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

Elaborated: Berco Andrei, 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

With the increasing demand for embedded systems in various applications, microcontrollers (MCUs) have become essential for efficient human-machine interaction. In modern embedded systems, input devices like keypads and output displays such as LCDs are widely used for interfacing with users. Implementing secure access mechanisms, such as password-protected systems, is crucial in fields like security, automation, and control systems.

This laboratory work aims to familiarize students with the principles of peripheral interaction using the STDIO library to facilitate communication between a keypad and an LCD. The project involves developing an MCU-based system that verifies a user-inputted code and provides visual feedback using LEDs and an LCD.

2. Overview of the Hardware and Software Components Used

Hardware Components

- **Microcontroller** (Arduino Mega 2560) – Serves as the processing unit for handling input and output operations.
- **Keypad 4x4** – A 16-button matrix used for user input.
- **LCD 2x16** – A character display used to provide real-time feedback.
- **LEDs (Red and Green)** – Indicate success or failure of the authentication process.
- **220 Ω Resistor** – Used for current-limiting to protect components.
- **Breadboard and Jumper Wires** – For prototyping and connecting components.
- **Power Supply (USB or Battery)** – Provides power to the MCU.

Software Components

- **Arduino IDE / ESP-IDF** – Development environments for programming the microcontroller.
- **STDIO Library** – Used for serial communication and user interaction.
- **Keypad Library** – Facilitates scanning and decoding of keypad inputs.
- **LiquidCrystal Library** – Enables interaction with the LCD display.

3. System Architecture Explanation and Solution Justification

The system follows a modular approach, where each peripheral is controlled by a dedicated function. The microcontroller continuously listens for user input via the keypad. When a user enters a code, the system compares it with a predefined password. If the input matches, a success message is displayed on the LCD, and a green LED turns on. Otherwise, an error message is displayed, and a red LED turns on. The solution is efficient, cost-effective, and applicable in real-world scenarios such as access control systems and embedded security applications.

Why this solution?

- **Reliability** – The system ensures accurate user input detection and feedback.
- **Scalability** – It can be extended to include more features, such as wireless communication.
- **Ease of Implementation** – Uses widely available components and libraries.
- **Low Power Consumption** – Suitable for battery-powered applications.

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

Context and Necessity

Context and Necessity In security-sensitive environments such as office buildings, laboratories, or industrial facilities, access control systems play a crucial role in restricting unauthorized entry.

A microcontroller-based solution using a keypad and LCD provides a cost-effective and reliable method for authentication.

This case study examines the implementation of an Arduino Uno-based access control system that verifies user input through a password mechanism and provides immediate feedback via an LCD and LEDs.

Practical Implementation A security door system is equipped with an Arduino Uno microcontroller, a 4x4 keypad for user authentication, and a 2x16 LCD for displaying status messages. LEDs indicate authentication success or failure.

Each LED has a well-defined role in the system:

- **Green LED** – Indicates successful authentication, unlocking the door.
- **Red LED** – Signals authentication failure, keeping the door locked.

Examples of Commands and Workflow:

1. A user enters a 4-digit passcode on the keypad.
2. The microcontroller compares the entered passcode with the stored password.
3. If the code is correct:
    - The LCD displays "Access Granted."
    - The green LED lights up, and the door unlocks.
4. If the code is incorrect:
    - The LCD displays "Access Denied."
    - The red LED lights up, preventing entry.

Extending the Case Study This solution can be adapted for other security and automation applications, such as:

- **Multi-Level Access Control:** Assigning different passwords for different security levels.
- **Remote Authentication:** Integrating the system with a Wi-Fi or Bluetooth module to allow remote control.
- **Data Logging:** Storing incorrect login attempts for security audits.

Benefits and Impact Using a microcontroller-based access control system offers multiple advantages:

- **Enhanced Security:** Prevents unauthorized access using password authentication.
- **User-Friendly Interface:** The LCD provides clear real-time feedback.
- **Cost-Effective:** Uses inexpensive and readily available components.
- **Scalability:** The system can be expanded with additional security features like biometric authentication or RFID-based access control.
-

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.
- An LCD display
- A keypad

