

## 第7章 外部中断/事件控制器

7.1 简介

7.2 典型应用步骤及常用库函数

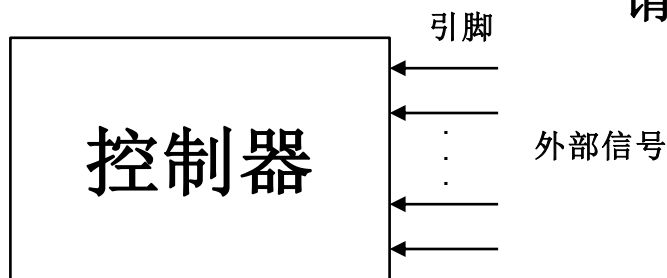
7.3 应用实例

---

# 7.1 简介

# 7.1 简介

## 7.1.1 概述



通过外部信号触发一个中断，从而启动一个对应于外部信号的服务。

- 1、STM32F4的每个IO都可以作为外部中断输入。
- 2、STM32F4的中断控制器支持23个外部中断/事件请求：

**EXTI线0~15：对应外部IO口的输入中断。**

**EXTI线16：**连接到PVD输出。

**EXTI线17：**连接到RTC闹钟事件。

**EXTI线18：**连接到USB OTG FS唤醒事件。

**EXTI线19：**连接到以太网唤醒事件。

**EXTI线20：**连接到USB OTG HS(在FS中配置)唤醒事件。

**EXTI线21：**连接到RTC入侵和时间戳事件。

**EXTI线22：**连接到RTC唤醒事件。

- 3、每个外部中断线可以独立的配置触发方式（上升沿，下降沿或者双边沿触发），触发/屏蔽，专用的状态位。

# 7.1 简介

## 7.1.2 结构

### 1. 中断请求

请求信号通过图7-1中的①②③④⑤的路径向NVIC产生中断请求。

