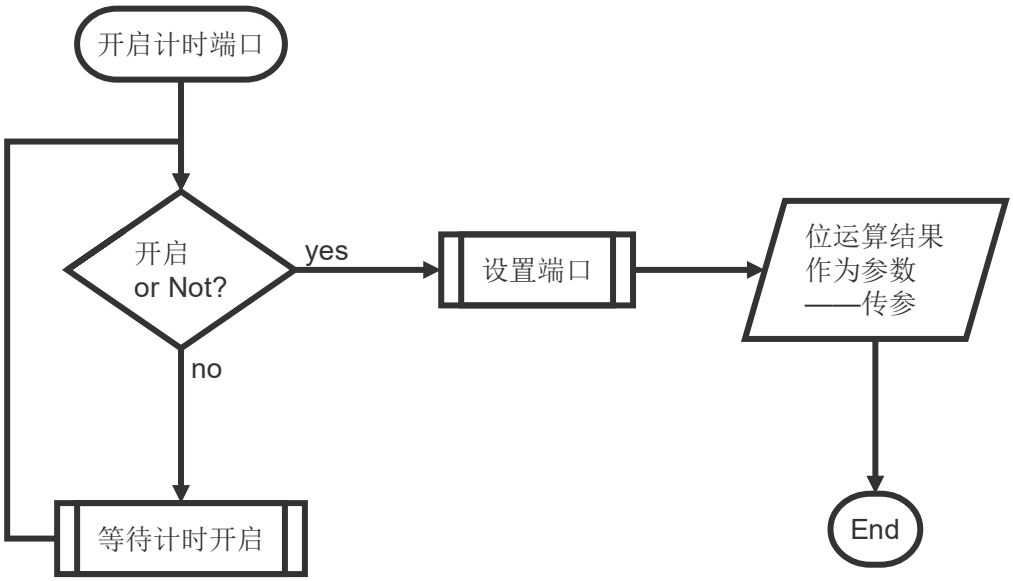
嵌入式系统导论实验报告

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

第13周

lab1

本次实验代码的主要行为是根据输入的按钮，对输出的LED灯进行操作。根据输入端口操作的不同，输出也会对应地不同。



实验结果展示

未修改之前灯的状态如下：

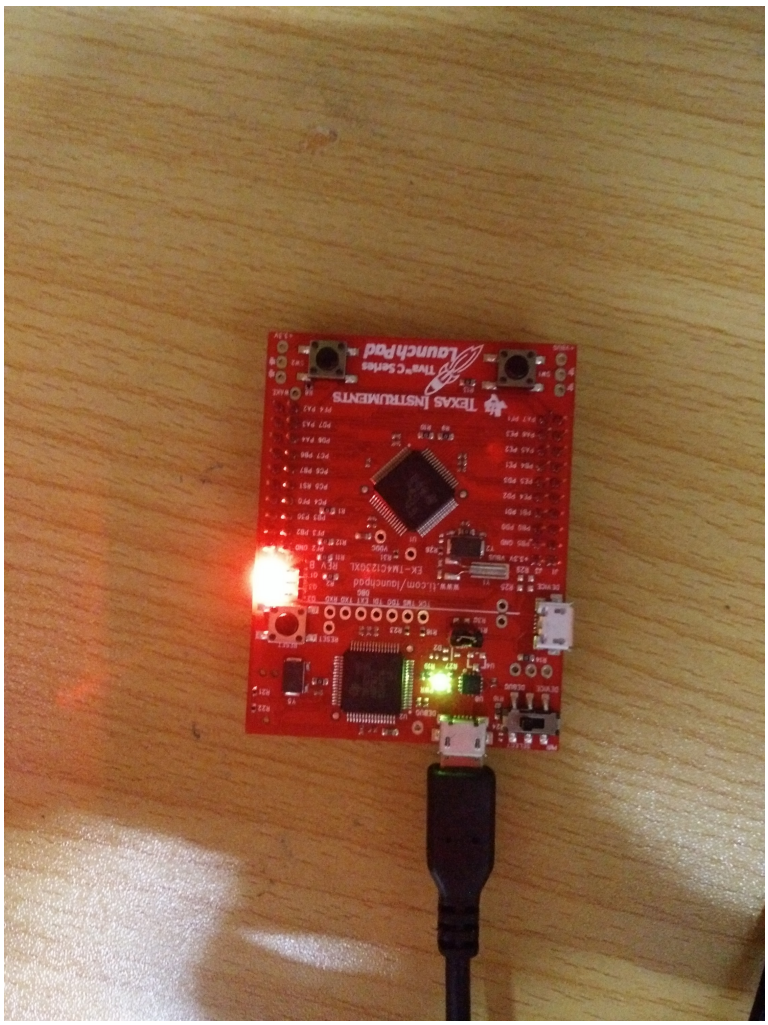
由于灯与对应的16进制关系如下：

$$LED\_RED \rightarrow 0x02$$
$$LED\_BLUE \rightarrow 0x04$$
$$LED\_GREEN \rightarrow 0x08$$

原来代码中的右移2位表示的是从`0x10`移动到`0x04`，此时灯显示为蓝色；现在笔者希望将其修改为红色，则需要在输出端口设置的时候，将右移两位转换为右移3位：

```
while(1){
    In = GPIO_PORTF_DATA_R&0x10;   // read PF4 into Sw1
    Out = In>>3;                    // shift into position PF2
    GPIO_PORTF_DATA_R = Out;        // output
}
```

此时的灯盏颜色为红色：



代码分析

等待计时端口开启

```
1	                while((SYSCTL_PRGPIO_R&0x20)==0){};
```

此处的使用方法是针对`SYSCTL_PRGPIO_R`进行监听，该变量的端口PF5决定了当前计时器是否处于开启状态——如果该端口处于高电平，则计时器处于开启状态；否则，则处于关闭状态。

于是在程序开始正常运行之前，需要确定计时器已经开启，这里`0x20`表示`00100000`

$$0x20 \rightarrow 0010|0000$$

