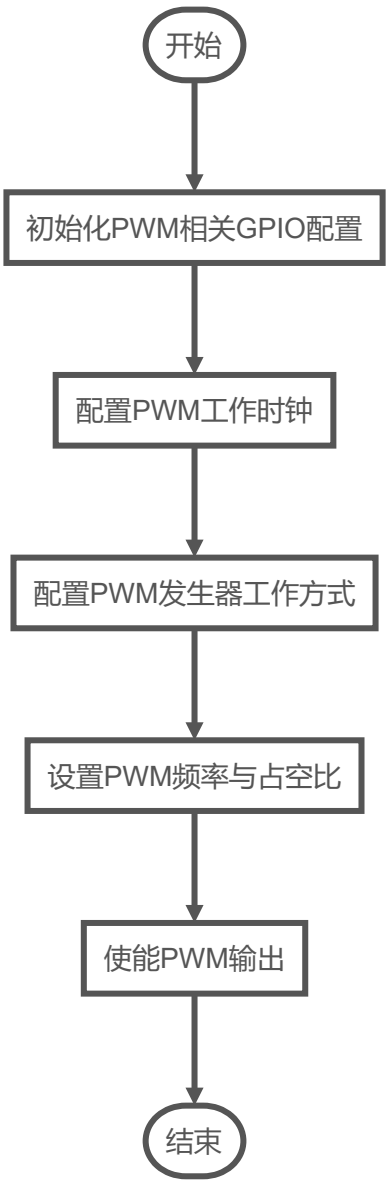
# 实验三

- 实验目标
    1. 了解 PWM 基本概念。
    2. 学习 TM4C129x Series Cortex-M4 的 PWM 工作原理与方式。
    3. 学习 PWM 相关库函数的使用。
    4. 学习对 PWM 模块输出周期和占空比的设置。
- 流程设计




- 代码

```
//PWM初始化
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
```

```
    GPIOPinConfigure(GPIO_PF1_M0PWM1);
    GPIOPinConfigure(GPIO_PF2_M0PWM2);
    GPIOPinConfigure(GPIO_PF3_M0PWM3);
    GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_1);
    GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_2);
    GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_3);
    PWMGenConfigure(PWM0_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN |
                    PWM_GEN_MODE_NO_SYNC);
    PWMGenConfigure(PWM0_BASE, PWM_GEN_1, PWM_GEN_MODE_DOWN |
                    PWM_GEN_MODE_NO_SYNC);
    PWMGenPeriodSet(PWM0_BASE, PWM_GEN_0, PWM_Frequence);
    PWMGenPeriodSet(PWM0_BASE, PWM_GEN_1, PWM_Frequence);
    PWMOutputState(PWM0_BASE, PWM_OUT_1_BIT | PWM_OUT_2_BIT | PWM_OUT_3_BIT, true);
    PWMGenEnable(PWM0_BASE, PWM_GEN_0);
    PWMGenEnable(PWM0_BASE, PWM_GEN_1);
}

//PWM输出控制
void PWM_Set(int index,float t)
{
    uint32_t PWMPulseWidth = PWM_Frequence * t;
    switch(index)
    {
    case 1:
        PWMPulseWidthSet(PWM0_BASE, PWM_OUT_1,PWMPulseWidth);
        break;
    case 2:
        PWMPulseWidthSet(PWM0_BASE, PWM_OUT_2,PWMPulseWidth);
        break;
    case 3:
        PWMPulseWidthSet(PWM0_BASE, PWM_OUT_3,PWMPulseWidth);
        break;
    default:
        break;
    }

}

//PWM测试函数
void PWM_Test()
{
    float t=0;
    for(;;)
    {
        PWM_Set(1,t);
        SysCtlDelay(10*16000000/3000);
        t += 0.01;
        if(t >= 1)t=0;
        PWM_Set(2,t);
        SysCtlDelay(10*16000000/3000);
        t += 0.01;
        if(t >= 1)t=0;
        PWM_Set(3,t);
        SysCtlDelay(10*16000000/3000);
        t += 0.01;
        if(t >= 1)t=0;
    }
}
```

```c
int main(void)
{
    System_Init();
    PWM_Test();
    while(1)
    {
        ;
    }
    return 0;
}
```

