華中科技大學

·DSP原理及应用·实验报告

专业 ： 人工智能

班级 ： 人工智能本硕博 2001 班

学 号 : U202015237

姓 名 : 罗亚文

报告日期: 2023-2-25

人工智能与自动化学院

目录

目录

1 定时器和GPIO实验 2
1.1 实验要求 2
1.2 实验要求 2
1.3 实验说明 2
1.4 实验思路 3
1.5 实验代码 3
1.6 实验调试 6
2 PWM和eCAP实验 6
2.1 实验要求 7
2.2 实验目的 7
2.3 实验说明 7
2.4 实验思路 7
2.5 实验代码 7
2.6 实验总结 10
3 ADC实验 10
3.1 实验要求 10
3.2 实验目的 11
3.3 实验说明 11
3.4 实验思路 11
3.5 实验代码 11
3.6 实验调试 15
4 综合实验 15
4.1 实验要求 15
4.2 实验目的 16
4.3 实验思路 16
4.4 实验代码 16
4.5 实验调试 21

1 定时器和GPIO实验

1.1 实验要求

(1)通过定时器实现延时，编写跑马灯程序；

(2)对主板上的按键GPIO12进行采样，添加软件去抖功能；

(3)实现按键对跑马灯动作的选择，至少两种跑马灯动作；

(4)学习TM1638驱动程序，完成数码管的显示功能。

1.2 实验要求

(1)掌握GPIO端口的配置和操作方法；

(2)掌握定时器的配置和操作方法；

(3)掌握定时器中断的配置；

(4)学会使用GPIO实现TM1638的驱动，实现数码管的显示功能。

1.3 实验说明

定时器控制LED灯闪烁和数码管的显示。

点亮LED灯的实验称得上是开发板中的“Hello World!”，一般初学者都会首先尝试此实验。在这个实验中，可以分别采用查询和中断的方式来获知定时器计数器的溢出。然后操纵GPIO控制LED亮或是不亮。自己设计跑马灯的程序，以定时器实现延时，练习定时器和GPIO的使用。

LED的控制是通过GPIO输出实现的，按键的采样则是通过GPIO输入实现的。通过读取GPIO12端口的状态，判断按键是否按下。机械按键存在抖动的问题，因此添加软件去抖程序实现更稳定的采样。

底板上有LED数码管显示模块及其驱动电路，数码管的驱动是通过TM1638实现的，TM1638是LED驱动控制专用电路，能实现LED驱动、键盘扫描等功能。其控制输入引脚只有三个，因此可以大大节约控制芯片的GPIO引脚的占用。通过一块TM1638数码管驱动芯片，可以实现8位数码管的驱动。TM1638的驱动程序是通过操作GPIO引脚实现的，TM1638的底层驱动程序已由例程给出，可参考例程学会TM1638的驱动方法。LED数码管的显示是对GPIO的巩固与加强，通过对TM1638的驱动，加深对GPIO功能的理解。

注：四个LED的GPIO端口分别为GPIO0、GPIO1、GPIO2、GPIO3。

考察的知识包含：时钟的配置、定时器配置、中断配置、GPIO配置。可参看实验板的配套程序(Clock电子钟的设计)

1.4 实验思路

该实验主要实现四个功能：

1、定时器计时并产生中断。

2、GPIO驱动LED模块亮灯，即实现跑马灯。

3、驱动LCD显示定时器中断。

4、外部输入按键GPIO12实现中断，使得跑马灯的模式进行切换。

该实验的大致思路如下：需要设置俩个中断，外部中断和定时器中断，外部中断用于切换跑马灯模式，定时器中断用于计时并且驱动LCD模块显示，而LED的跑马灯则由GPIO的接口输出可以较为简单的实现。

