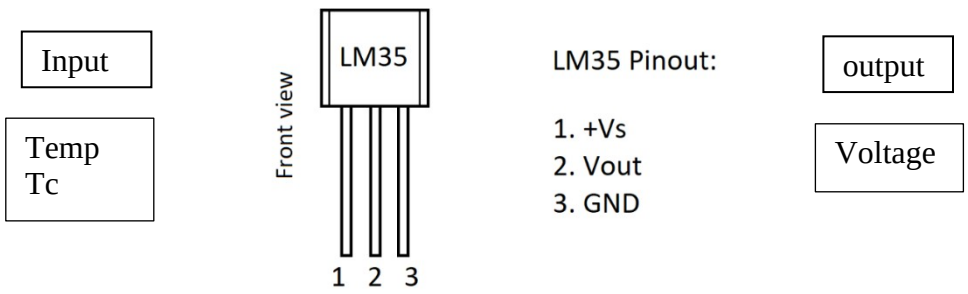
I. Introduction:

The objective of this lab session is to create a digital thermometer that displays temperature readings on the multiplexed 7-segment display of the EasyPIC 7 demo board using the LM35 temperature sensor. The LM35 sensor outputs a voltage increase of 10 mV for every degree Celsius and is designed to gauge temperatures ranging from 0 to 100°C. This session also explores the essential process of Analog-to-Digital Conversion (ADC), which is vital for transforming the analog voltage inputs into digital values that the microcontroller can interpret. Furthermore, the thermometer includes a button that allows users to toggle the display between Celsius and Fahrenheit. Additionally, three LEDs (HI, MED, LO) light up according to set temperature thresholds.

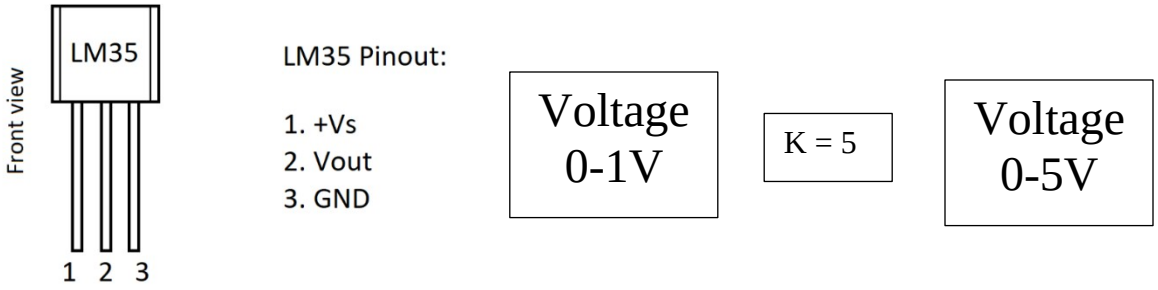
II. Procedure:

- ***Part A (Digital Thermometer with gain):***

In this section, we need to develop a program for a microcontroller to construct a digital thermometer that uses a multiplexed display. The objective is to ensure seamless operation and adjust the thermometer readings so they can be immediately observed on the display. The LM35 sensor measures a physical property, usually temperature, and converts it into an output voltage that corresponds to the temperature detected.



The LM35 sensor is designed to assess temperatures between 0 and 100°C, corresponding to an output voltage span of 0 to 1 V. To interface the output voltage from the LM35 sensor with the ADC in the PIC18 microcontroller, we amplified the analog output voltage by a factor of K = 5. The ADC is configured to handle a voltage range of 0V to 5V, using internal references VDD = 5V and VSS = 0V for the A/D conversion. As a result, adjustments were made to maximize the utilization of the ADC's full-scale range by scaling the LM35 sensor's maximum output of 1V to near the 5V upper limit.



Given that 1°C is equal to 10 mV, it follows that 100°C corresponds to 1000 mV, which is 1V. Thus, the operational characteristics of the LM35 sensor can be summarized as follows:

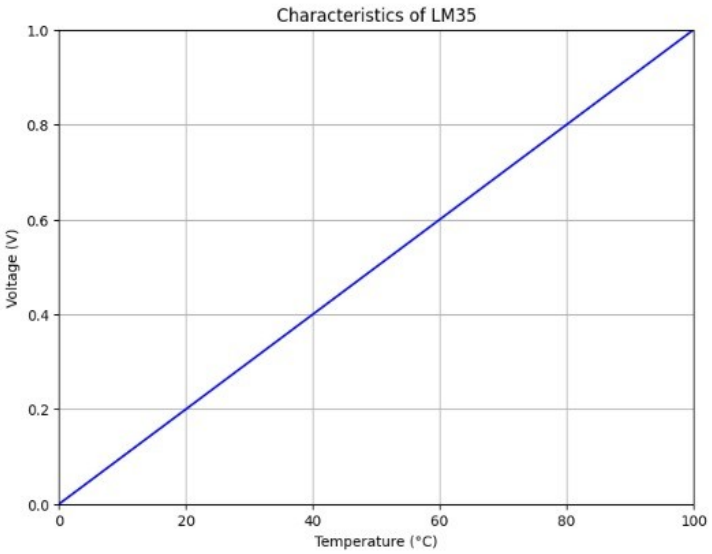


Figure 1: Characteristics of LM35 sensor

Since the interpretation of the LM35 sensor is linear, we got the slope:

$$V = a * Tc$$

where a = $\frac{1}{100}$ = 0.01. Therefore,

$$V = 0.01 * Tc$$

where V is the output voltage of the LM35. However, since the output voltage of the LM35 sensor is multiplied by a gain K = 5, we're left with:

$$5V = 5 * 0.01 * Tc$$

$$\rightarrow 5 * 0.01 * Tc = Vin$$

Where Vin = $\frac{V_{REF}}{2^n} * ADRESH$. Consequently, the following equation is derived:

$$Tc = \frac{100}{256} * ADRESH$$

A. MPLAB Code:

```
#include <p18cxxx.h>
#include <BCDlib.h>

char SScodes[] = {0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x63,0x39};
unsigned char i, Qstate;
unsigned char temp;
enum {degree = 10, C};
char digits[5] = {0, 0, 0, degree, C};

void setup(void);
void measureT(void);
```

```
void main (void) {
   setup();
   while(1) {
      measureT();
   }
}
void setup (void) {
   ANSELD = TRISD = 0x00;
   TRISA &= 0xF0;
   ANSELA &= 0xF0;
   TRISEbits.RE2 = ANSELEbits.ANSE2 = 1;

   ADCON0 = 0b00011101;
   ADCON1 = 0x00;
   ADCON2 = 0b00001001;

   INTCONbits.T0IE = 1;
   INTCONbits.GIE = 1;

   T0CON = 0b11010100;
   TMR0L = 256 - 125;
   i = 1;
   Qstate = 0b00001000;
}

#pragma code ISR = 0x0008
#pragma interrupt ISR

void ISR (void) {
   INTCONbits.T0IF = 0;
   TMR0L = 256 - 125;
   PORTD = 0x00;
   PORTA = Qstate;
   PORTD = SScodes[digits[i++]];
   Qstate >>= 1;

   if (i == 5) {
      i = 1;
      Qstate = 0b00001000;
    }
}

void measureT(void) {
   ADCON0bits.GO_DONE = 1;
   while(ADCON0bits.NOT_DONE);
```

```
  temp = (((int) ADRESH * 100) /256);
  Bin2Bcd (temp, digits);
}
```

Explanation of the code:

1) Global variables used:

1. SScodes[]: This array stores the 7-segment display codes for digits ranging from 0 to 9. Additionally, the codes for '°' (0x63) and 'C' (0x39) are included to enable their display on the 7-segment display. Given that only small integer values are required, the data type chosen for this variable is char, which typically represents 1 byte (8 bits) of data storage and sufficiently accommodates the necessary values.
2. i, Qstate, temp: These global variables are crucial for managing various aspects of the program. They help control the flow of the program, count interrupts, manage display states, and store the temperature value calculated after ADC conversion. The data type unsigned char is selected for these variables because it can represent values from 0 to 255, effectively using all 8 bits to represent positive integers. This choice is particularly suitable since the variables are used for counting or storing state information that does not require negative values. Given that the highest value considered in our application is 99, which is well within the 0 to 255 range, using unsigned char is both sufficient and memory-efficient for our needs.
3. enum {degree = 10, C}: This enumeration is used to enhance the readability of the program. Here, degree is defined to represent the number 10, and C is set to represent the number 10 + 1, which equals 11. Consequently, SScodes[degree]

maps to SScodes[10], and SScodes[C] maps to SScodes[11]. This approach simplifies the code by using descriptive names for specific indices in the SScodes array, which improves understanding and maintenance of the code.

4. digits[5] = {0, 0, 0, degree, C}: This array is designated to store the digits that will be displayed on the 7-segment display. Specifically, degree and C are consistently shown on digits[4] and digits[3] respectively, to represent degrees Celsius on the display. The array comprises 5 elements because the function Bin2Bcd returns 3 bytes, which are accommodated in digits[2:0]. This configuration ensures that the last two positions, digits[3:4], are reserved for degree and C, thereby maintaining the display of temperature units consistently. Using Bin2BcdE, which returns 5 bytes, would be inappropriate as it would alter the fixed assignment of degree and C in digits[3:4], which must remain unchanged to preserve the display format.

2) Functions:

I. The Setup function:

a) Set PORTD as digital outputs for the segments of the 7-segment display: TRISD = 0x00 ensures all pins of PORTD are configured as outputs, and ANSELD = 0x00 disables the analog functionality, making them strictly digital.

b) Configure the lower four bits (0-3) of PORTA for output, while the remaining bits retain their initial state: TRISA &= 0xF0 sets the lower four bits as outputs, and ANSEL &= 0xF0 ensures these bits are digital.

c) Set bit 2 of PORT E for analog input, linked to the transistor in the LM35 sensor: TRISBbits.RE2 = 1 configures the bit as an input, and ANSELBbits.ANSE2 = 1 enables its analog function.

d) Activate global interrupts and the Timer0 interrupt: INTCONbits.GIE = 1 enables all unmasked interrupts, and INTCONbits.T0IE = 1 specifically enables the Timer0 interrupt.

e) Program Timer0 (TMR0) to the binary setting 0b11010100, establishing precise timing for periodic interrupts (4 ms).

f) Initialize Timer0 to start counting from a calculated value to maintain a consistent 4 ms interval: TMR0L = 256 - 125 correctly sets the timer to overflow after 4 ms, calculated based on the timer's frequency and prescaler.

g) Enable the most significant digit (MSD) on the two-digit thermometer for the multiplexed display: Qstate = 0b00001000 sets the appropriate display state.

h) Set up the ADC Control Register 0 (ADCON0) to configure the analog input channel and ADC settings: ADCON0 = 0b00011101 selects RE2-AN7 as the analog channel (bits 6-2 set to ‘00111’), marks the ADC conversion as complete/not in progress (bit 1 ‘GO/DONE’ set to ‘0’), and turns on the ADC (bit 0 ‘ADON’ set to ‘1’). This configuration ensures the ADC is ready to convert the analog signal from the LM35 sensor into a digital value.