和表示计时器开启的参数`SYSCTL_PRGPIO_R`进行与运算而不为零，只有此时才可以跳出循环，只有此时才满足计时器开启的状态要求。

设置端口

对F端口的信息进行权限设置。需要先将设置的范围设定在F区域内，再针对F端口的某一个端口进行设置。

限定修改端口为F的操作：

```
1	SYSCTL_RCGC2_R = SYSCTL_RCGC2_GPIOF;
```

这里的左值与右值都是来自头文件的参量。

1. 允许GPIO端口F端口的修改：

```
GPIO_PORTF_CR_R = 0x1F;
```

这里的`GPIO_PORTF_CR_R`是来自头文件的参量，将`0x1F`进行赋值操作是确定将要修改的变量。参数`0x1F`的意义如下：

$$0x1F \rightarrow 0001|1111$$

这个参数很显然是表示`0-4`端口，所以上述代码的意义就是将“允许端口信息修改的权限”作用到`0-4`端口上。

2. 设置输出端口：

输入端口已经设置，接下来设置输出端口。我们希望输出端口可以是`0-3`端口：

```
GPIO_PORTF_DIR_R = 0x0E;
```

所以使用`0x0E`作为参数输入。因为

$$0x0E \rightarrow 1110$$

`1`值对应高电平，从右自左计数显示`1-3`位置都是高电平，所以至此就已经将端口`0-3`作为设置参数传入了。

3. 设置上拉端口的信息

设置上拉端口的信息是0和4，则需要使用16进制位符号表示上拉端口的信息：

`GPIO_PORTF_PUR_R = 0x11;`中右值参数对应着

$$0x11 \rightarrow 0001|0001$$

其中的`1`位表示上拉时我们希望有所动作的端口位置。在输入输出端口设置结束之后，由于目前为之我们修改的还是模拟信号的端口操作，而在单片机内部计算的实际操作使用的都是数字信号，因此，需要将单片机中的数字端口激活。与之前的操作参数一样，传入的参数也是`0x1F`对应着输入端口的最低5位位置信息。开启之后，为了避免噪声的干扰，还需要对数字端口进行清零操作：

```
GPIO_PORTF_DATA_R = 0;
```

