《按键检测》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：
按键检测		2025/03/25
上机实践编号：	组号：	上机实践时间：
		14：50~16：30

一、 目的与要求

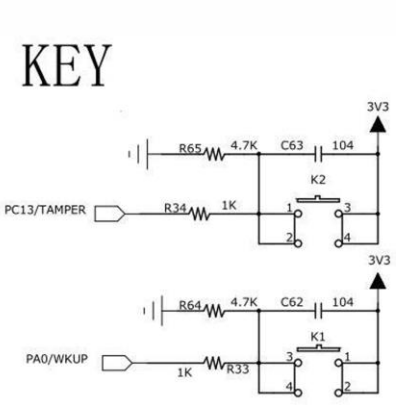
掌握如何对按键进行检测，且控制不同按键点亮不同的 LED 灯

二、 内容与实验原理

1. 实验内容：学习如何操作按键，并能用轮询方式检测按键

2. 实验原理：

（1）硬件设计：



从按键的原理图可知

- 按键在没有被按下的时候，GPIO引脚的输入状态为低电平(按键所在的电路不通，引脚接地)
- 当按键按下时，GPIO引脚的输入状态为高电平(按键所在的电路导通，引脚接到电源)。
- 只要我们检测引脚的输入电平，即可判断按键是否被按下。

（2）软件设计—编程要点：

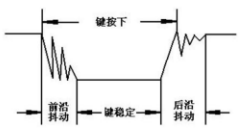
（1）使能GPIO端口时钟；

（2）初始化GPIO目标引脚为输入模式(引脚默认电平受按键电路影响，浮空/上拉/下拉均没有区别)；

（3） 编写简单测试程序，检测按键的状态，实现按键控制LED灯。

（3）软件设计—消抖：

- 按键机械触点断开、闭合时，由于触点的弹性作用，按键开关不会马上稳定接通或一下子断开，使用按键时会产生带波纹信号，需要用软件消抖处理滤波，不方便输入检测。



- 软件消抖：就是当检测到按键状态变化后，先等待一个延时时间，让抖动消失后再进行一次按键状态检测，如果与刚才检测到的状态相同，就可以确认按键已经稳定了。

- 硬件消抖：本实验板连接的按键带硬件消抖功能，它利用电容充放电的延时，消除了波纹，从而简化软件的处理，软件只需要直接检测引脚的电平即可。

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

前言：个人认为这次实验过于简单（给出的模板改两个字母就可以完成第二个任务了），于是我想是否能把这几次实验结合起来做些其他事情，于是便萌生了做一个智能红绿灯的想法，即该红绿灯能够通过按键控制，按下按键后，红绿灯会停在当前状态，即如果当前展现的是红灯，按下按键，红灯将暂时“卡住”，直到下次按下按键，红绿灯才重新交替闪烁。我在第 3 小节记录了这个过程。

1. 示例实验：

将示例工程打开并烧录，便可以实现 key1 控制红灯的亮灭，下面简单解释 key1 是如何控制红灯的亮灭。

首先要初始化 key1 按键：

```c
/**
  * @brief  配置按键用到的I/O口
  * @param  无
  * @retval 无
  */
void Key_GPIO_Config(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;

    /*开启按键GPIO口的时钟*/
    RCC_AHB1PeriphClockCmd(KEY1_GPIO_CLK|KEY2_GPIO_CLK,ENABLE);

  /*选择按键的引脚*/
    GPIO_InitStructure.GPIO_Pin = KEY1_PIN;

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

  /*设置引脚不上拉也不下拉*/
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;

  /*使用上面的结构体初始化按键*/
    GPIO_Init(KEY1_GPIO_PORT, &GPIO_InitStructure);

  /*选择按键的引脚*/
    GPIO_InitStructure.GPIO_Pin = KEY2_PIN;

  /*使用上面的结构体初始化按键*/
    GPIO_Init(KEY2_GPIO_PORT, &GPIO_InitStructure);
}
```

按键的初始化涉及 4 个方面，第一个是初始化按键 gpio 口的时钟，这里初始化了两个引脚的时钟（key1 和 key2），使用了“|”表示。第二个初始化按键的引脚为 key1 的引脚，这样就能控制对应的引脚。第三个初始化引脚的输入/输出模式，即该引脚是负责输入还是输出。第四个初始化引脚的上下拉情况。最后将初始化结构体传入库函数实现按键的初始化。

