Ministerul Educației și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică și Microelectronică

Departamentul Inginerie Software și Automatică

Report

Laboratory work nr. 4.1

Actuatori cu Interfață Binară - Control Releu

Embedded Systems

Prepared by:

Berco Andrei, FAF-221

Checked by:

Martîniuc Alexei, *university assistant*

Chișinău – 2025

Domain Situation Analysis

Description of the Technologies Used and the Context of the Developed Application

The developed application employs key technologies and libraries to interact with the microcontroller and control multiple tasks within a system. The core technology revolves around task scheduling and non-preemptive task execution, facilitated through a real-time operating system (RTOS) framework. This framework ensures that the tasks run sequentially without interruption, in a manner that allows for efficient synchronization and resource sharing.

Presentation of the Hardware and Software Components Used

- Microcontroller (Arduino Mega 2560): Represents the central processing unit of the system, managing inputs and outputs and interpreting commands transmitted via the serial interface.
- Visual Studio Code with PlatformIO: The IDE used to develop the application. PlatformIO is a development platform that facilitates writing and uploading code to the microcontroller.
- Relay – A relay is an electrically operated switch that allows you to control a circuit by using a low-power signal to activate a high-power circuit. It consists of a coil, which when energized, creates a magnetic field that causes the switch to close or open, depending on the relay type. Relays are commonly used in Arduino projects to control devices like motors, lights, and other high-voltage components because they provide electrical isolation between the control circuit and the device being controlled. This makes relays an essential component when working with circuits that require switching high currents or voltages.

Explanation of the System Architecture and Justification of the Chosen Solution

The system consists of a microcontroller unit (MCU) that controls a binary actuator (such as a light bulb or fan) via a relay. User commands are received through a terminal serial interface or a keypad. The system responds by controlling the relay (i.e., turning the actuator on or off) and provides feedback via the serial interface or an LCD display. The architecture is modular, with each peripheral (relay, keypad, LCD) managed by separate software modules, allowing for easier maintenance and reuse of code in future projects.

- Microcontroller (MCU): The MCU acts as the central processing unit, responsible for interpreting user commands, controlling peripherals (relay, keypad, LCD), and reporting the status of the actuator.
- Relay: The relay serves as the interface between the MCU and the actuator (light bulb or fan).

The relay allows the MCU to control the state (on/off) of the connected device by switching the high-power current.

- Keypad: The keypad is an optional input device that provides an alternative way for the user to enter commands. It is integrated into the system to offer flexibility in user interaction.
- LCD Display: The LCD is an optional output device used to display the status of the actuator (whether it's on or off). It can also provide feedback messages for the user, such as command confirmations or error messages.
- Serial Interface: The serial interface (such as a USB-to-serial connection) allows the user to send commands via a terminal (e.g., PuTTY, Arduino IDE serial monitor). The serial communication is used for receiving user inputs and providing feedback.

Relevant Case Study

A relevant case study for the proposed solution is the implementation of a temperature monitoring and filtering system in an embedded environment. Such a system can be used in industrial, medical, or home automation applications where accurate and stable temperature readings are required despite sensor noise.

