MSP430学习心经

MSP430学习心经（一）Clock

MSP430根据型号的不同最多可以选择使用3个振荡器。我们可以根据需要选择合适的振荡频率，并可以在不需要时随时关闭振荡器，以节省功耗。这3个振荡器分别为：  
（1）DCO    数控RC振荡器。它在芯片内部，不用时可以关闭。DCO的振荡频率会受周围环境温度和MSP430工作电压的影响，且同一型号的芯片所产生的频率也不相同。但DCO的调节功能可以改善它的性能，他的调节分为以下3步：

a：选择BCSCTL1.RSELx确定时钟的标称频率；

b：选择DCOCTL.DCOx在标称频率基础上分段粗调；

c：选择DCOCTL.MODx的值进行细调。

在430F2系列中DCO（最大16MHz）设置如下：

    BCSCTL1 =CALBC1\_8MHZ;                   //设定DCO为1MHZ  
    DCOCTL = CALDCO\_8MHZ;              //CALBC1\_XMHZ和CALDCO\_XMHZ共8个数据保存在Flash信息存储器Segment段0x10F8~0x10FF地址中。共可设置1，8，12，16MHZ。  
  
（2）LFXT1    接低频振荡器。典型为接32768HZ的时钟振荡器，在LF模式（XTS=0），直接连接在XIN与XOUT之间,此时振荡器不需要接负载电容，软件控制XCAPx位来设置LF模式下内部提供的负载电容。可选1、6、10、12.5pF。在HF模式（XTS=1），可以接450KHZ~8MHZ的标准晶体振荡器，此时需要接负载电容.LXFT1产生的频率信号为ACLK.低速时钟需要上百毫秒的建立时间才能稳定下来.

        F2xx增加了一个新的超低功耗晶振VLO（12kHz），具有500nA的待机模式。他的使用与LFXT1互斥。BCSCTL3|=LFXT1S1;//ACLK来源于VLO

（3）XT2    接450KHZ~8MHZ的标准晶体振荡器。外部标准晶体振荡器接在XT2IN和XT2OUT之间,此时需要接负载电容，不用时可以关闭。  
  
      低频振荡器主要用来降低能量消耗，如使用电池供电的系统，高频振荡器用来对事件做出快速反应或者供CPU进行大量运算。  
  
      MSP430的3种时钟信号：MCLK系统主时钟；SMCLK系统子时钟；ACLK辅助时钟。  
  
（1）MCLK系统主时钟。除了CPU运算使用此时钟以外，外围模块也可以使用。MCLK可以选择任何一个振荡器所产生的时钟信号并进行1、2、4、8分频作为其信号源。  
  
（2）SMCLK系统子时钟。供外围模块使用。并在使用前可以通过各模块的寄存器实现分频。SMCLK可以XT2CLK或者DCOCLK振荡器所产生的时钟信号并进行1、2、4、8分频作为其信号源。  
  
（3）ACLK辅助时钟。供外围模块使用。并在使用前可以通过各模块的寄存器实现分频。但ACLK只能由LFXT1进行1、2、4、8分频作为信号源。可以作为后台时钟用来唤醒CPU.  
  
      PUC复位后，MCLK和SMCLK的信号源为DCO，DCO的振荡频率为800K或1MHZ。ACLK的信号源为LFXT1。  
  
       MSP430内部含有晶体振荡器失效监测电路，当时钟信号丢失50us时，监测电路捕捉到振荡器失效。监测LFXT1（工作在高频模式或低频模式）和XT2输出的时钟信号，若失效，分别为LFXT1OF和XT2OF置位，振荡器失效标志位OFIFG置位。如果MCLK信号来自LFXT1或者XT2，那么MSP430自动把MCLK的信号切换为DCO，这样可以保证程序继续运行。

当测试到晶体振荡器失效（LFXT1OF或XT2OF）时，振荡器失效标志位OFIFG置位，并且锁定到POR。当OFIFG置位，MCLK来自DCO，并且如果OFIE置位，那么产生非屏蔽中断。中断得到响应后，OFIE自动复位。OFIFG必须软件清零。失效源可以通过测试各个失效位识别。

MSP430学习心经（二）TimerA

430F2xxx的TA是一个16位的定时器/计数器，有3个"捕获/比较寄存器",定时器(TA)与"捕获/比较寄存器"(CCRx)工作是相对独立的.