初始化按键后，我们还需要通过检测对应引脚状态来判断按键状态以此实现按键控制 led 灯的亮灭：

```c
/**
  * @brief   检测是否有按键按下
  * @param  GPIOx:具体的端口，x可以是（A...K）
  *   @param  GPIO_PIN:具体的端口位，可以是GPIO_PIN_x（x可以是0...15）
  * @retval  按键的状态
  *     @arg KEY_ON:按键按下
  *     @arg KEY_OFF:按键没按下
  */
uint8_t Key_Scan(GPIO_TypeDef* GPIOx,uint16_t GPIO_Pin)
{
    /*检测是否有按键按下 */
    if(GPIO_ReadInputDataBit(GPIOx,GPIO_Pin) == KEY_ON )
    {
        /*等待按键释放 */
        while(GPIO_ReadInputDataBit(GPIOx,GPIO_Pin) == KEY_ON);
        return  KEY_ON;
    }
    else
        return KEY_OFF;
}
```

这里的 key_Scan 函数会检测是否有按键按下，里面调用了关键函数 GPIO_ReadInputDataBit（一个库函数，读取当前对应引脚和端口的状态）来实际判断状态，同时这里用了忙等的方式使用户松开按键后才会返回结果。

需要注意的是，此处没有消抖处理是因为开发板的硬件已经做了消抖处理，无须额外的等待。

最后，使用上面编写的函数来实现按钮对红灯的控制：

```c
    /* 轮询按键状态，若按键按下则反转LED */
    while(1)
    {
        if( Key_Scan(KEY1_GPIO_PORT,KEY1_PIN) == KEY_ON  )
        {
            /*LED1反转*/
            LED1_TOGGLE;
        }
```

其中 LED1_TOGGLE 是提前定义好的宏，能够点亮红灯，这里就是不断轮询，查看用户是否按下按键，按下按键则反转红灯状态。

2. “key1 按键控制绿灯的亮灭，key2 按键控制蓝灯的亮灭”：

前面 key1 和 key2 的初始化代码无需变动，唯一要更改的部分就是 main 函数的轮询代码，将 main 函数的轮询改为两个灯并发轮询即可：

```
    /* 轮询按键状态，若按键按下则反转LED */
    while(1)
    {
        if( Key_Scan(KEY1_GPIO_PORT,KEY1_PIN) == KEY_ON  )
        {
            /*LED2反转*/
            LED2_TOGGLE;
        }

    if( Key_Scan(KEY2_GPIO_PORT,KEY2_PIN) == KEY_ON  )
        {
            /*LED3反转*/
            LED3_TOGGLE;
        }
    }
```

3. 探索“红绿灯”：

做这个项目的时候一波三折，刚开始想用已有的知识来做，但是一直没有想出来合适的解法，后面在网上查资料说要用中断，便转向使用中断实现，但是由于没有使用过中断，中间实现的过程一直报错，后面不了了之 ，直到一天后偶然又想了想，发现可以在不用中断的情况下实现，才成功完成。

使用中断：这个想法相当简单，主程序设定一个 paused 变量用来表示红绿灯是否停下，在按下按键后，产生外部中断，在中断处理函数中将 paused 变量取反（这样每次按下按键，状态就会切换），主函数中，在每个灯熄灭前，检查 paused 变量，如果 paused 变量置 1 了，就通过跳进空循环，不断检查下一次按键来实现冻结。

首先要更改 key1 的设置，使其能够相应中断：

```
//引脚定义
/*******************************************************/
#define KEY1_PIN                  GPIO_Pin_8
#define KEY1_GPIO_PORT            GPIOC
#define KEY1_GPIO_CLK             RCC_AHB1Periph_GPIOC

#define KEY1_INT_EXTI_LINE        EXTI_Line8
#define KEY1_INT_EXTI_PORTSOURCE  EXTI_PortSourceGPIOC
#define KEY1_INT_EXTI_PINSOURCE   EXTI_PinSource8
#define KEY1_INT_EXTI_IRQ         EXTI9_5_IRQn
#define KEY1_INT_EXTI_IRQHANDLER  EXTI9_5_IRQHandler
```

该处的代码是根据网上的代码照葫芦画瓢改的，设定了中断处理函数为中断线 EXTI5~9 共享的中断处理函数 EXTI9_5_IRQHandler 以及按键对应的中断线等初始化。

