MINISTRY OF EDUCATION, CULTURE AND RESEARCH OF REPUBLIC OF MOLDOVA
TECHNICAL UNIVERSITY OF MOLDOVA
FACULTY OF COMPUTERS, INFORMATICS AND MICROELECTRONICS
DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATICS

DANIELA COJOCARI, FAF-231

Report

Laboratory work n.1.2

of Embedded Systems

User Interaction: STDIO - LCD + Keypad

Checked by:
Alexei Martiniuc

Chișinău, 2026

1. Analysis of the Situation in the Field

1.1 Description of the technologies used and the context of the developed application.

STDIO Library – The stdio.h header provides generic file operation support and supplies functions with narrow character input/output capabilities. I/O streams are denoted by objects of type FILE that can only be accessed and manipulated through pointers of type FILE*. Each stream is associated with an external physical device (file, standard input stream, printer, serial port, etc) [1].

Serial Interface – A serial interface is a communication interface between two digital systems that transmits data as a series of voltage pulses down a wire. A ”1” is represented by a high logical voltage and a ”0” is represented by a low logical voltage. Essentially, the serial interface encodes the bits of a binary number by their ”temporal” location on a wire rather than their ”spatial” location within a set of wires. Encoding data bits by their ”spatial” location is referred to as a parallel interface and encoding bits by their ”temporal” location is referred to as a serial interface [2].

PlatformIO – PlatformIO is a cross-platform, cross-architecture, multiple framework, professional tool for embedded systems engineers and for software developers who write applications for embedded products. The build system structure automatically tags software dependencies and applies them using a modular hierarchy that takes away the usual complexity and pain [3].

Wokwi Simulator – Wokwi is an embedded systems and IoT simulator supporting ESP32, Arduino, and the Raspberry Pi Pico [4]. It provides a safe environment where mistakes don’t damage hardware, so you can experiment and debug confidently. Projects are easy to share and collaborate on, and you have access to unlimited virtual components without worrying about cost or stock. Its unique features include WiFi simulation, a virtual logic analyzer for capturing digital signals, advanced debugging with GDB, SD card simulation, a Chips API for creating custom components, and integration with Visual Studio Code, making it a comprehensive tool for learning, prototyping, and testing embedded systems.

Microcontroller – A microcontroller unit (MCU) is essentially a small computer on a single chip. It is designed to manage specific tasks within an embedded system without requiring a complex operating system. These compact integrated circuits (ICs) contain a processor core (or cores), random-access memory (RAM) and electrically erasable programmable read-only memory (EEPROM) for storing the custom programs that run on the microcontroller, even when the unit is disconnected from a power supply [5].

LCD display – LCD is a type of flat panel display technology used in various electronic devices like televisions, computer monitors, smartphones, and calculators. An LCD consists of a layer of liquid crystals

sandwiched between two transparent electrodes. When an electric current is applied, the crystals align to control the amount of light passing through them, creating the image you see on the screen. The main components of an LCD include the liquid crystals, the backlight, the color filters, and the electrodes. The liquid crystals control the light passing through them, the backlight provides the light source, the color filters produce the different colors, and the electrodes apply the electric current to manipulate the crystals. LCDs have a wide range of applications. They are commonly used in televisions, computer monitors, laptops, tablets, smartphones, digital cameras, portable gaming devices, and car displays. They are also found in industrial equipment, medical devices, and various consumer electronics [6].

Keypad 4x4 – The 4×4 keypad module is an input device, used to provide input value within a project. This module includes a total of 16 keys which provide 16 input values. These matrix keypad modules are made with flexible membranes and thin material. A 4×4 keypad module is used for entering passwords, dialing a number, browsing through menus and also even controlling robots. These 4×4 keypads can be observed in Telephones, Security Systems, and ATMs, to provide i/p data to the security system by the user. These keypads can also be utilized with Microcontrollers as well as Arduino platforms to execute different projects [7].

The developed application is an embedded user-interaction system built around a microcontroller platform such as an Arduino Uno. It demonstrates how a microcontroller can manage communication between input and output peripherals to create a simple human–machine interface. The user enters a code using a 4×4 matrix keypad, the microcontroller processes and verifies the input, and feedback is provided through an LCD display and LED indicators to signal whether the code is valid or invalid. The system uses the STDIO library together with a serial (UART) interface to handle text-based input/output and communication between the firmware and connected devices. The LCD provides visual feedback, while the keypad serves as the primary input method. Development and testing are supported by tools such as PlatformIO within Visual Studio Code and simulation through Wokwi. Overall, the project illustrates key concepts of embedded systems design, peripheral interfacing, serial communication, and modular firmware development.

1.2 Presentation of the hardware and software components used, explaining the role of each.

Arduino Uno – The Arduino Uno is a widely used microcontroller development board based on the ATmega328P chip. It features digital and analog I/O pins that allow it to interface with sensors, LEDs, and other peripherals. The board can be powered and programmed through a USB connection and supports serial communication, which is essential for receiving commands from a terminal in this project. It acts as

the central “brain” of the system, executing the code that interprets serial commands and controls external components like LEDs.

LEDs – Light-Emitting Diodes (LEDs) are simple semiconductor devices that emit light when an electric current passes through them. They are used in this project as visual indicators to show the result of serial commands (e.g., LED ON or LED OFF). Because LEDs require only small currents, they are suitable for direct control by microcontroller pins.

LCD 2x16 – The 16×2 LCD is an alphanumeric display module capable of showing two rows of sixteen characters each. In this project, it serves as the primary visual feedback device, displaying system messages such as prompts and validation results. The LCD interfaces with the microcontroller through digital data and control lines, allowing the firmware to send text and commands. It enables real-time communication with the user, making the system more intuitive and user-friendly.

Keypad 4x4 – The 4×4 matrix keypad is an input device consisting of sixteen buttons arranged in rows and columns. It allows the user to enter numeric or symbolic codes that are read by the microcontroller through a scanning process that detects which row and column are connected when a key is pressed. In this application, the keypad is used to input an access code, which the microcontroller processes and verifies. It provides a simple and reliable method for human interaction with the embedded system.

220 Ω resistor – The 220 Ω resistor is placed in series with the LED to limit the current flowing through it, preventing damage to both the LED and the microcontroller. Without a resistor, the LED could draw excessive current, potentially leading to failure of the component or the microcontroller pin. Choosing an appropriate resistor value ensures the LED operates within safe current limits.

Breadboard – A breadboard is a solderless prototyping board that allows components and wires to be connected easily without permanent soldering. It provides a flexible way to build and test circuits, making it ideal for educational projects and rapid experimentation. The Arduino, resistors, LEDs, and jumper wires are all placed onto the breadboard to create the necessary electrical connections for the circuit.

Connection cables (jumper wires) – Jumper wires are used to make electrical connections between the Arduino board, the breadboard, and the various components in the circuit. They come with connector pins that fit into the board headers or breadboard holes, enabling modular and reconfigurable wiring. They are essential for linking the microcontroller’s pins to the LED and resistor circuits.

