实验2

陈鸿煜

华中科技大学 自动控制系

实验2主要是对PWM模块(脉冲宽度调制)和eCAP(增强型捕获)模块的使用，当然其中也包括了对时钟、GPIO和中断的操作。

以下为PWM程序

```
#include "DSP28x_Project.h"
extern int PWMIntNo,PWMDuty,Tridir;
void InitPWM4Gpio()
{
    EALLOW;
    GpioCtrlRegs.GPAPUD.bit.GPIO6 = 1;    // Disable pull-up on GPIO6 (EPWM4A)
    GpioCtrlRegs.GPAPUD.bit.GPIO7 = 1;    // Disable pull-up on GPIO7 (EPWM4B)
    GpioCtrlRegs.GPAMUX1.bit.GPIO6 = 1;   // Configure GPIO6 as EPWM4A
    GpioCtrlRegs.GPAMUX1.bit.GPIO7 = 1;   // Configure GPIO7 as EPWM4B
    EDIS;
}

void InitPWM4()
{

  int PWMPRD,DeadTime;

  EALLOW;
  SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0;
  EDIS;

  PWMPRD = 3000;
  DeadTime = 480; // ?死区时间
  EPwm4Regs.TBPRD = PWMPRD;
  EPwm4Regs.TBPHS.half.TBPHS = 0; // Set Phase register to zero
  EPwm4Regs.TBCTL.bit.CLKDIV = 0; //CLKDIV = 0;
  EPwm4Regs.TBCTL.bit.HSPCLKDIV = 0; //HSPCLKDIV = 0;
  EPwm4Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Symmetrical mode
  EPwm4Regs.TBCTL.bit.PHSEN = TB_DISABLE; // Master module
  EPwm4Regs.TBCTL.bit.PRDLD = TB_SHADOW;
  EPwm4Regs.TBCTL.bit.SYNCOSEL = TB_CTR_ZERO; // Sync down-stream module
  EPwm4Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
  EPwm4Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
  EPwm4Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO_PRD; // load on CTR=Zero or PRD
  EPwm4Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO_PRD; // load on CTR=Zero or PRD
  EPwm4Regs.AQCTLA.bit.CAU = AQ_CLEAR; // set actions for EPWM1A
  EPwm4Regs.AQCTLA.bit.CAD = AQ_SET;
  EPwm4Regs.AQCTLA.bit.CBU = 0; // set actions for EPWM1A,do nothing
  EPwm4Regs.AQCTLA.bit.CBD = 0;
  EPwm4Regs.DBCTL.bit.OUT_MODE = 1; // enable Dead-band module
  EPwm4Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC; // Active Hi complementary
  EPwm4Regs.DBFED = DeadTime;
  EPwm4Regs.DBRED = DeadTime;

  PWMDuty = 1500;
```

```
  EPwm4Regs.CMPA.half.CMPA=PWMDuty;
  EPwm4Regs.CMPB = PWMDuty;

  // Enable CNT_zero interrupt using EPWM1 Time-base
  EPwm4Regs.ETSEL.bit.INTEN = 1;   // Enable EPWM1INT generation
  EPwm4Regs.ETSEL.bit.INTSEL = 1;  // Enable interrupt CNT_zero event
    //EPwm1Regs.ETSEL.bit.INTSEL = 3;  // Enable interrupt CNT_zero & CNT_PRD
event
    EPwm4Regs.ETPS.bit.INTPRD = 1;   // Generate interrupt on the 1st event
    EPwm4Regs.ETCLR.bit.INT = 1;     // Enable more interrupts

  EALLOW;
  SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;  //同步
  EDIS;

}

interrupt void EPWM4Int_isr(void)
{

  PWMIntNo++;
  // PWMIntNo &=0x1f;
  if(PWMIntNo & 0x1){
     if(Tridir==0){
       PWMDuty +=3;
       if(PWMDuty >= 750){Tridir=1;PWMDuty=750;}
       }
     else {
       PWMDuty -=3;
       if(PWMDuty <= 0){Tridir=0;PWMDuty=0;}
       }
  }
  // if(PWMIntNo == 0)PWMDuty++;
  // if(PWMDuty > 750)PWMDuty = 0;
  //EPwm4Regs.CMPA.half.CMPA=PWMDuty;
  EPwm4Regs.ETCLR.bit.INT = 1;
  PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
  EINT;
}
```

