UM0427

用户手册

基于 ARM 的 32 位 MCU STM32F101xx 和 STM32F103xx

固件库

简介

该文档描述了基于 ARM 的 32 位 MCU STM32F101xx 和 STM32F103xx 的固件库。

该库是一个固件包，其中包括了程序、数据结构和覆盖所有外设特性的宏单元。还包括设备驱动的描述以及每个外围模块的实例。该固件库使得用户在没有深入学习外围模块规格手册的情况下，也能够使用任何在用户应用中涉及到的设备。因此，使用该固件库可以节省您的许多时间，让您有更多的时间花费在编程方面，从而减少了在应用开发中的综合开销。

每个设备驱动包括一组涵盖所有外围功能的函数。每个驱动都是在标准 API(应用程序接口)下开发的，这使得使驱动结构、函数和参数名标准化。

驱动的源代码是用‘Strict ANSI-C’开发的（用于工程和实例文件的不严格的 ANSI-C）。它完整地记录在文档中，并且适应于 MISRA-C 2004（适应矩阵是可获得的）。用‘严格的 ANSI-C’编写整个库，使该库可以独立于软件工具链。只有启动文件需要依靠工具链。

该固件库通过校验所有库函数的输入值来实现运行时错误检测。该动态校验提高了软件的健壮性。运行时检测适合于用户应用程序的开发和调试。不过这增加了费用，可以在最终应用程序代码中移去，以最小化代码大小和执行速度。想要了解更多细节，请参阅章节 2.5: 运行时检测。

因为该固件库是通用的，并且包括了所有外围模块功能，所以应用程序代码的大小和执行速度可能不是最优的。对大多数应用程序来说，该固件库就这么使用。但是对于那些在代码大小和执行速度方面有严格要求的应用程序，该固件库驱动程序可以作为如何设置外围模块的一份参考资料，根据实际需求对其进行裁减。

此份固件库用户手册的整体架构如下：

定义，文档约定和固件库规则。

固件库概述（包的内容，库的架构），安装指南，库使用实例。

固件库具体描述：设置架构和每个外围模块的函数。

STM32F101xx 和 STM32F103xx 在整个文档中会被写作 STM32F10xxx

目录

1 文档和库的规则

该用户手册和固件库使用以下这些约定：

1.1 缩写

表 1 描述了文档中涉及的缩写

表 1．缩写列表

缩写	外围模块/单元
ADC	模拟/数字转换器
BKP	备份寄存器
CAN	控制器区域网络
DMA	DMA 控制器
EXTI	外部中断控制器
FLASH	Flash 存储器
GPIO	通用 I/O
I2C	Inter-integrated 电路
IWDG	独立看门狗
NVIC	嵌套向量中断控制器

PWR	电源控制
RCC	复位和时钟控制器
RTC	实时时钟
SPI	串行外设接口
SysTick	系统 tick 定时器
TIM	通用定时器
TIM1	先进的控制定时器
USART	通用同步异步接收传送器
WWDG	窗口看门狗

1.2 命名规则

该固件库使用以下命名规则：

●PPP 表示外围模块的缩写，例如 ADC。详见 Section 1.1：缩写。

●系统文件名和源/头文件名以‘stm32f10x_’的形式表示，例如 stm32f10x_conf.h。

●在单一文件中使用的常量在该文件中定义。在多个文件中使用的常量定义在头文件中。所有常量都以大写字母表示。

●寄存器当作常量看待，同样以大写字母表示。多数情况下， 在 STM32F10x 参考手册中使用相同的缩写。

●外围模块功能函数的名字，需要有相应的外围模块缩写加下划线这样的前缀。每个单词的首字符需要大写，例如 SPI_SendData。在一个函数名中，只允许有一条下划线，用来区分外围模块缩写

和剩下的函数名。

●使用 PPP_InitTypeDef 中指定的参数初始化 PPP 外围模块的函数，被命名为 PPP_Init，例如 TIM_Init。

●复位 PPP 外围模块寄存器为默认值的函数，命名为 PPP_DeInit，例如 TIM_DeInit。

●将 PPP_InitTypeDef 结构体每个成员设置为复位值的函数，命名为 PPP_StructInit，例如 USART_StructInit。

●用来使能或者禁止指定的 PPP 外围模块的函数，命名为 PPP_Cmd，例如 SPI_Cmd。

●用来使能或者禁止指定 PPP 外围模块的某个中断资源的函数，命名为 PPP_ITConfig，例如 RCC_ITConfig。

●用来使能或者禁止指定 PPP 外围模块的 DMA 接口的函数，命名为 PPP_DMAConfig，例如 TIM1_DMAConfig。

●用来设置某个外围模块的函数，总是以字符串‘Config’结尾，例如 GPIO_PinRemapConfig。

●用来检验指定 PPP 的标志是否被置位或清零的函数，命名为 PPP_GetFlagStatus，例如 I2C_GetFlagStatus。

●用来清除某个 PPP 的标志的函数，命名为 PPP_ClearFlag，例如 I2C_ClearFlag。

●用来检验指定 PPP 的中断是否发生的函数，命名为 PPP_GetITStatus，例如 I2C_GetITStatus。

●用来清除某个 PPP 中断挂起位的函数，命名为 PPP_ClearITPendingBit，例如 I2C_ClearITPendingBit。

1.3 代码标准

该章节描述在固件库中的代码标准。

1.3.1 变量

定义了 18 种具体的变量类型。其类型和大小都是固定的 ,在头文件 stm32f10x_type.h 中定义：

```
typedef signed long  s32;
typedef signed short s16;
typedef signed char  s8;
typedef volatile signed long  vs32;
typedef volatile signed short vs16;
typedef volatile signed char  vs8;
typedef unsigned long  u32;
typedef unsigned short u16;
typedef unsigned char  u8;
typedef unsigned long  const uc32;  /* Read Only */
typedef unsigned short const uc16;  /* Read Only */
typedef unsigned char  const uc8;   /* Read Only */
typedef volatile unsigned long  vu32;
typedef volatile unsigned short vu16;
typedef volatile unsigned char  vu8;
typedef volatile unsigned long  const vuc32;  /* Read Only */
typedef volatile unsigned short const vuc16;  /* Read Only */
```

```
typedef volatile unsigned char    const vuc8;     /* Read Only */
```

1.3.2 布尔(bool)类型

布尔类型在头文件 stm32f10x_type.h 中定义：

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

1.3.3 标志状态(FlagStatus)类型

标志状态类型在头文件 stm32f10x_type.h 中定义。该类型只可以被赋予以下两个值：SET 或者 RESET。

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

1.3.4 功能状态(FunctionalState)类型

功能状态类型在头文件 stm32f10x_type.h 中定义。该类型只可以被赋予以下两个值：ENABLE 或者 DISABLE。

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

1.3.5 错误状态(FunctionalState)类型

错误状态类型在头文件 stm32f10x_type.h 中定义。该类型只可以被赋予以下两个值：SUCCESS 或者 ERROR。

```
typedef enum
{
  ERROR   = 0,
  SUCCESS = !ER
} ErrorStatus;
```

1.3.6 外围模块

指向外围模块的指针，可以用来访问外围模块控制寄存器。此类型指针指向的数据结构代表了外围模块控制寄存器的映射。

外围模块控制寄存器结构

stm32f10x_map.h 包括了所有外围模块结构的定义。下面的实例给出了 SPI 寄存器结构的声明：

```
/*------------------ Serial Peripheral Interface ----------------*/
typedef struct
{
  vu16 CR1;
  u16 RESERVED0;
  vu16 CR2;
  u16 RESERVED1;
  vu16 SR;
  u16  RESERVED2;
  vu16 DR;
  u16  RESERVED3;
  vu16 CRCPR;
  u16 RESERVED4;
  vu16 RXCRCR;
```

```
    u16  RESERVED5;
    vu16 TXCRCR;
    u16  RESERVED6;
} SPI_TypeDef;
```

每个外围模块的寄存器名字是该寄存器的缩写，用大写表示。RESERVEDi（i 是一个整数，作为保留域的下标）表示保留域。

外围模块声明

所有的外围模块在 stm32f10x_map.h 中声明。下面的实例给出了 SPI 外围模块的声明：

```
#ifndef EXT
#Define EXT extern
#endif
...
#define PERIPH_BASE              ((u32)0x40000000)
#define APB1PERIPH_BASE          PERIPH_BASE
#define APB2PERIPH_BASE          (PERIPH_BASE + 0x10000)
...
/* SPI2 Base Address definition*/
#define SPI2_BASE                (APB1PERIPH_BASE + 0x3800)
```

```
...
/* SPI2 peripheral declaration*/
#ifndef DEBUG
...
#ifdef _SPI2
  #define SPI2                    ((SPI_TypeDef *) SPI2_BASE)
#endif /*_SPI2 */
...
#else    /* DEBUG */
...
#ifdef _SPI2
  EXT SPI_TypeDef                 *SPI2;
#endif /*_SPI2 */
...
#endif   /* DEBUG */
```

定义标签_SPI，用来在应用程序中引入 SPI 外围模块库。

定义 label_SPIn，用来访问 SPIn 的外围模块寄存器。例如，要想访问_SPI2 外围模块寄存器，必须在头文件 stm32f10x_conf.h 中定义标签_SPI2 标签。_SPI 和_SPIn 标签定义在头文件 stm32f10x_conf.h 中，如下：

#define _SPI

#define _SPI1

#define _SPI2

每个外围模块都有若干个专用寄存器，它们拥有不同的标志。每个外围模块都有专用结构体来定义各自的寄存器。标志的缩写需用大写字符，并且以‘PPP_FLAG_’开头。标志在头文件 stm32f10x_ppp.h 中定义，适用于每个外围模块。

为了进入调试模式，你必须在头文件 stm32f10x_conf.h 中定义标签 DEBUG。这样就构造了一个指向 SRAM 中外围模块结构体的指针，从而使调试变得更简单，并且所有寄存器设置可以通过转储一个外围变量来实现。在以上两种情况下，SPI2 是一个指向 SPI2 外围模块首地址的指针。

DEGUB 变量定义在头文件 stm32f10x_conf.h 中，如下：

#define DEBUG

调试模式在源文件 stm32f10x_lib.c 中初始化：

#ifdef DEBUG

void debug(void)

{

...

#ifdef _SPI2

```
  SPI2 = (SPI_TypeDef *)  SPI2_BASE;
#endif /*_SPI2 */
  ...
}
#endif /* DEBUG*/
```

注： 1 当选择调试模式时，**assert** 宏定义被扩展，并且在固件库代码中运行时检测被使能。

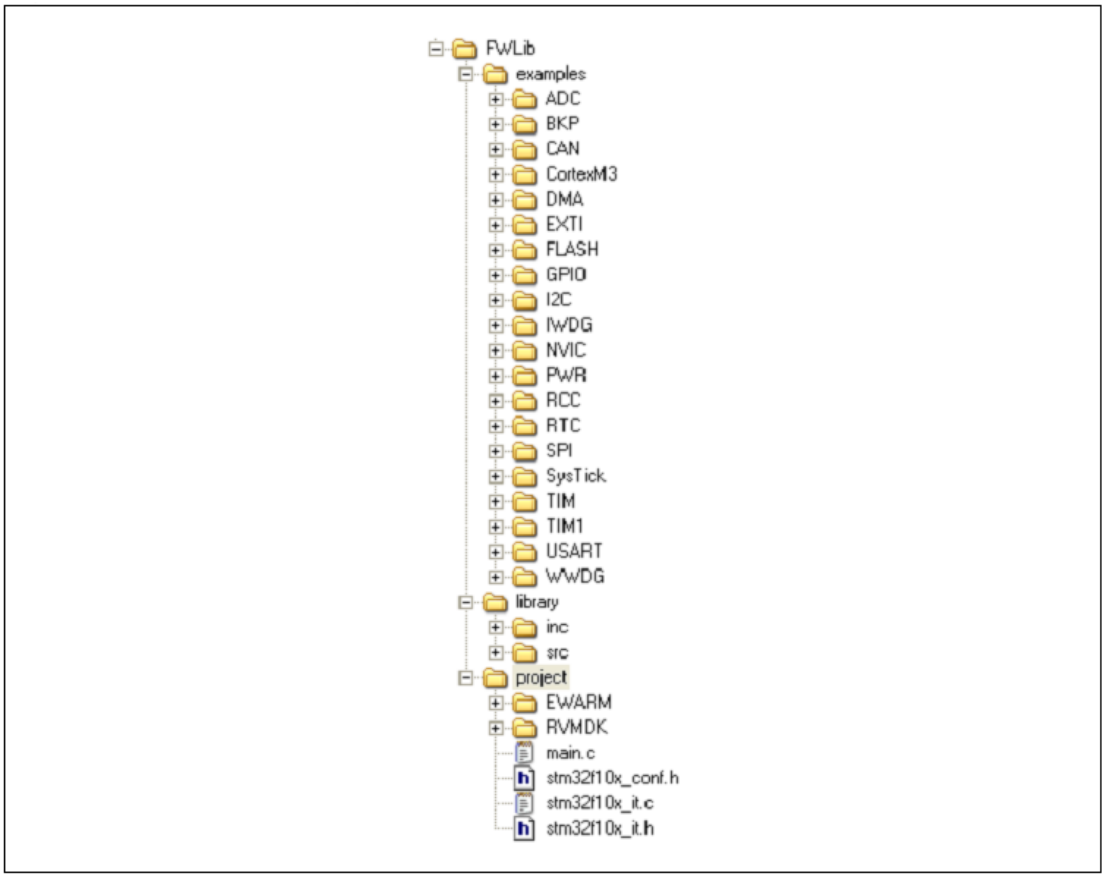
2 调试模式增加了代码大小，降低了代码性能。因此，推荐仅在调试应用程序的时候使用该模式，并在最终应用程序代码中去掉。

2 固件库

2.1 软件包描述

该固件库在一个单独的压缩包中提供。解压该压缩包会产生一个文件夹，**STM32F10xFWLib\FWLib**，它包含了如下子文件夹：

Figure 1. 固件库文件夹结构



2.1.1 示例（Examples）文件夹

Examples 文件夹中包括每个外围模块的子文件夹，子文件夹中提供了运行一个关于如何使用该外设的典型示例所需的最小文件集：

●**readme.txt -** 简短的文本文件，描述该示例以及如何使之工作，

●**stm32f10x_conf.h –** 头文件，配置所使用的外围模块，并且包括各种 DEFINE 语句，

●**stm32f10x_it.c –** 源文件，包括中断处理函数（如果某个功能没有使用，对应的函数可能是空的），

●**stm32f10x_it.h –** 头文件，包括所有中断处理函数的原型。

●main.c – 示例代码

注： 所有的示例都独立于软件工具链。

2.1.2 库（Library）文件夹

Library 文件夹包括子目录和构成库核心的文件：

inc 子文件夹包括固件库头文件。它们不需要用户修改：

stm32f10x_type.h：在所有其他文件中使用的普通数据类型和枚举，

stm32f10x_map.h：外围模块内存映射和寄存器数据结构，

stm32f10x_lib.h：主头文件，引用了所有其他头文件，

stm32f10x_ppp.h（每个外围模块对应一个头文件）：函数原型，数据结构和枚举，

cortexm3_macro.h：cortexm3_macro.s 的头文件。

src 子文件夹包括固件库源文件。它们不需要用户修改：

stm32f10x_ppp.c（每个外围模块对应一个源文件）：每个外围模块的函数体

stm32f10x_lib.c：所有外围模块指针初始化。

注： 所有库文件都是用‘Strict ANSI-C’编写，并且独立于软件工具链。

2.1.3 工程（Project）文件夹

Porject 文件夹包括一个标准的模板工程，该工程编译所有库文件和所有用于创建一个新工程所必需件的用户可修改文件：

stm32f10x_conf.h：配置头文件，包括所有外围模块的默认定义。

stm32f10x_it.c：源文件，包括中断处理函数（在这个模板中，函数体是空的）。

stm32f10x_it.h：头文件，包括所有中断处理函数原型。

main.c：主函数体。

- **EWARM, RVMDK**：由工具链使用，可参考同一目录下的 readme.txt 文件。

2.2 固件库文件描述

表 2 列出并描述了固件库使用的不同文件。

固件库的架构和文件包含关系显示在表 2 中。每个外围模块都有一个源程序文件，stm32f10x_ppp.c，一个头文件和 stm32f10x_ppp.h。stm32f10x_ppp.c 文件包括了使用 PPP 外围模块所需要的全部固件函数。一个单独的内存映射文件，stm32f10x_map.h，提供给所有的外围模块。它包含了调试和发布两种模式下所有的寄存器声明。

头文件 stm32f10x_lib.h 包含了所有外围模块的头文件。这是唯一一个需要在用户应用程序中引用的文件，它作为库的接口。

stm32f10x_conf.h 是唯一一个必须用户修改的文件。它用来在运行任何应用程序之前，详细说明连接固件库的参数设置。

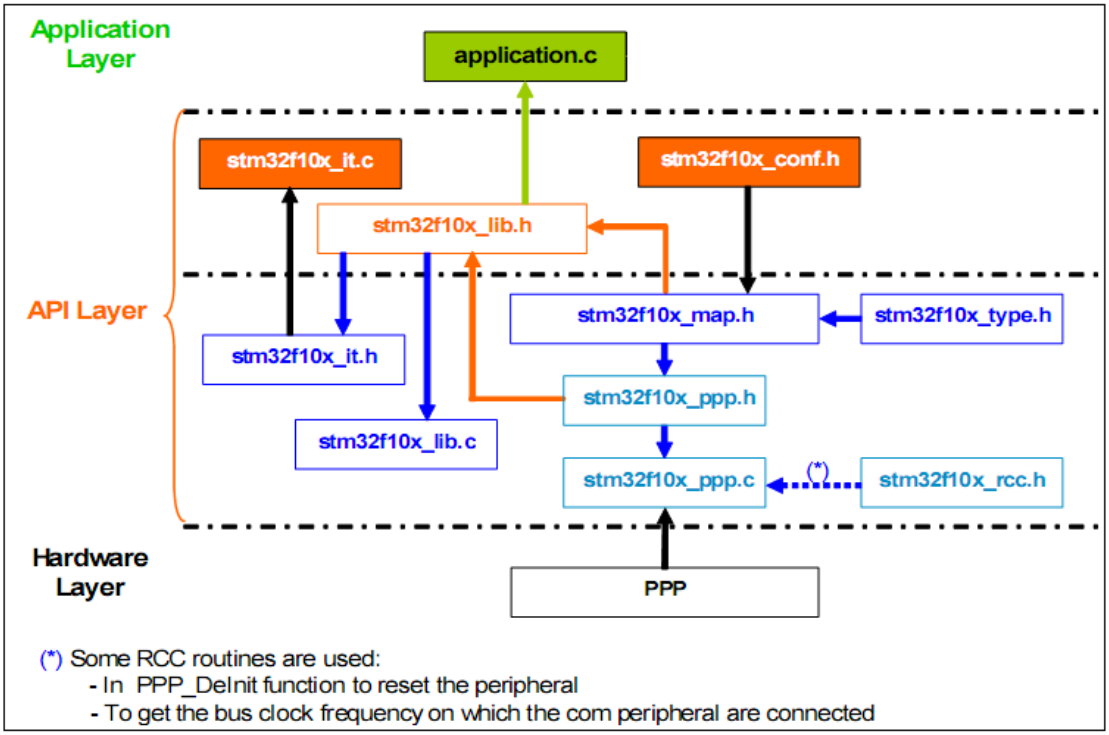
表 2. 固件库文件

文件名	描述

stm32f10x_conf.h	参数配置文件。 它要求用户在运行应用程序之前对它进行修改，定义需要与库进行交互的参数。 用户可以使用模板使能或者禁能外围模块，并且可以改变外部石英振荡器的数值。 该文件还可以在编译固件库之前决定使用调试或者发布模式。
main.c	主示例函数体。
stm32f10x_it.h	头文件，包括所有中断处理函数原型。
stm32f10x_it.c	外围模块中断处理函数文件。 用户可以引入在应用程序中需要使用的中断处理函数。如果有多个中断请求映射到同一个中断向量，该函数采用轮循外围中断标志的方式来确认中断源。这些函数的名字在固件库中提供。
stm32f10x_lib.h	头文件，包括所有外围模块的头文件。 这是唯一一个需要在用户应用程序中引用的文件，它作为库的接口。
stm32f10x_lib.c	调试模式初始化文件。 它包括变量指针的定义。每个指针指向相应外围模块的首地址和当调试模式使能时被调用的函数的定义。

	该函数初始化已定义的指针。
stm32f10x_map.h	该文件实现用于调试、释放模式的内存映射和寄存器物理地址定义。它提供给所有的外围模块。
stm32f10x_type.h	普通声明文件。 包括所有外围驱动程序使用的普通类型和常量。
stm32f10x_ppp.c	PPP 外围模块驱动程序源代码文件，用 C 语言编写。
stm32f10x_ppp.h	PPP 外围模块的头文件。包括 PPP 外围模块函数的定义和在这些函数中使用的变量的定义。
cortexm3_macro.h	cortexm3_macro.s 的头文件。
cortexm3_macro.s	专用的 Cortex-M3 指令的指令封装

Figure 2．固件库文件架构



2.3 外围模块的初始化和配置

本章节按部就班地描述如何初始化和配置外围模块。外围模块被写作 PPP。

1． 在主应用程序文件中，声明一个 **PPP_InitTypeDef** 结构，例如：

PPP_InitTypeDef PPP_InitStructure;

PPP_InitStructure 是一个位于数据存储区的有效变量。它可以初始化一个或者 多个 PPP 实例。

给 PPP_InitStructure 中的结构成员变量赋值。有以下两种方法：

按以下步骤配置整个结构体：

PPP_InitStructure.member1 = val1;

PPP_InitStructure.member2 = val2;

PPP_InitStructure.memberN = valN;

/* where N is the number of the structure members */

以上的初始化步骤可以合并为一行，以减少代码量：

PPP_InitTypeDef PPP_InitStructure = { val1, val2,.., valN}

仅配置一部分结构体成员：在这种情况下，用户可以修改 PPP_InitStructure 中已经被函数 **PPP_StructInit(..)**初始化的变量。这确保了其它 PPP_InitStructure 成员变量被初始化为一个恰当的值（多数情况下即他们的默认值）。

PPP_StructInit(&PPP_InitStructure);

PP_InitStructure.memberX = valX;

PPP_InitStructure.memberY = valY;

/*where X and Y are the members the user wants to configure*/

调用函数 **PPP_Init(..)**初始化 PPP 外围模块。

PPP_Init(PPP, &PPP_InitStructure);

PPP 外围模块被初始化，并且可以调用函数 **PPP_Cmd(..)**使能该外围模块。

PPP_Cmd(PPP, ENABLE);

可以通过调用一系列专用函数来使用这些外围模块。这些函数是外围模块的专用函数。要了解具体细节，请参阅章节 3: 外围固件概述。

注： 1 在配置外围模块之前，相应的时钟必须通过调用以下之一函数来使能：

RCC_AHBPeriphClockCmd(RCC_AHBPeriph_PPPx, ENABLE);

RCC_APB2PeriphClockCmd(RCC_APB2Periph_PPPx, ENABLE);

RCC_APB1PeriphClockCmd(RCC_APB1Periph_PPPx, ENABLE);

函数 PPP_DeInit(..)可以置所有 PPP 外围模块寄存器的值为默认值：

PPP_DeInit(PPP)

在配置外围模块完成后，修改外围模块的设置，可以按以下步骤：

PPP_InitStucture.memberX = valX;

PPP_InitStructure.memberY = valY; /* where X and Y are the only

members that user wants to modify*/

PPP_Init(PPP, &PPP_InitStructure);

2.4 Bit-Banding （位绑定）

Cortex-M3 内存映射包括两个 bit-band 内存区域。这些区域将存储器别名区域中的每个字，映射到存储器中 bit-band 区域的一位。在别名区域进行一个字的写操作，相当于在 bit-band 区域的目标位上进行一个“读-修改-写”操作。

所有的 STM32F10x 外围模块寄存器都映射到一个 bit-band 区域。这个特性有意识地使用在对单个位进行置位/清零操作的函数中，以减少代码长度，优化代码。

Section2.4.1 和 Section2.4.2 描述了在固件库中 bit-band 访问是如何使用的。

2.4.1 映射公式

映射公式显示了如何将别名区域中的字和 bit-band 区域中的目标位相联系。映射公式如下：

bit_word_offset = (byte_offset x 32) + (bit_number * 4)

bit_word_addr = bit_band_base + bit_word_offset

其中：

●bit_word_offset 是 bit_band 内存区域目标位的位置。

●bit_word_addr 是别名区域中字的地址，该字映射到目标位。

●bit_band_base 是别名区域的起始地址。

●byte_offset 是 bit_band 区域中包括了目标位的字节的字节号。

●bit_number 是目标位的位的位置（0-7）。

2.4.2 实现实例

以下实例显示了如何将 RCC_CR 寄存器的 PLLON[24]位映射到别名区：

```
/* Peripheral base address in the bit-band region */
#define PERIPH_BASE           ((u32)0x40000000)
/* Peripheral address in the alias region */
#define PERIPH_BB_BASE        ((u32)0x42000000)
/* ----- RCC registers bit address in the alias region ------ *
#define RCC_OFFSET            (RCC_BASE - PERIPH_BASE)
/* --- CR Register ---*/
/* Alias word address of PLLON bit */
#define CR_OFFSET             (RCC_OFFSET + 0x00)
#define PLLON_BitNumber       0x18
#define CR_PLLON_BB           (PERIPH_BB_BASE + (CR_OFFSET * 32
(PLLON_BitNumber * 4))
```

编写一个函数，该函数使能/禁止 PLL，如下：

```
...
#define CR_PLLON_Set          ((u32)0x01000000)
#define CR_PLLON_Reset        ((u32)0xFEFFFFFF)
...
void RCC_PLLCmd(FunctionalState NewState)
{
  if (NewState != DISABLE)
  { /* Enable PLL */
    RCC->CR |= CR_PLLON_Set;
  }
  else
  { /* Disable PLL */
    RCC->CR &= CR_PLLON_Reset;
  }
}
```

使用 bit-band 访问，这个函数可以按如下编写：

```
void RCC_PLLCmd(FunctionalState NewState)
{
```

```
*(vu32 *) CR_PLLON_BB = (u32)NewState;
}
```

2.5 运行时检测

该固件库通过检查所有库函数的输入值来完成运行时错误检测。运行时检测通过使用 **assert** 宏定义来实现。在有输入参数的所有库函数中，将使用该宏定义。这可以检查输入的参数值是否为一个允许量。

实例：函数 PWR_ClearFlag

stm32f10x_pwr.c:

```
void PWR_ClearFlag(u32 PWR_FLAG)
{
  /* Check the parameters */
  assert(IS_PWR_CLEAR_FLAG(PWR_FLAG));
PWR->CR |=  PWR_FLAG << 2;
}
```

stm32f10x_pwr.h:

```
/* PWR Flag */
#define PWR_FLAG_WU                 ((u32)0x00000001)
#define PWR_FLAG_SB                 ((u32)0x00000002)
#define PWR_FLAG_PVDO               ((u32)0x00000004)
```

```
#define IS_PWR_CLEAR_FLAG(FLAG) ((FLAG == PWR_FLAG_WU) || (FLAG ==
PWR_FLAG_SB))
```

如果传递给 **assert** 宏定义的表达式是错误的，那么函数 **assert_failed** 将会被调用，并且返回错误的源文件名和错误的行数。如果表达式正确，无返回值。

assert 宏定义在头文件 stm32f10x_conf.h 中实现：

```
/* Exported macro ------------------------------------------*/
#undef assert
#ifdef  DEBUG
/*******************************************************************
* Macro Name : assert
* Description: The assert macro is used to check function parameters
* It is used only if the library is compiled in DEBUG mode.
* Input : - expr: If expr is false, it calls assert_failed function
*   which reports the name of the source file and the source
* line number of the call that failed.
* If expr is true, it returns no value.
* Return : None
*******************************************************************/
#define assert(expr)((expr)?(void)0:assert_failed((u8 *)__FILE__,
__LINE__))
```

```
/* Exported functions ------------------------------------------- */
  void assert_failed(u8* file, u32 line);
#else
  #define assert(expr) ((void)0)
#endif /* DEBUG */
```

函数 **assert_failed** 在 main.c 或者任何用户 C 文件中实现：

```
#ifdef  DEBUG
/*******************************************************************
* Function Name  : assert_failed
* Description    : Reports the name of the source file and the
source line number
*                  where the assert error has occurred.
* Input          : - file: pointer to the source file name
*                  - line: assert error line source number
* Output         : None
* Return         : None
*******************************************************************/
void assert_failed(u8* file, u32 line)
{
  /* User can add his own implementation to report the file name and
```

```
line number,
        ex: printf("Wrong parameters value: file %s on line %d\r\n",
file, line) */
  /* Infinite loop */
  while (1)
  {
  }
}
#endif
```

注： 运行时检测，即该 assert 宏定义，只有当库在调试模式下编译时，才可以使用。

推荐在应用程序代码开发和调试阶段使用运行时检测，并在最终的应用程序中移去该功能，以达到最小化代码大小和运行速度的目的（因为它引入了不必要的开销）。

如果用户想要在最终应用程序中保留改功能，那么可以在调用库函数之前，重用库中的 **assert** 宏定义去测试参数值。

3 外围固件概述

本章节详细描述了固件库外围模块。相关函数也有详细描述，并且提供了使用实例。

函数将以以下形式进行描述：

表 3. 函数描述模式

函数名	外围模块函数的名字
函数原型	原型声明
功能描述	对函数是如何执行的简要说明
输入参数 {x}	输入参数的描述
输出参数 {x}	输出参数的描述
返回值	函数返回的值
前提条件	在调用在该函数前需要有的条件
调用的函数	其它被调用的库函数

4 模数转换器（ADC）

模数转换器包括一个具有多路复用信道选择器的输入端，实现近似转换。转换结果是 12 位。

ADC 固件库中使用的数据结构在 Section 4.1 中描述，固件库函数在 Section 4.2 中描述。

4.1 ADC寄存器结构

ADC 寄存器结构，ADC_TypeDef，定义在头文件 stm32f10x_map.h 中：

```
typedef struct
{
vu32 SR;
vu32 CR1;
vu32 CR2;
vu32 SMPR1;
vu32 SMPR2;
vu32 JOFR1;
vu32 JOFR2;
vu32 JOFR3;
vu32 JOFR4;
vu32 HTR;
vu32 LTR;
vu32 SQR1;
vu32 SQR2;
vu32 SQR3;
vu32 JSQR;
vu32 JDR1;
```

```
vu32 JDR2;

vu32 JDR3;

vu32 JDR4;

vu32 DR;

} ADC_TypeDef;
```

表 4 给出了 ADC 寄存器的列表：

表 4 ADC 寄存器

寄存器	描述
SR	ADC 状态寄存器
CR1	ADC 配置寄存器 1
CR2	ADC 配置寄存器 2
SMPR1	ADC 样本时间寄存器 1
SMPR2	ADC 样本时间寄存器 2
JOFR1	ADC 位移寄存器 1
JOFR2	ADC 位移寄存器 2
JOFR3	ADC 位移寄存器 3
JOFR4	ADC 位移寄存器 4
HTR	ADC 高压阀值寄存器
LTR	ADC 低压阀值寄存器

SQR1	ADC 用于常规组的序列选择器寄存器 1
SQR2	ADC 用于常规组的序列选择器寄存器 2
SQR3	ADC 用于常规组的序列选择器寄存器 3
JSQR	ADC 用于常规组的序列选择器寄存器
JDR1	ADC 数据转换注入组（Injected group）寄存器 1
JDR2	ADC 数据转换注入组（Injected group）寄存器 2
JDR3	ADC 数据转换注入组（Injected group）寄存器 3
JDR4	ADC 数据转换注入组（Injected group）寄存器 4
DR	ADC 常规组数据寄存器

这两个 ADC 外围模块在文件 stm32f10x_map 中声明：

```
...
#define PERIPH_BASE              ((u32)0x40000000)
#define APB1PERIPH_BASE          PERIPH_BASE
#define APB2PERIPH_BASE          (PERIPH_BASE + 0x10000)
#define AHBPERIPH_BASE           (PERIPH_BASE + 0x20000)
....
#define ADC1_BASE                (APB2PERIPH_BASE + 0x2400)
#define ADC2_BASE                (APB2PERIPH_BASE + 0x2800)
....
#ifndef DEBUG
```

```
...
#ifdef _ADC1
  #define ADC1                    ((ADC_TypeDef *) ADC1_BASE)
#endif /*_ADC1 */
#ifdef _ADC2
  #define ADC2                    ((ADC_TypeDef *) ADC2_BASE)
#endif /*_ADC2 */
...
#else    /* DEBUG */
...
#ifdef _ADC1
  EXT ADC_TypeDef                 *ADC1;
#endif /*_ADC1 */
#ifdef _ADC2
  EXT ADC_TypeDef                 *ADC2;
#endif /*_ADC2 */
...
#endif
```

当使用调试模式，_ADC1 和_ADC2 这两个指针在文件 stm32f10x_lib.c 中初始化：

```
...
#ifdef _ADC1
```

```
  ADC1 = (ADC_TypeDef *)  ADC1_BASE;
#endif /*_ADC1 */
#ifdef _ADC2
  ADC2 = (ADC_TypeDef *)  ADC2_BASE;
#endif /*_ADC2 */
...
```

为了访问 ADC 寄存器, _ADC, _ADC1 和_ADC2 必须在头文件 stm32f10x_conf.h 中定义：

```
...
#define _ADC
#define _ADC1
#define _ADC2
...
```

4.2 ADC库函数

表 5 列出了 ADC 固件库函数。

表 5 ADC 固件库函数

函数名	描述
ADC_DeInit	将 ADCx 外围模块寄存器复位为默认值。
ADC_Init	根据 ADC_InitStruct 中指定的参数初始化

	ADCx 外围模块。
ADC_StructInit	将 ADC_InitStruct 成员变量置为默认值。
ADC_Cmd	使能/禁止指定的 ADC 外围模块。
ADC_DMACmd	使能/禁止指定的 ADC DMA 请求。
ADC_ITConfig	使能/禁止指定的 ADC 中断。
ADC_ResetCalibration	复位选中的 ADC 校准寄存器。
ADC_GetResetCalibrationStatus	得到选中的 ADC 复位校准寄存器的状态。
ADC_StartCalibration	开始选中的 ADC 校准过程。
ADC_GetCalibrationStatus	得到选中的 ADC 校准状态。
ADC_SoftwareStartConvCmd	使能/禁止选中的 ADC 软件开始转换。
ADC_GetSoftwareStartConvStatus	得到选中的 ADC 软件开始转换状态。
ADC_DiscModeChannelCountConfig	配置选中的 ADC 常规组信道的非连续模式。
ADC_DiscModeCmd	使能/禁止指定的 ADC 常规组的非连续模式。
ADC_RegularChannelConfig	配置选中的 ADC 常规信道的相关等级（音序器(sequencer)中的等级）和采样时间。
ADC_ExternalTrigConvCmd	使能或禁止外部触发 ADCx 的转换。
ADC_GetConversionValue	返回常规信道的最后 ADCs 转换结果。
ADC_GetDualModeConversionValue	返回双重模式的最后的 ADCs 转换结果。
ADC_AutoInjectedConvCmd	在一个常规信道转换后使能或禁止被选的 ADC

	自动注入组的转换
ADC_InjectedDiscModeCmd	使能/禁止指定 ADC 注入组信道的非连续模式。
ADC_ExternalTrigInjectedConvConfig	配置 ADCx 注入信道转换的外部触发器。
ADC_ExternalTrigInjectedConvCmd	通过外部触发器使能/禁止 ADCx 注入信道转换
ADC_SoftwareStartInjectedConvCmd	使能/禁止选中的 ADC 开始注入信道转换。
ADC_GetSoftwareStartInjectedConvStatus	得到选中的 ADC 软件启动注入转换的状态。
ADC_InjectedChannelConfig	配置选中的 ADC 注入信道相应的音序器（sequencer）等级和采样时间。
ADC_InjectedSequencerLengthConfig	配置注入信道音序器（sequencer）的长度
ADC_SetInjectedOffset	设置注入信道转换值的位移。
ADC_GetInjectedConversionValue	返回选中的注入信道的 ADC 转换结果。
ADC_AnalogWatchdogCmd	使能/禁止单一/所有 常规信道或者注入信道的模拟看门狗。
ADC_AnalogWatchdogThresholdsConfig	配置模拟看门狗的高、低阀值。
ADC_AnalogWatchdogSingleChannelConfig	配置由模拟看门狗保护的单信道

ADC_TempSensorVrefintCmd	使能/禁止温度传感和 Vrefint 信道。
ADC_GetFlagStatus	检查指定的 ADC 标志是否置位
ADC_ClearFlag	清除 ADCx 的挂起标志
ADC_GetITStatus	检查指定 ADC 的中断是否发生
ADC_ClearITPendingBit	清除 ADCx 的中断挂起位。

4.2.1 函数ADC_DeInit

表 6 描述了函数 ADC_DeInit

表 6 函数 ADC_DeInit

函数名	ADC_DeInit
函数原型	void ADC_DeInit(ADC_TypeDef* ADCx)
功能描述	复位 ADCx 外围模块寄存器，使其为默认值。
输入参数	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块 ADC1 或者 ADC2.
输出参数	无
返回值	无
前提条件	无
调用的函数	RCC_APB2PeriphClockCmd().

实例：

```
/* Resets ADC2 */

ADC_DeInit(ADC2);
```

4.2.2 函数ADC_Init

表 7 描述了函数 ADC_Init

表 7 函数 ADC_Init

函数名	ADC_Init
函数原型	void ADC_Init(ADC_TypeDef* ADCx, ADC_InitTypeDef* ADC_InitStruct)
功能描述	根据 ADC_InitStruct 中指定的参数初始化 ADCx 外围模块。
输入参数 1	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块 ADC1 或者 ADC2.
输入参数 2	ADC_InitStruc :指向结构体 ADC_InitTypeDef 的指针，该结构包括了指定 ADC 外围模块的配置信息。请参阅章节 4.2.3：函数 ADC_StructInit 中有关 ADC_InitStruct 值的详细描述。
输出参数	无

返回值	无
前提条件	无
调用的函数	无

结构体 ADC_InitTypeDef

结构体 **ADC_InitTypeDef** 在头文件 stm32f10x_adc.h 中定义：

```
typedef struct
{
u32 ADC_Mode;
FunctionalState ADC_ScanConvMode;
FunctionalState ADC_ContinuousConvMode;
u32 ADC_ExternalTrigConv;
u32 ADC_DataAlign;
u8 ADC_NbrOfChannel;
} ADC_InitTypeDef;
```

ADC_Mode

DC_Mode 配置 ADC，使其工作在独立模式或者双重模式下。表 8 显示了各成员的值。

表 8 ADC_Mode 定义

ADC_Mode	描述

ADC_Mode_Independent	ADC1 和 ADC2 工作在独立模式
ADC_Mode_RegInjecSimult	ADC1 和 ADC2 工作在同步采样/保持 1 和 2 模式
ADC_Mode_RegSimult_AlterTrig	ADC1 和 ADC2 工作在同步采样/保持 2 和交替触发模式
ADC_Mode_InjecSimult_FastInterl	ADC1 和 ADC2 工作在同步采样/保持 1 和交叉存取 1 模式
ADC_Mode_InjecSimult_SlowInterl	ADC1 和 ADC2 工作在同步采样/保持 1 和交叉存取 2 模式
ADC_Mode_InjecSimult	ADC1 和 ADC2 工作在同步采样/保持 1 模式
ADC_Mode_RegSimult	ADC1 和 ADC2 工作在同步采样/保持 2 模式
ADC_Mode_FastInterl	ADC1 和 ADC2 工作在交叉存取 1 模式
ADC_Mode_SlowInterl	ADC1 和 ADC2 工作在交叉存取 2 模式
ADC_Mode_AlterTrig	ADC1 和 ADC2 工作在交替触发模式

ADC_ScanConvMode

ADC_ScanConvMode 指定转换是否工作在扫描（多信道）或单一（单信道）模式。这个成员变量可以被置为 ENABLE 或者 DISABLE。

ADC_ContinuousConvMode

ADC_ContinuousConvMode 指定转换是否工作在连续或单一模式。这个成员变量可以被置为 ENABLE 或者 DISABLE。

ADC_ExternalTrigConv

ADC_ExternalTrigConv 定义了外部触发器，该触发器可以用来开始常规信道的模数转换。表 9 显示了各成员的值。

表 9 ADC_ExternalTrigConv 的定义

ADC_ExternalTrigConv	描述
ADC_ExternalTrigConv_T1_CC1	定时器 1 捕获比较 1，用作外部触发转换。
ADC_ExternalTrigConv_T1_CC2	定时器 1 捕获比较 2，用作外部触发转换。
ADC_ExternalTrigConv_T1_CC3	定时器 1 捕获比较 3，用作外部触发转换。
ADC_ExternalTrigConv_T2_CC2	定时器 2 捕获比较 2，用作外部触发转换。
ADC_ExternalTrigConv_T3_TRGO	定时器 3 TRGO，用作外部触发转换。
ADC_ExternalTrigConv_T4_CC4	定时器 4 捕获比较 4，用作外部触发转换。
ADC_ExternalTrigConv_Ext_IT11	外部中断事件 11，用作外部触发转换。
ADC_ExternalTrigConv_None	由软件控制开始转换，而不是外部触发器。

ADC_DataAlign

ADC_DataAlign 指定 ADC 数据是左对齐还是右对齐。表 10 显示了各成员的值。

表 10 ADC_DataAlign 定义

ADC_DataAlign	描述
ADC_DataAlign_Right	ADC 数据右对齐
ADC_DataAlign_Left	ADC 数据左对齐

ADC_NbrOfChannel

ADC_NbrOfChannel 指定将要进行转换的 ADC 信道号（通过使用用于常规信道组的音序器）。该数字的范围是 1 – 16。

实例：

```
/* Initialize the ADC1 according to the ADC_InitStructure members */
ADC_InitTypeDef   ADC_InitStructure;
ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
ADC_InitStructure.ADC_ScanConvMode = Enable;
ADC_InitStructure.ADC_ContinuousConvMode = Disable;
ADC_InitStructure.ADC_ExternalTrigConv =
ADC_ExternalTrigConv_Ext_IT11;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_NbrOfChannel = 16;
ADC_Init(ADC1, &ADC_InitStructure);
```

注： 为了正确配置 ADC 信道转换，用户必须在调用 ADC_Init()后，再调用 ADC_ChannelConfig() 来为每个被使用的信道配置音序器级别和采样时间。

4.2.3 函数ADC_StructInit

表 11 描述了函数 ADC_StructInit

表 11 函数 ADC_StructInit

函数名	ADC_StructInit
函数原型	void ADC_StructInit(ADC_InitTypeDef* ADC_InitStruct)
功能描述	使用默认值初始化 ADC_InitStruct 各成员变量。
输入参数	ADC_InitStruct:指向结构体 ADC_InitTypeDef 的指针，用来初始化。
输出参数	无
返回值	无
前提条件	无
调用的函数	无

ADC_InitStruct 成员变量有如下的默认值：

表 12　ADC_InitStruct 的默认值

成员变量	默认值
ADC_Mode	ADC_Mode_Independent
ADC_ScanConvMode	DISABLE
ADC_ContinuousConvMode	DISABLE
ADC_ExternalTrigConv	ADC_ExternalTrigConv_T1_CC1
ADC_DataAlign	ADC_DataAlign_Right
ADC_NbrOfChannel	1

实例：

```
/* Initialize a ADC_InitTypeDef structure. */

ADC_InitTypeDef    ADC_InitStructure;

ADC_StructInit(&ADC_InitStructure);
```

4.2.4　函数ADC_Cmd

表 13 描述了函数 ADC_Cmd

表 13　函数 ADC_Cmd

函数名	ADC_Cmd

函数原型	void ADC_Cmd(ADC_TypeDef* ADCx, FunctionalState NewState)
功能描述	使能/禁止指定的 ADC 外围模块
输入参数 1	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块 ADC1 或者 ADC2.
输入参数 2	NewState: ADCx 外围模块的新状态 这个参数可以是：ENABLE 或 DISABLE。
输出参数	无
返回值	无
前提条件	无
调用的函数	无

实例：

```
/* Enable ADC1 */

ADC_Cmd(ADC1, ENABLE);
```

注： 在所有 ADC 配置函数后，都必须调用函数 ADC_Cmd。

4.2.5 函数ADC_DMACmd

表 14 描述了函数 ADC_DMACmd

表 14 函数 ADC_DMACmd

函数名	ADC_DMACmd
函数原型	ADC_DMACmd(ADC_TypeDef* ADCx, FunctionalState NewState)
功能描述	使能/禁止指定的 ADC DMA 请求
输入参数 1	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块 ADC1 或者 ADC2.
输入参数 2	NewState: ADC DMA 转移器的新状态 这个参数可以是：ENABLE 或 DISABLE。
输出参数	无
返回值	无
前提条件	无
调用的函数	无

实例：

```
/* Enable ADC2 DMA transfer */

ADC_DMACmd(ADC2, ENABLE);
```

4.2.6 函数ADC_ITConfig

表 15 描述了函数 ADC_ITConfig

表 15 函数 ADC_ITConfig

函数名	ADC_ITConfig
函数原型	void ADC_ITConfig(ADC_TypeDef* ADCx, u16 ADC_IT, FunctionalState NewState)
功能描述	使能/禁止指定的 ADC 中断
输入参数 1	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块 ADC1 或者 ADC2.
输入参数 2	ADC_IT: 指定 ADC 中断源是使能的或禁止的，关于该参数的可取值，请参考 ADC_IT 的部分。
输入参数 3	NewState: 指定的 ADC 中断的新状态 这个参数可以是：ENABLE 或 DISABLE。
输出参数	无
返回值	无
前提条件	无
调用的函数	无

ADC_IT

ADC_IT 用来使能或者禁止 ADC 中断。可以使用下述值的一个或者几个值的组合。

表 16 ADC_IT 定义

ADC_IT	描述
ADC_IT_EOC	EOC 中断屏蔽
ADC_IT_AWD	AWDOG 中断屏蔽
ADC_IT_JEOC	JEOC 中断屏蔽

实例：

```
/* Enable ADC2 EOC and AWDOG interrupts */

ADC_ITConfig(ADC2, ADC_IT_EOC | ADC_IT_AWD, ENABLE);
```

4.2.7 函数 ADC_ResetCalibration

表 17 描述了函数 ADC_ResetCalibration

表 17 函数 ADC_ResetCalibration

函数名	ADC_ResetCalibration
函数原型	void ADC_ResetCalibration(ADC_TypeDef* ADCx)
功能描述	重置选中的 ADC 校准寄存器
输入参数	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块 ADC1 或者 ADC2.

输出参数	无
返回值	无
前提条件	无
调用的函数	无

实例：

```
/* Reset the ADC1 Calibration registers */

ADC_ResetCalibration(ADC1);
```

4.2.8 函数ADC_GetResetCalibrationStatus

表 18 描述了函数 ADC_GetResetCalibration

表 18 函数 ADC_GetResetCalibration

函数名	ADC_GetResetCalibrationStatus
函数原型	FlagStatus ADC_GetResetCalibrationStatus(ADC_TypeDef* ADCx)
功能描述	得到选中的 ADC 重置校准寄存器的状态
输入参数	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块 ADC1 或者 ADC2.
输出参数	无

返回值	ADC 重置校准寄存器的新状态（SET 或 RESET）
前提条件	无
调用的函数	无

实例：

```
/* Get the ADC2 reset calibration registers status */
FlagStatus Status;
Status = ADC_GetResetCalibrationStatus(ADC2);
```

4.2.9 函数ADC_StartCalibration

表 19 描述了函数 ADC_StartCalibration

表 19 函数 ADC_StartCalibration

函数名	ADC_StartCalibration
函数原型	void ADC_StartCalibration(ADC_TypeDef* ADCx)
功能描述	开始选中的 ADC 校准过程
输入参数	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块 ADC1 或者 ADC2.
输出参数	无
返回值	无

前提条件	无
调用的函数	无

实例：

```
/* Start the ADC2 Calibration */

ADC_StartCalibration(ADC2);
```

4.2.10 函数ADC_GetCalibrationStatus

表 20 描述了函数 ADC_GetCalibrationStatus

表 20 函数 ADC_ GetCalibrationStatus

函数名	ADC_GetCalibrationStatus
函数原型	FlagStatus ADC_GetCalibrationStatus(ADC_TypeDef* ADCx)
功能描述	得到选中的 ADC 校准的状态
输入参数	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块 ADC1 或者 ADC2.
输出参数	无
返回值	ADC 校准的新状态（SET 或 RESET）
前提条件	无

调用的函数	无

实例：

```
/* Get the ADC2 calibration status */
FlagStatus Status;
Status = ADC_GetCalibrationStatus(ADC2);
```

4.2.11 函数ADC_SoftwareStartConvCmd

表 21 描述了函数 ADC_SoftwareStartConvCmd

表 21 函数 ADC_SoftwareStartConvCmd

函数名	ADC_SoftwareStartConvCmd
函数原型	void ADC_SoftwareStartConvCmd(ADC_TypeDef* ADCx, FunctionalState NewState)
功能描述	使能/禁止选中的 ADC 由软件控制开始转换。
输入参数 1	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块 ADC1 或者 ADC2.
输入参数 2	NewState: 选中的由软件发出开始信号的 ADC 的新状态

	这个参数可以是：ENABLE 或 DISABLE。
输出参数	无
返回值	无
前提条件	无
调用的函数	无

实例：

```
/* Start by software the ADC1 Conversion */

ADC_SoftwareStartConvCmd(ADC1, ENABLE);
```

4.2.12 函数ADC_GetSoftwareStartConvStatus

表 22 描述了函数 ADC_GetSoftwareStartConvStatus

表 22 函数 ADC_GetSoftwareStartConvStatus

函数名	ADC_GetSoftwareStartConvStatus
函数原型	FlagStatus ADC_GetSoftwareStartConvStatus(ADC_TypeDef* ADCx)
功能描述	获得选中 ADC 软件 开始转换状态。
输入参数	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块

	ADC1 或者 ADC2.
输出参数	无
返回值	ADC 软件开始转换的新状态（SET 或 RESET）
前提条件	无
调用的函数	无

实例：

```
/* Get the ADC1 conversion start bit */

FlagStatus Status;

Status = ADC_GetSoftwareStartConvStatus(ADC1);
```

4.2.13 函数ADC_DiscModeChannelCountConfig

表 23 描述了函数 ADC_DiscModeChannelCountConfig

表 23 函数 ADC_DiscModeChannelCountConfig

函数名	ADC_DiscModeChannelCountConfig
函数原型	void ADC_DiscModeChannelCountConfig(ADC_TypeDef* ADCx, u8 Number)
功能描述	配置选中的 ADC 常规组为非连续模式。

输入参数 1	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块 ADC1 或者 ADC2。
输入参数 2	Number: 非连续模式下常规信道计数值。 该值范围为 1 - 8。
输出参数	无
返回值	无
前提条件	无
调用的函数	无

实例：

```
/* Set the discontinuous mode channel count to 2 for ADC1 */
ADC_DiscModeChannelCountConfig(ADC1, 2);
```

4.2.14 函数ADC_DiscModeCmd

表 24 描述了函数 ADC_DiscModeCmd

表 24 函数 ADC_DiscModeCmd

函数名	ADC_DiscModeCmd
函数原型	void ADC_DiscModeCmd(ADC_TypeDef* ADCx, FunctionalState NewState)

功能描述	使能/禁止指定的 ADC 常规组信道的非连续模式。
输入参数 1	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块 ADC1 或者 ADC2。
输入参数 2	NewState: ADC 常规组信道下非连续模式的新状态 这个参数可以是：ENABLE 或 DISABLE。
输出参数	无
返回值	无
前提条件	无
调用的函数	无

实例：

```
/* Disable the discontinuous mode for ADC1 regular group channel */
ADC_DiscModeCmd(ADC1, ENABLE);
```

4.2.15 函数ADC_RegularChannelConfig

表 25 描述了函数 ADC_RegularChannelConfig

表 25 函数 ADC_RegularChannelConfig

函数名	ADC_RegularChannelConfig
函数原型	void ADC_RegularChannelConfig(ADC_TypeDef*

	ADCx, u8 ADC_Channel, u8 Rank, u8 ADC_SampleTime)
功能描述	为选中的 ADC 常规组信道配置相关的音序器（sequencer）等级和采样时间。
输入参数 1	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块 ADC1 或者 ADC2。
输入参数 2	ADC_Channel:需要配置的 ADC 信道 请参阅: ADC_Channel 的部分以获得该参数可取值的更多信息
输入参数 3	Rank:常规组音序器（sequencer）的等级。 参数范围是 1 - 16。
输入参数 4	ADC_SampleTime:将要为所选的信道设置的采样时间 请参阅: ADC_Channel 的部分以获得该参数可取值的更多信息
输出参数	无
返回值	无
前提条件	无
调用的函数	无

ADC_Channel

ADC_Channel 参数指定了 ADC 信道 ,该信道通过调用函数 ADC_RegularChannelConfig 来设

置。表 26 显示了 ADC_Channe 可能的取值：

表 26 ADC_Channel 取值

ADC_Channel	描述
ADC_Channel_0	ADC 信道 0 被选择
ADC_Channel_1	ADC 信道 1 被选择
ADC_Channel_2	ADC 信道 2 被选择
ADC_Channel_3	ADC 信道 3 被选择
ADC_Channel_4	ADC 信道 4 被选择
ADC_Channel_5	ADC 信道 5 被选择
ADC_Channel_6	ADC 信道 6 被选择
ADC_Channel_7	ADC 信道 7 被选择
ADC_Channel_8	ADC 信道 8 被选择
ADC_Channel_9	ADC 信道 9 被选择
ADC_Channel_10	ADC 信道 10 被选择
ADC_Channel_11	ADC 信道 11 被选择
ADC_Channel_12	ADC 信道 12 被选择
ADC_Channel_13	ADC 信道 13 被选择
ADC_Channel_14	ADC 信道 14 被选择
ADC_Channel_15	ADC 信道 15 被选择
ADC_Channel_16	ADC 信道 16 被选择

ADC_Channel_17	ADC 信道 17 被选择

ADC_SampleTime

该参数指定了选中信道的 ADC 采样时间。表 27 给出了 ADC_SampleTime 可能的取值。

表 27 ADC_SampleTime.取值

ADC_SampleTime	描述
ADC_SampleTime_1Cycles5	采样时间等于 1.5 个周期
ADC_SampleTime_7Cycles5	采样时间等于 7.5 个周期
ADC_SampleTime_13Cycles5	采样时间等于 13.5 个周期
ADC_SampleTime_28Cycles5	采样时间等于 28.5 个周期
ADC_SampleTime_41Cycles5	采样时间等于 41.5 个周期
ADC_SampleTime_55Cycles5	采样时间等于 55.5 个周期
ADC_SampleTime_71Cycles5	采样时间等于 71.5 个周期
ADC_SampleTime_239Cycles5	采样时间等于 239.5 个周期

实例：

```
/* Configures ADC1 Channel2 as: first converted channel with an 7.5

cycles sample time */

ADC_RegularChannelConfig(ADC1, ADC_Channel_2, 1,

ADC_SampleTime_7Cycles5);
```

```
/* Configures ADC1 Channel8 as: second converted channel with an 1.5
cycles sample time */
ADC_RegularChannelConfig(ADC1, ADC_Channel_8, 2,
ADC_SampleTime_1Cycles5);
```

4.2.16 函数ADC_ExternalTrigConvCmd

表 28 描述了函数 ADC_ExternalTrigConvCmd

表 28 函数 ADC_ExternalTrigConvCmd

函数名	ADC_ExternalTrigConvCmd
函数原型	void ADC_ExternalTrigConvCmd(ADC_TypeDef* ADCx, FunctionalState NewState)
功能描述	使能或禁止外部触发 ADCx 转换。
输入参数 1	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块 ADC1 或者 ADC2。
输入参数 2	NewState:选中的由外部触发转换的 ADC 的新状态 这个参数可以是：ENABLE 或 DISABLE。
输出参数	无
返回值	无

前提条件	无
调用的函数	无

实例：

/*Enable the start of conversion for ADC1 through exteral trigger */

ADC_ExternalTrigConvCmd(ADC1, ENABLE);

4.2.17 函数ADC_GetConversionValue

表 29 描述了函数 ADC_GetConversionValue

表 29 函数 ADC_GetConversionValue

函数名	ADC_GetConversionValue
函数原型	u16 ADC_GetConversionValue(ADC_TypeDef* ADCx)
功能描述	返回常规信道最后的 ADCx 转换结果。
输入参数	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块 ADC1 或者 ADC2。
输出参数	无
返回值	数据转换结果
前提条件	无

调用的函数	无

实例：

/*Returns the ADC1 Master data value of the last converted channel*/

u16 DataValue;

DataValue = ADC_GetConversionValue(ADC1);

4.2.18 函数ADC_GetDualModeConversionValue

表 30 描述了函数 ADC_GetDualModeConversionValue

表 30 函数 ADC_GetDualModeConversionValue

函数名	ADC_GetDualModeConversionValue
函数原型	u32 ADC_GetDualModeConversionValue()
功能描述	返回双重模式下的 ADC 的最后转换结果。
输入参数	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块 ADC1 或者 ADC2。
输出参数	无
返回值	数据转换结果
前提条件	无
调 用 的 函	无

数	

实例：

```
/* Returns the ADC1 and ADC2 last converted values*/
u32 DataValue;
DataValue = ADC_GetDualModeConversionValue();
```

4.2.19 函数ADC_AutoInjectedConvCmd

表 31 描述了函数 ADC_AutoInjectedConvCmd

表 31 函数 ADC_AutoInjectedConvCmd

函数名	ADC_AutoInjectedConvCmd
函数原型	void ADC_AutoInjectedConvCmd(ADC_TypeDef* ADCx, FunctionalState NewState)
功能描述	使能/禁止在常规组转换后，被选的 ADC 进行自动注入组转换。
输入参数 1	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块 ADC1 或者 ADC2。
输入参数 2	NewState: 选中的进行自动注入转换的 ADC 新状态

	这个参数可以是：ENABLE 或 DISABLE。
输出参数	无
返回值	无
前提条件	无
调用的函数	无

实例：

```
/* Enable the auto injected conversion for ADC2 */

ADC_AutoInjectedConvCmd(ADC2, ENABLE);
```

4.2.20 函数ADC_InjectedDiscModeCmd

表 32 描述了函数 ADC_InjectedDiscModeCmd

表 32 函数 ADC_AutoInjectedConvCmd

函数名	ADC_InjectedDiscModeCmd
函数原型	void ADC_InjectedDiscModeCmd(ADC_TypeDef* ADCx, FunctionalState NewState)
功能描述	使能/禁止指定 ADC 注入组信道的非连续模式。
输入参数 1	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块

	ADC1 或者 ADC2。
输入参数 2	NewState: 被选中的，注入组信道上为非连续模式的 ADC 的新状态 这个参数可以是：ENABLE 或 DISABLE。
输出参数	无
返回值	无
前提条件	无
调用的函数	无

实例：

```
/* Enable the injected discontinuous mode for ADC2 */
ADC_InjectedDiscModeCmd(ADC2, ENABLE);
```

4.2.21 函数ADC_ExternalTrigInjectedConvConfig

表 33 描述了函数 ADC_ExternalTrigInjectedConvConfig

表 33 函数 ADC_ExternalTrigInjectedConvConfig

函数名	ADC_ExternalTrigInjectedConvConfig
函数原型	void ADC_ExternalTrigInjectedConvConfig(ADC_TypeDef*

	ADCx, u32 ADC_ExternalTrigConv)
功能描述	为注入信道转换配置 ADCx 外部触发
输入参数 1	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块 ADC1 或者 ADC2。
输入参数 2	ADC_ExternalTrigInjecConv: 开始注入转换的 ADC 触发器 请参阅章节：ADC_ExternalTrigInjecConv 中关于参数可取值的说明。
输出参数	无
返回值	无
前提条件	无
调用的函数	无

ADC_ExternalTrigInjecConv

该参数指定 ADC 触发器，该触发器用来启动注入转换。表 34 给出了 ADC_ExternalTrigInjecConv 取值的列表。

表 34 ADC_ExternalTrigInjecConv 取值

ADC_ExternalTrigInjecConv	描述
ADC_ExternalTrigInjecConv_T1_TRGO	定时器 1 TRGO 事件作为注入转换的外部触发器

ADC_ExternalTrigInjecConv_T1_CC4	定时器 1 捕获比较 4 作为注入转换的外部触发器
ADC_ExternalTrigInjecConv_T2_TRGO	定时器 2 TRGO 事件作为注入转换的外部触发器
ADC_ExternalTrigInjecConv_T2_CC1	定时器 2 捕获比较 1 作为注入转换的外部触发器
ADC_ExternalTrigInjecConv_T3_CC4	定时器 3 捕获比较 4 作为注入转换的外部触发器
ADC_ExternalTrigInjecConv_T4_TRGO	定时器 4 TRGO 事件作为注入转换的外部触发器
ADC_ExternalTrigInjecConv_Ext_IT15	外部中断事件 15 注入转换的外部触发器
ADC_ExternalTrigInjecConv_None	注入转换由软件启动，而不是由外部触发器启动

实例：

```
/* Set ADC1 injected external trigger conversion start to Timer1
capture compare4 */
ADC_ExternalTrigInjectedConvConfig(ADC1,
ADC_ExternalTrigConv_T1_CC4);
```

4.2.22 函数ADC_ExternalTrigInjectedConvCmd

表 35 描述了函数 ADC_ExternalTrigInjectedConvCmd

表 35 函数 ADC_ExternalTrigInjectedConvCmd

函数名	ADC_ExternalTrigInjectedConvCmd
函数原型	void ADC_InjectedDiscModeCmd(ADC_TypeDef* ADCx, FunctionalState NewState)
功能描述	通过外部触发器使能/禁止 ADCx 注入信道转换
输入参数 1	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块 ADC1 或者 ADC2。
输入参数 2	NewState: 选中的用于启动注入转换的 ADC 外部触发器。 这个参数可以是：ENABLE 或 DISABLE。
输出参数	无
返回值	无
前提条件	无
调用的函数	无

实例：

/* Enable the start of injected conversion for ADC1 through exteral

```
trigger */
ADC_ExternalTrigInjectedConvCmd(ADC1, ENABLE);
```

4.2.23 函数ADC_SoftwareStartInjectedConvCmd

表 36 描述了函数 ADC_SoftwareStartInjectedConvCmd

表 36 函数 ADC_SoftwareStartInjectedConvCmd

函数名	ADC_SoftwareStartInjectedConvCmd
函数原型	void ADC_SoftwareStartInjectedConvCmd(ADC_TypeDef* ADCx, FunctionalState NewState)
功能描述	使能/禁止选中的 ADC 开始注入信道转换。
输入参数 1	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块 ADC1 或者 ADC2。
输入参数 2	NewState: 被选中的用于启动注入转换的 ADC 软件的新状态。 这个参数可以是：ENABLE 或 DISABLE。
输出参数	无
返回值	无
前提条件	无

调用的函数	无

实例：

/* Start by software the ADC2 Conversion */

ADC_SoftwareStartInjectedConvCmd(ADC2, ENABLE);

4.2.24 函数ADC_GetSoftwareStartInjectedConvStatus

表 37 描述了函数 ADC_GetSoftwareStartInjectedConvStatus

表 37 函数 ADC_GetSoftwareStartInjectedConvStatus

函数名	ADC_GetSoftwareStartInjectedConvStatus
函数原型	FlagStatus ADC_GetSoftwareStartInjectedConvStatus (ADC_TypeDef* ADCx)
功能描述	获得选中的由软件启动注入转换的 ADC 的状态。
输入参数	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块 ADC1 或者 ADC2。
输出参数	无
返回值	由软件启动注入转换的 ADC 新状态（SET 或者 RESET）
前提条件	无

调用的函数	无

实例：

```
/* Get the ADC1 injected conversion start bit */

FlagStatus Status;

Status = ADC_GetSoftwareStartInjectedConvStatus(ADC1);
```

4.2.25 函数ADC_InjectedChannelConfig

表 38 描述了函数 ADC_InjectedChannelConfig

表 38 函数 ADC_InjectedChannelConfig

函数名	ADC_InjectedChannelConfig
函数原型	void ADC_InjectedChannelConfig(ADC_TypeDef* ADCx, u8 ADC_Channel, u8 Rank, u8 ADC_SampleTime)
功能描述	配置选中的 ADC 注入信道相应的音序器（sequencer）等级和采样时间。
输入参数 1	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块 ADC1 或者 ADC2。
输入参数 2	ADC_Channel: 需要配置的 ADC 信道

	请参阅章节： ADC_Channel 中参数的可取取值
输入参数 3	Rank: 注入组音序器（sequencer）的等级
输入参数 4	ADC_SampleTime: 将要为选中信道设置的采样时间值
输出参数	无
返回值	无
前提条件	在指定总的注入信道数目之前必须先调用 ADC_InjectedSequencerLengthConfig。当每个注入信道等级小于 4 时，这是十分有必要的。
调用的函数	无

ADC_Channel

ADC_Channel 指定了需要进行配置的 ADC 信道。请参阅表 26 中 ADC_Channel 可能的取值。

ADC_SampleTime

ADC_SampleTime 指定了选中信道的 ADC 采样时间。请参阅表 27 中 ADC_SampleTime 可能的取值。

实例：

```
/* Configures ADC1 Channel12 as: second converted channel with an
28.5 cycles sample time */
```

```
ADC_InjectedChannelConfig(ADC1, ADC_Channel_12, 2,
ADC_SampleTime_28Cycles5);
/* Configures ADC2 Channel4 as: eleven converted channel with an
71.5 cycles sample time */
ADC_InjectedChannelConfig(ADC2, ADC_Channel_4, 11,
ADC_SampleTime_71Cycles5);
```

4.2.26 函数ADC_InjectedSequencerLengthConfig

表 39 描述了函数 ADC_InjectedSequencerLengthConfig

表 39 函数 ADC_InjectedSequencerLengthConfig

函数名	ADC_InjectedSequencerLengthConfig
函数原型	void ADC_InjectedSequencerLengthConfig(ADC_TypeDef* ADCx, u8 Length)
功能描述	配置注入信道音序器（sequencer）的长度
输入参数 1	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块 ADC1 或者 ADC2。
输入参数 2	Length: 音序器（sequencer）的长度 该参数的范围是 1 - 4

输出参数	无
返回值	无
前提条件	无
调用的函数	无

实例：

```
/* Set the ADC1 Sequencer length to 4 channels */

ADC_InjectedSequencerLengthConfig(ADC1, 4);
```

4.2.27 函数ADC_SetInjectedOffset

表 40 描述了函数 ADC_SetInjectedOffset

表 40 函数 ADC_SetInjectedOffset

函数名	ADC_SetInjectedOffset
函数原型	void ADC_SetInjectedOffset(ADC_TypeDef* ADCx, u8 ADC_InjectedChannel, u16 Offset)
功能描述	设置注入信道转换的偏移值。
输入参数 1	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块 ADC1 或者 ADC2。

输入参数 2	ADC_InjectedChannel: 需要设置偏移值的 ADC 注入信道 请参阅章节： ADC_InjectedChannel 中关于参数可取值的详细信息
输入参数 3	Offset: 选中的 ADC 注入信道的偏移值。 该参数是一个 12 位值
输出参数	无
返回值	无
前提条件	无
调用的函数	无

ADC_InjectedChannel

ADC_InjectedChannel 指定了需要设置偏移值的 ADC 注入信道。表 41 给出了参数的取值。

表 41 ADC_InjectedChannel 取值

ADC_InjectedChannel	描述
ADC_InjectedChannel_1	选择注入信道 1
ADC_InjectedChannel_2	选择注入信道 2
ADC_InjectedChannel_3	选择注入信道 3
ADC_InjectedChannel_4	选择注入信道 4

实例：

```
/* Set the offset 0x100 for the 3rd injected Channel of ADC1 */

ADC_SetInjectedOffset(ADC1, ADC_InjectedChannel_3, 0x100);
```

4.2.28 函数ADC_GetInjectedConversionValue

表 42 描述了函数 ADC_GetInjectedConversionValue

表 42 函数 ADC_GetInjectedConversionValue

函数名	ADC_GetInjectedConversionValue
函数原型	u16 ADC_GetInjectedConversionValue(ADC_TypeDef* ADCx, u8 ADC_InjectedChannel)
功能描述	返回选中的注入信道的 ADC 转换结果。
输入参数 1	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块 ADC1 或者 ADC2。
输入参数 2	ADC_InjectedChannel: 需要设置偏移值的 ADC 注入信道 请参阅章节： ADC_InjectedChannel 中关于参数取值的详细信息
输出参数	无

返回值	数据转换结果
前提条件	无
调用的函数	无

实例：

```
/* Return the ADC1 injected channel1 converted data value */
u16 InjectedDataValue;
InjectedDataValue = ADC_GetInjectedConversionValue(ADC1,
ADC_InjectedChannel_1);
```

4.2.29 函数ADC_AnalogWatchdogCmd

表 42 描述了函数 ADC_AnalogWatchdogCmd

表 42 函数 ADC_AnalogWatchdogCmd

函数名	ADC_AnalogWatchdogCmd
函数原型	void ADC_AnalogWatchdogCmd(ADC_TypeDef* ADCx, u32 ADC_AnalogWatchdog)
功能描述	使能/禁止一个/所有常规信道或者注入信道的模拟看门

	狗。
输入参数 1	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块 ADC1 或者 ADC2。
输入参数 2	ADC_AnalogWatchdog: ADC 模拟看门狗配置 请参阅章节： ADC_AnalogWatchdog 中关于该参数可取值的详细信息
输出参数	无
返回值	无
前提条件	无
调用的函数	无

ADC_AnalogWatchdog

ADC_AnalogWatchdog 指定 ADC 模拟看门狗的配置。表 44 给出了该参数可能的取值。

表 44 ADC_AnalogWatchdog 取值

ADC_AnalogWatchdog	描述
ADC_AnalogWatchdog_SingleRegEnable	单常规信道上的模拟看门狗
ADC_AnalogWatchdog_SingleInjecEnable	单注入信道上的模拟看门狗
ADC_AnalogWatchdog_SingleRegorInjecEnable	单常规信道或者注入信道的模拟看门狗

ADC_AnalogWatchdog_AllRegEnable	所有常规信道上的模拟看门狗
ADC_AnalogWatchdog_AllInjecEnable	所有注入信道上的模拟看门狗
ADC_AnalogWatchdog_AllRegAllInjecEnable	所有常规信道和注入信道上的模拟看门狗
ADC_AnalogWatchdog_None	没有信道需要模拟看门狗监视

实例：

```
/* Configue the Analog watchdog on all regular and injected channels
of ADC2 */
ADC_AnalogWatchdogCmd(ADC2,
ADC_AnalogWatchdog_AllRegAllInjecEnable);
```

4.2.30 函数ADC_AnalogWatchdogThresholdsConfig

表 45 描述了函数 ADC_AnalogWatchdogThresholdsConfig

表 45 函数 ADC_AnalogWatchdogThresholdsConfig

函数名	ADC_AnalogWatchdogThresholdsConfig
函数原型	void ADC_AnalogWatchdogThresholdsConfig(ADC_TypeDef * ADCx, u16 HighThreshold, u16 LowThreshold)

功能描述	配置模拟看门狗的高、低阀值。
输入参数 1	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块 ADC1 或者 ADC2。
输入参数 2	HighThreshold: ADC 模拟看门狗最高阀值。 该参数是一个 12 位的数值
输入参数 3	LowThreshold: ADC 模拟看门狗最低阀值。 该参数是一个 12 位的数值
输出参数	无
返回值	无
前提条件	无
调用的函数	无

实例：

```
/* Configue the Analog watchdog High and Low thresholds for ADC1 */
ADC_AnalogWatchdogThresholdsConfig(ADC1, 0x400, 0x100);
```

4.2.31 函数ADC_AnalogWatchdogSingleChannelConfig

表 46 描述了函数 ADC_AnalogWatchdogSingleChannelConfig

表 46 函数 ADC_AnalogWatchdogSingleChannelConfig

函数名	ADC_AnalogWatchdogSingleChannelConfig
函数原型	void ADC_AnalogWatchdogSingleChannelConfig (ADC_TypeDef* ADCx, u8 ADC_Channel)
功能描述	配置监视单信道的模拟看门狗
输入参数 1	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块 ADC1 或者 ADC2。
输入参数 2	ADC_Channel: 将要为其配置模拟看门狗的信道 请参阅 Section: ADC_Channel 中参数可取值的详细信息
输出参数	无
返回值	无
前提条件	无
调用的函数	无

实例：

```
/* Configue the Analog watchdog on Channel1 of ADC1 */
ADC_AnalogWatchdogSingleChannelConfig(ADC1, ADC_Channel_1);
```

4.2.32 函数ADC_TempSensorVrefintCmd

表 47 描述了函数 ADC_TempSensorVrefintCmd

表 47 函数 ADC_TempSensorVrefintCmd

函数名	ADC_TempSensorVrefintCmd
函数原型	Void ADC_TempSensorVrefintCmd(FunctionalState NewState)
功能描述	使能/禁止温度传感和 Vrefint 信道。
输入参数	NewState: 温度传感和 Vrefint 信道的新状态 该参数可以取两个值：ENABLE 或 DISABLE。
输出参数	无
返回值	无
前提条件	无
调用的函数	无

实例：

```
/* Enable the temperature sensor and vref internal channel */
ADC_TempSensorVrefintCmd(ENABLE);
```

4.2.33 函数ADC_GetFlagStatus

表 48 描述了函数 ADC_GetFlagStatus

表 48 函数 ADC_GetFlagStatus

函数名	ADC_GetFlagStatus
函数原型	FlagStatus ADC_GetFlagStatus(ADC_TypeDef* ADCx, u8 ADC_FLAG)
功能描述	检查指定的 ADC 标志是否置位
输入参数 1	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块 ADC1 或者 ADC2。
输入参数 2	ADC_FLAG: 指定需要检查的标志 请参阅章节： ADC_FLAG 中关于参数取值的详细信息。
输出参数	无
返回值	ADC_FLAG 的新状态 (SET 或 RESET)
前提条件	无
调用的函数	无

ADC_FLAG

Talbe 49 给出了 ADC_FLAG 的取值。

表 49 ADC_FLAG 取值

ADC_FLAG	描述
ADC_FLAG_AWD	模拟看门狗标志

ADC_FLAG_EOC	转换结束标志
ADC_FLAG_JEOC	注入组转换结束标志
ADC_FLAG_JSTRT	注入组转换开始标志
ADC_FLAG_STRT	常规组转换开始标志

实例：

```
/* Test if the ADC1 EOC flag is set or not */

FlagStatus Status;

Status = ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC);
```

4.2.34 函数ADC_ClearFlag

表 50 描述了函数 ADC_ClearFlag

表 50 函数 ADC_ClearFlag

函数名	ADC_ClearFlag
函数原型	void ADC_ClearFlag(ADC_TypeDef* ADCx, u8 ADC_FLAG)
功能描述	清除 ADCx 的挂起标志
输入参数 1	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块 ADC1 或者 ADC2。

输入参数 2	ADC_FLAG: 需要清除的标志。使用“|”可以使得一个或多个标志可以同时被清除。 请参阅章节：ADC_FLAG 中参数可取值的详细信息。
输出参数	无
返回值	无
前提条件	无
调用的函数	无

实例：

```
/* Clear the ADC2 STRT pending flag */

ADC_ClearFlag(ADC2, ADC_FLAG_STRT);
```

4.2.35 函数ADC_GetITStatus

表 51 描述了函数 ADC_GetITStatus

表 51 函数 ADC_GetITStatus

函数名	ADC_GetITStatus
函数原型	ITStatus ADC_GetITStatus(ADC_TypeDef* ADCx, u16 ADC_IT)
功能描述	检查指定 ADC 的中断是否发生

输入参数 1	ADCx: 其中 x 可以是 1 或 2，用来选择 ADC 外围模块 ADC1 或者 ADC2。
输入参数 2	ADC_IT: 需要检查的 ADC 中断源 请参阅章节：ADC_IT 中参数取值的详细信息。
输出参数	无
返回值	ADC_IT 的新状态（SET 或 RESET）
前提条件	无
调用的函数	无

实例：

```
/* Test if the ADC1 AWD interrupt has occurred or not */
ITStatus Status;
Status = ADC_GetITStatus(ADC1, ADC_IT_AWD);
```

4.2.36 函数ADC_ClearITPendingBit

表 52 描述了函数 ADC_ClearITPendingBit

表 52 函数 ADC_ClearITPendingBit

函数名	ADC_ClearITPendingBit
函数原型	void ADC_ClearITPendingBit(ADC_TypeDef* ADCx, u16

	ADC_IT)
功能描述	清除 ADCx 的中断挂起位。
输入参数 1	ADCx: 其中 x 可以是 1 或 2 ,用来选择 ADC 外围模块 ADC1 或者 ADC2。
输入参数 2	ADC_IT: 需要清除的中断挂起位 请参阅章节：ADC_IT 中参数可取值的详细信息。
输出参数	无
返回值	无
前提条件	无
调用的函数	无

实例：

```
/* Clear the ADC2 JEOC interrupt pending bit */
ADC_ClearITPendingBit(ADC2, ADC_IT_JEOC);
```

5 备份寄存器（BKP）

备份寄存器是共有 10 个 16 位的寄存器，可以存放 20 字节的用户应用数据。它们在备份领域中实现，当 V_{DD} 关掉时，仍旧由 V_{BAT} 驱动。

BKP 寄存器也用来管理窜改检测特性和 RTC 校准。

章节 5.1: BKP 寄存器结构描述了 BKP 固件库中使用的数据结构。章节 5.2: 固件库函数给出了库函数描述。

5.1 BKP寄存器结构

BKP 寄存器结构，BKP_TypeDef，定义在头文件 stm32f10x_map.h 中，如下：

```
typedef struct
{
  u32 RESERVED0;
  vu16 DR1;
  u16  RESERVED1;
  vu16 DR2;
  u16  RESERVED2;
  vu16 DR3;
  u16  RESERVED3;
  vu16 DR4;
  u16  RESERVED4;
  vu16 DR5;
  u16  RESERVED5;
  vu16 DR6;
  u16  RESERVED6;
```

```
    vu16 DR7;
    u16  RESERVED7;
    vu16 DR8;
    u16  RESERVED8;
    vu16 DR9;
    u16  RESERVED9;
    vu16 DR10;
    u16  RESERVED10;
    vu16 RTCCR;
    u16  RESERVED11;
    vu16 CR;
    u16  RESERVED12;
    vu16 CSR;
    u16  RESERVED13;
} BKP_TypeDef;
```

表 53 给出了 BKP 寄存器列表

表 53 BKP 寄存器列表

寄存器	描述
DR 1-10	数据备份寄存器 1 - 10
RTCCR	RTC 时钟校准寄存器

CR	备份控制寄存器
CSR	备份控制状态寄存器

BKP 外围模块在头文件 stm32f10x_map.h 中声明：

```
#define PERIPH_BASE              ((u32)0x40000000)
#define APB1PERIPH_BASE          PERIPH_BASE
#define APB2PERIPH_BASE          (PERIPH_BASE + 0x10000)
#define AHBPERIPH_BASE           (PERIPH_BASE + 0x20000)
#define BKP_BASE                 (APB1PERIPH_BASE + 0x6C00)
#ifndef DEBUG
...
#ifdef _BKP
  #define BKP                    ((BKP_TypeDef *) BKP_BASE)
#endif /*_BKP */
...
#else     /* DEBUG */
...
#ifdef _BKP
  EXT BKP_TypeDef                *BKP;
#endif /*_BKP */
```

```
...

#endif
```