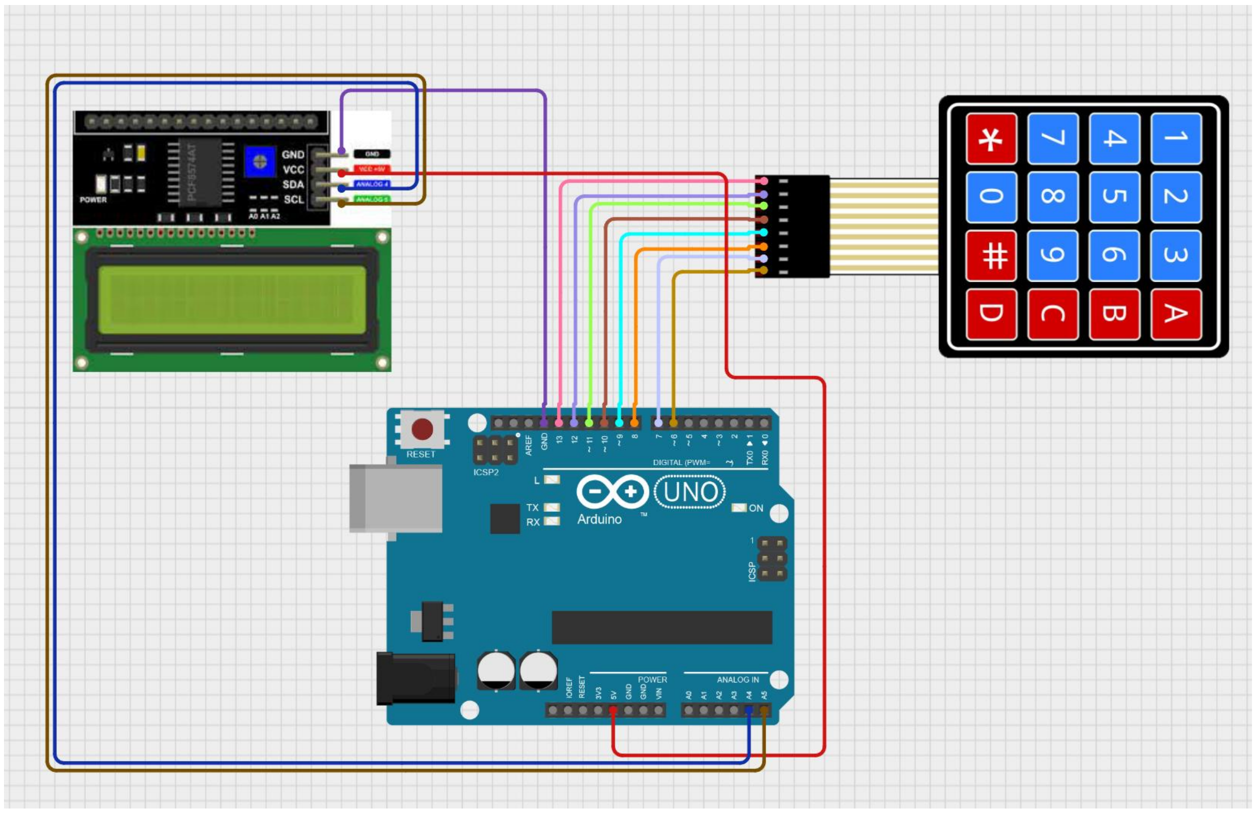


Figure 2.1 Electrical schematic

Arduino Board (Arduino 2560)

- **Central Component**: The Arduino 2560 is at the center of the diagram, serving as the main controller.
- **Connections**:
- **Power**: Connected to both 5V and GND.
- **Digital Pins**: Used for interfacing with the LCD display and keypad.

LCD Display

- **Position**: Located on the left side of the diagram.
- **Pins and Connections**:
- **Pin 1 (VSS)**: Connected to GND.
- **Pin 2 (VDD)**: Connected to 5V.
- **Pin 3 (VO)**: Connected through a potentiometer for contrast adjustment.
- **Pin 4 (RS)**: Connected to digital pin 12 on the Arduino.

- **Pin 5 (RW)**: Connected to GND.
- **Pin 6 (E)**: Connected to digital pin 11 on the Arduino.
- **Pins 11-14 (D4-D7)**: Connected to digital pins 5, 4, 3, and 2 on the Arduino.

Keypad

- **Position**: Located on the right side of the diagram.
- **Keys**: Includes numbers '0' to '9', '*', '#', 'A', 'B', 'C', and 'D'.
- **Pins and Connections**:
- **Columns (C1-C4)**: Connected to analog pins A0, A1, A2, and A3 on the Arduino.
- **Rows (R1-R4)**: Connected to analog pins A4, A5, and digital pins 8 and 9 on the Arduino.

2. Scheme bloc and algorithm

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

Flowchart – Serial Command Processing
(A Flowchart diagram illustrating the cycle: command reception → processing → execution → user feedback)

