STM32 之外部中断 **EXTI**

STM32 中断控制线支持 19 个外部中断 /事件请求，每个中断都有对应状态位和独立的触发与屏蔽设置。 STM32F103 的 19 个外部中断为：

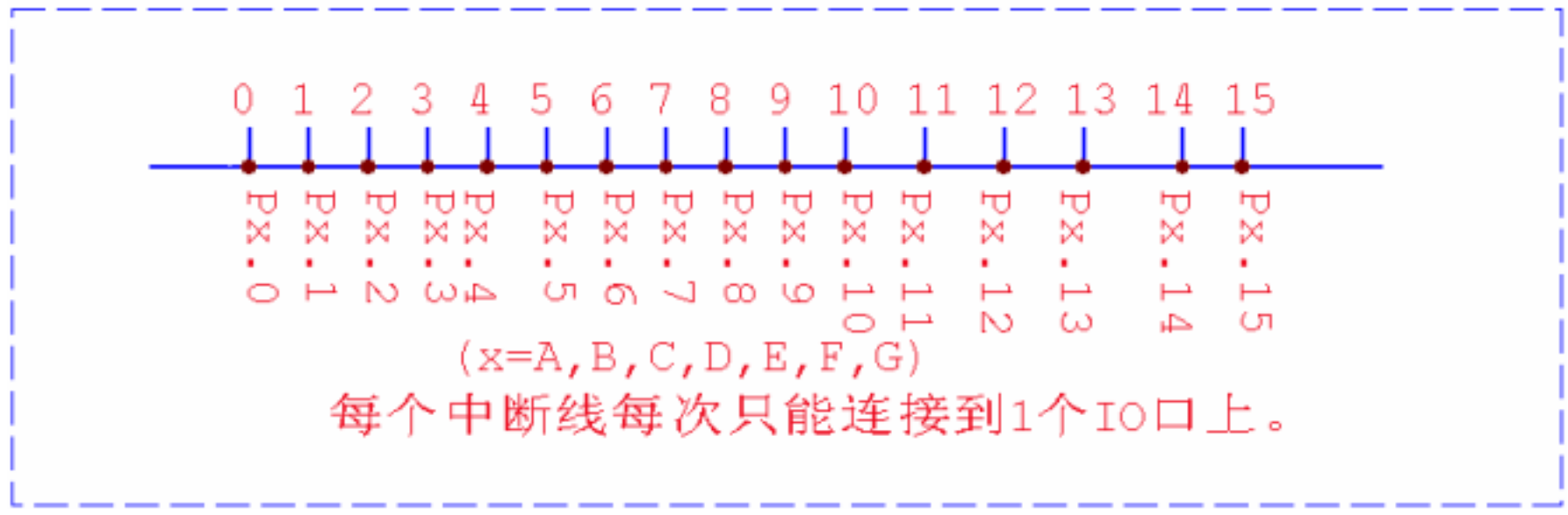
线 0-15：对应 GPIO 口的输入中断。

线 16：连接到 PVD 输出(掉电检测， 掉电时可立即保存重要数据作用)。

线 17：连接 RTC 闹钟事件。

线 18：连接到 USB 唤醒事件。

每个 IO 口都可作为外部中断输入口的 STM32 的 IO 口远多于 16 个，而 IO 口使用的中断线只有 16 个。为此， GPIO 管脚 GPIOx.0~GPIOx.15(X=A,F,G)分别对应中断线 0-15.这样每个中断线对应了 7 个 IO 口。如线 0 对应着 GPIOA.0,GPIOB.0,GPIOC.0,GPIOD.0,GPIOE.0,GPIOF.0,GPIOG0 但是每个中断线每次只能连接到 1 个 IO 口上。



