOlen	从 SPI 接收 FIFO 中读取接收数据的长度
SPI_GetTxFIFOlen	从 SPI 发送 FIFO 中读取发送数据的长度

SPI_ChangeDirection	重新设置 SPI 的数据传输模式
SPI_SetReadLen	设置读取数据长度
SPI_SetCSNumber	设置片选信号
SPI_GetFlagState	检查指定的 SPI 外设标志位
SPI_GetINTStatus	检查指定的 SPI 外设中断源
SPI_GDMACmd	设置 SPI 传输过程中 GDMA 请求类型

12.2.1 函数 SPI_DeInit

表 12-3 函数 SPI_DeInit

函数名	SPI_DeInit
函数原型	void SPI_DeInit(SPI_TypeDef* SPIx)
功能描述	关闭指定的 SPI 时钟
输入参数 1	SPIx: x 可以设置成 0 或者 1, 选择指定的 SPI 外设
输出参数	无
返回值	无
先决条件	无
被调用函数	RCC_PeriphClockCmd()

例:

```
/* Close SPI0 clock */
SPI_DeInit(SPI0);
```

12.2.2 函数 SPI_Init

表 12-4 函数 SPI_Init

函数名	SPI_Init
函数原型	void SPI_Init(SPI_TypeDef* SPIx, SPI_InitTypeDef* SPI_InitStruct)
功能描述	根据 SPI_InitStruct 中指定参数初始化 SPI 寄存器
输入参数 1	SPIx: x 可以设置成 0 或者 1, 选择指定的 SPI 外设
输入参数 2	SPI_InitStruct: 指向 SPI_InitTypeDef 的指针, SPI_InitStruct 中包含相关配置信息
输出参数	无
返回值	无
先决条件	无
被调用函数	无

```
typedef struct
{
    uint16_t SPI_Direction;
    uint16_t SPI_Mode;
    uint16_t SPI_DataSize;
    uint16_t SPI_CPOL;
```

```
uint16_t SPI_CPHA;
uint32_t SPI_SwapTxBitEn;
uint32_t SPI_SwapRxBitEn;
uint32_t SPI_SwapTxByteEn;
uint32_t SPI_SwapRxByteEn;
uint32_t SPI_ToggleEn;
uint32_t SPI_BaudRatePrescaler;
uint16_t SPI_FrameFormat;
uint32_t SPI_TxThresholdLevel;
uint32_t SPI_RxThresholdLevel;
uint32_t SPI_NDF;
} SPI_InitTypeDef;
```

SPI_Direction: 设置 SPI 的数据传输模式, 错误!未找到引用源。给出了该参数可取的值。

表 12-5 SPI_Direction 值

SPI_Direction	描述
SPI_Direction_FullDuplex	双向全双工模式
SPI_Direction_TxOnly	只发送模式
SPI_Direction_RxOnly	只接收模式
SPI_Direction_EEPROM	EEPROM 模式

SPI_Mode: 设置 SPI 的工作模式, 错误!未找到引用源。给出了该参数可取的值。

表 12-6 SPI_Mode 值

SPI_Direction	描述
SPI_Mode_Master	主设备模式
SPI_Mode_Slave	从设备模式

SPI_DataSize: 设置 SPI 的数据大小, 错误!未找到引用源。给出了该参数可取的值。

表 12-7 SPI_DataSize 值

SPI_DataSize	描述
SPI_DataSize_4b	SPI 发送接收 4 位帧结构
SPI_DataSize_5b	SPI 发送接收 5 位帧结构
SPI_DataSize_6b	SPI 发送接收 6 位帧结构
SPI_DataSize_7b	SPI 发送接收 7 位帧结构
SPI_DataSize_8b	SPI 发送接收 8 位帧结构
SPI_DataSize_9b	SPI 发送接收 9 位帧结构
SPI_DataSize_10b	SPI 发送接收 10 位帧结构
SPI_DataSize_11b	SPI 发送接收 11 位帧结构
SPI_DataSize_12b	SPI 发送接收 12 位帧结构
SPI_DataSize_13b	SPI 发送接收 13 位帧结构
SPI_DataSize_14b	SPI 发送接收 14 位帧结构
SPI_DataSize_15b	SPI 发送接收 15 位帧结构

SPI_DataSize_16b	SPI 发送接收 16 位帧结构
SPI_DataSize_17b	SPI 发送接收 17 位帧结构
SPI_DataSize_18b	SPI 发送接收 18 位帧结构
SPI_DataSize_19b	SPI 发送接收 19 位帧结构
SPI_DataSize_20b	SPI 发送接收 20 位帧结构
SPI_DataSize_21b	SPI 发送接收 21 位帧结构
SPI_DataSize_22b	SPI 发送接收 22 位帧结构
SPI_DataSize_23b	SPI 发送接收 23 位帧结构
SPI_DataSize_24b	SPI 发送接收 24 位帧结构
SPI_DataSize_25b	SPI 发送接收 25 位帧结构
SPI_DataSize_26b	SPI 发送接收 26 位帧结构
SPI_DataSize_27b	SPI 发送接收 27 位帧结构
SPI_DataSize_28b	SPI 发送接收 28 位帧结构
SPI_DataSize_29b	SPI 发送接收 29 位帧结构
SPI_DataSize_30b	SPI 发送接收 30 位帧结构
SPI_DataSize_31b	SPI 发送接收 31 位帧结构
SPI_DataSize_32b	SPI 发送接收 32 位帧结构

SPI_CPOL: 设置串行时钟的稳态，错误!未找到引用源。给出了该参数可取的值。

表 12-8 SPI_CPOL 值

SPI_CPOL	描述
SPI_CPOL_Low	时钟悬空低
SPI_CPOL_High	时钟悬空高

SPI_CPHA: 设置位捕获的时钟活动沿，错误!未找到引用源。给出了该参数可取的值。

表 12-9 SPI_CPHA 值

SPI_CPOL	描述
SPI_CPHA_1Edge	数据捕获于第一个时钟沿
SPI_CPHA_2Edge	数据捕获于第二个时钟沿

SPI_SwapRxBitEn: 设置 SPI 接收数据时，是否反转比特位接收次序。错误!未找到引用源。给出了该参数可取的值。

表 12-10 SPI_SwapRxBitEn 值

SPI_SwapRxBitEn	描述
SPI_SWAP_DISABLE	从 LSB 位接收数据
SPI_SWAP_ENABLE	从 MSB 位接收数据

SPI_SwapTxBitEn: 设置 SPI 发送数据时，是否反转比特位发送次序。错误!未找到引用源。给出了该参数可取的值。

表 12-11 SPI_SwapTxBitEn 值

SPI_SwapTxBitEn	描述
SPI_SWAP_DISABLE	从 LSB 位发送数据

SPI_SWAP_ENABLE	从 MSB 位发送数据
-----------------	-------------

SPI_SwapRxByteEn: 设置 SPI 接收数据时，是否反转字节接收次序。错误!未找到引用源。给出了该参数可取的值。

表 12-12 SPI_SwapRxByteEn 值

SPI_SwapRxByteEn	描述
SPI_SWAP_DISABLE	从低字节接收数据
SPI_SWAP_ENABLE	从高字节接收数据

SPI_SwapTxByteEn: 设置 SPI 发送数据时，是否反转字节发送次序。错误!未找到引用源。给出了该参数可取的值。

表 12-13 SPI_SwapTxByteEn 值

SPI_SwapTxByteEn	描述
SPI_SWAP_DISABLE	从低字节发送数据
SPI_SWAP_ENABLE	从高字节发送数据

SPI_ToggleEn: 设置 SPI 在连续发送数据时，是否触发片选信号。错误!未找到引用源。给出了该参数可取的值。

表 12-14 SPI_ToggleEn 值

SPI_ToggleEn	描述
DISABLE	每发送一个数据，片选信号不会拉高
ENABLE	每发送一个数据，片选信号会拉高

SPI_BaudRatePrescaler: 设置 SPI 外设的时钟分频系数。该频率值必须是系统时钟的偶分频值，并且最大频率为系统时钟的二分频值。

SPI_FrameFormat: 该参数用选择 SPI 的数据传输格式。错误!未找到引用源。给出了该参数可取的值。

表 12-15 SPI_FrameFormat 值

SPI_FrameFormat	描述
SPI_Frame_Motorola	Motorola 传输格式
SPI_Frame_TI_SSP	TI 传输格式
SPI_Frame_NS_MICROWIRE	保留
SPI_Frame_Reserve	保留

SPI_TxThresholdLevel: 设置发送 FIFO 的阈值，超过该阈值会触发 SPI_INT_TXE 中断。

SPI_RxThresholdLevel: 设置接收 FIFO 的阈值，如果接收 FIFO 中数据个数大于该阈值，会触发 SPI_INT_RXF 中断。

SPI_NDF: 设置 SPI 在 EEPROM 模式下，读取的数据长度阈值，该阈值为需要读取的数据长度减一。在其他传输模式下，该参数无效。例如，SPI 在 EEPROM 模式下读取 256 个数据，该参数应设置为 255。

SPI_RxDmaEn: 使能或者失能 DMA 数据接收功能。错误!未找到引用源。给出了该参数可取的值。

表 12-16 SPI_RxDmaEn 值

SPI_RxDmaEn	描述
DISABLE	失能 DMA 数据接收功能
ENABLE	使能 DMA 数据接收功能

SPI_TxDmaEn: 使能或者失能 DMA 数据发送功能。错误!未找到引用源。给出了该参数可取的值。

表 12-17 SPI_TxDmaEn 值

SPI_TxDmaEn	描述
DISABLE	失能 DMA 数据发送功能
ENABLE	使能 DMA 数据发送功能

SPI_RxWaterlevel: 设置 DMA 接收时的 water level 值, 该参数建议值为 DMA 的 Msize。

SPI_TxWaterlevel: 设置 DMA 发送时的 water level 值, 该参数建议值为发送 FIFO 的深度减去 DMA 的 Msize。

例:

```
/* Initialize SPI */
SPI_InitTypeDef SPI_InitStructure;
SPI_InitStructure.SPI_Direction = SPI_Direction_FullDuplex;
SPI_InitStructure.SPI_Mode = SPI_Mode_Master;
SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;
SPI_InitStructure.SPI_CPOL = SPI_CPOL_High;
SPI_InitStructure.SPI_CPHA = SPI_CPHA_1Edge;
SPI_InitStructure.SPI_BaudRatePrescaler = 4;
/* cause SPI_INT_RXF interrupt if data length in receive FIFO >= SPI_RxThresholdLevel + 1*/
SPI_InitStructure.SPI_RxThresholdLevel = 0;
SPI_InitStructure.SPI_FrameFormat = SPI_Frame_Motorola;
SPI_Init(SPI0, &SPI_InitStructure);
```

12.2.3 函数 SPI_StructInit

表 12-18 函数 SPI_StructInit

函数名	SPI_StructInit
函数原型	void SPI_StructInit(SPI_InitTypeDef* SPI_InitStruct)
功能描述	根据 SPI_InitStruct 中指定参数初始化 SPI 寄存器
输入参数 1	SPIx: x 可以设置成 0 或者 1, 选择指定的 SPI 外设
输入参数 2	SPI_InitStruct: 指向结构 SPI_InitTypeDef 的指针, 包含了外设 SPI 的配置信息
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Initialize SPI */
```

```
SPI_InitTypeDef SPI_InitStructure;  
SPI_StructInit(&SPI_InitStructure);
```

12.2.4 函数 SPI_Cmd

表 12-19 函数 SPI_Cmd

函数名	SPI_Cmd
函数原型	void SPI_Cmd(SPI_TypeDef* SPIx, FunctionalState NewState)
功能描述	使能或者失能指定的 SPI 外设模块
输入参数 1	SPIx: x 可以设置成 0 或者 1, 选择指定的 SPI 外设
输入参数 2	NewState: 外设 SPIx 的新状态 这个参数可以取: ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Enable SPI */  
SPI_Cmd(SPI0, ENABLE);
```

12.2.5 函数 SPI_SendBuffer

表 12-20 函数 SPI_SendBuffer

函数名	SPI_SendBuffer
函数原型	void SPI_SendBuffer(SPI_TypeDef* SPIx, uint8_t *pBuf, uint16_t len)
功能描述	通过 SPI 外设发送数据
输入参数 1	SPIx: x 可以设置成 0 或者 1, 选择指定的 SPI 外设
输入参数 2	pBuf: 指向发送数据的指针
输入参数 3	len: 数据长度
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Send data */  
uint8_t SPI_WriteBuf[16] = {0x00,0x01,0x02,0x03};  
SPI_SendBuffer(SPI0, SPI_WriteBuf, 4);
```

12.2.6 函数 SPI_SendWord

表 12-21 函数 SPI_SendWord

函数名	SPI_SendWord
函数原型	void SPI_SendWord(SPI_TypeDef *SPIx, uint32_t *pBuf, uint16_t len)
功能描述	通过 SPI 外设发送四字节数据
输入参数 1	SPIx: x 可以设置成 0 或者 1, 选择指定的 SPI 外设
输入参数 2	pBuf: 指向发送数据的指针
输入参数 3	len: 数据长度
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Send data */  
uint32_t SPI_WriteBuf[2] = {0x22446688,0x11335577};  
SPI_SendWord (SPI0, SPI_WriteBuf, 4);
```

12.2.7 函数 SPI_SendHalfWord

表 12-22 函数 SPI_SendHalfWord

函数名	SPI_SendHalfWord
函数原型	void SPI_SendHalfWord(SPI_TypeDef *SPIx, uint16_t *pBuf, uint16_t len)
功能描述	通过 SPI 外设发送二字节数据
输入参数 1	SPIx: x 可以设置成 0 或者 1, 选择指定的 SPI 外设
输入参数 2	pBuf: 指向发送数据的指针
输入参数 3	len: 数据长度
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Send data */  
uint32_t SPI_WriteBuf[2] = {0x6688,0x5577};  
SPI_SendHalfWord (SPI0, SPI_WriteBuf, 4);
```

12.2.8 函数 SPI_INTConfig

表 12-23 函数 SPI_INTConfig

函数名	SPI_INTConfig
函数原型	void SPI_INTConfig(SPI_TypeDef* SPIx, uint8_t SPI_IT, FunctionalState NewState)
功能描述	使能或者失能 SPI 指定的中断源
输入参数 1	SPIx: x 可以设置成 0 或者 1, 选择指定的 SPI 外设
输入参数 2	SPI_IT: 待使能或者失能的 SPI 中断源, 具体参阅 SPI_IT 介绍部分
输入参数 3	NewState: 中断的新状态 这个参数可以去 ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

SPI_IT: 允许的 SPI 中断类型如错误!未找到引用源。所示, 使用操作符“|”可以一次选中多个中断作为该参数的值。

表 12-24 SPI_IT 值

SPI_IT	描述
SPI_INT_TXE	发送缓冲区为空中断
SPI_INT_TXO	发送缓冲区溢出中断
SPI_INT_RXU	接收缓冲区 underflow 中断
SPI_INT_RXO	接收缓冲区溢出中断
SPI_INT_RXF	接收缓冲区已满中断
SPI_INT_MST	保留
SPI_INT_TUF	发送缓冲区 underflow 中断
SPI_INT_RIG	片选信号上升沿中断

例:

```
/* Enable receive FIFO full interrupt */
SPI_INTConfig(SPI0, SPI_INT_RXF, ENABLE);
```

12.2.9 函数 SPI_ClearINTPendingBit

表 12-25 函数 SPI_ClearINTPendingBit

函数名	SPI_ClearINTPendingBit
函数原型	void SPI_ClearINTPendingBit(SPI_TypeDef* SPIx, uint16_t SPI_IT)
功能描述	清除挂起的 SPI 中断标志位
输入参数 1	SPIx: x 可以设置成 0 或者 1, 选择指定的 SPI 外设
输入参数 2	SPI_IT: 待清除的 SPI 中断源, 具体参阅错误!未找到引用源。 SPI_IT 部分的介绍
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Clear receive FIFO overflow interrupt */
```

```
SPI_ClearINTPendingBit(SPIO, SPI_INT_RXO);
```

12.2.10 函数 SPI_SendData

表 12-26 函数 SPI_SendData

函数名	SPI_SendData
函数原型	void SPI_SendData(SPI_TypeDef* SPIx, uint16_t Data)
功能描述	通过 SPI 外设传输数据
输入参数 1	SPIx: x 可以设置成 0 或者 1, 选择指定的 SPI 外设
输入参数 2	Data: 传输的数据
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Send data */  
SPI_SendData (SPIO, 0x66);
```

12.2.11 函数 SPI_ReceiveData

表 12-27 函数 SPI_ReceiveData

函数名	SPI_ReceiveData
函数原型	uint16_t SPI_ReceiveData(SPI_TypeDef* SPIx)
功能描述	SPI 外设接收到的数据
输入参数 1	SPIx: x 可以设置成 0 或者 1, 选择指定的 SPI 外设
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Receive data */  
uint8_t value = 0;  
value = SPI_ReceiveData (SPIO);
```

12.2.12 函数 SPI_GetRxFIFOLen

表 12-28 函数 SPI_GetRxFIFOLen

函数名	SPI_GetRxFIFOLen
函数原型	uint8_t SPI_GetRxFIFOLen(SPI_TypeDef* SPIx)

功能描述	从 SPI 接收 FIFO 中读取接收数据的长度
输入参数	SPIx: x 可以设置成 0 或者 1, 选择指定的 SPI 外设
输出参数	无
返回值	数据长度
先决条件	无
被调用函数	无

例：

```
/* Get data number in RX FIFO */
uint8_t  number = 0;
number = SPI_GetRxFIFOLen (SPI0);
```

12.2.13 函数 SPI_GetTxFIFOLen

表 12-29 函数 SPI_GetTxFIFOLen

函数名	SPI_GetTxFIFOLen
函数原型	uint8_t SPI_GetTxFIFOLen (SPI_TypeDef* SPIx)
功能描述	从 SPI 发送 FIFO 中读取发送数据的长度
输入参数 1	SPIx: x 可以设置成 0 或者 1, 选择指定的 SPI 外设
输出参数	无
返回值	数据长度
先决条件	无
被调用函数	无

例：

```
/* Get data number in Tx FIFO */
uint8_t  number = 0;
number = SPI_GetTxFIFOLen (SPI0);
```

12.2.14 函数 SPI_ChangeDirection

表 12-30 函数 SPI_ChangeDirection

函数名	SPI_ChangeDirection
函数原型	void SPI_ChangeDirection(SPI_TypeDef* SPIx, uint16_t dir)
功能描述	重新设置 SPI 的数据传输模式
输入参数 1	SPIx: x 可以设置成 0 或者 1, 选择指定的 SPI 外设
输出参数 2	dir:制定的数据传输模式
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Configure transmission mode */  
SPI_ChangeDirection (SPIO, SPI_Direction_EEPROM);
```

12.2.15 函数 SPI_SetReadLen

表 12-31 函数 SPI_SetReadLen

函数名	SPI_SetReadLen
函数原型	void SPI_SetReadLen(SPI_TypeDef* SPIx, uint16_t len)
功能描述	设置 SPI 在 EEPROM 模式下读取的数据长度
输入参数 1	SPIx: x 可以设置成 0 或者 1, 选择指定的 SPI 外设
输出参数 2	len:读取数据的长度
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Configure length of data which need to receive in EEPROM mode */  
SPI_SetReadLen (SPIO,256);
```

12.2.16 函数 SPI_SetCSNumber

表 12-32 函数 SPI_SetCSNumber

函数名	SPI_SetCSNumber
函数原型	void SPI_SetCSNumber(SPI_TypeDef* SPIx, uint8_t number)
功能描述	设置片选信号索引号。如果选择外设为 SPIO, number 的取值固定为 0; 如果选择外设为 SPI1, number 的取值范围为 0、1 或者 2。
输入参数 1	SPIx: x 可以设置成 0 或者 1, 选择指定的 SPI 外设
输出参数 2	number:片选信号
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Configure CS number */  
SPI_SetCSNumber (SPI1,2);
```

12.2.17 函数 SPI_GetFlagState

表 12-33 函数 SPI_GetFlagState

函数名	SPI_GetFlagState
函数原型	FlagStatus SPI_GetFlagState(SPI_TypeDef* SPIx, uint8_t SPI_FLAG)
功能描述	检查指定的 SPI 外设标志位
输入参数 1	SPIx: x 可以设置成 0 或者 1, 选择指定的 SPI 外设
输出参数 2	SPI_FLAG:待指定的标志, 具体参阅 SPI_FLAG 介绍部分
输出参数	无
返回值	SPI 指定标志的新状态(SET 或者 RESET)
先决条件	无
被调用函数	无

SPI_FLAG: 允许的 SPI 状态类型如错误!未找到引用源。所示。

表 12-34 SPI_FLAG 值

SPI_IT	描述
SPI_FLAG_DCOL	数据传输错误, 该错误发生在当前外设处于主设备模式下, 被外部设备选择为从设备
SPI_FLAG_TXE	发送错误, 该错误发生在当前外设处于从设备下, 在发送 FIFO 为空下发送数据
SPI_FLAG_RFF	接收 FIFO 已满
SPI_FLAG_RFNE	接收 FIFO 不为空
SPI_FLAG_TFE	发送 FIFO 为空
SPI_FLAG_TFNF	发送 FIFO 未满
SPI_FLAG_BUSY	SPI 处于传输数据状态

例:

```
/* Check RX FIFO full or not */
SPI_GetFlagState (SPI0, SPI_FLAG_RFF);
```

12.2.18 函数 SPI_GetINTStatus

表 12-35 函数 SPI_GetINTStatus

函数名	SPI_GetINTStatus
函数原型	ITStatus SPI_GetINTStatus(SPI_TypeDef* SPIx, uint32_t SPI_IT)
功能描述	检查指定的 SPI 外设中断源
输入参数 1	SPIx: x 可以设置成 0 或者 1, 选择指定的 SPI 外设
输出参数 2	SPI_IT:待指定的 SPI 外设中断源, 具体参阅错误!未找到引用源。 SPI_IT 介绍部分
输出参数	无
返回值	中断源状态
先决条件	无
被调用函数	无