ADCON0: A/D CONTROL REGISTER 0

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
-	CHS<4:0>					GO/$\overline{\text{DONE}}$	ADON
bit 7							**bit 0**

R = Readable bit W = Writable bit U = Unimplemented bit, read as '0'
-n = Value at POR '1' = Bit is set '0' = Bit is cleared x = Bit is unknown

bit 7 **Unimplemented**: Read as '0'

Bit 6-2 **CHS<4:0>: Analog Channel Select bits**
00000 = AN0
00001 = AN1GO/$\overline{\text{DONE}}$
00010 = AN2
00011 = AN3
00100 = AN4
00101 = AN5(1)
00110 = AN6(1)
00111 = AN7(1)
01000 = AN8
01001 = AN9
01010 = AN10
01011 = AN11
01100 = AN12
01101 = AN13
01110 = AN14
01111 = AN15
10000 = AN16
10001 = AN17
10010 = AN18
10011 = AN19
10100 = AN20(1)
10101 = AN21(1)
10110 = AN22(1)
10111 = AN23(1)
11000 = AN24(1)
11001 = AN25(1)
11010 = AN26(1)
11011 = AN27(1)
11100 = Reserved
11101 = CTMU
11110 = DAC
11111 = FVR BUF2 (1.024V/2.048V/2.096V Volt Fixed Voltage Reference)(2)

bit 1 **GO/$\overline{\text{DONE}}$:** A/D Conversion Status bit
1 = A/D conversion cycle in progress. Setting this bit starts an A/D conversion cycle.
This bit is automatically cleared by hardware when the A/D conversion has completed.
0 = A/D conversion completed / not in progress)

bit 0 **ADON:** ADC Enable bit
1 = ADC is enabled
0 = ADC is disabled and consumes no operating current

Note 1: Available on PIC18(L)F4XK22 devices only.
Note 2: Allow greater than 15 µs acquisition time when measuring the Fixed Voltage Reference.

Figure 2: ADCON0

Configure A/D control register 1: **ADCON1** with the hex value **` 0x00`** to ensure that A/D $V_{REF+/-}$ are connected to internal signals.

ADCON1: A/D CONTROL REGISTER 1

R/W-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
TRIGSEL	–	–	–	PVCFG<1:0>		NVCFG<1:0>	
bit 7							**bit 0**

R = Readable bit W = Writable bit U = Unimplemented bit, read as '0'
-n = Value at POR '1' = Bit is set '0' = Bit is cleared x = Bit is unknown

bit 7 **TRIGSEL**: Special Trigger Select bit
1 = Selects the special trigger from CTMU
0 = Selects the special trigger from CCP5

bit 6-4 **Unimplemented:** Read as '0'

bit 3-2 **PVCFG<1:0>:** Positive Voltage Reference Configuration bits
00 = A/D VREF+ connected to internal signal, AVDD
01 = A/D VREF+ connected to external pin, VREF+
10 = A/D VREF+ connected to internal signal, FVR BUF2
11 = Reserved (by default, A/D VREF+ connected to internal signal, AVDD)

bit 1-0 **NVCFG<1:0>:** Negative Voltage Reference Configuration bits
00 = A/D VREF- connected to internal signal, AVSS
01 = A/D VREF- connected to external pin, VREF-
10 = Reserved (by default, A/D VREF- connected to internal signal, AVSS)
11 = Reserved (by default, A/D VREF- connected to internal signal, AVSS)

Figure 3: ADCON1

Set the A/D Control Register 2 (ADCON2) to configure the ADC's result format, acquisition time, and conversion clock: ADCON2 = 0b00001001. This setup involves:

a) **Left Justified Result Format:** Setting Bit 7 (ADFM) to 0 configures the ADC result to be left-justified, which is suitable for applications where 8 bits of resolution are sufficient, like measuring temperatures from 0 to 100°C.

b) **Acquisition Time:** Bits 5-3 (ACQT<2:0>) set to 001 specify an acquisition time of 2 TAD, ensuring adequate time for the ADC's hold capacitor to charge to the input voltage level.

c) **Conversion Clock:** Bits 2-0 (ADCS<2:0>) set to 001 select a conversion clock of FOSC/8, optimizing the ADC conversion speed relative to the oscillator frequency for accurate and timely readings. This configuration provides a balance between

conversion speed and accuracy, suitable for the range of temperature measurements in your application.

ADCON2: A/D CONTROL REGISTER 2

R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	–	ACQT<2:0>			ADCS<2:0>		
bit 7							bit 0

bit 7 **ADFM**: A/D Result Format Select bit
1 = Right justified
0 = Left justified

bit 6 Unimplemented: Read as '0'

bit 5-3 **ACQT<2:0>**: A/D Acquisition time select bits. Acquisition time is the duration that the A/D charge holding capacitor remains connected to A/D channel from the instant the GO/$\overline{\text{DONE}}$ bit is set until conversions begins.
000 = 0(1)
001 = 2 TAD
010 = 4 TAD
011 = 6 TAD
100 = 8 TAD
101 = 12 TAD
110 = 16 TAD
111 = 20 TAD

bit 2-0 **ADCS<2:0>**: A/D Conversion Clock Select bits
000 = FOSC/2
001 = FOSC /8
010 = FOSC /32
011 = FRC(1) (clock derived from a dedicated internal oscillator = 600 kHz nominal)
100 = FOSC /4
101 = FOSC /16
110 = FOSC /64
111 = FRC(1) (clock derived from a dedicated internal oscillator = 600 kHz nominal)

Note 1: When the A/D clock source is selected as FRC then the start of conversion is delayed by one instruction cycle after the GO/$\overline{\text{DONE}}$ bit is set to allow the SLEEP instruction to be executed.

Figure 4: ADCON2

II. The Interrupt Service Routine (ISR) function:

1) **Memory Location and Interrupt Handling**: ISR resides at 0x0008, handling Timer0 interrupts.

2) **Clearing Interrupt Flag and Restarting Timer**: Clear T0IF flag, reset Timer0 for counting**.**

3) **Display Update and Multiplexing:** Update PORTD for the display and configure PORTA for multiplexing purposes. The 7-segment codes are accessed from the SScodes[] array using the current digit index i. The process involves cycling through each digit by increasing the i index and adjusting Qstate, which manages the multiplexing, to choose the subsequent digit.
4) **Cycle Control and Multiplex Reset:** After displaying each digit, Qstate is shifted right by one bit to manage the multiplexing for the next digit's display. When i reaches 5, signifying that all digits have been displayed, it is reset to 1. Concurrently, Qstate is reset to 0b00001000, starting the next cycle with the most significant digit. This ensures the multiplexing sequence is correctly maintained for each subsequent display cycle..

III. The measureT function:

1) **Initiate ADC Conversion:** Set **`GO_DONE`** to start ADC conversion.
2) **Wait for Conversion Completion:** This step involves a loop that pauses execution until the ADC conversion is fully completed. It checks the NOT_DONE bit in the ADCON0 register to see if the conversion is still underway. The loop persists until this bit indicates that the conversion is finished.
3) **Calculate temperature:** The temperature is computed by using the ADC result stored in ADRESH along with the previously defined function. To ensure proper handling of the data, ADRESH is cast to (int) before multiplication. This casting is crucial as it allows the operation 100 * ADRESH to proceed under integer arithmetic, preventing overflow issues that arise with the maximum unsigned char value of 255. For example, without casting, the product 100 * 255 equals 25,500, which exceeds the unsigned char limit of 255. By casting ADRESH as (int), the

result remains within the acceptable range for integers, which is up to 65,536, thus accommodating the computation accurately.

4) **Convert to BCD:** Utilize Bin2Bcd function to convert voltage to Binary-Coded Decimal (BCD) format.