以下为eCAP程序

```
#include "DSP28x_Project.h"
extern long int li1,li2,li3,li4,PWM_HI,PWM_LO,PWM_PRD;

void InitCAPGpio()
{
  EALLOW;
  GpioCtrlRegs.GPAPUD.bit.GPIO5 = 0;    // Enable pull-up on GPIO5 (ECAP1)
  GpioCtrlRegs.GPAMUX1.bit.GPIO5 = 3;   // Configure GPIO5 as ECAP1
  EDIS;
}

void InitCAP()
{
```

```
  ECap1Regs.ECEINT.all = 0x0000;             // Disable all capture interrupts
  ECap1Regs.ECCLR.all = 0xFFFF;              // Clear all CAP interrupt flags

  ECap1Regs.ECCTL1.bit.CAP1POL = 0;
  ECap1Regs.ECCTL1.bit.CAP2POL = 1;
  ECap1Regs.ECCTL1.bit.CAP3POL = 0;
  ECap1Regs.ECCTL1.bit.CAP4POL = 1;
  ECap1Regs.ECCTL1.bit.CTRRST1 = 0;
  ECap1Regs.ECCTL1.bit.CTRRST2 = 0;
  ECap1Regs.ECCTL1.bit.CTRRST3 = 0;
  ECap1Regs.ECCTL1.bit.CTRRST4 = 0;
  ECap1Regs.ECCTL1.bit.CAPLDEN = 1;
  ECap1Regs.ECCTL1.bit.PRESCALE = 0;
  ECap1Regs.ECCTL2.bit.CAP_APWM = 0;
  ECap1Regs.ECCTL2.bit.CONT_ONESHT = 0;
  ECap1Regs.ECCTL2.bit.SYNCO_SEL = 2;
  ECap1Regs.ECCTL2.bit.SYNCI_EN = 0;
  ECap1Regs.ECCTL2.bit.TSCTRSTOP = 1; // 允许TSCTR
  ECap1Regs.ECEINT.bit.CEVT4 = 1; // CEVT4
}

interrupt void Ecap1Int_isr(void)
{
  li1=ECap1Regs.CAP1;
  li2=ECap1Regs.CAP2;
  li3=ECap1Regs.CAP3;
  li4=ECap1Regs.CAP4;
  PWM_HI=((li2-li1)+(li4-li3)) >> 1;
  PWM_LO=li3-li2;
  PWM_PRD=((li3-li1)+(li4-li2)) >> 1;

  ECap1Regs.ECCLR.bit.CEVT4 = 1;
  ECap1Regs.ECCLR.bit.INT = 1;
  PieCtrlRegs.PIEACK.all = PIEACK_GROUP4;
  EINT;
}
```

一、概念剖析

1. PWM

谈及PWM(脉冲宽度调制)，一个重要的概念就是占空比(Duty Ratio)。对于占空比我经常有一个误会：既然是占“空”比，应该是低电平占周期的比值，然而事实与其相反，占空比说的是高电平占周期的比值。占空比展开来说，就是通电时间相对于总时间所占的比例。

PWM的作用，一个是改变平均功率，一个就是提供对应的波形，我们使用DSP芯片，主要就是为了实现脉冲宽度调制，提供不同占空比的波形，当然，其中还有很多细节，比如死区，互补PWM，作为中断源触发中断等等。

PWM比GPIO和CPU时钟都难一些，因为它有很多子模块，不过这些子模块都比较简单，我将一一讲解。

时基子模块(Time Base)

四个ePWM模块都分别拥有自己的时基，并且28027的PWM拥有同步逻辑电路，可以让四个PWM一起工作，拥有相同的时钟。配置时钟同步使用`EPWMxSYNCI`。时基子模块有时钟和计数器(之前讲过时钟模块都有自己的计数器)，那么通过控制其时钟频率和计数器计数模式，就可以控制PWM的周期，并产生事件：

1. CTR = PRD

   时基计数器的计数值等于周期时

2. CTR = Zero

   时基计数器的计数值等于0时

3. CTR = CMPB

   时基计数器的计数值等于比较寄存器的值时

4. CTR = dir

   指示实际计数器的方向，递增时该信号为高电平

5. CTR = max

   计数值等于最大值产生该信号(TBCTR = 0XFFFF)

