第一部分：固件库概述

1 前言

本文档适用于赛元 SC32F1xxx 、SC32L1xxx、SC32Rxxxx系列的芯片。

赛元 SC32F1xxx 系列MCU的固件库，提供了一套赛元SC32F1xxx 、SC32L1xxx、SC32Rxxxx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 、SC32L1xxx、SC32Rxxxx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 、SC32L1xxx、SC32Rxxxx 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 、SC32L1xxx、SC32Rxxxx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
CAN	CAN通信
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

LPC	低功耗计数器
LPD	低电压监测模块
RTC	实时时钟模块
SmartCard	智能卡接口控制器
AES	高级加密运算模块
TRNG	真随机数发生器

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

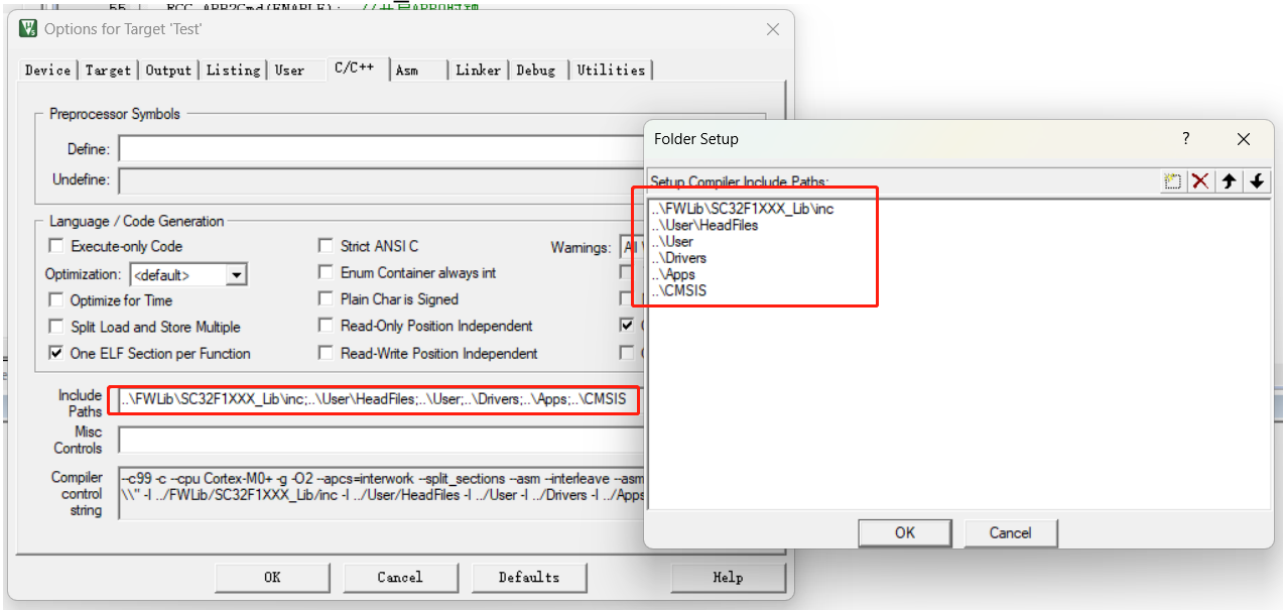
Table 2

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。

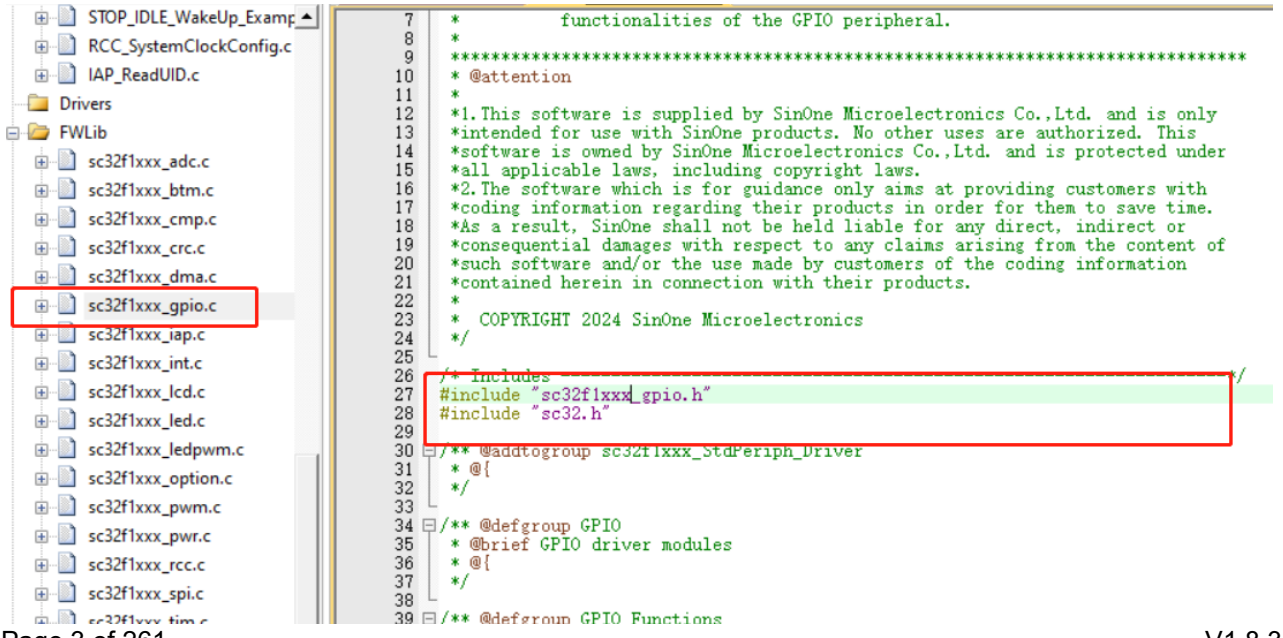
3 固件库函数调用方法

在工程相关的文件夹添加固件库SC32F1XXX_Lib。

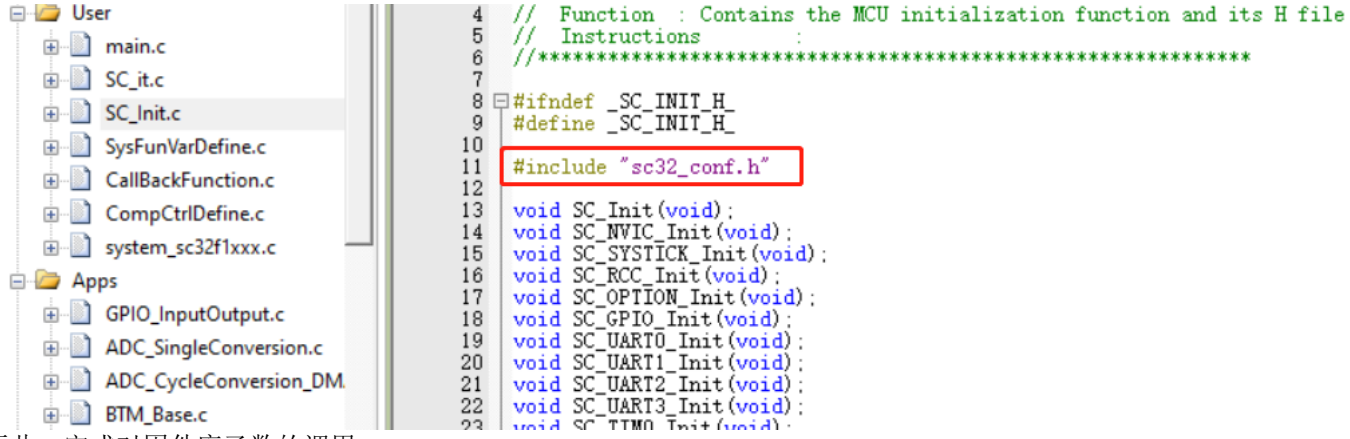
在工程内添加固件库SC32F1XXX_Lib中头文件的路径，路径指向inc文件夹、User文件夹和CMSIS文件夹。



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

4 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（入参选择参考具体型号的规格书）

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_InitStruct)
功能描述	根据ADC_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_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_9CLOCK	ADC采样时间约9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

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

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_Prescaler_3CLOCK
ADC_EAIN	ADC_EAIN_Less
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

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

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

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、SC32R601 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、SC32R601 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

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、SC32R601 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、SC32R601 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）

```
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
    }
```

5 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
}
```

6 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 固件库函数详解（入参选择参考具体型号的规格书）

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

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

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

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

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_0, 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_0);

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_0负输入通道 */
CMP_SetNegativeChannel(CMP_0, 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CMP_0正输入通道 */
CMP_SetPositiveChannel(CMP_0, 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
CMP_CMP0STA_Low	CMP0比较器正端电压小于负端电压
CMP_CMP0STA_High	CMP0比较器正端电压大于负端电压
CMP_CMP1STA_Low	CMP1比较器正端电压小于负端电压
CMP_CMP1STA_High	CMP1比较器正端电压大于负端电压
CMP_CMP2STA_Low	CMP2比较器正端电压小于负端电压
CMP_CMP2STA_High	CMP2比较器正端电压大于负端电压
CMP_CMP3STA_Low	CMP3比较器正端电压小于负端电压
CMP_CMP3STA_High	CMP3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

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 (CMP3, 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
```

7 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
```

8 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_TWI_QSPI0_TX	QSPI0_IDE-> TXDMAEN
DMA_Request_TWI_QSPI0_RX	QSPI0_IDE-> RXDMAEN
DMA_Request_TWI_QSPI1_TX	QSPI1_IDE-> TXDMAEN
DMA_Request_TWI_QSPI1_RX	QSPI1_IDE-> RXDMAEN
DMA_Request_TWI_SPI2_TX	TWI_SPI2_IDE-> TXDMAEN
DMA_Request_TWI_SPI2_RX	TWI_SPI2_IDE-> RXDMAEN
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

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
    }
}
```

9 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);
}
```

10 LCD 固件库函数

赛元SC32f10xx，SC32R803，SC32f12xx、SC32R805、SC32R806、SC32L14xx、SC32R807、SC32R808、SC32f11xx系列具有LCD显示驱动电路，可方便用户实行LCD的显示驱动。

SC32f10xx，SC32R803，SC32f12xx、SC32R805、SC32R806、SC32R808、SC32f11xx系列主要特点如下：

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。

SC32L14xx系列主要特点如下：

1. 电阻型LCD驱动与电容偏压型LCD驱动二选一；
2. 支持在STOP Mode下显示。

SC32R807系列主要特点如下：

1. 支持在STOP Mode下显示。

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_COMConfig	设置COM输出通道
LCD_SEGConfig	设置SEG输出通道
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_ SEGConfig

Table 154

函数名	LCD_ SEGConfig
函数原型	void LCD_SEGConfig (uint64_t SEGSelect, FunctionalState NewState)
功能描述	设置SEG输出通道
输入参数1	SEGSelect：SEG引脚选择
输入参数2	NewState: 外设LCD的新状态，这个参数可以取：ENABLE 或者 DISABLE
返回值	无

SEGSelect

SEGSelect选择LCD_Channel，参阅Table 155获得这个参数的所有成员。

Table 155

SEGSelect	描述
LCD_Channel_0	选择SEG0引脚输出
LCD_Channel_1	选择SEG1引脚输出
LCD_Channel_2	选择SEG2引脚输出
LCD_Channel_3	选择SEG3引脚输出
LCD_Channel_4	选择SEG4引脚输出
LCD_Channel_5	选择SEG5引脚输出
LCD_Channel_6	选择SEG6引脚输出
LCD_Channel_7	选择SEG7引脚输出
LCD_Channel_8	选择SEG8引脚输出
LCD_Channel_9	选择SEG9引脚输出
LCD_Channel_10	选择SEG10引脚输出
LCD_Channel_11	选择SEG11引脚输出
LCD_Channel_12	选择SEG12引脚输出
LCD_Channel_13	选择SEG13引脚输出
LCD_Channel_14	选择SEG14引脚输出
LCD_Channel_15	选择SEG15引脚输出
LCD_Channel_16	选择SEG16引脚输出
LCD_Channel_17	选择SEG17引脚输出
LCD_Channel_18	选择SEG18引脚输出
LCD_Channel_19	选择SEG19引脚输出
LCD_Channel_20	选择SEG20引脚输出
LCD_Channel_21	选择SEG21引脚输出
LCD_Channel_22	选择SEG22引脚输出
LCD_Channel_23	选择SEG23引脚输出
LCD_Channel_24	选择SEG24引脚输出
LCD_Channel_25	选择SEG25引脚输出
LCD_Channel_26	选择SEG26引脚输出
LCD_Channel_27	选择SEG27引脚输出

LCD_Channel_28	选择SEG28引脚输出
LCD_Channel_29	选择SEG29引脚输出
LCD_Channel_30	选择SEG30引脚输出
LCD_Channel_31	选择SEG31引脚输出
LCD_Channel_32	选择SEG32引脚输出
LCD_Channel_33	选择SEG33引脚输出
LCD_Channel_34	选择SEG34引脚输出

使用示例:

```
/*使能LCD的SEG0引脚*/
LCD_SEGConfig (LCD_Channel_0, ENABLE);
```

LCD_COMConfig

Table 156

函数名	LCD_COMConfig
函数原型	void LCD_COMConfig (LCD_COMEN_Typedef COMSelect, FunctionalState NewState)
功能描述	设置COM输出通道
输入参数1	COMSelect：COM引脚选择
输入参数2	NewState: 外设LCD的新状态，这个参数可以取：ENABLE 或者 DISABLE
返回值	无

COMSelect

COMSelect选择LCD_COMEN，参阅Table 157获得这个参数的所有成员。

Table 157

SEGSelect	描述
LCD_COMEN_0	使能COM0
LCD_COMEN_1	使能COM1
LCD_COMEN_2	使能COM2
LCD_COMEN_3	使能COM3
LCD_COMEN_4	使能COM4
LCD_COMEN_5	使能COM5
LCD_COMEN_6	使能COM6
LCD_COMEN_7	使能COM7

使用示例:

```
/*使能LCD的COM0通道*/
LCD_SEGConfig (LCD_COMEN_0, ENABLE);
```

LCD_Write

Table 158

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 159获得这个参数的所有成员。

Table 159

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

使用示例:

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

LCD_CustomModeScan

Table 160

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
```

11 LED 固件库函数

赛元SC32f10xx，SC32R803，SC32f12xx、SC32R805、SC32R806、SC32R808、SC32f11xx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 161

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 162

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_COMConfig	设置COM输出通道
LED_SEGConfig	设置SEG输出通道
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 163

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 164

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 165获得这个参数的所有成员。

Table 165

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 166获得这个参数的所有成员。

Table 166

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 167

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 168

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_ SEGConfig

Table 169

函数名	LED_ SEGConfig
函数原型	void LED_SEGConfig (uint64_t SEGSelect, FunctionalState NewState)
功能描述	设置SEG输出通道
输入参数1	SEGSelect：SEG引脚选择
输入参数2	NewState: 外设LED的新状态，这个参数可以取：ENABLE 或者 DISABLE
返回值	无

SEGSelect

SEGSelect选择LED_Channel，参阅Table 170获得这个参数的所有成员。

Table 170

SEGSelect	描述
LED_Channel_0	选择SEG0引脚输出
LED_Channel_1	选择SEG1引脚输出
LED_Channel_2	选择SEG2引脚输出
LED_Channel_3	选择SEG3引脚输出
LED_Channel_4	选择SEG4引脚输出
LED_Channel_5	选择SEG5引脚输出
LED_Channel_6	选择SEG6引脚输出
LED_Channel_7	选择SEG7引脚输出
LED_Channel_8	选择SEG8引脚输出
LED_Channel_9	选择SEG9引脚输出
LED_Channel_10	选择SEG10引脚输出
LED_Channel_11	选择SEG11引脚输出

LED_Channel_12	选择SEG12引脚输出
LED_Channel_13	选择SEG13引脚输出
LED_Channel_14	选择SEG14引脚输出
LED_Channel_15	选择SEG15引脚输出
LED_Channel_16	选择SEG16引脚输出
LED_Channel_17	选择SEG17引脚输出
LED_Channel_18	选择SEG18引脚输出
LED_Channel_19	选择SEG19引脚输出
LED_Channel_20	选择SEG20引脚输出
LED_Channel_21	选择SEG21引脚输出
LED_Channel_22	选择SEG22引脚输出
LED_Channel_23	选择SEG23引脚输出
LED_Channel_24	选择SEG24引脚输出
LED_Channel_25	选择SEG25引脚输出
LED_Channel_26	选择SEG26引脚输出
LED_Channel_27	选择SEG27引脚输出
LED_Channel_28	选择SEG28引脚输出
LED_Channel_29	选择SEG29引脚输出
LED_Channel_30	选择SEG30引脚输出
LED_Channel_31	选择SEG31引脚输出
LED_Channel_32	选择SEG32引脚输出
LED_Channel_33	选择SEG33引脚输出
LED_Channel_34	选择SEG34引脚输出

使用示例:

```
/*使能LED的SEG0通道*/
LED_SEGConfig (LED_Channel_0, ENABLE);
```

LED_COMConfig

Table 171

函数名	LED_COMConfig
函数原型	void LED_COMConfig (LED_COMEN_Typedef COMSelect, FunctionalState NewState)
功能描述	设置COM输出通道
输入参数1	COMSelect：COM引脚选择
输入参数2	NewState: 外设LCD的新状态，这个参数可以取：ENABLE 或者 DISABLE
返回值	无

COMSelect

COMSelect选择LED_COMEN，参阅 Table 172获得这个参数的所有成员。

Table 172

SEGSelect	描述
LED_COMEN_0	使能COM0
LED_COMEN_1	使能COM1
LED_COMEN_2	使能COM2
LED_COMEN_3	使能COM3
LED_COMEN_4	使能COM4
LED_COMEN_5	使能COM5
LED_COMEN_6	使能COM6
LED_COMEN_7	使能COM7

使用示例:

```
/*使能LED的COM0通道*/
```

LED_SEGConfig (LED_COMEN_0, ENABLE);

LED_Write

Table 173

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 174获得这个参数的所有成员。

Table 174

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

使用示例:

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

LED_CustomModeScan

Table 175

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
```

12 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 176

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 177

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(入参选择参考具体型号的规格书)

INT_DeInit

Table 178

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 179

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 180获得这个参数的所有成员。

Table 180

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 181获得这个参数的所有成员。

Table 181

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 182获得这个参数的所有成员。

Table 182

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

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 183

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 180
输入参数2	Trigger_Mode：选择中断触发方式，取值可参考 Table 181
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 184

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 180
输入参数2	INT_IT：选择中断触发方式，取值可参考Table 185
输入参数3	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 185获得这个参数的所有成员。

Table 185

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 186

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 180
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 187
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 187获得这个参数的所有成员。

Table 187

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 188

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 189

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 190

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);
    }
}
```

13 OP 固件库函数

赛元 SC32F12xx 、SC32R805、SC32R806、SC32R803、SC32F15Gx、SC32L14xx、SC32R807、SC32R808、SC32F11xx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 寄存器表格

Table 191

寄存器	描述
OP0_CON	OP0配置寄存器
OP1_CON	OP1配置寄存器
OP2_CON	OP2配置寄存器
OPX_CFG	OPx控制寄存器
OPX_STS	OPx中断位寄存器
OPX_IDE	OPx中断设置寄存器

OP 固件库函数列

Table 192

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_GainSelection	配置OPx运放倍数
OP_OffsetTrimConfig	配置OPAMP的微调值
OP_OutputSelection	配置OPx的输出通道
OP_OffsetSet	设置OP出厂校验值
OP_StructInit	把 OP_InitStruet 中的每一个参数按缺省值填入
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_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OPx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_OffsetZero	配置OPx的校验值
OP_GetCMPSTA	获得OPx比较器模式输出状态
OP_ITConfig	使能OPx中断
OP_GetFlagStatus	获得OPx标志状态
OP_ClearFlag	清除标志状态

OP 固件库函数详解(入参选择参考具体型号的规格书)

OP_DeInit

Table 193

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 194

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”与“sc32f1xxx_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SC32f15Gx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 195获得这个参数的所有成员。

Table 195

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 196获得这个参数的所有成员。

Table 196

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 197获得这个参数的所有成员。

Table 197

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 198获得这个参数的所有成员。

Table 198

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 199获得这个参数的所有成员。

Table 199

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 200获得这个参数的所有成员。

Table 200

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 201获得这个参数的所有成员。

Table 201

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 202获得这个参数的所有成员。

Table 202

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OP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 203获得这个参数的所有成员。

Table 203

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 204获得这个参数的所有成员。

Table 204

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 205获得这个参数的所有成员。

Table 205

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 206获得这个参数的所有成员。

Table 206

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 207获得这个参数的所有成员。

Table 207

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 208获得这个参数的所有成员。

Table 208

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 209

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 210

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_GainSelection

Table 211

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 212

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_OffsetSet（SC32f12xx、SC32R803、SC32R805、SC32R806、SC32L14xx、SC32R807、SC32R808、SC32f11xx 系列）

Table 213

函数名	OP_OffsetSet
函数原型	ErrorStatus OP_OffsetSet(OP_TypeDef* OPx)
功能描述	配置OP的offset为出厂设置
输入参数	OPx：选择OP外设
返回值	无

使用示例:

```
/ *配置OP输出端*/
OP_OffsetSet (OP);
```

OP_StructInit（SC32f15xx 系列）

Table 214

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（SC32f15xx 系列）

Table 215

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（SC32f15xx 系列）

Table 216

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_NonInvertInputSelection（SC32f15xx 系列）

Table 217

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 218

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 219

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 220

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 221获得这个类型的所有成员。

Table 221

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_OffsetZero（SC32f15xx 系列）

Table 222

函数名	OP_OffsetZero
函数原型	void OP_OffsetZero(OP_TypeDef* OPx)
功能描述	配置OPx 的正负端校验值
输入参数1	OPx_Readmos: 来选择 OPx 端口
返回值	无

使用示例:

```
/* 配置OP_0的校验值 */
OP_OffsetZero (OP_0);
```

OP_CMPTriggerCmd（SC32f15xx 系列）

Table 223

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 224

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 225获得这个类型的所有成员。

Table 225

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 226

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 227

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 228

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 229

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 230

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
  }
}
```

14 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 231

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_IAPPORA	使用A区设置IAP保护区域
OPTION_IAPPORB	使用B区设置IAP保护区域
OPTION_HIRC_PLL_Switch	切换HIRC/PLL时钟源选择
OPTION_DRV_EH_SetMode	设置COM端口的输出电流IOL驱动能力模式

OPTION 固件库函数详解

OPTION_WDTCmd

Table 232

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 */
RCC_Unlock(0xFF);
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_IFB, ENABLE);
OPTION_WDT_Cmd(ENABLE);
```

OPTION_LVRConfig

Table 233

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 234获得这个参数的所有成员。

Table 234

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 */
RCC_Unlock(0xFF);
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_IFB, ENABLE);
OPTION_LVRConfig (OPTION_LVR_2_9V);
```

OPTION_JTAGCmd

Table 235

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*/
RCC_Unlock(0xFF);
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_IFB, ENABLE);
OPTION_JTAGCmd (DISABLE);
```

OPTION_IAPPORA

Table 192

函数名	OPTION_IAPPORA
函数原型	void OPTION_IAPPORA (uint16_t IAPPROAST, uint16_t IAPPROAED)
功能描述	使用A区设置IAP保护区域
输入参数1	IAPPROAST：IAP 保护区域的起始扇区
输入参数2	IAPPROAED：IAP 保护区域的结束扇区
返回值	无

使用示例:

```
/ * 使用A区设置IAP保护区域*/
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_IFB, ENABLE);
OPTION_IAPPORA (497, 498);
```

OPTION_IAPPORB

Table 193

函数名	OPTION_IAPPORB
函数原型	void OPTION_IAPPORB (uint16_t IAPPROAST, uint16_t IAPPROAED)
功能描述	使用B区设置IAP保护区域
输入参数1	IAPPROAST：IAP 保护区域的起始扇区
输入参数2	IAPPROAED：IAP 保护区域的结束扇区
返回值	无

使用示例:

```
/ * 使用B区设置IAP保护区域*/
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_IFB, ENABLE);
OPTION_IAPPORA (497, 498);
```

OPTION_HIRC_PLL_Switch

Table 193

函数名	OPTION_HIRC_PLL_Switch
函数原型	void OPTION_HIRC_PLL_Switch(OPTION_HIRC_PLL_TypeDef OPTION_HIRC_PLL)
功能描述	切换HIRC/PLL时钟源选择。
输入参数	OPTION_HIRC_PLL：HIRC/PLL时钟源选择。
返回值	无

OPTION_HIRC_PLL

OPTION_HIRC_PLL选择切换HIRC/PLL时钟源选择，参阅Table 234获得这个参数的所有成员。

Table 236

OPTION_HIRC_PLL	描述
OPTION_PLL	使用 PLL 作为时钟源
OPTION_HIRC_DIV1	使用 HIRC/1 代替 PLL（需要 HIRC_NDIV=1）

