嵌入式系统导论实验报告

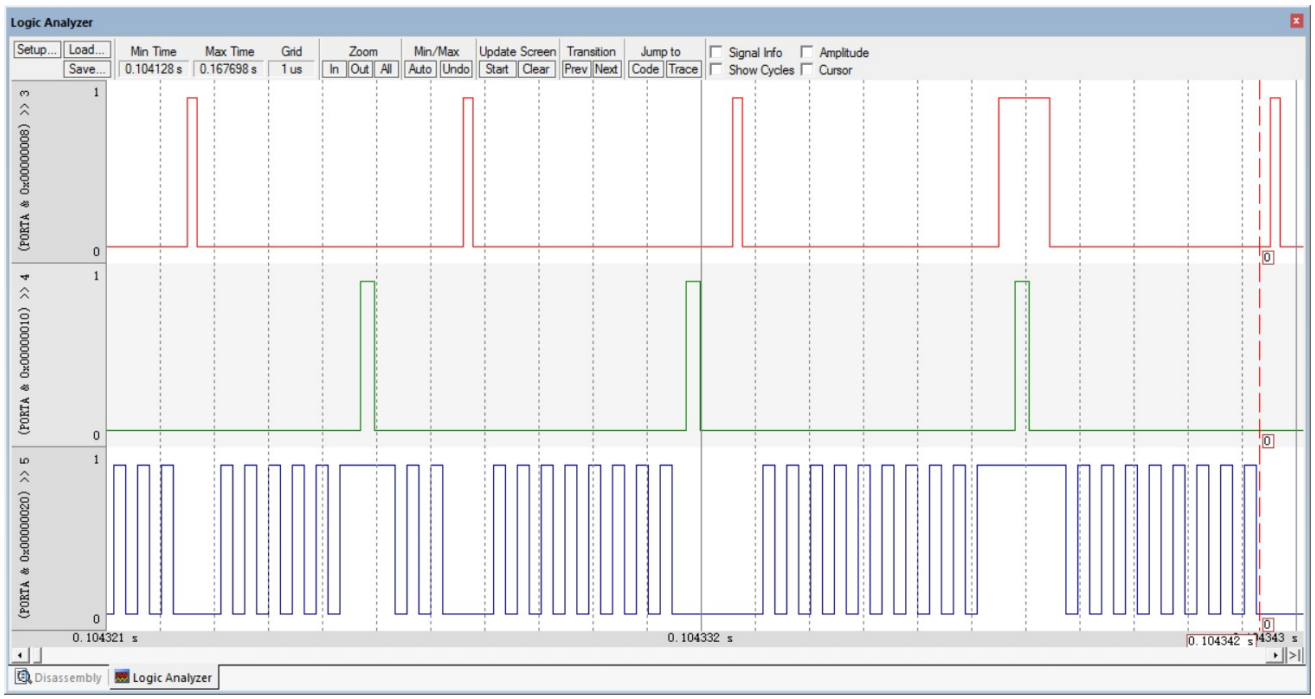
姓名	学号	班级	电话	邮箱
曹广杰	15352015	1501	13727022190	1553118845@qq.com

第15周嵌入式实验报告

实验题目：计时器中断

仿真程序

首先对当前的程序进行仿真操作，本次程序的输出端口为PORTA的3/4和5端口。检查这3个端口的输出波形如下：



可以看到：

1. 端口A3与A4均为检测的中断端口；
2. 每当中断发生的时候，A5端口会停止波动，程序转为服务中断信息；
3. 端口A3与端口A4的优先级有所不同，在二者同时触发中断的时候，系统会选择优先级较高的中断优先服务，待服务完成之后，再去服务另一个中断。

修改端口

仿真完成后把相应端口PA3,4,5换成PF1，2,3.重新对端口初始化设置。

需要修改输出的端口，就需要将所有输出端口对应的参数都做修改，代码中对输出端口执行的操作有如下几种：

1. 解锁；
2. 关闭模拟功能，开启数字功能；

3. 在中断中设置对应的输出波形；

前两个操作确保F端口可以输出正常的波形，后一个操作用于检测中断的信息。因为本次实验主要应对两种情况，即处理中断信息与非中断信息，所以至此就已经将所有的情况都考虑过了。

