**TECHNICAL UNIVERSITY OF MOLDOVA**

**FACULTY OF COMPUTERS, INFORMATICS   
AND MICROELECTRONICS**

**DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATICS**

**Laboratory work nr. 3  
User interaction - Sensors - Signal Conditioning**

A elaborat: **Brinza Cristian**

st. gr. FAF-212

A verificat: **Moraru Dumitru**

lect. univ

**Chișinău 2024**

# THE TASK OF THE LABORATORY WORK:

Configure the application to work with the STDIO library through the serial interface for text exchange via LCD+Keypad.

Create an MCU-based application that will condition the signal received from the sensor (see Lab 3.1), and will display the physical parameter at a terminal (LCD and/or Serial).

Each student will select a sensor either analog or digital (not binary) from the attached PDF or: http://www.37sensors.com/

* To acquire the signal from the sensor;
* To condition the signal involving digital filters and other methods;
* To display the data on the LCD and / or Serial display.

For Validation it is recommended to use a simulator, e.g. Proteus. The functionalities for each peripheral device (led, button, lcd, sensor) should be created in separate files, for the purpose of reuse in the following works, the use of magic number and CamelCase coding rules, clocking.

Mark 5 - the simple LCD/Seral display application of the parameter:

* for implementing ADC to Voltage + Voltage to Physical Parameter conversion
* for implementing the Salt and Pepper filter
* for weighted averaging filter implementation
* for immolation ALL the solution, including conditioning for a sensor other than the one from the laboratory (eg DHT or Ultrasonic)
* for demonstrating evidence of physical implementation

NOTE: maximum attendance possible only at the presentation of physical performance!!

* Use CamelCase coding conventions;
* Use **STDIO** to scan the keyboard and display on the LCD.
* Declare port variable like intended;
* The functionalities for each peripheral device (LED, button, LCD, keypad) should be implemented in separate files for the purpose of reuse in future projects.

# PROGRESS OF THE WORK

## 1 Main functions/methods used to execute the task

Explication about this chapterIn this chapter I will explain the functionality of diferent parts of the executed task

**In the Main Sketch File**:

* *setup():* Initializes the Arduino's serial communication, sets up the LCD display, and prepares any other necessary peripherals or variables for use. This function runs once at the beginning and is critical for ensuring that all components are correctly initialized before entering the main program loop.
* loop(): The heart of the Arduino sketch, containing the logic that's executed repeatedly. It involves reading sensor data, applying filtering methods to clean the data, converting the readings to meaningful values (e.g., temperature in Celsius, light intensity in lux), and displaying these values on an LCD. The loop ensures continuous monitoring and updating of environmental conditions.

**In Sensor Reading and Filtering:**

* readSensorWithMedianFilter(int pin): Reads multiple values from a specified analog pin, applies a median filter to these readings to remove outliers, and returns the median value. This method is crucial for obtaining stable readings from sensors that might be affected by short-term fluctuations or noise.
* getTemperature(float adcValue): Converts the filtered ADC value from the NTC temperature sensor to a temperature in Celsius using a specific formula. This method encapsulates the mathematical relationship between the sensor's electrical characteristics and the physical temperature.
* readSensorWithNoiseReduction(int pin): Similar to readSensorWithMedianFilter but aimed at the photoresistor sensor, it reads multiple values, discards the highest and lowest readings to reduce the impact of transient spikes or drops, and averages the rest. This method provides a simplified approach to mitigating noise.
* getLightIntensity(float adcValue): Converts the filtered ADC value from the photoresistor sensor into lux, representing the light intensity. It employs a specific calculation based on the sensor's resistance and characteristics defined by constants like GAMMA and RL10.

**In Display Management:**

* setup(): Initializes the LCD display settings, such as specifying the I2C address and dimensions. This setup is essential for ensuring that the display communicates correctly with the Arduino and is ready to show information.
* clear(): Clears the LCD display to prepare for new information to be shown, ensuring that the display is readable and that old data doesn't clutter the screen.
* setCursor(int col, int row): Positions the cursor at a specific column and row on the LCD, determining where the next piece of text will begin. This function is crucial for organizing the displayed information logically.
* print(String message): Displays a message at the cursor's current position on the LCD. This method is used to show sensor readings and other relevant information to the user.

