ECE 3720
Microcomputer Interfacing Laboratory
Section 7
Travis McCormick
Date Demonstrated: 04 / 22 / 2019
Microcontroller Thermostat

ABSTRACT:
A thermostat is designed and created with at PIC32 microcontroller as the processing unit. It will feature a temperature sensor, LCD display, and cooling fan with adjustable speed and temperature.

INTRODUCTION:

Temperature feedback and thermostats are critical components for more than just HVAC systems. Convection ovens, computer processor cooling systems, and vehicle engines all have some feature of temperature control. Using a TMP36 temperature sensor and an ADC, we can easily add temperature feedback to the PIC32 microcontroller and implement a cooling system with a fan when the temperature rises above a set threshold.

EXPERIMENTAL PROCEDURES:

To implement the thermostat, the project was divided into several different modules: LCD display, temperature sensing, user input, and output motor driver.

LCD Display:

The library for the LCM1602C LCD is based off the Hitachi HD44780 LCD driver datasheet and "LiquidCrystal" library for the Arduino. It is written in C++ to easily implement in future projects, but this required the X32 compiler to be updated to v2.15. The library by default will use port B on the PIC32 to communicate with the LCD because of the physical location of the pins are near each other. All operations with port B are masked, so only the bits required for the LCD are affected.

PIN:	1	2	3	4	5	6	7-14	15	16
FUNC:	GND	VDD	VO	RS	RW	E	D0-D7	LED+	LED-

Table 1: LCD Pinout

Connections to the LCD:

- VDD to 5V
- VO to GND
- RS to B5 on PIC32
- RW to GND
- E to B6 on PIC32
- D4 - D7 to B0 - B3 on the PIC32 respectively
- LED+ to 5V with 220Ω resistor
- LED- to GND

The LCD can communicate with the microcontroller in a few different ways. The easiest way is in 8-bit mode which is default with powering on, but this requires 8 data connections to the microcontroller in addition to RS, E, and RW if used. Another option is 4-bit mode which will require sending each byte in two halves, but it reduces the amount of physical connections. To send data to the LCD, bits are written to the data pins, RS is set high or low, and E is pulsed high for 1 millisecond and then held low for 1 millisecond. The RS bit tells the LCD whether to write the data to the display, or interpret it as a command. RS low means data is a command, and high means data is to be displayed. The sequence in Figure 1 needs to be sent to the LCD after being powered on in order to enable 4-bit mode.

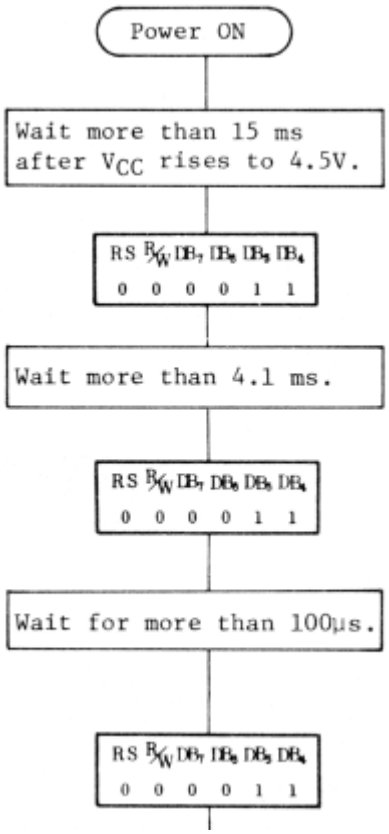

Figure 1: 4-bit Mode Sequence

After initializing the LCD in 4-bit mode, commands can be set to configure features such as character size, cursor blink, and display shift, then it is ready to display text. A full list of commands and their bit sequence is shown in the Hitachi HD44780 datasheet.

Temperature Sensor:

The TMP36 is a linear sensor that outputs 10 mV per degree Celsius before adding an offset of 0.5V. The sensor's supply is connected to 3.3V and its output is connected to A1 on the microcontroller. The internal ADC on the PIC32 is used to read this value. The reading is converted to a voltage, the 0.5V offset is subtracted, and then it is multiplied by 100 to obtain the temperature in degrees Celsius.

