BME 538: Microprocessor Systems

Course Project: Interfacing Analog Temperature Sensor with PIC24 MCU

Abie Thattamparambil

I. ABSTRACT

This laboratory project focuses on the interfacing of the TMP36 analog temperature sensor with a PIC24 microcontroller unit (MCU) using Proteus 8.11 VSM viewer. The primary objective is to gain hands-on experience in setting up analog-to-digital converters (ADCs) and configuring UART for asynchronous communication. The TMP36, known for its high precision and wide temperature range, outputs an analog voltage proportional to ambient temperature. The project involves configuring the PIC24FJ128GA010, establishing a UART connection, and displaying temperature readings on the Proteus Virtual Terminal. The report details the setup process, including clock configurations, UART settings, ADC setup, and timer configurations for periodic readings. By the end of the laboratory, students will have enhanced their skills in microcontroller interfacing and asynchronous communication.

II. INTRODUCTION

Maintaining precise temperature control is crucial in various applications such as drug fabrication, liquid heating, and equipment sterilization. The TMP36 device provides temperature readings across a broad range, making it suitable for biomedical applications. In this project, the focus is on interfacing the TMP36 temperature sensor with a PIC24 microcontroller to ensure accurate temperature detection and depiction. In order to achieve this, a 10-bit ADC will be utilized to convert the analog input from the TMP36 device and convert it into a digital code and process it in order to communicate the results via UART to the Proteus Virtual Terminal. The 10-bit ADC will be configured for manual sampling, auto conversion, & signed integer output format; furthermore, the AVDD and AVSS references must also be utilized along side the ADC clock which is set to 625 kHz in order to ensure a proper calculation and processing of the temperature from the TMP36 device. Additionally, the UART module must also be configured to a low speed mode, with 8 data bits, no parity and 1 stop bit as well as a baud rate of 19200. Lastly, a DC fan must also be configured to spin either clockwise or anticlockwise depending on the temperature acquired from the ADC conversion of the analog signal from the TMP36 device. Overall, through this, the code will be able to accurately process and depict the temperature from the TMP36 device and communicate it to the proteus virtual terminal via the asynchronous serial communication UART module and thus change the direction of the DC fan.

III. METHODS

The main components of this code are the configuration of the UART module and the configuration of the ADC function in order to acquire and display the temperature via the proteus virtual terminal. However, other functions such as (but not limited to) a timer, and, if-else statements are also utilized and configured in order to help achieve the overall objective of this lab.

```
#include "xc.h"
#include <p24FJ128GA010.h>
#include <string.h>
#include <stdio.h>
#pragma config FNOSC=FRCPLL
```

The section above are the header files used within the code which allow for specific register definitions and configurations to be utilized within the code. The pragma function sets the primary oscillator to Fast RC with the PLL module.

```
void InitializeUART();
void TransmitData(const char *data, uint32_t length);
void InitializeADC();
void ControlFan(float temperature);
int16_t ReadADCValue(void);
char OutputBuffer[1024];
```

This section of the code declares function prototypes for functions defined later in the code. It helps the compiler understand the functions before they are used.

```
void __attribute__((__interrupt__, no_auto_psv)) _U1TXInterrupt(void) {
   IFS0bits.U1TXIF = 0;
}

void __attribute__((__interrupt__, no_auto_psv)) _U1RXInterrupt(void) {
   IFS0bits.U1RXIF = 0;
}
```

These are interrupt service routines for UART transmit and receive interrupts. They clear the corresponding interrupt flags.

```
int main(void) {
   int16_t sensorValue;
   float scaledSensorValue;
   float currentTemperature;

   TRISBbits.TRISB3 = 1;
   CNPU1 = 0x0000;
   TRISE = 0;
   PORTE = 0;

   U1MODEbits.UARTEN = 0;
   InitializeUART();
   InitializeADC();
   IEC0bits.U1TXIE = 1;
   U1MODEbits.UARTEN = 1;
   U1STAbits.UTXEN = 1;
   T2CON = 0x8030;
```

```
    PR2 = 31249;

    while (1) {
        sensorValue = ReadADCValue();
        scaledSensorValue = (5.0 / 1024) * sensorValue;
        currentTemperature = (scaledSensorValue - 0.5) * 100;

        sprintf(OutputBuffer, "\r\nCurrent Temperature: %3.1f Celsius\r\n", (double)
currentTemperature);
        TransmitData(OutputBuffer, strlen(OutputBuffer));
        ControlFan(currentTemperature);
        LATD = sensorValue >> 2;
        while (TMR2);
    }

    U1MODEbits.UARTEN = 0;
    return 0;
}
```

