Semester Project

Digital Thermometer

Team: Joel Mendez, Jean Charles Michelet
Course Section: EECE 321-02
Instructor: Frantz Homicile

Overview

The main objective of this project was to design and create a digital thermometer using a PIC microcontroller and LM35 temperature sensor. The resulting temperature would be outputted on a 16x2 LCD. The PIC16F877A microcontroller was used for this project. Accompanying the PIC microcontroller, was the PICKIT 3 IN-CIRCUIT DEBUGGER, which allowed for the flashing of the program onto the PIC microcontroller. The code for this project was written in the C-programming language, using the MPLAB IDE and XC8 compiler. The MPLAB IPE was used to flash the hex file onto the PIC microcontroller. This project was our introduction to using the Analog-to-Digital converter (ADC) module on the PIC16F877A microcontroller.

Component Overview

MPLAB Software

MPLAB is an integrated development environment created by Microchip Technology, designed specifically for the development of embedded systems using PIC controllers. MPLAB provides the XC8 compiler, which will be used to convert our written C code to a readable hex file for our microcontroller. The MPLAB IDE offers many features that are incredibly helpful when developing embedded system applications.

PIC16F877A Microcontroller & PICKIT 3

The PIC16F877A is a 40-pin microcontroller that features 256 bytes of EEPROM memory, self-programming, and 8 channels of a 10-bit Analog-to-Digital converter. The PIC16F877A microcontroller exclusively operates via Flash memory. The PICKIT 3 IN-CIRCUIT DEBUGGER can be used to flash programs onto the PIC16F877A microcontroller. The PICKIT 3 offers a USB interface at up to 12 Mbits/s, real-time execution, voltage source, and up to 512K byte flash with the Programmer-to-Go feature. The PIC16F877A microcontroller and PICKIT 3 IN-CIRCUIT DEBUGGER are a great combination as they are affordable and offer an extensive range of features.