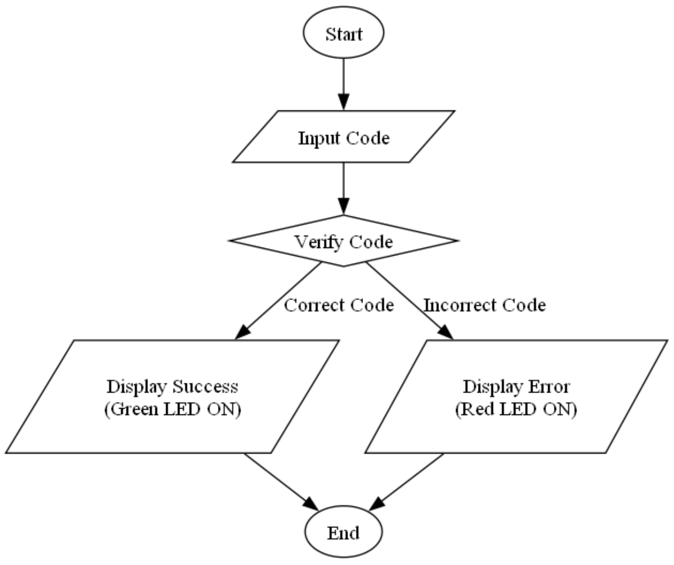


Figure 2.2 Flowchart

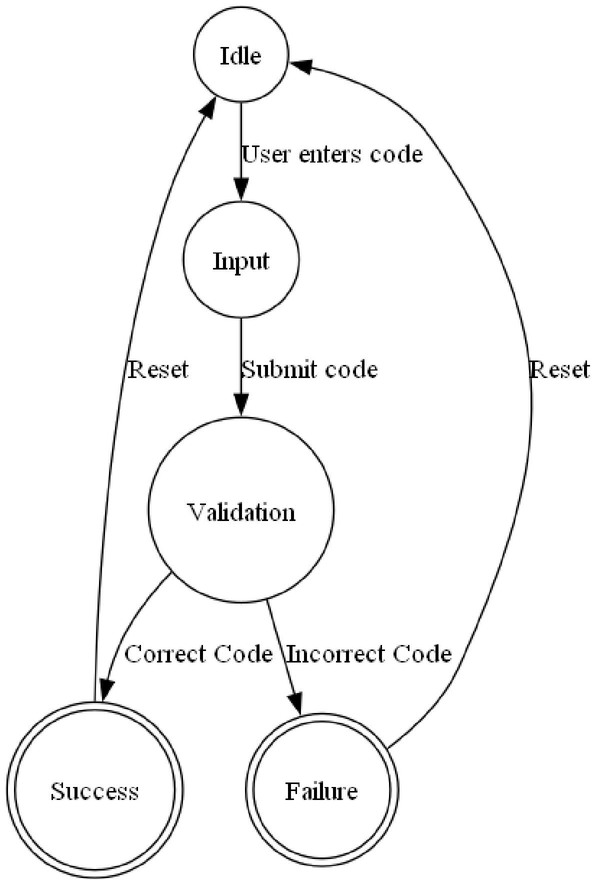
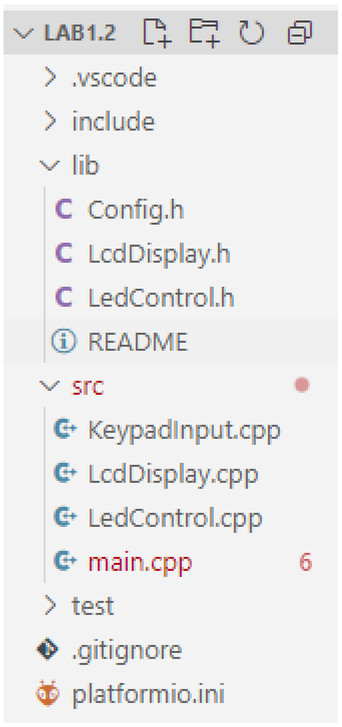


Figure 2.3 FSM diagram

3. Modular implementation

For better project organization, a modular architecture was used, dividing functionalities into separate files.



```
LAB1.2
  .vscode
  include
  lib
    Config.h
    LcdDisplay.h
    LedControl.h
    README
  src
    KeypadInput.cpp
    LcdDisplay.cpp
    LedControl.cpp
    main.cpp          6
  test
  .gitignore
  platformio.ini
```

Figure 3.1 Project organization

The project is well-structured, separating functionality into different modules using a layered approach. The organization follows best practices for embedded systems and modular programming.

File Organization:

1. lib/
   - Config.h – Stores hardware configuration settings, including pin definitions for peripherals.
   - LcdDisplay.h – Defines an interface for handling an LCD display.
   - LedControl.h – Provides an interface for controlling LEDs.
2. src/
   - KeypadInput.cpp – Manages keypad input handling, allowing users to enter a code.
   - LcdDisplay.cpp – Handles displaying messages on the LCD.
   - LedControl.cpp – Implements functions to turn LEDs on/off.
   - main.cpp – The main application logic that integrates all components.

Code Functionality Overview:

1. Configuration Module (Config.h)
   - Defines constants and pin assignments for keypad, LCD, and LED components.
   - Provides a predefined valid code for authentication.
2. Keypad Input Handling (KeypadInput.h/cpp)

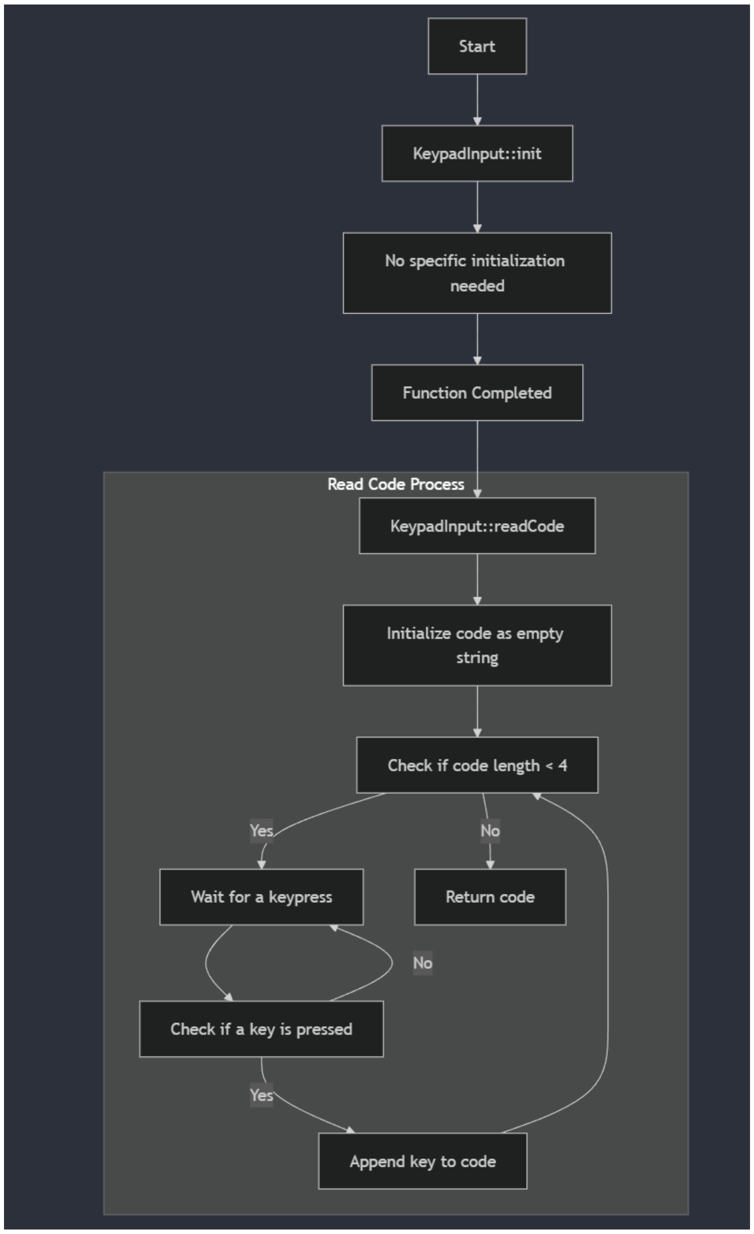


Figure 3.2: Flowchart for KeypadInput.cpp

- Implements a system to capture a 4-character input sequence.
- Waits for user input from a keypad and returns the entered code.

