Microprocessor Systems

Assignment #2

TinyML On AVR

**Group Members:**

* Moonis Amir Zaman (365753)
* Muhammad Ahmad Saleem (383118)

**Contents:**

* Introduction
* Simulation
* Features Implemented
* Practical Circuit
* AVR Code
* Python Code
* Conclusion

**Introduction**

In this report, we present our solution for the tinyML on AVR assignment, which aims to implement a temperature sensing system using an AVR microcontroller and a machine learning model. The system consists of several modules, such as sensor data acquisition, pre-processing, linear regression, min/max computation, serial port communication, and graphical user interface.

**Simulation**

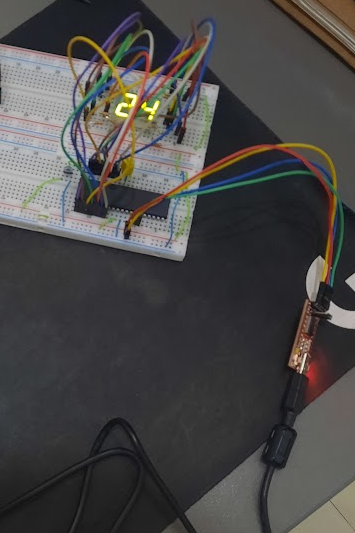
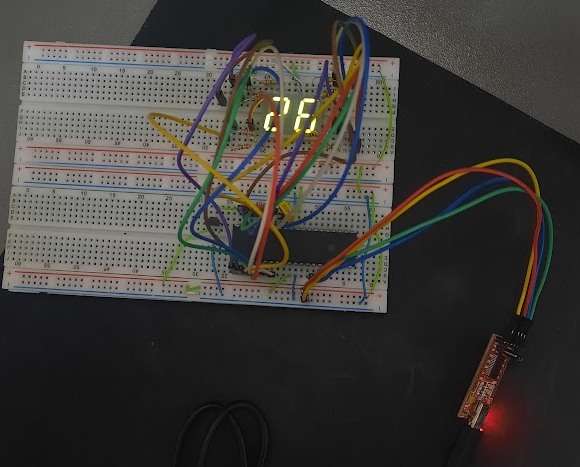
**A screenshot of a computer

Description automatically generated with medium confidence**

**Features Implemented**

* Acquired temperature data from LM35 sensor using AVR ADC
* Applied a 4-tap moving average filter to reduce noise in the sensor data
* Calibrated the sensor data using a linear regression model trained offline with scikit-learn
* Computed and displayed the instantaneous, minimum, and maximum temperature values on a seven segment display
* Communicated with a PC using serial port protocol and Python
* Displayed a time series graph of the temperature values on a PC using matplotlib
* Sent commands from PC to AVR to select which temperature value to display on the seven segment display

**Practical Circuit**

*Room temperature is being displayed*

**Note:** Video demonstration file is provided along with this report.

**AVR Code**

#include <avr/io.h>

#include <stdio.h>

#include <avr/interrupt.h>

#define *F\_CPU* 1000000

unsigned int result\_low = 0, result\_high = 0, result = 0;

volatile unsigned long temp = 0, temp\_avg = 0, Vread = 0, Vinp[4] = {0};

volatile unsigned long min\_temp = 9999, max\_temp = 0; //global variables to store min and max temp

unsigned char cycles = 0;

volatile char command = 'n'; //global variable to store command from pc

volatile unsigned char seven\_seg[] = {0x01, 0x4F, 0x12, 0x06, 0x4C, 0x24, 0x20, 0x0F, 0x00, 0x0C};

void send\_temp(unsigned long t)

{

char buffer[10];

*sprintf*(buffer, "%lu\r\n", t); //convert temp to string with newline

for(int i = 0; buffer[i] != '\0'; i++) //loop until end of string

{

while(!(UCSRA & (1 << UDRE))); //wait until UDR is empty

UDR = buffer[i]; //send one byte

}

}

int main(void)

{

//For 7SEG displays

DDRB = 0xFF;

DDRA = 0xFE;

