Ministry of Education and Research of the Republic of Moldova

Technical University of Moldova

Department of Software and Automation Engineering

REPORT

Laboratory work No. 1.2

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 project, we used a microcontroller (MCU) to control various tasks and peripheral devices such as an LCD, a keypad, and LEDs. The focus was on interaction with the user through input (keypad) and output (LCD and LEDs) devices. The main goal was to design a modular system with several distinct tasks, where tasks interact through global variables or signals (provider/consumer model).

Hardware Components

Microcontroller: An Arduino MEGA 2560.

LEDs: Used to indicate the status of the system.

Keypad 4x4: Used to input a password to grant or deny access.

LCD: Displayed messages to the user, such as prompts and results.

Breadboard and Jumper Wires: For the physical connections.

USB Power Supply: Power source for the MCU.

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

2. 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.

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

This type of system could be applied in various security or automation systems where a password needs to be validated before access is granted, and feedback is provided through LEDs and an LCD display.

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.
- A Keypad
- An LCD screen

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.
- LCD screen: Used to view access granted/denied, password input and results
- Keypad: for password input

The sketch shows how the components interact to enter a password and turn on/off the LED with serial commands.

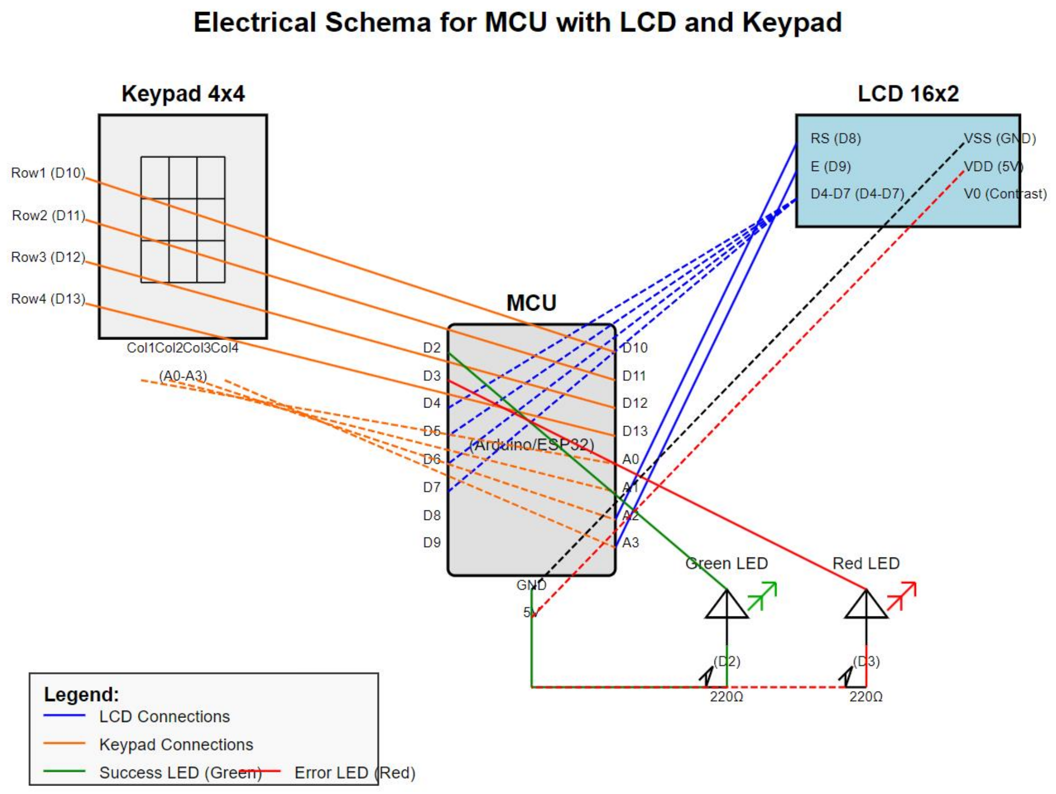
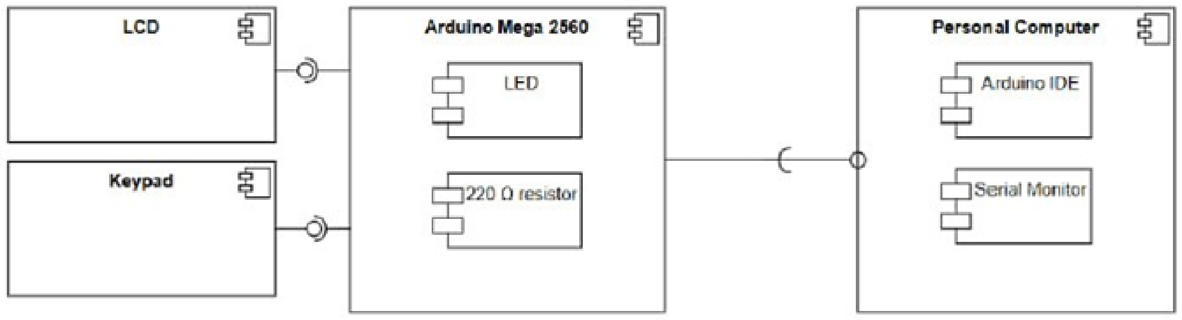


Figure 1 Electrical schematic

The electrical schematic shows how the PC can turn on/off the LED through commands after entering the correct password using the Keypad. The result of the password input is displayed on the LCD screen. After a successful password, we can use the serial command to turn on/off the LED.