This part of the code is the main function which does a variety of tasks that are vital to the system. It sets up communication via UART for serial transmission, initializes an analog temperature sensor through the Analog-to-Digital Converter (ADC), and configures output ports. It also configures Timer 2 to be utilized in order to create a 500 millisecond delay to help with the data processing. Inside the main loop, it continuously reads the analog sensor's output, calculates the corresponding temperature in Celsius, transmits this temperature information over UART, controls a fan's rotation direction based on the temperature, and updates an LED display. In the Current temperature calculation, the scaled sensor value is subtracted by 0.5V in order to account for the TMP36 devices offset. Overall, the program runs indefinitely, providing a continuous monitoring and feedback system for the temperature sensor, fan control, and visual indication through the Virtual Terminal.

```
void InitializeUART() {
    U1MODEbits.STSEL = 0;
    U1MODEbits.PDSEL = 0;
    U1MODEbits.BRGH = 0;
    U1BRG = (int) ((32000000UL / 2 / 19200) / 16) - 1;

    U1STAbits.UTXISEL0 = 0;
    U1STAbits.UTXISEL1 = 0;
    U1STAbits.URXISEL0 = 0;
    U1STAbits.URXISEL1 = 0;

    U1MODEbits.UARTEN = 1;
}
```

This function configures the UART module for communication. It sets the number of stop bits, parity bits, and the baud rate in accordance with the course projects specifications.

```
void TransmitData(const char *data, uint32_t length) {
    while (length) {
        while (!U1STAbits.TRMT);
        U1TXREG = *data;
        data++;
        length--;
    }
    while (!U1STAbits.TRMT);
}
```

This function sends a given data buffer of specified length over UART. It waits for the transmitter to be empty before sending each character in the buffer, ensuring that the transmission is complete before exiting the function.

```
void InitializeADC() {
   AD1CON2 = 0;
   AD1CHS = 0x0003;
   AD1PCFG = 0xFFF7;
   AD1CSSL = 0;
   AD1CON1 = 0x00E0;
   AD1CON3 = 0x1F02;
   IFS0bits.AD1IF = 0;
   AD1CON1bits.ADON = 1;
}
```

This function sets up the Analog-to-Digital Converter (ADC). It configures the ADC channel, digital input buffers, and conversion triggers. The ADC is enabled, and the interrupt flag is cleared to prepare for conversion.

```
int16_t ReadADCValue(void) {
   AD1CON1bits.SAMP = 1;
   while (!AD1CON1bits.DONE);
   return ADC1BUF0;
}
```