Architectural Design

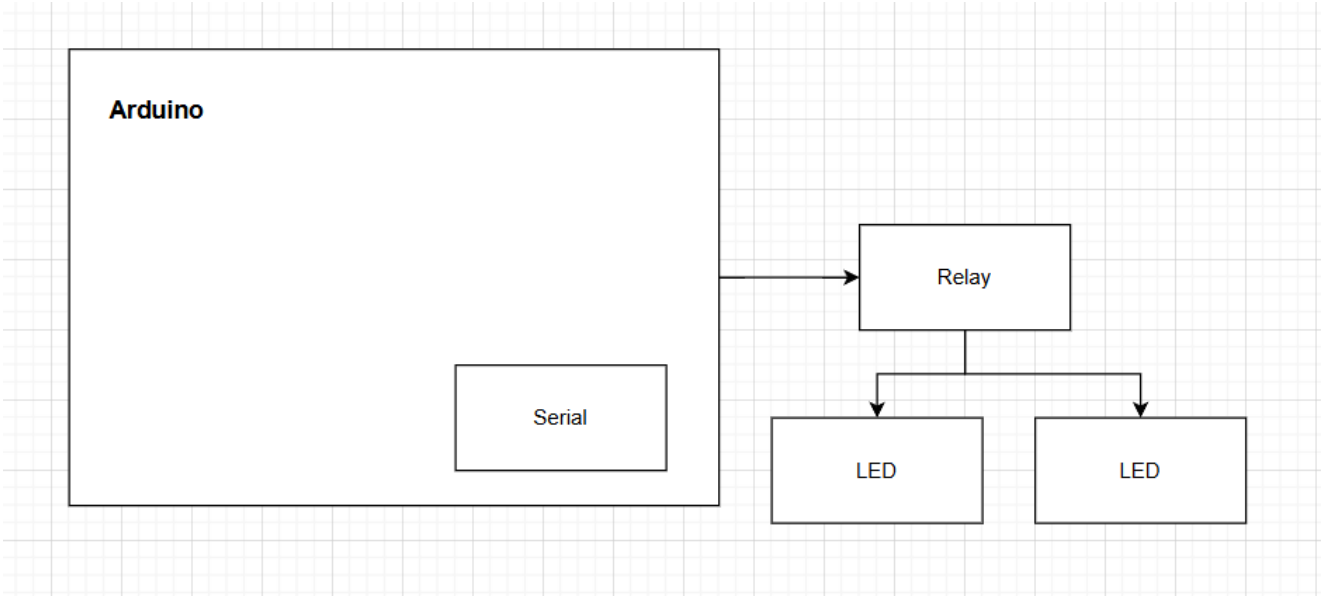
The architectural design of this system aims to create a modular, scalable, and maintainable framework for controlling binary actuators (such as lights or fans) through a relay, while providing feedback and user interaction via a serial interface or keypad. The system is built around the core principles of modularity, abstraction, and clarity, ensuring that it can be easily modified or extended in the future.

- Microcontroller Unit (MCU): The MCU is the heart of the system, managing input/output (I/O) operations, processing user commands, and controlling peripherals. It handles all of the logic, such as receiving commands, controlling the relay, and providing status feedback.
- Relay Module: The relay serves as the physical interface between the MCU and the actuator (light, fan, etc.). The relay can switch high-voltage or high-current devices on and off based on the signals from the MCU, which are generated via digital output pins.
- Keypad (Optional): The keypad serves as an alternative input device, allowing the user to input commands (e.g., relay on or relay off) via button presses. The keypad module scans the pressed keys and passes the corresponding command to the MCU.
- LCD Display (Optional): The LCD display shows real-time feedback to the user. It displays the status of the actuator (whether it's on or off) and other relevant information, such as

confirmation messages or errors.

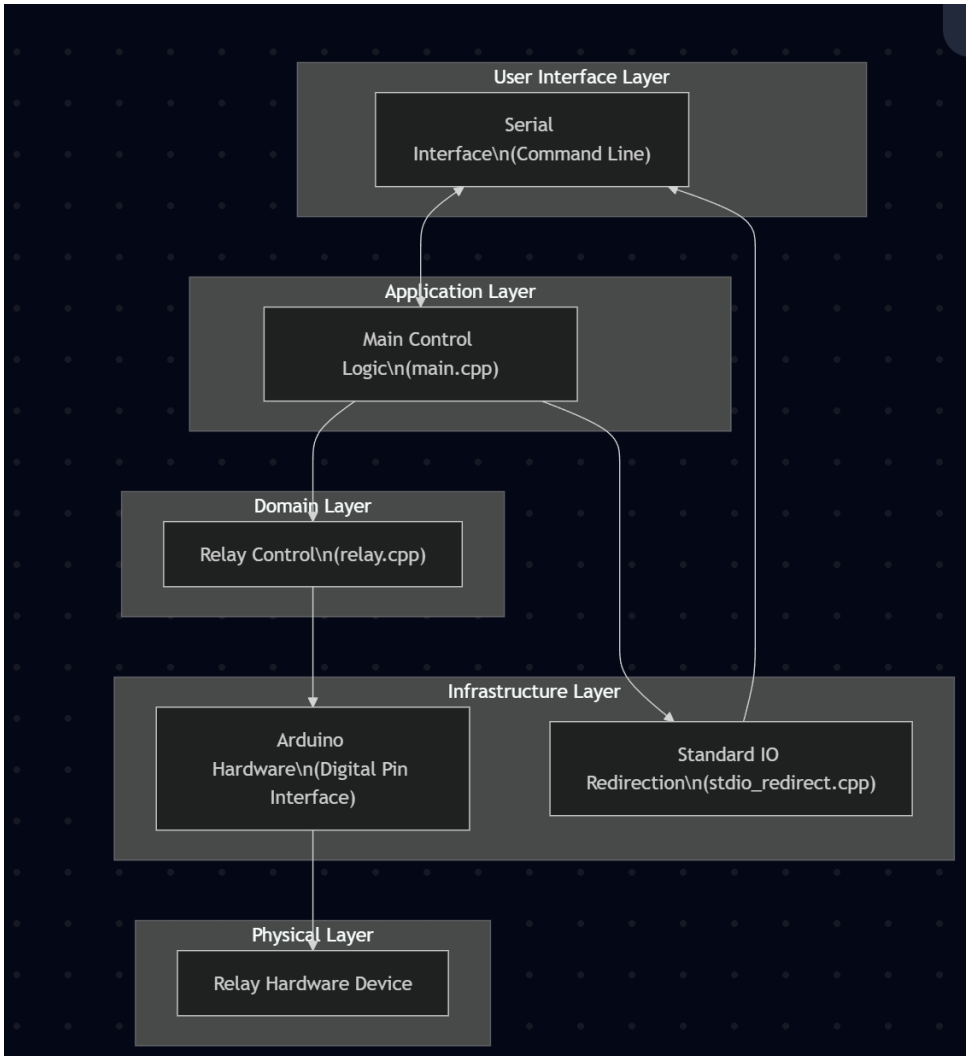
- Serial Interface: The serial interface (usually connected to a PC or terminal) allows the user to communicate with the MCU through text-based commands. This provides an alternate way to control the relay, making the system suitable for debugging and general use.

