嵌入式系统理论作业

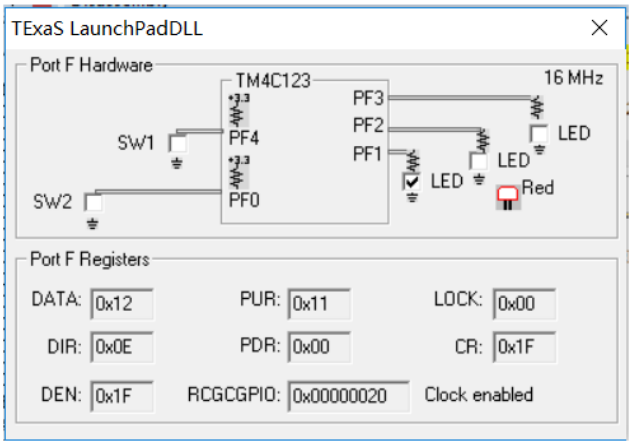
曹广杰

15352015 数据科学与计算机

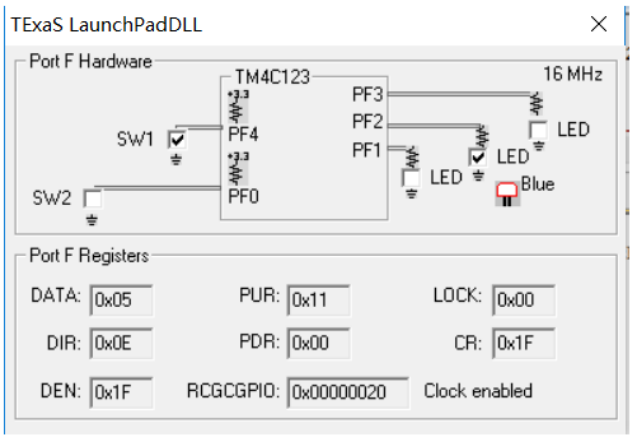
授课教师：郭雪梅

实验板的实验

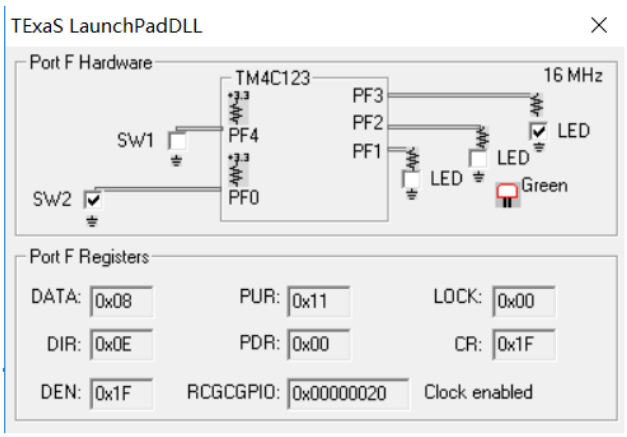
调试未修改之前的模拟LED灯的显示情况：



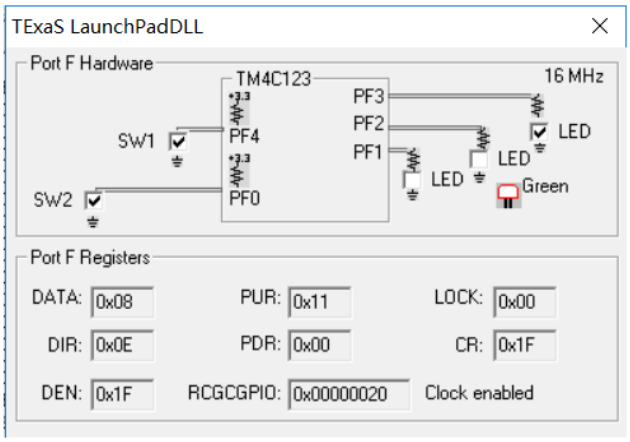
1.1 两个switch都不触发



1.2 仅触发第一个switch



1.3 仅触发第二个switch



1.4 同时触发两个switch

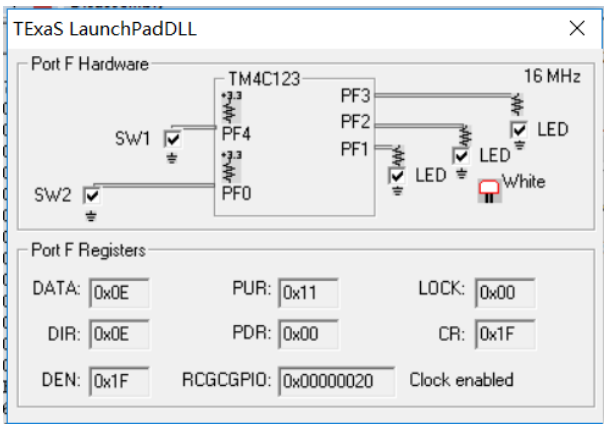
修改实验板实验的输出信息

修改程序改变按键对应的驱动灯颜色，做个前后对比，并作调试分析；

在修改之前需要找到管理输出的代码模块，如下：

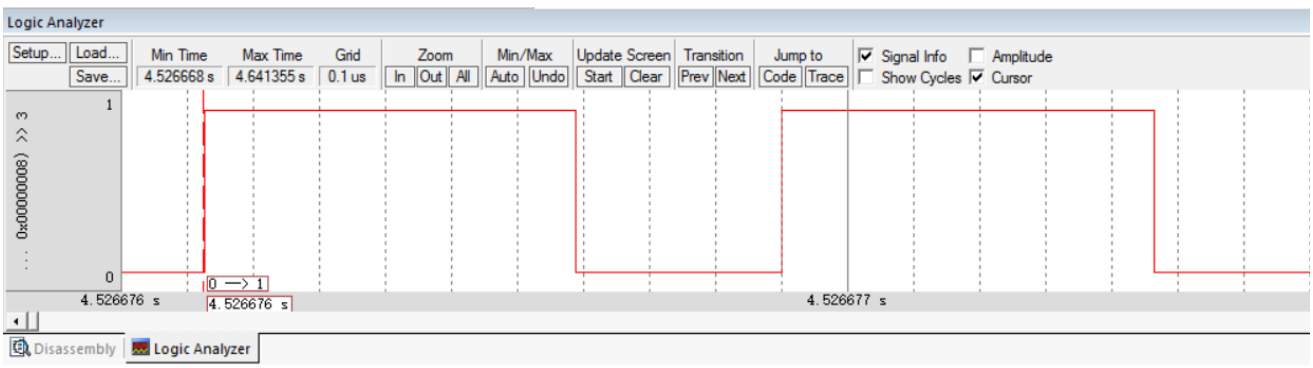
```
1  sw1pressed
2      MOV R0, #BLUE                ; R0 = BLUE (blue LED on)
3      BL  PortF_Output             ; turn the blue LED on
4      B   loop
5  sw2pressed
6      MOV R0, #RED                 ; R0 = RED (red LED on)
7      BL  PortF_Output             ; turn the red LED on
8      B   loop
9  bothpressed
10     MOV R0, #GREEN               ; R0 = GREEN (green LED on)
11     BL  PortF_Output             ; turn the green LED on
12     B   loop
13 nopressed
14     MOV R0, #GREEN               ; R0 = GREEN (green LED on)
15     BL  PortF_Output             ; turn all of the LEDs off
16     B   loop
```

可以看到，输出信息的输出随switch的触发状态改变，如果笔者希望修改输出状态，则修改当前的不同的输出状态即可。比如，笔者现在希望在按下两个switch的时候，所有的灯都能同时点亮。那么笔者只需要将触发的情况进行修改即可：将第10行的`#GREEN`修改为`#(RED+GREEN+BLUE)`即可实现三者的同时点亮：