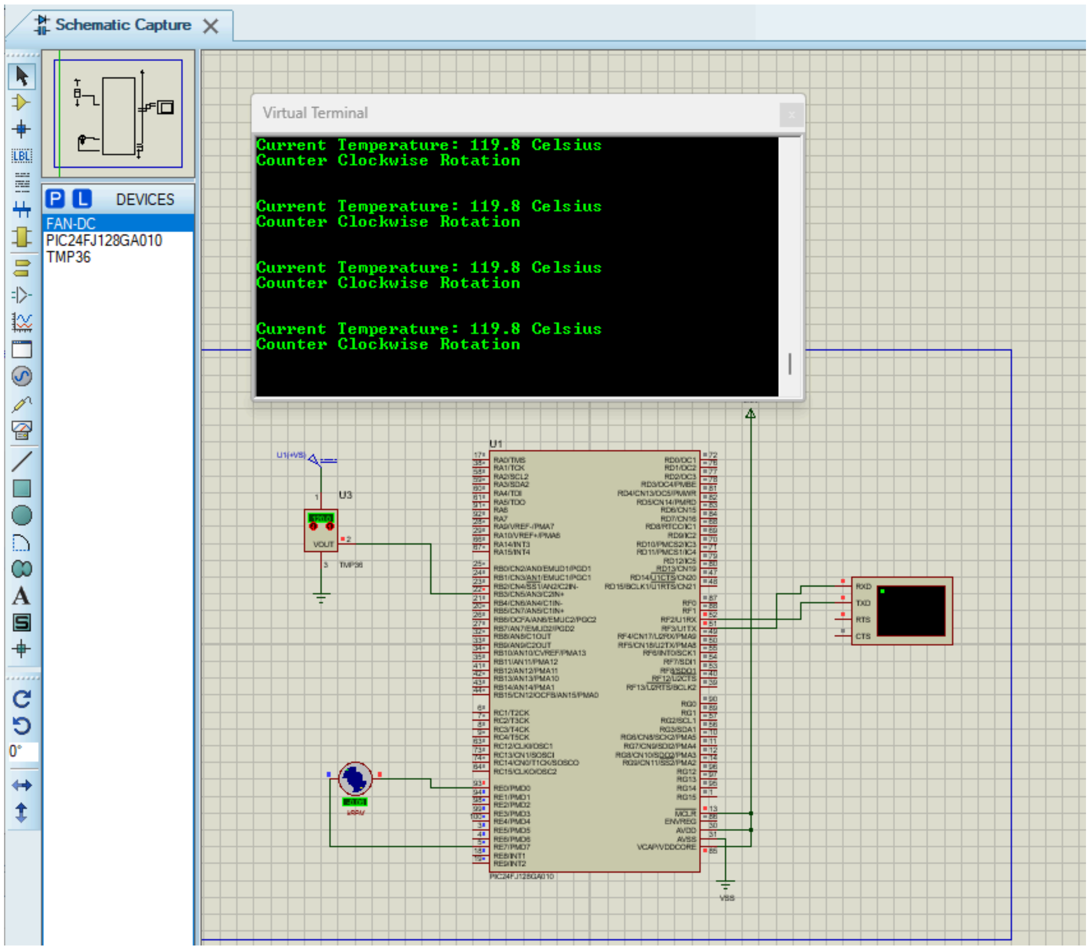
Reads the value from the ADC after initiating a sampling. It waits until the conversion is complete and then returns the 10-bit ADC result from the buffer.

```
void ControlFan(float temperature) {
   if (temperature < 30) {
      _RE0 = 0;
      _RE7 = 1;
      sprintf(OutputBuffer, "Clockwise Rotation\r\n\r\n");
      TransmitData(OutputBuffer, strlen(OutputBuffer));
   } else {
      _RE0 = 1;
      _RE7 = 0;
      sprintf(OutputBuffer, "Counter Clockwise Rotation\r\n\r\n");
      TransmitData(OutputBuffer, strlen(OutputBuffer));
  }
}
```

This function takes a temperature value as input and controls a fan's rotation direction based on that temperature. If the temperature is below 30 degrees Celsius, it sets specific output pins to initiate a clockwise rotation and transmits a corresponding message over UART. Otherwise, it sets different pins for counterclockwise rotation and transmits an alternative message over UART.

IV. RESULTS AND DISCUSSION

Screenshots depicting Virtual Terminal Display for the following temperatures: 120ºC, 20ºC, -20ºC, -40ºC



