《中断与定时器》实验报告

课程名称： 嵌入式系统设计	年级：23 级	上机实践成绩：
指导教师：郭建	姓名： 张建夫	
上机实践名称：	学号：10235101477	上机实践日期：
中断与定时器		2025/04/22
上机实践编号：	组号：	上机实践时间：
		14：50~16：30

一、 目的与要求

掌握如何对按键进行检测，且控制不同按键点亮不同的 LED 灯

了解中断的基本知识

掌握中断编程的具体流程，并能使用中断检测按键

掌握 STM32 定时器相关库函数的配置；

学会使用定时器进行定时并触发中断

二、 内容与实验原理

1. 实验内容：

了解中断的基础知识

掌握中断编程过程，并能用中断检测按键

定时器及中断相关库函数的配置

使用定时器控制 LED 灯亮灭的间隔时间

相关寄存器值，调整定时时间

2. 实验原理：

（1）中断基本知识：

F429 在内核水平上搭载了一个异常响应系统， 支持为数众多的系统异常和外部中断。

其中系统异常有 10 个，外部中断有 91 个。除了个别异常的优先级被绑定外，其它异常的优先级都是可编程的。

有关具体的系统异常和外部中断可在标准库文件 stm32f4xx.h 这个头文件查询到，在 IRQn_Type 这个结构体里面包含了 F4 系列全部的异常声明。（希望大家打开文件亲自查看）