同理可以修改其他的输出信息，满足我们输出的所有要求。

使用逻辑分析仪观察 portd.3 输出



修改程序，使用 portD.2 输出

首先需要找到几个关于PD3端口设置的代码语句，在端口设置的代码块中有：

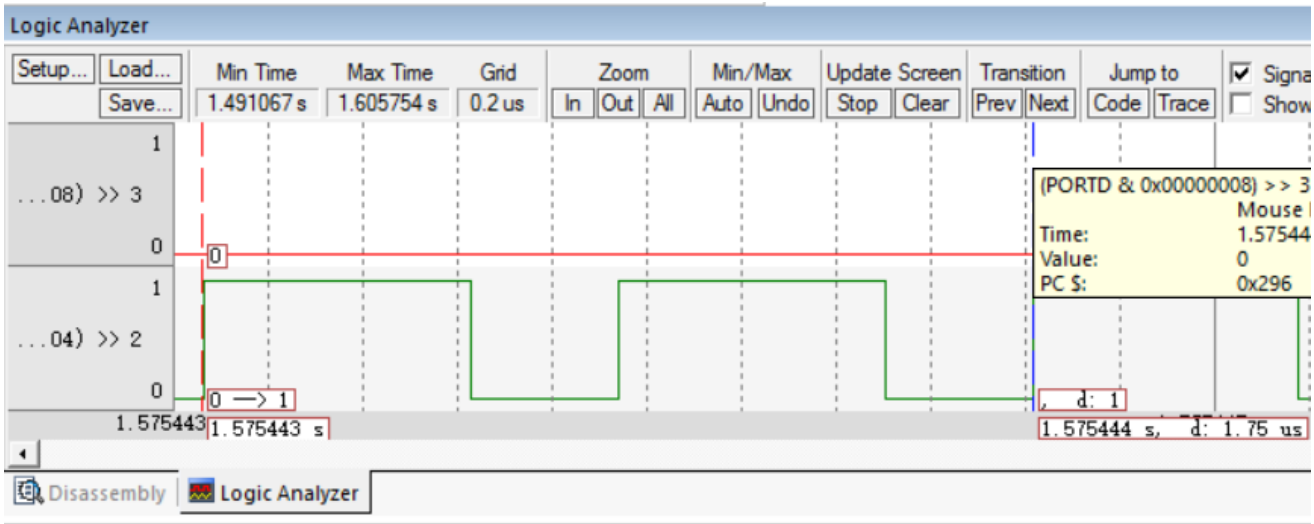
```
    ORR R0, R0, #0x08
```

这里表示将输出端口设置为PD3——使用16进制移位表示端口3；为此需要将该部分设置为0x04，以便将其转为PD2，因为二进制对于2的移位正好是4；

相对应的对于端口的配置，也应该延承下来。比如对于数字模式的启用与对于alt模式的禁用，依然是使用移位的方式表示：

```
    BIC R0, R0, #0x09
...
    ORR R0, R0, #0x05
```

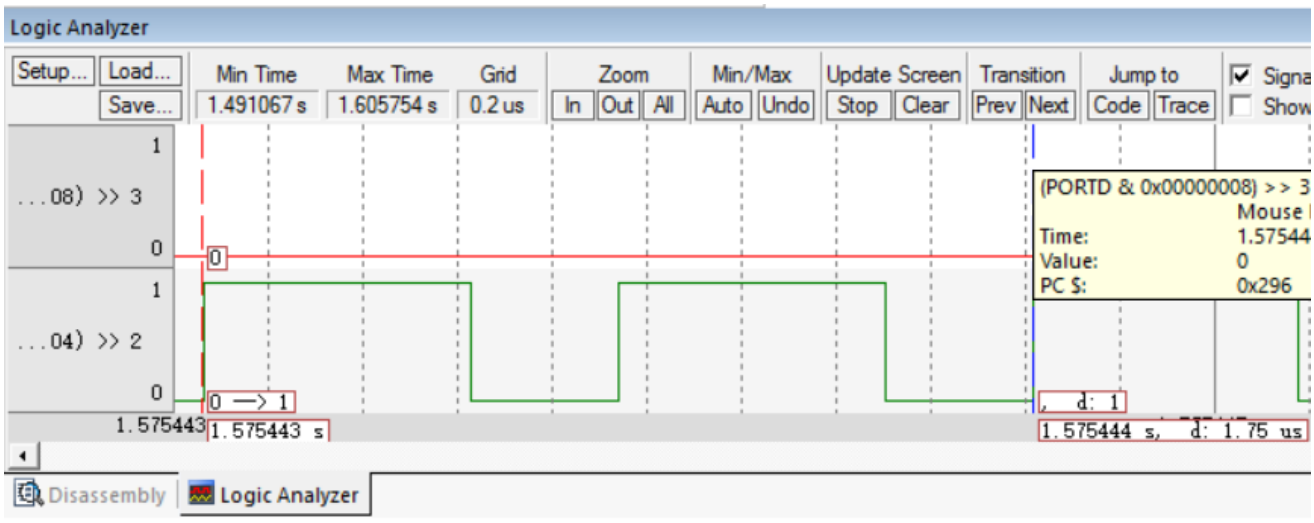
确定了输入端口与输出端口的或运算结果，将其转换为端口2，则对应的应该使用0x05表示两个端口的相对关系。



