实验3

陈鸿煜

华中科技大学 自动控制系

实验3主要是对ADC(模数转换模块)的使用。

以下为ADC程序：

```
#include "DSP28x_Project.h"
#define ADC_usDELAY 10000L
#define ADCSampT 39
#define TrigSelNo 11   //ADCTRIG11 - ePWM4, ADCSOCA
extern int adcptr;
extern unsigned int ADC_GD,ADC_FK;
extern float ADC_GDF,ADC_FKF;
float KP,KI,UK,EK,UK_1,EK_1;
interrupt void MyAdcInt1_isr(void);
void InitADC()
{

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

  /* Configure ADC pins using AIO regs*/
  // This specifies which of the possible AIO pins will be Analog input pins.
  // NOTE: AIO1,3,5,7-9,11,13,15 are analog inputs in all AIOMUX1
configurations.
  // Comment out other unwanted lines.
 /*
    EALLOW;
    GpioCtrlRegs.AIOMUX1.bit.AIO2 = 2;    // Configure AIO2 for A2 (analog input)
operation
    GpioCtrlRegs.AIOMUX1.bit.AIO4 = 2;    // Configure AIO4 for A4 (analog input)
operation
    GpioCtrlRegs.AIOMUX1.bit.AIO6 = 2;    // Configure AIO6 for A6 (analog input)
operation
    GpioCtrlRegs.AIOMUX1.bit.AIO10 = 2;   // Configure AIO10 for B2 (analog
input) operation
    GpioCtrlRegs.AIOMUX1.bit.AIO12 = 2;   // Configure AIO12 for B4 (analog
input) operation
    GpioCtrlRegs.AIOMUX1.bit.AIO14 = 2;   // Configure AIO14 for B6 (analog
input) operation
    EDIS;
  */
    DELAY_US(ADC_usDELAY);  // Delay before converting ADC channels

      EALLOW;
      AdcRegs.ADCCTL1.bit.ADCBGPWD  = 1;        // Power ADC BandGap
      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 = 1;     // Select Outside Reference
Voltage
      AdcRegs.ADCCTL1.bit.ADCREFSEL = 0;        // Select Internal Reference
Voltage
      EDIS;

      DELAY_US(ADC_usDELAY);  // Delay before converting ADC channels

      EALLOW;
      AdcRegs.ADCCTL1.bit.INTPULSEPOS   = 1;    //ADCINT1 trips after AdcResults
latch
      AdcRegs.INTSEL1N2.bit.INT1E     = 1;  //Enabled ADCINT1
      AdcRegs.INTSEL1N2.bit.INT1CONT  = 0;  //Disable ADCINT1 Continuous mode
      AdcRegs.INTSEL1N2.bit.INT1SEL = 0x09; //setup EOC9 to trigger ADCINT1 to
fire

      AdcRegs.ADCSAMPLEMODE.all = 0xff;  //Simultaneous sample for SOCAx and
SOCBx.

      //A01(IW),B01(IU),A2B2(OH),A3B3(VDC),A4B4(ASIN1),A5B5(ASIN2),A67B67(BAK)
      AdcRegs.ADCSOC0CTL.bit.CHSEL  = 0x07;   //A7,Result0;  B7,Result1
      AdcRegs.ADCSOC2CTL.bit.CHSEL  = 0x07;   //A7,Result2;  B7,Result3
      AdcRegs.ADCSOC4CTL.bit.CHSEL  = 0x07;   //A7,Result4;  B7,Result5
      AdcRegs.ADCSOC6CTL.bit.CHSEL  = 0x07;   //A7,Result6;  B7,Result7#define
      AdcRegs.ADCSOC8CTL.bit.CHSEL  = 0x07;   //A7,Result8;  B7,Result9

      AdcRegs.ADCSOC0CTL.bit.TRIGSEL    = TrigSelNo;      //ADCTRIG11- ePWM4,
ADCSOCA
      AdcRegs.ADCSOC2CTL.bit.TRIGSEL    = TrigSelNo;      //ADCTRIG11- ePWM4,
ADCSOCA
      AdcRegs.ADCSOC4CTL.bit.TRIGSEL    = TrigSelNo;      //ADCTRIG11- ePWM4,
ADCSOCA
      AdcRegs.ADCSOC6CTL.bit.TRIGSEL    = TrigSelNo;      //ADCTRIG11 - ePWM4,
ADCSOCA
      AdcRegs.ADCSOC8CTL.bit.TRIGSEL    = TrigSelNo;    //ADCTRIG11 - ePWM4,
ADCSOCA

      AdcRegs.ADCSOC0CTL.bit.ACQPS  = ADCSampT;   //Sample window is ADCSampT
cycles long
      AdcRegs.ADCSOC2CTL.bit.ACQPS  = ADCSampT;   //Sample window is ADCSampT
cycles long
      AdcRegs.ADCSOC4CTL.bit.ACQPS  = ADCSampT;   //Sample window is ADCSampT
cycles long
      AdcRegs.ADCSOC6CTL.bit.ACQPS  = ADCSampT;   //Sample window is ADCSampT
cycles long
      AdcRegs.ADCSOC8CTL.bit.ACQPS  = ADCSampT;   //Sample window is ADCSampT
cycles long
      EDIS;

   // Assumes ePWM1 clock is already enabled in InitSysCtrl();
    EPwm4Regs.ETSEL.bit.SOCAEN  = 1;        // Enable SOC on A group
    EPwm4Regs.ETSEL.bit.SOCASEL = 1;        // Enable event CTR = ZERO
    EPwm4Regs.ETPS.bit.SOCAPRD  = 1;        // Generate pulse on 1st event
    EPwm4Regs.ETCLR.bit.SOCA = 1;       // Clear SOCA flag
    }

interrupt void MyAdcInt1_isr(void)
{
```

```
    adcptr++;
    ADC_GD = AdcResult.ADCRESULT0;
    ADC_GD += AdcResult.ADCRESULT2;
    ADC_GD += AdcResult.ADCRESULT4;
    ADC_GD += AdcResult.ADCRESULT6;
    ADC_GD += AdcResult.ADCRESULT8;

    ADC_FK = AdcResult.ADCRESULT1;
    ADC_FK += AdcResult.ADCRESULT3;
    ADC_FK += AdcResult.ADCRESULT5;
    ADC_FK += AdcResult.ADCRESULT7;
    ADC_FK += AdcResult.ADCRESULT9;

    ADC_GDF= ADC_GDF *0.98 + ADC_GD*0.1611 * 0.02/10;
    ADC_FKF = ADC_FKF *0.98 + ADC_FK*0.1611 * 0.02/10;

    EK=ADC_GDF-ADC_FKF;
    KP=1;
    KI=1;
    UK=UK_1+KP*(EK-EK_1)+KI*EK;
    if(UK<0)
        UK=0;
    else if(UK>1)
        UK=1;
    EPwm4Regs.CMPA.half.CMPA=UK*12000;
    UK_1=UK;
    EK_1=EK;

    DELAY_US(ADC_usDELAY);
    AdcRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}
```

