32 位基于 ARM 微控制器 STM32F101xx 与 STM32F103xx 固件函数库

介绍

本手册介绍了 32 位基于 ARM 微控制器 STM32F101xx 与 STM32F103xx 的固件函数库。

该函数库是一个固件函数包，它由程序、数据结构和宏组成，包括了微控制器所有外设的性能特征。该函数库还包括每一个外设的驱动描述和应用实例。通过使用本固件函数库，无需深入掌握细节，用户也可以轻松应用每一个外设。因此，使用本固态函数库可以大大减少用户的程序编写时间，进而降低开发成本。

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

所有的驱动源代码都符合“Strict ANSI-C”标准（项目于范例文件符合扩充 ANSI-C 标准）。我们已经把驱动源代码文档化，他们同时兼容 MISRA-C 2004 标准（根据需要，我们可以提供兼容矩阵）。由于整个固态函数库按照“Strict ANSI-C”标准编写，它不受不同开发环境的影响。仅对话启动文件取决于开发环境。

该固态函数库通过校验所有库函数的输入值来实现实时错误检测。该动态校验提高了软件的鲁棒性。实时检测适合于用户应用程序的开发和调试。但这会增加了成本，可以在最终应用程序代码中移去，以优化代码大小和执行速度。想要了解更多细节，请参阅 Section 2.5。

因为该固件库是通用的，并且包括了所有外设的功能，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，用户可以直接使用之，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设的一份参考资料，根据实际需求对其进行调整。

此份固件库用户手册的整体架构如下：

- 定义，文档约定和固态函数库规则。
- 固态函数库概述（包的内容，库的架构），安装指南，库使用实例。
- 固件库具体描述：设置架构和每个外设的函数。

STM32F101xx 和 STM32F103xx 在整个文档中被写作 STM32F101x。

目录

表格目录

1. 文档和库规范

本用户手册和固态函数库按照以下章节所描述的规范编写。

1.1 缩写

Table 1. 本文档所有缩写定义

缩写	外设/单元
ADC	模数转换器
BKP	备份寄存器
CAN	控制器局域网模块
DMA	直接内存存取控制器
EXTI	外部中断事件控制器
FLASH	闪存存储器
GPIO	通用输入输出
I2C	内部集成电路
IWDG	独立看门狗
NVIC	嵌套中断向量列表控制器
PWR	电源/功耗控制
RCC	复位与时钟控制器
RTC	实时时钟
SPI	串行外设接口
SysTick	系统嘀嗒定时器
TIM	通用定时器
TIM1	高级控制定时器
USART	通用同步异步接收发射端
WWDG	窗口看门狗

1.2 命名规则

固态函数库遵从以下命名规则

PPP 表示任一外设缩写，例如：ADC。更多缩写相关信息参阅章节 1.1 缩写

系统、源程序文件和头文件命名都以***"stm32f10x_"***作为开头，例如：***stm32f10x_conf.h***。

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

寄存器作为常量处理。他们的命名都由英文字母大写书写。在大多数情况下，他们采用与缩写规范与本用户手册一致。

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

名为 ***PPP_Init*** 的函数，其功能是根据 ***PPP_InitTypeDef*** 中指定的参数，初始化外设 PPP，例如 ***TIM_Init***.

名为 ***PPP_DeInit*** 的函数，其功能为复位外设 PPP 的所有寄存器至缺省值，例如 ***TIM_DeInit***.

名为 ***PPP_StructInit*** 的函数，其功能为通过设置 ***PPP_InitTypeDef*** 结构中的各种参数来定义外设的功能，例如：***USART_StructInit***

名为 ***PPP_Cmd*** 的函数，其功能为使能或者失能外设 PPP，例如： ***SPI_Cmd***.

名为 ***PPP_ITConfig*** 的函数，其功能为使能或者失能来自外设 PPP 某中断源，例如： ***RCC_ITConfig***.

名为 ***PPP_DMAConfig*** 的函数，其功能为使能或者失能外设 PPP 的 DMA 接口，例如：***TIM1_DMAConfig***.

用以配置外设功能的函数，总是以字符串“Config”结尾，例如 ***GPIO_PinRemapConfig***.

名为 ***PPP_GetFlagStatus*** 的函数，其功能为检查外设 PPP 某标志位被设置与否，例如：***I2C_GetFlagStatus***.

名为 ***PPP_ClearFlag*** 的函数，其功能为清除外设 PPP 标志位，例如：***I2C_ClearFlag***.

名为 ***PPP_GetITStatus*** 的函数，其功能为判断来自外设 PPP 的中断发生与否，例如：***I2C_GetITStatus***.

名为 ***PPP_ClearITPendingBit*** 的函数，其功能为清除外设 PPP 中断待处理标志位，例如：***I2C_ClearITPendingBit***.

1.3 编码规则

本章节描述了固态函书库的编码规则。

1.3.1 变量

固态函数库定义了 24 个变量类型，他们的类型和大小是固定的。在文件 ***stm32f10x_type.h*** 中我们定义了这些变量：

```
typedef signed long s32;
typedef signed short s16;
typedef signed char s8;
typedef signed long const sc32; /* Read Only */
typedef signed short const sc16; /* Read Only */
typedef signed char const sc8; /* Read Only */
typedef volatile signed long vs32;
typedef volatile signed short vs16;
typedef volatile signed char vs8;
typedef volatile signed long const vsc32; /* Read Only */
typedef volatile signed short const vsc16; /* Read Only */
typedef volatile signed char const vsc8; /* Read Only */
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 布尔型

在文件 ***stm32f10x_type.h*** 中，布尔形变量被定义如下：

```
typedef enum
{
```

```
FALSE = 0,
TRUE = !FALSE
} bool;
```

1.3.3 标志位状态类型

在文件 ***stm32f10x_type.h*** 中，我们定义标志位类型（***FlagStatus*** type）的 2 个可能值为“设置”与“重置”（***SET*** or ***RESET***）。

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

1.3.4 功能状态类型

在文件 ***stm32f10x_type.h*** 中，我们定义功能状态类型（***FunctionalState*** type）的 2 个可能值为“使能”与“失能”（***ENABLE*** or ***DISABLE***）。

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

1.3.5 错误状态类型

在文件 ***stm32f10x_type.h*** 中，我们错误状态类型类型（***ErrorStatus*** type）的 2 个可能值为“成功”与“出错”（***SUCCESS*** or ***ERROR***）。

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

1.3.6 外设

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

外设控制寄存器结构

文件 ***stm32f10x_map.h*** 包含了所有外设控制寄存器的结构，下例为 SPI 寄存器结构的声明：

```
/*------------------ 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;
```

寄存器命名遵循上节的寄存器缩写命名规则。RESERVEDi（i 为一个整数索引值）表示被保留区域。

外设声明

文件 ***stm32f10x_map.h*** 包含了所有外设的声明，下例为 SPI 外设的声明：

```
#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 */
```

如果用户希望使用外设 SPI，那么必须在文件 ***stm32f10x_conf.h*** 中定义***_SPI*** 标签

通过定义标签***_SPIn***，用户可以访问外设 ***SPIn*** 的寄存器。例如，用户必须在文件 ***stm32f10x_conf.h*** 中定义标签***_SPI2***，否则是不能访问 ***SPI2*** 的寄存器的。在文件 ***stm32f10x_conf.h*** 中，用户可以按照下例定义标签 ***_SPI*** 和 ***_SPIn***

```
#define _SPI
#define _SPI1
#define _SPI2
```

每个外设都有若干寄存器专门分配给标志位。我们按照相应的结构定义这些寄存器。标志位的命名，同样遵循上节的外设缩写规范，以***‘PPP_FLAG_’***开始。对于不同的外设，标志位都被定义在相应的文件 ***stm32f10x_ppp.h*** 中。

用户想要进入除错（DEBUG）模式的话，必须在文件 ***stm32f10x_conf.h*** 中定义标签 ***DEBUG***。
这样会在 SRAM 的外设结构部分创建一个指针。因此我们可以简化除错过程，并且通过转储外设获得来获得所有寄存器的状态。在所有情况下，***SPI2*** 都是一个指向外设 SPI2 首地址的指针。

变量 DEBUG 可以仿照下例定义：

```
#define DEBUG 1
```

可以初始化 DEBUG 模式与文件 ***stm32f10x_lib.c*** 中如下：

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

Note：1 当用户选择 DEBUG 模式，宏 ***assert_param*** 被扩展，同时运行时间检查功能也在固态函数库代码中被激活。

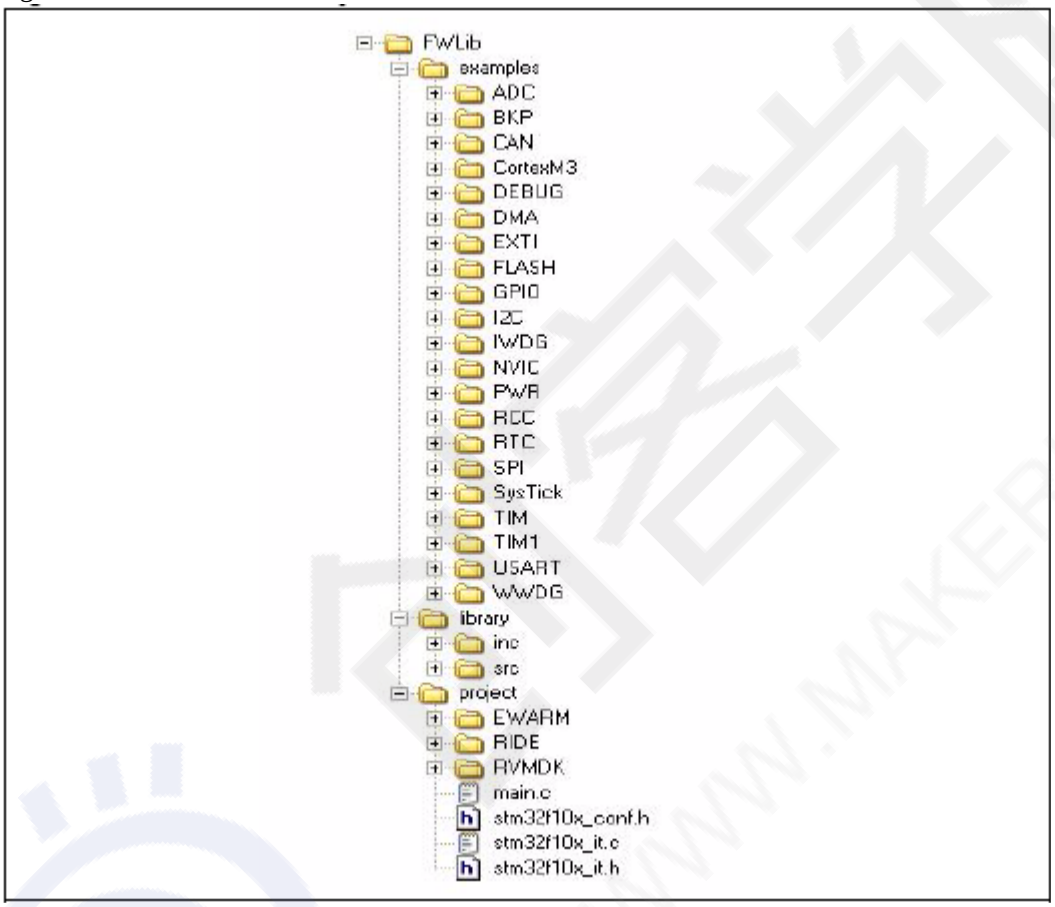
2 进入 DEBUG 模式会增大代码的尺寸，降低代码的运行效率。因此，我们强烈建议仅仅在除错的时候使用相应代码，在最终的应用程序中，删除它们。

2. 固件函数库

2.1 压缩包描述

STM32F10x 固件函数库被压缩在一个 zip 文件中。解压该文件会产生一个文件夹：***STM32F10xFWLib\FWLib***，包含如下所示的子文件夹：

Figure 1：固件函数库文件夹结构



2.1.1 文件夹Examples

文件夹 Examples，对应每一个 STM32 外设，都包含一个子文件夹。这些子文件夹包含了整套文件，组成典型的例子，来示范如何使用对应外设。这些文件有：
readme.txt：每个例子的简单描述和使用说明。
stm32f10x_conf.h：该头文件设置了所有使用到的外设，由不同的“DEFINE”语句组成。
stm32f10x_it.c：该源文件包含了所有的中断处理程序（如果未使用中断，则所有的函数体都为空）。
stm32f10x.it.h：该头文件包含了所有的中断处理程序的原形。
main.c：例程代码
注：所有的例程的使用，都不受不同软件开发环境的影响。

2.1.2 文件夹Library

文件夹 Library 包含组成固件函数库核心的所有子文件夹和文件：
■ 子文件夹 inc 包含了固件函数库所需的头文件，用户无需修改该文件夹：

- ***stm32f10x_type.h***：所有其他文件使用的通用数据类型和枚举。
- ***stm32f10x_map.h***：外设存储器映像和寄存器数据结构。
- ***stm32f10x_lib.h***：主头文件夹，包含了其他头文件。
- ***stm32f10x_ppp.h***：每个外设对应一个头文件，包含了该外设使用的函数原形，数据结构和枚举。
- ***cortexm3_macro.h***： 文件 cortexm3_macro.s 对应的头文件。

■ 子文件夹 src 包含了固件函数库所需的源文件，用户无需修改该文件夹：

- ***stm32f10x_ppp.c***：每个外设对应一个源文件，包含了该外设使用的函数体
- ***stm32f10x_lib.c***：初始化所有外设的指针

注：所有代码都按照 Strict ANSI-C 标准书写，都不受不同软件开发环境的影响。

2.1.3 文件夹Project

文件夹 Project 包含了一个标准的程序项目模板，包括库文件的编译和所有用户可修改的文件，可用以建立新的工程。

- ***stm32f10x_conf.h***：项目配置头文件，默认为设置了所有的外设。
- ***stm32f10x_it.c***：该源文件包含了所有的中断处理程序（所有的函数体默认为空）。

stm32f10x_it.h：该头文件包含了所有的中断处理程序的原形。
main.c：主函数体
文件夹 EWARM，RVMDK，RIDE：用于不同开发环境使用，详情查询各文件夹下的文件 readme.txt。

2.2 固件函数库文件描述

Table 2 列举和描述了固件函数库使用的所有文件。

固件函数库的体系和文件相互包括的联系表示在 Figure 2 中。每一个外设都有一个对应的源文件：stm32f10x_ppp.c 和一个对应的头文件：stm32f10x_ppp.h。
文件 stm32f10x_ppp.c 包含了使用外设 PPP 所需的所有固件函数。提供所有外设一个存储器映像文件 stm32f10x_map.h。它包含了所有寄存器的声明，既可以用于 Debug 模式也可以用于 release 模式。

头文件 stm32f10x_lib.h 包含了所有外设头文件的头文件。它是唯一一个用户需要包括在自己应用中的文件，起到应用和库之间界面的作用。

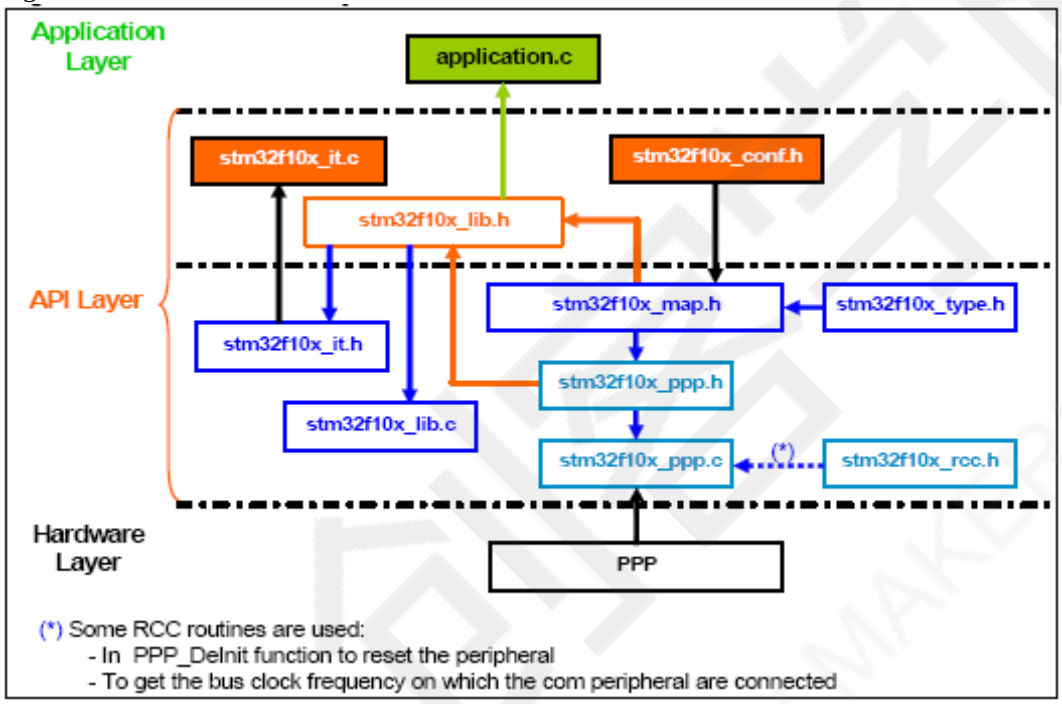
文件 stm32f10x_conf.h 是唯一一个需要由用户修改的文件。它作为应用和库之间的界面，指定了一系列参数。

Table 2. 固件函数库文件描述

文件名	描述
stm32f10x_conf.h	参数设置文件，起到应用和库之间界面的作用。 用户必须在运行自己的程序前修改该文件。 用户可以利用模板使能或者失能外设。也可以修改外部晶振的参数。 也可以是用该文件在编译前使能 Debug 或者 release 模式。
main.c	主函数体示例。
stm32f10x_it.h	头文件，包含所有中断处理函数原形。
stm32f10x_it.c	外设中断函数文件。 用户可以加入自己的中断程序代码。对于指向同一个中断向量的多个不同中断请求，可以利用函数通过判断外设的中断标志位来确定准确的中断源。固件函数库提供了这些函数的名称。
stm32f10x_lib.h	包含了所有外设的头文件的头文件。 它是唯一一个用户需要包括在自己应用中的文件，起到应用和库之间界面的作用。
stm32f10x_lib.c	Debug 模式初始化文件。

	它包括多个指针的定义，每个指针指向特定外设的首地址，以及在 Debug 模式被使能时，被调用的函数的定义。
stm32f10x_map.h	该文件包含了存储器映像和所有寄存器物理地址的声明，既可以用于 Debug 模式也可以用于 release 模式。所有外设都使用该文件。
stm32f10x_type.h	通用声明文件。 包含所有外设驱动使用的通用类型和常数。
stm32f10x_ppp.c	由 C 语言编写的外设 PPP 的驱动源程序文件。
stm32f10x_ppp.h	外设 PPP 的头文件。包含外设 PPP 函数的定义，和这些函数使用的变量。
cortexm3_macro.h	文件 cortexm3_macro.s 的头文件
cortexm3_macro.s	Cortex-M3 内核特殊指令的指令包装。

Figure 2. 固件函数库文件体系结构



2.3 外设的初始化和设置

本节按步骤描述了如何初始化和设置任意外设。这里 PPP 代表任意外设。

1. 在主应用文件中，声明一个结构 PPP_InitTypeDef，例如：

```
PPP_InitTypeDef PPP_InitStructure;
```

这里 PPP_InitStructure 是一个位于内存中的工作变量，用来初始化一个或者多个外设 PPP。

2. 为变量 PPP_InitStructure 的各个结构成员填入允许的值。可以采用以下 2 种方式：

a）按照如下程序设置整个结构体

```
PPP_InitStructure.member1 = val1;
PPP_InitStructure.member2 = val2;
PPP_InitStructure.memberN = valN;
/* where N is the number of the structure members */
```

以上步骤可以合并在同一行里，用以优化代码大小：

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

b）仅设置结构体中的部分成员：这种情况下，用户应当首先调用函数 PPP_SturcInit(..)来初始化变量 PPP_InitStructure，然后再修改其中需要修改的成员。这样可以保证其他成员的值（多为缺省值）被正确填入。

```
PPP_StructInit(&PPP_InitStructure);
PP_InitStructure.memberX = valX;
PPP_InitStructure.memberY = valY;
```

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

3. 调用函数 PPP_Init(..)来初始化外设 PPP。
4. 在这一步，外设 PPP 已被初始化。可以调用函数 PPP_Cmd(..)来使能之。

```
PPP_Cmd(PPP, ENABLE);
```

可以通过调用一系列函数来使用外设。每个外设都拥有各自的功能函数。更多细节参阅 Section3 外设固件概述。

注：1. 在设置一个外设前，必须调用以下一个函数来使能它的时钟：

```
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_PPPx, ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_PPPx, ENABLE);
RCC_APB1PeriphClockCmd(RCC_APB1Periph_PPPx, ENABLE);
```

2. 可以调用函数 PPP_Deinit(..)来把外设 PPP 的所有寄存器复位为缺省值：

```
PPP_DeInit(PPP)
```

3. 在外设设置完成以后，继续修改它的一些参数，可以参照如下步骤：

```
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 位段（Bit-Banding）

Cortex™-M3 存储器映像包括两个位段(bit-band)区。这两个位段区将别名存储器区中的每个字映射到位段存储器区的一个位，在别名存储区写入一个字具有对位段区的目标位执行读-改-写操作的相同效果。
所有 STM32F10x 外设寄存器都被映射到一个位段(bit-band)区。这个特性在各个函数中对单个比特进行置 1/置 0 操作时被大量使用，用以减小和优化代码尺寸。
Section 2.4.1 和 Section 2.4.2 给出了外设固件函数库中如何实现位段访问的描述。

2.4.1 映射公式

映射公式给出了别名区中的每个字是如何对应位带区的相应位的，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 是目标位在存取器位段区中的位置
bit_word_addr 是别名存储器区中字的地址，它映射到某个目标位。
bit_band_base 是别名区的起始地址。
byte_offset 是包含目标位的字节在位段里的序号
bit_number 是目标位所在位置（0-31）

2.4.2 应用实例

下例展现了如何把寄存器 RCC_CR 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))
```

编写一个使能/失能 PLL 的函数，步骤如下：

...

```
#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;
}
}
```

Using bit-band access this function will be coded as follows:

```
void RCC_PLLCmd(FunctionalState NewState)
{
*(vu32 *) CR_PLLON_BB = (u32)NewState;
}
```

2.5 运行时间检测

固件函数库通过检查库函书的输入来实现运行时间错误侦测。通过使用宏 ***assert_param*** 来实现运行时间检测。所有要求输入参数的函数都使用这个宏。它可以检查输入参数是否在允许的范围之内。

例：函数 PWR_ClearFlag

stm32f10x_pwr.c:

```
void PWR_ClearFlag(u32 PWR_FLAG)
{
/* Check the parameters */
assert_param(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))
```

如果传给宏 ***assert_param*** 的参数为 ***false***，则调用函数 ***assert_failed*** 并返回被错误调用的函数所在的文件名和行数。如果传给宏 ***assert_param*** 的参数为 ***true***，则无返回值。

宏 assert_param 编写于文件 stm32f10x_conf.h 中：

```
/* Exported macro -------------------------------------------------------------*/
#ifdef DEBUG
/*******************************************************************************
* Macro Name : assert_param
* Description : The assert_param macro is used for function's parameters check.
* 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_param(expr) ((expr) ? (void)0 : assert_failed((u8 *)__FILE__,
__LINE__))
/* Exported functions ------------------------------------------------------- */
void assert_failed(u8* file, u32 line);
#else
#define assert_param(expr) ((void)0)
#endif /* DEBUG */
```

函数 ***assert_failed*** 编写于文件 ***main.c*** 或者其他用户 C 文件：

```
#ifdef DEBUG
```

```
/*******************************************************************
* Function name : assert_failed
* Description : Reports the name of the source file and the source line number
* where the assert_param error has occurred.
* Input : - file: pointer to the source file name
* - line: assert_param 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：
运行时间检查，即宏 ***assert_param*** 应当只在库在 Debug 模式下编译时使用。建议在用户应用代码的开发和调试阶段使用运行时间检查，在最终的代码中去掉它们以改进代码尺寸和速度。

如果用户仍然希望在最终的代码中保留这项功能，可以在调用库函数前，重新使用宏 ***assert_param*** 来测试输入参数。

3. 外设固件概述

本节系统描述了每一个外设固件函数库。完整地描述所有相关函数并提供如何使用他们的例子。

函数的描述按如下格式进行：

Table 3. 函数描述格式

函数名	外设函数的名称
函数原形	原形声明
功能描述	简要解释函数是如何执行的
输入参数{x}	输入参数描述
输出参数{x}	输出参数描述
返回值	函数的返回值
先决条件	调用函数前应满足的要求
被调用函数	其他被该函数调用的库函数

4. 模拟/数字转换器

模拟/数字转换器（ADC）是一种提供可选择多通道输入，逐次逼近型的模数转换器。分辨率为 12 位。

Section 4.1 ADC 寄存器结构描述了固件函数库所使用的数据结构，Section 4.2 固件库函数介绍了函数库里的所有函数。

4.1 ADC寄存器结构

ADC 寄存器结构，***ADC_TypeDef***，定义于文件***"stm32f10x_map.h"***如下：

```
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;
```

Table 4 给出了 ADC 寄存器列表：

Table 4. ADC 寄存器

寄存器	描述
SR	ADC 状态寄存器
CR1	ADC 控制寄存器 1
CR2	ADC 控制寄存器 2
SMPR1	ADC 采样时间寄存器 1
SMPR2	ADC 采样时间寄存器 2
JOFR1	ADC 注入通道偏移寄存器 1
JOFR2	ADC 注入通道偏移寄存器 2
JOFR3	ADC 注入通道偏移寄存器 3
JOFR4	ADC 注入通道偏移寄存器 4
HTR	ADC 看门狗高阈值寄存器
LTR	ADC 看门狗低阈值寄存器
SQR1	ADC 规则序列寄存器 1
SQR2	ADC 规则序列寄存器 2
SQR3	ADC 规则序列寄存器 3
JSQR1	ADC 注入序列寄存器
DR1	ADC 规则数据寄存器 1

DR2	ADC 规则数据寄存器 2
DR3	ADC 规则数据寄存器 3
DR4	ADC 规则数据寄存器 4

2 个 ADC 外设声明于文件***"stm32f10x_map.h"***：

```
...
#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 Debug 模式时，初始化指针 ***ADC1*** 和 ***ADC2*** 与文件***"stm32f10x_lib.c"***：

```
...
#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*** 必须在文件***"stm32f10x_conf.h"***中定义如下：

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

4.2 ADC库函数

Table 5. 为 ADC 固件库函数列表

Table 5. ADC 固件库函数

函数名	描述
ADC_DeInit	将外设 ADCx 的全部寄存器重设为缺省值
ADC_Init	根据 ADC_InitStruct 中指定的参数初始化外设 ADCx 的寄存器
ADC_StructInit	把 ADC_InitStruct 中的每一个参数按缺省值填入
ADC_Cmd	使能或者失能指定的 ADC
ADC_DMACmd	使能或者失能指定的 ADC 的 DMA 请求
ADC_ITConfig	使能或者失能指定的 ADC 的中断

ADC

ADC_ResetCalibration	重置指定的 ADC 的校准寄存器
ADC_GetResetCalibrationStatus	获取 ADC 重置校准寄存器的状态
ADC_StartCalibration	开始指定 ADC 的校准程序
ADC_GetCalibrationStatus	获取指定 ADC 的校准状态
ADC_SoftwareStartConvCmd	使能或者失能指定的 ADC 的软件转换启动功能
ADC_GetSoftwareStartConvStatus	获取 ADC 软件转换启动状态
ADC_DiscModeChannelCountConfig	对 ADC 规则组通道配置间断模式
ADC_DiscModeCmd	使能或者失能指定的 ADC 规则组通道的间断模式
ADC_RegularChannelConfig	设置指定 ADC 的规则组通道，设置它们的转化顺序和采样时间
ADC_ExternalTrigConvConfig	使能或者失能 ADCx 的经外部触发启动转换功能
ADC_GetConversionValue	返回最近一次 ADCx 规则组的转换结果
ADC_GetDuelModeConversionValue	返回最近一次双 ADC 模式下的转换结果
ADC_AutoInjectedConvCmd	使能或者失能指定 ADC 在规则组转化后自动开始注入组转换
ADC_InjectedDiscModeCmd	使能或者失能指定 ADC 的注入组间断模式
ADC_ExternalTrigInjectedConvConfig	配置 ADCx 的外部触发启动注入组转换功能
ADC_ExternalTrigInjectedConvCmd	使能或者失能 ADCx 的经外部触发启动注入组转换功能
ADC_SoftwareStartinjectedConvCmd	使能或者失能 ADCx 软件启动注入组转换功能
ADC_GetsoftwareStartinjectedConvStatus	获取指定 ADC 的软件启动注入组转换状态
ADC_InjectedChannleConfig	设置指定 ADC 的注入组通道，设置它们的转化顺序和采样时间
ADC_InjectedSequencerLengthConfig	设置注入组通道的转换序列长度
ADC_SetinjectedOffset	设置注入组通道的转换偏移值
ADC_GetInjectedConversionValue	返回 ADC 指定注入通道的转换结果
ADC_AnalogWatchdogCmd	使能或者失能指定单个/全体，规则/注入组通道上的模拟看门狗
ADC_AnalogWatchdongThresholdsConfig	设置模拟看门狗的高/低阈值
ADC_AnalogWatchdongSingleChannelConfig	对单个 ADC 通道设置模拟看门狗
ADC_TampSensorVrefintCmd	使能或者失能温度传感器和内部参考电压通道
ADC_GetFlagStatus	检查制定 ADC 标志位置 1 与否
ADC_ClearFlag	清除 ADCx 的待处理标志位
ADC_GetITStatus	检查指定的 ADC 中断是否发生
ADC_ClearITPendingBit	清除 ADCx 的中断待处理位

4.2.1 函数ADC_DeInit

Table 6. 描述了函数 ADC_DeInit

Table 6 .函数 ADC_DeInit

函数名	ADC_DeInit
函数原形	void ADC_DeInit(ADC_TypeDef* ADCx)
功能描述	将外设 ADCx 的全部寄存器重设为缺省值
输入参数 1	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输出参数 2	无
返回值	无
先决条件	无
被调用函数	RCC_APB2PeriphClockCmd()

例：

```
/* Resets ADC2 */
ADC_DeInit(ADC2);
```

4.2.2 函数ADC_Init

Table 7. 描述了函数 ADC_Init

Table 7. 函数 ADC_Init

函数名	ADC_Init
函数原形	void ADC_Init(ADC_TypeDef* ADCx, ADC_InitTypeDef* ADC_InitStruct)
功能描述	根据 ADC_InitStruct 中指定的参数初始化外设 ADCx 的寄存器
输入参数 1	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输入参数 2	ADC_InitStruct：指向结构 ADC_InitTypeDef 的指针，包含了指定外设 ADC 的配置信息 参阅：4.2.3 ADC_StructInit 获得 ADC_InitStruct 值的完整描述
输出参数	无
返回值	无
先决条件	无
被调用函数	无

ADC_InitTypeDef structure

ADC_InitTypeDef 定义于文件“stm32f10x_adc.h”：

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

ADC_Mode

ADC_Mode 设置 ADC 工作在独立或者双 ADC 模式。参阅 Table 8.获得这个参数的所有成员。

Table 8. 函数 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 工作在同步规则和同步注入模式
ADC_Mode_RegSimult_AlterTrig	ADC1 和 ADC2 工作在同步规则模式和交替触发模式
ADC_Mode_InjecSimult_FastInterl	ADC1 和 ADC2 工作在同步规则模式和快速交替模式
ADC_Mode_InjecSimult_SlowInterl	ADC1 和 ADC2 工作在同步注入模式和慢速交替模式
ADC_Mode_InjecSimult	ADC1 和 ADC2 工作在同步注入模式
ADC_Mode_RegSimult	ADC1 和 ADC2 工作在同步规则模式
ADC_Mode_FastInterl	ADC1 和 ADC2 工作在快速交替模式
ADC_Mode_SlowInterl	ADC1 和 ADC2 工作在慢速交替模式
ADC_Mode_AlterTrig	ADC1 和 ADC2 工作在交替触发模式

ADC_ScanConvMode

ADC_ScanConvMode 规定了模数转换工作在扫描模式（多通道）还是单次（单通道）模式。可以设置这个参数为 ENABLE 或者 DISABLE。

ADC_ContinuousConvMode

ADC_ContinuousConvMode 规定了模数转换工作在连续还是单次模式。可以设置这个参数为 ENABLE 或者 DISABLE。

ADC_ExternalTrigConv

ADC_ExternalTrigConv 定义了使用外部触发来启动规则通道的模数转换，这个参数可以取的值见 Table 9.

Table 9. ADC_ExternalTrigConv 定义表

ADC_ExternalTrigConv	描述
ADC_ExternalTrigConv_T1_CC1	选择定时器 1 的捕获比较 1 作为转换外部触发
ADC_ExternalTrigConv_T1_CC2	选择定时器 1 的捕获比较 2 作为转换外部触发
ADC_ExternalTrigConv_T1_CC3	选择定时器 1 的捕获比较 3 作为转换外部触发
ADC_ExternalTrigConv_T2_CC2	选择定时器 2 的捕获比较 2 作为转换外部触发
ADC_ExternalTrigConv_T3_TRGO	选择定时器 3 的 TRGO 作为转换外部触发
ADC_ExternalTrigConv_T4_CC4	选择定时器 4 的捕获比较 4 作为转换外部触发
ADC_ExternalTrigConv_Ext_IT11	选择外部中断线 11 事件作为转换外部触发
ADC_ExternalTrigConv_None	转换由软件而不是外部触发启动

ADC_DataAlign

ADC_DataAlign 规定了 ADC 数据向左边对齐还是向右边对齐。这个参数可以取的值见 Table 10.

Table 10. ADC_DataAlign 定义表

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_NbreOfChannel 规定了顺序进行规则转换的 ADC 通道的数目。这个数目的取值范围是 1 到 16。

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);
```

注意：为了能够正确地配置每一个 ADC 通道，用户在调用 ADC_Init()之后，必须调用 ADC_ChannelConfig()来配置每个所使用通道的转换次序和采样时间。

4.2.3 函数ADC_StructInit

Table 11. 描述了函数 ADC_StructInit

Table 11. 函数 ADC_StructInit

函数名	ADC_StructInit
函数原形	void ADC_StructInit(ADC_InitTypeDef* ADC_InitStruct)
功能描述	把 ADC_InitStruct 中的每一个参数按缺省值填入
输入参数	ADC_InitStruct：指向结构 ADC_InitTypeDef 的指针，待初始化
输出参数	无
返回值	无
先决条件	无
被调用函数	无

ADC_InitStruct 的成员缺省值如下：

Table 12. ADC_InitStruct 缺省值

成员	缺省值
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

例：

```
/* Initialize a ADC_InitTypeDef structure. */
ADC_InitTypeDef ADC_InitStructure;
ADC_StructInit(&ADC_InitStructure);
```

4.2.4 函数ADC_Cmd

Table 13. 描述了函数 ADC_Cmd

Table 13. 函数 ADC_Cmd

函数名	ADC_Cmd
函数原形	void ADC_Cmd(ADC_TypeDef* ADCx, FunctionalState NewState)
功能描述	使能或者失能指定的 ADC
输入参数 1	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输入参数 2	NewState：外设 ADCx 的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable ADC1 */
ADC_Cmd(ADC1, ENABLE);
```

注意：函数 ADC_Cmd 只能在其他 ADC 设置函数之后被调用。

4.2.5 函数ADC_DMACmd

Table 14. 描述了函数 ADC_DMACmd

Table 14. 函数 ADC_DMACmd

函数名	ADC_DMACmd
函数原形	ADC_DMACmd(ADC_TypeDef* ADCx, FunctionalState NewState)
功能描述	使能或者失能指定的 ADC 的 DMA 请求
输入参数 1	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输入参数 2	NewState：ADC DMA 传输的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable ADC2 DMA transfer */
ADC_DMACmd(ADC2, ENABLE);
```

4.2.6 函数ADC_ITConfig

Table 15. 描述了函数 ADC_ITConfig

Table 15. 函数 ADC_ITConfig

函数名	ADC_ITConfig
函数原形	void ADC_ITConfig(ADC_TypeDef* ADCx, u16 ADC_IT, FunctionalState NewState)
功能描述	使能或者失能指定的 ADC 的中断
输入参数 1	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输入参数 2	ADC_IT：将要被使能或者失能的指定 ADC 中断源 参阅章节 ADC_IT 获得该参数可取值的更多细节
输入参数 3	NewState：指定 ADC 中断的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

ADC_IT

ADC_IT 可以用来使能或者失能 ADC 中断。可以使用下表中的一个参数，或者他们的组合。

Table 16. ADC_IT 定义表

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

例：

```
/* Enable ADC2 EOC and AWDOG interrupts */
ADC_ITConfig(ADC2, ADC_IT_EOC | ADC_IT_AWD, ENABLE);
```

4.2.7 函数ADC_ResetCalibration

Table 17. 描述了函数 ADC_ResetCalibration

Table 17. 函数 ADC_ResetCalibration

函数名	ADC_ResetCalibration
函数原形	void ADC_ResetCalibration(ADC_TypeDef* ADCx)
功能描述	重置指定的 ADC 的校准寄存器
输入参数	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Reset the ADC1 Calibration registers */
ADC_ResetCalibration(ADC1);
```

4.2.8 函数ADC_GetResetCalibrationStatus

Table 18. 描述了函数 ADC_ GetResetCalibrationStatus

Table 18. 函数 ADC_ GetResetCalibrationStatus

函数名	ADC_ GetResetCalibrationStatus
函数原形	FlagStatus ADC_GetResetCalibrationStatus(ADC_TypeDef* ADCx)
功能描述	获取 ADC 重置校准寄存器的状态
输入参数	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输出参数	无
返回值	ADC 重置校准寄存器的新状态（SET 或者 RESET）
先决条件	无
被调用函数	无

例：

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

4.2.9 函数ADC_StartCalibration

Table 19. 描述了函数 ADC_StartCalibration

Table 19. 函数 ADC_StartCalibration

函数名	ADC_StartCalibration
函数原形	void ADC_StartCalibration(ADC_TypeDef* ADCx)
功能描述	开始指定 ADC 的校准状态
输入参数	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Start the ADC2 Calibration */
ADC_StartCalibration(ADC2);
```

4.2.10 函数ADC_GetCalibrationStatus

Table 20. 描述了函数 ADC_GetCalibrationStatus

Table 20. 函数 ADC_GetCalibrationStatus

函数名	ADC_GetCalibrationStatus
函数原形	FlagStatus ADC_GetCalibrationStatus(ADC_TypeDef* ADCx)
功能描述	获取指定 ADC 的校准程序
输入参数	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输出参数	无
返回值	ADC 校准的新状态（SET 或者 RESET）
先决条件	无
被调用函数	无

例：

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

4.2.11 函数ADC_SoftwareStartConvCmd

Table 21. 描述了函数 ADC_SoftwareStartConvCmd

Table 21. 函数 ADC_SoftwareStartConvCmd

函数名	ADC_SoftwareStartConvCmd
函数原形	void ADC_SoftwareStartConvCmd(ADC_TypeDef* ADCx, FunctionalState NewState)
功能描述	使能或者失能指定的 ADC 的软件转换启动功能
输入参数 1	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输入参数 2	NewState：指定 ADC 的软件转换启动新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Start by software the ADC1 Conversion */
ADC_SoftwareStartConvCmd(ADC1, ENABLE);
```

4.2.12 函数ADC_GetSoftwareStartConvStatus

Table 22. 描述了函数 ADC_GetSoftwareStartConvStatus

Table 22. 函数 ADC_GetSoftwareStartConvStatus

函数名	ADC_GetSoftwareStartConvStatus
函数原形	FlagStatus ADC_GetCalibrationStatus(ADC_TypeDef* ADCx)
功能描述	获取 ADC 软件转换启动状态
输入参数	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输出参数	无
返回值	ADC 软件转换启动的新状态（SET 或者 RESET）
先决条件	无
被调用函数	无

例：

```
/* Get the ADC1 conversion start bit */
FlagStatus Status;
Status = ADC_GetSoftwareStartConvStatus(ADC1);
```

4.2.13 函数ADC_DiscModeChannelCountConfig

Table 23. 描述了函数 ADC_DiscModeChannelCountConfig

Table 23. 函数 ADC_DiscModeChannelCountConfig

函数名	ADC_DiscModeChannelCountConfig
函数原形	void ADC_DiscModeChannelCountConfig(ADC_TypeDef* ADCx, u8 Number)
功能描述	对 ADC 规则组通道配置间断模式
输入参数 1	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输入参数 2	Number：间断模式规则组通道计数器的值。这个值得范围为 1 到 8。
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

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

4.2.14 函数ADC_DiscModeCmd

Table 24. 描述了函数 ADC_DiscModeCmd

Table 24. 函数 ADC_DiscModeCmd

函数名	ADC_DiscModeCmd
函数原形	void ADC_DiscModeCmd(ADC_TypeDef* ADCx, FunctionalState NewState)
功能描述	使能或者失能指定的 ADC 规则组通道的间断模式
输入参数 1	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输入参数 2	NewState：ADC 规则组通道上间断模式的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

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

4.2.15 函数ADC_RegularChannelConfig

Table 25. 描述了函数 ADC_RegularChannelConfig

Table 25. 函数 ADC_RegularChannelConfig

函数名	ADC_RegularChannelConfig
函数原形	void ADC_RegularChannelConfig(ADC_TypeDef* ADCx, u8 ADC_Channel, u8 Rank, u8 ADC_SampleTime)
功能描述	设置指定 ADC 的规则组通道，设置它们的转化顺序和采样时间
输入参数 1	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输入参数 2	ADC_Channel：被设置的 ADC 通道 参阅章节 ADC_Channel 查阅更多该参数允许取值范围
输入参数 3	Rank：规则组采样顺序。取值范围 1 到 16。
输入参数 4	ADC_SampleTime：指定 ADC 通道的采样时间值 参阅章节 ADC_SampleTime 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

ADC_Channel

参数 ADC_Channel 指定了通过调用函数 ADC_RegularChannelConfig 来设置的 ADC 通道。Table 26. 列举了 ADC_Channel 可取的值：

Table 26. ADC_Channel 值

ADC_Channel	描述
ADC_Channel_0	选择 ADC 通道 0
ADC_Channel_1	选择 ADC 通道 1
ADC_Channel_2	选择 ADC 通道 2
ADC_Channel_3	选择 ADC 通道 3
ADC_Channel_4	选择 ADC 通道 4
ADC_Channel_5	选择 ADC 通道 5

ADC_Channel_6	选择 ADC 通道 6
ADC_Channel_7	选择 ADC 通道 7
ADC_Channel_8	选择 ADC 通道 8
ADC_Channel_9	选择 ADC 通道 9
ADC_Channel_10	选择 ADC 通道 10
ADC_Channel_11	选择 ADC 通道 11
ADC_Channel_12	选择 ADC 通道 12
ADC_Channel_13	选择 ADC 通道 13
ADC_Channel_14	选择 ADC 通道 14
ADC_Channel_15	选择 ADC 通道 15
ADC_Channel_16	选择 ADC 通道 16
ADC_Channel_17	选择 ADC 通道 17

ADC_SampleTime

ADC_SampleTime 设定了选中通道的 ADC 采样时间。Table 27. 列举了 ADC_SampleTime 可取的值：

Table 27. ADC_SampleTime 值：

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

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_ExternalTrigConvConfig

Table 28. 描述了函数 ADC_ExternalTrigConvConfig

Table 28. 函数 ADC_ExternalTrigConvConfig

函数名	ADC_ExternalTrigConvConfig
函数原形	void ADC_ExternalTrigConvCmd(ADC_TypeDef* ADCx, FunctionalState NewState)
功能描述	使能或者失能 ADCx 的经外部触发启动转换功能
输入参数 1	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输入参数 2	NewState：指定 ADC 外部触发转换启动的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/*Enable the start of conversion for ADC1 through exteral trigger */
ADC_ExternalTrigConvCmd(ADC1, ENABLE);
```

4.2.17 函数ADC_GetConversionValue

Table 29. 描述了函数 ADC_GetConversionValue

Table 29. 函数 ADC_GetConversionValue

函数名	ADC_GetConversionValue
函数原形	u16 ADC_GetConversionValue(ADC_TypeDef* ADCx)
功能描述	返回最近一次 ADCx 规则组的转换结果
输入参数	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输出参数	无
返回值	转换结果
先决条件	无
被调用函数	无

例：

```
/*Returns the ADC1 Master data value of the last converted channel*/
u16 DataValue;
DataValue = ADC_GetConversionValue(ADC1);
```

4.2.18 函数ADC_GetDuelModeConversionValue

Table 30. 描述了函数 ADC_GetDuelModeConversionValue

Table 30. 函数 ADC_GetDuelModeConversionValue

函数名	ADC_GetDuelModeConversionValue
函数原形	u32 ADC_GetDualModeConversionValue()
功能描述	返回最近一次双 ADC 模式下的转换结果
输入参数	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输出参数	无
返回值	转换结果
先决条件	无
被调用函数	无

例：

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

4.2.19 函数ADC_AutoInjectedConvCmd

Table 31. 描述了函数 ADC_AutoInjectedConvCmd

Table 31. 函数 ADC_AutoInjectedConvCmd

函数名	ADC_AutoInjectedConvCmd
函数原形	void ADC_AutoInjectedConvCmd(ADC_TypeDef* ADCx, FunctionalState NewState)
功能描述	使能或者失能指定 ADC 在规则组转化后自动开始注入组转换
输入参数 1	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输入参数 2	NewState：指定 ADC 自动注入转化的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable the auto injected conversion for ADC2 */
ADC_AutoInjectedConvCmd(ADC2, ENABLE);
```

4.2.20 函数ADC_InjectedDiscModeCmd

Table 32. 描述了函数 ADC_InjectedDiscModeCmd

Table 32. 函数 ADC_InjectedDiscModeCmd

函数名	ADC_InjectedDiscModeCmd
函数原形	void ADC_InjectedDiscModeCmd(ADC_TypeDef* ADCx, FunctionalState NewState)
功能描述	使能或者失能指定 ADC 的注入组间断模式
输入参数 1	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输入参数 2	NewState：ADC 注入组通道上间断模式的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

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

4.2.21 函数ADC_ExternalTrigInjectedConvConfig

Table 33. 描述了函数 ADC_ExternalTrigInjectedConvConfig

Table 33. 函数 ADC_ExternalTrigInjectedConvConfig

函数名	ADC_ExternalTrigInjectedConvConfig
函数原形	void ADC_ExternalTrigInjectedConvConfig(ADC_TypeDef* ADCx, u32 ADC_ExternalTrigConv)
功能描述	配置 ADCx 的外部触发启动注入组转换功能
输入参数 1	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输入参数 2	ADC_ExternalTrigConv：启动注入转换的 ADC 触发 参阅章节 ADC_ExternalTrigInjectedConv 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

ADC_ExternalTrigInjectedConv

ADC_ExternalTrigInjectedConv 指定了所使用的注入转换启动触发。Table 34. 列举了 ADC_ExternalTrigInjectedConv 可取的值：

Table 34. ADC_ExternalTrigInjectedConv 值

ADC_ExternalTrigInjectedConv	描述
ADC_ExternalTrigInjecConv_T1_TRGO	选择定时器 1 的 TRGO 作为注入转换外部触发
ADC_ExternalTrigInjecConv_T1_CC4	选择定时器 1 的捕获比较 4 作为注入转换外部触发
ADC_ExternalTrigInjecConv_T2_TRGO	选择定时器 2 的 TRGO 作为注入转换外部触发
ADC_ExternalTrigInjecConv_T2_CC1	选择定时器 2 的捕获比较 1 作为注入转换外部触发
ADC_ExternalTrigInjecConv_T3_CC4	选择定时器 3 的捕获比较 4 作为注入转换外部触发
ADC_ExternalTrigInjecConv_T4_TRGO	选择定时器 4 的 TRGO 作为注入转换外部触发
ADC_ExternalTrigInjecConv_Ext_IT15	选择外部中断线 15 事件作为注入转换外部触发
ADC_ExternalTrigInjecConv_None	注入转换由软件而不是外部触发启动

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

Table 35. 描述了函数 ADC_ExternalTrigInjectedConvCmd

Table 35. 函数 ADC_ExternalTrigInjectedConvCmd

函数名	ADC_ExternalTrigInjectedConvCmd
函数原形	void ADC_ExternalTrigInjectedConvCmd(ADC_TypeDef* ADCx, FunctionalState NewState)
功能描述	使能或者失能 ADCx 的经外部触发启动注入组转换功能
输入参数 1	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输入参数 2	NewState：指定 ADC 外部触发启动注入转换的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：
```
/* Enable the start of injected conversion for ADC1 through exteral
trigger */
ADC_ExternalTrigInjectedConvCmd(ADC1, ENABLE);
```

4.2.23 函数ADC_SoftwareStartinjectedConvCmd

Table 36. 描述了函数 ADC_SoftwareStartinjectedConvCmd

Table 36. 函数 ADC_SoftwareStartinjectedConvCmd

函数名	ADC_SoftwareStartinjectedConvCmd
函数原形	void ADC_SoftwareStartInjectedConvCmd(ADC_TypeDef* ADCx, FunctionalState NewState)
功能描述	使能或者失能 ADCx 软件启动注入组转换功能
输入参数 1	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输入参数 2	NewState：指定 ADC 软件触发启动注入转换的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：
```
/* Start by software the ADC2 Conversion */
ADC_SoftwareStartInjectedConvCmd(ADC2, ENABLE);
```

4.2.24 函数ADC_GetsoftwareStartinjectedConvStatus

Table 37. 描述了函数 ADC_GetsoftwareStartinjectedConvStatus

Table 37. 函数 ADC_GetsoftwareStartinjectedConvStatus

函数名	ADC_GetsoftwareStartinjectedConvStatus
函数原形	FlagStatus ADC_GetSoftwareStartInjectedConvStatus(ADC_TypeDef* ADCx)
功能描述	获取指定 ADC 的软件启动注入组转换状态
输入参数	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输出参数	无
返回值	ADC 软件触发启动注入转换的新状态
先决条件	无
被调用函数	无

例：

```
/* Get the ADC1 injected conversion start bit */
FlagStatus Status;
Status = ADC_GetSoftwareStartInjectedConvStatus(ADC1);
```

4.2.25 函数ADC_InjectedChannleConfig

Table 38. 描述了函数 ADC_InjectedChannleConfig

Table 38. 函数 ADC_InjectedChannleConfig

函数名	ADC_InjectedChannleConfig
函数原形	void ADC_InjectedChannelConfig(ADC_TypeDef* ADCx, u8 ADC_Channel, u8 Rank, u8 ADC_SampleTime)
功能描述	设置指定 ADC 的注入组通道，设置它们的转化顺序和采样时间
输入参数 1	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输入参数 2	ADC_Channel：被设置的 ADC 通道 参阅章节 ADC_Channel 查阅更多该参数允许取值范围
输入参数 3	Rank：规则组采样顺序。取值范围 1 到 4。
输入参数 4	ADC_SampleTime：指定 ADC 通道的采样时间值 参阅章节 ADC_SampleTime 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	之前必须调用函数 ADC_InjectedSequencerLengthConfig 来确定注入转换通道的数目。特别是在通道数目小于 4 的情况下，来正确配置每个注入通道的转化顺序。
被调用函数	无

ADC_Channel

参数 ADC_Channel 指定了需设置的 ADC 通道。Table 26. 列举了 ADC_Channel 可取的值。

ADC_SampleTime

ADC_SampleTime 设定了选中通道的 ADC 采样时间。Table 27. 列举了 ADC_SampleTime 可取的值。

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

Table 39. 描述了函数 ADC_InjectedSequencerLengthConfig

Table 39. 函数 ADC_InjectedSequencerLengthConfig

函数名	ADC_InjectedSequencerLengthConfig
函数原形	void ADC_InjectedSequencerLengthConfig(ADC_TypeDef* ADCx, u8 Length)
功能描述	设置注入组通道的转换序列长度
输入参数 1	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输入参数 2	Length：序列长度 这个参数取值范围 1 到 4。
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Set the ADC1 Sequencer length to 4 channels */
ADC_InjectedSequencerLengthConfig(ADC1, 4);
```

4.2.27 函数ADC_SetinjectedOffset

Table 40. 描述了函数 ADC_SetinjectedOffset

Table 40. 函数 ADC_SetinjectedOffset

函数名	ADC_SetinjectedOffset
函数原形	void ADC_SetInjectedOffset(ADC_TypeDef* ADCx, u8 ADC_InjectedChannel, u16 Offset)
功能描述	设置注入组通道的转换偏移值
输入参数 1	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输入参数 2	ADC_InjectedChannel：被设置转换偏移值的 ADC 注入通道 参阅章节 ADC_InjectedChannel 查阅更多该参数允许取值范围
输入参数 3	Offset：ADC 注入通道的转换偏移值 这个值是一个 12 位值。
输出参数	无
返回值	无
先决条件	无
被调用函数	无

ADC_InjectedChannel

参数 ADC_InjectedChannel 指定了必须设置转换偏移值的 ADC 通道。Table 41. 列举了 ADC_InjectedChannel 可取的值。Table 41. ADC_InjectedChannel 值

ADC_InjectedChannel	描述
ADC_InjectedChannel_1	选择注入通道 1
ADC_InjectedChannel_2	选择注入通道 2
ADC_InjectedChannel_3	选择注入通道 3
ADC_InjectedChannel_4	选择注入通道 4

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

Table 42. 描述了函数 ADC_GetInjectedConversionValue

Table 42. 函数 ADC_GetInjectedConversionValue

函数名	ADC_GetInjectedConversionValue
函数原形	u16 ADC_GetInjectedConversionValue(ADC_TypeDef* ADCx, u8 ADC_InjectedChannel)
功能描述	返回 ADC 指定注入通道的转换结果
输入参数 1	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输入参数 2	ADC_InjectedChannel：被转换的 ADC 注入通道 参阅章节 ADC_InjectedChannel 查阅更多该参数允许取值范围
输出参数	无
返回值	转换结果
先决条件	无
被调用函数	无

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

Table 43. 描述了函数 ADC_AnalogWatchdogCmd

Table 43. 函数 ADC_AnalogWatchdogCmd

函数名	ADC_AnalogWatchdogCmd
函数原形	void ADC_AnalogWatchdogCmd(ADC_TypeDef* ADCx, u32 ADC_AnalogWatchdog)
功能描述	使能或者失能指定单个/全体，规则/注入组通道上的模拟看门狗
输入参数 1	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输入参数 2	ADC_AnalogWatchdog：ADC 模拟看门狗设置 参阅章节 ADC_AnalogWatchdog 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

ADC_AnalogWatchdog

ADC_AnalogWatchdog 规定了 ADC 模拟看门狗的设置。Table 44.列举了 ADC_AnalogWatchdog 可取的值：

Table 44. ADC_AnalogWatchdog 值

ADC_AnalogWatchdog	描述
ADC_AnalogWatchdog_SingleRegEnable	单个规则通道上设置模拟看门狗
ADC_AnalogWatchdog_SingleInjecEnable	单个注入通道上设置模拟看门狗
ADC_AnalogWatchdog_SingleRegorInjecEnable	单个规则通道或者注入通道上设置模拟看门狗
ADC_AnalogWatchdog_AllRegEnable	所有规则通道上设置模拟看门狗
ADC_AnalogWatchdog_AllInjecEnable	所有注入通道上设置模拟看门狗
ADC_AnalogWatchdog_AllRegAllInjecEnable	所有规则通道和所有注入通道上上设置模拟看门狗
ADC_AnalogWatchdog_None	不设置模拟看门狗

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_AnalogWatchdongThresholdsConfig

Table 45. 描述了函数 ADC_AnalogWatchdongThresholdsConfig

Table 45. 函数 ADC_AnalogWatchdongThresholdsConfig

函数名	ADC_AnalogWatchdongThresholdsConfig
函数原形	void ADC_AnalogWatchdogThresholdsConfig(ADC_TypeDef* ADCx, u16 HighThreshold, u16 LowThreshold)
功能描述	设置模拟看门狗的高/低阈值
输入参数 1	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输入参数 2	HignThreshold：模拟看门狗的高阈值 这个参数是一个 12 位值。
输入参数 3	LowThreshold：模拟看门狗的低阈值 这个参数是一个 12 位值。
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

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

4.2.31 函数ADC_AnalogWatchdongSingleChannelConfig

Table 46. 描述了函数 ADC_AnalogWatchdongSingleChannelConfig

Table 46. 函数 ADC_AnalogWatchdongSingleChannelConfig

函数名	ADC_AnalogWatchdongSingleChannelConfig
函数原形	void ADC_AnalogWatchdogSingleChannelConfig(ADC_TypeDef* ADCx, u8 ADC_Channel)
功能描述	对单个 ADC 通道设置模拟看门狗
输入参数 1	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输入参数 2	ADC_Channel：被设置模拟看门狗的 ADC 通道 参阅章节 ADC_Channel 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

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

4.2.32 函数ADC_TampSensorVrefintCmd

Table 47. 描述了函数 ADC_TampSensorVrefintCmd

Table 47. 函数 ADC_TampSensorVrefintCmd

函数名	ADC_TampSensorVrefintCmd
函数原形	void ADC_TempSensorVrefintCmd(FunctionalState NewState)
功能描述	使能或者失能温度传感器和内部参考电压通道
输入参数	NewState：温度传感器和内部参考电压通道的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

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

4.2.33 函数ADC_GetFlagStatus

Table 48. 描述了函数 ADC_GetFlagStatus

Table 48. 函数 ADC_GetFlagStatus

函数名	ADC_GetFlagStatus
函数原形	FlagStatus ADC_GetFlagStatus(ADC_TypeDef* ADCx, u8 ADC_FLAG)
功能描述	检查制定 ADC 标志位置 1 与否
输入参数 1	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输入参数 2	ADC_FLAG：指定需检查的标志位 参阅章节 ADC_FLAG 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

ADC_FLAG

Table 49. 给出了 ADC_FLAG 的值

Table 49. ADC_FLAG 的值

ADC_AnalogWatchdog	描述
ADC_FLAG_AWD	模拟看门狗标志位
ADC_FLAG_EOC	转换结束标志位
ADC_FLAG_JEOC	注入组转换结束标志位
ADC_FLAG_JSTRT	注入组转换开始标志位
ADC_FLAG_STRT	规则组转换开始标志位

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

Table 50. 描述了函数 ADC_ClearFlag

Table 50. 函数 ADC_ClearFlag

函数名	ADC_ClearFlag
函数原形	void ADC_ClearFlag(ADC_TypeDef* ADCx, u8 ADC_FLAG)
功能描述	清除 ADCx 的待处理标志位
输入参数 1	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输入参数 2	ADC_FLAG：待处理的标志位，使用操作符“|”可以同时清除 1 个以上的标志位 参阅章节 ADC_FLAG 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear the ADC2 STRT pending flag */
ADC_ClearFlag(ADC2, ADC_FLAG_STRT);
```

4.2.35 函数ADC_GetITStatus

Table 51. 描述了函数 ADC_GetITStatus

Table 51. 函数 ADC_GetITStatus

函数名	ADC_GetITStatus
函数原形	ITStatus ADC_GetITStatus(ADC_TypeDef* ADCx, u16 ADC_IT)
功能描述	检查指定的 ADC 中断是否发生
输入参数 1	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输入参数 2	ADC_IT：将要被检查指定 ADC 中断源 参阅章节 ADC_IT 获得该参数可取值得更多细节
输出参数	无
返回值	无
先决条件	无
被调用函数	无

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

Table 52. 描述了函数 ADC_ClearITPendingBit

Table 52. 函数 ADC_ClearITPendingBit

函数名	ADC_ClearITPendingBit
函数原形	void ADC_ClearITPendingBit(ADC_TypeDef* ADCx, u16 ADC_IT)
功能描述	清除 ADCx 的中断待处理位
输入参数 1	ADCx：x 可以是 1 或者 2 来选择 ADC 外设 ADC1 或 ADC2
输入参数 2	ADC_IT：带清除的 ADC 中断待处理位 参阅章节 ADC_IT 获得该参数可取值得更多细节
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

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

5. 备份寄存器（BKP）

备份寄存器由 10 个 16 位寄存器组成，可用来存储 20 个字节的用户应用程序数据。他们处在备份域里，当 VDD 电源被切断，他们仍然由 VBAT 维持供电。当系统在待机模式下被唤醒，或系统复位或电源复位时，他们也不会被复位。

此外，BKP 控制寄存器用来管理侵入检测和 RTC 校准功能。

Section 5.1 BKP 寄存器结构描述了 BKP 固件函数库所使用的数据结构，Section 5.2 固件库函数介绍了函数库里的所有函数。

5.1 BKP寄存器结构

BKP 寄存器结构，***BKP_TypeDef***，在文件***"stm32f10x_map.h"***中定义如下：

```
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;
```

Table 53. 给出了 BKP 的寄存器列表：

Table 53. BKP 寄存器

寄存器	描述
DR 1-10	数据后备寄存器 1 到 10
RTCCR	RTC 时钟校准寄存器
CR	后备控制寄存器
CSR	后备控制状态寄存器

BKP 外设同时在文件***"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 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
```

使用 Debug 模式时，初始化 ***BKP*** 指针与文件***"stm32f10x_lib.c"***：

```
#ifdef _BKP
BKP = (BKP_TypeDef *) BKP_BASE;
#endif /*_BKP */
```

为了访问 BKP 寄存器，,***_BKP*** 必须在文件***"stm32f10x_conf.h"***中定义如下：

```
#define _BKP
```

5.2 BKP库函数

Table 54. 例举了 BKP 的库函数

Table 54. BKP 库函数

函数名	描述
BKP_DeInit	将外设 BKP 的全部寄存器重设为缺省值
BKP_TamperPinLevelConfig	设置侵入检测管脚的有效电平
BKP_TamperPinCmd	使能或者失能管脚的侵入检测功能
BKP_ITConfig	使能或者失能侵入检测中断
BKP_RTCOutputConfig	选择在侵入检测管脚上输出的 RTC 时钟源
BKP_SetRTCCalibrationValue	设置 RTC 时钟校准值
BKP_WriteBackupRegister	向指定的后备寄存器中写入用户程序数据
BKP_ReadBackupRegister	从指定的后备寄存器中读出数据
BKP_GetFlagStatus	检查侵入检测管脚事件的标志位被设置与否
BKP_ClearFlag	清除侵入检测管脚事件的待处理标志位
BKP_GetITStatus	检查侵入检测中断发生与否
BKP_ClearITPendingBit	清除侵侵入检测中断的待处理位

5.2.1 函数BKP_DeInit

Table 55. 描述了函数 BKP_DeInit

Table 55. 函数 BKP_DeInit

函数名	BKP_DeInit
函数原形	void BKP_DeInit(void)
功能描述	将外设 BKP 的全部寄存器重设为缺省值
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	RCC_BackupResetCmd

例：

```
/* Reset the BKP registers */
BKP_DeInit();
```

5.2.2 函数BKP_TamperPinLevelConfig

Table 56. 描述了函数 BKP_TamperPinLevelConfig

Table 56. 函数 BKP_TamperPinLevelConfig

函数名	BKP_TamperPinLevelConfig
函数原形	void BKP_TamperPinLevelConfig(u16 BKP_TamperPinLevel)
功能描述	设置侵入检测管脚的有效电平
输入参数	BKP_TamperPinLevel：侵入检测管脚的有效电平 参阅 Section：BKP_TamperPinLevel 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

BKP_TamperPinLevel

参数 BKP_TamperPinLevel 指定了侵入检测管脚的有效电平。它可以取下表中值：

Table 57. BKP_TamperPinLevel 值

BKP_TamperPinLevel	描述
BKP_TamperPinLevel_High	侵入检测管脚高电平有效
BKP_TamperPinLevel_Low	侵入检测管脚低电平有效

例：

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

5.2.3 函数BKP_TamperPinCmd

Table 58. 描述了函数 BKP_TamperPinCmd

Table 58. 函数 BKP_TamperPinCmd

函数名	BKP_TamperPinCmd
函数原形	void BKP_TamperPinCmd(FunctionalState NewState)
功能描述	使能或者失能管脚的侵入检测功能
输入参数	NewState：侵入检测功能的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable Tamper Pin functionality */
BKP_TamperPinCmd(ENABLE);
```

5.2.4 函数BKP_ITConfig

Table 59. 描述了函数 BKP_ITConfig

Table 59. 函数 BKP_ITConfig

函数名	BKP_ITConfig
函数原形	void BKP_ITConfig(FunctionalState NewState)
功能描述	使能或者失能侵入检测中断
输入参数	NewState：侵入检测中断的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

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

5.2.5 函数BKP_RTCOutputConfig

Table 60. 描述了函数 BKP_RTCOutputConfig

Table 60. 函数 BKP_RTCOutputConfig

函数名	BKP_RTCOutputConfig
函数原形	void BKP_RTCOutputConfig(u16 BKP_RTCOutputSource)
功能描述	选择在侵入检测管脚上输出的 RTC 时钟源
输入参数	BKP_RTCOutputSource：指定的 RTC 输出源 参阅 Section：BKP_RTCOutputSource 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	调用该函数前必须失能管脚的侵入检测功能
被调用函数	无

BKP_RTCOutputSource

参数 BKP_RTCOutputSource 用来选择 RTC 输出时钟源，它可以取下表中值：

Table 61. BKP_RTCOutputSource 值

BKP_RTCOutputSource	描述
BKP_RTCOutputSource_None	侵入检测管脚上无 RTC 输出
BKP_RTCOutputSource_CalibClock	侵入检测管脚上输出，其时钟频率为 RTC 时钟除以 64
BKP_RTCOutputSource_Alarm	侵入检测管脚上输出 RTC 闹钟脉冲
BKP_RTCOutputSource_Second	侵入检测管脚上输出 RTC 秒脉冲