Power supply (USB) – The USB power supply provides both power and data connectivity between the Arduino Uno and a computer. It supplies the 5 V needed to power the microcontroller and peripherals while also enabling code uploads and serial communication with the terminal. Using USB simplifies both development and testing, as the board can run directly from the computer without needing a separate external power source.

Visual Studio Code with PlatformIO extension – Visual Studio Code (VS Code) is a versatile code

editor, and when combined with the PlatformIO extension, it becomes a powerful environment for embedded systems development. PlatformIO supports multiple microcontroller platforms and frameworks and handles project configuration, compilation, and uploading of firmware to the Arduino. It simplifies dependency management and allows advanced features like unit testing and integrated terminal access. This environment is used to write, build, and deploy the application that reads serial input and controls the LED.

Serial terminal emulator (e.g., PlatformIO Serial Monitor, TeraTerm, PuTTY) – A serial terminal emulator is software that connects to the Arduino's serial port and allows text communication with the microcontroller. In this project, the terminal is used to send commands such as "led on" and "led off" to the Arduino and to view textual feedback confirming command reception. The emulator displays the serial output and serves as the user interface for testing serial communication functionality.

Hardware simulator - Wokwi – Wokwi is an embedded systems and IoT simulator that supports Arduino, ESP32, Raspberry Pi Pico, and other microcontroller platforms. It can be used directly in a browser or integrated into VS Code. Wokwi enables simulation of embedded circuits and serial communication without physical components, allowing developers to prototype and debug designs virtually. It offers features such as WiFi simulation, logic analyzers, and interactive virtual components, making it a convenient tool for experimentation and verification before deploying to real hardware.

1.3 Explanations of the system architecture and justification for the chosen solution.

The system is organized into clearly defined layers to separate responsibilities and simplify interaction with the hardware, and this modular approach is central to the chosen solution. The application layer, implemented in 'lab_1_2_app', handles the main logic of reading a code from the keypad, comparing it to a predefined value, and providing feedback through the LCD and LEDs. Using standard C functions for input and output allows the application to remain simple and readable, without requiring direct management of hardware details. This approach was chosen because it provides a consistent and portable interface for interacting with peripherals while keeping the core logic straightforward and maintainable.

The peripheral abstraction layer was implemented to manage the hardware interactions with the LCD, keypad, and LEDs. The LCD driver redirects 'stdout' to the display, handling special commands like clearing the screen, while the keypad driver maps key presses to 'stdin', enabling seamless reading of user input. The LED driver provides straightforward functions to turn LEDs on and off for visual feedback. This separation isolates hardware-specific operations from the application logic, making the system modular and flexible. It also allows hardware components to be modified or expanded without requiring changes to the main application code.

At the hardware level, the MCU handles low-level control of the peripherals. The LCD uses I2C, reducing pin usage and simplifying connections, the keypad scans digital pins in a standard matrix configuration, and the LEDs use digital output pins for immediate feedback. Choosing this hardware configuration is justified because it is widely available, efficient, and effective for implementing interactive embedded systems.

Combining these hardware choices with STDIO-based abstraction and modular drivers results in a system that is reliable, maintainable, and easily extensible, providing a clear and practical architecture for user interaction in embedded applications.

1.4 A relevant case study demonstrating the applicability of the proposed solution.

A relevant case study is presented in the article “Microcontroller Based Digital Door Lock Security System Using Keypad,” [8] which demonstrates a practical application of a microcontroller-based security system for homes, schools, and offices. The system uses a 4×4 keypad as the input unit for entering a Personal Identification Number (PIN) and a Liquid Crystal Display (LCD) to provide visual feedback to the user. A servo motor acts as the locking mechanism, while a programmed microcontroller processes the entered PIN and takes the appropriate action. When a user enters a PIN, the system captures it and compares it to the stored PINs. If a match is found, the LCD displays “access granted” and the door unlocks; if not, it displays “access denied” and the door remains locked. The system also supports changing the PIN: the user presses the ”#” button, enters the current PIN for verification, and then enters the new PIN twice. The system achieved a 100% success rate for registered users and a 100% failure rate for unauthorized attempts.

This case study is highly relevant to the proposed solution because it demonstrates the same interaction pattern of reading input from a keypad, processing it with a microcontroller, and providing visual feedback via an LCD. Just like in the project, the system relies on modular components, keypad for input, LCD for output, and microcontroller logic for decision-making, to perform a verification task. It validates that using STDIO-style abstraction for peripheral communication, combined with modular driver design, is effective for real-world applications where reliable input, output, and feedback are essential. Moreover, it highlights the practicality of the architecture for secure, interactive, and user-friendly embedded systems, which aligns directly with the objectives and design choices of the proposed solution.

2. Implementation

2.1 Architectural Sketch

The architectural diagram represents the layered structure of the embedded system designed for user interaction using a 4×4 keypad, LCD, and LEDs. Each layer has a specific role, enabling modularity, clarity, and hardware abstraction.