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

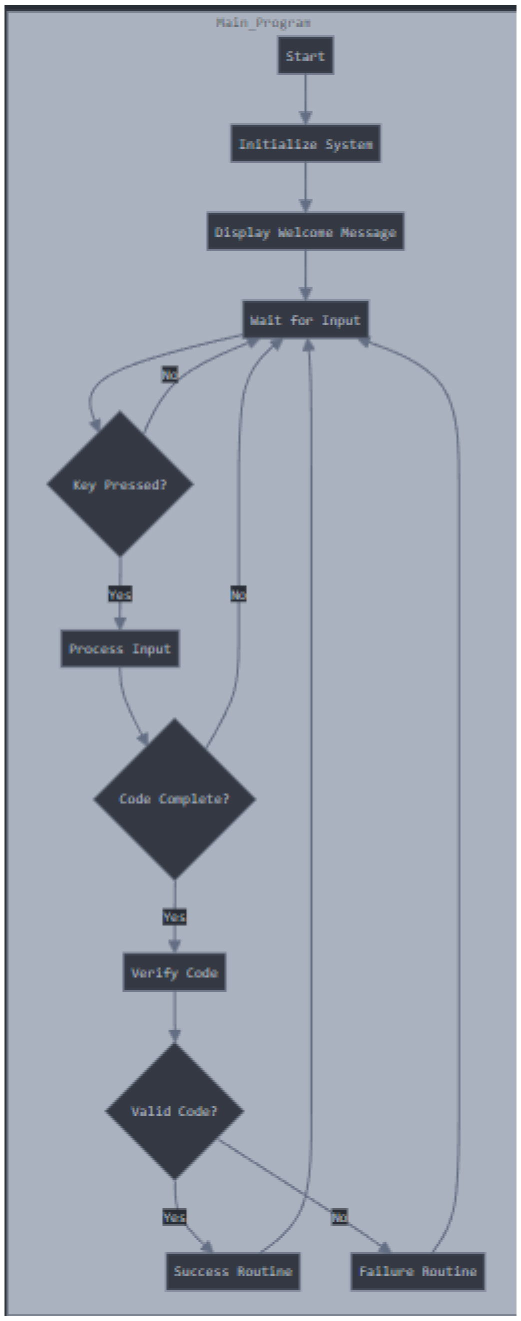


Figure 2 Flowchart-code verification

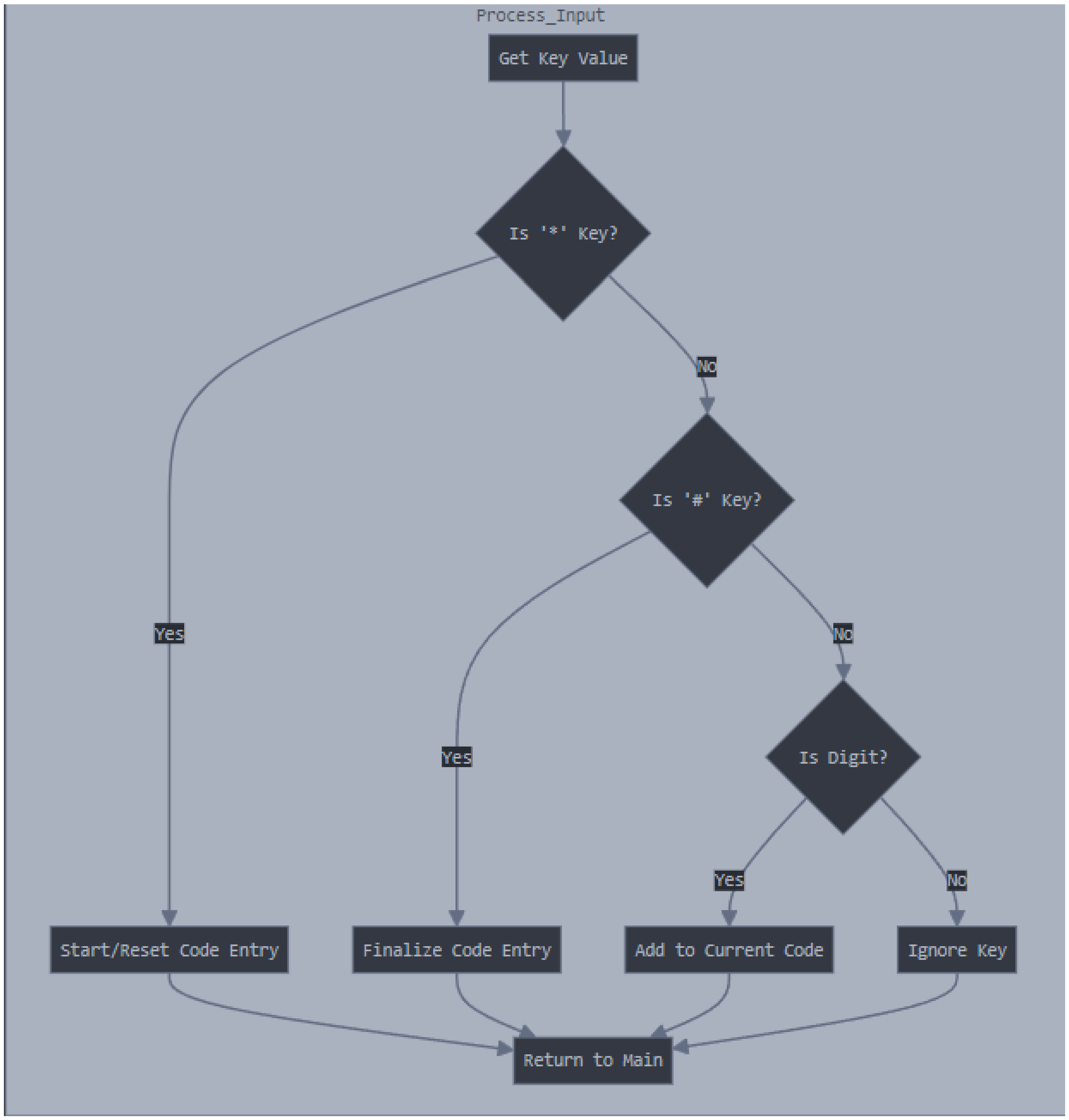