当使用调试模式时，BKP 指针在文件 stm32f10x_lib.c 中初始化：

```
#ifdef _BKP

  BKP = (BKP_TypeDef *)  BKP_BASE;

#endif /*_BKP */
```

为了访问备份寄存器，_BKP 必须在头文件 stm32f10x_conf.h 中定义，如下：

```
#define _BKP
```

5.2 固件库函数

表 54 列出了 BKP 的库函数。

表 54 BKP 库函数

函数名	描述
BKP_DeInit	将 BKP 外设寄存器复位到它们的默认值
BKP_TamperPinLevelConfig	配置窜改引脚的有效电平
BKP_TamperPinCmd	使能/禁止窜改引脚激活

BKP_ITConfig	使能/禁止窜改引脚中断
BKP_RTCCalibrationClockOutput Cmd	使能/禁止校准时钟的输出
BKP_SetRTCCalibrationValue	设置 RTC 时钟校准值
BKP_WriteBackupRegister	将用户数据写入指定的数据备份寄存器
BKP_ReadBackupRegister	从指定的数据备份寄存器中读出数据
BKP_GetFlagStatus	检查窜改引脚事件标志是否置位
BKP_ClearFlag	清除窜改引脚事件挂起标志
BKP_GetITStatus	检查窜改引脚中断是否发生
BKP_ClearITPendingBit	清除窜改引脚中断挂起位

5.2.1 函数BKP_DeInit

表 55 描述了函数 BKP_DeInit

表 55 函数 BKP_DeInit

函数名	BKP_DeInit
函数原型	void BKP_DeInit(void)
功能描述	将 BKP 外设寄存器复位到它们的默认值

输入参数	无
输出参数	无
返回值	无
前提条件	无
调用的函数	RCC_BackupResetCmd

实例：

```
/* Reset the BKP registers */

BKP_DeInit();
```

5.2.2 函数BKP_TamperPinLevelConfig

表 56 描述了函数 BKP_TamperPinLevelConfig

表 56 BKP_TamperPinLevelConfig

函数名	BKP_TamperPinLevelConfig
函数原型	void BKP_TamperPinLevelConfig(u16 BKP_TamperPinLevel)
功能描述	配置窜改引脚的有效电平
输入参数	BKP_TamperPinLevel: 窜改引脚的有效电平 请参阅章节: BKP_TamperPinLevel 中参数取值的详细信息

输出参数	无
返回值	无
前提条件	无
调用的函数	无

BKP_TamperPinLevel

输入参数 BKP_TamperPinLevel 用来选择窜改引脚的有效电平。它可以取下列其中之一值：

表 57 BKP_TamperPinLevel 取值

BKP_TamperPinLevel	描述
BKP_TamperPinLevel_High	窜改引脚高有效
BKP_TamperPinLevel_Low	窜改引脚低有效

实例：

```
/* Configure Tamper pin to be active on high level*/
BKP_TamperPinLevelConfig(BKP_TamperPinLevel_High);
```

5.2.3 函数BKP_TamperPinCmd

表 58 描述了函数 BKP_TamperPinCmd

表 58 函数 BKP_TamperPinCmd

函数名	BKP_TamperPinCmd
函数原型	void BKP_TamperPinCmd(FunctionalState NewState)
功能描述	使能/禁止窜改引脚激活
输入参数	NewState: 窜改引脚激活的新状态 该参数取值：ENABLE 或 DISABLE
输出参数	无
返回值	无
前提条件	无
调用的函数	无

实例：

```
/* Enable Tamper Pin functionality */

BKP_TamperPinCmd(ENABLE);
```

5.2.4 函数BKP_ITConfig

表 59 描述了函数 BKP_ITConfig

表 59 函数 BKP_ITConfig

函数名	BKP_ITConfig
函数原型	void BKP_ITConfig(FunctionalState NewState)
功能描述	使能/禁止窜改引脚中断
输入参数	NewState: 窜改引脚中断的新状态 该参数取值：ENABLE 或 DISABLE
输出参数	无
返回值	无
前提条件	无
调用的函数	无

实例：

```
/* Enable Tamper Pin interrupt */
BKP_ITConfig(ENABLE);
```

5.2.5 函数BKP_RTCCalibrationClockOutputCmd

表 60 描述了函数 BKP_RTCCalibrationClockOutputCmd

表 60 函数 BKP_RTCCalibrationClockOutputCmd

函数名	BKP_RTCCalibrationClockOutputCmd
函数原型	void

	BKP_RTCCalibrationClockOutputCmd(FunctionalState NewState)
功能描述	使能/禁止校准时钟的输出
输入参数	NewState: 校准时钟输出的新状态 该参数取值：ENABLE 或 DISABLE
输出参数	无
返回值	无
前提条件	在使用该函数前，窜改引脚功能需要先禁止
调用的函数	无

实例：

```
/* Output the RTC clock source with frequency divided by 64 on the
Tamper pad(if the Tamper Pin functionality is disabled) */
BKP_RTCCalibrationClockOutputCmd(ENABLE);
```

5.2.6 函数BKP_SetRTCCalibrationValue

表 61 描述了函数 BKP_SetRTCCalibrationValue

表 61 函数 BKP_SetRTCCalibrationValue

函数名	BKP_SetRTCCalibrationValue

函数原型	void BKP_RTCCalibrationClockOutputCmd(FunctionalState NewState)
功能描述	设置 RTC 时钟校准值
输入参数	CalibrationValue: RTC 时钟校准值 该参数取值从 0 到 0x7F
输出参数	无
返回值	无
前提条件	无
调用的函数	无

实例：

```
/* Set RTC clock calibration value to 0x7F (maximum) */
BKP_SetRTCCalibrationValue(0x7F);
```

5.2.7 函数BKP_WriteBackupRegister

表 62 描述了函数 BKP_WriteBackupRegister

表 62 函数 BKP_WriteBackupRegister

函数名	BKP_WriteBackupRegister

函数原型	void BKP_WriteBackupRegister(u16 BKP_DR, u16 Data)
功能描述	将用户数据写入指定的数据备份寄存器
输入参数 1	BKP_DR: 数据备份寄存器 请参阅章节: BKP_DR 中参数取值的详细信息
输入参数 2	Data: 需要写入的数据
输出参数	无
返回值	无
前提条件	无
调用的函数	无

BKP_DR

BKP_DR 用来选择数据寄存器。表 63 给出了参数可能的取值。

表 63 BKP_DR 取值

BKP_DR	描述
BKP_DR1	选择数据备份寄存器 1
BKP_DR2	选择数据备份寄存器 2
BKP_DR3	选择数据备份寄存器 3
BKP_DR4	选择数据备份寄存器 4
BKP_DR5	选择数据备份寄存器 5
BKP_DR6	选择数据备份寄存器 6

BKP_DR7	选择数据备份寄存器 7
BKP_DR8	选择数据备份寄存器 8
BKP_DR9	选择数据备份寄存器 9
BKP_DR10	选择数据备份寄存器 10

实例：

```
/* Write 0xA587 to Data Backup Register1 */

BKP_WriteBackupRegister(BKP_DR1, 0xA587);
```

5.2.8 函数BKP_ReadBackupRegister

表 64 描述了函数 BKP_ReadBackupRegister

表 64 函数 BKP_ReadBackupRegister

函数名	BKP_SetRTCCalibrationValue
函数原型	u16 BKP_ReadBackupRegister(u16 BKP_DR)
功能描述	从指定的数据备份寄存器中读出数据
输入参数	BKP_DR: 数据备份寄存器 请参阅章节: BKP_DR 中参数取值的详细信息
输出参数	无
返回值	指定的数据备份寄存器的内容

前提条件	无
调用的函数	无

实例：

```
/* Read Data Backup Register1 */

u16 Data;

Data = BKP_ReadBackupRegister(BKP_DR1);
```

5.2.9 函数BKP_GetFlagStatus

表 65 描述了函数 BKP_GetFlagStatus

表 65 函数 BKP_GetFlagStatus

函数名	BKP_GetFlagStatus
函数原型	FlagStatus BKP_GetFlagStatus(void)
功能描述	检查窜改引脚事件标志是否置位
输入参数	无
输出参数	无
返回值	窜改引脚事件标志的新状态（SET 或 RESET）
前提条件	无
调用的函数	无

实例：

```
/* Test if the Tamper Pin Event flag is set or not */
FlagStatus Status;
Status = BKP_GetFlagStatus();
if(Status == RESET)
{
...
}
else
{
...
}
```

5.2.10 函数BKP_ClearFlag

表 66 描述了函数 BKP_ClearFlag

表 66 函数 BKP_ClearFlag

函数名	BKP_ClearFlag
函数原型	void BKP_ClearFlag(void)

功能描述	清除窜改引脚事件挂起标志
输入参数	无
输出参数	无
返回值	无
前提条件	无
调用的函数	无

实例：

```
/* Clear Tamper Pin Event pending flag */

BKP_ClearFlag();
```

5.2.11 函数BKP_GetITStatus

表 67 描述了函数 BKP_GetITStatus

表 67 函数 BKP_GetITStatus

函数名	BKP_GetITStatus
函数原型	void BKP_ClearFlag(void)
功能描述	检查窜改引脚中断是否发生
输入参数	无
输出参数	无

返回值	窜改引脚中断的新状态（SET 或 RESET）
前提条件	无
调用的函数	无

实例：

```
/* Test if the Tamper Pin interrupt has occurred or not */
ITStatus Status;
Status = BKP_GetITStatus();
if(Status == RESET)
{
...
}
else
{
...
}
```

5.2.12 函数BKP_ClearITPendingBit

表 68 描述了函数 BKP_ClearITPendingBit

表 68 函数 BKP_ClearITPendingBit

函数名	BKP_ClearITPendingBit
函数原型	void BKP_ClearITPendingBit(void)
功能描述	清除窜改引脚中断挂起位
输入参数	无
输出参数	无
返回值	无
前提条件	无
调用的函数	无

实例：

```
/* Clear Tamper Pin interrupt pending bit */

BKP_ClearITPendingBit();
```

6 控制器局域网（CAN）

该外围模块与 CAN 网络相连接。它支持 CAN 协议 2.0A 和 B 版。它能够以较小的 CPU 负载，高效地管理大量输入报文。也能够满足输出报文的优先级需要。

Section6.1 描述了在 CAN 固件库中使用的数据结构。Section6.2 给出了固件库函数。

6.1 CAN寄存器结构

CAN 寄存器结构，CAN_TypeDef，在头文件 stm32f10x_map.h 中定义，如下：

```
typedef struct
{
  vu32 MCR;
  vu32 MSR;
  vu32 TSR;
  vu32 RF0R;
  vu32 RF1R;
  vu32 IER;
  vu32 ESR;
  vu32 BTR;
  u32 RESERVED0[88];
  CAN_TxMailBox_TypeDef sTxMailBox[3];
  CAN_FIFOMailBox_TypeDef sFIFOMailBox[2];
  u32 RESERVED1[12];
  vu32 FMR;
  vu32 FM0R;
  u32 RESERVED2[1];
  vu32 FS0R;
```

```
  u32 RESERVED3[1];
  vu32 FFA0R;
  u32 RESERVED4[1];
  vu32 FA0R;
  u32 RESERVED5[8];
  CAN_FilterRegister_TypeDef sFilterRegister[14];
} CAN_TypeDef;

typedef struct
{
  vu32 TIR;
  vu32 TDTR;
  vu32 TDLR;
  vu32 TDHR;
} CAN_TxMailBox_TypeDef;

typedef struct
{
  vu32 RIR;
  vu32 RDTR;
  vu32 RDLR;
```

```
  vu32 RDHR;
} CAN_FIFOMailBox_TypeDef;

typedef struct
{
vu32 FR0;
vu32 FR1;
} CAN_FilterRegister_TypeDef;
```

表 69 给出了 CAN 寄存器列表

表 69 CAN 寄存器

寄存器	描述
CAN_MCR	CAN 主机控制寄存器
CAN_MSR	CAN 主机状态寄存器
CAN_TSR	CAN 发送状态寄存器
CAN_RF0R	CAN 接收 FIFO 0 寄存器
CAN_RF1R	CAN 接收 FIFO 1 寄存器
CAN_IER	CAN 中断使能寄存器
CAN_ESR	CAN 错误状态寄存器
CAN_BTR	CAN 位定时寄存器

TIR	Tx 邮箱标志符寄存器
TDTR	邮箱数据长度控制和时间戳寄存器
TDLR	邮箱数据低位寄存器
TDHR	邮箱数据高位寄存器
RIR	Rx FIFO 邮箱标志符寄存器
RDTR	接收 FIFO 邮箱数据长度控制和时间戳寄存器
RDLR	接收 FIFO 邮箱数据低位寄存器
RDHR	接收 FIFO 邮箱数据高位寄存器
CAN_FMR	CAN 过滤主寄存器
CAN_FM0R	CAN 过滤模式寄存器
CAN_FSC0R	CAN 过滤规模寄存器
CAN_FFA0R	CAN 过滤 FIFO 分配寄存器
CAN_FA0R	CAN 过滤激活寄存器
CAN_FR0	过滤 x 寄存器 0
CAN_FR1	过滤 x 寄存器 1

CAN 外围模块也在头文件 stm32f10x_map.h 中声明：

```
#define PERIPH_BASE              ((u32)0x40000000)
#define APB1PERIPH_BASE          PERIPH_BASE
#define APB2PERIPH_BASE          (PERIPH_BASE + 0x10000)
#define AHBPERIPH_BASE           (PERIPH_BASE + 0x20000)
#define CAN_BASE                 (APB1PERIPH_BASE + 0x6400)
#ifndef DEBUG
...
#ifdef _CAN
  #define CAN                    ((CAN_TypeDef *) CAN_BASE)
#endif /*_CAN */
...
#else     /* DEBUG */
...
#ifdef _CAN
  EXT CAN_TypeDef                *CAN;
#endif /*_CAN */
...
#endif
```

当使用调试模式时，CAN 指针在头文件 stm32f10x_lib.c 中初始化：

```
#ifdef _CAN
  CAN = (CAN_TypeDef *)  CAN_BASE;
#endif /*_CAN */
```

为了访问 CAN 寄存器，_CAN 必须在头文件 stm32f10x_conf.h 中定义：

```
#define _CAN
```

6.2 固件库函数

表 70 列出了 CAN 的库函数。

表 70 CAN 库函数

函数名	描述
CAN_DeInit	将 CAN 外设寄存器复位为它们的默认值
CAN_Init	根据 CAN_InitStruct 中指定的参数初始化 CAN 外围模块。
CAN_FilterInit	根据 CAN_FilterInitStruct 中指定的参数初始化 CAN 外围模块。
CAN_StructInit	将 CAN_InitStruct 每个成员变量置为默

	认值
CAN_ITConfig	使能/禁止指定的 CAN 中断
CAN_Transmit	初始化报文的发送
CAN_TransmitStatus	检查报文的发送
CAN_CancelTransmit	取消一个发送请求
CAN_FIFORelease	释放一个 FIFO
CAN_MessagePending	返回挂起报文的数量
CAN_Receive	接收一个报文
CAN_Sleep	进入低功耗模式
CAN_WakeUp	唤醒 CAN
CAN_GetFlagStatus	检查指定的 CAN 标志是否置位
CAN_ClearFlag	清除 CAN 挂起标志
CAN_GetITStatus	检查指定的 CAN 中断是否发生
CAN_ClearITPendingBit	清除 CAN 中断挂起位

6.2.1 函数CAN_DeInit

表 71 描述了函数 CAN_DeInit

表 71 函数 CAN_DeInit

函数名	CAN_DeInit

函数原型	void CAN_DeInit(void)
功能描述	将 CAN 外设寄存器复位为它们的默认值
输入参数	无
输出参数	无
返回值	无
前提条件	无
调用的函数	RCC_APB1PeriphResetCmd()

实例：

```
/* Deinitialize the CAN */

CAN_DeInit();
```

6.2.2 函数CAN_Init

表 72 描述了函数 CAN_Init

表 72 函数 CAN_Init

函数名	CAN_Init
函数原型	u8 CAN_Init(CAN_InitTypeDef* CAN_InitStruct)
功能描述	根据 CAN_InitStruct 中指定的参数初始化 CAN 外围模块。
输入参数	CAN_InitStruct: 指向结构体 CAN_InitTypeDef 的指针 ,包

	含了 CAN 外围模块的配置信息。 请参阅章节: CAN_InitTypeDef structure 中参数取值的详细信息。
输出参数	无
返回值	一个说明 CAN 初始化是否成功的常量。 CANINITFAILED = 初始化失败 CANINITOK = 初始化成功
前提条件	无
调用的函数	无

CAN_InitTypeDef structure

结构体 CAN_InitTypeDef 定义在头文件 stm32f10x_can.h 中：

```
typedef struct
{
FunctionnalState CAN_TTCM;
FunctionnalState CAN_ABOM;
FunctionnalState CAN_AWUM;
FunctionnalState CAN_NART;
FunctionnalState CAN_RFLM;
FunctionnalState CAN_TXFP;
```

```
u8 CAN_Mode;
  u8 CAN_SJW;
u8 CAN_BS1;
u8 CAN_BS2;
u16 CAN_Prescaler;
} CAN_InitTypeDef;
```

CAN_TTCM

CAN_TTCM 用来使能/禁止时间触发通讯模式。该参数取值为 ENABLE 或 DISABLE。

CAN_ABOM

CAN_ABOM 用来使能/禁止总线自动关闭管理。该参数取值为 ENABLE 或 DISABLE。

CAN_AWUM

CAN_AWUM 用来使能/禁止自动唤醒模式。该参数取值为 ENABLE 或 DISABLE。

CAN_NART

CAN_NART 用来使能/禁止非自动重传模式。该参数取值为 ENABLE 或 DISABLE。

CAN_RFLM

CAN_RFLM 用来使能/禁止接收 FIFO 锁定模式。该参数取值为 ENABLE 或 DISABLE。

CAN_TXFP

CAN_TXFP 用来使能/禁止发送 FIFO 优先级。该参数取值为 ENABLE 或 DISABLE。

CAN_Mode

CAN_Mode 配置 CAN 工作模式。该参数可能的取值在表 73 中给出。

表 73 CAN_Mode 取值

CAN_Mode	描述
CAN_Mode_Normal	CAN 硬件工作在普通模式下
CAN_Mode_Silent	CAN 硬件工作在无声模式下
CAN_Mode_LoopBack	CAN 硬件工作在回送（loop back）模式下
CAN_Mode_Silent_LoopBack	CAN 硬件工作在回送（loop back）和无声相结合模式下

CAN_SJW

CAN_SJW 配置时间量子（time quanta）的最大数，允许 CAN 硬件将该值加长或者缩短一位来实现重新同步。表 74 给出了该成员变量可能的取值。

表 74 CAN_SJW 取值

CAN_SJW	Description
CAN_SJW_0tq	重新同步跳转宽度 = 0 个时间量子（time quanta）
CAN_SJW_1tq	重新同步跳转宽度 = 1 个时间量子（time quanta）
CAN_SJW_2tq	重新同步跳转宽度 = 2 个时间量子（time quanta）
CAN_SJW_3tq	重新同步跳转宽度 = 3 个时间量子（time quanta）

CAN_BS1

CAN_BS1 在位字段 1 中配置时间量子（time quanta）的数量。表 75 给出了该成员变量可能的取值。

表 75 CAN_BS1 取值

CAN_BS1	描述
CAN_BS1_1tq	位字段 1=1 个时间量子(time quanta)
……	……
CAN_BS1_16tq	位字段 1=16 时间量子（time quanta）

CAN_BS2

CAN_BS1 在位字段 2 中配置时间量子（time quanta）的数量。表 76 给出了该成员变量可能的取值。

表 76 CAN_BS2 取值

CAN_BS2	描述
CAN_BS2_1tq	位字段 2=1 个时间量子(time quanta)
……	……
CAN_BS2_8tq	位字段 2=16 时间量子（time quanta）

CAN_Prescaler

CAN_Prescaler 配置时间量子（time quanta）的长度。可从 1 到 1024。

实例：

```
/* Initialize the CAN as 1Mb/s in normal mode, receive FIFO locked: */

CAN_InitTypeDef CAN_InitStructure;

CAN_InitStructure.CAN_TTCM = DISABLE;

CAN_InitStructure.CAN_ABOM = DISABLE;

CAN_InitStructure.CAN_AWUM = DISABLE;

CAN_InitStructure.CAN_NART = DISABLE;

CAN_InitStructure.CAN_RFLM = ENABLE;

CAN_InitStructure.CAN_TXFP = DISABLE;
```

```
CAN_InitStructure.CAN_Mode = CAN_Mode_Normal;
CAN_InitStructure.CAN_BS1 = CAN_BS1_4tq;
CAN_InitStructure.CAN_BS2 = CAN_BS2_3tq;
CAN_InitStructure.CAN_Prescaler = 0;
CAN_Init(&CAN_InitStructure);
```

6.2.3 函数CAN_FilterInit

表 77 描述了函数 CAN_FilterInit

表 77 函数 CAN_FilterInit

函数名	CAN_FilterInit
函数原型	void CAN_FilterInit(CAN_FilterInitTypeDef* CAN_FilterInitStruct)
功能描述	根据 CAN_FilterInitStruct 中指定的参数初始化 CAN 外围模块。
输入参数	CAN_FilterInitStruct: 指向结构体 CAN_FilterInitTypeDef 的指针，它包含了配置信息。 请参阅章节: 结构体 CAN_FilterInitTypeDef 中参数可取值的详细信息。
输出参数	无
返回值	无

前提条件	无
调用的函数	无

CAN_FilterInitTypeDef structure

结构体 CAN_FilterInitTypeDef 定义在头文件 stm32f10x_can.h 中。

```
typedef struct
{
u8 CAN_FilterNumber;
u8 CAN_FilterMode;
u8 CAN_FilterScale;
u16 CAN_FilterIdHigh;
u16 CAN_FilterIdLow;
u16 CAN_FilterMaskIdHigh;
u16 CAN_FilterMaskIdLow;
u16 CAN_FilterFIFOAssignment;
FunctionalState CAN_FilterActivation;
} CAN_FilterInitTypeDef;
```

CAN_FilterNumber

CAN_FilterNumber 选择将要被初始化的过滤器。它的值为 0 - 13。

CAN_FilterMode

CAN_FilterMode 选择需要被初始化的模式。表 78 给出了该成员变量可能的取值。

表 78 CAN_FilterMode 取值

CAN_FilterMode	描述
CAN_FilterMode_IdMask	标识/屏蔽模式
CAN_FilterMode_IdList	标识列表模式

CAN_FilterScale

CAN_FilterScale 配置过滤器规模。表 79 给出了该成员变量可能的取值。

表 79 CAN_FilterScale 取值

CAN_FilterScale	描述
CAN_FilterScale_Two16bit	两个 16 位过滤器
CAN_FilterScale_One32bit	一个 32 位过滤器

CAN_FilterIdHigh

CAN_FilterIdHigh 用来选择过滤器标识号（对应一个 32-bit 配置的高 16-bit MSBs）。范围从 0x0000 到 0Xffff。

CAN_FilterIdLow

CAN_FilterIdLow 用来选择过滤器标识号（对应一个 32-bit 配置的低 16-bit LSBs）。范围从 0x0000 到 0Xffff。

CAN_FilterMaskIdHigh

CAN_FilterMaskIdHigh 根据模式来选择过滤器屏蔽号或者标识号（对应一个 32-bit 配置的高 16-bit MSBs）。范围从 0x0000 到 0Xffff。

CAN_FilterMaskIdLow

CAN_FilterMaskIdLow 根据模式用来选择过滤器屏蔽号或者标识号（对应一个 32-bit 配置的低 16-bit LSBs）。范围从 0x0000 到 0Xffff。

CAN_FilterFIFO

CAN_FilterFIFO 用来选择分配给过滤器的 FIFO（0 或 1）。表 80 给出了该成员变量可能的取值。

表 80 CAN_FilterFIFO 取值

CAN_FilterFIFO	描述
CAN_FilterFIFO0	过滤器 FIFO 0 分配给过滤器 x
CAN_FilterFIFO1	过滤器 FIFO 1 分配给过滤器 x

CAN_FilterActivation

CAN_FilterActivation 使能/禁止过滤器。它可以被置 ENABLE 或 DISABLE。

实例：

```
/* Initialize the CAN filter 2 */

CAN_FilterInitTypeDef CAN_FilterInitStructure;

CAN_FilterInitStructure.CAN_FilterNumber = 2;

CAN_FilterInitStructure.CAN_FilterMode = CAN_FilterMode_IdMask;

CAN_FilterInitStructure.CAN_FilterScale = CAN_FilterScale_One32bit;

CAN_FilterInitStructure.CAN_FilterIdHigh = 0x0F0F;

CAN_FilterInitStructure.CAN_FilterIdLow = 0xF0F0;

CAN_FilterInitStructure.CAN_FilterMaskIdHigh = 0xFF00;

CAN_FilterInitStructure.CAN_FilterMaskIdLow = 0x00FF;

CAN_FilterInitStructure.CAN_FilterFIFO = CAN_FilterFIFO0;

CAN_FilterInitStructure.CAN_FilterActivation = ENABLE;

CAN_FilterInit(&CAN_InitStructure);
```

6.2.4 函数CAN_StructInit

表 81 描述了函数 CAN_StructInit

表 81 函数 CAN_StructInit

函数名	CAN_StructInit

函数原型	void CAN_StructInit(CAN_InitTypeDef* CAN_InitStruct)
功能描述	将 CAN_InitStruct 每个成员变量置为默认值
输入参数	CAN_InitStruct: 指向将要被初始化的结构体 CAN_InitTypeDef。 请参阅表 82 中 CAN_InitStruct 成员变量的默认值。
输出参数	无
返回值	无
前提条件	无
调用的函数	无

表 82 CAN_InitStruct 默认值

成员变量	默认值
CAN_TTCM	禁止
CAN_ABOM	禁止
CAN_AWUM	禁止
CAN_NART	禁止
CAN_RFLM	禁止
CAN_TXFP	禁止
CAN_Mode	CAN_Mode_Normal
CAN_SJW	CAN_SJW_0tq
CAN_BS1	CAN_BS1_4tq

CAN_BS2	CAN_BS2_3tq
CAN_Prescaler	1

实例：

```
/* Initialize a CAN_InitTypeDef structure. */
CAN_InitTypeDef  CAN_InitStructure;
CAN_StructInit(&CAN_InitStructure);
```

6.2.5 函数CAN_ITConfig

表 83 描述了函数 CAN_ITConfig

表 83 函数 CAN_ITConfig

函数名	CAN_ITConfig
函数原型	void CAN_ITConfig(u32 CAN_IT, FunctionalState NewState)
功能描述	使能/禁止指定的 CAN 中断
输入参数 1	CAN_IT: 配置使能/禁止的 CAN 中断源 请参阅 Section: CAN_IT 中参数可取值的详细信息
输入参数 2	NewState: CAN 中断的新状态 该参数可以是：ENABLE 或 DISABLE

输出参数	无
返回值	无
前提条件	无
调用的函数	无

CAN_IT

CAN_IT 输入参数使能/禁止 CAN 中断。可以使用以下取值中的一个或多个组合：

表 84 CAN_IT 取值

CAN_IT	描述
CAN_IT_TME	发送邮箱空屏蔽
CAN_IT_FMP0	FIFO 0 报文挂起屏蔽
CAN_IT_FF0	FIFO 0 满屏蔽
CAN_IT_FOV0	FIFO 0 溢出屏蔽
CAN_IT_FMP1	FIFO 1 报文挂起屏蔽
CAN_IT_FF1	FIFO 1 满屏蔽
CAN_IT_FOV1	FIFO 1 溢出屏蔽
CAN_IT_EWG	错误警告屏蔽
CAN_IT_EPV	错误被动（Passive）屏蔽
CAN_IT_BOF	总线关闭屏蔽
CAN_IT_LEC	上次错误代码屏蔽

CAN_IT_ERR	错误屏蔽
CAN_IT_WKU	唤醒屏蔽
CAN_IT_SLK	睡眠标志屏蔽

实例：

```
/* Enable CAN FIFO 0 overrun interrupt */
CAN_ITConfig(CAN_IT_FOV0, ENABLE);
```

6.2.6 函数CAN_Transmit

表 85 描述了函数 CAN_Transmit

表 85 函数 CAN_Transmit

函数名	CAN_Transmit
函数原型	u8 CAN_Transmit(CanTxMsg* TxMessage)
功能描述	初始化报文的发送
输入参数	TxMessage: 指向一个结构体的指针 ,它包括 CAN ID, CAN DLC 和 CAN 数据
输出参数	无
返回值	返回值是用于发送的邮箱数或者是没有空邮箱时的 CAN_NO_MB 值

前提条件	无
调用的函数	无

CanTxMsg

结构体 CanTxMsg 定义在头文件 stm32f10x_can.h 中：

```
typedef struct
{
u32 StdId;
u32 ExtId;
u8 IDE;
u8 RTR;
u8 DLC;
u8 Data[8];
} CanTxMsg;
```

StdId

StdId 用来配置标准的标识符。该成员变量的范围从 0 到 0x7FF。

ExtId

ExtId 用来配置扩展的标识符。该成员变量的范围从 0 到 0x3FFFF。

IDE

IDE 用来为将要发送的报文配置标识符的类型。表 86 给出了该成员变量的可能取值。

表 86 IDE 取值

IDE	描述
CAN_ID_STD	使用标准 ID
CAN_ID_EXT	使用扩展 ID + 标准 ID

RTR

RTR 用来选择将要发送的报文的帧的类型 ,它可被设置成数据帧或远程控制帧(remote frame)。

表 87 RTR 取值

RTR	描述
CAN_RTR_DATA	数据帧
CAN_RTR_REMOTE	远程控制帧（remote frame）

DLC

DLC 用来配置将要发送的帧的长度。取值范围从 0 到 0x8。

Data[8]

Data[8]包含将要发送的数据。取值范围从 0 到 0xFF。

实例：

```
/* Send a message with the CAN */
CanTxMsg TxMessage;
TxMessage.StdId = 0x1F;
TxMessage.ExtId = 0x00;
TxMessage.IDE = CAN_ID_STD;
TxMessage.RTR = CAN_RTR_DATA;
TxMessage.DLC = 2;
TxMessage.Data[0] = 0xAA;
TxMessage.Data[1] = 0x55;
CAN_Transmit(&TxMessage);
```

6.2.7 函数CAN_TransmitStatus

表 88 描述了函数 CAN_TransmitStatus

表 88 函数 CAN_TransmitStatus

函数名	CAN_TransmitStatus
函数原型	u8 CAN_TransmitStatus(u8 TransmitMailbox)

功能描述	检查报文发送的状态
输入参数	TransmitMailbox: 用于发送的邮箱号码
输出参数	无
返回值	CANTXOK：CAN 驱动正在发送报文 CANTXPENDING：报文被挂起 CANTXFAILED：其他
前提条件	发送正在继续
调用的函数	无

实例：

```
/* Check the status of a transmission with the CAN */
CanTxMsg TxMessage;
...
switch(CAN_TransmitStatus(CAN_Transmit(&TxMessage))
{
case
CANTXOK: ...;break;
...
}
```

6.2.8 函数CAN_CancelTransmit

表 89 描述了函数 CAN_CancelTransmit

表 89 函数 CAN_CancelTransmit

函数名	CAN_CancelTransmit
函数原型	void CAN_CancelTransmit(u8 Mailbox)
功能描述	取消一个发送请求
输入参数	Mailbox number:邮箱号
输出参数	无
返回值	无
前提条件	在邮箱中有一个发送被挂起
调用的函数	无

实例：

```
/* Cancel a CAN transmit initiates by CANTransmit */
u8 MBNumber;
CanTxMsg TxMessage;
MBNumber = CAN_Transmit(&TxMessage);
if (CAN_TransmitStatus(MBNumber) == CANTXPENDING)
{
CAN_CancelTransmit(MBNumber);
```

}

6.2.9 函数CAN_FIFORelease

表 90 描述了函数 CAN_FIFORelease

表 90 函数 CAN_FIFORelease

函数名	CAN_FIFORelease
函数原型	void CAN_FIFORelease(u8 FIFONumber)
功能描述	释放一个 FIFO
输入参数	FIFO 号码: 需要释放的 FIFO，CANFIFO0 或 CANFIFO1
输出参数	无
返回值	无
前提条件	无
调用的函数	无

实例：

```
/* Release FIFO 0*/

CAN_FIFORelease(CANFIFO0);
```

6.2.10 函数CAN_MessagePending

表 91 描述了函数 CAN_MessagePending

表 91 函数 CAN_MessagePending

函数名	CAN_MessagePending
函数原型	u8 CAN_MessagePending(u8 FIFONumber)
功能描述	返回挂起报文的数量
输入参数	FIFO 号码： 接收 FIFO，CANFIFO0 或 CANFIFO1
输出参数	无
返回值	NbMessage，挂起报文的数量
前提条件	无
调用的函数	无

实例：

```
/* Check the number of pending messages for FIFO 0*/
u8 MessagePending = 0;
MessagePending = CAN_MessagePending(CANFIFO0);
```

6.2.11 函数CAN_Receive

表 92 描述了函数 CAN_Receive

表 92 函数 CAN_Receive

函数名	CAN_Receive
函数原型	void CAN_Receive(u8 FIFONumber, CanRxMsg* RxMessage)
功能描述	接收一个报文
输入参数	FIFO number: 接收 FIFO 的编号，CANFIFO0 或 CANFIFO1
输出参数	RxMessage: 指向一个包括 CAN ID,CAN DLC 和 CAN 数据的结构体。
返回值	无
前提条件	无
调用的函数	无

CanRxMsg structure

结构体 CanRxMsg 定义在头文件 stm32f10x_can.h 中：

```
typedef struct
{
u32 StdId;
u32 ExtId;
```

u8 IDE;

u8 RTR;

u8 DLC;

u8 Data[8];

u8 FMI;

} CanRxMsg;

StdId

StdId 用来配置标准的标识符。该成员变量的范围从 0 到 0x7FF。

ExtId

ExtId 用来配置扩展的标识符。该成员变量的范围从 0 到 0x3FFFF。

IDE

IDE 用来配置接收的报文的标识符类型。表 93 给出了该成员变量的可能取值。

表 93 IDE 取值

IDE	描述
CAN_ID_STD	使用标准 ID
CAN_ID_EXT	使用扩展 ID + 标准 ID

RTR

RTR 用来选择接收报文的帧类型。它可被设置成数据帧或远程控制帧（remote frame）。

表 94 RTR 取值

RTR	描述
CAN_RTR_DATA	数据帧
CAN_RTR_REMOTE	远程控制帧（remote frame）

DLC

DLC 用来配置将要发送的帧的长度。取值范围从 0 到 0x8。

Data[8]

Data[8]包含接收的数据。取值范围从 0 到 0xFF。

FMI

FMI 配置保存在邮箱里的报文对应的过滤器索引。（FMI configures the index of the filter the message stored in the mailbox passes through.）。FMI 取值范围从 0 到 0xFF。

实例：

```
/* Receive a message with the CAN */
CanRxMsg RxMessage;
```

CAN_Receive(&RxMessage);

6.2.12 函数CAN_Sleep

表 95 描述了函数 CAN_Sleep

表 95 函数 CAN_Sleep

函数名	CAN_Sleep
函数原型	u8 CAN_Sleep(void)
功能描述	将 CAN 置于低功耗模式
输入参数	无
输出参数	无
返回值	CANSLEEPOK（如果进入睡眠） CANSLEEPFAILED（其他情况）
前提条件	无
调用的函数	无

实例：

/* Enter the CAN sleep mode*/

CAN_Sleep();

6.2.13 函数CAN_WakeUp

表 96 描述了函数 CAN_WakeUp

表 96 函数 CAN_WakeUp

函数名	CAN_WakeUp
函数原型	u8 CAN_WakeUp(void)
功能描述	唤醒 CAN
输入参数	无
输出参数	无
返回值	CANWAKEUPOK（如果退出了睡眠模式） CANWAKEUPFAILED（其他情况）
前提条件	无
调用的函数	无

实例：

```
/* CAN waking up */
CAN_WakeUp();
```

6.2.14 函数CAN_GetFlagStatus

表 97 描述了函数 CAN_GetFlagStatus

表 97 函数 CAN_GetFlagStatus

函数名	CAN_GetFlagStatus
函数原型	FlagStatus CAN_GetFlagStatus(u32 CAN_FLAG)
功能描述	检查指定的 CAN 标志是否置位
输入参数	CAN_FLAG: 指定要检查的标志 请参阅 Section: CAN_FLAG 中参数可取值的详细信息
输出参数	无
返回值	CAN_FLAG 的新状态（SET 或 RESET）
前提条件	无
调用的函数	无

CAN_FLAG

CAN_FLAG 用来定义需要检查的标志的类型。表 98 给出了 CAN_FLAG 取值的描述。

表 98 CAN_FLAG 定义

CAN_FLAG	描述
CAN_FLAG_EWG	错误警告标志
CAN_FLAG_EPV	错误被动（Passive）标志
CAN_FLAG_BOF	总线关闭标志

实例：

```
/* Test if the CAN warning limit has been reached */

FlagStatus Status;

Status = CAN_GetFlagStatus(CAN_FLAG_EWG);
```

6.2.15 函数CAN_ClearFlag

表 99 描述了函数 CAN_ClearFlag

表 99 函数 CAN_ClearFlag

函数名	CAN_ClearFlag
函数原型	void CAN_ClearFlag(u32 CAN_Flag)
功能描述	清除 CAN 挂起标志
输入参数	CAN_FLAG: 指定需要清除的标志 请参阅 Section: CAN_FLAG 中参数可取值的详细信息
输出参数	无
返回值	无
前提条件	无
调用的函数	无

实例：

```
/* Clear the CAN bus-off state flag */

CAN_ClearFlag(CAN_FLAG_BOF);
```

6.2.16 函数CAN_GetITStatus

表 100 描述了函数 CAN_GetITStatus

表 100 函数 CAN_GetITStatus

函数名	CAN_GetITStatus
函数原型	ITStatus CAN_GetITStatus(u32 CAN_IT)
功能描述	检查指定的 CAN 中断是否发生
输入参数	CAN_IT: CAN 需要检查的中断源 请参阅 Section: CAN_IT 中参数可取值的详细信息
输出参数	无
返回值	CAN_IT 的新状态（SET 或 RESET）
前提条件	无
调用的函数	无

CAN_IT

CAN_IT 输入参数选择要被检查的中断。表 101 给出了 CAN_IT 取值的描述。

表 101 CAN_IT 取值

CAN_IT	描述
CAN_IT_RQCP0	邮箱 0 请求完成
CAN_IT_RQCP1	邮箱 1 请求完成
CAN_IT_RQCP2	邮箱 2 请求完成
CAN_IT_FMP0	FIFO 0 报文挂起
CAN_IT_FULL0	FIFO 0 存储了三个报文
CAN_IT_FOVR0	FIFO 0 溢出
CAN_IT_FMP1	FIFO 1 报文挂起
CAN_IT_FULL1	FIFO 1 存储了三个报文
CAN_IT_FOVR1	FIFO 1 溢出
CAN_IT_EWGF	到达警告限制
CAN_IT_EPVF	到达被动错误限制
CAN_IT_BOFF	进入总线关闭状态
CAN_IT_WKUI	低功耗模式下发现 SOF

实例：

```
/* Test if the CAN FIFO 0 overrun interrupt has occurred or not */

ITStatus Status;

Status = CAN_GetITStatus(CAN_IT_FOVR0);
```

6.2.17 函数CAN_ClearITPendingBit

表 102 描述了函数 CAN_ClearITPendingBit

表 102 函数 CAN_ClearITPendingBit

函数名	CAN_ClearITPendingBit
函数原型	void CAN_ClearITPendingBit(u32 CAN_IT)
功能描述	清除 CAN 中断挂起位
输入参数	CAN_IT: 指定需要清除的中断挂起位 请参阅 Section: CAN_IT 中参数取值的详细信息
输出参数	无
返回值	无
前提条件	无
调用的函数	无

实例：

```
/* Clear the CAN error passive overflow interrupt pending bit */
CAN_ClearITPendingBit(CAN_IT_EPVF);
```

7 DMA控制器（DMA Controller）

DMA 控制器可访问 7 种数据通道。由于外围部件映射到内存，那么外围部件间的数据传输像存储器/存储器的数据传输一样管理。

*7.1 小节：DMA 寄存器结构*描述了 DMA 固件库中使用的数据结构。

*7.2 小节：固件库函数*给出了所有的固件库函数。

7.1 DMA寄存器结构

DMA 寄存器结构，DMA_Channel_TypeDef 和 DMA_TypeDef，在 stm32f10x_map.h 按照如下定义：

```
typedef struct
{
  vu32 CCR;
  vu32 CNDTR;
  vu32 CPAR;
  vu32 CMAR;
} DMA_Channel_TypeDef;
typedef struct
{
  vu32 ISR;
```

```
vu32 IFCR;

} DMA_TypeDef;
```

表 103　　DMA 寄存器

寄存器	描述
ISR	DMA 中断状态寄存器
IFCR	DMA 中断标志清除寄存器
CCRx	DMA 通道配置寄存器
CNDTRx	DMA 通道数据传输数量寄存器
CPARx	DMA 通道外设地址寄存器
CMARx	DMA 通道存储器 0 地址寄存器

DMA 及其七个通道也在 stm32f10x_map.h 中进行了声明：

```
...

#define PERIPH_BASE              ((u32)0x40000000)

#define APB1PERIPH_BASE          PERIPH_BASE

#define APB2PERIPH_BASE          (PERIPH_BASE + 0x10000)

#define AHBPERIPH_BASE           (PERIPH_BASE + 0x20000)

....

#define DMA_BASE                 (AHBPERIPH_BASE + 0x0000)

#define DMA_Channel1_BASE        (AHBPERIPH_BASE + 0x0008)
```

```
#define DMA_Channel2_BASE       (AHBPERIPH_BASE + 0x001C)

#define DMA_Channel3_BASE       (AHBPERIPH_BASE + 0x0030)

#define DMA_Channel4_BASE       (AHBPERIPH_BASE + 0x0044)

#define DMA_Channel5_BASE       (AHBPERIPH_BASE + 0x0058)

#define DMA_Channel6_BASE       (AHBPERIPH_BASE + 0x006C)

#define DMA_Channel7_BASE       (AHBPERIPH_BASE + 0x0080)

....

#ifndef DEBUG

...

#ifdef _DMA

  #define DMA                   ((DMA_TypeDef *) DMA_BASE)

#endif /*_DMA */

#ifdef _DMA_Channel1

  #define DMA_Channel1          ((DMA_Channel_TypeDef *)
DMA_Channel1_BASE)

#endif /*_DMA_Channel1 */

#ifdef _DMA_Channel2

  #define DMA_Channel2          ((DMA_Channel_TypeDef *)
DMA_Channel2_BASE)

#endif /*_DMA_Channel2 */

#ifdef _DMA_Channel3
```

```
  #define DMA_Channel3            ((DMA_Channel_TypeDef *)
DMA_Channel3_BASE)
#endif /*_DMA_Channel3 */
#ifdef _DMA_Channel4
  #define DMA_Channel4            ((DMA_Channel_TypeDef *)
DMA_Channel4_BASE)
#endif /*_DMA_Channel4 */
#ifdef _DMA_Channel5
  #define DMA_Channel5            ((DMA_Channel_TypeDef *)
DMA_Channel5_BASE)
#endif /*_DMA_Channel5 */
#ifdef _DMA_Channel6
  #define DMA_Channel6            ((DMA_Channel_TypeDef *)
DMA_Channel6_BASE)
#endif /*_DMA_Channel6 */
#ifdef _DMA_Channel7
  #define DMA_Channel7            ((DMA_Channel_TypeDef *)
DMA_Channel7_BASE)
#endif /*_DMA_Channel7 */
...
#else    /* DEBUG */
```

```
...
#ifdef _DMA
EXT DMA_TypeDef                 *DMA;
#endif /*_DMA */
#ifdef _DMA_Channel1
  EXT DMA_Channel_TypeDef       *DMA_Channel1;
#endif /*_DMA_Channel1 */
#ifdef _DMA_Channel2
  EXT DMA_Channel_TypeDef       *DMA_Channel2;
#endif /*_DMA_Channel2 */
#ifdef _DMA_Channel3
  EXT DMA_Channel_TypeDef       *DMA_Channel3;
#endif /*_DMA_Channel3 */
#ifdef _DMA_Channel4
  EXT DMA_Channel_TypeDef       *DMA_Channel4;
#endif /*_DMA_Channel4 */
#ifdef _DMA_Channel5
  EXT DMA_Channel_TypeDef       *DMA_Channel5;
#endif /*_DMA_Channel5 */
#ifdef _DMA_Channel6
  EXT DMA_Channel_TypeDef       *DMA_Channel6;
```

```
#endif /*_DMA_Channel6 */

#ifdef _DMA_Channel7

  EXT DMA_Channel_TypeDef        *DMA_Channel7;

#endif /*_DMA_Channel7 */

…

#endif
```

当使用 Debug 模式时，DMA，_DMA_Channel1, _DMA_Channel2,……,以及_DMA_Channel7 的指针都在 stm32f10x_lib.c 中进行初始化：

```
…

#ifdef _DMA

  DMA = (DMA_TypeDef *)   DMA_BASE;

#endif /*_DMA */

#ifdef _DMA_Channel1

  DMA_Channel1 = (DMA_Channel_TypeDef *)   DMA_Channel1_BASE;

#endif /*_DMA_Channel1 */

#ifdef _DMA_Channel2

  DMA_Channel2 = (DMA_Channel_TypeDef *)   DMA_Channel2_BASE;

#endif /*_DMA_Channel2 */

#ifdef _DMA_Channel3
```

```
  DMA_Channel3 = (DMA_Channel_TypeDef *)  DMA_Channel3_BASE;
#endif /*_DMA_Channel3 */
#ifdef _DMA_Channel4
  DMA_Channel4 = (DMA_Channel_TypeDef *)  DMA_Channel4_BASE;
#endif /*_DMA_Channel4 */
#ifdef _DMA_Channel5
  DMA_Channel5 = (DMA_Channel_TypeDef *)  DMA_Channel5_BASE;
#endif /*_DMA_Channel5 */
#ifdef _DMA_Channel6
  DMA_Channel6 = (DMA_Channel_TypeDef *)  DMA_Channel6_BASE;
#endif /*_DMA_Channel6 */
#ifdef _DMA_Channel7
  DMA_Channel7 = (DMA_Channel_TypeDef *)  DMA_Channel7_BASE;
#endif /*_DMA_Channel7 */
...
```

为了进入 DMA 寄存器，_DMA, _DMA_Channel1 到 _DMA_Channel7 必须在 stm32f10x_conf.h 中进行如下定义：

```
...
#define _DMA
```

#define _DMA_Channel1

#define _DMA_Channel2

#define _DMA_Channel3

#define _DMA_Channel4

#define _DMA_Channel5

#define _DMA_Channel6

#define _DMA_Channel7

...

7.2 固件库函数

表 104 DMA 固件库函数

函数名	描述
DMA_DeInit	将 DAM 通道 x 的寄存器复位到默认的复位值
DMA_Init	按照 DMA_InitStruct 中指定的参数初始化 DMA 各通道
DMA_StructInit	为 DMA_InitStruct 的成员赋给默认值
DMA_Cmd	使能（或使不能）特定的 DMA 通道
DMA_ITConfig	使能（或使不能）特定的 DMA 通道中断
DMA_GetCurrDataCounter	返回现有的 DMA 通道传输剩余的数据单元
DMA_GetFlagStatus	检查特定的 DMA 各通道标志是否被置位了

DMA_ClearFlag	清除 DMA 各通道的挂起标志
DMA_GetITStatus	检查特定的 DMA 各通道中断是否发生
DMA_ClearITPendingBit	清除 DMA 各通道的挂起位

7.2.1 DMA_DeInit函数

表 105 DMA_DeInit 函数

函数名	DMA_DeInit
函数原型	void DMA_DeInit(DMA_Channel_TypeDef* DMA_Channelx)
功能描述	将 DMA 各通道寄存器复位为默认的复位值
输入参数	DMA_Channelx：x 可被设置为 1 到 7 来选择 DMA 各通道
输出参数	无
返回参数	无
前提条件	无
调用函数	RCC_AHBPeriphClockCmd().

实例：

```
/* Deinitialize the DMA Channel2 */
DMA_DeInit(DMA_Channel2);
```

7.2.2 DMA_Init函数

表 106 DMA_Init 函数

函数名	DMA_Init
函数原型	void DMA_Init(DMA_Channel_TypeDef* DMA_Channelx, DMA_InitTypeDef*DMA_InitStruct)
功能描述	根据 DMA_InitStruct 指定的参数初始化 DMA 各通道
输入参数 1	DMA_Channelx：x 可为 1 到 7 来选择 DMA 各频道
输入参数 2	DMA_InitStruct：指向 DMA_InitTypeDef，它包含特定 DMA 频道的配置信息。 更多细节参考：DMA_InitTypeDef 结构
输出参数	无
返回参数	无
前提条件	无
调用函数	无

DMA_InitTypeDef 结构

DMA_InitTypeDef 定义在 stm32f10x_dma.h 中：

```
typedef struct
{
  u32 DMA_PeripheralBaseAddr;
  u32 DMA_MemoryBaseAddr;
u32 DMA_DIR;
```

```
  u32 DMA_BufferSize;
  u32 DMA_PeripheralInc;
  u32 DMA_MemoryInc;
  u32 DMA_PeripheralDataSize;
  u32 DMA_MemoryDataSize;
  u32 DMA_Mode;
u32 DMA_Priority;
  u32 DMA_M2M;
} DMA_InitTypeDef;
```

DMA_PeripheralBaseAddr

这个成员用来定义 DMA 通道的外设基地址

DMA_MemoryBaseAddr

这个成员用来定义 DMA 通道的存储器基地址

DMA_DIR

DMA_DIR 指定该外设是 DMA 源还是目的地。表 107 给出了它的可用值。

表 107 DMA_DIR 定义

DMA_DIR	描述
DMA_DIR_PeripheralDST	DMA 目的地

DMA_DIR_PeripheralSRC	DMA 源

DMA_BufferSize

DMA_BufferSize 用来定义特定通道的缓冲区的数据单元的大小，这个数据单元根据传输方向，和 DMA_PeripheralDataSize 或 DMA_MemoryDataSize 成员的设置相同

DMA_PeripheralInc

DMA_PeripheralInc 指出外设的地址寄存器是否增加了。表 108 给出了它的可用值。

表 108 DMA_PeripheralInc 定义

DMA_PeripheralInc	描述
DMA_PeripheralInc_Enable	当前外设寄存器增加
DMA_PeripheralInc_Disable	当前外设寄存器不变

DMA_MemoryInc

DMA_MemoryInc 确定了存储器地址寄存器是否增加了。这个值在表 109 中给出。

表 109 DMA_MemoryInc 定义

DMA_MemoryInc	描述
DMA_MemoryInc_Enable	当前存储器寄存器增加
DMA_MemoryInc_Disable	当前存储器寄存器不变

DMA_PeripheralDataSize

DMA_PeripheralDataSize 配置了外设的数据宽度。表 110 给出了它的可用值

表 110 DMA_PeripheralDataSize 定义

DMA_PeripheralDataSize	描述
DMA_PeripheralDataSize_Byte	数据宽度=8 位
DMA_PeripheralDataSize_HalfWord	数据宽度=16 位
DMA_PeripheralDataSize_Word	数据宽度=32 位

DMA_MemoryDataSize

DMA_MemoryDataSize 定义了存储器的数据宽度。表 111 给出了它的可用值

表 111 DMA_MemoryDataSize 定义

DMA_MemoryDataSize	描述
DMA_MemoryDataSize_Byte	数据宽度=8 位
DMA_MemoryDataSize_HalfWord	数据宽度=16 位
DMA_MemoryDataSize_Word	数据宽度=32 位

DMA_Mode

DMA_Mode 配置了 DMA 通道的操作模式。表 112 给出了它的可用值

表 112 DMA_Moded 定义

DMA_Mode	描述
DMA_Mode_Circular	使用环形缓冲模式

DMA_Mode_Normal	使用正常缓冲模式

注意：

当选定的通道被配置为用来进行存储器到存储器数据传输时，不能使用环形缓冲模式（参考 *DMA_M2M*）

DMA_Priority

DMA_Priority 配置了 DMA 各通道的软件优先级。表 113 给出了它的可用值

表 113 DMA_Priority 定义

DMA_Priority	描述
DMA_Priority_VeryHigh	DMA 通道优先级非常高
DMA_Priority_High	DMA 通道优先级高
DMA_Priority_Medium	DMA 通道优先级中等
DMA_Priority_Low	DMA 通道优先级低

DMA_M2M

DMA_M2M 使能 DMA 通道的存储器到存储器传输。表 114 给出了它的可用值

表 114 DMA_M2M 定义

DMA_M2M	描述
DMA_M2M_Enable	DMA 通道配置了存储器到存储器传输
DMA_M2M_Disable	DMA 通道未配置存储器到存储器传输

实例：

```
/* Initialize the DMA Channel1 according to the DMA_InitStructure

members */

DMA_InitTypeDef    DMA_InitStructure;

DMA_InitStructure.DMA_PeripheralBaseAddr = 0x40005400;

DMA_InitStructure.DMA_MemoryBaseAddr = 0x20000100;

DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;

DMA_InitStructure.DMA_BufferSize = 256;

DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;

DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;

DMA_InitStructure.DMA_PeripheralDataSize =

DMA_PeripheralDataSize_HalfWord;

DMA_InitStructure.DMA_MemoryDataSize =

DMA_MemoryDataSize_HalfWord;

DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;

DMA_InitStructure.DMA_Priority = DMA_Priority_Medium;

DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;

DMA_Init(DMA_Channel1, &DMA_InitStructure);
```

7.2.3 DMA_StructInit函数

表 115 DMA_StructInit 函数

函数名	DMA_StructInit
函数原型	void DMA_StructInit(DMA_InitTypeDef* DMA_InitStruct)
功能描述	将 DMA_InitStruct 的每个成员复位为默认值
输入参数	DMA_InitStruc：指向需要初始化的 DMA_InitTypeDef 结构。
输出参数	无
返回参数	无
前提条件	无
调用函数	无

DMA_InitStruct 成员有如下缺省值：

表 116 DMA_InitStruct 缺省值

成员	缺省值
DMA_PeripheralBaseAddr	0
DMA_MemoryBaseAddr	0
DMA_DIR	DMA_DIR_PeripheralSRC
DMA_BufferSize	0
DMA_PeripheralInc	DMA_PeripheralInc_Disable
DMA_MemoryInc	DMA_MemoryInc_Disable
DMA_PeripheralDataSize	DMA_PeripheralDataSize_Byte

DMA_MemoryDataSize	DMA_MemoryDataSize_Byte
DMA_Mode	DMA_Mode_Normal
DMA_Priority	DMA_Priority_Low
DMA_M2M	DMA_M2M_Disable

实例：

```
/* Initialize a DMA_InitTypeDef structure */
DMA_InitTypeDef  DMA_InitStructure;
DMA_StructInit(&DMA_InitStructure);
```

7.2.4 DMA_Cmd函数

表 117 DMA_Cmd 函数

函数名	DMA_Cmd
函数原型	void DMA_Cmd(DMA_Channel_TypeDef* DMA_Channelx, FunctionalState NewState)
功能描述	使能（或禁能）特定的 DMA 通道
输入参数 1	DMA_Channelx：x 可为 1 到 7 来选择 DMA 通道。
输入参数 2	NewState：DMA 通道的新状态。 该参数可为：ENABLE 或 DISABLE

输出参数	无
返回参数	无
前提条件	无
调用函数	无

实例：

```
/* Enable DMA Channel7 */

DMA_Cmd(DMA_Channel7, ENABLE);
```

7.2.5 DMA_ITConfig函数

表 118 DMA_ITConfig 函数

函数名	DMA_ITConfig
函数原型	void DMA_ITConfig(DMA_Channel_TypeDef* DMA_Channelx, u32 DMA_IT, FunctionalState NewState)
功能描述	使能（或禁能）特定的 DMA 通道的中断
输入参数 1	DMA_Channelx：x 可为 1 到 7 来选择 DMA 频道
输入参数 2	DMA_IT：确定 DMA 通道的中断源是否被使能。使用运算符“|”可选择多个中断，参考 DMA_IT 章节以获得该参数的可取值
输入参数 3	NewState：指定的 DMA 通道中断的新状态。 该参数可为：ENABLE 或 DISABLE

输出参数	无
返回参数	无
前提条件	无
调用函数	无

DMA_IT

DMA_IT 使能（或使不能）DMA 通道的中断。以下给出了它的可用值，可单独或结合使用。

表 119 DMA_IT 值

DMA_IT	描述
DMA_IT_TC	传输结束中断屏蔽（mask）
DMA_IT_HT	半传输中断屏蔽（mask）
DMA_IT_TE	传输错误中断屏蔽（mask）

实例：

```
/* Enable DMA Channel5 complete transfer interrupt */

DMA_ITConfig(DMA_Channel5, DMA_IT_TC, ENABLE);
```

7.2.6 DMA_GetCurrDataCounter函数

表 120 DMA_GetCurrDataCounter 函数

函数名	DMA_GetCurrDataCounter
函数原型	u16 DMA_GetCurrDataCounter(DMA_Channel_TypeDef*

	DMA_Channelx)
功能描述	返回当前 DMA 各通道传输剩余的数据单元
输入参数	DMA_Channelx：x 可为 1 到 7 来选择 DMA 通道
输出参数	无
返回参数	当前 DMA 通道传输中剩余的数据单元的数量
前提条件	无
调用函数	无

实例：

```
/* Get  the number of remaining data units in the current DMA
Channel2 transfer */
u16 CurrDataCount;
CurrDataCount = DMA_GetCurrDataCounter(DMA_Channel2);
```

DMA_GetFlagStatus 函数

表 121 DMA_GetFlagStatus 函数

函数名	DMA_GetFlagStatus
函数原型	FlagStatus DMA_GetFlagStatus(u32 DMA_FLAG)
功能描述	检查特定的 DMA 通道的标志是否被置位了
输入参数	DMA_FLAG：指定需要检查的标志

	更多细节参考：DMA_FLAG
输出参数	无
返回参数	DMA_FLAG 的新状态（SET 或 RESET）
前提条件	无
调用函数	无

DMA_FLAG

DMA_FLAG 是用来定义将要被检查的标志类型。表 122 给出了它的描述。

表 122　　DMA_FLAG 定义

DMA_FLAG	描述
DMA_FLAG_GL1	通道 1 全局标志
DMA_FLAG_TC1	通道 1 传输结束标志
DMA_FLAG_HT1	通道 1 半传输标志
DMA_FLAG_TE1	通道 1 传输错误标志
DMA_FLAG_GL2	通道 2 全局标志
DMA_FLAG_TC2	通道 2 传输结束标志
DMA_FLAG_HT2	通道 2 半传输标志
DMA_FLAG_TE2	通道 2 传输错误标志
DMA_FLAG_GL3	通道 3 全局标志
DMA_FLAG_TC3	通道 3 传输结束标志
DMA_FLAG_HT3	通道 3 半传输标志

DMA_FLAG_TE3	通道 3 传输错误标志
DMA_FLAG_GL4	通道 4 全局标志
DMA_FLAG_TC4	通道 4 传输结束标志
DMA_FLAG_HT4	通道 4 半传输标志
DMA_FLAG_TE4	通道 4 传输错误标志
DMA_FLAG_GL5	通道 5 全局标志
DMA_FLAG_TC5	通道 5 传输结束标志
DMA_FLAG_HT5	通道 5 半传输标志
DMA_FLAG_TE5	通道 5 传输错误标志
DMA_FLAG_GL6	通道 6 全局标志
DMA_FLAG_TC6	通道 6 传输结束标志
DMA_FLAG_HT6	通道 6 半传输标志
DMA_FLAG_TE6	通道 6 传输错误标志
DMA_FLAG_GL7	通道 7 全局标志
DMA_FLAG_TC7	通道 7 传输结束标志
DMA_FLAG_HT7	通道 7 半传输标志
DMA_FLAG_TE7	通道 7 传输错误标志

实例：

/* Test if the DMA Channel6 half transfer interrupt flag is set or

not */

```
FlagStatus Status;

Status = DMA_GetFlagStatus(DMA_FLAG_HT6);
```

7.2.7 DMA_ClearFlag函数

表 123 DMA_ClearFlag 函数

函数名	DMA_ClearFlag
函数原型	void DMA_ClearFlag(u32 DMA_FLAG)
功能描述	清除 DMA 通道的挂起标志
输入参数	DMA_FLAG：需要清除的标志。使用“|”运算符可以清除多个标志 参考 *DMA_FLAG* 部分获得该参数的可取值的更多细节
输出参数	无
返回参数	无
前提条件	无
调用函数	无

实例：

```
/* Clear the DMA Channel3 transfer error interrupt pending bit */

DMA_ClearFlag(DMA_FLAG_TE3);
```

7.2.8 DMA_GetITStatus函数

表 124 DMA_GetITStatus 函数

函数名	DMA_GetITStatus
函数原型	ITStatus DMA_GetITStatus(u32 DMA_IT)
功能描述	检查特定的 DMA 通道的中断是否被发生
输入参数	DMA_IT：需要检查的 DMA 通道中断源 更多细节参考：*DMA_IT*
输出参数	无
返回参数	DMA_IT 的新状态（SET 或 RESET）
前提条件	无
调用函数	无

DMA_IT

DMA_IT 选择了将要被检查的中断。表 125 给出了它的描述。

表 125 DMA_IT 值

DMA_IT	描述
DMA_IT_GL1	通道 1 的全局中断
DMA_IT_TC1	通道 1 的传输结束中断
DMA_IT_HT1	通道 1 的半传输中断
DMA_IT_TE1	通道 1 的传输错误中断
DMA_IT_GL2	通道 2 的全局中断

DMA_IT_TC2	通道 2 的传输结束中断
DMA_IT_HT2	通道 2 的半传输中断
DMA_IT_TE2	通道 2 的传输错误中断
DMA_IT_GL3	通道 3 的全局中断
DMA_IT_TC3	通道 3 的传输结束中断
DMA_IT_HT3	通道 3 的半传输中断
DMA_IT_TE3	通道 3 的传输错误中断
DMA_IT_GL4	通道 4 的全局中断
DMA_IT_TC4	通道 4 的传输结束中断
DMA_IT_HT4	通道 4 的半传输中断
DMA_IT_TE4	通道 4 的传输错误中断
DMA_IT_GL5	通道 5 的全局中断
DMA_IT_TC5	通道 5 的传输结束中断
DMA_IT_HT5	通道 5 的半传输中断
DMA_IT_TE5	通道 5 的传输错误中断
DMA_IT_GL6	通道 6 的全局中断
DMA_IT_TC6	通道 6 的传输结束中断
DMA_IT_HT6	通道 6 的半传输中断
DMA_IT_TE6	通道 6 的传输错误中断
DMA_IT_GL7	通道 7 的全局中断
DMA_IT_TC7	通道 7 的传输结束中断

DMA_IT_HT7	通道 7 的半传输中断
DMA_IT_TE7	通道 7 的传输错误中断

实例：

```
/* Test if the DMA Channel7 transfer complete interrupt has occurred
or not */
ITStatus Status;
Status = DMA_GetITStatus(DMA_IT_TC7);
```

7.2.9 DMA_ClearITPendingBit函数

表 126 DMA_ClearITPendingBit 函数

函数名	DMA_ClearITPendingBit
函数原型	void DMA_ClearITPendingBit(u32 DMA_IT)
功能描述	清除 DMA 通道的中断挂起位
输入参数	DMA_IT：需要清除的 DMA 通道的挂起位。使用“|”运算符可以清除多个中断。 参考 *DMA_FLAG* 部分获得该参数的可取值的更多细节
输出参数	无
返回参数	无

前提条件	无
调用函数	无

实例：

```
/* Clear the DMA Channel5 global interrupt pending bit */

DMA_ClearITPendingBit(DMA_IT_GL5);
```

8 外部中断/事件控制器（EXTI）

外部中断/ 事件控制器（EXTI）包含高达 19 个边沿探测器，它被用来产生事件/ 中断请求。每个输入行都可以被独立配置来选择类型(脉冲或挂起)和相应的触发器事件(上升 ,下降或两者一起)。每一行都可以被独立屏蔽（masked ）。一个挂起寄存器保持了中断请求的状态。

*8.1 小节：EXTI 寄存器结构*描述了 EXTI 固件库的数据结构。

*8.2 小节：固件库函数*给出了该部件的所有库函数。

8.1 EXTI寄存器结构

EXTI 寄存器结构，EXTI_TypeDef 定义在 stm32f10x_map.h 中：

```
typedef struct
{
  vu32 IMR;
  vu32 EMR;
  vu32 RTSR;
  vu32 FTSR;
  vu32 SWIER;
  vu32 PR;
} EXTI_TypeDef;
```

表 127 EXTI 寄存器

寄存器	描述
IMR	中断屏蔽寄存器
EMR	事件屏蔽寄存器
RTSR	上升沿触发选择寄存器
FTSR	下降沿触发选择寄存器
SWIR	软件中断事件寄存器
PR	挂起寄存器

EXTI 部件也在同一个文件的进行如下声明：

```
...
#define PERIPH_BASE              ((u32)0x40000000)
#define APB1PERIPH_BASE          PERIPH_BASE
#define APB2PERIPH_BASE          (PERIPH_BASE + 0x10000)
#define AHBPERIPH_BASE           (PERIPH_BASE + 0x20000)
...
#define EXTI_BASE                (APB2PERIPH_BASE + 0x0400)
#ifndef DEBUG
...
#ifdef _EXTI
  #define EXTI                   ((EXTI_TypeDef *) EXTI_BASE)
```

```
#endif /*_EXTI */

...

#else     /* DEBUG */

...

#ifdef _EXTI

   EXT EXTI_TypeDef                  *EXTI;

#endif /*_EXTI */

...

#endif
```

当使用 Debug 模式时，EXTI 指针在 stm32f10x_lib.c 文件中进行初始化：

```
#ifdef _EXTI

EXTI = (EXTI_TypeDef *)    EXTI_BASE;

#endif /*_EXTI */
```

为了进入 EXTI 寄存器，_EXTI 必须在 stm32f10x_conf.h 进行如下定义：

```
#define _EXTI
```

8.2 固件库函数

表 128 列出了 EXTI 部件的所有固件库函数

表 128 EXTI 固件库函数

函数名	描述
EXTI_DeInit	将 EXTI 外设寄存器复位到默认值
EXTI_Init	根据 EXIT_InitStruct 中指定的参数初始化 EXTI 外设
EXTI_StructInit	将 EXTI_InitStruct 的每个成员置为缺省值
EXTI_GenerateSWInterrupt	产生一个软件中断
EXTI_GetFlagStatus	检查特定的 EXTI 线标志是否被置位了
EXTI_ClearFlag	清除 EXTI 的线挂起标志
EXTI_GetITStatus	检查特定的 EXTI 线是否被置位有效电平
EXTI_ClearITPendingBit	清除 EXTI 的线挂起位

8.2.1 EXTI_DeInit函数

表 129 EXTI_DeInit 函数

函数名	EXTI_DeInit
函数原型	void EXTI_DeInit(void)
功能描述	将 EXTI 外设寄存器复位到默认值

输入参数	无
输出参数	无
返回参数	无
前提条件	无
调用函数	无

实例：

```
/* Resets the EXTI registers to their default reset value */

EXTI_DeInit();
```

8.2.2 EXTI_Init函数

表 130 EXTI_Init 函数

函数名	EXTI_Init
函数原型	void EXTI_Init(EXTI_InitTypeDef* EXTI_InitStruct)
功能描述	根据 EXIT_InitStruct 中指定的参数初始化 EXTI 外设
输入参数	EXTI_InitStruct：指向 EXTI_InitTypeDef 的结构，它包含特定 EXTI 的配置信息。 参考 EXTI_InitTypeDef 结构 以获得该参数可取值的更多信息
输出参数	无

返回参数	无
前提条件	无
调用函数	无

EXTI_InitTypeDef 结构

EXTI_InitTypeDef 结构在 stm32f10x_exti.h 中进行定义：

```
typedef struct
{
  u32 EXTI_Line;
  EXTIMode_TypeDef EXTI_Mode;
  EXTIrigger_TypeDef EXTI_Trigger;
  FunctionalState EXTI_LineCmd;
} EXTI_InitTypeDef;
```

EXTI_Line

EXTI_Line 选择将被使能的（或未被使能的）外部线。表 131 给出了该成员的可用值。

表 131 EXTI_Line 值

EXTI_Line	描述
EXTI_Line0	外部中断线 0
EXTI_Line1	外部中断线 1

EXTI_Line2	外部中断线 2
EXTI_Line3	外部中断线 3
EXTI_Line4	外部中断线 4
EXTI_Line5	外部中断线 5
EXTI_Line6	外部中断线 6
EXTI_Line7	外部中断线 7
EXTI_Line8	外部中断线 8
EXTI_Line9	外部中断线 9
EXTI_Line10	外部中断线 10
EXTI_Line11	外部中断线 11
EXTI_Line12	外部中断线 12
EXTI_Line13	外部中断线 13
EXTI_Line14	外部中断线 14
EXTI_Line15	外部中断线 15
EXTI_Line16	外部中断线 16
EXTI_Line17	外部中断线 17
EXTI_Line18	外部中断线 18

EXTI_Mode

EXTI_Mode 为使能的线配置模式。表 132 给出了它的可用值。

表 132 EXTI_Mode 值

EXTI_Mode	描述
EXTI_Mode_Event	EXTI 线被配置为事件请求
EXTI_Mode_Interrupt	EXTI 线被配置为中断请求

EXTI_Trigger

EXTI 为使能线配置触发信号有效边沿。表 133 给出了该成员的可用值。

表 133 EXTI_Trigger 值

EXTI_Trigger	描述
EXTI_Trigger_Falling	中断请求被配置为输入线的下降沿
EXTI_Trigger_Rising	中断请求被配置为输入线的上升沿
EXTI_Trigger_Rising_Falling	中断请求被配置为输入线的上升沿和下降沿

EXTI_LineCmd

这个成员被用来定义被选择线的新状态。它可以被设置为 ENABLE 或 DISABLE。

实例：

```
/* Enables external lines 12 and 14 interrupt generation on falling
edge */
EXTI_InitTypeDef EXTI_InitStructure;
EXTI_InitStructure.EXTI_Line = EXTI_Line12 | EXTI_Line14;
```

```
EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;

EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;

EXTI_InitStructure.EXTI_LineCmd = ENABLE;

EXTI_Init(&EXTI_InitStructure);
```

8.2.3 EXTI_Struct函数

表 134 EXTI_Struct 函数

函数名	EXTI_Struct
函数原型	void EXTI_StructInit(EXTI_InitTypeDef*EXTI_InitStruct)
功能描述	将 EXTI_InitStruct 成员置为默认值
输入参数	EXTI_InitStruct：指向 EXTI_InitTypeDef 结构，该结构将被初始化。
输出参数	无
返回参数	无
前提条件	无
调用函数	无

表 135 给出了 EXTI_InitStruct 成员的默认值：

表 135 EXTI_InitStruct 缺省值

成员	默认值
EXTI_Line	EXTI_LineNone
EXTI_Mode	EXTI_Mode_Interrupt

EXTI_Trigger	EXTI_Trigger_Falling
EXTI_LineCmd	DISABLE

实例：

```
/* Initialize the EXTI Init Structure parameters */
EXTI_InitTypeDef EXTI_InitStructure;
EXTI_StructInit(&EXTI_InitStructure);
```

8.2.4 EXTI_GenerateSWInterrupt函数

表 136 EXTI_GenerateSWInterrupt 函数

函数名	EXTI_GenerateSWInterrupt
函数原型	void EXTI_GenerateSWInterrupt(u32 EXTI_Line)
功能描述	产生一个软件中断
输入参数	EXTI_Line：将被使能（或禁能）的 EXTI 线。参考 *EXTI_Line* 章节以获得该参数可取值的更多细节
输出参数	无
返回参数	无
前提条件	无
调用函数	无

实例：

```
/* Generate a software interrupt request */
EXTI_GenerateSWInterrupt(EXTI_Line6);
```

8.2.5 EXTI_GetFlagStatus函数

表 137 EXTI_GetFlagStatus 函数

函数名	EXTI_GetFlagStatus
函数原型	FlagStatus EXTI_GetFlagStatus(u32 EXTI_Line)
功能描述	检查特定的 EXTI 行标志是否被置位
输入参数	EXTI_Line：将要检查的 EXTI 线标志 参考 *EXTI_Line* 章节以获得该参数可取值的更多细节
输出参数	无
返回参数	EXTI_Line 的新状态（SET 或 RESET）
前提条件	无
调用函数	无

实例：

```
/* Get the status of EXTI line 8 */
FlagStatus EXTIStatus;
```

EXTIStatus = EXTI_GetFlagStatus(EXTI_Line8);

8.2.6 EXTI_ClearFlag函数

表 138 EXTI_ClearFlag 函数

函数名	EXTI_ClearFlag
函数原型	void EXTI_ClearFlag(u32 EXTI_Line)
功能描述	清除 EXTI 线的挂起标志
输入参数	EXTI_Line：将被清除的 EXTI 线标志 参考 *EXTI_Line* 章节以获得该参数可取值的更多细节
输出参数	无
返回参数	无
前提条件	无
调用函数	无

实例：

/* Clear the EXTI line 2 pending flag */

EXTI_ClearFlag(EXTI_Line2);

8.2.7 EXTI_GetITStatus函数

表 139 EXTI_GetITStatus 函数

函数名	EXTI_GetITStatus
函数原型	ITStatus EXTI_GetITStatus(u32 EXTI_Line)
功能描述	检查特定的 EXTI 行是否被置位有效电平
输入参数	EXTI_Line：需要检查的挂起位。 更多细节参考：EXTI_Line
输出参数	无
返回参数	EXTI_Line 的新状态（SET 或 RESET）
前提条件	无
调用函数	无

实例：

```
/* Get the status of EXTI line 8 */

ITStatus EXTIStatus;

EXTIStatus = EXTI_GetITStatus(EXTI_Line8);
```

8.2.8 EXTI_ClearITPendingBit函数

表 140 EXTI_ClearITPendingBit 函数

函数名	EXTI_ClearITPendingBit
函数原型	void EXTI_ClearITPendingBit(u32 EXTI_Line)
功能描述	清除 EXTI 的行挂起位
输入参数	EXTI_Line：将被清除的 EXTI 行挂起位

	更多细节参考：EXTI_Line
输出参数	无
返回参数	无
前提条件	无
调用函数	无

实例：

```
/* Clears the EXTI line 2 interrupt pending bit */
EXTI_ClearITpendingBit(EXTI_Line2);
```

9 闪存部件（Flash memory）

*9.1 小节：FLASH 寄存器结构*描述了 FLASH 固件库使用的数据结构。

*9.2 小节：固件库函数*给出了该部件的全部库函数。

9.1 FLASH寄存器结构

FLASH 寄存器结构，FLASH_TypeDef 和 OB_TypeDef，都在 stm32f10x_map.h 中进行如下定义：

```
typedef struct
{
  vu32 ACR;
  vu32 KEYR;
```

```
  vu32 OPTKEYR;
  vu32 SR;
  vu32 CR;
  vu32 AR;
  vu32 RESERVED;
  vu32 OBR;
  vu32 WRPR;
} FLASH_TypeDef;
typedef struct
{
  vu16 RDP;
  vu16 USER;
  vu16 Data0;
  vu16 Data1;
  vu16 WRP0;
  vu16 WRP1;
  vu16 WRP2;
  vu16 WRP3;
} OB_TypeDef;
```

表 141 和 142 分别列出了 FLASH 寄存器和选项字节寄存器（OB）。

表 141. FLASH 寄存器

寄存器	描述
ACR	闪存进入控制寄存器
KEYR	FPEC 关键寄存器
OPTKEYR	选项字节关键寄存器
SR	Flash 状态寄存器
CR	Flash 控制寄存器
AR	Flash 地址寄存器
OBR	选项字节及状态寄存器
WRPR	选项字节写保护寄存器

表 142 可选字节寄存器（OB）

寄存器	描述
RDP	读出选项字节
USER	用户选项字节
Data0	数据 0 选项字节
Data1	数据 1 选项字节
WRP0	写保护 0 的选项字节
WRP1	写保护 1 的选项字节
WRP2	写保护 2 的选项字节
WRP3	写保护 3 的选项字节

FLASH 部件在 stm32f10x_map.h 申明：

```
/* Flash registers base address */
#define FLASH_BASE                ((u32)0x40022000)
/* Flash Option Bytes base address */
#define OB_BASE                   ((u32)0x1FFFF800)
#ifndef DEBUG
...
#ifdef _FLASH
  #define FLASH                   ((FLASH_TypeDef *) FLASH_BASE)
  #define OB                      ((OB_TypeDef *) OB_BASE)
#endif /*_FLASH */
...
#else     /* DEBUG */
...
#ifdef _FLASH
  EXT FLASH_TypeDef               *FLASH;
  EXT OB_TypeDef                  *OB;
#endif /*_FLASH */
...
#endif
```

当使用 Debug 模式时，FLASH 和 OB 指针都在 stm32f10x_lib.c 中进行初始化：

```
#ifdef _FLASH
FLASH = (FLASH_TypeDef *)  FLASH_BASE;
OB = (OB_TypeDef *)  OB_BASE;
#endif /*_FLASH */
```

为了访问 FLASH 寄存器，_FLASH 必须在 stm32f10x_conf.h 中进行如下定义：

```
#define _FLASH
```

缺省情况下，只有执行 FLASH 配置（延时，预取，半周期）的函数被使能（见表 143）

为了使能 FLASH 程序/ 擦除/ 保护函数，_FLASH_PROG 必须在 stm32f10x_conf.h 中进行如下定义：

```
#define _FLASH_PROG
```

9.2 固件库函数

表 143 FLASH 的所有库函数

函数名	描述

FLASH_SetLatency	设置代码延时值
FLASH_HalfCycleAccessCmd	使能（或禁能）半周期 FLASH 访问
FLASH_PrefetchBufferCmd	使能（或禁能）预取缓冲区
FLASH_Unlock	解锁 FLASH 程序擦除控制器
FLASH_Lock	锁定 FLASH 程序擦除控制器
FLASH_ErasePage	擦除一个特定的 FLASH 页
FLASH_EraseAllPages	擦除所有 FLASH 页
FLASH_EraseOptionBytes	擦除 FLASH 选项字节
FLASH_ProgramWord	在特定地址写（program）一个字
FLASH_ProgramHalfWord	在特定地址写（program）一个半字
FLASH_ProgramOptionByteData	在特定的选项字节数据地址写（program）一个半字
FLASH_EnableWriteProtection	对需要写保护的页进行写保护
FLASH_ReadOutProtection	使能（或禁能）读出保护
FLASH_UserOptionByteConfig	写 FLASH 用户选项字节：IWDG_SW / RST_STOP / RST_STDBY
FLASH_GetUserOptionByte	返回 FLASH 用户选项字节值
FLASH_GetWriteProtectionOptionByte	返回 FLASH 写保护选项字节寄存器值
FLASH_GetReadOutProtectionStatus	检查 FLASH 读取保护状态是否被置位