```
/* Check receive FIFO full or not in SPI interrupt handle function */
```

```
BOOL isOccur = FALSE;
isOccur = SPI_GetINTStatus(SPI0, SPI_INT_RXF)
```

12.2.19 函数 SPI_GDMACmd

表 12-36 函数 SPI_GDMACmd

函数名	SPI_GDMACmd
函数原型	void SPI_GDMACmd(SPI_TypeDef* SPIx, uint16_t SPI_GDMAReq, FunctionalState NewState)
功能描述	设置 SPI 传输过程中 GDMA 请求类型
输入参数 1	SPIx: x 可以设置成 0 或者 1, 选择指定的 SPI 外设
输出参数 2	SPI_GDMAReq:待指定 GDMA 请求。具体参阅 SPI_GDMAReq 介绍部分
输入参数 3	NewState: GDMA 请求的新状态 可选参数: ENABLE(使能)或者 DISABLE(失能)
输出参数	无
返回值	无
先决条件	无
被调用函数	无

SPI_GDMAReq: 允许的数据传输类型如错误!未找到引用源。所示。

表 12-37 SPI_GDMAReq 值

SPI_GDMAReq	描述
SPI_GDMAReq_Tx	通过 SPI 外设发送数据
SPI_GDMAReq_Rx	通过 SPI 外设接收数据

例:

```
/* Configure GDMA request */  
SPI_GDMACmd (SPI0, SPI_GDMAReq_Tx,ENABLE);
```

13 三线 SPI(SPI3WIRE)

13.1 SPI3WIRE 寄存器结构

```
typedef struct
{
    __IO uint32_t RSVD0[12];
    __IO uint32_t CFGR;
    __IO uint32_t CR;
    __IO uint32_t INTCR;
    __I  uint32_t SR;
    __IO uint32_t RD0;
    __IO uint32_t RD1;
    __IO uint32_t RD2;
    __IO uint32_t RD3;
} SPI2WIRE_TypeDef;
```

错误!未找到引用源。例举 SPI3WIRE 所有寄存器

表 13-1 SPI3WIRE 寄存器

寄存器	描述
RSVD0[12]	保留
CFGReg	配置寄存器
CR	控制寄存器
INTCR	中断控制寄存器
SR	状态寄存器
RD0	接收数据寄存器 0
RD1	接收数据寄存器 1
RD2	接收数据寄存器 2
RD3	接收数据寄存器 3

13.2 SPI3WIRE 库函数

错误!未找到引用源。例举 SPI3WIRE 的所有库函数

表 13-2 SPI3WIRE 库函数

函数名	描述
SPI3WIRE_DeInit	关闭 SPI3WIRE 时钟源
SPI3WIRE_Init	根据 SPI3WIRE_InitStruct 中指定参数初始化 SPI3WIRE 寄存器
SPI3WIRE_StructInit	根据 SPI3WIRE_InitStruct 中指定参数初始化 SPI3WIRE 寄存器
SPI3WIRE_Cmd	使能或失能 SPI3WIRE
SPI3WIRE_SetResyncTime	设置 resync 信号的持续时间

SPI3WIRE_ResyncSignalCmd	使能或者失能 resync 信号输出
SPI3WIRE_INTConfig	使能或失能 SPI3WIRE 指定中断
SPI3WIRE_GetFlagStatus	检查 SPI3WIRE 指定的标志位状态
SPI3WIRE_ClearITPendingBit	清除 SPI3WIRE 的中断标志位
SPI3WIRE_GetRxDataLen	在 SPI3WIRE 每次读取操作中，获取接收数据的个数
SPI3WIRE_ClearRxFIFO	清除 SPI3WIRE 的接收 FIFO 中数据
SPI3WIRE_ClearRxDataLen	清除 SPI3WIRE 的接收数据个数
SPI3WIRE_StartWrite	通过 SPI3WIRE 发送数据
SPI3WIRE_StartRead	通过 SPI3WIRE 发送读取单个或者多个数据命令
SPI3WIRE_ReadBuf	通过 SPI3WIRE 读取接收 FIFO 中数据

13.2.1 函数 SPI3WIRE_DeInit

表 13-3 函数 SPI3WIRE_DeInit

函数名	SPI3WIRE_DeInit
函数原型	void SPI3WIRE_DeInit(void)
功能描述	关闭 SPI3WIRE 时钟源
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	RCC_PeriphClockCmd()

例：

```
/* Close SPI3WIRE clock */
SPI3WIRE_DeInit();
```

13.2.2 函数 SPI3WIRE_Init

表 13-4 函数 SPI3WIRE_Init

函数名	SPI3WIRE_Init
函数原型	void SPI3WIRE_Init(SPI3WIRE_InitTypeDef* SPI3WIRE_InitStruct)
功能描述	根据 SPI3WIRE_InitStruct 中指定参数初始化 SPI3WIRE 寄存器
输入参数	SPI3WIRE_InitStruct: 指向结构 SPI3WIRE_InitTypeDef 的指针，包含了外设 SPI3WIRE 的配置信息
输出参数	无
返回值	无
先决条件	无
被调用函数	无

typedef struct

{

```

uint32_t SPI3WIRE_SysClock;
uint32_t SPI3WIRE_Speed;
uint32_t SPI3WIRE_Mode;
uint32_t SPI3WIRE_ReadDelay;
uint32_t SPI3WIRE_OutputDelay;
uint32_t SPI3WIRE_ExtMode;
}SPI3WIRE_InitTypeDef;

```

SPI3WIRE_SysClock: 该参数固定为系统时钟。

SPI3WIRE_Speed: 该参数配置 SPI3WIRE 的输出时钟。

SPI3WIRE_Mode: 该参数用以设置 SPI3WIRE 的工作模式。错误!未找到引用源。给出了该参数可取的值。

表 13-5 mode 值

mode	描述
SPI3WIRE_2WIRE_MODE	两线 SPI
SPI3WIRE_3WIRE_MODE	三线 SPI

SPI3WIRE_ReadDelay: 该参数用以控制 SPI3WIRE 读取数据时的延时时间。该参数的取值范围为 0x00 至 0x1f。

延时时间公式为 $\text{delay time} = (\text{SPI3WIRE_ReadDelay} + 1) / (2 * \text{SPI3WIRE_Speed})$

SPI3WIRE_OutputDelay: 该参数用以控制 SPI3WIRE 是否输出延时。错误!未找到引用源。给出了该参数可取的值。

表 13-6 SPI3WIRE_OE_DELAY_NONE 值

SPI3WIRE_OE_DELAY_NONE	描述
SPI3WIRE_OE_DELAY_1T	延时 1T
SPI3WIRE_OE_DELAY_NONE	不产生延时

SPI3WIRE_ExtMode: 该参数用以控制 SPI3WIRE 是否为拓展模式。错误!未找到引用源。给出了该参数可取的值。

表 13-7 SPI3WIRE_ExtMode 值

SPI3WIRE_ExtMode	描述
SPI3WIRE_EXTEND_MODE	扩展模式
SPI3WIRE_NORMAL_MODE	正常模式

例：

```

/* Initialize IR */

SPI3WIRE_InitTypeDef SPI3WIRE_InitStruct;
SPI3WIRE_StructInit(&SPI3WIRE_InitStruct);
SPI3WIRE_InitStruct.SPI3WIRE_SysClock      = 20000000;
SPI3WIRE_InitStruct.SPI3WIRE_Speed        = 1000000;
SPI3WIRE_InitStruct.SPI3WIRE_Mode         = SPI3WIRE_2WIRE_MODE;
/* delay time = (SPI3WIRE_ReadDelay + 1) / (2 * SPI3WIRE_Speed).
The delay time from the end of address phase to the start of read data phase */

```

```
//delay time = (0x05 + 3)/(2 * speed) = 4us
SPI3WIRE_InitStruct.SPI3WIRE_ReadDelay      = 0x5;
SPI3WIRE_InitStruct.SPI3WIRE_OutputDelay     = SPI3WIRE_OE_DELAY_1T;
SPI3WIRE_InitStruct.SPI3WIRE_ExtMode        = SPI3WIRE_NORMAL_MODE;
SPI3WIRE_Init(&SPI3WIRE_InitStruct);
```

13.2.3 函数 SPI3WIRE_StructInit

表 13-8 函数 SPI3WIRE_StructInit

函数名	SPI3WIRE_StructInit
函数原型	void SPI3WIRE_StructInit(SPI3WIRE_InitTypeDef* SPI3WIRE_InitStruct)
功能描述	把 SPI3WIRE_InitStruct 中的每一个参数按缺省值填入
输入参数	SPI3WIRE_InitStruct: 指向结构 SPI3WIRE_InitTypeDef 的指针, 包含了外设 SPI3WIRE 的配置信息
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Configure the SPI3WIRE Init Structure parameter */
SPI3WIRE_InitTypeDef  SPI3WIRE_InitStruct;
SPI3WIRE_StructInit(&SPI3WIRE_InitStruct);
```

13.2.4 函数 SPI3WIRE_Cmd

表 13-9 函数 SPI3WIRE_Cmd

函数名	SPI3WIRE_Cmd
函数原型	void SPI3WIRE_Cmd(FunctionalState NewState)
功能描述	使能或失能 SPI3WIRE
输入参数	newState: SPI3WIRE 的新状态 可选参数: ENABLE(使能)或者 DISABLE(失能)
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Enable the SPI3WIRE */
SPI3WIRE_Cmd(ENABLE);
```

13.2.5 函数 SPI3WIRE_SetResyncTime

表 13-10 函数 SPI3WIRE_SetResyncTime

函数名	SPI3WIRE_SetResyncTime
函数原型	void SPI3WIRE_SetResyncTime(uint32_t value)
功能描述	设置 resync 信号的持续时间
输入参数	value: resync 信号的持续时间值 该参数的取值范围为 0x0 to 0xf, 单位为 $1/(2*\text{SPI3WIRE_Speed})$
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Send resync time. Resync signal time = 2*1/(2*SPI3WIRE_Speed) = 1us */
SPI3WIRE_SetResyncTime(2);
```

13.2.6 函数 SPI3WIRE_ResyncSignalCmd

表 13-11 函数 SPI3WIRE_ResyncSignalCmd

函数名	SPI3WIRE_ResyncSignalCmd
函数原型	void SPI3WIRE_ResyncSignalCmd(FunctionalState NewState)
功能描述	使能或者失能 resync 信号输出
输入参数	newState: resync 信号输出的新状态 可选参数：ENABLE(使能)或者 DISABLE(失能)
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable resync output */
SPI3WIRE_ResyncSignalCmd(ENABLE);
```

13.2.7 函数 SPI3WIRE_INTConfig

表 13-12 函数 SPI3WIRE_INTConfig

函数名	SPI3WIRE_INTConfig
函数原型	void SPI3WIRE_INTConfig(uint32_t SPI3WIRE_INT, FunctionalState newState)
功能描述	使能或失能 SPI3WIRE 指定中断
输入参数 1	SPI3WIRE_IT: 待使能或者失能的 SPI3WIRE 中断, 该参数可取值固定为 SPI3WIRE_INT_BIT

输入参数 2	newState: SPI3WIRE 中断的新状态 可选参数: ENABLE(使能)或者 DISABLE(失能)
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Enable the SPI3WIRE interrupt */
SPI3WIRE_INTConfig(SPI3WIRE_INT_BIT, ENABLE);
```

13.2.8 函数 SPI3WIRE_GetFlagStatus

表 13-13 函数 SPI3WIRE_GetFlagStatus

函数名	SPI3WIRE_GetFlagStatus
函数原型	FlagStatus SPI3WIRE_GetFlagStatus(uint32_t SPI3WIRE_FLAG)
功能描述	检查 SPI3WIRE 指定的标志位状态
输入参数	SPI3WIRE_FLAG: 待检查的 SPI3WIRE 标志位 参数 SPI3WIRE_FLAG_BUSY: SPI3WIRE 处于 busy 状态标志位 参数 SPI3WIRE_FLAG_INT_IND: SPI3WIRE 产生中断标志位 参数 SPI3WIRE_FLAG_RESYNC_BUSY: Resync 信号正在输出标志位
输出参数	无
返回值	SPI3WIRE_FLAG 的新状态 (SET 或者 RESET)
先决条件	无
被调用函数	无

例:

```
/* Check SPI3WIRE busy or not */
BOOL flagStatus = FALSE;
flagStatus =SPI3WIRE_GetFlagStatus (SPI3WIRE_FLAG_BUSY);
```

13.2.9 函数 SPI3WIRE_ClearINTPendingBit

表 13-14 函数 SPI3WIRE_ClearINTPendingBit

函数名	SPI3WIRE_ClearINTPendingBit
函数原型	void SPI3WIRE_ClearINTPendingBit(uint32_t SPI3WIRE_INT)
功能描述	清除 SPI3WIRE 的中断标志位
输入参数	SPI3WIRE_INT: 待清除的 SPI3WIRE 中断, 该参数可取值固定为 SPI3WIRE_INT_BIT
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear the SPI3WIRE interrupt */  
SPI3WIRE_ClearITPendingBit (SPI3WIRE_INT_BIT);
```

13.2.10 函数 SPI3WIRE_GetRxDataLen

表 13-15 函数 SPI3WIRE_GetRxDataLen

函数名	SPI3WIRE_GetRxDataLen
函数原型	GetRxDataLen
功能描述	在 SPI3WIRE 每次读取操作中，获取接收数据的个数
输入参数	无
输出参数	无
返回值	接收 FIFO 的数据个数
先决条件	无
被调用函数	无

例：

```
/* Get receive data length of SPI3WIRE */  
uint16_t receive_length = SPI3WIRE_GetRxDataLen();
```

13.2.11 函数 SPI3WIRE_ClearRx FIFO

表 13-16 函数 SPI3WIRE_ClearRx FIFO

函数名	SPI3WIRE_ClearRx FIFO
函数原型	void SPI3WIRE_ClearRx FIFO(void)
功能描述	清除 SPI3WIRE 的接收 FIFO 中数据
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear SPI3WIRE RX FIFO */  
SPI3WIRE_ClearRx FIFO();
```

13.2.12 函数 SPI3WIRE_ClearRxDataLen

表 13-17 函数 SPI3WIRE_ClearRxDataLen

函数名	SPI3WIRE_ClearRxDataLen
函数原型	void SPI3WIRE_ClearRxDataLen(void)

功能描述	清除 SPI3WIRE 的接收数据个数
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear number of receive data */
SPI3WIRE_ClearRxDataLen();
```

13.2.13 函数 SPI3WIRE_StartWrite

表 13-18 函数 SPI3WIRE_StartWrite

函数名	SPI3WIRE_StartWrite
函数原型	void SPI3WIRE_StartWrite(uint8_t address, uint8_t data)
功能描述	通过 SPI3WIRE 发送数据
输入参数 1	address: 需要写的地址
输入参数 2	data: 需要写的数据
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Send data */
SPI3WIRE_StartWrite(0x00, 0x66);
```

13.2.14 函数 SPI3WIRE_StartRead

表 13-19 函数 SPI3WIRE_StartRead

函数名	SPI3WIRE_StartRead
函数原型	void SPI3WIRE_StartRead(uint8_t address, uint32_t len)
功能描述	通过 SPI3WIRE 发送读取单个或者多个数据命令
输入参数 1	address: 需要读的地址
输入参数 2	len: 需要读取数据的个数
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Send read several data command */
```

```
SPI3WIRE_StartRead(0x20, 1);
```

13.2.15 函数 SPI3WIRE_ReadBuf

表 13-20 函数 SPI3WIRE_ReadBuf

函数名	SPI3WIRE_ReadBuf
函数原型	void SPI3WIRE_ReadBuf(uint8_t *pBuf, uint8_t readNum)
功能描述	通过 SPI3WIRE 读取接收 FIFO 中数据
输入参数 1	pBuf: 存放数据的 buffer 首地址
输入参数 2	readNum: 需要读取的数据长度
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Read data */  
uint8_t buffer[2] = {0, 0};  
uint8_t readLen = 2;  
SPI3WIRE_ReadBuf (buffer, readLen);
```

14 定时器和脉冲宽度调制(TIM&PWM)

14.1 TIM 寄存器结构

```
typedef struct
{
    __IO uint32_t LoadCount;
    __I  uint32_t CurrentValue;
    __IO uint32_t ControlReg;
    __I  uint32_t EOI;
    __I  uint32_t IntStatus;
} TIM_TypeDef;
```

错误!未找到引用源。例举 TIM 所有寄存器

表 14-1 TIM 寄存器

寄存器	描述
LoadCount	加载值寄存器
CurrentValue	当前计数寄存器
ControlReg	控制寄存器
EOI	中断清除寄存器
IntStatus	中断状态寄存器

14.2 TIM 库函数

错误!未找到引用源。例举 TIM 的所有库函数

表 14-2 TIM 库函数

函数名	描述
TIM_DeInit	关闭 TIM 时钟源
TIM_TimeBaseInit	根据 TIM_TimeBaseInitStruct 中指定参数初始化 TIMx 寄存器
TIM_StructInit	把 TIM_TimeBaseInitStruct 中的每一个参数按缺省值填入
TIM_Cmd	使能或失能 TIMx
TIM_ChangePeriod	重新设置 TIMx 的周期值
TIM_INTConfig	使能或失能 TIMx 中断
TIM_GetCurrentValue	获取 TIMx 当前计数值
TIM_GetINTStatus	检查 TIMx 中断标志位设置与否
TIM_ClearINT	清除 TIMx 中断
TIM_PWMChangeFreqAndDuty	修改 TIMx 对应的 PWM 频率和占空比
PWM_Deadzone_EMStop	双路互补输出 PWM 停止

14.2.1 函数 TIM_DeInit

表 14-3 函数 TIM_DeInit

函数名	TIM_DeInit
函数原型	void TIM_DeInit(void)
功能描述	关闭 TIM 时钟源
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	RCC_PeriphClockCmd()

例：

```
/* Close TIM clock */
TIM_DeInit();
```

14.2.2 函数 TIM_TimeBaseInit

表 14-4 函数 TIM_TimeBaseInit

函数名	TIM_TimeBaseInit
函数原型	void TIM_TimeBaseInit(TIM_TypeDef* TIMx, TIM_TimeBaseInitTypeDef* TIM_TimeBaseInitStruct)
功能描述	根据 TIM_TimeBaseInitStruct 中指定参数初始化 TIMx 寄存器
输入参数 1	TIMx: 来选择 TIMx 外设, x 可取值为 2 到 7
输入参数 2	TIM_TimeBaseInitStruct: 指向结构 TIM_TimeBaseInitTypeDef 的指针, 包含了外设 TIMx 的配置信息
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM_TimeBaseInitTypeDef structure

```
typedef struct
{
    uint16_t TIM_SOURCE_DIV;
    uint16_t TIM_Mode;
    uint16_t TIM_PWM_En;
    uint32_t TIM_Period;
    uint32_t TIM_PWM_High_Count;
    uint32_t TIM_PWM_Low_Count;
    uint8_t ClockDepend;
    uint32_t PWM_Deazone_Size;
```

```

    uint16_t PWMDeadZone_En;
    uint16_t PWM_Stop_State_P;
    uint16_t PWM_Stop_State_N;
} TIM_TimeBaseInitTypeDef;

```

TIM_SOURCE_DIV: 该参数用选择 TIMx 的时钟源。错误!未找到引用源。给出了该参数可取的值。

表 14-5 TIM_SOURCE_DIV 值

TIM_SOURCE_DIV	描述
TIM_CLOCK_DIVIDER_1	40M 时钟源
TIM_CLOCK_DIVIDER_2	20M 时钟源
TIM_CLOCK_DIVIDER_4	10M 时钟源
TIM_CLOCK_DIVIDER_8	5M 时钟源
TIM_CLOCK_DIVIDER_40	1M 时钟源

TIM_Mode: 该参数用选择 TIMx 的运行模式。错误!未找到引用源。给出了该参数可取的值。

表 14-6 TIM_Mode 值

TIM_Mode	描述
TIM_Mode_FreeRun	自由运行模式
TIM_Mode_UserDefine	用户定义模式

TIM_Period: 该参数用于设置 TIMx 的周期值。该参数取值范围为 0x00 至 0xFFFFFFFF。

TIM_PWM_En: 该参数用选择 TIMx 的 PWM 模式是否运行。错误!未找到引用源。给出了该参数可取的值。

注明：每个 TIM 都可以配置成 PWM 模式，但是只有 TIM2 和 TIM3 的 PWM 有互补输出功能。

表 14-7 TIM_Mode 值

TIM_Mode	描述
PWM_ENABLE	开启 PWM 模式
PWM_DISABLE	关闭 PWM 模式

TIM_PWM_High_Count: 该参数用于设置 TIMx 对应的 PWM 高电平部分的周期值。该参数取值范围为 0x00 至 0xFFFFFFFF。

TIM_PWM_Low_Count: 该参数用于设置 TIMx 对应的 PWM 低电平部分的周期值。该参数取值范围为 0x00 至 0xFFFFFFFF。

PWMDeadZone_En: 该参数用于配置双路 PWM 互补输出(带死区功能)功能。注明：只有 TIM2 和 TIM3 的 PWM 模式才有互补输出功能。

表 14-8 PWMDeadZone_En 值

TIM_EventMode	描述
DEADZONE_ENABLE	开启
DEADZONE_DISABLE	关闭

PWM_Deazone_Size: 该参数用于配置双路互补输出死区的大小。该参数取值范围为 0x00 至 0xFF

PWM_Stop_State_P: 该参数用于配置双路互补输出急停后 P 通道的电平状态。错误!未找到引用源。给出了该参数可取的值。

表 14-9 PWM_Stop_State_P 值

TIM_EventDuration	描述
PWM_STOP_AT_HIGH	急停后 P 通道停在高电平
PWM_STOP_AT_LOW	急停后 P 通道停在低电平

PWM_Stop_State_N: 该参数用于配置双路互补输出急停后 N 通道的电平状态。错误!未找到引用源。给出了该参数可取的值。

表 14-10 PWM_Stop_State_N 值

TIM_EventDuration	描述
PWM_STOP_AT_HIGH	急停后 N 通道停在高电平
PWM_STOP_AT_LOW	急停后 N 通道停在低电平

例：

```

/* Initialize TIM */
TIM_TimeBaseInitTypeDef TIM_InitStruct;
TIM_InitStruct.TIM_ClockSrc = TIM_CLOCK_10MHZ;
TIM_InitStruct.TIM_Period = 1000*10000 -1;
TIM_InitStruct.TIM_Mode = TIM_Mode_UserDefine;
TIM_InitStruct.TIM_EventMode = FALSE;
TIM_TimeBaseInit(TIM2, &TIM_InitStruct);

/* Initialize PWM */
TIM_StructInit(&TIM_InitStruct);
TIM_InitStruct.TIM_Mode = TIM_Mode_UserDefine;
TIM_InitStruct.TIM_PWM_En = PWM_ENABLE;
TIM_InitStruct.TIM_Period = 2000 * 10 - 1 ;
TIM_InitStruct.TIM_PWM_High_Count = 1000000 - 1 ;
TIM_InitStruct.TIM_PWM_Low_Count = 1000000 - 1 ;
TIM_InitStruct.TIM_Mode = 1;
TIM_TimeBaseInit(TIM2, &TIM_InitStruct);