Hardware Diagram



Both LED1 and LED2 are connected to two separate digital pins on the MCU. Each LED should be connected with a current-limiting resistor (typically 220Ω to 1kΩ) in series to prevent excess current from damaging the LED.

Layer Diagram



User Interface Layer:

- The Serial Interface provides a command-line interface for users to interact with the system
- Commands like "relay on" and "relay off" are entered through this interface

Application Layer:

- Contains the main control logic (main.cpp)
- Handles command parsing and processing
- Orchestrates the overall program flow
- Acts as the central coordinator between user input and system actions

Domain Layer:

- The Relay Control component (relay.cpp) encapsulates domain-specific logic
- Provides methods to control the relay state
- Maintains the state of the relay
- Abstracts the hardware control details from the application layer

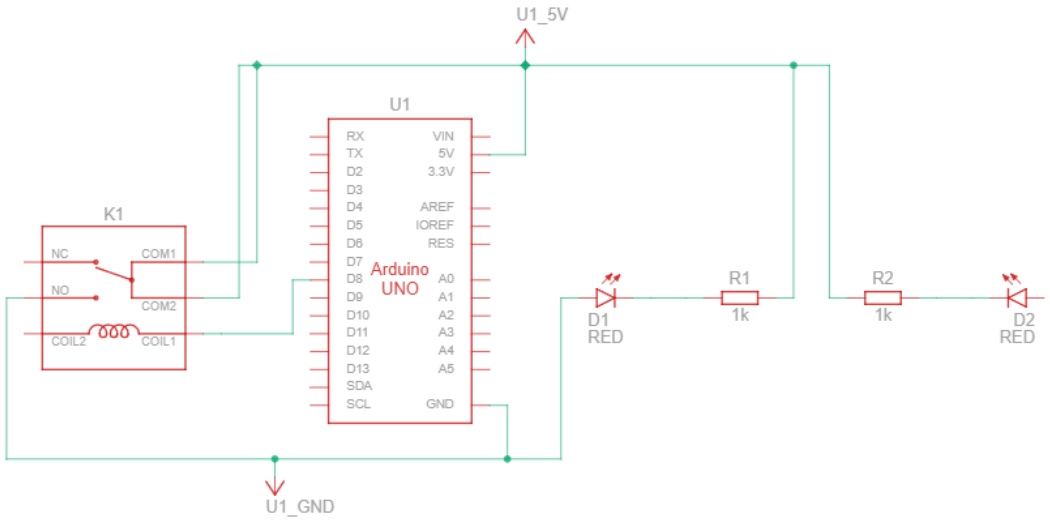
Infrastructure Layer:

- Standard IO Redirection (stdio_redirect.cpp) enables the use of standard C I/O functions
- Arduino Hardware interface manages direct interaction with the digital pins
- Provides the underlying platform services that support the higher-level functionality

