Ministry of Education and Research of the Republic of Moldova

Technical University of Moldova

Department of Software and Automation Engineering

REPORT

Laboratory work No. 4.1

Discipline: Embedded Systems

Student: Revenco Victor, FAF - 221

Checked: asist. univ., Martiniuc A.

Chișinău 2025

Analysis of the Situation in the Field

1. Description of the Technologies Used and Application Context

This project implements a modular system for controlling an LED through a relay, using commands through the STDIO interface, and displaying the state of the relay.

Hardware Components

- **Microcontroller**
- **Jumper Wires**
- **USB Power Supply**
- **Relay module**
- **LED**
- **Resistor**
- **Breadboard**

Software Components

- **PlatformIO with Visual Studio Code**: Integrated Development Environment (IDE)
- **C++ for Embedded Systems**: Programming language used for implementation
- **STDIO Functions (printf)**: For system reporting and monitoring
- **Serial Monitor**: For displaying system status

2. System Architecture Explanation and Solution Justification

The application is organized into several modules:

- **Main Module**: Sets up the system, creates FreeRTOS tasks, and initializes modules
- **Relay Module**: Offers methods to toggle the state of the relay
- **Serial I/O Module**: Configures STDIO for serial communication

3. Case Study

The system can have several practical uses, for example for switching on/off a light, a siren, a fan etc. Remotely start/stop industrial machines, irrigation systems and so on

4. Design

1. Architectural Sketch and Component Interconnection

- Serial Monitor for system status output
- Relay for turning on/off the LED

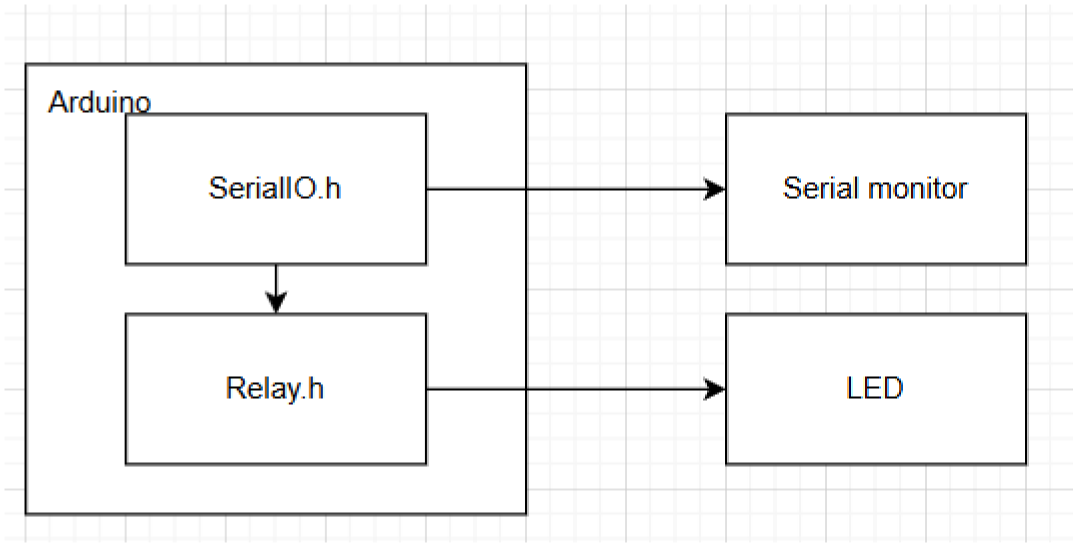


Figure 1 Architectural sketch

Component Description and Their Roles:

- **Microcontroller**: Core processing unit executing tasks
- **Ultrasonic Sensor**: Registers the distance to any object infront of it

The sketch shows how the components interact.