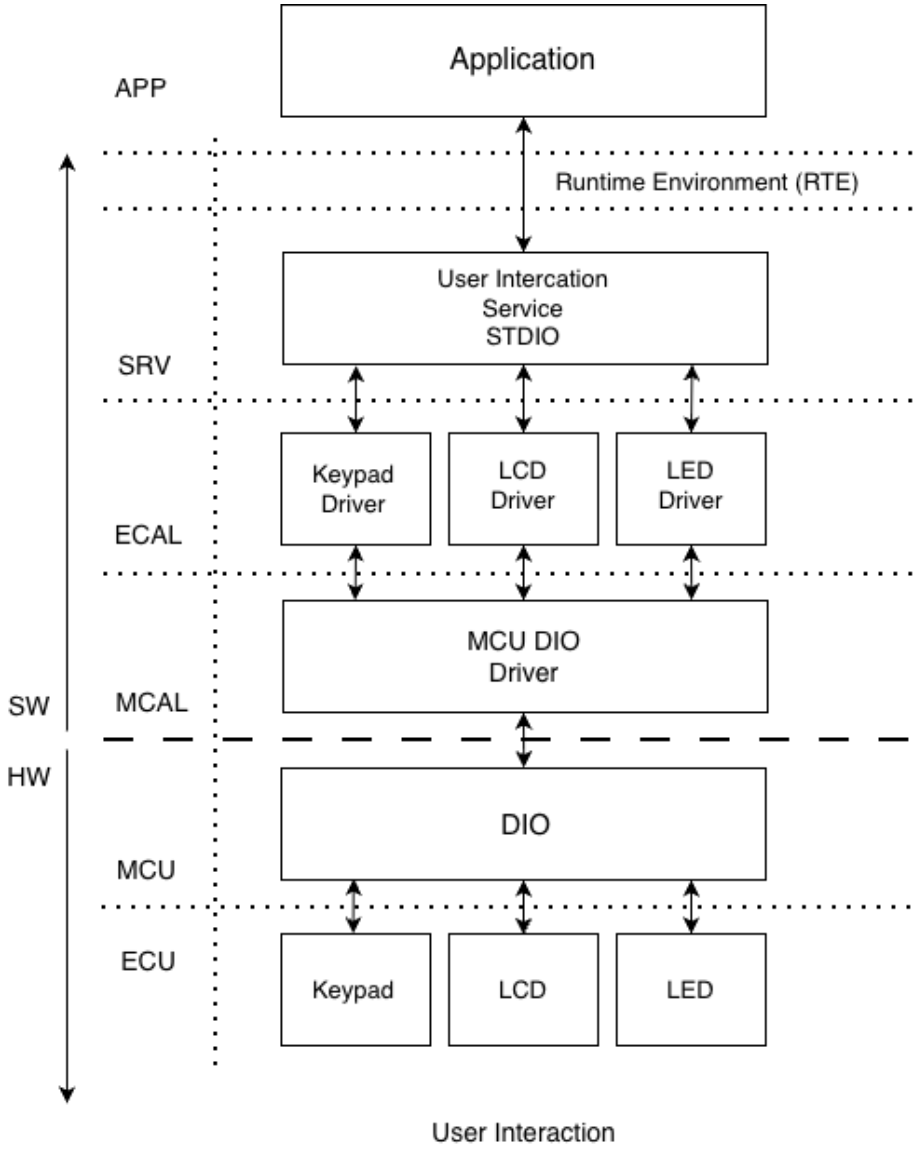

Figure 0.0.1 - Architetural Diagram

Application Layer

The Application Layer is the topmost layer of the system and contains the main program logic. In this project, it is implemented in the lab_1_2_app module. Its primary function is to manage the user interaction process: reading input from the keypad, comparing the input to a predefined code, and triggering the corresponding actions. When a correct code is entered, it displays a success message on the LCD and turns on

the green LED. Conversely, if the code is incorrect, it displays an error message and turns on the red LED. By isolating all application logic at this level, the system ensures that hardware-specific details are handled separately, making the code easier to maintain and extend.

User Interaction Service (STDIO)

The User Interaction Service, implemented using the STDIO library, serves as an abstraction layer between the application and the physical peripherals. The STDIO functions, such as printf and scanf, are redirected to the hardware drivers: stdout is mapped to the LCD, and stdin is mapped to the keypad. This allows the application to read and write text data in a standardized manner without needing to directly manage hardware communication. This design simplifies the development process, makes the code more portable, and allows the logic to remain hardware-independent while still interacting effectively with external devices.

Peripheral Drivers

The Peripheral Drivers layer provides a bridge between the abstract STDIO interface and the actual hardware. It contains separate drivers for each peripheral: the keypad, LCD, and LEDs. The keypad driver handles scanning the rows and columns of the 4×4 matrix and returns the pressed key to the STDIO interface. The LCD driver manages the display, including printing characters, controlling the cursor, and clearing the screen. The LED driver provides simple functions to turn the green and red LEDs on or off based on system events. These drivers encapsulate all hardware-specific operations, allowing the application to function without knowledge of pin configurations or communication protocols.

MCU DIO Driver

The MCU Digital Input/Output (DIO) Driver is responsible for low-level control of the microcontroller's pins. It acts as an interface between the peripheral drivers and the physical hardware. This layer ensures that higher-level drivers can manipulate the LEDs, keypad, and LCD without directly handling electrical signals, timing, or voltage levels. By isolating pin-level operations in a dedicated driver, the system improves modularity, making it easier to adapt to different microcontrollers or hardware revisions.

Digital I/O (DIO)

The Digital I/O (DIO) layer represents the microcontroller's actual digital pins used for communication with external devices. These pins carry the signals for input and output: they detect key presses from the keypad, send data and control signals to the LCD (via I2C), and supply current to the LEDs. The DIO layer abstracts the physical connection points and provides a standardized interface for the MCU DIO driver to interact with.

Hardware Components

At the base of the architecture are the physical hardware components: the 4×4 keypad, LCD display, and LEDs. The keypad serves as the primary input device, allowing the user to enter numeric codes. The

LCD provides visual feedback by displaying prompts, messages, and results. The LEDs offer immediate visual indication of the system status, turning green for a correct code and red for an incorrect code. These components form the tangible interface for user interaction, while all higher layers manage the logical processing and control signals needed for their operation.

The presented layered architecture, from the hardware components up to the application layer, effectively separates responsibilities and abstracts peripheral control. This structure demonstrates how modular drivers, STDIO-based interaction, and MCU-level control work together to create a maintainable, flexible, and user-friendly embedded system.

2.2 Functional Block Diagrams

LCD:

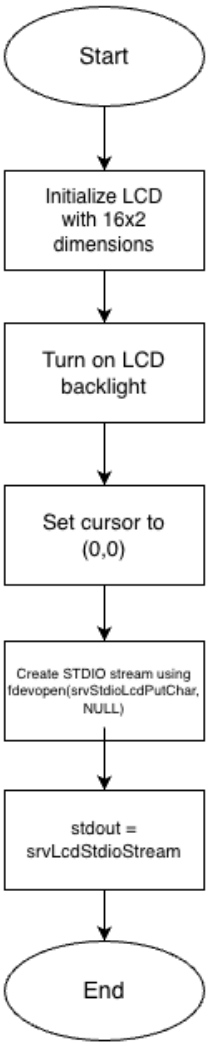

Figure 0.0.2 - srvStdioLcdSetup()

The *srvStdioLcdSetup()* flowchart [0.0.2] illustrates the initialization and configuration of the LCD for use with the standard output stream. First, the LCD is initialized with the defined number of columns and

rows, and the backlight is turned on. The cursor is then set to the starting position (first column of the first row). Next, a custom STDIO stream is created by linking srvStdioLcdPutChar to handle character output. Finally, the program redirects stdout to this custom stream, allowing all standard output functions, like printf, to display text directly on the LCD.

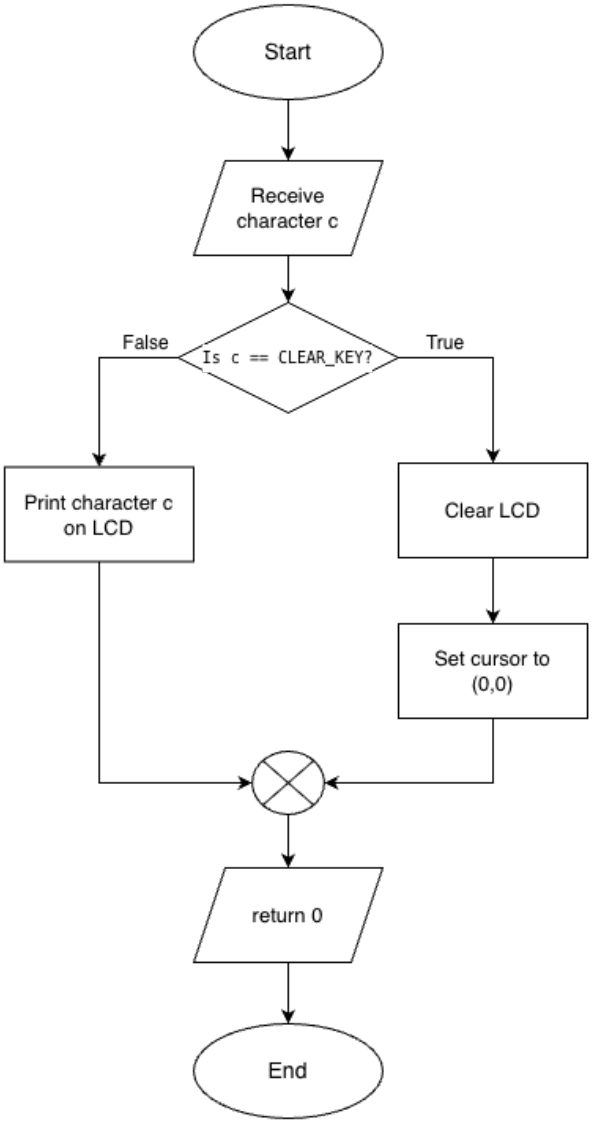

Figure 0.0.3 - srvStdioLcdPutChar()