例：

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

5.2.6 函数BKP_SetRTCCalibrationValue

Table 62. 描述了函数 BKP_SetRTCCalibrationValue

Table 62. 函数 BKP_SetRTCCalibrationValue

函数名	BKP_SetRTCCalibrationValue
函数原形	void BKP_SetRTCCalibrationValue(u8 CalibrationValue)
功能描述	设置 RTC 时钟校准值
输入参数	CalibrationValue：RTC 时钟校准值 该参数允许取值范围为 0 到 0x7F
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

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

5.2.7 函数BKP_WriteBackupRegister

Table 63. 描述了函数 BKP_WriteBackupRegister

Table 63. 函数 BKP_WriteBackupRegister

函数名	BKP_WriteBackupRegister
函数原形	void BKP_WriteBackupRegister(u16 BKP_DR, u16 Data)
功能描述	向指定的后备寄存器中写入用户程序数据
输入参数 1	BKP_DR：数据后备寄存器 参阅 Section：BKP_DR 查阅更多该参数允许取值范围
输入参数 2	Data：待写入的数据
输出参数	无
返回值	无
先决条件	无
被调用函数	无

BKP_DR

参数 BKP_DR 用来选择数据后备寄存器，Table 64. 例举了该参数可取的值：

Table 64. BKP_DR 值

BKP_DR	描述
BKP_DR1	选中数据寄存器 1
BKP_DR2	选中数据寄存器 2
BKP_DR3	选中数据寄存器 3
BKP_DR4	选中数据寄存器 4
BKP_DR5	选中数据寄存器 5
BKP_DR6	选中数据寄存器 6
BKP_DR7	选中数据寄存器 7
BKP_DR8	选中数据寄存器 8
BKP_DR9	选中数据寄存器 9
BKP_DR10	选中数据寄存器 10

例：

```
/* Write 0xA587 to Data Backup Register1 */
BKP_WriteBackupRegister(BKP_DR1, 0xA587);
```

5.2.8 函数BKP_ReadBackupRegister

Table 65. 描述了函数 BKP_ReadBackupRegister

Table 65. 函数 BKP_ReadBackupRegister

函数名	BKP_ReadBackupRegister
函数原形	u16 BKP_ReadBackupRegister(u16 BKP_DR)
功能描述	从指定的后备寄存器中读出数据
输入参数	BKP_DR：数据后备寄存器 参阅 Section：BKP_DR 查阅更多该参数允许取值范围
输出参数	无
返回值	指定的后备寄存器中的数据
先决条件	无
被调用函数	无

例：

```
/* Read Data Backup Register1 */
```

```
u16 Data;
Data = BKP_ReadBackupRegister(BKP_DR1);
```

5.2.9 函数BKP_GetFlagStatus

Table 66. 描述了函数 BKP_GetFlagStatus

Table 66. 函数 BKP_GetFlagStatus

函数名	BKP_GetFlagStatus
函数原形	FlagStatus BKP_GetFlagStatus(void)
功能描述	检查侵入检测管脚事件的标志位被设置与否
输入参数	无
输出参数	无
返回值	检查侵入检测管脚事件的标志位的新状态（SET 或者 RESET）
先决条件	无
被调用函数	无

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

Table 67. 描述了函数 BKP_ClearFlag

Table 67. 函数 BKP_ClearFlag

函数名	BKP_ClearFlag
函数原形	void BKP_ClearFlag(void)
功能描述	清除侵入检测管脚事件的待处理标志位
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear Tamper Pin Event pending flag */
BKP_ClearFlag();
```

5.2.11 函数BKP_GetITStatus

Table 68. 描述了函数 BKP_GetITStatus

Table 68. 函数 BKP_GetITStatus

函数名	BKP_GetITStatus
函数原形	ITStatus BKP_GetITStatus(void)
功能描述	检查侵入检测中断发生与否
输入参数	无
输出参数	无
返回值	检查检查侵入检测中断标志位的新状态（SET 或者 RESET）
先决条件	无
被调用函数	无

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

Table 69. 描述了函数 BKP_ClearITPendingBit

Table 69. 函数 BKP_ClearITPendingBit

函数名	BKP_ClearITPendingBit
函数原形	void BKP_ClearITPendingBit(void)
功能描述	清除侵侵入检测中断的待处理位
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear Tamper Pin interrupt pending bit */
BKP_ClearITPendingBit();
```

6 控制器局域网（CAN）

本外设作为 CAN 网络的界面，支持 CAN 协议 2.0A 和 2.0B。它的设计目标是，以最小的 CPU 负荷来高效处理大量收到的报文。它也支持报文发送的优先级要求（优先级特性可软件配置）。

Section 6.1 描述了 CAN 固件函数库所使用的数据结构，Section 6.2 固件库函数介绍了函数库里的所有函数。

6.1 CAN寄存器结构

CAN 寄存器结构，***CAN_TypeDef***，在文件***"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;
```

Table 70. 给出了 CAN 的寄存器列表：

Table 70. CAN 寄存器

寄存器	描述
CAN_MCR	CAN 主控制寄存器
CAN_MSR	CAN 主状态寄存器
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 中断允许寄存器
CAN_ESR	CAN 错误状态寄存器
CAN_BTR	CAN 位时间特性寄存器
TIR	发送邮箱标识符寄存器
TDTR	发送邮箱数据长度和时间戳寄存器
TDLR	发送邮箱低字节数据寄存器
TDHR	发送邮箱高字节数据寄存器
RIR	接收 FIFO 邮箱标识符寄存器
RDTR	接收 FIFO 邮箱数据长度和时间戳寄存器
RDLR	接收 FIFO 邮箱低字节数据寄存器
RDHR	接收 FIFO 邮箱高字节数据寄存器
CAN_FMR	CAN 过滤器主控寄存器
CAN_FM0R	CAN 过滤器模式寄存器
CAN_FSC0R	CAN 过滤器位宽寄存器
CAN_FFA0R	CAN 过滤器 FIFO 关联寄存器
CAN_FA0R	CAN 过滤器激活寄存器
CAN_FR0	过滤器组 0 寄存器
CAN_FR1	过滤器组 1 寄存器

CAN 外设同时在文件***"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 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
```

使用 Debug 模式时，初始化 ***CAN*** 指针于文件***"stm32f10x_lib.c"***：

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

为了访问 CAN 寄存器，,***_CAN*** 必须在文件***"stm32f10x_conf.h"***中定义如下：

```
#define _CAN
```

6.2 CAN库函数

Table 71. 例举了 CAN 的库函数

Table 71. CAN 库函数

函数名	描述
CAN_DeInit	将外设 CAN 的全部寄存器重设为缺省值
CAN_Init	根据 CAN_InitStruct 中指定的参数初始化外设 CAN 的寄存器
CAN_FilterInit	根据 CAN_FilterInitStruct 中指定的参数初始化外设 CAN 的寄存器
CAN_StructInit	把 CAN_InitStruct 中的每一个参数按缺省值填入
CAN_ITConfig	使能或者失能指定的 CAN 中断
CAN_Transmit	开始一个消息的传输
CAN_TransmitStatus	检查消息传输的状态
CAN_CancelTransmit	取消一个传输请求
CAN_FIFORelease	释放一个 FIFO
CAN_MessagePending	返回挂号的信息数量
CAN_Receive	接收一个消息
CAN_Sleep	使 CAN 进入低功耗模式
CAN_WakeUp	将 CAN 唤醒
CAN_GetFlagStatus	检查指定的 CAN 标志位被设置与否
CAN_ClearFlag	清除 CAN 的待处理标志位
CAN_GetITStatus	检查指定的 CAN 中断发生与否
CAN_ClearITPendingBit	清除 CAN 的中断待处理标志位

6.2.1 函数CAN_DeInit

Table 72. 描述了函数 CAN_DeInit

Table 72. 函数 CAN_DeInit

函数名	CAN_DeInit
函数原形	void CAN_DeInit(void)
功能描述	将外设 CAN 的全部寄存器重设为缺省值
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	RCC_APB1PeriphResetCmd()

例：

```
/* Deinitialize the CAN */
CAN_DeInit();
```

6.2.2 函数CAN_Init

Table 73. 描述了函数 CAN_Init

Table 73. 函数 CAN_Init

函数名	CAN_Init
函数原形	u8 CAN_Init(CAN_InitTypeDef* CAN_InitStruct)
功能描述	根据 CAN_InitStruct 中指定的参数初始化外设 CAN 的寄存器
输入参数	CAN_InitStruct：指向结构 CAN_InitTypeDef 的指针，包含了指定外设 CAN 的配置信息 参阅：Section：CAN_InitTypeDef 查阅更多该参数允许取值范围
输出参数	无
返回值	指示 CAN 初始化成功的常数 CANINITFAILED = 初始化失败 CANINITOK = 初始化成功
先决条件	无
被调用函数	无

CAN_InitTypeDef structure

CAN_InitTypeDef 定义于文件“stm32f10x_can.h”：

```
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 用来使能或者失能时间触发通讯模式，可以设置这个参数的值为 ENABLE 或者 DISABLE。

CAN_ABOM

CAN_ABOM 用来使能或者失能自动离线管理，可以设置这个参数的值为 ENABLE 或者 DISABLE。

CAN_AWUM

CAN_AWUM 用来使能或者失能自动唤醒模式，可以设置这个参数的值为 ENABLE 或者 DISABLE。

CAN_NART

CAN_NARM 用来使能或者失能非自动重传输模式，可以设置这个参数的值为 ENABLE 或者 DISABLE。

CAN_RFLM

CAN_RFLM 用来使能或者失能接收 FIFO 锁定模式，可以设置这个参数的值为 ENABLE 或者 DISABLE。

CAN_TXFP

CAN_TXFP 用来使能或者失能发送 FIFO 优先级，可以设置这个参数的值为 ENABLE 或者 DISABLE。

CAN_Mode

CAN_Mode 设置了 CAN 的工作模式，Table 74.给出了该参数可取的值

Table 74. CAN_Mode 值

CAN_Mode	描述
CAN_Mode_Normal	CAN 硬件工作在正常模式
CAN_Mode_Silent	CAN 硬件工作在静默模式
CAN_Mode_LoopBack	CAN 硬件工作在环回模式
CAN_Mode_Silent_LoopBack	CAN 硬件工作在静默环回模式

CAN_SJW

CAN_SJW 定义了重新同步跳跃宽度(SJW)，即在每位中可以延长或缩短多少个时间单位的上限，Table 75. 给出了该参数可取的值

Table 75. CAN_SJW 值

CAN_SJW	描述
CAN_SJW_1tq	重新同步跳跃宽度 1 个时间单位
CAN_SJW_2tq	重新同步跳跃宽度 2 个时间单位
CAN_SJW_3tq	重新同步跳跃宽度 3 个时间单位
CAN_SJW_4tq	重新同步跳跃宽度 4 个时间单位

CAN_BS1

CAN_BS1 设定了时间段 1 的时间单位数目，Table 76.给出了该参数可取的值

Table 76. CAN_BS1 值

CAN_BS1	描述
CAN_BS1_1tq	时间段 1 为 1 个时间单位
…	…
CAN_BS1_16tq	时间段 1 为 16 个时间单位

CAN_BS2

CAN_BS2 设定了时间段 1 的时间单位数目，Table 77.给出了该参数可取的值

Table 77. CAN_BS1 值

CAN_BS1	描述
CAN_BS2_1tq	时间段 2 为 1 个时间单位
…	…
CAN_BS2_8tq	时间段 2 为 8 个时间单位

CAN_Prescaler

CAN_Prescaler 设定了一个时间单位的长度，它的范围是 1 到 1024。

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

Table 78. 描述了函数 CAN_FilterInit

Table 78. 函数 CAN_FilterInit

函数名	CAN_DeInit
函数原形	void CAN_FilterInit(CAN_FilterInitTypeDef* CAN_FilterInitStruct)
功能描述	根据 CAN_FilterInitStruct 中指定的参数初始化外设 CAN 的寄存器
输入参数	CAN_FilterInitStruct：指向结构 CAN_FilterInitTypeDef 的指针，包含了相关配置信息 参阅：Section：CAN_FilterInitTypeDef 结构查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

CAN_FilterInitTypeDef structure

CAN_FilterInitTypeDef 定义于文件“stm32f10x_can.h”：

```
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 指定了待初始化的过滤器，它的范围是 1 到 13。

CAN_FilterMode

CAN_FilterMode 指定了过滤器将被初始化到的模式。Table 79. 给出了该参数可取的值

Table 79. CAN_FilterMode 值

CAN_FilterMode	描述
CAN_FilterMode_IdMask	标识符屏蔽位模式
CAN_FilterMode_IdList	标识符列表模式

CAN_FilterScale

CAN_FilterScale 给出了过滤器位宽，Table 80. 给出了该参数可取的值

Table 80. CAN_FilterScale 值

CAN_FilterScale	描述
CAN_FilterScale_Two16bit	2 个 16 位过滤器
CAN_FilterScale_One32bit	1 个 32 位过滤器

CAN_FilterIdHigh

CAN_FilterIdHigh 用来设定过滤器标识符（32 位位宽时为其高段位，16 位位宽时为第一个）。它的范围是 0x0000 到 0xFFFF。

CAN_FilterIdLow

CAN_FilterIdHigh 用来设定过滤器标识符（32 位位宽时为其低段位，16 位位宽时为第二个）。它的范围是 0x0000 到 0xFFFF。

CAN_FilterMaskIdHigh

CAN_FilterMaskIdHigh 用来设定过滤器屏蔽标识符或者过滤器标识符（32 位位宽时为其高段位，16 位位

宽时为第一个）。它的范围是 0x0000 到 0xFFFF。

CAN_FilterMaskIdLow

CAN_FilterMaskIdLow 用来设定过滤器屏蔽标识符或者过滤器标识符（32 位位宽时为其低段位，16 位位宽时为第二个）。它的范围是 0x0000 到 0xFFFF。

CAN_FilterFIFO

CAN_FilterFIFO 设定了指向过滤器的 FIFO（0 或 1），Table 81. 给出了该参数可取的值

Table 81. CAN_FilterFIFO 值

CAN_FilterFIFO	描述
CAN_FilterFIFO0	过滤器 FIFO0 指向过滤器 x
CAN_FilterFIFO1	过滤器 FIFO1 指向过滤器 x

CAN_FilterActivation

CAN_FilterActivation 使能或者失能过滤器。该参数可取的值为 ENABLE 或者 DISABLE。

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

Table 82. 描述了函数 CAN_StructInit

Table 82. 函数 CAN_StructInit

函数名	CAN_StructInit
函数原形	void CAN_StructInit(CAN_InitTypeDef* CAN_InitStruct)
功能描述	把 CAN_InitStruct 中的每一个参数按缺省值填入
输入参数	CAN_InitStruct：指向待初始化结构 CAN_InitTypeDef 的指针 参阅 Table 83. 查阅该结构成员缺省值
输出参数	无
返回值	无
先决条件	无
被调用函数	无

Table 83. CAN_InitStruct 结构缺省值

成员	缺省值
CAN_TTCM	DISABLE
CAN_ABOM	DISABLE
CAN_AWUM	DISABLE
CAN_NART	DISABLE
CAN_RFLM	DISABLE
CAN_TXFP	DISABLE
CAN_Mode	CAN_Mode_Normal
CAN_SJW	CAN_SJW_1tq
CAN_BS1	CAN_BS1_4tq
CAN_BS2	CAN_BS2_3tq
CAN_Prescaler	1

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

6.2.5 函数CAN_ITConfig

Table 84. 描述了函数 CAN_ITConfig

Table 84. 函数 CAN_ITConfig

函数名	CAN_ITConfig
函数原形	void CAN_ITConfig(u32 CAN_IT, FunctionalState NewState)
功能描述	使能或者失能指定的 CAN 中断
输入参数 1	CAN_IT：待使能或者失能的 CAN 中断 参阅 Section：CAN_IT 查阅更多该参数允许取值范围
输入参数 2	NewState：CAN 中断的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

CAN_IT

输入参数 CAN_IT 为待使能或者失能的 CAN 中断。可以使用下表中的一个参数，或者他们的组合。

Table 85. CAN_IT 值

CAN_IT	描述
CAN_IT_TME	发送邮箱空中断屏蔽
CAN_IT_FMP0	FIFO0 消息挂号中断屏蔽
CAN_IT_FF0	FIFO0 满中断屏蔽
CAN_IT_FOV0	FIFO0 溢出中断屏蔽
AN_IT_FMP1	FIFO1 消息挂号中断屏蔽
CAN_IT_FF1	FIFO1 满中断屏蔽
CAN_IT_FOV1	FIFO1 溢出中断屏蔽
CAN_IT_EWG	错误警告中断屏蔽
CAN_IT_EPV	错误被动中断屏蔽
CAN_IT_BOF	离线中断屏蔽
CAN_IT_LEC	上次错误号中断屏蔽
CAN_IT_ERR	错误中断屏蔽
CAN_IT_WKU	唤醒中断屏蔽
CAN_IT_SLK	睡眠标志位中断屏蔽

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

6.2.6 函数CAN_Transmit

Table 86. 描述了函数 CAN_Transmit

Table 86. 函数 CAN_Transmit

函数名	CAN_Transmit
函数原形	u8 CAN_Transmit(CanTxMsg* TxMessage)
功能描述	开始一个消息的传输
输入参数	TxMessage：指向某结构的指针，该结构包含 CAN id，CAN DLC 和 CAN data
输出参数	无
返回值	所使用邮箱的号码，如果没有空邮箱返回 CAN_NO_MB
先决条件	无
被调用函数	无

CanTxMsg

结构 CanTxMsg 定义于文件“stm32f10x_can.h”：

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

StdId

StdId 用来设定标准标识符。它的取值范围为 0 到 0x7FF

ExtId

ExtId 用来设定扩展标识符。它的取值范围为 0 到 0x3FFFF

IDE

IDE 用来设定消息标识符的类型，Table 87. 给出了该参数可取的值

Table 87. IDE 值

IDE	描述
CAN_ID_STD	使用标准标识符
CAN_ID_EXT	使用标准标识符 + 扩展标识符

RTR

RTR 用来设定待传输消息的帧类型。它可以设置为数据帧或者远程帧。

Table 88. RTR 值

RTR	描述
CAN_RTR_DATA	数据帧
CAN_RTR_REMOTE	远程帧

DLC

DLC 用来设定待传输消息的帧长度。它的取值范围是 0 到 0x8。

Data[8]

Data[8]包含了待传输数据，它的取值范围为 0 到 0xFF。

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

Table 89. 描述了函数 CAN_TransmitStatus

Table 89. 函数 CAN_TransmitStatus

函数名	CAN_TransmitStatus
函数原形	u8 CAN_TransmitStatus(u8 TransmitMailbox)
功能描述	检查消息传输的状态
输入参数	TransmitMailbox：用来传输的邮箱号码
输出参数	无
返回值	CANTXOK CAN 驱动是否在传输数据 CANTXPENDING 消息是否挂号 CANTXFAILED 其他
先决条件	传输进行中
被调用函数	无

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

Table 90. 描述了函数 CAN_CancelTransmit

Table 90. 函数 CAN_CancelTransmit

函数名	CAN_CancelTransmit
函数原形	void CAN_CancelTransmit(u8 Mailbox)
功能描述	取消一个传输请求
输入参数	邮箱号码
输出参数	无
返回值	无
先决条件	传输挂号于某邮箱
被调用函数	无

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

Table 91. 描述了函数 CAN_FIFORelease

Table 91. 函数 CAN_FIFORelease

函数名	CAN_FIFORelease
函数原形	void CAN_FIFORelease(u8 FIFONumber)
功能描述	释放一个 FIFO
输入参数	FIFO number：接收 FIFO，CANFIFO0 或者 CANFIFO0
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Release FIFO 0*/
CAN_FIFORelease(CANFIFO0);
```

6.2.10 函数CAN_MessagePending

Table 92. 描述了函数 CAN_MessagePending

Table 92. 函数 CAN_MessagePending

函数名	CAN_MessagePending
函数原形	u8 CAN_MessagePending(u8 FIFONumber)
功能描述	返回挂号的信息数量
输入参数	FIFO number：接收 FIFO，CANFIFO0 或者 CANFIFO0
输出参数	无
返回值	NbMessage 为挂号的信息数量
先决条件	无
被调用函数	无

例：

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

6.2.11 函数CAN_Receive

Table 93. 描述了函数 CAN_Receive

Table 93. 函数 CAN_Receive

函数名	CAN_Receive
函数原形	void CAN_Receive(u8 FIFONumber, CanRxMsg* RxMessage)
功能描述	接收一个消息
输入参数	FIFO number：接收 FIFO，CANFIFO0 或者 CANFIFO0
输出参数	RxMessage：指向某结构的指针，该结构包含 CAN id，CAN DLC 和 CAN data。
返回值	无
先决条件	无
被调用函数	无

CanRxMsg

结构 CanRxMsg 定义于文件“stm32f10x_can.h”：

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

StdId

StdId 用来设定标准标识符。它的取值范围为 0 到 0x7FF

ExtId

ExtId 用来设定扩展标识符。它的取值范围为 0 到 0x3FFFF

IDE

IDE 用来设定消息标识符的类型，Table 87. 给出了该参数可取的值

Table 94. IDE 值

IDE	描述
CAN_ID_STD	使用标准标识符
CAN_ID_EXT	使用标准标识符 + 扩展标识符

RTR

RTR 用来设定待传输消息的帧类型。它可以设置为数据帧或者远程帧。

Table 95. RTR 值

RTR	描述
CAN_RTR_DATA	数据帧
CAN_RTR_REMOTE	远程帧

DLC

DLC 用来设定待传输消息的帧长度。它的取值范围是 0 到 0x8。

Data[8]

Data[8]包含了待传输数据，它的取值范围为 0 到 0xFF。

FMI

FMI 设定为消息将要通过的过滤器索引，这些消息存储于邮箱中。该参数取值范围 0 到 0xFF。

例：

```
/* Receive a message with the CAN */
CanRxMsg RxMessage;
CAN_Receive(&RxMessage);
```

6.2.12 函数CAN_Sleep

Table 96. 描述了函数 CAN_Sleep

Table 96. 函数 CAN_Sleep

函数名	CAN_Sleep
函数原形	u8 CAN_Sleep(void)
功能描述	使 CAN 进入低功耗模式
输入参数	无
输出参数	无
返回值	CANSLEEPOK 进入睡眠模式 CANSLEEPFAILDED 其他
先决条件	无
被调用函数	无

例：

```
/* Enter the CAN sleep mode*/
CAN_Sleep();
```

6.2.13 函数CAN_WakeUp

Table 97. 描述了函数 CAN_WakeUp

Table 97. 函数 CAN_WakeUp

函数名	CAN_WakeUp
函数原形	u8 CAN_WakeUp(void)
功能描述	将 CAN 唤醒
输入参数	无
输出参数	无
返回值	CANWAKEUPOK 退出睡眠模式 CANWAKEUPFAILDED 其他
先决条件	无
被调用函数	无

例：

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

6.2.14 函数CAN_GetFlagStatus

Table 98. 描述了函数 CAN_GetFlagStatus

Table 98. 函数 CAN_GetFlagStatus

函数名	CAN_GetFlagStatus
函数原形	FlagStatus CAN_GetFlagStatus(u32 CAN_FLAG)
功能描述	检查指定的 CAN 标志位被设置与否
输入参数	CAN_FLAG：待检查的 CAN 标志位 参阅 Section：CAN_FLAG 查阅更多该参数允许取值范围
输出参数	无
返回值	CAN_FLAG 的新状态（SET 或者 RESET）
先决条件	无
被调用函数	无

CAN_FLAG

CAN_FLAG 用来定义待检查的标志位类型。见 Table 99. 查阅更多 CAN_FLAG 取值描述

Table 99. CAN_FLAG 值

CAN_FLAG	描述
CAN_FLAG_EWG	错误警告标志位
CAN_FLAG_EPV	错误被动标志位
CAN_FLAG_BOF	离线标志位

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

Table 100. 描述了函数 CAN_ClearFlag

Table 100. 函数 CAN_ClearFlag

函数名	CAN_ClearFlag
函数原形	void CAN_ClearFlag(u32 CAN_Flag)
功能描述	清除 CAN 的待处理标志位
输入参数	CAN_FLAG：待检查的 CAN 标志位 参阅 Section：CAN_FLAG 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear the CAN bus-off state flag */
CAN_ClearFlag(CAN_FLAG_BOF);
```

6.2.16 函数CAN_GetITStatus

Table 101. 描述了函数 CAN_GetITStatus

Table 101. 函数 CAN_GetITStatus

函数名	CAN_GetITStatus
函数原形	ITStatus CAN_GetITStatus(u32 CAN_IT)
功能描述	检查指定的 CAN 中断发生与否
输入参数	CAN_IT：待检查的 CAN 中断源 参阅 Section：CAN_IT 查阅更多该参数允许取值范围
输出参数	无
返回值	CAN_IT 的新状态（SET 或者 RESET）
先决条件	无
被调用函数	无

CAN_IT

输入参数 CAN_IT 选择待检查的中断。见 Table 102. 查阅更多 CAN_IT 取值描述

Table 102. CAN_IT 值

CAN_IT	描述
CAN_IT_RQCP0	邮箱 1 请求完成
CAN_IT_RQCP1	邮箱 2 请求完成
CAN_IT_RQCP2	邮箱 3 请求完成
CAN_IT_FMP0	FIFO0 消息挂号
CAN_IT_FULL0	FIFO0 已存入 3 消息
CAN_IT_FOVR0	FIFO0 溢出
CAN_IT_FMP1	FIFO1 消息挂号
CAN_IT_FULL1	FIFO1 已存入 3 消息
CAN_IT_FOVR1	FIFO1 溢出
CAN_IT_EWGF	上限到达警告
CAN_IT_EPVF	错误被动上限到达
CAN_IT_BOFF	进入离线状态
CAN_IT_WKUI	睡眠模式下 SOF 侦测

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

Table 103. 描述了函数 CAN_ClearITPendingBit

Table 103. 函数 CAN_ClearITPendingBit

函数名	CAN_ClearITPendingBit
函数原形	void CAN_ClearITPendingBit(u32 CAN_IT)
功能描述	清除 CAN 中断待处理标志位
输入参数	CAN_IT：待清除中断待处理标志位 参阅 Section：CAN_IT 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

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

7 DMA控制器（DMA）

DMA 控制器提供 7 个数据通道的访问。由于外设实现了向存储器的映射，因此数据对来自或者发向外设的数据传输，也可以像内存之间的数据传输一样管理。

Section 7.1 DMA 寄存器结构描述了固件函数库所使用的数据结构，Section 7.2 固件库函数介绍了函数库里的所有函数。

7.1 DMA寄存器结构

DMA 寄存器结构，***DMA_Cannel_TypeDef*** 和 ***DMA_TypeDef***，在文件***"stm32f10x_map.h"***中定义如下：

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

Table 104. 例举了 DMA 所有寄存器

Table 104. DMA 寄存器

寄存器	描述
ISR	DMA 中断状态寄存器
IFCR	DMA 中断标志位清除寄存器
CCRx	DMA 通道 x 设置寄存器
CNDTRx	DMA 通道 x 待传输数据数目寄存器
CPARx	DMA 通道 x 外设地址寄存器
CMARx	DMA 通道 x 内存地址寄存器

DMA 和它的 7 个通道也声明于文件***"stm32f10x.map"***：

```
...
#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
```

使用 Debug 模式时，初始化指针 ***DMA***, ***DMA_Channel1***, ***DMA_Channel2***, ...，和 ***DMA_Channel7*** 于文件“stm32f10x_lib.c”：

```
#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 */
...
```

为了访问 DMA 寄存器，，***_DMA***，***_DMA_Channel1*** 到***_DMA_Channel7*** 必须在文件***“stm32f10x_conf.h”***中定义如下：

```
...
#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 DMA库函数

Table 105. 例举了 DMA 的库函数

Table 105. DMA 库函数

函数名	描述
DMA_DeInit	将 DMA 的通道 x 寄存器重设为缺省值
DMA_Init	根据 DMA_InitStruct 中指定的参数初始化 DMA 的通道 x 寄存器
DMA_StructInit	把 DMA_InitStruct 中的每一个参数按缺省值填入
DMA_Cmd	使能或者失能指定的通道 x
DMA_ITConfig	使能或者失能指定的通道 x 中断
DMA_GetCurrDataCounte	返回当前 DMA 通道 x 剩余的待传输数据数目
DMA_GetFlagStatus	检查指定的 DMA 通道 x 标志位设置与否
DMA_ClearFlag	清除 DMA 通道 x 待处理标志位
DMA_GetITStatus	检查指定的 DMA 通道 x 中断发生与否
DMA_ClearITPendingBit	清除 DMA 通道 x 中断待处理标志位

7.2.1 函数DMA_DeInit

Table 106. 描述了函数 DMA_DeInit

Table 106. 函数 DMA_DeInit

函数名	DMA_DeInit
函数原形	void DMA_DeInit(DMA_Channel_TypeDef* DMA_Channelx)
功能描述	将 DMA 的通道 x 寄存器重设为缺省值
输入参数	DMA Channelx：x 可以是 1，2…，或者 7 来选择 DMA 通道 x
输出参数	无
返回值	无
先决条件	无
被调用函数	RCC_APBPeriphResetCmd()

例：

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

7.2.2 函数DMA_Init

Table 107. 描述了函数 DMA_Init

Table 107. 函数 DMA_Init

函数名	DMA_Init
函数原形	void DMA_Init(DMA_Channel_TypeDef* DMA_Channelx, DMA_InitTypeDef* DMA_InitStruct)
功能描述	根据 DMA_InitStruct 中指定的参数初始化 DMA 的通道 x 寄存器
输入参数 1	DMA Channelx：x 可以是 1，2…，或者 7 来选择 DMA 通道 x
输入参数 2	DMA_InitStruct：指向结构 DMA_InitTypeDef 的指针，包含了 DMA 通道 x 的配置信息 参阅：Section：DMA_InitTypeDef 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

DMA_InitTypeDef structure

DMA_InitTypeDef 定义于文件“stm32f10x_dma.h”：

```
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

该参数用以定义 DMA 外设基地址

DMA_MemoryBaseAddr

该参数用以定义 DMA 内存基地址

DMA_DIR

DMA_DIR 规定了外设是作为数据传输的目的地还是来源。Table 108. 给出了该参数的取值范围。

Table 108. DMA_DIR 值

DMA_DIR	描述
DMA_DIR_PeripheralDST	外设作为数据传输的目的地
DMA_DIR_PeripheralSRC	外设作为数据传输的来源

DMA_BufferSize

DMA_BufferSize 用以定义指定 DMA 通道的 DMA 缓存的大小，单位为数据单位。根据传输方向，数据单位等于结构中参数 DMA_PeripheralDataSize 或者参数 DMA_MemoryDataSize 的值。

DMA_PeripheralInc

DMA_PeripheralInc 用来设定外设地址寄存器递增与否。Table 109. 给出了该参数的取值范围。

Table 109. DMA_PeripheralInc 值

DMA_PeripheralInc	描述
DMA_PeripheralInc_Enable	外设地址寄存器递增
DMA_PeripheralInc_Disable	外设地址寄存器不变

DMA_MemoryInc

DMA_MemoryInc 用来设定内存地址寄存器递增与否。Table 110. 给出了该参数的取值范围。

Table 110. DMA_MemoryInc 值

DMA_MemoryInc	描述
DMA_PeripheralInc_Enable	内存地址寄存器递增
DMA_PeripheralInc_Disable	内存地址寄存器不变

DMA_PeripheralDataSize

DMA_PeripheralDataSize 设定了外设数据宽度。Table 111. 给出了该参数的取值范围。

Table 111. DMA_PeripheralDataSize 值

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

DMA_MemoryDataSize

DMA_MemoryDataSize 设定了外设数据宽度。Table 112. 给出了该参数的取值范围。

Table 112. DMA_MemoryDataSize 值

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

DMA_Mode

DMA_Mode 设置了 CAN 的工作模式，Table 113.给出了该参数可取的值

Table 113. DMA_Mode 值

DMA_Mode	描述
DMA_Mode_Circular	工作在循环缓存模式
DMA_Mode_Normal	工作在正常缓存模式

注意：当指定 DMA 通道数据传输配置为内存到内存时，不能使用循环缓存模式。（见 Section DMA_M2M）

DMA_Priority

DMA_Priority 设定 DMA 通道 x 的软件优先级。Table 114.给出了该参数可取的值。

Table 114. DMA_Priority 值

DMA_Mode	描述
DMA_Priority_VeryHigh	DMA 通道 x 拥有非常高优先级
DMA_Priority_High	DMA 通道 x 拥有高优先级
DMA_Priority_Medium	DMA 通道 x 拥有中优先级
DMA_Priority_Low	DMA 通道 x 拥有低优先级

DMA_M2M

DMA_M2M 使能 DMA 通道的内存到内存传输。Table 115.给出了该参数可取的值。

Table 115. DMA_M2M 值

DMA_M2M	描述
DMA_M2M_Enable	DMA 通道 x 设置为内存到内存传输
DMA_M2M_Disable	DMA 通道 x 没有设置为内存到内存传输

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 函数DMA_StructInit

Table 116. 描述了函数 DMA_StructInit

Table 116. 函数 DMA_StructInit

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

结构 DMA_InitStruct 的各个成员有如下的缺省值：

Table 117. 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

例：

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

7.2.4 函数DMA_Cmd

Table 118. 描述了函数 DMA_Cmd

Table 118. 函数 DMA_Cmd

函数名	DMA_Cmd
函数原形	void DMA_Cmd(DMA_Channel_TypeDef* DMA_Channelx, FunctionalState NewState)
功能描述	使能或者失能指定的通道 x
输入参数 1	DMA Channelx：x 可以是 1，2…，或者 7 来选择 DMA 通道 x
输入参数 2	NewState：DMA 通道 x 的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable DMA Channel7 */
DMA_Cmd(DMA_Channel7, ENABLE);
```

7.2.5 函数DMA_ITConfig

Table 119. 描述了函数 DMA_ITConfig

Table 119. 函数 DMA_ITConfig

函数名	DMA_ITConfig
函数原形	void DMA_ITConfig(DMA_Channel_TypeDef* DMA_Channelx, u32 DMA_IT, FunctionalState NewState)
功能描述	使能或者失能指定的通道 x 中断
输入参数 1	DMA Channelx：x 可以是 1，2…，或者 7 来选择 DMA 通道 x
输入参数 2	DMA_IT：待使能或者失能的 DMA 中断源，使用操作符“|”可以同时选中多个 DMA 中断源 参阅 Section：DMA_IT 查阅更多该参数允许取值范围
输入参数 3	NewState：DMA 通道 x 中断的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

DMA_IT

输入参数 DMA_IT 使能或者失能 DMA 通道 x 的中断。可以取下表的一个或者多个取值的组合作为该参数的值。

Table 120. DMA_IT 值

DMA_IT	描述
DMA_IT_TC	传输完成中断屏蔽
DMA_IT_HT	传输过半中断屏蔽
DMA_IT_TE	传输错误中断屏蔽

例：

```
/* Enable DMA Channel5 complete transfer interrupt */
DMA_ITConfig(DMA_Channel5, DMA_IT_TC, ENABLE);
```

7.2.6 函数DMA_GetCurrDataCounte

Table 121. 描述了函数 DMA_GetCurrDataCounte

Table 121. 函数 DMA_GetCurrDataCounte

函数名	DMA_GetCurrDataCounte
函数原形	u16 DMA_GetCurrDataCounter(DMA_Channel_TypeDef* DMA_Channelx)
功能描述	返回当前 DMA 通道 x 剩余的待传输数据数目
输入参数	DMA Channelx：x 可以是 1，2…，或者 7 来选择 DMA 通道 x
输出参数	无
返回值	当前 DMA 通道 x 剩余的待传输数据数目
先决条件	无
被调用函数	无

例：

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

7.2.7 函数DMA_GetFlagStatus

Table 122. 描述了函数 DMA_GetFlagStatus

Table 122. 函数 DMA_GetFlagStatus

函数名	DMA_GetFlagStatus
函数原形	FlagStatus DMA_GetFlagStatus(u32 DMA_FLAG)
功能描述	检查指定的 DMA 通道 x 标志位设置与否
输入参数	DMA_FLAG：待检查的 DMA 标志位 参阅 Section：DMA_FLAG 查阅更多该参数允许取值范围
输出参数	无
返回值	DMA_FLAG 的新状态（SET 或者 RESET）
先决条件	无
被调用函数	无

DMA_FLAG

参数 DMA_FLAG 定义了待检察的标志位类型。见 Table 123. 查阅更多 DMA_FLAG 取值描述

Table 123. DMA_FLAG 值

DMA_FLAG	描述
DMA_FLAG_GL1	通道 1 全局标志位
DMA_FLAG_TC1	通道 1 传输完成标志位
DMA_FLAG_HT1	通道 1 传输过半标志位
DMA_FLAG_TE1	通道 1 传输错误标志位
DMA_FLAG_GL2	通道 2 全局标志位
DMA_FLAG_TC2	通道 2 传输完成标志位
DMA_FLAG_HT2	通道 2 传输过半标志位
DMA_FLAG_TE2	通道 2 传输错误标志位
DMA_FLAG_GL3	通道 3 全局标志位
DMA_FLAG_TC3	通道 3 传输完成标志位
DMA_FLAG_HT3	通道 3 传输过半标志位
DMA_FLAG_TE3	通道 3 传输错误标志位
DMA_FLAG_GL4	通道 4 全局标志位
DMA_FLAG_TC4	通道 4 传输完成标志位
DMA_FLAG_HT4	通道 4 传输过半标志位
DMA_FLAG_TE4	通道 4 传输错误标志位
DMA_FLAG_GL5	通道 5 全局标志位
DMA_FLAG_TC5	通道 5 传输完成标志位
DMA_FLAG_HT5	通道 5 传输过半标志位
DMA_FLAG_TE5	通道 5 传输错误标志位
DMA_FLAG_GL6	通道 6 全局标志位
DMA_FLAG_TC6	通道 6 传输完成标志位
DMA_FLAG_HT6	通道 6 传输过半标志位
DMA_FLAG_TE6	通道 6 传输错误标志位
DMA_FLAG_GL7	通道 7 全局标志位
DMA_FLAG_TC7	通道 7 传输完成标志位
DMA_FLAG_HT7	通道 7 传输过半标志位
DMA_FLAG_TE7	通道 7 传输错误标志位

例：

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

```
not */
FlagStatus Status;
Status = DMA_GetFlagStatus(DMA_FLAG_HT6);
```

7.2.8 函数DMA_ClearFlag

Table 124. 描述了函数 DMA_ClearFlag

Table 124. 函数 DMA_ClearFlag

函数名	DMA_ClearFlag
函数原形	void DMA_ClearFlag(u32 DMA_FLAG)
功能描述	清除 DMA 通道 x 待处理标志位
输入参数	DMA_FLAG：待清除的 DMA 标志位，使用操作符“|”可以同时选中多个 DMA 标志位 参阅 Section：DMA_FLAG 查阅更多该参数允许取值范围 用户可以使用或操作选中多个标志位
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear the DMA Channel3 transfer error interrupt pending bit */
DMA_ClearFlag(DMA_FLAG_TE3);
```

7.2.9 函数DMA_GetITStatus

Table 125. 描述了函数 DMA_GetITStatus

Table 125. 函数 DMA_GetITStatus

函数名	DMA_GetITStatus
函数原形	ITStatus DMA_GetITStatus(u32 DMA_IT)
功能描述	检查指定的 DMA 通道 x 中断发生与否
输入参数	DMA_IT：待检查的 DMA 中断源 参阅 Section：DMA_IT 查阅更多该参数允许取值范围
输出参数	无
返回值	DMA_IT 的新状态（SET 或者 RESET）
先决条件	无
被调用函数	无

DMA_IT

参数 DMA_IT 定义了待检察的 DMA 中断。见 Table 126. 查阅更多该输入参数取值描述

Table 126. DMA_IT 值

DMA_IT	描述
DMA_IT_GL1	通道 1 全局中断
DMA_IT_TC1	通道 1 传输完成中断
DMA_IT_HT1	通道 1 传输过半中断
DMA_IT_TE1	通道 1 传输错误中断
DMA_IT_GL2	通道 2 全局中断
DMA_IT_TC2	通道 2 传输完成中断
DMA_IT_HT2	通道 2 传输过半中断
DMA_IT_TE2	通道 2 传输错误中断

DMA_IT_GL3	通道 3 全局中断
DMA_IT_TC3	通道 3 传输完成中断
DMA_IT_HT3	通道 3 传输过半中断
DMA_IT_TE3	通道 3 传输错误中断
DMA_IT_GL4	通道 4 全局中断
DMA_IT_TC4	通道 4 传输完成中断
DMA_IT_HT4	通道 4 传输过半中断
DMA_IT_TE4	通道 4 传输错误中断
DMA_IT_GL5	通道 5 全局中断
DMA_IT_TC5	通道 5 传输完成中断
DMA_IT_HT5	通道 5 传输过半中断
DMA_IT_TE5	通道 5 传输错误中断
DMA_IT_GL6	通道 6 全局中断
DMA_IT_TC6	通道 6 传输完成中断
DMA_IT_HT6	通道 6 传输过半中断
DMA_IT_TE6	通道 6 传输错误中断
DMA_IT_GL7	通道 7 全局中断
DMA_IT_TC7	通道 7 传输完成中断
DMA_IT_HT7	通道 7 传输过半中断
DMA_IT_TE7	通道 7 传输错误中断

例：

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

7.2.10 函数DMA_ClearITPendingBit

Table 127. 描述了函数 DMA_ClearITPendingBit

Table 127. 函数 DMA_ClearITPendingBit

函数名	DMA_ClearITPendingBit
函数原形	void DMA_ClearITPendingBit(u32 DMA_IT)
功能描述	清除 DMA 通道 x 中断待处理标志位
输入参数	DMA_IT：待清除的 DMA 中断待处理标志位 参阅 Section：DMA_IT 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

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/事件控制器由 19 个产生事件/中断要求的边沿检测器组成。每个输入线可以独立地配置输入类型（脉冲或挂起）和对应的触发事件（上升沿或下降沿或者双边沿都触发）。每个输入线都可以被独立的屏蔽。挂起寄存器保持着状态线的中断要求。

Section 8.1 EXTI 寄存器结构描述了固件函数库所使用的数据结构，Section 8.2 固件库函数介绍了函数库里的所有函数。

8.1 EXTI寄存器结构

EXTI 寄存器结构，***EXTI_TypeDef***，在文件***"stm32f10x_map.h"***中定义如下：

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

Table 128. 例举了 EXTI 所有寄存器

Table 128. 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

外设 EXTI 也在同一个文件声明如下：

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

使用 Debug 模式时，初始化指针 ***EXTI*** 于文件***"stm32f10x_lib.c"***：

```
#ifdef _EXTI
EXTI = (EXTI_TypeDef *) EXTI_BASE;
#endif /*_EXTI */
```

为了访问 EXTI 寄存器，,***_EXTI*** 必须在文件***"stm32f10x_conf.h"***中定义如下：

```
#define _EXTI
```

8.2 EXTI库函数

Table 129. 例举了 EXTI 的库函数

Table 129. EXTI 库函数

函数名	描述
EXTI_DeInit	将外设 EXTI 寄存器重设为缺省值
EXTI_Init	根据 EXTI_InitStruct 中指定的参数初始化外设 EXTI 寄存器
EXTI_StructInit	把 EXTI_InitStruct 中的每一个参数按缺省值填入
EXTI_GenerateSWInterrupt	产生一个软件中断
EXTI_GetFlagStatus	检查指定的 EXTI 线路标志位设置与否
EXTI_ClearFlag	清除 EXTI 线路挂起标志位
EXTI_GetITStatus	检查指定的 EXTI 线路触发请求发生与否
EXTI_ClearITPendingBit	清除 EXTI 线路挂起位

8.2.1 函数EXTI_DeInit

Table 130. 描述了函数 EXTI_DeInit

Table 130. 函数 EXTI_DeInit

函数名	EXTI_DeInit
函数原形	void EXTI_DeInit(void)
功能描述	将外设 EXTI 寄存器重设为缺省值
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Resets the EXTI registers to their default reset value */
EXTI_DeInit();
```

8.2.2 函数EXTI_Init

Table 131. 描述了函数 EXTI_Init

Table 131. 函数 EXTI_Init

函数名	EXTI_Init
函数原形	void EXTI_Init(EXTI_InitTypeDef* EXTI_InitStruct)
功能描述	根据 EXTI_InitStruct 中指定的参数初始化外设 EXTI 寄存器
输入参数	EXTI_InitStruct：指向结构 EXTI_InitTypeDef 的指针，包含了外设 EXTI 的配置信息 参阅 Section：EXTI_InitTypeDef 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

EXTI_InitTypeDef structure

EXTI_InitTypeDef 定义于文件“stm32f10x_exti.h”：

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

EXTI_Line

EXTI_Line 选择了待使能或者失能的外部线路。Table 132. 给出了该参数可取的值

Table 132. 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 设置了被使能线路的模式。Table 133. 给出了该参数可取的值

Table 133. EXTI_Mode 值

EXTI_Mode	描述
EXTI_Mode_Event	设置 EXTI 线路为事件请求
EXTI_Mode_Interrupt	设置 EXTI 线路为中断请求

EXTI_Trigger

EXTI_Trigger 设置了被使能线路的触发边沿。Table 134. 给出了该参数可取的值

Table 134. EXTI_Trigger 值

EXTI_Trigger	描述
EXTI_Trigger_Falling	设置输入线路下降沿为中断请求
EXTI_Trigger_Rising	设置输入线路上升沿为中断请求
EXTI_Trigger_Rising_Falling	设置输入线路上升沿和下降沿为中断请求

EXTI_LineCmd

EXTI_LineCmd 用来定义选中线路的新状态。它可以被设为 ENABLE 或者 DISABLE。

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 函数EXTI_StructInit

Table 135. 描述了函数 EXTI_StructInit

Table 135. 函数 EXTI_StructInit

函数名	EXTI_StructInit
函数原形	void EXTI_StructInit(EXTI_InitTypeDef*EXTI_InitStruct)
功能描述	把 EXTI_InitStruct 中的每一个参数按缺省值填入
输入参数	EXTI_InitStruct：指向结构 EXTI_InitTypeDef 的指针，待初始化
输出参数	无
返回值	无
先决条件	无
被调用函数	无

Table 136. 给出了 EXTI_InitStruct 各个成员的缺省值

Table 136. EXTI_InitStruct 缺省值

成员	缺省值
EXTI_Line	EXTI_LineNone
EXTI_Mode	EXTI_Mode_Interrupt
EXTI_Trigger	EXTI_Trigger_Falling
EXTI_LineCmd	DISABLE

例：

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

8.2.4 函数EXTI_GenerateSWInterrupt

Table 137. 描述了函数 EXTI_GenerateSWInterrupt

Table 137. 函数 EXTI_GenerateSWInterrupt

函数名	EXTI_GenerateSWInterrupt
函数原形	void EXTI_GenerateSWInterrupt(u32 EXTI_Line)
功能描述	产生一个软件中断
输入参数	EXTI_Line：待使能或者失能的 EXTI 线路 参阅 Section：EXTI_Line 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

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

8.2.5 函数EXTI_GetFlagStatus

Table 138. 描述了函数 EXTI_GetFlagStatus

Table 138. 函数 EXTI_GetFlagStatus

函数名	EXTI_GetFlagStatus
函数原形	FlagStatus EXTI_GetFlagStatus(u32 EXTI_Line)
功能描述	检查指定的 EXTI 线路标志位设置与否
输入参数	EXTI_Line：待检查的 EXTI 线路标志位 参阅 Section：EXTI_Line 查阅更多该参数允许取值范围
输出参数	无
返回值	EXTI_Line 的新状态（SET 或者 RESET）
先决条件	无
被调用函数	无

例：

```
/* Get the status of EXTI line 8 */
FlagStatus EXTIStatus;
EXTIStatus = EXTI_GetFlagStatus(EXTI_Line8);
```

8.2.6 函数EXTI_ClearFlag

Table 139. 描述了函数 EXTI_ClearFlag

Table 139. 函数 EXTI_ClearFlag

函数名	EXTI_ClearFlag
函数原形	void EXTI_ClearFlag(u32 EXTI_Line)
功能描述	清除 EXTI 线路挂起标志位
输入参数	EXTI_Line：待清除标志位的 EXTI 线路 参阅 Section：EXTI_Line 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear the EXTI line 2 pending flag */
EXTI_ClearFlag(EXTI_Line2);
```

8.2.7 函数EXTI_GetITStatus

Table 140. 描述了函数 EXTI_GetITStatus

Table 140. 函数 EXTI_GetITStatus

函数名	EXTI_GetITStatus
函数原形	ITStatus EXTI_GetITStatus(u32 EXTI_Line)
功能描述	检查指定的 EXTI 线路触发请求发生与否
输入参数	EXTI_Line：待检查 EXTI 线路的挂起位 参阅 Section：EXTI_Line 查阅更多该参数允许取值范围
输出参数	无
返回值	EXTI_Line 的新状态（SET 或者 RESET）
先决条件	无
被调用函数	无

例：

```
/* Get the status of EXTI line 8 */
ITStatus EXTIStatus;
EXTIStatus = EXTI_GetITStatus(EXTI_Line8);
```

8.2.8 函数EXTI_ClearITPendingBit

Table 141. 描述了函数 EXTI_ClearITPendingBit

Table 141. 函数 EXTI_ClearITPendingBit

函数名	EXTI_ClearITPendingBit
函数原形	void EXTI_ClearITPendingBit(u32 EXTI_Line)
功能描述	清除 EXTI 线路挂起位
输入参数	EXTI_Line：待清除 EXTI 线路的挂起位 参阅 Section：EXTI_Line 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

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

9 FLASH存储器(FLASH)

Section 9.1 FLASH 寄存器结构描述了固件函数库所使用的数据结构，Section 9.2 固件库函数介绍了函数库里的所有函数。

9.1 FLASH寄存器结构

FLASH 寄存器结构，***FLASH_TypeDef*** 和 ***OB_TypeDef***，在文件***"stm32f10x_map.h"***中定义如下：

```
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;
```

Table 142.和 Table 143.例举了 FLASH 所有寄存器和选择字节（Option Byte OB）寄存器

Table 142. FLASH 寄存器

寄存器	描述
ACR	FLASH 访问控制寄存器
KEYR	FPEC 密钥寄存器
OPTKEYR	选择字节密钥寄存器
SR	FLASH 状态寄存器
CR	FLASH 控制寄存器
AR	FLASH 地址寄存器
OBR	选择字节和状态寄存器
WRPR	选择字节写保护寄存器

Table 143. Option Byte（OB）寄存器

寄存器	描述
RDR	读出选择字节
USER	用户选择字节
Data0	Data0 选择字节
Data1	Data1 选择字节
WRP0	写保护 0 选择字节
WRP1	写保护 1 选择字节

WRP2	写保护 2 选择字节
WRP3	写保护 3 选择字节

FLASH 也在文件***"stm32f10x_map.h"***声明如下：

```
/* 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
```

使用 Debug 模式时，初始化指针 ***FLASH*** 和 ***OB*** 于文件***"stm32f10x_lib.c"***：

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

为了访问 EXTI 寄存器，***_FLASH*** 必须在文件***"stm32f10x_conf.h"***中定义如下：

```
#define _FLASH
```

在默认情况下，只有执行 FLASH 设置（延迟，预取指，半周期）的函数式允许执行的。

如果想要执行 FLASH 编写/擦除/保护函数，必须在文件***"stm32f10x_conf.h"***中定义***_FLASH_PROG*** 如下：

```
#define _FLASH_PROG
```

9.2 FLASH库函数

Table 144. 例举了 FLASH 的库函数

Table 144. FLASH 库函数

函数名	描述
FLASH_SetLatency	设置代码延时值
FLASH_HalfCycleAccessCmd	使能或者失能 FLASH 半周期访问
FLASH_PrefetchBufferCmd	使能或者失能预取指缓存
FLASH_Unlock	解锁 FLASH 编写擦除控制器
FLASH_Lock	锁定 FLASH 编写擦除控制器
FLASH_ErasePage	擦除一个 FLASH 页面
FLASH_EraseAllPages	擦除全部 FLASH 页面
FLASH_EraseOptionBytes	擦除 FLASH 选择字节
FLASH_ProgramWord	在指定地址编写一个字
FLASH_ProgramHalfWord	在指定地址编写半字
FLASH_ProgramOptionByteData	在指定 FLASH 选择字节地址编写半字
FLASH_EnableWriteProtection	对期望的页面写保护
FLASH_ReadOutProtection	使能或者失能读出保护
FLASH_UserOptionByteConfig	编写 FLASH 用户选择字节：IWDG_SW /RST_STOP /RST_STDBY
FLASH_GetUserOptionByte	返回 FLASH 用户选择字节的值
FLASH_GetWriteProtectionOptionByte	返回 FLASH 写保护选择字节的值
FLASH_GetReadOutProtectionStatus	检查 FLASH 读出保护设置与否

FLASH_GetPrefetchBufferStatus	检查 FLASH 预取指缓存设置与否
FLASH_ITConfig	使能或者失能指定 FLASH 中断
FLASH_GetFlagStatus	检查指定的 FLASH 标志位设置与否
FLASH_ClearFlag	清除 FLASH 待处理标志位
FLASH_GetStatus	返回 FLASH 状态
FLASH_WaitForLastOperation	等待某一个 Flash 操作完成，或者发生 TIMEOUT

9.2.1 函数FLASH_SetLatency

Table 145. 描述了函数 FLASH_SetLatency

Table 145. 函数 FLASH_SetLatency

函数名	FLASH_SetLatency
函数原形	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 FLASH_Latency 的值 参阅 Section：FLASH_Latency 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

FLASH_Latency

FLASH_Latency 用来设置 FLASH 存储器延时时钟周期数。见 Table 146. 查阅该参数可取的值

Table 146. FLASH_Latency 值

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

例：

```
/* Configure the Latency cycle: Set 2 Latency cycles */
FLASH_SetLatency(FLASH_Latency_2);
```

9.2.2 函数FLASH_HalfCycleAccessCmd

Table 147. 描述了函数 FLASH_HalfCycleAccessCmd

Table 147. 函数 FLASH_HalfCycleAccessCmd

函数名	FLASH_HalfCycleAccessCmd
函数原形	void FLASH_HalfCycleAccessCmd(u32 FLASH_HalfCycleAccess)
功能描述	使能或者失能 FLASH 半周期访问
输入参数	FLASH_HalfCycleAccess：FLASH_HalfCycle 访问模式 参阅 Section：FLASH_HalfCycle 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

FLASH_HalfCycleAccess

FLASH_HalfCycleAccess 用来选择 FLASH 半周期访问的模式。见 Table 148. 查阅该参数可取的值

Table 148. FLASH_HalfCycleAccess 值

FLASH_HalfCycleAccess	描述
FLASH_HalfCycleAccess_Enable	半周期访问使能
FLASH_HalfCycleAccess_Disable	半周期访问失能

例：

```
/* Enable the Half Cycle Flash access */
FLASH_HalfCycleAccessCmd(FLASH_HalfCycleAccess_Enable);
```

9.2.3 函数FLASH_PrefetchBufferCmd

Table 149. 描述了函数 FLASH_PrefetchBufferCmd

Table 149. 函数 FLASH_PrefetchBufferCmd

函数名	FLASH_PrefetchBufferCmd
函数原形	void FLASH_PrefetchBufferCmd(u32 FLASH_PrefetchBuffer)
功能描述	使能或者失能预取指缓存
输入参数	FLASH_PrefetchBuffer：预取指缓存状态 参阅 Section：FLASH_PrefetchBuffer 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

FLASH_PrefetchBuffer

FLASH_PrefetchBuffer 用来选择 FLASH 预取指缓存的模式。见 Table 150. 查阅该参数可取的值

Table 150. FLASH_PrefetchBuffer 值

FLASH_PrefetchBuffer	描述
FLASH_PrefetchBuffer_Enable	预取指缓存使能
FLASH_PrefetchBuffer_Disable	预取指缓存失能

例：

```
/* Enable The Prefetch Buffer */
FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
```

9.2.4 函数FLASH_Unlock

Table 151. 描述了函数 FLASH_Unlock

Table 151. 函数 FLASH_Unlock

函数名	FLASH_Unlock
函数原形	void FLASH_Unlock(void)
功能描述	解锁 FLASH 编写擦除控制器
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Unlocks the Flash
FLASH_Unlock();
```

9.2.5 函数FLASH_Lock

Table 152. 描述了函数 FLASH_Lock

Table 152. 函数 FLASH_Lock

函数名	FLASH_Lock
函数原形	void FLASH_Lock(void)
功能描述	锁定 FLASH 编写擦除控制器
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Locks the Flash */
FLASH_Lock();
```

9.2.6 函数FLASH_ErasePage

Table 153. 描述了函数 FLASH_ErasePage

Table 153. 函数 FLASH_ErasePage

函数名	FLASH_ErasePage
函数原形	FLASH_Status FLASH_ErasePage(u32 Page_Address)
功能描述	擦除一个 FLASH 页面
输入参数	无
输出参数	无
返回值	擦除操作状态
先决条件	无
被调用函数	无

例：

```
/* Erases the Flash Page 0 */
FLASH_Status status = FLASH_COMPLETE;
status = FLASH_ErasePage(0x08000000);
```

9.2.7 函数FLASH_EraseAllPages

Table 154. 描述了函数 FLASH_EraseAllPages

Table 154. 函数 FLASH_EraseAllPages

函数名	FLASH_EraseAllPages
函数原形	FLASH_Status FLASH_EraseAllPages(void)
功能描述	擦除全部 FLASH 页面
输入参数	无
输出参数	无
返回值	擦除操作状态
先决条件	无
被调用函数	无

例：

```
/* Erases the Flash */
FLASH_Status status = FLASH_COMPLETE;
status = FLASH_EraseAllPages();
```

9.2.8 函数FLASH_EraseOptionBytes

Table 155. 描述了函数 FLASH_EraseOptionBytes

Table 155. 函数 FLASH_EraseOptionBytes

函数名	FLASH_EraseOptionBytes
函数原形	FLASH_Status FLASH_EraseOptionBytes(void)
功能描述	擦除 FLASH 选择字节
输入参数	无
输出参数	无
返回值	擦除操作状态
先决条件	无
被调用函数	无

例：

```
/* Erases the Flash Option Bytes */
FLASH_Status status = FLASH_COMPLETE;
status = FLASH_EraseOptionBytes();
```

9.2.9 函数FLASH_ProgramWord

Table 156. 描述了函数 FLASH_ProgramWord

Table 156. 函数 FLASH_ProgramWord

函数名	FLASH_ProgramWord
函数原形	FLASH_Status FLASH_ProgramWord(u32 Address, u32 Data)
功能描述	在指定地址编写一个字
输入参数 1	Address：待编写的地址
输入参数 2	Data：待写入的数据
输出参数	无
返回值	编写操作状态
先决条件	无
被调用函数	无

例：

```
/* Writes the Data1 at the Address1 */
FLASH_Status status = FLASH_COMPLETE;
u32 Data1 = 0x1234567;
u32 Address1 = 0x8000000;
status = FLASH_ProgramWord(Address1, Data1);
```

9.2.10 函数FLASH_ProgramHalfWord

Table 157. 描述了函数 FLASH_ProgramHalfWord

Table 157. 函数 FLASH_ProgramHalfWord

函数名	FLASH_ProgramHalfWord
函数原形	FLASH_Status FLASH_ProgramHalfWord(u32 Address, u16 Data)
功能描述	在指定地址编写半字
输入参数 1	Address：待编写的地址
输入参数 2	Data：待写入的数据
输出参数	无
返回值	编写操作状态
先决条件	无
被调用函数	无

例：

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

9.2.11 函数FLASH_ProgramOptionByteData

Table 158. 描述了函数 FLASH_ProgramOptionByteData

Table 158. 函数 FLASH_ProgramOptionByteData

函数名	FLASH_ProgramOptionByteData
函数原形	FLASH_Status FLASH_ProgramOptionByteData(u32 Address, u8 Data)
功能描述	在指定 FLASH 选择字节地址编写半字
输入参数 1	Address：待编写的地址，该参数取值可以是 0x1FFF804 或者 0x1FFF806
输入参数 2	Data：待写入的数据
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

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

9.2.12 函数FLASH_EnableWriteProtection

Table 159. 描述了函数 FLASH_EnableWriteProtection

Table 159. 函数 FLASH_EnableWriteProtection

函数名	FLASH_EnableWriteProtection
函数原形	FLASH_Status FLASH_EnableWriteProtection(u32 FLASH_Pages)
功能描述	对期望的页面写保护
输入参数	FLASH_Page：待写保护页面的地址 参阅 Section：FLASH_Page 查阅更多该参数允许取值范围
输出参数	无
返回值	写保护操作状态
先决条件	无
被调用函数	无

FLASH_Pages

FLASH_Page 用来设置写保护的页面。见 Table 160. 查阅该参数可取的值

Table 160. FLASH_Page 值

FLASH_Page	描述
FLASH_WRProt_Pages0to3	写保护页面 0 到 3
FLASH_WRProt_Pages4to7	写保护页面 4 到 7
FLASH_WRProt_Pages8to11	写保护页面 8 到 11
FLASH_WRProt_Pages12to15	写保护页面 12 到 15
FLASH_WRProt_Pages16to19	写保护页面 16 到 19
FLASH_WRProt_Pages20to23	写保护页面 20 到 23
FLASH_WRProt_Pages24to27	写保护页面 24 到 27
FLASH_WRProt_Pages28to31	写保护页面 28 到 31
FLASH_WRProt_Pages32to35	写保护页面 32 到 35
FLASH_WRProt_Pages36to39	写保护页面 36 到 39
FLASH_WRProt_Pages40to43	写保护页面 40 到 43
FLASH_WRProt_Pages44to47	写保护页面 44 到 47
FLASH_WRProt_Pages48to51	写保护页面 58 到 51
FLASH_WRProt_Pages52to55	写保护页面 52 到 55
FLASH_WRProt_Pages56to59	写保护页面 56 到 59
FLASH_WRProt_Pages60to63	写保护页面 60 到 63
FLASH_WRProt_Pages64to67	写保护页面 64 到 67
FLASH_WRProt_Pages68to71	写保护页面 68 到 71
FLASH_WRProt_Pages72to75	写保护页面 72 到 75
FLASH_WRProt_Pages76to79	写保护页面 76 到 79
FLASH_WRProt_Pages80to83	写保护页面 80 到 83
FLASH_WRProt_Pages84to87	写保护页面 84 到 87
FLASH_WRProt_Pages88to91	写保护页面 88 到 91
FLASH_WRProt_Pages92to95	写保护页面 92 到 95
FLASH_WRProt_Pages96to99	写保护页面 92 到 99
FLASH_WRProt_Pages100to103	写保护页面 100 到 103
FLASH_WRProt_Pages104to107	写保护页面 104 到 107
FLASH_WRProt_Pages108to111	写保护页面 108 到 111
FLASH_WRProt_Pages112to115	写保护页面 112 到 115
FLASH_WRProt_Pages116to119	写保护页面 115 到 119

FLASH_WRProt_Pages120to123	写保护页面 120 到 123
FLASH_WRProt_Pages124to127	写保护页面 124 到 127
FLASH_WRProt_AllPages	写保护全部页面

例：

```
/* Protects the Pages0to3 and Pages108to111 */
FLASH_Status status = FLASH_COMPLETE;
status = FLASH_EnableWriteProtection
(FLASH_WRProt_Pages0to3|FLASH_WRProt_Pages108to111);
```

9.2.13 函数FLASH_ReadOutProtection

Table 161. 描述了函数 FLASH_ReadOutProtection

Table 161. 函数 FLASH_ReadOutProtection

函数名	FLASH_ReadOutProtection
函数原形	FLASH_Status FLASH_ReadOutProtection(FunctionalState NewState)
功能描述	使能或者失能读出保护
输入参数	NewState：读出保护的新状态。 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	保护操作状态
先决条件	如果用户在调用本函数之前编写过其他选择字节，那么必须在调用本函数之后重新编写选择字节，因为本操作会擦除所有选择字节
被调用函数	无

例：

```
/* Disables the ReadOut Protection */
FLASH_Status status = FLASH_COMPLETE;
status = FLASH_ReadOutProtection(DISABLE);
```

注意：为了安全地编写选择字节，用户必须遵从下列操作步骤：

1. 如果想要读保护 Flash 存储器，调用函数 FLASH_ReadOutProtection
2. 调用函数 FLASH_EnableWriteProtection 来写保护 Flash 存储器部分或者全部页面
3. 调用函数 FLASH_UserOptionByteConfig 来设置用户选择字节：IWDG_SW /RST_STOP /RST_STDBY
4. 调用函数 FLASH_ProgramOptionByteData 来对指定选择字节数据地址写入半字
5. 产生复位以装入新的选择字节

9.2.14 函数FLASH_UserOptionByteConfig

Table 162. 描述了函数 FLASH_UserOptionByteConfig

Table 162. 函数 FLASH_UserOptionByteConfig

函数名	FLASH_UserOptionByteConfig
函数原形	FLASH_Status FLASH_UserOptionByteConfig(u16 OB_IWDG, u16 OB_STOP, u16 OB_STDBY)
功能描述	编写 FLASH 用户选择字节：IWDG_SW /RST_STOP /RST_STDBY
输入参数 1	OB_IWDG：选择 IWDG 模式 参阅 Section：OB_IWDG 查阅更多该参数允许取值范围
输入参数 2	OB_STOP：当进入 STOP 模式产生复位事件 参阅 Section：OB_STOP 查阅更多该参数允许取值范围
输入参数 3	OB_STDBY：当进入 Standby 模式产生复位事件 参阅 Section：OB_STDBY 查阅更多该参数允许取值范围
输出参数	无
返回值	选择字节编写状态
先决条件	无
被调用函数	无

