STM32Cube高效开发教程（基础篇）

第7章 中断系统和外部中断

王维波
中国石油大学（华东）控制科学与工程学院

STM32Cube高效开发教程（基础篇）

作者：王维波，鄢志丹，王钊

人民邮电出版社

2021年9月出版

如果有读者需要本书课件的PPT版本用于备课，可以给作者发邮件免费获取，并可加入专门的教学和技术交流QQ群

邮箱：wangwb@upc. edu. cn

第7章 中断系统和外部中断

7.1 STM32F407的中断

7.2 外部中断EXTI

7.3 外部中断使用示例

7.1.1 中断向量表

NVIC--嵌套向量中断控制器(Nested vectored interrupt controller)

有82个可屏蔽中断通道，还有13个系统中断。82个可屏蔽中断和部分系统中断是可以配置优先级，总共有16个优先级。

ISR （Interrupt Service Routine，中断服务例程），对中断进行响应的函数，每个中断有一个ISR。

MCU对各个中断的ISR函数名已经定义好了，在MCU的启动文件startup_stm32f407xx.s中有这些ISR名称的定义。

表7-1 STM32F405/407的系统中断

优先级	优先级类型	名称	说明	ISR函数名
-3	固定	Reset	复位	
-2	固定	NMI	不可屏蔽中断，RCC时钟安全系统连接到NMI	NMI_Handler
-1	固定	HardFault	所有类型的错误	HardFault_Handler
0	可设置	MemManage	存储器管理	MemManage_Handler
1	可设置	BusFault	预取指失败，存储器访问失败	BusFault_Handler
2	可设置	UsageFault	未定义或非法指令	UsageFault_Handler
3	可设置	SVCall	通过SWI指令调用的系统服务	SVC_Handler
4	可设置	DebugMonitor	调试器监视	DebugMon_Handler
5	可设置	PendSV	可挂起的系统服务	PendSV_Handler
6	可设置	SysTick	系统嘀嗒定时器	SysTick_Handler

可屏蔽中断见表7-2

通常，一个中断号有多个中断事件源

7.1.2 中断优先级

NVIC采用4位二进制数设置中断优先级，分为抢占优先级（pre-emption priority）和次优先级（subpriority），优先级的数字越小表示优先级别越高。

这4位二进制数可以分为两段，一段用于设置抢占优先级，一段用于设置次优先级，所以有：

- 0位用于抢占优先级，4位用于次优先级
- 1位用于抢占优先级，3位用于次优先级
- 2位用于抢占优先级，2位用于次优先级
- 3位用于抢占优先级，1位用于次优先级
- 4位用于抢占优先级，0位用于次优先级

假设使用2位设置抢占优先级，2位设置次优先级，抢占优先级和次优先级的执行有如下的规律：

（1）如果两个中断的抢占优先级和次优先级都相同，哪个中断先发生就执行哪个中断的ISR。

（2）高抢占优先级的中断可以打断正在执行的低抢占优先级的ISR的执行。

例如中断A的抢占优先级为0，中断B的抢占优先级为1，在中断B的ISR正在执行时如果发生了中断A，就会立即去执行中断A的ISR，等中断A的ISR执行完后再返回到中断B的ISR继续执行。

（3）抢占优先级相同时，次优先级高的中断不能打断正在执行的次优先级低的ISR函数的执行。

例如，中断A和中断B的抢占优先级相同，但是中断A的次优先级为0，中断B的次优先级为1。那么，在中断B的ISR正在执行时如果发生了中断A，则中断A不能打断中断B的ISR的执行，只能等待中断B的ISR执行结束后再执行中断A的ISR。

7.1.3 中断设置相关函数

函数名	功能
HAL_NVIC_SetPriorityGrouping()	设置优先级分组方案
HAL_NVIC_SetPriority()	设置某个中断的抢占优先级和次优先级
HAL_NVIC_EnableIRQ()	启用某个中断
HAL_NVIC_DisableIRQ()	禁用某个中断

1. 函数HAL_NVIC_SetPriorityGrouping()