The *srvStdioLcdPutChar()* flowchart [0.0.3] shows the handling of the output of individual characters to the LCD. When a character is received, it first checks if the character matches the CLEAR_KEY code. If it does, the LCD is cleared and the cursor is reset to the first column of the first row. If the character is not CLEAR_KEY, it is printed directly to the current cursor position on the LCD. The function then returns 0, indicating successful handling of the character. This logic allows both standard character printing and display clearing via a single interface.

Keypad:

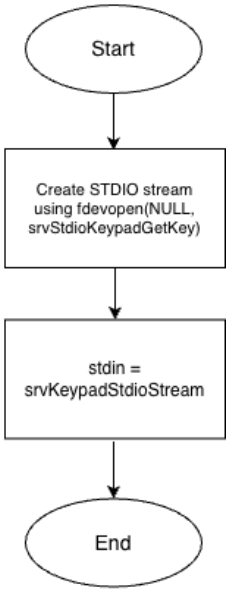

Figure 0.0.4 - srvStdioKeypadSetup()

The *srvStdioKeypadSetup()* flowchart [0.0.4] presents the configuration of the keypad as a standard input device. A custom STDIO stream is created and linked to the keypad key-reading function, after which the stream is assigned to stdin. This allows standard input functions, such as scanf, to read characters directly from the keypad, enabling seamless user interaction.

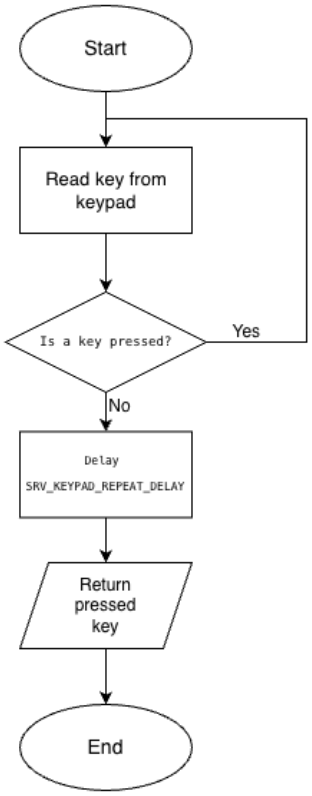

Figure 0.0.5 - srvStdioKeypadGetKey()

The *srvStdioKeypadGetKey()* flowchart [0.0.5] illustrates the process of reading a key from the 4×4 matrix keypad. The function continuously checks the keypad until a valid key press is detected. Once a key is pressed, the system waits for a short delay to prevent repeated rapid readings, then returns the detected character. This mechanism ensures reliable and stable keypad input for the system.

LED:

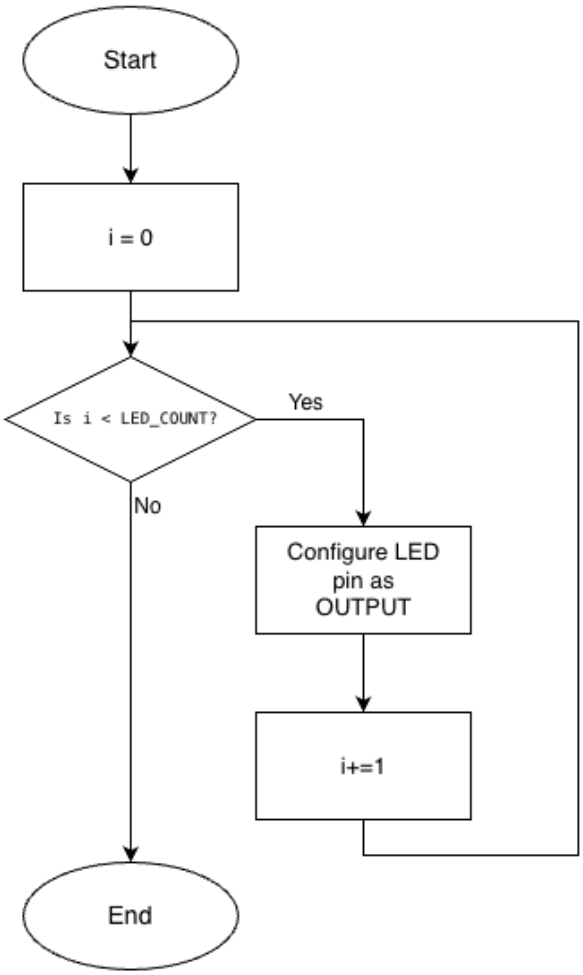

Figure 0.0.6 - ddLedSetup()

The *ddLedSetup()* flowchart [0.0.6] illustrates the initialization of the LED driver. The function iterates through all defined LED pins and configures each one as an output. This setup step prepares the microcontroller to control the LEDs by enabling digital output on the corresponding pins.

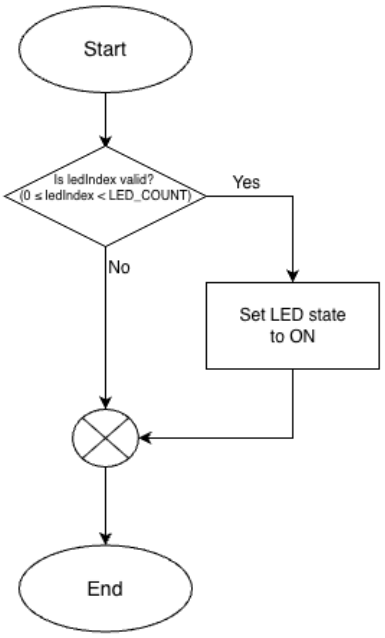

Figure 0.0.7 - ddLedTurnOn()

The *ddLedTurnOn()* flowchart [0.0.7] describes the process of turning on an LED. The function first verifies that the provided LED index is within the valid range. If the index is valid, the corresponding LED pin is activated, turning the LED on. If the index is invalid, no action is performed, ensuring safe operation.