修改的具体实现：

1. 解锁

```
  SYSCTL_RCGCGPIO_R |= 0x20;        // 1) activate clock for Port F
  while((SYSCTL_PRGPIO_R&0x20)==0){}; // allow time for clock to start
```

对参数`SYSCTL_RCGCGPIO_R`进行赋值，可以实现解锁的效果，即激活对于F端口的操作权限，这里的16进制数字`0x20`恰好表示F端口。激活端口之后就可以修改F端口的配置了。

2. F端口的配置

```
  GPIO_PORTF_AMSEL_R &= ~0xE;    // disable analog function
  GPIO_PORTF_PCTL_R &= ~0x00FFF000; // GPIO
  GPIO_PORTF_DIR_R |= 0xE;  // make PF3-1 outputs
  GPIO_PORTF_AFSEL_R &= ~0xE;// disable alt func on PF3-1
  GPIO_PORTF_DEN_R |= 0xE;  // enable digital I/O on PF3-1
                            // configure PF3-1 as GPIO
```

使用端口显示波形，则需要关闭端口的模拟功能，开启端口的数字功能，实现去噪的期望。所以需要对原有的PortA的部分都修改为PortF，同时将端口3、4和5对应的16进制数字都修改为端口1、2和3对应的数字，得到如上代码。

3. 设置中断信息

两个中断的函数内容如出一辙，主要是实现冲激函数，以便检查输出的周期，确定分频的效果：

```
void SysTick_Handler(void){
  //PA4 = 0x10;
  PA4 = 0x4;
  Counts = Counts + 1;
  PA4 = 0;
}
```

但是原来输出的端口是PortA的4端口，此时我们需要将其修改为PortF的2端口，得到代码如下：

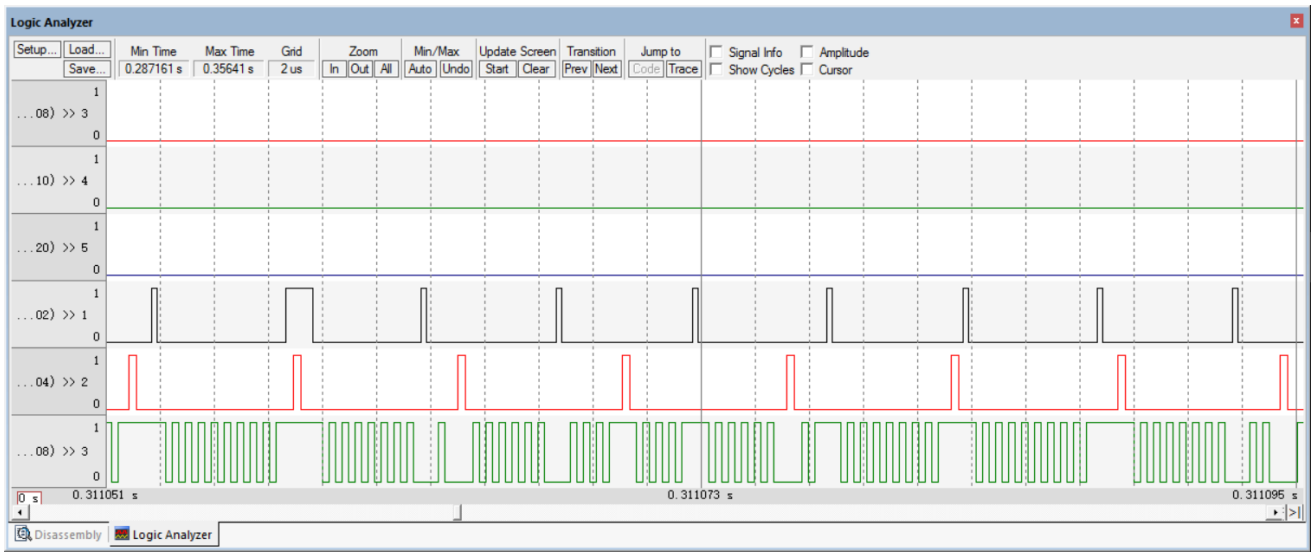
```
void SysTick_Handler(void){
  //PA4 = 0x10;
  PF2 = 0x4;
  Counts = Counts + 1;
  PF2 = 0;
}
```