B. Proteus simulation results:

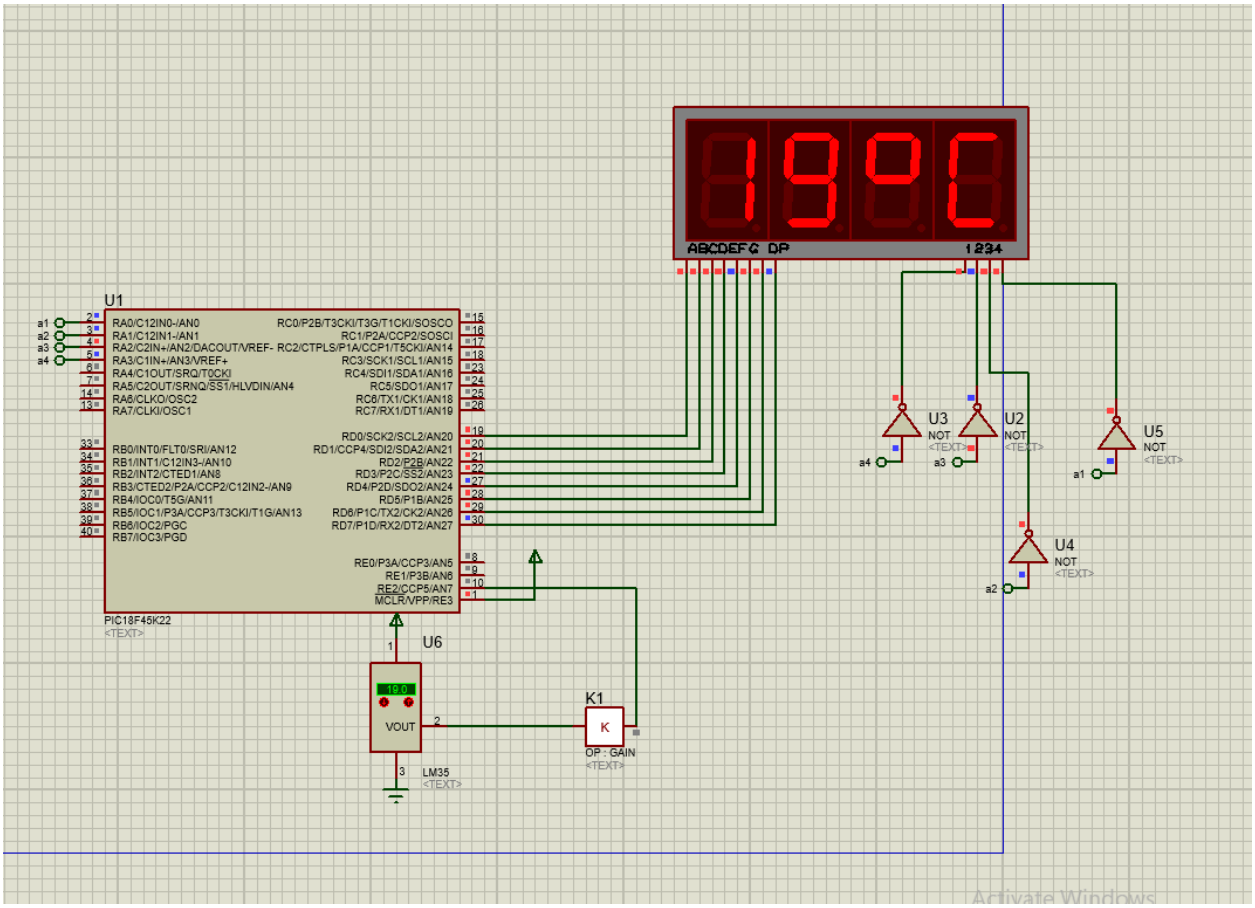


Figure 6: Simulation of a digital thermometer with gain and displaying the temperature 19°C

-

Since this multiplexed display is a common cathode display, we need to put an inverter for the output signal to work correctly. The pin RE3 was connected to power in order to energize the PIC18.

- ***Part d: Digital Thermometer without gain:***

This section is a revised version of part a and was completed prior to parts b and c in the lab. In this segment, we are tasked with designing a 2-digit thermometer capable of measuring temperatures from 0°C to 100°C. To accurately display temperature values without implementing a gain, it was necessary to adjust the Vref+ to approximately 1V. This adjustment required reconfiguring the ADC to utilize the FVR (fixed voltage reference) set to output 1024 mV, which is then applied as Vref+. Consequently, Vref+ is now set to 1.024 V, while Vref- remains unchanged at 0 V.

Following the same strategy for deriving an equation for Tc:

$$V = 0.01 * Tc = Vin$$

where Vin $= \frac{V_{REF}}{2^n} * ADRESH = \frac{V_{REF+-V_{REF-i.}}}{2^n} * ADRESH$ ¿ $= \frac{102.4}{2^n} * ADRESH =$

$\frac{102.4}{256} * ADRESH$

Consequently,

$$0.01 * Tc = \frac{102.4}{256} * ADRESH$$

$$\rightarrow Tc = \frac{1024}{256} * ADRESH$$

A. MPLAB Code:

```
#include <p18cxxx.h>
#include <BCDlib.h>

char SScodes[] = {0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x63,0x39};
unsigned char i, Qstate;
float temp;
enum {degree = 10, C};
char digits[5] = {0, 0, 0, degree, C};

void setup(void);
void measureT(void);

void main (void) {
   setup();
   while(1) {
      measureT();
   }
}
void setup (void) {
   ANSELD = TRISD = 0x00;
   TRISA &= 0xF0;
   ANSELA &= 0xF0;
   TRISEbits.RE2 = ANSELEbits.ANSE2 = 1;

   ADCON0 = 0b00011101;
   ADCON1 = 0x08;
   ADCON2 = 0b00001001;

   VREFCON0= 0b10010000;
   INTCONbits.T0IE = 1;
   INTCONbits.GIE = 1;
   T0CON = 0b11010100;
   TMR0L = 256 - 125;
   i = 1;
   Qstate = 0b00001000;
}
```

```
#pragma code ISR = 0x0008
#pragma interrupt ISR

void ISR (void) {
   INTCONbits.T0IF = 0;
   TMR0L = 256 - 125;
   PORTD = 0x00;
   PORTA = Qstate;
   PORTD = SScodes[digits[i++]];
   Qstate >>= 1;

   if (i == 5) {
      i = 1;
      Qstate = 0b00001000;
    }
}

void measureT(void){
   ADCON0bits.GO_DONE = 1;
   while(ADCON0bits.NOT_DONE);
   temp = (ADRESH * (102.4 /256));
   Bin2Bcd ((unsigned char) temp, digits);
}
```

B. The modifications:

1. `float temp;`

   The variable temp is calculated using the formula (ADRESH * (102.4 / 256)), which can result in decimal values depending on the values of ADRESH. The data type float is chosen for temp because it supports decimal values and is suitable for the range of values that temp can potentially take, ensuring accurate representation of temperature measurements in decimal form.

2. `ADCON1 = 0x08;`

In this revised setup, Vref+ is set at 1.024 V, a change from the earlier configuration in part a. The adjustment involves modifying ADCON1, which controls the configuration of the Positive/Negative Voltage References. To accommodate this change, ADCON1 is set to 0x08. Here, Bits 3-2 are configured as 10 to connect the A/D Vref+ to the internal signal from FVR BUF2, ensuring the reference voltage is precisely 1.024 V. Simultaneously, Bits 1-0 are set to 00, linking the A/D Vref- to the internal ground signal (AVSS), thereby maintaining a stable zero reference point for negative voltage measurements.

3.
```
VREFCON0= 0b10010000;
```

The register VREFCON0 is responsible for the configuration of FVR features. Fixed Voltage Reference Enable Bit (Bit 7) should be equal to **‘1’,** as well as Bit 6 which indicates that FVR is ready for use. Bits 5-4 should take the value **‘01’** corresponding to the explanation done before. Bits 3-2 and 1-0 should always be cleared.

VREFCON0: FIXED VOLTAGE REFERENCE CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-1	U-0	U-0	U-0	U-0
FVREN	FVRST	FVRS<1:0>		–	–	–	–
bit 7							**bit 0**

R = Readable bit W = Writable bit U = Unimplemented bit, read as ‘0’
-n = Value at POR ‘1’ = Bit is set ‘0’ = Bit is cleared x = Bit is unknown

bit 7 **FVREN:** Fixed Voltage Reference Enable bit
0 = Fixed Voltage Reference is disabled
1 = Fixed Voltage Reference is enabled

bit 6 **FVRST:** Fixed Voltage Reference Ready Flag bit
0 = Fixed Voltage Reference output is not ready or not enabled
1 = Fixed Voltage Reference output is ready for use

bit 5-4 **FVRS<1:0>:** Fixed Voltage Reference Selection bits
00 = Fixed Voltage Reference Peripheral output is off
01 = Fixed Voltage Reference Peripheral output is 1x (1.024V)
10 = Fixed Voltage Reference eripheral output is 2x (2.048V)(1)
11 = Fixed Voltage Reference Peripheral output is 4x (4.096V)(1)

bit 3-2 **Reserved:** Read as ‘0’. Maintain these bits clear.

bit 1-0 **Unimplemented:** Read as ‘0’.

Note 1: Fixed Voltage Reference output cannot exceed VDD.

Figure 6: VREFCON0

4.
```
temp = (ADRESH * (102.4 /256));
```

This modification is explained before in the calculation part. No casting was needed here since the maximum value that a float type can take is , so it can handles all possible values.

5.
```
Bin2Bcd ((unsigned char) temp, digits);
```

The casting was needed here since Bin2Bcddoesn't take float type variables as parameters; it only takes variables of type unsigned char.

C. Demo Board Implementation:

Our project employs the EasyPIC v7 Demo Board, which is programmed using the mikroProg Suite™ for PIC®. This board includes a 7-segment display that is operated by manually switching specific transistors on and off.

Specifically, transistors DIS0, DIS1, DIS2, and DIS3, which correspond to the pins RA0, RA1, RA2, and RA3 respectively, are utilized to control the activation of individual digits on the display. For our temperature sensing application, a yellow jumper has been placed on RE2 to facilitate the connection of the LM35 temperature sensor to the RE2 pin on the PIC18 microcontroller.

Additionally, considering that transistors are a fundamental component of sensors, their integration onto the Demo Board was necessary. The code we have developed is successfully functioning to display temperature readings on a 2-digit thermometer.