FLASH_GetPrefetchBufferStatus	检查 FLASH 预取缓冲区状态是否置位
FLASH_ITConfig	使能（或禁能）特定的 FLASH 中断
FLASH_GetFlagStatus	检查特定的 FLASH 标志是否被置位
FLASH_ClearFlag	清除 FLASH 挂起标志
FLASH_GetStatus	返回 FLASH 状态
FLASH_WaitForLastOperation	等待一个 Flash 操作来结束或一个 TIMEOUT 事件发生

9.2.1 FLASH_SetLatency函数

表 144 FLASH_SetLatency 函数

函数名	FLASH_SetLatency
函数原型	void FLASH_SetLatency(u32 FLASH_Latency)
功能描述	设置代码延时值
输入参数	FLASH_Latency 指定了 FLASH 延时值。 参考 :*FLASH_Latency* 部分获得该参数可取值的更多细节
输出参数	无
返回参数	无
前提条件	无
调用函数	无

FLASH_Latency

FLASH_Latency 是用来配置 FLASH 延时值的。表 145 给出它的可用值

表 145 FLASH_Latency 值

FLASH_Latency	描述
FLASH_Latency_0	0 个延时周期
FLASH_Latency_1	1 个延时周期
FLASH_Latency_2	2 个延时周期

实例：

Example:

/* Configure the Latency cycle: Set 2 Latency cycles */

FLASH_SetLatency(FLASH_Latency_2);

9.2.2 FLASH_HalfCycleAccessCmd函数

表 146 FLASH_HalfCycleAccessCmd 函数

函数名	FLASH_HalfCycleAccessCmd
函数原型	void FLASH_HalfCycleAccessCmd(u32 FLASH_HalfCycleAccess)
功能描述	使能（或禁能）半周期 Flash 访问
输入参数	FLASH_HalfCycleAccess：FLASH 半周期模式 参考：FLASH_HalfCycleAccess 部分获得该参数可取值的更多信

	息
输出参数	无
返回参数	无
前提条件	无
调用函数	无

FLASH_HalfCycleAccess

FLASH_HalfCycle 是用来选择 FLASH 半周期访问模式的。表 147 给出了该参数的可用值。

表 147 FLASH_HalfCycleAccess 值

FLASH_HalfCycleAccess	描述
FLASH_HalfCycleAccess_Enable	使能半周期访问
FLASH_HalfCycleAccess_Disable	禁能半周期访问

实例：

```
/* Enable the Half Cycle Flash access */

FLASH_HalfCycleAccessCmd(FLASH_HalfCycleAccess_Enable);
```

9.2.3 FLASH_PrefetchBufferCmd函数

表 148 FLASH_PrefetchBufferCmd 函数

函数名	FLASH_PrefetchBufferCmd
函数原型	void FLASH_PrefetchBufferCmd(u32 FLASH_PrefetchBuffer)
功能描述	使能（或禁能）预取缓冲区

输入参数	FLASH_PrefetchBuffer：领取缓冲区状态。 参考 :FLASH_PrefetchBuffer 部分获得该参数可取值的更多信息
输出参数	无
返回参数	无
前提条件	无
调用函数	无

FLASH_PrefetchBuffer

FLASH_PrefetchBuffer 是用来选择 FLASH 预取缓冲区的状态。表 149 给出了该参数的可用值。

表 149 FLASH_PrefetchBuffer 值

FLASH_PrefetchBuffer	描述
FLASH_PrefetchBuffer_Enable	预取缓冲区使能
FLASH_PrefetchBuffer_Disable	预取缓冲区禁能

实例：

```
/* Enable The Prefetch Buffer */

FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
```

9.2.4 FLASH_Unlock函数

表 150 FLASH_Unlock 函数

函数名	FLASH_Unlock
函数原型	void FLASH_Unlock(void)

功能描述	解锁 FLASH 程序擦除控制器
输入参数	无
输出参数	无
返回参数	无
前提条件	无
调用函数	无

实例：

```
/* Unlocks the Flash */

FLASH_Unlock();
```

9.2.5 FLASH_Lock函数

表 151 FLASH_Lock 函数

函数名	FLASH_Lock
函数原型	void FLASH_Lock(void)
功能描述	锁定 FLASH 程序擦除控制器
输入参数	无
输出参数	无
返回参数	无
前提条件	无
调用函数	无

实例：

```
/* Locks the Flash */

FLASH_Lock();
```

FLASH_ErasePage 函数

表 152　FLASH_ErasePage 函数

函数名	FLASH_ErasePage
函数原型	FLASH_Status FLASH_ErasePage(u32 Page_Address)
功能描述	擦除一个 FLASH 页
输入参数	FLASH_Page：需要擦除的页
输出参数	无
返回参数	擦除操作状态
前提条件	无
调用函数	无

实例：

```
/* Erases the Flash Page 0 */

FLASH_Status status = FLASH_COMPLETE;

status = FLASH_ErasePage(0x08000000);
```

9.2.6 FLASH_EraseAllPages函数

表 153 FLASH_EraseAllPages 函数

函数名	FLASH_EraseAllPages
函数原型	FLASH_Status FLASH_EraseAllPages(void)
功能描述	擦除所有的 FLASH 页
输入参数	无
输出参数	无
返回参数	擦除操作状态
前提条件	无
调用函数	无

实例：

```
/* Erases the Flash */

FLASH_Status status = FLASH_COMPLETE;

status = FLASH_EraseAllPages();
```

9.2.7 FLASH_EraseOptionBytes函数

表 154 FLASH_EraseOptionBytes 函数

函数名	FLASH_EraseOptionBytes
函数原型	FLASH_Status FLASH_EraseOptionBytes(void)
功能描述	擦除 FLASH 选项字节

输入参数	无
输出参数	无
返回参数	擦除操作状态
前提条件	无
调用函数	无

实例：

```
/* Erases the Flash Option Bytes */

FLASH_Status status = FLASH_COMPLETE;

status = FLASH_EraseOptionBytes();
```

9.2.8 FLASH_ProgramWord函数

表 155 FLASH_ProgramWord 函数

函数名	FLASH_ProgramWord
函数原型	FLASH_Status FLASH_ProgramWord(u32 Address, u32 Data)
功能描述	在特定地址编程一个字
输入参数 1	Address：将要编程的地址
输入参数 2	Data：指定被编程的数据
输出参数	无
返回参数	程序操作状态

前提条件	无
调用函数	无

实例：

```
/* Writes the Data1 at the Address1 */

FLASH_Status status = FLASH_COMPLETE;

u32 Data1 = 0x1234567;

u32 Address1 = 0x8000000;

status = FLASH_ProgramWord(Address1, Data1);
```

9.2.9 FLASH_ProgramHalfWord函数

表 156 FLASH_ProgramHalfWord 函数

函数名	FLASH_ProgramHalfWord
函数原型	FLASH_Status FLASH_ProgramHalfWord(u32 Address, u16 Data)
功能描述	在特定地址编程一个半字
输入参数 1	Address：被编程的地址
输入参数 2	Data：指定被编程的半字数据
输出参数	无
返回参数	程序操作状态
前提条件	无

调用函数	无

实例：

```
/* Writes the Data1 at the Address1 */
FLASH_Status status = FLASH_COMPLETE;
u16 Data1 = 0x1234;
u32 Address1 = 0x8000004;
status = FLASH_ProgramHalfWord(Address1, Data1);
```

9.2.10 FLASH_ProgramOptionByteData函数

表 157 FLASH_ProgramOptionByteData 函数

函数名	FLASH_ProgramOptionByteData
函数原型	FLASH_Status FLASH_ProgramOptionByteData(u32 Address, u8 Data)
功能描述	在指定的选项字节数据地址上编写一个半字
输入参数 1	Address :被编程的地址 ,该参数可为 :0x1FFFF804 或 0x1FFFF806
输入参数 2	Data：被编写的数据
输出参数	无
返回参数	无
前提条件	无
调用函数	无

实例：

```
/* Writes the Data1 at the Address1 */
FLASH_Status status = FLASH_COMPLETE;
u8 Data1 = 0x12;
u32 Address1 = 0x1FFFF804;
status = FLASH_ProgramOptionByteData(Address1, Data1);
```

9.2.11 FLASH_EnableWriteProtection函数

表 158 FLASH_EnableWriteProtection 函数

函数名	FLASH_EnableWriteProtection
函数原型	FLASH_Status FLASH_EnableWriteProtection(u32 FLASH_Pages)
功能描述	对需要写保护的页进行写保护
输入参数	FLASH_Pages：需要写保护的页的地址。 参考：*FLASH_Pages* 以获得该参数可取值的更多信息
输出参数	无
返回参数	写保护操作状态
前提条件	无
调用函数	无

FLASH_Pages

FLASH_Pages 是用来配置 FLASH 写保护的页的。表 159 给出了它的可用值。

表 159　　FLASH_Pages 值

FLASH_Pages	描述
FLASH_WRProt_Pages0to3	0 到 3 页的写保护
FLASH_WRProt_Pages4to7	4 到 7 页的写保护
FLASH_WRProt_Pages8to11	8 到 11 页的写保护
FLASH_WRProt_Pages12to15	12 到 15 页的写保护
FLASH_WRProt_Pages16to19	16 到 19 页的写保护
FLASH_WRProt_Pages20to23	20 到 23 页的写保护
FLASH_WRProt_Pages24to27	24 到 27 页的写保护
FLASH_WRProt_Pages28to31	28 到 31 页的写保护
FLASH_WRProt_Pages32to35	32 到 35 页的写保护
FLASH_WRProt_Pages36to39	36 到 39 页的写保护
FLASH_WRProt_Pages40to43	40 到 43 页的写保护
FLASH_WRProt_Pages44to47	44 到 47 页的写保护
FLASH_WRProt_Pages48to51	48 到 51 页的写保护
FLASH_WRProt_Pages52to55	52 到 55 页的写保护
FLASH_WRProt_Pages56to59	56 到 59 页的写保护
FLASH_WRProt_Pages60to63	60 到 63 页的写保护
FLASH_WRProt_Pages64to67	64 到 67 页的写保护
FLASH_WRProt_Pages68to71	68 到 71 页的写保护

FLASH_WRProt_Pages72to75	72 到 75 页的写保护
FLASH_WRProt_Pages76to79	76 到 79 页的写保护
FLASH_WRProt_Pages80to83	80 到 83 页的写保护
FLASH_WRProt_Pages84to87	84 到 87 页的写保护
FLASH_WRProt_Pages88to91	88 到 91 页的写保护
FLASH_WRProt_Pages92to95	92 到 95 页的写保护
FLASH_WRProt_Pages96to99	96 到 99 页的写保护
FLASH_WRProt_Pages100to103	100 到 103 页的写保护
FLASH_WRProt_Pages104to107	104 到 107 页的写保护
FLASH_WRProt_Pages108to111	108 到 111 页的写保护
FLASH_WRProt_Pages112to115	112 到 115 页的写保护
FLASH_WRProt_Pages116to119	116 到 119 页的写保护
FLASH_WRProt_Pages120to123	120 到 123 页的写保护
FLASH_WRProt_Pages124to127	124 到 127 页的写保护
FLASH_WRProt_AllPages	所有页的写保护

实例：

```
/* Protects the Pages0to3 and Pages108to111 */

FLASH_Status status = FLASH_COMPLETE;

status = FLASH_EnableWriteProtection

(FLASH_WRProt_Pages0to3|FLASH_WRProt_Pages108to111);
```

9.2.12 FLASH_ReadOutProtection函数

表 160 FLASH_ReadOutProtection 函数

函数名	FLASH_ReadOutProtection
函数原型	FLASH_Status FLASH_ReadOutProtection(FunctionalState NewState)
功能描述	使能（或禁能）读出保护
输入参数	NewState：读出保护的新状态。这个参数可被设置为：ENABLE 或 DISABLE
输出参数	无
返回参数	保护操作状态
前提条件	无
调用函数	无

实例：

```
/* Disables the ReadOut Protection */
FLASH_Status status = FLASH_COMPLETE;
```

9.2.13 FLASH_UserOptionByteConfig函数

表 161 FLASH_UserOptionByteConfig 函数

函数名	FLASH_UserOptionByteConfig
函数原型	FLASH_Status FLASH_UserOptionByteConfig(u16

	OB_IWDG, u16 OB_STOP, u16 OB_STDBY)
功能描述	编写 FLASH 用户选项字节：IWDG_SW / RST_STOP/RST_STDBY.
输入参数 1	OB_IWDG：选择 IWDG 模式 请参考 *OB_IWDG* 部分以获得关于该参数的值的详细信息
输入参数 2	OB_STOP：当进入 STOP 模式时重置事件 请参考 *OB_IWDG* 部分以获得关于该参数的值的详细信息
输入参数 3	OB_STDBY：当进入待命模式时重置事件 请参考 *OB_IWDG* 部分以获得关于该参数的值的详细信息
输出参数	无
返回参数	选项字节程序状态
前提条件	无
调用函数	无

OB_IWDG

这个参数用来配置 IWDG 模式。表 162 给出它的可用值。

表 162 OB_IWDG 值

OB_IWDG	描述
OB_IWDG_SW	软件 IWDG 被选择
OB_IWDG_HW	硬件 IWDG 被选择

OB_STOP

该参数指定当进入 STOP 模式时是否产生一个复位。表 163 给出它的可用值。

表 163 OB_STOP 值

OB_STOP	描述
OB_STOP_NoRST	当进入 STOP 模式时没有产生复位
OB_STOP_RST	当进入 STOP 模式时产生复位

OB_STDBY

OB_STDBY 用来区别当进入 STANDBY 模式时是否产生一个复位。表 164 给出了它的可用值。

表 164 OB_STDBY

OB_STDBY	描述
OB_STDBY_NoRST	当进入 STANDBY 模式时没有产生复位
OB_STDBY_RST	当进入 STANDBY 模式时产生复位

实例：

```
/* Option Bytes Configuration: software watchdog, Reset generation
when entering in STOP and No reset generation when entering in
STANDBY */
FLASH_Status status = FLASH_COMPLETE;
status = FLASH_UserOptionByteConfig(OB_IWDG_SW, OB_STOP_RST,
OB_STDBY_NoRST);
```

9.2.14 FLASH_GetUserOptionByte函数

表 165 FLASH_GetUserOptionByte 函数

函数名	FLASH_GetUserOptionByte
函数原型	u32 FLASH_GetUserOptionByte(void)
功能描述	返回 FLASH 用户选项字节值
输入参数	无
输出参数	无
返回参数	FLASH 用户选项字节值：IWDG_SW(Bit0)，RST_STOP(Bit1) 及 RST_STDBY(Bit2).
前提条件	无
调用函数	无

实例：

```
/* Gets the user option byte values */
u32 UserByteValue = 0x0;
u32 IWDGValue = 0x0, RST_STOPValue = 0x0, RST_STDBYValue = 0x0;
UserByteValue = FLASH_GetUserOptionByte();
IWDGValue = UserByteValue & 0x0001;
RST_STOPValue = UserByteValue & 0x0002;
RST_STDBYValue = UserByteValue & 0x0004;
```

9.2.15 FLASH_GetWriteProtectionOptionByte函数

表 166 FLASH_GetWriteProtectionOptionByte 函数

函数名	FLASH_GetWriteProtectionOptionByte
函数原型	u32 FLASH_GetWriteProtectionOptionByte(void)
功能描述	返回 FLASH 写保护选项字节寄存器值
输入参数	无
输出参数	无
返回参数	FLASH 写保护选项字节寄存器值
前提条件	无
调用函数	无

实例：

```
/* Gets the Write Protection option byte values */

u32 WriteProtectionValue = 0x0;

WriteProtectionValue = FLASH_GetWriteProtectionOptionByte();
```

9.2.16 FLASH_GetReadOutProtectionStatus函数

表 167 FLASH_GetReadOutProtectionStatus 函数

函数名	FLASH_GetReadOutProtectionStatus
函数原型	FlagStatus FLASH_GetReadOutProtectionStatus(void)
功能描述	检查 FLASH 读取保护状态是否被置位了
输入参数	无

输出参数	无
返回参数	FLASH 读出保护状态（SET 或 RESET）
前提条件	无
调用函数	无

实例：

```
/* Gets the ReadOut Protection status */

FlagStatus status = RESET;

status = FLASH_GetReadOutProtectionStatus();
```

9.2.17 FLASH_GetPrefetchBufferStatus函数

表 168 FLASH_GetPrefetchBufferStatus 函数

函数名	FLASH_GetPrefetchBufferStatus
函数原型	FlagStatus FLASH_GetPrefetchBufferStatus(void)
功能描述	检查 FLASH 预取缓冲区状态是否被置位
输入参数	无
输出参数	无
返回参数	FLASH 预取缓冲区状态（SET 或 RESET）
前提条件	无
调用函数	无

实例：

```
/* Gets the Prefetch Buffer status */

FlagStatus status = RESET;

status = FLASH_GetPrefetchBufferStatus();
```

9.2.18 FLASH_ITConfig函数

表 169 FLASH_ITConfig 函数

函数名	FLASH_ITConfig
函数原型	void FLASH_ITConfig(u16 FLASH_IT, FunctionalState NewState)
功能描述	使能（或禁能）特定的 FLASH 中断
输入参数 1	FLASH_IT :被使能(或禁能)的 FLASH 中断源。参考 :*FLASH_IT*
输入参数 2	NewState：特定 FLASH 中断的新状态。 该参数可为：ENABLE 或 DISABLE
输出参数	无
返回参数	无
前提条件	无
调用函数	无

FLASH_IT

该参数是用来使能（或禁能）FLASH 中断的。以下值可以单独或组合使用：

表 170 FLASH_IT 值

FLASH_IT	描述
FLASH_IT_ERROR	FPEC 错误中断源
FLASH_IT_EOP	FLASH 操作结束中断源

实例：

```
/* Enables the EOP Interrupt source */

FLASH_ITConfig(FLASH_IT_EOP, ENABLE);
```

9.2.19 FLASH_GetFlagStatus函数

表 171 FLASH_GetFlagStatus 函数

函数名	FLASH_GetFlagStatus
函数原型	FlagStatus FLASH_GetFlagStatus(u16 FLASH_FLAG)
功能描述	检查特定的 FLASH 标志是否被置位
输入参数	FLASH_FLAG：需要检查的标志 参考：*FLASH_FLAG* 部分以获取该参数可取值的更多信息
输出参数	无
返回参数	无
前提条件	无
调用函数	无

FLASH_FLAG

FLASH 标记,可以依靠引用 FLASH_GetFlagStatus 函数来进行检查。

表 172 FLASH_FLAG 定义

FLASH_FLAG	描述
FLASH_FLAG_BSY	FLASH 忙标志
FLASH_FLAG_EOP	FLASH 操作结束标志
FLASH_FLAG_PGERR	FLASH 程序错误标志
FLASH_FLAG_WRPRTERR	FLASH 写保护错误标志
FLASH_FLAG_OPTERR	FLASH 选项字节错误标志

实例：

```
/* Checks whether the EOP Flag Status is SET or not */

FlagStatus status = RESET;

status = FLASH_GetFlagStatus(FLASH_FLAG_EOP);
```

9.2.20 FLASH_ClearFlag函数

表 173 FLASH_ClearFlag 函数

函数名	FLASH_ClearFlag
函数原型	void FLASH_ClearFlag(u16 FLASH_Flag)
功能描述	清除 FLASH 挂起标志
输入参数	FLASH_FLAG：需要清除的标志 参考：*FLASH_FLAG* 部分以获取该参数可取值的更多信息
输出参数	无

返回参数	无
前提条件	无
调用函数	无

FLASH_FLAG

FLASH 标志可以通过调用 FLASH_ClearFlag 函数来清除，其定义如下：

表 174 FLASH_FLAG 定义

FLASH_FLAG	描述
FLASH_FLAG_BSY	FLASH 忙的标记
FLASH_FLAG_EOP	FLASH 操作结束的标记
FLASH_FLAG_PGERR	FLASH 程序错误的标记
FLASH_FLAG_WRPRTERR	FLASH 页写保护错误的标记

实例：

```
/* Clears all flags */

FLASH_ClearFlag(FLASH_FLAG_BSY|FLASH_FLAG_EOP|FLASH_FLAG_PGER

|FLASH_FLAG_WRPRTERR);
```

9.2.21 FLASH_GetStatus函数

表 175 描述了 FLASH_GetStatus 函数

表 175 FLASH_GetStatus 函数

函数名	FLASH_GetStatus

函数原型	FLASH_Status FLASH_GetStatus(void)
功能描述	返回 FLASH 状态
输入参数	无
输出参数	无
返回参数	FLASH 装态：返回值可以为：FLASH_BUSY，FLASH_ERROR_PG 或 FLASH_ERROR_WRP 或 FLASH_COMPLETE
前提条件	无
调用函数	无

实例：

```
/* Check for the Flash status */

FLASH_Status status = FLASH_COMPLETE;

status = FLASH_GetStatus();
```

9.2.22 FLASH_WaitForLastOperation函数

表 176 FLASH_WaitForLastOperation 函数

函数名	FLASH_WaitForLastOperation
函数原型	FLASH_Status FLASH_WaitForLastOperation(u32 Timeout)
功能描述	等待一个 Flash 操作来结束或一个 TIMEOUT 事件发生
输入参数	无

输出参数	无
返回参数	返回正确的操作状态。 这个参数可以是：FLASH_BUSY, FLASH_ERROR_PG 或 FLASH_ERROR_WRP 或 FLASH_COMPLETE or FLASH_TIMEOUT
前提条件	无
调用函数	无

实例：

```
/* Waits for the Flash operation to be completed */
FLASH_Status status = FLASH_COMPLETE;
status = FLASH_WaitForLastOperation();
```

10 通用输入输出接口（GPIO）

GPIO 驱动可以用作以下一些用途，包括引脚配置，单个位的置位/复位，锁定机制，从端口引脚读取数据，以及向端口引脚写入数据。

*10.1 小节：GPIO 寄存器结构*描述了 GPIO 固件库中使用的数据结构

*10.2 小节：固件库函数*列出了该器件的固件库函数

10.1 GPIO寄存器结构

GPIO 寄存器结构，GPIO_TypeDef，是在 stm32f10x_map.h 中进行如下定义的：

```
typedef struct
```

```
{
   vu32 CRL;
   vu32 CRH;
   vu32 IDR;
   vu32 ODR;
   vu32 BSRR;
   vu32 BRR;
   vu32 LCKR;
} GPIO_TypeDef;
typedef struct
{
   vu32 EVCR;
   vu32 MAPR;
   vu32 EXTICR[4];
} AFIO_TypeDef;
```

表 177 给出了 GPIO 的所有寄存器

表 177 GPIO 寄存器

寄存器	描述
CRL	端口控制寄存器低字节
CRH	端口控制寄存器高字节

IDR	输入数据寄存器
ODR	输出数据寄存器
BSRR	位置位/复位寄存器
BRR	位复位寄存器
LCKR	锁定寄存器
EVCR	事件控制寄存器
MAPR	重映射调试及 AF 寄存器
EXTICR	EXTI 0 到 15 线配置寄存器

五个 GPIO 部件在 stm32f10x_map.h 中进行声明：

```
...
#define PERIPH_BASE             ((u32)0x40000000)
#define APB1PERIPH_BASE          PERIPH_BASE
#define APB2PERIPH_BASE          (PERIPH_BASE + 0x10000)
#define AHBPERIPH_BASE           (PERIPH_BASE + 0x20000)
...
#define AFIO_BASE               (APB2PERIPH_BASE + 0x0000)
#define GPIOA_BASE               (APB2PERIPH_BASE + 0x0800)
#define GPIOB_BASE               (APB2PERIPH_BASE + 0x0C00)
#define GPIOC_BASE               (APB2PERIPH_BASE + 0x1000)
#define GPIOD_BASE               (APB2PERIPH_BASE + 0x1400)
#define GPIOE_BASE               (APB2PERIPH_BASE + 0x1800)
```

```
#ifndef DEBUG

...

#ifdef _AFIO
  #define AFIO                    ((AFIO_TypeDef *) AFIO_BASE)
#endif /*_AFIO */

#ifdef _GPIOA
  #define GPIOA                   ((GPIO_TypeDef *) GPIOA_BASE)
#endif /*_GPIOA */

#ifdef _GPIOB
  #define GPIOB                   ((GPIO_TypeDef *) GPIOB_BASE)
#endif /*_GPIOB */

#ifdef _GPIOC
  #define GPIOC                   ((GPIO_TypeDef *) GPIOC_BASE)
#endif /*_GPIOC */

#ifdef _GPIOD
  #define GPIOD                   ((GPIO_TypeDef *) GPIOD_BASE)
#endif /*_GPIOD */

#ifdef _GPIOE
  #define GPIOE                   ((GPIO_TypeDef *) GPIOE_BASE)
#endif /*_GPIOE */

...
```

```
#else     /* DEBUG */

...

#ifdef _AFIO

   EXT AFIO_TypeDef                *AFIO;

#endif /*_AFIO */

#ifdef _GPIOA

   EXT GPIO_TypeDef                *GPIOA;

#endif /*_GPIOA */

#ifdef _GPIOB

   EXT GPIO_TypeDef                *GPIOB;

#endif /*_GPIOB */

#ifdef _GPIOC

   EXT GPIO_TypeDef                *GPIOC;

#endif /*_GPIOC */

#ifdef _GPIOD

   EXT GPIO_TypeDef                *GPIOD;

#endif /*_GPIOD */

#ifdef _GPIOE

   EXT GPIO_TypeDef                *GPIOE;

#endif /*_GPIOE */

...
```

```
#endif
```

当使用 Debug 模式时，_AFIO, _GPIOA, _GPIOB, _GPIOC, _GPIOD 及_GPIOE 指针必须在 stm32f10x_lib.c 中进行初始化：

```
#ifdef _GPIOA
  GPIOA = (GPIO_TypeDef *)  GPIOA_BASE;
#endif /*_GPIOA */
#ifdef _GPIOB
  GPIOB = (GPIO_TypeDef *)  GPIOB_BASE;
#endif /*_GPIOB */
#ifdef _GPIOC
  GPIOC = (GPIO_TypeDef *)  GPIOC_BASE;
#endif /*_GPIOC */
#ifdef _GPIOD
  GPIOD = (GPIO_TypeDef *)  GPIOD_BASE;
#endif /*_GPIOD */
#ifdef _GPIOE
  GPIOE = (GPIO_TypeDef *)  GPIOE_BASE;
#endif /*_GPIOE */
#ifdef _AFIO
```

```
  AFIO = (AFIO_TypeDef *)  AFIO_BASE;
#endif /*_AFIO */
```

为了访问 GPIO 寄存器，_GPIO, _AFIO, _GPIOA, _GPIOB, _GPIOC, _GPIOD 及 _GPIOE 必须在 stm32f10x_conf.h 中进行定义：

```
#define _GPIO
#define _GPIOA
#define _GPIOB
#define _GPIOC
#define _GPIOD
#define _GPIOE
#define _AFIO
```

10.2 固件库函数

表 178 GPIO 固件库函数

函数名	描述
GPIO_DeInit	将 GPIO 部件的寄存器值复位为默认值
GPIO_AFIODeInit	将备用功能（重映射，事件控制及 EXTI 配置）寄存器复位为默认值

GPIO_Init	按照 GPIO_InitStruct 的指定的参数初始化 GPIO 部件
GPIO_StructInit	为 GPIO_InitStruct 各成员赋默认值
GPIO_ReadInputDataBit	读取特定输入端口引脚
GPIO_ReadInputData	读取特定的 GPIO 输入数据端口
GPIO_ReadOutputDataBit	读取特定的输出数据端口位
GPIO_ReadOutputData	读取特定的 GPIO 输出数据端口
GPIO_SetBits	置位选定的数据端口位
GPIO_ResetBits	清除选定的数据端口位
GPIO_WriteBit	置位或清除选定的数据端口位
GPIO_Write	写数据到特定的 GPIO 数据端口
GPIO_PinLockConfig	锁定 GPIO 引脚配置寄存器
GPIO_EventOutputConfig	选择 GPIO 引脚用作事件输出
GPIO_EventOutputCmd	使能（或禁能）事件输出
GPIO_PinRemapConfig	改变特定引脚的映射
GPIO_EXTILineConfig	选择 GPIO 引脚用作 EXTI 线

10.2.1 GPIO_DeInit函数

表 179 GPIO_DeInit 函数

函数名	GPIO_DeInit
函数原型	void GPIO_DeInit(GPIO_TypeDef* GPIOx)
功能描述	将 GPIO 部件的寄存器复位为默认值
输入参数	GPIOx：x 可为 A 到 E 来选择特定的 GPIO
输出参数	无
返回参数	无
前提条件	无
调用函数	RCC_APB2PeriphResetCmd()

实例：

```
/* Resets the GPIOA peripheral registers to their default reset
values */
GPIO_DeInit(GPIOA);
```

10.2.2 GPIO_AFIODeInit函数

表 180 GPIO_AFIODeInit 函数

函数名	GPIO_AFIODeInit
函数原型	void GPIO_AFIODeInit(void)
功能描述	将备用功能（重映射，事件控制及 EXTI 配置）寄存器复位为默认值
输入参数	无
输出参数	无

返回参数	无
前提条件	无
调用函数	RCC_APB2PeriphResetCmd()

实例：

```
/* Resets the Alternate functions registers to their default reset
values */
GPIO_AFIODeInit();
```

10.2.3 GPIO_Init函数

表 181 GPIO_Init 函数

函数名	GPIO_Init
函数原型	void GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct)
功能描述	按照 GPIO_InitStruct 的特定参数初始化 GPIO 部件
输入参数 1	GPIOx：x 可为 A 到 E 来选择特定的 GPIO
输入参数 2	GPIO_InitStruct：指向 GPIO_InitTypeDef 结构的指针，它包含特定 GPIO 部件的配置信息。参考 *GPIO_InitTypeDef 结构*
输出参数	无
返回参数	无
前提条件	无

调用函数	无

GPIO_InitTypeDef 结构

GPIO_InitTypeDef 在 stm32f10x_gpio.h 中如下定义：

```
typedef struct
{
  u16 GPIO_Pin;
  GPIOSpeed_TypeDef GPIO_Speed;
  GPIOMode_TypeDef GPIO_Mode;
} GPIO_InitTypeDef;
```

GPIO_Pin

这个成员选择了需要配置的 GPIO 引脚。使用“|”可完成多引脚的配置。以下值得任何结合都可以使用：

表 182　GPIO_Pin 值

GPIO_Pin	描述
GPIO_Pin_None	没有引脚被选择
GPIO_Pin_0	引脚 0 被选择
GPIO_Pin_1	引脚 1 被选择
GPIO_Pin_2	引脚 2 被选择
GPIO_Pin_3	引脚 3 被选择
GPIO_Pin_4	引脚 4 被选择

GPIO_Pin_5	引脚 5 被选择
GPIO_Pin_6	引脚 6 被选择
GPIO_Pin_7	引脚 7 被选择
GPIO_Pin_8	引脚 8 被选择
GPIO_Pin_9	引脚 9 被选择
GPIO_Pin_10	引脚 10 被选择
GPIO_Pin_11	引脚 11 被选择
GPIO_Pin_12	引脚 12 被选择
GPIO_Pin_13	引脚 13 被选择
GPIO_Pin_14	引脚 14 被选择
GPIO_Pin_15	引脚 15 被选择
GPIO_Pin_All	所有引脚都被选择

GPIO_Speed

GPIO_Speed 用来配置选定引脚的速度的。表 183 给出了该成员的可用值：

表 183　　GPIO_Speed 值

GPIO_Speed	描述
GPIO_Speed_10MHz	最大输出频率=10MHz
GPIO_Speed_2MHz	最大输出频率=2MHz
GPIO_Speed_50MHz	最大输出频率=50MHz

GPIO_Mode

GPIO_Mode 是用来配置选定引脚的操作模式的。表 184 给出了它的可用值。

表 184 GPIO_Mode 值

GPIO_Mode	描述
GPIO_Mode_AIN	模拟输入
GPIO_Mode_IN_FLOATING	浮点输入
GPIO_Mode_IPD	下拉输入
GPIO_Mode_IPU	上拉输入
GPIO_Mode_Out_OD	开漏输出
GPIO_Mode_Out_PP	推拉输出
GPIO_Mode_AF_OD	开漏输出备用功能
GPIO_Mode_AF_PP	推拉输出备用功能

注意：

1 当一个引脚被配置为输入上拉模式或下拉模式，Px_BSRR 和 Px_BRR 寄存器将被使用。

2 GPIO_Mode 允许配置输入/输出两个方向，相应的输入/输出配置：bits[7:4] GPIO_Mode 配置 GPIO 方向，同时 bits [4:0]定义配置。GPIO 方向有如下索引：

– GPIO 在输入模式 = 0x00

- GPIO 在输出模式 = 0x01

表 185 显示了所有的 GPIO_Mode 索引和代码

表 185 GPIO_Mode 索引和代码

GPIO 方向	索引	模式	配置	模式代码
		GPIO_Mode_AIN	0x00	0x00

GPIO 输入	0x00	GPIO_Mode_FLOATING	0x04	0x04
		GPIO_Mode_IPD	0x08	0x28
		GPIO_Mode_IPU	0x08	0x48
GPIO 输出	0x01	GPIO_Mode_Out_OD	0x04	0x14
		GPIO_Mode_Out_PP	0x00	0x10
		GPIO_Mode_AF_OD	0x0C	0x1C
		GPIO_Mode_AF_PP	0x08	0x18

实例：

```
/* Configure all the GPIOA in Input Floating mode */

GPIO_InitTypeDef GPIO_InitStructure;

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_All;

GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;

GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;

GPIO_Init(GPIOA, &GPIO_InitStructure);
```

10.2.4 GPIO_StructInit函数

表 186 GPIO_StructInit 函数

函数名	GPIO_StructInit
函数原型	void GPIO_StructInit(GPIO_InitTypeDef* GPIO_InitStruct)
功能描述	为 GPIO_InitStruct 各成员赋缺省值

输入参数	GPIO_InitStruct：指向 GPIO_InitTypeDef 结构，该结构将被初始化
输出参数	无
返回参数	无
前提条件	无
调用函数	无

表 187 给出了 GPIO_InitStruct 的缺省值

表 187 GPIO_InitStruct 缺省值

成员	缺省值
GPIO_Pin	GPIO_Pin_All
GPIO_Speed	GPIO_Speed_2MHz
GPIO_Mode	GPIO_Mode_IN_FLOATING

实例：

```
/* Initialize the GPIO Init Structure parameters */

GPIO_InitTypeDef GPIO_InitStructure;

GPIO_StructInit(&GPIO_InitStructure);
```

10.2.5 GPIO_ReadInputDataBit函数

表 188 GPIO_ReadInputDataBit 函数

函数名	GPIO_ReadInputDataBit

函数原型	u8 GPIO_ReadInputDataBit(GPIO_TypeDef* GPIOx, u16 GPIO_Pin)
功能描述	从特定的输入引脚读取
输入参数 1	GPIOx：x 可以为 A 到 E 来选择特定的 GPIO
输入参数 2	GPIO_Pin：被读取的引脚位 参考 GPIO_Pin
输出参数	无
返回参数	输入端口引脚值
前提条件	无
调用函数	无

实例：

```
/* Reads the seventh pin of the GPIOB and store it in ReadValue
variable */
u8 ReadValue;
ReadValue = GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_7);
```

10.2.6 GPIO_ReadInputData函数

表 189 GPIO_ReadInputData 函数

函数名	GPIO_ReadInputData
函数原型	u16 GPIO_ReadInputData(GPIO_TypeDef* GPIOx)

功能描述	从特定的 GPIO 输入数据端口读取
输入参数	GPIOx：x 可为 A 到 E 来选择特定的 GPIO
输出参数	无
返回参数	GPIO 输入数据端口值
前提条件	无
调用函数	无

实例：

```
/*Read the GPIOC input data port and store it in ReadValue
variable*/
u16 ReadValue;
ReadValue = GPIO_ReadInputData(GPIOC);
```

10.2.7 GPIO_ReadOutputDataBit函数

表 190 GPIO_ReadOutputDataBit 函数

函数名	GPIO_ReadOutputDataBit
函数原型	u8 GPIO_ReadOutputDataBit(GPIO_TypeDef* GPIOx, u16 GPIO_Pin)
功能描述	从特定的数据输出端口位读取
输入参数 1	GPIOx：x 可以为 A 到 E 来选择特定的 GPIO
输入参数 2	GPIO_Pin：需要读取的端口位

	参考 GPIO_Pin
输出参数	无
返回参数	输出端口引脚值
前提条件	无
调用函数	无

实例：

```
/* Reads the seventh pin of the GPIOB and store it in ReadValue

variable */

u8 ReadValue;

ReadValue = GPIO_ReadOutputDataBit(GPIOB, GPIO_Pin_7);
```

10.2.8 GPIO_ReadOutputData函数

表 191 GPIO_ReadOutputData 函数

函数名	GPIO_ReadOutputData
函数原型	u16 GPIO_ReadOutputData(GPIO_TypeDef* GPIOx)
功能描述	从特定的 GPIO 输出数据端口读取数据
输入参数	GPIOx：x 可为 A 到 E 来来选择特定的 GPIO
输出参数	无
返回参数	GPIO 输出数据端口值
前提条件	无

调用函数	无

实例：

```
/* Read the GPIOC output data port and store it in ReadValue
variable */
u16 ReadValue;
ReadValue = GPIO_ReadOutputData(GPIOC);
```

10.2.9 GPIO_SetBits函数

表 192 GPIO_SetBits 函数

函数名	GPIO_SetBits
函数原型	void GPIO_SetBits(GPIO_TypeDef* GPIOx, u16 GPIO_Pin)
功能描述	置位选定的数据端口位
输入参数 1	GPIOx：x 可为 A 到 E 来选择特定的 GPIO
输入参数 2	GPIO_Pin：确定要写入的端口位。 该参数可为 GPIO_Pin_x 的任意组合，其中 x 为 0 到 15。 参考 *GPIO_Pin* 部分以获得该参数可取值的更多信息
输出参数	无
返回参数	无
前提条件	无
调用函数	无

实例：

```
/* Set the GPIOA port pin 10 and pin 15 */

GPIO_SetBits(GPIOA, GPIO_Pin_10 | GPIO_Pin_15);
```

10.2.10 GPIO_ResetBits函数

表 193　GPIO_ResetBits 函数

函数名	GPIO_ResetBits
函数原型	void GPIO_ResetBits(GPIO_TypeDef* GPIOx, u16 GPIO_Pin)
功能描述	清除选定的数据端口位
输入参数 1	GPIOx：x 可为 A 到 E 来选择特定的 GPIO
输入参数 2	GPIO_Pin：指定被写入的端口位。 该参数可为 GPIO_Pin_x 的任意组合，其中 x 为 0 到 15。 参考 *GPIO_Pin* 部分以获得该参数可取值的更多信息
输出参数	无
返回参数	无
前提条件	无
调用函数	无

实例：

```
/* Clears the GPIOA port pin 10 and pin 15 */

GPIO_ResetBits(GPIOA, GPIO_Pin_10 | GPIO_Pin_15);
```

10.2.11 GPIO_WriteBit函数

表 194 GPIO_WriteBit 函数

函数名	GPIO_WriteBit
函数原型	void GPIO_WriteBit(GPIO_TypeDef* GPIOx, u16 GPIO_Pin, BitAction BitVal)
功能描述	置位或清除选定的数据端口位
输入参数 1	GPIOx：x 可为 A 到 E 来选择特定的 GPIO
输入参数 2	GPIO_Pin：被写入的端口位 参考：*GPIO_Pin* 部分以获得该参数可取值的更多信息
输入参数 3	BitVal：该参数指定被写入到选定位的值 BitVal 必须为 BitAction 枚举类型值 Bit_RESET：清除端口引脚 Bit_SET：置位端口引脚
输出参数	无
返回参数	无
前提条件	无
调用函数	无

实例：

/* Set the GPIOA port pin 15 */

GPIO_WriteBit(GPIOA, GPIO_Pin_15, Bit_SET);

10.2.12 GPIO_Write函数

表 195 GPIO_Write 函数

函数名	GPIO_Write
函数原型	void GPIO_Write(GPIO_TypeDef* GPIOx, u16 PortVal)
功能描述	写数据到指定的 GPIO 端口数据寄存器
输入参数 1	GPIOx：x 可为 A 到 E 来选择特定的 GPIO
输入参数 2	PortVal：写入到数据端口寄存器的值
输出参数	无
返回参数	无
前提条件	无
调用函数	无

实例：

/* Write data to GPIOA data port */

GPIO_Write(GPIOA, 0x1101);

10.2.13 GPIO_PinLockConfig函数

表 196 GPIO_PinLockConfig 函数

函数名	GPIO_PinLockConfig
函数原型	void GPIO_PinLockConfig(GPIO_TypeDef* GPIOx, u16 GPIO_Pin)
功能描述	锁定 GPIO 引脚配置寄存器
输入参数 1	GPIOx：x 可为 A 到 E 来选择特定的 GPIO
输入参数 2	GPIO_Pin：需要写入的端口位 参考：*GPIO_Pin* 部分以获得该参数可取值的更多信息
输出参数	无
返回参数	无
前提条件	无
调用函数	无

实例：

```
/* Lock GPIOA Pin0 and Pin1 */
GPIO_PinLockConfig(GPIOA, GPIO_Pin_0 | GPIO_Pin_1);
```

10.2.14 GPIO_EventOutputConfig函数

表 197 GPIO_EventOutputConfig 函数

函数名	GPIO_EventOutputConfig
函数原型	void GPIO_EventOutputConfig(u8 GPIO_PortSource, u8 GPIO_PinSource)

功能描述	选择 GPIO 引脚作为事件输出
输入参数 1	GPIO_PortSource：选择 GPIO 端口用作事件输出源 参考 *GPIO_PortSource* 部分以获得该参数可取值的更多信息
输入参数 2	GPIO_PinSource :事件输出引脚。该参数可为 GPIO_PinSourcex，这里 x 可为 0 到 15
输出参数	无
返回参数	无
前提条件	无
调用函数	无

GPIO_PortSource

这个参数可以选择 GPIO 端口源作为事件输出。表 198 给出它的可用值

表 198 GPIO_PortSource 值

GPIO_PortSource	描述
GPIO_PortSourceGPIOA	GPIOA 被选择
GPIO_PortSourceGPIOB	GPIOB 被选择
GPIO_PortSourceGPIOC	GPIOC 被选择
GPIO_PortSourceGPIOD	GPIOD 被选择
GPIO_PortSourceGPIOE	GPIOE 被选择

实例：

```
/* Selects the GPIOE pin 5 for EVENT output */

GPIO_EventOutputConfig(GPIO_PortSourceGPIOE, GPIO_PinSource5);
```

10.2.15 GPIO_EventOutputCmd函数

表 199 GPIO_EventOutputCmd 函数

函数名	GPIO_EventOutputCmd
函数原型	void GPIO_EventOutputCmd(FunctionalState NewState)
功能描述	使能（或禁能）事件输出
输入参数	NewState：事件输出的新状态 这个参数可为：ENABLE 或 DISABLE
输出参数	无
返回参数	无
前提条件	无
调用函数	无

实例：

```
/* Enable Event Ouput to the GPIOC pin 6 */
GPIO_EventOutputConfig(GPIO_PortSourceGPIOC, GPIO_PinSource6);
GPIO_EventOutputCmd(ENABLE);
```

10.2.16 GPIO_PinRemapConfig函数

表 200 GPIO_PinRemapConfig 函数

函数名	GPIO_PinRemapConfig
函数原型	void GPIO_PinRemapConfig(u32 GPIO_Remap, FunctionalState NewState)
功能描述	改变特定引脚的映射
输入参数 1	GPIO_Remap：选择引脚进行重映射 参考 *GPIO_Remap* 部分以获得该参数可取值的更多信息
输入参数 2	NewState：端口引脚重映射的新状态 该参数可为：ENABLE 或 DISABLE
输出参数	无
返回参数	无
前提条件	无
调用函数	无

10.2.17 GPIO_Remap

GPIO_Remap 用来改变备用功能映射。表 201 给出了它的可用值

表 201 GPIO_Remap 值

GPIO_Remap	描述
GPIO_Remap_SPI1	SPI1 备用功能映射
GPIO_Remap_I2C1	I2C1 备用功能映射
GPIO_Remap_USART1	USART1 备用功能映射
GPIO_Remap_USART2	USART2 备用功能映射

GPIO_PartialRemap_USART3	USART3 部分备用功能映射
GPIO_FullRemap_USART3	USART3 全部备用功能映射
GPIO_PartialRemap_TIM1	TIM1 部分备用功能映射
GPIO_FullRemap_TIM1	TIM1 全部备用功能映射
GPIO_PartialRemap1_TIM2	TIM2 部分 1 备用功能映射
GPIO_PartialRemap2_TIM2	TIM2 部分 2 备用功能映射
GPIO_FullRemap_TIM2	TIM2 全部备用功能映射
GPIO_PartialRemap_TIM3	TIM3 部分备用功能映射
GPIO_FullRemap_TIM3	TIM3 全部备用功能映射
GPIO_Remap_TIM4	TIM4 备用功能映射
GPIO_Remap1_CAN	CAN 备用功能映射
GPIO_Remap2_CAN	CAN 备用功能映射
GPIO_Remap_PD01	PD01 备用功能映射
GPIO_Remap_SWJ_NoJTRST	全部 SWJ 使能（JTAG-DP + SW-DP）但不包括 JTRST
GPIO_Remap_SWJ_JTAGDisable	JTAG-DP 禁能但 SW-DP 使能
GPIO_Remap_SWJ_Disable	全部 SWJ 禁能（JTAG-DP + SW-DP）

实例：

```
/* I2C1_SCL on PB.08, I2C1_SDA on PB.09 */

GPIO_PinRemapConfig(GPIO_Remap_I2C1, ENABLE);
```

10.2.18 GPIO_EXTILineConfig函数

表 202 GPIO_EXTILineConfig 函数

函数名	GPIO_EXTILineConfig
函数原型	void GPIO_EXTILineConfig(u8 GPIO_PortSource, u8 GPIO_PinSource)
功能描述	选择 GPIO 引脚作为 EXTI 线
输入参数 1	GPIO_PortSource：选择将要用作 EXTI 线的源的 GPIO 端口。 参考 GPIO_PortSource 部分以获得该参数可取值的更多信息
输入参数 2	GPIO_PinSource：需要配置的 EXTI 线 这个参数可以是 GPIO_PinSourcex，其中 x 为 0 到 15
输出参数	无
返回参数	无
前提条件	无
调用函数	无

实例：

```
/* Selects PB.08 as EXTI Line 8 */
GPIO_EXTILineConfig(GPIO_PortSource_GPIOB, GPIO_PinSource8);
```

11 内置集成电路（I2C）

I²C 总线接口是在微控制器和串行 I2C 总线间的接口。它提供了多主从的功能。它控制所有的 I²C 总线特殊的排序，协议，总裁以及定时。它也可以提供额外的功能，如 CRC 生成和检查，SM 总线和 PM 总线。

I²C 驱动可以被用来通过 I²C 接口传输和接收数据。执行状态依靠 I2C 驱动程序返回。

*11.1 小节：I²C 寄存器结构*描述了 I²C 固件库中使用的寄存器结构

*11.2 小节：固件库函数*说明了该部件的所有库函数

11.1 I2C寄存器结构

I²C 寄存器结构, I2C_TypeDef,是在 stm32f10x_map.h 中进行定义的：

```
typedef struct
{
  vu16 CR1;
u16 RESERVED0;
vu16 CR2;
u16 RESERVED1;
  vu16 OAR1;
  u16 RESERVED2;
  vu16 OAR2;
  u16 RESERVED3;
  vu16 DR;
```

```
    u16 RESERVED4;
    vu16 SR1;
    u16 RESERVED5;
    vu16 SR2;
    u16 RESERVED6;
    vu16 CCR;
    u16 RESERVED7;
  vu16 TRISE;
    u16 RESERVED8;
} I2C_TypeDef;
```

表 203 给出了所有 I2C 的寄存器

表 203　　I2C 寄存器

寄存器	描述
CR1	I²C 控制寄存器 1
CR2	I²C 控制寄存器 2
OAR1	I²C 自地址寄存器 1
OAR2	I²C 自地址寄存器 2（双重地址）
DR	I²C 数据寄存器
SR1	I²C 状态寄存器 1
SR2	I²C 状态寄存器 2

CCR	I²C 时钟控制寄存器
TRISE	I²C 上升时间寄存器

两个 I2C 部件都在 stm32f10x_map.h 中进行了声明：

```
...
#define PERIPH_BASE              ((u32)0x40000000)
#define APB1PERIPH_BASE          PERIPH_BASE
#define APB2PERIPH_BASE          (PERIPH_BASE + 0x10000)
#define AHBPERIPH_BASE           (PERIPH_BASE + 0x20000)
....
#define I2C1_BASE                (APB1PERIPH_BASE + 0x5400)
#define I2C2_BASE                (APB1PERIPH_BASE + 0x5800)
....
#ifndef DEBUG
...
#ifdef _I2C1
  #define I2C1                   ((I2C_TypeDef *) I2C1_BASE)
#endif /*_I2C1 */
#ifdef _I2C2
  #define I2C2                   ((I2C_TypeDef *) I2C2_BASE)
#endif /*_I2C2 */
...
```

```
#else     /* DEBUG */

...

#ifdef _I2C1

   EXT I2C_TypeDef                 *I2C1;

#endif /*_I2C1 */

#ifdef _I2C2

   EXT I2C_TypeDef                 *I2C2;

#endif /*_I2C2 */

...

#endif
```

当使用 Debug 模式时，_I2C1 和_I2C2 指针都在 stm32f10x_lib.c 中进行初始化。

```
...

#ifdef _I2C1

   I2C1 = (I2C_TypeDef *)    I2C1_BASE;

#endif /*_I2C1 */

#ifdef _I2C2

   I2C2 = (I2C_TypeDef *)    I2C2_BASE;

#endif /*_I2C2 */

...
```

为了访问 I²C 寄存器，_I2C, _I2C1 和_I2C2 必须在 stm32f10x_conf.h 中进行如下定义：

...

#define _I2C

#define _I2C1

#define _I2C2

...

11.2 固件库函数

表 204 给出了 I²C 固件库的所有库函数

表 204 I2C 固件库函数

函数名	描述
I2C_DeInit	将 I2Cx 外设寄存器复位到它们的默认值
I2C_Init	按照 I2C_InitStruct 的特定参数初始化 I2C 外设
I2C_StructInit	为 I2C_InitStruct 的成员赋默认值
I2C_Cmd	使能（或禁能）特定的 I²C 外设
I2C_DMACmd	使能（或禁能）特定的 I²C DMA 请求
I2C_DMALastTransferCmd	指出下一个 DMA 传输是最后一个
I2C_GenerateSTART	产生 I2Cx 通信的起始条件
I2C_GenerateSTOP	产生 I2Cx 通信的终止条件

I2C_AcknowledgeConfig	使能（或禁能）特定 I²C 的确认特性
I2C_OwnAddress2Config	配置特定的 I²C 自地址 2
I2C_DualAddressCmd	使能（或禁能）特定的 I²C 双重地址模式
I2C_GeneralCallCmd	使能（或禁能）特定的 I²C 通用调用特性
I2C_ITConfig	使能（或禁能）特定的 I²C 中断
I2C_SendData	通过 I2Cx 发送一个字节数据
I2C_ReceiveData	返回 I2Cx 最近接收的数据
I2C_Send7bitAddress	发送地址字节来选择从设备
I2C_ReadRegister	读取指定的 I²C 寄存器并返回其值
I2C_SoftwareResetCmd	使能（或禁能）特定的软件复位
I2C_SMBusAlertConfig	为特定的 I²C 驱动 SMBAlert 引脚为高或低
I2C_TransmitPEC	使能（或禁能）特定的 I²C PEC 传输
I2C_PECPositionConfig	选择特定的 PEC 位置
I2C_CalculatePEC	使能（或禁能）已传输字节的 PEC 计算值
I2C_GetPEC	为特定的 I²C 返回 PEC 值
I2C_ARPCmd	使能（或使不能）特定的 I²C ARP
I2C_StretchClockCmd	使能(或使不能)特定的 I2C 时钟伸展(stretching)
I2C_FastModeDutyCycleConfig	选择特定的 I2C 快速模式占空周期
I2C_GetLastEvent	返回最近的 I²C 事件
I2C_CheckEvent	检查最近的 I²C 事件是否等于那个已传输的参数

I2C_GetFlagStatus	检查特定的 I²C 标志是否被置位了
I2C_ClearFlag	清除 I²C 的挂起标志
I2C_GetITStatus	检查特定的 I²C 中断是否被发生了
I2C_ClearITPendingBit	清除 I²C 的中断挂起位

11.2.1 I2C_DeInit函数

表 205 I2C_DeInit 函数

函数名	I2C_DeInit
函数原型	void I2C_DeInit(I2C_TypeDef* I2Cx)
功能描述	将 I2Cx 外设寄存器复位到它们的默认值
输入参数	I2Cx：x 可为 1 或 2 来选择 I2C 外设
输出参数	无
返回参数	无
前提条件	无
调用函数	RCC_APB1PeriphClockCmd().

实例：

```
/* Deinitialize I2C2 interface*/

I2C_DeInit(I2C2);
```

11.2.2 I2C_Init函数

表 206 I2C_Init 函数

函数名	I2C_Init
函数原型	void I2C_Init(I2C_TypeDef* I2Cx, I2C_InitTypeDef* I2C_InitStruct)
功能描述	按照 I2C_InitStruct 的特定参数初始化 I2C 部件
输入参数 1	I2Cx：x 可为 1 或 2 来选择 I^2C 部件
输入参数 2	I2C_InitStruct 指向 I2C_InitTypeDef 结构 ,该结构包含了特定 I^2C 部件的配置信息。 参考 *I2C_InitTypeDef 结构*部分以获得关于该参数允许值的更多信息
输出参数	无
返回参数	无
前提条件	无
调用函数	无

I2C_InitTypeDef 结构

I2C_InitTypeDef 定义在 stm32f10x_i2c.h 文件中：

```
typedef struct
{
u16 I2C_Mode;
u16 I2C_DutyCycle;
  u16 I2C_OwnAddress1;
u16 I2C_Ack;
```

u16 I2C_AcknowledgedAddress;

u32 I2C_ClockSpeed;

} I2C_InitTypeDef;

I2C_Mode

I2C_Mode 是用来配置 I2C 的模式的。表 211 给出了它的可用值

表 207 I2C_Mode 定义

I2C_Mode	描述
I2C_Mode_I2C	I²C 被配置为 I2C 模式
I2C_Mode_SMBusDevice	I²C 被配置为 SMBus 设备模式
I2C_Mode_SMBusHost	I²C 被配置为 SMBus 主机模式

I2C_DutyCycle

I2C_DutyCycle 是用来选择 I²C 的快模式占空周期。表 208 给出了它的可用值

表 208 I2C_DutyCycle 定义

I2C_DutyCycle	描述
I2C_DutyCycle_16_9	I²C 的快模式 Tlow/Thigh=16/9
I2C_DutyCycle_2	I²C 的快模式 Tlow/Thigh=2

注意： 该成员只有当 I²C 工作在快模式时才有意义（工作时钟速度超过 100kHz）

I2C_OwnAddress1

该成员被用来配置自地址的第一个设备。它可以是 7 位或 10 位地址。

I2C_Ack

I2C_Ack 使能（或使不能）确认特性。表 209 给出了它的可用值

表 209 I2C_Ack 定义

I2C_Ack	描述
I2C_Ack_Enable	使能确认特性
I2C_Ack_Disable	禁能确认特性

I2C_AcknowledgedAddress

I2C_AcknowledgedAddress 定义了 7 位或 10 位地址的确认，表 210 给出了它的可用值

表 210 I2C_AcknowledgedAddress 定义

I2C_AcknowledgedAddress	描述
I2C_AcknowledgedAddress_7bit	确认 7 位地址
I2C_AcknowledgedAddress_10bit	确认 10 位地址

I2C_ClockSpeed

这个成员是用来配置时钟频率的。它的值必须在 400kHz 以下

实例：

```
/* Initialize the I2C1 according to the I2C_InitStructure members */

I2C_InitTypeDef   I2C_InitStructure;

I2C_InitStructure.I2C_Mode = I2C_Mode_SMBusHost;

I2C_InitStructure.I2C_DutyCycle = I2C_DutyCycle_2;

I2C_InitStructure.I2C_OwnAddress1 = 0x03A2;

I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;

I2C_InitStructure.I2C_AcknowledgedAddress =

I2C_AcknowledgedAddress_7bit;

I2C_InitStructure.I2C_ClockSpeed = 200000;

I2C_Init(I2C1, &I2C_InitStructure);
```

11.2.3 I2C_StructInit函数

表 211　　I2C_StructInit 函数

函数名	I2C_StructInit
函数原型	void I2C_StructInit(I2C_InitTypeDef* I2C_InitStruct)
功能描述	为 I2C_InitStruct 的成员赋默认值
输入参数	I2C_InitStruct：指向必须初始化的 I2C_InitTypeDef 结构
输出参数	无
返回参数	无
前提条件	无

调用函数	无

I2C_InitStruct 有如下缺省值：

表 212　　I2C_InitStruct 缺省值

成员	缺省值
I2C_Mode	I2C_Mode_I2C
I2C_DutyCycle	I2C_DutyCycle_2
I2C_OwnAddress1	0
I2C_Ack	I2C_Ack_Disable
I2C_AcknowledgedAddress	I2C_AcknowledgedAddress_7bit
I2C_ClockSpeed	5000

实例：

```
/* Initialize an I2C_InitTypeDef structure */

I2C_InitTypeDef  I2C_InitStructure;

I2C_StructInit(&I2C_InitStructure);
```

11.2.4 I2C_Cmd函数

函数名	I2C_Cmd
函数原型	void I2C_Cmd(I2C_TypeDef* I2Cx, FunctionalState NewState)
功能描述	使能（或禁能）特定的 I^2C 外设

输入参数 1	I2Cx：x 可为 1 或 2 来选择 I^2C 外设
输入参数 2	NewState：I^2C 部件的新状态 该参数可为：ENABLE 或 DISABLE
输出参数	无
返回参数	无
前提条件	无
调用函数	无

实例：

```
/* Enable I2C1 peripheral */

I2C_Cmd(I2C1, ENABLE);
```

11.2.5 I2C_DMACmd函数

表 214 I2C_DMACmd 函数

函数名	I2C_DMACmd
函数原型	I2C_DMACmd(I2C_TypeDef* I2Cx, FunctionalState NewState)
功能描述	使能（或禁能）特定的 I^2C 的 DMA 请求
输入参数 1	I2Cx：x 可为 1 或 2 来选择 I^2C 部件
输入参数 2	NewState：I^2C 的 DMA 传输的新状态 该参数可为：ENABLE 或 DISABLE

输出参数	无
返回参数	无
前提条件	无
调用函数	无

实例：

/* Enable I2C2 DMA transfer */

I2C_DMACmd(I2C2, ENABLE);

11.2.6 I2C_DMALastTransferCmd函数

表 215 I2C_DMALastTransferCmd 函数

函数名	I2C_DMALastTransferCmd
函数原型	I2C_DMALastTransferCmd(I2C_TypeDef* I2Cx, FunctionalState NewState)
功能描述	指出下一个 DMA 传输是最后一个
输入参数 1	I2Cx：x 可为 1 或 2 来选择 I2C 部件
输入参数 2	NewState：I2C 的最后 DMA 传输的新状态 该参数为：ENABLE 或 DISABLE
输出参数	无
返回参数	无
前提条件	无

调用函数	无

实例：

/* Specify that the next I2C2 DMA transfer is the last one */

I2C_DMALastTransferCmd(I2C2, ENABLE);

11.2.7 I2C_GenerateSTART函数

函数名	I2C_GenerateSTART
函数原型	void I2C_GenerateSTART(I2C_TypeDef* I2Cx, FunctionalState NewState)
功能描述	产生 I2C 通信的起始条件
输入参数 1	I2Cx：x 可为 1 或 2 来选择 I^2C 外设
输入参数 2	NewState：I^2C 起始条件生成的新状态 该参数为：ENABLE 或 DISABLE
输出参数	无
返回参数	无
前提条件	无
调用函数	无

实例：

/* Generate a START condition on I2C1 */

I2C_GenerateSTART(I2C1, ENABLE);

11.2.8 I2C_GenerateSTOP函数

表 217 I2C_GenerateSTOP 函数

函数名	I2C_GenerateSTOP
函数原型	void I2C_GenerateSTOP(I2C_TypeDef* I2Cx, FunctionalState NewState)
功能描述	产生 I2C 通信的停止条件
输入参数 1	I2Cx：x 可为 1 或 2 来选择 I2C 部件
输入参数 2	NewState：I2C 停止条件生成的新状态 该参数为：ENABLE 或 DISABLE
输出参数	无
返回参数	无
前提条件	无
调用函数	无

实例：

```
/* Generate a STOP condition on I2C2 */
I2C_GenerateSTOP(I2C2, ENABLE);
```

11.2.9 I2C_AcknowledgeConfig函数

表 218 I2C_AcknowledgeConfig 函数

函数名	I2C_AcknowledgeConfig
函数原型	void I2C_AcknowledgeConfig(I2C_TypeDef* I2Cx, FunctionalState NewState)
功能描述	使能（或禁能）特定的 I^2C 的确认特性
输入参数 1	I^2Cx：x 可为 1 或 2 来选择 I2C 外设
输入参数 2	NewState：I^2C 确认特性的新状态 该参数为：ENABLE 或 DISABLE
输出参数	无
返回参数	无
前提条件	无
调用函数	无

实例：

```
/* Enable the I2C1 Acknowledgement */
I2C_AcknowledgeConfig(I2C1, ENABLE);
```

11.2.10 I2C_OwnAddress2Config函数

表 219 I2C_OwnAddress2Config 函数

函数名	I2C_OwnAddress2Config
函数原型	void I2C_OwnAddress2Config(I2C_TypeDef* I2Cx, u8

	Address)
功能描述	配置特定的 I2C 自地址 2
输入参数 1	I2Cx：x 可为 1 或 2 来选择 I2C 部件
输入参数 2	地址：确认 7 位 I2C 自地址 2
输出参数	无
返回参数	无
前提条件	无
调用函数	无

实例：

```
/* Set the I2C1 own address2 to 0x38 */

I2C_OwnAddress2Config(I2C1, 0x38);
```

11.2.11 I2C_DualAddressCmd函数

表 220 I2C_DualAddressCmd 函数

函数名	I2C_DualAddressCmd
函数原型	void I2C_DualAddressCmd(I2C_TypeDef* I2Cx, FunctionalState NewState)
功能描述	使能（或禁能）特定的 I2C 双重地址模式
输入参数 1	I2Cx：x 可为 1 或 2 来选择 I2C 部件
输入参数 2	NewState：I2C 双重地址模式的新状态

	该参数可为：ENABLE 或 DISABLE
输出参数	无
返回参数	无
前提条件	无
调用函数	无

实例：

```
/* Enable the I2C2 dual addressing mode*/

I2C_DualAdressCmd(I2C2, ENABLE);
```

11.2.12 I2C_GeneralCallCmd函数

表 221 I2C_GeneralCallCmd 函数

函数名	I2C_GeneralCallCmd
函数原型	void I2C_GeneralCallCmd(I2C_TypeDef* I2Cx, FunctionalState NewState)
功能描述	使能（或禁能）特定 I^2C 的通用调用特性
输入参数 1	I2Cx：x 可为 1 或 2 来选择 I2C 部件
输入参数 2	NewState：I^2C 通用调用的新状态 该参数可为：ENABLE 或 DISABLE
输出参数	无
返回参数	无

前提条件	无
调用函数	无

实例：

```
/* Enable the I2C1 general call feature */

I2C_GeneralCallCmd(I2C1, ENABLE);
```

11.2.13 I2C_ITConfig函数

表 222 I2C_ITConfig 函数

函数名	I2C_ITConfig
函数原型	void I2C_ITConfig(I2C_TypeDef* I2Cx, u16 I2C_IT, FunctionalState NewState)
功能描述	使能（或禁能）特定的 I^2C 中断
输入参数 1	I2Cx：x 可为 1 或 2 来选择 I2C 部件
输入参数 2	I2C_IT：需要使能（或禁能）的 I^2C 中断源。参考 *I2C_IT* 部分以获得该参数可取值的更多信息
输入参数 3	NewState：指定的 I^2C 中断的新状态 该参数可为：ENABLE 或 DISABLE
输出参数	无
返回参数	无
前提条件	无

调用函数	无

I2C_IT

这个参数可以使能（或禁能）I2C 中断。以下值可单独或组合使用：

表 223 I2C_IT 定义

I2C_IT	描述
I2C_IT_BUF	缓冲区中断屏蔽
I2C_IT_EVT	事件中断屏蔽
I2C_IT_ERR	错误中断屏蔽

实例：

```
/* Enable I2C2 event and buffer interrupts */

I2C_ITConfig(I2C2, I2C_IT_BUF | I2C_IT_EVT, ENABLE);
```

11.2.14 I2C_SendData函数

表 224 I2C_SendData 函数

函数名	I2C_SendData
函数原型	void I2C_SendData(I2C_TypeDef* I2Cx, u8 Data)
功能描述	通过 I2C 发送一字节数据
输入参数 1	I2Cx：x 可为 1 或 2 来选择 I2C 外设
输入参数 2	Data：需要传送的字节
输出参数	无

返回参数	无
前提条件	无
调用函数	无

实例：

```
/* Transmit 0x5D byte on I2C2 */

I2C_SendData(I2C2, 0x5D);
```

11.2.15 I2C_ReceiveData函数

表 225 I2C_ReceiveData 函数

函数名	I2C_ReceiveData
函数原型	u8 I2C_ReceiveData(I2C_TypeDef* I2Cx)
功能描述	返回 I2Cx 外设最近接收的数据
输入参数	I2Cx：x 可为 1 或 2 来选择 I^2C 外设
输出参数	无
返回参数	接收字节
前提条件	无
调用函数	无

实例：

```
/* Read the received byte on I2C1 */

u8 ReceivedData;
```

ReceivedData = I2C_ReceiveData(I2C1);

11.2.16 I2C_Send7bitAddress函数

表 226 I2C_Send7bitAddress 函数

函数名	I2C_Send7bitAddress
函数原型	void I2C_Send7bitAddress(I2C_TypeDef* I2Cx, u8 Address, u8 I2C_Direction)
功能描述	传输地址字节来选择从设备
输入参数 1	I2Cx：x 可为 1 或 2 来选择 I^2C 外设
输入参数 2	Address：将要被发送的从设备地址
输入参数 3	I2C_Direction：确定 I^2C 设备为主设备或从设备。参考：*I2C_Direction* 部分以获得该参数可取值的更多信息
输出参数	无
返回参数	无
前提条件	无
调用函数	无

I2C_Direction

这个参数配置 I2C 接口为接收器或发送器模式（见表 227）

表 227 I2C_Direction

I2C_Direction	描述

I2C_Direction_Transmitter	选择发送方向
I2C_Direction_Receiver	选择接收方向

实例：

```
/* Send, as transmitter, the Slave device address 0xA8 in 7-bit
addressing mode in I2C1 */
I2C_Send7bitAddress(I2C1, 0xA8, I2C_Direction_Transmitter);
```

11.2.17 I2C_ReadRegister函数

表 228 I2C_ReadRegister 函数

函数名	I2C_ReadRegister
函数原型	u16 I2C_ReadRegister(I2C_TypeDef* I2Cx, u8 I2C_Register)
功能描述	读取特定的 I^2C 寄存器并返回其值
输入参数 1	I2Cx：x 可为 1 或 2 来选择 I^2C 部件
输入参数 2	I2C_Register：将要读取的寄存器 参考 *I2C_Register* 部分以获得该参数可取值的更多信息
输出参数	无
返回参数	已读寄存器的值（1）
前提条件	无

调用函数	无

当该寄存器被读取时一些标志会被清除

I2C_Register

通过调用 I2C_ReadRegister 函数可以读取 I2C 的寄存器。如表 229

表 229　　可读 I2C 寄存器

I2C_Register	描述
I2C_Register_CR1	I2C_CR1 寄存器被选定读取
I2C_Register_CR2	I2C_CR2 寄存器被选定读取
I2C_Register_OAR1	I2C_OAR1 寄存器被选定读取
I2C_Register_OAR2	I2C_OAR2 寄存器被选定读取
I2C_Register_DR	I2C_DR 寄存器被选定读取
I2C_Register_SR1	I2C_SR1 寄存器被选定读取
I2C_Register_SR2	I2C_SR2 寄存器被选定读取
I2C_Register_CCR	I2C_CCR 寄存器被选定读取
I2C_Register_TRISE	I2C_TRISE 寄存器被选定读取

实例：

```
/* Return the I2C_CR1 register value of I2C2 peripheral */

u16 RegisterValue;

RegisterValue = I2C_ReadRegister(I2C2, I2C_Register_CR1);
```

11.2.18 I2C_SoftwareResetCmd函数

表 230 I2C_SoftwareResetCmd 函数

函数名	I2C_SoftwareResetCmd
函数原型	I2C_SoftwareResetCmd(I2C_TypeDef* I2Cx, FunctionalState NewState)
功能描述	使能（或使不能）特定 I2C 软件复位
输入参数 1	I2Cx：x 可为 1 或 2 来选择 I2C 外设
输入参数 2	NewState：I2C 软件复位的新状态 该参数可为：ENABLE 或 DISABLE
输出参数	无
返回参数	无
前提条件	无
调用函数	无

实例：

```
/* Put under reset the I2C1 peripheral */
I2C_SoftwareResetCmd(I2C1, ENABLE);
```

11.2.19 I2C_SMBusAlertConfig函数

表 231 I2C_SMBusAlertConfig 函数

函数名	I2C_SMBusAlertConfig

函数原型	void I2C_SMBusAlertConfig(I2C_TypeDef* I2Cx, u16 I2C_SMBusAlert)
功能描述	为特定的 I2C 驱动 SMBAlert 引脚为高或低
输入参数 1	I2Cx：x 可为 1 或 2 来选择 I2C 部件
输入参数 2	I2C_SMBusAlert：SMBusAlert 引脚电平。 参考：*I2C_SMBusAlert* 部分以获得该参数可取值的更多信息
输出参数	无
返回参数	无
前提条件	无
调用函数	无

I2C_SMBusAlert

这个参数选择 SMBusAlert 引脚的有效（见表 232）

表 232 I2C_SMBusAlert 值

I2C_SMBusAlert	描述
I2C_SMBusAlert_Low	SMBusAlert 引脚驱动为低
I2C_SMBusAlert_High	SMBusAlert 引脚驱动为高

实例：

```
/* Let the I2C2 SMBusAlert pin High */

I2C_SMBusAlertConfig(I2C2, I2C_SMBusAlert_High);
```

11.2.20 I2C_TransmitPE函数

表 233 I2C_TransmitPE 函数

函数名	I2C_TransmitPE
函数原型	I2C_TransmitPEC(I2C_TypeDef* I2Cx, FunctionalState NewState)
功能描述	使能（或禁能）特定 I^2C 的 PEC 传输
输入参数 1	I2Cx：x 可为 1 或 2 来选择 I^2C 部件
输入参数 2	NewState：I^2C 的 PEC 传输的新状态 该参数可为：ENABLE 或 DISABLE
输出参数	无
返回参数	无
前提条件	无
调用函数	无

实例：

```
/* Enable the I2C1 PEC transfer */
I2C_TransmitPEC(I2C1, ENABLE);
```

11.2.21 I2C_PECPositionConfig函数

表 234 I2C_PECPositionConfig 函数

函数名	I2C_PECPositionConfig

函数原型	void I2C_PECPositionConfig(I2C_TypeDef* I2Cx, u16 I2C_PECPosition)
功能描述	选择特定的 I^2C 的 PEC 位置
输入参数 1	I2Cx：x 可为 1 或 2 来选择 I^2C 外设
输入参数 2	I2C_PECPosition：PEC 位置 参考：*I2C_PECPosition* 部分以获得该参数可取值的更多信息
输出参数	无
返回参数	无
前提条件	无
调用函数	无

I2C_PECPosition

该参数选择了 PEC 的位置（见表 235）

表 235 I2C_PECPosition 值

I2C_PECPosition	描述
I2C_PECPosition_Next	下一字节就是 PEC
I2C_PECPosition_Current	当前字节就是 PEC

实例：

```
/* Configure the PEC bit to indicvates that the next byte in shift

register is PEC for I2C2 */

I2C_PECPositionConfig(I2C2, I2C_PECPosition_Next);
```

11.2.22 I2C_CalculatePEC函数

表 236 I2C_CalculatePEC 函数

函数名	I2C_CalculatePEC
函数原型	void I2C_CalculatePEC(I2C_TypeDef* I2Cx, FunctionalState NewState)
功能描述	使能（或禁能）已传输字节的 PEC 计算值
输入参数 1	I2Cx：x 可为 1 或 2 来选择 I2C 部件
输入参数 2	NewState：PEC 计算值的新状态 该参数可为：ENABLE 或 DISABLE
输出参数	无
返回参数	无
前提条件	无
调用函数	无

实例：

```
/* Enable the PEC calculation for the transfered bytes from I2C2 */
I2C_CalculatePEC(I2C2, ENABLE);
```

11.2.23 I2C_GetPEC函数

表 237 I2C_GetPEC 函数

函数名	I2C_GetPEC

函数原型	u8 I2C_GetPEC(I2C_TypeDef* I2Cx)
功能描述	为特定 I^2C 接口返回 PEC 值
输入参数	I2Cx：x 可为 1 或 2 来选择 I^2C 外设
输出参数	无
返回参数	PEC 值
前提条件	无
调用函数	无

实例：

```
/* Returns the I2C2 PEC value */
u8 PECValue;
PECValue = I2C_GetPEC(I2C2);
```

11.2.24 I2C_ARPCmd函数

表 238 I2C_ARPCmd 函数

函数名	I2C_ARPCmd
函数原型	void I2C_ARPCmd(I2C_TypeDef* I2Cx, FunctionalState NewState)
功能描述	使能（或禁能）特定的 I^2C 的 ARP
输入参数 1	I2Cx：x 可为 1 或 2 来选择 I^2C 外设
输入参数 2	NewState：I2Cx 的 ARP 的新状态

	该参数可为：ENABLE 或 DISABLE
输出参数	无
返回参数	无
前提条件	无
调用函数	无

实例：

```
* Enable the I2C1 ARP feature */

I2C_ARPCmd(I2C1, ENABLE);
```

11.2.25 I2C_StretchClockCmd函数

表 239 I2C_StretchClockCmd 函数

函数名	I2C_StretchClockCmd
函数原型	void I2C_StretchClockCmd(I2C_TypeDef* I2Cx, FunctionalState NewState)
功能描述	使能（或禁能）特定的 I^2C 时钟伸展（stretching）
输入参数 1	I2Cx：x 可为 1 或 2 来选择 I^2C 外设
输入参数 2	NewState：时钟伸展的新状态 该参数可为：ENABLE 或 DISABLE
输出参数	无
返回参数	无

前提条件	无
调用函数	无

实例：

```
/* Enable the I2C2 clock stretching */

I2C_StretchClockCmd(I2C2, ENABLE);
```

11.2.26 I2C_FastModeDutyCycleConfig函数

表 240 I2C_FastModeDutyCycleConfig 函数

函数名	I2C_FastModeDutyCycleConfig
函数原型	void I2C_FastModeDutyCycleConfig(I2C_TypeDef* I2Cx, u16 I2C_DutyCycle)
功能描述	选择特定 I^2C 的快速模式占空周期
输入参数 1	I2Cx：x 可为 1 或 2 来选择 I^2C 外设
输入参数 2	I2C_DutyCycle：快速模式下占空周期 参考：*I2C_DutyCycl* 部分以获得该参数可取值的更多信息
输出参数	无
返回参数	无
前提条件	无
调用函数	无

I2C_DutyCycle

该参数配置了 I2C 快速模式下的职责圈（duty Cycle）(见表 241）

表 241 I2C_DutyCycle

I2C_DutyCycle	模式
I2C_DutyCycle_2	I2C 快模式 Tlow/Thigh=2
I2C_DutyCycle_16_9	I2C 快模式 Tlow/Thigh=16/9

实例：

```
/* Set the fast mode duty cyle to 16/9 for I2C2 */

I2C_FastModeDutyCycleConfig(I2C2, I2C_DutyCycle_16_9);
```

11.2.27 I2C_GetLastEvent函数

表 242 I2C_GetLastEvent 函数

函数名	I2C_GetLastEvent
函数原型	u32 I2C_GetLastEvent(I2C_TypeDef* I2Cx)
功能描述	返回最近 I2Cx 事件
输入参数	I2Cx：x 可为 1 或 2 来选择 I2C 外设
输出参数	无
返回参数	最近的 I2Cx 事件
前提条件	无
调用函数	无

实例：

```
/* Get last I2C1 event */

u32 Event;

Event = I2C_GetLastEvent(I2C1);
```

11.2.28 I2C_CheckEvent函数

表 243 I2C_CheckEvent 函数

函数名	I2C_CheckEvent
函数原型	ErrorStatus I2C_CheckEvent(I2C_TypeDef* I2Cx, u32 I2C_EVENT)
功能描述	检查最近的 I2C 事件是否等于那个已传输的参数
输入参数 1	I2Cx：x 可为 1 或 2 来选择 I2C 部件
输入参数 2	I2C_EVENT：指定被检查的事件 参考：*I2C_EVENT* 部分以获得该参数可取值的更多信息
输出参数	无
返回参数	一个 ErrorStatus 枚举值： SUCCESS：最近的事件与 I2C_EVENT 相同 ERROR：最近的事件与 I2C_EVENT 不同
前提条件	无
调用函数	无

I2C_EVENT

可以被 I2C_CheckEvent 函数检查的事件如表 244：

表 244　　I2C_EVENT

I2C_CheckEvent	描述
I2C_EVENT_SLAVE_RECEIVER_ADDRESS_MATCHED	EV1
I2C_EVENT_SLAVE_TRANSMITTER_ADDRESS_MATCHED	EV1
I2C_EVENT_SLAVE_RECEIVER_SECONDADDRESS_MATCHED	EV1
I2C_EVENT_SLAVE_TRANSMITTER_SECONDADDRESS_MATCHED	EV1
I2C_EVENT_SLAVE_GENERALCALLADDRESS_MATCHED	EV1
I2C_EVENT_SLAVE_BYTE_RECEIVED	EV2
I2C_EVENT_SLAVE_BYTE_TRANSMITTED	EV3
I2C_EVENT_SLAVE_ACK_FAILURE	EV3-1
I2C_EVENT_SLAVE_STOP_DETECTED	EV4
I2C_EVENT_MASTER_MODE_SELECT	EV5
I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED	EV6
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED	EV6
I2C_EVENT_MASTER_BYTE_RECEIVED	EV7
I2C_EVENT_MASTER_BYTE_TRANSMITTED	EV8
I2C_EVENT_MASTER_MODE_ADDRESS10	EV9

实例：

/* Check if the event happen on I2C1 is equal to

I2C_EVENT_MASTER_BYTE_RECEIVED */

ErrorStatus Status;

Status = I2C_CheckEvent(I2C1, I2C_EVENT_MSTER_BYTE_RECEIVED);

11.2.29 I2C_GetFlagStatus函数

表 245 I2C_GetFlagStatus 函数

函数名	I2C_GetFlagStatus
函数原型	FlagStatus I2C_GetFlagStatus(I2C_TypeDef* I2Cx, u32 I2C_FLAG)
功能描述	检查特定的 I2C 标志是否被置位了
输入参数 1	I2Cx：x 可为 1 或 2 来选择 I^2C 外设
输入参数 2	I2C_FLAG：确定将要被检查的标志 参考 *I2C_FLAG* 部分以获得该参数可取值的更多信息
输出参数	无
返回参数	I2C_FLAG 的新状态（SET 或 RESET）(1)
前提条件	无
调用函数	无

当该寄存器被读取时一些标志会被清除

I2C_FLAG

表 246 列出了 I2C_GetFlagStatus 函数可以检查的标志

表 246 I2C_FLAG 定义

I2C_FLAG	描述
I2C_FLAG_DUALF	双重标志（从模式）
I2C_FLAG_SMBHOST	SMBus 主机头（从模式）
I2C_FLAG_SMBDEFAULT	SMBus 默认头（从模式）
I2C_FLAG_GENCALL	通用调用头标志（从模式）
I2C_FLAG_TRA	发送器/接收器标志
I2C_FLAG_BUSY	总线忙的标志
I2C_FLAG_MSL	主机/从机标志
I2C_FLAG_SMBALERT	SMBus 警告标志
I2C_FLAG_TIMEOUT	超时或 Tlow 错误标志
I2C_FLAG_PECERR	接收 PEC 错误标志
I2C_FLAG_OVR	上溢/下溢 标志（从模式）
I2C_FLAG_AF	确认失败标志
I2C_FLAG_ARLO	失去总裁标志（主模式）
I2C_FLAG_BERR	总线错误标志
I2C_FLAG_TXE	数据寄存器空的标志（发送器）
I2C_FLAG_RXNE	数据寄存器非空的标志（接收器）
I2C_FLAG_STOPF	停止检测寄存器（从模式）
I2C_FLAG_ADD10	10 位头被发送标志（主模式）
I2C_FLAG_BTF	字节发送结束标志

I2C_FLAG_ADDR	地址发送标志（主模式）“ADSL” 地址匹配标志（从模式）“ENDAD”
I2C_FLAG_SB	起始位标志（主模式）

注意： I2C_GetFlagStatus 函数只用了 bits[27:0]来返回被选择的标志的状态。这个值与计算了的寄存器（包含了两个 I2C 状态寄存器，I2C_SR1 和 I2C_SR2）中的标志位置相关.

