Ryerson University

Department of Electrical, Computer, & Biomedical Engineering
Faculty of Engineering & Architectural Science

Course Number	BME 538
Course Title	Microprocessor Systems
Semester/Year	Fall 2023
Instructor	Sattar Hussain

Lab/Tutorial Number	Course Project

Report Title	Interfacing Sensors Using I2C Protocol

Section Number	03
Group Number	05
Submission Date	November 27, 2023
Due Date	December 1, 2023

Name	**Student Number**	**Signature***
Mohammed Sohaib Syed	501105890	MSS

BME 538
Course Project: Interfacing Sensors Using I2C Protocol

Mohammed Sohaib Syed

I. ABSTRACT

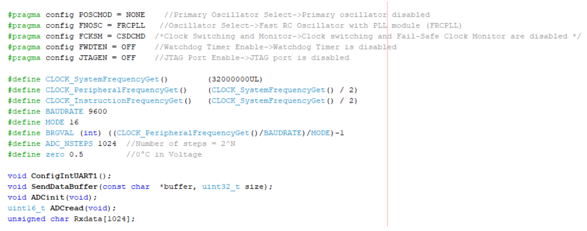
The course project revolves around interfacing the analog temperature sensor TMP36 through the use of ADC to digitize the signal and display it on a virtual terminal in Proteus 8.11 through the use of UART. The main objective of this project is to figure out how to derive a temperature reading from the ADC code received through converting the analog input from TMP36. Fan control must also be established, changing depending on whether the temperature value is above or below 30 °C.

II. INTRODUCTION

The Analog-to-Digital Converter (ADC) transforms an analog signal, such as voltage or current, into a digital code. This process involves a sampling period, which is the time interval between each conversion, and a resolution, defined by the number of bits used in the conversion. The conversion process is defined by the formula Vin/Vref x 2^N, where Vin represents the input voltage, Vref is the maximum voltage that can be measured, and N stands for the resolution (# of bits). The UART is an asynchronous communication interface that is responsible for transmitting and receiving bitwise data. As handshaking protocols are not used in this project, a baud rate is set to determine the speed at which signalling events occur. The TMP36 is a low-voltage, high-precision temperature sensor. It provides a voltage output that is linearly proportional to the Celsius temperature. Both of the PIC24 modules, ADC and UART, are required to interface the TMP36 sensor to output the correct temperature value on the virtual terminal and control the fan as intended.

III. DESIGN ANALYSIS

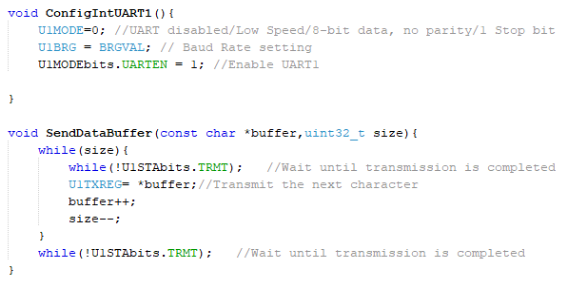
For the course project, the ADC module was programmed to read analog data from the TMP36 sensor through the RB0/AN0 pin, which was then converted into digital code and then into a temperature value. This value was then sent through UART1 to be displayed on the virtual terminal in the Proteus Schematic output.



```
#pragma config POSCMOD = NONE    //Primary Oscillator Select->Primary oscillator disabled
#pragma config FNOSC = FRCPLL   //Oscillator Select->Fast RC Oscillator with PLL module (FRCPLL)
#pragma config FCKSM = CSDCMD  /*Clock Switching and Monitor->Clock switching and Fail-Safe Clock Monitor are disabled */
#pragma config FWDTEN = OFF    //Watchdog Timer Enable->Watchdog Timer is disabled
#pragma config JTAGEN = OFF    //JTAG Port Enable->JTAG port is disabled

#define CLOCK_SystemFrequencyGet()        (32000000UL)
#define CLOCK_PeripheralFrequencyGet()    (CLOCK_SystemFrequencyGet() / 2)
#define CLOCK_InstructionFrequencyGet()   (CLOCK_SystemFrequencyGet() / 2)
#define BAUDRATE 9600
#define MODE 16
#define BRGVAL (int) ((CLOCK_PeripheralFrequencyGet()/BAUDRATE)/MODE)-1
#define ADC_NSTEPS 1024  //Number of steps = 2^N
#define zero 0.5         //0°C in Voltage

void ConfigIntUART1();
void SendDataBuffer(const char  *buffer, uint32_t size);
void ADCinit(void);
uint16_t ADCread(void);
unsigned char Rxdata[1024];
```