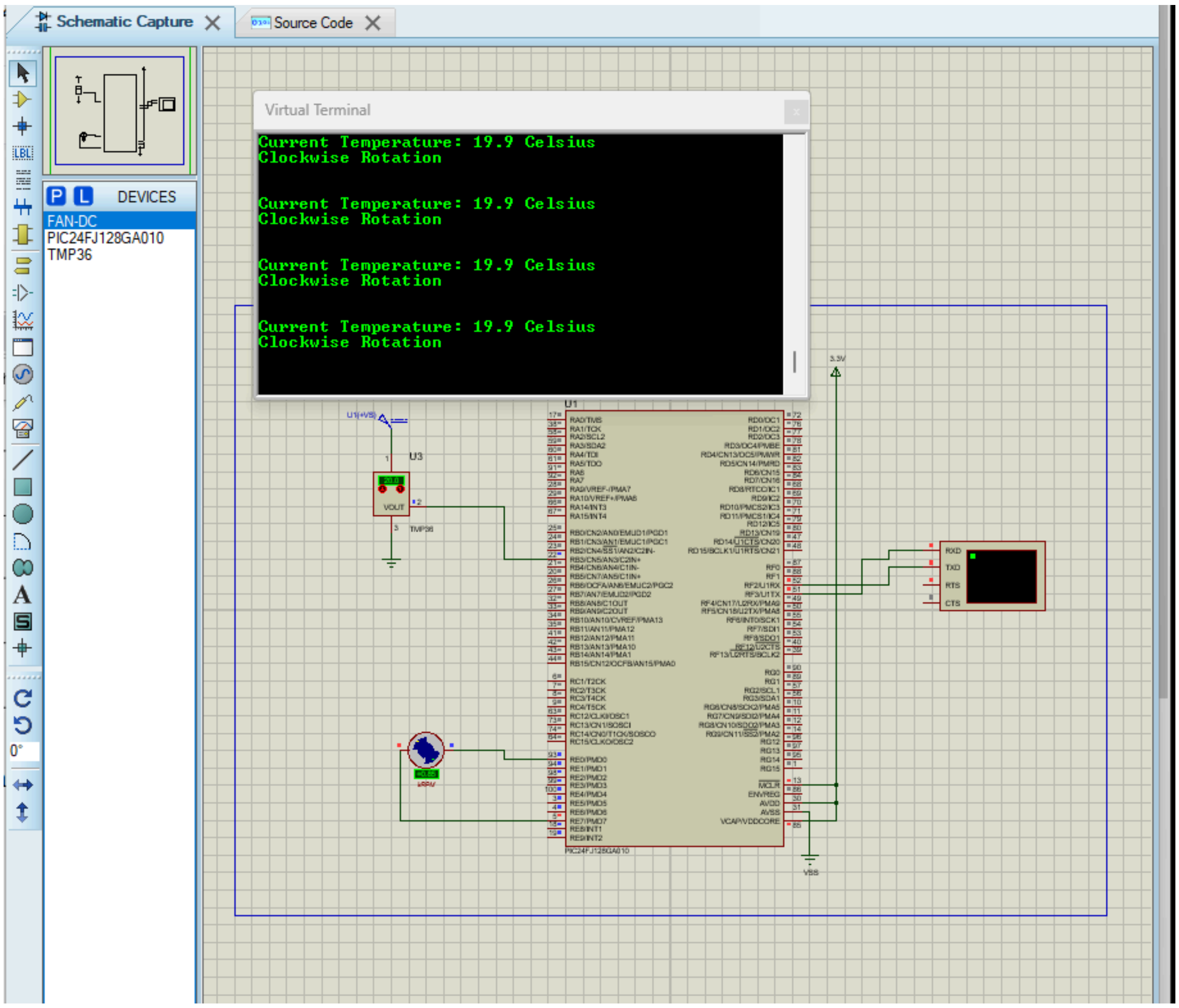
```
Current Temperature: 119.8 Celsius
Counter Clockwise Rotation


Current Temperature: 119.8 Celsius
Counter Clockwise Rotation


Current Temperature: 119.8 Celsius
Counter Clockwise Rotation


Current Temperature: 119.8 Celsius
Counter Clockwise Rotation
```