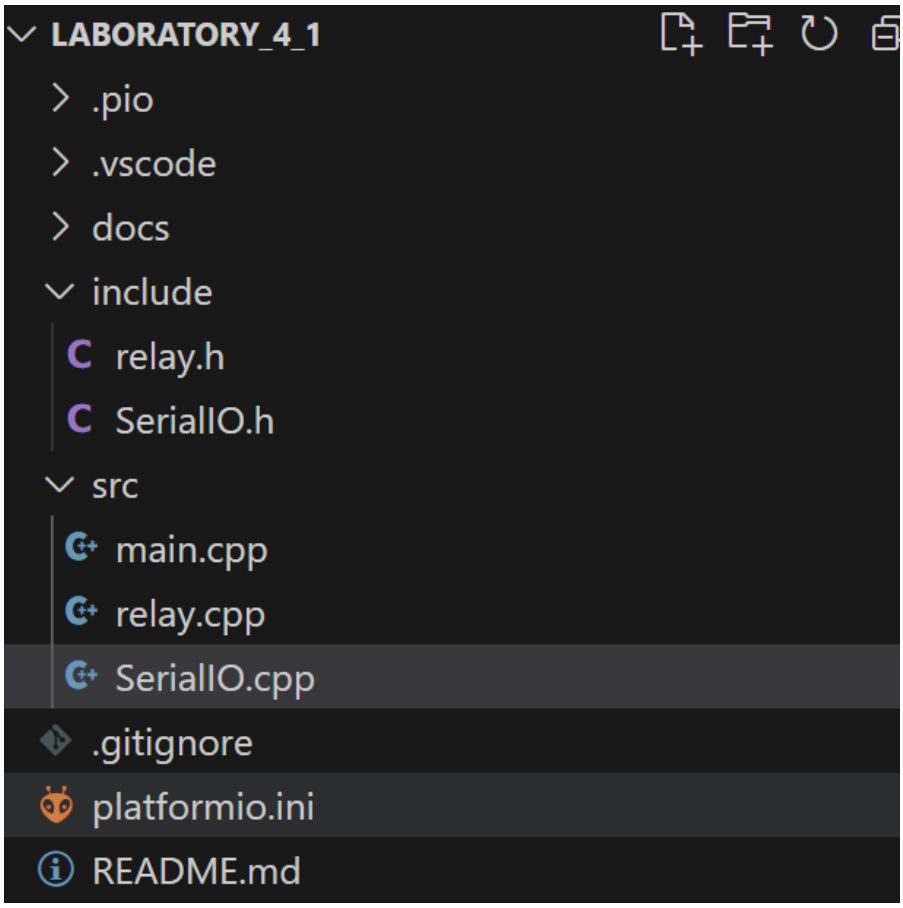
Physical Layer:

- The actual relay hardware device that is being controlled
- The physical component that switches the connected electrical circuit