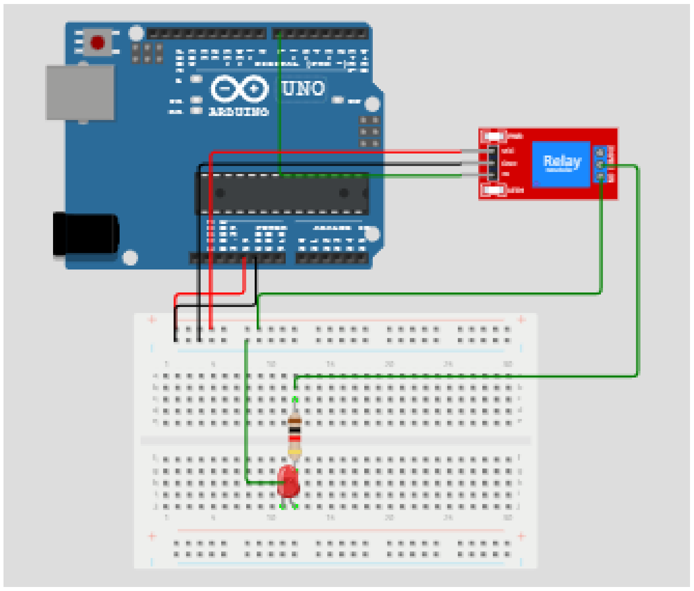


Figure 2 Electrical schematic

This image shows an **Arduino Uno** connected to a relay module through the breadboard. Through the breadboard we use a resistor and a LED.

Here’s a breakdown of the components and their connections:

1. Microcontroller (Arduino Uno)

- The **Arduino Uno** is the main processing unit, providing power and controlling the circuit.
- The 5V pin is connected to the breadboards ‘+’ row
- The GND pin is connected to the breadboards ‘-‘ row

2. Relay module

- The IN pin is connected to pin 7 on the MCU
- The GND pin is connected in the ‘-‘ row on the breadboard
- The VCC pin is connected in the ‘+’ row on the breadboard
- The NO pin is connected in the ‘+’ row on the breadboard
- The COM pin is connected right next to one side of the resistor

2. Schematic diagrams

To understand the system's behavior, a Flowchart and a Finite State Machine (FSM) are used.

Flowchart – Serial Command Processing

The FSM includes the following states:

1. Idle – the system starts in idle state waiting for input
2. On – if the user types “relay on” the relay turns ON, then returns to idle
3. Off – if the user types “relay off” the relay turns OFF, then returns to idle
4. Error – if the input is invalid, it prints error message, then returns to idle