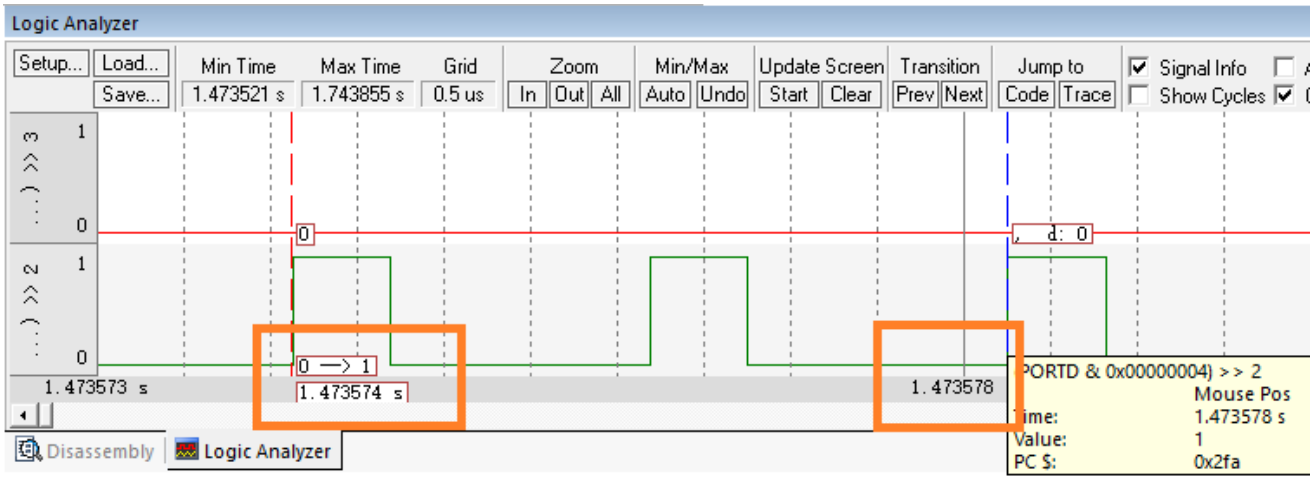
可以看到，修改之后的图形中，原来的PD3端口波形已经稳定，不再有方波出现。而PD2端口则表示为方波，说明对于输出端口的设置是成功的。

改变输出状态变化时间

在修改之前对时间周期进行记录。由上图可知，起始时间是1.575443s，终止时间由笔者的记录显示为：



可见，当前的两个周期用时0.000001s。接下来，笔者通过修改nop气泡的数量尝试对周期信息进行修改。笔者将气泡的数量添加到20个，此时对波形图进行截图：



可以看到，并不影响端口的波形信息，但是会影响波形的变化周期，因为笔者只在一个操作后添加了该变化，故周期的高低电平用时不均匀，但是依然可以看到非常明显的时间延长信息。