Figure 1: Preprocessor Configs and Macros

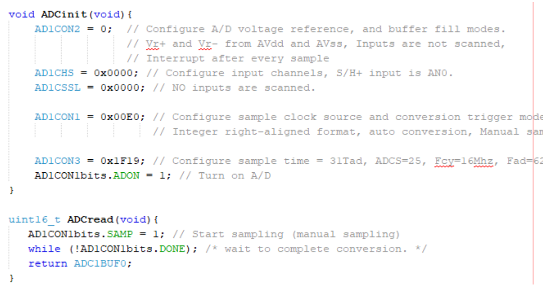
The preprocessor configurations/macros set the oscillator as FRCPLL, the system/instruction frequency as 32/16 Mhz, the baud rate (9600), low-speed mode (16), and the ADC steps (1024 for 10 bits). There are also a zero macro that are used later on in the calculation of the temperature from voltage. Then, the UART configuration, data sending, and ADC functions are prototyped. The Rx data array is also initialized to store the data that will be sent to the UART.



```
void ConfigIntUART1(){
    U1MODE=0; //UART disabled/Low Speed/8-bit data, no parity/1 Stop bit
    U1BRG = BRGVAL; // Baud Rate setting
    U1MODEbits.UARTEN = 1; //Enable UART1

}

void SendDataBuffer(const char *buffer,uint32_t size){
    while(size){
        while(!U1STAbits.TRMT);   //Wait until transmission is completed
        U1TXREG= *buffer;//Transmit the next character
        buffer++;
        size--;
    }
    while(!U1STAbits.TRMT);   //Wait until transmission is completed
}
```

Figure 2: UART Code

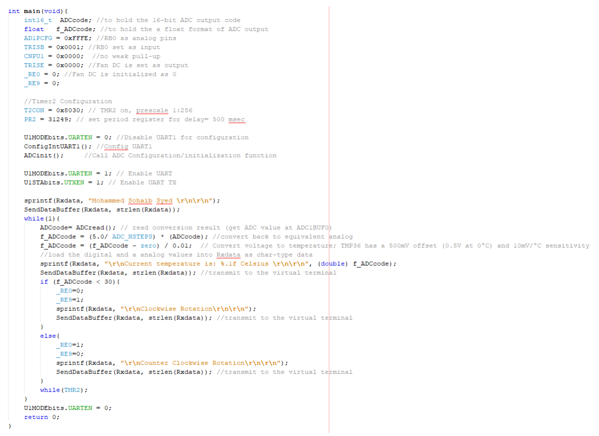
The ConfigIntUART1 function configures UART1, setting it to use one stop bit, no parity, 8 data bits, and low-speed mode. It also sets the baud rate to the BRGVAL macro, which has been defined earlier. The function then enables UART1. The SendDataBuffer function is responsible for sending the data through UART1 to be displayed on the Virtual Terminal. It takes a buffer and its size as arguments, sending each character in the buffer. After sending a character, it checks if the transmit shift register is empty, and only sends the next character if that is true.



```
void ADCinit(void){
    AD1CON2 = 0;  // Configure A/D voltage reference, and buffer fill modes.
                  // Vr+ and Vr- from AVdd and AVss, Inputs are not scanned,
                  // Interrupt after every sample
    AD1CHS = 0x0000; // Configure input channels, S/H+ input is AN0.
    AD1CSSL = 0x0000; // NO inputs are scanned.

    AD1CON1 = 0x00E0; // Configure sample clock source and conversion trigger mode.
                      // Integer right-aligned format, auto conversion, Manual sampling

    AD1CON3 = 0x1F19; // Configure sample time = 31Tad, ADCS=25, Fcy=16Mhz, Fad=625KHz
    AD1CON1bits.ADON = 1; // Turn on A/D
}

uint16_t ADCread(void){
   AD1CON1bits.SAMP = 1; // Start sampling (manual sampling)
   while (!AD1CON1bits.DONE); /* wait to complete conversion. */
   return ADC1BUF0;
}
```

Figure 3: ADC Code

The ADCinit and ADCread functions are used to initialize and read data from the ADC module. ADCinit sets up the ADC using the 5 registers: AD1CON1, AD1CON2, AD1CON3, AD1CHS, and AD1CSSL. First, AD1CON2 is configured to use AVdd/AVss as voltage references, interrupt at every conversion completion, use one 16-word buffer, and always use MUX A. Then AD1CHS is configured to use AN0 as the input channel, and AD1CSSL ignores the input channel in sequential scanning. AD1CON1 then sets the converter as off initially (good practice), outputs data in integer format, and enables manual sampling/auto conversion. Finally, AD1CON3 is configured to set sample time to 31 Tad, ADCS to 25, and AD1CON1 turns the ADC converter on. The ADCread function starts the ADC sampling manually and then waits for the conversion to complete before returning the result from the ADC buffer.

