# Week4-3 EXTI

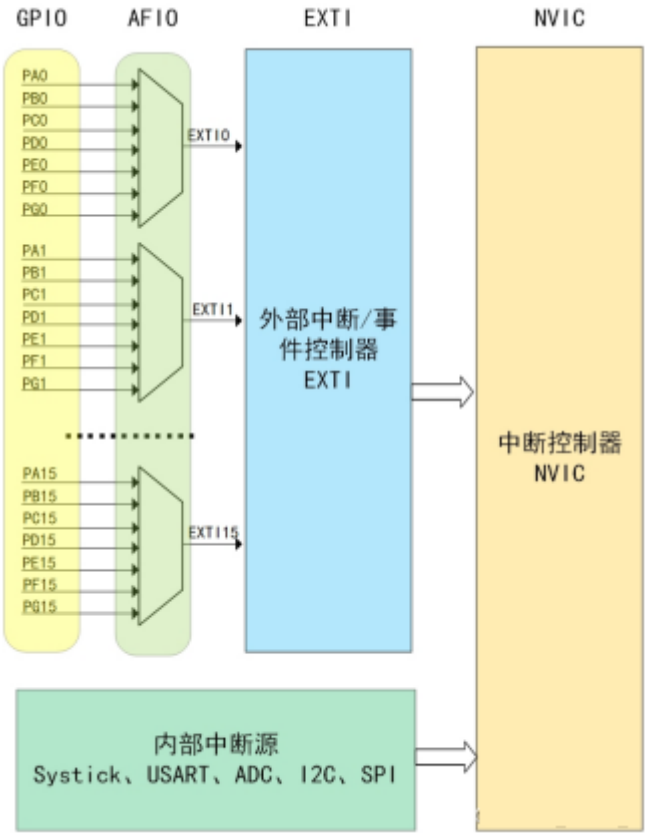
## 一、EXTI基本概念

### 1.1什么是EXTI

**EXTI（External interrupt/event controller）是外部中断/事件控制器**，管理了控制器的19/20个中断/事件线（STM32F103 系列的 EXTI 支持 19 个外部中断 / 事件请求，互联型系列的 STM32 支持 20 个）。每个中断/事件线都对应有一个边沿检测器，可以实现输入信号的上升沿检测和下降沿的检测。 EXTI 可以实现对每个中断/事件线进行单独配置，可以单独配置为**中断**或者**事件**，以及触发事件的属性。

### 1.2EXTI与NVIC

可以这样理解，**传入EXTI的仅仅是一个信号，EXTI的功能就是根据信号传入的"线"对信号做出相应的处理，然后将处理后的信号转向NVIC。** 就像一个分拣机器，传入的东西经过筛选处理被送往不同的地方，只是EXTI分拣的是信号罢了。 如果说NVIC是配置中断源，那么EXTI就是向NVIC传送中断信号。结合 *下图*，来看EXTI与NVIC的关系。