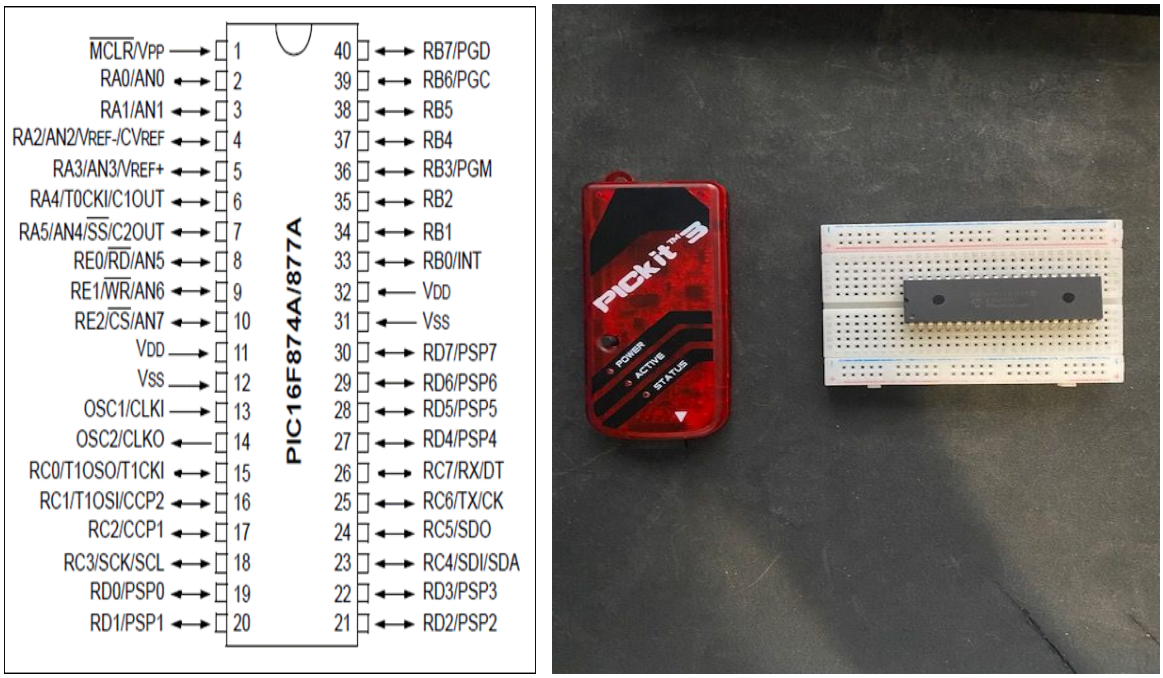

Figure 1: PIC16F877A Microcontroller and PICKIT 3

ADC in PIC Microcontroller PIC16F877A

There are many types of ADC modules available, with each one having its own benefits. The common ADCs include flash, successive approximation, and sigma-delta. The PIC16F877A uses the Successive approximation ADC, known as SAR. The SAR functions with the help of a comparator, accompanied by some logic conversions. The SAR uses a reference voltage and compares the input voltage with the reference voltage. This will be processed as a digital output. The speed at which these comparisons take place is entirely dependent on the clock frequency at which the PIC is operating. The PIC microcontroller that we are using has a 10-bit 8-channel ADC, which means our output value of the ADC will be between 0 and 10^(24). Pins labeled ANx can read the analog voltage supplied by the LM35. It is important that we specify in the code, which pin the analog input is coming from. In our case, we will be using AN4.

LM35 Temperature Sensor

The LM35 is a 3-pin local analog temperature sensor. The operating range for the LM35 is between 4 and 20 Volts, drawing in a max current of about 60 µA. The LM35 requires three connections: Vs, GND, and Vout. Vs and GND are for the voltage supply, while the Vout pin is responsible for reading the analog output. The LM35 is designed to work from 2°C to 150°C, with temperatures at 25°C being within 0.5°C of accuracy. The relationship between the voltage outputted by Vout and the temperature is linear. For every 10mV in output, that equals 1°C in temperature. Because this signal is analog, we must use the ADC module in our microcontroller to make sense of the signal.

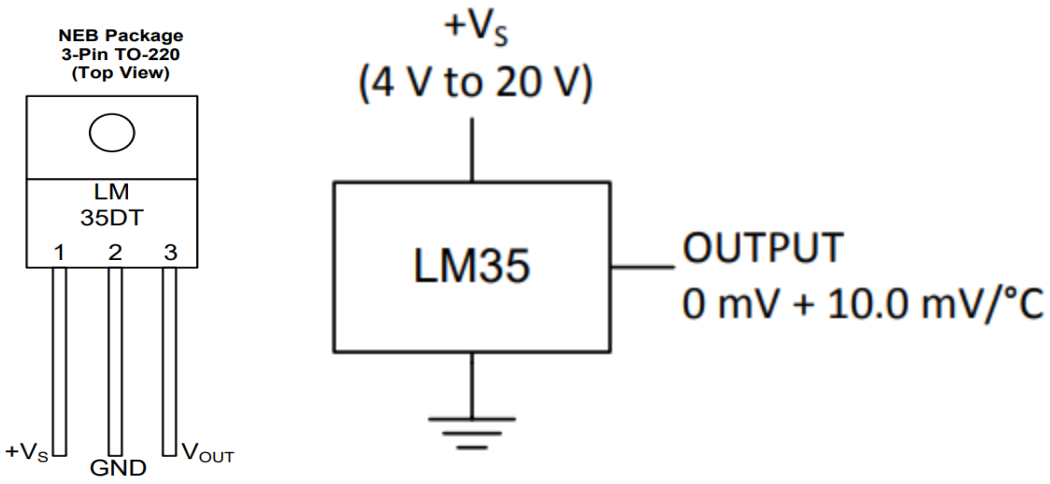

Figure 2: LM35 Schematic

16 x 2 LCD Display Module

The LCD module we used contains 16 columns and 2 rows in the display. Each display element can display 1 character or number. Each display element is simply a 5 by 8 dot matrix display. To display the character you want, you would set the corresponding bits in the dot matrix display to be high. There are 16 pins on the LCD that allows the displays to function properly: Vss,VDD,Vo,RS,RW,E,D0-D7,A, and K. Vss is the ground pin of the display, used to connect the GND terminal of the microcontroller. Vcc is the source pin, used to connect the display to a 5V power supply. Vo is the control pin, which is connected to the potentiometer, used to change the contrast on the display. RS is the pin that allows the toggle between command and data mode. RW is the pin that allows the toggle between read and write operations. E is the pin that enables the read/write operations. D0-D7 are the pins that allow data to be sent to the display. The last two pins are responsible for the backlight of the LCD, A being connected to +5V and K being connected to GND.

