1. 实验题目

- **仿真实验**
  - 分析改动优先级前后的优先级嵌套现象，解释中断流程
- **板级运行**
  - 板级实验，了解系统定时器中断和Time定时中断的作用原理和区别

2. 实验内容

- **仿真实验**
  - 修改两种中断的优先级
- **板级运行**
  - 将PA端口换成PF端口
  - 设置每0.2s Time定时中断并PF2切换亮灭，改变定时初值观察结果
  - 系统定时器中断时间重新设定为10ms,在其中断服务程序中对PF3切换状态

3. 实验过程

代码理解

晶体振荡器除了可以使用数字电路分频以外，其频率几乎无法改变。如果采用PLL(锁相环)(相位锁栓回路，PhaseLockedLoop)技术，除了可以得到较广的振荡频率范围以外，其频率的稳定度也很高。

```
#define SYSDIV 3
#define LSB 1
// bus frequency is 400MHz/(2*SYSDIV+1+LSB) = 400MHz/(2*3+1+1) = 50 MHz
void PLL_Init(void){
  // 1) configure the system to use RCC2 for advanced features
  //    such as 400 MHz PLL and non-integer System Clock Divisor
  SYSCTL_RCC2_R |= SYSCTL_RCC2_USERCC2;
  // 2) bypass PLL while initializing
  SYSCTL_RCC2_R |= SYSCTL_RCC2_BYPASS2;
  // 3) select the crystal value and oscillator source
  SYSCTL_RCC_R &= ~SYSCTL_RCC_XTAL_M;   // clear XTAL field
  SYSCTL_RCC_R += SYSCTL_RCC_XTAL_16MHZ;// configure for 16 MHz crystal
  SYSCTL_RCC2_R &= ~SYSCTL_RCC2_OSCSRC2_M;// clear oscillator source field
  SYSCTL_RCC2_R += SYSCTL_RCC2_OSCSRC2_MO;// configure for main oscillator source
  // 4) activate PLL by clearing PWRDN
  SYSCTL_RCC2_R &= ~SYSCTL_RCC2_PWRDN2;
  // 5) use 400 MHz PLL
  SYSCTL_RCC2_R |= SYSCTL_RCC2_DIV400;
  // 6) set the desired system divider and the system divider least significant bit
  SYSCTL_RCC2_R &= ~SYSCTL_RCC2_SYSDIV2_M;  // clear system clock divider field
  SYSCTL_RCC2_R &= ~SYSCTL_RCC2_SYSDIV2LSB; // clear bit SYSDIV2LSB

// set SYSDIV2 and SYSDIV2LSB fields
  SYSCTL_RCC2_R += (SYSDIV<<23)|(LSB<<22);  // divide by (2*SYSDIV+1+LSB)
```

```
  // 7) wait for the PLL to lock by polling PLLLRIS
  while((SYSCTL_RIS_R&SYSCTL_RIS_PLLLRIS)==0){};
  // 8) enable use of PLL by clearing BYPASS
  SYSCTL_RCC2_R &= ~SYSCTL_RCC2_BYPASS2;
}
```

此处直接系统中自带的400MHz的锁相环电路，主要就是直接使用RCC2上的PLL功能。设置SYSCTL_RCC2_R += (SYSDIV<<23)|(LSB<<22)也就是分频公式为(2*SYSDIV+1+LSB)，然后结合之前的的宏定义进行8分频，得到50MHz

```
volatile uint32_t Counts;
void SysTick_Init(uint32_t period){
  Counts = 0;
  NVIC_ST_CTRL_R = 0;            // disable SysTick during setup
  NVIC_ST_RELOAD_R = period - 1; // reload value
  NVIC_ST_CURRENT_R = 0;         // any write to current clears it
  NVIC_SYS_PRI3_R = (NVIC_SYS_PRI3_R&0x00FFFFFF)|0x40000000; //priority 2
  NVIC_ST_CTRL_R = 0x00000007;   // enable with core clock and interrupts
}
void SysTick_Handler(void){
  PA4 = 0x10;
  Counts = Counts + 1;
  PA4 = 0;
}
```

这段代码是系统定时器的初始化，与之前不同的是这里额外设置了NVIC_SYS_PRI3_R = (NVIC_SYS_PRI3_R&0x00FFFFFF)|0x40000000; 也就是其中断优先级为2，系统定时器到了计时时间就会执行系统中断。执行的内容为SysTick_Handler()，做的是将PA4置1，然后执行一次Counts++,之后PA4置0造成一个脉冲。

```
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
}
void Timer0A_Handler(void){
  PA3 = 0x08;
  TIMER0_ICR_R = TIMER_ICR_TATOCINT;// acknowledge timer0A timeout
  PA3 = 0;
}
```