其中 F4 系列的全部异常声明如下：

```
typedef enum IRQn
{
/******  Cortex-M4 Processor Exceptions Numbers ***********************************************************/
  NonMaskableInt_IRQn        = -14,    /*!< 2 Non Maskable Interrupt                                */
  MemoryManagement_IRQn      = -12,    /*!< 4 Cortex-M4 Memory Management Interrupt                 */
  BusFault_IRQn              = -11,    /*!< 5 Cortex-M4 Bus Fault Interrupt                         */
  UsageFault_IRQn            = -10,    /*!< 6 Cortex-M4 Usage Fault Interrupt                       */
  SVCall_IRQn                = -5,     /*!< 11 Cortex-M4 SV Call Interrupt                          */
  DebugMonitor_IRQn          = -4,     /*!< 12 Cortex-M4 Debug Monitor Interrupt                    */
  PendSV_IRQn                = -2,     /*!< 14 Cortex-M4 Pend SV Interrupt                          */
  SysTick_IRQn               = -1,     /*!< 15 Cortex-M4 System Tick Interrupt                      */
/******  STM32 specific Interrupt Numbers *****************************************************************/
  WWDG_IRQn                  = 0,      /*!< Window WatchDog Interrupt                               */
  PVD_IRQn                   = 1,      /*!< PVD through EXTI Line detection Interrupt               */
  TAMP_STAMP_IRQn            = 2,      /*!< Tamper and TimeStamp interrupts through the EXTI line   */
  RTC_WKUP_IRQn              = 3,      /*!< RTC Wakeup interrupt through the EXTI line              */
  FLASH_IRQn                 = 4,      /*!< FLASH global Interrupt                                  */
  RCC_IRQn                   = 5,      /*!< RCC global Interrupt                                    */
  EXTI0_IRQn                 = 6,      /*!< EXTI Line0 Interrupt                                    */
  EXTI1_IRQn                 = 7,      /*!< EXTI Line1 Interrupt                                    */
  EXTI2_IRQn                 = 8,      /*!< EXTI Line2 Interrupt                                    */
  EXTI3_IRQn                 = 9,      /*!< EXTI Line3 Interrupt                                    */
  EXTI4_IRQn                 = 10,     /*!< EXTI Line4 Interrupt                                    */
  DMA1_Stream0_IRQn          = 11,     /*!< DMA1 Stream 0 global Interrupt                          */
  DMA1_Stream1_IRQn          = 12,     /*!< DMA1 Stream 1 global Interrupt                          */
  DMA1_Stream2_IRQn          = 13,     /*!< DMA1 Stream 2 global Interrupt                          */
  DMA1_Stream3_IRQn          = 14,     /*!< DMA1 Stream 3 global Interrupt                          */
  DMA1_Stream4_IRQn          = 15,     /*!< DMA1 Stream 4 global Interrupt                          */
  DMA1_Stream5_IRQn          = 16,     /*!< DMA1 Stream 5 global Interrupt                          */
  DMA1_Stream6_IRQn          = 17,     /*!< DMA1 Stream 6 global Interrupt                          */
  ADC_IRQn                   = 18,     /*!< ADC1, ADC2 and ADC3 global Interrupts                   */

#if defined(STM32F429_439xx)
  CAN1_TX_IRQn               = 19,     /*!< CAN1 TX Interrupt                                       */
  CAN1_RX0_IRQn              = 20,     /*!< CAN1 RX0 Interrupt                                      */
  CAN1_RX1_IRQn              = 21,     /*!< CAN1 RX1 Interrupt                                      */
  CAN1_SCE_IRQn              = 22,     /*!< CAN1 SCE Interrupt                                      */
  EXTI9_5_IRQn               = 23,     /*!< External Line[9:5] Interrupts                           */
  TIM1_BRK_TIM9_IRQn         = 24,     /*!< TIM1 Break interrupt and TIM9 global interrupt          */
  TIM1_UP_TIM10_IRQn         = 25,     /*!< TIM1 Update Interrupt and TIM10 global interrupt        */
  TIM1_TRG_COM_TIM11_IRQn    = 26,     /*!< TIM1 Trigger and Commutation Interrupt and TIM11 global interrupt */
  TIM1_CC_IRQn               = 27,     /*!< TIM1 Capture Compare Interrupt                          */
  TIM2_IRQn                  = 28,     /*!< TIM2 global Interrupt                                   */
  TIM3_IRQn                  = 29,     /*!< TIM3 global Interrupt                                   */
  TIM4_IRQn                  = 30,     /*!< TIM4 global Interrupt                                   */
  I2C1_EV_IRQn               = 31,     /*!< I2C1 Event Interrupt                                    */
  I2C1_ER_IRQn               = 32,     /*!< I2C1 Error Interrupt                                    */
  I2C2_EV_IRQn               = 33,     /*!< I2C2 Event Interrupt                                    */
  I2C2_ER_IRQn               = 34,     /*!< I2C2 Error Interrupt                                    */
  SPI1_IRQn                  = 35,     /*!< SPI1 global Interrupt                                   */
  SPI2_IRQn                  = 36,     /*!< SPI2 global Interrupt                                   */
  USART1_IRQn                = 37,     /*!< USART1 global Interrupt                                 */
  USART2_IRQn                = 38,     /*!< USART2 global Interrupt                                 */
  USART3_IRQn                = 39,     /*!< USART3 global Interrupt                                 */