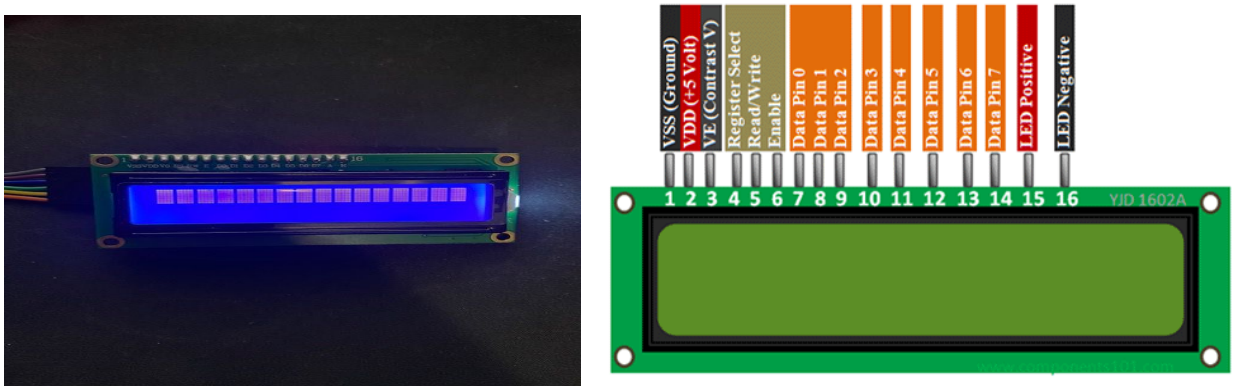

Figure 3: 16 x 2 LCD Display

Procedure

Code Explanation

After setting up the MPLAB IDE, we began writing the code for our microcontroller. The code that we use for the program will allow us to interface with the LCD and ACD modules in our project. The first part of the code defines the mapping we use between the LCD and the microcontroller. For example, because RS on the LCD is connected directly to RD2 on the microcontroller, we write #define RS RD2. We defined the clock frequency to be a value of 20000000 (20Mhz).

```
#define _XTAL_FREQ 20000000

#define RS RD2

#define EN RD3

#define D4 RD4

#define D5 RD5

#define D6 RD6

#define D7 RD7
```

Figure 4: Pin Definitions

The next part of the code involves the inclusion of the header file <xc.h>, which allows us to access the functions for PIC microcontrollers. The proceeding lines of code give additional information to the compiler such as the configuration of oscillator bits.

```
#include <xc.h>

#pragma config FOSC = HS        // Oscillator Selection bits (HS oscillator)

#pragma config WDTE = OFF       // Watchdog Timer Enable bit (WDT disabled)

#pragma config PWRTE = ON       // Power-up Timer Enable bit (PWRT enabled)

#pragma config BOREN = ON       // Brown-out Reset Enable bit (BOR enabled)

#pragma config LVP = OFF        // Low-Voltage (Single-Supply) In-Circuit Serial Programming Enable bit (RB3 is digital I/O, HV on MCLR must be used for programming)

#pragma config CPD = OFF        // Data EEPROM Memory Code Protection bit (Data EEPROM code protection off)

#pragma config WRT = OFF        // Flash Program Memory Write Enable bits (Write protection off; all program memory may be written to by EECON control)

#pragma config CP = OFF         // Flash Program Memory Code Protection bit (Code protection off)
```

Figure 5: Header Files & Configurations

The next part of the code involves the declaration of the functions that we utilized in the main function. These functions allowed us to interact with the LCD and ADC modules. The first two function declarations were for the ADC module. The function ADC_Initialize turned on the ADC module and selected a reference voltage. The function ADC_Read served the purpose of returning the analog signal.

```
void ADC_Initialize()

{

  ADCON0 = 0b01000001; //ADC ON and Fosc/16 is selected

  ADCON1 = 0b11000000; // Internal reference voltage is selected

}


unsigned int ADC_Read(unsigned char channel)

{

  ADCON0 &= 0x11000101; //Clearing the Channel Selection Bits

  ADCON0 |= channel<<3; //Setting the required Bits

  __delay_ms(2); //Acquisition time to charge hold capacitor

  GO_nDONE = 1; //Initializes A/D Conversion

  while(GO_nDONE); //Wait for A/D Conversion to complete

  return ((ADRESH<<8)+ADRESL); //Returns Result

}
```

Figure 6: ADC Module Functions

The next set of functions was created to interact between the LCD and the microcontroller. These series of functions perform various operations such as setting the bits on the screen, setting the cursor, initializing the LCD, and clearing the screen. These declared functions are too long to show in this portion of the project, so will be present in the appendix of this report.

