实验1

陈鸿煜

华中科技大学 自动控制系

实验1主要是对时钟、GPIO和中断的操作，这一节我们要掌握这三个重要的技能。

以下是实验一的代码，首先我会从代码中找到对应的知识点，简要谈一下各个功能的实现方式，然后结合代码进行深度分析。

```
/*
 * main.c
 */
#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

void InitPWM4Gpio();
void InitPWM4();
void InitCAPGpio();
void InitCAP();
void InitADC();

void HorseRunning(int16 no, int Running);
#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


int hourH = 0,hourL=0,minH=0,minL=0,secH=0,secL=0,TenmS = 0;
int Running = 0,NewLedEn = 0,KeyDLTime = 0;
int LedFlashCtr;
int period,hightime;
int PWM1Prd;
int ledtmp;
int adcptr;
unsigned int ADC_GD,ADC_FK;
float ADC_GDF,ADC_FKF;
long int li1,li2,li3,li4,PWM_HI,PWM_LO,PWM_PRD;

int PWMIntNo,PWMDuty,Tridir;
int ledkd,leddat;

void  Xint1_Init()
{
```

```
    EALLOW;
    GpioCtrlRegs.GPAMUX1.bit.GPIO12 = 0;
    GpioCtrlRegs.GPADIR.bit.GPIO12 = 0;
    GpioCtrlRegs.GPAPUD.bit.GPIO12 = 0;      // Pullup's enabled GPIO4
    GpioIntRegs.GPIOXINT1SEL.bit.GPIOSEL = 12;
    XIntruptRegs.XINT1CR.bit.POLARITY = 0;
    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 main(void) {

    int x,y;
      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;
    GpioDataRegs.GPACLEAR.bit.GPIO0 = 1;
    GpioDataRegs.GPACLEAR.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;

      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;
    EDIS;

  // MemCopy(&RamfuncsLoadStart, &RamfuncsLoadEnd, &RamfuncsRunStart);
    InitFlash();

    InitCpuTimers();      // 初始化定时器
    ConfigCpuTimer(&CpuTimer0, 60,10000);
    CpuTimer0Regs.TCR.bit.TSS = 0;
    CpuTimer0Regs.TCR.bit.TRB = 1;
    CpuTimer0.InterruptCount = 0;

    HorseIO_Init();
    Xint1_Init();
    TM1638_Init();        //初始化LED

    InitPWM4Gpio();
    InitPWM4();
    InitCAPGpio();
    InitCAP();
    InitADC();

    PieCtrlRegs.PIECTRL.bit.ENPIE = 1;      // Enable the PIE block
    PieCtrlRegs.PIEIER1.bit.INTx7 = 1;      // TINT0
    PieCtrlRegs.PIEIER1.bit.INTx4 = 1;      // XINT1
    PieCtrlRegs.PIEIER1.bit.INTx1 = 1;      // ADCINT1
    PieCtrlRegs.PIEIER3.bit.INTx4 = 1;      // EPWM4
    PieCtrlRegs.PIEIER4.bit.INTx1 = 1;      // ECAP1

    IER |= M_INT1;      /**使能CPU中断**/
    IER |= M_INT3;      /**使能CPU中断**/
    IER |= M_INT4;      /**使能CPU中断**/


    EINT;
    // ERTM;

    ledkd=0;
  while(1){
    if(NewLedEn==0) {
      if(ledkd==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);
        }
      else if(ledkd==1){
        leddat=PWM_HI;
        LED_Show(4,leddat /1000,0);
        leddat = leddat % 1000;
        LED_Show(3,leddat /100,0);
        leddat = leddat % 100;
        LED_Show(2,leddat /10,0);
```

```
            LED_Show(1,(leddat % 10),0);

            leddat=PWM_PRD;
            LED_Show(8,leddat /1000,0);
            leddat = leddat % 1000;