  EXTI15_10_IRQn             = 40,     /*!< External Line[15:10] Interrupts                         */
  RTC_Alarm_IRQn             = 41,     /*!< RTC Alarm (A and B) through EXTI Line Interrupt         */
  OTG_FS_WKUP_IRQn           = 42,     /*!< USB OTG FS Wakeup through EXTI line interrupt           */
  TIM8_BRK_TIM12_IRQn        = 43,     /*!< TIM8 Break Interrupt and TIM12 global interrupt         */
  TIM8_UP_TIM13_IRQn         = 44,     /*!< TIM8 Update Interrupt and TIM13 global interrupt        */
  TIM8_TRG_COM_TIM14_IRQn    = 45,     /*!< TIM8 Trigger and Commutation Interrupt and TIM14 global interrupt */
  TIM8_CC_IRQn               = 46,     /*!< TIM8 Capture Compare Interrupt                          */
  DMA1_Stream7_IRQn          = 47,     /*!< DMA1 Stream7 Interrupt                                  */
  FMC_IRQn                   = 48,     /*!< FMC global Interrupt                                    */
  SDIO_IRQn                  = 49,     /*!< SDIO global Interrupt                                   */
  TIM5_IRQn                  = 50,     /*!< TIM5 global Interrupt                                   */
  SPI3_IRQn                  = 51,     /*!< SPI3 global Interrupt                                   */
  UART4_IRQn                 = 52,     /*!< UART4 global Interrupt                                  */
  UART5_IRQn                 = 53,     /*!< UART5 global Interrupt                                  */
  TIM6_DAC_IRQn              = 54,     /*!< TIM6 global and DAC1&2 underrun error  interrupts       */
  TIM7_IRQn                  = 55,     /*!< TIM7 global interrupt                                   */
  DMA2_Stream0_IRQn          = 56,     /*!< DMA2 Stream 0 global Interrupt                          */
  DMA2_Stream1_IRQn          = 57,     /*!< DMA2 Stream 1 global Interrupt                          */
  DMA2_Stream2_IRQn          = 58,     /*!< DMA2 Stream 2 global Interrupt                          */
```

```
  DMA2_Stream3_IRQn           = 59,     /*!< DMA2 Stream 3 global Interrupt                    */
  DMA2_Stream4_IRQn           = 60,     /*!< DMA2 Stream 4 global Interrupt                    */
  ETH_IRQn                    = 61,     /*!< Ethernet global Interrupt                         */
  ETH_WKUP_IRQn               = 62,     /*!< Ethernet Wakeup through EXTI line Interrupt       */
  CAN2_TX_IRQn                = 63,     /*!< CAN2 TX Interrupt                                 */
  CAN2_RX0_IRQn               = 64,     /*!< CAN2 RX0 Interrupt                                */
  CAN2_RX1_IRQn               = 65,     /*!< CAN2 RX1 Interrupt                                */
  CAN2_SCE_IRQn               = 66,     /*!< CAN2 SCE Interrupt                                */
  OTG_FS_IRQn                 = 67,     /*!< USB OTG FS global Interrupt                       */
  DMA2_Stream5_IRQn           = 68,     /*!< DMA2 Stream 5 global interrupt                    */
  DMA2_Stream6_IRQn           = 69,     /*!< DMA2 Stream 6 global interrupt                    */
  DMA2_Stream7_IRQn           = 70,     /*!< DMA2 Stream 7 global interrupt                    */
  USART6_IRQn                 = 71,     /*!< USART6 global interrupt                           */
  I2C3_EV_IRQn                = 72,     /*!< I2C3 event interrupt                              */
  I2C3_ER_IRQn                = 73,     /*!< I2C3 error interrupt                              */
  OTG_HS_EP1_OUT_IRQn         = 74,     /*!< USB OTG HS End Point 1 Out global interrupt       */
  OTG_HS_EP1_IN_IRQn          = 75,     /*!< USB OTG HS End Point 1 In global interrupt        */
  OTG_HS_WKUP_IRQn            = 76,     /*!< USB OTG HS Wakeup through EXTI interrupt          */
  OTG_HS_IRQn                 = 77,     /*!< USB OTG HS global interrupt                       */
  DCMI_IRQn                   = 78,     /*!< DCMI global interrupt                             */
  CRYP_IRQn                   = 79,     /*!< CRYP crypto global interrupt                      */
  HASH_RNG_IRQn               = 80,     /*!< Hash and Rng global interrupt                     */
  FPU_IRQn                    = 81,     /*!< FPU global interrupt                              */
  UART7_IRQn                  = 82,     /*!< UART7 global interrupt                            */
  UART8_IRQn                  = 83,     /*!< UART8 global interrupt                            */
  SPI4_IRQn                   = 84,     /*!< SPI4 global Interrupt                             */
  SPI5_IRQn                   = 85,     /*!< SPI5 global Interrupt                             */
  SPI6_IRQn                   = 86,     /*!< SPI6 global Interrupt                             */
  SAI1_IRQn                   = 87,     /*!< SAI1 global Interrupt                             */
  LTDC_IRQn                   = 88,     /*!< LTDC global Interrupt                             */
  LTDC_ER_IRQn                = 89,     /*!< LTDC Error global Interrupt                       */
  DMA2D_IRQn                  = 90      /*!< DMA2D global Interrupt                            */
#endif /* STM32F429_439xx */
```