         计数器TAR三种模式(增计数，连续计数，增减计数)。分设置MC1、MC0位，设置后，TA就开始计数了。上电默认为MC1=0，MC0=0为停止。  
        那么欲定时一段时间周期的话,可以有两种方法:一是TA计数,计到一定值产生TA中断;二是用CCRx和计数值比较,一致的话产生CCRx中断.(比较模式，也是上电默认模式)中断。每当一定的时间间隔都会产生中断请求，那么，需将下一事件发生的时间在当前的中断程序中加到CCRx中。见程序：

#include "msp430x22x4.h"  
int main( void )  
{  
     WDTCTL=WDTPW+WDTHOLD;  
     BCSCTL1=CALBC1\_1MHZ;  
    DCOCTL = CALDCO\_1MHZ;  
     TACTL=TASSEL1+TACLR;//SMCLK=VLOCLK=12KHZ，清楚TAR  
     CCTL0=CCIE;//中断允许  
     CCR0=50000;//方波频率：1m/50000  
     P1DIR|=BIT0;  
     TACTL|=MC1;//连续计数  
     \_EINT();  
     \_BIS\_SR(LPM0\_bits);  
     return 0;  
}

#pragma vector=TIMERA0\_VECTOR  
\_\_interrupt void TIMER\_A(void)  
{  
     P1OUT^=BIT0;  
  CCR0+=50000;  
}

         捕获模式： 位于控制字CCTLX中的模式位CAPX的置位将选择捕获模式。捕获模式用于时间的精确定位，她可以用在速度计算或时间测量中.若在选定的输入引脚上发生选定脉冲触发沿.则定时器计数的值将被复制到捕获寄存器CCRX中.完成捕获后。控制字CCTLX中的中断标志CCIFGX置位。如果通用中断允许位GIE和相应的中断标志位CCIEX置位，则产生中断请求.复位表示在下一次捕获完成前捕获数据已被读取.如果捕获数据还未读取时第二次捕获数据已锁存，则寄存器CCTLX中的溢出位COVX置位，检查这一位可以使程序从失去同步状态中恢复.需要用软件来复位COVX.

      定时器暂停时捕获应该停止.顺序应是先停止捕获功能，再停止定时器计数.捕获功能重新开始时，顺序应先开始捕获功能，再开始定时器计数.     
附程序：

#include "msp430x22x4.h"  
volatile unsigned int cap\_value;             //存两次捕值之差  
int main( void )  
{  
     volatile unsigned int first\_value,second\_value,n,max=10;  
     WDTCTL=WDTPW+WDTHOLD;                      //关看门狗  
     BCSCTL1 =CALBC1\_8MHZ;                      //设定DCO为1MHZ  
// DCOCTL =CALBC1\_8MHZ;  
     DCOCTL = CALDCO\_8MHZ;  
     BCSCTL1 |=DIVA\_3;                          //ACLK输入八分频  
     do                                           //等待晶振稳定  
     {  
       IFG1 &=~OFIFG;  
       for(n=5;n>0;n--,max--);  
       if(max<=0)  
       {  
         BCSCTL3|=LFXT1S1;//aclk来源于VLO  
         break;  
       }  
     }     while(OFIFG&IFG1);     
    
     TACCTL2=CM1+CCIS\_1+CAP;                    //捕获模式,捕获上升沿,捕获内部ACLK  
     TACTL=TASSEL\_2+MC\_2+TACLR;                 //定时器时钟源为SMCLK,启动连续计数,清TAR  
     TACCTL2 &=~CCIFG;                          //清CCR2的标志位    
     while(!(CCIFG&TACCTL2));                   //查CCR2标志位  
     first\_value=TACCR2;                        //第一个捕获值存入first\_value  
     TACCTL2 &=~CCIFG;                          //清CCR2的标志位  
     while(!(CCIFG&TACCTL2));                   //查CCR2标志位  
     second\_value=TACCR2;                       //第二个捕获值存入second\_value  
     TACCTL2 &=~CCIFG;                          //清CCR2的标志位                        
     TACCTL2 &=~MC1;                            //关定时器A    
     cap\_value=(second\_value-first\_value);     
     return 0;    
}

