**Microprocessor Systems**

**Assignment-2**

**Report**

**Date:** 28-04-2023

**Batch & Section:** BEE-13A

**Group Members:**

Mehboob Khaliq (372682)

M. Saad Munawar (393188)

**Problem Statement:**

**A temperature sensor has been deployed in the environment, which is noisy and also needs to be calibrated. As an embedded system engineer, you will use your knowledge to acquire temperature sensor data, perform noise filtering, apply your machine learning model to calibrate the sensor (training will be done offline against some ground truth) and display the temperature value (Instantaneous, Max, Min) locally as well as send it remotely to a computer terminal for monitoring.**

**Components:**

ATmega16 AVR microcontroller, LM 35 Temperature Sensor, 7-Segment Display, UART, PC. Breadboard, Wires

**Working:**

**Step 1:**

* For step 1, we needed to display the temperature sensed by LM35 on the 7-Segments after a one second delay.
* For that we first configured PORT C (make sure to write JTD as 1 in the MCUCSR register in order to use PORT C pins as normal) and B as outputs where the two 7-Segments will be connected respectively. PORT A will be configured as input. (Not necessary as by configuring ADMUX it will happen automatically)
* Then we configured our timer registers. TCNT1 is set to ‘0’ to start counting at zero. TCCR1A is set as 0x00 and TCCR1B is set as 0x0B. This sets timer1 in CTC mode and pre-scales the clock, so the new clock is clk/64. The OCR1A is set as 15625, as after that many ticks, one second has passed. After enabling global interrupts, we enable the OCR1A compare interrupt (OCIE1A) in the TIMSK register. Timer 1 is used as it is a 16-bit timer, so it can count ticks till 216. The other 8-bit timers are unable to do this. After one second has passed the Timer1 compare A interrupt is set and the interrupt function is executed. This creates a 1 second delay.
* Next we configure the ADC registers. ADMUX is set to 0xE0, this sets the reference voltage as the internal 2.56 V, selects pin 0 of PORTA as the input, and makes the result left adjusted. Next the ADCSRA is set as 0x87, this enables the ADC with pre-scaled clock of clk/128. Also enable ADSC in ADCSRA to start conversion and check whether conversion has ended with while((ADCSRA & (1<<ADIF))==0).
* The LM35 is connected to pin 0 of PORTA. By making the result left adjusted and reference voltage 2.56, we get the exact temperature value from ADCH. This is because for 1 degree change in temperature there is a 10mV change. And for 2.56 reference voltage, the step size is 2.5 mV as the AVR is 10-bit. So for 4 steps of AVR there is a 1 degree change. This means that for every change the last 2 bits will remain the same, and as these 2 bits are stored in ADCL, so it can be ignored and we can get the direct value from ADCH.
* So by sending the ADCH value to PORTs B and C in the interrupt function, we manage to display the temperature value on the two 7 segments with a 1 second delay.

**Step 2:**

* For Step 2 we need to pre-process the data by a 4-tap moving average filter before sending it forward to the 7-segments.
* For that we make a 4 element array named temp with initial zero values. After each second ADCH value is sent to element ‘0’ of the array and the value of the 0th element replaces the value of the 1st element of the array and so on, with the value of the 3rd element being discarded.
* By the taking the sum of all the elements of the array and dividing it by 4 we pre-process the data. We have named this value as tep.
* It should be kept in mind that the first 4 values are garbage values.
* This processing is done in the interrupt function, and the tep value is sent to the 7 Segments in the while loop.
* We pre-process the data as it makes the graph smooth making it easier to analyze.

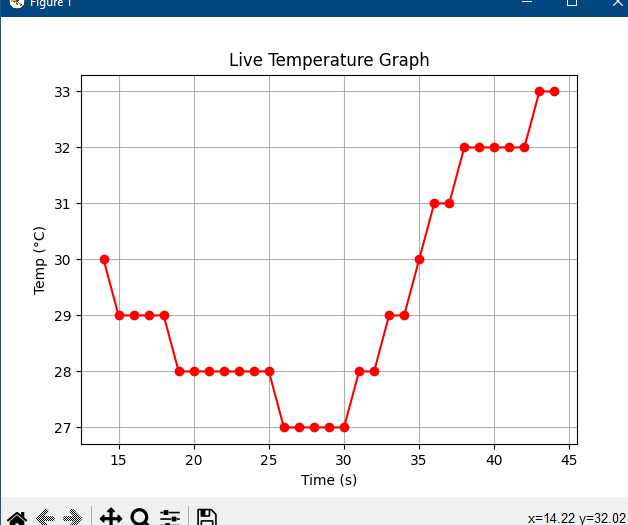
**Step 3:**

* For Step 3, we add min and max temperature as well.
* For that we make unsigned char max and min, and set them to 0 and 255 respectively.
* In the interrupt function the value is updated or left as is depending on whether the value is greater than the max value and lesser than the min value.
* Through manipulations in the code we can choose either min, max or tep values to be displayed on the 7 Segments.