使用示例:

```
/ * 切换PLL时钟源选择*/
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_IFB, ENABLE);
OPTION_HIRC_PLL_Switch(OPTION_PLL);
```

OPTION_DRV_EH_SetMode

Table 194

函数名	OPTION_DRV_EH_SetMode
函数原型	void OPTION_DRV_EH_SetMode(OPTION_DRV_EH_TypeDef OPTION_DRV_EH)
功能描述	设置COM端口的I/O驱动能力模式
输入参数	OPTION_DRV_EH：COM端口驱动功能选择。
返回值	无

OPTION_DRV_EH

OPTION_DRV_EH设置COM端口的I/O驱动能力模式，参阅Table 234获得这个参数的所有成员。

Table 237

OPTION_DRV_EH	描述
DRV_EH_NORMAL	当COM功能启用时，COM0~COM7具有强大的驱动能力
DRV_EH_STRONG	COM0~COM7端口始终具备强大的IOL驱动电流吸收能力

使用示例:

```
/ * COM0~COM7端口始终具备强大的IOL驱动电流吸收能力*/
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_IFB, ENABLE);
OPTION_DRV_EH_SetMode(DRV_EH_STRONG);
```

15 PWM 固件库函数

赛元SC32f10xx，SC32R803，SC32f12xx、SC32R805、SC32R806、SC32L14xx、SC32R807、SC32R808、SC32f11xx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 238

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 239

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 240

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 241

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 242获得这个参数的所有成员。

Table 242

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 243获得这个参数的所有成员。

Table 243

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 244获得这个参数的所有成员。

Table 244

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 245获得这个参数的所有成员。

Table 245

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 245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 246

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 247所示。

Table 247

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 248

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 249

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 250

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 251

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 242。
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 252

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 242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 253

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 254

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 255

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 256

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 257

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 258获得这个参数的所有成员。

Table 258

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 259获得这个参数的所有成员。

Table 259

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 260获得这个参数的所有成员。

Table 260

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 261

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 262所示。

Table 262

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 263

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 264

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 265。
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 265获得这个参数的所有成员。。

Table 265

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 266

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 263。
返回值	PWMx的新状态（SET 或者 RESET）

PWM_FLAG

Table 267给出了PWM_FLAG的值。

Table 267

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 268

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 263。
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);
  }
}
```

16 LEDPWM 固件库函数

赛元SC32f10xx，SC32R803，SC32f12xx、SC32R805、SC32R806、SC32R808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;
```

在文件“sc32r803.h”中定义如下：

```
typedef struct
{
  __IO uint32_t LEDPWM_CON;
  __IO uint32_t LEDPWM_CHN0;
  __IO uint32_t LEDPWM_STS;
  __IO uint32_t LEDPWM_INV0;
  __IO uint32_t LEDPWM_DFR;
  __IO uint32_t LEDPWM_CYCLE;
  __IO uint32_t LEDPWM_CHN1;
  __IO uint32_t LEDPWM_INV1;
  __IO uint32_t RESERVED1[3];
  __IO uint32_t LEDPWM_DT[39];
} LEDPWM_TypeDef;
```

LEDPWM 寄存器表格

Table 269

寄存器	描述
LEDPWM_CON	LEDPWM控制寄存器
LEDPWM_CHN/ LEDPWM_CHN0/ LEDPWM_CHN1	LEDPWM通道设置寄存器
LEDPWM_STS	LEDPWM状态标志寄存器
LEDPWM_INV/ LEDPWM_INV0/ LEDPWM_INV1	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 270

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(入参选择参考具体型号的规格书)

LEDPWM_DeInit（SC32F10xx、SC32F12xx、SC32R803、SC32R805、SC32R806、SC32R808 系列）

Table 271

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（SC32F10xx、SC32F12xx、SC32R803、SC32R805、SC32R806、SC32R808 系列）

Table 272

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 273获得这个参数的所有成员。

Table 273

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 243获得这个参数的所有成员。

Table 274

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 275获得这个参数的所有成员。

Table 275

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_0_31	选择通道0-31
LEDPWM_Channel_32	选择通道32
LEDPWM_Channel_33	选择通道33
略	略
LEDPWM_Channel_38	选择通道38
LEDPWM_Channel_32_38	选择通道32-38
LEDPWM_OutputChannel	描述

LEDPWM_LowPolarityChannl

LEDPWM_LowPolarityChannl设置LEDPWM反向输出通道，参阅Table 275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（SC32F10xx、SC32F12xx、SC32R803、SC32R805、SC32R806、SC32R808 系列）

Table 276

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 247所示。

Table 277

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（SC32F10xx、SC32F12xx、SC32R803、SC32R805、SC32R806、SC32R808 系列）

Table 278

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（SC32F10xx、SC32F12xx、SC32R803、SC32R805、SC32R806、SC32R808 系列）

Table 279

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 273。
返回值	无

使用示例:

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

LEDPWM_GetPrescaler（SC32F10xx、SC32F12xx、SC32R803、SC32R805、SC32R806、SC32R808 系列）

Table 280

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

使用示例:

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

LEDPWM_SetCycle（SC32F10xx、SC32F12xx、SC32R803、SC32R805、SC32R806、SC32R808 系列）

Table 281

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（SC32F10xx、SC32F12xx、SC32R803、SC32R805、SC32R806、SC32R808 系列）

Table 282

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（SC32F10xx、SC32F12xx、SC32R803、SC32R805、SC32R806、SC32R808 系列）

Table 283

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
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（SC32F10xx、SC32F12xx、SC32R803、SC32R805、SC32R806、SC32R808 系列）

Table 284

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
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（SC32F10xx、SC32F12xx、SC32R803、SC32R805、SC32R806、SC32R808 系列）

Table 285

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 286。
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 286获得这个参数的所有成员。。

Table 286

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（SC32F10xx、SC32F12xx、SC32R803、SC32R805、SC32R806、SC32R808 系列）

Table 287

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 288。
返回值	LEDPWMx的新状态（SET 或者 RESET）

LEDPWM_FLAG

Table 288给出了LEDPWM_FLAG的值。

Table 288

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（SC32F10xx、SC32F12xx、SC32R803、SC32R805、SC32R806、SC32R808 系列）

Table 289

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 288。
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%
```

17 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 290

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 291

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_PLLConfig	配置PLL时钟源
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_LCDLEDCLKConfig	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_PLLQCmd	启用或禁用 PLLQ 时钟

RCC 固件库函数详解（入参选择参考具体型号的规格书）

RCC_DeInit

Table 292

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 293

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、SC32R803、SC32F12xx、SC32R805、SC32R806、SC32L14xx、SC32R807、SC32R808、SC32F11xx 系列）

Table 294

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 295

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 296

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 297

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（SC32F10xx 系列）

Table 298

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（SC32F10xx 系列）

Table 299

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、SC32R803、SC32F12xx、SC32R805、SC32R806、SC32L14xx、SC32R807、SC32R808、SC32F11xx 系列）

Table 300

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 301

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 302

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 303

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 304

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 305获得这个参数的所有成员。

Table 305

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

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 306

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 307

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 308获得这个参数的所有成员。

Table 308

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 309

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 310

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 311

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 312

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 313获得这个参数的所有成员。

Table 313

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 314

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 315获得这个参数的所有成员。

Table 315

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 316

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 317

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 318

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 319

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 320获得这个参数的所有成员。

Table 320

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 321

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 322

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 323获得这个参数的所有成员。

Table 323

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 324

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 325获得这个参数的所有成员。

Table 325

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 326

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 327获得这个参数的所有成员。

Table 327

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

使用示例:

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

RCC_APB0PeriphClockCmd

Table 328

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 329获得这个参数的所有成员。

Table 329

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_QTWI0	QTWI0外设时钟
RCC_APB0Periph_QTWI2	QTWI2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

使用示例:

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

RCC_APB1PeriphClockCmd

Table 330

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 331获得这个参数的所有成员。

Table 331

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_QTWI1	QTWI1外设时钟
RCC_APB1Periph_QTWI3	QTWI3外设时钟
RCC_APB1Periph_ALL	挂载在APB1上所有外设时钟

使用示例:

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

RCC_APB2PeriphClockCmd

Table 332

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 333获得这个参数的所有成员。

Table 333

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

使用示例:

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

RCC_AHBPeriphResetCmd

Table 334

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 335

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 336

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 337

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 338

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 339获得这个参数的所有成员。

Table 339

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_CMP3	CMP3非屏蔽中断
RCC_NMIPeriph_CMP0	CMP0非屏蔽中断
RCC_NMIPeriph_ALL	开启所有外设NMI中断

使用示例:

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

RCC_ITConfig

Table 340

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 341

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 342获得这个参数的所有成员。

Table 342

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);
```

RCC_PLLQCmd

Table 341

函数名	RCC_PLLQCmd
函数原形	void RCC_PLLQCmd (FunctionalState NewState)
功能描述	启用或禁用 PLLQ 时钟。
输入参数	NewState：PLLQ时钟的新状态。
返回值	无

使用示例:

```
/* 启用 PLLQ 时钟 */
RCC_PLLQCmd (ENABLE);
```

18 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 343

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 344

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 345

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 346

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 347获得这个参数的所有成员。

Table 347

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 348获得这个参数的所有成员。

Table 348

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 349获得这个参数的所有成员。

Table 349

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 350获得这个参数的所有成员。

Table 350

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 351获得这个参数的所有成员。

Table 351

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 352获得这个参数的所有成员。

Table 352

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

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 353

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 354

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 355

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 356获得这个参数的所有成员。

Table 356

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 357

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 358获得这个参数的所有成员。

Table 358

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_TBIE	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
SPI_IT_QTWIE	SPI一帧数据传输完成中断使能位

使用示例:

```
/* 使能SPI0 外设总中断 */
```

SPI_ITConfig (SPI1, SPI_IT_INTEN，ENABLE);

SPI_SetMode

Table 359

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 360

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 361

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 362

函数名	SPI_ReceiveData
函数原形	Uint32_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 */
Uint32_t Data1
Data1 = SPI_ReceiveData(SPI1);
```

SPI_ReceiveDataFIFO

Table 363

函数名	SPI_ReceiveDataFIFO
函数原形	void SPI_ReceiveDataFIFO(SPI_TypeDef* SPIx, uint32_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 */
Uint32_t DataArray1[8];
Data1 = SPI_ReceiveDataFIFO(SPI0, DataArray1,8);
```

SPI_SendDataFIFO

Table 364

函数名	SPI_SendDataFIFO
函数原形	void SPI_SendDataFIFO(SPI_TypeDef* SPIx, uint32_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 365

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 366获得这个参数的所有成员。

Table 366

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 367

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 368

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 369获得这个参数的所有成员。

Table 369

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
SPI_PinRemap_C	SPI映射管脚组C

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);
}
```

19 QEP 固件库函数

QEP 寄存器结构

赛元SC32F15Gx、SC32R601系列QEP寄存器结构，QEP_TypeDef，在文件“SC32F15Gx.h、sc32r601.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;
} QEP_TypeDef;
```

QEP 寄存器表格

Table 370

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 固件库函数列表

Table 371

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 372

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 373

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;
} QEP_InitTypeDef ;
```

QEP_Mode

QEP_Mode 设置QEP的工作模式，参阅Table 374获得这个参数的所有成员。

Table 374

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 375获得这个参数的所有成员。

Table 375

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_QI	QEP引脚选通QI

QEP_CEN

QEP_CEN配置QEP计数方式，参阅Table 376获得这个参数的所有成员。

Table 376

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 377获得这个参数的所有成员。

Table 377

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 378获得这个参数的所有成员。

Table 378

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 379获得这个参数的所有成员。

Table 379

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 380获得这个参数的所有成员。

Table 380

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 381获得这个参数的所有成员。

Table 381

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 382

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 383

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 384

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 385

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 386

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 387

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 388

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 389

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 390

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 391

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 392

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 393获得这个参数的所有成员。

Table 393

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 394

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 395获得这个参数的所有成员。

Table 395

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 396

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 397获得这个参数的所有成员。

Table 397

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 398

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);
    }
}
```

20 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 399

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 400

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（入参选择参考具体型号的规格书）

TIM_DeInit

Table 401

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 402

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 403获得这个参数的所有成员。

Table 403

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 404获得这个参数的所有成员。

Table 404

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 405获得这个参数的所有成员。

Table 405

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 406获得这个参数的所有成员。

Table 406

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 407

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 408

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 409

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 410

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 411

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 412

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 413

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 403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 414

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 415

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 416获得这个参数的所有成员。

Table 416

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 417获得这个参数的所有成员。

Table 417

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 418

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	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 419

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 420

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 421

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 422

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 423获得这个参数的所有成员。

Table 423

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 424

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 425

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 426

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 427

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 428

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 429获得这个参数的所有成员。

Table 429

TIM_PinRemap	描述
TIM_PinRemap_Default	默认TIM管脚映射
TIM_PinRemap_A	TIM映射管脚组A
TIM_PinRemap_B	TIM映射管脚组B

使用示例:

```
/* 配置T2EX、T2CAP/T2管脚为PB7、PA12 */
TIM_PinRemapConfig(TIM2, TIM_PinRemap_A)
```

TIM_ITConfig

Table 430

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 431获得这个参数的所有成员。

Table 431

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 432

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 433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 433获得这个参数的所有成员。

Table 433

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 434

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 435

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 436获得这个参数的所有成员。

Table 436

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
    }
  }
```

21 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 437

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 固件库函数列表

Table 438

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_DeInit

Table 439

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 440

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 441获得这个参数的所有成员。

Table 441

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 442获得这个参数的所有成员。

Table 442

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

TWI_Stretch

TWI时钟延长使能设定，参阅Table 443获得这个参数的所有成员。

Table 443

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 444获得这个参数的所有成员。

Table 444

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 445

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 446

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 447

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 448获得这个参数的所有成员。

Table 448

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 449

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 450

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 451

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 452

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 453

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 454

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 455获得这个参数的所有成员。

Table 455

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 456

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 457

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 458

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 459获得这个参数的所有成员。

Table 459

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 460

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 461获得这个参数的所有成员。

Table 461

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
TWI_FLAG_QTWIF	TWI中断标志位

使用示例:

```
/* 获得TWI1标志位状态 */
TWI_GetFlagStatus(TWI1, TWI_FLAG_TWIF);
```

TWI_ClearFlag

Table 462

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 463

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 464

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 465

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 466获得这个参数的所有成员。

Table 466

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

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 467

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	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 468获得这个参数的所有成员。

Table 468

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;
  }
}
```

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 469

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 470

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 471

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 472

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 473获得这个参数的所有成员。

Table 473

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 474中的成员。

Table 474

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 = 32000000; //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 475

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 476

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 477

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 478

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 479

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 480给出了UART_PinRemap的值。

Table 480

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 481

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 482给出了所有UART_IT的取值列表。

Table 482

UART_IT	描述
UART_IT_WK	唤醒中断
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 483

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 484获得这个参数的所有成员。

Table 484

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_WK	唤醒中断标志位
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 485

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 486

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 487给出了所有UART_ DMAReq的取值列表。

Table 487

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 488

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 489获得这个参数的所有成员。

Table 489

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 490

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 491获得这个参数的所有成员。

Table 491

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 492

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 493

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 494

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 495获得这个参数的所有成员。

Table 495

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_CallID(uint8_t id)

Table 496

函数名	LIN_CallID(uint8_t id)
函数原型	uint8_t LIN_CallID(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_CallID(0x32);//获取id为0x32的PID

LINCalChecksum

Table 497

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 QSPI 固件库函数

SC32R803、SC32F11xx有QSPI0/1，QSPI仅支持主机模式，每个QSPI均可适配4种不同传输位宽：8/16/24/32 bits，且在不同位宽下提供固定8级FIFO，收发独立，支持单线、双线、四线通信，支持两种功能模式：QSPI半双工通信模式和直通模式，通信速率高达32MHz，两个QSPI可以不经过FIFO透传，支持DMA。

QSPI 寄存器结构

SPI 寄存器结构，SPI_TypeDef，在文件“sc32r803.h”中定义如下：

```
typedef struct
{
__IO uint32_t QSPIx_CON;
__IO uint32_t QSPIx_STS;
__IO uint32_t QSPIx_ADD;
__IO uint32_t QSPIx_DATA;
__IO uint32_t QSPIx_IDE;
__IO uint32_t QSPIx_DL;
__IO uint32_t RESERVED0;
__IO uint32_t QSPIx_REV;
} QSPI_TypeDeff;
```

QSPI 寄存器表格

Table 498

寄存器	描述
QSPIx_CON	QSPIx控制寄存器
QSPIx_STS	QSPIx标志状态位寄存器
QSPIx_DATA	QSPIx数据寄存器
QSPIx_IDE	QSPIx的中断使能及DMA控制寄存器
QSPIx_DL	QSPIx数据长度寄存器

QSPI 固件库函数列表

Table 499

函数名	描述
QSPI_DeInit	将外设 QSPIx寄存器重设为缺省值
QSPI_StructInit	根据 QSPI _InitStruct 中指定的参数初始化外设QSPIx 寄存器
QSPI_Init	把 QSPI_InitStruct 中的每一个参数按缺省值填入
QSPI_Cmd	使能或者失能 QSPI外设
QSPI_SetDataLength	设置传输数据宽度
QSPI_LModeSelect	设置QSPI的传输线路模式
QSPI_ModeSelect	设置QSPI 的工作模式
QSPI_Write_ComSet	设置QSPI的写入条件
QSPI_SendData8	通过外设 QSPI发送一个8位数据
QSPI_SendData16	通过外设 QSPI发送一个16位数据
QSPI_SendData32	通过外设 QSPI发送一个32位数据
QSPI_SendDataFIFO	QSPI通过FIFO发送数据
QSPI_SendMultipleData	通过外设 QSPI发送多个数据
QSPI_Read_ComSet	设置QSPI的读取条件
QSPI_CLKONLYSet	QSPI时钟开启
QSPI_ReceiveMultipleData	通过外设 QSPI接收多个数据
QSPI_ReceiveData8	返回通过 QSPI最近接收的8位数据
QSPI_ReceiveData16	返回通过 QSPI最近接收的16位数据
QSPI_ReceiveData32	返回通过 QSPI最近接收的32位数据

QSPI_Receivelen	QSPI接收数据长度
QSPI_ReceiveDataFIFO	QSPI通过FIFO接收数据
QSPI_ReceiveMultipleData	QSPI连续接收数据
QSPI_ITConfig	使能或者失能指定的 QSPI中断
QSPI_GetFlagStatus	获取QSPI中断标志位
QSPI_ClearFlag	清除 QSPI的待处理标志位
QSPI_DMACmd	使能或者失能指定QSPI的 DMA 请求

QSPI 固件库函数详解

QSPI _DeInit

Table 500

函数名	QSPI _DeInit
函数原型	void QSPI_DeInit(TWI_QSPIx_TypeDef* QSPIx)
功能描述	将外设 QSPIx 寄存器重设为缺省值
输入参数	QSPIx：选择 QSPI外设（x 可以是 0,1）
返回值	无

使用示例:

```
/* 将外设QSPI0 寄存器重设为缺省值*/
QSPI_DeInit(QSPI0);
```

QSPI_Init

Table 501

函数名	QSPI_Init
函数原形	void QSPI_Init(TWI_QSPIx_TypeDef* QSPIx,QSPI_InitTypeDef* QSPI_InitStruct)
功能描述	根据 QSPI_InitStruct 中指定的参数初始化外设 QSPIx 寄存器
输入参数 1	QSPIx：选择 QSPI外设（x 可以是 0,1）
输入参数 2	QSPI_InitStruct：指向结构 QSPI_InitTypeDef 的指针，包含了外设 QSPI的配置信息 参阅 Section：QSPI_InitTypeDef 查阅更多该参数允许取值范围
返回值	无

QSPI_InitStruct

QSPI_InitTypeDef 定义于文件“sc32f1xxx_qspi.h”：

```
typedef struct
{
uint32_t QSPI_SShift;
uint32_t QSPI_Prescaler;
uint32_t QSPI_DWidth;
int32_t QSP_LMode;
uint32_t QSPI_Mode;
uint32_t QSPI_CPMode;
uint32_t QSPI_RW;
uint32_t QSPI_CLKONLY;
}   QSPI_InitTypeDef ;
```

QSPI_SShift

QSPI的采样移位设定，参阅 Table 502获得这个参数的所有成员。

Table 502

QSPI_SShift	描述
QSPI_SShift_OFF	采样移位关闭
QSPI_SShift_HalfClock	半时钟采样移位打开

QSPI_Prescaler

QSPI的时钟频率档位设定，参阅 Table 503获得这个参数的所有成员。

Table 503

QSPI_Prescaler	描述
QSPI_Prescaler_1	时钟预分频值为 1
QSPI_Prescaler_2	时钟预分频值为 2
QSPI_Prescaler_4	时钟预分频值为 4
QSPI_Prescaler_8	时钟预分频值为 8
QSPI_Prescaler_16	时钟预分频值为 16
QSPI_Prescaler_32	时钟预分频值为 32
QSPI_Prescaler_64	时钟预分频值为 64
QSPI_Prescaler_128	时钟预分频值为 128
QSPI_Prescaler_256	时钟预分频值为 256
QSPI_Prescaler_512	时钟预分频值为 512
QSPI_Prescaler_1024	时钟预分频值为 1024
QSPI_Prescaler_2048	时钟预分频值为 2048
QSPI_Prescaler_4096	时钟预分频值为 4096

QSPI_DWidth

QSPI的传输数据的宽度设定，参阅 Table 504获得这个参数的所有成员。

Table 504

QSPI_DWidth	描述
QSPI_DWidth_8bit	传输数据宽度为 8 位
QSPI_DWidth_16bit	传输数据宽度为 16 位
QSPI_DWidth_24bit	传输数据宽度为 24 位
QSPI_DWidth_32bit	传输数据宽度为 32 位

QSP_LMode

QSPI的传输数据时使用的线数设定，参阅 Table 505获得这个参数的所有成员。

Table 505

QSP_LMode	描述
QSP_LMode_0Line	不使用任何线路进行传输
QSP_LMode_1Line	使用单线进行传输
QSP_LMode_2Line	使用双线进行传输
QSP_LMode_4Line	使用四线进行传输

QSPI_Mode

QSPI的工作模式设定，参阅 Table 506获得这个参数的所有成员。

Table 506

QSPI_Mode	描述
QSPI_Mode_SPI	SPI 模式
QSPI_Mode_QSPI	QSPI 模式
QSPI_Mode_QSPISTRT	QSPI STRT 模式，决定当前QSPI是否允许触发另一个QSPI
QSPI_Mode_QSPISTRR	QSPI STRR 模式，决定当前QSPI是否允许被另一个QSPI触发

QSPI_CPMode

QSPI的时钟相位设定，参阅 Table 507获得这个参数的所有成员。

Table 507

QSPI_CPMode	描述
QSPI_CPMode_Low	SCK周期的第一沿采集数据
QSPI_CPMode_High	SCK周期的第二沿采集数据

QSPI_RW

QSPI的读写控制设定，参阅Table 508获得这个参数的所有成员。

Table 508

QSPI_RW	描述
QSPI_RW_Write	主机发送数据，从机接收数据
QSPI_RW_Read	主机读取数据，从机发送数据

QSPI_CLKONLY

QSPI的只发时钟模式设定，参阅 Table 509获得这个参数的所有成员。

Table 509

QSPI_CLKONLY	描述
QSPI_CLKONLY_OFF	只发时钟模式禁止
QSPI_CLKONLY_ON	只发时钟模式使能

使用示例:

```
/* 根据 QSPI_InitStruct 中指定的参数初始化 QSPI0 */
QSPI_InitTypeDef QSPI_InitStruct;
QSPI_StructInit(&QSPI_InitStruct);
QSPI_InitStruct.QSPI_CLKONLY = QSPI_CLKONLY_ON;
QSPI_InitStruct.QSPI_CPMode = QSPI_CPMode_Low;
QSPI_InitStruct.QSPI_DWidth = QSPI_DWidth_32bit;
QSPI_InitStruct.QSPI_Prescaler = QSPI_Prescaler_16;
QSPI_InitStruct.QSPI_Mode = QSPI_Mode_QSPISTRT;
QSPI_InitStruct.QSPI_RW = QSPI_RW_Write;
QSPI_InitStruct.QSPI_SShift = QSPI_SShift_OFF;
QSPI_InitStruct.QSP_LMode = QSP_LMode_4Line;
QSPI_Init(QSPI0,&QSPI_InitStruct);
```

QSPI_StructInit

Table 510

函数名	QSPI_StructInit
函数原形	void QSPI_StructInit(QSPI_InitTypeDef* QSPI_InitStruct)
功能描述	根据 QSPI _InitStruct 中指定的参数初始化外设QSPIx 寄存器
输入参数	QSPI_InitStruct：指向结构 QSPI_InitTypeDef 的指针，待初始化
返回值	无