User Input:

The user input consists of a Grayhill 61C optical encoder along with a push button and debouncer circuit. Pin 6 on the encoder is VCC connected to 5 V and pin 1 is connected to GND. Pins 4 and 5 are pulled up to 5 V with 10KΩ resistors and connected to INT3 and INT0 on the microcontroller respectively. Interrupts are configured by monitoring the state of the other output and the direction of its own trigger as shown in Figure 2. The encoder is used to change the set temperature or the speed of the fan depending on what state the system is in.

Clockwise Rotation		
Position	Output A	Output B
1		
2	●	
3	●	●
4		●

● Indicates logic high; blank indicates logic low. Code repeats every 4 positions.

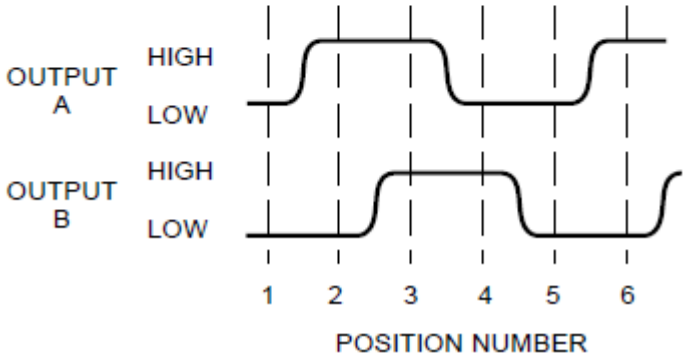

Figure 2: Encoder States

The push button is used to switch between the two different modes: temperature and fan speed. The NAND debouncer is made of an SR latch connected to a SPDT push button. The pole of the button is connected to ground and the two throws are connected to the inputs of the latch and are pulled up to 5V through 10KΩ resistors. The output of latch is connected to INT1 on the microcontroller. Pressing the button will trigger an interrupt and toggle between the settings to change the temperature or change the speed of the fan.

Output Motor Driver:

Pin B9 on the microcontroller is the PWM signal for the L293DNE driver chip. B9 will have a PWM signal either when the temperature is greater than the set temperature or when in the fan speed selection mode. The L293DNE chip is powered by 5V on pins 8 and 16, and grounded on pins 4 and 5. Pin 1 is pulled up to 3.3 V to always enable, and B9 is connected to pin 2 for speed control. Pins 3 and 4 are connected to the motor in parallel with a diode with flow direction from pin 4 to 3.

RESULSTS AND DISCUSSION:

Without a motor connected to the output, the thermostat functions exactly as expected. It starts initially in temperature mode with a set temperature of 70 degrees Fahrenheit. Turning the encoder changes the set temperature with a minimum of 0 degrees and a maximum of 999

degrees Fahrenheit. The lower bound is set to avoid having to deal with negative numbers and excessive code for a situation that is not common for this implementation. The same goes for the upper bound, but instead to avoid displaying more digits than realistically possible for this device. The output of the motor driver can be observed on the oscilloscope and will turn on with the set duty cycle when the temperature rises above 2 degrees more than what is set(default 50% duty cycle). The 2 degree threshold accounts for noise and instability of the temperature sensor, so the output is not being turned on and off too rapidly. Pressing the button will change the system into fan speed mode. Here the PWM will be on and the duty cycle will be displayed on the screen. Turning the encoder will change the duty cycle. Pressing the button again will switch back into temperature mode, and the PWM output will have the newly set duty cycle.

Issues:

The only main issue with the current state of the project occurs when a fan or motor is connected to the output. The fan causes the displayed temperature to bounce rapidly, and often will cause corrupt data to be sent to the LCD when changing modes or settings, putting the system in a state where it has to be reset. This likely occurs because of feedback to the microcontroller through grounding, or voltage spikes/drops due to the larger current draw from the fan. Another issue is the fact that the temperature sensor is not completely accurate to the average temperature of the room. Its reading can rapidly drop or rise simply from air flow in the room. In still air, it seems to have reasonable accuracy.