实例：

```
/* Return the I2C_FLAG_AF flag state of I2C2 peripheral */
Flagstatus Status;
Status = I2C_GetFlagStatus(I2C2, I2C_FLAG_AF);
```

11.2.30 I2C_ClearFlag函数

表 247 I2C_ClearFlag 函数

函数名	I2C_ClearFlag
函数原型	void I2C_ClearFlag(I2C_TypeDef* I2Cx, u32 I2C_FLAG)
功能描述	清除 I2Cx 的挂起标志
输入参数 1	I2Cx：x 可为 1 或 2 来选择 I^2C 外设
输入参数 2	I2C_FLAG：需清除的标志。参考 *I2C_FLAG* 部分以获得该参数可取值的更多信息 注 意 ： DUALF,SMBHOST,SMBDEFAULT,

	GENCALL,TRA,BUSY,MSL,TXE 及 RXNE 等标志不会因为调用这个函数而被清除
输出参数	无
返回参数	无
前提条件	无
调用函数	无

I2C_FLAG

表 248 列出了由 I2C_ClearFlag 函数可以清除的标志

表 248 I2C_FLAG 定义

I2C_FLAG	描述
I2C_FLAG_SMBALERT	SMBus 警告标志
I2C_FLAG_TIMEOUT	超时或 Tlow 错误标志
I2C_FLAG_PECERR	接收的 PEC 错误标志
I2C_FLAG_OVR	上溢/下溢 标志（从模式）
I2C_FLAG_AF	确认失败标志
I2C_FLAG_ARLO	失去总裁标志（主模式）
I2C_FLAG_BERR	总线错误标志
I2C_FLAG_STOPF	停止检测标志（从模式）
I2C_FLAG_ADD10	10 位头发送标志（主模式）
I2C_FLAG_BTF	字节传送结束标志
I2C_FLAG_ADDR	地址发送标志（主模式）“ADSL”

	地址符合标志（从模式）“ENDAD”
I2C_FLAG_SB	起始位标志（主模式）

实例：

/* Clear the Stop detection flag on I2C2 */

I2C_ClearFlag(I2C2, I2C_FLAG_STOPF);

11.2.31 I2C_GetITStatus函数

表 249 I2C_GetITStatus 函数

函数名	I2C_GetITStatus
函数原型	ITStatus I2C_GetITStatus(I2C_TypeDef* I2Cx, u32 I2C_IT)
功能描述	检查特定的 I^2C 中断是否产生
输入参数 1	I2Cx：x 可为 1 或 2 来选择 I2C 部件
输入参数 2	I2C_IT：确定需要检查的中断源 参考 *I2C_IT* 部分以获得该参数可取值的更多信息
输出参数	无
返回参数	I2C_IT 的新状态（SET 或 RESET）(1)
前提条件	无
调用函数	无

（1）当该寄存器被读取时一些标志会被清除

I2C_IT

该参数可以用来选择 I2C 中断标志，这些标志靠 I2C_GetITStatus 函数来检查（见表 250）

表 250 I2C_IT 定义

I2C_IT	描述
I2C_IT_SMBALERT	SMBus 警告标志
I2C_IT_TIMEOUT	超时或 Tlow 错误标志
I2C_IT_PECERR	接收的 PEC 错误标志
I2C_IT_OVR	上溢/下溢 标志（从模式）
I2C_IT_AF	确认失败标志
I2C_IT_ARLO	失去总裁标志（主模式）
I2C_IT_BERR	总线错误标志
I2C_IT_TXE	数据寄存器空的标志（发送器）
I2C_IT_RXNE	数据寄存器非空的标志（接收器）
I2C_IT_STOPF	停止检测寄存器（从模式）
I2C_IT_ADD10	10 位头发送标志（主模式）
I2C_IT_BTF	字节传送结束标志
I2C_IT_ADDR	地址发送标志（主模式）“ADSL” 地址符合标志（从模式）“ENDAD”
I2C_IT_SB	起始位标志（主模式）

实例：

```
/* Return the I2C_IT_OVR flag state of I2C1 peripheral */

ITstatus Status;
```

```
Status = I2C_GetITStatus(I2C1, I2C_IT_OVR);
```

11.2.32 I2C_ClearITPendingBit函数

表 251 I2C_ClearITPendingBit 函数

函数名	I2C_ClearITPendingBit
函数原型	void I2C_ClearITPendingBit(I2C_TypeDef* I2Cx, u32 I2C_IT)
功能描述	清除 I2Cx 的中断挂起位
输入参数 1	I2Cx：x 可为 1 或 2 来选择 I^2C 外设
输入参数 2	I2C_IT：确定需要清除的中断挂起位。参考 *I2C_IT* 部分以获得该参数可取值的更多信息 注 意 ： DUALF,SMBHOST,SMBDEFAULT, GENCALL,TRA,BUSY,MSL,TXE 及 RXNE 等标志不会因为调用该函数就被清除
输出参数	无
返回参数	无
前提条件	无
调用函数	无

I2C_IT

I2C_IT 参数是用来选择需要靠 I2C_ClearITPendingBit 函数来清除的 I^2C 中断挂起位的(见表 252)

表 252 I2C_IT 定义

I2C_IT	描述
I2C_IT_SMBALERT	SMBus 警告标志
I2C_IT_TIMEOUT	超时或 Tlow 错误标志
I2C_IT_PECERR	接收的 PEC 错误标志
I2C_IT_OVR	上溢/下溢 标志（从模式）
I2C_IT_AF	确认失败标志
I2C_IT_ARLO	失去总裁标志（主模式）
I2C_IT_BERR	总线错误标志
I2C_IT_STOPF	停止检测寄存器（从模式）
I2C_IT_ADD10	10 位头发送标志（主模式）
I2C_IT_BTF	字节传送结束标志
I2C_IT_ADDR	地址发送标志（主模式）“ADSL” 地址符合标志（从模式）“ENDAD”
I2C_IT_SB	起始位标志（主模式）

实例：

```
/* Clear the Timeout interrupt opending bit on I2C2 */
I2C_ClearITPendingBit(I2C2, I2C_IT_TIMEOUT);
```

12 独立看门狗 （IWDG）

独立看门狗（IWDG）用来解决由软件或者硬件的错误而引起的处理器故障。能在停止或待机（standby）模式下执行。

12.1 节 :IWDG 寄存器结构 描述了在 IWDG 固件库使用的数据结构。***12.2 节 :固件库函数*** 介绍了固件库函数。

12.1 IWDG 寄存器结构

IWDG 寄存器结构，在 stm32f10x_map.h 文件中定义了 IWDG_TypeDef ,。

如下：

```
typedef struct
{
vu32 KR;
vu32 PR;
vu32 RLR;
vu32 SR;
} IWDG_TypeDef;
```

表 253 列出了 IWDG 寄存器表：

表 253 IWDG 寄存器

寄存器	描述
KR	IWDG 关键寄存器
PR	IWDG 预分频寄存器
RLR	IWDG 重载寄存器
SR	IWDG 状态寄存器

IWDG 外围模块已在 stm32f10x_map.h 声明：

```
#define PERIPH_BASE ((u32)0x40000000)

#define APB1PERIPH_BASE PERIPH_BASE

#define APB2PERIPH_BASE (PERIPH_BASE + 0x10000)

#define AHBPERIPH_BASE (PERIPH_BASE + 0x20000)

#define IWDG_BASE (APB1PERIPH_BASE + 0x3000)

#ifndef DEBUG

...

#ifdef _IWDG

#define IWDG ((IWDG_TypeDef *) IWDG_BASE)

#endif /*_IWDG */

...

#else /* DEBUG */

...

#ifdef _IWDG

EXT IWDG_TypeDef *IWDG;

#endif /*_IWDG */

...

#endif
```

在使用调试模式的时候，IWDG 指针在 *stm32f10x_lib.c* 中初始化了：

```
#ifdef _IWDG
```

IWDG = (IWDG_TypeDef *) IWDG_BASE;

#endif /*_IWDG */

如果想要访问独立的看门狗寄存器，_IWDG 一定要在 *stm32f10x_conf.h* 定义，

如下：

#define _IWDG

12.2 固件库函数

表 254 列出了 IWDG 固件库函数。

表 254. IWDG 固件库函数

函数名	描述
IWDG_WriteAccessCmd	使能（或禁能）对 IWDG_PR 和 IWDG_RLR 寄存器的写访问
IWDG_SetPrescaler	设置 IWDG 预分频因子寄存器的值
IWDG_SetReload	设置 IWDG 重载寄存器的值
IWDG_ReloadCounter	通过在重载寄存器中定义了的值重载 IWDG 计数器
IWDG_Enable	使 能 IWDG
IWDG_GetFlagStatus	检验 IWDG 标志是否置位

12.2.1 IWDG_WriteAccessCmd 函数

表 255 描述了 IWDG_WriteAccessCmd 函数。

表 255.IWDG_WriteAccessCmd 函数

函数名	IWDG_WriteAccessCmd
函数原形	void IWDG_WriteAccessCmd(u16 IWDG_WriteAccess)
功能描述	使能（或禁能）对 IWDG_PR 和 IWDG_RLR 寄存器的写访问
输入参数	IWDG_WriteAccess: 对 IWDG_PR 和 IWDG_RLR 寄存器的写访问的新的状态 参考 :*IWDG_WriteAccess* 部分以获得该参数可取值的更多信息
输出参数	无
返回参数	无
准备条件	无
调用函数	无

IWDG_WriteAccess

这个参数使能对 IWDG_PR 和 IWDG_RLR 寄存器的写访问。（见表 256）

表 256. IWDG_WriteAccess 定义

IWDG_WriteAccess	描述
IWDG_WriteAccess_En	对 IWDG_PR 和 IWDG_RLR 寄存器的写访问被使能

able	
IWDG_WriteAccess_Disable	对 IWDG_PR 和 IWDG_RLR 寄存器的写访问被使能

如：

```
/* Enable write access to IWDG_PR and IWDG_RLR registers */
IWDG_WriteAccessCmd(IWDG_WriteAccess_Enable);
```

12.2.2 IWDG_SetPrescaler 函数

表 257 描述了 IWDG_SetPrescaler 函数。

表 257. IWDG_SetPrescaler 函数

函数名	IWDG_SetPrescaler
函数原形	void IWDG_SetPrescaler(u8 IWDG_Prescaler)
功能描述	设置 IWDG 预分频值
输入参数	IWDG_Prescaler:：IWDG 预分频寄存器的值 参考：*IWDG_Prescaler* 部分以获得该参数可取值的更多信息
输出参数	无
返回参数	无
准备条件	无
调用函数	无

IWDG_Prescaler

这个参数选择了 IWDG 预比例因子寄存器的值。（见表 258）

表 258.IWDG_Prescaler 定义

IWDG_Prescaler	描述
IWDG_Prescaler_4	IWDG 预分频因子设置为 4
IWDG_Prescaler_8	IWDG 预分频因子设置为 8
IWDG_Prescaler_16	IWDG 预分频因子设置为 16
IWDG_Prescaler_32	IWDG 预分频因子设置为 32
IWDG_Prescaler_64	IWDG 预分频因子设置为 64
IWDG_Prescaler_128	IWDG 预分频因子设置为 128
IWDG_Prescaler_256	IWDG 预分频因子设置为 256

如：

```
/* Set IWDG prescaler to 8 */

IWDG_SetPrescaler(IWDG_Prescaler_8);
```

12.2.3 IWDG_SetReload 函数

表 259 描述了 IWDG_SetReload 函数

表 259. IWDG_SetReload 函数

函数名	IWDG_SetReload

函数原形	void IWDG_SetReload(u16 Reload)
功能描述	设置 IWDG 的重载值
输入参数	Reload: IWDG 重载寄存器的值 这个参数一定是在’0’到’0x0FFF’之间的数
输出参数	无
返回参数	无
准备条件	无
调用函数	无

例子：

```
/* Set IWDG reload value to 0xFFF */
IWDG_SetReload(0xFFF);
```

12.2.4 IWDG_ReloadCounter 函数

表 260 描述了 IWDG_ReloadCounter 函数

表 260. IWDG_ReloadCounter 函数

函数名	IWDG_ReloadCounter
函数原形	void IWDG_ReloadCounter(void)
功能描述	通过在重载寄存器中定义了的值重载 IWDG 计数器 （对 IWDG_PR 和 IWDG_RLR 寄存器的写访问被禁能）

输入参数	无
输出参数	无
返回参数	无
准备条件	无
调用函数	无

例子：

```
/* Reload IWDG counter */
IWDG_ReloadCounter();
```

12.2.5 IWDG_Enable 函数

表 261 描述了 IWDG_Enable 函数。

表 261.IWDG_Enable 函数

函数名	IWDG_Enable
函数原形	void IWDG_Enable(void)
功能描述	使能 IWDG （禁能对 IWDG_PR 和 IWDG_RLR 寄存器的写访问）
输入参数	无
输出参数	无
返回参数	无
准备条件	无

调用函数	无

例子：

```
/* Enable IWDG */

IWDG_Enable();
```

12.2.6 IWDG_GetFlagStatus 函数

表 262 描述了 IWDG_GetFlagStatus 函数。

表 262. IWDG_GetFlagStatus 函数

函数名	IWDG_GetFlagStatus
函数原形	FlagStatus IWDG_GetFlagStatus(u16 IWDG_FLAG)
功能描述	检验指定的 IWDG 标志是否置位
输入参数	IWDG_FLAG:检查的标志 **参考：***IWDG_FLAG* 部分以获得该参数可取值的更多信息。
输出参数	无
返回参数	IWDG_FLAG 新的状态（置位或复位）
准备条件	无
调用函数	无

IWDG_FLAG

可以通过 IWDG_GetFlagStatus 函数来检查 IWDG 标志的状态，如表 263 所示。

表 263.IWDG_FLAG 定义

IWDG_FLAG	描述
IWDG_FLAG_PVU	预分频因子值更新了
IWDG_FLAG_RVU	重载值更新了

例子：

```
/* Test if a prescaler value update is on going */

FlagStatus Status;

Status = IWDG_GetFlagStatus(IWDG_FLAG_PVU);

if(Status == RESET)

{

...

}

else

{

...

}
```

13 嵌套向量中断控制器（NVIC）

NVIC 驱动器可用于多种用途.例如使能或关闭 IRQ 中断，使能或关闭单独的 IRQ 通道,同时改变 IRQ 通道的优先级。

13.1 节 :NVIC 寄存器结构 描述了在 NVIC 固件库中使用的数据结构。***13.2 节 :固件库函数*** 介绍了固件库函数。

13.1 NVIC 寄存器结构

NVIC 寄存器结构，NVIC_TypeDef,定义在 *stm32f10x_map.h* 文件中，

如：

```
typedef struct
{
vu32 Enable[2];
u32 RESERVED0[30];
vu32 Disable[2];
u32 RSERVED1[30];
vu32 Set[2];
u32 RESERVED2[30];
vu32 Clear[2];
u32 RESERVED3[30];
vu32 Active[2];
u32 RESERVED4[62];
```

```
vu32 Priority[11];

} NVIC_TypeDef;/* NVIC Structure */

typedef struct

{

vu32 CPUID;

vu32 IRQControlState;

vu32 ExceptionTableOffset;

vu32 AIRC;

vu32 SysCtrl;

vu32 ConfigCtrl;

vu32 SystemPriority[3];

vu32 SysHandlerCtrl;

vu32 ConfigFaultStatus;

vu32 HardFaultStatus;

vu32 DebugFaultStatus;

vu32 MemoryManageFaultAddr;

vu32 BusFaultAddr;

} SCB_TypeDef; /* System Control Block Structure */
```

表 264 给出了 NVIC 寄存器表

表 264.NVIC 寄存器

寄存器	描述
Enable	中断置位使能寄存器
Disable	中断清除使能寄存器
Set	中断置位挂起寄存器
Clear	中断清除挂起寄存器
Active	中断活动位寄存器
Priority	中断优先级寄存器
CPUID	CPUID 基寄存器
IRQControlState	中断控制状态寄存器
ExceptionTableOffset	向量表偏移寄存器
AIRC	应用中断/复位控制寄存器
SysCtrl	系统控制寄存器
ConfigCtrl	配置控制寄存器
SystemPriority	系统处理优先级寄存器
SysHandlerCtrl	系统处理控制和状态寄存器
ConfigFaultStatus	配置出错状态寄存器
HardFaultStatus	硬件出错状态寄存器
DebugFaultStatus	调试出错寄存器
MemoryManangeFault Addr	存贮器管理出错地址寄存器
BusFaultAddr	总线出错地址

NVIC 外围模块声明在 *stm32f10x_map.h* 文件中：

```
...
#define SCS_BASE ((u32)0xE000E000)
#define NVIC_BASE (SCS_BASE + 0x0100)
#define SCB_BASE (SCS_BASE + 0x0D00)
...
#ifndef DEBUG
...
#ifdef _NVIC
#define NVIC ((NVIC_TypeDef *) NVIC_BASE)
#define SCB ((SCB_TypeDef *) SCB_BASE)
#endif /*_NVIC */
...
#else /* DEBUG */
...
#ifdef _NVIC
EXT NVIC_TypeDef *NVIC;
EXT SCB_TypeDef *SCB;
#endif /*_NVIC */
...
#endif
```

在调试模式时，NVIC 和 SCB 指针在 *stm32f10x_lib.c* 文件中初始化：

```
#ifdef _NVIC
NVIC = (NVIC_TypeDef *) NVIC_BASE;
SCB = (SCB_TypeDef *) SCB_BASE;
#endif /*_NVIC */
```

如果要访问 NVIC 寄存器，一定要在 stm32f10x_conf.h 文件中定义 _NVIC，

如：

```
#define _NVIC
```

13.2 固件库函数

表 265 列出了 NVIC 固件库函数。

表 265.NVIC 固件库函数

函数名	描述
NVIC_DeInit	复位 NVIC 外围设备寄存器为默认的复位值
NVIC_SCBDeInit	复位 SCB 外围设备寄存器为默认的复位值
NVIC_PriorityGroupConfig	配置优先级组：优先级和子优先级
NVIC_Init	通过 NVIC_InitStruct 中的给定参数来初始化外围设备
NVIC_StructInit	给每一个 NVIC_InitStruct 成员填上默认值

NVIC_SETPRIMASK	使能 PRIMASK 优先级：把执行的优先级提升为 0.
NVIC_RESETPRIMASK	使 PRIMASK 优先级无效。
NVIC_SETFAULTMASK	使能 FAULTMASK 优先级：把执行的优先级提升为 - 1.
NVIC_RESETFAULTMASK	使 FAULTMASK 优先级无效。
NVIC_BASEPRICONFIG	执行优先级能够从 N(可配置的最低优先级)改变为 1.
NVIC_GetBASEPRI	返回 BASEPRI 掩码值
NVIC_GetCurrentPendingIRQChannel	返回当前挂起服务的 IRQ 通道标识符
NVIC_GetIRQChannelPendingBitStatus	核对指定 IRQ 通道挂起位是否设置。
NVIC_SetIRQChannelPendingBit	设置 NVIC 中断挂起位。
NVIC_ClearIRQChannelPendingBit	清除 NVIC 中断挂起位。
NVIC_GetCurrentActiveHandler	返回当前活动句柄（IRQ 通道和系统句柄）标识符
NVIC_GetIRQChannelActiveBitStatus	核对指定 IRQ 通道活动位是否设置。
NVIC_GetCPUID	返回 Cortex-M3 核的 ID 号，版本号和实现细节。
NVIC_SetVectorTable	设置向量表位置和偏移量.

NVIC_GenerateSystemReset	生成一个系统复位。
NVIC_GenerateCoreReset	生成一个核（Core+NVIC）复位。
NVIC_SystemLPConfig	选择系统进入低功耗模式的条件。
NVIC_SystemHandlerConfig	使能或关闭指定的系统句柄
NVIC_SystemHandlerPriorityConfig	配置指定的系统句柄的优先级。
NVIC_GetSystemHandlerPendingBitStatus	检查指定系统句柄挂起位是否设置。
NVIC_SetSystemHandlerPendingBit	设置系统句柄挂起位。
NVIC_ClearSystemHandlerPendingBit	清除系统句柄挂起位
NVIC_GetSystemHandlerActiveBitStatus	检查指定系统句柄活动位是否设置。
NVIC_GetFaultHandlerSources	返回系统默认句柄源
NVIC_GetFaultAddress	返回生成默认句柄的位置的地址

13.2.1 NVIC_DeInit 函数

表 266 描述了 NVIC_DeInit 函数。

表 266. NVIC_DeInit 函数

函数名	NVIC_DeInit
函数原型	void NVIC_DeInit(void)
行为描述	复位 NVIC 外围设备寄存器为默认的复位值
输入参数	无
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

例子：

```
/*复位 NVIC 外围设备寄存器为默认的复位值*/
NVIC_DeInit();
```

13.2.2 NVIC_SCBDeInit 函数

表 267 描述了 NVIC_SCBDeInit 函数

表 267. NVIC_SCBDeInit 函数

函数名	NVIC_SCBDeInit
函数原型	void NVIC_SCBDeInit(void)
行为描述	复位 SCB 外围设备寄存器为默认的复位值
输入参数	无
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

例子：

/*复位 SCB 外围设备寄存器为默认的复位值*/

NVIC_SCBDeInit();

13.2.3 NVIC_PriorityGoupConfig 函数

表 268 描述了 NVIC_PriorityGoupConfig 函数。

表 268. NVIC_PriorityGoupConfig 函数

函数名	NVIC_PriorityGroupConfig
函数原型	void NVIC_PriorityGroupConfig(u32 NVIC_PriorityGroup)
行为描述	配置优先级组：优先级和字优先级

输入参数	NVIC_PriorityGroup:优先级组位的长度 参考章节：NVIC_PriorityGroup 详细说明了参数允许的值。
输出参数	无
返回参数	无
调用前提条件	优先级组只可能配置一次。
调用函数	无

NVIC_PriorityGroup

这个配置了优先级组位数长度（见表 269）

表 269.NVIC_PriorityGroup

NVIC_PriorityGroup	描述
NVIC_PriorityGroup_0	0 位优先级（pre-emption priority） 4 位子优先级
NVIC_PriorityGroup_1	1 位优先级 3 位子优先级
NVIC_PriorityGroup_2	2 位优先级 2 位子优先级
NVIC_PriorityGroup_3	3 位优先级 1 位子优先级
NVIC_PriorityGroup_4	4 位优先级

	0 位子优先级

例子：

/* 配置优先级组为 1 位 */

NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);

13.2.4 NVIC_Init 函数

表 270 描述了 NVIC_Init 函数

表 270. NVIC_Init 函数

函数名	NVIC_Init
函数原型	void NVIC_Init(NVIC_InitTypeDef* NVIC_InitStruct)
行为描述	通过 NVIC_InitStruct 中的指定参数来初始化外围设备
输入参数	NVIC_InitStruct:一个指向包含指定 NVIC 外围设备配置信息的 NVIC_InitTypeDef 结构 的指针。 参考章节 :NVIC_InitTypeDef 结构 详细说明了参数允许的值。
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

NVIC_InitTypeDef 结构

NVIC_InitTypeDef 结构定义在 *stm32f10x_nvic.h* 文件中：

typedef struct

{

u8 NVIC_IRQChannel;

u8 NVIC_IRQChannelPreemptionPriority;

u8 NVIC_IRQChannelSubPriority;

FunctionalState NVIC_IRQChannelCmd;

} NVIC_InitTypeDef;

NVIC_IRQChannel

这个参数说明了 IRQ 通道是使能还是关闭的。IRQ 通道的清单在表 271 中给出。

表 271.NVIC_IRQChannels

NVIC_IRQChannel	描述
WWDG_IRQChannel	窗口看门狗中断
PVD_IRQChannel	PVD 通过 EXTI 线来检测中断
TAMPER_IRQChannel	篡改中断
RTC_IRQChannel	RTC 全局中断
FlashItf_IRQChannel	FLASH 全局中断
RCC_IRQChannel	RCC 全局中断

EXTI0_IRQChannel	EXTI Line0 中断
EXTI1_IRQChannel	EXTI Line1 中断
EXTI2_IRQChannel	EXTI Line2 中断
EXTI3_IRQChannel	EXTI Line3 中断
EXTI4_IRQChannel	EXTI Line4 中断
DMAChannel1_IRQChannel	DMA 通道 1 全局中断
DMAChannel2_IRQChannel	DMA 通道 2 全局中断
DMAChannel3_IRQChannel	DMA 通道 3 全局中断
DMAChannel4_IRQChannel	DMA 通道 4 全局中断
DMAChannel5_IRQChannel	DMA 通道 5 全局中断
DMAChannel6_IRQChannel	DMA 通道 6 全局中断
DMAChannel7_IRQChannel	DMA 通道 7 全局中断
ADC_IRQChannel	ADC 全局中断
USB_HP_CANTX_IRQChan	USB 高优先级中断 或 CAN TX 中断

nel	
USB_LP_CAN_RX0_IRQChannel	USB 低优先级中断 或 CAN RX0 中断
CAN_RX1_IRQChannel	CAN RX1 中断
CAN_SCE_IRQChannel	CAN SCE 中断
EXTI9_5_IRQChannel	EXTI Line[9:5]中断
TIM1_BRK_IRQChannel	TIM1 打断中断
TIM1_UP_IRQChannel	TIM1 上升中断（TIM1 UP Interrupt）
TIM1_TRG_COM_IRQChannel	TIM1 触发和通信中断
TIM1_CC_IRQChannel	TIM1 捕获比较中断
TIM2_IRQChannel	TIM2 全局中断
TIM3_IRQChannel	TIM 3 全局中断
TIM4_IRQChannel	TIM 4 全局中断
I2C1_EV_IRQChannel	I2C1 事件中断
I2C1_ER_IRQChannel	I2C1 错误中断
I2C2_EV_IRQChannel	I2C2 事件中断
I2C2_ER_IRQChannel	I2C2 错误中断
SPI1_IRQChannel	SPI1 全局中断
SPI2_IRQChannel	SPI2 全局中断
USART1_IRQChannel	USART1 全局中断

USART2_IRQChannel	USART 2 全局中断
USART3_IRQChannel	USART 3 全局中断
EXTI15_10_IRQChannel	EXTI Line[15:10] 中断
RTCAlarm_IRQChannel	通过 EXTILine 的 RTC 警告中断
USBWakeUp_IRQChannel	通过 EXTI Line 的 usb 由挂起到唤醒状态的中断

NVIC_IRQChannelPreemptionPriority

这个成员配置了在 NVIC_IRQChannel 成员中指定的 IRQ 通道的优先级。

这个成员的取值列在了表 272 中。

NVIC_IRQChannelSubPriority

这个成员配置了在 NVIC_IRQChannel 成员中指定的 IRQ 通道的子优先级。

这个成员的取值列在了表 272 中。

表 272 列出了根据 NVIC_PriorityGroupConfig 函数执行的优先级组配置的优先级和子优先级允许的值。

表 272 .预空优先级和子优先级值

NVIC_PriorityGroup	NVIC_IRQChannelPreemptionPri	NVIC_IRQChannelSubPriority	描述

	ority		
NVIC_PriorityGroup_0	0	0-15	0 位优先级 4 位子优先级
NVIC_PriorityGroup_1	0-1	0-7	1 位优先级 3 位子优先级
NVIC_PriorityGroup_2	0-3	0-3	2 位优先级 2 位子优先级
NVIC_PriorityGroup_3	0-7	0-1	3 位优先级 1 位子优先级
NVIC_PriorityGroup_4	0-15	0	4 位优先级 0 位子优先级

选择 PriorityGroup_0 时，NVIC_IRQChannelPreemptionPriority 成员在中断通道配置上无效。

选择 PriorityGroup_4 时，NVIC_IRQChannelSubPriority 成员在中断通道配置上无效。

NVIC_IRQChannelCmd

这个参数说明了在 *NVIC_IRQChannel* 成员定义的 IRQ 通道是使能还是无效的。这个参数既可以设置为 ENABLE 又可以设置为 DISABLE.

例子：

```
NVIC_InitTypeDef NVIC_InitStructure;
/* Configure the Priority Grouping with 1 bit */
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
/* Enable TIM3 global interrupt with Preemption Priority 0 and Sub
Priority as 2 */
NVIC_InitStructure.NVIC_IRQChannel = TIM3_IRQChannel;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 2;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_InitStructure(&NVIC_InitStructure);
/* Enable USART1 global interrupt with Preemption Priority 1 and Sub
Priority as 5 */
NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQChannel;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 5;
NVIC_InitStructure(&NVIC_InitStructure);
/* Enable RTC global interrupt with Preemption Priority 1 and Sub
Priority as 7 */
```

```
NVIC_InitStructure.NVIC_IRQChannel = RTC_IRQChannel;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 7;
NVIC_InitStructure(&NVIC_InitStructure);
/* Enable EXTI4 interrupt with Preemption Priority 1 and Sub
Priority as 7 */
NVIC_InitStructure.NVIC_IRQChannel = EXTI4_IRQChannel;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 7;
NVIC_InitStructure(&NVIC_InitStructure);
/* TIM3 interrupt priority is higher than USART1, RTC and EXTI4
interrupts priorities. USART1 interrupt priority is higher than RTC
and EXTI4 interrupts priorities. RTC interrupt priority is higher
than EXTI4 interrupt prioriy. */
```

13.2.5 NVIC_StructInit 函数

表 273 描述了 NVIC_StructInit 函数.

表 273. NVIC_StructInit 函数

函数名	NVIC_StructInit
函数原型	void NVIC_StructInit (NVIC_InitTypeDef* NVIC_InitStruct)
行为描述	给每一个 NVIC_InitStruct 成员填上默认值

输入参数	NVIC_InitStruct:指向将被初始化的 NVIC_InitTypeDef 结构的指针.
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

NVIC_InitStruct 成员的默认值如下:

表 274.NVIC_InitStruct 默认值

成员	默认值
NVIC_IRQChannel	0x0
NVIC_IRQChannelPreemptionPriority	0
NVIC_IRQChannelSubPriority	0
NVIC_IRQChannelCmd	DISABLE

例子:

```
/* 下面例子显示了如何初始化一个 NVIC_InitTypeDef 结构*/

NVIC_InitTypeDef NVIC_InitStructure;

NVIC_StructInit(&NVIC_InitStructure);
```

13.2.6 NVIC_SETPRIMASK 函数

表 275 描述了 NVIC_SETPRIMASK 函数

表 275. NVIC_SETPRIMASK 函数(1)(2)(3)

函数名	NVIC_SETPRIMASK
函数原型	void NVIC_SETPRIMASK(void)
行为描述	使能 PRIMASK 优先级：把执行的优先级提升为 0.
输入参数	无
输出参数	无
返回参数	无
调用前提条件	无
调用函数	__SETPRIMASK()

这个函数是由汇编器编码的

这个函数只影响组优先级,不影响子优先级.

在设置 PRIMASK 寄存器之前,当从一个例外返回使能其它例外时,推荐清除它

例子:

```
/* 使能 PRIMASK 优先级 */
NVIC_SETPRIMASK();
```

13.2.7 NVIC_RESETPRIMASK 函数

表 276 描述了 NVIC_RESETPRIMASK 函数.

表 276. NVIC_RESETPRIMASK 函数(1)

函数名	NVIC_RESETPRIMASK
函数原型	void NVIC_RESETPRIMASK(void)
行为描述	使 PRIMASK 优先级无效。
输入参数	无
输出参数	无
返回参数	无
调用前提条件	无
调用函数	__RESETPRIMASK()

这个函数是由汇编器编码的.

例子:

```
/* 使能 PRIMASK 优先级 */

NVIC_RESETPRIMASK();
```

13.2.8 NVIC_SETFAULTMASK 函数

表 277 描述了 NVIC_SETFAULTMASK 函数.

表 277. NVIC_SETFAULTMASK 函数

函数名	NVIC_SETFAULTMASK
函数原型	void NVIC_SETFAULTMASK(void)
行为描述	使能 FAULTMASK 优先级：把执行的优先级提升为 - 1.
输入参数	无
输出参数	无
返回参数	无
调用前提条件	无
调用函数	__SETFAULTMASK()

这个函数是由汇编器编码的.

这个函数只影响组优先级,不影响子优先级.

FAULTMASK 只有在执行优先级小于-1 时设置.设置 FaultMask 使例外句柄的优先级提升为 HardFault 级.FAULMASK 会在除 NMI 之外的所有其它例外返回时自动清除.

例子:

```
/* 使能 FAULTMASK 优先级 */
NVIC_SETFAULTMASK();
```

13.2.9 NVIC_RESETFAULTMASK 函数

表 278 描述了 NVIC_RESETFAULTMASK 函数.

表 278. NVIC_RESETFAULTMASK 函数(1)

函数名	NVIC_RESETFAULTMASK
函数原型	Void NVIC_RESETFAULTMASK(void)
行为描述	使 FAULTMASK 优先级无效。
输入参数	无
输出参数	无
返回参数	无
调用前提条件	无
调用函数	__RESETFAULTMASK()

这个函数是由汇编器编码的.

例子:

/*使 FAULTMASK 优先级无效*/

NVIC_RESETFAULTMASK();

13.2.10 NVIC_BASEPRICONFIG 函数

表 279 描述了 NVIC_BASEPRICONFIG 函数.

表 279. NVIC_BASEPRICONFIG 函数(1)(2)(3)

函数名	NVIC_BASEPRICONFIG
函数原型	void NVIC_BASEPRICONFIG(u32 NewPriority)
行为描述	执行优先级能够从 N(可配置的最低优先级)改变为 1.
输入参数	NewPriority:执行优先级的新优先级值

输出参数	无
返回参数	无
调用前提条件	无
调用函数	__BASEPRICONFIG()

这个函数是由汇编器编码的.

这个函数只影响组优先级,不影响子优先级.

BASEPRI 值可能从 N(可配置的最低优先级)到 1.把这个寄存器清 0 对当前优先级没有影响.一个非零值能以优先级掩码执行,当 BASEPRI 定义的优先级比当前执行优先级高时,将影响执行优先级.

例子:

```
/* 将执行优先级掩码为 10 */
__BASEPRICONFIG(10);
```

13.2.11 NVIC_GetBASEPRI 函数

表 280 描述了 NVIC_GetBASEPRI 函数.

表 280. NVIC_GetBASEPRI 函数(1)

函数名	NVIC_GetBASEPRI
函数原型	u32 NVIC_GetBASEPRI(void)
行为描述	返回 BASEPRI 掩码值
输入参数	无

输出参数	无
返回参数	无
调用前提条件	无
调用函数	__GetBASEPRI()

这个函数是由汇编器编码的.

例子:

```
/* 取得执行优先级的值*/
u32 BASEPRI_Mask = 0;
BASEPRI_Mask = NVIC_GetBASEPRI();
```

13.2.12 NVIC_GetCurrentPendingIRQChannel 函数

表 281 描述了 NVIC_GetCurrentPendingIRQChannel 函数.

表 281. NVIC_GetCurrentPendingIRQChannel 函数

函数名	NVIC_GetCurrentPendingIRQChannel
函数原型	u16 NVIC_GetCurrentPendingIRQChannel(void)
行为描述	返回当前挂起的 IRQ 通道（channel）标识符
输入参数	无
输出参数	无
返回参数	挂起 IRQ 通道的标识符
调用前提条件	无

调用函数	无

例子:

/* 取得当前挂起的 IRQ 通道（channel）标识符 */

u16 CurrentPendingIRQChannel;

CurrentPendingIRQChannel = NVIC_GetCurrentPendingIRQChannel();

13.2.13 NVIC_GetIRQChannelPendingBitStatus 函数

表 282 描述了 NVIC_GetIRQChannelPendingBitStatus 函数.

表 282. NVIC_GetIRQChannelPendingBitStatus 函数

函数名	NVIC_GetIRQChannelPendingBitStatus
函数原型	ITStatus NVIC_GetIRQChannelPendingBitStatus(u8 NVIC_IRQChannel)
行为描述	检查指定 IRQ 通道挂起位是否置位。
输入参数	NVIC_IRQChannel:要检查的的中断挂起位. 参考章节:*NVIC_IRQChannel* 详细说明了这个参数允许的值.
输出参数	无
返回参数	IRQ 通道挂起位的新状态(SET or RESET)
调用前提条件	无
调用函数	无

例子:

```
/* 取得 ADC_IRQChannel 的 IRQ 通道挂起位的状态 */
ITStatus IRQChannelPendingBitStatus;
IRQChannelPendingBitStatus =
NVIC_GetIRQChannelPendingBitStatus(ADC_IRQChannel);
```

13.2.14 NVIC_SetIRQChannelPendingBit 函数

表 283 描述了 NVIC_SetIRQChannelPendingBitStatus 函数.

表 283. NVIC_SetIRQChannelPendingBitStatus 函数

函数名	NVIC_SetIRQChannelPendingBit
函数原型	void NVIC_SetIRQChannelPendingBit(u8 NVIC_IRQChannel)
行为描述	设置 NVIC 中断挂起位。
输入参数	NVIC_IRQChannel:指定了要置位的中断挂起位 参考章节:*NVIC_IRQChannel* 详细说明了这个参数允许的值.
输出参数	无
返回参数	无
调用前提条件	无

调用函数	无

例子:

/* 设置 SPI1 全局中断挂起位*/

NVIC_SetIRQChannelPendingBit(SPI1_IRQChannel);

13.2.15 NVIC_ClearIRQChannelPendingBit 函数

表 284 描述了 NVIC_ClearIRQChannelPendingBit 函数

表 284. NVIC_ClearIRQChannelPendingBit 函数

函数名	NVIC_ClearIRQChannelPendingBit
函数原型	void NVIC_ClearIRQChannelPendingBit(u8 NVIC_IRQChannel)
行为描述	清除 NVIC 中断挂起位。
输入参数	NVIC_IRQChannel:指定了要清除的中断挂起位 参考章节:*NVIC_IRQChannel* 详细说明了这个参数允许的值.
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

例子:

```
/* 清除 ADC IRQ 通道的挂起位*/
NVIC_ClearIRQChannelPendingBit(ADC_IRQChannel);
```

13.2.16 NVIC_GetCurrentActiveHandler 函数

表 285 描述了 NVIC_GetCurrentActiveHandler 函数.

表 285. NVIC_GetCurrentActiveHandler 函数

函数名	NVIC_GetCurrentActiveHandler
函数原型	u16 NVIC_GetCurrentActiveHandler(void)
行为描述	返回当前活动句柄（IRQ 通道和系统句柄）的标识符
输入参数	无
输出参数	无
返回参数	活动句柄标识符
调用前提条件	无
调用函数	无

例子:

```
/* 当前活动句柄的标识符 */
u16 CurrentActiveHandler;
CurrentActiveHandler = NVIC_GetCurrentActiveHandler();
```

13.2.17 NVIC_GetIRQChannelActiveBitStatus 函数

表 286 描述了 NVIC_GetIRQChannelActiveBitStatus 函数.

表 286. NVIC_GetIRQChannelActiveBitStatus 函数

函数名	NVIC_GetIRQChannelActiveBitStatus
函数原型	ITStatus NVIC_GetIRQChannelActiveBitStatus(u8 NVIC_IRQChannel)
行为描述	检查指定 IRQ 通道活动位设置与否。
输入参数	NVIC_IRQChannel:指定了要检查的中断活动位. 参考章节:*NVIC_IRQChannel* 详细说明了这个参数允许的值.
输出参数	无
返回参数	IRQ 通道活动位的新状态(SET or RESET)
调用前提条件	无
调用函数	无

例子:

```
/* 取得 ADC_IRQChannel 的活动 IRQ 通道的状态 */
ITStatus IRQChannelActiveBitStatus;
IRQChannelActiveBitStatus =
NVIC_GetIRQChannelActiveBitStatus(ADC_IRQChannel);
```

13.2.18 NVIC_GetCPUID 函数

表 287 描述了 NVIC_GetCPUID 函数.

表 287. NVIC_GetCPUID 函数

函数名	NVIC_GetCPUID
函数原型	u32 NVIC_GetCPUID(void)
行为描述	返回 Cortex-M3 核的 ID 号，版本号和实现细节。
输入参数	无
输出参数	无
返回参数	CPU ID
调用前提条件	无
调用函数	无

例子:

```
/* 取得 CPU ID */
u32 CM3_CPUID;
CM3_CPUID = NVIC_GetCPUID();
```

13.2.19 NVIC_SetVectorTable 函数

表 288 描述了 NVIC_SetVectorTable 函数.

表 288. NVIC_SetVectorTable 函数

函数名	NVIC_SetVectorTable
函数原型	void NVIC_SetVectorTable(u32 NVIC_VectTab, u32 Offset)
行为描述	设置向量表的位置和偏移量.
输入参数 1	NVIC_VectTab:指定中断向量表在 RAM 还是在代码存储器中. 参考章节:*NVIC_VectTab* 详细说明了这个参数允许的值.
输入参数 2	Offset:向量表的基址偏移域 对于 FLASH 这个值一定要比 0x08000100 高,对于 RAM 这个值比 0x100 高.这个值一定要是 256(64*4)的倍数.
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

NVIC_VectTab

这个参数定义了表的基地址(见表 289).

表 289.NVIC_VectTab 值

NewTableBase	描述
NVIC_VectTab_FLASH	在 FLASH 上的向量表
NVIC_VectTab_RAM	在 RAM 上的向量表

例子:

```
/* 在 FLASH 中向量表的基址是 0x0 */
NVIC_SetVectorTable(NVIC_VectTab_FLASH, 0x0);
```

13.2.20 NVIC_GenerateSystemReset 函数

表 290 描述了 NVIC_GenerateSystemReset 函数.

表 290. NVIC_GenerateSystemReset 函数

函数名	NVIC_GenerateSystemReset
函数原型	void NVIC_GenerateSystemReset(void)
行为描述	生成一个系统复位。
输入参数	无
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

例子:

```
/*生成一个系统复位*/
NVIC_GenerateSystemReset();
```

13.2.21 NVIC_GenerateCoreReset 函数

表 291 描述了 NVIC_GenerateCoreReset 函数.

表 291. NVIC_GenerateCoreReset 函数

函数名	NVIC_GenerateCoreReset
函数原型	void NVIC_GenerateCoreReset(void)
行为描述	生成一个核（Core+NVIC）复位
输入参数	无
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

例子:

/*生成一个核复位*/

NVIC_GenerateCoreReset();

13.2.22 NVIC_SystemLPConfig 函数

表 292 描述了 NVIC_SystemLPConfig 函数

表 292. NVIC_SystemLPConfig 函数

函数名	NVIC_SystemLPConfig
函数原型	void NVIC_SystemLPConfig(u8 LowPowerMode, FunctionalState , NewState)

行为描述	选择系统进入低功耗模式的条件。
输入参数 1	LowPowerMode:系统为进入低功耗模式的新模式. 参考章节:*LowPowerMode* 详细说明了这个参数和取值.
输入参数 2	NewState:LP 条件的新状态. 这个参数可以是:ENABLE 或 DISABLE
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

LowPowerMode

这个参数了设备的低功耗模式(见表 293).

表 293.LowerPowerMode 定义

LowPowerMode	描述
NVIC_LP_SEVONPEND	从挂起状态唤醒
NVIC_LP_SLEEPDEEP	深度睡眠使能
NVIC_LP_SLEEPONEXIT	从 ISR 退出后睡眠(Sleep on ISR exit)

例子:

/*唤醒处于中断挂起状态的系统 */

NVIC_SystemLPConfig(SEVONPEND, ENABLE);

13.2.23 NVIC_SystemHandlerConfig 函数

表 294 描述了 NVIC_SystemHandlerConfig 函数

表 294. NVIC_SystemHandlerConfig 函数

函数名	NVIC_SystemHandlerConfig
函数原型	void NVIC_SystemHandlerConfig(u32 SystemHandler, FunctionalState , NewState)
行为描述	使能或关闭指定的系统句柄。
输入参数 1	SystemHandler:系统句柄有效或无效. 参考章节:*SystemHandler* 详细说明了这个参数和取值.
输入参数 2	NewState:指定系统句柄的新状态. 这个参数可以是:ENABLE 或 DISABLE
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

SystemHandler

这个参数选择系统句柄是使能还是关闭(见表 295).

表 295.SystemHandler 类别

SystemHandler	描述
SystemHandler_MemoryManage	内存管理句柄
SystemHandler_BusFault	总线故障句柄
SystemHandler_UsageFault	使用故障句柄

系统句柄参数允许同时配置 NVIC 寄存器,SCB 寄存器,索引位.系统句柄以 23 位编码.见:表 296,表 297,表 298 ,表 299, 表 300,表 301 ,表 302,表 303, 表 304 和表 305.

例子:

/* 使能内存管理句柄 */

NVIC_SystemHandlerConfig(SystemHandler_MemoryManage, ENABLE);

表 296

系统句柄	位																								
	2	1	0	9	8	7	6	5	4	3	2	1	0												x1F
SystemHandler_NMI (见表 297)	保留																		0x1F						x1F
SystemHandler_HardFault（见	保留			0	保留																				x0

表 298)									
SystemHandler_MemoryManage (见表 299)		0	1	0x0	0xD	0	0	es	0x10	x4343 0
SystemHandler_BusFault (见表 300)		1	1	1	0xE	1	0	es	0x11	x54 7931
SystemHandler_UsageFault (见表 301)		2	1	0x3	保留	2	0	es	0x12	x24 C232
SystemHandler_SVCall（见表 302)	保留			0x7	0xF	3	1	保留		x1FF4 0
SystemHandl	保		2	0x8	保留	0	2	保留		

er_ DebugMonitor (见表 303)	留							xA0080
SystemHandler_ PSV （见表 304)	保留		0xA	保留	2	2	0x1C	x28 29C
SystemHandler_ SysTick（见表 305)	保留		0xB	保留	3	2	0x1A	x2C 39A

表 297 SystemHandler_NMI 定义

位	NMI	
	寄存器/位	函数
[4：0]	- IRQControlState - NMIPENDSET[31]	NVIC_SetSystemHandlerPendingBit

5	未使用
[7：6]	未使用
[9：8]	未使用
[13：10]	未使用
[17：14]	未使用
[19：18]	未使用
[21：20]	未使用
22	未使用

表 298 SystemHandler_HardFault 定义

位	硬错误	
	寄存器/位	函数
[4：0]	未使用	
5	未使用	
[7：6]	未使用	
[9：8]	未使用	
[13：10]	未使用	
[17：14]	未使用	
[19：18]	- HardFaultStatus	NVIC_GetFaultHandlerSources
[21：20]		
22	未使用	

表 299 SystemHandler_MemoryManage 定义

位	内存管理	
	寄存器/位	函数
[4：0]	- SysHandlerCtrl - MEMFAULTENA[16]	NVIC_SystemHandlerConfig
5	未使用	
[7：6]	- SystemPriority[0] - PRI_4[7:0]	NVIC_SystemHandlerPriorityConfig
[9：8]		
[13：10]	- SysHandlerCtrl - MEMFAULTPENDED[13]	NVIC_GetSystemHandlerPendingBitStatus
[17：14]	- SysHandlerCtrl - MEMFAULTACT[0]	NVIC_GetSystemHandlerActiveBitStatus
[19：18]	- ConfigFaultStatus - [7:0]	NVIC_GetFaultHandlerSources
[21：20]		
22	- MemoryManageFaultAddr	NVIC_GetFaultAddress

表 300 SystemHandler_BusFault 函数

位	内存管理	
	寄存器/位	函数

[4：0]	- SysHandlerCtrl - BUSFAULTENA[17]	NVIC_SystemHandlerConfig
5	未使用	
[7：6]	- SystemPriority[0] - PRI_5[15:8]	NVIC_SystemHandlerPriorityConfig
[9：8]		
[13：10]	- SysHandlerCtrl - BUSFAULTPENDED[14]	NVIC_GetSystemHandlerPendingBitStatus
[17：14]	- SysHandlerCtrl - BUSFAULTACT[1]	NVIC_GetSystemHandlerActiveBitStatus
[19：18]	- ConfigFaultStatus - [15:8]	NVIC_GetFaultHandlerSources
[21：20]		
22	- BusFaultAddr	NVIC_GetFaultAddress

表 301 SystemHandler_UsageFault 定义

位	内存管理	
	寄存器/位	函数
[4：0]	- SysHandlerCtrl - USGFAULTENA[18]	NVIC_SystemHandlerConfig
5	未使用	
[7：6]	- SystemPriority[0]	NVIC_SystemHandlerPriorityConfig

[9：8]	- PRI_6[23:16]	
[13：10]	未使用	
[17：14]	- SysHandlerCtrl - USGFAULTACT[3]	NVIC_GetSystemHandlerActiveBitStatus
[19：18]	- ConfigFaultStatus - [31:16]	NVIC_GetFaultHandlerSources
[21：20]		
22	未使用	

表 302 SystemHandler_SVCall 定义

位	内存管理	
	寄存器/位	函数
[4：0]	未使用	
5	未使用	
[7：6]	- SystemPriority[1] - PRI_11[31:24]	NVIC_SystemHandlerPriorityConfig
[9：8]		
[13：10]	- SysHandlerCtrl - SVCALLPENDED[15]	NVIC_GetSystemHandlerPendingBitStatus
[17：14]	- SysHandlerCtrl - SVCALLACT[7]	NVIC_GetSystemHandlerActiveBitStatus
[19：18]	未使用	

[21：20]	未使用
22	未使用

表 303 SystemHandler_DebugMonitor 定义

位	内存管理	
	寄存器/位	函数
[4：0]	未使用	
5	未使用	
[7：6]	- SystemPriority[2] - PRI_12[7:0]	NVIC_SystemHandlerPriorityConfig
[9：8]		
[13：10]	未使用	
[17：14]	- SysHandlerCtrl - MONITORACT[8]	NVIC_GetSystemHandlerActiveBitStatus
[19：18]	- DebugFaultStatus	NVIC_GetFaultHandlerSources
[21：20]		
22	未使用	

表 304 SystemHandler_PSV 定义

位	内存管理	
	寄存器/位	函数
[4：0]	- IRQControlState	NVIC_SetSystemHandlerPendingBit

	- PENDSVSET[28]	
	- IRQControlState - PENDSVCLR[27]	NVIC_ClearSystemHandlerPendingBit
5	未使用	
[7：6]	- SystemPriority[2] - PRI_14[23:16]	NVIC_SystemHandlerPriorityConfig
[9：8]		
[13：10]	未使用	
[17：14]	- SysHandlerCtrl - PENDSVACT[10]	NVIC_GetSystemHandlerActiveBitStatus
[19：18]	未使用	
[21：20]	未使用	
22	未使用	

表 305 SystemHandler_SysTick 定义

位	内存管理	
	寄存器/位	函数
[4：0]	- IRQControlState - PENDSTSET[26]	NVIC_SetSystemHandlerPendingBit
	- IRQControlState - PENDSVCLR[25]	NVIC_ClearSystemHandlerPendingBit
5	未使用	

[7：6]	- SystemPriority[2] - PRI_15[31:24]	NVIC_SystemHandlerPriorityConfig
[9：8]		
[13：10]	未使用	
[17：14]	- SysHandlerCtrl - SYSTICKACT[11]	NVIC_GetSystemHandlerActiveBitStatus
[19：18]	未使用	
[21：20]	未使用	
22	未使用	

13.2.24 NVIC_SystemHandlerPriorityConfig 函数

表 306 描述了 NVIC_SystemHandlerPriorityConfig 函数.

表 306. NVIC_SystemHandlerPriorityConfig 函数

函数名	NVIC_SystemHandlerPriorityConfig
函数原型	void NVIC_SystemHandlerPriorityConfig(u32 SystemHandler, u8 SystemHandlerPreemptionPriority, u8 SystemHandlerSubPriority)
行为描述	配置指定系统句柄的优先级。
输入参数 1	SystemHandler:使能或关闭系统句柄. 参考章节:*SystemHandler* 详细说明了这个参数和取值.

输入参数 2	SystemHandlerPreemptionPriority:指定系统句柄的新的优先级组. **参考章节:** NVIC_IRQChannelPreemptionPriority 详细说明了这个参数和取值.
输入参数 3	SystemHandlerSubPriority:指定系统句柄的新子优先级. **参考章节:** NVIC_IRQChannelSubPriority 详细说明了这个参数和取值.
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

SystemHandler

这个参数选择了将要配置的系统句柄(见 表 307).

表 307.SystemHandler 类别

SystemHandler	描述
SystemHandler_MemoryManage	内存管理句柄
SystemHandler_BusFault	总线故障句柄
SystemHandler_UsageFault	使用故障句柄

SystemHandler_SVCall	SVCall 句柄
SystemHandler_DebugMonitor	调试监视句柄
SystemHandler_PSV	PSV 句柄
SystemHandler_SysTick	SysTick 句柄

例子:

/*使用内存管理句柄 */

```
NVIC_SystemHandlerPriorityConfig(SystemHandler_MemoryManage, 2, 8);
```

13.2.25 NVIC_GetSystemHandlerPendingBitStatus 函数

表 308 描述了 NVIC_GetSystemHandlerPendingBitStatus 函数.

表 308. NVIC_GetSystemHandlerPendingBitStatus 函数

函数名	NVIC_GetSystemHandlerPendingBitStatus
函数原型	ITStatus NVIC_GetSystemHandlerPendingBitStatus(u32 SystemHandler)
行为描述	检查指定系统句柄挂起位设置与否。
输入参数	SystemHandler:要检查的系统句柄挂起位. 参考章节: *SystemHandler* 详细说明了这个参数允许的值.
输出参数	无
返回参数	系统句柄挂起位的新状态(SET or RESET)

调用前提条件	无
调用函数	无

SystemHandler

这个参数选择了系统句柄(见 表 309).

表 309.systemHandler 类别

SystemHandler	描述
SystemHandler_MemoryManage	内存管理句柄
SystemHandler_BusFault	总线故障句柄
SystemHandler_SVCall	SVCall 句柄

例子:

```
/* 检查是否发生了内存管理故障*/
ITStatus MemoryHandlerStatus;
MemoryHandlerStatus
=NVIC_GetSystemHandlerPendingBitStatus(SystemHandler_MemoryManage);
```

13.2.26 NVIC_SetSystemHandlerPendingBit 函数

表 310 描述了 NVIC_SetSystemHandlerPendingBit 函数.

表 310. NVIC_SetSystemHandlerPendingBit 函数

函数名	NVIC_SetSystemHandlerPendingBit
函数原型	void NVIC_SetSystemHandlerPendingBit(u32 SystemHandler)
行为描述	设置系统句柄挂起位。
输入参数	SystemHandler:要设置的系统句柄挂起位. 参考章节: *SystemHandler* 详细说明了这个参数允许的值.
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

SystemHandler

这个参数选择了系统句柄(见表 311).

表 311.systemHandler 类别

SystemHandler	描述

SystemHandler_NMI	NMI 句柄
SystemHandler_PSV	PSV 句柄
SystemHandler_SysTick	SysTick 句柄

例子:

```
/* 设置 NMI 挂起位 */
NVIC_SetSystemHandlerPendingBit(SystemHandler_NMI);
```

13.2.27 NVIC_ClearSystemHandlerPendingBit 函数

表 312 描述了 NVIC_ClearSystemHandlerPendingBit 函数.

表 312. NVIC_ClearSystemHandlerPendingBit 函数

函数名	NVIC_ClearSystemHandlerPendingBit
函数原型	void NVIC_ClearSystemHandlerPendingBit(u32 SystemHandler)
行为描述	清除系统句柄挂起位。
输入参数	SystemHandler:要复位的系统句柄挂起位. 参考章节: *SystemHandler* 详细说明了这个参数允许的值.
输出参数	无
返回参数	无
调用前提条件	无

调用函数	无

SystemHandler

这个参数选择了系统句柄(见表 313).

表 313.systemHandler 类别

SystemHandler	描述
SystemHandler_PSV	PSV 句柄
SystemHandler_SysTick	SysTick 句柄

例子:

```
/* 清除 SysTick 挂起位 */
NVIC_ClearSystemHandlerPendingBit(SystemHandler_SysTick);
```

13.2.28 NVIC_GetSystemHandlerActiveBitStatus 函数

表 314 描述了 NVIC_GetSystemHandlerActiveBitStatus 函数.

表 314. NVIC_GetSystemHandlerActiveBitStatus 函数

函数名	NVIC_GetSystemHandlerActiveBitStatus
函数原型	ITStatus NVIC_GetSystemHandlerActiveBitStatus(u32 SystemHandler)
行为描述	检查指定系统句柄活动位设置与否。
输入参数	SystemHandler:要检查的系统句柄活动位.

	参考章节: *SystemHandler* 详细说明了这个参数允许的值.
输出参数	无
返回参数	系统句柄活动位的新状态(SET or RESET).
调用前提条件	无
调用函数	无

SystemHandler

这个参数选择了系统句柄(见表 315).

表 315.systemHandler 类别

SystemHandler	描述
SystemHandler_MemoryManage	内存管理句柄
SystemHandler_BusFault	总线故障句柄
SystemHandler_UsageFault	使用故障句柄
SystemHandler_SVCall	SVCall 句柄
SystemHandler_DebugMonitor	调试检视句柄
SystemHandler_PSV	PSV 句柄
SystemHandler_SysTick	SysTick 句柄

例子：

```
/* 检查总线故障是活动的还是堆栈的（Check if the Bus Fault is active or stacked） */
ITStatus BusFaultHandlerStatus;
BusFaultHandlerStatus =
NVIC_GetSystemHandlerActiveBitStatus(SystemHandler_BusFault);
```

13.2.29 NVIC_GetFaultHandlerSources 函数

表 316 描述了 NVIC_GetFaultHandlerSources 函数。

表 316. NVIC_GetFaultHandlerSources 函数

函数名	NVIC_GetFaultHandlerSources
函数原型	u32 NVIC_GetFaultHandlerSources(u32 SystemHandler)
行为描述	返回系统句柄故障源
输入参数	SystemHandler:将要返回故障源的系统句柄. 参考章节：*SystemHandler* 详细说明了这个参数允许的值.
输出参数	无
返回参数	故障句柄源
调用前提条件	无
调用函数	无

SystemHandler

这个参数选择了系统句柄（见表 317）。

表 317.systemHandler 类别

SystemHandler	描述
SystemHandler_HardFault	硬件故障句柄
SystemHandler_MemoryManage	内存管理句柄
SystemHandler_BusFault	总线故障句柄
SystemHandler_UsageFault	使用故障句柄
SystemHandler_DebugMonitor	调试监视句柄

例子：

```
/* 取得总线故障句柄源 */
u32 BusFaultHandlerSource;
BusFaultHandlerSource
=NVIC_GetFaultHandlerSources(SystemHandler_BusFault);
```

13.2.30 NVIC_GetFaultAddress 函数

表 318 描述了 NVIC_GetFaultAddress 函数

表 318. NVIC_GetFaultAddress 函数

函数名	NVIC_GetFaultAddress
函数原型	u32 NVIC_GetFaultAddress(u32 SystemHandler)
行为描述	返回生成故障句柄位置的地址
输入参数	SystemHandler：将要返回故障地址的系统句柄 参考章节: *SystemHandler* 详细说明了这个参数允许的值.
输出参数	无
返回参数	默认地址
调用前提条件	无
调用函数	无

SystemHandler

这个参数选择了系统句柄（见 表 319）。

表 319.SystemHandler 类别

SystemHandler	描述
SystemHandler_MemoryManage	内存管理句柄
SystemHandler_BusFault	总线故障句柄

例子：

```
/* 获得总线故障句柄的地址*/
u32 BusFaultHandlerAddress;
```

```
BusFaultHandlerAddress =

NVIC_GetFaultAddress(SystemHandler_BusFault);
```

14 Power control(PWR)

PWR 可用于多种目的，包括电源管理和低功耗模式选择。

14.1 节：*PWR 寄存器结构* 描述了 PWR 固件库中的数据结构。14.2 节：*固件库函数* 介绍了固件库函数。

14.1 PWR 寄存器结构

PWR 寄存器结构，PWR_TypeDef,定义在 *stm32f10x_map.h* 文件中。

如下：

```
typedef struct
{
vu32 CR;
vu32 CSR;
} PWR_TypeDef;
```

表 320 给出了 PWR 寄存器的清单。

表 320.PWR 寄存器

寄存器	描述
CR	电源控制寄存器
CSR	电源控制状态寄存器

PWR 外围模块声明在 *stm32f10x_map.h* 文件中。

```
#define PERIPH_BASE ((u32)0x40000000)

#define APB1PERIPH_BASE PERIPH_BASE

#define APB2PERIPH_BASE (PERIPH_BASE + 0x10000)

#define AHBPERIPH_BASE (PERIPH_BASE + 0x20000)

#define PWR_BASE (APB1PERIPH_BASE + 0x7000)

#ifndef DEBUG

...

#ifdef _PWR

#define PWR ((PWR_TypeDef *) PWR_BASE)

#endif /*_PWR */

...

#else /* DEBUG */

...

#ifdef _PWR

EXT PWR_TypeDef *PWR;

#endif /*_PWR */

...

#endif
```

使用调试模式时，PWR 指针初始化在 *stm32f10x_lib.c* 文件中：

```
#ifdef _PWR

PWR = (PWR_TypeDef *) PWR_BASE;
```

```
#endif /*_PWR */
```

要访问 PWR 寄存器，_PWR 一定要在 *stm32f10x_conf.h* 定义。

如下：

```
#define _PWR
```

14.2 固件库函数

表 321 给出了各种 PWR 库函数清单。

表 321.PWR 固件库函数

函数表	描述
PWR_DeInit	复位 PWR 外围寄存器为默认复位值
PWR_BackupAccessCmd	使能或关闭对 RTC 和备份寄存器的访问。
PWR_PVDCmd	使能或关闭电源电压探测器（PVD）。
PWR_PVDLevelConfig	配置由电源电压探测器检测的电压门限值。（ P
PWR_WakeUpPinCmd	使能或关闭唤醒引脚的功能
PWR_EnterSTOPMode	进入 STOP 模式
PWR_EnterSTANDBYMode	进入 STANDBY 模式
PWR_GetFlagStatus	核对指定的 PWR 标志位设置与否
PWR_ClearFlag	清空 PWR 挂起标志位。

14.2.1 PWR_DeInit 函数

表 322 描述了 PWR_DeInit 函数。

表 322. PWR_DeInit 函数

函数名	PWR_DeInit
函数原型	void PWR_DeInit(void)
行为描述	复位 PWR 外围寄存器为默认复位值
输入参数	无
输出参数	无
返回参数	无
调用前提条件	无
调用函数	RCC_APB1PeriphResetCmd

例子：

/*初始化 PWM 寄存器*/

```
PWR_DeInit();
```

14.2.2 PWR_BackupAccessCmd 函数

表 323 描述了 PWR_BackupAccessCmd 函数。

表 323. PWR_BackupAccessCmd 函数

函数名	PWR_BackupAccessCmd
函数原型	void

	PWR_BackupAccessCmd(FunctionalState NewState)
行为描述	使能或关闭对 RTC 和备份寄存器的访问。
输入参数	NewState:访问 RTC 和备份寄存器的新状态。这个参数可以是：ENABLE or DISABLE
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

例子：

```
/* 使能对 RTC 和备份寄存器的访问*/
PWR_BackupAccessCmd(ENABLE);
```

14.2.3 PWR_PVDCmd 函数

表 324 描述了 PWR_PVDCmd 函数。

表 324. PWR_PVDCmd 函数

函数名	PWR_PVDCmd
函数原型	void PWR_PVDCmd(FunctionalState NewState)
行为描述	使能或关闭电源电压探测器（PVD）。

输入参数	NewState:PVD 的新状态。 这个参数可以是：ENABLE or DISABLE
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

例子：

/*使能电源电压探测器(PVD) */

PWR_PVDCmd(ENABLE);

14.2.4 PWR_PVDLevelConfig 函数

表 325 描述了 PWR_PVDLevelConfig 函数

表 325. PWR_PVDLevelConfig 函数

函数名	PWR_PVDLevelConfig
函数原型	void PWR_PVDLevelConfig(u32 PWR_PVDLevel)
行为描述	配置由电源电压探测器探测的电压门限值。（ PVD ）
输入参数	PWR_PVDLevel:PVD 探测电平 参考章节：*PWR_PVDLevel* 详细说明了这个参数的允许值。
输出参数	无

返回参数	无
调用前提条件	无
调用函数	无

PWR_PVDLevel

这个参数配置了 PVD 探测电平值（见表 326 ）。

表 326.PWR_PVDLevel 值

PWR_PVDLevel	描述
PWR_PVDLevel_2V2	PVD 探测电平设置为 2.2V
PWR_PVDLevel_2V3	PVD 探测电平设置为 2.3V
PWR_PVDLevel_2V4	PVD 探测电平设置为 2.4V
PWR_PVDLevel_2V5	PVD 探测电平设置为 2.5V
PWR_PVDLevel_2V6	PVD 探测电平设置为 2.6V
PWR_PVDLevel_2V7	PVD 探测电平设置为 2.7V
PWR_PVDLevel_2V8	PVD 探测电平设置为 2.8V
PWR_PVDLevel_2V9	PVD 探测电平设置为 2.9V

例子：

```
/* 设置 PVD 探测电平为 2.5V */
PWR_PVDLevelConfig(PWR_PVDLevel_2V5);
```

14.2.5 PWR_WakeUpPinCmd 函数

表 327 描述了 PWR_WakeUpPinCmd 函数。

表 327. PWR_WakeUpPinCmd 函数

函数名	PWR_WakeUpPinCmd
函数原型	void PWR_WakeUpPinCmd(FunctionalState NewState)
行为描述	使能唤醒引脚的功能.
输入参数	NewState:唤醒引脚功能的新状态 这个参数可以是：ENALBLE or DISABLE
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

例子：

```
/* 用于唤醒功能的唤醒引脚 */
PWR_WakeUpPinCmd(ENABLE);
```

14.2.6 PWR_EnterSTOPMode 函数

表 328 描述了 PWR_EnterSTOPMode 函数。

表 328. PWR_EnterSTOPMode 函数

函数名	PWR_EnterSTOPMode
函数原型	void PWR_EnterSTOPMode(u32 PWR_Regulator, u8 PWR_STOPEntry)
行为描述	进入 STOP 模式
输入参数 1	PWR_Regulator:STOP 模式的校准状态。 参考章节 :*PWR_Regulator* 详细说明了这个参数的允许值。
输入参数 2	PWR_STOPEntry:指定是进入有 WFI 还是 WFE 指令的 STOP 模式. 参考章节：*PWR_STOPEntry* 详细说明了这个参数的允许值
输出参数	无
返回参数	无
调用前提条件	无
调用函数	__WFI(), __WFE()

PWR_Regulator

这个参数配置了 STOP 模式的校准状态。表 329 说明了 PWR_Regulator 的可能值。

表 329.PWR_Regulator 定义

PWR_Regulator	描述
PWR_Regulator_ON	校准器开启的 STOP 模式
PWR_Regulator_LowPower	在低功耗模式下带有校准器的 STOP 模式。

PWR_STOPEntry

这个参数定义了 STOP 进入模式。

表 330.PWR_STOPEntry 定义

PWR_Regulator	描述
PWR_STOPEntry_WFI	进入有 WFI 指令的 STOP 模式
PWR_STOPEntry_WFE	进入有 WFE 指令的 STOP 模式

例子：

```
/* 校准器开启的情况下将系统置于 STOP 模式中 */

PWR_EnterSTOPMode(PWR_Regulator_ON, PWR_STOPEntry_WFE);
```

14.2.7 PWR_EnterSTANDBYMode 函数

表 331 描述了 PWR_EnterSTANDBYMode 函数。

表 331. PWR_EnterSTANDBYMode 函数

函数名	PWR_EnterSTANDBYMode
函数原型	void PWR_EnterSTANDBYMode(void)
行为描述	进入 STANDBY 模式
输入参数	无
输出参数	无
返回参数	无
调用前提条件	无
调用函数	__WFI()

例子：

```
/*将系统置于 STANDBY 模式*/
PWR_EnterSTANDBYMode();
```

14.2.8 PWR_GetFlagStatus 函数

表 332 描述了 PWR_GetFlagStatus 函数。

表 322. PWR_GetFlagStatus 函数

函数名	PWR_GetFlagStatus
函数原型	FlagStatus PWR_GetFlagStatus(u32 PWR_FLAG)
行为描述	检查指定的 PWR 标志位设置与否

输入参数	PWR_FLAG:要检查的标志。 参考章节：*PWR_FLAG* 详细说明了这个参数的允许值
输出参数	无
返回参数	PWR_FLAG 的新状态（SET or RESET）
调用前提条件	无
调用函数	无

PWR_FLAG

通过调用 PWR_GetFlagStatus 函数检查的 PWR 标志列于表 333 中。

表 333.PWR_Flag 值

PWR_FLAG	描述
PWR_FLAG_WU	Wake-up 标志
PWR_FLAG_SB	StandBy 标志
PWR_FLAG_PVDO	PVD 输出(1)

1.这个标志是只读的，不能被清空。

例子：

```
/* 检查 StandBy 标志是否置位 t */
FlagStatus Status;
Status = PWR_GetFlagStatus(PWR_FLAG_SB);
if(Status == RESET)
{
```

```
...
}
else
{
...
}
```

14.2.9 PWR_ClearFlag 函数

表 334 描述了 PWR_ClearFlag 函数。

表 334. PWR_ClearFlag 函数

函数名	PWR_ClearFlag
函数原型	void PWR_ClearFlag(u32 PWR_FLAG)
行为描述	清空 PWR 挂起标志位。
输入参数	PWR_FLAG:要清空的标志。 参考章节：*PWR_FLAG* 详细说明了这个参数的允许值
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

例子：

```
/* 清除 StandBy 挂起标志位 */

PWR_ClearFlag(PWR_FLAG_SB);
```

15 重启和时钟控制（RCC）

RCC 可用于多种目的,包括时钟配置,外围设备重启和时钟管理.

15.1 节:*RCC 寄存器结构* 描述了 RCC 固件库的数据结构.15.2 节:*固件库函数*介绍了固件库函数.

15.1 RCC寄存器结构

RCC 寄存器结构,RCC_TypeDef,定义在 *stm32f10x_map.h* 文件中.

如:

```
typedef struct
{
vu32 CR;
vu32 CFGR;
vu32 CIR;
vu32 APB2RSTR;
vu32 APB1RSTR;
vu32 AHBENR;
vu32 APB2ENR;
vu32 APB1ENR;
vu32 BDCR;
vu32 CSR;
```

} RCC_TypeDef;

表 335 列出了 RCC 寄存器.

表 335.RCC 寄存器

寄存器	描述
CR	时钟控制寄存器
CFGR	时钟配置寄存器
CIR	时钟中断寄存器
APB2RSTR	APB2 外围设备复位寄存器
APB1RSTR	APB1 外围设备复位寄存器
AHBENR	AHB 外围设备时钟使能寄存器.
APB2ENR	APB2 外围设备时钟使能寄存器
APB1ENR	APB1 外围设备时钟使能寄存器
BDCR	备份域控制寄存器
CSR	控制/状态寄存器

RCC 外围设备声明在同一个文件中:

#define PERIPH_BASE ((u32)0x40000000)

#define APB1PERIPH_BASE PERIPH_BASE

#define APB2PERIPH_BASE (PERIPH_BASE + 0x10000)

#define AHBPERIPH_BASE (PERIPH_BASE + 0x20000)

#define RCC_BASE (AHBPERIPH_BASE + 0x1000)

```
#ifndef DEBUG

...

#ifdef _RCC

#define RCC ((RCC_TypeDef *) RCC_BASE)

#endif /*_RCC */

...

#else /* DEBUG */

...

#ifdef _RCC

EXT RCC_TypeDef *RCC;

#endif /*_RCC */

...

#endif
```

在调试模式下,RCC 指针初始化在 *stm32f10x_lib.c* 文件中:

```
#ifdef _RCC

RCC = (RCC_TypeDef *) RCC_BASE;

#endif /*_RCC */
```

要访问复位时钟寄存器,_RCC 一定要声明在 *stm32f10x_conf.h* 文件中.

如:

```
#define _RCC
```

15.2 固件库函数

表 336 列出了 RCC 库的各种函数.

表 336.RCC 固件库函数

函数名	描述
RCC_DeInit	复位 RCC 外围设备寄存器到默认复位值.
RCC_HSEConfig	配置额外高速振荡器(HSE)
RCC_WaitForHSEStartUp	等待 HSE 启动
RCC_AdjustHSICalibrationValue	校正内部高速振荡器(HSI)的刻度值.
RCC_HSICmd	使能或关闭内部高速振荡器(HIS).
RCC_PLLConfig	配置 PLL,时钟源和乘法因子.
RCC_PLLCmd	使能或关闭 PLL
RCC_SYSCLKConfig	配置系统时钟(SYSCLK).
RCC_GetSYSCLKSource	返回用做系统时钟的时钟源
RCC_HCLKConfig	配置 AHB 时钟(HCLK).
RCC_PCLK1Config	配置低速 APB 时钟(PCLK1).
RCC_PCLK2Config	配置高速 APB 时钟(PCLK2).
RCC_ITConfig	使能或关闭指定的 RCC 中断.
RCC_USBCLKConfig	配置 USB 时钟(USBCLK).
RCC_ADCCLKConfig	配置 ADC 时钟(ADCCKL).

RCC_LSEConfig	配置外部低速振荡器(LSE).
RCC_LSICmd	使能或关闭内部中断低速振荡器(LSI).
RCC_RTCCLKConfig	配置 RTC 时钟(RTCCLK).
RCC_RTCCLKCmd	使能或关闭 RTC 时钟.
RCC_GetClocksFreq	返回片上时钟的不同频率
RCC_AHBPeriphClockCmd	使能或关闭 AHB 外围设备时钟.
RCC_APB2PeriphClockCmd	使能或关闭高速 APB(APB2)外围设备时钟.
RCC_APB1PeriphClockCmd	使能或关闭低速 APB(APB1)外围设备时钟
RCC_APB2PeriphResetCmd	强制或释放高速 APB(APB2)外围设备复位.
RCC_APB1PeriphResetCmd	强制或释放低速 APB(APB1)外围设备复位.
RCC_BackupResetCmd	强制或释放备份域复位
RCC_ClockSecuritySystemCmd	使能或关闭时钟安全系统.
RCC_MCOConfig	选择在 MCO 引脚输出的时钟源
RCC_GetFlagStatus	检查指定的 RCC 标志设置与否.
RCC_ClearFlag	清除 RCC 复位标志.
RCC_GetITStatus	检查指定的 RCC 中断是否发生
RCC_ClearITPendingBit	清除 RCC 中断挂起位.

15.2.1 RCC_DeInit 函数

表 337 描述了 RCC_DeInit 函数.

表 337. RCC_DeInit 函数(1)(2)

函数名	RCC_DeInit
函数原型	void RCC_DeInit(void)
行为描述	复位 RCC 外围设备寄存器到默认复位值.
输入参数	无
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

这个函数不改变 RCC_CR 寄存器的 HSITRIM[4:0]位.

这个函数不复位 RCC_BDCR 和 RCC_CSR 寄存器.

例子:

```
/*复位 RCC 寄存器*/
RCC_DeInit();
```

15.2.2 RCC_HSEConfig 函数

表 338 描述了 RCC_HSEConfig 函数.

表 338.RCC_HSEConfig 函数

函数名	RCC_HSEConfig

函数原型	void RCC_HSEConfig(u32 RCC_HSE)
行为描述	配置外部高速振荡器(HSE)
输入参数	RCC_HSE:HSE 的新状态. **参考章节:*RCC_HSE*** 详细说明了这个参数允许的值.
输出参数	无
返回参数	无
调用前提条件	如果直接或通过 PLL 作为系统时钟,HSE 不能停止..
调用函数	无

RCC_HSE

这个参数配置了 HSE 状态(见表 339).

表 339.RCC_HSE 描述

RCC_HSE	描述
RCC_HSE_OFF	HSE 振荡器关闭
RCC_HSE_ON	HSE 振荡器开启
RCC_HSE_Bypass	伴随外部时钟的 HSE 振荡器

例子:

```
/* 使能 HSE */
RCC_HSEConfig(RCC_HSE_ON);
```

15.2.3 RCC_WaitForHSEStartUp 函数

表 340 描述了 RCC_WaitForHSEStartUp 函数.

