第一部分：固件库概述

1 前言

本文档适用于赛元 SC32F1xxx 系列的芯片。

赛元 SC32F1xxx 系列MCU的固件库，提供了一套赛元 SC32F1xxx 系列通用的API接口，以及API接口的应用实例，从而实现赛元 SC32F1xxx 系列MCU的程序标准化。

固件库中，每个外设驱动都由一组函数组成，这组函数覆盖了该外设所有功能。每个器件的开发都由一个通用API驱动，API对该驱动程序的结构，函数和参数名称都进行了标准化。

赛元 SC32F1xxx 系列MCU固件库的使用，使用户无需深入掌握外设配置细节，也可轻松应用每个外设，大大减少用户程序编写时间，降低开发成本。同时，由于赛元 SC32F1xxx 系列MCU的API一致，只需替换相关库文件，无需修改程序细节，便能实现项目程序在不同MCU之间的快速替换。

2 固件库架构规则

缩写含义

Table 1

ADC	模数转换器
BTM	低频时钟定时器
CMP	模拟比较器
CRC	循环冗余校验
DMA	直接内存存取控制器
GPIO	通用输入输出
LCD	LCD 显示驱动集成电路
LED	LED 显示驱动集成电路
INT	外部中断事件控制器
OP	运放及可编程增益放大器
OPTION	用户选项字节
PWM	脉宽调制
LEDPWM	LED 脉宽调制
SPI1_TWI1	串行外设接口与 I2C 内部集成电路集合
RCC	复位与时钟控制器
SPI	串行外设接口
TIM	定时器
TWI	I2C 内部集成电路
UART	通用异步收发器
IAP	在应用中编程
PGA	可变增益放大器
DAC	数模转换器
QEP	独立正交编码捕捉模块
TEMPER	内部温度传感器
CAN	CAN 通信
VREF	内部基准电压
WDT	看门狗
PWR	电源控制器
NVIC	中断控制器

命名规则

固件库遵从以下命名规则：

PPP表示任一外设缩写，例如：ADC。

系统、源文件和头文件命名都以“sc32f1xxx_”作为开头，例如：sc32f1xxx_adc.c，sc32f1xxx_adc.h。

常量仅被应用于一个文件的，定义于该文件中；被应用于多个文件的，在对应头文件中定义。所有常量都由英文字母大写书写。

外设函数的命名以该外设的缩写加下划线为开头。每个单词的第一个字母都由英文字母大写书写，例如：SPI_SendData。在函数名中，只允许存在一个下划线，用以分隔外设缩写和函数名的其他部分。

名为PPP_Init的函数，功能为初始化外设PPP，例如：ADC_Init。

名为PPP_DeInit的函数，功能为复位外设PPP的所有寄存器至缺省值，例如：RCC_DeInit。

名为PPP_Cmd的函数，功能为使能外设PPP，例如：PWM_Cmd。

名为PPP_ITConfig的函数，功能为使能或失能来自外设PPP的中断源，例如：TIM_ITConfig。

名为PPP_GetFlagStatus的函数，功能为检查外设PPP某标志位被设置与否，例如：UART_GetFlagStatus。

名为PPP_ClearFlag的函数，功能为清除外设PPP的标志位，例如：UART_ClearFlag。

编码规则

布尔型

在文件sc32f1xxx.h中，布尔型被定义如下：

```
typedef enum{FALSE=0,TRUE=!FALSE}boolType;
```

状态类型

标志位状态、中断状态、位状态被定义如下：

```
typedef enum{RESET=0,SET=!RESET}FlagStatus,ITStatus;
```

功能状态被定义如下：

```
typedef enum{DISABLE=0,ENABLE=!DISABLE}FunctionalState;
```

错误状态被定义如下：

```
typedefe num{SUCCESS=0, ERROR=! SUCCESS}ErrorStatus;
```

外设

用户可以通过指向各个外设的指针访问各外设的控制寄存器。这些指针所指向的数据结构与各个外设的控制寄存器布局一一对应。

文件sc32f1xxx.h包含了所有外设控制寄存器的结构，下例为 SPI寄存器结构的声明：

```
typedef struct
{
  __IO uint32_t SPI_CON;       /*!< SPIControl register,     Address offset: 0x00 */
  __IO uint32_t SPI_STS;       /*!< SPIStatus register,      Address offset: 0x04 */
  __IO uint32_t RESERVED;      /*!< Reserved,                Address offset: 0x08 */
  __IO uint32_t SPI_DATA;      /*!< SPIData register,      Address offset: 0x0C */
  __IO uint32_t SPI_IDE;       /*!< SPIInterrupt /Dma Enable register,   Address offset: 0x10 */
} SPI_TypeDef;
```

寄存器命名遵循上节的寄存器缩写命名规则。RESERVED表示被保留区域。

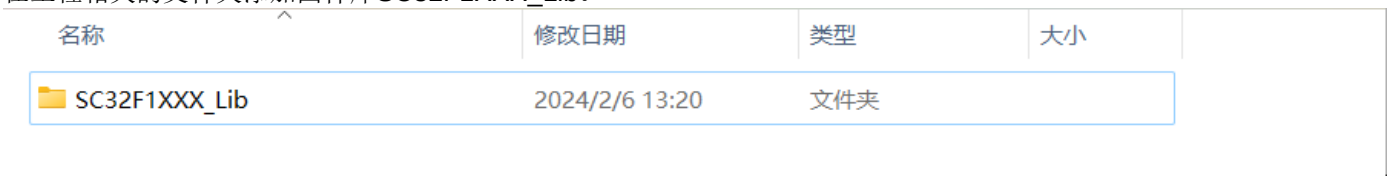
固件库文件结构

Table 2

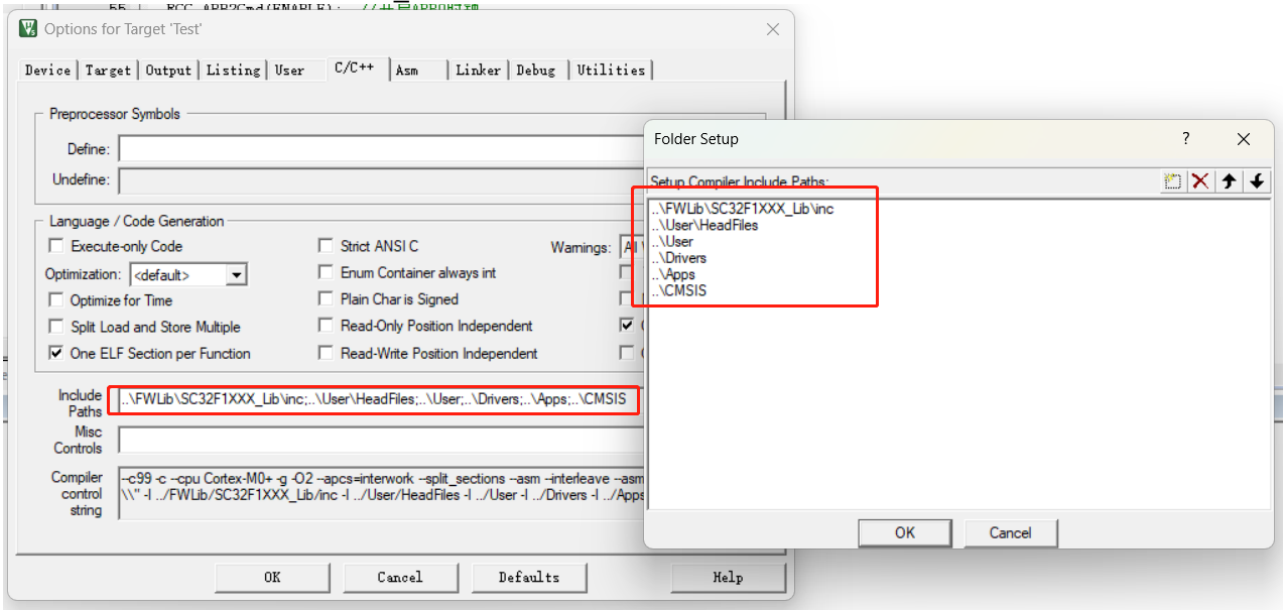
sc32f1xxx_yyy.c	外设的驱动源文件，包含了该外设的通用 API。sc32f1xxx 代表该系列 IC，yyy 代表外设缩写。该文件包含头文件 sc32f1xxx _yyy.h
sc32f1xxx_yyy.h	外设的驱动头文件，包含 API 相关定义。
sc32f1xxx.h	固件库的通用头文件，包含整个固件库通用的类型说明及定义等。
SC_it.c/h	中断服务函数源/头文件，包含了 IC 所有中断服务函数，中断处理在这里执行。

3 固件库函数调用方法

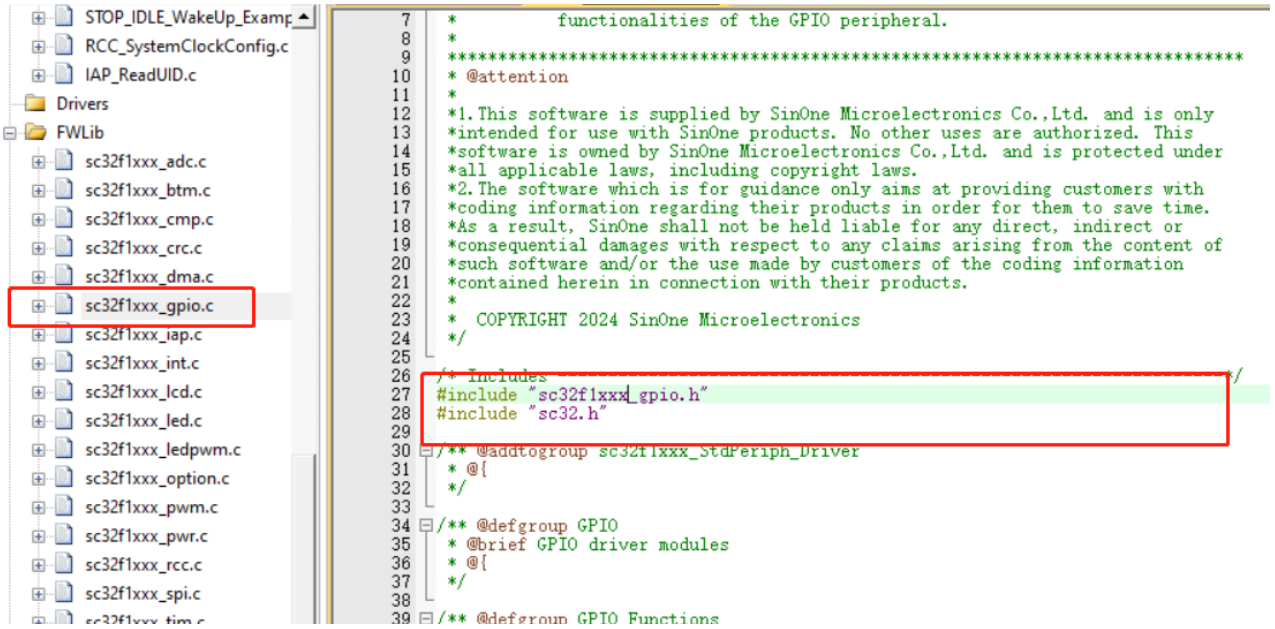
在工程相关的文件夹添加固件库SC32F1XXX_Lib。



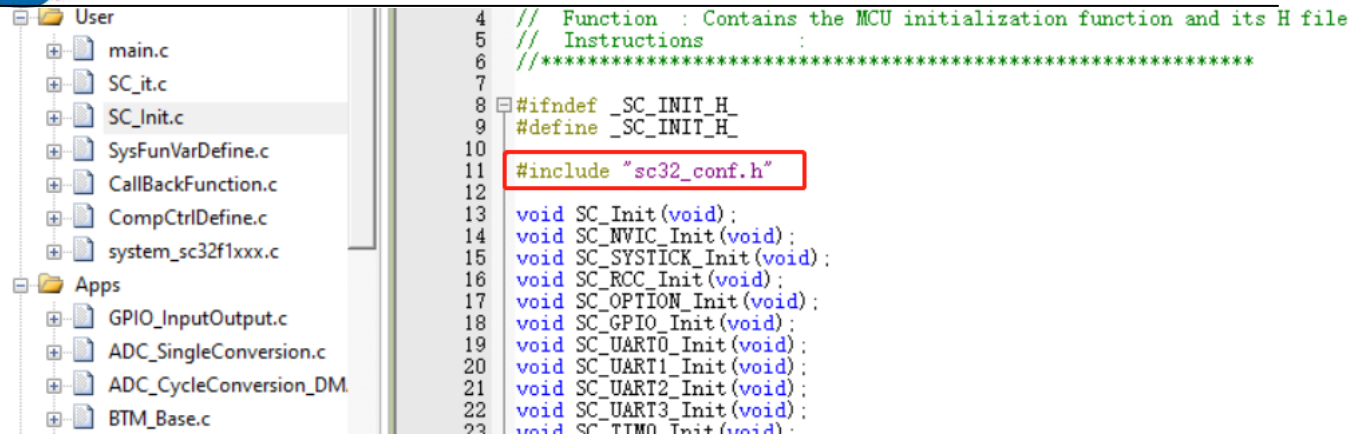
在工程内添加固件库SC32F1XXX_Lib中头文件的路径，路径指向inc文件夹、User文件夹和CMSIS文件夹。



在工程内添加固件库。



调用固件库函数的文件内包含相关的头文件。



至此，完成对固件库函数的调用。

第二部分：固件库函数说明

ADC 固件库函数

SC32F1xxx 系列提供一个多位ADC逐次逼近型模数转换器。每个通道的A/D转换可在单次、连续采样模式下进行。ADC的结果存储在一个的32位数据寄存器中。

ADC 寄存器结构

ADC 寄存器结构，ADC_TypeDef，在文件“sc32f1xxx.h”中定义如下：

```
typedef struct
{
  __IO uint32_t ADC_CON;
  __IO uint32_t ADC_STS;
  __IO uint32_t ADC_VALUE;
  __IO uint32_t ADC_CFG;
} ADC_TypeDef;
```

在一些型号如（SC32F15Gx）中定义如下：

```
typedef struct
{
  __IO uint32_t ADC_CON;
  __IO uint32_t ADC_STS;
  __IO uint32_t ADC_VALUE;
  __IO uint32_t ADC_CFG;
  __IO uint32_t ADC_TH_CFG;
  __IO uint32_t ADC_LOWTH;
  __IO uint32_t ADC_UPTH;
  __IO uint32_t ADC_IDE;
  __IO uint32_t RESERVED0;
  __IO uint32_t ADC_SQCNT;
  __IO uint32_t ADC_SQ0;
} ADC_TypeDef;
```

ADC 寄存器表格

Table 3

寄存器	描述
ADC_CON	ADC 控制寄存器
ADC_STS	ADC 标志状态位寄存器
ADC_VALUE	ADC 转换数值寄存器
ADC_CFG	ADC 端口设置寄存器
ADC_TH_CFG	ADC 通道阈值使能寄存器
ADC_LOWTH	ADC 阈值下限设置寄存器
ADC_UPTH	ADC 阈值上限设置寄存器
ADC_IDE	ADC 中断使能寄存器
ADC_SQCNT	序列通道数设置寄存器
ADC_SQ0	ADC 序列设置寄存器

ADC 固件库函数列表

Table 4

函数名	描述
ADC_DeInit	ADC 相关寄存器复位至缺省值
ADC_Init	根据 ADC_InitStruct 中指定的参数初始化外设 ADCx 的寄存器
ADC_StructInit	把 ADC_InitStruct 中的每一个参数按缺省值填入

ADC_Cmd	使能或者失能指定的 ADC
ADC_ConvModeConfig	配置 ADC 的转换模式
ADC_SetChannel	配置 ADC 输入通道
ADC_GetChannel	获取 ADC 输入通道
ADC_SoftwareStartConv	使能或者失能指定的 ADC 的软件转换启动功能
ADC_GetConversionValue	返回最近一次 ADCx 的转换结果
ADC_ITConfig	使能或者失能指定的 ADC 的中断
ADC_GetFlagStatus	检查指定 ADC 标志位设置与否
ADC_ClearFlag	清除ADCx 的待处理标志位
ADC_DMACmd	使能或者失能指定的 ADC 的DMA 请求
ADC_SequenceChannelConfig	配置 ADC 序列采样通道序列设置
ADC_SequenceGroupConfig	配置 ADC 序列采样位置和采样个数配置
ADC_SetThresholds	配置 ADC 的上下限阈值
ADC_ThresholdsChannelConfig	使能 ADC 通道阈值

ADC 固件库函数详解

ADC_DeInit

Table 5

函数名	ADC_DeInit
函数原型	void ADC_DeInit(ADC_TypeDef* ADCx)
功能描述	ADC 相关寄存器复位至缺省值
输入参数	ADCx：选择 ADC 外设
返回值	无

使用示例:

```
/* ADC寄存器复位 */
ADC_DeInit(ADC);
```

ADC_Init

Table 6

函数名	ADC_Init
函数原型	void ADC_Init(ADC_TypeDef* ADCx, ADC_InitTypeDef* ADC_Base_InitStruct)
功能描述	根据 ADC_Base_InitStruct 中指定的参数初始化外设 ADCx 的寄存器
输入参数 1	ADCx：选择 ADC 外设
输入参数 2	ADC_Base_InitStruct：指向结构 ADC_InitTypeDef 的指针，包含了指定外设 ADC 的配置信息
返回值	无

ADC_InitTypeDef

ADC_InitTypeDef 定义于文件“sc32f1xxx_adc.h”：

```
typedef struct
{
uint16_t ADC_ConvMode;
uint32_t ADC_Prescaler;
uint32_t ADC_EAIN;
uint32_t ADC_VREF;
} ADC_InitTypeDef;
```

ADC_ConvMode

ADC_ConvMode设置ADC工作在单次或者连续转换模式，参阅Table 7获得这个参数的所有成员。

Table 7

ADC_ConvMode	描述
ADC_ConvMode_Single	ADC 工作在单次转换模式

ADC_ConvMode_Continuous	ADC 工作在连续转换模式
ADC_ConvMode_Sequence	ADC 工作在序列采样模式

ADC_ Prescaler

ADC_Prescaler设置ADC的采样时间，参阅Table 8获得这个参数的所有成员。

Table 8

ADC_Prescaler	描述
ADC_Prescaler_3CLOCK	ADC 采样时间约 3 个系统时钟
ADC_Prescaler_6CLOCK	ADC 采样时间约 6 个系统时钟
ADC_Prescaler_16CLOCK	ADC 采样时间约 16 个系统时钟
ADC_Prescaler_32CLOCK	ADC 采样时间约 32 个系统时钟
ADC_Prescaler_15CLOCK	ADC 采样时间约 15 个系统时钟
ADC_Prescaler_30CLOCK	ADC 采样时间约 30 个系统时钟
ADC_Prescaler_60CLOCK	ADC 采样时间约 60 个系统时钟
ADC_Prescaler_120CLOCK	ADC 采样时间约 120 个系统时钟
ADC_Prescaler_480CLOCK	ADC 采样时间约 480 个系统时钟

ADC_EAIN

ADC_EAIN设置ADC的输入通道，参阅Table 9获得这个参数的所有成员。

Table 9

ADC_EAIN	描述
ADC_EAIN_Less	无外部 ADC 输入
ADC_EAIN_0	AIN0 为 ADC 输入
ADC_EAIN_1	AIN1 为 ADC 输入
ADC_EAIN_2	AIN2 为 ADC 输入
ADC_EAIN_3	AIN3 为 ADC 输入
ADC_EAIN_4	AIN4 为 ADC 输入
ADC_EAIN_5	AIN5 为 ADC 输入
ADC_EAIN_6	AIN6 为 ADC 输入
ADC_EAIN_7	AIN7 为 ADC 输入
ADC_EAIN_8	AIN8 为 ADC 输入
ADC_EAIN_9	AIN9 为 ADC 输入
ADC_EAIN_10	AIN10 为 ADC 输入
ADC_EAIN_11	AIN11 为 ADC 输入
ADC_EAIN_12	AIN12 为 ADC 输入
ADC_EAIN_13	AIN13 为 ADC 输入
ADC_EAIN_14	AIN14 为 ADC 输入
ADC_EAIN_15	AIN15 为 ADC 输入
ADC_EAIN_16	AIN16 为 ADC 输入
ADC_EAIN_17	AIN17 为 ADC 输入
ADC_EAIN_All	所有 AINx 均为 ADC 输入

SC32f10xx可选 (ADC_EAIN_Less - ADC_EAIN_16，与ADC_EAIN_All)
SC32f11xx 可选(ADC_EAIN_Less - ADC_EAIN_17，与ADC_EAIN_All)
SC32f12xx、SC32f15xx 可选(ADC_EAIN_Less - ADC_EAIN_15，与ADC_EAIN_All)

ADC_VREF

ADC_VREF设置ADC的参考电压，参阅Table 10获得这个参数的所有成员。

Table 10

ADC_ VREF	描述
ADC_VREF_VDD	ADC 参考电压为 VDD
ADC_VREF_2_048V	ADC 参考电压为内部准确的 2.048V
ADC_VREF_1_024V	ADC 参考电压为内部准确的 1.024V
ADC_VREF_2_4V	ADC 参考电压为内部准确的 2.4V
ADC_RefSource _VDD	ADC 参考电压为 VDD

ADC_RefSource_VREF	ADC 参考电压为模块基准源

SC32f10xx、SC32f11xx、SC32f12xx可选 (ADC_VREF_VDD - ADC_VREF_2_4V)
SC32f15xx 可选(ADC_VREF_VDD - ADC_RefSource_VREF)

使用示例:

```
/* 根据ADC_InitStruct 中指定的参数初始化ADC */
ADC_InitTypeDef   ADC_InitStruct;
ADC_InitStruct.ADC_EAIN = ADC_EAIN_7;          //选定AIN7为ADC输入，并自动将上拉电阻移除
ADC_InitStruct. ADC_ConvMode = ADC_ConvMode_Single;   //ADC为单次转换模式
ADC_InitStruct.ADC_VREF = ADC_VREF_2_4V;   //ADC的基准电压为2.4V
ADC_InitStruct.ADC_Prescaler = ADC_Prescaler_6CLOCK;  //采样时间约6个系统时钟
ADC_Init(ADC,&ADC_InitStruct);
```

ADC_StructInit

Table 11

函数名	ADC_StructInit
函数原形	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，待初始化
返回值	无

ADC_InitStruct

ADC_InitStruct的成员缺省值如Table 12所示。

Table 12

成员	缺省值
ADC_ConvMode	ADC_ConvMode_Single
ADC_Prescaler	ADC_EAIN_Less
ADC_EAIN	ADC_Prescaler_3CLOCK
ADC_VREF	参考电压 VDD

使用示例:

```
/* 把 ADC_InitStruct 中的每一个参数按缺省值填入 */
ADC_InitTypeDef  ADC_InitStruct;
ADC_StructInit(&ADC_InitStruct);
```

ADC_Cmd

Table 13

函数名	ADC_Cmd
函数原形	void ADC_Cmd(ADC_TypeDef* ADCx, FunctionalState NewState)
功能描述	使能或者失能指定的 ADC
输入参数 1	ADCx：选择 ADC 外设
输入参数 2	NewState：ADC 功能开启/关闭，可取值 ENABLE 或 DISABLE
返回值	无

使用示例:

```
/* 使能ADC功能 */
ADC_Cmd(ADC,ENABLE);
```

ADC_ConvModeConfig

Table 14

函数名	ADC_ConvModeConfig
函数原形	void ADC_ConvModeConfig(ADC_TypeDef* ADCx, ADC_ConvMode_TypeDef ADC_ConvMode)
功能描述	配置 ADC 的转换模式
输入参数 1	ADCx：选择 ADC 外设
输入参数 2	ADC_ConvMode：ADC 转换选择
返回值	无

使用示例:

```
/* 配置ADC为单次转换模式 */
ADC_ConvModeConfig(ADC, ADC_ConvMode_Single);
```

ADC_SetChannel

Table 15

函数名	ADC_SetChannel
函数原形	void ADC_SetChannel(ADC_TypeDef* ADCx, ADC_ChannelTypedef ADC_Channel)
功能描述	配置 ADC 输入通道
输入参数 1	ADCx：选择 ADC 外设
输入参数 2	ADC_Channel：ADC 输入通道选择
返回值	无

ADC_Channel

参数 ADC_Channel 指定了通过函数ADC_SetChannel来设置的 ADC 通道。Table 16列举了 ADC_Channel 可取的值。

Table 16

ADC_Channel	描述
ADC_Channel_0	选择 ADC 通道 0
ADC_Channel_1	选择 ADC 通道 1
ADC_Channel_2	选择 ADC 通道 2
ADC_Channel_3	选择 ADC 通道 3
ADC_Channel_4	选择 ADC 通道 4
ADC_Channel_5	选择 ADC 通道 5
ADC_Channel_6	选择 ADC 通道 6
ADC_Channel_7	选择 ADC 通道 7
ADC_Channel_8	选择 ADC 通道 8
ADC_Channel_9	选择 ADC 通道 9
ADC_Channel_10	选择 ADC 通道 10
ADC_Channel_11	选择 ADC 通道 11
ADC_Channel_12	选择 ADC 通道 12
ADC_Channel_13	选择 ADC 通道 13
ADC_Channel_14	选择 ADC 通道 14
ADC_Channel_15	选择 ADC 通道 15
ADC_Channel_16	选择 ADC 通道 16
ADC_Channel_17	选择 ADC 通道 17
ADC_Channel_VDD_D4	ADC 输入为 1/4 VDD
ADC_Channel_PGA	ADC 输入为 PGA
ADC_Channel_OP	ADC 输入为 OP
ADC_Channel_TEMP	ADC 输入为 TEMP

SC32f10xx (ADC_Channel_0 - ADC_Channel_16)
SC32f11xx可选 (ADC_Channel_0 - ADC_Channel_17与ADC_Channel_PGA)
SC32f12xx可选 (ADC_Channel_0 - ADC_Channel_17与ADC_Channel_OP)
SC32f15xx 可选(ADC_Channel_0 - ADC_Channel_15与ADC_Channel_TEMP)

使用示例:

```
/* 配置AIN9为ADC输入通道 */
ADC_SetChannel(ADC,ADC_Channel_9);
```

ADC_GetChannel

Table 17

函数名	ADC_GetChannel

函数原形	ADC_ChannelTypedef ADC_GetChannel(ADC_TypeDef* ADCx)
功能描述	获取 ADC 输入通道
输入参数	ADCx：选择 ADC 外设
返回值	ADC 通道类型

使用示例:

```
/* 获得ADC的输入通道类型 */
ADC_ChannelTypedef  ADC_Channel;
ADC_Channel = ADC_GetChannel(ADC);
```

ADC_SoftwareStartConv

Table 18

函数名	ADC_SoftwareStartConv
函数原形	void ADC_SoftwareStartConv(ADC_TypeDef* ADCx)
功能描述	使能或者失能指定的 ADC 的软件转换启动功能
输入参数	ADCx：选择 ADC 外设
返回值	无

使用示例:

```
/* ADC开始触发 */
ADC_SoftwareStartConv(ADC);
```

ADC_GetConversionValue

Table 19

函数名	ADC_GetConversionValue
函数原形	uint16_t ADC_GetConversionValue(ADC_TypeDef* ADCx)
功能描述	返回最近一次 ADCx 的转换结果
输入参数	ADCx：选择 ADC 外设
返回值	最近一次 ADCx 的转换结果

使用示例:

```
/* 获得返回最近一次ADC的转换结果 */
uint16_t   ADC_Value;
ADC_Value = ADC_GetConversionValue(ADC);
```

ADC_ITConfig

Table 20

函数名	ADC_ITConfig
函数原形	void ADC_ITConfig(ADC_TypeDef* ADCx, uint16_t ADC_IT, FunctionalState NewState)
功能描述	使能或者失能指定的 ADC 的中断
输入参数 1	ADCx：选择 ADC 外设
输入参数 2	ADC_IT：ADC 中断请求，定义在 ADC_IT_TypeDef 中
输入参数 3	NewState：ADC 中断开启/关闭
返回值	无

ADC_IT

ADC_IT 可以用来使能或者失能 ADC 中断。参阅Table 21获得这个参数的所有成员。。

Table 21

ADC_IT	描述
ADC_IT_ADCIF	ADC 中断请求
ADC_IT_INTEN	中断请求CPU的使能控制位
ADC_IT_EOCIE	ADC（每一次)转换完成中断使能位
ADC_IT_EOSIE0	序列采样及转换完成中断使能位
ADC_IT_UPTHIE	上阈值溢出中断使能位
ADC_IT_LOWTHIE	下阈值溢出中断使能位

SC32f10xx、SC32f11xx、SC32f12xx (ADC_IT_ADCIF)
SC32f15xx 可选(ADC_IT_INTEN - ADC_IT_LOWTHIE)

使用示例:

```
/* 使能ADC中断 */
ADC_ITConfig(ADC, ADC_IT_ADCIF,ENABLE);
```

ADC_GetFlagStatus

Table 22

函数名	ADC_GetFlagStatus
函数原形	FlagStatus ADC_GetFlagStatus(ADC_TypeDef* ADCx, uint16_t ADC_FLAG)
功能描述	检查指定 ADC 标志位设置与否
输入参数 1	ADCx：选择 ADC 外设
输入参数 2	ADC_FLAG：指定需检查的标志位
返回值	ADC 标志位状态

ADC_FLAG

Table 23给出了ADC_FLAG的值。

Table 23

ADC_ FLAG	描述
ADC_Flag_ADCIF	ADC 转换完成/中断请求标志
ADC_Flag_EOSIF0	序列采样及转换完成中断标志位
ADC_Flag_BUSY	硬件触发ADC状态位
ADC_Flag_UPTHIF	上阈值溢出标志位
ADC_Flag_LOWTHIF	下阈值溢出标志位
ADC_Flag_OVERRUN	无法及时处理请求而发生溢出标志位

SC32f10xx、SC32f11xx、SC32f12xx (ADC_Flag_ADCIF)
SC32f15xx 可选(ADC_Flag_EOSIF0 - ADC_Flag_OVERRUN)

使用示例:

```
/* 检查ADC标志位设置与否 */
FlagStatus  ADC_Status;
ADC_Status = ADC_GetFlagStatus(ADC, ADC_Flag_ADCIF);
```

ADC_ClearFlag

Table 24

函数名	ADC_ClearFlag
函数原形	void ADC_ClearFlag(ADC_TypeDef* ADCx, uint16_t ADC_FLAG)
功能描述	清除ADCx 的待处理标志位
输入参数 1	ADCx：选择 ADC 外设
输入参数 2	ADC_FLAG：指定需清除的标志位，定义在 ADC_FLAG_TypeDef 中
返回值	无

使用示例:

```
/*清除ADC标志位*/
ADC_ClearFlag(ADC, ADC_Flag_ADCIF);
```

ADC_DMACmd

Table 25

函数名	ADC_DMACmd
函数原形	void ADC_DMACmd(ADC_TypeDef* ADCx, FunctionalState NewState)
功能描述	使能或者失能指定的 ADC 的DMA 请求
输入参数 1	ADCx：选择 ADC 外设
输入参数 2	NewState：ADC 的 DMA 请求开启/关闭

返回值	无

使用示例:

```
/*使能ADC 的 DMA 请求*/
ADC_DMACmd(ADC, ENABLE);
```

ADC_SequenceChannelConfig (适用 SC32f15xx 系列)

Table 26

函数名	ADC_SequenceChannelConfig
函数原形	void ADC_SequenceChannelConfig(ADC_TypeDef* ADCx,ADC_SQRank_Typedef ADC_SQRank,ADC_ChannelTypedef ADC_Channel)
功能描述	配置 ADC 采样序列信号选择
输入参数 1	ADCx：选择 ADC 外设
输入参数 2	ADC_SQRank：ADC 采样序列信号选择
输入参数 3	ADC_Channel：ADC 通道选择
返回值	无

ADC_SQRank

ADC_SQRank 可以用来选择ADC采样序列信号。参阅Table 27获得这个参数的所有成员。。

Table 27

ADC_SQRank	描述
ADC_SQRank_0	ADC 序列通道 0
ADC_SQRank_1	ADC 序列通道 1
ADC_SQRank_2	ADC 序列通道 2
ADC_SQRank_3	ADC 序列通道 3
ADC_SQRank_4	ADC 序列通道 4
ADC_SQRank_5	ADC 序列通道 5
ADC_SQRank_6	ADC 序列通道 6
ADC_SQRank_7	ADC 序列通道 7
ADC_SQRank_8	ADC 序列通道 8
ADC_SQRank_9	ADC 序列通道 9
ADC_SQRank_10	ADC 序列通道 10
ADC_SQRank_11	ADC 序列通道 11
ADC_SQRank_12	ADC 序列通道 12
ADC_SQRank_13	ADC 序列通道 13
ADC_SQRank_14	ADC 序列通道 14
ADC_SQRank_15	ADC 序列通道 15

参数 ADC_Channel 指定了通过函数ADC_SetChannel来设置的 ADC 通道。Table 16列举了 ADC_Channel 可取的值。

使用示例:

```
/* ADC采样序列信号选择 */
ADC_SequenceChannelConfig(ADC, ADC_SQRank_0, ADC_Channel_0);
```

ADC_SequenceGroupConfig(适用 SC32f15xx 系列)

Table 28

函数名	ADC_SequenceGroupConfig
函数原形	void ADC_SequenceGroupConfig(ADC_TypeDef* ADCx,ADC_SQGroup_Typedef ADC_SQGroup,uint8_t SQStr,uint8_t SQCnt)
功能描述	配置 ADC 序列采样位置和采样号
输入参数 1	ADCx：选择 ADC 外设
输入参数 2	ADC_SQGroup：ADC 采样序列
输入参数 3	SQStr：用于设定各序列每次启动时的起始 DSn
输入参数 4	SQCnt：采样序列的采样个数
返回值	无

ADC_SQGroup

ADC_SQGroup 可以用来选择ADC采样序列。参阅Table 29获得这个参数的所有成员。。

Table 29

ADC_SQGroup	描述
ADC_SQGroup_0	ADC 序列组 0
ADC_SQGroup_1	ADC 序列组 1
ADC_SQGroup_2	ADC 序列组 2
ADC_SQGroup_3	ADC 序列组 3

使用示例:

```
/* ADC配置ADC序列采样位置和采样号 */
ADC_SequenceGroupConfig(ADC, ADC_SQGroup_0,0x04 ,0x04);
```

ADC_SetThresholds(适用 SC32f15xx 系列)

Table 30

函数名	ADC_SetThresholds
函数原形	void ADC_SetThresholds(ADC_TypeDef* ADCx, uint16_t HighThreshold,uint16_t LowThreshold)
功能描述	设置上下限阈值
输入参数 1	ADCx：选择 ADC 外设
输入参数 2	HighThreshold：设置上限阈值
输入参数 3	LowThreshold：设置下限阈值
返回值	无

使用示例:

```
/* 设置上下限阈值 */
ADC_SetThresholds(ADC,0xFF,0x00);
```

ADC_ThresholdsChannelConfig(适用 SC32f15xx 系列)

Table 31

函数名	ADC_ThresholdsChannelConfig
函数原形	Void ADC_ThresholdsChannelConfig(ADC_TypeDef* ADCx, uint32_t ADC_Thresholds_Channel, FunctionalState NewState)
功能描述	使能通道使用阈值功能
输入参数 1	ADCx：选择 ADC 外设
输入参数 2	ADC_Thresholds_Channel：选择开启阈值功能的通道
输入参数 3	NewState： ADC 通道使用阈值功能功能开启/关闭，可取值 ENABLE 或 DISABLE
返回值	无

ADC_Thresholds_Channel

ADC_Thresholds_Channel 可以用来选择ADC采样序列信号。参阅Table 32获得这个参数的所有成员。。

Table 32

ADC_Thresholds_Channel	描述
ADC_Thresholds_Channel_Less	ADC 阈值限制通道无
ADC_Thresholds_Channel_0	ADC 阈值限制通道 0
ADC_Thresholds_Channel_1	ADC 阈值限制通道 1
ADC_Thresholds_Channel_2	ADC 阈值限制通道 2
ADC_Thresholds_Channel_3	ADC 阈值限制通道 3
ADC_Thresholds_Channel_4	ADC 阈值限制通道 4
ADC_Thresholds_Channel_5	ADC 阈值限制通道 5
ADC_Thresholds_Channel_6	ADC 阈值限制通道 6
ADC_Thresholds_Channel_7	ADC 阈值限制通道 7
ADC_Thresholds_Channel_8	ADC 阈值限制通道 8
ADC_Thresholds_Channel_9	ADC 阈值限制通道 9

ADC_Thresholds_Channel_10	ADC 阈值限制通道 10
ADC_Thresholds_Channel_11	ADC 阈值限制通道 11
ADC_Thresholds_Channel_12	ADC 阈值限制通道 12
ADC_Thresholds_Channel_13	ADC 阈值限制通道 13
ADC_Thresholds_Channel_14	ADC 阈值限制通道 14
ADC_Thresholds_Channel_15	ADC 阈值限制通道 15
ADC_Thresholds_Channel_ALL	ADC 阈值限制通道 ALL

使用示例:

```
/* 设置上下限阈值 */
ADC_ThresholdsChannelConfig(ADC, ADC_Channel_0, ENABLE);
```

综合使用示例:（ADC 采集 AIN7 电压）

```
RCC_APB2Cmd(ENABLE);  //开启APB2时钟
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ALL, ENABLE); //使能ADC时钟

volatile uint16_t ADC_Value;
 ADC_DeInit(ADC);   // ADC相关寄存器复位至缺省值
    {
            ADC_InitTypeDef ADC_InitStruct;
            ADC_InitStruct.ADC_EAIN = ADC_EAIN_7;      //设定AIN7为ADC输入，并自动将上拉电阻移除
            ADC_InitStruct.ADC_ConvMode = ADC_ConvMode_Single;   //单次转换模式
            ADC_InitStruct.ADC_VREF = ADC_VREF_2_4V;    //基准电压为2.4V
            ADC_InitStruct.ADC_Prescaler = ADC_Prescaler_6CLOCK;  //ADC采样时间约6个系统时钟
            ADC_Init(ADC,&ADC_InitStruct);  //初始化ADC
    }
    ADC_SetChannel(ADC,ADC_Channel_7);   //ADC_CON寄存器内配置AIN7为 ADC 的输入
    ADC_Cmd(ADC,ENABLE);  //使能ADC
 while(1)
    {
            ADC_SoftwareStartConv(ADC);      //开始AD转换
            while(!ADC_GetFlagStatus(ADC,ADC_Flag_ADCIF));  //等待转换完成
            ADC_ClearFlag(ADC,ADC_Flag_ADCIF);         //清除标志位
ADC_Value = ADC_GetConversionValue(ADC); //获取ADC单次转换值
    }
```

4 BTM 固件库函数

SC32F1xxx系列内建一个Base Timer（BTM），可以按照15.625ms ~ 32s的间隔产生中断。32kHz LIRC及外接32.768kHz晶体振荡器LXT都可作为BTM的时钟源。BTM产生的中断可以将 CPU 从 STOP mode唤醒。

BTM 寄存器结构

BTM 寄存器结构，BTM_TypeDef，在文件“sc32f1xxx.h”中定义如下：

```
typedef struct
{
__IO uint32_t BTM_CON;
__IO uint32_t BTM_STS;
} BTM_TypeDef;
```

BTM 寄存器表格

Table 33

寄存器	描述
BTM_CON	低频定时器控制寄存器
BTM_STS	BTM标志位寄存器

BTM 固件库函数列表

Table 34

函数名	描述
BTM_DeInit	BTM 相关寄存器复位至缺省值
BTM_FSConfig	低频时钟中断频率选择
BTM_Cmd	使能或者使能 BTM
BTM_ITConfig	使能或者失能 BTM 中断
BTM_GetFlagStatus	获得 BTM 中断标志状态
BTM_ClearFlag	清除 BTM 中断标志位

BTM 固件库函数详解

BTM_DeInit

Table 35

函数名	BTM_DeInit
函数原型	void BTM_DeInit(BTM_TypeDef* BTMx)
功能描述	BTM 相关寄存器复位至缺省值
输入参数	BTMx：选择 BTM 外设
返回值	无

使用示例:

```
/* BTM寄存器设置为复位值 */
BTM_DeInit(BTM);
```

BTM_FSConfig

Table 36

函数名	BTM_FSConfig
函数原型	void BTM_FSConfig(BTM_TypeDef* BTMx, BTM_FreqSelect_TypeDef BTM_FreqSelect)
功能描述	低频时钟中断频率选择
输入参数 1	BTMx：选择 BTM 外设
输入参数 2	BTM_FreqSelect：BTM 中断频率选择
返回值	无

BTM_FreqSelect

BTM_FreqSelect 可以用来选择BTM进入中断的时间。可以使用Table 37中的一个参数，或者他们的组合。

Table 37

BTM_FreqSelect	描述
BTM_FreqSelect_15_625MS	BTM 每 15.625ms 进入中断
BTM_FreqSelect_31_25MS	BTM 每 31.25ms 进入中断
BTM_FreqSelect_62_5MS	BTM 每 62.5ms 进入中断
BTM_FreqSelect_125MS	BTM 每 125ms 进入中断
BTM_FreqSelect_250MS	BTM 每 250ms 进入中断
BTM_FreqSelect_500MS	BTM 每 500ms 进入中断
BTM_FreqSelect_1S	BTM 每 1s 进入中断
BTM_FreqSelect_2S	BTM 每 2s 进入中断
BTM_FreqSelect_4S	BTM 每 4s 进入中断
BTM_FreqSelect_8S	BTM 每 8s 进入中断
BTM_FreqSelect_16S	BTM 每 16s 进入中断
BTM_FreqSelect_32S	BTM 每 32s 进入中断

使用示例:

```
/*设置BTM溢出时间为250MS*/
BTM_FSConfig(BTM,BTM_FreqSelect_250MS);
```

BTM_Cmd

Table 38

函数名	BTM_Cmd
函数原型	void BTM_Cmd(BTM_TypeDef* BTMx, FunctionalState NewState)
功能描述	使能或者失能 BTM
输入参数 1	BTMx：选择 BTM 外设
输入参数 2	NewState: BTM 外设的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* 使能BTM */
BTM_Cmd(BTM,ENABLE);
```

BTM_ITConfig

Table 39

函数名	BTM_ITConfig
函数原型	void BTM_ITConfig(BTM_TypeDef* BTMx,uint16_t BTM_IT, FunctionalState NewState)
功能描述	使能或者失能 BTM 中断
输入参数 1	BTMx：选择 BTM 外设
输入参数 2	BTM_IT ：BTM 中断请求，定义在 BTM_IT_TypeDef 中
输入参数 3	NewState：BTM 中断使能或失能，可取值 ENABLE 或 DISABLE
返回值	无

BTM_IT

BTM_IT 可以用来使能或者失能 BTM 中断。可以使用Table 40中的一个参数，或者他们的组合。

Table 40

BTM_IT	描述
BTM_IT_INT	BTM 中断请求

使用示例:

```
/* 使能BTM中断 */
BTM_ITConfig(BTM,BTM_IT_INT,ENABLE);
```

BTM_GetFlagStatus

Table 41

函数名	BTM_GetFlagStatus
函数原型	FlagStatus BTM_GetFlagStatus(BTM_TypeDef* BTMx, BTM_FLAG_TypeDef BTM_FLAG)
功能描述	获得 BTM 中断标志状态
输入参数 1	BTMx：选择 BTM 外设
输入参数 2	BTM_FLAG：指定需检查的标志位
返回值	BTM 标志位状态

BTM_FLAG

Table 42给出了BTM_FLAG的值。

Table 42

BTM_ FLAG	描述
BTM_FLAG_IF	BTM 中断请求标志

使用示例:

```
/* 获得BTM中断标志位*/
FlagStatus BTM_Status;
BTM_Status = BTM_GetFlagStatus(BTM,BTM_FLAG_IF);
```

BTM_ClearFlag

Table 43

函数名	BTM_ClearFlag
函数原型	void BTM_ClearFlag(BTM_TypeDef* BTMx, BTM_FLAG_TypeDef BTM_FLAG)
功能描述	清除 BTM 中断标志位
输入参数 1	BTMx：选择 BTM 外设
输入参数 2	BTM_FLAG：指定需检查的标志位，定义在 BTM_FLAG_TypeDef 中
返回值	无

使用示例:

```
/* 清除BTM中断标志位 */
BTM_ClearFlag(BTM,BTM_FLAG_IF);
```

综合使用示例:（BTM 每 250ms 产生一个中断）

```
BTM_DeInit(BTM);  //BTM寄存器复位至缺省值
BTM_FSConfig(BTM,BTM_FreqSelect_250MS); //设置BTM中断时间为250MS
BTM_Cmd(BTM,ENABLE);  //使能BTM

BTM_ITConfig(BTM,BTM_IT_INT,ENABLE); //使能BTM中断
NVIC_EnableIRQ(BTM_IRQn);

void BTM_IRQHandler()
{
BTM_ClearFlag(BTM,BTM_FLAG_IF);    //清中断标志
}
```

5 CMP 固件库函数

SC32F1xxx 系列内建一个模拟比较器（CMP），CMP中断可唤醒STOP Mode。可用于报警器电路、电源电压监测电路、过零检测电路等。

此比较器具有四个模拟信号正输入端：CMP0~3，可通过CMPIS [1:0]切换选择。负输入端电压可通CMPRF[3:0]切换为CMPR脚上的外部电压或内部的16档比较电压中的一种。

通过CMPIM[1:0]可以方便的设定比较器的中断模式，当CMPIM[1:0]所设定的中断条件发生时比较器中断标志CMPIF会被置1，该中断标志需要软件清除。

CMP 寄存器结构

CMP 寄存器结构，CMP_TypeDef，在文件“sc32f1xxx.h”中定义如下：

```
typedef struct
{
__IO uint32_t CMP_STS;
__IO uint32_t CMP_CFG;
} CMP_TypeDef;
```

在一些型号如（SC32F15Gx）中定义如下：

```
typedef struct
{
__IO uint32_t CMP_STS;
__IO uint32_t CMP_CON;
__IO uint32_t CMP_IDE;
__IO uint32_t CMP_CFG;
} CMP_TypeDef;
```

CMP 寄存器表格

Table 44

寄存器	描述
CMP_STS	模拟比较器状态寄存器
CMP_CON	比较器控制寄存器
CMP_IDE	比较器0/1/2中断使能寄存器
CMP_CFG	模拟比较器配置寄存器

CMP 固件库函数列表

Table 45

函数名	描述
CMP_DeInit	CMP 相关寄存器复位至缺省值
CMP_Init	根据 CMP_InitStruct 中指定的参数初始化外设 CMP 寄存器
CMP_StructInit	把 CMP_InitStruct 中的每一个参数按缺省值填入
CMP_Cmd	使能或失能 CMP 外设
CMP_SetNegativeChannel	配置 CMP 负输入通道
CMP_GetNegativeChannel	获取 CMP 负输入通道
CMP_SetPositiveChannel	配置 CMP 正输入通道
CMP_GetPositiveChannel	获取 CMP 正输入通道
CMP_NegativeSelection	配置模拟比较器负端比较电压
CMP_GetNegativeSelection	获取模拟比较器负端比较电压
CMP_ITConfig	配置 CMP 中断使能
CMP_GetCMPSTA	获得模拟比较器输出状态
CMP_GetFlagStatus	获得 CMP 标志状态
CMP_ClearFlag	清除标志状态

CMP_DeInit

Table 46

函数名	CMP_DeInit
函数原型	void CMP_DeInit(CMP_TypeDef* CMPx)
功能描述	CMP 相关寄存器复位至缺省值
输入参数	CMPx：选择 CMPx 外设
返回值	无

使用示例:

```
/* CMP 寄存器设置为复位值 */
  CMP_DeInit(CMP);
```

CMP_Init

Table 47

函数名	CMP_Init
函数原型	void CMP_Init(CMP_TypeDef* CMPx, CMP_InitTypeDef* CMP_InitStruct)
功能描述	根据 CMP_InitStruct 中指定的参数初始化外设 CMP 寄存器
输入参数 1	CMPx：选择 CMPx 外设
输入参数 2	CMP_InitStruct：指向结构 CMP_InitTypeDef 的指针，包含了 CMPx 外设的配置信息
返回值	无

CMP_InitTypeDef

CMP_InitTypeDef 定义于文件“sc32f1xxx_cmp.h”的所有成员：

```
typedef struct
{
    uint16_t CMP_Negative;
    uint16_t CMP_Positive;
    uint16_t CMP_VREF;
    uint16_t CMP_CMPRF;
    uint16_t CMP_TriggerMode;
    uint16_t CMP_HYS;
} CMP_InitTypeDef;
```

CMP_Negative

CMP_Negative设置模拟比较器负端比较电压，参阅Table 48获得这个参数的所有成员。

Table 48

CMP_Negative	描述
CMP_Negative_CMPR	选用 CMPR 为模拟比较器负端的比较电压
CMP_Negative_1D16VDD	选用 1/16VDD 为模拟比较器负端的比较电压
CMP_Negative_2D16VDD	选用 2/16VDD 为模拟比较器负端的比较电压
CMP_Negative_3D16VDD	选用 3/16VDD 为模拟比较器负端的比较电压
……	……
CMP_Negative_13D16VDD	选用 13/16VDD 为模拟比较器负端的比较电压
CMP_Negative_14D16VDD	选用 14/16VDD 为模拟比较器负端的比较电压
CMP_Negative_15D16VDD	选用 15/16VDD 为模拟比较器负端的比较电压
CMP3_Negative_1D16VDD	选用 1/16VDD 为模拟比较器负端的比较电压
CMP3_Negative_2D16VDD	选用 2/16VDD 为模拟比较器负端的比较电压
CMP3_Negative_3D16VDD	选用 3/16VDD 为模拟比较器负端的比较电压
……	……
CMP3_Negative_13D16VDD	选用 13/16VDD 为模拟比较器负端的比较电压
CMP3_Negative_14D16VDD	选用 14/16VDD 为模拟比较器负端的比较电压
CMP3_Negative_15D16VDD	选用 15/16VDD 为模拟比较器负端的比较电压

SC32f10xx、SC32f11xx、SC32f12xx可选(CMP_Negative_CMPR - CMP_Negative_15D16VDD)
SC32f15xx可选(CMP3_Negative_1D16VDD - CMP3_Negative_15D16VDD)

CMP_Positive

CMP_Positive设置模拟比较器正端输入电压，参阅Table 49获得这个参数的所有成员。

Table 49

CMP_ Positive	描述
CMP_Positive_CMP0	选用 CMP0 为模拟比较器正端的输入
CMP_Positive_CMP1	选用 CMP1 为模拟比较器正端的输入
CMP_Positive_CMP2	选用 CMP2 为模拟比较器正端的输入
CMP_Positive_CMP3	选用 CMP3 为模拟比较器正端的输入
CMP_Positive_1_5V	模拟比较器正端的输入为内部 1.5V 基准电压
CMP_Positive_CMPP	选用 CMPP 为模拟比较器正端的输入
CMP_Positive_OP1O	选用 OP1O 为模拟比较器正端的输入
CMP_Positive_OP2O	选用 OP2O 为模拟比较器正端的输入

SC32f10xx、SC32f11xx、SC32f12xx可选(CMP_Positive_CMP0 - CMP_Positive_CMP3)
SC32f15xx可选(CMP_Positive_CMP0 - CMP_Positive_OP2O)

CMP_ VREF（SC32F15xx系列可选）

CMP_ VREF设置模拟比较器正端输入电压，参阅Table 50获得这个参数的所有成员。

Table 50

CMP_ VREF	描述
CMP_RefSource_VDD	选用 VDD 为模拟比较器参考电压
CMP_RefSource_VREF	选用 VREF 为模拟比较器参考电压
CMP0_1_2RefSource_VDD	CMP0_1_2 参考电压 VDD
CMP0_1_2RefSource_VREF	CMP0_1_2 参考电压 VREF
CMP3_RefSource_VDD	CMP3 参考电压 VDD
CMP3_RefSource_VREF	CMP3 参考电压 VREF

CMP_TriggerMode

CMP_TriggerMode设置模拟比较器中断模式，参阅Table 51获得这个参数的所有成员。

Table 51

CMP_TriggerMode	描述
CMP_TriggerMode_Disable	不产生中断
CMP_TriggerMode_RISE	上升沿中断
CMP_TriggerMode_FALL	下降沿中断
CMP_TriggerMode_RISE_FALL	双沿中断
CMP0_1_2TriggerMode_Disable	CMP0_1_2 不产生中断
CMP0_1_2TriggerMode_RISE	CMP0_1_2 上升沿中断
CMP0_1_2TriggerMode_FALL	CMP0_1_2 下降沿中断
CMP0_1_2TriggerMode_RISE_FALL	CMP0_1_2 双沿中断
CMP3_TriggerMode_Disable	CMP3 不产生中断
CMP3_TriggerMode_RISE	CMP3 上升沿中断
CMP3_TriggerMode_FALL	CMP3 下降沿中断
CMP3_TriggerMode_RISE_FALL	CMP3 双沿中断

SC32f10xx、SC32f11xx、SC32f12xx可选(CMP_TriggerMode - CMP_TriggerMode_RISE_FALL)
SC32f15xx可选(CMP0_1_2TriggerMode_Disable - CMP3_TriggerMode_RISE_FALL)

CMP_ HYS（SC32F15xx系列可选）

CMP_ HYS设置迟滞（回差）电压，参阅Table 52获得这个参数的所有成员。

Table 52

CMP_HYS	描述

CMP_HYS_0mV	迟滞（回差）电压 0mV
CMP_HYS_5mV	迟滞（回差）电压 5mV
CMP_HYS_10mV	迟滞（回差）电压 10mV
CMP_HYS_20mV	迟滞（回差）电压 20mV

使用示例:

```
/* 根据CMP_InitStruct中指定的参数初始化外设CMP */
CMP_InitTypeDef CMP_InitStruct;
CMP_InitStruct->CMP_Negative    = CMP_Negative_CMPN; //选择负输入端CMPN
CMP_InitStruct->CMP_Positive    = CMP_Positive_CMPP;//选择正输入端CMPP
CMP_InitStruct->CMP_VREF        = CMP_RefSource_VDD;// 选择参考电压VDD
CMP_InitStruct->CMP_CMPRF = CMP_CMPRF_1D16CMP_VREF; //选用 1/16VDD 为CMP负端的比较电压
CMP_InitStruct->CMP_TriggerMode  = CMP_TriggerMode_ RISE; //上升沿中断
CMP_InitStruct->CMP_HYS         = CMP_HYS_0mV;// 迟滞（回差）电压0mV
CMP_Init(CMP,&CMP_InitStruct);
```

CMP_StructInit

Table 53

函数名	CMP_StructInit
函数原形	void CMP_StructInit(CMP_InitTypeDef* CMP_InitStruct)
功能描述	把 CMP_InitStruct 中的每一个参数按缺省值填入
输入参数	CMP_InitStruct：指向结构 CMP_InitTypeDef 的指针，待初始化
返回值	无

CMP_InitStruct

CMP_InitStruct的成员缺省值如Table 54所示。

Table 54

成员	缺省值
CMP_Negative	CMP_Negative_CMPR
CMP_Positive	CMP_Positive_CMPP
CMP_CMPRF	CMP_CMPRF_1D16CMP_VREF
CMP_VREF	CMP_RefSource_VDD
CMP_HYS	CMP_HYS_0mV
CMP_TriggerMode	CMP_TriggerMode_Disable

（注：SC32f15xx系列有CMP_CMPRF、CMP_VREF、CMP_HYS）

使用示例:

```
/* 把 CMP_InitStruct 中的每一个参数按缺省值填入 */
CMP_InitTypeDef  CMP_InitStruct;
CMP_StructInit(&CMP_InitStruct);
```

CMP_Cmd

Table 55

函数名	CMP_Cmd
函数原型	void CMP_Cmd(CMP_TypeDef* CMPx, FunctionalState NewState)
功能描述	使能或失能 CMP
输入参数 1	CMPx：选择 CMPx 外设
输入参数 2	NewState：CMP 功能开启/关闭，可取值 ENABLE 或 DISABLE
返回值	无

使用示例:

```
/* 使能CMP功能 */
CMP_Cmd(CMP,ENABLE);
```

CMP_NegativeSelection (适用 SC32f15xx 系列)

Table 56

函数名	CMP_NegativeVoltage
函数原型	void CMP_NegativeVoltage(CMP_TypeDef* CMPx, CMP_CMPRF_TypeDef CMP_CMPRF)
功能描述	配置 CMP 负输入通道电压
输入参数 1	CMPx：选择 CMPx 外设
输入参数 2	CMP_CMPRF：CMP 负输入通道电压选择
返回值	无

CMP_CMPRF_TypeDef可参阅Table 48获得这个类型的所有成员

使用示例:

```
/* 配置CMP负输入通道参考电压 */
CMP_ NegativeVoltage (CMP, CMP_Negative_1D16VDD);
```

CMP_GetNegativeSelection (适用 SC32f15xx 系列)

Table 57

函数名	CMP_GetNegativeSelection
函数原型	CMP_NegativeSelect_TypeDef CMP_GetNegativeSelection (CMP_TypeDef* CMPx)
功能描述	获取 CMP 负输入通道
输入参数	CMPx：选择 CMPx 外设
返回值	CMP 负输入通道参考电压

使用示例:

```
/* 获取CMP负输入通道参考电压 */
CMP_GetNegativeChannel(CMP);
```

CMP_SetNegativeChannel

Table 58

函数名	CMP_SetNegativeChannel
函数原型	void CMP_SetNegativeChannel(CMP_TypeDef* CMPx, CMP_Negative_TypeDef CMP_Negative_Channel)
功能描述	配置 CMP 负输入通道
输入参数 1	CMPx：选择 CMPx 外设
输入参数 2	CMP_Negative_Channel：CMP 负输入通道选择
返回值	无

CMP_Negative

CMP_Negative_TypeDef 可参阅Table 48获得这个类型的所有成员

使用示例:

```
/* 配置CMP负输入通道 */
CMP_SetNegativeChannel(CMP, CMP_Negative_1D16VDD);
```

CMP_GetNegativeChannel

Table 59

函数名	CMP_GetNegativeChannel
函数原型	CMP_Negative_TypeDef CMP_GetNegativeChannel(CMP_TypeDef* CMPx)
功能描述	获取 CMP 负输入通道
输入参数	CMPx：选择 CMPx 外设
返回值	CMP 负输入通道

使用示例:

```
/* 获取CMP负输入通道 */
CMP_GetNegativeChannel(CMP);
```

CMP_SetPositiveChannel

Table 60

函数名	CMP_SetPositiveChannel
函数原型	void CMP_SetPositiveChannel(CMP_TypeDef* CMPx, CMP_Positive_TypeDef CMP_Positive_Channel)
功能描述	配置 CMP 正输入通道
输入参数 1	CMPx：选择 CMPx 外设
输入参数 2	CMP_Positive_Channel：CMP 正输入通道选择
返回值	无

CMP_Positive

CMP_Positive_TypeDef 可参阅Table 49获得这个类型的所有成员

使用示例:

```
/* 配置CMP0正输入通道 */
CMP_SetPositiveChannel(CMP0, CMP_Positive_CMP0);
```

CMP_GetPositiveChannel

Table 61

函数名	CMP_GetPositiveChannel
函数原型	CMP_Positive_TypeDef CMP_GetPositiveChannel(CMP_TypeDef* CMPx)
功能描述	获取 CMP 正输入通道
输入参数	CMPx：选择 CMPx 外设
返回值	CMP 正输入通道

使用示例:

```
/* 获取CMP正输入通道 */
CMP_GetPositiveChannel(CMP);
```

CMP_ GetCMPSTA

Table 62

函数名	CMP_GetCMPSTA
函数原型	CMP_CMPSTA_TypeDef CMP_GetCMPSTA(CMP_TypeDef* CMPx)
功能描述	获得模拟比较器输出状态
输入参数	CMPx：选择 CMPx 外设
返回值	模拟比较器输出状态

CMP_CMPSTA_TypeDef

CMP_CMPSTA_TypeDef 返回模拟比较器输出状态，参阅Table 63获得这个类型的所有成员。

Table 63

成员	状态
CMP_CMPSTA_Low	比较器正端电压小于负端电压
CMP_CMPSTA_High	比较器正端电压大于负端电压

使用示例:

```
/* 获得模拟比较器输出状态 */
CMP_CMPSTA_TypeDef CMP_Type;
CMP_Type = CMP_GetCMPSTA(CMP);
```

CMP_GetFlagStatus

Table 64

函数名	CMP_GetFlagStatus
函数原型	FlagStatus CMP_GetFlagStatus(CMP_TypeDef* CMPx, CMP_FLAG_TypeDef CMP_FLAG)
功能描述	获得 CMP 标志状态
输入参数 1	CMPx：选择 CMPx 外设
输入参数 2	CMP_FLAG：CMP 标志位选择

返回值	CMP_FLAG 的新状态（SET 或者 RESET）

CMP_ FLAG

Table 65给出了CMP_FLAG的值。

Table 65

CMP_ FLAG	描述
CMP_FLAG_IF	模拟比较器中断标志位
CMP_FLAG_CMP0	CMP0 中断标志位
CMP_FLAG_CMP1	CMP1 中断标志位
CMP_FLAG_CMP2	CMP2 中断标志位
CMP_FLAG_CMP3	CMP3 中断标志位

SC32f10xx、SC32f11xx、SC32f12xx可选(CMP_FLAG_IF)
SC32f15xx可选(CMP_FLAG_CMP0 - CMP_FLAG_CMP3)

使用示例:

```
/* 获得CMP标志状态并返回 */
FlagStatus CMP_Status;
CMP_Status = CMP_GetFlagStatus(CMP,CMP_FLAG_IF);
```

CMP_ITConfig (该函数 SC32f15xx 可用)

Table 66

函数名	CMP_ITConfig
函数原型	void CMP_ITConfig(CMP_TypeDef* CMPx, uint16_t CMP_IT, FunctionalState NewState)
功能描述	使能 CMP 中断
输入参数 1	CMPx：选择 CMPx 外设
输入参数 2	CMP_IT：CMP 中断选择
输入参数 3	NewState：CMP 中断使能。可取值 ENABLE 或 DISABLE
返回值	无

Table 65给出了CMP_FLAG的值。

Table 67

CMP_IT	描述
CMP0_1_2IT_INT	中断使能
CMP0_1_2IT_CMP0	CMP0 中断
CMP0_1_2IT_CMP1	CMP1 中断
CMP0_1_2IT_CMP2	CMP2 中断
CMP3_IT_INT	CMP3 中断

使用示例:

```
/* CMP3中断使能 */
CMP_ITConfig (CMP, CMP3_IT_INT);
```

CMP_ClearFlag

Table 68

函数名	CMP_ClearFlag
函数原型	void CMP_ClearFlag(CMP_TypeDef* CMPx, CMP_FLAG_TypeDef CMP_FLAG)
功能描述	清除标志状态
输入参数 1	CMPx：选择 CMPx 外设
输入参数 2	CMP_FLAG：中断标志位选择
返回值	无

使用示例:

```
/* 清除CMP标志*/
CMP_GetFlagStatus(CMP, CMP_FLAG_IF);
```

综合使用示例:（获取模拟比较器输出状态并开启上升沿中断）

```
CMP_DeInit(CMP);  //CMP相关寄存器复位至缺省值
{
CMP_InitTypeDef CMP_InitStruct;
CMP_InitStruct.CMP_Negative = CMP_Negative_1D16VDD; //选用 1/16VDD 为模拟比较器负端的比较电压
CMP_InitStruct.CMP_Positive = CMP_Positive_CMP0; //选用 CMP0 为模拟比较器正端的输入
CMP_InitStruct.CMP_TriggerMode = CMP_TriggerMode_RISE;  //上升沿中断
CMP_Init(CMP,&CMP_InitStruct); //初始化CMP
}
CMP_Cmd(CMP,ENABLE);       //使能CMP
NVIC_EnableIRQ(CMP_IRQn);  //开中断
volatile CMP_CMPSTA_TypeDef CMP_Type;
while(1)
{    CMP_Type = CMP_GetCMPSTA(CMP);   } //获取模拟比较器输出状态
void CMP_IRQHandler()      //中断服务函数
{    CMP_ClearFlag(CMP,CMP_FLAG_IF); } //清中断标志
```

6 CRC 固件库函数

SC32F1xxx系列内建一个CRC校验模块，使用多项式发生器从一个8 位/16 位/32位的数据字中产生CRC 码。在众多的应用中，基于CRC的技术还常用来验证数据传输或存储的完整性。根据功能安全标准的规定，这些技术提供了验证 Flash 完整性的方法。CRC计算单元有助于在运行期间计算软件的签名，并将该签名与链接时生成并存储在指定存储单元的参考签名加以比较。

CRC 寄存器结构

CRC 寄存器结构，CRC_TypeDef，在文件“sc32f1xxx.h”中定义如下：

```
typedef struct
{
__IO uint32_t CRC_DR;
__IO uint32_t CRC_CON;
__IO uint32_t CRC_INT;
__IO uint32_t CRC_POL;
} CRC_TypeDef;
```

CRC 寄存器表格

Table 69

寄存器	描述
CMP_DR	CRC 数据寄存器
CMP_CON	CRC 控制寄存器
CMP_INT	CRC 初始值寄存器
CMP_POL	CRC 多项式设置寄存器

CRC 固件库函数列表

Table 70

函数名	描述
CRC_DeInit	CRC 相关寄存器复位至缺省值
CRC_Init	根据 CRC_InitStruct 中指定的参数初始化外设 CRC 寄存器
CRC_PolynomialSizeSelect	设置 CRC 多项式大小
CRC_SetInitRegister	设置 CRC 初值大小
CRC_SetPolynomial	设置 POL 多项式系数大小
CRC_ResetDR	重置 CRCDR 寄存器。
CRC_CalcCRC	获取 32 位 CRC 校验值
CRC_CalcCRC16bits	获取 16 位 CRC 校验值
CRC_CalcCRC8bits	获取 8 位 CRC 校验值
CRC_GetCRC	读出 CRC 计算结果
CRC_Accumulate	累计计算 CRC 校验值
CRC_Calculate	从第一个数开始计算 CRC 校验值
CRC_Handle_8	从第一个数开始计算 8 位 CRC 校验值
CRC_Handle_16	从第一个数开始计算 16 位 CRC 校验值

CRC 固件库函数详解

CRC_DeInit

Table 71

函数名	CRC_DeInit
函数原型	void CRC_DeInit(void)
功能描述	CRC 相关寄存器复位至缺省值

输入参数	无
返回值	无

使用示例:

```
/* CMP寄存器设置为复位值 */
CRC_DeInit();
```

CRC_Init

Table 72

函数名	CRC_Init
函数原型	void CRC_Init(CRC_InitTypeDef* CRC_InitStruct)
功能描述	根据 CRC_InitStruct 中指定的参数初始化外设 CRC 寄存器
输入参数	CRC_InitStruct：指向结构 CRC_InitTypeDef 的指针，包含了外设 CRC 的配置信息
返回值	无

CRC_InitTypeDef

CRC_InitTypeDef 定义于文件“sc32f1xxx_crc.h”：

```
typedef struct
{
uint16_t DefaultPolynomialUse;
uint16_t DefaultInitValueUse;
uint32_t GeneratingPolynomial;
uint32_t InitValue;
uint32_t CRCSize;
} CRC_InitTypeDef;
```

DefaultPolynomialUse

DefaultPolynomialUse设置可编程多项式POL是否为默认值，参阅Table 73获得这个参数的所有成员。

Table 73

DefaultPolynomialUse	描述
DEFAULT_Polynomial_Enable	选择 POL 为默认值 0x04C11DB7
DEFAULT_Polynomial_Disable	选择 POL 不为默认值

DefaultInitValueUse

DefaultInitValueUse设置可编程CRC初始值是否为默认值，参阅Table 74获得这个参数的所有成员。

Table 74

DefaultInitValueUse	描述
DEFAULT_InitValue_Enable	选择初始值为默认值 0xFFFFFFFF
DEFAULT_InitValue_Disable	选择初始值不为默认值

GeneratingPolynomial

选择POL不为默认值时，合理写入要用于CRC 计算的多项式系数GeneratingPolynomial。

InitValue

选择CRC初始值不为默认值时，合理写入初始值InitValue。

CRCSize

CRCSize设置多项式的大小，参阅Table 75获得这个参数的所有成员。

Table 75

CRCSize	描述
CRC_POLYSIZE_7B	7 位多项式
CRC_POLYSIZE_8B	8 位多项式
CRC_POLYSIZE_16B	16 位多项式
CRC_POLYSIZE_32B	32 位多项式

使用示例:

```
/*根据CRC_InitStructure中指定的参数初始化CRC */
CRC_InitTypeDef CRC_InitStruct;
```

```
CRC_InitStruct.DefaultPolynomialUse = DEFAULT_Polynomial_Disable; //选择POL不为默认值
CRC_InitStruct.DefaultInitValueUse = DEFAULT_InitValue_Disable; //选择初始值不为默认值
CRC_InitStruct.CRCSize = CRC_POLYSIZE_16B; //16 位多项式
CRC_InitStruct.InitValue = 0x0000;  //初始值为0
CRC_InitStruct.GeneratingPolynomial = 0x8005; //POL多项式系数为8005
CRC_Init(&CRC_InitStruct);
```

CRC_ PolynomialSizeSelect

Table 76

函数名	CRC_PolynomialSizeSelect
函数原形	void CRC_PolynomialSizeSelect(CRC_POLYSIZE_TypeDef CRC_PolSize)
功能描述	设置 CRC 多项式的大小
输入参数	CRC_PolSize：CRC 多项式的大小，参见 Table 75
返回值	无

使用示例:

```
/* 设置CRC多项式的大小为32 */
CRC_PolynomialSizeSelect(CRC_POLYSIZE_32B);
```

CRC_ SetInitRegister

Table 77

函数名	CRC_SetInitRegister
函数原形	void CRC_SetInitRegister(uint32_t CRC_InitValue)
功能描述	设置 CRC 初值大小
输入参数	CRC_InitValue：CRC 初值
返回值	无

使用示例:

```
/* 设置CRC初值大小 */
CRC_SetInitRegister(0xFFFFFFFF);
```

CRC_ SetPolynomial

Table 78

函数名	CRC_SetPolynomial
函数原形	void CRC_SetPolynomial(uint32_t CRC_Pol)
功能描述	设置 POL 多项式系数大小
输入参数	CRC_Pol：POL 多项式系数
返回值	无

使用示例:

```
/* 设置POL多项式系数 */
CRC_SetPolynomial(0x04C11DB7);
```

CRC_ ResetDR

Table 79

函数名	CRC_ResetDR
函数原型	void CRC_ ResetDR(void)
功能描述	重置 CRCDR 寄存器
输入参数	无
返回值	无

使用示例:

```
/* 重置 CRC CRCDR寄存器 */
CRC_RstCRCDR();
```

CRC_ CalcCRC

Table 80

函数名	CRC_CalcCRC
函数原型	uint32_t CRC_CalcCRC(uint32_t CRC_Data)
功能描述	获取 32 位 CRC 校验值
输入参数	CRC_Data：需要 CRC 校验的数据
返回值	32 位 CRC 校验值

使用示例:

```
/* 获取32位CRC校验值 */
uint32_t CRC_Value1；
CRC_Value1 = CRC_CalcCRC(0xAA);
```

CRC_ CalcCRC16bits

Table 81

函数名	CRC_CalcCRC16bits
函数原型	uint32_t CRC_CalcCRC16bits(uint16_t CRC_Data)
功能描述	获取 16 位 CRC 校验值
输入参数	CRC_Data：需要 CRC 校验的数据
返回值	16 位 CRC 校验值

使用示例:

```
/* 获取16位CRC校验值 */
uint32_t CRC_Value2；
CRC_Value2 = CRC_CalcCRC16bits(0xAA);
```

CRC_ CalcCRC8bits

Table 82

函数名	CRC_CalcCRC8bits
函数原型	uint32_t CRC_CalcCRC8bits(uint8_t CRC_Data)
功能描述	获取 8 位 CRC 校验值
输入参数	CRC_Data：需要 CRC 校验的数据
返回值	8 位 CRC 校验值

使用示例:

```
/* 获取8位CRC校验值 */
uint32_t CRC_Value3；
CRC_Value3 = CRC_CalcCRC8bits(0xAA);
```

CRC_ GetCRC

Table 83

函数名	CRC_GetCRC
函数原型	uint32_t CRC_GetCRC(void)
功能描述	读出 CRC 计算结果
输入参数	无
返回值	CRC 计算结果

使用示例:

```
/* 读出CRC计算结果 */
uint32_t CRC_Value4；
CRC_Value4 = CRC_GetCRC();
```

CRC_ Accumulate

Table 84

函数名	CRC_Accumulate

函数原型	uint32_t CRC_Accumulate(CRC_InputData_Format_TypeDef InputDataFormat, uint32_t pBuffer[], uint32_t BufferLength)
功能描述	累计计算 CRC 校验值
输入参数 1	InputDataFormat：输入的待校验数据类型，参见 Table 20
输入参数 2	pBuffer[]：待校验数组
输入参数 3	BufferLength：待校验数组长度
返回值	CRC 计算结果

InputDataFormat

InputDataFormat设置输入的待校验数据类型，参阅Table 85获得这个参数的所有成员。

Table 85

InputDataFormat	描述
CRC_InputData_Format_BYTES	8 位数据
CRC_InputData_Format_HALFWORDS	16 位数据
CRC_InputData_Format_WORDS	32 位数据

CRC_Calculate

Table 86

函数名	CRC_ Calculate
函数原型	uint32_t CRC_Calculate(CRC_InputData_Format_TypeDef InputDataFormat, uint32_t pBuffer[], uint32_t BufferLength)
功能描述	从第一个数开始计算 CRC 校验值
输入参数 1	InputDataFormat：输入的待校验数据类型，参见 Table 20
输入参数 2	pBuffer[]：待校验数组
输入参数 3	BufferLength：待校验数组长度
返回值	CRC 计算结果

使用示例:

```
/* 从CRC_Buff数组第一个数开始计算8个数据的CRC校验值后，
从第9个数据开始累计计算完全部16个数据的CRC校验值 */
volatile uint32_t CRC_Result;
uint8_t CRC_Buff[16] = {0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88,0x99,0xaa,0xbb,0xcc,0xdd
,0xee,0xff,0x00};
CRC_Result = CRC_Calculate(CRC_InputData_Format_BYTES,(uint32_t *)CRC_Buff,8);
CRC_Result = CRC_Accumulate(CRC_InputData_Format_BYTES,(uint32_t *)(CRC_Buff + 8),8);
```

CRC_Handle_8

Table 87

函数名	CRC_Handle_8
函数原型	static uint32_t CRC_Handle_8(uint8_t pBuffer[], uint32_t BufferLength)
功能描述	从第一个数开始计算 8 位 CRC 校验值
输入参数 1	pBuffer[]：待校验数组
输入参数 2	BufferLength：待校验数组长度
返回值	CRC 计算结果

使用示例:

```
/* 从CRC_Buff数组第一个数开始计算8个8位数据的CRC校验值*/
volatile uint32_t CRC_Result;
uint8_t CRC_Buff[8] = {0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88};
CRC_Result = CRC_Handle_8(CRC_Buff, BufferLength);
```

CRC_Handle_16

Table 88

函数名	CRC_Handle_16

函数原型	static uint32_t CRC_Handle_16(uint16_t pBuffer[], uint32_t BufferLength)
功能描述	从第一个数开始计算 16 位 CRC 校验值
输入参数 1	pBuffer[]：待校验数组
输入参数 2	BufferLength：待校验数组长度
返回值	CRC 计算结果

使用示例:

```
/* 从CRC_Buff数组第一个数开始计算8个16位数据的CRC校验值*/
volatile uint32_t CRC_Result;
uint16_t CRC_Buff[8] = {0x1111,0x2222,0x3333,0x4444,0x5555,0x6666,0x7777,0x8888};
CRC_Result = CRC_Handle_16(CRC_Buff, BufferLength);
```

综合使用示例（获取 32 位 CRC 校验值）

```
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_CRC,ENABLE);  //开启CRC时钟
CRC_DeInit();  //CRC相关寄存器复位至缺省值
{
CRC_InitTypeDef CRC_InitStruct;
CRC_InitStruct.DefaultPolynomialUse = DEFAULT_Polynomial_Enable; //选择Pol为默认值0x04C11DB7
CRC_InitStruct.DefaultInitValueUse = DEFAULT_InitValue_Enable;  //选择初始值为默认值0xFFFFFFFF
CRC_InitStruct.CRCSize = CRC_POLYSIZE_32B; //32位多项式
CRC_Init(&CRC_InitStruct);
}
volatile uint32_t CRC_Result;
uint32_t CRC_Buff[16] = {0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88,0x99,
0xaa,0xbb,0xcc,0xdd,0xee,0xff,0x00};
CRC_Result = CRC_Calculate(CRC_InputData_Format_WORDS,CRC_Buff,16); //获取CRC校验值
```

7 DMA 固件库函数、

直接存储器访问(DMA)控制器用于高速数据传输。DMA控制器可以从一个地址到另一个地址传输数据，无需CPU介入。通过DMA进行数据传输可减少CPU的工作量，将节省下的CPU资源做其他应用。DMA控制器包含4个通道，每个通道都直接连接专用的硬件DMA请求，每个通道都同样支持软件触发。DMA控制器支持4级通道优先级，用于处理DMA请求间的优先级，确保同一时刻只有一个DMA通道工作。DMA控制器也支持单一传输和批量传输，请求源可以是软件请求或接口请求，内存之间的数据传输是使用软件请求。

DMA 寄存器结构

DMA 寄存器结构，DMA_TypeDef，在文件“sc32f1xxx.h”中定义如下：

```
typedef struct
{
__IO uint32_t DMA_SADR;
__IO uint32_t DMA_DADR;
__IO uint32_t DMA_CFG;
__IO uint32_t DMA_CNT;
__IO uint32_t DMA_STS;
} DMA_TypeDef;
```

DMA 寄存器表格

Table 89

寄存器	描述
DMA_SADR	DMA 源地址缓存寄存器
DMA_DADR	DMA 目标地址缓存寄存器
DMA_CFG	DMA 控制/配置寄存器
DMA_CNT	DMA 计数器缓存寄存器
DMA_STS	DMA 状态寄存器

DMA 固件库函数列表

Table 90

函数名	描述
DMA_DeInit	将外设 DMAx 寄存器重设为缺省值
DMA_Init	根据 DMA_InitStruct 中指定的参数初始化外设 DMAx 寄存器
DMA_StructInit	把 DMA_InitStruct 中的每一个参数按缺省值填入
DMA_Cmd	使能或者失能 DMA 外设
DMA_PauseCmd	使能或失能指定 DMA 的传输暂停
DMA_CHRQCmd	DMA 请求 DMA 使能
DMA_ChannelReset	复位指定 DMA 的传输通道
DMA_SetSrcAddress	设置指定 DMA 的源地址
DMA_SetDstAddress	设置指定 DMA 的目标地址
DMA_SetCurrDataCounter	设置指定 DMA 的传输数量
DMA_GetCurrDataCounter	获得指定 DMA 的传输数量
DMA_SoftwareTrigger	软件触发指定 DMA 的搬运
DMA_GetStatus	获得指定 DMA 的传输状态
DMA_ITConfig	使能或者失能指定的 DMA 中断
DMA_GetFlagStatus	获得指定 DMA 的标志位状态
DMA_ClearFlag	清除指定 DMA 的标志位状态
DMA_DMACmd	使能或失能 DMA 请求 DMA

DMA 固件库函数详解

DMA_DeInit

Table 91

函数名	DMA_DeInit
函数原型	void DMA_DeInit(DMA_TypeDef* DMAx)
功能描述	将外设 DMAx 寄存器重设为缺省值
输入参数	DMAx：x 可以是 0 到 3，来选择 DMA 外设
返回值	无

使用示例:

DMA_DeInit(DMA0); //将外设 DMA0 寄存器重设为缺省值

DMA_Init

Table 92

函数名	DMA_Init
函数原形	void DMA_Init(DMA_TypeDef* DMAx, DMA_InitTypeDef* DMA_InitStruct)
功能描述	根据 DMA_InitStruct 中指定的参数初始化外设 DMAx 寄存器
输入参数 1	DMAx：x 可以是 0 到 3，来选择 DMA 外设
输入参数 2	DMA_InitStruct：指向结构 DMA_InitTypeDef 的指针，包含了外设 GPIO 的配置信息 参阅 Section：DMA_InitTypeDef 查阅更多该参数允许取值范围
返回值	无

DMA_ InitTypeDef

DMA_InitTypeDef 定义于文件“sc32f1xxx_dma.h”：

```
typedef struct
{
uint16_t DMA_Priority;
uint16_t DMA_CircularMode;
uint16_t DMA_DataSize;
uint16_t DMA_TargetMode;
uint16_t DMA_SourceMode;
uint16_t DMA_Burst;
uint32_t DMA_BufferSize;
uint32_t DMA_Request;
uint32_t DMA_SrcAddress;
uint32_t DMA_DstAddress;
} DMA_InitTypeDef;
```

DMA_Priority

DMA_Priority设置通道优先级，参阅Table 93获得这个参数的所有成员。

Table 93

DMA_Priority	描述
DMA_Priority_LOW	通道优先级为低
DMA_Priority_MEDIUM	通道优先级为中
DMA_Priority_HIGH	通道优先级为高
DMA_Priority_VERY_HIGH	通道优先级为非常高

DMA_ CircularMode

DMA_ CircularMode设置DMACNT计数是否为循环模式，参阅Table 94获得这个参数的所有成员。

Table 94

DMA_CircularMode	描述
DMA_CircularMode_Disable	关闭循环模式
DMA_CircularMode_Enable	使能循环模式

DMA_ DataSize

DMA_ DataSize设置DMA 源/目标地址的传输宽度，参阅Table 95获得这个参数的所有成员。

Table 95

DMA_DataSize	描述

DMA_DataSize_Byte	传输宽度为 8 位
DMA_DataSize_HakfWord	传输宽度为 16 位
DMA_DataSize_Word	传输宽度为 32 位

DMA_ TargetMode

DMA_ TargetMode设置DMA目标地址的增减模式，参阅Table 96获得这个参数的所有成员。

Table 96

DMA_TargetMode	描述
DMA_TargetMode_FIXED	无增量(固定地址模式)
DMA_TargetMode_INC	增量模式
DMA_TargetMode_DEC	减量模式
DMA_TargetMode_INC_CIRC	递增循环模式

DMA_ SourceMode

DMA_ SourceMode设置DMA源地址的增减模式，参阅Table 97获得这个参数的所有成员。

Table 97

DMA_SourceMode	描述
DMA_SourceMode_FIXED	无增量(固定地址模式)
DMA_SourceMode_INC	增量模式
DMA_SourceMode_DEC	减量模式
DMA_SourceMode_INC_CIRC	递增循环模式

DMA_ Burst

DMA_ Burst设置单次/批量传输模式，或批量传输时Burst的大小，参阅Table 98获得这个参数的所有成员。

Table 98

DMA_Burst	描述
DMA_Burst_Disable	单次传输
DMA_Burst_1B	批量传输，Burst=1
DMA_Burst_2B	批量传输，Burst=2
DMA_Burst_4B	批量传输，Burst=4
DMA_Burst_8B	批量传输，Burst=8
DMA_Burst_16B	批量传输，Burst=16
DMA_Burst_32B	批量传输，Burst=32
DMA_Burst_64B	批量传输，Burst=64
DMA_Burst_128B	批量传输，Burst=128

DMA_ BufferSize

DMA_ BufferSize设置DMA计数器缓存寄存器的值DMACNT。

DMA_ Request

DMA_ Request设置DMA请求源，参阅Table 99获得这个参数的所有成员。

Table 99

DMA_Request	描述
DMA_Request_Null	禁用 DMA 外设请求
DMA_Request_UART0_TX	UART0_IDE->TXDMAEN
DMA_Request_UART0_RX	UART0_IDE->RXDMAEN
DMA_Request_UART1_TX	UART1_IDE->TXDMAEN
DMA_Request_UART1_RX	UART1_IDE->RXDMAEN
DMA_Request_SPI0_TX	SPI0_IDE->TXDMAEN
DMA_Request_SPI0_RX	SPI0_IDE->RXDMAEN
DMA_Request_SPI1_TX	SPI1_IDE->TXDMAEN
DMA_Request_SPI1_RX	SPI1_IDE->RXDMAEN
DMA_Request_TWI0_TX	TWI0_IDE->TXDMAEN
DMA_Request_TWI0_RX	TWI0_IDE->RXDMAEN
DMA_Request_TIM1_TI	TIM1_IDE->TIDE

DMA_Request_TIM1_CAPF	TIM1_IDE->CAPFDE
DMA_Request_TIM1_CAPR	TIM1_IDE->CAPRDE
DMA_Request_TIM2_TI	TIM2_IDE->TIDE
DMA_Request_TIM2_CAPF	TIM2_IDE->CAPFDE
DMA_Request_TIM2_CAPR	TIM2_IDE->CAPRDE
DMA_Request_TIM6_TI	TIM6_IDE->TIDE
DMA_Request_TIM6_CAPF	TIM6_IDE->CAPFDE
DMA_Request_TIM6_CAPR	TIM6_IDE->CAPRDE
DMA_Request_ADC	ADCCON->DMAEN
DMA_Request_DMA0	DMA0_CFG->CHRQ
DMA_Request_DMA1	DMA1_CFG->CHRQ
DMA_Request_DMA2	DMA2_CFG->CHRQ
DMA_Request_DMA3	DMA3_CFG->CHRQ

SC32f10xx、SC32f11xx、SC32f15xx可选(DMA_Request_Null- DMA_Request_DMA3)
SC32f12xx可选(DMA_Request_Null- DMA_Request_DMA1)

DMA_ SrcAddress

DMA_ SrcAddress设置DMA传输的源地址。

DMA_ DstAddress

DMA_ DstAddress设置DMA传输的目标地址。

使用示例:

```
/* 根据 DMA_InitStruct 中指定的参数初始化 DMA0，将Test1Data的数据搬移到Test2Data */
uint8_t  Test1Data[12] = "hello world", Test2Data[12] = {0};

DMA_InitTypeDef  DMA_InitStruct;
DMA_InitStruct.DMA_Priority = DMA_Priority_LOW;   //设置为低优先级
DMA_InitStruct.DMA_DataSize = DMA_DataSize_Byte;  //DMA传输宽度为8bit
DMA_InitStruct.DMA_CircularMode = DMA_CircularMode_Disable;;   //单次传输
DMA_InitStruct.DMA_TargetMode = DMA_TargetMode_INC; //目标地址递增循环
DMA_InitStruct.DMA_SourceMode = DMA_SourceMode_INC; //源地址递增循环
DMA_InitStruct.DMA_Burst = DMA_Burst_INC4;  //批量传输，BURSIZE大小为 4
DMA_InitStruct.DMA_Request = DMA_Request_TIM1_TI; //DMA触发源为TIM1发送数据
DMA_InitStruct.DMA_SrcAddress = (uint32_t)Test1Data;  //源地址为Test1Data数组地址
DMA_InitStruct.DMA_DstAddress = (uint32_t)Test2Data;  //目标地址为Test2Data数组地址
DMA_InitStruct.DMA_BufferSize = 12;  //传输12个数据，DMACNT=12
DMA_Init(DMA0,&DMA_InitStruct);
```

DMA_StructInit

Table 100

函数名	DMA_StructInit
函数原形	void DMA_StructInit(DMA_InitTypeDef* DMA_InitStruct)
功能描述	把 DMA_InitStruct 中的每一个参数按缺省值填入
输入参数	DMA_InitStruct：指向结构 DMA_InitTypeDef 的指针，待初始化
返回值	无

DMA_InitStruct

DMA_InitStruct的成员缺省值如Table 101所示。

Table 101

成员	缺省值
DMA_Priority	DMA_Priority_LOW
DMA_CircularMode	DMA_CircularMode_Disable
DMA_DataSize	DMA_DataSize_Byte
DMA_TargetMode	DMA_TargetMode_FIXED
DMA_SourceMode	DMA_SourceMode_FIXED

DMA_Burst	DMA_Burst_Disable
DMA_Request	DMA_Request_Null

使用示例:

```
/* 把 DMA_InitStruct 中的每一个参数按缺省值填入*/
DMA_InitTypeDef   DMA_InitStruct;
DMA_StructInit(&DMA_InitStruct);
```

DMA_Cmd

Table 102

函数名	DMA_Cmd
函数原形	void DMA_Cmd(DMA_TypeDef* DMAx, FunctionalState NewState)
功能描述	使能或者失能 DMA 外设
输入参数 1	DMAx：x 可以是 0 到 3，来选择 DMA 外设
输入参数 2	NewState: 外设 DMAx 的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* 使能DMA0 外设 */
DMA_Cmd(DMA0, ENABLE);
```

DMA_CHRQCmd

Table 103

函数名	DMA_CHRQCmd
函数原形	void DMA_CHRQCmd(DMA_TypeDef* DMAx, FunctionalState NewState)
功能描述	使能或失能 DMA 请求 DMA
输入参数 1	DMAx：x 可以是 0 到 3，来选择 DMA 外设
输入参数 2	NewState: DMAx 传输暂停的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* 暂停DMA0，请求其他DMA通道 */
DMA_CHRQCmd(DMA0, ENABLE);
```

DMA_PauseCmd

Table 104

函数名	DMA_PauseCmd
函数原形	void DMA_PauseCmd(DMA_TypeDef* DMAx, FunctionalState NewState)
功能描述	使能或失能指定 DMA 的传输暂停
输入参数 1	DMAx：x 可以是 0 到 3，来选择 DMA 外设
输入参数 2	NewState: DMAx 传输暂停的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* 暂停DMA0传输 */
DMA_PauseCmd(DMA0, ENABLE);
```

DMA_ ChannelReset

Table 105

函数名	DMA_ ChannelReset
函数原形	void DMA_ ChannelReset (DMA_TypeDef* DMAx)
功能描述	复位指定 DMA 的传输通道

输入参数	DMAx：x 可以是 0 到 3，来选择 DMA 外设
返回值	无

使用示例:

```
/* 复位DMA0传输通道 */
DMA_ ChannelReset (DMA0);
```

DMA_SetSrcAddress

Table 106

函数名	DMA_SetSrcAddress
函数原形	void DMA_SetSrcAddress(DMA_TypeDef* DMAx, uint32_t SrcAddress)
功能描述	设置指定 DMA 的源地址
输入参数 1	DMAx：x 可以是 0 到 3，来选择 DMA 外设
输入参数 2	SrcAddress：DMAx 传输的数据源地址
返回值	无

使用示例:

```
/* 设置Addre1为DMA0的源地址 */
uint32_t Addre11= 0x20000004;
DMA_SetSrcAddress(DMA0, Addre11);
```

DMA_SetDstAddress

Table 107

函数名	DMA_SetDstAddress
函数原形	void DMA_SetDstAddress(DMA_TypeDef* DMAx, uint32_t DstAddress)
功能描述	设置指定 DMA 的目标地址
输入参数 1	DMAx：x 可以是 0 到 3，来选择 DMA 外设
输入参数 2	DstAddress：DMAx传输的数据目标地址
返回值	无

使用示例:

```
/* 设置Addre1为DMA0的目标地址 */
uint32_t Addre11= 0x20000008;
DMA_SetDstAddress(DMA0, Addre11);
```

DMA_SetCurrDataCounter

Table 108

函数名	DMA_SetCurrDataCounter
函数原形	void DMA_SetCurrDataCounter(DMA_TypeDef* DMAx, uint32_t Counter)
功能描述	设置指定 DMA 的传输数量
输入参数 1	DMAx：x 可以是 0 到 3，来选择 DMA 外设
输入参数 2	DMACounter：DMA 传输数据的数量
返回值	无

使用示例:

```
/* 设置DMA0的传输数量为100 */
DMA_SetCurrDataCounter(DMA0, 100);
```

DMA_GetCurrDataCounter

Table 109

函数名	DMA_GetCurrDataCounter
函数原形	uint32_t DMA_GetCurrDataCounter(DMA_TypeDef *DMAx)
功能描述	获得指定 DMA 的传输数量
输入参数	DMAx：x 可以是 0 到 3，来选择 DMA 外设

返回值	DMA 的传输数量

使用示例:

```
/* 获得DMA0的传输数量 */
uint32_t Number;
Number = DMA_GetCurrDataCounter(DMA0);
```

DMA_SoftwareTrigger

Table 110

函数名	DMA_SoftwareTrigger
函数原形	void DMA_SoftwareTrigger(DMA_TypeDef *DMAx)
功能描述	软件触发指定 DMA 的搬运
输入参数	DMAx：x 可以是 0 到 3，来选择 DMA 外设
返回值	无

使用示例:

```
/* 软件触发一次DMA0的搬运 */
DMA_SoftwareTriggerr(DMA0);
```

DMA_GetStatus

Table 111

函数名	DMA_GetStatus
函数原形	DMA_State_TypeDef DMA_GetStatus(DMA_TypeDef* DMAx)
功能描述	获得指定 DMA 的传输状态
输入参数	DMAx：x 可以是 0 到 3，来选择 DMA 外设
返回值	DMA 的传输状态

DMA_ State_TypeDef

DMA_ State_TypeDef返回DMA状态机状态，参阅Table 112获得这个类型的所有成员。

Table 112

成员	状态
DMA_State_IDLE	空闲
DMA_State_SOURCE	写入源地址
DMA_State_BUSY	读取源地址数据，并写入目的地址
DMA_State_DESTINATION	写入目的地址数据
DMA_State_HANG	挂起等待中(有通道在忙，其他通道请求挂起)
DMA_State_PAUSE	暂停等待中(批量传输模式时 PAUSE 写 1 后)
DMA_State_BURST	burst 传输中
DMA_State_STOP	burst 传输停止(或 pause，或 dmacnt 计数到 0，或 bursize 计数到 0)

使用示例:

```
/* 获得DMA0的传输状态 */
DMA_StateTypeDef DMA_State；
DMA_State = DMA_GetStatus(DMA0);
```

DMA_ITConfig

Table 113

函数名	DMA_ITConfig
函数原形	void DMA_ITConfig(DMA_TypeDef* DMAx, uint32_t DMA_IT, FunctionalState NewState)
功能描述	使能或者失能指定的 DMA 中断
输入参数 1	DMAx：x 可以是 0 到 3，来选择 DMA 外设
输入参数 2	DMA_IT：待使能或者失能的 DMA 中断源 参阅 Section：DMA_IT 查阅更多该参数允许取值范围
输入参数 3	NewState：DMAx 中断的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

DMA_ IT

DMA_IT 可以用来使能或者失能DMA中断。可以使用Table 114中的一个参数，或者他们的组合。

Table 114

DMA _IT	描述
DMA_IT_INTEN	DMA 的 CPU 中断请求
DMA_IT_TCIE	DMA 传输完成中断
DMA_IT_HTIE	DMA 传输一半中断
DMA_IT_TEIE	DMA 传输错误中断

使用示例:

```
/* 使能DMA0传输完成中断 */
DMA_ITConfig (DMA0, DMA_IT_INTEN | DMA_IT_TCIE，ENABLE);
```

DMA_GetFlagStatus

Table 115

函数名	DMA_GetFlagStatus
函数原形	FlagStatus DMA_GetFlagStatus(DMA_TypeDef* DMAx, DMA_Flag_TypeDef DMA_FLAG)
功能描述	检查指定的 DMA 标志位设置与否
输入参数 1	DMAx：x 可以是 0 到 3，来选择 DMA 外设
输入参数 2	DMA_FLAG：待检查的 DMA 标志位 参阅 Section：DMA_FLAG 查阅更多该参数允许取值范围
返回值	DMA_FLAG 的新状态

DMA_ FLAG

Table 116给出了DMA_FLAG的值。

Table 116

DMA _IT	描述
DMA_FLAG_GIF	DMA 全局中断标志位
DMA_FLAG_TCIF	DMA 传输完成中断标志位
DMA_FLAG_HTIF	DMA 传输一半中断标志位
DMA_FLAG_TEIF	DMA 传输错误中断标志位

使用示例:

```
/* 获得DMA传输完成标志位状态 */
DMA_GetFlagStatus(DMA0, DMA_FLAG_TCIF);
```

DMA_ClearFlag

Table 117

函数名	DMA_ClearFlag
函数原形	void DMA_ClearFlag(DMA_TypeDef* DMAx, uint32_t DMA_FLAG)
功能描述	清除 DMAx 的待处理标志位
输入参数 1	DMAx：x 可以是 0 到 3，来选择 DMA 外设
输入参数 2	DMA_FLAG：待清除的 DMA 标志位 参阅 Section：DMA_FLAG 查阅更多该参数允许取值范围
返回值	无

使用示例:

```
/* 清除DMA0传输完成标志位 */
DMA_ClearFlag(DMA0, DMA_FLAG_TCIF);
```

DMA_ DMACmd

Table 118

函数名	DMA_DMACmd
函数原形	void DMA_DMACmd(DMA_TypeDef* DMAx, uint32_t DMA_DMARequest, FunctionalState NewState)

功能描述	使能或失能 DMA 请求 DMA
输入参数 1	DMAx：x 可以是 0 到 3，来选择 DMA 外设
输入参数 2	DMA_DMARequest：请求的其他 DMA 通道
输入参数 3	NewState: DMA 请求 DMA 的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* 当前通道请求其他通道*/
DMA_DMACmd(DMA0, DMA_DMAReq_CHRQ,ENABLE);
```

综合使用示例:（使用 DMA0 和 DMA1 实现 UART1 自发自收）

```
volatile uint8_t  TestData[12];
volatile uint8_t UART1_FLAG = 0;
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA, ENABLE); //开启DMA时钟
RCC_APB0Cmd(ENABLE); //开启UART1时钟
RCC_APB0PeriphClockCmd(RCC_APB0Periph_UART1,ENABLE);
    /* UART1初始化 */
    /* 设置为10位模式,使能发送/接收开关 */
    UART_DeInit(UART1);
    {
      UART_InitTypeDef UART_InitStruct;

      UART_InitStruct.UART_BaudRate = 115200;
      UART_InitStruct.UART_ClockFrequency = 16500000;
      UART_InitStruct.UART_Mode = UART_Mode_10B;
      UART_Init(UART1, &UART_InitStruct);
    }
    UART_TXCmd(UART1,ENABLE);
    UART_RXCmd(UART1,ENABLE);
    UART_ITConfig(UART1, UART_IT_EN | UART_IT_RX,ENABLE); //使能UART1接收中断
    NVIC_EnableIRQ(UART1_3_IRQn);

    /* UART1触发DMA 使能：UART1接收触发DMA0，UART1发送触发DMA1 */
    UART_DMACmd(UART1,UART_DMAReq_TX | UART_DMAReq_RX,ENABLE);
    DMA_DeInit(DMA0);  //DMA0复位至缺省值
    DMA_DeInit(DMA1);  //DMA1复位至缺省值
    {
DMA_InitTypeDef DMA_InitStruct;
        /* DMA0初始化 */
        DMA_InitStruct.DMA_Priority = DMA_Priority_LOW;  //通道优先级为低
DMA_InitStruct.DMA_DataSize = DMA_DataSize_Byte; //8位数据传输
        DMA_InitStruct.DMA_CircularMode = DMA_CircularMode_Enable; //使能计数器缓存寄存器为循环模式
        DMA_InitStruct.DMA_Request = DMA_Request_UART1_RX; //DMA0请求源为UART1接收数据
        DMA_InitStruct.DMA_Burst = DMA_Burst_INC1; //批量传输
        DMA_InitStruct.DMA_SrcAddress = (uint32_t)(&(UART1->UART_DATA)); //源地址为UART1数据寄存器
        DMA_InitStruct.DMA_SourceMode = DMA_SourceMode_FIXED; //地址固定不变
        DMA_InitStruct.DMA_DstAddress = (uint32_t)TestData; //目标地址为TestData数组地址
        DMA_InitStruct.DMA_TargetMode = DMA_TargetMode_INC_CIRC; //配置为递增循环模式
        DMA_InitStruct.DMA_BufferSize = 12; //写入DMA计数器缓存寄存器的值
        DMA_Init(DMA0,&DMA_InitStruct);
        /* DMA1初始化 */
        DMA_InitStruct.DMA_Request = DMA_Request_UART1_TX; //DMA1请求源为UART1发送数据
        DMA_InitStruct.DMA_DstAddress = (uint32_t)(&(UART1->UART_DATA)); //目标地址为UART1数据寄存器
        DMA_InitStruct.DMA_TargetMode = DMA_TargetMode_FIXED;//地址固定不变
```

```
        DMA_InitStruct.DMA_SrcAddress = (uint32_t)TestData; //源地址为TestData数组地址
        DMA_InitStruct.DMA_SourceMode = DMA_SourceMode_INC_CIRC; //配置为递增循环模式
DMA_Init(DMA1,&DMA_InitStruct);
    }
DMA_Cmd(DMA0,ENABLE); //使能DMA0
DMA_Cmd(DMA1,ENABLE); //使能DMA1
    while(1)
    {
if(UART1_FLAG) //如果UART1接收完所有数据
        {
              UART1_FLAG = 0;
              DMA_SoftwareTrigger(DMA1);   //使能DMA1软件请求
             {
uint32_t delay_time = 400000;
                while(delay_time--);
}
              if(DMA_GetFlagStatus(DMA1,DMA_FLAG_TCIF)) //如果DMA1发送完成
              {
                 DMA_ClearFlag(DMA1,DMA_FLAG_TCIF); //清除发送完成标志
                 DMA_SetCurrDataCounter(DMA1,12); //重新写入DMA1计数器缓存寄存器的值
                 UART_DMACmd(UART1,UART_DMAReq_TX,DISABLE); //重新使能UART1发送触发DMA
                 UART_DMACmd(UART1,UART_DMAReq_TX,ENABLE);
              }
}
void UART1_3_IRQHandler()      //UART1中断服务函数
{
  UART_ClearFlag(UART1,UART_Flag_RX);
    if(DMA_GetFlagStatus(DMA0,DMA_FLAG_TCIF)) //如果DMA0接收完成
    {
        UART1_FLAG = 1;
DMA_ClearFlag(DMA0,DMA_FLAG_TCIF); //清除接受完成标志
    }
}
```

8 GPIO 固件库函数

赛元 SC32F1xxx系列提供了最多46个可控制的双向GPIO端口，输入输出控制寄存器用来控制各端口的输入输出状态。此46个IO口同其他功能复用。

GPIO 寄存器结构

GPIO 寄存器结构，GPIO_TypeDef，在文件“sc32f1xxx.h”中定义如下：

```
typedef struct
{
  __IO uint32_t PIN;
  __IO uint32_t RESERVED0[7];
  __IO uint32_t PXCON;
  __IO uint32_t RESERVED1[7];
  __IO uint32_t PXPH;
  __IO uint32_t RESERVED2[7];
  __IO uint32_t PXLEV;
} GPIO_TypeDef;
```

GPIO 寄存器表格

Table 119

寄存器	描述
PIN	PX端口数据寄存器
PXCON	PX口输入/输出控制寄存器
PXPH	PX口上拉电阻控制寄存器
PXLEV	IOH设置寄存器 IO驱动等级

GPIO 固件库函数列表

Table 120

函数名	描述
GPIO_DeInit	GPIO 相关寄存器复位至缺省值
GPIO_Init	根据 GPIO_InitStruct 中指定的参数初始化外设 GPIOx 寄存器
GPIO_SetDriveLevel	IO 驱动等级设置
GPIO_ReadData	读取指定的 GPIO 端口数据
GPIO_ReadDataBit	读取指定端口管脚的状态
GPIO_SetBits	设置指定的数据端口位
GPIO_ResetBits	清除指定的数据端口位
GPIO_Write	向指定 GPIO 数据端口写入数据
GPIO_WriteBit	设置或者清除指定的数据端口位
GPIO_TogglePins	翻转指定的数据端口位
PA_BIT	设置 PA 端口管脚
PB_BIT	设置 PB 端口管脚
PC_BIT	设置 PC 端口管脚
PD_BIT	设置 PD 端口管脚
PA_OT	翻转 PA 端口管脚
PB_OT	翻转 PB 端口管脚
PC_OT	翻转 PC 端口管脚
PD_OT	翻转 PD 端口管脚

GPIO 固件库函数详解

GPIO_DeInit

Table 121

函数名	GPIO_DeInit
函数原型	void GPIO_DeInit(GPIO_TypeDef* GPIOx)
功能描述	GPIO 相关寄存器复位至缺省值
输入参数	GPIOx：x 可以是 A、B、C，D 来选择 GPIO 外设(具体选择需以规格书为准)
返回值	无

使用示例:

```
/*GPIOA寄存器复位*/
GPIO_DeInit(GPIOA);
```

GPIO_Init

Table 122

函数名	GPIO_Init
函数原型	void GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct)
功能描述	根据 GPIO_InitStruct 中指定的参数初始化外设 GPIOx 寄存器
输入参数 1	GPIOx：x 可以是 A、B、C，D 来选择 GPIO 外设(具体选择需以规格书为准)
输入参数 2	GPIO_InitStruct：指向结构 GPIO_InitTypeDef 的指针，包含了外设 GPIO 的配置信息
返回值	无

GPIO_ InitTypeDef

GPIO_InitTypeDef 定义于文件“sc32f1xxx_gpio.h”：

```
typedef struct
{
   uint16_t GPIO_Pin;
   uint16_t GPIO_Mode;
   uint16_t GPIO_DriveLevel;
} GPIO_InitTypeDef;
```

GPIO_Pin

GPIO_Pin设置GPIO的Pin脚，参阅Table 123获得这个参数的所有成员，可以使用其中的一个参数，或者他们的组合。

Table 123

GPIO_Pin	描述
GPIO_Pin_0	选择 Px0 管脚(x 为 A/B/C)
GPIO_Pin_1	选择 Px1 管脚
GPIO_Pin_2	选择 Px2 管脚
GPIO_Pin_3	选择 Px3 管脚
GPIO_Pin_4	选择 Px4 管脚
GPIO_Pin_5	选择 Px5 管脚
GPIO_Pin_6	选择 Px6 管脚
GPIO_Pin_7	选择 Px7 管脚
GPIO_Pin_8	选择 Px8 管脚
GPIO_Pin_9	选择 Px9 管脚
GPIO_Pin_10	选择 Px10 管脚
GPIO_Pin_11	选择 Px11 管脚
GPIO_Pin_12	选择 Px12 管脚
GPIO_Pin_13	选择 Px13 管脚
GPIO_Pin_14	选择 Px14 管脚
GPIO_Pin_15	选择 Px15 管脚
GPIO_PIN_LNIB	选择低八位管脚 Px0-Px7
GPIO_PIN_HNIB	选择高八位管脚 Px8-Px15
GPIO_PIN_All	选择所有管脚 Px0-Px15

GPIO_Mode

GPIO_Mode设置GPIO的输入/输出模式，参阅Table 124获得这个参数的所有成员。

Table 124

GPIO_Mode	描述
GPIO_Mode_IN_PU	Pxy 为上拉输入模式
GPIO_Mode_IN_HI	Pxy 为高阻输入模式
GPIO_Mode_OUT_PP	Pxy 为强推挽输出模式

GPIO_DriveLevel

GPIO_DriveLevel设置IO驱动等级，参阅Table 125获得这个参数的所有成员。

Table 125

GPIO_DriveLevel	描述
GPIO_DriveLevel_0	设置 P1 高四位 IOH 等级 0（最大）
GPIO_DriveLevel_1	设置 P1 高四位 IOH 等级 1
GPIO_DriveLevel_2	设置 P1 高四位 IOH 等级 2
GPIO_DriveLevel_3	设置 P1 高四位 IOH 等级 3（最小）

使用示例:

```
/* 根据 GPIO_InitStruct 中指定的参数，将PA0初始化为上拉输入模式 */
GPIO_InitTypeDef GPIO_InitStruct;
GPIO_InitStruct.GPIO_Pin = GPIO_Pin_0;
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN_PU;
GPIO_InitStruct.GPIO_DriveLevel = GPIO_DriveLevel_3;
GPIO_Init(GPIOA,&GPIO_InitStruct);
```

GPIO_SetDriveLevel

Table 126

函数名	GPIO_SetDriveLevel
函数原型	void GPIO_SetDriveLevel(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_DriveLevel_TypeDef GPIO_DriveLevel)
功能描述	IO 驱动等级设置
输入参数 1	GPIOx：x 可以是 A、B、C，D 来选择 GPIO 外设(具体选择需以规格书为准)
输入参数 2	GPIO_Pin：待设置的端口位
输入参数 3	GPIO_DriveLevel：IO 驱动等级选择
返回值	无

使用示例:

```
/*将PA0的驱动等级设为2 */
GPIO_SetDriveLevel(GPIOA, GPIO_Pin_0, GPIO_DriveLevel_2);
```

GPIO_ReadData

Table 127

函数名	GPIO_ReadData
函数原型	uint16_t GPIO_ReadData(GPIO_TypeDef* GPIOx)
功能描述	读取指定的 GPIO 端口数据
输入参数	GPIOx：x 可以是 A、B、C，D 来选择 GPIO 外设(具体选择需以规格书为准)
返回值	GPIO 端口数据

使用示例:

```
/*读取GPIOA的端口数据*/
uint16_t  ReadPortData;
ReadPortData = GPIO_ReadData(GPIOA);
```

GPIO_ReadDataBit

Table 128

函数名	GPIO_ReadDataBit
函数原型	BitAction GPIO_ReadDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
功能描述	读取指定端口管脚的状态
输入参数 1	GPIOx：x 可以是 A、B、C，D 来选择 GPIO 外设(具体选择需以规格书为准)
输入参数 2	GPIO_Pin：待读取的端口位，取值参考Table 123
返回值	输出端口管脚的状态

BitAction

BitAction返回GPIO端口的状态，参阅Table 129获得这个类型的所有成员。

Table 129

成员	状态
Bit_RESET	返回值 0
Bit_SET	返回值 1

使用示例:

```
/*读取PA1的状态*/
BitAction  ReadPin;
ReadPin = GPIO_ReadDataBit(GPIOA, GPIO_Pin_1);
```

GPIO_SetBits

Table 130

函数名	GPIO_SetBits
函数原型	void GPIO_SetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
功能描述	设置指定的数据端口位
输入参数 1	GPIOx：x 可以是 A、B、C，D 来选择 GPIO 外设(具体选择需以规格书为准)
输入参数 2	GPIO_Pin：待设置的端口位，取值参考 Table 123
返回值	无

使用示例:

```
/*设置PA11和PA12的值*/
GPIO_SetBits(GPIOA, GPIO_Pin_11 | GPIO_Pin_12);
```

GPIO_ResetBits

Table 131

函数名	GPIO_ResetBits
函数原型	void GPIO_ResetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
功能描述	清除指定的数据端口位
输入参数 1	GPIOx：x 可以是 A、B、C，D 来选择 GPIO 外设(具体选择需以规格书为准)
输入参数 2	GPIO_Pin：待设置的端口位，取值参考 Table 123
返回值	无

使用示例:

```
/*清除PA11和PA12的值*/
GPIO_ResetBits(GPIOA, GPIO_Pin_11 | GPIO_Pin_12);
```

GPIO_Write

Table 132

函数名	GPIO_Write
函数原型	void GPIO_Write(GPIO_TypeDef* GPIOx, uint16_t PortVal)
功能描述	向指定GPIO 数据端口写入数据
输入参数 1	GPIOx：x 可以是 A、B、C，D 来选择 GPIO 外设(具体选择需以规格书为准)
输入参数 2	PortVal: 待写入端口数据寄存器的值
返回值	无

使用示例:

```
/*向GPIOB写入数据*/
GPIO_Write(GPIOB, 0x1101);
```

GPIO_WriteBit

Table 133

函数名	GPIO_WriteBit
函数原型	void GPIO_WriteBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, BitAction BitVal)
功能描述	设置或者清除指定的数据端口位
输入参数 1	GPIOx：x 可以是 A、B、C，D 来选择 GPIO 外设(具体选择需以规格书为准)
输入参数 2	GPIO_Pin：待设置的端口位，取值参考 Table 123
输入参数 3	BitVal：该参数指定了待写入的值，取值参考 Table 129
返回值	无

使用示例:

```
/*设置PB15的值*/
GPIO_WriteBit(GPIOB, GPIO_Pin_15, Bit_SET);
```

GPIO_TogglePins

Table 134

函数名	GPIO_TogglePins
函数原型	void GPIO_TogglePins(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
功能描述	翻转指定的数据端口位
输入参数 1	GPIOx：x 可以是 A、B、C，D 来选择 GPIO 外设(具体选择需以规格书为准)
输入参数 2	GPIO_Pin：待设置的端口位，该参数可以取 GPIO_Pin_x(x 可以是 0-15)的任意组合
返回值	无

使用示例:

```
/*翻转PB12*/
GPIO_TogglePins (GPIOB, GPIO_Pin_12);
```

PA_BIT

Table 135

函数名	PA_BIT
函数原型	PA_BIT(n)
功能描述	设置 PA 端口管脚
输入参数 1	n：n 可以是 0 至 15，用 n 来选择 GPIOA 的端口
返回值	无

使用示例:

```
/*设置PA12为高电平*/
PA_BIT(12)=1;
```

PB_BIT

Table 136

函数名	PB_BIT
函数原型	PB_BIT(n)
功能描述	设置 PB 端口管脚
输入参数 1	n：n 可以是 0 至 15，用 n 来选择 GPIOB 的端口
返回值	无

使用示例:

```
/*设置PB12为高电平*/
PB_BIT(12)=1;
```

PC_BIT

Table 137

函数名	PC_BIT
函数原型	PC_BIT(n)
功能描述	设置 PC 端口管脚
输入参数 1	n：n 可以是 0 至 15，用 n 来选择 GPIOC 的端口
返回值	无

使用示例:

```
/*设置PC12为低电平*/
PC_BIT(12)=0;
```

PD_BIT

Table 138

函数名	PD_BIT
函数原型	PD_BIT(n)
功能描述	设置 PD 端口管脚
输入参数 1	n：n 可以是 0 至 15，用 n 来选择 GPIOD 的端口
返回值	无

使用示例:

```
/*设置PD12为低电平*/
PD_BIT(12)=0;
```

PA_OT

Table 139

函数名	PA_OT
函数原型	PA_OT(n)
功能描述	翻转 PA 端口管脚
输入参数 1	n：n 可以是 0 至 15，用 n 来选择 GPIOA 的端口
返回值	无

使用示例:

```
/*翻转PA12引脚*/
PA_OT(12);
```

PB_OT

Table 140

函数名	PB_OT
函数原型	PB_OT(n)
功能描述	翻转 PB 端口管脚
输入参数 1	n：n 可以是 0 至 15，用 n 来选择 GPIOB 的端口
返回值	无

使用示例:

```
/*翻转PB12引脚*/
PB_OT(12);
```

PC_OT

Table 141

函数名	PC_OT
函数原型	PC_OT(n)
功能描述	翻转 PC 端口管脚
输入参数 1	n：n 可以是 0 至 15，用 n 来选择 GPIOC 的端口
返回值	无

使用示例:

```
/*翻转PC12引脚*/
```

PC_OT(12);

PD_OT

Table 142

函数名	PD_OT
函数原型	PD_OT(n)
功能描述	翻转 PD 端口管脚
输入参数 1	n：n 可以是 0 至 15，用 n 来选择 GPIOD 的端口
返回值	无

使用示例:

```
/*翻转PD12引脚*/
PD_OT(12);
```

综合使用示例:（初始化 PA0 为高电平后翻转 PA0）

```
unsigned long cnt_flag = 50000;
GPIO_DeInit(GPIOA);
{
  GPIO_InitTypeDef GPIO_InitStruct;
  GPIO_InitStruct.GPIO_Pin = GPIO_Pin_0;
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT_PP;
  GPIO_Init(GPIOA,&GPIO_InitStruct);
}
while(1)
{
  cnt_flag = 50000;
  while(cnt_flag--);
  GPIO_TogglePins(GPIOA,GPIO_Pin_0);
}
```

9 LCD 固件库函数

赛元SC32f10xx，SC32f11xx，SC32f12xx系列具有LCD显示驱动电路，可方便用户实行LCD的显示驱动。其主要特点如下：

1. LCD和LED显示驱动二选一；
2. LCD和LED显示驱动共用相关IO口和寄存器。

LCD 寄存器结构

LCD 寄存器结构，LCD_LED_TypeDef，在文件“sc32f1xxx.h”中定义如下：

```
typedef struct
{
__IO uint32_t DDR_CON;
__IO uint32_t DDR_CFG;
__IO uint32_t SEG_EN;
__IO uint32_t RESERVED0;
__IO uint32_t COM_EN;
__IO uint32_t RESERVED1[39];
__IO uint32_t SEGRn[28];
} LCD_LED_TypeDef;
```

LED 寄存器表格

Table 143

寄存器	描述
DDR_CON	显示驱动控制寄存器0
DDR_CFG	显示驱动配置寄存器
SEG_EN	SEG口使能寄存器
COM_EN	COM口使能寄存器
SEGRn(n=0~35)	SEGn显示RAM(n=0~34)

LCD 固件库函数列

Table 144

函数名	描述
LCD_DeInit	LCD 相关寄存器复位至缺省值
LCD_Init	根据 LCD_InitStruct 中指定的参数初始化外设 LCD 寄存器
LCD_StructInit	把 LCD_StructInit 中的每一个参数按缺省值填入
LCD_Cmd	使能或失能 LCD 功能
LCD_Write	向 SEG RAM 写入数据
LCD_CustomModeScan	切换 COM 口输出

LCD 固件库函数详解

LCD_DeInit

Table 145

函数名	LCD_DeInit
函数原型	void LCD_DeInit(void)
功能描述	LCD 相关寄存器复位至缺省值
输入参数	无
返回值	无

使用示例:

```
/* LCD相关寄存器复位至缺省值 */
LCD_DeInit();
```

LCD_Init

Table 146

函数名	LCD_Init
函数原型	void LCD_Init(LCD_InitTypeDef* LCD_InitStruct)
功能描述	根据 LCD_InitStruct 中指定的参数初始化外设 LCD 寄存器
输入参数	LCD_InitStruct：指向结构 LCD_InitTypeDef 的指针，包含了 LCD 外设的配置信息
返回值	无

LCD_InitTypeDef

LCD_InitTypeDef 定义于文件“sc32f1xxx_lcd.h”：

```
typedef struct
{
uint16_t LCD_FrameFre;
uint16_t LCD_Duty;
uint16_t LCD_VOIRSIF;
uint16_t LCD_Bias;
uint16_t LCD_ResSel;
uint16_t LCD_Voltage;
uint32_t LCD_ComPin;
uint32_t LCD_SegPin;
} LCD_InitTypeDef;
```

LCD_FrameFre

LCD_FrameFre设置LCD频帧，参阅Table 147获得这个参数的所有成员。

Table 147

LCD_FrameFre	描述
LCD_FrameFre_A32Hz	A 波形频帧 32Hz
LCD_FrameFre_A64Hz	A 波形频帧 64Hz
LCD_FrameFre_A128Hz	A 波形频帧 128Hz
LCD_FrameFre_B64Hz	B 波形频帧 64Hz
LCD_FrameFre_B128Hz	B 波形频帧 128Hz
LCD_FrameFre_B256Hz	B 波形频帧 256Hz
LCD_FrameFre_ACustom	A 波形开启自定义帧频模式
LCD_FrameFre_BCustom	B 波形开启自定义帧频模式

LCD_ Duty

LCD_ Duty设置LCD占空比，参阅Table 148获得这个参数的所有成员。

Table 148

LCD_ Duty	描述
LCD_Duty_1_8	1/8 占空比，S4~S34 为 segment，C0~C7 为 common
LCD_Duty_1_6	1/6 占空比，S2~S34 为 segment，C2~C7 为 common
LCD_Duty_1_5	1/5 占空比，S1~S34 为 segment，C3~C7 为 common
LCD_Duty_1_4_SEG0_34COM4_7	1/4 占空比，S0~S34 为 segment，C4~C7 为 common
LCD_Duty_1_4_SEG4_34COM0_3	1/4 占空比，S4~S34 为 segment，C0~C3 为 common

LCD_ VOIRSIF

LCD_ VOIRSIF选择是否打开快速充电功能，参阅Table 149获得这个参数的所有成员。

Table 149

LCD_ VOIRSIF	描述
LCD_VOIRSIF_Disable	关闭快速充电
LCD_VOIRSIF_Enable	打开快速充电

LCD_ Bias

LCD_ Bias设置LCD偏置电压，参阅Table 150获得这个参数的所有成员。

Table 150

LCD_ Bias	描述
LCD_Bias_1_4	1/4 偏置电压
LCD_Bias_1_3	1/3 偏置电压

LCD_ ResSel

LCD_ ResSel设置LCD电压输出口分压电阻，参阅Table 151获得这个参数的所有成员。

Table 151

LCD_ ResSel	描述
LCD_ResSel_33K	设定内部分压电阻总电阻值为 33k
LCD_ResSel_100K	设定内部分压电阻总电阻值为 100k
LCD_ResSel_300K	设定内部分压电阻总电阻值为 300k
LCD_ResSel_800K	设定内部分压电阻总电阻值为 800k

LCD_ Voltage

LCD_Voltage设置相应值来进行LCD电压调节，公式为VLCD = VDD*(17+VLCD[3:0]/32)。

LCD_ ComPin

LCD_ComPin使能COMx口显示驱动输出(x= 0~7)。

LCD_ SegPin

LCD_SegPin使能SEGx口显示驱动输出(x= 0~27)。

使用示例:

```
/* 根据 LCD_InitStruct 中指定的参数初始化 LCD */
LCD_InitTypeDef LCD_InitStruct;
LCD_InitStruct.LCD_Bias = LCD_Bias_1_4;    //1/4偏置电压
LCD_InitStruct.LCD_Duty = LCD_Duty_1_4_SEG0_34COM4_7;    //1/4占空比
LCD_InitStruct.LCD_FrameFre = LCD_FrameFre_A64Hz;  //A波形频帧64Hz
LCD_InitStruct.LCD_ResSel = LCD_ResSel_100K;    //内部分压电阻总电阻值为100k
LCD_InitStruct.LCD_ComPin = 0xF0;     //使能COM4-7
LCD_InitStruct.LCD_SegPin = 0xFF;     //使能SEG0-7
LCD_InitStruct.LCD_VOIRSIF = LCD_VOIRSIF_Disable;    //关闭快充电
LCD_InitStruct.LCD_Voltage = 7;       //VLCD = VDD*0.75
LCD_Init(&LCD_InitStruct);
```

LCD_StructInit

Table 152

函数名	LCD_StructInit
函数原型	void LCD_StructInit(LCD_InitTypeDef* LCD_InitStruct)
功能描述	把 LCD_StructInit 中的每一个参数按缺省值填入
输入参数	LCD_InitStruct: 指向结构 LCD_InitTypeDef 的指针，包含了 LCD 外设的配置信息
返回值	无

使用示例:

```
/* 把LCD_StructInit中的每一个参数按缺省值填入 */
LCD_InitTypeDef LCD_InitStruct;
LCD_ StructInit (&LCD_InitStruct);
```

LCD_Cmd

Table 153

函数名	LCD_Cmd
函数原型	void LCD_Cmd(FunctionalState NewState)
功能描述	使能或失能 LCD 功能

输入参数	NewState: 外设LCD的新状态，这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/*使能LCD功能*/
LCD_Cmd(ENABLE);
```

LCD_Write

Table 154

函数名	LCD_Write
函数原型	void LCD_Write(LCD_RAMRegister_Typedef LCD_RAMRegister, uint8_t LCD_Data)
功能描述	写入 LCD 显示内容
输入参数 1	LCD_RAMRegister：LCD RAM 寄存器选择
输入参数 2	LCD_LED_Data：LCD 输出内容
返回值	无

LCD_ RAMRegister

LCD_ RAMRegister选择LCD RAM寄存器，参阅Table 155获得这个参数的所有成员。

Table 155

LCD_ RAMRegister	描述
LCD_RAMRegister_0	选择 RAM 寄存器 0
LCD_RAMRegister_1	选择 RAM 寄存器 1
LCD_RAMRegiste _2	选择 RAM 寄存器 2
LCD_RAMRegister_3	选择 RAM 寄存器 3
LCD_RAMRegister_4	选择 RAM 寄存器 4
LCD_RAMRegister_5	选择 RAM 寄存器 5
LCD_RAMRegister_6	选择 RAM 寄存器 6
LCD_RAMRegister_7	选择 RAM 寄存器 7
LCD_RAMRegister_8	选择 RAM 寄存器 8
LCD_RAMRegister_9	选择 RAM 寄存器 9
LCD_RAMRegister_10	选择 RAM 寄存器 10
LCD_RAMRegister_11	选择 RAM 寄存器 11
LCD_RAMRegister_12	选择 RAM 寄存器 12
LCD_RAMRegister_13	选择 RAM 寄存器 13
LCD_RAMRegister_14	选择 RAM 寄存器 14
LCD_RAMRegister_15	选择 RAM 寄存器 15
LCD_RAMRegister_16	选择 RAM 寄存器 16
LCD_RAMRegister_17	选择 RAM 寄存器 17
LCD_RAMRegister_18	选择 RAM 寄存器 18
LCD_RAMRegister_19	选择 RAM 寄存器 19
LCD_RAMRegister_20	选择 RAM 寄存器 20
LCD_RAMRegister_21	选择 RAM 寄存器 21
LCD_RAMRegister_22	选择 RAM 寄存器 22
LCD_RAMRegister_23	选择 RAM 寄存器 23
LCD_RAMRegister_24	选择 RAM 寄存器 24
LCD_RAMRegister_25	选择 RAM 寄存器 25
LCD_RAMRegister_26	选择 RAM 寄存器 26
LCD_RAMRegister_27	选择 RAM 寄存器 27
LCD_RAMRegister_28	选择 RAM 寄存器 28
LCD_RAMRegister_29	选择 RAM 寄存器 29
LCD_RAMRegister_30	选择 RAM 寄存器 30

LCD_RAMRegister_31	选择 RAM 寄存器 31
LCD_RAMRegister_32	选择 RAM 寄存器 32
LCD_RAMRegister_33	选择 RAM 寄存器 33
LCD_RAMRegister_34	选择 RAM 寄存器 34

SC32f10xx、SC32f11xx可选(LCD_RAMRegister_0- LCD_RAMRegister_27)
SC32f12xx可选(LCD_RAMRegister_0- LCD_RAMRegister_34)

使用示例:

```
/*对RAM寄存器0写入0x20*/
LCD_Write (LCD_RAMRegister_0,0x20);
```

LCD_CustomModeScan

Table 156

函数名	LCD_CustomModeScan
函数原型	void LCD_CustomModeScan(void)
功能描述	切换 COM 口输出
输入参数	无
返回值	无

使用示例:

```
/*切换一次COM口输出 */
void LCD_CustomModeScan();
```

综合使用示例（输出 LCD 波形）

```
RCC_APB2Cmd(ENABLE); //开启时钟
RCC_APB2PeriphClockCmd(RCC_APB2Periph_LCD_LED, ENABLE);

LCD_DeInit(); //将LCD相关寄存器复位至缺省值
{
    LCD_InitTypeDef LCD_InitStruct;
    LCD_InitStruct.LCD_Bias = LCD_Bias_1_4; //1/4偏置电压
    LCD_InitStruct.LCD_Duty = LCD_Duty_1_4_SEG0_34COM4_7; //1/4占空比
    LCD_InitStruct.LCD_FrameFre = LCD_FrameFre_A64Hz; //A波形频帧64Hz
    LCD_InitStruct.LCD_ResSel = LCD_ResSel_100K; //内部分压电阻总电阻值为100k
    LCD_InitStruct.LCD_ComPin = 0xF0;   //使能COM4-7
    LCD_InitStruct.LCD_SegPin = 0xFFFF;   //使能SEG0-15
    LCD_InitStruct.LCD_VOIRSIF = LCD_VOIRSIF_Disable; //关闭快充电
    LCD_InitStruct.LCD_Voltage = 7;   //VLCD = VDD*0.75
    LCD_Init(&LCD_InitStruct);  //初始化LCD
}
    LCD_Cmd(ENABLE); //使能LCD
    LCD_Write(LCD_RAMRegister_0,0xAA); //对RAM寄存器0写入0xAA
```

10 LED 固件库函数

赛元SC32f10xx，SC32f11xx，SC32f12xx系列具有LED显示驱动电路，可方便用户实行LED的显示驱动。其主要特点如下：

1. LCD和LED显示驱动二选一；
2. LCD和LED显示驱动共用相关IO口和寄存器。

LED 寄存器结构

LED 寄存器结构，LCD_LED_TypeDef，在文件“sc32f1xxx.h”中定义如下：

```
typedef struct
{
__IO uint32_t DDR_CON;
__IO uint32_t DDR_CFG;
__IO uint32_t SEG_EN;
__IO uint32_t RESERVED0;
__IO uint32_t COM_EN;
__IO uint32_t RESERVED1[39];
__IO uint32_t SEGRn[28];
} LCD_LED_TypeDef;
```

LED **寄存器表格**

Table 157

寄存器	描述
DDR_CON	显示驱动控制寄存器0
DDR_CFG	显示驱动配置寄存器
SEG_EN	SEG口使能寄存器
COM_EN	COM口使能寄存器
SEGRn(n=0~34)	SEGn显示RAM(n=0~34)

LED 固件库函数列

Table 158

函数名	描述
LED_DeInit	LED 相关寄存器复位至缺省值
LED_Init	根据 LED_InitStruct 中指定的参数初始化外设 LED 寄存器
LED_StructInit	把 LED_StructInit 中的每一个参数按缺省值填入
LED_Cmd	使能或失能 LED 功能
LED_Write	向 SEG RAM 写入数据
LED_CustomModeScan	切换 COM 口输出

LED 固件库函数详解

LED_DeInit

Table 159

函数名	LED_DeInit
函数原型	void LED_DeInit(void)
功能描述	LED 相关寄存器复位至缺省值
输入参数	无
返回值	无

使用示例:

```
/* LED相关寄存器复位至缺省值 */
LED_DeInit();
```

LED_Init

Table 160

函数名	LED_Init
函数原型	void LED_Init(LED_InitTypeDef* LED_InitStruct)
功能描述	根据 LED_InitStruct 中指定的参数初始化外设 LED 寄存器
输入参数	LED_InitStruct：指向结构 LED_InitTypeDef 的指针，包含了 LED 外设的配置信息
返回值	无

LED_InitTypeDef

LED_InitTypeDef 定义于文件“sc32f1xxx_led.h”：

```
typedef struct
{
uint16_t LED_FrameFre;
uint16_t LED_Duty;
uint32_t LED_ComPin;
uint32_t LED_SegPin;
} LED_InitTypeDef;
```

LED_FrameFre

LED_FrameFre设置LED频帧，参阅Table 161获得这个参数的所有成员。

Table 161

LED_FrameFre	描述
LED_FrameFre_A32Hz	A 波形频帧 32Hz
LED_FrameFre_A64Hz	A 波形频帧 64Hz
LED_FrameFre_A128Hz	A 波形频帧 128Hz
LED_FrameFre_B64Hz	B 波形频帧 64Hz
LED_FrameFre_B128Hz	B 波形频帧 128Hz
LED_FrameFre_B256Hz	B 波形频帧 256Hz
LED_FrameFre_ACustom	A 波形开启自定义帧频模式
LED_FrameFre_BCustom	B 波形开启自定义帧频模式

LED_ Duty

LED_ Duty设置LED占空比，参阅Table 162获得这个参数的所有成员。

Table 162

LED_ Duty	描述
LED_Duty_1_8	1/8 占空比，S4~S34 为 segment，C0~C7 为 common
LED_Duty_1_6	1/6 占空比，S2~S34 为 segment，C2~C7 为 common
LED_Duty_1_5	1/5 占空比，S1~S34 为 segment，C3~C7 为 common
LED_Duty_1_4_SEG0_34COM4_7	1/4 占空比，S0~S34 为 segment，C4~C7 为 common
LED_Duty_1_4_SEG4_34COM0_3	1/4 占空比，S4~S34 为 segment，C0~C3 为 common

LED_ ComPin

LED_ComPin使能COMx口显示驱动输出(x= 0~7)。

LED_ SegPin

LED_SegPin使能SEGx口显示驱动输出(x= 0~34)。

使用示例:

```
/* 根据 LED_InitStruct 中指定的参数初始化 LED */
LED_InitTypeDef LED_InitStruct;
LED_InitStruct.LED_FrameFre = LED_FrameFre_A64Hz; //A波形频帧64Hz
LED_InitStruct.LED_Duty = LED_Duty_1_4_SEG0_34COM4_7; //1/4占空比
LED_InitStruct.LED_ComPin = 0xF0;  //使能COM4-7
LED_InitStruct.LED_SegPin = 0xFF;  //使能SEG0-7
LED_Init(&LED_InitStruct);
```

LED_Cmd

Table 163

函数名	LED_Cmd
函数原型	void LED_Cmd(FunctionalState NewState)
功能描述	使能或失能 LED 功能
输入参数	NewState: 外设LED的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/*使能LED功能*/
LED_Cmd(ENABLE);
```

LED_StructInit

Table 164

函数名	LED_StructInit
函数原型	void LED_StructInit(LED_InitTypeDef* LED_InitStruct)
功能描述	把 LED_StructInit 中的每一个参数按缺省值填入
输入参数	LED_InitStruct: 指向结构 LED_InitTypeDef 的指针，包含了 LED 外设的配置信息
返回值	无

使用示例:

```
/* 把LED_StructInit中的每一个参数按缺省值填入 */
LED_InitTypeDef LED_InitStruct;
LED_ StructInit (&LED_InitStruct);
```

LED_Write

Table 165

函数名	LED_Write
函数原型	void LED_Write(LED_RAMRegister_Typedef LED_RAMRegister, uint8_t LED_Data)
功能描述	写入 LED 显示内容
输入参数 1	LED_RAMRegister：输出通道选择
输入参数 2	LED_Data：输出内容
返回值	无

LED_ RAMRegister

LED_ RAMRegister选择LED RAM寄存器，参阅Table 166获得这个参数的所有成员。

Table 166

LED_ RAMRegister	描述
LED_ RAMRegister _0	选择 RAM 寄存器 0
LED_ RAMRegister _1	选择 RAM 寄存器 1
LED_ RAMRegister _2	选择 RAM 寄存器 2
LED_ RAMRegister _3	选择 RAM 寄存器 3
LED_ RAMRegister _4	选择 RAM 寄存器 4
LED_ RAMRegister _5	选择 RAM 寄存器 5
LED_ RAMRegister _6	选择 RAM 寄存器 6
LED_ RAMRegister _7	选择 RAM 寄存器 7
LED_ RAMRegister _8	选择 RAM 寄存器 8
LED_ RAMRegister _9	选择 RAM 寄存器 9
LED_ RAMRegister _10	选择 RAM 寄存器 10
LED_ RAMRegister _11	选择 RAM 寄存器 11
LED_ RAMRegister _12	选择 RAM 寄存器 12
LED_ RAMRegister _13	选择 RAM 寄存器 13
LED_ RAMRegister _14	选择 RAM 寄存器 14

LED_ RAMRegister _15	选择 RAM 寄存器 15
LED_ RAMRegister _16	选择 RAM 寄存器 16
LED_ RAMRegister _17	选择 RAM 寄存器 17
LED_ RAMRegister _18	选择 RAM 寄存器 18
LED_ RAMRegister _19	选择 RAM 寄存器 19
LED_ RAMRegister _20	选择 RAM 寄存器 20
LED_ RAMRegister _21	选择 RAM 寄存器 21
LED_ RAMRegister _22	选择 RAM 寄存器 22
LED_ RAMRegister _23	选择 RAM 寄存器 23
LED_ RAMRegister _24	选择 RAM 寄存器 24
LED_ RAMRegister _25	选择 RAM 寄存器 25
LED_ RAMRegister _26	选择 RAM 寄存器 26
LED_ RAMRegister _27	选择 RAM 寄存器 27
LED_ RAMRegister _28	选择 RAM 寄存器 28
LED_ RAMRegister _29	选择 RAM 寄存器 29
LED_ RAMRegister _30	选择 RAM 寄存器 30
LED_ RAMRegister _31	选择 RAM 寄存器 31
LED_ RAMRegister _32	选择 RAM 寄存器 32
LED_ RAMRegister _33	选择 RAM 寄存器 33
LED_ RAMRegister _34	选择 RAM 寄存器 34

SC32f10xx、SC32f11xx可选(LED_RAMRegister_0- LED_RAMRegister_27)
SC32f12xx可选(LED_RAMRegister_0- LED_RAMRegister_34)

使用示例:

```
/*对RAM寄存器0写入0x20*/
LED_Write (LED_ RAMRegister_0,0x20);
```

LED_CustomModeScan

Table 167

函数名	LED_CustomModeScan
函数原型	void LED_CustomModeScan(void)
功能描述	切换 COM 口输出
输入参数	无
返回值	无

使用示例:

```
/*切换一次COM口输出 */
void LED_CustomModeScan();
```

综合使用示例（输出 LED 波形）

```
RCC_APB2Cmd(ENABLE); //开启时钟
RCC_APB2PeriphClockCmd(RCC_APB2Periph_LCD_LED, ENABLE);

LED_DeInit(); //将LED相关寄存器复位至缺省值
{
    LED_InitTypeDef LED_BaseInitStruct;
    LED_BaseInitStruct.LED_FrameFre = LED_FrameFre_A64Hz; //A波形频帧64Hz
    LED_BaseInitStruct.LED_Duty = LED_Duty_1_4_SEG0_34COM4_7; //1/4占空比
    LED_BaseInitStruct.LED_ComPin = 0xF0;   //使能COM4-7
    LED_BaseInitStruct.LED_SegPin = 0xFF;   //使能SEG0-7
    LED_Init(&LED_BaseInitStruct); //初始化LED
}
    LED_Cmd(ENABLE); //使能LED
    LED_Write(LED_RAMRegister_0,0xAA); //对RAM寄存器0写入0xAA
```

11 INT 固件库函数

赛元 SC32F1xxx系列有16个外部中断源，共占用4个中断向量，可以分别设定每个中断源的触发条件为上升、下降或上下沿。

INT 寄存器结构

INT 寄存器结构，INT_TypeDef，在文件“sc32f1xxx.h”中定义如下：

```
typedef struct
{
  __IO uint32_t INTF_IE;
  __IO uint32_t RESERVED0[7];
  __IO uint32_t INTR_IE;
  __IO uint32_t RESERVED1[7];
  __IO uint32_t INT_SEL0;
  __IO uint32_t RESERVED2[7];
  __IO uint32_t INT_SEL1;
  __IO uint32_t RESERVED3[7];
  __IO uint32_t INTF_CON;
  __IO uint32_t RESERVED4[7];
  __IO uint32_t INTR_CON;
  __IO uint32_t RESERVED5[7];
  __IO uint32_t INTF_STS;
  __IO uint32_t RESERVED6[7];
  __IO uint32_t INTR_STS;
  __IO uint32_t RESERVED7[7];
} INT_TypeDef;
```

INT 寄存器表格

Table 168

寄存器	描述
INTF_IE	INT中断下降沿使能寄存器
INTR_IE	INT中断上升沿使能寄存器
INT_SEL0	外部中断端口选择寄存器0
INT_SEL1	外部中断端口选择寄存器1
INTF_CON	外部中断下降沿控制寄存器
INTR_CON	外部中断上升沿控制寄存器
INTF_STS	外部中断下降沿标志寄存器
INTR_STS	外部中断上升沿标志寄存器

INT 固件库函数列表

Table 169

函数名	描述
INT_DeInit	INT 相关寄存器复位至缺省值
INT_Init	根据 INT_InitStruct 中指定的参数初始化外设 INT 寄存器
INT_TriggerMode	设置INT 的模式
INT_ITConfig	INT 中断使能与失能配置
INT_GetFlagStatus	检查指定的 INT 线路标志位设置与否
INT_ClearFlag	清除INT 线路挂起标志位
FT_BIT	清除下降沿标志位
RT_BIT	清除上升沿标志位

INT 固件库函数列表

INT_DeInit

Table 170

函数名	INT_DeInit
函数原型	void INT_DeInit(void)
功能描述	INT 相关寄存器复位至缺省值
输入参数	无
返回值	无

使用示例:

```
/*将INT相关寄存器复位至缺省值*/
INT_DeInit();
```

INT_Init

Table 171

函数名	INT_Init
函数原型	void INT_Init(INT_InitTypeDef* INT_InitStruct)
功能描述	根据 INT_InitStruct 中指定的参数初始化外设 INT 寄存器
输入参数	INT_InitStruct：指向结构 INT_InitTypeDef 的指针，包含了外设 INT 的配置信息
返回值	无

INT_InitTypeDef

INT_InitTypeDef 定义于文件“sc32f1xxx_int.h”：

```
typedef struct
{
uint16_t INT_Channel;
uint16_t INT_Trigger;
uint16_t INT_INTSEL;
} INT_InitTypeDef;
```

INT_Channel

INT_Channel设置INT通道，参阅Table 172获得这个参数的所有成员。

Table 172

INT_Channel	描述
INT_Channel_0	选择通道 0
INT_Channel_1	选择通道 1
INT_Channel_2	选择通道 2
INT_Channel_3	选择通道 3
INT_Channel_4	选择通道 4
INT_Channel_5	选择通道 5
INT_Channel_6	选择通道 6
INT_Channel_7	选择通道 7
INT_Channel_8	选择通道 8
INT_Channel_9	选择通道 9
INT_Channel_10	选择通道 10
INT_Channel_11	选择通道 11
INT_Channel_12	选择通道 12
INT_Channel_13	选择通道 13
INT_Channel_14	选择通道 14
INT_Channel_15	选择通道 15

INT_Trigger

INT_Trigger设置外部中断触发模式，参阅Table 173获得这个参数的所有成员。

Table 173

INT_Trigger	描述
INT_Trigger_Null	INT 无触发
INT_Trigger_Rising	INT 上升沿触发
INT_Trigger_Falling	INT 下降沿触发
INT_Trigger_Rising_Falling	INT 上升沿触发和下降沿触发

INT_INTSEL

INT_INTSEL选择外部中断端口，参阅Table 174获得这个参数的所有成员。

Table 174

INT_INTSEL	描述
INT_INTSEL_PA	外部中断端口选择 A 端口
INT_INTSEL_PB	外部中断端口选择 B 端口
INT_INTSEL_PC	外部中断端口选择 C 端口
INT_INTSEL_PD	外部中断端口选择 D 端口

SC32f10xx、SC32f12xx、SC32f15xx可选(INT_INTSEL_PA - INT_INTSEL_PC)
SC32f11xx可选(INT_INTSEL_PA - INT_INTSEL_PD)

使用示例:

```
/* 根据 INT_InitStruct 中指定的参数，将INT03 (PB3) 初始化为上升沿触发模式 */
INT_InitTypeDef   INT_InitStruct;
INT_InitStruct.INT_Channel = INT_Channel_3;
INT_InitStruct.INT_INTSEL = INT_INTSEL_PB;
INT_InitStruct.INT_Trigger = INT_Trigger_Rising;
INT_Init(&INT_InitStruct);
```

INT_TriggerMode

Table 175

函数名	INT_TriggerMode
函数原型	void INT_TriggerMode(INT_Channel_Typedef INT_Channel,INT_Trigger_TypeDef Trigger_Mode)
功能描述	设置INT 的模式
输入参数 1	INT_ Channel：INTx 通道选择，取值可参考 Table 172
输入参数 2	Trigger_Mode：选择中断触发方式，取值可参考 Table 173
返回值	无

使用示例:

```
/* INT01的中断模式为下降沿触发 */
INT_TriggerMode (INT_Channel_1,INT_IT_Falling,ENABLE);
```

INT_ITConfig

Table 176

函数名	INT_ITConfig
函数原型	void INT_ITConfig(uint16_t INT_Channel, uint16_t INT_IT, FunctionalState NewState)
功能描述	INT 中断使能与失能配置
输入参数 1	INT_ Channel：INTx 通道选择，取值可参考 Table 172
输入参数 2	INT_IT：选择中断触发方式，取值可参考 Table 177
输入参数 2	NewState：INT 中断使能或失能
返回值	无

INT_IT

INT_IT选择INT中断模式，参阅Table 177获得这个参数的所有成员。

Table 177

INT_IT	描述
INT_IT_Rising	上升沿触发
INT_IT_Falling	下降沿触发

使用示例:

```
/* 使能INT01的中断模式为下降沿触发 */
INT_ITConfig(INT_Channel_1,INT_IT_Falling,ENABLE);
```

INT_GetFlagStatus

Table 178

函数名	INT_GetFlagStatus
函数原型	FlagStatus INT_GetFlagStatus(INT_Channel_Typedef INT_Channel, INT_Flag_TypeDef INT_Flag)
功能描述	检查指定的 INT 线路标志位设置与否
输入参数 1	INT_ Channel：INTx 通道选择，取值可参考 Table 172
输入参数 2	INT_Flag：选择 INTx 为上升沿中断或下降沿中断，取值可参考 Table 179
返回值	INTx 的新状态（SET 或者 RESET）

INT_Flag

INT_Flag选择INT中断标志位，参阅Table 179获得这个参数的所有成员。

Table 179

INT_Flag	描述
INT_Flag_Rising	上升沿触发
INT_Flag_Falling	下降沿触发

使用示例:

```
/* 检查上升沿中断INT03的标志位设置与否 */
FlagStatus  INT_Status;
INT_Status = INT_GetFlagStatus(INT_Channel_3, INT_Flag_Rising);
```

INT_ClearFlag

Table 180

函数名	INT_ClearFlag
函数原型	void INT_ClearFlag(uint32_t INT_Channel)
功能描述	清除INT 线路挂起标志位
输入参数 1	INT_ Channel：INTx 通道选择(x 可以是 0-15)
输入参数 2	INT_Flag：INTx 中断标志
返回值	无

使用示例:

```
/*清除上升沿中断INT03的标志位*/
INT_ClearFlag(INT_Channel_3);
```

FT_BIT

Table 181

函数名	FT_BIT
函数原型	FT_BIT (n)
功能描述	清除下降沿中断标志
输入参数 1	n：n 可以是 0 至 15，用 n 来选择 GPIO 的端口
返回值	无

使用示例:

```
/*清除INT01下降沿中断标志*/
FT_BIT (1);
```

RT_BIT

Table 182

函数名	RT_BIT
函数原型	RT_BIT (n)
功能描述	清除上升沿中断标志

输入参数	n：n 可以是 0 至 15，用 n 来选择 GPIO 的端口
返回值	无

使用示例:

```
/*清除INT01上升沿中断标志*/
RT_BIT (1);
```

综合使用示例（PB2 设置为外部上升沿中断输入）

```
GPIO_DeInit(GPIOB);  //PB2初始化为输入带上拉
{
    GPIO_InitTypeDef GPIO_InitStruct;
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN;
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_2;
    GPIO_Init(GPIOB,&GPIO_InitStruct);
}
INT_DeInit(); //INT相关寄存器复位至缺省值
{
    INT_InitTypeDef INT_InitStruct;
    INT_InitStruct.INT_Trigger = INT_Trigger_Rising; //上升沿触发
    INT_InitStruct.INT_INTSEL = INT_INTSEL_PB;
    INT_InitStruct.INT_Channel = INT_Channel_2;  //PB2输入
    INT_Init(&INT_InitStruct);  //初始化INT
}
INT_ITConfig(INT_Channel_2,INT_IT_Rising,ENABLE); //使能INT02上升沿中断
NVIC_EnableIRQ(INT1_7_IRQn);
while(1);
void EXTI1_7_IRQHandler()
{
    if(INT_GetFlagStatus(INT_Channel_2,INT_Flag_Rising)) //检测到PB2输入上升沿
    {
        INT_ClearFlag(INT_Channel_2);
    }
}
```

12 OP 固件库函数

赛元 SC32F12xx 与SC32F15Gx系列提供了运放及可编程增益放大器，OP的输出可被选为用于AD转换器的模拟输入或者CMP0正端的模拟输入。

OP 寄存器结构

OP 寄存器结构，OP_TypeDef，在文件“sc32f1xxx.h”中定义如下：

```
typedef struct
{
 __IO uint32_t OP_CON;

} OP_TypeDef;
```

在一些型号如（SC32F15Gx）中定义如下：

```
typedef struct
{
   __IO uint32_t OP0_CON;
   __IO uint32_t OP1_CON;
   __IO uint32_t RESERVED0;
   __IO uint32_t OP2_CON;
   __IO uint32_t OPX_CFG;
   __IO uint32_t OPX_STS;
   __IO uint32_t OPX_IDE;
} OP_TypeDef;
```

OP 固件库函数列

Table 183

函数名	描述
OP_DeInit	将外设 OPx 寄存器重设为缺省值
OP_Init	根据 OP_InitStruct 中指定的参数初始化外设 OP 寄存器
OP_Cmd	使能或者失能 OP 外设
OP_StructInit	把 OP_InitStruet 中的每一个参数按缺省值填入
OP_OffsetTrimConfig	配置 OPAMP 的微调值
OP_InputSelection	配置 OPx 输入通道选择
OP_ReferenceVref	OP_REFSEL 运放模块基准 OPx_VREF 源选择
OP_GainSelection	配置 OPx 运放倍数
OP_OutputSelection	配置 OPx 的输出通道
OP_NonInvertInputSelection	配置 OPx 的运放同相端输入选择
OP_InverInputVoltage	配置 OPx 的运放反相端输入电压选择
OP_CMPTriggerCmd	使能 OP 工作模式
OP_CMPTriggerConfig	配置 OPx 的触发条件
OP_ReadOffset	获取 OPx 正负端校验值
OP_GetCMPSTA	获得 OP 比较器模式输出状态
OP_ITConfig	使能 OP 中断
OP_GetFlagStatus	获得 OP 标志状态
OP_ClearFlag	清除标志状态

OP 固件库函数详解

OP_DeInit

Table 184

函数名	OP_DeInit
函数原型	void OP_DeInit(OP_TypeDef* OPx)
功能描述	OP 相关寄存器复位至缺省值
输入参数	OPx：选择 OP 外设
返回值	无

使用示例:

```
/* OP相关寄存器复位至缺省值 */
OP_DeInit(OP);
```

OP_Init

Table 185

函数名	OP_Init
函数原型	void OP_Init(OP_TypeDef* OPx, OP_InitTypeDef* OP_InitStruct)
功能描述	OP 初始化配置
输入参数 1	OPx：选择 OP 外设
输入参数 2	OP_InitStruct：指向结构 OP_InitTypeDef 的指针，包含了外设 OP 的配置信息
返回值	无

OP_InitTypeDef

OP_InitTypeDef 定义于文件“sc32f12xx_op.h”的所有成员：

```
typedef struct
{
uint32_t OP_ShortCircuit;
uint16_t OP_FDBResisrance;
uint16_t OP_PGAGain;
uint16_t OP_Posittive;
uint16_t OP_Negative;
uint16_t OP_Output;
} OP_InitTypeDef;
```

定义于文件“sc32f15xx_op.h”的所有成员：

```
typedef struct
{
uint8_t OP_REFSEL;
uint8_t OP_CMPIM2;
uint8_t OP_CMPIM1;
uint8_t OP_OutputPin;
uint8_t OP_InvertInput;
uint8_t OP_NonInvertInput;
uint16_t OP_PGAGain;
uint32_t OP_OPRF;
uint32_t OP_FDBR;
} OP_InitTypeDef;
```

OP_REFSEL

OP_REFSEL运放模块基准OPx_VREF源选择，参阅Table 186获得这个参数的所有成员。

Table 186

OP_REFSEL;	描述
OP_RefSource_VDD	模块基准源为 VDD
OP_RefSource_VREF	模块基准源为 VREF

OP_CMPIM1和OP_CMPIM2

OP_CMPIM1和OP_CMPIM2分别为OP1和OP2比较触发条件选择，参阅 Table 187获得这个参数的所有成员。

Table 187

OP_CMPIM2	描述
OP_TriggerMode_Disable	不触发

OP_TriggerMode_RISE	上升沿触发
OP_TriggerMode_FALL	下降沿触发
OP_TriggerMode_RISE_FALL	双沿触发

OP_OutputPin

OP_OutputPin运放输出连接选择，参阅 Table 188获得这个参数的所有成员。

Table 188

OP_OutputPin	描述
OP_OutputPin_Disable	运放输出连接到 OPxO
OP_OutputPin_Enable	运放输出与 OPxO 的连接断开

OP_InvertInput

OP_InvertInput运放反相端输入选择，参阅 Table 189获得这个参数的所有成员。

Table 189

OP_InvertInput	描述
OP_InvertInput_OPxN	选用 OPxN（外部引脚）
OP_InvertInput_DAC	选用 DAC 输出
OP_InvertInput_OPRF	选用 OPRF[3:0]设定值
OP_InvertInput_Res	接反馈电阻 R2

OP_NonInvertInpu

OP_NonInvertInpu运放同相端输入选择，参阅 Table 190获得这个参数的所有成员。

Table 190

OP_NonInvertInpu	描述
OP_NonInvertInput_VSS	内部接 VSS
OP_NonInvertInput_VREF	VREFL（内建的 Vref'，不用再从外面给）
OP_NonInvertInput_VREF_Div2	输入差分模式，偏置电压为 VREF/2
OP_NonInvertInput_OPxP	选用 OPxP（外部引脚）

OP_PGAGain

OP_PGAGain选择内部增益档位，参阅Table 196获得这个参数的所有成员。

Table 191

OP_PGAGain	描述
OP_PGAGain_NonInvert4_Invert3	同相增益 4 倍，反相增益 3 倍
OP_PGAGain_NonInvert8_Invert7	同相增益 8 倍，反相增益 7 倍
OP_PGAGain_NonInvert16_Invert15	同相增益 16 倍，反相增益 15 倍
OP_PGAGain_NonInvert32_Invert31	同相增益 32 倍，反相增益 31 倍

OP_OPRF

OP_OPRF运放反相端输入电压选择，参阅 Table 192获得这个参数的所有成员。

Table 192

OP_OPRF	描述
OP_OPRF_1D16OP_VREF	运放反相端输入电压 1/16 OPx_VREF
OP_OPRF_2D16OP_VREF	运放反相端输入电压 2/16 OPx_VREF
OP_OPRF_3D16OP_VREF	运放反相端输入电压 3/16 OPx_VREF
OP_OPRF_4D16OP_VREF	运放反相端输入电压 4/16 OPx_VREF
OP_OPRF_5D16OP_VREF	运放反相端输入电压 5/16 OPx_VREF
OP_OPRF_6D16OP_VREF	运放反相端输入电压 6/16 OPx_VREF
OP_OPRF_7D16OP_VREF	运放反相端输入电压 7/16 OPx_VREF
OP_OPRF_8D16OP_VREF	运放反相端输入电压 8/16 OPx_VREF
OP_OPRF_9D16OP_VREF	运放反相端输入电压 9/16 OPx_VREF
OP_OPRF_10D16OP_VREF	运放反相端输入电压 10/16 OPx_VREF
OP_OPRF_11D16OP_VREF	运放反相端输入电压 11/16 OPx_VREF
OP_OPRF_12D16OP_VREF	运放反相端输入电压 12/16 OPx_VREF
OP_OPRF_13D16OP_VREF	运放反相端输入电压 13/16 OPx_VREF

OP_OPRF_14D16OP_VREF	运放反相端输入电压 14/16 OPx_VREF
OP_OPRF_15D16OP_VREF	运放反相端输入电压 15/16 OPx_VREF

OP_FDBR

OP_FDBR运放模块的反馈电阻R1连接选择，参阅 Table 193获得这个参数的所有成员。

Table 193

OP_FDBR	描述
OP_FDBR_VSS	内部接 VSS
OP_FDBR_OPxN	OPxN（外部引脚）

使用示例:

```
//OP0初始化配置,反相放大配置
  OP_InitTypeDef OP0_InitStruct;                    //定义PGA初始化结构体变量
  OP_StructInit(&OP0_InitStruct);
  OP0_InitStruct.OP_FDBR = OP_FDBR_OPxN;              //运放反馈电阻接反相输入端
  OP0_InitStruct.OP_GAN = OP_GAN_Invert7;             //反相7倍增益
  OP0_InitStruct.OP_InvertInput = OP_InvertInput_Res;    //运放反相端接反馈电阻
  OP0_InitStruct.OP_NonInvertInput= OP_NonInvertInput_VSS; //运放同相端接地
  OP0_InitStruct.OP_OutputPin= OP_OutputPin_Enable;      //OP0输出端口失能
  OP_Init(OP0,&OP0_InitStruct);                     //对OP0进行初始化
```

OP_ShortCircuit

OP_ShortCircuit设置同相端和反相端是否短接，参阅Table 194获得这个参数的所有成员。

Table 194

OP_ShortCircuit	描述
OP_ShortCircuit_OFF	同相端和反相端不短接
OP_ShortCircuit_ON	同相端和反相端短接

OP_FDBResisrance

OP_FDBResisrance设置反馈电阻端连接选择位，参阅Table 195获得这个参数的所有成员。

Table 195

OP_FDBResisrance	描述
OP_FDBResisrance_VSS	反馈电阻端连接选择内部接 VSS
OP_FDBResisrance_OPN	反馈电阻端连接选择接 OP_N 端口
OP_FDBResisrance_VDD	反馈电阻端连接选择接 VDD 端

OP_PGAGain

OP_PGAGain选择内部增益档位，参阅Table 196获得这个参数的所有成员。

Table 196

OP_PGAGain	描述
OP_PGAGain_NonInvert8_Invert7	同相增益 8 倍，反相增益 7 倍
OP_PGAGain_NonInvert16_Invert15	同相增益 16 倍，反相增益 15 倍
OP_PGAGain_NonInvert32_Invert31	同相增益 32 倍，反相增益 31 倍
OP_PGAGain_NonInvert64_Invert63	同相增益 64 倍，反相增益 63 倍

OP_Posittive

OP_Posittive设置OP正端信号连接选择，参阅Table 197获得这个参数的所有成员。

Table 197

OP_Posittive	描述
OP_Posittive_OPP0	OP 正端信号连接选择 OPP0
OP_Posittive_OPP1	OP 正端信号连接选择 OPP1
OP_Posittive_VSS	OP 正端信号连接选择 VSS
OP_Posittive_1_2V	OP 正端信号连接选择内部 1.2V
OP_Posittive_VDD	OP 正端信号连接选择 VDD

OP_Negative

OP_ Negative设置OP负端信号连接选择，参阅Table 198获得这个参数的所有成员。

Table 198

OP_ Negative	描述
OP_Negative_OPN	OP 负端信号连接选择 OPN
OP_Negative_PGA	OP 负端信号连接选择 PGA

OP_Output

OP_Output选择OP输出端连接选择位，参阅Table 199获得这个参数的所有成员。

Table 199

OP_Output	描述
OP_Output_OFF	输出端 OP_O 不开启
OP_Output_ON	输出端 OP_O 开启

使用示例:

```
/* 根据 OP_InitTypeDef 中指定的参数，将 OP 初始化 */
OP_InitTypeDef OP_InitStruct;
OP_InitStruct.OP_ShortCircuit = OP_ShortCircuit_OFF;
OP_InitStruct.OP_FDBResisrance = OP_FDBResisrance_VSS;
OP_InitStruct OP_PGAGain = OP_PGAGain_NonInvert8_Invert7;
OP_InitStruct.OP_Posittive = OP_Posittive_OPP0;
OP_InitStruct.OP_Negative = OP_Negative_OPN;
OP_InitStruct. OP_Output = OP_Output_ON;
OP_Init (&OP_InitStruct);
```

OP_Cmd

Table 200

函数名	OP_Cmd
函数原型	void OP_Cmd(OP_TypeDef* OPx, FunctionalState NewState)
功能描述	使能或失能 OP 模块电源
输入参数 1	OPx：选择 OP 外设
输入参数 2	NewState:OP 模块电源的状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/ * 使能OP模块电源*/
OP_Cmd (ENABLE);
```

OP_OffsetTrimConfig

Table 201

函数名	OP_OffsetTrimConfig
函数原型	void OP_OffsetTrimConfig(OP_TypeDef* OPx, uint32_t OP_TrimValueH, uint32_t OP_TrimValueL)
功能描述	配置 OPAMP 的微调值
输入参数 1	OPx：选择 OP 外设
输入参数 2	OP_TrimValueH:高端 OPAMP 的微调值
输入参数 3	OP_TrimValueL:低端 OPAMP 的微调值
返回值	无

使用示例:

```
/ *配置OPAMP的微调值 */
OP_OffsetTrimConfig(OP, 0xFF, 0x00);
```

OP_StructInit

Table 202

函数名	OP_StructInit
函数原形	void OP_StructInit(OP_InitTypeDef* OP_InitStruet)
功能描述	把 OP_InitStruet 中的每一个参数按缺省值填入
输入参数	OPx: 来选择 OP 外设
返回值	无

使用示例:

```
/* 把 OP_InitStruet 中的每一个参数按缺省值填入*/
OP_InitTypeDef OP_InitStruet;
OP_StructInit (&OP_InitStruet);
```

OP_InputSelection

Table 203

函数名	OP_InputSelection
函数原型	void OP_InputSelection(OP_Periph_TypeDef OPx,OP_InvertInput_TypeDef OPInvertInput)
功能描述	配置 OPx 输入通道选择
输入参数 1	OPx: 来选择 OP 外设
输入参数 2	OPInvertInput: 运放反相端输入选择
返回值	无

使用示例:

```
/ *配置OP1选用OPxN（外部引脚） */
OP_InputSelection (OP1, OP_InvertInput_OPxN);
```

OP_ReferenceVref

Table 204

函数名	OP_ReferenceVref
函数原型	void OP_ReferenceVref(OP_Periph_TypeDef OPx,OP_VREF_TypeDef OPVREF)
功能描述	OP_REFSEL 运放模块基准 OPx_VREF 源选择
输入参数 1	OPx: 来选择 OP 外设
输入参数 2	OPVREF: 运放模块基准 OPx_VREF 源选择
返回值	无

使用示例:

```
/ *配置OP1运放模块基准OPx_VREF源为VDD */
OP_ReferenceVref (OP1, OP_RefSource_VDD);
```

OP_GainSelection

Table 205

函数名	OP_GainSelection
函数原型	void OP_GainSelection(OP_TypeDef* OPx, OP_PGAGain_TypeDef PGAGain)
功能描述	配置 OP 增益
输入参数 1	OPx：选择 OP 外设
输入参数 2	PGAGain: 配置 OP 增益
返回值	无

使用示例:

```
/ *配置OP增益*/
OP_GainSelection(OP, OP_PGAGain_NonInvert8_Invert7);
```

OP_OutputSelection

Table 206

函数名	OP_OutputSelection
函数原型	void OP_OutputSelection(OP_TypeDef* OPx, OP_Output_TypeDef OPOutput)

功能描述	配置 OP 输出端
输入参数 1	OPx：选择 OP 外设
输入参数 2	OPOutput: 配置 OP 输出端
返回值	无

使用示例:

```
/ *配置OP输出端*/
OP_OutputSelection(OP, OP_Output_ON);
```

OP_NonInvertInputSelection（SC32f15xx 系列）

Table 207

函数名	OP_NonInvertInputSelection
函数原型	void OP_NonInvertInputSelection(OP_Periph_TypeDef OPx, OP_NonInvertInput_TypeDef OPNonInvertInput)
功能描述	配置 OPx 的运放同相端输入选择
输入参数 1	OPx: 来选择 OP 外设
输入参数 2	OPNonInvertInput: 运放同相端输入选择
返回值	无

使用示例:

```
/ *配置OP1内部接VSS */
OP_NonInvertInputSelection (OP1, OP_NonInvertInput_VSS);
```

OP_InverInputVoltage（SC32f15xx 系列）

Table 208

函数名	OP_InverInputVoltage
函数原型	void OP_InverInputVoltage(OP_Periph_TypeDef OPx, OP_OPRF_TypeDef OP_OPRF)
功能描述	配置 OPx 的运放反相端输入电压选择
输入参数 1	OPx: 来选择 OP 外设
输入参数 2	OP_OPRF: 运放反相端输入电压选择
返回值	无

使用示例:

```
/ *配置OP1运放反相端输入电压1/16 OPx_VREF */
OP_ InverInputVoltage (OP1, OP_OPRF_1D16OP_VREF);
```

OP_CMPTriggerConfig（SC32f15xx 系列）

Table 209

函数名	OP_CMPTriggerConfig
函数原型	void OP_CMPTriggerConfig(OP_Periph_TypeDef OPx, OP_TriggerMode_TypeDef TriggerMode)
功能描述	配置 OPx 的触发条件
输入参数 1	OPx: 来选择 OP 外设
输入参数 2	TriggerMode: OPx 的触发条件
返回值	无

使用示例:

```
/ *配置OP1上升沿触发 */
OP_CMPTriggerConfig (OP1, OP_TriggerMode_RISE);
```

OP_ReadOffset（SC32f15xx 系列）

Table 210

函数名	OP_ReadOffset
函数原型	uint8_t OP_ReadOffset(OP_PNMos_TypeDef OPx_Readmos)
功能描述	获取 OPx 的正负端校验值
输入参数 1	OPx_Readmos: 来选择 OPx 端口

返回值	OPx 的端口校验值

OP_PNMos_TypeDef

OP_PNMos_TypeDef返回模拟比较器选择端口，参阅Table 211获得这个类型的所有成员。

Table 211

成员	状态
OP0_Pmos	比较器 OP0 正端
OP0_Nmos	比较器 OP0 负端
OP1_Pmos	比较器 OP1 正端
OP1_Nmos	比较器 OP1 负端
OP2_Pmos	比较器 OP2 正端
OP2_Nmos	比较器 OP2 负端

使用示例:

```
/* 获得模拟比较器比较器OP0正端的校验值 */
OP_ReadOffset(OP0_Pmos)
```

OP_CMPTriggerCmd（SC32f15xx 系列）

Table 212

函数名	OP_CMPTriggerCmd
函数原型	void OP_CMPTriggerCmd(OP_Periph_TypeDef OPx, FunctionalState NewState)
功能描述	使能 OP 工作模式
输入参数 1	OPx: 来选择 OP 外设
输入参数 2	NewState: 外设 PGA 外设的校准的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

OP_GetCMPSTA（SC32f15xx 系列）

Table 213

函数名	OP_GetCMPSTA
函数原型	OP_CMPSTA_TypeDef OP_GetCMPSTA(OP_Periph_TypeDef OPx);
功能描述	获得 OP 比较器模式输出状态
输入参数	OPx：选择 OPx 外设
返回值	模拟比较器输出状态

OP_CMPSTA_TypeDef

OP_CMPSTA_TypeDef返回模拟比较器输出状态，参阅Table 214获得这个类型的所有成员。

Table 214

成员	状态
OP_CMPSTA_Low	比较器正端电压小于负端电压
OP_CMPSTA_High	比较器正端电压大于负端电压

使用示例:

```
/* 获得模拟比较器输出状态 */
OP_CMPSTA_TypeDef OP_Type;
OP_Type = OP_GetCMPSTA(OP1);
```

OP_GetFlagStatus（SC32f15xx 系列）

Table 215

函数名	OP_GetFlagStatus
函数原型	FlagStatus OP_GetFlagStatus(OP_FLAG_TypeDef OP_FLAG)
功能描述	获得 OP 标志状态
输入参数	OP_FLAG：OP 标志位选择
返回值	OP_FLAG 的新状态（SET 或者 RESET）

OP_ FLAG

Table 65给出了OP_FLAG的值。

Table 216

OP_FLAG	描述
OP_FLAG_OPCMP1	OP 模拟比较器 CMP1 中断标志位
OP_FLAG_OPCMP2	OP 模拟比较器 CMP2 中断标志位

使用示例:

```
/* 获得OP1标志状态并返回 */
FlagStatus OP_Status;
OP_Status =OP_GetFlagStatus(OP1, OP_FLAG_OPCMP1);
```

OP_ITConfig（SC32f15xx 系列）

Table 217

函数名	CMP_ITConfig
函数原型	void OP_ITConfig(uint16_t OP_IT, FunctionalState NewState)
功能描述	使能 OP 中断
输入参数 1	OP_IT：选择 OPx 外设中断使能位
输入参数 2	NewState：CMP 中断使能。可取值 ENABLE 或 DISABLE
返回值	无

OP_IT

Table 65给出了OP_IT的值。

Table 218

OP_IT	描述
OP_IT_INT	OP 中断标志位
OP_IT_OPCMP1	OP 模拟比较器 CMP1 中断
OP_IT_OPCMP2	OP 模拟比较器 CMP2 中断

使用示例:

```
/* 使能OP1中断 */
OP_ITConfig(OP_IT_OPCMP1, ENABLE);
```

OP_ClearFlag（SC32f15xx 系列）

Table 219

函数名	OP_ClearFlag
函数原型	void OP_ClearFlag(uint16_t OP_FLAG)
功能描述	清除标志状态
输入参数	OP_FLAG：中断标志位选择
返回值	无

使用示例:

```
/* 清除OP标志*/
OP_GetFlagStatus(OP_FLAG_OPCMP1);
```

综合使用示例:

```
static volatile FlagStatus ADC_Flag;
/* OP 同相反相放大配置 */

void OP_AmplifiesVoltage_Function(void)
{
  GPIO_InitTypeDef GPIO_InitStruct; //定义GPIO初始化结构体变量
  /* PA3设置为高阻输入 */
```

```
  GPIO_InitStruct.GPIO_Pin = GPIO_Pin_3;
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN_HI;
  GPIO_Init(GPIOA, &GPIO_InitStruct);
  //ADC初始化配置
  volatile uint16_t ADC_Value;
  RCC_APB2Cmd(ENABLE);                    //ADC、PGA外设挂载在APB2时钟上
  ADC_InitTypeDef ADC_InitStruct;         //定义ADC初始化结构体变量
  ADC_StructInit(&ADC_InitStruct);
  ADC_InitStruct.ADC_ConvMode = ADC_ConvMode_Single;  //ADC单次转换模式
  ADC_InitStruct.ADC_Prescaler = ADC_Prescaler_3CLOCK;//转换时间为3个周期
  ADC_InitStruct.ADC_VREF = ADC_VREF_VDD;     //参考电压为VDD
  ADC_Init(ADC, &ADC_InitStruct);         //根据结构体标志位配置ADC
  ADC_SetChannel(ADC,ADC_Channel_PGA);        //选择PGA通道
  ADC_ITConfig(ADC,ADC_IT_ADCIF,ENABLE);      //使能ADC中断
  NVIC_EnableIRQ(ADC_IRQn);
  ADC_Cmd(ADC,ENABLE);                        //使能ADC；
  //PGA初始化配置,同相相放大
  OP_InitTypeDef OP_InitStruct;           //定义PGA初始化结构体变量
  OP_InitStruct.OP_ShortCircuit = OP_ShortCircuit_OFF;//PGA同相反相不短接
  OP_InitStruct.OP_FDBResisrance = OP_FDBResisrance_VSS; //反馈电阻端连接VSS
  OP_InitStruct.OP_PGAGain = OP_PGAGain_NonInvert8_Invert7;//同相8倍反相7倍增益
  OP_InitStruct.OP_Posittive = OP_Posittive_OPP0;   //同相端OP_0
  OP_InitStruct.OP_Negative = OP_Negative_PGA;      //反相端接内部电阻
  OP_InitStruct.OP_Output = OP_Output_ON;       //连接OP_O输出端口
  OP_Init(OP,&OP_InitStruct);             //对OP进行初始化
  OP_Cmd(OP,ENABLE);                      //OP使能
  while(1)
  {
    ADC_SoftwareStartConv(ADC); //开启转换
    while(ADC_Flag == RESET); //等待ADC转换完成
    ADC_Value = ADC_GetConversionValue(ADC);  //获取ADC转换值
    printf("\r\n ADC Conversion Value is 0x%x \r\n",ADC_Value);
    for (int i=0;i<5000;i++);
  }
}

void OP_AmplifiesVoltage_ADCHandler(void)
{
  if(ADC_GetFlagStatus(ADC,ADC_Flag_ADCIF)) //判断ADC标志位
  {
    ADC_ClearFlag(ADC,ADC_Flag_ADCIF);  //清除ADC标志位
    ADC_Flag = SET;  //自定义标志置起
  }
}
```

13 OPTION 固件库函数

芯片内部有单独的一块Flash区域用于保存客户的上电初始值设置，此区域称为Customer Option区域。用户在烧写IC时将此部分代码写入IC内部，IC在复位初始化时，就会将此设置调入SFR作为初始设置。对OPTION的操作可以对LVR、看门狗、JTAG引脚等设置项做出改变。

OPTION 固件库函数列

Table 220

函数名	描述
OPTION_WDTCmd	使能或失能 WDT 外设
OPTION_LVRConfig	设置 LVR 电压档位
OPTION_JTAGCmd	使能或失能 JTAG 功能

OPTION 固件库函数详解

OPTION_WDTCmd

Table 221

函数名	OPTION_WDTCmd
函数原型	void OPTION_WDTCmd(FunctionalState NewState)
功能描述	使能或失能 WDT 外设
输入参数	NewState: WDT 外设的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/*启动WDT */
OPTION_WDT_Cmd(ENABLE);
```

OPTION_LVRConfig

Table 222

函数名	OPTION_LVRConfig
函数原型	void OPTION_LVRConfig(OPTION_LVR_TypeDef OPTION_LVR)
功能描述	设置 LVR 电压档位
输入参数	LVR 电压档位
返回值	无

OPTION_LVR

OPTION_LVR选择LVR电压，参阅Table 223获得这个参数的所有成员。

Table 223

OPTION_LVR	描述
OPTION_LVR_DISABLE	LVR 不使能
OPTION_LVR_1_9V	LVR 电压为 1.9V
OPTION_LVR_2_9V	LVR 电压为 2.9V
OPTION_LVR_3_7V	LVR 电压为 2.7V
OPTION_LVR_4_3V	LVR 电压为 3.3V

使用示例:

```
/* LVR电压档位设置为2.9V */
OPTION_LVRConfig (OPTION_LVR_2_9V);
```

OPTION_JTAGCmd

Table 224

函数名	OPTION_JTAGCmd
函数原型	void OPTION_JTAGCmd(FunctionalState NewState)
功能描述	使能或失能 JTAG 功能
输入参数	NewState: JTAG 功能的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/ * JTAG管脚可以作为普通IO使用*/
OPTION_JTAGCmd (DISABLE);
```

14 PWM 固件库函数

赛元SC32f10xx，SC32f11xx，SC32f12xx系列提供了8路共用周期、单独可调占空比的16位PWM输出，PWM输出可设置正反向。

PWM 寄存器结构

PWM 寄存器结构，PWM_TypeDef，在文件“sc32f1xxx.h”中定义如下：

```
typedef struct
{
  __IO uint32_t PWM_CON;
  __IO uint32_t PWM_CHN;
  __IO uint32_t PWM_STS;
  __IO uint32_t PWM_INV;
  __IO uint32_t PWM_DFR;
  __IO uint32_t PWM_FLT;
  __IO uint32_t PWM_CYCLE;
  __IO uint32_t RESERVED[5];
  __IO uint32_t PWM_DT[8];
} PWM_TypeDef;
```

PWM **寄存器表格**

Table 225

寄存器	描述
PWM_CON	PWM 控制寄存器
PWM_CHN	PWM 通道设置寄存器
PWM_STS	PWM 状态标志寄存器
PWM_DFR	PWM 死区设置寄存器
PWM_FLT	PWM 故障检测设置寄存器
PWM_CYCLE	PWM 周期寄存器
PWM_DTn	PWM 通道 n duty 寄存器

PWM 固件库函数列表

Table 226

函数名	描述
PWM_DeInit	PWM 相关寄存器复位至缺省值
PWM_Init	PWM 初始化配置
PWM_StructInit	把 PWM_InitStruct 中的每一个参数按缺省值填入
PWM_RisingDeadTimeConfig	PWM 互补模式下，上升沿死区时间配置
PWM_FallingDeadTimeConfig	PWM 互补模式下，下降沿死区时间配置
PWM_Cmd	PWM 功能开关配置
PWM_SetPrescaler	PWM 的频率设置
PWM_GetPrescaler	获取 PWM 的频率值
PWM_SetCycle	PWM 的周期设置
PWM_GetCycle	获取 PWM 的周期值
PWM_SetDuty	PWM 占空比长度设置
PWM_GetDuty	PWM 获取占空比长度值
PWM_FDInit	PWM 故障检测模式设置
PWM_FDStructInit	把 PWM_FDInitStruct 中的每一个参数按缺省值填入
PWM_FDCmd	PWM 故障检测使能
PWM_ITConfig	PWM 中断配置函数

PWM_GetFlagStatus	获取 PWM 中断标志位
PWM_ClearFlag	清除 PWM 中断标志位

PWM 固件库函数详解

PWM_DeInit

Table 227

函数名	PWM_DeInit
函数原型	void PWM_DeInit(PWM_TypeDef* PWMx)
功能描述	PWM 相关寄存器复位至缺省值
输入参数	PWMx：x 可以是 0，来选择 PWM 外设
返回值	无

使用示例:

```
/* PWM0寄存器复位*/
PWM_DeInit(PWM0);
```

PWM_Init

Table 228

函数名	PWM_Init
函数原型	void PWM_Init(PWM_TypeDef* PWMx, PWM_InitTypeDef* PWM_InitStruct)
功能描述	PWM 初始化配置
输入参数 1	PWMx：x 可以是 0，来选择 PWM 外设
输入参数 2	PWM_InitStruct：指向结构 PWM_InitTypeDef 的指针，包含了外设 PWM 的配置信息
返回值	无

PWM_InitTypeDef

PWM_InitTypeDef 定义于文件“sc32f1xxx_pwm.h”：

```
typedef struct
{
uint16_t PWM_Prescaler;
uint16_t PWM_AlignedMode;
uint16_t PWM_WorkMode;
uint16_t PWM_Cycle;
uint32_t PWM_OutputChannel;
uint32_t PWM_LowPolarityChannl;
}
```

PWM_Prescaler

PWM_Prescaler设置PWM时钟频率，参阅Table 229获得这个参数的所有成员。

Table 229

PWM_Prescaler	描述
PWM_PRESCALER_DIV1	时钟频率为 1 分频
PWM_PRESCALER_DIV2	时钟频率为 2 分频
PWM_PRESCALER_DIV4	时钟频率为 4 分频
PWM_PRESCALER_DIV8	时钟频率为 8 分频
PWM_PRESCALER_DIV16	时钟频率为 16 分频
PWM_PRESCALER_DIV32	时钟频率为 32 分频
PWM_PRESCALER_DIV64	时钟频率为 64 分频
PWM_PRESCALER_DIV128	时钟频率为 128 分频
PWM_PRESCALER_DIV256	时钟频率为 256 分频

PWM_AlignedMode

PWM_AlignedMode设置PWM对齐模式，参阅Table 230获得这个参数的所有成员。

Table 230

PWM_AlignedMode	描述

PWM_AlignmentMode_Edge	边沿对齐
PWM_AlignmentMode_Center	中心对齐

PWM_WorkMode

PWM_WorkMode设置PWM工作模式，参阅Table 231获得这个参数的所有成员。

Table 231

PWM_WorkMode	描述
PWM_WorkMode_Independent	独立模式
PWM_WorkMode_Complementary	互补模式

PWM_Cycle

PWM_Cycle设置PWM周期，此数值代表 PWM输出波形的 (周期 – 1); 也就是说PWM输出的周期值（PWMPD[15:0] + 1 ）* PWM时钟。

PWM_OutputChannel

PWM_OutputChannel设置PWM输出通道，参阅Table 232获得这个参数的所有成员。

Table 232

PWM_OutputChannel	描述
PWMChannel_Less	不选择通道
PWM_Channel_0	选择通道 0
PWM_Channel_1	选择通道 1
PWM_Channel_2	选择通道 2
PWM_Channel_3	选择通道 3
PWM_Channel_4	选择通道 4
PWM_Channel_5	选择通道 5
PWM_Channel_6	选择通道 6
PWM_Channel_7	选择通道 7
PWM_Channel_All	选择所有通道

PWM_LowPolarityChannl

PWM_LowPolarityChannl设置PWM反向输出通道，参阅Table 232获得这个参数的所有成员。

使用示例:

```
/* 根据 PWM_InitStruct 中指定的参数，初始化PWM0 */
PWM_InitTypeDef  PWM_InitStruct;
PWM_InitStruct.PWM_AlignedMode = PWM_AlignmentMode_Edge;  //配置为边沿对齐模式
PWM_InitStruct.PWM_Prescaler = PWM_PRESCALER_DIV8;  //频率为Fsource/8
PWM_InitStruct.PWM_OutputChannel = PWM_Channel_4 | PWM_Channel_5; //输出通道为PWM4和PWM5
PWM_InitStruct.PWM_LowPolarityChannl = PWM_Channel_5;   //PWM5输出反向
PWM_InitStruct.PWM_WorkMode = PWM_Mode_Complementary; //配置为互补模式
PWM_InitStruct.PWM_Cycle = 999;   //PWM输出的周期值为(999+1) * PWM时钟
PWM_Init(PWM0,&PWM_InitStruct);
```

PWM_StructInit

Table 233

函数名	PWM_StructInit
函数原形	void PWM_StructInit(PWM_InitTypeDef* PWM_InitStruct)
功能描述	把 PWM_InitStruct 中的每一个参数按缺省值填入
输入参数	PWM_InitStruct：指向结构 PWM_InitTypeDef 的指针，待初始化
返回值	无

PWM_InitStruct

PWM_InitStruct的成员缺省值如Table 234所示。

Table 234

成员	缺省值
PWM_AlignedMode	PWM_AlignmentMode_Edge
PWM_Cycle	0x0000

PWM_LowPolarityChannl	PWMChannel_Less
PWM_OutputChannel	PWMChannel_Less
PWM_Prescaler	PWM_PRESCALER_DIV1
PWM_WorkMode	PWM_WorkMode_Independent

使用示例:

```
/* 把 PWM_InitStruct 中的每一个参数按缺省值填入*/
PWM_InitTypeDef   PWM_InitStruct;
PWM_StructInit(&PWM_InitStruct);
```

PWM_RisingDeadTimeConfig

Table 235

函数名	PWM_RisingDeadTimeConfig
函数原型	void PWM_RisingDeadTimeConfig(PWM_TypeDef* PWMx, uint8_t PWM_RisingDeadTime)
功能描述	PWM 互补模式下，上升沿死区时间配置
输入参数 1	PWMx：x 可以是 0，来选择 PWM 外设
输入参数 2	PWM_RisingDeadTime：PWM 上升沿死区时间 = 4*x / FpCLK0 (x 为填入寄存器的值，取值范围为 0~15)
返回值	无

使用示例:

```
/* PWM0上升沿死区时间配置 */
PWM_RisingDeadTimeConfig(PWM0,8);  //PWM上升沿死区时间为32 / FpCLK0
```

PWM_FallingDeadTimeConfig

Table 236

函数名	PWM_FallingDeadTimeConfig
函数原型	void PWM_FallingDeadTimeConfig(PWM_TypeDef* PWMx, uint8_t PWM_FallingDeadTime)
功能描述	PWM 互补模式下，下降沿死区时间配置
输入参数 1	PWMx：x 可以是 0，来选择 PWM 外设
输入参数 2	PWM_FallingDeadTime：PWM 下降沿死区时间 = 4*x / FpCLK0 (x 为填入寄存器的值，取值范围为 0~15)
返回值	无

使用示例:

```
/* PWM0下降沿死区时间配置 */
PWM_FallingDeadTimeConfig(PWM0,8);  //PWM下降沿死区时间为32 / FpCLK0
```

PWM_Cmd

Table 237

函数名	PWM_Cmd
函数原型	void PWM_Cmd(PWM_TypeDef* PWMx, FunctionalState NewState)
功能描述	PWM 功能启动/关闭选择
输入参数 1	PWMx：x 可以是 0，来选择 PWM 外设
输入参数 2	NewState：PWM 使能或失能
返回值	无

使用示例:

```
/*使能PWM1*/
PWM_Cmd(PWM1,ENABLE);
```

PWM_SetPrescaler

Table 238

函数名	PWM_SetPrescaler
函数原型	void PWM_SetPrescaler(PWM_TypeDef* PWMx, PWM_Prescaler_TypeDef PWM_Prescaler)

功能描述	设置 PWM 的频率
输入参数 1	PWMx：x 可以是 0，来选择 PWM 外设
输入参数 2	PWM_Prescaler：选择 PWM 的频率，取值可参考 Table 229。
返回值	无

使用示例:

```
/*设置PWM1频率为Fsource/8*/
PWM_SetPrescaler(PWM1,PWM_PRESCALER_DIV8);
```

PWM_GetPrescaler

Table 239

函数名	PWM_GetPrescaler
函数原型	PWM_Prescaler_TypeDef PWM_GetPrescaler(PWM_TypeDef* PWMx)
功能描述	获取 PWM 的频率值
输入参数	PWMx：x 可以是 0，来选择 PWM 外设
返回值	PWM 的频率，返回表格 Table 229 内的值。

使用示例:

```
/* 获取PWM1的频率 */
PWM_Prescaler_TypeDef   PWM_Prescaler;
PWM_Prescaler = PWM_GetPrescaler(PWM1);
```

PWM_SetCycle

Table 240

函数名	PWM_SetCycle
函数原型	void PWM_SetCycle(PWM_TypeDef* PWMx, uint32_t PWM_Cycle)
功能描述	PWM 的周期设置
输入参数 1	PWMx：x 可以是 0，来选择 PWM 外设
输入参数 2	PWM_Cycle：PWM 的周期值
返回值	无

使用示例:

```
/* 设置PWM1的周期值 */
PWM_SetCycle(PWM1,999);  //PWM输出的周期值为(999+1) * PWM时钟
```

PWM_GetCycle

Table 241

函数名	PWM_GetCycle
函数原型	uint16_t PWM_GetCycle(PWM_TypeDef* PWMx)
功能描述	获取 PWM 的周期值
输入参数 1	PWMx：x 可以是 0，来选择 PWM 外设
返回值	PWM 的周期值

使用示例:

```
/* 获取PWM1的周期值 */
uint16_t   PWM_Cycle;
PWM_Cycle = PWM_GetCycle(PWM1);
```

PWM_SetDuty

Table 242

函数名	PWM_SetDuty
函数原型	void PWM_SetDuty(PWM_TypeDef* PWMx, PWM_Channel_Typedef PWM_Channel, uint16_t PWM_Duty)
功能描述	PWM 占空比长度设置
输入参数 1	PWMx：x 可以是 0，来选择 PWM 外设

输入参数 2	PWM_Channel：PWM 通道选择，取值可参考 Table 54。
输入参数 3	PWM_Duty：PWM 的占空比长度
返回值	无

使用示例:

```
/* 配置PWM06的占空比长度 */
PWM_SetDuty(PWM0, PWM_Channel_6,500);
```

PWM_GetDuty

Table 243

函数名	PWM_GetDuty
函数原型	uint16_t PWM_GetDuty(PWM_TypeDef* PWMx, PWM_Channel_Typedef PWM_Channel)
功能描述	获取 PWM 的占空比长度值
输入参数 1	PWMx：x 可以是 0，来选择 PWM 外设
输入参数 2	PWM_Channel：PWM 通道选择，取值可参考 Table 54。
返回值	PWM 的占空比长度值

使用示例:

```
/* 获取PWM06的占空比长度值 */
uint16_t  PWM_Duty;
PWM_Duty = PWM_GetDuty(PWM0, PWM_Channel_6);
```

PWM_FDInit

Table 244

函数名	PWM_FDInit
函数原型	void PWM_FDInit(PWM_TypeDef* PWMx, PWM_FDInitTypeDef* PWM_FDInitStruct)
功能描述	PWM 故障检测模式设置
输入参数 1	PWMx：x 可以是 0，来选择 PWM 外设
输入参数 2	PWM_FDInitStruct：指向结构 PWM_FDInitTypeDef 的指针，包含了 PWM 故障检测的配置信息
返回值	无

PWM_FDInitTypeDef

PWM_FDInitTypeDef 定义于文件“sc32f1xxx_pwm.h”：

```
typedef struct
{
uint16_t PWM_FDMode;
uint16_t PWM_FDVoltage;
uint16_t PWM_FDFilteringTime;
} PWM_FDInitTypeDef;
```

PWM_FDMode

PWM_FDMode设置PWM故障检测模式，参阅Table 245获得这个参数的所有成员。

Table 245

PWM_FDMode	描述
PWM_FDMode_Latch	锁存模式
PWM_FDMode_Immediate	立即模式

PWM_FDVoltage

PWM_FDVoltage设置PWM故障检测有效电平，参阅Table 246获得这个参数的所有成员。

Table 246

PWM_FDVoltage	描述
PWM_FDVoltage_Low	故障检测低电平有效
PWM_FDVoltage_High	故障检测高电平有效

PWM_FDFilteringTime

PWM_FDFilteringTime设置PWM故障检测滤波时间，参阅Table 247获得这个参数的所有成员。

Table 247

PWM_FDFilteringTime	描述

PWM_FilteringTime_0us	滤波时间为 0us
PWM_FilteringTime_1us	滤波时间为 1us
PWM_FilteringTime_4us	滤波时间为 4us
PWM_FilteringTime_16us	滤波时间为 16us

使用示例:

```
/* PWM0故障检测模式设置 */
PWM_FDInitTypeDef   PWM_FDInitStruct;
PWM_FDInitStruct.PWM_FDMode = PWM_FDMode_Latch;  //设置为锁存模式
PWM_FDInitStruct. PWM_FDFilteringTime = PWM_FilteringTime_1us; //滤波时间为1us
PWM_FDInitStruct.PWM_FDVoltage = PWM_FDVoltage_High; //故障检测高电平有效
PWM_FDInit(PWM0,&PWM_FDInitStruct);
```

PWM_FDStructInit

Table 248

函数名	PWM_FDStructInit
函数原形	void PWM_FDStructInit(PWM_FDInitTypeDef* PWM_FDInitStruct)
功能描述	把 PWM_FDInitStruct 中的每一个参数按缺省值填入
输入参数	PWM_FDInitStruct：指向结构 PWM_FDInitTypeDef 的指针，待初始化
返回值	无

PWM_FDInitStruct

PWM_FDInitStruct的成员缺省值如Table 249所示。

Table 249

成员	缺省值
PWM_FDMode	PWM_FDMode_Latch
PWM_FDFilteringTime	PWM_FilteringTime_0us
PWM_FDVoltage	PWM_FDVoltage_Low

使用示例:

```
/* 把 PWM_FDInitStruct 中的每一个参数按缺省值填入*/
PWM_FDInitTypeDef   PWM_FDInitStruct;
PWM_FDStructInit(&PWM_FDInitStruct);
```

PWM_FDCmd

Table 250

函数名	PWM_FDCmd
函数原型	void PWM_FDCmd(PWM_TypeDef* PWMx, FunctionalState NewState)
功能描述	PWM 故障检测功能开启/关闭
输入参数 1	PWMx：x 可以是 0，来选择 PWM 外设
输入参数 2	NewState：故障检测功能开启/关闭
返回值	无

使用示例:

```
/* 开启多功能PWM故障检测模式 */
PWM_FDCmd(PWM0,ENABLE);
```

PWM_ITConfig

Table 251

函数名	PWM_ITConfig
函数原型	void PWM_ITConfig(PWM_TypeDef* PWMx, uint16_t PWM_IT, FunctionalState NewState)
功能描述	PWM 中断配置函数
输入参数 1	PWMx：x 可以是 0，来选择 PWM 外设
输入参数 2	PWM_IT：PWM 中断请求，取值可参考 Table 252。
输入参数 3	NewState：PWM 中断开启/关闭

返回值	无

PWM_IT

PWM_IT 用来使能或者失能 PWM 中断。参阅Table 252获得这个参数的所有成员。。

Table 252

PWM_IT	描述
PWM_IT_INTEN	PWM 中断请求

使用示例:

```
/ *开启PWM0的中断*/
PWM_ITConfig(PWM0, PWM_IT_INTEN, ENABLE);
```

PWM_GetFlagStatus

Table 253

函数名	PWM_GetFlagStatus
函数原型	FlagStatus PWM_GetFlagStatus(PWM_TypeDef* PWMx, uint16_t PWM_FLAG)
功能描述	获取 PWM 中断标志位
输入参数 1	PWMx：x 可以是 0，来选择 PWM 外设
输入参数 2	PWM_FLAG：PWM 中断标志，取值可参考 Table 61。
返回值	PWMx 的新状态（SET 或者 RESET）

PWM_FLAG

Table 254给出了PWM_FLAG的值。

Table 254

PWM_FLAG	描述
PWM_Flag_PWMIF	PWM 中断请求标志
PWM_Flag_FLTSTA	PWM 故障检测状态标志

使用示例:

```
/* 获取PWM0中断请求标志位 */
FlagStatus  PWM_Status;
PWM_Status  = PWM_GetFlagStatus(PWM0, PWM_Flag_PWMIF);
```

PWM_ClearFlag

Table 255

函数名	PWM_ClearFlag
函数原型	void PWM_ClearFlag(PWM_TypeDef* PWMx, uint16_t PWM_FLAG)
功能描述	清除 PWM 中断标志位
输入参数 1	PWMx：x 可以是 0，来选择 PWM 外设
输入参数 2	PWM_FLAG：PWM 中断标志，取值可参考 Table 61。
返回值	无

使用示例:

```
/* 清除PWM0中断请求标志位 */
PWM_ ClearFlag(PWM0, PWM_Flag_PWMIF);
```

综合使用示例:（PWM0 互补模式输出占空比为 20%和 80%的波形，并开启故障检测功能）

```
RCC_APB0Cmd(ENABLE);  //开启PWM0时钟
RCC_APB0PeriphClockCmd(RCC_APB0Periph_PWM0, ENABLE);

PWM_DeInit(PWM0); //将PWM0相关寄存器复位至缺省值
{
  PWM_InitTypeDef PWM_InitStruct;
  PWM_InitStruct.PWM_AlignedMode = PWM_AlignmentMode_Edge; //边沿对齐
  PWM_InitStruct.PWM_Prescaler = PWM_PRESCALER_DIV16;  //16分频
  PWM_InitStruct.PWM_OutputChannel = PWM_Channel_4 | PWM_Channel_5;//使能PWM4和PWM5输出
```

```
  PWM_InitStruct.PWM_LowPolarityChannl = PWM_Channel_5; //PWM5输出波形反向
  PWM_InitStruct.PWM_WorkMode = PWM_WorkMode_Complementary; //互补模式
  PWM_InitStruct.PWM_Cycle = 999; //周期1000
  PWM_Init(PWM0, &PWM_InitStruct); //初始化PWM0
}
PWM_SetDuty(PWM0, PWM_Channel_4, 200); //PWM4占空比20%，PWM5占空比80%
PWM_Cmd(PWM0, ENABLE); //使能PWM0;
{
  PWM_FDInitTypeDef PWM_FDInitStruct;
  PWM_FDInitStruct.PWM_FDMode = PWM_FDMode_Latch; //故障检测为锁存模式
  PWM_FDInitStruct.PWM_FDVoltage = PWM_FDVoltage_Low; //低电平有效
  PWM_FDInitStruct.PWM_FDFilteringTime = PWM_FilteringTime_1us; //滤波时间1us
  PWM_FDInit(PWM0, &PWM_FDInitStruct); //PWM0故障检测初始化
}
PWM_FDCmd(PWM0, ENABLE); //使能PWM0故障检测功能
PWM_ITConfig(PWM0, PWM_IT_INTEN, ENABLE); //使能PWM0中断
while(1)
{
  if(PWM_GetFlagStatus(PWM0, PWM_Flag_FLTSTA)) //故障检测有效
  {
    PWM_ClearFlag(PWM0, PWM_Flag_FLTSTA);
  }
}
```

15 LEDPWM 固件库函数

赛元SC32f10xx，SC32f11xx，SC32f12xx系列系列提供了多路共用周期、单独可调占空比的8位PWM1输出。

LEDPWM 寄存器结构

LEDPWM 寄存器结构，LEDPWM_TypeDef，在文件“sc32f1xxx.h”中定义如下：

```
typedef struct
{
  __IO uint32_t LEDPWM_CON;
  __IO uint32_t LEDPWM_CHN;
  __IO uint32_t LEDPWM_STS;
  __IO uint32_t LEDPWM_INV;
  __IO uint32_t RESERVED0[2];
  __IO uint32_t LEDPWM_CYCLE;
  __IO uint32_t RESERVED1[5];
  __IO uint32_t LEDPWM_DT[32];
} LEDPWM_TypeDef;
```

LEDPWM 寄存器表格

Table 256

寄存器	描述
LEDPWM_CON	LEDPWM 控制寄存器
LEDPWM_CHN	LEDPWM 通道设置寄存器
LEDPWM_STS	LEDPWM 状态标志寄存器
LEDPWM_INV	LEDPWM 故障检测设置寄存器
LEDPWM_CYCLE	LEDPWM 周期寄存器
LEDPWM_DTn	LEDPWM 通道 n duty 寄存器

LEDPWM 固件库函数列表

Table 257

函数名	描述
LEDPWM_DeInit	LEDPWM 相关寄存器复位至缺省值
LEDPWM_Init	LEDPWM 初始化配置
LEDPWM_StructInit	把 LEDPWM_InitStruct 中的每一个参数按缺省值填入
LEDPWM_Cmd	LEDPWM 功能开关配置
LEDPWM_SetPrescaler	LEDPWM 的频率设置
LEDPWM_GetPrescaler	获取 LEDPWM 的频率值
LEDPWM_SetCycle	LEDPWM 的周期设置
LEDPWM_GetCycle	获取 LEDPWM 的周期值
LEDPWM_SetDuty	LEDPWM 占空比长度设置
LEDPWM_GetDuty	获取 LEDPWM 占空比长度值
LEDPWM_ITConfig	LEDPWM 中断配置函数
LEDPWM_GetFlagStatus	获取 LEDPWM 中断标志位
LEDPWM_ClearFlag	清除 LEDPWM 中断标志位

LEDLEDPWM 固件库函数详解

LEDPWM_DeInit

Table 258

函数名	LEDPWM_DeInit

函数原型	void LEDPWM_DeInit(void)
功能描述	LEDPWM 相关寄存器复位至缺省值
返回值	无

使用示例:

```
/* LEDPWM0寄存器复位*/
LEDPWM_DeInit();
```

LEDPWM_Init

Table 259

函数名	LEDPWM_Init
函数原型	void LEDPWM_Init(LEDPWM_InitTypeDef* LEDPWM_InitStruct)
功能描述	LEDPWM 初始化配置
输入参数	LEDPWM_InitStruct：指向结构 LEDPWM_InitTypeDef 的指针，包含了外设 LEDPWM 的配置信息
返回值	无

LEDPWM_InitTypeDef

LEDPWM_InitTypeDef 定义于文件“sc32f1xxx_ledpwm.h”：

```
typedef struct
{
uint16_t LEDPWM_Prescaler;
uint16_t LEDPWM_AlignedMode;
uint8_t LEDPWM_Cycle;
uint64_t LEDPWM_OutputChannel;
uint64_t LEDPWM_LowPolarityChannl;
}
```

LEDPWM_Prescaler

LEDPWM_Prescaler设置LEDPWM时钟频率，参阅Table 260获得这个参数的所有成员。

Table 260

LEDPWM_Prescaler	描述
LEDPWM_PRESCALER_DIV1	时钟频率为 1 分频
LEDPWM_PRESCALER_DIV2	时钟频率为 2 分频
LEDPWM_PRESCALER_DIV4	时钟频率为 4 分频
LEDPWM_PRESCALER_DIV8	时钟频率为 8 分频
LEDPWM_PRESCALER_DIV16	时钟频率为 16 分频
LEDPWM_PRESCALER_DIV32	时钟频率为 32 分频
LEDPWM_PRESCALER_DIV64	时钟频率为 64 分频
LEDPWM_PRESCALER_DIV128	时钟频率为 128 分频
LEDPWM_PRESCALER_DIV256	时钟频率为 256 分频

LEDPWM_AlignedMode

LEDPWM_AlignedMode设置LEDPWM对齐模式，参阅Table 230获得这个参数的所有成员。

Table 261

LEDPWM_AlignedMode	描述
LEDPWM_AlignmentMode_Edge	边沿对齐
LEDPWM_AlignmentMode_Center	中心对齐

LEDPWM_Cycle

LEDPWM_Cycle设置LEDPWM周期，此数值代表 LEDPWM输出波形的 (周期 – 1); 也就是说LEDPWM输出的周期值为（LEDPWMPD[7:0] + 1 ）* LEDPWM时钟。

LEDPWM_OutputChannel

LEDPWM_OutputChannel设置LEDPWM输出通道，参阅Table 262获得这个参数的所有成员。

Table 262

LEDPWM_OutputChannel	描述

LEDPWMChannel_Less	不选择通道
LEDPWM_Channel_0	选择通道 0
LEDPWM_Channel_1	选择通道 1
LEDPWM_Channel_2	选择通道 2
LEDPWM_Channel_3	选择通道 3
LEDPWM_Channel_4	选择通道 4
LEDPWM_Channel_5	选择通道 5
LEDPWM_Channel_6	选择通道 6
LEDPWM_Channel_7	选择通道 7
略	略
LEDPWM_Channel_30	选择通道 30
LEDPWM_Channel_31	选择通道 31
LEDPWM_Channel_All	选择所有通道

（不同芯片型号通道选择不一样，具体需以规格书为准）

LEDPWM_LowPolarityChannl

LEDPWM_LowPolarityChannl设置LEDPWM反向输出通道，参阅Table 262获得这个参数的所有成员。

使用示例:

```
LEDPWM_InitStruct.LEDPWM_AlignedMode = PWM_AlignmentMode_Edge;//LEDPWM对齐模式为边沿对齐
LEDPWM_InitStruct.LEDPWM_Cycle = (uint16_t)(32000000 / 256 / 1000);  //LEDPWM的频率为1000Hz
LEDPWM_InitStruct.LEDPWM_OutputChannel = LEDPWM_Channel_0;    //使能LEDPWM0输出
LEDPWM_InitStruct.LEDPWM_Prescaler = PWM_PRESCALER_DIV256;  //LEDPWM时钟分频为1分配
LEDPWM_Init(&LEDPWM_InitStruct);       //配置LEDPWM
```

LEDPWM_StructInit

Table 263

函数名	LEDPWM_StructInit
函数原形	void LEDPWM_StructInit(LEDPWM_InitTypeDef* LEDPWM_InitStruct)
功能描述	把 LEDPWM_InitStruct 中的每一个参数按缺省值填入
输入参数	LEDPWM_InitStruct：指向结构 LEDPWM_InitTypeDef 的指针，待初始化
返回值	无

LEDPWM_InitStruct

LEDPWM_InitStruct的成员缺省值如Table 234所示。

Table 264

成员	缺省值
LEDPWM_AlignedMode	LEDPWM_AlignmentMode_Edge
LEDPWM_Cycle	0x0000
LEDPWM_LowPolarityChannl	LEDPWMChannel_Less
LEDPWM_OutputChannel	LEDPWMChannel_Less
LEDPWM_Prescaler	LEDPWM_PRESCALER_DIV1

使用示例:

```
/* 把 LEDPWM_InitStruct 中的每一个参数按缺省值填入*/
LEDPWM_InitTypeDef   LEDPWM_InitStruct;
LEDPWM_StructInit(&LEDPWM_InitStruct);
```

LEDPWM_Cmd

Table 265

函数名	LEDPWM_Cmd
函数原型	void LEDPWM_Cmd(FunctionalState NewState)
功能描述	LEDPWM 功能启动/关闭选择
输入参数	NewState：LEDPWM 使能或失能
返回值	无

使用示例:

```
/*使能LEDPWM*/
LEDPWM_Cmd(ENABLE);
```

LEDPWM_SetPrescaler

Table 266

函数名	LEDPWM_SetPrescaler
函数原型	void LEDPWM_SetPrescaler(LEDPWM_Prescaler_TypeDef LEDPWM_Prescaler)
功能描述	设置 LEDPWM 的频率
输入参数	LEDPWM_Prescaler：选择 LEDPWM 的频率，取值可参考 Table 260。
返回值	无

使用示例:

```
/*设置LEDPWM频率为Fsource/8*/
LEDPWM_SetPrescaler(LEDPWM_PRESCALER_DIV8);
```

LEDPWM_GetPrescaler

Table 267

函数名	LEDPWM_GetPrescaler
函数原型	LEDPWM_Prescaler_TypeDef LEDPWM_GetPrescaler(void)
功能描述	获取 LEDPWM 的频率值
返回值	LEDPWM 的频率，返回表格 Table 51 内的值。

使用示例:

```
/* 获取LEDPWM的频率 */
LEDPWM_Prescaler_TypeDef   LEDPWM_Prescaler;
LEDPWM_Prescaler = LEDPWM_GetPrescaler();
```

LEDPWM_SetCycle

Table 268

函数名	LEDPWM_SetCycle
函数原型	void LEDPWM_SetCycle(uint8_t LEDPWM_Cycle)
功能描述	LEDPWM 的周期设置
输入参数 1	LEDPWM_Cycle：LEDPWM 的周期值
返回值	无

使用示例:

```
/* 设置LEDPWM的周期值 */
LEDPWM_SetCycle(LEDPWM1,999);  //LEDPWM输出的周期值为(999+1) * LEDPWM时钟
```

LEDPWM_GetCycle

Table 269

函数名	LEDPWM_GetCycle
函数原型	Uint8_t LEDPWM_GetCycle(void)
功能描述	获取 LEDPWM 的周期值
输入参数 1	LEDPWMx：x 可以是 0 或 1，来选择 LEDPWM 外设
返回值	LEDPWM 的周期值

使用示例:

```
/* 获取LEDPWM的周期值 */
Uint8_t   LEDPWM_Cycle;
LEDPWM_Cycle = LEDPWM_GetCycle(LEDPWM1);
```

LEDPWM_SetDuty

Table 270

函数名	LEDPWM_SetDuty
函数原型	void LEDPWM_SetDuty(LEDPWM_Channel_Typedef LEDPWM_Channel, uint8_t LEDPWM_Duty)
功能描述	LEDPWM 占空比长度设置
输入参数 1	LEDPWM_Channel：LEDPWM 通道选择，取值可参考 Table 54。
输入参数 2	LEDPWM_Duty：LEDPWM 的占空比长度
返回值	无

使用示例:

```
/* 配置LEDPWM的占空比长度 */
LEDPWM_SetDuty(LEDPWM, LEDPWM_Channel_6,500);
```

LEDPWM_GetDuty

Table 271

函数名	LEDPWM_GetDuty
函数原型	uint8_t LEDPWM_GetDuty(LEDPWM_Channel_Typedef LEDPWM_Channel)
功能描述	获取 LEDPWM 的占空比长度值
输入参数 1	LEDPWM_Channel：LEDPWM 通道选择，取值可参考 Table 54。
返回值	LEDPWM 的占空比长度值

使用示例:

```
/* 获取LEDPWM06的占空比长度值 */
uint16_t  LEDPWM_Duty;
LEDPWM_Duty = LEDPWM_GetDuty(LEDPWM, LEDPWM_Channel_6);
```

LEDPWM_ITConfig

Table 272

函数名	LEDPWM_ITConfig
函数原型	void LEDPWM_ITConfig(uint16_t LEDPWM_IT, FunctionalState NewState)
功能描述	LEDPWM 中断配置函数
输入参数 1	LEDPWMx：x 可以是 0 或 1，来选择 LEDPWM 外设
输入参数 2	LEDPWM_IT：LEDPWM 中断请求，取值可参考 Table 273。
输入参数 3	NewState：LEDPWM 中断开启/关闭
返回值	无

LEDPWM_IT

LEDPWM_IT 用来使能或者失能 LEDPWM 中断。参阅Table 252获得这个参数的所有成员。。

Table 273

LEDPWM_IT	描述
LEDPWM_IT_INTEN	LEDPWM 中断请求

使用示例:

```
/ *开启LEDPWM的中断*/
LEDPWM_ITConfig(LEDPWM, LEDPWM_IT_INTEN, ENABLE);
```

LEDPWM_GetFlagStatus

Table 274

函数名	LEDPWM_GetFlagStatus
函数原型	FlagStatus LEDPWM_GetFlagStatus(uint16_t LEDPWM_FLAG)
功能描述	获取 LEDPWM 中断标志位
输入参数 1	LEDPWM_FLAG：LEDPWM 中断标志，取值可参考 Table 275。
返回值	LEDPWMx 的新状态（SET 或者 RESET）

LEDPWM_FLAG

Table 254给出了LEDPWM_FLAG的值。

Table 275

LEDPWM_FLAG	描述
LEDPWM_Flag_LEDPWMIF	LEDPWM 中断请求标志

使用示例:

```
/* 获取LEDPWM中断请求标志位 */
FlagStatus  LEDPWM_Status;
LEDPWM_Status  = LEDPWM_GetFlagStatus(LEDPWM, LEDPWM_Flag_LEDPWMIF);
```

LEDPWM_ClearFlag

Table 276

函数名	LEDPWM_ClearFlag
函数原型	void LEDPWM_ClearFlag(uint16_t LEDPWM_FLAG)
功能描述	清除 LEDPWM 中断标志位
输入参数 1	LEDPWM_FLAG：LEDPWM 中断标志，取值可参考 Table 275。
返回值	无

使用示例:

```
/* 清除LEDPWM中断请求标志位 */
LEDPWM_ ClearFlag(LEDPWM, LEDPWM_Flag_LEDPWMIF);
```

综合使用示例:（LEDPWM 输出占空比为 50 的波形）

```
LED_InitTypeDef LED_InitStruct; //定义LED初始化结构体变量

/* 使能PWM0时钟 */
RCC_APB2Cmd(ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_LEDPWM, ENABLE);
/* 结构体成员初始化 */
LED_StructInit(&LED_InitStruct);  /* 设置PWM配置结构体 */
LEDPWM_InitStruct.LEDPWM_AlignedMode = PWM_AlignmentMode_Edge;        //LEDPWM对齐模式为边沿对齐
LEDPWM_InitStruct.LEDPWM_Cycle = (uint16_t)(32000000 / 256 / 1000);  //LEDPWM的频率为1000Hz
LEDPWM_InitStruct.LEDPWM_OutputChannel = LEDPWM_Channel_0;    //使能LEDPWM输出
LEDPWM_InitStruct.LEDPWM_Prescaler = PWM_PRESCALER_DIV256;  //LEDPWM时钟分频为1分配
LEDPWM_Init(&LEDPWM_InitStruct);       //配置LEDPWM

LEDPWM_SetDuty(LEDPWM_Channel_0, LEDPWM_InitStruct.LEDPWM_Cycle * 0.5);      //配置占空比为50%
```

16 RCC 固件库函数

RCC 寄存器结构

RCC 寄存器结构，RCC_TypeDef，在文件“sc32f1xxx.h”中定义如下：

```
typedef struct
{
__IO uint32_t RCC_KEY;
__IO uint32_t RESERVED0;
__IO uint32_t RCC_CFG0;
__IO uint32_t RCC_CFG1;
__IO uint32_t RCC_STS;
__IO uint32_t RESERVED1;
__IO uint32_t SYST_CALIB;
__IO uint32_t NMI_CFG;
} RCC_TypeDef;
```

RCC 寄存器表格

Table 277

寄存器	描述
RCC_KEY	RCC 保护寄存器
RCC_CFG0	系统时钟源选择寄存器
RCC_CFG1	外设时钟源选择寄存器
RCC_STS	时钟状态寄存器
SYST_CALIB	Systick 校准参数寄存器
NMI_CFG	非可屏蔽中断(NMI中断)配置寄存器

RCC 固件库函数列

Table 278

函数名	描述
RCC_DeInit	将外设 RCCx 寄存器重设为缺省值
RCC_Unlock	解锁 RCC 寄存器操作功能
RCC_HXTCmd	使能或者失能 HXT
RCC_LXTCmd	使能或者失能 LXT
RCC_HIRCCmd	使能或者失能 HIRC
RCC_LIRCCmd	使能或者失能 LIRC
RCC_PLLCmd	使能或者失能 PLL
RCC_PLLRCmd	使能或者失能 PLLR
RCC_HIRCDIV1Cmd	使能或者失能 HIRCDIV1Cmd
RCC_APB0Cmd	使能或者失能 APB0
RCC_APB1Cmd	使能或者失能 APB1
RCC_APB2Cmd	使能或者失能 APB2
RCC_SYSCLKConfig	系统时钟配置
RCC_GetSYSCLKSource	获取系统时钟源
RCC_SystickCLKConfig	系统滴答时钟（SystickCLK）时钟源配置
RCC_SystickSetCounter	系统滴答时钟计数值设置
RCC_SystickCmd	系统滴答时钟使能
RCC_SystickGetFlagStatus	系统滴答时钟标志位获取
RCC_HCLKConfig	设置 HCLK 时钟（HCLK）
RCC_APB0Config	设置 APB0 时钟（APB0）
RCC_APB1Config	设置 APB1 时钟（APB1）

RCC_APB2Config	设置 APB2 时钟（APB2）
RCC_GetClocksFreq	获取各个时钟总线的频率
RCC_WaitConfig	配置 RCC wait 数量
RCC_PWM0CLKConfig	配置 PWM0 时钟源
RCC_LCDCLKConfig	设置 LCD 时钟（LCDCLK）
RCC_BTMCLKConfig	设置 BTM 时钟（BTMCLK）
RCC_AHBPeriphClockCmd	使能或者失能 AHB 外设时钟
RCC_APB2PeriphClockCmd	使能或者失能 APB2 外设时钟
RCC_APB1PeriphClockCmd	使能或者失能 APB1 外设时钟
RCC_APB0PeriphClockCmd	使能或者失能 APB0 外设时钟
RCC_AHBPeriphResetCmd	强制或者释放高速 AHB 外设复位
RCC_APB2PeriphResetCmd	强制或者释放高速 APB（APB2）外设复位
RCC_APB1PeriphResetCmd	强制或者释放低速 APB（APB1）外设复位
RCC_APB0PeriphResetCmd	强制或者释放低速 APB（APB0）外设复位
RCC_NMICmd	配置部分外设的 NMI 中断
RCC_ITConfig	使能或失能 RCC 中断
RCC_GetFlagStatus	获取 RCC 的标志位

RCC 固件库函数详解

RCC_DeInit

Table 279

函数名	RCC_DeInit
函数原型	void RCC_DeInit(void)
功能描述	将外设 RCCx 寄存器重设为缺省值
输入参数	无
返回值	无

使用示例:

```
/* 将外设 RCC 寄存器重设为缺省值*/
RCC_DeInit();
```

RCC_Unlock

Table 280

函数名	RCC_Unlock
函数原形	ErrorStatus RCC_Unlock(uint8_t TimeLimit)
功能描述	解锁 RCC 寄存器操作功能
输入参数	TimeLimit：操作使能开关及时限设置，需要大于等于 0x40
返回值	RCC 解锁结果(SUCCESS 或 ERROR)

使用示例:

```
/* 解锁RCC寄存器操作功能*/
ErrorStatus  RCC_UnlockStatus;
RCC_UnlockStatus = RCC_Unlock(0x50); //大于等于0x40，返回SUCCESS
```

RCC_HXTCmd（SC32F10xx、SC32F11xx、SC32F12xx 系列）

Table 281

函数名	RCC_HXTCmd
函数原形	void RCC_HXTCmd(FunctionalState NewState)
功能描述	使能或者失能 HXT
输入参数	NewState: HXT 时钟的新状态 这个参数可以取：ENABLE 或者 DISABLE

返回值	无

使用示例:

```
/* 使能HXT */
RCC_HXTCmd(ENABLE);
```

RCC_LXTCmd

Table 282

函数名	RCC_LXTCmd
函数原形	void RCC_LXTCmd(FunctionalState NewState)
功能描述	使能或者失能 LXT
输入参数	NewState: LXT 时钟的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* 使能LXT */
RCC_LXTCmd(ENABLE);
```

RCC_HIRCCmd

Table 283

函数名	RCC_HIRCCmd
函数原形	void RCC_HIRCCmd(FunctionalState NewState)
功能描述	使能或者失能 HIRC
输入参数	NewState: HIRC 时钟的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/*使能HIRC */
RCC_Unlock(0x50);
RCC_HIRCCmd(ENABLE);
```

RCC_LIRCCmd

Table 284

函数名	RCC_LIRCCmd
函数原形	void RCC_LIRCCmd(FunctionalState NewState)
功能描述	使能或者失能 LIRC
输入参数	NewState: LIRC 时钟的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/*使能LIRC */
RCC_Unlock(0x50);
RCC_LIRCCmd(ENABLE);
```

RCC_PLLCmd

Table 285

函数名	RCC_PLLCmd
函数原形	void RCC_PLLCmd (FunctionalState NewState)
功能描述	使能或者失能 PLL
输入参数	NewState: PLL 时钟的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/*使能PLL */
```

RCC_Unlock(0x50);
RCC_PLLCmd (ENABLE);

RCC_PLLRCmd

Table 286

函数名	RCC_PLLRCmd
函数原形	void RCC_PLLRCmd (FunctionalState NewState)
功能描述	使能或者失能 PLLR
输入参数	NewState: PLL 时钟的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/*使能PLLR */
RCC_Unlock(0x50);
RCC_PLLRCmd (ENABLE);
```

RCC_HIRCDIV1Cmd（SC32F15xx、SC32F11xx、SC32F12xx 系列）

Table 287

函数名	RCC_HIRCDIV1Cmd
函数原形	void RCC_HIRCDIV1Cmd(FunctionalState NewState)
功能描述	使能或者失能 HIRCDIV1Cmd
输入参数	NewState: HIRCDIV1 的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/*使能RCC_HIRCDIV1Cmd */
RCC_Unlock(0x50);
RCC_ RCC_HIRCDIV1Cmd (ENABLE);
```

RCC_APB0Cmd

Table 288

函数名	RCC_APB0Cmd
函数原形	void RCC_APB0Cmd(FunctionalState NewState)
功能描述	使能或者失能 APB0
输入参数	NewState: APB0 时钟的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/*使能APB0*/
RCC_Unlock(0x50);
RCC_APB0Cmd(ENABLE);
```

RCC_APB1Cmd

Table 289

函数名	RCC_APB1Cmd
函数原形	void RCC_APB1Cmd(FunctionalState NewState)
功能描述	使能或者失能 APB1
输入参数	NewState: APB1 时钟的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/*使能APB1 */
RCC_APB1Cmd(ENABLE);
```

RCC_APB2Cmd

Table 290

函数名	RCC_APB2Cmd
函数原形	void RCC_APB2Cmd(FunctionalState NewState)
功能描述	使能或者失能 APB2
输入参数	NewState: APB2 时钟的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/*使能APB2 */
RCC_APB2Cmd(ENABLE);
```

RCC_SYSCLKConfig

Table 291

函数名	RCC_ SYSCLKConfig
函数原形	ErrorStatus RCC_SYSCLKConfig(RCC_SYSCLKSource_TypeDef RCC_SYSCLKSource)
功能描述	系统时钟配置
输入参数	RCC_SYSCLKSource: 系统时钟（HCLK 分频之前）源
返回值	SUCCESS 或 ERROR

RCC_SYSCLKSource

RCC_SYSCLKSource设置系统时钟源，参阅Table 292获得这个参数的所有成员。

Table 292

RCC_SYSCLKSource	描述
RCC_SYSCLKSource_HIRC	系统时钟源为 HIRC
RCC_SYSCLKSource_LIRC	系统时钟源为 LIRC
RCC_SYSCLKSource_LXT	系统时钟源为 LXT
RCC_SYSCLKSource_HXT	系统时钟源为 HXT

SC32f10xx、SC32f11xx、SC32f12xx可选(RCC_SYSCLKSource_HIRC - RCC_SYSCLKSource_HXT)
SC32f15xx可选(RCC_SYSCLKSource_HIRC - RCC_SYSCLKSource_LXT)

使用示例:

```
/* 设置系统时钟源为HIRC */
ErrorStatus RCC_SYSCLK；
RCC_SYSCLK = RCC_SYSCLKConfig(RCC_SYSCLKSource_HIRC);
```

RCC_GetSYSCLKSource

Table 293

函数名	RCC_GetSYSCLKSource
函数原形	RCC_SYSCLKSource_TypeDef RCC_GetSYSCLKSource(void)
功能描述	获取系统时钟
输入参数	无
返回值	系统时钟源

使用示例:

```
/* 获取系统时钟源*/
RCC_SYSCLKSource_TypeDef RCC_SYSCLKSource;
RCC_SYSCLKSource = RCC_GetSYSCLKSource();
```

RCC_SystickCLKConfig

Table 294

函数名	RCC_SystickCLKConfig
函数原形	void RCC_SystickCLKConfig(RCC_SysTickSource_TypeDef RCC_SysTickSource)

功能描述	SYSTICK 时钟配置（SYSTICK）
输入参数	RCC_ SysTickSource: SYSTICK 时钟源
返回值	无

RCC_ SysTickSource

RCC_ SysTickSource设置滴答定时器的时钟源，参阅Table 295获得这个参数的所有成员。

Table 295

RCC_ SysTickSource	描述
RCC_SysTickSource_HCLK_DIV8	SYSTICK 时钟源为 HCLK/8
RCC_SysTickSource_HIRC_DIV2	SYSTICK 时钟源为 HIRC/2
RCC_SysTickSource_HXT_DIV2	SYSTICK 时钟源为 HXT/2
RCC_SysTickSource_LIRC	SYSTICK 时钟源为 LIRC
RCC_SysTickSource_LXT	SYSTICK 时钟源为 LXT
RCC_SysTickSource_HCLK	SYSTICK 时钟源为 HCLK

使用示例:

```
/* 设置SYSTICK时钟源为HCLK 8分频 */
RCC_Unlock(0x50);
RCC_SysTickSource(RCC_SysTickSource_HCLK_DIV8);
```

RCC_SystickSetCounter

Table 296

函数名	RCC_SystickSetCounter
函数原形	void RCC_SystickSetCounter(uint32_t Counter)
功能描述	系统滴答时钟计数值设置
输入参数	Counter: SYSTICK 时钟计数值
返回值	无

使用示例:

```
/* 设置SYSTICK时钟计数值*/
RCC_Unlock(0x50);
RCC_SystickSetCounter(160000);
```

RCC_SystickCmd

Table 297

函数名	RCC_SystickCmd
函数原形	void RCC_SystickCmd(FunctionalState NewState)
功能描述	SYSTICK 时钟使能
输入参数	NewState: SYSTICK 时钟的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* 使能SYSTICK时钟*/
RCC_Unlock(0x50);
RCC_SystickCmd(ENABLE);
```

RCC_SystickGetFlagStatus

Table 298

函数名	RCC_SystickGetFlagStatus
函数原形	FlagStatus RCC_SystickGetFlagStatus(void)
功能描述	SYSTICK 时钟标志获取
输入参数	无
返回值	SYSTICK 时钟标志状态

使用示例:

```
/* 获取SYSTICK时钟标志*/
FlagStatus SystickFlag;
RCC_Unlock(0x50);
SystickFlag = RCC_SystickGetFlagStatus();
```

RCC_HCLKConfig

Table 299

函数名	RCC_HCLKConfig
函数原形	void RCC_HCLKConfig(RCC_HCLK_TypeDef RCC_HCLKCLKSource)
功能描述	HCLK 时钟配置
输入参数	RCC_HCLKCLKSource: 系统时钟分频选择
返回值	无

RCC_HCLK

RCC_HCLK设置系统分频，参阅Table 300获得这个参数的所有成员。

Table 300

RCC_ HCLK	描述
RCC_SYSCLK_Div1	HCLK = SYSCLK/1
RCC_SYSCLK_Div2	HCLK = SYSCLK/2
RCC_SYSCLK_Div4	HCLK = SYSCLK/4
RCC_SYSCLK_Div8	HCLK = SYSCLK/8
RCC_SYSCLK_Div16	HCLK = SYSCLK/16

使用示例:

```
/* 设置HCLK为SYSCLK 1分频*/
RCC_HCLKConfig(RCC_SYSCLK_Div1);
```

RCC_APB0Config

Table 301

函数名	RCC_APB0Config
函数原形	void RCC_APB0Config(RCC_PCLK_TypeDef RCC_APB0CLKSource)
功能描述	APB0 时钟配置
输入参数	RCC_APB0CLK: HCLK 时钟分频选择
返回值	无

RCC_PCLK_TypeDef

RCC_PCLK_TypeDef设置系统分频，参阅Table 302获得这个参数的所有成员。

Table 302

RCC_PCLK_TypeDef	描述
RCC_HCLK_Div1	APB0CLK= HCLK /1
RCC_HCLK_Div2	APB0CLK = HCLK/2
RCC_HCLK_Div4	APB0CLK = HCLK/4
RCC_HCLK_Div8	APB0CLK = HCLK/8
RCC_HCLK_Div16	APB0CLK = HCLK/16
RCC_HCLK_Div32	APB0CLK = HCLK/32
RCC_HCLK_Div64	APB0CLK = HCLK/64
RCC_HCLK_Div128	APB0CLK = HCLK/128

使用示例:

```
/* 设置APB0时钟源为HCLK 1分频*/
RCC_APB0Config(RCC_HCLK_Div1);
```

RCC_APB1Config

Table 303

函数名	RCC_APB1Config

函数原形	void RCC_APB1Config(RCC_PCLK_TypeDef RCC_APB1CLKSource)
功能描述	APB1 时钟配置
输入参数	RCC_APB1CLK: HCLK 时钟分频选择
返回值	无

使用示例:

```
/* 设置APB1时钟源为HCLK 1分频*/
RCC_APB1Config(RCC_HCLK_Div1);
```

RCC_APB2Config

Table 304

函数名	RCC_APB2Config
函数原形	void RCC_APB2Config(RCC_PCLK_TypeDef RCC_APB2CLKSource)
功能描述	APB2 时钟配置
输入参数	RCC_APB2CLK: HCLK 时钟分频选择
返回值	无

使用示例:

```
/* 设置APB2时钟源为HCLK 1分频*/
RCC_APB2Config(RCC_HCLK_Div1);
```

RCC_GetClocksFreq

Table 305

函数名	RCC_GetClocksFreq
函数原形	void RCC_GetClocksFreq(RCC_ClocksTypeDef* RCC_Clocks)
功能描述	获取各个时钟总线的频率
输入参数	RCC_APB2CLK: HCLK 时钟分频选择
返回值	无

RCC_ClocksTypeDef

RCC_ClocksTypeDef 定义于文件“sc32f1xxx_rcc.h”：

```
typedef struct
{
  uint32_t SYSCLK_Frequency;
  uint32_t HCLK_Frequency;
  uint32_t PCLK0_Frequency;
  uint32_t PCLK1_Frequency;
  uint32_t PCLK2_Frequency;
} RCC_ClocksTypeDef;
```

使用示例:

```
/* 设置APB2时钟源为HCLK 1分频*/
RCC_ClocksTypeDef RCC_Clocks;
RCC_GetClocksFreq(&RCC_Clocks);
```

RCC_WaitConfig

Table 306

函数名	RCC_WaitConfig
函数原形	void RCC_WaitConfig(RCC_Wait_TypeDef RCC_Wait)
功能描述	设置 RCC wait 个数
输入参数	RCC_Wait：wait 个数选择
返回值	无

RCC_Wait

RCC_Wait时钟wait个数选择，参阅 Table 307获得这个参数的所有成员。

Table 307

RCC_Wait	描述
RCC_WAIT_0	wait 个数 0

RCC_WAIT_1	wait 个数 1
RCC_WAIT_2	wait 个数 2
RCC_WAIT_3	wait 个数 3

使用示例:

```
/* 时钟wait个数0*/
RCC_Unlock(0x50);
RCC_WaitConfig (RCC_WAIT_0);
```

RCC_PWM0CLKConfig

Table 308

函数名	RCC_PWM0CLKConfig
函数原形	void RCC_PWM0CLKConfig(RCC_PWM0CLKSource_TypeDef RCC_PWM0CLKSource)
功能描述	配置 PWM0 时钟源
输入参数	RCC_PWM0CLKSource: PWM0 时钟源选择
返回值	无

使用示例:

```
/* 设置PWM0时钟源为PCLK*/
RCC_Unlock(0x50);
RCC_PWM0CLKConfig(RCC_PWM0CLKSource_PCLK);
```

RCC_LCDCLEDLKConfig

Table 309

函数名	RCC_LCDLEDCLKConfig
函数原形	void RCC_LCDLEDCLKConfig(RCC_LCDLEDCLKSource_TypeDef RCC_LCDLEDCLKSource)
功能描述	设置 LCD/LED 时钟配置（LCD/LED）
输入参数	LCDCLKSource: LCD/LED 时钟源
返回值	无

RCC_LCDLEDCLKSource

LCDCLKSource时钟源选择，参阅Table 310获得这个参数的所有成员。

Table 310

RCC_LCDLEDCLKSource	描述
RCC_LCDLEDCLKSource_LIRC	LCD 时钟源为 LIRC
RCC_LCDCLEDLKSource_LXT	LCD 时钟源为 LXT

使用示例:

```
/* 设置LCD时钟源为LIRC*/
RCC_Unlock(0x50);
RCC_LCDCLKConfig(RCC_LCDLEDCLKSource_LIRC);
```

RCC_BTMCLKConfig

Table 311

函数名	RCC_BTMCLKConfig
函数原形	void RCC_BTMCLKConfig(RCC_BTMCLKSource_TypeDef RCC_BTMCLKSource)
功能描述	设置 BTM 时钟配置（BTM）
输入参数	BTMCLKSource: BTM 时钟源
返回值	无

RCC_BTMCLKSource

BTMCLKSource时钟源选择，参阅Table 312获得这个参数的所有成员。

Table 312

RCC_BTMCLKSource	描述
RCC_BTMCLKSource_LIRC	BTM 时钟源为 LIRC

RCC_BTMCLKSource_LXT	BTM 时钟源为 LXT

使用示例:

```
/* 设置BTM时钟源为LIRC*/
RCC_Unlock(0x50);
RCC_BTMCLKConfig(RCC_BTMCLKSource_LIRC);
```

RCC_AHBPeriphClockCmd

Table 313

函数名	RCC_AHBPeriphClockCmd
函数原形	void RCC_AHBPeriphClockCmd(uint32_t RCC_AHBPeriph, FunctionalState NewState)
功能描述	使能或者失能 AHB 外设时钟
输入参数 1	RCC_AHBPeriph:指定 AHB 时钟总线上的外设
输入参数 2	NewState: AHB 时钟的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

RCC_AHBPeriph

AHBPeriph外设时钟选择，参阅Table 314获得这个参数的所有成员。

Table 314

RCC_AHBPeriph	描述
RCC_AHBPeriph_DMA	DMA 外设时钟
RCC_AHBPeriph_CRC	CRC 外设时钟
RCC_AHBPeriph_IFB	IFB 外设时钟
RCC_AHBPeriph_ALL	挂载在 AHB 上所有外设时钟
RCC_AHBPeriph_CAN	CAN 外设时钟

SC32f10xx、SC32f12xx可选(RCC_AHBPeriph_DMA - RCC_AHBPeriph_ALL)
SC32f15xx可选(RCC_AHBPeriph_DMA - RCC_AHBPeriph_CAN)

使用示例:

```
/* 使能AHB上的DMA外设 */
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA,ENABLE);
```