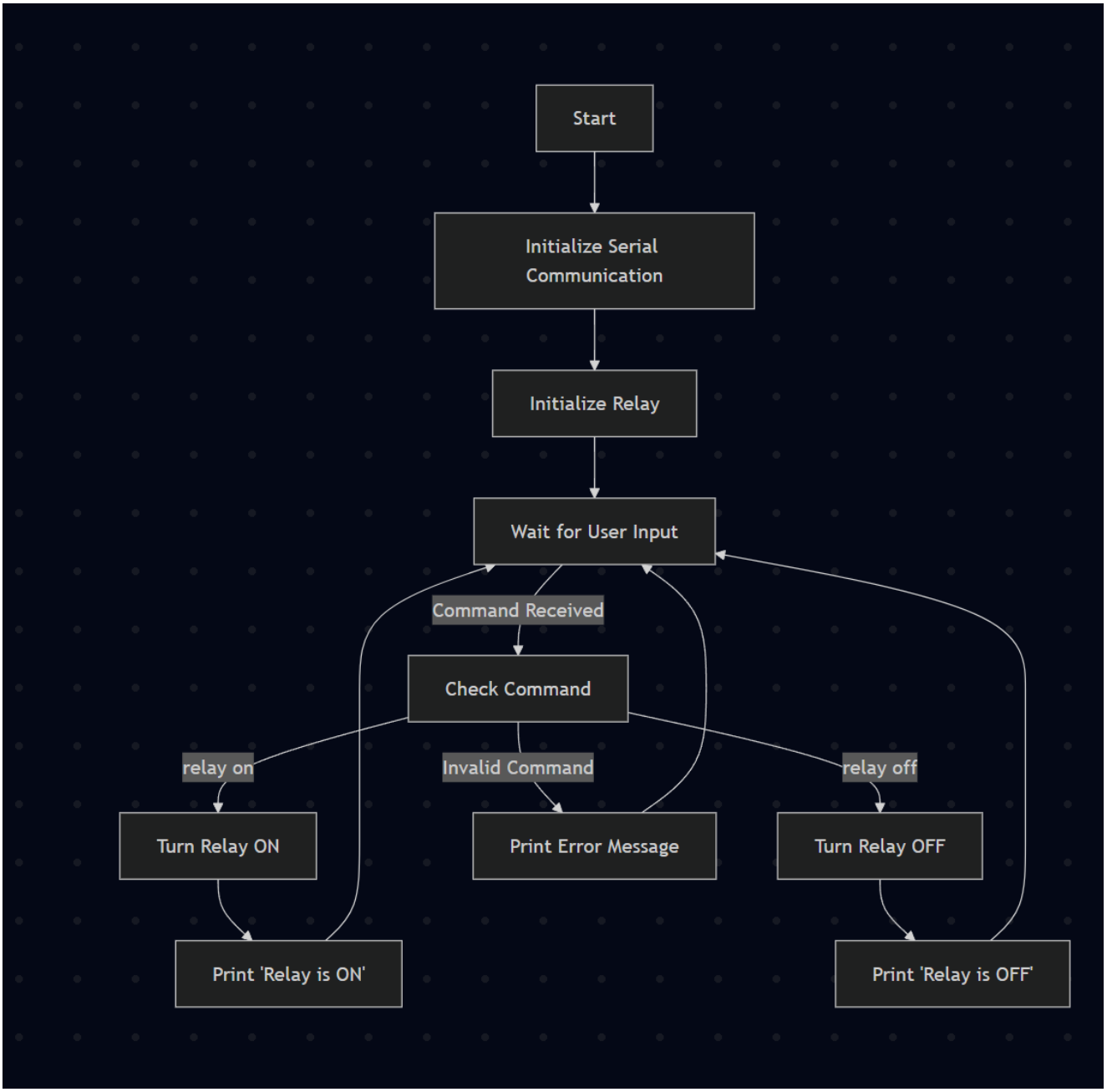
Electrical Scheme



Project Structure and Modular Implementation



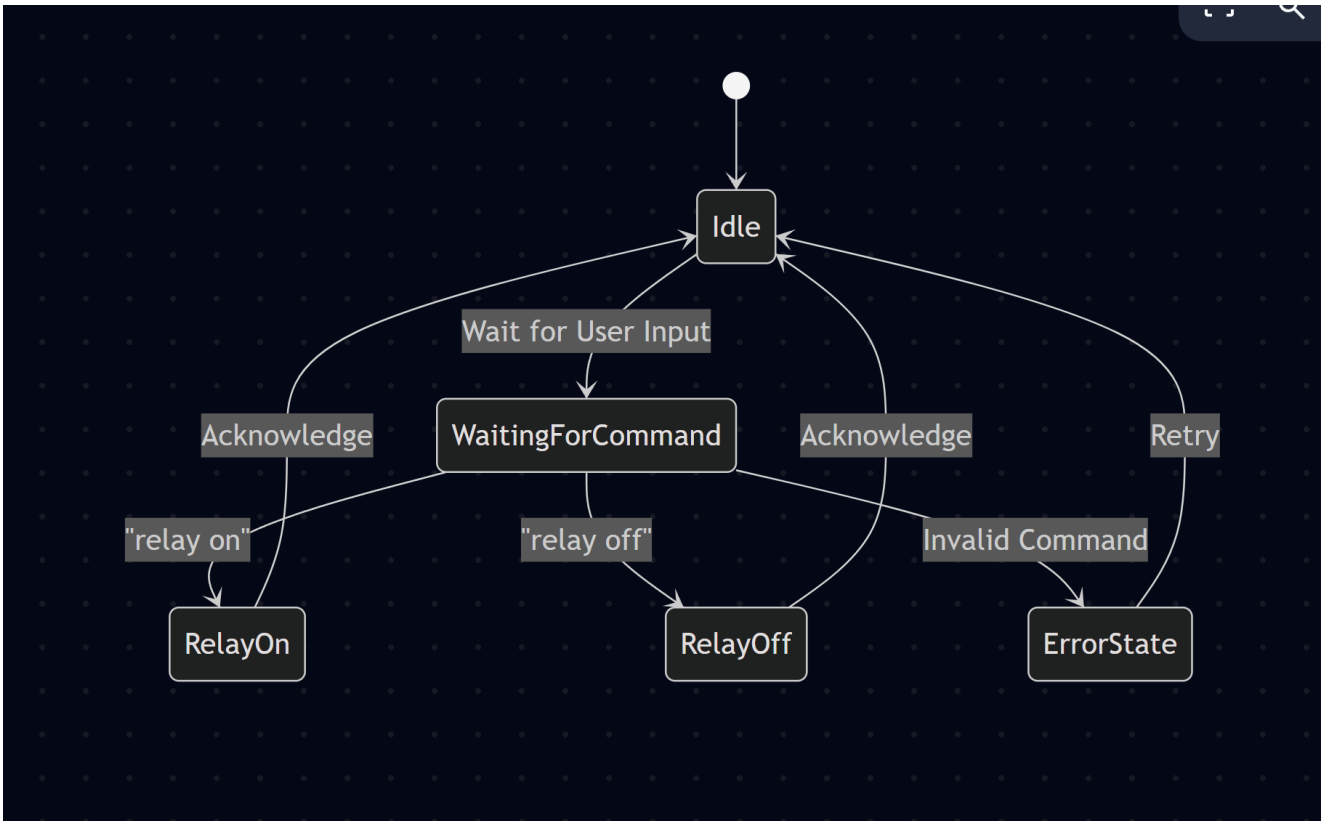
Block Diagram of System Behaviour



- The system starts by initializing serial communication.
- The relay is also initialized with an initial state of OFF.
- The system enters a loop, waiting for user input via the serial monitor.
- When a command is received, it checks whether it is "relay on" or "relay off".
- If "relay on", the relay is turned **ON**, and a confirmation message is printed.
- If "relay off", the relay is turned **OFF**, and a confirmation message is printed.
- If the command is invalid, an error message is displayed.

- The process repeats indefinitely.