```

14.2.3 函数 TIM_StructInit

表 14-11 函数 TIM_StructInit

函数名	TIM_StructInit
函数原型	void TIM_StructInit(TIM_TimeBaseInitTypeDef* TIM_TimeBaseInitStruct)
功能描述	把 TIM_TimeBaseInitStruct 中的每一个参数按缺省值填入

输入参数	TIM_TimeBaseInitStruct: 指向结构 TIM_TimeBaseInitStruct 的指针, 包含了外设 TIMx 的配置信息
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Configure the TIMx Init Structure parameter */
TIM_TimeBaseInitTypeDef TIM_InitStruct;
TIM_StructInit(&TIM_InitStruct);
```

14.2.4 函数 **TIM_Cmd**

表 14-12 函数 **TIM_Cmd**

函数名	TIM_Cmd
函数原型	void TIM_Cmd(TIM_TypeDef* TIMx, FunctionalState NewState)
功能描述	使能或失能指定 TIM
输入参数 1	TIMx: 来选择 TIMx 外设, x 可取值为 2 到 7
输入参数 2	NewState: TIMx 的新状态 可选参数: ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Enable TIM2 */
TIM_Cmd (TIM2, ENABLE);
```

14.2.5 函数 **TIM_ChangePeriod**

表 14-13 函数 **TIM_ChangePeriod**

函数名	TIM_ChangePeriod
函数原型	void TIM_ChangePeriod(TIM_TypeDef* TIMx, uint32_t period)
功能描述	重新设置 TIM 的周期值
输入参数 1	TIMx: 来选择 TIMx 外设, x 可取值为 2 到 7
输入参数 2	period: TIMx 的周期值
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Change TIM2 period value */  
TIM_ChangePeriod (TIM2, 1000*10000 -1);
```

14.2.6 函数 TIM_INTConfig

表 14-14 函数 TIM_INTConfig

函数名	TIM_INTConfig
函数原型	void TIM_INTConfig(TIM_TypeDef* TIMx, FunctionalState NewState)
功能描述	使能或失能 TIM 中断
输入参数 1	TIMx: 来选择 TIMx 外设, x 可取值为 2 到 7
输入参数 2	NewState: TIMx 中断的新状态 可选参数: ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable TIM2 interrupt */  
TIM_INTConfig (TIM2, ENABLE);
```

14.2.7 函数 TIM_GetCurrentValue

表 14-15 函数 TIM_GetCurrentValue

函数名	TIM_GetCurrentValue
函数原型	uint32_t TIM_GetCurrentValue(TIM_TypeDef* TIMx)
功能描述	获取 TIMx 的当前计数值
输入参数	TIMx: 来选择 TIMx 外设, x 可取值为 2 到 7
输出参数	无
返回值	TIMx 当前的计数值
先决条件	无
被调用函数	无

例：

```
/* Get TIM2 current value of counter */  
uint32_t value =TIM_GetCurrentValue (TIM2);
```

14.2.8 函数 TIM_GetINTStatus

表 14-16 函数 TIM_GetINTStatus

函数名	TIM_GetINTStatus
函数原型	ITStatus TIM_GetINTStatus(TIM_TypeDef* TIMx)
功能描述	检查 TIMx 中断标志位设置与否
输入参数	TIMx: 来选择 TIMx 外设, x 可取值为 2 到 7
输出参数	无
返回值	TIMx 中断标志位的值, 该值为枚举 ITStatus 的其中一个值。 RESET: 未产生中断 SET: 产生中断
先决条件	无
被调用函数	无

例:

```
/* Get TIM2 interrupt status */
BOOL intValue =TIM_GetINTStatus (TIM2);
```

14.2.9 函数 TIM_ClearINT

表 14-17 函数 TIM_ClearINT

函数名	TIM_ClearINT
函数原型	void TIM_ClearINT(TIM_TypeDef* TIMx)
功能描述	清除 TIMx 中断
输入参数	TIMx: 来选择 TIMx 外设, x 可取值为 2 到 7
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Clear TIM2 interrupt status */
TIM_ClearINT (TIM2);
```

14.2.10 函数 TIM_PWMChangeFreqAndDuty

表 14-18 函数 TIM_PWMChangeFreqAndDuty

函数名	TIM_PWMChangeFreqAndDuty
函数原型	void TIM_PWMChangeFreqAndDuty(TIM_TypeDef *TIMx, uint32_t high_count, uint32_t low_count)
功能描述	修改 TIMx 对应的 PWM 的频率和占空比
输入参数 1	TIMx: 来选择 TIMx 外设, x 可取值为 2 到 7
输入参数 2	high_count: PWM 单周期高电平加载值
输入参数 3	low_count: PWM 单周期低电平加载值
输出参数	无

返回值	无
先决条件	无
被调用函数	无

例：

```
/* Change TIM2 PWM high count and low count */  
TIM_PWMChangeFreqAndDuty (TIM2,0x100, 0x100);
```

14.2.11 函数 PWM_Deadzone_EMStop

表 14-19 函数 PWM_Deadzone_EMStop

函数名	PWM_Deadzone_EMStop
函数原型	void PWM_Deadzone_EMStop(PWM_TypeDef *PWMD)
功能描述	双路互补输出 PWM 急停
输入参数	PWMD：双路互补输出 PWM 外设，可取值为 PWM2 或 PWM3
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Stop PWM2 output */  
PWM_Deadzone_EMStop(PWM2);
```

15 通用异步接收发送端(UART)

15.1 UART 寄存器架构

```
typedef struct {
    __IO  uint32_t  DLL;
    __IO  uint32_t  DLH_INTCR;
    __IO  uint32_t  INTID_FCR;
    __IO  uint32_t  LCR;
    __IO  uint32_t  MCR;
    __I   uint32_t  LSR;
    __I   uint32_t  MSR;
    __IO  uint32_t  SPR;
    __IO  uint32_t  STSR;
    __IO  uint32_t  RB_THR;
    __IO  uint32_t  MISCR;
} UART_TypeDef;
```

错误!未找到引用源。例举 UART 所有寄存器

表 15-1 UART 寄存器

寄存器	描述
DLL	分频寄存器(L)
DLH_INTCR	分频寄存器(H)_中断控制寄存器
INTID_FCR	中断类型寄存器_FIFO 控制寄存器
LCR	Line 控制寄存器
MCR	Model 控制寄存器
LSR	Line 状态寄存器
MSR	Model 状态寄存器
SPR	暂存寄存器
STSR	
RB_THR	接收缓存寄存器/发送缓存寄存器
MISCR	混合寄存器

15.2 UART 库函数

表 15-2 例举 UART 的所有库函数

函数名	描述
UART_DeInit	关闭 UART 时钟
UART_Init	根据 UART_InitStruct 中指定参数初始化 UART 寄存器
UART_StructInit	把 UART_InitStruct 中的每一个值按照缺省值填入

UART_ReceiveData	通过 UART 接收数据
UART_SendData	通过 UART 发送数据
UART_INTConfig	使能或者失能 UART 指定类型的中断
UART_GetFlagState	检查 UART 外设中指定的状态标志位
UART_LoopBackCmd	使能或者失能 loop back 功能
UART_ClearTxFifo	清空 UART 发送 FIFO 中的数据
UART_ClearRxFifo	清空 UART 接收 FIFO 中的数据
UART_GetTxFIFOLen	从 UART 发送 FIFO 中读取数据的长度
UART_GetRxFIFOLen	从 UART 接收 FIFO 中读取数据的长度
UART_ReceiveByte	从 UART 的接收 FIFO 中读取一个字节
UART_SendByte	向 UART 的发送 FIFO 中写一个字节
UART_GetIID	获取 UART 中断源类型

15.2.1 函数 UART_DeInit

表 15-3 函数 UART_DeInit

函数名	UART_DeInit
函数原型	UART_DeInit(UART_TypeDef* UARTx)
功能描述	关闭 UART 时钟
输入参数 1	UARTx: 指向选择的 UART 模块, x 可设置为 0, 1 或者 2
输出参数	无
返回值	无
先决条件	无
被调用函数	RCC_PeriphClockCmd()

例:

```
/* Deinitialize Data UART */
UART_DeInit(UART);
```

15.2.2 函数 UART_Init

表 15-4 函数 UART_Init

函数名	UART_Init
函数原型	UART_Init(UART_TypeDef* UARTx, UART_InitTypeDef* UART_InitStruct)
功能描述	根据 UART_InitStruct 中指定参数初始化 UART 寄存器
输入参数 1	UARTx: 指向选择的 UART 模块, x 可设置为 0, 1 或者 2
输入参数 2	UART_InitStruct: 指向 UART_InitTypeDef 的指针, UART_InitStruct 中包含相关配置信息
输出参数	无
返回值	无
先决条件	无
被调用函数	无

```
typedef struct
{
    uint16_t ovsr_adj;
    uint16_t div;
    uint16_t ovsr;
    uint16_t wordLen;
    uint16_t parity;
    uint16_t stopBits;
    uint16_t autoFlowCtrl;
    uint16_t rxTriggerLevel;
    uint16_t dmaEn;
}UART_InitTypeDef;
```

ovsr_adj、div 以及 ovsr: 该三个参数设置 UART 的波特率校验参数。错误!未找到引用源。给出了该参数可取的值。

表 15-5 ovsr_adj 、div、ovsr 值与波特率关系表

ovsr_adj	div	ovsr	波特率值
0x7F7	2589	7	1200Hz
0x24A	271	10	9600Hz
0x222	271	5	14400Hz
0x5AD	165	7	19200Hz
0x5AD	110	7	28800Hz
0x222	85	7	38400Hz
0x5AD	55	7	57600Hz
0x7EF	35	9	76800Hz
0x252	20	12	115200Hz
0x555	25	7	128000Hz
0x252	15	12	153600Hz
0x252	10	12	230400Hz
0x252	5	12	460800Hz
0	8	5	500000Hz
0x3F7	4	5	921600Hz
0	4	5	1000000Hz
0x2AA	2	9	1382400Hz
0x5F7	2	8	1444400Hz
0x492	2	8	1500000Hz
0x3F7	2	5	1843200Hz
0	2	5	2000000Hz
0x400	1	14	2100000Hz
0x2AA	1	9	2764800Hz
0x492	1	8	3000000Hz
0x112	1	7	3250000Hz

0x5F7	1	5	3692300Hz
0x36D	1	5	3750000Hz
0	1	5	4000000Hz
0x36D	1	1	6000000Hz

parity: 设置 UART 的校验方式。错误!未找到引用源。给出了该参数可取的值。

表 15-6 parity 值

parity	描述
UART_PARITY_NO_PARTY	奇偶失能
UART_PARITY_ODD	偶模式
UART_PARITY_EVEN	奇模式

stopBits: 设置 UART 通信停止位的位数。错误!未找到引用源。给出了该参数可取的值。

表 15-7 stopBits 值

stopBits	描述
UART_STOP_BITS_1	在帧结尾传输 1 个停止位
UART_STOP_BITS_2	在帧结尾传输 2 个停止位

wordLen: 该参数设置 UART 的数据长度。错误!未找到引用源。给出了该参数可取的值。

表 15-8 wordLen 值

wordLen	描述
UART_WROD_LENGTH_7BIT	7 位数据
UART_WROD_LENGTH_8BIT	8 位数据

dmaEn: 设置是否使用 GDMA 传输模式。错误!未找到引用源。给出了该参数可取的值。

表 15-9 dmaEn 值

dmaEn	描述
UART_DMA_ENABLE	使能 UART 的 GDMA 的传输模式
UART_DMA_DISABLE	失能 UART 的 GDMA 的传输模式

autoFlowCtrl: 硬件流控制。错误!未找到引用源。给出了该参数可取的值。

表 15-10 autoFlowCtrl 值

autoFlowCtrl	描述
UART_AUTO_FLOW_CTRL_EN	使能硬件流控制
UART_AUTO_FLOW_CTRL_DIS	失能硬件流控制

rxTriggerLevel: 设置 UART 接收 FIFO 的阈值，该值取值范围为 0 到 29。

idle_time: 当 UART 接收 FIFO 的数据个数小于设置的阈值时，设置触发接收超时中断的超时时间值。错误!未找到引用源。给出了该参数可取的值。

表 15-11 idle_time 值

idle_time	描述
UART_RX_IDLE_1BYTE	当前波特率下接收 1 个字节的时间

UART_RX_IDLE_2BYTE	当前波特率下接收 2 个字节的时间
UART_RX_IDLE_4BYTE	当前波特率下接收 4 个字节的时间
UART_RX_IDLE_8BYTE	当前波特率下接收 8 个字节的时间
UART_RX_IDLE_16BYTE	当前波特率下接收 16 个字节的时间
UART_RX_IDLE_32BYTE	当前波特率下接收 32 个字节的时间
UART_RX_IDLE_64BYTE	当前波特率下接收 64 个字节的时间
UART_RX_IDLE_128BYTE	当前波特率下接收 128 个字节的时间
UART_RX_IDLE_256BYTE	当前波特率下接收 256 个字节的时间
UART_RX_IDLE_512BYTE	当前波特率下接收 512 个字节的时间
UART_RX_IDLE_1024BYTE	当前波特率下接收 1024 个字节的时间
UART_RX_IDLE_2048BYTE	当前波特率下接收 2048 个字节的时间
UART_RX_IDLE_4096BYTE	当前波特率下接收 4096 个字节的时间
UART_RX_IDLE_8192BYTE	当前波特率下接收 8192 个字节的时间
UART_RX_IDLE_16384BYTE	当前波特率下接收 16384 个字节的时间
UART_RX_IDLE_32768BYTE	当前波特率下接收 32768 个字节的时间

TxWaterlevel: 当 UART 通过 GDMA 发送数据时, 设置 UART 发送的 water level 值, 该参数取值范围为 1~16。

该参数推荐设置值 TxWaterlevel = 16 - GDMA_Msize。

RxWaterlevel: 当 UART 通过 GDMA 接收数据时, 设置 UART 接收的 water level 值, 该参数取值范围为 1~32。

该参数推荐设置值 RxWaterlevel = GDMA_Msize。

TxDmaEn: 使能或者失能 GDMA 发送功能。错误!未找到引用源。给出了该参数可取的值。

表 15-12 TxDmaEn 值

TxDmaEn	描述
ENABLE	使能 GDMA 发送功能
DISABLE	失能 GDMA 发送功能

RxDmaEn: 使能或者失能 GDMA 接收功能。错误!未找到引用源。给出了该参数可取的值。

表 15-13 RxDmaEn 值

RxDmaEn	描述
ENABLE	使能 GDMA 接收功能
DISABLE	失能 GDMA 接收功能

例:

```
/* Initialize Data UART */
UART_InitTypeDef  UART_InitStruct;
/* Configure baudrate to 115200 */
UART_InitStruct.div = 20;
UART_InitStruct.ovsr = 12;
UART_InitStruct.ovsr_adj = 0x252;
/* Configure uart parameters */
UART_InitStruct.parity = UART_PARITY_NO_PARTY;
```

```
UART_InitStruct.stopBits = UART_STOP_BITS_1;  
UART_InitStruct.wordLen = UART_WROD_LENGTH_8BIT;  
UART_InitStruct.dmaEn = UART_DMA_DISABLE;  
UART_InitStruct.autoFlowCtrl = UART_AUTO_FLOW_CTRL_DIS;  
UART_InitStruct.rxTriggerLevel = 16;  
UART_InitStruct.idle_time = UART_RX_IDLE_2BYTE;  
UART_InitStruct.TxWaterlevel = 15;  
UART_InitStruct.RxWaterlevel = 1;  
UART_InitStruct.TxDmaEn = DISABLE;  
UART_InitStruct.RxDmaEn = DISABLE;  
UART_Init(UART, &UART_InitStruct);
```

15.2.3 函数 **UART_StructInit**

表 15-14 函数 **UART_StructInit**

函数名	UART_StructInit
函数原型	void UART_StructInit(UART_InitTypeDef* UART_InitStruct)
功能描述	把 UART_InitStruct 中的每一个值按照缺省值填入
输入参数 1	UART_InitStruct: 指向结构体 UART_InitTypeDef 的指针, 待初始化
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Initialize Data UART by default value */  
UART_InitTypeDef  UART_InitStruct;  
UART_StructInit(&UART_InitStruct);
```

15.2.4 函数 **UART_ReceiveData**

表 15-15 函数 **UART_ReceiveData**

函数名	UART_ReceiveData
函数原型	void UART_ReceiveData(UART_TypeDef* UARTx, uint8_t* outBuf, uint16_t count)
功能描述	从 UART 接收 FIFO(接收 FIFO 的长度为 16)读取数据
输入参数 1	UARTx: 指向选择的 UART 模块, x 可设置为 0, 1 或者 2
输入参数 2	outBuf: 将接收到的数据存储到这个指针指向的空间
输入参数 3	Count: 接收的字节数
输出参数	无
返回值	无
先决条件	无

被调用函数	无
-------	---

例：

```
/* Receive data from RX FIFO of data uart */
uint8_t dataBuf[16] = {0};
UART_ReceiveData(UART, dataBuf, 16);
```

15.2.5 函数 UART_SendData

表 15-16 函数 UART_SendData

函数名	UART_SendData
函数原型	UART_SendData(UART_TypeDef* UARTx, uint8_t* inBuf, uint16_t count)
功能描述	通过 UART 往发送 FIFO(发送 FIFO 长度为 16)中写数据
输入参数 1	UARTx: 指向选择的 UART 模块, x 可设置为 0, 1 或者 2
输入参数 2	inBuf: 将这个指针指向的内容通过 UART 发送出去
输入参数 3	Count: 发送的字节数
输出参数	无
返回值	无
先决条件	无
被调用函数	无

```
/* Send data to TX FIFO of data uart */
uint8_t dataBuf[6] = {1,2,3,4,5,6};
UART_SendData(UART, dataBuf, 6);
```

15.2.6 函数 UART_INTConfig

表 15-17 函数 UART_INTConfig

函数名	UART_INTConfig
函数原型	void UART_INTConfig(UART_TypeDef* UARTx, uint32_t UART_IT, FunctionalState newState)
功能描述	使能或者失能 UART 指定类型的中断
输入参数 1	UARTx: 指向选择的 UART 模块, x 可设置为 0, 1 或者 2
输入参数 2	UART_IT: 待使能或者失能的 UART 中断源, 具体参阅 UART_IT 介绍部分
输入参数 3	NewState: 中断的新状态 这个参数可以去 ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

UART_IT: 允许的 UART 中断类型如错误!未找到引用源。所示, 使用操作符 “|” 可以一次选中多个中断作为该参数的值。

表 15-18 UART_IT 值

UART_IT	描述
UART_INT_RD_AVA	接收中断
UART_INT_FIFO_EMPTY	发送缓冲区为空中断
UART_INT_LINE_STS	UART 接收总线错误中断
UART_INT_MODEM_STS	Modem 状态中断
UART_INT_IDLE	UART 空闲中断

例：

```
/* Enable RX interrupt and line status interrupt */
UART_INTConfig(UART, UART_INT_RD_AVA | UART_INT_LINE_STS, ENABLE);
```

15.2.7 函数 **UART_GetFlagState**

表 15-19 函数 **UART_GetFlagStat**

函数名	UART_GetFlagStat
函数原型	FlagStatus UART_GetFlagState(UART_TypeDef* UARTx, uint32_t UART_FLAG)
功能描述	检查 UART 外设中指定的状态标志位
输入参数 1	UARTx: 指向选择的 UART 模块, x 可设置为 0, 1 或者 2
输入参数 2	UART_FLAG:待检测的标志类型
输出参数	无
返回值	无
先决条件	无
被调用函数	无

UART_FLAG: 允许的 UART 标志类型如错误!未找到引用源。所示。

表 15-20 **UART_FLAG** 值

UART_FLAG	描述
UART_FLAG_RX_DATA_RDY	接收到数据
UART_FLAG_RX_OVERRUN	溢出错误
UART_FLAG_PARITY_ERR	校验错误
UART_FLAG_FRAME_ERR	数据帧错误
UART_FLAG_BREAK_ERR	中断错误
UART_FLAG_THR_EMPTY	发送缓冲区为空
UART_FLAG_THR_TSR_EMPTY	发送缓冲区与移位寄存器均为空
UART_FLAG_RX_FIFO_ERR	校验错误或者数据帧错误或者中断错误

例：

```
/* Wait tx fifo empty */
while(UART_GetFlagState(UART, UART_FLAG_THR_TSR_EMPTY) != SET);
```

15.2.8 函数 **UART_ClearTxFifo**

表 15-21 函数 **UART_ClearTxFifo**

函数名	UART_ClearTxFifo
函数原型	void UART_ClearTxFifo(UART_TypeDef* UARTx)
功能描述	清空 UART 发送 FIFO 中的数据
输入参数 1	UARTx: 指向选择的 UART 模块, x 可设置为 0, 1 或者 2
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Clear Tx FIFO */
UART_ClearTxFifo(UART);
```

15.2.9 函数 **UART_LoopBackCmd**

表 15-22 函数 **UART_LoopBackCmd**

函数名	UART_LoopBackCmd
函数原型	void UART_LoopBackCmd(UART_TypeDef *UARTx, FunctionalState NewState)
功能描述	使能或者失能 loop back 功能
输入参数 1	UARTx: 指向选择的 UART 模块, x 可设置为 0, 1 或者 2
输入参数 2	NewState: Loop back 的新状态 这个参数可以取: ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Enable loop back function */
UART_LoopBackCmd (UART, ENABLE);
```

15.2.10 函数 **UART_ClearRxFifo**

表 15-23 函数 **UART_ClearRxFifo**

函数名	UART_ClearRxFifo
函数原型	void UART_ClearRxFifo(UART_TypeDef* UARTx)
功能描述	清空 UART 接收 FIFO 中的数据
输入参数 1	UARTx: 指向选择的 UART 模块, x 可设置为 0, 1 或者 2
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* ClearRx FIFO */
UART_ClearRxFifo(UART);
```

15.2.11 函数 **UART_GetTxFIFOLen**

表 15-24 函数 **UART_GetTxFIFOLen**

函数名	UART_GetTxFIFOLen
函数原型	<code>uint8_t UART_GetTxFIFOLen(UART_TypeDef *UARTx)</code>
功能描述	从 UART 发送 FIFO 中读取数据的长度
输入参数	UARTx: 指向选择的 UART 模块, x 可设置为 0, 1 或者 2
输出参数	无
返回值	数据长度
先决条件	无
被调用函数	无

例：

```
/* Get data number in TX FIFO */
uint8_t  number = 0;
number = UART_GetTxFIFOLen (UART);
```