RCC_APB0PeriphClockCmd

Table 315

函数名	RCC_APB0PeriphClockCmd
函数原形	void RCC_APB0PeriphClockCmd(uint32_t RCC_APB0Periph, FunctionalState NewState)
功能描述	使能或者失能 APB0 外设时钟
输入参数 1	RCC_APB0Periph:指定 APB0 时钟总线上的外设
输入参数 2	NewState: APB0 时钟的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

RCC_APB0Periph

APB0Periph外设时钟选择，参阅Table 316获得这个参数的所有成员。

Table 316

RCC_APB0Periph	描述
RCC_APB0Periph_TIM0	TIM0 外设时钟
RCC_APB0Periph_TIM1	TIM1 外设时钟
RCC_APB0Periph_TIM2	TIM2 外设时钟
RCC_APB0Periph_TIM3	TIM3 外设时钟
RCC_APB0Periph_TWI0	TWI0 外设时钟
RCC_APB0Periph_SPI0	SPI0 外设时钟
RCC_APB0Periph_UART0	UART0 外设时钟
RCC_APB0Periph_UART1	UART1 外设时钟

RCC_APB0Periph_PWM0	PWM0 外设时钟
RCC_APB0Periph_UART5	UART5 外设时钟
RCC_APB0Periph_ALL	挂载在 APB0 上所有外设时钟

SC32f10xx可选(RCC_APB0Periph_TIM0 - RCC_APB0Periph_TIM3，RCC_APB0Periph_TWI0，RCC_APB0Periph_SPI0，RCC_APB0Periph_UART0，RCC_APB0Periph_UART1，RCC_APB0Periph_PWM0，RCC_APB0Periph_ALL)
SC32f11xx、SC32f12xx可选(RCC_APB0Periph_TIM0 - RCC_APB0Periph_TIM3，RCC_APB0Periph_TWI0，RCC_APB0Periph_SPI0，RCC_APB0Periph_UART0，RCC_APB0Periph_UART1，RCC_APB0Periph_UART5，RCC_APB0Periph_PWM0，RCC_APB0Periph_ALL)
SC32f15xx可选(RCC_APB0Periph_TIM0，RCC_APB0Periph_TIM1，RCC_APB0Periph_TWI0，RCC_APB0Periph_SPI0，RCC_APB0Periph_UART0，RCC_APB0Periph_UART1RCC_APB0Periph_ALL)

使用示例:

```
/* 使能APB0上的TIM0外设 */
RCC_APB0PeriphClockCmd(RCC_APB0Periph_TIM0,ENABLE);
```

RCC_APB1PeriphClockCmd

Table 317

函数名	RCC_APB1PeriphClockCmd
函数原形	void RCC_APB1PeriphClockCmd(uint32_t RCC_APB1Periph, FunctionalState NewState)
功能描述	使能或者失能 APB1 外设时钟
输入参数 1	RCC_APB1Periph:指定 APB1 时钟总线上的外设
输入参数 2	NewState: APB1 时钟的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

RCC_APB1Periph

APB1Periph外设时钟选择，参阅Table 318获得这个参数的所有成员。

Table 318

RCC_APB1Periph	描述
RCC_APB1Periph_TIM2	TIM2 外设时钟
RCC_APB1Periph_TIM3	TIM3 外设时钟
RCC_APB1Periph_SPI1_TWI1	SPI1_TWI1 外设时钟
RCC_APB1Periph_SPI1	SPI1 外设时钟
RCC_APB1Periph_TIM4	TIM4 外设时钟
RCC_APB1Periph_TIM5	TIM5 外设时钟
RCC_APB1Periph_TIM6	TIM6 外设时钟
RCC_APB1Periph_TIM7	TIM7 外设时钟
RCC_APB1Periph_TWI1	TWI1 外设时钟
RCC_APB1Periph_UART2	UART2 外设时钟
RCC_APB1Periph_UART4	UART4 外设时钟
RCC_APB1Periph_ALL	挂载在 APB1 上所有外设时钟

SC32f10xx可选(RCC_APB1Periph_TIM4 - RCC_APB1Periph_TIM7，RCC_APB1Periph_TWI1，RCC_APB1Periph_UART2，RCC_APB1Periph_ALL)
SC32f12xx可选(RCC_APB1Periph_TIM4 - RCC_APB1Periph_TIM7，RCC_APB1Periph_TWI1，RCC_APB1Periph_SPI1，RCC_APB1Periph_UART4，RCC_APB1Periph_UART2，RCC_APB1Periph_ALL)
SC32f15xx可选(RCC_APB1Periph_TIM2，RCC_APB1Periph_TIM3 ，RCC_APB1Periph_SPI1_TWI1 RCC_APB1Periph_UART2 ，RCC_APB0Periph_ALL)

使用示例:

```
/* 使能APB1上的TIM4外设 */
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4,ENABLE);
```

RCC_APB2PeriphClockCmd

Table 319

函数名	RCC_APB2PeriphClockCmd
函数原形	void RCC_APB2PeriphClockCmd(uint32_t RCC_APB2Periph, FunctionalState NewState)
功能描述	使能或者失能 APB2 外设时钟
输入参数 1	RCC_APB2Periph:指定 APB2 时钟总线上的外设
输入参数 2	NewState: APB2 时钟的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

RCC_APB2Periph

APB2Periph外设时钟选择，参阅Table 320获得这个参数的所有成员。

Table 320

RCC_APB2Periph	描述
RCC_APB2Periph_LEDPWM	LEDPWM 外设时钟
RCC_APB2Periph_LCD_LED	LCD_LED 外设时钟
RCC_APB2Periph_UART3	UART3 外设时钟
RCC_APB2Periph_QEP0	QEP0 外设时钟
RCC_APB2Periph_QEP1	QEP1 外设时钟
RCC_APB2Periph_ADC	ADC 外设时钟
RCC_APB2Periph_ALL	挂载在 APB2 上所有外设时钟

SC32f10xx、SC32f11xx、SC32f12xx可选(RCC_APB2Periph_LEDPWM，RCC_APB2Periph_LCD_LED，RCC_APB2Periph_UART3，RCC_APB2Periph_ALL)
SC32f15xx可选(RCC_APB2Periph_QEP0，RCC_APB2Periph_QEP1 ，RCC_APB2Periph_ADC，RCC_APB0Periph_ALL)

使用示例:

```
/* 使能APB2上的LEDPWM外设 */
RCC_APB2PeriphClockCmd(RCC_APB2Periph_LEDPWM,ENABLE);
```

RCC_AHBPeriphResetCmd

Table 321

函数名	RCC_AHBPeriphResetCmd
函数原形	void RCC_AHBPeriphResetCmd(uint32_t RCC_AHBPeriph, FunctionalState NewState)
功能描述	强制或者释放高速 AHB 外设时钟
输入参数 1	RCC_AHBPeriph:指定 AHB 时钟总线上的外设
输入参数 2	NewState: AHB 外设复位新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* 复位AHB上的DMA外设 */
RCC_AHBPeriphResetCmd(RCC_AHBPeriph_DMA,ENABLE);
```

RCC_APB0PeriphResetCmd

Table 322

函数名	RCC_APB0PeriphResetCmd
函数原形	void RCC_APB0PeriphResetCmd(uint32_t RCC_APB0Periph, FunctionalState NewState)
功能描述	强制或者释放高速 APB0 外设时钟
输入参数 1	RCC_APB0Periph:指定 APB0 时钟总线上的外设
输入参数 2	NewState: APB0 外设复位新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* 复位APB0上的TIM0外设 */
```

RCC_APB0PeriphResetCmd(RCC_APB0Periph_TIM0,ENABLE);

RCC_APB1PeriphResetCmd

Table 323

函数名	RCC_APB1PeriphResetCmd
函数原形	void RCC_APB1PeriphResetCmd(uint32_t RCC_APB1Periph, FunctionalState NewState)
功能描述	强制或者释放高速 APB1 外设时钟
输入参数 1	RCC_APB1Periph:指定 APB1 时钟总线上的外设
输入参数 2	NewState: APB1 外设复位新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* 复位APB1上的TIM4外设 */
RCC_APB1PeriphResetCmd(RCC_APB1Periph_TIM4,ENABLE);
```

RCC_APB2PeriphResetCmd

Table 324

函数名	RCC_APB2PeriphResetCmd
函数原形	void RCC_APB2PeriphResetCmd(uint32_t RCC_APB2Periph, FunctionalState NewState)
功能描述	强制或者释放高速 APB2 外设时钟
输入参数 1	RCC_APB2Periph:指定 APB2 时钟总线上的外设
输入参数 2	NewState: APB2 外设复位新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* 复位APB2上的LEDPWM外设 */
RCC_APB2PeriphResetCmd(RCC_APB2Periph_LEDPWM,ENABLE);
```

RCC_NMICmd

Table 325

函数名	RCC_NMICmd
函数原形	void RCC_NMICmd(uint32_t RCC_NMIPeriph, FunctionalState NewState)
功能描述	使能 NMI 内外设中断
输入参数 1	RCC_NMIPeriph:指定 NMI 可控制时钟总线上的外设
输入参数 2	NewState: NMI 外设复位新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

RCC_NMIPeriph

RCC_NMIPeriph设置NMI可控制时钟总线上的外设，参阅Table 326获得这个参数的所有成员。

Table 326

RCC_NMIPeriph	描述
RCC_NMIPeriph_CSS	CSS 非屏蔽中断
RCC_NMIPeriph_CMP	CMP 非屏蔽中断
RCC_NMIPeriph_INT0	外部中断 INT0 非屏蔽中断
RCC_NMIPeriph_SRAMPE	SRAM 奇偶校验错误中断
RCC_NMIPeriph_OP2	OP2_CMP 非屏蔽中断
RCC_NMIPeriph_OP1	OP1_CMP 非屏蔽中断
RCC_NMIPeriph_CMP	CMP 非屏蔽中断
RCC_NMIPeriph_MCMP0	MCMP0 非屏蔽中断
RCC_NMIPeriph_ALL	开启所有外设 NMI 中断

SC32f10xx、SC32f11xx、SC32f12xx可选(RCC_NMIPeriph_CSS，RCC_NMIPeriph_CMP，RCC_NMIPeriph_INT0，RCC_NMIPeriph_ALL)

SC32f15xx可选(RCC_NMIPeriph_INT0- RCC_NMIPeriph_ALL)

使用示例:

```
/* 使能NMI上的INT0外设中断 */
RCC_NMICmd(RCC_NMIPeriph_INT0, ENABLE);
```

RCC_ITConfig

Table 327

函数名	RCC_ITConfig
函数原形	void RCC_ITConfig(FunctionalState NewState)
功能描述	使能或者失能指定的 RCC 中断
输入参数	NewState：RCCx 中断的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* 使能RCC 外设总中断 */
RCC_ITConfig (ENABLE);
```

RCC_GetFlagStatus

Table 328

函数名	RCC_GetFlagStatus
函数原形	FlagStatus RCC_GetFlagStatus(uint32_t RCC_FLAG)
功能描述	检查指定的 RCC 标志位设置与否
输入参数	RCC_FLAG：待检查的 RCC 标志位
返回值	RCC_FLAG 的新状态

RCC_FLAG

RCC_FLAG设置系统分频，参阅Table 329获得这个参数的所有成员。

Table 329

RCC_FLAG	描述
RCC_FLAG_CLKIF	时钟源异常标志位

使用示例:

```
/* 获得RCC 时钟源异常标志位状态 */
RCC_GetFlagStatus(RCC_FLAG_CLKIF);
```

17 SPI 固件库函数

SC32F1xxx 系列有 SPI0和SPI1。SPI0具有16位8级FIFO缓存，发送接收独立，有FIFO数据传输一半中断以及对应标志位，方便客户及时读取写入数据。SPI1无FIFO缓存。

SPI 寄存器结构

SPI 寄存器结构，SPI_TypeDef，在文件“sc32f1xxx.h”中定义如下：

```
typedef struct
{
__IO uint32_t SPI_CON;
__IO uint32_t SPI_STS;
__IO uint32_t RESERVED;
__IO uint32_t SPI_DATA;
__IO uint32_t SPI_IDE;
} SPI_TypeDef;
```

SPI 寄存器表格

Table 330

寄存器	描述
SPI_CON	SPI 控制寄存器
SPI_STS	SPI 标志位寄存器
SPI_DATA	SPI 数据寄存器
SPI_IDE	通信口 DMA 控制寄存器

SPI 固件库函数列

Table 331

函数名	描述
SPI_DeInit	将外设 SPIx 寄存器重设为缺省值
SPI_Init	根据 SPI_InitStruct 中指定的参数初始化外设 SPIx 寄存器
SPI_StructInit	把 SPI_InitStruct 中的每一个参数按缺省值填入
SPI_Cmd	使能或者失能 SPI 外设
SPI_ITConfig	使能或者失能指定的 SPI 中断
SPI_DMACmd	使能或者失能指定 SPI 的 DMA 请求
SPI_SetMode	设置 SPI 的工作模式
SPI_DataSizeConfig	设置 SPI 的数据宽度
SPI_SendData	通过外设 SPIx 发送一个数据
SPI_ReceiveData	返回通过 SPIx 最近接收的数据
SPI_SendDataFIFO	SPIx 通过 FIFO 发送数据
SPI_ReceiveDataFIFO	SPIx 通过 FIFO 接收数据
SPI_GetFlagStatus	检查指定的 SPI 标志位设置与否
SPI_ClearFlag	清除指定的 SPI 标志位
SPI_PinRemapConfig	设置 SPI 的重映射管脚

SPI 固件库函数详解

SPI_DeInit

Table 332

函数名	SPI_DeInit
函数原型	void SPI_DeInit(SPI_TypeDef* SPIx)

功能描述	将外设 SPIx 寄存器重设为缺省值
输入参数	SPIx：选择 SPI 外设（x 可以是 0,1,2,具体需以规格书为准）
返回值	无

使用示例:

```
/* 将外设 SPI0 寄存器重设为缺省值*/
SPI_DeInit(SPI0);
```

SPI_Init

Table 333

函数名	SPI_Init
函数原形	void SPI_Init(SPI_TypeDef* SPIx, SPI_InitTypeDef* SPI_InitStruct)
功能描述	根据 SPI_InitStruct 中指定的参数初始化外设 SPIx 寄存器
输入参数 1	SPIx：选择 SPI 外设（x 可以是 0,1,2,具体需以规格书为准）
输入参数 2	SPI_InitStruct：指向结构 SPI_InitTypeDef 的指针，包含了外设 GPIO 的配置信息 参阅 Section：SPI_InitTypeDef 查阅更多该参数允许取值范围
返回值	无

SPI_InitStruct

SPI_InitTypeDef 定义于文件“sc32f1xxx_spi.h”：

```
typedef struct
{
uint16_t SPI_Mode;
uint16_t SPI_DataSize
uint16_t SPI_CPHA;
uint16_t SPI_CPOL;
uint32_t SPI_FirstBit;
uint32_t SPI_Prescaler;
} SPI_InitTypeDef ;
```

SPI_Mode

SPI工作模式设定，参阅Table 334获得这个参数的所有成员。

Table 334

SPI_Mode	描述
SPI_Mode_Slave	SPI 作为从机
SPI_Mode_Master	SPI 作为主机

SPI_DataSize

SPI传输宽度设定，参阅Table 335获得这个参数的所有成员。

Table 335

SPI_DataSize	描述
SPI_DataSize_8B	SPI 传输宽度为 8 位
SPI_DataSize_16B	SPI 传输宽度为 16 位

SPI_CPHA

SPI时钟相位设定，参阅Table 336获得这个参数的所有成员。

Table 336

SPI_CPHA	描述
SPI_CPHA_1Edge	SCK 周期的第一沿采集数据
SPI_CPHA_2Edge	SCK 周期的第二沿采集数据

SPI_CPOL

SPI时钟极性设定，参阅Table 337获得这个参数的所有成员。

Table 337

SPI_CPOL	描述
SPI_CPOL_Low	SCK 在空闲状态下为低电平
SPI_CPOL_High	SCK 在空闲状态下为高电平

SPI_FirstBit

SPI传送方向设定，参阅Table 338获得这个参数的所有成员。

Table 338

SPI_FirstBit	描述
SPI_FirstBit_MSB	MSB 优先发送
SPI_FirstBit_LSB	LSB 优先发送

SPI_Prescaler

SPI时钟预分频，参阅Table 339获得这个参数的所有成员。

Table 339

SPI_Prescaler	描述
SPI_Prescaler_1	时钟预分频值为 1
SPI_Prescaler_2	时钟预分频值为 2
SPI_Prescaler_4	时钟预分频值为 4
SPI_Prescaler_8	时钟预分频值为 8
SPI_Prescaler_16	时钟预分频值为 16
SPI_Prescaler_32	时钟预分频值为 32
SPI_Prescaler_64	时钟预分频值为 64
SPI_Prescaler_128	时钟预分频值为 128
SPI_Prescaler_256	时钟预分频值为 256
SPI_Prescaler_512	时钟预分频值为 512
SPI_Prescaler_1024	时钟预分频值为 1024
SPI_Prescaler_2048	时钟预分频值为 2048
SPI_Prescaler_4096	时钟预分频值为 4096

SC32f10xx、SC32f11xx、SC32f12xx可选(SPI_Prescaler_1- SPI_Prescaler_1024)
SC32f15xx可选(SPI_Prescaler_1- SPI_Prescaler_4096)

使用示例:

```
/* 根据 SPI_InitStruct 中指定的参数初始化 SPI0 */
SPI_InitTypeDef   SPI_InitStruct;
SPI_InitStruct.SPI_Mode = SPI_Mode_Master;  //主机模式
SPI_InitStruct.SPI_DataSize = SPI_DataSize_16B; //16位传输模式
SPI_InitStruct.SPI_FirstBit = SPI_FirstBit_LSB;  //LSB优先发送
SPI_InitStruct.SPI_CPHA = SPI_CPHA_1Edge;  //SCK周期的第一沿采集数据
SPI_InitStruct.SPI_CPOL = SPI_CPOL_Low;  //SCK在空闲状态下为低电平
SPI_InitStruct.SPI_Prescaler = SPI_Prescaler_2;  //时钟频率为Fpclk/2
SPI_Init(SPI0,&SPI_InitStruct);
```

SPI_StructInit

Table 340

函数名	SPI_StructInit
函数原形	void SPI_StructInit(SPI_InitTypeDef* SPI_InitStruct)
功能描述	把 SPI_InitStruct 中的每一个参数按缺省值填入
输入参数	SPI_InitStruct：指向结构 SPI_InitTypeDef 的指针，待初始化
返回值	无

使用示例:

```
/* 把 SPI_InitStruct 中的每一个参数按缺省值填入 */
SPI_InitTypeDef  SPI_InitStruct;
SPI_InitStruct = SPI_StructInit(&SPI_InitStruct);
```

SPI_Cmd

Table 341

函数名	SPI_Cmd
函数原形	void SPI_Cmd(SPI_TypeDef* SPIx, FunctionalState NewState)

功能描述	使能或者失能 SPI 外设
输入参数 1	SPIx：选择 SPI 外设（x 可以是 0,1,2,具体需以规格书为准）
输入参数 2	NewState: 外设 SPIx 的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* 使能SPI0 外设 */
SPI_Cmd(SPI0, ENABLE);
```

SPI_DMACmd

Table 342

函数名	SPI_DMACmd
函数原形	void SPI_DMACmd(SPI_TypeDef* SPIx, uint16_t SPI_DMAReq, FunctionalState NewState)
功能描述	使能或者失能指定 SPI 的 DMA 请求
输入参数 1	SPIx：选择 SPI 外设（x 可以是 0,1,2,具体需以规格书为准）
输入参数 2	SPI_DMAReq：来选择 SPIDMA 请求
输入参数 3	NewState: SPIx DMA 传输的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

SPI_DMAReq

SPI DMA请求选择，参阅Table 343获得这个参数的所有成员。

Table 343

SPI_DMAReq	描述
SPI_DMAReq_RX	DMA 接收功能
SPI_DMAReq_TX	DMA 发送功能

使用示例:

```
/* 使能SPI1发送的DMA请求 */
SPI_DMACmd (SPI1, SPI_DMAReq_TX, ENABLE);
```

SPI_ITConfig

Table 344

函数名	SPI_ITConfig
函数原形	void SPI_ITConfig(SPI_TypeDef* SPIx, uint16_t SPI_IT, FunctionalState NewState)
功能描述	使能或者失能指定的 SPI 中断
输入参数 1	SPIx：选择 SPI 外设（x 可以是 0,1,2,具体需以规格书为准）
输入参数 2	SPI_IT：待使能或者失能的 SPI 中断源
输入参数 3	NewState：SPIx 中断的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

SPI_IT

SPI IT请求选择，参阅Table 345获得这个参数的所有成员。

Table 345

SPI_IT	描述
SPI_IT_INTEN	中断请求 CPU 的屏蔽位
SPI_IT_RXNE	接收缓冲区非空中断使能
SPI_IT_TB	发送缓存器为空时的中断使能
SPI_IT_RX	接收 FIFO 溢出中断使能
SPI_IT_RXH	RXHIF 置起中断使能
SPI_IT_TXH	TXHIF 置起中断使能

使用示例:

```
/* 使能SPI0 外设总中断 */
SPI_ITConfig (SPI1, SPI_IT_INTEN，ENABLE);
```

SPI_SetMode

Table 346

函数名	SPI_SetMode
函数原形	void SPI_SetMode(SPI_TypeDef* SPIx, SPI_Mode_TypeDef SPI_Mode)
功能描述	设置 SPIx 的工作模式
输入参数 1	SPIx：选择 SPI 外设（x 可以是 0,1,2,具体需以规格书为准）
输入参数 2	SPI_Mode: 工作模式
返回值	无

使用示例:

```
/* 设置SPI1为主机 */
SPI_SetMode(SPI1,SPI_Mode_Maste);
```

SPI_DataSizeConfig

Table 347

函数名	SPI_DataSizeConfig
函数原形	void SPI_DataSizeConfig(SPI_TypeDef* SPIx, SPI_DataSize_TypeDef SPI_DataSize)
功能描述	设置 SPIx 的数据宽度
输入参数 1	SPIx：选择 SPI 外设（x 可以是 0,1,2,具体需以规格书为准）
输入参数 2	SPI_DataSize: 传输数据宽度
返回值	无

使用示例:

```
/* 设置SPI1数据传输宽度为8bit */
SPI_DataSizeConfig(SPI1,SPI_DataSize_8B);
```

SPI_SendData

Table 348

函数名	SPI_SendData
函数原形	void SPI_SendData(SPI_TypeDef* SPIx,uint16_t Data)
功能描述	通过外设 SPIx 发送一个数据
输入参数 1	SPIx：选择 SPI 外设（x 可以是 0,1,2,具体需以规格书为准）
输入参数 2	Data: 待发送的数据
返回值	无

使用示例:

```
/* SPI发送数据0x12 */
SPI_SendData(SPI1,0x12);
```

SPI_ReceiveData

Table 349

函数名	SPI_ReceiveData
函数原形	uint16_t SPI_ReceiveData(SPI_TypeDef* SPIx)
功能描述	返回通过 SPIx 最近接收的数据
输入参数	SPIx：选择 SPI 外设（x 可以是 0,1,2,具体需以规格书为准）
返回值	返回 SPIx 接收的数据

使用示例:

```
/* 把SPI接收的数据赋值给Data1 */
uint16_t Data1
Data1 = SPI_ReceiveData(SPI1);
```

SPI_ReceiveDataFIFO

Table 350

函数名	SPI_ReceiveDataFIFO
函数原形	void SPI_ReceiveDataFIFO(SPI_TypeDef* SPIx, uint16_t* Data, uint16_t length)
功能描述	返回通过 SPIx 最近接收的数据
输入参数 1	SPIx：选择 SPI 外设（x 可以是 0,1,2,具体需以规格书为准）
输入参数 2	* Data：读取数据的存放地址
输入参数 3	Length：读取数据的长度
返回值	无

使用示例:

```
/* 把SPI0接收的数据存放到DataArray1 */
Uint16_t DataArray1[8];
Data1 = SPI_ReceiveDataFIFO(SPI0, DataArray1,8);
```

SPI_SendDataFIFO

Table 351

函数名	SPI_SendDataFIFO
函数原形	void SPI_SendDataFIFO(SPI_TypeDef* SPIx, uint16_t* Data, uint16_t length)
功能描述	通过外设 SPIx 发送一个数据
输入参数 1	SPIx：选择 SPI 外设（x 可以是 0,1,2,具体需以规格书为准）
输入参数 2	Data: 待发送的数据
输入参数 3	length:发送数据长度
返回值	无

使用示例:

```
/* SPI0发送数据DataArray1 */
uint8_t DataArray1[8]= {1,2,3,4,5,6,7,8};
SPI_SendData(SPI0, DataArray1,8);
```

SPI_GetFlagStatus

Table 352

函数名	SPI_GetFlagStatus
函数原形	FlagStatus SPI_GetFlagStatus(SPI_TypeDef* SPIx, SPI_FLAG_TypeDef SPI_FLAG)
功能描述	检查指定的 SPI 标志位设置与否
输入参数 1	SPIx：选择 SPI 外设（x 可以是 0,1,2,具体需以规格书为准）
输入参数 2	SPI_FLAG：待检查的 SPI 标志位
返回值	SPI_FLAG 的新状态

SPI_FLAG

SPI 中断标志位选择，参阅Table 353获得这个参数的所有成员。

Table 353

SPI_FLAG	描述
SPI_Flag_SPIF	SPI 数据传送标志位
SPI_Flag_RINEIF	接收 FIFO 非空标志位
SPI_Flag_TXEIF	发送缓存为空标志位
SPI_Flag_RXFIF	接收 FIFO 已满状态位
SPI_Flag_RXHIF	接收 FIFO 内有效数据超过一半状态位/中断标志位
SPI_Flag_TXHIF	发送 FIFO 内有效数据不满一半状态位/中断标志位
SPI_Flag_WCOL	写入冲突标志位
SPI_FLAG_QTWIF	SPI1 数据传送标志位（SC32F15xx 系列 SPI1 中断标志位）

使用示例:

```
/* 获得SPI标志位状态 */
SPI_GetFlagStatus(SPI1, SPI_Flag_SPIF);
```

SPI_ClearFlag

Table 354

函数名	SPI_ClearFlag
函数原形	void SPI_ClearFlag(SPI_TypeDef* SPIx, uint32_t SPI_FLAG)
功能描述	清除 SPIx 的待处理标志位
输入参数 1	SPIx：选择 SPI 外设（x 可以是 0,1,2,具体需以规格书为准）
输入参数 2	SPI_FLAG：待清除的 SPI 标志位
返回值	无

使用示例:

```
/* 清除SPI1外设标志位 */
SPI_ClearFlag(SPI1, SPI_Flag_SPIF);
```

SPI_PinRemapConfig

Table 355

函数名	SPI_PinRemapConfig
函数原形	void SPI_PinRemapConfig(SPI_TypeDef* SPIx, SPI_PinRemap_TypeDef SPI_PinRemap)
功能描述	配置 SPIx 的重映射管脚
输入参数 1	SPIx：选择 SPI 外设（x 可以是 0,1,2,具体需以规格书为准）
输入参数 2	SPI_PinRemap：SPI 重映射管脚
返回值	无

SPI_Remap

SPI 映射管脚选择，参阅Table 356获得这个参数的所有成员。

Table 356

SPI_PinRemap	描述
SPI_PinRemap_Default	默认 SPI 管脚映射
SPI_PinRemap_A	SPI 映射管脚组 A
SPI_PinRemap_B	SPI 映射管脚组 B

使用示例:

```
/* 把SPI1管脚设置为重映射组A */
SPI_PinRemapConfig(SPI1, SPI_PinRemap_A);
```

综合使用示例（SPI 主机发送 1Byte 数据）

```
volatile uint8_t SPI0_Data, SPI1_Data;
SPI_InitTypeDef SPI_InitStruct; //定义SPI初始化结构体变量
/* 使能SPI0和SPI1的时钟 */
RCC_APB0Cmd(ENABLE);  //SPI0和1外设挂载在APB0时钟上
RCC_APB0PeriphClockCmd(RCC_APB0Periph_SPI0, ENABLE);
RCC_APB1Cmd(ENABLE);
SPI_InitStruct.SPI_CPHA = SPI_CPHA_1Edge;     //SPI第一沿采集数据
SPI_InitStruct.SPI_CPOL = SPI_CPOL_Low;       //SPI空闲电平为低电平
SPI_InitStruct.SPI_DataSize = SPI_DataSize_8B;  //数据长度为8位
SPI_InitStruct.SPI_FirstBit = SPI_FirstBit_MSB;     //数值传输为高位在前
SPI_InitStruct.SPI_Mode = SPI_Mode_Master;      //SPI工作在主机模式
SPI_InitStruct.SPI_Prescaler = SPI_Prescaler_1024;        //SPI传输频率为APB0时钟1分频
/*  配置SPI0为主机 */
SPI_Init(SPI0, &SPI_InitStruct);
/* 使能SPI0中断 */
SPI_ITConfig(SPI0, SPI_IT_INTEN, ENABLE);
NVIC_EnableIRQ(SPI0_IRQn);
/* 使能SPI0 */
SPI_Cmd(SPI0, ENABLE);
while(1)
{
```

```
  SPI_SendData(SPI0, 0x65);        //主机发送
  while(SPI_GetFlagStatus(SPI0, SPI_Flag_SPIF) == RESET);
  SPI_ClearFlag(SPI0, SPI_Flag_SPIF);
}
```

18 QEP 固件库函数

QEP 寄存器结构

赛元SC32F15Gx 系列QEP寄存器结构，QEP_TypeDef，在文件“SC32F15Gx.h”中定义如下：

```
typedef struct
{
  __IO uint32_t QEP_CON;
   __IO uint32_t QEP_PCNT;
   __IO uint32_t QEP_PMAX;
   __IO uint32_t QEP_STS;
   __IO uint32_t QEP_IDE;
} PCAP_TypeDef;
```

QEP 寄存器表格

Table 357

寄存器	描述
QEP_CON	QEP 控制寄存器
QEP_PCNT	QEP 模块计数寄存器
QEP_PMAX	QEP 门限值寄存器
QEP_STS	QEP 标志制寄存器
QEP_IDE	QEP 中断使能寄存器

QEP 固件库函数列

Table 358

函数名	描述
QEP_DeInit	将外设 QEP 寄存器重设为缺省值
QEP_Init	初始化 QEP 寄存器
QEP_StructInit	根据 QEP _InitStruct 中指定的参数初始化外设 QEP 寄存器
QEP_Cmd	使能或者失能 QEP 外设
QEP_QEPnICmd	使能或者失能 QEP 的 QEPnI 通道
QEP_QSRCModeSelect	配置 QEPx 的工作模式
QEP_QFDIVSelect	配置 QEPx 数字输入滤波器滤波宽度选择
QEP_CountSelect	配置 QEP 计数方式
QEP_SetCnt	设置 QEPx 的计数值
QEP_GetCnt	获取 QEPx 的计数值
QEP_SetPmax	设置 QEPx 的计数最大值
QEP_GetPmax	获取 QEPx 的计数最大值
QEP_PinRemapConfig	配置 QEPx 管脚的重映射
QEP_ITConfig	使能或者失能指定的 QEPx 中断
QEP_GetFlagStatus	检查指定的 QEPx 标志位设置与否
QEP_ClearFlag	清除 QEP 的待处理标志位

QEP 固件库函数详解

QEP_DeInit

Table 359

函数名	QEP_DeInit
函数原型	void QEP_DeInit(QEP_TypeDef* QEPx)
功能描述	将外设 QEPx 寄存器重设为缺省值

输入参数	QEPx：x 可为 0 或 1，选择 QEPx 外设
返回值	无

使用示例:

```
/* 将外设 QEPx 寄存器重设为缺省值*/
QEPx_DeInit(QEP0);
```

QEP _Init

Table 360

函数名	QEP _Init
函数原形	void QEP_Init(QEP_TypeDef* QEPx, QEP_InitTypeDef* QEP_InitStruct)
功能描述	根据 QEP_InitStruct 中指定的参数初始化外设 QEP 寄存器
输入参数 1	QEPx：x 可为 0 或 1，选择 QEPx 外设
输入参数 2	QEP_InitStruct：指向结构 QEP_InitTypeDef 的指针
返回值	无

QEP_InitStruct

QEP_InitTypeDef 定义于文件“sc32f1xxx_QPE.h”：

```
typedef struct
{
  uint32_t QEP_Mode;
  uint32_ QEP_GATE;
  uint32_t QEP_CEN;
  uint32_t QEP_QFDIV;
  uint32_t QEP_QAP;
  uint32_t QEP_QBP;
  uint32_t QEP_IndexMode;
  uint32_t QEP_QEPnI;
  uint32_t QEP_SWAP;
  uint32_t QEP_PMAX;
} PCAP_InitTypeDef ;
```

QEP_Mode

QEP_Mode 设置QEP的工作模式，参阅Table 361获得这个参数的所有成员。

Table 361

QEP_Mode	描述
QEP_Mode_Orthogonal	正交计数模式
QEP_Mode_Direction	方向计数模式
QEP_Mode_Dipulse	双脉冲计数模式

QEP_GATE

QEP_GATE引脚选通，参阅Table 362获得这个参数的所有成员。

Table 362

QEP_GATE	描述
QEP_GATE_Default	QEP 引脚选通无
QEP_GATE_QA	QEP 引脚选通 QA
QEP_GATE_QB	QEP 引脚选通 QB
QEP_GATE_QA_QB	QEP 引脚选通 QA 和 QB

QEP_CEN

QEP_CEN配置QEP计数方式，参阅Table 363获得这个参数的所有成员。

Table 363

QEP_CEN	描述
QEP_CEN_Default	QEP 计数方式不使能
QEP_CEN_Rising	QEP 上升沿计数使能
QEP_CEN_Falling	QEP 下降沿计数使能

QEP_QFDIV

QEP_QFDIV数字输入滤波器滤波宽度选择，参阅Table 364获得这个参数的所有成员。

Table 364

QEP_QFDIV	描述
QEP_FilteringDIV_1	数字输入滤波器滤波宽度时钟分频值为 1
QEP_FilteringDIV_2	数字输入滤波器滤波宽度时钟分频值为 2
QEP_FilteringDIV_4	数字输入滤波器滤波宽度时钟分频值为 4
QEP_FilteringDIV_16	数字输入滤波器滤波宽度时钟分频值为 16
QEP_FilteringDIV_32	数字输入滤波器滤波宽度时钟分频值为 32
QEP_FilteringDIV_64	数字输入滤波器滤波宽度时钟分频值为 64
QEP_FilteringDIV_128	数字输入滤波器滤波宽度时钟分频值为 128

QEP_QAP与QEP_QBP

QEP_QAP与QEP_QBP 输入极性，参阅Table 365获得这个参数的所有成员。

Table 365

QEP_QAP 与 QEP_QBP	描述
QEP_Polarity_Positive	正极
QEP_Polarity_Negative	负极

QEP_IndexMode

QEP_IndexMode配置 index事件复位选择，参阅Table 366获得这个参数的所有成员。

Table 366

QEP_IndexMode	描述
QEP_IndexMode_Enable	index 事件复位使能
QEP_IndexMode_Disable	index 事件复位不使能

QEP_QEPnI

QEP_QEPnI 输入极性，参阅Table 367获得这个参数的所有成员。

Table 367

QEP_QAP与QEP_QBP	描述
QEP_QEPnI_Forward	正极
QEP_QEPnI_Reverse	负极

QEP_SWAP

QEP_SWAP配置是否交换正交输入方向，参阅Table 368获得这个参数的所有成员。

Table 368

QEP_SWAP	描述
QEP_SWAP_Default	不交换正交输入方向
QEP_SWAP_Exchange	交换正交输入方向

使用示例:

```
/* 根据 QEP_InitStruct 中指定的参数初始化QEP0 */
QEP_InitTypeDef QEP_InitStruct;                    //定义QEP初始化结构体变量
QEP_InitStruct->QEP_Mode     = QEP_Mode_Orthogonal;// 正交计数模式
QEP_InitStruct->QEP_CEN      = QEP_CEN_Default;// QEP计数方式不使能
QEP_InitStruct->QEP_GATE     = QEP_GATE_Default;// QEP引脚选通无
QEP_InitStruct->QEP_QFDIV    = QEP_FilteringDIV_1;/ 数字输入滤波器滤波宽度时钟分频值为 1
QEP_InitStruct->QEP_QAP      = QEP_Polarity_Positive;//正极
QEP_InitStruct->QEP_QBP      = QEP_Polarity_Positive; //正极
QEP_InitStruct->QEP_IndexMode = QEP_IndexMode_Disable;// index事件复位不使能
QEP_InitStruct->QEP_SWAP     = QEP_SWAP_Default;// 不交换正交输入方向
QEP_InitStruct->QEP_QEPnI    = QEP_QEPnI_Forward;// 正极
QEP _Init(QEP0,& QEP _InitStruct);
```

QEP _StructInit

Table 369

函数名	QEP _StructInit
函数原形	void QEP_StructInit(QEP_InitTypeDef* QEP_InitStruct)
功能描述	把 QEP _InitStruct 中的每一个参数按缺省值填入

输入参数	QEP _InitStruct：指向结构 QEP _InitTypeDef 的指针，待初始化
返回值	无

使用示例:

```
/* 把QEP _InitStruct 中的每一个参数按缺省值填入 */
QEP _InitTypeDef   QEP _InitStruct;
QEP_StructInit(&QEP _InitStruct);
```

QEP_Cmd

Table 370

函数名	QEP_Cmd
函数原形	void QEP_Cmd(QEP_TypeDef* QEPx, FunctionalState NewState)
功能描述	使能或者失能 QEP 外设
输入参数 1	QEPx：x 可为 0 或 1，选择 QEPx 外设
输入参数 2	NewState: 外设 QEPx 的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* 使能QEPx外设 */
QEP_Cmd (QEP0， ENABLE);
```

QEP_QEPnICmd

Table 371

函数名	QEP_QEPnICmd
函数原形	void QEP_QEPnICmd(QEP_TypeDef* QEPx, FunctionalState NewState)
功能描述	使能或者失能 QEP 的 QEPnI 通道
输入参数 1	QEPx：x 可为 0 或 1，选择 QEPx 外设
输入参数 2	NewState: PCAP 的 7/5 滤波的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* 使能QEP的 QEPnI通道*/
QEP_QEPnICmd (QEP0， ENABLE);
```

QEP_QSRCModeSelect

Table 372

函数名	QEP_QSRCModeSelect
函数原形	void QEP_QSRCModeSelect(QEP_TypeDef* QEPx, QEP_Mode_TypeDef QEP_Mode)
功能描述	配置 QEPx 的工作模式
输入参数 1	QEPx：x 可为 0 或 1，选择 QEPx 外设
输入参数 2	QEP_Mode: QEP 工作模式
返回值	无

使用示例:

```
/* 配置QEP 工作模式为正交计数模式 */
QEP_QSRCModeSelect(QEP0, QEP_Mode_Orthogonal);
```

QEP_QFDIVSelect

Table 373

函数名	QEP_QFDIVSelect
函数原形	void QEP_QFDIVSelect(QEP_TypeDef* QEPx, QEP_FilteringDIV_TypeDef QEP_FilteringDIV)
功能描述	配置 QEPx 数字输入滤波器滤波宽度选择
输入参数 1	QEPx：x 可为 0 或 1，选择 QEPx 外设

输入参数 2	QEP_FilteringDIV: QEPx 数字输入滤波器滤波宽度选择
返回值	无

使用示例:

```
/* 设置QEPx数字输入滤波器滤波宽度时钟分频值为1 */
QEP_QFDIVSelect(QEP0, QEP_FilteringDIV_1);
```

QEP_CountSelect

Table 374

函数名	QEP_CountSelect
函数原形	void QEP_CountSelect(QEP_TypeDef* QEPx, QEP_CEN_TypeDef QEP_CEN)
功能描述	配置 QEP 计数方式
输入参数 1	QEPx：x 可为 0 或 1，选择 QEPx 外设
输入参数 2	QEP_CEN: QEP 计数方式选择
返回值	无

使用示例:

```
/* QEP0上升沿计数使能 */
QEP_CountSelect(QEP0, QEP_CEN_Rising);
```

QEP_SetCnt

Table 375

函数名	QEP_SetCnt
函数原形	void QEP_SetCnt(QEP_TypeDef* QEPx,uint16_t QEP_CountValue)
功能描述	设置 QEPx 的计数值
输入参数 1	QEPx：x 可为 0 或 1，选择 QEPx 外设
输入参数 2	QEP_CountValue: 设置 QEP 计数
返回值	无

使用示例:

```
/*设置QEP0的计数值*/
QEP_SetCnt(QEP0,0XFF);
```

QEP_GetCnt

Table 376

函数名	QEP_GetCnt
函数原形	uint16_t QEP_GetCnt(QEP_TypeDef* QEPx)
功能描述	获取 QEPx 计数值
输入参数	QEPx：x 可为 0 或 1，选择 QEPx 外设
返回值	QEP 计数值

使用示例:

```
/* 获取QEP0计数值 */
QEP_GetCnt(QEP0);
```

QEP_SetPmax

Table 377

函数名	QEP_SetPmax
函数原形	void QEP_SetPmax(QEP_TypeDef* QEPx,uint16_t QEP_PmaxValue)
功能描述	设置 QEPx 的计数值最大值
输入参数 1	QEPx：x 可为 0 或 1，选择 QEPx 外设
输入参数 2	QEP_PmaxValue: 设置 QEP 计数最大值
返回值	无

使用示例:

```
/*设置QEP0的计数值最大值*/
```

QEP_SetPmax (QEP0,0XFF);

QEP_GetPmax

Table 378

函数名	QEP_GetPmax
函数原形	uint16_t QEP_GetPmax(QEP_TypeDef* QEPx)
功能描述	获取 QEPx 的计数值最大值
输入参数	QEPx：x 可为 0 或 1，选择 QEPx 外设
返回值	QEP 计数值最大值

使用示例:

```
/* 获取QEP0计数值最大值 */
QEP_GetPmax (QEP0);
```

QEP_PinRemapConfig

Table 379

函数名	QEP_PinRemapConfig
函数原型	void QEP_PinRemapConfig(QEP_TypeDef* QEPx, QEP_PinRemap_TypeDef QEP_Remap)
功能描述	配置 QEPx 管脚的重映射
输入参数 1	QEPx：x 可为 0 或 1，选择 QEPx 外设
输入参数 2	QEP_Remap：QEPx 管脚的选择
返回值	无

QEP_Remap

QEP管脚映射设定，参阅Table 380获得这个参数的所有成员。

Table 380

QEP_PinRemap	描述
QEP_PinRemap_Default	默认 QEP 管脚映射
QEP_PinRemap_A	QEP 映射管脚组 A

使用示例:

```
/* 配置QEP引脚映射*/
QEP_PinRemapConfig(QEP0, QEP_PinRemap_A);
```

QEP_ITConfig

Table 381

函数名	QEP_ITConfig
函数原形	void QEP_ITConfig(QEP_TypeDef* QEPx, uint16_t QEP_IT, FunctionalState NewState)
功能描述	使能或者失能指定的 QEPx 中断
输入参数 1	QEPx：x 可为 0 或 1，选择 QEPx 外设
输入参数 2	QEP_IT：待使能或者失能的 QEPx 中断源
输入参数 3	NewState：QEP 中断的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

QEP_IT

QEP_IT请求选择，参阅Table 382获得这个参数的所有成员。

Table 382

QEP_IT	描述
QEP_IT_INT	QEPx 中断
QEP_IT_PCU	位置计数器下溢中断
QEP_IT_PCO	位置计数器溢出中断
QEP_IT_IERIndex	index 事件重置中断
QEP_IT_UPEVNTUnit	Unit position event 中断

使用示例:

```
/* 使能QEP0 外设总中断 */
QEP_ITConfig (QEP0,QEP_IT_INT，ENABLE);
```

QEP_GetFlagStatus

Table 383

函数名	QEP_GetFlagStatus
函数原形	FlagStatus QEP_GetFlagStatus(QEP_TypeDef* QEPx, uint16_t QEP_FLAG)
功能描述	检查指定的 QEPx 标志位设置与否
输入参数 1	QEPx：x 可为 0 或 1，选择 QEPx 外设
输入参数 2	QEP_FLAG：待检查的 QEPx 标志位
返回值	QEP_FLAG 的新状态

QEP_FLAG

QEP 中断标志位选择，参阅Table 384获得这个参数的所有成员。

Table 384

QEP_FLAG	描述
QEP_Flag_PCU	计数器下溢中断标志
QEP_Flag_PCO	计数器溢出中断标志
QEP_Flag_IER	index 事件重置中断标志
QEP_Flag_UPEVNT	单位位置事件标志
QEP_Flag_DQ	正交方向标志

使用示例:

```
/* 获得QEP0计数器下溢中断标志 */
QEP_GetFlagStatus (QEP0, QEP_Flag_PCU);
```

QEP_ClearFlag

Table 385

函数名	QEP_ClearFlag
函数原形	void QEP_ClearFlag(QEP_TypeDef* QEPx, uint16_t QEP_FLAG)
功能描述	清除 QEP 的待处理标志位
输入参数 1	QEPx：x 可为 0 或 1，选择 QEPx 外设
输入参数 2	QEP_FLAG：待清除的 QEP 标志位
返回值	无

使用示例:

```
/* 清除QEP外设标志位 */
QEP_ClearFlag(PCAP, QEP_Flag_PCU);
```

综合使用示例（QEP0 获取正交编码器码值）

```
void QEP_CaptureEncodingValues_Function(void)
{
  uint32_t QEPCnt;
  /* 使能QEP时钟 */
  RCC_APB2Cmd(ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_QEP0,ENABLE);    //开启QEP0对应外设时钟
  //QEP0配置初始化结构体
  QEP_InitTypeDef QEP0_InitStruct;              //定义QEP0初始化结构体
  QEP_StructInit(&QEP0_InitStruct);
  QEP0_InitStruct.QEP_Mode = QEP_Mode_Orthogonal;      //选择正交计数模式
  QEP0_InitStruct.QEP_CEN = QEP_CEN_Rising;           //设置上升沿计数使能
  QEP0_InitStruct.QEP_GATE = QEP_GATE_QA_QB;          //使能A/B/I引脚
  QEP0_InitStruct.QEP_QFDIV = QEP_FilteringDIV_2;      //滤波1/2分频
  QEP0_InitStruct.QEP_QAP = QEP_Polarity_Positive;     //QEP A端口输入极性为正向
```

```
    QEP0_InitStruct.QEP_QBP = QEP_Polarity_Positive;      //QEP B端口输入极性为正向
    QEP0_InitStruct.QEP_IndexMode = QEP_IndexMode_Enable; //Index事件复位使能
    QEP0_InitStruct.QEP_QEPnI = QEP_QEPnI_Forward;        //Index正向复位
    QEP0_InitStruct.QEP_SWAP = QEP_SWAP_Default;          //正交输入方向不交换
    QEP0_InitStruct.QEP_PMAX = 1000;                      //配置计数最大值
    QEP_Init(QEP0,&QEP0_InitStruct);
    QEP_ITConfig(QEP0,QEP_IT_INT|QEP_IT_PCU|QEP_IT_PCO|QEP_IT_IER|QEP_IT_UPEVNT,ENABLE);
    QEP_Cmd(QEP0,ENABLE);
    QEPCnt = QEP_GetCnt(QEP0);
    while(1)
    {
       for(uint16_t i=0;i<65535;i++)
     {
        for(uint16_t j=0;j<30;j++);
     }
     QEPCnt = QEP_GetCnt(QEP0);
     printf("QEP计数值为:0x=%d",QEPCnt);
    }
}
```

19 TIM 固件库函数

SC32F1xxx 系列有多个独立 16 位定时/计数器，具有计数方式和定时方式两种工作模式。
TIM是可通过可编程预分频器驱动的 16 位自动装载计数器。其有 4 种工作模式：

1. 模式 0：16 位捕获模式，新增PWM捕获
2. 模式 1：16 位自动重载定时/计数器模式
3. 模式 3：可编程时钟输出模式
4. 模式 4：PWM输出模式

TIM 寄存器结构

TIM 寄存器结构，TIM_TypeDef，在文件“sc32f1xxx.h”中定义如下：

```
typedef struct
{
  __IO uint32_t TIM_CON;
  __IO uint32_t TIM_CNT;
  __IO uint32_t TIM_RLD;
  __IO uint32_t TIM_STS;
  __IO uint32_t TIM_PDTA_RCAP;
  __IO uint32_t TIM_PDTB_FCAP;
  __IO uint32_t TIM_IDE;
} TIM_TypeDef;
```

TIM 寄存器**表格**

Table 386

寄存器	描述
TIM_CON	TIM控制寄存器
TIM_CNT	定时器计数值寄存器
TIM_RLD	定时器重载寄存器
TIM_STS	定时标志制寄存器
TIM_PDTA_RCAP	PWMA占空比设定寄存器/上升沿数据捕获寄存器
TIM_PDTB_FCAP	PWMB占空比设定寄存器/下降沿数据捕获寄存器
TIM_IDE	通信口DMA控制寄存器

TIM 固件库函数列表

Table 387

函数名	描述
TIM_DeInit	TIMx 相关寄存器复位至缺省值
TIM_TIMBaseInit	根据 TIM_TimeBaseInitStruct 中指定的参数初始化TIMx 的时间基数单位
TIM_TimeBaseStructInit	把 TIM_TimeBaseInitStruct 中的每一个参数按缺省值填入
TIM_Cmd	使能或者失能 TIMx 外设
TIM_SetCounter	设置 TIMx 计数器寄存器值
TIM_GetCounter	获得 TIMx 计数器的值
TIM_SetAutoreload	设置 TIMx 自动重装载寄存器值
TIM_GetAutoreload	获得 TIMx 自动重装载的值
TIM_SetPerscaler	设置 TIMx 预分频
TIM_GetPrescaler	获得 TIMx 预分频值
TIM_ICInit	根据 TIM_IC_InitStruct 中指定的参数初始化TIMx的输入捕获模式
TIM_ICStructInit	把 TIM_IC_InitStruct 中的每一个参数按缺省值填入
TIM_ICCmd	使能或者失能 TIMx 的捕获功能
TIM_GetRisingCapture	获得 TIMx 上升沿捕获值

TIM_GetFailingCapture	获得 TIMx 下降沿捕获值
TIM_PWMInit	根据 TIM_PWM_InitStruct 中指定的参数初始化TIMx的PWM模式
TIM_PWMStructInit	把 TIM_PWM_InitStruct 中的每一个参数按缺省值填入
TIM_PWMSetDuty	TIMx的PWM占空比设置
TIM_PWMGetDuty	获取TIMx的PWM占空比值
TIM_ClockOutputCmd	使能或者失能 TIMx 的可编程时钟输出
TIM_PinRemapConfig	配置TIM2/3/7管脚的重映射
TIM_ITConfig	使能或者失能指定的 TIM 中断
TIM_GetFlagStatus	检查指定的 TIM 标志位设置与否
TIM_ClearFlag	清除 TIMx 的指定标志位
TIM_DMACmd	使能或者失能指定的 TIMx 的 DMA 请求

TIM 固件库函数详解

TIM_DeInit

Table 388

函数名	TIM_DeInit
函数原型	void TIM_DeInit(TIM_TypeDef* TIMx)
功能描述	TIMx 相关寄存器复位至缺省值
输入参数	TIMx：x 可以是 0~7，来选择 TIM 外设
返回值	无

例:

```
/* TIM0寄存器复位 */
TIM_DeInit(TIM0);
```

TIM_TIMBaseInit

Table 389

函数名	TIM_TIMBaseInit
函数原型	void TIM_TIMBaseInit(TIM_TypeDef* TIMx, TIM_TimeBaseInitTypeDef* TIM_TimeBaseInitStruct)
功能描述	根据 TIM_TimeBaseInitStruct 中指定的参数初始化 TIMx 的时间基数单位
输入参数 1	TIMx：x 可以是 0~7，来选择 TIM 外设
输入参数 2	TimeBaseInitStruct：指向结构 TIM_TimeBaseInitTypeDef 的指针，包含了 TIMx 时间基数单位的配置信息
返回值	无

TIM_TimeBaseInit

TIM_TimeBaseInitTypeDef 定义于文件“sc32f1xxx_tim.h”：

```
typedef struct
{
uint16_t TIM_Prescaler;
uint16_t TIM_WorkMode
uint16_t TIM_CounterMode;
uint16_t TIM_EXENX;
uint16_t TIM_Preload;
} TIM_TimeBaseInitTypeDef;
```

TIM_Prescaler

TIM时钟频率档位设定，参阅Table 390获得这个参数的所有成员。

Table 390

TIM_Prescaler	描述
TIM_PRESCALER_1	时钟预分频值为 1
TIM_PRESCALER_2	时钟预分频值为 2
TIM_PRESCALER_4	时钟预分频值为 4

TIM_PRESCALER_8	时钟预分频值为 8
TIM_PRESCALER_16	时钟预分频值为 16
TIM_PRESCALER_32	时钟预分频值为 32
TIM_PRESCALER_64	时钟预分频值为 64
TIM_PRESCALER_128	时钟预分频值为 128

TIM_WorkMode

TIM工作模式设定，参阅Table 391获得这个参数的所有成员。

Table 391

TIM_WorkMode	描述
TIM_WorkMode_Timer	工作在定时器方式
TIM_WorkMode_Counter	工作在计数器方式

TIM_CounterMode

TIM计数模式设定，参阅Table 392获得这个参数的所有成员。

Table 392

TIM_CounterMode	描述
TIM_CounterMode_Up	TIMn 为递增的定时/计数器
TIM_CounterMode_Down_UP	TIMn 作为递增/递减的定时/计数器，TnEX 用来选择计数方向

TIM_EXENX

TIM重载模式设定，参阅Table 393获得这个参数的所有成员。

Table 393

TIM_EXENX	描述
TIM_EXENX_Disable	忽略 TnEX 引脚上的事件
TIM_EXENX_Enable	检测到 TnEX 引脚上一个下降沿，产生一个重载

TIM_Preload

TIM重载值设置。

使用示例:

```
/* 根据TIM_TimeBaseInitStruct 中指定的参数初始化TIM0的时间基数单位 */
TIM_TimeBaseInitTypeDef   TIM_BaseInitStruct;
TIM_BaseInitStruct.TIM_Prescaler = TIM_PRESCALER_1;  //TIM0时钟频率为Fsource/1
TIM_BaseInitStruct.TIM_WorkMode = TIM_WorkMode_Timer;  //设置为定时器模式
TIM_BaseInitStruct.TIM_CounterMode = TIM_CounterMode_Up;  //计数方式为向上计数
TIM_BaseInitStruct.TIM_EXENX = TIM_EXENX_Disable;  //捕获模式不使能，即设置为重载模式
TIM_BaseInitStruct.TIM_Preload = 65536-1000;    //计数初值为65566-1000
TIM_TIMBaseInit(TIM0,&TIM_BaseInitStruct);
```

TIM_TimeBaseStructInit

Table 394

函数名	TIM_TimeBaseStructInit
函数原型	void TIM_TimeBaseStructInit(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_TimeBaseInitTypeDef 的指针，待初始化
返回值	无

使用示例:

```
/* 把 TIM_TimeBaseInitStruct 中的每一个参数按缺省值填入 */
TIM_TimeBaseInitTypeDef   TIM_TimeBaseInitStruct;
TIM_TimeBaseStructInit(&TIM_TimeBaseInitStruct);
```

TIM_Cmd

Table 395

函数名	TIM_Cmd
函数原型	void TIM_Cmd(TIM_TypeDef* TIMx, FunctionalState NewState)

功能描述	使能或者失能 TIMx 外设
输入参数 1	TIMx：x 可以是 0~7，来选择 TIM 外设
输入参数 2	NewState：TIM 功能开启/关闭
返回值	无

使用示例:

```
/* 使能Timer0 */
TIM_Cmd(TIM0, ENABLE);
```

TIM_SetCounter

Table 396

函数名	TIM_SetCounter
函数原型	void TIM_SetCounter(TIM_TypeDef* TIMx, uint32_t Counter)
功能描述	设置 TIMx 计数器寄存器值
输入参数 1	TIMx：x 可以是 0~7，来选择 TIM 外设
输入参数 2	Counter：TIM 计数值
返回值	无

使用示例:

```
/* 设置TIM0计数初值为 1000 */
TIM_SetCounter(TIM0,1000);
```

TIM_GetCounter

Table 397

函数名	TIM_GetCounter
函数原型	uint32_t TIM_GetCounter(TIM_TypeDef* TIMx)
功能描述	获得 TIMx 计数器的值
输入参数	TIMx：x 可以是 0~7，来选择 TIM 外设
返回值	TIMx 计数值

使用示例:

```
/* 获取TIM0计数值 */
uint32_t  TIM_Counter;
TIM_Counter = TIM_GetCounter(TIM0);
```

TIM_SetAutoreload

Table 398

函数名	TIM_SetAutoreload
函数原型	void TIM_SetAutoreload(TIM_TypeDef* TIMx, uint16_t Autoreload)
功能描述	设置 TIMx 自动重装载寄存器值
输入参数 1	TIMx：x 可以是 0~7，来选择 TIM 外设
输入参数 2	Autoreload：TIMx 计数重载值
返回值	无

使用示例:

```
/* 设置TIM0计数重载值为 49535 */
TIM_SetAutoreload(TIM0,49535);
```

TIM_GetAutoreload

Table 399

函数名	TIM_GetAutoreload
函数原型	Uint16_t TIM_GetAutoreload(TIM_TypeDef* TIMx)
功能描述	获得 TIMx 自动重装载的值
输入参数	TIMx：x 可以是 0~7，来选择 TIM 外设
返回值	TIMx 计数重载值

使用示例:

```
/* 获取TIM0计数重载值 */
uint32_t  TIM_Autoreload;
TIM_Autoreload = TIM_GetAutoreload(TIM0);
```