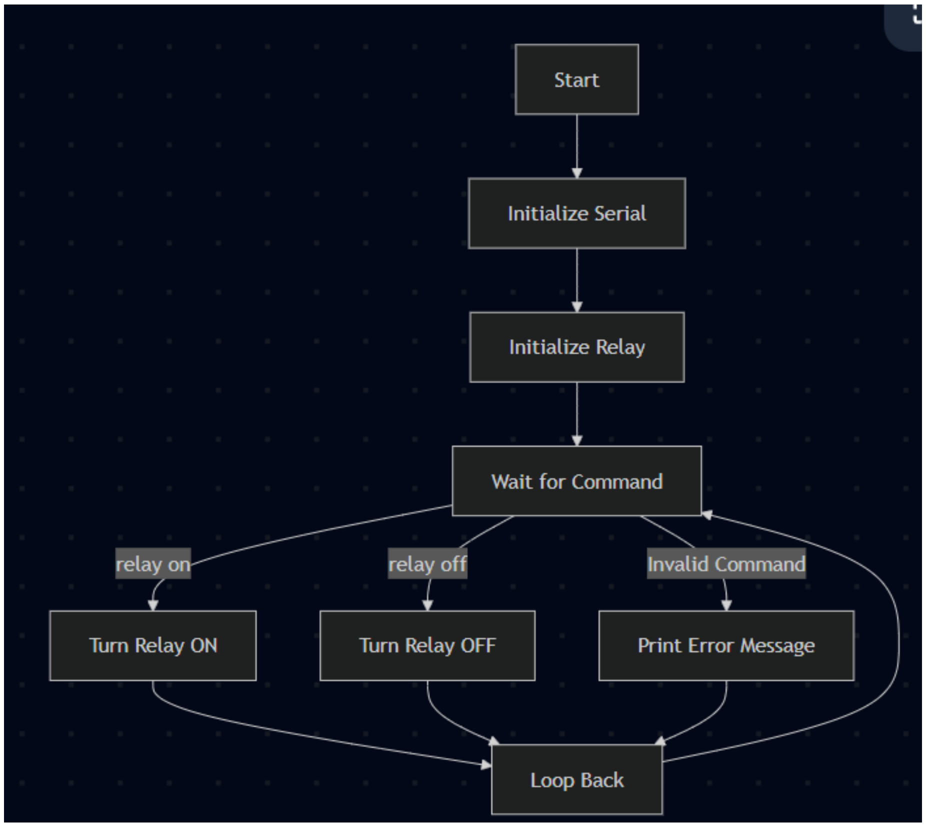


Figure 3 System

The **Main Program Flowchart** represents the overall flow of the relay control system. The process begins with initializing serial communication, followed by setting up the relay on the specified pin. Once these initializations are completed, the system enters a waiting state, listening for commands from the user.

When a command is received, the system checks if the input is `"relay on"`, which leads to the relay being turned on, or `"relay off"`, which turns the relay off. If the command is neither of these, the system prints an error message indicating that the input is invalid. After processing the command, the program returns to the waiting state, ready to accept the next command. This cycle repeats continuously.

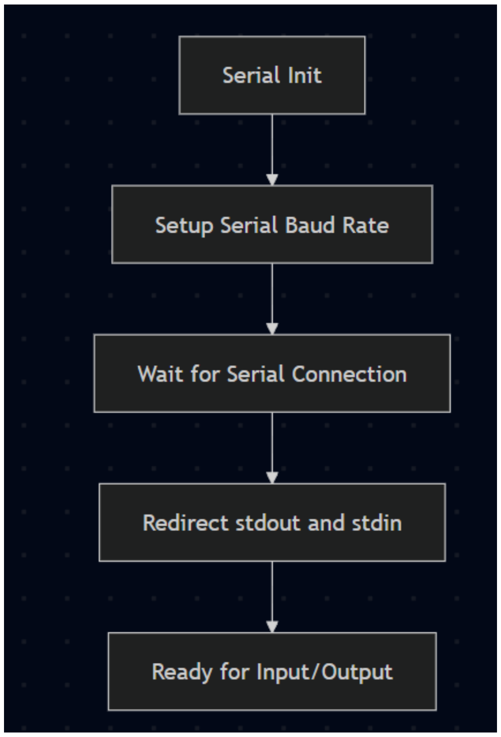


Figure 4 SerialIO

The **SerialIO Flowchart** shows the steps involved in setting up serial communication. The system begins by initializing the serial interface with a specified baud rate. It then waits for the serial connection to be established and redirects the standard output (`stdout`) and input (`stdin`) to the serial interface, allowing the program to interact with the user via the serial console.