15.2.12 函数 **UART_GetRxFIFOLen**

表 15-25 函数 **UART_GetRxFIFOLen**

函数名	UART_GetRxFIFOLen
函数原型	<code>uint8_t UART_GetRxFIFOLen(UART_TypeDef *UARTx)</code>
功能描述	从 UART 接收 FIFO 中读取数据的长度
输入参数	UARTx: 指向选择的 UART 模块, x 可设置为 0, 1 或者 2
输出参数	无
返回值	数据长度
先决条件	无
被调用函数	无

例：

```
/* Get data number in RX FIFO */
uint8_t  number = 0;
number = UART_GetRxFIFOLen (UART);
```

15.2.13 函数 **UART_ReceiveByte**

表 15-26 函数 **UART_ReceiveByte**

函数名	UART_ReceiveByte
函数原型	uint8_t UART_ReceiveByte(UART_TypeDef* UARTx)
功能描述	从 UART 的接收 FIFO 中读取一个字节
输入参数 1	UARTx: 指向选择的 UART 模块, x 可设置为 0, 1 或者 2
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Receive one byte from Rx FIFO of uart */
uint8_t data = UART_ReceiveByte (UART);
```

15.2.14 函数 UART_SendByte

表 15-27 函数 UART_SendByte

函数名	UART_SendByte
函数原型	void UART_SendByte(UART_TypeDef* UARTx, uint8_t data)
功能描述	向 UART 的发送 FIFO 中写一个字节
输入参数 1	UARTx: 指向选择的 UART 模块, x 可设置为 0, 1 或者 2
输入参数 2	data: 需要发送的数据
输出参数	无
返回值	无
先决条件	无
被调用函数	无

```
/* Send one byte to TX FIFO of data uart */
UART_SendData(UART, 0x66);
```

15.2.15 函数 UART_GetIID

表 15-28 函数 UART_GetIID

函数名	UART_GetIID
函数原型	uint16_t UART_GetIID(UART_TypeDef* UARTx)
功能描述	获取 UART 中断源类型
输入参数 1	UARTx: 指向选择的 UART 模块, x 可设置为 0, 1 或者 2
输出参数	无
返回值	中断类型
先决条件	无
被调用函数	无

例:

```
/* Read interrupt id */
```

```
uint32_t int_status = UART_GetIID(UART);
```

16 按键扫描(KEYSCAN)

16.1 KEYCAN 寄存器架构

```
typedef struct
{
    __IO uint32_t CLKDIV;
    __IO uint32_t TIMERCR;
    __IO uint32_t CR;
    __IO uint32_t COLCR;
    __IO uint32_t ROWCR;
    __I  uint32_t FIFODATA;
    __IO uint32_t INTMASK;
    __IO uint32_t INTCLR;
    __I  uint32_t STATUS;
} KEYSCAN_TypeDef;
```

错误!未找到引用源。例举 KEYSCAN 所有寄存器

表 16-1 KEYSCAN 寄存器

寄存器	描述
CLKDIV	时钟配置寄存器 0
TIMERCR	时钟控制寄存器 1
CR	控制寄存器
COLCR	列控制寄存器
ROWCR	行控制寄存器
FIFODATA	KEYSCAN 硬件 FIFO
INTMASK	中断屏蔽寄存器
STATUS	状态寄存器

16.2 KEYSCAN 库函数

表 16-2 例举 KEYSCAN 的所有库函数

函数名	描述
KeyScan_Init	根据 KeyScan_InitStruct 中指定参数初始化 KeyScan 寄存器
KeyScan_DeInit	关闭 KeyScan 时钟
KeyScan_StructInit	把 KeyScan_InitStruct 中的每一个值按照缺省值填入
KeyScan_INTConfig	使能或者失能 KeyScan 指定的中断源
KeyScan_INTMask	屏蔽 KeyScan 的中断源
KeyScan_Read	从 KeyScan 的 FIFO 中读取数据
KeyScan_Cmd	使能或者失能 KeyScan 外设

KeyScan_GetFifoDataNum	得到 FIFO 中数据的长度
KeyScan_ClearINTPendingBit	清除 KeyScan 外设某一中断源中断挂起位
KeyScan_ClearFlags	清除指定状态标志位
KeyScan_GetFlagState	检测指定的状态是否被置位
KeyScan_FilterDataConfig	KeyScan 数据过滤机制配置
KeyScan_debounceConfig	KeyScan 滤波配置
KeyScan_ReadFifoData	从 FIFO 中读取一个数据

16.2.1 函数 KeyScan_Init

表 16-3 函数 KeyScan_Init

函数名	KeyScan_Init
函数原型	void KeyScan_Init(KEYSCAN_TypeDef* KeyScan, KEYSCAN_InitTypeDef* KeyScan_InitStruct)
功能描述	根据 KeyScan_InitStruct 中指定参数初始化 KeyScan 寄存器
输入参数 1	KeyScan: 指向选择的 KeyScan 模块
输入参数 2	KeyScan_InitStruct: 指向 KeyScan_InitTypeDef 的指针, KeyScan_InitStruct 中包含相关配置信息
输出参数	无
返回值	无
先决条件	无
被调用函数	无

```

typedef struct
{
    uint16_t      rowSize;
    uint16_t      colSize;
    uint16_t      scanInterval;
    uint32_t      debounceEn;
    uint32_t      scantimerEn;
    uint32_t      detecttimerEn;
    uint16_t      debounceTime;
    uint32_t      detectMode;
    uint32_t      fifoOvrCtrl;
    uint16_t      maxScanData;
    uint32_t      scanmode;
    uint16_t      clockdiv;
    uint8_t       delayclk;
    uint16_t      fifottriggerlevel;
    uint8_t       debouncecnt;
    uint8_t       releasecnt;
    uint8_t       keylimit;
}KEYSCAN_InitTypeDef;

```

rowSize: 该参数设置矩阵键盘的行数，该参数的取值范围为 1 到 8。

colSize: 该参数设置矩阵键盘的列数，该参数的取值范围为 1 到 26。

clockdiv: 该参数设置矩阵键盘扫描时钟 scan_clock，该参数的取值范围为 1 到 0x7ff， $\text{scan_clock} = (\text{clockdiv} + 1)/5\text{M}$

delayclk: 该参数设定矩阵键盘硬件去抖/扫描间隔/按键释放检测时钟 delay_clock，该参数的取值范围为 1 到 0x3f， $\text{delay_clock} = (\text{delayclk} + 1)/\text{scan_clock}$

scanInterval: 该参数设置相邻扫描的时间间隔，此参数在长按某按键时起作用。如果长按，硬件会检测到，从而进行下一次按键扫描。该参数和 delayclk 配合生效，实际 scan interval time = scanInterval * delay_clock。

debounceEn: 设置是否开启去抖动功能。错误!未找到引用源。给出了该参数可取的值。

表 16-4 debounceEn 值

debounceEn	描述
KeyScan_Debounce_Enable	使能 keyscan 的去抖动功能
KeyScan_Debounce_Disable	失能 keyscan 的去抖动功能

debouncecnt: 设置 keyscan 的去抖动时间。实际 Debounce time = debouncecnt * delay_clock。

detectMode: 设置 keyscan 的运行模式。错误!未找到引用源。给出了该参数可取的值。

表 16-5 detectMode 值

detectMode	描述
KeyScan_Detect_Mode_Edge	边沿探测模式
KeyScan_Detect_Mode_Level	电平探测模式

releasecnt: 设置按键释放检测时间。实际 Release time = releasecnt * delay_clock。

detecttimerEn: 设置 keyscan 的按键释放检测模式。错误!未找到引用源。给出了该参数可取的值。

表 16-6 detecttimerEn 值

detectMode	描述
KeyScan_Release_Detect_Enable	开启按键释放检测
KeyScan_Release_Detect_Disable	关闭按键释放检测

fifoOvrCtrl: 设置如果 FIFO 的数据已满时的数据存储模式。错误!未找到引用源。给出了该参数可取的值。

表 16-7 fifoOvrCtrl 值

fifoOvrCtrl	描述
KeyScan_FIFO_OVR_CTRL_DIS_ALL	当 FIFO 已满，丢弃所有最新的数据
KeyScan_FIFO_OVR_CTRL_DIS_LAST	当 FIFO 已满，丢弃之前的数据

maxScanData: 设置每次扫描允许的最大数据个数。该参数取值范围为 1 到 0x1A。

例：

```
/* Initialize KeyScan */
KEYSCAN_InitTypeDef keyScanInitStruct;
KeyScan_StructInit(&keyScanInitStruct);
/* Debounce time is 16 ms */
keyScanInitStruct.debounceTime = (16 * 32);
keyScanInitStruct.rowSize = KEYPAD_ROW_SIZE;
keyScanInitStruct.colSize = KEYPAD_COLUMN_SIZE;
KeyScan_Init(KEYSCAN, &keyScanInitStruct);
```

16.2.2 函数 KeyScan_DeInit

表 16-8 函数 KeyScan_DeInit

函数名	KeyScan_DeInit
函数原型	void KeyScan_DeInit(KEYSCAN_TypeDef* KeyScan)
功能描述	关闭 KeyScan 时钟
输入参数 1	KeyScan: 指向选择的 KeyScan 模块
输出参数	无
返回值	无
先决条件	无
被调用函数	RCC_PeriphClockCmd()

```
/* Deinitialize keyscan */
KeyScan_DeInit(KEYSCAN);
```

16.2.3 函数 KeyScan_StructInit

表 16-9 函数 KeyScan_StructInit

函数名	KeyScan_StructInit
函数原型	void KeyScan_StructInit(KEYSCAN_InitTypeDef* KeyScan_InitStruct)
功能描述	把 KeyScan_InitStruct 中的每一个值按照缺省值填入
输入参数 1	KeyScan_InitStruct: 指向结构体 KeyScan_InitTypeDef 的指针, 待初始化
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Initialize KeyScan */
KEYSCAN_InitTypeDef keyScanInitStruct;
KeyScan_StructInit(&keyScanInitStruct);
```

16.2.4 函数 KeyScan_INTConfig

表 16-10 函数 KeyScan_INTConfig

函数名	KeyScan_INTConfig
函数原型	void KeyScan_INTConfig(KEYSCAN_TypeDef* KeyScan, uint32_t KeyScan_IT, FunctionalState newState)
功能描述	使能或者失能 KeyScan 指定的中断源
输入参数 1	KeyScan: 指向选择的 KeyScan 模块
输入参数 2	KeyScan_IT: 待使能或者失能的 KeyScan 中断源, 具体参阅 KeyScan_IT 介绍部分
输入参数 3	NewState: 中断的新状态 这个参数可以取 ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

KeyScan_IT: 允许的 KeyScan 中断类型如错误!未找到引用源。所示, 使用操作符 “|” 可以一次选中多个中断作为该参数的值。

表 16-11 KeyScan_IT 值

KeyScan_IT	描述
KEYSCAN_INT_THRESHOLD	FFIO 数据超过阈值中断
KEYSCAN_INT_OVER_READ	FFIO 数据误读中断
KEYSCAN_INT_SCAN_END	单次扫描结束中断
KEYSCAN_INT_FIFO_NOT_EMPTY	FIFO 不为空中断
KEYSCAN_INT_ALL_RELEASE	所有按键释放中断

例:

```
/* Enable scan end interrupt */
KeyScan_INTConfig(KEYSCAN, KEYSCAN_INT_SCAN_END, ENABLE);
```

16.2.5 函数 KeyScan_INTMask

表 16-12 函数 KeyScan_INTMask

函数名	KeyScan_INTMask
函数原型	void KeyScan_INTMask(KEYSCAN_TypeDef* KeyScan, FunctionalState newState)
功能描述	屏蔽 KeyScan 的中断源
输入参数 1	KeyScan: 指向选择的 KeyScan 模块
输入参数 2	NewState: 中断的新状态 这个参数可以取 ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无

被调用函数	无
-------	---

例：

```
/* Mask scan end interrupt */
KeyScan_INTMask(KEYSCAN, ENABLE);
```

16.2.6 函数 KeyScan_Read

表 16-13 函数 KeyScan_Read

函数名	KeyScan_Read
函数原型	void KeyScan_Read(KEYSCAN_TypeDef* KeyScan, uint8_t* outBuf, uint16_t count)
功能描述	从 KeyScan 的 FIFO 中读取数据
输入参数 1	KeyScan: 指向选择的 KeyScan 模块
输入参数 2	outBuf: 将接收到的数据存储到这个指针指向的空间
输入参数 3	count: 读取 FIFO 中数据的个数
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Read FIFO data */
uint8_t dataBuf[26] = {0};
KeyScan_Read(KEYSCAN, dataBuf, 26);
```

16.2.7 函数 KeyScan_Cmd

表 16-14 函数 KeyScan_Cmd

函数名	KeyScan_Cmd
函数原型	void KeyScan_Cmd(KEYSCAN_TypeDef* KeyScan, FunctionalState NewState)
功能描述	使能或者失能 KeyScan 外设
输入参数 1	KeyScan: 指向选择的 KeyScan 模块
输入参数 2	NewState: 中断的新状态 这个参数可以取 ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable keyscan */
KeyScan_Cmd(KEYSCAN, ENABLE);
```

16.2.8 函数 KeyScan_GetFifoDataNum

表 16-15 函数 KeyScan_GetFifoDataNum

函数名	KeyScan_GetFifoDataNum
函数原型	uint16_t KeyScan_GetFifoDataNum(KEYSCAN_TypeDef* KeyScan)
功能描述	得到 FIFO 中数据的长度
输入参数 1	KeyScan: 指向选择的 KeyScan 模块
输出参数	无
返回值	FIFO 中数据的长度
先决条件	无
被调用函数	无

例：

```
/* Get data length in FIFO */
uint8_t len = 0;
len = KeyScan_GetFifoDataNum(KEYSCAN);
```

16.2.9 函数 KeyScan_ClearINTPendingBit

表 16-16 函数 KeyScan_ClearINTPendingBit

函数名	KeyScan_ClearINTPendingBit
函数原型	void KeyScan_ClearINTPendingBit(KEYSCAN_TypeDef* KeyScan, uint32_t KeyScan_IT)
功能描述	清除 KeyScan 外设某一中断源中断挂起位
输入参数 1	KeyScan: 指向选择的 KeyScan 模块
输入参数 2	KeyScan_IT: 挂起的 KeyScan 中断源, 具体参阅 16.2.4 节 KeyScan_IT 介绍部分
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear scan end interrupt */
KeyScan_ClearINTPendingBit(KEYSCAN, KEYSCAN_INT_SCAN_END);
```

16.2.10 函数 KeyScan_ClearFlags

表 16-17 函数 KeyScan_ClearFlags

函数名	KeyScan_ClearFlags
函数原型	void KeyScan_ClearFlags(KEYSCAN_TypeDef* KeyScan, uint32_t KeyScan_FLAG)
功能描述	清除指定状态标志位
输入参数 1	KeyScan: 指向选择的 KeyScan 模块

输入参数 2	KeyScan_FLAG:待清除的标志位, 具体参阅 KeyScan_FLAG 介绍部分
输出参数	无
返回值	无
先决条件	无
被调用函数	无

KeyScan_FLAG: 允许的 KeyScan 状态类型如所示, 使用操作符 “|” 可以一次选中多个中断作为该参数的值。

表 16-18 KeyScan_FLAG 值

KeyScan_FLAG	描述
KEYSCAN_FLAG_FIFOLIMIT	FIFO 数据超过限制标志
KEYSCAN_FLAG_DATAFILTER	FIFO 过滤按键标志
KEYSCAN_FLAG_OVR	FIFO 溢出标志

例:

```
/* Clear scan FIFO overflow flag */
KeyScan_ClearFlags (KEYSCAN, KEYSCAN_FLAG_OVR);
```

16.2.11 函数 KeyScan_GetFlagState

表 16-19 函数 KeyScan_GetFlagState

函数名	KeyScan_GetFlagState
函数原型	FlagStatus KeyScan_GetFlagState(KEYSCAN_TypeDef* KeyScan, uint32_t KeyScan_FLAG)
功能描述	检测指定的状态是否被置位
输入参数 1	KeyScan: 指向选择的 KeyScan 模块
输入参数 2	KeyScan_FLAG:检测的状态标志, 具体参阅 16.2.10 节 KeyScan_FLAG 介绍部分
输出参数	无
返回值	指定标志位状态
先决条件	无
被调用函数	无

例:

```
/* Get FIFO overflow flag state */
bool state = FALSE;
state = KeyScan_GetFlagState (KEYSCAN, KEYSCAN_FLAG_OVR);
```

16.2.12 函数 KeyScan_FilterDataConfig

表 16-20 函数 KeyScan_FilterDataConfig

函数名	KeyScan_FilterDataConfig
函数原型	void KeyScan_FilterDataConfig(KEYSCAN_TypeDef *KeyScan, uint16_t data, FunctionalState NewState)
功能描述	配置 KeyScan 单按键过滤功能

输入参数 1	KeyScan: 指向选择的 KeyScan 模块
输入参数 2	data: 被过滤的按键
输入参数 3	NewState: 是否开启按键数据过滤功能
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Configure filter key of KeyScan */
KeyScan_FilterDataConfig(KEYSCAN,key_temp,ENABLE);
```

16.2.13 函数 KeyScan_debounceConfig

表 16-21 函数 KeyScan_debounceConfig

函数名	KeyScan_debounceConfig
函数原型	void KeyScan_debounceConfig(KEYSCAN_TypeDef *KeyScan, uint8_t time, FunctionalState NewState)
功能描述	配置 KeyScan 硬件消除抖动功能
输入参数 1	KeyScan: 指向选择的 KeyScan 模块
输入参数 2	time: 抖动时间配置
输入参数 3	NewState: 是否开启按键硬件消抖功能
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Configure debounce of KeyScan */
KeyScan_debounceConfig (KEYSCAN, 0x20, ENABLE);
```

16.2.14 函数 KeyScan_ReadFifoData

表 16-22 函数 KeyScan_ReadFifoData

函数名	KeyScan_ReadFifoData
函数原型	uint16_t KeyScan_ReadFifoData(KEYSCAN_TypeDef *KeyScan)
功能描述	从 FIFO 中读取一个数据
输入参数	KeyScan: 指向选择的 KeyScan 模块
输出参数	无
返回值	FIFO 数据
先决条件	无
被调用函数	无

例：

```
uint16_t data = 0;  
data = KeyScan_ReadFifoData (KEYSCAN);
```

17 红外(IR)

17.1 IR 寄存器结构

```
typedef struct
{
    __IO  uint32_t CLK_DIV;
    __IO  uint32_t TX_CONFIG;
    __IO  uint32_t TX_SR;
    __IO  uint32_t RESERVER;
    __IO  uint32_t TX_INT_CLR;
    __IO  uint32_t TX_FIFO;
    __IO  uint32_t RX_CONFIG;
    __IO  uint32_t RX_SR;
    __IO  uint32_t RX_INT_CLR;
    __IO  uint32_t RX_CNT_INT_SEL;
    __IO  uint32_t RX_FIFO;
    __IO  uint32_t IR_VERSION;
    __IO  uint32_t RSVD1[8];
    __IO  uint32_t DMA_CONFIG;
} IR_TypeDef;
```

错误!未找到引用源。例举 IR 所有寄存器

表 17-1 IR 寄存器

寄存器	描述
CLK_DIV	时钟分频寄存器
TX_CONFIG	发送模式下配置寄存器
TX_SR	发送模式下状态寄存器
RESERVER	保留
TX_INT_CLR	发送模式下中断清除寄存器
TX_FIFO	发送模式下数据寄存器
RX_CONFIG	接收模式下配置寄存器
RX_SR	接收模式下状态寄存器
RX_INT_CLR	接收模式下中断清除寄存器
RX_CNT_INT_SEL	接收模式下计数器阈值配置寄存器
RX_FIFO	接收模式下数据寄存器
IR_VERSION	保留
RSVD1[8]	保留
DMA_CONFIG	DMA 控制寄存器

17.2 IR 库函数

错误!未找到引用源。例举 IR 的所有库函数

表 17-2 IR 库函数

函数名	描述
IR_DelInit	关闭红外时钟源
IR_Init	根据 IR_InitStruct 中指定参数初始化红外寄存器
IR_StructInit	把 IR_InitStruct 中的每一个参数按缺省值填入
IR_Cmd	使能或者失能红外发送或者接收模式
IR_StartManualRxTrigger	启动红外手动接收模式
IR_SetRxCounterThreshold	设置触发接收计数器阈值中断的时间阈值
IR_SendBuf	发送红外数据
IR_ReceiveBuf	接收红外数据
IR_INTConfig	使能或者失能红外指定的中断源
IR_MaskINTConfig	屏蔽或不屏蔽红外指定的中断源
IR_GetINTStatus	获得红外指定的中断源状态
IR_ClearINTPendingBit	清除红外某一中断源中断挂起位
IR_GetTxFIFOFreeLen	获取发送 FIFO 的空余长度
IR_GetRxDataLen	获取接收 FIFO 的数据长度
IR_SendData	往红外发送 FIFO 写一个数据
IR_ReceiveData	从红外接收 FIFO 中读取一个数据
IR_SetTxThreshold	设置触发 IR_INT_TF_LEVEL 中断的 FIFO 阈值
IR_SetRxThreshold	设置触发 IR_INT_RF_LEVEL 中断的 FIFO 阈值
IR_ClearTxFIFO	清除发送 FIFO
IR_ClearRxFIFO	清除接收 FIFO
IR_GetFlagStatus	检测红外指定标志的状态是否被置位
IR_SetTxInverse	翻转发送数据类型
IR_TxOutputInverse	翻转红外输出的电平极性
IR_SendCompenBuf	发送带波形补偿功能的数据

17.2.1 函数 IR_DelInit

表 17-3 函数 IR_DelInit

函数名	IR_DelInit
函数原型	void IR_DelInit(void)
功能描述	关闭红外时钟源
输入参数	无
输出参数	无
返回值	无
先决条件	无

被调用函数	RCC_PeriphClockCmd()
-------	----------------------

例：

```
/* Close IR clock */
IR_DeInit();
```

17.2.2 函数 IR_Init

表 17-4 函数 IR_Init

函数名	IR_Init
函数原型	void IR_Init(IR_InitTypeDef* IR_InitStruct)
功能描述	根据 IR_InitStruct 中指定参数初始化 IR 寄存器
输入参数	IR_InitStruct: 指向 IR_InitTypeDef 的指针, IR_InitStruct 中包含相关配置信息
输出参数	无
返回值	无
先决条件	无
被调用函数	无