3. LCD Display (LcdDisplay.h/cpp)

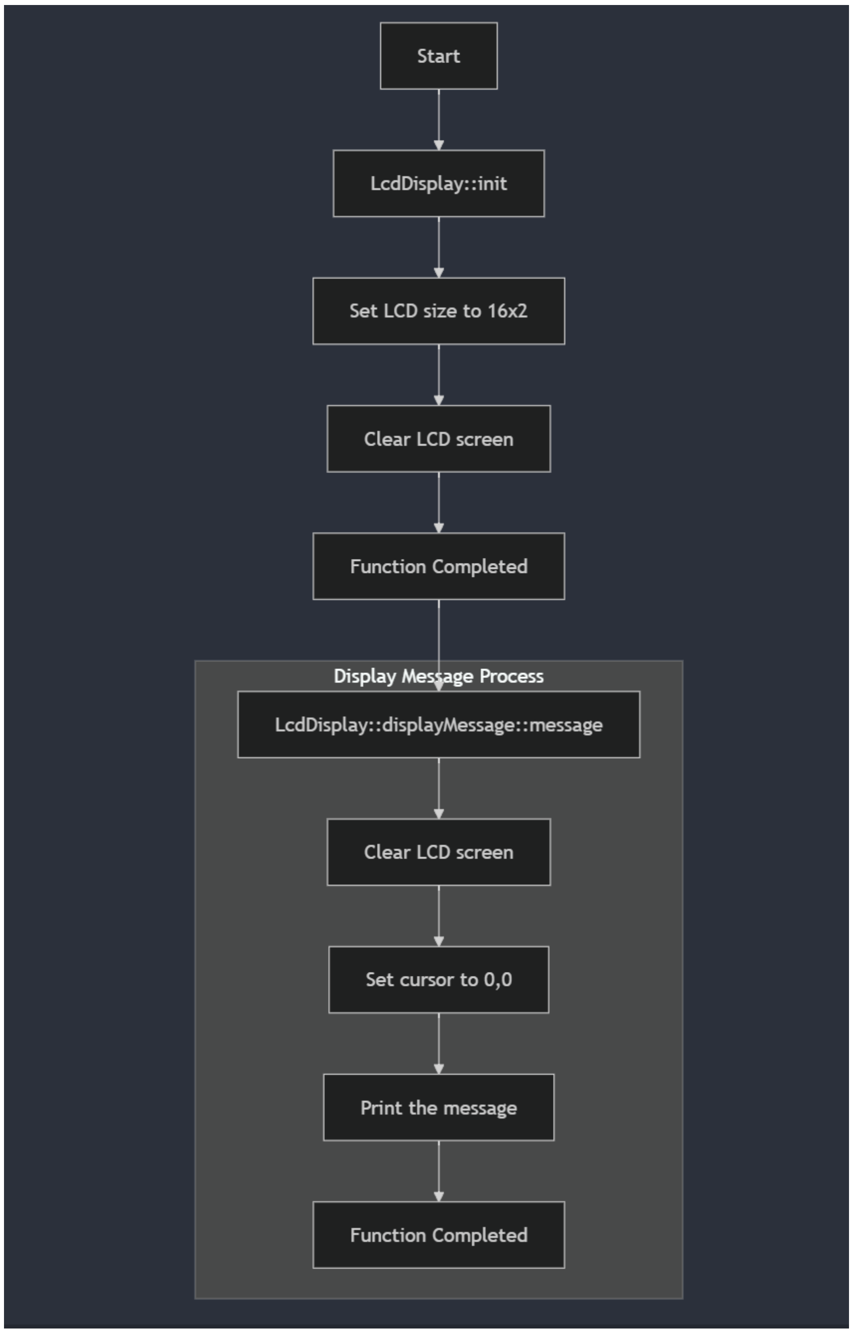


Figure 3.3: Flowchart for LcdcDisplay.cpp

- Displays messages based on system state.

- Updates messages dynamically depending on input correctness.