            LED_Show(7,leddat /100,0);
            leddat = leddat % 100;
            LED_Show(6,leddat /10,0);
            LED_Show(5,(leddat % 10),1);
            }

          else if(ledkd==2){
            leddat=ADC_FKF;
            LED_Show(4,leddat /1000,0);
            leddat = leddat % 1000;
            LED_Show(3,leddat /100,1);
            leddat = leddat % 100;
            LED_Show(2,leddat /10,0);
            LED_Show(1,(leddat % 10),0);

            leddat=ADC_GDF;
            LED_Show(8,leddat /1000,0);
            leddat = leddat % 1000;
            LED_Show(7,leddat /100,1);
            leddat = leddat % 100;
            LED_Show(6,leddat /10,0);
            LED_Show(5,(leddat % 10),1);
            }

          NewLedEn = 1;
          }
        }
}

void HorseRunning(int16 no, int Running)
{
    if(Running == 0) {
        if(no & 0x1)Led0Blink();
        else Led0Blank();
        if(no & 0x2)Led1Blink();
        else Led1Blank();
        if(no & 0x4)Led2Blink();
        else Led2Blank();
        if(no & 0x8)Led3Blink();
        else Led3Blank();
    } else if(Running == 1) {
        if(no & 0x1)Led3Blink();
        else Led3Blank();
        if(no & 0x2)Led2Blink();
        else Led2Blank();
        if(no & 0x4)Led1Blink();
        else Led1Blank();
        if(no & 0x8)Led0Blink();
        else Led0Blank();
    } else if(Running == 2) {
        if(no & 0x1)Led0Blink();
        else Led0Blank();
        if(no & 0x2)Led2Blink();
```

```
        else Led2Blank();
        if(no & 0x4)Led1Blink();
        else Led1Blank();
        if(no & 0x8)Led3Blink();
        else Led3Blank();
    }

}


interrupt void myXint1_isr(void)
{
    if((Running == 0)&&(KeyDLTime > 20)){
      EALLOW;
      CpuTimer0Regs.TCR.bit.TSS = 1;
      CpuTimer0Regs.TCR.bit.TRB = 1;
      CpuTimer0Regs.TCR.bit.TSS = 0;
      EDIS;
      Running = 1;
      KeyDLTime = 0;
      }
    else if((Running == 1)&&(KeyDLTime > 20)){
      Running = 2;
      KeyDLTime = 0;
      }
    else if((Running == 2)&&(KeyDLTime > 20)){
      Running = 0;
      hourH = 0;hourL=0;minH=0;minL=0;secH=0;secL=0;TenmS = 0;
      KeyDLTime = 0;
      EALLOW;
      // CpuTimer0Regs.TCR.bit.TSS = 1;
      // CpuTimer0Regs.TCR.bit.TRB = 1;
      EDIS;
      }
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}

interrupt void cpu_timer0_isr(void) {

    KeyDLTime++;

    LedFlashCtr++;
    if((LedFlashCtr % 10)==0)NewLedEn = 0;
    if(Running == 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(hourL==4 && hourH==2){
            hourL=0;
            hourH=0;
        }
        else if(hourL==10){
            hourL=0;
            hourH++;

          }
      }
    HorseRunning((LedFlashCtr & 0xf0)>>4, Running);
      PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}
```