OB_IWDG

OB_IWDG 用来选择 IWDG(独立看门狗)的模式。见 Table 163. 查阅该参数可取的值

Table 163. OB_IWDG 值

OB_IWDG	描述
OB_IWDG_SW	选择软件独立看门狗
OB_IWDG_HW	选择硬件独立看门狗

OB_STOP

OB_STOP 用来选择进入 STOP 模式是否产生复位。见 Table 164. 查阅该参数可取的值

Table 164. OB_STOP 值

OB_STOP	描述
OB_STOP_NoRST	进入 STOP 模式不产生复位
OB_STOP_RST	进入 STOP 模式产生复位

OB_STDBY

OB_STDBY 用来选择进入 Standby 模式是否产生复位。见 Table 165. 查阅该参数可取的值

Table 165. OB_STDBY 值

OB_STDBY	描述
OB_STDBY_NoRST	进入 Standby 模式不产生复位
OB_STDBY_RST	进入 Standby 模式产生复位

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);
```

9.2.15 函数FLASH_GetUserOptionByte

Table 166. 描述了函数 FLASH_GetUserOptionByte

Table 166. 函数 FLASH_GetUserOptionByte

函数名	FLASH_GetUserOptionByte
函数原形	u32 FLASH_GetUserOptionByte(void)
功能描述	返回 FLASH 用户选择字节的值
输入参数	无
输出参数	无
返回值	FLASH 用户选择字节的值：IWDG_SW(Bit0), RST_STOP(Bit1) and RST_STDBY(Bit2)
先决条件	无
被调用函数	无

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;
```

9.2.16 函数FLASH_GetWriteProtectionOptionByte

Table 167. 描述了函数 FLASH_GetWriteProtectionOptionByte

Table 167. 函数 FLASH_GetWriteProtectionOptionByte

函数名	FLASH_GetWriteProtectionOptionByte
函数原形	u32 FLASH_GetWriteProtectionOptionByte(void)
功能描述	返回 FLASH 写保护选择字节的值
输入参数	无
输出参数	无
返回值	FLASH 写保护选择字节的值
先决条件	无
被调用函数	无

例：

```
/* Gets the Write Protection option byte values */
u32 WriteProtectionValue = 0x0;
WriteProtectionValue = FLASH_GetWriteProtectionOptionByte();
```

9.2.17 函数FLASH_GetReadOutProtectionStatus

Table 168. 描述了函数 FLASH_GetReadOutProtectionStatus

Table 168. 函数 FLASH_GetReadOutProtectionStatus

函数名	FLASH_GetReadOutProtectionStatus
函数原形	FlagStatus FLASH_GetReadOutProtectionStatus(void)
功能描述	检查 FLASH 读出保护设置与否
输入参数	无
输出参数	无
返回值	FLASH 读出保护状态（SET 或者 RESET）
先决条件	无
被调用函数	无

例：

```
/* Gets the ReadOut Protection status */
FlagStatus status = RESET;
status = FLASH_GetReadOutProtectionStatus();
```

9.2.18 函数FLASH_GetPrefetchBufferStatus

Table 169. 描述了函数 FLASH_GetPrefetchBufferStatus

Table 169. 函数 FLASH_GetPrefetchBufferStatus

函数名	FLASH_GetPrefetchBufferStatus
函数原形	FlagStatus FLASH_GetPrefetchBufferStatus(void)
功能描述	检查 FLASH 预取指缓存设置与否
输入参数	无
输出参数	无
返回值	FLASH 预取指缓存状态（SET 或者 RESET）
先决条件	无
被调用函数	无

例：

```
/* Gets the Prefetch Buffer status */
FlagStatus status = RESET;
status = FLASH_GetPrefetchBufferStatus();
```

9.2.19 函数FLASH_ITConfig

Table 170. 描述了函数 FLASH_ITConfig

Table 170. 函数 FLASH_ITConfig

函数名	FLASH_ITConfig
函数原形	void FLASH_ITConfig(u16 FLASH_IT, FunctionalState NewState)
功能描述	使能或者失能指定 FLASH 中断
输入参数 1	FLASH_IT：待使能或者失能的指定 FLASH 中断源 参阅 Section：FLASH_IT 查阅更多该参数允许取值范围
输入参数 2	NewState：指定 FLASH 中断的新状态。 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

FLASH_IT

FLASH_IT 用来使能或者失能指定的 FLASH 中断。见 Table 171. 查阅该参数可取的值

Table 171. 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

例：

```
/* Enables the EOP Interrupt source */
FLASH_ITConfig(FLASH_IT_EOP, ENABLE);
```

9.2.20 函数FLASH_GetFlagStatus

Table 172. 描述了函数 FLASH_GetFlagStatus

Table 172. 函数 FLASH_GetFlagStatus

函数名	FLASH_GetFlagStatus
函数原形	FlagStatus FLASH_GetFlagStatus(u16 FLASH_FLAG)
功能描述	检查指定的 FLASH 标志位设置与否
输入参数	FLASH_FLAG：待检查的标志位 参阅 Section：FLASH_FLAG 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

FLASH_FLAG

FLASH_FLAG 为能够被函数 FLASH_GetFlagStatus 检查的标志位。它们列举于下表：

Table 173. FLASH_FLAG 值

FLASH_FLAG	描述
FLASH_FLAG_BSY	FLASH 忙标志位
FLASH_FLAG_EOP	FLASH 操作结束标志位
FLASH_FLAG_PGERR	FLASH 编写错误标志位

FLASH_FLAG_WRPRTERR	FLASH 页面写保护错误标志位
FLASH_FLAG_OPTERR	FLASH 选择字节错误标志位

例：
```
/* Checks whether the EOP Flag Status is SET or not */
FlagStatus status = RESET;
status = FLASH_GetFlagStatus(FLASH_FLAG_EOP);
```

9.2.21 函数FLASH_ClearFlag

Table 174. 描述了函数 FLASH_ClearFlag

Table 174. 函数 FLASH_ClearFlag

函数名	FLASH_ClearFlag
函数原形	void FLASH_ClearFlag(u16 FLASH_Flag)
功能描述	清除 FLASH 待处理标志位
输入参数	FLASH_FLAG：待清除的标志位 参阅 Section：FLASH_FLAG 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

FLASH_FLAG

FLASH_FLAG 为能够被函数 FLASH_ClearFlag 清除的标志位。它们列举于下表：

Table 175. FLASH_FLAG 值

FLASH_FLAG	描述
FLASH_FLAG_BSY	FLASH 忙标志位
FLASH_FLAG_EOP	FLASH 操作结束标志位
FLASH_FLAG_PGERR	FLASH 编写错误标志位
FLASH_FLAG_WRPRTERR	FLASH 页面写保护错误标志位

例：
```
/* Clears all flags */
FLASH_ClearFlag(FLASH_FLAG_BSY|FLASH_FLAG_EOP|FLASH_FLAG_PGER
|FLASH_FLAG_WRPRTERR);
```

9.2.22 函数FLASH_GetStatus

Table 176. 描述了函数 FLASH_GetStatus

Table 176. 函数 FLASH_GetStatus

函数名	FLASH_GetStatus
函数原形	FLASH_Status FLASH_GetStatus(void)
功能描述	返回 FLASH 状态
输入参数	无
输出参数	无
返回值	FLASH_Status：返回值可以是：FLASH_BUSY，FLASH_ERROR_PG，FLASH_ERROR_WRP 或者 FLASH_COMPLETE
先决条件	无
被调用函数	无

例：

```
/* Check for the Flash status */
FLASH_Status status = FLASH_COMPLETE;
status = FLASH_GetStatus();
```

9.2.23 函数FLASH_WaitForLastOperation

Table 177. 描述了函数 FLASH_WaitForLastOperation

Table 177. 函数 FLASH_WaitForLastOperation

函数名	FLASH_WaitForLastOperation
函数原形	FLASH_Status FLASH_WaitForLastOperation(u32 Timeout)
功能描述	等待某一个 Flash 操作完成，或者发生 TIMEOUT
输入参数	无
输出参数	无
返回值	返回适当的操作状态。 这个参数可以是：FLASH_BUSY，FLASH_ERROR_PG，FLASH_ERROR_WRP，FLASH_COMPLETE 或者 FLASH_TIMEOUT
先决条件	无
被调用函数	无

例：

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

10 通用输入/输出（GPIO）

GPIO 驱动可以用作多个用途，包括管脚设置，单位设置/重置，锁定机制，从端口管脚读入或者向端口管脚写入数据。

Section 10.1 GPIO 寄存器结构描述了固件函数库所使用的数据结构，Section 10.2 固件库函数介绍了函数库里的所有函数。

10.1 GPIO寄存器结构

GPIO 寄存器结构，***GPIO_TypeDef*** 和 ***AFIO_TypeDef***，在文件***"stm32f10x_map.h"***中定义如下：

```
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;
```

Table 178.例举了 GPIO 所有寄存器

Table 178. GPIO 寄存器

寄存器	描述
CRL	端口配置低寄存器
CRH	端口配置高寄存器
IDR	端口输入数据寄存器
ODR	端口输出数据寄存器
BSRR	端口位设置/复位寄存器
BRR	端口位复位寄存器
LCKR	端口配置锁定寄存器
EVCR	事件控制寄存器
MAPR	复用重映射和调试 I/O 配置寄存器
EXTICR	外部中断线路 0-15 配置寄存器

五个 GPIO 外设声明于文件***"stm32f10x_map.h"***：

```
...
#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
```

使用 Debug 模式时，初始化指针 ***AFIO, GPIOA, GPIOB, GPIOC, GPIOD*** 和 ***GPIOE*** 于文件***“stm32f10x_lib.c”***：

```
#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*** 和***_GPIOE*** 必须在文件***“stm32f10x_conf.h”***中定义如下：

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

```
#define _GPIOE
#define _AFIO
```

10.2 GPIO库函数

Table 179. 例举了 GPIO 的库函数

Table 179. GPIO 库函数

函数名	描述
GPIO_DeInit	将外设 GPIOx 寄存器重设为缺省值
GPIO_AFIODeInit	将复用功能（重映射事件控制和 EXTI 设置）重设为缺省值
GPIO_Init	根据 GPIO_InitStruct 中指定的参数初始化外设 GPIOx 寄存器
GPIO_StructInit	把 GPIO_InitStruct 中的每一个参数按缺省值填入
GPIO_ReadInputDataBit	读取指定端口管脚的输入
GPIO_ReadInputData	读取指定的 GPIO 端口输入
GPIO_ReadOutputDataBit	读取指定端口管脚的输出
GPIO_ReadOutputData	读取指定的 GPIO 端口输出
GPIO_SetBits	设置指定的数据端口位
GPIO_ResetBits	清除指定的数据端口位
GPIO_WriteBit	设置或者清除指定的数据端口位
GPIO_Write	向指定 GPIO 数据端口写入数据
GPIO_PinLockConfig	锁定 GPIO 管脚设置寄存器
GPIO_EventOutputConfig	选择 GPIO 管脚用作事件输出
GPIO_EventOutputCmd	使能或者失能事件输出
GPIO_PinRemapConfig	改变指定管脚的映射
GPIO_EXTILineConfig	选择 GPIO 管脚用作外部中断线路

10.2.1 函数GPIO_DeInit

Table 180. 描述了函数 GPIO_DeInit

Table 180. 函数 GPIO_DeInit

函数名	GPIO_DeInit
函数原形	void GPIO_DeInit(GPIO_TypeDef* GPIOx)
功能描述	将外设 GPIOx 寄存器重设为缺省值
输入参数	GPIOx：x 可以是 A，B，C，D 或者 E，来选择 GPIO 外设
输出参数	无
返回值	无
先决条件	无
被调用函数	RCC_APB2PeriphResetCmd()

例：

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

10.2.2 函数GPIO_AFIODeInit

Table 181. 描述了函数 GPIO_AFIODeInit

Table 181. 函数 GPIO_AFIODeInit

函数名	GPIO_AFIODeInit
函数原形	void GPIO_AFIODeInit(void)
功能描述	将复用功能（重映射事件控制和 EXTI 设置）重设为缺省值
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	RCC_APB2PeriphResetCmd()

例：

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

10.2.3 函数GPIO_Init

Table 182. 描述了函数 GPIO_Init

Table 182. 函数 GPIO_Init

函数名	GPIO_Init
函数原形	void GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct)
功能描述	根据 GPIO_InitStruct 中指定的参数初始化外设 GPIOx 寄存器
输入参数 1	GPIOx：x 可以是 A，B，C，D 或者 E，来选择 GPIO 外设
输入参数 2	GPIO_InitStruct：指向结构 GPIO_InitTypeDef 的指针，包含了外设 GPIO 的配置信息 参阅 Section：GPIO_InitTypeDef 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

GPIO_InitTypeDef structure

GPIO_InitTypeDef 定义于文件***"stm32f10x_gpio.h"***：

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

GPIO_Pin

该参数选择待设置的 GPIO 管脚，使用操作符"|"可以一次选中多个管脚。可以使用下表中的任意组合。

Table 183. GPIO_Pin 值

GPIO_Pin	描述
GPIO_Pin_None	无管脚被选中
GPIO_Pin_0	选中管脚 0
GPIO_Pin_1	选中管脚 1
GPIO_Pin_2	选中管脚 2
GPIO_Pin_3	选中管脚 3
GPIO_Pin_4	选中管脚 4
GPIO_Pin_5	选中管脚 5
GPIO_Pin_6	选中管脚 6
GPIO_Pin_7	选中管脚 7
GPIO_Pin_8	选中管脚 8
GPIO_Pin_9	选中管脚 9
GPIO_Pin_10	选中管脚 10
GPIO_Pin_11	选中管脚 11
GPIO_Pin_12	选中管脚 12
GPIO_Pin_13	选中管脚 13
GPIO_Pin_14	选中管脚 14
GPIO_Pin_15	选中管脚 15
GPIO_Pin_All	选中全部管脚

GPIO_Speed

GPIO_Speed 用以设置选中管脚的速率。Table 184. 给出了该参数可取的值

Table 184. GPIO_Speed 值

GPIO_Speed	描述
GPIO_Speed_10MHz	最高输出速率 10MHz
GPIO_Speed_2MHz	最高输出速率 2MHz
GPIO_Speed_50MHz	最高输出速率 50MHz

GPIO_Mode

GPIO_Mode 用以设置选中管脚的工作状态。Table 185. 给出了该参数可取的值

Table 185. GPIO_Mode 值

GPIO_Speed	描述
GPIO_Mode_AIN	模拟输入
GPIO_Mode_IN_FLOATING	浮空输入
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	推挽输出
GPIO_Mode_AF_OD	复用开漏输出
GPIO_Mode_AF_PP	复用推挽输出

注意：

- 当某管脚设置为上拉或者下拉输入模式，使用寄存器 Px_BSRR 和 PxBRR
- GPIO_Mode 允许同时设置 GPIO 方向（输入/输出）和对应的输入/输出设置，：位[7:4]对应 GPIO 方向，位[4:0]对应配置。GPIO 方向有如下索引
  - GPIO 输入模式 = 0x00
  - GPIO 输出模式 = 0x01

Table 186. 给出了所有 GPIO_Mode 的索引和编码

Table 186. GPIO_Mode 的索引和编码

GPIO方向	索引	模式	设置	模式代码
GPIO Input	0x00	GPIO_Mode_AIN	0x00	0x00
		GPIO_Mode_IN_FLOATING	0x04	0x04
		GPIO_Mode_IPD	0x08	0x28
		GPIO_Mode_IPU	0x08	0x48
GPIO Output	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

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 函数GPIO_StructInit

Table 187. 描述了函数 GPIO_StructInit

Table 187. 函数 GPIO_StructInit

函数名	GPIO_StructInit
函数原形	void GPIO_StructInit(GPIO_InitTypeDef* GPIO_InitStruct)
功能描述	把 GPIO_InitStruct 中的每一个参数按缺省值填入
输入参数	GPIO_InitStruct：指向结构 GPIO_InitTypeDef 的指针，待初始化
输出参数	无
返回值	无
先决条件	无
被调用函数	无

Table 188. 给出了 GPIO_InitStruct 各个成员的缺省值

Table 188. 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

例：

```
/* Initialize the GPIO Init Structure parameters */
GPIO_InitTypeDef GPIO_InitStructure;
GPIO_StructInit(&GPIO_InitStructure);
```

10.2.5 函数GPIO_ReadInputDataBit

Table 189. 描述了函数 GPIO_ReadInputDataBit

Table 189. 函数 GPIO_ReadInputDataBit

函数名	GPIO_ReadInputDataBit
函数原形	u8 GPIO_ReadInputDataBit(GPIO_TypeDef* GPIOx, u16 GPIO_Pin)
功能描述	读取指定端口管脚的输入
输入参数 1	GPIOx：x 可以是 A，B，C，D 或者 E，来选择 GPIO 外设
输入参数 2	GPIO_Pin：待读取的端口位 参阅 Section：GPIO_Pin 查阅更多该参数允许取值范围
输出参数	无
返回值	输入端口管脚值
先决条件	无
被调用函数	无

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 函数GPIO_ReadInputData

Table 190. 描述了函数 GPIO_ReadInputData

Table 190. 函数 GPIO_ReadInputData

函数名	GPIO_ReadInputData
函数原形	u16 GPIO_ReadInputData(GPIO_TypeDef* GPIOx)
功能描述	读取指定的 GPIO 端口输入
输入参数	GPIOx：x 可以是 A，B，C，D 或者 E，来选择 GPIO 外设
输出参数	无
返回值	GPIO 输入数据端口值
先决条件	无
被调用函数	无

例：

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

10.2.7 函数GPIO_ReadOutputDataBit

Table 191. 描述了 GPIO_ReadOutputDataBit

Table 191. 函数 GPIO_ReadOutputDataBit

函数名	GPIO_ReadOutputDataBit
函数原形	u8 GPIO_ReadOutputDataBit(GPIO_TypeDef* GPIOx, u16 GPIO_Pin)
功能描述	读取指定端口管脚的输出
输入参数 1	GPIOx：x 可以是 A，B，C，D 或者 E，来选择 GPIO 外设
输入参数 2	GPIO_Pin：待读取的端口位 参阅 Section：GPIO_Pin 查阅更多该参数允许取值范围
输出参数	无
返回值	输出端口管脚值
先决条件	无
被调用函数	无

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 函数GPIO_ReadOutputData

Table 192. 描述了函数 GPIO_ReadOutputData

Table 192. 函数 GPIO_ReadOutputData

函数名	GPIO_ReadOutputData
函数原形	u16 GPIO_ReadOutputData(GPIO_TypeDef* GPIOx)
功能描述	读取指定的 GPIO 端口输出
输入参数	GPIOx：x 可以是 A，B，C，D 或者 E，来选择 GPIO 外设
输出参数	无
返回值	GPIO 输出数据端口值
先决条件	无
被调用函数	无

例：

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

10.2.9 函数GPIO_SetBits

Table 193. 描述了 GPIO_SetBits

Table 193. 函数 GPIO_SetBits

函数名	GPIO_SetBits
函数原形	void GPIO_SetBits(GPIO_TypeDef* GPIOx, u16 GPIO_Pin)
功能描述	设置指定的数据端口位
输入参数 1	GPIOx：x 可以是 A，B，C，D 或者 E，来选择 GPIO 外设
输入参数 2	GPIO_Pin：待设置的端口位 该参数可以取 GPIO_Pin_x(x 可以是 0-15)的任意组合 参阅 Section：GPIO_Pin 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Set the GPIOA port pin 10 and pin 15 */
GPIO_SetBits(GPIOA, GPIO_Pin_10 | GPIO_Pin_15);
```

10.2.10 函数GPIO_ResetBits

Table 194. 描述了 GPIO_ResetBits

Table 194. 函数 GPIO_ResetBits

函数名	GPIO_ResetBits
函数原形	void GPIO_ResetBits(GPIO_TypeDef* GPIOx, u16 GPIO_Pin)
功能描述	清除指定的数据端口位
输入参数 1	GPIOx：x 可以是 A，B，C，D 或者 E，来选择 GPIO 外设
输入参数 2	GPIO_Pin：待清除的端口位 该参数可以取 GPIO_Pin_x(x 可以是 0-15)的任意组合 参阅 Section：GPIO_Pin 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clears the GPIOA port pin 10 and pin 15 */
GPIO_ResetBits(GPIOA, GPIO_Pin_10 | GPIO_Pin_15);
```

10.2.11 函数GPIO_WriteBit

Table 195. 描述了 GPIO_WriteBit

Table 195. 函数 GPIO_WriteBit

函数名	GPIO_WriteBit
函数原形	void GPIO_WriteBit(GPIO_TypeDef* GPIOx, u16 GPIO_Pin, BitAction BitVal)
功能描述	设置或者清除指定的数据端口位
输入参数 1	GPIOx：x 可以是 A，B，C，D 或者 E，来选择 GPIO 外设
输入参数 2	GPIO_Pin：待设置或者清除指的端口位 该参数可以取 GPIO_Pin_x(x 可以是 0-15)的任意组合 参阅 Section：GPIO_Pin 查阅更多该参数允许取值范围
输入参数 3	BitVal: 该参数指定了待写入的值 该参数必须取枚举 BitAction 的其中一个值 Bit_RESET: 清除数据端口位 Bit_SET: 设置数据端口位
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Set the GPIOA port pin 15 */
GPIO_WriteBit(GPIOA, GPIO_Pin_15, Bit_SET);
```

10.2.12 函数GPIO_Write

Table 196. 描述了 GPIO_Write

Table 196. 函数 GPIO_Write

函数名	GPIO_Write
函数原形	void GPIO_Write(GPIO_TypeDef* GPIOx, u16 PortVal)
功能描述	向指定 GPIO 数据端口写入数据
输入参数 1	GPIOx：x 可以是 A，B，C，D 或者 E，来选择 GPIO 外设
输入参数 2	PortVal: 待写入端口数据寄存器的值
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Write data to GPIOA data port */
GPIO_Write(GPIOA, 0x1101);
```

10.2.13 函数GPIO_PinLockConfig

Table 197. 描述了 GPIO_PinLockConfig

Table 197. 函数 GPIO_PinLockConfig

函数名	GPIO_PinLockConfig
函数原形	void GPIO_PinLockConfig(GPIO_TypeDef* GPIOx, u16 GPIO_Pin)
功能描述	锁定 GPIO 管脚设置寄存器
输入参数 1	GPIOx：x 可以是 A，B，C，D 或者 E，来选择 GPIO 外设
输入参数 2	GPIO_Pin：待锁定的端口位 该参数可以取 GPIO_Pin_x(x 可以是 0-15)的任意组合 参阅 Section：GPIO_Pin 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

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

10.2.14 函数GPIO_EventOutputConfig

Table 198. 描述了 GPIO_EventOutputConfig

Table 198. 函数 GPIO_EventOutputConfig

函数名	GPIO_EventOutputConfig
函数原形	void GPIO_EventOutputConfig(u8 GPIO_PortSource, u8 GPIO_PinSource)
功能描述	选择 GPIO 管脚用作事件输出
输入参数 1	GPIO_PortSource: 选择用作事件输出的 GPIO 端口 参阅 Section：GPIO_PortSource 查阅更多该参数允许取值范围
输入参数 2	GPIO_PinSource：事件输出的管脚 该参数可以取 GPIO_PinSourcex(x 可以是 0-15)
输出参数	无
返回值	无
先决条件	无
被调用函数	无

GPIO_PortSource

GPIO_PortSource 用以选择用作事件输出的 GPIO 端口。Table 199. 给出了该参数可取的值

Table 199. GPIO_PortSource 值

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

例：

```
/* Selects the GPIOE pin 5 for EVENT output */
GPIO_EventOutputConfig(GPIO_PortSourceGPIOE, GPIO_PinSource5);
```

10.2.15 函数GPIO_EventOutputCmd

Table 200. 描述了 GPIO_EventOutputCmd

Table 200. 函数 GPIO_EventOutputCmd

函数名	GPIO_EventOutputCmd
函数原形	void GPIO_EventOutputCmd(FunctionalState NewState)
功能描述	使能或者失能事件输出
输入参数 1	NewState: 事件输出的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

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

10.2.16 函数GPIO_PinRemapConfig

Table 201. 描述了 GPIO_ PinRemapConfig

Table 201. 函数 GPIO_ PinRemapConfig

函数名	GPIO_ PinRemapConfig
函数原形	void GPIO_PinRemapConfig(u32 GPIO_Remap, FunctionalState NewState)
功能描述	改变指定管脚的映射
输入参数 1	GPIO_Remap: 选择重映射的管脚 参阅 Section：GPIO_Remap 查阅更多该参数允许取值范围
输入参数 2	NewState: 管脚重映射的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

GPIO_Remap

GPIO_Remap 用以选择用作事件输出的 GPIO 端口。Table 202. 给出了该参数可取的值

Table 202. GPIO_Remap 值

GPIO_Remap	描述
GPIO_Remap_SPI1	SPI1 复用功能映射
GPIO_Remap_I2C1	I2C1 复用功能映射
GPIO_Remap_USART1	USART1 复用功能映射
GPIO_PartialRemap_USART3	USART2 复用功能映射
GPIO_FullRemap_USART3	USART3 复用功能完全映射
GPIO_PartialRemap_TIM1	USART3 复用功能部分映射
GPIO_FullRemap_TIM1	TIM1 复用功能完全映射
GPIO_PartialRemap1_TIM2	TIM2 复用功能部分映射 1
GPIO_PartialRemap2_TIM2	TIM2 复用功能部分映射 2
GPIO_FullRemap_TIM2	TIM2 复用功能完全映射
GPIO_PartialRemap_TIM3	TIM3 复用功能部分映射
GPIO_FullRemap_TIM3	TIM3 复用功能完全映射
GPIO_Remap_TIM4	TIM4 复用功能映射
GPIO_Remap1_CAN	CAN 复用功能映射 1
GPIO_Remap2_CAN	CAN 复用功能映射 2
GPIO_Remap_PD01	PD01 复用功能映射
GPIO_Remap_SWJ_NoJTRST	除 JTRST 外 SWJ 完全使能（JTAG+SW-DP）
GPIO_Remap_SWJ_JTAGDisable	JTAG-DP 失能 + SW-DP 使能
GPIO_Remap_SWJ_Disable	SWJ 完全失能（JTAG+SW-DP）

例：

```
/* I2C1_SCL on PB.08, I2C1_SDA on PB.09 */
GPIO_PinRemapConfig(GPIO_Remap_I2C1, ENABLE);
```

10.2.17 函数GPIO_EXTILineConfig

Table 203. 描述了 GPIO_EXTILineConfig

Table 203. 函数 GPIO_EXTILineConfig

函数名	GPIO_EXTILineConfig
函数原形	void GPIO_EXTILineConfig(u8 GPIO_PortSource, u8 GPIO_PinSource)
功能描述	选择 GPIO 管脚用作外部中断线路
输入参数 1	GPIO_PortSource: 选择用作外部中断线源的 GPIO 端口 参阅 Section：GPIO_PortSource 查阅更多该参数允许取值范围
输入参数 2	GPIO_PinSource：待设置的外部中断线路 该参数可以取 GPIO_PinSourcex(x 可以是 0-15)
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

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

11 内部集成电路（I^2C）

I2C 总线接口连接微控制器和串行 I2C 总线。它提供多主机功能，控制所有 I2C 总线特定的时序、协议、仲裁和定时。支持标准和快速两种模式，同时与 SMBus 2.0 兼容。I2C 总线有多种用途，包括 CRC 码的生成和校验、SMBus(系统管理总线 System Management Bus) PMBus(电源管理总线 Power Management Bus)。

I2C 驱动可以用来通过 I2C 界面发送和接收数据，还可以返回传输操作的状态。

Section 11.1 I2C 寄存器结构描述了固件函数库所使用的数据结构，Section 11.2 固件库函数介绍了函数库里的所有函数。

11.1 I2C寄存器结构

I2C 寄存器结构，***I2C_TypeDeff***，在文件***"stm32f10x_map.h"***中定义如下：

```
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;
```

Table 204.例举了 I2C 所有寄存器

Table 204. 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 自身地址寄存器 1
OAR2	I2C 自身地址寄存器 2
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

2 个 I2C 外设声明于文件***"stm32f10x_map.h"***：

```
...
#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
```

使用 Debug 模式时，初始化指针 ***I2C1, I2C2*** 于文件***"stm32f10x_lib.c"***：

```
...
#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, _I2C2*** 必须在文件***"stm32f10x_conf.h"***中定义如下：

```
...
#define _I2C
#define _I2C1
#define _I2C2
...
```

11.2 I2C库函数

Table 205. 例举了 I2C 的库函数

Table 205. I2C 库函数

函数名	描述
I2C_DeInit	将外设 I2Cx 寄存器重设为缺省值
I2C_Init	根据 I2C_InitStruct 中指定的参数初始化外设 I2Cx 寄存器
I2C_StructInit	把 I2C_InitStruct 中的每一个参数按缺省值填入
I2C_Cmd	使能或者失能 I2C 外设
I2C_DMACmd	使能或者失能指定 I2C 的 DMA 请求
I2C_DMALastTransferCmd	使下一次 DMA 传输为最后一次传输
I2C_GenerateSTART	产生 I2Cx 传输 START 条件
I2C_GenerateSTOP	产生 I2Cx 传输 STOP 条件
I2C_AcknowledgeConfig	使能或者失能指定 I2C 的应答功能
I2C_OwnAddress2Config	设置指定 I2C 的自身地址 2
I2C_DualAddressCmd	使能或者失能指定 I2C 的双地址模式
I2C_GeneralCallCmd	使能或者失能指定 I2C 的广播呼叫功能
I2C_ITConfig	使能或者失能指定的 I2C 中断
I2C_SendData	通过外设 I2Cx 发送一个数据
I2C_ReceiveData	返回通过 I2Cx 最近接收的数据
I2C_Send7bitAddress	向指定的从 I2C 设备传送地址字

I2C_ReadRegister	读取指定的 I2C 寄存器并返回其值
I2C_SoftwareResetCmd	使能或者失能指定 I2C 的软件复位
I2C_SMBusAlertConfig	驱动指定 I2Cx 的 SMBusAlert 管脚电平为高或低
I2C_TransmitPEC	使能或者失能指定 I2C 的 PEC 传输
I2C_PECPositionConfig	选择指定 I2C 的 PEC 位置
I2C_CalculatePEC	使能或者失能指定 I2C 的传输字 PEC 值计算
I2C_GetPEC	返回指定 I2C 的 PEC 值
I2C_ARPCmd	使能或者失能指定 I2C 的 ARP
I2C_StretchClockCmd	使能或者失能指定 I2C 的时钟延展
I2C_FastModeDutyCycleConfig	选择指定 I2C 的快速模式占空比
I2C_GetLastEvent	返回最近一次 I2C 事件
I2C_CheckEvent	检查最近一次 I2C 事件是否是输入的事件
I2C_GetFlagStatus	检查指定的 I2C 标志位设置与否
I2C_ClearFlag	清除 I2Cx 的待处理标志位
I2C_GetITStatus	检查指定的 I2C 中断发生与否
I2C_ClearITPendingBit	清除 I2Cx 的中断待处理位

11.2.1 函数I2C_DeInit

Table 206. 描述了函数 I2C_DeInit

Table 206. 函数 I2C_DeInit

函数名	I2C_DeInit
函数原形	void I2C_DeInit(I2C_TypeDef* I2Cx)
功能描述	将外设 I2Cx 寄存器重设为缺省值
输入参数	I2Cx：x 可以是 1 或者 2，来选择 I2C 外设
输出参数	无
返回值	无
先决条件	无
被调用函数	RCC_APB1PeriphClockCmd().

例：

```
/* Deinitialize I2C2 interface*/
I2C_DeInit(I2C2);
```

11.2.2 函数I2C_ Init

Table 207. 描述了函数 I2C_Init

Table 207. 函数 I2C_Init

函数名	I2C_Init
函数原形	void I2C_Init(I2C_TypeDef* I2Cx, I2C_InitTypeDef* I2C_InitStruct)
功能描述	根据 I2C_InitStruct 中指定的参数初始化外设 I2Cx 寄存器
输入参数 1	I2Cx：x 可以是 1 或者 2，来选择 I2C 外设
输入参数 2	I2C_InitStruct：指向结构 I2C_InitTypeDef 的指针，包含了外设 GPIO 的配置信息 参阅 Section：I2C_InitTypeDef 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

I2C_InitTypeDef structure

I2C_InitTypeDef 定义于文件“stm32f10x_i2c.h”：

```
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 用以设置 I2C 的模式。Table 208. 给出了该参数可取的值

Table 208. I2C_Mode 值

I2C_Mode	描述
I2C_Mode_I2C	设置 I2C 为 I2C 模式
I2C_Mode_SMBusDevice	设置 I2C 为 SMBus 设备模式
I2C_Mode_SMBusHost	设置 I2C 为 SMBus 主控模式

I2C_DutyCycle

I2C_DutyCycle 用以设置 I2C 的占空比。Table 209. 给出了该参数可取的值

Table 209. I2C_DutyCycle 值

I2C_DutyCycle	描述
I2C_DutyCycle_16_9	I2C 快速模式 Tlow / Thigh = 16/9
I2C_DutyCycle_2	I2C 快速模式 Tlow / Thigh = 2

注意：该参数只有在 I2C 工作在快速模式（时钟工作频率高于 100KHz）下才有意义。

I2C_OwnAddress1

该参数用来设置第一个设备自身地址，它可以是一个 7 位地址或者一个 10 位地址。

I2C_Ack

I2C_Ack 使能或者失能应答（ACK），Table 210. 给出了该参数可取的值

Table 210. I2C_Ack 值

I2C_Ack	描述
I2C_Ack_Enable	使能应答（ACK）
I2C_Ack_Disable	失能应答（ACK）

I2C_AcknowledgedAddress

I2C_AcknowledgedAddres 定义了应答 7 位地址还是 10 位地址。Table 211. 给出了该参数可取的值

Table 211. I2C_AcknowledgedAddres 值

I2C_AcknowledgedAddres	描述
I2C_AcknowledgeAddress_7bit	应答 7 位地址
I2C_AcknowledgeAddress_10bit	应答 10 位地址

I2C_ClockSpeed

该参数用来设置时钟频率，这个值不能高于 400KHz。

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 函数I2C_ StructInit

Table 212. 描述了函数 I2C_StructInit

Table 212. 函数 I2C_StructInit

函数名	I2C_StructInit
函数原形	void I2C_StructInit(I2C_InitTypeDef* I2C_InitStruct)
功能描述	把 I2C_InitStruct 中的每一个参数按缺省值填入
输入参数	I2C_InitStruct：指向结构 I2C_InitTypeDef 的指针，待初始化
输出参数	无
返回值	无
先决条件	无
被调用函数	无

Table 213. 给出了 I2C_InitStruct 各个成员的缺省值

Table 213. 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_AcknowledgedAddres	I2C_AcknowledgedAddress_7bit
I2C_ ClockSpeed	5000

例：

```
/* Initialize an I2C_InitTypeDef structure */
I2C_InitTypeDef I2C_InitStructure;
I2C_StructInit(&I2C_InitStructure);
```

11.2.4 函数I2C_ Cmd

Table 214. 描述了函数 I2C_ Cmd

Table 214. 函数 I2C_ Cmd

函数名	I2C_ Cmd
函数原形	void I2C_Cmd(I2C_TypeDef* I2Cx, FunctionalState NewState)
功能描述	使能或者失能 I2C 外设
输入参数 1	I2Cx：x 可以是 1 或者 2，来选择 I2C 外设
输入参数 2	NewState: 外设 I2Cx 的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable I2C1 peripheral */
I2C_Cmd(I2C1, ENABLE);
```

11.2.5 函数I2C_ DMACmd

Table 215. 描述了函数 I2C_ DMACmd

Table 215. 函数 I2C_ DMACmd

函数名	I2C_ DMACmd
函数原形	I2C_DMACmd(I2C_TypeDef* I2Cx, FunctionalState NewState)
功能描述	使能或者失能指定 I2C 的 DMA 请求
输入参数 1	I2Cx：x 可以是 1 或者 2，来选择 I2C 外设
输入参数 2	NewState: I2Cx DMA 传输的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable I2C2 DMA transfer */
I2C_DMACmd(I2C2, ENABLE);
```

11.2.6 函数I2C_ DMALastTransferCmd

Table 216. 描述了函数 I2C_ DMALastTransferCmd

Table 216. 函数 I2C_ DMALastTransferCmd

函数名	I2C_ DMALastTransferCmd
函数原形	I2C_DMALastTransferCmd(I2C_TypeDef* I2Cx, FunctionalState NewState)
功能描述	使下一次 DMA 传输为最后一次传输
输入参数 1	I2Cx：x 可以是 1 或者 2，来选择 I2C 外设
输入参数 2	NewState: I2Cx DMA 最后一次传输的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Specify that the next I2C2 DMA transfer is the last one */
I2C_DMALastTransferCmd(I2C2, ENABLE);
```

11.2.7 函数I2C_ GenerateSTART

Table 217. 描述了函数 I2C_ GenerateSTART

Table 217. 函数 I2C_ GenerateSTART

函数名	I2C_ GenerateSTART
函数原形	void I2C_GenerateSTART(I2C_TypeDef* I2Cx, FunctionalState NewState)
功能描述	产生 I2Cx 传输 START 条件
输入参数 1	I2Cx：x 可以是 1 或者 2，来选择 I2C 外设
输入参数 2	NewState: I2Cx START 条件的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Generate a START condition on I2C1 */
I2C_GenerateSTART(I2C1, ENABLE);
```

11.2.8 函数I2C_ GenerateSTOP

Table 218. 描述了函数 I2C_ GenerateSTOP

Table 218. 函数 I2C_ GenerateSTOP

函数名	I2C_ GenerateSTOP
函数原形	void I2C_GenerateSTOP(I2C_TypeDef* I2Cx, FunctionalState NewState)
功能描述	产生 I2Cx 传输 STOP 条件
输入参数 1	I2Cx：x 可以是 1 或者 2，来选择 I2C 外设
输入参数 2	NewState: I2Cx STOP 条件的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

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

11.2.9 函数I2C_ AcknowledgeConfig

Table 219. 描述了函数 I2C_ AcknowledgeConfig

Table 219. 函数 I2C_ AcknowledgeConfig

函数名	I2C_ AcknowledgeConfig
函数原形	void I2C_AcknowledgeConfig(I2C_TypeDef* I2Cx, FunctionalState NewState)
功能描述	使能或者失能指定 I2C 的应答功能
输入参数 1	I2Cx：x 可以是 1 或者 2，来选择 I2C 外设
输入参数 2	NewState: I2Cx 应答的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

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

11.2.10 函数I2C_ OwnAddress2Config

Table 220. 描述了函数 I2C_ OwnAddress2Config

Table 220. 函数 I2C_ OwnAddress2Config

函数名	I2C_ OwnAddress2Config
函数原形	void I2C_OwnAddress2Config(I2C_TypeDef* I2Cx, u8 Address)
功能描述	设置指定 I2C 的自身地址 2
输入参数 1	I2Cx：x 可以是 1 或者 2，来选择 I2C 外设
输入参数 2	Address: 指定的 7 位 I2C 自身地址 2
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Set the I2C1 own address2 to 0x38 */
I2C_OwnAddress2Config(I2C1, 0x38);
```

11.2.11 函数I2C_ DualAddressCmd

Table 221. 描述了函数 I2C_ DualAddressCmd

Table 221. 函数 I2C_ DualAddressCmd

函数名	I2C_ DualAddressCmd
函数原形	void I2C_DualAddressCmd(I2C_TypeDef* I2Cx, FunctionalState NewState)
功能描述	使能或者失能指定 I2C 的双地址模式
输入参数 1	I2Cx：x 可以是 1 或者 2，来选择 I2C 外设
输入参数 2	NewState: I2Cx 双地址模式的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable the I2C2 dual addressing mode*/
I2C_DualAdressCmd(I2C2, ENABLE);
```

11.2.12 函数I2C_ GeneralCallCmd

Table 222. 描述了函数 I2C_ GeneralCallCmd

Table 222. 函数 I2C_ GeneralCallCmd

函数名	I2C_ GeneralCallCmd
函数原形	void I2C_GeneralCallCmd(I2C_TypeDef* I2Cx, FunctionalState NewState)
功能描述	使能或者失能指定 I2C 的广播呼叫功能
输入参数 1	I2Cx：x 可以是 1 或者 2，来选择 I2C 外设
输入参数 2	NewState: I2Cx 广播呼叫的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable the I2C1 general call feature */
I2C_GeneralCallCmd(I2C1, ENABLE);
```

11.2.13 函数I2C_ ITConfig

Table 223. 描述了函数 I2C_ITConfig

Table 223. 函数 I2C_ITConfig

函数名	I2C_ITConfig
函数原形	void I2C_ITConfig(I2C_TypeDef* I2Cx, u16 I2C_IT, FunctionalState NewState)
功能描述	使能或者失能指定的 I2C 中断
输入参数 1	I2Cx：x 可以是 1 或者 2，来选择 I2C 外设
输入参数 2	I2C_IT：待使能或者失能的 I2C 中断源 参阅 Section：I2C_IT 查阅更多该参数允许取值范围
输入参数 3	NewState：I2Cx 中断的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

I2C_IT

输入参数 I2C_IT 使能或者失能 I2C 的中断。可以取下表的一个或者多个取值的组合作为该参数的值。

Table 224. I2C_IT 值

I2C_IT	描述
I2C_IT_BUF	缓存中断屏蔽
I2C_IT_EVT	事件中断屏蔽
I2C_IT_ERR	错误中断屏蔽

例：

```
/* Enable I2C2 event and buffer interrupts */
I2C_ITConfig(I2C2, I2C_IT_BUF | I2C_IT_EVT, ENABLE);
```

11.2.14 函数I2C_ SendData

Table 225. 描述了函数 I2C_ SendData

Table 225. 函数 I2C_ SendData

函数名	I2C_ SendData
函数原形	void I2C_SendData(I2C_TypeDef* I2Cx, u8 Data)
功能描述	通过外设 I2Cx 发送一个数据
输入参数 1	I2Cx：x 可以是 1 或者 2，来选择 I2C 外设
输入参数 2	Data: 待发送的数据
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Transmit 0x5D byte on I2C2 */
I2C_SendData(I2C2, 0x5D);
```

11.2.15 函数I2C_ ReceiveData

Table 226. 描述了函数 I2C_ ReceiveData

Table 226. 函数 I2C_ReceiveData

函数名	I2C_ ReceiveData
函数原形	u8 I2C_ReceiveData(I2C_TypeDef* I2Cx)
功能描述	返回通过 I2Cx 最近接收的数据
输入参数	I2Cx：x 可以是 1 或者 2，来选择 I2C 外设
输出参数	无
返回值	接收到的字
先决条件	无
被调用函数	无

例：

```
/* Read the received byte on I2C1 */
u8 ReceivedData;
ReceivedData = I2C_ReceiveData(I2C1);
```

11.2.16 函数I2C_ Send7bitAddress

Table 227. 描述了函数 I2C_ Send7bitAddress

Table 227. 函数 I2C_ Send7bitAddress

函数名	I2C_ Send7bitAddress
函数原形	void I2C_Send7bitAddress(I2C_TypeDef* I2Cx, u8 Address, u8 I2C_Direction)
功能描述	向指定的从 I2C 设备传送地址字
输入参数 1	I2Cx：x 可以是 1 或者 2，来选择 I2C 外设
输入参数 2	Address: 待传输的从 I2C 地址
输入参数 3	I2C_Direction：设置指定的 I2C 设备工作为发射端还是接收端 参阅 Section：I2C_Direction 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

I2C_Direction

该参数设置 I2C 界面为发送端模式或者接收端模式（见 Table 228.）。

Table 228. I2C_Direction 值

I2C_Direction	描述
I2C_Direction_Transmitter	选择发送方向
I2C_Direction_Receiver	选择接收方向

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 函数I2C_ ReadRegister

Table 229. 描述了函数 I2C_ ReadRegister

Table 229. 函数 I2C_ ReadRegister

函数名	I2C_ ReadRegister
函数原形	u16 I2C_ReadRegister(I2C_TypeDef* I2Cx, u8 I2C_Register)
功能描述	读取指定的 I2C 寄存器并返回其值
输入参数 1	I2Cx：x 可以是 1 或者 2，来选择 I2C 外设
输入参数 2	I2C_Register：待读取的 I2C 寄存器 参阅 Section：I2C_Register 查阅更多该参数允许取值范围
输出参数	无
返回值	被读取的寄存器值 [1.]
先决条件	无
被调用函数	无

1. 读取寄存器可能会清除某些标志位

I2C_Register

Table 230. 给出了所有可以被函数 I2C_ ReadRegister 读取的寄存器列表

Table 230. I2C_Register 值

I2C_Register	描述
I2C_Register_CR1	选择读取寄存器 I2C_CR1
I2C_Register_CR2	选择读取寄存器 I2C_CR2
I2C_Register_OAR1	选择读取寄存器 I2C_OAR1
I2C_Register_OAR2	选择读取寄存器 I2C_OAR2
I2C_Register_DR	选择读取寄存器 I2C_DR
I2C_Register_SR1	选择读取寄存器 I2C_SR1
I2C_Register_SR2	选择读取寄存器 I2C_SR2
I2C_Register_CCR	选择读取寄存器 I2C_ CCR
I2C_Register_TRISE	选择读取寄存器 I2C_TRISE

例：

```
/* Return the I2C_CR1 register value of I2C2 peripheral */
u16 RegisterValue;
RegisterValue = I2C_ReadRegister(I2C2, I2C_Register_CR1);
```

11.2.18 函数I2C_ SoftwareResetCmd

Table 231. 描述了函数 I2C_ SoftwareResetCmd

Table 231. 函数 I2C_ SoftwareResetCmd

函数名	I2C_ SoftwareResetCmd
函数原形	I2C_SoftwareResetCmd(I2C_TypeDef* I2Cx, FunctionalState NewState)
功能描述	使能或者失能指定 I2C 的软件复位
输入参数 1	I2Cx：x 可以是 1 或者 2，来选择 I2C 外设
输入参数 2	NewState：I2Cx 软件复位的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

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

11.2.19 函数I2C_ SMBusAlertConfig

Table 232. 描述了函数 I2C_ SMBusAlertConfig

Table 232. 函数 I2C_ SMBusAlertConfig

函数名	I2C_ SMBusAlertConfig
函数原形	void I2C_SMBusAlertConfig(I2C_TypeDef* I2Cx, u16 I2C_SMBusAlert)
功能描述	驱动指定 I2Cx 的 SMBusAlert 管脚电平为高或低
输入参数 1	I2Cx：x 可以是 1 或者 2，来选择 I2C 外设
输入参数 2	I2C_ SMBusAlert：SMBusAlert 管脚电平 参阅 Section：I2C_ SMBusAlert 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

I2C_SMBusAlert

I2C_ SMBusAlert 用以设置 SMBusAlert 管脚的有效电平。Table 233. 给出了该参数可取的值

Table 233. I2C_ SMBusAlert 值

I2C_ SMBusAlert	描述
I2C_SMBusAlert_Low	驱动 SMBusAlert 管脚电平为高
I2C_SMBusAlert_High	驱动 SMBusAlert 管脚电平为低

例：

```
/* Let the I2C2 SMBusAlert pin High */
I2C_SMBusAlertConfig(I2C2, I2C_SMBusAlert_High);
```

11.2.20 函数I2C_ TransmitPEC

Table 234. 描述了函数 I2C_ TransmitPEC

Table 234. 函数 I2C_ TransmitPEC

函数名	I2C_ TransmitPEC
函数原形	I2C_TransmitPEC(I2C_TypeDef* I2Cx, FunctionalState NewState)
功能描述	使能或者失能指定 I2C 的 PEC 传输
输入参数 1	I2Cx：x 可以是 1 或者 2，来选择 I2C 外设
输入参数 2	NewState：I2CxPEC 传输的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

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

11.2.21 函数I2C_ PECPositionConfig

Table 235. 描述了函数 I2C_ PECPositionConfig

Table 235. 函数 I2C_ PECPositionConfig

函数名	I2C_ PECPositionConfig
函数原形	void I2C_PECPositionConfig(I2C_TypeDef* I2Cx, u16 I2C_PECPosition)
功能描述	选择指定 I2C 的 PEC 位置
输入参数 1	I2Cx：x 可以是 1 或者 2，来选择 I2C 外设
输入参数 2	I2C_ PECPosition：PEC 位置 参阅 Section：I2C_ PECPositiont 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

I2C_PECPosition

I2C_ PECPosition 用以设置 PEC 位置。Table 235. 给出了该参数可取的值

Table 236. I2C_ PECPosition 值

I2C_ PECPosition	描述
I2C_PECPosition_Next	PEC 位提示下一字为 PEC
I2C_PECPosition_Current	PEC 位提示当前字为 PEC

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 函数I2C_ CalculatePEC

Table 237. 描述了函数 I2C_ CalculatePEC

Table 237. 函数 I2C_ CalculatePEC

函数名	I2C_ CalculatePEC
函数原形	void I2C_CalculatePEC(I2C_TypeDef* I2Cx, FunctionalState NewState)
功能描述	使能或者失能指定 I2C 的传输字 PEC 值计算
输入参数 1	I2Cx：x 可以是 1 或者 2，来选择 I2C 外设
输入参数 2	NewState：I2Cx 传输字 PEC 值计算的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

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

11.2.23 函数I2C_ GetPEC

Table 238. 描述了函数 I2C_ GetPEC

Table 238. 函数 I2C_ GetPEC

函数名	I2C_ GetPEC
函数原形	u8 I2C_GetPEC(I2C_TypeDef* I2Cx)
功能描述	返回指定 I2C 的 PEC 值
输入参数	I2Cx：x 可以是 1 或者 2，来选择 I2C 外设
输出参数	无
返回值	PEC 值
先决条件	无
被调用函数	无

例：

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

11.2.24 函数I2C_ ARPCmd

Table 239. 描述了函数 I2C_ ARPCmd

Table 239. 函数 I2C_ ARPCmd

函数名	I2C_ ARPCmd
函数原形	void I2C_ARPCmd(I2C_TypeDef* I2Cx, FunctionalState NewState)
功能描述	使能或者失能指定 I2C 的 ARP
输入参数 1	I2Cx：x 可以是 1 或者 2，来选择 I2C 外设
输入参数 2	NewState：I2Cx ARP 的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable the I2C1 ARP feature */
I2C_ARPCmd(I2C1, ENABLE);
```

11.2.25 函数I2C_ StretchClockCmd

Table 240. 描述了函数 I2C_ StretchClockCmd

Table 240. 函数 I2C_ StretchClockCmd

函数名	I2C_ StretchClockCmd
函数原形	void I2C_StretchClockCmd(I2C_TypeDef* I2Cx, FunctionalState NewState)
功能描述	使能或者失能指定 I2C 的时钟延展
输入参数 1	I2Cx：x 可以是 1 或者 2，来选择 I2C 外设
输入参数 2	NewState：I2Cx 时钟延展的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable the I2C2 clock stretching */
I2C_StretchClockCmd(I2C2, ENABLE);
```

11.2.26 函数I2C_ FastModeDutyCycleConfig

Table 241. 描述了函数 I2C_ FastModeDutyCycleConfig

Table 241. 函数 I2C_ FastModeDutyCycleConfig

函数名	I2C_ FastModeDutyCycleConfig
函数原形	void I2C_FastModeDutyCycleConfig(I2C_TypeDef* I2Cx, u16 I2C_DutyCycle)
功能描述	选择指定 I2C 的快速模式占空比
输入参数 1	I2Cx：x 可以是 1 或者 2，来选择 I2C 外设
输入参数 2	I2C_DutyCycle：快速模式占空比 参阅 Section：I2C_DutyCycle 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

I2C_DutyCycle

I2C_DutyCycle 用以设置 I2C 快速模式的占空比。Table 242. 给出了该参数可取的值

Table 242. I2C_DutyCycle 值

I2C_DutyCycle	描述
I2C_DutyCycle_16_9	I2C 快速模式 Tlow / Thigh = 16/9
I2C_DutyCycle_2	I2C 快速模式 Tlow / Thigh = 2

例：

```
/* Set the fast mode duty cyle to 16/9 for I2C2 */
I2C_FastModeDutyCycleConfig(I2C2, I2C_DutyCycle_16_9);
```

11.2.27 函数I2C_ GetLastEvent

Table 243. 描述了函数 I2C_ GetLastEvent

Table 243. 函数 I2C_ GetLastEvent

函数名	I2C_ GetLastEvent
函数原形	u32 I2C_GetLastEvent(I2C_TypeDef* I2Cx)
功能描述	返回最近一次 I2C 事件
输入参数	I2Cx：x 可以是 1 或者 2，来选择 I2C 外设
输出参数	无
返回值	最近一次 I2C 事件
先决条件	无
被调用函数	无

例：

```
/* Get last I2C1 event */
u32 Event;
Event = I2C_GetLastEvent(I2C1);
```

11.2.28 函数 I2C_ CheckEvent

Table 244. 描述了函数 I2C_ CheckEvent

Table 244. 函数 I2C_ CheckEvent

函数名	I2C_ CheckEvent
函数原形	ErrorStatus I2C_CheckEvent(I2C_TypeDef* I2Cx, u32 I2C_EVENT)
功能描述	检查最近一次 I2C 事件是否是输入的事件
输入参数 1	I2Cx：x 可以是 1 或者 2，来选择 I2C 外设
输入参数 3	I2C_Event：待检查的事件 参阅 Section：I2C_Event 查阅更多该参数允许取值范围
输出参数	无
返回值	ErrorStatus 枚举值： SUCCESS：最近一次 I2C 事件是 I2C_Event ERROR：最近一次 I2C 事件不是 I2C_Event
先决条件	无
被调用函数	无

I2C_Event

Table 245.列举了所有可以被函数 I2C_ CheckEvent 检查的事件。

Table 245. I2C_Event 值

I2C_Event	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

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 函数I2C_ GetFlagStatus

Table 246. 描述了函数 I2C_ GetFlagStatus

Table 246. 函数 I2C_ GetFlagStatus

函数名	I2C_ GetFlagStatus
函数原形	FlagStatus I2C_GetFlagStatus(I2C_TypeDef* I2Cx, u32 I2C_FLAG)
功能描述	检查指定的 I2C 标志位设置与否
输入参数 1	I2Cx：x 可以是 1 或者 2，来选择 I2C 外设
输入参数 2	I2C_FLAG：待检查的 I2C 标志位 参阅 Section：I2C_FLAG 查阅更多该参数允许取值范围
输出参数	无
返回值	I2C_FLAG 的新状态 [1.]
先决条件	无
被调用函数	无

1. 读取寄存器可能会清除某些标志位

I2C_FLAG

Table 247. 给出了所有可以被函数 I2C_ GetFlagStatus 检查的标志位列表

Table 247. I2C_FLAG 值

I2C_FLAG	描述
I2C_FLAG_DUALF	双标志位（从模式）
I2C_FLAG_SMBHOST	SMBus 主报头（从模式）
I2C_FLAG_SMBDEFAULT	SMBus 缺省报头（从模式）
I2C_FLAG_GENCALL	广播报头标志位（从模式）
I2C_FLAG_TRA	发送/接收标志位
I2C_FLAG_BUSY	总线忙标志位
I2C_FLAG_MSL	主/从标志位
I2C_FLAG_SMBALERT	SMBus 报警标志位
I2C_FLAG_TIMEOUT	超时或者 Tlow 错误标志位
I2C_FLAG_PECERR	接收 PEC 错误标志位
I2C_FLAG_OVR	溢出/不足标志位（从模式）
I2C_FLAG_AF	应答错误标志位
I2C_FLAG_ARLO	仲裁丢失标志位（主模式）
I2C_FLAG_BERR	总线错误标志位
I2C_FLAG_TXE	数据寄存器空标志位（发送端）
I2C_FLAG_RXNE	数据寄存器非空标志位（接收端）
I2C_FLAG_STOPF	停止探测标志位（从模式）
I2C_FLAG_ADD10	10 位报头发送（主模式）
I2C_FLAG_BTF	字传输完成标志位
I2C_FLAG_ADDR	地址发送标志位（主模式）“ADSL” 地址匹配标志位（从模式）“ENDAD”
I2C_FLAG_SB	起始位标志位（主模式）

注意：只有位[27：0]被函数 I2C_ GetFlagStatus 用来返回指定的标志位状态。值对应经计算的寄存器中的

标志位位置，该寄存器包含 2 个 I2C 状态寄存器 I2C_SR1 和 I2C_SR2。

例：

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

11.2.30 函数I2C_ ClearFlag

Table 248. 描述了函数 I2C_ ClearFlag

Table 248. 函数 I2C_ ClearFlag

函数名	I2C_ ClearFlag
函数原形	void I2C_ClearFlag(I2C_TypeDef* I2Cx, u32 I2C_FLAG)
功能描述	清除 I2Cx 的待处理标志位
输入参数 1	I2Cx：x 可以是 1 或者 2，来选择 I2C 外设
输入参数 2	I2C_FLAG：待清除的 I2C 标志位 参阅 Section：I2C_FLAG 查阅更多该参数允许取值范围 注意：标志位 DUALF, SMBHOST, SMBDEFAULT, GENCALL, TRA, BUSY,MSL, TXE 和 RXNE 不能被本函数清除
输出参数	无
返回值	无
先决条件	无
被调用函数	无

I2C_FLAG

Table 249. 给出了所有可以被函数 I2C_ ClearFlag 清除的标志位列表

Table 249. I2C_FLAG 值

I2C_FLAG	描述
I2C_FLAG_SMBALERT	SMBus 报警标志位
I2C_FLAG_TIMEOUT	超时或者 Tlow 错误标志位
I2C_FLAG_PECERR	接收 PEC 错误标志位
I2C_FLAG_OVR	溢出/不足标志位（从模式）
I2C_FLAG_AF	应答错误标志位
I2C_FLAG_ARLO	仲裁丢失标志位（主模式）
I2C_FLAG_BERR	总线错误标志位
I2C_FLAG_STOPF	停止探测标志位（从模式）
I2C_FLAG_ADD10	10 位报头发送（主模式）
I2C_FLAG_BTF	字传输完成标志位
I2C_FLAG_ADDR	地址发送标志位（主模式）“ADSL” 地址匹配标志位（从模式）“ENDAD”
I2C_FLAG_SB	起始位标志位（主模式）

例：

```
/* Clear the Stop detection flag on I2C2 */
I2C_ClearFlag(I2C2, I2C_FLAG_STOPF);
```

11.2.31 函数I2C_ GetITStatus

Table 250. 描述了函数 I2C_ GetITStatus

Table 250. 函数 I2C_ GetITStatus

函数名	I2C_ GetITStatus
函数原形	ITStatus I2C_GetITStatus(I2C_TypeDef* I2Cx, u32 I2C_IT)
功能描述	检查指定的 I2C 中断发生与否
输入参数 1	I2Cx：x 可以是 1 或者 2，来选择 I2C 外设
输入参数 2	I2C_IT：待检查的 I2C 中断源 参阅 Section：I2C_IT 查阅更多该参数允许取值范围
输出参数	无
返回值	I2C_IT 的新状态（SET 或者 RESET）1.
先决条件	无
被调用函数	无

1. 读取寄存器可能会清除某些标志位

I2C_IT

Table 251. 给出了所有可以被函数 I2C_ GetITStatus 检查的中断标志位列表

Table 251. I2C_IT 值

I2C_IT	描述
I2C_IT_SMBALERT	SMBus 报警标志位
I2C_IT_TIMEOUT	超时或者 Tlow 错误标志位
I2C_IT_PECERR	接收 PEC 错误标志位
I2C_IT_OVR	溢出/不足标志位（从模式）
I2C_IT_AF	应答错误标志位
I2C_IT_ARLO	仲裁丢失标志位（主模式）
I2C_IT_BERR	总线错误标志位
I2C_IT_STOPF	停止探测标志位（从模式）
I2C_IT_ADD10	10 位报头发送（主模式）
I2C_IT_BTF	字传输完成标志位
I2C_IT_ADDR	地址发送标志位（主模式）“ADSL” 地址匹配标志位（从模式）“ENDAD”
I2C_IT_SB	起始位标志位（主模式）

例：

```
/* Return the I2C_IT_OVR flag state of I2C1 peripheral */
ITstatus Status;
Status = I2C_GetITStatus(I2C1, I2C_IT_OVR);
```

11.2.32 函数I2C_ ClearITPendingBit

Table 252. 描述了函数 I2C_ ClearITPendingBit

Table 252. 函数 I2C_ ClearITPendingBit

函数名	I2C_ ClearITPendingBit
函数原形	void I2C_ClearITPendingBit(I2C_TypeDef* I2Cx, u32 I2C_IT)
功能描述	清除 I2Cx 的中断待处理位
输入参数 1	I2Cx：x 可以是 1 或者 2，来选择 I2C 外设
输入参数 2	I2C_IT：待检查的 I2C 中断源 参阅 Section：I2C_IT 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

I2C_IT

Table 253. 给出了所有可以被函数 I2C_ ClearITPendingBit 清除的中断待处理位列表

Table 253. I2C_IT 值

I2C_IT	描述
I2C_IT_SMBALERT	SMBus 报警标志位
I2C_IT_TIMEOUT	超时或者 Tlow 错误标志位
I2C_IT_PECERR	接收 PEC 错误标志位
I2C_IT_OVR	溢出/不足标志位（从模式）
I2C_IT_AF	应答错误标志位
I2C_IT_ARLO	仲裁丢失标志位（主模式）
I2C_IT_BERR	总线错误标志位
I2C_IT_STOPF	停止探测标志位（从模式）
I2C_IT_ADD10	10 位报头发送（主模式）
I2C_IT_BTF	字传输完成标志位
I2C_IT_ADDR	地址发送标志位（主模式）“ADSL” 地址匹配标志位（从模式）“ENDAD”
I2C_IT_SB	起始位标志位（主模式）

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）用来解决应软件或者硬件引起的处理器故障。它也可以在停止（Stop）模式和待命（Standby）模式下工作。

Section 12.1 IWDG 寄存器结构描述了固件函数库所使用的数据结构，Section 12.2 固件库函数介绍了函数库里的所有函数。

12.1 IWDG寄存器结构

IWDG 寄存器结构，***IWDG_TypeDeff***，在文件***"stm32f10x_map.h"***中定义如下：

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

Table 254.例举了 IWDG 所有寄存器

Table 254. IWDG 寄存器

寄存器	描述
KR	IWDG 键值寄存器
PR	IWDG 预分频寄存器
RLR	IWDG 重装载寄存器
SR	IWDG 状态寄存器

IWDG 外设声明于文件"stm32f10x_map.h"：

```
#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
```

使用 Debug 模式时，初始化指针 ***IWDG*** 于文件***"stm32f10x_lib.c"***：

```
#ifdef _IWDG
IWDG = (IWDG_TypeDef *) IWDG_BASE;
#endif /*_IWDG */
```

为了访问 IWDG 寄存器， ***_IWDG*** 必须在文件***"stm32f10x_conf.h"***中定义如下：

```
#define _IWDG
```

12.2 IWDG库函数

Table 255. 例举了 IWDG 的库函数

Table 255. IWDG 库函数

函数名	描述
IWDG_WriteAccessCmd	使能或者失能对寄存器 IWDG_PR 和 IWDG_RLR 的写操作
IWDG_SetPrescaler	设置 IWDG 预分频值
IWDG_SetReload	设置 IWDG 重装载值
IWDG_ReloadCounter	按照 IWDG 重装载寄存器的值重装载 IWDG 计数器
IWDG_Enable	使能 IWDG
IWDG_GetFlagStatus	检查指定的 IWDG 标志位被设置与否

12.2.1 函数IWDG_WriteAccessCmd

Table 256. 描述了函数 IWDG_WriteAccessCmd

Table 256. 函数 IWDG_WriteAccessCmd

函数名	IWDG_WriteAccessCmd
函数原形	void IWDG_WriteAccessCmd(u16 IWDG_WriteAccess)
功能描述	使能或者失能对寄存器 IWDG_PR 和 IWDG_RLR 的写操作
输入参数	IWDG_WriteAccess：对寄存器 IWDG_PR 和 IWDG_RLR 的写操作的新状态 参阅 Section：IWDG_WriteAccess 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

IWDG_WriteAccess

该参数使能或者失能对寄存器 IWDG_PR 和 IWDG_RLR 的写操作（见 Table. 257）。

Table 257. IWDG_WriteAccess 值

IWDG_WriteAccess	描述
IWDG_WriteAccess_Enable	使能对寄存器 IWDG_PR 和 IWDG_RLR 的写操作
IWDG_WriteAccess_Disable	失能对寄存器 IWDG_PR 和 IWDG_RLR 的写操作

例：

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

12.2.2 函数IWDG_SetPrescaler

Table 258. 描述了函数 IWDG_SetPrescaler

Table 258. 函数 IWDG_SetPrescaler

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 预分频值 参阅 Section：IWDG_Prescaler 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

IWDG_Prescaler

该参数设置 IWDG 预分频值（见 Table. 259）。

Table 259. IWDG_Prescaler 值

IWDG_Prescaler	描述
IWDG_Prescaler_4	设置 IWDG 预分频值为 4
IWDG_Prescaler_8	设置 IWDG 预分频值为 8
IWDG_Prescaler_16	设置 IWDG 预分频值为 16
IWDG_Prescaler_32	设置 IWDG 预分频值为 32
IWDG_Prescaler_64	设置 IWDG 预分频值为 64
IWDG_Prescaler_128	设置 IWDG 预分频值为 128
IWDG_Prescaler_256	设置 IWDG 预分频值为 256

例：

```
/* Set IWDG prescaler to 8 */
IWDG_SetPrescaler(IWDG_Prescaler_8);
```

12.2.3 函数IWDG_SetReload

Table 260. 描述了函数 IWDG_SetReload

Table 260. 函数 IWDG_ SetReload

函数名	IWDG_ SetReload
函数原形	void IWDG_SetReload(u16 Reload)
功能描述	设置 IWDG 重装载值
输入参数	IWDG_Reload：IWDG 重装载值 该参数允许取值范围为 0 – 0x0FFF
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

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

12.2.4 函数IWDG_ReloadCounter

Table 261. 描述了函数 IWDG_ReloadCounter

Table 261. 函数 IWDG_ReloadCounter

函数名	IWDG_ReloadCounter
函数原形	void IWDG_ReloadCounter(void)
功能描述	按照 IWDG 重装载寄存器的值重装载 IWDG 计数器
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

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

12.2.5 函数IWDG_Enable

Table 262. 描述了函数 IWDG_Enable

Table 262. 函数 IWDG_Enable

函数名	IWDG_Enable
函数原形	void IWDG_Enable(void)
功能描述	使能 IWDG
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable IWDG */
IWDG_Enable();
```