//initializing UART

UBRRL=0xC ; // set Baud Rate to 4800

UCSRB |= (1<<RXEN) | (1<<RXCIE) | (1 << TXEN); //enable receiver and receiver interrupt

UCSRC |= (1<<URSEL)|(1<<UCSZ0)|(1<<UCSZ1) ; // set data size (8 bits)

//Configuring ADC

//2.56V reference, right adjust, and ADC0 as input

ADMUX = 0b11000000;

//8 prescalar, no interrupt, no auto trigger

ADCSRA = 0b10000011;

TCNT1 = 0; // Initializing timer 1

OCR1A = 15625;

TCCR1A = 0x00; // WGM13:WGM10 = 0100 for CTC mode

TCCR1B = 0x0B; //CS12 : CS10 = 011 for clk/64 prescalar

sei(); // enable global interrupt

TIMSK = 0x10 ; // enable compare match of timer 1

while (1)

{

ADCSRA |= (1 << ADSC); //start conversion

while (ADCSRA & (1 << ADSC)); //polling

//reading digital output

result\_low = ADCL;

//we only want value from last two bits of ADCH (they are 8th and 9th bits)

result\_high = ADCH & 0b00000011;

//complete digital value

result = result\_low + (256 \* result\_high);

Vread = ((unsigned long) result \* 2560ul) / 1023ul;

if (cycles < 4)

{

Vinp[cycles] = Vread;

temp = Vread;

cycles++;

}

if (cycles >= 4)

{

for(int j = 0; j < 3; j++)

{

Vinp[j] = Vinp[j + 1];

}

Vinp[3] = Vread;

temp\_avg = (Vinp[3] + Vinp[2] + Vinp[1] + Vinp[0]) / 4;

//Linear regression equation, to calibrate the reading

temp = (1.0712 \* temp\_avg) - 0.5320;

}

if(temp < min\_temp) //update min temp

{

min\_temp = temp;

}

if(temp > max\_temp) //update max temp

{

max\_temp = temp;

}

}

return 0;

}

ISR(USART\_RXC\_vect)

{

command = UDR; //read data from UDR

}

ISR (TIMER1\_COMPA\_vect)

{

if(command == 'n') //nominal value

{

PORTB = seven\_seg[(temp / 100) % 10];

PORTA = (seven\_seg[(temp / 10) % 10]) << 1;

}

else if(command == 'm') //minimum value

{

PORTB = seven\_seg[(min\_temp / 100) % 10];

PORTA = (seven\_seg[(min\_temp / 10) % 10]) << 1;

}

else if(command == 'M') //maximum value

{

PORTB = seven\_seg[(max\_temp / 100) % 10];

PORTA = (seven\_seg[(max\_temp / 10) % 10]) << 1;

}

send\_temp(temp / 10);