**Median Filter Implementation for NTC Temperature Sensor:**

The Median Filter is a non-linear digital filtering technique, often used to remove noise from a signal. It works by organizing a set of data points in order, and then selecting the middle value. If the dataset has an even number of points, the average of the two middle numbers is taken. This method is particularly effective in removing 'salt and pepper' noise without significantly blurring the signal.

*readSensorWithMedianFilter* function captures *numReadings (10)* readings from the specified analog pin. It then sorts these readings using a simple bubble sort algorithm. Finally, it selects the median value from the sorted array. This filtered reading is more robust against transient spikes or drops in sensor output, providing a more stable and accurate representation of the sensor's measured parameter over time.

**Noise Reduction for Photoresistor Sensor**

The method used in *readSensorWithNoiseReduction* is a form of outlier removal intended to mitigate the impact of extreme values, which may result from environmental noise or sensor anomalies. By reading multiple values, identifying the maximum and minimum readings, and excluding these from the total before calculating the average, this method effectively reduces the influence of sudden spikes or drops. This approach, while not exactly the *Salt and Pepper* method traditionally used in image processing, serves a similar purpose in mitigating the effect of extreme outliers on the sensor's average reading.

**Calculating Temperature**

The *getTemperature* function applies a formula that converts the a*nalog-to-digital converter (ADC)* value from the *NTC* sensor into a temperature reading in *Celsius*. It uses the B-value or Beta parameter of the thermistor (specified by *BETA*), a known constant that describes the change in the material's resistance with temperature. The formula involves a logarithm and the Steinhart-Hart equation, providing an accurate conversion from resistance to temperature.

