FANSHAWE COLLEGE

Temperature Display Using Voltage Variation

LEVEL 3

ELECTRONICS & EMBEDDED SYSTEMS DEVELOPMENT

Course Code: ELNC-6010

Prepared By:

Nishaben Desai (ID: 1086134)

Manan Patel (ID: 1202955)

Viral Gajera (ID: 1196217)

Table of Contents

1. Introduction 3

2. Objective 3

3. Theoretical Background 4

3.1 Microcontroller Operation 4
3.2 Analog-to-Digital Conversion (ADC) 4
3.3 USART Communication 4
3.4 LED Indicators 4

4. Apparatus/Equipment 5

5. Procedure 6

6. Results and Observations 7

7. Analysis and Discussion 7

8. Conclusion 8

9. Recommendations 9

10. References 10

11. Appendices 11

Introduction

This project focuses on creating a temperature measurement system using the PIC18F45K22 microcontroller. The system operates by varying the voltage levels through a potentiometer, displaying the corresponding temperature on an output device via serial communication. This project emphasizes the application of microcontroller technology learned during the course, integrating both analog and digital inputs to control outputs such as LEDs and temperature display.

Objective

The primary objective of this project is to design and implement a system that measures temperature by varying voltage levels, utilizing the PIC18F45K22 microcontroller. The system should accurately display temperature readings through a USART interface and provide visual feedback via LEDs.

Theoretical Background

In this project, several key concepts and technologies were utilized to create a functioning temperature measurement system. Understanding these concepts is critical for analyzing the system's operation and performance. Below, we discuss the relevant theory.

Microcontroller Operation

A microcontroller is a compact integrated circuit designed to govern a specific operation in an embedded system. The PIC18F45K22 microcontroller, used in this project, is part of the PIC18 family and is known for its efficiency in handling various control tasks. It features several input/output ports, ADC channels, and communication interfaces, making it ideal for applications such as temperature monitoring.

Analog-to-Digital Conversion (ADC)

Analog-to-digital conversion is the process of converting an analog signal (such as voltage) into a digital value that can be processed by a microcontroller. The PIC18F45K22 is equipped with multiple ADC channels that convert the analog input from the potentiometer into a digital value. This digital value is then used to determine the corresponding temperature, which is displayed via the USART interface.

USART Communication

USART (Universal Synchronous Asynchronous Receiver Transmitter) is a serial communication protocol used for transmitting data between devices. In this project, the USART module of the PIC18F45K22 microcontroller is used to send the temperature data to a terminal program (Tera Term) for display. The communication involves configuring the baud rate, enabling the transmitter and receiver, and sending data in the appropriate format.

LED Indicators

LEDs (Light Emitting Diodes) are used in this project to provide visual feedback on the system's operation. A red LED indicates that the temperature is high, while a green LED indicates that the temperature is low. This simple feedback mechanism helps users easily monitor the system's status.

Apparatus/Equipment

The following equipment and components were used to build and test the temperature measurement system:

Microcontroller: PIC18F45K22

Inputs:

- Pushbutton (Digital input)
- Potentiometer (Analog input)

Outputs:

- LED (Red indicates high tempreature; Green indicates low tempreature)
- Tera Term (USART-based output for displaying temperature)

Breadboard and connecting wires

Power Supply

Procedure

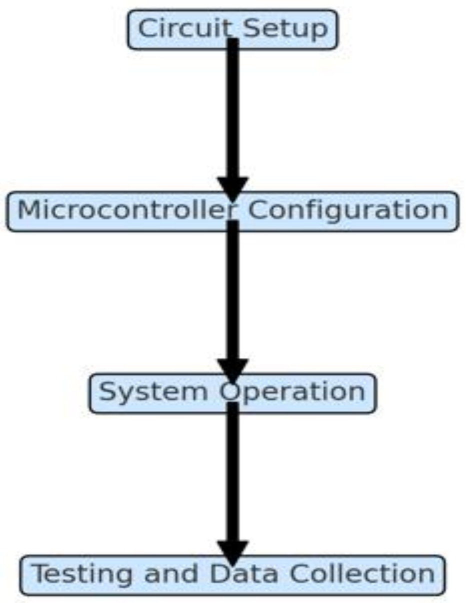


Fig 1. System operation flowchart

1. Circuit Setup: The components were arranged on the breadboard. The pushbutton and potentiometer served as the input devices, while the LEDs and Tera Term interface served as the outputs.

