```
/*
 * File:   thermo_4bit_lcd.c
 * Author: Phil
 *
 * Created on 25 October 2016, 17:55
 *  LM35 temp sensor based thermometer with 4 bit interface LCD
 *  8 bit commands are written to the LCD by setting RS  to 1 (data) or 0 (command)
 *  then the upper nibble is sent to PORTC followed by a E clock pulse, then
 *  the lower nibble us sent to PORTC followed by an E clock pulse
 *  Interrupt driven ADC and LCD refresh
 */



// PIC16F690 Configuration Bit Settings


// 'C' source line config statements

// This project demonstrates how to play a musical tune using two arrays and a for loop "Happy birthday"

#include <xc.h>


// #pragma config statements should precede project file includes.

// Use project enums instead of #define for ON and OFF.


// CONFIG

#pragma config FOSC = INTRCIO   // Oscillator Selection bits (INTOSCIO oscillator: I/O function on RA4/OSC2/CLKOUT pin, I/O function on RA5/OSC1/CLKIN)

#pragma config WDTE = OFF       // Watchdog Timer Enable bit (WDT disabled and can be enabled by SWDTEN bit of the WDTCON register)

#pragma config PWRTE = OFF      // Power-up Timer Enable bit (PWRT disabled)
```

```
#pragma config MCLRE = ON      // MCLR Pin Function Select bit (MCLR pin function is MCLR)

#pragma config CP = OFF        // Code Protection bit (Program memory code protection is disabled)

#pragma config CPD = OFF       // Data Code Protection bit (Data memory code protection is disabled)

#pragma config BOREN = OFF     // Brown-out Reset Selection bits (BOR disabled)

#pragma config IESO = OFF      // Internal External Switchover bit (Internal External Switchover mode is disabled)

#pragma config FCMEN = OFF     // Fail-Safe Clock Monitor Enabled bit (Fail-Safe Clock Monitor is disabled)

#define _XTAL_FREQ 4000000     // internal clock frequency 4MHz



#define  RS  PORTAbits.RA0     // RS = 1 = data register, 0 = Instruction register pin 19

#define E PORTAbits.RA1        // enable write pin 18

#define D4  PORTCbits.RC0      // D7 MSB ...D4 LSB of 4 bit LCD data bus

#define D5  PORTCbits.RC1

#define D6 PORTCbits.RC2

#define D7  PORTCbits.RC3


char counter = 0;

unsigned int temp, temp_celcius = 0;

unsigned int thous, huns, tens, units = 0;

unsigned int hi_nibble_thous, lo_nibble_thous,hi_nibble_huns, lo_nibble_huns = 0;

unsigned int hi_nibble_tens, lo_nibble_tens,hi_nibble_units, lo_nibble_units = 0;

void init_ports()

{

  OSCCONbits.IRCF2 = 1;       // 4MHz clock

  OSCCONbits.IRCF1 = 1;

  OSCCONbits.IRCF0 = 0;
```

```
  TRISC = 0x00;      // RC4 diagnostic LED to show entry into ADC interrupt routine
                     // RC5 diagnostic LED to show entry into LCD refresh interrupt routine
  TRISA = 0x00;    //set to outputs - RA0 = RS, RA1 = E (latch pulse)
  PORTA = 0x00;
  PORTC = 0x00;             // clear PORTC
  ANSEL = 0x00;             // ADC module off
  ANSELH = 0x00;            //
  CM1CON0 = 0x00;           // comparators off
  CM2CON0 = 0x00;
}
void enable_interrupts()
{
   INTCONbits.GIE = 1;       // enable global interrupts
   PIE1bits.ADIE = 1;        // ADC interrupt enable set
   INTCONbits.PEIE = 1;      // enable peripheral interrupts eg ADC
   PIR1bits.ADIF = 0;        // clear ADC interrupt flag, ready for next interrupt
   // LCD refresh interrupt
   TMR0 = 0x00;              // start TMR0 from a count of zero
   INTCONbits.T0IE =  1;     // interrupts enabled for TMR0
   OPTION_REGbits.T0CS = 0;   // internal clock fosc/4
   OPTION_REGbits.T0SE = 0;   // TMR0 increments on low to high clock
   OPTION_REGbits.PSA = 0;    // pre-scaler assigned to TMR0
