Ministry of Education of Republic of Moldova

Technical University of Moldova

Faculty of Computers, Informatics and Microelectronics

Software Engineering Department

Embedded Systems

Laboratory work work #1.1

User interaction via Serial Communication. Led

During the preparation of this report, the author used various AI Tools such as ChatGPT, Gemini, Claude to generate/augment the content, generate the code and structure the directory. The resulting information was reviewed, validated, and adjusted to meet the requirements of the laboratory assignment.

Author: Timur Cravtov
std. gr. FAF-231

Verified by:
Alexei Martîniuc
asist. univ

Chișinău, 2026

Contents

1 Objective of the work

2 System design

2.1 System architecture

The system is designed to control an LED connected to an Arduino board via serial communication. The schema of the system architecture is presented in Figure 1. The main components of the system include:

- **Arduino Board:** The microcontroller that interfaces with the LED and handles serial communication.
- **LED:** The output device that will be turned on or off based on commands received via serial communication.
- **Serial Communication Interface:** Facilitates the exchange of commands between the user and the Arduino board.
- **User Input:** The user sends commands ("led on" or "led off") through a serial terminal to control the LED state.

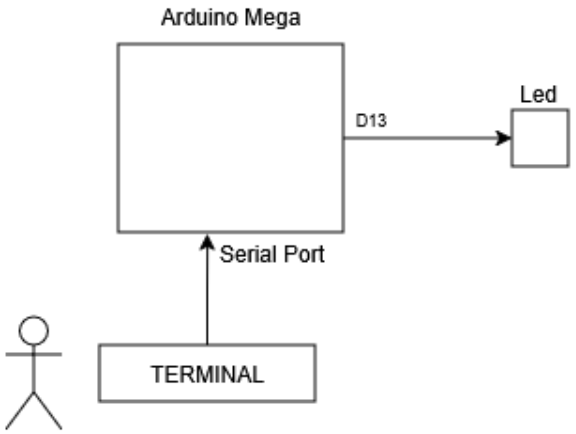


Figure 1: System Architecture Diagram

2.2 Flowchart of the program

The flowchart in Figure 2 illustrates the logic of the program running on the Arduino board. The program continuously listens for user input via serial communication and processes commands to control the LED state.