2. Microcontroller Configuration: The PIC18F45K22 microcontroller was configured using the provided code. The system initializes by setting up the oscillator, configuring ports, and setting up the ADC module for temperature measurement.

3. System Operation: Upon pressing the pushbutton, the system initiates temperature measurement. The potentiometer varies the input voltage, which the ADC module converts into a corresponding temperature reading. The output is displayed via the USART interface on Tera Term, with the LED providing visual feedback.

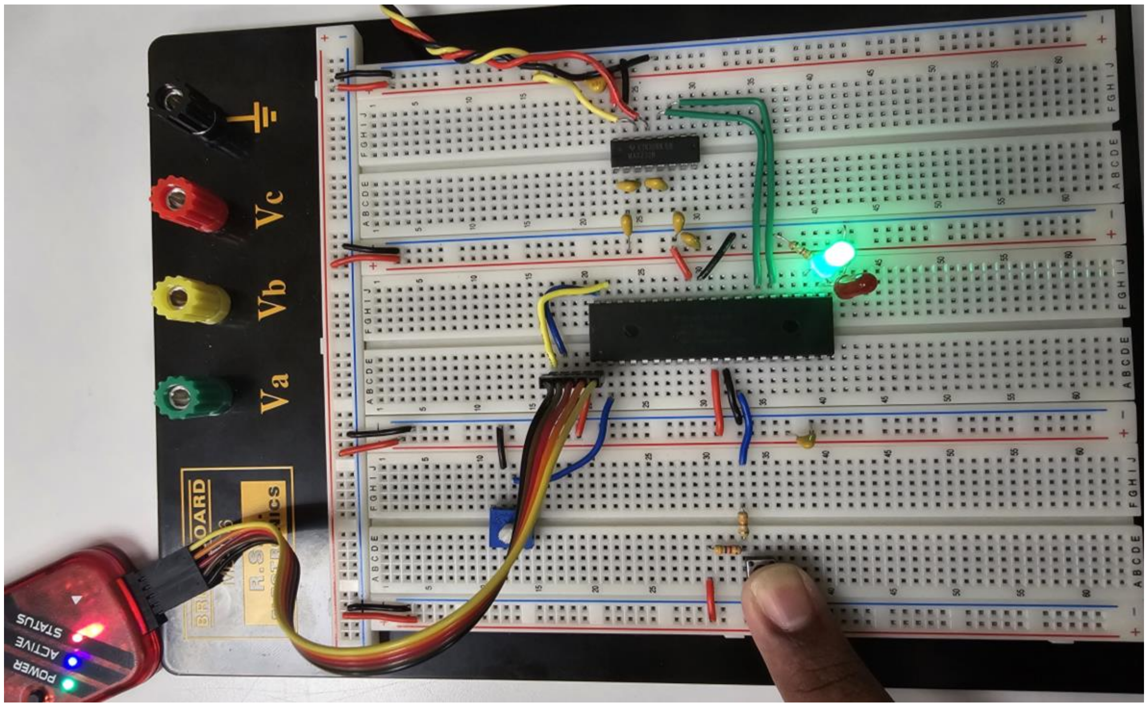
4. Testing and Data Collection: The system was tested by varying the potentiometer to simulate different temperature levels. The corresponding temperature was observed on the Tera Term display, with the LED indicating the operational status of the conversion.

