```c
/*
  Clayton Auld
  EE 444 Embedded Systems Design
  Design Project: MSP430F5529 - Flex Sensor Voltage over UART

  This program sets up the MSP430F5529 Launchpad to communicate with a PC over UART/USB at 115200 baud.

  Anything sent over UART will be echoed back. Pressing the return key reports the number of characters entered.

  When "MEAS" is sent, the chip reports voltage on P6.0 (2.5 V max), and time since "MEAS"
  was entered. The measurements are taken twice per second.

  Sending "STOP" will stop the measurements.

  Pressing Switch 1 will also start and stop the measurements.

  The chip is running at 8 MHz MCLK
*/

#include <msp430.h>                                     // Include MSP430 header file
#include <stdio.h>                                      // Enable sprintf function
//extern int IncrementVcore(void);                      // Declare prototype for IncrementVcore
//extern int DecrementVcore(void);                      // Declare prototype for DecrementVcore

// ************ Variables *************************************
char string[70];                                   // Set up string for output over UART or LCD
int count = 0;                                          // Set a counter variable
unsigned int i=0;                                       // Another counter variable for USCI TX
unsigned int start = 0;                                 // Flag for UART start or stop input received
char buffer[4];                                         // Buffer for RX data
float time = 0.0;                                       // Counter to calculate time since MEAS received over UAR

// Variables for ADC Calibration
unsigned int CAL_ADC2_5VREF_FACTOR;
unsigned int CAL_ADC_GAIN_FACTOR;
unsigned int CAL_ADC_OFFSET;

union float2bytes { float f; char b[sizeof(float)]; };  // Create a union to store the byte array for voltage
                                                        // We can then use this to transmit the bytes over UART c
                                                        // serial communication in a properly formatted data pack
union float2bytes volts;                                // Union variable to store voltage bytes for transmissior

void main(void)
{
    //***************************** Core Voltage Settings *******************************
    //DecrementVcore();
    //DecrementVcore();
    //IncrementVcore();                                   // Increase the core voltage to next level higher

    //************************* UCS Control Register Settings ************************
    UCSCTL1 = DCORSEL_4;                                // Set DCO frequency range

    /* Crystal is 32,768 kHz. To get a 25 MHz clock solve this: 32768*(x-1)=25,000,000
        Remember that the FLL multiplies by n+1 and not n
    */

    UCSCTL2 = 244 | FLLD_0;                             // Set n multiplier for FLL prescaler and proper FLL divi
    UCSCTL3 = SELREF_0;                                 // Set SELREF to XT1CLK
    UCSCTL4 = SELM__DCOCLK |                            // Select DCOCLK as MCLK source
              SELS__DCOCLK;                             // Select DCOCLK as SMCLK source
```

```
UCSCTL5 = DIVS__1 | DIVM__1;                          // Set SMCLCK divider to 1 and MCLK divider to 1.
UCSCTL8 ^= SMCLKREQEN;                                // Disable clock request logic for SMCLK

//*************** TIMER SETTINGS ***************************************************
TA1CTL = TASSEL__ACLK |                               // TA1 source = SMCLK
         TACLR |                                      // Clear TA1 timer
         TAIE |                                       // Enable timer interrupts
         MC__STOP;                                    // Set TA1 to STOP
         //ID__8;                                     // Set TA1 first divider to 8
TA1EX0 = TAIDEX_7;                                    // Set TA1 second divider to divide by 8

TA1CCTL1 = OUTMOD_7;                                  // PWM output mode: 7 - PWM reset/set

TA1CCR0 |= 2042;                                      // Count is set to produce a 2 Hz signal
//TA1CCR1 = 1020;           //** DEBUGGING            // PWM signal with 50% duty cycle

//*************** ADC12_A Settings **************************************************

CAL_ADC2_5VREF_FACTOR = *(unsigned int *)0x01A2C;
CAL_ADC_GAIN_FACTOR = *(unsigned int *)0x01A16;
CAL_ADC_OFFSET = *(unsigned int *)0x01A18;

REFCTL0 = REFMSTR |                                   // REFCTL0 Master control ON
          REFVSEL_2 |                                 // REFCTL0 reference voltage = 2.5 V