12.2.6 函数IWDG_GetFlagStatus

Table 263. 描述了函数 IWDG_GetFlagStatus

Table 263. 函数 IWDG_GetFlagStatus

函数名	IWDG_GetFlagStatus
函数原形	FlagStatus IWDG_GetFlagStatus(u16 IWDG_FLAG)
功能描述	检查指定的 IWDG 标志位被设置与否
输入参数	IWDG_FLAG：待检查的 I2C 标志位 参阅 Section：IWDG_FLAG 查阅更多该参数允许取值范围
输出参数	无
返回值	IWDG_FLAG 的新状态（SET 或者 RESET）
先决条件	无
被调用函数	无

IWDG_FLAG

Table 264. 给出了所有可以被函数 IWDG_GetFlagStatus 清除的标志位列表

Table 264. IWDG_FLAG 值

IWDG_FLAG	描述
IWDG_FLAG_PVU	预分频值更新进行中
IWDG_FLAG_RVU	重装载值更新进行中

例：

```
/* 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 驱动有多种用途：例如使能或者失能 IRQ 中断，使能或者失能单独的 IRQ 通道，改变 IRQ 通道的优先级等等。

Section 13.1 NVIC 寄存器结构描述了固件函数库所使用的数据结构，Section 13.2 固件库函数介绍了函数库里的所有函数。

13.1 NVIC寄存器结构

NVIC 寄存器结构，***NVIC_TypeDeff***，在文件***"stm32f10x_map.h"***中定义如下：

```
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 */
```

Table 265.例举了 NVIC 所有寄存器

Table 265. NVIC 寄存器

寄存器	描述
Enable	中断设置使能寄存器
Disable	中断清除使能寄存器
Set	中断设置待处理寄存器
Clear	中断清除待处理寄存器
Active	中断活动位寄存器
Priority	中断优先级寄存器
CPUID	CPU ID 基寄存器
IRQControlStatus	中断控制状态寄存器
ExceptionTableOffset	向量表移位寄存器

AIRC	应用控制/重置寄存器
SysCtrl	系统控制寄存器
ConfigCtrl	设置控制寄存器
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	设置错误状态寄存器
HardFaultStatus	硬件错误状态寄存器
DebugFaultStatus	除错错误寄存器
MemorymanageFaultAddr	存储器管理错误地址寄存器
BusFaultAddr	总线错误地址寄存器

NVIC 外设声明于文件***"stm32f10x_map.h"***：

```
...
#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
```

使用 Debug 模式时，初始化指针 ***NVIC, SCB*** 于文件***"stm32f10x_lib.c"***：

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

为了访问 NVIC 寄存器，,***_NVIC*** 必须在文件***"stm32f10x_conf.h"***中定义如下：

```
#define _NVIC
```

13.2 NVIC库函数

Table 266. 例举了 NVIC 的库函数

Table 266. NVIC 库函数

函数名	描述
NVIC_DeInit	将外设 NVIC 寄存器重设为缺省值
NVIC_SCBDeInit	将外设 SCB 寄存器重设为缺省值
NVIC_PriorityGroupConfig	设置优先级分组：先占优先级和从优先级
NVIC_Init	根据 NVIC_InitStruct 中指定的参数初始化外设 NVIC 寄存器
NVIC_StructInit	把 NVIC_InitStruct 中的每一个参数按缺省值填入
NVIC_SETPRIMASK	使能 PRIMASK 优先级：提升执行优先级至 0
NVIC_RESETPRIMASK	失能 PRIMASK 优先级
NVIC_SETFAULTMASK	使能 FAULTMASK 优先级：提升执行优先级至-1
NVIC_RESETFAULTMASK	失能 FAULTMASK 优先级
NVIC_BASEPRICONFIG	改变执行优先级从 N（最低可设置优先级）提升至 1
NVIC_GetBASEPRI	返回 BASEPRI 屏蔽值
NVIC_GetCurrentPendingIRQChannel	返回当前待处理 IRQ 标识符
NVIC_GetIRQChannelPendingBitStatus	检查指定的 IRQ 通道待处理位设置与否
NVIC_SetIRQChannelPendingBit	设置指定的 IRQ 通道待处理位
NVIC_ClearIRQChannelPendingBit	清除指定的 IRQ 通道待处理位
NVIC_GetCurrentActiveHandler	返回当前活动的 Handler（IRQ 通道和系统 Handler）的标识符
NVIC_GetIRQChannelActiveBitStatus	检查指定的 IRQ 通道活动位设置与否
NVIC_GetCPUID	返回 ID 号码，Cortex-M3 内核的版本号和实现细节
NVIC_SetVectorTable	设置向量表的位置和偏移
NVIC_GenerateSystemReset	产生一个系统复位
NVIC_GenerateCoreReset	产生一个内核（内核+NVIC）复位
NVIC_SystemLPConfig	选择系统进入低功耗模式的条件
NVIC_SystemHandlerConfig	使能或者失能指定的系统 Handler
NVIC_SystemHandlerPriorityConfig	设置指定的系统 Handler 优先级
NVIC_GetSystemHandlerPendingBitStatus	检查指定的系统 Handler 待处理位设置与否
NVIC_SetSystemHandlerPendingBit	设置系统 Handler 待处理位
NVIC_ClearSystemHandlerPendingBit	清除系统 Handler 待处理位
NVIC_GetSystemHandlerActiveBitStatus	检查系统 Handler 活动位设置与否
NVIC_GetFaultHandlerSources	返回表示出错的系统 Handler 源
NVIC_GetFaultAddress	返回产生表示出错的系统 Handler 所在位置的地址

13.2.1 函数NVIC_DeInit

Table 267. 描述了函数 NVIC_DeInit

Table 267. 函数 NVIC_DeInit

函数名	NVIC_DeInit
函数原形	void NVIC_DeInit(void)
功能描述	将外设 NVIC 寄存器重设为缺省值
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Resets the NVIC registers to their default reset value */
NVIC_DeInit();
```

13.2.2 函数NVIC_SCBDeInit

Table 268. 描述了函数 NVIC_SCBDeInit

Table 268. 函数 NVIC_SCBDeInit

函数名	NVIC_SCBDeInit
函数原形	void NVIC_SCBDeInit(void)
功能描述	将外设 SCB 寄存器重设为缺省值
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Resets the SCB registers to their default reset value */
NVIC_SCBDeInit();
```

13.2.3 函数NVIC_PriorityGroupConfig

Table 269. 描述了函数 NVIC_PriorityGroupConfig

Table 269. 函数 NVIC_PriorityGroupConfig

函数名	NVIC_PriorityGroupConfig
函数原形	void NVIC_PriorityGroupConfig(u32 NVIC_PriorityGroup)
功能描述	设置优先级分组：先占优先级和从优先级
输入参数	NVIC_PriorityGroup：优先级分组位长度 参阅 Section：NVIC_PriorityGroup 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	优先级分组只能设置一次
被调用函数	无

NVIC_PriorityGroup
该参数设置优先级分组位长度（见 Table 270.）

Table 270. NVIC_PriorityGroup 值

NVIC_PriorityGroup	描述
NVIC_PriorityGroup_0	先占优先级 0 位 从优先级 4 位
NVIC_PriorityGroup_1	先占优先级 1 位 从优先级 3 位
NVIC_PriorityGroup_2	先占优先级 2 位 从优先级 2 位
NVIC_PriorityGroup_3	先占优先级 3 位 从优先级 1 位
NVIC_PriorityGroup_4	先占优先级 4 位 从优先级 0 位

例：

```
/* Configure the Priority Grouping with 1 bit */
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
```

13.2.4 函数NVIC_Init

Table 271. 描述了函数 NVIC_Init

Table 271. 函数 NVIC_Init

函数名	NVIC_Init
函数原形	void NVIC_Init(NVIC_InitTypeDef* NVIC_InitStruct)
功能描述	根据 NVIC_InitStruct 中指定的参数初始化外设 NVIC 寄存器
输入参数	NVIC_InitStruct：指向结构 NVIC_InitTypeDef 的指针，包含了外设 GPIO 的配置信息 参阅 Section：NVIC_InitTypeDef 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

NVIC_InitTypeDef structure

NVIC_InitTypeDef 定义于文件“stm32f10x_nvic.h”：

```
typedef struct
{
u8 NVIC_IRQChannel;
u8 NVIC_IRQChannelPreemptionPriority;
u8 NVIC_IRQChannelSubPriority;
FunctionalState NVIC_IRQChannelCmd;
} NVIC_InitTypeDef;
```

NVIC_IRQChannel

该参数用以使能或者失能指定的 IRQ 通道。Table 272. 给出了该参数可取的值

Table 272. NVIC_IRQChannel 值

NVIC_IRQChannel	描述
WWDG_IRQChannel	窗口看门狗中断
PVD_IRQChannel	PVD 通过 EXTI 探测中断
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	外部中断线 0 中断
EXTI1_IRQChannel	外部中断线 1 中断
EXTI2_IRQChannel	外部中断线 2 中断
EXTI3_IRQChannel	外部中断线 3 中断
EXTI4_IRQChannel	外部中断线 4 中断
DMAChannel1_IRQChannel	DMA 通道 1 中断
DMAChannel2_IRQChannel	DMA 通道 2 中断
DMAChannel3_IRQChannel	DMA 通道 3 中断
DMAChannel4_IRQChannel	DMA 通道 4 中断
DMAChannel5_IRQChannel	DMA 通道 5 中断
DMAChannel6_IRQChannel	DMA 通道 6 中断
DMAChannel7_IRQChannel	DMA 通道 7 中断
ADC_IRQChannel	ADC 全局中断
USB_HP_CANTX_IRQChannel	USB 高优先级或者 CAN 发送中断
USB_LP_CAN_RX0_IRQChannel	USB 低优先级或者 CAN 接收 0 中断
CAN_RX1_IRQChannel	CAN 接收 1 中断
CAN_SCE_IRQChannel	CAN SCE 中断

EXTI9_5_IRQChannel	外部中断线 9-5 中断
TIM1_BRK_IRQChannel	TIM1 暂停中断
TIM1_UP_IRQChannel	TIM1 刷新中断
TIM1_TRG_COM_IRQChannel	TIM1 触发和通讯中断
TIM1_CC_IRQChannel	TIM1 捕获比较中断
TIM2_IRQChannel	TIM2 全局中断
TIM3_IRQChannel	TIM3 全局中断
TIM4_IRQChannel	TIM4 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	USART2 全局中断
USART3_IRQChannel	USART3 全局中断
EXTI15_10_IRQChannel	外部中断线 15-10 中断
RTCAlarm_IRQChannel	RTC 闹钟通过 EXTI 线中断
USBWakeUp_IRQChannel	USB 通过 EXTI 线从悬挂唤醒中断

NVIC_IRQChannelPreemptionPriority

该参数设置了成员 NVIC_IRQChannel 中的先占优先级，Table. 273 列举了该参数的取值。

NVIC_IRQChannelSubPriority

该参数设置了成员 NVIC_IRQChannel 中的从优先级，Table. 273 列举了该参数的取值。

Table. 273 给出了由函数 NVIC_PriorityGroupConfig 设置的先占优先级和从优先级可取的值

Table 273. 先占优先级和从优先级值(1)(2)

NVIC_PriorityGroup	NVIC_IRQChannel 的先占优先级	NVIC_IRQChannel 的从优先级	描述
NVIC_PriorityGroup_0	0	0-15	先占优先级 0 位 从优先级 4 位
NVIC_PriorityGroup_1	0-1	0-7	先占优先级 1 位 从优先级 3 位
NVIC_PriorityGroup_2	0-3	0-3	先占优先级 2 位 从优先级 2 位
NVIC_PriorityGroup_3	0-7	0-1	先占优先级 3 位 从优先级 1 位
NVIC_PriorityGroup_4	0-15	0	先占优先级 4 位 从优先级 0 位

1. 选中 NVIC_PriorityGroup_0，则参数 NVIC_IRQChannelPreemptionPriority 对中断通道的设置不产生影响。
2. 选中 NVIC_PriorityGroup_4，则参数 NVIC_IRQChannelSubPriority 对中断通道的设置不产生影响。

NVIC_IRQChannelCmd

该参数指定了在成员 NVIC_IRQChannel 中定义的 IRQ 通道被使能还是失能。这个参数取值为 ENABLE 或者 DISABLE。

例：

```
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 函数NVIC_StructInit

Table 274. 描述了函数 NVIC_StructInit

Table 274. 函数 NVIC_StructInit

函数名	NVIC_StructInit
函数原形	void NVIC_StructInit (NVIC_InitTypeDef* NVIC_InitStruct)
功能描述	把 NVIC_InitStruct 中的每一个参数按缺省值填入
输入参数	NVIC_InitStruct：指向结构 NVIC_InitTypeDef 的指针，待初始化
输出参数	无
返回值	无
先决条件	无
被调用函数	无

Table 275. 给出了 NVIC_InitStruct 各个成员的缺省值

Table 275. NVIC_InitStruct 缺省值

成员	缺省值
NVIC_IRQChannel	0x0
NVIC_IRQChannelPreemptionPriority	0
NVIC_IRQChannelSubPriority	0
NVIC_IRQChannelCmd	DISABLE

例：

```
/* The following example illustrates how to initialize a
NVIC_InitTypeDef structure */
NVIC_InitTypeDef NVIC_InitStructure;
NVIC_StructInit(&NVIC_InitStructure);
```

13.2.6 函数NVIC_SETPRIMASK

Table 276. 描述了函数 NVIC_SETPRIMASK

Table 276. 函数 NVIC_SETPRIMASK (1)(2)(3)

函数名	NVIC_SETPRIMASK
函数原形	void NVIC_SETPRIMASK(void)
功能描述	使能 PRIMASK 优先级：提升执行优先级至 0
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	__SETPRIMASK()

1. 该函数由汇编语言书写。
2. 该函数只影响组优先级，不影响从优先级。
3. 在设置 PRIMASK 寄存器前，建议在从为了使能一个例外中另一个例外返回时，清除该寄存器

例：

```
/* Enable the PRIMASK priority */
NVIC_SETPRIMASK();
```

13.2.7 函数NVIC_RESETPRIMASK

Table 277. 描述了函数 NVIC_RESETPRIMASK

Table 277. 函数 NVIC_RESETPRIMASK (1)

函数名	NVIC_Init
函数原形	void NVIC_RESETPRIMASK(void)
功能描述	失能 PRIMASK 优先级
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	__RESETPRIMASK()

1. 该函数由汇编语言书写。

例：

```
/* Disable the PRIMASK priority */
NVIC_RESETPRIMASK();
```

13.2.8 函数NVIC_SETFAULTMASK

Table 278. 描述了函数 NVIC_SETFAULTMASK

Table 278. 函数 NVIC_SETFAULTMASK (1)(2)(3)

函数名	NVIC_SETFAULTMASK
函数原形	void NVIC_SETFAULTMASK(void)
功能描述	使能 FAULTMASK 优先级：提升执行优先级至-1
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	__SETFAULTMASK()

1. 该函数由汇编语言书写。
2. 该函数只影响组优先级，不影响从优先级。
3. FAULTMASK 只有在执行优先级值小于-1 的情况下才能被设置，设置 FAULTMASK 将它的执行优先级提升到 HardFAULT 的级别。每当从除 NMI 之外的例外中返回，FAULTMASK 会被自动清除。

例：

```
/* Enable the FAULTMASK priority */
NVIC_SETFAULTMASK();
```

13.2.9 函数NVIC_RESETFAULTMASK

Table 279. 描述了函数 NVIC_ RESETFAULTMASK

Table 279. 函数 NVIC_ RESETFAULTMASK (1)

函数名	NVIC_RESETFAULTMASK
函数原形	void NVIC_RESETFAULTMASK(void)
功能描述	失能 FAULTMASK 优先级
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	__RESETFAULTMASK()

1. 该函数由汇编语言书写。

例：

```
/* Enable the PRIMASK priority */
NVIC_RESETPRIMASK();
```

13.2.10 函数NVIC_BASEPRICONFIG

Table 280. 描述了函数 NVIC_ BASEPRICONFIG

Table 280. 函数 NVIC_ BASEPRICONFIG (1)(2)(3)

函数名	NVIC_ BASEPRICONFIG
函数原形	void NVIC_BASEPRICONFIG(u32 NewPriority)
功能描述	改变执行优先级从 N（最低可设置优先级）提升至 1
输入参数	NewPriority：执行优先级的新优先级值
输出参数	无
返回值	无
先决条件	无
被调用函数	__BASEPRICONFIG()

1. 该函数由汇编语言书写。
2. 该函数只影响组优先级，不影响从优先级。
3. 可以改变执行优先级，从N（最低可设置优先级）提升至1。将该寄存器清除至0不会影响当前的优先级，它的非零值起到优先级屏蔽的作用，执行后当BASEPRI定义的优先级高于当前优先级时，该操作将起作用。

例：

```
/* Mask the execution priority to 10 */
__BASEPRICONFIG(10);
```

13.2.11 函数NVIC_GetBASEPRI

Table 281. 描述了函数 NVIC_GetBASEPRI

Table 281. 函数 NVIC_GetBASEPRI (1)

函数名	NVIC_GetBASEPRI
函数原形	u32 NVIC_GetBASEPRI(void)
功能描述	返回 BASEPRI 屏蔽值
输入参数	无
输出参数	无
返回值	BASEPRI 屏蔽值
先决条件	无
被调用函数	__GetBASEPRI()

1. 该函数由汇编语言书写。

例：

```
/* Get the execution priority to value */
u32 BASEPRI_Mask = 0;
BASEPRI_Mask = NVIC_GetBASEPRI();
```

13.2.12 函数NVIC_GetCurrentPendingIRQChannel

Table 282. 描述了函数 NVIC_GetCurrentPendingIRQChannel

Table 282. 函数 NVIC_GetCurrentPendingIRQChannel

函数名	NVIC_GetCurrentPendingIRQChannel
函数原形	u16 NVIC_GetCurrentPendingIRQChannel(void)
功能描述	返回当前待处理 IRQ 标识符
输入参数	无
输出参数	无
返回值	待处理 IRQ 标识符
先决条件	无
被调用函数	无

例：

```
/* Get the current pending IRQ channel identifier */
u16 CurrentPendingIRQChannel;
CurrentPendingIRQChannel = NVIC_GetCurrentPendingIRQChannel();
```

13.2.13 函数NVIC_GetIRQChannelPendingBitStatus

Table 283. 描述了函数 NVIC_GetIRQChannelPendingBitStatus

Table 283. 函数 NVIC_GetIRQChannelPendingBitStatus

函数名	NVIC_GetIRQChannelPendingBitStatus
函数原形	ITStatus NVIC_GetIRQChannelPendingBitStatus(u8 NVIC_IRQChannel)
功能描述	检查指定的IRQ通道待处理位设置与否
输入参数	NVIC_IRQChannel： 待检查的 IRQ 通道待处理位 参阅 Section：NVIC_IRQChannel 查阅更多该参数允许取值范围
输出参数	无
返回值	待检查 IRQ 待处理位的新状态（SET 或者 RESET）
先决条件	无
被调用函数	无

例：

```
/* Get the IRQ channel pending bit status of the ADC_IRQChannel */
ITStatus IRQChannelPendingBitStatus;
IRQChannelPendingBitStatus =
NVIC_GetIRQChannelPendingBitStatus(ADC_IRQChannel);
```

13.2.14 函数NVIC_SetIRQChannelPendingBit

Table 284. 描述了函数 NVIC_ SetIRQChannelPendingBit

Table 284. 函数 NVIC_ SetIRQChannelPendingBit

函数名	NVIC_ SetIRQChannelPendingBit
函数原形	void NVIC_SetIRQChannelPendingBit(u8 NVIC_IRQChannel)
功能描述	设置指定的IRQ通道待处理位
输入参数	NVIC_IRQChannel： 待设置的 IRQ 通道待处理位 参阅 Section：NVIC_IRQChannel 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Set SPI1 Global interrupt pending bit */
NVIC_SetIRQChannelPendingBit(SPI1_IRQChannel);
```

13.2.15 函数NVIC_ClearIRQChannelPendingBit

Table 285. 描述了函数 NVIC_ ClearIRQChannelPendingBit

Table 285. 函数 NVIC_ ClearIRQChannelPendingBit

函数名	NVIC_ ClearIRQChannelPendingBit
函数原形	void NVIC_ClearIRQChannelPendingBit(u8 NVIC_IRQChannel)
功能描述	清除指定的IRQ通道待处理位
输入参数	NVIC_IRQChannel： 待清除的 IRQ 通道待处理位 参阅 Section：NVIC_IRQChannel 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear ADC IRQ Channel Pending bit */
NVIC_ClearIRQChannelPendingBit(ADC_IRQChannel);
```

13.2.16 函数NVIC_GetCurrentActiveHandler

Table 286. 描述了函数 NVIC_GetCurrentActiveHandler

Table 286. 函数 NVIC_GetCurrentActiveHandler

函数名	NVIC_GetCurrentActiveHandler
函数原形	u16 NVIC_GetCurrentActiveHandler(void)
功能描述	返回当前活动的Handler（IRQ通道和系统Handler）的标识符
输入参数	无
输出参数	无
返回值	活动 Handler 的标识符
先决条件	无
被调用函数	无

例：

```
/* Get the current active Handler identifier */
u16 CurrentActiveHandler;
CurrentActiveHandler = NVIC_GetCurrentActiveHandler();
```

13.2.17 函数NVIC_GetIRQChannelActiveBitStatus

Table 287. 描述了函数 NVIC_GetIRQChannelActiveBitStatus

Table 287. 函数 NVIC_GetIRQChannelActiveBitStatus

函数名	NVIC_GetIRQChannelActiveBitStatus
函数原形	ITStatus NVIC_GetIRQChannelActiveBitStatus(u8 NVIC_IRQChannel)
功能描述	检查指定的IRQ通道活动位设置与否
输入参数	NVIC_IRQChannel： 待检查的指定中断活动位 参阅 Section：NVIC_IRQChannel 查阅更多该参数允许取值范围
输出参数	无
返回值	指定中断活动位的新状态（SET 或者 RESET）
先决条件	无
被调用函数	无

例：

```
/* Get the active IRQ channel status of the ADC_IRQChannel */
ITStatus IRQChannelActiveBitStatus;
IRQChannelActiveBitStatus =
NVIC_GetIRQChannelActiveBitStatus(ADC_IRQChannel);
```

13.2.18 函数NVIC_GetCPUID

Table 288. 描述了函数 NVIC_GetCPUID

Table 288. 函数 NVIC_GetCPUID

函数名	NVIC_GetCPUID
函数原形	u32 NVIC_GetCPUID(void)
功能描述	返回ID号码，Cortex-M3内核的版本号和实现细节
输入参数	无
输出参数	无
返回值	CPU ID
先决条件	无
被调用函数	无

例：

```
/* Gets the CPU ID */
u32 CM3_CPUID;
CM3_CPUID = NVIC_GetCPUID();
```

13.2.19 函数NVIC_SetVectorTable

Table 289. 描述了函数 NVIC_SetVectorTable

Table 289. 函数 NVIC_SetVectorTable

函数名	NVIC_SetVectorTable
函数原形	void NVIC_SetVectorTable(u32 NVIC_VectTab, u32 Offset)
功能描述	设置向量表的位置和偏移
输入参数 1	NVIC_VectTab：指定向量表位置在 RAM 还是在程序存储器 参阅 Section：NVIC_VectTab 查阅更多该参数允许取值范围
输入参数 2	Offset：向量表基地址的偏移量 对 FLASH，该参数值必须高于 0x08000100；对 RAM 必须高于 0x100。它同时必须是 256（64×4）的整数倍
输出参数	无
返回值	指定中断活动位的新状态（SET 或者 RESET）
先决条件	无
被调用函数	无

NVIC_VectTab

该参数设置向量表基地址（见 Table 290.）

Table 290. NVIC_ VectTab 值

NVIC_ VectTab	描述
NVIC_VectTab_FLASH	向量表位于 FLASH
NVIC_VectTab_RAM	向量表位于 RAM

例：

```
/* Vector Table is in FLASH at 0x0 */
NVIC_SetVectorTable(NVIC_VectTab_FLASH, 0x0);
```

13.2.20 函数NVIC_GenerateSystemReset

Table 291. 描述了函数 NVIC_GenerateSystemReset

Table 291. 函数 NVIC_GenerateSystemReset

函数名	NVIC_GenerateSystemReset
函数原形	void NVIC_GenerateSystemReset(void)
功能描述	产生一个系统复位
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Generate a system reset */
NVIC_GenerateSystemReset();
```

13.2.21 函数NVIC_GenerateCoreReset

Table 292. 描述了函数 NVIC_GenerateCoreReset

Table 292. 函数 NVIC_GenerateCoreReset

函数名	NVIC_GenerateCoreReset
函数原形	void NVIC_GenerateCoreReset(void)
功能描述	产生一个内核（内核+NVIC）复位
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Generate a core reset */
NVIC_GenerateCoreReset();
```

13.2.22 函数NVIC_SystemLPConfig

Table 293. 描述了函数 NVIC_SystemLPConfig

Table 293. 函数 NVIC_SystemLPConfig

函数名	NVIC_SystemLPConfig
函数原形	void NVIC_SystemLPConfig(u8 LowPowerMode, FunctionalState NewState)
功能描述	选择系统进入低功耗模式的条件
输入参数 1	LowPowerMode：系统进入低功耗模式的新模式 参阅 Section：LowPowerMode 查阅更多该参数允许取值范围
输入参数 2	NewState：LP 条件的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

LowPowerMode

该参数设置了设备的低功耗模式（见 Table 294.）

Table 294. LowPowerMode 值

LowPowerMode	描述
NVIC_LP_SEVONPEND	根据待处理请求唤醒
NVIC_LP_SLEEPDEEP	深度睡眠使能
NVIC_LP_SLEEPONEXIT	退出 ISR 后睡眠

例：

```
/* wakeup the system on interrupt pending */
NVIC_SystemLPConfig(SEVONPEND, ENABLE);
```

13.2.23 函数NVIC_SystemHandlerConfig

Table 295. 描述了函数 NVIC_SystemHandlerConfig

Table 295. 函数 NVIC_SystemHandlerConfig

函数名	NVIC_SystemHandlerConfig
函数原形	void NVIC_SystemHandlerConfig(u32 SystemHandler, FunctionalState NewState)
功能描述	使能或者失能指定的系统 Handler
输入参数 1	SystemHandler：待使能或者失能指定的系统 Handler 参阅 Section：LowPowerMode 查阅更多该参数允许取值范围
输入参数 2	NewState：指定系统 Handler 的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

SystemHandler

该参数设置了待使能或者失能指定的系统Handler（见 Table 296.）

Table 296. SystemHandler 值

SystemHandler	描述
SystemHandler_MemoryManage	存储器管理 Handler
SystemHandler_BusFault	总线错误 Handler
SystemHandler_UsageFault	使用错误 Handler

该参数允许同时设置 NVIC 寄存器，SCB 寄存器和索引位。SystemHandler 有 23 位编码长度，详情参阅 Table 297. – Table. 306

例：

```
/* Enable the Memory Manage Handler */
NVIC_SystemHandlerConfig(SystemHandler_MemoryManage, ENABLE);
```

Table 297. SystemHandler 定义

System Handler	Bits																					Value
	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
SystemHandler_ NMI (see *Table 298*)	Reserved																		0x1F					0x1F
SystemHandler_ HardFault (see *Table 299*)	Reserved			0	Reserved																			0x0
SystemHandler_ MemoryManage (see *Table 300*)	0	0		1	0x0					0xD				0	0			Res	0x10					0x43430
SystemHandler_ BusFault (see *Table 301*)	1	1		1	1					0xE				1	0			Res	0x11					0x547931
SystemHandler_ UsageFault (see *Table 302*)	-	2		1	0x3					Reserved				2	0			Res	0x12					0x24C232
SystemHandler_ SVCall (see *Table 303*)	Reserved				0x7					0xF				3	1			Reserved						0x1FF40
SystemHandler_ DebugMonitor (see *Table 304*)	Reserved			2	0x8					Reserved				0	2			Reserved						0xA0080
SystemHandler_ PSV (see *Table 305*)	Reserved				0xA					Reserved				2	2				0x1C					0x2829C
SystemHandler_ SysTick (see *Table 306*)	Reserved				0xB					Reserved				3	2				0x1A					0x2C39A

Table 298. SystemHandler_NMI 定义

Bits	NMI	
	Registers/Bits	Functions
[4:0]	- IRQControlState - NMIPENDSET[31]	NVIC_SetSystemHandlerPendingBit
5	Not Used	
[7:6]	Not Used	
[9:8]	Not Used	
[13:10]	Not Used	
[17:14]	Not Used	
[19:18]	Not Used	
[21:20]	Not Used	
22	Not Used	

Table 299. SystemHandler_HardFault 定义

Bits	Hard Fault	
	Registers/Bits	Functions
[4:0]	Not Used	
5	Not Used	
[7:6]	Not Used	
[9:8]	Not Used	
[13:10]	Not Used	
[17:14]	Not Used	
[19:18]	- HardFaultStatus	NVIC_GetFaultHandlerSources
[21:20]		
22	Not Used	

Table 300. SystemHandler_MemoryManage 定义

Bits	Memory Manage	
	Registers/Bits	Functions
[4:0]	- SysHandlerCtrl - MEMFAULTENA[16]	NVIC_SystemHandlerConfig
5	Not Used	
[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

Table 301. SystemHandler_BusFault 定义

Bits	Bus Fault	
	Registers/Bits	Functions
[4:0]	- SysHandlerCtrl - BUSFAULTENA[17]	NVIC_SystemHandlerConfig
5	Not Used	
[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

Table 302. SystemHandler_UsageFault 定义

Bits	Usage Fault	
	Registers/Bits	Functions
[4:0]	- SysHandlerCtrl - USGFAULTENA[18]	NVIC_SystemHandlerConfig
5	Not Used	
[7:6]	- SystemPriority[0] - PRI_6[23:16]	NVIC_SystemHandlerPriorityConfig
[9:8]		
[13:10]	Not Used	
[17:14]	- SysHandlerCtrl - USGFAULTACT[3]	NVIC_GetSystemHandlerActiveBitStatus
[19:18]	- ConfigFaultStatus - [31:16]	NVIC_GetFaultHandlerSources
[21:20]		
22	Not Used	

Table 303. SystemHandler_SVCall 定义

Bits	SVCall	
	Registers/Bits	Functions
[4:0]	Not Used	
5	Not Used	
[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]	Not Used	
[21:20]	Not Used	
22	Not Used	

Table 304. SystemHandler_DebugMoniter 定义

Bits	Debug Monitor	
	Registers/Bits	Functions
[4:0]	Not Used	
5	Not Used	
[7:6]	- SystemPriority[2] - PRI_12[7:0]	NVIC_SystemHandlerPriorityConfig
[9:8]		
[13:10]	Not Used	
[17:14]	- SysHandlerCtrl - MONITORACT[8]	NVIC_GetSystemHandlerActiveBitStatus
[19:18]	- DebugFaultStatus	NVIC_GetFaultHandlerSources
[21:20]		
22	Not Used	

Table 305. SystemHandler_PSV 定义

Bits	PSV	
	Registers/Bits	Functions
[4:0]	- IRQControlState - PENDSVSET[28]	NVIC_SetSystemHandlerPendingBit
	- IRQControlState - PENDSVCLR[27]	NVIC_ClearSystemHandlerPendingBit
5	Not Used	
[7:6]	- SystemPriority[2] - PRI_14[23:16]	NVIC_SystemHandlerPriorityConfig
[9:8]		
[13:10]	Not Used	
[17:14]	- SysHandlerCtrl - PENDSVACT[10]	NVIC_GetSystemHandlerActiveBitStatus
[19:18]	Not Used	
[21:20]	Not Used	
22	Not Used	

Table 306. SystemHandler_Systick 定义

Bits	SysTick	
	Registers/Bits	Functions
[4:0]	- IRQControlState - PENDSTSET[26]	NVIC_SetSystemHandlerPendingBit
	- IRQControlState - PENDSVCLR[25]	NVIC_ClearSystemHandlerPendingBit
5	Not Used	
[7:6]	- SystemPriority[2] - PRI_15[31:24]	NVIC_SystemHandlerPriorityConfig
[9:8]		
[13:10]	Not Used	
[17:14]	- SysHandlerCtrl - SYSTICKACT[11]	NVIC_GetSystemHandlerActiveBitStatus
[19:18]	Not Used	
[21:20]	Not Used	
22	Not Used	

13.2.24 函数NVIC_SystemHandlerPriorityConfig

Table 307. 描述了函数 NVIC_SystemHandlerPriorityConfig

Table 307. 函数 NVIC_SystemHandlerPriorityConfig

函数名	NVIC_SystemHandlerPriorityConfig
函数原形	void NVIC_SystemHandlerPriorityConfig(u32 SystemHandler, u8 SystemHandlerPreemptionPriority, u8 SystemHandlerSubPriority)
功能描述	设置指定的系统 Handler 优先级
输入参数 1	SystemHandler：待使能或者失能的指定系统 Handler 参阅 Section：SystemHandler 查阅更多该参数允许取值范围
输入参数 2	SystemHandlerPreemptionPriority：指定系统 Handler 的新组优先级 参阅 Section：SystemHandlerPreemptionPriority 查阅更多该参数允许取值范围
输入参数 3	SystemHandlerSubPriority：指定系统 Handler 的新从优先级 参阅 Section：SystemHandlerSubPriority 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

SystemHandler

该参数指定了待设置的系统Handler（见 Table 308.）

Table 308. SystemHandler 类型

NVIC_ VectTab	描述
SystemHandler_MemoryManage	存储器管理 Handler
SystemHandler_BusFault	总线错误 Handler
SystemHandler_UsageFault	使用错误 Handler
SystemHandler_SVCall	SVCall Handler
SystemHandler_DebugMonitor	除错监控 Handler
SystemHandler_PSV	PSV Handler
SystemHandler_SysTick	系统滴答定时器 Handler

例：

```
/* Enable the Memory Manage Handler */
NVIC_SystemHandlerPriorityConfig(SystemHandler_MemoryManage, 2, 8);
```

13.2.25 函数NVIC_GetSystemHandlerPendingBitStatus

Table 309. 描述了函数 NVIC_GetSystemHandlerPendingBitStatus

Table 309. 函数 NVIC_GetSystemHandlerPendingBitStatus

函数名	NVIC_GetSystemHandlerPendingBitStatus
函数原形	ITStatus NVIC_GetSystemHandlerPendingBitStatus(u32 SystemHandler)
功能描述	检查指定的系统 Handler 待处理位设置与否
输入参数	SystemHandler：待使能或者失能的指定系统 Handler 参阅 Section：SystemHandler 查阅更多该参数允许取值范围
输出参数	无
返回值	系统 Handler 待处理位的新状态（SET 或者 RESET）
先决条件	无
被调用函数	无

SystemHandler

该参数指定系统Handler（见 Table 310.）

Table 310. SystemHandler 类型

NVIC_ VectTab	描述
SystemHandler_MemoryManage	存储器管理 Handler
SystemHandler_BusFault	总线错误 Handler
SystemHandler_SVCall	SVCall Handler

例：

```
/* Check if the Memory Manage Fault has occured */
ITStatus MemoryHandlerStatus;
MemoryHandlerStatus
=NVIC_GetSystemHandlerPendingBitStatus(SystemHandler_MemoryManage);
```

13.2.26 函数NVIC_SetSystemHandlerPendingBit

Table 311. 描述了函数 NVIC_ SetSystemHandlerPendingBit

Table 311. 函数 NVIC_ SetSystemHandlerPendingBit

函数名	NVIC_ SetSystemHandlerPendingBit
函数原形	void NVIC_SetSystemHandlerPendingBit(u32 SystemHandler)
功能描述	设置系统 Handler 待处理位
输入参数	SystemHandler：待设置的指定系统 Handlerr 待处理位 参阅 Section：SystemHandler 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

SystemHandler

该参数指定系统Handler（见 Table 312.）

Table 312. SystemHandler 类型

NVIC_ VectTab	描述
SystemHandler_NMI	NMI Handler
SystemHandler_PSV	PSV Handler
SystemHandler_SysTick	系统滴答定时器 Handler

例：

```
/* Set NMI Pending Bit */
NVIC_SetSystemHandlerPendingBit(SystemHandler_NMI);
```

13.2.27 函数NVIC_ClearSystemHandlerPendingBit

Table 313. 描述了函数 NVIC_ClearSystemHandlerPendingBit

Table 313. 函数 NVIC_ClearSystemHandlerPendingBit

函数名	NVIC_ClearSystemHandlerPendingBit
函数原形	void NVIC_ClearSystemHandlerPendingBit(u32 SystemHandler)
功能描述	清除系统 Handler 待处理位
输入参数	SystemHandler：待清除的指定系统 Handlerr 待处理位 参阅 Section：SystemHandler 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

SystemHandler

该参数指定系统Handler（见 Table 314.）

Table 314. SystemHandler 类型

NVIC_ VectTab	描述
SystemHandler_PSV	PSV Handler
SystemHandler_SysTick	系统滴答定时器 Handler

例：

```
/* Clear SysTick Pending Bit */
NVIC_ClearSystemHandlerPendingBit(SystemHandler_SysTick);
```

13.2.28 函数NVIC_GetSystemHandlerActiveBitStatus

Table 315. 描述了函数 NVIC_ GetSystemHandlerActiveBitStatus

Table 315. 函数 NVIC_ GetSystemHandlerActiveBitStatus

函数名	NVIC_ GetSystemHandlerActiveBitStatus
函数原形	ITStatus NVIC_GetSystemHandlerActiveBitStatus(u32 SystemHandler)
功能描述	检查系统 Handler 活动位设置与否
输入参数	SystemHandler：待检查的系统 Handler 活动位 参阅 Section：SystemHandler 查阅更多该参数允许取值范围
输出参数	无
返回值	系统 Handler 活动位的新状态（SET 或者 RESET）
先决条件	无
被调用函数	无

SystemHandler

该参数指定系统Handler（见 Table 316.）

Table 316. SystemHandler 类型

NVIC_ VectTab	描述
SystemHandler_MemoryManage	存储器管理 Handler
SystemHandler_BusFault	总线错误 Handler
SystemHandler_UsageFault	使用错误 Handler
SystemHandler_DebugMonitor	除错监控 Handler
SystemHandler_PSV	PSV Handler
SystemHandler_SysTick	系统滴答定时器 Handler

例：

```
/* Check if the Bus Fault is active or stacked */
ITStatus BusFaultHandlerStatus;
BusFaultHandlerStatus =
NVIC_GetSystemHandlerActiveBitStatus(SystemHandler_BusFault);
```

13.2.29 函数NVIC_GetFaultHandlerSources

Table 317. 描述了函数 NVIC_ GetFaultHandlerSources

Table 317. 函数 NVIC_ GetFaultHandlerSources

函数名	NVIC_ GetFaultHandlerSources
函数原形	u32 NVIC_GetFaultHandlerSources(u32 SystemHandler)
功能描述	返回表示出错的系统 Handler 源
输入参数	SystemHandler：待返回表示出错的系统 Handler 源 参阅 Section：SystemHandler 查阅更多该参数允许取值范围
输出参数	无
返回值	表示出错的系统 Handler 源
先决条件	无
被调用函数	无

SystemHandler

该参数指定系统Handler（见 Table 318.）

Table 318. SystemHandler 类型

NVIC_ VectTab	描述
SystemHandler_HardFault	硬件错误 Handler
SystemHandler_MemoryManage	存储器管理 Handler
SystemHandler_BusFault	总线错误 Handler
SystemHandler_UsageFault	使用错误 Handler
SystemHandler_DebugMonitor	除错监控 Handler

例：

```
/* Gets the sources of the Bus Fault Handler */
u32 BusFaultHandlerSource;
BusFaultHandlerSource
=NVIC_GetFaultHandlerSources(SystemHandler_BusFault);
```

13.2.30 函数NVIC_GetFaultAddress

Table 319. 描述了函数 NVIC_GetFaultAddress

Table 319. 函数 NVIC_GetFaultAddress

函数名	NVIC_GetFaultAddress
函数原形	u32 NVIC_GetFaultAddress(u32 SystemHandler)
功能描述	返回产生表示出错的系统 Handler 所在位置的地址
输入参数	SystemHandler：待返回产生表示出错的系统 Handler 所在位置的地址 参阅 Section：SystemHandler 查阅更多该参数允许取值范围
输出参数	无
返回值	表示出错的系统 Handler 所在位置的地址
先决条件	无
被调用函数	无

SystemHandler

该参数指定系统Handler（见 Table 320.）

Table 320. SystemHandler 类型

NVIC_ VectTab	描述
SystemHandler_MemoryManage	存储器管理 Handler
SystemHandler_BusFault	总线错误 Handler

例：

```
/* Gets the address of the Bus Fault Handler */
u32 BusFaultHandlerAddress;
BusFaultHandlerAddress =
NVIC_GetFaultAddress(SystemHandler_BusFault);
```

14 功耗控制（PWR）

PWR 有多种用途，包括功耗管理和低功耗模式选择。

Section 14.1 PWR 寄存器结构描述了固件函数库所使用的数据结构，Section 14.2 固件库函数介绍了函数库里的所有函数。

14.1 PWR寄存器结构

PWR 寄存器结构，***PWR_TypeDeff***，在文件***"stm32f10x_map.h"***中定义如下：

```
typedef struct
{
vu32 CR;
vu32 CSR;
} PWR_TypeDef;
```

Table 321.例举了PWR所有寄存器

Table 321. PWR 寄存器

寄存器	描述
CR	功耗控制寄存器
CSR	功耗控制状态寄存器

PWR 外设声明于文件***"stm32f10x_map.h"***：

```
#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Debug模式时，初始化指针***PWR***于文件***"stm32f10x_lib.c"***：

```
#ifdef _PWR
PWR = (PWR_TypeDef *) PWR_BASE;
#endif /*_PWR */
```

为了访问PWR寄存器，，***_PWR***必须在文件***"stm32f10x_conf.h"***中定义如下：

```
#define _PWR
```

14.2 PWR库函数

Table 322. 例举了PWR的库函数

Table 322. PWR 库函数

函数名	描述
PWR_DeInit	将外设 PWR 寄存器重设为缺省值
PWR_BackupAccessCmd	使能或者失能 RTC 和后备寄存器访问
PWR_PVDCmd	使能或者失能可编程电压探测器（PVD）
PWR_PVDLevelConfig	设置 PVD 的探测电压阈值
PWR_WakeUpPinCmd	使能或者失能唤醒管脚功能
PWR_EnterSTOPMode	进入停止（STOP）模式
PWR_EnterSTANDBYMode	进入待命（STANDBY）模式
PWR_GetFlagStatus	检查指定 PWR 标志位设置与否
PWR_ClearFlag	清除 PWR 的待处理标志位

14.2.1 函数PWR_DeInit

Table 323. 描述了函数 PWR_DeInit

Table 323. 函数 PWR_DeInit

函数名	PWR_DeInit
函数原形	void PWR_DeInit(void)
功能描述	将外设 I2Cx 寄存器重设为缺省值
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	RCC_APB1PeriphClockCmd().

例：

```
/* Deinitialize the PWR registers */
PWR_DeInit();
```

14.2.2 函数PWR_BackupAccessCmd

Table 324. 描述了函数 PWR_BackupAccessCmd

Table 324. 函数 PWR_BackupAccessCmd

函数名	PWR_BackupAccessCmd
函数原形	void PWR_BackupAccessCmd(FunctionalState NewState)
功能描述	使能或者失能 RTC 和后备寄存器访问
输入参数	NewState: RTC 和后备寄存器访问的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable access to the RTC and backup registers */
PWR_BackupAccessCmd(ENABLE);
```

14.2.3 函数PWR_PVDCmd

Table 325. 描述了函数 PWR_PVDCmd

Table 325. 函数 PWR_PVDCmd

函数名	PWR_PVDCmd
函数原形	void PWR_PVDCmd(FunctionalState NewState)
功能描述	使能或者失能可编程电压探测器（PVD）
输入参数	NewState: PVD 的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable the Power Voltage Detector(PVD) */
PWR_PVDCmd(ENABLE);
```

14.2.4 函数PWR_PVDLevelConfig

Table 326. 描述了 PWR_PVDLevelConfig

Table 326. 函数 PWR_PVDLevelConfig

函数名	PWR_PVDLevelConfig
函数原形	void PWR_PVDLevelConfig(u32 PWR_PVDLevel)
功能描述	设置 PVD 的探测电压阈值
输入参数	PWR_PVDLevel：PVD 的探测电压阈值 参阅 Section：PWR_PVDLevel 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

PWR_PVDLevel
该参数设置了 PVD 的探测电压阈值（见 Table 327.）

Table 327. PWR_PVDLevel 值

PWR_PVDLevel	描述
PWR_PVDLevel_2V2	PVD 探测电压阈值 2.2V
PWR_PVDLevel_2V3	PVD 探测电压阈值 2.3V
PWR_PVDLevel_2V4	PVD 探测电压阈值 2.4V
PWR_PVDLevel_2V5	PVD 探测电压阈值 2.5V
PWR_PVDLevel_2V6	PVD 探测电压阈值 2.6V
PWR_PVDLevel_2V7	PVD 探测电压阈值 2.7V
PWR_PVDLevel_2V8	PVD 探测电压阈值 2.8V
PWR_PVDLevel_2V9	PVD 探测电压阈值 2.9V

例：
```
/* Set PVD detection level to 2.5V */
PWR_PVDLevelConfig(PWR_PVDLevel_2V5);
```

14.2.5 函数PWR_WakeUpPinCmd

Table 328. 描述了函数 PWR_WakeUpPinCmd

Table 328. 函数 PWR_WakeUpPinCmd

函数名	PWR_WakeUpPinCmd
函数原形	void PWR_WakeUpPinCmd(FunctionalState NewState)
功能描述	使能或者失能唤醒管脚功能
输入参数	NewState: 唤醒管脚功能的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：
```
/* WakeUp pin used for wake-up function */
PWR_WakeUpPinCmd(ENABLE);
```

14.2.6 函数PWR_EnterSTOPMode

Table 329. 描述了函数 PWR_EnterSTOPMode

Table 329. 函数 PWR_EnterSTOPMode

函数名	PWR_EnterSTOPMode
函数原形	void PWR_EnterSTOPMode(u32 PWR_Regulator, u8 PWR_STOPEntry)
功能描述	进入停止（STOP）模式
输入参数 1	PWR_Regulator: 电压转换器在停止模式下的状态 参阅 Section：PWR_ Regulator 查阅更多该参数允许取值范围
输入参数 2	PWR_STOPEntry: 选择使用指令 WFE 还是 WFI 来进入停止模式 参阅 Section：PWR_STOPEntry 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	__WFI(), __WFE()

PWR_Regulator

该参数设置了电压转换器在停止模式下的状态（见Table 330.）

Table 330. PWR_Regulator 值

PWR_Regulator	描述
PWR_Regulator_ON	停止模式下电压转换器 ON
PWR_Regulator_LowPower	停止模式下电压转换器进入低功耗模式

PWR_STOPEntry

该参数选择使用指令 WFE 还是 WFI 来进入停止模式（见 Table 331.）

Table 331. PWR_Regulator 值

PWR_STOPEntry	描述
PWR_STOPEntry_WFI	使用指令 WFI 来进入停止模式
PWR_STOPEntry_WFE	使用指令 WFE 来进入停止模式

例：

```
/* Put the system in STOP mode with regulator on */
PWR_EnterSTOPMode(PWR_Regulator_ON, PWR_STOPEntry_WFE);
```

14.2.7 函数PWR_EnterSTANDBYMode

Table 332. 描述了函数 PWR_EnterSTANDBYMode

Table 332. 函数 PWR_EnterSTANDBYMode

函数名	PWR_EnterSTANDBYMode
函数原形	void PWR_EnterSTANDBYMode(void)
功能描述	进入待命（STANDBY）模式
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	__WFI(),

例：

```
/* Put the system in STANDBY mode */
PWR_EnterSTANDBYMode();
```

14.2.8 函数PWR_GetFlagStatus

Table 333. 描述了函数PWR_GetFlagStatus

Table 333. 函数 PWR_GetFlagStatus

函数名	PWR_GetFlagStatus
函数原形	FlagStatus PWR_GetFlagStatus(u32 PWR_FLAG)
功能描述	检查指定 PWR 标志位设置与否
输入参数	PWR_FLAG：待检查的 PWR 标志位 参阅 Section：PWR_FLAG 查阅更多该参数允许取值范围
输出参数	无
返回值	PWR_FLAG 的新状态（SET 或者 RESET）
先决条件	无
被调用函数	无

PWR_FLAG

Table 334. 给出了所有可以被函数PWR_ GetFlagStatus检查的标志位列表

Table 334. PWR_FLAG 值

PWR_FLAG	描述
PWR_FLAG_WU	唤醒标志位
PWR_FLAG_SB	待命（Standby）标志位
PWR_FLAG_PVDO	PVD 输出 (1)

1. 该标志位为只读，不能被清除

例：

```
/* Test if the StandBy flag is set or not */
FlagStatus Status;
Status = PWR_GetFlagStatus(PWR_FLAG_SB);
if(Status == RESET)
{
...
}
else
{
...
}
```

14.2.9 函数PWR_ClearFlag

Table 335. 描述了函数PWR_ClearFlag

Table 335. 函数 PWR_ClearFlag

函数名	PWR_ClearFlag
函数原形	void PWR_ClearFlag(u32 PWR_FLAG)
功能描述	清除 PWR 的待处理标志位
输入参数	PWR_FLAG：待清除的 PWR 待处理标志位 参阅 Section：PWR_FLAG 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear the StandBy pending flag */
PWR_ClearFlag(PWR_FLAG_SB);
```

15 复位和时钟设置（RCC）

RCC 有多种用途，包括时钟设置，外设复位和时钟管理。

Section 15.1 RCC 寄存器结构描述了固件函数库所使用的数据结构，Section 15.2 固件库函数介绍了函数库里的所有函数。

15.1 RCC寄存器结构

RCC 寄存器结构，***RCC_TypeDeff***，在文件***"stm32f10x_map.h"***中定义如下：

```
typedef struct
{
vu32 CR;
vu32 CFGR;
vu32 CIR;
vu32 APB2RSTR;
vu32 APB1RSTR;
vu32 AHBENR;
vu32 APB2ENR;
vu32 APB1ENR;
vu32 BDCR;
vu32 CSR;
} RCC_TypeDef;
```

Table 336.例举了RCC所有寄存器

Table 336. 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 外设复位寄存器
APB1RSTR	APB1 外设复位寄存器
AHBENR	AHB 外设时钟使能寄存器
APB2ENR	APB2 外设时钟使能寄存器
APB1ENR	APB1 外设时钟使能寄存器
BDCR	备份域控制寄存器
CSR	控制/状态寄存器

RCC 外设声明于文件***"stm32f10x_map.h"***：

```
#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
```

使用Debug模式时，初始化指针***RCC***于文件***“stm32f10x_lib.c”***：

```
#ifdef _RCC
RCC = (RCC_TypeDef *) RCC_BASE;
#endif /*_RCC */
```

为了访问RCC寄存器，***_RCC***必须在文件***“stm32f10x_conf.h”***中定义如下：

```
#define _RCC
```

15.2 RCC库函数

Table 337. 例举了RCC的库函数

Table 337. RCC 库函数

函数名	描述
RCC_DeInit	将外设 RCC 寄存器重设为缺省值
RCC_HSEConfig	设置外部高速晶振（HSE）
RCC_WaitForHSEStartUp	等待 HSE 起振
RCC_AdjustHSICalibrationValue	调整内部高速晶振（HSI）校准值
RCC_HSICmd	使能或者失能内部高速晶振（HSI）
RCC_PLLConfig	设置 PLL 时钟源及倍频系数
RCC_PLLCmd	使能或者失能 PLL
RCC_SYSCLKConfig	设置系统时钟（SYSCLK）
RCC_GetSYSCLKSource	返回用作系统时钟的时钟源
RCC_HCLKConfig	设置 AHB 时钟（HCLK）
RCC_PCLK1Config	设置低速 AHB 时钟（PCLK1）
RCC_PCLK2Config	设置高速 AHB 时钟（PCLK2）
RCC_ITConfig	使能或者失能指定的 RCC 中断
RCC_USBCLKConfig	设置 USB 时钟（USBCLK）
RCC_ADCCLKConfig	设置 ADC 时钟（ADCCLK）
RCC_LSEConfig	设置外部低速晶振（LSE）
RCC_LSICmd	使能或者失能内部低速晶振（LSI）
RCC_RTCCLKConfig	设置 RTC 时钟（RTCCLK）
RCC_RTCCLKCmd	使能或者失能 RTC 时钟
RCC_GetClocksFreq	返回不同片上时钟的频率
RCC_AHBPeriphClockCmd	使能或者失能 AHB 外设时钟
RCC_APB2PeriphClockCmd	使能或者失能 APB2 外设时钟
RCC_APB1PeriphClockCmd	使能或者失能 APB1 外设时钟
RCC_APB2PeriphResetCmd	强制或者释放高速 APB（APB2）外设复位
RCC_APB1PeriphResetCmd	强制或者释放低速 APB（APB1）外设复位
RCC_BackupResetCmd	强制或者释放后备域复位
RCC_ClockSecuritySystemCmd	使能或者失能时钟安全系统
RCC_MCOConfig	选择在 MCO 管脚上输出的时钟源
RCC_GetFlagStatus	检查指定的 RCC 标志位设置与否
RCC_ClearFlag	清除 RCC 的复位标志位
RCC_GetITStatus	检查指定的 RCC 中断发生与否
RCC_ClearITPendingBit	清除 RCC 的中断待处理位

15.2.1 函数RCC_DeInit

Table 338. 描述了函数 RCC_DeInit

Table 338. 函数 RCC_DeInit(1)(2)

函数名	RCC_DeInit
函数原形	void RCC_DeInit(void)
功能描述	将外设 RCC 寄存器重设为缺省值
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

1. 该函数不改动寄存器 RCC_CR 的 HSITRIM[4:0]位。
2. 该函数不重置寄存器 RCC_BDCR 和寄存器 RCC_CSR。

例：

```
/* Deinitialize the RCC registers */
RCC_DeInit();
```

15.2.2 函数RCC_HSEConfig

Table 339. 描述了函数RCC_HSEConfig

Table 339. 函数 RCC_HSEConfig

函数名	RCC_HSEConfig
函数原形	void RCC_HSEConfig(u32 RCC_HSE)
功能描述	设置外部高速晶振（HSE）
输入参数	RCC_HSE: HSE 的新状态 参阅 Section：RCC_HSE 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	如果 HSE 被直接或者通过 PLL 用于系统时钟，那么它不能被停振
被调用函数	无

RCC_HSE

该参数设置了HSE的状态（见Table 340.）。

Table 340. RCC_HSE 定义

RCC_HSE	描述
RCC_HSE_OFF	HSE 晶振 OFF
RCC_HSE_ON	HSE 晶振 ON
RCC_HSE_Bypass	HSE 晶振被外部时钟旁路

例：

```
/* Enable the HSE */
RCC_HSEConfig(RCC_HSE_ON);
```

15.2.3 函数RCC_WaitForHSEStartUp

Table 341. 描述了函数 RCC_WaitForHSEStartUp

Table 341. 函数 RCC_WaitForHSEStartUp

函数名	RCC_WaitForHSEStartUp
函数原形	ErrorStatus RCC_WaitForHSEStartUp(void)
功能描述	等待 HSE 起振 该函数将等待直到 HSE 就绪，或者在超时的情况下退出
输入参数	无
输出参数	无
返回值	一个 ErrorStatus 枚举值： SUCCESS：HSE 晶振稳定且就绪 ERROR：HSE 晶振未就绪
先决条件	无
被调用函数	无

例：

```
ErrorStatus HSEStartUpStatus;
/* Enable HSE */
RCC_HSEConfig(RCC_HSE_ON);
/* Wait till HSE is ready and if Time out is reached exit */
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 函数RCC_AdjustHSICalibrationValue

Table 342. 描述了函数 RCC_AdjustHSICalibrationValue

Table 342. 函数 RCC_AdjustHSICalibrationValue

函数名	RCC_AdjustHSICalibrationValue
函数原形	void RCC_AdjustHSICalibrationValue(u8 HSICalibrationValue)
功能描述	调整内部高速晶振（HSI）校准值
输入参数	HSICalibrationValue： 校准补偿值 该参数取值必须在 0 到 0x1F 之间
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Set HSI calibration value to c0x1F (maximum) */
RCC_AdjustHSICalibrationValue(0x1F);
```

15.2.5 函数RCC_HSICmd

Table 343. 描述了函数 RCC_HSICmd

Table 343. 函数 RCC_HSICmd

函数名	RCC_HSICmd
函数原形	void RCC_HSICmd(FunctionalState NewState)
功能描述	使能或者失能内部高速晶振（HSI）
输入参数	NewState：HSI 新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	如果 HSI 被直接或者通过 PLL 用于系统时钟，或者 FLASH 编写操作进行中，那么它不能被停振
被调用函数	无

例：

```
/* Enable Internal High Speed oscillator */
RCC_HSICmd(ENABLE);
```

15.2.6 函数RCC_PLLConfig

Table 344. 描述了函数 RCC_PLLConfig

Table 344. 函数 RCC_PLLConfig

函数名	RCC_PLLConfig
函数原形	void RCC_PLLConfig(u32 RCC_PLLSource, u32 RCC_PLLMul)
功能描述	设置 PLL 时钟源及倍频系数
输入参数 1	RCC_PLLSource：PLL 的输入时钟源 参阅 Section：RCC_PLLSource 查阅更多该参数允许取值范围
输入参数 2	RCC_PLLMul：PLL 倍频系数 参阅 Section：RCC_PLLMul 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

RCC_PLLSource

RCC_PLLSource 用以设置 PLL 的输入时钟源。Table 345. 给出了该参数可取的值

Table 345. RCC_PLLSource 值

RCC_PLLSource	描述
RCC_PLLSource_HSI_Div2	PLL 的输入时钟 = HSI 时钟频率除以 2
RCC_PLLSource_HSE_Div1	PLL 的输入时钟 = HSE 时钟频率
RCC_PLLSource_HSE_Div2	PLL 的输入时钟 = HSE 时钟频率除以 2

RCC_PLLMul

该参数用以设置 PLL 的倍频系数。Table 346. 给出了该参数可取的值

Table 346. RCC_PLLMul 值

RCC_PLLMul	描述
RCC_PLLMul_2	PLL 输入时钟 x 2
RCC_PLLMul_3	PLL 输入时钟 x 3
RCC_PLLMul_4	PLL 输入时钟 x 4
RCC_PLLMul_5	PLL 输入时钟 x 5
RCC_PLLMul_6	PLL 输入时钟 x 6
RCC_PLLMul_7	PLL 输入时钟 x 7
RCC_PLLMul_8	PLL 输入时钟 x 8
RCC_PLLMul_9	PLL 输入时钟 x 9
RCC_PLLMul_10	PLL 输入时钟 x 10
RCC_PLLMul_11	PLL 输入时钟 x 11
RCC_PLLMul_12	PLL 输入时钟 x 12
RCC_PLLMul_13	PLL 输入时钟 x 13
RCC_PLLMul_14	PLL 输入时钟 x 14
RCC_PLLMul_15	PLL 输入时钟 x 15
RCC_PLLMul_16	PLL 输入时钟 x 16

警告：必须正确设置软件，使 PLL 输出时钟频率不超过 72 MHz

例：

```
/* Set PLL clock output to 72MHz using HSE (8MHz) as entry clock */
RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);
```

15.2.7 函数RCC_PLLCmd

Table 347. 描述了函数 RCC_PLLCmd

Table 347. 函数 RCC_PLLCmd

函数名	RCC_PLLCmd
函数原形	void RCC_PLLCmd(FunctionalState NewState)
功能描述	使能或者失能 PLL
输入参数	NewState：PLL 新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	如果 PLL 被用于系统时钟，，那么它不能被失能
被调用函数	无

例：

```
/* Enable the PLL */
RCC_PLLCmd(ENABLE);
```

15.2.8 函数RCC_SYSCLKConfig

Table 348. 描述了函数RCC_SYSCLKConfig

Table 348. 函数 RCC_SYSCLKConfig

函数名	RCC_SYSCLKConfig
函数原形	void RCC_SYSCLKConfig(u32 RCC_SYSCLKSource)
功能描述	设置系统时钟（SYSCLK）
输入参数	RCC_SYSCLKSource: 用作系统时钟的时钟源 参阅 Section：RCC_SYSCLKSource 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

RCC_SYSCLKSource

该参数设置了系统时钟（见Table 349）。

Table 349. RCC_SYSCLKSource 值

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：

```
/* Select the PLL as system clock source */
RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);
```

15.2.9 函数RCC_GetSYSCLKSource

Table 350. 描述了函数 RCC_GetSYSCLKSource

Table 350. 函数 RCC_GetSYSCLKSource

函数名	RCC_GetSYSCLKSource
函数原形	u8 RCC_GetSYSCLKSource(void)
功能描述	返回用作系统时钟的时钟源
输入参数	无
输出参数	无
返回值	用作系统时钟的时钟源： 0x00：HSI 作为系统时钟 0x04：HSE 作为系统时钟 0x08：PLL 作为系统时钟
先决条件	无
被调用函数	无

例：

```
/* Test if HSE is used as system clock */
if(RCC_GetSYSCLKSource() != 0x04)
{
}
else
{
}
```

15.2.10 函数RCC_HCLKConfig

Table 351. 描述了函数RCC_HCLKConfig

Table 351. 函数 RCC_HCLKConfig

函数名	RCC_HCLKConfig
函数原形	void RCC_HCLKConfig(u32 RCC_HCLK)
功能描述	设置 AHB 时钟（HCLK）
输入参数	RCC_HCLK: 定义 HCLK，该时钟源自系统时钟（SYSCLK） 参阅 Section：RCC_HCLK 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

RCC_HCLK

该参数设置了AHB时钟，Table 352. 给出了该参数可取的值。

Table 352. RCC_HCLK 值

RCC_HCLK	描述
RCC_SYSCLK_Div1	AHB 时钟 = 系统时钟
RCC_SYSCLK_Div2	AHB 时钟 = 系统时钟 / 2
RCC_SYSCLK_Div4	AHB 时钟 = 系统时钟 / 4
RCC_SYSCLK_Div8	AHB 时钟 = 系统时钟 / 8
RCC_SYSCLK_Div16	AHB 时钟 = 系统时钟 / 16
RCC_SYSCLK_Div64	AHB 时钟 = 系统时钟 / 64
RCC_SYSCLK_Div128	AHB 时钟 = 系统时钟 / 128
RCC_SYSCLK_Div256	AHB 时钟 = 系统时钟 / 256
RCC_SYSCLK_Div512	AHB 时钟 = 系统时钟 / 512

例：

```
/* Configure HCLK such as HCLK = SYSCLK */
RCC_HCLKConfig(RCC_SYSCLK_Div1);
```

15.2.11 函数RCC_PCLK1Config

Table 353. 描述了函数RCC_PCLK1Config

Table 353. 函数 RCC_PCLK1Config

函数名	RCC_PCLK1Config
函数原形	void RCC_PCLK1Config(u32 RCC_PCLK1)
功能描述	设置低速 AHB 时钟（PCLK1）
输入参数	RCC_PCLK1: 定义 PCLK1，该时钟源自 AHB 时钟（HCLK） 参阅 Section：RCC_PCLK1 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

RCC_PCLK1

该参数设置了低速AHB时钟（PCLK1），Table 354. 给出了该参数可取的值。

Table 354. RCC_PCLK1 值

RCC_PCLK1	描述
RCC_HCLK_Div1	APB1 时钟 = HCLK
RCC_HCLK_Div2	APB1 时钟 = HCLK / 2
RCC_HCLK_Div4	APB1 时钟 = HCLK / 4
RCC_HCLK_Div8	APB1 时钟 = HCLK / 8
RCC_HCLK_Div16	APB1 时钟 = HCLK / 16

例：

```
/* Configure PCLK1 such as PCLK1 = HCLK/2 */
RCC_PCLK1Config(RCC_HCLK_Div2);
```

15.2.12 函数RCC_PCLK2Config

Table 355. 描述了函数RCC_PCLK2Config

Table 355. 函数 RCC_PCLK2Config

函数名	RCC_PCLK2Config
函数原形	void RCC_PCLK2Config(u32 RCC_PCLK2)
功能描述	设置高速 AHB 时钟（PCLK2）
输入参数	RCC_PCLK2: 定义 PCLK2，该时钟源自 AHB 时钟（HCLK） 参阅 Section：RCC_PCLK2 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

RCC_PCLK2

该参数设置了高速AHB时钟（PCLK2），Table 356. 给出了该参数可取的值。

Table 356. RCC_PCLK1 值

RCC_PCLK2	描述
RCC_HCLK_Div1	APB2 时钟 = HCLK
RCC_HCLK_Div2	APB2 时钟 = HCLK / 2
RCC_HCLK_Div4	APB2 时钟 = HCLK / 4
RCC_HCLK_Div8	APB2 时钟 = HCLK / 8
RCC_HCLK_Div16	APB2 时钟 = HCLK / 16

例：

```
/* Configure PCLK2 such as PCLK2 = HCLK */
RCC_PCLK2Config(RCC_HCLK_Div1);
```

15.2.13 函数RCC_ITConfig

Table 357. 描述了函数RCC_ITConfig

Table 357. 函数 RCC_ITConfig

函数名	RCC_ITConfig
函数原形	void RCC_ITConfig(u8 RCC_IT, FunctionalState NewState)
功能描述	使能或者失能指定的 RCC 中断
输入参数 1	RCC_IT：待使能或者失能的 RCC 中断源 参阅 Section：RCC_IT 查阅更多该参数允许取值范围
输入参数 2	NewState：RCC 中断的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

RCC_IT

输入参数 RCC_IT 使能或者失能 RCC 的中断。可以取下表的一个或者多个取值的组合作为该参数的值。

Table 358. 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	HSI 就绪中断
RCC_IT_HSERDY	HSE 就绪中断
RCC_IT_PLLRDY	PLL 就绪中断

例：

```
/* Enable PLL Ready interrupt */
RCC_ITConfig(RCC_IT_PLLRDY, ENABLE);
```