1.5 实验代码

实现的代码如下：

```
#include "DSP28x_Project.h"
#include "LED_TM1638.h"

 interrupt void cpu_timer0_isr(void);  //timer0
 interrupt void myXint1_isr(void);     //xint1
// interrupt void EPWM4Int_isr(void);    //EPWM4
// interrupt void Ecap1Int_isr(void);    //ECAP1
// interrupt void MyAdcInt1_isr(void);   //ADCINT1

//初始化变量
       int hourH =0 ;int hourL =0;
       int minH = 0;int minL = 0;
       int secH = 0;int secL =0; int TenmS =0;
       int HorseType = 0;//跑马灯的类型
       int keyDLTime = 0;//设置去抖动
       int LedFlashCtr = 0 ;//用于数码管的显示
       int NewLedEn = 0;
       int Begin =1;

void HorseRunning(int16 no);

#define Led0Blink() GpioDataRegs.GPACLEAR.bit.GPIO0 = 1
#define Led1Blink() GpioDataRegs.GPACLEAR.bit.GPIO1 = 1
#define Led2Blink() GpioDataRegs.GPACLEAR.bit.GPIO2 = 1
#define Led3Blink() GpioDataRegs.GPACLEAR.bit.GPIO3 = 1
#define Led0Blank() GpioDataRegs.GPASET.bit.GPIO0 = 1
#define Led1Blank() GpioDataRegs.GPASET.bit.GPIO1 = 1
#define Led2Blank() GpioDataRegs.GPASET.bit.GPIO2 = 1
#define Led3Blank() GpioDataRegs.GPASET.bit.GPIO3 = 1

void  Xint1_Init()
{
    EALLOW;
    GpioCtrlRegs.GPAPUD.bit.GPIO12 = 0;
    GpioCtrlRegs.GPAMUX1.bit.GPIO12 = 0;
    GpioCtrlRegs.GPADIR.bit.GPIO12 = 0;
```

```
    GpioIntRegs.GPIOXINT1SEL.bit.GPIOSEL = 12;
    XIntruptRegs.XINT1CR.bit.POLARITY = 0;//下降沿
    XIntruptRegs.XINT1CR.bit.ENABLE = 1;
    EDIS;
}

void HorseIO_Init()
{
    EALLOW;
    GpioDataRegs.GPASET.bit.GPIO0 = 1;
    GpioDataRegs.GPASET.bit.GPIO1 = 1;
    GpioDataRegs.GPASET.bit.GPIO2 = 1;
    GpioDataRegs.GPASET.bit.GPIO3 = 1;
    GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 0;
    GpioCtrlRegs.GPADIR.bit.GPIO0 = 1;
    GpioCtrlRegs.GPAMUX1.bit.GPIO1 = 0;
    GpioCtrlRegs.GPADIR.bit.GPIO1 = 1;
    GpioCtrlRegs.GPAMUX1.bit.GPIO2 = 0;
    GpioCtrlRegs.GPADIR.bit.GPIO2 = 1;
    GpioCtrlRegs.GPAMUX1.bit.GPIO3 = 0;
    GpioCtrlRegs.GPADIR.bit.GPIO3 = 1;
    EDIS;

}

void DelaymS(int tm)
{
  int i;
  unsigned int j;
    for(i = 0;i < tm ;i++){
      j =  60000;
      while(j != 0)j--;
      }
}
void main(void)
{
      InitSysCtrl();    //初始化系统时钟，选择内部晶振1，10MHZ，12倍频，2分频，初始化外设时
钟，低速外设,4分频
      DINT;                 //关总中断
      IER = 0x0000;         //关CPU中断使能
      IFR = 0x0000;     //清CPU中断标志
      InitPieCtrl();    //关pie中断
      InitPieVectTable();   //清中断向量表
      EALLOW;               //配置中断向量表
      PieVectTable.TINT0 = &cpu_timer0_isr;
      PieVectTable.XINT1 = &myXint1_isr;
      // PieVectTable.ECAP1_INT = &Ecap1Int_isr;
      // PieVectTable.EPWM4_INT = &EPWM4Int_isr;
      // PieVectTable.ADCINT1 = &MyAdcInt1_isr;
      PieCtrlRegs.PIEIER1.bit.INTx7 = 1;
      PieCtrlRegs.PIEIER1.bit.INTx4 = 1;
      IER |= 1;
      EDIS;


     //  MemCopy(&RamfuncsLoadStart, &RamfuncsLoadEnd, &RamfuncsRunStart);
      InitFlash();
      HorseIO_Init();
      Xint1_Init();
```

```
        TM1638_Init();          //初始化LED


        InitCpuTimers();        // 初始化定时器
        ConfigCpuTimer(&CpuTimer0, 60, 10000);//设置时钟的响应时间为0.01s
        EALLOW;
        CpuTimer0Regs.TCR.bit.TRB = 1;
        CpuTimer0Regs.TCR.bit.TSS = 0;
        EDIS;



        EINT;//中断总使能
        ERTM;


        while(1)
        {
          //if(NewLedEn == 0)
            {
                LED_Show(1,(TenmS % 10),0);
                LED_Show(2,(TenmS / 10),0);
                LED_Show(3,secL,1);
                LED_Show(4,secH,0);
                LED_Show(5,minL,1);
                LED_Show(6,minH,0);
                LED_Show(7,hourL,1);
                LED_Show(8,hourH,0);
                NewLedEn = 1;
            }

        }

}

void HorseRunning(int no)
{
    if(no & 0x1)Led0Blink();
    else Led0Blank();
    if(no & 0x2)Led1Blink();
    else Led1Blank();
    if(no & 0x4)Led2Blink();
    else Led2Blank();
    if(no & 0x8)Led3Blink();
    else Led3Blank();
}
void HorseRunning1(int no)
{
    if(no & 0x1)    Led0Blink();
    else    Led0Blank();
    if(no & 0x2)
        {Led0Blink();Led1Blink();}
    else
        {Led0Blank();Led1Blank();}
    if(no & 0x4)
        {Led2Blink();Led1Blink();}
    else
        {Led2Blank();Led1Blank();}
    if(no & 0x8)
        {Led3Blink();Led2Blink();}
```

```
    else
        {Led3Blank();Led2Blank();}

}


interrupt void myXint1_isr(void)
{
    Begin = 1;
    if((HorseType == 0)&&(keyDLTime>20))
        {
            HorseType = 1; keyDLTime = 0;
        }
        else if((HorseType == 1)&&(keyDLTime>20))
        {
            HorseType = 0; keyDLTime = 0;
        }
   PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;//手动清楚这里的中断标志 all=0x0001也行
   EALLOW;
   CpuTimer0Regs.TCR.bit.TRB = 1;
   CpuTimer0Regs.TCR.bit.TSS = 0;
   EDIS;
}

interrupt void cpu_timer0_isr(void) {
    keyDLTime++;
    LedFlashCtr++;
    if((LedFlashCtr & 0xf)==0) NewLedEn = 0;
    if(Begin == 1){
        TenmS ++;
        if(TenmS == 100){TenmS = 0; secL++;}
        if(secL == 10){secL = 0; secH++;}
        if(secH == 6){secH = 0; minL++;}
        if(minL == 10){minL = 0; minH++;}
        if(minH == 6){minH = 0; hourL++;}
        if(hourH == 2 && hourL ==4){hourH = 0; hourL =0;}
        else if(hourL == 10){hourL = 0; hourH++;}
    }
    if(HorseType == 0 && Begin == 1){HorseRunning((LedFlashCtr&0xf0)>>4);}
    else if(HorseType == 1 && Begin == 1){HorseRunning1((LedFlashCtr&0xf0)>>4);}
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}
```

1.6 实验调试

在实验中主要遇见的问题是最初的代码，外部输入GPIO12按键无法触发中断，按键不起作用。

最后调试发现是没有开PIEACK，改正代码如下：

```
PieCtrlRegs.PIEIER1.bit.INTx7 = 1;
PieCtrlRegs.PIEIER1.bit.INTx4 = 1;
IER |= 1;
```