中断初始化函数如下：

```
static void EXTI_NVIC_Config(void)
{
        NVIC_InitTypeDef NVIC_InitStruct;

        NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);

        NVIC_InitStruct.NVIC_IRQChannel = KEY1_INT_EXTI_IRQ;
        NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
        NVIC_InitStruct.NVIC_IRQChannelSubPriority = 1;
        NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
        NVIC_Init(&NVIC_InitStruct);
}


void EXTI_Key_Config(void)
{
        /*定义一个GPIO_InitTypeDef类型的结构体*/
        GPIO_InitTypeDef GPIO_InitStructure;

        EXTI_InitTypeDef  EXTI_InitStruct;

        /*开启LED相关的GPIO外设时钟*/
        RCC_AHB1PeriphClockCmd ( RCC_AHB1Periph_GPIOA, ENABLE);

        //RCC_APB2PeriphClockCmd ( RCC_APB2Periph_SYSCFG, ENABLE);

        EXTI_NVIC_Config();

        /*选择要控制的GPIO引脚*/
    GPIO_InitStructure.GPIO_Pin =  KEY1_PIN;
        /*设置引脚模式为输入模式*/
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
    /*设置引脚为浮空模式*/
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
        /*调用库函数，使用上面配置的GPIO_InitStructure初始化GPIO*/
        GPIO_Init(KEY1_GPIO_PORT, &GPIO_InitStructure);

        SYSCFG_EXTILineConfig(KEY1_INT_EXTI_PORTSOURCE, KEY1_INT_EXTI_PINSOURCE);

        EXTI_InitStruct.EXTI_Line = KEY1_INT_EXTI_LINE;
        EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;
     EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Rising;
        EXTI_InitStruct.EXTI_LineCmd = ENABLE;
        EXTI_Init(&EXTI_InitStruct);
}
```

两个函数均为从网上拷贝后稍微修改得到，stm32f4 系列的微处理器需要初始化中断向量表和中断引脚。

中断处理函数如下：

```
152     * @}
153     */
154
155   void KEY1_INT_EXTI_IRQHANDLER(void)
156   {
157       if( ( EXTI_GetITStatus(KEY1_INT_EXTI_LINE) ) != RESET )
158           {
159               paused=!paused;
160           }
161           EXTI_ClearITPendingBit(KEY1_INT_EXTI_LINE);
162   }
163
164   /************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

使用了两个库函数来实现主要逻辑，这里修改的是 stm32f4xx_it.c 文件（之前是直接在 main.c 里加的，查了资料发现中断处理函数应该加在 stm32f4xx_it.c，并且要加上相应的头文件）

主函数如下：

```
102     while(1)
103     {
104         if(!paused){
105             LED1_ON;
106             LED2_OFF;
107             DelayNS(1500);
108             if(paused)continue;
109             LED1_OFF;
110             LED2_ON;
111             DelayNS(1500);
112         }
113         else{
114         }
115     }
```

这里的逻辑就是在灯的间隙判断 paused 变量的值，一但 paused 被置位，就会使整个循环空转，直到下次按下按键，才会恢复。（所有代码见附件）

写好代码后编译，得到如下报错：

```
..\..\Output\按键.axf: Error: L6200E: Symbol paused multiply defined (by stm32f4xx_it.o and main.o).
Not enough information to list image symbols.
Not enough information to list load addresses in the image map.
Finished: 2 information, 0 warning and 1 error messages.
"..\..\Output\按键.axf" - 1 Error(s), 0 Warning(s).
Target not created.
```

该报错的原因根据网上的资料是有源文件未被包含进项目，但是我查了几次，所有新加的源文件和头文件都应该纳入了本项目，但是报错仍旧，遂放弃。

不用中断（在本实验的基础上加东西）：此处的思路是，在每个灯亮的过程中不断检查按键的输入，如果按下则反转 paused 的值，并在灯灭前检查 paused，paused 为 1 则跳入空循环，空循环中不断检查按键是否按下，以随时返回灯亮的状态。

以下是主循环：

```
    while(1)
    {
        if(!paused){

        // red light on
        LED1_ON;
        LED2_OFF;

        int i;
        for(i=0;i<=2000;i++){
        if( Key_Scan(KEY1_GPIO_PORT,KEY1_PIN) == KEY_ON  ) // check if pressed
        {
            paused = !paused;
        }
        DelayNS(1);
        }

        // paused, get out of the loop
        if(paused)continue;

        // green light on
        LED1_OFF;
        LED2_ON;

        for(i=0;i<=2000;i++){
        if( Key_Scan(KEY1_GPIO_PORT,KEY1_PIN) == KEY_ON  ) // check if pressed
        {
            paused = !paused;
        }
        DelayNS(1);
        }
    }
    else{
        if( Key_Scan(KEY1_GPIO_PORT,KEY1_PIN) == KEY_ON  )
        {
            paused = !paused;
        }

        }

    }
```

与中断的区别就是，中断的代码要简洁很多，也更为灵敏，而我这个代码实际上就是把中断处理函数里面的内容写了三次（红灯亮时，绿灯亮时，循环空转时），其他的代码和原实验里面的内容相同，此处不再赘述（所有代码和演示见附件）

五、实验总结

本次实验我收获颇多，通过加入按键这一个功能，使我使用开发板的能力大大增强，尤其是对多文件编写项目的能力和端口映射的重要性。此外，通过对自己想法的探索，发现自己对嵌入式编程的认识仍然不够深入，还需要不断地学习，同时也很好的激发了我对嵌入式开发探索的兴趣，后面有机会找一些有意思的嵌入式项目做一些实际的开发。