对于每个中断线，我们可以设置相应的触发方式（**上升沿触发**，**下降沿触发**，**边沿触发**）以及使能。

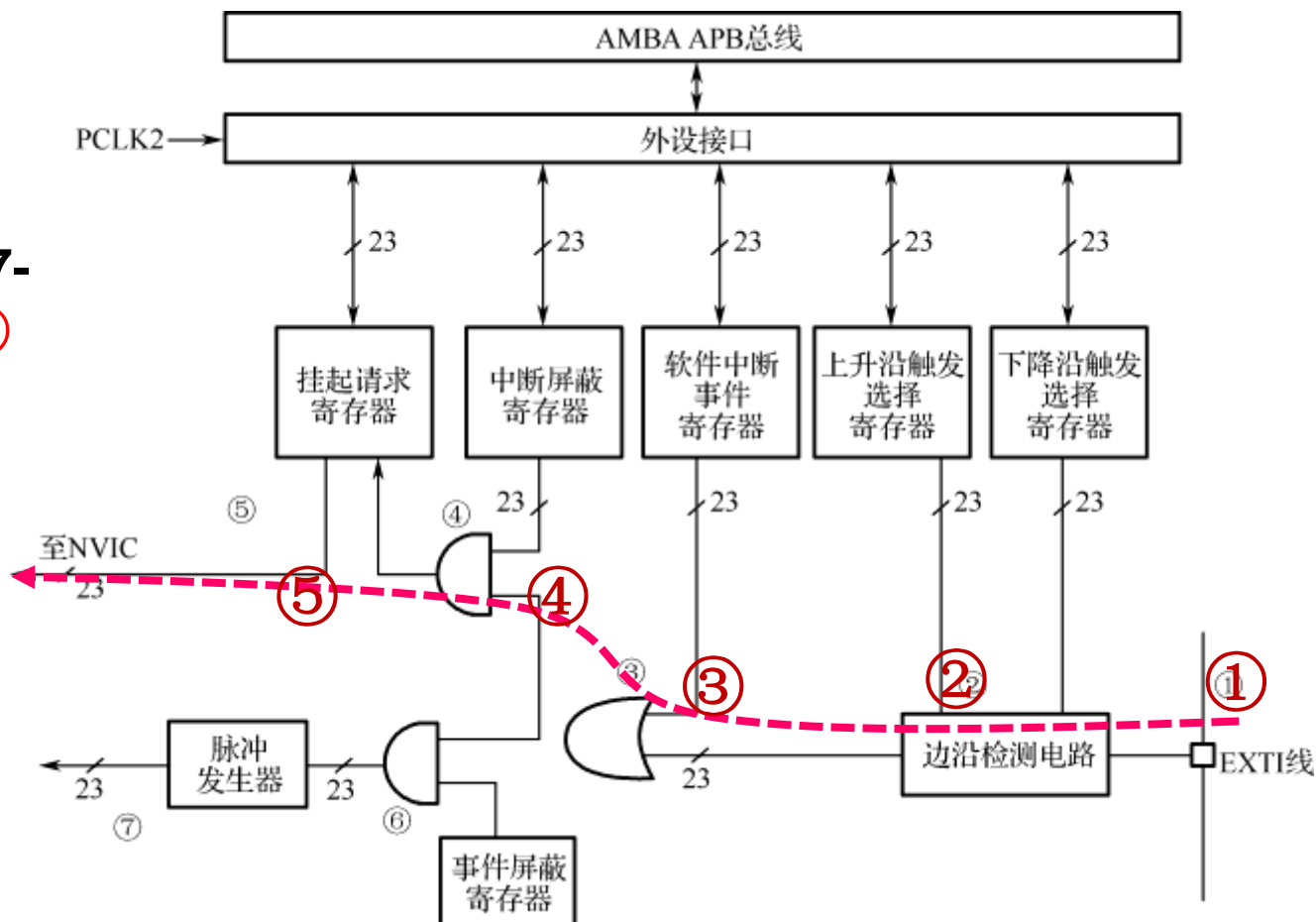


图7-1 EXTI的结构图

# 7.1 简介

## 7.1.2 结构

### 2、触发事件

请求信号通过图7-1中的①②③⑥⑦的路径产生触发事件。

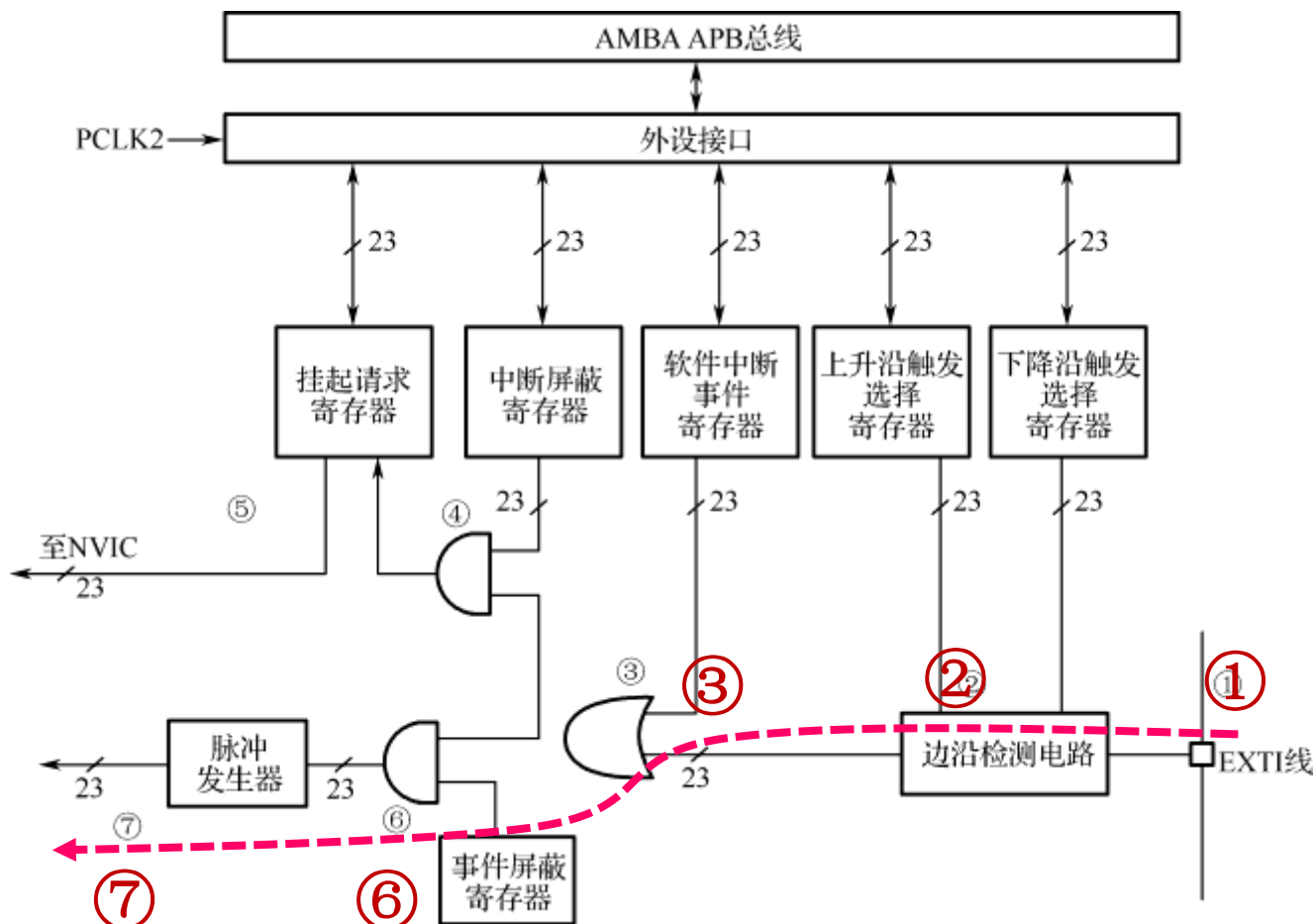


图7-1 EXTI的结构图

# 7.1 简介

## 7.1.3 GPIO相关EXTI线

**GPIOx.0映射到EXTI0**

**GPIOx.1映射到EXTI1**

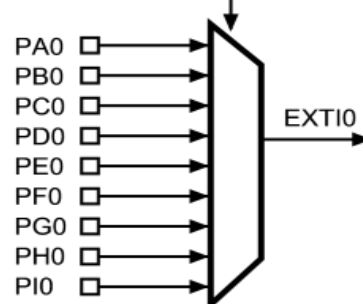
...

**GPIOx.15映射到EXTI15**

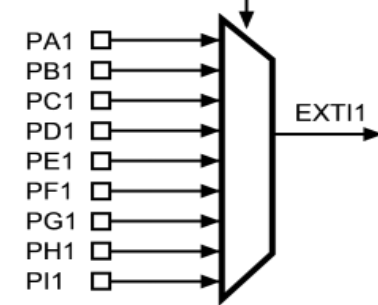
另外七根 EXTI 线连接方式如下：

- EXTI 线 16 连接到 PVD 输出
- EXTI 线 17 连接到 RTC 闹钟事件
- EXTI 线 18 连接到 USB OTG FS 唤醒事件
- EXTI 线 19 连接到以太网唤醒事件
- EXTI 线 20 连接到 USB OTG HS（在 FS 中配置）唤醒事件
- EXTI 线 21 连接到 RTC 入侵和时间戳事件