15.2.14 函数RCC_USBCLKConfig

Table 359. 描述了函数RCC_USBCLKConfig

Table 359. 函数 RCC_USBCLKConfig

函数名	RCC_USBCLKConfig
函数原形	void RCC_USBCLKConfig(u32 RCC_USBCLKSource)
功能描述	设置 USB 时钟（USBCLK）
输入参数	RCC_USBCLKSource: 定义 USBCLK，该时钟源自 PLL 输出 参阅 Section：RCC_USBCLKSource 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

RCC_USBCLKSource

该参数设置了USB时钟（USBCLK），Table 360. 给出了该参数可取的值。

Table 360. RCC_USBCLKSource 值

RCC_USBCLKSource	描述
RCC_USBCLKSource_PLLCLK_1Div5	USB 时钟 = PLL 时钟除以 1.5
RCC_USBCLKSource_PLLCLK_Div1	USB 时钟 = PLL 时钟

例：
```
/* PLL clock divided by 1.5 used as USB clock source */
RCC_USBCLKConfig(RCC_USBCLKSource_PLLCLK_1Div5);
```

15.2.15 函数RCC_ADCCLKConfig

Table 361. 描述了函数RCC_ADCCLKConfig

Table 361. 函数 RCC_ADCCLKConfig

函数名	RCC_ADCCLKConfig
函数原形	void ADC_ADCCLKConfig(u32 RCC_ADCCLKSource)
功能描述	设置 ADC 时钟（ADCCLK）
输入参数	RCC_ADCCLKSource: 定义 ADCCLK，该时钟源自 APB2 时钟（PCLK2） 参阅 Section：RCC_ADCCLKSource 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

RCC_ADCCLKSource

该参数设置了ADC时钟（ADCCLK），Table 362. 给出了该参数可取的值。

Table 362. RCC_ADCCLKSource 值

RCC_ADCCLKSource	描述
RCC_PCLK2_Div2	ADC 时钟 = PCLK / 2
RCC_PCLK2_Div4	ADC 时钟 = PCLK / 4
RCC_PCLK2_Div6	ADC 时钟 = PCLK / 6
RCC_PCLK2_Div8	ADC 时钟 = PCLK / 8

例：
```
/* Configure ADCCLK such as ADCCLK = PCLK2/2 */
RCC_ADCCLKConfig(RCC_PCLK2_Div2);
```

15.2.16 函数RCC_LSEConfig

Table 363. 描述了函数RCC_LSEConfig

Table 363. 函数 RCC_LSEConfig

函数名	RCC_LSEConfig
函数原形	void RCC_LSEConfig(u32 RCC_HSE)
功能描述	设置外部低速晶振（LSE）
输入参数	RCC_LSE: LSE 的新状态 参阅 Section：RCC_HSE 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

RCC_LSE
该参数设置了HSE的状态（见Table 364.）。

Table 364. RCC_LSE 定义

RCC_LSE	描述
RCC_LSE_OFF	LSE 晶振 OFF
RCC_LSE_ON	LSE 晶振 ON
RCC_LSE_Bypass	LSE 晶振被外部时钟旁路

例：
```
/* Enable the LSE */
RCC_LSEConfig(RCC_LSE_ON);
```

15.2.17 函数RCC_LSICmd

Table 365. 描述了函数 RCC_LSICmd

Table 365. 函数 RCC_LSICmd

函数名	RCC_LSICmd
函数原形	void RCC_LSICmd(FunctionalState NewState)
功能描述	使能或者失能内部低速晶振（LSI）
输入参数	NewState：LSI 新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	如果 IWDG 运行的话，LSI 不能被失能
被调用函数	无

例：
```
/* Enable the Internal Low Speed oscillator */
RCC_LSICmd(ENABLE);
```

15.2.18 函数RCC_RTCCLKConfig

Table 366. 描述了函数RCC_RTCCLKConfig

Table 366. 函数 RCC_RTCCLKConfig

函数名	RCC_RTCCLKConfig
函数原形	void RCC_RTCCLKConfig(u32 RCC_RTCCLKSource)
功能描述	设置 RTC 时钟（RTCCLK）
输入参数	RCC_RTCCLKSource: 定义 RTCCLK 参阅 Section：RCC_RTCCLKSource 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	RTC 时钟一经选定即不能更改，除非复位后备域
被调用函数	无

RCC_RTCCLKSource

该参数设置了RTC时钟（RTCCLK），Table 367. 给出了该参数可取的值。

Table 367. 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 时钟频率除以 128 作为 RTC 时钟

例：

```
/* Select the LSE as RTC clock source */
RCC_RTCCLKConfig(RCC_RTCCLKSource_LSE);
```

15.2.19 函数RCC_RTCCLKCmd

Table 368. 描述了函数 RCC_RTCCLKCmd

Table 368. 函数 RCC_RTCCLKCmd

函数名	RCC_RTCCLKCmd
函数原形	void RCC_RTCCLKCmd(FunctionalState NewState)
功能描述	使能或者失能 RTC 时钟
输入参数	NewState：RTC 时钟的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	该函数只有在通过函数 RCC_RTCCLKConfig 选择 RTC 时钟后，才能调用
被调用函数	无

例：

```
/* Enable the RTC clock */
RCC_RTCCLKCmd(ENABLE);
```

15.2.20 函数RCC_GetClocksFreq

Table 369. 描述了函数 RCC_GetClocksFreq

Table 369. 函数 RCC_GetClocksFreq

函数名	RCC_GetClocksFreq
函数原形	void RCC_GetClocksFreq(RCC_ClocksTypeDef* RCC_Clocks)
功能描述	返回不同片上时钟的频率
输入参数	RCC_Clocks：指向结构 RCC_ClocksTypeDef 的指针，包含了各个时钟的频率 参阅 Section：RCC_Clocks 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

RCC_ClocksTypeDef structure

RCC_ClocksTypeDef 定义于文件“stm32f10x_rcc.h”：

```
typedef struct
{
u32 SYSCLK_Frequency;
u32 HCLK_Frequency;
u32 PCLK1_Frequency;
u32 PCLK2_Frequency;
u32 ADCCLK_Frequency;
}RCC_ClocksTypeDef;
```

SYSCLK_Frequency

该成员返回 SYSCLK 的频率，单位 Hz

HCLK_Frequency

该成员返回 HCLK 的频率，单位 Hz

PCLK1_Frequency

该成员返回 PCLK1 的频率，单位 Hz

PCLK2_Frequency

该成员返回 PCLK2 的频率，单位 Hz

ADCCLK_Frequency

该成员返回 ADCCLK 的频率，单位 Hz

例：

```
/* Get the frequencies of different on chip clocks */
RCC_ClocksTypeDef RCC_Clocks;
RCC_GetClocksFreq(&RCC_Clocks);
```

15.2.21 函数RCC_AHBPeriphClockCmd

Table 370. 描述了函数RCC_AHBPeriphClockCmd

Table 370. 函数 RCC_AHBPeriphClockCmd

函数名	RCC_AHBPeriphClockCmd
函数原形	void RCC_AHBPeriphClockCmd(u32 RCC_AHBPeriph, FunctionalState NewState)
功能描述	使能或者失能 AHB 外设时钟
输入参数 1	RCC_AHBPeriph: 门控 AHB 外设时钟 参阅 Section：RCC_AHBPeriph 查阅更多该参数允许取值范围
输入参数 2	NewState：指定外设时钟的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

RCC_AHBPeriph

该参数被门控的AHB外设时钟，可以取下表的一个或者多个取值的组合作为该参数的值。

Table 371. 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

1. SRAM 和 FLITF 时钟只能在睡眠（SLEEP）模式下被失能。

例：

```
/* Enable DMA clock */
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA);
```

15.2.22 函数RCC_APB2PeriphClockCmd

Table 372. 描述了函数RCC_APB2PeriphClockCmd

Table 372. 函数 RCC_APB2PeriphClockCmd

函数名	RCC_APB2PeriphClockCmd
函数原形	void RCC_APB2PeriphClockCmd(u32 RCC_APB2Periph, FunctionalState NewState)
功能描述	使能或者失能 APB2 外设时钟
输入参数 1	RCC_APB2Periph: 门控 APB2 外设时钟 参阅 Section：RCC_APB2Periph 查阅更多该参数允许取值范围
输入参数 2	NewState：指定外设时钟的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

RCC_APB2Periph

该参数被门控的APB2外设时钟，可以取下表的一个或者多个取值的组合作为该参数的值。

Table 373. RCC_AHB2Periph 值

RCC_AHB2Periph	描述
RCC_APB2Periph_AFIO	功能复用 IO 时钟
RCC_APB2Periph_GPIOA	GPIOA 时钟
RCC_APB2Periph_GPIOB	GPIOB 时钟
RCC_APB2Periph_GPIOC	GPIOC 时钟
RCC_APB2Periph_GPIOD	GPIOD 时钟
RCC_APB2Periph_GPIOE	GPIOE 时钟
RCC_APB2Periph_ADC1	ADC1 时钟
RCC_APB2Periph_ADC2	ADC2 时钟
RCC_APB2Periph_TIM1	TIM1 时钟
RCC_APB2Periph_SPI1	SPI1 时钟
RCC_APB2Periph_USART1	USART1 时钟
RCC_APB2Periph_ALL	全部 APB2 外设时钟

例：

```
/* Enable GPIOA, GPIOB and SPI1 clocks */
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB |
RCC_APB2Periph_SPI1, ENABLE);
```

15.2.23 函数RCC_APB1PeriphClockCmd

Table 374. 描述了函数RCC_APB1PeriphClockCmd

Table 374. 函数 RCC_APB1PeriphClockCmd

函数名	RCC_APB1PeriphClockCmd
函数原形	void RCC_APB1PeriphClockCmd(u32 RCC_APB1Periph, FunctionalState NewState)
功能描述	使能或者失能 APB1 外设时钟
输入参数 1	RCC_APB1Periph：门控 APB1 外设时钟 参阅 Section：RCC_APB1Periph 查阅更多该参数允许取值范围
输入参数 2	NewState：指定外设时钟的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

RCC_APB1Periph

该参数被门控的APB1外设时钟，可以取下表的一个或者多个取值的组合作为该参数的值。

Table 375. RCC_AHB1Periph 值

RCC_AHB1Periph	描述
RCC_APB1Periph_TIM2	TIM2 时钟
RCC_APB1Periph_TIM3	TIM3 时钟
RCC_APB1Periph_TIM4	TIM4 时钟
RCC_APB1Periph_WWDG	WWDG 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	BKP 时钟
RCC_APB1Periph_PWR	PWR 时钟
RCC_APB1Periph_ALL	全部 APB1 外设时钟

例：
```
/* Enable BKP and PWR clocks */
RCC_APB1PeriphClockCmd(RCC_APB1Periph_BKP | RCC_APB1Periph_PWR,
ENABLE);
```

15.2.24 函数RCC_APB2PeriphResetCmd

Table 376. 描述了函数RCC_APB2PeriphResetCmd

Table 376. 函数 RCC_APB2PeriphResetCmd

函数名	RCC_APB2PeriphResetCmd
函数原形	void RCC_APB2PeriphResetCmd(u32 RCC_APB2Periph, FunctionalState NewState)
功能描述	强制或者释放高速 APB（APB2）外设复位
输入参数 1	RCC_APB2Periph: APB2 外设复位 参阅 Section：RCC_APB2Periph 查阅更多该参数允许取值范围
输入参数 2	NewState：指定 APB2 外设复位的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：
```
/* Enter the SPI1 peripheral to reset */
RCC_APB2PeriphResetCmd(RCC_APB2Periph_SPI1, ENABLE);
/* Exit the SPI1 peripheral from reset */
RCC_APB2PeriphResetCmd(RCC_APB2Periph_SPI1, DISABLE);
```

15.2.25 函数RCC_APB1PeriphResetCmd

Table 377. 描述了函数RCC_APB1PeriphResetCmd

Table 377. 函数 RCC_APB1PeriphResetCmd

函数名	RCC_APB1PeriphResetCmd
函数原形	void RCC_APB1PeriphResetCmd(u32 RCC_APB1Periph, FunctionalState NewState)
功能描述	强制或者释放低速 APB（APB1）外设复位
输入参数 1	RCC_APB1Periph: APB1 外设复位 参阅 Section：RCC_APB1Periph 查阅更多该参数允许取值范围
输入参数 2	NewState：指定 APB1 外设复位的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：
```
/* Enter the SPI2 peripheral to reset */
RCC_APB1PeriphResetCmd(RCC_APB1Periph_SPI2, ENABLE);
/* Exit the SPI2 peripheral from reset */
RCC_APB1PeriphResetCmd(RCC_APB1Periph_SPI2, DISABLE);
```

15.2.26 函数RCC_BackupResetCmd

Table 378. 描述了函数RCC_BackupResetCmd

Table 378. 函数 RCC_BackupResetCmd

函数名	RCC_BackupResetCmd
函数原形	void RCC_BackupResetCmd(FunctionalState NewState)
功能描述	强制或者释放后备域复位
输入参数	NewState：后备域复位的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Reset the entire Backup domain */
RCC_BackupResetCmd(ENABLE);
```

15.2.27 函数RCC_ClockSecuritySystemCmd

Table 379. 描述了函数RCC_ClockSecuritySystemCmd

Table 379. 函数 RCC_ClockSecuritySystemCmd

函数名	RCC_ClockSecuritySystemCmd
函数原形	void RCC_ClockSecuritySystemCmd(FunctionalState NewState)
功能描述	使能或者失能时钟安全系统
输入参数	NewState：时钟安全系统的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable the Clock Security System */
RCC_ClockSecuritySystemCmd(ENABLE);
```

15.2.28 函数RCC_MCOConfig

Table 380. 描述了函数RCC_MCOConfig

Table 380. 函数 RCC_MCOConfig

函数名	RCC_MCOConfig
函数原形	void RCC_MCOConfig(u8 RCC_MCO)
功能描述	选择在 MCO 管脚上输出的时钟源
输入参数	RCC_MCO: 指定输出的时钟源 参阅 Section：RCC_MCO 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

RCC_MCO

该参数设置了指定输出的时钟源，Table 381. 给出了该参数可取的值。

Table 381. RCC_MCO 值

RCC_MCO	描述
RCC_MCO_NoClock	无时钟被选中
RCC_MCO_SYSCLK	选中系统时钟
RCC_MCO_HSI	选中 HSI
RCC_MCO_HSE	选中 HSE
RCC_MCO_PLLCLK_Div2	选中 PLL 时钟除以 2

警告：当选中系统时钟作为 MCO 管脚的输出时，注意它的时钟频率不超过 50MHz(最大 I/O 速率)。

例：

```
/* Output PLL clock divided by 2 on MCO pin */
RCC_MCOConfig(RCC_MCO_PLLCLK_Div2);
```

15.2.29 函数RCC_GetFlagStatus

Table 382. 描述了函数RCC_ GetFlagStatus

Table 382. 函数 RCC_ GetFlagStatus

函数名	RCC_ GetFlagStatus
函数原形	FlagStatus RCC_GetFlagStatus(u8 RCC_FLAG)
功能描述	检查指定的 RCC 标志位设置与否
输入参数	RCC_FLAG：待检查的 RCC 标志位 参阅 Section：RCC_FLAG 查阅更多该参数允许取值范围
输出参数	无
返回值	RCC_FLAG 的新状态（SET 或者 RESET）
先决条件	无
被调用函数	无

RCC_FLAG

Table 383. 给出了所有可以被函数RCC_ GetFlagStatus检查的标志位列表

Table 383. RCC_FLAG 值

RCC_FLAG	描述
RCC_FLAG_HSIRDY	HSI 晶振就绪
RCC_FLAG_HSERDY	HSE 晶振就绪
RCC_FLAG_PLLRDY	PLL 就绪
RCC_FLAG_LSERDY	LSI 晶振就绪
RCC_FLAG_LSIRDY	LSE 晶振就绪
RCC_FLAG_PINRST	管脚复位
RCC_FLAG_PORRST	POR/PDR 复位
RCC_FLAG_SFTRST	软件复位
RCC_FLAG_IWDGRST	IWDG 复位
RCC_FLAG_WWDGRST	WWDG 复位
RCC_FLAG_LPWRRST	低功耗复位

例：

```
/* Test if the PLL clock is ready or not */
FlagStatus Status;
Status = RCC_GetFlagStatus(RCC_FLAG_PLLRDY);
if(Status == RESET)
{
...
}
else
```

```
{
...
}
```

15.2.30 函数RCC_ClearFlag

Table 384. 描述了函数RCC_ ClearFlag

Table 384. 函数 RCC_ ClearFlag

函数名	RCC_ ClearFlag
函数原形	void RCC_ClearFlag(void)
功能描述	清除 RCC 的复位标志位
输入参数	RCC_FLAG：清除的 RCC 复位标志位 可以清除的复位标志位有： RCC_FLAG_PINRST, RCC_FLAG_PORRST, RCC_FLAG_SFTRST, RCC_FLAG_IWDGRST, RCC_FLAG_WWDGRST, RCC_FLAG_LPWRRST
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear the reset flags */
RCC_ClearFlag();
```

15.2.31 函数RCC_GetITStatus

Table 385. 描述了函数RCC_ GetITStatus

Table 385. 函数 RCC_ GetITStatus

函数名	RCC_ GetITStatus
函数原形	ITStatus RCC_GetITStatus(u8 RCC_IT)
功能描述	检查指定的 RCC 中断发生与否
输入参数	RCC_IT：待检查的 RCC 中断源 参阅 Section：RCC_IT 查阅更多该参数允许取值范围
输出参数	无
返回值	RCC_IT 的新状态
先决条件	无
被调用函数	无

RCC_IT

Table 386. 给出了所有可以被函数RCC_ GetITStatus检查的中断标志位列表

Table 386. RCC_IT 值

RCC_IT	描述
RCC_IT_LSIRDY	LSI 晶振就绪中断
RCC_IT_LSERDY	LSE 晶振就绪中断
RCC_IT_HSIRDY	HSI 晶振就绪中断
RCC_IT_HSERDY	HSE 晶振就绪中断
RCC_IT_PLLRDY	PLL 就绪中断
RCC_IT_CSS	时钟安全系统中断

例：

```
/* Test if the PLL Ready interrupt has occurred or not */
ITStatus Status;
Status = RCC_GetITStatus(RCC_IT_PLLRDY);
```

```
if(Status == RESET)
{
...
}
else
{
...
}
```

15.2.32 函数RCC_ClearITPendingBit

Table 387. 描述了函数RCC_ClearITPendingBit

Table 387. 函数 RCC_ClearITPendingBit

函数名	RCC_ClearITPendingBit
函数原形	void RCC_ClearITPendingBit(u8 RCC_IT)
功能描述	清除 RCC 的中断待处理位
输入参数	RCC_IT：待检查的 RCC 中断源 参阅 Section：RCC_IT 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

RCC_IT

Table 388. 给出了所有可以被函数RCC_ ClearITPendingBit清除的中断待处理位列表

Table 388. RCC_IT 值

RCC_IT	描述
RCC_IT_LSIRDY	LSI 晶振就绪中断
RCC_IT_LSERDY	LSE 晶振就绪中断
RCC_IT_HSIRDY	HSI 晶振就绪中断
RCC_IT_HSERDY	HSE 晶振就绪中断
RCC_IT_PLLRDY	PLL 就绪中断
RCC_IT_CSS	时钟安全系统中断

例：

```
/* Clear the PLL Ready interrupt pending bit */
RCC_ClearITPendingBit(RCC_IT_PLLRDY);
```

16 实时时钟（RTC）

RTC 提供了一系列连续工作的计数器，配合适当的软件，具有提供时钟-日历的功能。写入计数器的值可以设置整个系统的时间/日期。

Section 16.1 RTC 寄存器结构描述了固件函数库所使用的数据结构，Section 16.2 固件库函数介绍了函数库里的所有函数。

16.1 RTC寄存器结构

寄存器结构，***RTC_TypeDeff***，在文件***"stm32f10x_map.h"***中定义如下：

```
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;
```

Table 389.例举了RTC所有寄存器

Table 389. RTC 寄存器

寄存器	描述
CRH	控制寄存器高位
CRL	控制寄存器低位
PRLH	预分频装载寄存器高位
PRLL	预分频装载寄存器低位
DIVH	预分频分频因子寄存器高位
DIVL	预分频分频因子寄存器低位
CNTH	计数器寄存器高位
CNTL	计数器寄存器低位
ALRH	闹钟寄存器高位
ALRL	闹钟寄存器低位

RTC 外设声明于文件***"stm32f10x_map.h"***：

```
...
#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
```

使用Debug模式时，初始化指针***RTC***于文件***"stm32f10x_lib.c"***：

```
#ifdef _RTC
RTC = (RTC_TypeDef *) RTC_BASE;
#endif /*_RTC */
```

为了访问RTC寄存器，,***_RTC***必须在文件***"stm32f10x_conf.h"***中定义如下：

```
#define _RTC
```

16.2 RTC库函数

Table 390. 例举了RTC的库函数

Table 390. RTC 库函数

函数名	描述
RTC_ITConfig	使能或者失能指定的 RTC 中断
RTC_EnterConfigMode	进入 RTC 配置模式
RTC_ExitConfigMode	退出 RTC 配置模式
RTC_GetCounter	获取 RTC 计数器的值
RTC_SetCounter	设置 RTC 计数器的值
RTC_SetPrescaler	设置 RTC 预分频的值
RTC_SetAlarm	设置 RTC 闹钟的值
RTC_GetDivider	获取 RTC 预分频分频因子的值
RTC_WaitForLastTask	等待最近一次对 RTC 寄存器的写操作完成
RTC_WaitForSynchro	等待 RTC 寄存器(RTC_CNT, RTC_ALR and RTC_PRL)与 RTC 的 APB 时钟同步
RTC_GetFlagStatus	检查指定的 RTC 标志位设置与否
RTC_ClearFlag	清除 RTC 的待处理标志位
RTC_GetITStatus	检查指定的 RTC 中断发生与否
RTC_ClearITPendingBit	清除 RTC 的中断待处理位

16.2.1 函数RTC_ITConfig

Table 391. 描述了函数RTC_ITConfig

Table 391. 函数 RTC_ITConfig

函数名	RTC_ITConfig
函数原形	void RTC_ITConfig(u16 RTC_IT, FunctionalState NewState)
功能描述	使能或者失能指定的 RTC 中断
输入参数 1	RTC_IT：待使能或者失能的 RTC 中断源 参阅 Section：RTC_IT 查阅更多该参数允许取值范围
输入参数 2	NewState：RTC 中断的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	在使用本函数前必须先调用函数 RTC_WaitForLastTask()，等待标志位 RTOFF 被设置
被调用函数	无

RTC_IT

输入参数 RTC_IT 使能或者失能 RTC 的中断。可以取下表的一个或者多个取值的组合作为该参数的值。

Table 392. RTC_IT 值

RTC_IT	描述
RTC_IT_OW	溢出中断使能
RTC_IT_ALR	闹钟中断使能
RTC_IT_SEC	秒中断使能

例：

```
/* Wait until last write operation on RTC registers is terminated */
RTC_WaitForLastTask();
/* Alarm interrupt enabled */
RTC_ITConfig(RTC_IT_ALR, ENABLE);
```

16.2.2 函数RTC_EnterConfigMode

Table 393. 描述了函数RTC_EnterConfigMode

Table 393. 函数 RTC_EnterConfigMode

函数名	RTC_EnterConfigMode
函数原形	void RTC_EnterConfigMode(void)
功能描述	进入 RTC 配置模式
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable the configuration mode */
RTC_EnterConfigMode();
```

16.2.3 函数RTC_ExitConfigMode

Table 394. 描述了函数RTC_ExitConfigMode

Table 394. 函数 RTC_ExitConfigMode

函数名	RTC_ExitConfigMode
函数原形	void RTC_ExitConfigMode(void)
功能描述	退出 RTC 配置模式
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Exit the configuration mode */
RTC_ExitConfigMode();
```

16.2.4 函数RTC_GetCounter

Table 395. 描述了函数RTC_GetCounter

Table 395. 函数 RTC_GetCounter

函数名	RTC_GetCounter
函数原形	u32 RTC_GetCounter(void)
功能描述	获取 RTC 计数器的值
输入参数	无
输出参数	无
返回值	RTC 计数器的值
先决条件	无
被调用函数	无

例：

```
/* Gets the counter value */
u32 RTCCounterValue;
RTCCounterValue = RTC_GetCounter();
```

16.2.5 函数RTC_SetCounter

Table 396. 描述了函数RTC_SetCounter

Table 396. 函数 RTC_SetCounter

函数名	RTC_SetCounter
函数原形	void RTC_SetCounter(u32 CounterValue)
功能描述	设置 RTC 计数器的值
输入参数	CounterValue：新的 RTC 计数器值
输出参数	无
返回值	无
先决条件	在使用本函数前必须先调用函数 RTC_WaitForLastTask()，等待标志位 RTOFF 被设置
被调用函数	RTC_EnterConfigMode() RTC_ExitConfigMode()

例：

```
/* Wait until last write operation on RTC registers is terminated */
RTC_WaitForLastTask();
/* Sets Counter value to 0xFFFF5555 */
RTC_SetCounter(0xFFFF5555);
```

16.2.6 函数RTC_SetPrescaler

Table 397. 描述了函数RTC_SetPrescaler

Table 397. 函数 RTC_SetPrescaler

函数名	RTC_SetPrescaler
函数原形	void RTC_SetPrescaler(u32 PrescalerValue)
功能描述	设置 RTC 预分频的值
输入参数	PrescalerValue：新的 RTC 预分频值
输出参数	无
返回值	无
先决条件	在使用本函数前必须先调用函数 RTC_WaitForLastTask()，等待标志位 RTOFF 被设置
被调用函数	RTC_EnterConfigMode() RTC_ExitConfigMode()

例：

```
/* Wait until last write operation on RTC registers is terminated */
RTC_WaitForLastTask();
/* Sets Prescaler value to 0x7A12 */
RTC_SetPrescaler(0x7A12);
```

16.2.7 函数RTC_SetAlarm

Table 398. 描述了函数RTC_ SetAlarm

Table 398. 函数 RTC_ SetAlarm

函数名	RTC_ SetAlarm
函数原形	void RTC_SetAlarm(u32 AlarmValue)
功能描述	设置 RTC 闹钟的值
输入参数	AlarmValue：新的 RTC 闹钟值
输出参数	无
返回值	无
先决条件	在使用本函数前必须先调用函数 RTC_WaitForLastTask()，等待标志位 RTOFF 被设置
被调用函数	RTC_EnterConfigMode() RTC_ExitConfigMode()

例：

```
/* Wait until last write operation on RTC registers is terminated */
RTC_WaitForLastTask();
/* Sets Alarm value to 0xFFFFFFFA */
RTC_SetAlarm(0xFFFFFFFA);
```

16.2.8 函数RTC_GetDivider

Table 399. 描述了函数RTC_GetDivider

Table 399. 函数 RTC_GetDivider

函数名	RTC_GetDivider
函数原形	u32 RTC_GetDivider(void)
功能描述	获取 RTC 预分频分频因子的值
输入参数	无
输出参数	无
返回值	RTC 预分频分频因子的值
先决条件	无
被调用函数	无

例：

```
/* Gets the current RTC Divider value */
u32 RTCDividerValue;
RTCDividerValue = RTC_GetDivider();
```

16.2.9 函数RTC_WaitForLastTask

Table 400. 描述了函数RTC_WaitForLastTask

Table 400. 函数 RTC_WaitForLastTask

函数名	RTC_WaitForLastTask
函数原形	void RTC_WaitForLastTask(void)
功能描述	等待最近一次对 RTC 寄存器的写操作完成
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Wait until last write operation on RTC registers is terminated */
RTC_WaitForLastTask();
/* Sets Alarm value to 0x10 */
RTC_SetAlarm(0x10);
```

16.2.10 函数RTC_WaitForSynchro

Table 401. 描述了函数RTC_WaitForSynchro

Table 401. 函数 RTC_WaitForSynchro

函数名	RTC_WaitForSynchro
函数原形	void RTC_WaitForSynchro(void)
功能描述	等待最近一次对 RTC 寄存器的写操作完成
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Wait until the RTC registers are synchronized with RTC APB clock */
RTC_WaitForSynchro();
```

16.2.11 函数RTC_GetFlagStatus

Table 402. 描述了函数RTC_ GetFlagStatus

Table 402. 函数 RTC_ GetFlagStatus

函数名	RTC_ GetFlagStatus
函数原形	FlagStatus RTC_GetFlagStatus(u16 RTC_FLAG)
功能描述	检查指定的 RTC 标志位设置与否
输入参数 2	RTC_FLAG：待检查的 RTC 标志位 参阅 Section：RTC_FLAG 查阅更多该参数允许取值范围
输出参数	无
返回值	RTC_FLAG 的新状态（SET 或者 RESET）
先决条件	无
被调用函数	无

RTC_FLAG

Table 403. 给出了所有可以被函数RTC_ GetFlagStatus检查的标志位列表

Table 403. RTC_FLAG 值

RTC_FLAG	描述
RTC_FLAG_RTOFF	RTC 操作 OFF 标志位
RTC_FLAG_RSF	寄存器已同步标志位
RTC_FLAG_OW	溢出中断标志位
RTC_FLAG_ALR	闹钟中断标志位
RTC_FLAG_SEC	秒中断标志位

例：

```
/* Gets the RTC overflow interrupt status */
FlagStatus OverrunFlagStatus;
OverrunFlagStatus = RTC_GetFlagStatus(RTC_Flag_OW);
```

16.2.12 函数RTC_ClearFlag

Table 404. 描述了函数RTC_ClearFlag

Table 404. 函数 RTC_ClearFlag

函数名	RTC_ClearFlag
函数原形	void RTC_ClearFlag(u16 RTC_FLAG)
功能描述	清除 RTC 的待处理标志位
输入参数	RTC_FLAG：待清除的 RTC 标志位 参阅 Section：I2C_FLAG 查阅更多该参数允许取值范围 注意：标志位 RTC_FLAG_RTOFF 不能用软件清除，标志位 RTC_FLAG_RSF 只有在 APB 复位，或者 APB 时钟停止后，才可以清除
输出参数	无
返回值	无
先决条件	在使用本函数前必须先调用函数 RTC_WaitForLastTask()，等待标志位 RTOFF 被设置
被调用函数	无

例：

```
/* Wait until last write operation on RTC registers is terminated */
RTC_WaitForLastTask();
/* Clears the RTC overflow flag */
RTC_ClearFlag(RTC_FLAG_OW);
```

16.2.13 函数RTC_GetITStatus

Table 405. 描述了函数RTC_GetITStatus

Table 405. 函数 RTC_GetITStatus

函数名	RTC_GetITStatus
函数原形	ITStatus RTC_GetITStatus(u16 RTC_IT)
功能描述	检查指定的 RTC 中断发生与否
输入参数 2	RTC_IT：待检查的 RTC 中断 参阅 Section：RTC_IT 查阅更多该参数允许取值范围
输出参数	无
返回值	RTC_IT 的新状态（SET 或者 RESET）
先决条件	无
被调用函数	无

例：

```
/* Gets the RTC Second interrupt status */
ITStatus SecondITStatus;
SecondITStatus = RTC_GetITStatus(RTC_IT_SEC);
```

16.2.14 函数RTC_ClearITPendingBit

Table 406. 描述了函数RTC_ClearITPendingBit

Table 406. 函数 RTC_ClearITPendingBit

函数名	RTC_ClearITPendingBit
函数原形	ITStatus RTC_GetITStatus(u16 RTC_IT)
功能描述	清除 RTC 的中断待处理位
输入参数 2	RTC_IT：待清除的 RTC 中断待处理位 参阅 Section：RTC_IT 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	在使用本函数前必须先调用函数 RTC_WaitForLastTask()，等待标志位 RTOFF 被设置
被调用函数	无

例：

```
/* Wait until last write operation on RTC registers is terminated */
RTC_WaitForLastTask();
/* Clears the RTC Second interrupt */
RTC_ClearITPendingBit(RTC_IT_SEC);
```

17 串行外设接口（SPI）

串行外设接口（SPI）提供与外部设备进行同步串行通讯的功能。接口可以被设置工作在主模式或者从模式。

Section 17.1 SPI 寄存器结构描述了固件函数库所使用的数据结构，Section 17.2 固件库函数介绍了函数库里的所有函数。

17.1 SPI寄存器结构

SPI 寄存器结构，***SPI_TypeDeff***，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;
```

Table 407.例举了SPI所有寄存器

Table 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 CRC 多项式寄存器
RxCRCR	SPI 接收 CRC 寄存器
TxCRCR	SPI 发送 CRC 寄存器

2 个 SPI 外设声明于文件***"stm32f10x_map.h"***：

```
...
#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
```

使用Debug模式时，初始化指针***SPI1, SPI2***于文件***"stm32f10x_lib.c"***：

```
...
#ifdef _SPI1
SPI1 = (SPI_TypeDef *) SPI1_BASE;
#endif /*_SPI1 */
#ifdef _SPI2
SPI2 = (SPI_TypeDef *) SPI2_BASE;
#endif /*_SPI2 */
...
```

为了访问 SPI 寄存器，,***_SPI，_SPI1, _SPI2*** 必须在文件***"stm32f10x_conf.h"***中定义如下：

```
...
#define _SPI
#define _SPI1
#define _SPI2
...
```

17.2 SPI库函数

Table 408. 例举了SPI的库函数

Table 408. SPI 库函数

函数名	描述
SPI_DeInit	将外设 SPIx 寄存器重设为缺省值
SPI_Init	根据 SPI_InitStruct 中指定的参数初始化外设 SPIx 寄存器
SPI_StructInit	把 SPI_InitStruct 中的每一个参数按缺省值填入
SPI_Cmd	使能或者失能 SPI 外设
SPI_ITConfig	使能或者失能指定的 SPI 中断
SPI_DMACmd	使能或者失能指定 SPI 的 DMA 请求
SPI_SendData	通过外设 SPIx 发送一个数据
SPI_ReceiveData	返回通过 SPIx 最近接收的数据
SPI_DMALastTransferCmd	使下一次 DMA 传输为最后一次传输
SPI_NSSInternalSoftwareConfig	为选定的 SPI 软件配置内部 NSS 管脚
SPI_SSOutputCmd	使能或者失能指定的 SPI SS 输出
SPI_DataSizeConfig	设置选定的 SPI 数据大小
SPI_TransmitCRC	发送 SPIx 的 CRC 值
SPI_CalculateCRC	使能或者失能指定 SPI 的传输字 CRC 值计算
SPI_GetCRC	返回指定 SPI 的发送或者接受 CRC 寄存器值
SPI_GetCRCPolynomial	返回指定 SPI 的 CRC 多项式寄存器值
SPI_BiDirectionalLineConfig	选择指定 SPI 在双向模式下的数据传输方向
SPI_GetFlagStatus	检查指定的 SPI 标志位设置与否
SPI_ClearFlag	清除 SPIx 的待处理标志位
SPI_GetITStatus	检查指定的 SPI 中断发生与否
SPI_ClearITPendingBit	清除 SPIx 的中断待处理位

17.2.1 函数SPI_DeInit

Table 409. 描述了函数 SPI_DeInit

Table 409. 函数 SPI_DeInit

函数名	SPI_DeInit
函数原形	void SPI_DeInit(SPI_TypeDef* SPIx)
功能描述	将外设 SPIx 寄存器重设为缺省值
输入参数	SPIx：x 可以是 1 或者 2，来选择 SPI 外设
输出参数	无
返回值	无
先决条件	无
被调用函数	对 SPI1，RCC_APB2PeriphClockCmd(). 对 SPI2，RCC_APB1PeriphClockCmd().

例：

```
/* Deinitialize the SPI2 */
SPI_DeInit(SPI2);
```

17.2.2 函数SPI_Init

Table 410. 描述了函数 SPI_Init

Table 410. 函数 SPI_Init

函数名	SPI_Init
函数原形	void SPI_Init(SPI_TypeDef* SPIx, SPI_InitTypeDef* SPI_InitStruct)
功能描述	根据 SPI_InitStruct 中指定的参数初始化外设 SPIx 寄存器
输入参数 1	SPIx：x 可以是 1 或者 2，来选择 SPI 外设
输入参数 2	SPI_InitStruct：指向结构 SPI_InitTypeDef 的指针，包含了外设 SPI 的配置信息 参阅 Section：SPI_InitTypeDef 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

SPI_InitTypeDef structure

SPI_InitTypeDef 定义于文件“stm32f10x_spi.h”：

```
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;
```

SPI_Direction

SPI_Dirention 设置了 SPI 单向或者双向的数据模式。见 Table 411. 查阅该参数可取的值。

Table 411. SPI_Mode 值

SPI_Mode	描述
SPI_Direction_2Lines_FullDuplex	SPI 设置为双线双向全双工
SPI_Direction_2Lines_RxOnly	SPI 设置为双线单向接收
SPI_Direction_1Line_Rx	SPI 设置为单线双向接收
SPI_Direction_1Line_Tx	SPI 设置为单线双向发送

SPI_Mode

SPI_Mode 设置了 SPI 工作模式。见 Table 412. 查阅该参数可取的值。

Table 412. SPI_Mode 值

SPI_Mode	描述
SPI_Mode_Master	设置为主 SPI
SPI_Mode_Slave	设置为从 SPI

SPI_DataSize

SPI_DataSize 设置了 SPI 的数据大小。见 Table 413. 查阅该参数可取的值。

Table 413. SPI_DataSize 值

SPI_DataSize	描述
SPI_DataSize_16b	SPI 发送接收 16 位帧结构
SPI_DataSize_8b	SPI 发送接收 8 位帧结构

SPI_CPOL

SPI_CPOL 选择了串行时钟的稳态。见 Table 414. 查阅该参数可取的值。

Table 414. SPI_ SPI_CPOL 值

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

SPI_CPHA

SPI_CPHA 设置了位捕获的时钟活动沿。见 Table 415. 查阅该参数可取的值。

Table 415. SPI_SPI_CPHA 值

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

SPI_NSS

SPI_NSS 指定了 NSS 信号由硬件（NSS 管脚）还是软件（使用 SSI 位）管理。见 Table 416. 查阅该参数可取的值。

Table 416. SPI_NSS 值

SPI_NSS	描述
SPI_NSS_Hard	NSS 由外部管脚管理
SPI_NSS_Soft	内部 NSS 信号有 SSI 位控制

SPI_BaudRatePrescaler

SPI_BaudRatePrescaler 用来定义波特率预分频的值，这个值用以设置发送和接收的 SCK 时钟。见 Table 417. 查阅该参数可取的值。

Table 417. SPI_BaudRatePrescaler 值

SPI_NSS	描述
SPI_BaudRatePrescaler2	波特率预分频值为 2
SPI_BaudRatePrescaler4	波特率预分频值为 4

SPI_BaudRatePrescaler8	波特率预分频值为 8
SPI_BaudRatePrescaler16	波特率预分频值为 16
SPI_BaudRatePrescaler32	波特率预分频值为 32
SPI_BaudRatePrescaler64	波特率预分频值为 64
SPI_BaudRatePrescaler128	波特率预分频值为 128
SPI_BaudRatePrescaler256	波特率预分频值为 256

注意：通讯时钟由主 SPI 的时钟分频而得，不需要设置从 SPI 的时钟。

SPI_FirstBit

SPI_FirstBit 指定了数据传输从 MSB 位还是 LSB 位开始。见 Table 418. 查阅该参数可取的值。

Table 418. SPI_FirstBit 值

SPI_FirstBit	描述
SPI_FisrtBit_MSB	数据传输从 MSB 位开始
SPI_FisrtBit_LSB	数据传输从 LSB 位开始

SPI_CRCPolynomial

SPI_CRCPolynomial 定义了用于 CRC 值计算的多项式。

例：

```
/* Initialize the SPI1 according to the SPI_InitStructure members */
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 函数SPI_StructInit

Table 419. 描述了函数SPI_StructInit

Table 419. 函数 SPI_StructInit

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

Table 420. 给出了SPI_InitStruct各个成员的缺省值

Table 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：
```
/* Initialize an SPI_InitTypeDef structure */
SPI_InitTypeDef SPI_InitStructure;
SPI_StructInit(&SPI_InitStructure);
```

17.2.4 函数SPI_Cmd

Table 421. 描述了函数SPI_ Cmd

Table 421. 函数 SPI_ Cmd

函数名	SPI_ Cmd
函数原形	void SPI_Cmd(SPI_TypeDef* SPIx, FunctionalState NewState)
功能描述	使能或者失能 SPI 外设
输入参数 1	SPIx：x 可以是 1 或者 2，来选择 SPI 外设
输入参数 2	NewState: 外设 SPIx 的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：
```
/* Enable SPI1 */
SPI_Cmd(SPI1, ENABLE);
```

17.2.5 函数SPI_ITConfig

Table 422. 描述了函数SPI_ITConfig

Table 422. 函数 SPI_ITConfig

函数名	SPI_ITConfig
函数原形	void SPI_ITConfig(SPI_TypeDef* SPIx, u16 SPI_IT, FunctionalState NewState)
功能描述	使能或者失能指定的 SPI 中断
输入参数 1	SPIx：x 可以是 1 或者 2，来选择 SPI 外设
输入参数 2	SPI_IT：待使能或者失能的 SPI 中断源 参阅 Section：SPI_IT 查阅更多该参数允许取值范围
输入参数 3	NewState：SPIx 中断的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

SPI_IT

输入参数 SPI_IT 使能或者失能 SPI 的中断。可以取下表的一个或者多个取值的组合作为该参数的值。

Table 423. SPI_IT 值

SPI_IT	描述
SPI_IT_TXE	发送缓存空中断屏蔽
SPI_IT_RXNE	接收缓存非空中断屏蔽
SPI_IT_ERR	错误中断屏蔽

例：

```
/* Enable SPI2 Tx buffer empty interrupt */
SPI_ITConfig(SPI2, SPI_IT_TXE, ENABLE);
```

17.2.6 函数SPI_DMACmd

Table 424. 描述了函数SPI_ DMACmd

Table 424. 函数 SPI_ DMACmd

函数名	SPI_ DMACmd
函数原形	void SPI_DMACmd(SPI_TypeDef* SPIx, u16 SPI_DMAReq, FunctionalState NewState)
功能描述	使能或者失能指定 SPI 的 DMA 请求
输入参数 1	SPIx：x 可以是 1 或者 2，来选择 SPI 外设
输入参数 2	SPI_DMAReq：待使能或者失能的 SPI DMA 传输请求 参阅 Section：SPI_DMAReq 查阅更多该参数允许取值范围
输入参数 3	NewState: SPIx DMA 传输的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

SPI_DMAReq

SPI_DMAReq 使能或者失能 SPI Tx 和/或 SPI Rx 的 DMA 传输请求。见 Table 425. 查阅该参数可取的值。

Table 425. SPI_DMAReq 值

SPI_DMAReq	描述
SPI_DMAReq_Tx	选择 Tx 缓存 DMA 传输请求
SPI_DMAReq_Rx	选择 Rx 缓存 DMA 传输请求

例：

```
/* Enable SPI2 Rx buffer DMA transfer request */
SPI_DMACmd(SPI2, SPI_DMAReq_Rx, ENABLE);
```

17.2.7 函数SPI_SendData

Table 426. 描述了函数SPI_ SendData

Table 426. 函数 SPI_ SendData

函数名	SPI_ SendData
函数原形	void SPI_SendData(SPI_TypeDef* SPIx, u16 Data)
功能描述	通过外设 SPIx 发送一个数据
输入参数 1	SPIx：x 可以是 1 或者 2，来选择 SPI 外设
输入参数 2	Data：待发送的数据
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Send 0xA5 through the SPI1 peripheral */
SPI_SendData(SPI1, 0xA5);
```

17.2.8 函数SPI_ReceiveData

Table 427. 描述了函数SPI_ ReceiveData

Table 427. 函数 SPI_ReceiveData

函数名	SPI_ ReceiveData
函数原形	u16 SPI_ReceiveData(SPI_TypeDef* SPIx)
功能描述	返回通过 SPIx 最近接收的数据
输入参数	SPIx：x 可以是 1 或者 2，来选择 SPI 外设
输出参数	无
返回值	接收到的字
先决条件	无
被调用函数	无

例：

```
/* Read the most recent data received by the SPI2 peripheral */
u16 ReceivedData;
ReceivedData = SPI_ReceiveData(SPI2);
```

17.2.9 函数SPI_NSSInternalSoftwareConfig

Table 428. 描述了函数SPI_NSSInternalSoftwareConfig

Table 428. 函数 SPI_NSSInternalSoftwareConfig

函数名	SPI_NSSInternalSoftwareConfig
函数原形	void SPI_NSSInternalSoftwareConfig(SPI_TypeDef* SPIx, u16 SPI_NSSInternalSoft)
功能描述	为选定的 SPI 软件配置内部 NSS 管脚
输入参数 1	SPIx：x 可以是 1 或者 2，来选择 SPI 外设
输入参数 2	SPI_NSSInternalSoft：SPI NSS 内部状态 参阅 Section：SPI_NSSInternalSoft 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

SPI_NSSInternalSoft

SPI_NSSInternalSoft 内部设置或者重置 NSS 管脚。见 Table 429. 查阅该参数可取的值。

Table 429. SPI_DMAReq 值

SPI_NSSInternalSoft	描述
SPI_NSSInternalSoft_Set	内部设置 NSS 管脚
SPI_NSSInternalSoft_Reset	内部重置 NSS 管脚

例：

```
/* Set internaly by software the SPI1 NSS pin */
SPI_NSSInternalSoftwareConfig(SPI1, SPI_NSSInternalSoft_Set);
/* Reset internaly by sofwtare the SPI2 NSS pin */
SPI_NSSInternalSoftwareConfig(SPI2, SPI_NSSInternalSoft_Reset);
```

17.2.10 函数SPI_SSOutputCmd

Table 430. 描述了函数SPI_SSOutputCmd

Table 430. 函数 SPI_SSOutputCmd

函数名	SPI_SSOutputCmd
函数原形	void SPI_SSOutputCmd(SPI_TypeDef* SPIx, FunctionalState NewState)
功能描述	使能或者失能指定的 SPI SS 输出
输入参数 1	SPIx：x 可以是 1 或者 2，来选择 SPI 外设
输入参数 2	NewState: SPI SS 输出的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable the SPI1 SS output: single master mode */
SPI_SSOutputCmd(SPI1, ENABLE);
```

17.2.11 函数SPI_DataSizeConfig

Table 431. 描述了函数SPI_DataSizeConfig

Table 431. 函数 SPI_DataSizeConfig

函数名	SPI_DataSizeConfig
函数原形	void SPI_DataSizeConfig(SPI_TypeDef* SPIx, u16 SPI_DatSize)
功能描述	设置选定的 SPI 数据大小
输入参数 1	SPIx：x 可以是 1 或者 2，来选择 SPI 外设
输入参数 2	SPI_DataSize：SPI 数据大小 参阅 Section：SPI_DataSize 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

SPI_DataSize

SPI_DataSize 设置 8 位或者 16 位数据帧结构。见 Table 432. 查阅该参数可取的值。

Table 432. SPI_DMAReq 值

SPI_DataSize	描述
SPI_DataSize_8b	设置数据为 8 位
SPI_DataSize_16b	设置数据为 16 位

例：

```
/* Set 8bit data frame format for SPI1 */
SPI_DataSizeConfig(SPI1, SPI_DataSize_8b);
/* Set 16bit data frame format for SPI2 */
SPI_DataSizeConfig(SPI2, SPI_DataSize_16b);
```

17.2.12 函数SPI_TransmitCRC

Table 433. 描述了函数SPI_ TransmitCRC

Table 433. 函数 SPI_ TransmitCRC

函数名	SPI_ TransmitCRC
函数原形	SPI_TransmitCRC(SPI_TypeDef* SPIx, FunctionalState NewState)
功能描述	使能或者失能指定 SPI 的 CRC 传输
输入参数 1	SPIx：x 可以是 1 或者 2，来选择 SPI 外设
输入参数 2	NewState：SPIxCRC 传输的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable the CRC transfer for SPI1 */
SPI_TransmitCRC(SPI1);
```

17.2.13 函数SPI_CalculateCRC

Table 434. 描述了函数SPI_ CalculateCRC

Table 434. 函数 SPI_ CalculateCRC

函数名	SPI_ CalculateCRC
函数原形	void SPI_CalculateCRC(SPI_TypeDef* SPIx, FunctionalState NewState)
功能描述	使能或者失能指定 SPI 的传输字 CRC 值计算
输入参数 1	SPIx：x 可以是 1 或者 2，来选择 SPI 外设
输入参数 2	NewState：SPIx 传输字 CRC 值计算的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable the CRC calculation for the transfered bytes from SPI2 */
SPI_CalculateCRC(SPI2, ENABLE);
```

17.2.14 函数SPI_GetCRC

Table 435. 描述了函数SPI_ GetCRC

Table 435. 函数 SPI_ GetCRC

函数名	SPI_ GetCRC
函数原形	u16 SPI_GetCRC(SPI_TypeDef* SPIx)
功能描述	返回指定 SPI 的 CRC 值
输入参数 1	SPIx：x 可以是 1 或者 2，来选择 SPI 外设
输入参数 2	SPI_CRC：待读取的 CRC 寄存器 参阅 Section：SPI_CRC 查阅更多该参数允许取值范围
输出参数	无
返回值	CRC 值
先决条件	无
被调用函数	无

SPI_CRC

SPI_CRC 选择 SPI Rx 或者 SPI Tx 的 CRC 寄存器。见 Table 436. 查阅该参数可取的值。

Table 436. SPI_CRC 值

SPI_CRC	描述
SPI_CRC_Tx	选择 Tx CRC 寄存器
SPI_CRC_Rx	选择 Rx CRC 寄存器

例：

```
/* Returns the SPI1 transmit CRC register */
u16 CRCValue;
CRCValue = SPI_GetCRC(SPI1, SPI_CRC_Tx);
```

17.2.15 函数SPI_GetCRCPolynomial

Table 437. 描述了函数SPI_GetCRCPolynomial

Table 437. 函数 SPI_GetCRCPolynomial

函数名	SPI_GetCRCPolynomial
函数原形	u16 SPI_GetCRCPolynomial(SPI_TypeDef* SPIx)
功能描述	返回指定 SPI 的 CRC 多项式寄存器值
输入参数	SPIx：x 可以是 1 或者 2，来选择 SPI 外设
输出参数	无
返回值	CRC 多项式寄存器值
先决条件	无
被调用函数	无

例：

```
/* Returns the SPI2 CRC polynomial register */
u16 CRCPolyValue;
CRCPolyValue = SPI_GetCRCPolynomial(SPI2);
```

17.2.16 函数SPI_BiDirectionalLineConfig

Table 438. 描述了函数SPI_BiDirectionalLineConfig

Table 438. 函数 SPI_BiDirectionalLineConfig

函数名	SPI_BiDirectionalLineConfig
函数原形	SPI_BiDirectionalLineConfig(SPI_TypeDef* SPIx, u16 SPI_Direction)
功能描述	选择指定 SPI 在双向模式下的数据传输方向
输入参数 1	SPIx：x 可以是 1 或者 2，来选择 SPI 外设
输入参数 2	SPI_Direction：待读取的 CRC 寄存器 参阅 Section：SPI_CRC 查阅更多该参数允许取值范围
输出参数	无
返回值	CRC 值
先决条件	无
被调用函数	无

SPI_Direction

SPI_Direction选择SPI在双向模式下的数据传输方向。见Table 439. 查阅该参数可取的值。

Table 439. SPI_CRC 值

SPI_Direction	描述
SPI_Direction_Tx	选择 Tx 发送方向
SPI_Direction_Rx	选择 Rx 接受方向

例：

```
/* Set the SPI2 in bidirectional transmit only mode */
SPI_BiDirectionalLineConfig(SPI_Direction_Tx);
```

17.2.17 函数SPI_GetFlagStatus

Table 440. 描述了函数SPI_ GetFlagStatus

Table 440. 函数 SPI_ GetFlagStatus

函数名	SPI_ GetFlagStatus
函数原形	FlagStatus SPI_GetFlagStatus(SPI_TypeDef* SPIx, u16 SPI_FLAG)
功能描述	检查指定的 SPI 标志位设置与否
输入参数 1	SPIx：x 可以是 1 或者 2，来选择 SPI 外设
输入参数 2	SPI_FLAG：待检查的 SPI 标志位 参阅 Section：SPI_FLAG 查阅更多该参数允许取值范围
输出参数	无
返回值	SPI_FLAG 的新状态（SET 或者 RESET）
先决条件	无
被调用函数	无

SPI_FLAG

Table 441. 给出了所有可以被函数SPI_ GetFlagStatus检查的标志位列表

Table 441. SPI_FLAG 值

SPI_FLAG	描述
SPI_FLAG_BSY	忙标志位
SPI_FLAG_OVR	超出标志位
SPI_FLAG_MODF	模式错位标志位
SPI_FLAG_CRCERR	CRC 错误标志位
SPI_FLAG_TXE	发送缓存空标志位
SPI_FLAG_RXNE	接受缓存非空标志位

例：

17.2.18 函数SPI_ClearFlag

Table 442. 描述了函数SPI_ ClearFlag

Table 442. 函数 SPI_ ClearFlag

函数名	SPI_ ClearFlag
函数原形	void SPI_ClearFlag(SPI_TypeDef* SPIx, u16 SPI_FLAG)
功能描述	清除 SPIx 的待处理标志位
输入参数 1	SPIx：x 可以是 1 或者 2，来选择 SPI 外设
输入参数 2	SPI_FLAG：待清除的 SPI 标志位 参阅 Section：SPI_FLAG 查阅更多该参数允许取值范围 注意：标志位 BSY, TXE 和 RXNE 由硬件重置
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear the SPI2 Overrun pending bit */
SPI_ClearFlag(SPI2, SPI_FLAG_OVR);
```

17.2.19 函数SPI_GetITStatus

Table 443. 描述了函数SPI_ GetITStatus

Table 443. 函数 SPI_ GetITStatus

函数名	SPI_ GetITStatus
函数原形	ITStatus SPI_GetITStatus(SPI_TypeDef* SPIx, u8 SPI_IT)
功能描述	检查指定的 SPI 中断发生与否
输入参数 1	SPIx：x 可以是 1 或者 2，来选择 SPI 外设
输入参数 2	SPI_IT：待检查的 SPI 中断源 参阅 Section：SPI_IT 查阅更多该参数允许取值范围
输出参数	无
返回值	SPI_IT 的新状态
先决条件	无
被调用函数	无

SPI_IT

Table 444. 给出了所有可以被函数SPI_ GetITStatus检查的中断标志位列表

Table 444. SPI_IT 值

SPI_IT	描述
SPI_IT_OVR	超出中断标志位
SPI_IT_MODF	模式错误标志位
SPI_IT_CRCERR	CRC 错误标志位
SPI_IT_TXE	发送缓存空中断标志位
SPI_IT_RXNE	接受缓存非空中断标志位

例：

```
/* Test if the SPI1 Overrun interrupt has occurred or not */
ITStatus Status;
Status = SPI_GetITStatus(SPI1, SPI_IT_OVR);
```

17.2.20 函数SPI_ClearITPendingBit

Table 445. 描述了函数SPI_ ClearITPendingBit

Table 445. 函数 SPI_ ClearITPendingBit

函数名	SPI_ ClearITPendingBit
函数原形	void SPI_ClearITPendingBit(SPI_TypeDef* SPIx, u8 SPI_IT)
功能描述	清除 SPIx 的中断待处理位
输入参数 1	SPIx：x 可以是 1 或者 2，来选择 SPI 外设
输入参数 2	SPI_IT：待检查的 SPI 中断源 参阅 Section：SPI_IT 查阅更多该参数允许取值范围 注意：中断标志位 BSY, TXE 和 RXNE 由硬件重置
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear the SPI2 CRC error interrupt pending bit */
SPI_ClearITPendingBit(SPI2, SPI_IT_CRCERR);
```

18 Cortex系统定时器（SysTick）

SysTick 提供 1 个 24 位、降序、零约束、写清除的计数器，具有灵活的控制机制。

Section 18.1 SysTick 寄存器结构描述了固件函数库所使用的数据结构，Section 18.2 固件库函数介绍了函数库里的所有函数。

18.1 SysTick寄存器结构

SYSTICK 寄存器结构，***SysTick_TypeDeff***，在文件***"stm32f10x_map.h"***中定义如下：

```
typedef struct
{
vu32 CTRL;
vu32 LOAD;
vu32 VAL;
vuc32 CALIB;
} SysTick_TypeDef;
```

Table 446.例举了SysTick所有寄存器

Table 446. SysTick 寄存器

寄存器	描述
CTRL	SysTick 控制和状态寄存器
LOAD	SysTick 重装载值寄存器
VAL	SysTick 当前值寄存器
CALIB	SysTick 校准值寄存器

SysTick 外设声明于文件***"stm32f10x_map.h"***：

```
#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
```

使用Debug模式时，初始化指针***SysTick***于文件***"stm32f10x_lib.c"***：

```
#ifdef _SysTick
SysTick = (SysTick_TypeDef *) SysTick_BASE;
#endif /*_SysTick */
```

为了访问SysTick寄存器，,***_SysTick***必须在文件***"stm32f10x_conf.h"***中定义如下：

```
#define _SysTick
```

18.2 SysTick库函数

Table 447. 例举了SysTick的库函数

Table 447. SysTick 库函数

函数名	描述
SysTick_CLKSourceConfig	设置 SysTick 时钟源
SysTick_SetReload	设置 SysTick 重装载值
SysTick_CounterCmd	使能或者失能 SysTick 计数器
SysTick_ITConfig	使能或者失能 SysTick 中断
SysTick_GetCounter	获取 SysTick 计数器的值
SysTick_GetFlagStatus	检查指定的 SysTick 标志位设置与否

18.2.1 函数SysTick_CLKSourceConfig

Table 448. 描述了函数SysTick_CLKSourceConfig

Table 448. 函数 SysTick_CLKSourceConfig

函数名	SysTick_CLKSourceConfig
函数原形	void SysTick_CLKSourceConfig(u32 SysTick_CLKSource)
功能描述	设置 SysTick 时钟源
输入参数	SysTick_CLKSource：SysTick 时钟源 参阅 Section：SysTick_CLKSource 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

SysTick_CLKSource

SysTick_CLKSource 选择 SysTick 时钟源，见表 449. 查阅更多该参数可取的值。

Table 449. SysTick_CLKSource 值

SysTick_CLKSource	描述
SysTick_CLKSource_HCLK_Div8	SysTick 时钟源为 AHB 时钟除以 8
SysTick_CLKSource_HCLK	SysTick 时钟源为 AHB 时钟

例：

```
/* AHB clock selected as SysTick clock source */
SysTick_CLKSourceConfig(SysTick_CLKSource_HCLK);
```

18.2.2 函数SysTick_SetReload

Table 450. 描述了函数SysTick_SetReload

Table 450. 函数 SysTick_SetReload

函数名	SysTick_SetReload
函数原形	void SysTick_SetReload(u32 Reload)
功能描述	设置 SysTick 重装载值
输入参数	Reload：重装载值 该参数取值必须在 1 和 0x00FFFFFF 之间
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Set SysTick reload value to 0xFFFF */
SysTick_SetReload(0xFFFF);
```

18.2.3 函数SysTick_CounterCmd

Table 451. 描述了函数SysTick_CounterCmd

Table 451. 函数 SysTick_CounterCmd

函数名	SysTick_CounterCmd
函数原形	void SysTick_CounterCmd(u32 SysTick_Counter)
功能描述	使能或者失能 SysTick 计数器
输入参数	SysTick_Counter：SysTick 计数器新状态 参阅 Section：SysTick_Counter 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

SysTick_Counter

SysTick_Counter 选择 SysTick 计数器的状态，见表 452. 查阅更多该参数可取的值。

Table 452. SysTick_Counter 值

SysTick_Counter	描述
SysTick_Counter_Disable	失能计数器
SysTick_Counter_Enable	使能计数器
SysTick_Counter_Clear	清除计数器值为 0

例：

```
/* Enable SysTick counter */
SysTick_CounterCmd(SysTick_Counter_Enable);
```

18.2.4 函数SysTick_ITConfig

Table 453. 描述了函数SysTick_ITConfig

Table 453. 函数 SysTick_ITConfig

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

例：

```
/* Enable SysTick interrupt */
SysTick_ITConfig(ENABLE);
```

18.2.5 函数SysTick_GetCounter

Table 454. 描述了函数SysTick_GetCounter

Table 454. 函数 SysTick_GetCounter

函数名	SysTick_GetCounter
函数原形	u32 SysTick_GetCounter(void)
功能描述	获取 SysTick 计数器的值
输入参数	无
输出参数	无
返回值	SysTick 计数器的值
先决条件	无
被调用函数	无

例：

```
/* Get SysTick current counter value */
u32 SysTickCurrentCounterValue;
SysTickCurrentCounterValue = SysTick_GetCounter();
```

18.2.6 函数SysTick_GetFlagStatus

Table 455. 描述了函数SysTick _ GetFlagStatus

Table 455. 函数 SysTick _ GetFlagStatus

函数名	SysTick _ GetFlagStatus
函数原形	FlagStatus SysTick_GetFlagStatus(u8 SysTick_FLAG)
功能描述	检查指定的 SysTick 标志位设置与否
输入参数 2	SysTick_FLAG：待检查的 SysTic 标志位 参阅 Section：SysTick_FLAG 查阅更多该参数允许取值范围
输出参数	无
返回值	SysTick_FLAG 的新状态
先决条件	无
被调用函数	无

SysTick_FLAG

Table 456. 给出了所有可以被函数SysTick_ GetITStatus检查的中断标志位列表

Table 456. SysTick_FLAG 值

SysTick_FLAG	描述
SysTick_FLAG_COUNT	自从上一次被读取，计数器计数至 0
SysTick_FLAG_SKEW	由于时钟频率，校准值不精确等于 10ms
SysTick_FLAG_NOREF	参考时钟未提供

例：

```
/* Test if the Count flag is set or not */
FlagStatus Status;
Status = SysTick_GetFlagStatus(SysTick_FLAG_COUNT);
if(Status == RESET)
{
...
}
else
{
...
}
```

19 通用定时器（TIM）

通用定时器是一个通过可编程预分频器驱动的 16 位自动装载计数器构成。

它适用于多种场合，包括测量输入信号的脉冲长度(输入采集)或者产生输出波形(输出比较和 PWM)。

使用定时器预分频器和 RCC 时钟控制器预分频器，脉冲长度和波形周期可以在几个微秒到几个毫秒间调整。

Section 19.1 TIM 寄存器结构描述了固件函数库所使用的数据结构，Section 19.2 固件库函数介绍了函数库里的所有函数。

19.1 TIM寄存器结构

TIM 寄存器结构，***TIM_TypeDeff***，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;
```

Table 457.例举了TIM所有寄存器

Table 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/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/比较使能寄存器
CNT	计数器寄存器
PSC	预分频寄存器
APR	自动重装载寄存器
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	连续模式的 DMA 地址寄存器

三个 TIM 外设声明于文件***"stm32f10x_map.h"***:

```
...
#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
```

使用Debug模式时，初始化指针***TIM2, TIM3***和***TIM4***于文件***"stm32f10x_lib.c"***:

```
...
#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 */
...
```

为了访问 TIM 寄存器，,***_TIM，_TIM2, _TIM3*** 和***_TIM4*** 必须在文件***"stm32f10x_conf.h"***中定义如下：