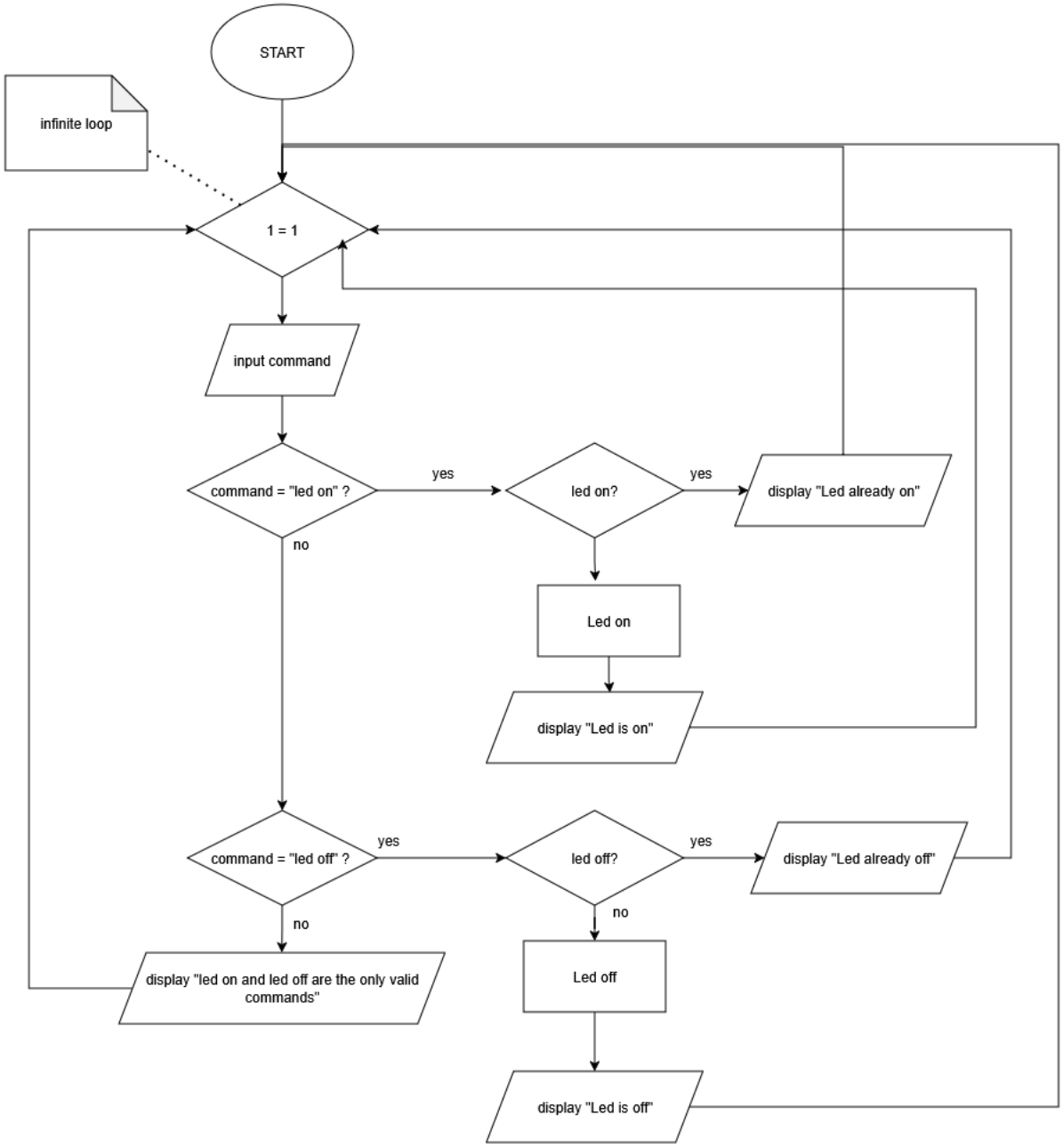


Figure 2: Program Flowchart

In this flowchart, there is an infinite loop that waits for user input. When a command is received, it checks if the command is "led on" or "led off" and turns the LED on or off accordingly. If the command is unrecognized, it prompts the user to enter a valid command.

2.3 Electronic circuit

Figure 3 illustrates the electronic circuit designed for controlling an LED using an Arduino board. The circuit consists of an LED connected to a digital output pin of the Arduino through a current-limiting resistor. The ground pin of the Arduino is connected

to the cathode of the LED, while the anode is connected to the resistor, which in turn is connected to the designated digital pin.

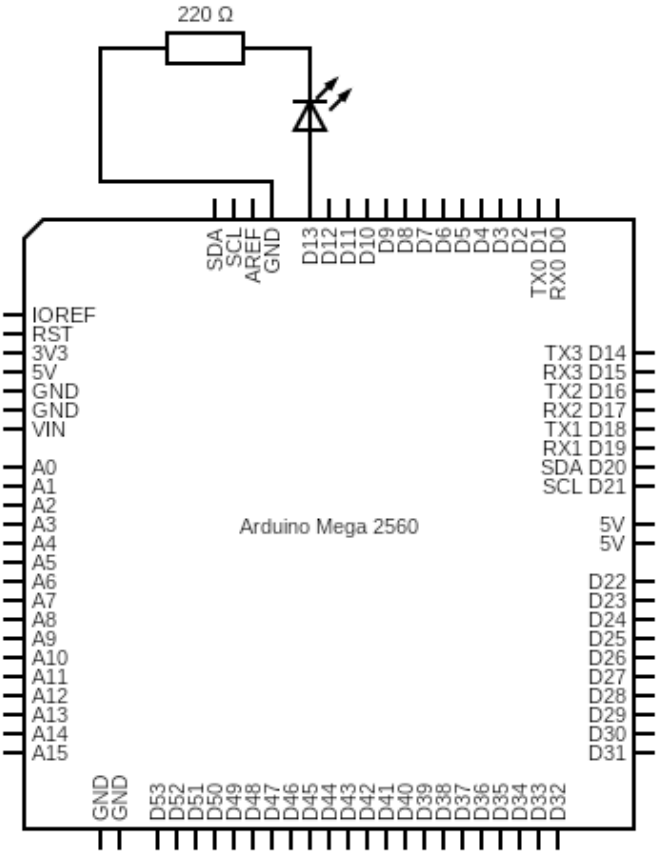


Figure 3: Circuit diagram of the LED control system

Since the Arduino board operates at 5V, a resistor value of 220Ω is chosen to limit the current through the LED to a safe level, preventing damage to both the LED and the Arduino pin.

Following the Ohm's law, the current flowing through the LED can be calculated as:

$$I = \frac{V_{supply} - V_{LED}}{R} = \frac{5V - 2V}{220\Omega} \approx 13.64mA$$

which is within the safe operating limits for both the LED and the Arduino pin.

2.4 Layered design (hardware/software)

2.4.1 LED Layered design

Figure 4 presents the layered design for the LED control module. The design is structured into six layers:

- **APP** - outermost application layer, uses higher-level functions to control the LED such as `ledOn()`
- **SRV** - service layer, provides services to the application layer and interacts with the lower layers. For example, it may include functions to manage LED states.
- **ECAL** - embedded component abstraction layer, abstracts the hardware specifics of the LED, providing a uniform interface for higher layers.
- **MCAL** - microcontroller abstraction layer, interfaces directly with the microcontroller's hardware registers to control the LED.
- **MCU** - microcontroller layer, represents the actual microcontroller hardware that controls the LED. Here - D13 pin of Arduino.
- **ECU** - embedded component unit layer, represents the physical LED component connected to the microcontroller.