一、概念剖析

提及ADC，我们知道需要经过四个过程：采样、保持、量化、编码。在实际电路中(比如28027的ADC模块)，往往采样保持同时进行，量化编码同时进行，这对应着ADC模块中的“采样/保持(S/H)电路(Sample and Hold)”和”转换内核(多为主次比较型)“。并且ADC模块具有多路复用功能，可以进行通道选择，这样就可以输入多组模拟信号。

教材直接解释了ADC模块的各种寄存器，但是首先我们需要先学习一些ADC的术语。

1. 基本术语

分辨率和通道

当您阅读任何微控制器或ADC IC的参数规格时，ADC的参数信息将使用“通道”和“分辨率（位）”等术语给出。

例如，Arduino UNO的ATmega328有一个8通道10位ADC。术语“8通道”意味着ATmega328微控制器上有8个引脚可以读取模拟电压，每个引脚可以读取10位分辨率的电压。并非微控制器上的每个引脚都可以读取模拟电压，这因不同型号的微控制器而异。

假设我们的 ADC 范围是从 0V 到 5V，并且我们有一个 10 位 ADC，这意味着我们的输入电压 0-5 伏将被分成 1024 级离散模拟值（0000000000—1111111111，2^10 = 1024）。1024 是 10 位 ADC 的分辨率，类似地，8 位 ADC 的分辨率为 512 (2^8)，16 位 ADC 的分辨率为 65536 (2^16)。

这样，如果实际输入电压为 0V，则 MCU 的 ADC 会将其读取为 0，如果为 5V，则 MCU 将读取为 1024，如果是2.5V，则 MCU 将读取为 512。我们可以使用以下公式根据 ADC 的分辨率和参考电压计算 MCU 读取的数字值。