表 340. RCC_WaitForHSEStartUp 函数

函数名	RCC_WaitForHSEStartUp
函数原型	ErrorStatus RCC_WaitForHSEStartUp(void)
行为描述	等待 HSE 启动 这个函数等待 HSE 准备好,或超时时间到来
输入参数	无
输出参数	无
返回参数	一个 ErrorStatus 枚举值: SUCCESS:HSE 振荡器稳定并且可以使用. ERROR:HSE 振荡器没有准备好.
调用前提条件	无
调用函数	无

例子:

```
ErrorStatus HSEStartUpStatus;

/* 使能 HSE */

RCC_HSEConfig(RCC_HSE_ON);

/*等待 HSE 准备好,若超时时间到来则退出 */

HSEStartUpStatus = RCC_WaitForHSEStartUp();
```

```
if(HSEStartUpStatus == SUCCESS)
{
/* Add here PLL ans system clock config */
}
else
{
/* Add here some code to deal with this error */
}
```

15.2.4 RCC_AdjustHSICalibrationValue 函数

表 341 描述了 RCC_AdjustHSICalibrationValue 函数.

表 341. RCC_AdjustHSICalibrationValue 函数

函数名	RCC_AdjustHSICalibrationValue
函数原型	void RCC_AdjustHSICalibrationValue(u8 HSICalibrationValue)
行为描述	校正内部高速振荡器(HSI)标度值.
输入参数	HSICalibrationValue:校正标度值 这个参数一定是在 0 到 0x1F 之间的值.
输出参数	无
返回参数	无

调用前提条件	无
调用函数	无

例子:

```
/* 设置 HIS 标度值为 c0x1F(最大值) */

RCC_AdjustHSICalibrationValue(0x1F);
```

15.2.5 RCC_HSICmd 函数

表 342 描述了 RCC_HSICmd 函数.

表 342. RCC_HSICmd 函数

函数名	RCC_HSICmd
函数原型	void RCC_HSICmd(FunctionalState NewState)
行为描述	使能内部高速振荡器(HIS).
输入参数	NewStateHSI 的新状态. 这个参数可以是:ENABLE 或 DISABLE.
输出参数	无
返回参数	无
调用前提条件	如果直接或通过 PLL 作为系统时钟,或如果闪存程序操作在运行,HIS 不能停止.
调用函数	无

例子:

```
/* 使能内部高速振荡器 */

RCC_HSICmd(ENABLE);
```

15.2.6 RCC_PLLConfig 函数

表 343 描述了 RCC_PLLConfig 函数.

表 343. RCC_PLLConfig 函数

函数名	RCC_PLLConfig
函数原型	void RCC_PLLConfig(u32 RCC_PLLSource, u32 RCC_PLLMul)
行为描述	配置 PLL 时钟源和乘法因子.
输入参数 1	RCC_PLLSource:PLL 进入时钟源 参考章节:*RCC_PLLSource* 详细说明了这个参数的允许值.
输入参数 2	RCC_PLLMul:PLL 乘法因子. **参考章节:*RCC_PLLMul*** 详细说明了这个参数的允许值.
输出参数	无
返回参数	无
调用前提条件	这个函数只有在 PLL 关闭时才可以使用.
调用函数	无

RCC_PLLSource

这个参数选择了 PLL 输入时钟源 (见 表 344).

表 344.RCC_PLLSource 定义

RCC_PLLSource	描述
RCC_PLLSource_HSI_Div2	PLL 时钟输入等于 HIS 时钟二分频
RCC_PLLSource_HSE_Div1	PLL 时钟输入等于 HIS 时钟
RCC_PLLSource_HSE_Div2PLL clock entry	PLL 时钟输入等于 HIS 时钟二分频

RCC_PLLMul

这个参数选择了 PLL 乘法因子(见表 345).

表 345.RCC_PLLMul 定义

RCC_PLLMul	描述
RCC_PLLMul_2	PLL 时钟输入值乘 2
RCC_PLLMul_3	PLL 时钟输入值乘 3
RCC_PLLMul_4	PLL 时钟输入值乘 4
RCC_PLLMul_5	PLL 时钟输入值乘 5
RCC_PLLMul_6	PLL 时钟输入值乘 6
RCC_PLLMul_7	PLL 时钟输入值乘 7
RCC_PLLMul_8	PLL 时钟输入值乘 8

RCC_PLLMul_9	PLL 时钟输入值乘 9
RCC_PLLMul_10	PLL 时钟输入值乘 10
RCC_PLLMul_11	PLL 时钟输入值乘 11
RCC_PLLMul_12	PLL 时钟输入值乘 12
RCC_PLLMul_13	PLL 时钟输入值乘 13
RCC_PLLMul_14	PLL 时钟输入值乘 14
RCC_PLLMul_15	PLL 时钟输入值乘 15
RCC_PLLMul_16	PLL 时钟输入值乘 16

注意:必须由软件正确配置 PLL,以产生不超过 72 MHz 的 PLL 输出频率.

例子:

```
/* 使用 HSE(8MHz)作为输入时钟,设置 PLL 时钟输出为 72MHz*/

RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);
```

15.2.7 RCC_PLLCmd 函数

表 346 描述了 RCC_PLLCmd 函数.

表 346. RCC_PLLCmd 函数

函数名	RCC_PLLCmd
函数原型	void RCC_PLLCmd(FunctionalState NewState)
行为描述	使能或关闭 PLL

输入参数	NewState:PLL 的新状态. 这个参数可以是:ENABLE 或 DISABLE
输出参数	无
返回参数	无
调用前提条件	若 PLL 作为系统时钟,则它不能被禁止
调用函数	无

例子:

```
/* 使能 PLL */

RCC_PLLCmd(ENABLE);
```

15.2.8 RCC_SYSCLKConfig 函数

表 347 描述了 RCC_SYSCLKConfig 函数.

表 347. RCC_SYSCLKConfig 函数

函数名	RCC_SYSCLKConfig
函数原型	void RCC_SYSCLKConfig(u32 RCC_SYSCLKSource)
行为描述	配置系统时钟(SYSCLK).
输入参数	RCC_SYSCLKSource:作为系统时钟的时钟源. 参考章节:*RCC_SYSCLKSource* 详细说明了这个参数的允许值.
输出参数	无

返回参数	无
调用前提条件	无
调用函数	无

RCC_SYSCLKSource

这个参数选择了系统时钟源(见表 348).

表 348.RCC_SYSCLKSource 定义

RCC_SYSCLKSource	描述
RCC_SYSCLKSource_HSI	选择 HSI 作为系统时钟.
RCC_SYSCLKSource_HSE	选择 HSE 作为系统时钟.
RCC_SYSCLKSource_PLLCLK	选择 PLL 作为系统时钟.

例子:

/* Select the PLL as system clock source */

RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);

15.2.9 RCC_GetSYSCLKSource 函数

表 349 描述了 RCC_GetSYSCLKSource 函数.

表 349. RCC_GetSYSCLKSource 函数

函数名	RCC_GetSYSCLKSource
函数原型	u8 RCC_GetSYSCLKSource(void)
行为描述	返回用作系统时钟的时钟源
输入参数	无
输出参数	无
返回参数	时钟源作为系统时钟使用.返回值可以是: - 0x00:使用 HSI 作为系统时钟 - 0x04:使用 HSE 作为系统时钟 - 0x08:使用 PLL 作为系统时钟.
调用前提条件	无
调用函数	无

例子:

```
/* 检测 HSE 是否作为系统时钟 */
if(RCC_GetSYSCLKSource() != 0x04)
{
}
else
{
}
```

15.2.10 RCC_HCLKConfig 函数

表 350 描述了 RCC_HCLKConfig 函数.

表 350. RCC_HCLKConfig 函数

函数名	RCC_HCLKConfig
函数原型	void RCC_HCLKConfig(u32 RCC_HCLK)
行为描述	配置 AHB 时钟(HCLK).
输入参数	RCC_HCLK:定义了 AHB 时钟.这个时钟是从系统时钟(SYSCLK)得到的. 参考章节:*RCC_HCLK* 详细说明了这个参数的允许值.
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

RCC_HCLK

RCC_HCLK 配置了 AHB 时钟.表 351 介绍了这个参数使用的值.

表 351.RCC_HCLK 值

RCC_HCLK	描述
RCC_SYSCLK_Div1	AHB 时钟等于 SYSCLK
RCC_SYSCLK_Div2	AHB 时钟等于 SYSCLK/2

RCC_SYSCLK_Div4	AHB 时钟等于 SYSCLK/4
RCC_SYSCLK_Div8	AHB 时钟等于 SYSCLK/8
RCC_SYSCLK_Div16	AHB 时钟等于 SYSCLK/16
RCC_SYSCLK_Div64	AHB 时钟等于 SYSCLK/64
RCC_SYSCLK_Div128	AHB 时钟等于 SYSCLK/128
RCC_SYSCLK_Div256	AHB 时钟等于 SYSCLK/256
RCC_SYSCLK_Div512	AHB 时钟等于 SYSCLK/512

例子:

```
/* 配置 HCLK, 例如 HCLK = SYSCLK */

RCC_HCLKConfig(RCC_SYSCLK_Div1);
```

15.2.11 RCC_PCLK1Config 函数

表 352 描述了 RCC_PCLK1Config 函数.

表 352. RCC_PCLK1Config 函数

函数名	RCC_PCLK1Config
函数原型	void RCC_PCLK1Config(u32 RCC_PCLK1)
行为描述	配置低速 APB 时钟(PCLK1).
输入参数	RCC_PCLK1:定义 APB1 时钟.这个时钟是从 AHB 时钟得到的(HCLK). 参考章节:*RCC_PCLK1* 详细说明了这个参数的允许值.

输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

RCC_PCLK1

RCC_PCLK1 配置了 APB1 时钟.表 353 介绍了这个参数的使用值.

表 353.RCC_PCLK1 值

RCC_PCLK1	描述
RCC_HCLK_Div1	APB1 时钟等于 HCLK
RCC_HCLK_Div2	APB1 时钟等于 HCLK/2
RCC_HCLK_Div4	APB1 时钟等于 HCLK/4
RCC_HCLK_Div8	APB1 时钟等于 HCLK/8
RCC_HCLK_Div16	APB1 时钟等于 HCLK/16

例子:

```
/* 配置 PCLK1,例如 PCLK1 = HCLK/2 */

RCC_PCLK1Config(RCC_HCLK_Div2);
```

15.2.12 RCC_PCLK2Config 函数

表 354 描述了 RCC_PCLK2Config 函数.

表 354. RCC_PCLK2Config 函数

函数名	RCC_PCLK2Config
函数原型	void RCC_PCLK2Config(u32 RCC_PCLK2)
行为描述	配置 APB 时钟(PCLK2).
输入参数	RCC_PCLK2:定义 APB2 时钟.这个时钟是从 AHB 时钟得到的(HCLK). 参考章节:*RCC_PCLK2* 详细说明了这个参数的允许值.
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

RCC_PCLK2

RCC_PCLK2 配置了 APB2 的时钟,表 355 介绍了这个参数的允许值.

表 355. RCC_PCLK2 值

RCC_PCLK2	描述
RCC_HCLK_Div1	APB2 时钟等于 HCLK
RCC_HCLK_Div2	APB2 时钟等于 HCLK/2
RCC_HCLK_Div4	APB2 时钟等于 HCLK/4
RCC_HCLK_Div8	APB2 时钟等于 HCLK/8
RCC_HCLK_Div16	APB2 时钟等于 HCLK/16

例子:

```
/* 配置 PCLK2,例如 PCLK2 = HCLK */

RCC_PCLK2Config(RCC_HCLK_Div1);
```

15.2.13 RCC_ITConfig 函数

表 356 描述了 RCC_ITConfig 函数.

表 356. RCC_ITConfig 函数

函数名	RCC_ITConfig
函数原型	void RCC_ITConfig(u8 RCC_IT, FunctionalState NewState)
行为描述	使能或关闭特定的 RCC 中断.
输入参数 1	RCC_IT:指定要使能或要关闭的 RCC 中断源. **参考章节:*RCC_IT*** 详细说明了这个参数的允许值.
输入参数 2	NewState:指定的 RCC 中断的新状态. 这个参数可以是:ENABLE 或 DISABLE
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

RCC_IT

RCC_IT 使能或关闭 RCC 中断,下表列出了这个参数的一个或组合值.

表 357.RCC_IT 值

RCC_IT	描述
RCC_IT_LSIRDY	LSI 就绪中断
RCC_IT_LSERDY	LSE 就绪中断
RCC_IT_HSIRDY	HIS 就绪中断
RCC_IT_HSERDY	HSE 就绪中断
RCC_IT_PLLRDY	PLL 就绪中断

例子:

/*使能 PLL 就绪中断*/

RCC_ITConfig(RCC_IT_PLLRDY, ENABLE);

15.2.14 RCC_USBCLKConfig 函数

表 358 描述了 RCC_USBCLKConfig 函数.

表 358. RCC_USBCLKConfig 函数

函数名	RCC_USBCLKConfig
函数原型	void RCC_USBCLKConfig(u32 RCC_USBCLKSource)
行为描述	配置 USB 时钟(USBCLK).
输入参数	RCC_USBCLKSource:指定 USB 时钟源,这个时钟来源于

	PLL 输出. 参考章节:*RCC_USBCLKSource*详细说明了这个参数的允许值.
输出参数	无
返回参数	无
调用前提条件	USB 需要 48MHz 时钟来正确执行.用户一定要根据 PLL 乘法因子和 PLL 时钟源频率选择 USB 分割因子来达到 48MHz 的频率.一旦 USB 时钟使能,则 USB 分割因子就不可以更改.
调用函数	无

RCC_USBCLKSource

这个参数选择了 USB 时钟源(见表 359).

表 359.RCC_USBCLKSource 值

RCC_USBCLKSource	描述
RCC_USBCLKSource_PLLCLK_1Div5	USB 时钟源等于选定的 PLL 时钟的 1.5 分频
RCC_USBCLKSource_PLLCLK_Div1	USB 时钟源等于选定的 PLL 时钟

例子:

```
/* PLL 时钟 1.5 分频后用作 USB 时钟源 */

RCC_USBCLKConfig(RCC_USBCLKSource_PLLCLK_1Div5);
```

15.2.15 RCC_ADCCLKConfig 函数

表 360 描述了 RCC_ADCCLKConfig 函数.

表 360.RCC_ADCCLKConfig 函数

函数名	RCC_ADCCLKConfig
函数原型	void RCC_ADCCLKConfig(u32 RCC_ADCCLK)
行为描述	配置 ADC 时钟(ADCCKL).
输入参数	RCC_ADCCLK:定义了 ADC 时钟.这个时钟来自 APB2 时钟(PCLK2). 参考章节:*RCC_ADCCLK* 详细说明了这个参数的允许值.
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

RCC_ADCCLK

RCC_ADCCLK 配置了 ADC 时钟.表 361 介绍了这个参数的使用值.

表 361. RCC_ADCCLK 值

RCC_ADCCLK	描述
RCC_PCLK2_Div2	ADC 时钟等于 PCLK/2
RCC_PCLK2_Div4	ADC 时钟等于 PCLK/4

RCC_PCLK2_Div6	ADC 时钟等于 PCLK/6
RCC_PCLK2_Div8	ADC 时钟等于 PCLK/8

例子:

```
/* 配置 ADCCLK,例如 ADCCLK = PCLK2/2 */

RCC_ADCCLKConfig(RCC_PCLK2_Div2);
```

15.2.16 RCC_LSEConfig 函数

表 362 描述了 RCC_LSEConfig 函数.

表 362. RCC_LSEConfig 函数

函数名	RCC_LSEConfig
函数原型	void RCC_LSEConfig(u32 RCC_LSE)
行为描述	配置外部低速振荡器(LSE).
输入参数	RCC_LSE:LSE 的新状态 **参考章节:*RCC_LSE*** 详细说明了这个参数的允许值.
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

RCC_LSE

这个参数配置了 LSE 的状态(见表 363).

表 363. RCC_LSE 值

RCC_LSE	描述
RCC_LSE_OFF	LSE 振荡器关闭
RCC_LSE_ON	LSE 振荡器开启
RCC_LSE_Bypass	带有外部时钟的 LSE 振荡器

例子:

```
/* 使能 LSE */

RCC_LSEConfig(RCC_LSE_ON);
```

15.2.17 RCC_LSICmd 函数

表 364 描述了 RCC_LSICmd 函数.

表 364. RCC_LSICmd 函数

函数名	RCC_LSICmd
函数原型	void RCC_LSICmd(FunctionalState NewState)
行为描述	使能或关闭内部低速振荡器(LSI).
输入参数	NewState:LSI 的新状态 这个参数的值可以是:ENABLE 或 DISABLE
输出参数	无
返回参数	无

调用前提条件	如果 IWDG 运行,则 LSI 不能被禁止
调用函数	无

例子:

```
/* 使能内部低速振荡器*/

RCC_LSICmd(ENABLE);
```

15.2.18 RCC_RTCCLKConfig 函数

表 365 描述了 RCC_RTCCLKConfig 函数.

表 365. RCC_RTCCLKConfig 函数

函数名	RCC_RTCCLKConfig
函数原型	void RCC_RTCCLKConfig(u32 RCC_RTCCLKSource)
行为描述	配置 RTC 时钟(RTCCLK).
输入参数	RCC_RTCCLKSource:RTC 时钟源. 参考章节:*RCC_RTCCLKSource* 详细说明了这个参数的允许值.
输出参数	无
返回参数	无
调用前提条件	只要选择了 RTC 时钟,RTC 时钟就不能改变直到备份域复位
调用函数	无

RCC_RTCCLKSource

这个参数选择了 RTC 时钟源(见表 366).

表 366. RCC_RTCCLKSource 值

RCC_RTCCLKSource	描述
RCC_RTCCLKSource_LSE	选择 LSE 作为 RTC 时钟.
RCC_RTCCLKSource_LSI	选择 LSI 作为 RTC 时钟
RCC_RTCCLKSource_HSE_Div128	选择 HSE 时钟的 128 分频作为 RTC 时钟

例子:

```
/* 选择 LSE 作为 RTC 时钟源*/

RCC_RTCCLKConfig(RCC_RTCCLKSource_LSE);
```

15.2.19 RCC_RTCCLKCmd 函数

表 367 描述了 RCC_RTCCLKCmd 函数.

表 367. RCC_RTCCLKCmd 函数

函数名	RCC_RTCCLKCmd
函数原型	void RCC_RTCCLKCmd(FunctionalState NewState)
行为描述	使能或关闭 RTC 时钟.
输入参数	NewState:.RTC 时钟的新状态.

	这个参数可以是:ENABLE 或 DISABLE
输出参数	无
返回参数	无
调用前提条件	仅当使用 RCC_RTCCLKConfig 函数选择 RTC 时钟后,才使用这个函数.
调用函数	无

例子:

```
/* 使能 RTC 时钟*/
RCC_RTCCLKCmd(ENABLE);
```

15.2.20 RCC_GetClocksFreq 函数

表 368 描述了 RCC_GetClocksFreq 函数.

表 368. RCC_GetClocksFreq 函数

函数名	RCC_GetClocksFreq
函数原型	void RCC_GetClocksFreq(RCC_ClocksTypeDef* RCC_Clocks)
行为描述	返回片上时钟的不同频率
输入参数	RCC_Clocks:.指向包括时钟频率的 RCC_ClocksTypeDef 结构的指针 参考章节:*RCC_ClocksTypeDef 结构* 详细说明了这个参数

	允许的值.
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

RCC_ClocksTypeDef 结构

RCC_ClocksTypeDef 结构定义在 *stm32f10x_rcc.h* 文件中:

```
typedef struct
{
u32 SYSCLK_Frequency;
u32 HCLK_Frequency;
u32 PCLK1_Frequency;
u32 PCLK2_Frequency;
u32 ADCCLK_Frequency;
}RCC_ClocksTypeDef;
```

SYSCLK_Frequency

这个成员返回 SYSCLK 时钟频率.单位 Hz.

HCLK_Frequency

这个成员返回 HCLK 时钟频率.单位 Hz.

PCLK1_Frequency

这个成员返回 PCLK1 时钟频率.单位 Hz.

PCLK2_Frequency

这个成员返回 PCLK2 时钟频率.单位 Hz.

ADCCLK_Frequency

这个成员返回 ADCCLK 时钟频率.单位 Hz.

例子:

/*获得板上时钟的不同频率 */

RCC_ClocksTypeDef RCC_Clocks;

RCC_GetClocksFreq(&RCC_Clocks);

15.2.21 RCC_AHBPeriphClockCmd 函数

表 369 描述了 RCC_AHBPeriphClockCmd 函数.

表 369. RCC_AHBPeriphClockCmd 函数

函数名	RCC_AHBPeriphClockCmd
函数原型	void RCC_AHBPeriphClockCmd(u32 RCC_AHBPeriph, FunctionalState NewState) RCC_Clocks)
行为描述	使能或关闭 AHB 外围设备时钟.

输入参数 1	RCC_AHBPeriph:用于门控时钟的 AHB 外围设备 参考章节:*RCC_AHBPeriph 结构* 详细说明了这个参数允许的值.
输入参数 2	NewState:指定的外围设备时钟的新状态. 这个参数可以是:ENALBLE 或 DISABLE.
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

RCC_AHBPeriph

这个参数选择了门控时钟的 AHB 外围设备.下表列出了这个参数的一个或组合值.

表 370.RCC_AHBPeriph 值(1)

RCC_AHBPeriph	描述
RCC_AHBPeriph_DMA	DMA 时钟
RCC_AHBPeriph_SRAM	SRAM 时钟
RCC_AHBPeriph_FLITF	FLITF 时钟

SRAM 和 FLITF 时钟只有在睡眠的模式下可以无效.

例子:

/* 使能 DMA clock */

RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA);

15.2.22 RCC_APB2PeriphClockCmd 函数

表 371 描述了 RCC_APB2PeriphClockCmd 函数.

表 371. RCC_APB2PeriphClockCmd 函数

函数名	RCC_APB2PeriphClockCmd
函数原型	void RCC_APB2PeriphClockCmd(u32 RCC_APB2Periph, FunctionalState NewState)
行为描述	使能或关闭高速 APB(APB2)外围设备时钟.
输入参数 1	RCC_AHB2Periph:用于门控时钟的 AHB2 外围设备 涉及章节:*RCC_AHB2Periph 结构* 详细说明了这个参数允许的值.
输入参数 2	NewState:专用外围设备时钟的新状态. 这个参数可以是:ENALBLE 或 DISABLE.
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

RCC_APB2Periph

这个参数选择了用来门控时钟的 APB2 外围设备.下表介绍了这个参数的一个或组合值.

表 372.RCC_APB2Periph 值

RCC_APB2Periph	描述
RCC_APB2Periph_AFIO	交替功能 I/O 时钟
RCC_APB2Periph_GPIOA	IO 端口 A 时钟
RCC_APB2Periph_GPIOB	IO 端口 B 时钟
RCC_APB2Periph_GPIOC	IO 端口 C 时钟
RCC_APB2Periph_GPIOD	IO 端口 D 时钟
RCC_APB2Periph_GPIOE	IO 端口 E 时钟
RCC_APB2Periph_ADC1	ADC 1 接口时钟
RCC_APB2Periph_ADC2	ADC 2 接口时钟
RCC_APB2Periph_TIM1	TIM1 时钟
RCC_APB2Periph_SPI1	SPI1 时钟
RCC_APB2Periph_USART1	USART1 时钟
RCC_APB2Periph_ALL	所有 APB2 外围设备时钟

例子:

```
/* 使能 GPIOA, GPIOB 和 SPI1 时钟*/

RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB |

RCC_APB2Periph_SPI1, ENABLE);
```

15.2.23 RCC_APB1PeriphClockCmd 函数

表 373 描述了 RCC_APB1PeriphClockCmd 函数.

表 373. RCC_APB1PeriphClockCmd 函数

函数名	RCC_APB1PeriphClockCmd
函数原型	void RCC_APB1PeriphClockCmd(u32 RCC_APB1Periph, FunctionalState NewState)
行为描述	使能或关闭低速 APB(APB1)外围设备时钟
输入参数 1	RCC_APB1Periph: 用于门控时钟的 APB1 外围设备 涉及章节:*RCC_AHB1Periph 结构* 详细说明了这个参数允许的值.
输入参数 2	NewState:专用外围设备时钟的新状态. 这个参数可以是:ENALBLE 或 DISABLE.
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

RCC_APB1Periph

这个参数选择了用于门控时钟的 APB1 外围设备.下表介绍了这个参数的一个或组合值.

表 374.RCC_APB1Periph 值

RCC_APB1Periph	描述
RCC_APB1Periph_TIM2	TIM2 时钟
RCC_APB1Periph_TIM3	TIM3 时钟
RCC_APB1Periph_TIM4	TIM4 时钟
RCC_APB1Periph_WWDG	窗口看门狗时钟
RCC_APB1Periph_SPI2	SPI2 时钟
RCC_APB1Periph_USART2	USART2 时钟
RCC_APB1Periph_USART3	USART3 时钟
RCC_APB1Periph_I2C1	I2C1 时钟
RCC_APB1Periph_I2C2	I2C2 时钟
RCC_APB1Periph_USB	USB 时钟
RCC_APB1Periph_CAN	CAN 时钟
RCC_APB1Periph_BKP	备份接口时钟
RCC_APB1Periph_PWR	电源控制接口时钟
RCC_APB1Periph_ALL	所有 APB1 外围设备时钟

例子:

```
/* 使能 BKP 和 PWR 使能*/
RCC_APB1PeriphClockCmd(RCC_APB1Periph_BKP | RCC_APB1Periph_PWR,
ENABLE);
```

15.2.24 RCC_APB2PeriphResetCmd 函数

表 375 描述了 RCC_APB2PeriphResetCmd 函数.

表 375. RCC_APB2PeriphResetCmd 函数

函数名	RCC_APB2PeriphResetCmd
函数原型	void RCC_APB2PeriphResetCmd(u32 RCC_APB2Periph, FunctionalState NewState)
行为描述	强迫或释放高速 APB(APB2)外围设备复位.
输入参数 1	RCC_APB2Periph:复位的 APB2 外围设备. 参考章节:*RCC_AHB2Periph 结构* 详细说明了这个参数允许的值.
输入参数 2	NewState:指定外围设备复位的新状态. 这个参数可以是:ENALBLE 或 DISABLE.
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

例子:

```
/* 输入要复位的 SPI 外设*/

RCC_APB2PeriphResetCmd(RCC_APB2Periph_SPI1, ENABLE);
```

```
/* 将 SPI 外设从复位状态退出*/

RCC_APB2PeriphResetCmd(RCC_APB2Periph_SPI1, DISABLE);
```

15.2.25 RCC_APB1PeriphResetCmd 函数

表 376 描述了 RCC_APB1PeriphResetCmd 函数.

表 376. RCC_APB1PeriphResetCmd 函数

函数名	RCC_APB1PeriphResetCmd
函数原型	void RCC_APB1PeriphResetCmd(u32 RCC_APB1Periph, FunctionalState NewState)
行为描述	强制或释放低速 APB(APB1)外围设备复位.
输入参数 1	RCC_APB1Periph:指定要复位的 APB1 外围设备. 参考章节:*RCC_AHB1Periph 结构* 详细说明了这个参数允许的值.
输入参数 2	NewState:指定外围设备复位的新状态. 这个参数可以是:ENALBLE 或 DISABLE.
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

例子:

```
/* 输入要复位的 SPI2 外设 */

RCC_APB1PeriphResetCmd(RCC_APB1Periph_SPI2, ENABLE);

/*将 SPI2 外设从复位状态退出*/

RCC_APB1PeriphResetCmd(RCC_APB1Periph_SPI2, DISABLE);
```

15.2.26 RCC_BackupResetCmd 函数

表 377 描述了 RCC_BackupResetCmd 函数.

表 377. RCC_BackupResetCmd 函数

函数名	RCC_BackupResetCmd
函数原型	void RCC_BackupResetCmd(FunctionalState NewState)
行为描述	强制或释放备份域复位
输入参数	NewState:备份域复位的新状态. 这个参数可以是:ENABLE 或 DISABLE
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

例子:

```
/* 复位整个备份域*/

RCC_BackupResetCmd(ENABLE);
```

15.2.27 RCC_ClockSecuritySystemCmd 函数

表 378 描述了 RCC_ClockSecuritySystemCmd 函数.

表 378. RCC_ClockSecuritySystemCmd 函数

函数名	RCC_ClockSecuritySystemCmd
函数原型	void RCC_ClockSecuritySystemCmd(FunctionalState NewState)
行为描述	使能或关闭时钟安全系统.
输入参数	NewState:时钟安全系统的新状态. 这个参数可以是:ENABLE 或 DISABLE
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

例子:

```
/* 使能时钟安全系统 */

RCC_ClockSecuritySystemCmd(ENABLE);
```

15.2.28 RCC_MCOConfig 函数

表 379 描述了 RCC_MCOConfig 函数.

表 379. RCC_MCOConfig 函数

函数名	RCC_MCOConfig
函数原型	void RCC_MCOConfig(u8 RCC_MCO)
行为描述	选择 MCO 引脚上的输出时钟源
输入参数	RCC_MCO:详细说明要输出的时钟源. **参考章节:*RCC_MCO*** 详细说明了这个参数允许的值.
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

RCC_MCO

RCC_MCO 选择了 MCO 引脚上的输出时钟源.表 380 介绍了这个参数使用的值.

表 380.RCC_MCO 值

RCC_MCO	描述
RCC_MCO_NoClock	没有选择时钟
RCC_MCO_SYSCLK	选择系统时钟
RCC_MCO_HSI	选择 HIS 振荡时钟

RCC_MCO_HSE	选择 HSE 振荡时钟
RCC_MCO_PLLCLK_Div2	选择 PLL 时钟两分频

警告:选择系统时钟输出到 MCO 时,确保最大 I/O 速率不超过 50MHz.

例子:

```
/*MCO 引脚上被 2 分频的 PLL 输出时钟 */
RCC_MCOConfig(RCC_MCO_PLLCLK_Div2);
```

15.2.29 RCC_GetFlagStatus 函数

表 381 描述了 RCC_GetFlagStatus 函数.

表 381. RCC_GetFlagStatus 函数

函数名	RCC_GetFlagStatus
函数原型	FlagStatus RCC_GetFlagStatus(u8 RCC_FLAG)
行为描述	检查指定的 RCC 标志设置与否.
输入参数	RCC_FLAG:要检查的标志. **参考章节:*RCC_FLAG*** 详细说明了这个参数允许的值.
输出参数	无
返回参数	RCC_FLAG 的新状态(SET 或 RESET).

调用前提条件	无
调用函数	无

RCC_FLAG

表 382 列出了能被 RCC_GetFlagStatus 函数所检查的 RCC 标志.

表 382.RCC_FLAG 值

RCC_FLAG	描述
RCC_FLAG_HSIRDY	HIS 振荡时钟就绪
RCC_FLAG_HSERDY	HSE 振荡时钟就绪
RCC_FLAG_PLLRDY	PLL 时钟就绪
RCC_FLAG_LSERDY	LSE 振荡时钟就绪
RCC_FLAG_LSIRDY	LSI 振荡时钟就绪
RCC_FLAG_PINRST	引脚复位
RCC_FLAG_PORRST	POR/PDR 复位
RCC_FLAG_SFTRST	软件复位
RCC_FLAG_IWDGRST	独立看门狗复位
RCC_FLAG_WWDGRST	窗口看门狗复位
RCC_FLAG_LPWRRST	低电量复位

例子:

```
/* 检查 PLL 时钟是否就绪*/

FlagStatus Status;
```

```
Status = RCC_GetFlagStatus(RCC_FLAG_PLLRDY);
if(Status == RESET)
{
...
}
else
{
...
}
```

15.2.30 RCC_ClearFlag 函数

表 383 描述了 RCC_ClearFlag 函数.

表 383. RCC_ClearFlag 函数

函数名	RCC_ClearFlag
函数原型	void RCC_ClearFlag(void)
行为描述	清除 RCC 复位标志. 复位标志为: RCC_FLAG_PINRST, RCC_FLAG_PORRST, RCC_FLAG_SFTRST, RCC_FLAG_IWDGRST, RCC_FLAG_WWDGRST,

	RCC_FLAG_LPWRRST
输入参数	无
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

例子:

```
/* 清除复位标志位 */

RCC_ClearFlag();
```

15.2.31 RCC_GetITStatus 函数

表 384 描述了 RCC_GetITStatus 函数.

表 384. RCC_GetITStatus 函数

函数名	RCC_GetITStatus
函数原型	ITStatus RCC_GetITStatus(u8 RCC_IT)
行为描述	检查指定的 RCC 中断是否发生
输入参数	RCC_IT:检查 RCC 中断源. **参考章节:*RCC_IT*** 详细说明了这个参数允许的值.
输出参数	无
返回参数	RCC_IT 的新状态(SET or RESET).

调用前提条件	无
调用函数	无

RCC_IT

RCC_IT 使能或关闭 RCC 中断.下表列出了这个参数的一个或组合值:

表 385.RCC_IT 值

RCC_IT	描述
RCC_IT_LSIRDY	LSI 就绪中断
RCC_IT_LSERDY	LSE 就绪中断
RCC_IT_HSIRDY	HIS 就绪中断
RCC_IT_HSERDY	HSE 就绪中断
RCC_IT_PLLRDY	PLL 就绪中断
RCC_IT_CSS	时钟安全系统中断

例子:

```
/* 检查 PLL 就绪中断是否发生 */

ITStatus Status;

Status = RCC_GetITStatus(RCC_IT_PLLRDY);

if(Status == RESET)

{

...

}
```

```
else
{
...
}
```

15.2.32 RCC_ClearITPendingBit 函数

表 368 描述了 RCC_ClearITPendingBit 函数.

表 368. RCC_ClearITPendingBit 函数

函数名	RCC_ClearITPendingBit
函数原型	Void RCC_ClearITPendingBit(u8 RCC_IT)
行为描述	清除 RCC 中断挂起位.
输入参数	RCC_IT:指定要清除的中断挂起位.. **参考章节:*RCC_IT*** 详细说明了这个参数允许的值.
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

RCC_IT

RCC_IT 使能或关闭 RCC 中断.下表列出了这个参数的一个或组合值:

表 387.RCC_IT 值

RCC_IT	描述
RCC_IT_LSIRDY	LSI 就绪中断
RCC_IT_LSERDY	LSE 就绪中断
RCC_IT_HSIRDY	HIS 就绪中断
RCC_IT_HSERDY	HSE 就绪中断
RCC_IT_PLLRDY	PLL 就绪中断
RCC_IT_CSS	时钟安全系统中断

例子:

```
/* 清除 PLL 的就绪中断挂起位*/

RCC_ClearITPendingBit(RCC_IT_PLLRDY);
```

16 实时时钟(RTC)

RTC 通过合适的软件提供一套连续的运行计数器来提供时钟日历功能.可以通过写计数器值来设置当前的时间/日期.

16.1 节:*RTC 寄存器结构*描述了 RTC 固件库中的数据结构.16.2 节:*固件库函数*介绍了固件库函数.

16.1 RTC 寄存器结构

RTC 寄存器结构,RTC_TypeDef,定义在 stm32f10x_map.h 文件中,

如:

```
typedef struct
{
vu16 CRH;
u16 RESERVED1;
vu16 CRL;
u16 RESERVED2;
vu16 PRLH;
u16 RESERVED3;
vu16 PRLL;
u16 RESERVED4;
vu16 DIVH;
```

```
u16 RESERVED5;

vu16 DIVL;

u16 RESERVED6;

vu16 CNTH;

u16 RESERVED7;

vu16 CNTL;

u16 RESERVED8;

vu16 ALRH;

u16 RESERVED9;

vu16 ALRL;

u16 RESERVED10;

} RTC_TypeDef;
```

表 388 给出了 RTC 寄存器列表.

表 388.RTC 寄存器

寄存器	描述
CRH	控制寄存器高
CRL	控制寄存器低
PRLH	预分频数加载寄存器高
PRLL	预分频数加载寄存器低
DIVH	分割数寄存器高

DIVL	分割数寄存器低
CNTH	计数寄存器高
CNTL	计数寄存器低
ALRH	警报寄存器高
ALRL	警报寄存器低

RTC 外围设备声明在 stm3210x_map.h 文件中:

...

```
#define PERIPH_BASE ((u32)0x40000000)

#define APB1PERIPH_BASE PERIPH_BASE

#define APB2PERIPH_BASE (PERIPH_BASE + 0x10000)

#define AHBPERIPH_BASE (PERIPH_BASE + 0x20000)

...

#define RTC_BASE (APB1PERIPH_BASE + 0x2800)

#ifndef DEBUG

...

#ifdef _RTC

#define RTC ((RTC_TypeDef *) RTC_BASE)

#endif /*_RTC */

...

#else /* DEBUG */

...
```

```
#ifdef _RTC

EXT RTC_TypeDef *RTC;

#endif /*_RTC */

...

#endif
```

在调试模式下,通过 stm32f10x_lib.c 文件对 RTC 指针初始化中:

```
#ifdef _RTC

RTC = (RTC_TypeDef *) RTC_BASE;

#endif /*_RTC */
```

要访问 RTC 寄存器,_RTC 一定要在 stm32f10x_conf.h 文件中定义如下.

如:

```
#define _RTC
```

16.2 固件库函数

表 389 给出了 RTC 固件库函数列表.

表 389.RTC 固件库函数

函数名	描述
RTC_ITConfig	使能或关闭指定的 RTC 中断
RTC_EnterConfigMode	进入 RTC 配置模式

RTC_ExitConfigMode	跳出 RTC 配置模式
RTC_GetCounter	获得 RTC 计数器值
RTC_SetCounter	设置 RTC 计数器值
RTC_GetPrescaler	获得 RTC 预分频数值
RTC_SetPrescaler	设置 RTC 预分频数值
RTC_SetAlarm	设置 RTC 警报值
RTC_GetDivider	获得 RTC 分割值
RTC_WaitForLastTask	等待 RTC 寄存器写操作完成
RTC_WaitForSynchro	等待 RTC 寄存器(RTC_CNT,RTC_ALR 和 RTC_PRL)与 RTC APB 时钟同步
RTC_GetFlagStatus	检查指定的 RTC 标志设置与否.
RTC_ClearFlag	清除 RTC 挂起标志位
RTC_GetITStatus	检查指定的 RTC 中断发生与否.
RTC_ClearITPendingBit	清除 RTC 中断挂起位.

16.2.1 RTC_ITConfig 函数

表 390 描述了 RTC_ITConfig 函数

表 390.RTC_ITConfig 函数

函数名	RTC_ITConfig
函数原型	void RTC_ITConfig(u16 RTC_IT, FunctionalState NewState)
行为描述	使能或关闭指定的 RTC 中断
输入参数 1	RTC_IT:要使能或关闭的 RTC 中断源. **参考章节:*RCC_IT*** 详细说明了这个参数允许的值.
输入参数 2	NewState:指定 RTC 中断的新状态. 这个参数可以是:ENALBLE 或 DISABLE.
输出参数	无
返回参数	无
调用前提条件	在使用这个函数之前，必需调用 RTC_WaitForLastTask()函数(等待直到 RTOFF 标志置位)
调用函数	无

RTC_IT

RTC_IT 使能 RTC 中断.下表列出了这个参数的一个或组合值.

表 391.RTC_IT 值

RTC_IT	描述
RTC_IT_OW	溢出中断使能
RTC_IT_ALR	警报中断使能

RTC_IT_SEC	第二次中断使能

例子:

```
/* 等待直到 RTC 寄存器上的最后一个写操作完成*/
RTC_WaitForLastTask();
/* 警报中断使能 */
RTC_ITConfig(RTC_IT_ALR, ENABLE);
```

16.2.2 RTC_EnterConfigMode 函数

表 392 描述了 RTC_EnterConfigMode 函数.

表 392. RTC_EnterConfigMode 函数

函数名	RTC_EnterConfigMode
函数原型	void RTC_EnterConfigMode(void)
行为描述	进入 RTC 配置模式
输入参数	无
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

例子:

```
/* 进入配置模式*/
```

RTC_EnterConfigMode();

16.2.3 RTC_ExitConfigMode 函数

表 393 描述了 RTC_ExitConfigMode 函数.

表 393. RTC_ExitConfigMode 函数

函数名	RTC_ExitConfigMode
函数原型	void RTC_ExitConfigMode(void)
行为描述	跳出 RTC 配置模式
输入参数	无
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

例子:

/*退出配置模式 */

RTC_ExitConfigMode();

16.2.4 RTC_GetCouter 函数

表 394 描述了 RTC_GetCouter 函数.

表 394. RTC_GetCouter 函数

函数名	RTC_GetCounter
函数原型	u32 RTC_GetCounter(void)
行为描述	获得 RTC 计数器值
输入参数	无
输出参数	无
返回参数	RTC 计数值
调用前提条件	无
调用函数	无

例子:

```
/* 获得计数器的值*/

u32 RTCCounterValue;

RTCCounterValue = RTC_GetCounter();
```

16.2.5 RTC_SetCounter 函数

表 395 描述了 RTC_SetCounter 函数.

表 395. RTC_SetCounter 函数

函数名	RTC_SetCounter
函数原型	void RTC_SetCounter(u32 CounterValue)
行为描述	设置 RTC 计数器值
输入参数	CounterValue:RTC 计数新值

输出参数	无
返回参数	无
调用前提条件	在调用这个函数之前,要调用 RTC_WaitForLastTask()函数(等待直到 RTOFF 标志置位)
调用函数	RTC_EnterConfigMode() RTC_ExitConfigMode()

例子:

```
/* 等待直到 RTC 寄存器上的最后一个写操作完成*/
RTC_WaitForLastTask();
/* 设置计数器的值为 0xFFFF5555 */
RTC_SetCounter(0xFFFF5555);
```

16.2.6 RTC_GetPrescaler 函数

表 396 描述了 RTC_GetPrescaler 函数.

表 396. RTC_GetPrescaler 函数

函数名	RTC_GetPrescaler
函数原型	u32 RTC_GetPrescaler(void)
行为描述	获得 RTC 预分频数值
输入参数	无

输出参数	无
返回参数	RTC 预分频数值
调用前提条件	无
调用函数	无

例子:

```
/* 获得当前 RTC 预分频数的值 */
u32 RTCPrescalerValue;
RTCPrescalerValue = RTC_GetPrescaler();
```

16.2.7 RTC_SetPrescaler 函数

表 397 描述了 RTC_SetPrescaler 函数.

表 397. RTC_SetPrescaler 函数

函数名	RTC_SetPrescaler
函数原型	void RTC_SetPrescaler(u32 PrescalerValue)
行为描述	设置 RTC 预分频数值
输入参数	PrescalerValue:RTC 预分频数新值.
输出参数	无
返回参数	无
调用前提条件	在使用这个函数之前，需要调用

	RTC_WaitForLastTask()函数(等待直到 RTOFF 标志置位)
调用函数	RTC_EnterConfigMode() RTC_ExitConfigMode()

例子:

```
/* 等待直到 RTC 寄存器上最后一个写操作完成*/
RTC_WaitForLastTask();
/* 设置预分频数为 0x7A12 */
RTC_SetPrescaler(0x7A12);
```

16.2.8 RTC_SetAlarm 函数

表 398 描述了 RTC_SetAlarm 函数.

表 398. RTC_SetAlarm 函数

函数名	RTC_SetAlarm
函数原型	void RTC_SetAlarm(u32 AlarmValue)
行为描述	设置 RTC 警报值
输入参数	AlarmValue:RTC 警报新值.
输出参数	无
返回参数	无
调用前提条件	在使用这个函数之前,需要调用 RTC_WaitForLastTask()

	函数(等待直到 RTOFF 标志置位)
调用函数	RTC_EnterConfigMode() RTC_ExitConfigMode()

例子:

```
/*等待直到 RTC 寄存器上最后一个写操作完成*/
RTC_WaitForLastTask();
/* 设置警报值为 0xFFFFFFFA */
RTC_SetAlarm(0xFFFFFFFA);
```

16.2.9 RTC_GetDivider 函数

表 399 描述了 RTC_GetDivider 函数.

表 399. RTC_GetDivider 函数

函数名	RTC_GetDivider
函数原型	u32 RTC_GetDivider(void)
行为描述	获得 RTC 分割值
输入参数	无
输出参数	无
返回参数	RTC 分割值
调用前提条件	无
调用函数	无

例子:

```
/*获得当前的 RTC 分割值*/
u32 RTCDividerValue;
RTCDividerValue = RTC_GetDivider();
```

16.2.10 RTC_WaitForLastTask 函数

表 400 描述了 RTC_WaitForLastTask 函数.

表 400. RTC_WaitForLastTask 函数

函数名	RTC_WaitForLastTask
函数原型	void RTC_WaitForLastTask(void)
行为描述	等待直到 RTC 寄存写操作完成 这个函数一定要在任何对 RTC 寄存器写操作之前调用.
输入参数	无
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

例子:

```
/*等待直到 RTC 寄存器上最后一个写操作完成*/
RTC_WaitForLastTask();
```

/*设置警报值为 0x10 */

RTC_SetAlarm(0x10);

16.2.11 RTC_WaitForSynchro 函数

表 401 描述了 RTC_WaitForSynchro 函数.

表 401. RTC_WaitForSynchro 函数

函数名	RTC_WaitForSynchro
函数原型	void RTC_WaitForSynchro(void)
行为描述	等待 RTC 寄存器(RTC_CNT,RTC_ALR 和 RTC_PRL)与 RTC APB 时钟同步 这个函数一定要在任何对 APB 复位或 APB 时钟停止后的写操作之前调用.
输入参数	无
输出参数	无
返回参数	无
调用前提条件	无
调用函数	无

例子:

u32 RTCPrescalerValue;

/* 等待直到 RTC 寄存器和 RTC APB 时钟同步 */

```
RTC_WaitForSynchro();

/* 获得当前 RTC 预分频数的值*/

RTCPrescalerValue = RTC_GetPrescaler();
```

16.2.12 RTC_GetFlagStatus 函数

表 402 描述了 RTC_GetFlagStatus 函数.

表 402. RTC_GetFlagStatus 函数

函数名	RTC_GetFlagStatus
函数原型	FlagStatus RTC_GetFlagStatus(u16 RTC_FLAG)
行为描述	检查指定的 RTC 标志设置与否.
输入参数	RTC_FLAG:指定要检查的标志. 参考章节:*RTC_FLAG* 详细说明了这个参数允许的值
输出参数	无
返回参数	RTC_FLAG 的新状态(SET 或 RESET).
调用前提条件	无
调用函数	无

RTC_FLAG

表 403 列出了通过调用 RTC_GetFlagStatus 函数返回的 RTC 标志.

表 403.RTC_FLAG 值

RTC_FLAG	描述
RTC_FLAG_RTOFF	RTC 操作关闭标志
RTC_FLAG_RSF	寄存器同步标志
RTC_FLAG_OW	溢出中断标志
RTC_FLAG_ALR	警报中断标志
RTC_FLAG_SEC	第二中断标志

例子:

```
/* 取得 RTC 溢出中断状态 */

FlagStatus OverrunFlagStatus;

OverrunFlagStatus = RTC_GetFlagStatus(RTC_Flag_OW);
```

16.2.13 RTC_ClearFlag 函数

表 404 描述了 RTC_ClearFlag 函数.

表 404. RTC_ClearFlag 函数

函数名	RTC_ClearFlag
函数原型	void RTC_ClearFlag(u16 RTC_FLAG)
行为描述	清除 RTC 挂起标志
输入参数	RTC_FLAG:将被清除的标志 参考章节:*RTC_FLAG* 详细说明了这个参数允许的值 RTC_FLAG_RTOFF 不能由软件清除,RTC_FLAG_RSF 只

	有在 SPB 复位或 APB 时钟停止之后清除.
输出参数	无
返回参数	无
调用前提条件	在使用这个函数之前,调用 RTC_WaitForLastTask()函数(等待直到 RTOFF 标志置位)
调用函数	无

例子:

```
/* 等待 RTC 寄存器上的最后一个写操作完成*/
RTC_WaitForLastTask();
/* 清除 RTC 溢出标志*/
RTC_ClearFlag(RTC_FLAG_OW);
```

16.2.14 RTC_GetITStatus 函数

表 405 描述了 RTC_GetITStatus 函数.

表 405. RTC_GetITStatus 函数

函数名	RTC_GetITStatus
函数原型	ITStatus RTC_GetITStatus(u16 RTC_IT)
行为描述	检查指定的 RTC 中断发生与否.
输入参数	RTC_IT:要检查的 RTC 中断源. **参考章节:*RTC_IT*** 详细说明了这个参数允许的值

输出参数	无
返回参数	RTC_IT 的新状态(SET 或 RESET)
调用前提条件	无
调用函数	无

例子:

```
/*取得 RTC 第二次中断的状态*/

ITStatus SecondITStatus;

SecondITStatus = RTC_GetITStatus(RTC_IT_SEC);
```

16.2.15 RTC_ClearITPendingBit 函数

表 406 描述了 RTC_ClearITPendingBit 函数.

表 406. RTC_ClearITPendingBit 函数

函数名	RTC_ClearITPendingBit
函数原型	void RTC_ClearITPendingBit(u16 RTC_IT)
行为描述	清除 RTC 中断挂起位.
输入参数	RTC_IT:将要清除的中断挂起位. **参考章节:*RTC_IT*** 详细说明了这个参数允许的值
输出参数	无
返回参数	无
调用前提条件	在使用这个函数之前,调用 RTC_WaitForLastTask()函数

	(等待直到 RTOFF 标志置位).
调用函数	无

例子:

```
/* 等待直到 RTC 寄存器上的最后一次写操作完成 */
RTC_WaitForLastTask();
/* 清除 RTC 第二次中断*/
RTC_ClearITPendingBit(RTC_IT_SEC);
```

17 串行外设接口(SPI)

串行外设接口(SPI)允许和外部设备进行同步通信。接口可以配置为用主模式或从模式操作。

17.1 SPI 寄存器结构 描述了在 SPI 固件库中使用的数据结构。*17.2 固件库函数* 介绍了固件库函数。

17.1 SPI 寄存器结构

SPI 寄存器结构，SPI_TypeDef，在文件 stm32f10x_map.h 中定义如下：

```
typedef struct
{
vu16 CR1;
u16 RESERVED0;
vu16 CR2;
```

```
u16 RESERVED1;

vu16 SR;

u16 RESERVED2;

vu16 DR;

u16 RESERVED3;

vu16 CRCPR;

u16 RESERVED4;

vu16 RXCRCR;

u16 RESERVED5;

vu16 TXCRCR;

u16 RESERVED6;

} SPI_TypeDef;
```

表 407 给出了 SPI 的寄存器列表

表 407. SPI 寄存器

寄存器	描述
CR1	SPI 控制寄存器 1
CR2	SPI 控制寄存器 2
SR	SPI 状态寄存器
DR	SPI 数据寄存器
CRCPR	SPI 循环冗余校验多项式寄存器

RxCRCR	SPI 接收 CRC 寄存器
TxCRCR	SPI 发送 CRC 寄存器

这两个 SPI 外设在文件 *stm32f10x_map.h* 中声明如下：

```
...
#define PERIPH_BASE ((u32)0x40000000)
#define APB1PERIPH_BASE PERIPH_BASE
#define APB2PERIPH_BASE (PERIPH_BASE + 0x10000)
#define AHBPERIPH_BASE (PERIPH_BASE + 0x20000)
....
#define SPI1_BASE (APB2PERIPH_BASE + 0x3000)
#define SPI2_BASE (APB1PERIPH_BASE + 0x3800)
....
#ifndef DEBUG
...
#ifdef _SPI1
#define SPI1 ((SPI_TypeDef *) SPI1_BASE)
#endif /*_SPI1 */
#ifdef _SPI2
#define SPI2 ((SPI_TypeDef *) SPI2_BASE)
#endif /*_SPI2 */
```

```
...
#else /* DEBUG */
...
#ifdef _SPI1
EXT SPI_TypeDef *SPI1;
#endif /*_SPI1 */
#ifdef _SPI2
EXT SPI_TypeDef *SPI2;
#endif /*_SPI2 */
...
#endif
```

当使用 Debug，指针_SPI1 和_SPI2 在文件 *stm32f10x_lib.c* 中被初始化：

```
...
#ifdef _SPI1
SPI1 = (SPI_TypeDef *) SPI1_BASE;
#endif /*_SPI1 */
#ifdef _SPI2
SPI2 = (SPI_TypeDef *) SPI2_BASE;
#endif /*_SPI2 */
...
```

要访问 SPI 寄存器，_SPI、_SPI1 和_SPI2 必须在文件 *stm32f10x_conf.h* 中定义如下：

...

#define _SPI

#define _SPI1

#define _SPI2

...

17.2 固件库函数

表 408 列出了 SPI 库中的各种函数。

表 408. SPI 固件库函数

函数名	描述
SPI_DeInit	将 SPIx 外设寄存器重置为他们的缺省值
SPI_Init	根据 SPI_InitStruct.中的指定参数初始化 SPIx 外设
SPI_StructInit	使用缺省值填充 SPI_InitStruct 的成员
SPI_Cmd	使能或取消特定的 SPI 外设
SPI_ITConfig	使能或取消特定的 SPI 中断
SPI_DMACmd	使能或取消某个 SPI 的 DMA 接口
SPI_SendData	通过某个 SPI 外设传输数据
SPI_ReceiveData	通过 SPIx 外设返回最新接收到的数据
SPI_NSSInternalSoftwareC	为特定的 SPI 接口用软件配置 NSS 针脚

onfig	
SPI_SSOutputCmd	为选定的 SPI 接口使能或取消 SS 输出
SPI_DataSizeConfig	为选定的 SPI 接口配置数据大小
SPI_TransmitCRC	传输某个 SPI 接口的 CRC 校验值
SPI_CalculateCRC	使能或取消传送字节的 CRC 校验值的计算
SPI_GetCRC	返回特定 SPI 接口传输或接收的 CRC 寄存器的值
SPI_GetCRCPolynomial	返回特定 SPI 接口的 CRC 多项式寄存器的值
SPI_BiDirectionalLineConfig	为特定的 SPI 在双向模式中选择数据传输方向
SPI_GetFlagStatus	检查指定 SPI 的标志是否被设置
SPI_ClearFlag	清除某个 SPI 的挂起标志
SPI_GetITStatus	检查特定的 SPI 中断是否出现
SPI_ClearITPendingBit	清除某个 SPI 的中断挂起位

17.2.1 SPI_DeInit 函数

表 409 描述了 SPI_DeInit 函数。

表 409. SPI_DeInit 函数

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

行为描述	将 SPIx 外设寄存器重置为他们的缺省值
输入参数	SPI*x* x 为 1 或 2 用于选定 SPI 外设
输出参数	无
返回值	无
前提条件	无
调用函数	为 SPI1 调用 RCC_APB2PeriphClockCmd() 为 SPI2 调用 RCC_APB1PeriphClockCmd()

例:

```
/* 初始化 SPI2 */
SPI_DeInit(SPI2);
```

17.2.2 SPI_Init 函数

表 410 描述了 SPI_Init 函数。

表 410. SPI_Init 函数

函数名	SPI_Init
函数原型	void SPI_Init(SPI_TypeDef* SPIx, SPI_InitTypeDef* SPI_InitStruct)
行为描述	根据 SPI_InitStruct.中的特定参数初始化 SPIx 外设

输入参数 1	SPI*x*：x 为 1 或 2 用于选定 SPI 外设
输入参数 2	SPI_InitStruct ：指向一个包含特定 SPI 外设配置信息的 SPI_InitTypeDef 结构体的指针。 参考 *SPI_InitTypeDef 结构* 部分以了解参数的允许值方面的的更多信息。
输出参数	无
返回值	无
前提条件	无
调用函数	无

SPI_InitTypeDef 结构

SPI_InitTypeDef 在文件 *stm32f10x_spi.h* 中定义如下：

```
typedef struct
{
u16 SPI_Direction;
u16 SPI_Mode;
u16 SPI_DataSize;
u16 SPI_CPOL;
u16 SPI_CPHA;
u16 SPI_NSS;
u16 SPI_BaudRatePrescaler;
u16 SPI_FirstBit;
```

u16 SPI_CRCPolynomial;

} SPI_InitTypeDef;

成员 SPI_Direction

SPI_Direction 配置 SPI 单向或双向数据传输模式，参看表 411 获取这个成员可用的值。

表 411. SPI_Direction 的定义

SPI_Direction	描述
SPI_Direction_2Lines_FullDuplex	SPI 配置成两条线的单向全双工通信
SPI_Direction_2Lines_RxOnly	SPI 配置成两条线的单向仅 Rx 通信
SPI_Direction_1Line_Rx	SPI 配置成一条线的双向仅 Rx 通信
SPI_Direction_1Line_Tx	SPI 配置成一条线的双向仅 Tx 通信

成员 SPI_Mode

SPI_Mode 配置 SPI 的操作模式。参考表 412 获取这个成员的可用值。

表 412. SPI_Mode 定义

SPI_Mode	描述
SPI_Mode_Master	SPI 配置为一个主机

SPI_Mode_Slave	SPI 配置为一个从机

成员 SPI_DataSize

SPI_DataSize 配置 SPI 的数据大小。参考表 413 获取这个成员的可用值。

表 413. SPI_DataSize 定义

SPI_DataSize	描述
SPI_DataSize_16b	用 SPI 16 位数据帧格式传输或接收
SPI_DataSize_8b	用 SPI 8 位数据帧格式传输或接收

成员 SPI_CPOL

SPI_CPOL 选择串行时钟频率的稳定状态。参考表 414 获取这个成员的可用值。

表 414. SPI_CPOL 定义

SPI_CPOL	描述
SPI_CPOL_High	时钟高时活动
SPI_CPOL_Low	时钟低时活动

成员 SPI_CPHA

SPI_CPHA 配置用于位捕获的时钟边沿。参考表 415 获取这个成员的可用值。

表 415. SPI_CPHA 定义

SPI_CPHA	描述
SPI_CPHA_2Edge	数据在时钟第二个边沿时捕捉
SPI_CPHA_1Edge	数据在时钟第一个边沿时捕捉

成员 SPI_NSS

SPI_NSS 指定 NSS 由硬件（NSS 针脚）还是软件（使用 SSI 位）管理。参考表 416 获取这个成员的可用值。

表 416. SPI_NSS 定义

SPI_NSS	描述
SPI_NSS_Hard	NSS 由外部针脚管理
SPI_NSS_Soft	内部 NSS 信号由 SSI 控制

成员 SPI_BaudRatePrescaler

SPI_BaudRatePrescaler 用来定义波特率预分频数的值，这个因子将用来配置传输或接收 SCK 的时钟频率。参考表 417 获取这个成员的可用值。

表 417. SPI_BaudRatePrescaler 定义

SPI_BaudratePrescaler	描述
SPI_BaudRatePrescaler2	波特率预分频数为 2
SPI_BaudRatePrescaler4	波特率预分频数为 4

SPI_BaudRatePrescaler8	波特率预分频数为 8
SPI_BaudRatePrescaler16	波特率预分频数为 16
SPI_BaudRatePrescaler32	波特率预分频数为 32
SPI_BaudRatePrescaler64	波特率预分频数为 64
SPI_BaudRatePrescaler128	波特率预分频数为 128
SPI_BaudRatePrescaler256	波特率预分频数为 256

注意：通信时钟源于主时钟，从时钟不必设置。

成员 SPI_FirstBit

SPI_FirstBit 指定传输数据从 MSB 还是 LSB 开始。参考表 418 获取这个成员的可用值。

表 418. SPI_FirstBit 定义

SPI_FirstBit	描述
SPI_FisrtBit_MSB	传输的第一位是 MSB
SPI_FisrtBit_LSB	传输的第一位是 LSB

成员 SPI_CRCPolynomial

SPI_CRCPolynomial 定义用于 CRC 计算的多项式。

例：

```
/* 根据 SPI_InitStructure 成员初始化 SPI */

SPI_InitTypeDef SPI_InitStructure;

SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
```

```
SPI_InitStructure.SPI_Mode = SPI_Mode_Master;

SPI_InitStructure.SPI_DatSize = SPI_DatSize_16b;

SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low;

SPI_InitStructure.SPI_CPHA = SPI_CPHA_2Edge;

SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;

SPI_InitStructure.SPI_BaudRatePrescaler =

SPI_BaudRatePrescaler_128;

SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;

SPI_InitStructure.SPI_CRCPolynomial = 7;

SPI_Init(SPI1, &SPI_InitStructure);.
```

17.2.3 SPI_StructInit 函数

表 419 描述了 SPI_StructInit 函数。

表 419. SPI_StructInit 函数

函数名	SPI_StructInit
函数原型	void SPI_StructInit(SPI_InitTypeDef* SPI_InitStruct)
行为描述	使用缺省值填充 SPI_InitStruct 每一个成员
输入参数	SPI_InitStruct: 指向一个将被初始化的 SPI_InitTypeDef 结构的指针

输出参数	无
返回值	无
前提条件	无
调用函数	无

参考表 420 中 SPI_InitStruct 成员的缺省值。

表 420. SPI_InitStruct 缺省值

成员	缺省值
SPI_Direction	SPI_Direction_2Lines_FullDuplex
SPI_Mode	SPI_Mode_Slave
SPI_DataSize	SPI_DataSize_8b
SPI_CPOL	SPI_CPOL_Low
SPI_CPHA	SPI_CPHA_1Edge
SPI_NSS	SPI_NSS_Hard
SPI_BaudRatePrescaler	SPI_BaudRatePrescaler_2
SPI_FirstBit	SPI_FirstBit_MSB
SPI_CRCPolynomial	7

例:

```
/* 初始化一个 SPI_InitTypeDef 结构 */
```

SPI_InitTypeDef SPI_InitStructure;

SPI_StructInit(&SPI_InitStructure);

17.2.4 SPI_Cmd 函数

表 421 描述了 SPI_Cmd 函数。

表 421. SPI_Cmd 函数

函数名	SPI_Cmd
函数原型	void SPI_Cmd(SPI_TypeDef* SPIx, FunctionalState NewState)
行为描述	使能或关闭特定的外设
输入参数 1	SPIx: x 可能是 1 或 2 来选择外设
输入参数 2	NewState: 这个 SPI 外设的新状态，这个参数可能是 ENABLE 或 DISABLE
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

/* 使能 SPI1 */

SPI_Cmd(SPI1, ENABLE);

17.2.5 SPI_ITConfig 函数

表 422 介绍了 SPI_ITConfig 函数

表 422. SPI_ITConfig 函数

函数名	SPI_ITConfig
函数原型	void SPI_ITConfig(SPI_TypeDef* SPIx, u8 SPI_IT, FunctionalState NewState)
行为描述	使能或关闭特定的 SPI 中断
输入参数 1	SPIx: x 可能是 1 或 2 来选择外设
输入参数 2	SPI_IT: 将被使能或关闭的中断源 参考 *SPI_IT* 部分获得对该参数更细致的了解
输入参数 3	NewState: 指定 SPI 中断的新状态，这个参数可能是 ENABLE 或 DISABLE
输出参数	无
返回值	无
前提条件	无
调用函数	无

参数 SPI_IT

SPI_IT 使能或关闭中断。参考表 423 获取这个参数的可用值。

表 423. SPI_IT 标志

SPI_IT 描述

SPI_IT	描述
SPI_IT_TXE	Tx 缓冲区为空的中断屏蔽
SPI_IT_RXNE	Rx 缓冲区为空的中断屏蔽
SPI_IT_ERR	错误中断屏蔽

例：

```
/* 使能 SPI2 Tx 缓冲区为空的中断 */
SPI_ITConfig(SPI2, SPI_IT_TXE, ENABLE);
```

17.2.6 SPI_DMACmd 函数

表 424 描述了 SPI_DMACmd 函数

表 424. SPI_DMACmd 函数

函数名	SPI_DMACmd
函数原型	void SPI_DMACmd(SPI_TypeDef* SPIx, u16 SPI_DMAReq, FunctionalState NewState)

行为描述	使能或关闭某个的 SPI 外设的 DMA 接口
输入参数 1	SPIx: x 可能是 1 或 2 来选择外设
输入参数 2	SPI_DMAReq: 将被使能或关闭的 SPI DMA 传输请求 参考 *SPI_DMAReq* 部分获得对该参数更细致的了解
输入参数 3	NewState: 选中的 SPI DMA 传输请求的新状态，这个参数可能是 ENABLE 或 DISABLE
输出参数	无
返回值	无
前提条件	无
调用函数	无

SPI_DMAReq

SPI_DMAReq 使能或关闭 SPI Tx 和/或 Rx DMA 传输请求。参考表 425 获取这个参数的可用值。

表 425. SPI_DMAReq 的可用值

SPI_DMAReq	描述
SPI_DMAReq_Tx	选中 Tx 缓冲区 DMA 传输请求
SPI_DMAReq_Rx	选中 Rx 缓冲区 DMA 传输请求

例：

```
/* 使能 SPI2 Rx 缓冲区的 DMA 传输请求 */

SPI_DMACmd(SPI2, SPI_DMAReq_Rx, ENABLE);
```

17.2.7 SPI_SendData 函数

表 426 描述了 SPI_SendData 函数。

表 426. SPI_SendData 函数

函数名	SPI_SendData
函数原型	void SPI_SendData(SPI_TypeDef* SPIx, u16 Data)
行为描述	通过某个 SPI 外设传输数据
输入参数 1	SPIx: x 可能是 1 或 2 来选择外设
输入参数 2	Data：将被传输的字节或半字
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 通过 SPI1 外设发送 0xA5 */

SPI_SendData(SPI1, 0xA5);
```

17.2.8 SPI_ReceiveData 函数

表 427 描述了 SPI_ReceiveData 函数。

表 427. SPI_ReceiveData 函数

函数名	SPI_ReceiveData
函数原型	u16 SPI_ReceiveData(SPI_TypeDef* SPIx)
行为描述	返回从某个的 SPI 外设最近接收到的数据
输入参数	SPIx: x 可能是 1 或 2 来选择外设
输出参数	无
返回值	接收到的数据的值
前提条件	无
调用函数	无

例：

```
/* 读取从 SPI2 外设最近接收到的数据 */

u16 ReceivedData;

ReceivedData = SPI_ReceiveData(SPI2);
```

17.2.9 SPI_NSSInternalSoftwareConfig 函数

表 428 描述了 SPI_NSSInternalSoftwareConfig 函数。

表 428. SPI_NSSInternalSoftwareConfig 函数

函数名	SPI_NSSInternalSoftwareConfig
函数原型	void SPI_NSSInternalSoftwareConfig(SPI_TypeDef* SPIx, u16 SPI_NSSInternalSoft)
行为描述	为特定的 SPI 接口用软件配置 NSS 针脚
输入参数 1	SPIx: x 可能是 1 或 2 来选择外设
输入参数 2	SPI_NSSInternalSoft: SPI NSS 的内部状态 参考 SPI_NSSInternalSoft 部分获取更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

参数 SPI_NSSInternalSoft

SPI_NSSInternalSoft 内部设置或重置 NSS 引脚。参看表 429 获取该参数的允许值。

表 429. SPI_NSSInternalSoft 的允许值

SPI_NSSInternalSoft	描述
SPI_NSSInternalSoft_Set	内部设置 NSS 针脚
SPI_NSSInternalSoft_Reset	内部重置 NSS 针脚

例：

```
/* 软件内部设置 SPI1 NSS 针脚 */
SPI_NSSInternalSoftwareConfig(SPI1, SPI_NSSInternalSoft_Set);
/* 软件内部重置 SPI2 NSS 针脚 */
SPI_NSSInternalSoftwareConfig(SPI2, SPI_NSSInternalSoft_Reset);
```

17.2.10 SPI_SSOutputCmd 函数

表 430 描述了 SPI_SSOutputCmd 函数。

Table 430. SPI_SSOutputCmd function

函数名	SPI_SSOutputCmd
函数原型	void SPI_SSOutputCmd(SPI_TypeDef* SPIx, FunctionalState
行为描述	使能或关闭选定的 SPI 接口的 SS 输出
输入参数 1	SPIx: x 可能是 1 或 2 来选择外设
输入参数 2	NewState: 选定的 SPI SS 输出的新状态 ,这个参数可能是 ENABLE 或 DISABLE
输出参数	无
返回值	无
前提条件	无

调用函数	无

例：

```
/* 使能 SPI1 SS 输出:单主机模式 */

SPI_SSOutputCmd(SPI1, ENABLE);
```

17.2.11 SPI_DatSizeConfig 函数

表 431 描述了 SPI_DatSizeConfig 函数

表 431. SPI_DatSizeConfig 函数

函数名	SPI_DataSizeConfig
函数原型	void SPI_DataSizeConfig(SPI_TypeDef* SPIx, u16 SPI_DatSize)
行为描述	为选定的 SPI 接口配置数据大小
输入参数 1	SPIx: x 可能是 1 或 2 来选择外设
输入参数 2	SPI_DataSize: SPI 数据大小 参考 SPI_DataSize 部分获取更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

参数 void SPI_DataSizeConfig(SPI_TypeDef* SPIx, u16 SPI_DatSize) **SPI_DataSize**

SPI_DataSize 设置 8 位或 16 位数据帧格式。参看表 432 获取该参数的允许值。

表 432. SPI_DataSize 允许值

SPI_NSSInternalSoft	描述
SPI_DataSize_8b	设置数据大小为 8 位
SPI_DataSize_16b	设置数据大小为 16 位

例：

```
/* 为 SPI1 设置 8 位数据帧格式 */
SPI_DataSizeConfig(SPI1, SPI_DataSize_8b);
/* 为 SPI2 设置 16 位数据帧格式 */
SPI_DataSizeConfig(SPI2, SPI_DataSize_16b);
```

17.2.12 SPI_TransmitCRC 函数

表 433 描述了 SPI_TransmitCRC 函数。

表 433. SPI_TransmitCRC 函数

函数名	SPI_TransmitCRC

函数原型	void SPI_TransmitCRC(SPI_TypeDef* SPIx)
行为描述	传送某个 SPI 的 CRC 校验值
输入参数	SPIx: x 可能是 1 或 2 来选择外设
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 使能 SPI1 的 CRC 信息传送*/
SPI_TransmitCRC(SPI1);
```

17.2.13 SPI_CalculateCRC 函数

表 434 描述了 SPI_CalculateCRC 函数。

表 434. SPI_CalculateCRC 函数

函数名	SPI_CalculateCRC
函数原型	void SPI_CalculateCRC(SPI_TypeDef* SPIx, FunctionalState NewState)
行为描述	使能或取消传送字节的 CRC 校验值的计算

输入参数 1	SPIx: x 可能是 1 或 2 来选择外设
输入参数 2	NewState: 选定 SPI 接口的 CRC 计算的新状态，这个参数可能是 ENABLE 或 DISABLE
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 使能来自 SPI2 的传输字节的 CRC 计算 */
SPI_CalculateCRC(SPI2, ENABLE);
```

17.2.14 SPI_GetCRC 函数

表 435 描述了 SPI_GetCRC 函数。

表 435. SPI_GetCRC function

函数名	SPI_GetCRC
函数原型	u16 SPI_GetCRC(SPI_TypeDef* SPIx, u8 SPI_CRC)
行为描述	返回特定 SPI 外设传送或接收的 CRC 寄存器的值
输入参数 1	SPIx: x 可能是 1 或 2 来选择外设

输入参数 2	SPI_CRC: 将被读取的 CRC 寄存器 参考 *SPI_CRC* 部分获取更多细节
输出参数	无
返回值	所选择的 CRC 寄存器的值
前提条件	无
调用函数	无

参数 SPI_CRC

SPI_CRC 选择 SPI Rx 或 SPI Tx CRC 寄存器。参看 436 获取该参数的允许值

表 436. SPI_CRC 允许值

SPI_CRC Description

SPI_NSSInternalSoft	描述
SPI_CRC_Tx	选择 Tx CRC 寄存器
SPI_CRC_Rx	选择 Rx CRC 寄存器

例：

```
/* 返回 SPI1 的传送 CRC 寄存器的值 */
u16 CRCValue;
CRCValue = SPI_GetCRC(SPI1, SPI_CRC_Tx);
```

17.2.15 SPI_GetCRCPolynomial 函数

表 437 描述了 SPI_GetCRCPolynomial 函数。

表 437. SPI_GetCRCPolynomial 函数

函数名	SPI_GetCRCPolynomial
函数原型	u16 SPI_GetCRCPolynomial(SPI_TypeDef* SPIx)
行为描述	返回特定 SPI 接口的 CRC 多项式寄存器的值
输入参数	SPIx: x 可能是 1 或 2 来选择外设
输出参数	无
返回值	CRC 多项式寄存器的值
前提条件	无
调用函数	无

例：

```
/* 返回 SPI2 的 CRC 多项式寄存器 */
u16 CRCPolyValue;
CRCPolyValue = SPI_GetCRCPolynomial(SPI2);
```

17.2.16 SPI_BiDirectionalLineConfig 函数

表 438 描述了 SPI_BiDirectionalLineConfig 函数。

表 438. SPI_BiDirectionalLineConfig function

函数名	SPI_BiDirectionalLineConfig
函数原型	void SPI_BiDirectionalLineConfig(SPI_TypeDef* SPIx, u16 SPI_Direction)
行为描述	为特定的 SPI 接口在双向模式时选择数据传输方向
输入参数 1	SPIx: x 可能是 1 或 2 来选择外设
输入参数 2	SPI_Direction: 在双向模式中的数据传输方向 参考 SPI_ Direction 部分获取更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

参数 SPI_Direction

SPI_Direction 在双向模式中配置数据传输方向。参看 439 获取该参数的允许值

表 439. SPI_Direction 允许值

SPI_ Direction	描述
SPI_ Direction_Tx	选择 Tx 传送方向

SPI_ Direction_Rx	选择 Rx 接收方向

例：

/* 在双向传输模式中设置 SPI2*/

SPI_BiDirectionalLineConfig(SPI_Direction_Tx);

17.2.17 SPI_GetFlagStatus 函数

表 440 描述了 SPI_GetFlagStatus 函数。

表 440. SPI_GetFlagStatus 函数

函数名	SPI_ GetFlagStatus
函数原型	FlagStatus SPI_GetFlagStatus(SPI_TypeDef* SPIx, u16 SPI_FLAG)
行为描述	检查指定的 SPI 标记是否被设置
输入参数 1	SPIx: x 可能是 1 或 2 来选择外设
输入参数 2	SPI_ FLAG: 将检查的标记 参考 *SPI_ FLAG* 部分获取更多细节
输出参数	无
返回值	SPI_FLAG 标记的新状态
前提条件	无

调用函数	无

参数 SPI_FLAG

可调用 SPI_GetFlagStatus 函数检查的 SPI 标记见表 441.

表 441. SPI_FLAG 标记

SPI_ FLAG	描述
SPI_FLAG_BSY	忙碌标记
SPI_FLAG_OVR	溢出标记
SPI_FLAG_MODF	模式错误标记
SPI_FLAG_CRCERR	CRC 校验错误标记
SPI_FLAG_TXE	传输缓冲为空标记
SPI_FLAG_RXNE	接收缓冲不空标记

例：

```
/* 检查 SPI1 传输缓冲为空标记是否被设置 */
FlagStatus Status;
Status = SPI_GetFlagStatus(SPI1, SPI_FLAG_TXE);
```

17.2.18 SPI_ClearFlag 函数

表 442 描述了 SPI_ClearFlag 函数.

表 442. SPI_ClearFlag 函数

函数名	SPI_ ClearFlag
函数原型	void SPI_ClearFlag(SPI_TypeDef* SPIx, u16 SPI_FLAG)
行为描述	清除某个 SPI 外设的挂起位
输入参数 1	SPIx: x 可能是 1 或 2 来选择外设
输入参数 2	SPI_ FLAG: 将被清除的标记 参考 *SPI_ FLAG* 部分获取更多细节 注意：BSY, TXE 和 RXNE 标记是被硬件重置
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 清除 SPI2 的溢出挂起位 */
SPI_ClearFlag(SPI2, SPI_FLAG_OVR);
```

17.2.19 SPI_GetITStatus 函数

表 443 描述了 SPI_GetITStatus 函数。

表 443. SPI_GetITStatus 函数

函数名	SPI_ GetITStatus
函数原型	ITStatus SPI_GetITStatus(SPI_TypeDef* SPIx, u8 SPI_IT)
行为描述	检查特定的 SPI 中断是否发生
输入参数 1	SPIx: x 可能是 1 或 2 来选择外设
输入参数 2	SPI_IT: 要被检查的 SPI 中断源 参考 *SPI_ IT* 部分获取更多细节
输出参数	无
返回值	SPI 的新状态
前提条件	无
调用函数	无

参数 SPI_IT

可调用 SPI_GetITStatus 函数检查的 SPI 标记见表 444.