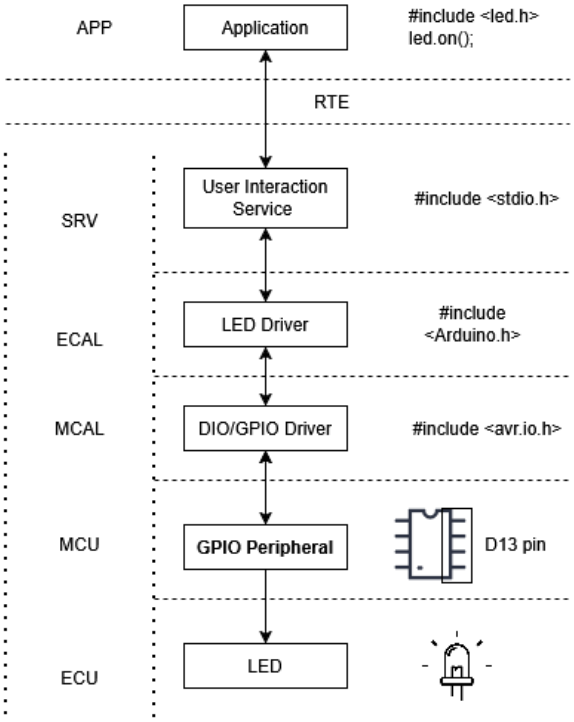


Figure 4: LED Layered Design

2.4.2 Serial I/O Layered design

Figure 5 illustrates the layered design for the Serial I/O module. The design is structured into six layers:

- **APP** - outermost application layer, uses higher-level functions to handle serial communication such as `readCommand()`

- **SRV** - service layer, provides services to the application layer and interacts with the lower layers. For example, it may include functions to manage serial data.
- **ECAL** - embedded component abstraction layer, abstracts the hardware specifics of the serial communication, providing a uniform interface for higher layers.
- **MCAL** - microcontroller abstraction layer, interfaces directly with the microcontroller's hardware registers to manage serial communication.
- **MCU** - microcontroller layer, represents the actual microcontroller hardware that handles serial communication. Here - UART module of Arduino.
- **ECU** - embedded component unit layer, represents the physical serial communication interface connected to the microcontroller, such as user terminal

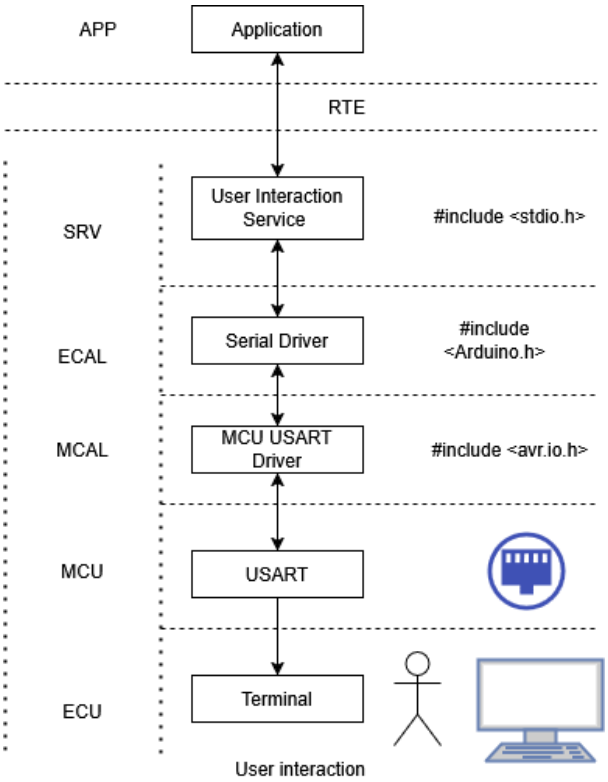


Figure 5: Serial I/O Layered Design

3 Results

The implemented system successfully allows the user to control the LED state via serial commands. When the user inputs "led on", the LED lights up, and when "led off" is entered, the LED turns off. The system responds correctly to valid commands and prompts the user for valid input when unrecognized commands are received.

The steps shows in the figures below are the following:

1. Figure 6 shows the SimulIDE project setup for the LED control system.
2. Figure 7 displays the serial monitor prompting the user for input.
3. Figure 8 illustrates the LED turned on after receiving the ”led on” command.
4. Figure 9 shows the serial monitor indicating that the LED is already on when the ”led on” command is sent again.
5. Figure 10 depicts the LED turned off after receiving the ”led off” command.
6. Figure 11 presents the serial monitor displaying an invalid command message when an unrecognized command is entered.