PF2是尚未声明的变量，所以代码不能运行，因此需要将之前的变量声明部分进行修改替换。

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

对应的计算方式也非常简单，由于F端口的基地址为`0x40025000`，所以对应的输出端口的设置也应该在基地址的基础上进行16进制的加法运算，得到以上地址。此后需要将代码中所有的`PA3`修改为`PF1`、`PA4`修改为`PF2`、`PA5`修改为`PF3`。至此输出的信息就已经由PortA端口转移到PortF端口了。

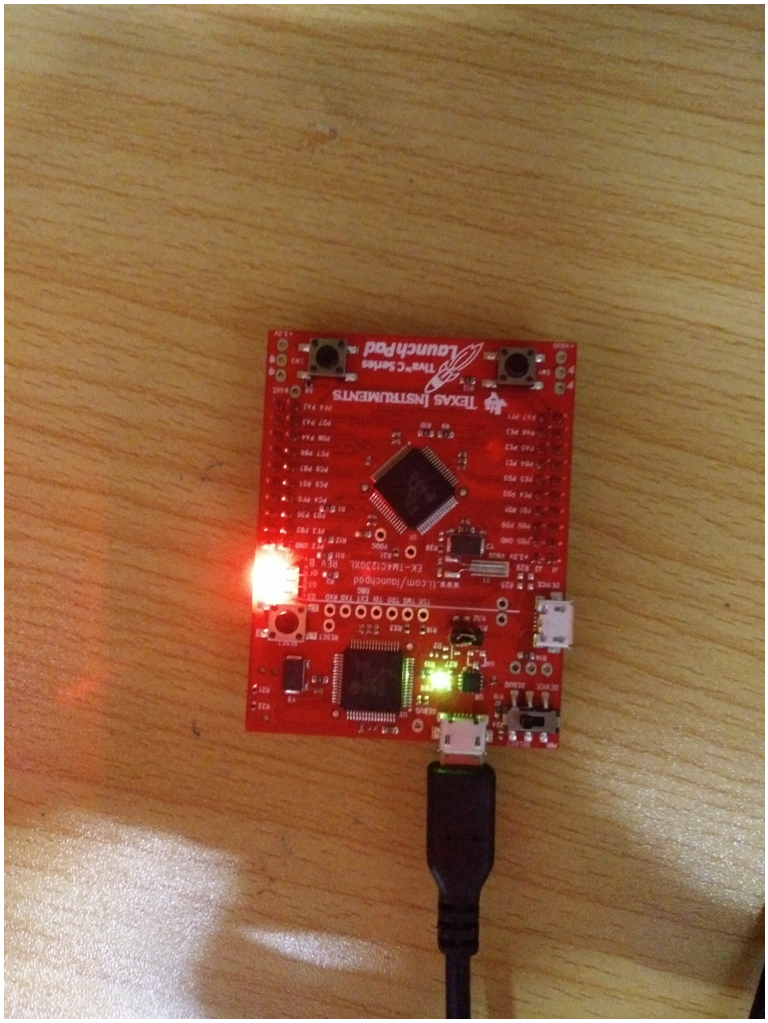
此时的输出波形如下：



修改计时器中断

- 设置每1s Time定时中断并PF2切换亮灭
- 对系统定时器中断时间重新设定为10ms,

在其中断服务程序中观察PF2与PF3的切换状态。



频率分析

在本次实验中，实验板的核心定时器频率是400MHz，代码的初始阶段对这个频率进行了分频操作。

```
#define SYSDIV 3
#define LSB 1
```

对于参数的设定将根据公式

$$400MHz/(2*SYSDIV+1+LSB)=400MHz/(2*3+1+1)=50MHz$$

将当前代码的运行环境的频率设置为了50MHz。

修改系统定时器中断时间

针对系统定时器的函数是`SysTick_Init`，传入的参数表示对当前频率的均分份数。为了将系统定时器的周期调节为10ms，需要设置其频率为：

$$\begin{aligned} f &= \frac{1}{T} \\ &= \frac{1}{10^{-2}s} \\ &= 10^2 Hz \end{aligned}$$