使用示例:

```
/* 把 QSPI_InitStruct 中的每一个参数按缺省值填入 */
QSPI_InitTypeDef  *QSPI_InitStruct;
QSPI_StructInit(&QSPI_InitStruct);
```

QSPI_Cmd

Table 511

函数名	QSPI_Cmd
函数原形	void QSPI_Cmd(TWI_QSPIx_TypeDef* QSPIx, FunctionalState NewState)
功能描述	使能或者失能 QSPI外设
输入参数 1	QSPIx：选择QSPI外设
输入参数 2	NewState: 外设 QSPI1 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QSPI1 外设 */
QSPI_Cmd(QSPI1, ENABLE);
```

QSPI_SetDataLength

Table 512

函数名	QSPI_Cmd
函数原形	void QSPI_SetDataLength(TWI_QSPIx_TypeDef *QSPIx,QSPI_DWidth_TypeDef QSPI_DWidth)
功能描述	设置传输数据宽度
输入参数 1	QSPIx：选择QSPI外设
输入参数 2	QSPI_DWidth：传输数据宽度
返回值	无

QSPI_DWidth

QSPI_DWidth传输数据宽度选择，参阅 Table 504获得这个参数的所有成员。
使用示例:

```
/* 设置QSPI0的传输数据宽度 */
QSPI_SetDataLength(QSPI0, QSPI_DWidth_8bit);
```

QSPI_LModeSelect

Table 513

函数名	QSPI_LModeSelect
函数原形	void QSPI_LModeSelect(TWI_QSPIx_TypeDef *QSPIx,QSPI_LMode_TypeDef QSPI_LMode)
功能描述	设置QSPI的传输线路模式
输入参数 1	QSPIx：选择QSPI外设
输入参数 2	QSPI_LMode：传输线路模式
返回值	无

QSPI_LMode

QSPI_LMode传输数据宽度选择，参阅 Table 505 获得这个参数的所有成员。
使用示例:

```
/* 设置QSPI0使用双线进行传输 */
QSPI_LModeSelect (QSPI0, QSPI_LMode_2Line);
```

QSPI_ModeSelect

Table 514

函数名	QSPI_ModeSelect
函数原形	void QSPI_ModeSelect(TWI_QSPIx_TypeDef *QSPIx,QSPI_Mode_TypeDef QSPI_Mode)
功能描述	设置 QSPI的工作模式
输入参数 1	QSPIx：选择QSPI外设
输入参数 2	QSPI_Mode：传输线路模式
返回值	无

QSPI_Mode

QSPI_Mode传输数据宽度选择，参阅 Table 506 获得这个参数的所有成员。
使用示例:

```
/* 设置使用QSPI模式传输*/
QSPI_LModeSelect (QSPI0, QSPI_Mode_QSPI);
```

QSPI_Write_ComSet

Table 515

函数名	QSPI_Write_ComSet
函数原形	void QSPI_Write_ComSet(TWI_QSPIx_TypeDef *QSPIx, int32_t LMODE, uint32_t DWIDTH,uint32_t CLKONLY)
功能描述	设置QSPI的写入条件
输入参数 1	QSPIx：选择QSPI外设
输入参数 2	LMODE：传输线路模式

输入参数 3	DWIDTH：传输数据宽度
输入参数 4	CLKONLY：只发时钟模式设定
返回值	无

使用示例:

```
/* 设置QSPI的写入条件 */
QSPIx_Write_ComSet (QSPI0, QSP_LMode_2Line, QSPI_DWidth_32bit ,QSPI_CLKONLY_ON);
```

QSPI_SendData8

Table 516

函数名	QSPI_SendData8
函数原形	void QSPI_SendData8(TWI_QSPIx_TypeDef *QSPIx,uint8_t Data)
功能描述	通过外设QSPI发送一个8位数据
输入参数 1	QSPIx：选择QSPI外设
输入参数 2	Data：待传输数据
返回值	无

使用示例:

```
/* 通过外设QSPI发送一个8位数据 */
uint8_t Data1 = 0x5A;
QSPI_SendData8(QSPI0, Data1);
```

QSPI_SendData16

Table 517

函数名	QSPI_SendData16
函数原形	void QSPI_SendData16(TWI_QSPIx_TypeDef *QSPIx,uint16_t Data)
功能描述	通过外设QSPI发送一个16位数据
输入参数 1	QSPIx：选择QSPI外设
输入参数 2	Data：待传输数据
返回值	无

使用示例:

```
/* 通过外设QSPI发送一个16位数据 */
Uint16_t Data1 = 0x5A5A;
QSPI_SendData8(QSPI0, Data1);
```

QSPI_SendData32

Table 518

函数名	QSPI_SendData32
函数原形	void QSPI_SendData32(TWI_QSPIx_TypeDef *QSPIx,uint32_t Data)
功能描述	通过外设QSPI发送一个32位数据
输入参数 1	QSPIx：选择QSPI外设
输入参数 2	Data：待传输数据
返回值	无

使用示例:

```
/* 通过外设QSPI发送一个32位数据 */
Uint32_t Data1 = 0x5A5A 4A4A;
QSPI_SendData8(QSPI0, Data1);
```

QSPI_SendDataFIFO

Table 519

函数名	QSPI_SendDataFIFO
函数原形	void QSPI_SendDataFIFO (TWI_QSPIx_TypeDef *QSPIx, uint32_t* Data, uint16_t length)

功能描述	QSPI通过 FIFO 发送数据
输入参数 1	QSPIx：选择QSPI外设
输入参数 2	Data：待传输数据
输入参数 3	Length：数据的长度
返回值	无

使用示例:

```
/* QSPI通过 FIFO 发送数据 */
uint8_t DataArray1[8]= {1,2,3,4,5,6,7,8};
QSPI_SendDataFIFO (QSPI0, DataArray1, 8);（这个8正确吗，不确定写的对不对）
```

QSPI_SendMultipleData

Table 520

函数名	QSPI_SendMultipleData
函数原形	void QSPI_SendMultipleData(TWI_QSPIx_TypeDef *QSPIx, uint32_t *buf, uint32_t len)
功能描述	通过外设QSPI发送多个数据
输入参数 1	QSPIx：选择QSPI外设
输入参数 2	buf：待传输数据
输入参数 3	Len：数据的长度
返回值	无

使用示例:

```
/* 通过外设QSPI发送多个数据 */
uint8_t DataArray1[8]= {1,2,3,4,5,6,7,8};
QSPI_SendMultipleData (QSPI0, DataArray1, 8);
```

QSPI_Read_ComSet

Table 521

函数名	QSPI_Read_ComSet
函数原形	void QSPI_Read_ComSet(TWI_QSPIx_TypeDef *QSPIx, int32_t LMODE, uint32_t DWIDTH,uint32_t CLKONLY, uint32_t length)
功能描述	设置QSPI的读取条件
输入参数 1	QSPIx：选择QSPI外设
输入参数 2	LMODE：传输线路模式
输入参数 3	DWIDTH：传输数据宽度
输入参数 4	CLKONLY：只发时钟模式设定
输入参数 5	length：数据的长度
返回值	无

使用示例:

```
/* 通过外设QSPI发送一个32位数据 */
QSPIx_Read_ComSet(QSPI0, QSP_LMode_2Line, QSPI_DWidth_32bit ,QSPI_CLKONLY_ON，8);
```

QSPI_CLKONLYSet

Table 522

函数名	QSPI_CLKONLYSet
函数原形	void QSPI_CLKONLYSet(QSPI_TypeDef *QSPIx, uint32_t CLKONLY)
功能描述	QSPI时钟开启使能
输入参数 1	QSPIx：选择QSPI外设
输入参数 2	CLKONLY：选择QSPI时钟开启使能
返回值	无

使用示例:

```
/*QSPI0时钟开启*/
QSPI_CLKONLYSet(QSPI0, QSPI_CLKONLY_ON);
```

QSPI_ReceiveData8

Table 523

函数名	QSPI_ReceiveData8
函数原形	uint8_t QSPI_ReceiveData8(TWI_QSPIx_TypeDef *QSPIx)
功能描述	返回通过 QSPI最近接收的8位数据
输入参数 1	QSPIx：选择QSPI外设
返回值	QSPIx_DATA

使用示例:

```
/* 把QSPI0接收的数据赋值给Data1*/
Uint8_t Data0
QSPI_ReceiveData8 (QSPI0);
```

QSPI_ReceiveData16

Table 524

函数名	QSPI_ReceiveData16
函数原形	uint8_t QSPI_ReceiveData16(TWI_QSPIx_TypeDef *QSPIx)
功能描述	返回通过 QSPI最近接收的16位数据
输入参数	QSPIx：选择QSPI外设
返回值	QSPIx_DATA

使用示例:

```
/* 把QSPI0接收的数据赋值给Data1*/
Uint16_t Data0
QSPI_ReceiveData16 (QSPI0,);
```

QSPI_ReceiveData32

Table 525

函数名	QSPI_ReceiveData32
函数原形	uint8_t QSPI_ReceiveData32(TWI_QSPIx_TypeDef *QSPIx)
功能描述	返回通过 QSPI最近接收的32位数据
输入参数	QSPIx：选择QSPI外设
返回值	TWI_QSPIx_DATA

使用示例:

```
/* 把QSPI0接收的数据赋值给Data1*/
Uint32_t Data0
Data1 = QSPI_ReceiveData32 (QSPI0,);
```

QSPI_Receivelen

Table 526

函数名	QSPI_Receivelen
函数原形	void QSPI_Receivelen(QSPI_TypeDef *QSPIx,uint32_t datalen)
功能描述	设置QSPI接收数据长度
输入参数 1	QSPIx：选择QSPI外设
输入参数 2	Datalen：设置QSPI接收数据长度
返回值	TWI_QSPIx_DATA

使用示例:

```
/* 把QSPI0接收的数据8个*/
QSPI_Receivelen (QSPI0,8);
```

QSPI_ReceiveDataFIFO

Table 527

函数名	QSPI_ReceiveDataFIFO
函数原形	void QSPI_ReceiveDataFIFO (TWI_QSPIx_TypeDef *QSPIx, uint32_t* Data, uint16_t length)
功能描述	QSPI通过 FIFO 发送数据
输入参数 1	QSPIx：选择QSPI外设
输入参数 2	Data：待传输数据
输入参数 3	Length：数据的长度
返回值	无

使用示例:

```
/* 把QSPI0接收的数据赋值给Data1 */
Uint32_t DataArray1[8];
Data1 =  QSPI_ReceiveDataFIFO (QSPI0, DataArray1, 8);
```

QSPIx_ReceiveMultipleData

Table 528

函数名	QSPIx_ReceiveMultipleData
函数原形	void QSPIx_ReceiveMultipleData(TWI_QSPIx_TypeDef *QSPIx,uint8_t *buf, uint32_t length)
功能描述	通过外设QSPI接收多个数据
输入参数 1	QSPIx：选择QSPI外设
输入参数 2	buf：待传输数据
输入参数 3	length：数据的长度
返回值	无

使用示例:

```
/* 通过外设QSPI发送多个数据 */
uint8_t DataArray1[8]= {1,2,3,4,5,6,7,8};
QSPI_SendMultipleData (QSPI0, DataArray1, 8);
```

QSPI_ITConfig

Table 529

函数名	QSPI_ITConfig
函数原形	void QSPI_ITConfig(TWI_QSPIx_TypeDef* QSPIx,uint32_t QSPI_IT, FunctionalState NewState)
功能描述	使能或者失能指定的 QSPI中断
输入参数 1	QSPIx：选择QSPI外设
输入参数 2	QSPI_IT：指定要启用或禁用的 QSPI 中断源
输入参数 3	NewState: 外设 QSPIx的新状态 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

QSPI_IT

QSPI中断选择，参阅 Table 530获得这个参数的所有成员。

Table 530

QSPI_IT	描述
QSPI_IT_INTEN	中断请求 CPU的使能控制位
QSPI_IT_RXNE	接收FIFO非空中断使能位
QSPI_IT_TB	发送FIFO为空时的中断使能位
QSPI_IT_RX	接收FIFO已满中断使能位
QSPI_IT_RXH	接收FIFO内有效数据超过一半中断使能位
QSPI_IT_TXH	发送FIFO内有效数据不满一半中断使能位
QSPI_IT_QTWIE	QSPI一帧数据传输完成中断使能位

使用示例:

```
/*打开中断请求 CPU的使能和QSPI一帧数据传输完成中断使能*/
QSPI_ITConfig(QSPI0, QSPI_IT_INTEN|QSPI_IT_QTWIE, ENABLE)
```

QSPI_GetFlagStatus

Table 531

函数名	QSPI_GetFlagStatus
函数原形	FlagStatus QSPI_GetFlagStatus (TWI_QSPIx_TypeDef* QSPIx, QSPI_FLAG_TypeDef QSPI_FLAG)
功能描述	获取QSPI中断标志位
输入参数 1	QSPIx：选择QSPI外设
输入参数 2	QSPI_FLAG：指定要获取的标志位
返回值	QSPI_FLAG的新状态

QSPI_FLAG

QSPI标志位选择，参阅 Table 532获得这个参数的所有成员。

Table 532

QSPI_FLAG	描述
QSPI_Flag_RINEIF	接收FIFO非空状态位
QSPI_Flag_TXEIF	发送FIFO为空标志位
QSPI_Flag_RXFIF	接收FIFO已满状态位
QSPI_Flag_RXHIF	接收FIFO内有效数据超过一半状态位/中断状态
QSPI_Flag_TXHIF	发送FIFO内有效数据不满一半状态位/中断状态位
QSPI_Flag_DLUFIF	DL下溢标志位
QSPI_Flag_WCOL	写入冲突标志位
QSPI_Flag_QTWIF	QSPI数据传送标志位
QSPI_Flag_BUSY	忙碌状态位

使用示例:

```
/*获取QSPI0的忙碌状态位标志*/
QSPI_GetFlagStatus(QSPI0,QSPI_Flag_BUSY)
```

QSPI_ClearFlag

Table 533

函数名	QSPI_ClearFlag
函数原形	void QSPI_ClearFlag (TWI_QSPIx_TypeDef* QSPIx, uint32_t QSPI_FLAG)
功能描述	清除QSPI的待处理标志位
输入参数 1	QSPIx：选择QSPI外设
输入参数 2	QSPI_FLAG：指定要清除的标志位
返回值	无

使用示例:

```
/*清除QSPI0的接收FIFO已满状态位标志*/
QSPI_GetFlagStatus(QSPI0, QSPI_Flag_RXFIF)
```

QSPI_DMACmd

Table 534

函数名	QSPI_DMACmd
函数原形	void QSPI_DMACmd (TWI_SPIx_TypeDef* QSPIx, uint16_t QSPI_DMAReq, FunctionalState NewState);
功能描述	使能或者失能指定QSPIx的DMA请求
输入参数 1	QSPIx：选择QSPI外设
输入参数 2	QSPI_DMAReq：指定要启用或禁用的 DMA 请求类型
输入参数 3	NewState: 外设 QSPIx的新状态 这个参数可以取：ENABLE 或者 DISABLE

返回值	无

QSPI_DMAReq

QSPI_DMAReq指定 DMA 请求，参阅 Table 535获得这个参数的所有成员。

Table 535

QSPI_DMAReq	描述
QSPI_DMAReq_RX	DMA中断请求QSPI接受使能控制
QSPI_DMAReq_TX	DMA中断请求QSPI发送使能控制

使用示例:

```
/*使能QSPI0的DMA接收请求*/
QSPI_DMACmd (QSPI0, QSPI_DMAReq_RX, ENABLE)
```

综合使用示例（QSPI 读取 FALSH ID）

```
volatile uint8_t  QSPI_Flag=0;
void QSPI_ReadID_Example()
{
   uint16_t READID = 0;
   GPIO_InitTypeDef GPIO_InitStruct; //定义GPIO初始化结构体变量
   GPIO_InitStruct.GPIO_Pin = GPIO_Pin_13;
   GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT_PP;
   GPIO_Init(GPIOB, &GPIO_InitStruct);
   PB_BIT(13)=1;
   /*开启QSPI0时钟*/
   RCC_APB0Cmd( ENABLE );
   RCC_APB0PeriphClockCmd(RCC_APB0Periph_QTWI0,ENABLE);
  QSPI_InitTypeDef QSPI_InitStruct;
   QSPI_StructInit(&QSPI_InitStruct);
   QSPI_InitStruct.QSPI_CLKONLY = QSPI_CLKONLY_OFF;//只发时钟关闭
   QSPI_InitStruct.QSPI_CPMode = QSPI_CPMode_Low;//空闲时低电平和第一沿采样
   QSPI_InitStruct.QSPI_Prescaler = QSPI_Prescaler_1;//时钟分频16
   QSPI_InitStruct.QSPI_Mode = QSPI_Mode_QSPI;//QSPI模式
   QSPI_InitStruct.QSPI_SShift = QSPI_SShift_HalfClock;//时钟偏移
   QSPI_Init(QSPI0,&QSPI_InitStruct);
   QSPI_Cmd(QSPI0, ENABLE);
   while(1)
   {
   QSPI_Write_ComSet(QSPI0,QSPI_LMode_1Line,QSPI_DWidth_8bit,QSPI_CLKONLY_OFF);
   //CS 拉低
   PB_BIT(13) = 0;
   // 设置发送命令  0X94 Read Manufacturer/Device ID
   QSPI_SendData16(QSPI0,0x94);
   while(READ_BIT(QSPI0->QSPI_STS, TWI_QSPIx_STS_BUSY)); //等待BUSY清零
  // 设置 24bit 地址大小发送
   QSPI_Write_ComSet(QSPI0,QSPI_LMode_4Line,QSPI_DWidth_24bit,QSPI_CLKONLY_OFF);
   QSPI_SendData32(QSPI0,0x000000);
   while(READ_BIT(QSPI0->QSPI_STS, TWI_QSPIx_STS_BUSY)); //等待BUSY清零
   // 设置 3个Dummy
   QSPI_Write_ComSet(QSPI0,QSPI_LMode_0Line,QSPI_DWidth_24bit,QSPI_CLKONLY_OFF);
   QSPI_SendData32(QSPI0,0x000000);
   QSPI_Read_ComSet(QSPI0,QSPI_LMode_4Line,QSPI_DWidth_16bit,1);//,QSPI_CLKONLY_ON
    QSPI_CLKONLYSet(QSPI0, QSPI_CLKONLY_ON);
   QSPI_Receivelen(QSPI0,1);
   while(READ_BIT(QSPI0->QSPI_STS, TWI_QSPIx_STS_BUSY)); //等待BUSY清零
   READID = QSPI0->QSPI_DATA;
   //CS 拉高
   PB_BIT(13) = 1;
```

```
    if(READID == 0xEF17)
    {
    printf("读取ID成功");
    }
        }
}
```

24 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_DMAENCmd	开启DMA协助连续编程
IAP_CONTBurnlength	IAP写数据加速

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、SC32R601 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);
```

IAP_DMAENCmd（SC32F15G、SC32R601 系列）

Table 557

函数名	IAP_DMAENCmd
函数原型	void IAP_DMAENCmd(FunctionalState NewState)
功能描述	开启DMA协助连续编程
输入参数	NewState：IAP DMA 功能开启/关闭
返回值	无

使用示例:

```
/* IAP复位后从APP区域启动 */
IAP_DMAENCmd (ENABLE);
```

IAP_CONTBurnlength（SC32F15G、SC32R601 系列）

Table 558

函数名	IAP_CONTBurnlength
函数原型	void IAP_CONTBurnlength(IAP_BTLD_TypeDef Burnlength)
功能描述	IAP写数据加速
输入参数	Burnlength：选择加速写入数据长度
返回值	无

使用示例:

```
/* 选择加速写入数据长度8byte */
IAP_CONTBurnlength (0x08);
```

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
```

25 PGA 固件库函数

赛元SC32R803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 559

寄存器	描述
PGA_CON	PGA控制寄存器

PGA 固件库函列表

Table 560

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 561

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 562

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 563获得这个参数的所有成员。

Table 563

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 564获得这个参数的所有成员。

Table 564

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 565获得这个参数的所有成员。

Table 565

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 566

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 567

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 568

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
  }
}
```

26 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 569

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 570

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 571

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 572

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 573

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 574获得这个参数的所有成员。

Table 574

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 575

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 576获得这个参数的所有成员。

Table 576

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 577

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 578

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 579获得这个参数的所有成员。

Table 579

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++);
       }
    }
}
```

27 CAN 固件库函数

赛元 SC32R803、SC32F11xx系列与SC32F15xx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，以SC32R803为例，CAN_TypeDef在文件“sc32r803.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

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 580

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_FDMODECmd	配置 CAN 发送帧的格式
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_RxDataALLStorageMode	使能或失能 CAN 完全存储模式
CAN_PTBAutoRetransMode	配置 CAN PTB 自动重传模式
CAN_STBAutoRetransMode	配置 CAN STB 自动重传模式
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_TxDCompensateCmd	配置 CAN 发送延迟补偿
CAN_TransStop	配置 CAN 发送中止
CAN_RecieveConfig	CAN 数据接收读取
CAN_ErrorThresholdconfig	配置 CANx 错误阈值。
CAN_TIMEPOSConfig	TTCAN 选择帧开始作为时间戳位置
CAN_TIMEENCmd	TTCAN 启用和禁用
CAN_TIMECounterCmd	TTCAN 计数启用和禁用
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_PTBTransStop	配置 PTB 发送中止
CAN_ReadBuffDataSize	获取已存储消息的大小
CAN_ReadErrorCode	获取 CAN 传输的错误类型
CAN_ReadALC	获取 CAN 总线仲裁中丢失（竞争失败）的位置（节点/站点）
CAN_GetRACTIVE	获取是否有接收操作正在进行
CAN_GetTACTIVE	获取是否有发送操作正在进行

CAN 固件库函数详解

CAN_DeInit

Table 581

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 582

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;
FunctionalState CANTXEN;
FunctionalState CANRXEN;
FunctionalState CANTimeStampEN;
FunctionalState CANTimeStampCntEN;
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 583获得这个参数的所有成员。

Table 583

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 584获得这个参数的所有成员。

Table 584

CANROMSelect	描述
CAN_ROM_Old	丢弃新接收的报文。
CAN_ROM_New	最早的报文将丢弃

CAN_MOD

CAN工作模式设定，参阅Table 585获得这个参数的所有成员。

Table 585

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 586获得这个参数的所有成员。

Table 586

CANTSMODESelect	描述
CAN_TxFIFO	FIFO 发送
CAN_IDPriority	以 ID 为优先级发送

CANFDISOSelect

CAN FD模式下发送IOS配置，参阅Table 587获得这个参数的所有成员。

Table 587

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 588获得这个参数的所有成员。

Table 588

CANTimeStampPositon	描述
CAN_TimeStampPosition_SOF	时间戳位置 SOF
CAN_TimeStampPosition_EOF	时间戳位置 EOF

CANTBUFSelect

CAN发送数据存贮选择位，参阅Table 589获得这个参数的所有成员。

Table 589

CANTBUFSelect	描述
CAN_TBUFTx_PTB	CAN 发送数据 PTB 存储
CAN_TBUFTx_STB	CAN 发送数据 STB 存储

TransMod

CAN发送数据方式设置，参阅Table 590获得这个参数的所有成员。

Table 590

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(36000000,8,CAN_Baudrate_250kHz);    //快速时钟波特率为250k
CAN_SBaudRate_Select(36000000,8,CAN_Baudrate_250kHz);    //慢速时钟波特率为250k
CAN_InitStruct.CANFDFrame = CAN_FDMode_on;       //CAN_FD帧
CAN_InitStruct.CANFDISOSelect = CAN_ISO;     //采用ISO CAN_FD模式
CAN_InitStruct.CANTBUFSelect = CAN_TBUFTx_STB;//采用次级发送缓存
CAN_InitStruct.CANROMSelect = CAN_ROM_Old;    //丢弃老报文
CAN_InitStruct.CANTSMODESelect = CAN_TxFIFO;    //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 591

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 592

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 589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 593

函数名	CAN_FBaudRate_Select
函数原型	ErrorStatus CAN_FBaudRate_Select (CAN_TypeDef *CANx,uint32_t CAN_Clk, uint32_t CAN_FPrescaler, uint32_t BaudRate)
功能描述	配置 CAN 外设的高速波特率
输入参数 1	CANx：来选择 CAN 外设
输入参数 2	CAN_Clk：CAN 工作时钟
输入参数 3	CAN_FPrescaler ： 预分频
输入参数 4	BaudRate：选择配置 CAN 的低速波特率
返回值	设置状态

BaudRate

CAN高速波特率设定，参阅Table 594获得这个参数的所有成员。

Table 594

BaudRate	描述
CAN_Baudrate_125kHz	高速波特率设置为 125KHZ
CAN_Baudrate_250kHz	高速波特率设置为 250KHZ
CAN_Baudrate_500kHz	高速波特率设置为 500KHZ
CAN_Baudrate_800kHz	高速波特率设置为 800KHZ
CAN_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(CAN,36000000,8,CAN_Baudrate_250kHz);
```