前八个为处理器异常，可以看出，这八个异常是所有异常中优先级最高的。

（2）NVIC 介绍：

- NVIC 是嵌套向量中断控制器，控制着整个芯片中断相关的功能，它跟内核紧密耦合，是内核里面的一个外设。
- 在 NVIC 有一个专门的寄存器：中断优先级寄存器 NVIC_IPRx（在 F429 中x=0...90），用来配置外部中断的优先级， IPR 宽度为 8bit，原则上每个外部中断可配置的优先级为0~255，数值越小，优先级越高。
- 实际用于表达优先级的只有高 4bit，这4bit又被分组成抢占优先级和子优先级。如有多个中断同时响应
  - ✓ 抢占优先级高的就会抢占优先级低的，其优先得到执行，
  - ✓ 如果抢占优先级相同，就比较子优先级。
  - ✓ 如果抢占优先级和子优先级都相同的话，就比较他们的硬件中断编号，编号越小，优先级越高。

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
用于表达优先级				未使用，读回为 0			

其中抢占优先级和子优先级需要在 NVIC 中先初始化才能设置中断，其中由于这两个中断共享四位，可以根据这两种中断又分成五组优先级：

优先级分组真值表

优先级分组	主优先级	子优先级	描述
NVIC_PriorityGroup_0	0	0-15	主-0bit，子-4bit
NVIC_PriorityGroup_1	0-1	0-7	主-1bit，子-3bit
NVIC_PriorityGroup_2	0-3	0-3	主-2bit，子-2bit
NVIC_PriorityGroup_3	0-7	0-1	主-3bit，子-1bit
NVIC_PriorityGroup_4	0-15	0	主-4bit，子-0bit

（3）中断编程：

由于对应的引脚需要使用对应的中断线进行处理，在编程前要搞清楚中断和引脚的对应关系，二者以及对应的中断处理函数如下：

GPIO引脚	中断标志位	中断处理函数
PA0~PH0	EXTI0	EXTI0_IRQHandler
PA1~PH1	EXTI1	EXTI1_IRQHandler
PA2~PH2	EXTI2	EXTI2_IRQHandler
PA3~PH3	EXTI3	EXTI3_IRQHandler
PA4~PH4	EXTI4	EXTI4_IRQHandler
PA5~PH5	EXTI5	EXTI9_5_IRQHandler
PA6~PH6	EXTI6	
PA7~PH7	EXTI7	
PA8~PH8	EXTI8	
PA9~PH9	EXTI9	
PA10~PH10	EXTI10	EXTI15_10_IRQHandler
PA11~PH11	EXTI11	
PA12~PH12	EXTI12	
PA13~PH13	EXTI13	
PA14~PH14	EXTI14	
PA15~PH15	EXTI15	

其中 EXIT9_5_TRQHandler 对应中断线 5 到 9 的中断处理函数，EXIT15_10 同理。

除了上面的中断线以外，还有 7 根特殊的中断线：

- EXTI 线 16 连接到 PVD 输出
- EXTI 线 17 连接到 RTC 闹钟事件
- EXTI 线 18 连接到 USB OTG FS 唤醒事件
- EXTI 线 19 连接到以太网唤醒事件
- EXTI 线 20 连接到 USB OTG HS（在 FS 中配置）唤醒事件
- EXTI 线 21 连接到 RTC 入侵和时间戳事件
- EXTI 线 22 连接到 RTC 唤醒事件

三、 使用环境

调用 dxdiag 工具：

Operating System: Windows 11 家庭中文版 64-bit (10.0, Build 22621) (22621.ni_release.220506-1250)
Language: Chinese (Simplified) (Regional Setting: Chinese (Simplified))
System Manufacturer: HP
System Model: HP Pavilion Aero Laptop 13-be2xxx
BIOS: F.13 (type: UEFI)
Processor: AMD Ryzen 5 7535U with Radeon Graphics (12 CPUs), ~2.9GHz
Memory: 16384MB RAM
Available OS Memory: 15574MB RAM
Page File: 27604MB used, 5685MB available
Windows Dir: C:\WINDOWS
DirectX Version: DirectX 12
DX Setup Parameters: Not found
User DPI Setting: 144 DPI (150 percent)

System DPI Setting: 192 DPI (200 percent)
DWM DPI Scaling: UnKnown
Miracast: Available, with HDCP
Microsoft Graphics Hybrid: Not Supported

四、主要实验内容和结果展示

本次实验使用的是上节课的模板，需要改动的地方较多，尤其是 bsp_key.c 和 bsp_key.h，在这个实验里我将这两个文件作为 bsp_exit.c 和 bsp_exit.h 进行修改，包括按键的初始化等操作。