Functional Block Diagrams

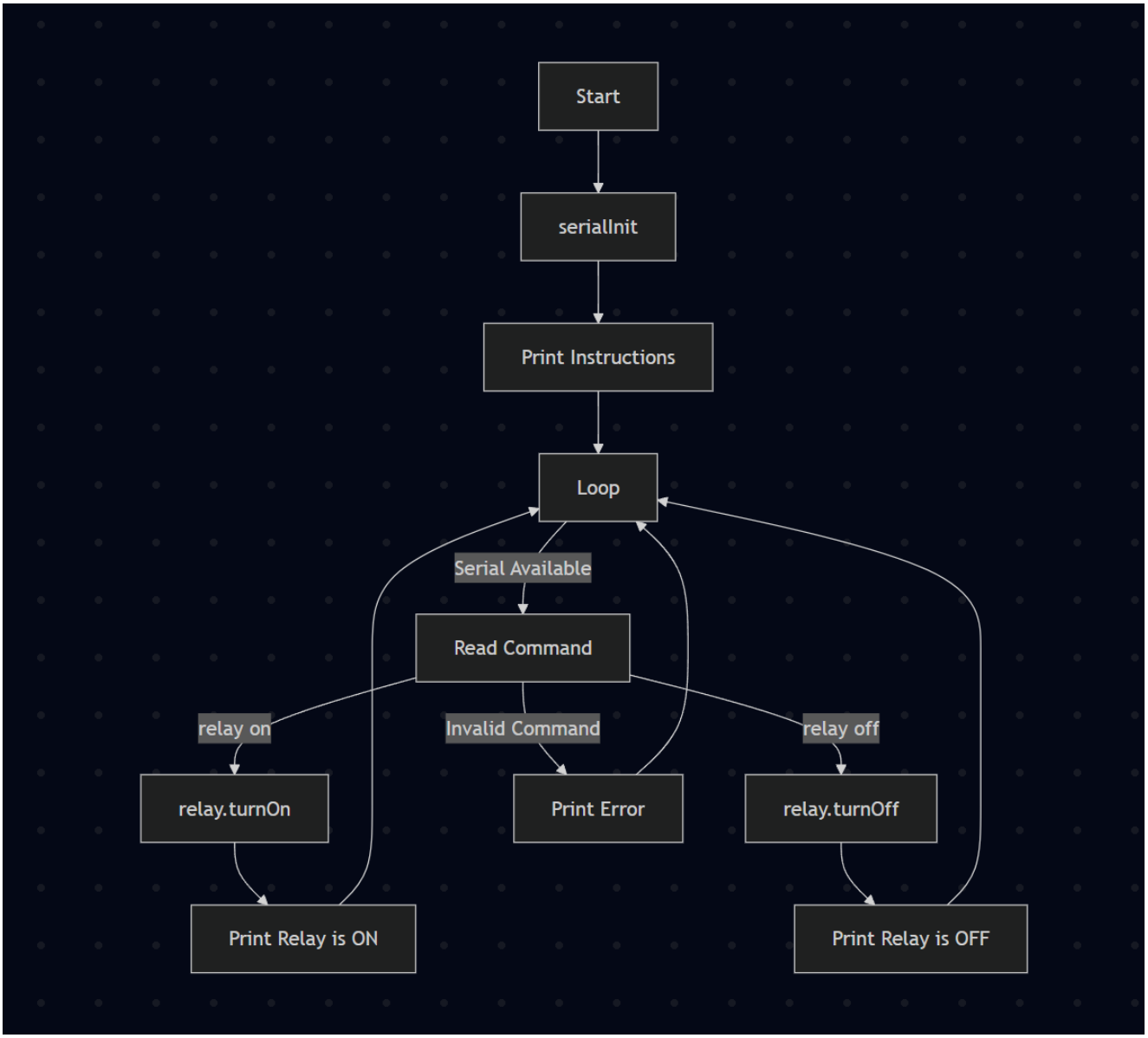


Description:

- The system starts in the Idle state.
- When user input is received, it transitions to WaitingForCommand.
- If the input is "relay on", it moves to RelayOn, where the relay is activated.
- If the input is "relay off", it moves to RelayOff, where the relay is deactivated.
- If an invalid command is received, it transitions to ErrorState, displays an error message, and returns to Idle.
- After each command execution, the system returns to Idle, waiting for new input.

Code flowchart:

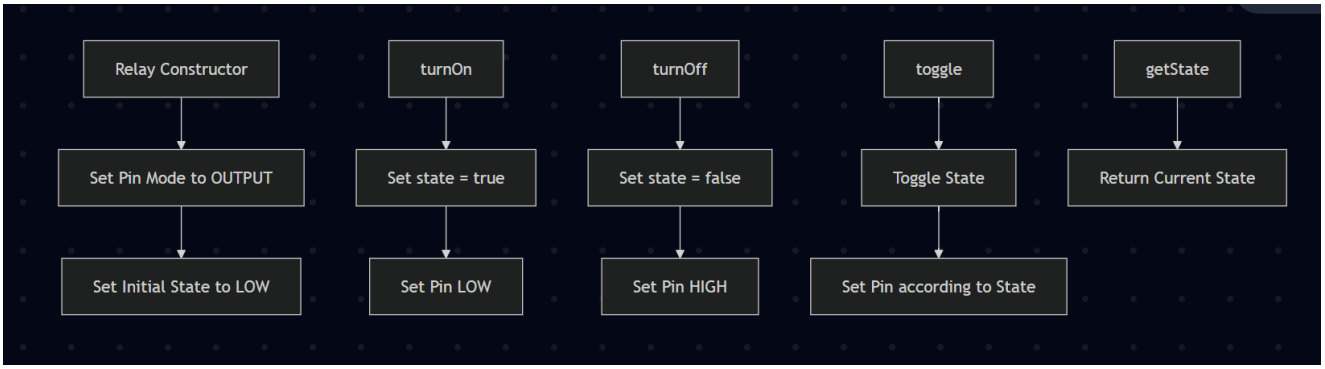
Main.cpp



Description:

1. The system initializes serial communication using serialInit().
2. A message is printed to the user, explaining the available commands (relay on and relay off).
3. The program enters an infinite loop, waiting for user input.
4. If data is available in the serial buffer, it reads the command.
5. Depending on the command:
    - "relay on" → Calls relay.turnOn() and prints "Relay is ON".
    - "relay off" → Calls relay.turnOff() and prints "Relay is OFF".
    - Invalid commands → Display an error message.
6. The process repeats indefinitely.