          REFOUT |                                    // REFCTL0 output reference available externally
          REFON;                                      // Turn on reference voltage

ADC12CTL0 = ADC12SHT0_4 |                             // Set ADC12CLK to 4*0x001 (1024) clock cycles
            ADC12MSC |                                // ADC12 Multiple Sample-Conversion
            ADC12ON;                                  // Turn on ADC12_A

ADC12CTL1 = ADC12SHS_0 |                              // Set Sample/Hold source to ADC12SC
            ADC12SHP |                                // Turn on Sample/Hold Pulse Mode
            ADC12DIV_7 |                              // Divide input clock by 8
            ADC12SSEL_3 |                             // Set clock source to SMCLK
            ADC12CONSEQ_1;                            // ADC12 Conversion Sequence Select: Sequence-of-channels

ADC12CTL2 = ADC12RES_2 |                               // Set conversion resolution to 12 bits
            ADC12TCOFF;                                // Turn off the ADC temperature sensor to save power

//*************** ADC12_A Sequence Settings *****************************************
ADC12MCTL0 = ADC12SREF_1 |                            // Select reference V(R+) = AVCC and V(R-) = AVSS
             ADC12INCH_0;                             // Set input channel to A0

ADC12MCTL1 = ADC12SREF_1 |                            // Select reference V(R+) = AVCC and V(R-) = AVSS
             ADC12INCH_0;                             // Set input channel to A0

ADC12MCTL2 = ADC12SREF_1 |                            // Select reference V(R+) = AVCC and V(R-) = AVSS
             ADC12INCH_0;                             // Set input channel to A0

ADC12MCTL3 = ADC12EOS |                               // Set MEM9 as end of sequence
             ADC12SREF_1 |                            // Select reference V(R+) = AVCC and V(R-) = AVSS
             ADC12INCH_0;                             // Set input channel to A0

ADC12IE = ADC12IE3;                                   // Enable interrupt when MEM3 is written (EOS)

ADC12CTL0 |= ADC12ENC;                                // Enable conversion

//*********************** UART Settings *****************************************
```

```
    UCA1CTL1 |= UCSWRST |                              // USCI Software Reset
                UCSSEL_2;                              // USCI 0 Clock Source: 2 (SMCLK)

    // ************** Baud rate oversampling settings requires some calculations.
    // **********Refer to Raskovic's slides in ...USCI_UART_with_ink.pdf page 51

    // Baud Rate = 115200 at 8 MHz clock

    UCA1BR0 = 4;                                       // UCBRx = 4; Set low byte to 4; Baud rate = 115200
    UCA1BR1 = 0;                                       // UCBRx = 0; Set high byte to 0
    UCA1MCTL |= UCBRF_5 +                              // UCBRFx = 5
                UCBRS_3 +                              // UCBRSx = 3
                UCOS16;                                // Enable oversampling mode

    // ***************** End baud rate settings******************************

    UCA1CTL1 &= ~UCSWRST;                              // Start the USCI state machine
    UCA1IE |= UCRXIE;                                  // Enable interrupt on USCI_A1 RX

    //************************* Port Settings ***************************************

    P1DIR |= BIT0;                                     // Set Port 1 Pin 0 as output (LED1)
    P1OUT ^= ~BIT0;                                    // Turn off LED1

    P2DIR |= BIT0 | BIT2;    //** DEBUGGING            // Set Port 2 Pins 0, 2 as output
    P2SEL |= BIT0 | BIT2;    //** DEBUGGING            // Set Port 2 Pins 0, 2 as aux function (TA1.1 | SMCLK)
    P2REN |= BIT1;                                     // Enable pullup/pulldown resistor on Port 2 Pin 1 (Switc
    P2OUT |= BIT1;                                     // Set pullup/pulldown to pullup resistor on Port 2 Pin 1
    P2IFG &= ~BIT1;                                    // Clear interrupt flag on Port 2 Pin 1 (Switch1).
    P2IE  |= BIT1;                                     // Enable interrupts on Port 2 Pin 1 (Switch1).

    P4DIR |= BIT7;                                     // Set Port 4 Pin 7 output (LED2)
    P4OUT ^= ~BIT7;                                    // Turn off LED2
    P4SEL |= BIT4 | BIT5;                              // Set Port 4, Pins 5 and 6 as USCI RX and TX

    P6REN |= BIT0;                                     // Enable pullup/pulldown resistor on Port 6 Pin 0 (A0).