4. LED Control (LedControl.h/cpp)

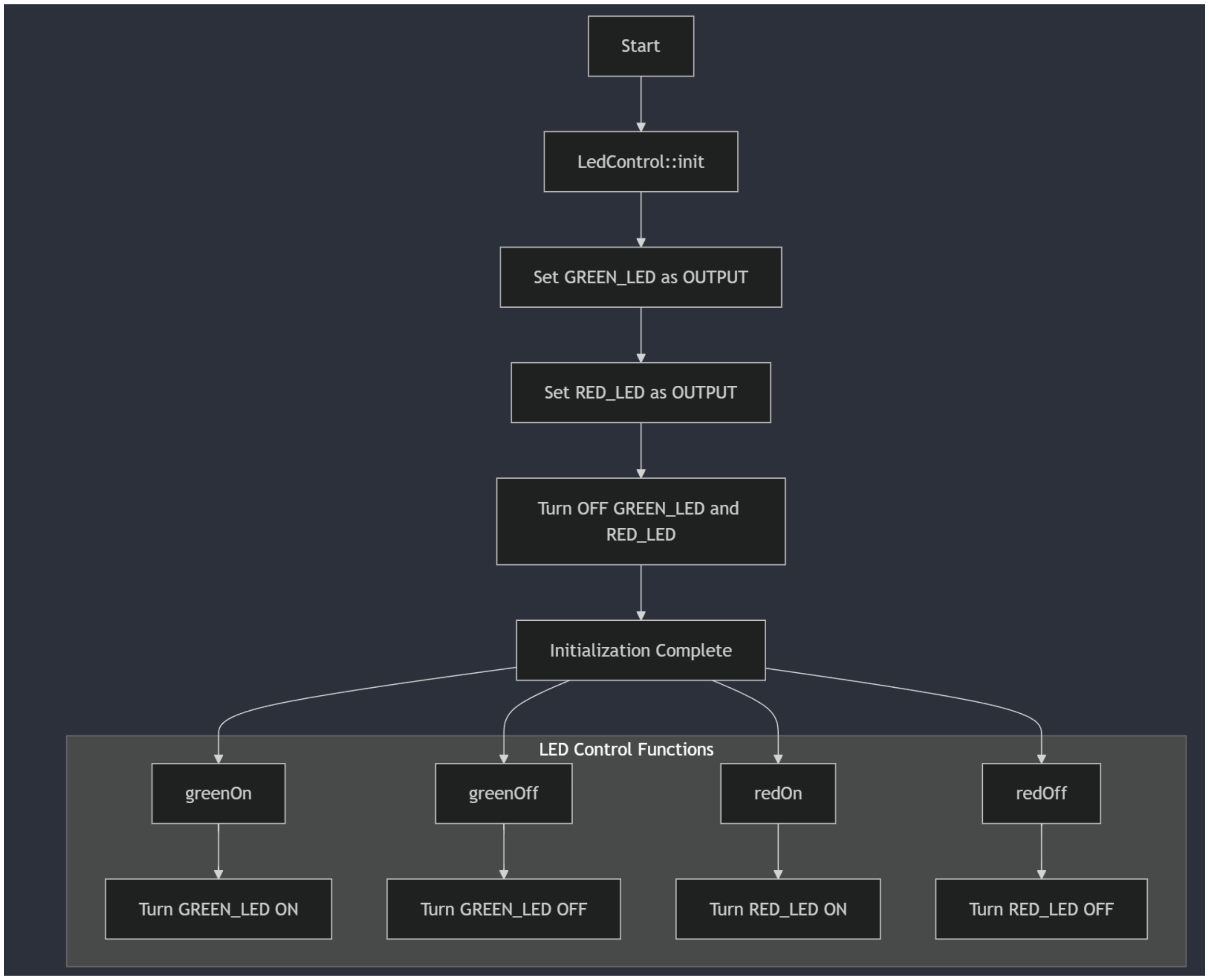


Figure 3.4: Flowchart for LedControl.cpp

- Controls two LEDs: one indicating correct input and another for incorrect input.
- Provides initialization and state-changing functions.

