```
void AdcOffsetSelfCal()
{
    Uint16 AdcConvMean;
    EALLOW;
    AdcRegs.ADCCTL1.bit.ADCREFSEL = 0;                //Select
internal reference mode
    AdcRegs.ADCCTL1.bit.VREFLOCONV = 1;               //Select
VREFLO internal connection on B5
    AdcChanSelect(13);                               //Select channel
B5 for all SOC
    AdcRegs.ADCOFFTRIM.bit.OFFTRIM = 80;              //Apply
artificial offset (+80) to account for a negative offset that may
reside in the ADC core
    AdcConvMean = AdcConversion();                   //Capture ADC
conversion on VREFLO
    AdcRegs.ADCOFFTRIM.bit.OFFTRIM = 80 - AdcConvMean;  //Set offtrim
register with new value (i.e remove artical offset (+80) and create
a two's compliment of the offset error)
    AdcRegs.ADCCTL1.bit.VREFLOCONV = 0;               //Select
external ADCIN5 input pin on B5
    EDIS;
}

void InitSysCtrl(void)
{
   DisableDog();
        EALLOW;
        SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 1; // Enable ADC
peripheral clock
        (*Device_cal)();
        SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 0; // Return ADC clock to
original state
        EDIS;
   IntOsc1Sel();
   InitPll(DSP28_PLLCR,DSP28_DIVSEL);
   InitPeripheralClocks();
}

void IntOsc1Sel (void) {
    EALLOW;
    SysCtrlRegs.CLKCTL.bit.INTOSC1OFF = 0;
    SysCtrlRegs.CLKCTL.bit.OSCCLKSRCSEL=0;  // Clk Src = INTOSC1
    SysCtrlRegs.CLKCTL.bit.XCLKINOFF=1;     // Turn off XCLKIN
    SysCtrlRegs.CLKCTL.bit.XTALOSCOFF=1;    // Turn off XTALOSC
```

```
    SysCtrlRegs.CLKCTL.bit.INTOSC2OFF=1;    // Turn off INTOSC2
    EDIS;
}


void InitPll(Uint16 val, Uint16 divsel)
{
   volatile Uint16 iVol;

   // Make sure the PLL is not running in limp mode
   if (SysCtrlRegs.PLLSTS.bit.MCLKSTS != 0)
   {
      EALLOW;
      // OSCCLKSRC1 failure detected. PLL running in limp mode.
      // Re-enable missing clock logic.
      SysCtrlRegs.PLLSTS.bit.MCLKCLR = 1;
      EDIS;
      // Replace this line with a call to an appropriate
      // SystemShutdown(); function.
      asm("       ESTOP0");     // Uncomment for debugging purposes
   }

   // DIVSEL MUST be 0 before PLLCR can be changed from
   // 0x0000. It is set to 0 by an external reset XRSn
   // This puts us in 1/4
   if (SysCtrlRegs.PLLSTS.bit.DIVSEL != 0)
   {
       EALLOW;
       SysCtrlRegs.PLLSTS.bit.DIVSEL = 0;
       EDIS;
   }

   // Change the PLLCR
   if (SysCtrlRegs.PLLCR.bit.DIV != val)
   {

      EALLOW;
      // Before setting PLLCR turn off missing clock detect logic
      SysCtrlRegs.PLLSTS.bit.MCLKOFF = 1;
      SysCtrlRegs.PLLCR.bit.DIV = val;
      EDIS;
      DisableDog();
      while(SysCtrlRegs.PLLSTS.bit.PLLLOCKS != 1)
      {
```

```
            // Uncomment to service the watchdog
            // ServiceDog();
        }

        EALLOW;
        SysCtrlRegs.PLLSTS.bit.MCLKOFF = 0;
        EDIS;
    }

    // If switching to 1/2
    if((divsel == 1)||(divsel == 2))
    {
        EALLOW;
        SysCtrlRegs.PLLSTS.bit.DIVSEL = divsel;
        EDIS;
    }
    if(divsel == 3)
    {
        EALLOW;
        SysCtrlRegs.PLLSTS.bit.DIVSEL = 2;
        DELAY_US(50L);
        SysCtrlRegs.PLLSTS.bit.DIVSEL = 3;
        EDIS;
    }
}

void InitPeripheralClocks(void)
{
   EALLOW;

// LOSPCP prescale register settings, normally it will be set to
default values

   GpioCtrlRegs.GPAMUX2.bit.GPIO18 = 3;  // GPIO18 = XCLKOUT
   SysCtrlRegs.LOSPCP.all = 0x0002;

// XCLKOUT to SYSCLKOUT ratio.  By default XCLKOUT = 1/4 SYSCLKOUT
   SysCtrlRegs.XCLK.bit.XCLKOUTDIV=2; // Set XCLKOUT = SYSCLKOUT/1

// Peripheral clock enables set for the selected peripherals.
// If you are not using a peripheral leave the clock off
// to save on power.
//
// Note: not all peripherals are available on all 2802x derivates.
```

```
// Refer to the datasheet for your particular device.
//
// This function is not written to be an example of efficient code.

   SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 0;       // ADC
   SysCtrlRegs.PCLKCR3.bit.COMP1ENCLK = 0;     // COMP1
   SysCtrlRegs.PCLKCR3.bit.COMP2ENCLK = 0;     // COMP2
   SysCtrlRegs.PCLKCR3.bit.CPUTIMER0ENCLK = 1; // CPU Timer-0
   SysCtrlRegs.PCLKCR3.bit.CPUTIMER1ENCLK = 0; // CPU Timer-1
   SysCtrlRegs.PCLKCR3.bit.CPUTIMER2ENCLK = 0; // CPU Timer-2
   SysCtrlRegs.PCLKCR1.bit.ECAP1ENCLK = 1;     // eCAP1
   SysCtrlRegs.PCLKCR1.bit.EPWM1ENCLK = 1;     // EPWM1
   SysCtrlRegs.PCLKCR1.bit.EPWM2ENCLK = 1;     // EPWM2
   SysCtrlRegs.PCLKCR1.bit.EPWM3ENCLK = 1;     // EPWM3
   SysCtrlRegs.PCLKCR1.bit.EPWM4ENCLK = 1;     // EPWM4
   SysCtrlRegs.PCLKCR3.bit.GPIOINENCLK = 1;    // GPIO
   SysCtrlRegs.PCLKCR0.bit.HRPWMENCLK=0;       // HRPWM
   SysCtrlRegs.PCLKCR0.bit.I2CAENCLK = 0;      // I2C
   SysCtrlRegs.PCLKCR0.bit.SCIAENCLK = 0;      // SCI-A
   SysCtrlRegs.PCLKCR0.bit.SPIAENCLK = 0;      // SPI-A

  // SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;      // Enable TBCLK
within the EPWM

   EDIS;
}
void InitFlash(void)
{
   EALLOW;
   //Enable Flash Pipeline mode to improve performance
   //of code executed from Flash.
   FlashRegs.FOPT.bit.ENPIPE = 1;

   //               CAUTION
   //Minimum waitstates required for the flash operating
   //at a given CPU rate must be characterized by TI.
   //Refer to the datasheet for the latest information.

  #if (CPU_FRQ_60MHZ)
   //Set the Paged Waitstate for the Flash
   FlashRegs.FBANKWAIT.bit.PAGEWAIT = 2;

   //Set the Random Waitstate for the Flash
   FlashRegs.FBANKWAIT.bit.RANDWAIT = 2;
```

```
   //Set the Waitstate for the OTP
   FlashRegs.FOTPWAIT.bit.OTPWAIT = 2;

  #elif (CPU_FRQ_40MHZ)
   //Set the Paged Waitstate for the Flash
   FlashRegs.FBANKWAIT.bit.PAGEWAIT = 1;

   //Set the Random Waitstate for the Flash
   FlashRegs.FBANKWAIT.bit.RANDWAIT = 1;

   //Set the Waitstate for the OTP
   FlashRegs.FOTPWAIT.bit.OTPWAIT = 1;
  #endif
   //               CAUTION
   //ONLY THE DEFAULT VALUE FOR THESE 2 REGISTERS SHOULD BE USED
   FlashRegs.FSTDBYWAIT.bit.STDBYWAIT = 0x01FF;
   FlashRegs.FACTIVEWAIT.bit.ACTIVEWAIT = 0x01FF;
   EDIS;

   //Force a pipeline flush to ensure that the write to
   //the last register configured occurs before returning.

   asm(" RPT #7 || NOP");
}

void InitPieCtrl(void)
{
    // Disable Interrupts at the CPU level:
    DINT;

    // Disable the PIE
    PieCtrlRegs.PIECTRL.bit.ENPIE = 0;

    // Clear all PIEIER registers:
    PieCtrlRegs.PIEIER1.all = 0;
    PieCtrlRegs.PIEIER2.all = 0;
    PieCtrlRegs.PIEIER3.all = 0;
    PieCtrlRegs.PIEIER4.all = 0;
    PieCtrlRegs.PIEIER5.all = 0;
    PieCtrlRegs.PIEIER6.all = 0;
    PieCtrlRegs.PIEIER7.all = 0;
    PieCtrlRegs.PIEIER8.all = 0;
    PieCtrlRegs.PIEIER9.all = 0;
```

```
    PieCtrlRegs.PIEIER10.all = 0;
    PieCtrlRegs.PIEIER11.all = 0;
    PieCtrlRegs.PIEIER12.all = 0;

    // Clear all PIEIFR registers:
    PieCtrlRegs.PIEIFR1.all = 0;
    PieCtrlRegs.PIEIFR2.all = 0;
    PieCtrlRegs.PIEIFR3.all = 0;
    PieCtrlRegs.PIEIFR4.all = 0;
    PieCtrlRegs.PIEIFR5.all = 0;
    PieCtrlRegs.PIEIFR6.all = 0;
    PieCtrlRegs.PIEIFR7.all = 0;
    PieCtrlRegs.PIEIFR8.all = 0;
    PieCtrlRegs.PIEIFR9.all = 0;
    PieCtrlRegs.PIEIFR10.all = 0;
    PieCtrlRegs.PIEIFR11.all = 0;
    PieCtrlRegs.PIEIFR12.all = 0;

}

void InitPieVectTable(void)
{
    int16   i;
    Uint32 *Source = (void *) &PieVectTableInit;
    Uint32 *Dest = (void *) &PieVectTable;

    // Do not write over first 3 32-bit locations (these locations are
    // initialized by Boot ROM with boot variables)

    Source = Source + 3;
    Dest = Dest + 3;

    EALLOW;
    for(i=0; i < 125; i++)
        *Dest++ = *Source++;
    EDIS;

    // Enable the PIE Vector Table
    PieCtrlRegs.PIECTRL.bit.ENPIE = 1;

}
```