2 PWM和eCAP实验

2.1 实验要求

(1)输出带死区的互补PWM波形；

(2)通过eCAP单元捕获其中一路PWM波形并获取其周期和占空比。

2.2 实验目的

(1) 掌握PWM的周期、死区电平和死区时间的设定方法；

(2) 掌握PWM计数周期中断和比较值匹配中断的配置方法。

(3) 掌握eCAP单元的操作方法及其中断的配置；

(4) 掌握eCAP单元时间测量和占空比测量的方法。

2.3 实验说明

输出 PWM 波形，并通过 eCAP 捕获 PWM 脉冲。

配置 EPWMxA 和 EPWMxB 输出带死区的互补 PWM 波形，输出端口在转接板上的 X3 端子上（排针端子），具体芯片引脚参见第一章中的引脚定义。

在输出 PWM 的基础上，将其中一路 PWM 输出引脚接到 eCAP 捕获引脚上，通过捕获单元获取 PWM 的周期和占空比信息。

2.4 实验思路

该实验主要实现俩个功能：

1、配置PWM输出指定的脉冲波形从GPIO口输出所生成的波形。

2、eCAP对输出的PWM信号进行捕获，输出PWM的周期以及死区信号的时钟周期数。

该实验的大致思路如下：需要设置eCAP触发一个中断，并在中断服务程序中对PWM信号的相关内容进行计算，驱动LED模块展示输出结果。

2.5 实验代码

实现的代码如下：

```
#include "DSP28x_Project.h"
#include "LED_TM1638.h"
#include "DSP2802x_ECap.h"

// interrupt void EPWM4Int_isr(void);    //EPWM4
 interrupt void Ecap1Int_isr(void);    //ECAP1

//初始化变量
int t1 = 0; int t2 = 0;int t3 = 0;int t4 = 0;
int DutyOnTime = 0;int DutyOffTime = 0;
int Period = 0;
```

```
float rate = 0;

void Gpio_Init(void);
void InitPWM4(void);
void InitCap1(void);
void DelaymS(int tm);

void DelaymS(int tm)
{
  int i;
  unsigned int j;
    for(i = 0;i < tm ;i++){
      j =  60000;
      while(j != 0)j--;
      }
}

void InitCap1(void){
    ECap1Regs.ECEINT.all = 0x0000;//中断向量初始化
    ECap1Regs.ECCLR.all = 0xFFFF;
    ECap1Regs.ECCTL1.bit.PRESCALE = 0;//1分频
    ECap1Regs.ECCTL1.bit.CAP1POL = 0;//上下沿均触发
    ECap1Regs.ECCTL1.bit.CAP2POL = 1;
    ECap1Regs.ECCTL1.bit.CAP3POL = 0;
    ECap1Regs.ECCTL1.bit.CAP4POL = 1;
    ECap1Regs.ECCTL1.bit.CTRRST1 = 1;//差分模式
    ECap1Regs.ECCTL1.bit.CTRRST2 = 1;
    ECap1Regs.ECCTL1.bit.CTRRST3 = 1;
    ECap1Regs.ECCTL1.bit.CTRRST4 = 1;
    ECap1Regs.ECCTL1.bit.CAPLDEN = 1;//允许装载
    ECap1Regs.ECCTL2.bit.CAP_APWM = 0;//用于捕获
    ECap1Regs.ECCTL2.bit.CONT_ONESHT = 0;//连续
    ECap1Regs.ECCTL2.bit.SYNCI_EN = 0;
    ECap1Regs.ECCTL2.bit.SYNCO_SEL = 0;
    ECap1Regs.ECCTL2.bit.TSCTRSTOP = 1;//自由运行

    ECap1Regs.ECEINT.bit.CEVT4 = 1;//设置中断


}


void InitPWM4(void){
    int PWMPRD,DeadTime;//周期和死区时间
    EALLOW;
    SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0;//设置epwm同步暂停
    EDIS;
    PWMPRD = 3750;
    DeadTime = 180;
    EPwm4Regs.TBPRD = PWMPRD;
    EPwm4Regs.TBPHS.half.TBPHS = 0;//忽略同步事件
    EPwm4Regs.TBCTL.bit.CLKDIV = 0;//预分频为1
    EPwm4Regs.TBCTL.bit.HSPCLKDIV = 0;
    EPwm4Regs.TBCTL.bit.PRDLD = 0;//使用映射寄存器
    EPwm4Regs.TBCTL.bit.CTRMODE = 2;//对称模式
    EPwm4Regs.TBCTL.bit.PHSEN = 0;//不从时基相位寄存器进行装载
    EPwm4Regs.TBCTL.bit.SYNCOSEL = 1;//
```

```
    EPwm4Regs.CMPCTL.bit.SHDWAMODE = 0;//使用映射寄存器
    EPwm4Regs.CMPCTL.bit.SHDWBMODE = 0;
    EPwm4Regs.CMPCTL.bit.LOADAMODE = 2;//计数器=0或T时候装载
    EPwm4Regs.CMPCTL.bit.LOADBMODE = 2;
    EPwm4Regs.CMPA.half.CMPA = (PWMPRD / 2);//脉冲占比50%
    EPwm4Regs.CMPB = (PWMPRD / 2);

    EPwm4Regs.AQCTLA.bit.CAU = 2;//配置动作
    EPwm4Regs.AQCTLA.bit.CAD = 1;
    EPwm4Regs.AQCTLA.bit.CBU = 0;
    EPwm4Regs.AQCTLA.bit.CBD = 0;

    EPwm4Regs.DBCTL.bit.HALFCYCLE = 0;//全周期
    EPwm4Regs.DBCTL.bit.IN_MODE = 0;//A是源
    EPwm4Regs.DBCTL.bit.POLSEL = 2;
    EPwm4Regs.DBCTL.bit.OUT_MODE = 3;
    EPwm4Regs.DBFED = DeadTime;
    EPwm4Regs.DBRED = DeadTime;

    EALLOW;
    SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;//设置epwm同步暂停
    EDIS;

}

void Gpio_Init()
{
    EALLOW;
    GpioCtrlRegs.GPADIR.bit.GPIO6 = 1; //设置为输出
    GpioCtrlRegs.GPADIR.bit.GPIO7 = 1;
    GpioCtrlRegs.GPAPUD.bit.GPIO5 = 0;//上拉
    GpioCtrlRegs.GPAPUD.bit.GPIO6 = 1;
    GpioCtrlRegs.GPAPUD.bit.GPIO7 = 1;
    GpioCtrlRegs.GPAMUX1.bit.GPIO5 = 3;//设置外设功能
    GpioCtrlRegs.GPAMUX1.bit.GPIO6 = 1;
    GpioCtrlRegs.GPAMUX1.bit.GPIO7 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO5 =1;//初始化数值
    GpioDataRegs.GPACLEAR.bit.GPIO6 =1;
    GpioDataRegs.GPACLEAR.bit.GPIO7 =1;

    EDIS;

}

void main(void)
{
      InitSysCtrl();    //初始化系统时钟，选择内部晶振1，10MHZ，12倍频，2分频，初始化外设时
钟，低速外设,4分频
      DINT;                 //关总中断
      IER = 0x0000;         //关CPU中断使能
      IFR = 0x0000;     //清CPU中断标志
      InitPieCtrl();    //关pie中断
      InitPieVectTable();   //清中断向量表

      EALLOW;               //配置中断向量表
      PieVectTable.ECAP1_INT = &Ecap1Int_isr;
      PieCtrlRegs.PIEIER4.bit.INTx1 = 1;
      IER |= 8;
```

```
    EDIS;


   //  MemCopy(&RamfuncsLoadStart, &RamfuncsLoadEnd, &RamfuncsRunStart);
    InitFlash();
    Gpio_Init();
    InitPWM4();
    InitCap1();
    TM1638_Init();        //初始化LED

    EINT;//中断总使能
    ERTM;


    while(1)
    {
      DelaymS(2000);
      LED_Show(1,(DutyOnTime % 10),0);
      LED_Show(2,((DutyOnTime / 10) % 10),0);
      LED_Show(3,((DutyOnTime / 100) % 10),1);
      LED_Show(4,((DutyOnTime / 1000) ),0);
      LED_Show(5,(Period % 10),0);
      LED_Show(6,((Period / 10) % 10),0);
      LED_Show(7,((Period / 100) % 10),1);
      LED_Show(8,((Period / 1000) ),0);
      DelaymS(2000);
    }
}

interrupt void Ecap1Int_isr(void)
{
    t1 =  ECap1Regs.CAP2;
    t2 =  ECap1Regs.CAP3;
    t3 =  ECap1Regs.CAP4;
    t4 =  ECap1Regs.CAP1;
    Period  =  t1 + t2;
    DutyOnTime = t1;
    DutyOffTime = t2;
    rate = DutyOnTime/DutyOffTime;
    ECap1Regs.ECCLR.bit.CEVT4 = 1;
    ECap1Regs.ECCLR.bit.INT = 1;
    PieCtrlRegs.PIEACK.all = 8;
}
```