一、概念剖析

1. 时钟

在实验一中，涉及时钟的代码只有一行

```
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
```

时钟模块有**三个重点**：

1. 选择晶振，设置晶振频率
2. 设置PLL模块，进行倍频分频
3. 要注意系统时钟和外设时钟都要配置

以下我列出了**学习时钟时概念上的困惑**：

1. 什么是晶振，和时钟有什么关系

   晶振是一种控制频率元件，在电路模块中提供频率脉冲信号源，给电路提供一定频率的稳定的震荡（脉冲）信号，比如石英钟，就是通过对脉冲记数来走时的。在单片机内，晶振用于产生周期性的时钟信号，所以时钟和晶振的关系就是：晶振构成振荡器，振荡器的输出生成时钟脉冲信号。

2. 什么是定时器？教材中章节名明明是《时钟与系统控制》，为什么要提到CPU定时器？

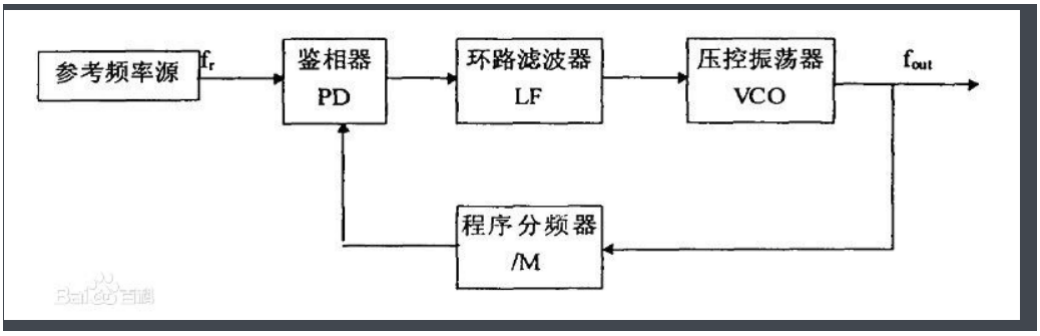
   在单片机中，时钟和定时器往往是配套出现的。定时器是一种工作在计数模式下，只计数固定周期脉冲的计数器；由于脉冲周期固定，由计数的值可以计算出时间，所以定时器有定时功能。在使用单片机时，我们往往需要每隔一段时间来触发某个功能，此时就需要使用定时器。

3. PLL锁相环是什么？

   Phase Locked Loop，是一种利用反馈控制原理实现相位和频率同步的技术，一般由鉴相器，滤波器，压控震荡器和分频器构成，它的作用是将电路输出的时钟与其外部的参考时钟保持同步。PLL需要有一个参考频率Fi。输出频率为Fo，参考频率与输出频率同时送入鉴相器。鉴相器的作用是检测输入信号和输出信号的相位差，并将检测出的相位差信号转换成uD（t）电压信号输出。当输出

信号的频率与输入信号的频率相等时，输出电压与输入电压保持固定的相位差值，即输出电压与输入电压的相位被锁住，这就是锁相环的名称由来。

当我们需要倍频时，显然在反馈通路上添加分频器即可。



4. 系统时钟和外设时钟有什么联系？什么是外设？

   这里的外设指的是片内外设，单片机内部的外设一般包括：串口控制模块，SPI模块，I2C模块，A/D模块，PWM模块，CAN模块，EEPROM，比较器模块，等等，它们都集成在单片机内部，有相对应的内部控制寄存器，可通过单片机指令直接控制。 外设指的是单片机外部的外围功能模块，比如键盘控制芯片，液晶，A/D转换芯片，等等。

   为什么有了系统时钟，还需要外设时钟呢？我们可以打个比方：系统是单片机的大脑，外设是单片机的手和脚，有的时候我们只需要动动脑筋就行，这个时候可以给外设时钟设置的慢一些，这样可以减少功耗，所以就给系统和外设分别配置了时钟。学习单片机时经常会遇到不同模块重复配置了一些设备，此时我们可以从耦合度和功耗的角度进行考量。

定时器

教材中，CPU定时器涉及四个寄存器：`PRDH:PRD` `TIMH:TIM` `TDDRH:TDDR` `PSCH:PSC`。定时器由两个部分组成：16位分频器和32位计数器。分频器对系统时钟进行分频，计数器减计数，减到0时发出中断TINT。

这四个寄存器中，PRDH:PRD 和 TIMH:TIM 为一组，可以理解为：PRD为用户设置计数器计数值存放处，TIM为计数器本身的计数值。每次开始计数时，先要把用户设置值(PRD)导入到计数器(TIM)，计数器再进行减计数。

TDDRH:TDDR 和 PSCH:PSC 为一组，与计数器的那一组类似：TDDR用于装载用户设置的分频系数，PSC为分频器本身的计数值。使用时TDDR内装载的值写入PSC中。

2. GPIO

GPIO的操作有很多，这里我们只看跑马灯部分

```
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;
}
```