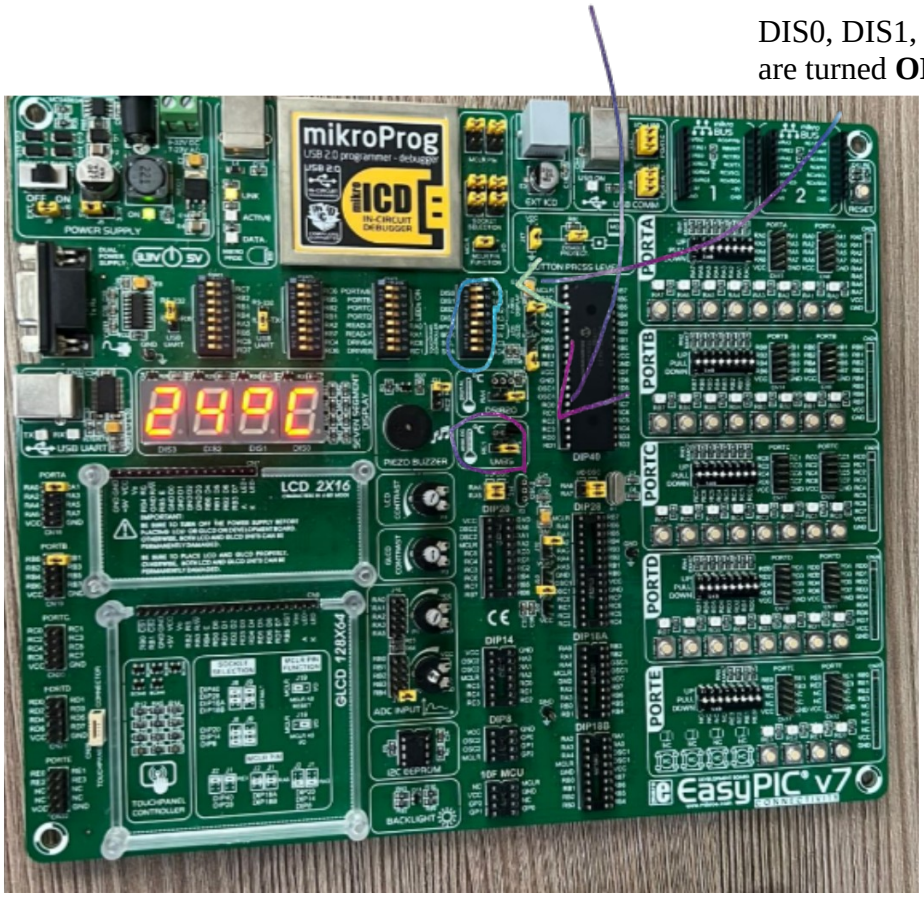


Figure 7: Displaying the temperature in Celsius

- ***Part b: Digital Thermometer without gain Celsius/Fahrenheit:***

This section is an enhanced version of part d, where we are tasked with creating a 2-digit thermometer capable of measuring temperatures ranging from 0°C to 100°C. To improve the user interface of our design, we incorporated a feature that allows the measured temperature to be displayed in either Celsius or Fahrenheit, controlled by the state of a designated push button. This button is connected to the RB0 pin on the PIC18 microcontroller. By default, when the push button is not pressed, the temperature is shown in Celsius. However, pressing the button switches the display to show the temperature in Fahrenheit.

The following conversion formula between Celsius and Fahrenheit is implemented in our code:

$$\rightarrow Tf = 1.8 * Tc + 32$$

Tf represents the temperature in Fahrenheit and TcTc represents the temperature in Celsius. This formula is utilized to convert the temperature readings between Celsius and Fahrenheit based on the user's selection. The converted temperature value is then displayed accordingly.

A. MPLAB Code:

```
#include <p18cxxx.h>
#include <BCDlib.h>

char SScodes[] ={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x63,0x39,0x71};
unsigned char i, Qstate;
float temp;
enum {degree = 10, C, F};
char digits[5] = {0, 0, 0, degree, C};
```

```
void setup(void);
void measureT(void);

void main (void) {
   setup();
   while(1) {
      measureT();
   }
}

void setup (void) {
   ANSELD = TRISD = 0x00;
   TRISA &= 0xF0;
   ANSELA &= 0xF0;
   TRISBbits.RB7 = 0;
   TRISBbits.RB6 = 0;
   TRISBbits.RB5 = ;
   TRISBbits.RB0 = 1;
   ANSELBbits.ANSB0 = 0;
   TRISEbits.RE2 = ANSELEbits.ANSE2 = 1;

   INTCON2bits.RBPU = 0;
   INTCONbits.T0IE = 1;
   INTCONbits.GIE = 1;

   ADCON0 = 0b00011101;
   ADCON1 = 0x08;
   ADCON2 = 0b00001001;

   WPUB = 0b00000001;
   VREFCON0 = 0b10010000;
   T0CON = 0b11010100;
   TMR0L = 256 - 125;
   i = 1;
   Qstate = 0b00001000;
}

#pragma code ISR = 0x0008
#pragma interrupt ISR

void ISR (void) {
   INTCONbits.T0IF = 0;
   TMR0L = 256 - 125;
   PORTD = 0x00;
```

```
  PORTA = Qstate;
  PORTD = SScodes[digits[i++]];
  Qstate >>= 1;

  if (i == 5) {
      i = 1;
      Qstate = 0b00001000;
  }
}

void measureT(void) {
  ADCON0bits.GO_DONE = 1;
  while (ADCON0bits.NOT_DONE);
  temp = (ADRESH * (102.4 /256));

  if (INTCONbits.INT0IF) {
    INTCONbits.INT0IF = 0;
    if (digits[4] == C)
      digits[4] = F;
    else digits[4] = C;
  }

  if (digits[4] == F)
    temp = 1.8 * temp + 32;

  Bin2Bcd ((unsigned char) temp, digits);
}
```

B. The modifications:

1. `char SScodes[] ={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x63,0x39,0x71};`

   This array contains the 7-segment display codes for digits zero through nine and includes the codes for '°' (0x63) and 'C' (0x39) for displaying these symbols on the 7-segment display. Furthermore, the code for 'F' (0x71) has been added to this array. This addition enables the display to show the letter 'F' when the user presses the push button to switch the temperature reading to Fahrenheit, replacing the 'C' displayed for Celsius.

2. `enum {degree = 10, C, F};`

Consistent with the approach in Part a, this enumeration enhances the readability of the program. Here, 'F' is assigned the number 12, aligning it with its position in the SScodes array. Thus, SScodes[F] directly refers to SScodes[12], simplifying the reference to the Fahrenheit symbol in the code.

```
3.          TRISBbits.RB0 = 1;
        ANSELBbits.ANSB0 = 0;
```

This code configures the RB0 pin of PORT B as a digital input to accommodate the push button. Setting TRISBbits.RB0 to 1 designates the pin as an input, while ANSELBbits.ANSB0 set to 0 ensures that the pin functions as a digital input rather than an analog input. This setup is necessary for correctly reading the state of the push button connected to RB0.

```
4.      INTCON2bits.RBPU = 0;
        WPUB = 0b00000001;
```

These settings are implemented to activate the internal weak pull-up resistor on RB0. By setting INTCON2bits.RBPU to 0, the internal pull-up resistors are enabled for PORTB. The WPUB is set to 0b00000001 specifically targeting RB0, ensuring that this pin has a pull-up resistor enabled. This configuration is essential when RB0 is used as an input pin, such as for a push button, to ensure stable input readings.

```
5. void measureT(void) {
   ADCON0bits.GO_DONE = 1;
   while (ADCON0bits.NOT_DONE);
   temp = (ADRESH * (102.4 /256));

   if (INTCONbits.INT0IF) {
      INTCONbits.INT0IF = 0;
      if (digits[4] == C)
         digits[4] = F;
      else digits[4] = C;
```

```
  }

  if (digits[4] == F)
     temp = 1.8 * temp + 32;

  Bin2Bcd ((unsigned char) temp, digits);
}
```

This modification is implemented to support the newly added functionality of displaying temperature in either Celsius ('C') or Fahrenheit ('F'), based on user interaction with the push button connected to RB0 on PORT B. When the push button is pressed, the INT0IF flag in the INTCONbits register is set to 1.

This flag remains set until it is manually cleared in the code, which helps address any debouncing issues. If digits[4], which displays the temperature unit, is showing 'C' and the push button is pressed, it should switch to 'F' to indicate that the temperature is now being displayed in Fahrenheit. Conversely, if digits[4] is already displaying 'F', the display should continue to show 'F'. The temperature is initially calculated using the equation: temp = (ADRESH * (102.4 / 256));.

However, if digits[4] is set to 'F', indicating Fahrenheit, the temperature calculation adjusts to temp = 1.8 * temp + 32; to convert Celsius to Fahrenheit. Similar to part d, casting is necessary here as well because Bin2Bcd does not accept float type variables; it only accepts variables of type 'unsigned char'. This step ensures that the variable temp, which may contain a float due to the conversion formula, is appropriately processed for display.