设置优先级分组策略

```
void HAL_NVIC_SetPriorityGrouping (uint32_t PriorityGroup);
```

优先级分组策略宏定义

```
//0位用于抢占优先级，4位用于次优先级
#define NVIC_PRIORITYGROUP_0         0x00000007U
//1位用于抢占优先级，3位用于次优先级
#define NVIC_PRIORITYGROUP_1         0x00000006U
//2位用于抢占优先级，2位用于次优先级
#define NVIC_PRIORITYGROUP_2         0x00000005U
//3位用于抢占优先级，1位用于次优先级
#define NVIC_PRIORITYGROUP_3         0x00000004U
//4位用于抢占优先级，0位用于次优先级
#define NVIC_PRIORITYGROUP_4         0x00000003U
```

2. 函数HAL_NVIC_SetPriority()

设置某个中断的抢占优先级和次优先级

```
void HAL_NVIC_SetPriority (IRQn_Type IRQn, uint32_t PreemptPriority,
      uint32_t SubPriority);
```

中断号枚举类型IRQn_Type

```
typedef enum
{
      WWDG_IRQn              = 0,
       PVD_IRQn              = 1,
       EXTI0_IRQn            = 6,
      FPU_IRQn               = 81
} IRQn_Type;
```

3. 函数HAL_NVIC_EnableIRQ()