1. 示例实验：

按照 ppt 上的教程，先要在头文件（bsp_exit.h，我没有改上一次课的文件名，实际上是 bsp_key.h）对按键进行映射：

```
6  //按键初始化
7  #define KEY1_INT_GPIO_PIN          GPIO_Pin_0
8  #define KEY1_INT_GPIO_PORT         GPIOA
9  #define KEY1_INT_GPIO_CLK          RCC_AHB1Periph_GPIOA
10
11 #define KEY1_INT_EXTI_LINE         EXTI_Line0
12 #define KEY1_INT_EXTI_PORTSOURCE   EXTI_PortSourceGPIOA
13 #define KEY1_INT_EXTI_PINSOURCE    EXTI_PinSource0
14 #define KEY1_INT_EXTI_IRQ          EXTI0_IRQn
15 #define KEY1_IRQHANDLER            EXTI0_IRQHandler
16
17
18 void EXTI_Key_Config(void);
```

其中由于按键 key1 对应的 GPIO 引脚为 0，端口为 A，根据前一部分的引脚对应关系图，应选择中断线 0，中断线的源端口应选择 A，对应的中断处理函数为中断线 0 处理函数，另外，还需要写上中断和按键引脚的初始化函数的声明，在.c 文件中写实现。

下面是该初始化函数的实现：

```c
void EXTI_Key_Config(void)
{
	GPIO_InitTypeDef GPIO_InitStructure;

	EXTI_InitTypeDef  EXTI_InitStruct;

	/* 开启gpio的时钟 */
	RCC_AHB1PeriphClockCmd ( KEY1_INT_GPIO_CLK, ENABLE);

	/* 使能SYSCFG时钟 */
	RCC_APB2PeriphClockCmd ( RCC_APB2Periph_SYSCFG, ENABLE);

	/* 配置NVIC */
	NVIC_Configuration();

	/* 选择key1 */
	GPIO_InitStructure.GPIO_Pin =  KEY1_INT_GPIO_PIN;

	/* 设置引脚为输入模式 */
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;

	/* 既不上拉也不下拉*/
	GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;

	/* 初始化按键 */
	GPIO_Init(KEY1_INT_GPIO_PORT, &GPIO_InitStructure);

	/* 连接 EXIT 中断源到key1的引脚 */
	SYSCFG_EXTILineConfig(KEY1_INT_EXTI_PORTSOURCE, KEY1_INT_EXTI_PINSOURCE);

	/* 选择 EXIT 中断源 */
	EXTI_InitStruct.EXTI_Line = KEY1_INT_EXTI_LINE;

	/* 中断模式 */
	EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;

	/* 上升沿触发 */
	EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Rising;

	/* 使能中断线 */
	EXTI_InitStruct.EXTI_LineCmd = ENABLE;
	EXTI_Init(&EXTI_InitStruct);
}
```

除了上节课对 GPIO 端口初始化外，我们加上了对中断线的初始化，以及让 GPIO 端口映射到中断线的代码，其中最需要关注的是中断控制器的初始化（NVIC）函数：

```c
static void NVIC_Configuration(void)
{
	NVIC_InitTypeDef NVIC_InitStruct;

	NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);

	NVIC_InitStruct.NVIC_IRQChannel = KEY1_INT_EXTI_IRQ;
	NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 1;
	NVIC_InitStruct.NVIC_IRQChannelSubPriority = 1;
	NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
	NVIC_Init(&NVIC_InitStruct);
}
```

该函数对 NVIC 的四个要素（中断源 IRQChannel， 抢占优先级 Preemption，子优先级 Sub，中断使能 IRQChannelCmd）进行了初始化，当时看到 ppt 提供的函数时，我很疑惑为什么该函数加了个 static 的静态变量，便查了一下，发现该函数实际上只有该文件使用，如果其他文件初始化了别的 NVIC（使用相同的函数名），不加 static，就会链接错误。

接着就需要编写中断处理函数：

```c
void KEY1_IRQHANDLER(void)
{
	// 确保产生了EXIT_LINE中断
    if( ( EXTI_GetITStatus(KEY1_INT_EXTI_LINE) ) != RESET )
	{
		LED1_TOGGLE;
		EXTI_ClearITPendingBit(KEY1_INT_EXTI_LINE);
	}
}
```

该函数在 stm32f4xx_it.c 中编写，虽然中断函数在

startup_stm32f429_439xx.s 中已有，但是是空实现，而自己实现的中断处理函数一般写在 stm32f4xx_it.c 中。

这个函数中的 EXTI_GetITStatus 用来获取 EXTI 0 的中断标志位状态，用于检查是否真的产生了中断，而 EXTI_ClearITPendingBit 用于清除标志位，表示本次中断处理完毕。