Changes to Consider:

The LCD is programmed to run in 4-bit mode to reduce physical connections on the microcontroller. One way to reduce connections is to run the LCD in 8-bit mode, but use a shift register connected to the microcontroller serially. Using the shift register still requires three connections to the LCD, with a net save of one pin, but if time is a critical aspect of the project, this would be a good idea. Running the LCD in 4-bit mode requires two sends per byte, and each send requires a certain time delay. The shift register will allow an 8-bit write with only three pins, meaning data is sent once so only one delay. The current library already has longer delays required to ensure there are no issues, and because this project is not very time sensitive. Also, this implementation uses timer 1 to create 1 millisecond delays, which can be replaced with a precise for loop, or somehow with interrupts and complex time tracking. The easiest way is to just delay the whole program, and disable interrupts until printing is finished, or data can be sent out of order and display garbage.

CONCLUSION:

The thermostat project works well as it is. The issue with the motor causing unwanted feedback could potentially be fixed by isolating the motor from the rest of the circuit maybe with optocouplers with the PWM or a relay on the enable pin of the motor driver. As for the “inaccuracy” of the temperature sensor, this is just the nature of the sensor in free air. Using a higher precision sensor or applying noise filters could help with this issue. The sensor can also be mounted in contact with a surface whose temperature needs to be monitored.

REFERENCES:

Clemson University's ECE 372 Lab Powerpoints
Grayhill 61C Optical Encoder Datasheet
Hitachi HD44780 LCD Driver Datasheet
PIC32 Datasheet
TMP36 Temperature Sensor Datasheet

CIRCUIT DIAGRAM:

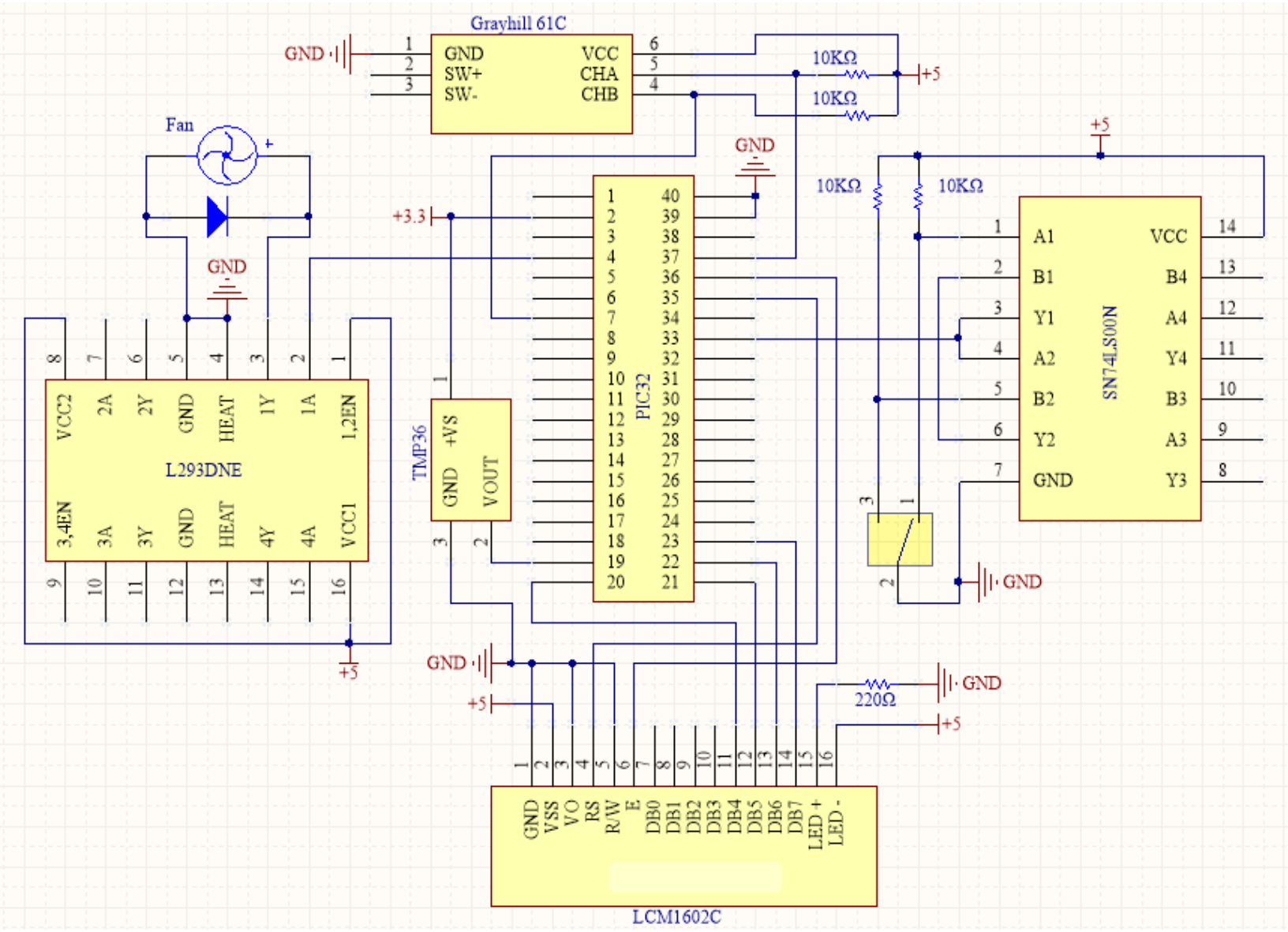

Figure 3: Circuit diagram

CODE:

NOTE: The following code is written in C++, not C. Beginning of new files are marked in bold.

