实验 7　外部中断实验

这一章，我们将向大家介绍如何使用 STM32 的外部输入中断。在前面几章的学习中，我们掌握了 STM32 的 IO 口最基本的操作。 本章我们将介绍如何将 STM32 的 IO 口作为外部中断输入，在本章中，我们将以中断的方式，实现按键控制 LED 亮灭的功能。 本章分为以下学习目标：

1、了解 STM32 的中断模式 NVIC。

2、了解外部中断的使用方式。

1.1 中断的概念

如果学员有单片机的基础，那么就会知道中断的概念。所谓中断是指 CPU 在执行当前程序的过程中，由于某种随机出现的外设请求或 CPU 内部的异常事件，使 CPU 暂停正在执行的程序而转去 执行相应的服务处理程序；当服务处理 程序运行完毕后，CPU 再返回到暂停处继续执行原来的程序。 而有些中断还能够被其他高优先级的中断所中断，那么这种情况又叫做中断的嵌套。

1.2 STM32 的中断

1) STM32 中断分组

STM32 的中断一共有 68 个可屏蔽中断通道(不包含 16 个 Cortex™-M3 的 中断线)；16 个可编程的优先等级；什么叫做可编程等级呢？ 首先我们来看一下 STM32 中断等级模式，STM32 使用一个 4 位寄存器 来设置中断等级的，4 位长度可以设置 16 个优先等级（2 的 4 次方等于 16)； 而这个优先等级是可以设置的，所以叫做可编程等级。而且呢，这个 4 位长 度的优先等级还可以分为两部分，一部分叫做抢占优先级，一部分叫做响应 优先级（有些资料也叫做亚优先级） （比如说： 4 位里面设置 1 位抢占优先级，

3 位的响应优先级。）在这里大家可能有点糊涂了，分成抢占优先级和响应优先级又是什么用呢？

在 STM32 的 NVIC 中规定： 抢占优先级高的可以中断抢占优先级比它低 的中断函数，不过相同抢占优先级的之间是不能相互中断的。而响应优先级 之间是

不能相互中断的，有人或许既然都不能中断了，那响应优先级还有什 么意义呢？如果这两个中断同时到达，则中断控制器根据他们的响应优先级 高低来决定先处理哪一个；如果他们的抢占式优先级和响应优先级都相等， 则根据他们在中断表中的排位顺序决定先处理哪一个。

STM32 的中断分组可以有 5 种分组：

组	中断配置寄存器	中断优先寄存器	配置结果
0	111	0-4	0 位抢占，4 位从
1	110	1-3	1 位抢占，3 位从
2	101	2-2	2 位抢占，2 位从
3	100	3-1	3 位抢占，1 位从
4	011	4-0	4 位抢占，0 位从

所以一般我们使用中断的时候，要先设置 NVIC 的分组。一般来说在配置中断 NVIC 变量前就设置分组了。

2) STM32 中断分组库函数。

从上面的学习我们知道，在使用中断的时候，先要设置 NVIC 的分组。 而怎么设置呢？

在 V3.5 的库中，它定义了一个设置分组函数 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

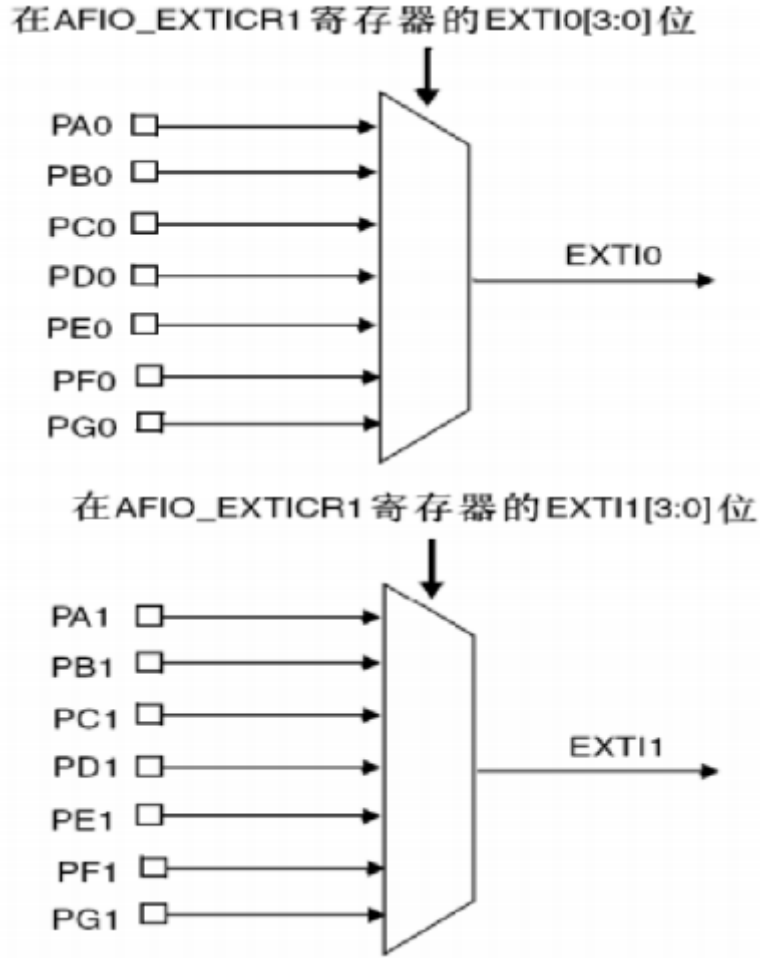
NVIC_PriorityGroup 这个参数是用来选择设置分组的参数，我们知道它一共 可以分为 5 个组别，所以，在 V3.5 的库中，5 个组别也帮你定义好了名字：