### 1.3EXTI与GPIO

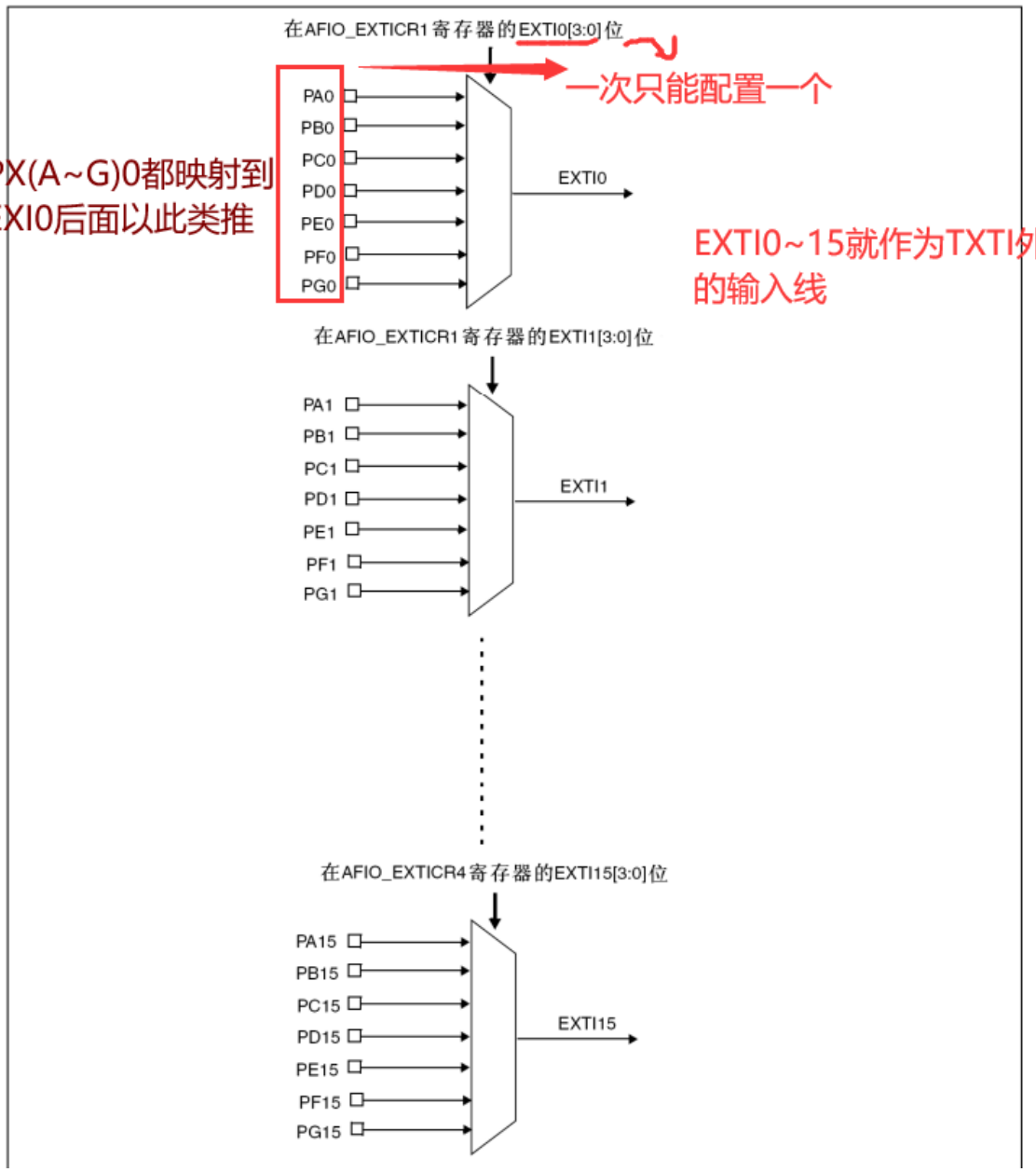
**STM32的所有GPIO都引入到EXTI 外部中断线上，使得所有的GPIO 都能作为外部中断的输入源。**EXTI支持配置20个中断和事件屏蔽位；

- GPIO端口以下图的方式连接到16个外部中断/事件线上；EXTI_Line0 — EXTI_Line15；
- EXTI_Line16 连接到PVD输出；
- EXTI_Line17连接到RTC闹钟事件;

- EXTI_Line18连接到USB唤醒事件;
- EXTI_Line19连接到以太网唤醒事件(只适用于互联型产品);

112通用I/O端口以下图的方式连接到16个外部中断/事件线上：

图20　外部中断通用I/O映像




注意：PAx~PGx端口的中断事件都连接到了EXTIx，**即同一时刻EXTx只能相应一个端口的事件触发，不能够同一时间响应所有GPIO 端口的事件，但可以分时复用。**

AFIO (alternate-function I/O)，指GPIO端口的复用功能。

当把GPIO 用作EXTI 外部中断或使用重映射功能的时候，**必须开启AFIO时钟**，而在使用默认复用功能的时候，就不必开启AFIO 时钟了。端口复用和重映射的概念均在*Week4-1GPIO*中提到过~

