嵌入式系统导论实验报告

姓名	学号	班级	电话	邮箱
曹广杰	15352015	1501	13727022190	1553118845@qq.com

实验要求

输入：PF0， PF4

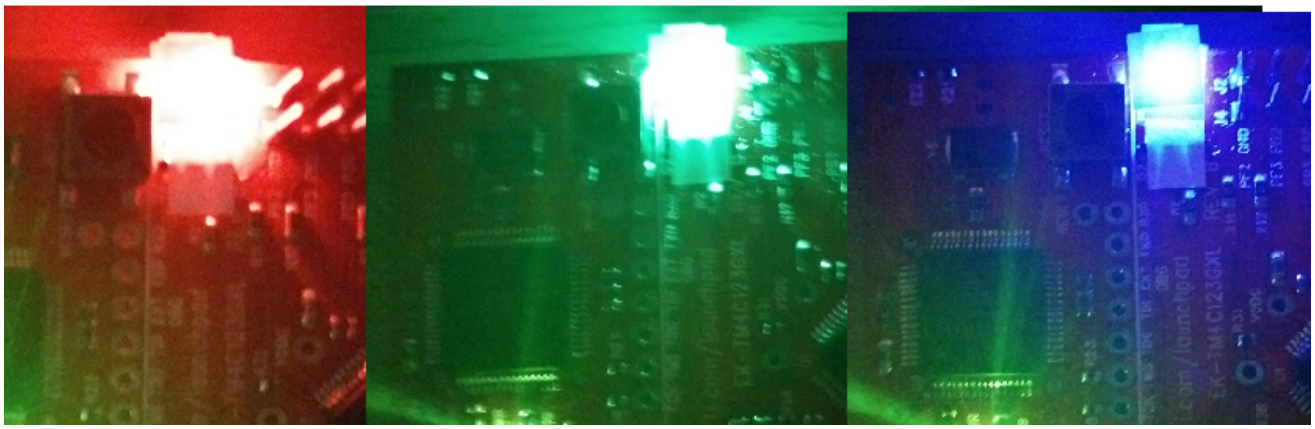
输出：LED灯

输入一个使用外部触发中断，一个使用查询方式；多种灯的变换模式、变化速度；

设计思路与实现功能

1. sw1用于中断，sw2用于查询功能；
2. 初始化之后，需要使用sw2的查询功能，允许按钮的中断操作；
3. 按钮触发的中断优先级高于`Timer0A`中断的优先级；
4. 使用LED的呼吸灯效果显示按钮触发的情况；

实验结果



实验基础

1. 确定输出LED灯的接口信息：

```
#define PF1          (*((volatile uint32_t *)0x40025008))
#define PF2          (*((volatile uint32_t *)0x40025010))
#define PF3          (*((volatile uint32_t *)0x40025020))
```

