TECHNICAL UNIVERSITY OF MOLDOVA

FACULTY OF COMPUTERS INFORMATICS AND MICROELECTRONICS

DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATION

Laboratory work 3.2

Subject: Sensors

Done by: **Konjevic Alexandra,**
st. gr. FAF-213

Verified by: **Moraru Dumitru,**
university lecturer

CHIŞINĂU, 2024

THE TASKS OF THE LABORATORY WORK

Create an MCU-based application that conditions the signal obtained from the sensor (see Lab 3.1) and displays the physical parameter on a terminal (LCD and/or Serial). Each student will select an analog or digital sensor from the attached PDF.

1. Acquire the signal from the sensor;
2. Condition the signal involving digital filters and other methods;
3. Display the data on an LCD display and/or Serial.

Recommendations:

1. For validation, it is recommended to use a simulator, e.g., Proteus.
2. The functionalities for each peripheral device (LED, button, LCD, sensor) should be implemented in separate files for the purpose of reuse in future projects.
3. Utilize magic number and CamelCase coding conventions.

PROGRESS OF THE WORK

1.1 Description

The program that I implemented for this laboratory work is an Arduino-based application for temperature sensing and display. It utilizes a sensor connected to the analog pin A0 to measure temperature. The program implements three methods for temperature calculation and smoothing: ADC conversion, salt-and-pepper filtering, and moving average filtering.

First of all, there is the `adc` function converts the analog readings from the sensor into temperature values using the Steinhart-Hart equation, which considers the sensor's nominal resistance, beta coefficient, and reference resistor value.

The following function - `salt_pepper` - calculates the average of the last 10 analog readings and then converts this average into temperature using the adc function. This serves as a simple filtering method to reduce noise in the temperature readings.

Next filtering function is the `smooth` function, which implements a moving average filter to smooth out the temperature readings. It maintains a buffer of the last numReadings analog readings, updates the total sum of these readings, and calculates the moving average. The moving average is then converted to temperature using the adc function.

This program sets up serial communication at a baud rate of 9600 and initializes the standard output to redirect to the serial port. In the loop function, it continuously reads analog readings from the sensor, calculates temperature using the three methods, and prints the results via serial communication. Additionally, it incorporates a delay of 2000 milliseconds between consecutive readings to avoid rapid serial output.

The `io.hpp` header file provides functions for redirecting standard output to the serial port, enabling the use of printf-like syntax for serial communication. This allows for convenient formatting and printing of temperature readings.

1.2 Block Diagram

The following diagram shows the setup function of the program. Here we can see how each of the functions is executed, and the results are printed on the serial monitor.

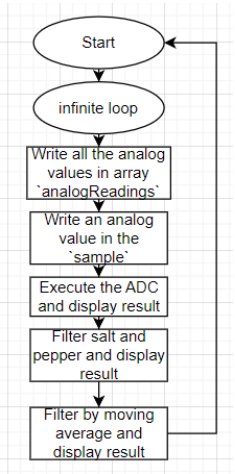

Figure 1. Block diagram of the program setup

Next, we have the diagram of the first function – the `adc` (Analog Digital Conversion) function. Here we can see that the main logic consists in using the Steinhart-Hart equation to convert the analog signal received from the sensor, in a digital value.

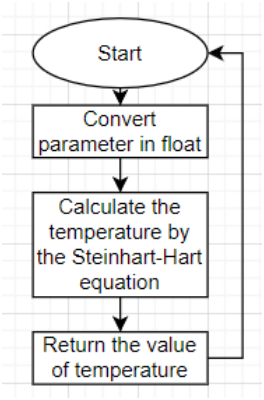

Figure 2. Block diagram of the adc function

In the figure below, we can see the flow chart for the next function – `salt_pepper` function.

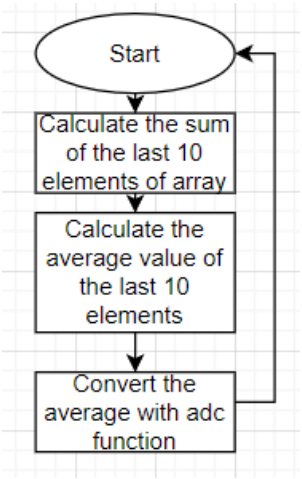

Figure 3. Block diagram of the salt_pepper function

And last, but not least, we can see the flow chart of the function that performs the filtering using the moving average method.