所以需要将$50MHz = 5 \times 10^7 Hz$均分为5×10^5份，故而传入参数应为50 0000，得到修改后的代码如下：

```
                        SysTick_Init(500000);
```

修改Time0定时器的中断时间

对Time0的设置与对系统定时器的设置一样，传入的参数也会对频率实现均分的效果。有所不同的是，在Time0对应的函数中有一个参数可以用于二级分频，最后分频的份数是二者分频份数的乘积：

```
void Timer0A_Init(unsigned short period){
  volatile uint32_t delay;
  /**/
  TIMER0_TAPR_R = 999;              // 1s
  /**/
}
```

从0到999，这里表示在传入参数`delay`的基础上继续均分1000份，综上为均分为`1000xdelay`份。

此时可以观察到PF2的输出信息如下：

可以看到此时的PF2端口输出（为了便于观察周期信息这里使用冲击函数作为PF2输出）输出周期为0.01s，恰好为我们预期的10ms设置。

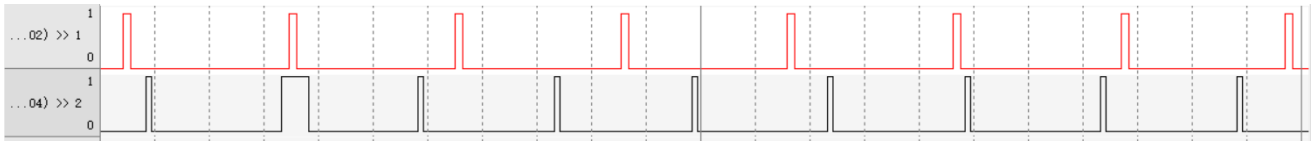
对调优先级顺序

首先需要找到对应的优先级操作信息：

```
void Timer0A_Init(unsigned short period){ volatile uint32_t delay;
  /**/
  NVIC_PRI4_R = (NVIC_PRI4_R&0x00FFFFFF)|0x60000000; // 8) priority 3
  NVIC_EN0_R = NVIC_EN0_INT19;      // 9) enable interrupt 19 in NVIC
  TIMER0_CTL_R |= 0x00000001;       // 10) enable timer0A
}
void SysTick_Init(uint32_t period){
  /**/
  NVIC_SYS_PRI3_R = (NVIC_SYS_PRI3_R&0x00FFFFFF)|0x40000000; //priority 2
  NVIC_ST_CTRL_R = 0x00000007;   // enable with core clock and interrupts
}
```