   OPTION_REGbits.PS2 = 1;    //1:256 pre-scaler gives time to overflow of 65.536ms
   OPTION_REGbits.PS1 = 1;
   OPTION_REGbits.PS0 = 1;
   INTCONbits.T0IF = 0;
}
```

```
void adc_config()
{

   //****************select input pin for ADC*********************************
   TRISBbits.TRISB4 = 1;      // RB4 is an input for A/D converter
   ANSELHbits.ANS10 = 1;      // channel 10 analog input enabled (RB4 pin 13)
   ADCON0bits.CHS3 = 1;       // AN10 select bits for connection to sample & hold
   ADCON0bits.CHS2 = 0;       // AN10 select bits for connection to sample & hold
   ADCON0bits.CHS1 = 1;       // AN10 select bits for connection to sample & hold
   ADCON0bits.CHS0 = 0;       // AN10 select bits for connection to sample & hold
   // ****************set voltage conversion reference************************
   ADCON0bits.VCFG = 0;       // use internal 5V voltage reference (Vdd)

   ADCON1bits.ADCS2 = 0;      //Fosc/8 is the conversion clock
   ADCON1bits.ADCS1 = 0;      //This is selected because the conversion
   ADCON1bits.ADCS0 = 1;      //clock period (Tad) must be greater than 1.5us.
                              //With a Fosc of 4MHz, Fosc/8 results in a Tad
                              //of 2us.
   //MORE DETAILED EXPLAINATION OF ADCS<2.0>
   // We then select our conversion clock source.
   //The conversion clock must be selected so that the period is at least 1.5us
   //(referred to as TAD in the datasheet). Since we are using a 4MHz clock,
   //selecting Fosc/8 will provide a TAD of 2us (1/4,000,000 * 8 = 0.000002s = 2us).
   //We could select Fosc/16, 32, or 64, but then the conversion would take longer
   //than it has to.
   //****************RH justification setting*********************************
   ADCON0bits.ADFM = 1;       // Right hand justified conversion result
```

```
   // **************Turn ON ADC modulw**************************************
   ADCON0bits.ADON = 1;       // ADC module now switched on, ready for conversion
   ADCON0bits.GO_DONE = 1; // start A to D conversion
}

void interrupt adc()
{
   if (PIR1bits.ADIF == 1)     // test to see if ADC caused interrupt?
   {                    // True
      //temp = ADRESL + (ADRESH << 8);          // Moved to main loop
      //temp_celcius = (float)temp * 4.8875855;
      PIR1bits.ADIF = 0;      // clear ADC interrupt flag ready for next interrupt
      //ADCON0bits.GO_DONE = 1; // start A to D conversion  // Moved to main loop
      PORTCbits.RC4 = ~PORTCbits.RC4;// diagnostic LED to show entry to isr
   }
   if (INTCONbits.T0IF == 1)   // test to see if ADC caused interrupt?
   {                    //True
      if(counter > 15)       //TMR0 configuered to overflow every 65.5ms
      {                    // so to achieve a 1sec refresh rate of LCD
                           // counter needs 15 * 65.5ms to achieve 1sec
         write_temp();      // refresh temp reading on LCD
         counter = 0;       // reset counter from 15 to 0
         TMR0 = 0x00;       // start TMR0 counting from zero
         INTCONbits.T0IF = 0;// clear TMR0 interrupt flag to enable further interrupts
         PORTCbits.RC5 = ~PORTCbits.RC5;// diagnostic LED to show entry to isr
      }
      counter = counter + 1;  // when counter less the 15, increment it by 1
```

```
      INTCONbits.T0IF = 0;    // clear TMR0 interrupt flag to allow further interrupts
   }
}
void function_set (char a, char b, char c, char d, char e)//function to latch commands (RS=0)
// or data (RS=1) in 4 bit nibbles to the LCD, hi nibble first, then low nibble, each nibble