Now that we are done defining pins, including the header files, and declaring functions, we can begin writing the main portion of our code. First, we defined three float variables: adc, volt, and temp. These variables will be used in the computational aspects of our code. We also defined five integer values: c1,c2,c3,c4, and temp1, which would also be used for computations in our code. After that we called the ADC_Initialize and Lcd_Start functions, which initialized the ADC and LCD modules.

```
int main()

{

    float adc;

    float volt, temp;

    int c1, c2, c3, c4, temp1;

    ADC_Initialize();


    unsigned int a;

    TRISD = 0x00;

    Lcd_Start();
```

Figure 7: Main Function

The last section of our code is the main while loop. In this section of the code, we are going to read the analog signal supplied by the LM35 and immediately convert it to temperature. The first section of the while loop deals with the calculations of those numbers. We retrieved the analog voltage using the ADC_Read function and set it equal to our adc variable. To convert this number into a voltage, we then multiplied by 4.88281. After dividing this result by 10 and setting temp1 to 100 times that result, we now have the temperature reading. However, because the LCD

display requires bit-by-bit manipulation of each element on the screen, we have to get the individual digits of our temperature. Using the Modulo operation, we can get each digit individually for our temperature.

```
while(1)

{

    adc = (ADC_Read(4)); // Reading ADC values

    volt = adc*4.88281; // Convert it into the voltage

    temp=volt/10.0;  // Getting the temperature values

    temp1 = temp*100;


    c1 = (temp1/1000)%10;

    c2 = (temp1/100)%10;

    c3 = (temp1/10)%10;

    c4 = (temp1/1)%10;
```

Figure 8: Getting Temperature in While Loop

This part of the while loop involves getting the temperature output and displaying it on the screen. The code involves setting the clearing of the screen using Lcd_Clear, setting the cursor using Lcd_Set_Cursor, and using Lcd_Print_String & Lcd_Print_Char to print the elements to the screen. Each iteration of this loop was delayed by 3 seconds, to make the temperature readable on screen.

```
    Lcd_Clear();

    Lcd_Set_Cursor(1,1);

    Lcd_Print_String("LM35");

    Lcd_Set_Cursor(2,1);

    Lcd_Print_String("TEMP:");

    Lcd_Set_Cursor(2,5);

    Lcd_Print_Char(c1+'0');

    Lcd_Print_Char(c2+'0');

    Lcd_Print_String(".");

    Lcd_Print_Char(c3+'0');

    Lcd_Print_Char(c4+'0');

    Lcd_Print_Char(0xDF);

    Lcd_Print_String("C");

    __delay_ms(3000);
```

Figure 9: Displaying Temperature in While Loop

Flashing PIC16F877A Microcontroller

Now that we have completed the code for the project, we have to flash this code to the microcontroller. Before we could begin the flashing process, we have to set up the circuit that will allow us to flash the program onto the microcontroller. The PICKIT 3 has 6 pins to connect to your microcontroller. Pin 1 is known as the Master Clear/Reset pin (MCLR). Pin 2 is VDD, which is where we connect our +5V source. Pin 3 is VSS, which is where we connect our ground source. Pin 4 is known as Program Data (PGD), which is responsible for data transfer. Pin 5 is

known as Program Clock (PGC), which we connect to the PGC pin on the microcontroller. The last pin is the LVP pin, which is reserved for future connections and was not used during this project.