C. Proteus simulation results:

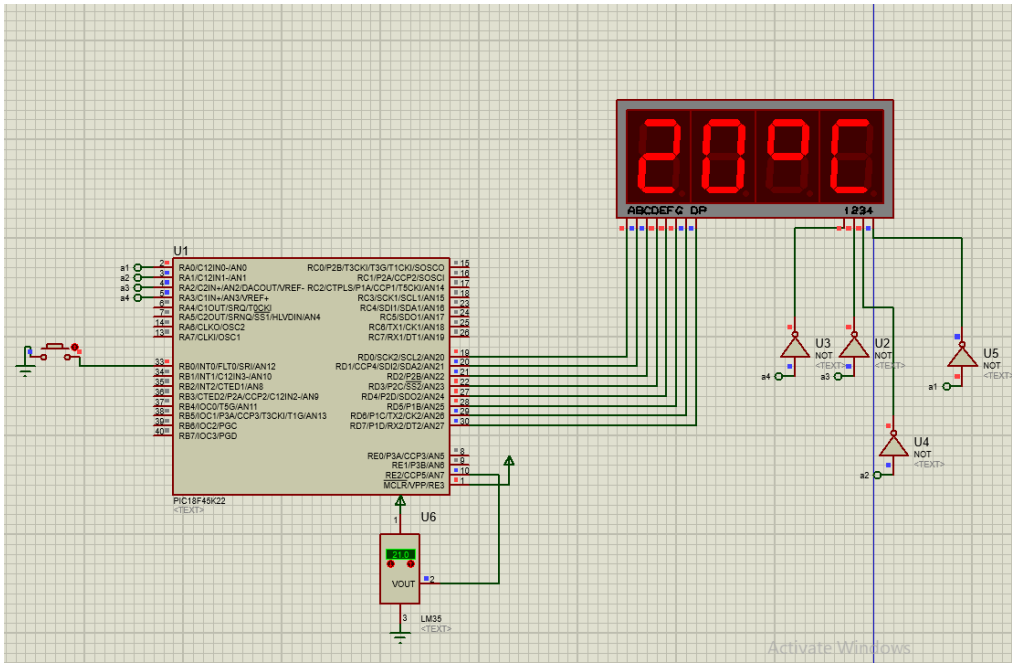


Figure 8: Simulation of the 2-digit thermometer and displaying 20°C

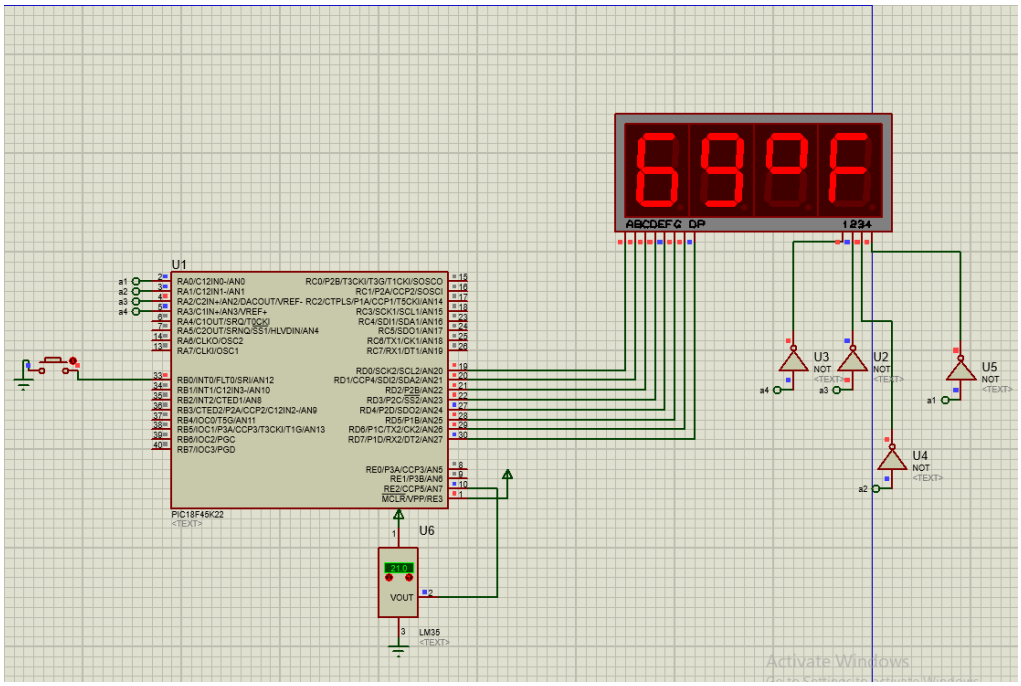


Figure 9: Simulation of the 2-digit thermometer and displaying the equivalent temperature of 20°C in Fahrenheit (69°F)

D. Demo Board Implementation:

In this segment of the project, we employ transistors DIS0, DIS1, DIS2, and DIS3, which correspond to pins RA0, RA1, RA2, and RA3, respectively, to control the activation of individual digits on the display.

Furthermore, a yellow jumper is used to connect the LM35 temperature sensor to the RE2 pin of the PIC18 microcontroller, facilitating temperature measurement. Recognizing that transistors are fundamental components in sensors, their inclusion on the DemoBoard is essential for operation.

Additionally, beyond what was established in Part d, a yellow jumper has also been placed to the ground on j17 because the internell pull up is on consequently u have to put ground on the other side.

Yellow jumper on RE2.
Placement of a transistor.

RB0 to ground because the internal pull up is on

DIS0, DIS1, DIS2 and DIS3 are turned **ON**.

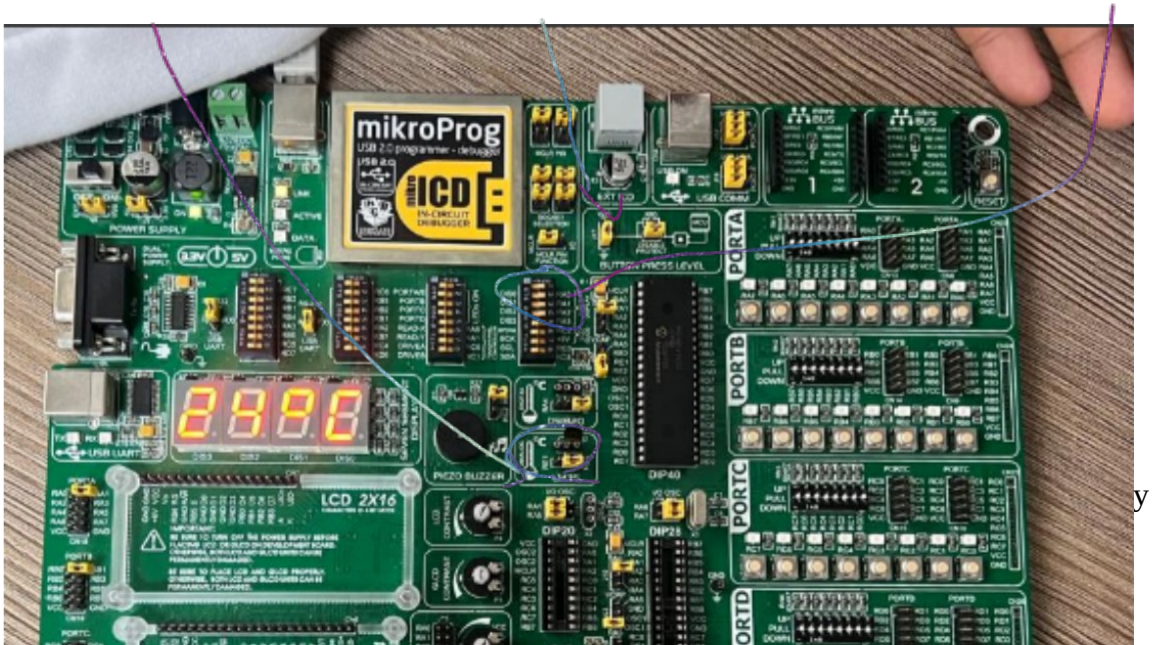


Figure 10: Displaying 24°C

Once the push button connected to pin RB0 is pressed, 24°C is converted to Fahrenheit and the result is displayed on the 7-segment display.

The push button connected to RB0 is pressed.

Figure 11: Displaying the equivalent temperature of 24°C in Fahrenheit (75°F)

- ***<u>Part c: Digital Thermometer with indicators:</u>***

This is a modified version of part ***b***. In this part, a ***Control*** function was added to control the state of three LEDS: HI (connected to RB6), MED (connected to RB5) and LO (connected to RB7) depending on the temperature value as the following:

$$0\ °C \le T_C < 20°C, \qquad \textit{Turn on LO LED}$$
$$20°C \le T_C \le 30°C, \qquad \textit{Turn on MED LED}$$
$$30°C < T_C \le 100°C, \qquad \textit{Turn on HI LED}$$

NB: These three LEDS also turn ON under the same conditions, but the same temperatures values are converted to Fahrenheit.

A. MPLAB Code:

```
#include <p18cxxx.h>
#include <BCDlib.h>

char SScodes[] ={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x63,0x39,0x71};
unsigned char i, Qstate;
float temp;
enum {degree = 10, C, F};
char digits[5] = {0, 0, 0, degree, C};

#define LO PORTBbits.RB5
#define MID PORTBbits.RB6
#define HI PORTBbits.RB7

void setup(void);
void measureT(void);
void controlLamp (void);

void main (void) {
  setup();
  while(1) {
    measureT();
  }
}

void setup (void) {
  ANSELD = TRISD = 0x00;
  TRISA &= 0xF0;
  ANSELA &= 0xF0;
  TRISBbits.RB0 = 1;
  TRISBbits.RB5 = TRISBbits.RB6 = TRISBbits.RB7 = 0;
  ANSELBbits.ANSB0 = ANSELBbits.ANSB5 = 0;
  INTCON2bits.RBPU = 0;
  WPUB = 0b00000001;
```

```
   TRISEbits.RE2 = ANSELEbits.ANSE2 = 1;
   ADCON0 = 0b00011101;
   ADCON1 = 0x08;
   ADCON2 = 0b00001001;
   VREFCON0= 0b10010000;
   INTCONbits.T0IE = 1;
   INTCONbits.GIE = 1;
   T0CON = 0b11010100;
   TMR0L = 256 - 125;
   i = 1;
   Qstate = 0b00001000;
   HI =0;
   MID =0;
   LO = 0;
}

#pragma code ISR = 0x0008
#pragma interrupt ISR

void ISR (void) {
   INTCONbits.T0IF = 0;
    TMR0L = 256 - 125;
    PORTD = 0x00;
    PORTA = Qstate;

    PORTD = SScodes[digits[i++]];
    Qstate >>= 1;

     if (i == 5) {
      i = 1;
      Qstate = 0b00001000;
    }
}

void measureT(void){
   ADCON0bits.GO_DONE = 1;
   while(ADCON0bits.NOT_DONE);
   temp = (ADRESH * (102.4 /256));
   if (INTCONbits.INT0IF) {
      INTCONbits.INT0IF = 0;
      if (digits[4] == C)
         digits[4] = F;
      else digits[4]= C;
   }
```

```
  if (digits[4] == F)
     temp = 1.8 * temp + 32;

  Bin2Bcd ((unsigned char) temp, digits);

  if (ADRESH >= 75 && ADRESH < 250)   { //30 to 100 degree
     LO = 0;
     MID = 0;
     HI = 1;
  }
  else if (ADRESH >= 50 && ADRESH < 75){ //20 to 30 degree
     LO = 0;
     MID = 1;
     HI = 0;
  }
  else if (ADRESH >= 0 && ADRESH < 50 ) { //0 to 20 degree
     LO = 1;
     MID = 0;
     HI = 0;
  }
  else  {
     LO = 0;
     MID = 0;
     HI = 0;
  }
}
```

B. The modifications:

1.
```
#define LO PORTBbits.RB5
#define MID PORTBbits.RB6
#define HI PORTBbits.RB7
```

This is being done to make the code more readable. The LED LO is connected to pin RB6 on PORTB, LED HI is connected to pin RB7, and LED MED is connected to pin RB5.

The numbers 75 and 50 are not direct readings of temperature in Celsius or Fahrenheit but rather correspond to specific values of ADRESH, the register utilized for analog-to-digital conversion, under set reference temperature conditions: Tc = 0°C, Tc = 20°C, Tc = 30°C and Tc = 100°C, respectively. For example, at a temperature of 20°C, ADRESH is calibrated to 50, calculated as 50 = 20/0.4. This method is particularly useful since temperatures are frequently displayed in both Celsius and Fahrenheit. By determining the ADRESH values for these baseline temperatures, the system ensures precise accuracy across both temperature scales without the need for ongoing conversions. This streamlines the operation of the system and maintains consistent accuracy, regardless of the temperature unit displayed.