```
...
#define _TIM
#define _TIM2
#define _TIM3
#define _TIM4
...
```

19.2 TIM库函数

Table 458. 例举了TIM的库函数

Table 458. TIM 库函数

函数名	描述
TIM_DeInit	将外设 TIMx 寄存器重设为缺省值
TIM_TimeBaseInit	根据 TIM_TimeBaseInitStruct 中指定的参数初始化 TIMx 的时间基数单位
TIM_OCInit	根据 TIM_OCInitStruct 中指定的参数初始化外设 TIMx
TIM_ICInit	根据 TIM_ICInitStruct 中指定的参数初始化外设 TIMx
TIM_TimeBaseStructInit	把 TIM_TimeBaseInitStruct 中的每一个参数按缺省值填入
TIM_OCStructInit	把 TIM_OCInitStruct 中的每一个参数按缺省值填入
TIM_ICStructInit	把 TIM_ICInitStruct 中的每一个参数按缺省值填入
TIM_Cmd	使能或者失能 TIMx 外设
TIM _ITConfig	使能或者失能指定的 TIM 中断
TIM_DMAConfig	设置 TIMx 的 DMA 接口
TIM_DMACmd	使能或者失能指定的 TIMx 的 DMA 请求
TIM_InternalClockConfig	设置 TIMx 内部时钟
TIM_ITRxExternalClockConfig	设置 TIMx 内部触发为外部时钟模式
TIM_TIxExternalClockConfig	设置 TIMx 触发为外部时钟
TIM_ETRClockMode1Config	配置 TIMx 外部时钟模式 1
TIM_ETRClockMode2Config	配置 TIMx 外部时钟模式 2
TIM_ETRConfig	配置 TIMx 外部触发
TIM_SelectInputTrigger	选择 TIMx 输入触发源
TIM_PrescalerConfig	设置 TIMx 预分频
TIM_CounterModeConfig	设置 TIMx 计数器模式
TIM_ForcedOC1Config	置 TIMx 输出 1 为活动或者非活动电平
TIM_ForcedOC2Config	置 TIMx 输出 2 为活动或者非活动电平
TIM_ForcedOC3Config	置 TIMx 输出 3 为活动或者非活动电平
TIM_ForcedOC4Config	置 TIMx 输出 4 为活动或者非活动电平
TIM_ARRPreloadConfig	使能或者失能 TIMx 在 ARR 上的预装载寄存器
TIM_SelectCCDMA	选择 TIMx 外设的捕获比较 DMA 源
TIM_OC1PreloadConfig	使能或者失能 TIMx 在 CCR1 上的预装载寄存器
TIM_OC2PreloadConfig	使能或者失能 TIMx 在 CCR2 上的预装载寄存器
TIM_OC3PreloadConfig	使能或者失能 TIMx 在 CCR3 上的预装载寄存器
TIM_OC4PreloadConfig	使能或者失能 TIMx 在 CCR4 上的预装载寄存器
TIM_OC1FastConfig	设置 TIMx 捕获比较 1 快速特征

TIM_OC2FastConfig	设置 TIMx 捕获比较 2 快速特征
TIM_OC3FastConfig	设置 TIMx 捕获比较 3 快速特征
TIM_OC4FastConfig	设置 TIMx 捕获比较 4 快速特征
TIM_ClearOC1Ref	在一个外部事件时清除或者保持 OCREF1 信号
TIM_ClearOC2Ref	在一个外部事件时清除或者保持 OCREF2 信号
TIM_ClearOC3Ref	在一个外部事件时清除或者保持 OCREF3 信号
TIM_ClearOC4Ref	在一个外部事件时清除或者保持 OCREF4 信号
TIM_UpdateDisableConfig	使能或者失能 TIMx 更新事件
TIM_EncoderInterfaceConfig	设置 TIMx 编码界面
TIM_GenerateEvent	设置 TIMx 事件由软件产生
TIM_OC1PolarityConfig	设置 TIMx 通道 1 极性
TIM_OC2PolarityConfig	设置 TIMx 通道 2 极性
TIM_OC3PolarityConfig	设置 TIMx 通道 3 极性
TIM_OC4PolarityConfig	设置 TIMx 通道 4 极性
TIM_UpdateRequestConfig	设置 TIMx 更新请求源
TIM_SelectHallSensor	使能或者失能 TIMx 霍尔传感器接口
TIM_SelectOnePulseMode	设置 TIMx 单脉冲模式
TIM_SelectOutputTrigger	选择 TIMx 触发输出模式
TIM_SelectSlaveMode	选择 TIMx 从模式
TIM_SelectMasterSlaveMode	设置或者重置 TIMx 主/从模式
TIM_SetCounter	设置 TIMx 计数器寄存器值
TIM_SetAutoreload	设置 TIMx 自动重装载寄存器值
TIM_SetCompare1	设置 TIMx 捕获比较 1 寄存器值
TIM_SetCompare2	设置 TIMx 捕获比较 2 寄存器值
TIM_SetCompare3	设置 TIMx 捕获比较 3 寄存器值
TIM_SetCompare4	设置 TIMx 捕获比较 4 寄存器值
TIM_SetIC1Prescaler	设置 TIMx 输入捕获 1 预分频
TIM_SetIC2Prescaler	设置 TIMx 输入捕获 2 预分频
TIM_SetIC3Prescaler	设置 TIMx 输入捕获 3 预分频
TIM_SetIC4Prescaler	设置 TIMx 输入捕获 4 预分频
TIM_SetClockDivision	设置 TIMx 的时钟分割值
TIM_GetCapture1	获得 TIMx 输入捕获 1 的值
TIM_GetCapture2	获得 TIMx 输入捕获 2 的值
TIM_GetCapture3	获得 TIMx 输入捕获 3 的值
TIM_GetCapture4	获得 TIMx 输入捕获 4 的值
TIM_GetCounter	获得 TIMx 计数器的值
TIM_GetPrescaler	获得 TIMx 预分频值
TIM_GetFlagStatus	检查指定的 TIM 标志位设置与否
TIM_ClearFlag	清除 TIMx 的待处理标志位
TIM_GetITStatus	检查指定的 TIM 中断发生与否
TIM_ClearITPendingBit	清除 TIMx 的中断待处理位

19.2.1 函数TIM_DeInit

Table 459. 描述了函数 TIM_DeInit

Table 459. 函数 TIM_DeInit

函数名	TIM_DeInit
函数原形	void TIM_DeInit(TIM_TypeDef* TIMx)
功能描述	将外设 TIMx 寄存器重设为缺省值
输入参数	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输出参数	无
返回值	无
先决条件	无
被调用函数	RCC_APB1PeriphClockCmd().

例：

```
/* Resets the TIM2 */
TIM_DeInit(TIM2);
```

19.2.2 函数TIM_TimeBaseInit

Table 460. 描述了函数 TIM_TimeBaseInit

Table 460. 函数 TIM_TimeBaseInit

函数名	TIM_TimeBaseInit
函数原形	void TIM_TimeBaseInit(TIM_TypeDef* TIMx, TIM_TimeBaseInitTypeDef* TIM_TimeBaseInitStruct)
功能描述	根据 TIM_TimeBaseInitStruct 中指定的参数初始化 TIMx 的时间基数单位
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIMTimeBase_InitStruct：指向结构 TIM_TimeBaseInitTypeDef 的指针，包含了 TIMx 时间基数单位的配置信息 参阅 Section：TIM_TimeBaseInitTypeDef 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM_TimeBaseInitTypeDef structure

TIM_TimeBaseInitTypeDef 定义于文件“stm32f10x_tim.h”：

```
typedef struct
{
u16 TIM_Period;
u16 TIM_Prescaler;
u8 TIM_ClockDivision;
u16 TIM_CounterMode;
} TIM_TimeBaseInitTypeDef;
```

TIM_Period

TIM_Period 设置了在下一个更新事件装入活动的自动重装载寄存器周期的值。它的取值必须在 0x0000 和 0xFFFF 之间。

TIM_Prescaler

TIM_Prescaler 设置了用来作为 TIMx 时钟频率除数的预分频值。它的取值必须在 0x0000 和 0xFFFF 之间。

TIM_ClockDivision

TIM_ClockDivision 设置了时钟分割。该参数取值见下表。

Table 461. TIM_ClockDivision 值

TIM_ClockDivision	描述
TIM_CKD_DIV1	TDTS = Tck_tim
TIM_CKD_DIV2	TDTS = 2Tck_tim
TIM_CKD_DIV4	TDTS = 4Tck_tim

TIM_CounterMode

TIM_CounterMode 选择了计数器模式。该参数取值见下表。

Table 462. TIM_CounterMode 值

TIM_CounterMode	描述
TIM_CounterMode_Up	TIM 向上计数模式
TIM_CounterMode_Down	TIM 向下计数模式
TIM_CounterMode_CenterAligned1	TIM 中央对齐模式 1 计数模式
TIM_CounterMode_CenterAligned2	TIM 中央对齐模式 2 计数模式
TIM_CounterMode_CenterAligned3	TIM 中央对齐模式 3 计数模式

例：

```
TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
TIM_TimeBaseStructure.TIM_Period = 0xFFFF;
TIM_TimeBaseStructure.TIM_Prescaler = 0xF;
TIM_TimeBaseStructure.TIM_ClockDivision = 0x0;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM2, & TIM_TimeBaseStructure);
```

19.2.3 函数TIM_OCInit

Table 463. 描述了函数 TIM_OCInit

Table 463. 函数 TIM_OCInit

函数名	TIM_OCInit
函数原形	void TIM_OCInit(TIM_TypeDef* TIMx, TIM_OCInitTypeDef* TIM_OCInitStruct)
功能描述	根据 TIM_OCInitStruct 中指定的参数初始化外设 TIMx
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_OCInitStruct：指向结构 TIM_OCInitTypeDef 的指针，包含了 TIMx 时间基数单位的配置信息 参阅 Section：TIM_OCInitTypeDef 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM_OCInitTypeDef structure

TIM_OCInitTypeDef 定义于文件***"stm32f10x_tim.h"***：

```
typedef struct
{
u16 TIM_OCMode;
u16 TIM_Channel;
u16 TIM_Pulse;
u16 TIM_OCPolarity;
} TIM_OCInitTypeDef;
```

TIM_OCMode

TIM_OCMode 选择定时器模式。该参数取值见下表。

Table 464. TIM_OCMode 定义

TIM_OCMode	描述
TIM_OCMode_Timing	TIM 输出比较时间模式
TIM_OCMode_Active	TIM 输出比较主动模式
TIM_OCMode_Inactive	TIM 输出比较非主动模式
TIM_OCMode_Toggle	TIM 输出比较触发模式
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

TIM_Channel

TIM_Channel 选择通道。该参数取值见下表。

Table 465. TIM_Channel 值

TIM_Channel	描述
TIM_Channel_1	使用 TIM 通道 1
TIM_Channel_2	使用 TIM 通道 2
TIM_Channel_3	使用 TIM 通道 3
TIM_Channel_4	使用 TIM 通道 4

TIM_Pulse

TIM_Pulse 设置了待装入捕获比较寄存器的脉冲值。它的取值必须在 0x0000 和 0xFFFF 之间。

TIM_OCPolarity

TIM_OCPolarity输出极性。该参数取值见下表。

Table 466. TIM_OCPolarity 值

TIM_OCPolarity	描述
TIM_OCPolarity_High	TIM 输出比较极性高
TIM_OCPolarity_Low	TIM 输出比较极性低

例：

```
/* Configures the TIM2 Channel1 in PWM Mode */
TIM_OCInitTypeDef TIM_OCInitStructure;
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
TIM_OCInitStructure.TIM_Channel = TIM_Channel_1;
TIM_OCInitStructure.TIM_Pulse = 0x3FFF;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
TIM_OCInit(TIM2, & TIM_OCInitStructure);
```

19.2.4 函数TIM_ICInit

Table 467. 描述了函数 TIM_ICInit

Table 467. 函数 TIM_ICInit

函数名	TIM_OCInit
函数原形	void TIM_ICInit(TIM_TypeDef* TIMx, TIM_ICInitTypeDef* TIM_ICInitStruct)
功能描述	根据 TIM_ICInitStruct 中指定的参数初始化外设 TIMx
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_ICInitStruct：指向结构 TIM_ICInitTypeDef 的指针，包含了 TIMx 的配置信息 参阅 Section：TIM_ICInitTypeDef 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM_ICInitTypeDef structure

TIM_ICInitTypeDef 定义于文件***"stm32f10x_tim.h"***：

```
typedef struct
{
u16 TIM_ICMode;
u16 TIM_Channel;
u16 TIM_ICPolarity;
u16 TIM_ICSelection;
u16 TIM_ICPrescaler;
u16 TIM_ICFilter;
} TIM_ICInitTypeDef;
```

TIM_ICMode

TIM_ICMode选择了TIM输入捕获模式。该参数取值见下表。

Table 468. TIM_ICMode 定义

TIM_ICMode	描述
TIM_ICMode_ICAP	TIM 使用输入捕获模式
TIM_ICMode_PWMI	TIM 使用输入 PWM 模式

TIM_Channel

TIM_Channel 选择通道。该参数取值见下表。

Table 469. TIM_Channel 值

TIM_Channel	描述
TIM_Channel_1	使用 TIM 通道 1
TIM_Channel_2	使用 TIM 通道 2
TIM_Channel_3	使用 TIM 通道 3
TIM_Channel_4	使用 TIM 通道 4

TIM_ICPolarity

TIM_ICPolarity输入活动沿。该参数取值见下表。

Table 470. TIM_Channel 值

TIM_OCPolarity	描述
TIM_ICPolarity_Rising	TIM 输入捕获上升沿
TIM_ICPolarity_Falling	TIM 输入捕获下降沿

TIM_ICSelection

TIM_ICSelection选择输入。该参数取值见下表。

Table 471. TIM_ICSelection 值

TIM_ICSelection	描述
TIM_ICSelection_DirectTI	TIM 输入 2，3 或 4 选择对应地与 IC1 或 IC2 或 IC3 或 IC4 相连
TIM_ICSelection_IndirectTI	TIM 输入 2，3 或 4 选择对应地与 IC2 或 IC1 或 IC4 或 IC3 相连
TIM_ICSelection_TRC	TIM 输入 2，3 或 4 选择与 TRC 相连

TIM_ICPrescaler

TIM_ICPrescaler设置输入捕获预分频器。该参数取值见下表。

Table 472. TIM_ICPrescaler 值

TIM_ICPrescaler	描述
TIM_ICPSC_DIV1	TIM 捕获在捕获输入上每探测到一个边沿执行一次
TIM_ICPSC_DIV2	TIM 捕获每 2 个事件执行一次
TIM_ICPSC_DIV3	TIM 捕获每 3 个事件执行一次
TIM_ICPSC_DIV4	TIM 捕获每 4 个事件执行一次

TIM_ICFilter

TIM_ICFilter选择输入比较滤波器。该参数取值在0x0和0xF之间。

例：

```
/* The following example illustrates how to configure the TIM2 in
PWM Input mode : The external signal is connected to TIM2 CH1 pin,
the Rising edge is used as active edge, the TIM2 CCR1 is used to
compute the frequency value the TIM2 CCR2 is used to compute the
duty cycle value */
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 函数TIM_TimeBaseStructInit

Table 473. 描述了函数TIM_TimeBaseStructInit

Table 473. 函数 TIM_TimeBaseStructInit

函数名	TIM_TimeBaseStructInit
函数原形	void TIM_TimeBaseStructInit(TIM_TimeBaseInitTypeDef* TIM_TimeBaseInitStruct)
功能描述	把 TIM_TimeBaseInitStruct 中的每一个参数按缺省值填入
输入参数	TIM_TimeBaseInitStruct：指向结构 TIM_TimeBaseInitTypeDef 的指针，待初始化
输出参数	无
返回值	无
先决条件	无
被调用函数	无

Table 474. 给出了TIM_TimeBaseInitStruct各个成员的缺省值

Table 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：

```
/* The following example illustrates how to initialize a
TIM_BaseInitTypeDef structure */
TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure;
TIM_TimeBaseStructInit(& TIM_TimeBaseInitStructure);
```

19.2.6 函数TIM_OCStructInit

Table 475. 描述了函数TIM_OCStructInit

Table 475. 函数 TIM_TimeBaseStructInit

函数名	TIM_TimeBaseStructInit
函数原形	void TIM_OCStructInit(TIM_OCInitTypeDef* TIM_OCInitStruct)
功能描述	把 TIM_OCInitStruct 中的每一个参数按缺省值填入
输入参数	TIM_OCInitStruct：指向结构 TIM_OCInitTypeDef 的指针，待初始化
输出参数	无
返回值	无
先决条件	无
被调用函数	无

Table 476. 给出了TIM_OCInitStruct各个成员的缺省值

Table 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：

```
/* The following example illustrates how to initialize a
TIM_OCInitTypeDef structure */
TIM_OCInitTypeDef TIM_OCInitStructure;
TIM_OCStructInit(& TIM_OCInitStructure);
```

19.2.7 函数TIM_ICStructInit

Table 477. 描述了函数TIM_ICStructInit

Table 477. 函数 TIM_ICStructInit

函数名	TIM_ICStructInit
函数原形	void TIM_ICStructInit(TIM_ICInitTypeDef* TIM_ICInitStruct)
功能描述	把 TIM_ICInitStruct 中的每一个参数按缺省值填入
输入参数	TIM_ICInitStruct：指向结构 TIM_ICInitTypeDef 的指针，待初始化
输出参数	无
返回值	无
先决条件	无
被调用函数	无

Table 478. 给出了TIM_ICInitStruct各个成员的缺省值

Table 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：

```
/* The following example illustrates how to initialize a
TIM_ICInitTypeDef structure */
TIM_ICInitTypeDef TIM_ICInitStructure;
TIM_ICStructInit(& TIM_ICInitStructure);
```

19.2.8 函数TIM_Cmd

Table 479. 描述了函数TIM_Cmd

Table 479. 函数 TIM_Cmd

函数名	TIM_Cmd
函数原形	void TIM_Cmd(TIM_TypeDef* TIMx, FunctionalState NewState)
功能描述	使能或者失能 TIMx 外设
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	NewState: 外设 TIMx 的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enables the TIM2 counter */
TIM_Cmd(TIM2, ENABLE);
```

19.2.9 函数TIM _ITConfig

Table 480. 描述了函数TIM_ITConfig

Table 480. 函数 TIM_ITConfig

函数名	TIM_ITConfig
函数原形	void TIM_ITConfig(TIM_TypeDef* TIMx, u16 TIM_IT, FunctionalState NewState)
功能描述	使能或者失能指定的 TIM 中断
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_IT：待使能或者失能的 TIM 中断源 参阅 Section：TIM_IT 查阅更多该参数允许取值范围
输入参数 3	NewState：TIMx 中断的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM_IT

输入参数 TIM_IT 使能或者失能 TIM 的中断。可以取下表的一个或者多个取值的组合作为该参数的值。

Table 481. TIM_IT 值

TIM_IT	描述
TIM_IT_Update	TIM 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 触发中断源

例：

```
/* Enables the TIM2 Capture Compare channel 1 Interrupt source */
TIM_ITConfig(TIM2, TIM_IT_CC1, ENABLE );
```

19.2.10 函数TIM_DMAConfig

Table 482. 描述了函数TIM_DMAConfig

Table 482. 函数 TIM_DMAConfig

函数名	TIM_DMAConfig
函数原形	void TIM_DMAConfig(TIM_TypeDef* TIMx,u8 TIM_DMABase, u16 TIM_DMABurstLength)
功能描述	设置 TIMx 的 DMA 接口
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_DMABase：DMA 传输起始地址 参阅 Section：TIM_ DMABase 查阅更多该参数允许取值范围
输入参数 3	TIM_DMABurstLength：DMA 连续传送长度 参阅 Section：TIM_ DMABurstLength 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM_DMABase

TIM_DMABase 设置 DMA 传输起始地址。可以取下表的值。

Table 483. TIM_DMABase 值

TIM_DMABase	描述
TIM_DMABase_CR1	TIM CR1 寄存器作为 DMA 传输起始
TIM_DMABase_CR2	TIM CR2 寄存器作为 DMA 传输起始
TIM_DMABase_SMCR	TIM SMCR 寄存器作为 DMA 传输起始
TIM_DMABase_DIER	TIM DIER 寄存器作为 DMA 传输起始
TIM_DMABase_SR	TIM SR 寄存器作为 DMA 传输起始
TIM_DMABase_EGR	TIM EGR 寄存器作为 DMA 传输起始
TIM_DMABase_CCMR1	TIM CCMR1 寄存器作为 DMA 传输起始
TIM_DMABase_CCMR2	TIM CCMR2 寄存器作为 DMA 传输起始
TIM_DMABase_CCER	TIM CCER 寄存器作为 DMA 传输起始
TIM_DMABase_CNT	TIM CNT 寄存器作为 DMA 传输起始
TIM_DMABase_PSC	TIM PSC 寄存器作为 DMA 传输起始
TIM_DMABase_ARR	TIM APR 寄存器作为 DMA 传输起始
TIM_DMABase_CCR1	TIM CCR1 寄存器作为 DMA 传输起始
TIM_DMABase_CCR2	TIM CCR2 寄存器作为 DMA 传输起始
TIM_DMABase_CCR3	TIM CCR3 寄存器作为 DMA 传输起始
TIM_DMABase_CCR4	TIM CCR4 寄存器作为 DMA 传输起始
TIM_DMABase_DCR	TIM DCR 寄存器作为 DMA 传输起始

TIM_DMABurstLength

TIM_DMABurstLength 设置 DMA 连续传送长度。可以取下表的值。

Table 484. TIM_DMABurstLength 值

TIM_DMABurstLength	描述
TIM_DMABurstLength_1Byte	TIM DMA 连续传送长度 1 字
TIM_DMABurstLength_2Bytes	TIM DMA 连续传送长度 2 字
TIM_DMABurstLength_3Bytes	TIM DMA 连续传送长度 3 字
TIM_DMABurstLength_4Bytes	TIM DMA 连续传送长度 4 字
TIM_DMABurstLength_5Bytes	TIM DMA 连续传送长度 5 字
TIM_DMABurstLength_6Bytes	TIM DMA 连续传送长度 6 字
TIM_DMABurstLength_7Bytes	TIM DMA 连续传送长度 7 字
TIM_DMABurstLength_8Bytes	TIM DMA 连续传送长度 8 字
TIM_DMABurstLength_9Bytes	TIM DMA 连续传送长度 9 字
TIM_DMABurstLength_10Bytes	TIM DMA 连续传送长度 10 字
TIM_DMABurstLength_11Bytes	TIM DMA 连续传送长度 11 字
TIM_DMABurstLength_12Bytes	TIM DMA 连续传送长度 12 字
TIM_DMABurstLength_13Bytes	TIM DMA 连续传送长度 13 字
TIM_DMABurstLength_14Bytes	TIM DMA 连续传送长度 14 字
TIM_DMABurstLength_15Bytes	TIM DMA 连续传送长度 15 字
TIM_DMABurstLength_16Bytes	TIM DMA 连续传送长度 16 字
TIM_DMABurstLength_17Bytes	TIM DMA 连续传送长度 17 字
TIM_DMABurstLength_18Bytes	TIM DMA 连续传送长度 18 字

例：

```
/* Configures the TIM2 DMA Interface to transfer 1 byte and to use
the CCR1 as base address */
TIM_DMAConfig(TIM2, TIM_DMABase_CCR1, TIM_DMABurstLength_1Byte)
```

19.2.11 函数TIM_DMACmd

Table 485. 描述了函数TIM_DMACmd

Table 485. 函数 TIM_DMACmd

函数名	TIM_DMACmd
函数原形	void TIM_DMACmd(TIM_TypeDef* TIMx, u16 TIM_DMASource, FunctionalState Newstate)
功能描述	使能或者失能指定的 TIMx 的 DMA 请求
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_DMASource：待使能或者失能的 TIM 中断源 参阅 Section：TIM_DMASource 查阅更多该参数允许取值范围
输入参数 3	NewState：DMA 请求的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM_DMASource

输入参数 TIM_DMASource 使能或者失能 TIM 的中断。可以取下表的值。

Table 486. TIM_DMASource 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 DMA 源

例：

```
/* TIM2 Capture Compare 1 DMA Request Configuration */
TIM_DMACmd(TIM2, TIM_DMA_CC1, ENABLE);
```

19.2.12 函数TIM_InternalClockConfig

Table 487. 描述了函数TIM_InternalClockConfig

Table 487. 函数 TIM_InternalClockConfig

函数名	TIM_InternalClockConfig
函数原形	void TIM_DMACmd(TIM_TypeDef* TIMx, u16 TIM_DMASource, FunctionalState Newstate)
功能描述	设置 TIMx 内部时钟
输入参数	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Selects the internal clock for TIM2 */
TIM_InternalClockConfig(TIM2);
```

19.2.13 函数TIM_ITRxExternalClockConfig

Table 488. 描述了函数TIM_ITRxExternalClockConfig

Table 488. 函数 TIM_ITRxExternalClockConfig

函数名	TIM_ITRxExternalClockConfig
函数原形	void TIM_ITRxExternalClockConfig(TIM_TypeDef* TIMx, u16 TIM_InputTriggerSource)
功能描述	设置 TIMx 内部触发为外部时钟模式
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_InputTriggerSource：输入触发源 参阅 Section：TIM_InputTriggerSource 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM_InputTriggerSource

TIM_InputTriggerSource 选择 TIM 输入触发。见 Table 489. 参阅该参数的取值。

Table 489. TIM_InputTriggerSource 值

TIM_InputTriggerSource	描述
TIM_TS_ITR0	TIM 内部触发 0
TIM_TS_ITR1	TIM 内部触发 1
TIM_TS_ITR2	TIM 内部触发 2
TIM_TS_ITR3	TIM 内部触发 3

例：

```
/* TIM2 internal trigger 3 used as clock source */
TIM_ITRxExternalClockConfig(TIM2, TIM_TS_ITR3);
```

19.2.14 函数TIM_TIxExternalClockConfig

Table 490. 描述了函数TIM_TIxExternalClockConfig

Table 490. 函数 TIM_TIxExternalClockConfig

函数名	TIM_TIxExternalClockConfig
函数原形	void TIM_TIxExternalClockConfig(TIM_TypeDef* TIMx, u16 TIM_TIxExternalCLKSource, u8 TIM_ICPolarity, u8 ICFilter)
功能描述	设置 TIMx 触发为外部时钟
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_ TIxExternalCLKSource：触发源 参阅 Section：TIM_ TIxExternalCLKSource 查阅更多该参数允许取值范围
输入参数 3	TIM_ ICPolarity：指定的 TI 极性 参阅 Section：TIM_ ICPolarity 查阅更多该参数允许取值范围
输入参数 4	ICFilter：指定的输入比较滤波器。该参数取值在 0x0 和 0xF 之间。
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM_TIxExternalCLKSource

TIM_TIxExternalCLKSource选择TIMx外部时钟源。见Table 491. 参阅该参数的取值。

Table 491. TIM_TIxExternalCLKSource 值

TIM_TIxExternalCLKSource	描述
TIM_TS_TI1FP1	TIM IC1 连接到 TI1
TIM_TS_TI1FP2	TIM IC2 连接到 TI2
TIM_TS_TI1F_ED	TIM IC1 连接到 TI1：使用边沿探测

例：

```
/* Selects the TI1 as clock for TIM2: the external clock is
connected to TI1 input pin, the rising edge is the active edge and
no filter sampling is done (ICFilter = 0) */
TIM_TIxExternalClockConfig(TIM2, TIM_TS_TI1FP1,
TIM_ICPolarity_Rising, 0);
```

19.2.15 函数TIM_ETRClockMode1Config

Table 492. 描述了函数TIM_ETRClockMode1Config

Table 492. 函数 TIM_ETRClockMode1Config

函数名	TIM_ETRClockMode1Config
函数原形	void TIM_ETRClockMode1Config(TIM_TypeDef* TIMx, u16 TIM_ExtTRGPrescaler, u16 TIM_ExtTRGPolarity, u16 ExtTRGFilter)
功能描述	配置 TIMx 外部时钟模式 1
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_ExtTRGPrescaler：外部触发预分频 参阅 Section：TIM_ExtTRGPrescaler 查阅更多该参数允许取值范围
输入参数 3	TIM_ExtTRGPolarity：外部时钟极性 参阅 Section：TIM_ExtTRGPolarity 查阅更多该参数允许取值范围
输入参数 4	ExtTRGFilter：外部触发滤波器。该参数取值在 0x0 和 0xF 之间。
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM_ExtTRGPrescaler

TIM_ExtTRGPrescaler设置TIMx外部触发预分频。见Table 493. 参阅该参数的取值。

Table 493. TIM_ExtTRGPrescaler 值

TIM_ExtTRGPrescaler	描述
TIM_ExtTRGPSC_OFF	TIM ETRP 预分频 OFF
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

TIM_ExtTRGPolarity

TIM_ExtTRGPolarity设置TIMx外部触发极性。见Table 494. 参阅该参数的取值。

Table 494. TIM_ExtTRGPolarity 值

TIM_ExtTRGPolarity	描述
TIM_ExtTRGPolarity_Inverted	TIM 外部触发极性翻转：低电平或下降沿有效
TIM_ExtTRGPolarity_NonInverted	TIM 外部触发极性非翻转：高电平或上升沿有效

例：

```
/* Selects the external clock Mode 1 for TIM2: the external clock is
connected to ETR input pin, the rising edge is the active edge, no
filter sampling is done (ExtTRGFilter = 0) and the prescaler is
fixed to TIM_ExtTRGPSC_DIV2 */
TIM_ExternalCLK1Config(TIM2, TIM_ExtTRGPSC_DIV2,
```

```
TIM_ExtTRGPolarity_NonInverted, 0x0);
```

19.2.16 函数TIM_ETRClockMode2Config

Table 495. 描述了函数TIM_ETRClockMode2Config

Table 495. 函数 TIM_ETRClockMode2Config

函数名	TIM_ETRClockMode2Config
函数原形	void TIM_ETRClockMode2Config(TIM_TypeDef* TIMx, u16 TIM_ExtTRGPrescaler, u16 TIM_ExtTRGPolarity, u16 ExtTRGFilter)
功能描述	配置TIMx外部时钟模式2
输入参数1	TIMx：x可以是2，3或者4，来选择TIM外设
输入参数2	TIM_ExtTRGPrescaler：外部触发预分频 参阅Section：TIM_ExtTRGPrescaler查阅更多该参数允许取值范围
输入参数3	TIM_ExtTRGPolarity：外部时钟极性 参阅Section：TIM_ExtTRGPolarity查阅更多该参数允许取值范围
输入参数4	ExtTRGFilter：外部触发滤波器。该参数取值在0x0和0xF之间。
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Selects the external clock Mode 2 for TIM2: the external clock is
connected to ETR input pin, the rising edge is the active edge, no
filter sampling is done (ExtTRGFilter = 0) and the prescaler is
fixed to TIM_ExtTRGPSC_DIV2 */
TIM_ExternalCLK2Config(TIM2, TIM_ExtTRGPSC_DIV2,
TIM_ExtTRGPolarity_NonInverted, 0x0);
```

19.2.17 函数TIM_ETRConfig

Table 496. 描述了函数TIM_ETRConfig

Table 496. 函数 TIM_ETRConfig

函数名	TIM_ETRConfig
函数原形	void TIM_ETRConfig(TIM_TypeDef* TIMx, u16 TIM_ExtTRGPrescaler, u16 TIM_ExtTRGPolarity, u8 ExtTRGFilter)
功能描述	配置 TIMx 外部触发
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_ExtTRGPrescaler：外部触发预分频 参阅 Section：TIM_ExtTRGPrescaler 查阅更多该参数允许取值范围
输入参数 3	TIM_ExtTRGPolarity：外部时钟极性 参阅 Section：TIM_ExtTRGPolarity 查阅更多该参数允许取值范围
输入参数 4	ExtTRGFilter：外部触发滤波器。该参数取值在 0x0 和 0xF 之间。
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Configure the External Trigger (ETR) for TIM2: the rising edge is
the active edge, no filter sampling is done (ExtTRGFilter = 0) and
the prescaler is fixed to TIM_ExtTRGPSC_DIV2 */
TIM_ExternalCLK2Config(TIM2, TIM_ExtTRGPSC_DIV2,
TIM_ExtTRGPolarity_NonInverted, 0x0);
```

19.2.18 函数TIM_SelectInputTrigger

Table 497. 描述了函数TIM_SelectInputTrigger

Table 497. 函数 TIM_SelectInputTrigger

函数名	TIM_SelectInputTrigger
函数原形	void TIM_SelectInputTrigger(TIM_TypeDef* TIMx, u16 TIM_InputTriggerSource)
功能描述	选择 TIMx 输入触发源
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_InputTriggerSource：输入触发源 参阅 Section：TIM_InputTriggerSource 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM_InputTriggerSource

TIM_InputTriggerSource选择TIMx输入触发源。见Table 498. 参阅该参数的取值。

Table 498. TIM_InputTriggerSource 值

TIM_InputTriggerSource	描述
TIM_TS_ITR0	TIM 内部触发 0
TIM_TS_ITR1	TIM 内部触发 1
TIM_TS_ITR2	TIM 内部触发 2
TIM_TS_ITR3	TIM 内部触发 3
TIM_TS_TI1F_ED	TIM TL1 边沿探测器
TIM_TS_TI1FP1	TIM 经滤波定时器输入 1
TIM_TS_TI2FP2	TIM 经滤波定时器输入 2
TIM_TS_ETRF	TIM 外部触发输入

例：

```
/* Selects the Internal Trigger 3 as input trigger fot TIM2 */
void TIM_SelectInputTrigger(TIM2, TIM_TS_ITR3);
```

19.2.19 函数TIM_PrescalerConfig

Table 499. 描述了函数TIM_PrescalerConfig

Table 499. 函数 TIM_PrescalerConfig

函数名	TIM_PrescalerConfig
函数原形	void TIM_PrescalerConfig(TIM_TypeDef* TIMx, u16 Prescaler,u16 TIM_PSCReloadMode)
功能描述	设置 TIMx 预分频
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_PSCReloadMode：预分频重载模式 参阅 Section：TIM_PSCReloadMode 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM_PSCReloadMode

TIM_PSCReloadMode选择预分频重载模式。见Table 500. 参阅该参数的取值。

Table 500. TIM_PSCReloadMode 值

TIM_PSCReloadMode	描述
TIM_PSCReloadMode_Update	TIM 预分频值在更新事件装入
TIM_PSCReloadMode_Immediate	TIM 预分频值即时装入

例：

```
/* Configures the TIM2 new Prescaler value */
u16 TIMPrescaler = 0xFF00;
TIM_PrescalerConfig(TIM2, TIMPrescaler,
TIM_PSCReloadMode_Immediate);
```

19.2.20 函数TIM_CounterModeConfig

Table 501. 描述了函数TIM_CounterModeConfig

Table 501. 函数 TIM_CounterModeConfig

函数名	TIM_CounterModeConfig
函数原形	void TIM_CounterModeConfig(TIM_TypeDef* TIMx, u16 TIM_CounterMode)
功能描述	设置 TIMx 计数器模式
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_CounterMode：待使用的计数器模式 参阅 Section：TIM_CounterMode 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Selects the Center Aligned counter Mode 1 for the TIM2 */
TIM_CounterModeConfig(TIM2, TIM_Counter_CenterAligned1);
```

19.2.21 函数TIM_ForcedOC1Config

Table 502. 描述了函数TIM_ForcedOC1Config

Table 502. 函数 TIM_ForcedOC1Config

函数名	TIM_ForcedOC1Config
函数原形	void TIM_ForcedOC1Config(TIM_TypeDef* TIMx, u16 TIM_ForcedAction)
功能描述	置 TIMx 输出 1 为活动或者非活动电平
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_ForcedAction：输出信号的设置动作 参阅 Section：TIM_ForcedAction 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM_ForcedAction

输出信号的设置动作取值见下表。

Table 503. TIM_ForcedAction 值

TIM_ForcedAction	描述
TIM_ForcedAction_Active	置为 OCxREF 上的活动电平
TIM_ForcedAction_InActive	置为 OCxREF 上的非活动电平

例：

```
/* Forces the TIM2 Output Compare 1 signal to the active level */
TIM_ForcedOC1Config(TIM2, TIM_ForcedAction_Active);
```

19.2.22 函数TIM_ForcedOC2Config

Table 504. 描述了函数TIM_ForcedOC2Config

Table 504. 函数 TIM_ForcedOC2Config

函数名	TIM_ForcedOC2Config
函数原形	void TIM_ForcedOC2Config(TIM_TypeDef* TIMx, u16 TIM_ForcedAction)
功能描述	置 TIMx 输出 2 为活动或者非活动电平
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_ForcedAction：输出信号的设置动作 参阅 Section：TIM_ForcedAction 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Forces the TIM2 Output Compare 2 signal to the active level */
TIM_ForcedOC2Config(TIM2, TIM_ForcedAction_Active);
```

19.2.23 函数TIM_ForcedOC3Config

Table 505. 描述了函数TIM_ForcedOC3Config

Table 505. 函数 TIM_ForcedOC3Config

函数名	TIM_ForcedOC3Config
函数原形	void TIM_ForcedOC3Config(TIM_TypeDef* TIMx, u16 TIM_ForcedAction)
功能描述	置 TIMx 输出 3 为活动或者非活动电平
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_ForcedAction：输出信号的设置动作 参阅 Section：TIM_ForcedAction 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Forces the TIM2 Output Compare 3 signal to the active level */
TIM_ForcedOC3Config(TIM2, TIM_ForcedAction_Active);
```

19.2.24 函数TIM_ForcedOC4Config

Table 506. 描述了函数TIM_ForcedOC4Config

Table 506. 函数 TIM_ForcedOC4Config

函数名	TIM_ForcedOC4Config
函数原形	void TIM_ForcedOC4Config(TIM_TypeDef* TIMx, u16 TIM_ForcedAction)
功能描述	置 TIMx 输出 4 为活动或者非活动电平
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_ForcedAction：输出信号的设置动作 参阅 Section：TIM_ForcedAction 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Forces the TIM2 Output Compare 4 signal to the active level */
TIM_ForcedOC4Config(TIM2, TIM_ForcedAction_Active);
```

19.2.25 函数TIM_ARRPreloadConfig

Table 507. 描述了函数TIM_ARRPreloadConfig

Table 507. 函数 TIM_ARRPreloadConfig

函数名	TIM_ARRPreloadConfig
函数原形	void TIM_ARRPreloadConfig(TIM_TypeDef* TIMx, FunctionalState Newstate)
功能描述	使能或者失能 TIMx 在 ARR 上的预装载寄存器
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	NewState: TIM_CR1 寄存器 ARPE 位的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enables the TIM2 Preload on ARR Register */
TIM_ARRPreloadConfig(TIM2, ENABLE);
```

19.2.26 函数TIM_SelectCCDMA

Table 508. 描述了函数TIM_SelectCCDMA

Table 508. 函数 TIM_SelectCCDMA

函数名	TIM_SelectCCDMA
函数原形	void TIM_SelectCCDMA(TIM_TypeDef* TIMx, FunctionalState Newstate)
功能描述	选择 TIMx 外设的捕获比较 DMA 源
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	NewState: 捕获比较 DMA 源的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Selects the TIM2 Capture Compare DMA source */
TIM_SelectCCDMA(TIM2, ENABLE);
```

19.2.27 函数TIM_OC1PreloadConfig

Table 509. 描述了函数TIM_OC1PreloadConfig

Table 509. 函数 TIM_OC1PreloadConfig

函数名	TIM_OC1PreloadConfig
函数原形	void TIM_OC1PreloadConfig(TIM_TypeDef* TIMx, u16 TIM_OCPreload)
功能描述	使能或者失能 TIMx 在 CCR1 上的预装载寄存器
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_OCPreload：输出比较预装载状态 参阅 Section：TIM_OCPreload 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM_OCPreload

输出比较预装载状态可以使能或者失能如下表。

Table 510. TIM_OCPreload 值

TIM_OCPreload	描述
TIM_OCPreload_Enable	TIMx 在 CCR1 上的预装载寄存器使能
TIM_OCPreload_Disable	TIMx 在 CCR1 上的预装载寄存器失能

例：

```
/* Enables the TIM2 Preload on CC1 Register */
TIM_OC1PreloadConfig(TIM2, TIM_OCPreload_Enable);
```

19.2.28 函数TIM_OC2PreloadConfig

Table 511. 描述了函数TIM_OC2PreloadConfig

Table 511. 函数 TIM_OC2PreloadConfig

函数名	TIM_OC2PreloadConfig
函数原形	void TIM_OC2PreloadConfig(TIM_TypeDef* TIMx, u16 TIM_OCPreload)
功能描述	使能或者失能 TIMx 在 CCR2 上的预装载寄存器
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_OCPreload：输出比较预装载状态 参阅 Section：TIM_OCPreload 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enables the TIM2 Preload on CC2 Register */
TIM_OC2PreloadConfig(TIM2, TIM_OCPreload_Enable);
```

19.2.29 函数TIM_OC3PreloadConfig

Table 512. 描述了函数TIM_OC3PreloadConfig

Table 512. 函数 TIM_OC3PreloadConfig

函数名	TIM_OC3PreloadConfig
函数原形	void TIM_OC3PreloadConfig(TIM_TypeDef* TIMx, u16 TIM_OCPreload)
功能描述	使能或者失能 TIMx 在 CCR3 上的预装载寄存器
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_OCPreload：输出比较预装载状态 参阅 Section：TIM_OCPreload 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enables the TIM2 Preload on CC3 Register */
TIM_OC3PreloadConfig(TIM2, TIM_OCPreload_Enable);
```

19.2.30 函数TIM_OC4PreloadConfig

Table 513. 描述了函数TIM_OC4PreloadConfig

Table 513. 函数 TIM_OC4PreloadConfig

函数名	TIM_OC4PreloadConfig
函数原形	void TIM_OC4PreloadConfig(TIM_TypeDef* TIMx, u16 TIM_OCPreload)
功能描述	使能或者失能 TIMx 在 CCR4 上的预装载寄存器
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_OCPreload：输出比较预装载状态 参阅 Section：TIM_OCPreload 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enables the TIM2 Preload on CC4 Register */
TIM_OC4PreloadConfig(TIM2, TIM_OCPreload_Enable);
```

19.2.31 函数TIM_OC1FastConfig

Table 514. 描述了函数TIM_OC1FastConfig

Table 514. 函数 TIM_OC1FastConfig

函数名	TIM_OC1FastConfig
函数原形	void TIM_OC1FastConfig(TIM_TypeDef* TIMx, u16 TIM_OCFast)
功能描述	设置 TIMx 捕获比较 1 快速特征
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_OCFast：输出比较快速特征状态 参阅 Section：TIM_OCFast 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM_OCFast

输出比较快速特征性能可以使能或者失能如下表。

Table 515. TIM_OCPreload 值

TIM_OCFast	描述
TIM_OCFast_Enable	TIMx 输出比较快速特征性能使能
TIM_OCFast_Disable	TIMx 输出比较快速特征性能失能

例：

```
/* Use the TIM2 OC1 in fast Mode */
TIM_OC1FastConfig(TIM2, TIM_OCFast_Enable);
```

19.2.32 函数TIM_OC2FastConfig

Table 516. 描述了函数TIM_OC2FastConfig

Table 516. 函数 TIM_OC2FastConfig

函数名	TIM_OC2FastConfig
函数原形	void TIM_OC2FastConfig(TIM_TypeDef* TIMx, u16 TIM_OCFast)
功能描述	设置 TIMx 捕获比较 2 快速特征
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_OCFast：输出比较快速特征状态 参阅 Section：TIM_OCFast 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Use the TIM2 OC2 in fast Mode */
TIM_OC2FastConfig(TIM2, TIM_OCFast_Enable);
```

19.2.33 函数TIM_OC3FastConfig

Table 517. 描述了函数TIM_OC3FastConfig

Table 517. 函数 TIM_OC3FastConfig

函数名	TIM_OC3FastConfig
函数原形	void TIM_OC3FastConfig(TIM_TypeDef* TIMx, u16 TIM_OCFast)
功能描述	设置 TIMx 捕获比较 3 快速特征
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_OCFast：输出比较快速特征状态 参阅 Section：TIM_OCFast 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Use the TIM2 OC3 in fast Mode */
TIM_OC3FastConfig(TIM2, TIM_OCFast_Enable);
```

19.2.34 函数TIM_OC4FastConfig

Table 518. 描述了函数TIM_OC4FastConfig

Table 518. 函数 TIM_OC4FastConfig

函数名	TIM_OC4FastConfig
函数原形	void TIM_OC4FastConfig(TIM_TypeDef* TIMx, u16 TIM_OCFast)
功能描述	设置 TIMx 捕获比较 4 快速特征
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_OCFast：输出比较快速特征状态 参阅 Section：TIM_OCFast 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Use the TIM2 OC4 in fast Mode */
TIM_OC4FastConfig(TIM2, TIM_OCFast_Enable);
```

19.2.35 函数TIM_ClearOC1Ref

Table 519. 描述了函数TIM_ClearOC1Ref

Table 519. 函数 TIM_ClearOC1Ref

函数名	TIM_ClearOC1Ref
函数原形	void TIM_ClearOC1Ref(TIM_TypeDef* TIMx, u16 TIM_OCClear)
功能描述	在一个外部事件时清除或者保持 OCREF1 信号
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_OCClear：输出比较清除使能位状态 参阅 Section：TIM_OCClear 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM_OCClear

输出比较清除使能位的值列举如下表。

Table 520. TIM_OCClear 值

TIM_OCClear	描述
TIM_OCClear_Enable	TIMx 输出比较清除使能
TIM_OCClear_Disable	TIMx 输出比较清除失能

例：

```
/* Enable the TIM2 Channel1 Ouput Compare Refence clear bit */
TIM_ClearOC1Ref(TIM2, TIM_OCClear_Enable);
```

19.2.36 函数TIM_ClearOC2Ref

Table 521. 描述了函数TIM_ClearOC2Ref

Table 521. 函数 TIM_ClearOC2Ref

函数名	TIM_ClearOC2Ref
函数原形	void TIM_ClearOC2Ref(TIM_TypeDef* TIMx, u16 TIM_OCClear)
功能描述	在一个外部事件时清除或者保持 OCREF2 信号
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_OCClear：输出比较清除使能位状态 参阅 Section：TIM_OCClear 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable the TIM2 Channel2 Ouput Compare Refence clear bit */
TIM_ClearOC2Ref(TIM2, TIM_OCClear_Enable);
```

19.2.37 函数TIM_ClearOC3Ref

Table 522. 描述了函数TIM_ClearOC3Ref

Table 522. 函数 TIM_ClearOC3Ref

函数名	TIM_ClearOC3Ref
函数原形	void TIM_ClearOC3Ref(TIM_TypeDef* TIMx, u16 TIM_OCClear)
功能描述	在一个外部事件时清除或者保持 OCREF3 信号
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_OCClear：输出比较清除使能位状态 参阅 Section：TIM_OCClear 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable the TIM2 Channel3 Ouput Compare Refence clear bit */
TIM_ClearOC3Ref(TIM2, TIM_OCClear_Enable);
```

19.2.38 函数TIM_ClearOC4Ref

Table 523. 描述了函数TIM_ClearOC4Ref

Table 523. 函数 TIM_ClearOC4Ref

函数名	TIM_ClearOC4Ref
函数原形	void TIM_ClearOC4Ref(TIM_TypeDef* TIMx, u16 TIM_OCClear)
功能描述	在一个外部事件时清除或者保持 OCREF4 信号
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_OCClear：输出比较清除使能位状态 参阅 Section：TIM_OCClear 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable the TIM2 Channel4 Ouput Compare Refence clear bit */
TIM_ClearOC4Ref(TIM2, TIM_OCClear_Enable);
```

19.2.39 函数TIM_UpdateDisableConfig

Table 524. 描述了函数TIM_UpdateDisableConfig

Table 524. 函数 TIM_UpdateDisableConfig

函数名	TIM_UpdateDisableConfig
函数原形	void TIM_UpdateDisableConfig(TIM_TypeDef* TIMx, FunctionalState Newstate)
功能描述	使能或者失能 TIMx 更新事件
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	NewState：TIMx_CR1 寄存器 UDIS 位的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enables the Update event for TIM2 */
TIM_UpdateDisableConfig(TIM2, DISABLE);
```

19.2.40 函数TIM_EncoderInterfaceConfig

Table 525. 描述了函数TIM_EncoderInterfaceConfig

Table 525. 函数 TIM_EncoderInterfaceConfig

函数名	TIM_EncoderInterfaceConfig
函数原形	void TIM_EncoderInterfaceConfig(TIM_TypeDef* TIMx, u8 TIM_EncoderMode, u8 TIM_IC1Polarity, u8 TIM_IC2Polarity)
功能描述	设置 TIMx 编码界面
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_EncoderMode：触发源 参阅 Section：TIM_EncoderMode 查阅更多该参数允许取值范围
输入参数 3	TIM_ IC1Polarity：TI1 极性 参阅 Section：TIM_ ICPolarity 查阅更多该参数允许取值范围
输入参数 4	TIM_ IC2Polarity：TI2 极性 参阅 Section：TIM_ ICPolarity 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM_EncoderMode

TIM_EncoderMode选择TIMx编码模式。见Table 526. 参阅该参数的取值。

Table 526. TIM_EncoderMode 值

TIM_EncoderMode	描述
TIM_EncoderMode_TI1	使用 TIM 编码模式 1
TIM_EncoderMode_TI1	使用 TIM 编码模式 2
TIM_EncoderMode_TI12	使用 TIM 编码模式 3

例：

```
/* Configures the encoder mode TI1 for TIM2 */
TIM_EncoderInterfaceConfig(TIM2, TIM_EncoderMode_TI1,
```

```
TIM_ICPolarity_Rising, TIM_ICPolarity_Rising);
```

19.2.41 函数TIM_GenerateEvent

Table 527. 描述了函数TIM_GenerateEvent

Table 527. 函数 TIM_GenerateEvent

函数名	TIM_GenerateEvent
函数原形	void TIM_GenerateEvent(TIM_TypeDef* TIMx, u16 TIM_EventSource)
功能描述	设置 TIMx 事件由软件产生
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_EventSource：TIM 软件事件源 参阅 Section：TIM_EventSource 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM_EventSource

TIM_EventSource 选择 TIM 软件事件源。见 Table 528. 参阅该参数的取值。

Table 528. TIM_EventSource 值

TIM_EventSource	描述
TIM_EventSource_Update	TIM 更新事件源
TIM_EventSource_CC1	TIM 捕获比较 1 事件源
TIM_EventSource_CC2	TIM 捕获比较 2 事件源
TIM_EventSource_CC3	TIM 捕获比较 3 事件源
TIM_EventSource_CC4	TIM 捕获比较 4 事件源
TIM_EventSource_Trigger	TIM 触发事件源

例：

```
/* Selects the Trigger software Event generation for TIM2 */
TIM_GenerateEvent(TIM2, TIM_EventSource_Trigger);
```

19.2.42 函数TIM_OC1PolarityConfig

Table 529. 描述了函数TIM_OC1PolarityConfig

Table 529. 函数 TIM_OC1PolarityConfig

函数名	TIM_OC1PolarityConfig
函数原形	void TIM_OC1PolarityConfig(TIM_TypeDef* TIMx, u16 TIM_OCPolarity)
功能描述	设置 TIMx 通道 1 极性
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_OCPolarity：输出比较极性 参阅 Section：TIM_OCPolarity 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Selects the Polarity high for TIM2 channel 1 output compare */
TIM_OC1PolarityConfig(TIM2, TIM_OCPolarity_High);
```

19.2.43 函数TIM_OC2PolarityConfig

Table 530. 描述了函数TIM_OC2PolarityConfig

Table 530. 函数 TIM_OC2PolarityConfig

函数名	TIM_OC2PolarityConfig
函数原形	void TIM_OC2PolarityConfig(TIM_TypeDef* TIMx, u16 TIM_OCPolarity)
功能描述	设置 TIMx 通道 2 极性
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_OCPolarity：输出比较极性 参阅 Section：TIM_OCPolarity 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Selects the Polarity high for TIM2 channel 2 output compare */
TIM_OC1PolarityConfig(TIM2, TIM_OCPolarity_High);
```

19.2.44 函数TIM_OC3PolarityConfig

Table 531. 描述了函数TIM_OC1PolarityConfig

Table 531. 函数 TIM_OC1PolarityConfig

函数名	TIM_OC3PolarityConfig
函数原形	void TIM_OC3PolarityConfig(TIM_TypeDef* TIMx, u16 TIM_OCPolarity)
功能描述	设置 TIMx 通道 3 极性
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_OCPolarity：输出比较极性 参阅 Section：TIM_OCPolarity 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Selects the Polarity high for TIM2 channel 3 output compare */
TIM_OC1PolarityConfig(TIM2, TIM_OCPolarity_High);
```

19.2.45 函数TIM_OC4PolarityConfig

Table 532. 描述了函数TIM_OC4PolarityConfig

Table 532. 函数 TIM_OC4PolarityConfig

函数名	TIM_OC4PolarityConfig
函数原形	void TIM_OC4PolarityConfig(TIM_TypeDef* TIMx, u16 TIM_OCPolarity)
功能描述	设置 TIMx 通道 4 极性
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_OCPolarity：输出比较极性 参阅 Section：TIM_OCPolarity 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Selects the Polarity high for TIM2 channel 4 output compare */
TIM_OC1PolarityConfig(TIM2, TIM_OCPolarity_High);
```

19.2.46 函数TIM_UpdateRequestConfig

Table 533. 描述了函数TIM_UpdateRequestConfig

Table 533. 函数 TIM_UpdateRequestConfig

函数名	TIM_UpdateRequestConfig
函数原形	void TIM_UpdateRequestConfig(TIM_TypeDef* TIMx, u16 TIM_UpdateSource)
功能描述	设置 TIMx 更新请求源
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_UpdateSource：TIM 更新请求源 参阅 Section：TIM_UpdateSource 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM_UpdateSource

TIM_UpdateSource 选择 TIM 更新源。见 Table 534. 参阅该参数的取值。

Table 534. TIM_UpdateSource 值

TIM_UpdateSource	描述
TIM_UpdateSource_Global	生成重复的脉冲：在更新事件时计数器不停止
TIM_UpdateSource_Regular	生成单一的脉冲：计数器在下一个更新事件停止

例：

```
/* Selects the regular update source for TIM2 */
TIM_UpdateRequestConfig(TIM2, TIM_UpdateSource_Regular);
```

19.2.47 函数TIM_SelectHallSensor

Table 535. 描述了函数TIM_SelectHallSensor

Table 535. 函数 TIM_SelectHallSensor

函数名	TIM_SelectHallSensor
函数原形	void TIM_SelectHallSensor(TIM_TypeDef* TIMx, FunctionalState Newstate)
功能描述	使能或者失能 TIMx 霍尔传感器接口
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	NewState：TIMx 霍尔传感器接口的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Selects the Hall Sensor Interface for TIM2 */
TIM_SelectHallSensor(TIM2, ENABLE);
```

19.2.48 函数TIM_SelectOnePulseMode

Table 536. 描述了函数TIM_SelectOnePulseMode

Table 536. 函数 TIM_SelectOnePulseMode

函数名	TIM_SelectOnePulseMode
函数原形	void TIM_SelectOnePulseMode(TIM_TypeDef* TIMx, u16 TIM_OPMode)
功能描述	设置 TIMx 单脉冲模式
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_OPMode：OPM 模式 参阅 Section：TIM_OPMode 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM_OPMode

TIM_OPMode 选择 TIM 更新源。见 Table 537. 参阅该参数的取值。

Table 537. TIM_OPMode 值

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	生成单一的脉冲：计数器在下一个更新事件停止

例：

```
/* Selects the Single One Pulse Mode for TIM2 */
TIM_SelectOnePulseMode(TIM2, TIM_OPMode_Single);
```

19.2.49 函数TIM_SelectOutputTrigger

Table 538. 描述了函数TIM_SelectOutputTrigger

Table 538. 函数 TIM_SelectOutputTrigger

函数名	TIM_SelectOutputTrigger
函数原形	void TIM_SelectOutputTrigger(TIM_TypeDef* TIMx, u16 TIM_TRGOSource)
功能描述	选择 TIMx 触发输出模式
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_TRGOSource：触发输出模式 参阅 Section：TIM_TRGOSource 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM_TRGOSource

TIM_TRGOSource 选择 TIM 触发输出源。见 Table 539. 参阅该参数的取值。

Table 539. TIM_TRGOSource 值

TIM_TRGOSource	描述
TIM_TRGOSource_Reset	使用寄存器 TIM_EGR 的 UG 位作为触发输出（TRGO）
TIM_TRGOSource_Enable	使用计数器使能 CEN 作为触发输出（TRGO）
TIM_TRGOSource_Update	使用更新事件作为触发输出（TRGO）
TIM_TRGOSource_OC1	一旦捕获或者比较匹配发生，当标志位 CC1F 被设置时触发输出发送一个肯定脉冲（TRGO）
TIM_TRGOSource_OC1Ref	使用 OC1REF 作为触发输出（TRGO）
TIM_TRGOSource_OC2Ref	使用 OC2REF 作为触发输出（TRGO）
TIM_TRGOSource_OC3Ref	使用 OC3REF 作为触发输出（TRGO）
TIM_TRGOSource_OC4Ref	使用 OC4REF 作为触发输出（TRGO）

例：

```
/* Selects the update event as Trigger Output for TIM2 */
TIM_SelectOutputTrigger(TIM2, TIM_TRGOSource_Update);
```

19.2.50 函数TIM_SelectSlaveMode

Table 540. 描述了函数TIM_SelectSlaveMode

Table 540. 函数 TIM_SelectSlaveMode

函数名	TIM_SelectSlaveMode
函数原形	void TIM_SelectSlaveMode(TIM_TypeDef* TIMx, u16 TIM_SlaveMode)
功能描述	选择 TIMx 从模式
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_SlaveMode：TIM 从模式 参阅 Section：TIM_SlaveMode 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM_SlaveMode

TIM_SlaveMode 选择 TIM 从模式。见 Table 541. 参阅该参数的取值。

Table 541. TIM_SlaveMode 值

TIM_SlaveMode	描述
TIM_SlaveMode_Reset	选中触发信号（TRGI）的上升沿重初始化计数器并触发寄存器的更新
TIM_SlaveMode_Gated	当触发信号（TRGI）为高电平计数器时钟使能
TIM_SlaveMode_Trigger	计数器在触发（TRGI）的上升沿开始
TIM_SlaveMode_External1	选中触发（TRGI）的上升沿作为计数器时钟

例：

```
/* Selects the Gated Mode as Slave Mode for TIM2 */
TIM_SelectSlaveMode(TIM2, TIM_SlaveMode_Gated);
```

19.2.51 函数TIM_SelectMasterSlaveMode

Table 542. 描述了函数TIM_SelectMasterSlaveMode

Table 542. 函数 TIM_SelectMasterSlaveMode

函数名	TIM_SelectMasterSlaveMode
函数原形	void TIM_SelectMasterSlaveMode(TIM_TypeDef* TIMx, u16 TIM_MasterSlaveMode)
功能描述	设置或者重置 TIMx 主/从模式
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_MasterSlaveMode：定时器主/从模式 参阅 Section：TIM_MasterSlaveMode 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM_MasterSlaveMode

TIM_MasterSlaveMode 选择 TIM 主/从模式。见 Table 543. 参阅该参数的取值。

Table 543. TIM_MasterSlaveMode 值

TIM_MasterSlaveMode	描述
TIM_MasterSlaveMode_Enable	TIM 主/从模式使能
TIM_MasterSlaveMode_Disable	TIM 主/从模式失能

例：

```
/* Enables the Master Slave Mode for TIM2 */
TIM_SelectMasterSlaveMode(TIM2, TIM_MasterSlaveMode_Enable);
```

19.2.52 函数TIM_SetCounter

Table 544. 描述了函数TIM_SetCounter

Table 544. 函数 TIM_SetCounter

函数名	TIM_SetCounter
函数原形	void TIM_SetCounter(TIM_TypeDef* TIMx, u16 Counter)
功能描述	设置 TIMx 计数器寄存器值
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	Counter：计数器寄存器新值
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Sets the TIM2 new Counter value */
u16 TIMCounter = 0xFFFF;
TIM_SetCounter(TIM2, TIMCounter);
```

19.2.53 函数TIM_SetAutoreload

Table 545. 描述了函数TIM_ SetAutoreload

Table 545. 函数 TIM_ SetAutoreload

函数名	TIM_ SetAutoreload
函数原形	void TIM_SetCounter(TIM_TypeDef* TIMx, u16 Counter)
功能描述	设置 TIMx 自动重装载寄存器值
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	Autoreload：自动重装载寄存器新值
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Sets the TIM2 new Autoreload value */
u16 TIMAutoreload = 0xFFFF;
TIM_SetAutoreload(TIM2, TIMAutoreload);
```

19.2.54 函数TIM_SetCompare1

Table 546. 描述了函数TIM_SetCompare1

Table 546. 函数 TIM_SetCompare1

函数名	TIM_SetCompare1
函数原形	void TIM_SetCompare1(TIM_TypeDef* TIMx, u16 Compare1)
功能描述	设置 TIMx 捕获比较 1 寄存器值
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	Compare1：捕获比较 1 寄存器新值
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Sets the TIM2 new Output Compare 1 value */
u16 TIMCompare1 = 0x7FFF;
TIM_SetCompare1(TIM2, TIMCompare1);
```

19.2.55 函数TIM_SetCompare2

Table 547. 描述了函数TIM_SetCompare2

Table 547. 函数 TIM_SetCompare2

函数名	TIM_SetCompare2
函数原形	void TIM_SetCompare2(TIM_TypeDef* TIMx, u16 Compare2)
功能描述	设置 TIMx 捕获比较 2 寄存器值
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	Compare2：捕获比较 2 寄存器新值
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Sets the TIM2 new Output Compare 2 value */
u16 TIMCompare2 = 0x7FFF;
TIM_SetCompare2(TIM2, TIMCompare2);
```

19.2.56 函数TIM_SetCompare3

Table 548. 描述了函数TIM_SetCompare3

Table 548. 函数 TIM_SetCompare3

函数名	TIM_SetCompare3
函数原形	void TIM_SetCompare3(TIM_TypeDef* TIMx, u16 Compare3)
功能描述	设置 TIMx 捕获比较 3 寄存器值
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	Compare1：捕获比较 3 寄存器新值
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Sets the TIM2 new Output Compare 3 value */
u16 TIMCompare3 = 0x7FFF;
TIM_SetCompare3(TIM2, TIMCompare3);
```

19.2.57 函数TIM_SetCompare4

Table 549. 描述了函数TIM_SetCompare4

Table 549. 函数 TIM_SetCompare4

函数名	TIM_SetCompare4
函数原形	void TIM_SetCompare4(TIM_TypeDef* TIMx, u16 Compare4)
功能描述	设置 TIMx 捕获比较 4 寄存器值
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	Compare4：捕获比较 4 寄存器新值
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Sets the TIM2 new Output Compare 4 value */
u16 TIMCompare4 = 0x7FFF;
TIM_SetCompare4(TIM2, TIMCompare4);
```

19.2.58 函数TIM_SetIC1Prescaler

Table 550. 描述了函数TIM_SetIC1Prescaler

Table 550. 函数 TIM_SetIC1Prescaler

函数名	TIM_SetIC1Prescaler
函数原形	void TIM_SetIC1Prescaler(TIM_TypeDef* TIMx, u16 TIM_IC1Prescaler)
功能描述	设置 TIMx 输入捕获 1 预分频
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_IC1Prescaler：输入捕获 1 预分频 参阅 Section：TIM_IC1Prescaler 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Sets the TIM2 Input Capture 1 Prescaler */
TIM_SetIC1Prescaler(TIM2, TIM_ICPSC_Div2);
```

19.2.59 函数TIM_SetIC2Prescaler

Table 551. 描述了函数TIM_SetIC2Prescaler

Table 551. 函数 TIM_SetIC2Prescaler

函数名	TIM_SetIC2Prescaler
函数原形	void TIM_SetIC2Prescaler(TIM_TypeDef* TIMx, u16 TIM_IC2Prescaler)
功能描述	设置 TIMx 输入捕获 2 预分频
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_IC2Prescaler：输入捕获 2 预分频 参阅 Section：TIM_IC1Prescaler 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Sets the TIM2 Input Capture 2 Prescaler */
TIM_SetIC2Prescaler(TIM2, TIM_ICPSC_Div2);
```

19.2.60 函数TIM_SetIC3Prescaler

Table 552. 描述了函数TIM_SetIC3Prescaler

Table 552. 函数 TIM_SetIC3Prescaler

函数名	TIM_SetIC3Prescaler
函数原形	void TIM_SetIC3Prescaler(TIM_TypeDef* TIMx, u16 TIM_IC3Prescaler)
功能描述	设置 TIMx 输入捕获 3 预分频
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_IC1Prescaler：输入捕获 3 预分频 参阅 Section：TIM_IC1Prescaler 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Sets the TIM2 Input Capture 3 Prescaler */
TIM_SetIC3Prescaler(TIM2, TIM_ICPSC_Div2);
```

19.2.61 函数TIM_SetIC4Prescaler

Table 553. 描述了函数TIM_SetIC4Prescaler

Table 553. 函数 TIM_SetIC4Prescaler

函数名	TIM_SetIC4Prescaler
函数原形	void TIM_SetIC1Prescaler(TIM_TypeDef* TIMx, u16 TIM_IC4Prescaler)
功能描述	设置 TIMx 输入捕获 4 预分频
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_IC1Prescaler：输入捕获 4 预分频 参阅 Section：TIM_IC1Prescaler 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Sets the TIM2 Input Capture 4 Prescaler */
TIM_SetIC4Prescaler(TIM2, TIM_ICPSC_Div2);
```

19.2.62 函数TIM_SetClockDivision

Table 554. 描述了函数TIM_SetClockDivision

Table 554. 函数 TIM_SetClockDivision

函数名	TIM_SetClockDivision
函数原形	void TIM_SetClockDivision(TIM_TypeDef* TIMx, u16 TIM_CKD)
功能描述	设置 TIMx 的时钟分割值
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_CKD：时钟分割值 参阅 Section：TIM_ClockDivision 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Sets the TIM2 CKD value */
TIM_SetClockDivision(TIM2, TIM_CKD_DIV4);
```

19.2.63 函数TIM_GetCapture1

Table 555. 描述了函数TIM_GetCapture1

Table 555. 函数 TIM_GetCapture1

函数名	TIM_GetCapture1
函数原形	u16 TIM_GetCapture1(TIM_TypeDef* TIMx)
功能描述	获得 TIMx 输入捕获 1 的值
输入参数	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输出参数	无
返回值	输入捕获 1 的值
先决条件	无
被调用函数	无

例：

```
/* Gets the Input Capture 1 value of the TIM2 */
u16 ICAP1value = TIM_GetCapture1(TIM2);
```

19.2.64 函数TIM_GetCapture2

Table 556. 描述了函数TIM_GetCapture2

Table 556. 函数 TIM_GetCapture2

函数名	TIM_GetCapture2
函数原形	u16 TIM_GetCapture2(TIM_TypeDef* TIMx)
功能描述	获得 TIMx 输入捕获 2 的值
输入参数	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输出参数	无
返回值	输入捕获 2 的值
先决条件	无
被调用函数	无

例：

```
/* Gets the Input Capture 2 value of the TIM2 */
u16 ICAP2value = TIM_GetCapture2(TIM2);
```

19.2.65 函数TIM_GetCapture3

Table 557. 描述了函数TIM_GetCaptur3

Table 557. 函数 TIM_GetCapture3

函数名	TIM_GetCapture3
函数原形	u16 TIM_GetCapture3(TIM_TypeDef* TIMx)
功能描述	获得 TIMx 输入捕获 3 的值
输入参数	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输出参数	无
返回值	输入捕获 3 的值
先决条件	无
被调用函数	无

例：

```
/* Gets the Input Capture 3 value of the TIM2 */
u16 ICAP3value = TIM_GetCapture3(TIM2);
```

19.2.66 函数TIM_GetCapture4

Table 558. 描述了函数TIM_GetCapture4

Table 558. 函数 TIM_GetCapture4

函数名	TIM_GetCapture4
函数原形	u16 TIM_GetCapture4(TIM_TypeDef* TIMx)
功能描述	获得 TIMx 输入捕获 4 的值
输入参数	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输出参数	无
返回值	输入捕获 4 的值
先决条件	无
被调用函数	无

例：

```
/* Gets the Input Capture 4 value of the TIM2 */
u16 ICAP4value = TIM_GetCapture4(TIM2);
```

19.2.67 函数TIM_GetCounter

Table 559. 描述了函数TIM_GetCounter

Table 559. 函数 TIM_GetCounter

函数名	TIM_GetCounter
函数原形	u16 TIM_GetCounter(TIM_TypeDef* TIMx)
功能描述	获得 TIMx 计数器的值
输入参数	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输出参数	无
返回值	计数器的值
先决条件	无
被调用函数	无