**Calculating Light Intensity**

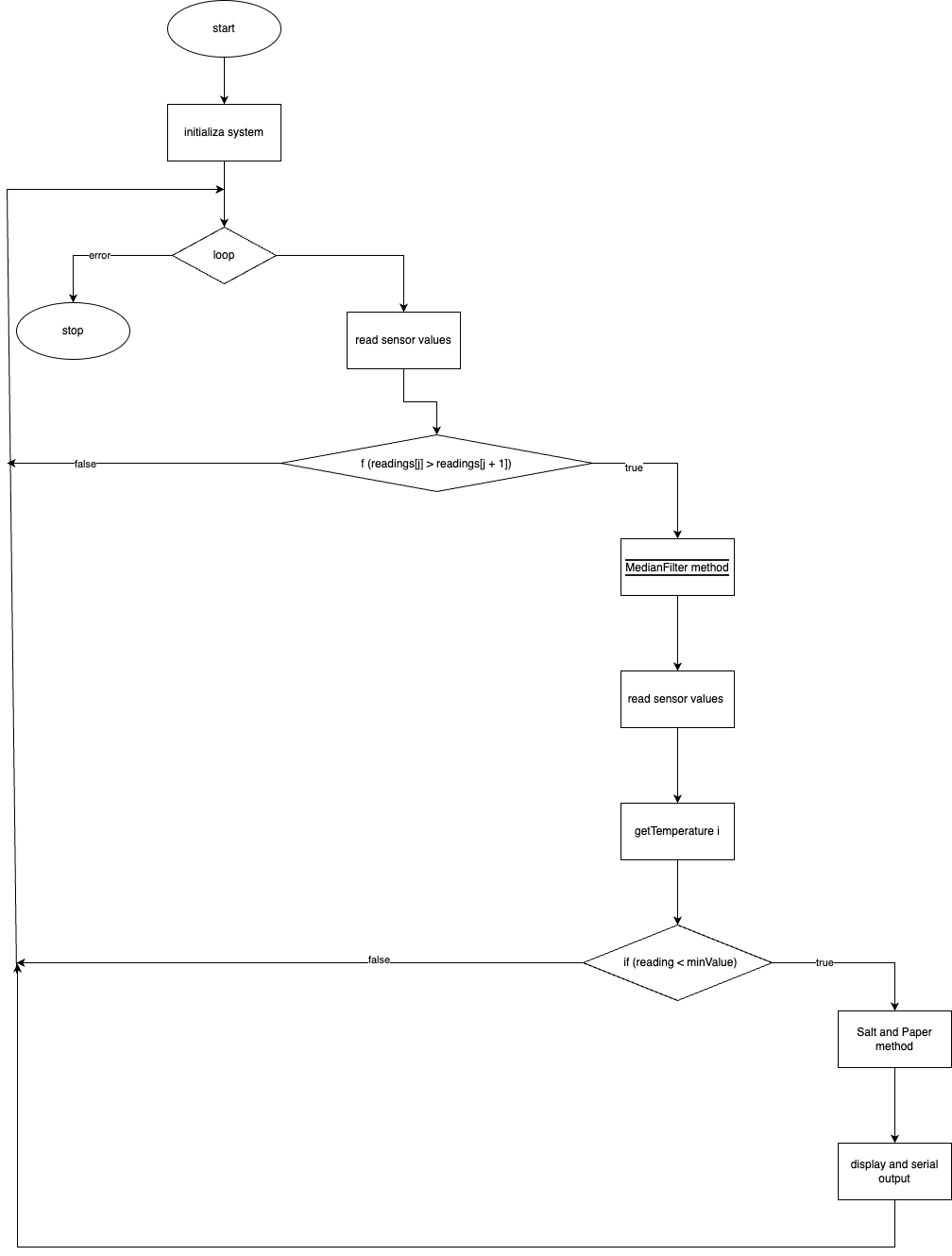
The getLightIntensity function converts the *ADC* value from the photoresistor into *lux*, a unit of illuminance. First, it calculates the voltage across the photoresistor, then computes the resistance of the photoresistor using a fixed series resistor (assumed to be *2000* ohms in your calculation). Using the *GAMMA* constant and the *RL10* value (which you need to adjust based on your specific sensor's characteristics), it then calculates the illuminance in lux. This formula is specific to the characteristics of the photoresistor used and allows for accurate light intensity measurements.

## Block Diagram

The diagram is consisting of the following main blocks:

* **Main Program (sketch.ino):** The central point that initializes and controls other components (Keypad, LCD Display, and LEDs) and manages the program flow.
* **KeypadControl Class**: Represents the logic for interfacing with the keypad, including setup, password verification, input handling, and feedback display.
* **LcdDisplay Class**: Handles the initialization and control of the LCD display, including setup and message display.
* **LEDControl Class**: Manages the LED indicators for indicating lock status, including initialization and control.
* **Arduino Board**: The physical layer where the keypad, LCD display, and LEDs are connected