CAN_SBaudRate_Select

Table 595

函数名	CAN_SBaudRate_Select
函数原型	ErrorStatus CAN_SBaudRate_Select (CAN_TypeDef *CANx, uint32_t CAN_Clk, uint32_t CAN_SPrescaler, uint32_t BaudRate)
功能描述	配置 CAN 外设的低速波特率
输入参数 1	CANx：来选择 CAN 外设
输入参数 2	CAN_Clk：CAN 工作时钟
输入参数 3	CAN_SPrescaler：预分频
输入参数 4	BaudRate：选择配置 CAN 的低速波特率
返回值	设置状态

BaudRate为CAN低速波特率设定，参阅Table 594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(CAN,36000000,8,CAN_Baudrate_250kHz);
```

CAN_FDMODECmd

Table 596

函数名	CAN_FDMODECmd
函数原型	void CAN_FDMODECmd (CAN_TypeDef *CANx, FunctionalState NewState)
功能描述	配置 CAN 发送帧的格式
输入参数 1	CANx: 来选择 CAN 外设
输入参数 2	NewState：使能 CAN 发送 FD 帧 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/ *使能CAN 发送FD帧*/
CAN_FDMODECmd (CAN, ENABLE);
```

CAN_Trans_Select

Table 597

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 590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_RxDataALLStorageMode

Table 598

函数名	CAN_RxDataALLStorageMode
函数原型	void CAN_RxDataALLStorageMode (CAN_TypeDef *CANx, FunctionalState NewState)
功能描述	使能或失能 CAN 完全存储模式
输入参数 1	CANx: 来选择 CAN 外设
输入参数 2	NewState：使能或失能 CAN 完全存储模式 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

/ *使能CAN 完全存储模式*/
CAN_RxDataALLStorageMode (CAN, ENABLE);

CAN_PTBAutoRetransMode

Table 599

函数名	CAN_PTBAutoRetransMode
函数原型	void CAN_PTBAutoRetransMode (CAN_TypeDef *CANx, FunctionalState NewState)
功能描述	配置 CAN PTB 自动重传模式
输入参数 1	CANx: 来选择 CAN 外设
输入参数 2	NewState：使能或失能 CAN PTB 自动重传模式 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:
/ *配置CAN PTB 自动重传模式*/
CAN_PTBAutoRetransMode (CAN, ENABLE);

CAN_STBAutoRetransMode

Table 600

函数名	CAN_STBAutoRetransMode
函数原型	void CAN_STBAutoRetransMode (CAN_TypeDef *CANx, FunctionalState NewState)
功能描述	配置 CAN STB 自动重传模式
输入参数 1	CANx: 来选择 CAN 外设
输入参数 2	NewState：使能或失能 CAN STB 自动重传模式 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:
/ *配置CAN STB 自动重传模式*/
CAN_STBAutoRetransMode (CAN, ENABLE);

CAN_RxThresholdConfig

Table 601

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 602

函数名	CAN_FliterInit
函数原型	void CAN_FliterInit (CAN_TypeDef *CANx, CAN_FliterTypeDef *CAN_FliterStruct)
功能描述	配置 CAN 过滤器
输入参数 1	CANx: 来选择 CAN 外设
输入参数 2	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 603获得这个参数的所有成员。

Table 603

CAN_Fliter	描述
CAN_Fliter0	选择过滤器 0
CAN_Fliter1	选择过滤器 1
CAN_Fliter2	选择过滤器 2
CAN_Fliter3	选择过滤器 3
CAN_Fliter4	选择过滤器 4
CAN_Fliter5	选择过滤器 5
CAN_Fliter6	选择过滤器 6
CAN_Fliter7	选择过滤器 7

CANFliterFrame

CAN过滤帧格式设置位，参阅Table 604获得这个参数的所有成员。

Table 604

CANFliterFrame	描述
CAN_Fliter_Standard	只接收标准帧
CAN_Fliter_Extension	只接收扩展帧
CAN_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 = CAN_Fliter_Default;//过滤器帧格式不设限制
CAN_FliterStruct.CAN_Fliter = CAN_Fliter0;        //配置过滤器0
CAN_FliterInit(&CAN_FliterStruct);
```

CAN_ModeSelect

Table 605

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 585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 606

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[16];
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 607获得这个参数的所有成员。

Table 607

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 608获得这个参数的所有成员。

Table 608

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 609获得这个参数的所有成员。

Table 609

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 610获得这个参数的所有成员。

Table 610

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_TxDCompensateCmd

Table 611

函数名	CAN_TxDCompensateCmd
函数原型	void CAN_TxDCompensateCmd (CAN_TypeDef *CANx, FunctionalState NewState)
功能描述	配置 CAN 发送延迟补偿
输入参数 1	CANx: 来选择 CAN 外设
输入参数 2	NewState：使能或失能 CAN 发送延迟补偿 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/ * 配置 CAN 发送延迟补偿*/
CAN_TxDCompensateCmd (CAN, ENABLE);
```

CAN_TransStop

Table 612

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:

```
/ * 设置CAN STB传输中止*/
CAN_TransStop (CAN);
```

CAN_RecieveConfig

Table 613

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[16];/*!< 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_ErrorThresholdconfig

Table 614

函数名	CAN_ErrorThresholdconfig
函数原型	void CAN_ErrorThresholdconfig (CAN_TypeDef* CANx, uint8_t Errorcnt)
功能描述	配置 CANx 错误阈值。
输入参数 1	CANx: 来选择 CAN 外设
输入参数 2	Errorcnt: 设置 CANx 错误阈值
返回值	无

使用示例:

```
/*配置 CANx 错误阈值*/
CAN_ErrorThresholdconfig ( CAN, 0xF0 );
```

CAN_TIMEPOSConfig

Table 615

函数名	CAN_TIMEPOSConfig
函数原形	void CAN_TIMEPOSConfig (CAN_TypeDef* CANx, CANTimeStampPosition_TypeDef CANTimeStampPosition)
功能描述	TTCAN 选择帧开始作为时间戳位置
输入参数 1	CANx: 来选择 CAN 外设
输入参数 2	CANTimeStampPosition：选择帧开始作为时间戳位置
返回值	无

CANTimeStampPosition

CANTimeStampPosition TTCAN 选择帧开始作为时间戳位置，参阅 Table 616获得这个参数的所有成员。

Table 616

CANTimeStampPosition	描述
CAN_TimeStampPosition_SOF	选择帧开始作为时间戳位置
CAN_TimeStampPosition_EOF	选择帧结尾作为时间戳位置

使用示例:

```
/* TTCAN 选择帧开始作为时间戳位置 */
CAN_TIMEPOSConfig (CAN, CAN_TimeStampPosition_SOF);
```

CAN_TIMEENCmd

Table 617

函数名	CAN_TIMEENCmd
函数原型	void CAN_TIMEENCmd (CAN_TypeDef* CANx, FunctionalState NewState)
功能描述	TTCAN 启用和禁用
输入参数 1	CANx: 来选择 CAN 外设
输入参数 2	NewState：使能或失能 TTCAN 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/ * 使能TTCAN*/
CAN_TxDCompensateCmd (CAN, ENABLE);
```

CAN_TIMECounterCmd

Table 618

函数名	CAN_TIMECounterCmd
函数原型	void CAN_TIMECounterCmd (CAN_TypeDef* CANx, FunctionalState NewState)
功能描述	TTCAN 计数启用和禁用
输入参数 1	CANx: 来选择 CAN 外设
输入参数 2	NewState：使能或失能 TTCAN 计数 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/ * 使能TTCAN 计数*/
CAN_TIMECounterCmd (CAN, ENABLE);
```

CAN_TxDCompensateCmd

Table 619

函数名	CAN_TxDCompensateCmd
函数原型	void CAN_TxDCompensateCmd (CAN_TypeDef *CANx, FunctionalState NewState)
功能描述	配置 CAN 发送延迟补偿
输入参数 1	CANx: 来选择 CAN 外设
输入参数 2	NewState：使能或失能 CAN 发送延迟补偿 这个参数可以取：ENABLE 或者 DISABLE
返回值	无

使用示例:

```
/ * 配置 CAN 发送延迟补偿*/
CAN_TxDCompensateCmd (CAN, ENABLE);
```

CAN_ITConfig

Table 620

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_INT

CAN_INT请求选择，参阅Table 621获得这个参数的所有成员。

Table 621

CAN_INT	描述
CAN_INTEN	CAN 总中断使能
CAN_RIEEN	接收中断请求
CAN_ROIEEN	接收溢出中断请求
CAN_RFIEEN	接收 RB 已满中断请求
CAN_RAFIEEN	接收超过或等于接收阈值中断请求
CAN_TPIEEN	PTB 发送完成中断请求
CAN_TSIEEN	STB 发送完成中断请求
CAN_EIEEN	错误中断请求
CAN_ALIEEN	传输终止中断请求
CAN_BEIEEN	总线错误中断请求
CAN_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, CAN_RIEEN,ENABLE);
```

CAN_GetFlagStatus

Table 622

函数名	CAN_GetFlagStatus
函数原形	FlagStatus 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 623获得这个参数的所有成员。

Table 623

CAN_FLAG	描述
CAN_FIAG_RIF	接收中断标志位
CAN_FIAG_TSFF	发送区域满标志位
CAN_FIAG_AIF	传输中止标志位
CAN_FIAG_EIF	错误中断标志位
CAN_FIAG_TSIF	PTB 发送完成标志位
CAN_FIAG_TPIF	STB 发送完成标志位
CAN_FIAG_RAFIF	阈值警告标志位
CAN_FIAG_RFIF	RB 已满标志位
CAN_FIAG_ROIF	RB 溢出标志位
CAN_FIAG_ALIF	仲裁丢失中断标志
CAN_FIAG_BEIF	总线错误中断标志
CAN_FIAG_EPIF	错误被动中断标志
CAN_FIAG_EPASS	错误被动中断使能位
CAN_FIAG_EWARN	错误计数阈值警告标志位

使用示例:

```
/* 获取接收中断标志状态 */
CAN_GetFlagStatus(CAN, CAN_FIAG_RIF);
```

CAN_ClearFlag

Table 624

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 623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_FIAG_RIF);
```

CAN_GetTECNT

Table 625

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 626

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 627

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

```
CanRxMsg RxMessage;
static volatile FlagStatus CAN_Flag = RESET;

//CAN发送数据配置
void CAN_Transmit(void)
{
        CAN_TBUFSelect(CAN, CAN_TBUFTx_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_TBUFSelect(CAN, CAN_TBUFTx_PTB);      //当前报文写入区域选择PTB
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 = 0x222;                        //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] = 0x11111111;        //写入待发送数据
CAN_Tx_msg.Tx_msg[1] = 0x22222222;
CAN_Tx_msg.Tx_msg[2] = 0x33333333;
CAN_Tx_msg.Tx_msg[3] = 0x44444444;
CAN_Tx_msg.Tx_msg_len = 4;
CAN_Trans_Init(CAN,&CAN_Tx_msg);

CAN_Trans_Select(CAN,CAN_TransMod_TPE);                 //传输PTB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_TBUFSelect(CAN, CAN_TBUFTx_STB);//当前报文写入区域选择STB
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标准
CAN_Tx_msg.CAN_TXID = 0x333;                                //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] = 0x31323334;        //写入待发送数据
CAN_Tx_msg.Tx_msg[1] = 0x35363738;
CAN_Tx_msg.Tx_msg[2] = 0x30303030;
CAN_Tx_msg.Tx_msg[3] = 0x33333333;
CAN_Tx_msg.Tx_msg_len = 4;
CAN_Trans_Init(CAN, &CAN_Tx_msg);

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_TBUFSelect(CAN, CAN_TBUFTx_STB);//当前报文写入区域选择STB
CAN_Tx_msg.CAN_IDE = CAN_Extended_format;          //选择拓展帧
CAN_Tx_msg.CAN_RTR = CAN_Remote_frame;                  //选择远程帧
CAN_Tx_msg.CAN_BRS = CAN_Disable_BRS;          //选择不变波特率
CAN_Tx_msg.CAN_FDF = CAN_Standard_frame;        //选择CAN2.0标准
CAN_Tx_msg.CAN_TXID = 0x444;                                //配置发送帧id
CAN_Tx_msg.Data_len = CAN_DLC_8;                        //配置发送数据长度,配置DLC长度
CAN_Tx_msg.Tx_msg_len = 0;
CAN_Trans_Init(CAN, &CAN_Tx_msg);
```

```
    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 = CAN_Fliter_Default;//过滤器帧格式不设限制
    CAN_FliterStruct.CAN_Fliter = CAN_Fliter0;       //配置过滤器0
    CAN_FliterInit(CAN,&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 = CAN_Fliter_Default;//过滤器帧格式不设限制
    CAN_FliterStruct.CAN_Fliter = CAN_Fliter1;       //配置过滤器1
    CAN_FliterInit(CAN,&CAN_FliterStruct);
  //配置过滤器2
    CAN_FliterStruct.Acode = 0x333;              //设置过滤id 0x333
    CAN_FliterStruct.Amask = 0x000;              //设置过滤掩码
    CAN_FliterStruct.CANFliterFrame = CAN_Fliter_Default;//过滤器帧格式不设限制
    CAN_FliterStruct.CAN_Fliter = CAN_Fliter2;       //配置过滤器2
    CAN_FliterInit(CAN,&CAN_FliterStruct);
  //配置过滤器3
    CAN_FliterStruct.Acode = 0x444;              //设置过滤id 0x444
    CAN_FliterStruct.Amask = 0x000;              //设置过滤掩码
    CAN_FliterStruct.CANFliterFrame = CAN_Fliter_Default;//过滤器帧格式不设限制
    CAN_FliterStruct.CAN_Fliter = CAN_Fliter3;       //配置过滤器3
    CAN_FliterInit(CAN,&CAN_FliterStruct);
}


void CAN_Communication(void)
{

  UART_PinRemapConfig(UART1,UART_PinRemap_A);   //打印重映射至UART1 TX PB4

  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=36MHz
  RCC_HCLKConfig(RCC_SYSCLK_Div1);                                     //HCLK = SYSCLK/1
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_CAN,ENABLE);     //使能CAN时钟

  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(36000000,8,CAN_Baudrate_250kHz);    //快速时钟波特率为250k
  CAN_SBaudRate_Select(36000000,8,CAN_Baudrate_250kHz);    //慢速时钟波特率为250k
  CAN_InitStruct.CANFDFrame = CAN_FDMode_on;     //CAN_FD帧
  CAN_InitStruct.CANFDISOSelect = CAN_ISO;    //采用ISO CAN_FD模式
  CAN_InitStruct.CANTBUFSelect = CAN_TBUFTx_STB;//采用次级发送缓存
```

```
    CAN_InitStruct.CANROMSelect = CAN_ROM_Old;    //丢弃老报文
    CAN_InitStruct.CANTSMODESelect = CAN_TxFIFO;    //STB采用FIFO发送模式
    CAN_InitStruct.CAN_MOD = CAN_Normal;       //正常模式：CAN_Normal 只听模式：CAN_Listenonly 待机模式：
CAN_StandBy
                                                //外部环回模式：CAN_LoopBack 内部环回模式：CAN_LoopBackIn 自
应答环回模式:CAN_LoopBackSack
    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, CAN_INTEN|CAN_RIEEN|CAN_ROIEEN|CAN_TSIEEN |CAN_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_Communication_CANHandler(void)
{
    if(CAN_GetFlagStatus(CAN, CAN_RTIE_RIF) == SET) //接收中断
    {

       CAN_ClearFlag(CAN, CAN_RTIE_RIF);
    }

    if(CAN_GetFlagStatus(CAN, CAN_RTIE_RAFIF) == SET)//RB slots阈值警告中断
    {
       CAN_Flag = SET;
       CAN_ClearFlag(CAN, CAN_RTIE_RAFIF);
    }

    if(CAN_GetFlagStatus(CAN, CAN_RTIE_TPIF) == SET)    //PTB传输完成中断
    {
       CAN_ClearFlag(CAN, CAN_RTIE_TPIF);
    }

    if(CAN_GetFlagStatus(CAN, CAN_RTIE_TSIF) == SET)//STB传输完成中断
    {
       CAN_ClearFlag(CAN, CAN_RTIE_TSIF);
    }

}
```

28 VREF 固件库函数

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 628

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 629

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 630获得这个参数的所有成员。

Table 630

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 631

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 632

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);
```

29 Temper 固件库函数

赛元SC32F15Gx、SC32L14xx、SC32R807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 633

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 634

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 635

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 636

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 637

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 638

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++);
      }
    }
}
```

30 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 639

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 640

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 641

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 642

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 643获得这个参数的所有成员。

Table 643

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 644

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 */
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_IFB,ENABLE);
WDT_Cmd(ENABLE);
```

WDT_SetReload

Table 645

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();
}
```

31 PWR 固件库函数

PWR 固件库函列表

Table 646

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 647

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 648获得这个参数的所有成员。

Table 648

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 649

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 650获得这个参数的所有成员。

Table 650

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);
```

32 NVIC 固件库函数

NVIC 固件库函数列表

Table 651

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

NVIC 固件库函数详解

NVIC_SetPriorityGrouping

Table 652

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 653

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 654

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 655获得这个参数的所有成员。

Table 655

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 656

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 655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 657

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 655获得IRQn的所有成员。

使用示例:

```
NVIC_DisableIRQ (INT0_IRQn);//禁用INT0中断
```

NVIC_GetPendingIRQ

Table 658

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 655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 659

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 655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 660

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 655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 661

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 655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 662

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 663

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 655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 664

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
```

33 LPC 固件库函数

赛元SC32L14xx、SC32R807系列内部集成一组低功耗计数器（简称LPC），可与线性/增量编码器等设备连接，用于获取计数、方向等信息。LPC支持在STOP模式下计数，减少MCU唤醒频率，降低整体功耗。

LPC 寄存器结构

赛元SC32L14xx、SC32R807系列LPC寄存器结构，LPC_TypeDef，在文件“sc32L14xx.h”和“sc32r807.h”中定义如下：

```
typedef struct
{
  __IO uint32_t LPC_CON;
  __IO uint32_t LPC_STS;
  __IO uint32_t LPCA_CNT;
  __IO uint32_t LPCB_CNT;
  __IO uint32_t LPCA_MAX;
  __IO uint32_t LPCB_MAX;
  __IO uint32_t LPC_IDE;
} LPC_TypeDef;
```

LPC 寄存器表格

Table 663

寄存器	描述
LPC_CON	LPC低功耗计数器控制寄存器
LPC_STS	LPC低功耗计数器标志状态位寄存器
LPCA_CNT	INTA有效边沿计数值寄存器
LPCB_CNT	INTB有效边沿计数值寄存器
LPCA_MAX	INTA有效边沿计数溢出值寄存器
LPCB_MAX	INTB有效边沿计数溢出值寄存器
LPC_IDE	LPC低功耗计数器中断使能寄存器

LPC 固件库函数列表

Table 664

函数名	描述
LPC_DeInit	将LPCx寄存器设置为缺省值
LPC_Init	根据LPC_InitStruct 中指定的参数初始化外设LPCx寄存器
LPC_StructInit	把LPC_InitStruct 中的每一个参数按缺省值填入
LPC_Cmd	使能或者失能LPCx外设
LPC_TriggerMode	LPCx触发方式选择
LPC_SetOverflowCounter	设置LPCx有效边沿计数溢出值
LPC_GetOverflowCounter	获取LPCx有效边沿计数溢出值
LPC_SetCounter	设置LPCx有效边沿计数值
LPC_GetCounter	获取LPCx有效边沿计数值
LPC_ITConfig	使能或者失能指定的LPCx中断
LPC_GetFlagStatus	获取指定LPCx的标志位状态
LPC_ClearFlag	清除指定LPCx的标志位状态

LPC 固件库函数详解

LPC_DeInit

Table 665

函数名	LPC_DeInit
函数原型	void LPC_DeInit (LPC_TypeDef* LPCx)
功能描述	将LPCx寄存器设置为缺省值
输入参数	LPCx：选择LPC外设
返回值	无

使用示例:

```
/* LPC寄存器复位 */
LPC_DeInit(LPC);
```

LPC_Init

Table 666

函数名	LPC_Init
函数原型	void LPC_Init(LPC_TypeDef* LPCx,LPC_InitTypeDef* LPC_InitStruct)
功能描述	根据LPC_InitStruct 中指定的参数初始化外设LPCx寄存器
输入参数1	LPCx：选择LPC外设
输入参数2	LPC_InitStruct：指向结构LPC_InitTypeDef 的指针，包含了外设LPC的配置信息
返回值	无

LPC_InitTypeDef

LPC_InitTypeDef 定义于文件“sc32f1xxx_lpcounter.h”：

```
typedef struct
{
    uint32_t LPC_INTATrigger;
    uint32_t LPC_INTBTrigger;
    uint32_t LPC_INTAOverValue;
    uint32_t LPC_INTBOverValue;
    uint32_t LPC_InitDir;
    uint32_t LPC_DisDirEN;
} LPC_InitTypeDef;
```

LPC_INTATrigger

LPC INTA上升沿/下降沿计数选择，参阅Table 667获得这个参数的所有成员。

Table 667

LPC_INTATrigger	描述
LPC_Trigger_Rising	检测到INTA引脚上一个上升沿，产生一个计数
LPC_Trigger_Falling	检测到INTA引脚上一个下降沿，产生一个计数

LPC_INTBTrigger

LPC INTB上升沿/下降沿计数选择，参阅Table 668获得这个参数的所有成员。

Table 668

LPC_INTBTrigger	描述
LPC_Trigger_Rising	检测到INTB引脚上一个上升沿，产生一个计数
LPC_Trigger_Falling	检测到INTB引脚上一个下降沿，产生一个计数

LPC_INTAOverValue

INTA有效边沿计数溢出值设置。

LPC_INTBOverValue

INTA有效边沿计数溢出值设置。

LPC_InitDir

初始方向设置，参阅Table 669获得这个参数的所有成员。

Table 669

LPC_InitDir	描述
LPC_INITDIR_Forward	设定初始方向为“正转”
LPC_INITDIR_Reversal	设定初始方向为“反转”

LPC_DisDirEN

方向判断屏蔽设置，选择“ENABLE”或者“DISABLE”。

使用示例:

```
/* 根据LPC _InitStruct 中指定的参数初始化LPC */
LPC_InitTypeDef LPC_InitStruct;
LPC_InitStruct.LPC_InitDir = LPC_INITDIR_Forward; //初始方向为正转，正转计入INTA_CNT，反转计入INTB_CNT
LPC_InitStruct.LPC_DisDirEN = ENABLE;  //方向屏蔽使能，屏蔽方向计数，有边沿即放入INTA_CNT或者INTB_CNT
LPC_InitStruct.LPC_INTATrigger = LPC_Trigger_Rising; //上升沿计数
LPC_InitStruct.LPC_INTBTrigger = LPC_Trigger_Rising; //上升沿计数
LPC_InitStruct.LPC_INTAOverValue = 0XA;  //INTA_CNT溢出计数
LPC_InitStruct.LPC_INTBOverValue = 0XA;  //INTB_CNT溢出数
LPC_Init(LPC,&LPC_InitStruct);
```

LPC_StructInit

Table 670

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

使用示例:

```
/* 把 LPC_InitStruct 中的每一个参数按缺省值填入 */
LPC_InitTypeDef LPC_InitStruct;                    //定义LPC初始化结构体
LPC_StructInit(&LPC_InitStruct);
```

LPC_Cmd

Table 671

函数名	LPC_Cmd
函数原型	void LPC_Cmd(LPC_TypeDef* LPCx,FunctionalState NewState)
功能描述	使能或者失能LPCx外设
输入参数1	LPCx：选择LPC外设
输入参数2	NewState：LPC功能开启/关闭，可取值ENABLE或DISABLE
返回值	无

使用示例:

```
/* LPC使能 */
LPC_Cmd(LPC,ENABLE);
```

LPC_TriggerMode

Table 672

函数名	LPC_TriggerMode
函数原型	void LPC_TriggerMode (LPC_TypeDef* LPCx, LPC_INTSEL_Typedef LPC_INTSEL, LPC_Trigger_TypeDef Trigger_Mode)
功能描述	LPCx触发方式选择
输入参数1	LPCx：选择LPC外设
输入参数2	LPC_INTSEL：选择LPCx通道，INTA或者INTB
输入参数3	Trigger_Mode：选择触发模式，上升沿或者下降沿
返回值	无

LPC_INTSEL

选择LPCx通道，INTA或者INTB，参阅Table 673获得这个参数的所有成员。

Table 673

LPC_INTSEL	描述

LPC_INTSEL_INTA	选择LPCx的有效沿来源于INTA
LPC_INTSEL_INTB	选择LPCx的有效沿来源于INTB

Trigger_Mode

选择触发模式，上升沿或者下降沿，参阅Table 674获得这个参数的所有成员。

Table 674

Trigger_Mode	描述
LPC_Trigger_Rising	检测到INTA或者INTB引脚上一个上升沿，产生一个计数
LPC_Trigger_Falling	检测到INTA或者INTB引脚上一个下降沿，产生一个计数