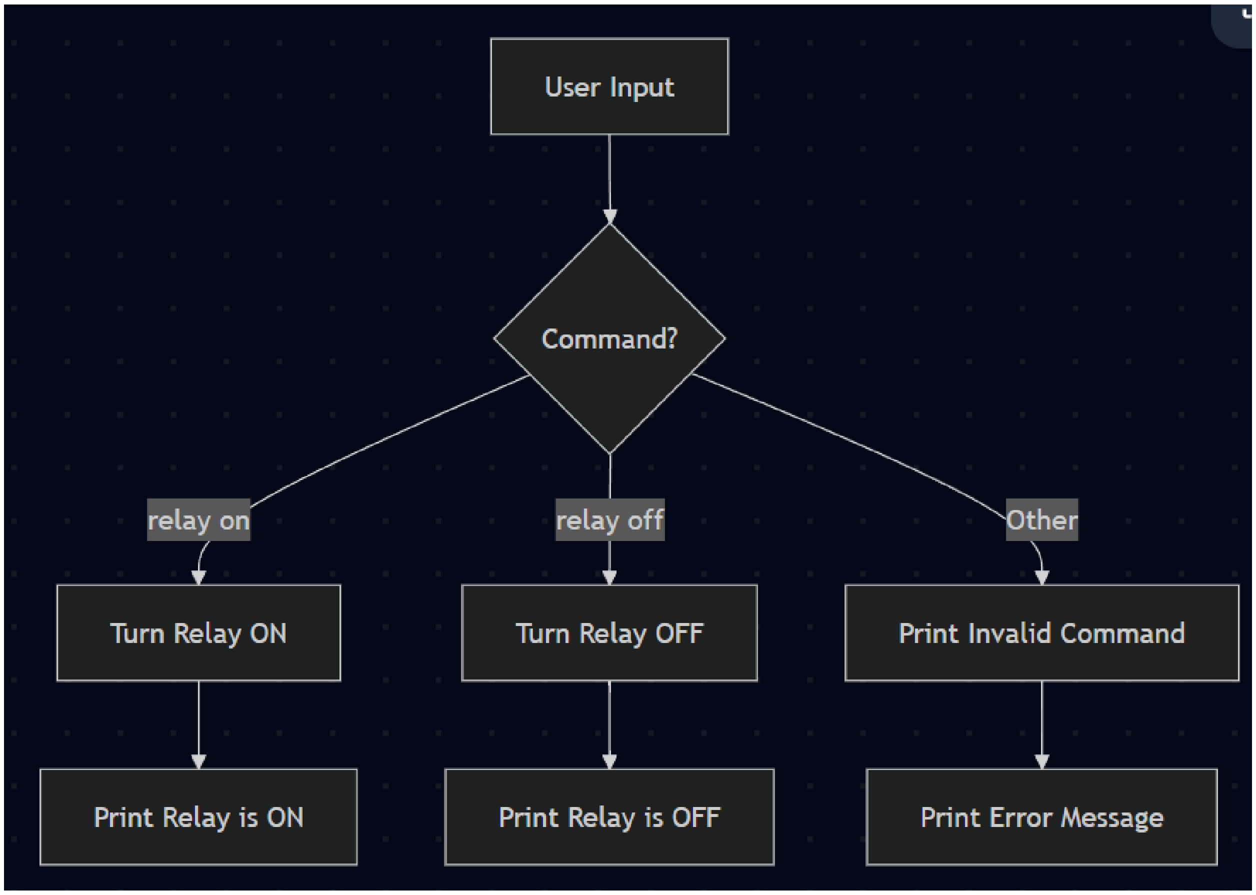


Figure 5 User command

The **User Command Flowchart** details the decision-making process when a command is received from the user. After receiving input, the system checks whether the command is `"relay on"` or `"relay off"`. If the input matches one of these, the relay is controlled accordingly, and a confirmation message is printed. If the command is invalid, an error message is displayed, prompting the user to enter a valid command.