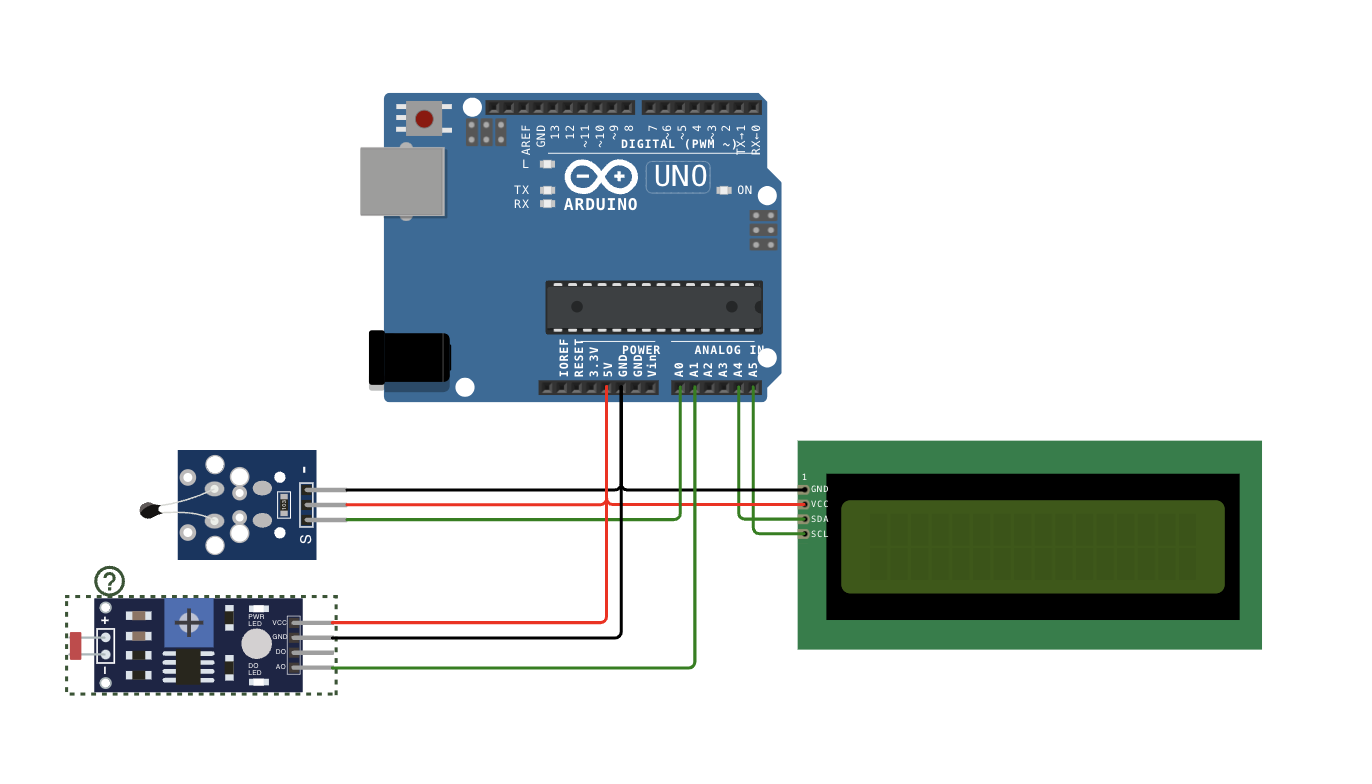
The Figure 1 depicted the UML program flow.