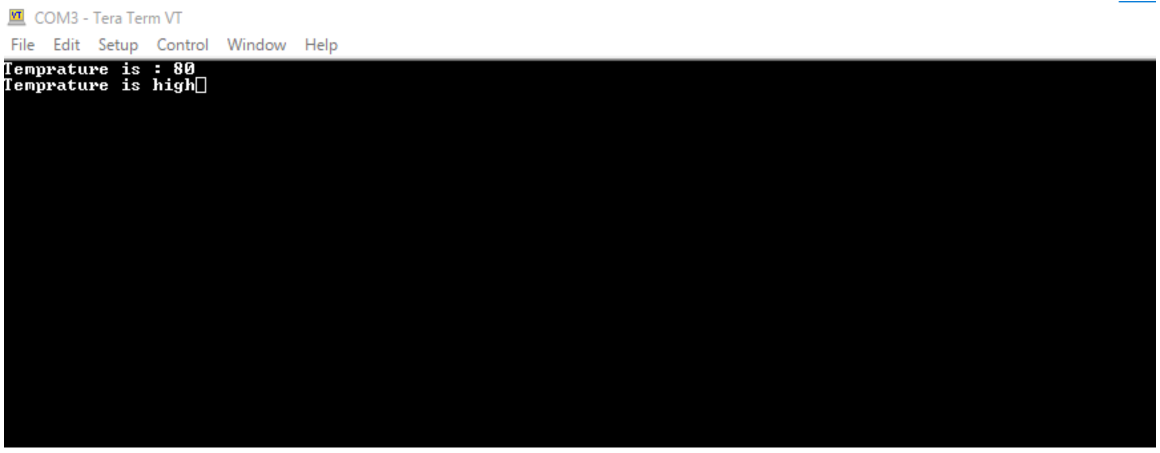
Results and Observations

Temperature Display: The temperature displayed on Tera Term correctly reflected the voltage variations induced by the potentiometer.

LED Indicators: The LEDs functioned as expected, with the red LED lighting up when the temperature is high and the green LED when temperature is low.

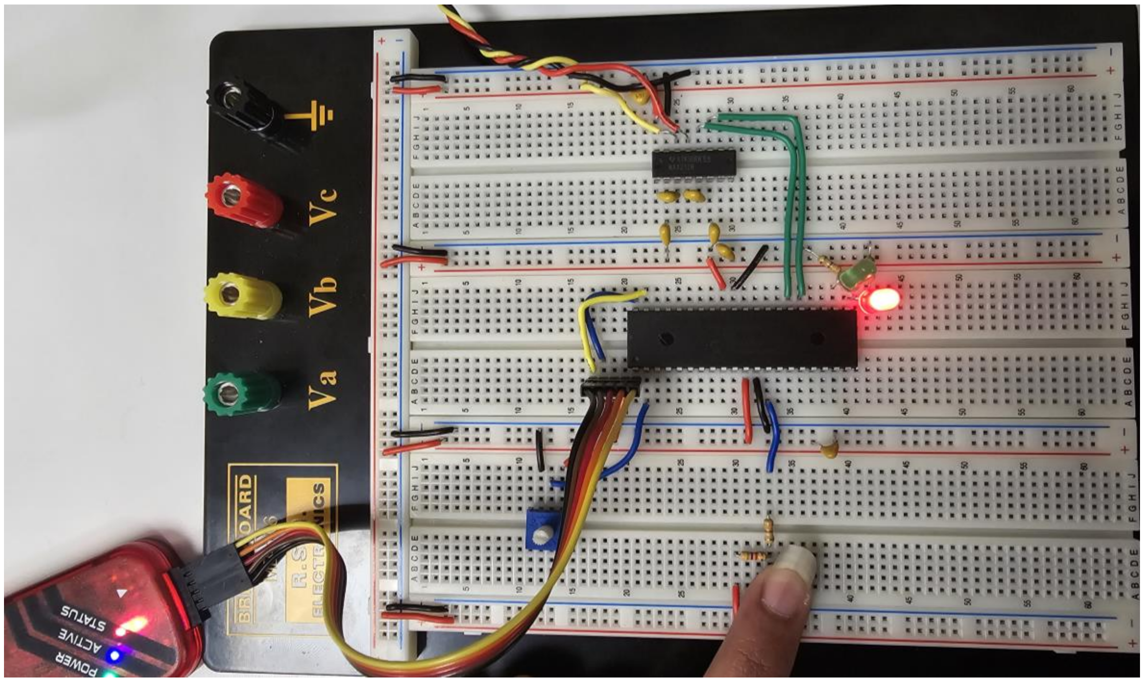
Below are images depicting the system's operation:

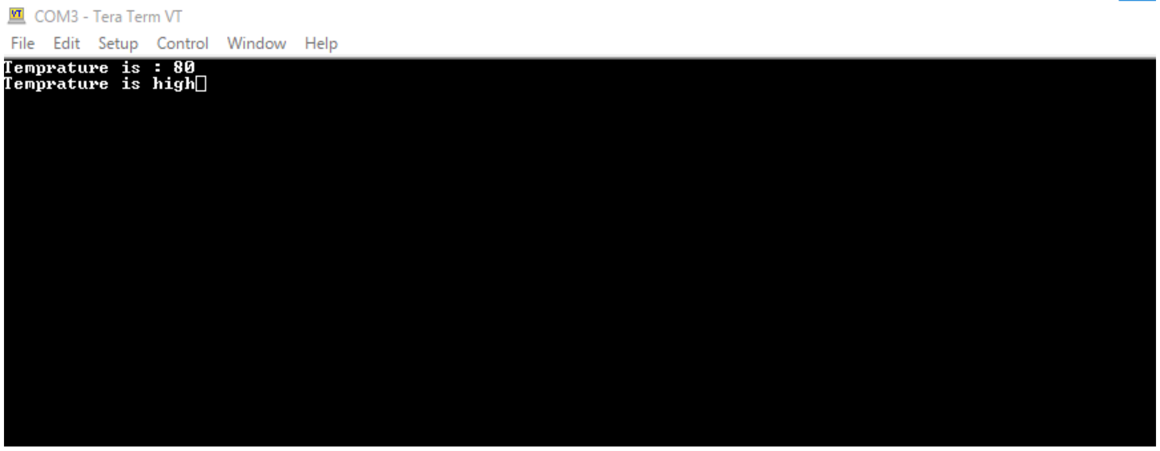




```
Temprature is : 80
Temprature is high
```

Fig2. Temperature is Low (green led on)





```
Temprature is : 80
Temprature is high
```

Fig3. Temperature is High (Red led on)

Analysis and Discussion

The system's behavior was consistent with the theoretical expectations. The ADC module accurately converted the analog voltage variations into digital temperature readings. The USART interface effectively communicated these readings to the Tera Term display, providing real-time feedback on temperature levels. The LED indicators served as a simple yet effective means of providing visual status updates.

One potential area for improvement identified during the analysis was the calibration of the potentiometer and the corresponding temperature readings. Slight variations in the potentiometer could lead to significant changes in the temperature output, suggesting that the system might benefit from more precise calibration or the use of a more sensitive temperature sensor.

Another area of interest was the response time of the system. While the system was able to display temperature readings quickly, there was a noticeable lag when the input voltage was changed rapidly. This lag could be attributed to the processing time of the microcontroller or the communication speed of the USART interface. Future iterations of the project could explore optimizing these aspects.