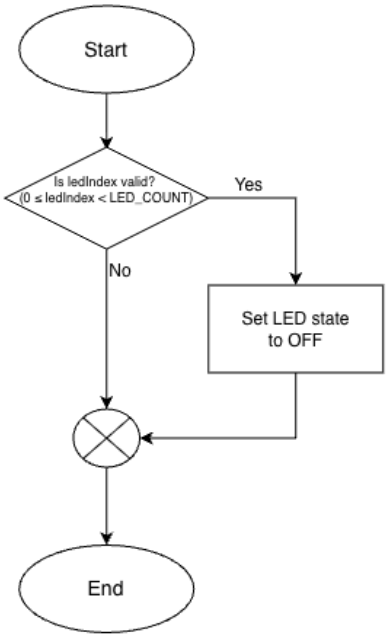

Figure 0.0.8 - ddLedTurnOff()

The *ddLedTurnOff()* flowchart [0.0.8] shows how an LED is turned off. The function checks whether the specified LED index is valid. When the index is within the allowed range, the corresponding LED pin is deactivated, turning the LED off. If the index is invalid, the function performs no action, preventing

unintended behavior.

Lab1_2:

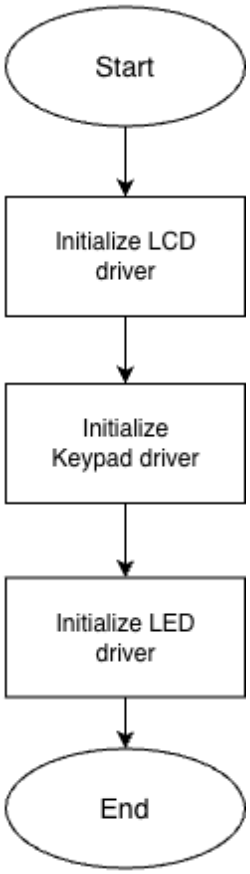

Figure 0.0.9 - lab1_2Setup()

The *lab1_2Setup()* flowchart [0.0.9] presents the initialization phase of the embedded system. The LCD service is initialized to enable text display, the keypad service is configured to provide user input through the standard input stream, and the LED driver is initialized to allow visual feedback. After these components are prepared, the system is ready to interact with the user.

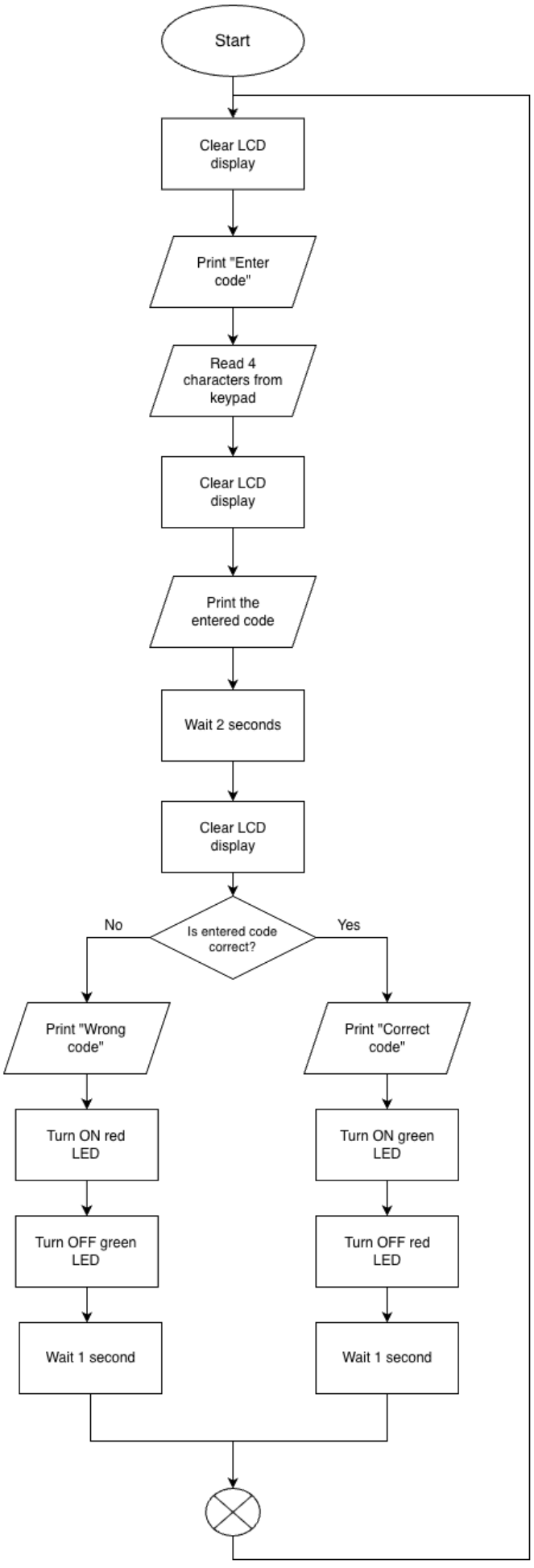

Figure 0.0.10 - lab1_2Loop()

The *lab1_2Loop()* flowchart [0.0.10] illustrates the main operational cycle of the system. The LCD is cleared and the user is prompted to enter a four-digit code using the keypad. After the input is received, the entered code is displayed briefly for confirmation. The system then compares the entered code with the predefined access code. If the codes match, a success message is displayed, the green LED is activated, and the red LED is turned off. If the codes do not match, an error message is displayed, the red LED is activated, and the green LED is turned off. After a short delay, the process repeats, allowing continuous user interaction.