此处F端口的基地址为`0x40025000`，因此，在实验板的三个输出端口按顺序表示为以上的16进制格式。

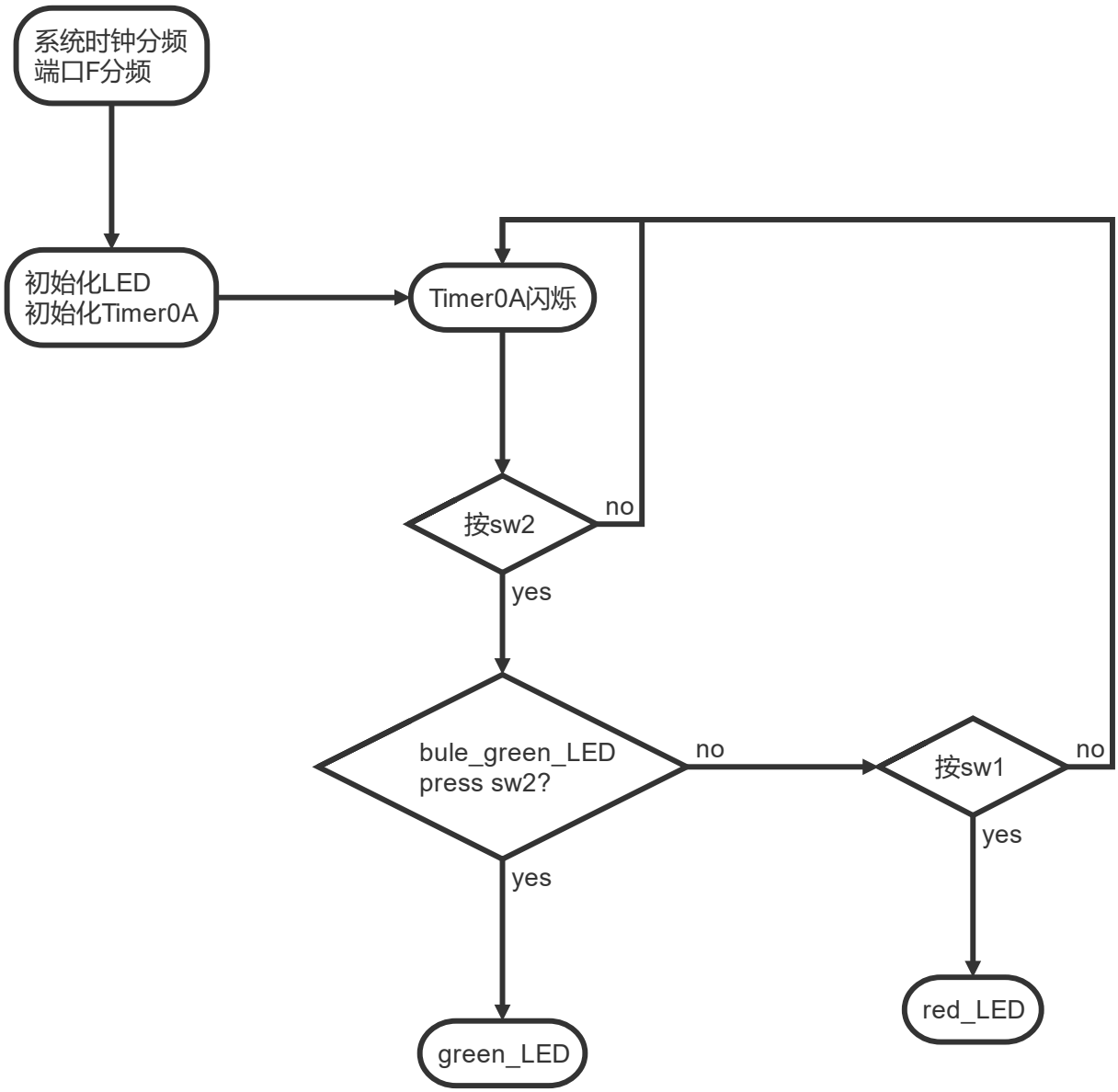
2. 确定三种颜色的表示方法：

```
#define RED       0x02
#define BLUE      0x04
#define GREEN     0x08
```

使用的时候将颜色的值赋值给总线`LEDS`即可。

实验流程

笔者设计的实验流程如下：



设备初始化

1. 对系统时钟分频：

   调用函数`PLL_Init()`对系统时钟进行分频操作，配置系统时钟并从PLL中获得系统时钟的频率。调用`SysTick_Init()`对系统时钟初始化。

2. 初始化F端口

   初始化F端口有如下几个步骤：

   1. 激活F端口的时钟，并确认时钟已经开始正常计数；
   2. 设置其为输出端口；
   3. 关闭该端口的模拟功能与换挡功能，并开启数字功能；

   综上可以得到初始化F端口的函数如下：

```
// 1) activate clock for Port F ,  allow time for clock to start
  SYSCTL_RCGCGPIO_R |= 0x20;
  while((SYSCTL_PRGPIO_R&0x20)==0){};
// make PF3-1 outputs
  GPIO_PORTF_DIR_R |= 0xE;
// disable analog function and alt func on PF3-1;
// enable digital I/O on PF3-1
  GPIO_PORTF_AMSEL_R &= ~0xE;
  GPIO_PORTF_AFSEL_R &= ~0xE;
  GPIO_PORTF_DEN_R |= 0xE;
// GPIO
  GPIO_PORTF_PCTL_R &= ~0x00FFF000;
```

值得一提的是设置`GPIO`的步骤。

`GPIO`是通用IO接口，是结构较为简单的外部设备，接口至少有两个寄存器："通用IO控制寄存器"与"通用IO数据寄存器"。

3. 设置`LED`灯的初始状态

由上文得知了三个端口`PF1`，`PF2`以及`PF3`，其实这三者可以合并为一个变量，并通过对变量的赋值实现对于灯的颜色以及亮度的把控：

```
#define PF1         (*((volatile uint32_t *)0x40025008))
#define PF2         (*((volatile uint32_t *)0x40025010))
#define PF3         (*((volatile uint32_t *)0x40025020))
#define LED         (*((volatile uint32_t *)0x40025038))
```