        通过比较模式，输出得到PWM波形（脉宽调制，PWM是一种对模拟信号电平进行数字编码的方法，是利用微处理器的数字输出来对模拟电路进行控制的一种非常有效的技术，广泛应用在从测量、通信到功率控制与变换的许多领域中。）TimerA可以得到两路，分别以CCR0位计数周期。设置CCR1，CCR2的值（小于CCR0）形成一定的占空比。输出时候将IO口设置为外部模块输出。

附程序：

int main( void )  
{  
     WDTCTL=WDTPW+WDTHOLD;                      //关看门狗  
     BCSCTL1 =CALBC1\_1MHZ;                      //设定DCO为1MHZ  
     DCOCTL = CALDCO\_1MHZ;          
     TACTL=TASSEL1+TAR;                         //SMCLK为时钟源,清TAR  
     CCR0=512;                                  //设定PWM周期  
     CCTL1 |=OUTMOD\_7;                          //CCR1输出为reset/set模式  
     CCR1=384;                                  //CCR1的PWM占空比设定  
     CCTL2 |=OUTMOD\_7;                          //CCR2输出为reset/set模式  
     CCR2=128;                                  //CCR2的PWM占空比设定  
     P1SEL |=BIT2+BIT3;                         //TA1,TA2输出功能  
     P1DIR |=BIT2+BIT3;                        
     TACTL |=MC0;                               //启动定时器A增计数模式  
     \_BIS\_SR(CPUOFF);    
     return 0;  
}

中断：

Two interrupt vectors are associated with the 16-bit Timer\_A module:  
    TACCR0 interrupt vector for TACCR0 CCIFG  
    TAIV interrupt vector for all other CCIFG flags and TAIFG  
In capture mode any CCIFG flag is set when a timer value is captured in the  
associated TACCRx register. In compare mode, any CCIFG flag is set if TAR  
counts to the associated TACCRx value. Software may also set or clear any  
CCIFG flag. All CCIFG flags request an interrupt when their corresponding  
CCIE bit and the GIE bit are set.

The TACCR1 CCIFG, TACCR2 CCIFG, and TAIFG flags are prioritized and  
combined to source a single interrupt vector. The interrupt vector register TAIV  
is used to determine which flag requested an interrupt.

For example:  
//  
//  
// As coded and with TACLK = 32768Hz, toggle rates are:  
// P1.1 = TACCR0 = 32768/(2\*4) = 4096Hz  
// P1.2 = TACCR1 = 32768/(2\*16) = 1024Hz  
// P1.3 = TACCR2 = 32768/(2\*100) = 163.84Hz  
// P1.0 = overflow = 32768/(2\*65536) = 0.25Hz  
//  
//              MSP430F22x4  
//            -----------------  
//        /|\|              XIN|-  
//         | |                 | 32kHz  
//         --|RST          XOUT|-  
//           |                 |  
//           |         P1.1/TA0|--> TACCR0  
//           |         P1.2/TA1|--> TACCR1  
//           |         P1.3/TA2|--> TACCR2  
//           |             P1.0|--> Overflow/software  
//  
// A. Dannenberg  
// Texas Instruments Inc.  
// April 2006  
// Built with IAR Embedded Workbench Version: 3.41A  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
#include "msp430x22x4.h"

void main(void)  
{  
WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT  
P1SEL |= 0x0E;                            // P1.1 - P1.3 option select  
P1DIR |= 0x0F;                            // P1.0 - P1.3 outputs  
TACCTL0 = OUTMOD\_4 + CCIE;                // TACCR0 toggle, interrupt enabled  
TACCTL1 = OUTMOD\_4 + CCIE;                // TACCR1 toggle, interrupt enabled  
TACCTL2 = OUTMOD\_4 + CCIE;                // TACCR2 toggle, interrupt enabled  
TACTL = TASSEL\_1 + MC\_2 + TAIE;           // ACLK, contmode, interrupt enabled