TIM_SetPerscaler

Table 400

函数名	TIM_SetPerscaler
函数原型	void TIM_SetPerscaler(TIM_TypeDef* TIMx, TIM_Prescaler_TypeDef TIM_Perscaler)
功能描述	设置 TIMx 预分频
输入参数 1	TIMx：x 可以是 0~7，来选择 TIM 外设
输入参数 2	TIM_Perscaler：根据 TIM_Prescaler_TypeDef 定义的类型选择 TIMx 的频率，参阅 Table 390 获得这个参数的所有成员
返回值	无

使用示例:

```
/*设置TIM0的预分频为Fsource/4 */
TIM_SetPerscaler(TIM0,TIM_PRESCALER_4);
```

TIM_GetPrescaler

Table 401

函数名	TIM_GetPrescaler
函数原型	TIM_Prescaler_TypeDef TIM_GetPrescaler(TIM_TypeDef* TIMx)
功能描述	获得 TIMx 预分频值
输入参数	TIMx：x 可以是 0~7，来选择 TIM 外设
返回值	TIMx 预分频

使用示例:

```
/* 获得TIM0预分频值 */
TIM_Prescaler_TypeDef TIM0Prescaler;
TIM0Prescaler = TIM_GetPrescaler(TIM0);
```

TIM_ICInit

Table 402

函数名	TIM_ICInit
函数原型	void TIM_ICInit(TIM_TypeDef* TIMx, TIM_IC_InitTypeDef* TIM_IC_InitStruct)
功能描述	根据 TIM_IC_InitStruct 中指定的参数初始化外设 TIMx
输入参数 1	TIMx：x 可以是 0~7，来选择 TIM 外设
输入参数 2	TIM_IC_InitStruct：指向结构 TIM_IC_InitTypeDef 的指针，初始化 TIMx 的输入捕获
返回值	无

TIM_IC_InitStruct

TIM_IC_InitTypeDef 定义于文件“sc32f1xxx_tim.h”：

```
typedef struct
{
uint16_t TIM_RICPIN;
uint16_t TIM_FICPIN
}TIM_IC_InitTypeDef;
```

TIM_RICPIN

TIM上升沿管脚选择，参阅Table 403获得这个参数的所有成员。

Table 403

TIM_RICPIN	描述
TIM_RICPin_Disable	不开启上升沿捕获
TIM_RICPin_Tn	上升沿捕获管脚为 Tn 管脚

TIM_FICPIN

TIM下降沿管脚选择，参阅Table 404获得这个参数的所有成员。

Table 404

TIM_FICPIN	描述
TIM_FICPin_Disable	不开启下降沿捕获
TIM_FICPin_Tn	下降沿捕获管脚为 Tn 管脚
TIM_FICPin_TnEx	下降沿捕获管脚为 TnEx 管脚

使用示例:

```
/* TIM0初始化为T0脚输入的上升沿捕获模式 */
TIM_IC_InitTypeDef TIM_IC_InitStruct;
TIM_IC_InitStruct.TIM_FICPIN = TIM_FICPin_Disable; //下降沿捕获不使能
TIM_IC_InitStruct.TIM_RICPIN = TIM_RICPin_Tn;  //使能Tn脚输入的上升沿捕获
TIM_ICInit(TIM0,&TIM_IC_InitStruct);
```

TIM_ICStructInit

Table 405

函数名	TIM_ICStructInit
函数原型	void TIM_ICStructInit(TIM_IC_InitTypeDef* TIM_IC_InitStruct)
功能描述	把 TIM_IC_InitStruct 中的每一个参数按缺省值填入
输入参数 1	TIM_IC_InitStruct：指向结构 TIM_IC_InitTypeDef 的指针，待初始化
返回值	无

使用示例:

```
/* TIM_IC_InitStruct 中的每一个参数按缺省值填入*/
TIM_IC_InitTypeDef IM_IC_InitStruct;
TIM_ICStructInit(&TIM_RICPIN);
```

TIM_ICCmd

Table 406

函数名	TIM_ICCmd
函数原型	void TIM_ICCmd(TIM_TypeDef* TIMx, FunctionalState NewState)
功能描述	使能或者失能 TIMx 的捕获功能
输入参数 1	TIMx：x 可以是 0~7，来选择 TIM 外设
输入参数 2	NewState：TIMx 捕获功能的开启/关闭
返回值	无

使用示例:

```
/* 使能TIM0捕获功能  */
TIM_ICCmd(TIM0,ENABLE);
```

TIM_GetRisingCapture

Table 407

函数名	TIM_GetRisingCapture
函数原型	uint32_t TIM_GetRisingCapture(TIM_TypeDef* TIMx)
功能描述	获得 TIMx 上升沿捕获值
输入参数	TIMx：x 可以是 0~7，来选择 TIM 外设
返回值	TIMx 的上升沿捕获值

使用示例:

```
/* 获得TIM0的上升沿捕获值  */
uint32_t  TIM_RisingValue;
TIM_RisingValue = TIM_GetRisingCapture(TIM0);
```

TIM_GetFailingCapture

Table 408

函数名	TIM_GetFailingCapture
函数原型	uint32_t TIM_GetFailingCapture(TIM_TypeDef* TIMx)
功能描述	获得 TIMx 下降沿捕获值
输入参数	TIMx：x 可以是 0~7，来选择 TIM 外设
返回值	TIMx 的下降沿捕获值

使用示例:

```
/* 获得TIM0的下降沿捕获值 */
uint32_t  TIM_FailingValue;
TIM_FailingValue = TIM_GetFailingCapture (TIM0);
```

TIM_PWMInit

Table 409

函数名	TIM_PWMInit
函数原型	void TIM_PWMInit(TIM_TypeDef* TIMx, TIM_PWM_InitTypeDef* TIM_PWM_InitStruct)
功能描述	根据 TIM_PWM_InitStruct 中指定的参数初始化 TIMx 的 PWM 模式
输入参数 1	TIMx：x 可以是 0~7，来选择 TIM 外设
输入参数 2	TIM_PWM_InitStruct：指向结构 TIM_PWM_InitTypeDef 的指针，初始化 TIMx 的 PWM 模式
返回值	无

TIM_PWM_InitTypeDef

TIM_PWM_InitTypeDef 定义于文件“sc32f1xxx_tim.h”：

```
typedef struct
{
   uint16_t TIM_PWMOutputChannl;
   uint16_t TIM_PWMLowPolarityChannl;
}TIM_PWM_InitTypeDef;
```

TIM_PWMOutputChannl 与 PWMLowPolarityChannl

PWM输出通道与反向通道选择，参阅Table 410获得这个参数的所有成员。

Table 410

TIM_PWMChannel	描述
TIM_PWMChannel_Less	不选择 TIM PWM 通道
TIM_PWMChannel_PWMB	TIM PWM 通道 B
TIM_PWMChannel_PWMA	TIM PWM 通道 A
TIM_PWMChannel_ALL	TIM PWM 通道 A 和通道 B

使用示例:

```
/* 根据 TIM_PWM_InitStruct 中指定的参数初始化TIM0的PWM模式 */
TIM_PWM_InitTypeDef   TIM_PWM_InitStruct;
TIM_PWM_InitStruct.TIM_PWMLowPolarityChannl = TIM_PWMChannel_A | TIM_PWMChannel_B;
                                                    //使能PWMA/B的波形输出
TIM_PWM_InitStruct.TIM_PWMOutputChannl = TIM_PWMChannel_PWMB;  //PWMB波形输出反向
TIM_PWMInit(TIM0,&TIM_PWM_InitStruct);
```

TIM_PWMStructInit

Table 411

函数名	TIM_PWMStructInit
函数原型	void TIM_PWMStructInit(TIM_PWM_InitTypeDef* TIM_PWM_InitStruct)
功能描述	把 TIM_PWM_InitStruct 中的每一个参数按缺省值填入
输入参数 1	TIM_PWM_InitStruct：指向 TIM_PWM_InitTypeDef 结构的指针，待初始化
返回值	无

使用示例:

```
/* 把 TIM_PWM_InitStruct 中的每一个参数按缺省值填入 */
TIM_PWM_InitTypeDef TIM_PWM_InitStruct ;
TIM_PWMStructInit(&TIM_PWM_InitStruct);
```

TIM_PWMSetDuty

Table 412

函数名	TIM_PWMSetDuty
函数原型	void TIM_PWMSetDuty(TIM_TypeDef* TIMx, TIM_PWMChannel_Typedef TIM_PWMChannel, uint16_t PWM_DutyValue)
功能描述	TIMx 的 PWM 占空比设置
输入参数 1	TIMx：x 可以是 0~7，来选择 TIM 外设
输入参数 2	TIM_PWMChannel：选择 PWM 通道
输入参数 3	PWM_DutyValue：TIMx 的 PWM 输出占空比寄存器的值
返回值	无

使用示例:

```
/* 设置TIM0  PWMA通道占空比寄存器的值为500 */
TIM_PWMSetDuty(TIM0, TIM_PWMChannel_PWMA, 500);
```

TIM_PWMGetDuty

Table 413

函数名	TIM_PWMGetDuty
函数原型	uint16_t TIM_PWMGetDuty(TIM_TypeDef* TIMx, TIM_PWMChannel_Typedef TIM_PWMChannel)
功能描述	获取 TIMx 的 PWM 占空比值
输入参数 1	TIMx：x 可以是 0~7，来选择 TIM 外设
输入参数 2	TIM_PWMChannel：选择 PWM 通道
返回值	TIMx 的 PWM 占空比值

使用示例:

```
/* 获得TIM0  PWMA通道占空比寄存器的值  */
uint16_t   TIM_PWMDuty;
TIM_PWMDuty = TIM_PWMGetDuty(TIM0, TIM_PWMChannel_PWMA);
```

TIM_ClockOutputCmd

Table 414

函数名	TIM_ClockOutputCmd
函数原型	void TIM_ClockOutputCmd(TIM_TypeDef* TIMx, FunctionalState NewState)
功能描述	使能或者失能 TIMx 的可编程时钟输出
输入参数 1	TIMx：x 可以是 0~7，来选择 TIM 外设
输入参数 2	NewState：TIMx 的可编程时钟输出开启/关闭
返回值	无

使用示例:

```
/* 使能TIM0的可编程时钟输出 */
TIM_ClockOutputCmd(TIM0, ENABLE)
```

TIM_PinRemapConfig

Table 415

函数名	TIM_PinRemapConfig
函数原型	void TIM_PinRemapConfig(TIM_TypeDef* TIMx, TIM_PinRemap_TypeDef TIM_Remap)
功能描述	配置 TIM2/3/7 管脚的重映射
输入参数 1	TIMx：x 可以是 2、3、7，来选择 TIM 外设
输入参数 2	TIM_Remap：TIM2/3/7 管脚的选择

返回值	无

TIM_PinRemap

TIM管脚映射设定，参阅Table 416获得这个参数的所有成员。

Table 416

TIM_PinRemap	描述
TIM_PinRemap_Default	默认 TIM 管脚映射
TIM_PinRemap_A	TIM 映射管脚组 A

使用示例:

```
/* 配置T2EX、T2CAP/T2管脚为PB7、PA12 */
TIM_PinRemapConfig(TIM2, TIM_PinRemap_A)
```

TIM_ITConfig

Table 417

函数名	TIM_ITConfig
函数原型	void TIM_ITConfig(TIM_TypeDef* TIMx, uint16_t TIM_IT, FunctionalState NewState)
功能描述	使能或者失能指定的 TIM 中断
输入参数 1	TIMx：x 可以是 0~7，来选择 TIM 外设
输入参数 2	TIM_IT：TIMx 中断请求，定义在 TIM_IT_TypeDef 中
输入参数 3	NewState：TIMx 中断开启/关闭
返回值	无

TIM_IT

TIM中断使能设定，参阅Table 418获得这个参数的所有成员。

Table 418

TIM_IT	描述
TIM_IT_INTEN	中断请求 CPU 的使能控制
TIM_IT_TI	定时器溢出中断
TIM_IT_EXR	外部事件输入上升沿中断
TIM_IT_EXF	外部事件输入下降沿中断

使用示例:

```
/* 使能TIM0定时器溢出中断 */
TIM_ITConfig(TIM0, TIM_IT_INTEN | TIM_IT_TI ,ENABLE);
```

TIM_GetFlagStatus

Table 419

函数名	TIM_GetFlagStatus
函数原型	FlagStatus TIM_GetFlagStatus(TIM_TypeDef* TIMx, TIM_Flag_TypeDef TIM_FLAG)
功能描述	检查指定的 TIM 标志位设置与否
输入参数 1	TIMx：x 可以是 0~7，来选择 TIM 外设
输入参数 2	TIM_FLAG：TIMx 的标志位,参阅 Table 420 获得这个参数的所有成员
返回值	TIMx 的指定标志状态

TIM_FLAG

TIM标志位选择，参阅Table 420获得这个参数的所有成员。

Table 420

TIM_FLAG	描述
TIM_Flag_TI	定时器溢出标志位
TIM_Flag_EXR	Tn 引脚外部事件输入上升沿被检测到的标志位
TIM_Flag_EXF	外部事件输入下降沿被检测到的标志位

使用示例:

```
/* 检查TIM0定时器溢出标志位设置与否 */
FlagStatus  TIM_Status;
TIM_Status  = TIM_GetFlagStatus (TIM0, TIM_Flag_TI);
```

TIM_ClearFlag

Table 421

函数名	TIM_ClearFlag
函数原型	void TIM_ClearFlag(TIM_TypeDef* TIMx, uint16_t TIM_FLAG)
功能描述	清除 TIMx 的指定标志位
输入参数 1	TIMx：x 可以是 0~7，来选择 TIM 外设
输入参数 2	TIM_FLAG：TIMx 的标志位，定义在 TIM_Flag_TypeDef 中
返回值	无

使用示例:

```
/* 清除TIM0的定时器溢出标志位 */
TIM_ClearFlag(TIM0, TIM_Flag_TI);
```

TIM_DMACmd

Table 422

函数名	TIM_DMACmd
函数原型	void TIM_DMACmd(TIM_TypeDef* TIMx, uint16_t TIM_DMAReq, FunctionalState NewState)
功能描述	使能或者失能指定的 TIMx 的 DMA 请求
输入参数 1	TIMx：x 可以是 0~7，来选择 TIM 外设
输入参数 2	TIM_DMAReq：TIMx 的 DMA 请求源，定义在 TIM_DMAReq_TypeDef 中
输入参数 3	NewState：TIMx 的 DMA 请求开启/关闭
返回值	无

TIM_DMAReq

TIM DMA请求选择，参阅Table 423获得这个参数的所有成员。

Table 423

TIM_DMAReq	描述
TIM_DMAReq_TI	定时器溢出事件触发 DMA 请求
TIM_DMAReq_CAPR	上升沿捕获事件触发 DMA 请求
TIM_DMAReq_CAPF	下降沿捕获事件触发 DMA 请求

使用示例:

```
/* TIM0定时器溢出事件触发DMA请求使能 */
TIM_DMACmd(TIM0, TIM_DMAReq_TI,ENABLE);
```

综合使用示例（TIM 定时时间为 1mS）

```
void TIM_TimeBase_Function(void)
{
  TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStruct;

  /* 使能TIM0时钟 */ RCC_APB0Cmd(ENABLE);
  RCC_APB0PeriphClockCmd(RCC_APB0Periph_TIM0, ENABLE);

  /* TIM配置结构体初始化 */ TIM_TimeBaseStructInit(&TIM_TimeBaseInitStruct);

  /* 设置TIM配置结构体 */
  TIM_TimeBaseInitStruct.TIM_CounterMode = TIM_CounterMode_Up;                //TIM工作在向上计数模式
  TIM_TimeBaseInitStruct.TIM_EXENX = TIM_EXENX_Disable;       //外部重加载管脚失能
  TIM_TimeBaseInitStruct.TIM_Preload = 65535 - (32000000 / 1000);          //设置TIM的定时频率为
1000Hz@APB0Clock=32M
  TIM_TimeBaseInitStruct.TIM_Prescaler = TIM_PRESCALER_1;     //TIM分频为1分频
  TIM_TimeBaseInitStruct.TIM_WorkMode = TIM_WorkMode_Timer;//TIM工作在定时器模式，时钟源为APB0时
钟
```

```
  /* TIM时基配置 */
  TIM_TIMBaseInit(TIM0, &TIM_TimeBaseInitStruct);
  /* 使能TIM0中断 */
  TIM_ITConfig(TIM0, TIM_IT_TI | TIM_IT_INTEN, ENABLE);
  NVIC_EnableIRQ(TIMER0_IRQn);
  /* TIM外设初始化 */ TIM_Cmd(TIM0, ENABLE);
  while(1);
}

void TIM_TimeBase_TIM0Handler(void)
{
  /* 判断TIM0中断是否置起 */
  if(TIM_GetFlagStatus(TIM0, TIM_Flag_TI) == SET)
  {
    TIM_ClearFlag(TIM0, TIM_Flag_TI);        //清除TIM0中断
  }
}
```

20 TWI 固件库函数

SC32F1xxx 系列有TWI0 和TWI1，可配置为主模式或从属模式，支持DMA。主机模式下TWI通信速率默认为最小分频档位（Fpclk/4）。

TWI 寄存器结构

TWI 寄存器结构，TWI_TypeDef，在文件“sc32f1xxx.h”中定义如下：

```
typedef struct
{
__IO uint32_t TWI_CON;
__IO uint32_t TWI_STS;
__IO uint32_t TWI_ADD;
__IO uint32_t TWI_DATA;
__IO uint32_t TWI_IDE;
} TWI_TypeDef;
```

TWI 寄存器表格

Table 424

寄存器	描述
TWI_CON	TWI控制寄存器
TWI_STS	TWI标志位寄存器
TWI_ADD	TWI地址寄存器
TWI_DATA	TWI数据寄存器
TWI_IDE	通信口DMA控制寄存器

TWI 固件库函数列

Table 425

函数名	描述
TWI_DeInit	将外设 TWIx 寄存器重设为缺省值
TWI_Init	根据 TWI_InitStruct 中指定的参数初始化外设 TWIx 寄存器
TWI_StructInit	把 TWI_InitStruct 中的每一个参数按缺省值填入
TWI_Cmd	使能或者失能 TWI 外设
TWI_DMACmd	使能或者失能指定 TWI 的 DMA 请求
TWI_GenerateSTART	产生 TWIx 传输 START 条件
TWI_GenerateSTOP	产生 TWIx 传输 STOP 条件
TWI_AcknowledgeConfig	使能或者失能指定 TWI 的应答功能
TWI_GeneralCallCmd	使能或者失能指定 TWI 的广播呼叫功能
TWI_ITConfig	使能或者失能指定的 TWI 中断
TWI_SendData	通过外设 TWIx 发送一个数据
TWI_ReceiveData	返回通过 TWIx 最近接收的数据
TWI_Send7bitAddress	向指定的从 TWI 设备传送地址字
TWI_StretchClockConfig	使能或者失能指定 TWI 的时钟延长模式
TWI_GetFlagStatus	检查指定的 TWI 标志位设置与否
TWI_ClearFlag	清除 TWIx 的待处理标志位
TWI_SetNbytes	设置 Nbytes 个数
TWI_GetNbytes	获取剩余 Nbytes 个数
TWI_GetStateMachine	获取 TWI 状态机
TWI_PinRemapConfig	TWI 管脚重映射配置

TWI 固件库函数详解

TWI_DeInit

Table 426

函数名	TWI_DeInit
函数原型	void TWI_DeInit(TWI_TypeDef* TWIx)
功能描述	将外设 TWIx 寄存器重设为缺省值
输入参数	TWIx：x 可以是 0 或者 1，来选择 TWI 外设
返回值	无

使用示例:

TWI_DeInit(TWI0); //将外设 TWI0 寄存器重设为缺省值

TWI_Init

Table 427

函数名	TWI_Init
函数原形	void TWI_Init(TWI_TypeDef* TWIx, TWI_InitTypeDef* TWI_InitStruct)
功能描述	根据 TWI_InitStruct 中指定的参数初始化外设 TWIx 寄存器
输入参数 1	TWIx：x 可以是 0 或者 1，来选择 TWI 外设
输入参数 2	TWI_InitStruct：指向结构 TWI_InitTypeDef 的指针，包含了外设 TWI 的配置信息 参阅 Section：TWI_InitTypeDef 查阅更多该参数允许取值范围
返回值	无

TWI_InitTypeDef

TWI_InitTypeDef 定义于文件“sc32f1xxx.h”：

```
typedef struct
{
uint16_t TWI_Ack;
uint16_t TWI_Prescaler
uint16_t TWI_Stretch;
uint16_t TWI_GeneralCall;
uint32_t TWI_SlaveAdress;
} TWI_InitTypeDef;
```

TWI_Ack

TWI的ACK应答功能使能设定，参阅Table 428获得这个参数的所有成员。

Table 428

TWI_Ack	描述
TWI_Ack_Disable	Ack 响应失能
TWI_Ack_Enable	Ack 响应使能

TWI_Prescaler

主机模式下TWI的通信速率设定，参阅Table 429获得这个参数的所有成员。

Table 429

TWI_Prescaler	描述
TWI_PRESCALER_4096	TWI 时钟预分频值为 4096
TWI_PRESCALER_2048	TWI 时钟预分频值为 2048
TWI_PRESCALER_1024	TWI 时钟预分频值为 1024
TWI_PRESCALER_512	TWI 时钟预分频值为 512
TWI_PRESCALER_256	TWI 时钟预分频值为 256
TWI_PRESCALER_128	TWI 时钟预分频值为 128
TWI_PRESCALER_64	TWI 时钟预分频值为 64
TWI_PRESCALER_32	TWI 时钟预分频值为 32
TWI_PRESCALER_16	TWI 时钟预分频值为 16
TWI_PRESCALER_8	TWI 时钟预分频值为 8

TWI_PRESCALER_4	TWI 时钟预分频值为 4
TWI0_PRESCALER_4096	TWI0 时钟预分频值为 4096
TWI0_PRESCALER_2048	TWI0 时钟预分频值为 2048
TWI0_PRESCALER_1024	TWI0 时钟预分频值为 1024
TWI0_PRESCALER_512	TWI0 时钟预分频值为 512
TWI0_PRESCALER_256	TWI0 时钟预分频值为 256
TWI0_PRESCALER_128	TWI0 时钟预分频值为 128
TWI0_PRESCALER_64	TWI0 时钟预分频值为 64
TWI0_PRESCALER_32	TWI0 时钟预分频值为 32
TWI0_PRESCALER_16	TWI0 时钟预分频值为 16
TWI0_PRESCALER_8	TWI0 时钟预分频值为 8
TWI0_PRESCALER_4	TWI0 时钟预分频值为 4
TWI1_Prescaler_1	TWI1 时钟预分频值为 1
TWI1_Prescaler_2	TWI1 时钟预分频值为 2
TWI1_Prescaler_4	TWI1 时钟预分频值为 4
TWI1_Prescaler_8	TWI1 时钟预分频值为 8
TWI1_Prescaler_16	TWI1 时钟预分频值为 16
TWI1_Prescaler_32	TWI1 时钟预分频值为 32
TWI1_Prescaler_64	TWI1 时钟预分频值为 64
TWI1_Prescaler_128	TWI1 时钟预分频值为 128
TWI1_Prescaler_256	TWI1 时钟预分频值为 256
TWI1_Prescaler_512	TWI1 时钟预分频值为 512
TWI1_Prescaler_1024	TWI1 时钟预分频值为 1024
TWI1_Prescaler_2048	TWI1 时钟预分频值为 2048
TWI1_Prescaler_4096	TWI1 时钟预分频值为 4096

SC32f10xx、SC32f11xx、SC32f12xx可选(TWI_PRESCALER_4096- TWI_PRESCALER_4)
SC32f15xx可选(TWI0_PRESCALER_4096- TWI1_Prescaler_4096)

TWI_Stretch

TWI时钟延长使能设定，参阅Table 430获得这个参数的所有成员。

Table 430

TWI_Stretch	描述
TWI_Stretch_Disable	禁止时钟延长
TWI_Stretch_Enable	允许时钟延长，主机需要支持时钟延长功能

TWI_GeneralCall

TWI通用地址响应使能设定，参阅Table 431获得这个参数的所有成员。

Table 431

TWI_Stretch	描述
TWI_GeneralCall_Disable	禁止响应通用地址 00H
TWI_GeneralCall_Enable	允许响应通用地址 00H

TWI_SlaveAdress

TWI从机地址，不能写为全0，00H为通用地址寻址专用。

使用示例:

```
/* 根据 TWI_InitStruct 中指定的参数初始化 TWI0 */
TWI_InitTypeDef TWI_InitStruct;
  TWI_InitStruct.TWI_Ack = TWI_Ack_Enable;
  TWI_InitStruct.TWI_GeneralCall = TWI_GeneralCall_Disable;
  TWI_InitStruct.TWI_SlaveAdress = 0x12;
  TWI_Init(TWI0, &TWI_InitStruct);
```

TWI_StructInit

Table 432

函数名	TWI_StructInit
函数原形	void TWI_StructInit(TWI_InitTypeDef* TWI_InitStruct)
功能描述	把 TWI_InitStruct 中的每一个参数按缺省值填入
输入参数	TWI_InitStruct：指向结构 TWI_InitTypeDef 的指针，待初始化
返回值	无

使用示例:

```
/* 把 TWI_InitStruct 中的每一个参数按缺省值填入 */
TWI_InitTypeDef  TWI_InitStruct;
TWI_StructInit(&TWI_InitStruct);
```

TWI_Cmd

Table 433

函数名	TWI_Cmd
函数原形	void TWI_Cmd(TWI_TypeDef* TWIx, FunctionalState NewState)
功能描述	使能或者失能 TWI 外设
输入参数 1	TWIx：x 可以是 0 或者 1，来选择 TWI 外设
输入参数 2	NewState: 外设 TWIx 的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* 使能TWI1 外设 */
TWI_Cmd(TWI1, ENABLE);
```

TWI_DMACmd

Table 434

函数名	TWI_DMACmd
函数原形	void TWI_DMACmd(TWI_TypeDef* TWIx, TWI_DMAReq_TypeDef TWI_DMAReq, FunctionalState NewState)
功能描述	使能或者失能指定 TWI 的 DMA 请求
输入参数 1	TWIx：x 可以是 0 或者 1，来选择 TWI 外设
输入参数 2	TWI_DMAReq：来选择 TWI DMA 请求
输入参数 3	NewState: TWIx DMA 传输的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

TWI_DMAReq

TWI DMA源选择，参阅Table 435获得这个参数的所有成员。

Table 435

TWI_DMAReq	描述
TWI_DMAReq_RX	DMA 接收通道
TWI_DMAReq_TX	DMA 发送通道

使用示例:

```
/* 使能TWI1 发送的DMA请求 */
TWI_DMACmd (TWI1, TWI_DMAReq_TX, ENABLE);
```

TWI_GenerateSTART

Table 436

函数名	TWI_GenerateSTART
函数原形	void TWI_GenerateSTART(TWI_TypeDef* TWIx, FunctionalState NewState)
功能描述	产生 TWIx 传输 START 条件
输入参数	TWIx：x 可以是 0 或者 1，来选择 TWI 外设

输入参数 2	NewState: TWIx Satrt 信号的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* TWI1发送起始信号 */
TWI_GenerateSTART (TWI1,ENABLE);
```

TWI_GenerateSTOP

Table 437

函数名	TWI_GenerateSTOP
函数原形	void TWI_GenerateSTOP(TWI_TypeDef* TWIx, FunctionalState NewState)
功能描述	产生 TWIx 传输 STOP 条件
输入参数 1	TWIx：x 可以是 0 或者 1，来选择 TWI 外设
输入参数 2	NewState: TWIx Stop 信号的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* TWI1发送STOP信号 */
TWI_GenerateSTOP(TWI1, ENABLE);
```

TWI_AcknowledgeConfig

Table 438

函数名	TWI_AcknowledgeConfig
函数原形	void TWI_AcknowledgeConfig(TWI_TypeDef* TWIx, FunctionalState NewState)
功能描述	使能或者失能指定 TWI 的应答功能
输入参数 1	TWIx：x 可以是 0 或者 1，来选择 TWI 外设
输入参数 2	NewState: TWIx 应答的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* 使能TWI1的应答功能 */
TWI_AcknowledgeConfig (TWI1, ENABLE);
```

TWI_StretchClockConfig

Table 439

函数名	TWI_StretchClockConfig
函数原形	void TWI_StretchClockConfig(TWI_TypeDef* TWIx, FunctionalState NewState)
功能描述	使能或者失能指定 TWI 的时钟延长功能
输入参数 1	TWIx：x 可以是 0 或者 1，来选择 TWI 外设
输入参数 2	NewState: TWIx 应答的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* 使能TWI1的时钟延长功能 */
TWI_StretchClockConfig(TWI1, ENABLE);
```

TWI_GeneralCallCmd

Table 440

函数名	TWI_GeneralCallCmd
函数原形	void TWI_GeneralCallCmd(TWI_TypeDef* TWIx, FunctionalState NewState)
功能描述	使能或者失能指定 TWI 的广播呼叫功能
输入参数 1	TWIx：x 可以是 0 或者 1，来选择 TWI 外设

输入参数 2	NewState: TWIx 广播呼叫的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* 使能TWI1 外设的广播呼叫功能 */
TWI_GeneralCallCmd(TWI1, ENABLE);
```

TWI_ITConfig

Table 441

函数名	TWI_ITConfig
函数原形	void TWI_ITConfig(TWI_TypeDef* TWIx, uint16_t TWI_IT, FunctionalState NewState)
功能描述	使能或者失能指定的 TWI 中断
输入参数 1	TWIx：x 可以是 0 或者 1，来选择 TWI 外设
输入参数 2	TWI_IT：待使能或者失能的 TWI 中断源 参阅 Section：TWI_IT 查阅更多该参数允许取值范围
输入参数 3	NewState：TWIx 中断的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

TWI_IT

TWI中断选择，参阅Table 442获得这个参数的所有成员。

Table 442

TWI_IT	描述
TWI_IT_INT	中断请求 CPU 的使能控制

使用示例:

```
/* 使能TWI1外设总中断 */
TWI_ITConfig (TWI1, TWI_IT_INT，ENABLE);
```

TWI_SendData

Table 443

函数名	TWI_SendData
函数原形	void TWI_SendData(TWI_TypeDef* TWIx, uint8_t Data)
功能描述	通过外设 TWIx 发送一个数据
输入参数 1	TWIx：x 可以是 0 或者 1，来选择 TWI 外设
输入参数 2	Data: 待发送的数据
返回值	无

使用示例:

```
/* TWI1发送数据Data1 */
uint8_t Data1 = 0x5A;
TWI_SendData(TWI1, Data1);
```

TWI_ReceiveData

Table 444

函数名	TWI_ReceiveData
函数原形	uint16_t TWI_ReceiveData(TWI_TypeDef* TWIx)
功能描述	返回通过 TWIx 最近接收的数据
输入参数	TWIx：x 可以是 0 或者 1，来选择 TWI 外设
返回值	TWIx 最近接收的数据

使用示例:

```
/* 把TWI1接收的数据赋值给Data1 */
Uint16_t Data1
Data1 = TWI_ReceiveData(TWI1);
```

TWI_Send7bitAddress

Table 445

函数名	TWI_Send7bitAddress
函数原形	void TWI_Send7bitAddress(TWI_TypeDef* TWIx, uint8_t Address, TWI_Command_TypeDef TWI_Command)
功能描述	向指定的从 TWI 设备传送地址字
输入参数 1	TWIx：x 可以是 0 或者 1，来选择 TWI 外设
输入参数 2	Address: 待传输的从 TWI 地址
输入参数 3	TWI_Command：设置指定的 TWI 指令是读指令还是写指令
返回值	无

TWI_Command

TWI主机指令选择，参阅Table 446获得这个参数的所有成员。

Table 446

TWI_Command	描述
TWI_Command_Write	TWI 主机向从机写数据
TWI_Command_Read	TWI 主机向从机读数据

使用示例:

```
/* TWI向从机发送写命令 */
uint8_t Address1 = 0x12;
TWI_Send7bitAddress(TWI1, Address1, TWI_Command_Write);
```

TWI_GetFlagStatus

Table 447

函数名	TWI_GetFlagStatus
函数原形	FlagStatus TWI_GetFlagStatus(TWI_TypeDef* TWIx, TWI_FLAG_TypeDef TWI_FLAG)
功能描述	检查指定的 TWI 标志位设置与否
输入参数 1	TWIx：x 可以是 0 或者 1，来选择 TWI 外设
输入参数 2	TWI_FLAG：待检查的 TWI 标志位 参阅 Section：TWI_FLAG 查阅更多该参数允许取值范围
返回值	TWI_FLAG 的新状态 1

TWI_FLAG

TWI标志位选择，参阅Table 448获得这个参数的所有成员。

Table 448

TWI_FLAG	描述
TWI_FLAG_TWIF	TWI 中断标志位
TWI_FLAG_TXRXnE	TWI 传输完成标志位
TWI_FLAG_GCA	TWI 通用地址响应标志位
TWI_FLAG_MSTR	TWI 主/从机模式标志

使用示例:

```
/* 获得TWI1标志位状态 */
TWI_GetFlagStatus(TWI1, TWI_FLAG_TWIF);
```

TWI_ClearFlag

Table 449

函数名	TWI_ClearFlag
函数原形	void TWI_ClearFlag(TWI_TypeDef* TWIx, TWI_FLAG_TypeDef TWI_FLAG)
功能描述	清除 TWIx 的待处理标志位
输入参数 1	TWIx：x 可以是 0 或者 1，来选择 TWI 外设
输入参数 2	TWI_FLAG：待清除的 TWI 标志位
返回值	无

使用示例:

```
/* 清除TWI1外设标志位状态 */
TWI_ClearFlag(TWI1, TWI_FLAG_TWIF);
```

TWI_SetNbytes

Table 450

函数名	TWI_SetNbytes
函数原形	void TWI_SetNbytes(TWI_TypeDef* TWIx, uint8_t Nbytes)
功能描述	设置 TWIx 的 Nbytes 个数
输入参数 1	TWIx：x 可以是 0 或者 1，来选择 TWI 外设
输入参数 2	Nbytes: Nbytes 个数
返回值	无

使用示例:

```
/* 设置TWI1的Nbytes个数为12 */
TWI_SetNbytes(TWI1, 12);
```

TWI_GetNbytes

Table 451

函数名	TWI_GetNbytes
函数原形	uint8_t TWI_GetNbytes(TWI_TypeDef* TWIx)
功能描述	获取设置 TWIx 的 Nbytes 个数
输入参数 1	TWIx：x 可以是 0 或者 1，来选择 TWI 外设
返回值	Nbytes 个数

使用示例:

```
/* 获取设置TWI1的Nbytes个数 */
uint8_t NbytesNum;
NbytesNum = TWI_GetNbytes(TWI1);
```

TWI_PinRemapConfig

Table 452

函数名	TWI_PinRemapConfig
函数原形	void TWI_PinRemapConfig(TWI_TypeDef* TWIx, TWI_PinRemap_TypeDef TWI_Remap)
功能描述	配置 TWIx 的重映射管脚
输入参数 1	TWIx：x 可以是 0 或者 1，来选择 TWI 外设
输入参数 2	TWI_Remap：TWI 重映射管脚
返回值	无

TWI_PinRemap

TWI 映射管脚选择，参阅Table 453获得这个参数的所有成员。

Table 453

TWI_PinRemap	描述
TWI_PinRemap_Deflaut	默认 TWI 管脚映射
TWI_PinRemap_A	TWI 映射管脚组 A
TWI_PinRemap_B	TWI 映射管脚组 B
TWI_PinRemap_C	TWI 映射管脚组 C
TWI_PinRemap_D	TWI 映射管脚组 D
TWI_PinRemap_E	TWI 映射管脚组 E
TWI_PinRemap_F	TWI 映射管脚组 F
TWI_PinRemap_G	TWI 映射管脚组 G

SC32f10xx、SC32f15xx可选(TWI_PinRemap_Deflaut - TWI_PinRemap_C)
SC32f12xx可选(TWI_PinRemap_Deflaut - TWI_PinRemap_E)
SC32f11xx可选(TWI_PinRemap_Deflaut - TWI_PinRemap_G)

使用示例:

```
/* 把TWI1管脚设置为重映射组A */
TWI_PinRemapConfig(TWI1, TWI_PinRemap_A);
```

TWI_GetStateMachine

Table 454

函数名	TWI_GetStateMachine
函数原形	TWI_StateMachine_TypeDef TWI_GetStateMachine(TWI_TypeDef* TWIx)
功能描述	获取 TWIx 的状态机
输入参数 1	TWIx：x 可以是 0 或者 1，来选择 TWI 外设
返回值	TWIx 的状态机

TWI_StateMachine

TWI 状态机类型，参阅Table 455获得这个参数的所有成员。

Table 455

TWI_StateMachine	描述
TWI_Slave_Idle	从机处于空闲状态
TWI_Slave_ReceivedaAddress	从机正在接收第一帧地址和读写位
TWI_Slave_ReceivedaData	从机接收数据状态
TWI_Slave_SendData	从机发送数据状态
TWI_Slave_ReceivedaUACK	在从机发送数据状态中，当主机回 UACK 时跳转到此状态
TWI_Slave_DisableACK	从机处于发送状态时，将 AA 写 0 会进入此状态
TWI_Slave_AddressError	从机的地址与主机发送的地址不匹配会跳转到此状态
TWI_Master_Idle	主机为空闲状态
TWI_Master_SendAddress	主机发送起始条件或主机正在发送从设备地址
TWI_Master_SendData	主机发送数据
TWI_Master_ReceivedaData	主机接收数据
TWI_Master_ReceivedaUACK	主机发送停止条件或接收到从机的 UACK 信号

使用示例:

```
/* 获取TWI1的状态机 */
TWI_StateMachine_TypeDef StateMachine；
StateMachine = TWI_GetStateMachine(TWI1);
```

综合使用示例（TWI 主机向从机发送 1Byte 数据）

```
FlagStatus TWI0_Flag = RESET;
TWI_InitTypeDef TWI_InitStruct; //定义TWI初始化结构体变量

/* 使能TWI0和TWI1的时钟 */
RCC_APB0Cmd(ENABLE);  //TWI0外设挂载在APB0时钟上
RCC_APB0PeriphClockCmd(RCC_APB0Periph_TWI0, ENABLE);
TWI_StructInit(&TWI_InitStruct);
/* 配置TWI0作为主机 */
TWI_InitStruct.TWI_Ack = TWI_Ack_Disable;
TWI_InitStruct.TWI_Prescaler = TWI_PRESCALER_1024;
/*  根据结构体配置初始化串口 */
TWI_Init(TWI0, &TWI_InitStruct);
/* 使能TWI中断 */
TWI_ITConfig(TWI0, TWI_IT_INT, ENABLE);
NVIC_EnableIRQ(TWI0_IRQn);
/* 使能接收发送口 */
TWI_Cmd(TWI0, ENABLE);

/* 主发从收 */
```

```
TWI_GenerateSTART(TWI0, ENABLE);
while(TWI0_Flag == RESET);       //等待起始信号发送完成
TWI0_Flag = RESET;
TWI_Send7bitAddress(TWI0, 0x12, TWI_Command_Write);
while(TWI0_Flag == RESET);       //等待地址发送完成
TWI0_Flag = RESET;
/* 主机发送地址帧（写），且收到从机的ACK */
if(TWI_GetFlagStatus(TWI0, TWI_FLAG_TXRXnE))
{
  TWI_ClearFlag(TWI0, TWI_FLAG_TXRXnE);

  TWI_SendData(TWI0, 0x34);
  while(TWI0_Flag == RESET);     //等待数据发送完成
  TWI0_Flag = RESET;

  /* 主机发送完数据，且接收到从机的ACK */
  if(TWI_GetFlagStatus(TWI0, TWI_FLAG_TXRXnE))
  {
    TWI_ClearFlag(TWI0, TWI_FLAG_TXRXnE);
  }
}

TWI_GenerateSTOP(TWI0, ENABLE);

void TWI_Communication_TWI0Handler(void)
{
  if(TWI_GetFlagStatus(TWI0, TWI_FLAG_TWIF))
  {
    TWI_ClearFlag(TWI0, TWI_FLAG_TWIF);

    TWI0_Flag = SET;
  }
}
```

21 SPI1_TWI1 固件库函数

赛元SC32F15Gx 系列SPI1和TWI1集成在一起，可配置为主模式或从属模式，支持DMA。主机模式下TWI通信速率默认为最小分频档位（Fpclk/4），SPI1模式可从任意分频。

SPI1_TWI1 寄存器结构

SPI1_TWI1 寄存器结构，SPI1_TWI1_TypeDef，在文件“sc32f15Gx.h”中定义如下：

```
typedef struct
{
  __IO uint32_t SPI1_TWI1_CON;
  __IO uint32_t SPI1_TWI1_STS;
  __IO uint32_t TWI1_ADD;
  __IO uint32_t SPI1_TWI1_DATA;
  __IO uint32_t SPI1_TWI1_IDE;
} SPITWI_TypeDef
```

SPI1_TWI1 **寄存器表格**

Table 456

寄存器	描述
SPI1_TWI1_CON	SPI1_TWI1控制寄存器
SPI1_TWI1_STS	SPI1_TWI1标志位寄存器
TWI1_ADD	TWI1地址寄存器
SPI1_TWI1_DATA	SPI1_TWI1数据寄存器
SPI1_TWI1_IDE	通信口SPI1_TWI1控制寄存器

SPI1_TWI1 固件库函数列

Table 457

函数名	描述
SPI1_TWI1_DeInit	将外设 SPI1_TWI1 寄存器重设为缺省值
SPI1_StructInit	把 SPI1_InitStruct 中的每一个参数按缺省值填入
SPI1_Init	把 SPI1 初始化
SPI1_Cmd	设置 SPI1 使能
SPI1_SetMode	设置 SPI1 的工作模式
SPI1_DataSizeConfig	设置 SPI1 的数据宽度
SPI1_SendData	通过外设 SPI1 发送一个数据
SPI1_ReceiveData	返回通过 SPI1 最近接收的数据
SPI1_ITConfig	使能或者失能指定的 SPI1 中断
SPI1_GetFlagStatus	检查指定的 SPI1 标志位设置与否
SPI1_ClearFlag	清除 SPI1 的待处理标志位
TWI1_StructInit	把 TWI1_InitStruct 中的每一个参数按缺省值填入
TWI1_Init	根据 TWI1_InitStruct 中指定的参数初始化外设 TWI1 寄存器
TWI1_Cmd	使能或者失能 TWI1 外设
TWI1_AcknowledgeConfig	使能或者失能指定 TWI1 的应答功能
TWI1_GeneralCallCmd	使能或者失能指定 TWI1 的广播呼叫功能
TWI1_StretchClockConfig	使能或者失能指定 TWI1 的时钟延长功能
TWI1_SetNbytes	设置 TWI1 的 Nbytes 个数
TWI1_GetNbytes	获取 TWI1 的 Nbytes 个数
TWI1_GenerateSTART	产生 TWI1 传输 START 条件
TWI1_GenerateSTOP	产生 TWI1 传输 STOP 条件

TWI1_Send7bitAddress	向指定的从 TWI1 设备传送地址字
TWI1_SendData	通过外设 TWI1 发送一个数据
TWI1_ReceiveData	返回通过 TWI1 最近接收的数据
TWI1_GetStateMachine	获取 TWI1 的状态机
TWI1_ITConfig	使能或者失能指定的 TWI1 中断
TWI1_GetFlagStatus	检查指定的 TWI1 标志位设置与否
TWI1_ClearFlag	清除 TWI1 的待处理标志位
SPI1_DMACmd	使能或者失能 DMA 触发指定的 SPI1
SPI1_TWI1_PinRemapConfig	配置 SPI1_TWI1 的重映射管脚

SPI1_TWI1 固件库函数详解

SPI1_TWI1_DeInit

Table 458

函数名	SPI1_TWI1_DeInit
函数原型	void SPI1_TWI1_DeInit(SPITWI_TypeDef* SPIx_TWIx)
功能描述	将外设 SPI1_TWI1 寄存器重设为缺省值
输入参数	SPI1_TWI1：可选择 SPI1 或 TWI1 外设
返回值	无

使用示例:

```
SPI1_TWI1_DeInit(SPI1); //将外设 SPI1 寄存器重设为缺省值
```

SPI1_Init

Table 459

函数名	SPI1_Init
函数原形	void SPI1_Init(SPITWI_TypeDef* SPIx, SPI1_InitTypeDef* SPI1_InitStruct)
功能描述	根据 SPI1_InitStruct 中指定的参数初始化外设 SPI1 寄存器
输入参数 1	SPIx：选择 SPI1 外设
输入参数 2	SPI1_InitStruct：指向结构 SPI1_InitTypeDef 的指针，包含了外设 SPI1 的配置信息 参阅 Section：SPI1_InitTypeDef 查阅更多该参数允许取值范围
返回值	无

SPI1_InitStruct

SPI1_InitTypeDef 定义于文件“sc32f1xxx_SPI1_TWI1.h”：

```
typedef struct
{
uint16_t SPI1_Mode;
uint16_t SPI1_DataSize
uint16_t SPI1_CPHA;
uint16_t SPI1_CPOL;
uint32_t SPI1_FirstBit;
uint32_t SPI1_Prescaler;
} SPI1_InitTypeDef ;
```

SPI1_Mode

SPI1工作模式设定，参阅Table 460获得这个参数的所有成员。

Table 460

SPI1_Mode	描述
SPI1_Mode_Slave	SPI1 作为从机
SPI1_Mode_Master	SPI1 作为主机

SPI1_DataSize

SPI1传输宽度设定，参阅Table 461获得这个参数的所有成员。

Table 461

SPI1_DataSize	描述
SPI1_DataSize_8B	SPI1 传输宽度为 8 位
SPI1_DataSize_16B	SPI1 传输宽度为 16 位

SPI1_CPHA

SPI1时钟相位设定，参阅Table 462获得这个参数的所有成员。

Table 462

SPI1_CPHA	描述
SPI1_CPHA_1Edge	SCK 周期的第一沿采集数据
SPI1_CPHA_2Edge	SCK 周期的第二沿采集数据

SPI1_CPOL

SPI1时钟极性设定，参阅Table 463获得这个参数的所有成员。

Table 463

SPI1_CPOL	描述
SPI1_CPOL_Low	SCK 在空闲状态下为低电平
SPI1_CPOL_High	SCK 在空闲状态下为高电平

SPI1_FirstBit

SPI1传送方向设定，参阅Table 464获得这个参数的所有成员。

Table 464

SPI1_FirstBit	描述
SPI1_FirstBit_MSB	MSB 优先发送
SPI1_FirstBit_LSB	LSB 优先发送

SPI1_Prescaler

SPI1时钟预分频，参阅Table 465获得这个参数的所有成员。

Table 465

SPI1_Prescaler	描述
SPI1_TWI1_Prescaler_1	时钟预分频值为 1
SPI1_TWI1_Prescaler_2	时钟预分频值为 2
SPI1_TWI1_Prescaler_4	时钟预分频值为 4
SPI1_TWI1_Prescaler_8	时钟预分频值为 8
SPI1_TWI1_Prescaler_16	时钟预分频值为 16
SPI1_TWI1_Prescaler_32	时钟预分频值为 32
SPI1_TWI1_Prescaler_64	时钟预分频值为 64
SPI1_TWI1_Prescaler_128	时钟预分频值为 128
SPI1_TWI1_Prescaler_256	时钟预分频值为 256
SPI1_TWI1_Prescaler_512	时钟预分频值为 512
SPI1_TWI1_Prescaler_1024	时钟预分频值为 1024
SPI1_TWI1_Prescaler_2048	时钟预分频值为 2048
SPI1_TWI1_Prescaler_4096	时钟预分频值为 4096

使用示例:

```
/* 根据 SPI1_InitStruct 中指定的参数初始化 SPI1 */
SPI1_InitTypeDef   SPI1_InitStruct;
SPI1_InitStruct.SPI1_Mode = SPI1_Mode_Master;  //主机模式
SPI1_InitStruct.SPI1_DataSize = SPI1_DataSize_16B; //16位传输模式
SPI1_InitStruct.SPI1_FirstBit = SPI1_FirstBit_LSB;  //LSB优先发送
SPI1_InitStruct.SPI1_CPHA = SPI1_CPHA_1Edge;  //SCK周期的第一沿采集数据
SPI1_InitStruct.SPI1_CPOL = SPI1_CPOL_Low;  //SCK在空闲状态下为低电平
SPI1_InitStruct.SPI1_Prescaler = SPI1_Prescaler_2;  //时钟频率为Fpclk/2
SPI1_Init(SPI1,&SPI1_InitStruct);
```

SPI1_StructInit

Table 466

函数名	SPI1_StructInit
函数原形	void SPI1_StructInit(SPI1_InitTypeDef* SPI1_InitStruct)
功能描述	把 SPI1_InitStruct 中的每一个参数按缺省值填入
输入参数	SPI1_InitStruct：指向结构 SPI1_InitTypeDef 的指针，待初始化
返回值	无

使用示例:

```
/* 把 SPI1_InitStruct 中的每一个参数按缺省值填入 */
SPI1_InitTypeDef  SPI1_InitStruct;
SPI1_InitStruct = SPI1_StructInit(&SPI1_InitStruct);
```

SPI1_Cmd

Table 467

函数名	SPI1_Cmd
函数原形	void SPI1_Cmd(SPITWI_TypeDef* SPIx, FunctionalState NewState)
功能描述	使能或者失能 SPI1 外设
输入参数 1	SPIx：选择 SPI1 外设
输入参数 2	NewState: 外设 SPI1 的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* 使能SPI1 外设 */
SPI1_Cmd(SPI1, ENABLE);
\
```

SPI1_SetMode

Table 468

函数名	SPI1_SetMode
函数原形	void SPI1_SetMode(SPITWI_TypeDef* SPIx, SPI1_Mode_TypeDef SPI1_Mode)
功能描述	设置 SPI1 的工作模式
输入参数 1	SPIx：选择 SPI1 外设
输入参数 2	SPI1_Mode: 工作模式
返回值	无

使用示例:

```
/* 设置SPI1为主机 */
SPI1_SetMode(SPI1,SPI1_Mode_Maste);
```

SPI1_DataSizeConfig

Table 469

函数名	SPI1_DataSizeConfig
函数原形	void SPI1_DataSizeConfig(SPITWI_TypeDef* SPIx, SPI1_DataSize_TypeDef SPI1_DataSize)
功能描述	设置 SPIx 的数据宽度
输入参数 1	SPIx：选择 SPI1 外设
输入参数 2	SPI1_DataSize: 传输数据宽度
返回值	无

使用示例:

```
/* 设置SPI1数据传输宽度为8bit */
SPI1_DataSizeConfig(SPI1,SPI1_DataSize_8B);
```

SPI1_SendData

Table 470

函数名	SPI1_SendData

函数原形	void SPI1_SendData(SPITWI_TypeDef* SPIx, uint16_t Data)
功能描述	通过外设 SPI1 发送一个数据
输入参数 1	SPIx：来选择 SPI1 外设
输入参数 2	Data: 待发送的数据
返回值	无

使用示例:

```
/* SPI1发送数据0x12 */
SPI1_SendData(SPI1,0x12);
```

SPI1_ReceiveData

Table 471

函数名	SPI1_ReceiveData
函数原形	uint16_t SPI1_ReceiveData(SPITWI_TypeDef* SPIx)
功能描述	返回通过 SPI1 最近接收的数据
输入参数	SPIx：选择 SPI1 外设
返回值	返回 SPI1 接收的数据

使用示例:

```
/* 把SPI1接收的数据赋值给Data1 */
uint16_t Data1
Data1 = SPI1_ReceiveData(SPI1);
```

SPI1_ITConfig

Table 472

函数名	SPI1_ITConfig
函数原形	void SPI1_ITConfig(SPITWI_TypeDef* SPIx, uint16_t SPI1_IT, FunctionalState NewState)
功能描述	使能或者失能指定的 SPI1 中断
输入参数 1	SPIx：选择 SPI1 外设
输入参数 2	SPI1_IT：待使能或者失能的 SPI1 中断源
输入参数 3	NewState：SPI1 中断的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

SPI1_IT

SPI1_IT请求选择，参阅Table 473获得这个参数的所有成员。

Table 473

SPI1_IT	描述
SPI1_IT_INTEN	中断请求 CPU 的屏蔽位
SPI1_IT_TB	发送缓存器为空时的中断使能

使用示例:

```
/* 使能SPI1 外设总中断 */
SPI1_ITConfig (SPI1, SPI1_IT_INTEN，ENABLE);
```

SPI1_GetFlagStatus

Table 474

函数名	SPI1_GetFlagStatus
函数原形	FlagStatus SPI1_GetFlagStatus(SPITWI_TypeDef* SPIx, SPI1_FLAG_TypeDef SPI1_FLAG)
功能描述	检查指定的 SPI1 标志位设置与否
输入参数 1	SPIx：选择 SPI1 外设
输入参数 2	SPI1_FLAG：待检查的 SPI1 标志位
返回值	SPI1_FLAG 的新状态

SPI1_FLAG

SPI1 中断标志位选择，参阅Table 475获得这个参数的所有成员。

Table 475

SPI1_FLAG	描述
SPI1_Flag_TXEIF	发送缓存为空标志位
SPI1_Flag_WCOL	写入冲突标志位

使用示例:

```
/* 获得SPI1发送缓存为空标志位 */
SPI1_GetFlagStatus(SPI1, SPI1_Flag_TXEIF);
```

SPI1_ClearFlag

Table 476

函数名	SPI1_ClearFlag
函数原形	void SPI1_ClearFlag(SPITWI_TypeDef* SPIx, uint32_t SPI1_FLAG)
功能描述	清除 SPI1 的待处理标志位
输入参数 1	SPIx：选择 SPI1 外设
输入参数 2	SPI1_FLAG：待清除的 SPI1 标志位
返回值	无

使用示例:

```
/* 清除SPI1外设标志位 */
SPI1_ClearFlag(SPI1, SPI1_Flag_SPI1F);
```

TWI1_StructInit

Table 477

函数名	TWI1_StructInit
函数原形	void TWI1_StructInit(TWI1_InitTypeDef* TWI1_InitStruct)
功能描述	把 TWI1_InitStruct 中的每一个参数按缺省值填入
输入参数	TWI1_InitStruct：指向结构 TWI1_InitTypeDef 的指针，待初始化
返回值	无

使用示例:

```
/* 把 TWI1_InitStruct 中的每一个参数按缺省值填入 */
TWI1_InitTypeDef  TWI1_InitStruct;
TWI1_StructInit(&TWI1_InitStruct);
```

TWI1_Init

Table 478

函数名	TWI1_Init
函数原形	void TWI1_Init(SPITWI_TypeDef* TWIx, TWI1_InitTypeDef* TWI_InitStruct)
功能描述	根据 TWI1_InitStruct 中指定的参数初始化外设 TWIx 寄存器
输入参数 1	TWIx：选择 TWI1 外设
输入参数 2	TWI1_InitStruct：指向结构 TWI1_InitTypeDef 的指针，包含了外设 TWI1 的配置信息 参阅 Section：TWI1_InitTypeDef 查阅更多该参数允许取值范围
返回值	无

TWI1_InitTypeDef

TWI1_InitTypeDef 定义于文件“SC32F1xxx_SPI1_TWI1.h”：

```
typedef struct
{
uint16_t TWI1_Ack;
uint16_t TWI1_Prescaler
uint16_t TWI1_Stretch;
uint16_t TWI1_GeneralCall;
uint32_t TWI1_SlaveAdress;
} TWI1_InitTypeDef;
```

TWI1_Ack

TWI1的ACK应答功能使能设定，参阅Table 479获得这个参数的所有成员。

Table 479

TWI1_Ack	描述
TWI1_Ack_Disable	Ack 响应失能
TWI1_Ack_Enable	Ack 响应使能

TWI1_Prescaler

主机模式下TWI1的通信速率设定，参阅Table 480获得这个参数的所有成员。

Table 480

TWI1_Prescaler	描述
SPI1_TWI1_Prescaler_1	时钟预分频值为 1
SPI1_TWI1_Prescaler_2	时钟预分频值为 2
SPI1_TWI1_Prescaler_4	时钟预分频值为 4
SPI1_TWI1_Prescaler_8	时钟预分频值为 8
SPI1_TWI1_Prescaler_16	时钟预分频值为 16
SPI1_TWI1_Prescaler_32	时钟预分频值为 32
SPI1_TWI1_Prescaler_64	时钟预分频值为 64
SPI1_TWI1_Prescaler_128	时钟预分频值为 128
SPI1_TWI1_Prescaler_256	时钟预分频值为 256
SPI1_TWI1_Prescaler_512	时钟预分频值为 512
SPI1_TWI1_Prescaler_1024	时钟预分频值为 1024

TWI1_Stretch

TWI1时钟延长使能设定，参阅Table 481获得这个参数的所有成员。

Table 481

TWI1_Stretch	描述
TWI1_Stretch_Disable	禁止时钟延长
TWI1_Stretch_Enable	允许时钟延长，主机需要支持时钟延长功能

TWI1_GeneralCall

TWI1通用地址响应使能设定，参阅Table 482获得这个参数的所有成员。

Table 482

TWI1_Stretch	描述
TWI1_GeneralCall_Disable	禁止响应通用地址 00H
TWI1_GeneralCall_Enable	允许响应通用地址 00H

TWI1_SlaveAdress

TWI1从机地址，不能写为全0，00H为通用地址寻址专用。

使用示例:

```
/* 根据 TWI1_InitStruct 中指定的参数初始化 TWI1 */
TWI1_InitTypeDef TWI1_InitStruct;
  TWI1_InitStruct.TWI1_Ack = TWI1_Ack_Enable;
  TWI1_InitStruct.TWI1_GeneralCall = TWI1_GeneralCall_Disable;
  TWI1_InitStruct.TWI1_SlaveAdress = 0x12;
  TWI1_Init(TWI0, &TWI1_InitStruct);
```

TWI1_Cmd

Table 483

函数名	TWI1_Cmd
函数原形	void TWI1_Cmd(SPITWI_TypeDef* TWIx, FunctionalState NewState)
功能描述	使能或者失能 TWI1 外设
输入参数 1	TWIx：选择 TWI1 外设
输入参数 2	NewState: 外设 TWI1 的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* 使能TWI1 外设 */
TWI1_Cmd(TWI1, ENABLE);
```