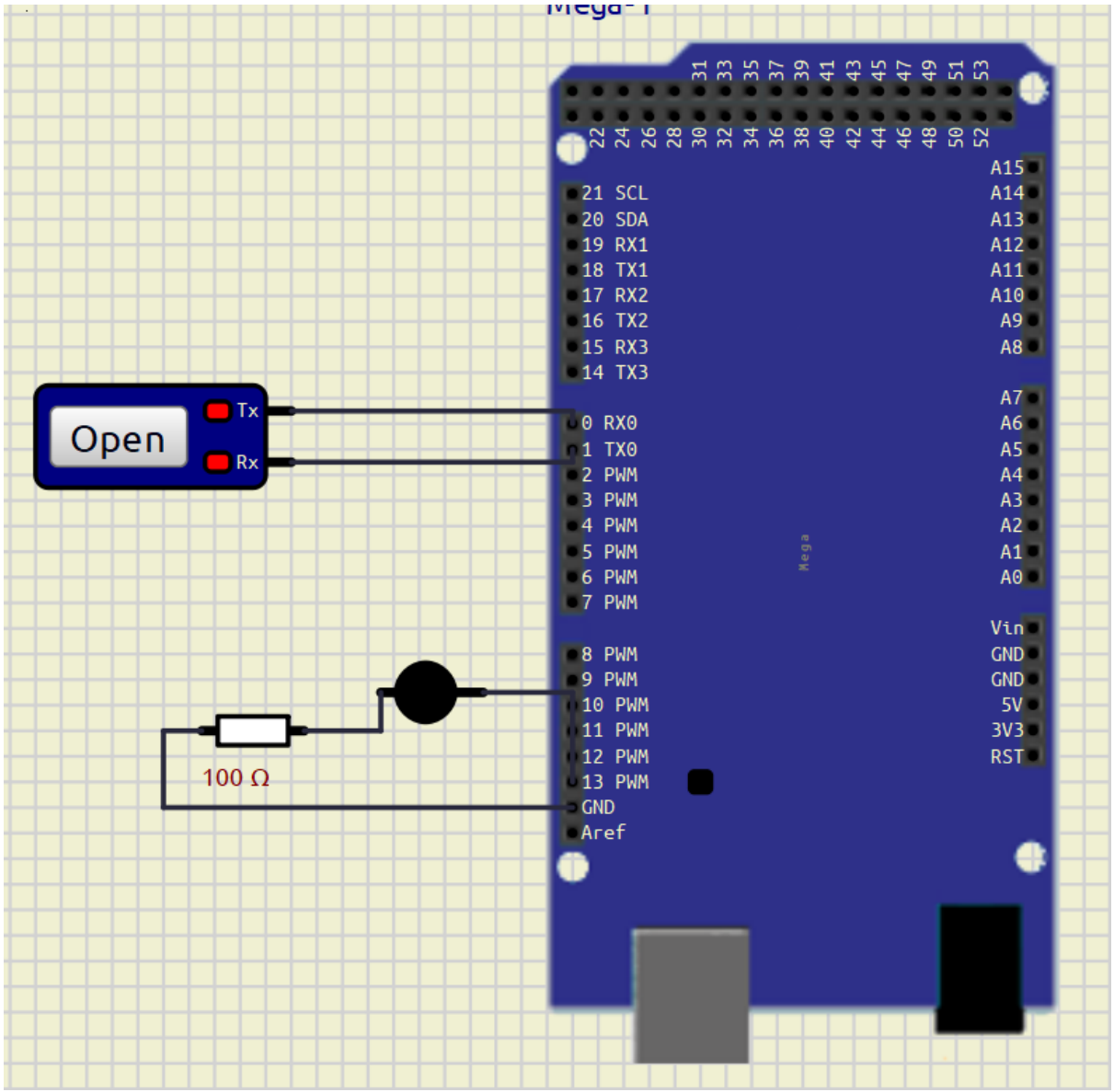


Figure 6: SimulIDE project setup for LED control

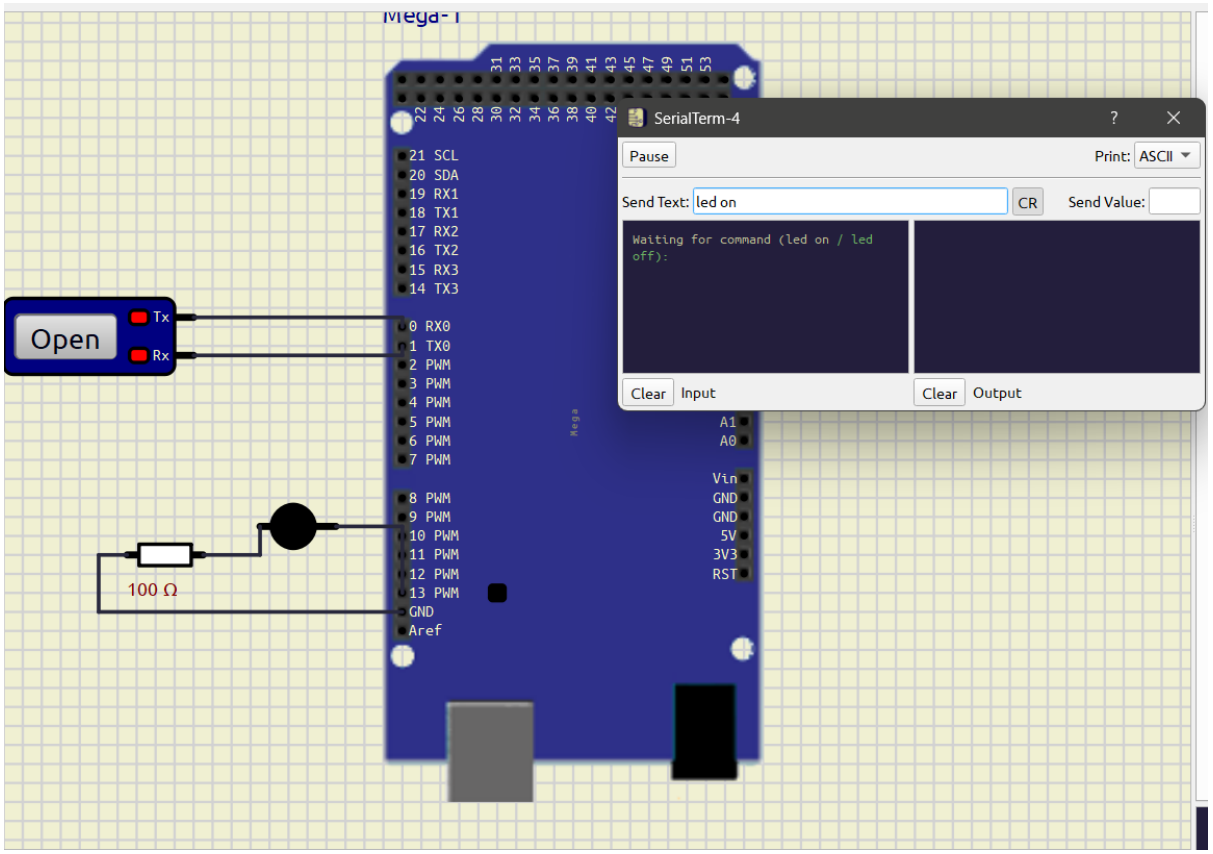


Figure 7: Serial monitor prompting for input

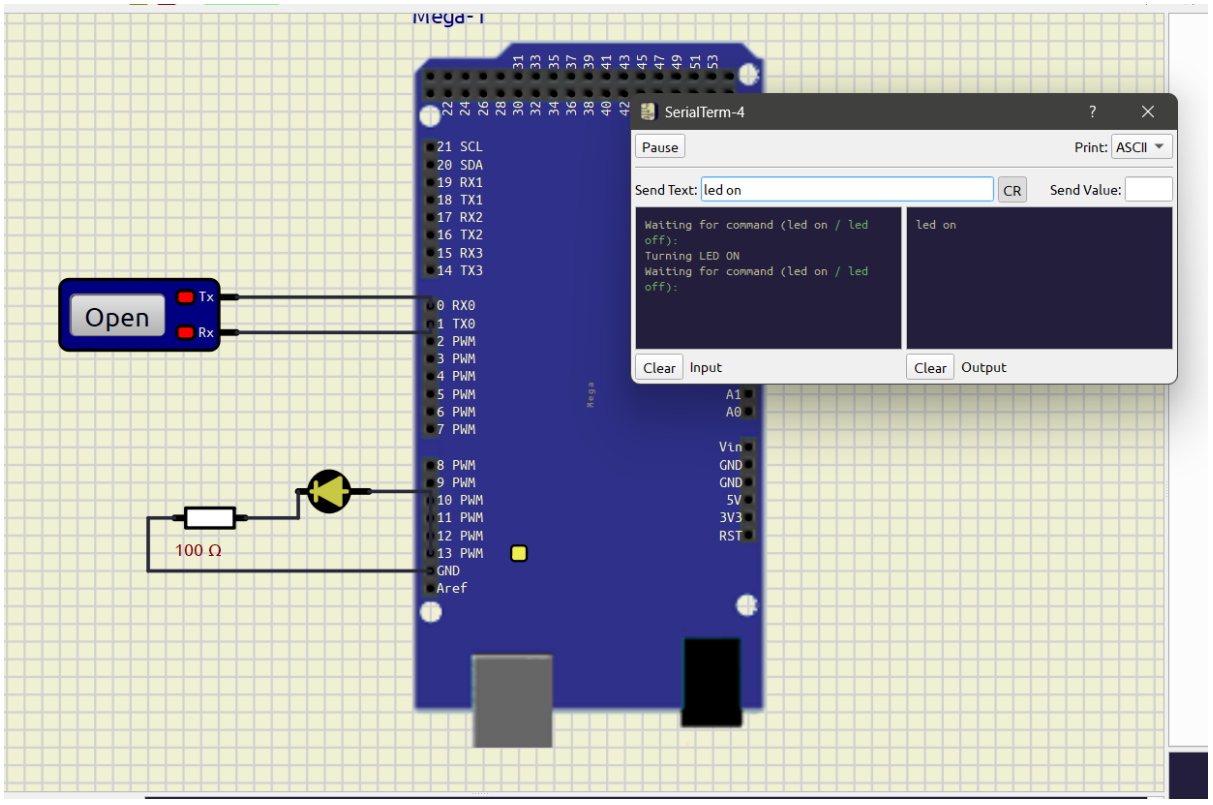


Figure 8: LED turned on after receiving ”led on” command

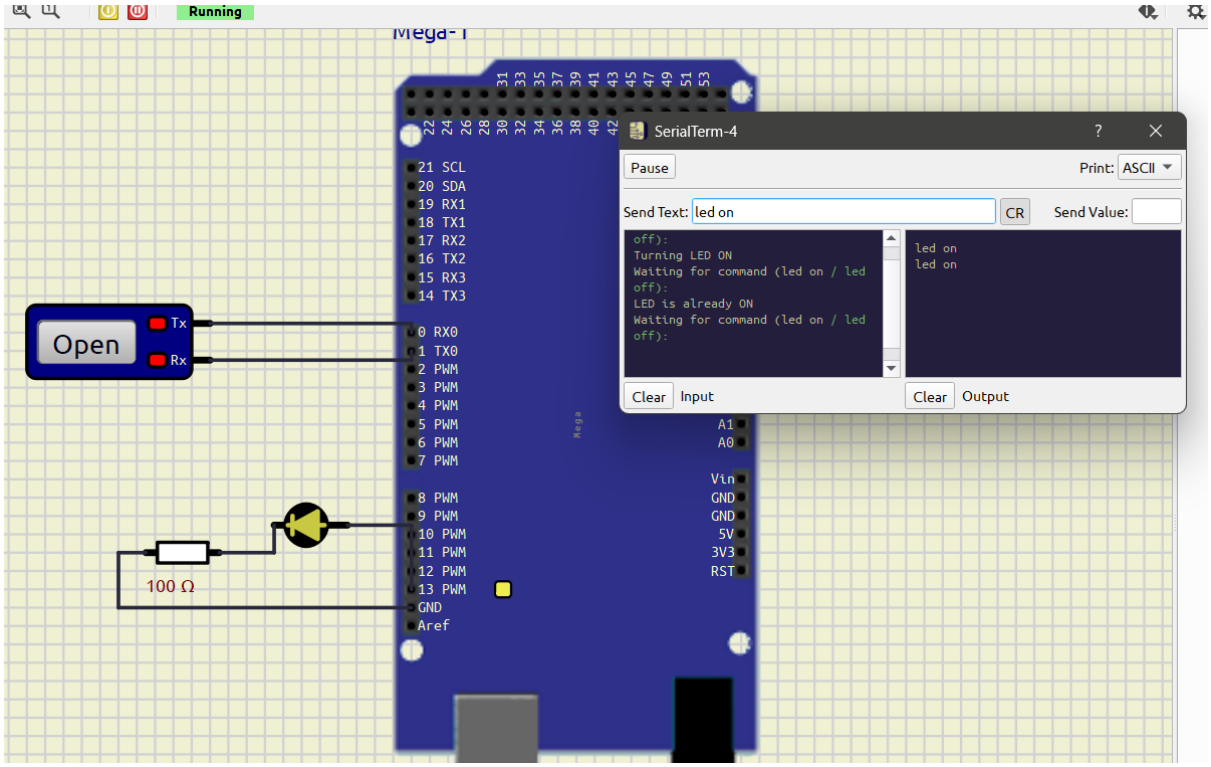


Figure 9: Serial monitor showing that LED is already on

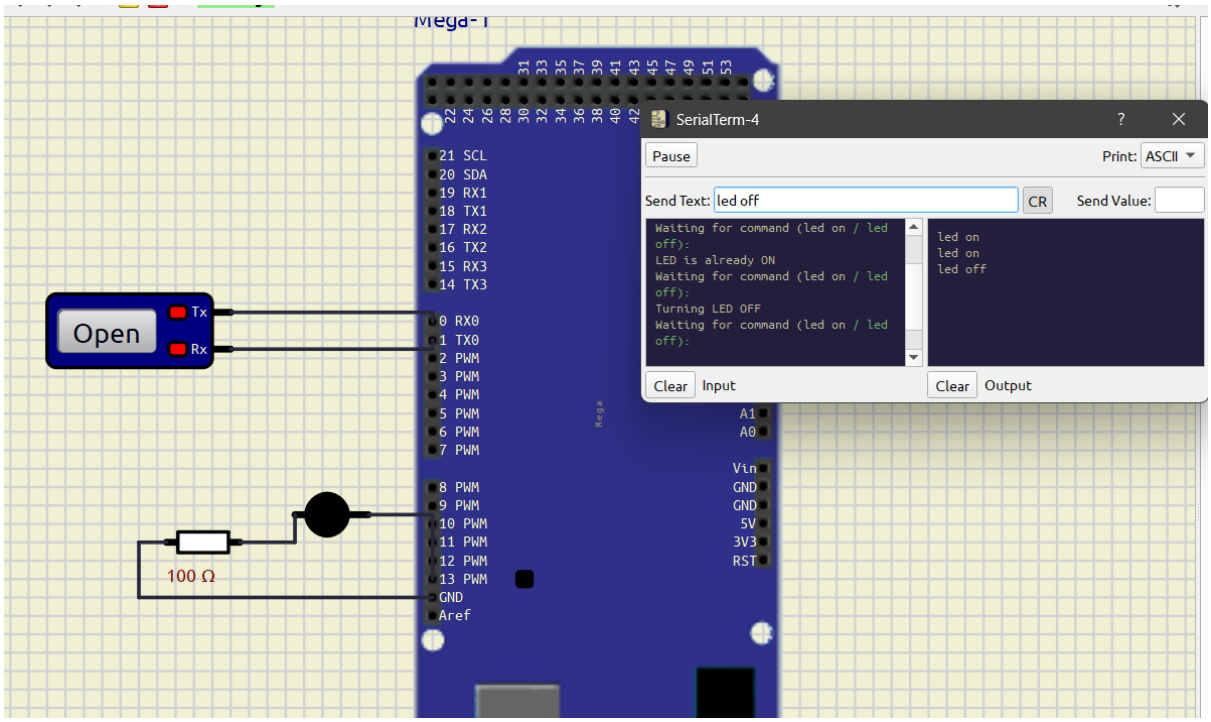


Figure 10: LED turned off after receiving "led off" command

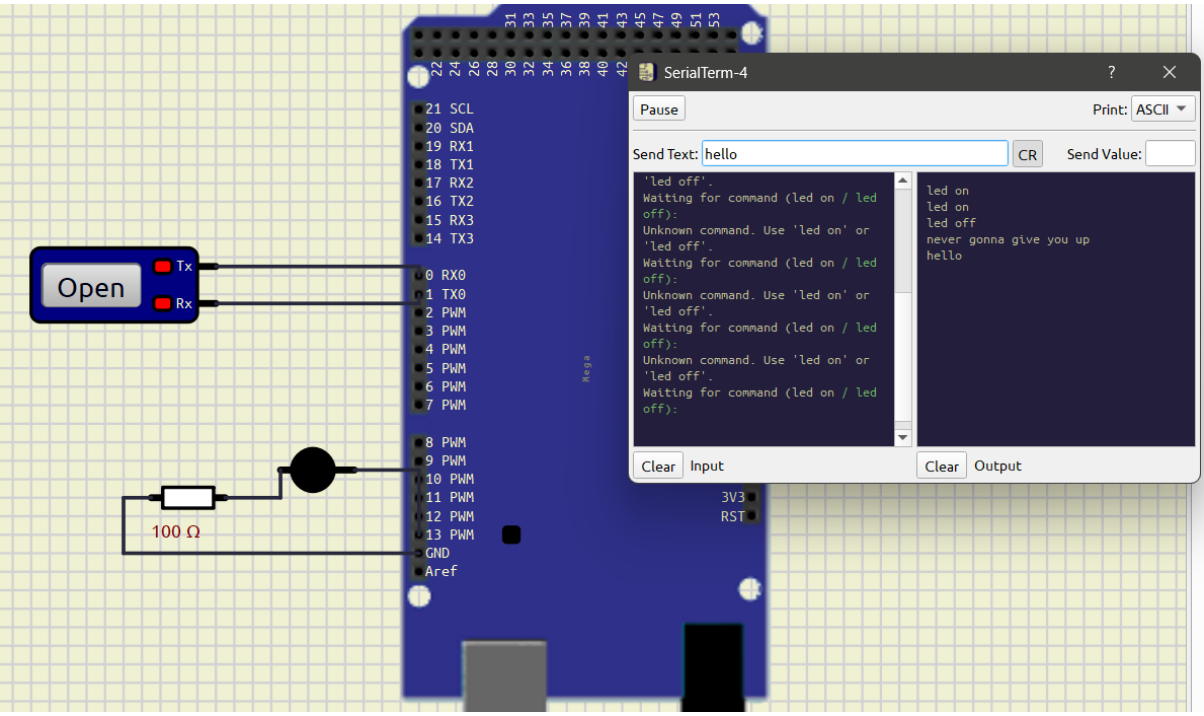


Figure 11: Serial monitor showing invalid command message

4 Conclusion

References

[1] GitHub repository `https://github.com/TimurCravtov/EmbeddedSystemsLabs`

[2] Arduino LED Control `https://roboticsbackend.com/arduino-led-complete-tutorial/`

[3] Circuit Diagram Builder `https://www.circuit-diagram.org/docs`

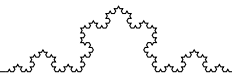