接下来对高低电平分别实现相同的延时操作，检查输出的信息，判断脉宽的变化规律。

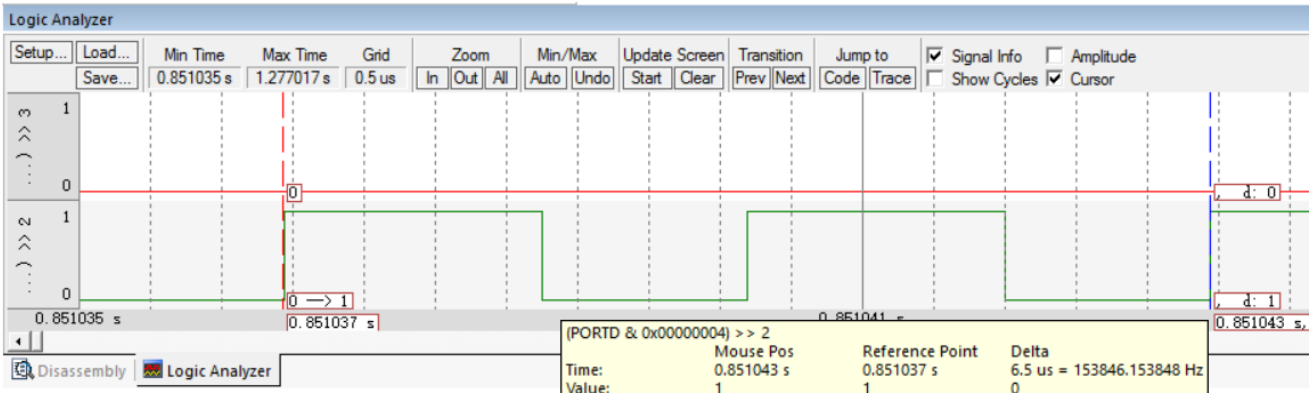
回顾一下原有的代码信息：

```
1  loop
2      LDR R1,[R0]
3      AND R1,#0x01          ; Isolate PD0
4      EOR R1,#0x01          ; NOT state of PD0 read into R1
5      STR R1,[R0]
6      nop
7      nop
8      LSL R1,#2             ; SHIFT left negated state of PD0 read into R1
9      STR R1,[R0]           ; Write to PortD DATA register to update LED on PD3
10     B loop                          ; unconditional branch to 'loop'
```

可以看到原有的代码信息中只有在第一次将数据写入R0对应的寄存器中才有对应的延时操作，此时笔者将延时操作同时添加到第6行和第9行之下，获得如下代码。

```
1  loop
2      LDR R1,[R0]
3      AND R1,#0x01          ; Isolate PD0
4      EOR R1,#0x01          ; NOT state of PD0 read into R1
5      STR R1,[R0]
6      ;20个Nop实现延时
7      LSL R1,#2             ; SHIFT left negated state of PD0 read into R1
8      STR R1,[R0]           ; Write to PortD DATA register to update LED on PD3
9      ;20个Nop实现延时
10     B loop                          ; unconditional branch to 'loop'
```

获得输出图像如下：



可以看到输出的波形中两个周期用时：

$$\begin{aligned} t_{period} &= t_{end} - t_{begin} \\ &= 0.851043s - 0.851037s \\ &= 0.000006s \end{aligned}$$

而之前的输出中，两个周期的用时是：

$$\begin{aligned} t_{period} &= t_{end} - t_{begin} \\ &= 1.575444s - 1.575443s \\ &= 0.000001s \end{aligned}$$

总用时较之之前的运作时间延长为之前的6倍。

分析结果：

这是在我们将气泡数量增加20倍之后得到的数据，仅仅延长为6倍，这说明在两个周期中，气泡仅仅是占据了很少的部分，绝大数据还是在用于计算以及设置端口的输出。

汇编程序的分析

本次实验的汇编代码主要分为3个部分，初始化部分(GPIO_Init)，开始部分(Start)，以及循环部分(loop)，开始部分包含初始化部分，从开始部分的代码到初始化部分的代码需要使用跳转机制。