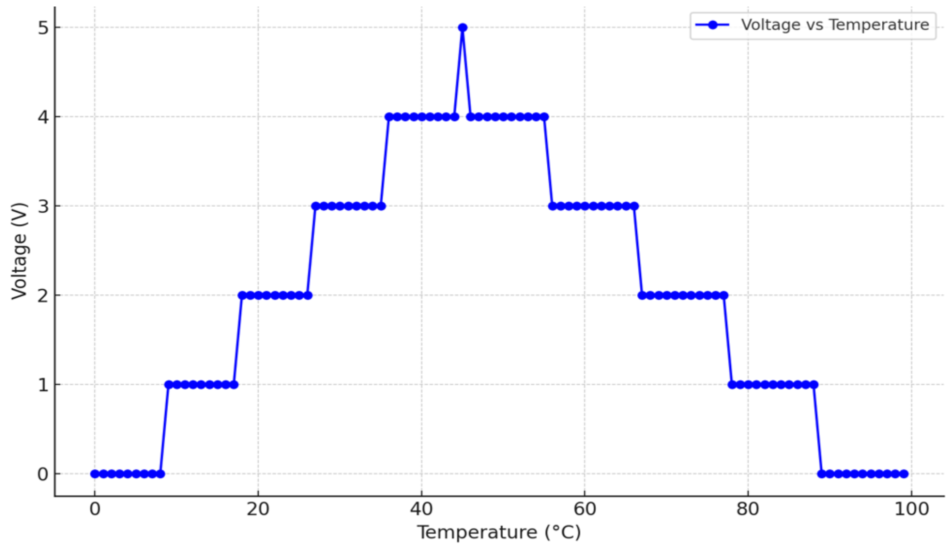


Fig 4. Voltage vs Temperature

The graph depicts the relationship between temperature and voltage in a system where, initially, as the temperature increases from 0°C to 45°C, the voltage output rises linearly from 0V to 5V, indicating a direct proportionality. Beyond 45°C, as the temperature continues to rise from 46°C to 99°C, the voltage output decreases linearly from 5V back to 0V. This behavior suggests a system designed to increase voltage with rising temperature up to a threshold, after which the voltage decreases, potentially indicating different operational responses to varying temperature ranges.

Conclusion

The project successfully demonstrated the temperature measurement system using the PIC18F45K22 microcontroller. The system performed as expected, with accurate temperature readings displayed via the USART interface and appropriate LED indications. This project effectively applied the microcontroller principles learned in the course, integrating both analog and digital inputs to control the output.

The analysis of the system's performance highlighted several strengths, including the reliability of the ADC conversion and the effectiveness of the USART communication. However, it also identified areas for potential improvement, such as the calibration of the input sensor and the response time of the system. These insights provide valuable direction for future development and optimization of similar systems.

Recommendations

Future improvements could include calibrating the system for more precise temperature readings and incorporating additional sensors to measure environmental factors such as humidity. Further experiments could explore the impact of different input configurations and microcontroller settings on system performance.

Additionally, exploring alternative communication protocols, such as I2C or SPI, could enhance the speed and reliability of data transmission. Implementing a feedback loop to automatically adjust the input parameters based on the output readings could also improve the system's accuracy and adaptability to changing conditions.