Figure 1. Virtual Terminal Displaying 120ºC



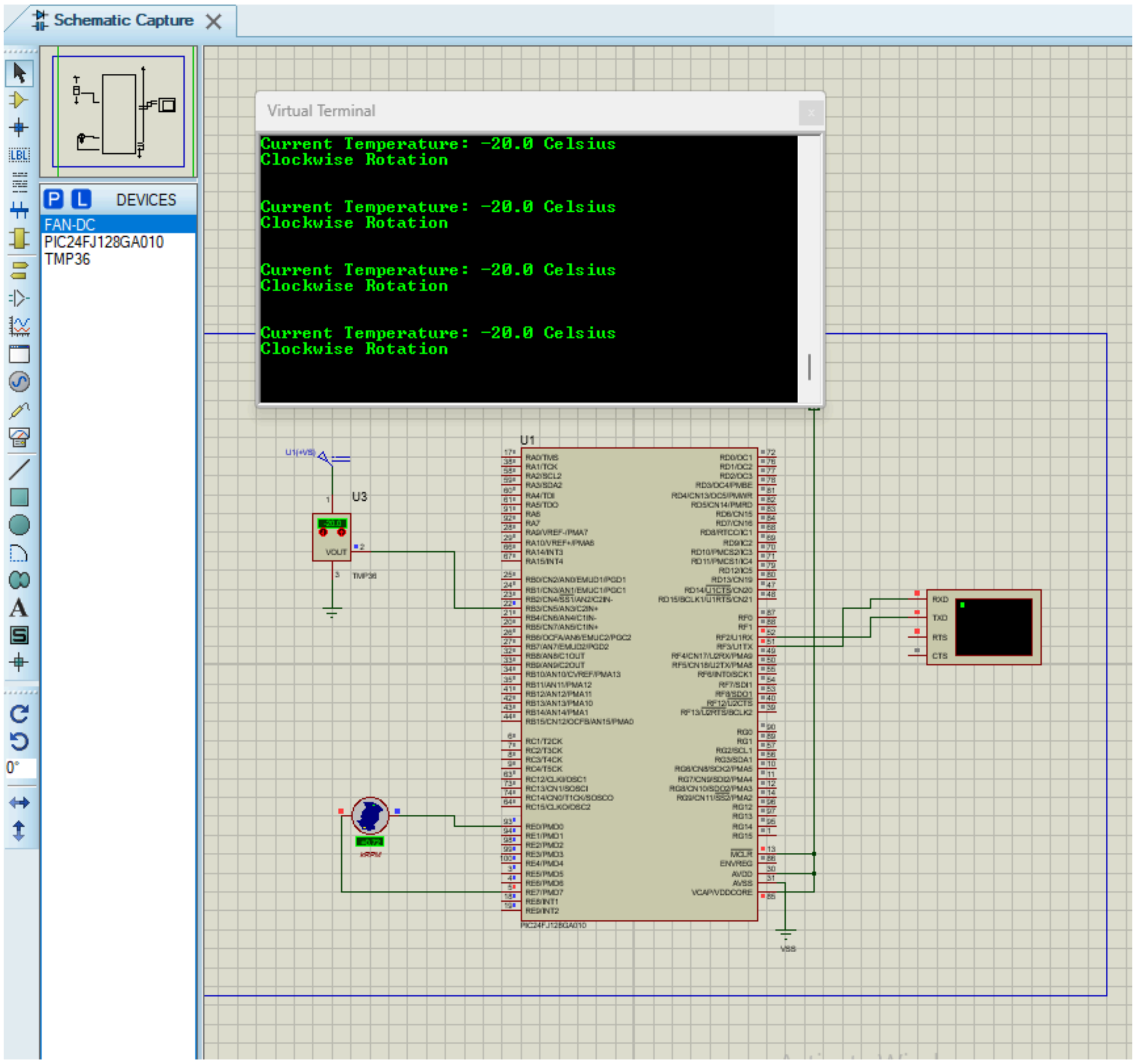
```
Current Temperature: 19.9 Celsius
Clockwise Rotation


Current Temperature: 19.9 Celsius
Clockwise Rotation


Current Temperature: 19.9 Celsius
Clockwise Rotation


Current Temperature: 19.9 Celsius
Clockwise Rotation
```