**Figure 1** Program schema.

## 3 Simulated or real assembled electrical schematic diagram:

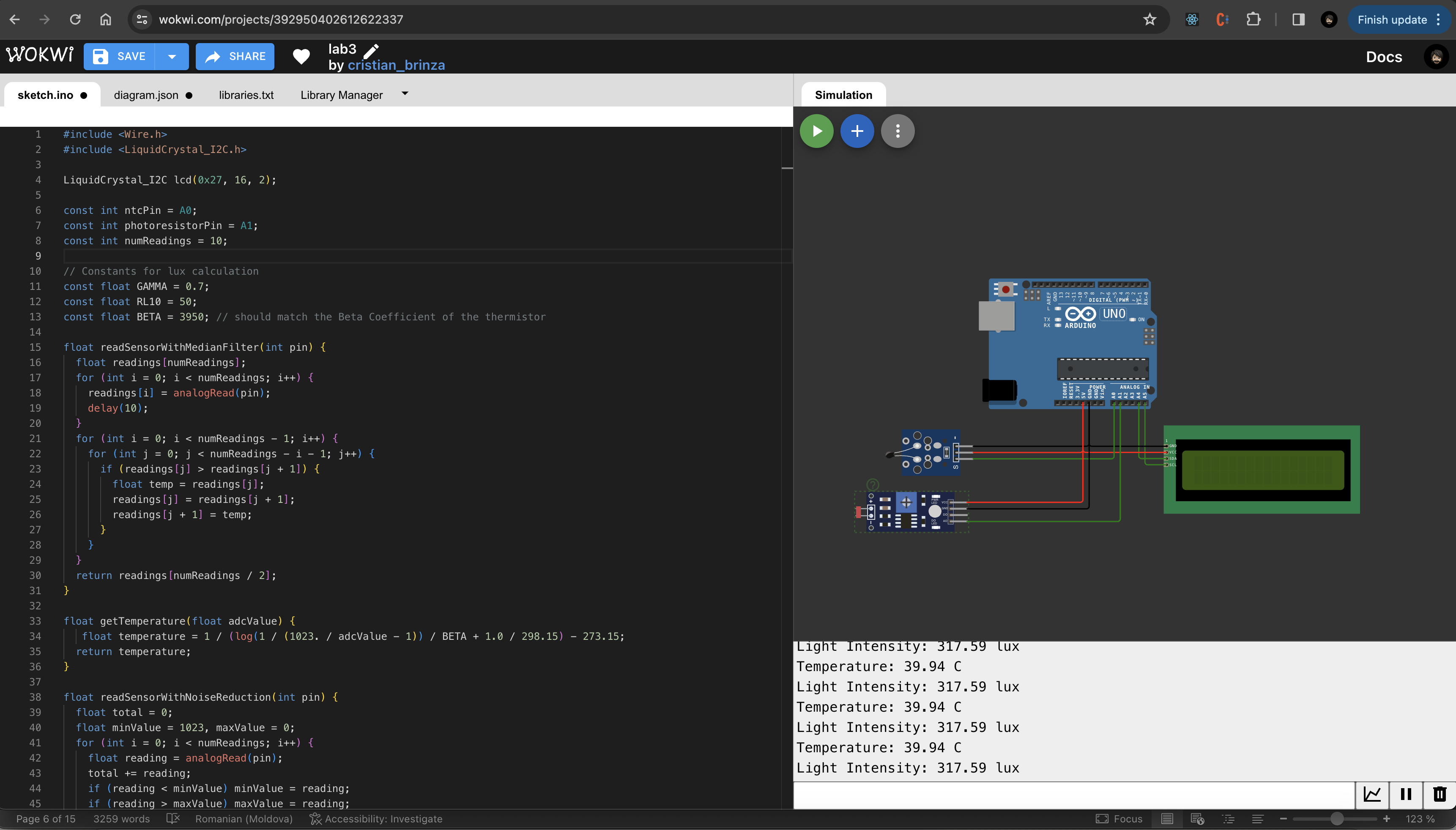
The Figure 2 depicted the phisical board schema.



**Figure 2** Phisical board schema.

## 4 Screenshots of the simulation execution:

The Figure 3 depicted a sceenshot of the WOKWI simulation :



**Figure 3** Screenshot of the simulation.

## CONCLUSION

This project embarked on the journey of developing an environmental monitoring system, utilizing Arduino to measure temperature and light intensity with precision, and display these values in an understandable format on an LCD. Witnessing the transformation of raw sensor data into meaningful environmental parameters was both enlightening and gratifying. We delved into the intricacies of coding practices, ensuring clarity and maintainability through the use of CamelCase naming conventions and modular code organization, which streamlined the debugging process and enhanced the system's scalability.

A significant achievement was the implementation of filtering techniques to mitigate sensor noise, ensuring the reliability of our readings. This not only highlighted the importance of data processing in embedded systems but also served as a practical lesson in enhancing sensor data accuracy through software methods.

The Arduino platform proved to be an invaluable asset, offering extensive support and resources that facilitated our learning and problem-solving processes. This project underscored the critical role of integrating hardware with software, bringing to light the seamless interaction between the Arduino board, sensors, and LCD display.

In conclusion, this endeavor was more than a mere academic exercise; it was a comprehensive learning experience that spanned the realms of coding discipline, data processing, and system integration. The skills acquired and the challenges overcome throughout this project have laid a solid foundation for future projects in embedded systems and environmental monitoring. The successful completion of this project not only signifies our capability to tackle complex technical challenges but also ignites a passion for further exploration and innovation in the field of embedded system design.