```
typedef struct
{
    uint32_t IR_Clock;
    uint32_t IR_Freq; /*!< Specifies the clock frequency. This parameter is IR carrier freqency whose unit is KHz.
                        This parameter can be a value of @ref IR_Frequency */
    uint32_t IR_DutyCycle; /*!< Specifies the IR duty cycle. */
    uint32_t IR_Mode; /*!< Specifies the IR mode.
                        This parameter can be a value of @ref IR_Mode */
    uint32_t IR_TxIdleLevel; /*!< Specifies the IR output level in Tx mode
                                This parameter can be a value of @ref IR_Idle_Status */
    uint32_t IR_TxInverse; /*!< Specifies inverse FIFO data or not in TX mode
                                This parameter can be a value of @ref IR_TX_Data_Type */
    uint32_t IR_TxFIFOThrLevel; /*!< Specifies TX FIFO interrupt threshold in TX mode. When TX FIFO depth
                                 <= threshold value, trigger interrupt.
                                 This parameter can be a value of @ref IR_Tx_Threshold */
    uint32_t IR_RxStartMode; /*!< Specifies Start mode in RX mode
                                This parameter can be a value of @ref IR_Rx_Start_Mode */
    uint32_t IR_RxFIFOThrLevel; /*!< Specifies RX FIFO interrupt threshold in RX mode. when RX FIFO
                                 depth > threshold value, trigger interrupt.
                                 This parameter can be a value of @ref IR_Rx_Threshold */
    uint32_t IR_RXFIFOFullCtrl; /*!< Specifies data discard mode in RX mode when RX FIFO is overflow.
                                This parameter can be a value of @ref IR_RX_FIFO_DISCARD_SETTING */
    uint32_t IR_RxTriggerMode; /*!< Specifies trigger in RX mode
                                This parameter can be a value of @ref IR_RX_Trigger_Mode */
    uint32_t IR_RxFilterTime; /*!< Specifies filter time in RX mode
```

```

    This parameter can be a value of @ref IR_RX_Filter_Time */

    uint32_t IR_RxCntThrType; /*!< Specifies counter level type when trigger IR_INT_RX_CNT_THR interrupt in
                                RX mode

    This parameter can be a value of @ref IR_RX_COUNTER_THRESHOLD_TYPE */

    uint32_t IR_RxCntThr; /*!< Specifies counter threshold value when trigger IR_INT_RX_CNT_THR interrupt in
                           RX mode */

}IR_InitTypeDef;

```

IR_Clock: 该参数设置当前系统时钟频率。

IR_Freq: 该参数设置红外的发送或者接收频率。该参数的取值范围需满足以下条件：

$IR_DIV_NUM = (IR_Clock / IR_Freq) - 1$ ， IR_DIV_NUM 的取值范围为0至4095。其中， IR_Clock 是系统时钟频率。例如，当系统时钟为40MHz时，如果需要设置载波频率为38KHz，则

$IR_DIV_NUM = (40000 / 38) - 1 = 1051$ ， IR_DIV_NUM 的值在0至4095之间，满足条件。

IR_DutyCycle: 该参数设置红外载波的占空比。例如，需要输出占空比为 1/3 的载波，该参数设置为 3。该参数的取值范围需要满足以下条件：

$IR_DUTY_NUM = (IR_DIV_NUM + 1) / IR_DutyCycle - 1$ ， IR_DUTY_NUM 的取值范围为0至4095。

例如，当需要输出占空比为 1/3 且载波频率为 38KHz 的载波时，需要检验以下公式的值是否满足条件， $IR_DUTY_NUM = (IR_DIV_NUM + 1) / 3 - 1 = 349$ ， IR_DUTY_NUM 的值在 0 至 4095 之间，满足条件。

IR_Mode: 设置红外运行模式。错误!未找到引用源。给出了该参数可取的值。

表 17-5 IR_Mode 值

IR_Mode	描述
IR_MODE_TX	红外发送模式
IR_MODE_RX	红外接收模式

IR_TxIdleLevel: 设置红外空闲状态的电平极性。错误!未找到引用源。给出了该参数可取的值。

表 17-6 IR_TxIdleLevel 值

IR_TxIdleLevel	描述
IR_IDLE_OUTPUT_HIGH	空闲状态输出高电平
IR_IDLE_OUTPUT_LOW	空闲状态输出低电平

IR_TxInverse: 设置是否反转红外发送数据类型。错误!未找到引用源。给出了该参数可取的值。

表 17-7 IR_TxInverse 值

IR_TxInverse	描述
IR_TX_DATA_NORMAL	不反转红外发送数据类型
IR_TX_DATA_INVERSE	反转红外发送数据类型

IR_TxFIFOThrLevel: 设置红外发送 FIFO 的阈值大小，该参数范围为 0 到 32。

IR_RxStartMode: 设置红外接收模式的启动方式。错误!未找到引用源。给出了该参数可取的值。

表 17-8 IR_RxStartMode 值

IR_RxStartMode	描述
IR_RX_AUTO_MODE	红外自动接收模式
IR_RX_MANUAL_MODE	红外手动接收模式

IR_RxFIFOThrLevel: 设置红外接收 FIFO 的阈值大小，该参数范围为 0 到 32。

IR_RxFIFOFullCtrl: 设置红外接收 FIFO 已满后数据覆盖方式。错误!未找到引用源。给出了该参数可取的值。

表 17-9 IR_RxFIFOFullCtrl 值

IR_RxFIFOFullCtrl	描述
IR_RX_FIFO_FULL_DISCARD_NEWEST	丢弃新接收的数据
IR_RX_FIFO_FULL_DISCARD_OLDEST	丢弃最早接收的数据

IR_RxTriggerMode: 设置红外接收模式下的触发方式。错误!未找到引用源。给出了该参数可取的值。

表 17-10 IR_RxTriggerMode 值

IR_RxTriggerMode	描述
IR_RX_FALL_EDGE	下降沿触发
IR_RX_RISING_EDGE	上升沿触发
IR_RX_DOUBLE_EDGE	下降沿或者上升沿均可触发

IR_RxFilterTime: 设置红外接收模式的硬件过滤杂波时间阈值。错误!未找到引用源。给出了该参数可取的值。

表 17-11 IR_RxFilterTime 值

IR_RxFilterTime	描述
IR_RX_FILTER_TIME_50ns	过滤低于 50ns 的杂波数据
IR_RX_FILTER_TIME_75ns	过滤低于 75ns 的杂波数据
IR_RX_FILTER_TIME_100ns	过滤低于 100ns 的杂波数据
IR_RX_FILTER_TIME_125ns	过滤低于 125ns 的杂波数据
IR_RX_FILTER_TIME_150ns	过滤低于 150ns 的杂波数据
IR_RX_FILTER_TIME_175ns	过滤低于 175ns 的杂波数据
IR_RX_FILTER_TIME_200ns	过滤低于 200ns 的杂波数据

IR_RxCntThrType: 设置触发接收计数器阈值中断的电平类型。错误!未找到引用源。给出了该参数可取的值。

表 17-12 IR_RxCntThrType 值

IR_RxCntThrType	描述
IR_RX_Count_Low_Level	红外接收电平类型为低电平
IR_RX_Count_High_Level	红外接收电平类型为高电平

IR_RxCntThr: 设置触发接收计数器阈值中断的时间阈值，其单位是载波周期值。该参数的取值范围为 0x0 至 0x7fffffff。该参数可用于决定是否需要结束红外接收。例如，当该参数设置为 0x3ff 且载波频率为 38KHz 时，实际检测结束信号的时间阈值为(0x3ff-1)/38000=26.89ms，即当设定的红外电平类型的持续时间为

26.89ms 后，会触发计数器阈值中断。

例：

```
/* Initialize IR */  
IR_InitTypeDef IR_InitStruct;  
IR_StructInit(&IR_InitStruct);  
IR_InitStruct.IR_Freq          = 38;  
IR_InitStruct.IR_DutyCycle    = 2;  
IR_InitStruct.IR_Mode         = IR_MODE_TX;  
IR_InitStruct.IR_TxInverse    = IR_TX_DATA_NORMAL;  
IR_InitStruct.IR_TxFIFOThrLevel = 2;  
IR_Init(&IR_InitStruct);  
IR_Cmd(IR_MODE_TX, ENABLE);
```

17.2.3 函数 IR_StructInit

表 17-13 函数 IR_StructInit

函数名	IR_StructInit
函数原型	IR_StructInit(IR_InitTypeDef* IR_InitStruct)
功能描述	把 IR_InitStruct 中的每一个值按照缺省值填入
输入参数	IR_InitStruct: 指向结构体 IR_InitTypeDef 的指针，包含了外设 IR 的配置参数
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Initialize IR */  
IR_InitTypeDef IRInitStruct;  
IR_StructInit(&IRInitStruct);
```

17.2.4 函数 IR_Cmd

表 17-14 函数 IR_Cmd

函数名	IR_Cmd
函数原型	void IR_Cmd(uint32_t mode, FunctionalState NewState)
功能描述	使能或者失能 IR 外设的指定模式
输入参数 1	mode: 设置红外运行模式，可选参数如下： IR_MODE_TX: 红外发送模式 IR_MODE_RX: 红外接收模式
输入参数 2	NewState: 中断的新状态 这个参数可以取 ENABLE 或者 DISABLE

输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable IR Tx mode */
IR_Cmd(IR_MODE_TX, ENABLE);
```

17.2.5 函数 IR_StartManualRxTrigger

表 17-15 函数 IR_StartManualRxTrigger

函数名	IR_StartManualRxTrigger
函数原型	void IR_StartManualRxTrigger(void)
功能描述	启动红外手动接收模式
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Start IR manual receive mode */
IR_StartManualRxTrigger (IR);
```

17.2.6 函数 IR_SetRxCounterThreshold

表 17-16 函数 IR_SetRxCounterThreshold

函数名	IR_SetRxCounterThreshold
函数原型	void IR_SetRxCounterThreshold(uint32_t IR_RxCntThrType, uint32_t IR_RxCntThr)
功能描述	设置触发接收计数器阈值中断的时间阈值
输入参数 1	IR_RxCntThrType: 触发接收计数器阈值中断的电平类型,, 可选参数如下： IR_RX_Count_Low_Level: 红外接收电平类型为低电平 IR_RX_Count_High_Level: 红外接收电平类型为高电平
输入参数 2	IR_RxCntThr: 触发接收计数器阈值中断的时间阈值, 该参数的取值范围为 0x0 至 0x7fffffff
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Configure receive counter threshold type and value */
```

```
IR_SetRxCounterThreshold (IR_RX_Count_Low_Level, 0x23a);
```

17.2.7 函数 IR_SendBuf

表 17-17 函数 IR_SendBuf

函数名	IR_SendBuf
函数原型	<code>void IR_SendBuf(uint32_t *pBuf, uint32_t len, FunctionalState IsLastPacket)</code>
功能描述	发送红外数据
输入参数 1	pBuf: 发送数据缓冲区首地址
输入参数 2	len: 发送数据的长度
输入参数 3	IsLastPacket: 缓冲区最后一个数据是否是红外发送的最后一个数据包 这个参数可以取 ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Send data */  
uint32_t buf[32];  
IR_SendBuf(buf, 32, DISABLE);
```

17.2.8 函数 IR_ReceiveBuf

表 17-18 函数 IR_ReceiveBuf

函数名	IR_ReceiveBuf
函数原型	<code>void IR_ReceiveBuf(uint32_t *pBuf, uint32_t len)</code>
功能描述	接收红外数据
输入参数 1	pBuf: 接收数据缓冲区首地址
输入参数 2	len: 接收的数据长度
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Receive data */  
uint32_t buf[32];  
uint32_t len = 0;  
len = IR_GetRxDataLen();  
IR_ReceiveBuf(buf, len);
```

17.2.9 函数 IR_INTConfig

表 17-19 函数 IR_INTConfig

函数名	IR_INTConfig
函数原型	void IR_INTConfig(uint32_t IR_INT, FunctionalState newState)
功能描述	使能或者失能红外指定的中断源
输入参数 1	IR_INT: 待使能或者失能的红外中断源, 具体参阅错误!未找到引用源。关于 IR_INT 介绍部分
输入参数 2	newState: 红外中断的新状态 这个参数可以取 ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

IR_INT: 允许的红外中断类型如错误!未找到引用源。所示, 使用操作符“|”可以一次选中多个中断作为该参数的值。

表 17-20 IR_INT 值

IR_INT	描述
IR_INT_TF_EMPTY	检测到接收缓冲区为空会产生该中断
IR_INT_TF_LEVEL	检测到发送缓冲区的数据个数大于设置的接收阈值时会产生该中断
IR_INT_TF_OF	检测到发送缓冲区溢出会产生该中断
IR_INT_RF_FULL	检测到发送缓冲区已满会产生该中断
IR_INT_RF_LEVEL	检测到接收缓冲区的数据个数大于设置的接收阈值时会产生该中断
IR_INT_RX_CNT_OF	检测到计数器溢出会产生该中断
IR_INT_RF_OF	检测到接收缓冲区溢出会产生该中断
IR_INT_RX_CNT_THR	检测到接收计数器达到设定阈值会产生该中断
IR_INT_RF_ERROR	当接收缓冲区已经为空时, 读取接收缓冲区会产生该中断

例:

```
/* Enable IR threshold interrupt. when TX FIFO offset <= threshold value, trigger interrupt*/
IR_INTConfig(IR_INT_TF_LEVEL, ENABLE);
```

17.2.10 函数 IR_MaskINTConfig

表 17-21 函数 IR_MaskINTConfig

函数名	IR_MaskINTConfig
函数原型	void IR_MaskINTConfig(uint32_t IR_INT, FunctionalState newState)
功能描述	屏蔽或不屏蔽红外指定的中断源
输入参数 1	IR_INT: 待屏蔽或者不屏蔽的红外中断源, 具体参阅错误!未找到引用源。关于 IR_INT 介绍部分
输入参数 2	newState: 红外中断的新状态

	这个参数可以取 ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Mask IR interrupt */
IR_MaskINTConfig(IR_INT_RF_LEVEL | IR_INT_RX_CNT_THR, ENABLE);
```

17.2.11 函数 IR_GetINTStatus

表 17-22 函数 IR_GetINTStatus

函数名	IR_GetINTStatus
函数原型	ITStatus IR_GetINTStatus(uint32_t IR_INT)
功能描述	获得红外指定的中断源状态
输入参数	IR_INT: 红外指定中断源, 具体参阅错误!未找到引用源。关于 IR_INT 介绍部分
输出参数	无
返回值	红外指定中断的新状态 (SET 或者 RESET)
先决条件	无
被调用函数	无

例：

```
/* Get IR the specified interrupt status */
ITStatus IR_Status = RESET;
IR_Status = IR_GetINTStatus(IR_INT_RF_LEVEL);
```

17.2.12 函数 IR_ClearINTPendingBit

表 17-23 函数 IR_ClearINTPendingBit

函数名	IR_ClearINTPendingBit
函数原型	void IR_ClearINTPendingBit(uint32_t IR_CLEAR_INT)
功能描述	清除红外某一中断源中断挂起位
输入参数	IR_CLEAR_INT: 红外中断清除类型, 具体参阅错误!未找到引用源。关于 IR_CLEAR_INT 介绍部分
输出参数	无
返回值	无
先决条件	无
被调用函数	无

IR_CLEAR_INT: 允许的红外中断清除类型如错误!未找到引用源。所示, 使用操作符 “|” 可以一次选中多个中断作为该参数的值。

表 17-24 IR_CLEAR_INT 值

IR_CLEAR_INT	描述
IR_INT_TF_EMPTY_CLR	清除接收缓冲区为空中断
IR_INT_TF_LEVEL_CLR	清除发送缓冲区的数据个数大于设置的接收阈值中断
IR_INT_TF_OF_CLR	清除发送缓冲区溢出中断
IR_INT_RF_FULL_CLR	清除发送缓冲区已满中断
IR_INT_RF_LEVEL_CLR	清除接收缓冲区的数据个数大于设置的接收阈值中断
IR_INT_RX_CNT_O_CLR	清除计数器溢出中断
IR_INT_RF_OF_CLR	清除接收缓冲区溢出中断
IR_INT_RX_CNT_THR_CLR	清除接收计数器达到设定阈值中断
IR_INT_RF_ERROR_CLR	清除当接收缓冲区已经为空时读取接收缓冲区产生的中断

例：

```
/* Clear receive FIFO threshold interrupt */
IR_ClearINTPendingBit(IR_INT_RF_LEVEL_CLR);
```

17.2.13 函数 IR_GetTxFIFOFreeLen

表 17-25 函数 IR_GetTxFIFOFreeLen

函数名	IR_GetTxFIFOFreeLen
函数原型	uint16_t IR_GetTxFIFOFreeLen(void)
功能描述	获取发送 FIFO 的空余长度
输入参数	无
输出参数	无
返回值	发送 FIFO 的空余长度
先决条件	无
被调用函数	无

例：

```
/* Get Tx FIFO free length */
uint16_t len = 0;
len = IR_GetTxFIFOFreeLen();
```

17.2.14 函数 IR_GetRxDataLen

表 17-26 函数 IR_GetRxDataLen

函数名	IR_GetRxDataLen
函数原型	uint16_t IR_GetRxDataLen(void)
功能描述	获取接收 FIFO 的数据长度
输入参数	无
输出参数	无
返回值	接收 FIFO 的数据长度
先决条件	无

被调用函数	无
-------	---

例：

```
/* Get Rx FIFO length */  
uint16_t len = 0;  
len = IR_GetRxFIFOLen();
```

17.2.15 函数 IR_SendData

表 17-27 函数 IR_SendData

函数名	IR_SendData
函数原型	void IR_SendData(uint32_t data)
功能描述	往红外发送 FIFO 写一个数据
输入参数	data: 红外发送数据
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Send one data */  
IR_SendData(0x032A);
```

17.2.16 函数 IR_ReceiveData

表 17-28 函数 IR_ReceiveData

函数名	IR_ReceiveData
函数原型	uint32_t IR_ReceiveData(void)
功能描述	从红外接收 FIFO 中读取一个数据
输入参数	无
输出参数	无
返回值	红外接收数据
先决条件	无
被调用函数	无

例：

```
/* Read one data */  
uint32_t data = 0;  
data = IR_ReceiveData();
```

17.2.17 函数 IR_SetTxThreshold

表 17-29 函数 IR_SetTxThreshold

函数名	IR_SetTxThreshold
函数原型	void IR_SetTxThreshold(uint8_t thd)
功能描述	设置触发 IR_INT_TF_LEVEL 中断的 FIFO 阈值
输入参数	thd: 发送 FIFO 的阈值大小, 该参数范围为 0 到 32
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Configure Tx threshold level, when TX FIFO offset <= threshold value, trigger interrupt*/
IR_SetTxThreshold(2);
```

17.2.18 函数 IR_SetRxThreshold

表 17-30 函数 IR_SetRxThreshold

函数名	IR_SetRxThreshold
函数原型	void IR_SetRxThreshold(uint8_t thd)
功能描述	设置触发 IR_INT_RF_LEVEL 中断的 FIFO 阈值
输入参数	thd: 接收 FIFO 的阈值大小, 该参数范围为 0 到 32
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Configure Rx threshold level, when RX FIFO offset >= threshold value, trigger interrupt*/
IR_SetRxThreshold(2);
```

17.2.19 函数 IR_ClearTxFIFO

表 17-31 函数 IR_ClearTxFIFO

函数名	IR_ClearTxFIFO
函数原型	void IR_ClearTxFIFO(void)
功能描述	清除发送 FIFO
输入参数	无
输出参数	无
返回值	无

先决条件	无
被调用函数	无

例：

```
/*Clear Tx FIFO */
IR_ClearTxFIFO();
```

17.2.20 函数 IR_ClearRxFIFO

表 17-32 函数 IR_ClearRxFIFO

函数名	IR_ClearRxFIFO
函数原型	void IR_ClearRxFIFO(void)
功能描述	清除接收 FIFO
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/*Clear Rx FIFO */
IR_ClearRxFIFO();
```

17.2.21 函数 IR_GetFlagStatus

表 17-33 函数 IR_GetFlagStatus

函数名	IR_GetFlagStatus
函数原型	FlagStatus IR_GetFlagStatus(uint32_t IR_FLAG)
功能描述	检测红外指定标志的状态是否被置位
输入参数	IR_FLAG:检测的状态标志，具体参阅 IR_FLAG 介绍部分
输出参数	无
返回值	红外指定标志的新状态 (SET 或者 RESET)
先决条件	无
被调用函数	无

IR_FLAG: 允许的 IR 状态类型如错误!未找到引用源。所示，使用操作符“|”可以一次选中多个中断作为该参数的值。

表 17-34 IR_FLAG 值

IR_FLAG	描述
IR_FLAG_TF_EMPTY	发送 FIFO 为空
IR_FLAG_TF_FULL	发送 FIFO 已满
IR_FLAG_TX_RUN	红外发送中
IR_FLAG_RF_EMPTY	接收 FIFO 为空

IR_FLAG_RX_RUN	红外接收中
----------------	-------

例：

```
/* Get Tx FIFO empty flag status */
bool state = FALSE;
state = IR_GetFlagStatus (IR_FLAG_TF_EMPTY);
```

17.2.22 函数 IR_SetTxInverse

表 17-35 函数 IR_SetTxInverse

函数名	IR_SetTxInverse
函数原型	void IR_SetTxInverse(FunctionalState NewState)
功能描述	翻转发送数据类型
输入参数	NewState: 是否翻转发送数据类型 这个参数可以取 ENABLE 或者 DISABLE ENABLE: 翻转发送数据类型 DISABLE: 不翻转发送数据类型
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/*Inverse data type */
IR_SetTxInverse(ENABLE);
```

17.2.23 函数 IR_TxOutputInverse

表 17-36 函数 IR_TxOutputInverse

函数名	IR_TxOutputInverse
函数原型	void IR_TxOutputInverse(FunctionalState NewState)
功能描述	翻转红外输出的电平极性
输入参数	NewState: 是否翻转红外输出的电平极性 这个参数可以取 ENABLE 或者 DISABLE ENABLE: 翻转红外输出的电平极性 DISABLE: 不翻转外输出的电平极性
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/*Inverse IR output */
```

```
IR_TxOutputInverse (ENABLE);
```