Figure 3 Code verification flowchart

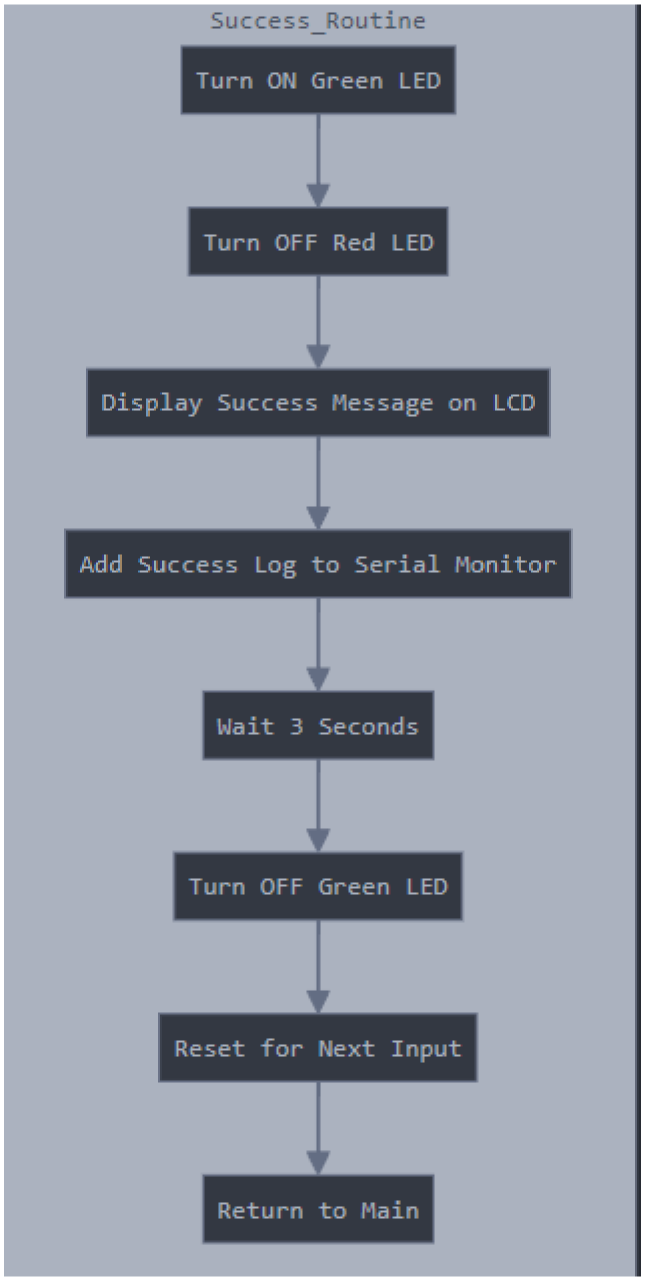


Figure 4 Success routine flowchart

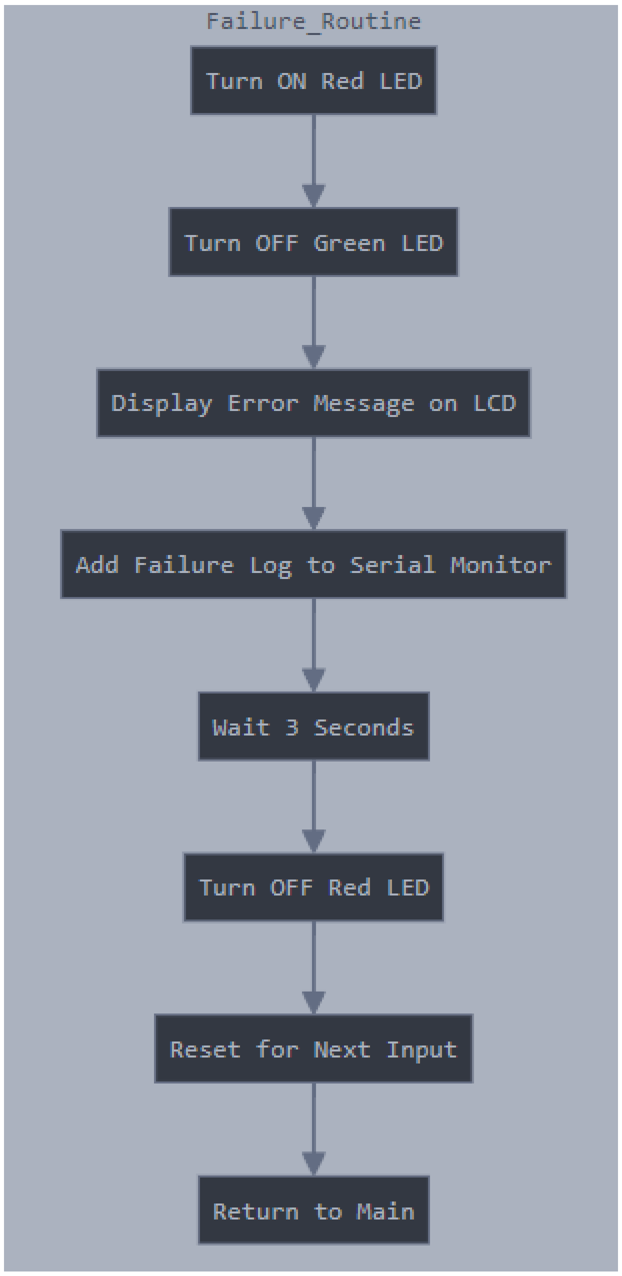