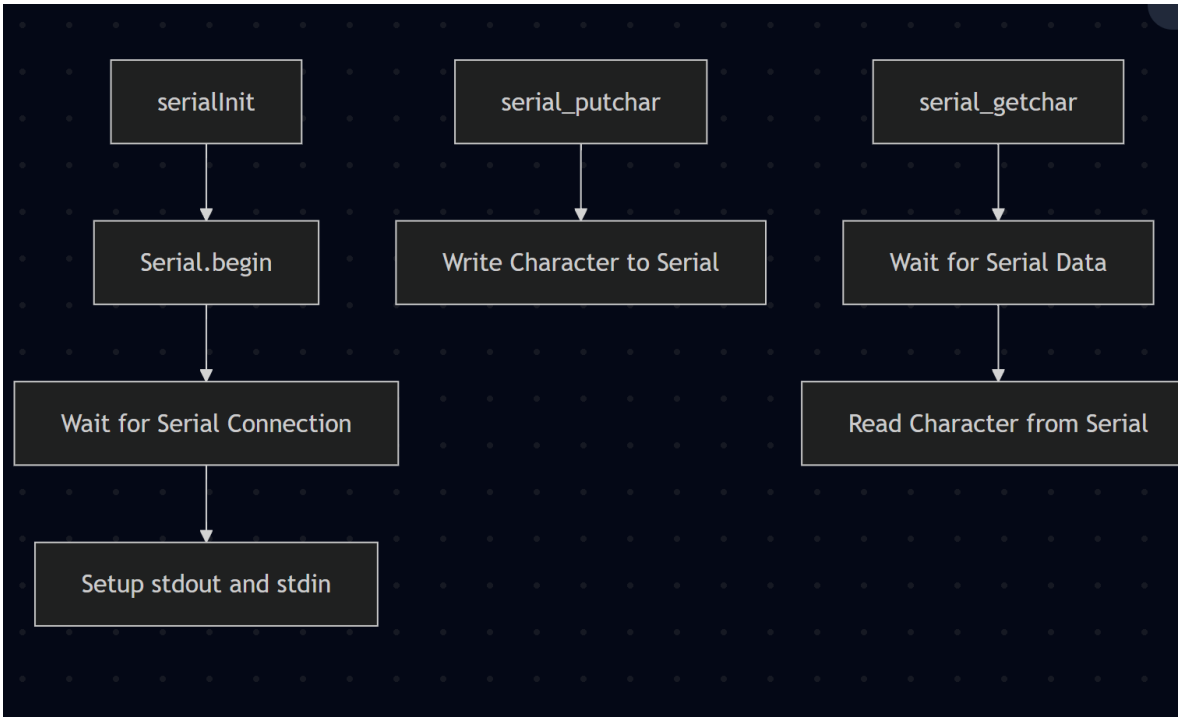
Relay.cpp



Description:

- Constructor: When a Relay object is created, it sets the pin mode to OUTPUT and initializes the state to LOW (off).
- turnOn(): Sets the relay to ON (LOW signal) and updates the state.
- turnOff(): Sets the relay to OFF (HIGH signal) and updates the state.
- toggle(): Switches between ON and OFF states dynamically.
- getState(): Returns the current state of the relay (ON/OFF).

SerialIO.cpp



Description:

- serialInit():
  - Initializes serial communication at 115200 baud rate.
  - Waits for the serial port to be available.
  - Redirects stdout and stdin to use the serial connection.
- serial_putchar(): Writes a character to the serial output.
- serial_getchar(): Waits for input from the serial monitor and reads the received character.

Conclusion

In this laboratory work, we successfully designed and implemented a modular system to control binary actuators (such as lights or fans) using a relay, driven by a microcontroller (Arduino Uno). The system receives commands from the user through a serial interface, processes these commands, and toggles the state of the relay accordingly.

During the preparation of this report, the author used Grok for generating/strengthening the content. The resulting information was reviewed, validated, and adjusted according to the requirements of the laboratory work.

References

1. IoT-24 : Lab 3 2. - https://www.youtube.com/watch?v=ODJjs82LrtM
2. Arduino Documentation about Seria - https://docs.arduino.cc/language-reference/en/functions/communication/serial/
3. PlatformIO Documentation - https://docs.platformio.org/en/latest/integration/ide/pioide.html

Appendix - Source Code

Main.cpp

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

#define RELAY_PIN 8
```

```
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");
    }
  }
}
```

Relay.cpp

```
#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;
}
```

SerialIO.cpp

```
#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;
}
```