在NVIC控制器中启用某个中断的功能

```
void  HAL_NVIC_EnableIRQ (IRQn_Type IRQn);
```

第7章 中断系统和外部中断

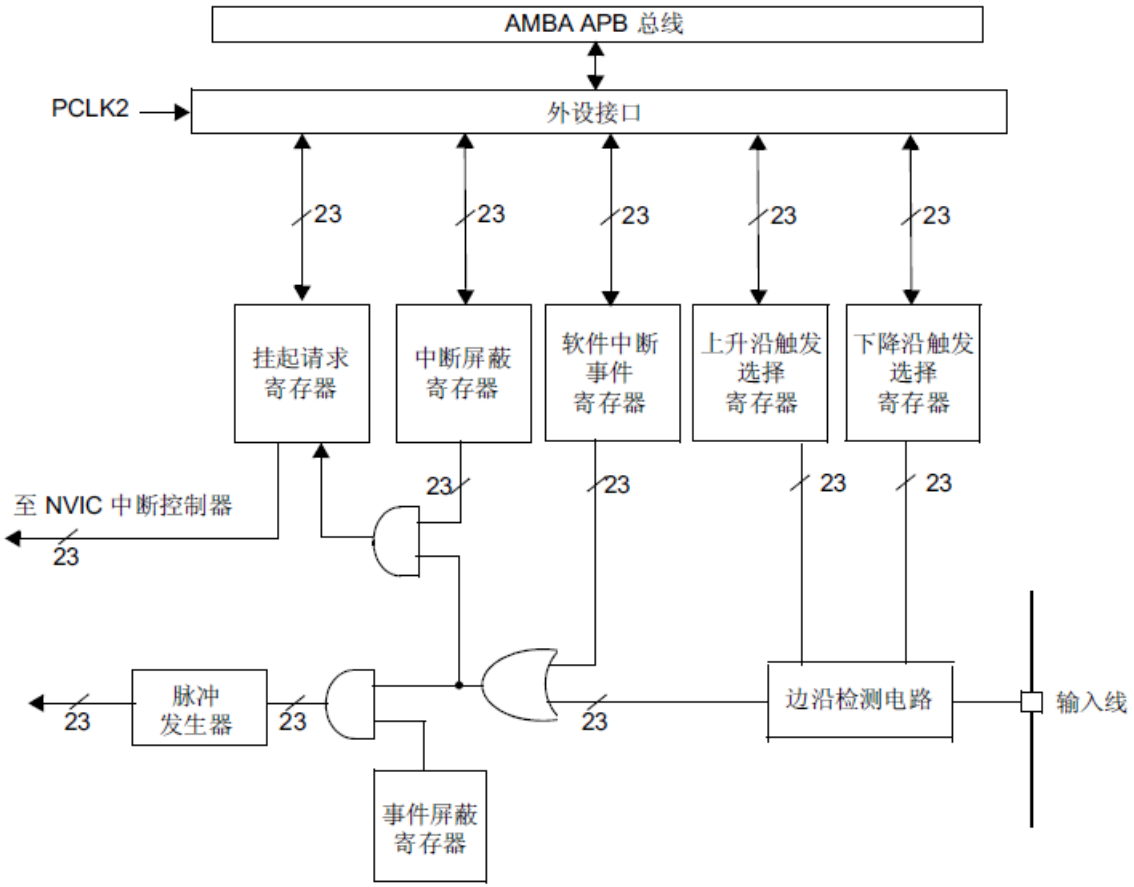
7.1 STM32F407的中断

7.2 外部中断EXTI

7.3 外部中断使用示例

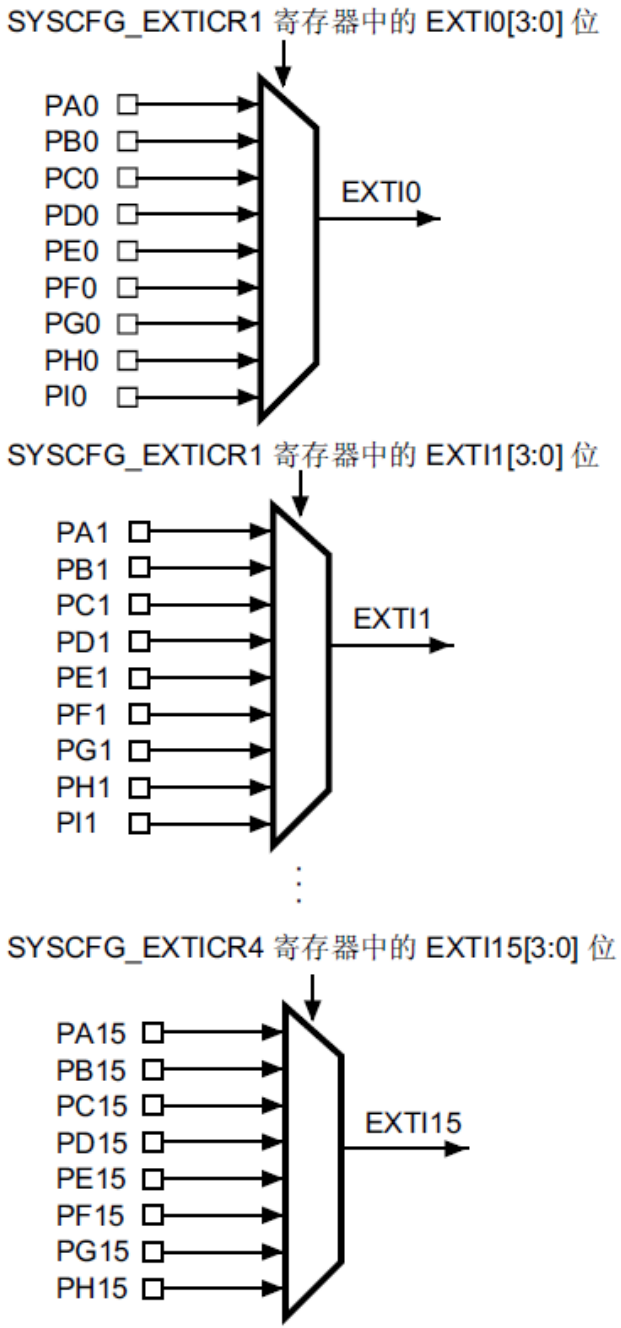
7.2.1 外部中断功能和外部中断线

STM32F407有23个外部中断，每个输入线都可以单独配置触发事件，如上跳沿触发、下跳沿触发或边沿触发。



EXTI0至EXTI15这16个外部中断以GPIO引脚为输入线，每个GPIO端口引脚都可以作为某个EXTI的输入线

EXTI0可以选择PA0、PB0至PI0中的某个引脚作为输入线，如果设置了PA0为 EXTI0的输入线，那么PB0、PC0等就不能再作为EXTI0的输入线



若是共用的ISR函数，需要在ISR函数里再判断具体是哪个EXTI线产生的中断，然后做相应的处理。

位置	中断名称	说明	ISR函数名称
6	EXTI0	EXTI线0中断	EXTI0_IRQHandler
7	EXTI1	EXTI线1中断	EXTI1_IRQHandler
8	EXTI2	EXTI线2中断	EXTI2_IRQHandler
9	EXTI3	EXTI线3中断	EXTI3_IRQHandler
10	EXTI4	EXTI线4中断	EXTI4_IRQHandler
23	EXTI9_5	EXTI线[9:5]中断	EXTI9_5_IRQHandler
40	EXTI15_10	EXTI线[15:10]中断	EXTI15_10_IRQHandler