例：

```
/* Gets TIM2 counter value */
u16 TIMCounter = TIM_GetCounter(TIM2);
```

19.2.68 函数TIM_GetPrescaler

Table 560. 描述了函数TIM_GetPrescaler

Table 560. 函数 TIM_GetPrescaler

函数名	TIM_GetPrescaler
函数原形	u16 TIM_GetPrescaler (TIM_TypeDef* TIMx)
功能描述	获得 TIMx 预分频值
输入参数	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输出参数	无
返回值	预分频的值
先决条件	无
被调用函数	无

例：

```
/* Gets TIM2 prescaler value */
u16 TIMPrescaler = TIM_GetPrescaler(TIM2);
```

19.2.69 函数TIM_GetFlagStatus

Table 561. 描述了函数TIM_ GetFlagStatus

Table 561. 函数 TIM_ GetFlagStatus

函数名	TIM_ GetFlagStatus
函数原形	FlagStatus TIM_GetFlagStatus(TIM_TypeDef* TIMx, u16 TIM_FLAG)
功能描述	检查指定的 TIM 标志位设置与否
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_FLAG：待检查的 TIM 标志位 参阅 Section：TIM_FLAG 查阅更多该参数允许取值范围
输出参数	无
返回值	TIM_FLAG 的新状态（SET 或者 RESET）
先决条件	无
被调用函数	无

TIM_FLAG

Table 562. 给出了所有可以被函数TIM_ GetFlagStatus检查的标志位列表

Table 562. TIM_FLAG 值

TIM_FLAG	描述
TIM_FLAG_Update	TIM 更新标志位
TIM_FLAG_CC1	TIM 捕获/比较 1 标志位
TIM_FLAG_CC2	TIM 捕获/比较 2 标志位
TIM_FLAG_CC3	TIM 捕获/比较 3 标志位
TIM_FLAG_CC4	TIM 捕获/比较 4 标志位
TIM_FLAG_Trigger	TIM 触发标志位
TIM_FLAG_CC1OF	TIM 捕获/比较 1 溢出标志位
TIM_FLAG_CC2OF	TIM 捕获/比较 2 溢出标志位
TIM_FLAG_CC3OF	TIM 捕获/比较 3 溢出标志位
TIM_FLAG_CC4OF	TIM 捕获/比较 4 溢出标志位

例：

```
/* Check if the TIM2 Capture Compare 1 flag is set or reset */
if(TIM_GetFlagStatus(TIM2, TIM_FLAG_CC1) == SET)
{
```

```
}
```

19.2.70 函数TIM_ClearFlag

Table 563. 描述了函数TIM_ ClearFlag

Table 563. 函数 TIM_ ClearFlag

函数名	TIM_ ClearFlag
函数原形	void TIM_ClearFlag(TIM_TypeDef* TIMx, u32 TIM_FLAG)
功能描述	清除 TIMx 的待处理标志位
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_FLAG：待清除的 TIM 标志位 参阅 Section：TIM_FLAG 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear the TIM2 Capture Compare 1 flag */
TIM_ClearFlag(TIM2, TIM_FLAG_CC1);
```

19.2.71 函数TIM_GetITStatus

Table 564. 描述了函数TIM_ GetITStatus

Table 564. 函数 TIM_ GetITStatus

函数名	TIM_ GetITStatus
函数原形	ITStatus TIM_GetITStatus(TIM_TypeDef* TIMx, u16 TIM_IT)
功能描述	检查指定的 TIM 中断发生与否
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_IT：待检查的 TIM 中断源 参阅 Section：TIM_IT 查阅更多该参数允许取值范围
输出参数	无
返回值	TIM_IT 的新状态
先决条件	无
被调用函数	无

例：

```
/* Check if the TIM2 Capture Compare 1 interrupt has occured or not
*/
if(TIM_GetITStatus(TIM2, TIM_IT_CC1) == SET)
{
}
```

19.2.72 函数TIM_ClearITPendingBit

Table 565. 描述了函数TIM_ ClearITPendingBit

Table 565. 函数 TIM_ ClearITPendingBit

函数名	TIM_ ClearITPendingBit
函数原形	void TIM_ClearITPendingBit(TIM_TypeDef* TIMx, u16 TIM_IT)
功能描述	清除 TIMx 的中断待处理位
输入参数 1	TIMx：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM_IT：待检查的 TIM 中断待处理位 参阅 Section：TIM_IT 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear the TIM2 Capture Compare 1 interrupt pending bit */
TIM_ClearITPendingBit(TIM2, TIM_IT_CC1);
```

20 高级控制定时器（TIM1）

高级控制定时器(TIM1)由一个 16 位的自动装载计数器组成，它由一个可编程预分频器驱动。

它适合多种用途，包含测量输入信号的脉冲宽度(输入捕获)，或者产生输出波形(输出比较，PWM，嵌入死区时间的互补 PWM 等)。

使用定时器预分频器和 RCC 时钟控制预分频器，可以实现脉冲宽度和波形周期从几个微秒到几个毫秒的调节。

Section 20.1 TIM1 寄存器结构描述了固件函数库所使用的数据结构，Section 20.2 固件库函数介绍了函数库里的所有函数。

20.1 TIM1 寄存器结构

TIM1 寄存器结构，***TIM1_TypeDeff***，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;
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;
```

Table 566.例举了TIM1所有寄存器

Table 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/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/比较使能寄存器
CNT	计数器寄存器
PSC	预分频寄存器
APR	自动重装载寄存器
RCR	周期计数寄存器
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	刹车和死区寄存器
DCR	DMA 控制寄存器
DMAR	连续模式的 DMA 地址寄存器

TIM1 外设声明于文件***“stm32f10x_map.h”***：

```
...
#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
```

使用Debug模式时，初始化指针***TIM1***于文件***“stm32f10x_lib.c”***：

```
...
#ifdef _TIM1
TIM1 = (TIM1_TypeDef *) TIM1_BASE;
#endif /*_TIM1 *//
...
```

为了访问 TIM1 寄存器，,***_TIM1*** 必须在文件***“stm32f10x_conf.h”***中定义如下：

```
...
#define _TIM1
...
```

20.2 TIM1 库函数

Table 567. 例举了TIM1的库函数

Table 567. TIM1 库函数

函数名	描述
TIM1_DeInit	将外设 TIM1 寄存器重设为缺省值
TIM1_TIM1BaseInit	根据 TIM1_TIM1BaseInitStruct 中指定的参数初始化 TIM1 的时间基数单位
TIM1_OC1Init	根据 TIM1_OCInitStruct 中指定的参数初始化 TIM1 通道 1
TIM1_OC2Init	根据 TIM1_OCInitStruct 中指定的参数初始化 TIM1 通道 2
TIM1_OC3Init	根据 TIM1_OCInitStruct 中指定的参数初始化 TIM1 通道 3
TIM1_OC4Init	根据 TIM1_OCInitStruct 中指定的参数初始化 TIM1 通道 4
TIM1_BDTRConfig	设置刹车特性，死区时间，锁电平，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外设 TIM1
TIM1_PWMIConfig	根据 TIM1_ICInitStruct 中指定的参数设置外设 TIM1 工作在 PWM 输入模式
TIM1_TIM1BaseStructInit	把 TIM1_TIM1BaseInitStruct 中的每一个参数按缺省值填入
TIM1_OCStructInit	把 TIM1_OCInitStruct 中的每一个参数按缺省值填入
TIM1_ICStructInit	把 TIM1_ICInitStruct 中的每一个参数按缺省值填入
TIM1_BDTRStructInit	把 TIM1_BDTRInitStruct 中的每一个参数按缺省值填入
TIM1_Cmd	使能或者失能 TIM1 外设
TIM1_CtrlPWMOutputs	使能或者失能 TIM1 外设的主输出
TIM1_ITConfig	使能或者失能指定的 TIM1 中断
TIM1_DMAConfig	设置 TIM1 的 DMA 接口
TIM1_DMACmd	使能或者失能指定的 TIM1 的 DMA 请求
TIM1_InternalClockConfig	设置 DMA 内部时钟
TIM1_ETRClockMode1Config	配置 TIM1 外部时钟模式 1
TIM1_ETRClockMode2Config	配置 TIM1 外部时钟模式 2
TIM1_ETRConfig	配置 TIM1 外部触发
TIM1_ITRxExternalClockConfig	设置 TIM1 内部触发为外部时钟模式
TIM1_TIxExternalClockConfig	设置 TIM1 触发为外部时钟
TIM1_SelectInputTrigger	选择 TIM1 输入触发源
TIM1_UpdateDisableConfig	使能或者失能 TIM1 更新事件
TIM1_UpdateRequestConfig	设置 TIM1 更新请求源
TIM1_SelectHallSensor	使能或者失能 TIM1 霍尔传感器接口
TIM1_SelectOnePulseMode	设置 TIM1 单脉冲模式
TIM1_SelectOutputTrigger	选择 TIM1 触发输出模式
TIM1_SelectSlaveMode	选择 TIM1 从模式
TIM1_SelectMasterSlaveMode	设置或者重置 TIM1 主/从模式
TIM1_EncoderInterfaceConfig	设置 TIM1 编码界面
TIM1_PrescalerConfig	设置 TIM1 预分频
TIM1_CounterModeConfig	设置 TIM1 计数器模式
TIM1_ForcedOC1Config	置 TIM1 输出 1 为活动或者非活动电平
TIM1_ForcedOC2Config	置 TIM1 输出 2 为活动或者非活动电平
TIM1_ForcedOC3Config	置 TIM1 输出 3 为活动或者非活动电平
TIM1_ForcedOC4Config	置 TIM1 输出 4 为活动或者非活动电平

TIM1_ARRPreloadConfig	使能或者失能 TIM1 在 ARR 上的预装载寄存器
TIM1_SelectCOM	选择 TIM1 外设的通讯事件
TIM1_SelectCCDMA	选择 TIM1 外设的捕获比较 DMA 源
TIM1_CCPreloadControl	设置或者重置 TIM1 捕获比较控制位
TIM1_OC1PreloadConfig	使能或者失能 TIM1 在 CCR1 上的预装载寄存器
TIM1_OC2PreloadConfig	使能或者失能 TIM1 在 CCR2 上的预装载寄存器
TIM1_OC3PreloadConfig	使能或者失能 TIM1 在 CCR3 上的预装载寄存器
TIM1_OC4PreloadConfig	使能或者失能 TIM1 在 CCR4 上的预装载寄存器
TIM1_OC1FastConfig	设置 TIM1 捕获比较 1 快速特征
TIM1_OC2FastConfig	设置 TIM1 捕获比较 2 快速特征
TIM1_OC3FastConfig	设置 TIM1 捕获比较 3 快速特征
TIM1_OC4FastConfig	设置 TIM1 捕获比较 4 快速特征
TIM1_ClearOC1Ref	在一个外部事件时清除或者保持 OCREF1 信号
TIM1_ClearOC2Ref	在一个外部事件时清除或者保持 OCREF2 信号
TIM1_ClearOC3Ref	在一个外部事件时清除或者保持 OCREF3 信号
TIM1_ClearOC4Ref	在一个外部事件时清除或者保持 OCREF4 信号
TIM1_GenerateEvent	设置 TIM1 事件由软件产生
TIM1_OC1PolarityConfig	设置 TIM1 通道 1N 极性
TIM1_OC1NPolarityConfig	设置 TIM1 通道 1N 极性
TIM1_OC2PolarityConfig	设置 TIM1 通道 2 极性
TIM1_OC2NPolarityConfig	设置 TIM1 通道 2N 极性
TIM1_OC3PolarityConfig	设置 TIM1 通道 3 极性
TIM1_OC3NPolarityConfig	设置 TIM1 通道 3N 极性
TIM1_OC4PolarityConfig	设置 TIM1 通道 4 极性
TIM1_SetCounter	设置 TIM1 计数器寄存器值
TIM1_CCxCmd	使能或者失能 TIM1 捕获比较通道 x
TIM1_CCxNCmd	使能或者失能 TIM1 捕获比较通道 xN
TIM1_SelectOCxM	选择 TIM1 输出比较模式。 本函数在改变输出比较模式前失能选中的通道。用户必须使用函数 TIM1_CCxCmd 和 TIM1_CCxNCmd 来使能这个通道。
TIM1_SetAutoreload	设置 TIM1 自动重装载寄存器值
TIM1_SetCompare1	设置 TIM1 捕获比较 1 寄存器值
TIM1_SetCompare2	设置 TIM1 捕获比较 2 寄存器值
TIM1_SetCompare3	设置 TIM1 捕获比较 3 寄存器值
TIM1_SetCompare4	设置 TIM1 捕获比较 4 寄存器值
TIM1_SetIC1Prescaler	设置 TIM1 输入捕获 1 预分频
TIM1_SetIC2Prescaler	设置 TIM1 输入捕获 2 预分频
TIM1_SetIC3Prescaler	设置 TIM1 输入捕获 3 预分频
TIM1_SetIC4Prescaler	设置 TIM1 输入捕获 4 预分频
TIM1_SetClockDivision	设置 TIM1 的时钟分割值
TIM1_GetCapture1	获得 TIM1 输入捕获 1 的值
TIM1_GetCapture2	获得 TIM1 输入捕获 2 的值
TIM1_GetCapture3	获得 TIM1 输入捕获 3 的值
TIM1_GetCapture4	获得 TIM1 输入捕获 4 的值
TIM1_GetCounter	获得 TIM1 计数器的值
TIM1_GetPrescaler	获得 TIM1 预分频值
TIM1_GetFlagStatus	检查指定的 TIM1 标志位设置与否
TIM1_ClearFlag	清除 TIM1 的待处理标志位
TIM1_GetITStatus	检查指定的 TIM1 中断发生与否
TIM1_ClearITPendingBit	清除 TIM1 的中断待处理位

20.2.1 函数TIM1_DeInit

Table 568. 描述了函数 TIM1_DeInit

Table 568. 函数 TIM1_DeInit

函数名	TIM1_DeInit
函数原形	void TIM1_DeInit(void)
功能描述	将外设 TIM1 寄存器重设为缺省值
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	RCC_APB2PeriphClockCmd().

例：

```
/* Resets the TIM1 */
TIM1_DeInit();
```

20.2.2 函数TIM1_TIM1BaseInit

Table 569. 描述了函数 TIM1_TIM1BaseInit

Table 569. 函数 TIM1_TIM1BaseInit

函数名	TIM1_TIM1BaseInit
函数原形	void TIM1_TIM1BaseInit(TIM1_TIM1BaseInitTypeDef* TIM1_BaseInitStruct)
功能描述	根据 TIM1_TIM1BaseInitStruct 中指定的参数初始化 TIM1 的时间基数单位
输入参数	TIM1TIM1Base_InitStruct：指向结构 TIM1_TIM1BaseInitTypeDef 的指针，包含了 TIM1 时间基数单位的配置信息 参阅 Section：TIM1_TIM1BaseInitTypeDef 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM1_TIM1BaseInitTypeDef structure

TIM1_TIM1BaseInitTypeDef 定义于文件***"stm32f10x_TIM1.h"***：

```
typedef struct
{
u16 TIM1_Period;
u16 TIM1_Prescaler;
u8 TIM1_ClockDivision;
u16 TIM1_CounterMode;
u8 TIM1_RepetitionCounter;
} TIM1_TIM1BaseInitTypeDef;
```

TIM1_Period

TIM1_Period 设置了在下一个更新事件装入活动的自动重装载寄存器周期的值。它的取值必须在 0x0000 和 0xFFFF 之间。

TIM1_Prescaler

TIM1_Prescaler 设置了用来作为 TIM1 时钟频率除数的预分频值。它的取值必须在 0x0000 和 0xFFFF 之间。

TIM1_ClockDivision

TIM1_ClockDivision 设置了时钟分割。该参数取值见下表。

Table 570. TIM1_ClockDivision 值

TIM1_ClockDivision	描述
TIM1_CKD_DIV1	TDTS = Tck_TIM1
TIM1_CKD_DIV2	TDTS = 2Tck_TIM1
TIM1_CKD_DIV4	TDTS = 4Tck_TIM1

TIM1_CounterMode

TIM1_CounterMode 选择了计数器模式。该参数取值见下表。

Table 571. TIM1_CounterMode 值

TIM1_CounterMode	描述
TIM1_CounterMode_Up	TIM1 向上计数模式
TIM1_CounterMode_Down	TIM1 向下计数模式
TIM1_CounterMode_CenterAligned1	TIM1 中央对齐模式 1 计数模式
TIM1_CounterMode_CenterAligned2	TIM1 中央对齐模式 2 计数模式
TIM1_CounterMode_CenterAligned3	TIM1 中央对齐模式 3 计数模式

TIM1_RepetitionCounter

TIM1_RepetitionCounter设置了周期计数器值。RCR向下计数器每次计数至0，会产生一个更新事件且计数器重新由RCR值（N）开始计数。

这意味着在PWM模式（N+1）对应着：

- 边沿对齐模式下PWM周期数
- 中央对齐模式下PWM半周期数

它的取值必须在0x00和0xFF之间。

20.2.3 函数TIM1_OC1Init

Table 572. 描述了函数 TIM1_OC1Init

Table 572. 函数 TIM1_OC1Init

函数名	TIM1_OC1Init
函数原形	void TIM1_OC1Init(TIM1_OCInitTypeDef* TIM1_OCInitStruct)
功能描述	根据 TIM1_OCInitStruct 中指定的参数初始化 TIM1 通道 1
输入参数	TIM1_OCInitStruct：指向结构 TIM1_OCInitTypeDef 的指针，包含了 TIM1 时间基数单位的配置信息。参阅 Section：TIM1_OCInitTypeDef 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM1_OCInitTypeDef structure

TIM1_OCInitTypeDef 定义于文件“stm32f10x_TIM1.h”：

```
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 选择定时器模式。该参数取值见下表。

Table 573. TIM1_OCMode 定义

TIM1_OCMode	描述
TIM1_OCMode_TIM1ing	TIM1 输出比较时间模式
TIM1_OCMode_Active	TIM1 输出比较主动模式
TIM1_OCMode_Inactive	TIM1 输出比较非主动模式
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选择输出比较状态。该参数取值见下表。

Table 574. TIM1_OutputState 值

TIM1_OutputState	描述
TIM1_OutputState_Disable	失能输出比较状态
TIM1_OutputState_Enable	使能输出比较状态

TIM1_OutputNState

TIM1_OutputNState选择互补输出比较状态。该参数取值见下表。

Table 575. TIM1_OutputNState 值

TIM1_OutputNState	描述
TIM1_OutputState_Disable	失能输出比较 N 状态
TIM1_OutputState_Enable	使能输出比较 N 状态

TIM1_Pulse

TIM1_Pulse 设置了待装入捕获比较寄存器的脉冲值。它的取值必须在 0x0000 和 0xFFFF 之间。

TIM1_OCPolarity

TIM1_OCPolarity输出极性。该参数取值见下表。

Table 576. TIM1_OCPolarity 值

TIM1_OCPolarity	描述
TIM1_OCPolarity_High	TIM1 输出比较极性高
TIM1_OCPolarity_Low	TIM1 输出比较极性低

TIM1_OCNPolarity

TIM1_OCNPolarity互补输出极性。该参数取值见下表。

Table 577. TIM1_OCNPolarity 值

TIM1_OCNPolarity	描述
TIM1_OCNPolarity_High	TIM1 输出比较 N 极性高
TIM1_OCNPolarity_Low	TIM1 输出比较 N 极性低

TIM1_OCIdleState

TIM1_OCIdleState选择空闲状态下的非工作状态。该参数取值见下表。

Table 578. TIM1_OCIdleState 值

TIM1_OCIdleState	描述
TIM1_OCIdleState_Set	当 MOE=0 设置 TIM1 输出比较空闲状态
TIM1_OCIdleState_Reset	当 MOE=0 重置 TIM1 输出比较空闲状态

TIM1_OCNIdleState

TIM1_OCNIdleState选择空闲状态下的非工作状态。该参数取值见下表。

Table 579. TIM1_OCNIdleState 值

TIM1_OCNIdleState	描述
TIM1_OCNIdleState_Set	当 MOE=0 设置 TIM1 输出比较 N 空闲状态
TIM1_OCNIdleState_Reset	当 MOE=0 重置 TIM1 输出比较 N 空闲状态

例:

```
/* Configures the TIM1 Channel1 in PWM Mode */
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);
```

20.2.4 函数TIM1_OC2Init

Table 580. 描述了函数 TIM1_OC2Init

Table 580. 函数 TIM1_OC2Init

函数名	TIM1_OC2Init
函数原形	void TIM1_OC2Init(TIM1_OCInitTypeDef* TIM1_OCInitStruct)
功能描述	根据 TIM1_OCInitStruct 中指定的参数初始化 TIM1 通道 2
输入参数	TIM1_OCInitStruct：指向结构 TIM1_OCInitTypeDef 的指针，包含了 TIM1 时间基数单位的配置信息 参阅 Section：TIM1_OCInitTypeDef 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Configures the TIM1 Channel2 in PWM Mode */
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);
```

20.2.5 函数TIM1_OC3Init

Table 581. 描述了函数 TIM1_OC3Init

Table 581. 函数 TIM1_OC3Init

函数名	TIM1_OC3Init
函数原形	void TIM1_OC3Init(TIM1_OCInitTypeDef* TIM1_OCInitStruct)
功能描述	根据 TIM1_OCInitStruct 中指定的参数初始化 TIM1 通道 3
输入参数	TIM1_OCInitStruct：指向结构 TIM1_OCInitTypeDef 的指针，包含了 TIM1 时间基数单位的配置信息 参阅 Section：TIM1_OCInitTypeDef 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Configures the TIM1 Channel3 in PWM Mode */
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);
```

20.2.6 函数TIM1_OC4Init

Table 582. 描述了函数 TIM1_OC4Init

Table 582. 函数 TIM1_OC4Init

函数名	TIM1_OC4Init
函数原形	void TIM1_OC4Init(TIM1_OCInitTypeDef* TIM1_OCInitStruct)
功能描述	根据 TIM1_OCInitStruct 中指定的参数初始化 TIM1 通道 4
输入参数	TIM1_OCInitStruct：指向结构 TIM1_OCInitTypeDef 的指针，包含了 TIM1 时间基数单位的配置信息 参阅 Section：TIM1_OCInitTypeDef 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Configures the TIM1 Channel4 in PWM Mode */
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);
```

20.2.7 函数TIM1_BDTRConfig

Table 583. 描述了函数 TIM1_BDTRConfig

Table 583. 函数 TIM1_BDTRConfig

函数名	TIM1_BDTRConfig
函数原形	void TIM1_BDTRConfig(TIM1_BDTRInitTypeDef *TIM1_BDTRInitStruct)
功能描述	设置刹车特性，死区时间，锁电平，OSSI，OSSR 状态和 AOE（自动输出使能）
输入参数	TIM1_BDTRInitStruct：指向结构 TIM1_BDTRInitTypeDef 的指针，包含了 TIM1 的 BDTR 寄存器的配置信息 参阅 Section：TIM1_BDTRInitTypeDef 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM1_BDTRInitStruct structure

TIM1_BDTRInitStruct structure 定义于文件***"stm32f10x_TIM1.h"***：

```
typedef struct
{
u16 TIM1_OSSRState;
u16 TIM1_OSSIState;
u16 TIM1_LOCKLevel;
u16 TIM1_DeadTIM1;
u16 TIM1_Break;
u16 TIM1_BreakPolarity;
u16 TIM1_AutomaticOutput;
} TIM1_BDTRInitTypeDef;
```

TIM1_OSSRState

TIM1_OSSRState 设置在运行模式下非工作状态选项。该参数取值见下表。

Table 584. TIM1_OSSRState 值

TIM1_OSSRState	描述
TIM1_OSSRState_Enable	使能 TIM1 OSSR 状态
TIM1_OSSRState_Disable	失能 TIM1 OSSR 状态

TIM1_OSSIState

TIM1_OSSIState 设置在运行模式下非工作状态选项。该参数取值见下表。

Table 585. TIM1_OSSIState 值

TIM1_OSSIState	描述
TIM1_OSSIState_Enable	使能 TIM1 OSSI 状态
TIM1_OSSIState_Disable	失能 TIM1 OSSI 状态

TIM1_LOCKLevel

TIM1_LOCKLevel 设置了锁电平参数。该参数取值见下表。

Table 586. TIM1_LOCKLevel 值

TIM1_LOCKLevel	描述
TIM1_LOCKLevel_OFF	不锁任何位
TIM1_LOCKLevel_1	使用锁电平 1
TIM1_LOCKLevel_2	使用锁电平 2
TIM1_LOCKLevel_3	使用锁电平 3

TIM1_DeadTIM1

TIM1_DeadTIM1 指定了输出打开和关闭状态之间的延时。

TIM1_Break

TIM1_Break 使能或者失能 TIM1 刹车输入。该参数取值见下表。

Table 587. TIM1_OSSIState 值

TIM1_Break	描述
TIM1_Break_Enable	使能 TIM1 刹车输入
TIM1_Break_Disable	失能 TIM1 刹车输入

TIM1_BreakPolarity

TIM1_BreakPolarity 设置 TIM1 刹车输入管脚极性。该参数取值见下表。

Table 588. TIM1_BreakPolarity 值

TIM1_BreakPolarity	描述
TIM1_BreakPolarity_Low	TIM1 刹车输入管脚极性低
TIM1_BreakPolarity_High	TIM1 刹车输入管脚极性高

TIM1_AutomaticOutput

TIM1_AutomaticOutput 使能或者失能自动输出功能该参数取值见下表。

Table 589. TIM1_AutomaticOutput 值

TIM1_AutomaticOutput	描述
TIM1_AutomaticOutput_Enable	自动输出功能使能
TIM1_AutomaticOutput_Disable	自动输出功能失能

例：

```
/* OSSR, OSSI, Automatic Output enable, Break, dead TIM1 and Lock
Level configuration*/
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_DeadTIM1 = 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);
```

20.2.8 函数TIM1_ICInit

Table 590. 描述了函数 TIM1_ICInit

Table 590. 函数 TIM1_ICInit

函数名	TIM1_OCInit
函数原形	void TIM1_ICInit(TIM1_ICInitTypeDef* TIM1_ICInitStruct)
功能描述	根据 TIM1_ICInitStruct 中指定的参数初始化外设 TIM1
输入参数	TIM1_ICInitStruct：指向结构 TIM1_ICInitTypeDef 的指针，包含了 TIM1 的配置信息。参阅 Section：TIM1_ICInitTypeDef 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM1_ICInitTypeDef structure

TIM1_ICInitTypeDef 定义于文件***"stm32f10x_TIM1.h"***：

```
typedef struct
{
u16 TIM1_Channel;
u16 TIM1_ICPolarity;
u16 TIM1_ICSelection;
u16 TIM1_ICPrescaler;
u16 TIM1_ICFilter;
} TIM1_ICInitTypeDef;
```

TIM1_Channel

TIM1_Channel 选择通道。该参数取值见下表。

Table 591. TIM1_Channel 值

TIM1_Channel	描述
TIM1_Channel_1	使用 TIM1 通道 1
TIM1_Channel_2	使用 TIM1 通道 2
TIM1_Channel_3	使用 TIM1 通道 3
TIM1_Channel_4	使用 TIM1 通道 4

TIM1_ICPolarity

TIM1_ICPolarity输入活动沿。该参数取值见下表。

Table 592. TIM1_Channel 值

TIM1_OCPolarity	描述
TIM1_ICPolarity_Rising	TIM1 输入捕获上升沿
TIM1_ICPolarity_Falling	TIM1 输入捕获下降沿

TIM1_ICSelection

TIM1_ICSelection选择输入。该参数取值见下表。

Table 593. TIM1_ICSelection 值

TIM1_ICSelection	描述
TIM1_ICSelection_DirectTI	TIM1 输入 2，3 或 4 选择对应地与 IC1 或 IC2 或 IC3 或 IC4 相连
TIM1_ICSelection_IndirectTI	TIM1 输入 2，3 或 4 选择对应地与 IC2 或 IC1 或 IC4 或 IC3 相连
TIM1_ICSelection_TRC	TIM1 输入 2，3 或 4 选择与 TRC 相连

TIM1_ICPrescaler

TIM1_ICPrescaler设置输入捕获预分频器。该参数取值见下表。

Table 594. TIM1_ICPrescaler 值

TIM1_ICPrescaler	描述
TIM1_ICPSC_DIV1	TIM1 捕获在捕获输入上每探测到一个边沿执行一次
TIM1_ICPSC_DIV2	TIM1 捕获每 2 个事件执行一次
TIM1_ICPSC_DIV3	TIM1 捕获每 3 个事件执行一次
TIM1_ICPSC_DIV4	TIM1 捕获每 4 个事件执行一次

TIM1_ICFilter

TIM1_ICFilter选择输入比较滤波器。该参数取值在0x0和0xF之间。

例：

```
/* TIM1 Input Capture Channel 1 mode Configuration */
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);
```

20.2.9 函数TIM1_PWMIConfig

Table 595. 描述了函数 TIM1_PWMIConfig

Table 595. 函数 TIM1_PWMIConfig

函数名	TIM1_PWMIConfig
函数原形	TIM1_PWMIConfig(TIM1_ICInitTypeDef* TIM1_ICInitStruct)
功能描述	根据 TIM1_ICInitStruct 中指定的参数设置外设 TIM1 工作在 PWM 输入模式
输入参数	TIM1_ICInitStruct：指向结构 TIM1_ICInitTypeDef 的指针，包含了 TIM1 的配置信息。 参阅 Section：TIM1_ICInitTypeDef 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* TIM1 PWM Input Channel 1 mode Configuration */
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);
```

20.2.10 函数TIM1_TimeBaseStructInit

Table 596. 描述了函数TIM1_TimeBaseStructInit

Table 596. 函数 TIM1_TimeBaseStructInit

函数名	TIM1_TimeBaseStructInit
函数原形	void TIM1_TimeBaseStructInit(TIM1_TimeBaseInitTypeDef* TIM1_TimeBaseInitStruct)
功能描述	把 TIM1_TimeBaseInitStruct 中的每一个参数按缺省值填入
输入参数	TIM1_TimeBaseInitStruct：指向结构 TIM1_TimeBaseInitTypeDef 的指针，待初始化
输出参数	无
返回值	无
先决条件	无
被调用函数	无

Table 597. 给出了TIM1_TimeBaseInitStruct各个成员的缺省值

Table 597. TIM1_TimeBaseInitStruct 缺省值

成员	缺省值
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：

```
/* The following example illustrates how to initialize a
TIM1_BaseInitTypeDef structure */
TIM1_TimeBaseInitTypeDef TIM1_TimeBaseInitStructure;
TIM1_TimeBaseStructInit(& TIM1_TimeBaseInitStructure);
```

20.2.11 函数TIM1_OCStructInit

Table 598. 描述了函数TIM1_OCStructInit

Table 598. 函数 TIM1_TimeBaseStructInit

函数名	TIM1_TimeBaseStructInit
函数原形	void TIM1_OCStructInit(TIM1_OCInitTypeDef* TIM1_OCInitStruct)
功能描述	把 TIM1_OCInitStruct 中的每一个参数按缺省值填入
输入参数	TIM1_OCInitStruct：指向结构 TIM1_OCInitTypeDef 的指针，待初始化
输出参数	无
返回值	无
先决条件	无
被调用函数	无

Table 599. 给出了TIM1_OCInitStruct各个成员的缺省值

Table 599. TIM1_OCInitStruct 缺省值

成员	缺省值
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_OCNPolarity_High
TIM1_OCIdleState	TIM1_OCIdleState_Reset
TIM1_OCNIdleState	TIM1_OCNIdleState_Reset

例：

```
/* The following example illustrates how to initialize a
TIM1_OCInitTypeDef structure */
TIM1_OCInitTypeDef TIM1_OCInitStructure;
TIM1_OCStructInit(& TIM1_OCInitStructure);
```

20.2.12 函数TIM1_ICStructInit

Table 600. 描述了函数TIM1_ICStructInit

Table 600. 函数 TIM1_ICStructInit

函数名	TIM1_ICStructInit
函数原形	void TIM1_ICStructInit(TIM1_ICInitTypeDef* TIM1_ICInitStruct)
功能描述	把 TIM1_ICInitStruct 中的每一个参数按缺省值填入
输入参数	TIM1_ICInitStruct：指向结构 TIM1_ICInitTypeDef 的指针，待初始化
输出参数	无
返回值	无
先决条件	无
被调用函数	无

Table 601. 给出了TIM1_ICInitStruct各个成员的缺省值

Table 601. TIM1_ICInitStruct 缺省值

成员	缺省值
TIM1_ICMode	TIM1_ICMode_ICAP
TIM1_Channel	TIM1_Channel_1
TIM1_ICPolarity	TIM1_ICPolarity_Rising
TIM1_ICPrescaler	TIM1_ICPSC_DIV1
TIM1_ICFilter	TIM1_ICFilter_Mask

例：

```
/* The following example illustrates how to initialize a
TIM1_ICInitTypeDef structure */
TIM1_ICInitTypeDef TIM1_ICInitStructure;
TIM1_ICStructInit(& TIM1_ICInitStructure);
```

20.2.13 函数TIM1_BDTRStructInit

Table 602. 描述了函数 TIM1_BDTRStructInit

Table 602. 函数 TIM1_BDTRStructInit

函数名	TIM1_TimeBaseStructInit
函数原形	void TIM1_TimeBaseStructInit(TIM1_TimeBaseInitTypeDef* TIM1_TimeBaseInitStruct)
功能描述	把 TIM1_BDTRInitStruct 中的每一个参数按缺省值填入
输入参数	TIM1_TimeBaseInitStruct：指向结构 TIM1_TimeBaseInitTypeDef 的指针，待初始化
输出参数	无
返回值	无
先决条件	无
被调用函数	无

Table 603. 给出了TIM1_TimeBaseInitStruct各个成员的缺省值

Table 603. TIM1_TimeBaseInitStruct 缺省值

成员	缺省值
TIM1_OSSRState	TIM1_OSSRState_Disable
TIM1_OSSIState	TIM1_OSSIState_Disable
TIM1_LOCKLevel	TIM1_LOCKLevel_OFF
TIM1_DeadTime	TIM1_DeadTime_Reset_Mask
TIM1_Break	TIM1_Break_Disable
TIM1_BreakPolarity	TIM1_BreakPolarity_Low
TIM1_AutomaticOutput	TIM1_AutomaticOutput_Disable

例：

```
/* The following example illustrates how to initialize a
TIM1_BDTRInitTypeDef structure */
TIM1_BDTRInitTypeDef TIM1_BDTRInitStructure;
TIM1_BDTRStructInit(& TIM1_BDTRInitStructure);
```

20.2.14 函数TIM1_Cmd

Table 604. 描述了函数TIM1_Cmd

Table 604. 函数 TIM1_Cmd

函数名	TIM1_Cmd
函数原形	void TIM1_Cmd(FunctionalState NewState)
功能描述	使能或者失能 TIM1 外设
输入参数	NewState: 外设 TIM1 的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enables the TIM1 counter */
TIM1_Cmd(ENABLE);
```

20.2.15 函数TIM1_CtrlPWMOutputs

Table 605. 描述了函数TIM1_CtrlPWMOutputs

Table 605. 函数 TIM1_CtrlPWMOutputs

函数名	TIM1_CtrlPWMOutputs
函数原形	void TIM1_CtrlPWMOutputs(FunctionalState Newstate)
功能描述	使能或者失能 TIM1 的主输出
输入参数	NewState: 外设 TIM1 主输出的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enables the TIM1 peripheral Main Outputs. */
TIM1_CtrlPWMOutputs(ENABLE);
```

20.2.16 函数TIM1_ITConfig

Table 606. 描述了函数TIM1_ITConfig

Table 606. 函数 TIM1_ITConfig

函数名	TIM1_ITConfig
函数原形	void TIM1_ITConfig(u16 TIM1_IT, FunctionalState NewState)
功能描述	使能或者失能指定的 TIM1 中断
输入参数 1	TIM1_IT：待使能或者失能的 TIM 中断源 参阅 Section：TIM1_IT 查阅更多该参数允许取值范围
输入参数 2	NewState：TIM1 中断的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM1_IT

输入参数 TIM1_IT 使能或者失能 TIM 的中断。可以取下表的一个或者多个取值的组合作为该参数的值。

Table 607. TIM1_IT 值

TIM1_IT	描述
TIM1_IT_Update	TIM1 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 刹车中断源

例：

```
/* Enables the TIM1 Capture Compare channel 1 Interrupt source */
TIM1_ITConfig(TIM1_IT_CC1, ENABLE );
```

20.2.17 函数TIM1_DMAConfig

Table 608. 描述了函数TIM1_DMAConfig

Table 608. 函数 TIM1_DMAConfig

函数名	TIM1_DMAConfig
函数原形	void TIM1_ITConfig(u16 TIM1_IT, FunctionalState NewState)
功能描述	设置 TIM1 的 DMA 接口
输入参数 1	TIM1_DMABase：DMA 传输起始地址 参阅 Section：TIM1_ DMABase 查阅更多该参数允许取值范围
输入参数 2	TIM1_DMABurstLength：DMA 连续传送长度 参阅 Section：TIM1_ DMABurstLength 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM1_DMABase

TIM1_DMABase 设置 DMA 传输起始地址。可以取下表的值。

Table 609. TIM1_DMABase 值

TIM1_DMABase	描述
TIM1_DMABase_CR1	TIM1 CR1 寄存器作为 DMA 传输起始
TIM1_DMABase_CR2	TIM1 CR2 寄存器作为 DMA 传输起始
TIM1_DMABase_SMCR	TIM1 SMCR 寄存器作为 DMA 传输起始
TIM1_DMABase_DIER	TIM1 DIER 寄存器作为 DMA 传输起始
TIM1_DMABase_SR	TIM1 SR 寄存器作为 DMA 传输起始
TIM1_DMABase_EGR	TIM1 EGR 寄存器作为 DMA 传输起始
TIM1_DMABase_CCMR1	TIM1 CCMR1 寄存器作为 DMA 传输起始
TIM1_DMABase_CCMR2	TIM1 CCMR2 寄存器作为 DMA 传输起始
TIM1_DMABase_CCER	TIM1 CCER 寄存器作为 DMA 传输起始
TIM1_DMABase_CNT	TIM1 CNT 寄存器作为 DMA 传输起始
TIM1_DMABase_PSC	TIM1 PSC 寄存器作为 DMA 传输起始
TIM1_DMABase_ARR	TIM1 APR 寄存器作为 DMA 传输起始
TIM1_DMABase_RCR	TIM1RCR 寄存器作为 DMA 传输起始
TIM1_DMABase_CCR1	TIM1 CCR1 寄存器作为 DMA 传输起始
TIM1_DMABase_CCR2	TIM1 CCR2 寄存器作为 DMA 传输起始
TIM1_DMABase_CCR3	TIM1 CCR3 寄存器作为 DMA 传输起始
TIM1_DMABase_CCR4	TIM1 CCR4 寄存器作为 DMA 传输起始
TIM1_DMABase_BDTR	TIM1 BDTR 寄存器作为 DMA 传输起始
TIM1_DMABase_DCR	TIM1 DCR 寄存器作为 DMA 传输起始

TIM1_DMABurstLength

TIM1_DMABurstLength 设置 DMA 连续传送长度。可以取下表的值。

Table 610. TIM1_DMABurstLength 值

TIM1_DMABurstLength	描述
TIM1_DMABurstLength_1Byte	TIM1 DMA 连续传送长度 1 字
TIM1_DMABurstLength_2Bytes	TIM1 DMA 连续传送长度 2 字
TIM1_DMABurstLength_3Bytes	TIM1 DMA 连续传送长度 3 字
TIM1_DMABurstLength_4Bytes	TIM1 DMA 连续传送长度 4 字

TIM1_DMABurstLength_5Bytes	TIM1 DMA 连续传送长度 5 字
TIM1_DMABurstLength_6Bytes	TIM1 DMA 连续传送长度 6 字
TIM1_DMABurstLength_7Bytes	TIM1 DMA 连续传送长度 7 字
TIM1_DMABurstLength_8Bytes	TIM1 DMA 连续传送长度 8 字
TIM1_DMABurstLength_9Bytes	TIM1 DMA 连续传送长度 9 字
TIM1_DMABurstLength_10Bytes	TIM1 DMA 连续传送长度 10 字
TIM1_DMABurstLength_11Bytes	TIM1 DMA 连续传送长度 11 字
TIM1_DMABurstLength_12Bytes	TIM1 DMA 连续传送长度 12 字
TIM1_DMABurstLength_13Bytes	TIM1 DMA 连续传送长度 13 字
TIM1_DMABurstLength_14Bytes	TIM1 DMA 连续传送长度 14 字
TIM1_DMABurstLength_15Bytes	TIM1 DMA 连续传送长度 15 字
TIM1_DMABurstLength_16Bytes	TIM1 DMA 连续传送长度 16 字
TIM1_DMABurstLength_17Bytes	TIM1 DMA 连续传送长度 17 字
TIM1_DMABurstLength_18Bytes	TIM1 DMA 连续传送长度 18 字

例：

```
/* Configures the TIM1 DMA Interface to transfer 1 byte and to use
the CCR1 as base address */
TIM1_DMAConfig(TIM1_DMABase_CCR1, TIM1_DMABurstLength_1Byte)
```

20.2.18 函数TIM1_DMACmd

Table 611. 描述了函数TIM1_DMACmd

Table 611. 函数 TIM1_DMACmd

函数名	TIM1_DMACmd
函数原形	void TIM1_DMACmd(u16 TIM1_DMASource, FunctionalState Newstate)
功能描述	使能或者失能指定的 TIM1 的 DMA 请求
输入参数 1	TIM1_DMASource：待使能或者失能的 TIM 中断源 参阅 Section：TIM1_DMASource 查阅更多该参数允许取值范围
输入参数 2	NewState：DMA 请求的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM1_DMASource

输入参数 TIM1_DMASource 使能或者失能 TIM 的中断。可以取下表的值。

Table 612. TIM1_DMASource 值

TIM1_DMASource	描述
TIM1_DMA_Update	TIM1 更新 DMA 源
TIM1_DMA_CC1	TIM1 捕获/比较 1DMA 源
TIM1_DMA_CC2	TIM1 捕获/比较 2DMA 源
TIM1_DMA_CC3	TIM1 捕获/比较 3DMA 源
TIM1_DMA_CC4	TIM1 捕获/比较 4DMA 源
TIM1_DMA_COM	TIM1 COM DMA 源
TIM1_DMA_Trigger	TIM1 触发 DMA 源

例：

```
/* TIM1 Capture Compare 1 DMA Request Configuration */
TIM1_DMACmd(TIM1_DMA_CC1, ENABLE);
```

20.2.19 函数TIM1_InternalClockConfig

Table 613. 描述了函数TIM1_InternalClockConfig

Table 613. 函数 TIM1_InternalClockConfig

函数名	TIM1_InternalClockConfig
函数原形	void TIM1_InternalClockConfig(void)
功能描述	设置 TIM1 内部时钟
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Selects the internal clock for TIM2 */
TIM1_InternalClockConfig(TIM2);
```

20.2.20 函数TIM1_ETRClockMode1Config

Table 614. 描述了函数TIM1_ETRClockMode1Config

Table 614. 函数 TIM1_ETRClockMode1Config

函数名	TIM1_ETRClockMode1Config
函数原形	void TIM1_ETRClockMode1Config(u16 TIM1_ExtTRGPrescaler, u16 TIM1_ExtTRGPolarity, u16 ExtTRGFilter)
功能描述	配置 TIM1 外部时钟模式 1
输入参数 1	TIM1_ExtTRGPrescaler：外部触发预分频 参阅 Section：TIM1_ExtTRGPrescaler 查阅更多该参数允许取值范围
输入参数 2	TIM1_ExtTRGPolarity：外部时钟极性 参阅 Section：TIM1_ExtTRGPolarity 查阅更多该参数允许取值范围
输入参数 3	ExtTRGFilter：外部触发滤波器。该参数取值在 0x0 和 0xF 之间。
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM1_ExtTRGPrescaler

TIM1_ExtTRGPrescaler设置TIM1外部触发预分频。见Table 493. 参阅该参数的取值。

Table 615. TIM1_ExtTRGPrescaler 值

TIM1_ExtTRGPrescaler	描述
TIM1_ExtTRGPSC_OFF	TIM1 ETRP 预分频 OFF
TIM1_ExtTRGPSC_DIV2	TIM1 ETRP 频率除以 2
TIM1_ExtTRGPSC_DIV4	TIM1 ETRP 频率除以 4
TIM1_ExtTRGPSC_DIV8	TIM1 ETRP 频率除以 8

TIM1_ExtTRGPolarity

TIM1_ExtTRGPolarity设置TIM1外部触发极性。见Table 494. 参阅该参数的取值。

Table 616. TIM1_ExtTRGPolarity 值

TIM1_ExtTRGPolarity	描述
TIM1_ExtTRGPolarity_Inverted	TIM1 外部触发极性翻转：低电平或下降沿有效
TIM1_ExtTRGPolarity_NonInverted	TIM1 外部触发极性非翻转：高电平或上升沿有效

例：

```
/* Selects the external clock Mode 1 for TIM1: the external clock is
connected to ETR input pin, the rising edge is the active edge, no
filter sampling is done (ExtTRGFilter = 0) and the prescaler is
fixed to TIM1_ExtTRGPSC_DIV2 */
TIM1_ExternalCLK1Config(TIM1_ExtTRGPSC_DIV2,
TIM1_ExtTRGPolarity_NonInverted, 0x0);
```

20.2.21 函数TIM1_ETRClockMode2Config

Table 617. 描述了函数TIM1_ETRClockMode2Config

Table 617. 函数 TIM1_ETRClockMode2Config

函数名	TIM1_ETRClockMode2Config
函数原形	void TIM1_ETRClockMode2Config(u16 TIM1_ExtTRGPrescaler, u16 TIM1_ExtTRGPolarity, u16 ExtTRGFilter)
功能描述	配置 TIM1 外部时钟模式 2
输入参数 1	TIM1_ExtTRGPrescaler：外部触发预分频 参阅 Section：TIM1_ExtTRGPrescaler 查阅更多该参数允许取值范围
输入参数 2	TIM1_ExtTRGPolarity：外部时钟极性 参阅 Section：TIM1_ExtTRGPolarity 查阅更多该参数允许取值范围
输入参数 3	ExtTRGFilter：外部触发滤波器。该参数取值在 0x0 和 0xF 之间。
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Selects the external clock Mode 2 for TIM1: the external clock is
connected to ETR input pin, the rising edge is the active edge, no
filter sampling is done (ExtTRGFilter = 0) and the prescaler is
fixed to TIM1_ExtTRGPSC_DIV2 */
TIM1_ExternalCLK2Config(TIM1_ExtTRGPSC_DIV2,
TIM1_ExtTRGPolarity_NonInverted, 0x0);
```

20.2.22 函数TIM1_ETRConfig

Table 618. 描述了函数TIM1_ETRConfig

Table 618. 函数 TIM1_ETRConfig

函数名	TIM1_ETRConfig
函数原形	void TIM1_ETRConfig(u16 TIM1_ExtTRGPrescaler, u16 TIM1_ExtTRGPolarity, u8 ExtTRGFilter)
功能描述	配置 TIM1 外部触发
输入参数 1	TIM1_ExtTRGPrescaler：外部触发预分频 参阅 Section：TIM1_ExtTRGPrescaler 查阅更多该参数允许取值范围
输入参数 2	TIM1_ExtTRGPolarity：外部时钟极性 参阅 Section：TIM1_ExtTRGPolarity 查阅更多该参数允许取值范围
输入参数 3	ExtTRGFilter：外部触发滤波器。该参数取值在 0x0 和 0xF 之间。
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Configure the External Trigger (ETR) for TIM1: the rising edge is
the active edge, no filter sampling is done (ExtTRGFilter = 0) and
the prescaler is fixed to TIM1_ExtTRGPSC_DIV2 */
TIM1_ExternalCLK2Config(TIM1_ExtTRGPSC_DIV2,
TIM1_ExtTRGPolarity_NonInverted, 0x0);
```

20.2.23 函数TIM1_ITRxExternalClockConfig

Table 619. 描述了函数TIM1_ITRxExternalClockConfig

Table 619. 函数 TIM1_ITRxExternalClockConfig

函数名	TIM1_ITRxExternalClockConfig
函数原形	void TIM1_ITRxExternalClockConfig(u16 TIM1_InputTriggerSource)
功能描述	设置 TIM1 内部触发为外部时钟模式
输入参数	TIM1_InputTriggerSource：输入触发源 参阅 Section：TIM1_InputTriggerSource 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM1_InputTriggerSource

TIM1_InputTriggerSource 选择 TIM 输入触发。见 Table 620. 参阅该参数的取值。

Table 620. TIM1_InputTriggerSource 值

TIM1_InputTriggerSource	描述
TIM1_TS_ITR0	TIM1 内部触发 0
TIM1_TS_ITR1	TIM1 内部触发 1
TIM1_TS_ITR2	TIM1 内部触发 2
TIM1_TS_ITR3	TIM1 内部触发 3

例：

```
/* TIM1 internal trigger 3 used as clock source */
TIM1_ITRxExternalClockConfig(TIM1_TS_ITR3);
```

20.2.24 函数TIM1_TIxExternalClockConfig

Table 621. 描述了函数TIM1_TIxExternalClockConfig

Table 621. 函数 TIM1_TIxExternalClockConfig

函数名	TIM1_TIxExternalClockConfig
函数原形	void TIM1_TIxExternalClockConfig(u16 TIM1_TIxExternalCLKSource, u16 TIM1_ICPolarity, u16 ICFilter)
功能描述	设置 TIM1 触发为外部时钟
输入参数 1	TIM1_ TIxExternalCLKSource：触发源 参阅 Section：TIM1_ TIxExternalCLKSource 查阅更多该参数允许取值范围
输入参数 2	TIM1_ ICPolarity：指定的 TI 极性 参阅 Section：TIM1_ ICPolarity 查阅更多该参数允许取值范围
输入参数 3	ICFilter：指定的输入比较滤波器。该参数取值在 0x0 和 0xF 之间。
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM1_TIxExternalCLKSource

TIM1_TIxExternalCLKSource选择TIM1外部时钟源。见Table 622. 参阅该参数的取值。

Table 622. TIM1_TIxExternalCLKSource 值

TIM1_TIxExternalCLKSource	描述
TIM1_TS_TI1FP1	TIM1 IC1 连接到 TI1
TIM1_TS_TI1FP2	TIM1 IC2 连接到 TI2
TIM1_TS_TI1F_ED	TIM1 IC1 连接到 TI1：使用边沿探测

例：

```
/* Selects the TI1 as clock for TIM1: the external clock is
connected to TI1 input pin, the rising edge is the active edge and
no filter sampling is done (ICFilter = 0) */
TIM1_TIxExternalClockConfig(TIM1_TS_TI1FP1, TIM1_ICPolarity_Rising, 0);
```

20.2.25 函数TIM1_SelectInputTrigger

Table 623. 描述了函数TIM1_SelectInputTrigger

Table 623. 函数 TIM1_SelectInputTrigger

函数名	TIM1_SelectInputTrigger
函数原形	void TIM1_SelectInputTrigger(TIM1_TypeDef* TIM1, u16 TIM1_InputTriggerSource)
功能描述	选择 TIM1 输入触发源
输入参数	TIM1_InputTriggerSource：输入触发源 参阅 Section：TIM1_InputTriggerSource 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM1_InputTriggerSource

TIM1_InputTriggerSource选择TIM1输入触发源。见Table 498. 参阅该参数的取值。

Table 624. TIM1_InputTriggerSource 值

TIM1_InputTriggerSource	描述
TIM1_TS_ITR0	TIM1 内部触发 0
TIM1_TS_ITR1	TIM1 内部触发 1
TIM1_TS_ITR2	TIM1 内部触发 2
TIM1_TS_ITR3	TIM1 内部触发 3
TIM1_TS_TI1F_ED	TIM 1 TL1 边沿探测器
TIM1_TS_TI1FP1	TIM1 经滤波定时器输入 1
TIM1_TS_TI2FP2	TIM1 经滤波定时器输入 2
TIM1_TS_ETRF	TIM1 外部触发输入

例：

```
/* Selects the Internal Trigger 3 as input trigger fot TIM1 */
void TIM1_SelectInputTrigger(TIM1_TS_ITR3);
```

20.2.26 函数TIM1_UpdateDisableConfig

Table 625. 描述了函数TIM1_UpdateDisableConfig

Table 625. 函数 TIM1_UpdateDisableConfig

函数名	TIM1_UpdateDisableConfig
函数原形	void TIM1_UpdateDisableConfig(FunctionalState Newstate)
功能描述	使能或者失能 TIM1 更新事件
输入参数	NewState：TIM1_CR1 寄存器 UDIS 位的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enables the Update event for TIM1 */
TIM1_UpdateDisableConfig(DISABLE);
```

20.2.27 函数TIM1_UpdateRequestConfig

Table 626. 描述了函数TIM1_UpdateRequestConfig

Table 626. 函数 TIM1_UpdateRequestConfig

函数名	TIM1_UpdateRequestConfig
函数原形	void TIM1_UpdateRequestConfig(u8 TIM1_UpdateSource)
功能描述	设置 TIM1 更新请求源
输入参数	TIM1_UpdateSource：TIM 更新请求源 参阅 Section：TIM1_UpdateSource 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM1_UpdateSource

TIM1_UpdateSource 选择 TIM1 更新源。见 Table 534. 参阅该参数的取值。

Table 627. TIM1_UpdateSource 值

TIM1_UpdateSource	描述
TIM1_UpdateSource_Global	生成重复的脉冲：在更新事件时计数器不停止
TIM1_UpdateSource_Regular	生成单一的脉冲：计数器在下一个更新事件停止

例：
```
/* Selects the regular update source for TIM1 */
TIM1_UpdateRequestConfig(TIM1_UpdateSource_Regular);
```

20.2.28 函数TIM1_SelectHallSensor

Table 628. 描述了函数TIM1_SelectHallSensor

Table 628. 函数 TIM1_SelectHallSensor

函数名	TIM1_SelectHallSensor
函数原形	void TIM1_SelectHallSensor(TIM1_TypeDef* TIM1, FunctionalState Newstate)
功能描述	使能或者失能 TIM1 霍尔传感器接口
输入参数	NewState：TIM1 霍尔传感器接口的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：
```
/* Selects the Hall Sensor Interface for TIM1 */
TIM1_SelectHallSensor(ENABLE);
```

20.2.29 函数TIM1_SelectOnePulseMode

Table 629. 描述了函数TIM1_SelectOnePulseMode

Table 629. 函数 TIM1_SelectOnePulseMode

函数名	TIM1_SelectOnePulseMode
函数原形	void TIM1_SelectOnePulseMode(u16 TIM1_OPMode)
功能描述	设置 TIM1 单脉冲模式
输入参数	TIM1_OPMode：OPM 模式 参阅 Section：TIM1_OPMode 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM1_OPMode

TIM1_OPMode 选择 TIM1 更新源。见 Table 630. 参阅该参数的取值。

Table 630. TIM1_OPMode 值

TIM1_OPMode	描述
TIM1_OPMode_Repetitive	生成重复的脉冲：在更新事件时计数器不停止
TIM1_OPMode_Single	生成单一的脉冲：计数器在下一个更新事件停止

例：
```
/* Selects the Single One Pulse Mode for TIM1 */
TIM1_SelectOnePulseMode(TIM1_OPMode_Single);
```

20.2.30 函数TIM1_SelectOutputTrigger

Table 631. 描述了函数TIM1_SelectOutputTrigger

Table 631. 函数 TIM1_SelectOutputTrigger

函数名	TIM1_SelectOutputTrigger
函数原形	void TIM1_SelectOutputTrigger(u16 TIM1_TRGOSource)
功能描述	选择 TIM1 触发输出模式
输入参数	TIM1_TRGOSource：触发输出模式 参阅 Section：TIM1_TRGOSource 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM1_TRGOSource

TIM1_TRGOSource 选择 TIM 触发输出源。见 Table 632. 参阅该参数的取值。

Table 632. TIM1_TRGOSource 值

TIM1_TRGOSource	描述
TIM1_TRGOSource_Reset	使用寄存器 TIM1_EGR 的 UG 位作为触发输出（TRGO）
TIM1_TRGOSource_Enable	使用计数器使能 CEN 作为触发输出（TRGO）
TIM1_TRGOSource_Update	使用更新事件作为触发输出（TRGO）
TIM1_TRGOSource_OC1	一旦捕获或者比较匹配发生，当标志位 CC1F 被设置时触发输出发送一个肯定脉冲（TRGO）
TIM1_TRGOSource_OC1Ref	使用 OC1REF 作为触发输出（TRGO）
TIM1_TRGOSource_OC2Ref	使用 OC2REF 作为触发输出（TRGO）
TIM1_TRGOSource_OC3Ref	使用 OC3REF 作为触发输出（TRGO）
TIM1_TRGOSource_OC4Ref	使用 OC4REF 作为触发输出（TRGO）

例：

```
/* Selects the update event as TRGO for TIM1 */
TIM1_SelectOutputTrigger(TIM1_TRGOSource_Update);
```

20.2.31 函数TIM1_SelectSlaveMode

Table 633. 描述了函数TIM1_SelectSlaveMode

Table 633. 函数 TIM1_SelectSlaveMode

函数名	TIM1_SelectSlaveMode
函数原形	void TIM1_SelectSlaveMode(u16 TIM1_SlaveMode)
功能描述	选择 TIM1 从模式
输入参数	TIM1_SlaveMode：TIM 从模式 参阅 Section：TIM1_SlaveMode 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM1_SlaveMode

TIM1_SlaveMode 选择 TIM 从模式。见 Table 541. 参阅该参数的取值。

Table 634. TIM1_SlaveMode 值

TIM1_SlaveMode	描述
TIM1_SlaveMode_Reset	选中触发信号（TRGI）的上升沿重初始化计数器并触发寄存器的更新
TIM1_SlaveMode_Gated	当触发信号（TRGI）为高电平计数器时钟使能
TIM1_SlaveMode_Trigger	计数器在触发（TRGI）的上升沿开始
TIM1_SlaveMode_External1	选中触发（TRGI）的上升沿作为计数器时钟

例：

```
/* Selects the Gated Mode as Slave Mode for TIM1 */
TIM1_SelectSlaveMode(TIM1_SlaveMode_Gated);
```

20.2.32 函数TIM1_SelectMasterSlaveMode

Table 635. 描述了函数TIM1_SelectMasterSlaveMode

Table 635. 函数 TIM1_SelectMasterSlaveMode

函数名	TIM1_SelectMasterSlaveMode
函数原形	void TIM1_SelectMasterSlaveMode(u16 TIM1_MasterSlaveMode)
功能描述	设置或者重置 TIM1 主/从模式
输入参数	TIM1_MasterSlaveMode：定时器主/从模式 参阅 Section：TIM1_MasterSlaveMode 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM1_MasterSlaveMode

TIM1_MasterSlaveMode 选择 TIM 主/从模式。见 Table 543. 参阅该参数的取值。

Table 636. TIM1_MasterSlaveMode 值

TIM1_MasterSlaveMode	描述
TIM1_MasterSlaveMode_Enable	TIM1 主/从模式使能
TIM1_MasterSlaveMode_Disable	TIM1 主/从模式失能

例：

```
/* Enables the Master Slave Mode for TIM2 */
TIM1_SelectMasterSlaveMode(TIM2, TIM1_MasterSlaveMode_Enable);
```

20.2.33 函数TIM1_EncoderInterfaceConfig

Table 637. 描述了函数TIM1_EncoderInterfaceConfig

Table 637. 函数 TIM1_EncoderInterfaceConfig

函数名	TIM1_EncoderInterfaceConfig
函数原形	void TIM1_EncoderInterfaceConfig(u16 TIM1_EncoderMode, u16 TIM1_IC1Polarity, u16 TIM1_IC2Polarity)
功能描述	设置 TIM1 编码界面
输入参数 1	TIM1_EncoderMode：触发源 参阅 Section：TIM1_EncoderMode 查阅更多该参数允许取值范围
输入参数 2	TIM1_ IC1Polarity：TI1 极性 参阅 Section：TIM1_ ICPolarity 查阅更多该参数允许取值范围
输入参数 3	TIM1_ IC2Polarity：TI2 极性 参阅 Section：TIM1_ ICPolarity 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM1_EncoderMode

TIM1_EncoderMode选择TIM1编码模式。见Table 526. 参阅该参数的取值。

Table 638. TIM1_EncoderMode 值

TIM1_EncoderMode	描述
TIM1_EncoderMode_TI1	使用 TIM1 编码模式 1
TIM1_EncoderMode_TI1	使用 TIM1 编码模式 2
TIM1_EncoderMode_TI12	使用 TIM1 编码模式 3

例：

```
/* uses of the TIM1 Encoder interface */
TIM1_EncoderInterfaceConfig(TIM1_EncoderMode_1,
TIM1_ICPolarity_Rising,
TIM1_ICPolarity_Rising);
```

20.2.34 函数TIM1_PrescalerConfig

Table 639. 描述了函数TIM1_PrescalerConfig

Table 639. 函数 TIM1_PrescalerConfig

函数名	TIM1_PrescalerConfig
函数原形	void TIM1_PrescalerConfig(u16 Prescaler, u16 TIM1_PSCReloadMode)
功能描述	设置 TIM1 预分频
输入参数	TIM1_PSCReloadMode：预分频重载模式 参阅 Section：TIM1_PSCReloadMode 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM1_PSCReloadMode

TIM1_PSCReloadMode选择预分频重载模式。见Table 640. 参阅该参数的取值。

Table 640. TIM1_PSCReloadMode 值

TIM1_PSCReloadMode	描述
TIM1_PSCReloadMode_Update	TIM1 预分频值在更新事件装入
TIM1_PSCReloadMode_Immediate	TIM1 预分频值即时装入

例：

```
/* Sets the TIM1 new Prescaler value */
u16 TIM1Prescaler = 0xFF00;
TIM1_SetPrescaler(TIM1Prescaler, TIM1_PSCReloadMode_Update);
```

20.2.35 函数TIM1_CounterModeConfig

Table 641. 描述了函数TIM1_CounterModeConfig

Table 641. 函数 TIM1_CounterModeConfig

函数名	TIM1_CounterModeConfig
函数原形	void TIM1_CounterModeConfig(u16 TIM1_CounterMode)
功能描述	设置 TIM1 计数器模式
输入参数	TIM1_CounterMode：待使用的计数器模式 参阅 Section：TIM1_CounterMode 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Selects the Center Aligned counter Mode 1 for the TIM1 */
TIM1_CounterModeConfig(TIM1_Counter_CenterAligned1);
```

20.2.36 函数TIM1_ForcedOC1Config

Table 642. 描述了函数TIM1_ForcedOC1Config

Table 642. 函数 TIM1_ForcedOC1Config

函数名	TIM1_ForcedOC1Config
函数原形	void TIM1_ForcedOC1Config(u16 TIM1_ForcedAction)
功能描述	置 TIM1 输出 1 为活动或者非活动电平
输入参数	TIM1_ForcedAction：输出信号的设置动作 参阅 Section：TIM1_ForcedAction 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM1_ForcedAction

输出信号的设置动作取值见下表。

Table 643. TIM1_ForcedAction 值

TIM1_ForcedAction	描述
TIM1_ForcedAction_Active	置为 OCxREF 上的活动电平
TIM1_ForcedAction_InActive	置为 OCxREF 上的非活动电平

例：

```
/* Forces the TIM1 Channel1 Output to the active level */
TIM1_ForcedOC1Config(TIM1_ForcedAction_Active);
```

20.2.37 函数TIM1_ForcedOC2Config

Table 644. 描述了函数TIM1_ForcedOC2Config

Table 644. 函数 TIM1_ForcedOC2Config

函数名	TIM1_ForcedOC2Config
函数原形	void TIM1_ForcedOC2Config(u16 TIM1_ForcedAction)
功能描述	置 TIM1 输出 2 为活动或者非活动电平
输入参数	TIM1_ForcedAction：输出信号的设置动作 参阅 Section：TIM1_ForcedAction 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Forces the TIM1 Channel2 Output to the active level */
TIM1_ForcedOC2Config(TIM1_ForcedAction_Active);
```

20.2.38 函数TIM1_ForcedOC3Config

Table 645. 描述了函数TIM1_ForcedOC3Config

Table 645. 函数 TIM1_ForcedOC3Config

函数名	TIM1_ForcedOC3Config
函数原形	void TIM1_ForcedOC3Config(u16 TIM1_ForcedAction)
功能描述	置 TIM1 输出 3 为活动或者非活动电平
输入参数	TIM1_ForcedAction：输出信号的设置动作 参阅 Section：TIM1_ForcedAction 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Forces the TIM1 Channel3 Output to the active level */
TIM1_ForcedOC3Config(TIM1_ForcedAction_Active);
```

20.2.39 函数TIM1_ForcedOC4Config

Table 646. 描述了函数TIM1_ForcedOC4Config

Table 646. 函数 TIM1_ForcedOC4Config

函数名	TIM1_ForcedOC4Config
函数原形	void TIM1_ForcedOC4Config(u16 TIM1_ForcedAction)
功能描述	置 TIM1 输出 4 为活动或者非活动电平
输入参数	TIM1_ForcedAction：输出信号的设置动作 参阅 Section：TIM1_ForcedAction 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Forces the TIM1 Channel4 Output to the active level */
TIM1_ForcedOC4Config(TIM1_ForcedAction_Active);
```

20.2.40 函数TIM1_ARRPreloadConfig

Table 647. 描述了函数TIM1_ARRPreloadConfig

Table 647. 函数 TIM1_ARRPreloadConfig

函数名	TIM1_SelectCOM
函数原形	void TIM1_ARRPreloadConfig(FunctionalState Newstate)
功能描述	使能或者失能 TIM1 在 ARR 上的预装载寄存器
输入参数	NewState: TIM1_CR1 寄存器 ARPE 位的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enables the TIM1 Preload on ARR Register */
TIM1_ARRPreloadConfig(ENABLE);
```

20.2.41 函数TIM1_SelectCOM

Table 648. 描述了函数TIM1_SelectCOM

Table 648. 函数 TIM1_SelectCOM

函数名	TIM1_SelectCOM
函数原形	void TIM1_SelectCOM(FunctionalState Newstate)
功能描述	选择 TIM1 外设的通讯事件
输入参数	NewState: 通讯事件的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Selects the TIM1 Commutation event */
TIM1_SelectCOM(ENABLE);
```