// followed by a E latch pulse to latch command or data into LCD
{
   RS = a;        // RA0 pin 19
   D7 = b;        // RC3 pin 6
   D6 = c;        // RC2 pin 14
   D5 = d;        // RC1 pin 15
   D4 = e;        // RC0 pin 16
}
void clock ()
{
   E = 1;         // E = RA1 pin 18
   __delay_ms(1);  // E latch pulse to get commands into LCD
   E = 0;
}
void lcd_config()
{
   __delay_ms(100);           // power on, wait for 100ms
   function_set(0,0,0,1,1);    // RS=0, D7 = 0, D6 = 0, D5 = 1, D4 = 1 - 8 bit mode (1)
                        // 8 bit function set (lower 4 bits are irrelevant)
   __delay_ms(5);
   clock();
```

```
function_set(0,0,0,1,1);    // RS=0, D7 = 0, D6 = 0, D5 = 1, D4 = 1 - 8 bit mode (2)
                    // 8 bit function set (lower 4 bits are irrelevant)
__delay_ms(1);
clock();

function_set(0,0,0,1,1);    // RS=0, D7 = 0, D6 = 0, D5 = 1, D4 = 1 - 8 bit mode (3)
                    // 8 bit function set (lower 4 bits are irrelevant)
 __delay_ms(1);
clock();

function_set(0,0,0,1,0);    // RS=0, D7 = 0, D6 = 0, D5 = 1, D4 = 0 - 4 bit mode (4)
                    //initial function set to change from 8 bit interface to 4 bit
                    // lower 4 bits are irrelevant
 __delay_ms(1);
clock();

function_set(0,0,0,1,0);    // RS=0, D7 = 0, D6 = 0, D5 = 1, D4 = 0 - 4 bit mode (5a)
 __delay_ms(1);           // two nibble 4 bit functions set, send 0010 first then 1100
clock();                 // (1100 = N F **) N = number of LCD lines (0 = 1, 1 = 2 lines)
                    // F = font size (0 = small, 1 = big)

function_set(0,1,1,0,0);    // RS=0, D7 = 1, D6 = 1, D5 = 0, D4 = 0 - 4 bit mode (5b)
 __delay_ms(1);           // see 5a above for description
clock();

function_set(0,1,1,0,0);    // RS=0, D7 = 1, D6 = 1, D5 = 0, D4 = 0 - 4 bit mode    (6a)
```

```
   __delay_ms(1);           // display ON/ OFF control
   clock();
   function_set(0,1,0,0,0);    // RS=0, D7 = 1, D6 = 0, D5 = 0, D4 = 0 - 4 bit mode (6b)
    __delay_ms(1);
    clock();
   function_set(0,0,0,0,0);    // RS=0, D7 = 0, D6 = 0, D5 = 0, D4 = 0 - 4 bit mode  (7a)
    __delay_ms(1);           // clear display control
   clock();

   function_set(0,0,0,0,1);    // RS=0, D7 = 1, D6 = 1, D5 = 1, D4 = 0 - 4 bit mode  (7b)
    __delay_ms(1);
   clock();

   function_set(0,0,0,0,0);    // RS=0, D7 = 1, D6 = 1, D5 = 1, D4 = 0 - 4 bit mode  (8a)
    __delay_ms(1);           //entry mode set first nibble
   clock();

   function_set(0,0,1,1,0);    // RS=0, D7 = 1, D6 = 1, D5 = 1, D4 = 0 - 4 bit mode   (8b)
    __delay_ms(1);           // entry mode set second nibble - I/D  S as required
   clock();
                        // END OF INITIALISATION

//*****************************************************************************
****
   function_set(0,0,0,0,0);    // RS=0, D7 = 0, D6 = 0, D5 = 0, D4 = 0 - 4 bit mode
    __delay_ms(1);           // display ON/OFF control                      (9a)
  clock();
```

```
   function_set(0,1,1,0,0);    // RS=0, D7 = 1, D6 = 1, D5 = 0, D4 = 0 - 4 bit mode    (9b)
    __delay_ms(1);            // D (display) = 1(ON), C (cursor) and B (blinking cursor) as required
   clock();
}
void text()
{
   function_set(0,1,0,0,0); // command mode RS = 0 position 80 to write "Temp = "
   clock();
   function_set(0,0,0,0,0);
   clock();
   function_set(1,0,1,0,1);// data mode RS = 1 write T
   clock();
   function_set(1,0,1,0,0);
   clock();
   function_set(1,0,1,1,0);// data mode RS = 1 write e
   clock();
   function_set(1,0,1,0,1);
   clock();
   function_set(1,0,1,1,0);// data mode RS = 1 write m