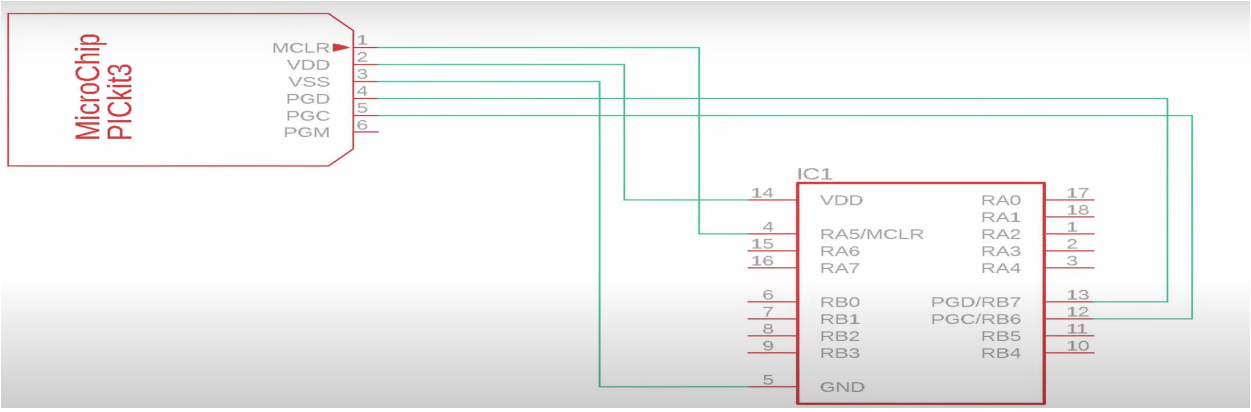

Figure 10: PICKIT 3 Schematic

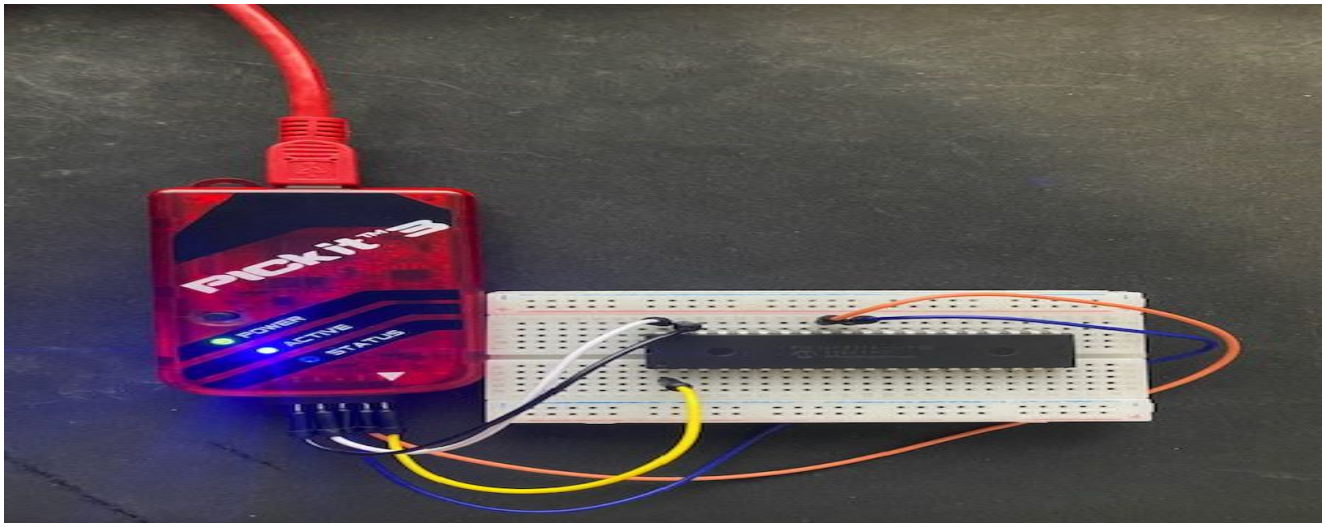

Figure 10: PICKIT 3 & PIC16F877A Flashing Circuit

After setting up the circuit, we were now ready to begin the flashing process. We used the MPLAB IPE to flash the microcontroller. After loading the hex file into the program and following a few intermediate steps, we were able to successfully flash the PIC microcontroller.

```
**************************************************

Connecting to MPLAB PICkit 3...

Currently loaded firmware on PICkit 3
Firmware Suite Version.....01.56.09
Firmware type..............Midrange
Programmer to target power is enabled - VDD = 4.750000 volts.
Target device PIC16F877A found.
Device Revision ID = 8
Loading code from C:\Users\joelm\MPLABXProjects\Project1.X\ProjectTest.X\dist\default\production\ProjectTest.X.production.hex...
2022-12-09 19:53:19 -0500 - Hex file loaded successfully.
2022-12-09 19:53:23 -0500 - Programming...


Device Erased...

Programming...

The following memory area(s) will be programmed:
program memory: start address = 0x0, end address = 0x7ff
configuration memory
Programming/Verify complete
2022-12-09 19:53:28 -0500 - Programming complete
***   Hold In Reset mode is enabled   ***
```

Figure 11: Flashing PIC16F877A using MPLAB IPE

Constructing the Circuit

Now that we have the program on the PIC microcontroller, it is time we create a working circuit to bring the project to life. Figure 12 shows the schematic of the circuit we are constructing for this project.