}

**Python Code**

**Code for graph plotting, and serial communication:**

import serial

import matplotlib.pyplot as plt

import matplotlib.animation as animation

ser = serial.Serial('COM5', 4800, timeout=1) #open serial port

ser.bytesize = serial.EIGHTBITS #set data size

ser.parity = serial.PARITY\_NONE #set parity

ser.stopbits = serial.STOPBITS\_ONE #set stop bits

ser.encoding = 'ascii' #set encoding

ser.newline = '\n' #set line ending

temp\_list = [] #list to store temperature values

fig = plt.figure() #create a figure object

ax = fig.add\_subplot(1, 1, 1) #create an axes object

def animate(i): #define a function to update the plot

data = ser.readline() #read one line from serial port

if data: #if data is not empty

temp = int(data) #convert data to integer

temp\_list.append(temp) #append temp to list

ax.clear() #clear the axes

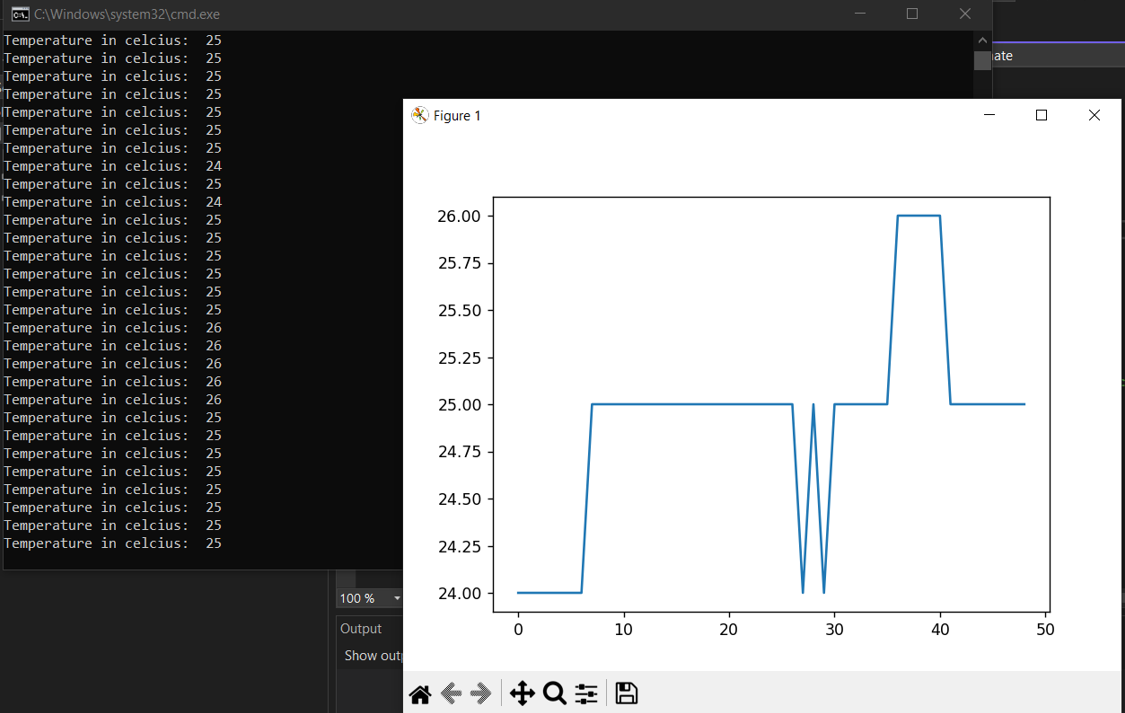
print("Temperature in celcius: ", temp) #print temp to console

ax.plot(temp\_list) #plot list

ani = animation.FuncAnimation(fig, animate, interval=1000) #create an animation object that calls animate function every 1000 ms

plt.show() #show plot

**Output:**

****

**Code for linear regression:**

# Importing scikit-learn library and LinearRegression class

from sklearn.linear\_model import LinearRegression

import numpy as np

# Create two arrays that hold 20 values each for x and y

x = [24, 26, 23, 35, 50, 25, 28] # sensor data voltage in volts

y = [26.2, 27.4, 23, 36.9, 53.1, 26.9, 28.9] # temperature reading in degrees Celsius

# Reshape x array to a 2D array with one column

x = np.array(x).reshape(-1,1)

# Creating an instance of the LinearReg class and fitting it to the x and y arrays

reg = LinearRegression().fit(x,y)

# Print the coefficients and intercept of the fitted model

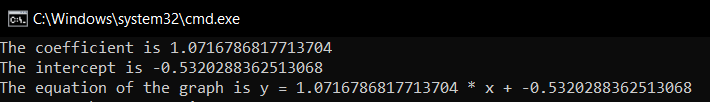
print("The coefficient is", reg.coef\_[0])

print("The intercept is", reg.intercept\_)

# Print the equation of the graph using the coefficients and intercept

print("The equation of the graph is y =", reg.coef\_[0], "\* x +", reg.intercept\_)

**Output:**



**Conclusion**

In this report, I have presented the design and implementation of a tinyML system for temperature sensing using AVR microcontroller. I have described the steps involved in data acquisition, pre-processing, linear regression, min/max computation, serial port communication, and graphical user interface. I have also demonstrated the use of machine learning model to calibrate the sensor data and display the temperature value on a seven segment display and a PC terminal. The results show that the system is effective, safe, and stable for temperature sensing applications.