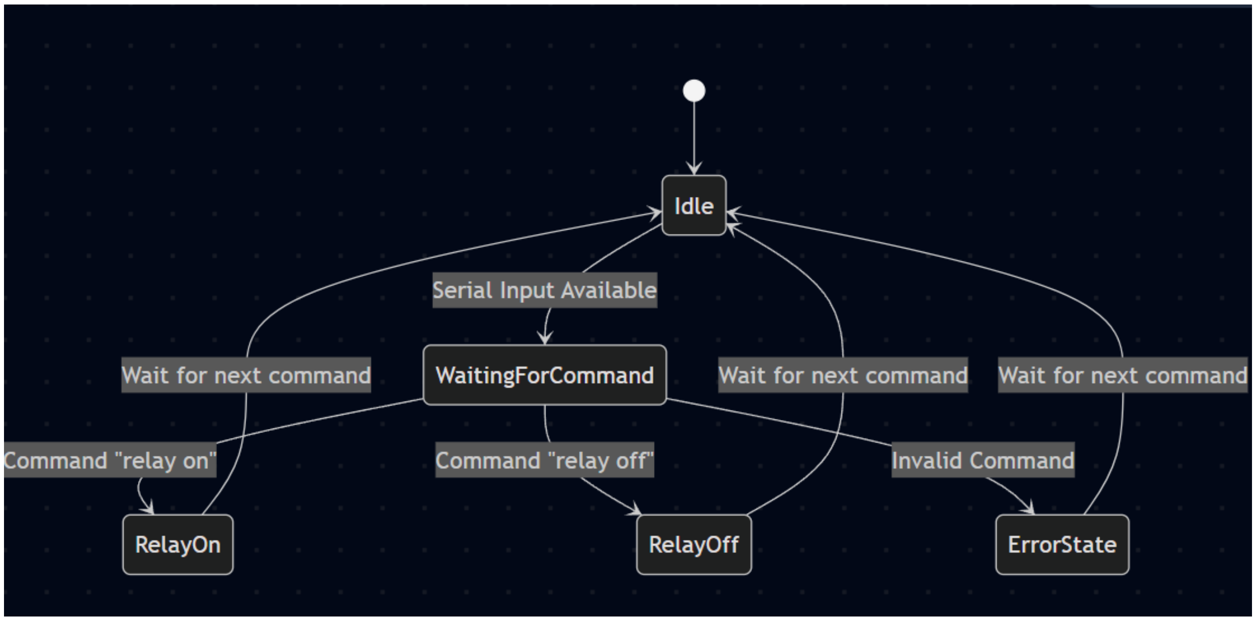


Figure 6 FSM

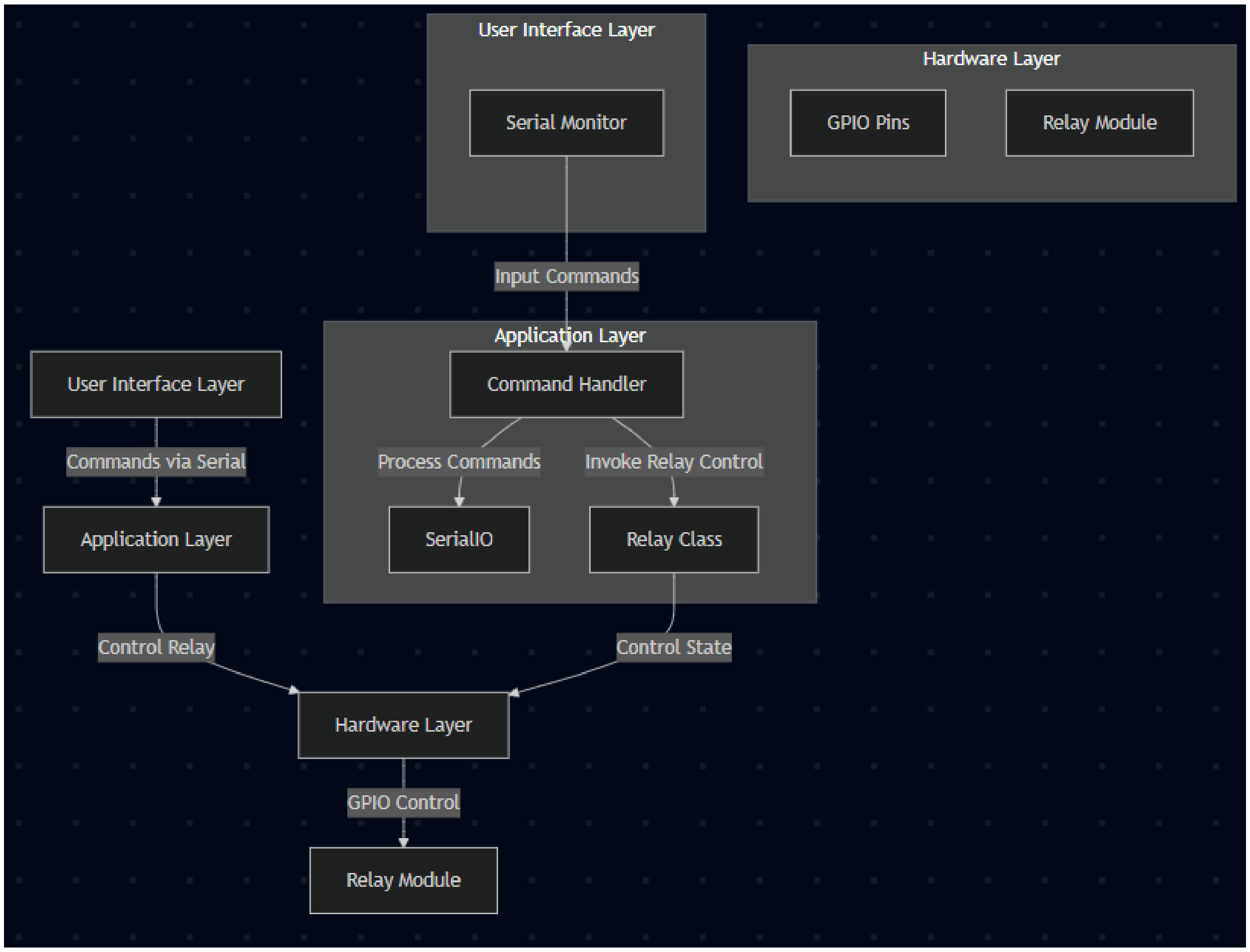
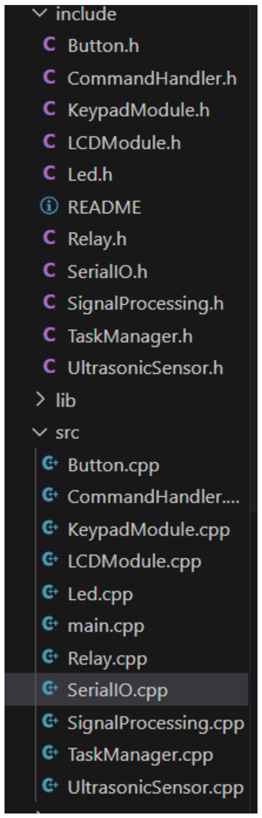


