Ministry of Education and Research of the Republic of Moldova
Technical University of Moldova
Department of Software and Automation Engineering

REPORT

Laboratory work No. 1.1

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

In this laboratory work, we focus on serial communication and user interaction with a microcontroller using the STDIO library.

In this application, the user can control an LED using text commands sent through a serial terminal. The commands are interpreted and executed, providing a response through the same serial interface.

2. Overview of the Hardware and Software Components Used

Hardware Components

1. Arduino Board:
    - We use the Arduino Mega 2560, which provides multiple UART ports and sufficient hardware resources for the application.
2. LED:
    - The LED is controlled through a digital pin.
3. USB Port and UART Converter:
    - Used for communication between the PC and the microcontroller.

Software Components

1. PlatformIO (in Visual Studio Code):
    - Used for writing and compiling C/C++ code for Arduino.
2. Arduino.h Library
3. Serial communication

3. System Architecture Explanation and Solution Justification

The system architecture is based on a simple input-output model, where a button and a serial interface are used to control an LED.

1. Modularity – The code is structured into multiple files for better organization.
2. Extensibility – Separating functionalities allows the quick addition of new components, such as sensors or other output devices.
3. Ease of Use – Serial commands provide a simple method of interaction with the system, facilitating testing and debugging.

This architecture was chosen for its simplicity and clarity, making it suitable for demonstrating the basic principles of microcontroller programming.

4. Case Study: LED Control in an Industrial Automation System

Context and Necessity

In the field of industrial automation, serial communication is essential for interaction between equipment, sensors, and operators. A system for monitoring and controlling industrial equipment needs to be fast, reliable, and easy to operate. An operator should be able to quickly check the status of a machine and intervene when necessary without being physically present.

In this context, a microcontroller-based system using a serial interface for sending commands represents an efficient and affordable solution. This case study examines the use of an Arduino Mega 2560 microcontroller for controlling signal LEDs on a production line.

Practical Implementation

A practical scenario for this project is its use in a smart lighting system, where the LED could be replaced with a relay controlling a real lamp.

- The user can turn the light on/off via a serial command.
- The system can be extended to include light sensors, allowing automatic lighting control based on ambient light levels.
- It can be connected to an IoT server to enable remote control via a mobile app.

Thus, this project provides an ideal starting point for implementing smart home solutions and demonstrates how embedded technologies can be used in a practical context.

Design

1. Architectural Sketch and Component Interconnection

The system architecture is based on the interaction between:

- An Arduino Mega 2560 microcontroller,
- A PC for sending serial commands,
- An LED connected to a digital pin of the microcontroller.

Component Description and Their Roles:

- Arduino Mega 2560: The main board that receives the commands.
- LED: The LED turns on/off based on the received command.
- Serial Monitor: Used for sending commands and viewing responses.

The sketch shows how the components interact to enable/disable the LED through serial interface. The communication takes place using the PC with the Platformio serial terminal.

The Arduino receives the commands and processes them through a software module. Based on the commands, it turns on/off the LED.

To complete the electrical circuit, the cathode of the LED is connected to the GND of the microcontroller.

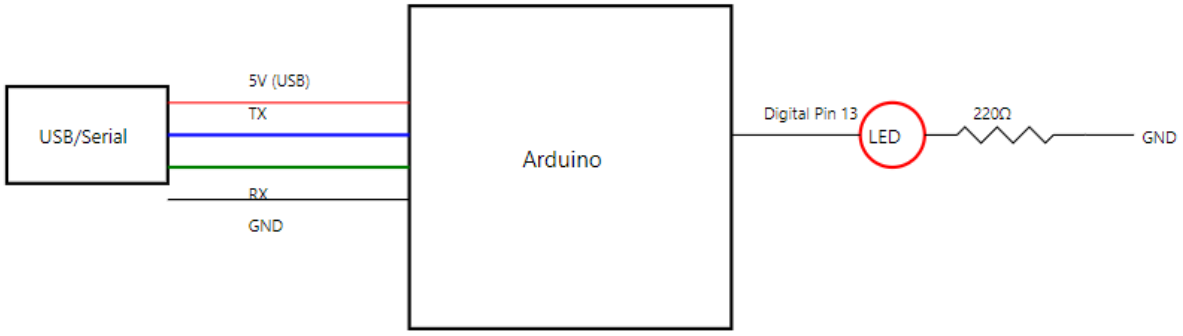

Figure 1 Electrical schematic

The electrical schematic shows how the PC can turn on/off the LED through commands.