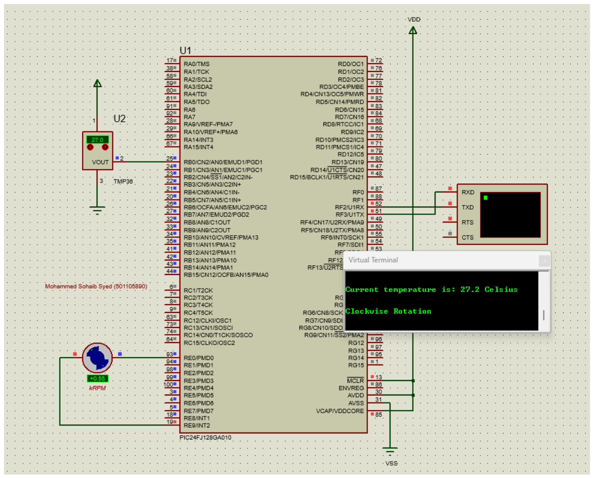


```
int main(void){
    int16_t  ADCcode; //to hold the 16-bit ADC output code
    float   f_ADCcode; //to hold the a float format of ADC output
    AD1PCFG = 0xFFFE; //RB0 as analog pins
    TRISB = 0x0001; //RB0 set as input
    CNPU1 = 0x0000;  //no weak pull-up
    TRISE = 0x0000; //Fan DC is set as output
    _RE0 = 0; //Fan DC is initialized as 0
    _RE9 = 0;

    //Timer2 Configuration
    T2CON = 0x8030; // TMR2 on, prescale 1:256
    PR2 = 31249; // set period register for delay= 500 msec

    U1MODEbits.UARTEN = 0; //Disable UART1 for configuration
    ConfigIntUART1(); //Config UART1
    ADCinit();     //Call ADC Configuration/initialization function

    U1MODEbits.UARTEN = 1; // Enable UART
    U1STAbits.UTXEN = 1; // Enable UART TX

    sprintf(Rxdata, "Mohammed Sohaib Syed \r\n\r\n");
    SendDataBuffer(Rxdata, strlen(Rxdata));
    while(1){
        ADCcode= ADCread(); // read conversion result (get ADC value at ADC1BUF0)
        f_ADCcode = (5.0/ ADC_NSTEPS) * (ADCcode); //convert back to equivalent analog
        f_ADCcode = (f_ADCcode - zero) / 0.01;  // Convert voltage to temperature; TMP36 has a 500mV offset (0.5V at 0°C) and 10mV/°C sensitivity
        //load the digital and a analog values into Rxdata as char-type data
        sprintf(Rxdata, "\r\nCurrent temperature is: %.1f Celsius \r\n\r\n", (double) f_ADCcode);
        SendDataBuffer(Rxdata, strlen(Rxdata)); //transmit to the virtual terminal
        if (f_ADCcode < 30){
            _RE0=0;
            _RE9=1;
            sprintf(Rxdata, "\r\nClockwise Rotation\r\n\r\n");
            SendDataBuffer(Rxdata, strlen(Rxdata)); //transmit to the virtual terminal
        }
        else{
            _RE0=1;
            _RE9=0;
            sprintf(Rxdata, "\r\nCounter Clockwise Rotation\r\n\r\n");
            SendDataBuffer(Rxdata, strlen(Rxdata)); //transmit to the virtual terminal
        }
        while(TMR2);
    }
    U1MODEbits.UARTEN = 0;
    return 0;
}
```