20.2.42 函数TIM1_SelectCCDMA

Table 649. 描述了函数TIM1_SelectCCDMA

Table 649. 函数 TIM1_SelectCCDMA

函数名	TIM1_SelectCCDMA
函数原形	void TIM1_SelectCCDMA(FunctionalState Newstate)
功能描述	选择 TIM1 外设的捕获比较 DMA 源
输入参数	NewState: 捕获比较 DMA 源的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Selects the TIM1 Capture Compare DMA source */
TIM1_SelectCCDMA(ENABLE);
```

20.2.43 函数TIM1_CCPreloadControl

Table 650. 描述了函数 TIM1_CCPreloadControl

Table 650. 函数 TIM1_CCPreloadControl

函数名	TIM1_CCPreloadControl
函数原形	void TIM1_CCPreloadControl(FunctionalState Newstate)
功能描述	设置或者重置 TIM1 捕获比较控制位
输入参数	NewState: 捕获比较 DMA 源的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Selects the TIM1 Capture Compare Preload Control */
TIM1_CCPreloadControl(ENABLE);
```

20.2.44 函数TIM1_OC1PreloadConfig

Table 651. 描述了函数TIM1_OC1PreloadConfig

Table 651. 函数 TIM1_OC1PreloadConfig

函数名	TIM1_OC1PreloadConfig
函数原形	void TIM1_OC1PreloadConfig(u16 TIM1_OCPreload)
功能描述	使能或者失能 TIM1 在 CCR1 上的预装载寄存器
输入参数	TIM1_OCPreload：输出比较预装载状态 参阅 Section：TIM1_OCPreload 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM1_OCPreload

输出比较预装载状态可以使能或者失能如下表。

Table 652. TIM1_OCPreload 值

TIM1_OCPreload	描述
TIM1_OCPreload_Enable	TIM1 在 CCR1 上的预装载寄存器使能
TIM1_OCPreload_Disable	TIM1 在 CCR1 上的预装载寄存器失能

例：

```
/* Enables the TIM1 Preload on CC1 Register */
TIM1_OC1PreloadConfig(TIM1_OCPreload_Enable);
```

20.2.45 函数TIM1_OC2PreloadConfig

Table 653. 描述了函数TIM1_OC2PreloadConfig

Table 653. 函数 TIM1_OC2PreloadConfig

函数名	TIM1_OC2PreloadConfig
函数原形	void TIM1_OC2PreloadConfig(u16 TIM1_OCPreload)
功能描述	使能或者失能 TIM1 在 CCR2 上的预装载寄存器
输入参数	TIM1_OCPreload：输出比较预装载状态 参阅 Section：TIM1_OCPreload 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enables the TIM1 Preload on CC2 Register */
TIM1_OC2PreloadConfig(TIM1_OCPreload_Enable);
```

20.2.46 函数TIM1_OC3PreloadConfig

Table 654. 描述了函数TIM1_OC3PreloadConfig

Table 654. 函数 TIM1_OC3PreloadConfig

函数名	TIM1_OC3PreloadConfig
函数原形	void TIM1_OC3PreloadConfig(u16 TIM1_OCPreload)
功能描述	使能或者失能 TIM1 在 CCR3 上的预装载寄存器
输入参数	TIM1_OCPreload：输出比较预装载状态 参阅 Section：TIM1_OCPreload 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enables the TIM1 Preload on CC3 Register */
TIM1_OC3PreloadConfig(TIM1_OCPreload_Enable);
```

20.2.47 函数TIM1_OC4PreloadConfig

Table 655. 描述了函数TIM1_OC4PreloadConfig

Table 655. 函数 TIM1_OC4PreloadConfig

函数名	TIM1_OC4PreloadConfig
函数原形	void TIM1_OC4PreloadConfig(u16 TIM1_OCPreload)
功能描述	使能或者失能 TIM1 在 CCR4 上的预装载寄存器
输入参数	TIM1_OCPreload：输出比较预装载状态 参阅 Section：TIM1_OCPreload 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enables the TIM1 Preload on CC4 Register */
TIM1_OC4PreloadConfig(TIM1_OCPreload_Enable);
```

20.2.48 函数TIM1_OC1FastConfig

Table 656. 描述了函数TIM1_OC1FastConfig

Table 656. 函数 TIM1_OC1FastConfig

函数名	TIM1_OC1FastConfig
函数原形	void TIM1_OC1FastConfig(u16 TIM1_OCFast)
功能描述	设置 TIM1 捕获比较 1 快速特征
输入参数	TIM1_OCFast：输出比较快速特征状态 参阅 Section：TIM1_OCFast 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM1_OCFast

输出比较快速特征性能可以使能或者失能如下表。

Table 657. TIM1_OCPreload 值

TIM1_OCFast	描述
TIM1_OCFast_Enable	TIM1 输出比较快速特征性能使能
TIM1_OCFast_Disable	TIM1 输出比较快速特征性能失能

例：

```
/* Use the TIM1 OC1 in fast Mode */
TIM1_OC1FastConfig(TIM1_OCFast_Enable);
```

20.2.49 函数TIM1_OC2FastConfig

Table 658. 描述了函数TIM1_OC2FastConfig

Table 658. 函数 TIM1_OC2FastConfig

函数名	TIM1_OC2FastConfig
函数原形	void TIM1_OC2FastConfig(u16 TIM1_OCFast)
功能描述	设置 TIM1 捕获比较 2 快速特征
输入参数	TIM1_OCFast：输出比较快速特征状态 参阅 Section：TIM1_OCFast 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Use the TIM1 OC2 in fast Mode */
TIM1_OC2FastConfig(TIM1_OCFast_Enable);
```

20.2.50 函数TIM1_OC3FastConfig

Table 659. 描述了函数TIM1_OC3FastConfig

Table 659. 函数 TIM1_OC3FastConfig

函数名	TIM1_OC3FastConfig
函数原形	void TIM1_OC3FastConfig(u16 TIM1_OCFast)
功能描述	设置 TIM1 捕获比较 3 快速特征
输入参数	TIM1_OCFast：输出比较快速特征状态 参阅 Section：TIM1_OCFast 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Use the TIM1 OC3 in fast Mode */
TIM1_OC3FastConfig(TIM1_OCFast_Enable);
```

20.2.51 函数TIM1_OC4FastConfig

Table 660. 描述了函数TIM1_OC4FastConfig

Table 660. 函数 TIM1_OC4FastConfig

函数名	TIM1_OC4FastConfig
函数原形	void TIM1_OC4FastConfig(u16 TIM1_OCFast)
功能描述	设置 TIM1 捕获比较 4 快速特征
输入参数	TIM1_OCFast：输出比较快速特征状态 参阅 Section：TIM1_OCFast 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Use the TIM1 OC4 in fast Mode */
TIM1_OC4FastConfig(TIM1_OCFast_Enable);
```

20.2.52 函数TIM1_ClearOC1Ref

Table 661. 描述了函数TIM1_ClearOC1Ref

Table 661. 函数 TIM1_ClearOC1Ref

函数名	TIM1_ClearOC1Ref
函数原形	void TIM1_ClearOC1Ref(u16 TIM1_OCClear)
功能描述	在一个外部事件时清除或者保持 OCREF1 信号
输入参数	TIM1_OCClear：输出比较清除使能位状态 参阅 Section：TIM1_OCClear 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM1_OCClear

输出比较清除使能位的值列举如下表。

Table 662. TIM1_OCClear 值

TIM1_OCClear	描述
TIM1_OCClear_Enable	TIM1 输出比较清除使能
TIM1_OCClear_Disable	TIM1 输出比较清除失能

例：

```
/* Enable the TIM1 Channel1 Ouput Compare Refence clear bit */
TIM1_ClearOC1Ref(TIM1_OCClear_Enable);
```

20.2.53 函数TIM1_ClearOC2Ref

Table 663. 描述了函数TIM1_ClearOC2Ref

Table 663. 函数 TIM1_ClearOC2Ref

函数名	TIM1_ClearOC2Ref
函数原形	void TIM1_ClearOC2Ref(u16 TIM1_OCClear)
功能描述	在一个外部事件时清除或者保持 OCREF2 信号
输入参数	TIM1_OCClear：输出比较清除使能位状态 参阅 Section：TIM1_OCClear 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable the TIM1 Channel2 Ouput Compare Refence clear bit */
TIM1_ClearOC2Ref(TIM1_OCClear_Enable);
```

20.2.54 函数TIM1_ClearOC3Ref

Table 664. 描述了函数TIM1_ClearOC3Ref

Table 664. 函数 TIM1_ClearOC3Ref

函数名	TIM1_ClearOC3Ref
函数原形	void TIM1_ClearOC3Ref(u16 TIM1_OCClear)
功能描述	在一个外部事件时清除或者保持 OCREF3 信号
输入参数	TIM1_OCClear：输出比较清除使能位状态 参阅 Section：TIM1_OCClear 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable the TIM1 Channel3 Ouput Compare Refence clear bit */
TIM1_ClearOC3Ref(TIM1_OCClear_Enable);
```

20.2.55 函数TIM1_ClearOC4Ref

Table 665. 描述了函数TIM1_ClearOC4Ref

Table 665. 函数 TIM1_ClearOC4Ref

函数名	TIM1_ClearOC4Ref
函数原形	void TIM1_ClearOC4Ref(u16 TIM1_OCClear)
功能描述	在一个外部事件时清除或者保持 OCREF4 信号
输入参数 1	TIM1：x 可以是 2，3 或者 4，来选择 TIM 外设
输入参数 2	TIM1_OCClear：输出比较清除使能位状态 参阅 Section：TIM1_OCClear 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable the TIM1 Channel4 Ouput Compare Refence clear bit */
TIM1_ClearOC4Ref(TIM1_OCClear_Enable);
```

20.2.56 函数TIM1_GenerateEvent

Table 666. 描述了函数TIM1_GenerateEvent

Table 666. 函数 TIM1_GenerateEvent

函数名	TIM1_GenerateEvent
函数原形	void TIM1_GenerateEvent(u16 TIM1_EventSource)
功能描述	设置 TIM1 事件由软件产生
输入参数	TIM1_EventSource：TIM 软件事件源 参阅 Section：TIM1_EventSource 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM1_EventSource

TIM1_EventSource 选择 TIM 软件事件源。见 Table 667. 参阅该参数的取值。

Table 667. TIM1_EventSource 值

TIM1_EventSource	描述
TIM1_EventSource_Update	TIM 更新事件源
TIM1_EventSource_CC1	TIM 捕获比较 1 事件源
TIM1_EventSource_CC2	TIM 捕获比较 2 事件源
TIM1_EventSource_CC3	TIM 捕获比较 3 事件源
TIM1_EventSource_CC4	TIM 捕获比较 4 事件源
TIM1_EventSource_Trigger	TIM 触发事件源

例：

```
/* Selects the Trigger software Event generation for TIM1 */
TIM1_GenerateEvent(TIM1_EventSource_Trigger);
```

20.2.57 函数TIM1_OC1PolarityConfig

Table 668. 描述了函数TIM1_OC1PolarityConfig

Table 668. 函数 TIM1_OC1PolarityConfig

函数名	TIM1_OC1PolarityConfig
函数原形	void TIM1_OC1PolarityConfig(u16 TIM1_OCPolarity)
功能描述	设置 TIM1 通道 1 极性
输入参数	TIM1_OCPolarity：输出比较极性 参阅 Section：TIM1_OCPolarity 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM1_OCPolarity

TIM1_OCPolarity 选择 TIM1 极性（见 Table 699.）

Table 699. TIM1_OCPolarity 值

TIM1_OCPolarity	描述
TIM1_OCPolarity_High	TIM1 输出比较极性高
TIM1_OCPolarity_Low	TIM1 输出比较极性低

例：

```
/* Selects the Polarity high for TIM1 channel 1 output compare */
TIM1_OC1PolarityConfig(TIM1_OCPolarity_High);
```

20.2.58 函数TIM1_OC1NPolarityConfig

Table 670. 描述了函数TIM1_OC1NPolarityConfig

Table 670. 函数 TIM1_OC1NPolarityConfig

函数名	TIM1_OC1NPolarityConfig
函数原形	void TIM1_OC1NPolarityConfig(u16 TIM1_OCNPolarity)
功能描述	设置 TIM1 通道 1N 极性
输入参数	TIM1_OCNPolarity：输出比较极性 参阅 Section：TIM1_OCNPolarity 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Selects the Polarity high for TIM1 channel 1N output compare */
TIM1_OC1NPolarityConfig(TIM1_OCNPolarity_High);
```

20.2.59 函数TIM1_OC2PolarityConfig

Table 671. 描述了函数TIM1_OC2PolarityConfig

Table 671. 函数 TIM1_OC2PolarityConfig

函数名	TIM1_OC2PolarityConfig
函数原形	void TIM1_OC2PolarityConfig(u16 TIM1_OCPolarity)
功能描述	设置 TIM1 通道 2 极性
输入参数	TIM1_OCPolarity：输出比较极性 参阅 Section：TIM1_OCPolarity 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Selects the Polarity high for TIM1 channel 2 output compare */
TIM1_OC2PolarityConfig(TIM1_OCPolarity_High);
```

20.2.60 函数TIM1_OC2NPolarityConfig

Table 672. 描述了函数TIM1_OC2PolarityConfig

Table 672. 函数 TIM1_OC2NPolarityConfig

函数名	TIM1_OC2NPolarityConfig
函数原形	void TIM1_OC2NPolarityConfig(u16 TIM1_OCNPolarity)
功能描述	设置 TIM1 通道 2N 极性
输入参数	TIM1_OCNPolarity：输出比较极性 参阅 Section：TIM1_OCNPolarity 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Selects the Polarity high for TIM1 channel 2N output compare */
TIM1_OC2NPolarityConfig(TIM1_OCNPolarity_High);
```

20.2.61 函数TIM1_OC3PolarityConfig

Table 673. 描述了函数TIM1_OC3PolarityConfig

Table 673. 函数 TIM1_OC3PolarityConfig

函数名	TIM1_OC3PolarityConfig
函数原形	void TIM1_OC3PolarityConfig(u16 TIM1_OCPolarity)
功能描述	设置 TIM1 通道 3 极性
输入参数	TIM1_OCPolarity：输出比较极性 参阅 Section：TIM1_OCPolarity 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Selects the Polarity high for TIM1 channel 3 output compare */
TIM1_OC3PolarityConfig(TIM1_OCPolarity_High);
```

20.2.62 函数TIM1_OC3NPolarityConfig

Table 674. 描述了函数TIM1_OC3PolarityConfig

Table 674. 函数 TIM1_OC3NPolarityConfig

函数名	TIM1_OC1NPolarityConfig
函数原形	void TIM1_OC3NPolarityConfig(u16 TIM1_OCNPolarity)
功能描述	设置 TIM1 通道 3N 极性
输入参数	TIM1_OCNPolarity：输出比较极性 参阅 Section：TIM1_OCNPolarity 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Selects the Polarity high for TIM1 channel 3N output compare */
TIM1_OC3NPolarityConfig(TIM1_OCNPolarity_High);
```

20.2.63 函数TIM1_OC4PolarityConfig

Table 675. 描述了函数TIM1_OC4PolarityConfig

Table 675. 函数 TIM1_OC4PolarityConfig

函数名	TIM1_OC1PolarityConfig
函数原形	void TIM1_OC4PolarityConfig(u16 TIM1_OCPolarity)
功能描述	设置 TIM1 通道 4 极性
输入参数	TIM1_OCPolarity：输出比较极性 参阅 Section：TIM1_OCPolarity 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Selects the Polarity high for TIM1 channel 4 output compare */
TIM1_OC4PolarityConfig(TIM1_OCPolarity_High);
```

20.2.64 函数TIM1_CCxCmd

Table 676. 描述了TIM1_CCxCmd

Table 676. 函数 TIM1_CCxCmd

函数名	TIM1_CCxCmd
函数原形	void TIM1_CCxCmd(u16 TIM1_Channel, FunctionalState Newstate)
功能描述	使能或者失能 TIM1 捕获比较通道 x
输入参数 1	TIM1_Channel：TIM1 通道 参阅 Section：TIM1_Channel 查阅更多该参数允许取值范围
输入参数 2	NewState: TIM1 通道 CCxE 位的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enables the TIM1 channel 4 */
TIM1_CCxCmd(TIM1_Channel_4, ENABLE);
```

20.2.65 函数TIM1_CCxNCmd

Table 677. 描述了TIM1_CCxNCmd

Table 677. 函数 TIM1_CCxNCmd

函数名	TIM1_CCxNCmd
函数原形	void TIM1_CCxNCmd(u16 TIM1_Channel, FunctionalState Newstate)
功能描述	使能或者失能 TIM1 捕获比较通道 xN
输入参数 1	TIM1_Channel：TIM1 通道 参阅 Section：TIM1_Channel 查阅更多该参数允许取值范围
输入参数 2	NewState: TIM1 通道 CCxNE 位的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enables the TIM1 channel 3N */
TIM1_CCxNCmd(TIM1_Channel_3, ENABLE);
```

20.2.66 函数TIM1_SelectOCxM

Table 678. 描述了TIM1_SelectOCxM

Table 678. 函数 TIM1_SelectOCxM

函数名	TIM1_SelectOCxM
函数原形	void TIM1_SelectOCxM(u16 TIM1_Channel, u16 TIM1_OCMode)
功能描述	选择 TIM1 输出比较模式。 本函数在改变输出比较模式前失能选中的通道。用户必须使用函数 TIM1_CCxCmd 和 TIM1_CCxNCmd 来使能这个通道。
输入参数 1	TIM1_Channel：TIM1 通道 参阅 Section：TIM1_Channel 查阅更多该参数允许取值范围
输入参数 2	TIM1_OCMode: TIM1 通道 CCxNE 位的新状态 参阅 Section：TIM1_OCMode 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM1_OCMode

TIM1_OCMode 选择定时器模式。该参数取值见下表。

Table 679. TIM1_OCMode 定义

TIM1_OCMode	描述
TIM1_OCMode_TIM1ing	TIM1 输出比较时间模式
TIM1_OCMode_Active	TIM1 输出比较主动模式
TIM1_OCMode_Inactive	TIM1 输出比较非主动模式
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	置活动电平为 OCxREF

TIM1_ForcedAction_InActive	置非活动电平为 OCxREF

例：
```
/* Selects the TIM1 Channel 1 PWM2 Mode */
TIM1_SelectOCxM(TIM1_Channel_1, TIM1_OCMode_PWM2);
```

20.2.67 函数TIM1_SetCounter

Table 680. 描述了函数TIM1_SetCounter

Table 680. 函数 TIM1_SetCounter

函数名	TIM1_SetCounter
函数原形	void TIM1_SetCounter(TIM1_TypeDef* TIM1, u16 Counter)
功能描述	设置 TIM1 计数器寄存器值
输入参数	Counter：计数器寄存器新值
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：
```
/* Sets the TIM1 new Counter value */
u16 TIM1Counter = 0xFFFF;
TIM1_SetCounter(TIM1Counter);
```

20.2.68 函数TIM1_SetAutoreload

Table 681. 描述了函数TIM1_ SetAutoreload

Table 681. 函数 TIM1_ SetAutoreload

函数名	TIM1_ SetAutoreload
函数原形	void TIM1_SetCounter(TIM1_TypeDef* TIM1, u16 Counter)
功能描述	设置 TIM1 自动重装载寄存器值
输入参数	Autoreload：自动重装载寄存器新值
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：
```
/* Sets the TIM1 new Autoreload value */
u16 TIM1Autoreload = 0xFFFF;
TIM1_SetAutoreload(TIM1Autoreload);
```

20.2.69 函数TIM1_SetCompare1

Table 682. 描述了函数TIM1_SetCompare1

Table 682. 函数 TIM1_SetCompare1

函数名	TIM1_SetCompare1
函数原形	void TIM1_SetCompare1(TIM1_TypeDef* TIM1, u16 Compare1)
功能描述	设置 TIM1 捕获比较 1 寄存器值
输入参数	Compare1：捕获比较 1 寄存器新值
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Sets the TIM1 new Output Compare 1 value */
u16 TIM1Compare1 = 0x7FFF;
TIM1_SetCompare1(TIM1Compare1);
```

20.2.70 函数TIM1_SetCompare2

Table 683. 描述了函数TIM1_SetCompare2

Table 683. 函数 TIM1_SetCompare2

函数名	TIM1_SetCompare2
函数原形	void TIM1_SetCompare2(TIM1_TypeDef* TIM1, u16 Compare2)
功能描述	设置 TIM1 捕获比较 2 寄存器值
输入参数	Compare2：捕获比较 2 寄存器新值
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Sets the TIM1 new Output Compare 2 value */
u16 TIM1Compare2 = 0x7FFF;
TIM1_SetCompare2(TIM1Compare2);
```

20.2.71 函数TIM1_SetCompare3

Table 684. 描述了函数TIM1_SetCompare3

Table 684. 函数 TIM1_SetCompare3

函数名	TIM1_SetCompare3
函数原形	void TIM1_SetCompare3(TIM1_TypeDef* TIM1, u16 Compare3)
功能描述	设置 TIM1 捕获比较 3 寄存器值
输入参数	Compare1：捕获比较 3 寄存器新值
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Sets the TIM1 new Output Compare 3 value */
u16 TIM1Compare3 = 0x7FFF;
TIM1_SetCompare1(TIM1Compare3);
```

20.2.72 函数TIM1_SetCompare4

Table 685. 描述了函数TIM1_SetCompare4

Table 685. 函数 TIM1_SetCompare4

函数名	TIM1_SetCompare4
函数原形	void TIM1_SetCompare4(TIM1_TypeDef* TIM1, u16 Compare4)
功能描述	设置 TIM1 捕获比较 4 寄存器值
输入参数	Compare4：捕获比较 4 寄存器新值
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Sets the TIM1 new Output Compare 4 value */
u16 TIM1Compare4 = 0x7FFF;
TIM1_SetCompare1(TIM1Compare4);
```

20.2.73 函数TIM1_SetIC1Prescaler

Table 686. 描述了函数TIM1_SetIC1Prescaler

Table 686. 函数 TIM1_SetIC1Prescaler

函数名	TIM1_SetIC1Prescaler
函数原形	void TIM1_SetIC1Prescaler(TIM1_TypeDef* TIM1, u16 TIM1_IC1Prescaler)
功能描述	设置 TIM1 输入捕获 1 预分频
输入参数	TIM1_IC1Prescaler：输入捕获 1 预分频 参阅 Section：TIM1_IC1Prescaler 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM1_ICPrescaler

TIM1_ICPrescaler设置输入捕获预分频器。该参数取值见下表。

Table 687. TIM1_ICPrescaler 值

TIM1_ICPrescaler	描述
TIM1_ICPSC_DIV1	TIM1 捕获在捕获输入上每探测到一个边沿执行一次
TIM1_ICPSC_DIV2	TIM1 捕获每 2 个事件执行一次
TIM1_ICPSC_DIV3	TIM1 捕获每 3 个事件执行一次
TIM1_ICPSC_DIV4	TIM1 捕获每 4 个事件执行一次

例：

```
/* Sets the TIM1 Input Capture 1 Prescaler */
TIM1_SetIC1Prescaler(TIM1_ICPSC_Div2);
```

20.2.74 函数TIM1_SetIC2Prescaler

Table 688. 描述了函数TIM1_SetIC2Prescaler

Table 688. 函数 TIM1_SetIC2Prescaler

函数名	TIM1_SetIC2Prescaler
函数原形	void TIM1_SetIC2Prescaler(TIM1_TypeDef* TIM1, u16 TIM1_IC2Prescaler)
功能描述	设置 TIM1 输入捕获 2 预分频
输入参数	TIM1_IC2Prescaler：输入捕获 2 预分频 参阅 Section：TIM1_IC1Prescaler 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Sets the TIM1 Input Capture 2 Prescaler */
TIM1_SetIC2Prescaler(TIM1_ICPSC_Div2);
```

20.2.75 函数TIM1_SetIC3Prescaler

Table 689. 描述了函数TIM1_SetIC3Prescaler

Table 689. 函数 TIM1_SetIC3Prescaler

函数名	TIM1_SetIC3Prescaler
函数原形	void TIM1_SetIC3Prescaler(TIM1_TypeDef* TIM1, u16 TIM1_IC3Prescaler)
功能描述	设置 TIM1 输入捕获 3 预分频
输入参数	TIM1_IC1Prescaler：输入捕获 3 预分频 参阅 Section：TIM1_IC1Prescaler 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Sets the TIM1 Input Capture 3 Prescaler */
TIM1_SetIC3Prescaler(TIM1_ICPSC_Div2);
```

20.2.76 函数TIM1_SetIC4Prescaler

Table 690. 描述了函数TIM1_SetIC4Prescaler

Table 690. 函数 TIM1_SetIC4Prescaler

函数名	TIM1_SetIC4Prescaler
函数原形	void TIM1_SetIC1Prescaler(TIM1_TypeDef* TIM1, u16 TIM1_IC4Prescaler)
功能描述	设置 TIM1 输入捕获 4 预分频
输入参数	TIM1_IC1Prescaler：输入捕获 4 预分频 参阅 Section：TIM1_IC1Prescaler 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Sets the TIM1 Input Capture 4 Prescaler */
TIM1_SetIC4Prescaler(TIM1_ICPSC_Div2);
```

20.2.77 函数TIM1_SetClockDivision

Table 691. 描述了函数TIM1_SetClockDivision

Table 691. 函数 TIM1_SetClockDivision

函数名	TIM1_SetClockDivision
函数原形	void TIM1_SetClockDivision(TIM1_TypeDef* TIM1, u16 TIM1_CKD)
功能描述	设置 TIM1 的时钟分割值
输入参数	TIM1_CKD：时钟分割值 参阅 Section：TIM1_ClockDivision 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM1_CKD

TIM1_CKD选择了TIM1的时钟分割。（见Table 692.）

Table 692. TIM1_CKD 值

TIM1_CKD	描述
TIM1_CKD_DIV1	TDTS = Tck_TIM1
TIM1_CKD_DIV2	TDTS = 2Tck_TIM1
TIM1_CKD_DIV4	TDTS = 4Tck_TIM1

例：

```
/* Sets the TIM1 CKD value */
TIM1_SetClockDivision(TIM1_CKD_DIV4);
```

20.2.78 函数TIM1_GetCapture1

Table 693. 描述了函数TIM1_GetCapture1

Table 693. 函数 TIM1_GetCapture1

函数名	TIM1_GetCapture1
函数原形	u16 TIM1_GetCapture1(TIM1_TypeDef* TIM1)
功能描述	获得 TIM1 输入捕获 1 的值
输入参数	无
输出参数	无
返回值	输入捕获 1 的值
先决条件	无
被调用函数	无

例：

```
/* Gets the Input Capture 1 value of the TIM1 */
u16 IC1value = TIM1_GetCapture1();
```

20.2.79 函数TIM1_GetCapture2

Table 694. 描述了函数TIM1_GetCapture2

Table 694. 函数 TIM1_GetCapture2

函数名	TIM1_GetCapture2
函数原形	u16 TIM1_GetCapture2(TIM1_TypeDef* TIM1)
功能描述	获得 TIM1 输入捕获 2 的值
输入参数	无
输出参数	无
返回值	输入捕获 2 的值
先决条件	无
被调用函数	无

例：

```
/* Gets the Input Capture 2 value of the TIM1 */
u16 IC2value = TIM1_GetCapture2();
```

20.2.80 函数TIM1_GetCapture3

Table 695. 描述了函数TIM1_GetCaptur3

Table 695. 函数 TIM1_GetCapture3

函数名	TIM1_GetCapture3
函数原形	u16 TIM1_GetCapture3(TIM1_TypeDef* TIM1)
功能描述	获得 TIM1 输入捕获 3 的值
输入参数	无
输出参数	无
返回值	输入捕获 3 的值
先决条件	无
被调用函数	无

例：

```
/* Gets the Input Capture 3 value of the TIM1 */
u16 IC3value = TIM1_GetCapture3();
```

20.2.81 函数TIM1_GetCapture4

Table 696. 描述了函数TIM1_GetCapture4

Table 696. 函数 TIM1_GetCapture4

函数名	TIM1_GetCapture4
函数原形	u16 TIM1_GetCapture4(TIM1_TypeDef* TIM1)
功能描述	获得 TIM1 输入捕获 4 的值
输入参数	无
输出参数	无
返回值	输入捕获 4 的值
先决条件	无
被调用函数	无

例：

```
/* Gets the Input Capture 4 value of the TIM1 */
u16 IC4value = TIM1_GetIC4();
```

20.2.82 函数TIM1_GetCounter

Table 697. 描述了函数TIM1_GetCounter

Table 697. 函数 TIM1_GetCounter

函数名	TIM1_GetCounter
函数原形	u16 TIM1_GetCounter(TIM1_TypeDef* TIM1)
功能描述	获得 TIM1 计数器的值
输入参数	无
输出参数	无
返回值	计数器的值
先决条件	无
被调用函数	无

例：

```
/* Gets TIM1 counter value */
u16 TIM1Counter = TIM1_GetCounter();
```

20.2.83 函数TIM1_GetPrescaler

Table 698. 描述了函数TIM1_GetPrescaler

Table 698. 函数 TIM1_GetPrescaler

函数名	TIM1_GetPrescaler
函数原形	u16 TIM1_GetPrescaler (TIM1_TypeDef* TIM1)
功能描述	获得 TIM1 预分频值
输入参数	无
输出参数	无
返回值	预分频的值
先决条件	无
被调用函数	无

例：

```
/* Gets TIM1 prescaler value */
u16 TIM1Prescaler = TIM1_GetPrescaler();
```

20.2.84 函数TIM1_GetFlagStatus

Table 699. 描述了函数TIM1_ GetFlagStatus

Table 699. 函数 TIM1_ GetFlagStatus

函数名	TIM1_ GetFlagStatus
函数原形	FlagStatus TIM1_GetFlagStatus(TIM1_TypeDef* TIM1, u16 TIM1_FLAG)
功能描述	检查指定的 TIM 标志位设置与否
输入参数	TIM1_FLAG：待检查的 TIM 标志位 参阅 Section：TIM1_FLAG 查阅更多该参数允许取值范围
输出参数	无
返回值	TIM1_FLAG 的新状态（SET 或者 RESET）
先决条件	无
被调用函数	无

TIM1_FLAG

Table 700. 给出了所有可以被函数TIM1_ GetFlagStatus检查的标志位列表

Table 700. TIM1_FLAG 值

TIM1_FLAG	描述
TIM1_FLAG_Update	TIM1 更新标志位
TIM1_FLAG_CC1	TIM1 捕获/比较 1 标志位
TIM1_FLAG_CC2	TIM1 捕获/比较 2 标志位
TIM1_FLAG_CC3	TIM1 捕获/比较 3 标志位
TIM1_FLAG_CC4	TIM1 捕获/比较 4 标志位
TIM1_FLAG_COM	TIM1 COM 标志位
TIM1_FLAG_Trigger	TIM1 触发标志位
TIM1_FLAG_BRK	TIM1 刹车标志位
TIM1_FLAG_CC1OF	TIM1 捕获/比较 1 溢出标志位
TIM1_FLAG_CC2OF	TIM1 捕获/比较 2 溢出标志位
TIM1_FLAG_CC3OF	TIM1 捕获/比较 3 溢出标志位
TIM1_FLAG_CC4OF	TIM1 捕获/比较 4 溢出标志位

例：

```
/* Check if the TIM1 Capture Compare 1 flag is set or reset */
if(TIM1_GetFlagStatus(TIM1_FLAG_CC1) == SET)
{
}
```

20.2.85 函数TIM1_ClearFlag

Table 701. 描述了函数TIM1_ ClearFlag

Table 701. 函数 TIM1_ ClearFlag

函数名	TIM1_ ClearFlag
函数原形	void TIM1_ClearFlag(TIM1_TypeDef* TIM1, u32 TIM1_FLAG)
功能描述	清除 TIM1 的待处理标志位
输入参数	TIM1_FLAG：待清除的 TIM 标志位 参阅 Section：TIM1_FLAG 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear the TIM1 Capture Compare 1 flag */
TIM1_ClearFlag(TIM1_FLAG_CC1);
```

20.2.86 函数TIM1_GetITStatus

Table 702. 描述了函数TIM1_ GetITStatus

Table 702. 函数 TIM1_ GetITStatus

函数名	TIM1_ GetITStatus
函数原形	ITStatus TIM1_GetITStatus(TIM1_TypeDef* TIM1, u16 TIM1_IT)
功能描述	检查指定的 TIM 中断发生与否
输入参数	TIM1_IT：待检查的 TIM 中断源 参阅 Section：TIM1_IT 查阅更多该参数允许取值范围
输出参数	无
返回值	TIM1_IT 的新状态
先决条件	无
被调用函数	无

例：

```
/*Check if the TIM1 Capture Compare 1 interrupt has occured or not*/
if(TIM1_GetITStatus(TIM1_IT_CC1) == SET)
{
}
```

20.2.87 函数TIM1_ClearITPendingBit

Table 703. 描述了函数TIM1_ ClearITPendingBit

Table 703. 函数 TIM1_ ClearITPendingBit

函数名	TIM1_ ClearITPendingBit
函数原形	void TIM1_ClearITPendingBit(TIM1_TypeDef* TIM1, u16 TIM1_IT)
功能描述	清除 TIM1 的中断待处理位
输入参数	TIM1_IT：待检查的 TIM 中断待处理位 参阅 Section：TIM1_IT 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear the TIM1 Capture Compare 1 interrupt pending bit */
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）提供了一种灵活的方法来与使用工业标准 NRZ 异步串行数据格式的外部设备之间进行全双工数据交换。USART 利用分数波特率发生器提供宽范围的波特率选择。它支持同步单向通信和半双工单线通信。它也支持 LIN(局部互连网)，智能卡协议和 IrDA(红外数据组织)SIR ENDEC 规范，以及调制解调器(CTS/RTS)操作。它还允许多处理器通信。使用多缓冲器配置的 DMA 方式，可以实现高速数据通信。

Section 21.1 USART 寄存器结构描述了固件函数库所使用的数据结构，Section 21.2 固件库函数介绍了函数库里的所有函数。

21.1 USART寄存器结构

USART 寄存器结构，***USART_TypeDeff***，在文件***"stm32f10x_map.h"***中定义如下：

```
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;
```

Table 704.例举了USART所有寄存器

Table 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 保护时间和预分频寄存器

3 个 USART 外设声明于文件***"stm32f10x_map.h"***：

```
...
#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
```

使用Debug模式时，初始化指针***USART1, USART2***和***USART3***于文件***"stm32f10x_lib.c"***：

```
...
#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 */
...
```

为了访问USART寄存器，,***_USART，_USART1, _USART2***和***_USART3***必须在文件***"stm32f10x_conf.h"***中定义如下：

```
...
#define _USART
#define _USART1
#define _USART2
#define _USART3
```

21.2 USART库函数

Table 705. 例举了USART的库函数

Table 705. USART 库函数

函数名	描述
USART_DeInit	将外设 USARTx 寄存器重设为缺省值
USART_Init	根据 USART_InitStruct 中指定的参数初始化外设 USARTx 寄存器
USART_StructInit	把 USART_InitStruct 中的每一个参数按缺省值填入
USART_Cmd	使能或者失能 USART 外设
USART_ITConfig	使能或者失能指定的 USART 中断
USART_DMACmd	使能或者失能指定 USART 的 DMA 请求
USART_SetAddress	设置 USART 节点的地址
USART_WakeUpConfig	选择 USART 的唤醒方式
USART_ReceiverWakeUpCmd	检查 USART 是否处于静默模式
USART_LINBreakDetectLengthConfig	设置 USART LIN 中断检测长度
USART_LINCmd	使能或者失能 USARTx 的 LIN 模式
USART_SendData	通过外设 USARTx 发送单个数据

USART_ReceiveData	返回 USARTx 最近接收到的数据
USART_SendBreak	发送中断字
USART_SetGuardTime	设置指定的 USART 保护时间
USART_SetPrescaler	设置 USART 时钟预分频
USART_SmartCardCmd	使能或者失能指定 USART 的智能卡模式
USART_SmartCardNackCmd	使能或者失能 NACK 传输
USART_HalfDuplexCmd	使能或者失能 USART 半双工模式
USART_IrDAConfig	设置 USART IrDA 模式
USART_IrDACmd	使能或者失能 USART IrDA 模式
USART_GetFlagStatus	检查指定的 USART 标志位设置与否
USART_ClearFlag	清除 USARTx 的待处理标志位
USART_GetITStatus	检查指定的 USART 中断发生与否
USART_ClearITPendingBit	清除 USARTx 的中断待处理位

21.2.1 函数USART_DeInit

Table 706. 描述了函数 USART_DeInit

Table 706. 函数 USART_DeInit

函数名	USART_DeInit
函数原形	void USART_DeInit(USART_TypeDef* USARTx)
功能描述	将外设 USARTx 寄存器重设为缺省值
输入参数	USARTx：x 可以是 1，2 或者 3，来选择 USART 外设
输出参数	无
返回值	无
先决条件	无
被调用函数	RCC_APB2PeriphResetCmd() RCC_APB1PeriphResetCmd()

例：

```
/* Resets the USART1 registers to their default reset value */
USART_DeInit(USART1);
```

21.2.2 函数USART_Init

Table 707. 描述了函数 USART_Init

Table 707. 函数 USART_Init

函数名	USART_Init
函数原形	void USART_Init(USART_TypeDef* USARTx, USART_InitTypeDef* USART_InitStruct)
功能描述	根据 USART_InitStruct 中指定的参数初始化外设 USARTx 寄存器
输入参数 1	USARTx：x 可以是 1，2 或者 3，来选择 USART 外设
输入参数 2	USART_InitStruct：指向结构 USART_InitTypeDef 的指针，包含了外设 USART 的配置信息。参阅 Section：USART_InitTypeDef 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

USART_InitTypeDef structure

USART_InitTypeDef 定义于文件***"stm32f10x_usart.h"***：

```
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;
```

Table 708. 描述了结构 USART_InitTypeDef 在同步和异步模式下使用的不同成员。

Table 708. USART_InitTypeDef 成员 USART 模式对比

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

该成员设置了 USART 传输的波特率，波特率可以由以下公式计算：

IntegerDivider = ((APBClock) / (16 * (USART_InitStruct->USART_BaudRate)))

FractionalDivider = ((IntegerDivider - ((u32) IntegerDivider)) * 16) + 0.5

USART_WordLength

USART_WordLength 提示了在一个帧中传输或者接收到的数据位数。Table 709. 给出了该参数可取的值。

Table 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 定义了发送的停止位数目。Table 710. 给出了该参数可取的值。

Table 710. USART_StopBits 定义

USART_StopBits	描述
USART_StopBits_1	在帧结尾传输 1 个停止位
USART_StopBits_0.5	在帧结尾传输 0.5 个停止位
USART_StopBits_2	在帧结尾传输 2 个停止位
USART_StopBits_1.5	在帧结尾传输 1.5 个停止位

USART_Parity

USART_Parity 定义了奇偶模式。Table 711. 给出了该参数可取的值。

Table 711. USART_Parity 定义

USART_Parity	描述
USART_Parity_No	奇偶失能
USART_Parity_Even	偶模式
USART_Parity_Odd	奇模式

注意：奇偶校验一旦使能，在发送数据的 MSB 位插入经计算的奇偶位（字长 9 位时的第 9 位，字长 8 位时的第 8 位）。

USART_HardwareFlowControl

USART_HardwareFlowControl 指定了硬件流控制模式使能还是失能。Table 712. 给出了该参数可取的值。

Table 712. USART_HardwareFlowControl 定义

USART_HardwareFlowControl	描述
USART_HardwareFlowControl_None	硬件流控制失能
USART_HardwareFlowControl_RTS	发送请求 RTS 使能
USART_HardwareFlowControl_CTS	清除发送 CTS 使能
USART_HardwareFlowControl_RTS_CTS	RTS 和 CTS 使能

USART_Mode

USART_Mode 指定了使能或者失能发送和接收模式。Table 713. 给出了该参数可取的值。

Table 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 提示了 USART 时钟使能还是失能。Table 714. 给出了该参数可取的值。

Table 714. USART_CLOCK 定义

USART_CLOCK	描述
USART_Clock_Enable	时钟高电平活动
USART_Clock_Disable	时钟低电平活动

USART_CPOL

USART_CPOL 指定了下 SLCK 引脚上时钟输出的极性。Table 715. 给出了该参数可取的值。

Table 715. USART_CPOL 定义

USART_CPOL	描述
USART_CPOL_High	时钟高电平
USART_CPOL_Low	时钟低电平

USART_CPHA

USART_CPHA 指定了下 SLCK 引脚上时钟输出的相位，和 CPOL 位一起配合来产生用户希望的时钟/数据的采样关系。Table 716. 给出了该参数可取的值。

Table 716. USART_CPHA 定义

USART_CPHA	描述
USART_CPHA_1Edge	时钟第一个边沿进行数据捕获
USART_CPHA_2Edge	时钟第二个边沿进行数据捕获

USART_LastBit

USART_LastBit 来控制是否在同步模式下，在 SCLK 引脚上输出最后发送的那个数据字 (MSB)对应的时钟脉冲。Table 717. 给出了该参数可取的值。

Table 717. USART_LastBit 定义

USART_LastBit	描述
USART_LastBit_Disable	最后一位数据的时钟脉冲不从 SCLK 输出
USART_LastBit_Enable	最后一位数据的时钟脉冲从 SCLK 输出

例：

```
/* The following example illustrates how to configure the 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 函数USART_StructInit

Table 718. 描述了函数USART_StructInit

Table 718. 函数 USART_StructInit

函数名	USART_StructInit
函数原形	void USART_StructInit(USART_InitTypeDef* USART_InitStruct)
功能描述	把 USART_InitStruct 中的每一个参数按缺省值填入
输入参数	USART_InitStruct：指向结构 USART_InitTypeDef 的指针，待初始化
输出参数	无
返回值	无
先决条件	无
被调用函数	无

Table 719. 给出了USART_InitStruct各个成员的缺省值

Table 719. USART_InitStruct 缺省值

成员	缺省值
USART_BaudRate	9600
USART_WordLength	USART_WordLength_8b
USART_StopBits	USART_StopBits_1
USART_Parity	USART_Parity_No
USART_HardwareFlowControl	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：

```
/* The following example illustrates how to initialize a
USART_InitTypeDef structure */
USART_InitTypeDef USART_InitStructure;
USART_StructInit(&USART_InitStructure);
```

21.2.4 函数USART_Cmd

Table 720. 描述了函数USART_ Cmd

Table 720. 函数 USART_ Cmd

函数名	USART_ Cmd
函数原形	void USART_Cmd(USART_TypeDef* USARTx, FunctionalState NewState)
功能描述	使能或者失能 USART 外设
输入参数 1	USARTx：x 可以是 1，2 或者 3，来选择 USART 外设
输入参数 2	NewState: 外设 USARTx 的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable the USART1 */
USART_Cmd(USART1, ENABLE);
```

21.2.5 函数USART_ITConfig

Table 721. 描述了函数USART_ITConfig

Table 721. 函数 USART_ITConfig

函数名	USART_ITConfig
函数原形	void USART_ITConfig(USART_TypeDef* USARTx, u16 USART_IT, FunctionalState NewState)
功能描述	使能或者失能指定的 USART 中断
输入参数 1	USARTx：x 可以是 1，2 或者 3，来选择 USART 外设
输入参数 2	USART_IT：待使能或者失能的 USART 中断源 参阅 Section：USART_IT 查阅更多该参数允许取值范围
输入参数 3	NewState：USARTx 中断的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

USART_IT

输入参数 USART_IT 使能或者失能 USART 的中断。可以取下表的一个或者多个取值的组合作为该参数的值。

Table 722. USART_IT 值

USART_IT	描述
USART_IT_PE	奇偶错误中断
USART_IT_TXE	发送中断
USART_IT_TC	传输完成中断
USART_IT_RXNE	接收中断
USART_IT_IDLE	空闲总线中断
USART_IT_LBD	LIN 中断检测中断
USART_IT_CTS	CTS 中断
USART_IT_ERR	错误中断

例：

```
/* Enables the USART1 transmit interrupt */
USART_ITConfig(USART1, USART_IT_Transmit ENABLE);
```

21.2.6 函数USART_DMACmd

Table 723. 描述了函数USART_ DMACmd

Table 723. 函数 USART_ DMACmd

函数名	USART_ DMACmd
函数原形	USART_DMACmd(USART_TypeDef* USARTx, FunctionalState NewState)
功能描述	使能或者失能指定 USART 的 DMA 请求
输入参数 1	USARTx：x 可以是 1，2 或者 3，来选择 USART 外设
输入参数 2	USART_DMAreq：指定 DMA 请求 参阅 Section：USART_DMAreq 查阅更多该参数允许取值范围
输入参数 3	NewState: USARTx DMA 请求源的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

USART_DMAreq

USART_DMAreq选择待使能或者失能的DMA请求。Table 724. 给出了该参数可取的值。

Table 724. USART_LastBit 值

USART_DMAreq	描述
USART_DMAReq_Tx	发送 DMA 请求
USART_DMAReq_Rx	接收 DMA 请求

例：

```
/* Enable the DMA transfer on Rx and Tx action for USART2 */
USART_DMACmd(USART2, USART_DMAReq_Rx | USART_DMAReq_Tx, ENABLE);
```

21.2.7 函数USART_SetAddress

Table 725. 描述了函数USART_SetAddress

Table 725. 函数 USART_SetAddress

函数名	USART_SetAddress
函数原形	void USART_SetAddress(USART_TypeDef* USARTx, u8 USART_Address)
功能描述	设置 USART 节点的地址
输入参数 1	USARTx：x 可以是 1，2 或者 3，来选择 USART 外设
输入参数 2	USART_Address：提示 USART 节点的地址。
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Sets the USART2 address node to 0x5 */
USART_SetAddress(USART2, 0x5);
```

21.2.8 函数USART_WakeUpConfig

Table 726. 描述了函数USART_WakeUpConfig

Table 726. 函数 USART_WakeUpConfig

函数名	USART_WakeUpConfig
函数原形	void USART_WakeUpConfig(USART_TypeDef* USARTx, u16 USART_WakeUp)
功能描述	选择 USART 的唤醒方式
输入参数 1	USARTx：x 可以是 1，2 或者 3，来选择 USART 外设
输入参数 2	USART_WakeUp：USART 的唤醒方式 参阅 Section：USART_WakeUp 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

USART_WakeUp

USART_WakeUp选择USART的唤醒方式。Table 727. 给出了该参数可取的值。

Table 727. USART_WakeUp 值

USART_WakeUp	描述
USART_WakeUp_IdleLine	空闲总线唤醒
USART_WakeUp_AddressMark	地址标记唤醒

例：

```
/* Selects the IDLE Line as USART1 WakeUp */
USART_WakeUpConfig(USART1, USART_WakeUpIdleLine);
```

21.2.9 函数USART_ReceiverWakeUpCmd

Table 728. 描述了函数USART_ReceiverWakeUpCmd

Table 728. 函数 USART_ReceiverWakeUpCmd

函数名	USART_ReceiverWakeUpCmd
函数原形	void USART_ReceiverWakeUpCmd(USART_TypeDef* USARTx, FunctionalState Newstate)
功能描述	检查 USART 是否处于静默模式
输入参数 1	USARTx：x 可以是 1，2 或者 3，来选择 USART 外设
输入参数 2	NewState: USART 静默模式的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* USART3 in normal mode */
USART_ReceiverWakeUpCmd(USART3, DISABLE);
```

21.2.10 函数USART_LINBreakDetectiLengthConfig

Table 729. 描述了函数USART_LINBreakDetectiLengthConfig

Table 729. 函数 USART_LINBreakDetectiLengthConfig

函数名	USART_LINBreakDetectiLengthConfig
函数原形	void USART_LINBreakDetectLengthConfig(USART_TypeDef* USARTx, u16 USART_LINBreakDetectLength)
功能描述	设置 USART LIN 中断检测长度
输入参数 1	USARTx：x 可以是 1，2 或者 3，来选择 USART 外设
输入参数 2	USART_LINBreakDetectLength：LIN 中断检测长度 参阅 Section：USART_LINBreakDetectLength 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

USART_LINBreakDetectLength

USART_LINBreakDetectLength选择USART的唤醒方式。Table 730. 给出了该参数可取的值。

Table 730. USART_LINBreakDetectLength 值

USART_LINBreakDetectLength	描述
USART_LINBreakDetectLength_10b	10 位中断检测
USART_LINBreakDetectLength_11b	11 位中断检测

例：

```
/* Selects 10 bit break detection for USART1 */
USART_LINBreakDetectLengthConfig(USART1,
USART_LINDetectLength_10b);
```

21.2.11 函数USART_LINCmd

Table 731. 描述了函数USART_LINCmd

Table 731. 函数 USART_LINCmd

函数名	USART_LINCmd
函数原形	void USART_LINCmd(USART_TypeDef* USARTx, FunctionalState Newstate)
功能描述	使能或者失能 USARTx 的 LIN 模式
输入参数 1	USARTx：x 可以是 1，2 或者 3，来选择 USART 外设
输入参数 2	NewState: USART LIN 模式的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable the USART2 LIN mode */
USART_LINCmd(USART2, ENABLE);
```

21.2.12 函数USART_SendData

Table 732. 描述了函数USART_ SendData

Table 732. 函数 USART_ SendData

函数名	USART_ SendData
函数原形	void USART_SendData(USART_TypeDef* USARTx, u8 Data)
功能描述	通过外设 USARTx 发送单个数据
输入参数 1	USARTx：x 可以是 1，2 或者 3，来选择 USART 外设
输入参数 2	Data: 待发送的数据
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Send one HalfWord on USART3 */
USART_SendData(USART3, 0x26);
```

21.2.13 函数USART_ReceiveData

Table 733. 描述了函数USART_ ReceiveData

Table 733. 函数 USART_ReceiveData

函数名	USART_ ReceiveData
函数原形	u8 USART_ReceiveData(USART_TypeDef* USARTx)
功能描述	返回 USARTx 最近接收到的数据
输入参数	USARTx：x 可以是 1，2 或者 3，来选择 USART 外设
输出参数	无
返回值	接收到的字
先决条件	无
被调用函数	无

例：

```
/* Receive one halfword on USART2 */
u16 RxData;
RxData = USART_ReceiveData(USART2);
```

21.2.14 函数USART_SendBreak

Table 734. 描述了函数USART_SendBreak

Table 734. 函数 USART_SendBreak

函数名	USART_SendBreak
函数原形	void USART_SendBreak(USART_TypeDef* USARTx)
功能描述	发送中断字
输入参数	USARTx：x 可以是 1，2 或者 3，来选择 USART 外设
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Send break character on USART1 */
USART_SendBreak(USART1);
```

21.2.15 函数USART_SetGuardTime

Table 735. 描述了函数USART_SetGuardTime

Table 735. 函数 USART_SetGuardTime

函数名	USART_SetGuardTime
函数原形	void USART_SetGuardTime(USART_TypeDef* USARTx, u8 USART_GuardTime)
功能描述	设置指定的 USART 保护时间
输入参数 1	USARTx：x 可以是 1，2 或者 3，来选择 USART 外设
输入参数 2	USART_GuardTime: 指定的保护时间
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Set the guard time to 0x78 */
USART_SetGuardTime(0x78);
```

21.2.16 函数USART_SetPrescaler

Table 736. 描述了函数USART_SetPrescaler

Table 736. 函数 USART_SetPrescaler

函数名	USART_SetPrescaler
函数原形	void USART_SetPrescaler(USART_TypeDef* USARTx, u8 USART_Prescaler)
功能描述	设置 USART 时钟预分频
输入参数 1	USARTx：x 可以是 1，2 或者 3，来选择 USART 外设
输入参数 2	USART_Prescaler: 时钟预分频
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Set the system clock prescaler to 0x56 */
USART_SetPrescaler(0x56);
```

21.2.17 函数USART_SmartCardCmd

Table 737. 描述了函数USART_SmartCardCmd

Table 737. 函数 USART_SmartCardCmd

函数名	USART_SmartCardCmd
函数原形	void USART_SmartCardCmd(USART_TypeDef* USARTx, FunctionalState Newstate)
功能描述	使能或者失能指定 USART 的智能卡模式
输入参数 1	USARTx：x 可以是 1，2 或者 3，来选择 USART 外设
输入参数 2	NewState: USART 智能卡模式的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable the USART1 Smart Card mode */
USART_SmartCardCmd(USART1, ENABLE);
```

21.2.18 函数USART_SmartCardNackCmd

Table 738. 描述了函数USART_SmartCardNackCmd

Table 738. 函数 USART_SmartCardNackCmd

函数名	USART_SmartCardNackCmd
函数原形	void USART_SmartCardNACKCmd(USART_TypeDef* USARTx, FunctionalState Newstate)
功能描述	使能或者失能 NACK 传输
输入参数 1	USARTx：x 可以是 1，2 或者 3，来选择 USART 外设
输入参数 2	NewState: NACK 传输的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable the USART1 NACK transmission during parity error */
USART_SmartCardNACKCmd(USART1, ENABLE);
```

21.2.19 函数USART_HalfDuplexCmd

Table 739. 描述了函数USART_HalfDuplexCmd

Table 739. 函数 USART_HalfDuplexCmd

函数名	USART_HalfDuplexCmd
函数原形	void USART_HalfDuplexCmd(USART_TypeDef* USARTx, FunctionalState Newstate)
功能描述	使能或者失能 USART 半双工模式
输入参数 1	USARTx：x 可以是 1，2 或者 3，来选择 USART 外设
输入参数 2	NewState: USART 半双工模式传输的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enabe HalfDuplex mode for USART2 */
USART_HalfDuplexCmd(USART2, ENABLE);
```

21.2.20 函数USART_IrDAConfig

Table 740. 描述了函数 USART_IrDAConfig

Table 740. 函数 USART_IrDAConfig

函数名	USART_IrDAConfig
函数原形	void USART_IrDAConfig(USART_TypeDef* USARTx, u16 USART_IrDAMode)
功能描述	设置 USART IrDA 模式
输入参数 1	USARTx：x 可以是 1，2 或者 3，来选择 USART 外设
输入参数 2	USART_IrDAMode：LIN 中断检测长度 参阅 Section：USART_IrDAMode 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

USART_IrDAMode

USART_IrDAMode选择IrDA的模式。Table 741. 给出了该参数可取的值。

Table 741. USART_IrDAMode 值

USART_IrDAMode	描述
USART_IrDAMode_LowPower	IrDA 低功耗模式
USART_IrDAMode_Normal	IrDA 正常模式

例：

```
/* USART2 IrDA Low Power Selection */
USART_IrDAConfig(USART2,USART_IrDAMode_LowPower);
```

21.2.21 函数USART_IrDACmd

Table 742. 描述了函数USART_IrDACmd

Table 742. 函数 USART_IrDACmd

函数名	USART_IrDACmd
函数原形	void USART_IrDACmd(USART_TypeDef* USARTx, FunctionalState Newstate)
功能描述	使能或者失能 USART IrDA 模式
输入参数 1	USARTx：x 可以是 1，2 或者 3，来选择 USART 外设
输入参数 2	NewState: USART　IrDA 模式的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable the USART1 IrDA Mode */
USART_IrDACmd(USART1, ENABLE);
```

21.2.22 函数USART_GetFlagStatus

Table 743. 描述了函数USART_ GetFlagStatus

Table 743. 函数 USART_ GetFlagStatus

函数名	USART_ GetFlagStatus
函数原形	FlagStatus USART_GetFlagStatus(USART_TypeDef* USARTx, u16 USART_FLAG)
功能描述	检查指定的 USART 标志位设置与否
输入参数 1	USARTx：x 可以是 1，2 或者 3，来选择 USART 外设
输入参数 2	USART_FLAG：待检查的 USART 标志位 参阅 Section：USART_FLAG 查阅更多该参数允许取值范围
输出参数	无
返回值	USART_FLAG 的新状态（SET 或者 RESET）
先决条件	无
被调用函数	无

USART_FLAG

Table 744. 给出了所有可以被函数USART_ GetFlagStatus检查的标志位列表

Table 744. USART_FLAG 值

USART_FLAG	描述
USART_FLAG_CTS	CTS 标志位
USART_FLAG_LBD	LIN 中断检测标志位
USART_FLAG_TXE	发送数据寄存器空标志位
USART_FLAG_TC	发送完成标志位
USART_FLAG_RXNE	接收数据寄存器非空标志位
USART_FLAG_IDLE	空闲总线标志位
USART_FLAG_ORE	溢出错误标志位
USART_FLAG_NE	噪声错误标志位
USART_FLAG_FE	帧错误标志位
USART_FLAG_PE	奇偶错误标志位

例：

```
/* Check if the transmit data register is full or not */
FlagStatus Status;
Status = USART_GetFlagStatus(USART1, USART_FLAG_TXE);
```

21.2.23 函数USART_ClearFlag

Table 745. 描述了函数USART_ ClearFlag

Table 745. 函数 USART_ ClearFlag

函数名	USART_ ClearFlag
函数原形	void USART_ClearFlag(USART_TypeDef* USARTx, u16 USART_FLAG)
功能描述	清除 USARTx 的待处理标志位
输入参数 1	USARTx：x 可以是 1，2 或者 3，来选择 USART 外设
输入参数 2	USART_FLAG：待清除的 USART 标志位 参阅 Section：USART_FLAG 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear Overrun error flag */
USART_ClearFlag(USART1,USART_FLAG_OR);
```

21.2.24 函数USART_GetITStatus

Table 746. 描述了函数USART_ GetITStatus

Table 746. 函数 USART_ GetITStatus

函数名	USART_ GetITStatus
函数原形	ITStatus USART_GetITStatus(USART_TypeDef* USARTx, u16 USART_IT)
功能描述	检查指定的 USART 中断发生与否
输入参数 1	USARTx：x 可以是 1，2 或者 3，来选择 USART 外设
输入参数 2	USART_IT：待检查的 USART 中断源 参阅 Section：USART_IT 查阅更多该参数允许取值范围
输出参数	无
返回值	USART_IT 的新状态
先决条件	无
被调用函数	无

USART_IT

Table 747. 给出了所有可以被函数USART_ GetITStatus检查的中断标志位列表

Table 747. USART_IT 值

USART_IT	描述
USART_IT_PE	奇偶错误中断
USART_IT_TXE	发送中断
USART_IT_TC	发送完成中断
USART_IT_RXNE	接收中断
USART_IT_IDLE	空闲总线中断
USART_IT_LBD	LIN 中断探测中断
USART_IT_CTS	CTS 中断
USART_IT_ORE	溢出错误中断
USART_IT_NE	噪音错误中断
USART_IT_FE	帧错误中断

例：

```
/* Get the USART1 Overrun Error interrupt status */
ITStatus ErrorITStatus;
ErrorITStatus = USART_GetITStatus(USART1, USART_IT_OverrunError);
```

21.2.25 函数USART_ClearITPendingBit

Table 748. 描述了函数USART_ ClearITPendingBit

Table 748. 函数 USART_ ClearITPendingBit

函数名	USART_ ClearITPendingBit
函数原形	void USART_ClearITPendingBit(USART_TypeDef* USARTx, u16 USART_IT)
功能描述	清除 USARTx 的中断待处理位
输入参数 1	USARTx：x 可以是 1，2 或者 3，来选择 USART 外设
输入参数 2	USART_IT：待检查的 USART 中断源 参阅 Section：USART_IT 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear the Overrun Error interrupt pending bit */
USART_ClearITPendingBit(USART1,USART_IT_OverrunError);
```

22 窗口看门狗（WWDG）

窗口看门狗用来检测是否发生过软件错误。通常软件错误是由外部干涉或者不可预见的逻辑冲突引起的，这些错误将打断正常的程序流程。

Section 22.1 WWDG 寄存器结构描述了固件函数库所使用的数据结构，Section 22.2 固件库函数介绍了函数库里的所有函数。

22.1 WWDG寄存器结构

WWDG 寄存器结构，***WWDG_TypeDeff***，在文件***"stm32f10x_map.h"***中定义如下：

```
typedef struct
{
vu32 CR;
vu32 CFR;
vu32 SR;
} WWDG_TypeDef;
```

Table 749.例举了WWDG所有寄存器

Table 749. WWDG 寄存器

寄存器	描述
CR	WWDG 控制寄存器
CFR	WWDG 设置寄存器
SR	WWDG 状态寄存器

WWDG 外设声明于文件***"stm32f10x_map.h"***：

```
#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
```

使用Debug模式时，初始化指针***WWDG***于文件***"stm32f10x_lib.c"***：

```
#ifdef _WWDG
WWDG = (WWDG_TypeDef *) WWDG_BASE;
#endif /*_WWDG */
```

为了访问WWDG寄存器，***_WWDG***必须在文件***"stm32f10x_conf.h"***中定义如下：

```
#define _WWDG
```

22.2 WWDG库函数

Table 750. 例举了WWDG的库函数

Table 750. WWDG 库函数

函数名	描述
WWDG_DeInit	将外设 WWDG 寄存器重设为缺省值
WWDG_SetPrescaler	设置 WWDG 预分频值
WWDG_SetWindowValue	设置 WWDG 窗口值
WWDG_EnableIT	使能 WWDG 早期唤醒中断（EWI）
WWDG_SetCounter	设置 WWDG 计数器值
WWDG_Enable	使能 WWDG 并装入计数器值
WWDG_GetFlagStatus	检查 WWDG 早期唤醒中断标志位被设置与否
WWDG_ClearFlag	清除早期唤醒中断标志位

22.1.1 函数WWDG_DeInit

Table 751. 描述了函数 WWDG_DeInit

Table 751. 函数 WWDG_DeInit

函数名	WWDG_DeInit
函数原形	void WWDG_DeInit(WWDG_TypeDef* WWDGx)
功能描述	将外设 WWDG 寄存器重设为缺省值
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	RCC_APB1PeriphResetCmd()

例：

```
/* Deinitialize the WWDG registers */
WWDG_DeInit();
```

22.1.2 函数WWDG_SetPrescaler

Table 752. 描述了函数 WWDG_SetPrescaler

Table 752. 函数 WWDG_SetPrescaler

函数名	WWDG_SetPrescaler
函数原形	void WWDG_SetPrescaler(u32 WWDG_Prescaler)
功能描述	设置 WWDG 预分频值
输入参数	WWDG_Prescaler：指定 WWDG 预分频 参阅 Section：WWDG_Prescaler 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

WWDG_Prescaler

该参数设置 WWDG 预分频值（见 Table. 753）。

Table 753. WWDG_Prescaler 值

WWDG_Prescaler	描述
WWDG_Prescaler_1	WWDG 计数器时钟为（PCLK/4096）/1
WWDG_Prescaler_2	WWDG 计数器时钟为（PCLK/4096）/2
WWDG_Prescaler_4	WWDG 计数器时钟为（PCLK/4096）/4
WWDG_Prescaler_8	WWDG 计数器时钟为（PCLK/4096）/8

例：

```
/* Set WWDG prescaler to 8 */
WWDG_SetPrescaler(WWDG_Prescaler_8);
```

22.1.3 函数WWDG_SetWindowValue

Table 754. 描述了函数 WWDG_SetWindowValue

Table 754. 函数 WWDG_SetWindowValue

函数名	WWDG_SetWindowValue
函数原形	void WWDG_SetWindowValue(u8 WindowValue)
功能描述	设置 WWDG 窗口值
输入参数	WindowValue r：指定的窗口值。该参数取值必须在 0x40 与 0x7F 之间。
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Set WWDG window value to 0x50 */
WWDG_SetWindowValue(0x50);
```

22.1.4 函数WWDG_EnableIT

Table 755. 描述了函数 WWDG_EnableIT

Table 755. 函数 WWDG_EnableIT

函数名	WWDG_EnableIT
函数原形	void WWDG_EnableIT(void)
功能描述	使能 WWDG 早期唤醒中断（EWI）
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Enable WWDG Early wakeup interrupt */
WWDG_EnableIT();
```

22.1.5 函数WWDG_SetCounter

Table 756. 描述了函数 WWDG_SetCounter

Table 756. 函数 WWDG_SetCounter

函数名	WWDG_SetCounter
函数原形	void WWDG_SetCounter(u8 Counter)
功能描述	设置 WWDG 计数器值
输入参数	Counter：指定看门狗计数器值。该参数取值必须在 0x40 与 0x7F 之间。
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Set WWDG counter value to 0x70 */
WWDG_SetCounter(0x70);
```

22.1.6 函数WWDG_Enable

Table 757. 描述了函数 WWDG_Enable

Table 757. 函数 WWDG_Enable

函数名	WWDG_Enable
函数原形	Void WWDG_Enable(u8 Counter)
功能描述	使能 WWDG 并装入计数器值 (1)
输入参数	Counter：指定看门狗计数器值。该参数取值必须在 0x40 与 0x7F 之间。
输出参数	无
返回值	无
先决条件	无
被调用函数	无

1. WWDG 一旦被使能就不能被失能。

例：

```
/* Enable WWDG and set counter value to 0x7F */
WWDG_Enable(0x7F);
```

22.1.7 函数WWDG_GetFlagStatus

Table 758. 描述了函数WWDG_GetFlagStatus

Table 758. 函数 WWDG_GetFlagStatus

函数名	WWDG_GetFlagStatus
函数原形	FlagStatus WWDG_GetFlagStatus(void)
功能描述	检查 WWDG 早期唤醒中断标志位被设置与否
输入参数	无
输出参数	无
返回值	早期唤醒中断标志位的新状态（SET 或者 RESET）
先决条件	无
被调用函数	无

例：

```
/* Test if the counter has reached the value 0x40 */
FlagStatus Status;
Status = WWDG_GetFlagStatus();
if(Status == RESET)
{
...
}
else
{
...
}
```

22.1.8 函数WWDG_ClearFlag

Table 759. 描述了函数WWDG_ClearFlag

Table 759. 函数 WWDG_ClearFlag

函数名	WWDG_ClearFlag
函数原形	void WWDG_ClearFlag(void)
功能描述	清除早期唤醒中断标志位
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例：

```
/* Clear EWI flag */
WWDG_ClearFlag();
```

23 修订记录

参考英文版。

版权声明

参考英文版。