Figure 7 Layered diagram

This layered architecture clearly separates the responsibilities of each part of the system. The user interface handles the input, the application layer processes the commands and manages the relay, and the hardware layer controls the physical relay module. This modular approach makes the system easier to understand, maintain, and extend in the future.

- **Modular implementation**



```
include
  Button.h
  CommandHandler.h
  KeypadModule.h
  LCDModule.h
  Led.h
  README
  Relay.h
  SerialIO.h
  SignalProcessing.h
  TaskManager.h
  UltrasonicSensor.h
lib
src
  Button.cpp
  CommandHandler....
  KeypadModule.cpp
  LCDModule.cpp
  Led.cpp
  main.cpp
  Relay.cpp
  SerialIO.cpp
  SignalProcessing.cpp
  TaskManager.cpp
  UltrasonicSensor.cpp
```

Figure 7 Project organization

In the SerialIO.h we have the following methods for reading and writing the text:

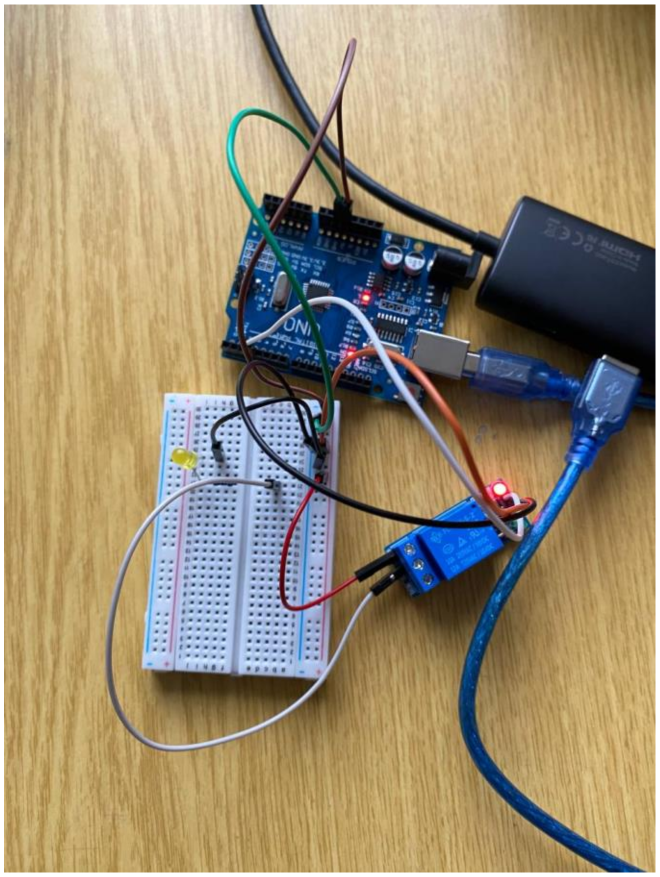
1. serial_putc(char c, FILE *stream) – it uses Serial.write to write the text
2. serial_getc(FILE *stream) – it uses Serial.read serial is not available
3. printDistance(float distance) – it prints the distance registered by sensor