Figure 5 Failure routine

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

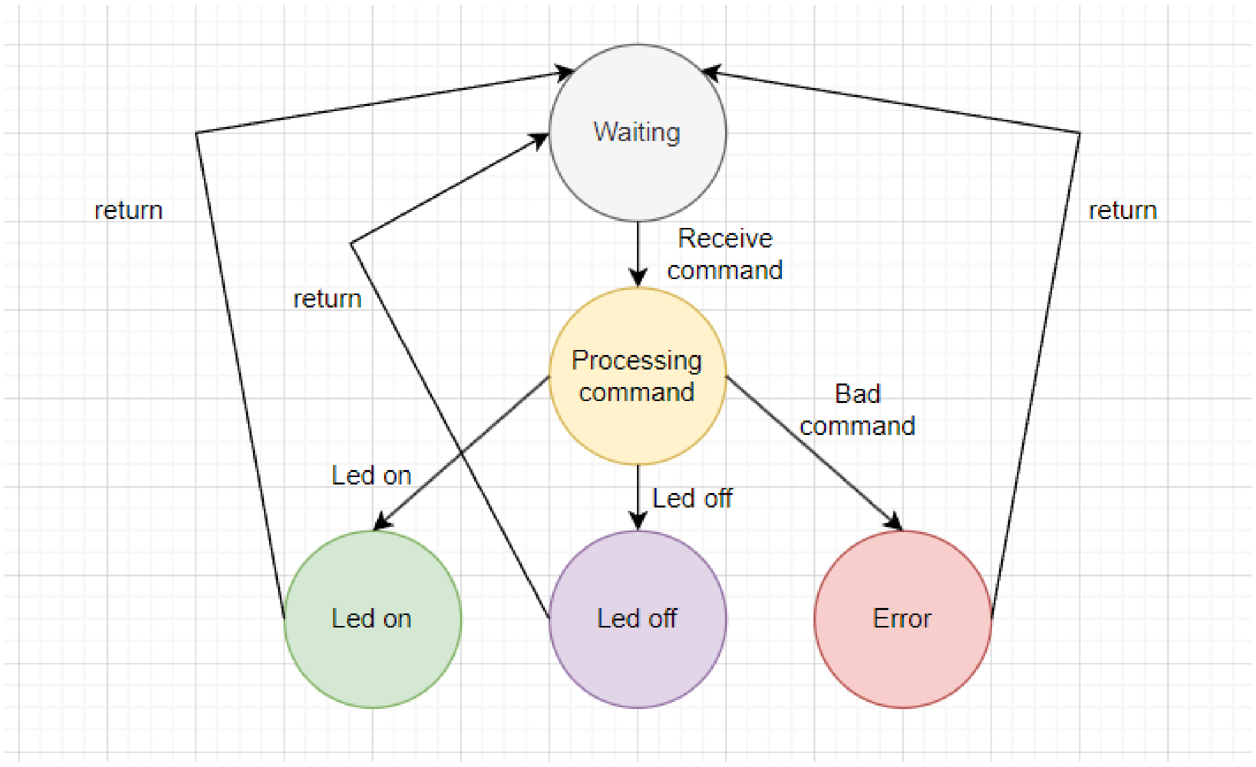
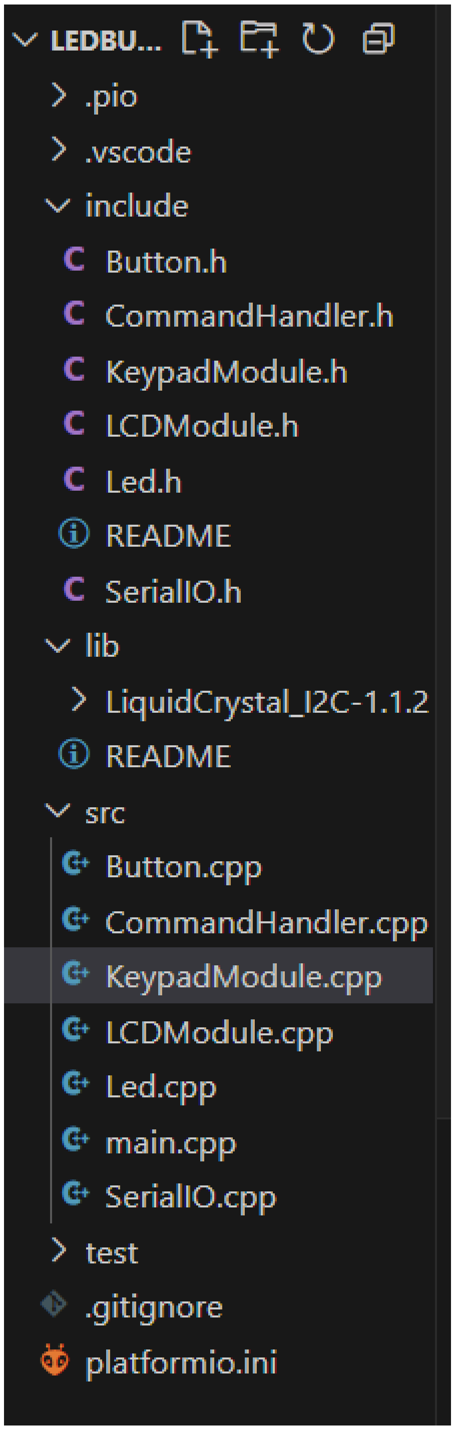