- EXTI 线 22 连接到 RTC 唤醒事件

SYSCFG\_EXTICR1 寄存器中的 EXTI0[3:0] 位

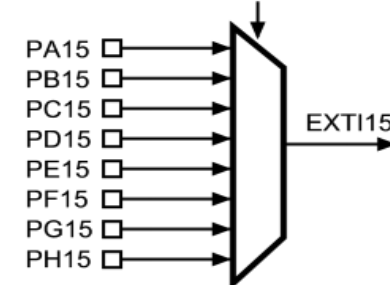


SYSCFG\_EXTICR1 寄存器中的 EXTI1[3:0] 位



⋮

SYSCFG\_EXTICR4 寄存器中的 EXTI15[3:0] 位



# 7.1 简介

## 外部中断服务

- 1、16个中断线**没有**对应16个独立的中断服务；
- 2、IO口外部中断在中断向量表中只分配了7个中断向量，也就是只能使用7个中断服务函数

位置	优先级	优先级类型	名称	说明	地址
6	13	可设置	EXTI0	EXTI 线 0 中断	0x0000 0058
7	14	可设置	EXTI1	EXTI 线 1 中断	0x0000 005C
8	15	可设置	EXTI2	EXTI 线 2 中断	0x0000 0060
9	16	可设置	EXTI3	EXTI 线 3 中断	0x0000 0064
10	17	可设置	EXTI4	EXTI 线 4 中断	0x0000 0068
23	30	可设置	EXTI9_5	EXTI 线 [9:5] 中断	0x0000 009C
40	47	可设置	EXTI15_10	EXTI 线 [15:10] 中断	0x0000 00E0