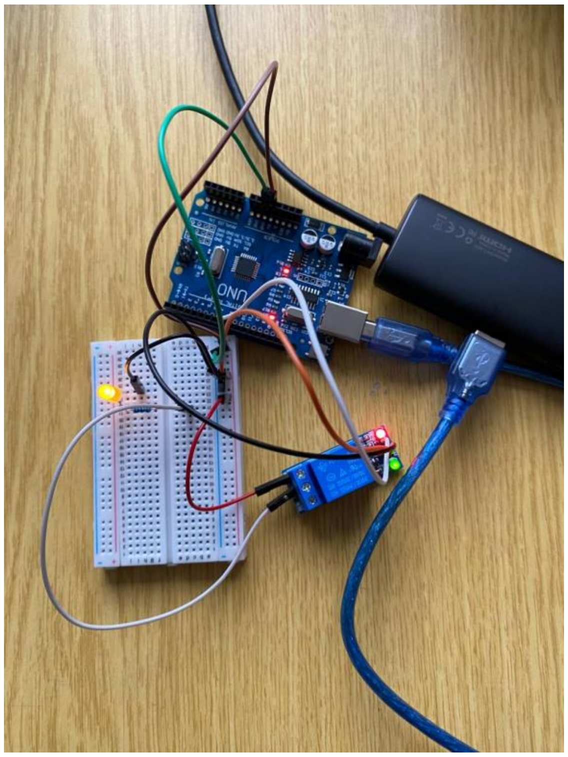
In the Relay.h we have the following methods to change the state of the relay:

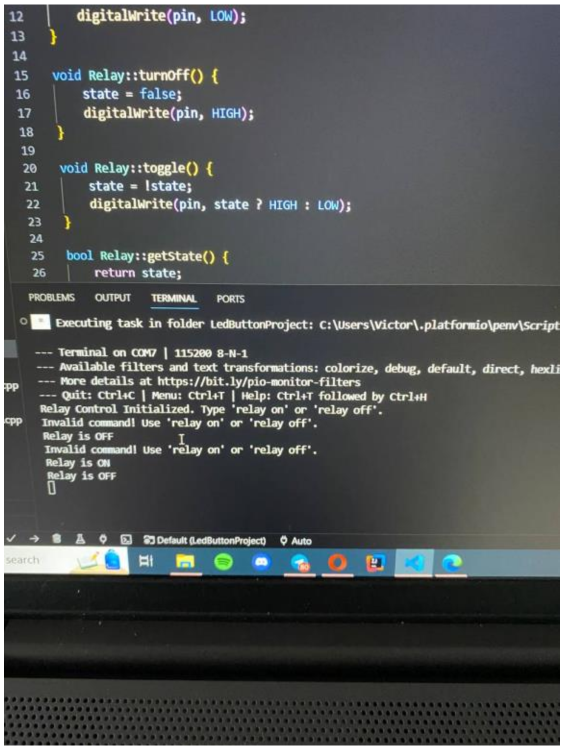
1. turnOn() – changes the state of the relay module to ON
2. turnOff() – changes the state of the relay module to OFF