接下来设置数字端口的信息——在输出端口中设置输出的LED灯的信息。

```
GPIO_PORTF_DATA_R = GPIO_PORTF_DATA_R
                  | LED_RED
                  | LED_BLUE
                  | LED_GREEN;
```

综合红色、蓝色与绿色的信息作为输出。

至此，端口设置结束。

为头文件中的过程代码传入参数

激活输入端口为PF4：`In = GPIO_PORTF_DATA_R&;0x10;`

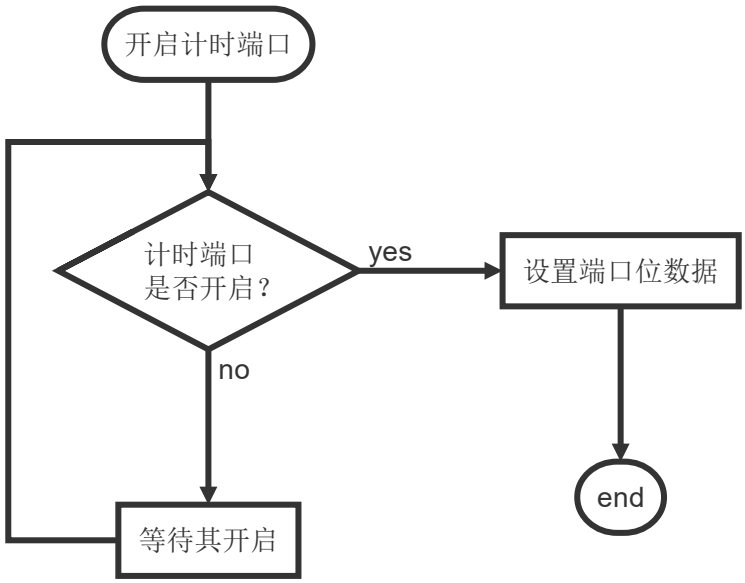
设置输出端口为PF2：`Out = In>;>;2;`

将位运算得到的值返回给算法：`GPIO_PORTF_DATA_R = Out;`

在这个过程中，`GPIO_PORTF_DATA_R`的不同位置表示不同的输出。移位操作实现了输入与输出的转换。

lab2代码分析

第二个实验根据按钮的操作对应修改LED的程序。



实验结果展示

尚未修改的原始代码运行结果如下：

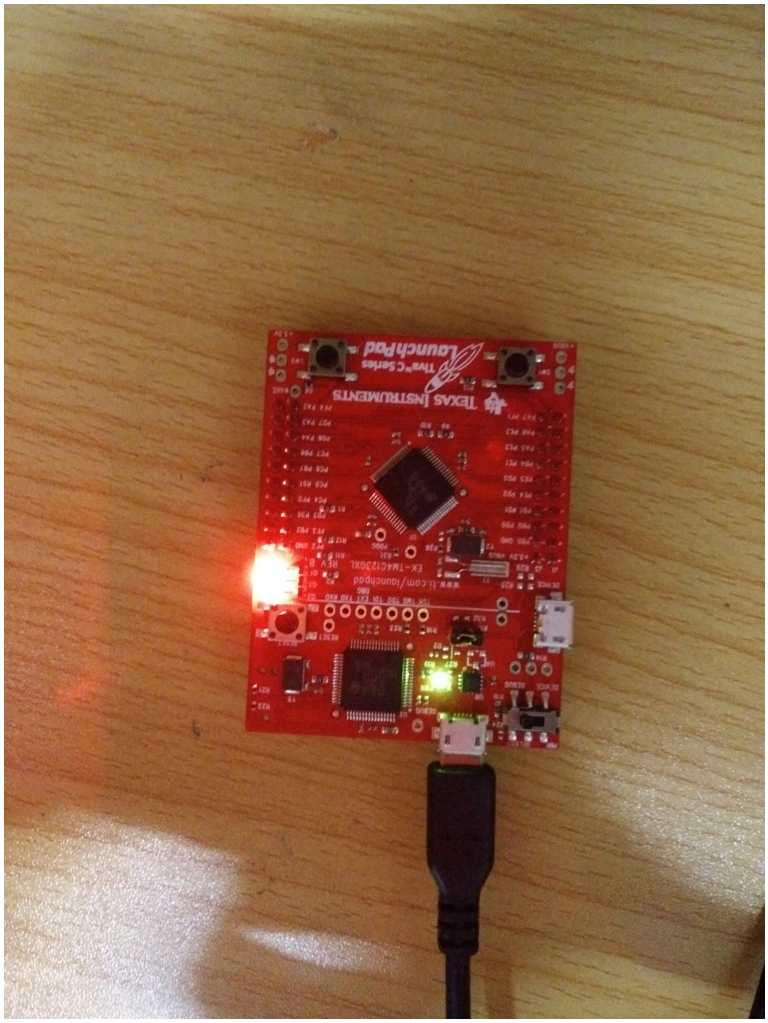
修改输出端口为PF1使LED灯显示为红色。

首先观察输出端口在宏定义中的实现：

```
1    #define PF2   (*((volatile uint32_t *)0x40025010))
```