**在EXTI中，有三种触发中断的方式：上升沿触发，下降沿触发，双边沿触发。**根据不同的电路，我们选择不同的触发方式，以确保中断能够被正常触发。

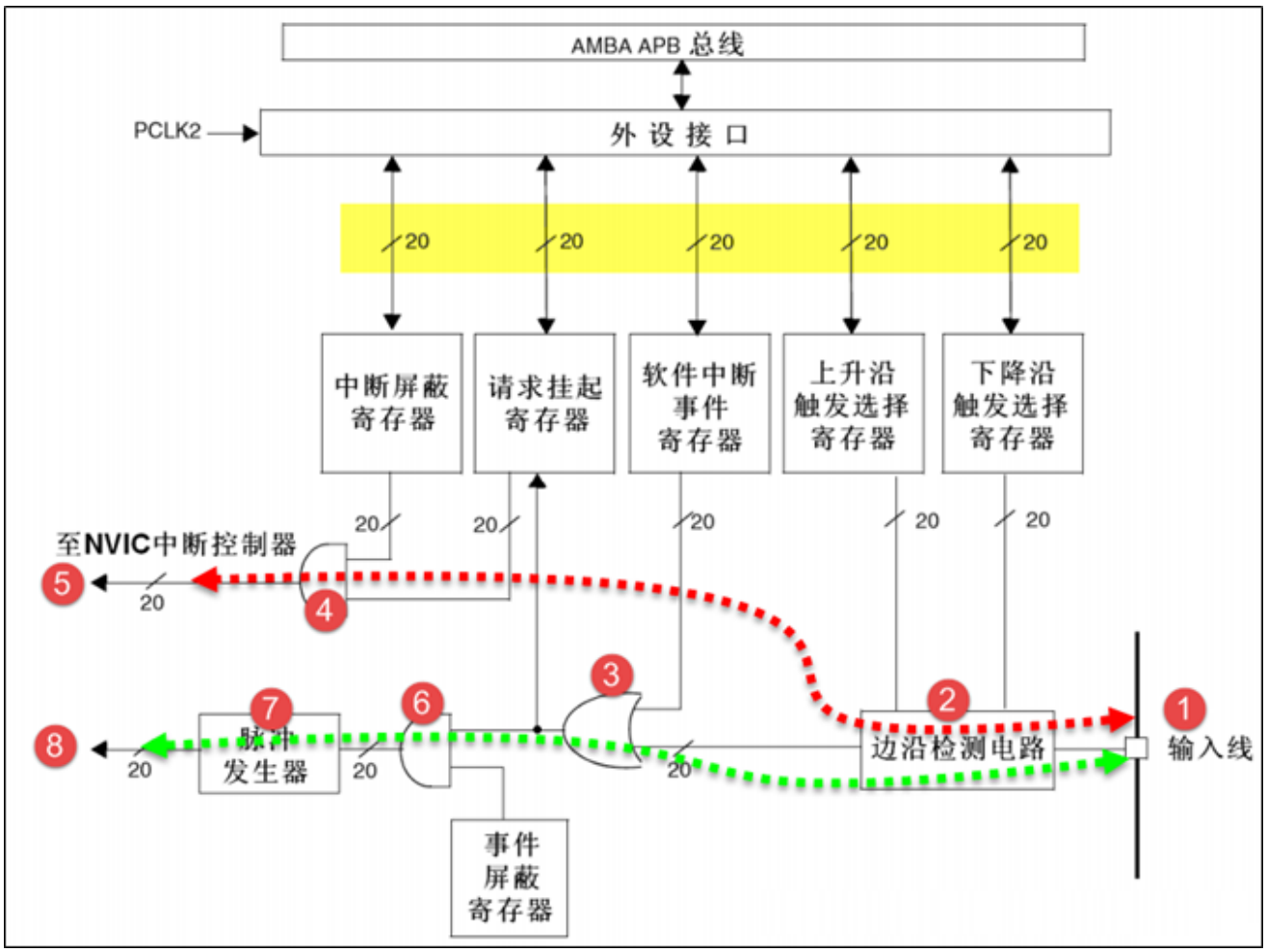
## 1.3EXTI功能框图

我们需要知道，EXTI有以下两种模式：

**中断模式：**是指外部信号产生电平变化时， EXTI 将该信号给 NVIC 处理，从而触发中断，执行中断服务函数，完成对应操作。

**事件模式：**是指外部信号产生电平变化时，EXTI 根据配置，联动 ADC 或 TIM 执行相关操作。

**EXTI功能框图：**




图中，20/代表着有20条相同的线路。**中断模式是红色线路1-2-4-5，事件模式是绿色线路1-2-3-6-7-8。**可以看到，外部中断的功能可以配置六个寄存器：

- 中断屏蔽寄存器（EXTI_IMR）
- 事件屏蔽寄存器（EXTI_EMR）
- 上升沿触发选择寄存器（EXTI_RTSR）
- 下降沿触发选择寄存器（EXTI_FTSR）
- 软件中断事件寄存器（EXTI_SWIER）
- 挂起寄存器（EXTI_PR）

它们的说明请移步Document-*STM32中文参考手册_V10.pdf* p138-140，在此不做展开。

想进一步了解框图原理可以移步链接 中断-NVIC与EXTI外设详解(超全面) STM32系统学习——EXTI 和 【STM32】NVIC EXTI 外部中断详解【原理】 。

# 二、EXTI配置

## 2.1EXTI常用库函数

- ```
  void GPIO_EXTILineConfig(uint8_t GPIO_PortSource, uint8_t GPIO_PinSource);
  //设置IO口与中断线的映射关系
  //exp: GPIO_EXTILineConfig(GPIO_PortSourceGPIOE,GPIO_PinSource2);
  ```

- ```
  void EXTI_Init(EXTI_InitTypeDef* EXTI_InitStruct);
  //初始化中断线：触发方式等
  typedef struct
  {
    uint32_t EXTI_Line;   //指定要配置的中断线
    EXTIMode_TypeDef EXTI_Mode;   //模式：事件 OR中断
    EXTITrigger_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;
   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);
  ```

- ```
  ITStatus EXTI_GetITStatus(uint32_t EXTI_Line);
  //判断中断线中断状态，是否发生
  ```

- ```
  void EXTI_ClearITPendingBit(uint32_t EXTI_Line);
  //清除中断线上的中断标志位
  ```

## 2.2中断服务函数

为了便于理解，这里我们来看中断配置代码。

```
void NVIC_Configuration(void)
{
  NVIC_InitTypeDef NVIC_InitStructure;