最后就是主函数的编写：

```
29  int main(void)
30  {
31      LED_GPIO_Config();
32      EXTI_Key_Config();
33      while(1){}
34  }
35
36
```

显然，主函数只需要在所有东西初始化（led 灯和中断）后忙等中断发生即可。

演示视频见提交文件。

2. 利用中断服务，通过 key2 按键，来控制红灯、绿灯、蓝灯的轮流亮。即按一次按键 2，红灯亮，在按一次，绿灯亮、在按一次，蓝灯亮，在按一次红灯亮……：

该题要求使用 key2 按键，首先要先改变 bsp_key.h 中的宏定义，刚开始做时，我并没有意识到中断线和引脚端口的强对应性，没有改中断线编号：

```
7   #define KEY2_INT_GPIO_PIN          GPIO_Pin_13
8   #define KEY2_INT_GPIO_PORT         GPIOC
9   #define KEY2_INT_GPIO_CLK          RCC_AHB1Periph_GPIOC
10
11  #define KEY2_INT_EXTI_LINE         EXTI_Line0
12  #define KEY2_INT_EXTI_PORTSOURCE   EXTI_PortSourceGPIOC
13  #define KEY2_INT_EXTI_PINSOURCE    EXTI_PinSource0
14  #define KEY2_INT_EXTI_IRQ          EXTI0_IRQn
15  #define KEY2_IRQHANDLER            EXTI0_IRQHandler
16
```

使我不论怎么 debug 都无法使用 key2 控制灯的亮灭，后面又仔细看了一遍 ppt，才发现引脚和中断线是有对应关系的，改了过来：

```
6   //按键初始化
7   #define KEY2_INT_GPIO_PIN          GPIO_Pin_13
8   #define KEY2_INT_GPIO_PORT         GPIOC
9   #define KEY2_INT_GPIO_CLK          RCC_AHB1Periph_GPIOC
10
11  #define KEY2_INT_EXTI_LINE         EXTI_Line13
12  #define KEY2_INT_EXTI_PORTSOURCE   EXTI_PortSourceGPIOC
13  #define KEY2_INT_EXTI_PINSOURCE    EXTI_PinSource13
14  #define KEY2_INT_EXTI_IRQ          EXTI15_10_IRQn
15  #define KEY2_IRQHANDLER            EXTI15_10_IRQHandler
```

EXIT15_10_TRQHandler 表示的是中断线 10 到 15 的中断处理函数。

另外，为了使红绿蓝交替亮，而所有关于灯的变量的定义都位于 bsp_led.c 中，我便在其中设置了一个 int 变量 cur_led 表示当前 led 的颜色，同时软编码了 led 的红绿蓝颜色，方便后面写代码：

```
20  int cur_led = 0;

32  #define red 0
33  #define green 1
34  #define blue 2
```

（这三条语句写在 bsp_led.h 中）

接着修改中断服务函数如下：

```
void KEY2_IRQHANDLER(void)
{
        // 确保产生了EXIT_LINE中断
    if( ( EXTI_GetITStatus(KEY2_INT_EXTI_LINE) ) != RESET )
        {
            switch(cur_led){
                case red:
                    LED_RED;
                break;
                case green:
                    LED_GREEN;
                break;
                case blue:
                    LED_BLUE;
                break;
                default:
                    break;
            }
            cur_led = (cur_led + 1) % 3;
        EXTI_ClearITPendingBit(KEY2_INT_EXTI_LINE);
        }
}
```

通过每次中断发生时改变为 cur_led 的颜色（switch 语句），并增加 cur_led（后面的赋值语句）的值实现红绿蓝的循环亮。

在实现的过程中由于对 c 语言引用外部变量不熟悉，在第一次编译时得到如下报错：

```
linking...
..\..\Output\按键.axf: Error: L6200E: Symbol cur_led multiply defined (by stm32f4xx_it.o and main.o).
..\..\Output\按键.axf: Error: L6200E: Symbol cur_led multiply defined (by bsp_led.o and main.o).
Not enough information to list image symbols.
```