Figure 6 FSM diagram

3. Modular implementation



```
LEDBU...
  .pio
  .vscode
  include
    Button.h
    CommandHandler.h
    KeypadModule.h
    LCDModule.h
    Led.h
    README
    SerialIO.h
  lib
    LiquidCrystal_I2C-1.1.2
    README
  src
    Button.cpp
    CommandHandler.cpp
    KeypadModule.cpp
    LCDModule.cpp
    Led.cpp
    main.cpp
    SerialIO.cpp
  test
  .gitignore
  platformio.ini
```

Figure7 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

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

1. handleCommand() – it displays on serial monitor message of command in case on/off
2. processSerialInput() – checks if command is on/off

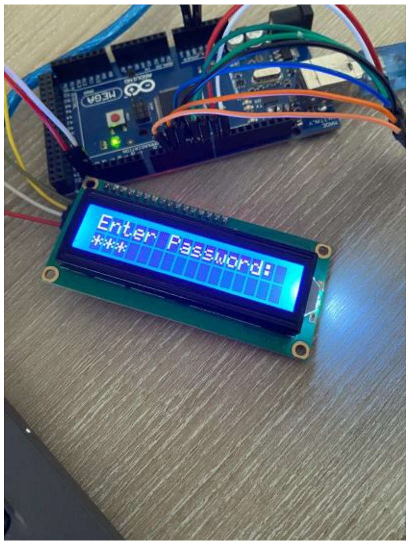
In the file KeypadModule.cpp we implement all methods in the KeypadModule.h:

1. handleKeyPress() – process password
2. resetPassword() – go back to input password screen

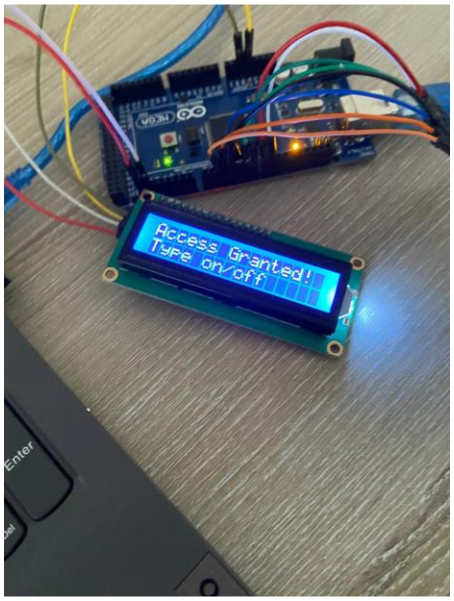
In the file LCDModule.cpp we implement all methods in the LCDModule.h