教材上只讲了会发生这些事件，但是没说这些事件该怎么用，其实这些事件产生后，是通过动作限定器的寄存器表现出来的。动作限定器规定了产生事件时，会触发哪些“动作”，所以我们不需要手动监听事件，交给动作限定器即可。所以教材里说的”事件“其实是相对于动作限定器而言的。

当然，时基子模块也可以进行分频，计数模式也可以设置先递增后递减，它也有自己的状态寄存器，以显示自身状态。

注意：计数器递增或递减模式时，计算计数周期时，计数值要加*1*

学习计数器时，教材中把计数器的计数值称为"CTR"，我一直以为是ctrl而感到疑惑，后来才发现是COUNTER的缩写。

时基周期的映射寄存器(Shadow Register)

教材里把shadow翻译为映射，然而网络上直接译为影子，这里我们就根据教材来，称其为映射寄存器。之所以称为"Shadow"，是因为两个寄存器地址相同，可以认为是“身体和影子”。

为什么要这样设计呢？因为所有真正需要起作用的寄存器(有效寄存器)可以在同一个时间(发生更新事件时)被更新为所对应的预装载寄存器的内容，这样可以保证多个通道的操作能够准确地同步。如果没有影子寄存器，软件更新寄存器时，则直接更新了真正操作的寄存器，因为软件不可能在一个相同的时刻同时更新多个寄存器，结果造成多个通道的时序不能同步，如果再加上例如中断等其它因素，多个通道的时序关系有可能会混乱，造成是不可预知的结果。

那么，什么时候映射寄存器内的值会写入有效寄存器呢？对于时基周期的映射，当时基计数器等于=0时，进行写入。

注意：网络上可能会把影子寄存器叫做预装载寄存器，把有效寄存器称作影子寄存器。

如果要使用映射寄存器，直接操作相应的控制寄存器即可。比如时基周期的映射寄存器，启动方式为：

```
EPwmxRegs.TBCTL.bit.PRDLD = TB_SHADOW;
```

计数器-比较子模块(Count-Compare)

每个ePWM有两个计数器(CMPA & CMPB)，每个计数器都可以产生事件：

1. CTR = CMPA

   时基计数器的值等于比较器A寄存器的值

2. CTR = CPMB

   时基计数器的值等于比较器B寄存器的值

这两个比较器是独立的，产生的事件也互相独立。

比较子模块的映射寄存器

相比于时基子模块，比较子模块的映射寄存器比较特殊：它需要配置有效寄存器装载的时基，可以是时基计数器的值为0，也可以是时基计数器的值为周期值。

动作限定器子模块(Action Qualifier)

刚接触这个模块时，比较疑惑一点：什么是"限定"？根据解释，这个模块是根据不同的事件产生动作，那么为什么要用"限定"这个词？后来知道，因为这个词是直译过来的，英文的Qualifier，更合理应该翻译为"资格"，表示哪个事件有资格触发动作。所以"动作限定器"可以解释为：限定哪个事件可以触发动作的寄存器。这里的动作，就是指ePWMxA和ePWMxB产生的输出

动作限定器可以根据以下事件产生动作(置零、清零、切换)：

1. CTR = PRD
2. CTR = Zero
3. CTR = CMPA
4. CTR = CPMB

死区发生器子模块(Dead-Band Generator)

首先解释一下什么是死区，为什么要有死区。PPT上的解释为："在利用dsp对整流器或逆变器进行控制时，为了避免桥臂的上下管同时导通而导致短路，需要对输出PWM信号的上升沿或者下降沿进行延时，也就是设置PWM死区。"，我们可以通过互补PWM来理解，理想情况下，互补PWM可以保证两个输出信号在同一时刻是互补的，但是在电平交换时，肯定会存在一定的"混乱"状态，即不清楚是高电平还是低电平，那么这时可能会出现两个pwm输出都是高电平的情况。如果pwm作为桥臂上下管的输入电源，同时导通就会导致短路，所以需要一个"时延"来延迟高电平到来的时间，这样就可以确认一个器件关闭后再打开另一个器件。这个"时延"就是死区。

死区发生器中，涉及了"极性控制"，他的功能我理解了，在书中193页有具体讲解，但是其中为什么要命名为"高低电平有效/互补"，我还没搞懂，搞懂了回来写一写。

事件触发器子模块(Event Triggers)