另外7个EXTI线连接的不是某个实际的引脚，而是其他外设产生的事件信号。这7个EXTI线的中断有单独的ISR函数。

- EXTI线16连接PVD输出
- EXTI线17连接RTC闹钟事件
- EXTI线18连接USB OTG FS 唤醒事件
- EXTI线19连接以太网唤醒事件
- EXTI线20连接USB OTG HS 唤醒事件
- EXTI线21连接RTC入侵和时间戳事件
- EXTI线22连接RTC唤醒事件（后面实例中常用）

7.2.2 外部中断相关函数

STM32F407有23个外部中断，每个输入线都可以单独配置触发事件，如上跳沿触发、下跳沿触发或边沿触发。

函数名	功能描述
__HAL_GPIO_EXTI_GET_IT()	检查某个外部中断线是否有挂起（Pending）的中断
__HAL_GPIO_EXTI_CLEAR_IT()	清除某个外部中断先的挂起标志位
__HAL_GPIO_EXTI_GENERATE_SWIT()	在某个外部中断线上产生软中断
HAL_GPIO_EXTI_IRQHandler()	外部中断ISR函数中调用的通用处理函数
HAL_GPIO_EXTI_Callback()	外部中断处理的回调函数，需要用户重新实现

1. 读取中断标志

在HAL库中，以“__HAL”为前缀的都是宏函数

```
#define __HAL_GPIO_EXTI_GET_FLAG(__EXTI_LINE__)
        (EXTI->PR & (__EXTI_LINE__))
```

它的功能就是检查外部中断挂起寄存器（EXTI_PR）中某个中断线的挂起标志位知否置位。

函数的返回值只要不等于0（用宏定义符号RESET表示），就表示外部中断线挂起标志位被置位，有未处理的中断事件。

2. 在某个外部中断线上产生软中断

```
#define __HAL_GPIO_EXTI_GENERATE_SWIT(__EXTI_LINE__)
        (EXTI->SWIER |= (__EXTI_LINE__))
```

它实际上就是将外部中断的软中断寄存器（EXTI_SWIER）中对应于中断线__EXTI_LINE__的位置1，以产生软件中断。

3.外部中断ISR函数以及中断处理回调函数

EXTI0到EXTI15的中断ISR函数具有相同的代码框架

```
void EXTI0_IRQHandler(void)
{
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0);
}

void EXTI9_5_IRQHandler(void)
{
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_5);
}

void EXTI15_10_IRQHandler(void)
{
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_11);
}
```

这些ISR函数都调用了函数HAL_GPIO_EXTI_IRQHandler()

HAL_GPIO_EXTI_IRQHandler()是外部中断处理通用函数

```
void HAL_GPIO_EXTI_IRQHandler(uint16_t GPIO_Pin)
{
  /* EXTI line interrupt detected */
  if(__HAL_GPIO_EXTI_GET_IT(GPIO_Pin) != RESET)    //检测中断标志
  {
    __HAL_GPIO_EXTI_CLEAR_IT(GPIO_Pin);          //清除中断挂起标志
    HAL_GPIO_EXTI_Callback(GPIO_Pin);            //执行回调函数
  }
}
```

实际的中断处理在回调函数HAL_GPIO_EXTI_Callback()里完成

这个回调函数在文件stm32f4xx_hal_gpio.c中有代码框架

```
__weak  void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  /* 使用UNUSED()函数避免编译时出现未使用变量的警告 */
  UNUSED(GPIO_Pin);
  /* 注意: 不要直接修改这个函数，如需使用回调函数，在用户文件中重新实现
     这个函数   */
}
```