17.2.24 函数 IR_SendCompenBuf

表 17-37 函数 IR_SendCompenBuf

函数名	IR_SendCompenBuf
函数原型	<code>void IR_SendCompenBuf(IR_TX_COMPEN_TYPE comp_type, uint32_t *pBuf, uint32_t len, FunctionalState IsLastPacket)</code>
功能描述	发送带波形补偿功能的数据
输入参数 1	<code>comp_type</code> : 波形补偿类型, 这个参数可以取如下值: <code>IR_COMPEN_FLAG_1_2_CARRIER</code> : 无载波数据段补偿 1/2 载波周期值 <code>IR_COMPEN_FLAG_1_4_CARRIER</code> : 无载波数据段补偿 1/4 载波周期值 <code>IR_COMPEN_FLAG_1_N_SYSTEM_CLK</code> : 具体补偿时间可参阅 错误!未找到引用源 。介绍部分
输入参数 2	<code>pBuf</code> : 发送数据缓冲区首地址
输入参数 3	<code>len</code> : 发送数据的长度
输入参数 4	<code>IsLastPacket</code> : 缓冲区最后一个数据是否是红外发送的最后一个数据包 这个参数可以取 <code>ENABLE</code> 或者 <code>DISABLE</code>
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Send data */  
uint32_t buf[32];  
IR_SendCompenBuf(IR_COMPEN_FLAG_1_2_CARRIER, buf, 32, DISABLE);
```

18 集成电路内置音频总线(I2S)

18.1 I2S 寄存器结构

```

typedef struct
{
    __O  uint32_t TX_DR;
    __IO uint32_t CTRL0;
    __IO uint32_t CTRL1;
    __IO uint32_t DSP_INT_CR;
    __I   uint32_t RX_DR;
    __I   uint32_t FIFO_SR;
    __IO uint32_t ERROR_CNT_SR;
    __IO uint32_t BCLK_DIV;
    __IO uint32_t DMA_TRDLR;
    __I   uint32_t SR;
} I2S_TypeDef;

```

错误!未找到引用源。例举 I2S 所有寄存器

表 18-1 I2S 寄存器

寄存器	描述
TX_DR	发送 FIFO 数据寄存器
CTRL0	控制寄存器 0
CTRL1	控制寄存器 1
DSP_INT_CR	中断控制寄存器
RX_DR	接收 FIFO 数据寄存器
FIFO_SR	FIFO 状态寄存器
ERROR_CNT_SR	错误状态寄存器
BCLK_DIV	时钟分频寄存器
DMA_TRDLR	DMA 配置寄存器
SR	状态寄存器

18.2 I2S 库函数

错误!未找到引用源。例举 I2S 的所有库函数

表 18-2 I2S 库函数

函数名	描述
I2S_DeInit	关闭 I2S 时钟源
I2S_Init	根据 I2S_InitStruct 中指定参数初始化 I2S 寄存器

I2S_StructInit	把 I2S_InitStruct 中的每一个参数按缺省值填入
I2S_Cmd	使能或者失能 I2S 的指定模式
I2S_INTConfig	使能或者失能 I2S 指定的中断源
I2S_GetINTStatus	获得 I2S 指定的中断源状态
I2S_SendData	往发送 FIFO 写一个数据
I2S_ReceiveData	从接收 FIFO 中读取一个数据
I2S_GetTxFIFOFreeLen	获取发送 FIFO 的空余长度
I2S_GetRxFIFOLen	获取接收 FIFO 的数据长度
I2S_GetTxErrCnt	获得发送错误计数器值
I2S_GetRxErrCnt	获得接收错误计数器值
I2S_SwapBytesForSend	翻转需要发送数据的高低字节
I2S_SwapBytesForRead	翻转接收数据的高低字节
I2S_SwapLRChDataForSend	翻转需要发送的左右声道数据
I2S_SwapLRChDataForRead	翻转接收到的左右声道数据

18.2.1 函数 I2S_DeInit

表 18-3 函数 I2S_DeInit

函数名	I2S_DeInit
函数原型	void I2S_DeInit(I2S_TypeDef *I2Sx)
功能描述	关闭 I2S 时钟源
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	RCC_PeriphClockCmd()

例：

```
/* Close I2S0 clock */
I2S_DeInit(I2S0);
```

18.2.2 函数 I2S_Init

表 18-4 函数 I2S_Init

函数名	I2S_Init
函数原型	void I2S_Init(I2S_TypeDef *I2Sx, I2S_InitTypeDef *I2S_InitStruct)
功能描述	根据 I2S_InitStruct 中指定参数初始化 I2S 寄存器
输入参数 1	I2Sx: x 可以设置成 0 或者 1, 选择指定的 I2S 外设
输入参数 2	I2S_InitStruct: 指向 I2S_InitTypeDef 的指针, I2S_InitStruct 中包含相关配置信息
输出参数	无
返回值	无

先决条件	无
被调用函数	无

```

typedef struct
{
    uint32_t I2S_ClockSource;           /*!< Specifies the I2S clock source.
                                         This parameter can be a value of @ref I2S_clock_Source*/
    uint32_t I2S_BClockMi; /*!< Specifies the BLCK clock speed. BCLK = 40MHz*(I2S_BClockNi/I2S_BClockMi).
                                         This parameter must range from 1 to 0xffff */
    uint32_t I2S_BClockNi;            /*!< Specifies the BLCK clock speed.
                                         This parameter must range from 1 to 0x7FFF */
    uint32_t I2S_DeviceMode;          /*!< Specifies the I2S device mode.
                                         This parameter can be a value of @ref I2S_device_mode*/
    uint32_t I2S_ChannelType;          /*!< Specifies the channel type used for the I2S communication.
                                         This parameter can be a value of @ref I2S_Channel_Type */
    uint32_t I2S_TxChSequence; /*!< Specifies the transmission channel sequence used for the I2S communication.
                                         This parameter can be a value of @ref I2S_Tx_Ch_Sequence*/
    uint32_t I2S_RxChSequence; /*!< Specifies the receiving channel sequence used for the I2S communication.
                                         This parameter can be a value of @ref I2S_Rx_Ch_Sequence*/
    uint32_t I2S_DataFormat;           /*!< Specifies the I2S Data format mode.
                                         This parameter can be a value of @ref I2S_Format_Mode*/
    uint32_t I2S_TxBitSequence;        /*!< Specifies the I2S Data bits sequences.
                                         This parameter can be a value of @ref I2S_Tx_Bit_Sequence*/
    uint32_t I2S_RxBitSequence;        /*!< Specifies the I2S Data bits sequences.
                                         This parameter can be a value of @ref I2S_Rx_Bit_Sequence*/
    uint32_t I2S_DataWidth;            /*!< Specifies the I2S Data width.
                                         This parameter can be a value of @ref I2S_Data_Width */
    uint32_t I2S_MCLKOutput;          /*!< Specifies the I2S MCLK output frequency.
                                         This parameter can be a value of @ref I2S_MCLK_Output */
    uint32_t I2S_DMACmd;              /*!< Specifies the I2S DMA control.
                                         This parameter can be a value of @ref FunctionalState */
    uint32_t I2S_TxWaterlevel;         /*!< Specifies the dma watermark level in transmit mode.
                                         This parameter must range from 1 to 63 */
    uint32_t I2S_RxWaterlevel;         /*!< Specifies the dma watermark level in receive mode.
                                         This parameter must range from 1 to 63 */
}I2S_InitTypeDef;

```

I2S_ClockSource: 设置 I2S 时钟源。错误!未找到引用源。给出了该参数可取的值。

表 18-5 I2S_ClockSource 值

I2S_ClockSource	描述
I2S_CLK_40M	40M
I2S_CLK_128fs	128*fs
I2S_CLK_256fs	256*fs

I2S_BClockMi: 设置 I2S BClock 输出的参数 Mi, 该参数范围为 1 到 0xffff。

I2S_BClockNi: 设置 I2S BClock 输出的参数 Ni, 该参数范围为 1 到 0x7FFF。其中, BCLK = 40MHz*(I2S_BClockNi/I2S_BClockMi)

I2S_DeviceMode: 设置 I2S 设备模式。错误!未找到引用源。给出了该参数可取的值。

表 18-6 I2S_DeviceMode 值

I2S_DeviceMode	描述
I2S_DeviceMode_Master	主设备模式
I2S_DeviceMode_Slave	从设备模式

I2S_ChannelType: 设置 I2S 传输通道类型。错误!未找到引用源。给出了该参数可取的值。

表 18-7 I2S_ChannelType 值

I2S_ChannelType	描述
I2S_Channel_Mono	单声道输出
I2S_Channel_stereo	双声道输出

I2S_TxChSequence: 设置 I2S 发送数据时的通道次序。错误!未找到引用源。给出了该参数可取的值。

表 18-8 I2S_TxChSequence 值

I2S_TxChSequence	描述
I2S_TX_CH_L_R	左声道/右声道
I2S_TX_CH_R_L	右声道/左声道
I2S_TX_CH_L_L	左声道/左声道
I2S_TX_CH_R_R	右声道/右声道

I2S_RxChSequence: 设置 I2S 接收数据时的通道次序。错误!未找到引用源。给出了该参数可取的值。

表 18-9 I2S_RxChSequence 值

I2S_RxChSequence	描述
I2S_RX_CH_L_R	左声道/右声道
I2S_RX_CH_R_L	右声道/左声道
I2S_RX_CH_L_L	左声道/左声道
I2S_RX_CH_R_R	右声道/右声道

I2S_DataFormat: 设置 I2S 的数据格式。错误!未找到引用源。给出了该参数可取的值。

表 18-10 I2S_DataFormat 值

I2S_DataFormat	描述
I2S_Mode	I2S 格式
Left_Justified_Mode	左对齐格式
PCM_Mode_A	PCM A 格式
PCM_Mode_B	PCM B 格式

I2S_TxBitSequence: 设置 I2S 发送数据时的比特位次序。错误!未找到引用源。给出了该参数可取的值。

表 18-11 I2S_TxBitSequence 值

I2S_TxBitSequence	描述
I2S_TX_MSB_First	先发送最高比特位
I2S_TX_LSB_First	先发送最低比特位

I2S_RxBitSequence: 设置 I2S 接收数据时的比特位次序。错误!未找到引用源。给出了该参数可取的值。

表 18-12 I2S_RxBitSequence 值

I2S_RxBitSequence	描述
I2S_RX_MSB_First	先接收最高比特位
I2S_RX_LSB_First	先接收最低比特位

I2S_DataWidth: 设置 I2S 的数据宽度。错误!未找到引用源。给出了该参数可取的值。

表 18-13 I2S_DataWidth 值

I2S_DataWidth	描述
I2S_Width_16Bits	16bit 数据宽度
I2S_Width_24Bits	24bit 数据宽度
I2S_Width_8Bits	8bit 数据宽度

I2S_MCLKOutput: 设置 MCLK 输出频率。错误!未找到引用源。给出了该参数可取的值。

表 18-14 I2S_MCLKOutput 值

I2S_MCLKOutput	描述
I2S_MCLK_128fs	128*fs
I2S_MCLK_256fs	256*fs

I2S_DMACmd: 设置是否开启 DMA 数据传输功能。错误!未找到引用源。给出了该参数可取的值。

表 18-15 I2S_DMACmd 值

I2S_DMACmd	描述
I2S_DMA_ENABLE	使能 DMA 数据传输功能
I2S_DMA_DISABLE	失能 DMA 数据传输功能

I2S_TxWaterlevel: 设置 I2S 通过 GDMA 发送中的 watermark 大小。该参数的取值范围为 0x0 至 0x3f。

I2S_RxWaterlevel: 设置 I2S 通过 GDMA 接收中的 watermark 大小。该参数的取值范围为 0x0 至 0x3f。

例：

```
/* Initialize I2S */
I2S_InitTypeDef I2S_InitStruct;
I2S_StructInit(&I2S_InitStruct);
I2S_InitStruct.I2S_ClockSource      = I2S_CLK_40M;
I2S_InitStruct.I2S_BClockMi        = 0x271; /* <!BCLOCK = 16K */
I2S_InitStruct.I2S_BClockNi        = 0x10;
I2S_InitStruct.I2S_DeviceMode     = I2S_DeviceMode_Master;
I2S_InitStruct.I2S_ChannelType    = I2S_Channel_Mono;
I2S_InitStruct.I2S_DataFormat      = I2S_Mode;
```

```
I2S_Init(I2S0, &I2S_InitStruct);
```

18.2.3 函数 I2S_StructInit

表 18-16 函数 I2S_StructInit

函数名	I2S_StructInit
函数原型	I2S_StructInit(I2S_InitTypeDef* I2S_InitStruct)
功能描述	把 I2S_InitStruct 中的每一个值按照缺省值填入
输入参数	I2S_InitStruct: 指向结构体 I2S_InitTypeDef 的指针, 包含了外设 I2S 的配置参数
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Initialize I2S*/
I2S_InitTypeDef I2SInitStruct;
I2S_StructInit(&I2SInitStruct);
```

18.2.4 函数 I2S_Cmd

表 18-17 函数 I2S_Cmd

函数名	I2S_Cmd
函数原型	void I2S_Cmd(I2S_TypeDef *I2Sx, uint32_t mode, FunctionalState NewState)
功能描述	使能或者失能 I2S 的指定模式
输入参数 1	I2Sx: x 可以设置成 0 或者 1, 选择指定的 I2S 外设
输入参数 2	mode: 设置 I2S 模式, 具体参阅错误!未找到引用源。关于 mode 介绍部分
输入参数 3	NewState: I2S 的新状态 这个参数可以取 ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

mode: 允许的 I2S 模式如错误!未找到引用源。所示, 使用操作符 “|” 可以一次选中多个模式作为该参数的值。

表 18-18 mode 值

mode	描述
I2S_MODE_TX	发送模式
I2S_MODE_RX	接收模式

例:

```
/* Enable Tx and Rx mode */
```

```
I2S_Cmd(I2S0, I2S_MODE_TX | I2S_MODE_RX, ENABLE);
```

18.2.5 函数 I2S_INTConfig

表 18-19 函数 I2S_INTConfig

函数名	I2S_INTConfig
函数原型	void I2S_INTConfig(I2S_TypeDef *I2Sx, uint32_t I2S_INT, FunctionalState NewState)
功能描述	使能或者失能 I2S 指定的中断源
输入参数 1	I2Sx: x 可以设置成 0 或者 1, 选择指定的 I2S 外设
输入参数 2	I2S_INT: 待使能或者失能的 I2S 中断源, 具体参阅错误!未找到引用源。关于 I2S_INT 介绍部分
输入参数 3	NewState: 中断的新状态 这个参数可以取 ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

I2S_INT: 允许的 I2S 中断类型如错误!未找到引用源。所示, 使用操作符 “|” 可以一次选中多个中断作为该参数的值。

表 18-20 I2S_INT 值

I2S_INT	描述
I2S_INT_TX_IDLE	检测到发送处于空闲状态会产生该中断
I2S_INT_RF_EMPTY	检测到接收缓冲区为空会产生该中断
I2S_INT_TF_EMPTY	检测到发送缓冲区为空会产生该中断
I2S_INT_RF_FULL	检测到接收缓冲区已满会产生该中断
I2S_INT_TF_FULL	检测到发送缓冲区已满会产生该中断
I2S_INT_RX_READY	检测到接收准备好会产生该中断
I2S_INT_TX_READY	检测发送准备好会产生该中断

例:

```
/* Enable Tx FIFO full interrupt */
I2S_INTConfig(I2S0, I2S_INT_TF_FULL, ENABLE);
```

18.2.6 函数 I2S_GetINTStatus

表 18-21 函数 I2S_GetINTStatus

函数名	I2S_GetINTStatus
函数原型	ITStatus I2S_GetINTStatus(I2S_TypeDef *I2Sx, uint32_t I2S_INT)
功能描述	获得 I2S 指定的中断源状态
输入参数 1	I2Sx: x 可以设置成 0 或者 1, 选择指定的 I2S 外设
输入参数 2	I2S_INT: 指定的中断源类型, 具体参阅错误!未找到引用源。关于 I2S_INT 介绍部分

输出参数	无
返回值	I2S 指定中断的新状态(SET 或者 RESET)
先决条件	无
被调用函数	无

例：

```
/* Get I2S the specified interrupt status */
ITStatus IR_Status = RESET;
IR_Status = I2S_GetINTStatus(I2S0, I2S_INT_TF_FULL);
```

18.2.7 函数 I2S_SendData

表 18-22 函数 I2S_SendData

函数名	I2S_SendData
函数原型	void I2S_SendData(I2S_TypeDef *I2Sx, uint32_t Data)
功能描述	往发送 FIFO 写一个数据
输入参数 1	I2Sx: x 可以设置成 0 或者 1, 选择指定的 I2S 外设
输入参数 2	Data: I2S 数据
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Send one data */
I2S_SendData(I2S0, 0x6688);
```

18.2.8 函数 I2S_ReceiveData

表 18-23 函数 I2S_ReceiveData

函数名	I2S_ReceiveData
函数原型	uint32_t I2S_ReceiveData(I2S_TypeDef *I2Sx)
功能描述	从接收 FIFO 中读取一个数据
输入参数	I2Sx: x 可以设置成 0 或者 1, 选择指定的 I2S 外设
输出参数	无
返回值	接收数据
先决条件	无
被调用函数	无

例：

```
/* Read one data */
uint32_t data = 0;
data = I2S_ReceiveData(I2S0);
```

18.2.9 函数 I2S_GetTxFIFOFreeLen

表 18-24 函数 I2S_GetTxFIFOFreeLen

函数名	I2S_GetTxFIFOFreeLen
函数原型	uint8_t I2S_GetTxFIFOFreeLen(I2S_TypeDef *I2Sx)
功能描述	获取发送 FIFO 的空余长度
输入参数	I2Sx: x 可以设置成 0 或者 1, 选择指定的 I2S 外设
输出参数	无
返回值	发送 FIFO 的空余长度
先决条件	无
被调用函数	无

例：

```
/* Get Tx FIFO free length */
uint16_t len = 0;
len = I2S_GetTxFIFOFreeLen(I2S0);
```

18.2.10 函数 I2S_GetRxFIFOLen

表 18-25 函数 I2S_GetRxFIFOLen

函数名	I2S_GetRxFIFOLen
函数原型	uint8_t I2S_GetRxFIFOLen(I2S_TypeDef *I2Sx)
功能描述	获取接收 FIFO 的数据长度
输入参数	I2Sx: x 可以设置成 0 或者 1, 选择指定的 I2S 外设
输出参数	无
返回值	接收 FIFO 的数据长度
先决条件	无
被调用函数	无

例：

```
/* Get Rx FIFO length */
uint16_t len = 0;
len = I2S_GetRxFIFOLen(I2S0);
```

18.2.11 函数 I2S_GetTxErrCnt

表 18-26 函数 I2S_GetTxErrCnt

函数名	I2S_GetTxErrCnt
函数原型	uint8_t I2S_GetTxErrCnt(I2S_TypeDef *I2Sx)
功能描述	获得发送错误计数器值
输入参数	I2Sx: x 可以设置成 0 或者 1, 选择指定的 I2S 外设

输出参数	无
返回值	发送错误计数器值
先决条件	无
被调用函数	无

例：

```
/* Get Tx error counter value*/
uint8_t value = 0;
value = I2S_GetTxErrCnt (I2S0);
```

18.2.12 函数 I2S_GetRxErrCnt

表 18-27 函数 I2S_GetRxErrCnt

函数名	I2S_GetRxErrCnt
函数原型	uint8_t I2S_GetRxErrCnt(I2S_TypeDef *I2Sx)
功能描述	获得接收错误计数器值
输入参数	I2Sx: x 可以设置成 0 或者 1, 选择指定的 I2S 外设
输出参数	无
返回值	接收错误计数器值
先决条件	无
被调用函数	无

例：

```
/* Get Rx error counter value*/
uint8_t value = 0;
value = I2S_GetRxErrCnt (I2S0);
```

18.2.13 函数 I2S_SwapBytesForSend

表 18-28 函数 I2S_SwapBytesForSend

函数名	I2S_SwapBytesForSend
函数原型	void I2S_SwapBytesForSend(I2S_TypeDef *I2Sx, FunctionalState NewState)
功能描述	翻转需要发送数据的高低字节
输入参数 1	I2Sx: x 可以设置成 0 或者 1, 选择指定的 I2S 外设
输入参数 2	NewState: 是否翻转发送数据 这个参数可以取 ENABLE 或者 DISABLE ENABLE: 翻转发送数据的高低字节 DISABLE: 不翻转发送数据的高低字节
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Swap data for transmission */  
I2S_SwapBytesForSend(I2S0, ENABLE);
```

18.2.14 函数 I2S_SwapBytesForRead

表 18-29 函数 I2S_SwapBytesForRead

函数名	I2S_SwapBytesForRead
函数原型	void I2S_SwapBytesForRead(I2S_TypeDef *I2Sx, FunctionalState NewState)
功能描述	翻转接收数据的高低字节
输入参数 1	I2Sx: x 可以设置成 0 或者 1, 选择指定的 I2S 外设
输入参数 2	NewState: 是否翻转接收数据 这个参数可以取 ENABLE 或者 DISABLE ENABLE: 翻转接收数据的高低字节 DISABLE: 不翻转接收数据的高低字节
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Swap data for reception */  
I2S_SwapBytesForRead(I2S0, ENABLE);
```

18.2.15 函数 I2S_SwapLRChDataForSend

表 18-30 函数 I2S_SwapLRChDataForSend

函数名	I2S_SwapLRChDataForSend
函数原型	void I2S_SwapLRChDataForSend(I2S_TypeDef *I2Sx, FunctionalState NewState)
功能描述	翻转需要发送的左右声道数据
输入参数 1	I2Sx: x 可以设置成 0 或者 1, 选择指定的 I2S 外设
输入参数 2	NewState: 是否翻转发送的左右声道数据 这个参数可以取 ENABLE 或者 DISABLE ENABLE: 翻转发送的左右声道数据 DISABLE: 不翻转发送的左右声道数据
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Swap L/R data for transmission */
```

I2S_SwapLRChDataForSend (I2S0, ENABLE);

18.2.16 函数 I2S_SwapLRChDataForRead

表 18-31 函数 I2S_SwapLRChDataForRead

函数名	I2S_SwapLRChDataForRead
函数原型	void I2S_SwapLRChDataForRead(I2S_TypeDef *I2Sx, FunctionalState NewState)
功能描述	翻转需要接收的左右声道数据
输入参数 1	I2Sx: x 可以设置成 0 或者 1, 选择指定的 I2S 外设
输入参数 2	NewState: 是否翻转接收的左右声道数据 这个参数可以取 ENABLE 或者 DISABLE ENABLE: 翻转接收的左右声道数据 DISABLE: 不翻转接收的左右声道数据
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Swap L/R data for reception */  
I2S_SwapLRChDataForRead (I2S0, ENABLE);
```

19 8080 并行接口控制器(IF8080)

19.1 IF8080 寄存器结构