References

https://ww1.microchip.com/downloads/en/DeviceDoc/40001412G.pdf

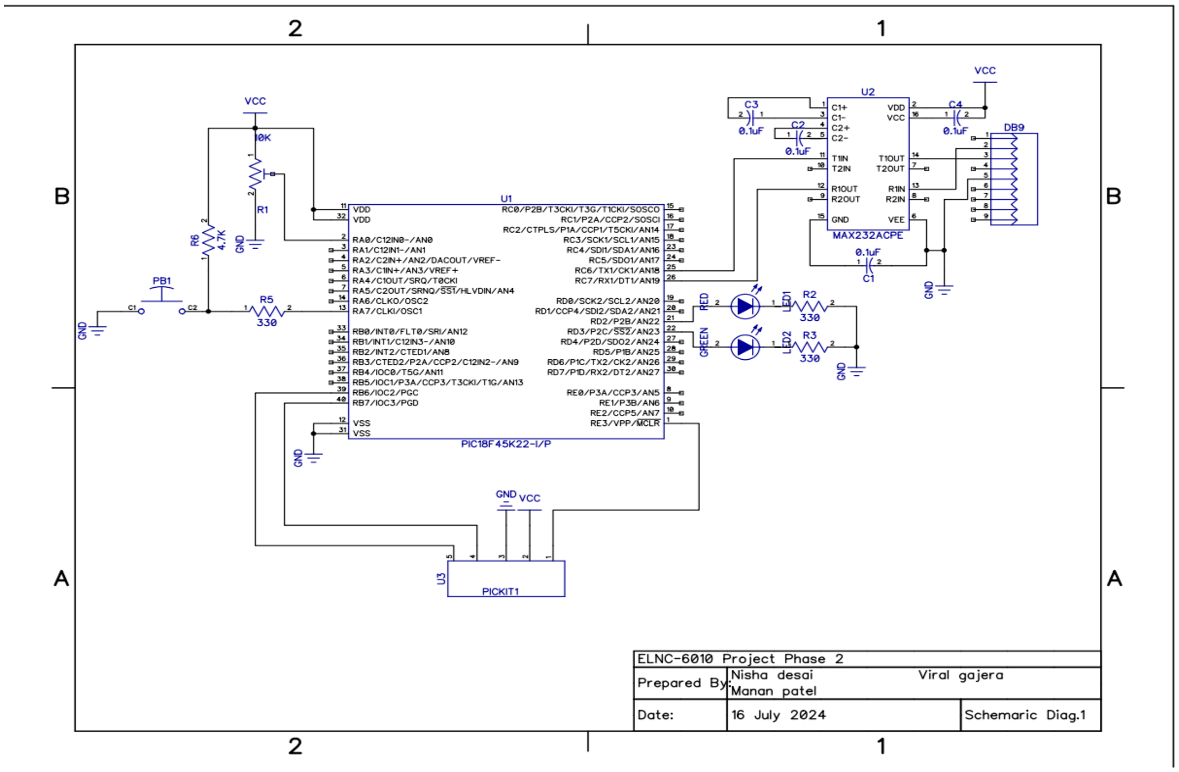
https://www.sciencedirect.com/topics/earth-and-planetary-sciences/temperature-sensor#:~:text=A%20temperature%20sensor%20is%20an,conditioners%2C%20and%20water%20temperature%20monitoring.

https://nusmods.com/courses/EE2028/microcontroller-programming-and-interfacing

Appendices

Appendix A: Schematic Diagram

Refer to the attached schematic diagram for the circuit layout.



Appendix B: Code

The full code used for this project is provided below:

/*Use of AI / Cognitive Assistance Software is not allowed in any evaluation, assessment or exercise.*/