main.cpp

```
#include <plib.h>
#include "LCD_Interface.h"

#define TEMP 0
#define FAN 1
#define SET 2
#define FAN_SPEED OC3RS

int mode = TEMP;
int temp = 70;
int setTemp = 70;
int fanSpeed = 50;
LCD_Interface LCD;

void init();
void initLCD();
void sampleTemp();
void updateLCD(int val, int type);
void fan();

extern "C"{ //C++ apparently doesn't like interrupts
  void __ISR(3,ipl1) encoderA(){
    if(INTCONbits.INT0EP == !PORTCbits.RC8){ //Encoder clockwise turn
      if(mode == TEMP){
        setTemp++;
        if(setTemp > 999){
          setTemp = 999;
        }
      }
      else if(mode == FAN){
        fanSpeed++;
        if(fanSpeed > 100){
          fanSpeed = 100;
        }
      }
    }
    else{ //Encoder counterclockwise turn
      if(mode == TEMP){
        setTemp--;
        if(setTemp < 0){
          setTemp = 0;
        }
      }
      else if(mode == FAN){
        fanSpeed--;
        if(fanSpeed < 0){
          fanSpeed = 0;
        }
      }
    }
    if(INTCONbits.INT0EP){
      INTCONbits.INT0EP = 0;
    }
    else{
      INTCONbits.INT0EP = 1;
    }

    if(mode == TEMP){
      updateLCD(setTemp, SET);
```

```
    }
    else if(mode == FAN){
      updateLCD(fanSpeed, FAN);
      FAN_SPEED = fanSpeed;
    }
    IFS0bits.INT0IF = 0;
  }

  void __ISR(15,ipl2) encoderB(){
    if(INTCONbits.INT3EP == PORTBbits.RB7){ //Encoder clockwise turn
      if(mode == TEMP){
        setTemp++;
        if(setTemp > 999){
          setTemp = 999;
        }
      }
      else if(mode == FAN){
        fanSpeed++;
        if(fanSpeed > 100){
          fanSpeed = 100;
        }
      }
    }
    else{ //Encoder counterclockwise turn
      if(mode == TEMP){
        setTemp--;
        if(setTemp < 0){
          setTemp = 0;
        }
      }
      else if(mode == FAN){
        fanSpeed--;
        if(fanSpeed < 0){
          fanSpeed = 0;
        }
      }
    }
    if(INTCONbits.INT3EP){
      INTCONbits.INT3EP = 0;
    }
    else{
      INTCONbits.INT3EP = 1;
    }
    if(mode == TEMP){
      updateLCD(setTemp, SET);
    }
    else if(mode == FAN){
      updateLCD(fanSpeed, FAN);
      FAN_SPEED = fanSpeed;
    }
    IFS0bits.INT3IF = 0;
  }

  void __ISR(7, ipl3) switchMode(){
    if(mode == TEMP){
      mode = FAN;
    }
    else if(mode == FAN){
      mode = TEMP;
    }
    initLCD();
    IFS0bits.INT1IF = 0;
  }
}

int main(){
```

```
    init();
    initLCD();

    while(1){
        if(mode == TEMP){
            sampleTemp();
            fan();
        }
    }
    return 0;
}

void initLCD(){
    //Sets initial display text. ß is ASCII equivalent for degrees symbol on LCD
    IEC0bits.INT0IE = 0;
    IEC0bits.INT1IE = 0;
    IEC0bits.INT3IE = 0;
    LCD.clear();
    if(mode == TEMP){
        LCD.moveCursor(0,0);
        LCD.print("Temp: ");
        updateLCD(temp, TEMP);
        LCD.print("ßF");
        LCD.moveCursor(1,0);
        LCD.print("Set Temp: ");
        updateLCD(setTemp, SET);
        LCD.print("ßF");
    }
    if(mode == FAN){
        LCD.moveCursor(0,0);
        LCD.print("Fan Speed:");
        updateLCD(fanSpeed, FAN);
        LCD.print("%");
    }
    IEC0bits.INT0IE = 1;
    IEC0bits.INT1IE = 1;
    IEC0bits.INT3IE = 1;
}

void init(){
    INTEnableSystemMultiVectoredInt();

    //Configure optical encoder and interrupts
    TRISCbits.TRISC8 = 1;
    TRISBbits.TRISB7 = 1;
    PPSInput(2, INT3, RPC8);
    INTCONbits.INT0EP = 0;
    IEC0bits.INT0IE = 1;
    IFS0bits.INT0IF = 0;
    IPC0bits.INT0IP = 1;
    INTCONbits.INT3EP = 1;
    IPC3bits.INT3IP = 2;
    IFS0bits.INT3IF = 0;
    IEC0bits.INT3IE = 1;

    //Configure ADC
    TRISAbits.TRISA1 = 1;
    ANSELAbits.ANSA1 = 1;
    AD1CHSbits.CH0SA = 1;
    AD1CON1bits.FORM = 0;
    AD1CON1bits.SSRC = 7;
    AD1CON1bits.ASAM = 0;
    AD1CON2bits.VCFG = 0;
    AD1CON2bits.CSCNA = 0;
    AD1CON2bits.SMPI = 0;
    AD1CON2bits.BUFM = 0;
```

```
  AD1CON2bits.ALTS = 0;
  AD1CON3bits.ADRC = 0;
  AD1CON3bits.SAMC = 16;
  AD1CON3bits.ADCS = 1;
  AD1CON1bits.ON = 1;

  //Configure PWM for fan
  T2CONbits.ON = 0;
  T2CONbits.T32 = 0;
  TMR2 = 0;
  T2CONbits.TCKPS = 0;
  T2CONbits.TCS = 0;
  PR2 = 100;
  T2CONbits.ON = 1;
  TRISBbits.TRISB9 = 0;
  LATBbits.LATB9 = 0;
  PPSOutput(4, RPB9, OC3);
  OC3CONbits.ON = 0;
  OC3CONbits.OC32 = 0;
  OC3CONbits.OCTSEL = 0;
  OC3CONbits.OCM = 0b110;
  FAN_SPEED = fanSpeed;
  OC3CONbits.ON = 1;

  //Switch modes push button interrupt
  TRISCbits.TRISC4 = 1;
  PPSInput(4 ,INT1, RPC4);
  INTCONbits.INT1EP = 1;
  IPC1bits.INT1IP = 3;
  IFS0bits.INT1IF = 0;
  IEC0bits.INT1IE = 1;
}

void updateLCD(int val, int type){
  /*
    Turning the encoder fast will result in an interrupt while the LCD is printing and the interrupt will
      trigger another print before the first can finish. This will send corrupt data to the LCD.
   */
  IEC0bits.INT0IE = 0;
  IEC0bits.INT1IE = 0;
  IEC0bits.INT3IE = 0;
  if(type == TEMP){
    LCD.moveCursor(0,6);
  }
  else if(type == SET){
    LCD.moveCursor(1,10);
  }
  else if(type == FAN){
    LCD.moveCursor(1,4);
  }

  LCD.print(val);
  //Override previous digits on LCD only when number of digits change
  if(val == 99 || val == 9 || val == 100 || val == 10){
    if(type == TEMP || type == SET){
      LCD.print("ßF ");
    }
    else if(type == FAN){
      LCD.print("%  ");
    }
  }
  IEC0bits.INT0IE = 1;
  IEC0bits.INT1IE = 1;
  IEC0bits.INT3IE = 1;
}
```

```
void sampleTemp(){
   /*
      1 Poll for ADC conversion
      2 Convert sample to voltage, then to degrees C, then to degrees F
    */
   AD1CON1bits.SAMP = 1;
   while(!AD1CON1bits.DONE); //1
   temp = ADC1BUF0;
   temp = (((temp  * 3.3 * 100) / 1024) - 50) * 1.8 + 32; //2
   if(temp < 0){
      temp = 0;   //Just avoiding negative temperatures, too much work and likely not going to happen
   }else if(temp > 999){
      temp = 999; //Not going to happen either
   }
   updateLCD(temp, TEMP);
   AD1CON1bits.DONE = 0;
}

void fan(){
   //Turn fan on when temperature is greater than set temp + noise error threshold
   if(temp >= setTemp + 2){
      FAN_SPEED = fanSpeed;
   }
   else{
      FAN_SPEED = 0;
   }
}
```