表 444. SPI_IT 标记

SPI_ IT	描述
SPI_IT_OVR	溢出中断标记
SPI_IT_MODF	模式错误中断标记
SPI_IT_CRCERR	CRC 校验错误中断标记

SPI_IT_TXE	传输缓冲为空中断标记
SPI_IT_RXNE	接收缓冲不空中断标记

例：:

```
/* 检查 SPI1 的溢出中断是否发生 */
ITStatus Status;
Status = SPI_GetITStatus(SPI1, SPI_IT_OVR);
```

17.2.20 SPI_ClearITPendingBit 函数

表 445 描述了 SPI_ClearITPendingBit 函数

表 445. SPI_ClearITPendingBit 函数

函数名	SPI_ ClearITPendingBit
函数原型	void SPI_ClearITPendingBit(SPI_TypeDef* SPIx, u8 SPI_IT)
行为描述	清除某个 SPI 外设的中断挂起位
输入参数 1	SPIx: x 可能是 1 或 2 来选择外设
输入参数 2	SPI_IT: 指定将被清除的中断挂起位 参考 *SPI_ IT* 部分获取更多细节 注意：BSY, TXE 和 RXNE 中断是被硬件重置
输出参数	无

返回值	无
前提条件	无
调用函数	无

例：

```
/* 清除 SPI2 CRC 错误中断挂起位 */
SPI_ClearITPendingBit(SPI2, SPI_IT_CRCERR);
```

18 Coretex系统计时器（SysTick)

Systick 提供了一个简单的 24 位递减的,到零重装,写清零计数器，并带有灵活的控制机制。18.1: SysTick 寄存器结构 描述了在 SysTick 固件库中使用的数据结构。18.2: 固件库函数介绍了固件库函数。

18.1 SysTick 寄存器结构

SysTick 寄存器结构，SysTick_TypeDef,在 stm32f10x_map.h 文件中定义如下:

```
typedef struct
{
vu32 CTRL;
vu32 LOAD;
vu32 VAL;
```

vuc32 CALIB;

} SysTick_TypeDef;

表 446 列出了 SysTick 的寄存器。

表 446. SysTick 寄存器

寄存器	描述
CTRL	SysTick 控制和状态寄存器
LOAD	SysTick 重装值寄存器
VAL	SysTick 当前值寄存器
CALIB	SysTick 校准值寄存器

The SysTick 外设在 *stm32f10x_map*.h 文件中声明：

```
#define SCS_BASE ((u32)0xE000E000)

#define SysTick_BASE (SCS_BASE + 0x0010)

#ifndef DEBUG

...

#ifdef _SysTick

#define SysTick ((SysTick_TypeDef *) SysTick_BASE)

#endif /*_SysTick */

...

#else /* DEBUG */

...
```

```
#ifdef _SysTick
EXT SysTick_TypeDef *SysTick;
#endif /*_SysTick */
...
#endif
```

当使用调试模式，SysTick 指针在 *stm32f10x_lib.c* 文件中初始化：

```
#ifdef _SysTick
SysTick = (SysTick_TypeDef *) SysTick_BASE;
#endif /*_SysTick */
```

To access the SysTick registers, _SysTick must be defined in *stm32f10x_conf.h* as follows:

```
#define _SysTick
```

18.2 固件库函数

表 447 列出了 SysTick 的库函数。

表 447. SysTick 固件库函数

函数名	描述
SysTick_CLKSourceConfig	配置 SysTick 的时钟源
SysTick_SetReload	设置 SysTick 的重载值

SysTick_CounterCmd	使能或取消 SysTick 计数器
SysTick_ITConfig	使能或取消 SysTick 中断
SysTick_GetCounter	获取 SysTick 计数器值
SysTick_GetFlagStatus	检查指定的 SysTick 标记是否被设置

18.2.1 SysTick_CLKSourceConfig 函数

表 448 描述了 SysTick_CLKSourceConfig 函数

表 448. SysTick_CLKSourceConfig 函数

函数名	SysTick_CLKSourceConfig
函数原型	void SysTick_CLKSourceConfig(u32 SysTick_CLKSource)
行为描述	配置 SysTick 的时钟源
输入参数	SysTick_CLKSource: SysTick 的时钟源 参考 SysTick_CLKSource 部分获取该参数的允许值
输出参数	无
返回值	无
前提条件	无
调用函数	无

SysTick_CLKSource

SysTick_CLKSource 选择 SysTick 时钟源。参考表 449 查看该参数的允许值。

表 449. SysTick_CLKSource 的允许值

SysTick_CLKSource	描述
SysTick_CLKSource_HCLK_Div8	SysTick 时钟源为 AHB 时钟的 1/8
SysTick_CLKSource_HCLK	SysTick 时钟源为 AHB 时钟

例：

```
/* 选择 AHB 时钟作为 SysTick 的时钟源 */
SysTick_CLKSourceConfig(SysTick_CLKSource_HCLK);
```

18.2.2 SysTick_SetReload 函数

表 450 描述了 SysTick_SetReload 函数

表 450. SysTick_SetReload 函数

函数名	SysTick_ SetReload
函数原型	void SysTick_SetReload(u32 Reload)

行为描述	设置 SysTick 的重载值
输入参数	Reload：SysTick 重载一个新值，这个参数必须在 1 和 0x00FFFFFF 之间
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 设置 SysTick 的重载值为 0xFFFF */
SysTick_SetReload(0xFFFF);
```

18.2.3 SysTick_CounterCmd 函数

表 451 描述了 SysTick_CounterCmd 函数。

表 451. SysTick_CounterCmd 函数

函数名	SysTick_CounterCmd
函数原型	void SysTick_CounterCmd(u32 SysTick_Counter)
行为描述	使能或取消 SysTick 计数
输入参数	SysTick_Counter: SysTick 计数器的新状态

	参考 SysTick_ Counter 部分获取该参数的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

参数 SysTick_Counter

SysTick_Counter 选择 SysTick 的计数器状态。参考表 452 查看该参数的允许值。

表 452. SysTick_Counter 的允许值

SysTick_ Counter	描述
SysTick_Counter_Disable	使能计数器
SysTick_Counter_Enable	关闭计数器
SysTick_Counter_Clear	清零计数器

Example:

```
/* 使能 SysTick 计数 */

SysTick_CounterCmd(SysTick_Counter_Enable);
```

18.2.4 SysTick_ITConfig 函数

表 453 描述了 SysTick_ITConfig 函数

表 453. SysTick_ITConfig 函数

函数名	SysTick_ ITConfig
函数原型	void SysTick_ITConfig(FunctionalState NewState)
行为描述	使能或取消 SysTick 中断
输入参数	NewState: SysTick 中断的新状态 这个参数可以是 ENABLE 或是 DISABLE
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 使能 SysTick 中断*/
SysTick_ITConfig(ENABLE);
```

18.2.5 SysTick_GetCounter 函数

表 454 描述了 SysTick_GetCounter 函数。

表 454. SysTick_GetCounter 函数

函数名	SysTick_ GetCounter
函数原型	U32 SysTick_GetCounter(void)
行为描述	获取 SysTick 计数器的值
输入参数	无
输出参数	无
返回值	SysTick 的当前值
前提条件	无
调用函数	无

例:

```
/* 获取 SysTick 当前计数值 */
u32 SysTickCurrentCounterValue;
SysTickCurrentCounterValue = SysTick_GetCounter();
```

18.2.6 SysTick_GetFlagStatus 函数

表 455 描述了 SysTick_GetFlagStatus 函数

表 455. SysTick_GetFlagStatus 函数

函数名	SysTick_ GetFlagStatus
函数原型	FlagStatus SysTick_GetFlagStatus(u8 SysTick_FLAG)
行为描述	检查特定的 SysTick 标记是否被设置
输入参数	SysTick_FLAG: 将检查的标记 参考 *SysTick_ FLAG* 部分获取该参数的更多细节
输出参数	无
返回值	SysTick_FLAG 的新状态(SET 或 RESET)
前提条件	无
调用函数	无

参数 SysTick_FLAG

可以调用 SysTick_GetFlagStatus 函数检查的 SysTick 标记列举如下：

表 456. SysTick 标记

SysTick_ FLAG	描述
SysTick_ FLAG_COUNT	1 =自从上次读取计数器数到 0
SysTick_FLAG_SKEW	1 = 由于时钟频率的原因,校准值不是准确的 10ms
SysTick_FLAG_NOREF	1 = 没有提供参考时钟

例：

```
/* 检查计数标记是否被设置 */
FlagStatus Status;
Status = SysTick_GetFlagStatus(SysTick_FLAG_COUNT);
if(Status == RESET)
{
...
}
else
{
...
}
```

19 通用计时器 (TIM)

这个计时器含有一个由可编程的预分频数驱动的 16 位的自动重载计数器。他可以被用作许多用途，包括输入信号脉冲长度的测量（输入捕获）和输出波形的生成（输出比较, PWM)。使用计时器预分频数和 CPU 时钟预分频数可以将脉冲长度和波形周期在几微秒到若干毫秒的范围内调制。*18.1: SysTick 寄存器结构* 描述了在 TIM 的固件库中使用的数据结构。*18.2:固件库函数* 介绍了固件库函数。

19.1 TIM 寄存器结构

TIM 寄存器结构，TIM_TypeDef，在文件 stm32f10x_map.h 中定义如下：

```
typedef struct
{
vu16 CR1;
u16 RESERVED0;
vu16 CR2;
u16 RESERVED1;
vu16 SMCR;
u16 RESERVED2;
vu16 DIER;
u16 RESERVED3;
vu16 SR;
u16 RESERVED4;
vu16 EGR;
u16 RESERVED5;
vu16 CCMR1;
u16 RESERVED6;
vu16 CCMR2;
u16 RESERVED7;
```

```
vu16 CCER;
u16 RESERVED8;
vu16 CNT;
u16 RESERVED9;
vu16 PSC;
u16 RESERVED10;
vu16 ARR;
u16 RESERVED11[3];
vu16 CCR1;
u16 RESERVED12;
vu16 CCR2;
u16 RESERVED13;
vu16 CCR3;
u16 RESERVED14;
vu16 CCR4;
u16 RESERVED15[3];
vu16 DCR;
u16 RESERVED16;
vu16 DMAR;
u16 RESERVED17;
} TIM_TypeDef;
```

表 457 列出了 TIM 的所有寄存器。.

表 457. TIM 寄存器

寄存器	描述
CR1	控制寄存器 1
CR2	控制寄存器 2
SMCR	从模式控制寄存器
DIER	DMA 和中断使能寄存器
SR	状态寄存器
EGR	事件生成寄存器
CCMR1	捕获/比较模式寄存器 1
CCMR2	捕获/比较模式寄存器 2
CCER	捕获/比较使能寄存器
CNT	计数寄存器
PSC	预分频数寄存器
ARR	自动重载寄存器
CCR1	捕获/比较寄存器 1
CCR2	捕获/比较寄存器 2
CCR3	捕获/比较寄存器 3
CCR4	捕获/比较寄存器 4
DCR	DMA 控制寄存器

DMAR	DMA 猝发模式下的地址寄存器

三个 TIM 外设在同一个文件中声明：

```
...
#define PERIPH_BASE ((u32)0x40000000)
#define APB1PERIPH_BASE PERIPH_BASE
#define APB2PERIPH_BASE (PERIPH_BASE + 0x10000)
#define AHBPERIPH_BASE (PERIPH_BASE + 0x20000)
#define TIM2_BASE (APB1PERIPH_BASE + 0x0000)
#define TIM3_BASE (APB1PERIPH_BASE + 0x0400)
#define TIM4_BASE (APB1PERIPH_BASE + 0x0800)
...
#ifndef DEBUG
...
#ifdef _TIM2
#define TIM2 ((TIM_TypeDef *) TIM2_BASE)

#endif /*_TIM2 */
#ifdef _TIM3
#define TIM3 ((TIM_TypeDef *) TIM3_BASE)
#endif /*_TIM3 */
```

```
#ifdef _TIM4

#define TIM4 ((TIM_TypeDef *) TIM4_BASE)

#endif /*_TIM4 */

...

#else /* DEBUG */

...

#ifdef _TIM2

EXT TIM_TypeDef *TIM2;

#endif /*_TIM2 */

#ifdef _TIM3

EXT TIM_TypeDef *TIM3;

#endif /*_TIM3 */

#ifdef _TIM4

EXT TIM_TypeDef *TIM4;

#endif /*_TIM4 */

...

#endif
```

当使用调试模式，TIM2, TIM3 和 TIM4 的指针 在文件 *stm32f10x_lib.c* 中初始化如下：:

```
...

#ifdef _TIM2

TIM2 = (TIM_TypeDef *) TIM2_BASE;
```

```
#endif /*_TIM2 */

#ifdef _TIM3

TIM3 = (TIM_TypeDef *) TIM3_BASE;

#endif /*_TIM3 */

#ifdef _TIM4

TIM4 = (TIM_TypeDef *) TIM4_BASE;

#endif /*_TIM4 */

...
```

要访问 TIM 寄存器，_TIM，_TIM2，_TIM3 和 _TIM4 必须在文件 stm32f10x_conf.h 定义如下：

```
...

#define _TIM

#define _TIM2

#define _TIM3

#define _TIM4

...
```

19.2 固件库函数

表 458 列出了 TIM 库中的各种函数。

表 458. TIM library firmware 函数

函数名	描述
TIM_DeInit	重置 TIMx 外设寄存器为默认的复位值
TIM_TimeBaseInit	根据 TIM_TimeBaseInitStruct 中的特定参数初始化 TIMx 的时间基单元
TIM_OCInit	根据 TIM_OCInitStruct 中的特定参数初始化 TIMx 外设
TIM_ICInit	根据 TIM_ICInitStruct 中的特定参数初始化 TIMx 外设
TIM_TimeBaseStructInit	使用缺省值填充 TIM_TimeBaseInitStruct 的每个成员
TIM_OCStructInit	使用缺省值填充 TIM_ OCInitStruct 的每个成员
TIM_ICStructInit	使用缺省值填充 TIM_ ICInitStruct 的每个成员
TIM_Cmd	使能或关闭特定 TIM 外设
TIM _ITConfig	使能或关闭特定 TIM 中断
TIM _ DMAConfig	配置 TIMx 的 DMA 接口
TIM _ DMACmd	使能或关闭 TIMx 的 DMA 请求
TIM _ InternalClockConfig	配置 TIMx 的内部时钟

TIM_ITRxExternalClockConfig	配置 TIMx 的内部触发器作为外部时钟
TIM_TIxExternalClockConfig	配置 TIMx 的触发器作为外部时钟
TIM_ETRClockMode1Config	配置 TIMx 的外部时钟模型 1
TIM_ETRClockMode2Config	配置 TIMx 的外部时钟模型 2
TIM_ETRConfig	配置 TIMx 的外部触发器(ETR)
TIM_ SelectInputTrigger	选择 TIMx 的输入触发源
TIM_ PrescalerConfig	配置 TIMx 的预分频数
TIM_ CounterModeConfig	指定要使用的 TIMx 计数器模式
TIM_ ForcedOC1Config	设置 TIMx 输出 1 的波形为有效或无效电平
TIM_ ForcedOC2Config	设置 TIMx 输出 2 的波形为有效或无效电平
TIM_ ForcedOC3Config	设置 TIMx 输出 3 的波形为有效或无效电平
TIM_ForcedOC4Config	设置 TIMx 输出 4 的波形为有效或无效电平
TIM_ ARRPreloadConfig	使能或关闭某个 TIM 外设 ARR 寄存器的预加载寄存器
TIM_ SelectCCDMA	选择 TIMx 外设的捕获比较 DMA 源
TIM_ OC1PreloadConfig	使能或关闭 TIMx 外设 CCR1 寄存器的预加载寄存器
TIM_ OC2PreloadConfig	使能或关闭 TIMx 外设 CCR2 寄存器的预加载寄存器

TIM_OC3PreloadConfig	使能或关闭 TIMx 外设 CCR3 寄存器的预加载寄存器
TIM_OC4PreloadConfig	使能或关闭 TIMx 外设 CCR4 寄存器的预加载寄存器
TIM_OC1FastConfig	配置某个 TIM 的输出比较 1 的快速特性
TIM_OC2FastConfig	配置某个 TIM 的输出比较 2 的快速特性
TIM_OC3FastConfig	配置某个 TIM 的输出比较 3 的快速特性
TIM_OC4FastConfig	配置某个 TIM 的输出比较 4 的快速特性
TIM_ClearOC1Ref	当一个外部事件发生时,清除或安全保护 OCREF1 信号
TIM_ClearOC2Ref	当一个外部事件发生时,清除或安全保护 OCREF2 信号
TIM_ClearOC3Ref	当一个外部事件发生时,清除或安全保护 OCREF3 信号
TIM_ClearOC4Ref	当一个外部事件发生时,清除或安全保护 OCREF4 信号
TIM_UpdateDisableConfig	使能或关闭某个 TIM 的更新事件
TIM_EncoderInterfaceConfig	配置某个 TIM 的编码器接口
TIM_GenerateEvent	配置某个 TIM 事件以软件方式生成
TIM_OC1PolarityConfig	配置某个 TIM 通道 1 的极性
TIM_OC2PolarityConfig	配置某个 TIM 通道 2 的极性
TIM_OC3PolarityConfig	配置某个 TIM 通道 3 的极性
TIM_OC4PolarityConfig	配置某个 TIM 通道 4 的极性
TIM_UpdateRequestConfig	配置某个 TIM 的更新请求源

TIM_SelectHallSensor	配置某个 TIM 的霍尔传感器接口
TIM_SelectOnePulseMode	让某个 TIM 选择单脉冲模式
TIM_SelectOutputTrigger	让某个 TIM 选择触发器输出模式
TIM_SelectSlaveMode	让某个 TIM 选择从模式
TIM_SelectMasterSlaveMode	设置或重置某个 TIM 为主模式或从模式
TIM_SetAutoreload	设置某个 TIM 的自动重载寄存器的值
TIM_SetCompare1	设置某个 TIM 捕获比较 1 寄存器的值
TIM_SetCompare2	设置某个 TIM 捕获比较 2 寄存器的值
TIM_SetCompare3	设置某个 TIM 捕获比较 3 寄存器的值
TIM_SetCompare4	设置某个 TIM 捕获比较 4 寄存器的值
TIM_SetIC1Prescaler	设置某个 TIM 输入捕获 1 的预分频数
TIM_SetIC2Prescaler	设置某个 TIM 输入捕获 2 的预分频数
TIM_SetIC3Prescaler	设置某个 TIM 输入捕获 3 的预分频数
TIM_SetIC4Prescaler	设置某个 TIM 输入捕获 4 的预分频数
TIM_SetClockDivision	设置某个 TIM 的时钟分割值
TIM_GetCapture1	读取某个 TIM 输入捕获 1 的值
TIM_GetCapture2	读取某个 TIM 输入捕获 2 的值
TIM_GetCapture3	读取某个 TIM 输入捕获 3 的值

TIM_GetCapture4	读取某个 TIM 输入捕获 4 的值
TIM_GetCounter	读取某个 TIM 计数器的值
TIM_GetPrescaler	读取某个 TIM 预分频数
TIM_GetFlagStatus	检查某个 TIM 的特定标记是否被设定
TIM_ClearFlag	清除某个 TIM 的挂起标记
TIM_GetITStatus	检查特定的 TIM 中断是否发生
TIM_ClearITPendingBit	清除某个 TIM 的中断挂起位

19.2.1 TIM_DeInit 函数

表 459 描述了 TIM_DeInit 函数。

表 459. TIM_DeInit 函数

函数名	TIM_DeInit
函数原型	void TIM_DeInit(TIM_TypeDef* TIMx)
行为描述	重置 TIMx 外设寄存器为其默认复位值
输入参数	TIMx: 此处 x 可以是 2，3 或 4 以选择外设
输出参数	无
返回值	无
前提条件	无
调用函数	RCC_APB1PeriphResetCmd

例：

```
/* 重置 TIM2 */
TIM_DeInit(TIM2);
```

19.2.2 TIM_TimeBaseInit 函数

表 460 描述了 TIM_TimeBaseInit 函数。

表 460. TIM_TimeBaseInit 函数

函数名	TIM_ TimeBaseInit
函数原型	void TIM_TimeBaseInit(TIM_TimeBaseInitTypeDef* TIM_BaseInitStruct)
行为描述	根据 TIM_TimeBaseInitStruct 中的特定参数初始化某个 TIM 的时间基单元
输入参数	TIM_BaseInitStruct: 指向一个包含 TIM 时间基配置信息的 TIM_BaseInitTypeDef 结构的指针 参考 TIM_TimeBaseInitTypeDef 结构 获取该参数允许值的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

TIM_TimeBaseInitTypeDef 结构

TIM_BaseInitTypeDef 在文件 *stm32f10x_tim.h* 中定义如下：

```
typedef struct
{
u16 TIM_Period;
u16 TIM_Prescaler;
u8 TIM_ClockDivision;
```

u16 TIM_CounterMode;

} TIM_BaseInitTypeDef;

成员 TIM_Period

TIM_Period 配置周期值，它将在下一次更新事件时被加载入有效的自动重载寄存器里。这个成员必须是一个介于 0x0000 和 0xFFFF 之间的数值。

成员 TIM_Prescaler

TIM_Prescaler 配置用来分割 TIM 时钟的预分频数。这个成员必须是一个介于 0x0000 和 0xFFFF 之间的数值。

成员 TIM_ClockDivision

TIM_ClockDivision 配置时钟分割。这个成员可以被设置为如下值：

表 461. TIM_ClockDivision 定义

TIM_ClockDivision	描述
TIM_CKD_DIV1	TDTS = Tck_tim
TIM_CKD_DIV2	TDTS = 2Tck_tim
TIM_CKD_DIV4	TDTS = 4Tck_tim

成员 TIM_CounterMode

TIM_CounterMode 选择计数器模式。这个成员可以被设置为如下值：

表 462. TIM_CounterMode 定义

TIM_ CounterMode	描述
TIM_Counter_Up	TIM 向上计数模式
TIM_Counter_Down	TIM 向下计数模式
TIM_Counter_CenterAligned1	TIM 中心对齐模式 1 计数模式
TIM_Counter_CenterAligned2	TIM 中心对齐模式 2 计数模式
TIM_Counter_CenterAligned3	TIM 中心对齐模式 3 计数模式

19.2.3 TIM_OCInit 函数

表 463 描述了 TIM_OCInit 函数。

表 463. TIM_OCInit 函数

函数名	TIM_ OCInit
函数原型	void TIM_OCInit(TIM_TypeDef* TIMx, TIM_OCInitTypeDef* TIM_OCInitStruct)
行为描述	根据 TIM_ OCInitStruct 中的特定参数初始化 TIMx 外设
输入参数 1	TIMx: 此处 x 可以是 2，3 或 4 以选择 TIM 外设
输入参数 2	TIM_ OCInitStruct: 指向一个包含特定 TIMx 外设的配置信息的

	TIM_ OCInitTypeDef 结构的指针 参考 TIM_ InitTypeDef 结构 获取该参数允许值的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

TIM_InitTypeDef 结构

TIM_OCInitTypeDef 结构在文件 *stm32f10x_tim.h* 中定义如下：

```
typedef struct
{
u16 TIM_OCMode;
u16 TIM_Channel;
u16 TIM_Pulse;
u16 TIM_OCPolarity;
} TIM_OCInitTypeDef;
```

成员 TIM_OCMode

TIM_OCMode 选择计时器模式。这个成员可以被设置为如下值：

表 464. TIM_OCMode 定义

TIM_ OCMode	描述
TIM_OCMode_Timing	TIM 输出比较计时模式

TIM_OCMode_Active	TIM 输出比较有效模式
TIM_OCMode_Inactive	TIM 输出比较无效模式
TIM_OCMode_Toggle	TIM 输出比较锁定模式
TIM_OCMode_PWM1	TIM 脉冲宽度调制模式 1
TIM_OCMode_PWM2	TIM 脉冲宽度调制模式 2

成员 TIM_Channel

TIM_Channel 选择通道。这个成员可以被设置为如下值：

表 465. TIM_Channel 定义

TIM_ Channel	描述
TIM_Channel_1	通道 1 被使用
TIM_Channel_2	通道 2 被使用
TIM_Channel_3	通道 3 被使用
TIM_Channel_4	通道 4 被使用

成员 TIM_Pulse

TIM_Pulse 配置将被加载在捕获比较寄存器中的脉冲值。这个成员必须是一个介于 0x0000 和 0xFFFF 之间的数值。

成员 TIM_OCPolarity

TIM_OCPolarity 配置输出极性。这个成员可以被设置为如下值：

表 466. TIM_OCPolarity 定义

TIM_ OCPolarity	描述
TIM_OCPolarity_High	TIM 输出比较极为高
TIM_OCPolarity_Low	TIM 输出比较极为低

例：

```
/* 配置 TIM2 的通道 1 为 PWM 模式*/

TIM_OCInitTypeDef TIM_OCStructure;

TIM_BaseInitTypeDef TIM_TimeBase;

TIM_TimeBase.TIM_Period = 0xFFFF;

TIM_TimeBase.TIM_Prescaler = 0xF;

TIM_TimeBase.TIM_ClockDivision = 0x0;

TIM_TimeBase.TIM_CounterMode = TIM_CounterMode_Up;

TIM_TimeBaseInit(TIM2, & TIM_TimeBase);

TIM_Structure.TIM_OCMode = TIM_OCMode_PWM1;

TIM_Structure.TIM_Channel = TIM_Channel_1;

TIM_Structure.TIM_Pulse = 0x3FFF;

TIM_Structure.TIM_OCPolarity = TIM_OCPolarity_High;

TIM_OCInit(TIM2, & TIM_OCStructure);
```

19.2.4 TIM_ICInit 函数

表 467 描述了 TIM_ICInit 函数。

表 467. TIM_ICInit 函数

函数名	TIM_ ICInit
函数原型	void TIM_ICInit(TIM_TypeDef* TIMx, TIM_ICInitTypeDef* TIM_ICInitStruct)
行为描述	根据 TIM_ ICInitStruct 中的特定参数初始化 TIMx
输入参数 1	TIMx: 此处 x 可以是 2，3 或 4 以选择外设
输入参数 2	TIM_ ICInitStruct: 指向一个包含特定 TIM 外设的配置信息的 TIM_ ICInitTypeDef 结构的指针 参考 TIM_ ICInitTypeDef 结构 获取该参数允许值的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

TIM_ICInitTypeDef 结构

TIM_ICInitTypeDef 结构在文件 *stm32f10x_tim.h* 中定义如下：

```
typedef struct
{
u16 TIM_ICMode;
u16 TIM_Channel;
u16 TIM_ICPolarity;
u16 TIM_ICSelection;
u16 TIM_ICPrescaler;
u16 TIM_ICFilter;
} TIM_ICInitTypeDef;
```

成员 TIM_ICMode

TIM_ICMode 选择 TIM 输入捕获模式。这个成员可以被设置为如下值：

表 468. TIL_ICMode definition

TIM_ ICMode	描述
TIM_ ICMode_ICAP	TIM 在输入捕获模式中使用
TIM_ ICMode_PWMI	TIM 在 PWM 模式中使用

成员 TIM_Channel

TIM_Channel 选择通道。这个成员可以被设置为如下值：

表 469. TIM_Channel 定义

TIM_ Channel	描述
TIM_Channel_1	通道 1 被使用

TIM_Channel_2	通道 2 被使用
TIM_Channel_3	通道 3 被使用
TIM_Channel_4	通道 4 被使用

成员 TIM_ICPolarity

TIM_ICPolarity 配置输入信号的有效边沿。这个成员可以被设置为如下值：

表 470. TIM_ICPolarity 定义

TIM_ ICPolarity	描述
TIM_ICPolarity_Rising	TIM 输入捕获上升沿
TIM_ICPolarity_Falling	TIM 输入捕获下降沿

成员 TIM_ICSelection

TIM_ICSelection 选择输入。这个成员可以被设置为如下值：

表 471. TIM_ICSelection 定义

TIM_ ICSelection	描述
TIM_ICSelection_DirectTI	TIM 输入 2 ,3 或 4 被选中分别连接到 IC1 或 IC2 或 IC3 或 IC4
TIM_ICSelection_IndirectTI	TIM 输入 2 ,3 或 4 被选中分别连接到 IC2 或 IC1 或 IC4 或 IC3
TIM_ICSelection_TRGI	输入 2，3 或 4 被选中连接到 TRGI

成员 TIM_ICPrescaler

TIM_ICPrescaler 配置输入捕获预分频数。这个成员可以被设置为如下值之一：

表 472. TIM_ICPrescaler 定义

TIM_ICPrescaler	描述
TIM_ICPSC_DIV1	每检测到捕获输入的一个边沿 TIM 捕获执行一次
TIM_ICPSC_DIV2	每两次事件 TIM 捕获执行一次
TIM_ICPSC_DIV3	每四次事件 TIM 捕获执行一次
TIM_ICPSC_DIV4	每八次事件 TIM 捕获执行一次

成员 TIM_ICFilter

TIM_ICFilter 选择输入捕获过滤器。这个成员可以是 0x0 和 0xF 之间的数值。

例：

```
/* 下面的例子介绍了怎样配置 TIM2 为 PWM 输入模式。外部信号被连接到 TIM2 CH1 针脚，上升沿被用作有效边沿，TIM2 CCR1 被用作计算频率值，TIM2 CCR2 被用作计算占空比 */
TIM_DeInit(TIM2);
TIM_ICStructInit(&TIM_ICInitStructure);
TIM_ICInitStructure.TIM_ICMode = TIM_ICMode_PWMI;
TIM_ICInitStructure.TIM_Channel = TIM_Channel_1;
```

```
TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Rising;
TIM_ICInitStructure.TIM_ICSelection = TIM_ICSelection_DirectTI;
TIM_ICInitStructure.TIM_ICPrescaler = TIM_ICPSC_DIV1;
TIM_ICInitStructure.TIM_ICFilter = 0x0;
TIM_ICInit(TIM2, &TIM_ICInitStructure);
```

19.2.5 TIM_TimeBaseStructInit 函数

表 473 描述了 TIM_TimeBaseStructInit 函数。

表 473. TIM_TimeBaseStructInit 函数

函数名	TIM_ TimeBaseStructInit
函数原型	void TIM_TimeBaseStructInit(TIM_TimeBaseInitTypeDef* TIM_TimeBaseInitStruct)
行为描述	使用缺省值填充 TIM_TimeBaseInitStruct 的每一个成员
输入参数	TIM_TimeBaseInitStruct: 指向一个将被初始化的 TIM_ TimeBaseInitTypeDef 结构的指针
输出参数	无
返回值	无
前提条件	无
调用函数	无

TIM_TimeBaseInitStruct 成员有下列缺省值：

表 474. TIM_TimeBaseInitStruct 缺省值

成员	缺省值
TIM_Period	TIM_Period_Reset_Mask
TIM_Prescaler	TIM_Prescaler_Reset_Mask
TIM_CKD	TIM_CKD_DIV1
TIM_CounterMode	TIM_CounterMode_Up

例：

```
/* 下面的例子介绍了怎样初始化一个 TIM_BaseInitTypeDef 结构 */
TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure;
TIM_TimeBaseStructInit(& TIM_TimeBaseInitStructure);
```

19.2.6 TIM_OCStructInit 函数

表 475 描述了 TIM_OCStructInit 函数。

表 475. TIM_OCStructInit 函数

函数名	TIM_ OCStructInit

函数原型	void TIM_OCStructInit(TIM_OCInitTypeDef* TIM_OCInitStruct)
行为描述	使用缺省值填充 TIM_ OCInitStruct 的每一个成员
输入参数	TIM_OCInitStruct: 指向一个将被初始化的 TIM_OCInitStruct 结构的指针
输出参数	无
返回值	无
前提条件	无
调用函数	无

TIM_OCInitStruct 成员有下列缺省值：

表 476. TIM_OCInitStruct 缺省值

成员	缺省值
TIM_ OCMode	TIM_ OCMode_Timing
TIM_ Channel	TIM_Channel_1
TIM_ Pulse	TIM_Pulse_Reset_Mask
TIM_ OCPolarity	TIM_OCPolarity_High

例：

```
/* 下面的例子介绍了怎样初始化一个 TIM_OCInitTypeDef 结构 */

TIM_OCInitTypeDef TIM_OCInitStructure;
```

TIM_OCStructInit(& TIM_OCInitStructure);

19.2.7 TIM_ICStructInit 函数

表 475 描述了 TIM_ICStructInit 函数。

表 477. TIM_ICStructInit 函数

函数名	TIM_ ICStructInit
函数原型	void TIM_ICStructInit(TIM_ICInitTypeDef* TIM_ICInitStruct)
行为描述	使用缺省值填充 TIM_ICInitStruct 的每一个成员
输入参数	TIM_ICInitStruct: 指向一个将被初始化的 TIM_ICInitStruct 结构的指针
输出参数	无
返回值	无
前提条件	无
调用函数	无

TIM_ICInitStruct 成员有下列缺省值：

表 478. TIM_ICInitStruct 缺省值

成员	缺省值

TIM_ICMode	TIM_ICMode_ICAP
TIM_Channel	TIM_Channel_1
TIM_ICPolarity	TIM_ICPolarity_Rising
TIM_ICSelection	TIM_ICSelection_DirectTI
TIM_ICPrescaler	TIM_ICPSC_DIV1
TIM_ICFilter	TIM_ICFilter_Mask

例：

```
/* 下面的例子介绍了怎样初始化一个 TIM_ICInitTypeDef 结构 */

TIM_ICInitTypeDef TIM_ICInitStructure;

TIM_ICStructInit(& TIM_ICInitStructure);
```

19.2.8 TIM_Cmd 函数

表 479 描述了 TIM_Cmd 函数。

表 479. TIM_Cmd 函数

函数名	TIM_ Cmd
函数原型	void TIM_Cmd(TIM_TypeDef* TIMx, FunctionalState NewState)
行为描述	使能或关闭特定的 TIM 外设
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择外设

输入参数 2	NewState: 此 TIM 外设的新状态，该参数可以是 ENABLE 或 DISABLE。
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 使能 TIM2 计时器*/
TIM_Cmd(TIM2, ENABLE);
```

19.2.9 TIM_ITConfig 函数

表 480 describes the TIM_ITConfig 函数.

表 480. TIM_ITConfig 函数

函数名	TIM_ ITConfig
函数原型	void TIM_ITConfig(TIM_TypeDef* TIMx, u16 TIM_IT, FunctionalState NewState)
行为描述	使能或关闭特定的 TIM 中断
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择外设

输入参数 2	TIM_IT: 将被使能或关闭的 TIM 中断源 参考 *TIM_IT* 部分获取对该参数允许值的更多细节
输入参数 3	NewState: 特定 TIM 中断的新状态，该参数可以是 ENABLE 或 DISABLE。
输出参数	无
返回值	无
前提条件	无
调用函数	无

参数 TIM_IT

TIM_IT 将被使能或关闭的 TIM 中断。它可以使用下列值的一个或多个:

表 481. TIM_IT 的允许值

TIM_IT	缺省值
TIM_IT_Update	TIM 更新中断源
TIM_IT_CC1	TIM 捕获/比较 1 中断源
TIM_IT_CC2	TIM 捕获/比较 2 中断源
TIM_IT_CC3	TIM 捕获/比较 3 中断源
TIM_IT_CC4	TIM 捕获/比较 4 中断源
TIM_IT_Trigger	TIM 触发器中断源

例：

```
/* 使能 TIM2 捕获比较通道 1 中断源 */
TIM_ITConfig(TIM2, TIM_IT_CC1, ENABLE );
```

19.2.10 TIM_DMAConfig 函数

表 482 描述了 TIM_DMAConfig 函数。

表 482. TIM_DMAConfig function

函数名	TIM_DMAConfig
函数原型	void TIM_DMAConfig(TIM_TypeDef* TIMx,u8 TIM_DMABase, u16 TIM_DMABurstLength)
行为描述	配置某个 TIM 的 DMA 接口
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择外设
输入参数 2	TIM_DMABase: DMA 基址 参考 *TIM_ DMABase* 部分获取对该参数允许值的更多细节
输入参数 3	TIM_DMABurstLength: DMA 猝发长度 参考 TIM_ DMABurstLength 部分获取对该参数允许值的更多细节
输出参数	无
返回值	无
前提条件	无

调用函数	无

参数 TIM_DMABase

TIM_DMABase 选择 TIM DMA 的基址。它可以被设置为下列值之一：

表 483. TIM_DMABase 的允许值

TIM_ DMABase	描述
TIM_DMABase_CR1	TIM CR1 寄存器用作 DMA 的基
TIM_DMABase_CR2	TIM CR2 寄存器用作 DMA 的基
TIM_DMABase_SMCR	TIM SMCR 寄存器用作 DMA 的基
TIM_DMABase_DIER	TIM DIER 寄存器用作 DMA 的基
TIM_DMABase_SR	TIM SR 寄存器用作 DMA 的基
TIM_DMABase_EGR	TIM EGR 寄存器用作 DMA 的基
TIM_DMABase_CCMR1	TIM CCMR1 寄存器用作 DMA 的基
TIM_DMABase_CCMR2	TIM CCMR2 寄存器用作 DMA 的基
TIM_DMABase_CCER	TIM CCER 寄存器用作 DMA 的基
TIM_DMABase_CNT	TIM CNT 寄存器用作 DMA 的基
TIM_DMABase_PSC	TIM PSC 寄存器用作 DMA 的基
TIM_DMABase_ARR	TIM ARR 寄存器用作 DMA 的基
TIM_DMABase_CCR1	TIM CCR1 寄存器用作 DMA 的基
TIM_DMABase_CCR2	TIM CCR2 寄存器用作 DMA 的基

TIM_DMABase_CCR3	TIM CCR3 寄存器用作 DMA 的基
TIM_DMABase_CCR4	TIM CCR4 寄存器用作 DMA 的基
TIM_DMABase_DCR	TIM DCR 寄存器用作 DMA 的基

参数 TIM_DMABurstLength

TIM_DMABurstLength 选择 TIM DMA 猝发长度. 参考表 484 获取该参数的允许值：

表 484. TIM_DMABurstLength 允许值

TIM_ DMABurstLength	描述
TIM_DMABurstLength_1Byte	TIM DMA 猝发长度为 1 字节
TIM_DMABurstLength_2Byte	TIM DMA 猝发长度为 2 字节
TIM_DMABurstLength_3Byte	TIM DMA 猝发长度为 3 字节
TIM_DMABurstLength_4Byte	TIM DMA 猝发长度为 4 字节
TIM_DMABurstLength_5Byte	TIM DMA 猝发长度为 5 字节
TIM_DMABurstLength_6Byte	TIM DMA 猝发长度为 6 字节
TIM_DMABurstLength_7Byte	TIM DMA 猝发长度为 7 字节
TIM_DMABurstLength_8Byte	TIM DMA 猝发长度为 8 字节
TIM_DMABurstLength_9Byte	TIM DMA 猝发长度为 9 字节
TIM_DMABurstLength_10Byte	TIM DMA 猝发长度为 10 字节
TIM_DMABurstLength_11Byte	TIM DMA 猝发长度为 11 字节
TIM_DMABurstLength_12Byte	TIM DMA 猝发长度为 12 字节

TIM_DMABurstLength_13Byte	TIM DMA 猝发长度为 13 字节
TIM_DMABurstLength_14Byte	TIM DMA 猝发长度为 14 字节
TIM_DMABurstLength_15Byte	TIM DMA 猝发长度为 15 字节
TIM_DMABurstLength_16Byte	TIM DMA 猝发长度为 16 字节
TIM_DMABurstLength_17Byte	TIM DMA 猝发长度为 17 字节
TIM_DMABurstLength_18Byte	TIM DMA 猝发长度为 18 字节

例：

/* 配置 TIM2 DMA 接口，传输 1 个字节且使用 CCR1 寄存器作为基址 */

TIM_DMAConfig(TIM2, TIM_DMABase_CCR1, TIM_DMABurstLength_1Byte)

19.2.11 TIM_DMACmd 函数

表 485 描述了 TIM_DMACmd 函数。

表 485. TIM_DMACmd 函数

函数名	TIM_ DMACmd
函数原型	void TIM_DMACmd(TIM_TypeDef* TIMx, u16 TIM_DMASource, FunctionalState Newstate)
行为描述	使能或关闭 TIMx 的 DMA 请求

输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择外设
输入参数 2	TIM_DMASource: DMA 请求源 参考 *TIM_ DMASource* 部分获取对该参数允许值的更多细节
输入参数 3	NewState: 此 TIM DMA 请求源的新状态，该参数可以是 ENABLE 或 DISABLE。
输出参数	无
返回值	无
前提条件	无
调用函数	无

参数 TIM_DMASource

TIM_DMASource 选择 DMA 请求源，它可以是下列值中的一个或多个：

表 486. TIM_DMASource 的允许值

TIM_ DMASource	描述
TIM_DMA_Update	TIM 更新 DMA 源
TIM_DMA_CC1	TIM 捕获/比较 1DMA 源
TIM_DMA_CC2	TIM 捕获/比较 2DMA 源
TIM_DMA_CC3	TIM 捕获/比较 3DMA 源
TIM_DMA_CC4	TIM 捕获/比较 4DMA 源
TIM_DMA_Trigger	TIM 触发器 DMA 源

例：

/* TIM2 捕获比较 1 的 DMA 请求配置 */

TIM_DMACmd(TIM2, TIM_DMA_CC1, ENABLE);

19.2.12 TIM_InternalClockConfig 函数

表 487 描述了 TIM_InternalClockConfig 函数。

表 487. TIM_InternalClockConfig 函数

函数名	TIM_InternalClockConfig
函数原型	void TIM_InternalClockConfig(TIM_TypeDef* TIMx)
行为描述	配置 TIM 的内部时钟
输入参数	TIMx：此处 x 可以是 2，3 或 4 以选择外设
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

/* 为 TIM2 选择内部时钟 */

TIM_InternalClockConfig(TIM2);

19.2.13 TIM_ITRxExternalClockConfig 函数

表 488 描述了 TIM_ITRxExternalClockConfig 函数。

表 488. TIM_ITRxExternalClockConfig 函数

函数名	TIM_ ITRxExternalClockConfig
函数原型	void TIM_ITRxExternalClockConfig(TIM_TypeDef* TIMx, u16 TIM_InputTriggerSource)
行为描述	配置 TIM 的内部触发器作为外部时钟
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择外设
输入参数 2	TIM_InputTriggerSource: 输入触发源 参考 TIM_ InputTriggerSource 部分获取对该参数允许值的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

参数 TIM_InputTriggerSource

TIM_InputTriggerSource 选择 TIM 输入触发器。参考表 489 获取该参数的允许值。

表 489. TIM_InputTriggerSource 允许值

TIM_ InputTriggerSource	描述
TIM_TS_ITR0	TIM 内部触发器 0
TIM_TS_ITR1	TIM 内部触发器 1
TIM_TS_ITR2	TIM 内部触发器 2
TIM_TS_ITR3	TIM 内部触发器 3

例：

```
/* TIM2 内部触发器 3 用作时钟源 */

TIM_ITRxExternalClockConfig(TIM2, TIM_TS_ITR3);
```

19.2.14 TIM_TIxExternalClockConfig 函数

表 490 描述了 TIM_TIxExternalClockConfig 函数。

表 490. TIM_TIxExternalClockConfig 函数

函数名	TIM_ TIxExternalClockConfig
函数原型	void TIM_TIxExternalClockConfig(TIM_TypeDef* TIMx, u16 TIM_TIxExternalCLKSource, u8 TIM_ICPolarity, u8 ICFilter)
行为描述	配置 TIM 的触发器作为外部时钟

输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择外设
输入参数 2	TIM_ TIxExternalCLKSource: 触发器源 参考 TIM_ TIxExternalCLKSource 部分获取对该参数允许值的更多细节
输入参数 3	TIM_ICPolarity: 指定 TI 的极性 参考 *TIM_ ICPolarity* 部分获取对该参数允许值的更多细节
输入参数 4	ICFilter: 指定输入捕获过滤器。该参数可以是 0x0 到 0xF 之间的值。
输出参数	无
返回值	无
前提条件	无
调用函数	无

参数 TIM_TIxExternalCLKSource

TIM_TIxExternalCLKSource 选择 TIM TIx 作为外部时钟源。它可以是下列值的一个或几个：

表 491. TIM_TIxExternalCLKSource 允许值

TIM_ TIxExternalCLKSource	描述
TIM_TS_TI1FP1	TIM IC1 被映射到 T11
TIM_TS_TI2FP2	TIM IC2 被映射到 T12
TIM_TS_TI1F_ED	TIM IC1 被映射到 T11，边沿检测被使用

Example:

/* 选择 TI1 作为 TIM2 的时钟：外部时钟被连接到 TI1 输入端，上升沿被用作有效边沿，且没有过滤器采样被使用 (ICFilter = 0) */

TIM_TIxExternalClockConfig(TIM2, TIM_TS_TI1FP1,

TIM_ICPolarity_Rising, 0);

19.2.15 TIM_ETRClockMode1Config 函数

表 492 describes the TIM_ETRClockMode1Config 函数。

表 492. TIM_ETRClockMode1Config 函数

函数名	TIM_ ETRClockMode1Config
函数原型	void TIM_ETRClockMode1Config(TIM_TypeDef* TIMx, u16 TIM_ExtTRGPrescaler, u16 TIM_ExtTRGPolarity, u16 ExtTRGFilter)
行为描述	配置 TIM 外部时钟为模式 1
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择外设
输入参数 2	TIM_ExtTRGPrescaler: 外部触发器预分频数 参考 TIM_ ExtTRGPrescaler 部分获取对该参数允许值的更多细节
输入参数 3	TIM_ExtTRGPolarity: 外部触发器极性 参考 TIM_ ExtTRGPolarity 部分获取对该参数允许值的更多细节
输入参数 4	ExtTRGFilter: 外部触发器过滤器。该参数可以是 0x0 到 0xF 之间的

	值。
输出参数	无
返回值	无
前提条件	无
调用函数	无

参数 TIM_ExtTRGPrescaler

TIM_ExtTRGPrescaler 配置某个 TIM 外部触发器的预分频数。它可以被设置为下列值之一：

表 493. TIM_ExtTRGPrescaler 的允许值

TIM_ ExtTRGPrescaler	描述
TIM_ExtTRGPSC_OFF	TIM ETRP 的预分频数关闭
TIM_ExtTRGPSC_DIV2	TIM ETRP 频率除以 2
TIM_ExtTRGPSC_DIV4	TIM ETRP 频率除以 4
TIM_ExtTRGPSC_DIV8	TIM ETRP 频率除以 8

参数 TIM_ExtTRGPolarity

TIM_ExtTRGPolarity 配置 TIM 外部触发器的极性。它可以被设置为下列值之一：

表 494. TIM_ExtTRGPolarity 允许值

TIM_ ExtTRGPolarity	描述
TIM_ExtTRGPolarity_Inverted	TIM 外部触发器极性倒转：低有效或下降沿有效

TIM_ExtTRGPolarity_NonInverted	TIM 外部触发器极性不倒转：高有效或上升沿有效

例：

/* 选择 TIM2 的外部时钟模式 1: 外部时钟连接到 ETR 输入端，上升沿被用作有效边沿，且没有过滤器采样被使用(ExtTRGFilter = 0) 且预分频数固定为 TIM_ExtTRGPSC_DIV2 */

TIM_ExternalCLK1Config(TIM2, TIM_ExtTRGPSC_DIV2, TIM_ExtTRGPolarity_NonInverted, 0x0);

19.2.16 TIM_ETRClockMode2Config 函数

表 495 描述了 TIM_ETRClockMode2Config 函数。

表 495. TIM_ETRClockMode2Config 函数

函数名	TIM_ ETRClockMode2Config
函数原型	void TIM_ETRClockMode2Config(TIM_TypeDef* TIMx, u16 TIM_ExtTRGPrescaler, u16 TIM_ExtTRGPolarity, u16 ExtTRGFilter)
行为描述	配置 TIMx 外部时钟为模式 2
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择外设
输入参数 2	TIM_ExtTRGPrescaler: 外部触发器预分频数 参考 TIM_ ExtTRGPrescaler 部分获取对该参数允许值的更多细节

输入参数 3	TIM_ExtTRGPolarity: 外部触发器极性 参考 TIM_ ExtTRGPolarity 部分获取对该参数允许值的更多细节
输入参数 4	ExtTRGFilter: 外部触发器过滤器。该参数可以是 0x0 到 0xF 之间的值。
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

/* TIM2 选择外部时钟模式 2: 外部时钟连接到 ETR 输入端，上升沿被用作有效边沿，且没有过滤器采样被使用(ExtTRGFilter = 0) 且预分频数固定为 TIM_ExtTRGPSC_DIV2 */

TIM_ExternalCLK2Config(TIM2, TIM_ExtTRGPSC_DIV2, TIM_ExtTRGPolarity_NonInverted, 0x0);

19.2.17 TIM_ETRConfig 函数

表 495 描述了 TIM_ETRConfig 函数。

表 496. TIM_ETRConfig 函数

函数名	TIM_ ETRConfig
函数原型	void TIM_ETRConfig(TIM_TypeDef* TIMx, u16 TIM_ExtTRGPrescaler, u16 TIM_ExtTRGPolarity, u8 ExtTRGFilter)
行为描述	配置 TIM 外部时钟为模式 2
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择外设
输入参数 2	TIM_ExtTRGPrescaler: 外部触发器预分频数 参考 TIM_ ExtTRGPrescaler 部分获取对该参数允许值的更多细节
输入参数 3	TIM_ExtTRGPolarity: 外部触发器极性 参考 TIM_ ExtTRGPolarity 部分获取对该参数允许值的更多细节
输入参数 4	ExtTRGFilter: 外部触发器过滤器。该参数可以是 0x0 到 0xF 之间的值。
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 为 TIM2 配置外部触发器:上升沿被用作有效边沿 ,且没有过滤器采样被使用(ExtTRGFilter = 0) 且预分频数固定为 TIM_ExtTRGPSC_DIV2 */
TIM_ExternalCLK2Config(TIM2, TIM_ExtTRGPSC_DIV2, TIM_ExtTRGPolarity_NonInverted, 0x0);
```

19.2.18 TIM_SelectInputTrigger 函数

表 497 描述了 TIM_SelectInputTrigger 函数。

表 497. TIM_SelectInputTrigger 函数

函数名	TIM_ SelectInputTrigger
函数原型	void TIM_SelectInputTrigger(TIM_TypeDef* TIMx, u16 TIM_InputTriggerSource)
行为描述	选择某个 TIM 的输入触发器源
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输入参数 2	TIM_InputTriggerSource: 输入触发源 参考 TIM_ InputTriggerSource 部分获取对该参数允许值的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

参数 TIM_InputTriggerSource

TIM_InputTriggerSource 选择输入触发器源。这个参数可以设为下列值之一：

表 498. TIM_InputTriggerSource 的允许值

TIM_ InputTriggerSource	描述
TIM_TS_ITR0	TIM 内部触发器 0
TIM_TS_ITR1	TIM 内部触发器 1
TIM_TS_ITR2	TIM 内部触发器 2
TIM_TS_ITR3	TIM 内部触发器 3
TIM_TS_TI1F_ED	TIM TI1 边沿探测器
TIM_TS_TI1FP1	TIM 过滤计时器输入 1
TIM_TS_TI1FP2	TIM 过滤计时器输入 2
TIM_TS_ETRF	TIM 外部触发器输入

例：

```
/* 选择内部触发器 3 作为 TIM2 的输入触发器 */
void TIM_SelectInputTrigger(TIM2, TIM_TS_ITR3);
```

19.2.19 TIM_PrescalerConfig 函数

表 499 描述了 TIM_PrescalerConfig 函数。

表 499. TIM_PrescalerConfig 函数

函数名	TIM_ PrescalerConfig
函数原型	void TIM_PrescalerConfig(TIM_TypeDef* TIMx, u16 Prescaler, u16 TIM_PSCReloadMode)
行为描述	配置 TIM 的预分频数
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择外设
输入参数 2	Prescaler: TIMx 预分频数的新值
输入参数 3	TIM_PSCReloadMode: TIM 预分频数重载模式 参考 *TIM_ PSCReloadMode* 部分获取对该参数允许值的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

参数 TIM_PSCReloadMode

TIM_PSCReloadMode 选择 TIM 预分频数重载模式。这个参数可以设为下列值之一：

表 500. TIM_PSCReloadMode 的允许值

TIM_ PSCReloadMode	描述
TIM_PSCReloadMode_Update	发生更新事件时预分频数载入

TIM_PSCReloadMode_Immediate	预分频数立即载入

Example:

```
/*为 TIM2 的预分频数设置新值 */
u16 TIMPrescaler = 0xFF00;
TIM_PrescalerConfig(TIM2, TIMPrescaler,
TIM_PSCReloadMode_Immediate);
```

19.2.20 TIM_CounterModeConfig 函数

表 501 描述了 TIM_CounterModeConfig 函数.

表 501. TIM_CounterModeConfig 函数

函数名	TIM_ CounterModeConfig
函数原型	void TIM_CounterModeConfig(TIM_TypeDef* TIMx, u16 TIM_CounterMode)
行为描述	指定 TIM 的计数模式
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择外设
输入参数 2	TIM_CounterMode: 将使用的计数模式 参考 *TIM_ CounterMode* 部分获取对该参数允许值的更多细节

输出参数	无
返回值	无
前提条件	无
调用函数	无

Example:

/* 为 TIM2 选择中心对齐模式 1 */

TIM_CounterModeConfig(TIM2, TIM_Counter_CenterAligned1);

19.2.21 TIM_ForcedOC1Config 函数

表 502 描述了 TIM_ForcedOC1Config 函数

表 502. TIM_ForcedOC1Config 函数

函数名	TIM_ ForcedOC1Config
函数原型	void TIM_ForcedOC1Config(TIM_TypeDef* TIMx, u16 TIM_ForcedAction)
行为描述	强制某个 TIM 输出 1 的信号到有效或无效的电平
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输入参数 2	TIM_ ForcedAction: 作用在输出信号上的动作 参考 *TIM_ ForcedAction* 部分获取对该参数允许值的更多细节

输出参数	无
返回值	无
前提条件	无
调用函数	无

参数 TIM_ForcedAction

强制行为列举如下：

表 503. TIM_ForcedAction 的允许值

TIM_ForcedAction	描述
TIM_ForcedAction_Active	在 OCxREF 强制为有效电平
TIM_ForcedAction_InActive	在 OCxREF 强制为无效电平

例：

```
/* 强制 TIM2 输出比较 1 为有效电平 */

TIM_ForcedOC1Config(TIM2, TIM_ForcedAction_Active);
```

19.2.22 TIM_ForcedOC2Config 函数

表 504 描述了 TIM_ForcedOC2Config 函数

表 504. TIM_ForcedOC3Config 函数

函数名	TIM_ ForcedOC2Config
函数原型	void TIM_ForcedOC2Config(TIM_TypeDef* TIMx, u16 TIM_ForcedAction)
行为描述	强制 TIMx 输出 2 的信号到有效或无效的电平
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择外设
输入参数 2	TIM_ ForcedAction: 作用在输出信号上的动作 参考 *TIM_ ForcedAction* 部分获取对该参数允许值的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 强制 TIM2 输出比较 2 为有效电平 */
TIM_ForcedOC2Config(TIM2, TIM_ForcedAction_Active);
```

19.2.23 TIM_ForcedOC3Config 函数

表 505 描述了 TIM_ForcedOC3Config 函数

表 505. TIM_ForcedOC3Config 函数

函数名	TIM_ ForcedOC3Config
函数原型	void TIM_ForcedOC3Config(TIM_TypeDef* TIMx, u16 TIM_ForcedAction)
行为描述	强制某个 TIM 输出 3 的信号到有效或无效的电平
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输入参数 2	TIM_ ForcedAction: 作用在输出信号上的动作 参考 *TIM_ ForcedAction* 部分获取对该参数允许值的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 强制 TIM2 输出比较 3 为有效电平*/
TIM_ForcedOC3Config(TIM2, TIM_ForcedAction_Active);
```

19.2.24 TIM_ForcedOC4Config 函数

表 506 描述了 TIM_ForcedOC4Config 函数

表 506. TIM_ForcedOC4Config 函数

函数名	TIM_ ForcedOC4Config
函数原型	void TIM_ForcedOC4Config(TIM_TypeDef* TIMx, u16 TIM_ForcedAction)
行为描述	强制某个 TIM 输出 4 的信号到有效或无效的电平
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输入参数 2	TIM_ ForcedAction: 作用在输出信号上的动作 参考 *TIM_ ForcedAction* 部分获取对该参数允许值的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 强制 TIM2 输出比较 4 为有效电平 */
TIM_ForcedOC4Config(TIM2, TIM_ForcedAction_Active);
```

19.2.25 TIM_ARRPreloadConfig 函数

表 507 描述了 TIM_ARRPreloadConfig 函数。

表 507. TIM_ARRPreloadConfig 函数

函数名	TIM_ ARRPreloadConfig
函数原型	void TIM_ARRPreloadConfig(TIM_TypeDef* TIMx, FunctionalState Newstate)
行为描述	使能或关闭某个 TIM 外设在 ARR 寄存器上的预载入寄存器
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择外设
输入参数 2	NewState: 在 TIMx_CR1 寄存器中的 ARPE 位的新状态 这个参数可以是 ENABLE 或 DISABLE
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 使能 TIM2 在 ARR 寄存器上的预载入 */
TIM_ARRPreloadConfig(TIM2, ENABLE);
```

19.2.26 TIM_SelectCCDMA 函数

表 508 描述了 TIM_SelectCCDMA 函数。

表 508. TIM_SelectCCDMA 函数

函数名	TIM_ SelectCCDMA
函数原型	void TIM_SelectCCDMA(TIM_TypeDef* TIMx, FunctionalState Newstate)
行为描述	选择 TIM 外设的捕获比较 DMA 源
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择外设
输入参数 2	NewState: 捕获比较 DMA 源的新状态 这个参数可以是 ENABLE 或 DISABLE
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 选择 TIM2 的捕获比较 DMA 源 */
TIM_SelectCCDMA(TIM2, ENABLE);
```

19.2.27 TIM_OC1PreloadConfig 函数

表 509 描述了 TIM_OC1PreloadConfig 函数。

表 509. TIM_OC1PreloadConfig 函数

函数名	TIM_ OC1PreloadConfig
函数原型	void TIM_OC1PreloadConfig(TIM_TypeDef* TIMx, u16 TIM_OCPreload)
行为描述	使能或关闭某个 TIM 在 CCR1 寄存器上的预载入寄存器
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择外设
输入参数 2	TIM_OCPreload: 输出比较预载入状态 参考 *TIM_ OCPreload* 部分获取对该参数允许值的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

参数 TIM_OCPreload

能被使能或关闭的输出比较预载入的状态列举如下：

表 510. TIM_OCPreload 状态

TIM_ OCPreload	描述
TIM_ OCPreload_Enable	使能某个 TIM 的在 CCR1 上的预载入寄存器
TIM_OCPreload_Disable	关闭某个 TIM 的在 CCR1 上的预载入寄存器

例：

```
/* 使能 TIM2 在 CC1 寄存器上的预载入 */

TIM_OC1PreloadConfig(TIM2, TIM_OCPreload_Enable);
```

19.2.28 TIM_OC2PreloadConfig 函数

表 511 描述了 TIM_OC2PreloadConfig 函数。

表 511. TIM_OC2PreloadConfig 函数

函数名	TIM_ OC2PreloadConfig
函数原型	void TIM_OC2PreloadConfig(TIM_TypeDef* TIMx, u16 TIM_OCPreload)
行为描述	使能或关闭某个 TIM 在 CCR2 寄存器上的预载入寄存器
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择外设
输入参数 2	TIM_OCPreload: 输出比较预载入状态 参考 *TIM_ OCPreload* 部分获取对该参数允许值的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 使能 TIM2 在 CC2 Register 的预载入 */

TIM_OC2PreloadConfig(TIM2, TIM_OCPreload_Enable);
```

19.2.29 TIM_OC3PreloadConfig 函数

表 512 描述了 TIM_OC3PreloadConfig 函数。

表 512. TIM_OC3PreloadConfig 函数

函数名	TIM_ OC3PreloadConfig
函数原型	void TIM_OC3PreloadConfig(TIM_TypeDef* TIMx, u16 TIM_OCPreload)
行为描述	使能或关闭某个 TIM 在 CCR3 寄存器上的预载入寄存器
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择外设
输入参数 2	TIM_OCPreload: 设定输出比较预载入状态 参考 *TIM_ OCPreload* 部分获取对该参数允许值的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 使能 TIM2 在 CC3 寄存器上的预载入 */

TIM_OC3PreloadConfig(TIM2, TIM_OCPreload_Enable);
```

19.2.30 TIM_OC4PreloadConfig 函数

表 513 描述了 TIM_OC4PreloadConfig 函数。

表 513. TIM_OC4PreloadConfig 函数

函数名	TIM_ OC4PreloadConfig
函数原型	void TIM_OC4PreloadConfig(TIM_TypeDef* TIMx, u16 TIM_OCPreload)
行为描述	使能或关闭某个 TIM 在 CCR4 寄存器上的预载入寄存器
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择外设
输入参数 2	TIM_OCPreload: 输出比较预载入状态 参考 *TIM_ OCPreload* 部分获取对该参数允许值的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 使能 TIM2 在 CC4 寄存器上的预载入 */

TIM_OC4PreloadConfig(TIM2, TIM_OCPreload_Enable);
```

19.2.31 TIM_OC1FastConfig 函数

表 514 描述了 TIM_OC1FastConfig 函数。

表 514. TIM_OC1FastConfig 函数

函数名	TIM_OC1FastConfig
函数原型	void TIM_OC1FastConfig(TIM_TypeDef* TIMx, u16 TIM_OCFast)
行为描述	配置某个 TIM 的输出比较 1 的快速特性
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择外设
输入参数 2	TIM_ OCFast: 输出比较快速特性的状态 参考 *TIM_ OCFast* 部分获取对该参数允许值的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

TIM_OCFast

可使用的输出比较预加载的状态列举如下：

表 515. TIM_OCFast 状态

TIM_ OCFast	描述
TIM_ OCFast_Enable	使能某个 TIM 的输出比较快速能力
TIM_ OCFast_Disable	关闭某个 TIM 的输出比较快速能力

例：

```
/* 在快速模式中使用 TIM2 的输出比较 1 */

TIM_OC1FastConfig(TIM2, TIM_OCFast_Enable);
```

19.2.32 TIM_OC2FastConfig 函数

*表 516*描述了 TIM_OC2FastConfig 函数。

表 516. TIM_OC2FastConfig 函数

函数名	TIM_OC2FastConfig
函数原型	void TIM_OC2FastConfig(TIM_TypeDef* TIMx, u16 TIM_OCFast)
行为描述	配置某个 TIM 的输出比较 2 的快速特性
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择外设
输入参数 2	TIM_ OCFast: 输出比较快速特性的状态 参考 *TIM_ OCFast* 部分获取对该参数允许值的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 在快速模式中使用 TIM2 的输出比较 2 */
TIM_OC2FastConfig(TIM2, TIM_OCFast_Enable);
```

19.2.33 TIM_OC3FastConfig 函数

*表 517*描述了 TIM_OC3FastConfig 函数。

表 517. TIM_OC3FastConfig 函数

函数名	TIM_OC3FastConfig
函数原型	void TIM_OC3FastConfig(TIM_TypeDef* TIMx, u16 TIM_OCFast)
行为描述	配置某个 TIM 的输出比较 3 的快速特性
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择外设
输入参数 2	TIM_ OCFast: 输出比较快速特性的状态 参考 *TIM_ OCFast* 部分获取对该参数允许值的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 在快速模式中使用 TIM2 的输出比较 3 */
TIM_OC3FastConfig(TIM2, TIM_OCFast_Enable);
```

19.2.34 TIM_OC4FastConfig 函数

表 518 描述了 TIM_OC4FastConfig 函数。

表 518. TIM_OC4FastConfig 函数

函数名	TIM_OC4FastConfig
函数原型	void TIM_OC4FastConfig(TIM_TypeDef* TIMx, u16 TIM_OCFast)
行为描述	配置某个 TIM 的输出比较 4 的快速特性
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择外设
输入参数 2	TIM_ OCFast: 输出比较快速特性的状态 参考 *TIM_ OCFast* 部分获取对该参数允许值的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 在快速模式中使用 TIM2 的输出比较 4 */
TIM_OC4FastConfig(TIM2, TIM_OCFast_Enable);
```

19.2.35 TIM_ClearOC1Ref 函数

表 519 描述了 TIM_ClearOC1Ref 函数。

表 519. TIM_ClearOC1Ref 函数

函数名	TIM_ ClearOC1Ref
函数原型	void TIM_ClearOC1Ref(TIM_TypeDef* TIMx, u16 TIM_OCClear)
行为描述	在一个外部事件发生时清除或安全保护 OCREF1 信号
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择外设
输入参数 2	TIM_OCClear: 输出比较清除使能位的新状态 参考 *TIM_ OCClear* 部分获取对该参数允许值的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

参数 TIM_OCClear

可用的输出比较参考清除位的值列举在表 520 中。

表 520. TIM_OCClear

TIM_ OCClear	描述
TIM_ OCClear _Enable	使能某个 TIM 的输出比较的清除

TIM_ OCClear _Disable	关闭某个 TIM 的输出比较的清除

例：

```
/* 使能 TIM2 通道 1 的输出比较参考清除位 */
TIM_ClearOC1Ref(TIM2, TIM_OCClear_Enable);
```

19.2.36 TIM_ClearOC2Ref 函数

表 520 描述了 TIM_ClearOC2Ref 函数。

表 520. TIM_ClearOC2Ref 函数

函数名	TIM_ ClearOC2Ref
函数原型	void TIM_ClearOC2Ref(TIM_TypeDef* TIMx, u16 TIM_OCClear)
行为描述	在一个外部事件发生时清除或安全保护 OCREF2 信号
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择外设
输入参数 2	TIM_OCClear: 输出比较清除使能位的新状态 参考 *TIM_ OCClear* 部分获取对该参数允许值的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 使能 TIM2 通道 2 的输出比较参考清除位 */

TIM_ClearOC2Ref(TIM2, TIM_OCClear_Enable);
```

19.2.37 TIM_ClearOC3Ref 函数

表 521 描述了 TIM_ClearOC3Ref 函数。

表 521. TIM_ClearOC3Ref 函数

函数名	TIM_ ClearOC3Ref
函数原型	void TIM_ClearOC3Ref(TIM_TypeDef* TIMx, u16 TIM_OCClear)
行为描述	在一个外部事件发生时清除或安全保护 OCREF3 信号
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输入参数 2	TIM_OCClear: 输出比较清除使能位的新状态 参考 *TIM_ OCClear* 部分获取对该参数允许值的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 使能 TIM2 通道 3 的输出比较参考清除位 */

TIM_ClearOC3Ref(TIM2, TIM_OCClear_Enable);
```

19.2.38 TIM_ClearOC4Ref 函数

表 522 描述了 TIM_ClearOC4Ref 函数。

表 522. TIM_ClearOC4Ref 函数

函数名	TIM_ ClearOC4Ref
函数原型	void TIM_ClearOC4Ref(TIM_TypeDef* TIMx, u16 TIM_OCClear)
行为描述	在一个外部事件发生时清除或安全保护 OCREF4 信号
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输入参数 2	TIM_OCClear: 输出比较清除使能位的新状态 参考 *TIM_ OCClear* 部分获取对该参数允许值的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 使能 TIM2 通道 4 的输出比较参考清除位 */

TIM_ClearOC4Ref(TIM2, TIM_OCClear_Enable);
```

19.2.39 TIM_UpdateDisableConfig 函数

表 524 描述了 TIM_UpdateDisableConfig 函数。

表 524. TIM_UpdateDisableConfig 函数

函数名	TIM_ UpdateDisableConfig
函数原型	void TIM_UpdateDisableConfig(TIM_TypeDef* TIMx, FunctionalState Newstate)
行为描述	使能或关闭 TIM 更新事件
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择外设
输入参数 2	NewState: TIMx_CR1 寄存器中 UDIS 位的新状态 这个参数可以是 ENABLE 或 DISABLE.
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 为 TIM2 使能更新事件 */

TIM_UpdateDisableConfig(TIM2, DISABLE);
```

19.2.40 TIM_EncoderInterfaceConfig 函数

表 525 描述了 TIM_EncoderInterfaceConfig 函数。

表 525. TIM_EncoderInterfaceConfig 函数

函数名	TIM_ EncoderInterfaceConfig
函数原型	void TIM_EncoderInterfaceConfig(TIM_TypeDef* TIMx, u8 TIM_EncoderMode, u8 TIM_IC1Polarity, u8 TIM_IC2Polarity)
行为描述	配置 TIM 的编码器接口
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输入参数 2	IM_EncoderMode: 编码器模式 参考 *TIM_EncoderMode* 部分获取该参数允许值的更多细节
输入参数 3	TIM_IC1Polarity: TI1 的极性 参考 *TIM_ICPolarity* 部分获取该参数允许值的更多细节
输入参数 4	TIM_IC2Polarity: TI2 的极性 参考 *TIM_ICPolarity* 部分获取该参数允许值的更多细节
输出参数	无
返回值	无

前提条件	无
调用函数	无

参数 TIM_EncoderMode

TIM_EncoderMode 选择 TIM 的编码器模式。这个参数可以是下列值之一：

表 526. TIM_EncoderMode 定义

TIM_ EncoderMode	描述
TIM_EncoderMode_TI1	TIM 编码器使用模式 1
TIM_EncoderMode_TI2	TIM 编码器使用模式 2
TIM_EncoderMode_TI12	TIM 编码器使用模式 3

例：

```
/* 为 TIM2 配置编码器模式 T11 */

TIM_EncoderInterfaceConfig(TIM2, TIM_EncoderMode_TI1,

TIM_ICPolarity_Rising, TIM_ICPolarity_Rising);
```

19.2.41 TIM_GenerateEvent 函数

*表 527*描述了 TIM_GenerateEvent 函数.

表 527. TIM_GenerateEvent function

函数名	TIM_ GenerateEvent
函数原型	void TIM_GenerateEvent(TIM_TypeDef* TIMx, u16 TIM_EventSource)
行为描述	配置 TIM 事件以软件方式生成
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择外设
输入参数 2	TIM_EventSource: TIM 软件事件源 参考 *TIM_ EventSource* 部分获取该参数允许值的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

参数 TIM_EventSource

TIM_EventSource 选择 TIM 软件事件源。这个参数可以被设置为下列值之一：

表 528. TIM_EventSource 的允许值

TIM_ EventSource	描述
TIM_EventSource_Update	TIM 更新事件源
TIM_EventSource_CC1	TIM 输出比较 1 事件源
TIM_EventSource_CC2	TIM 输出比较 2 事件源

TIM_EventSource_CC3	TIM 输出比较 3 事件源
TIM_EventSource_CC4	TIM 输出比较 4 事件源
TIM_EventSource_Trigger	TIM 触发器事件源

例：

```
/* 为 TIM2 选择触发器软件事件生成源*/
TIM_GenerateEvent(TIM2, TIM_EventSource_Trigger);
```

19.2.42 TIM_OC1PolarityConfig 函数

表 529 描述了 TIM_OC1PolarityConfig 函数。

表 529. TIM_OC1PolarityConfig 函数

函数名	TIM_ OC1PolarityConfig
函数原型	void TIM_OC1PolarityConfig(TIM_TypeDef* TIMx, u16 TIM_Polarity)
行为描述	配置某个 TIM 通道 1 的极性
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输入参数 2	TIM_OCPolarity: 输出比较的极性 参考 *TIM_ OCPolarity* 部分获取该参数允许值的更多细节
输出参数	无

返回值	无
前提条件	无
调用函数	无

例：

```
/*为 TIM2 通道 1 输出比较选择极性高 */
TIM_OC1PolarityConfig(TIM2, TIM_OCPolarity_High);
```

19.2.43 TIM_OC2PolarityConfig 函数

表 530 描述了 TIM_OC2PolarityConfig 函数。

表 530. TIM_OC2PolarityConfig 函数

函数名	TIM_ OC2PolarityConfig
函数原型	void TIM_OC2PolarityConfig(TIM_TypeDef* TIMx, u16 TIM_Polarity)
行为描述	配置某个 TIM 通道 2 的极性
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输入参数 2	TIM_OCPolarity: 输出比较的极性 参考 *TIM_ OCPolarity* 部分获取该参数允许值的更多细节
输出参数	无

返回值	无
前提条件	无
调用函数	无

例：

```
/*为 TIM2 通道 2 输出比较选择极性高 */
TIM_OC2PolarityConfig(TIM2, TIM_OCPolarity_High);
```

19.2.44 TIM_OC3PolarityConfig 函数

表 531 描述了 TIM_OC3PolarityConfig 函数。

表 531. TIM_OC3PolarityConfig 函数

函数名	TIM_ OC3PolarityConfig
函数原型	void TIM_OC3PolarityConfig(TIM_TypeDef* TIMx, u16 TIM_Polarity)
行为描述	配置某个 TIM 通道 3 的极性
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择外设
输入参数 2	TIM_OCPolarity: 输出比较的极性 参考 *TIM_ OCPolarity* 部分获取该参数允许值的更多细节
输出参数	无

返回值	无
前提条件	无
调用函数	无

例：

```
/*为 TIM2 通道 3 输出比较选择极性高 */
TIM_OC3PolarityConfig(TIM2, TIM_OCPolarity_High);
```

19.2.45 TIM_OC4PolarityConfig 函数

*表 532*描述了 TIM_OC4PolarityConfig 函数。

表 532. TIM_OC4PolarityConfig 函数

函数名	TIM_ OC4PolarityConfig
函数原型	void TIM_OC4PolarityConfig(TIM_TypeDef* TIMx, u16 TIM_Polarity)
行为描述	配置某个 TIM 通道 4 的极性
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输入参数 2	TIM_OCPolarity: 输出比较的极性 参考 *TIM_ OCPolarity* 部分获取该参数允许值的更多细节
输出参数	无

返回值	无
前提条件	无
调用函数	无

例：

/*为 TIM2 通道 4 输出比较选择极性高 */

TIM_OC4PolarityConfig(TIM2, TIM_OCPolarity_High);

19.2.46 TIM_UpdateRequestConfig 函数

表 533 描述了 TIM_UpdateRequestConfig 函数。

表 533. TIM_UpdateRequestConfig 函数

函数名	TIM_ UpdateRequestConfig
函数原型	void TIM_UpdateRequestConfig(TIM_TypeDef* TIMx, u16 TIM_UpdateSource)
行为描述	配置某个 TIM 的更新请求源
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输入参数 2	TIM_UpdateSource: 更新请求源 参考 *TIM_ UpdateSource* 部分获取该参数允许值的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

参数 TIM_UpdateSource

TIM_UpdateSource 选择某个 TIM 的的更新源。这个参数可以被设置以下值之一：

表 534. TIM_UpdateSource 的允许值

TIM_ UpdateSource	描述
TIM_UpdateSource_Global	更新的源是计数器上溢出/下溢出，或 UG 位的设置，或通过从模式控制器生成的更新
TIM_UpdateSource_Regular	更新的源是计数器上溢出/下溢出

例：

```
/* 为 TIM2 选择常规更新源 */
TIM_UpdateRequestConfig(TIM2, TIM_UpdateSource_Regular);
```

19.2.47 TIM_SelectHallSensor 函数

表 535 描述了 TIM_SelectHallSensor 函数。

表 535. TIM_SelectHallSensor 函数

函数名	TIM_ SelectHallSensor
函数原型	void TIM_SelectHallSensor(TIM_TypeDef* TIMx, FunctionalState Newstate)
行为描述	使能或关闭某个 TIM 的霍尔感应器接口
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输入参数 2	NewState: 霍尔感应器的新状态 这个参数可以是 ENABLE 或 DISABLE。

输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 使能 TIM2 的霍尔感应器接口 */
TIM_SelectHallSensor(TIM2, ENABLE);
```

19.2.48 TIM_SelectOnePulseMode 函数

表 536 描述了 TIM_SelectOnePulseMode 函数。

表 536. TIM_SelectOnePulseMode 函数

函数名	TIM_ SelectOnePulseMode
函数原型	void TIM_SelectOnePulseMode(TIM_TypeDef* TIMx, u16 TIM_OPMode)
行为描述	选择某个 TIM 为单脉冲模式
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输入参数 2	TIM_OPMode: 单脉冲模式 参考 *TIM_OPMode* 部分获取该参数的允许值的更多细节
输出参数	无
返回值	无

前提条件	无
调用函数	无

TIM_OPMode

TIM_OPMode 选择更新源. 这个参数可以被设置以下值之一：

表 537. TIM_OPMode 定义

TIM_ OPMode	描述
TIM_OPMode_Repetitive	生成重复的脉冲：在更新事件时计数器不停止
TIM_OPMode_Single	生成单个的脉冲 :在下一个更新事件时计数器停止

例：

```
/* 为 TIM2 选择单脉冲模式 */
TIM_SelectOnePulseMode(TIM2, TIM_OPMode_Single);
```

19.2.49 TIM_SelectOutputTrigger 函数

表 538 描述了 TIM_SelectOutputTrigger 函数。

表 538. TIM_SelectOutputTrigger 函数

函数名	TIM_ SelectOutputTrigger

函数原型	void TIM_SelectOutputTrigger(TIM_TypeDef* TIMx, u16 TIM_TRGOSource)
行为描述	选择某个 TIM 的触发器输出模式
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输入参数 2	TIM_ TRGOSource: 触发器输出源 参考 *TIM_ TRGOSource* 部分获取该参数的允许值的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

参数 TIM_TRGOSource

TIM_TRGOSource 选择 TIMx 触发器输出源。这个参数可以被设置以下值之一：

表 539. TIM8TRGOSource 的允许值

TIM_ TRGOSource	描述
TIM_TRGOSource_Reset	TIM_EGR 寄存器的 UG 位被用作触发器输出(TRGO)
TIM_TRGOSource_Enable	计数器使能 CEN 被用作触发器输出(TRGO)
TIM_TRGOSource_Update	更新事件被选作触发器输出(TRGO)
TIM_TRGOSource_OC1	一旦一个捕获或一个比较匹配发生 ,CC1IF 标记将被设置,触发器发送一个正脉冲(TRGO)

TIM_TRGOSource_OC1Ref	OC1REF 被用作触发器输出（TRGO）
TIM_TRGOSource_OC2Ref	OC2REF 被用作触发器输出(TRGO)
TIM_TRGOSource_OC3Ref	OC3REF 被用作触发器输出(TRGO)
TIM_TRGOSource_OC4Ref	OC4REF 被用作触发器输出(TRGO)

例：

```
/* 为 TIM2 选择更新事件作为触发器输出(TRGO) */
TIM_SelectOutputTrigger(TIM2, TIM_TRGOSource_Update);
```

19.2.50 TIM_SelectSlaveMode 函数

表 540 描述了 TIM_SelectSlaveMode 函数.

表 540. TIM_SelectSlaveMode 函数

函数名	TIM_ SelectSlaveMode
函数原型	void TIM_SelectSlaveMode(TIM_TypeDef* TIMx, u16 TIM_SlaveMode)
行为描述	选择某个 TIM 为从模式
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输入参数 2	TIM_SlaveMode: TIM 从模式 参考 *TIM_ SlaveMode* 部分获取该参数的允许值的更多细节

输出参数	无
返回值	无
前提条件	无
调用函数	无

参数 TIM_SlaveMode

TIM_SlaveMode 选择 TIM 的从模式这个参数可以被设置以下值之一：

表 541. TIM_SlaveMode 定义

TIM_ SlaveMode	描述
TIM_SlaveMode_Reset	选定的触发器信号(TRGI)的上升沿重新初始化计数器，且引发寄存器的更新
TIM_SlaveMode_Gated	当触发器信号(TRGI)为高时，计数器时钟使能
TIM_SlaveMode_Trigger	在触发器信号(TRGI)的上升沿计数器开始
TIM_SlaveMode_External1	选定的触发器信号(TRGI)的上升沿作为作为计数器时钟

例：

```
/* 选择 Gated 模式作为 TIM2 的从模式 */

TIM_SelectSlaveMode(TIM2, TIM_SlaveMode_Gated);
```

19.2.51 TIM_SelectMasterSlaveMode 函数

表 542 描述了 TIM_SelectMasterSlaveMode 函数。

表 542. TIM_SelectMasterSlaveMode 函数

函数名	TIM_ SelectMasterSlaveMode
函数原型	void TIM_SelectMasterSlaveMode(TIM_TypeDef* TIMx, u16 TIM_MasterSlaveMode)
行为描述	设置/重置 TIM 为主/从模式
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输入参数 2	TIM_ MasterSlaveMode: 计时器主从模式 参考 TIM_ MasterSlaveMode 部分获取该参数的允许值的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

参数 TIM_MasterSlaveMode

TIM_MasterSlaveMode 选择某个 TIM 为主从模式。参考表 *543* 获取这个参数的可用值。

表 543. TIM_MasterSlaveMode 定义

TIM_ MasterSlaveMode	描述
TIM_MasterSlaveMode_Enable	使能主从模式
TIM_MasterSlaveMode_Disable	关闭主从模式

例:

```
/* 为 TIM2 使能主从模式 */
TIM_SelectMasterSlaveMode(TIM2, TIM_MasterSlaveMode_Enable);
```

19.2.52 TIM_SetCounter 函数

表 544 描述了 TIM_SetCounter 函数。

表 544. TIM_SetCounter 函数

函数名	TIM_ SetCounter
函数原型	void TIM_SetCounter(TIM_TypeDef* TIMx, u16 Counter)
行为描述	设置某个 TIM 计数器寄存器的值
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输入参数 2	Counter: 计数器寄存器的新值
输出参数	无

返回值	无
前提条件	无
调用函数	无

例：

```
/* 设置 TIM2 新的计数器值*/
u16 TIMCounter = 0xFFFF;
TIM_SetCounter(TIM2, TIMCounter);
```

19.2.53 TIM_SetAutoreload 函数

表 545 描述了 TIM_SetAutoreload 函数。

表 545. TIM_SetAutoreload 函数

函数名	TIM_ SetAutoreload
函数原型	void TIM_SetAutoreload(TIM_TypeDef* TIMx, u16 Autoreload)
行为描述	设置某个 TIM 自动重载寄存器的值
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择外设
输入参数 2	Autoreload: 自动重载寄存器的新值
输出参数	无
返回值	无

前提条件	无
调用函数	无

例：

```
/* 为 TIM2 设置新的自动重载值 */
u16 TIMAutoreload = 0xFFFF;
TIM_SetAutoreload(TIM2, TIMAutoreload);
```

19.2.54 TIM_SetCompare1 函数

表 546 描述了 TIM_SetCompare1 函数.