2.6 实验总结

该实验的关键在于对PWM、eCAP的初始化配置。最终输出的周期差了2，在排除了只进了一次eCAP中断的可能后，确定了是由于硬件的问题导致的误差。

3 ADC实验

3.1 实验要求

(1)通过ADC读取滑动变阻器(目标板上AD1)中间抽头上的电压值，并将结果在数码管上

显示。

(2)通过ADC读取目标板上NTC1两端的电压差值，并将结果在数码管上显示。

3.2 实验目的

(1) 掌握 ADC 的配置和操作方法；

(2) 掌握使用 PWM 触发 AD 采样的方法。

3.3 实验说明

本实验在前面实验的基础上，添加了 ADC 的使用以及 ADC 中断的配置。通过 PWM 触发 AD 采样（也可尝试其他触发 AD 采样的方法，多方面了解和掌握 ADC 的操作），并将采样结果换算成实际的电压结果，在数码管上显示出来。

数码管上显示 AD 采样结果的时候，需要将数据的每一位输出到数码管的相应的位置上，还涉及到小数点位置的显示，因此，建议编写数码管上层驱动程序，实现浮点数的显示功能。

本实验中，ADC 的转换触发采用 PWM 控制，复习了上一实验中对 PWM 的配置。

3.4 实验思路

该实验主要实现一个功能：

1、配置ADC对模拟电压进行转换并显示至LCD模块。

该实验的大致思路如下：需要设置PWM产生SOC事件触发ADC转换，并在ADC转换完成后触发中断，在中断服务程序中计算模拟电压并显示至LCD模块。

3.5 实验代码

实现的代码如下：

```
#include "DSP28x_Project.h"
#include "LED_TM1638.h"
#include "DSP2802x_Adc.h"

#define ADC_usDELAY  1000L

interrupt void MyAdcInt1_isr(void);   //ADCINT1
void Init_Adc(void);
void InitPWM4(void);
void DelaymS(int tm);

//初始化变量
int High = 0;int Low = 0;int Derta = 0;
float high = 0; float low = 0;float derta = 0;
int ADCSampT = 12;

void Init_Adc(void)
```

```
{
    extern void DSP28x_usDelay(Uint32 Count);

    EALLOW;
    SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 1;
    (*Device_cal)();
    EDIS;
    DELAY_US(ADC_usDELAY);

    EALLOW;
    AdcRegs.ADCCTL1.bit.ADCBGPWD  = 1;      // Power ADC BG
    AdcRegs.ADCCTL1.bit.ADCREFPWD = 1;      // Power reference
    AdcRegs.ADCCTL1.bit.ADCPWDN   = 1;      // Power ADC
    AdcRegs.ADCCTL1.bit.ADCENABLE = 1;      // Enable ADC
    AdcRegs.ADCCTL1.bit.ADCREFSEL = 0;      // Select interal BG使用内部参考电压
    EDIS;
    DELAY_US(ADC_usDELAY);
    AdcOffsetSelfCal();//自矫正
    DELAY_US(ADC_usDELAY);
    EALLOW;
    AdcRegs.ADCCTL1.bit.INTPULSEPOS = 1;

    AdcRegs.INTSEL1N2.bit.INT1CONT = 0;//不连续中断
    AdcRegs.INTSEL1N2.bit.INT1E = 1;//中断使能
    AdcRegs.INTSEL1N2.bit.INT1SEL = 7;//7次转换后结束

    AdcRegs.ADCSAMPLEMODE.all = 0;//顺序采样
    AdcRegs.ADCSOC0CTL.bit.TRIGSEL = 11;
    AdcRegs.ADCSOC1CTL.bit.TRIGSEL = 11;
    AdcRegs.ADCSOC2CTL.bit.TRIGSEL = 11;
    AdcRegs.ADCSOC3CTL.bit.TRIGSEL = 11;
    AdcRegs.ADCSOC4CTL.bit.TRIGSEL = 11;
    AdcRegs.ADCSOC5CTL.bit.TRIGSEL = 11;
    AdcRegs.ADCSOC6CTL.bit.TRIGSEL = 11;
    AdcRegs.ADCSOC7CTL.bit.TRIGSEL = 11;
    AdcRegs.ADCSOC0CTL.bit.CHSEL = 1;//前四个通道实现对ADA1过采样
    AdcRegs.ADCSOC1CTL.bit.CHSEL = 1;
    AdcRegs.ADCSOC2CTL.bit.CHSEL = 1;
    AdcRegs.ADCSOC3CTL.bit.CHSEL = 1;
    AdcRegs.ADCSOC4CTL.bit.CHSEL = 9;//后四个通道实现对ADB1过采样
    AdcRegs.ADCSOC5CTL.bit.CHSEL = 9;
    AdcRegs.ADCSOC6CTL.bit.CHSEL = 9;
    AdcRegs.ADCSOC7CTL.bit.CHSEL = 9;
    AdcRegs.ADCSOC0CTL.bit.ACQPS = ADCSampT;//设置采样周期
    AdcRegs.ADCSOC1CTL.bit.ACQPS = ADCSampT;
    AdcRegs.ADCSOC2CTL.bit.ACQPS = ADCSampT;
    AdcRegs.ADCSOC3CTL.bit.ACQPS = ADCSampT;
    AdcRegs.ADCSOC4CTL.bit.ACQPS = ADCSampT;
    AdcRegs.ADCSOC5CTL.bit.ACQPS = ADCSampT;
    AdcRegs.ADCSOC6CTL.bit.ACQPS = ADCSampT;
    AdcRegs.ADCSOC7CTL.bit.ACQPS = ADCSampT;
    EDIS;
}

void InitPWM4(void){
    int PWMPRD,DeadTime;//周期和死区时间
    EALLOW;
    SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0;//设置epwm同步暂停
```

```
    EDIS;
    PWMPRD = 3000;
    DeadTime = 120;
    EPwm4Regs.TBPRD = PWMPRD;
    EPwm4Regs.TBPHS.half.TBPHS = 0;//忽略同步事件
    EPwm4Regs.TBCTL.bit.CLKDIV = 0;//预分频为1
    EPwm4Regs.TBCTL.bit.HSPCLKDIV = 0;
    EPwm4Regs.TBCTL.bit.PRDLD = 0;//使用映射寄存器