5. Main Application (main.cpp)

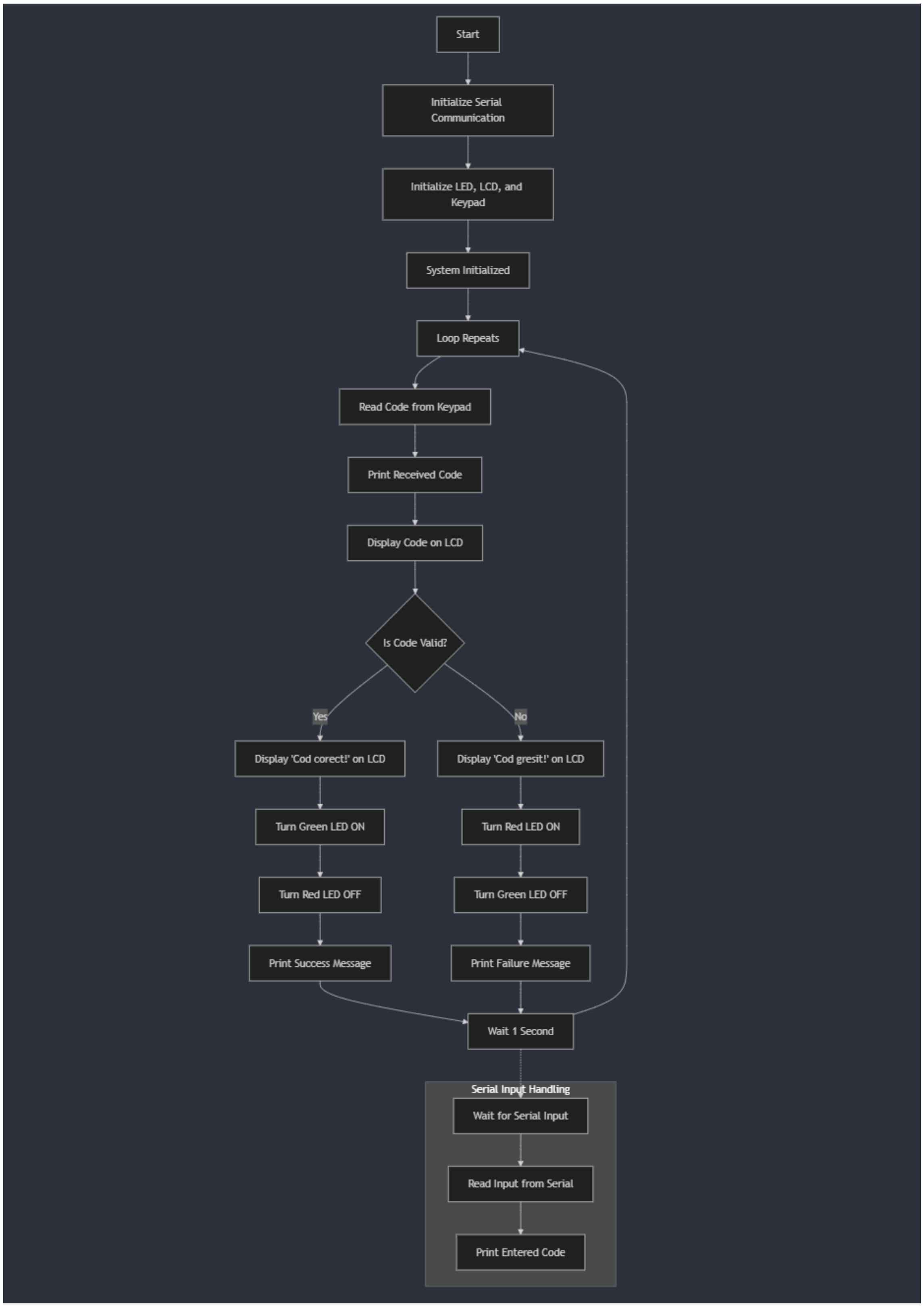


Figure 3.5: Flowchart for main.cpp

- Initializes the system components.
- Continuously reads user input and checks against a predefined code.
- Displays appropriate messages and toggles LED states based on input correctness.
- Includes serial input handling for debugging or alternative code entry.