此段代码为Timer0的初始化和其计时完成之后会调用的中断函数。上面函数中TIMER0_TAPR_R为预分频数，period为周期大小，这两个可以设置计时的时间，此处设置的预分频数TIMER0_TAPR_R=49，也就是间隔50计数一次，此时通过之前分频的PLL50Mhz，计算我们得到此时是1us计时一次，此时当达到周期要求时调用中断函数执行，执行的为将PA3置1，设置计时结束flag，再把PA3置0，此时PA3也是一个脉冲。 NVIC_PRI4_R = (NVIC_PRI4_R&0x00FFFFFF)|0x60000000; 这里把Timer0的优先级设置为3。

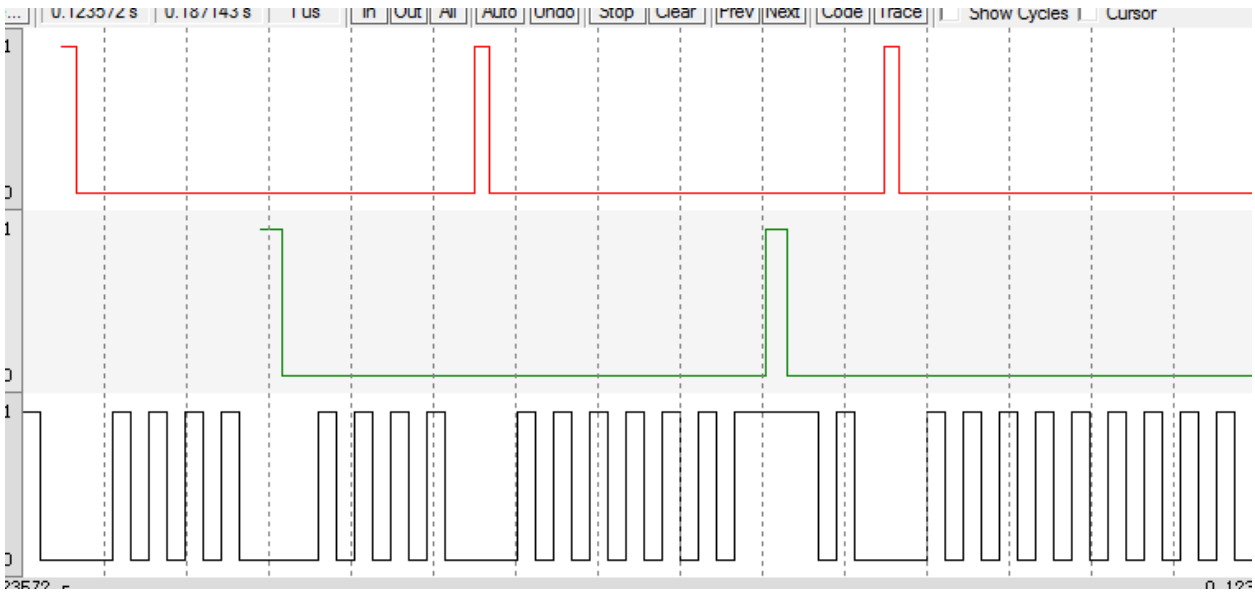
如果我们需要修改上述两个中断执行的函数名，需要在startup.s进行设定

```
int main(void){
  DisableInterrupts();
  PLL_Init();                // configure for 50 MHz clock
  SYSCTL_RCGCGPIO_R |= 0x01;   // 1) activate clock for Port A
  while((SYSCTL_PRGPIO_R&0x01) == 0){};
  GPIO_PORTA_AMSEL_R &= ~0x38;    // disable analog function
  GPIO_PORTA_PCTL_R &= ~0x00FFF000; // GPIO
  GPIO_PORTA_DIR_R |= 0x38;  // make PA5-3 outputs
  GPIO_PORTA_AFSEL_R &= ~0x38;// disable alt func on PA5-3
  GPIO_PORTA_DEN_R |= 0x38;  // enable digital I/O on PA5-3
                             // configure PA5-3 as GPIO
  Timer0A_Init(5);           // 200 kHz
  SysTick_Init(304);         // 164 kHz
  EnableInterrupts();
  while(1){
    PA5 = PA5^0x20;
  }
}
```

main函数中执行的是初始化PLL得到50MHz时钟，然后初始PA5-3。之后将Timer0周期设置为5，也就是5us执行一次中断，此时频率为200KHz(1/5us)，再讲系统定时器设置为164Khz(50MHz/304)。最后设置一个死循环使得PA5在0-1跳变。