此时对于LED的赋值可以直接作用到三个灯上，例如：

在期望使用端口`PF1`输出的时候，只能给`PF1`赋值`0x02`，以此计算出期望的16进制数字：

```
                                PF1 = 0x02
```

其实使用LED很显然也可以计算出同样的16进制数字：

```
                                LED = 0x02
```

输出的16进制数字是完全一样的。所以可以设置初始状态：`LED = 0x00`；同样的，可以根据颜色的搭配获得多种颜色：

```
#define RED       0x02
#define BLUE      0x04
#define GREEN     0x08
// YELLOW = RED + GRREN
// PURPLE = BLUE + RED
// WHITE = RED + GREEN + BLUE
// DRAK = 0
```

Timer0A中断

实验的过程中，笔者修改代码之前首先需要理解当前的每一个部分的代码代表什么，所以需要对代码的每一处进行分析。该section分析Timer0A的中断机制。

Timer0A的中断被安置在初始化设定之后，在设备完成初始化之后，开始使用中断运行实现设计好的机制，便于观察当前所处的状态。

```
                    #define F1HZ (50000000/1)

                    Timer0A_Init(&UserTask, F1HZ);
```

在Timer0A的实现中，传入的参数是：

1. 中断执行的具体操作的函数的函数指针；
2. 运行频率；

其中函数`UserTask`定义为：

```
const long COLORWHEEL[WHEELSIZE]
    = {RED, RED+GREEN, GREEN, GREEN+BLUE,
        BLUE, BLUE+RED, RED+GREEN+BLUE, 0};

void UserTask(void){
  static int i = 0;
  LEDS = COLORWHEEL[i&(WHEELSIZE-1)];
  i = i + 1;
}
```

该函数的设计正是使得LED灯闪烁已有的多个颜色，由于该函数指针作为参数传入了中断中，故而中断会按照既定的顺序闪烁这几个颜色。

设置Port的查询方式

在完成了初始化之后，需要对按钮进行设置。因为本次实验需要通过按钮进行触发。

查询机制表示非中断的按钮检查与设置方式，不同于中断的实现是写在中断的回调函数里，查询机制的实现位置是main函数内部。main函数内部有一个while循环，该循环在运行之前，由于已经触发了`Timer0A`的中断，因而进入循环之前，`Timer0A`的中断一直在运行，也就是LED灯还一直在闪烁。此时，程序进入while循环。

```
while(1){
        // 读取按钮的状态
         In = GPIO_PORTF_DATA_R&0x11;
         if(In == 0x10){      // SW2 被按下
              LEDS = 0x00;
              /*特定的闪烁模式，表征按钮被按下*/
             // 允许端口PF1&PF4使用中断
              GPIO_PORTF_IM_R |= 0x11;
         }
  }
```

可以看到第8行代码，是允许PF1和PF4使用中断的，也就是说在此之前，即便是按下其他的按钮也不会引起中断，只有在按下port4之后，另一个按钮按下才会有反应。

设置按钮的中断触发

笔者在设计的时候将第一个按钮作为中断功能展示的按钮，但是在刚初始化结束之后还不能触发，需要先通过按下port4的按钮，对中断请求予以批准。

设置中断触发的模式如下：

```
1  void GPIOPortF_Handler(void){
2      // 读取当前的按钮状态
3      In = GPIO_PORTF_DATA_R&0x11;
4      LEDS = 0x00;
5      int i=0;
6
7      // 检测到Port0被按下
8      if(In == 0x01){
9          /*Red LED shining...*/
10     }
11     else{
12         /*Green LED shining...*/
13     }
14     GPIO_PORTF_ICR_R = 0x11;    // acknowledge flag4 FLAG 4
15 }
```

该中断函数的名字是不能随便改动的，因为这个函数的名字在文件`startup`中有对应的设置，对于整个系统来说，已经将其设置为了按钮触发中断的回调函数。在该函数中，首先读取按钮的状态，根据按钮的状态执行不同的操作。

对于代码第14行，acknowledge flag，是保证当前的中断代码运行结束之后可以回到之前的进程中，而不是始终在该中断处停留。

这样，在中断请求被批准之后，就可以执行以上的代码了。该终端内部的操作在之后会回到之前执行的进程中去，应该是`Timer0A`的中断。

创新点功能实现

1. 呼吸灯的设计

   早在之前使用`vivado`进行`FPGA`开发的时候就看到有同学做出了呼吸灯的展示效果，当时就非常惊艳于呼吸灯显示出的简单的设计与唯美的变换，所以这次实验笔者也打算做一个呼吸灯的显示。