表 546. TIM_SetCompare1 函数

函数名	TIM_ SetCompare1
函数原型	void TIM_SetCompare1(TIM_TypeDef* TIMx, u16 Compare1)
行为描述	设置某个 TIM 捕获比较 1 寄存器的值
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输入参数 2	Compare1: 捕获比较 1 寄存器的新值
输出参数	无
返回值	无
前提条件	无

调用函数	无

例：

```
/* 为 TIM2 设置捕获比较 1 的新值*/
u16 TIMCompare1 = 0x7FFF;
TIM_SetCompare1(TIM2, TIMCompare1);
```

19.2.55 TIM_SetCompare2 函数

*表 547*描述了 TIM_SetCompare2 函数.

表 547. TIM_SetCompare2 函数

函数名	TIM_ SetCompare2
函数原型	void TIM_SetCompare2(TIM_TypeDef* TIMx, u16 Compare2)
行为描述	设置某个 TIM 捕获比较 2 寄存器的值
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输入参数 2	Compare2: 捕获比较 2 寄存器的新值
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 为 TIM2 设置捕获比较 2 的新值*/
u16 TIMCompare2 = 0x7FFF;
TIM_SetCompare2(TIM2, TIMCompare2);
```

19.2.56 TIM_SetCompare3 函数

表 548 描述了 TIM_SetCompare3 函数.

表 548. TIM_SetCompare3 函数

函数名	TIM_ SetCompare3
函数原型	void TIM_SetCompare3(TIM_TypeDef* TIMx, u16 Compare3)
行为描述	设置某个 TIM 捕获比较 3 寄存器的值
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输入参数 2	Compare3: 捕获比较 3 寄存器的新值
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 为 TIM2 设置捕获比较 3 的新值*/
u16 TIMCompare3 = 0x7FFF;
TIM_SetCompare3(TIM2, TIMCompare3);
```

19.2.57 TIM_SetCompare4 函数

表 549 描述了 TIM_SetCompare4 函数.

表 549. TIM_SetCompare4 函数

函数名	TIM_ SetCompare4
函数原型	void TIM_SetCompare4(TIM_TypeDef* TIMx, u16 Compare4)
行为描述	设置某个 TIM 捕获比较 4 寄存器的值
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输入参数 2	Compare4: 捕获比较 4 寄存器的新值
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 为 TIM2 设置捕获比较 4 的新值*/
u16 TIMCompare4 = 0x7FFF;
TIM_SetCompare4TIM2, TIMCompare4);
```

19.2.58 TIM_SetIC1Prescaler 函数

表 550 描述了 TIM_SetIC1Prescaler 函数。

表 550. TIM_SetIC1Prescaler 函数

函数名	TIM_ SetIC1Prescaler
函数原型	void TIM_SetIC1Prescaler(TIM_TypeDef* TIMx, u16 TIM_IC1Prescaler)
行为描述	设置某个 TIM 输入捕获 1 的预分频数
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输入参数 2	TIM_IC1Prescaler: 输入捕获 1 的预分频数 参考 *TIM_ICPrescaler* 部分获取该参数允许值的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 设置 TIM2 输入捕获 1 的预分频数 */

TIM_SetIC1Prescaler(TIM2, TIM_ICPSC_Div2);
```

19.2.59 TIM_SetIC2Prescaler 函数

表 551 描述了 TIM_SetIC2Prescaler 函数。

表 551. TIM_SetIC1Prescaler 函数

函数名	TIM_ SetIC2Prescaler
函数原型	void TIM_SetIC2Prescaler(TIM_TypeDef* TIMx, u16 TIM_IC2Prescaler)
行为描述	设置某个 TIM 输入捕获 2 的预分频数
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择外设
输入参数 2	TIM_IC2Prescaler: 输入捕获 2 的预分频数 参考 *TIM_ICPrescaler* 部分获取该参数允许值的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/*设置 TIM2 输入捕获 2 的预分频数*/

TIM_SetIC2Prescaler(TIM2, TIM_ICPSC_Div2);
```

19.2.60 TIM_SetIC3Prescaler 函数

表 552 描述了 TIM_SetIC3Prescaler 函数。

表 552. TIM_SetIC3Prescaler 函数

函数名	TIM_ SetIC3Prescaler
函数原型	void TIM_SetIC3Prescaler(TIM_TypeDef* TIMx, u16 TIM_IC3Prescaler)
行为描述	设置某个 TIM 输入捕获 3 的预分频数
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输入参数 2	TIM_IC3Prescaler: 输入捕获 3 的预分频数 参考 *TIM_ICPrescaler* 部分获取该参数允许值的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/*设置 TIM2 输入捕获 3 的预分频数*/

TIM_SetIC3Prescaler(TIM2, TIM_ICPSC_Div2);
```

19.2.61 TIM_SetIC4Prescaler 函数

表 553 描述了 TIM_SetIC4Prescaler 函数。

表 553. TIM_SetIC4Prescaler 函数

函数名	TIM_ SetIC4Prescaler
函数原型	void TIM_SetIC4Prescaler(TIM_TypeDef* TIMx, u16 TIM_IC4Prescaler)
行为描述	设置某个 TIM 输入捕获 4 的预分频数
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输入参数 2	TIM_IC4Prescaler: 输入捕获 4 的预分频数 参考 *TIM_ICPrescaler* 部分获取该参数允许值的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

/*设置 TIM2 输入捕获 4 的预分频数*/

TIM_SetIC3Prescaler(TIM2, TIM_ICPSC_Div2);

19.2.62 TIM_SetClockDivision 函数

表 554 描述了 TIM_SetClockDivision 函数。

表 554. TIM_SetClockDivision 函数

函数名	TIM_ SetClockDivision
函数原型	void TIM_SetClockDivision(TIM_TypeDef* TIMx, u16 TIM_CKD)
行为描述	设置某个 TIM 时钟分割值
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择外设
输入参数 2	TIM_CKD: 时钟分割值 参考 TIM_ ClockDivision 部分获取该参数允许值的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

/* 设置 TIM2 的时钟分割值 */

TIM_SetClockDivision(TIM2, TIM_CKD_DIV4);

19.2.63 TIM_GetCapture1 函数

表 555 描述了 TIM_GetCapture1 函数。

表 555. TIM_GetCapture1 函数

函数名	TIM_ GetCapture1
函数原型	u16 TIM_GetCapture1(TIM_TypeDef* TIMx)
行为描述	获取某个 TIM 的输入捕获 1 的值
输入参数	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 获取 TIM2 的输入捕获 1 的值 */

u16 ICAP1value = TIM_GetCapture1(TIM2);
```

19.2.64 TIM_GetCapture2 函数

表 556 描述了 TIM_GetCapture2 函数。

表 556. TIM_GetCapture2 函数

函数名	TIM_ GetCapture2
函数原型	u16 TIM_GetCapture2(TIM_TypeDef* TIMx)
行为描述	获取某个 TIM 的输入捕获 2 的值
输入参数	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 获取 TIM2 的输入捕获 2 的值 */
u16 ICAP2value = TIM_GetCapture2(TIM2);
```

19.2.65 TIM_GetCapture3 函数

表 557 描述了 TIM_GetCapture3 函数。

表 557. TIM_GetCapture3 函数

函数名	TIM_ GetCapture3
函数原型	u16 TIM_GetCapture3(TIM_TypeDef* TIMx)
行为描述	获取某个 TIM 的输入捕获 3 的值
输入参数	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 获取 TIM2 的输入捕获 3 的值 */
u16 ICAP3value = TIM_GetCapture3(TIM2);
```

19.2.66 TIM_GetCapture4 函数

表 558 描述了 TIM_GetCapture4 函数。

表 558. TIM_GetCapture4 函数

函数名	TIM_ GetCapture4
函数原型	u16 TIM_GetCapture4(TIM_TypeDef* TIMx)

行为描述	获取某个 TIM 的输入捕获 4 的值
输入参数	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 获取 TIM2 的输入捕获 4 的值 */
u16 ICAP4value = TIM_GetCapture4(TIM2);
```

19.2.67 TIM_GetCounter 函数

表 559 描述了 TIM_GetCounter 函数。

表 559. TIM_GetCounter 函数

函数名	TIM_ GetCounter
函数原型	ut16 TIM_GetCounter(TIM_TypeDef* TIMx)
行为描述	获取某个 TIM 的计数器的值
输入参数	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输出参数	无

返回值	无
前提条件	无
调用函数	无

例：

```
/* 获取 TIM2 计数器的值*/
u16 TIMCounter = TIM_GetCounter(TIM2);
```

19.2.68 TIM_GetPrescaler 函数

表 560 描述了 TIM_GetPrescaler 函数。

表 559. TIM_ GetPrescaler 函数

函数名	TIM_GetPrescaler
函数原型	ut16 TIM_GetPrescaler (TIM_TypeDef* TIMx)
行为描述	获取某个 TIM 的预分频数的值
输入参数	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输出参数	无
返回值	无
前提条件	无
调用函数	无

例:

```
/* 获取 TIM2 预分频数的值*/
u16 TIMPrescaler = TIM_GetPrescaler(TIM2);
```

19.2.69 TIM_GetFlagStatus 函数

表 561 描述了 TIM_GetFlagStatus 函数。

表 561. TIM_GetFlagStatus 函数

函数名	TIM_ GetFlagStatus
函数原型	FlagStatus TIM_GetFlagStatus(TIM_TypeDef* TIMx, u16 TIM_FLAG)
行为描述	检查是否特定的 TIM 标记被设置
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输入参数 2	TIM_FLAG: 用于检查的标记 参考 *TIM_FLAG* 部分获取该参数允许值的更多细节
输出参数	无
返回值	TIM_FLAG 的新状态(SET 或 RESET)
前提条件	无
调用函数	无

TIM_FLAG

可调用函数 TIM_GetFlagStatus 来检查的 TIMx 标记列举在如下表格中：

表 562. TIM_FLAG definition

TIM_ FLAG	描述
TIM_FLAG_Update	TIM 更新标记
TIM_FLAG_CC1	TIM 捕获比较 1 标记
TIM_FLAG_CC2	TIM 捕获比较 2 标记
TIM_FLAG_CC3	TIM 捕获比较 3 标记
TIM_FLAG_CC4	TIM 捕获比较 4 标记
TIM_FLAG_Trigger	TIM 触发器标记
TIM_FLAG_CC1OF	TIM 捕获比较 1 溢出标记
TIM_FLAG_CC2OF	TIM 捕获比较 2 溢出标记
TIM_FLAG_CC3OF	TIM 捕获比较 3 溢出标记
TIM_FLAG_CC4OF	TIM 捕获比较 4 溢出标记

例：

```
/* 检查 TIM2 捕获比较 1 标记是否被设置 */

if(TIM_GetFlagStatus(TIM2, TIM_FLAG_CC1) == SET)

{

}
```

19.2.70 TIM_ClearFlag 函数

表 563 描述了 TIM_ClearFlag 函数。

表 563. TIM_ClearFlag 函数

函数名	TIM_ ClearFlag
函数原型	void TIM_ClearFlag(TIM_TypeDef* TIMx, u16 TIM_Flag)
行为描述	清除 TIM 的挂起标记
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输入参数 2	TIM_FLAG: 用于清除的标记 参考 *TIM_FLAG* 部分获取该参数允许值的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 清除 TIM2 捕获比较 1 标记 */
TIM_ClearFlag(TIM2, TIM_FLAG_CC1);
```

19.2.71 TIM_GetITStatus 函数

表 564 描述了 TIM_GetITStatus 函数。

表 564. TIM_GetITStatus 函数

函数名	TIM_ GetITStatus
函数原型	ITStatus TIM_GetITStatus(TIM_TypeDef* TIMx, u16 TIM_IT)
行为描述	检查特定 TIM 中断是否发生
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择外设
输入参数 2	TIM_ IT: 规定要检查的 TIM 中断源 参考 *TIM_ IT* 部分获取该参数允许值的更多细节
输出参数	无
返回值	TIM_IT 的新状态（SET 或 RESET）
前提条件	无
调用函数	无

例：

```
/*检查 TIM2 捕获比较 1 中断是否发生 */
if(TIM_GetITStatus(TIM2, TIM_IT_CC1) == SET)
{
}
```

19.2.72 TIM_ClearITPendingBit 函数

表 565 描述了 TIM_ClearITPendingBit 函数.

表 565. TIM_ClearITPendingBit 函数

函数名	TIM_ ClearITPending
函数原型	void TIM_ClearITPendingBit(TIM_TypeDef* TIMx, u16 TIM_IT)
行为描述	清除 TIMx 的中断挂起位
输入参数 1	TIMx：此处 x 可以是 2，3 或 4 以选择 TIM 外设
输入参数 2	TIM_ IT: 指定要清除的中断挂起位 参考 *TIM_ IT* 部分获取该参数允许值的更多细节
输出参数	无
返回值	无
前提条件	无
调用函数	无

例：

```
/* 清除 TIM2 的捕获比较 1 中断挂起位*/
TIM_ClearITPendingBit(TIM2, TIM_IT_CC1);
```

20 高级控制计时器（TIM1）

TIM1 包括一个由可编程的预分频器驱动的 16 位自动重载计数器。

这个计时器用途很多，包括输入信号脉冲长度的测量（输入捕捉）或是输出波形的形成（输出比较，PWM，互补的带有时限插入的 PWM）

通过使用计时器预分频器和 CPU 时钟预分频器，脉冲长度和波形的周期可以调制为几微秒到几

毫秒。

20.1： TIM1 寄存器结构体描述了 TIM1 固件库中使用的数据结构。20.2：固件库函数列出了固件库里相关的函数。

20.1 TIM1 寄存器结构体

TIM1 寄存器结构体，TIM1_TypeDef 是定义在 stm32f10x_map.h 文件中，其结构如下：

```
typedef struct
{
  vu16 CR1;
  u16 RESERVED0;
  vu16 CR2;
  u16 RESERVED1;
  vu16 SMCR;
  u16 RESERVED2;
  vu16 DIER;
  u16 RESERVED3;
  vu16 SR;
  u16 RESERVED4;
  vu16 EGR;
  u16 RESERVED5;
  vu16 CCMR1;
  u16 RESERVED6;
```

```
vu16 CCMR2;
u16 RESERVED7;
vu16 CCER;
u16 RESERVED8;
vu16 CNT;
u16 RESERVED9;
vu16 PSC;
u16 RESERVED10;
vu16 ARR;
u16 RESERVED11;
vu16 RCR;
u16 RESERVED12;
vu16 CCR1;
u16 RESERVED13;
vu16 CCR2;
u16 RESERVED14;
vu16 CCR3;
        u16 RESERVED15;
        vu16 CCR4;
        u16 RESERVED16;
       vu16 BDTR;
```

```
    u16 RESERVED17;

    vu16 DCR;

    u16 RESERVED18;

    vu16 DMAR;

    u16 RESERVED19;

} TIM1_TypeDef;
```

表 566 给出了 TIM1 寄存器的清单。

表 566.TIM1 寄存器

寄存器	描述
CR1	控制寄存器 1
CR2	控制寄存器 2
SMCR	从模式控制寄存器
DIER	DMA 和中断使能寄存器
SR	状态寄存器
EGR	事件产生寄存器
CCMR1	捕获/比较模式寄存器 1
CCMR2	捕获/比较模式寄存器 2
CCER	捕获/比较模式寄存器
CNT	计数器寄存器
PSC	预分频寄存器
ARR	自动重载寄存器

RCR	循环计数器寄存器
CCR1	捕获/比较寄存器 1
CCR2	捕获/比较寄存器 2
CCR3	捕获/比较寄存器 3
CCR4	捕获/比较寄存器 4
BDTR	间断和空载时间寄存器
DCR	DMA 控制寄存器
DMAR	脉冲模式下 DMA 地址寄存器

TIM1 的边界在 stm32f10x_map 声明为：

…

```
#define PERIPH_BASE              ((u32)0x40000000)
#define APB1PERIPH_BASE          PERIPH_BASE
#define APB2PERIPH_BASE          (PERIPH_BASE + 0x10000)
#define AHBPERIPH_BASE           (PERIPH_BASE + 0x20000)
```

….

```
#define TIM1_BASE                (APB2PERIPH_BASE + 0x2C00)
```

….

```
#ifndef DEBUG
```

…

```
#ifdef _TIM1
```

```
#define TIM1                    ((TIM1_TypeDef *) TIM1_BASE)

#endif /*_TIM1 */

...

#else     /* DEBUG */

...

#ifdef _TIM1

  EXT TIM1_TypeDef                  *TIM1;

#endif /*_TIM1 */

..

#endif
```

当使用 Debug 模式，_TIM1 指针在 stm32f10x_lib.c 文件中被初始化：

```
...

#ifdef _TIM1

  TIM1 = (TIM1_TypeDef *)    TIM1_BASE;

#endif /*_TIM1 *//

...
```

为了访问 TIM1 寄存器，_TIM1 必须在 stm32f10x_conf.h 被定义，如下：

```
...

#define _TIM1

...
```

20.2 固件库函数

表 567 给出了 TIM1 库中大量函数的清单。

表 567.TIM1 固件库函数

寄存器	描述
TIM1_DeInit	重置 TIM1 外围寄存器为他们的默认重置值。
TIM1_TimeBaseInit	根据 TIM1_TimeBaseInitStruct 中指定的参数初始化 TIM1 时间基础模块。
TIM1_OC1Init	根据 TIM1_OCInitStruct 中指定的参数初始化 TIM1 信道 1。
TIM1_OC2Init	根据 TIM1_OCInitStruct 中指定的参数初始化 TIM1 信道 2。
TIM1_OC3Init	根据 TIM1_OCInitStruct 中指定的参数初始化 TIM1 信道 3。
TIM1_OC4Init	根据 TIM1_OCInitStruct 中指定的参数初始化 TIM1 信道 4。
TIM1_BDTRConfig	配置：间断特征，空载时间，锁级别，OSSI，OSSR 状态和 AOE（自动输出使能）
TIM1_ICInit	根据 TIM1_ICInitStruct 中指定的参数初始化 TIM1 外围设备。

TIM1_PWMIConfig	根据 TIM1_ICInitStruct 中指定的参数配置 TIM1 外围设备为 PWM 输入模式。
TIM1_TimeBaseStructInit	将 TIM1_TimeBaseInitStruct 中的成员赋为默认值
TIM1_OCStructInit	将 TIM1_OCInitStruct 中的成员赋为默认值
TIM1_ICStructInit	将 TIM1_ICInitStruct 中的成员赋为默认值
TIM1_BDTRStructInit	将 TIM1_BDTRInitStruct 中的成员赋为默认值
TIM1_Cmd	使能或禁用特定的 TIM1 外围设备
TIM1_CtrlPWMOutputs	使能或禁用 TIM1 外围主输出
TIM1_ITConfig	使能或禁用特定 TIM1 中断
TIM1_DMAConfig	配置 TIM1 的 DMA 接口
TIM1_DMACmd	使能或禁用 TIM1 的 DMA 请求
TIM1_InternalClockConfig	配置 TIM1 内部时钟
TIM1_ETRClockMode1Config	配置 TIM1 的外部时钟模式 1
TIM1_ETRClockMode2Config	配置 TIM1 的外部时钟模式 2
TIM1_ETRConfig	配置 TIM1 外部触发器
TIM1_ITRxExternalClockConfig	将外部时钟配置为 TIM1 内部触发器
TIM1_TIxExternalClockConfig	将外部时钟配置为 TIM1 触发器

TIM1_SelectInputTrigger	选择 TIM1 输入触发器源
TIM1_UpdateDisableConfig	使能或是阻止 TIM1 更新事件
TIM1_UpdateRequestConfig	选择 TIM1 更新请求中断源
TIM1_SelectHallSensor	使能或禁用 TIM1 的霍耳传感器接口
TIM1_SelectOnePulseMode	使能或禁用 TIM1 的单脉冲模式
TIM1_SelectOutputTrigger	选择 TIM1 触发器输出模式
TIM1_SelectSlaveMode	选择 TIM1 从模式
TIM1_SelectMasterSlaveMode	设置或重设 TIM1 的主/从模式
TIM1_EncoderInterfaceConfig	配置 TIM1 编码器接口
TIM1_PrescalerConfig	配置 TIM 预分频器
TIM1_CounterModeConfig	指明被使用的 TIM1 计数器模式
TIM1_ForcedOC1Config	强制 TIM1 信道 1 的输出波形到有效或无效电平
TIM1_ForcedOC2Config	强制 TIM1 信道 2 的输出波形到有效或无效电平
TIM1_ForcedOC3Config	强制 TIM1 信道 3 的输出波形到有效或无效电平
TIM1_ForcedOC4Config	强制 TIM1 信道 4 的输出波形到有效或无效电平
TIM1_ARRPreloadConfig	使能或禁用 TIM1 外围 ARR 预载寄存器
TIM1_SelectCOM	选择 TIM1 外围设备交换事件

TIM1_SelectCCDMA	选择 TIM1 外围设备捕获比较 DMA 源
TIM1_CCPreloadContro	设置或重置 TIM1 外围设备捕获比较预载控制位
TIM1_OC1PreloadConfi	使能或禁用 TIM1 外围 CCR1 预载寄存器
TIM1_OC2PreloadConfi	使能或禁用 TIM1 外围 CCR2 预载寄存器
TIM1_OC3PreloadConfi	使能或禁用 TIM1 外围 CCR3 预载寄存器
TIM1_OC4PreloadConfi	使能或禁用 TIM1 外围 CCR4 预载寄存器
TIM1_OC1FastConfig	配置 TIM1 捕获比较器 1 快速性能
TIM1_OC2FastConfig	配置 TIM1 捕获比较器 2 快速性能
TIM1_OC3FastConfig	配置 TIM1 捕获比较器 3 快速性能
TIM1_OC4FastConfig	配置 TIM1 捕获比较器 4 快速性能
TIM1_ClearOC1Ref	清除或维护在外部事件中的 OCREF1 信号
TIM1_ClearOC2Ref	清除或维护在外部事件中的 OCREF2 信号
TIM1_ClearOC3Ref	清除或维护在外部事件中的 OCREF3 信号
TIM1_ClearOC4Ref	清除或维护在外部事件中的 OCREF4 信号
TIM1_GenerateEvent	配置 TIM1 事件为软件产生
TIM1_OC1PolarityConfig	配置 TIM1 信道 1 极性

TIM1_OC1NPolarityConfig	配置 TIM1 信道 1N 极性
TIM1_OC2PolarityConfig	配置 TIM1 信道 2 极性
TIM1_OC2NPolarityConfig	配置 TIM1 信道 2N 极性
TIM1_OC3PolarityConfig	配置 TIM1 信道 3 极性
TIM1_OC3NPolarityConfig	配置 TIM1 信道 3N 极性
TIM1_OC4PolarityConfig	配置 TIM1 信道 4 极性
TIM1_CCxCmd	使能或禁用 TIM1 捕获比较信道 x
TIM1_CCxNCmd	使能或禁用 TIM1 捕获比较信道 xN
TIM1_SelectOCxM	选择 TIM1 输出比较模式。 这个函数在改变输出比较模式之前禁用选择的信道。用户不得不使用 TIM1_CCxCmd 和 TIM1_CCxNCmd 函数使能这个信道。
TIM1_SetCounter	设置 TIM1 计数器寄存器的值
TIM1_SetAutoreload	设置 TIM1 自动重载寄存器的值
TIM1_SetCompare1	设置 TIM1 捕获比较 1 寄存器的值
TIM1_SetCompare2	设置 TIM1 捕获比较 2 寄存器的值
TIM1_SetCompare3	设置 TIM1 捕获比较 3 寄存器的值
TIM1_SetCompare4	设置 TIM1 捕获比较 4 寄存器的值
TIM1_SetIC1Prescaler	设置 TIM1 输入捕获 1 预分频器
TIM1_SetIC2Prescaler	设置 TIM1 输入捕获 2 预分频器
TIM1_SetIC3Prescaler	设置 TIM1 输入捕获 3 预分频器

TIM1_SetIC4Prescaler	设置 TIM1 输入捕获 4 预分频器
TIM1_SetClockDivision	设置 TIM1 时钟分隔的值
TIM1_GetCapture1	获取 TIM1 输入捕获 1 的值
TIM1_GetCapture2	获取 TIM1 输入捕获 2 的值
TIM1_GetCapture3	获取 TIM1 输入捕获 3 的值
TIM1_GetCapture4	获取 TIM1 输入捕获 4 的值
TIM1_GetCounter	获取 TIM1 计数器的值
TIM1_GetPrescaler	获取预分频器的值
TIM1_GetFlagStatus	检查指定的 TIM1 标记是否被置位
TIM1_ClearFlag	清除 TIM1 未决的标记
TIM1_GetITStatus	检查指定的 TIM1 中断是否发生
TIM1_ClearITPendingBit	清楚 TIM1 中断未决位。

20.2.1 TIM1_DeInit 函数

表 568 描述了 TIM1_DeInit 函数。

表 568 TIM1_DeInit 函数

函数名	TIM1_DeInit
函数原型	void TIM1_DeInit(void)
行为描述	重置 TIM1 外围寄存器到默认值的重置值
输入参数	无

输出参数	无
返回参数（值）	无
前提条件	无
调用函数	RCC_APB2PeriphResetCmd

例：

```
/* 将 TIM1 复位 */
TIM1_DeInit();
```

20.2.2 TIM1_TimeBaseInit 函数

表 569 描述了 TIM1_TimeBaseInit 函数。

表 569 TIM1_TimeBaseInit 函数

函数名	TIM1_TimeBaseInit
函数原型	Void TIM1_TimeBaseInit(TIM1_TimeBaseInitTypeDef* TIM1_BaseInitStruct)
行为描述	根据 TIM1_TimeBaseInitStruct 中指明的参数初始化 TIM1 时间基础模块
输入参数	TIM1_BaseInitStruct：指向一个包含特定的 TIM1 时间基础单元的配置信息的 TIM1_BaseInitTypeDef 结构体。

	关于这个参数可用的值的更多详细信息请参阅章节: TIM1_TimeBaseInitTypeDef 结构体。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

TIM1_TimeBaseInitTypeDef 结构

TIM1_TimeBaseInitTypeDef 结构在 stm32f10x_tim1.h 文件中被定义：

typedef struct

{

u16 TIM1_Period;

u16 TIM1_Prescaler;

u16 TIM1_ClockDivision;

u16 TIM1_CounterMode;

u8 TIM1_RepetitionCounter;

} TIM1_BaseInitTypeDef;

TIM1_Period

TIM1_Period 配置活动的自动重载寄存器中在下一次更新事件中导入的周期值。这个成员必须是一个在 0x0000 和 0xFFFF 之间的数字。

TIM1_Prescaler

TIM1_Prescaler 配置用于分割 TIM1 时钟的预分频器的值。这个成员必须是一个在 0x0000 和 0xFFFF 之间的数字。

TIM1_ClockDivision

TIM1_ClockDivision 配置时钟分隔。这个成员可以被设置成下列的值之一：

表 570.TIM1_ClockDivision

TIM1_ClockDivision	描述
TIM1_CKD_DIV1	$T_{DTS}=T_{ck\_tim}$
TIM1_CKD_DIV1	$T_{DTS}=2*T_{ck\_tim}$
TIM1_CKD_DIV1	$T_{DTS}=4*T_{ck\_tim}$

20.2.3 TIM1_CounterMode

TIM1_CounterMode 选择计数器的模式。这个成员可以是下列值中之一：

表 571. TIM1_CounterMode 定义

TIM1_ClockDivision	描述
TIM1_Counter_Up	TIM1 顺序计数模式
TIM1_Counter_Down	TIM1 倒序计数模式
TIM1_Counter_CenterAlig ned1	TIM1 中间对齐模式 1 计数模式
TIM1_Counter_CenterAlig ned2	TIM1 中间对齐模式 2 计数模式

TIM1_Counter_CenterAlig ned3	TIM1 中间对齐模式 3 计数模式

TIM1_RepetitionCounter

TIM1_RepetitionCounter 配置循环计数器的值。每一次 RCR 倒计数器到达 0，将会产生一个更新事件并且计数重新从 RCR 值（N）开始。

这个意味着在 PWM 模式(N+1)符合以下各条：

PWM 周期的数目处于边沿对齐模式

半 PWM 周期的数目在中间对齐 center-aligned 模式

这个成员必须是在 0x00 到 0xFF 之间的数字。

20.2.4 TIM1_OC1Init 函数

表 572 描述了 TIM1_OC1Init 函数。

表 572.TIM1_OC1Init 函数

函数名	TIM1_OC1Init
函数原型	Void TIM1_OC1Init(TIM1_OCInitTypeDef* TIM1_OCInitStruct)
行为描述	根据 TIM1_OCInitStruct 中指明的参数初始化 TIM1 信道 1
输入参数	TIM1_OCInitStruct :指向一个包含特定的 TIM1 外围设备的配置信息的 TIM1_OCInitTypeDef 结构

	体。 关于这个参数可用的值的更多详细信息请参阅章节: TIM1_OCInitTypeDef 结构体。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

TIM1_OCInitTypeDef 结构体

TIM1_OCInitTypeDef 结构体在 stm32f10x_tim1.h 中被定义：

typedef struct

```
{
  u16 TIM1_OCMode;
  u16 TIM1_OutputState;
  u16 TIM1_OutputNState;
  u16 TIM1_Pulse;
  u16 TIM1_OCPolarity;
  u16 TIM1_OCNPolarity;
  u16 TIM1_OCIdleState;
  u16 TIM1_OCNIdleState;
} TIM1_OCInitTypeDef;
```

TIM1_OCMode

TIM1_OCMode 选择 TIM1 模式，这个成员能够被设置为下列值中的一个。

表 573.TIM1_OCMode 定义

TIM1_OCMode	描述
TIM1_OCMode_Timing	TIM1 输出比较时间模式
TIM1_OCMode_Active	TIM1 输出比较有效模式
TIM1_OCMode_Inactive	TIM1 输出比较无效模式
TIM1_OCMode_Toggle	TIM1 输出比较触发模式
TIM1_OCMode_PWM1	TIM1 脉冲宽度调制模式 1
TIM1_OCMode_PWM2	TIM1 脉冲宽度调制模式 2

TIM1_OutputState

TIM1_OutputState 选择 TIM1 输出比较状态，这个成员能够被设置为下列值中的一个。

表 574. TIM1_OutputState **定义**

TIM1_OutputState	描述
TIM1_OutputState_Disable	TIM1 输出比较状态禁用
TIM1_OutputState_Enable	TIM1 输出比较状态使能

TIM1_OutputNState

TIM1_OutputNState 选择 TIM1 补充输出比较状态，这个成员能够被设置为下列值中的一个。

表 575. TIM1_OutputNState 定义

TIM1_OutputNState	描述
TIM1_OutputNState_Disable	TIM1 输出 N 比较状态禁用
TIM1_OutputNState_Enable	TIM1 输出 N 比较状态使能

TIM1_Pulse

TIM1_Pulse 配置要被载入捕获比较寄存器中的脉冲值。这个成员必须是 0x0000 到 0xFFFF 中的一个数。

TIM1_OCPolarity

TIM1_OCPolarity 配置输出极性。这个成员能够被设置成下面值中的一个。

表 576. TIM1_OCPolarity 定义

TIM1_OCPolarity	描述
TIM1_OCPolarity_High	输出比较极性高
TIM1_OCPolarity_Low	输出比较极性低

TIM1_OCNPolarity

TIM1_OCNPolarity 配置补充输出极性。这个成员能够被设置成下面值中的一个。

表 577. TIM1_OCNPolarity 定义

TIM1_OCNPolarity	描述
TIM1_OCNPolarity_High	输出比较 N 极性高
TIM1_OCNPolarity_Low	输出比较 N 极性低

TIM1_OCIdleState

TIM1_OCIdleState 为空闲状态选择关闭状态。这个成员可以被设置成下列值之一。

表 578. TIM1_OCIdleState 定义

TIM1_OcIdleState	描述
TIM1_OCIdleState_Set	当 MOE = 0 设置 TIM1 输出 OC 空闲状态置位
TIM1_OCIdleState_Reset	当 MOE = 0 重置 TIM1 输出 OC 空闲状态复位

TIM1_OCNIdleState

TIM1_OCNIdleState 为空闲状态选择关闭状态。这个成员可以被设置成下列值之一。

表 579. TIM1_OCNIdleState 定义

TIM1_OcIdleState	描述
TIM1_OCNIdleState_Set	当 MOE = 0 设置 TIM1 输出 OCN 空闲状态置位
TIM1_OCNIdleState_Reset	当 MOE = 0 重置 TIM1 输出 OCN 空闲状态复位

例：

```
/* 将 TIM1 通道 1 配置成 PWM 模式*/

TIM1_OCInitTypeDef TIM1_OCInitStructure;

TIM1_OCInitStructure.TIM1_OCMode = TIM1_OCMode_PWM1;
```

```
TIM1_OCInitStructure.TIM1_OutputState = TIM1_OutputState_Enable;
TIM1_OCInitStructure.TIM1_OutputNState = TIM1_OutputNState_Enable;
TIM1_OCInitStructure.TIM1_Pulse = 0x7FF;
TIM1_OCInitStructure.TIM1_OCPolarity = TIM1_OCPolarity_Low;
TIM1_OCInitStructure.TIM1_OCNPolarity = TIM1_OCNPolarity_Low;
TIM1_OCInitStructure.TIM1_OCIdleState = TIM1_OCIdleState_Set;
TIM1_OCInitStructure.TIM1_OCNIdleState = TIM1_OCIdleState_Reset;
TIM1_OC1Init(&TIM1_OCInitStructure);
```

20.2.5 TIM1_OC2Init 函数

表 580 描述了 TIM1_OC2Init 函数。

表 580. TIM1_OC2Init 函数

函数名	TIM1_OC2Init
函数原型	void TIM1_OC2Init(TIM1_OCInitTypeDef* TIM1_OCInitStruct)
行为描述	根据 TIM1_OCInitStruct 中指明的参数初始化 TIM1 信道 2
输入参数	TIM1_OCInitStruct :指向一个包含特定的 TIM1 外围设备的配置信息的 TIM1_OCInitTypeDef 结构体。

	关于这个参数可用的值的更多详细信息请参阅章节: TIM1_OCInitTypeDef 结构体。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/*将 TIM1 通道 2 配置成 PWM 模式*/
TIM1_OCInitTypeDef TIM1_OCInitStructure;
TIM1_OCInitStructure.TIM1_OCMode = TIM1_OCMode_PWM1;
TIM1_OCInitStructure.TIM1_OutputState = TIM1_OutputState_Enable;
TIM1_OCInitStructure.TIM1_OutputNState = TIM1_OutputNState_Enable;
TIM1_OCInitStructure.TIM1_Pulse = 0x7FF;
TIM1_OCInitStructure.TIM1_OCPolarity = TIM1_OCPolarity_Low;
TIM1_OCInitStructure.TIM1_OCNPolarity = TIM1_OCNPolarity_Low;
TIM1_OCInitStructure.TIM1_OCIdleState = TIM1_OCIdleState_Set;
TIM1_OCInitStructure.TIM1_OCNIdleState = TIM1_OCIdleState_Reset;

TIM1_OC2Init(&TIM1_OCInitStructure);
```

20.2.6 TIM1_OC3Init 函数

表 581 描述了 TIM1_OC3Init 函数。

表 581. TIM1_OC3Init 函数

函数名	TIM1_OC3Init
函数原型	void TIM1_OC3Init(TIM1_OCInitTypeDef* TIM1_OCInitStruct)
行为描述	通过在TIM1_OCInitStruct中指明的参数初始化 TIM1 信道 3
输入参数	TIM1_OCInitStruct :指向一个包含特定的 TIM1 外围设备的配置信息的 TIM1_OCInitTypeDef 结构体。 关于这个参数可用的值的更多详细信息请参阅章节: TIM1_OCInitTypeDef 结构体。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/*将 TIM1 通道 3 配置成 PWM 模式*/
TIM1_OCInitTypeDef TIM1_OCInitStructure;
TIM1_OCInitStructure.TIM1_OCMode = TIM1_OCMode_PWM1;
TIM1_OCInitStructure.TIM1_OutputState = TIM1_OutputState_Enable;
TIM1_OCInitStructure.TIM1_OutputNState = TIM1_OutputNState_Enable;
```

```
TIM1_OCInitStructure.TIM1_Pulse = 0x7FF;

TIM1_OCInitStructure.TIM1_OCPolarity = TIM1_OCPolarity_Low;

TIM1_OCInitStructure.TIM1_OCNPolarity = TIM1_OCNPolarity_Low;

TIM1_OCInitStructure.TIM1_OCIdleState = TIM1_OCIdleState_Set;

TIM1_OCInitStructure.TIM1_OCNIdleState = TIM1_OCIdleState_Reset;


TIM1_OC3Init(&TIM1_OCInitStructure);
```

20.2.7 TIM1_OC4Init 函数

表 582 描述了 TIM1_OC4Init 函数。

表 582. TIM1_OC4Init 函数

函数名	TIM1_OC4Init
函数原型	void TIM1_OC4Init(TIM1_OCInitTypeDef* TIM1_OCInitStruct)
行为描述	通过在TIM1_OCInitStruct中指明的参数初始化 TIM1 信道 4
输入参数	TIM1_OCInitStruct :指向一个包含特定的 TIM1 外围设备的配置信息的 TIM1_OCInitTypeDef 结构体。 关于这个参数可用的值的更多详细信息请参阅章节: TIM1_OCInitTypeDef 结构体。

输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/*将 TIM1 通道 4 配置成 PWM 模式*/

TIM1_OCInitTypeDef TIM1_OCInitStructure;

TIM1_OCInitStructure.TIM1_OCMode = TIM1_OCMode_PWM1;

TIM1_OCInitStructure.TIM1_OutputState = TIM1_OutputState_Enable;

TIM1_OCInitStructure.TIM1_Pulse = 0x7FF;

TIM1_OCInitStructure.TIM1_OCPolarity = TIM1_OCPolarity_Low;

TIM1_OCInitStructure.TIM1_OCIdleState = TIM1_OCIdleState_Set;


TIM1_OC4Init(&TIM1_OCInitStructure);
```

20.2.8 TIM1_BDTRConfig 函数

表 583 描述了 TIM1_BDTRConfig 函数。

表 583. TIM1_BDTRConfig 函数

函数名	TIM1_BDTRConfig
函数原型	void TIM1_BDTRConfig(TIM1_BDTRInitTypeDef

	*TIM1_BDTRInitStruct)
行为描述	配置：间断特征，滞后时间，锁级别，OSSI，OSSR 状态和 AOE（自动输出使能）
输入参数	TIM1_BDTRInitStruct：指向一个包含 TIM1 外围设备的 BDTR 寄存器配置信息的 TIM1_BDTRInitTypeDef 结构体。 关于这个参数可用的值的更多详细信息请参阅章节: TIM1_BDTRInitTypeDef 结构体。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

TIM1_BDTRInitStruct 结构体

TIM1_BDTRInitStruct 结构体是被定义在 stm32f10x_tim1.h 文件中的。

typedef struct

{

u16 TIM1_OSSRState;

u16 TIM1_OSSIState;

u16 TIM1_LOCKLevel;

u16 TIM1_DeadTime;

u16 TIM1_Break;

u16 TIM1_BreakPolarity;

u16 TIM1_AutomaticOutput

} TIM1_BDTRInitTypeDef;

TIM1_OSSRState

TIM1_OSSRState 配置被用于运行模式的关闭状态选择。这个成员能够被设置成下列值之一。

表 584. TIM1_OSSRState 定义

TIM1_OSSRState	描述
TIM1_OSSRState_Enable	TIM1 OSSR 状态是使能的
TIM1_OSSRState_Disable	TIM1 OSSR 状态是禁用的

TIM1_OSSIState

TIM1_OSSIState 选择被用于空闲状态的关闭状态。这个成员能够被设置成下列值之一。

表 585. TIM1_OSSIState 定义

TIM1_OSSIState	描述
TIM1_OSSIState_Enable	TIM1 OSSI 状态是使能的
TIM1_OSSIState_Disable	TIM1 OSSI 状态是禁用的

TIM1_LOCKLevel

TIM1_LOCKLevel 配置锁级别参数。这个成员能够被设置成下列值之一。

表 586. TIM1_LOCKLevel 定义

TIM1_LOCKLevel	描述
TIM1_LOCKLevel_OFF	没有位被锁
TIM1_LOCKLevel_1	使用锁级别 1
TIM1_LOCKLevel_2	使用锁级别 2
TIM1_LOCKLevel_3	使用锁级别 3

TIM1_DeadTime

TIM1_DeadTime 指明输出关闭和打开之间的延迟时间。

TIM1_Break

TIM1_Break 使能或禁止 TIM1 Break 输出。这个成员可以被设置成以下值之一。

表 587. TIM1_Break 定义

TIM1_Break	描述
TIM1_Break_Enable	TIM1 间断输入被使能
TIM1_Break_Disable	TIM1 间断输入被禁止

TIM1_BreakPolarity

TIM1_BreakPolarity 配置 TIM1_Break 输入的高低。这个成员可以被设置成下列值之一。

表 588. TIM1_BreakPolarity 定义

TIM1_BreakPolarity	描述
TIM1_BreakPolarity_Low	TIM1 间断输入引脚极性为低

TIM1_BreakPolarity_High	TIM1 间断输入引脚极性为高

TIM1_AutomaticOutput

TIM1_AutomaticOutput 使能或禁用自动输出特征。这个成员可以被设置为下列值之一。

表 589. TIM1_AutomaticOutput 定义

TIM1_AutomaticOutput	描述
TIM1_AutomaticOutput_Enable	TIM1 自动输出使能
TIM1_AutomaticOutput_Disable	TIM1 自动输出禁用

例：

```
/* OSSR, OSSI, , 自动输出使能,间断,空转时间和锁级别配置*/
TIM1_BDTRInitTypeDef TIM1_BDTRInitStructure;
TIM1_BDTRInitStructure.TIM1_OSSRState = TIM1_OSSRState_Enable;
TIM1_BDTRInitStructure.TIM1_OSSIState = TIM1_OSSIState_Enable;
TIM1_BDTRInitStructure.TIM1_LOCKLevel = TIM1_LOCKLevel_1;
TIM1_BDTRInitStructure.TIM1_DeadTime = 0x05;
TIM1_BDTRInitStructure.TIM1_Break = TIM1_Break_Enable;
TIM1_BDTRInitStructure.TIM1_BreakPolarity =
TIM1_BreakPolarity_High;
TIM1_BDTRInitStructure.TIM1_AutomaticOutput =
TIM1_AutomaticOutput_Enable;
```

TIM1_BDTRConfig(&TIM1_BDTRInitStructure);

20.2.9 TIM1_ICInit 函数

表 590 描述了 TIM1_ICInit 函数。

表 590. TIM1_ICInit 函数

函数名	TIM1_ICInit
函数原型	void TIM1_ICInit(TIM1_ICInitTypeDef* TIM1_ICInitStruct)
行为描述	通过指明 TIM1_ICInitStruct 中的参数初始化 TIM1 外围设备。
输入参数	TIM1_ICInitStruct：指向一个包含特定 TIM1 外围设备的配置信息的 TIM1_ICInitTypeDef 结构体。 关于这个参数可用的值的更多详细信息请参阅章节: TIM1_ICInitTypeDef 结构体。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

TIM1_ICInitTypeDef 结构体

TIM1_ICInitTypeDef 结构体被定义在 stm32f10x_tim1.h 文件中。

```
typedef struct
{
  u16 TIM1_Channel;
  u16 TIM1_ICPolarity;
  u16 TIM1_ICSelection;
  u16 TIM1_ICPrescaler;
  u8 TIM1_ICFilter;
} TIM1_ICInitTypeDef;
```

TIM1_Channel

TIM1_Channel 选择 TIM1 通道。这个成员可以被设置为下列值之一。

表 591. TIM1_Channel 定义

TIM1_Channel	描述
TIM1_Channel_1	TIM1 通道 1 被使用
TIM1_Channel_2	TIM1 通道 2 被使用
TIM1_Channel_3	TIM1 通道 3 被使用
TIM1_Channel_4	TIM1 通道 4 被使用

TIM1_ICPolarity

TIM1_ICPolarity 选择输入信号的活动沿。这个成员可以被设置为下列值之一。

表 592. TIM1_ICPolarity 定义

TIM1_ICPolarity	描述

TIM1_ICPolarity_Rising	TIM1 输入上升沿捕获
TIM1_ICPolarity_Falling	TIM1 输入下降沿捕获

TIM1_ICSelection

TIM1_ICSelection 选择被使用的输入。这个成员可以被设置为下列值之一。

表 593. TIM1_ICSelection 定义

TIM1_ICSelection	描述
TIM1_ICSelection_DirectTI	TIM1 的 1 ,2 或 3 或 4 输入被选择分别连接到IC1或IC2或IC3或IC4
TIM1_ICSelection_IndirectTI	TIM1 的 1 ,2 或 3 或 4 输入被选择分别连接到IC2或IC1或IC4或IC3
TIM1_ICSelection_TRGI	TIM1 的 1 ,2 或 3 或 4 输入被选择分别连接到 TRGI

TIM1_Icprescaler

TIM1_Icprescaler 配置输入捕获预分频器。这个成员可以被设置为下列值之一。

表 594. TIM1_Icprescaler 定义

TIM1_Icprescaler	描述
TIM1_ICPSC_DIV1	在每次捕获输入中探测到信号沿的时候进行捕获
TIM1_ICPSC_DIV2	每 2 次事件发生捕获执行一次

TIM1_ICPSC_DIV4	每 4 次事件发生捕获执行一次
TIM1_ICPSC_DIV8	每 8 次事件发生捕获执行一次

TIM1_ICFilter

TIM1_ICFilter 指明输入捕获滤波器。这个成员可以被设置为下列值之一。

例：

```
/* TIM1  输入捕获通道 1 模式配置  */

TIM1_ICInitTypeDef TIM1_ICInitStructure;

TIM1_ICInitStructure.TIM1_Channel = TIM1_Channel_1;

TIM1_ICInitStructure.TIM1_ICPolarity = TIM1_ICPolarity_Falling;

TIM1_ICInitStructure.TIM1_ICSelection = TIM1_ICSelection_DirectTI;

TIM1_ICInitStructure.TIM1_ICPrescaler = TIM1_ICPSC_DIV2;

TIM1_ICInitStructure.TIM1_ICFilter = 0x0;

TIM1_ICInit(&TIM1_ICInitStructure);
```

20.2.10 TIM1_PWMIConfig 函数

表 595 描述了 TIM1_PWMIConfig 函数。

表 595. TIM1_PWMIConfig 函数

函数名	TIM1_PWMIConfig
函数原型	TIM1_PWMIConfig(TIM1_ICInitTypeDef* TIM1_ICInitStruct

行为描述	根据 TIM1_ICInitStruct 中指定的参数将 TIM1 外围设备配置成 PWM 输入模式。
输入参数	TIM1_ICInitStruct：指向一个包含特定 TIM1 外围设备的配置信息的 TIM1_ICInitTypeDef 结构体。 关于这个参数可用的值的更多详细信息请参阅章节: TIM1_ICInitTypeDef 结构体。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* TIM1 PWM 输入通道 1 模式配置 */

TIM1_ICInitTypeDef TIM1_ICInitStructure;

TIM1_ICInitStructure.TIM1_Channel = TIM1_Channel_1;

TIM1_ICInitStructure.TIM1_ICPolarity = TIM1_ICPolarity_Rising;

TIM1_ICInitStructure.TIM1_ICSelection = TIM1_ICSelection_DirectTI;

TIM1_ICInitStructure.TIM1_ICPrescaler = TIM1_ICPSC_DIV1;

TIM1_ICInitStructure.TIM1_ICFilter = 0x0;


TIM1_PWMIConfig(&TIM1_ICInitStructure);
```

20.2.11 TIM1_TimeBaseStructInit 函数

表 596 描述了 TIM1_TimeBaseStructInit 函数。

表 596. TIM1_TimeBaseStructInit 函数

函数名	TIM1_TimeBaseStructInit
函数原型	Void TIM1_TimeBaseStructInit(TIM1_TimeBaseInitTypeDef* TIM1_TimeBaseInitStruct)
行为描述	将 TIM1_TimeBaseInitStruct 中的成员赋为默认值
输入参数	TIM1_ICInitStruct ：指向一个将被初始化的 TIM1_TimeBaseInitTypeDef 结构体。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

TIM1_TimeBaseInitStruct 的成员有以下这些默认值：

表 597. TIM1_TimeBaseInitStruct 默认值

成员	默认值
TIM1_Period	TIM1_Period_Reset_Mask
TIM1_Prescaler	TIM1_Prescaler_Reset_Mask
TIM1_CKD	TIM1_CKD_DIV1

TIM1_CounterMode	TIM1_CounterMode_Up
TIM1_RepetitionCounter	TIM1_RepetitionCounter_Reset_Mask

例

```
/* 下面的例子讲解了如何初始化一个 TIM1_BaseInitTypeDef 结构体 */
TIM1_TimeBaseInitTypeDef TIM1_TimeBaseInitStructure;
TIM1_TimeBaseStructInit(& TIM1_TimeBaseInitStructure);
```

20.2.12 TIM1_OCStructInit 函数

表 598 描述了 TIM1_OCStructInit 函数。

表 598. TIM1_OCStructInit 函数

函数名	TIM1_OCStructInit
函数原型	void TIM1_OCStructInit(TIM1_OCInitTypeDef* TIM1_OCInitStruct)
行为描述	将 TIM1_OCInitStruct 中的成员赋为默认值
输入参数	TIM1_OCInitStruct：指向一个将被初始化的 TIM1_OCInitTypeDef 结构体。
输出参数	无
返回参数（值）	无
前提条件	无

调用函数	无

TIM1_OCInitStruct 的成员有以下这些默认值：

表 599 TIM1_TimeBaseInitStruct 默认值

成员	默认值
TIM1_OCMode	TIM1_OCMode_Timing
TIM1_OutputState	TIM1_OutputState_Disable
TIM1_OutputNState	TIM1_OutputNState_Disable
TIM1_Pulse	TIM1_Pulse_Reset_Mask
TIM1_OCPolarity	TIM1_OCPolarity_High
TIM1_OCNPolarity	TIM1_OCPolarity_High
TIM1_OCIdleState	TIM1_OCIdleState_Reset
TIM1_OCNIdleState	TIM1_OCNIdleState_Reset

例：

/*下面的例子讲解了如何初始化一个 TIM1_OCInitTypeDef 结构体 */

TIM1_OCInitTypeDef TIM1_OCInitStructure;

TIM1_OCStructInit(& TIM1_OCInitStructure);

20.2.13 TIM1_ICStructInit 函数

表 600 描述了 TIM1_ICStructInit 函数。

表 600. TIM1_ICStructInit 函数

函数名	TIM1_ICStructInit
函数原型	void TIM1_ICStructInit(TIM1_ICInitTypeDef* TIM1_ICInitStruct)
行为描述	将 TIM1_ICInitStruct 中的成员赋为默认值
输入参数	TIM1_ICInitStruct ：指向一个将被初始化的 TIM1_ICInitTypeDef 结构体。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

TIM1_ICInitStruct 的成员有以下这些默认值：

表 601.TIM1_TimeBaseInitStruct 默认值

成员	默认值
TIM1_Channel	TIM1_Channel_1
TIM1_ICSelection	TIM1_ICSelection_DirectT
TIM1_ICPolarity	TIM1_ICPolarity_Rising
TIM1_ICPrescaler	TIM1_ICPSC_DIV1

TIM1_ICFilter	TIM1_ICFilter_Mask

例：

/*下面的例子讲解了如何初始化一个 TIM1_ICInitTypeDef 结构体*/

TIM1_ICInitTypeDef TIM1_ICInitStructure;

TIM1_ICStructInit(& TIM1_ICInitStructure);

20.2.14 TIM1_BDRAStructInit 函数

表 602 描述了 TIM1_BDRAStructInit 函数。

表 602. TIM1_BDRAStructInit 函数

函数名	TIM1_BDRAStructInit
函数原型	void TIM1_BDTRStructInit(TIM1_BDTRInitTypeDef* TIM1_BDTRInitStruct)
行为描述	将 TIM1_BDTRStruct 中的成员赋为默认值
输入参数	TIM1_BDTRInitStruct：指向一个将被初始化的 TIM1_BDTRInitStruct 结构体。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

TIM1_BDTRInitStruct 的成员有以下这些默认值：

表 603.TIM1_TimeBaseInitStruct 默认值

成员	默认值
TIM1_Channel	TIM1_Channel_1
TIM1_ICSelection	TIM1_ICSelection_DirectT
TIM1_ICPolarity	TIM1_ICPolarity_Rising
TIM1_ICPrescaler	TIM1_ICPSC_DIV1
TIM1_ICFilter	TIM1_ICFilter_Mask

例：

/*下面的例子讲解了如何初始化一个 TIM1_ BDTRInitTypeDef 结构体*/

```
TIM1_BDTRInitTypeDef TIM1_BDTRInitStructure;

TIM1_BDTRStructInit(& TIM1_BDTRInitStructure);
```

20.2.15 TIM1_Cmd 函数

表 604 描述了 TIM1_Cmd 函数。

表 604. TIM1_Cmd 函数

函数名	TIM1_Cmd
函数原型	void TIM1_Cmd(FunctionalState NewState)
行为描述	使能或禁用特定的外围设备
输入参数	NewState：TIM1 外围设备的新状态。 这个参数能够取：ENABLE 或者 DISABLE

输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 使能 TIM1 计数器*/
TIM1_Cmd(ENABLE);
```

20.2.16 TIM1_CtrlPWMOutputs 函数

表 605 描述了 TIM1_CtrlPWMOutputs 函数。

表 605. TIM1_CtrlPWMOutputs 函数

函数名	TIM1_CtrlPWMOutputs
函数原型	void TIM1_CtrlPWMOutputs(FunctionalState Newstate
行为描述	使能或禁用 TIM1 外设主输出
输入参数	Newstate：TIM1 外围设备主输出的新状态。 这个参数能够取：ENABLE 或者 DISABLE
输出参数	无
返回参数（值）	无
前提条件	无

调用函数	无

例：

```
/* 使能 TIM1 外设主输出 */
TIM1_CtrlPWMOutputs(ENABLE);
```

20.2.17 TIM1_ITconfig 函数

表 606 描述了 TIM1_ITconfig 函数。

表 606. TIM1_ITconfig 函数

函数名	TIM1_ITconfig
函数原型	void TIM1_ITConfig(u16 TIM1_IT, FunctionalState NewState)
行为描述	使能会阻止特定 TIM1 中断
输入参数 1	TIM1_IT:TIM1 中断源能被使能或禁用 更多关于该参数的可用值的细节请参阅章节: TIM1_IT。
输入参数 2	NewState：特定 TIM1 中断的新状态。 这个参数能够取：ENABLE 或者 DISABLE
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

TIM1_IT

TIM1_IT 使能或禁用 TIM1 中断。可以是用下列值中的一个或者组合：

表 607. TIM1_IT 值

TIM1_IT	描述
TIM1_IT_Update	TIM1 更新中断源
TIM1_IT_CC1	TIM1 捕获/比较 1 中断源
TIM1_IT_CC2	TIM1 捕获/比较 2 中断源
TIM1_IT_CC3	TIM1 捕获/比较 3 中断源
TIM1_IT_CC4	TIM1 捕获/比较 4 中断源
TIM1_IT_COM	TIM1 COM 中断源
TIM1_IT_Trigger	TIM1 触发中断源
TIM1_IT_BRK	TIM1 间断中断源

例：

```
/* 使能 TIM1 捕获/比较 1 中断源*/
TIM1_ITConfig(TIM1_IT_CC1, ENABLE );
```

20.2.18 TIM1_DMAConfig 函数

表 608 描述了 TIM1_DMAConfig 函数。

表 608. TIM1_DMAConfig 函数

函数名	TIM1_DMAConfig

函数原型	void TIM1_DMAConfig(u8 TIM1_DMABase, u16 TIM1_DMABurstLength)
行为描述	配置 TIM1 的 DMA 接口
输入参数 1	TIM1_DMABase:DMA 基础地址 更多关于该参数的可用值的细节请参阅章节: TIM1_DMABase。
输入参数 2	TIM1_DMABurstLength：DMA 脉冲长度 更多关于该参数的可用值的细节请参阅章节: TIM1_DMABurstLength。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

TIM1_DMABase

TIM1_DMABase 选择 TIM1 DMA 基址。

表 599 TIM1_TimeBaseInitStruct 默认值

成员	默认值
TIM1_DMABase_CR1	CR1 寄存器用作 DMA 基址
TIM1_DMABase_CR2	CR2 寄存器用作 DMA 基址
TIM1_DMABase_SMCR	SMCR 寄存器用作 DMA 基址

TIM1_DMABase_DIER	DIER 寄存器用作 DMA 基址
TIM1_DMABase_SR	SR 寄存器用作 DMA 基址
TIM1_DMABase_EGR	EGR 寄存器用作 DMA 基址
TIM1_DMABase_CCMR1	CCMR1 寄存器用作 DMA 基址
TIM1_DMABase_CCMR2	CCMR2 寄存器用作 DMA 基址
TIM1_DMABase_CCER	CCER 寄存器用作 DMA 基址
TIM1_DMABase_CNT	CNT 寄存器用作 DMA 基址
TIM1_DMABase_PSC	PSC 寄存器用作 DMA 基址
TIM1_DMABase_ARR	ARR 寄存器用作 DMA 基址
TIM1_DMABase_RCR	RCR 寄存器用作 DMA 基址
TIM1_DMABase_CCR1	CCR1 寄存器用作 DMA 基址
TIM1_DMABase_CCR2	CCR2 寄存器用作 DMA 基址
TIM1_DMABase_CCR3	CCR3 寄存器用作 DMA 基址
TIM1_DMABase_CCR4	CCR4 寄存器用作 DMA 基址
TIM1_DMABase_BDTR	BDTR 寄存器用作 DMA 基址
TIM1_DMABase_DCR	DCR 寄存器用作 DMA 基址

TIM1_DMABurstLength

这个参数配置 TIM1 DMA 脉冲长度（见表 610）。

表 610 .TIM1_DMABurstLength 值

TIM1_DMABurstLength	描述
TIM1_DMABurstLength_1Byte	DMA 脉冲长度 1b
TIM1_DMABurstLength_2Bytes	DMA 脉冲长度 2b
TIM1_DMABurstLength_3Bytes	DMA 脉冲长度 3b
TIM1_DMABurstLength_4Bytes	DMA 脉冲长度 4b
TIM1_DMABurstLength_5Bytes	DMA 脉冲长度 5b
TIM1_DMABurstLength_6Bytes	DMA 脉冲长度 6b
TIM1_DMABurstLength_7Bytes	DMA 脉冲长度 7b
TIM1_DMABurstLength_8Bytes	DMA 脉冲长度 8b
TIM1_DMABurstLength_9Bytes	DMA 脉冲长度 9b
TIM1_DMABurstLength_10Bytes	DMA 脉冲长度 10b
TIM1_DMABurstLength_11Bytes	DMA 脉冲长度 11b
TIM1_DMABurstLength_12Bytes	DMA 脉冲长度 12b
TIM1_DMABurstLength_13Bytes	DMA 脉冲长度 13b
TIM1_DMABurstLength_14Bytes	DMA 脉冲长度 14b
TIM1_DMABurstLength_15Bytes	DMA 脉冲长度 15b
TIM1_DMABurstLength_16Bytes	DMA 脉冲长度 16b
TIM1_DMABurstLength_17Bytes	DMA 脉冲长度 17b
TIM1_DMABurstLength_18Bytes	DMA 脉冲长度 18b

例：

/* 配置 TIM1 DMA 接口来传输 1 字节，使用 CCR1 作为基址*/

TIM1_DMAConfig(TIM1_DMABase_CCR1, TIM1_DMABurstLength_1Byte)

20.2.19 TIM1_DMACmd 函数

表 611 描述了 TIM1_DMACmd 函数。

表 611. TIM1_DMACmd 函数

函数名	TIM1_DMACmd
函数原型	void TIM1_DMACmd(u16 TIM1_DMASource, FunctionalState Newstate)
行为描述	使能或禁用 TIM1 的 DMA 请求
输入参数 1	TIM1_DMASource：DMA 请求源 更多关于该参数的可用值的细节请参阅章节: TIM1_DMASource。
输入参数 2	Newstate：DMA 请求源的新状态 这个参数能够取：ENABLE 或者 DISABLE。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

TIM1_DMASource

TIM1_DMASource 选择 TIM1 DMA 请求源。可以是用下列值中的一个或者组合：

表 612 TIM1_DMASource 值

TIM1_DMASource	描述
TIM1_DMA_Update	TIM1 更新 DMA 源
TIM1_DMA_CC1	TIM1 捕获/比较 1 DMA 源
TIM1_DMA_CC2	TIM1 捕获/比较 2 DMA 源
TIM1_DMA_CC3	TIM1 捕获/比较 3 DMA 源
TIM1_DMA_CC4	TIM1 捕获/比较 4 DMA 源
TIM1_DMA_COM	TIM1 COM DMA 源
TIM1_DMA_Trigger	TIM1 触发器 DMA 源

例：

```
/* TIM1 捕获比较 1 DMA 请求配置 */
TIM1_DMACmd(TIM1_DMA_CC1, ENABLE);
```

20.2.20 TIM1_InternalClockConfig 函数

表 613 描述了 TIM1_InternalClockConfig 函数。

表 613 .TIM1_InternalClockConfig 函数

函数名	TIM1_InternalClockConfig
函数原型	void TIM1_InternalClockConfig(void)
行为描述	配置 TIM1 的内部时钟
输入参数	无
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 为 TIM1 选择内部时钟 */
TIM1_InternalClockConfig();
```

20.2.21 TIM1_ETRClockMode1Config 函数

表 614 描述了 TIM1_ETRClockMode1Config 函数。

表 614 . TIM1_ETRClockMode1Config 函数

函数名	TIM1_ETRClockMode1Config
函数原型	Void TIM1_ETRClockMode1Config (u16 TIM1_ExtTRGPrescaler, u16 TIM1_ExtTRGPolarity, u16 ExtTRGFilter)
行为描述	配置 TIM1 的外部时钟模式 1
输入参数 1	TIM1_ExtTRGPrescaler：外部触发预分频器

	更多关于该参数的可用值的细节请参阅章节：TIM1_ExtTRGPrescaler。
输入参数 2	TIM1_ExtTRGPolarity：外部时钟极性 更多关于该参数的可用值的细节请参阅章节：TIM1_ExtTRGPolarity。
输入参数 3	ExtTRGFilter：指定外部触发滤波器，这个成员能够是）0x0 到 0xF 之间的值
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

TIM1_ExTRGPrescaler

TIM1_ExTRGPrescaler 选择外部的触发器预分频器。这个成员能够被设置成下列值之一：

表 615. TIM1_ExTRGPrescaler 值

TIM1_ExTRGPrescaler	描述
TIM1_ExtTRGPSC_OFF	预分频器关闭
TIM1_ExtTRGPSC_DIV2	ETRP 频率除 2
TIM1_ExtTRGPSC_DIV4	ETRP 频率除 4
TIM1_ExtTRGPSC_DIV8	ETRP 频率除 8

TIM1_ExtTRGPolarity

TIM1_ExtTRGPolarity 配置外部触发器极性。这个成员可以被设置成下列值之一。

表 616. TIM1_ExtTRGPolarity 值

TIM1_ExtTRGPolarity	描述
TIM1_ExtTRGPolarity_Inverted	外部触发极性转换：低触发或下降沿触发
TIM1_ExtTRGPolarity_NonInverted	外部触发极性转换：高触发或上升沿触发

例：

/* 为 TIM1 选择外部时钟模式 1，外部时钟连接到 ETR 输入引脚，上升沿是活动沿，无过滤器采样(ExtTRGFilter = 0) 预分频器选定为 TIM1_ExtTRGPSC_DIV2 */

TIM1_ExternalCLK1Config(TIM1_ExtTRGPSC_DIV2,

TIM1_ExtTRGPolarity_NonInverted, 0x0);

20.2.22 TIM1_ETRClockMode2Config 函数

表 617 描述了 TIM1_ETRClockMode2Config 函数。

表 617 . TIM1_ETRClockMode2Config 函数

函数名	TIM1_ETRClockMode2Config
函数原型	void TIM1_ETRClockMode2Config(u16 TIM1_ExtTRGPrescaler, u16 TIM1_ExtTRGPolarity, u16 ExtTRGFilter)

行为描述	配置 TIM1 的外部时钟模式 2
输入参数 2	TIM1_ExtTRGPrescaler：指定外部触发预分频器 更多关于该参数的可用值的细节请参阅章节：TIM1_ExtTRGPrescaler。
输入参数 3	TIM1_ExtTRGPolarity：指定外部时钟极性 更多关于该参数的可用值的细节请参阅章节：TIM1_ExtTRGPolarity。
输入参数 4	ExtTRGFilter：指定外部触发滤波器，这个成员能够是）0x0 到 0xF 之间的值
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 为 TIM1 选择外部时钟模式 2，外部时钟连接到 ETR 输入引脚，上升沿是活动沿，无过滤器采样(ExtTRGFilter = 0) 预分频器选定为 TIM1_ExtTRGPSC_DIV2 */
TIM1_ExternalCLK2Config(TIM1_ExtTRGPSC_DIV2,
TIM1_ExtTRGPolarity_NonInverted, 0x0);
```

20.2.23 TIM1_ETRConfig 函数

表 618 描述了 TIM1_ETRConfig 函数。

表 618 . TIM1_ETRConfig 函数

函数名	TIM1_ETRConfig
函数原型	void TIM1_ETRConfig(u16 TIM1_ExtTRGPrescaler, u16 TIM1_ExtTRGPolarity, u8 ExtTRGFilter)
行为描述	配置 TIM1 的外部触发器（ETR）
输入参数 1	TIM1_ExtTRGPrescaler：指定外部触发预分频器 更多关于该参数的可用值的细节请参阅章节：TIM1_ExtTRGPrescaler。
输入参数 2	TIM1_ExtTRGPolarity：指定外部时钟极性 更多关于该参数的可用值的细节请参阅章节：TIM1_ExtTRGPolarity。
输入参数 3	ExtTRGFilter：指定外部触发滤波器，这个成员能够取 0x0 到 0xF 之间的值
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

/*为 TIM1 配置外部触发器(ETR)，外部时钟连接到 ETR 输入引脚，上升沿是活动沿，无过滤器采样(ExtTRGFilter = 0) 预分频器选定为 TIM1_ExtTRGPSC_DIV2 */

TIM1_ExternalCLK2Config(TIM1_ExtTRGPSC_DIV2,

TIM1_ExtTRGPolarity_NonInverted, 0x0);

20.2.24 TIM1_ITRxExternalClockConfig 函数

表 619 描述了 TIM1_ITRxExternalClockConfig 函数。

表 619 . TIM1_ITRxExternalClockConfig 函数

函数名	TIM1_ITRxExternalClockConfig
函数原型	void TIM1_ITRxExternalClockConfig(u16 TIM1_InputTriggerSource)
行为描述	将 TIM1 内部触发器配置为外部时钟
输入参数 2	TIM1_InputTriggerSource：输入触发器源 更多关于该参数的可用值的细节请参阅章节: TIM1_InputTriggerSource。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

TIM1_InputTriggerSource

TIM1_InputTriggerSource 选择 TIM1 输入触发器（见表 620）。

表 620. TIM1_InputTriggerSource 值

TIM1_InputTriggerSource	描述
TIM1_TS_ITR0	TIM1 内部触发器 1
TIM1_TS_ITR1	TIM1 内部触发器 2
TIM1_TS_ITR2	TIM1 内部触发器 3
TIM1_TS_ITR3	TIM1 内部触发器 4

例：

```
/* TIM1 内部触发器 3 用作时钟源 */
TIM1_ITRxExternalClockConfig(TIM1_TS_ITR3);
```

20.2.25 TIM1_TixExternalClockConfig 函数

表 621 描述了 TIM1_TixExternalClockConfig 函数。

表 621. TIM1_TixExternalClockConfig 函数

函数名	TIM1_TixExternalClockConfig
函数原型	void TIM1_TIxExternalClockConfig(u16 TIM1_TIxExternalCLKSource, u16 TIM1_ICPolarity, u16 ICFilter)

行为描述	将 TIM1 输入触发器配置为外部时钟
输入参数 1	TIM1_TIxExternalCLKSource：触发源 更多关于该参数的可用值的细节请参阅章节：TIM1_TIxExternalCLKSource。
输入参数 2	TIM1_ICPolarit：T1 极性 更多关于该参数的可用值的细节请参阅章节：TIM1_ICPolarit。
输入参数 3	ExtTRGFilter：指定输入捕获滤波器，这个成员能够是 0x0 到 0xF 之间的值
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

TIM1_TIcExternalCLKSource

TIM1_TIcExternalCLKSource 选择 TIM1 TIx 外部时钟资源。可以使用下列值中的一个或组合：

表 622. TIM1_TIcExternalCLKSource 值

TIM1_TIcExternalCLKSource	描述
TIM1_TS_TI1FP1	IC1 被映射到 TI1
TIM1_TS_TI2FP2	IC2 被映射到 TI2

TIM1_TS_TI1F_ED	IC1 被映射到 TI1：使用了边沿检波器

例：

/*选择 TI1 作为 TIM1 的时：外部时钟连接到 TI1 输入引脚，上升沿是有效沿无过滤器采样 (ICFilter = 0) */

TIM1_TIxExternalClockConfig(TIM1_TS_TI1FP1, TIM1_ICPolarity_Rising,

0);

20.2.26 TIM1_SelectInputTrigger 函数

表 623 描述了 TIM1_SelectInputTrigger 函数。

表 623. TIM1_SelectInputTrigger 函数

函数名	TIM1_SelectInputTrigger
函数原型	void TIM1_SelectInputTrigger(u16 TIM1_InputTriggerSource)
行为描述	选择 TIM1 输入触发器源
输入参数	TIM1_InputTriggerSource：输入触发器源 更多关于该参数的可用值的细节请参阅章节：TIM_InputTriggerSource。
输出参数	无
返回参数（值）	无
前提条件	无

调用函数	无

TIM1_InputTriggerSource

TIM1_InputTriggerSource 选择 TIM1 输入触发器源。这个成员能够被设置成下列值之一。

表 624. TIM1_InputTriggerSource 值

TIM1_InputTriggerSource	描述
TIM1_TS_ITR0	TIM1 内部触发器 0
TIM1_TS_ITR1	TIM1 内部触发器 1
TIM1_TS_ITR2	TIM1 内部触发器 2
TIM1_TS_ITR3	TIM1 内部触发器 3
TIM1_TS_TI1F_ED	TIM1 TI1 边沿检波器
TIM1_TS_TI1FP1	TIM1 过滤定时器输入 1
TIM1_TS_TI2FP2	TIM1 过滤定时器输入 2
TIM1_TS_ETRF	TIM1 外部触发器输入

例：

```
/* 选择内部触发器 3 作为 TIM1 的输入触发器 */
void TIM1_SelectInputTrigger(TIM1_TS_ITR3);
```

20.2.27 TIM1_UpdateDisableConfig 函数

表 625 描述了 TIM1_UpdateDisableConfig 函数。

表 625. TIM1_UpdateDisableConfig 函数

函数名	TIM1_UpdateDisableConfig
函数原型	void TIM1_UpdateDisableConfig(FunctionalState Newstate)
行为描述	使能或禁用 TIM1 更新事件
输入参数	Newstate：TIM1_CR1 的 UDIS 位的新状态 这个参数能够取：ENABLE 或者 DISABLE
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 为 TIM1 使能更新事件 */
TIM1_UpdateDisableConfig(DISABLE);
```

20.2.28 TIM1_UpdateRequestConfig 函数

表 626 描述了 TIM1_UpdateRequestConfig 函数。

表 626. TIM1_UpdateRequestConfig 函数

函数名	TIM1_UpdateRequestConfig
函数原型	void TIM1_UpdateRequestConfig(

	u8 TIM1_UpdateSource)
行为描述	选择 TIM1 更新请求源
输入参数	TIM1_UpdateSource：更新请求源 更多关于该参数的可用值的细节请参阅章节: TIM1_UpdateSource。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

TIM1_UpdateSourse

TIM1_UpdateSourse 选择 TIM1 更新源（见表 627）

表 627. TIM1_UpdateSourse 值

TIM1_UpdateSourse	描述
TIM1_UpdateSource_Global	更新源是计数器上溢/下溢，UG 位置位或通过从模式控制器产生的更新
TIM1_UpdateSource_Regular	更新源是计数器上溢/下溢

例：

```
/* 为 TIM1 选择有序更新源 */

TIM1_UpdateRequestConfig(TIM1_UpdateSource_Regular);
```

20.2.29 TIM1_SelectHallSensor 函数

表 628 描述了 TIM1_SelectHallSensor 函数。

表 628. TIM1_SelectHallSensor 函数

函数名	TIM1_SelectHallSensor
函数原型	void TIM1_SelectHallSensor(FunctionalState Newstate)
行为描述	使能或禁用 TIM1 的霍耳传感器接口
输入参数	Newstate :在 TIM1_CR2 中的 TI1S 位的新状态 这个参数能够取：ENABLE 或者 DISABLE
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 为 TIM1 选择霍耳传感器接口 */
TIM1_SelectHallSensor(ENABLE);
```

20.2.30 TIM1_SelectOnePulseMode 函数

表 629 描述了 TIM1_SelectOnePulseMode 函数。

表 629. TIM1_SelectOnePulseMode 函数

函数名	TIM1_SelectOnePulseMode
函数原型	void TIM1_SelectOnePulseMode(u16 TIM1_OPMode)
行为描述	使能或禁用 TIM1 的单脉冲模式
输入参数	TIM1_OPMode：指明要被使用的一脉冲模式 更多关于该参数的可用值的细节请参阅章节: TIM1_OPMode。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

TIM1_OPMode

TIM1_OPMode 选择 TIM1 更新源（见表 630）。

表 630. TIM1_OPMode 值

TIM1_OPMode	描述
TIM1_OPMode_Single	TIM1 单一脉冲模式
TIM1_OPMode_Repetitive	TIM1 循环脉冲模式

例：

```
/*为 TIM1 选择单一脉冲模式 */
TIM1_SelectOnePulseMode(TIM1_OPMode_Single);
```

20.2.31 TIM1_SelectOutputTrigger 函数

表 631 描述了 TIM1_SelectOutputTrigger 函数。

表 631. TIM1_SelectOutputTrigger 函数

函数名	TIM1_SelectOutputTrigger
函数原型	Void TIM1_SelectOutputTrigger(u16 TIM1_TRGOSource)
行为描述	选择 TIM1 触发器的输出模式
输入参数	TIM1_TRGOSource：TRGO 源 更多关于该参数的可用值的细节请参阅章节: TIM1_TRGOSource。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

TIM1_TRGOSourse

TIM1_TRGOSourse 选择 TIM1 TRGO 源（见表 632）

表 632. TIM1_TRGOSourse 值

TIM1_TRGOSourse	描述
TIM1_TRGOSource_Reset	来自 TIM1_EGR 寄存器的 UG 位被用作触发器输出（TRGO）
TIM1_TRGOSource_Enable	计数器使能 CEN 被用作触发器输出（TRGO）
TIM1_TRGOSource_Update	更新事件被选作触发器输出（TRGO）
TIM1_TRGOSource_OC1	一旦发生了一次捕获或者比较匹配，当 CC1IF 标记要被置位时，触发器输出发送一个正脉冲（TRGO）。 The trigger output send a positive pulse when the CC1IF flag is to be set, as soon as a capture or a compare match occurred.
TIM1_TRGOSource_OC1Ref	OC1REF 信号被用作触发器输出（TRGO）
TIM1_TRGOSource_OC2Ref	OC2REF 信号被用作触发器输出（TRGO）
TIM1_TRGOSource_OC3Ref	OC3REF 信号被用作触发器输出（TRGO）
TIM1_TRGOSource_OC4Ref	OC4REF 信号被用作触发器输出（TRGO）

例：

```
/* 为 TIM1 选择更新事件作为 TRGO */
TIM1_SelectOutputTrigger(TIM1_TRGOSource_Update);
```

20.2.32 TIM1_SelectSlaveMode 函数

表 633 描述了 TIM1_SelectSlaveMode 函数。

表 633. TIM1_SelectSlaveMode 函数

函数名	TIM1_SelectSlaveMode
函数原型	void TIM1_SelectSlaveMode(u16 TIM1_SlaveMode)
行为描述	选择 TIM1 从模式
输入参数	TIM1_SlaveMode：TIM1 从模式 更多关于该参数的可用值的细节请参阅章节: TIM1_SlaveMode。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

TIM1_SlaveMode

TIM1_SlaveMode 选择 TIM1 从模式（见表 634）

表 634. TIM1_SlaveMode 定义

TIM1_SlaveMode	描述
TIM1_SlaveMode_Reset	被选择的触发器信号（TRGI）的上升沿重新初始化计数器并且触发一个寄存器更新事件
TIM1_SlaveMode_Gated	当触发器信号为高，计数器时钟被使能

TIM1_SlaveMode_Trigger	计数器在触发器 TRGI 的上升沿时启动
TIM1_SlaveMode_External1	被选的触发器的上升沿为计数器提供时钟

例：

```
/* 为 TIM1 选择门控模式作为从模式 */
TIM1_SelectSlaveMode(TIM1_SlaveMode_Gated);
```

20.2.33 TIM1_SelectMasterSlaveMode 函数

表 635 描述了 TIM1_SelectMasterSlaveMode 函数。

表 635. TIM1_SelectMasterSlaveMode 函数

函数名	TIM1_SelectMasterSlaveMode
函数原型	void TIM1_SelectMasterSlaveMode(u16 TIM1_MasterSlaveMode)
行为描述	设置或重设 TIM1 的主/从模式
输入参数	TIM1_MasterSlaveMode：定时器主从模式 更多关于该参数的可用值的细节请参阅章节: TIM_MasterSlaveMode。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

TIM1_MasterSlaveMode

TIM1_MasterSlaveMode 选择 TIMx 主从模式（见表 636）

表 636. TIM1_MasterSlaveMode 定义

TIM1_MasterSlaveMode	描述
TIM1_MasterSlaveMode_Enable	使能主从模式
TIM1_MasterSlaveMode_Disable	禁用主从模式

例：

```
/* 为 TIM2s 使能主从模式 */
TIM1_SelectMasterSlaveMode(TIM2, TIM1_MasterSlaveMode_Enable);
```

20.2.34 TIM1_EncoderInterfaceConfig 函数

表 637 描述了 TIM1_EncoderInterfaceConfig 函数。

表 637. TIM1_EncoderInterfaceConfig 函数

函数名	TIM1_EncoderInterfaceConfig
函数原型	void TIM1_EncoderInterfaceConfig(u16 TIM1_EncoderMode, u16 TIM1_IC1Polarity, u16 TIM1_IC2Polarity)
行为描述	配置 TIM1 编码器接口
输入参数 1	TIM1_EncoderMode：TIM1 编码器模式 更多关于该参数的可用值的细节请参阅章节: TIM1_EncoderMode。
输入参数 2	TIM1_IC1Polarity：TI1 极性