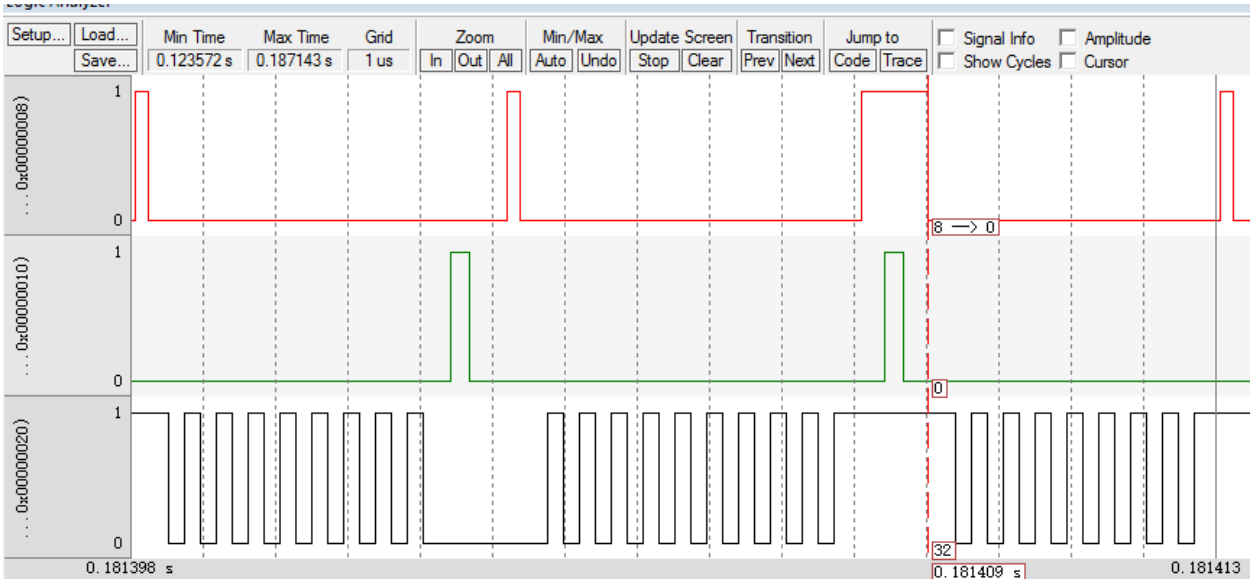
仿真实验

- 根据之前的代码分析，在初始化完成后PA5死循环取反，也就是PA5输出为矩形波。



此时我们能看到的是当Timer0或者是系统计时器计时结束时都会产生中断调用相应的中断函数，我们从PA3和PA4的矩形脉冲中就可以看出来，两者之一执行中断时PA5被挂起，不在执行自异或的取反操作。