    EPwm4Regs.TBCTL.bit.CTRMODE = 2;//对称模式
    EPwm4Regs.TBCTL.bit.PHSEN = 0;//不从时基相位寄存器进行装载
    EPwm4Regs.TBCTL.bit.SYNCOSEL = 1;//

    EPwm4Regs.CMPCTL.bit.SHDWAMODE = 0;//使用映射寄存器
    EPwm4Regs.CMPCTL.bit.SHDWBMODE = 0;
    EPwm4Regs.CMPCTL.bit.LOADAMODE = 2;//计数器=0或T时候装载
    EPwm4Regs.CMPCTL.bit.LOADBMODE = 2;
    EPwm4Regs.CMPA.half.CMPA = (PWMPRD / 2);//脉冲占比50%
    EPwm4Regs.CMPB = (PWMPRD / 2);

    EPwm4Regs.AQCTLA.bit.CAU = 2;//配置动作
    EPwm4Regs.AQCTLA.bit.CAD = 1;
    EPwm4Regs.AQCTLA.bit.CBU = 0;
    EPwm4Regs.AQCTLA.bit.CBD = 0;

    EPwm4Regs.DBCTL.bit.HALFCYCLE = 0;//全周期
    EPwm4Regs.DBCTL.bit.IN_MODE = 0;//A是源
    EPwm4Regs.DBCTL.bit.POLSEL = 2;
    EPwm4Regs.DBCTL.bit.OUT_MODE = 3;
    EPwm4Regs.DBFED = DeadTime;
    EPwm4Regs.DBRED = DeadTime;

    EPwm4Regs.ETSEL.bit.SOCAEN = 1;
    EPwm4Regs.ETSEL.bit.SOCASEL = 1;
    EPwm4Regs.ETPS.bit.SOCAPRD = 1;
    EPwm4Regs.ETCLR.bit.SOCA = 1;
    EALLOW;
    SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;//设置epwm同步暂停
    EDIS;

}


void DelaymS(int tm)
{
  int i;
  unsigned int j;
    for(i = 0;i < tm ;i++){
      j =  60000;
      while(j != 0)j--;
      }
}
void main(void)
{
      High = AdcResult.ADCRESULT0 + AdcResult.ADCRESULT1;
      High += AdcResult.ADCRESULT2;
      InitSysCtrl();    //初始化系统时钟，选择内部晶振1，10MHZ，12倍频，2分频，初始化外设时钟，低速外设,4分频
      DINT;                 //关总中断
      IER = 0x0000;         //关CPU中断使能
```

```
        IFR = 0x0000;      //清CPU中断标志
        InitPieCtrl();     //关pie中断
        InitPieVectTable();    //清中断向量表
        EALLOW;                //配置中断向量表
        PieVectTable.rsvd10_1 = &MyAdcInt1_isr;
        PieCtrlRegs.PIEIER10.bit.INTx1 = 1;
        IER |= 0x0200;
        EDIS;


       //  MemCopy(&RamfuncsLoadStart, &RamfuncsLoadEnd, &RamfuncsRunStart);
        InitFlash();
        TM1638_Init();   //初始化LED
        Init_Adc();
        InitPWM4();

        EINT;//中断总使能
        ERTM;


        while(1)
        {
            {
//                LED_Show(1,(int)(high*100)%10,0);
//                LED_Show(2,(int)(high*10)%10,0);
//                LED_Show(3,(int)(high)%10,0);
//
//                LED_Show(4,(int)(low*100)%10,0);
//                LED_Show(5,(int)(low*10)%10,0);
//                LED_Show(6,(int)(low)%10,0);
//
//                LED_Show(7,(int)(derta*10) % 10,0);
//                LED_Show(8,(int)(derta) % 10,0);

            }

        }

}



interrupt void MyAdcInt1_isr(void) {

    High = AdcResult.ADCRESULT0 + AdcResult.ADCRESULT1;
    High += AdcResult.ADCRESULT2;
  //  High += AdcResult.ADCRESULT2 + AdcResult.ADCRESULT3; 为什么result3是0？？？
    Low = AdcResult.ADCRESULT4 + AdcResult.ADCRESULT5;
    Low += AdcResult.ADCRESULT6;
    Derta = High - Low;
    high = High/(3*4096*1.0) * 3.30;
    low = Low/(3*4096*1.0) * 3.30;
    derta = Derta/(3*4096*1.0) * 3.30;
    if (derta < 0){
        derta = -derta;
    }
    AdcRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP10;//! ! !
```

```
    EPwm4Regs.ETCLR.bit.SOCA = 1;
    LED_Show(1,(int)(high*100)%10,0);//high即为0.037
    LED_Show(2,(int)(high*10)%10,0);
    LED_Show(3,(int)(high*1)%10,0);

    LED_Show(4,(int)(low*100)%10,0);
    LED_Show(5,(int)(low*10)%10,0);
    LED_Show(6,(int)(low)%10,0);

    LED_Show(7,(int)(derta*10) % 10,0);
    LED_Show(8,(int)(derta*1) % 10,0);
    DelaymS(300);

}
```