Results

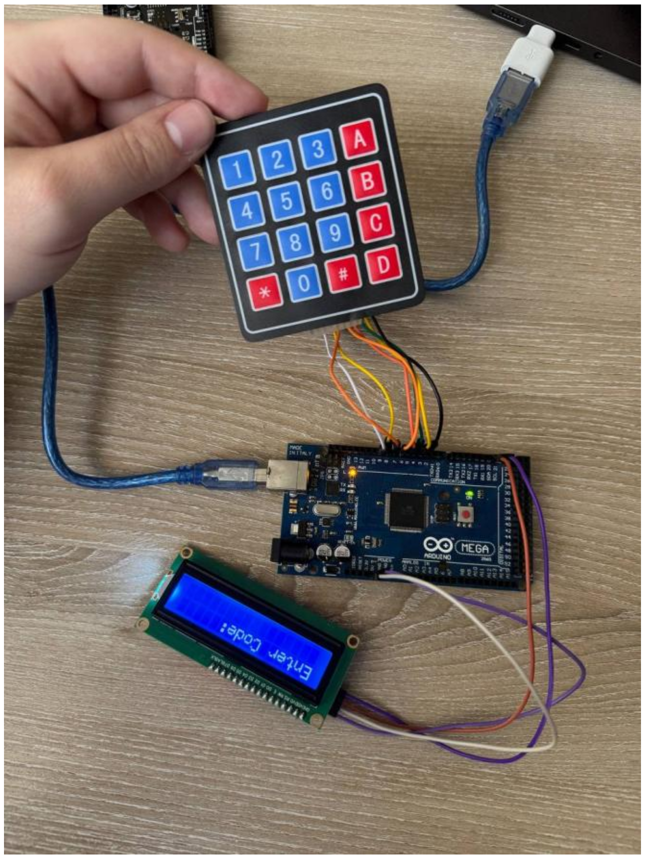


Figure 4.1 Start up with „Enter code" mesasge on display

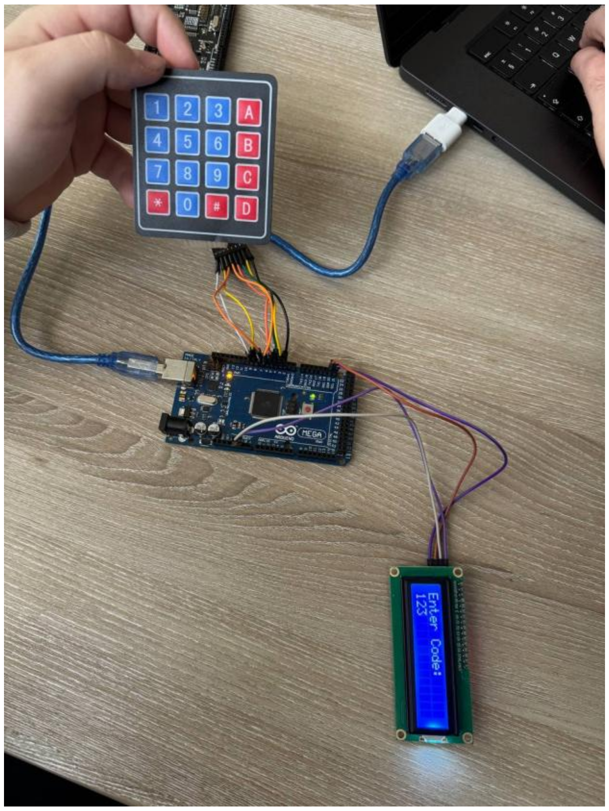


Figure 4.2 Entered wrong code (123) using keypad

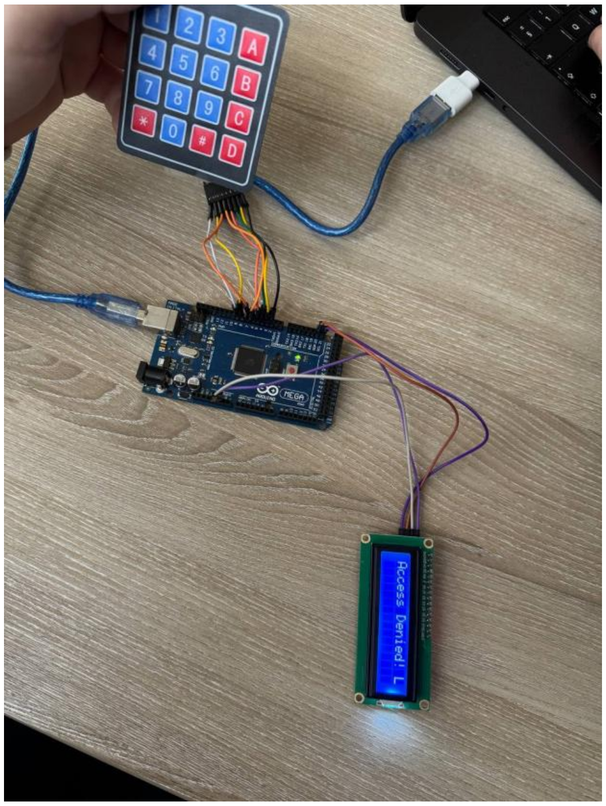


Figure 4.3 „Access denied" message for the incorrect code and turned off led

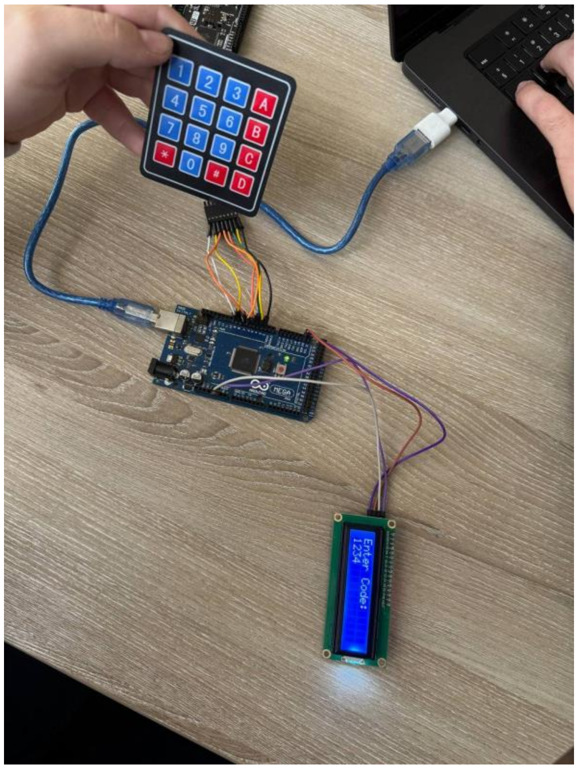


Figure 4.4 Entered correct code (1234) using keypad

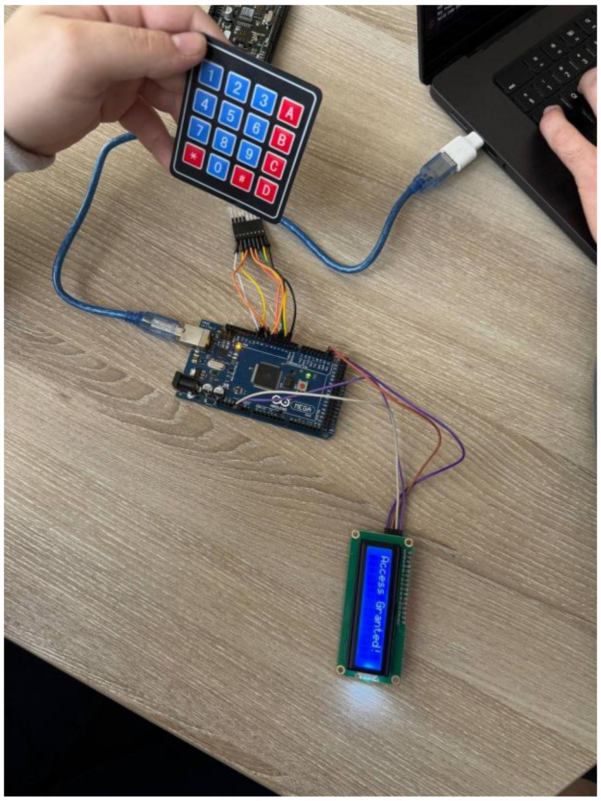


Figure 4.5 „Access granted" message for the correct code and turned on led

Conclusions

After completing this laboratory work, the following results were achieved:

- A functional system was successfully implemented, enabling interaction between a 4x4 keypad and an LCD display using the STDIO library.
- The application correctly detects the input code, verifies its validity, and displays appropriate messages on the LCD.
- A modular architecture was employed, ensuring clear separation of functionalities for peripheral control.
- The system was tested on a microcontroller platform (such as Arduino Uno or ESP32) and validated using a simulator to confirm its correctness.
- A visual indicator (LED) was integrated to provide immediate feedback on code validation—green for a valid code and red for an invalid code.

System Performance Analysis

The system demonstrated stable and efficient operation, with reliable communication between the keypad and LCD via the STDIO library. Key performance aspects include:

- **Response Time**: Code input and validation occur almost instantly, with minimal latency.
- **Reliability**: The application accurately detects user input, validates the code, and provides real-time feedback via both the LCD and LED indicators.
- **Modularity**: The structured design allows easy extension, such as integrating additional sensors or modifying input validation rules.

Identified Limitations

Despite the system's functionality, several areas for improvement were identified:

1. **Basic Code Validation** – Currently, the program only checks for an exact match, without handling partial matches or errors such as accidental extra digits.

2. **Limited User Interaction** – The LCD display is only used for displaying messages, but additional features (e.g., showing previous attempts or guiding the user through input) could enhance usability.
3. **Fixed Code** – The valid code is predefined, making the system less flexible. Implementing a dynamic or user-configurable code would increase versatility.

Impact of the Technology Used in Real-World Applications

This project serves as a practical example of user interaction with embedded systems and has various real-world applications, such as:

- **Security Systems**: Keypads and LCDs are commonly used in access control systems for PIN verification.
- **Industrial Automation**: Similar implementations can be used to control machinery through secure code-based access.
- **IoT and Smart Devices**: Keypad-LCD combinations are often found in home automation systems, vending machines, and ATMs.