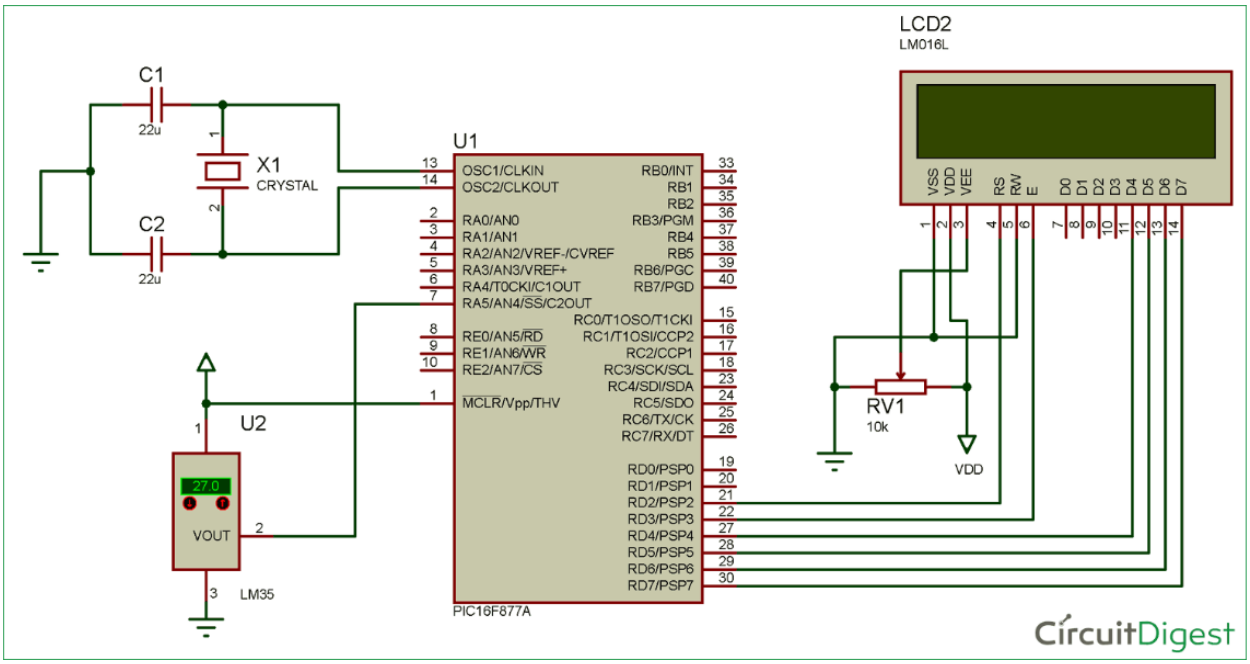

Figure 12: Circuit Schematic

Let's break the circuit down into smaller pieces. The top left portion deals with the microcontroller clock signal. The PIC16F877A microcontroller does not come with an internal oscillator, so we must supply one ourselves. For this project, we used a 20 Mhz Crystal

Oscillator. The microcontroller clock signal will govern the conversion rate of the analog-to-digital operations, with the clock speed determining the rate at which that signal can be sampled.

The bottom left portion is the LM35 sensor connections. These connections are relatively simple, with Pin 1 going directly to our +5V source. Pin 2 is connected directly to the Analog read pin on the microcontroller. This is the connection that will transfer the analog signal to the microcontroller for conversion. Finally, Pin 3 is connected directly to ground.

The last part of the circuit is the LCD connections with the microcontroller. Vss and RW are directly connected to ground. Because RW is connected to ground, the write operation will be performed in the LCD. VDD is connected directly to the +5V source. This connection will give power to the LCD screen. VEE is connected directly to the 10K potentiometer, which is in-turn connected to both ground and the +5V source. This connection allows us to adjust the contrast on the screen using the potentiometer. RS and E are connected to RD2 and RD3 on the microcontroller. This connection allows data provided to be stored in the register of the LCD and enables the write operations. Pins D4-D7 are connected to RD4-RD7 on the microcontroller, which allows for the data transfer from the microcontroller to the LCD.