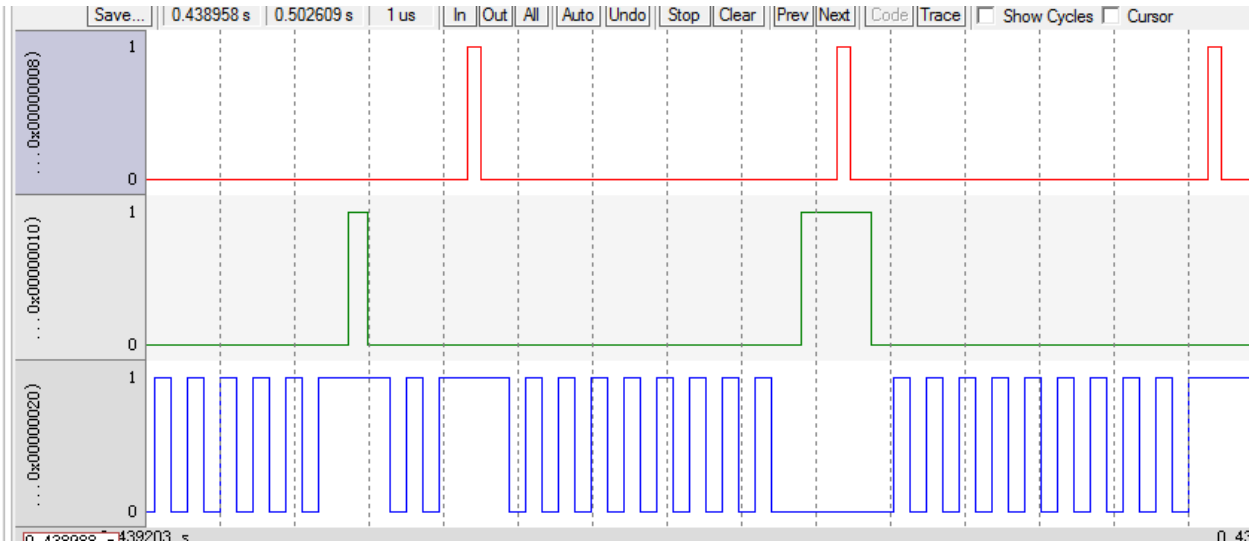
- 当出现Timer0和系统定时器中断同时产生时



此时系统定时器的优先级为2高于timer0的3(越小越高)，出现PA3正在执行置1然后设置计时flag时被系统定时器的引起的中断挂起，PA4出现脉冲，之后PA3才继续执行置0操作。此时肉眼可见PA3的脉冲宽度由于被系统定时器中断挂起的原因而宽了很多。此时因为PA5没有定义优先级所以是最低优先级，所以不论在timer0或者系统计时器执行中断时他都会被挂起。

- **修改优先级**

```
void SysTick_Init(uint32_t period){
  NVIC_SYS_PRI3_R = (NVIC_SYS_PRI3_R&0x00FFFFFF)|0x60000000; //priority 3
}
void Timer0A_Init(unsigned short period){ volatile uint32_t delay;
  NVIC_PRI4_R = (NVIC_PRI4_R&0x00FFFFFF)|0x40000000; // 8) priority 2
}
```



此时因为优先级对调，所以当系统定时器出现执行中断时，若此时Timer0也要执行中断，那么系统定时器的中断就会被挂起，先执行timer0的中断，所以此时的中断嵌套执行顺序发生了变化。说明中断执行顺序和优先级有关。

板级运行

- 首先我们修改main函数中的端口初始化部分

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

SYSCTL_RCGCGPIO_R |= SYSCTL_RCGC2_GPIOF;   // 1) activate clock for Port F
while((SYSCTL_PRGPIO_R&0x20) == 0){};
GPIO_PORTF_AMSEL_R &= ~0x0E;    // disable analog function
GPIO_PORTF_PCTL_R &= ~0x0000FFF0; // GPIO
GPIO_PORTF_DIR_R |= 0x0E;  // make PF3-1 outputs
GPIO_PORTF_AFSEL_R &= ~0x0E;// disable alt func on PF3-1
GPIO_PORTF_DEN_R |= 0x0E;  // enable digital I/O on PF3-1
                           // configure PF3-1 as GPIO
```

- 然后修改中断执行内容
  - 设置每0.2s Time定时中断并PF2切换亮灭，改变定时初值观察结果

```
void Timer0A_Handler(void){
  PF2 =PF2^0x04;
  TIMER0_ICR_R = TIMER_ICR_TATOCINT;// acknowledge timer0A timeout
}
```

此时需要Timer0位5hz，所以(TIMER0_TAPR_R+1)×period=10,000,000

Timer0A_Init中