将这两个部分的优先级顺序对调，即将两行代码对调，分别放置于不同的函数之中。

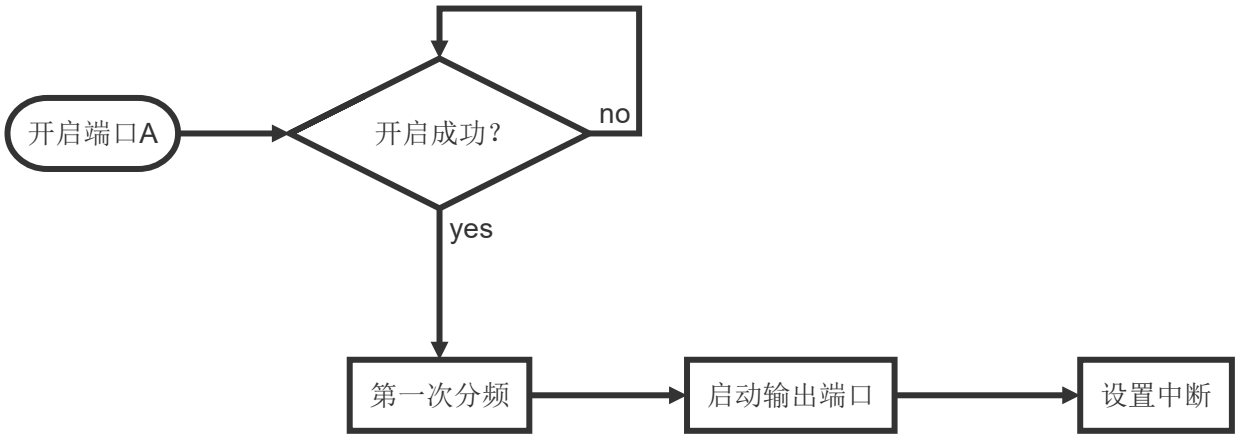
得到运行结果如下：



由于优先级对调，在二者同时触发中断的时候，优先级较高的F1端口会打断F2端口的优先级。得到实验效果如上。

程序分析

流程图如下：



第一次分频

```
#define SYSDIV 3
#define LSB 1

SYSCTL_RCC2_R += (SYSDIV<<23)|(LSB<<22);  // divide by (2*SYSDIV+1+LSB)
```

该部分中的LSB表示最低位的Bit，左移22位，则原本系统计时器400MHz的时候，应该是从最低位0位开始计数的计时器修改为从新的最低位22位开始计数，由此实现分频效果：

$$
\begin{aligned}
T_{new} &= \frac{T}{2 \times SYSDIV + 1 + LSB} \\
&= \frac{T}{2 \times 2 + 1 + 1} \\
&= 50MHz
\end{aligned}
$$

配置A端口

1. 为A端口解锁

```
SYSCTL_RCGCGPIO_R |= 0x01;   // 1) activate clock for Port A
while((SYSCTL_PRGPIO_R&0x01) == 0){};
```

变量`SYSCTL_RCGCGPIO_R`控制当前端口的锁定状态设置，其下的循环函数用于鉴定当前是否已经完成解锁操作。对于A端口，16进制数字`0x01`就是对于A端口的表示。

2. 关闭模拟功能

```
GPIO_PORTA_AMSEL_R &= ~0x38;    // disable analog function
GPIO_PORTA_PCTL_R &= ~0x00FFF000; // GPIO
GPIO_PORTA_DIR_R |= 0x38;  // make PA5-3 outputs
```

控制关闭模拟功能的参数是`GPIO_PORTA_AMSEL_R`，这里为了关闭其模拟功能需要将传入的参数取非。传入的参数正是需要关闭模拟的A系列端口的16进制表示。因为`0x38`正是`111000`的16进制表示。

3. 开启数字功能

```
GPIO_PORTA_AFSEL_R &= ~0x38;// disable alt func on PA5-3
GPIO_PORTA_DEN_R |= 0x38;  // enable digital I/O on PA5-3
```

原理同上，只是左值参数有所修改。

第二次分频

```
void Timer0A_Init(unsigned short period){ volatile uint32_t delay;
  SYSCTL_RCGCTIMER_R |= 0x01;      // 0) activate timer0
  delay = SYSCTL_RCGCTIMER_R;      // allow time to finish activating
  TIMER0_CTL_R &= ~0x00000001;     // 1) disable timer0A during setup
  TIMER0_CFG_R = 0x00000004;       // 2) configure for 16-bit timer mode
  TIMER0_TAMR_R = 0x00000002;      // 3) configure for periodic mode
  TIMER0_TAILR_R = period - 1;     // 4) reload value
  TIMER0_TAPR_R = 49;              // 5) 1us timer0A
  TIMER0_ICR_R = 0x00000001;       // 6) clear timer0A timeout flag
  TIMER0_IMR_R |= 0x00000001;      // 7) arm timeout interrupt
  NVIC_PRI4_R = (NVIC_PRI4_R&0x00FFFFFF)|0x60000000; // 8) priority 3
  NVIC_EN0_R = NVIC_EN0_INT19;     // 9) enable interrupt 19 in NVIC
  TIMER0_CTL_R |= 0x00000001;      // 10) enable timer0A
}
```

控制计时器的使用情况

1. 激活计时器；
2. 授权其中断正在运行的程序；
3. setup阶段关闭计时器，不允许其中断进程的启动；

```
  SYSCTL_RCGCTIMER_R |= 0x01;
  delay = SYSCTL_RCGCTIMER_R;
  TIMER0_CTL_R &= ~0x00000001;
```

配置格式与重载数据信息

1. 设置模式为16bit计时模式；
2. 设置为周期模式；
3. 重载数据；

```
  TIMER0_CFG_R = 0x00000004;
  TIMER0_TAMR_R = 0x00000002;
  TIMER0_TAILR_R = period - 1;
```