[4] SimulIDE - Simple real time electronics simulator `https://simulide.com/p/mcus-/l`

A Source Code

Besides this appendix, the source code is available in the GitHub repository [1] with all build instructions.

`main.cpp`

```
1 #include <Arduino.h>
2
3 #include <stdio.h>
```



```
4  #include <stdlib.h>
5  #include <led/led.h>
6  #include <serialio/serialio.h>
7  #include <string.h>
8  #include <LedController.h>
9
10 const uint8_t ledPinNum = 13;
11
12 Led led(ledPinNum);
13 LedController ledController(&led);
14
15 void setup() {
16
17   // start serial communication
18   Serial.begin(9600);
19   delay(100);
20
21   redirectSerialToStdio();
22
23 }
24
25 bool equalsIgnoreCase(const char* a, const char* b) {
26   return strcasecmp(a, b) == 0;
27 }
28
29 void loop() {
30
31     // reading command from serial
32     char buffer[10] = {0};
33     // fflush(stdout);
34
35     // promt to read data
36     printf("Waiting for command (led on / led off): \n");
37     readLine(buffer, sizeof(buffer));
38
39     // process command introduced by user
40     ledController.processCommand(buffer);
41
```

```
}
```

serialio.h

```h
#pragma once

#include <Arduino.h>

/// @brief Functions for serial input/output redirection and line reading
void readLine(char* buffer, size_t maxLen);
void redirectSerialToStdio();
```

serialio.cpp

```cpp
#include <Arduino.h>
#include <serialio/serialio.h>
#include <stdio.h>

#include <stdlib.h>

// add a char to serial output
int serial_putchar(char c, FILE* stream) {
  Serial.write(c);
  return 0;
}

// get a char from serial input
int serial_getchar(FILE* stream) {
  while (!Serial.available());
  return Serial.read();
}