2.
```
TRISBbits.RB5 = TRISBbits.RB6 = TRISBbits.RB7 = 0;
ANSELBbits.ANSB0 = ANSELBbits.ANSB5 = 0;
```

These modifications indicate that the pins RB5-RB7 of PORT B should be configured as digital output pin since they are connected to LEDS.

3.
```
HI =0;
MID =0;
LO = 0;
```

In the setup function the three LEDs are initialized to 0;

4.
```
if (ADRESH >= 75 && ADRESH < 250)  {
    LO = 0;
    MID = 0;
    HI = 1;
  }
  else if (ADRESH >= 50 && ADRESH < 75){
    LO = 0;
```

```
    MID = 1;
    HI = 0;
}
else if (ADRESH >= 0 && ADRESH < 50) {
    LO = 1;
    MID = 0;
    HI = 0;
}
else  {
    LO = 0;
    MID = 0;
    HI = 0;
}
```

Initially, all three LEDs are off. Each LED will illuminate based on specific conditions outlined earlier. The values 75 and 50, which were previously calculated, represent thresholds for the ADRESH register, used to assess the current temperature. If ADRESH >= 75 && ADRESH < 250 evaluates as true, this indicates a high temperature, and the HI Lamp should be activated. Conversely, if ADRESH >= 50 && ADRESH < 70, the temperature is moderate, and the MED Lamp should be turned on. If the condition ADRESH >= 0 && ADRESH < 50 is met, it suggests a low temperature, prompting the LO Lamp to be activated.

Else neither of the conditions are met no LED will be on. This setup ensures that the appropriate LED lights up according to the temperature range detected, providing a clear visual indication of the temperature status.

C. Proteus simulation results:

- CASE 1: LO LED (connected to RB5) is ON since 18°C is less than 20°C. Converting 18°C to Fahrenheit, we obtained 65°F and the LO LED should also stay ON.

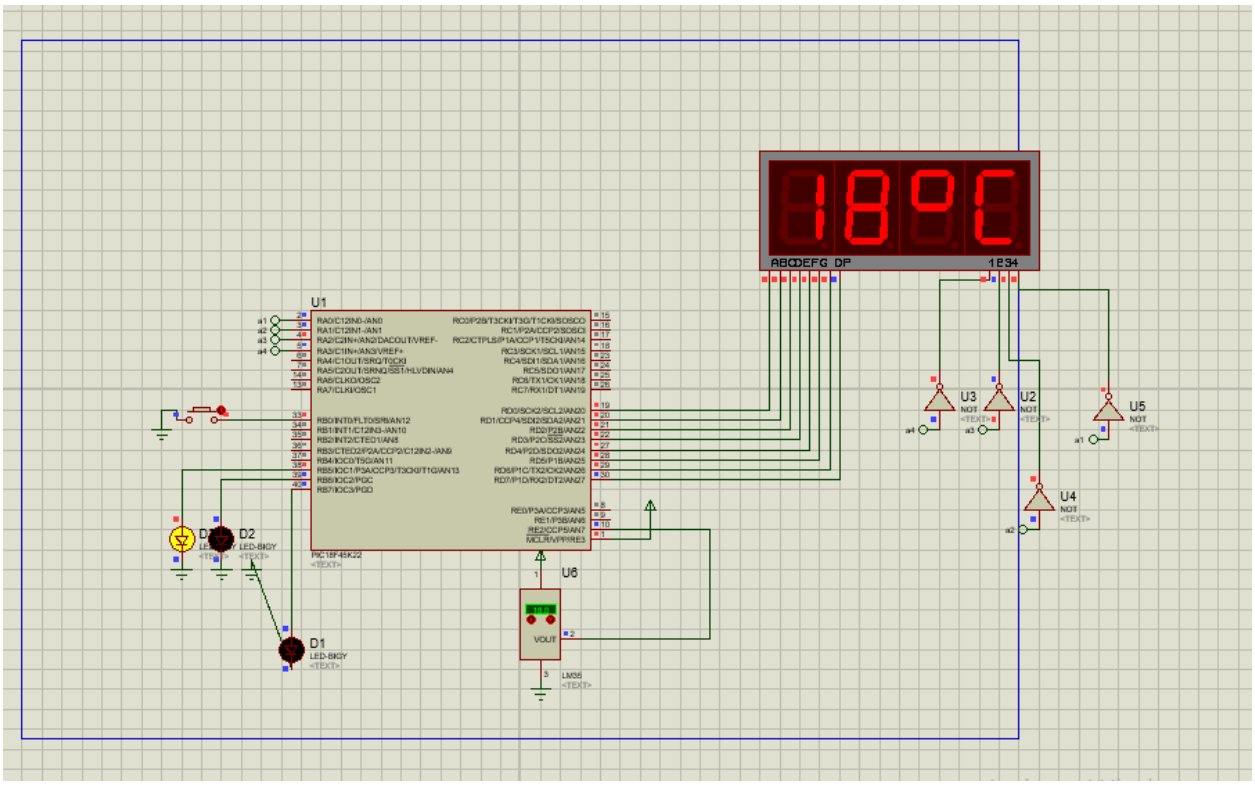


Figure 12: Case of Tc where LOWLED is ON

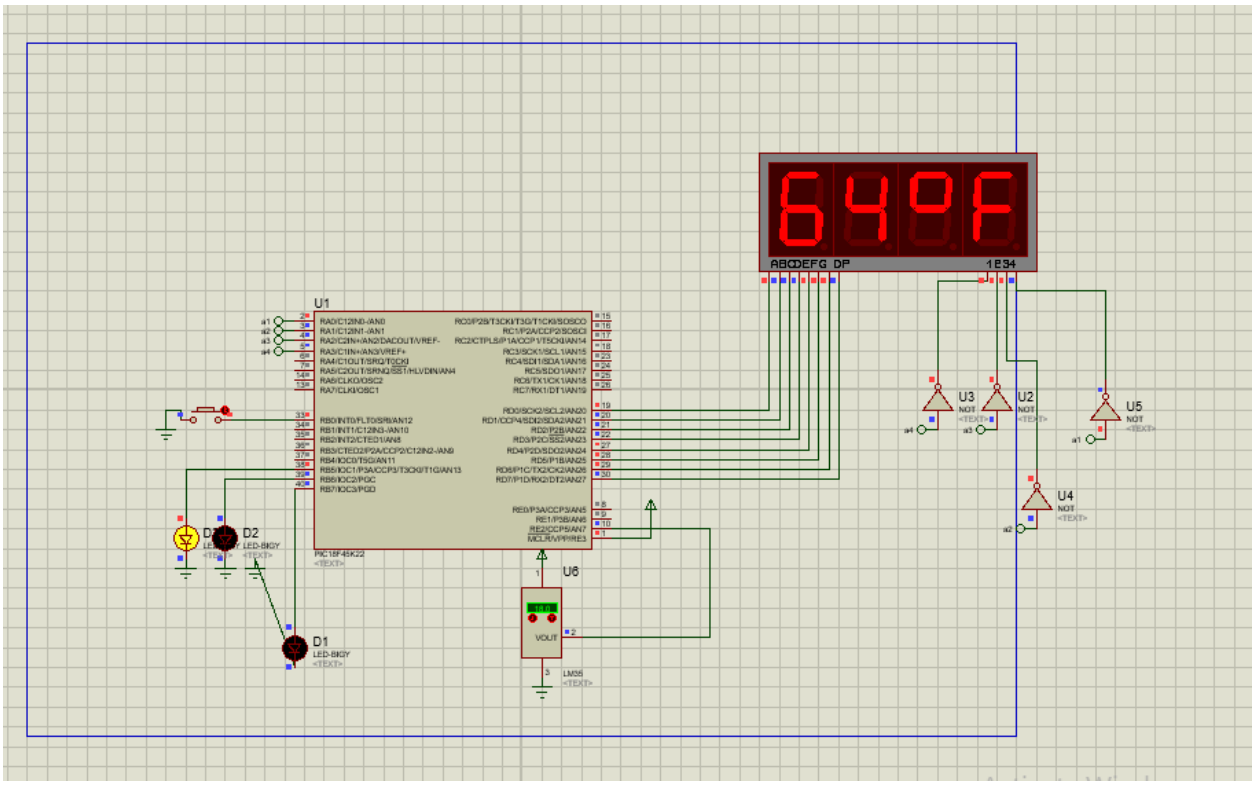


Figure 13: Case of Tf where LOWLED is ON

- CASE 2: MID LED (connected to RB6) is ON since 22°C falls between 20°C and 30°C. Converting 22°C to Fahrenheit, we obtained 71°F and the MID LED should also stay ON.