   呼吸灯显示为灯缓缓变亮与缓缓变暗，随着亮度的变化，LED灯给人以晶莹剔透的感觉。实现的时候主要的操作是修改LED灯数字脉冲波的占空比：

   ```
   1          // 由暗变亮
   2          for(i=0;i<PWM;i+=rate){
   3                  LEDS = RED;
   4                  delay(i);
   5                  LEDS = 0x00;
   6                  delay(PWM-i);
   7          }
   8          // 由亮变暗
   9          for(i=PWM;i>0;i-=rate){
   10                 LEDS = RED;
   11                 delay(i);
   ```

```
                LEDS = 0x00;
                delay(PWM-i);
        }
```

修改占空比就可以改变灯的亮度，使得亮度缓慢地变化就可以实现呼吸灯的表现效果。

2. 中断的加锁操作

尽管实验中要求需要使用两种作用机制——中断与查询，但是笔者并不想一开始就使用两种，因而笔者为中断模式设置了开关。也就是在完成初始化但是没有做任何操作之前，中断是无法触发的。

实现方式非常简单，主要是使用`GPIO_PORTF_IM_R |= 0x11`，该语句允许中断的发生，笔者将其放置在查询操作的最后，也就是只有在执行完查询操作之后才可以使用中断操作：

```
int main(void){
  /* Initialization */
  while(1){

    In = GPIO_PORTF_DATA_R&0x11;
    if(In == 0x10){      // SW2 pressed
      /* Shining ... */
      // 允许使用中断
      GPIO_PORTF_IM_R |= 0x11;
    }

  }
}
```

此后就进入另一个模式：

1. 有两种中断：`Timer0A`与`GPIOPortF_Handler`；
2. `GPIOPortF_Handler`优先级略高于`Timer0A`；
3. 按下按钮之后不再进入查询模式，而是直接进入`GPIOPortF_Handler`执行中断回调函数

3. 修改闪烁频率的设计

使用一些常规的方法实现闪烁频率的修改，既可以在轮训中使用也可以在中断中使用：

```
if(In == 0x01){
    Timer0A_Init(&UserTask, F16HZ);  // initialize timer0A (1 Hz)
}
```