（我第一次使用的时候是将 cur_led 的定义写在了 bsp_key.h 中）

后面查了相关资料发现还要在中断函数所在文件（stm32f4xx_it.c）里加上

```
extern int cur_led;
```

并将头文件的定义挪到源文件里，至此，实验成功。

演示视频见提交文件。

3. 利用中断机制，使得红灯 0.5s 翻转一次：

   此处需要注意的是，ppt 所给代码实际上是 tim6 的中断实现，应改为 tim7 的中断实现。

   首先是定时器的头文件定义：

```
#define BASIC_TIM                   TIM7
#define BASIC_TIM_CLK               RCC_APB1Periph_TIM7

#define BASIC_TIM_IRQn          TIM7_IRQn
#define BASIC_TIM_IRQHandler    TIM7_IRQHandler

void TIMx_Configuration(void);
```

相比于按键的头文件定义要简洁不少，第一行表示使用的是基本定时器 tim7（基本定时器只有两个，tim6 和 tim7），第二行表示的是 tim7 使用的时钟源， 第三行则是定时器的中断源，第四行是定时器的中断处理函数，最后一行是定时器中断的初始化函数声明。

在改成 tim7 时，由于对 tim7 中断函数的名字不熟悉，产生如下报错：

```
.\..\User\tim\bsp_basic_tim.c(6): error:  #20: identifier "TIM7_DAC_IRQn" is undefined
    NVIC_InitStructure.NVIC_IRQChannel = BASIC_TIM_IRQn;
.\..\User\tim\bsp_basic_tim.c: 0 warnings, 1 error
```

后面查看库函数定义才发现 tim7 的中断函数名字里并没有 DAC，改后即可运行。

接着是具体的初始化部分：

```
static void TIMx_NVIC_Configuration(void)
{   NVIC_InitTypeDef NVIC_InitStructure;
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);

    NVIC_InitStructure.NVIC_IRQChannel = BASIC_TIM_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 3;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}

static void TIM_Mode_Config(void)
{
        TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
        RCC_APB1PeriphClockCmd(BASIC_TIM_CLK, ENABLE);
        TIM_TimeBaseStructure.TIM_Period = 5000-1;
        TIM_TimeBaseStructure.TIM_Prescaler = 9000-1;
        TIM_TimeBaseInit(BASIC_TIM, &TIM_TimeBaseStructure);
        TIM_ClearFlag(BASIC_TIM, TIM_FLAG_Update);
        TIM_ITConfig(BASIC_TIM,TIM_IT_Update,ENABLE);
        TIM_Cmd(BASIC_TIM, ENABLE);
}
void TIMx_Configuration(void)
{
        TIMx_NVIC_Configuration();
        TIM_Mode_Config();
}
```

由于定时器中断和按键中断都是中断，二者都初始化了中断控制 NVIC，与按键中断不同的是，定时器中断使用的中断优先组号为 0，表示不能进行中断的嵌套。

其他需要解释的是定时器的预分频概念，用来决定定时器隔多久发出一次中断，对应到代码中的这两行：

```
        TIM_TimeBaseStructure.TIM_Period = 5000-1;
        TIM_TimeBaseStructure.TIM_Prescaler = 9000-1;
```

由于时钟源频率为 90MHz，即一个时钟持续 1/90M 秒，为了得到先得到 100 微秒（ppt 中为 100 微秒，便沿用）的间隔（即预分频，先分一次频率，再根据这个频率得到具体的时间间隔），（100 微秒） 除以 （1/90M 秒） 得到 9000，即需要多少个时钟周期得到 100 微秒，我们的预分频就为 9000，同时由于时钟是从 0 开始计时，实际的初始化值应为 9000-1（当时看到代码时感觉非

常奇怪为什么是 8999，便查了一下资料），而为了得到 0.5s 的周期，需要 5000*100 微秒，所以分频的大小为 5000-1（减 1 的原因同上），这样就得到了 0.5s 的中断周期。

这个部分是我花时间最多的试图搞懂的一个部分，由于这个 TIM_Period 和 TIM_Prescaler 的值设置为 0~65535(都是 16 位寄存器)，这个分频不是随便设置的，当时我就在想预分频为什么不设置为 1ms，这样不就更方便设置秒级的数据了吗，后来发现这样预分频就得是 90000，超出了寄存器限制。