使用示例:

```
/* 设置LPC INTA上升沿计数 */
LPC_TriggerMode(LPC, LPC_INTSEL_INTA, LPC_Trigger_Rising);
```

LPC_SetOverflowCounter

Table 675

函数名	LPC_SetOverflowCounter
函数原型	void LPC_SetOverflowCounter (LPC_TypeDef* LPCx, LPC_INTSEL_Typedef LPC_INTSEL,uint8_t Counter)
功能描述	设置LPCx有效边沿计数溢出值
输入参数1	LPCx：选择LPC外设
输入参数2	LPC_INTSEL：选择LPCx通道，INTA或者INTB
输入参数3	Counter：有效边沿计数溢出值
返回值	无

LPC_INTSEL

选择LPCx通道，INTA或者INTB，参阅Table 673获得这个参数的所有成员。

Counter

有效边沿计数溢出值。

使用示例:

```
/* 设置LPC INTA通道有效边沿计数溢出值为0x0F  */
LPC_SetOverflowCounter (LPC, LPC_INTSEL_INTA,0x0F );
```

LPC_GetOverflowCounter

Table 676

函数名	LPC_GetOverflowCounter
函数原型	uint8_t LPC_GetOverflowCounter (LPC_TypeDef* LPCx, LPC_INTSEL_Typedef LPC_INTSEL)
功能描述	获取LPCx有效边沿计数溢出值
输入参数1	LPCx：选择LPC外设
输入参数2	LPC_INTSEL：选择LPCx通道，INTA或者INTB
返回值	LPCx有效边沿计数溢出值

LPC_INTSEL

选择LPCx通道，INTA或者INTB，参阅Table 673获得这个参数的所有成员。

使用示例:

```
uint8_t   OverflowCounter;
/* 获取LPC INTA通道有效边沿计数溢出值  */
OverflowCounter = LPC_ GetOverflowCounter (LPC, LPC_INTSEL_INTA);
```

LPC_SetCounter

Table 677

函数名	LPC_SetCounter
函数原型	void LPC_SetCounter (LPC_TypeDef* LPCx, LPC_INTSEL_Typedef LPC_INTSEL,uint8_t Counter)

功能描述	设置LPCx有效边沿计数值
输入参数1	LPCx：选择LPC外设
输入参数2	LPC_INTSEL：选择LPCx通道，INTA或者INTB
输入参数3	Counter：有效边沿计数值
返回值	无

LPC_INTSEL

选择LPCx通道，INTA或者INTB，参阅Table 673获得这个参数的所有成员。

Counter

有效边沿计数值。

使用示例:

```
/* 设置LPC INTA通道有效边沿计数值为0x0F  */
LPC_SetCounter (LPC, LPC_INTSEL_INTA,0x0F);
```

LPC_GetCounter

Table 678

函数名	LPC_GetCounter
函数原型	uint8_t LPC_GetCounter (LPC_TypeDef* LPCx, LPC_INTSEL_Typedef LPC_INTSEL)
功能描述	获取LPCx有效边沿计数值
输入参数1	LPCx：选择LPC外设
输入参数2	LPC_INTSEL：选择LPCx通道，INTA或者INTB
返回值	LPCx有效边沿计数值

LPC_INTSEL

选择LPCx通道，INTA或者INTB，参阅Table 673获得这个参数的所有成员。

使用示例:

```
uint8_t   Counter;
/* 获取LPC INTA通道有效边沿计数值*/
Counter = LPC_ GetCounter (LPC, LPC_INTSEL_INTA);
```

LPC_ITConfig

Table 679

函数名	LPC_ITConfig
函数原型	void LPC_ITConfig(LPC_TypeDef* LPCx,uint8_t LPC_IT,FunctionalState NewState)
功能描述	使能或者失能指定的LPCx中断
输入参数1	LPCx：选择LPC外设
输入参数2	LPC_IT：待使能或者失能的LPC中断源
输入参数3	NewState：LPC中断功能开启/关闭，可取值ENABLE或DISABLE
返回值	无

LPC_IT

LPC中断选择，参阅Table 680获得这个参数的所有成员。

Table 680

LPC_IT	描述
LPC_IT_INTEN	中断请求 CPU的使能控制位
LPC_IT_DIRIE	方向跳变中断使能位
LPC_IT_CAIE	LPCA_CNT计数溢出中断使能位
LPC_IT_CBIE	LPCB_CNT计数溢出中断使能位
LPC_IT_RFAIE	INTA输入有效沿中断使能位
LPC_IT_RFBIE	INTB输入有效沿中断使能位

使用示例:

```
/*打开中断请求 CPU的使能和方向跳变中断使能*/
LPC_ITConfig(LPC, LPC_IT_INTEN| LPC_IT_DIRIE, ENABLE);
```

LPC_GetFlagStatus

Table 681

函数名	LPC_GetFlagStatus
函数原型	FlagStatus LPC_GetFlagStatus(LPC_TypeDef* LPCx,LPC_Flag_TypeDef LPC_FLAG)
功能描述	获取指定LPCx的标志位状态
输入参数1	LPCx：选择LPC外设
输入参数2	LPC_FLAG：选择LPC的标志位
返回值	LPC _FLAG 的新状态

LPC_FLAG

选择LPC的标志位，参阅Table 682获得这个参数的所有成员。

Table 682

LPC_FLAG	描述
LPC_FLAG_FLIPIF	方向翻转标志位
LPC_FLAG_DIRIF	该位为状态位，只读，由硬件置1及清0
LPC_FLAG_CAIF	LPCA计数溢出标志位
LPC_FLAG_CBIF	LPCB计数溢出标志位
LPC_FLAG_RFAIF	INTA输入有效沿被检测到的标志位
LPC_FLAG_RFBIF	INTB输入有效沿被检测到的标志位

使用示例:

```
/* 检查方向翻转标志位设置与否 */
FlagStatus  LPC_Status;
LPC_Status  = LPC _GetFlagStatus (LPC, LPC_FLAG_FLIPIF);
```

LPC_ClearFlag

Table 683

函数名	LPC_ClearFlag
函数原型	void LPC_ClearFlag(LPC_TypeDef* LPCx,uint8_t LPC_FLAG)
功能描述	清除指定LPCx的标志位状态
输入参数1	LPCx：选择LPC外设
输入参数2	LPC _FLAG：LPC的标志位，定义在LPC_Flag_TypeDef中
返回值	无

LPC_FLAG

选择LPC的标志位，参阅Table 682获得这个参数的所有成员。

使用示例:

```
/* 清除方向翻转标志位 */
LPC _ ClearFlag (LPC, LPC_FLAG_FLIPIF);
```

综合使用示例

```
volatile FlagStatus LPC_INTAFlag = RESET;
volatile FlagStatus LPC_INTBFlag = RESET;

void LPC_Example_Function(void)
{
  GPIO_InitTypeDef GPIOInit_PD00_Struct;
    GPIOInit_PD00_Struct.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1;
    GPIOInit_PD00_Struct.GPIO_Mode = GPIO_Mode_IN_PU;
    GPIOInit_PD00_Struct.GPIO_DriveLevel = 0;
    GPIO_Init(GPIOD,&GPIOInit_PD00_Struct);

    RCC_APB1Cmd(ENABLE);
    LPC_InitTypeDef LPC_InitStruct;
    LPC_StructInit(&LPC_InitStruct);
```

```
        LPC_InitStruct.LPC_InitDir = LPC_INITDIR_Forward; //初始方向为正转，正转计入INTA_CNT，反转计入INTB_CNT
        LPC_InitStruct.LPC_DisDirEN = ENABLE;  //方向屏蔽使能，屏蔽方向计数，有边沿即放入INTA_CNT或者INTB_CNT
        LPC_InitStruct.LPC_INTATrigger = LPC_Trigger_Rising; //上升沿计数
        LPC_InitStruct.LPC_INTBTrigger = LPC_Trigger_Rising; //上升沿计数
        LPC_InitStruct.LPC_INTAOverValue = 0XA;  //INTA_CNT溢出计数
        LPC_InitStruct.LPC_INTBOverValue = 0XA;  //INTB_CNT溢出数
        LPC_Init(LPC,&LPC_InitStruct);
        LPC_ITConfig(LPC,LPC_IT_INTEN | LPC_IT_CBIE | LPC_IT_CAIE,ENABLE);//开启中断
        NVIC_EnableIRQ(LPC_IRQn);
        LPC_Cmd(LPC,ENABLE);

        while(1)
        {
                printf("Enter STOP mode\r\n");
                PWR_EnterSTOPMode(PWR_STOPEntry_WFE);
                printf("Exit the STOP mode\r\n");
                if(LPC_INTAFlag != RESET)
                {
                        LPC_INTAFlag = RESET;
                        printf("INTA Wake up the STOP mode\r\n");
                }
                if(LPC_INTBFlag != RESET)
                {
                        LPC_INTBFlag = RESET;
                        printf("INTB Wake up the STOP mode\r\n");
                }
                uint32_t delay_time=10000;
                while(delay_time--);
        }
    }

    void LPC_Example_IRQHandler(void)
    {
        if(LPC_GetFlagStatus(LPC,LPC_FLAG_FLIPIF) != RESET) //方向翻转标志位
        {
                LPC_ClearFlag(LPC,LPC_FLAG_FLIPIF);
        }
        if(LPC_GetFlagStatus(LPC,LPC_FLAG_DIRIF) != RESET) //方向状态位
        {
        }
        else
        {
        }
        if(LPC_GetFlagStatus(LPC,LPC_FLAG_CAIF) != RESET) //LPCA计数溢出标志位
        {
                LPC_ClearFlag(LPC,LPC_FLAG_CAIF);
                LPC_INTAFlag = SET;
        }
        if(LPC_GetFlagStatus(LPC,LPC_FLAG_CBIF) != RESET) //LPCB计数溢出标志位
        {
                LPC_ClearFlag(LPC,LPC_FLAG_CBIF);
                LPC_INTBFlag = SET;
        }
        if(LPC_GetFlagStatus(LPC,LPC_FLAG_RFAIF) != RESET)//INTA输入有效沿被检测到的标志位
        {
                LPC_ClearFlag(LPC,LPC_FLAG_RFAIF);
        }
```

```
    if(LPC_GetFlagStatus(LPC,LPC_FLAG_RFBIF) != RESET)//INTB输入有效沿被检测到的标志位
    {
            LPC_ClearFlag(LPC,LPC_FLAG_RFBIF);
    }
}
```

34 LPD 固件库函数

赛元SC32L14xx、SC32R807系列内建有低电压监测电路（LPD），监测电源电压VDD，并将其与LPD门限电压VLPD阈值进行比较。

LPD 寄存器结构

赛元SC32L14xx、SC32R807系列LPD寄存器结构，LPC_TypeDef，在文件“sc32L14xx.h”或“sc32r807.h”中定义如下：

```
typedef struct
{
  __IO uint32_t LPD_CON;
  __IO uint32_t LPD_IDE;
} LPD_TypeDef;
```

LPC 寄存器表格

Table 684

寄存器	描述
LPD_CON	LPD控制寄存器
LPD_IDE	LPD中断使能寄存器

LPD 固件库函数列表

Table 685

函数名	描述
LPD_DeInit	将LPDx寄存器设置为缺省值
LPD_Cmd	使能或者失能LPDx外设
LPD_VLPDConfig	LPDx门限电压档位设置
LPD_ITConfig	使能或者失能指定的LPDx中断
LPD_GetFlagStatus	获取指定LPDx的标志位状态
LPD_ClearFlag	清除指定LPDx的标志位状态

LPD 固件库函数详解

LPD_DeInit

Table 686

函数名	LPD_DeInit
函数原型	void LPD_DeInit (LPD_TypeDef* LPDx)
功能描述	将LPDx寄存器设置为缺省值
输入参数	LPDx：选择LPD外设
返回值	无

使用示例:

```
/* LPD寄存器复位 */
LPD_DeInit(LPD);
```

LPD_Cmd

Table 687

函数名	LPD_Cmd
函数原型	void LPD_Cmd (LPD_TypeDef* LPDx, FunctionalState NewState)
功能描述	使能或者失能LPDx外设
输入参数1	LPDx：选择LPD外设
输入参数2	NewState：LPD功能开启/关闭，可取值ENABLE或DISABLE

返回值	无

使用示例:

```
/* LPD使能 */
LPD_Cmd(LPD,ENABLE);
```

LPD_VLPDConfig

Table 688

函数名	LPD_VLPDConfig
函数原型	void LPD_VLPDConfig (LPD_TypeDef* LPDx, LPD_IS_TypeDef LPD_VLPD)
功能描述	LPDx门限电压档位设置
输入参数1	LPDx：选择LPD外设
输入参数2	LPD_VLPD：LPD门限电压档位选择
返回值	无

LPD_VLPD

LPD门限电压档位选择，参阅Table 689获得这个参数的所有成员。

Table 689

LPD_VLPD	描述
VLPD_1_85V_typ	LPD门限电压为1.85V
VLPD_2_05V_typ	LPD门限电压为2.05V
VLPD_2_25V_typ	LPD门限电压为2.25V
VLPD_2_45V_typ	LPD门限电压为2.45V
VLPD_2_65V_typ	LPD门限电压为2.65V
VLPD_2_85V_typ	LPD门限电压为2.85V
VLPD_3_05V_typ	LPD门限电压为3.05V
VLPD_3_25V_typ	LPD门限电压为3.25V

使用示例:

```
/* 设置LPD门限电压为3.05V */
LPD_VLPDConfig (LPD, VLPD_3_05V_typ);
```

LPD_ITConfig

Table 690

函数名	LPD_ITConfig
函数原型	void LPD_ITConfig (LPD_TypeDef* LPDx, uint16_t LPD_IT, FunctionalState NewState)
功能描述	使能或者失能指定的LPDx中断
输入参数1	LPDx：选择LPD外设
输入参数2	LPD_IT：待使能或者失能的LPD中断源
输入参数3	NewState：LPD中断功能开启/关闭，可取值ENABLE或DISABLE
返回值	无

LPD_IT

LPD中断选择，参阅Table 691获得这个参数的所有成员。

Table 691

LPD_IT	描述
LPD_IT_INTEN	中断请求 CPU的使能控制位

使用示例:

```
/*打开中断请求 CPU的使能*/
LPD_ITConfig(LPD, LPD_IT_INTEN, ENABLE);
```

LPD_GetFlagStatus

Table 692

函数名	LPD_GetFlagStatus
函数原型	FlagStatus LPD_GetFlagStatus (LPD_TypeDef* LPDx, uint32_t LPD_FLAG)

功能描述	获取指定LPDx的标志位状态
输入参数1	LPDx：选择LPD外设
输入参数2	LPD_FLAG：选择LPD的标志位
返回值	LPD _FLAG 的新状态

LPD_FLAG

选择LPD的标志位，参阅Table 693获得这个参数的所有成员。

Table 693

LPD_FLAG	描述
LPD_Flag_LPDOF	LPD状态标志位
LPD_Flag_LPDIF	LPD中断请求标志

使用示例:

```
/* 检查LPD状态标志位设置与否 */
FlagStatus  LPD_Status;
LPD_Status  = LPD_GetFlagStatus (LPD, LPD_Flag_LPDOF);
```

LPD_ClearFlag

Table 694

函数名	LPD_ClearFlag
函数原型	void LPD_ClearFlag (LPD_TypeDef* LPDx, uint32_t LPD_FLAG)
功能描述	清除指定LPDx的标志位状态
输入参数1	LPDx：选择LPD外设
输入参数2	LPD _FLAG：LPD的标志位，定义在LPD_FLAG_TypeDef中
返回值	无

LPD_FLAG

选择LPD的标志位，参阅 Table 530获得这个参数的所有成员。

使用示例:

```
/* 清除LPD中断请求标志 */
LPD_ ClearFlag (LPD, LPD_Flag_LPDIF);
```

综合使用示例

```
void LPD_Example_Function(void)
{
   RCC_APB2Cmd(ENABLE);  //LPD外设挂载在APB2时钟上
   GPIO_InitTypeDef GPIO_InitStruct; //定义GPIO初始化结构体变量
   /* PC5设置为推挽输出 */
   GPIO_InitStruct.GPIO_Pin = GPIO_Pin_5;
   GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT_PP;
   GPIO_Init(GPIOB, &GPIO_InitStruct);
   /* PC6设置为推挽输出 */
   GPIO_InitStruct.GPIO_Pin = GPIO_Pin_6;
   GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT_PP;
   GPIO_Init(GPIOB, &GPIO_InitStruct);

    GPIO_ResetBits( GPIOB, GPIO_Pin_6);

   LPD_VLPDConfig(LPD,VLPD_3_05V_typ);
    NVIC_EnableIRQ(LPD_IRQn);
   LPD_ITConfig(LPD,LPD_IT_INTEN,ENABLE);
   LPD_Cmd(LPD,ENABLE);

  while(1)
  {
    if(LPD_GetFlagStatus(LPD,LPD_Flag_LPDOF))
```

```
    {
      GPIO_ResetBits( GPIOB, GPIO_Pin_5);
    }
    else
    {
      GPIO_SetBits( GPIOB, GPIO_Pin_5);
    }
  }
}

void LPD_Example_LPDIRQHandler(void)
{
  GPIO_SetBits( GPIOB, GPIO_Pin_6 );
  LPD_ClearFlag(LPD,LPD_Flag_LPDIF);
}
```

35 RTC 固件库函数

赛元SC32L14xx、SC32R807系列内部集成一个实时时钟模块（简称RTC），RTC是一个独立的BCD定时器/计数器，可提供具有可编程闹钟功能的时钟/日历。

RTC 寄存器结构

赛元SC32L14xx、SC32R807系列RTC寄存器结构，RTC_TypeDef，在文件“sc32L14xx.h”和“sc32r807.h”中定义如下：

```
typedef struct
{
  __IO uint32_t RTC_CON;
  __IO uint32_t RTC_CFG0;
  __IO uint32_t RTC_CFG1;
  __IO uint32_t RTC_ALARM;
  __IO uint32_t RTC_STS;
} RTC_TypeDef;
```

RTC寄存器表格

Table 695

寄存器	描述
RTC_CON	RTC控制寄存器
RTC_CFG0	RTC配置寄存器0
RTC_CFG1	RTC配置寄存器1
RTC_ALARM	RTC闹钟设置寄存器
RTC_STS	RTC标志寄存器

RTC 固件库函数列表

Table 696

函数名	描述
RTC_DeInit	将RTCx寄存器设置为缺省值
RTC_Init	根据RTC_InitStruct 中指定的参数初始化外设RTCx寄存器
RTC_StructInit	把RTC_InitStruct 中的每一个参数按缺省值填入
RTC_SetDate	设置RTCx日期
RTC_SetTime	设置RTCx时间
RTC_SetAlarm	设置RTCx闹钟
RTC_GetDate	获取RTCx日期
RTC_GetTime	获取RTCx时间
RTC_ITConfig	使能或者失能指定的RTCx中断
RTC_SetCT	设置固定周期中断时间
RTC_GetFlagStatus	获取指定RTCx的标志位状态
RTC_ClearFlag	清除指定RTCx的标志位状态

RTC 固件库函数详解

RTC_DeInit

Table 697

函数名	RTC_DeInit
函数原型	void RTC_DeInit (RTC_TypeDef* RTCx)
功能描述	将RTCx寄存器设置为缺省值
输入参数	RTCx：选择RTC外设

返回值	无

使用示例:

```
/* RTC寄存器复位 */
RTC_DeInit(RTC);
```

RTC_Init

Table 698

函数名	RTC_Init
函数原型	void RTC_Init (RTC_TypeDef* RTCx, RTC_InitTypeDef* RTC_InitStruct)
功能描述	根据RTC_InitStruct 中指定的参数初始化外设RTCx寄存器
输入参数1	RTCx：选择RTC外设
输入参数2	RTC_InitStruct：指向结构RTC_InitTypeDef 的指针，包含了外设RTC的配置信息
返回值	无

RTC_InitTypeDef

RTC_InitTypeDef 定义于文件“sc32f1xxx_rtc.h”：

```
typedef struct
{
    uint32_t RTC_AMPM;
    uint32_t RTC_RTCSEL;
} RTC_InitTypeDef;
```

RTC_AMPM

RTC小时系统设置，参阅Table 699获得这个参数的所有成员。

Table 699

RTC_AMPM	描述
RTC_HourFormat12	12 小时系统
RTC_HourFormat24	24小时系统

RTC_RTCSEL

RTC时钟源选择，参阅Table 700获得这个参数的所有成员。

Table 700

RTC_RTCSEL	描述
RTC_ClockSource_NON	无时钟，即关闭RTC时钟源
RTC_ClockSource_LXT	选择LXT为RTC时钟源

使用示例:

```
/* 根据RTC_InitStruct 中指定的参数初始化RTC */
RTC_InitTypeDef RTC_InitStruct; //定义RTC初始化结构体变量
RTC_InitStruct.RTC_RTCSEL = RTC_ClockSource_LXT; //RTC选择时钟源LXT
RTC_InitStruct.RTC_AMPM = RTC_HourFormat24; //24小时制
RTC_Init(RTC,&RTC_InitStruct);
```

RTC_StructInit

Table 701

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

使用示例:

```
/* 把 RTC_InitStruct 中的每一个参数按缺省值填入 */
RTC_InitTypeDef RTC_InitStruct;                    //定义RTC初始化结构体
RTC_StructInit(&RTC_InitStruct);
```

RTC_SetDate

Table 702

函数名	RTC_SetDate
函数原型	void RTC_SetDate (RTC_TypeDef* RTCx, RTC_DateTypeDef* RTC_Date)
功能描述	设置RTCx日期
输入参数1	RTCx：选择RTC外设
输入参数2	RTC_Date：指向结构RTC_DateTypeDef的指针，包含了RTC日期设置的配置信息
返回值	无

RTC_DateTypeDef

RTC_DateTypeDef 定义于文件“sc32f1xxx_rtc.h”：

```
typedef struct
{
    uint32_t RTC_YEAR;
    uint32_t RTC_MONTH;
    uint32_t RTC_WEEK;
    uint32_t RTC_DAY;
} RTC_DateTypeDef;
```

RTC_YEAR

年计数值设置。

RTC_MONTH

月计数值设置。

RTC_WEEK

周计数值设置。

RTC_DAY

日计数值设置。

使用示例:

```
/* 根据RTC_DateInitStruct 中指定的参数初始化日期设置 */
RTC_DateTypeDef RTC_DateInitStruct;
RTC_DateInitStruct.RTC_YEAR = 25; //2025年
RTC_DateInitStruct.RTC_MONTH = 7; //7月
RTC_DateInitStruct.RTC_WEEK = 2;  //星期二
RTC_DateInitStruct.RTC_DAY = 1;   //1号
RTC_SetDate(RTC,&RTC_DateInitStruct);
```

RTC_SetTime

Table 703

函数名	RTC_SetTime
函数原型	void RTC_SetTime (RTC_TypeDef* RTCx, RTC_TimeTypeDef* RTC_Time)
功能描述	设置RTCx时间
输入参数1	RTCx：选择RTC外设
输入参数2	RTC_Time：指向结构RTC_TimeTypeDef的指针，包含了RTC时间设置的配置信息
返回值	无

RTC_TimeTypeDef

RTC_TimeTypeDef 定义于文件“sc32f1xxx_rtc.h”：

```
typedef struct
{
    uint32_t RTC_HOUR;
    uint32_t RTC_MIN;
    uint32_t RTC_SEC;
} RTC_TimeTypeDef;
```

RTC_HOUR

小时计数值设置。

RTC_MIN

分钟计数值设置。

RTC_SEC

秒计数值设置。

使用示例:

```
/* 根据RTC_TimeInitStruct 中指定的参数初始化时间设置 */
RTC_TimeTypeDef RTC_TimeInitStruct;
RTC_TimeInitStruct.RTC_HOUR = 9;  //9时
RTC_TimeInitStruct.RTC_MIN = 0;   //0分
RTC_TimeInitStruct.RTC_SEC = 0;   //0秒
RTC_SetTime(RTC,&RTC_TimeInitStruct);
```

RTC_SetAlarm

Table 704

函数名	RTC_SetAlarm
函数原型	void RTC_SetAlarm (RTC_TypeDef* RTCx, RTC_AlarmTypeDef* RTC_Alarm)
功能描述	设置RTCx闹钟
输入参数1	RTCx：选择RTC外设
输入参数2	RTC_Alarm：指向结构RTC_AlarmTypeDef的指针，包含了RTC闹钟设置的配置信息
返回值	无

RTC_AlarmTypeDef

RTC_AlarmTypeDef 定义于文件“sc32f1xxx_rtc.h”：

```
typedef struct
{
    uint32_t RTC_Alarm_WW;
    uint32_t RTC_Alarm_WH;
    uint32_t RTC_Alarm_WM;
} RTC_AlarmTypeDef;
```

