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。

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。

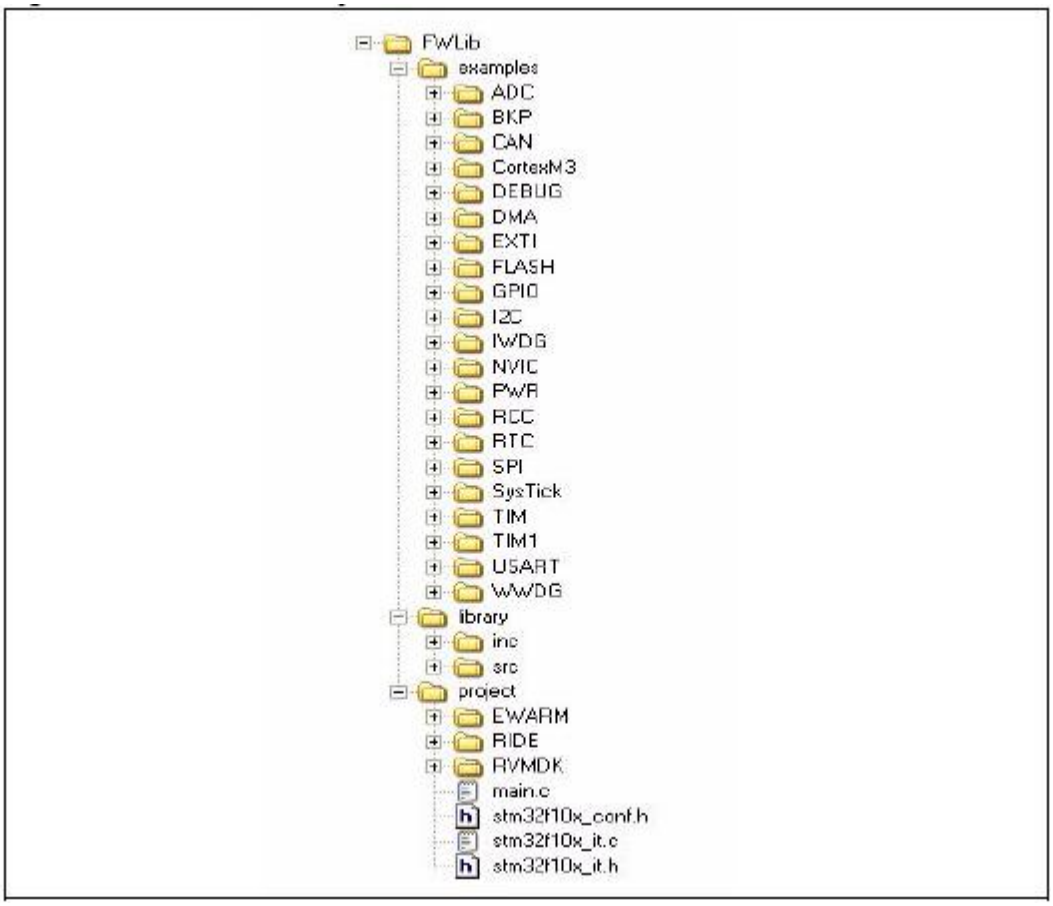
2 进入DEBUG模式会增大代码的尺寸，降低代码的运行效率。因此，我们强烈建议仅仅在除错的时候使用相应代码，在最终的应用程序中，删除它们。

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。

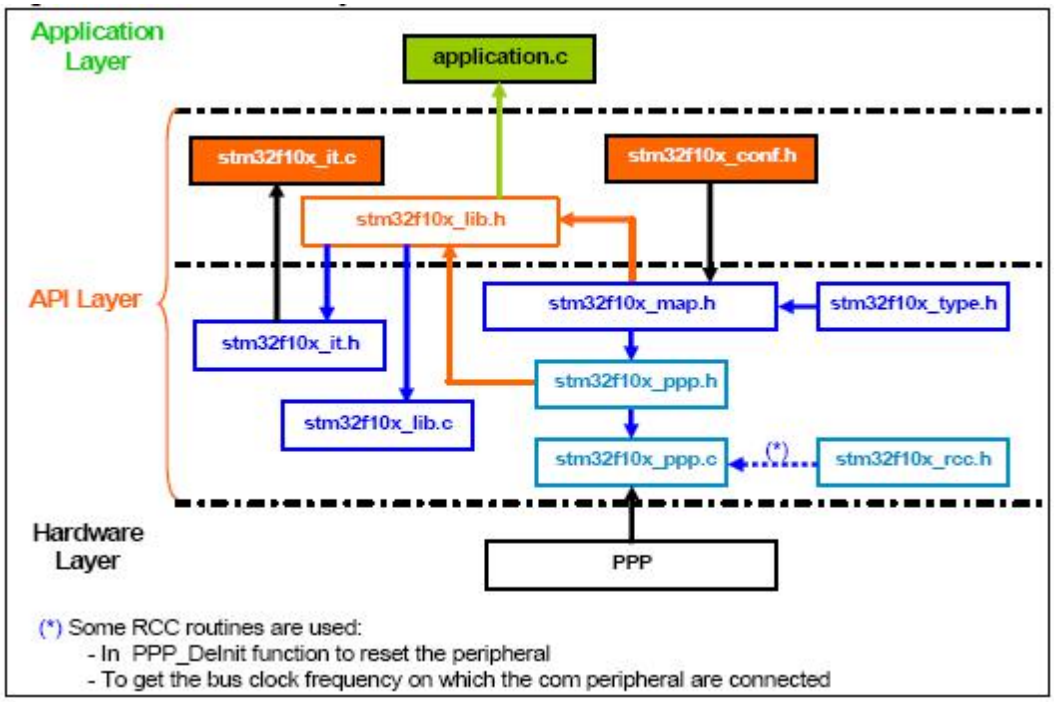
头文件stm32f10x_lib.h包含了所有外设头文件的头文件。它是唯一一个用户需要包括在自己应用中的文件，起到应用和库之间界面的作用。

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

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

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 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);
```

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	Bites																							Value
	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
System Handler_NMI(see table 298)	Reserved																		0x1F					0x1F
System Handler_HardFault (see table 299)	Reserved			0	Reserved																			0x0
System Handler_Memory Manage(see table 300)	0	0	1	0x0	0xD	0	0	Res	0x10															0x43430
System Handler_BusFault(see table 301)	1	1	1	1	0xE	1	0	Res	0x11															0x547931
System Handler_UsageFault(see table 302)	-	2	1	0x3	Reserved	2	0	Res	0x12															0x24c232
System Handler_SVCall(see table 303)	Reserved			0x7	0xF	3	1	Reserved																0x1FF40
System Handler_DebugMonitor(see table 304)	Reserved		2	0x8	Reserved	0	2	Reserved																0xA0080

System Handler_ PSV(see table 305)	Reserved	0xA	Reserved	2	2		0x1C	0x28 29C
System Handler_ SysTick(s ee table 306)	Reserved	0xB	Reserved	3	2		0x1A	0x2C 39A

Table 298. SystemHandler_NMI 定义

Table 299. SystemHandler_HardFault 定义

Table 300. SystemHandler_MemoryManage 定义

Table 301. SystemHandler_BusFault 定义

Table 302. SystemHandler_UsageFault 定义

Table 303. SystemHandler_SVCall定义

Table 304. SystemHandler_DebugMoniter 定义

Table 305. SystemHandler_PSV 定义

Table 306. SystemHandler_Systick 定义

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);
```

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
{
...
}
```

23 修订记录

参考英文版。

版权声明

参考英文版。