  /* 配置NVIC为优先级组1 */
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);

  /* 配置中断源：按键1 */
  NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn;  //配置为EXTI0通道
  /* 配置抢占优先级 */
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
  /* 配置子优先级 */
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
  /* 使能中断通道 */
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure); //将上述配置参数传入中断初始化函数
}
```

除了中断线的配置，我们还要配置对应引脚。

```
void EXTI_Key_Config(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;
  EXTI_InitTypeDef EXTI_InitStructure;

  /*开启按键GPIO口的时钟*/
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_AFIO,ENABLE);

  /* 配置 NVIC 中断*/
  NVIC_Configuration();

/*--------------------------KEY1配置-----------------------------*/
  /* 选择按键用到的GPIO */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
  /* 配置为浮空输入 */
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(KEY1_INT_GPIO_PORT, &GPIO_InitStructure);

  /* 选择EXTI的信号源 */
  GPIO_EXTILineConfig(GPIO_PortSourceGPIOA, GPIO_PinSource0);
  EXTI_InitStructure.EXTI_Line = EXTI_Line0;

  /* EXTI为中断模式 */
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
  /* 上升沿中断 */
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
  /* 使能中断 */
  EXTI_InitStructure.EXTI_LineCmd = ENABLE;
  EXTI_Init(&EXTI_InitStructure);
}
```

至此，中断的配置完毕。相信你已经看出来，上述代码是**将PA0配置为上升沿中断。**不过，现在只能够说该中断已经配置完毕，但我们还不能使用它。我们还缺少它的**中断服务函数。**

我们在*1.3EXTI与GPIO*一节中知道，GPIO连接到16个外部中断/事件线上。那么是不是16个中断线就可以分配16个中断服务函数呢?