Results

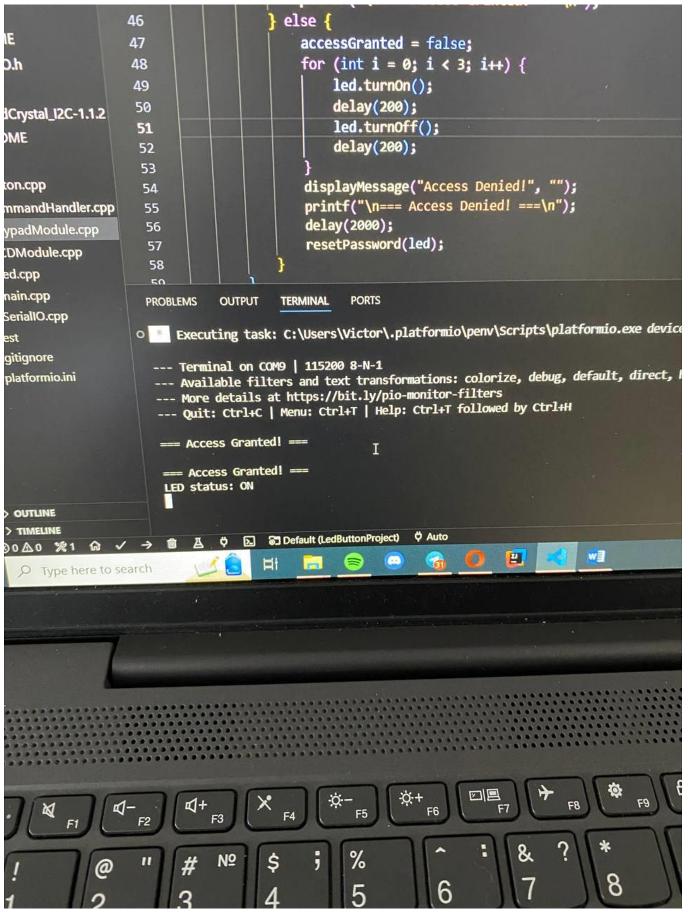
Enter password:



Access granted:

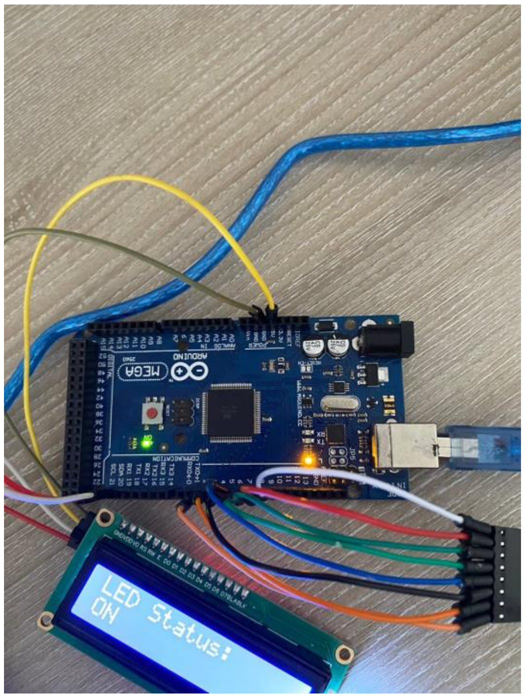


Turning on led:

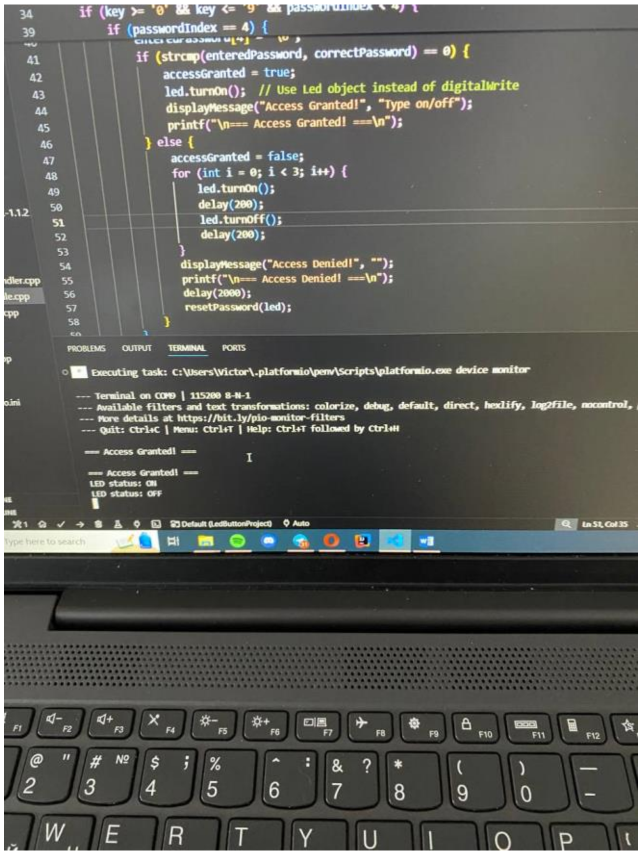


```
} else {
    accessGranted = false;
    for (int i = 0; i < 3; i++) {
        led.turnOn();
        delay(200);
        led.turnOff();
        delay(200);
    }
    displayMessage("Access Denied!", "");
    printf("\n=== Access Denied! ===\n");
    delay(2000);
    resetPassword(led);
}
```

```
Executing task: C:\Users\Victor\.platformio\penv\Scripts\platformio.exe device

--- Terminal on COM9 | 115200 8-N-1
--- Available filters and text transformations: colorize, debug, default, direct,
--- More details at https://bit.ly/pio-monitor-filters
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H

=== Access Granted! ===

=== Access Granted! ===
LED status: ON
```