Figure 4: Main Function

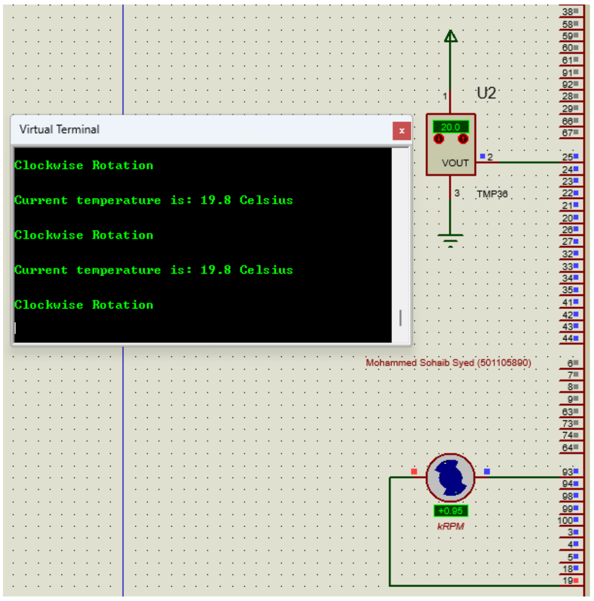
The main function begins by initializing variables to store the converted ADC values, configuring the microcontroller ports to read from pin RB0, setting pin RB0 as analog, and setting PORTD as output to control the fan. Timer 2 is then configured to set a 500-millisecond delay between successive readings. UART1 and ADC configuration functions are called, and both are now enabled. An initial welcome message is sent to the virtual terminal. In the main while loop, the function reads the ADC value, which represents the current temperature. This reading is first converted to a voltage, then to a temperature in Celsius using predefined macros that represent the voltage at 0°C and the voltage-to-temperature conversion ratio. The conversion to Celsius works as the TMP36 outputs 500 mV at 0°C, with a 10 mV/°C sensitivity. Therefore, subtracting 0.5V from the measured voltage and dividing by 0.01V/°C gives the temperature in Celsius. The fan's rotation direction is clockwise if the temperature is below 30 °C or counter-clockwise if it's above 30 °C. The control is implemented by setting digital outputs _RE0 and _RE9 (fan direction depends on which pin sends the signal). The loop also implements the delay set through Timer 2. Throughout the main function (mainly in the loop), the SendDataBuffer function is called to transmit the acquired values through the UART to the virtual terminal.

IV. RESULTS



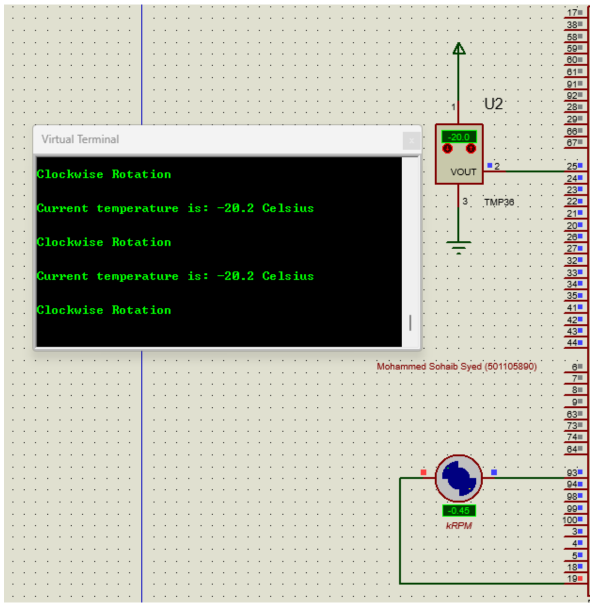
```
Current temperature is: 27.2 Celsius

Clockwise Rotation
```

Figure 9: Proteus Schematic



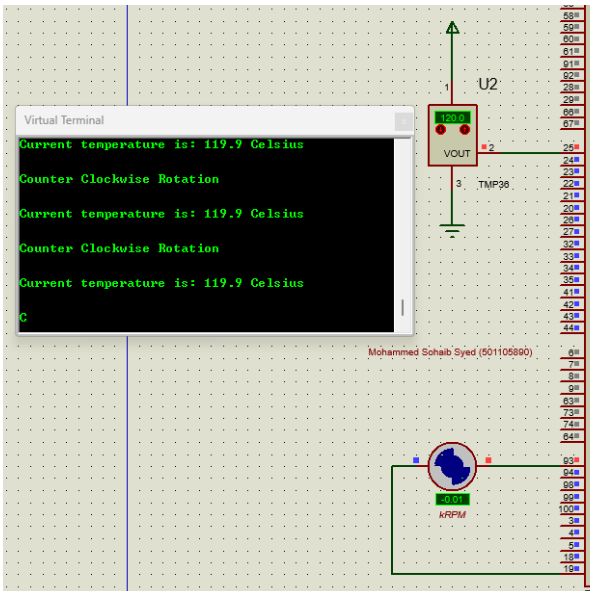
```
Clockwise Rotation
Current temperature is: 19.8 Celsius
Clockwise Rotation
Current temperature is: 19.8 Celsius
Clockwise Rotation
```