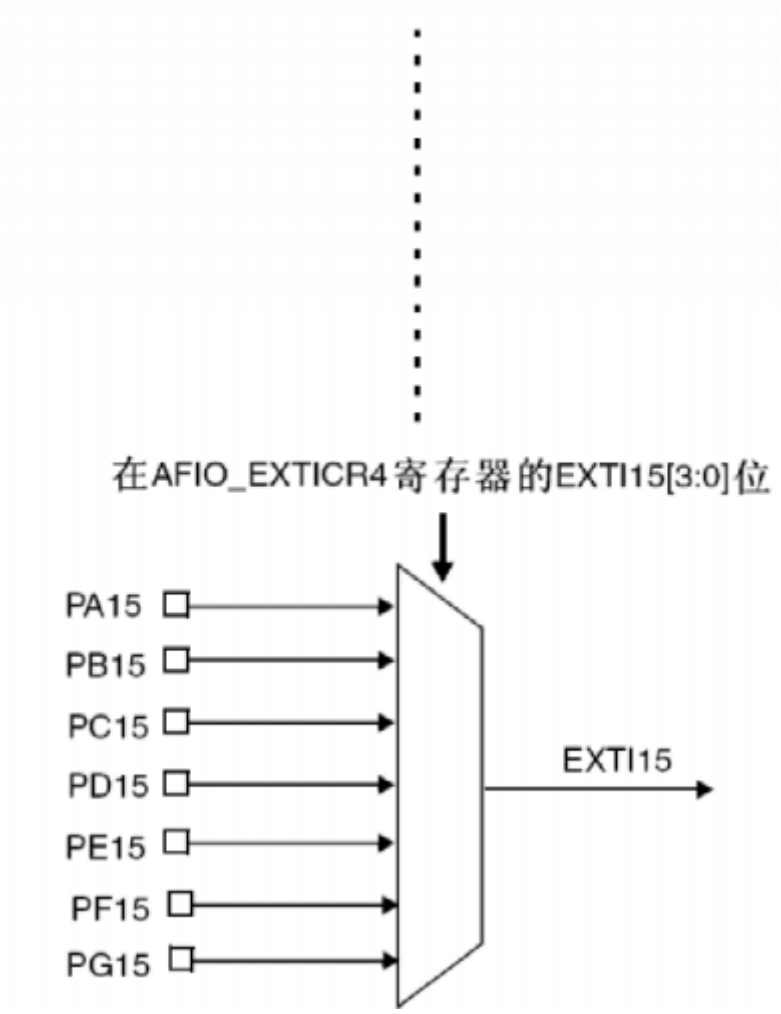
NVIC_PriorityGroup	描述
NVIC_PriorityGroup_0	先占优先级 0 位 从优先级 4 位
NVIC_PriorityGroup_1	先占优先级 1 位 从优先级 3 位
NVIC_PriorityGroup_2	先占优先级 2 位 从优先级 2 位
NVIC_PriorityGroup_3	先占优先级 3 位 从优先级 1 位
NVIC_PriorityGroup_4	先占优先级 4 位 从优先级 0 位