3. toggle() – changes the state to the opposite, if ON then OFF, if OFF then ON

Results







```cpp
    digitalWrite(pin, LOW);
}

void Relay::turnOff() {
    state = false;
    digitalWrite(pin, HIGH);
}

void Relay::toggle() {
    state = !state;
    digitalWrite(pin, state ? HIGH : LOW);
}

bool Relay::getState() {
    return state;
```

```
 *  Executing task in folder LedButtonProject: C:\Users\Victor\.platformio\penv\Script

--- Terminal on COM7 | 115200 8-N-1
--- Available filters and text transformations: colorize, debug, default, direct, hexli
--- More details at https://bit.ly/pio-monitor-filters
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H
Relay Control Initialized. Type 'relay on' or 'relay off'.
Invalid command! Use 'relay on' or 'relay off'.
Relay is OFF
Invalid command! Use 'relay on' or 'relay off'.
Relay is ON
Relay is OFF
```

Conclusions

In this project, we created a relay control system that enables users to operate a relay module using basic serial commands such as "on" or "off". The system integrates core ideas of hardware control, serial communication, and state management, and is organized around the use of a Relay class and a finite state machine (FSM) to govern system behavior.

The Relay class manages the relay's state, providing methods for controlling, toggling, and getting its current status. The main software receives user input, processes commands, and sends feedback over the serial interface. This design keeps the system responsive and user-friendly, with error handling that alerts the user if an invalid command is entered.

Bibliography

1. Official Arduino Documentation
   - Arduino Reference – Serial Communication https://www.arduino.cc/reference/en/#communication
   - Arduino Mega 1280 Pinout & Datasheet https://docs.arduino.cc/hardware/mega-1280
2. PlatformIO Official Documentation
   - PlatformIO for Arduino Development https://docs.platformio.org/en/latest/platforms/atmelavr.html
3. TUM Courses
   - Introducere în Sistemele Embedded și Programarea Microcontrolerelor
   - Principiile comunicației seriale și utilizarea interfeței UART

Appendix

1. **GitHub**: https://github.com/Kipitokisk/SI_Lab

```
#include "SerialIO.h"
#include "Relay.h"

#define RELAY_PIN 8  // Change as per your setup
```

```
Relay relay(RELAY_PIN);

void setup() {
    serialInit();
    printf("Relay Control Initialized. Type 'relay on' or 'relay off'.\n");
}

void loop() {
    char command[20];
    if (Serial.available() > 0) {
        Serial.readBytesUntil('\n', command, sizeof(command) -1);
        command[strcspn(command, "\r\n")] = 0;

        if (strcmp(command, "relay on") == 0) {
            relay.turnOn();
            printf("Relay is ON\n");
        }
        else if (strcmp(command, "relay off") == 0) {
            relay.turnOff();
            printf("Relay is OFF\n");
        }
        else {
            printf("Invalid command! Use 'relay on' or 'relay off'.\n");
        }
    }
}
```

```
#include "Relay.h"

Relay::Relay(uint8_t pin) {
    this->pin = pin;
    this->state = false;
    pinMode(pin, OUTPUT);
    digitalWrite(pin, LOW);
}

void Relay::turnOn() {
    state = true;
    digitalWrite(pin, LOW);
}

void Relay::turnOff() {
    state = false;
    digitalWrite(pin, HIGH);
}

void Relay::toggle() {
    state = !state;
    digitalWrite(pin, state ? HIGH : LOW);
}

bool Relay::getState() {
    return state;
}
```

```
#include "SerialIO.h"

int serial_putchar(char c, FILE* f) {
    Serial.write(c);
    return c;
}

int serial_getchar(FILE* f) {
    while (!Serial.available());
    return Serial.read();
}

FILE serial_stdout;

void serialInit() {
    Serial.begin(115200);
    while (!Serial);

    fdev_setup_stream(&serial_stdout, serial_putchar, serial_getchar,
_FDEV_SETUP_WRITE);
    stdout = &serial_stdout;
    stdin = &serial_stdout;
}

void printRawDistance(float distance) {
    char buffer[10];
    dtostrf(distance, 5, 2, buffer);
    printf("Distance: %s cm\n", buffer);
}

void printSPDistance(float distance) {
    char buffer[10];
    dtostrf(distance, 5, 2, buffer);
    printf("Filtered distance: %s cm\n", buffer);
}

void printWADistance(float distance) {
    char buffer[10];
    dtostrf(distance, 5, 2, buffer);
    printf("Weighted average: %s cm\n", buffer);
}
```