Improvement Suggestions

To enhance the project, the following improvements could be considered:

1. **Enhanced Error Handling** – Implementing input correction mechanisms to allow for minor mistakes.
2. **User-Friendly Features** – Providing real-time feedback during code entry (e.g., showing entered digits as '*' for security).
3. **Dynamic Code Configuration** – Allowing users to set or change the valid code instead of relying on a hardcoded value.
4. **Wireless Integration** – Using an ESP32 module to enable remote authentication via Bluetooth or Wi-Fi.
5. **Logging and Security** – Storing failed attempts to enhance security and prevent unauthorized access.

By implementing these optimizations, the system could be made more robust, user-friendly, and adaptable to a wider range of applications.

Note on AI Tool Usage

During the drafting of this report, the author used ChatGPT for generating and structuring the content. The resulting information was reviewed, validated, and adjusted according to the requirements of the laboratory work, ensuring technical accuracy and clarity of explanations. The use of this AI tool was aimed at structuring and optimizing the presentation of information without replacing personal analysis and understanding of the subject.

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/KaBoomKaBoom/ES_Labs.git
2. **Config.h**

```
#include <stdint.h>  // Include this to use uint8_t
```

```
#ifndef CONFIG_H
#define CONFIG_H

#define VALID_CODE "1234"    // Codul valid pentru deblocare

// Pini Keypad
#define ROWS 4
#define COLS 4
const uint8_t rowPins[ROWS] = {9, 8, 7, 6};    // Pini pentru rânduri
const uint8_t colPins[COLS] = {5, 4, 3, 2};    // Pini pentru coloane

// Pini LCD
#define LCD_RS 12
#define LCD_EN 11
#define LCD_D4 10
#define LCD_D5 9
#define LCD_D6 8
#define LCD_D7 7

// Pini LED-uri
#define GREEN_LED A0
#define RED_LED A1

#endif
```

3. KeypadInput.h

```
#include <Arduino.h>
#ifndef KEYPAD_INPUT_H
#define KEYPAD_INPUT_H
```

```
class KeypadInput {
public:
void init();
String readCode();
};

#endif
```

4. LcdDisplay.h

```
#include <Arduino.h>
#ifndef LCD_DISPLAY_H
#define LCD_DISPLAY_H

class LcdDisplay {
public:
void init();
void displayMessage(String message);
};

#endif
```

5. LedControl.h

```
#ifndef LED_CONTROL_H
#define LED_CONTROL_H

class LedControl {
public:
void init();
void greenOn();
void greenOff();
void redOn();
void redOff();
};
```

```
#endif
```

6. KeypadInput.cpp

```
#include <Keypad.h>
#include "Config.h"
#include "KeypadInput.h"

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

Keypad keypad = Keypad(makeKeymap(keys), rowPinsArray, colPinsArray, ROWS, COLS);

void KeypadInput::init() {
// Inițializare Keypad (nu necesită configurații suplimentare)
}


String KeypadInput::readCode() {
String code = "";
char key;
while (code.length() < 4) {
key = keypad.getKey();
if (key) {
code += key;
}
}
return code;
}
```

7. LcdDisplay.cpp

```
#include <LiquidCrystal.h>
#include "Config.h"
#include "LcdDisplay.h"

LiquidCrystal lcd(LCD_RS, LCD_EN, LCD_D4, LCD_D5, LCD_D6, LCD_D7);

void LcdDisplay::init() {
lcd.begin(16, 2);
lcd.clear();
}

void LcdDisplay::displayMessage(String message) {
lcd.clear();
lcd.setCursor(0, 0);
lcd.print(message);
}
```

8. LedControl.cpp

```
#include <Arduino.h>
#include "Config.h"
#include "LedControl.h"

void LedControl::init() {
pinMode(GREEN_LED, OUTPUT);
pinMode(RED_LED, OUTPUT);
digitalWrite(GREEN_LED, LOW);
digitalWrite(RED_LED, LOW);
}
```

```
void LedControl::greenOn() {
digitalWrite(GREEN_LED, HIGH);
}

void LedControl::greenOff() {
digitalWrite(GREEN_LED, LOW);
}

void LedControl::redOn() {
digitalWrite(RED_LED, HIGH);
}

void LedControl::redOff() {
digitalWrite(RED_LED, LOW);
}
```

9. main.cpp

```
#include <Arduino.h>
#include "Config.h"
#include "LedControl.h"
#include "LcdDisplay.h"
#include "KeypadInput.h"

LedControl led;
LcdDisplay lcd;
KeypadInput keypad;

void setup() {
Serial.begin(9600);  // Initialize serial communication
led.init();
lcd.init();
keypad.init();
```

```
printf("System initialized.\n");  // Using printf() to display an initial message
}

void loop() {
String code = keypad.readCode();  // Get code from the keypad

// Using printf() to print the code received from the keypad
printf("Received code: %s\n", code.c_str());

lcd.displayMessage("Cod: " + code);
delay(500);

if (code == VALID_CODE) {
lcd.displayMessage("Cod corect!");
led.greenOn();
led.redOff();
printf("Code correct. LED green turned on.\n");  // Inform about the correct code
} else {
lcd.displayMessage("Cod gresit!");
led.redOn();
led.greenOff();
printf("Code incorrect. LED red turned on.\n");  // Inform about the incorrect code
}
delay(1000);
}

void getInputFromSerial() {
char inputBuffer[10];  // Define a buffer to store input
printf("Enter a code: ");
```

```
// Wait until data is available
while (!Serial.available()) {
delay(100);
}

// Use scanf() to read a string from serial input
scanf("%s", inputBuffer);  // Read input into the buffer

printf("You entered: %s\n", inputBuffer);  // Output the received code
}
```