2.3 Electrical Sketch

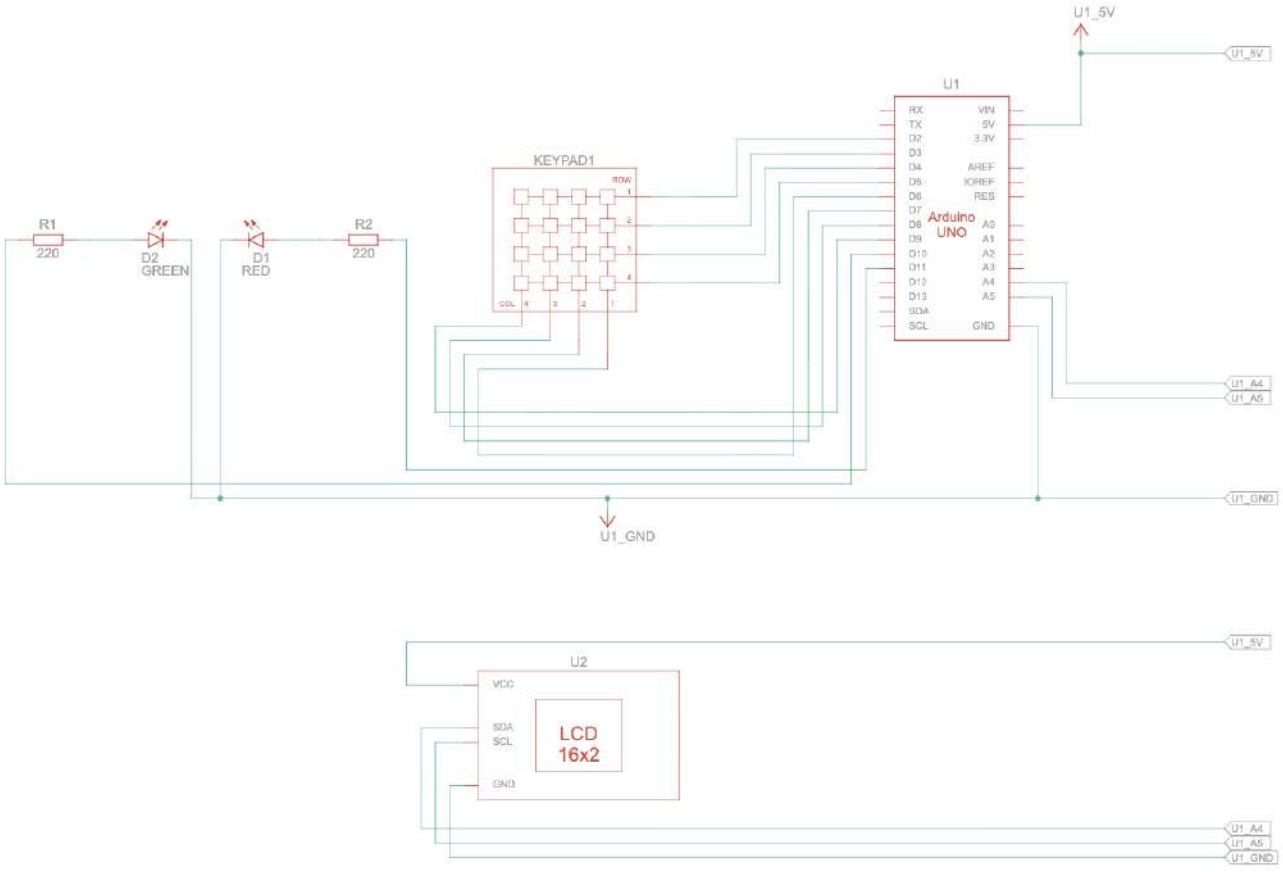

Figure 0.0.11 - Electrical Sketch

The electrical sketch [0.0.11] represents a microcontroller-based access control system built around an Arduino Uno. The circuit integrates a 4×4 matrix keypad for user input, a 16×2 LCD with an I2C interface for visual feedback, and two indicator LEDs that signal whether the entered code is correct or incorrect. All

components share a common power and ground reference provided by the Arduino.

The Arduino Uno serves as the central control unit. Its 5 V output supplies power to the peripheral modules, while the GND pin provides the common reference required for correct operation. The analog pins A4 and A5 are used for I2C communication, forming the data and clock lines for the LCD module. Digital I/O pins are used to interface with the keypad rows and columns and to control the LEDs.

The 4×4 matrix keypad is connected using eight digital pins. Four lines are assigned to the rows and four to the columns, allowing the microcontroller to detect which key is pressed by scanning the matrix. This arrangement reduces the number of required pins compared to wiring each key individually and matches the keypad driver configuration used in the software.

The LCD module operates via an I2C interface, which simplifies wiring by requiring only four connections: VCC, GND, SDA, and SCL. The VCC pin is connected to the Arduino's 5 V supply, and GND is connected to the common ground. The SDA line connects to Arduino pin A4, and the SCL line connects to A5. This interface allows the microcontroller to send text and control commands to the display while minimizing pin usage.

Two LEDs provide visual status indication. Each LED is connected in series with a 220 Ω current-limiting resistor to protect both the LED and the microcontroller output pin. The green LED indicates a correct code entry, while the red LED signals an incorrect code. In each case, the resistor connects to the assigned Arduino digital output pin, the resistor leads to the LED anode, and the LED cathode connects to ground. When the Arduino outputs a HIGH signal, current flows through the resistor and LED to ground, illuminating the LED.

All grounds in the circuit are tied together to ensure a common electrical reference. The design follows safe current-limiting practices, uses appropriate communication interfaces, and matches the pin assignments and functionality defined in the software. The resulting system allows a user to enter a code via the keypad, view prompts and feedback on the LCD, and receive visual confirmation through the LEDs.

3. Results

The experimental results confirm the correct operation of the assembled system and validate both the hardware connections and the software logic.

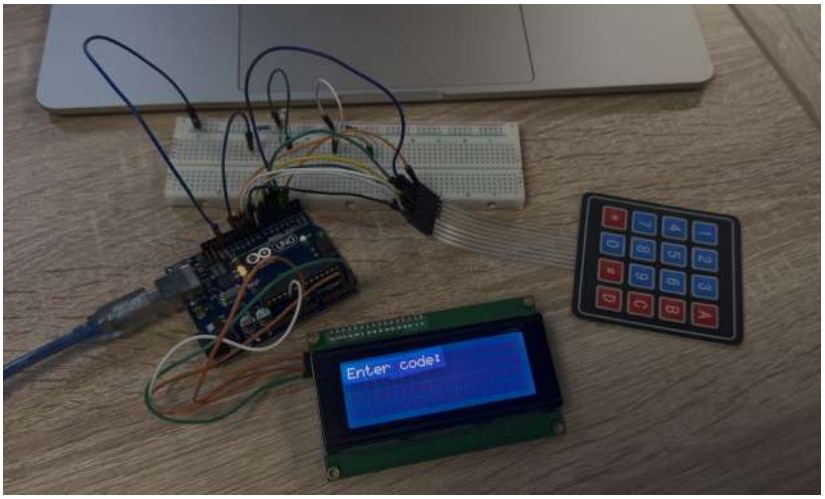

Figure 0.0.12 - Initial display of entering the code

In the first image [0.0.12], the complete hardware setup is shown functioning as intended. The LCD displays the message “Enter code:”, indicating that the system has powered on, initialized all peripherals, and is waiting for user input from the keypad.

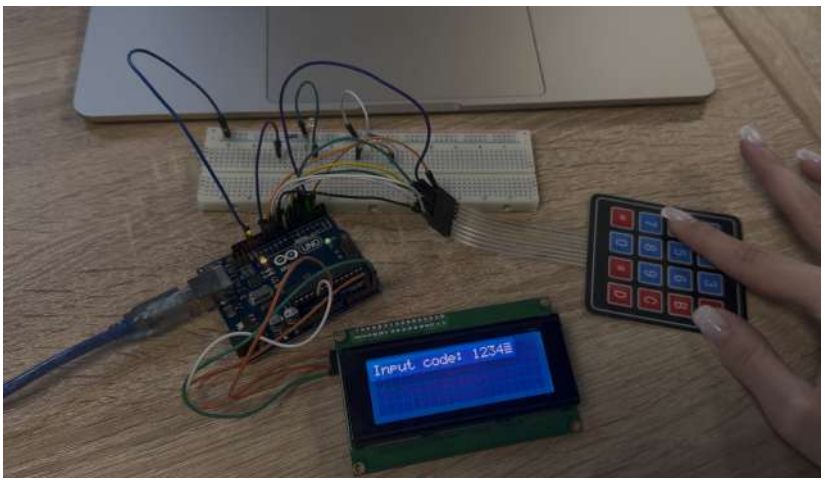

Figure 0.0.13 - Input display of the correct code

In the second image [0.0.13], after the user enters the correct four-digit sequence, the LCD displays “Input code: 1234”. This confirms that the keypad is correctly detected by the microcontroller, each key press is properly read, and the characters are transmitted through the standard input stream to the program. It also demonstrates that the LCD is correctly receiving and displaying formatted output.