RTC_Alarm_WW

闹钟星期设置。

RTC_Alarm_WH

闹钟小时设置。

RTC_Alarm_WM

闹钟分钟设置。

使用示例:

```
/* 根据RTC_AlarmInitStruct 中指定的参数初始化闹钟设置 */
RTC_AlarmTypeDef RTC_AlarmInitStruct;  //设置闹钟
RTC_AlarmInitStruct.RTC_Alarm_WW = RTC_Alarm_Wednesday | RTC_Alarm_Thursday; //星期三、星期四
RTC_AlarmInitStruct.RTC_Alarm_WH = 12;  //12时
RTC_AlarmInitStruct.RTC_Alarm_WM = 0;  //0分
RTC_SetAlarm(RTC,&RTC_AlarmInitStruct);
```

RTC_GetDate

Table 705

函数名	RTC_GetDate
函数原型	void RTC_GetDate (RTC_TypeDef* RTCx, RTC_DateTypeDef* RTC_Date)
功能描述	获取RTCx日期
输入参数1	RTCx：选择RTC外设
输入参数2	RTC_Date：指向结构RTC_DateTypeDef的指针，包含了RTC日期信息
返回值	无

使用示例:

```
/* 读取日期到结构体RTC_DateTypeDef */
RTC_DateTypeDef RTC_Date_Read;
RTC_GetDate(RTC,&RTC_Date_Read);
```

RTC_GetTime

Table 706

函数名	RTC_GetTime
函数原型	void RTC_GetTime (RTC_TypeDef* RTCx , RTC_TimeTypeDef* RTC_Time)
功能描述	获取RTCx时间
输入参数1	RTCx：选择RTC外设
输入参数2	RTC_Time：指向结构RTC_TimeTypeDef的指针，包含了RTC时间信息
返回值	无

使用示例:

```
/* 读取日期到结构体RTC_TimeTypeDef */
RTC_TimeTypeDef RTC_Time_Read;
RTC_GetTime(RTC,&RTC_Time_Read);
```

RTC_ITConfig

Table 707

函数名	RTC_ITConfig
函数原型	void RTC_ITConfig (RTC_TypeDef* RTCx, uint16_t RTC_IT, FunctionalState NewState)
功能描述	使能或者失能指定的RTCx中断
输入参数1	RTCx：选择RTC外设
输入参数2	RTC_IT：待使能或者失能的RTC中断源
输入参数3	NewState：RTC中断功能开启/关闭，可取值ENABLE或DISABLE
返回值	无

RTC_IT

RTC中断选择，参阅Table 708获得这个参数的所有成员。

Table 708

RTC_IT	描述
RTC_IT_INTEN	中断请求 CPU的使能控制位
RTC_IT_WALIE	闹钟中断使能控制位

使用示例:

```
/*打开中断请求 CPU的使能和闹钟中断使能*/
RTC_ITConfig(RTC, RTC_IT_INTEN| RTC_IT_WALIE, ENABLE);
```

RTC_SetCT

Table 709

函数名	RTC_SetCT
函数原型	void RTC_SetCT (RTC_TypeDef* RTCx, RTC_CT_TypeDef RTC_CT)
功能描述	设置固定周期中断时间
输入参数1	RTCx：选择RTC外设
输入参数2	RTC_CT：固定周期中断时间设置
返回值	无

RTC_CT

固定周期中断时间选择，参阅Table 710获得这个参数的所有成员。

Table 710

RTC_CT	描述
RTC_CT_None	不使用固定周期中断功能
RTC_CT_0_5Second	0.5 秒一次（与秒累加同步）
RTC_CT_1Second	1 秒一次（与秒累加同时）
RTC_CT_1Minute	1 分钟一次（每分钟的 00 秒）
RTC_CT_1Hour	1 小时一次（每小时的 00 分 00 秒）

RTC_CT_1Day	1 日一次（24小时制时为每日的 00 点 00 分 00 秒，12小时制时为每日的 AM12 时 00 分 00 秒）
RTC_CT_1Month	1 个月一次（24小时制时为每月的 1 日上午 00 点 00 分 00 秒，12小时制时为每月的 1 日上午12 时 00 分 00 秒）

使用示例:

```
/*1s产生一次中断*/
RTC_SetCT(RTC,RTC_CT_1Second);
```

RTC_GetFlagStatus

Table 711

函数名	RTC_GetFlagStatus
函数原型	FlagStatus RTC_GetFlagStatus (RTC_TypeDef* RTCx, RTC_FLAG_TypeDef RTC_FLAG)
功能描述	获取指定RTCx的标志位状态
输入参数1	RTCx：选择RTC外设
输入参数2	RTC_FLAG：选择RTC的标志位
返回值	RTC_FLAG 的新状态

RTC_FLAG

选择RTC的标志位，参阅Table 712获得这个参数的所有成员。

Table 712

RTC_FLAG	描述
RTC_Flag_RTCCTIF	固定周期中断标志位
RTC_Flag_WALIF	闹钟检测标志位

使用示例:

```
/* 检查固定周期中断标志位设置与否 */
FlagStatus  RTC_Status;
RTC_Status  = RTC_GetFlagStatus (RTC, RTC_Flag_RTCCTIF);
```

RTC_ClearFlag

Table 713

函数名	RTC_ClearFlag
函数原型	void RTC_ClearFlag(RTC_TypeDef* RTCx,uint8_t RTC_FLAG)
功能描述	清除指定RTCx的标志位状态
输入参数1	RTCx：选择RTC外设
输入参数2	RTC_FLAG：RTC的标志位，定义在RTC_FLAG_TypeDef中
返回值	无

RTC_FLAG

选择RTC的标志位，参阅 Table 713获得这个参数的所有成员。

使用示例:

```
/* 清除固定周期中断标志位 */
RTC_ClearFlag (RTC, RTC_Flag_RTCCTIF);
```

综合使用示例

```
RTC_DateTypeDef RTC_Date_Read;
RTC_TimeTypeDef RTC_Time_Read;

void RTC_Example_Function(void)
{
  RCC_APB2Cmd(ENABLE);  //RTC外设挂载在APB2时钟上
  RTC_InitTypeDef RTC_InitStruct; //定义RTC初始化结构体变量
  /* 结构体成员初始化 */
```

```
  RTC_StructInit(&RTC_InitStruct);

  /* 使能RTC时钟 */
  //使能LXT
  RCC_Unlock(0xFF);
  RCC_LXTCmd(ENABLE);
  RTC_InitStruct.RTC_RTCSEL = RTC_ClockSource_LXT; //RTC选择时钟源LXT
  RTC_InitStruct.RTC_AMPM = RTC_HourFormat24;    //24小时制
  RTC_Init(RTC,&RTC_InitStruct);

  RTC_DateTypeDef RTC_DateInitStruct;
  RTC_DateInitStruct.RTC_YEAR = 25; //2025年
  RTC_DateInitStruct.RTC_MONTH = 7; //7月
  RTC_DateInitStruct.RTC_WEEK = 2;  //星期二
  RTC_DateInitStruct.RTC_DAY = 1;   //1号
  RTC_SetDate(RTC,&RTC_DateInitStruct);

  RTC_TimeTypeDef RTC_TimeInitStruct;
  RTC_TimeInitStruct.RTC_HOUR = 9;  //9时
  RTC_TimeInitStruct.RTC_MIN = 0;   //0分
  RTC_TimeInitStruct.RTC_SEC = 0;   //0秒
  RTC_SetTime(RTC,&RTC_TimeInitStruct);

  RTC_AlarmTypeDef RTC_AlarmInitStruct;  //设置闹钟
  RTC_AlarmInitStruct.RTC_Alarm_WW = RTC_Alarm_Wednesday | RTC_Alarm_Thursday; //星期三、星期四
  RTC_AlarmInitStruct.RTC_Alarm_WH = 12;  //12时
  RTC_AlarmInitStruct.RTC_Alarm_WM = 0;  //0分
  RTC_SetAlarm(RTC,&RTC_AlarmInitStruct);

  NVIC_EnableIRQ(RTC_IRQn);
  RTC_ITConfig(RTC,RTC_IT_WALIE,ENABLE); //闹钟中断使能
  RTC_ITConfig(RTC,RTC_IT_INTEN,ENABLE); //总中断使能
  RTC_SetCT(RTC,RTC_CT_1Second); //1s产生一次中断

   while(1)
   {
     printf("进入STOP模式！\r\n");
     PWR_EnterSTOPMode(PWR_STOPEntry_WFE);

     //读取日期和时间
     RTC_GetDate(RTC,&RTC_Date_Read);
     RTC_GetTime(RTC,&RTC_Time_Read);
     printf("\r\n Text! \r\n");
     printf("%.2d-%.2d-%.2d-%.2d\r\n", 2000 +
     RTC_Date_Read.RTC_YEAR,RTC_Date_Read.RTC_MONTH,RTC_Date_Read.RTC_DAY,RTC_Date_Read.R
     TC_WEEK);
     printf("%.2d:%.2d:%.2d\r\n",
     RTC_Time_Read.RTC_HOUR,RTC_Time_Read.RTC_MIN,RTC_Time_Read.RTC_SEC);
  }
}

void RTC_Example_RTCHandler(void)
{
  if(RTC_GetFlagStatus(RTC,RTC_Flag_WALIF) == SET)
  {
     printf("\r\n RTC INTERRUPT WAIF Text! \r\n");
     RTC_ClearFlag(RTC,RTC_Flag_WALIF);//清WALIF标志
  }
  if(RTC_GetFlagStatus(RTC,RTC_Flag_RTCCTIF) == SET)
```

```
  {
    RTC_ClearFlag(RTC,RTC_Flag_RTCCTIF); //清RTCCTIF标志
    //读取日期和时间
    RTC_GetDate(RTC,&RTC_Date_Read);
    RTC_GetTime(RTC,&RTC_Time_Read);
    printf("%.2d-%.2d-%.2d-%.2d\r\n", 2000 +
    RTC_Date_Read.RTC_YEAR,RTC_Date_Read.RTC_MONTH,RTC_Date_Read.RTC_DAY,RTC_Date_Read.R
    TC_WEEK);
    printf("%.2d:%.2d:%.2d\r\n",
    RTC_Time_Read.RTC_HOUR,RTC_Time_Read.RTC_MIN,RTC_Time_Read.RTC_SEC);
    printf("\r\n RTC INTERRUPT RTCCTIF Text! \r\n");
  }
}
```

36 SmartCard 固件库函数

赛元SC32L14xx系列内置的智能卡接口控制器(Smart Card controller)基于ISO/IEC 7816-3标准，可通过两线进行八位数据的串行通信，可软件控制GPIO引脚作为智能卡复位功能和智能卡插入检测功能。

SmartCard 寄存器结构

赛元SC32L14xx系列SmartCard寄存器结构，Smart_Card_TypeDef，在文件“sc32L14xx.h”中定义如下：

```
typedef struct
{
  __IO uint32_t SC0_CON;
  __IO uint32_t SC0_STS;
  __IO uint32_t SC0_BAUD;
  __IO uint32_t SC0_DATA;
  __IO uint32_t SC0_IDE;
} Smart_Card_TypeDef;
```

SmartCard寄存器表格

Table 714

寄存器	描述
SC0_CON	SmartCard控制寄存器
SC0_STS	SmartCard标志寄存器
SC0_BAUD	SmartCard波特率配置寄存器
SC0_DATA	SmartCard数据寄存器
SC0_IDE	SmartCard中断使能寄存器

SmartCard 固件库函数列表

Table 715

函数名	描述
Smart_Card_DeInit	将Smart_Card寄存器设置为缺省值
Smart_Card_Init	根据SC_InitStruct中指定的参数初始化外设Smart_Card寄存器
Smart_Card_StructInit	把SC_InitStruct中的每一个参数按缺省值填入
Smart_Card_Cmd	使能或者失能Smart_Card外设
Smart_Card_CKEN_Cmd	Smart_Card时钟输出使能
Smart_Card_TREN_Cmd	Smart_Card发送接收使能
Smart_Card_BaudConfig	Smart_Card波特率配置
Smart_Card_BaudStructInit	把SC_BaudInitStruct中的每一个参数按缺省值填入
Smart_Card_SendData	通过外设Smart_Card发送一个数据
Smart_Card_ReceiveData	返回通过Smart_Card最近接收的数据
Smart_Card_PinRemapConfig	配置Smart_Card 管脚的重映射
Smart_Card_ITConfig	使能或者失能指定的Smart_Card中断
Smart_Card_GetFlagStatus	获取指定Smart_Card的标志位状态
Smart_Card_ClearFlag	清除指定Smart_Card的标志位状态

SmartCard 固件库函数详解

Smart_Card_DeInit

Table 716

函数名	Smart_Card_DeInit
函数原型	void Smart_Card_DeInit (Smart_Card_TypeDef* SCx)
功能描述	将Smart_Card寄存器设置为缺省值

输入参数	SCx：选择Smart_Card外设
返回值	无

使用示例:

```
/* Smart_Card寄存器复位 */
Smart_Card_DeInit(Smart_Card);
```

Smart_Card_Init

Table 717

函数名	Smart_Card_Init
函数原型	void Smart_Card_Init (Smart_Card_TypeDef* SCx, SmartCard_InitTypeDef* SC_InitStruct)
功能描述	根据SC_InitStruct中指定的参数初始化外设Smart_Card寄存器
输入参数1	SCx：选择Smart_Card外设
输入参数2	SC_InitStruct：指向结构SmartCard_InitTypeDef 的指针，包含了外设Smart_Card的配置信息
返回值	无

SmartCard_InitTypeDef

SmartCard_InitTypeDef 定义于文件“sc32f1xxx_smartcard.h”：

```
typedef struct
{
    uint32_t SC_Parity;
    uint32_t SC_TransReadError;
    uint32_t LPC_ SC_EncodeMethod;
    uint32_t LPC_ SC_ErrorETU;
    uint32_t LPC_ SC_TransReceEn;
    uint32_t LPC_ SC_ClockOutEn;
} SmartCard_InitTypeDef;
```

SC_Parity

奇偶校验选择，参阅Table 718获得这个参数的所有成员。

Table 718

SC_Parity	描述
SC_PCS_NoneParity	不校验
SC_PCS_EvenParity	偶校验

SC_TransReadError

数据发送接收校验出错重发控制，参阅Table 719获得这个参数的所有成员。

Table 719

SC_TransReadError	描述
SC_TRER_SetITFlag	校验出错，直接置位中断标志位
SC_TRER_LowLevelACK	接收数据校验出错会发送一个低电平应答，发送数据后接收到低电平应答会重发该数据

SC_EncodeMethod

编码方式控制，参阅Table 720获得这个参数的所有成员。

Table 720

SC_EncodeMethod	描述
SC_CONS_Positive	正向约定，LSB传输，正逻辑电平
SC_CONS_Negative	反向约定，MSB传输，反逻辑电平

SC_ErrorETU

Stop及Error Signal长度选择，参阅Table 721获得这个参数的所有成员。

Table 721

SC_ErrorETU	描述
SC_ERS_2ETU	Stop及Error Signal长度均为2个ETU
SC_ERS_2ETU_	Stop及Error Signal长度均为2个ETU

SC_ERS_1_5ETU	Stop及Error Signal长度均为1.5个ETU
SC_ERS_1ETU	Stop及Error Signal长度均为1个ETU

SC_TransReceEn

发送接收使能，参阅Table 722获得这个参数的所有成员。

Table 722

SC_TransReceEn	描述
SC_TREN_RXEN	接收使能，发送禁止
SC_TREN_TXEN	发送使能，接收禁止, 发送完一帧数据之后接口会释放SC_DAT，开始检测停止位上的Error Signal

SC_ClockOutEn

时钟输出使能，参阅Table 723获得这个参数的所有成员。

Table 723

SC_ClockOutEn	描述
SC_CKEN_Disable	禁止时钟输出
SC_CKEN_Enable	打开时钟输出

使用示例:

```
/* 根据Smart_Card_InitStruct中指定的参数初始化Smart_Card */
SmartCard_InitTypeDef Smart_Card_InitStruct;
Smart_Card_InitStruct.SC_Parity = SC_PCS_EvenParity;              //选择偶校验
Smart_Card_InitStruct.SC_TransReadError = SC_TRER_LowLevelACK;    //接收数据校验出错发送一个低电平应答，发送数据后接收到低电平应答会重发该数据
Smart_Card_InitStruct.SC_EncodeMethod = SC_CONS_Positive;         //正向约定，LSB传输，正逻辑电平
Smart_Card_InitStruct.SC_ErrorETU = SC_ERS_2ETU_;                 //Stop及Error Signal长度均为2个ETU
Smart_Card_InitStruct.SC_TransReceEn = SC_TREN_RXEN;              //接收使能，发送禁止
Smart_Card_InitStruct.SC_ClockOutEn = SC_CKEN_Disable;            //禁止时钟输出
Smart_Card_Init(Smart_Card,&Smart_Card_InitStruct);
```

Smart_Card_StructInit

Table 724

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

使用示例:

```
/* 把SC_InitStruct中的每一个参数按缺省值填入 */
SmartCard_InitTypeDef Smart_Card_InitStruct;      //定义Smart_Card 初始化结构体
Smart_Card_StructInit(&Smart_Card_InitStruct);
```

Smart_Card_Cmd

Table 725

函数名	Smart_Card_Cmd
函数原型	void Smart_Card_Cmd (Smart_Card_TypeDef* SCx, FunctionalState NewState)
功能描述	使能或者失能Smart_Card外设
输入参数1	SCx：选择Smart_Card外设
输入参数2	NewState：Smart_Card 功能开启/关闭，可取值ENABLE或DISABLE
返回值	无

使用示例:

```
/* LPC使能 */
Smart_Card_Cmd(Smart_Card,ENABLE);
```

Smart_Card_CKEN_Cmd

Table 726

函数名	Smart_Card_CKEN_Cmd
函数原型	void Smart_Card_CKEN_Cmd (Smart_Card_TypeDef* SCx, FunctionalState NewState)
功能描述	Smart_Card时钟输出使能
输入参数1	SCx：选择Smart_Card 外设
输入参数2	NewState：Smart_Card 功能开启/关闭，可取值ENABLE或DISABLE
返回值	无

使用示例:

```
/* 关闭Smart_Card 时钟输出 */
Smart_Card_CKEN_Cmd(Smart_Card,DISABLE);
```

Smart_Card_TREN_Cmd

Table 727

函数名	Smart_Card_TREN_Cmd
函数原型	void Smart_Card_TREN_Cmd (Smart_Card_TypeDef* SCx, SmartCard_TREN_TypeDef NewState)
功能描述	Smart_Card发送接收使能
输入参数1	SCx：选择Smart_Card 外设
输入参数2	NewState：发送接收使能
返回值	无

NewState

发送接收使能，参阅Table 722获得这个参数的所有成员。

使用示例:

```
/* Smart_Card接收使能  */
Smart_Card_TREN_Cmd(Smart_Card,SC_TREN_RXEN);
```

Smart_Card_BaudConfig

Table 728

函数名	Smart_Card_BaudConfig
函数原型	void Smart_Card_BaudConfig (Smart_Card_TypeDef* SCx, SmartCard_BaudInitTypeDef* SC_BaudInitStruct)
功能描述	Smart_Card波特率配置
输入参数1	SCx：选择Smart_Card 外设
输入参数2	SC_BaudInitStruct：指向结构SmartCard_BaudInitTypeDef 的指针，包含了外设Smart_Card的波特率配置信息
返回值	无

SmartCard_BaudInitTypeDef

SmartCard_BaudInitTypeDef 定义于文件“sc32f1xxx_smartcard.h”：

```
typedef struct
{
    uint32_t SC_ExtraGuardTime;
    uint32_t SC_ClockCycle;
    uint32_t SC_ETUClockCycle;
} SmartCard_BaudInitTypeDef;
```

SC_ExtraGuardTime

Smart_Card 扩展保护时间设置，拓展保护时间为EGT [7:0]个ETU时钟周期，即：SC通信过程中的实际扩展保护时间T_{EGT} = T_{ETU}* EGT [7:0]。设置值范围：0x00~0xFF

SC_ClockCycle

Smart_Card 时钟周期设置，时钟周期T_{SC} = (SCCK[4:0]+1)*2 / f_{SYS}。设置值范围：0x00~0x1F。

SC_ETUClockCycle

ETU时钟周期设置，ETU时钟周期为(ETUCK [11:0] + 1)个SC时钟周期，即：ETU时钟周期

$T_{ETU} = T_{SC} * (ETUCK [11:0] + 1)$

ETUCK[11:0] 必须大于 0x004。

使用示例:

```
/*  根据Smart_Card_BaudInitStruct中指定的参数初始化Smart_Card波特率  */
SmartCard_BaudInitTypeDef Smart_Card_BaudInitStruct;
Smart_Card_BaudInitStruct.SC_ExtraGuardTime = 0x5;   //扩展保护时间
Smart_Card_BaudInitStruct.SC_ClockCycle = 0x02;       //CLK时钟周期=((SC_ClockCycle+1)*2)/ 主频24M
Smart_Card_BaudInitStruct.SC_ETUClockCycle = 0x174;  //ETU时钟周期 = CLK时钟周期
*(SC_ETUClockCycle+1)
Smart_Card_BaudConfig(Smart_Card,&Smart_Card_BaudInitStruct);
```

Smart_Card_BaudStructInit

Table 729

函数名	Smart_Card_BaudStructInit
函数原型	void Smart_Card_BaudStructInit (SmartCard_BaudInitTypeDef* SC_BaudInitStruct)
功能描述	把SC_BaudInitStruct中的每一个参数按缺省值填入
输入参数	SC_BaudInitStruct：指向结构SmartCard_BaudInitTypeDef 的指针，包含了外设Smart_Card的波特率配置信息
返回值	无

使用示例:

```
/* 把SC_BaudInitStruct中的每一个参数按缺省值填入 */
SmartCard_BaudInitTypeDef Smart_Card_BaudInitStruct;   //定义Smart_Card 初始化结构体
Smart_Card_BaudStructInit(&Smart_Card_BaudInitStruct);
```

Smart_Card_SendData

Table 730

函数名	Smart_Card_SendData
函数原型	void Smart_Card_SendData (Smart_Card_TypeDef* SCx, uint8_t Data)
功能描述	通过外设Smart_Card发送一个数据
输入参数1	SCx：选择Smart_Card 外设
输入参数2	Data：Smart_Card发送的数据，8位
返回值	无

使用示例:

```
/* Smart_Card发送数据Data1 */
uint8_t Data1 = 0x5A;
Smart_Card_SendData (Smart_Card, Data1);
```

Smart_Card_ReceiveData

Table 731

函数名	Smart_Card_ReceiveData
函数原型	uint8_t Smart_Card_ReceiveData (Smart_Card_TypeDef* SCx)
功能描述	返回通过Smart_Card最近接收的数据
输入参数	SCx：选择Smart_Card 外设
返回值	Smart_Card最近接收的数据

使用示例:

```
/* 把Smart_Card接收的数据赋值给Data1 */
Uint8_t Data1
Data1 = Smart_Card_ReceiveData (Smart_Card);
```

Smart_Card_PinRemapConfig

Table 732

函数名	Smart_Card_PinRemapConfig
函数原型	void Smart_Card_PinRemapConfig (Smart_Card_TypeDef* SCx, SmartCard_PinRemap_TypeDef SC_Remap)
功能描述	配置Smart_Card 管脚的重映射
输入参数1	SCx：选择Smart_Card 外设
输入参数2	SC_Remap：待使能或者失能的LPC中断源
返回值	无

SC_Remap

Smart_Card信号口映射控制，参阅Table 733获得这个参数的所有成员。

Table 733

SC_Remap	描述
SC_PinRemap_Default	默认Smart_Card管脚映射
SC_PinRemap_A	Smart_Card映射管脚组A

使用示例:

```
/* 把Smart_Card管脚设置为重映射组A */
Smart_Card_PinRemapConfig (Smart_Card, SC_PinRemap_A);
```

Smart_Card_ITConfig

Table 734

函数名	Smart_Card_ITConfig
函数原型	void Smart_Card_ITConfig (Smart_Card_TypeDef* SCx, uint16_t SC_IT, FunctionalState NewState)
功能描述	使能或者失能指定的Smart_Card中断
输入参数1	SCx：选择Smart_Card 外设
输入参数2	SC_IT：待使能或者失能的Smart_Card中断源
输入参数3	NewState：Smart_Card中断功能开启/关闭，可取值ENABLE或DISABLE
返回值	无

SC_IT

Smart_Card中断选择，参阅Table 735获得这个参数的所有成员。

Table 735

SC_IT	描述
SC_IT_INTEN	中断请求CPU 的屏蔽位
SC_IT_RXIE	接收中断使能控制位
SC_IT_TXIE	发送中断使能控制位
SC_IT_ERRIE	通讯异常中断使能位

使用示例:

```
/*打开中断请求 CPU的使能和接收中断使能*/
Smart_Card_ITConfig(Smart_Card,SC_IT_INTEN | SC_IT_RXIE,ENABLE);
```