——实际上，**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**，如下表所示：

| 位置 | 优先级 | 优先级类型 | 名称 | 说明 | 地址 |
|---|---|---|---|---|---|
| 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 |

从表中可以看出，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。

下面是中断服务函数列表：

- EXTI0_IRQHandler
- EXTI1_IRQHandler
- EXTI2_IRQHandler
- EXTI3_IRQHandler
- EXTI4_IRQHandler
- EXTI9_5_IRQHandler
- EXTI15_10_IRQHandler

当中断被触发后，程序要马上跳转到中断函数去执行中断操作，这个函数在工程创建时默认时没有的。需要你自己去添加。而且需要注意的是，中断函数的名称必须是由标准库提供的，否则无法识别。

我们打开startup_stm32f10x_hd.s这个文件，在里面能找到这么一段代码：

```
  ; External Interrupts
  DCD    WWDG_IRQHandler          ; Window Watchdog
  DCD    PVD_IRQHandler           ; PVD through EXTI Line detect
  DCD    TAMPER_IRQHandler        ; Tamper
  DCD    RTC_IRQHandler           ; RTC
  DCD    FLASH_IRQHandler         ; Flash
  DCD    RCC_IRQHandler           ; RCC
  DCD    EXTI0_IRQHandler         ; EXTI Line 0
  DCD    EXTI1_IRQHandler         ; EXTI Line 1
  DCD    EXTI2_IRQHandler         ; EXTI Line 2
  DCD    EXTI3_IRQHandler         ; EXTI Line 3
  …
  …
  …
```

不难看出，EXTI0_IRQHandler 就是中断线0的中断函数，所以，我们把这个函数添加到工程中即可。最好添加到stm32f10x_it.c 这个文件中，方便管理。

可以在这个函数中添加想要的功能，代码如下：

```
void EXTI0_IRQHandler(void)
{
 //确保是否产生了EXTI Line中断
  if(EXTI_GetITStatus(EXTI_Line0) != RESET)
  {
     /******/
      //LED闪烁相关代码
     /******/
  //清除中断标志位
     EXTI_ClearITPendingBit(EXTI_Line0);
  }
}
```

## 2.3EXTI配置步骤

1. 初始化IO口为输入。 `GPIO_Init();`
2. 开启IO口复用时钟。 `RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);`
3. 设置IO口与中断线的映射关系。 `void GPIO_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();`
6. 编写中断服务函数。 `EXTIx_IRQHandler();`
7. 清除中断标志位。 `EXTI_ClearITPendingBit();`

# 三、EXTI实验

下面通过一个实验来帮助大家更好理解~来源：[中断-NVIC与EXTI外设详解(超全面)](#)