   clock();
   function_set(1,1,1,0,1);
   clock();
   function_set(1,0,1,1,1);//data mode RS = 1 write p
   clock();
   function_set(1,0,0,0,0);
   clock();
   function_set(1,0,0,1,0);//data mode RS = 1 write space
```

```
    clock();
    function_set(1,0,0,0,0);
    clock();
    function_set(1,0,0,1,1);// data mode RS = 1 write =
    clock();
    function_set(1,1,1,0,1);
    clock();
    function_set(0,1,0,0,0); // command mode RS = 0 start writing at position 8C (deg C)
    clock();
    function_set(0,1,1,0,0);
    clock();
    function_set(1,1,1,0,1);//data mode RS = 1 write degree
    clock();
    function_set(1,1,1,1,1);
    clock();
    function_set(1,0,1,0,0);//data mode RS = 1 write C
    clock();
    function_set(1,0,0,1,1);
    clock();
}
write_temp()
    {

    thous = (temp_celcius/1000) + 0x30; // split thous digit and add offset
    // 0x30 as LCD character ROM numeric symbols start at base address 0x30 (0011 0000)
    // i.e. zero = 0x30, 1 = 0x31 etc.....
    huns = ((temp_celcius/100)%10) + 0x30; // split huns digit and add offset
```

```
   tens = ((temp_celcius/10)%10) + 0x30; // split tens digit and add offset
   units = (temp_celcius%10) + 0x30; // split units digit and add offset

   // this code suppresses the leading zero in the display for temps below 100 deg C
   if (thous > 0x30) // if thous digit is > 0
   {
      thous = ((temp_celcius/1000)) + 0x30; // display thous digit
   }
   else
   {
      thous = 0x20; // otherwise display 'white space' instead
   }

//*****************************************************************************
*
   function_set(0,1,0,0,0); // command mode RS = 0 position 8B to write actual temperature data
   clock();
   function_set(0,0,1,1,1);
   clock();

   RS = 1;
      hi_nibble_thous = ((thous >> 4) & 0b00001111); //data mode RS = 1 write thous digit
      PORTC = hi_nibble_thous;
      clock();
   RS = 1;
     lo_nibble_thous = (thous & 0b00001111);
      PORTC = lo_nibble_thous;
      clock();
```

```
RS = 1;
   hi_nibble_huns = ((huns >> 4) & 0b00001111); //data mode RS = 1 write huns digit
   PORTC = hi_nibble_huns;
   clock();
RS = 1;
  lo_nibble_huns = (huns & 0b00001111);
   PORTC = lo_nibble_huns;
   clock();

RS = 1;

   hi_nibble_tens = ((tens >> 4) & 0b00001111); //data mode RS = 1 write tens digit
   PORTC = hi_nibble_tens;
   clock();
RS = 1;
  lo_nibble_tens = (tens & 0b00001111);
   PORTC = lo_nibble_tens;
   clock();

   PORTC =0b00000010;   //data mode RS = 1 write decimal point
   clock();
   PORTC = 0b00001110;
   clock();

RS = 1;
```

```
    hi_nibble_units = ((units >> 4) & 0b00001111); //data mode RS = 1 write units digit
    PORTC = hi_nibble_units;
    clock();
  RS = 1;
    lo_nibble_units = (units & 0b00001111);
    PORTC = lo_nibble_units;
    clock();
  }

void main(void)
{
  init_ports();
  lcd_config();
  enable_interrupts();
  adc_config();
  text();                 // Prints "Temp =     deg C" on LCD




 while(1)                 // infinite loop
  {
   __delay_us(10);           //Wait the acquisition time (about 10us).
   ADCON0bits.GO_DONE = 1;     //start the conversion
   while(ADCON0bits.GO_DONE ==1) //wait for the conversion to end
       {
          ;
```

```
        }
     temp = ADRESL + (ADRESH << 8);    // combine ADC conversion result into a 10 bit temp
     temp_celcius = (float)temp * 4.8828; // transform temp into celcius units
     //write_temp();              // moved to ISR to allow interrupt by TMR0
    // __delay_ms(500);           // instead of burning uC cycles in __delay_ms()
   }
}
```