TWI1_GenerateSTART

Table 484

函数名	TWI1_GenerateSTART
函数原形	void TWI1_GeneralCallCmd(SPITWI_TypeDef* TWIx, FunctionalState NewState)
功能描述	产生 TWI1 传输 START 条件
输入参数	TWIx：选择 TWI1 外设
输入参数 2	NewState: TWI1 Satrt 信号的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* TWI1发送起始信号 */
TWI1_GenerateSTART (TWI1,ENABLE);
```

TWI1_GenerateSTOP

Table 485

函数名	TWI1_GenerateSTOP
函数原形	void TWI1_GenerateSTOP(SPITWI_TypeDef * TWIx, FunctionalState NewState)
功能描述	产生 TWI1 传输 STOP 条件
输入参数 1	TWIx：选择 TWI1 外设
输入参数 2	NewState: TWI1 Stop 信号的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* TWI1发送STOP信号 */
TWI1_GenerateSTOP(TWI1, ENABLE);
```

TWI1_AcknowledgeConfig

Table 486

函数名	TWI1_AcknowledgeConfig
函数原形	void TWI1_AcknowledgeConfig(SPITWI_TypeDef * TWIx, FunctionalState NewState)
功能描述	使能或者失能指定 TWI1 的应答功能
输入参数 1	TWIx：选择 TWI1 外设
输入参数 2	NewState: TWI1 应答的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* 使能TWI1的应答功能 */
TWI1_AcknowledgeConfig (TWI1, ENABLE);
```

TWI1_GeneralCallCmd

Table 487

函数名	TWI1_GeneralCallCmd
函数原形	void TWI1_GeneralCallCmd(SPITWI_TypeDef * TWIx, FunctionalState NewState)
功能描述	使能或者失能指定 TWI1 的广播呼叫功能
输入参数 1	TWIx：选择 TWI1 外设
输入参数 2	NewState: TWI1 广播呼叫的新状态 这个参数可以取：ENABLE 或者 DISABLE

返回值	无

使用示例:

```
/* 使能TWI1 外设的广播呼叫功能 */
TWI1_GeneralCallCmd(TWI1, ENABLE);
```

TWI1_StretchClockConfig

Table 488

函数名	TWI1_StretchClockConfig
函数原形	void TWI1_StretchClockConfig(SPITWI_TypeDef * TWIx, FunctionalState NewState)
功能描述	使能或者失能指定 TWI1 的时钟延长功能
输入参数 1	TWIx：选择 TWI1 外设
输入参数 2	NewState: TWI1 应答的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* 使能TWI1的时钟延长功能 */
TWI1_StretchClockConfig(TWI1, ENABLE);
```

TWI1_GeneralCallCmd

Table 489

函数名	TWI1_GeneralCallCmd
函数原形	void TWI1_GeneralCallCmd(SPITWI_TypeDef * TWIx, FunctionalState NewState)
功能描述	使能或者失能指定 TWI1 的广播呼叫功能
输入参数 1	TWIx：选择 TWI1 外设
输入参数 2	NewState: TWI1 广播呼叫的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* 使能TWI1 外设的广播呼叫功能 */
TWI0_GeneralCallCmd(TWI1, ENABLE);
```

TWI1_ITConfig

Table 490

函数名	TWI1_ITConfig
函数原形	void TWI1_ITConfig(SPITWI_TypeDef * TWIx, uint16_t TWI1_IT, FunctionalState NewState)
功能描述	使能或者失能指定的 TWI1 中断
输入参数 1	TWIx：选择 TWI1 外设
输入参数 2	TWI1_IT：待使能或者失能的 TWI1 中断源 参阅 Section：TWI1_IT 查阅更多该参数允许取值范围
输入参数 3	NewState：TWI1 中断的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

TWI1_IT

TWI1中断选择，参阅Table 491获得这个参数的所有成员。

Table 491

TWI1_IT	描述
TWI1_IT_INT	中断请求 CPU 的使能控制

使用示例:

```
/* 使能TWI1外设总中断 */
TWI1_ITConfig (TWI1, TWI1_IT_INT，ENABLE);
```

TWI1_SendData

Table 492

函数名	TWI1_SendData
函数原形	void TWI1_SendData(SPITWI_TypeDef * TWIx, uint8_t Data)
功能描述	通过外设 TWI1 发送一个数据
输入参数 1	TWIx：选择 TWI1 外设
输入参数 2	Data: 待发送的数据
返回值	无

使用示例:

```
/* TWI1发送数据Data1 */
uint8_t Data1 = 0x5A;
TWI0_SendData(TWI1, Data1);
```

TWI1_ReceiveData

Table 493

函数名	TWI1_ReceiveData
函数原形	uint16_t TWI1_ReceiveData(SPITWI_TypeDef * TWIx)
功能描述	返回通过 TWI1 最近接收的数据
输入参数	TWIx：选择 TWI1 外设
返回值	TWIx 最近接收的数据

使用示例:

```
/* 把TWI1接收的数据赋值给Data1 */
Uint16_t Data1
Data1 = TWI1_ReceiveData(TWI1);
```

TWI1_Send7bitAddress

Table 494

函数名	TWI1_Send7bitAddress
函数原形	void TWI1_Send7bitAddress(SPITWI_TypeDef * TWIx, uint8_t Address, TWI1_Command_TypeDef TWI1_Command)
功能描述	向指定的从 TWI1 设备传送地址字
输入参数 1	TWIx：选择 TWI1 外设
输入参数 2	Address: 待传输的从 TWI1 地址
输入参数 3	TWI1_Command：设置指定的 TWI1 指令是读指令还是写指令
返回值	无

TWI1_Command

TWI1主机指令选择，参阅Table 495获得这个参数的所有成员。

Table 495

TWI1_Command	描述
TWI1_Command_Write	TWI1 主机向从机写数据
TWI1_Command_Read	TWI1 主机向从机读数据

使用示例:

```
/* TWI1向从机发送写命令 */
uint8_t Address1 = 0x12;
TWI1_Send7bitAddress(TWI1, Address1, TWI1_Command_Write);
```

TWI1_GetFlagStatus

Table 496

函数名	TWI1_GetFlagStatus
函数原形	FlagStatus TWI1_GetFlagStatus(SPITWI_TypeDef * TWIx, TWI1_FLAG_TypeDef TWI1_FLAG)
功能描述	检查指定的 TWI1 标志位设置与否
输入参数 1	TWIx：选择 TWI1 外设

输入参数 2	TWI1_FLAG：待检查的 TWI1 标志位 参阅 Section：TWI1_FLAG 查阅更多该参数允许取值范围
返回值	TWI1_FLAG 的新状态

TWI1_FLAG

TWI1标志位选择，参阅Table 497获得这个参数的所有成员。

Table 497

TWI1_FLAG	描述
TWI1_FLAG_QTWIF	TWI1 中断标志位
TWI1_FLAG_TXRXnE	TWI1 传输完成标志位
TWI1_FLAG_GCA	TWI1 通用地址响应标志位

使用示例:

```
/* 获得TWI1标志位状态 */
TWI0_GetFlagStatus(TWI1, TWI1_FLAG_TWI0F);
```

TWI1_ClearFlag

Table 498

函数名	TWI1_ClearFlag
函数原形	void TWI1_ClearFlag(SPITWI_TypeDef * TWIx, TWI1_FLAG_TypeDef TWI1_FLAG)
功能描述	清除 TWI1 的待处理标志位
输入参数 1	TWIx：选择 TWI1 外设
输入参数 2	TWI1_FLAG：待清除的 TWI1 标志位
返回值	无

使用示例:

```
/* 清除TWI1外设标志位状态 */
TWI1_ClearFlag(TWI1, TWI1_FLAG_TWI1F);
```

TWI1_SetNbytes

Table 499

函数名	TWI1_SetNbytes
函数原形	void TWI1_SetNbytes(SPITWI_TypeDef * TWIx, uint8_t Nbytes)
功能描述	设置 TWI1 的 Nbytes 个数
输入参数 1	TWIx：选择 TWI1 外设
输入参数 2	Nbytes: Nbytes 个数
返回值	无

使用示例:

```
/* 设置TWI1的Nbytes个数为12 */
TWI1_SetNbytes(TWI1, 12);
```

TWI1_GetNbytes

Table 500

函数名	TWI1_GetNbytes
函数原形	uint8_t TWI1_GetNbytes(SPITWI_TypeDef * TWIx)
功能描述	获取设置 TWIx 的 Nbytes 个数
输入参数 1	TWIx：选择 TWI1 外设
返回值	Nbytes 个数

使用示例:

```
/* 获取设置TWI1的Nbytes个数 */
uint8_t NbytesNum;
NbytesNum = TWI1_GetNbytes(TWI1);
```

SPI1_DMACmd

Table 501

函数名	SPI1_DMACmd
函数原形	void SPI1_DMACmd(SPITWI_TypeDef* SPIx, SPI1_DMAReq_TypeDef SPI1_DMAReq, FunctionalState NewState)
功能描述	使能或者失能 DMA 触发指定的 SPI1_TWI1
输入参数 1	SPIx：选择 SPI1 外设
输入参数 2	SPI1_DMAReq：指定 DMA 请求
输入参数 3	NewState：SPI1/TWI1 DMA 触发中断的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

SPITWI_DMAReq

SPITWI_DMAReq指定 DMA 请求，参阅Table 502获得这个参数的所有成员。

Table 502

SPI1TWI1_DMAReq	描述
SPI1_DMAReq_RX	DMA 中断请求 SPI1 接受使能控制
SPI1_DMAReq_TX	DMA 中断请求 SPI1 发送使能控制

使用示例:

```
/* 使能DMA中断请求SPI1接受使能控制 */
SPI1 _DMACmd (SPI1, SPI1 _DMAReq_RX，ENABLE);
```

SPI1_TWI1_PinRemapConfig

Table 503

函数名	SPI1_TWI1_PinRemapConfig
函数原形	void SPI1_TWI1_PinRemapConfig(SPITWI_TypeDef * SPIx_TWIx, SPI1_TWI1_PinRemap_TypeDef SPI1_TWI1_Remap)
功能描述	配置 SPI1_TWI1 的重映射管脚
输入参数 1	SPIx_TWIx：来选择 SPI1/TWI1 外设
输入参数 2	SPI1_TWI1_Remap：SPI1/TWI1 重映射管脚
返回值	无

TWI1_PinRemap

TWI1 映射管脚选择，参阅Table 504获得这个参数的所有成员。

Table 504

SPI1_TWI1_PinRemap	描述
SPI1_TWI1_PinRemap_Deflaut	默认 SPI1/TWI1 管脚映射
SPI1_TWI1_PinRemap_A	SPI1/TWI1 映射管脚组 A
SPI1_TWI1_PinRemap_B	SPI1/TWI1 映射管脚组 B
SPI1_TWI1_PinRemap_C	SPI1/TWI1 映射管脚组 C

使用示例:

```
/* 把TWI1管脚设置为重映射组A */
SPI1_TWI1_PinRemapConfig (TWI1, SPI1_TWI1_PinRemap_A);
```

TWI1_GetStateMachine

Table 505

函数名	TWI1_GetStateMachine
函数原形	TWI1_StateMachine_TypeDef TWI1_GetStateMachine(SPITWI_TypeDef * TWIx)
功能描述	获取 TWI1 的状态机
输入参数 1	TWIx：选择 TWI1 外设
返回值	TWIx 的状态机

TWI1_StateMachine

TWI1 状态机类型，参阅Table 506获得这个参数的所有成员。

Table 506

TWI1_StateMachine	描述
TWI1_Slave_Idle	从机处于空闲状态
TWI1_Slave_ReceivedaAddress	从机正在接收第一帧地址和读写位
TWI1_Slave_ReceivedaData	从机接收数据状态
TWI1_Slave_SendData	从机发送数据状态
TWI1_Master_Idle	主机为空闲状态
TWI1_Master_SendAddress	主机发送起始条件或主机正在发送从设备地址
TWI1_Master_SendData	主机发送数据
TWI1_Master_ReceivedaData	主机接收数据

使用示例:

```
/* 获取TWI1的状态机 */
TWI1_StateMachine_TypeDef StateMachine；
StateMachine = TWI1_GetStateMachine(TWI1);
```

综合使用示例（以配置 SPI1 为从机模式为例）

```
static volatile FlagStatus SPI1_Flag = RESET;
const uint8_t Slave_SendData[8] = {0x08,0x09,0x10,0x11,0x12,0x13,0x14,0x15};

void SPI_Communication_Function(void)
{

  GPIO_InitTypeDef GPIO_InitStruct; //定义GPIO初始化结构体变量
  volatile uint8_t Slave_ReceiveData[8];
  volatile uint8_t SPI1_Data, SPI1_Num = 0;
  GPIO_InitStruct.GPIO_Pin = GPIO_Pin_2;
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT_PP;
  GPIO_Init(GPIOA, &GPIO_InitStruct);

  /* 使能SPI1的时钟 */
  RCC_APB1Cmd(ENABLE);//SPI1外设挂载在APB1时钟上
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_SPI1_TWI1, ENABLE);

  /* 配置SPI1为从机 */
  SPI1_InitTypeDef SPI1_InitStruct;              //定义SPI0初始化结构体变量
  SPI1_InitStruct.SPI1_CPHA = SPI1_CPHA_1Edge;             //SPI第一沿采集数据
  SPI1_InitStruct.SPI1_CPOL = SPI1_CPOL_High;     //SPI空闲电平为高电平
  SPI1_InitStruct.SPI1_DataSize = SPI1_DataSize_8B;       //数据长度为8位
  SPI1_InitStruct.SPI1_FirstBit = SPI1_FirstBit_MSB;//数值传输为高位在前
  SPI1_InitStruct.SPI1_Prescaler = SPI1_TWI1_Prescaler_1024;//SPI传输频率为APB1时钟1024分频
  SPI1_InitStruct.SPI1_Mode = SPI1_Mode_Slave;     //SPI工作在从机模式
  SPI1_Init(SPI1, &SPI1_InitStruct);

  /* 使能SPI1中断 */
  SPI1_ITConfig(SPI1, SPI1_IT_INT, ENABLE);
  NVIC_EnableIRQ(SPI1_TWI1_IRQn);
  /* 使能SPI1 */
  SPI1_Cmd(SPI1, ENABLE);

  while(1)
  {
    GPIO_SetBits(GPIOA, GPIO_Pin_2);
   //SPI1发送第一个数据
   SPI1_SendData(SPI1,Slave_SendData[SPI1_Num]);
```

```
    while(SPI1_Num < 7)
    {
      while(SPI1_Flag == RESET);
      SPI1_Flag = RESET;
      Slave_ReceiveData[SPI1_Num] = SPI1_ReceiveData(SPI1);//从机SPI1接收数据
      SPI1_Num ++;
      SPI1_SendData(SPI1, Slave_SendData[SPI1_Num]);       //从机SPI1发送数据
    }

    while(SPI1_Flag == RESET);
    Slave_ReceiveData[SPI1_Num] = SPI1_ReceiveData(SPI1);       //SPI1接收最后一个数据
    for(int i=0 ;i <= 7 ;i++)      printf("SPI1从机接收到的数据为“0x%x” \r\n",Slave_ReceiveData[i]);

    /* 清除标志位，为下一次通信做准备 */
    SPI1_Num = 0;
    SPI1_Flag = RESET;

  }
}


void SPI_Communication_SPI1Handler(void)
{

  if(SPI1_GetFlagStatus(SPI1, SPI1_FLAG_QTWIF))
  {
    SPI1_ClearFlag(SPI1, SPI1_FLAG_QTWIF);
    SPI1_Flag = SET;
  }
}
```

22 UART 固件库函数

SC32F1xxx 系列有多个UART，具有四种通信模式可选，支持从STOP模式唤醒。UART0和UART1可产生DMA请求。

UART 寄存器结构

UART 寄存器结构，UART_TypeDef，在文件“sc32f1xxx.h”中定义如下：

```
typedef struct
{
  __IO uint32_t UART_CON;
  __IO uint32_t UART_STS;
  __IO uint32_t UART_BAUD;
  __IO uint32_t UART_DATA;
  __IO uint32_t UART_IDE;
} UART_TypeDef;
```

UART 寄存器**表格**

Table 507

寄存器	描述
UART_CON	UART控制寄存器
UART_STS	UART标志位寄存器
UART_BAUD	UART波特率寄存器
UART_DATA	UART数据寄存器
UART_IDE	通信口DMA控制寄存器

UART 固件库函数列表

Table 508

函数名	描述
UART_DeInit	将 UARTx 寄存器设置为缺省值
UART_Init	根据 UART_InitStruct 中指定的参数初始化外设 UARTx 寄存器
UART_TXCmd	使能或者失能UARTx发送数据
UART_RXCmd	使能或者失能UARTx接收数据
UART_SendData	通过UARTx发送数据
UART_ReceiveData	通过UARTx接收数据
UART_PinRemapConfig	配置UARTx 管脚的重映射
UART_ITConfig	使能或者失能指定的 UART 中断
UART_GetFlagStatus	检查指定的UART 标志位设置与否
UART_ClearFlag	清除UARTx 的待处理标志位
UART_DMACmd	使能或者失能指定的UARTx的DMA请求
UART_LIN_MODE	配置工作模式为LIN模式
UART_LIN_BKSIZE	配置同步间隔段
UART_SendBreak	发送同步间隔段
UART_LIN_SLVARENE	使能LIN自动重同步
UART_LIN_LBDL	配置LIN断路检测长度
LIN_CalID	获取PID
LINCalChecksum	获取LIN校验值

UART 固件库函数详解

UART_DeInit

Table 509

函数名	UART_DeInit
函数原型	void UART_DeInit(UART_TypeDef* UARTx)
功能描述	将 UARTx 寄存器设置为缺省值
输入参数	UARTx：来选择 UART 外设（x 具体以规格书为准）
返回值	无

使用示例:

```
/* UART0寄存器复位 */
UART0_DeInit(UART0);
```

UART_Init

Table 510

函数名	UART_Init
函数原型	void UART_Init(UART_TypeDef* UARTx, UART_InitTypeDef* UART_InitStruct)
功能描述	根据 UART_InitStruct 中指定的参数初始化外设 UARTx 寄存器
输入参数 1	UARTx：来选择 UART 外设（x 具体以规格书为准）
输入参数 2	UART_InitStruct：指向结构 UART_InitTypeDef 的指针，包含了外设 UART 的配置信息
返回值	无

UART_InitTypeDef

UART_InitTypeDef 定义于文件“sc32f1xxx_uart.h”：

```
typedef struct
{
uint32_t UART_ClockFrequency;
uint32_t UART_Mode;
uint32_t UART_BaudRate;
} UART_InitTypeDef;
```

UART_ClockFrequency

Uart外设的工作频率。

UART_Mode

UART的工作模式，参阅Table 511获得这个参数的所有成员。

Table 511

UART_Mode	描述
UART_Mode_8B	8 位半双工同步通信模式
UART_Mode_10B	10 位全双工异步通信模式
UART_Mode_11B	11 位全双工异步通信模式

UART_BaudRate

在10位或11位全双工异步通信工作模式下，波特率就为写入值。在8位半双工同步通信模式，参数可以是的Table 512中的成员。

Table 512

UART_BaudRate	描述
UART_PRESCALER_12	UART 挂载 APB 总线时钟的 12 分频
UART_PRESCALER_4	UART 挂载 APB 总线时钟的 4 分频

使用示例:

```
/* 根据UART_InitStruct 中指定的参数初始化UART1 */
UART_InitTypeDef UART_InitStruct;
UART_InitStruct.UART_BaudRate = 115200;  //波特率为115200
UART_InitStruct.UART_ClockFrequency = 3200000; //时钟频率为32MHz
UART_InitStruct.UART_Mode = UART_Mode_10B;  //工作模式为十位全双工异步通信
UART_Init(UART1, &UART_InitStruct);
```

UART_TXCmd

Table 513

函数名	UART_TXCmd
函数原型	void UART_TXCmd(UART_TypeDef* UARTx, FunctionalState NewState)
功能描述	使能或者失能 UARTx 发送数据
输入参数 1	UARTx：来选择 UART 外设（x 具体以规格书为准）
输入参数 2	NewState：UARTx 发送数据功能开启/关闭
返回值	无

使用示例:

```
/* 使能UART1发送数据 */
UART_TXCmd(UART1,ENABLE);
```

UART_RXCmd

Table 514

函数名	UART_RXCmd
函数原型	void UART_RXCmd(UART_TypeDef* UARTx, FunctionalState NewState)
功能描述	使能或者失能 UARTx 接收数据
输入参数 1	UARTx：来选择 UART 外设（x 具体以规格书为准）
输入参数 2	NewState：UARTx 接收数据功能开启/关闭
返回值	无

使用示例:

```
/* 使能UART1接收数据 */
UART_RXCmd(UART1,ENABLE);
```

UART_SendData

Table 515

函数名	UART_SendData
函数原型	void UART_SendData(UART_TypeDef* UARTx, uint16_t Data)
功能描述	通过 UARTx 发送数据
输入参数 1	UARTx：来选择 UART 外设（x 具体以规格书为准）
输入参数 2	Data：UARTx 发送的数据，8 位或者 9 位(9 位只对模式 3 有效)
返回值	无

使用示例:

```
/* UART1发送8位数据0x55 */
UART_SendData(UART1,0x55);
```

UART_ReceiveData

Table 516

函数名	UART_ReceiveData
函数原型	uint16_t UART_ReceiveData(UART_TypeDef* UARTx)
功能描述	通过 UARTx 接收数据
输入参数	UARTx：来选择 UART 外设（x 具体以规格书为准）
返回值	UARTx 接收到的数据

使用示例:

```
/* UART1接收数据 */
uint16_t  UART_RData;
UART_RData = UART_ReceiveData(UART1);
```

UART_PinRemapConfig

Table 517

函数名	UART_PinRemapConfig

函数原型	void UART_PinRemapConfig(UART_TypeDef* UARTx, UART_PinRemap_TypeDef UART_Remap)
功能描述	配置 UARTx 管脚的重映射
输入参数 1	UARTx：来选择 UART 外设（x 具体以规格书为准）
输入参数 2	UART_Remap：UARTx 管脚的选择
返回值	无

Table 518给出了UART_PinRemap的值。

Table 518

UART_PinRemap	描述
UART_PinRemap_Default	UART 信号口映射组默认
UART_PinRemap_A	UART 信号口映射组 A

使用示例:

```
/* 配置 UART1 管脚的重映射 */
UART_PinRemapConfig(UART1, UART_PinRemap_A);
```

UART_ITConfig

Table 519

函数名	UART_ITConfig
函数原型	void UART_ITConfig(UART_TypeDef* UARTx, uint16_t UART_IT, FunctionalState NewState)
功能描述	使能或者失能指定的 UART 中断
输入参数 1	UARTx：来选择 UART 外设（x 具体以规格书为准）
输入参数 2	UART_IT：UARTx 中断请求，定义在 UART_IT_TypeDef 中
输入参数 3	NewState：UARTx 中断开启/关闭
返回值	无

UART_IT

Table 520给出了所有UART_IT的取值列表。

Table 520

UART_IT	描述
UART_IT_EN	中断请求 CPU 的屏蔽位
UART_IT_TX	发送中断使能控制位
UART_IT_RX	接收中断使能控制位
UART_IT_BK	断开中断控制位
UART_IT_SL	LIN 从机报头错误中断使能
UART_IT_SY	波特率同步完成中断使能位

使用示例:

```
/* 使能UART1接收中断 */
UART_ITConfig(UART1, UART_IT_EN | UART_IT_RX, ENABLE);
```

UART_GetFlagStatus

Table 521

函数名	UART_GetFlagStatus
函数原型	FlagStatus UART_GetFlagStatus(UART_TypeDef* UARTx, UART_FLAG_TypeDef UART_FLAG)
功能描述	检查指定的 UART 标志位设置与否
输入参数 1	UARTx：来选择 UART 外设（x 具体以规格书为准）
输入参数 2	UART_FLAG：UARTx 的标志位
返回值	UART 标志位状态

UART_FLAG

UART标志位选择，参阅Table 522获得这个参数的所有成员。

Table 522

UART_ FLAG	描述

UART_Flag_RX	接收中断标志位
UART_Flag_TX	发送中断标志位
UART_Flag_BK	断开中断标志位
UART_Flag_SY	波特率同步完成标志位
UART_Flag_SLVHE	LIN 从机报头错误标志
UART_Flag_SLVYN	LIN 同步状态位

使用示例:

```
/* 检查UART1接收数据标志位设置与否 */
FlagStatus  UART_Status;
UART_Status  = UART_GetFlagStatus(UART1, UART_Flag_RX);
```

UART_ClearFlag

Table 523

函数名	UART_ClearFlag
函数原型	void UART_ClearFlag(UART_TypeDef* UARTx, uint16_t UART_FLAG)
功能描述	清除 UARTx 的待处理标志位
输入参数 1	UARTx：来选择 UART 外设（x 具体以规格书为准）
输入参数 2	UART_FLAG：UARTx 的标志位
返回值	无

使用示例:

```
/* 清除UART1接收数据标志位 */
UART_ClearFlag(UART1, UART_Flag_RX);
```

UART_DMACmd

Table 524

函数名	UART_DMACmd
函数原型	void UART_DMACmd(UART_TypeDef* UARTx, uint16_t UART_DMAReq, FunctionalState NewState)
功能描述	使能或者失能指定的 UARTx 的 DMA 请求
输入参数 1	UARTx：来选择 UART 外设（x 具体以规格书为准）
输入参数 2	UART_DMAReq：UARTx 的 DMA 请求源，定义在 UART_DMAReq_TypeDef 中
输入参数 3	NewState：UARTx 的 DMA 请求开启/关闭
返回值	无

UART_DMAReq

Table 525给出了所有UART_ DMAReq的取值列表。

Table 525

UART_ DMAReq	描述
UART_DMAReq_RX	UART 接收通道触发 DMA
UART_DMAReq_TX	UART 发送通道触发 DMA

使用示例:

```
/* 使能UART1发送数据触发DMA */
UART_DMACmd(UART1, UART_DMAReq_TX, ENABLE);
```

UART_LIN_MODE

Table 526

函数名	UART_LIN_MODE
函数原型	void UART_LIN_MODE(UART_TypeDef* UARTx, UART_LINMODE_TypeDef UART_LINMODE)
功能描述	配置工作模式为 LIN 模式
输入参数 1	UARTx：x 配置 2，来选择 UART2 外设

输入参数 2	UART_LINMODE：指向结构 UART_LINMODE_TypeDef 的指针，包含了外设 LIN 的配置信息
返回值	无

UART_LINMODE

设置UART的LIN工作模式，参阅Table 511获得这个参数的所有成员。

Table 527

UART_LINMODE	描述
UART_MASTER	LIN 主机工作模式
UART_SLAVER	LIN 从机工作模式

使用示例:

UART_LIN_MODE(UART2, UART_MASTER)//配置为LIN主机工作模式

UART_LIN_BKSIZE

Table 528

函数名	UART_LIN_BKSIZE
函数原型	void UART_LIN_BKSIZE(UART_TypeDef* UARTx, UART_BKSIZE_TypeDef BKSIZE)
功能描述	配置同步间隔段
输入参数 1	UARTx：x 配置 2，来选择 UART2 外设
输入参数 2	BKSIZE：指向结构 UART_BKSIZE_TypeDef 的指针，配置同步间隔段长度
返回值	无

BKSIZE

设置UART的LIN同步间隔段长度，参阅Table 511获得这个参数的所有成员。

Table 529

BKSIZE	描述
UART_BKSIZE_10	用 10 位时间长度发送
UART_BKSIZE_13	用 13 位时间长度发送

使用示例:

UART_LIN_BKSIZE(UART2, UART_BKSIZE_10)//配置同步间隔段用 10 位时间长度发送

UART_SendBreak

Table 530

函数名	UART_SendBreak
函数原型	void UART_SendBreak()
功能描述	发送同步间隔段
输入参数	无
返回值	无

使用示例:

UART_SendBreak();//发送同步间隔段

UART_LIN_SLVARENE

Table 531

函数名	UART_LIN_SLVARENE
函数原型	void UART_LIN_SLVARENE(UART_TypeDef* UARTx, FunctionalState NewState)
功能描述	使能自动重同步
输入参数 1	UARTx：x 配置 2，来选择 UART2 外设
输入参数 2	NewState：指向结构 FunctionalState 的指针，可以配置 Enable 或 Disable
返回值	无

使用示例:

UART_LIN_SLVARENE(UART2, Enable)//使能自动重同步

UART_LIN_LBDL

Table 532

函数名	UART_LIN_LBDL
函数原型	void UART_LIN_LBDL(UART_TypeDef* UARTx, UART_LBDL_TypeDef LBDL)
功能描述	配置 LIN 断路检测长度
输入参数 1	UARTx：x 配置 2，来选择 UART2 外设
输入参数 2	LBDL：指向结构 UART_LBDL_TypeDef 的指针，配置 LIN 断路检测长度
返回值	无

LBDL

设置LBDL工作模式，参阅Table 511获得这个参数的所有成员。

Table 533

LBDL	描述
UART_LBDL_10	LIN 断路检测长度 10 位
UART_LBDL_11	LIN 断路检测长度 11 位

使用示例:

UART_LIN_LBDL(UART2, UART_LBDL_10)//配置LIN断路检测长度10位

LIN_CallD(uint8_t id)

Table 534

函数名	LIN_CallD(uint8_t id)
函数原型	uint8_t LIN_CallD(uint8_t id)
功能描述	获取 PID
输入参数 1	id：从机 ID 值
返回值	获取 PID

使用示例:

LIN_CallD(0x32);//获取id为0x32的PID

LINCalChecksum

Table 535

函数名	LINCalChecksum
函数原型	uint8_t LINCalChecksum(uint8_t id, uint8_t *data, uint8_t len)
功能描述	获取校验值
输入参数 1	id：从机 ID 值，当 ID 为 0 时，为普通校验
输入参数 2	*data：发送数据的数组
输入参数 3	Len：发送数据的长度
返回值	LIN 校验值

使用示例:

LINCalChecksum(0x32,&data, 8);//获取id为32，发送数据长度为8的数组的校验值

综合使用示例（UART 发送 1Byte 数据）

```
UART_InitTypeDef UART_InitStruct; //定义UART初始化结构体变量
/* 使能UART2的时钟 */
RCC_APB1Cmd(ENABLE); //UART2外设挂载在APB1时钟上
RCC_APB1PeriphClockCmd(RCC_APB1Periph_UART2, ENABLE);
UART_InitStruct.UART_BaudRate = 115200; //串口波特率为115200
UART_InitStruct.UART_ClockFrequency = 32000000; //APB时钟频率配置为32MHz
UART_InitStruct.UART_Mode = UART_Mode_10B; //工作在模式1 /* 根据结构体配置初始化串口 */
UART_Init(UART2, &UART_InitStruct); /* 使能串口接收中断 */
UART_ITConfig(UART2, UART_IT_EN | UART_IT_RX, ENABLE);
NVIC_EnableIRQ(UART0_2_IRQn); /* 使能接收发送口 */
UART_TXCmd(UART2, ENABLE);
```

```
while(1)
{
  UART_SendData(UART2, 0x5A);
  while(UART_GetFlagStatus(UART2, UART_Flag_TX) == RESET);
  UART_ClearFlag(UART2, UART_Flag_TX);
}
```

23 IAP 固件库函数

SC32F1xxx IAP操作需要先解锁，后进行扇擦或全擦于指定位置，被保护的区域禁止IAP操作。

IAP 寄存器结构

IAP 寄存器结构，IAP_TypeDef，在文件“sc32f1xxx.h”中定义如下：

```
typedef struct
{
__IO uint32_t IAPKEY;
__IO uint32_t IAP_SNB;
__IO uint32_t RESERVED0;
__IO uint32_t IAP_CON;
} IAP_TypeDef;
```

IAP 寄存器表格

Table 536

寄存器	描述
IAP_KEY	数据保护寄存器
IAP_SNB	IAP扇区编号设置寄存器
IAP_CON	IAP控制寄存器

IAP 固件库函列表

Table 537

函数名	描述
IAP_Unlock	解除对 FLASH 控制寄存器的访问
IAP_Lock	恢复对 FLASH 控制寄存器的访问
IAP_WriteCmd	使能或失能 IAP 编程功能
IAP_EraseSector	擦除指定的 IAP 扇区
IAP_EEPROMEraseSector	擦除 EEPROM 指定的 IAP 扇区
IAP_ProgramWord	在指定地址对一个字（32 位）进行编程
IAP_ProgramHalfWord	在指定地址对一个半字（16 位）进行编程
IAP_ProgramByte	在指定地址对一个字节（8 位）进行编程
IAP_ReadWord	IAP 读一个字（32 位）的数据
IAP_ReadHalfWord	IAP 读一个半字（16 位）的数据
IAP_ReadByte	IAP 读一个字节（8 位）的数据
IAP_ProgramWordArray	在指定地址对批量字（32 位）进行编程
IAP_ProgramHalfWordArray	在指定地址对批量半字（16 位）进行编程
IAP_ProgramByteArray	在指定地址对批量字节（8 位）进行编程
IAP_ReadWordArray	IAP 读批量字（32 位）的数据
IAP_ReadHalfWordArray	IAP 读批量半字（16 位）的数据
IAP_ReadByteArray	IAP 读批量字节（8 位）的数据
IAP_SoftwareReset	IAP 软件复位，并配置启动区域

IAP 固件库函数详解

IAP_Unlock

Table 538

函数名	IAP_Unlock
函数原型	boolType IAP_Unlock(void)
功能描述	解除对 FLASH 控制寄存器的访问

输入参数	无
返回值	IAP 解锁结果(TURE 或 FALSE)

使用示例:

```
/* 解除对 FLASH 控制寄存器的访问 */
IAP_Unlock();
```

IAP_Lock

Table 539

函数名	IAP_Lock
函数原型	void IAP_Lock(void)
功能描述	恢复对 FLASH 控制寄存器的访问
输入参数	无
返回值	无

使用示例:

```
/* 恢复对 FLASH 控制寄存器的访问 */
IAP_Lock();
```

IAP_WriteCmd

Table 540

函数名	IAP_WriteCmd
函数原型	void IAP_WriteCmd(FunctionalState NewState)
功能描述	使能或失能 IAP 编程功能
输入参数	NewState：IAP 编程功能开启/关闭
返回值	无

使用示例:

```
/* 恢复对 FLASH 控制寄存器的访问 */
IAP_WriteCmd(ENABLE);
```

IAP_EraseSector

Table 541

函数名	IAP_EraseSector
函数原型	void IAP_EraseSector(uint32_t IAP_Sector)
功能描述	擦除指定的 IAP 扇区
输入参数	IAP_Sector：待擦除的扇区编号
返回值	无

使用示例:

```
/* 擦除的扇区 0x10 */
IAP_EraseSector(0x10);
```

IAP_EEPROMEraseSector（SC32F11、SC32F12、SC32F15 系列）

Table 542

函数名	IAP_EEPROMEraseSector
函数原型	void IAP_EEPROMEraseSector (uint32_t IAP_Sector)
功能描述	擦除指定的 IAP 扇区
输入参数	IAP_Sector：待擦除的扇区编号
返回值	无

使用示例:

```
/* 擦除的扇区 0x01 */
IAP_EEPROMEraseSector (0x01);
```

IAP_ProgramWord

Table 543

函数名	IAP_ProgramWord
函数原型	boolType IAP_ProgramWord(uint32_t Address, uint32_t Data)
功能描述	在指定地址对一个字（32 位）进行编程
输入参数 1	Address：待写入数据的地址
输入参数 2	Data：待写入的 32 位数据
返回值	写入成功\失败

使用示例:

```
/* 向 0x801FE00 写入 32 位数据 */
IAP_ProgramWord(0x801FE00,0x12345678);
```

IAP_ProgramHalfWord

Table 544

函数名	IAP_ProgramHalfWord
函数原型	boolType IAP_ProgramHalfWord(uint32_t Address, uint16_t Data)
功能描述	在指定地址对一个半字（16 位）进行编程
输入参数 1	Address：待写入数据的地址
输入参数 2	Data：待写入的 16 位数据
返回值	写入成功\失败

使用示例:

```
/* 向 0x801FE00 写入 16 位数据 */
IAP_ProgramHalfWord(0x801FE00,0x1234);
```

IAP_ProgramByte

Table 545

函数名	IAP_ProgramByte
函数原型	boolType IAP_ProgramByte(uint32_t Address, uint8_t Data)
功能描述	在指定地址对一个字节（8 位）进行编程
输入参数 1	Address：待写入数据的地址
输入参数 2	Data：待写入的 8 位数据
返回值	写入成功\失败

使用示例:

```
/* 向 0x801FE00 写入 8 位数据 */
IAP_ProgramHalfWord(0x801FE00,0x12);
```

IAP_ReadWord

Table 546

函数名	IAP_ReadWord
函数原型	uint32_t IAP_ReadWord(uint32_t Address)
功能描述	IAP 读一个字（32 位）的数据
输入参数	Address：待读取数据的地址
返回值	读到的 32 位数据

使用示例:

```
/* 读取 0x801FE00 的 32 位数据 */
uint32_t ReadData;
ReadData = IAP_ReadWord(0x801FE00);
```

IAP_ReadHalfWord

Table 547

函数名	IAP _ReadHalfWord
函数原型	uint16_t IAP_ReadHalfWord(uint32_t Address)

功能描述	IAP 读一个半字（16 位）的数据
输入参数	Address：待读取数据的地址
返回值	读到的 16 位数据

使用示例:

```
/* 读取 0x801FE00 的 16 位数据 */
  uint16_t ReadData;
  ReadData = IAP_ReadWord(0x801FE00);
```

IAP_ReadByte

Table 548

函数名	IAP _ReadByte
函数原型	uint8_t IAP_ReadByte(uint32_t Address)
功能描述	IAP 读一个字节（8 位）的数据
输入参数	Address：待读取数据的地址
返回值	读到的 8 位数据

使用示例:

```
/* 读取 0x801FE00 的 8 位数据 */
  uint8_t ReadData;
  ReadData = IAP_ReadWord(0x801FE00);
```

IAP_ProgramWordArray

Table 549

函数名	IAP_ProgramWordArray
函数原型	uint8_t IAP_ProgramWordArray(uint32_t Address, uint32_t* ByteArray, uint8_t ArraySize)
功能描述	在指定地址对批量字（32 位）进行编程
输入参数 1	Address：待写入数据的地址
输入参数 2	ByteArray：待写入数据的数组指针
输入参数 3	ArraySize：待写入数据的大小
返回值	写入成功数据的大小

使用示例:

```
/* 向 0x803FE00 写入 32 位数组数据 */
  uint32_t Array_Write[3]={0x11223344,0x22334455,0x33445566};
  uint8_t writeFlag;
  writeFlag = IAP_ProgramWordArray(0x803FE00,Array_Write,3);
```

IAP_ProgramHalfWordArray

Table 550

函数名	IAP_ProgramHalfWordArray
函数原型	uint8_t IAP_ProgramHalfWordArray(uint32_t Address, uint16_t* ByteArray, uint8_t ArraySize)
功能描述	在指定地址对批量半字（16 位）进行编程
输入参数 1	Address：待写入数据的地址
输入参数 2	ByteArray：待写入数据的数组指针
输入参数 3	ArraySize：待写入数据的大小
返回值	写入成功数据的大小

使用示例:

```
/* 向 0x803FE00 写入 16 位数组数据 */
  uint16_t Array_Write[5]={0x1111,0x2222,0x3333,0x4444,0x5555};
  uint8_t writeFlag;
  writeFlag = IAP_ ProgramHalfWordArray(0x803FE00,Array_Write,5);
```

IAP_ProgramByteArray

Table 551

函数名	IAP_ProgramByteArray
函数原型	uint8_t IAP_ProgramByteArray(uint32_t Address, uint8_t* ByteArray, uint8_t ArraySize)
功能描述	在指定地址对批量字节（8 位）进行编程
输入参数 1	Address：待写入数据的地址
输入参数 2	ByteArray：待写入数据的数组指针
输入参数 3	ArraySize：待写入数据的大小
返回值	写入成功数据的大小

使用示例:

```
/* 向 0x803FE00 写入 8 位数组数据 */
uint8_t Array_Write[5]={ 0x99,0x88,0x77,0x66,0x55};
uint8_t readflag;
writeFlag = IAP_ ProgramByteArray(0x803FE00,Array_Write,5);
```

IAP_ReadWordArray

Table 552

函数名	IAP_ ReadWordArray
函数原型	uint8_t IAP_ReadWordArray(uint32_t Address, uint32_t* ByteArray, uint8_t ArraySize)
功能描述	IAP 读批量字（32 位）的数据
输入参数 1	Address：待读取数据的地址
输入参数 2	ByteArray：待读取数据的数组指针
输入参数 3	ArraySize：待读取数据的大小
返回值	读取数据的大小

使用示例:

```
/* 读取 0x803FE00 的批量 32 位数据 */
uint32_t Array_Read[3];
uint8_t readflag;
readflag = IAP_ReadWordArray(0x803FE00,Array_Read,3);
```

IAP_ReadHalfWordArray

Table 553

函数名	IAP_ReadHalfWordArray
函数原型	uint8_t IAP_ReadHalfWordArray(uint32_t Address, uint16_t* ByteArray, uint8_t ArraySize)
功能描述	IAP 读批量半字（16 位）的数据
输入参数 1	Address：待读取数据的地址
输入参数 2	ByteArray：待读取数据的数组指针
输入参数 3	ArraySize：待读取数据的大小
返回值	读取数据的大小

使用示例:

```
/* 读取 0x803FE00 的批量 16 位数据 */
Uint16_t Array_Read[3];
uint8_t readflag;
readflag = IAP_ReadWordArray(0x803FE00,Array_Read,3);
```

IAP_ReadByteArray

Table 554

函数名	IAP_ReadByteArray
函数原型	uint8_t IAP_ReadByteArray(uint32_t Address, uint8_t* ByteArray, uint8_t ArraySize)
功能描述	IAP 读批量字节（8 位）的数据
输入参数 1	Address：待读取数据的地址

输入参数 2	ByteArray：待读取数据的数组指针
输入参数 3	ArraySize：待读取数据的大小
返回值	读取数据的大小

使用示例:

```
/* 读取 0x803FE00 的批量 8 位数据 */
Uint8_t Array_Read[3];
uint8_t readflag;
readflag = IAP_ReadWordArray(0x803FE00,Array_Read,3);
```

IAP_SoftwareReset

Table 555

函数名	IAP_SoftwareReset
函数原型	void IAP_SoftwareReset(IAP_BTLD_TypeDef IAP_BTLDType)
功能描述	IAP 软件复位，并配置启动区域
输入参数	IAP_BTLDType：IAP 复位后启动区域
返回值	无

IAP_BTLDType

IAP复位后启动区域选择，参阅Table 556获得这个参数的所有成员。

Table 556

UART_ FLAG	描述
IAP_BTLD_APPROM	IAP 复位后从 APPROM 区域启动
IAP_BTLD_LDROM	IAP 复位后从 LDROM 区域启动
IAP_BTLD_SRAM	IAP 复位后从 SRAM 区域启动

使用示例:

```
/* IAP复位后从APP区域启动 */
IAP_SoftwareReset(IAP_BTLD_APPROM);
```

综合使用示例（向 0x801FE00 写入 0x12345678）

```
IAP_Unlock();  //解锁IAP操作
IAP_EraseSector((0x801FE00 - FLASH_BASE) / 512);  //擦除地址所在扇区，每个扇区大小为512Byte
IAP_WriteCmd(ENABLE);        //开启写使能
IAP_ProgramWord(0x801FE00,0x12345678);   //向目标地址写入特定数据
IAP_Lock();      //上锁IAP操作，并且复位IAP操作寄存器
```

24 PGA 固件库函数

赛元SC32f11xx系列内建一个可变增益放大器，此PGA有两个输入通道。PGA的输出可被选为用于AD转换器的模拟输入或者CMP0正端的模拟输入。

PGA 寄存器结构

PGA寄存器结构，PGA_TypeDef，在文件“sc32f1xxx.h”中定义如下：

```
typedef struct
{
    __IO uint32_t PGA_CON;
} PGA_TypeDef;
```

PGA 寄存器表格

Table 557

寄存器	描述
PGA_CON	PGA控制寄存器

PGA 固件库函列表

Table 558

函数名	描述
PGA_DeInit	将外设 PGA 寄存器重设为缺省值
PGA_Init	根据 PGA_InitStruct 中指定的参数初始化外设 PGA 寄存器
PGA_Cmd	使能 PGA 寄存器
PGA_OffsetTrimConfig	配置 PGA 的微调值
PGA_StartCalibration	启动或停止所选 PGA 外设的校准

PGA 固件库函数详解

PGA_ DeInit

Table 559

函数名	PGA_DeInit
函数原型	void PGA_DeInit(PGA_TypeDef* PGAx)
功能描述	将外设 PGA 寄存器重设为缺省值
输入参数	PGAx: 来选择 PGA 外设
返回值	无

使用示例:

```
/*将外设 PGA 寄存器重设为缺省值*/
PGA_DeInit(PGA);
```

PGA_ Init

Table 560

函数名	PGA_Init
函数原型	void PGA_Init(PGA_TypeDef* PGAx, PGA_InitTypeDef* PGA_InitStruct)
功能描述	根据 PGA_InitStruct 中指定的参数初始化外设 PGA 寄存器
输入参数 1	PGAx: 来选择 PGA 外设
输入参数 2	PGA_InitStruct:指向结构 PGA_InitTypeDef 的指针，包含了外设 PGA 的配置信息 参阅 Section：PGA_InitTypeDef 查阅更多该参数允许取值范围
返回值	无

PGA _InitStruct

PGA_InitTypeDef 定义于文件“sc32f1xxx_pga.h”：

```
typedef struct
{
uint16_t PGA_INSEL;
uint16_t PGA_COM;
uint16_t PGA_GAN;
} PGA_InitTypeDef;
```

PGA_INSEL

PGA信号输入端设定，参阅Table 561获得这个参数的所有成员。

Table 561

PGA_INSEL	描述
PGA_INSEL_PGAI0	PGA 信号输入端设定为 PGAI0
PGA_INSEL_PGAI1	PGA 信号输入端设定为 PGAI1

PGA_COM

PGA共模电压控制位，参阅Table 562获得这个参数的所有成员。

Table 562

PGA_COM	描述
PGA_COM_0V	共模电压为 0V
PGA_COM_1_2V	共模电压为 1.2V

PGA_GAN

PGA放大倍数设定，参阅Table 563获得这个参数的所有成员。

Table 563

PGA_GAN	描述
PGA_GAN_Invert19	反相输入放大 19 倍
PGA_GAN_Invert99	反相输入放大 99 倍
PGA_GAN_NonInvert20	同相输入放大 20 倍
PGA_GAN_NonInvert100	同相输入放大 100 倍

使用示例:

```
/* 根据 PGA_InitStruct 中指定的参数初始化 PGA */
PGA_InitTypeDef PGA_InitStruct;                 //定义PGA初始化结构体变量
PGA_InitStruct.PGA_COM = PGA_COM_0V;            //共模0V
PGA_InitStruct.PGA_GAN = PGA_GAN_NonInvert20;   //同相放大20倍
PGA_InitStruct.PGA_INSEL = PGA_INSEL_PGAI1;     //选择PGA1输入通道
PGA_Init(PGA,&PGA_InitStruct);
```

PGA_Cmd

Table 564

函数名	PGA_Cmd
函数原形	void PGA_Cmd(PGA_TypeDef* PGAx, FunctionalState NewState)
功能描述	使能或者失能 PGA 外设
输入参数 1	PGAx: 来选择 PGA 外设
输入参数 2	NewState: 外设 PGA 的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* 使能PGA 外设 */
PGA_Cmd(PGA, ENABLE);
```

PGA_OffsetTrimConfig

Table 565

函数名	PGA_OffsetTrimConfig
函数原型	void PGA_OffsetTrimConfig(PGA_TypeDef* PGAx, uint32_t PGA_TrimValue)
功能描述	配置 PGA 的微调值

输入参数 1	PGAx: 来选择 PGA 外设
输入参数 2	PGA_TrimValue: 微调值。该参数可以是小于等于 0x0000001F 的任意值
返回值	无

使用示例:

```
/ *配置PGA的微调值 */
PGA_OffsetTrimConfig (PGA, 0x0A);
```

PGA_StartCalibration

Table 566

函数名	PGA_StartCalibration
函数原型	void PGA_StartCalibration(PGA_TypeDef* PGAx, FunctionalState NewState)
功能描述	启动或停止所选 PGA 外设的校准
输入参数 1	PGAx: 来选择 PGA 外设
输入参数 2	NewState: 外设 PGA 外设的校准的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/ *开启 PGA 外设校准*/
PGA_StartCalibration(PGA, ENABLE);
```

综合使用示例

```
static volatile FlagStatus ADC_Flag;
/* PGA 同相反相放大配置 */
//(PGA_MODE:NonInvert_MODE=0,OP_MODE:Invert_MODE=1)
#define PGA_MODE 1

void PGA_AmplifiesVoltage_Function(void)
{
//ADC初始化配置
volatile uint16_t ADC_Value;
RCC_APB2Cmd(ENABLE);                    //ADC、PGA外设挂载在APB2时钟上
ADC_InitTypeDef ADC_InitStruct;         //定义ADC初始化结构体变量

ADC_StructInit(&ADC_InitStruct);
ADC_InitStruct.ADC_ConvMode = ADC_ConvMode_Single;  //ADC单次转换模式
ADC_InitStruct.ADC_Prescaler = ADC_Prescaler_3CLOCK;//转换时间为3个周期
ADC_InitStruct.ADC_VREF = ADC_VREF_VDD;     //参考电压为VDD
ADC_Init(ADC, &ADC_InitStruct);             //根据结构体标志位配置ADC
ADC_SetChannel(ADC,ADC_Channel_PGA);        //选择PGA通道
ADC_ITConfig(ADC,ADC_IT_ADCIF,ENABLE);      //使能ADC中断
NVIC_EnableIRQ(ADC_IRQn);
ADC_Cmd(ADC,ENABLE);                        //使能ADC；

PGA_InitTypeDef PGA_InitStruct;             //定义PGA初始化结构体变量
PGA_InitStruct.PGA_COM = PGA_COM_0V;        //共模0V
PGA_InitStruct.PGA_GAN = PGA_GAN_NonInvert20;   //同相放大20倍
PGA_InitStruct.PGA_INSEL = PGA_INSEL_PGAI1;     //选择PGA1输入通道
PGA_Init(PGA,&PGA_InitStruct);
PGA_Cmd(PGA,ENABLE);                        //PGA使能

while(1)
{
ADC_SoftwareStartConv(ADC); //开启转换
while(ADC_Flag == RESET); //等待ADC转换完成
```

```
ADC_Value = ADC_GetConversionValue(ADC);  //获取ADC转换值
printf("\r\n ADC Conversion Value is 0x%x \r\n",ADC_Value);
for (int i=0;i<5000;i++);
}
}
void PGA_AmplifiesVoltage_ADCHandler(void)
{
  if(ADC_GetFlagStatus(ADC,ADC_Flag_ADCIF)) //判断ADC标志位
  {
    ADC_ClearFlag(ADC,ADC_Flag_ADCIF);  //清除ADC标志位
    ADC_Flag = SET;  //自定义标志置起
  }
}
```

25 DAC 固件库函数

赛元SC32f15xx内部集成一个独立的10 Bits数模转换器（DAC）。此DAC有两个独立的输出端口DACOUT0和DACOUT1，DAC也可在芯片内部输出到OP1/OP2的反相端。

DAC 寄存器结构

DAC寄存器结构，PGA_TypeDef，在文件“sc32f15xx.h”中定义如下：

```
typedef struct
{
    __IO uint32_t DAC_STS;
    __IO uint32_t DAC_IN;
    __IO uint32_t DAC_CFG;
} DAC_TypeDef;
```

DAC 寄存器表格

Table 567

寄存器	描述
DAC_STS	DAC状态寄存器
DAC_IN	DAC转换寄存器
DAC_CFG	DAC配置寄存器

DAC 固件库函列表

Table 568

函数名	描述
DAC_DeInit	将外设 DAC 寄存器重设为缺省值
DAC_Cmd	使能 DAC 寄存器
DAC_ChannelConfig	配置 DAC 的输出通道
DAC_VrefConfig	配置 DAC 的参考电压
DAC_SetConversionValue	配置 DAC 装换值
DAC_GetFlagStatus	获取 DAC 标志位

DAC 固件库函数详解

DAC_ DeInit

Table 569

函数名	DAC_DeInit
函数原型	void DAC_DeInit(DAC_TypeDef* DACx)
功能描述	将外设 DAC 寄存器重设为缺省值
输入参数	DACx: 来选择 DAC 外设
返回值	无

使用示例:

```
/*将外设 DAC 寄存器重设为缺省值*/
DAC_DeInit (DAC);
```

DAC_Cmd

Table 570

函数名	DAC_Cmd
函数原形	void DAC_Cmd(DAC_TypeDef* DACx, FunctionalState NewState)
功能描述	使能或者失能 DAC 外设
输入参数 1	DACx: 来选择 DAC 外设

输入参数 2	NewState: 外设 DAC 的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* 使能DAC 外设 */
DAC_Cmd (DAC, ENABLE);
```

DAC_ChannelConfig

Table 571

函数名	DAC_ChannelConfig
函数原型	void DAC_ChannelConfig(DAC_TypeDef* DACx, DAC_Channel_TypeDef DAC_Channel, FunctionalState NewState)
功能描述	配置 DAC 的输出通道
输入参数 1	DACx: 来选择 DAC 外设
输入参数 2	DAC_Channel: 选择 DAC 的输出通道
输入参数 3	NewState: 外设 DAC 的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

DAC_Channel

配置DAC 的输出通道，参阅Table 572获得这个参数的所有成员。

Table 572

DAC_Channel	描述
DAC_Channel_0	DAC 的输出通道选择 DAC_Channel_0
DAC_Channel_1	DAC 的输出通道选择 DAC_Channel_1

使用示例:

```
/ *配置DAC 的输出通道DAC_Channel_0 */
DAC_ChannelConfig (DAC, DAC_Channel_0，ENABLE);
```

DAC_VrefConfig

Table 573

函数名	DAC_VrefConfig
函数原型	void DAC_VrefConfig(DAC_TypeDef* DACx, DAC_REF_TypeDef DAC_REF)
功能描述	配置 DAC 的参考电压
输入参数 1	DACx: 来选择 DAC 外设
输入参数 2	DAC_REF: 选择 DAC 的参考电压
返回值	无

DAC_REF

配置DAC的参考电压，参阅Table 574获得这个参数的所有成员。

Table 574

DAC_REF	描述
DAC_RefSource_VDD	DAC 的参考电压选择 VDD
DAC_RefSource_VREF	DAC 的参考电压选择 VREF

使用示例:

```
/ *配置DAC 的参考电压选择VDD */
DAC_VrefConfig(DAC, DAC_RefSource_VDD);
```

DAC_SetConversionValue

Table 575

函数名	DAC_SetConversionValue
函数原型	void DAC_SetConversionValue(DAC_TypeDef* DACx, uint16_t DAC_Value)
功能描述	配置 DAC 的转化值

输入参数 1	DACx: 来选择 DAC 外设
输入参数 2	DAC_Value: 设置 DAC 的转化值
返回值	无

使用示例:

```
/ *配置DAC 的转化值*/
DAC_SetConversionValue (DAC, 0xFF);
```

DAC_GetFlagStatus

Table 576

函数名	DAC_GetFlagStatus
函数原型	FlagStatus DAC_GetFlagStatus(DAC_TypeDef* DACx, uint16_t DAC_FLAG)
功能描述	获取 DAC 的标志位
输入参数 1	DACx: 来选择 DAC 外设
输入参数 2	DAC_FLAG: 选择 DAC 的标志位
返回值	无

DAC_FLAG

DAC的标志位，参阅 Table 577获得这个参数的所有成员。

Table 577

DAC_FLAG	描述
DAC_DACSTA_Endconversion	DAC 转换已完成
DAC_DACSTA_Inconversion	DAC 转换正在转换

使用示例:

```
/ *获取DAC的标志位 */
  DAC_GetFlagStatus(DAC, DAC_DACSTA_Endconversion)；
```

综合使用示例

```
void DAC_Output_Function(void)
{
  volatile uint32_t ADC_Value = 0;
  RCC_APB2Cmd(ENABLE);   //DAC外设挂载在APB2时钟上
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC,ENABLE); //开启ADC外设时钟
  /*DAC初始化*/
  DAC_DeInit(DAC);   //复位DAC输出
  DAC_ChannelConfig(DAC,DAC_Channel_0,ENABLE);//配置DAC_OUTPUT0使能
  DAC_VrefConfig(DAC,DAC_RefSource_VREF);    //配置DAC基准电压为Vref
  DAC_SetConversionValue(DAC,100); //配置转化值,DAC输出电压：DACOUT=(Vref/1024)*DAC_Value
  /*配置VREF基准电源值*/
  Vref_LevelConfig(VREF,Vref_InReferenceVoltage_2_048);    //配置基准源的参考电压为2.048
  DAC_Cmd(DAC,ENABLE);   //使能DAC输出

  /*ADC初始化配置*/
  ADC_DeInit(ADC);                          //ADC寄存器复位
  ADC_InitTypeDef ADC_InitStruct;
  ADC_StructInit(&ADC_InitStruct);
  ADC_InitStruct.ADC_ConvMode = ADC_ConvMode_Single; //设置ADC为单次转换模式
  ADC_InitStruct.ADC_EAIN = ADC_EAIN_5;             //使能AIN5通道
  ADC_InitStruct.ADC_Prescaler = ADC_Prescaler_3CLOCK;//ADC采样周期数设置为3个时钟周期
  ADC_InitStruct.ADC_VREF = ADC_RefSource_VREF;      //ADC参考电压选择基准源电压
  ADC_Init(ADC,&ADC_InitStruct);

  ADC_SetChannel(ADC,ADC_Channel_5); //选择ADC输入通道
```

```
    ADC_Cmd(ADC,ENABLE);       //使能ADC
    ADC_SoftwareStartConv(ADC); //触发ADC转换

    while(1)
    {
       if(ADC_GetFlagStatus(ADC,ADC_Flag_ADCIF) == SET)
       {
          ADC_ClearFlag(ADC,ADC_Flag_ADCIF);
          ADC_Value = ADC_GetConversionValue(ADC);
          printf("....................................\r\n");
          printf("DAC输出电压采样值为0x%x\r\n",ADC_Value);
          ADC_SoftwareStartConv(ADC); //触发ADC转换
       }
       for(uint16_t i=0;i<65535;i++)
       {
          for(uint16_t j=0;j<30;j++);
       }
    }
}
```