3.6 实验调试

该实验主要遇见了俩个问题：

1、中断的PIE向量表设置错误，使用的是第十组的PIE向量但是设置PIEACK和PIE使能设置错误，中断程序地址赋值错误。修改后的部分代码如下：

```
PieVectTable.rsvd10_1 = &MyAdcInt1_isr;
PieCtrlRegs.PIEIER10.bit.INTx1 = 1;
PieCtrlRegs.PIEACK.all = PIEACK_GROUP10;//! ! !
```

2、在模拟电压的小数点输出由于数据类型的转换没有考虑全面浮点数和整数之间的运算转换关系导致最初输出显示错误，修改后的代码如下：

```
LED_Show(1,(int)(high*100)%10,0);//high即为0.037
LED_Show(2,(int)(high*10)%10,0);
LED_Show(3,(int)(high*1)%10,0);

LED_Show(4,(int)(low*100)%10,0);
LED_Show(5,(int)(low*10)%10,0);
LED_Show(6,(int)(low)%10,0);

LED_Show(7,(int)(derta*10) % 10,0);
LED_Show(8,(int)(derta*1) % 10,0);
DelaymS(300);
```

3、在最初程序修改后，测出来的模拟电压都是零漂值，最后发现是硬件问题，无法产生正确的模拟电压。

4 综合实验

4.1 实验要求

(1) 通过 ADC 获取目标板上滑动变阻器 AD1 中间抽头的电压，并实时显示在上排数码管上；从目标板的 J33 的第一脚引到 DSP 的模拟输入端。

(2) 将（1）中得到的采样结果转换成 PWM 的占空比输出到目标板上 J50，将 PWM 信号低通滤波得到与 PWM 占空比成正比关系的电压，相当于 DAC。目标板的 J52 的第一脚为 DAC 的输出电压值。

(3) 通过另一路 ADC 对（2）中 DAC 得到的电压进行采样，并实时显示在下排数码管上。可以 J52 上的 DAC1 信号直接接到 DSP 的模拟输入端；也可以将 J52 的引脚短接，从目标板 J33 的第四脚引到 DSP 的模拟输入端。

(4) 通过对采样电阻大小的选取和 PWM 占空比的控制，使得两次采样的结果尽可能相等。可以使用闭环控制方法，将第二次的采样结果作为反馈量，第一次的采样结果作为给定参考量，PWM 的占空比作为通过 PI 控制等控制方法使反馈量跟踪到给定量，也可以设计其他控制策略。

4.2 实验目的

本实验综合所学的 DSP 的知识和前面做过的实验，设计一个涉及到多个外设的嵌入式系统，要求能够熟练配置系统时钟和各个外设。

考察对芯片多个外设的配置的掌握情况，考察学生的综合设计和嵌入式开发的能力。

4.3 实验思路

该实验主要实现三个功能：

1、配置PWM产生事件触发ADC对输入的模拟信号进行AD转换。

2、根据转换的结果配置PWM产生对应占空比的模拟信号。

3、ADC对原输入信号和PWM产生的模拟信号进行转换并且根据差值在中断程序中调整PWM的占空比，利用PI调节实现俩个信号的一致。

该实验的大致思路如下：要设置PWM产生SOC事件触发ADC转换并且根据转换结果输出对应占空比的信号从而生称模拟信号，并在ADC转换完成后触发中断，在中断服务程序进行PI调节并将原模拟和新信号输出至LCD模块。