Figure 5: Temperature 20 °C



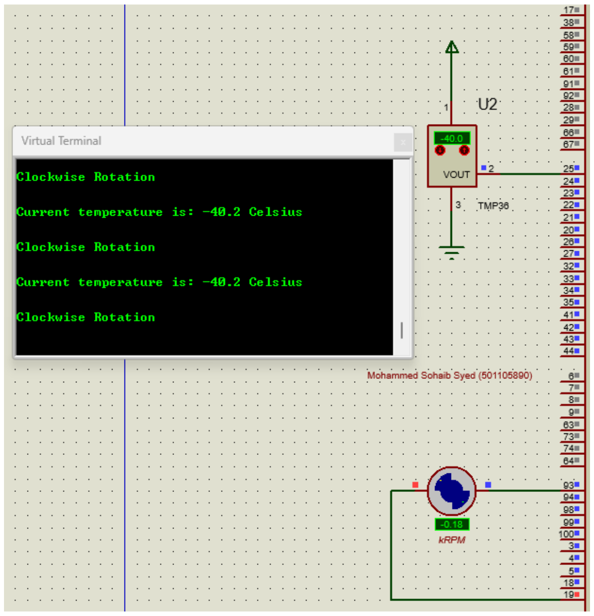
```
Clockwise Rotation
Current temperature is: -20.2 Celsius
Clockwise Rotation
Current temperature is: -20.2 Celsius
Clockwise Rotation
```

Figure 7: Temperature -20 °C



```
Current temperature is: 119.9 Celsius
Counter Clockwise Rotation
Current temperature is: 119.9 Celsius
Counter Clockwise Rotation
Current temperature is: 119.9 Celsius
C
```

Figure 6: Temperature 120 °C



```
Clockwise Rotation
Current temperature is: -40.2 Celsius
Clockwise Rotation
Current temperature is: -40.2 Celsius
Clockwise Rotation
```

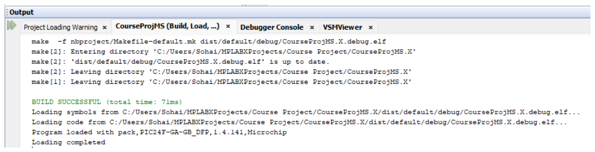
Figure 8: Temperature -40 °C

The successful output of the course project can be seen in the above figures. When the temperatures are set through the TMP36 temperature sensor, the VT outputs that temperature. The fan direction also changes depending on whether the temperature is above or below 30 °C and is also displayed on the VT. There is a discrepancy in the temperature values displayed on the VT that can be attributed to accuracy lost in rounding and the offset provided by the TMP36 sensor..

V. DISCUSSION AND CONCLUSIONS

The project was completed successfully, and the results that were obtained were in accordance with what was expected. Throughout the duration of this project, I was able to effectively apply the knowledge of the PIC24 ADC and UART modules to interface the TMP36 sensor. A way this project could be improved is to try to get the exact temperature value output on the VT instead of being off by 0.1-0.2 as it is currently.

VI. APPENDIX



```
make  -f nbproject/Makefile-default.mk dist/default/debug/CourseProjMS.X.debug.elf
make[2]: Entering directory 'C:/Users/Sohai/MPLABXProjects/Course Project/CourseProjMS.X'
make[2]: 'dist/default/debug/CourseProjMS.X.debug.elf' is up to date.
make[2]: Leaving directory 'C:/Users/Sohai/MPLABXProjects/Course Project/CourseProjMS.X'
make[1]: Leaving directory 'C:/Users/Sohai/MPLABXProjects/Course Project/CourseProjMS.X'

BUILD SUCCESSFUL (total time: 71ms)
Loading symbols from C:/Users/Sohai/MPLABXProjects/Course Project/CourseProjMS.X/dist/default/debug/CourseProjMS.X.debug.elf...
Loading code from C:/Users/Sohai/MPLABXProjects/Course Project/CourseProjMS.X/dist/default/debug/CourseProjMS.X.debug.elf...
Program loaded with pack,PIC24F-GA-GB_DFP,1.4.141,Microchip
Loading completed
```

Figure 10: Succesful Build Proof

REFERENCES

[1] S. Hussain (2023), “BME538 Lec 8 UART” (Class Notes), Toronto Metropolitan University. Available: https://courses.torontomu.ca/d2l/le/content/792243/viewContent/5394732/View

[2] S. Hussain (2023), “BME538 Lec 9 ADC” (Class Notes), Toronto Metropolitan University. Available: https://courses.torontomu.ca/d2l/le/content/792243/viewContent/5403237/View