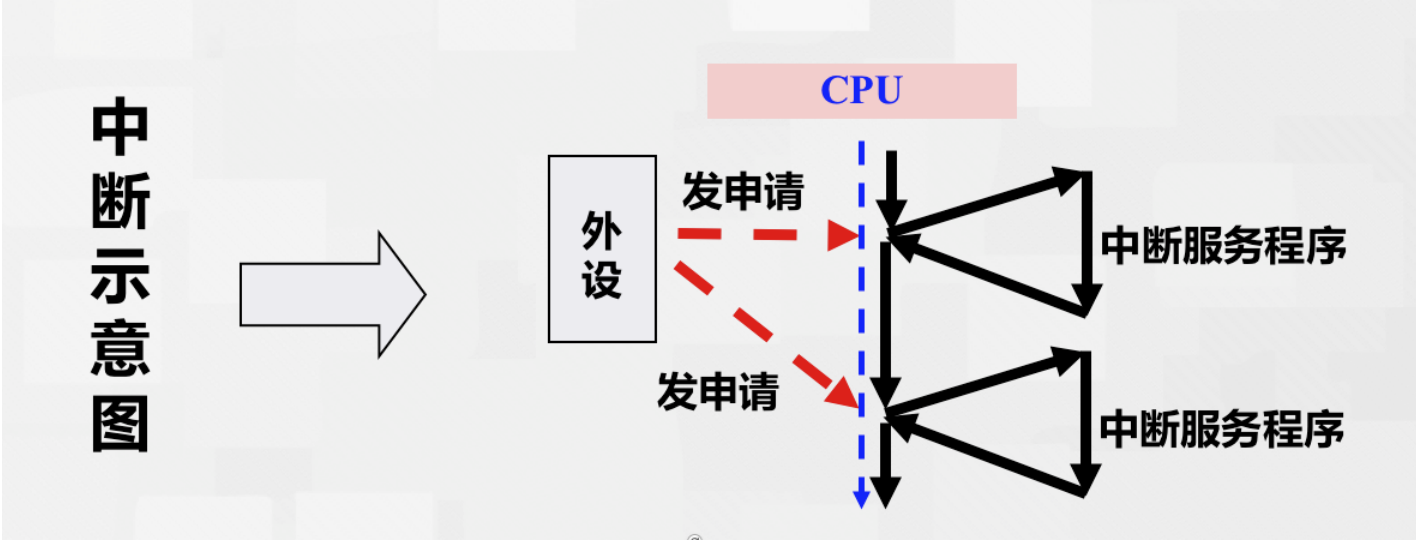
控制组STM32培训4——中断

1. 什么是中断?



CPU执行程序时，由于发生了某种随机的事件(外部或内部)，引起CPU暂时中断正在运行的程序，转去执行一段特殊的服务程序(中断服务子程序或中断处理程序)，以处理该事件，该事件处理完后又返回被中断的程序继续执行，这一过程称为中断。

example:

- 吃饭的时候电话响了->中断吃饭去接电话->接电话事件处理完毕继续吃饭

ARM Coetex-M3内核共支持256个中断，其中16个内部中断，240个外部中断和可编程的256级中断优先级的设置。STM32目前支持的中断共84个（16个内部+68个外部），还有16级可编程的中断优先级的设置，仅使用中断优先级设置8bit中的高4位。

STM32可支持68个中断通道，已经固定分配给相应的外部设备，每个中断通道都具备自己的中断优先级控制字节(8位，但是STM32中只使用4位，高4位有效)，每4个通道的8位中断优先级控制字构成一个32位的优先级寄存器。68个通道的优先级控制字至少构成17个32位的优先级寄存器。

2. 什么是中断优先级?