- 3、外部中断线**5~9**分配一个中断向量，共用一个服务函数；  
外部中断线**10~15**分配一个中断向量，共用一个中断服务函数。

# 7.1 简介

---

## 外部中断服务

### 4、中断服务函数列表：

```
EXTI0_IRQHandler  
EXTI1_IRQHandler  
EXTI2_IRQHandler  
EXTI3_IRQHandler  
EXTI4_IRQHandler  
EXTI9_5_IRQHandler  
EXTI15_10_IRQHandler
```



---

## 7.2 典型应用步骤及常用库函数

# 7.2典型应用步骤及常用库函数

## 7.2.1 典型应用步骤

外部中断初始化程序

- ① 使能GPIO口的时钟、使能SYSCFG时钟：  
`RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);`
- ② 初始化IO口为输入。  
`GPIO_Init();`
- ③ 设置IO口与中断线的映射关系。  
`void SYSCFG_EXTILineConfig();`
- ④ 初始化线上中断，设置触发条件等。  
`EXTI_Init();`
- ⑤ 配置中断分组（NVIC），并使能中断。  
`NVIC_Init();`

中断服务程序

- ⑥ 编写中断服务函数。  
`EXTIx_IRQHandler();`  
响应中断后在中断服务程序中要先判断中断来源，  
`EXTI_GetITStatus();`  
然后响应中断后在中断服务程序中要 清除中断标志位  
`EXTI_ClearITPendingBit();`
- ⑦ 编写中断服务程序处理内容。

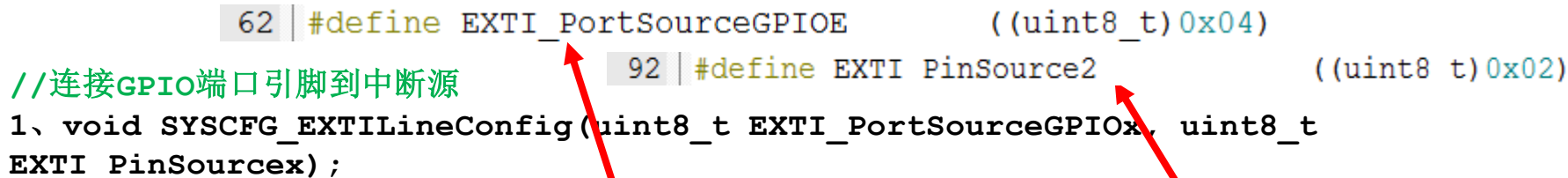
简单的可以在  
`EXTIx_IRQH  
andler();`  
内实现，复杂  
的设置标志位  
后在主程序实  
现。

# 7.2典型应用步骤及常用库函数

## 7.2.2 常用库函数

stm32f4xx\_exti.h  
stm32f4xx\_exti.c

```
62 | #define EXTI_PortSourceGPIOE      ((uint8_t)0x04)
//连接GPIO端口引脚到中断源
1、 void SYSCFG_EXTILineConfig(uint8_t EXTI_PortSourceGPIOx, uint8_t
EXTI_PinSourcex);
92 | #define EXTI_PinSource2          ((uint8_t)0x02)
```



例: `SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOE, EXTI_PinSource2);`

//初始化中断线: 选择中断源、中断模式、触发方式、使能等

2、 `void EXTI_Init(EXTI_InitTypeDef* EXTI_InitStruct);`

//判断中断线中断状态, 是否发生

3、 `ITStatus EXTI_GetITStatus(uint32_t EXTI_Line);`

//清除中断线上的中断标志位

4、 `void EXTI_ClearITPendingBit(uint32_t EXTI_Line);`

//这个函数非常重要, 在使用外部中断的时候一定要先使能SYSCFG时钟//设置IO口与中断线的映射关系

5、 `RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);`//使能SYSCFG时钟

# 7.2 典型应用步骤及常用库函数

## 7.2.2 常用库函数

```
void EXTI_Init(EXTI_InitTypeDef* EXTI_InitStruct);
```

```
typedef struct
{
    uint32_t EXTI_Line;      //指定要配置的中断线
    EXTI_Mode_TypeDef EXTI_Mode; //模式：事件 OR 中断
    EXTI_Trigger_TypeDef EXTI_Trigger; //触发方式：上升沿/下降沿/双沿触发
    FunctionalState EXTI_LineCmd; //使能 OR 失能
}EXTI_InitTypeDef; 105 #define EXTI_Line0 ((uint32_t)0x000001)
106 #define EXTI_Line1 ((uint32_t)0x000002)
107 #define EXTI_Line2 ((uint32_t)0x000004)
```