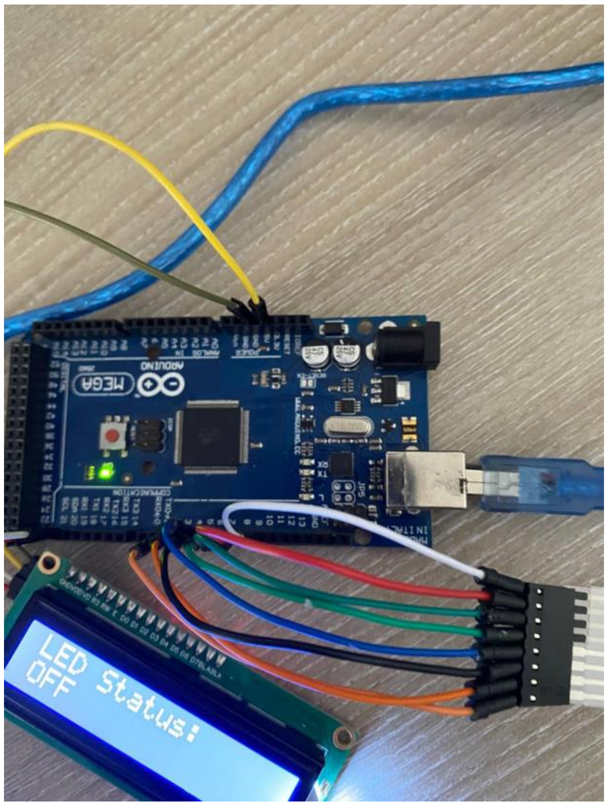


Turning off led:



```
    if (key >= '0' && key <= '9' && passwordIndex < 4) {
      if (passwordIndex == 4) {
        if (strcmp(enteredPassword, correctPassword) == 0) {
          accessGranted = true;
          led.turnOn();  // Use Led object instead of digitalWrite
          displayMessage("Access Granted!", "Type on/off");
          printf("\n=== Access Granted! ===\n");
        } else {
          accessGranted = false;
          for (int i = 0; i < 3; i++) {
            led.turnOn();
            delay(200);
            led.turnOff();
            delay(200);
          }
          displayMessage("Access Denied!", "");
          printf("\n=== Access Denied! ===\n");
          delay(2000);
          resetPassword(led);
        }
```

```
Executing task: C:\Users\Victor\.platformio\penv\Scripts\platformio.exe device monitor

--- Terminal on COM9 | 115200 8-N-1
--- Available filters and text transformations: colorize, debug, default, direct, hexlify, log2file, nocontrol,
--- More details at https://bit.ly/pio-monitor-filters
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H

=== Access Granted! ===

=== Access Granted! ===
LED status: ON
LED status: OFF
```



Conclusions

System Performance Analysis, Identification of Potential Limitations, and Improvement Proposals:

The system performs as expected, with reliable interactions between the user and the hardware components (keypad, LED, and LCD). The tasks run sequentially, with clear feedback provided via the LED and LCD, based on user input. The modular design ensures that each task is well-defined and operates independently, promoting code reusability and maintainability.

However, the system could be enhanced in a few areas:

Error Handling: The current implementation does not handle unexpected inputs or errors very well. Adding a timeout feature or better feedback when the user makes mistakes (e.g., entering too many wrong passwords) could improve the user experience.

Scalability: The system is designed to handle a single password check and a simple LED toggle. It can be expanded to manage more complex tasks, like multiple users or a more sophisticated security system (e.g., fingerprint or RFID authentication).

Power Efficiency: In the current configuration, the system continuously checks for key presses and performs tasks, which could result in higher power consumption. Adding power-saving techniques, such as putting the system into sleep mode when idle, could be beneficial, especially for battery-powered applications.

Conclusions from the Laboratory Results:

The results obtained from the laboratory task demonstrate the system's functionality as expected. The sequential tasks of reading user input, checking the password, and providing feedback through an LED and LCD were successfully implemented. The integration of modular code made it easier to troubleshoot, and the system reliably responded to user input. The use of a keypad and LCD for user interaction was effective, providing a simple yet functional interface for the application.