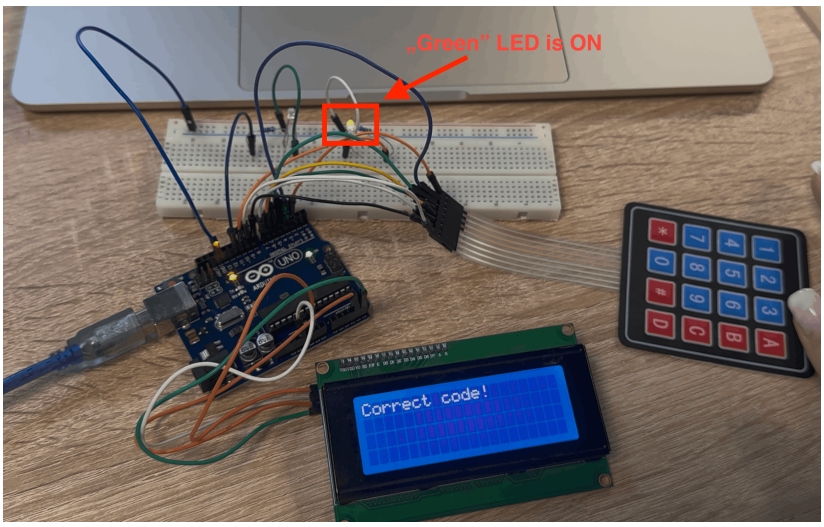

Figure 0.0.14 - Correct code processing

In the third image [0.0.14], the LCD displays “Correct code” and the green LED is illuminated. This shows that the program successfully compares the entered sequence with the stored reference code and executes the correct branch of the decision logic. The activation of the green LED confirms proper control of the output pins and correct wiring of the indicator LED circuit.

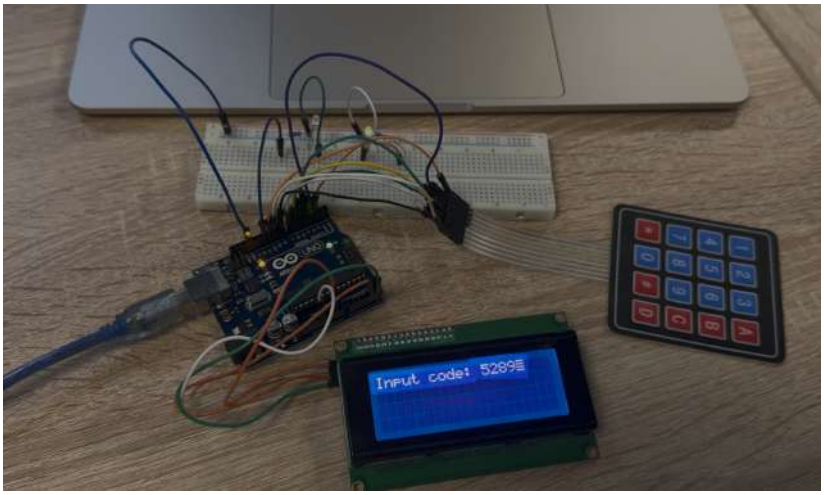

Figure 0.0.15 - Input display of the wrong code

In the fourth image [0.0.15], the LCD displays “Input code: 5289”, representing an incorrect sequence entered via the keypad. This verifies that the system continues to accept new inputs and display them correctly after completing a previous verification cycle.

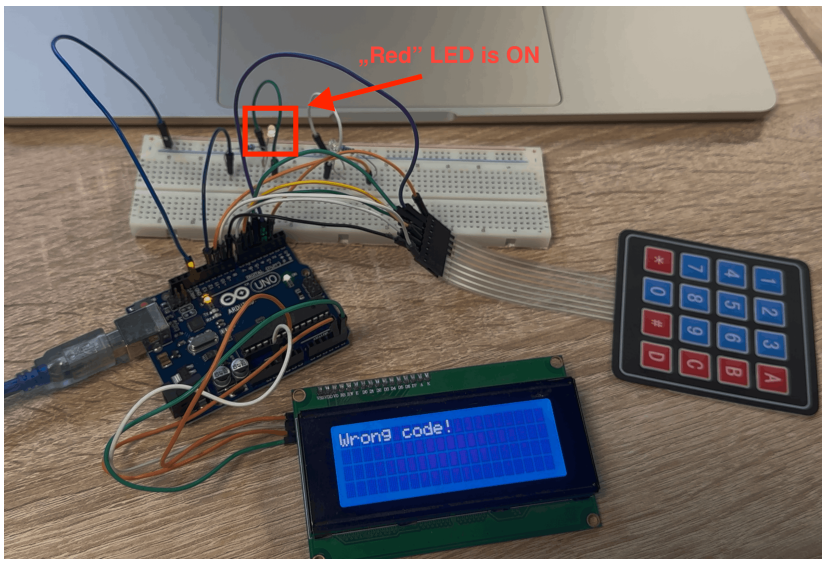

Figure 0.0.16 - Wrong code processing

In the fifth image [0.0.16], the LCD displays “Wrong code” and the red LED is illuminated. This demonstrates that when the entered sequence does not match the stored code, the system executes the alternative branch of the decision logic. The red LED provides a clear visual indication of an incorrect entry, confirming proper functionality of the error signaling mechanism.

4. Conclusion

The developed system successfully integrates an Arduino Uno microcontroller, a 4×4 matrix keypad, an I²C LCD display, and LED indicators into a functional access verification application. The hardware assembly operates reliably, and the electrical connections ensure stable communication between all modules. During testing, the system consistently powered up, displayed the user prompt, accepted keypad input, and provided immediate feedback via the LCD and LEDs.

The keypad interface correctly captures each key press and transmits the characters to the microcontroller through the implemented standard input stream. The LCD module provides clear guidance and feedback messages, improving usability and allowing the user to verify their input before validation. The LED indicators serve as an intuitive visual output, clearly distinguishing between successful and unsuccessful authentication attempts.

From a software perspective, the modular structure using dedicated drivers for the LCD, keypad, and LEDs improves readability, maintainability, and scalability of the application. Redirecting standard input and output streams to hardware devices simplifies user interaction and demonstrates an effective method for embedded system interface design. The decision logic reliably compares the entered code with the stored reference and executes the appropriate response branch.

The experimental results confirm that the system performs as intended under repeated operation, handling both correct and incorrect inputs without malfunction. This demonstrates robustness and confirms that the integration between hardware and software components is stable.

Overall, the project meets its objectives by providing a simple, responsive, and user-friendly access control mechanism. The implemented solution demonstrates fundamental embedded systems concepts including peripheral interfacing, user input handling, output signaling, and structured program design.

5. Note Regarding the Use of AI Tools

During the preparation of this report, the author used ChatGPT to help structure points and organize content. All information generated with the tool was reviewed, validated, and adjusted to meet the requirements of the laboratory work.