## 3.1实验任务

利用按键产生一个下降沿，让系统产生一个中断，执行中断服务函数，函数将GPIO电平翻转使得灯，按一下亮按一下灭。

## 3.2实验原理

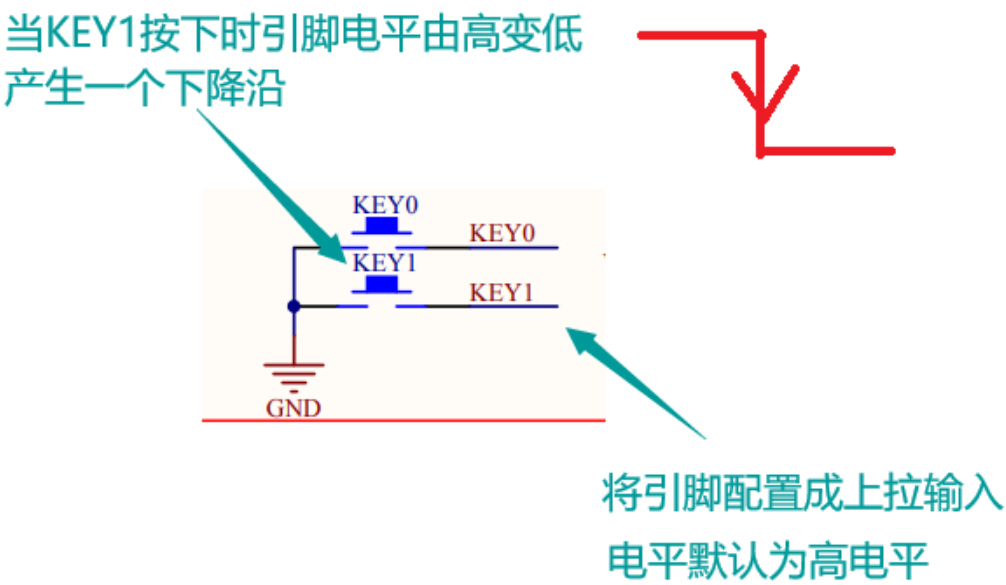
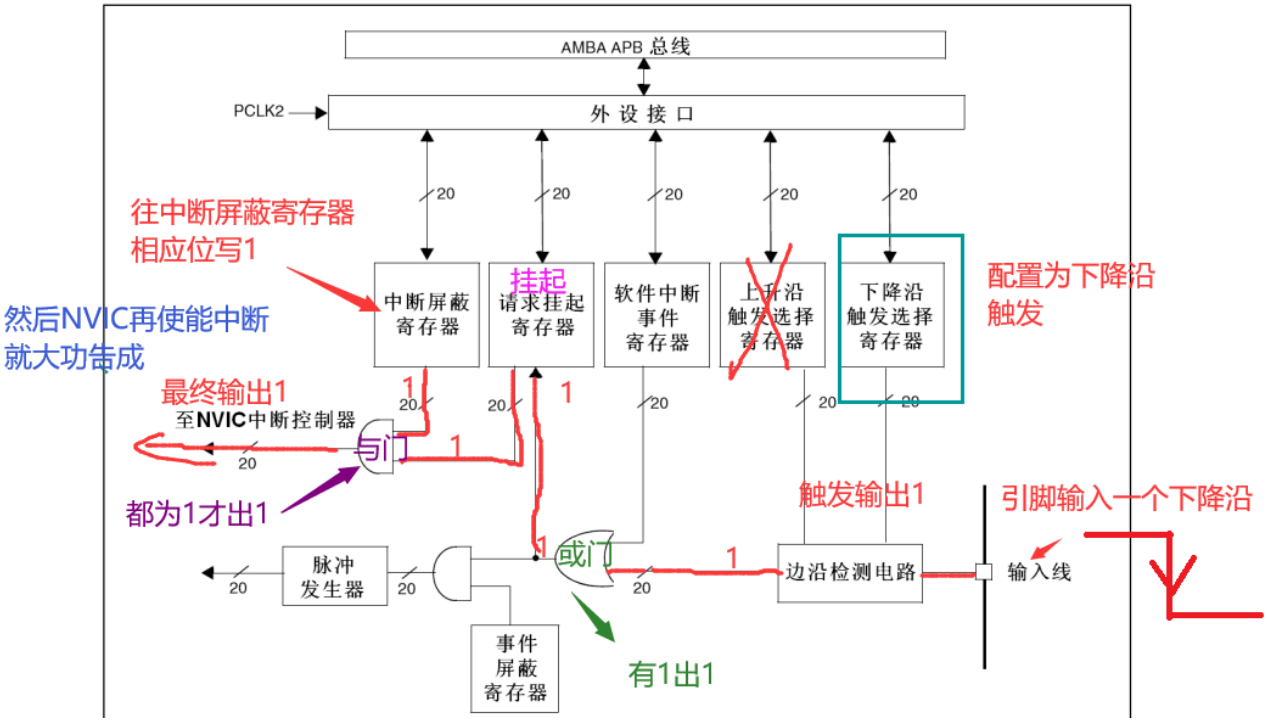