```
typedef struct
{
    __IO uint32_t CTRL0;
    __IO uint32_t CTRL1;
    __IO uint32_t IMR;
    __I  uint32_t SR;
    __O  uint32_t ICR;
    __IO uint32_t CFG;
    __IO uint32_t FIFO;
    __I  uint32_t RESV0;
    __I  uint32_t RXDATA;
    __IO uint32_t TX_LEN;
    __I  uint32_t TX_CNT;
    __I  uint32_t RESV1;
    __IO uint32_t RX_LEN;
    __I  uint32_t RX_CNT;
    __IO uint32_t CMD1;
    __IO uint32_t CMD2;
    __IO uint32_t CMD3;
} IF8080_TypeDef;
```

错误!未找到引用源。例举 IF8080 所有寄存器

表 19-1 IF8080 寄存器

寄存器	描述
CTRL0	控制寄存器 0
CTRL1	控制寄存器 1
IMR	中断屏蔽寄存器
SR	状态寄存器
ICR	中断清除寄存器
CFG	配置寄存器
FIFO	数据缓冲区寄存器
RXDATA	数据接收寄存器
TX_LEN	发送数据长度寄存器
TX_CNT	发送数据计数寄存器
RX_LEN	接收数据长度寄存器
RX_CNT	接收数据计数寄存器
CMD1	

CMD2	
CMD3	

19.2 IF8080 库函数

错误!未找到引用源。例举 IF8080 的所有库函数

表 19-2IF8080 库函数

函数名	描述
IF8080_DelInit	关闭 IF8080 时钟源
IF8080_PinGroupConfig	配置 IF8080 管脚组
IF8080_Init	根据 IF8080_InitStruct 中指定参数初始化 IF8080 寄存器
IF8080_StructInit	把 IF8080_InitStruct 中的每一个参数按缺省值填入
IF8080_AutoModeCmd	使能或者失能 IF8080
IF8080_SendCommand	手动模式下发送一个字节命令
IF8080_SendData	手动模式下发送数据
IF8080_ReceiveData	手动模式下接收数据
IF8080_Write	手动模式下发送命令与数据
IF8080_Read	手动模式下发送命令后读取数据
IF8080_SetCmdSequence	自动模式下配置命令序列
IF8080_MaskINTConfig	屏蔽或不屏蔽 IF8080 指定的中断源
IF8080_GetINTStatus	获得 IF8080 指定的中断源状态
IF8080_GetFlagStatus	检测 IF8080 指定标志的状态是否被置位
IF8080_SwitchMode	动态切换 IF8080 运行模式
IF8080_GDMAConfig	配置 GDMA LLI
IF8080_GDMACmd	使能或者失能 GDMA 功能
IF8080_SetCS	片选信号拉高
IF8080_ResetCS	片选信号拉低
IF8080_ClearINTPendingBit	清除 IF8080 某一中断源中断挂起位
IF8080_SetTxDataLen	配置 IF8080 发送数据长度
IF8080_GetTxDataLen	读取 IF8080 发送数据长度
IF8080_GetTxCounter	读取 IF8080 已经发送的数据长度
IF8080_SetRxDataLen	配置 IF8080 发送数据长度
IF8080_GetRxDataLen	读取 IF8080 发送数据长度
IF8080_GetRxCounter	读取 IF8080 已经发送的数据长度
IF8080_ClearTxCounter	清除数据发送计数器
IF8080_ClearRxCounter	清除数据发送计数器
IF8080_WriteFIFO	手动模式下发送命令与数据
IF8080_ReadFIFO	手动模式下发送命令后读取数据
IF8080_ClearFIFO	清除 FIFO
IF8080_VsyncCmd	使能或者失能 Vsync 开始功能

19.2.1 函数 IF8080_DeInit

表 19-3 函数 IF8080_DeInit

函数名	IF8080_DeInit
函数原型	void IF8080_DeInit(void)
功能描述	关闭 IF8080 时钟源
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	RCC_PeriphClockCmd()

例：

```
/* Close IF8080 clock */
IF8080_DeInit();
```

19.2.2 函数 IF8080_PinGroupConfig

表 19-4 函数 IF8080_PinGroupConfig

函数名	IF8080_PinGroupConfig
函数原型	void IF8080_PinGroupConfig(uint32_t IF8080_PinGroupType)
功能描述	配置 IF8080 管脚组
输入参数 1	IF8080_PinGroupType: 设置 IF8080 管脚组，可选参数如下： IF8080_PinGroup_DISABLE: 失能 IF8080 管脚组 IF8080_PinGroup_1: CS(P3_3), RD(P3_2), DCX(P3_4), WR(P3_5), D0(P0_2), D1(P0_4), D2(P1_3), D3(P1_4), D4(P4_0), D5(P4_1), D6(P4_2), D7(P4_3) IF8080_PinGroup_2: CS(P3_3), DCX(P3_4), WR(P3_2), RD(P2_0) D0(P3_5), D1(P0_1), D2(P0_2), D3(P0_4), D4(P4_0), D5(P4_1), D6(P4_2), D7(P4_3)
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
IF8080_PinGroupConfig(IF8080_PinGroup_2);
```

19.2.3 函数 IF8080_Init

表 19-5 函数 IF8080_Init

函数名	IF8080_Init
函数原型	void IF8080_Init(IF8080_InitTypeDef *IF8080_InitStruct)

功能描述	根据 IF8080_InitStruct 中指定参数初始化 IF8080 寄存器
输入参数	IF8080_InitStruct: 指向 IF8080_InitTypeDef 的指针, IF8080_InitStruct 中包含相关配置信息
输出参数	无
返回值	无
先决条件	无
被调用函数	无

typedef struct

```
{
    uint32_t IF8080_ClockDiv;                                /*!< Specifies the IF8080 clock speed. */
    uint32_t IF8080_Mode;                                    /*!< Specifies the IF8080 operation mode. */
    uint32_t IF8080_AutoModeDirection;                      /*!< Specifies the IF8080 read or write operation. */
    uint32_t IF8080_InitGuardTimeCmd;                        /*!< Specifies the init guard time function. */
    uint32_t IF8080_InitGuardTime;                           /*!< Specifies the guard time. This parameter is 0~3T
of divider clock. */
    uint32_t IF8080_CmdGuardTimeCmd;                         /*!< Specifies the command guard time function.
*/
    uint32_t IF8080_CmdGuardTime;                            /*!< Specifies the guard time. This parameter is
0~3T of divider clock. */
    uint32_t IF8080_GuardTimeCmd;                           /*!< Specifies the guard time function. The guard
time only available for hardware continuously write mode*/
    uint32_t IF8080_GuardTime;                              /*!< Specifies the guard time. This parameter is 0~3T
of divider clock. */
    uint32_t IF8080_WRDelay;                               /*!< Specifies the delay time for WR. This parameter
is half or one bus clock cycle. */
    uint32_t IF8080_8BitSwap;                             /*!< Specifies the FIFO data format. */
    uint32_t IF8080_16BitSwap;                            /*!< Specifies the FIFO data format. */
    uint32_t IF8080_TxThr;                                /*!< Specifies the TX FIFO threshold value. This value
can be from 0 to 16. */
    uint32_t IF8080_TxDMACmd;                            /*!< Specifies the TX DMA status in auto mode. */
    uint32_t IF8080_VsyncCmd;                            /*!< Specifies the Vsync signal. */
    uint32_t IF8080_VsyncPolarity;                         /*!< Specifies the Vsync trigger polarity.*/
}IF8080_InitTypeDef;
```

IF8080_ClockDiv: 设置 IF8080 时钟分频值。错误!未找到引用源。给出了该参数可取的值。

表 19-6 IF8080_ClockDiv 值

IF8080_ClockDiv	描述
IF8080_CLOCK_DIV_2	二分频
IF8080_CLOCK_DIV_3	三分频
IF8080_CLOCK_DIV_4	四分频
IF8080_CLOCK_DIV_5	五分频
IF8080_CLOCK_DIV_6	六分频

IF8080_CLOCK_DIV_7	七分频
IF8080_CLOCK_DIV_8	八分频

IF8080_Mode: 设置 IF8080 运行模式。错误!未找到引用源。给出了该参数可取的值。

表 19-7 IF8080_Mode 值

IF8080_Mode	描述
IF8080_MODE_AUTO	自动模式
IF8080_MODE_MANUAL	手动模式

IF8080_InitGuardTimeCmd: 设置是否打开保护时间功能。错误!未找到引用源。给出了该参数可取的值。

表 19-8 IF8080_InitGuardTimeCmd 值

IF8080_InitGuardTimeCmd	描述
IF8080_INIT_GUARD_TIME_ENABLE	使能保护时间功能
IF8080_INIT_GUARD_TIME_DISABLE	关闭保护时间功能

IF8080_InitGuardTime: 设置 IF8080 保护时间参数。错误!未找到引用源。给出了该参数可取的值。

表 19-9 IF8080_InitGuardTime 值

IF8080_InitGuardTime	描述
IF8080_INIT_GUARD_TIME_1T	1T(分频后的时钟)
IF8080_INIT_GUARD_TIME_2T	2T(分频后的时钟)
IF8080_INIT_GUARD_TIME_3T	3T(分频后的时钟)
IF8080_INIT_GUARD_TIME_4T	4T(分频后的时钟)

IF8080_CmdGuardTimeCmd: 设置是否打开保护时间功能。错误!未找到引用源。给出了该参数可取的值。

表 19-10 IF8080_CmdGuardTimeCmd 值

IF8080_CmdGuardTimeCmd	描述
IF8080_CMD_GUARD_TIME_ENABLE	使能保护时间功能
IF8080_CMD_GUARD_TIME_DISABLE	关闭保护时间功能

IF8080_CmdGuardTime: 设置 IF8080 保护时间参数。错误!未找到引用源。给出了该参数可取的值。

表 19-11 IF8080_CmdGuardTime 值

IF8080_CmdGuardTime	描述
IF8080_CMD_GUARD_TIME_1T	1T(分频后的时钟)
IF8080_CMD_GUARD_TIME_2T	2T(分频后的时钟)
IF8080_CMD_GUARD_TIME_3T	3T(分频后的时钟)
IF8080_CMD_GUARD_TIME_4T	4T(分频后的时钟)

IF8080_GuardTimeCmd: 设置是否打开保护时间功能。错误!未找到引用源。给出了该参数可取的值。

表 19-12 IF8080_GuardTimeCmd 值

IF8080_GuardTimeCmd	描述
IF8080_GUARD_TIME_ENABLE	使能保护时间功能
IF8080_GUARD_TIME_DISABLE	关闭保护时间功能

IF8080_GuardTime: 设置 IF8080 保护时间参数。错误!未找到引用源。给出了该参数可取的值。

表 19-13 IF8080_GuardTime 值

IF8080_GuardTime	描述
IF8080_GUARD_TIME_1T	1T(分频后的时钟)
IF8080_GUARD_TIME_2T	2T(分频后的时钟)
IF8080_GUARD_TIME_3T	3T(分频后的时钟)
IF8080_GUARD_TIME_4T	4T(分频后的时钟)

IF8080_8BitSwap: 设置 IF8080 数据的高低字节翻转次序。错误!未找到引用源。给出了该参数可取的值。

表 19-14 IF8080_8BitSwap 值

IF8080_8BitSwap	描述
IF8080_8BitSwap_DISABLE	先发送高字节再发送低字节
IF8080_8BitSwap_ENABLE	先发送低字节再发送高字节

IF8080_16BitSwap: 设置 IF8080 高低 16 位数据的翻转次序。错误!未找到引用源。给出了该参数可取的值。

表 19-15 IF8080_16BitSwap 值

IF8080_16BitSwap	描述
IF8080_16BitSwap_DISABLE	先发送高 16 位数据再发送低 16 位数据
IF8080_16BitSwap_ENABLE	先发送低 16 位数据再发送高 16 位数据

IF8080_TxThr: 设置触发 IF8080 阈值中断的阈值大小。该参数的取值范围为 0x0 至 0x10。

IF8080_TxDMACmd: 设置 IF8080 GDMA 发送功能。错误!未找到引用源。给出了该参数可取的值。

表 19-16 IF8080_TxDMACmd 值

IF8080_TxDMACmd	描述
IF8080_TX_DMA_ENABLE	使能 GDMA 功能与内部 FIFO 控制
IF8080_TX_DMA_DISABLE	失能 GDMA 功能与内部 FIFO 控制

IF8080_VsyncCmd: 设置 IF8080 GDMA 发送功能。错误!未找到引用源。给出了该参数可取的值。

表 19-17 IF8080_VsyncCmd 值

IF8080_VsyncCmd	描述
IF8080_VSYNC_ENABLE	使能 Vsync 功能
IF8080_VSYNC_DISABLE	失能 Vsync 功能

IF8080_VsyncPolarity: 设置 IF8080 GDMA 发送功能。错误!未找到引用源。给出了该参数可取的值。

表 19-18 IF8080_VsyncPolarity 值

IF8080_VsyncPolarity	描述
IF8080_VSYNC_POLARITY_ENABLE	使能 Vsync 优先级功能
IF8080_VSYNC_POLARITY_DISABLE	失能 Vsync 优先级功能

例：

```
/* Initialize IF8080 */
```

```
IF8080_InitTypeDef IF8080_InitStruct;
IF8080_StructInit(&IF8080_InitStruct);
IF8080_InitStruct(IF8080_ClockDiv      = IF8080_CLOCK_DIV_2;
IF8080_InitStruct(IF8080_Mode        = IF8080_MODE_MANUAL;
IF8080_InitStruct(IF8080_GuardTimeCmd = IF8080_GUARD_TIME_DISABLE;
IF8080_InitStruct(IF8080_GuardTime   = IF8080_GUARD_TIME_2T;
IF8080_InitStruct(IF8080_8BitSwap    = IF8080_8BitSwap_ENABLE;
IF8080_InitStruct(IF8080_16BitSwap   = IF8080_16BitSwap_DISABLE;
IF8080_InitStruct(IF8080_TxThr       = 10;
IF8080_InitStruct(IF8080_TxDMACmd   = IF8080_TX_DMA_DISABLE;
IF8080_Init(&IF8080_InitStruct);
```

19.2.4 函数 IF8080_StructInit

表 19-19 函数 IF8080_StructInit

函数名	IF8080_StructInit
函数原型	IF8080_StructInit(IF8080_InitTypeDef* IF8080_InitStruct)
功能描述	把 IF8080_InitStruct 中的每一个值按照缺省值填入
输入参数	IF8080_InitStruct: 指向结构体 IF8080_InitTypeDef 的指针, 包含了外设 IF8080 的配置参数
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Initialize IF8080 */
IF8080_InitTypeDef IF8080_InitStruct;
IF8080_StructInit(&IF8080_InitStruct);
```

19.2.5 函数 IF8080_AutoModeCmd

表 19-20 函数 IF8080_AutoModeCmd

函数名	IF8080_AutoModeCmd
函数原型	void IF8080_AutoModeCmd (FunctionalState NewState)
功能描述	使能或者失能 IF8080
输入参数	NewState:IF8080 的新状态 这个参数可以取 ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无

被调用函数	无
-------	---

例：

```
/* Enable IF8080 */  
IF8080_AutoModeCmd (ENABLE);
```

19.2.6 函数 IF8080_SendCommand

表 19-21 函数 IF8080_SendCommand

函数名	IF8080_SendCommand
函数原型	void IF8080_SendCommand(uint8_t cmd)
功能描述	手动模式下发送一个字节命令
输入参数	cmd: command 值，该参数的取值范围为 0x0 至 0xFF
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
IF8080_SendCommand(0x2A);
```

19.2.7 函数 IF8080_SendData

表 19-22 函数 IF8080_SendData

函数名	IF8080_SendData
函数原型	void IF8080_SendData(uint8_t *pBuf, uint32_t len)
功能描述	手动模式下发送数据
输入参数 1	pBuf: 发送数据缓冲区的首地址
输入参数 2	len: 发送数据长度
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Send data */  
uint8_t buf[2] = {0x11, 0x22};  
IF8080_SendData (buf, 2);
```

19.2.8 函数 IF8080_ReceiveData

表 19-23 函数 IF8080_ReceiveData

函数名	IF8080_ReceiveData
函数原型	void IF8080_ReceiveData (uint8_t *pBuf, uint32_t len)
功能描述	手动模式下接收数据
输入参数 1	pBuf: 接收数据缓冲区的首地址
输入参数 2	len: 接收数据长度
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Read data */
uint8_t buf[2] = {0, 0};
IF8080_ReadData (buf, 2);
```

19.2.9 函数 IF8080_Write

表 19-24 函数 IF8080_Write

函数名	IF8080_Write
函数原型	void IF8080_Write(uint8_t cmd, uint8_t *pBuf, uint32_t len)
功能描述	手动模式下发送命令与数据
输入参数 1	cmd: command 值, 该参数的取值范围为 0x0 至 0xFF
输入参数 2	pBuf: 发送数据缓冲区的首地址
输入参数 3	len: 发送数据长度
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Send command and data */
uint8_t buf[2] = {0x11, 0x22};
IF8080_SendData (0x2A, buf, 2);
```

19.2.10 函数 IF8080_Read

表 19-25 函数 IF8080_Read

函数名	IF8080_ReadData
函数原型	void IF8080_Read(uint8_t cmd, uint8_t *pBuf, uint32_t len)
功能描述	手动模式下发送命令后读取数据
输入参数 1	cmd: command 值, 该参数的取值范围为 0x0 至 0xFF
输入参数 2	pBuf: 接收数据缓冲区的首地址

输入参数 3	len: 接收数据长度
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Send command and read data */
uint8_t buf[2] = {0, 0};
IF8080_Read(0x2A, buf, 2);
```

19.2.11 函数 IF8080_SetCmdSequence

表 19-26 函数 IF8080_SetCmdSequence

函数名	IF8080_SetCmdSequence
函数原型	FlagStatus IF8080_SetCmdSequence(uint8_t *pCmdBuf, uint8_t len)
功能描述	自动模式下配置命令序列
输入参数 1	pCmdBuf: 命令序列缓冲区的首地址
输入参数 2	len: 命令长度
输出参数	无
返回值	函数执行状态 (SET 或者 RESET) SET: 配置命令序列成功 RESET: 配置命令序列失败
先决条件	无
被调用函数	无

例：

```
uint8_t cmd = 0x2C;
/* Configure command */
IF8080_SetCmdSequence(&cmd, 1);
```

19.2.12 函数 IF8080_MaskINTConfig

表 19-27 函数 IF8080_MaskINTConfig

函数名	IF8080_MaskINTConfig
函数原型	void IF8080_MaskINTConfig(uint32_t IF8080_INT_MSK, FunctionalState NewState)
功能描述	屏蔽或不屏蔽 IF8080 指定的中断源
输入参数 1	IF8080_INT_MSK: 待屏蔽或者不屏蔽的 IF8080 中断源，具体参阅错误!未找到引用源。 关于 IF8080_INT_MSK 介绍部分
输入参数 2	NewState: IF8080 中断的新状态 这个参数可以取 ENABLE 或者 DISABLE
输出参数	无

返回值	无
先决条件	无
被调用函数	无

IF8080_INT_MSK: 允许的 IF8080 中断屏蔽类型如错误!未找到引用源。所示，使用操作符“|”可以一次选中多个中断作为该参数的值。

表 19-28 IF8080_INT_MSK 值

IF8080_INT_MSK	描述
IF8080_INT_TF_EMPTY_MSK	检测到发送缓冲区为空会产生该中断
IF8080_INT_TF_OF_MSK	检测到发送缓冲区的数据个数大于设置的接收阈值时会产生该中断
IF8080_INT_TF_LEVEL_MSK	检测到发送缓冲区溢出会产生该中断

例：

```
/* Enable Tx FIFO threshold interrupt */
IF8080_MaskINTConfig(IF8080_INT_TF_LEVEL_MSK);
```

19.2.13 函数 IF8080_GetINTStatus

表 19-29 函数 IF8080_GetINTStatus

函数名	IF8080_GetINTStatus
函数原型	ITStatus IF8080_GetINTStatus(uint32_t IF8080_INT)
功能描述	获得 IF8080 指定的中断源状态
输入参数	IF8080_INT：指定的中断源类型，具体参阅错误!未找到引用源。关于 IF8080_INT 介绍部分
输出参数	无
返回值	IF8080 指定中断的新状态(SET 或者 RESET)
先决条件	无
被调用函数	无

IF8080_INT: 允许的 IF8080 中断类型如错误!未找到引用源。所示，使用操作符“|”可以一次选中多个中断作为该参数的值。

表 19-30 IF8080_INT 值

IF8080_INT	描述
IF8080_INT_SR_AUTO_DONE	检测到自动模式下数据传输完成后会产生该中断
IF8080_INT_SR_TF_EMPTY	检测到发送缓冲区为空会产生该中断
IF8080_INT_SR_TF_OF	检测到发送缓冲区溢出会产生该中断
IF8080_INT_SR_TF_LEVEL	检测到发送缓冲区的数据个数大于设置的接收阈值时会产生该中断

例：

```
/* Get IF8080 the specified interrupt status */
ITStatus IF8080_Status = RESET;
IF8080_Status = IF8080_GetINTStatus(IF8080_INT_SR_AUTO_DONE);
```

19.2.14 函数 IF8080_GetFlagStatus

表 19-31 函数 IF8080_GetFlagStatus

函数名	IF8080_GetFlagStatus
函数原型	FlagStatus IF8080_GetFlagStatus(uint32_t IF8080_FLAG)
功能描述	检测 IF8080 指定标志的状态是否被置位
输入参数	IF8080_FLAG:检测的状态标志, 具体参阅 IR_FLAG 介绍部分
输出参数	无
返回值	无
先决条件	红外指定标志的新状态 (SET 或者 RESET)
被调用函数	无

IF8080_FLAG: 允许的 IF8080 状态类型如表 19.1 所示, 使用操作符 “|” 可以一次选中多个中断作为该参数的值。

表 19.1 IF8080_FLAG 值

IF8080_FLAG	描述
IF8080_FLAG_TF_EMPTY	发送 FIFO 为空
IF8080_FLAG_TF_FULL	发送 FIFO 已满

例:

```
/* Get Tx FIFO empty flag status */
bool state = FALSE;
state = IF8080_GetFlagStatus (IF8080_FLAG_TF_EMPTY);
```