	更多关于该参数的可用值的细节请参阅章节: TIM1_IC1Polarity。
输入参数 3	TIM1_IC2Polarity：TI2 极性 更多关于该参数的可用值的细节请参阅章节: TIM1_IC2Polarity。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

TIM1_EncoderMode

TIM1_EncoderMode 选择 TIM1 编码器模式（见表 638）

表 638. TIM1_EncoderMode 定义

TIM1_EncoderMode	描述
TIM1_EncoderMode_TI1	使用 TIM1 编码器模式 1
TIM1_EncoderMode_TI2	使用 TIM1 编码器模式 2
TIM1_EncoderMode_TI12	使用 TIM1 编码器模式 3

例：

```
/* TIM1 编码器接口的使用*/
TIM1_EncoderInterfaceConfig(TIM1_EncoderMode_1,
TIM1_ICPolarity_Rising,
```

TIM1_ICPolarity_Rising);

20.2.35 TIM1_PrescalerConfig 函数

表 639 描述了 TIM1_PrescalerConfig 函数。

表 639. TIM1_PrescalerConfig 函数

函数名	TIM1_PrescalerConfig
函数原型	void TIM1_PrescalerConfig(u16 Prescaler, u16 TIM1_PSCReloadMode)
行为描述	配置 TIM1 预分频器
输入参数 1	Prescaler: 新的 TIM1 预分频器值
输入参数 2	TIM1_PSCReloadMode：TIM1 预分频器重载模式 更多关于该参数的可用值的细节请参阅章节: TIM1_PSCReloadMode。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

TIM1_PSCRloadMode

TIM1_PSCRloadMode 使用下列的值之一：

表 640. TIM1_PSCRloadMode 值

TIM1_PSCRloadMode	描述
TIM1_PSCReloadMode_Update	预分频器在更新事件发生时被载入
TIM1_PSCReloadMode_Immediate	预分频器立刻被载入

例：

```
/* 设置新的 TIM1 预分频器值 */
u16 TIM1Prescaler = 0xFF00;
TIM1_SetPrescaler(TIM1Prescaler, TIM1_PSCReloadMode_Update);
```

20.2.36 TIM1_CounterModeConfig 函数

表 641 描述了 TIM1_CounterModeConfig 函数。

表 641. TIM1_CounterModeConfig 函数

函数名	TIM1_CounterModeConfig
函数原型	void TIM1_CounterModeConfig(u16 TIM1_CounterMode)
行为描述	指定将被使用的 TIM1 计数器模式
输入参数 1	TIM1_CounterMode：将被使用的计数器模式 更多关于该参数的可用值的细节请参阅章节: TIM1_CounterMode。
输出参数	无
返回参数（值）	无

前提条件	无
调用函数	无

例：

/* 为 TIM1 选择中间对齐计数器模式 */

TIM1_CounterModeConfig(TIM1_Counter_CenterAligned1);

20.2.37 TIM1_ForcedOC1Config 函数

表 642 描述 TIM1_ForcedOC1Config 函数。

表 642. TIM1_ForcedOC1Config 函数

函数名	TIM1_ForcedOC1Config
函数原型	void TIM1_ForcedOC1Config(u16 TIM1_ForcedAction)
行为描述	强制 TIM1 信道 1 的输出波形到有效电平或者无效电平
输入参数 1	TIM1_ForcedAction :指定对输出波形的强制行为 更多关于该参数的可用值的细节请参阅章节: TIM1_ForcedAction。
输出参数	无
返回参数（值）	无
前提条件	无

调用函数	无

TIM1_ForcedAction

被迫行为可以是表 643 中的：

表 643. TIM1_ForcedAction 值

TIM1_ForcedAction	描述
TIM1_ForcedAction_Active	强制 OCxREF 为有效电平
TIM1_ForcedAction_InActive	强制 OCxREF 为无效电平

例：

```
/* 强制 TIMI 通道 1 输出为有效电平 */
TIM1_ForcedOC1Config(TIM1_ForcedAction_Active);
```

20.2.38 TIM1_ForcedOC2Config 函数

表 644 描述了 TIM1_ForcedOC2Config 函数。

表 644. TIM1_ForcedOC2Config 函数

函数名	TIM1_ForcedOC2Config
函数原型	void TIM1_ForcedOC2Config(u16 TIM1_ForcedAction)
行为描述	强制 TIM1 信道 2 的输出波形到有效或无效电平。

输入参数 1	TIM1_ForcedAction :指定对输出波形的强制行为 更多关于该参数的可用值的细节请参阅章节: TIM1_ForcedAction。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 强制 TIMI 通道 2 输出为有效电平 */
TIM1_ForcedOC2Config(TIM1_ForcedAction_Active);
```

20.2.39 TIM1_ForcedOC3Config 函数

表 645 描述了 TIM1_ForcedOC3Config 函数。

表 645. TIM1_ForcedOC3Config 函数

函数名	TIM1_ForcedOC3Config
函数原型	void TIM1_ForcedOC3Config(u16 TIM1_ForcedAction)
行为描述	强制 TIM1 信道 3 的输出波形到有效或无效电平。

输入参数 1	TIM1_ForcedAction :指定对输出波形的强制行为 更多关于该参数的可用值的细节请参阅章节: TIM1_ForcedAction。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 强制 TIMI 通道 3 输出为有效电平 */
TIM1_ForcedOC3Config(TIM1_ForcedAction_Active);
```

20.2.40 TIM1_ForcedOC4Config 函数

表 646 描述了 TIM1_ForcedOC4Config 函数。

表 646. TIM1_ForcedOC4Config 函数

函数名	TIM1_ForcedOC4Config
函数原型	void TIM1_ForcedOC4Config(u16 TIM1_ForcedAction)
行为描述	强制 TIM1 信道 4 的输出波形到有效或无效电平。
输入参数 1	TIM1_ForcedAction :指定对输出波形的强制行为

	更多关于该参数的可用值的细节请参阅章节：TIM1_ForcedAction。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

/*强制 TIMI 通道 4 输出为有效级别*/

TIM1_ForcedOC4Config(TIM1_ForcedAction_Active);

20.2.41 TIM1_ARRPreloadConfig 函数

表 647 描述了 TIM1_ARRPreloadConfig 函数。

表 647. TIM1_ARRPreloadConfig 函数

函数名	TIM1_ARRPreloadConfig
函数原型	void TIM1_ARRPreloadConfig(FunctionalState Newstate)
行为描述	使能或禁用 TIM1 外设 ARR 预载寄存器
输入参数 1	Newstate：在 TIM1_CR1 寄存器中的 ARPE 位的新状态 这个参数能够取：ENABLE 或者 DISABLE

输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 使能 ARR 寄存器的 TIM1 预载 */

TIM1_ARRPreloadConfig(ENABLE);
```

20.2.42 TIM1_SelectCOM 函数

表 648 描述了 TIM1_SelectCOM 函数。

表 648. TIM1_SelectCOM 函数

函数名	TIM1_SelectCOM
函数原型	void TIM1_SelectCOM(FunctionalState Newstate)
行为描述	选择 TIM1 外围设备交换事件
输入参数 1	Newstate：交换事件的新状态 这个参数能够取：ENABLE 或者 DISABLE
输出参数	无
返回参数（值）	无
前提条件	无

调用函数	无

例：

/* 选择 TIMI 交换事件 */

TIM1_SelectCOM (ENABLE);

20.2.43 TIM1_SelectCCDMA 函数

表 649 描述了 TIM1_SelectCCDMA 函数。

表 649. TIM1_SelectCCDMA 函数

函数名	TIM1_SelectCCDMA
函数原型	void TIM1_SelectCOM(FunctionalState Newstate)
行为描述	选择 TIM1 外围设备捕获比较 DMA 源
输入参数 1	Newstate：捕获比较 DMA 源的新状态 这个参数能够取：ENABLE 或者 DISABLE
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

/* 选择 TIM1 捕获比较 DMA 源 */

TIM1_SelectCCDMA(ENABLE);

20.2.44 TIM1_CCPreloadControl 函数

表 650 描述了 TIM1_CCPreloadControl 函数。

表 650. TIM1_CCPreloadControl 函数

函数名	TIM1_CCPreloadControl
函数原型	void TIM1_CCPreloadControl(FunctionalState Newstate)
行为描述	设置或重置 TIM1 外围设备捕获比较预载控制位
输入参数 1	Newstate：捕获比较预载控制位的新状态 这个参数能够取：ENABLE 或者 DISABLE
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

/* 选择 TIM1 捕获比较预载控制 */

TIM1_CCPreloadControl(ENABLE);

20.2.45 TIM1_OC1PreLoadConfig 函数

表 651 描述了 TIM1_OC1PreLoadConfig 函数。

表 651. TIM1_OC1PreLoadConfig 函数

函数名	TIM1_OC1PreLoadConfig
函数原型	void TIM1_OC1PreloadConfig(u16 TIM1_OCPreload)
行为描述	使能或禁用 CCR1 上 TIM1 预载寄存器
输入参数	TIM1_OCPreload：输出比较重载状态 更多关于该参数的可用值的细节请参阅章节: TIM1_OCPreload。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

TIM1_OCPreload

输出比较重载状态在表 652 中被列出。

表 652 TIM1_OCPreload 状态

TIM1_OCPreload	描述
TIM1_OCPreload_Enable	CCR1 上 TIM1 重载寄存器使能
TIM1_OCPreload_Dis	CCR1 上 TIM1 重载寄存禁用

able	

例：

/*使能 CC1 寄存器 TIM1 重载 */

TIM1_OC1PreloadConfig(TIM1_OCPreload_Enable);

20.2.46 TIM1_OC2PreloadConfig 函数

表 653 描述了 TIM1_OC2PreloadConfig 函数 。

表 653. TIM1_OC2PreloadConfig 函数

函数名	TIM1_OC2PreloadConfig
函数原型	void TIM1_OC2PreloadConfig(u16 TIM1_OCPreload)
行为描述	使能或禁用 CCR2 上的 TIM1 预载寄存器
输入参数	TIM1_OCPreload：输出比较重载状态 更多关于该参数的可用值的细节请参阅章节: TIM1_OCPreload。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 使能 CC2 寄存器 TIM1 重载*/
TIM1_OC2PreloadConfig(TIM1_OCPreload_Enable);
```

20.2.47 TIM1_OC3PreloadConfig函数

表 654 描述了 TIM1_OC3PreloadConfig 函数

表 654

函数名	TIM1_OC3PreloadConfig
函数原型	void TIM1_OC3PreloadConfig(u16 TIM1_OCPreload)
行为描述	使能或禁用 CCR3 上的 TIM1 预载寄存器
输入参数	TIM1_OCPreload：输出比较预置状态 更多关于该参数的可用值的细节请参阅章节: TIM1_OCPreload。
输出参数	无
返回参数	无
前驱条件	无
调用函数	无

例：

```
/* 使能 CC3 寄存器 TIM1 重载 */
TIM1_OC3PreloadConfig(TIM1_OCPreload_Enable);
```

20.2.48 TIM1_OC4PreloadConfig 函数

表 655 描述了 TIM1_OC4PreloadConfig 函数。

表 655 TIM1_OC4eloadConfig 函数

函数名	TIM1_OC2PreloadConfig
函数原型	void TIM1_OC4reloadConfig(u16 TIM1_OCPreload)
行为描述	使能或禁用 CCR4 上的 TIM1 预载寄存器
输入参数	TIM1_OCPreload：输出比较重载状态 更多关于该参数的可用值的细节请参阅章节: TIM1_OCPreload。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 使能 CC4 寄存器 TIM1 重载 */
TIM1_OC4PreloadConfig(TIM1_OCPreload_Enable);
```

20.2.49 TIM1_OC1FastConfig 函数

表 656 描述了 TIM1_OC1FastConfig 函数。

表 656. TIM1_OC1FastConfig 函数

函数名	TIM1_OC1FastConfig
函数原型	void TIM1_OC1FastConfig(u16 TIM1_OCFast)
行为描述	配置 TIM1 捕获比较 1 快速性能
输入参数	TIM1_OCFast：输出比较快速特性状态 更多关于该参数的可用值的细节请参阅章节: TIM1_OCFast。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

TIM1_OCFast

输出比较重载状态被列在表 657。

表 657.TIM1_OCFast 状态

TIM1_OCFast	描述
TIM1_OCFast_Enable	TIM1 输出比较快速性能使能
TIM1_OCFast_Disable	TIM1 输出比较快速性能禁用

例：

```
/* 使用快速模式的 TIM1 OC1 */
```

TIM1_OC1FastConfig(TIM1_OCFast_Enable);

20.2.50 TIM1_OC2FastConfig 函数

表 658 描述了 TIM1_OC2FastConfig 函数。

表 656. TIM1_OC2astConfig 函数

函数名	TIM1_OC2astConfig
函数原型	void TIM1_OC2astConfig(u16 TIM1_OCFast)
行为描述	配置 TIM1 捕获比较 2 快速性能
输入参数	TIM1_OCFast：输出比较快速特性状态 更多关于该参数的可用值的细节请参阅章节：TIM1_OCFast。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

/*使用快速模式的 TIM1 OC2 */

TIM1_OC2FastConfig(TIM1_OCFast_Enable);

20.2.51 TIM1_OC3FastConfig 函数

表 659 描述了 TIM1_OC3FastConfig 函数。

表 659 TIM1_OC3astConfig 函数

函数名	TIM1_OC3astConfig
函数原型	void TIM1_OC3astConfig(u16 TIM1_OCFast)
行为描述	配置 TIM1 捕获比较 3 快速性能
输入参数	TIM1_OCFast：输出比较快速特性状态 更多关于该参数的可用值的细节请参阅章节：TIM1_OCFast。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/*使用快速模式的 TIM1 OC3 */
TIM1_OC3astConfig(TIM1_OCFast_Enable);
```

20.2.52 TIM1_OC4FastConfig 函数

表 660 描述了 TIM1_OC4FastConfig 函数。

表 660.IM1_OC4astConfig 函数

函数名	TIM1_OC4astConfig

函数原型	void TIM1_OC4astConfig(u16 TIM1_OCFast)
行为描述	配置 TIM1 捕获比较 4 快速性能
输入参数	TIM1_OCFast：输出比较快速特性状态 更多关于该参数的可用值的细节请参阅章节: TIM1_OCFast。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

/*使用快速模式的 TIM1 OC4 */

TIM1_OC4astConfig(TIM1_OCFast_Enable);

20.2.53 TIM1_ClearOC1Ref 函数

表 661 描述了 TIM1_ClearOC1Ref 函数。

表 661. TIM1_ClearOC1Ref 函数

函数名	TIM1_ClearOC1Ref
函数原型	void TIM1_ClearOC1Ref(u16 TIM1_OCClear)
行为描述	清除或维护在外部事件中的 OCREF1 信号
输入参数	TIM1_OCClear：输出比较清除使能位的新状态

	更多关于该参数的可用值的细节请参阅章节: TIM1_OCClear。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

TIM1_OCClear

被使用的输出比较引用清除位的值在表 662 中被列出：

表 662 TIM1_OCClear

TIM1_OCPreload	描述
TIM1_OCClear_Enable	TIM1 输出比较清除使能
TIM1_OCClear_Disable	TIM1 输出比较清除禁用

例：

```
/* 使能 TIM1 通道 1 输出比较清除使能位 */
TIM1_ClearOC1Ref(TIM1_OCClear_Enable);
```

20.2.54 TIM1_ClearOC2Ref 函数

表 663 描述了 TIM1_ClearOC2Ref 函数。

表 663. TIM1_ClearOC2Ref 函数

函数名	TIM1_ClearOC2Ref
函数原型	void TIM1_ClearOC2Ref(u16 TIM1_OCClear)
行为描述	清除或维护在外部事件中的 OCREF2 信号
输入参数	TIM1_OCClear：输出比较清除使能位的新状态 更多关于该参数的可用值的细节请参阅章节: TIM1_OCClear。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 使能 TIM1 通道 2 输出比较清除使能位 */
TIM1_ClearOC2Ref(TIM1_OCClear_Enable);
```

20.2.55 TIM1_ClearOC3Ref 函数

表 664 描述了 TIM1_ClearOC3Ref 函数。

表 664. TIM1_ClearOC3Ref 函数

函数名	TIM1_ClearOC3Ref
函数原型	void TIM1_ClearOC3Ref(u16 TIM1_OCClear)
行为描述	清除或维护在外部事件中的 OCREF3 信号
输入参数	TIM1_OCClear：输出比较清除使能位的新状态 更多关于该参数的可用值的细节请参阅章节: TIM1_OCClear。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/*使能 TIM1 通道 3 输出比较清除使能位 */

TIM1_ClearOC3Ref(TIM1_OCClear_Enable);
```

20.2.56 TIM1_ClearOC4Ref 函数

表 665 TIM1_ClearOC4Ref 函数。

表 665. TIM1_ClearOC4Ref 函数

函数名	TIM1_ClearOC4Ref
函数原型	void TIM1_ClearOC4Ref(u16 TIM1_OCClear)
行为描述	清除或维护在外部事件中的 OCREF4 信号

输入参数	TIM1_OCClear：输出比较清除使能位的新状态 更多关于该参数的可用值的细节请参阅章节: TIM1_OCClear。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/*使能 TIM1 通道 4 输出比较清除使能位 */
TIM1_ClearOC4Ref(TIM1_OCClear_Enable);
```

20.2.57 TIM1_GenerateEvent 函数

表 666 描述了 TIM1_GenerateEvent 函数。

表 666. TIM1_GenerateEvent 函数

函数名	TIM1_GenerateEvent
函数原型	void TIM1_GenerateEvent(u16 TIM1_EventSource)
行为描述	配置将由软件引发的 TIM1 事件
输入参数	TIM1_EventSource：指明 TIM1 软件事件源 更多关于该参数的可用值的细节请参阅章节: TIM1_EventSource:。

输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

TIM1_EventSource

TIM1 事件软件资源能够被选择通过使用下列值中的一个或是一组：

表 667. TIM1_EventSource 值

TIM1_EventSource	描述
TIM1_EventSource_Update	TIM1 更新事件源
TIM1_EventSource_CC1	TIM1 捕获/比较 1 事件源
TIM1_EventSource_CC2	TIM1 捕获/比较 2 事件源
TIM1_EventSource_CC3	TIM1 捕获/比较 3 事件源
TIM1_EventSource_CC4	TIM1 捕获/比较 4 事件源
TIM1_EventSource_COM	TIM1COM 事件源
TIM1_EventSource_Trigger	TIM1 触发器事件源
TIM1_EventSource_Break	TIM1 间断事件源

例：

/*选择用于 TIM1 的触发器软件事件*/

```
TIM1_GenerateEvent(TIM1_EventSource_Trigger);
```

20.2.58 TIM1_OC1Polarity 函数

表 668 描述了 TIM1_OC1Polarity 函数。

表 668. TIM1_OC1Polarity 函数

函数名	TIM1_OC1Polarity
函数原型	void TIM1_OC1PolarityConfig(u16 TIM1_OCPolarity)
行为描述	配置 TIM1 通道 1 极性
输入参数	TIM1_OCPolarity：输出比较极性 更多关于该参数的可用值的细节请参阅章节: TIM1_OCPolarity。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

TIM1_OCPolarity

TIM1_OCPolarity 选择 TIM1 极性（见表 669）:

表 669. TIM1_OCPolarity 值

TIM1_EventSource	描述
TIM1_OCPolarity_High	TIM1 输出极性高
TIM1_OCPolarity_L	TIM1 输出极性低

ow	

例：

/* 为 TIM1 通道 1 输出比较选择高极性 */

TIM1_OC1PolarityConfig(TIM1_OCPolarity_High);

20.2.59 TIM1_OC1NpolarityConfig 函数

表 670 描述了 TIM1_OC1NpolarityConfig 函数。

表 670. TIM1_OC1NPolarity 函数

函数名	TIM1_OC1NPolarity
函数原型	void TIM1_OC1NPolarityConfig(u16 TIM1_OCPolarity)
行为描述	配置 TIM1 通道 1N 极性
输入参数	TIM1_OCPolarity：输出比较 N 极性 更多关于该参数的可用值的细节请参阅章节: TIM1_OCPolarity。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

/*为 TIM1 通道 1N 输出比较选择高极性*/

TIM1_OC1NPolarityConfig(TIM1_OCPolarity_High);

20.2.60 TIM1_OC2PolarityConfig 函数

表 671 描述了 TIM1_OC2PolarityConfig 函数。

表 671. TIM1_OC2Polarity 函数

函数名	TIM1_OC2Polarity
函数原型	void TIM1_OC2PolarityConfig(u16 TIM1_OCPolarity)
行为描述	配置 TIM1 通道 2 极性
输入参数	TIM1_OCPolarity：输出比较极性 更多关于该参数的可用值的细节请参阅章节: TIM1_OCPolarity。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

/*为 TIM1 通道 2 输出比较选择高极性*/

TIM1_OC2PolarityConfig(TIM1_OCPolarity_High);

20.2.61 TIM1_OC2NpolarityConfig 函数

表 672 描述了 TIM1_OC2NpolarityConfig 函数。

表 672. TIM1_OC1NPolarity 函数

函数名	TIM1_OC2NPolarity
函数原型	void TIM1_OC2NPolarityConfig(u16 TIM1_OCPolarity)
行为描述	配置 TIM1 通道 2N 极性
输入参数	TIM1_OCPolarity：输出比较 N 极性 更多关于该参数的可用值的细节请参阅章节: TIM1_OCPolarity。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

/*为 TIM1 通道 2N 输出比较选择高极性*/

TIM1_OC2NPolarityConfig(TIM1_OCPolarity_High);

20.2.62 TIM1_OC3polarityConfig 函数

表 673 描述了 TIM1_OC3polarityConfig 函数。

表 670. TIM1_OC3Polarity 函数

函数名	TIM1_OC3Polarity
函数原型	void TIM1_OC3PolarityConfig(u16 TIM1_OCPolarity)
行为描述	配置 TIM1 通道 3 极性
输入参数	TIM1_OCPolarity：输出比较极性 更多关于该参数的可用值的细节请参阅章节: TIM1_OCPolarity。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

/*为 TIM1 通道 3 输出比较选择高极性*/

```
TIM1_OC3PolarityConfig(TIM1_OCPolarity_High);
```

20.2.63 TIM1_OC3NpolarityConfig 函数

表 674 描述了 TIM1_OC3NpolarityConfig 函数。

表 670. TIM1_OC3NPolarity 函数

函数名	TIM1_OC3NPolarity
函数原型	void TIM1_OC3NPolarityConfig(u16 TIM1_OCPolarity)
行为描述	配置 TIM1 通道 1N 极性
输入参数	TIM1_OCPolarity：输出比较 N 极性 更多关于该参数的可用值的细节请参阅章节: TIM1_OCPolarity。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

/*为 TIM1 通道 3N 输出比较选择高极性*/

TIM1_OC3NPolarityConfig(TIM1_OCPolarity_High);

20.2.64 TIM1_OC4PolarityConfig 函数

表 675 描述了 TIM1_OC4PolarityConfig 函数。

表 675. TIM1_OC4Polarity 函数

函数名	TIM1_OC4Polarity

函数原型	void TIM1_OC4PolarityConfig(u16 TIM1_OCPolarity)
行为描述	配置 TIM1 通道 4 极性
输入参数	TIM1_OCPolarity：输出比较 N 极性 更多关于该参数的可用值的细节请参阅章节: TIM1_OCPolarity。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/*为 TIM1 通道 4 输出比较选择高极性*/
TIM1_OC4PolarityConfig(TIM1_OCPolarity_High);
```

20.2.65 TIM1_CCxCmd 函数

表 676 描述了 TIM1_CCxCmd 函数。

表 676. TIM1_CCxCmd 函数

函数名	TIM1_CCxCmd
函数原型	void TIM1_CCxCmd(u16 TIM1_Channel, FunctionalState

	Newstate)
行为描述	使能或禁用 TIM1 捕获比较通道 x
输入参数 1	TIM1_Channel：TIM1 通道 更多关于该参数的可用值的细节请参阅章节: TIM1_Channel。
输入参数 2	Newstate：指定 TIM1 通道 CCxE 位新状态 这个参数能够取：ENABLE 或者 DISABLE
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 使能 TIM1 通道 4 */

TIM1_CCxCmd(TIM1_Channel_4, ENABLE);
```

20.2.66 TIM1_CCxNCmd 函数

表 677 描述了 TIM1_CCxNCmd 函数。

表 677. TIM1_CCxNCmd 函数

函数名	TIM1_CCxNCmd
函数原型	void TIM1_CCxNCmd(

	u16 TIM1_Channel, FunctionalState Newstate)
行为描述	使能或禁用 TIM1 捕获比较通道 xN
输入参数 1	TIM1_Channel：TIM1 通道 更多关于该参数的可用值的细节请参阅章节: TIM1_Channel。
输入参数 2	Newstate：指明 TIM1 通道 CCxNE 位新状态 这个参数能够取：ENABLE 或者 DISABLE
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/*使能 TIM1 通道 3N  */
TIM1_CCxCmd(TIM1_Channel_3, ENABLE);
```

20.2.67 TIM1_SelectOCxM 函数

表 678 描述了 TIM1_SelectOCxM 函数。

表 678. TIM1_SelectOCxM 函数

函数名	TIM1_SelectOCxM

函数原型	void TIM1_SelectOCxM(u16 TIM1_Channel, u16 TIM1_OCMode)
行为描述	选择 TIM1 输出比较模式。
输入参数 1	TIM1_Channel：TIM1 通道 更多关于该参数的可用值的细节请参阅章节: TIM1_Channel。
输入参数 2	TIM1_OCMode：TIM1 输出比较模式 更多关于该参数的可用值的细节请参阅章节: TIM1_OCMode。
输出参数	无
返回参数（值）	无
前提条件	在改变输出比较模式之前，该函数仅用选择的通道 ,用户需要使用 TIM1_CCxCmd 和 TIM1_CCxNCmd 函数使能通道。
调用函数	无

TIM1_OCMode

TIM1_OCMode 选择 TIM1 输出比较模式（见表 679）:

表 679. TIM1_OCMode 定义

TIM1_OCMode	描述
TIM1_OCMode_Timing	TIM1 输出比较定时模式
TIM1_OCMode_Active	TIM1 输出比较有效模式

TIM1_OCMode_Inactive	TIM1 输出比较不有效模式
TIM1_OCMode_Toggle	TIM1 输出比较触发模式
TIM1_OCMode_PWM1	TIM1 脉冲宽度调制模式 1
TIM1_OCMode_PWM2	TIM1 脉冲宽度调制模式 2
TIM1_ForcedAction_Active	强制在 OCxREF 上有效级别
TIM1_ForcedAction_InActive	强制在 OCxREF 上不有效级别

例：

```
/*为 TIM1 通道 1 选择 PWM2 模式 */
TIM1_SelectOCxM(TIM1_Channel_1, TIM1_OCMode_PWM2);
```

20.2.68 TIM1_SetCounter 函数

表 680 描述了 TIM1_SetCounter 函数。

表 680 .TIM1_SetCounter 函数

函数名	TIM1_SetCounter
函数原型	void TIM1_SetCounter(u16 Counter)
行为描述	设置 TIM1 计数器寄存器的值
输入参数	Counter：指定计数器寄存器的新值
输出参数	无
返回参数（值）	无

前提条件	无
调用函数	无

例：

/* 为 TIM1 计数器设置新的值 */

```
u16 TIM1Counter = 0xFFFF;

TIM1_SetCounter(TIM1Counter);
```

20.2.69 TIM1_SetAutoreload 函数

表 681 描述了 TIM1_SetAutoreload 函数。

表 681 . TIM1_SetAutoreload 函数

函数名	TIM1_SetAutoreload
函数原型	void TIM1_SetAutoreload(u16 Autoreload)
行为描述	设置 TIM1 自动重载寄存器的值
输入参数	Autoreload：TIM1 周期新值
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

/* 为 TIM1 自动重载寄存器设定新的值 */

```
u16 TIM1Autoreload = 0xFFFF;

TIM1_SetAutoreload(TIM1Autoreload);
```

20.2.70 TIM1_SetCompare1 函数

表 682 描述了 TIM1_SetCompare1 函数。

表 682 . TIM1_SetCompare1 函数

函数名	TIM1_SetCompare1
函数原型	void TIM1_SetCompare1(u16 Compare1)
行为描述	设置 TIM1 捕获比较 1 寄存器的值
输入参数	Compare1:TIM1 捕获比较 1 寄存器新值
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 为 TIM1 捕获比较 1 寄存器的设置新值 */

u16 TIM1Compare1 = 0x7FFF;

TIM1_SetCompare1(TIM1Compare1);
```

20.2.71 TIM1_SetCompare2 函数

表 683 描述了 TIM1_SetCompare2 函数。

表 683 . TIM1_SetCompare2 函数

函数名	TIM1_SetCompare2
函数原型	void TIM1_SetCompare2(u16 Compare2)
行为描述	设置 TIM1 捕获比较 2 寄存器的值
输入参数	Compare2:TIM1 捕获比较 2 寄存器新值
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/*为 TIM1 捕获比较 2 寄存器的设置新值 */
u16 TIM1Compare2 = 0x7FFF;
TIM1_SetCompare2(TIM1Compare2);
```

20.2.72 TIM1_SetCompare3 函数

表 684 描述了 TIM1_SetCompare3 函数。

表 6824. TIM1_SetCompare3 函数

函数名	TIM1_SetCompare3
函数原型	void TIM1_SetCompare3(u16 Compare3)
行为描述	设置 TIM1 捕获比较 3 寄存器的值

输入参数	Compare3:TIM1 捕获比较 3.寄存器新值
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

/*为 TIM1 捕获比较 3 寄存器的设置新值*/

```
u16 TIM1Compare3 = 0x7FFF;

TIM1_SetCompare3(TIM1Compare3);
```

20.2.73 TIM1_SetCompare4 函数

表 685 描述了 TIM1_SetCompare4 函数。

表 682 . TIM1_SetCompare4 函数

函数名	TIM1_SetCompare4
函数原型	void TIM1_SetCompare4(u16 Compare4)
行为描述	设置 TIM1 捕获比较 4 寄存器的值
输入参数	Compare4:TIM1 捕获比较 4 寄存器新值
输出参数	无
返回参数（值）	无
前提条件	无

调用函数	无

例：

/*为 TIM1 捕获比较 4 寄存器的设置新值*/

```
u16 TIM1Compare4 = 0x7FFF;

TIM1_SetCompare4(TIM1Compare4);
```

20.2.74 TIM1_SetIC1Prescaler 函数

表 686 描述了 TIM1_SetIC1Prescaler 函数。

表 686 . TIM1_SetIC1Prescaler 函数

函数名	TIM1_SetIC1Prescaler
函数原型	void TIM1_SetIC1Prescaler(u16 TIM1_IC1Prescaler)
行为描述	设置 TIM1 输入捕获 1 预分频器
输入参数	TIM1_IC1Prescaler：输入捕获 1 预分频器 更多关于该参数的可用值的细节请参阅章节: TIM1_IC1Prescaler。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

TIM1_ICPrescaler

TIM1_ICPrescaler 选择 TIM1 输入捕获预分频器（见表 687）。

表 687. TIM1_ICPrescaler 值

TIM1_ICPrescaler	描述
TIM1_ICPSC_DIV1	每次捕获输入上检测到信号沿的时候捕获一次
TIM1_ICPSC_DIV2	每两次事件捕获完成 2 次
TIM1_ICPSC_DIV4	每两次事件捕获完成 4 次
TIM1_ICPSC_DIV8	每两次事件捕获完成 8 次

例：

/*设置 TIM1 输入捕获 1 预分频器*/

TIM1_SetIC1Prescaler(TIM1_ICPSC_Div2);

20.2.75 TIM1_SetIC2Prescaler 函数

表 688 描述了 TIM1_SetIC2Prescaler 函数。

表 688. TIM1_SetIC2Prescaler 函数

函数名	TIM1_SetIC2Prescaler
函数原型	void TIM1_SetIC2Prescaler(u16 TIM1_IC1Prescaler)
行为描述	设置 TIM1 输入捕获 2 预分频器

输入参数	TIM1_IC1Prescaler：输入捕获 2 预分频器 更多关于该参数的可用值的细节请参阅章节: TIM1_IC1Prescaler。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

/*设置 TIM1 输入捕获 2 预分频器*/

TIM1_SetIC2Prescaler(TIM1_ICPSC_Div2);

20.2.76 TIM1_SetIC3Prescaler 函数

表 689 描述了 TIM1_SetIC3Prescaler 函数。

表 688. TIM1_SetIC3Prescaler 函数

函数名	TIM1_SetIC3Prescaler
函数原型	void TIM1_SetIC3Prescaler(u16 TIM1_IC1Prescaler)
行为描述	设置 TIM1 输入捕获 3 预分频器
输入参数	TIM1_IC1Prescaler：输入捕获 3 预分频器 更多关于该参数的可用值的细节请参阅章节:

	TIM1_IC1Prescaler。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 设置 TIM1 输入捕获 3 预分频器 */
TIM1_SetIC3Prescaler(TIM1_ICPSC_Div2);
```

20.2.77 TIM1_SetIC4Prescaler 函数

表 690 描述了 TIM1_SetIC4Prescaler 函数。

表 690. TIM1_SetIC4Prescaler 函数

函数名	TIM1_SetIC4Prescaler
函数原型	void TIM1_SetIC4Prescaler(u16 TIM1_IC1Prescaler)
行为描述	设置 TIM1 输入捕获 4 预分频器
输入参数	TIM1_IC1Prescaler：输入捕获 4 预分频器 更多关于该参数的可用值的细节请参阅章节: TIM1_IC1Prescaler。
输出参数	无

返回参数（值）	无
前提条件	无
调用函数	无

例：

/*设置 TIM1 输入捕获 4 预分频器*/

TIM1_SetIC4Prescaler(TIM1_ICPSC_Div2);

20.2.78 TIM1_SetClockDivision 函数

表 691 描述了 TIM1_SetClockDivision 函数。

表 691. TIM1_SetClockDivision 函数

函数名	TIM1_ SetClockDivision
函数原型	void TIM1_SetClockDivision(u16 TIM1_CKD)
行为描述	设置 TIM1 时钟分隔值
输入参数	TIM1_CKD：时钟分隔值 更多关于该参数的可用值的细节请参阅章节：TTIM1_CKD。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

TIM1_CKD

TIM1_CKD 选择 TIM1 时钟分配（见表 692）。

表 692. TIM1_CKD 值

TIM1_CKD	描述
TIM1_CKD_DIV1	T（DTS） = T（ck_tim）
TIM1_CKD_DIV2	T（DTS） = 2*T（ck_tim）
TIM1_CKD_DIV4	T（DTS） = 4*T（ck_tim）

例：

/*设置 TIM1 时钟分隔值*/

```
TIM1_SetClockDivision(TIM1_CKD_DIV4);
```

20.2.79 TIM1_GetCapture1 函数

表 693 描述了 TIM1_GetCapture1 函数。

表 693. TIM1_GetCapture1 函数

函数名	TIM1_GetCapture1
函数原型	u16 TIM1_GetCapture1(void)
行为描述	获取 TIM1 输入捕获 1 的值
输入参数	无
输出参数	无
返回参数（值）	无

前提条件	无
调用函数	无

例：

```
/* 获取 TIM1 输入捕获 1 的值 */
u16 IC1value = TIM1_GetCapture1();
```

20.2.80 TIM1_GetCapure2 函数

表 694 描述了 TIM1_GetCapure2 函数。

表 693. TIM1_GetCapture2 函数

函数名	TIM1_GetCapture2
函数原型	u16 TIM1_GetCapture2(void)
行为描述	获取 TIM1 输入捕获 2 的值
输入参数	无
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 获取 TIM1 输入捕获 2 的值 */
u16 IC2value = TIM1_GetCapture2();
```

20.2.81 TIM1_GetCapure3 函数

表 695 描述了 TIM1_GetCapure3 函数。

表 695. TIM1_GetCapture3 函数

函数名	TIM1_GetCapture3
函数原型	u16 TIM1_GetCapture3(void)
行为描述	获取 TIM1 输入捕获 3 的值
输入参数	无
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 获取 TIM1 输入捕获 3 的值 */
u16 IC3value = TIM1_GetCapture3();
```

20.2.82 TIM1_GetCapure4 函数

表 696 描述了 TIM1_GetCapure4 函数。

表 696. TIM1_GetCapture4 函数

函数名	TIM1_GetCapture4
函数原型	u16 TIM1_GetCapture4(void)
行为描述	获取 TIM1 输入捕获 4 的值
输入参数	无
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 获取 TIM1 输入捕获 4 的值 */
u16 IC4value = TIM1_GetCapture4();
```

20.2.83 TIM1_GetConuter 函数

表 697 描述了 TIM1_GetCounter 函数。

表 697. TIM1_GetConuter 函数

函数名	TIM1_GetConuter
函数原型	void TIM1_GetCounter(void)
行为描述	获取 TIM1 计数器的值
输入参数	无
输出参数	无

返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 获取 TIM1 计数器的值 */

u16 TIM1Counter = TIM1_GetCounter()
```

20.2.84 TIM1_GetPrescaler 函数

表 698 描述了 TIM1_GetPrescaler 函数。

表 698. TIM1_GetPrescaler 函数

函数名	TIM1_GetPrescaler
函数原型	void TIM1_GetPrescaler(void)
行为描述	获取 TIM1 预分频器的值
输入参数	无
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 获取 TIM1 预分频器的值 */
```

u16 TIM1Prescaler = TIM1_GetPrescaler();

20.2.85 TIM1_GetFlagStatus 函数

表 699 描述了 TIM1_GetFlagStatus 函数。

表 699. TIM1_GetFlagStatus 函数

函数名	TIM1_GetFlagStatus
函数原型	FlagStatus TIM1_GetFlagStatus(u16 TIM1_FLAG)
行为描述	检查指定的 TIM1 标记是否被置位
输入参数	TIM1_FLAG：指定要被检查的标记 更多关于该参数的可用值的细节请参阅章节: TIM1_FLAG。
输出参数	无
返回参数（值）	TIM1_FLAG 的新状态 (置位或复位)
前提条件	无
调用函数	无

TIM1_Flag

TIM1 能被检查的标记在表 700 中被列出：

表 700. TIM1_Flag 值

TIM1_CKD	描述

TIM1_FLAG_Update	TIM1 更新标记
TIM1_FLAG_CC1	TIM1 捕获/比较 1 标记
TIM1_FLAG_CC2	TIM1 捕获/比较 2 标记
TIM1_FLAG_CC3	TIM1 捕获/比较 3 标记
TIM1_FLAG_CC4	TIM1 捕获/比较 4 标记
TIM1_FLAG_COM	TIM1 COM 标记
TIM1_FLAG_Trigger	TIM1 触发标记
TIM1_FLAG_BRK	TIM1 间隔标记
TIM1_FLAG_CC1OF	TIM1 捕获/比较 1 溢出标记
TIM1_FLAG_CC2OF	TIM1 捕获/比较 2 溢出标记
TIM1_FLAG_CC3OF	TIM1 捕获/比较 3 溢出标记
TIM1_FLAG_CC4OF	TIM1 捕获/比较 4 溢出标记

例：

```
/*检查 TIM1 捕获/比较 1 标记是置位还是复位*/
if(TIM1_GetFlagStatus(TIM1_FLAG_CC1) == SET)
{
}
```

20.2.86 TIM1_ClearFlag 函数

表 701 描述了 TIM1_ClearFlag 函数。

表 701. TIM1_ClearFlag 函数

函数名	TIM1_ClearFlag
函数原型	void TIM1_ClearFlag(u16 TIM1_Flag)
行为描述	清除 TIM1 未决的标记
输入参数	TIM1_FLAG：要被清除的标记 更多关于该参数的可用值的细节请参阅章节: TIM1_FLAG。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 清除捕获/比较 1 标记 */
TIM1_ClearFlag(TIM1_FLAG_CC1);
```

20.2.87 TIM1_GetITStatus 函数

表 702 描述了 TIM1_GetITStatus 函数。

表 702. TIM1_GetITStatus 函数

函数名	TIM1_GetITStatus
函数原型	ITStatus TIM1_GetITStatus(u16 TIM1_IT)

行为描述	检查指定的 TIM1 中断是否发生
输入参数	TIM1_IT：要被检查的 TIM1 中断源 更多关于该参数的可用值的细节请参阅章节: TIM1_IT。
输出参数	无
返回参数（值）	TIM1_IT 的新状态 (置位或复位)
前提条件	无
调用函数	无

例：

```
/* 检查捕获/比较 1 中断是否发生 */
if(TIM1_GetITStatus(TIM1_IT_CC1) == SET)
{
}
```

20.2.88 TIM1_ClearITPendingBit 函数

表 703 描述了 TIM1_ClearITPendingBit 函数。

表 703. TIM1_ClearITPendingBit 函数

函数名	TIM1_ClearITPendingBit
函数原型	void TIM1_ClearITPendingBit(u16 TIM1_IT)
行为描述	清楚 TIM1 中断未决位

输入参数	TIM1_IT：要被清除的中断未决位 更多关于该参数的可用值的细节请参阅章节: TIM1_IT。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 清除捕获/比较 1 中断未决位 */
TIM1_ClearITPendingBit(TIM1_IT_CC1);
```

21 通用同步/异步收发器（USART）

通用同步/异步收发器（USART）同外部设备按照工业标准的 NRZ 异步串行数据格式进行可变的全双工数据交换。SCI 提了一个非常宽的基于小数波特率产生器系统的波特率范围。USART 接口也支持兼容 IrDA SIR ENDCE 规范的智能卡协议。它能进行单线半双工通信，同步传输和调制操作（CTS/RTS）。

章节 21.1：USART 寄存器结构体描述了用于 USART 库中的数据结构。章节 21.2：固件库函数列出了相关的库函数。

21.1 USART 寄存器结构体

USART 寄存器结构体，USART_TypeDef,被定义在 stm32f10x.h 文件中的，如下：

```
typedef struct
{
  vu16 SR;
  u16 RESERVED1;
  vu16 DR;
  u16 RESERVED2;
  vu16 BRR;
  u16 RESERVED3;
  vu16 CR1;
  u16 RESERVED4;
  vu16 CR2;
  u16 RESERVED5;
  vu16 CR3;
  u16 RESERVED6;
  vu16 GTPR;
  u16 RESERVED7;
} USART_TypeDef;
```

表 704 列出了所有的 USART 寄存器。

表 704. USART 寄存器

寄存器	描述
SR	USART 状态寄存器
DR	USART 数据寄存器
BRR	USART 波特率寄存器
CR1	USART 控制寄存器 1
CR2	USART 控制寄存器 2
CR3	USART 控制寄存器 3
GTPR	USART 预警时间和预分频寄存器

这三个 USART 外围设备在 stm32f10x_map.h 中被声明。

…

```
#define PERIPH_BASE              ((u32)0x40000000)
#define APB1PERIPH_BASE          PERIPH_BASE
#define APB2PERIPH_BASE          (PERIPH_BASE + 0x10000)
#define AHBPERIPH_BASE           (PERIPH_BASE + 0x20000)
#define USART1_BASE              (APB2PERIPH_BASE + 0x3800)
#define USART2_BASE              (APB1PERIPH_BASE + 0x4400)
#define USART3_BASE              (APB1PERIPH_BASE + 0x4800)
#ifndef DEBUG
…
#ifdef _USART1
```

```
  #define USART1                ((USART_TypeDef *) USART1_BASE)
#endif /*_USART1 */
#ifdef _USART2
  #define USART2                ((USART_TypeDef *) USART2_BASE)
#endif /*_USART2 */
#ifdef _USART3
  #define USART3                ((USART_TypeDef *) USART3_BASE)
#endif /*_USART3 */
...
#else    /* DEBUG */
...
#ifdef _USART1
  EXT USART_TypeDef             *USART1;
#endif /*_USART1 */
#ifdef _USART2
  EXT USART_TypeDef             *USART2;
#endif /*_USART2 */
#ifdef _USART3
  EXT USART_TypeDef             *USART3;
#endif /*_USART3 */
...
```

```
#endif
```

当使用 Debug 模式时，_USART1,_USART2 和_USART3 指针在 stm32f10x_lib.c 中被初始化：

```
        ...
#ifdef _USART1
  USART1 = (USART_TypeDef *) USART1_BASE;
#endif /*_USART1 */
#ifdef _USART2
  USART2 = (USART_TypeDef *) USART2_BASE;
#endif /*_USART2 */
#ifdef _USART3
  USART3 = (USART_TypeDef *) USART3_BASE;
#endif /*_USART3 */
...
```

要访问 USART 寄存器_USART, _USART1, _USART2, _USART3 必须在 stm32f10x_conf.h 中定义，如下：

```
...
#define _USART
#define _USART1
#define _USART2
#define _USART3
```

21.2 固件库函数

表 705 列出了 USART 库中的各种函数。

表 705. USART 固件库函数

函数名	描述
USART_DeInit	重置 USARTx 外设寄存器为默认重置值
USART_Init	根据 USART_InitStruct 中指定的参数初始化 USARTx 外围设备
USART_StructInit	用默认值填充每一个 USART_InitStruct 成员
USART_Cmd	使能或禁用特定的 USART 外围设备
USART_ITConfig	使能或禁用特定的 USART 中断
USART_DMACmd	使能或禁用特定的 USART DMA 接口
USART_SetAddress	设置 USART 节点的地址
USART_WakeUpConfig	选择 USART 的唤醒方法
USART_ReceiverWakeUpCmd	决定 USART 是否在无声模式
USART_LINBreakDetectionConfi	设置 USART LIN 间隔检波长度
USART_LINCmd	使能或禁用 USART LIN 模式
USART_SendData	通过 USARTx 外围设备传输单个数据
USART_ReceiveData	返回最近由 USARTx 外围设备接收的数据

USART_SendBreak	传输间断字符
USART_SetGuardTime	设置指定的 USART 预警时间
USART_SetPrescaler	设置 USART 时钟预分频器
USART_SmartCardCmd	使能或禁用 USART 智能卡模式
USART_SmartCardNackCmd	使能或禁用 NACK 传输
USART_HalfDuplexCmd	使能或禁用 USART 半双工模式
USART_IrDAConfig	配置 USART IrDA 模式
USART_IrDACmd	使能或禁用 USART IrDA 模式
USART_GetFlagStatus	检测指明的 USART 标记是否被置位
USART_ClearFlag	清除 USARTx 未决标记
USART_GetITStatus	检测指明的 USART 中断是否发生
USART_ClearITPendingBit	清除 USARTx 中断未决位

21.2.1 USART_DeInit 函数

表 706 描述了 USART_DeInit 函数。

表 706. USART_DeInit 函数

函数名	USART_DeInit
函数原型	void USART_DeInit(USART_TypeDef* USARTx)
行为描述	重置 USARTx 外围寄存器为默认重置值

输入参数	USARTx：x 可以为 1，2 或 3 以选择 USART 的外围设备
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	RCC_APB2PeriphResetCmd() RCC_APB1PeriphResetCmd()

例：

```
/* 重置 USARTx 外围寄存器为默认重置值 */

USART_DeInit(USART1);
```

21.2.2 USART_Init 函数

表 707 描述了 USART_Init 函数。

表 707. USART_Init 函数

函数名	USART_Init
函数原型	void USART_Init(USART_TypeDef* USARTx, USART_InitTypeDef* USART_InitStruct)
行为描述	根据 USART_InitStruct 中的指定的参数初始化 USARTx 外围设备

输入参数	USART_InitStruct：指向包含特定的 USART 外围设备配置信息的 USART_InitTypeDef 结构体 更多关于该参数的可用值的细节请参阅章节：USART_InitTypeDef 结构体。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

USART_InitTypeDef 结构体

USART_InitTypeDef 结构体被定义在 stm32f10x_usart.h 文件中：

typedef struct

```
{
  u32 USART_BaudRate;
  u16 USART_WordLength;
  u16 USART_StopBits;
  u16 USART_Parity;
  u16 USART_HardwareFlowControl;
  u16 USART_Mode;
  u16 USART_Clock;
  u16 USART_CPOL;
  u16 USART_CPHA;
```

```
  u16 USART_LastBit;

} USART_InitTypeDef;
```

表 708 描述了 USART_InitTypeDef 结构体用在同步和异步模式的成员。

表 708. USART_InitTypeDef 成员与之相对的 USART 模式

成员	异步模式	同步模式
USART_BaudRate	X	X
USART_WordLength	X	X
USART_StopBits	X	X
USART_Parity	X	X
USART_HardwareFlowControl	X	X
USART_Mode	X	X
USART_Clock		X
USART_CPOL		X
USART_CPHA		X
USART_LastBit		X

USART_BaudRate

这个成员配置 USART 通信波特率。波特率用下面的公式计算：

IntegerDivider = ((APBClock) / (16 * (USART_InitStruct->USART_BaudRate)))

FractionalDivider = ((IntegerDivider - ((u32) IntegerDivider)) * 16) + 0.5

USART_WordLength

USART_WordLength 指出在发送或是接收中一个桢中数据位的个数，该成员的值见表 709：

表 709. USART_WordLength 定义

USART_WordLength	描述
USART_WordLength_8b	8 位数据
USART_WordLength_9b	9 位数据

USART_StopBits

USART_StopBits 定义传输停止位的个数。该成员值见表 710：

表 710. USART_WordLength 定义

USART_StopBits	描述
USART_StopBits_1	1 停止位在帧结尾被传输
USART_StopBits_0_5	0.5 停止位在帧结尾被传输
USART_StopBits_2	2 停止位在帧结尾被传输
USART_StopBits_1_5	1.5 停止位在帧结尾被传输

USART_Parity

USART_Parity 定义了奇偶模式。该成员值见表 711：

表 711. USART_Parity 定义

USART_Parity	描述

USART_Parity_No	不校验
USART_Parity_Even	偶校验
USART_Parity_Odd	奇校验

USART_HardwareFlowControl

USART_HardwareFlowControl 指明硬件流控制模式是否被使能。该成员值见表 712：

表 712. USART_HardwareFlowControl 定义

USART_HardwareFlowControl	描述
USART_HardwareFlowControl_None	HFC 禁用
USART_HardwareFlowControl_RTS	RTS 使能
USART_HardwareFlowControl_CTS	CTS 使能
USART_HardwareFlowControl_RTS_CTS	RTS 和 CTS 使能

USART_Mode

USART_Mode 指明接收或是发送模式是否被使能，该成员的值见表 713：

表 713. USART_Mode 定义

USART_Mode	描述
USART_Mode_Tx	发送使能
USART_Mode_Rx	接收使能

USART_Clock

USART_Clock 指明 USART_Clock 成员的 USART 时钟是否被激活。该成员值见表 714：

表 714. USART_Clock 定义

USART_Clock	描述
USART_Clock_Enable	USART 时钟使能
USART_Clock_Disable	USART 时钟禁用

USART_CPOL

USART_CPOL 指明串行时钟的稳定状态值。该成员的值见表 715：

表 715. USART_CPOL 定义

USART_CPOL	描述
USART_CPOL_High	时钟为高有效
USART_CPOL_Low	时钟为低有效

USART_CPHA

USART_CPHA 定义时钟传输哪一位为捕获位。该成员的值见表 716：

表 716. USART_CPHA 定义

USART_CPHA	描述
USART_CPHA_1Edge	数据在第一个时钟边缘被捕获
USART_CPHA_2Edge	数据在第二个时钟边缘被捕获

USART_LastBit

USART_LastBit 定义了在异步模式中与最后被传输的数据位（MSB）相关的钟脉冲是否必须在 SCLK 引脚输出。该成员的值见表 717：

表 717. USART_LastBit 定义

USART_LastBit	描述

USART_LastBit_Disable	最后数据位的时钟脉冲不输出到 SCLK 引脚
USART_LastBit_Enable	最后数据位的时钟脉冲输出到 SCLK 引脚

例:

```
/* 下面的例子讲解了如何配置 USART1 */
USART_InitTypeDef USART_InitStructure;
USART_InitStructure.USART_BaudRate = 9600;
USART_InitStructure.USART_WordLength = USART_WordLength_8b;
USART_InitStructure.USART_StopBits = USART_StopBits_1;
USART_InitStructure.USART_Parity = USART_Parity_Odd;
USART_InitStructure.USART_HardwareFlowControl =
USART_HardwareFlowControl_RTS_CTS;
USART_InitStructure.USART_Mode = USART_Mode_Tx |USART_Mode_Rx;
USART_InitStructure.USART_Clock = USART_Clock_Disable;
USART_InitStructure.USART_CPOL = USART_CPOL_High;
USART_InitStructure.USART_CPHA = USART_CPHA_1Edge;
USART_InitStructure.USART_LastBit = USART_LastBit_Enable;
USART_Init(USART1, &USART_InitStructure);
```

21.2.3 USART_StructInit 函数

表 718 描述了 USART_StructInit 函数。

表 718. USART_StructInit 函数

函数名	USART_StructInit
函数原型	void USART_StructInit(USART_InitTypeDef* USART_InitStruct)
行为描述	用默认值填充每一个 USART_InitStruct 成员
输入参数	USART_InitStruct ：指向会被初始化的 USART_InitTypeDef 结构体
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

USART_StructInit 成员有下列默认值：

表 719. USART_StructInit 默认值

成员	默认值
USART_BaudRate	9600
USART_WordLength	USART_WordLength_8b
USART_StopBits	USART_StopBits_1
USART_Parity	USART_Parity_No
USART_HardwareFlow	USART_HardwareFlowControl_None
USART_Mode	USART_Mode_Rx | USART_Mode_Tx

USART_Clock	USART_Clock_Disable
USART_CPOL	USART_CPOL_Low
USART_CPHA	USART_CPHA_1Edge
USART_LastBit	USART_LastBit_Disable

例：

```
/* 下面的例子讲解了如何初始化 USART_InitTypeDef 结构体 */
USART_InitTypeDef USART_InitStructure;
USART_StructInit(&USART_InitStructure);
```

21.2.4 USART_Cmd 函数

表 720 描述了 USART_Cmd 函数。

表 720. USART_Cmd 函数

函数名	USART_Cmd
函数原型	void USART_Cmd(USART_TypeDef* USARTx, FunctionalState NewState)
行为描述	使能或禁用特定的 USART 外围设备
输入参数 1	USARTx：x 可以为 1，2 或 3 以选择 USART 的外围设备
输入参数 2	Newstate：USARTx 外围设备的新状态 这个参数能够取：ENABLE 或者 DISABLE

输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 使能 USART1 */

USART_Cmd(USART1, ENABLE);
```

21.2.5 USART_ITConfig 函数

表 721 描述了 USART_ITConfig 函数。

表 721. USART_ITConfig 函数

函数名	USART_ITConfig
函数原型	void USART_ITConfig(USART_TypeDef* USARTx, u16 USART_IT, FunctionalState NewState)
行为描述	使能或禁用特定的 USART 中断
输入参数 1	USARTx：x 可以为 1，2 或 3 以选择 USART 的外围设备
输入参数 2	USART_IT：指明要使能或禁用的 USART 中断源。 更多关于该参数的可用值的细节请参阅章节：

	USART_IT。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

USART_IT

USART_IT 被用于使能或禁用 USART 中断。这个参数可以引用表 722 中的值。

表 722. USART_IT 值

USART_IT	描述
USART_IT_PE	奇偶错误中断
USART_IT_TXE	传输中断
USART_IT_TC	传输接受中断
USART_IT_RXNE	接收中断
USART_IT_IDLE	IDLE 线性中断
USART_IT_LBD	LIN 间隔侦测中断
USART_IT_CTS	CTS 中断
USART_IT_ERR	错误中断

例：

```
/* 使能 USART1 发送中断 */

USART_ITConfig(USART1, USART_IT_Transmit ENABLE);
```

21.2.6 USART_DMACmd 函数

表 723 描述了 USART_DMACmd 函数。

表 723. USART_DMACmd 函数

函数名	USART_DMACmd
函数原型	void USART_DMACmd(USART_TypeDef* USARTx, u16 USART_DMAReq, FunctionalState Newstate)
行为描述	使能或禁用特定的 USART DMA 接口
输入参数 1	USARTx：x 可以为 1，2 或 3 以选择 USART 的外围设备
输入参数 2	USART_DMAReq：指定 DMA 请求 更多关于该参数的可用值的细节请参阅章节：USART_DMAReq。
输入参数 3	Newstate：DMA 请求源的新状态 这个参数能够取：ENABLE 或者 DISABLE
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

USART_DMAReq

USART_DMAReq 选择要被使能或禁用的 DMA 请求。这个参数的值如表所示。

表 724. USART_DMAReq 值

USART_DMAReq	描述
USART_DMAReq_Tx	发送 DMA 请求
USART_DMAReq_Rx	接收 DMA 请求

例：

```
/* 使能 USART2 的 Rx 和 Tx 的 DMA 传输 */
USART_DMACmd(USART2, USART_DMAReq_Rx | USART_DMAReq_Tx, ENABLE);
```

21.2.7 USART_SetAddress 函数

表 725 描述了 USART_SetAddress 函数。

表 725. USART_SetAddress 函数

函数名	USART_SetAddress
函数原型	void USART_SetAddress(USART_TypeDef* USARTx, u8 USART_Address)
行为描述	设置 USART 节点的地址
输入参数 1	USARTx：x 可以为 1，2 或 3 以选择 USART 的外围设备
输入参数 2	USART_Address：指出了 USART 节点的地址
输出参数	无

返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 置 USART2 地址为 0x5 */
USART_SetAddress(USART2, 0x5);
```

USART_WakeUpConfig 函数

表 726 描述了 USART_WakeUpConfig 函数。

表 726. USART_WakeUpConfig 函数

函数名	USART_WakeUpConfig
函数原型	void USART_WakeUpConfig(USART_TypeDef* USARTx, u16 USART_WakeUp)
行为描述	选择 USART 的唤醒方法
输入参数 1	USARTx：x 可以为 1，2 或 3 以选择 USART 的外围设备
输入参数 2	USART_WakeUp：指明了 USART 的唤醒方法 更多关于该参数的可用值的细节请参阅章节：USART_WakeUp。
输出参数	无

返回参数（值）	无
前提条件	无
调用函数	无

USART_WakeUp

USART_WakeUp 选择唤醒方式。这个参数从表 727 中引用值。

表 727. USART_WakeUp 值

USART_WakeUp	描述
USART_WakeUp_IdleLine	IDLE 线路唤醒
USART_WakeUp_AddressMark	地址标记唤醒

例：

```
/* 选择 IDLE 线路为 USART1 的唤醒方式*/

USART_WakeUpConfig(USART1, USART_WakeUpIdleLine);
```

21.2.8 USART_ReceiverWakeUpCmd 函数

表 728 描述了 USART_ReceiverWakeUpCmd 函数。

表 728. USART_ReceiverWakeUpCmd 函数

函数名	USART_ReceiverWakeUpCmd
函数原型	void USART_ReceiverWakeUpCmd(USART_TypeDef*

	USARTx, FunctionalState Newstate)
行为描述	决定 USART 是否在无声模式
输入参数 1	USARTx：x 可以为 1，2 或 3 以选择 USART 的外围设备
输入参数 2	Newstate：USART 模式的新状态 这个参数能够取：ENABLE 或者 DISABLE
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* USART3 处于正常模式*/
USART_ReceiverWakeUpCmd(USART3, DISABLE);
```

21.2.9 USART_LINBreakDetectLengthConfig 函数

表 729 描述了 USART_LINBreakDetectLengthConfig 函数。

表 729. USART_ReceiverWakeUpCmd 函数

函数名	USART_LINBreakDetectLengthConfig
函数原型	void USART_LINBreakDetectLengthConfig(USART_TypeDef*

	USARTx, u16 USART_LINBreakDetectLength)
行为描述	设置 USART LIN 间隔检波长度
输入参数 1	USARTx：x 可以为 1，2 或 3 以选择 USART 的外围设备
输入参数 2	USART_LINBreakDetectLength 指明了 LIN 间断检测长度 更多关于该参数的可用值的细节请参阅章节：USART_LINBreakDetectLength。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

USART_LINBreakDetectLength

USART_LINBreakDetectLength 选择 LIN 间断检波长度。这个参数的值引用表 730 中的值。

表 730. USART_LINBreakDetectLength 值

USART_LINBreakDetectLength	描述
USART_LINBreakDetectLength_10b	10 位间断检测
USART_LINBreakDetectLength_11b	11 位间断检测

例：

```
/* 为 USART1 选择 10 间断检测 */

USART_LINBreakDetectLengthConfig(USART1,
```

USART_LINDetectLength_10b);

21.2.10 USART_LINCmd 函数

表 731 描述了 USART_LINCmd 函数。

表 731. USART_LINCmd 函数

函数名	USART_LINCmd
函数原型	void USART_LINCmd(USART_TypeDef* USARTx, FunctionalState Newstate)
行为描述	使能或禁用 USART LIN 模式
输入参数 1	USARTx：x 可以为 1，2 或 3 以选择 USART 的外围设备
输入参数 2	Newstate：USART LIN 模式的新状态 这个参数能够取：ENABLE 或者 DISABLE
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 使能 USART2 LIN 模式 */
USART_LINCmd(USART2, ENABLE);
```

21.2.11 USART_SendData 函数

表 732 描述了 USART_SendData 函数。

表 732. USART_SendData 函数

函数名	USART_SendData
函数原型	void USART_SendData(USART_TypeDef* USARTx, u16 Data)
行为描述	传输单个数据通过 USARTx 外围设备
输入参数 1	USARTx：x 可以为 1，2 或 3 以选择 USART 的外围设备
输入参数 2	Data：要被传输的数据
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 在 USART3 上发送一个半字 */
USART_SendData(USART3, 0x26);
```

21.2.12 USART_ReceiveData 函数

表 733 描述了 USART_ReceiveData 函数。

表 733. USART_ReceiveData 函数

函数名	USART_ReceiveData
函数原型	u16 USART_ReceiveData(USART_TypeDef* USARTx)
行为描述	返回最近由 USARTx 外围设备接收的数据
输入参数 1	USARTx：x 可以为 1，2 或 3 以选择 USART 的外围设备
输出参数	无
返回参数（值）	接收的数据
前提条件	无
调用函数	无

例：

```
/*在 USART2 上接收一个半字 */

u16 RxData;

RxData = USART_ReceiveData(USART2);
```

21.2.13 USART_SendBreak 函数

表 734 描述了 USART_SendBreak 函数。

表 734. USART_SendBreak 函数

函数名	USART_SendBreak

函数原型	void USART_SendBreak(USART_TypeDef* USARTx)
行为描述	传输间断符
输入参数 1	USARTx：x 可以为 1，2 或 3 以选择 USART 的外围设备
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 在 USART1 发送间断符 */
USART_SendBreak(USART1);
```

21.2.14 USART_SetGuardTime 函数

表 735 描述了 USART_SetGuardTime 函数。

表 735. USART_SetGuardTime 函数

函数名	USART_SetGuardTime
函数原型	void USART_SetGuardTime(USART_TypeDef* USARTx, u8 USART_GuardTime)

行为描述	设置指定的 USART 预警时间
输入参数 1	USARTx：x 可以为 1，2 或 3 以选择 USART 的外围设备
输入参数 2	USART_GuardTime：指定预警时间
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 置预警时间为 0x78 */

USART_SetGuardTime(0x78);
```

21.2.15 USART_SetPrescaler 函数

表 736 描述了 USART_SetPrescaler 函数。

表 736. USART_SetPrescaler 函数

函数名	USART_SetPrescaler
函数原型	void USART_SetPrescaler(USART_TypeDef* USARTx, u8 USART_Prescaler)
行为描述	设置 USART 时钟预分频器

输入参数 1	USARTx：x 可以为 1，2 或 3 以选择 USART 的外围设备
输入参数 2	USART_Prescaler：指明预分频器
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 置系统时钟预分频器为 0x56 */
USART_SetPrescaler(0x56);
```

21.2.16 USART_SmartCardCmd 函数

表 737 描述了 USART_SmartCardCmd 函数。

表 737. USART_SmartCardCmd 函数

函数名	USART_SmartCardCmd
函数原型	void USART_SmartCardCmd(USART_TypeDef* USARTx, FunctionalState Newstate)
行为描述	使能或禁用 USART 智能卡模式
输入参数 1	USARTx：x 可以为 1，2 或 3 以选择 USART 的外

	围设备
输入参数 2	Newstate：智能卡模式的新状态 这个参数能够取：ENABLE 或者 DISABLE
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 使能 USART1 智能卡模式*/

USART_SmartCardCmd(USART1, ENABLE);
```

21.2.17 USART_SmartCardNACKCmd 函数

表 738 描述了 USART_SmartCardNACKCmd 函数。

表 738. USART_SmartCardNACKCmd 函数

函数名	USART_SmartCardNACKCmd
函数原型	void USART_SmartCardNACKCmd(USART_TypeDef* USARTx, FunctionalState Newstate)
行为描述	使能或禁用 NACK 传输

输入参数 1	USARTx：x 可以为 1，2 或 3 以选择 USART 的外围设备
输入参数 2	Newstate：NACK 传输的新状态 这个参数能够取：ENABLE 或者 DISABLE
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 奇偶错误期间使能 USART1 NACK 传输 */
USART_SmartCardNACKCmd(USART1, ENABLE);
```

21.2.18 USART_HalfDuplexCmd 函数

表 739 描述了 USART_HalfDuplexCmd 函数。

表 739. USART_HalfDuplexCmd 函数

函数名	USART_HalfDuplexCmd
函数原型	void USART_HalfDuplexCmd(USART_TypeDef* USARTx, FunctionalState Newstate)
行为描述	使能或禁用 USART 半双工模式

输入参数 1	USARTx：x 可以为 1，2 或 3 以选择 USART 的外围设备
输入参数 2	Newstate：半双工模式的新状态 这个参数能够取：ENABLE 或者 DISABLE
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/*使能 USART2 的半双工模式*/
USART_HalfDuplexCmd(USART2, ENABLE);
```

21.2.19 USART_IrDAConfig 函数

表 740 描述了 USART_IrDAConfig 函数。

表 740. USART_IrDAConfig 函数

函数名	USART_IrDAConfig
函数原型	void USART_IrDAConfig(USART_TypeDef* USARTx, u16 USART_IrDAMode)
行为描述	配置 USART IrDA 模式

输入参数 1	USARTx：x 可以为 1，2 或 3 以选择 USART 的外围设备
输入参数 2	USART_IrDAMode：指定 IrDA 模式 更多关于该参数的可用值的细节请参阅章节：USART_IrDAMode。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

USART_IrDAMode

USART_IrDAMode 选择 IrDA 模式。这个参数可以引用表 741。

表 741. USART_IrDAMode 值

USART_IrDAMode	描述
USART_IrDAMode_LowPower	IrDA 低能耗模式
USART_IrDAMode_Normal	IrDA 普通模式

例：

```
/* USART2 IrDA 低能耗选择 */

USART_IrDAConfig(USART2,USART_IrDAMode_LowPower);
```

21.2.20 USART_IrDACmd 函数

表 742 描述了 USART_IrDACmd 函数。

表 742. USART_IrDACmd 函数

函数名	USART_IrDACmd
函数原型	void USART_IrDACmd(USART_TypeDef* USARTx, FunctionalState Newstate)
行为描述	使能或禁用 USART IrDA 模式
输入参数 1	USARTx：x 可以为 1，2 或 3 以选择 USART 的外围设备
输入参数 2	Newstate：IrDA 模式的新状态 这个参数能够取：ENABLE 或者 DISABLE
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 使能 USART1 IrDA 模式 */
USART_IrDACmd(USART1, ENABLE);
```

21.2.21 USART_GetFlagStatus 函数

表 743 描述了 USART_GetFlagStatus 函数。

表 743. USART_GetFlagStatus 函数

函数名	USART_GetFlagStatus
函数原型	FlagStatus USART_GetFlagStatus(USART_TypeDef* USARTx, u16 USART_FLAG)
行为描述	检测指明的 USART 标记是否被置位
输入参数 1	USARTx：x 可以为 1，2 或 3 以选择 USART 的外围设备
输入参数 2	USART_FLAG：指明要检测的标记 更多关于该参数的可用值的细节请参阅章节：USART_FLAG。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

USART_FLAG

USART 能被检查的标记被列在下表中：

表 744. USART_FLAG 定义

USART_FLAG	描述
USART_FLAG_CTS	CTS 标记
USART_FLAG_LBD	LIN 间隔检测标记
USART_FLAG_TXE	传输数据寄存器空标记
USART_FLAG_TC	传输完成标记
USART_FLAG_RXNE	读数据寄存器非空标记
USART_FLAG_IDLE	Idle 线性检测
USART_FLAG_ORE	超出限定错误
USART_FLAG_NE	噪音错误
USART_FLAG_FE	帧错误
USART_FLAG_PE	奇偶校验错误

例：

```
/* 检查发送数据寄存器是否满 */
FlagStatus Status;
Status = USART_GetFlagStatus(USART1, USART_FLAG_TXE);
```

21.2.22 USART_ClearFlag 函数

表 745 描述了 USART_ClearFlag 函数。

表 745. USART_ClearFlag 函数

函数名	USART_ClearFlag
函数原型	void USART_ClearFlag(USART_TypeDef* USARTx, u16 USART_FLAG
行为描述	清除 USARTx 未决标记
输入参数 1	USARTx：x 可以为 1，2 或 3 以选择 USART 的外围设备
输入参数 2	USART_FLAG：指明要被清除的标记 更多关于该参数的可用值的细节请参阅章节: USART_FLAG。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 清除溢出错误标志 */
USART_ClearFlag(USART1,USART_FLAG_OR);
```

21.2.23 USART_GetITStatus 函数

表 746 描述了 USART_GetITStatus 函数。

表 746. USART_GetITStatus 函数

函数名	USART_GetITStatus
函数原型	ITStatus USART_GetITStatus(USART_TypeDef* USARTx, u16 USART_IT)
行为描述	检测指明的 USART 中断是否发生
输入参数 1	USARTx：x 可以为 1，2 或 3 以选择 USART 的外围设备
输入参数 2	USART_IT：指明要检测的 USART 中断源 更多关于该参数的可用值的细节请参阅章节: USART_IT。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

USART_IT

USART_IT 是被用于读取 USART 中断未决位的状态。这个参数的值从表 747 中引用。

表 747. USART_IT 定义

USART_IT	描述
USART_IT_PE	奇偶校验错误中断
USART_IT_TXE	传输中断
USART_IT_TC	传输完成中断

USART_IT_RXNE	接收中断
USART_IT_IDLE	IDLE 线性中断
USART_IT_LBD	LIN 间隔检测中断
USART_IT_CTS	CTS 中断
USART_IT_ORE	超过限度错误中断
USART_IT_NE	噪音错误中断
USART_IT_FE	帧错误中断

例：

```
/* 获取 USART1 溢出错误中断状态 */

ITStatus ErrorITStatus;

ErrorITStatus = USART_GetITStatus(USART1, USART_IT_OverrunError);
```

21.2.24 USART_ClearITPendingBit 函数

表 748 描述了 USART_ClearITPendingBit 函数。

表 748. USART_ClearITPendingBit 定义

函数名	USART_ClearITPendingBit
函数原型	void USART_ClearITPendingBit(USART_TypeDef* USARTx, u16 USART_IT)
行为描述	清除 USARTx 中断未决位

输入参数 1	USARTx：x 可以为 1，2 或 3 以选择 USART 的外围设备
输入参数 2	USART_IT：指明要被清除的中断未决位 更多关于该参数的可用值的细节请参阅章节: USART_IT。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 清除 USART1 溢出错误中断未决位 */

USART_ClearITPendingBit(USART1,USART_IT_OverrunError);
```

22 窗口看门狗（WWDG）

窗口看门狗允许检测是否有软件错误发生。一个软件错误通常由外部冲突或者导致应用程序跳出它的正常运转的不可预料的合理条件引发的。

章节 22.1 :WWDG 寄存器描述了 WWDG 固件库中使用的数据结构。章节 22.2 :固件库函数 介绍库函数。

22.1 WWDG 寄存器

WWDG 寄存器结构体，WWDG_TypeDef,是被定义在 stm32f10x_mao.h 文件中，如下：

```
typedef struct
{
  vu32 CR;
  vu32 CFR;
  vu32 SR;
} WWDG_TypeDef;
```

表 749 给出了 WWDG 的寄存器列表。

表 749. WWDG 寄存器

寄存器	描述
CR	WWDG 控制寄存器
CFR	WWDG 配置寄存器

SR	WWDG 状态寄存器

WW 外设在 stm32f10x_map.h 文件中声明，其声明如下：

```
#define PERIPH_BASE              ((u32)0x40000000)
#define APB1PERIPH_BASE          PERIPH_BASE
#define APB2PERIPH_BASE          (PERIPH_BASE + 0x10000)
#define AHBPERIPH_BASE           (PERIPH_BASE + 0x20000)
#define WWDG_BASE                (APB1PERIPH_BASE + 0x2C00)
#ifndef DEBUG
...
#ifdef _WWDG
  #define WWDG                   ((WWDG_TypeDef *) WWDG_BASE)
#endif /*_WWDG */
...
#else    /* DEBUG */
...
#ifdef _WWDG
  EXT WWDG_TypeDef               *WWDG;
#endif /*_WWDG */
...
#endif
```

当使用 Debug 模式时，WWDG 指针在 stm32f10x_conf.h 中被初始化。

#ifdef _WWDG

WWDG = (WWDG_TypeDef *) WWDG_BASE;

#endif /*_WWDG */

为了访问窗口看门狗寄存器，必须在 stm32f10x_conf.h,定义_WWDG，如下：

#define _WWDG

22.2 固件库函数

表 750 给出了 WWDG 库中的各种函数。

表 750. WWDG 固件库函数

函数名	描述
WWDG_DeInit	重置 WWDG 外围设备寄存器为他们的默认重置值
WWDG_SetPrescaler	设置 WWDG 预分频器
WWDG_SetWindow Value	设置 WWDG 窗口值
WWDG_EnableIT	使能 WWDG 提前唤醒中断（EWI）
WWDG_SetCounter	设置 WWDG 计数器的值
WWDG_Enable	使能 WWDG 和载入计数器的值
WWDG_GetFlagStat us	检测提前唤醒中断标记是否被置位
WWDG_ClearFlag	清除提前唤醒中断标记

22.2.1 WWDG_DeInit 函数

表 751 描述了 WWDG_DeInit 函数。

表 748. WWDG_DeInit 函数

函数名	WWDG_DeInit
函数原型	void WWDG_DeInit(void)
行为描述	重置 WWDG 外围设备寄存器为他们的默认重置值
输入参数	无
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	RCC_APB1PeriphResetCmd

例：

```
/* 重置 WDG 寄存器 */

WWDG_DeInit();
```

22.2.2 WWDG_SetPrescaler 函数

表 752 描述了 WWDG_SetPrescaler 函数。

表 752. WWDG_SetPrescaler 函数

函数名	WWDG_SetPrescaler
函数原型	void WWDG_SetPrescaler(u32 WWDG_Prescaler)
行为描述	设置 WWDG 预分频器
输入参数	WWDG_Prescaler：指明 WWDG 预分频器 更多关于该参数的可用值的细节请参阅章节: WWDG_Prescaler。
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

WWDG_Prescaler

WWDG_Prescaler 选择 WWDG 预分频器。这个参数的取值参阅表 753。

表 753. WWDG_Prescaler 值

WWDG_Prescaler	描述
WWDG_Prescaler_1	WWDG 计数器时钟=(PCLK/4096）/1
WWDG_Prescaler_2	WWDG 计数器时钟=(PCLK/4096）/2
WWDG_Prescaler_4	WWDG 计数器时钟=(PCLK/4096）/4
WWDG_Prescaler_8	WWDG 计数器时钟=(PCLK/4096）/8

例：

/*设置 WWDG 预分频器为 8 */

WWDG_SetPrescaler(WWDG_Prescaler_8);

22.2.3 WWDG_SetWindowValue 函数

表 754 描述了 WWDG_SetWindowValue 函数。

表 754. WWDG_SetWindowValue 函数

函数名	WWDG_SetWindowValue
函数原型	void WWDG_SetWindowValue(u8 WindowValue)
行为描述	设置 WWDG 窗口值
输入参数	WindowValue：指明要和倒计数器比较的窗口值 这个个参数的值必须小于 0x80
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

/* 置 WWDG 窗口值为 0x50 */

WWDG_SetWindowValue(0x50);

22.2.4 WWDG_EnableIT 函数

表 755 描述了 WWDG_EnableIT 函数。

表 755 .WWDG_EnableIT 函数

函数名	WWDG_EnableIT
函数原型	void WWDG_EnableIT(void)
行为描述	使能 WWDG 提前唤醒中断
输入参数	无
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 使能 WWDG 提前唤醒中断 */

WWDG_EnableIT();
```

22.2.5 22.2.5 WWDG_SetCounter 函数

表 756 描述了 WWDG_SetCounter 函数。

表 756. WWDG_SetCounter 函数

函数名	WWDG_SetCounter
函数原型	void WWDG_SetCounter(u8 Counter)

行为描述	设置 WWDG 计数器的值
输入参数	Counter:：指定看门狗计数器的值 这个参数必须是 0x40 到 0x70 之间的一个数
输出参数	无
返回参数（值）	无
前提条件	无
调用函数	无

例：

```
/* 置 WWDG 计数器的值为 0x70 */
WWDG_SetCounter(0x70);
```

22.2.6 WWDG_Enable 函数

表 757 描述了 WWDG_Enable 函数。

表 757. WWDG_Enable 函数

函数名	WWDG_Enable
函数原型	void WWDG_Enable(u8 Counter)(1)
行为描述	使能 WWDG 和载入计数器的值
输入参数	计数器：指定看门狗计数器的值 这个参数必须是 0x40 到 0x7F 之间的一个数
输出参数	无

返回参数（值）	无
前提条件	无
调用函数	无

1. 一旦被使能，WWDG 不能再被禁用。

例：

```
/* 使能 WWDG 并且置计数器的值为 0x7F */
WWDG_Enable(0x7F);
```

22.2.7 WWDG_GetFlagStatus 函数

表 758 描述了 WWDG_ GetFlagStatus 函数。

表 758. WWDG_GetFlagStatus 定义

函数名	WWDG_GetFlagStatus
函数原型	FlagStatus WWDG_GetFlagStatus(void)
行为描述	检测提前唤醒中断标记是否被置位
输入参数	无
输出参数	无
返回参数（值）	提前唤醒中断比较的新状态（置位或复位）
前提条件	无
调用函数	无

例：

```
/* 检查计数器的值是否达到 0x40 */
```

```
FlagStatus Status;

Status = WWDG_GetFlagStatus();

if(Status == RESET)

{

…

}

else

{

…

}
```

22.2.8 WWDG_ClearFlag 函数

表 759 描述了 WWDG_ClearFlag 函数。

表 759. WWDG_ClearFlag 定义

函数名	WWDG_ClearFlag
函数原型	void WWDG_ClearFlag(void)
行为描述	清除提前唤醒中断标记
输入参数	无
输出参数	无
返回参数（值）	无

前提条件	无
调用函数	无

例：

```
/* 清除 EWI 标志 */

WWDG_ClearFlag();
```

23 版本历史

表 760.版本历史

日期	版本	变化
2007-5-28	1	初始版本

24 版权声明：

MXCHIP Corporation 拥有对该中文版文档的所有权和使用权

意法半导体（ST）拥有对英文原版文档的所有权和使用权