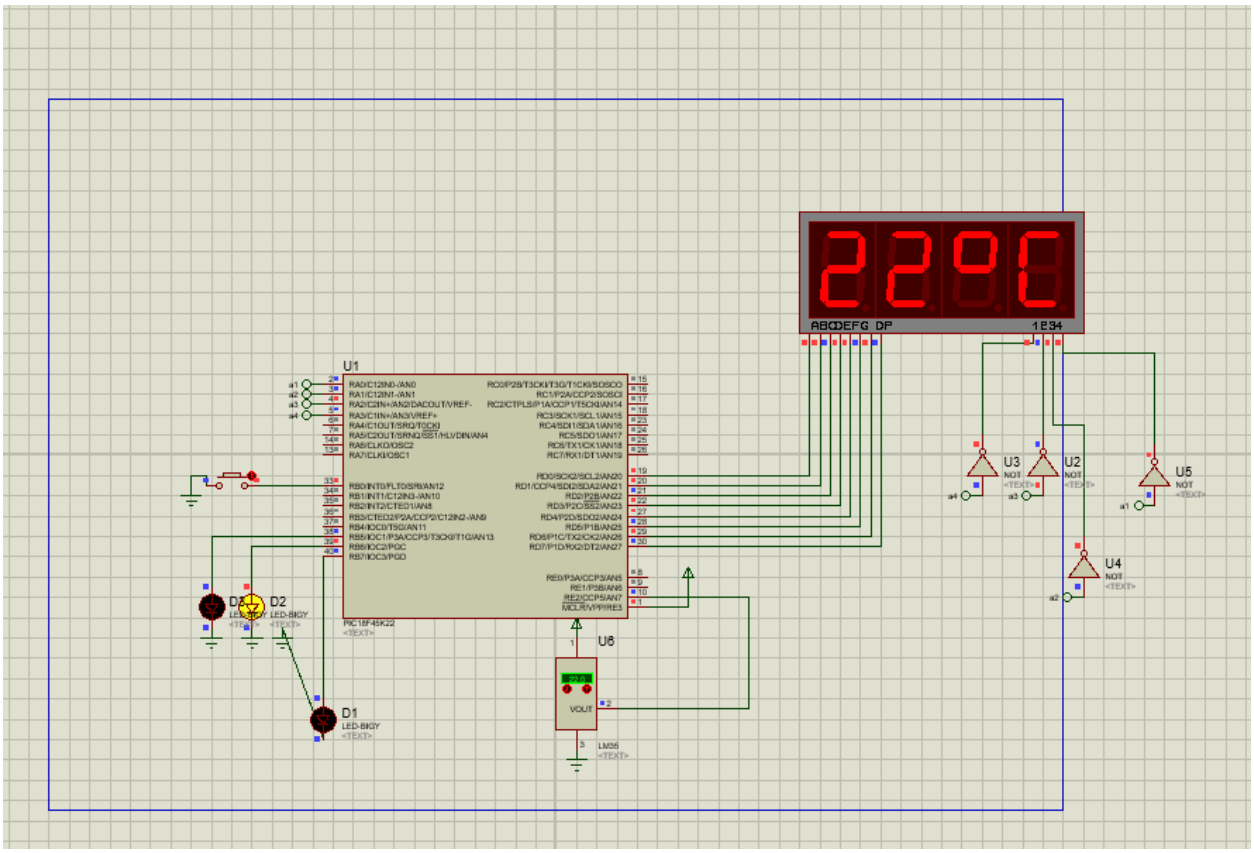


Figure 14: Case of Tc where MIDLED is ON

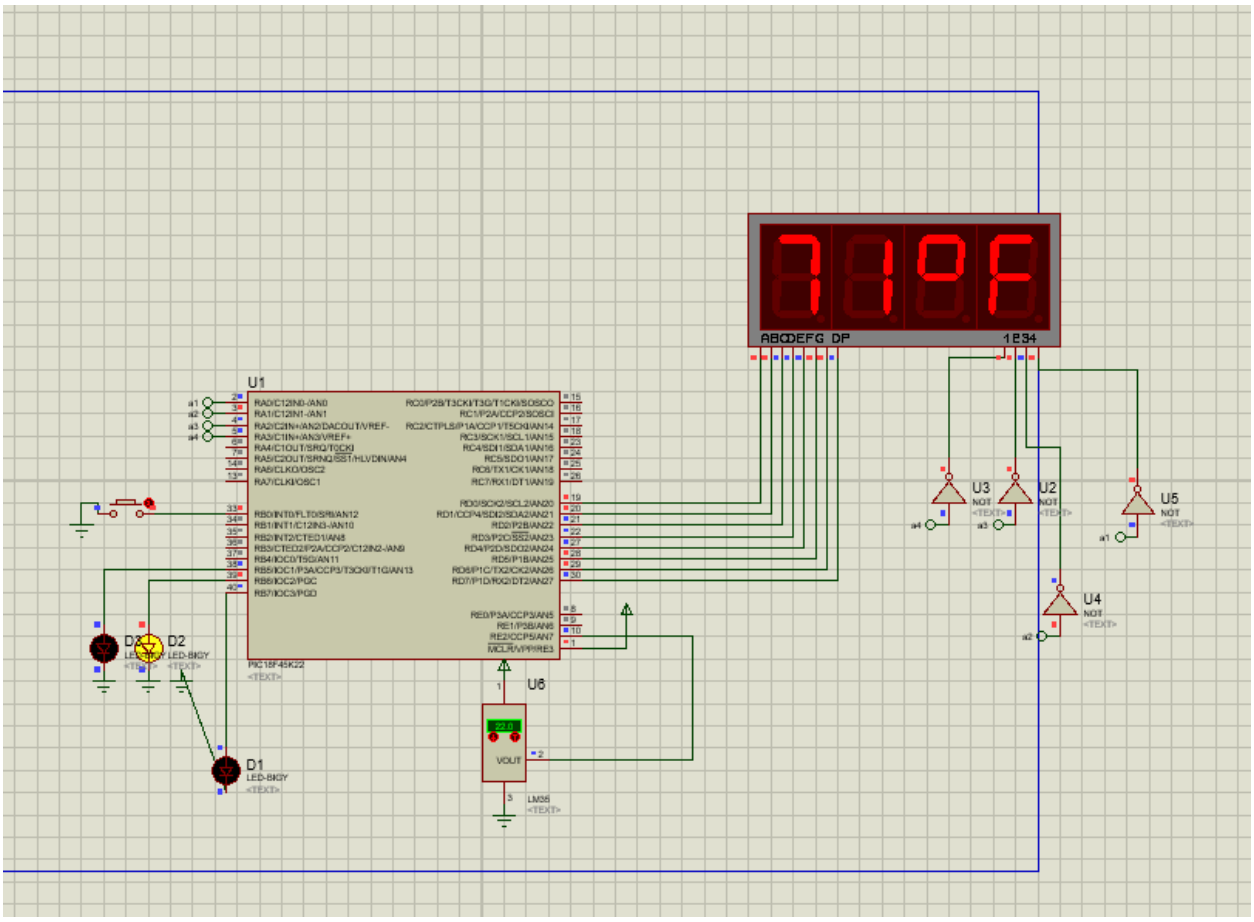


Figure 15: Case of Tf where MID LED is ON

- <u>CASE 3</u>: HI LED (connected to RB7) is ON since 36°C is greater then 30°C. Converting 36°C to Fahrenheit, we obtained 96°F and the HI LED should also stay ON.