2.1 抢占优先级

高抢占式优先级的中断事件会打断当前的主程序/中断程序运行，俗称中断嵌套。

2.2 响应优先级

- 在抢占式优先级相同的情况下，高响应优先级的中断优先被响应；
- 在抢占式优先级相同的情况下，如果有低响应优先级中断正在执行，高响应优先级的中断要等待已被响应的低响应优先级中断执行结束后才能得到响应—(不能嵌套)。

2.3 判断中断是否会被响应

- 先看抢占优先级，抢占优先级相同看响应优先级

2.4 优先级冲突的处理

两个中断同时到来

- 抢占优先级相同：看响应优先级。响应优先级高的先执行
- 抢占式优先级和响应优先级都相等：根据他们在中断表中的排位顺序决定先处理哪一个

2.5 STM32 中断优先级的定义

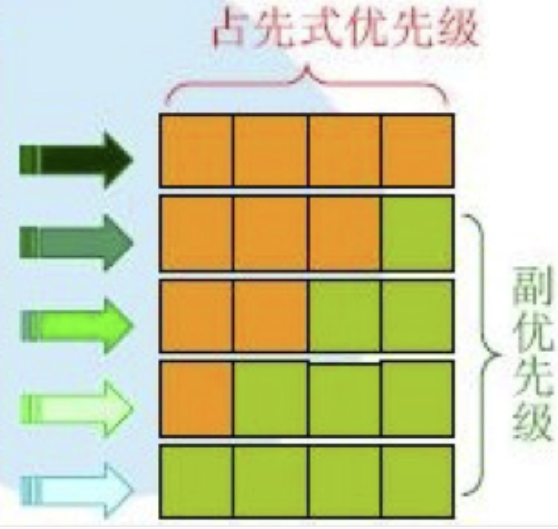
STM32中指定中断优先级的寄存器宽度为4位，这4个寄存器位的分组方式如下：

- 第0组：所有4位用于指定响应优先级
- 第1组：最高1位用于指定抢占式优先级，最低3位用于指定响应优先级
- 第2组：最高2位用于指定抢占式优先级，最低2位用于指定响应优先级
- 第3组：最高3位用于指定抢占式优先级，最低1位用于指定响应优先级
- 第4组：所有4位用于指定抢占式优先级

- 占先式优先级与副优先级的分配
  - 4个描述优先级位有下列5种组合使用方式
  - “优先级组别”决定如何解释这4位。

优先级组别	占先式优先级	副优先级
4	4位/16级	0位/0级
3	3位/8级	1位/2级
2	2位/4级	2位/4级
1	1位/2级	3位/8级
0	0位/0级	4位/16级

4个优先级描述位可以有5种组合方式。



3. 外部中断程序的编写

STM32 的每一个GPIO都能配置成一个外部中断触发源，这点也是STM32 的强大之处。STM32 通过根据引脚的序号不同将众多中断触发源分成不同的组，比如：`PA0，PB0，PC0，PD0，PE0，PF0，PG0`为第一组，那么依此类推，我们能得出一共有16 组，STM32 规定，每一组中同时只能有一个中断触发源工作，那么，最多工作的也就是16个外部中断。STM32F103 的中断控制器支持 19 个外部中断/事件请求。每个中断设有状态位，每个中断/事件都有独立的触发和屏蔽设置。

STM32F103 的19个外部中断

- `EXTI Line 0-15`：外部IO输入中断

GPIO引脚	中断标志位	中断处理函数
PA0~PG0	EXTI0	EXTI0_IRQHandler
PA1~PG1	EXTI1	EXTI1_IRQHandler
PA2~PG2	EXTI2	EXTI2_IRQHandler
PA3~PG3	EXTI3	EXTI3_IRQHandler
PA4~PG4	EXTI4	EXTI4_IRQHandler
PA5~PG5	EXTI5	EXTI9_5_IRQHandler
PA6~PG6	EXTI6	
PA7~PG7	EXTI7	
PA8~PG8	EXTI8	
PA9~PG9	EXTI9	
PA10~PG10	EXTI10	EXTI15_10_IRQHandler
PA11~PG11	EXTI11	
PA12~PG12	EXTI12	
PA13~PG13	EXTI13	
PA14~PG14	EXTI14	
PA15~PG15	EXTI15	