1.3 EXTI 外部中断

1) EXTI 外部中断的结构

STM32 不像 51 一样，只有两个外部中断 IO，STM32 的 IO 口每个都可 以用来做外部中断。而它的外部中断结构是怎么样的呢？首先我们来看一个 图：





从图上我们可以很清楚的看到，STM32 的外部中断是有 16 条外部中断 通道，分别对应着每组 IO 口的 Px0 到 Px15。从图上我们也可以知道，虽然 STM32 的每个 IO 口都能用作外部中断，但是呢，它一次性同时使用的外部 中断只能有 16 个，而且同时做外部中断的 IO 口序号也是不能相同的，比如，你使用 PA0 做外部中断了，那么就不能同时使用 PB0 做外部中断了。不过 同时使用 16 个外部中断，也足够我们使用的了。

外部中断的设置步骤：

1) 一般设置一个外部中断的步骤呢，如下：

2) 设置中断分组。 （一般在系统开机之后开始设置，而且一个程序中只 设置一次。 ）

3) 配置 IO 口的模式。(比如你要使用下降沿触发，那么就选择上拉输入 模式，要使用上升沿触发，那么就选择下拉输入模式。)

4) 选择要用作外部中断的 IO 口作为输入。

5）开启要使用的中断通道并设置外部中断的参数（如：中断优先级、中 断触发模式）

1.4 V3.5 库函数介绍

外部中断的函数是放在 stm32f10x_exti.c 中的，而 NVIC 它是放在 misc.c 中。 透过上面的操作步骤，如果我们使用 V3.5 库函数的话，一般会使用的库函数可以有：

1) RCC_APB2PeriphClockCmd()函数

我们要打开 GPIOE 及管脚复用的时钟。

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOE, ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);
```

2) GPIO_Init()函数

IO 模式设置为上拉输入：

```
GPIO_InitStructure.GPIO_Pin=k_left;
GPIO_InitStructure.GPIO_Mode=GPIO_Mode_IPU;
GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
GPIO_Init(GPIOE,&GPIO_InitStructure);
```

3) GPIO_EXTILineConfig()函数

这个函数用来设置作为输入的 IO 口，它有两个输入参数： 第一个参数是使用选择 GPIO 的组别，我们要使用的是 PE2，所以设 置为：
GPIO_PortSourceGPIOE。 第二个参数是设置 IO 的序号，我们使用的是 PE2，所以我们为：
GPIO_PinSource2 所以设置代码为：

```
GPIO_EXTILineConfig(GPIO_PortSourceGPIOE, GPIO_PinSource2);
```

4) NVIC_Init()函数：

用来设置中断的优先级和打开总中断。这个要输入一个结构体指针。 这个结构体的参数分别有四个成员：
第一个成员是 NVIC_IRQChannelPreemptionPriority，表示抢占优先级的 等级，我们设置为 0。

第二个成员是 NVIC_IRQChannelSubPriority，表示响应优先级的等级， 我们也设置为 0。

第三个成员是 NVIC_IRQChannel，表示选择你要设置的全局中断，我们 要设置的中断是外部中断 0，所以我们设置为：EXTI0_IRQn。

第四个成员是 NVIC_IRQChannelCmd,表示要设置的状态,我们是要打 开中断的，所以我们设置为：ENABLE。

所以最后的设置如下：

```
/* 设置 NVIC 参数 */
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
    NVIC_InitStructure.NVIC_IRQChannel = EXTI2_IRQn;     //打开 EXTI2 的全局中断
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0; //抢占优先级为 0
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;         //响应优先级为 0
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;         //使能
    NVIC_Init(&NVIC_InitStructure);
```

5) EXTI_Init()函数：

这个函数是用来初始化外部中断的，它有一个参数，用来传递一个结构 体，这个结构体有 4 个成员。

第一个成员：EXTI_Line，表示外部中断的通道。我们要使用的通道 0， 所以设置为：EXTI_Line0。

第二个成员：EXTI_LineCmd，表示要设置的状态，我们是要打开使能， 所以设置为：ENABLE。

第三个成员：EXTI_Mode，表示要使用的模式，我们设置为中断模式， 所以设置为：EXTI_Mode_Interrupt。

第四个成员：EXTI_Trigger，表示中断触发模式，我们这里使用上升沿 触发，所以设置为：EXTI_Trigger_Rising。

设置代码为：