\_\_bis\_SR\_register(LPM3\_bits + GIE);       // Enter LPM3 w/interrupt  
}

// Timer A0 interrupt service routine  
#pragma vector=TIMERA0\_VECTOR  
\_\_interrupt void Timer\_A0(void)  
{  
TACCR0 += 4;                              // Add Offset to TACCR0  
}

// Timer\_A3 Interrupt Vector (TAIV) handler  
#pragma vector=TIMERA1\_VECTOR  
\_\_interrupt void Timer\_A1(void)  
{  
switch (\_\_even\_in\_range(TAIV, 10))        // Efficient switch-implementation  
{  
    case 2: TACCR1 += 16;                  // Add Offset to TACCR1  
             break;  
    case 4: TACCR2 += 100;                 // Add Offset to TACCR2  
             break;  
    case 10: P1OUT ^= 0x01;                 // Timer\_A3 overflow  
             break;  
}  
}

输出模式：

MSP430学习心经（三）复位、BOR

POR&PUC

POR是上电复位信号，它只在以下三个事件发生时产生：  
1、芯片上电。  
2、—RST/NMI设置成复位模式，在RST/NMI引脚上出现低电平信号。3、PORON=1时，一个SVS低状态（不懂）。  
  
PUC信号是上电清除信号，POR信号的产生总会产生PUC信号，但PUC信号的发生不会产生POR信号。产生它的事件为：  
1、发生POR信号。  
2、处于看门狗模式下，看门狗定时时间到。  
3、看门狗定时器写入错误的安全键值。  
4、FLASH存储器写入错误的安全键值。  
5、一个CPU指令从外设地址范围0h~01ffh取数据。  
POR和PUC两者的关系：POR信号的产生会导致“系统复位”并“产生PUC信号”。而PUC信号不会引起POR信号的产生。

无论是POR信号还是PUC信号触发的复位,都会使MSP430从地址0xFFFE处读取复位中断向量,程序从中断向量所指的地址处开始执行。触发PUC信号的条件中,除了POR产生触发PUC信号外,其他的可通过读取相应的中断向量来判断是何种原因引起的PUC信号,以便作出相应的处理。  
         系统复位（指POR）后的状态为：（1）—RST/NMI管脚功能被设置为复位功能;（2）所有I/O管脚被设置为输入;（3）外围模块被初始化,其寄存器值为相关手册上的默认值;（4）状态寄存器SR复位;（5）看门狗激活,进入工作模式;（6）程序计数器PC载入0xFFFE处的地址,微处理器从此地址开始执行程序。  
         典型的复位电路有一下3种：  
（1）       在RST/NMI管脚上接100K欧的上拉电阻。  
（2）在（1）的基础上再接0.1uf的电容,电容的一端接地,可以使复位更加可靠。  
（3）在（2）的基础上,再在电阻上并接一个型号为IN4008的二极管,可以可靠的实现系统断电后立即上电。

BOR

         在没有BOR的芯片中“如果芯片的上电是周期性的，则掉电VCC必须降低到Vmin，以保证VCC再次加载时发生新的POR信号。如果在一个周期中VCC没有下降到低于Vmin，或者因为发生干扰，那么POR信号就不会发生，这样上电后的初始状态将是不正确的。  
        对于带BOR的模块，应该是“当VCC超过Vcc(start)后POR信号变得有效，直到VCC超过V(B\_IT+)，然后再经过一个延时t(BOR)后。延时t(BOR)会根据Vcc电压变高的倾斜角度的减小而相应的延长。滞后  
Vhys(B\_IT-)使得VCC必须降到V(B\_IT-)之下才能保证通过BOR电路再一次产生POR信号。而V(B\_IT-)是比Vmin高的，这就使得当VCC没有降到Vmin时BOR能够提供一个有效的电源失效重起信号。

        高精度设备的的电池更换会引起电压波动。零功率低压重置（BOR）功能用来低电压条件下重置MSP430,预防器件不可知的行为。