这样就需要通过相关的配置来决定对应的中断线被配置到哪个 GPIO 上了。

GPIO 和中断线的映射关系如下图：

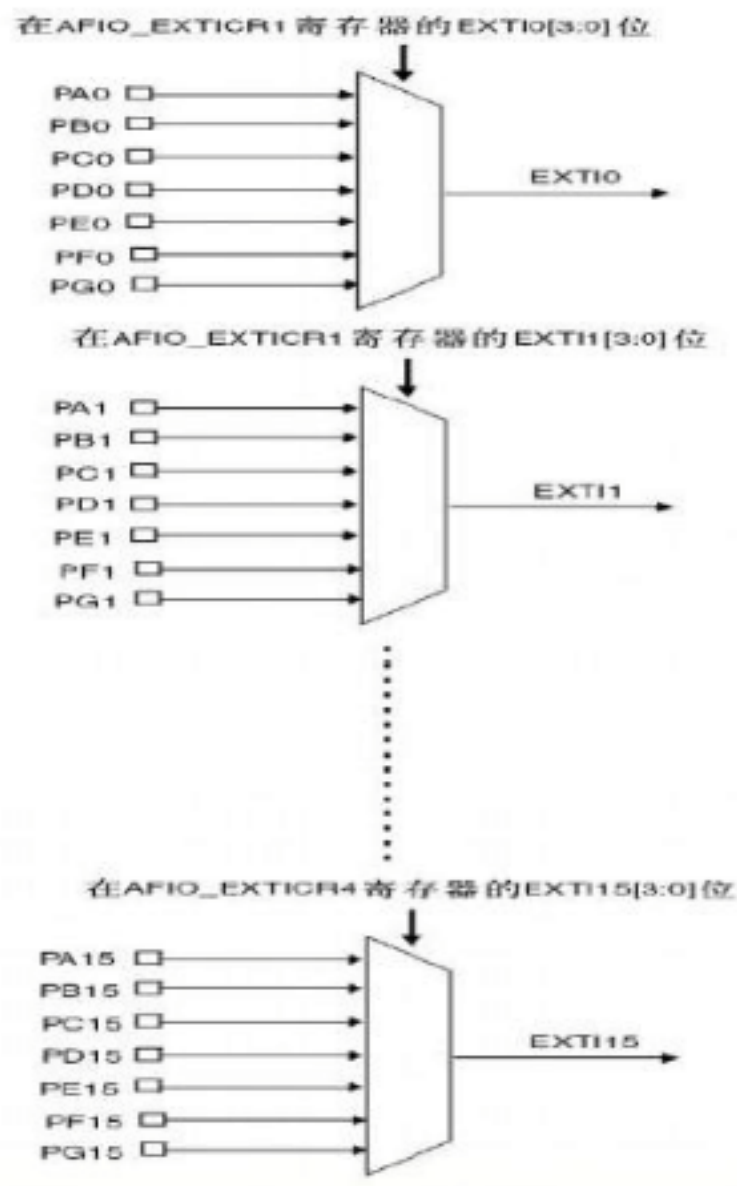


图 7.1.1 GPIO 和中断线的映射关系图

GPIO 与中断映射配置通过函数来实现

```
void GPIO_EXTILineConfig ( uint8_t  GPIO_PortSource,
                           uint8_t  GPIO_PinSource
                         )
```

例如：

GPIO_EXTILineConfig(GPIO_PortSourceGPIOE , GPIO_PinSource2)；

即外部中断线 2 就和 GPIOE 映射起来了， 显然是 GPIOE.2 与 EXTI2 中断线连接起来了。

接着需设置中断触发方式：

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

例如：

```
EXTI_InitTypeDef    EXTI_InitStructure;
EXTI_InitStructure.EXTI_Line=EXTI_Line4;
EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
EXTI_InitStructure.EXTI_LineCmd = ENABLE;
EXTI_Init(&EXTI_InitStructure);
```

以上例子设置了中断线 line4 的中断为下降沿触发。

设置好了中断线和 GPIO 的映射关系， 然后又设置好了中断的触发模式等参数。既然是外部中断，设计到中断我们当然需要设置 **NVCI** 的中断优先级。

```
NVIC_InitTypeDef NVIC_InitStructure;
NVIC_InitStructure.NVIC_IRQChannel = EXTI2_IRQn;//使能按键外部中断通道
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x02;  //抢占优先级 2，
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x02;  //子优先级 2
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;      //使能外部中断通道
NVIC_Init(&NVIC_InitStructure);      //中断优先级分组初始化
```

配置完中断优先级后，接着是编写中断服务函数。中断服务函数名字在 MDK 中事先有定义。但是不得不提的是， STM32 的 IO 口外部中断只有 6 个,分别是：

```
EXPORT
EXT0_IRQHandle      EXPORT----------------------    中断线 0 对应的中断函数
EXT1_IRQHandle      EXPORT----------------------    中断线 1 对应的中断函数
EXT2_IRQHandle      EXPORT----------------------    中断线 2 对应的中断函数
EXT3_IRQHandle      EXPORT----------------------    中断线 3 对应的中断函数
EXT4_IRQHandle      EXPORT----------------------    中断线 4 对应的中断函数
EXT9_5_IRQHandle EXPORT----------------------      中断线 5-9 对应的中断函数
EXT15_10_IRQHandle ----------------------------    中断线 10-15 共用的中断函数
```

编写中断服务函数经常需要使用两个函数。

第一个是判断某个中断线上的中断是否发生 (即标志位是否置位)：

```
ITStatus  EXTI_GetITStatus(uint32_t  EXTI_line);
//放在中断服务函数开头 ,监测中断标志位
```

第二个是清除某个中断线上的中断标志位 (即清除标志位)：

```
void    EXTI_ClearITPendingBit(uint32_t EXTI_Line) ；//放在函数的结尾
```

常用外部中断服务函数的格式：

```
//格式一
void    EXTI3_IRQHandler(void)
 {
          If( EXTI_GetITStatus(EXTI_Line3)!-=RESET)//  判断是否置位
          {
               Dosomething();//执行中断事件
               EXTI_ClearITPendingBit(EXTI_Line3)  ；//清零 LINE 上标志位
          }
 }
//=============================================================
//格式二
void    EXTI3_IRQHandler(void)
 {
          If( EXTI_GetFlagStatus(EXTI_Line3)!-=RESET)// 判断是否置位
          {
               Dosomething();//执行中断事件
               EXTI_ClearFlag(EXTI_Line3)  ；//清零 LINE 上标志位
          }
 }
//=============================================================
```

区别：EXTI_GetITStatus()会先判断该中断是否使能，若使能了再判断中断标志位，而 EXTI_GetFlagStatus()直接判断中断标志位是否置位 (马虎点)。所以说方式一的 EXTI_GetITStatus()会好点，一般配合 EXTI_ClearITPendingBit() 使用。

设计二室： sickhum

2014年 12月 8日 22:51:40