```
/* 设置外部中断的模式 */
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);
```

6) 外部中断的中断函数

在库函数中，每个外部中断它都帮你定义好了中断名字，只要我们 使 用 就 可以了，它一般定义在启动文件中，大家可以打开 startup_stm32f10x_hd.s 查看 264 行之后，都是帮起好的中断函数。 而外部中断 0 的中断函数是 void EXTI0_IRQHandler(void)；在这里 大家要注意的是，外部中断虽然同时可以使用 16 个外部中断，但是它并 没有 16 个中断函数。它们分别是：

EXTI0_IRQHandler

EXTI1_IRQHandler

EXTI2_IRQHandler

EXTI3_IRQHandler

EXTI4_IRQHandler

EXTI9_5_IRQHandler

EXTI15_10_IRQHandler

从上面可以看出，只有从外部中断 0 到外部中断 4 有独立的中断函数， 而从外部中断 5 到外部中断 9 是共用一个中断函数，而外部中断 10 到外部 中断 15 共用一个中断函数。而我们如何区分呢？ 在进入中断函数之后，我们可以先检测一下你想要的中断标志，如果中 断标志设置了，那么就是你想要的中断设置了，如果没有那么就是不是你想 要的中断。

7) EXTI_GetITStatus()函数

这个函数可以用来读取你想要的中断标志。它有一个参数，用来选择你 要读取的中断标志位。比如你想要读取 EXTI0 那么就设置为 EXTI_Line0。 同时它还会返回一个参数。 SET （非零）： 表示标志位已设置； RESET （0）： 表示标

志位未设置。 所以我们要读取外部中断 0 的代码为：

```
if (EXTI_GetITStatus(EXTI_Line0))
{
}
```

8) EXTI_ClearITPendingBit()函数

这个函数的作用是用来清除中断标志，外部中断的中断标志要自己软件 手动清除。它有一个输入参数，用来选择你要清除的外部中断。它的参数其 实跟上面的 EXTI_GetITStatus()函数是一样的。

1.5 例程程序

1) 初始化函数

```
/*******************************************************************************
***********
* 函 数 名          : exti_init
* 函数功能          : 外部中断 2 端口初始化函数
* 输    入          : 无
* 输    出          : 无
*******************************************************************************
**********/
void exti_init()  //外部中断初始化
{
   GPIO_InitTypeDef GPIO_InitStructure;

   EXTI_InitTypeDef EXTI_InitStructure;

   NVIC_InitTypeDef NVIC_InitStructure;

   /* 开启 GPIO 时钟 */
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);
```

```
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOE,ENABLE);

    GPIO_InitStructure.GPIO_Pin=k_left;
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_IPU;
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
    GPIO_Init(GPIOE,&GPIO_InitStructure);

    GPIO_EXTILineConfig(GPIO_PortSourceGPIOE, GPIO_PinSource2);//选择
GPIO 管脚用作外部中断线路
    //此处一定要记住给端口管脚加上中断外部线路
    /* 设置外部中断的模式 */
    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);

    /* 设置 NVIC 参数 */
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
    NVIC_InitStructure.NVIC_IRQChannel = EXTI2_IRQn;     //打开 EXTI2
的全局中断
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0; //抢占优
先级为 0
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;        //响应
优先级为 0
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;        //使能
    NVIC_Init(&NVIC_InitStructure);
}
```

按键端口定义

```
#define k_left GPIO_Pin_2  //K1 PE2
```

2) 中断函数

```
void EXTI2_IRQHandler()     //外部中断 2 中断函数
{
    if(EXTI_GetITStatus(EXTI_Line2)==SET)
    {
        EXTI_ClearITPendingBit(EXTI_Line0);//清除 EXTI 线路挂起位
        delay_ms(10);//消抖处理
        if(GPIO_ReadInputDataBit(GPIOE,GPIO_Pin_2)==Bit_RESET)
//k_left 按键按下
        {
            if(GPIO_ReadOutputDataBit(GPIOC,GPIO_Pin_0)==Bit_RESET)
            {
                //LED 熄灭
               GPIO_SetBits(GPIOC,GPIO_Pin_0);
            }
            else
            {
               //LED 发光
                GPIO_ResetBits(GPIOC,GPIO_Pin_0);
            }
        }
        while(GPIO_ReadInputDataBit(GPIOE,GPIO_Pin_2)==0);
    }
}
```

这个函数是 K_left 的外部中断函数，在进入中断函数之后先检测相应的中 断的标志是否设置，有人可能觉得这个有点多余，但是从上面我们讲到 中断的时候，我们知道有些时候是几个外部中断共用一个外部中断函数的，所以 我们要读取清楚。做完中断处理，离开中断函数之前，记得要清除相应的中断标 志。

3) 主函数很简单，这里就贴出来了，大家参考例程就可以了。