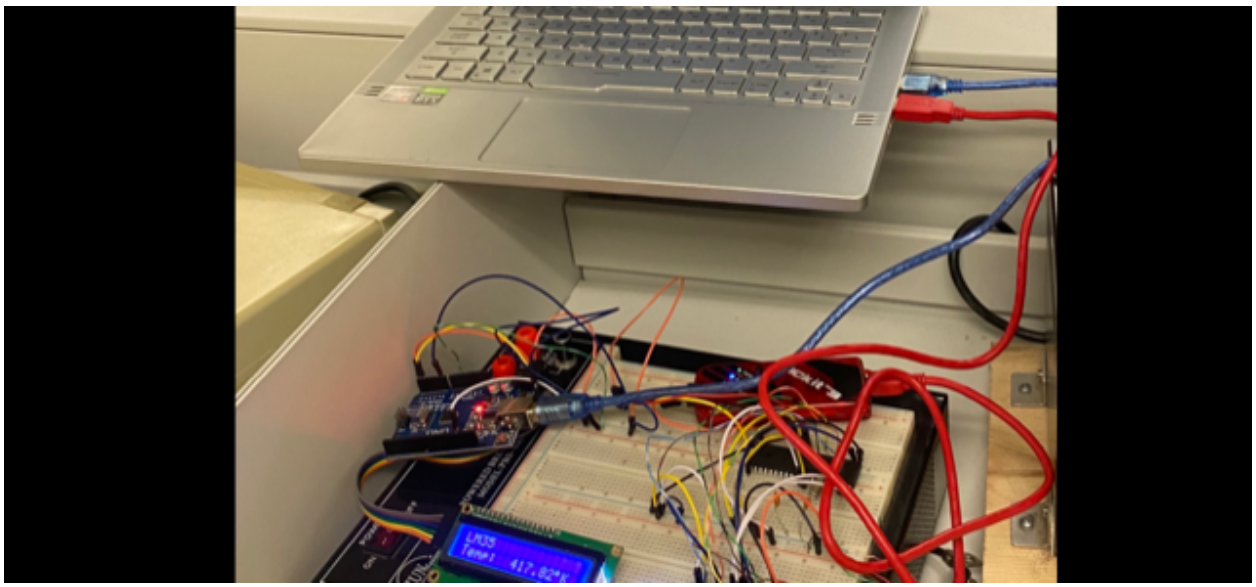

Figure 13: Implementation of Circuit

Results and Conclusion

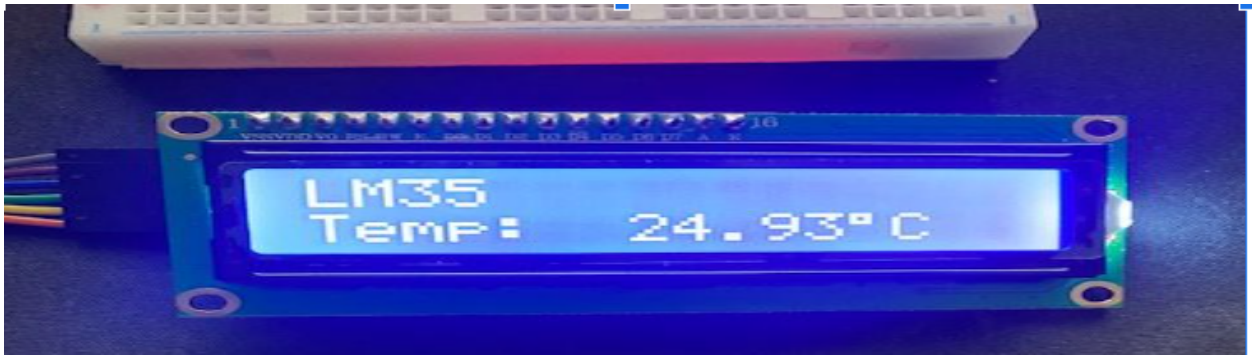

Figure 14: LM35 Temperature Reading on LCD Display

This project was an introduction to interfacing with the LCD and ADC modules. We learned how to utilize the ADC modules on the PIC microcontroller and how to interact with the LCD display. The temperature reading also appears to be accurate when we tested the system. The temperature was reading around 25°C, which is right around room temperature. Overall, this project has been a great learning experience. The concepts learned during this project will certainly be beneficial in future embedded system applications.