Smart_Card_GetFlagStatus

Table 736

函数名	Smart_Card_GetFlagStatus
函数原型	FlagStatus Smart_Card_GetFlagStatus (Smart_Card_TypeDef* SCx, uint32_t SC_FLAG)
功能描述	获取指定Smart_Card的标志位状态
输入参数1	SCx：选择Smart_Card 外设
输入参数2	SC_FLAG：选择Smart_Card的标志位
返回值	SC_FLAG 的新状态

SC_FLAG

选择mart_Card的标志位，参阅 Table 530获得这个参数的所有成员。

Table 737

SC_FLAG	描述
SC_Flag_RC	接收完成标志位
SC_Flag_TC	发送完成标志位
SC_Flag_ROVF	接收溢出标志位
SC_Flag_FER	接收帧错误标志位
SC_Flag_WTER	等待超时标志位
SC_Flag_RPER	接收数据奇偶校验错误标志位
SC_Flag_TPER	发送数据奇偶校验错误标志位
SC_Flag_RBUSY	数据接收忙状态位
SC_Flag_TBUSY	数据发送忙状态位
SC_Flag_WTRT	等待数据重发状态位

使用示例:

```
/* 检查发送完成标志位设置与否 */
FlagStatus  Smart_Card_Status;
Smart_Card_Status  = Smart_Card_GetFlagStatus (Smart_Card, SC_Flag_TC);
```

Smart_Card_ClearFlag

Table 738

函数名	Smart_Card_ClearFlag
函数原型	void Smart_Card_ClearFlag (Smart_Card_TypeDef* SCx, uint32_t SC_FLAG)
功能描述	清除指定Smart_Card的标志位状态
输入参数1	SCx：选择Smart_Card 外设
输入参数2	SC_FLAG：Smart_Card的标志位，定义在SmartCard_FLAG_TypeDef中
返回值	无

SC_FLAG

选择Smart_Card的标志位，参阅Table 737获得这个参数的所有成员。

使用示例:

```
/* 清除发送完成标志位 */
Smart_Card_ClearFlag (Smart_Card, SC_Flag_TC);
```

综合使用示例

```
uint8_t SC_RCFlag = 0;
uint8_t SC_TCFlag = 0;
uint8_t SC_ROVFFlag = 0;
uint8_t SC_FERFlag = 0;
uint8_t SC_WTERFlag = 0;
uint8_t SC_RPERFlag = 0;
uint8_t SC_TPERFlag = 0;
uint8_t SC_RBUSYFlag = 0;
uint8_t SC_TBUSYFlag = 0;
uint8_t SC_WTRTFlag = 0;

uint8_t Receive_ATR_Info[16];//ATR信息
uint8_t Receive_HOTATR_Info[16];//ATR信息
uint8_t Receive_PPS_Info[16];//ATR信息
uint8_t Receive_commend_Info[16];//ATR信息
const uint8_t ICCID[2] = {0x2F, 0xE2};

uint8_t Timer2_Flag = 0;
uint32_t Timer2_cnt = 0;

void Smart_Card_Activation_ColdReduction(void);
```

```
void Smart_Card_Activation_HOTReduction(void);
void SendCommend(void);
void SendPPS(void);
void Smart_Card_STOP(void);
void Delay_X10us(uint32_t count);
void Delay_X1ms(uint32_t count1);
void Delay_X1s(uint32_t count2);

void Smart_Card_Communication_Function(void)
{
  volatile uint16_t SC_RxData[5];   //发送5个8位自定义数据（若要修改数据个数需修改相应数组大小和发送/接收for循环大小）

    RCC_APB0Config(RCC_HCLK_Div1);
    RCC_APB0Cmd(ENABLE);
    RCC_APB0PeriphClockCmd(RCC_APB0Periph_TIM2,ENABLE);
    TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStruct;
    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 - 240;     //设置TIM的定时器频率为100kHz@APB0Clock=24M
    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_TIMBaseInit(TIM2, &TIM_TimeBaseInitStruct);
    TIM_ITConfig(TIM2, TIM_IT_TI | TIM_IT_INTEN, ENABLE);
    NVIC_EnableIRQ(TIMER2_IRQn);
    TIM_Cmd(TIM2, DISABLE);

    GPIO_InitTypeDef GPIO_InitStruct; //定义GPIO初始化结构体变量
     /* PC5设置为推挽输出 */
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_5;
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT_PP;
    GPIO_Init(GPIOB, &GPIO_InitStruct);  //SC_VCC
    /* PC6设置为推挽输出 */
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_6;
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT_PP;
    GPIO_Init(GPIOB, &GPIO_InitStruct);  //SC_RST
    /* PC11设置为推挽输出 */
     GPIO_InitStruct.GPIO_Pin = GPIO_Pin_11;
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT_PP;
    GPIO_Init(GPIOB, &GPIO_InitStruct);  //SC_VPP

    GPIO_ResetBits( GPIOB, GPIO_Pin_5);
    GPIO_ResetBits( GPIOB, GPIO_Pin_6);
    GPIO_ResetBits( GPIOB, GPIO_Pin_11);

    /* 使能SC时钟 */
    RCC_APB1Cmd(ENABLE);                                //Smart_Card挂载在APB1时钟上
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_SC,ENABLE);       //开启Smart_Card时钟
    SmartCard_InitTypeDef Smart_Card_InitStruct;                    //定义Smart_Card初始化结构体变量
    Smart_Card_StructInit(&Smart_Card_InitStruct);
    Smart_Card_InitStruct.SC_Parity = SC_PCS_EvenParity;                //选择偶校验
    Smart_Card_InitStruct.SC_TransReadError = SC_TRER_LowLevelACK; //接收数据校验出错发送一个低电平应答，发送数据后接收到低电平应答会重发该数据
    Smart_Card_InitStruct.SC_EncodeMethod = SC_CONS_Positive;               //正向约定，LSB传输，正逻辑电平
```

```
	Smart_Card_InitStruct.SC_ErrorETU = SC_ERS_2ETU_;			//Stop及Error Signal长度均为2个
ETU
	Smart_Card_InitStruct.SC_TransReceEn = SC_TREN_RXEN;		//接收使能，发送禁止
	Smart_Card_InitStruct.SC_ClockOutEn = SC_CKEN_Disable;		//禁止时钟输出
	Smart_Card_Init(Smart_Card,&Smart_Card_InitStruct);

	SmartCard_BaudInitTypeDef Smart_Card_BaudInitStruct;		//定义Smart_Card初始化结构体变
量
	Smart_Card_BaudStructInit(&Smart_Card_BaudInitStruct);
	Smart_Card_BaudInitStruct.SC_ExtraGuardTime = 0x5;	//扩展保护时间
	Smart_Card_BaudInitStruct.SC_ClockCycle = 0x02;		//CLK时钟周期=((SC_ClockCycle+1)*2)/ 主频24M
	Smart_Card_BaudInitStruct.SC_ETUClockCycle = 0x174;	//ETU时钟周期 = CLK时钟周期
*(SC_ETUClockCycle+1)
	Smart_Card_BaudConfig(Smart_Card,&Smart_Card_BaudInitStruct);

	Smart_Card_PinRemapConfig(Smart_Card,SC_PinRemap_Default);

	/* 使能串口接收中断 */
	Smart_Card_ITConfig(Smart_Card,SC_IT_INTEN | SC_IT_RXIE | SC_IT_TXIE | SC_IT_ERRIE,ENABLE);
	NVIC_EnableIRQ(UART1_3_5_7816_IRQn);

	Smart_Card_Cmd(Smart_Card,ENABLE);

	Smart_Card_Activation_ColdReduction();
	Delay_X10us(50);
	Smart_Card_Activation_HOTReduction();
	Delay_X10us(50);
	SendPPS();
	Delay_X10us(50);
	SendCommend();
	Delay_X10us(50);
	Smart_Card_STOP();

	while(1)
	{

	}
}

void Smart_Card_Activation_ColdReduction(void)//冷复位后读取ATR
{
	uint32_t i;
	GPIO_ResetBits( GPIOB, GPIO_Pin_5);
	GPIO_ResetBits( GPIOB, GPIO_Pin_6);
	GPIO_ResetBits( GPIOB, GPIO_Pin_11);
	Smart_Card_CKEN_Cmd(Smart_Card,DISABLE); //关闭时钟
	Smart_Card_Cmd(Smart_Card,DISABLE);			//关闭模块
	Delay_X10us(10);

	//T1
	GPIO_SetBits( GPIOB, GPIO_Pin_5);
	GPIO_SetBits( GPIOB, GPIO_Pin_11);
	Smart_Card_Cmd(Smart_Card,ENABLE);			//打开模块//
	Smart_Card_TREN_Cmd(Smart_Card,SC_TREN_RXEN);		//SC接收使能位
	Delay_X10us(1);

	//T2
	Smart_Card_CKEN_Cmd(Smart_Card,ENABLE);			//打开时钟
	Delay_X10us(6);
```

```
        //T3
        GPIO_SetBits( GPIOB, GPIO_Pin_6);
        for(i=0; i<13; i++) //ATR为15byte
 {
   while(SC_RBUSYFlag==1){}
   while(!(SC_RCFlag==1)){}
   SC_RCFlag=0;
   Receive_ATR_Info[i] = Smart_Card_ReceiveData(Smart_Card);
 }
        for(i=0; i<13; i++)
        {
                printf("ATR的值为%x\r\n",Receive_ATR_Info[i]);
        }
}

void Smart_Card_Activation_HOTReduction(void)
{
        unsigned char i;
        //T4
        GPIO_ResetBits( GPIOB, GPIO_Pin_6);  //拉低SC_RST，使Smart_Card复位
        Delay_X10us(10);
        //T5
        GPIO_SetBits( GPIOB, GPIO_Pin_6);
        Delay_X10us(6);
        for(i=0; i<13; i++) //ATR为15byte
        {
                while(SC_RBUSYFlag==1){};
                while(!SC_RCFlag){};
                SC_RCFlag=0;
                Receive_HOTATR_Info[i] = Smart_Card_ReceiveData(Smart_Card);
        }
        for(i=0; i<13; i++)
        {
                printf("ATR的值为%x\r\n",Receive_HOTATR_Info[i]);
        }
}

void Smart_Card_STOP(void)
{
        //T7
        Delay_X10us(1);
        GPIO_ResetBits( GPIOB, GPIO_Pin_6);
        //T8
        Delay_X10us(3);
        Smart_Card_CKEN_Cmd(Smart_Card,DISABLE);
        //T9
        Delay_X10us(1);
        GPIO_ResetBits( GPIOB, GPIO_Pin_5);
        GPIO_ResetBits( GPIOB, GPIO_Pin_11);
}

void Smart_Card_SendDataCust(uint32_t data)
{
        Smart_Card_TREN_Cmd(Smart_Card,SC_TREN_TXEN);                   //SC发送接收使能位

//      SC_SendData(Smart_Card,data);
        Smart_Card->SC0_DATA=data;
        while(Smart_Card_GetFlagStatus(Smart_Card,SC_Flag_TBUSY)==1);
        while(!(SC_TCFlag==1))
  SC_TCFlag=0;
}
```

```
void SendPPS(void)
{
	unsigned char i;
	Smart_Card_SendDataCust(0XFF);	//PPST
	Smart_Card_SendDataCust(0X10);	//PPS0
	Smart_Card_SendDataCust(0X94);	//PPS1
	Smart_Card_SendDataCust(0X78);	//PPS2
	Smart_Card_TREN_Cmd(Smart_Card,SC_TREN_RXEN);		//SC发送接收使能位
	for(i=0;i<4;i++)
	{
		while(!(SC_RCFlag==1));
		SC_RCFlag=0;
		Receive_PPS_Info[i] = Smart_Card_ReceiveData(Smart_Card);
	}
	for(i=0;i<4;i++)
	{
		printf("PPS的值为%x\r\n",Receive_PPS_Info[i]);
	}
}

void SendCommend(void)
{
	unsigned char i;
	//判断是否是白卡
	//进入3F00主文件
	Smart_Card_SendDataCust(0xA0);
	Smart_Card_SendDataCust(0XA4);
	Smart_Card_SendDataCust(0X00);
	Smart_Card_SendDataCust(0X00);
	Smart_Card_SendDataCust(0X02);
	Delay_X10us(20000);
	Smart_Card_SendDataCust(0X3F);
	Smart_Card_SendDataCust(0X00);
//	//进入7F20文件
//	Delay_X10us(20000);
//	Smart_Card_SendDataCust(0xA0);
//	Smart_Card_SendDataCust(0XA4);
//	Smart_Card_SendDataCust(0X00);
//	Smart_Card_SendDataCust(0X00);
//	Smart_Card_SendDataCust(0X02);
//	Delay_X10us(20000);
//	Smart_Card_SendDataCust(0X2F);
//	Smart_Card_SendDataCust(0XE2);
	//进入7F20文件
//	Delay_X10us(20000);
//	Smart_Card_SendDataCust(0xA0);
//	Smart_Card_SendDataCust(0XB0);
//	Smart_Card_SendDataCust(0X00);
//	Smart_Card_SendDataCust(0X00);
//	Smart_Card_SendDataCust(0X0A);
	Smart_Card_TREN_Cmd(Smart_Card,SC_TREN_RXEN);		//SC发送接收使能位
	for(i=0;i<=1;i++)
	{
		while(!(SC_RCFlag==1));
		SC_RCFlag=0;
		Receive_commend_Info[i] = Smart_Card_ReceiveData(Smart_Card);
	}
	for(i=0;i<1;i++)
	{
```

```
                printf("commend的值为%x\r\n",Receive_commend_Info[i]);
        }
}

//精确定时器   X*10us
void Delay_X10us(uint32_t count)
{
        TIM_ITConfig(TIM2, TIM_IT_INTEN, ENABLE); //使能中断
        TIM_Cmd(TIM2, ENABLE);
        while(Timer2_cnt<count)
        {
        }
        TIM_Cmd(TIM2, DISABLE);
        TIM_ITConfig(TIM2, TIM_IT_INTEN, DISABLE); //关闭中断
        Timer2_cnt=0;
}
//精确定时器   X*1ms
void Delay_X1ms(uint32_t count1)
{
        uint32_t count100;
        count100 = count1*100;
        TIM_ITConfig(TIM2, TIM_IT_INTEN, ENABLE); //使能中断
        TIM_Cmd(TIM2, ENABLE);
        while(Timer2_cnt<count100)
        {
        }
        TIM_Cmd(TIM2, DISABLE);
        TIM_ITConfig(TIM2, TIM_IT_INTEN, DISABLE); //关闭中断
        Timer2_cnt=0;
}
//精确定时器   X*1s
void Delay_X1s(uint32_t count2)
{
        uint32_t count100000;
        count100000 = count2*100000;
        TIM_ITConfig(TIM2, TIM_IT_INTEN, ENABLE); //使能中断
        TIM_Cmd(TIM2, ENABLE);
        while(Timer2_cnt<count100000)
        {
        }
        TIM_Cmd(TIM2, DISABLE);
        TIM_ITConfig(TIM2, TIM_IT_INTEN, DISABLE); //关闭中断
        Timer2_cnt=0;
}



void Smart_Card_Communication_SC0Handler(void)
{
 if(Smart_Card_GetFlagStatus(Smart_Card,SC_Flag_RC))  //接收完成标志
 {
                SC_RCFlag = 1;
  Smart_Card_ClearFlag(Smart_Card,SC_Flag_RC);
 }
        if(Smart_Card_GetFlagStatus(Smart_Card,SC_Flag_TC))  //发送完成标志
 {
                SC_TCFlag = 1;
  Smart_Card_ClearFlag(Smart_Card,SC_Flag_TC);
 }
        if(Smart_Card_GetFlagStatus(Smart_Card,SC_Flag_ROVF))  //接收溢出标志
```

```
 {
                SC_ROVFFlag = 1;
  Smart_Card_ClearFlag(Smart_Card,SC_Flag_ROVF);
 }
        if(Smart_Card_GetFlagStatus(Smart_Card,SC_Flag_FER))  //接收帧错误标志
 {
                SC_FERFlag = 1;
  Smart_Card_ClearFlag(Smart_Card,SC_Flag_FER);

 }
        if(Smart_Card_GetFlagStatus(Smart_Card,SC_Flag_WTER))  //等待超时标志
 {
                SC_WTERFlag = 1;
  Smart_Card_ClearFlag(Smart_Card,SC_Flag_WTER);
 }
        if(Smart_Card_GetFlagStatus(Smart_Card,SC_Flag_RPER))  //接收数据奇偶校验错误标志
 {
                SC_RPERFlag = 1;
  Smart_Card_ClearFlag(Smart_Card,SC_Flag_RPER);
 }
        if(Smart_Card_GetFlagStatus(Smart_Card,SC_Flag_TPER))  //发送数据奇偶校验错误标志
 {
                SC_TPERFlag = 1;
  Smart_Card_ClearFlag(Smart_Card,SC_Flag_TPER);
 }
        if(Smart_Card_GetFlagStatus(Smart_Card,SC_Flag_RBUSY))  //数据接收忙状态
 {
                SC_RBUSYFlag = 1;
  Smart_Card_ClearFlag(Smart_Card,SC_Flag_RBUSY);
 }
        if(Smart_Card_GetFlagStatus(Smart_Card,SC_Flag_TBUSY))  //数据发送忙状态
 {
                SC_TBUSYFlag = 1;
  Smart_Card_ClearFlag(Smart_Card,SC_Flag_TBUSY);
 }
        if(Smart_Card_GetFlagStatus(Smart_Card,SC_Flag_WTRT))  //等待数据重发状态
 {
                SC_WTRTFlag = 1;
  Smart_Card_ClearFlag(Smart_Card,SC_Flag_WTRT);
 }
}

void Smart_Card_TIMER2_IRQHandler(void)
{
        if(TIM_GetFlagStatus(TIM2, TIM_Flag_TI) == SET)
        {
                TIM_ClearFlag(TIM2, TIM_Flag_TI);       //清除TIM0中断
                Timer2_Flag=1;
                Timer2_cnt++;
        }
}
```

37 AES 固件库函数

赛元SC32L14xx、SC32R807系列内置的高级加密运算模块（AES）可对数据进行加密或解密，支持多种链接模式（ECB、CBC、CTR），支持128/192/256位密钥大小。

AES 寄存器结构

赛元SC32L14xx、SC32R807系列SmartCard寄存器结构，AES_TRNG_TypeDef，在文件“sc32L14xx.h”和“sc32r807.h”中定义如下：

```
typedef struct
{
  __IO uint32_t RESERVED0;
  __IO uint32_t RESERVED1;
  __IO uint32_t RESERVED2;
  __IO uint32_t RESERVED3;
  __IO uint32_t AES_CONFIG;
  __IO uint32_t TRNG_DATA;
  __IO uint32_t TRNG_STS;
} AES_TRNG_TypeDef;
```

AES寄存器表格

Table 739

寄存器	描述
AES_CONFIG	AES配置寄存器
TRNG_DATA	TRNG数据寄存器
TRNG_STS	TRNG标志寄存器

AES 固件库函数列表

Table 740

函数名	描述
AES_DeInit	将AES寄存器设置为缺省值
AES_ECBModeConfig	AES算法的电子密码本（ECB）模式加密和解密配置函数
AES_CBCModeConfig	AES算法的密码分组链接（CBC）模式加密和解密配置函数
AES_CTRModeConfig	AES算法的计数器（CTR）模式加密和解密配置函数
AES_ITConfig	使能或者失能指定的AES中断
AES_GetFlagStatus	获取指定AES的标志位状态
AES_ClearFlag	清除指定AES的标志位状态

AES 固件库函数详解

AES_DeInit

Table 741

函数名	AES_DeInit
函数原型	void AES_DeInit(void)
功能描述	将AES寄存器设置为缺省值
输入参数	无
返回值	无

使用示例:

```
/* AES寄存器复位 */
AES_DeInit();
```

AES_ECBModeConfig

Table 742

函数名	AES_ECBModeConfig
函数原型	ErrorStatus AES_ECBModeConfig(AES_MODE_TypeDef AES_Mode,uint8_t *key,AES_KEYSIZE_TypeDef keysize,uint8_t *input, uint32_t length, uint8_t *output);
功能描述	AES算法的电子密码本（ECB）模式加密和解密配置函数
输入参数1	AES_Mode：AES工作模式的新状态
输入参数2	key：算法密钥的地址
输入参数3	keysize：输入密钥的长度
输入参数4	input：待计算数据的地址
输入参数5	length：待加密和解密的数据长度必须是16的倍数
输入参数6	output：输出计算后数据保存的地址
返回值	设置状态

AES_Mode

AES工作模式选择，参阅Table 743获得这个参数的所有成员。

Table 743

AES_Mode	描述
AES_MODE_ENCRYPT	AES加密模式
AES_MODE_ENCRYPT	AES解密模式

key

算法密钥的地址。

keysize

输入密钥的长度选择，参阅Table 744获得这个参数的所有成员。

Table 744

keysize	描述
AES_KEYSIZE_128B	AES的KEY大小为128bit
AES_KEYSIZE_192B	AES的KEY大小为192bit
AES_KEYSIZE_256B	AES的KEY大小为256bit

input

待计算数据的地址。

Length

待加密和解密的数据长度必须是16的倍数。

Output

输出计算后数据保存的地址。

使用示例:

```
/* ECB模式相关数据 */
const uint32_t ECB_KEY[]=
{
    0xa8013dfe, 0xa209e877,
    0xf34c181e, 0x8b1bfbe3
};
const uint32_t  ECB_Encrypt[]=
{
  0xd8415837, 0xe4aed0f0, 0x5407ccca, 0xd919bdc9,
  0xf22c91a4, 0x9e01d6f6, 0x859aee1c, 0xe78e7c32,
  0xde4efe74, 0xaf3f04d7, 0x877698cc, 0x48b7d44e,
  0x2ab1244b, 0x557040bf, 0x901649bf, 0xc6ed0f47,
  0xa16bd6be, 0xa6c56564, 0xf4180825, 0xd4b0e940
};
const uint32_t  ECB_Decrypt[]=
{
```

```
    0x24a29fc2, 0x8d245c39, 0x2bd2d4a7, 0xad66bc64,
    0x834fe7f5, 0x685ec79a, 0x7dca1aac, 0x60ec8397,
    0xf49bc3c3, 0xc0cec7c2, 0x51883a0f, 0x9eeb4a68,
    0xba6319ff, 0x3de98191, 0x4ed381fe, 0x9b87a699,
    0xeb364a75, 0x938c61df, 0x7fce561b, 0x9affef00
};
AES_ECBModeConfig(AES_MODE_DECRYPT,(uint8_t*)&ECB_KEY[0],AES_KEYSIZE_128B,(uint8_t*)&ECB_
Decrypt[0],80,(uint8_t*)&AES_OutputData[0]);
```

AES_CBCModeConfig

Table 745

函数名	AES_CBCModeConfig
函数原型	ErrorStatus AES_CBCModeConfig (AES_MODE_TypeDef AES_Mode, uint8_t *key, AES_KEYSIZE_TypeDef keysize,uint8_t* Vector,uint8_t *input, uint32_t length, uint8_t *output)
功能描述	AES算法的密码分组链接（CBC）模式加密和解密配置函数
输入参数1	AES_Mode：AES工作模式的新状态
输入参数2	key：算法密钥的地址
输入参数3	keysize：输入密钥的长度
输入参数4	Vector：算法初始化向量的地址
输入参数5	input：待计算数据的地址
输入参数6	length：待加密和解密的数据长度必须是16的倍数
输入参数7	output：输出计算后数据保存的地址
返回值	设置状态

AES_Mode

AES工作模式选择，参阅Table 743获得这个参数的所有成员。

key

算法密钥的地址。

keysize

输入密钥的长度选择，参阅Table 744获得这个参数的所有成员。

Vector

算法初始化向量的地址。

input

待计算数据的地址。

Length

待加密和解密的数据长度必须是16的倍数。

Output

输出计算后数据保存的地址。

使用示例:

```
/* CBC模式相关数据 */
const uint32_t CBC_KEY[]=
{
    0x22eeffe6, 0x7b637127,
    0x2becad7c, 0xa09f9a27
};
const uint32_t CBC_IV[]=
{
    0x566fbcfa, 0x9048d378,
    0x1c4f78a9, 0x39bd698e
};
const uint32_t  CBC_Encrypt[]=
{
   0xd862fc44, 0x2f836da7, 0xee3a3d26, 0x1d7c9879,
   0x9b0ef74a, 0xf811abab, 0x562cbf95, 0xa048d55b,
   0x9002bcdd, 0xd0eda99b, 0x09a39700, 0x34060d1f,
```

```
    0x5d62a82d, 0xe0d99337, 0xec8545c1, 0x889da1f8,
    0xc49befb4, 0xfff98225, 0xeb964f86, 0x1ae20972
};
const uint32_t  CBC_Decrypt[]=
{
    0x4fd68736, 0xc6f8a019, 0xd1af97d8, 0xb2b5c17e,
    0x448e572f, 0x9989f418, 0xf1adeaf2, 0xcdb1ed04,
    0x8a4faa10, 0xb5f4ce4c, 0xc698eb1d, 0x2732915b,
    0x5b44522c, 0x8b69aad2, 0xace951b7, 0xff5c09d9,
    0x14f33eb2, 0x79321093, 0xeebcae83, 0x28d09c17
};
AES_CBCModeConfig(AES_MODE_ENCRYPT,(uint8_t*)&CBC_KEY[0],AES_KEYSIZE_128B,(uint8_t*)&CBC_
IV[0],(uint8_t*)&CBC_Encrypt[0],80,(uint8_t*)&AES_OutputData[0]);
```