修饰符__weak用来定义“弱”函数。如果在用户文件里重新实现了这些函数，就编译用户重新实现的函数。

弱函数一般用作中断处理的回调函数，例如这里的HAL_GPIO_EXTI_Callback()，如果用户重新实现了这个函数，就编译用户重新实现的这个函数。

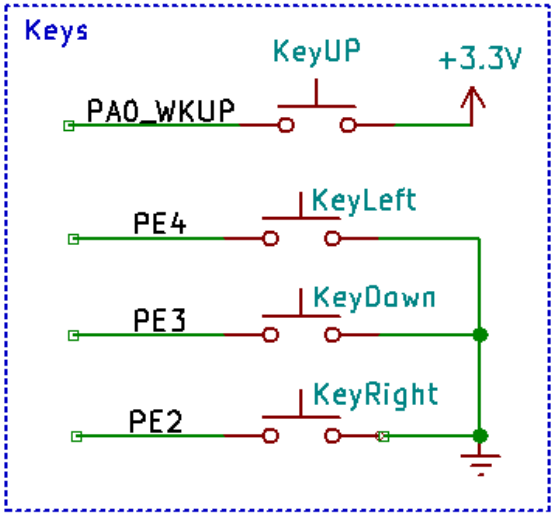
第7章 中断系统和外部中断

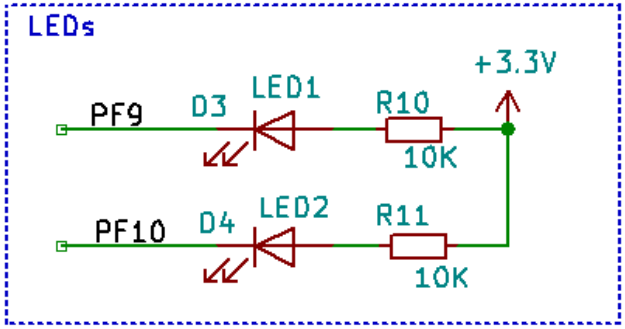
7.1 STM32F407的中断

7.2 外部中断EXTI

7.3 外部中断使用示例

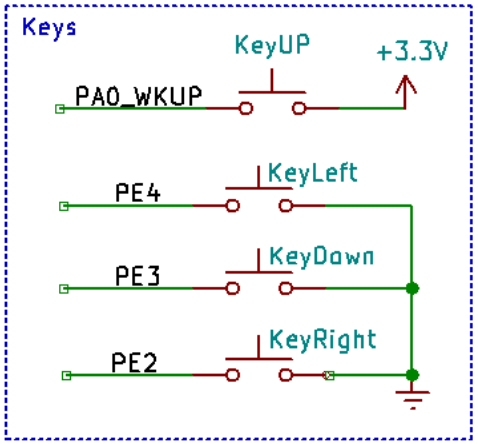
7.3.1 实例功能和CubeMX项目设置

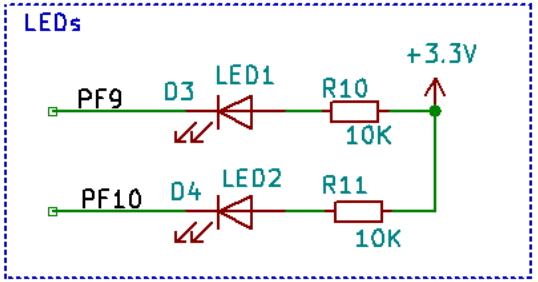




- 4个按键都采用外部中断方式输入
- KeyLeft使LED1输出翻转，KeyRigh使LED2输出翻转
- KeyUp使LED1和LED2输出翻转
- KeyDown使产生EXTI0软中断，模拟KeyUp按键按下