Appendix

```
#define _XTAL_FREQ 20000000


#define RS RD2

#define EN RD3

#define D4 RD4

#define D5 RD5

#define D6 RD6

#define D7 RD7


#include <xc.h>


#pragma config FOSC = HS        // Oscillator Selection bits (HS
oscillator)

#pragma config WDTE = OFF       // Watchdog Timer Enable bit (WDT
disabled)

#pragma config PWRTE = ON       // Power-up Timer Enable bit (PWRT
enabled)

#pragma config BOREN = ON       // Brown-out Reset Enable bit (BOR
enabled)

#pragma config LVP = OFF        // Low-Voltage (Single-Supply) In-Circuit
Serial Programming Enable bit (RB3 is digital I/O, HV on MCLR must be used
for programming)

#pragma config CPD = OFF        // Data EEPROM Memory Code Protection bit
(Data EEPROM code protection off)
```

```
#pragma config WRT = OFF          // Flash Program Memory Write Enable bits (Write protection off; all program memory may be written to by EECON control)

#pragma config CP = OFF           // Flash Program Memory Code Protection bit (Code protection off)


void ADC_Initialize()

{

  ADCON0 = 0b01000001; //ADC ON and Fosc/16 is selected

  ADCON1 = 0b11000000; // Internal reference voltage is selected

}


unsigned int ADC_Read(unsigned char channel)

{

  ADCON0 &= 0x11000101; //Clearing the Channel Selection Bits

  ADCON0 |= channel<<3; //Setting the required Bits

  __delay_ms(2); //Acquisition time to charge hold capacitor

  GO_nDONE = 1; //Initializes A/D Conversion

  while(GO_nDONE); //Wait for A/D Conversion to complete

  return ((ADRESH<<8)+ADRESL); //Returns Result

}



//LCD Functions Developed by Circuit Digest.
```

```
void Lcd_SetBit(char data_bit) //Based on the Hex value Set the Bits of
the Data Lines

{

    if(data_bit& 1)

        D4 = 1;

    else

        D4 = 0;


    if(data_bit& 2)

        D5 = 1;

    else

        D5 = 0;


    if(data_bit& 4)

        D6 = 1;

    else

        D6 = 0;


    if(data_bit& 8)

        D7 = 1;

    else

        D7 = 0;
```

```
}

void Lcd_Cmd(char a)

{

    RS = 0;

    Lcd_SetBit(a); //Incoming Hex value

    EN  = 1;

         __delay_ms(4);

        EN  = 0;

}

Lcd_Clear()

{

    Lcd_Cmd(0); //Clear the LCD

    Lcd_Cmd(1); //Move the curser to first position

}

void Lcd_Set_Cursor(char a, char b)

{

    char temp,z,y;

    if(a== 1)
```

```
    {
      temp = 0x80 + b - 1; //80H is used to move the curser
        z = temp>>4; //Lower 8-bits
        y = temp & 0x0F; //Upper 8-bits
        Lcd_Cmd(z); //Set Row
        Lcd_Cmd(y); //Set Column
    }
    else if(a== 2)
    {
        temp = 0xC0 + b - 1;
        z = temp>>4; //Lower 8-bits
        y = temp & 0x0F; //Upper 8-bits
        Lcd_Cmd(z); //Set Row
        Lcd_Cmd(y); //Set Column
    }
}

void Lcd_Start()
{
  Lcd_SetBit(0x00);
  for(int i=1065244; i<=0; i--)  NOP();
```

```
  Lcd_Cmd(0x03);

      __delay_ms(5);

  Lcd_Cmd(0x03);

      __delay_ms(11);

  Lcd_Cmd(0x03);

  Lcd_Cmd(0x02); //02H is used for Return home -> Clears the RAM and
initializes the LCD

  Lcd_Cmd(0x02); //02H is used for Return home -> Clears the RAM and
initializes the LCD

  Lcd_Cmd(0x08); //Select Row 1

  Lcd_Cmd(0x00); //Clear Row 1 Display

  Lcd_Cmd(0x0C); //Select Row 2

  Lcd_Cmd(0x00); //Clear Row 2 Display

  Lcd_Cmd(0x06);

}


void Lcd_Print_Char(char data)  //Send 8-bits through 4-bit mode

{

   char Lower_Nibble,Upper_Nibble;

   Lower_Nibble = data&0x0F;

   Upper_Nibble = data&0xF0;
```

```
    RS = 1;             // => RS = 1

    Lcd_SetBit(Upper_Nibble>>4);               //Send upper half by shifting
by 4

    EN = 1;

    for(int i=2130483; i<=0; i--)  NOP();

    EN = 0;

    Lcd_SetBit(Lower_Nibble); //Send Lower half

    EN = 1;

    for(int i=2130483; i<=0; i--)  NOP();

    EN = 0;

}


void Lcd_Print_String(char *a)

{

    int i;

    for(i=0;a[i]!='\0';i++)

        Lcd_Print_Char(a[i]);  //Split the string using pointers and call
the Char function

}



int main()

{
```

```
    float adc;
    float volt, temp;
    int c1, c2, c3, c4, temp1;
    ADC_Initialize();

    unsigned int a;
    TRISD = 0x00;
    Lcd_Start();
    while(1)
    {
        adc = (ADC_Read(4)); // Reading ADC values
        volt = adc*4.88281; // Convert it into the voltage
        temp=volt/10.0;  // Getting the temperature values
        temp1 = temp*100;


        c1 = (temp1/1000)%10;
        c2 = (temp1/100)%10;
        c3 = (temp1/10)%10;
        c4 = (temp1/1)%10;
```

```
        Lcd_Clear();

        Lcd_Set_Cursor(1,1);

        Lcd_Print_String("LM35");

        Lcd_Set_Cursor(2,1);

        Lcd_Print_String("TEMP:");

        Lcd_Set_Cursor(2,5);

        Lcd_Print_Char(c1+'0');

        Lcd_Print_Char(c2+'0');

        Lcd_Print_String(".");

        Lcd_Print_Char(c3+'0');

        Lcd_Print_Char(c4+'0');

        Lcd_Print_Char(0xDF);

        Lcd_Print_String("C");

          __delay_ms(3000);

    }

    return 0;

}
```