Stm32f4xx\_exti.h

例：

```
EXTI_InitStructure.EXTI_Line =EXTI_Line2;
EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
EXTI_InitStructure.EXTI_LineCmd = ENABLE;
EXTI_Init(&EXTI_InitStructure);
```

```
66 typedef enum
67 {
68     EXTI_Trigger_Rising = 0x08,
69     EXTI_Trigger_Falling = 0x0C,
70     EXTI_Trigger_Rising_Falling = 0x10
71 }EXTI_Trigger_TypeDef;
```

```
54 typedef enum
55 {
56     EXTI_Mode_Interrupt = 0x00,
57     EXTI_Mode_Event = 0x04
58 }EXTI_Mode_TypeDef;
```

---

## 7.3 应用实例

## 7.3 应用实例

要求：

按照图7-3中的控制电路，编写程序将PA0配置为外部中断输入，将中断输入触发方式配置为上升沿触发，并编写中断服务程序，实现LED灯的控制。检测到按键动作将LED灯状态反转（忽略去抖动操作）。

### 1. 编程要点

(1) 使能GPIOA时钟和SYSCFG时钟。调用函数：

`RCC_AHB1PeriphClockCmd();`

(2) 初始化PA0为输入模式。

(3) 设置EXTI线0的工作模式、触发方式和使能。

(4) 设置NVIC中断组和EXTI线0中断通道的优先级和使能状态。

(5) 编写中断服务程序。需要判断中断线状态，在确认后，清除中断挂起标志。 添加一个函数

`RCC_APB2PeriphClockCmd();`

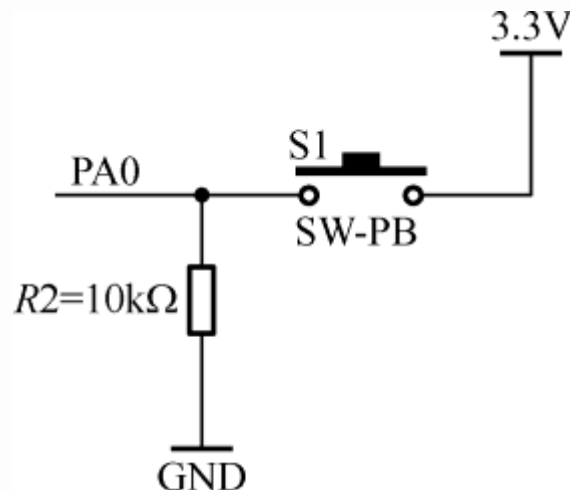


图7-3 外部中断触发电路图

# 10.3 外部中断应用

## 2. 主程序

```
int main(void)
{
    /* LED 端口初始化 */
    LED_Config();
    /* 初始化EXTI中断，按下按键会触发中断，
     * 触发中断会进入stm32f4xx_it.c文件中的函数，反转LED灯。
     */
    EXTI_Key_Config();
    /* 等待中断，由于使用中断方式，不用扫描按键 */
    while(1)
    {
    }
}
```

在函数中，首先初始化LED灯的GPIO引脚，然后初始化EXTI中断相关内容，**最后在while(1)的空循环中等待中断。**

# 10.3 外部中断应用

## 3. EXTI中断初始化函数

```
void EXTI_Key_Config(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    EXTI_InitTypeDef EXTI_InitStructure;
    //-----第一步-----
    /*开启按键GPIO口的时钟*/ ①
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA ,ENABLE);
    /* 使能 SYSCFG 时钟 ， 使用GPIO外部中断时必须使能SYSCFG时钟*/
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
    //-----第二步-----
    /* 选择按键1的引脚 */ ②
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
    /* 设置引脚为输入模式 */
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
    /* 设置引脚不上拉也不下拉 */
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
    /* 使用上面的结构体初始化按键 */
    GPIO_Init(GPIOA, &GPIO_InitStructure);
    //-----第三步-----
    /* 连接 EXTI 中断源 到key1引脚 */ ③
    SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA,EXTI_PinSource0);
}
```