Figure 2 Virtual Terminal Displaying 20ºC



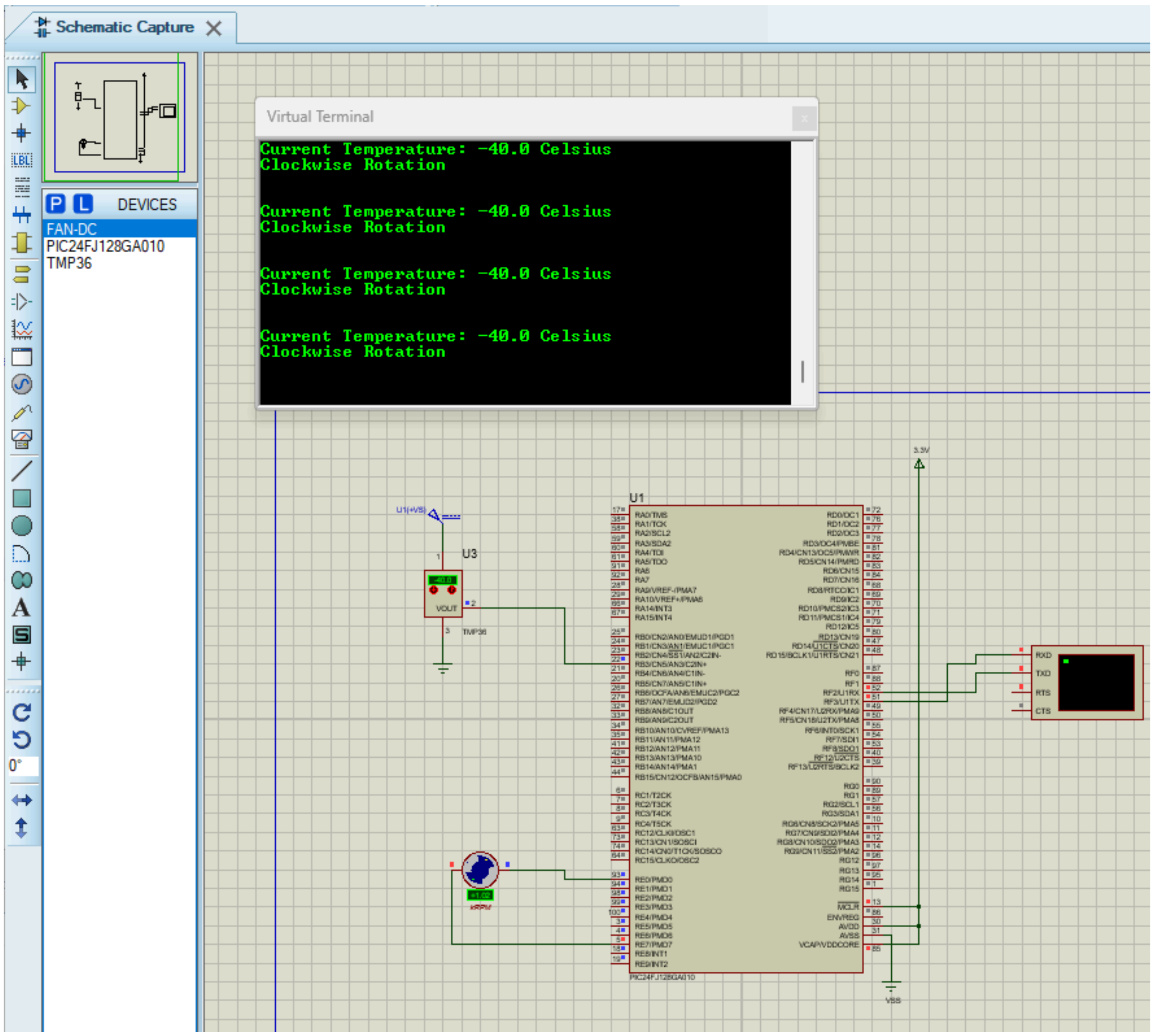
```
Current Temperature: -20.0 Celsius
Clockwise Rotation


Current Temperature: -20.0 Celsius
Clockwise Rotation


Current Temperature: -20.0 Celsius
Clockwise Rotation


Current Temperature: -20.0 Celsius
Clockwise Rotation
```