在按下按键之后，原来的闪烁灯由频率为1Hz转变为频率为16Hz，闪烁速度更快。这可以代表一种模式的切换。

总结与体会

本次实验中使用了很多综合的概念，通过一些综合性较强的实验可以对使用到的概念有更加深刻的理解。

- GPIO的概念

GPIO（General-Purpose IO ports），通用IO接口。

1. 通常GPIO结构简单；
2. 有时候会被作为输入，这样的设备只要求一位；

3. 接口至少有两个寄存器——“通用IO控制寄存器”与“通用IO数据寄存器”；

- 中断的概念

中断是一种使CPU中止正在执行的程序而转去处理特殊事件的操作；

中断源：这些引起中断的事件；

1. 可能来自外部设备的输入/输出请求；
2. 可能是计算机的一些异常事故；
3. 可能来自其它内部原因。

中断是CPU和外部设备进行交互的有效方法。这种输交互可以避免因反复查询外部设备的状态而浪费时间，提高了CPU的效率。

附录

笔者只修改了一个文件："PeriodicTimer0AInts.c"

```
// PeriodicTimer0AInts.c

#include "tm4c123gh6pm.h"
#include <stdint.h>
#include "PLL.h"
#include "Timer0A.h"


#define PF1       (*((volatile uint32_t *)0x40025008))
#define PF2       (*((volatile uint32_t *)0x40025010))
#define PF3       (*((volatile uint32_t *)0x40025020))
#define LEDS      (*((volatile uint32_t *)0x40025038))

#define RED       0x02
#define BLUE      0x04
#define GREEN     0x08
#define WHEELSIZE 8           // must be an integer multiple of 2
                              //    red, yellow,    green, light blue, blue, purple,
white,          dark
const long COLORWHEEL[WHEELSIZE] = {RED, RED+GREEN, GREEN, GREEN+BLUE, BLUE, BLUE+RED,
RED+GREEN+BLUE, 0};

void DisableInterrupts(void); // Disable interrupts
void EnableInterrupts(void);  // Enable interrupts
long StartCritical (void);    // previous I bit, disable interrupts
void EndCritical(long sr);    // restore I bit to previous value
void WaitForInterrupt(void);  // low power mode

void UserTask(void){
  static int i = 0;
  LEDS = COLORWHEEL[i&(WHEELSIZE-1)];
  i = i + 1;
}
void PortF_Init(void){

    SYSCTL_RCGCGPIO_R |= 0x00000020;             // 1) activate clock for Port F
```

```
    GPIO_PORTF_LOCK_R = 0x4C4F434B;              // 2) unlock GPIO Port F
  GPIO_PORTF_PUR_R = 0x11;                           // enable pull-up on PF0 and PF4 SW1
  GPIO_PORTF_CR_R = 0x1F;                            // allow changes to PF4-0 SW2
    // only PF0 needs to be unlocked, other bits can't be locked
    GPIO_PORTF_AMSEL_R = 0x00;                       // 3) disable analog on PF
    GPIO_PORTF_DIR_R = 0x0E;                             // 5) PF4,PF0 in, PF3-1 out
    GPIO_PORTF_AFSEL_R = 0x00;                       // 6) disable alt funct on PF7-0
    GPIO_PORTF_DEN_R = 0x1F;                             // 7) enable digital I/O on PF4-
0

//  GPIO_PORTF_IM_R |= 0x11;      // (f) arm interrupt on PF4 and PF1 *** No IME bit as
mentioned in Book ***
  NVIC_PRI7_R = (NVIC_PRI7_R&0xFF00FFFF)|0x00600000; // (g) priority 3
  NVIC_EN0_R = 0x40000000;      // (h) enable interrupt 30 in NVIC
  EnableInterrupts();           // (i) Clears the I bit
}

unsigned long In;
const int PWM =  100000;
const int rate = 1000;
void delay(int time){
    int i=0;
    for(i=0;i<time;i++);
}
void GPIOPortF_Handler(void){
    In = GPIO_PORTF_DATA_R&0x11;             // read switch status
    LEDS = 0x00;
    int i=0;
    for(i=0;i<10000000;i++);

    if(In == 0x01){                  // SW1 pressed
        for(i=0;i<PWM;i+=rate){
                LEDS = RED;
                delay(i);
                LEDS = 0x00;
                delay(PWM-i);
        }
        for(i=PWM;i>0;i-=rate){
                LEDS = RED;
                delay(i);
                LEDS = 0x00;
                delay(PWM-i);
        }
    }
    else{
        for(i=0;i<PWM;i+=rate){
                LEDS = GREEN;
                delay(i);
                LEDS = 0x00;
                delay(PWM-i);
        }
        for(i=PWM;i>0;i-=rate){

                LEDS = GREEN;
```

```
                delay(i);
                LEDS = 0x00;
                delay(PWM-i);
        }
    }

    for(i=0;i<10000000;i++){}
  GPIO_PORTF_ICR_R = 0x11;    // acknowledge flag4 FLAG 4
}

// if desired interrupt frequency is f, Timer0A_Init parameter is busfrequency/f
#define F16HZ (50000000/16)
#define F1HZ (50000000/1)
#define F20KHZ (50000000/20000)
//debug code
int main(void){
  PLL_Init();                      // bus clock at 50 MHz
  PortF_Init();
  LEDS = 0;                        // turn all LEDs off
  Timer0A_Init(&UserTask, F1HZ);  // initialize timer0A (1 Hz)
  EnableInterrupts();

  while(1){
//    WaitForInterrupt();
        In = GPIO_PORTF_DATA_R&0x11;             // read switch status
        if(In == 0x10){     // SW2 pressed
            LEDS = 0x00;
            int i=0;
            for(i=0;i<10000000;i++){}
            for(i=0;i<PWM;i+=rate){
                    LEDS = BLUE;
                    delay(i);
                    LEDS = 0x00;
                    delay(PWM-i);
            }
            for(i=PWM;i>0;i-=rate){
                    LEDS = BLUE;
                    delay(i);
                    LEDS = 0x00;
                    delay(PWM-i);
            }
            for(i=0;i<10000000;i++){}
            In = 0x00;
            GPIO_PORTF_IM_R |= 0x11;  // (f) arm interrupt on PF4 and PF1
        }else if(In == 0x01){
            Timer0A_Init(&UserTask, F16HZ);  // initialize timer0A (1 Hz)
        }
  }
}
```