When the user sends a valid command, the board changes the LED state. If the command sent is "led on," the microcontroller sets the pin to HIGH, allowing current to flow and turning on the LED. In the case of the "led off" command, the pin is set to LOW, turning off the LED.

2. Schematic diagrams

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

Flowchart – Serial Command Processing

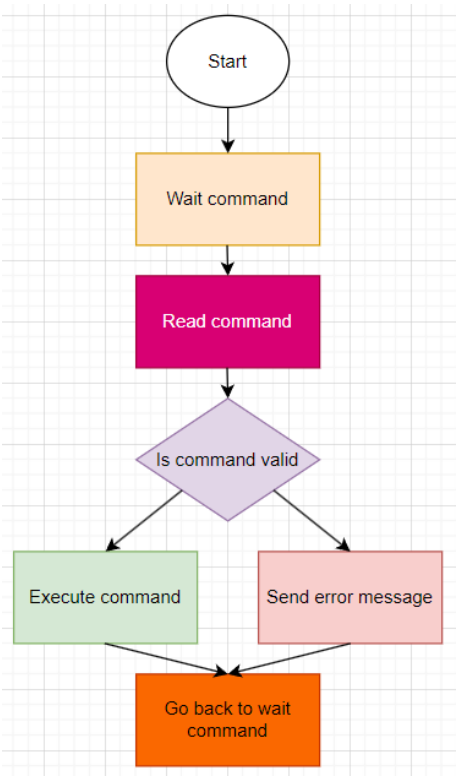

Figure 2 Flowchart

The FSM includes the following states:

1. Idle – Waits for commands from the user.
2. Processing – Processes the received command.
3. Led on – Turns on the LED and confirms via Serial.
4. Led off – Turns off the LED and confirms via Serial.
5. Error – Display error message for bad command