而中断函数部分与按键中断大同小异，只不过调用的检查中断位是否设置的函数以及清除标志位的函数不同：

```
154  void  BASIC_TIM_IRQHandler (void)
155 {
156         // 确保中断位设置
157         if ( TIM_GetITStatus( BASIC_TIM, TIM_IT_Update) != RESET )
158         {
159             LED1_TOGGLE;
160         TIM_ClearITPendingBit(BASIC_TIM , TIM_IT_Update);
161         }
162 }
163
```

接下来是主函数：

```
29  int main(void)
30 {
31      LED_GPIO_Config();
32      TIMx_Configuration();
33      while(1){}
34 }
35
```

将按键中断初始化换为定时器中断初始化即可。

演示视频见提交文件。

4. 参考所给项目，让白灯先亮 6s，后熄灭，此时紫灯亮 6s，后熄灭，在青灯亮 6s 后熄灭，如此往复。（自己测试是否正确可以拿出自己手机的秒表做一个近似估计）

这个实验和之前的实验 2 很像，有了实验 2 的经验，直接修改定时器的中断处理函数就行：

```
156  void  BASIC_TIM_IRQHandler (void)
157 {
158         // 确保中断位设置
159         if ( TIM_GetITStatus( BASIC_TIM, TIM_IT_Update) != RESET )
160         {
161                 switch(cur_led){
162                 case red:
163                     LED_WHITE;
164                 break;
165                 case green:
166                     LED_PURPLE;
167                 break;
168                 case blue:
169                     LED_CYAN;
170                 break;
171                 default:
172                     break;
173             }
174             cur_led = (cur_led + 1) % 3;
175         TIM_ClearITPendingBit(BASIC_TIM , TIM_IT_Update);
176         }
177 }
```

思路与实验 2 的完全一样，通过一个全局变量 cur_led 控制灯的颜色，在每次中断发生后更改 cur_led 的值，使下一次产生的颜色发生改变。

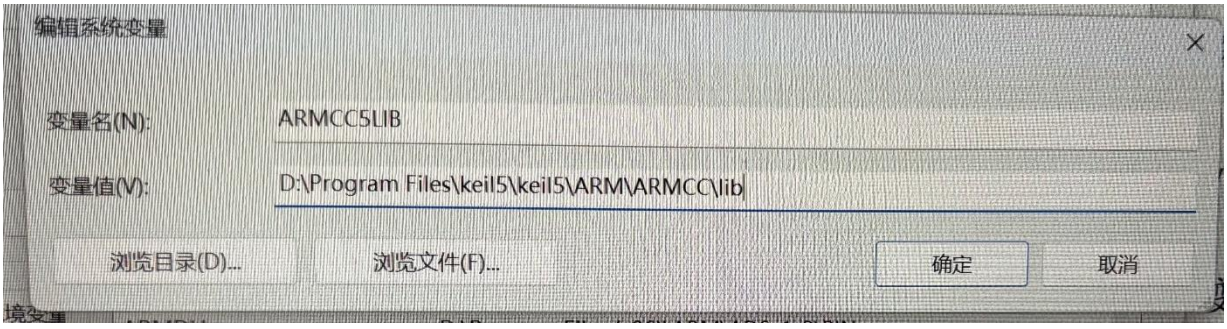
演示视频见提交文件（视频正好 18 秒，一个周期，白紫青）。

5. 其他

在上面的实验完成前，由于主机安装了 ads 环境，与 keil5 环境冲突，编译时有如下报错：

..\..\Output\按键.axf: Error: L6411E: No compatible library exists with a definition of startup symbol __main.

通过查找资料，在系统环境变量中将 ads 的系统路径删除并在系统变量中加上如下语句：



（打开高级系统设置后截图键的中断好像失效了，于是使用手机拍的）

即可解决该环境冲突错误。

五、实验总结

本次实验让我对硬件中断有了一个初步但较为全面的的认识，尤其是做实验的过程带来的各种问题，有一些是自己看不懂向网上找答案的，也有一些是对语言知识点的不熟悉导致的编程错误。我上次的实验试图做出一个中断相关的项目，但是没有做出来，这次算是理解了上次实验中自己的错误。除中断之外，我还学会了定时器的使用，以及定时器的预分频和设置时间间隔的原理，在反复研读 ppt 的过程中，也对 NVIC 这个中断控制器有了一定的了解，每个中断，无论是按键的还是定时器的，都需要初始化这个来实现中断，而且中断线和 GPIO 端口是有对应关系的，不能随便使用。