Figure 3. Virtual Terminal Displaying -20ºC



```
Current Temperature: -40.0 Celsius
Clockwise Rotation

Current Temperature: -40.0 Celsius
Clockwise Rotation

Current Temperature: -40.0 Celsius
Clockwise Rotation

Current Temperature: -40.0 Celsius
Clockwise Rotation
```

Figure 4. Virtual Terminal Displaying -40ºC

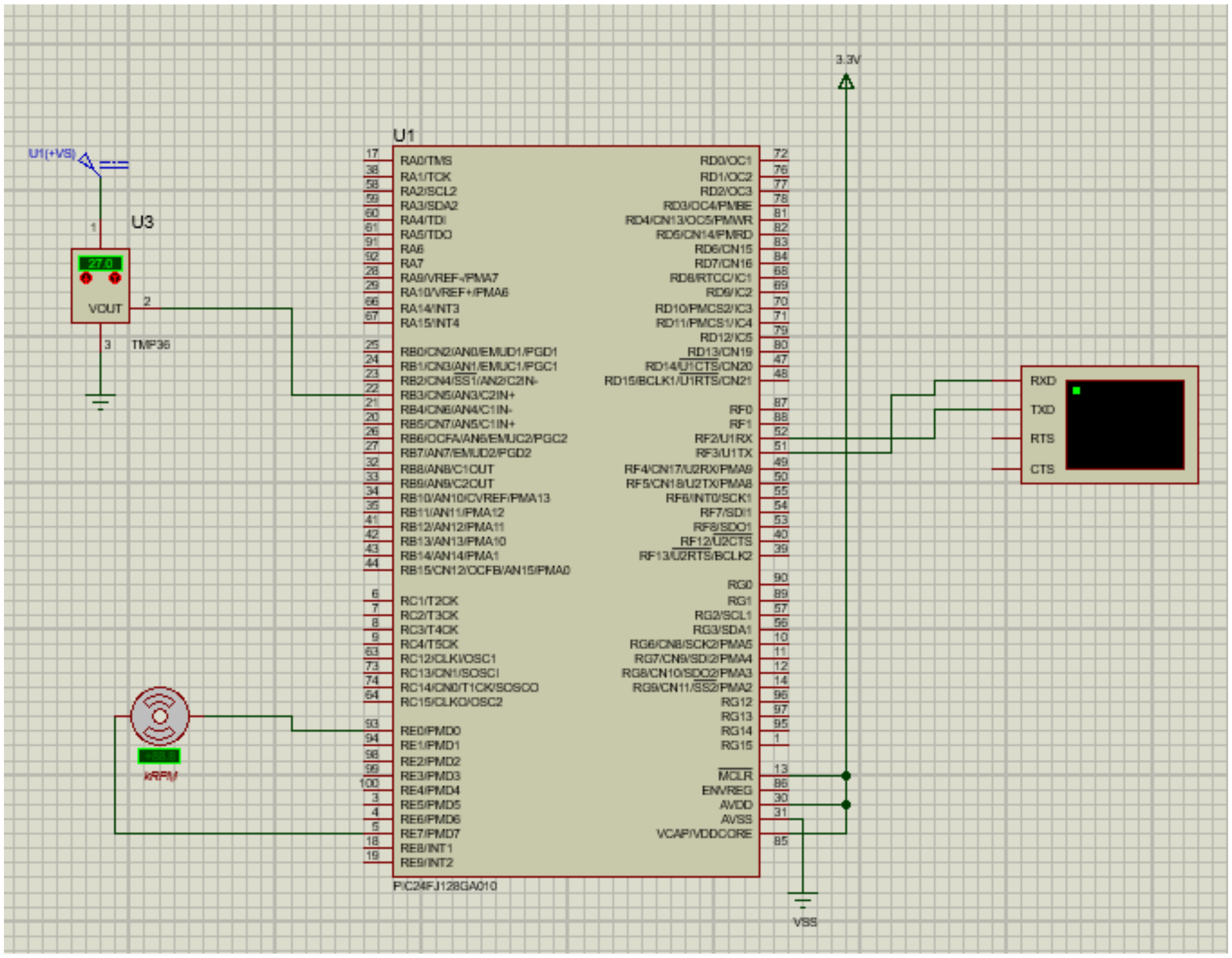
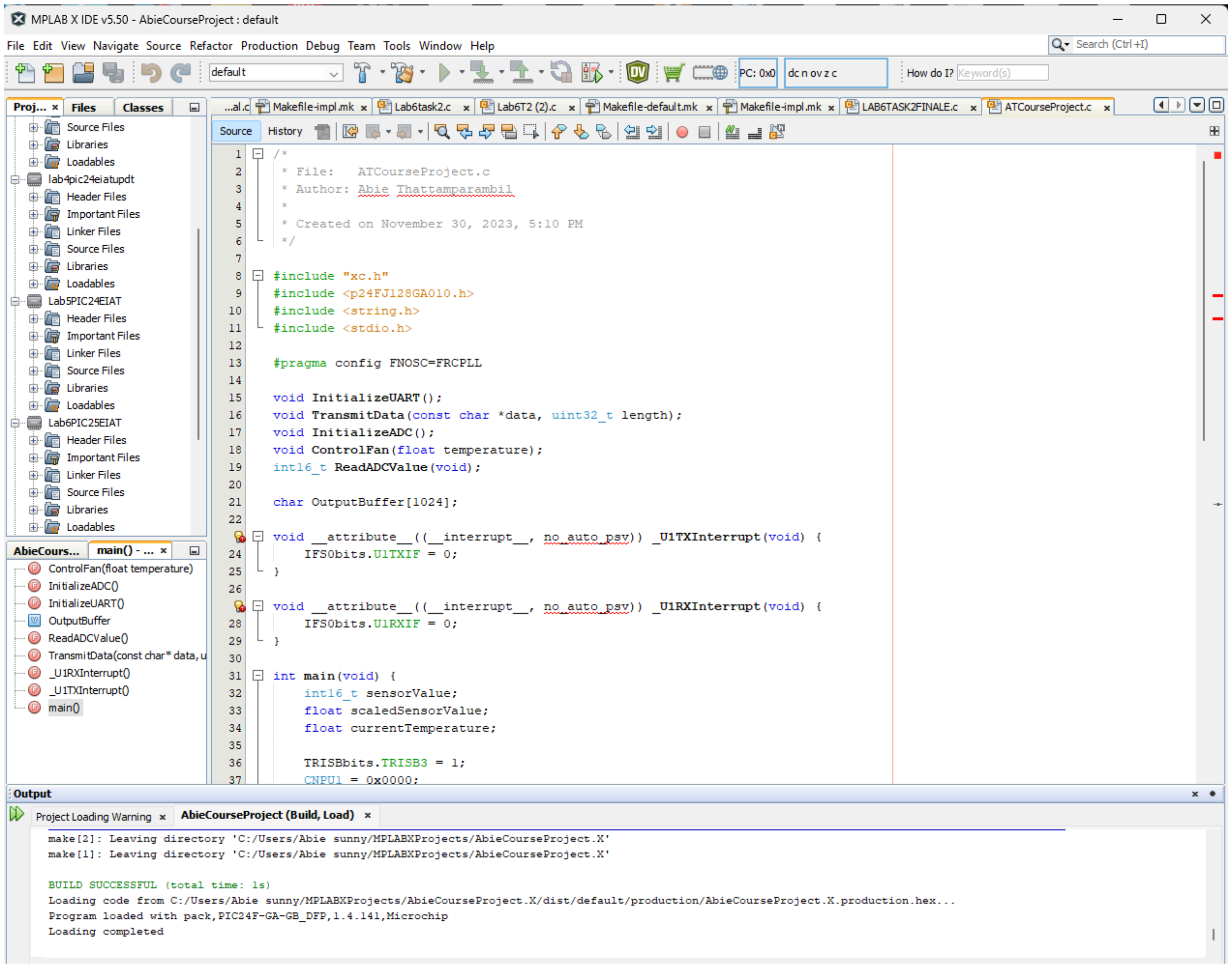