**Step 4:**

* For this step, we need to send temperature values to the PC and display it on TeraTerm.
* We first set the baud rate to 4800 by setting the UBRRL as 0xC. Next we enable the transmitter by enabling TXEN in the UCSRB register. Then we set bit size to 8, stop bit to 1, and update UCSRC by enabling URSEL, UCSZ1 and UCSZ0 in the UCSRC register.
* Before sending, check whether register is empty with while(!(UCSRA & (1<<UDRE))). After checking send the tep value to the UDR. Make sure to add 48 or ‘0’ to the tep value before sending to TeraTerm as value displayed is according to ASCII table, so we need to send corresponding decimal value.
* In this way we sent the temperature value to be displayed on PC via UART.

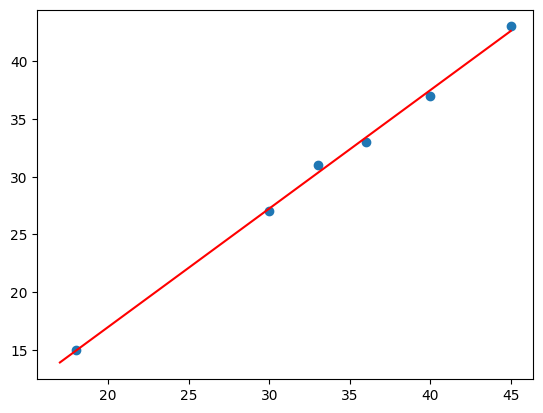
**Step 5:**

* For this step we need to configure the serial port on PC using python and display the data coming from microcontroller as a graph.
* Python has a library called pySerial which allows us to configure the serial port and use it to transmit and receive data through UART.
* A python script was made with an object of class serial. It was initialised with a baudrate of 4800 and the port was set to ‘COM3’ where the atmega16 was connected with the UART to USB cable.
* Function serial.readline() is used to read the temperature sent by the microcontroller, and outputs a binary encoded value. The function reads till and combines all it encounters a newline character, which we send after the two digits of temperature. .decode() and .strip() are added that convert it to simple a string and int() is used to convert it to an integer value.
* This integer value of temperature is passed to matplotlib graph function. For every value of temperature, a variable for time is incremented. Since the values are sent every second, time is used as the x axis, with the unit of seconds. Labels for the axes along with the graph title are added. This is done inside an infinite while loop, and the matplot graph keeps updating.

**Step 6:**

* For this step we need to send commands from the PC to the AVR to display the maximum, minimum or nominal values on the 7-segment display.
* To send messages to the avr, pySerial has a function called serial.write(). This has to be combined with .encode so that the character is converted to binary. We sent the characters ‘h’, ‘l’ and ‘n’ for max, min, and nominal respectively.
* Instead of writing the characters in a terminal, we used pynput, a python library which has a keyboard monitor. We imported listener from pynput, and defined the function for the key presses of h, l and n, so that they serial.write() the character when the key is pressed.
* A problem we faced was that the program cannot read and write to the serial port at the same time, because the main while loop had to continuously check for data from the serial port and plot the graph. To solve this, we used multithreading in python, which allows multiple tasks to run at the same time. The graphing loop and the keyboard listener were converted to functions, which were run as separate threads. This allowed flawless simultaneous reading and writing to the serial port, with smooth graphing that isn’t interrupted by sending commands.

**Step 7:**

* For this step, we had to train a machine learning model to calibrate the lm-35 temperature sensor against readings of actual thermometer.
* 6 values of temperature were taken and the values displayed on 7 seg were noted.
* ****Scikit-learn library’s linear regression was used to develop the ml model, with the temperature values as the y variable and the 7 seg values as the x variable.
* For y = mx+c, it gave m = 1.02 and c = -3.3.
* This equation was implemented in the avr’s code.