    P6OUT |= BIT0;                                     // Set pullup/pulldown to pullup resistor on Port 6 Pin 0

    //*******************************************************************************


    _EINT();                                           // Globally enable interrupts

    //while(1) {}                                      //

    LPM0;                                              // Enter low-power mode 0; ACLK, SMCLK active; CPU, MCLK

    /* Before entering LPM1 we need to wait
       3*(reference clock period) otherwise DCO will drift
       This comes from device erratasheet.
    */
    //__delay_cycles(100000);                          // Bug fix for entering LPM1
    //LPM1;                                            // Enter low-power mode 1;

    //LPM3;                                            // Enter low-power mode 3; ACLK active; CPU, SMCLK, MCLK,

    //return 0;

}                                                      //
```

```
//******************** UART Interrupt Service Routine **************************

//__attribute__((interrupt(USCI_A1_VECTOR))) void __USCI_ISR(void) // USCI ISR
void USCI_A1_ISR(void) __interrupt [USCI_A1_VECTOR]
{
    switch(UCA1IV)
    {
        case 0: break;                                  // Vector 0 - no interrupt
        case 2:                                         // Vector 2 - RXIFG
            i=0;                                        //
            string[0] = UCA1RXBUF;                      // Repeat entered characters
            //string[1] = 0x0;                            // Add null strings for end of string detection
            //string[2] = 0x0;                            // Add null strings for end of string detection
            UCA1IE |= UCTXIE;                           // Enable interrupt on USCI_A1 TX
            while (!(UCA1IFG & UCTXIFG));               // USCI_A1 TX buffer ready?
            UCA1TXBUF = string[i++];                    // Echo character received
            count++;                                    // Add up the number of received characters

//****************************** Check for characters of interest and process them accordingly ****************
            // Create a buffer array to hold last 4 characters received.
            // This can be thought of as a software shift register for 8 bit characters
            buffer[3] = buffer[2];
            buffer[2] = buffer[1];
            buffer[1] = buffer[0];
            buffer[0] = UCA1RXBUF;

            if (UCA1RXBUF == 0x00d)                     // Return key pressed
            {
                count--;
                i=0;                                    //
                if (buffer[1] == 0x00d)
                    sprintf(string, "Characters Entered: %d\n\r", count);
                else sprintf(string, "\nCharacters Entered: %d\n\r", count);        //
                UCA1IE |= UCTXIE;                       // Enable interrupt on USCI_A1 TX
                while (!(UCA1IFG & UCTXIFG));           // USCI_A1 TX buffer ready?
                UCA1TXBUF = string[i++];                //

            }

            else if (buffer[3] == 0x4D /* M */ && buffer[2] == 0x45 /* E */ && buffer[1] == 0x41 /* A */ && buffe
            {
                start = 1;                              // Set start flag to 1
                P1OUT ^= BIT0;                          // Toggle LED1
                TA1CTL |= MC__UP;                       // Set TA1 to start counting UP
            }

            else if (buffer[3] == 0x53 /* S */ && buffer[2] == 0x54 /* T */ && buffer[1] == 0x4F /* O */ && buffe
            {
                start = 0;                              // Set start flag to 0
                time = 0;                               // Reset the timer variable
                P1OUT ^= BIT0;                          // Toggle LED1
                TA1CTL |= MC__STOP |                    // Set TA1 to STOP
                          TACLR;                        // Clear the timer
            }
            break;
//***************************************** End checking for characters of interest ***************************

        case 4:                                         // Vector 4 - TXIFG
```

```c
            UCA1TXBUF = string[i++];                    // Send next character
            if (i >= sizeof(string) || string[i] == 0x0)// Detect null character or end of string
            {
                UCA1IE &= ~UCTXIE;                      // Disable interrupt on USCI_A1 TX
            }
            break;                                      //
        default: break;                                 //
    }
}
//******************** ADC12 Interrupt Service Routine ***************************

void __ADC12_ISR(void) __interrupt [ADC12_VECTOR]       // ADC12 ISR
//__attribute__((interrupt(ADC12_VECTOR))) void __ADC12_ISR(void) // ADC12 ISR
{
    float volts_raw=0.0;
    int counter;                                        //
    float time1 = 0.0;                                  // Local variable for time calculation
    switch(ADC12IV)                                     //
    {
        case ADC12IV_ADC12IFG3:                         // Vector 20: ADC12IFG7
            ADC12IFG &= ~ADC12IFG3;                     // Clear interrupt flag
            for (counter=0; counter<4; counter++)       //
            {
                volts_raw += *(&ADC12MEM0 + counter);
            }
            //volts = volts_raw/4*(2.497/4095);
            /******************************************** TO DO: ADC Calibration ******************************
            To calibrate the ADC and use the voltage reference offsets, use the following formula taken from

            https://e2e.ti.com/support/microcontrollers/msp430/f/166/t/204428

            and from pages 82 and 83 of the document slau208p.pdf (MSP430x5xx and MSP430x6xx Family User's Guide)

            ADC(calibrated) = ( (ADC(raw) x CAL_ADC15VREF_FACTOR / 2^15) x (CAL_ADC_GAIN_FACTOR / 2^15) ) + CAL_A

            The values in the equation are available in the TLV structure in the datasheet for each chip.