Figure 5. Schematic Capture for Course Project



```c
/*
 * File:   ATCourseProject.c
 * Author: Abie Thattamparambil
 *
 * Created on November 30, 2023, 5:10 PM
 */

#include "xc.h"
#include <p24FJ128GA010.h>
#include <string.h>
#include <stdio.h>

#pragma config FNOSC=FRCPLL

void InitializeUART();
void TransmitData(const char *data, uint32_t length);
void InitializeADC();
void ControlFan(float temperature);
int16_t ReadADCValue(void);

char OutputBuffer[1024];

void __attribute__((__interrupt__, no_auto_psv)) _U1TXInterrupt(void) {
    IFS0bits.U1TXIF = 0;
}

void __attribute__((__interrupt__, no_auto_psv)) _U1RXInterrupt(void) {
    IFS0bits.U1RXIF = 0;
}

int main(void) {
    int16_t sensorValue;
    float scaledSensorValue;
    float currentTemperature;

    TRISBbits.TRISB3 = 1;
    CNPU1 = 0x0000;
```

```
make[2]: Leaving directory 'C:/Users/Abie sunny/MPLABXProjects/AbieCourseProject.X'
make[1]: Leaving directory 'C:/Users/Abie sunny/MPLABXProjects/AbieCourseProject.X'

BUILD SUCCESSFUL (total time: 1s)
Loading code from C:/Users/Abie sunny/MPLABXProjects/AbieCourseProject.X/dist/default/production/AbieCourseProject.X.production.hex...
Program loaded with pack,PIC24F-GA-GB_DFP,1.4.141,Microchip
Loading completed
```

Figure 6. Compiled C Code for Course Project

Figures 5 and 6 depict the Proteus schematic utilized for this project and the corresponding code building successfully overall implying a lack of errors in the design for both of these elements of the project. Figures 1 through 4 on the other hand, depict the code functioning as intended and not only depicting the current temperature as per the configuration of the TMP36 device but also the direction of the fans spin which is also animated when the file is debugged. Overall, executing as intended and demonstrating an understanding of interfacing sensors.

V. CONCLUSION

In conclusion, the primary goal of this laboratory project, "Interfacing Analog Temperature Sensor with PIC24 MCU," was to master the configuration and utilization of analog-to-digital converters (ADCs) and UART communication for interfacing a temperature sensor with a microcontroller. The project required a comprehensive understanding of the nuances behind configuring clock frequencies, UART settings, ADC parameters, and timer functionalities to achieve the desired outcome. As for the results of the lab, the code utilizes the ADC in order to effectively calculate the temperature in celsius given the TMP36 as an input device and not only displays the information on the proteus virtual terminal but also utilizes said information to control the direction that the DC fan spins. To summerize, the project guided students through the process of interfacing the temperature sensor, reading analog signals, and transmitting the results asynchronously through UART; thus the objectives of the lab were met, providing a blend of theoretical understanding and practical application in the field of microcontroller-based temperature sensing and communication.

Overall, through the functionality of the code and the proteus schematic that accompanies it, it is safe to say that the overall criteria for successfully accomplishing the goals of this lab were met as well.