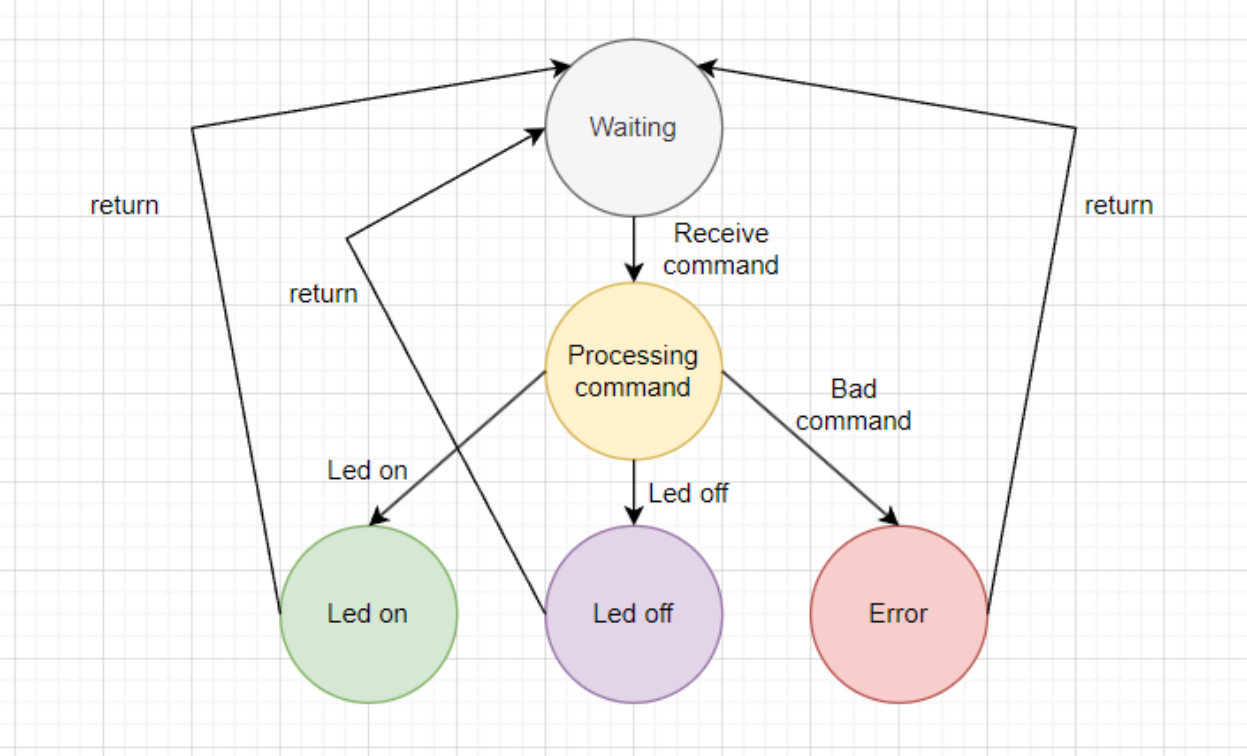

Figure 3 FSM diagram

3. Modular implementation

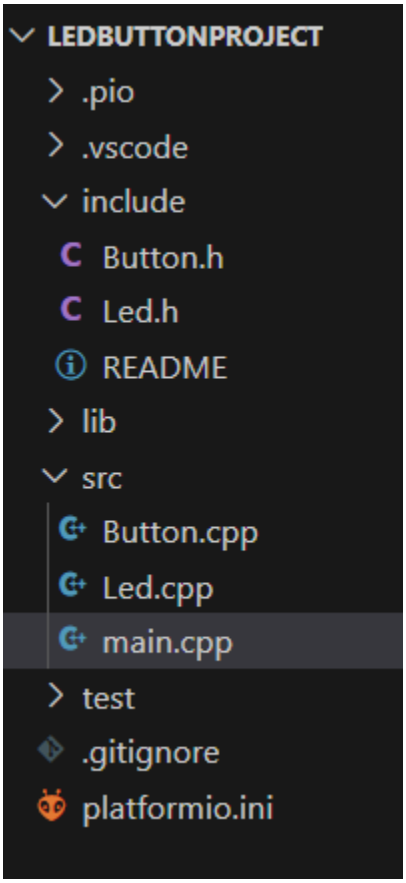

Figure 4 Project organization

This header file, named *Led.h*, defines the interface for controlling an LED. It declares 3 functions without providing their implementations:

1. toggle() – This function is used to change the state of the LED (State: ON or OFF)
2. turnOn() – This function is used to change the state of the LED to ON
3. turnOff() – This function is used to change the state of the LED to OFF

In the main.cpp 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

In the file Led.cpp we implement all methods in the Led.h:

1. toggle() – it changes the state using '!', and writes to the pin the state HIGH if its LOW and vice versa

2. turnOn() – sets the state to true and sets the pin to HIGH
3. turnOff() – sets the state to false and sets the pin to LOW

Conclusions

The project shows a reliable way to control a LED using text commands. The response time is minimal, and the code offers an easy way to debug and extend. Platformio helps with dependency management.

Limitations:

- The system currently lacks error handling for invalid inputs beyond basic message responses.
- The project is not optimized for power consumption, which could be an issue for battery-powered applications.

Possible improvements:

- Adding an OLED display or a status LED to indicate system states.
- Implementing remote control using a Bluetooth or WiFi module, allowing usage with mobile app.
- Optimizing power usage by implementing low-power modes when the system is idle.

For this report some of the text has been generated with the help of ChatGPT.

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
2.

```h
#ifndef LED_H
#define LED_H

#include <Arduino.h>

class Led {
public:
    Led(uint8_t pin);
    void toggle();
    void turnOn();
    void turnOff();

private:
    uint8_t pin;
    bool state;
};

#endif
```

```cpp
#include "Led.h"

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

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

void Led::turnOn() {
    state = true;
```

```
    digitalWrite(pin, HIGH);
}

void Led::turnOff() {
    state = false;
    digitalWrite(pin, LOW);
}
```

```
#include <Arduino.h>
#include "Button.h"
#include "Led.h"

#define BUTTON_PIN 2
#define LED_PIN 13

Button button(BUTTON_PIN);
Led led(LED_PIN);

int serial_putc(char c, FILE *stream) {
  Serial.write(c);
  return 0;
}

int serial_getc(FILE *stream) {
  while (!Serial.available()) {}
  return Serial.read();
}

FILE serial_stdout;
FILE serial_stdin;

void setup() {
    Serial.begin(115200);
    led.turnOn();

    fdev_setup_stream(&serial_stdout, serial_putc, NULL, _FDEV_SETUP_WRITE);

    stdout = &serial_stdout;

    printf("System Ready! Send 'led on' or 'led off' to control the LED.\n");
}

void loop() {
  // int someDelay = random(10, 500);
```

```
  // led.toggle();
  // delay(someDelay);
  //   if (button.isPressed()) {
  //       led.toggle();
  //       delay(200); // Debounce
  //   }
  char command[10];

    if (Serial.available() > 0) {
        printf("Enter command: ");
        Serial.readBytesUntil('\n', command, sizeof(command) - 1);
        command[strcspn(command, "\r\n")] = 0;

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