// reads a line from stdin into buffer, up to maxLen-1 characters, and discards the rest of the line if too long
void readLine(char* buffer, size_t maxLen) {
  size_t i = 0;
  while (i < maxLen - 1) {
    int c = fgetc(stdin);
    if (c == '\n' || c == '\r' || c == EOF) {
      break;
```

```
26      }
27      buffer[i++] = (char)c;
28    }
29
30    buffer[i] = '\0';
31
32    // should i clear the rest of the line if it was too long?
33    // i guess so
34    if (i == maxLen - 1) {
35      int c;
36      do {
37        c = fgetc(stdin);
38      } while (c != '\n' && c != '\r' && c != EOF);
39    }
40  }
41
42  // Helper functions for redirecting Serial to stdio
43  void redirectSerialToStdio() {
44    static FILE uartout;
45    static FILE uartin;
46
47    // setup the UART streams for input and output
48    fdev_setup_stream(&uartout, serial_putchar, NULL,
        _FDEV_SETUP_WRITE);
49    fdev_setup_stream(&uartin, NULL, serial_getchar,
        _FDEV_SETUP_READ);
50
51    // redirect standard input and output to the UART
52    stdout = &uartout;
53    stdin = &uartin;
54    stderr = &uartout; // never used this, but why not. output
        errors to serial as well
55  }
```

led.h

```
1  #pragma once
2
3  #include <Arduino.h>
4
```

```
/// @brief A simple LED control class
class Led {
public:
  explicit Led(uint8_t pin);

  void on();
  boolean isOn();
  void off();
  void toggle();

private:
  uint8_t pin_;
};
```

led.cpp

```
#include <led/led.h>

Led::Led(uint8_t pin) : pin_(pin) {
  pinMode(pin_, OUTPUT);
}

void Led::on() {
  digitalWrite(pin_, HIGH);
}

boolean Led::isOn() {
  return digitalRead(pin_) == HIGH;
}

void Led::off() {
  digitalWrite(pin_, LOW);
}

void Led::toggle() {
  digitalWrite(pin_, !digitalRead(pin_));
}
```