事件触发器用于接收时基模块、计数比较模块等子模块产生的事件，可以设置发生1/2/3个事件后(预分频)，产生中断或者启动ADC。

中断名称为`EPWMxINT`，每个ePWM可以产生一个中断；ADC命令为`ADCTRIG2x+3`和`ADCTRIG2x+4`,每个ePWM可以产生两个ADC命令。

中断直接连接到PIE，直接到PIE中断向量表里就能找到。

PWM用于DAC

因为PWM输出的信号基本都是高频的(系统时钟频率很高，MHz级)，所以我们给PWM输出加一个低通滤波器，高频信号就被滤除，只剩下直流信号，然而我们知道，直流信号就是电压平均值，那么经过低通滤波后的输出电压就正比于占空比，即模拟信号。

2. eCAP

值得吐槽的是，28027只有一个捕获模块，但是厂商热心地给这个模块命名为eCAP1，导致我在学习时一直以为有多个eCAP，就像PWM一样。不过还有一个容易混淆地点：eCAP模块拥有四个时间戳捕捉寄存器(CAP1 - CAP4)，刚开始学时我把cCAP和CAP搞混了。CAP1 - 4是eCAP模块地寄存器，用于存储捕捉地计数值。

增强型捕获模块(Enhanced Capture)，用于对外部事件进行捕捉。这里我要解释两个词：“外部”和“捕捉”。

1. 外部：这里的外部是物理意义的外部，即单片机的外部，这个外部事件是需要通过引脚接入到单片机的，eCAP模块会监听这个引脚，从而进行捕捉，产生记录。
2. 捕捉：其实就是对事件进行标记，记录事件发生的时间。比如PWM，我们可以通过对他的上升沿和下降沿进行捕获从而获取其周期和占空比。

捕捉到事件以后，可以产生动作，不过这里的动作并不只是产生中断这么简单，它可以对事件进行计数、计时。至于如何实现，后面会提到。

既然要统计捕捉到事件的时间，肯定需要时钟和计数器，这里我们直接使用系统时钟，并内置了32位时基计数器。当然，对于高频信号输入，每次事件都捕获没什么意义，所以厂家贴心的内置了预分频器。

我们也可以使用限定器选择触发事件的边沿，因为eCAP有四个CAP寄存器，可以对一个连续事件捕获四次。如果我们只想进行一组捕获(只捕获四次)后就停止，就需要使用eCAP模块中的模四计数器和一个停止寄存器(2位)，停止寄存器用于标志发生几次事件后停止捕捉，和模四计数器的计数值进行比较，相等时停止捕捉，这就是单触发模式。当然也可以使用连续捕获模式，捕获值不停地被写入CAP1-4。这里提到了一个词：循环缓冲区，一开始我以为深度为4的循环缓冲区对应四个CAP寄存器，然而并不是这样的。循环缓冲区是一个独立的功能，简单的理解是：在连续捕获模式中，要不停地进行读(读取时钟计数器的值)和写(写到CAP寄存器中)操作，这样会产生冲突从而丢失数据，设置一个深度为4的缓冲区从而保证数据的可靠性与实时性。

预分频器

分频倍数只能是2的倍数，N = 2 - 62，当N = 1 时分频器被旁路。

边缘极性选择与限定器

这里的“限定”和PWM的动作限定器意思类似，就是限定哪个边沿可以触发事件。边缘极性选择与限定器，就是指这个限定器可以限定是上升沿或/和下降沿可以触发事件。

32位计数器

32位计数器（TSCTR）通过系统时钟为事件捕捉提供时间基准。也就是：该计数器提供了事件捕获的时基，由系统时钟来驱动。

4个事件中的任何一个都可以复位32位计数器，这对于偏差的捕捉非常有用。先捕捉32位计数器的值，之后任何一个LD1~LD4信号都可以将其复位为0。这样做的好处是可以统计绝对时间和相对时间：如果每捕捉一个事件就清零，那么下一次捕捉到事件的时间就是上一个事件发生后的相对时间；如果不清零，统计到的就是绝对时间。

CAP1 - CAP4时间戳捕捉寄存器

每个寄存器有32位，直接连接到时钟定时器上，可以进行计数值的装载。

中断

eCAP可以产生7种中断，在捕捉时间时，可以产生中断。并且可以单独使能/禁止不同的中断事件。在任何其他中断脉冲产生前，中断服务程序必须清除全局中断标志位。