```
TIMER0_TAPR_R = 199;
```

周期修改为

```
Timer0A_Init(50000);
```

  - 系统定时器中断时间重新设定为10ms,在其中断服务程序中对PF3切换状态

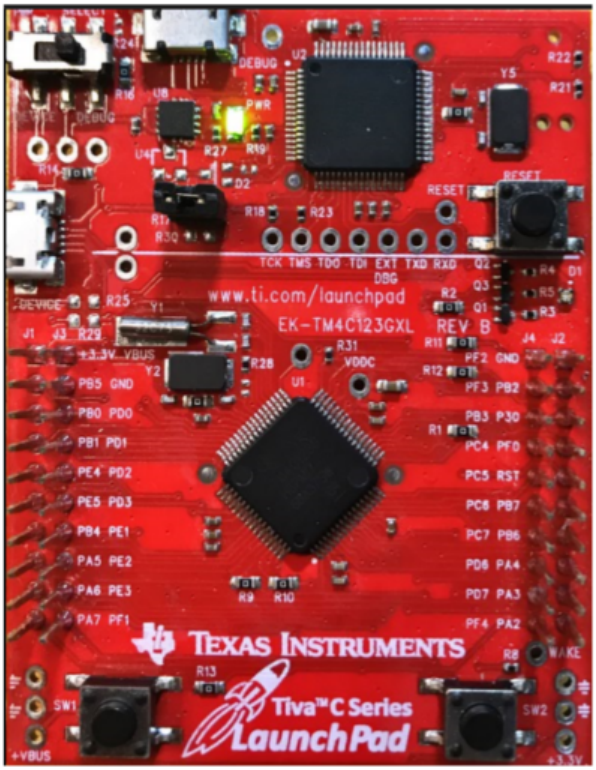
```
void SysTick_Handler(void){
  PF3 = PF3^0x08;
  Counts = Counts + 1;
}
```

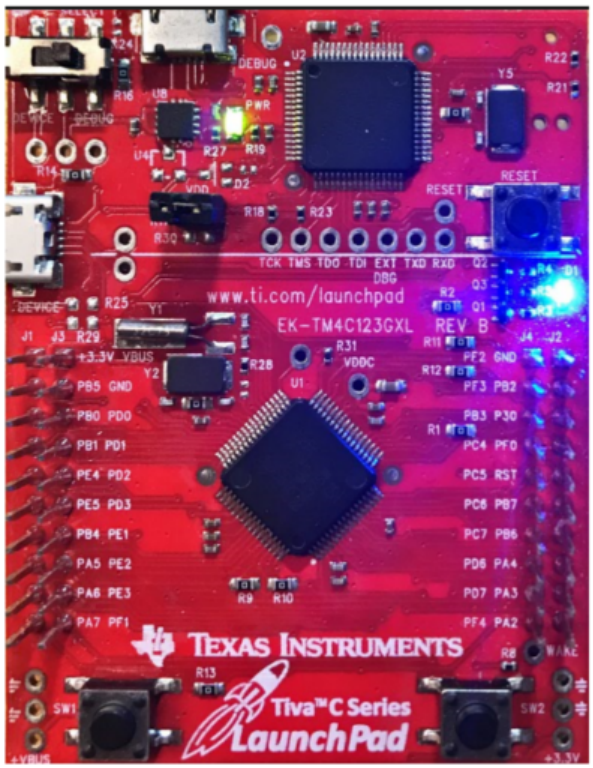
此时定时10ms也就是100hz，50MHz/100hz=500000;

所以main修改为

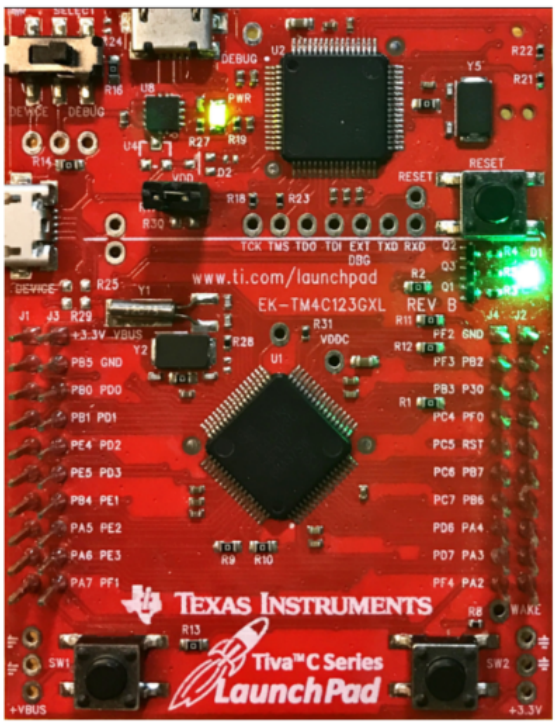
```
SysTick_Init(500000)
```

- 效果如下
  - 设置每0.2s Time定时中断并PF2切换亮灭





  - 系统定时器中断时间重新设定为10ms,在其中断服务程序中对PF3切换状态



不过这个100hz已经超过人眼可范围了一般会被认为是常亮了

5. 实验心得

这次的实验主要Time和系统定时器的使用，这两者都是用来做计时器的，用法的话根源上还是分频，主要就是分频时候的计算，其他的没有什么困难，这两个的分频方式来说Time的限制比较大，而系统定时器，由于有24bit，所以用来分频限制较小。