LCD_Interface.h

```
/*
 * File:   LCD_Interface.h
 * Author: Travis McCormick
 *
 * Created on April 16, 2019, 3:55 PM
 *
 * Description: This is a class used for interfacing Hitachi HD44780 chipset 2x16 LCDs
 *           with the PIC32 microcontroller. This library is intended to interface
 *           with the LCD in 4-bit mode with Port B on the microcontroller.
 */

#ifndef LCD_INTERFACE_H
#define      LCD_INTERFACE_H

//Use lower 6 bits of Port B to control LCD
#define PORT_MASK 0xFF80
#define LCD_PORT TRISB
#define LCD_PINS ANSELB
#define LCD_DATA LATB
#define LCD_ENABLE LATBbits.LATB6
#define LCD_WRITE_RS LATBbits.LATB5
#define LCD_READ_RS LATBbits.LATB5
#define TIMER T1CONbits
#define COUNT TMR1
#define PER PR1
#define ROWS 2
#define COLS 16

class LCD_Interface {
public:
   LCD_Interface();
   void print(const char str[]);
   void print(int num);
```

```
    void moveCursor(int row, int col);
    void cursorOff();
    void cursorOn();
    void clear();

private:
    void sendData();
    void delay(int ms);
    void write(int data);

};

#endif
```