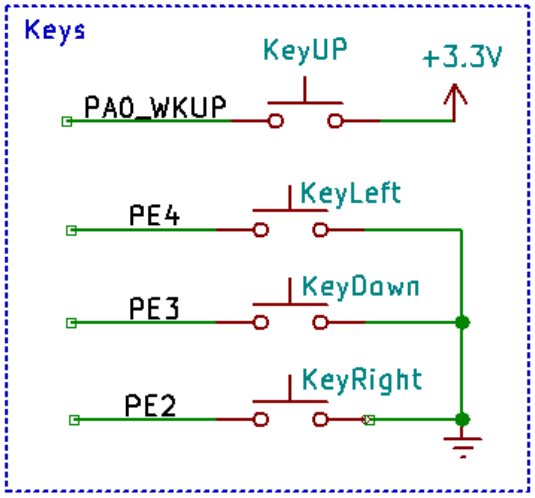
GPIO引脚以及外部中断优先级设置

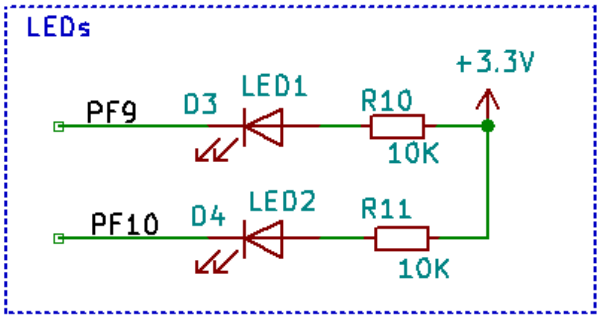




用户标签	引脚名称	引脚功能	GPIO模式	上拉或下拉	抢占优先级	次优先级
LED1	PF9	GPIO_Output	推挽输出			
LED2	PF10	GPIO_Output	推挽输出			
KeyRight	PE2	GPIO_EXTI2	下跳沿触发外部中断	上拉	2	0
KeyDown	PE3	GPIO_EXTI3	下跳沿触发外部中断	上拉	1	2
KeyLeft	PE4	GPIO_EXTI4	下跳沿触发外部中断	上拉	1	1
KeyUP	PA0	GPIO_EXTI0	上跳沿触发外部中断	下拉	1	0