(ADC 分辨率 / 参考电压) = (ADC 数字值 / 实际电压值)。

28027的ADC模块分辨率为12位，有16个通道。

参考电压

您应该了解的另一个重要术语是“参考电压”。在 ADC 转换过程中，通过将其与已知电压进行比较来找到未知电压的值，这个已知电压，称为参考电压。

通常所有 MCU 都有一个设置内部参考电压的选项，这意味着您可以使用软件（程序）在内部将此电压设置为某个可用值。

在 Arduino UNO 板中，参考电压在内部默认设置为 5V，如果需要，用户也可以在软件中进行所需的更改，再通过 Vref 引脚在外部设置此参考电压。

需要注意的是，测量的模拟电压值应始终小于参考电压值，并且参考电压值应始终小于单片机的工作电压值。

采样窗口

当对某个引脚采样时，采样电容的电压需要一段时间才能累积到精确的采样电压，所以这里需要给足够的时间段进行采样，这个时间段就是采样窗口，一般以时钟周期为单位。

采样频率

也被称为采样速率，我们知道，需要事件进行触发，然后才能进行采样。当触发信号产生时，采样才开始。所以两个触发信号之间的间隔就是采样时间，采样时间的倒数就是采样频率。

序列发生器

序列信号是指在同步脉冲作用下循环地产生一串周期性的二进制信号，能产生这种信号的逻辑器件就称为序列信号发生器或序列发生器。比如可以周期产生01001011001序列信号的信号发生器，就是序列发生器。教材中写道：“与之前的那些ADC不同，这个ADC不基于序列发生器(sequencer)，而是基于SOC”。什么是基于序列发生器呢？因为事件触发ad转换动作，需要选择通道，对通道进行排序，这个排序的功能就由序列发生器提供，所以就叫基于序列发生器。当然，28027提供了更先进的方式：SOC。

SOC

一般说来， SoC(System on Chip)称为系统级芯片，也有称片上系统，意指它是一个产品，是一个有专用目标的集成电路，其中包含完整系统并有嵌入软件的全部内容。同时它又是一种技术，用以实现从确定系统功能开始，到软/硬件划分，并完成设计的整个过程。

片上系统是将计算机系统的许多元素组合到单个芯片中的集成电路。SoC 始终包含 CPU，但也可能包括系统内存、外围设备控制器（用于 USB、存储）和更高级的外围设备，如图形处理单元 （GPU）、专用神经网络电路、无线电调制解调器（用于蓝牙或 Wi-Fi）等。片上系统方法与具有CPU芯片和独立控制器芯片，GPU和RAM的传统PC形成鲜明对比，可以根据需要更换，升级或互换。SoC的使用使计算机更小，更快，更便宜，耗电更少。

在28027的ADC模块中，SOC特指用于处理模数转换的系统。它可以配置该模块的触发源、转换通道和采样窗口的尺寸。28027的ADC模块有16个SOC。

2. 具体配置

概述

ADC的配置，大体可以分为八点：

1. 配置引脚和工作（参考）电源
2. 配置转换顺序和采样方式
3. 配置转换速度(采样保持时间，转换时间)
4. 配置触发方式
5. 配置优先级
6. 配置结果读取方式
7. 配置中断方式
8. 其他特殊功能

触发源、通道和采样窗口

28027的ADC模块由SOC驱动，所以触发源、转换通道和采样窗口都是由SOC的寄存器配置。

1. SOCx的触发源由`ADCSOCxCTL`寄存器以及`ADCINTSOCSEL1/ADCINTSOCSEL2`联合配置
2. 通道和采样窗口尺寸由`ADCSOCxCTL`配置

由于输入的不一致性，有的输入需要很短的时间就能把采样电容器充电完成，有的需要更多的时间。为了保证充电完成，每个`ADCSOCxCTL`寄存器都包含一个六位字段`ADQPS`，可以设置S/H窗的长度。