            *************************************************************************************************
            volts.f = (((volts_raw * CAL_ADC2_5VREF_FACTOR / 32768)*(CAL_ADC_GAIN_FACTOR / 32768)) + CAL_ADC_OFFS
            //volts_bytes.f = volts_calibrated;
            time1 = time * 0.5;                         // Calculate number of seconds since TEMP
            sprintf(string, "Time since MEAS: %.1f seconds\n\rP6.0 Voltage: %.3f V\n\r----------\n\r", time1, vol
            sprintf(string, volts.b);

            /* To send the formatted data over UART to the proper module as a 4-byte array simply add this line:

            sprintf(string1, volts.b);

            where "string1" is the string to be sent over the proper UART port. In this program "string" is tied
            UART connected to the PC. Another version (ie. "string1") could be tied to the UART module connected
            the Bluetooth chip.
            */

            while (!(UCA1IFG & UCTXIFG));               // USCI_A1 TX buffer ready?
            i=0;                                        //
            UCA1IE |= UCTXIE;                           // Enable interrupt on USCI_A1 TX
            UCA1TXBUF = string[i++];                    //
            ADC12CTL0 ^= ADC12ENC;
            break;
        default: break;
```

```
    }
}

//******************** ADC12 Interrupt Service Routine ***************************
//__attribute__((interrupt(TIMER0_A1_VECTOR))) void __T0A1_ISR(void)  // Timer0_A1 ISR
void __TIMER_A_ISR(void) __interrupt [TIMER1_A1_VECTOR]
{
    switch(TA1IV)
    {
    case 14:
      if (start == 1)
      {
          ADC12CTL0 |= ADC12SC | ADC12ENC;              // Start ADC sample-and-conversion
          time++;                                       // Iterate the time variable
          P4OUT ^= BIT7;                                // Toggle LED2
      }
      break;
    default: break;
    }
}


//******************** Port2 (Switch1) Interrupt Service Routine ***************************
void __SWITCH_ISR(void) __interrupt [PORT2_VECTOR]

/* This switch shows a lot of bouncing that has to be accounted for in software.
   The basic way of doing this is adding delay, turning off interrupts on the switch during the ISR,
   and clearing interrupt flags as the first and last tasks of the ISR. This is not the most low-power
   way of doing things, but it **mostly** works. A better way of doing this would be to use a timer
   and check that a specific amount of time has passed before triggering the ISR process.
*/

{
  switch(P2IV)
  {
    case P2IV_P2IFG1:                           // Start TimerA1 when switch is pressed
        P2IE ^= BIT1;                           // Disable switch interrupt for switch debouncing
        __delay_cycles(5000);                   // Basic switch debouncing delay
        P2IFG = 0x0;                            // Clear Port2 ISR flags
        if (start == 0)
        {
            start = 1;                          // Set start flag to 1
            P1OUT ^= BIT0;                      // Toggle LED1
            TA1CTL |= MC__UP;                   // Set TA1 to start counting UP
        }
        else if (start == 1)
        {
            start = 0;                          // Set start flag to 0
            time = 0;                           // Reset the timer variable
            P1OUT ^= BIT0;                      // Toggle LED1
            TA1CTL |= MC__STOP |                // Set TA1 to STOP
                      TACLR;                     // Clear the timer
        }
        P2IE ^= BIT1;                           // Reenable switch interrupt
        __delay_cycles(5000);                   // Add delay for switch debouncing
      break;
    default:
      break;
  }
}
```