书上对GPIO的引入比较跳跃，这里我们从基础概念开始讲起。

General Purpose Input Output(GPIO)，通用输入输出端口。它的核心功能就是读取外部输入，也可以从此端口向外部输出信号(高低电平)。但是书上列举了一堆别的功能：上拉电阻、多路复用、触发中断和各种寄存器。这些其实都是依据核心功能引申出的花式用法。DSP的开发商帮我们从硬件内部完成了这些功能，就不需要我们开发时通过软件实现，这样做节约了开发时间，不过学习的时候容易摸不着头脑。接下来我将逐一解释这些花式用法。

1) 多路复用

首先我们解释什么是多路复用，以及为什么要引入多路复用。

DSP中使用了流水线技术，可以提高运行速度，那么在IO口，我们也可以使用类似的方式提高系统效率，这就是"多路复用"，因为IO口并不是每时每刻都接收输入/输出，此时可以给它多连几个外设端口以提高效率。但是多路复用会存在一个问题：如果多条通路同时有数据，会存在撞车的问题，此时就需要一位交警来进行交通梳理，这位交警大哥就是多路复用寄存器(MUX)，其实就是一个数据选择器，选谁谁能走，不选就禁能。我们可以从教材中看到，MUX有许多保留位，可以留给我们进行设置。

不过我第一次看MUX寄存器表时有些懵，这里我截取教材中的部分表格进行解说：

	复位时为默认状态基本的I/O功能	**外设选择1**	**外设选择2**	**外设选择3**
GPAMUX1寄存器位	(GPAMUX1 位 = 00)	(GPAMUX1 位 = 01)	(GPAMUX1 位 = 10)	(GPAMUX1 位 = 11)
1, 0	GPIO0	EPWM1A(O)	保留	保留
3, 2	GPIO1	EPWM1B(O)	保留	COMP1OUT(O)

我们看表格的第三行，MUX的1，0位选中了GPIO0，代表这两位用于进行GPIO0的多路复用选择，当MUX = 00时，允许GPIO0通过，当MUX = 01时，允许PWM通过，其余两位保留，并且保留位用户不能自己定义，只能空着。

2) 方向寄存器

IO口可以输入也可以输出，通过方向寄存器进行设置

3) 上拉使能

刚接触上拉和下拉时，容易产生一个误会：如果设置某一位上拉，那么这一位就一直处于上拉使能状态。然而在单片机中，上拉使能是提供了一个默认的高电平状态。如果后续改变为低电平，默认状态结束。

4) 中断选择

有时，IO口伴随着中断，比如按一下按钮，就开始计时。这时我们可以令IO口作为中断源。

3. 中断

我们关注中断的开、关，以及中断向量表、中断源的配置

```
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;
EDIS;
```

中断的核心部分有三个：**中断使能**、**中断向量表**、**外设中断扩展(PIE)**。其中PIE内有一个重要的概念：**中断源**。

中断使能

在DSP中，存在可屏蔽中断和非屏蔽中断，只有可屏蔽中断需要中断使能。

对于可屏蔽中断，在我们使用的28027中，受三个控制位控制：IFR, IER, INTM。其中IER和INTM为中断使能位，IFR为中断标志位，表示中断需要响应。

中断向量

中断向量有两个等级：CPU级和外设级(PIE)，CPU级中断是真正的中断，外设级中断是通过外设触发的CPU级中断。

通用的可屏蔽CPU级中断有14个(INT1 - INT14)，PIE中断把一组外设中断连接到对应的CPU级中断

外设中断扩展

外设中断，首先要配置好中断源，对应到相应的外设中断向量，然后对应到CPU级的中断向量。

我在学习PIE中断向量时遇到疑惑：向量表中各种INT1.1, INT1.2, 一直到INT12.8，这些向量怎么配置到我们想要的触发源。然而开发商已经给我们配置好了，或者说触发源已经固定了。在表格的第五列“说明”中，我们可以看到配置好的触发源，配置中断函数时，直接用即可。比如配置定时器0(TINT0)复位时触发:

```
PieVectTable.TINT0 = &cpu_timer0_isr;// 等号左边是中断向量，右边是中断函数地址
```