LCD_Interface.cpp

```
/*
 * File:   LCD_Interface.cpp
 * Author: Travis
 *
 * Created on April 16, 2019, 3:55 PM
 */

#include "LCD_Interface.h"
#include <plib.h>

LCD_Interface::LCD_Interface() {
    //Initialize delay timer
    TIMER.ON = 0;
    TIMER.TCKPS = 1;
    TIMER.TCS = 0;
    COUNT = 0;
    PER = 60;

    //Configure lower 7 bits of port
    LCD_PORT &= PORT_MASK;
    LCD_PINS &= PORT_MASK;
    LCD_DATA &= PORT_MASK;
    LCD_WRITE_RS = 0;

    //Requires wait at least 15 ms after Vcc gets to 4.5V
    delay(20);
    //Data sequence to set to LCD to 4-bit mode
    LCD_DATA = (LCD_DATA & PORT_MASK) | 0x03;
    sendData();
    delay(5);
    sendData();
    delay(5);
    sendData();
    delay(1);
    LCD_DATA = (LCD_DATA & PORT_MASK) | 0x02;
    sendData();
    delay(1);
    write(0x02);

    //Set LCD settings: Font size, number of lines, cursor, etc.
    write(0x08); //2 lines and 5x8 character size
    write(0x08); //Turn display off
    write(0x01); //Clear display (takes more time)
    delay(10);
    write(0x06); //Allow cursor to move with prints
    write(0x0C); //Turn display on
}
```

```
void LCD_Interface::delay(int ms){
  //Time delay in milliseconds
  for(int i = 0; i < ms; i++){
    COUNT = 0;
    TIMER.ON = 1;
    while(COUNT < PER);
    TIMER.ON = 0;
  }
}

void LCD_Interface::sendData(){
  //Create pulse for LCD driver's enable bit
  LCD_ENABLE = 1;
  delay(1);
  LCD_ENABLE = 0;
  delay(1);
}

void LCD_Interface::write(int data){
  int _rs = LCD_READ_RS;

  /*
    Send upper 4 bits, then send lower 4 bits

    1 Zero out 7 bits for LCD
    2 Copy 4 bits of data to lower 4 bits of port
    3 Write RS bit before sending data
   */
  LCD_DATA = (LCD_DATA & PORT_MASK); //1
  LCD_DATA = (LCD_DATA & PORT_MASK) | ((data >> 4) & 0x0F); //2
  LCD_WRITE_RS = _rs; //3
  sendData();
  LCD_DATA = (LCD_DATA & PORT_MASK); //1
  LCD_DATA = (LCD_DATA & PORT_MASK) | (data & 0x0F); //2
  LCD_WRITE_RS = _rs; //3
  sendData();
}

void LCD_Interface::print(const char str[]){
  /*
    RS = 1 means data is sent
    RS = 0 means LCD commands are sent
   */
  LCD_WRITE_RS = 1;
  while(*str != '\0'){
    write(*str);
    str++;
  }
  LCD_WRITE_RS = 0;
}

void LCD_Interface::print(int num){
  LCD_WRITE_RS = 1;
  /*
    1 Count how many digits are in the number
    2 Print each individual digit
   */
  //1
  int digits = 0;
  int temp = num;
  while(temp > 0){
    digits++;
    temp /= 10;
  }
  //2
```

```
    while(digits > 1){
        int mult = 1;
        for(int i = 1; i < digits; i++){
            mult *= 10;
        }
        write((num / (mult)) + '0');
        num %= (mult);
        digits--;
    }
    write(num + '0');
    LCD_WRITE_RS = 0;
}

void LCD_Interface::clear(){
    //Clear display
    write(0x01);
    delay(10);
}

void LCD_Interface::cursorOn(){
    //Turns on display with blinking cursor
    write(0x0F);
}

void LCD_Interface::cursorOff(){
    //Turns off blinking cursor
    write(0x0C);
}

void LCD_Interface::moveCursor(int row, int col){
    //Makes sure cursor coordinate is within boundaries
    if(row >= ROWS){
        row = ROWS - 1;
    }
    if(col >= COLS){
        col = COLS - 1;
    }
    if(row < 0){
        row = 0;
    }
    if(col < 0){
        col = 0;
    }

    /*
        First row = 0x00 DD RAM offset
        Second row = 0x40 DD RAM offset

        Set row offset bit, then set column offset
        Upper bit tells it to write DD RAM address (cursor position);
     */
    int pos = (row << 6) | col;
    write(0x80 | pos);
    delay(10);
}
```