其中，`0x40025010`表示位运算的结果，基地址为`0x40025000`，此处设置输出端口为PF2，即`0x04`，将基地址与之相加，可以获得以上的位地址信息。现在将其修改为PF1输出，于是需要将`0x04`修改为`0x02`——对应的地址为`0x40025008`；

并将其下对应的`0x04`端口修改为`0x02`即可。



修改输入端口，将对应PF4的`0x10`修改为PF1对应的`0x01`即可将输入位的作用对调。

代码分析

初始化操作

激活计时器并设置端口信息

激活计时器，并确定计时器已经激活：

```
    SYSCTL_RCGCGPIO_R |= 0x20;
    while((SYSCTL_PRGPIO_R&0x20)==0){};
```

设置端口的操作

由于在本次实验中使用数字信号功能就需要关闭模拟信号功能以及换挡功能，否则的会导致噪声的产生。

此后使用循环操作：

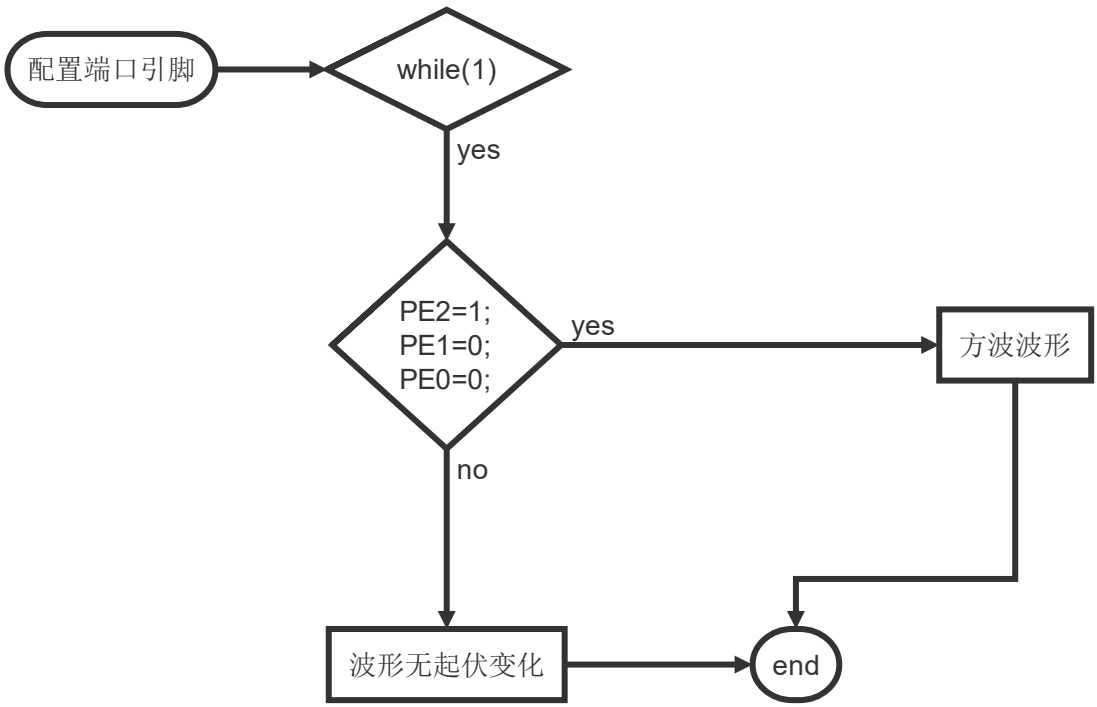
```
while(1){
    SSR_On();// Make PF2 high
    // wait for button press
    // wait for button release
    SSR_Off();// Make PF2 low
    // wait for button press
    // wait for button release
}
```