- 运行效果

  D1,D2,D3会轮流呼吸。

- 思考题

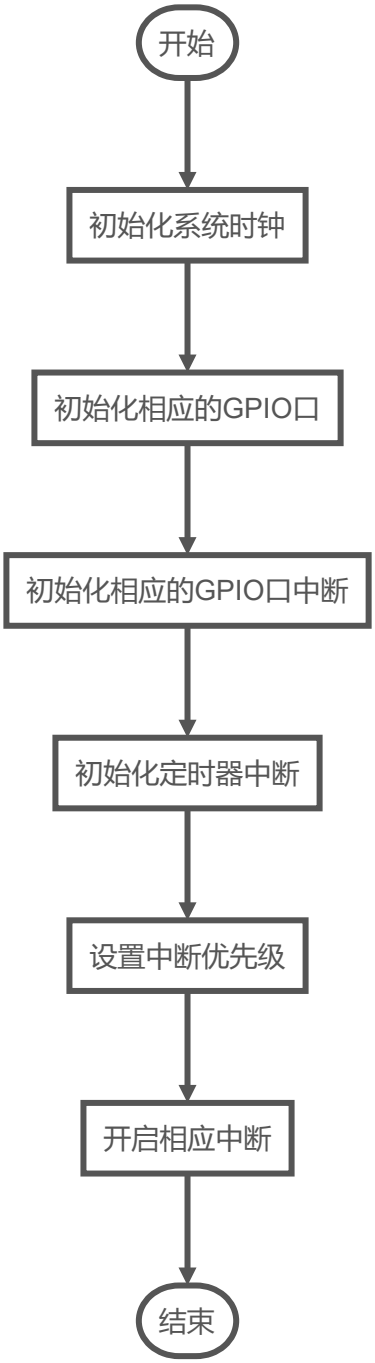
  1. 占空比不变时，PWM频率高低不影响灯的亮度。
  2. flip过大，灯一下子变亮，看不清灯慢慢变亮的过程；flip过小，灯变亮速度很慢，效果也不明显。
  3. 直接使用寄存器，运行效率高，但是配置繁琐，记忆不便。使用库函数，记忆和使用方便，但是运行效率不如直接配置寄存器。

# 实验四

- 实验目标

  1. 了解 M4 中断的概念，中断处理系统的工作原理。
  2. 了解对中断的设置与控制方法。
  3. 熟悉中断处理过程，掌握中断处理子程序的书写格式和使用方法。