/*==

File Name: ELNC6013ND.c

Author: Nishaben Desai

Date: 19/06/2024

Modified: None

 Fanshawe College, 2024

```
  Description: This program will generate clock using USART
===========================================================================*/


/* Preprocessor ===========================================================
  Hardware Configuration Bits ==========================================*/
#pragma config FOSC     = INTIO67
#pragma config PLLCFG   = OFF
#pragma config PRICLKEN = ON
#pragma config FCMEN    = OFF
#pragma config IESO     = OFF
#pragma config PWRTEN   = OFF
#pragma config BOREN    = ON
#pragma config BORV     = 285
#pragma config WDTEN    = OFF
#pragma config PBADEN   = OFF
#pragma config LVP      = OFF
#pragma config MCLRE    = EXTMCLR


// Libraries ==============================================================
#include <p18f45k22.h>
#include <stdio.h>
#include <stdlib.h>


// Constants  =============================================================
#define TRUE    1

#define FALSE   0

#define LED1    LATDbits.LATD2

#define LED2    LATDbits.LATD3

#define PB1 PORTAbits RA7

#define PBMASK 0x80

#define PB1PRESS 0x00

#define BYTE 8

#define LOWVALUE 25

#define ADCCHANNEL 0

#define TEMPCHAN 0

#define LEVEL1 45


// Functions  ==================================================================


/*>>> setoscillator: =========================================================

Author:     Nishaben Desai

Date:       19/06/2024

Modified:   None

Desc:       This function will configure the OSC module by setting 4Mhz frequency

            and wait for the frequency to be stable.

Input:      None

Returns:    None

 ==========================================================================*/

void setOscillator(void)
```

```
{
   OSCCON = 0x52;
   while(!OSCCONbits.HFIOFS);

} // eo setOscillator::

/*>>> configurePorts: ===========================================================
Author:    Nishaben Desai
Date:      19/06/2024
Modified:  None
Desc:      This function will configure the I/O port modules.
Input:     None
Returns:   None
 ==============================================================================*/
void configurePorts(void)
{
   ANSELA = 0x01;  //set all pins on port A to analog operation
   LATA = 0x00;    //output voltage is 0
   TRISA = 0xFF;   //set input 1

   ANSELB = 0x00;  //set all pins on port B to digital operation
   LATB = 0x00;    //output voltage is 0
   TRISB = 0xFF;   //set input 1

   ANSELC = 0x00;  //set all pins on port C to digital operation
   LATC = 0x00;    //output voltage is 0

   TRISC = 0xFF;   //set input 1


   ANSELD = 0x00;  //set all pins on port D to digital operation

   LATD = 0x00;    //output voltage is 0

   TRISD = 0xF0;   //set input 1


   ANSELE = 0x00;  //set all pins on port E to digital operation

   LATE = 0x00;    //output voltage is 0

   TRISE = 0xFF;   //set input 1


} // eo configurePorts::


/*>>> sp1Config: ===========================================================

Author:     Nishaben Desai

Date:       19/06/2024

Modified:   None

Desc:       This function will on Serial port,TX & RX enabled, 8bit transmission,

            No parity, 9600 baud rate.

Input:      None

Returns:    None

 ============================================================================*/

void sp1Config()

{

   SPBRG1 = LOWVALUE; //low byte
   BAUDCON1 = 0X40;

   RCSTA1 = 0X90;

   TXSTA1 = 0X26;

}//eo sp1Config::


/*>>> configADC: ===========================================================

Author:    Nishaben Desai

Date:      05/06/2024

Modified:  None

Desc:      This function will configure the ADC modules.

Input:     None

Returns:   None

 ==============================================================================*/

void configADC(void)

{

   ADCON0=0x01;    //turn on ADC module

   ADCON1=0x00;    //select the refrence

   ADCON2=0xA9;    //set the clock &time and justify right

}//eo configADC::


/*>>>                                                              systemInititalizion:
===========================================================

Author:    Nishaben Desai

Date:      19/06/2024

Modified:  None

Desc:      This function will call all three functions,which are setOscillator,
           configurePorts,configADC.
Input:     None
Returns:   None
 ================================================================================*/
void systemInititalizion()
{
   setOscillator();
   configurePorts();
   sp1Config();
   configADC();

}// systemInititalizion::

/*>>> getADCSample: ==========================================================
Author:    Nishaben Desai
Date:      05/06/2024
Modified:  None
Desc:      This function will sample from the ADC modules.
Input:     char
Returns:   int
 ================================================================================*/
int getADCSample(char chan)
{
   ADCON0bits.CHS=chan; //select channel
   ADCON0bits.GO=TRUE; //start conversion
   while(ADCON0bits.GO);   //conversion finish go

   return ADRES;       //return value


}//eo getADCSample::


/*=== MAIN: FUNCTION ===================================================

 ==============================================================================*/

void main( void )

{

   char pushbutton = 0;

   int adcResult = 0;

   int temresult = 0;

   systemInititalizion();

   while(TRUE)

   {

      pushbutton = PORTA & PBMASK;

      switch(pushbutton)

      {

            case PB1PRESS:

            adcResult = getADCSample(TEMPCHAN);

            //intf("%d",adcResult);

            temresult = adcResult * 0.09765625;

            printf("\033[1\033[1f\rTemprature is : %d" , temresult);

            if( temresult < LEVEL1)

            {
                LED2 = TRUE;//green led on
                LED1 = FALSE;
                printf("\033[2\033[2H");
                printf("\033[2\033[2f");
                printf("\033[2K\033[2fTemprature is low");
            }
            else
            {
                LED1 = TRUE; //red led on
                LED2 = FALSE;
                printf("\033[2\033[2H");
                printf("\033[2\033[2f");
                printf("\033[2K\033[2fTemprature is high");

            }
            break;
        default:
                LED1 = FALSE; //red led on
                LED2 = FALSE;
            break;
    }//eo switch*
  }//eo while
} // eo main::
```