Bibliography

[1] File input/output.
https://en.cppreference.com/w/c/io.html

[2] What is a serial interface?
https://academicweb.nd.edu/~lemmon/courses/ee224/web-manual/web-manual/lab9/node4.html

[3] What is PlatformIO?
https://docs.platformio.org/en/latest/what-is-platformio.html

[4] Welcome to Wokwi!
https://docs.wokwi.com/

[5] What is a microcontroller?
https://www.ibm.com/think/topics/microcontroller

[6] What is a liquid crystal display (LCD)?
https://www.lenovo.com/us/en/glossary/what-is-lcd/

[7] 4×4 Keypad Module : Specifications, PinOut, Interfacing, Working and Its Applications
https://www.watelectronics.com/4x4-keypad-module/

[8] Microcontroller Based Digital Door Lock Security System Using Keypad https://www.researchgate.net/publication/336253923_Microcontroller_Based_Digital_Door_Lock_Security_System_Using_Keypad

Appendix - Source Code

dd_led:

```
// dd_led.cpp


#include "dd_led.h"


// LED pins array
static const int ledPins[LED_COUNT] = {
    LED_GREEN_PIN,
    LED_RED_PIN
};


void ddLedSetup() {
    for (int i = 0; i < LED_COUNT; i++) {
        pinMode(ledPins[i], OUTPUT);
    }
}


void ddLedTurnOn(int ledIndex) {
    if (ledIndex >= 0 && ledIndex < LED_COUNT) {
        digitalWrite(ledPins[ledIndex], HIGH);
    }
}


void ddLedTurnOff(int ledIndex) {
    if (ledIndex >= 0 && ledIndex < LED_COUNT) {
        digitalWrite(ledPins[ledIndex], LOW);
    }
}
```

```
// dd_led.h


#ifndef DD_LED_H
```

```
#define DD_LED_H

#include <Arduino.h>

// LED array configuration
#define LED_COUNT 2

// LED indices for easy reference
#define LED_GREEN 0
#define LED_RED 1

// Default pin assignments
#define LED_GREEN_PIN 10
#define LED_RED_PIN 11

void ddLedSetup();
void ddLedTurnOn(int ledIndex);
void ddLedTurnOff(int ledIndex);

#endif //DD_LED_H
```

srv_stdio_keypad:

```
// srv_stdio_keypad.cpp

#include "srv_stdio_keypad.h"

// Include STDIO library
#include <stdio.h>

// Include keypad library
#include <Keypad.h>

#define SRV_KEYPAD_REPEAT_DELAY 100
```

```
// Configuration for the keypad
const byte ROWS = 4; // four rows
const byte COLS = 4; // four columns

// Define the cymbols on the buttons of the keypads
char hexaKeys[ROWS][COLS] = {
    {'1', '2', '3', 'A'},
    {'4', '5', '6', 'B'},
    {'7', '8', '9', 'C'},
    {'*', '0', '#', 'D'}};
byte rowPins[ROWS] = {2, 3, 4, 5}; // connect to the row pinouts of the keypad
byte colPins[COLS] = {6, 7, 8, 9}; // connect to the column pinouts of the keypad

// initialize an instance of class NewKeypad
Keypad customKeypad = Keypad(makeKeymap(hexaKeys), rowPins, colPins, ROWS, COLS);

// Define the function to get a character from the keypad driver
int srvStdioKeypadGetKey(FILE *stream)
{
    char customKey;
    do
    {
        customKey = customKeypad.getKey();
    } while (customKey==NO_KEY);

    delay(SRV_KEYPAD_REPEAT_DELAY); // Read frequency 1/SRV_KEYPAD_REPEAT_DELAY
       Hz

    return customKey;
}



void srvStdioKeypadSetup()
```

```
{
    // Does not require any setup

    // Define stream
    FILE *srvKeypadStdioStream = fdevopen(NULL, srvStdioKeypadGetKey);

    // Link stream to keypad
    stdin = srvKeypadStdioStream;
}
```

```
// srv_stdio_keypad.h

#ifndef SRV_STDIO_KEYPAD_H
#define SRV_STDIO_KEYPAD_H

void srvStdioKeypadSetup();
int srvStdioKeypadGetKey();

#endif //SRV_STDIO_KEYPAD_H
```

srv_stdio_lcd:

```
// srv_stdio_lcd.cpp

#include "srv_stdio_lcd.h"

#include <LiquidCrystal_I2C.h>

#include <stdio.h>

// Code ASCII used to clear the LCD
#define CLEAR_KEY 0x1b

int lcdColumns = 16;
```

```
int lcdRows = 2;


// Create an LCD object
LiquidCrystal_I2C lcd(0x27, lcdColumns, lcdRows);  // Set the LCD I2C address,
   columns, and rows


int srvStdioLcdPutChar(char c, FILE *stream)
{
    // Print a character to the LCD
    if (c == CLEAR_KEY) {
        lcd.clear();
        lcd.setCursor(0, 0);
    } else {
        lcd.print(c);
    }
    return 0;
}


void srvStdioLcdSetup()
{
    // Initialize the LCD
    lcd.init();


    // Turn on the backlight
    lcd.backlight();


    // Set the cursor to the first column of the first row
    lcd.setCursor(0, 0);


    FILE *srvLcdStdioStream = fdevopen(srvStdioLcdPutChar, NULL);


    stdout = srvLcdStdioStream;
}
```

```
// srv_stdio_lcd.h


#ifndef SRV_STDIO_LCD_H
#define SRV_STDIO_LCD_H


void srvStdioLcdSetup();


#endif //SRV_STDIO_LCD_H
```

lab_1_2_app:

```
// lab_1_2_app.cpp


#include "lab_1_2_app.h"
#include "srv_stdio_lcd/srv_stdio_lcd.h"
#include "srv_stdio_keypad/srv_stdio_keypad.h"
#include "dd_led/dd_led.h"
#include "stdio.h"
#include <Arduino.h>


void lab1_2Setup() {
    // Initialize LCD driver
    srvStdioLcdSetup();


    // Initialize keypad driver
    srvStdioKeypadSetup();


    // Initialize LED driver
    ddLedSetup();
}


char code[4] = {'1', '2', '3', '4'};
char input[4];
```

```
void lab1_2Loop() {
    // Clear the LCD
    putchar(0x1b);

    printf("Enter code: ");

    // Read a code from the keypad
    scanf("%c%c%c%c", &input[0], &input[1], &input[2], &input[3]);

    // Clear the LCD
    putchar(0x1b);

    // Print input code to console
    printf("Input code: %c%c%c%c\n", input[0], input[1], input[2], input[3]);

    delay(2000);

    // Clear the LCD
    putchar(0x1b);

    if (input[0] == code[0] && input[1] == code[1] && input[2] == code[2] &&
       input[3] == code[3]) {
        printf("Correct code!");
        ddLedTurnOn(LED_GREEN);
        ddLedTurnOff(LED_RED);
        delay(1000);

    } else {
        printf("Wrong code!");
        ddLedTurnOn(LED_RED);
        ddLedTurnOff(LED_GREEN);
        delay(1000);

    }
```

```
}
```

```
// lab_1_2_app.h


#ifndef LAB_1_2_APP_H
#define LAB_1_2_APP_H


void lab1_2Setup();
void lab1_2Loop();


#endif //LAB_1_2_APP_H
```

Github link: https://github.com/DannaCojocari/Sisteme_Incorporate/tree/main/src/lab_1_2