初始化代码主要用于实现接口的配置；

初始化部分

初始化部分中首先是对两个寄存器储存值的修改：

```
1     LDR R1, =SYSCTL_RCGCGPIO_R
2     LDR R0, [R1]                  ; R0 = [R1]
3     ORR R0, R0, #0x08             ; R0 = R0|0x08
4     STR R0, [R1]                  ; [R1] = R0
```

此处将上文中的运算结果添加到当前的R1与R0寄存器中，将R0的值与3进行或运算，再将R0对应的数值赋值回R1寄存器。

之后是一系列的气泡，用于等待当前的运算结束。

```
    LDR R1, =GPIO_PORTD_DIR_R       ; R1 = &GPIO_PORTD_DIR_R
    LDR R0, [R1]                    ; R0 = [R1]
    ORR R0, R0, #0x08               ; R0 = R0|0x08 (make PD3 output)
    BIC R0, R0, #0x01               ; R0 = R0 & NOT(0x01) (make PD0 input)
    STR R0, [R1]                    ; [R1] = R0
```

该部分语句用于

1. 对寄存器r1与r0进行赋值操作，赋值为0x40007400；
2. 设置了input接口为PD0，output的接口为PD3；
3. BCI是位清除指令，这里是将最后一位清零，此时表示输出端口的数据转变为PD0，同理，输入端口的数据就是PD3（0x08表示移位运算后的结果）
4. 将R0寄存器中的内容赋值给R1寄存器中数值对应的内存地址中

为端口函数确定规律

```
    LDR R1, =GPIO_PORTD_AFSEL_R     ; R1 = &GPIO_PORTD_AFSEL_R
    LDR R0, [R1]                    ; R0 = [R1]
    BIC R0, R0, #0x05               ; R0 = R0&~0x09 (disable alt funct on PD3,PD0)
    STR R0, [R1]                    ; [R1] = R0
```

这段代码是：

1. 将[R1]中数值在内存中对应的值赋值到R0中；
2. 通过对最低5位比特位的清零，取消换挡操作
3. 并将处理之后的值，返回给R1寄存器中的数值所对应的内存数据中

激活数字接口

```
    LDR R1, =GPIO_PORTD_DEN_R       ; R1 = &GPIO_PORTD_DEN_R
    LDR R0, [R1]                    ; R0 = [R1]
    ORR R0, R0, #0x05               ; R0 = R0|0x09 (enable digital I/O on PD3,PD0)
    STR R0, [R1]                    ; [R1] = R0
```

这段代码用于继续设置端口信息：

1. 首先需要将R1端口的数值设置为端口设置的初始数值；
2. 将R1在内存中对应的值赋值给寄存器R0，以便进行后续的操作；
3. 将后5位设置为1，以实现开启数字接口的作用；
4. 完成修改之后，将值赋值回R1对应的内存数据中；

至此，对于端口信息的配置就已经完成了。接下来就是设置随着PD0端口的输入数据而变化的输出数据了。

设置输出数据

根据从PD0端口输入的值对PD0端口的值取异或操作，以实现对于方波形图的实现。

```
    LDR R1,[R0]
    AND R1,#0x01        ; Isolate PD0
    EOR R1,#0x01        ; NOT state of PD0 read into R1
    STR R1,[R0]
```

EOR表示异或操作，该操作可以实现对于PD0输入的变量的波形构成，使得每一次输出的变量都与上一个周期输入的变量相反。此时已经对输入端口PD0的数据进行计算计算的结果显示在该寄存器的最低位，并获得了对应的方波波形，下一步就是将该数据输出到PD3接口中——将最低位的数据转移到第3位。

```
    LSL R1,#3           ; SHIFT left negated state of PD0 read into R1
    STR R1,[R0]         ; Write to PortD DATA register to update LED on PD3
```

将R1寄存器左移3位，此时最低位的数据则转移到第3位，故而方波波形也转移到第3位，以PD3作为输出端口，则可以在PD3端口获得方波波形的图形。

寄存器结构总结

寄存器是32位的一个储存序列，在实现波形或者数据的时候可以对某一个位进行操作，以获得相应的波形，将修改的位移动到输出位即可。