AES_CTRModeConfig

Table 746

函数名	AES_CTRModeConfig
函数原型	ErrorStatus AES_CTRModeConfig (AES_MODE_TypeDef AES_Mode, uint8_t *key, AES_KEYSIZE_TypeDef keysize,uint8_t* Vector,uint8_t *input, uint32_t length, uint8_t *output)
功能描述	AES算法的计数器（CTR）模式加密和解密配置函数
输入参数1	AES_Mode：AES工作模式的新状态
输入参数2	key：算法密钥的地址
输入参数3	keysize：输入密钥的长度
输入参数4	Vector：算法初始化向量的地址
输入参数5	input：待计算数据的地址
输入参数6	length：待加密和解密的数据长度必须是16的倍数
输入参数7	output：输出计算后数据保存的地址
返回值	设置状态

AES_Mode

AES工作模式选择，参阅Table 743获得这个参数的所有成员。

key

算法密钥的地址。

keysize

输入密钥的长度选择，参阅Table 744获得这个参数的所有成员。

Vector

算法初始化向量的地址。

input

待计算数据的地址。

Length

待加密和解密的数据长度必须是16的倍数。

Output

输出计算后数据保存的地址。

使用示例:

```
/* CTR模式相关数据 */
const uint32_t CTR_KEY[]=
{
    0xe2381c91, 0xc4a00b35,
    0x4dd32aed, 0x5f421dcb
};
const uint32_t CTR_IV[]=
{
    0xbe875f9b, 0xbffdfa56,
    0x7a830008, 0xd02a7a2b
};
```

```
const uint32_t  CTR_Encrypt[]=
{
   0xdc6d550d, 0x57614677, 0x97657df8, 0x73b917ca,
   0x71f360dc, 0xffeafefc, 0x6f6b97d5, 0xc5c4df06,
   0x836ffb0d, 0x32ea38da, 0x1304a313, 0x8fb0bb5a,
   0x876616b9, 0x262ef614, 0x97ae70e9, 0x05dde53b,
   0xd9e89585, 0x48bc1062, 0xc406e64f, 0x0d46ab46
};
const uint32_t  CTR_Decrypt[]=
{
   0x39b6ccd4, 0x38c4e38a, 0x0ef898e7, 0xf24c5da8,
   0x5fe1fbbc, 0x59ef4517, 0xf242cccf, 0xe25a6bcb,
   0xd2fb19d5, 0x88d3c7ec, 0xf9ee566e, 0xdd750cbd,
   0xefbed92c, 0x8503f890, 0x256c52e2, 0xe9e82e50,
   0xf2133f32, 0x2e07a71a, 0x7110ebc1, 0x3c21896d
};
AES_CTRModeConfig(AES_MODE_ENCRYPT,(uint8_t*)&CTR_KEY[0],AES_KEYSIZE_128B,(uint8_t*)&CTR_I
V[0],(uint8_t*)&CTR_Encrypt[0],80,(uint8_t*)&AES_OutputData[0]);
```

AES_ITConfig

Table 747

函数名	AES_ITConfig
函数原型	void AES_ITConfig (uint16_t AES_IT, FunctionalState NewState)
功能描述	使能或者失能指定的AES中断
输入参数1	AES_IT：待使能或者失能的AES中断源
输入参数2	NewState：AES中断功能开启/关闭，可取值ENABLE或DISABLE
返回值	无

AES_IT

AES中断选择，参阅 Table 748获得这个参数的所有成员。

Table 748

AES_IT	描述
AES_IT_INTEN	中断请求CPU 的屏蔽位

使用示例:

```
/*打开中断请求 CPU的使能*/
AES_ITConfig (AES_IT_INTEN,ENABLE);
```

AES_GetFlagStatus

Table 749

函数名	AES_GetFlagStatus
函数原型	FlagStatus AES_GetFlagStatus (AES_Flag_TypeDef AES_FLAG)
功能描述	获取指定AES的标志位状态
输入参数	AES_FLAG：选择AES的标志位
返回值	AES_FLAG的新状态

SC_FLAG

选择mart_Card的标志位，参阅 Table 750获得这个参数的所有成员。

Table 750

AES_FLAG	描述
AES_Flag_BUSY	AES计算中状态位
AES_Flag_CCFIF	AES计算完成标志位

使用示例:

```
/* 检查AES计算完成标志位设置与否 */
FlagStatus  AES_Status;
AES_Status  = AES_GetFlagStatus (AES, AES_Flag_CCFIF);
```

AES_ClearFlag

Table 751

函数名	AES_ClearFlag
函数原型	void AES_ClearFlag (uint16_t AES_FLAG)
功能描述	清除指定AES的标志位状态
输入参数	AES_FLAG：AES 的标志位，定义在AES_Flag_TypeDef中
返回值	无

AES_FLAG

选择AES的标志位，参阅 Table 750获得这个参数的所有成员。

使用示例:

```
/* 清除AES计算完成标志位 */
AES_ClearFlag (AES_Flag_CCFIF);
```

综合使用示例

```
/*ECB 模式相关数据*/
const uint32_t ECB_KEY[]=
{
    0xa8013dfe, 0xa209e877,
    0xf34c181e, 0x8b1bfbe3
};
const uint32_t  ECB_Encrypt[]=
{
   0xd8415837, 0xe4aed0f0, 0x5407ccca, 0xd919bdc9,
   0xf22c91a4, 0x9e01d6f6, 0x859aee1c, 0xe78e7c32,
   0xde4efe74, 0xaf3f04d7, 0x877698cc, 0x48b7d44e,
   0x2ab1244b, 0x557040bf, 0x901649bf, 0xc6ed0f47,
   0xa16bd6be, 0xa6c56564, 0xf4180825, 0xd4b0e940
};
const uint32_t  ECB_Decrypt[]=
{
   0x24a29fc2, 0x8d245c39, 0x2bd2d4a7, 0xad66bc64,
   0x834fe7f5, 0x685ec79a, 0x7dca1aac, 0x60ec8397,
   0xf49bc3c3, 0xc0cec7c2, 0x51883a0f, 0x9eeb4a68,
   0xba6319ff, 0x3de98191, 0x4ed381fe, 0x9b87a699,
   0xeb364a75, 0x938c61df, 0x7fce561b, 0x9affef00
};

/*CBC 模式相关数据*/
const uint32_t CBC_KEY[]=
{
    0x22eeffe6, 0x7b637127,
    0x2becad7c, 0xa09f9a27
};
const uint32_t CBC_IV[]=
{
    0x566fbcfa, 0x9048d378,
    0x1c4f78a9, 0x39bd698e
};
const uint32_t  CBC_Encrypt[]=
{
   0xd862fc44, 0x2f836da7, 0xee3a3d26, 0x1d7c9879,
   0x9b0ef74a, 0xf811abab, 0x562cbf95, 0xa048d55b,
   0x9002bcdd, 0xd0eda99b, 0x09a39700, 0x34060d1f,
   0x5d62a82d, 0xe0d99337, 0xec8545c1, 0x889da1f8,
   0xc49befb4, 0xfff98225, 0xeb964f86, 0x1ae20972
};
const uint32_t  CBC_Decrypt[]=
{
```

```
    0x4fd68736, 0xc6f8a019, 0xd1af97d8, 0xb2b5c17e,
    0x448e572f, 0x9989f418, 0xf1adeaf2, 0xcdb1ed04,
    0x8a4faa10, 0xb5f4ce4c, 0xc698eb1d, 0x2732915b,
    0x5b44522c, 0x8b69aad2, 0xace951b7, 0xff5c09d9,
    0x14f33eb2, 0x79321093, 0xeebcae83, 0x28d09c17
};

/*CTR 模式相关数据*/
const uint32_t CTR_KEY[]=
{
   0xe2381c91, 0xc4a00b35,
    0x4dd32aed, 0x5f421dcb
};
const uint32_t CTR_IV[]=
{
   0xbe875f9b, 0xbffdfa56,
    0x7a830008, 0xd02a7a2b
};
const uint32_t  CTR_Encrypt[]=
{
    0xdc6d550d, 0x57614677, 0x97657df8, 0x73b917ca,
    0x71f360dc, 0xffeafefc, 0x6f6b97d5, 0xc5c4df06,
    0x836ffb0d, 0x32ea38da, 0x1304a313, 0x8fb0bb5a,
    0x876616b9, 0x262ef614, 0x97ae70e9, 0x05dde53b,
    0xd9e89585, 0x48bc1062, 0xc406e64f, 0x0d46ab46
};
const uint32_t  CTR_Decrypt[]=
{
    0x39b6ccd4, 0x38c4e38a, 0x0ef898e7, 0xf24c5da8,
    0x5fe1fbbc, 0x59ef4517, 0xf242cccf, 0xe25a6bcb,
    0xd2fb19d5, 0x88d3c7ec, 0xf9ee566e, 0xdd750cbd,
    0xefbed92c, 0x8503f890, 0x256c52e2, 0xe9e82e50,
    0xf2133f32, 0x2e07a71a, 0x7110ebc1, 0x3c21896d
};

uint32_t AES_OutputData[50];
volatile FlagStatus AES_CalcuFlag = RESET;

void AES_EncryptionAndDecryption_Function(void)
{
      uint32_t i;
      RCC_AHBPeriphClockCmd(RCC_AHBPeriph_AES,ENABLE);//开启AES时钟

      AES_ITConfig(AES_IT_INTEN,ENABLE);
      NVIC_EnableIRQ(AES_IRQn);
      /*ECB加解密数据*/
      printf("************************************************************\r\n");
      printf("Start the encryption and decryption of ECB mode data\r\n");
      printf("ECB data to be encrypted = { ");
      for(i=0;i<20;i++)
      {
          printf("%x ", ECB_Encrypt[i]);
      }
      printf("}\r\n");
      printf("ECB data to be decrypted = { ");
      for(i=0;i<20;i++)
      {
          printf("%x ", ECB_Decrypt[i]);
      }
      printf("}\r\n");
```

```
AES_ECBModeConfig(AES_MODE_ENCRYPT,(uint8_t*)&ECB_KEY[0],AES_KEYSIZE_128B,(uint8_t*)&E
CB_Encrypt[0],80,(uint8_t*)&AES_OutputData[0]);
while(AES_CalcuFlag == RESET);
AES_CalcuFlag = RESET;
printf("The data is encrypted using AES, and the encrypted data is = { ");
for(i=0;i<20;i++)
{
    printf("%x ", AES_OutputData[i]);
}
printf("}\r\n");


AES_ECBModeConfig(AES_MODE_DECRYPT,(uint8_t*)&ECB_KEY[0],AES_KEYSIZE_128B,(uint8_t*)&E
CB_Decrypt[0],80,(uint8_t*)&AES_OutputData[0]);
while(AES_CalcuFlag == RESET);
AES_CalcuFlag = RESET;
printf("The data is decrypted using AES, and the decrypted data is = { ");
for(i=0;i<20;i++)
{
    printf("%x ", AES_OutputData[i]);
}
printf("}\r\n");
printf("************************************************************\r\n");

/*CBC加解密数据*/
printf("************************************************************\r\n");
printf("Start the encryption and decryption of CBC mode data\r\n");
printf("CBC data to be encrypted = { ");
for(i=0;i<20;i++)
{
   printf("%x ", CBC_Encrypt[i]);
 }
printf("}\r\n");
printf("CBC data to be decrypted = { ");
for(i=0;i<20;i++)
{
 printf("%x ", CBC_Decrypt[i]);
 }
printf("}\r\n");

AES_CBCModeConfig(AES_MODE_ENCRYPT,(uint8_t*)&CBC_KEY[0],AES_KEYSIZE_128B,(uint8_t*)&C
BC_IV[0],(uint8_t*)&CBC_Encrypt[0],80,(uint8_t*)&AES_OutputData[0]);
while(AES_CalcuFlag == RESET);
AES_CalcuFlag = RESET;
printf("The data is encrypted using AES, and the encrypted data is = { ");
for(i=0;i<20;i++)
{
    printf("%x ", AES_OutputData[i]);
}
printf("}\r\n");


AES_CBCModeConfig(AES_MODE_DECRYPT,(uint8_t*)&CBC_KEY[0],AES_KEYSIZE_128B,(uint8_t*)&C
BC_IV[0],(uint8_t*)&CBC_Decrypt[0],80,(uint8_t*)&AES_OutputData[0]);
while(AES_CalcuFlag == RESET);
AES_CalcuFlag = RESET;
printf("The data is decrypted using AES, and the decrypted data is = { ");
for(i=0;i<20;i++)
{
    printf("%x ", AES_OutputData[i]);
 }
```

```
    printf("}\r\n");
    printf("************************************************************\r\n");

    /*CTR加解密数据*/
    printf("************************************************************\r\n");
    printf("Start the encryption and decryption of CTR mode data\r\n");
    printf("CTR data to be encrypted = { ");
    for(i=0;i<20;i++)
    {
        printf("%x ", CTR_Encrypt[i]);
    }
    printf("}\r\n");
    printf("CTR data to be decrypted = { ");
    for(i=0;i<20;i++)
    {
        printf("%x ", CTR_Decrypt[i]);
    }
    printf("}\r\n");

    AES_CTRModeConfig(AES_MODE_ENCRYPT,(uint8_t*)&CTR_KEY[0],AES_KEYSIZE_128B,(uint8_t*)&C
    TR_IV[0],(uint8_t*)&CTR_Encrypt[0],80,(uint8_t*)&AES_OutputData[0]);
    while(AES_CalcuFlag == RESET);
    AES_CalcuFlag = RESET;
    printf("The data is encrypted using AES, and the encrypted data is = { ");
    for(i=0;i<20;i++)
    {
        printf("%x ", AES_OutputData[i]);
    }
    printf("}\r\n");


   AES_CTRModeConfig(AES_MODE_ENCRYPT,(uint8_t*)&CTR_KEY[0],AES_KEYSIZE_128B,(uint8_t*)&CT
   R_IV[0],(uint8_t*)&CTR_Decrypt[0],80,(uint8_t*)&AES_OutputData[0]);
  while(AES_CalcuFlag == RESET);
    AES_CalcuFlag = RESET;
    printf("The data is decrypted using AES, and the decrypted data is = { ");
    for(i=0;i<20;i++)
    {
        printf("%x ", AES_OutputData[i]);
    }
    printf("}\r\n");
    printf("************************************************************\r\n");
   while(1)
    {

    }
}

//AES中断服务函数
void AES_CalcuCompletedHandler(void)
{
   if(AES_GetFlagStatus(AES_Flag_CCFIF) != RESET)
    {
        AES_ClearFlag(AES_Flag_CCFIF);
        AES_CalcuFlag = SET;
    }
}
```

38 TRNG 固件库函数

赛元SC32L14xx、SC32R807系列内部集成一个真随机数发生器（简称TRNG），可以产生32位的真随机数，随机数由输入时钟经过模拟熵源生成。

TRNG 寄存器结构

赛元SC32L14xx、SC32R807系列TRNG寄存器结构，AES_TRNG_TypeDef，在文件“sc32L14xx.h”和“sc32r807.h”中定义如下：

```
typedef struct
{
  __IO uint32_t RESERVED0;
  __IO uint32_t RESERVED1;
  __IO uint32_t RESERVED2;
  __IO uint32_t RESERVED3;
  __IO uint32_t AES_CONFIG;
  __IO uint32_t TRNG_DATA;
  __IO uint32_t TRNG_STS;
} AES_TRNG_TypeDef;
```

TRNG寄存器表格

Table 752

寄存器	描述
AES_CONFIG	AES配置寄存器
TRNG_DATA	TRNG数据寄存器
TRNG_STS	TRNG标志寄存器

TRNG 固件库函数列表

Table 753

函数名	描述
TRNG_DeInit	将TRNG寄存器设置为缺省值
TRNG_Cmd	使能或者失能TRNG外设
TRNG_TLSELConfig	TRNG随机数产生方式选择
TRNG_TLDIVConfig	TRNG随机数产生速率选择
TRNG_GetRandomNumber	返回一个32位随机数
TRNG_ITConfig	使能或者失能指定的TRNG中断
TRNG_GetFlagStatus	获取指定TRNG的标志位状态
TRNG_ClearFlag	清除指定TRNG的标志位状态

TRNG 固件库函数详解

TRNG_DeInit

Table 754

函数名	TRNG_DeInit
函数原型	void TRNG_DeInit (void)
功能描述	将TRNG寄存器设置为缺省值
输入参数	无
返回值	无

使用示例:

```
/* TRNG寄存器复位 */
TRNG_DeInit ();
```

TRNG_Cmd

Table 755

函数名	TRNG_Cmd
函数原型	void TRNG_Cmd(FunctionalState NewState)
功能描述	使能或者失能TRNG外设
输入参数	NewState：TRNG 功能开启/关闭，可取值ENABLE或DISABLE
返回值	无

使用示例:

```
/* TRNG使能 */
TRNG_Cmd(ENABLE);
```

TRNG_TLSELConfig

Table 756

函数名	TRNG_TLSELConfig
函数原型	void TRNG_TLSELConfig(uint8_t TRNG_TLSEL)
功能描述	TRNG随机数产生方式选择
输入参数	TRNG_TLSEL：TRNG时钟f_{HCLK}=24MHz，TLSEL[2:0]=000~011，取值0x00~0x03
返回值	无

使用示例:

```
/* TRNG随机数产生方式选择 */
TRNG_TLSELConfig(0x00);
```

TRNG_TLDIVConfig

Table 757

函数名	TRNG_TLDIVConfig
函数原型	void TRNG_TLDIVConfig(uint8_t TRNG_TLDIV)
功能描述	TRNG随机数产生速率选择
输入参数	TRNG_TLDIV：TRNG时钟f_{HCLK}=24MHz， TLDIV[2:0]=101，取值0x05
返回值	无

使用示例:

```
/* TRNG随机数产生速率选择 */
TRNG_TLDIVConfig(0x05);
```

TRNG_GetRandomNumber

Table 758

函数名	TRNG_GetRandomNumber
函数原型	uint32_t TRNG_GetRandomNumber(void)
功能描述	返回一个32位随机数
输入参数	无
返回值	一个32位随机数

使用示例:

```
/* 返回一个32位随机数 */
volatile uint32_t Trng_Num;
Trng_Num = TRNG_GetRandomNumber();
```

TRNG_ITConfig

Table 759

函数名	TRNG_ITConfig
函数原型	void TRNG_ITConfig(uint16_t TRNG_IT,FunctionalState NewState)
功能描述	使能或者失能指定的TRNG中断
输入参数1	TRNG_IT：待使能或者失能的TRNG中断源

输入参数2	NewState：TRNG中断功能开启/关闭，可取值ENABLE或DISABLE
返回值	无

SC_IT

TRNG中断选择，参阅Table 760获得这个参数的所有成员。

Table 760

TRNG_IT	描述
TRNG_IT_INTEN	中断请求CPU 的屏蔽位
TRNG_IT_DRDYIE	TRNG数据生成完毕中断使能位

使用示例:

```
/*打开中断请求 CPU的使能和TRNG数据生成完毕中断使能*/
TRNG_ITConfig(TRNG_IT_INTEN | TRNG_IT_DRDYIE,ENABLE);
```

TRNG_GetFlagStatus

Table 761

函数名	TRNG_GetFlagStatus
函数原型	FlagStatus TRNG_GetFlagStatus(TRNG_Flag_TypeDef TRNG_FLAG)
功能描述	获取指定TRNG的标志位状态
输入参数	TRNG_FLAG：选择TRNG的标志位
返回值	TRNG_FLAG 的新状态

TRNG_FLAG

选择TRNG的标志位，参阅Table 762获得这个参数的所有成员。

Table 762

TRNG_FLAG	描述
TRNG_FLAG_SEIF	TRNG种子错误中断标志位
TRNG_FLAG_DRDYIF	TRNG数据生成完毕标志位

使用示例:

```
/* 检查TRNG数据生成完毕标志位设置与否 */
FlagStatus  TRNG_Status;
TRNG_Status  = TRNG_GetFlagStatus (TRNG_FLAG_DRDYIF);
```

TRNG_ClearFlag

Table 763

函数名	TRNG_ClearFlag
函数原型	void TRNG_ClearFlag(uint8_t TRNG_FLAG)
功能描述	清除指定TRNG的标志位状态
输入参数2	TRNG_FLAG：TRNG的标志位，定义在TRNG_Flag_TypeDef中
返回值	无

TRNG_FLAG

选择TRNG的标志位，参阅 Table 762获得这个参数的所有成员。

使用示例:

```
/* 清除TRNG种子错误中断标志位 */
TRNG_ClearFlag(TRNG_FLAG_SEIF);
```

综合使用示例

```
volatile uint32_t Trng_Num;
volatile FlagStatus TRNG_Flag = RESET;

void TRNG_OutputRandomNumFunction(void)
{
   RCC_AHBPeriphClockCmd(RCC_AHBPeriph_AES,ENABLE);//开启AES时钟
   TRNG_TLSELConfig(0x00);
   TRNG_TLDIVConfig(0x05);
```

```
    TRNG_ITConfig(TRNG_IT_INTEN | TRNG_IT_DRDYIE,ENABLE);
    NVIC_EnableIRQ(TRNG_IRQn);
    TRNG_Cmd(ENABLE);

      while(1)
     {
       if(TRNG_Flag)
       {
            TRNG_Flag = RESET;
            printf("The random number has been generated success：0x%x ", Trng_Num);
            uint32_t delay_time = 50000;
            while(delay_time--);
            TRNG_Cmd(ENABLE);
        }
      }
}

void TRNG_GenerationCompleteHandler(void)
{
      TRNG_Cmd(DISABLE); //停止无限生成
      if(TRNG_GetFlagStatus(TRNG_FLAG_DRDYIF) != RESET) //生成完成中断
      {
            Trng_Num = TRNG_GetRandomNumber();
            TRNG_Flag =SET;
      }
      if(TRNG_GetFlagStatus(TRNG_FLAG_SEIF) != RESET) //生成错误中断
      {
            TRNG_ClearFlag(TRNG_FLAG_SEIF);
      }
}
```

规格更改记录

版本	记录	日期
V1.8.3	1.修改项 （1）添加对SC32F11xx芯片的适配 （2）将“sc32r803_qspi.h”修改为“sc32f1xxx_qspi.h” （3）将TWI_QSPIx_TypeDef 修改为 QSPI_TypeDef （4）修改OPTION_IAPPORA和OPTION_IAPPORB的使用案例 （5）修改WDT_Cmd的使用案例 （6）新增RCC固件库函数 （7）新增OPTION固件库函数	2026年7月
V1.8.2	1.修改项 （1）添加对SC32R808芯片的适配	2026年5月
V1.8.1	1.修改项 （1）添加对SC32R807芯片的适配	2026年4月
V1.8.0	1.修改项 （1）添加对SC32L14xx芯片的适配 （2）修改文件名称为“赛元SC32F1_32L1_32R_M0+系列固件库使用手册” （3）修改“CAN固件库函数”章节部分函数原型入参与枚举成员 （4）修改前言适用型号描述 2.新增项 （1）新增LPC固件库函数 （2）新增LPD固件库函数 （3）新增RTC固件库函数 （4）新增SmartCard固件库函数 （5）新增AES固件库函数 （6）新增TRNG固件库函数	2025年8月
V1.7.3	1.修改项 （1）添加对SC32R806芯片的适配	2025年6月
V1.7.2	1.修改项 （1）修改ADC综合使用示例 （2）添加对SC32R805芯片的适配	2025年4月
V1.7.1	1.修改项 （1）ADC相关库函数增加适用型号 （2）QEP寄存器结构体增加适用型号 （3）IAP相关库函数增加适用型号 （4）文档文字描述优化	2025年1月
V1.7.0	1.修改项 补充枚举UART_WK, RCC_APB1Periph_QTWI0/1 等 2.新增项 补充缺少的函数 QSPI_Receivelen(QSPI_TypeDef *QSPIx,uint32_t datalen) QSPI_CLKONLYSet(QSPI_TypeDef *QSPIx, uint32_t CLKONLY) 等OPTION_IAPPORA (uint16_t IAPPROAST, uint16_t IAPPROAED) OPTION_IAPPORB (uint16_t IAPPROBST, uint16_t IAPPROBED)	2024年12月
V1.6.0	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

V1.5.0	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
V1.4.0	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
V1.3.0	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
V1.2.0	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
V1.1.0	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 （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	2023年11月

	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	
V1.0.0	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。