4.4 实验代码

实现的代码如下：

```
//homework4
#include "DSP28x_Project.h"
#include "LED_TM1638.h"
#include "DSP2802x_Adc.h"

#define ADC_usDELAY  1000L
```

```
interrupt void MyAdcInt1_isr(void);   //ADCINT1
void Init_Adc(void);
void InitPWM4(void);
void DelaymS(int tm);

//初始化变量
int Target = 0;int Now = 0;
float target = 0; float now = 0;
int ADCSampT = 12;
int Derta = 0;//记录偏差
int CMPA;

void Init_Adc(void)
{

    extern void DSP28x_usDelay(Uint32 Count);

    EALLOW;
    SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 1;
    (*Device_cal)();
    EDIS;
    DELAY_US(ADC_usDELAY);

    EALLOW;
    AdcRegs.ADCCTL1.bit.ADCBGPWD  = 1;      // Power ADC BG
    AdcRegs.ADCCTL1.bit.ADCREFPWD = 1;      // Power reference
    AdcRegs.ADCCTL1.bit.ADCPWDN   = 1;      // Power ADC
    AdcRegs.ADCCTL1.bit.ADCENABLE = 1;      // Enable ADC
    AdcRegs.ADCCTL1.bit.ADCREFSEL = 0;      // Select interal BG使用内部参考电压
    EDIS;
    DELAY_US(ADC_usDELAY);
    AdcOffsetSelfCal();//自矫正
    DELAY_US(ADC_usDELAY);
    EALLOW;
    AdcRegs.ADCCTL1.bit.INTPULSEPOS = 1;

    AdcRegs.INTSEL1N2.bit.INT1CONT = 0;//不连续中断
    AdcRegs.INTSEL1N2.bit.INT1E = 1;//中断使能
    AdcRegs.INTSEL1N2.bit.INT1SEL = 7;//7次转换后结束

    AdcRegs.ADCSAMPLEMODE.all = 0;//顺序采样
    AdcRegs.ADCSOC0CTL.bit.TRIGSEL = 11;
    AdcRegs.ADCSOC1CTL.bit.TRIGSEL = 11;
    AdcRegs.ADCSOC2CTL.bit.TRIGSEL = 11;
    AdcRegs.ADCSOC3CTL.bit.TRIGSEL = 11;
    AdcRegs.ADCSOC4CTL.bit.TRIGSEL = 11;
    AdcRegs.ADCSOC5CTL.bit.TRIGSEL = 11;
    AdcRegs.ADCSOC6CTL.bit.TRIGSEL = 11;
    AdcRegs.ADCSOC7CTL.bit.TRIGSEL = 11;
    AdcRegs.ADCSOC0CTL.bit.CHSEL = 1;//前四个通道实现对ADA1过采样
    AdcRegs.ADCSOC1CTL.bit.CHSEL = 1;
    AdcRegs.ADCSOC2CTL.bit.CHSEL = 1;
    AdcRegs.ADCSOC3CTL.bit.CHSEL = 1;
    AdcRegs.ADCSOC4CTL.bit.CHSEL = 9;//后四个通道实现对ADB1过采样
    AdcRegs.ADCSOC5CTL.bit.CHSEL = 9;
    AdcRegs.ADCSOC6CTL.bit.CHSEL = 9;
    AdcRegs.ADCSOC7CTL.bit.CHSEL = 9;
    AdcRegs.ADCSOC0CTL.bit.ACQPS = ADCSampT;//设置采样周期
    AdcRegs.ADCSOC1CTL.bit.ACQPS = ADCSampT;
```

```
    AdcRegs.ADCSOC2CTL.bit.ACQPS = ADCSampT;
    AdcRegs.ADCSOC3CTL.bit.ACQPS = ADCSampT;
    AdcRegs.ADCSOC4CTL.bit.ACQPS = ADCSampT;
    AdcRegs.ADCSOC5CTL.bit.ACQPS = ADCSampT;
    AdcRegs.ADCSOC6CTL.bit.ACQPS = ADCSampT;
    AdcRegs.ADCSOC7CTL.bit.ACQPS = ADCSampT;
    EDIS;
}

//Epwm4产生ADC触发信号
void InitPWM4(void){
    int PWMPRD,DeadTime;//周期和死区时间
    EALLOW;
    SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0;//设置epwm同步暂停
    EDIS;
    PWMPRD = 3000;
    DeadTime = 120;
    EPwm4Regs.TBPRD = PWMPRD;
    EPwm4Regs.TBPHS.half.TBPHS = 0;//忽略同步事件
    EPwm4Regs.TBCTL.bit.CLKDIV = 0;//预分频为1
    EPwm4Regs.TBCTL.bit.HSPCLKDIV = 0;
    EPwm4Regs.TBCTL.bit.PRDLD = 0;//使用映射寄存器
    EPwm4Regs.TBCTL.bit.CTRMODE = 2;//对称模式
    EPwm4Regs.TBCTL.bit.PHSEN = 0;//不从时基相位寄存器进行装载
    EPwm4Regs.TBCTL.bit.SYNCOSEL = 1;//

    EPwm4Regs.CMPCTL.bit.SHDWAMODE = 0;//使用映射寄存器
    EPwm4Regs.CMPCTL.bit.SHDWBMODE = 0;
    EPwm4Regs.CMPCTL.bit.LOADAMODE = 2;//计数器=0或T时候装载
    EPwm4Regs.CMPCTL.bit.LOADBMODE = 2;
    EPwm4Regs.CMPA.half.CMPA = (PWMPRD / 2);//脉冲占比50%
    EPwm4Regs.CMPB = (PWMPRD / 2);

    EPwm4Regs.AQCTLA.bit.CAU = 2;//配置动作
    EPwm4Regs.AQCTLA.bit.CAD = 1;
    EPwm4Regs.AQCTLA.bit.CBU = 0;
    EPwm4Regs.AQCTLA.bit.CBD = 0;

    EPwm4Regs.DBCTL.bit.HALFCYCLE = 0;//全周期
    EPwm4Regs.DBCTL.bit.IN_MODE = 0;//A是源
    EPwm4Regs.DBCTL.bit.POLSEL = 2;
    EPwm4Regs.DBCTL.bit.OUT_MODE = 3;
    EPwm4Regs.DBFED = DeadTime;
    EPwm4Regs.DBRED = DeadTime;

    EPwm4Regs.ETSEL.bit.SOCAEN = 1;
    EPwm4Regs.ETSEL.bit.SOCASEL = 1;
    EPwm4Regs.ETPS.bit.SOCAPRD = 1;
    EPwm4Regs.ETCLR.bit.SOCA = 1;
    EALLOW;
    SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;//设置epwm同步暂停
    EDIS;

}



//Epwm1产生DAC 生成和target差不多的模拟信号   并且
void InitPWM1(void){
```

```
    //配置PWM输出 GPIO0
    EALLOW;
    GpioCtrlRegs.GPADIR.bit.GPIO0 = 1; //设置为输出
    GpioCtrlRegs.GPAPUD.bit.GPIO0 = 1;//上拉
    GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 1;//设置外设功能
    GpioDataRegs.GPACLEAR.bit.GPIO0 =1;//初始化数值