## BIBLIOGRAPHY

1. EDUCBA: *Introduction to Embedded Systems.* Cursuri electronice online ©2020 [quote 10.02.2024] Available: <https://www.educba.com/what-is-embedded-systems/>
2. ARDUINO: *Arduino UNO.* The official website of Arduino modules, ©2020 [quote 10.02.2024]. Available: <https://www.arduino.cc/>.
3. TUTORIALSPOINT: *Embedded Systems Tutorial*. Comprehensive guide for beginners and professionals to learn Embedded System basics to advanced concepts, including microcontrollers, processors, and real-time operating systems. ©2023 [quote 10.02.2024] Available: <https://www.tutorialspoint.com/embedded_systems/index.htm>
4. GURU99: *Embedded Systems Tutorial: What is, History & Characteristics*. A detailed introduction to embedded systems, covering microcontrollers, microprocessors, and the architecture of embedded systems, as well as their applications and advantages. ©2023 [quote 10.02.2024] Available: <https://www.guru99.com/embedded-systems-tutorial.html>
5. JAVATPOINT: *Embedded Systems Tutorial.* Offers basic and advanced concepts of Embedded System, designed for beginners and professionals. ©2023 [quote 10.02.2024] Available: <https://www.javatpoint.com/embedded-systems-tutorial>
6. DEEPBLUE*: Embedded Systems Tutorials Introduction* | Embedded Systems Online Course. ©2023 [quote 10.02.2024] Available: <https://deepbluembedded.com/embedded-systems-tutorials/>

**APPENDIX 1**

Code of main.ino:

#include <Wire.h>

#include <LiquidCrystal\_I2C.h>

LiquidCrystal\_I2C lcd(0x27, 16, 2);

const int ntcPin = A0;

const int photoresistorPin = A1;

const int numReadings = 10;

// Constants for lux calculation

const float GAMMA = 0.7;

const float RL10 = 50;

const float BETA = 3950; // should match the Beta Coefficient of the thermistor

float readSensorWithMedianFilter(int pin) {

float readings[numReadings];

for (int i = 0; i < numReadings; i++) {

readings[i] = analogRead(pin);

delay(10);

}

for (int i = 0; i < numReadings - 1; i++) {

for (int j = 0; j < numReadings - i - 1; j++) {

if (readings[j] > readings[j + 1]) {

float temp = readings[j];

readings[j] = readings[j + 1];

readings[j + 1] = temp;

}

}

}

return readings[numReadings / 2];

}

float getTemperature(float adcValue) {

float temperature = 1 / (log(1 / (1023. / adcValue - 1)) / BETA + 1.0 / 298.15) - 273.15;

return temperature;

}

float readSensorWithNoiseReduction(int pin) {

float total = 0;

float minValue = 1023, maxValue = 0;

for (int i = 0; i < numReadings; i++) {

float reading = analogRead(pin);

total += reading;

if (reading < minValue) minValue = reading;

if (reading > maxValue) maxValue = reading;

delay(10);

}

total = total - minValue - maxValue;

return total / (numReadings - 2);

}

float getLightIntensity(float adcValue) {

float voltage = adcValue / 1024. \* 5;

float resistance = 2000 \* voltage / (1 - voltage / 5);

float lux = pow(RL10 \* 1e3 \* pow(10, GAMMA) / resistance, (1 / GAMMA));

return lux;

}

void setup() {

Serial.begin(9600);

lcd.init();

lcd.backlight();

}

void loop() {

float ntcTemperature = getTemperature(readSensorWithMedianFilter(ntcPin));

float lightIntensity = getLightIntensity(readSensorWithNoiseReduction(photoresistorPin));

lcd.clear();

lcd.setCursor(0, 0);

lcd.print("Temp: ");

lcd.print(ntcTemperature, 2);

lcd.print(" C");

lcd.setCursor(0, 1);

lcd.print("Light: ");

lcd.print(lightIntensity, 2);

lcd.print(" lux");

Serial.print("Temperature: ");

Serial.print(ntcTemperature, 2);

Serial.println(" C");

Serial.print("Light Intensity: ");

Serial.print(lightIntensity, 2);

Serial.println(" lux");

delay(2000);

}