- 流程设计




- 代码

```
//按键初始化
void KEY_Init()
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOP);
    GPIOPinTypeGPIOInput(GPIO_PORTP_BASE, GPIO_PIN_2);
    GPIOPadConfigSet(GPIO_PORTP_BASE, GPIO_PIN_2,
                     GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);
    GPIOPinTypeGPIOInput(GPIO_PORTN_BASE, GPIO_PIN_3|GPIO_PIN_2);
    GPIOPadConfigSet(GPIO_PORTN_BASE, GPIO_PIN_3|GPIO_PIN_2,
                     GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);
```

```
13
14      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
15      GPIODirModeSet(GPIO_PORTD_BASE, GPIO_PIN_1, GPIO_DIR_MODE_OUT);
16      GPIOPadConfigSet(GPIO_PORTD_BASE, GPIO_PIN_1,
17                       GPIO_STRENGTH_8MA_SC, GPIO_PIN_TYPE_STD);
18      GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_1, 0);
19      GPIOPinTypeGPIOInput(GPIO_PORTD_BASE, GPIO_PIN_0);
20      GPIOPadConfigSet(GPIO_PORTD_BASE, GPIO_PIN_0,
21                       GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);
22
23      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOH);
24      GPIODirModeSet(GPIO_PORTH_BASE, GPIO_PIN_3|GPIO_PIN_2, GPIO_DIR_MODE_OUT);
25      GPIOPadConfigSet(GPIO_PORTH_BASE, GPIO_PIN_3|GPIO_PIN_2,
26                       GPIO_STRENGTH_8MA_SC, GPIO_PIN_TYPE_STD);
27      GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3|GPIO_PIN_2, 0);
28
29      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOM);
30      GPIODirModeSet(GPIO_PORTM_BASE, GPIO_PIN_3, GPIO_DIR_MODE_OUT);
31      GPIOPadConfigSet(GPIO_PORTM_BASE, GPIO_PIN_3,
32                       GPIO_STRENGTH_8MA_SC, GPIO_PIN_TYPE_STD);
33      GPIOPinWrite(GPIO_PORTM_BASE, GPIO_PIN_3, 0);
34  }
35
36  //按键中断初始化
37  void KEY_Int_Init()
38  {
39
40      GPIOIntRegister(GPIO_PORTP_BASE, PortPIntHandler);
41      GPIOIntTypeSet(GPIO_PORTP_BASE, GPIO_PIN_2, GPIO_FALLING_EDGE);
42      GPIOIntEnable(GPIO_PORTP_BASE, GPIO_PIN_2);
43
44      GPIOIntRegister(GPIO_PORTN_BASE, PortNIntHandler);
45      GPIOIntTypeSet(GPIO_PORTN_BASE, GPIO_PIN_3|GPIO_PIN_2, GPIO_FALLING_EDGE);
46      GPIOIntEnable(GPIO_PORTN_BASE, GPIO_PIN_3|GPIO_PIN_2);
47
48
49      GPIOIntRegister(GPIO_PORTD_BASE, PortDIntHandler);
50      GPIOIntTypeSet(GPIO_PORTD_BASE, GPIO_PIN_0, GPIO_FALLING_EDGE);
51      GPIOIntEnable(GPIO_PORTD_BASE, GPIO_PIN_0);
52
53  }
54
55  //定时器中断初始化
56  void TimerIntInitial(void)
57  {
58      SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
59      TimerConfigure(TIMER0_BASE, TIMER_CFG_SPLIT_PAIR  | TIMER_CFG_B_PERIODIC);
60      TimerLoadSet(TIMER0_BASE, TIMER_B, g_ui32SysClock/1000 );
61      IntRegister(INT_TIMER0B,Timer0BIntHandler);
62      TimerIntEnable(TIMER0_BASE, TIMER_TIMB_TIMEOUT);
63  }
64
65  //按键中断处理函数
66  void PortNIntHandler(void)
67  {
68      unsigned long Status;
69      Status=GPIOIntStatus(GPIO_PORTN_BASE,true);
70      if(Status==GPIO_INT_PIN_2)
71      {   for(int i =0; i<5; i++) {
72              LED_Open(2);
```

```
            SysCtlDelay(100*(16000000/3000));//2
            LED_Close(2);
            SysCtlDelay(100*(16000000/3000));//2
        }
    }
    if(Status==GPIO_INT_PIN_3)
    {   for(int i =0; i<5; i++) {
            LED_Open(4);
            SysCtlDelay(100*(16000000/3000));//2
            LED_Close(4);
            SysCtlDelay(100*(16000000/3000));//2
        }
    }
    GPIOIntClear(GPIO_PORTN_BASE,Status);

}

void PortDIntHandler(void)
{
    unsigned long Status;
    Status=GPIOIntStatus(GPIO_PORTD_BASE,true);
    if(Status==GPIO_INT_PIN_0)
    {   for(int i =0; i<5; i++) {
            LED_Open(3);
            SysCtlDelay(200*(16000000/3000));//2
            LED_Close(3);
            SysCtlDelay(200*(16000000/3000));//2
        }
    }
    GPIOIntClear(GPIO_PORTD_BASE,Status);
}

//定时器中断处理函数
void Timer0BIntHandler(void)
{   //volatile uint32_t i;
    unsigned long Status;
    TimerDisable(TIMER0_BASE, TIMER_B);
    Status=TimerIntStatus(TIMER0_BASE,true);

    if(Status==TIMER_TIMB_TIMEOUT)
    {
        GPIOPinWrite(GPIO_PORTL_BASE, GPIO_PIN_1, 0xff);
        SysCtlDelay(500*(16000000/3000));//2
        GPIOPinWrite(GPIO_PORTL_BASE, GPIO_PIN_1, 0x00);
        SysCtlDelay(500*(16000000/3000));//2
    }

    TimerIntClear(TIMER0_BASE, Status);
    TimerLoadSet(TIMER0_BASE, TIMER_B, g_ui32SysClock/1000 );
    TimerEnable(TIMER0_BASE, TIMER_B);
}

//中断设置函数
void NVIC_Configure()
{
    IntPrioritySet(INT_GPION,0x30);
    IntPrioritySet(INT_GPIOD,0x40);
    IntPrioritySet(INT_TIMER0B, 0xe0);

    IntEnable(INT_GPION);
```

```
    IntEnable(INT_GPIOD);
    IntEnable(INT_TIMER0B);
}

int main(void)
{
    System_Init();
    NVIC_Configure();
    IntMasterEnable();
    TimerEnable(TIMER0_BASE, TIMER_B);

    while(1)
    {

    }

    return 0;
}

```

- 运行效果

  在没有触发外部中断时,定时器会以一定频率进行闪烁，当高优先级外部中断被触发时，会进入外部中断处理函数，定时器中断挂起。当外部中断处理函数完成时，返回定时器中断，定时器中断处理函数继续运行。更高等级的外部中断触发时，低等级的外部中断处理函数会挂起。知道更高等级的外部中断处理函数完成才会继续运行。

- 思考题

  中断优先级十分重要，系统优先响应高等级的中断。