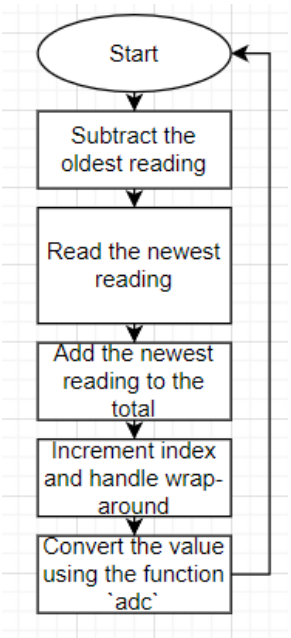

Figure 4. Block diagram of moving average function

1.3 Electrical Schematic

To simulate the electrical schematic for this project, I began by placing an Arduino Uno board at the center. I then connected the NTC (negative temperature coefficient) temperature sensor, which is an analog sensor. The temperature sensor module includes a 10K NTC thermistor in series with a 10K resistor. This setup produces a voltage that depends on the temperature. You can read this voltage by connecting the OUT pin of the thermistor to an analog input pin and then using the analogRead() function.

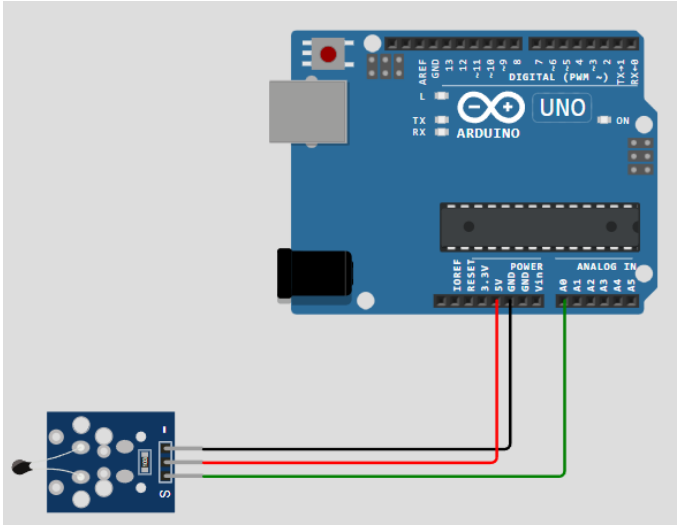

Figure 5. Electric circuit

1.4 Running Simulation

In the image below we can see the simulation. In the serial monitor are displayed the three calculations of the temperature: first one, by applying only the analog to digital conversion, second one, after applying the salt pepper filter and the last one, after applying the moving average filter.

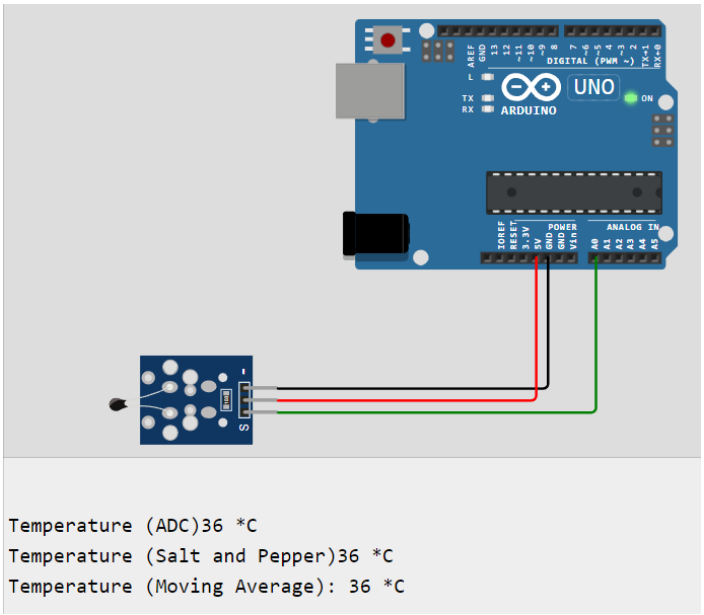

Figure 6. Simulation with all the results on monitor

CONCLUSION

In this lab work, we explored the process of measuring temperature using an NTC thermistor sensor connected to an Arduino microcontroller. We began by setting up the hardware configuration, including connecting the sensor to the appropriate pin on the Arduino and configuring necessary parameters such as the nominal resistance, beta coefficient, and reference resistor value.

We then implemented a program to read analog signals from the sensor, convert them into temperature values using the Steinhart-Hart equation, and display the results over the serial monitor. Additionally, we introduced a moving average filter to smooth out the temperature readings and reduce noise in the measurements.

Throughout the lab work, we encountered various programming concepts and techniques, including analog signal processing, mathematical modeling of sensor behavior, and implementing data filtering algorithms. By combining theoretical knowledge with practical experimentation, we gained insights into the complexities of sensor interfacing and data processing in embedded systems.