图19　外部中断/事件控制器框图




## 3.3编程要点

1. 初始化用来产生中断的 GPIO；
2. 初始化 EXTI；
3. 配置 NVIC；
4. 编写中断服务函数；

## 3.4参考代码

**exti.c**

```
#include "exti.h"

static void NVIC_EXTI_Config(void)
{
   NVIC_InitTypeDef  NVIC_InitStruct;
   //中断优先级分组这里是组1
   NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
   //选择中断源
   NVIC_InitStruct.NVIC_IRQChannel= EXTI15_10_IRQn;
   //设置抢占式优先级
   NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority =1;
   //设置响应式优先级
   NVIC_InitStruct.NVIC_IRQChannelSubPriority =1;
   //使能中断源
   NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
   //调用NVIC初始化函数
   NVIC_Init(&NVIC_InitStruct);
}
```

```
void EXTI_Key_Config(void)
{
    GPIO_InitTypeDef  GPIO_InitStruct;
    EXTI_InitTypeDef  EXTI_InitStruct;

    RCC_APB2PeriphClockCmd(EXTI_Key1_GPIO_CLK,ENABLE);
    //上拉输入
    GPIO_InitStruct.GPIO_Mode=GPIO_Mode_IPU;
    GPIO_InitStruct.GPIO_Pin= EXTI_Key1_GPIO_PIN ;
  GPIO_Init(EXTI_Key1_GPIO_POTR,&GPIO_InitStruct);
    //配置NVIC中断
    NVIC_EXTI_Config();
  //一定要使能外设AFIO外设的时钟
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);
    //选择信号源
    GPIO_EXTILineConfig(GPIO_PortSourceGPIOA,GPIO_PinSource15);
    //选择中断线
    EXTI_InitStruct.EXTI_Line = EXTI_Line15;
    //选择模式这里选择中断模式
    EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;
    //下降沿模式
    EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Falling;
    //使能中断
    EXTI_InitStruct.EXTI_LineCmd = ENABLE;
    EXTI_Init(&EXTI_InitStruct);
}
```

**exti.h**

```
#ifndef __EXTI_H
#define __EXTI_H

#include "stm32f10x.h"

#define   EXTI_Key1_GPIO_PIN        GPIO_Pin_15
#define   EXTI_Key1_GPIO_POTR       GPIOA
#define   EXTI_Key1_GPIO_CLK        RCC_APB2Periph_GPIOA

void EXTI_Key_Config(void);
#endif /*__EXTI_H */
```

**main.c**

```
#include "stm32f10x.h"
#include "led.h"
#include "exti.h"

#define SOFT_DELAY Delay(0x0FFFFF);


void Delay(__IO u32 nCount);
```

```
int main(void)
{
   /* LED 端口初始化 */
   LED_GPIO_Config();
  EXTI_Key_Config();
   LED_G(OFF);
   LED_R(OFF);

   while(1)
   {
   }
}

void Delay(__IO uint32_t nCount)    //简单的延时函数
{
   for(; nCount != 0; nCount--);
}
```

**中断服务函数**

```
void EXTI15_10_IRQHandler(void)
{
   //防抖
     Delay(0x0FFFF);
   //判断按键是否按下
   if( GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_15)== 0)
   {

   if(  EXTI_GetITStatus(EXTI_Line15) != RESET )
     {
         //电平翻转
        GPIOA->ODR ^= GPIO_Pin_8;
      }
   //清除中断挂起位 必须
    EXTI_ClearITPendingBit(EXTI_Line15);
   }
}
```