对于第二个参数`TIMER0_TAMR_R`，可以设置为1或者2，设置为1时，标志着其只运行一遍，而设置为2时则会周期运行。

超时设置以及优先级设定

1. 清除超时flag;
2. 设置优先级

```
  TIMER0_ICR_R = 0x00000001;       // 6) clear timer0A timeout flag
  TIMER0_IMR_R |= 0x00000001;      // 7) arm timeout interrupt
  NVIC_PRI4_R = (NVIC_PRI4_R&0x00FFFFFF)|0x60000000; // 8) priority 3
```

设置优先级的时候，优先级数字越低则对应的优先级越高。这种设置也同样适用于对系统定时器的控制：

```
void SysTick_Init(uint32_t period){
  /***/
  NVIC_SYS_PRI3_R = (NVIC_SYS_PRI3_R&0x00FFFFFF)|0x40000000; //priority 2
  NVIC_ST_CTRL_R = 0x00000007;   // enable with core clock and interrupts
}
```

由于系统计时器的优先级为2（2<3），所以在二者同时中断的时候，系统会优先为系统定时器服务，即系统定时器会中断Time0计时器。此谓中断的中断。

正在运行的程序

第一层的中断Time0在作用的时候回中断某一些程序，这里为了使得实验结果更加明显，添加了一个优先级最低的线程：

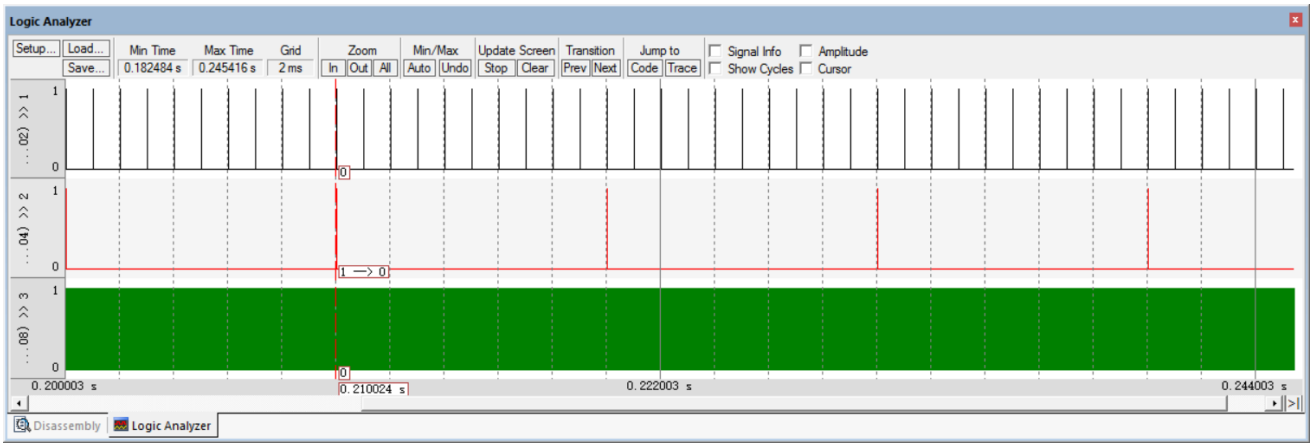
```
  while(1){
    //PA5 = PA5^0x20;
    PA5 = PA5^0x8;
  }
```

对A5的输出端口使用疑惑函数形成矩形方波波形。

问题阐释

在配置Time0定时器的时候，模拟结果显示分频并非是传入参数与内部`TIMER0_TAPR_R`共同作用，而是只有传入的参数起作用。

以下为笔者在实验的时候按照传入参数（即delay）为50000，内部变量`TIMER0_TAPR_R`为999配置的运行结果：



可以看到第一行（表示PF1输出端口的输出信息）的周期并非为我们预期的1s，而是：

$$
\begin{aligned}
T &= \frac{1}{f} \\
&= \frac{1}{50MHz/50000Hz} \\
&= 10^{-3}s
\end{aligned}
$$

对此笔者也不能解释，多方求问终究无果。