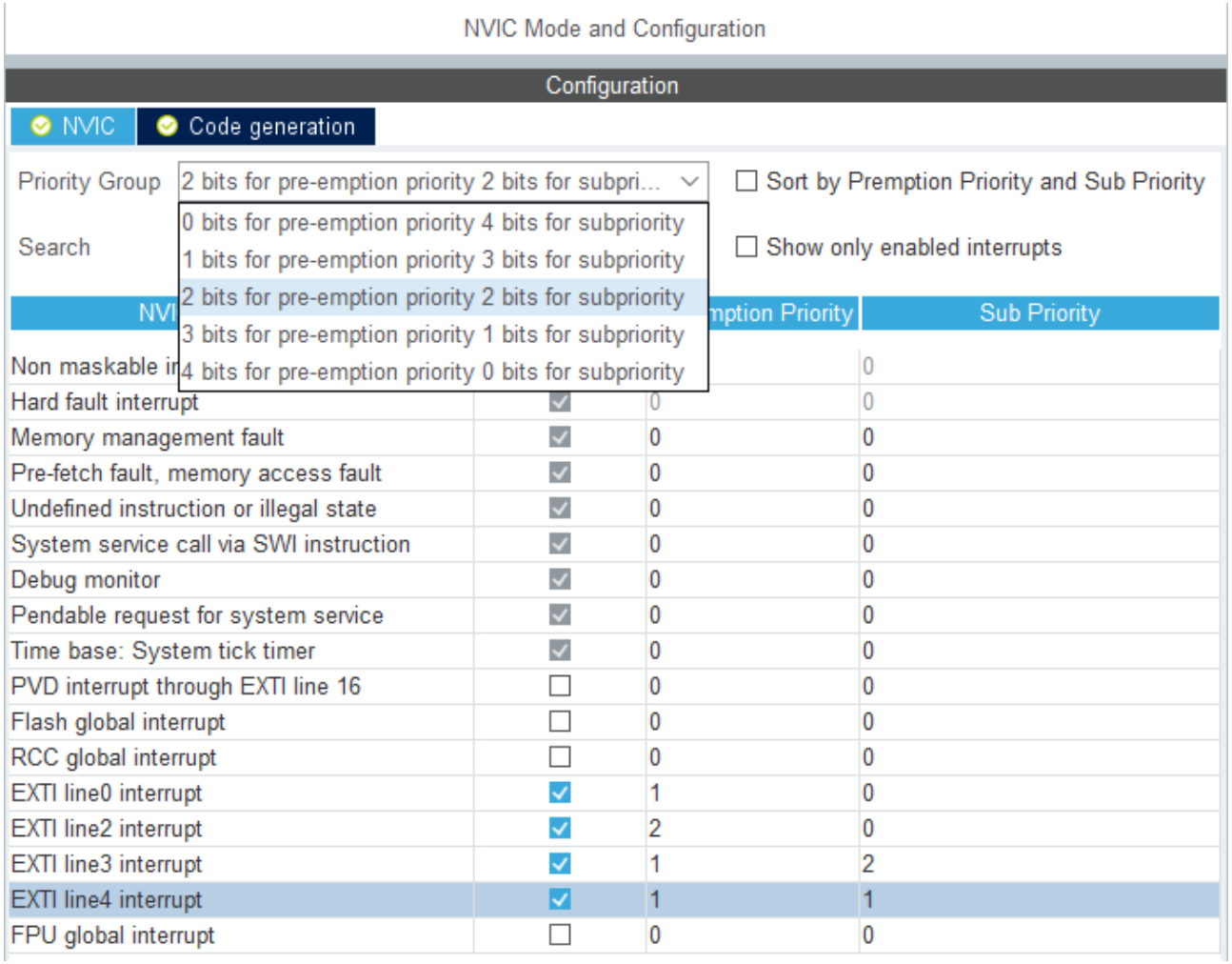
CubeMX中的GPIO设置





Pin Name	User Label	GPIO mode	GPIO Pull-up/Pull-down	GPIO output level
PA0-WKUP	KeyUp	External Interrupt Mode with Rising edge ...	Pull-down	n/a
PE2	KeyRight	External Interrupt Mode with Falling edge ...	Pull-up	n/a
PE3	KeyDown	External Interrupt Mode with Falling edge ...	Pull-up	n/a
PE4	KeyLeft	External Interrupt Mode with Falling edge ...	Pull-up	n/a
PF9	LED1	Output Push Pull	No pull-up and no pull-down	Low
PF10	LED2	Output Push Pull	No pull-up and no pull-down	Low

NVIC中设置中断优先级，注意外部中断的抢占优先级不能设置为0，因为SysTick中断优先级为0



7.3.2 项目初始代码分析

1. 主程序

在HAL_Init()函数中设置优先级分组策略。HAL_Init()里调用了一个弱函数HAL_MspInit()，在STM32CubeMX导出的项目中有一个文件stm32f4xx_hal_msp.c，在这个文件里重新实现了函数HAL_MspInit()，其代码如下：

```
void HAL_MspInit(void)
{
  __HAL_RCC_SYSCFG_CLK_ENABLE();
  __HAL_RCC_PWR_CLK_ENABLE();
  HAL_NVIC_SetPriorityGrouping(NVIC_PRIORITYGROUP_2);
}
```

由于在CubeMX中为LED和按键的引脚都定义了用户标签，在main.h中生成了如下的一些宏定义，在生成的代码中使用了这些宏，在自己编写的代码中也可以使用这些宏，这样可以使程序可读性更强，也便于移植到其他开发板。

```
/* 文件：main.h，  STM32CubeMX中定义GPIO引脚标签的宏定义 */
#define KeyRight_Pin                    GPIO_PIN_2
#define KeyRight_GPIO_Port              GPIOE
#define KeyRight_EXTI_IRQn              EXTI2_IRQn

#define KeyDown_Pin                     GPIO_PIN_3
#define KeyDown_GPIO_Port               GPIOE
#define KeyDown_EXTI_IRQn               EXTI3_IRQn

#define KeyLeft_Pin                     GPIO_PIN_4
#define KeyLeft_GPIO_Port               GPIOE
#define KeyLeft_EXTI_IRQn               EXTI4_IRQn
```

2. GPIO和EXTI中断初始化

文件gpio.c中的函数MX_GPIO_Init()实现GPIO引脚和EXTI中断的初始化。

```
/* EXTI 中断初始化设置 */
 HAL_NVIC_SetPriority(EXTI0_IRQn, 1, 0);    //设置中断优先级
 HAL_NVIC_EnableIRQ(EXTI0_IRQn);            //使能中断

 HAL_NVIC_SetPriority(EXTI2_IRQn, 2, 0);
 HAL_NVIC_EnableIRQ(EXTI2_IRQn);
```

看示例程序的完整源代码

3. EXTI中断的ISR函数

EXTI0至EXTI4都有独立的ISR函数，在文件stm32f4xx_it.c中自动生成了这4个ISR函数的代码框架

```
void EXTI0_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI0_IRQn 0 */

  /* USER CODE END EXTI0_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0);
  /* USER CODE BEGIN EXTI0_IRQn 1 */

  /* USER CODE END EXTI0_IRQn 1 */
}
```

7.3.3 编写用户功能代码

1. 重新实现外部中断回调函数

可以在任何一个源程序文件里重新实现外部中断的回调函数HAL_GPIO_EXTI_Callback()，且无需在头文件里申明函数原型。本示例在文件gpio.c中重新实现这个函数。

按键的外部中断处理程序功能是：

- 按下KeyUp键时使两个LED翻转输出
- 按下KeyRight键时使LED2输出翻转
- 按下KeyDown键时产生EXTI0软中断，模拟按键KeyUp按下
- 按下KeyLeftt键时使LED1输出翻转