- `EXTI LINE 16`：连接到 PVD 输出。
- `EXTI LINE 17`：连接到 RTC 闹钟事件。
- `EXTI LINE 18`：连接到 USB 唤醒事件。

3.1 时钟使能

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);

/* 使能AFIO端口复用时钟 */
RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
```

3.2 配置GPIO

```
void initGPIO(void) {
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOE, ENABLE);
    /* 使能端口复用时钟 */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
    GPIO_InitTypeDef gpio_InitStruct;
    gpio_InitStruct.GPIO_Mode = GPIO_Mode_IPU;  // 上拉输入
    gpio_InitStruct.GPIO_Pin = GPIO_Pin_2;
    gpio_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOE, &gpio_InitStruct);
}
```

3.3 配置外部中断

由于我们要配置中断的引脚为`PE2`，对应的中断为`EXTI2`，中断线路为`EXTI_Line_2`，下面做出相应初始化操作。

```
void initEXTI(void) {
    /* GPIO与外部中断线路映射 */
    GPIO_EXTILineConfig(GPIO_PortSourceGPIOE, GPIO_PinSource2);

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

外部中断触发方式

```
exti_InitStruct.EXTI_Trigger = EXTI_Trigger_Falling;
```

- `EXTI_Trigger_Falling`：下降沿触发
- `EXTI_Trigger_Rising`：上升沿触发
- `EXTI_Trigger_Rising_Falling`：任意电平（上升沿和下降沿）触发

3.4 配置中断优先级

```
void initNVIC(void) {
    /* 设置优先级分组为第二组 -- 2位抢占2位响应 */
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
    NVIC_InitTypeDef NVIC_InitStructure;//定义结构体
    /* 使能外部中断所在的通道 */
    NVIC_InitStructure.NVIC_IRQChannel = EXTI2_IRQn;
    /* 抢占优先级 2 */
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority
                                        = 0x02;
    /* 响应优先级 2 */
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x02;

    /* 使能外部中断通道 */
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;

    NVIC_Init(&NVIC_InitStructure);
}
```

3.5 编写中断函数

中断入口函数名可在启动汇编代码中找到，也可在其中定义

```
DCD     EXTI0_IRQHandler            ; EXTI Line 0
DCD     EXTI1_IRQHandler            ; EXTI Line 1
DCD     EXTI2_IRQHandler            ; EXTI Line 2
DCD     EXTI3_IRQHandler            ; EXTI Line 3
DCD     EXTI4_IRQHandler            ; EXTI Line 4
```

编写相应的中断函数如下

```
void EXTI2_IRQHandler(void) {
    if (EXTI_GetITStatus(EXTI_Line2) != RESET) {
        /* 中断发生 */

        /* 写中断逻辑代码 */

        /* 清除中断标志位 */
        EXTI_ClearITPendingBit(EXTI_Line2);
    }
}
```

example: 响应按键中断

```
void EXTI2_IRQHandler(void) {
    if (EXTI_GetITStatus(EXTI_Line2) != RESET) {
        /* 中断发生 */

        /* 写中断逻辑代码 */
        delay_ms(10);  // 消抖
        if (READ_BUTTON == RESET) {
            /* 按下 */
            /* 做出对应操作 */
        }
        while(READ_BUTTON == RESET);  // 等待松手
        /* 清除中断标志位 */
        EXTI_ClearITPendingBit(EXTI_Line2);
    }
}
```

3.6 在程序中产生软中断

`void EXTI_GenerateSWInterrupt(u32 EXTI_Line)`

example

```
/* 模拟7号线外部中断触发 */
EXTI_GenerateSWInterrupt(EXTI_Line7);
```