    EDIS;

    int PWMPRD;//周期和死区时间
    EALLOW;
    SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0;//设置epwm同步暂停
    EDIS;
    PWMPRD = 3000;
    EPwm1Regs.TBPRD = PWMPRD;
    EPwm1Regs.TBPHS.half.TBPHS = 0;//忽略同步事件
    EPwm1Regs.TBCTL.bit.CLKDIV = 0;//预分频为1
    EPwm1Regs.TBCTL.bit.HSPCLKDIV = 0;
    EPwm1Regs.TBCTL.bit.PRDLD = 0;//使用映射寄存器
    EPwm1Regs.TBCTL.bit.CTRMODE = 2;//对称模式
    EPwm1Regs.TBCTL.bit.PHSEN = 0;//不从时基相位寄存器进行装载
    EPwm1Regs.TBCTL.bit.SYNCOSEL = 1;//

    EPwm1Regs.CMPCTL.bit.SHDWAMODE = 0;//使用映射寄存器
    EPwm1Regs.CMPCTL.bit.SHDWBMODE = 0;
    EPwm1Regs.CMPCTL.bit.LOADAMODE = 2;//计数器=0或T时候装载
    EPwm1Regs.CMPCTL.bit.LOADBMODE = 2;
    EPwm1Regs.CMPA.half.CMPA = (int)((1-(target/3.3))*3000);//脉冲占比50%
    EPwm1Regs.CMPB = (int)((1-(target/3.3))*3000);;

    EPwm1Regs.AQCTLA.bit.CAU = 1;//配置动作
    EPwm1Regs.AQCTLA.bit.CAD = 2;
    EPwm1Regs.AQCTLA.bit.CBU = 0;
    EPwm1Regs.AQCTLA.bit.CBD = 0;

    EPwm1Regs.DBCTL.bit.HALFCYCLE = 0;//全周期
    EPwm1Regs.DBCTL.bit.IN_MODE = 0;//A是源
    EPwm1Regs.DBCTL.bit.POLSEL = 0;
    EPwm1Regs.DBCTL.bit.OUT_MODE = 0;//直接生成需要的模拟信号

//    EPwm1Regs.ETSEL.bit.INTEN = 1;
//    EPwm1Regs.ETSEL.bit.INTSEL = 1;
//    EPwm1Regs.ETPS.bit.INTPRD = 1;
//    EPwm1Regs.ETCLR.bit.INT = 1;
    EALLOW;
    SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;//设置epwm同步暂停
    EDIS;

}

void DelaymS(int tm)
{
  int i;
  unsigned int j;
    for(i = 0;i < tm ;i++){
      j =  60000;
      while(j != 0)j--;
      }
```

```
}
void main(void)
{
      InitSysCtrl();    //初始化系统时钟，选择内部晶振1，10MHZ，12倍频，2分频，初始化外设时钟，低速外设,4分频
      DINT;                 //关总中断
      IER = 0x0000;         //关CPU中断使能
      IFR = 0x0000;     //清CPU中断标志
      InitPieCtrl();    //关pie中断
      InitPieVectTable();   //清中断向量表
      EALLOW;               //配置中断向量表
      PieVectTable.rsvd10_1 = &MyAdcInt1_isr;
      PieCtrlRegs.PIEIER10.bit.INTx1 = 1;
      IER |= 0x0200;
      EDIS;


     //  MemCopy(&RamfuncsLoadStart, &RamfuncsLoadEnd, &RamfuncsRunStart);
      InitFlash();
      TM1638_Init();   //初始化LED
      Init_Adc();
      InitPWM4();
      InitPWM1();

      EINT;//中断总使能
      ERTM;


      while(1)
      {
          {
//              LED_Show(1,(int)(high*100)%10,0);
//              LED_Show(2,(int)(high*10)%10,0);
//              LED_Show(3,(int)(high)%10,0);
//
//              LED_Show(4,(int)(low*100)%10,0);
//              LED_Show(5,(int)(low*10)%10,0);
//              LED_Show(6,(int)(low)%10,0);
//
//              LED_Show(7,(int)(derta*10) % 10,0);
//              LED_Show(8,(int)(derta) % 10,0);

          }

      }

}



interrupt void MyAdcInt1_isr(void) {

    Target = AdcResult.ADCRESULT0 + AdcResult.ADCRESULT1;
    Target += AdcResult.ADCRESULT2 + AdcResult.ADCRESULT3;
    target = Target/(4*4096*1.0) * 3.30;

    Now =  AdcResult.ADCRESULT4 + AdcResult.ADCRESULT5;
    Now +=  AdcResult.ADCRESULT6 + AdcResult.ADCRESULT7;
```

```
    now = Now/(4*4096*1.0) * 3.30;

//    EALLOW;
//    SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0;//设置epwm同步暂停
//    EDIS;
      Derta +=  (int)((((target - now) /3.3))*3000);
      CMPA += ((int)(0*Derta) + (int)((((target - now) /3.3))*3000));
      if (CMPA < 0){CMPA = 0;}
      else if  (CMPA > 3000){CMPA = 3000;}
      EPwm1Regs.CMPA.half.CMPA = CMPA;//根据差值进行改变


    //EPwm1Regs.CMPA.half.CMPA += (int)((((target - now) /3.3))*3000);//根据差值进行改变
//    EALLOW;
//    SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;//设置epwm同步暂停
//    EDIS;

    AdcRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP10;//! ! !

    EPwm4Regs.ETCLR.bit.SOCA = 1;
    LED_Show(1,(int)(target*1000)%10,0);//
    LED_Show(2,(int)(target*100)%10,0);
    LED_Show(3,(int)(target*10)%10,0);
    LED_Show(4,(int)(target)%10,0);

    LED_Show(5,(int)(now*1000)%10,0);//
    LED_Show(6,(int)(now*100)%10,0);
    LED_Show(7,(int)(now*10)%10,0);
    LED_Show(8,(int)(now)%10,0);

    DelaymS(200);

}
```

4.5 实验调试

该实验主要问题是最初信号追不上，总是有0.2左右的误差，并且原模拟信号也一直在改变，后来经过检查发现是杜邦线和硬件插孔之间的接触问题，如果按紧则可以很快的实现稳定的信号追踪。