This lab work provided valuable hands-on experience in working with sensors and microcontrollers, as well as enhancing our understanding of signal conditioning techniques for improving measurement accuracy and reliability. Overall, it served as an effective learning opportunity to develop essential skills in sensor integration and data processing for real-world applications.

APPENDIX A: Source code

```
#include "io.hpp"

#define ntc_pin A0               // Pin,to which the voltage divider is
connected
#define nominal_resistance 10000 // Nominal resistance at 25ºC
#define nominal_temperature 25   // temperature for nominal resistance (almost
always 25º C)
#define rate 50                  // Number of samples
#define beta 3950                // The beta coefficient
#define Rref 10000               // Value of  resistor used for the voltage
divider
#define numReadings 10
#define BAUD_RATE 9600

int analogReadings[rate];
int readings[numReadings];
int readIndex = 0;
long total = 0;

float adc(int sample)
{
    float sampleFloat = 0;
    sampleFloat = sample;
    sampleFloat = 1023 / sampleFloat - 1;
    sampleFloat = Rref / sampleFloat;

    float temperature;
    temperature = sampleFloat / nominal_resistance;      // (R/Ro)
    temperature = log(temperature);                      // ln(R/Ro)
    temperature /= beta;                                 // 1/B * ln(R/Ro)
    temperature += 1.0 / (nominal_temperature + 273.15); // + (1/To)
    temperature = 1.0 / temperature;                     // Invert
    temperature -= 273.15;                               // convert absolute
temp to C

    return temperature;
}

float salt_pepper()
{
    float average = 0;
    int sum = 0;
    for (int i = rate - 10; i < rate; i++)
    {
```

```
        sum += analogReadings[i];
    }
    average = sum / 10;
    float converted_average = adc(average);
    return converted_average;
}

long smooth()
{
    total = total - readings[readIndex];       // Subtract the oldest reading
    readings[readIndex] = analogRead(ntc_pin); // Read the newest reading
    total = total + readings[readIndex];       // Add the newest reading to the
total
    readIndex = (readIndex + 1) % numReadings; // Increment index and handle
wrap-around
    float result = total / numReadings;
    float convertedResult = adc(result);
    return convertedResult;
}

void setup(void)
{
    redirect_stdout();
    Serial.begin(BAUD_RATE);
    printf("Serial communication started\n");
}

void loop(void)
{
    for (int i = 0; i < rate; i++)
    {
        analogReadings[i] = analogRead(ntc_pin);
        delay(5);
    }

    int sample = analogRead(ntc_pin);

    float result = adc(sample);
    printf("Temperature (ADC)");
    printf("%d", (int)result);
    printf(" *C\n");

    float salt_pepper_result = salt_pepper();
    printf("Temperature (Salt and Pepper)");
    printf("%d", (int)salt_pepper_result);
```

```
    printf(" *C\n");

    float moving_average_result = smooth();
    printf("Temperature (Moving Average): ");
    printf("%d", (int)moving_average_result);
    printf(" *C\n");

    printf("\n\n");

    delay(2000);
}
```

io.hpp:

```
#pragma once
#include <stdio.h>

// https://forum.arduino.cc/t/printf-on-arduino/888528/3
FILE f_out;

int sput(char c, __attribute__((unused)) FILE *f)
{
    if (c == '\n')
    {
        return !Serial.write("\r\n");
    }
    return !Serial.write(c);
}

void redirect_stdout()
{
    //  https://www.nongnu.org/avr-libc/user-
manual/group__avr__stdio.html#gaf41f158c022cbb6203ccd87d27301226
    fdev_setup_stream(&f_out, sput, nullptr, _FDEV_SETUP_WRITE);
    stdout = &f_out;
}
```

BIBLIOGRAPHY

[1] Ntc-temperature-sensor, Analog temperature sensor: NTC, [Quoted: 10.03.2024], access link: https://docs.wokwi.com/parts/wokwi-ntc-temperature-sensor

[2] NTC Temperature Sensor With Arduino, [Quoted: 10.03.2024], access link: https://www.instructables.com/NTC-Temperature-Sensor-With-Arduino/

[3] Temperature Sensor (Analog Input), [Quoted: 10.03.2024], access link: https://www.instructables.com/Temperature-Sensor-LED-Bar-Graph/

[4] Program example, Interfacing NTC Sensor with Arduino Uno, [Quoted: 10.03.2024], access link: https://www.theelectronics.co.in/2023/08/interfacing-ntc-sensor-with-arduino-uno.html

[5] Tutorial NTC, NTC Simulation with Arduino in Wokwi | Thermistor Simulation, [Quoted: 10.03.2024], access link:

https://www.youtube.com/watch?v=yFO6dTx0ZgY&ab_channel=SatyamSingh