Impact of the Technology Used in Real-World Applications:

The technologies employed in this project, particularly the use of microcontrollers (MCUs), keypads, LCDs, and LEDs, are widely used in real-world applications such as access control systems, home automation, and security systems. The ability to control peripherals based on user input makes this system highly adaptable for various scenarios where authentication is required. The principles of modular design, sequential task management, and user interaction are key components of modern embedded systems, and this approach could be expanded to larger, more complex systems in various industries.

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 şi Programarea Microcontrolerelor
   - Principiile comunicației seriale și utilizarea interfeței UART

Appendix

1. **GitHub**: https://github.com/Kipitokisk/SI_Lab
2.

```
#include "CommandHandler.h"
#include "LCDModule.h"
#include <stdio.h>
#include <string.h>

char command[10];

void processSerialInput() {
    if (scanf("%s", command) == 1) {
        for (char *p = command; *p; p++) {
            *p = tolower(*p);
        }

        if (strcmp(command, "on") == 0 || strcmp(command, "off") == 0) {
            handleCommand(command);
        } else {
            printf("Invalid command! Use 'on' or 'off'\n");
        }
    }
}

```

```
void handleCommand(const char* cmd) {
    if (strcmp(cmd, "on") == 0) {
        digitalWrite(LED_BUILTIN, HIGH);
        displayMessage("LED Status:", "ON");
        printf("LED status: ON\n");
    } else if (strcmp(cmd, "off") == 0) {
        digitalWrite(LED_BUILTIN, LOW);
        displayMessage("LED Status:", "OFF");
        printf("LED status: OFF\n");
    }
}

```

```
#include "KeypadModule.h"
#include "LCDModule.h"
#include "SerialIO.h"
#include <stdio.h>
#include <string.h>
#include "Led.h"

const byte ROWS = 4;
const byte COLS = 4;
char keys[ROWS][COLS] = {
    {'1', '2', '3', 'A'},
    {'4', '5', '6', 'B'},
    {'7', '8', '9', 'C'},
    {'*', '0', '#', 'D'}
};
byte rowPins[ROWS] = {9, 8, 7, 6};
byte colPins[COLS] = {5, 4, 3, 2};

Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);
const char correctPassword[] = "1234";
char enteredPassword[5] = {0};
int passwordIndex = 0;
bool accessGranted = false;

void keypadInit() {
    memset(enteredPassword, 0, sizeof(enteredPassword));
}

char getKeyInput() {
    return keypad.getKey();
}
```

```
void handleKeyPress(char key, Led &led) {
    if (key >= '0' && key <= '9' && passwordIndex < 4) {
        enteredPassword[passwordIndex] = key;
        lcdShowPasswordChar(passwordIndex);
        passwordIndex++;

        if (passwordIndex == 4) {
            enteredPassword[4] = '\0';
            if (strcmp(enteredPassword, correctPassword) == 0) {
                accessGranted = true;
                led.turnOn();  // Use Led object instead of digitalWrite
                displayMessage("Access Granted!", "Type on/off");
                printf("\n=== Access Granted! ===\n");
            } else {
                accessGranted = false;
                for (int i = 0; i < 3; i++) {
                    led.turnOn();
                    delay(200);
                    led.turnOff();
                    delay(200);
                }
                displayMessage("Access Denied!", "");
                printf("\n=== Access Denied! ===\n");
                delay(2000);
                resetPassword(led);
            }
        }
    } else if (key == '*') {
        resetPassword(led);
    }
}

void resetPassword(Led &led) {
    passwordIndex = 0;
    memset(enteredPassword, 0, sizeof(enteredPassword));
    accessGranted = false;
    led.turnOff();
    displayMessage("Enter Password:", "");
}



#include "LCDModule.h"
```

```
LiquidCrystal_I2C lcd(0x27, 16, 2);

void lcdInit() {
    lcd.init();
    lcd.backlight();
    lcd.clear();
    lcd.print("Enter Password:");
    lcd.setCursor(0, 1);
}

void lcdShowMessage(const char* message) {
    lcd.clear();
    lcd.print(message);
}

void lcdShowPasswordChar(int position) {
    lcd.setCursor(position, 1);
    lcd.print('*');
}

void displayMessage(const char* line1, const char* line2) {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print(line1);
    lcd.setCursor(0, 1);
    lcd.print(line2);
}



#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 "SerialIO.h"
#include "KeypadModule.h"
#include "LCDModule.h"
#include "CommandHandler.h"
#include "Led.h"

#define LED_PIN 13

Led led(LED_PIN);

void setup() {
    serialInit();
    lcdInit();
    keypadInit();
    pinMode(LED_PIN, OUTPUT);
    digitalWrite(LED_PIN, LOW);
}

void loop() {
    char key = getKeyInput();
    if (key) {
        handleKeyPress(key, led);
    }

    if (Serial.available() > 0) {
        processSerialInput();
    }
}
```