每个SOC都可以独立配置输入触发器，当然多个不同的SOC也可以配置成使用同一个触发源。当我们需要使用连续转换时，可以配置ADC模块的中断做为触发源。

每个SOC都可以独立配置转换哪个通道的输入。SOC有两种采样模式：

1. 顺序采样

   当被配置为顺序采样时，可以通过寄存器配置要转换的通道

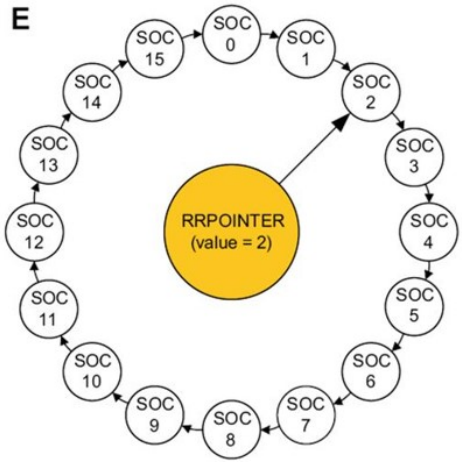
2. 同步采样

   也就是说并行采样，我们知道28027有两个采样保持电路，所以只能同步采样两个通道，可以通过寄存器进行配置。

轮询机制

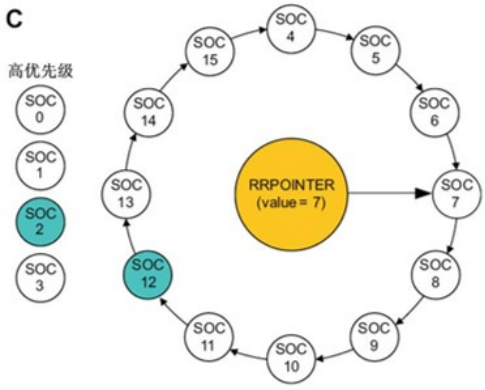
在顺序采样模式中，如果几个SOC同时被触发，但是采样保持电路只能一次运行一个(顺序模式中)，那么就需要一种机制来决定优先级。默认的机制为**轮询机制**，当然也可以使用`ADCSOCPRIORITYCTL`寄存器来分配优先级。

在轮询机制中，各个SOC的优先级在整体上是一样的，当发生优先级的竞争时，谁的优先级高纯靠运气，优先级仅取决于轮询指针(RRPOINTER)。轮询指针的值实时保存到寄存器`ADCSOCPRIORITYCTL`中，它指向上一个被转换的SOC。下一个比轮询指针大的值就是当前优先级最高的SOC，并在SOC15后又绕回SOC0。复位时，RRPOINTER的值为32，此时SOC0为最高优先级。



分配优先级

如果不使用默认优先级机制(轮询机制)，那么可以自主分配优先级。`ADCSOCPRIORITYCTL`寄存器中的`SOCPRIORITY`字段可以为所有SOC分配高优先级。当某个或某些SOC被配置为高优先级时，他们会被从轮询环中取出，单独摆成一列，当发生优先级竞争时，优先被取出的SOC。并且被配置为高优先级的SOC之间也有优先级，标号越小优先级越高。



同步采样模式

有时要求采样的两个信号之间延迟很短，或者最好能够同时得到采样结果，由于28027有两个独立的采样保持电路那么我们可以使用同步采样模式对两个通道进行同时采样。同步采样模式使用`ADCSAMPLEMODE`寄存器对一对SOCx进行配置。

偶数编号的SOCx和**它之后**的奇数编号的SOCx(比如SOC0和SOC1)配成一对，一起连接一个使能位(此时为SIMULEN0)。

同步采样的配对规则如下：

1. 首先转换A通道
2. A通道的转换结果存放在偶数编号的`ADCRESULTx`寄存器中，B通道的转换结果则存放在奇数编号的`ADCRESULTx`寄存器中
3. 同步采样时，一般我们只配置偶数SOCx。

参考电压

默认情况选择的是内部参考电压(0 ~ 3.3V)。

使用外部参考电压时，要用VREFHI/VREFLO管脚作为参考电压的引脚。

3. 中断操作

在系统时钟的定时器中，有TINT中断；在GPIO中，有外部中断XINT；在捕获模块中，有捕获中断ECAP_INT；在脉冲宽度调制模块中，有中断EPWM_INT；那么在数模转换模块里，也会有数模转换中断**ADCINT**，并且有九种不同的中断。

当然，由于ADC模块有16组SOC，不可能所有SOC都能作为触发源，我们需要通过寄存器`INTSELxNy`来配置由哪个SOC作为触发源。当转换开始或转换结束时，每个SOC都会发出自己的脉冲`EOCx`，这个就是中断触发源。

此外，ADCINT1和ADCINT2信号可以配置成产生一个SOCx触发事件，这对连续转换来说非常有用。