19.2.15 函数 IF8080_SwitchMode

表 19-32 函数 IF8080_SwitchMode

函数名	IF8080_SwitchMode
函数原型	void IF8080_SwitchMode(uint32_t mode)
功能描述	动态切换 IF8080 运行模式
输入参数	mode: 设置 IF8080 运行模式, 可选参数如下: IF8080_MODE_AUTO: 自动模式 IF8080_MODE_MANUAL: 手动模式
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Enable Auto mode */
IF8080_Cmd(DISABLE);
IF8080_SwitchMode(IF8080_MODE_AUTO);
```

19.2.16 函数 IF8080_GDMALLIConfig

表 19-33 函数 IF8080_GDMALLIConfig

函数名	IF8080_GDMALLIConfig
函数原型	void IF8080_GDMALLIConfig (FunctionalState NewState)
功能描述	使能或者失能 GDMA 功能
输入参数	NewState: GDMA 功能的新状态, 可选参数如下: ENABLE: 使能 GDMA 功能 DISABLE: 失能 GDMA 功能
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Enable IF8080 GDMA function */
IF8080_GDMALLIConfig (ENABLE);
```

19.2.17 函数 IF8080_GDMACmd

表 19-34 函数 IF8080_GDMACmd

函数名	IF8080_GDMACmd
函数原型	void IF8080_GDMACmd(FunctionalState NewState)
功能描述	使能或者失能 GDMA 功能
输入参数	NewState: GDMA 功能的新状态, 可选参数如下: ENABLE: 使能 GDMA 功能 DISABLE: 失能 GDMA 功能
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例:

```
/* Enable IF8080 GDMA function */
IF8080_GDMACmd(ENABLE);
```

19.2.18 函数 IF8080_SetCS

表 19-35 函数 IF8080_SetCS

函数名	IF8080_SetCS
函数原型	void IF8080_SetCS(void)
功能描述	片选信号拉高
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Pull CS up */  
IF8080_SetCS();
```

19.2.19 函数 IF8080_ResetCS

表 19-36 函数 IF8080_ResetCS

函数名	IF8080_ResetCS
函数原型	void IF8080_ResetCS (void)
功能描述	片选信号拉低
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Pull CS down */  
IF8080_ResetCS();
```

19.2.20 函数 IF8080_ClearINTPendingBit

表 19-37 函数 IF8080_ClearINTPendingBit

函数名	IF8080_ClearINTPendingBit
函数原型	void IF8080_ClearINTPendingBit(uint32_t IF8080_CLEAR_INT)
功能描述	清除 IF8080 某一中断源中断挂起位
输入参数	IF8080_CLEAR_INT: IF8080 中断清除类型，具体参阅错误!未找到引用源。关于 IF8080_CLEAR_INT 介绍部分
输出参数	无

返回值	无
先决条件	无
被调用函数	无

IF8080_CLEAR_INT: 允许的 IF8080 中断清除类型如错误!未找到引用源。所示，使用操作符“|”可以一次选中多个中断作为该参数的值。

表 19-38 IF8080_CLEAR_INT 值

IF8080_CLEAR_INT	描述
IF8080_INT_AUTO_DONE_CLR	清除自动模式下传输完成中断
IF8080_INT_TF_EMPTY_CLR	清除发送缓冲区为空中断
IF8080_INT_TF_OF_CLR	清除发送缓冲区溢出中断
IF8080_INT_TF_LEVEL_CLR	清除发送缓冲区的数据个数大于设置的接收阈值中断

例：

```
/* Clear Tx FIFO threshold interrupt */
IF8080_ClearINTPendingBit(IF8080_INT_TF_LEVEL_CLR);
```

19.2.21 函数 IF8080_SetTxDataLen

表 19-39 函数 IF8080_SetTxDataLen

函数名	IF8080_SetTxDataLen
函数原型	uint32_t IF8080_SetTxDataLen(uint32_t len)
功能描述	配置 IF8080 发送数据长度
输入参数	len: 发送数据长度，该参数的取值范围为 0x0 至 0xFFFF
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Configure pixel number */
IF8080_SetTxDataLen(240*240*2);
```

19.2.22 函数 IF8080_GetTxDataLen

表 19-40 函数 IF8080_GetTxDataLen

函数名	IF8080_GetTxDataLen
函数原型	uint32_t IF8080_GetTxDataLen(void)
功能描述	读取 IF8080 发送数据长度
输入参数	无
输出参数	无
返回值	IF8080 发送的数据长度
先决条件	无

被调用函数	无
-------	---

例：

```
/* GetTx data length */  
uint32_t len = 0;  
len = IF8080_SetTxDataLen();
```

19.2.23 函数 IF8080_SetTxDataLen

表 19-41 函数 IF8080_SetTxDataLen

函数名	IF8080_SetTxDataLen
函数原型	uint32_t IF8080_SetTxDataLen (void)
功能描述	配置 IF8080 发送数据长度
输入参数	无
输出参数	无
返回值	IF8080 已经发送的数据长度
先决条件	无
被调用函数	无

例：

```
/* GetTx data length */  
uint32_t len = 0;  
len = IF8080_SetTxDataLen();
```

19.2.24 函数 IF8080_SetRxDataLen

表 19-42 函数 IF8080_SetRxDataLen

函数名	IF8080_SetRxDataLen
函数原型	uint32_t IF8080_SetRxDataLen (uint32_t len)
功能描述	配置 IF8080 接收数据长度
输入参数	len：接收数据长度，该参数的取值范围为 0x0 至 0xFFFF
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Configure pixel number */  
IF8080_SetRxDataLen (240*240*2);
```

19.2.25 函数 IF8080_GetRxDataLen

表 19-43 函数 IF8080_GetRxDataLen

函数名	IF8080_GetRxDataLen
函数原型	uint32_t IF8080_GetRxDataLen (void)
功能描述	读取 IF8080 发送数据长度
输入参数	无
输出参数	无
返回值	IF8080 发送的数据长度
先决条件	无
被调用函数	无

例：

```
/* GetTx data length */  
uint32_t len = 0;  
len = IF8080_GetRxDataLen();
```

19.2.26 函数 IF8080_GetRxCounter

表 19-44 函数 IF8080_GetRxCounter

函数名	IF8080_GetRxCounter
函数原型	uint32_t IF8080_GetRxCounter (void)
功能描述	读取 IF8080 已经发送的数据长度
输入参数	无
输出参数	无
返回值	IF8080 已经发送的数据长度
先决条件	无
被调用函数	无

例：

```
/* GetTx data length */  
uint32_t len = 0;  
len = IF8080_GetRxCounter();
```

19.2.27 函数 IF8080_ClearTxCounter

表 19-45 函数 IF8080_ClearTxCounter

函数名	IF8080_ClearTxCounter
函数原型	void IF8080_ClearTxCounter (void)
功能描述	清除数据发送计数器
输入参数	无

输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear data counter */  
IF8080_ClearTxCounter();
```

19.2.28 函数 IF8080_WriteFIFO

表 19-46 函数 IF8080_WriteFIFO

函数名	IF8080_WriteFIFO
函数原型	void IF8080_WriteFIFO (void)
功能描述	清除 FIFO
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear FIFO */  
IF8080_WriteFIFO();
```

19.2.29 函数 IF8080_ReadFIFO

表 19-47 函数 IF8080_ReadFIFO

函数名	IF8080_ReadFIFO
函数原型	void IF8080_ReadFIFO (void)
功能描述	清除 FIFO
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear FIFO */  
IF8080_ReadFIFO();
```

19.2.30 函数 IF8080_ClearFIFO

表 19-48 函数 IF8080_ClearFIFO

函数名	IF8080_ClearFIFO
函数原型	void IF8080_ClearFIFO(void)
功能描述	清除 FIFO
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear FIFO */  
IF8080_ClearFIFO();
```

19.2.31 函数 IF8080_VsyncCmd

表 19-49 函数 IF8080_VsyncCmd

函数名	IF8080_VsyncCmd
函数原型	void IF8080_VsyncCmd (void)
功能描述	清除 FIFO
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear FIFO */  
IF8080_VsyncCmd();
```

20 外设使用流程指南

20.1 外设初始化流程

IO 外设的初始化主要分为如下几个部分：

- 设置外设的 PAD 与引脚复用
- 使能外设的时钟信号
- 设置外设初始化参数

- 使能外设

流程如图 20-1 所示。其中 XXX 为需要进行初始化的外设名称，例如 GPIO、I2C、SPI。

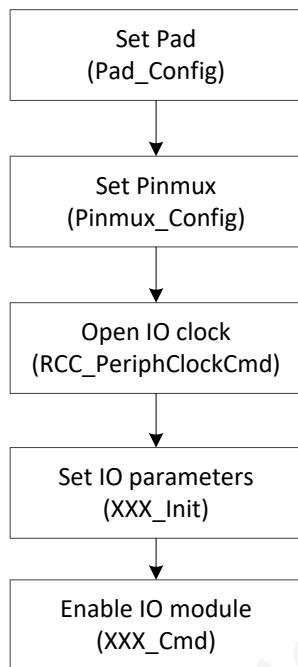


图 20-1 初始化流程图

20.1.1 PAD 配置

通过调用 void Pad_Config 函数配置引脚 pad 状态，例如：

```
Pad_Config(P0_5, PAD_PINMUX_MODE, PAD_IS_PWRON, PAD_PULL_NONE, PAD_OUT_ENABLE,  
PAD_OUT_HIGH);
```

20.1.2 PinMux 配置

通过调用 void Pinmux_Config 函数配置引脚 pinmux 状态，例如：

```
Pinmux_Config(P0_5, GPIO_FUN);
```

20.1.3 时钟配置

通过调用 RCC_PeriphClockCmd 函数开启 GPIO 时钟，例如：

```
RCC_PeriphClockCmd(APBPeriph_GPIO, APBPeriph_GPIO_CLOCK, ENABLE);
```

20.1.4 中断配置

可以调用 NVIC_Init 函数进行 IRQ 中断的使能，例如：

```
NVIC_InitTypeDef NVIC_InitStruct;
NVIC_InitStruct.NVIC_IRQChannel = GPIO5_IRQn;
NVIC_InitStruct.NVIC_IRQChannelPriority = 3;
NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStruct);
```

20.1.5 通用输入/输出(GPIO)初始化

通过调用 GPIO_Init 函数对 GPIO 进行初始化，如错误!未找到引用源。所示。

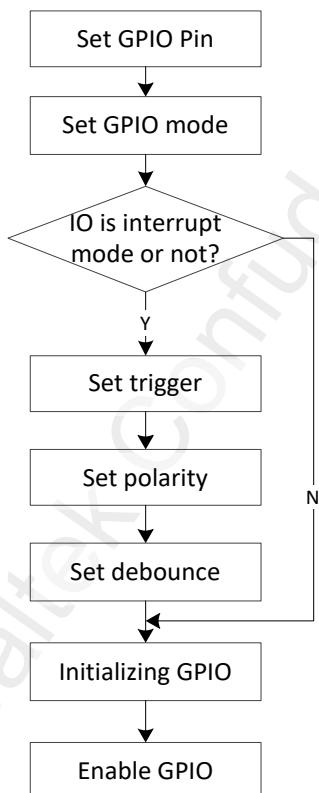


图 20-2 GPIO 初始化流程图

各模式的参考配置参数如下：

20.1.5.1 输入模式

```
GPIO_InitTypeDef GPIO_InitStruct;
GPIO_InitStruct.GPIO_Pin = GPIO_GetPin(P0_5) | GPIO_GetPin(P0_6);
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN;
```

```
GPIO_InitStruct.GPIO_ITCmd = DISABLE;  
GPIO_Init(&GPIO_InitStruct);
```

20.1.5.2 输出模式

```
GPIO_InitTypeDef GPIO_InitStruct;  
GPIO_StructInit(&GPIO_InitStruct);  
GPIO_InitStruct.GPIO_Pin = GPIO_GetPin(P0_5);  
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT;  
GPIO_InitStruct.GPIO_ITCmd = DISABLE;  
GPIO_Init(&GPIO_InitStruct);
```

20.1.5.3 中断模式

```
GPIO_InitTypeDef GPIO_InitStruct;  
GPIO_StructInit(&GPIO_InitStruct);  
GPIO_InitStruct.GPIO_Pin = GPIO_GetPin(P0_5);  
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN;  
GPIO_InitStruct.GPIO_ITCmd = ENABLE;  
GPIO_InitStruct.GPIO_ITTrigger = GPIO_INT_Trigger_EDGE;  
GPIO_InitStruct.GPIO_ITPolarity = GPIO_INT_POLARITY_ACTIVE_LOW;  
GPIO_InitStruct.GPIO_ITDebounce = GPIO_INT_DEBOUNCE_ENABLE;  
GPIO_Init(&GPIO_InitStruct);  
GPIO_INTConfig(GPIO_GetPin(P0_5), ENABLE);
```

20.2 管脚分配定义(PAD) 与引脚复用(PINMUX)使用流程

20.2.1 PAD 和 PINMUX 简介

PINMUX 是 Pin Multiplexing 缩写，即引脚复用。通过 PINMUX，可以对指定管脚配置成特定功能，例如 GPIO、I2C、SPI 等功能。PAD 是用于控制管脚的工作模式、输入/输出状态以及唤醒系统等。PINMUX 电路和所有 IO 模块(除 RTC 和 LPC)都是在 OFF 区，进入 DLPS 后会掉电，寄存器也会被复位，所以在 DLPS 时无法工作。PAD 电路是在 ON 区，DLPS 时不掉电，PAD 寄存器设定和功能依然有效。如果在进入 DLPS 后，管脚需要维持在特定状态或者唤醒系统，需要对 PAD 进行相关配置。**错误!未找到引用源。**是 PINMUX 和 PAD 电路示意图。

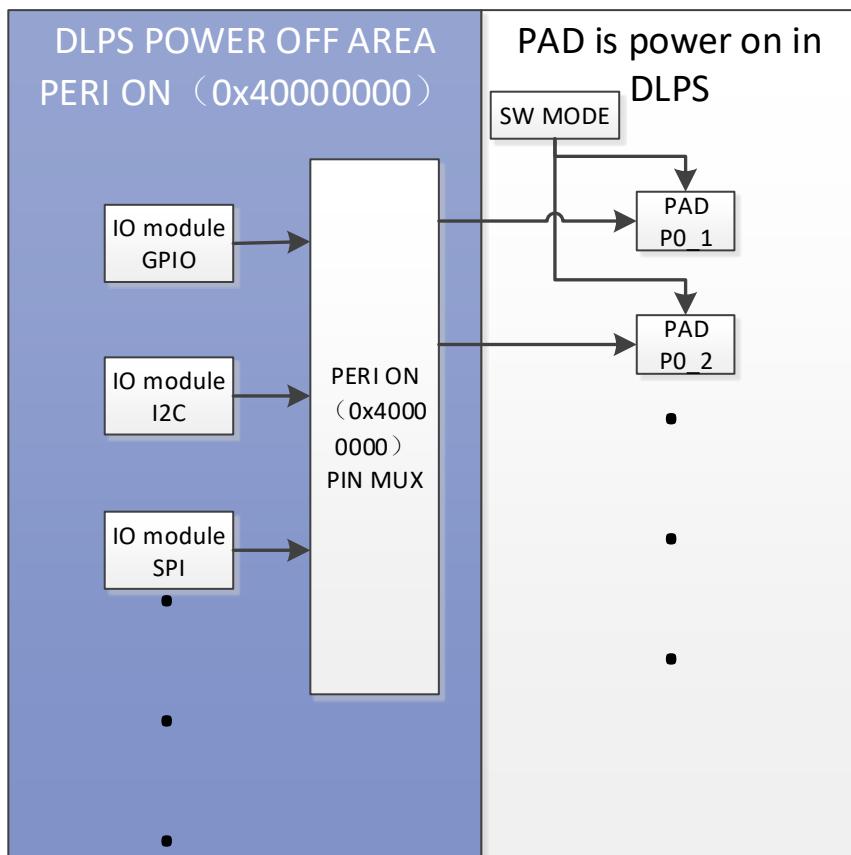


图 20-3 外设和 PAD 电路示意图

20.2.2 PAD 唤醒设定

通过 PAD, RTL8762D 管脚具有系统唤醒功能。使用时，需要在 APP 中使能管脚的系统唤醒功能，同时设定唤醒电平的极性。

例 1: GPIO KEY 唤醒

```
//board.h 中定义管脚有意义的名称
#define KEY_HOME    P3_2
//使能 KEY 的唤醒功能，并且设定是低电平唤醒
System_WakeUpPinEnable(KEY_HOME, 1, 0);
```

例 2: Keypad 按键唤醒

```
//board.h 中定义管脚有意义的名称
#define KEYPAD_ROW0 P4_0
#define KEYPAD_ROW1 P4_1
#define KEYPAD_COLUMN0 P2_2
#define KEYPAD_COLUMN1 P2_3
//使能键盘行的唤醒功能，并且设定是低电平唤醒
```

```
System_WakeUp_Pin_Enable(KEYPAD_ROW0, 1, 0);
System_WakeUp_Pin_Enable(KEYPAD_ROW1, 1, 0);
```

例 3：UART RX 唤醒

```
//board.h 中定义管脚有意义的名称
#define UART_TX_PIN P4_0
#define UART_RX_PIN P4_1

//使能 UART_RX_PIN 的唤醒功能， 并且设定是低电平唤醒
System_WakeUp_Pin_Enable(UART_RX_IN, 1, 0);
```

20.2.3 进出 DLPS 时 PAD 切换流程

进入 DLPS 后，PINMUX 与 IO 模块(除 RTC 和 LPC)会掉电，而 PAD 可以正常工作。为了控制管脚电平状态，在进入 DLPS 前，相关管脚必须设置成软件模式。而退出 DLPS 后，PINMUX 部分会重新上电并恢复寄存器。这时候为了使用 IO 相关功能，管脚必须重新设置为 PINMUX 模式。设定流程如下：APP 初始化时注册进入 DLPS 前的回调函数以及退出 DLPS 后的回调函数；在进入 DLPS 前的回调函数中通过 PAD 设定管脚为 SW Mode；在退出 DLPS 后的回调函数中通过 PAD 设定管脚为 PINMUX Mode。

例：遥控器应用进出 DLPS 时 PAD 模式切换设定

```
//board.h 中定义管脚定义
#define KEY_HOME P3_2
#define KEYPAD_ROW0 P4_0
#define KEYPAD_ROW1 P4_1
#define KEYPAD_COLUMN0 P2_2
#define KEYPAD_COLUMN1 P2_3

//初始化时注册进入 DLPS 前的回调函数以及退出 DLPS 后的回调函数
DLPS_IO_RegUserDlpsEnterCb(RcuEnterDlpsSet);
DLPS_IO_RegUserDlpsExitCb(RcuExitDlpsInit);
```

```
//进入 DLPS 前的回调函数
void RcuEnterDlpsSet(void)
{
    //GPIO KEY 设定成 SW MODE， 默认状态拉高
    Pad_Config(KEY_HOME, PAD_SW_MODE, PAD_IS_PWRON, PAD_PULL_UP, PAD_OUT_DISABLE, PAD_OUT_LOW);
    //Keypad 行设定成 SW MODE， 默认状态拉高；列设定成 SW MODE， 默认状态输出低
    Pad_Config(KEYPAD_ROW0, PAD_SW_MODE, PAD_IS_PWRON, PAD_PULL_UP, PAD_OUT_DISABLE,
               PAD_OUT_LOW);
    Pad_Config(KEYPAD_ROW1, PAD_SW_MODE, PAD_IS_PWRON, PAD_PULL_UP, PAD_OUT_DISABLE,
               PAD_OUT_LOW);
    Pad_Config(KEYPAD_COLUMN0, PAD_SW_MODE, PAD_IS_PWRON, PAD_PULL_NONE, PAD_OUT_ENABLE,
```

```
PAD_OUT_LOW);
Pad_Config(KEYPAD_COLUMN1, PAD_SW_MODE, PAD_IS_PWRON, PAD_PULL_NONE, PAD_OUT_ENABLE,
PAD_OUT_LOW);
}
```

```
//退出 DLPS 后的回调函数
void RcuExitDlpsInit(void)
{
    //GPIO KEY 设定成 PINMUX MODE, 默认状态拉高
    Pad_Config(KEY_HOME, PAD_PINMUX_MODE, PAD_IS_PWRON, PAD_PULL_UP, PAD_OUT_DISABLE,
    PAD_OUT_LOW);
    //Keypress 行设定成 PINMUX MODE, 默认状态拉高; 列设定成 PINMUX MODE, 默认状态输出低
    Pad_Config(KEYPAD_ROW0, PAD_PINMUX_MODE, PAD_IS_PWRON, PAD_PULL_UP, PAD_OUT_DISABLE,
    PAD_OUT_LOW);
    Pad_Config(KEYPAD_ROW1, PAD_PINMUX_MODE, PAD_IS_PWRON, PAD_PULL_UP, PAD_OUT_DISABLE,
    PAD_OUT_LOW);
    Pad_Config(KEYPAD_COLUMN0, PAD_PINMUX_MODE, PAD_IS_PWRON, PAD_PULL_NONE, PAD_OUT_ENABLE,
    PAD_OUT_LOW);
    Pad_Config(KEYPAD_COLUMN1, PAD_PINMUX_MODE, PAD_IS_PWRON, PAD_PULL_NONE, PAD_OUT_ENABLE,
    PAD_OUT_LOW);
}
```

20.2.4 DLPS 时管脚输出电平设定流程

进入 DLPS 时 IO 模块处于掉电状态，所以无法输出确定的信号。为了让某个管脚在进入 DLPS 后依旧维持输出一个确定的信号，需要将管脚切换至 SW mode。比如：输出高电平来点亮 LED 指示灯。设定方式：PAD 设定为 SW Mode 且 Output enable，设定输出高或者低电平。

例：进 DLPS 时点亮 LED 灯

```
//board.h 中定义管脚有意义的名称
#define LED_PIN P3_3

//PAD 设定为 SW Mode, Output Enable 且输出高电平
Pad_Config(LED_PIN, PAD_SW_MODE, PAD_IS_PWRON, PAD_PULL_NONE, PAD_OUT_ENABLE,
PAD_OUT_HIGH);
```

20.2.5 PAD 防漏电设定

需要满足下面两个规则：

- 不能 Input 且 Floating

即没有使用到的 Pin，包括 IC 本身没有出的 Pin，必须设为 SW Mode、Input Mode、Pull Down 或者 Pull Up，不能设置为 Pull None（Pull None 即为 Floating 状态）

未使用的 Pin, 正确的设定方式:

Pad_Config(PIN_unused, PAD_SW_MODE, PAD_IS_PWRON, PAD_PULL_DOWN, PAD_OUT_DISABLE, PAD_OUT_LOW);

- 不要在同一条线上存在相反方向的 Drive or Pull

比如:

- Bee PAD Drive or Pull High, 但是与 Bee 对接的元件是 Drive or Pull Low
- Bee PAD Drive or Pull Low, 但是与 Bee 对接的元件是 Drive or Pull High
- PAD Output High, 并且 Pull Down
- PAD Output Low, 并且 Pull High