# 10.3 外部中断应用

## 3. EXTI中断初始化函数

```
//-----第四步-----  
/* 选择 EXTI 中断源 */  
EXTI_InitStructure.EXTI_Line = EXTI_Line0;  
/* 中断模式 */  
EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;  
/* 上升沿触发 */  
EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;  
/* 使能中断/事件线 */  
EXTI_InitStructure.EXTI_LineCmd = ENABLE;  
EXTI_Init(&EXTI_InitStructure);  
//-----第五步-----  
/*配置NVIC为优先级组1，保证整个程序使用的中断处于同一组*/  
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);  
/*配置中断源：按键1*/  
NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn;  
/*配置抢占优先级：1*/  
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority=1;  
/*配置响应优先级：1*/  
NVIC_InitStructure.NVIC_IRQChannelSubPriority=1;  
/*使能中断通道*/  
NVIC_InitStructure.NVIC_IRQChannelCmd=ENABLE;  
NVIC_Init(&NVIC_InitStructure);  
//-----其他的外部中断 配置过程一样-----  
} //初始化结束
```

④

⑤

# 10.3 外部中断应用

## 4. 中断服务程序

中断服务程序的名称一定要和启动文件 `startup_stm32f429_439xx.s` 中定义的保持一致。

```
void EXTI0_IRQHandler(void)
{
    //确保是否产生了EXTI Line中断
    if (EXTI_GetITStatus(EXTI_Line0) != RESET)
    {
        //清除中断标志位，否则难以响应后面的中断
        EXTI_ClearITPendingBit(EXTI_Line0);
        // LED 取反
        LED_TOGGLE;
    }
}
```

⑥

⑦ 响应中断的目的

实现功能比较简单，直接  
在服务程序内实现

中断服务程序实在响应中断后，自动调用的，可以放置于工程的任何地方。

; Vector Table Mapped to Address 0 at Reset

```
AREA      RESET, DATA, READONLY
EXPORT   __Vectors
EXPORT   __Vectors_End
EXPORT   __Vectors_Size
```

```
__Vectors  DCD      __initial_sp           ; Top of Stack
           DCD      Reset_Handler         ; Reset Handler
           DCD      NMI_Handler           ; NMI Handler
           DCD      HardFault_Handler     ; Hard Fault Handler
           DCD      MemManage_Handler     ; MPU Fault Handler
           DCD      BusFault_Handler      ; Bus Fault Handler
           DCD      UsageFault_Handler    ; Usage Fault Handler
           DCD      0                     ; Reserved
           DCD      0                     ; Reserved
           DCD      0                     ; Reserved
           DCD      SVC_Handler           ; SVC Call Handler
           DCD      DebugMon_Handler     ; Debug Monitor Handler
           DCD      0                     ; Reserved
           DCD      PendSV_Handler        ; PendSV Handler
           DCD      SysTick_Handler       ; SysTick Handler
```

; External Interrupts

```
DCD      WWDG_IRQHandler                 ; Window WatchDog
DCD      PVD_IRQHandler                 ; PVD through EXTI Line detection
DCD      TAMP_STAMP_IRQHandler          ; Tamper and TimeStamps through the EXTI line
DCD      RTC_WKUP_IRQHandler            ; RTC Wakeup through the EXTI line
DCD      FLASH_IRQHandler               ; FLASH
DCD      RCC_IRQHandler                 ; RCC
DCD      EXTI0_IRQHandler               ; EXTI Line0
DCD      EXTI1_IRQHandler               ; EXTI Line1
DCD      EXTI2_IRQHandler               ; EXTI Line2
DCD      EXTI3_IRQHandler               ; EXTI Line3
DCD      EXTI4_IRQHandler               ; EXTI Line4
```

```
EXPORT   WWDG_IRQHandler                [WEAK]
EXPORT   PVD_IRQHandler                 [WEAK]
EXPORT   TAMP_STAMP_IRQHandler           [WEAK]
EXPORT   RTC_WKUP_IRQHandler             [WEAK]
EXPORT   FLASH_IRQHandler                [WEAK]
EXPORT   RCC_IRQHandler                 [WEAK]
EXPORT   EXTI0_IRQHandler                [WEAK]
EXPORT   EXTI1_IRQHandler                [WEAK]
EXPORT   EXTI2_IRQHandler                [WEAK]
EXPORT   EXTI3_IRQHandler                [WEAK]
EXPORT   EXTI4_IRQHandler                [WEAK]
```

分配一个字单元，用于存储中断服务函数的入口地址，当CPU相应中断时，直接将这个单元中的地址放在PC中。

[WEAK] 是弱定义的意思，如果外部定义了，优先执行外部定义，否则下面的函数定义。

也就是说中断服务可以在其他文件中实现。

与上面定义的外部中断0服务程序（C语言）函数名一致。

```
void EXTI0_IRQHandler(void)
{
```