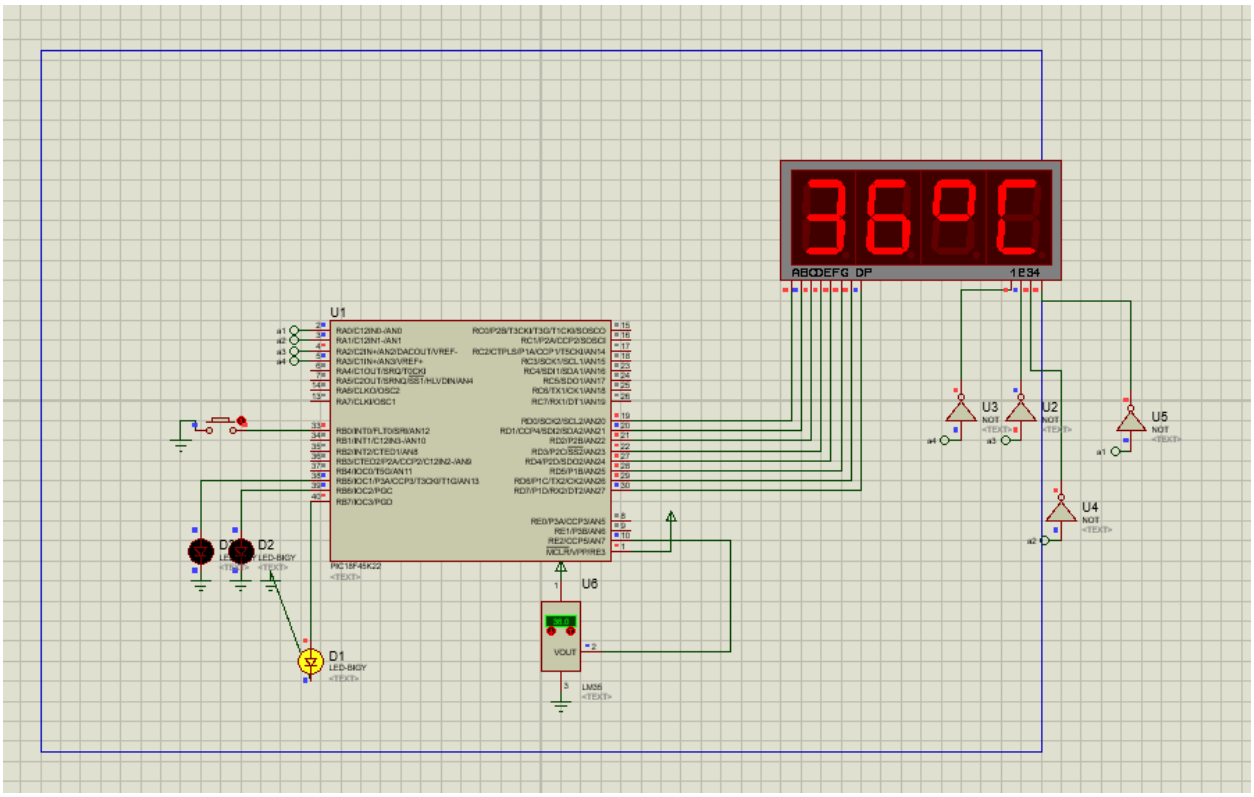


Figure 16: Case of Tc where HI LED is ON

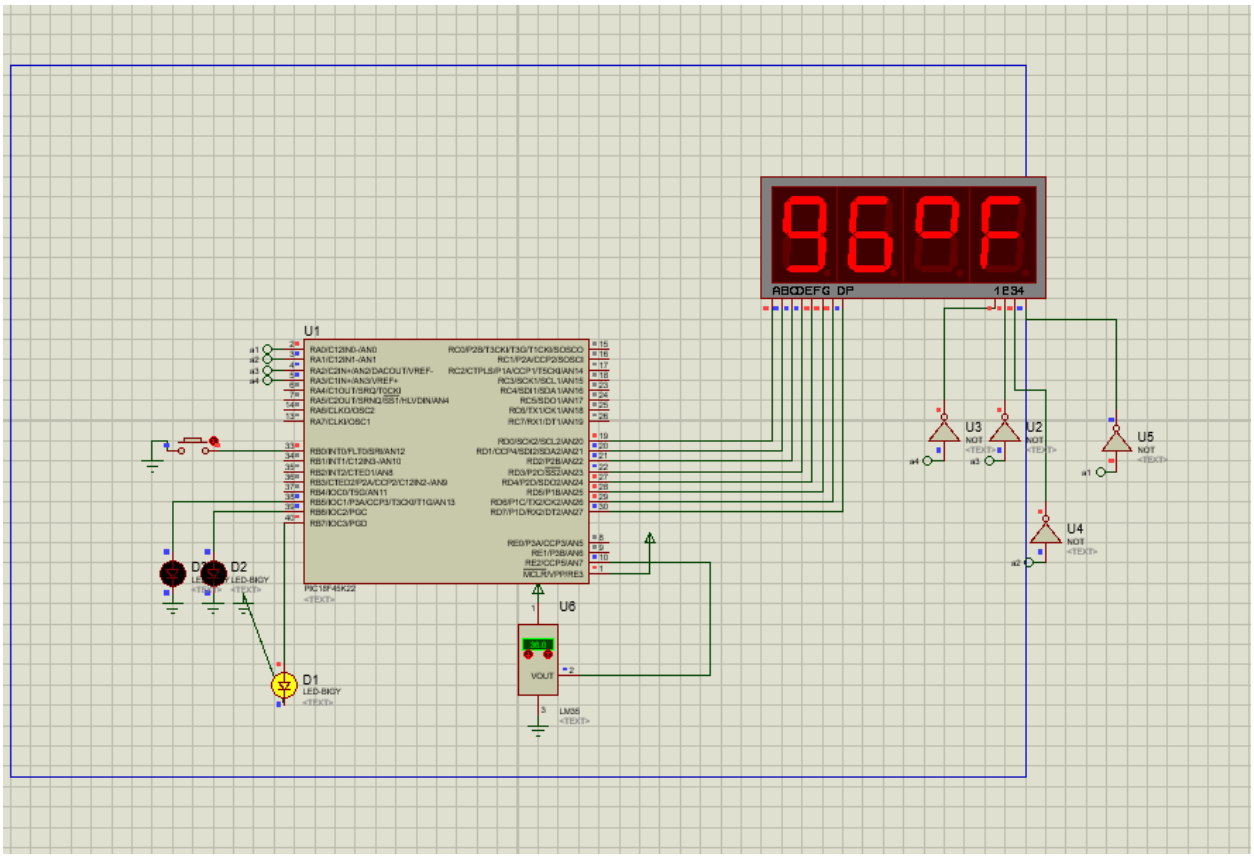


Figure 17: Case of Tf where HI LED is ON

D. Demo Board Implementation:

In this part, all steps done in Part ***b*** are the same, except that in our implementation, we could see a LED that turns ON depending on the temperature value to ensure the work Additionally, we switched the Port B (SW3) switch to ensure the led connection.

In Figure 18, since 24°C is considered as a medium temperature, the MID LED which is connected to pin RB6 is ON.

In Figure 19, since 86°F is considered as a HIGH temperature, the HI LED which is connected to pin RB7 is ON.

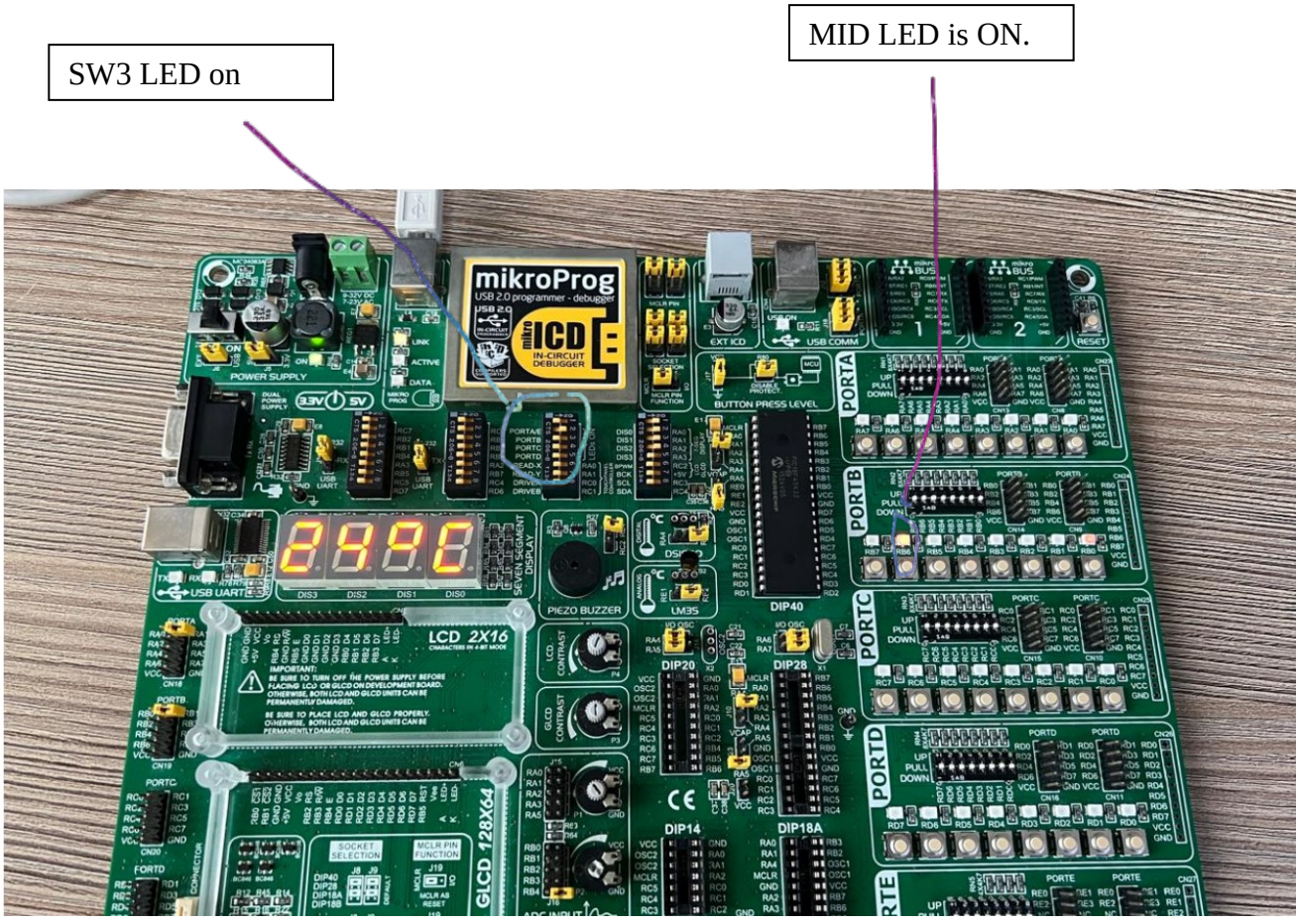


Figure 18: Displaying a medium temperature in Celsius

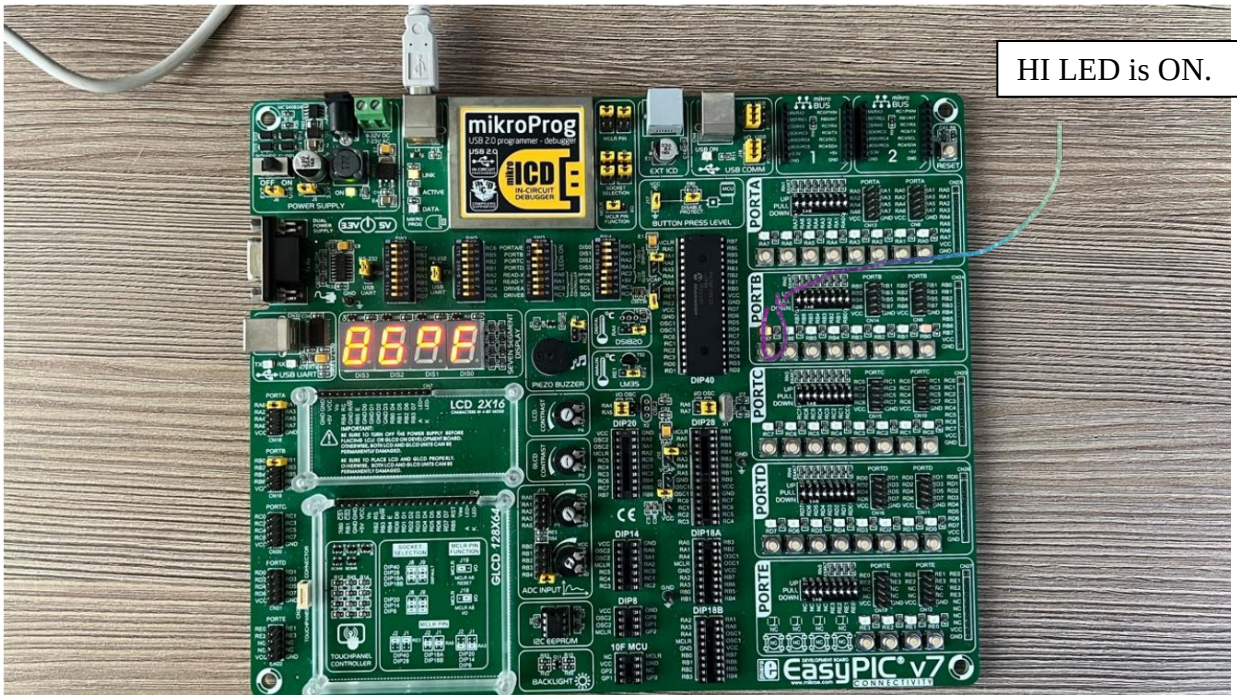


Figure 19: Displaying a HIGH temperature in Fahrenheit

III. Conclusion:

In conclusion, this experiment successfully implemented a digital thermometer using the LM35 temperature sensor and EasyPIC 7 demo board. The system offers temperature measurement capabilities ranging from 0°C to 100°C, with options to display readings in Celsius or Fahrenheit. Additionally, LED indicators provide feedback on temperature levels categorized as high, medium, or low. By configuring the A/D converter and utilizing voltage amplification, the system achieves accurate temperature readings. Overall, the experiment met its objectives by combining temperature calibration, unit selection, and temperature level indication, demonstrating practical microcontroller programming capabilities.