这样就可以通过按钮控制PF2的信号变化了。

lab3代码分析

通过对PortE输入端口的设置影响输出信息，并观察分析波形。

实验代码流程图



代码分析

在配置完成信息之后，就会进入循环，如果不满足循环的if语句，输出波形就会始终为0；

首先确定按键状态与传感器状态都正常：

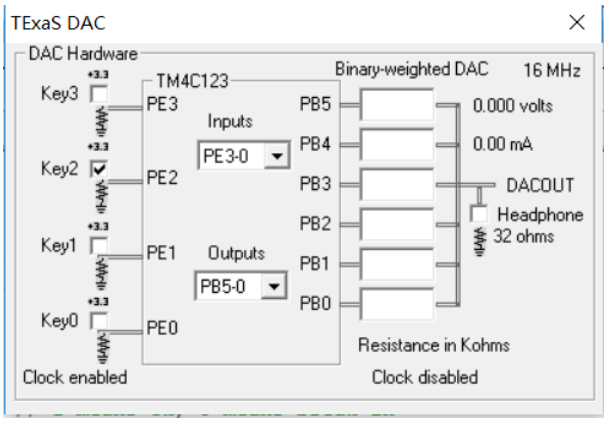
```
        arm = GPIO_PORTE_DATA_R&0x04;
        sensor = GPIO_PORTE_DATA_R&0x03;
```

以上代码是对特定比特位的检查，如果两个条件都符合，则继续下面的步骤，否则关闭LED灯。

如果符合条件，设置计时器作为触发，并将频率设置为5Hz：

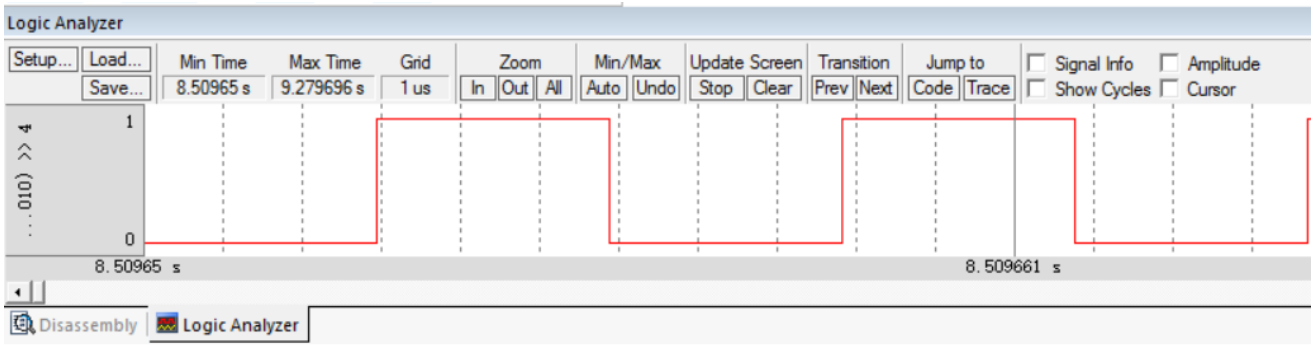
```
            GPIO_PORTE_DATA_R ^= 0x10;
            delay=100;
```

而满足if语句的方法为：



其中的设置为将第4位的信息设置为1，同时前两位置为0，以满足if中的条件。该界面在**Peripharals**中的**TExasDAC**设置。

在满足if语句之后，执行的操作是对输出位不断进行异或操作，由此形成了矩形方波。获得波形如下：



可以看到周期为100ms，即频率为5Hz。