下页是回调函数完整代码

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if  (GPIO_Pin == KeyUp_Pin)  //PA0=KeyUp, 使两个LED输出翻转
    {
        HAL_GPIO_TogglePin(LED1_GPIO_Port,LED1_Pin);
        HAL_GPIO_TogglePin(LED2_GPIO_Port,LED2_Pin);
        HAL_Delay(500);//软件消除按键抖动的影响
    }
    else if(GPIO_Pin == KeyRight_Pin) //PE2=KeyRight, 使LED2 输出翻转
    {
        HAL_GPIO_TogglePin(LED2_GPIO_Port,LED2_Pin);
        HAL_Delay(1000);  //软件消除按键抖动的影响,观察优先级的作用
    }
    else if (GPIO_Pin == KeyDown_Pin) //PE3=KeyDown，产生EXTI0 软中断
    {
        __HAL_GPIO_EXTI_GENERATE_SWIT(GPIO_PIN_0);//产生EXTI0 软中断
        HAL_Delay(1000); //这个延时也是必要的，否则由于按键抖动，会两次触发
    }
    else if (GPIO_Pin == KeyLeft_Pin) //PE4=KeyLeft，使LED1输出翻转
    {
        HAL_GPIO_TogglePin(LED1_GPIO_Port,LED1_Pin);
        HAL_Delay(1000);  //软件消除按键抖动的影响,观察优先级的作用
    }
}
```

2. 对函数HAL_GPIO_EXTI_IRQHandler()的代码改造

运行时却发现按键按下后的响应并不如预期，例如按下KeyUp键后，两个LED会亮灭两次，虽然已经加了延时进行按键消抖处理，但是还是有按键抖动的影响。

```
void HAL_GPIO_EXTI_IRQHandler(uint16_t GPIO_Pin)
{
  /* EXTI line interrupt detected */
  if(__HAL_GPIO_EXTI_GET_IT(GPIO_Pin) != RESET)    //检测中断标志
  {
    __HAL_GPIO_EXTI_CLEAR_IT(GPIO_Pin);           //清除中断挂起标志
    HAL_GPIO_EXTI_Callback(GPIO_Pin);             //执行回调函数
  }
}
```

其中两行代码有问题，调换一下顺序就可以了。

7.3.4 中断优先级的运行时测试

1. 抢占优先级不同

按键	抢占优先级	动作
KeyLeft	1	使LED1输出翻转
KeyRight	2	使LED2输出翻转

- 按下KeyLeft键后再快速按下KeyRight键，KeyRight键控制的LED2并不会立刻变化，需等待1秒后才变化。

这是因为KeyRight键中断的优先级低，只能等待KeyLeft键中断的ISR执行完之后才能执行KeyRight键的ISR。

抢占优先级不同

按键	抢占优先级	动作
KeyLeft	1	使LED1输出翻转
KeyRight	2	使LED2输出翻转

- 按下KeyRight键后快速再按下KeyLeft键，KeyLeft键控制的LED1会立刻变化。

这是因为KeyLeft键中断的优先级高于KeyRight键中断的优先级，会打断KeyRight键中断ISR的执行而立即去执行KeyLeft键中断的ISR。

2. 抢占优先级相同

按键	抢占优先级	次优先级	动作
KeyUp	1	0	使LED1和LED2输出翻转
KeyDown	1	2	产生EXTI0软中断，即KeyUp的动作

运行时会发现：按下KeyUp键两个LED立刻翻转变化，按下KeyDown键一秒钟后两个LED才翻转变化，而不是立刻变化。

这是因为2个中断抢占优先级相同时，不能发生抢占。

抢占优先级相同

按下KeyUp键时（EXTI0中断），执行的代码是：

```
HAL_GPIO_TogglePin(LED1_GPIO_Port,LED1_Pin);
HAL_GPIO_TogglePin(LED2_GPIO_Port,LED2_Pin);
HAL_Delay(500);                    //软件消除按键抖动的影响
```

按下KeyDown键时（EXTI3中断），执行的代码是：

```
__HAL_GPIO_EXTI_GENERATE_SWIT(GPIO_PIN_0); //产生EXTI0 软中断
HAL_Delay(1000);
```

KeyDown键按下时，因为EXTI0和EXTI3的抢占优先级相同，所以不能立刻执行EXTI0的中断响应代码，而是要等EXTI3的响应代码执行完（有延时1000ms）后再去执行EXTI0的响应代码。

练习任务

1. 使用按键控制蜂鸣器，按下KeyLeft键BEEP输出1，按下KeyRight键BEEP输出0