26 CAN 固件库函数

赛元 SC32F11xx 与 SC32F15Gx 系列提供CAN外设支持协议 CAN2.0和CAN FD协议，具有8组接收缓存和8组发送缓存，支持FIFO发送和ID优先级发送两种发送模式，内设有8组接收滤波器。

CAN 寄存器结构

CAN寄存器结构，CAN_TypeDef，在文件“sc32f1xxx.h”中定义如下：

```
typedef struct
{
__IO uint32_t CAN_TTSL;
__IO uint32_t CAN_TTSH;
__IO uint32_t CAN_CFG_STAT;
__IO uint32_t CAN_RTIE;
__IO uint32_t CAN_S_SEG;
__IO uint32_t CAN_F_SEG;
__IO uint32_t CAN_EALCAP;
__IO uint32_t CAN_ACFCTRL;
__IO uint32_t CAN_ACF;
__IO uint32_t TTCAN_CFG;
__IO uint32_t TTCAN_RFMSG;
__IO uint32_t TTCAN_TRIG_CFG;
__IO uint32_t TTCAN_WTRIG;
__IO uint32_t CAN_IDE;
__IO uint32_t CAN_TIML;
__IO uint32_t CAN_TIMH;
} CAN_TypeDef;
```

CAN 寄存器表格

Table 578

寄存器	描述
CAN_TTSL	发送帧时间戳寄存器低位
CAN_TTSH	发送帧时间戳寄存器高位
CAN_CFG_STAT	CAN配置寄存器包含状态位
CAN_RTIE	CAN中断控制及中断标志位寄存器
CAN_S_SEG	timing配置寄存器
CAN_F_SEG	timing配置寄存器
CAN_EALCAP	收发错误计数寄存器
CAN_ACFCTRL	接收过滤控制寄存器
CAN_ACF	接收掩码配置寄存器
TTCAN_CFG	时间戳控制寄存器
TTCAN_RFMSG	参考消息控制寄存器
TTCAN_TRIG_CFG	发射触发控制寄存器
TTCAN_WTRIG	监控触发时间控制寄存器
CAN_IDE	中断及DMA控制使能
CAN_TIML	时间戳timer的读写寄存器
CAN_TIMH	时间戳timer的读写寄存器

CAN 固件库函列表

Table 579

函数名	描述
CAN_DeInit	将外设 CAN 寄存器重设为缺省值

CAN_Init	根据 CAN_InitStruct 中指定的参数初始化外设 CAN 寄存器
CAN_StructInit	把 CAN_InitStruct 中的每一个参数按缺省值填入
CAN_TBUFSelect	配置 CAN 发送数据存储方式
CAN_FBaudRate_Select	配置 CAN 外设的高速波特率
CAN_SBaudRate_Select	配置 CAN 外设的低速波特率
CAN_Trans_Select	配置选择 CAN 外设的发送模式
CAN_RxThresholdConfig	设置 CAN 外设阈值
CAN_FliterInit	配置 CAN 过滤器
CAN_ModeSelect	配置 CAN 工作模式
CAN_Trans_Init	CAN 发送的数据初始化，发送数据按 CanTxMsg 中的每一个参数按缺省值填入
CAN_TransStop	配置 CAN 发送中止
CAN_RecieveConfig	CAN 数据接收读取
CAN_GetRTSL	获取 CAN 接收时间戳的低位
CAN_GetRTSH	获取 CAN 接收时间戳的高位
CAN_GetTTSL	获取 CAN 发送时间戳的低位
CAN_GetTTSH	获取 CAN 发送时间戳的高位
CAN_ITConfig	使能 CAN 中断
CAN_GetFlagStatus	获取 CAN 中断标志位状态
CAN_ClearFlag	清除 CAN 中断标志位
CAN_GetTECNT	获取 CAN 发送数据错误数量
CAN_GetRECNT	获取 CAN 接收数据错误数量
CAN_ResetCmd	使能 CAN 外设复位

CAN 固件库函数详解

CAN_DeInit

Table 580

函数名	CAN_DeInit
函数原型	void CAN_DeInit(CAN_TypeDef* CANx)
功能描述	将外设 CAN 寄存器重设为缺省值
输入参数	CANx: 来选择 CAN 外设
返回值	无

使用示例:

```
/*将外设 CAN 寄存器重设为缺省值*/
CAN_DeInit(CAN);
```

CAN_ Init

Table 581

函数名	CAN_Init
函数原型	void CAN_Init(CAN_TypeDef *CANx, CAN_InitTypeDef *CAN_InitStruct)
功能描述	根据 CAN_InitTypeDef 中指定的参数初始化外设 CAN 寄存器
输入参数 1	CANx: 来选择 CAN 外设
输入参数 2	CAN_InitStruct:指向结构 CAN_InitTypeDef 的指针，包含了外设 CAN 的配置信息 参阅 Section：CAN_InitTypeDef 查阅更多该参数允许取值范围
返回值	无

CAN_InitStruct

CAN_InitTypeDef 定义于文件“sc32f1xxx_can.h”：

```
typedef struct
{
```

```
uint32_t S_SJW;
uint32_t S_SEG1;
uint32_t S_SEG2;
uint32_t S_PRESCALER;
uint32_t F_SJW;
uint32_t F_SEG1;
uint32_t F_SEG2;
uint32_t F_PRESCALER;
uint32_t CANFDFrame;
uint32_t CANROMSelect;
uint32_t CAN_MOD;
uint32_t CANTSMODESelect;
uint32_t CANFDISOSelect;
uint16_t CANTimeStampPositon;
uint16_t CANTBUFSelect;
uint16_t  TransMod;
FunctionalState CANTXEN;
FunctionalState CANRXEN;
FunctionalState CANTimeStamp;
FunctionalState CANTimeStampCnt;
FunctionalState CAN_PTB_AutoRetrans;
FunctionalState CAN_STB_AutoRetrans;
}CAN_InitTypeDef;
```

CANFDFrame

CAN发送标准帧与扩张帧设定，参阅Table 582获得这个参数的所有成员。

Table 582

CANFDFrame	描述
CAN_Standard_frame	报文设置为 2.0 帧
CAN_FD_frame	报文设置为远端帧

CANROMSelect

CAN接收缓冲区溢出模式选择位，参阅Table 583获得这个参数的所有成员。

Table 583

CANROMSelect	描述
CANROM_Old	丢弃新接收的报文。
CANROM_New	最早的报文将丢弃

CAN_MOD

CAN工作模式设定，参阅Table 584获得这个参数的所有成员。

Table 584

CAN_MOD	描述
CAN_Normal	普通模式
CAN_Listenonly	只听模式
CAN_StandBy	待机模式
CAN_LoopBack	外部循环模式
CAN_LoopBackIn	内部循环模式
CAN_LoopBackSack	外环模式自动应答模式

CANTSMODESelect

CAN发送模式设定，参阅Table 585获得这个参数的所有成员。

Table 585

CANTSMODESelect	描述
TX_FIFO	FIFO 发送
ID_priority	以 ID 为优先级发送

CANFDISOSelect

CAN FD模式下发送IOS配置，参阅Table 586获得这个参数的所有成员。

Table 586

CANFDISOSelect	描述
CAN_BOSCH	非 ISO 模式
CAN_ISO	ISO 模式

CANTimeStampPositon

CAN时间戳选择位，参阅Table 587获得这个参数的所有成员。

Table 587

CANTimeStampPositon	描述
CANTimeStampPosition_SOF	时间戳位置 SOF
CANTimeStampPosition_EOF	时间戳位置 EOF

CANTBUFSelect

CAN发送数据存贮选择位，参阅Table 588获得这个参数的所有成员。

Table 588

CANTBUFSelect	描述
CANTBUF_Tx_PTB	CAN 发送数据 PTB 存贮
CANTBUF_Tx_STB	CAN 发送数据 STB 存贮

TransMod

CAN发送数据方式设置，参阅Table 589获得这个参数的所有成员。

Table 589

TransMod	描述
CAN_TransMod_TPE	PTB 信息进行传输
CAN_TransMod_TSONE	STB 传输一帧报文
CAN_TransMod_TSALL	STB 传输所有报文

使用示例:

```
/* 根据CAN_InitStruct 中指定的参数初始化 CAN */
CAN_InitTypeDef CAN_InitStruct;                    //定义CAN初始化结构体
CAN_StructInit(&CAN_InitStruct);
CAN_FBaudRate_Select(Baudrate_250kHz);     //快速时钟波特率为250k
CAN_SBaudRate_Select(Baudrate_250kHz);     //慢速时钟波特率为250k
CAN_InitStruct.CANFDFrame = FD_Mode_on;      //CAN_FD帧
CAN_InitStruct.CANFDISOSelect = CAN_ISO;     //采用ISO CAN_FD模式
CAN_InitStruct.CANTBUFSelect = CANTBUF_Tx_STB;//采用次级发送缓存
CAN_InitStruct.CANROMSelect = CANROM_Old;    //丢弃老报文
CAN_InitStruct.CANTSMODESelect = TX_FIFO;    //STB采用FIFO发送模式
CAN_InitStruct.CAN_MOD = CAN_Normal;       //正常模式
CAN_InitStruct.CANTXEN = ENABLE;           //CAN TX使能
CAN_InitStruct.CANRXEN = ENABLE;           //CAN RX使能
CAN_Init(CAN, &CAN_InitStruct);
```

CAN_StructInit

Table 590

函数名	CAN_StructInit
函数原形	void CAN_StructInit(CAN_InitTypeDef* CAN_InitStruct)
功能描述	把 CAN_InitStruct 中的每一个参数按缺省值填入
输入参数	CAN_InitStruct：指向结构 CAN_InitTypeDef 的指针，待初始化
返回值	无

使用示例:

```
/* 把 CAN_InitStruct 中的每一个参数按缺省值填入 */
CAN_InitTypeDef CAN_InitStruct;                    //定义CAN初始化结构体
CAN_StructInit(&CAN_InitStruct);
```

CAN_TBUFSelect

Table 591

函数名	CAN_TBUFSelect
函数原形	void CAN_TBUFSelect(CAN_TypeDef *CANx, CANTBUF_TypeDef CANTBUF)
功能描述	配置 CAN 发送数据存储方式
输入参数 1	CANx: 来选择 CAN 外设
输入参数 2	CANTBUF：来选择 CAN 发送数据存贮方式
返回值	无

CANTBUF 为CAN发送数据存贮方式选择位，参阅Table 588获得这个参数的所有成员。

使用示例:

```
/* 选择STB为发送数据区 */
CAN_TBUFSelect(CAN, CANTBUF_Tx_STB);
```

CAN_FBaudRate_Select

Table 592

函数名	CAN_FBaudRate_Select
函数原型	void CAN_FBaudRate_Select(unsigned int CAN_Clk,unsigned int CAN_FPrescaler,CAN_BaudRate BaudRate)
功能描述	配置 CAN 外设的高速波特率
输入参数 1	CAN_Clk: CAN 工作时钟
输入参数 2	CAN_FPrescaler :预分频
输入参数 3	BaudRate: 选择配置 CAN 的低速波特率
返回值	无

BaudRate

CAN高速波特率设定，参阅Table 593获得这个参数的所有成员。

Table 593

BaudRate	描述
Baudrate_125kHz	高速波特率设置为 125KHZ
Baudrate_250kHz	高速波特率设置为 250KHZ
Baudrate_500kHz	高速波特率设置为 500KHZ
Baudrate_800kHz	高速波特率设置为 800KHZ
Baudrate_1000kHz	高速波特率设置为 1000KHZ

使用示例:

```
/ *配置 CAN 外设的高速波特率250khz*/
CAN_FBaudRate_Select(Baudrate_250kHz);
```

CAN_SBaudRate_Select

Table 594

函数名	CAN_SBaudRate_Select
函数原型	void CAN_SBaudRate_Select(unsigned int CAN_Clk,unsigned int CAN_Prescaler,CAN_BaudRate BaudRate)
功能描述	配置 CAN 外设的低速波特率
输入参数 1	CAN_Clk: CAN 工作时钟
输入参数 2	CAN_SPrescaler :预分频
输入参数 3	BaudRate: 选择配置 CAN 的低速波特率
返回值	无

BaudRate为CAN低速波特率设定，参阅Table 593获得这个参数的所有成员。

使用示例:

```
/ *配置 CAN 外设的低速波特率250khz*/
CAN_SBaudRate_Select(32000000,8,Baudrate_250kHz);
```

CAN_Trans_Select

Table 595

函数名	CAN_Trans_Select
函数原型	void CAN_Trans_Select(CAN_TypeDef *CANx,CAN_TransMod_TypeDef TransModSelect)
功能描述	配置 CAN 发送方式
输入参数 1	CANx: 来选择 CAN 外设
输入参数 2	TransModSelect: 配置 CAN 发送方式
返回值	无

TransModSelect为CAN发送方式选择位，参阅Table 589获得这个参数的所有成员。

使用示例:

```
/ *选择STB逐个发送*/
CAN_Trans_Select(CAN, CAN_TransMod_TSONE);
```

CAN_RxThresholdConfig

Table 596

函数名	CAN_RxThresholdConfig
函数原型	void CAN_RxThresholdConfig(CAN_TypeDef *CANx, int Rth)
功能描述	配置 CAN 接收阈值
输入参数 1	CANx: 来选择 CAN 外设
输入参数 2	Rth：配置 CAN 接收阈值
返回值	无

使用示例:

```
/ *设置CAN接收阈值为3*/
CAN_Trans_Select(CAN, 3 );
```

CAN_FliterInit

Table 597

函数名	CAN_FliterInit
函数原型	void CAN_FliterInit(CAN_FliterTypeDef *CAN_FliterStruct)
功能描述	配置 CAN 过滤器
输入参数 1	CAN_FliterStruct: 指向结构 CAN_FliterTypeDef 的指针，待初始化
返回值	无

CAN_FliterTypeDef

CAN_FliterTypeDef 定义于文件“sc32f1xxx_can.h”：

```
typedef struct
{
uint32_t Acode;
uint32_t Amask;
uint8_t CAN_Fliter;
uint8_t CANFliterFrame;
}CAN_FliterTypeDef;
```

CAN_Fliter

CAN过滤器选择位，参阅Table 598获得这个参数的所有成员。

Table 598

CAN_Fliter	描述
Fliter0	选择过滤器 0
Fliter1	选择过滤器 1
Fliter2	选择过滤器 2
Fliter3	选择过滤器 3
Fliter4	选择过滤器 4
Fliter5	选择过滤器 5
Fliter6	选择过滤器 6

Fliter7	选择过滤器 7

CANFliterFrame

CAN过滤帧格式设置位，参阅Table 599获得这个参数的所有成员。

Table 599

CANROMSelect	描述
Fliter_Standard	只接收标准帧
Fliter_Extension	只接收扩展帧
Fliter_Default	不做帧格式过滤

使用示例:

```
//配置过滤器0
CAN_FliterTypeDef CAN_FliterStruct; //定义过滤器结构体
CAN_FliterStruct.Acode = 0x111; //设置过滤id 0x111
CAN_FliterStruct.Amask = 0x000; //设置过滤掩码
CAN_FliterStruct.CANFliterFrame = Fliter_Default;//过滤器帧格式不设限制
CAN_FliterStruct.CAN_Fliter = Fliter0; //配置过滤器0
CAN_FliterInit( &CAN_FliterStruct);
```

CAN_ModeSelect

Table 600

函数名	CAN_ModeSelect
函数原型	void CAN_ModeSelect(CAN_TypeDef *CANx, ModSelect_TypeDef CAN_MOD);
功能描述	配置 CAN 工作模式
输入参数 1	CANx: 来选择 CAN 外设
输入参数 2	CAN_MOD：配置 CAN 的工作模式
返回值	无

CAN_MOD为CAN工作模式配置位，参阅Table 584获得这个参数的所有成员

使用示例:

```
/ *配置 CAN 工作模式为普通模式*/
CAN_ModeSelect(CAN, CAN_Normal);
```

CAN_Trans_Init

Table 601

函数名	CAN_Trans_Init
函数原型	void CAN_Trans_Init(CAN_TypeDef *CANx, CanTxMsg *CAN_Tx_msg)
功能描述	CAN 发送的数据初始化，发送数据按 CanTxMsg 中的每一个参数按缺省值填入
输入参数 1	CANx: 来选择 CAN 外设
输入参数 2	CAN_Tx_msg:指向结构 CanTxMsg 的指针，包含了 CAN_Tx_msg 的配置信息
返回值	无

CanTxMsg

CanTxMsg 定义于文件“sc32f1xxx_can.h”：

```
typedef struct
{
uint32_t CAN_TXID;
uint32_t Tx_msg[8];
uint16_t Data_len;
uint8_t CAN_RTR;
uint8_t CAN_FDF;
uint8_t CAN_BRS;
uint8_t CAN_IDE;
uint8_t Tx_msg_len;
}CanTxMsg;
```

CAN_RTR

CAN发送数据是否为远程帧设定，参阅Table 602获得这个参数的所有成员。

Table 602

CAN_RTR	描述
CAN_Data_frame	发送数据为数据帧
CAN_Remote_frame	发送数据为远程帧

CAN_FDF

CAN发送数据是否为标准帧设定，参阅Table 603获得这个参数的所有成员。

Table 603

CAN_FDF	描述
CAN_Standard_frame	发送数据为标准帧
CAN_FD_frame	发送数据为 FD 帧

CAN_BRS

CAN发送数据是否为可变波特率，参阅Table 604获得这个参数的所有成员。

Table 604

CAN_BRS	描述
CAN_Disable_BRS	发送数据为不可变波特率
CAN_Enable_BRS	发送数据为可变波特率

CAN_IDE

CAN发送数据ID是否为扩展格式设定，参阅Table 605获得这个参数的所有成员。

Table 605

CANTSMODESelect	描述
CAN_Standard_format	数据发送 ID 为标准格式
CAN_Extended_format	数据发送 ID 为扩展格式

使用示例:

```
/*配置发送数据信息*/
CanTxMsg CAN_Tx_msg;                        //定义CAN发送结构体
CAN_Tx_msg.CAN_IDE = CAN_Extended_format;  //选择拓展帧
CAN_Tx_msg.CAN_RTR = CAN_Data_frame;                //选择数据帧
CAN_Tx_msg.CAN_BRS = CAN_Disable_BRS;       //选择不变波特率
CAN_Tx_msg.CAN_FDF = CAN_FD_frame;                  //选择CAN_FD帧格式
CAN_Tx_msg.CAN_TXID = 0x111;                        //配置发送帧id
CAN_Tx_msg.Data_len = CAN_DLC_16;                   //配置发送数据长度,配置DLC长度
CAN_Tx_msg.Tx_msg[0] = 0x12345678;                  //写入待发送数据
CAN_Tx_msg.Tx_msg[1] = 0x01020304;
CAN_Tx_msg.Tx_msg[2] = 0x05060708;
CAN_Tx_msg.Tx_msg[3] = 0x12345678;
CAN_Tx_msg.Tx_msg_len = 4;                          //使用TXBUF个数
CAN_Trans_Init(CAN, &CAN_Tx_msg);                   //CAN发送初始化
```

CAN_TransStop

Table 606

函数名	CAN_TransStop
函数原型	void CAN_TransStop(CAN_TypeDef *CANx)
功能描述	设置 CAN STB 传输中止
输入参数 1	CANx: 来选择 CAN 外设
返回值	无

使用示例:

```
/ * STB传输中止*/
CAN_Trans_Select(CAN);
```

CAN_RecieveConfig

Table 607

函数名	CAN_RecieveConfig
函数原型	void CAN_RecieveConfig(CAN_TypeDef *CANx, CanRxMsg* RxMessage)
功能描述	CAN 数据接收读取
输入参数 1	CANx: 来选择 CAN 外设
输入参数 2	RxMessage:指向结构 CanRxMsg 的指针，可读取 CanRxMsg 配置信息
返回值	无

CanRxMsg

CanRxMsg 定义于文件“sc32f1xxx_can.h”：

```
typedef struct
{
uint32_t CAN_RXID; /*!< This member is the ID of the CAN receive data */
uint32_t Rx_msg[8];/*!< This member is the receive data of CAN*/
uint8_t CAN_IDE; /*!< This member is The frame format of the received data */
uint16_t Data_len;/*!< This member is the length of the receive data of CAN*/
uint8_t CAN_RTR; /*!< This member is The frame format of the received data*/
uint8_t CAN_FDF; /*!< This member configures whether the received data of CAN is FD*/
uint8_t CAN_BRS;  /*!<This member Gets whether the baud rate of the received data is variable */
} CanRxMsg;
```

使用示例:

```
/*读取CAN接收的数据信息*/
CAN_RecieveConfig(CAN,&RxMessage);//接收
printf("%x\r\n",RxMessage.CAN_RXID);
for(int num=0;num<=15;num++)
printf("%x\r\n",RxMessage.Rx_msg[num]);
```

CAN_GetRTSL

Table 608

函数名	CAN_GetRTSL
函数原型	uint32_t CAN_GetRTSL(CAN_TypeDef *CANx)
功能描述	获取 CAN 接收时间戳的低位
输入参数	CANx: 可以选择 CAN
返回值	无

使用示例:

```
/ *获取CAN接收时间戳的低位*/
CAN_GetRTSL (CAN);
```

CAN_GetRTSH

Table 609

函数名	CAN_GetRTSH
函数原型	uint32_t CAN_GetRTSH(CAN_TypeDef *CANx)
功能描述	获取 CAN 接收时间戳的高位
输入参数	CANx: 可以选择 CAN
返回值	无

使用示例:

```
/ *获取CAN接收时间戳的高位*/
CAN_GetRTSH (CAN);
```

CAN_GetTTSL

Table 610

函数名	CAN_GetTTSL
函数原型	uint32_t CAN_GetTTSL(CAN_TypeDef *CANx)
功能描述	获取 CAN 发送时间戳的低位

输入参数	CANx: 可以选择 CAN
返回值	无

使用示例:

```
/ *获取CAN发送时间戳的低位*/
CAN_GetTTSL (CAN);
```

CAN_GetTTSH

Table 611

函数名	CAN_GetTTSH
函数原型	uint32_t CAN_GetTTSH(CAN_TypeDef *CANx)
功能描述	获取 CAN 发送时间戳的高位
输入参数	CANx: 可以选择 CAN
返回值	无

使用示例:

```
/ *获取CAN发送时间戳的高位*/
CAN_GetTTSH (CAN);
```

CAN_ITConfig

Table 612

函数名	CAN_ITConfig
函数原形	void CAN_ITConfig(CAN_TypeDef* CANx, uint32_t CAN_IT, FunctionalState NewState)
功能描述	使能或者失能指定的 CAN 中断
输入参数 1	CANx: 来选择 CAN 外设
输入参数 2	CAN_IT：待使能或者失能的 CAN 中断源
输入参数 3	NewState：CAN 中断的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

CAN_IT

CAN_IT请求选择，参阅Table 613获得这个参数的所有成员。

Table 613

CAN_IT	描述
RIEEN	接收中断请求
ROIEEN	接收溢出中断请求
RFIEEN	接收 RB 已满中断请求
RAFIEEN	接收超过或等于接收阈值中断请求
TPIEEN	PTB 发送完成中断请求
TSIEEN	STB 发送完成中断请求
EIEEN	错误中断请求
ALIEEN	传输终止中断请求
BEIEEN	总线错误中断请求
EPIEEN	被动错误中断请求

使用示例:

```
/* 使能CAN外设总接收中断 */
CAN_ITConfig(CAN, RIEEN,ENABLE);
```

CAN_GetFlagStatus

Table 614

函数名	CAN_GetFlagStatus
函数原形	ITStatus CAN_GetFlagStatus(CAN_TypeDef *CANx, uint32_t CAN_Flag)
功能描述	获取 CAN 中断标志位状态
输入参数 1	CANx: 来选择 CAN 外设
输入参数 2	CAN_Flag：获取 CAN_Flag 中断标志位的状态

返回值	标志位状态

CAN_Flag

CAN中断标志选择位，参阅Table 615获得这个参数的所有成员。

Table 615

CAN_IT	描述
RIF_FIAG	接收中断标志位
TSFF_FLAG	发送区域满标志位
AIF_FLAG	传输中止标志位
EIF_FLAG	错误中断标志位
TSIF_FLAG	PTB 发送完成标志位
TPIF_FLAG	STB 发送完成标志位
RAFIF_FLAG	阈值警告标志位
RFIF_FLAG	RB 已满标志位
ROIF_FLAG	RB 溢出标志位
ALIF_FLAG	仲裁丢失中断标志
BEIF_FLAG	总线错误中断标志
EPIF_FLAG	错误被动中断标志

使用示例:

```
/* 获取接收中断标志状态 */
CAN_GetFlagStatus(CAN, CAN_RTIE_RIF);
```

CAN_ClearFlag

Table 616

函数名	CAN_ClearFlag
函数原形	void CAN_ClearFlag(CAN_TypeDef *CANx, uint32_t CAN_Flag)
功能描述	清除 CAN 中断标志位
输入参数 1	CANx: 来选择 CAN 外设
输入参数 2	CAN_Flag：清除 CAN_Flag 中断标志位
返回值	无

CAN_Flag中断标志选择位，参阅Table 615获得这个参数的所有成员。

使用示例:

```
/* 清除CAN接收中断标志 */
CAN_ClearFlag (CAN, CAN_RTIE_RIF);
```

CAN_GetTECNT

Table 617

函数名	CAN_GetTECNT
函数原形	uint8_t CAN_GetTECNT(CAN_TypeDef* CANx)
功能描述	获取 CAN 传输错误数据量
输入参数	CANx: 来选择 CAN 外设
返回值	CAN 传输错误数据量

使用示例:

```
/* 获取 CAN 传输错误数据量 */
CAN_GetTECNT (CAN);
```

CAN_GetRECNT

Table 618

函数名	CAN_GetRECNT
函数原形	uint8_t CAN_GetRECNT(CAN_TypeDef* CANx)
功能描述	获取 CAN 接收错误数据量

输入参数	CANx: 来选择 CAN 外设
返回值	CAN 接收错误数据量

使用示例:

```
/* 获取 CAN 接收错误数据量 */
CAN_GetRECNT (CAN);
```

CAN_ResetCmd

Table 619

函数名	CAN_ResetCmd
函数原形	void CAN_ResetCmd(CAN_TypeDef *CANx, FunctionalState NewState)
功能描述	复位 CAN 外设。
输入参数 1	CANx: 来选择 CAN 外设
输入参数 2	NewState: 外设 CAN 复位状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* 复位CAN外设 */
CAN_ResetCmd(CAN, ENABLE);
```

、

综合使用示例

```
void CAN_Transmit(void)
{
CAN_TBUFSelect(CAN, CANTBUF_Tx_STB);//当前报文写入区域选择STB
/*配置发送数据信息*/
CanTxMsg CAN_Tx_msg; //定义CAN发送结构体
CAN_Tx_msg.CAN_IDE = CAN_Extended_format;//选择拓展帧
CAN_Tx_msg.CAN_RTR = CAN_Data_frame;//选择数据帧
CAN_Tx_msg.CAN_BRS = CAN_Disable_BRS;//选择不变波特率
CAN_Tx_msg.CAN_FDF = CAN_FD_frame;//选择CAN_FD帧格式
CAN_Tx_msg.CAN_TXID = 0x111;//配置发送帧id
CAN_Tx_msg.Data_len = CAN_DLC_16;//配置发送数据长度,配置DLC长度
CAN_Tx_msg.Tx_msg[0] = 0x12345678;//写入待发送数据
CAN_Tx_msg.Tx_msg[1] = 0x01020304;
CAN_Tx_msg.Tx_msg[2] = 0x05060708;
CAN_Tx_msg.Tx_msg[3] = 0x12345678;
CAN_Tx_msg.Tx_msg_len = 4;//使用TXBUF个数
CAN_Trans_Init(CAN, &CAN_Tx_msg);//CAN发送初始化

CAN_Trans_Select(CAN,CAN_TransMod_TSONE);//传输STB中的一帧报文
while(CAN->CAN_CFG_STAT & CAN_CFG_STAT_TACTIVE);//等待发送结束
for(int time = 0;time <=10000;time++);
}

//CAN过滤器配置
void CAN_FliterConfig(void)
{
//配置过滤器0
CAN_FliterTypeDef CAN_FliterStruct;//定义过滤器结构体
CAN_FliterStruct.Acode = 0x111;//设置过滤id 0x111
CAN_FliterStruct.Amask = 0x000;//设置过滤掩码
CAN_FliterStruct.CANFliterFrame = Fliter_Default;//过滤器帧格式不设限制
```

```
CAN_FliterStruct.CAN_Fliter = Fliter0;//配置过滤器0
CAN_FliterInit(&CAN_FliterStruct);
//配置过滤器1
CAN_FliterStruct.Acode = 0x222;//设置过滤id 0x222
CAN_FliterStruct.Amask = 0x000;//设置过滤掩码
CAN_FliterStruct.CANFliterFrame = Fliter_Default;//过滤器帧格式不设限制
CAN_FliterStruct.CAN_Fliter = Fliter1;//配置过滤器1
CAN_FliterInit(&CAN_FliterStruct);
}
void CAN_Communication(void)

{
RCC_Unlock(0xff); //解锁RCC
RCC_HIRCCmd(ENABLE ); //使能HIRC时钟
RCC_SYSCLKConfig(RCC_SYSCLKSource_HIRC); //设置HIRC为系统时钟,SYSCLK=32Mhz
RCC_HCLKConfig(RCC_SYSCLK_Div1); //HCLK = SYSCLK/1
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_CAN,ENABLE); //使能CAN时钟
GPIO_InitTypeDef GPIOC_InitStruct; //定义GPIOC结构体
GPIOC_InitStruct.GPIO_Pin  = GPIO_Pin_6 | GPIO_Pin_7; //选择PC06 PC07口
GPIOC_InitStruct.GPIO_Mode = GPIO_Mode_IN_PU; //上拉输入
GPIO_Init(GPIOC,&GPIOC_InitStruct); //初始化

CAN_FliterConfig();//过滤器初始化
CAN_InitTypeDef CAN_InitStruct; //定义CAN初始化结构体
CAN_StructInit(&CAN_InitStruct);
CAN_FBaudRate_Select(Baudrate_250kHz);//快速时钟波特率为250k
CAN_SBaudRate_Select(Baudrate_250kHz);//慢速时钟波特率为250k
CAN_InitStruct.CANFDFrame = FD_Mode_on;//CAN_FD帧
CAN_InitStruct.CANFDISOSelect = CAN_ISO;//采用ISO CAN_FD模式
CAN_InitStruct.CANTBUFSelect = CANTBUF_Tx_STB;//采用次级发送缓存
CAN_InitStruct.CANROMSelect = CANROM_Old;//丢弃老报文
CAN_InitStruct.CANTSMODESelect = TX_FIFO;//STB采用FIFO发送模式
CAN_InitStruct.CAN_MOD = CAN_Normal;//正常模式
CAN_InitStruct.CANTXEN = ENABLE; //CAN TX使能
CAN_InitStruct.CANRXEN = ENABLE; //CAN RX使能
CAN_Init(CAN, &CAN_InitStruct);
CAN_RxThresholdConfig(CAN, 3) ;
CAN_ITConfig(CAN, RIEEN|ROIEEN|TSIEEN |RAFIEEN ,ENABLE);
NVIC_EnableIRQ(CAN_IRQn);

CAN_Transmit();//发送函数
printf("Sending Success!\r\n");
while(1)
{
 if(CAN_Flag == SET)
{
for(int i=0;i<3;i++)
{
CAN_RecieveConfig(CAN,&RxMessage);//接收
printf("第%d个数据",i+1);
printf("%x\r\n",RxMessage.CAN_RXID);
for(int num=0;num<=15;num++)
printf("%x\r\n",RxMessage.Rx_msg[num]);
}
CAN_Flag = RESET;
   }
  }
}
```

```
void CAN_IRQHandler (void)
{
if(CAN->CAN_RTIE & CAN_RTIE_RIF) //接收中断
{
   if(CAN_GetFlagStatus(CAN, CAN_RTIE_RIF) == SET)
      {
      CAN_ClearFlag(CAN, CAN_RTIE_RIF);
      }
}

if(CAN->CAN_RTIE & CAN_RTIE_RAFIF)          //RB slots阈值警告中断
{
   if(CAN_GetFlagStatus(CAN, CAN_RTIE_RAFIF) == SET)
   {
      CAN_Flag = SET;
   CAN_ClearFlag(CAN, CAN_RTIE_RAFIF);
   }
}

if(CAN->CAN_RTIE & CAN_RTIE_TPIF)          //PTB传输完成中断
{
   if(CAN_GetFlagStatus(CAN, CAN_RTIE_TPIF) == SET)
   {
   CAN_ClearFlag(CAN, CAN_RTIE_TPIF);
   }
}
if(CAN->CAN_RTIE & CAN_RTIE_TSIF)          //STB传输完成中断
{
   if(CAN_GetFlagStatus(CAN, CAN_RTIE_TSIF) == SET)
   {
   CAN_ClearFlag(CAN, CAN_RTIE_TSIF);
   }
}
}
```

27 VREF

赛元SC32F15Gx有内部基准电压，可通过配置给ADC，DAC，CMP，OP提供基准电压。

VREF 寄存器结构

VREF寄存器结构，VREF_TypeDef，在文件“SC32F15Gx.h”中定义如下：

```
typedef struct
{
  __IO uint32_t VREF_CFG;
} VREF_TypeDef;
```

VREF 固件库函列表

Table 620

函数名	描述
Vref_LevelConfig	设置 Vref 参考电压
Vref_VMIDCmd	使能 VMID 端口
Vref_DIVENCmd	使能内部基准分压电路

VREF 固件库函数详解

Vref_LevelConfig

Table 621

函数名	Vref_LevelConfig
函数原型	ErrorStatus Vref_LevelConfig(VREF_TypeDef* Vrefx, Vref_ReferenceVoltage_TypeDef Vref_ReferenceVoltage)
功能描述	设置 Vref 参考电压
输入参数 1	Vrefx: 来选择 Vref 外设
输入参数 2	Vref_ReferenceVoltage：Vref 参考电压选择
返回值	无

Vref_ReferenceVoltage

Vref_ReferenceVoltage设置Vref参考电压，参阅Table 622获得这个参数的所有成员。

Table 622

Vref_ReferenceVoltage	描述
Vref_ReferenceVoltage_Default	电压参考电平无
Vref_ExReferenceVoltage	参考电压为外接基准
Vref_InReferenceVoltageEx_2_048	电压参考电平 2.048
Vref_InReferenceVoltageEx_1_024	电压参考电平 1.024
Vref_InReferenceVoltageEx_2_4	电压参考电平 2.4
Vref_ReferenceVoltage_2_048	电压参考电平 2.048
Vref_ReferenceVoltage_1_024	电压参考电平 1.024
Vref_ReferenceVoltage_2_4	电压参考电平 2.4

使用示例:

```
/*设置Vref参考电压*/
Vref_LevelConfig(Vref, Vref_ReferenceVoltage_2_048);
```

Vref_VMIDCmd

Table 623

函数名	Vref_VMIDCmd
函数原形	void Vref_VMIDCmd(VREF_TypeDef* Vrefx, FunctionalState NewState)

功能描述	使能 VMID 端口
输入参数 1	Vrefx: 来选择 Vref 外设
输入参数 2	NewState: 外设 VMID 端口 的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* 使能VMID 端口 */
Vref_VMIDCmd(Vref, ENABLE);
```

Vref_DIVENCmd

Table 624

函数名	Vref_DIVENCmd
函数原形	void Vref_DIVENCmd(VREF_TypeDef* Vrefx, FunctionalState NewState)
功能描述	使能或者失能内部基准分压电路
输入参数 1	Vrefx: 来选择 Vref 外设
输入参数 2	NewState: 内部基准分压 的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/* 使能内部基准分压 */
Vref_DIVENCmd (Vref, ENABLE);
```

28 Temper 固件库函数

赛元SC32F15Gx系列有内部温度传感器，可通过ADC检测内部温度传感器电压。

Temper 寄存器结构

Temper寄存器结构，TS_TypeDef，在文件“SC32F15Gx.h”中定义如下：

```
typedef struct
{
  __IO uint32_t TS_CFG;

} TS_TypeDef;
```

Temper 固件库函列表

Table 625

函数名	描述
TemperSensor_DeInit	将外设 Temper 寄存器重设为缺省值
TemperSensor_Cmd	使能温度传感器
TemperSensor_OffSetCmd	使能温度传感器抵消 offset （写 1）
TemperSensor_OffResetCmd	使能温度传感器抵消 offset （写 0）
TemperSensor_ReadValue	获取温度传感器的校验值

Temper 固件库函数详解

TemperSensor_DeInit

Table 626

函数名	TemperSensor_DeInit
函数原型	void TemperSensor_DeInit(void)
功能描述	将外设 Temper寄存器重设为缺省值
返回值	无

使用示例:

```
/*将外设 Temper寄存器重设为缺省值*/
TemperSensor_DeInit(void);
```

TemperSensor_Cmd

Table 627

函数名	TemperSensor_Cmd
函数原型	void TemperSensor_Cmd(TS_TypeDef* TSx, FunctionalState NewState)
功能描述	使能温度传感器
输入参数 1	TSx: 来选择 TS 外设
输入参数 2	NewState: 外设 TS 的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/*使能温度传感器 */
TemperSensor_Cmd (TS, ENABLE);
```

TemperSensor_OffSetCmd

Table 628

函数名	TemperSensor_OffSetCmd
函数原形	void TemperSensor_OffSetCmd (TS_TypeDef* TSx)
功能描述	使能温度传感器抵消 offset（写 1）
输入参数	TSx: 来选择 TS 外设
返回值	无

使用示例:

```
/* 使能温度传感器抵消offset */
TemperSensor_OffSetCmd (Vref);
```

TemperSensor_OffResetCmd

Table 629

函数名	TemperSensor_OffResetCmd
函数原形	void TemperSensor_OffResetCmd(TS_TypeDef* TSx)
功能描述	使能温度传感器抵消 offset（写 0）
输入参数	TSx: 来选择 TS 外设
返回值	无

使用示例:

```
/* 使能温度传感器抵消offset */
TemperSensor_OffResetCmd (Vref);
```

TemperSensor_ReadValue

Table 630

函数名	TemperSensor_ReadValue
函数原形	uint16_t TemperSensor_ReadValue(void)
功能描述	获取温度传感器校验值
输入参数	无
返回值	返回温度传感器校验值

使用示例:

```
/* 返回温度传感器校验值 */
uint16_t read;
read = TemperSensor_ReadValue ();
```

综合使用示例

```
void TemperatureSensor_Function(void)
{
  uint8_t TemperSensorCnt = 0;
  uint16_t ADC_TemperValue = 0;
  uint16_t ADC_TemperValue0 = 0;
  uint16_t ADC_TemperValue1 = 0;

  RCC_APB2Cmd(ENABLE);                          //ADC外设挂载在APB2时钟上
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC,ENABLE);  //开启ADC外设时钟

  /*Vref基准源配置*/
  Vref_LevelConfig(VREF,Vref_ReferenceVoltage_2_4);   //配置基准源的参考电压为2.4V
  Vref_ChannelConfig(VREF,Vref_InReferenceVoltage);   //内部基准电压使能

  /*ADC初始化*/
  ADC_InitTypeDef ADC_InitStruct; //定义ADC初始化结构体变量
  ADC_StructInit(&ADC_InitStruct);
```

```
    ADC_InitStruct.ADC_ConvMode = ADC_ConvMode_Single;      //ADC单次转换模式
    ADC_InitStruct.ADC_Prescaler = ADC_Prescaler_60CLOCK;   //转换时间为60个周期
    ADC_InitStruct.ADC_SPMODE = ADC_SampleMode_Single;      //设置ADC单采样模式
    ADC_InitStruct.ADC_VREF = ADC_RefSource_VREF;           //参考电压为VDD
    ADC_InitStruct.ADC_EAIN = ADC_EAIN_Less;                //把AIN0设为ADC输入,并自动将上拉电阻移除
    ADC_Init(ADC, &ADC_InitStruct);                         //根据结构体标志位配置ADC

    ADC_SetChannel(ADC,ADC_Channel_TEMP);   //设置采样通道为温度传感器

    ADC_Cmd(ADC, ENABLE);   //使能ADC

    TemperSensor_Cmd(TS,ENABLE);    //使能温度传感器
    while(1)
    {
       if(TemperSensorCnt == 0)
       {
          TemperSensorCnt++;
          TemperSensor_OffResetCmd (TS);   //转化第一次温度传感器值
          ADC_SoftwareStartConv(ADC); //触发ADC转换
          ADC_TemperValue0 = ADC_GetConversionValue(ADC);
       }
       else if(TemperSensorCnt == 1)
       {
          TemperSensorCnt++;
         TemperSensor_OffSetCmd (TS);    //转化第二次温度传感器值
          ADC_SoftwareStartConv(ADC); //触发ADC转换
          ADC_TemperValue1 = ADC_GetConversionValue(ADC);
       }
       else
       {
          TemperSensorCnt = 0;
          ADC_TemperValue = (ADC_TemperValue0 + ADC_TemperValue1)/2;
          printf("获取当前芯片温度为:0x%x\r\n",ADC_TemperValue);
       }
       for(uint16_t i=0;i<65535;i++)
      {
        for(uint16_t j=0;j<30;j++);
      }
    }
}
```

29 WDT 固件库函数

SC32F1xxx系列内建一个独立的硬件看门狗WDT，其时钟源为内部的32kHz振荡器LIRC。用户可以通过编程器的Customer Option中的ENWDT控制位选择是否开启看门狗复位功能。

硬件看门狗WDT，具有安全性高、定时准确及使用灵活的优点。此看门狗外设可检测并解决由软件错误导致的故障，并在计数器达到给定的溢出时间时触发系统复位。

WDT由其内部低频振荡器驱动，因此即便在主时钟发生故障时仍然保持工作状态。

WDT 寄存器结构

WDT 寄存器结构，WDT_TypeDef，在文件“sc32f1xxx.h”中定义如下：

```
typedef struct
{
__IO uint32_t RESERVED0[3];
__IO uint32_t WDT_CON;
__IO uint32_t WDT_CFG;
} WDT_TypeDef;
```

WDT 寄存器表格

Table 631

寄存器	描述
WDTCON	WDT控制寄存器
WDTCFG	WDT设置寄存器

WDT 固件库函列表

Table 632

函数名	描述
WDT_DeInit	WDT 相关寄存器复位至缺省值
WDT_SetOverTime	设置 WDT 的复位时间
WDT_Cmd	WTD 功能启动/关闭选择
WDT_SetReload	WDT 喂狗

WDT 固件库函数详解

WDT_DeInit

Table 633

函数名	WDT_DeInit
函数原型	void WDT_DeInit(void)
功能描述	WDT 相关寄存器复位至缺省值
输入参数	无
返回值	无

使用示例:

```
/* WDT寄存器设置为复位值 */
WDT_DeInit();
```

WDT_SetOverTime

Table 634

函数名	WDT_SetOverTime
函数原型	void WDT_SetOverTime (WDT_OverTime_TypeDef WDT_OverTime)
功能描述	WDT 初始化配置函数
输入参数	WDT_OverTime：WDT 溢出时间选择
返回值	无

OverTime

OverTime设置WDT的复位时间，参阅Table 635获得这个参数的所有成员。

Table 635

WDT_OverTime	描述
WDT_OverTime_500MS	WDT 复位时间为 500mS
WDT_OverTime_250MS	WDT 复位时间为 250MS
WDT_OverTime_125MS	WDT 复位时间为 125MS
WDT_OverTime_62_5MS	WDT 复位时间为 62.5mS
WDT_OverTime_31_5MS	WDT 复位时间为 31.5mS
WDT_OverTime_15_75MS	WDT 复位时间为 15.75mS
WDT_OverTime_7_88MS	WDT 复位时间为 7.88mS
WDT_OverTime_3_94MS	WDT 复位时间为 3.94mS

使用示例:

```
/* 设置WDT溢出时间为500MS */
WDT_SetOverTime (WDT_OverTime_500MS);
```

WDT_Cmd

Table 636

函数名	WDT_Cmd
函数原型	void WDT_Cmd(FunctionalState NewState)
功能描述	WTD 功能启动/关闭选择
输入参数	NewState：WDT 功能开启/关闭，可取值 ENABLE 或 DISABLE
返回值	无

使用示例:

```
/* WDT使能 */
WDT_Cmd(ENABLE);
```

WDT_SetReload

Table 637

函数名	WDT_SetReload
函数原型	void WDT_SetReload(void)
功能描述	看门狗喂狗，计数值清 0
输入参数	无
返回值	无

使用示例:

```
WDT_SetReload(); //喂狗
```

综合使用示例（设置 WDT 为 500mS）

```
WDT_SetOverTime(WDT_OverTime_500MS);
while(1)
{
  WDT_SetReload();
}
```

30 PWR 固件库函数

PWR 固件库函列表

Table 638

函数名	描述
PWR_EnterIDLEMode	进入 IDEL 模式
PWR_EnterSTOPMode	进入 STOP 模式

PWR 固件库函数详解

PWR_EnterIDLEMode

Table 639

函数名	PWR_EnterIDLEMode
函数原型	void PWR_EnterIDLEMode(uint8_t PWR_IDLEEntry)
功能描述	进入 IDEL 模式
输入参数	PWR_SLEEPEntry 进入方式选择
返回值	无

PWR_IDLEEntry

PWR_IDLEEntry设置进入IDLE的方式，参阅Table 640获得这个参数的所有成员。

Table 640

PWR_IDLEEntry	描述
PWR_IDLEEntry_WFI	进入 WFI 低功耗
PWR_IDLEEntry_WFE	进入 WFE 低功耗

使用示例:

```
PWR_EnterIDLEMode(PWR_IDLEEntry_WF);
```

PWR_EnterSTOPMode

Table 641

函数名	PWR_EnterSTOPMode
函数原型	void PWR_EnterSTOPMode(uint8_t PWR_STOPEntry)
功能描述	进入 STOP 模式
输入参数	PWR_STOPEntry 进入方式选择
返回值	无

PWR_STOPEntry

PWR_STOPEntry设置进入STOP的方式，参阅Table 642获得这个参数的所有成员。

Table 642

PWR_STOPEntry	描述
PWR_STOPEntry_WFI	进入 WFI 低功耗
PWR_STOPEntry_WFE	进入 WFE 低功耗

使用示例:

```
PWR_EnterSTOPMode(PWR_STOPEntry_WFI);
```

31 NVIC 固件库函数

21.1 NVIC 固件库函列表

Table 643

函数名	描述
NVIC_SetPriorityGrouping	配置优先级分组
NVIC_GetPriorityGrouping	获取优先级分组
NVIC_EnableIRQ	中断控制器启用中断
NVIC_GetEnableIRQ	获取控制器启用的中断
NVIC_DisableIRQ	中断控制器禁用中断
NVIC_GetPendingIRQ	获取特定于设备的中断源的中断位挂起
NVIC_SetPendingIRQ	设置外设中断的挂起位
NVIC_ClearPendingIRQ	清除外设中断的挂起位
NVIC_SetPriority	设置外设中断的优先级
NVIC_GetPriority	获取外设中断的优先级
NVIC_SystemReset	启动系统重置请求

21.2 NVIC 固件库函数详解

NVIC_SetPriorityGrouping

Table 644

函数名	NVIC_SetPriorityGrouping
函数原型	__NVIC_SetPriorityGrouping(X) (void)(X);
功能描述	配置优先级分组
输入参数	X：设置优先级分组
返回值	无

使用示例:

```
NVIC_SetPriorityGrouping (0);//设置优先级分组为0
```

NVIC_GetPriorityGrouping

Table 645

函数名	NVIC_GetPriorityGrouping
函数原型	__NVIC_GetPriorityGrouping() (0U)
功能描述	获取优先级分组
输入参数	无
返回值	无

使用示例:

```
NVIC_GetPriorityGrouping ();//获取优先级分组
```

NVIC_EnableIRQ

Table 646

函数名	NVIC_EnableIRQ
函数原型	__NVIC_EnableIRQ(IRQn_Type IRQn)
功能描述	中断控制器启用中断
输入参数	IRQn：外设中断设置
返回值	无

IRQn 参数的所有成员

IRQn设置外设外部中断，参阅Table 647获得这个参数的所有成员。

Table 647

IRQn_Type	描述
INT0_IRQn	外部中断 0
INT1_7_IRQn	外部中断 1-7
INT8_11_IRQn	外部中断 8-11
INT12_15_IRQn	外部中断 12-15
RCC_IRQn	RCC 时钟中断
BTM_IRQn	低频时钟中断
UART0_2_4_IRQn	串口 0/2/4 中断
UART1_3_5_IRQn	串口 1/3/5 中断
SPI0_IRQn	SPI0 中断
SPI1_IRQn	SPI1 中断
DMA0_IRQn	DMA0 中断
DMA1_IRQn	DMA1 中断
DMA2_IRQn	DMA2 中断
DMA3_IRQn	DMA3 中断
TIMER0_IRQn	定时器 0 中断
TIMER1_IRQn	定时器 1 中断
TIMER2_IRQn	定时器 2 中断
TIMER3_IRQn	定时器 3 中断
TIMER4_5_IRQn	定时器 4/5 中断
TIMER6_7_IRQn	定时器 6/7 中断
PWM0_IRQn	PWM0 中断
…	…
TWI0_IRQn	TWI0 中断
TWI1_IRQn	TWI1 中断
ADC_IRQn	模数转换中断
CMP_IRQn	模拟比较器中断
CAN_IRQHandler	CAN 通信中断
TK_IRQn	触控中断

（不同芯片型号中断不一样，具体选择需以规格书为准）

使用示例:

```
NVIC_EnableIRQ (INT0_IRQn);//使能INT0中断
```

NVIC_GetEnableIRQ

Table 648

函数名	NVIC_GetEnableIRQ
函数原型	__NVIC_GetEnableIRQ(IRQn_Type IRQn)
功能描述	获取控制器启用的中断
输入参数	IRQn：外设中断设置
返回值	无

参阅Table 647获得IRQn的所有成员。

使用示例:

```
NVIC_GetEnableIRQ (INT0_IRQn);//获取INT0是否使能中断
```

NVIC_DisableIRQ

Table 649

函数名	NVIC_DisableIRQ
函数原型	__NVIC_DisableIRQ(IRQn_Type IRQn)
功能描述	中断控制器禁用中断
输入参数	IRQn：外设中断设置

返回值	无

参阅Table 647获得IRQn的所有成员。

使用示例:

NVIC_DisableIRQ (INT0_IRQn);//禁用INT0中断

NVIC_GetPendingIRQ

Table 650

函数名	NVIC_GetPendingIRQ
函数原型	__NVIC_GetPendingIRQ(IRQn_Type IRQn)
功能描述	获取特定于设备的中断源的中断位挂起
输入参数	IRQn：外设中断设置
返回值	无

参阅Table 647获得IRQn的所有成员。

使用示例:

NVIC_GetPendingIRQ (INT0_IRQn);// 获取外设的中断源的中断位是否挂起

NVIC_SetPendingIRQ

Table 651

函数名	NVIC_SetPendingIRQ
函数原型	__NVIC_SetPendingIRQ(IRQn_Type IRQn)
功能描述	设置外设中断的挂起位
输入参数	IRQn：外设中断设置
返回值	无

参阅Table 647获得IRQn的所有成员。

使用示例:

NVIC_SetPendingIRQ (INT0_IRQn);// 设置外部中断的挂起位

NVIC_ClearPendingIRQ

Table 652

函数名	NVIC_ClearPendingIRQ
函数原型	__NVIC_ClearPendingIRQ(IRQn_Type IRQn)
功能描述	清除外设中断的挂起位
输入参数	IRQn：外设中断设置
返回值	无

参阅Table 647获得IRQn的所有成员。

使用示例:

NVIC_SetPendingIRQ (INT0_IRQn);// 设置外部中断的挂起位

NVIC_ClearPendingIRQ

Table 653

函数名	NVIC_ClearPendingIRQ
函数原型	__NVIC_ClearPendingIRQ(IRQn_Type IRQn)
功能描述	清除外设中断的挂起位
输入参数	IRQn：外设中断设置
返回值	无

参阅Table 647获得IRQn的所有成员。

使用示例:

NVIC_ClearPending (INT0_IRQn);// 清除外设中断的挂起位

NVIC_SetPriority

Table 654

函数名	NVIC_SetPriority
函数原型	__NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority)
功能描述	设置中断的优先级
输入参数 1	IRQn：外设中断设置
输入参数 2	priority：设置中断优先级
返回值	无

参阅获得IRQn的所有成员。

使用示例:

```
NVIC_SetPriority (INT0_IRQn ,0);//设置INT0_IRQn优先级分组为0
```

NVIC_GetPriority

Table 655

函数名	NVIC_GetPriority
函数原型	__NVIC_GetPriority(IRQn_Type IRQn)
功能描述	获取中断的优先级
输入参数	IRQn_Type IRQn：外设中断设置
返回值	无

参阅Table 647获得IRQn的所有成员。

使用示例:

```
NVIC_GetPriority (INT0_IRQn);// 获取INT0_IRQn外部中断的优先级
```

NVIC_SystemReset

Table 656

函数名	NVIC_SystemReset
函数原型	__NVIC_SystemReset(void)
功能描述	启动系统重置请求
输入参数	无
返回值	无

使用示例:

```
NVIC_SystemReset ();//系统重置请求
```

规格更改记录

版本	记录	日期
V1.6	1. 新增项 （1）Time外设PWM通道入参枚举增加TIM_PWMChannel_ALL （2）新增DAC固件库函数 2. 优化项 增加函数入参不同型号的选择说明	2024年9月
V1.5	1.新增项 （1）新增SPI1_TWI1固件库函数 （2）新增VREF固件库函数 （3）新增Temper固件库函数 （4）新增QEP固件库函数 （5）新增CAN时间戳读取函数 （6）新增OP校验值读取函数 （7）添加部分枚举变量成员的描述	2024年6月
V1.4	1.新增项 （1）新增PGA固件库函数 （2）新增CAN固件库函数	2024年3月
V1.3	1.新增项 （1）固件库新增OP固件库函数 （2）RCC增加PWM0时钟源选择函数： void RCC_PWM0CLKConfig(RCC_PWM0CLKSource_TypeDef RCC_PWM0CLKSource),	2024年1月
V1.2	1.修正项 （1）芯片型号统一，从SC32修正为SC32F1XXX 2.优化项 （1）添加部分枚举变量成员的描述 3.新增项 （1）GPIO库 新增PA_OT、PB_OT、PC_OT、PA_BIT、PB_BIT、PC_BIT位操作语句的说明 （2）CMP正端通道切换函数： void CMP_SetPositiveChannel(CMP_TypeDef* CMPx, CMP_Positive_TypeDef CMP_Positive_Channel) （3）CMP负端通道切换函数： void CMP_SetNegativeChannel(CMP_TypeDef* CMPx, CMP_Negative_TypeDef CMP_Negative_Channel) （4）CMP正端通道查询函数： CMP_Positive_TypeDef CMP_GetPositiveChannel (CMP_TypeDef* CMPx) （5）CMP负端通道查询函数： CMP_Negative_TypeDef CMP_GetNegativeChannel (CMP_TypeDef* CMPx) （6）RCC库新增晶振停振检测功能NMI函数： void RCC_EnableCSS(viod)	2023年12月
V1.1	1.修正项 （1）因GPIO模式设置由原来的两个结构体变量合并为一个，故修改GPIO初始化结构体的成员、初始化过程、枚举成员的描述 （2）INT_ClearFlag函数修改为： void INT_ClearFlag(uint32_t INT_Channel) 即将其原始入参修改为一个入参 （3）删除OPTION_StartupAreaConfig()函数 2.优化项 无 3.新增项	2023年11月

	（1）新增NVIC固件库 （2）GPIO固件库新增IO驱动等级设置函数: void GPIO_SetDriveLevel(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_DriveLevel_TypeDef GPIO_DriveLevel); （3）新增IAP单个读函数: uint8_t IAP_ReadByte(uint32_t Address) （4）批量写入32位数据：uint8_t IAP_ProgramWordArray(uint32_t Address, uint32_t* ByteArray, uint8_t ArraySize) （5）批量写入16位数据：uint8_t IAP_ProgramHalfWordArray(uint32_t Address, uint16_t* ByteArray, uint8_t ArraySize) （6）批量写入8位数据：uint8_t IAP_ProgramByteArray(uint32_t Address, uint8_t* ByteArray, uint8_t ArraySize) （7）批量读出32位数据：uint8_t IAP_ReadWordArray(uint32_t Address, uint32_t* ByteArray, uint8_t ArraySize) （8）批量读出16位数据：uint8_t IAP_ReadHalfWordArray(uint32_t Address, uint16_t* ByteArray, uint8_t ArraySize) （9）批量读出8位数据：uint8_t IAP_ReadByteArray(uint32_t Address, uint8_t* ByteArray, uint8_t ArraySize) （10）补上WDT_Cmd函数	
V1.0	初版	2023年4月

声明

深圳市赛元微电子股份有限公司（以下简称赛元）保留随时对赛元产品、文档或服务进行变更、更正、增强、修改和改进的权利，恕不另行通知。赛元认为提供的信息是准确可信的。本文档信息于 2023 年04月开始使用。在实际进行生产设计时，请参阅各产品最新的数据手册等相关资料。