**TECHNICAL UNIVERSITY OF MOLDOVA**

**FACULTY OF COMPUTERS, INFORMATICS   
AND MICROELECTRONICS**

**DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATICS**

**Laboratory work nr. 1.2  
User interaction - LCD+Keypad STDIO**

A elaborat: **Brinza Cristian**

st. gr. FAF-212

A verificat: **Moraru Dumitru**

lect. univ

**Chișinău 2024**

# THE TASK OF THE LABORATORY WORK:

Configure the application to work with the STDIO library through the serial interface for text exchange via LCD+Keypad.

Design an MCU-based application for detecting a code from a 4x4 keyboard, verifying the code and displaying a message on an LCD for a valid code, a green LED should light up, for an invalid code, a red LED.

* Use CamelCase coding conventions;
* Use **STDIO** to scan the keyboard and display on the LCD.
* Declare port variable like intended;
* The functionalities for each peripheral device (LED, button, LCD, keypad) should be implemented in separate files for the purpose of reuse in future projects.

# PROGRESS OF THE WORK

## 1 Main functions/methods used to execute the task

Explication about this chapterIn this chapter I will explain the functionality of diferent parts of the executed task:

1. In the **KeypadControl.h** file:

* **KeypadControl():** Constructor for initializing the keypad control.
* **setupKeypad():** Sets up the keypad by defining the rows and columns pins and setting up the keypad layout. setupKeypad(): Sets up the keypad by defining the rows and columns pins and setting up the keypad layout.
* **verifyPassword(**char\* inputPassword**):** Verifies the input password and chack it to not mach against the correct password stored in the code.
* **getKeyPressed():** Reads the key pressed on the keypad and returns the corresponding character(chack is the input is equal to predefined matrix).
* **checkInputTimeout(**unsigned long timeout**):** Checks if there has been a timeout for input entry, the value is unasinger.
* **showInputFeedback(**boolean correct**):** Provides visual feedback (e.g., LED blinking) based on whether the input password was correct or incorrect.

1. In the **LcdDisplay.h** file:

* **LcdDisplay():** Constructor for initializing the LCD display.
* **setupLcd():** Sets up the LCD display by defining the pin connections to the Arduino and initializing an instance of the LiquidCrystal class.
* **clearDisplay():** Clears the content displayed on the LCD screen.
* **printMessage(**char\* message**):** Prints a message on the LCD screen.

1. In the **LEDControl.h** file:

* **initializeLed():** Initializes the LEDs used for indicating lock status by setting the pin modes of the red and green LEDs to OUTPUT.

1. In the **sketch.ino** file:

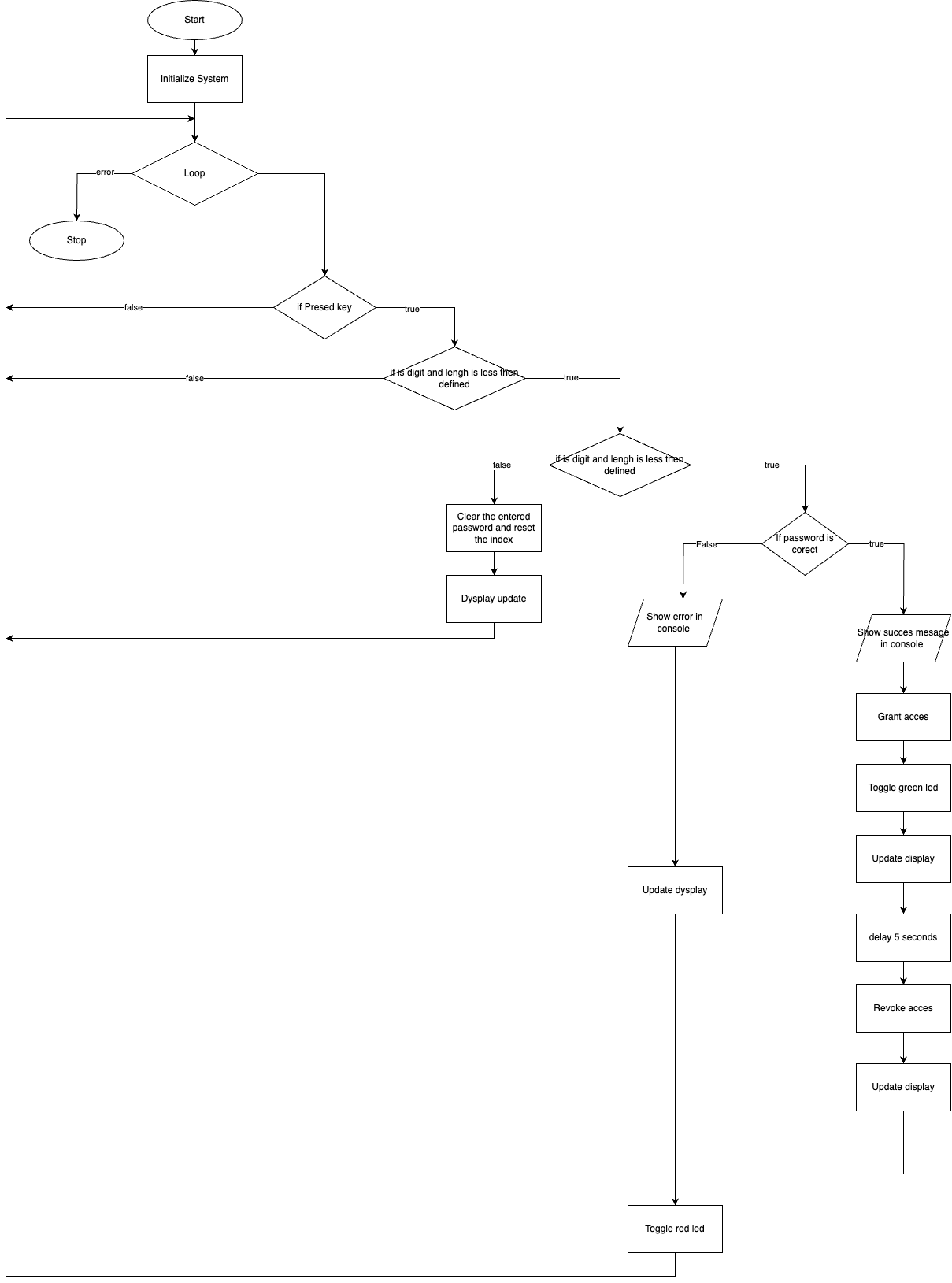
* **setup():** Initializes the Arduino's digital and serial communication, initializes the LCD display, displays initial instructions on the LCD, initializes LED indicators to show the lock's status, and prints lab and system readiness information on the serial monitor.
* **loop():** The main program loop that runs continuously after setup. It typically contains the core logic of the program, including reading input from the keypad, verifying the password, updating the LCD display, and controlling the LEDs based on the lock's status.

## Block Diagram

The diagram is consisting of the following main blocks:

* **Main Program (**sketch.ino**):** The central point that initializes and controls other components (Keypad, LCD Display, and LEDs) and manages the program flow.
* **KeypadControl Class:** Represents the logic for interfacing with the keypad, including setup, password verification, input handling, and feedback display.
* **LcdDisplay Class:** Handles the initialization and control of the LCD display, including setup and message display.
* **LEDControl Class:** Manages the LED indicators for indicating lock status, including initialization and control.
* **Arduino Board:** The physical layer where the keypad, LCD display, and LEDs are connected

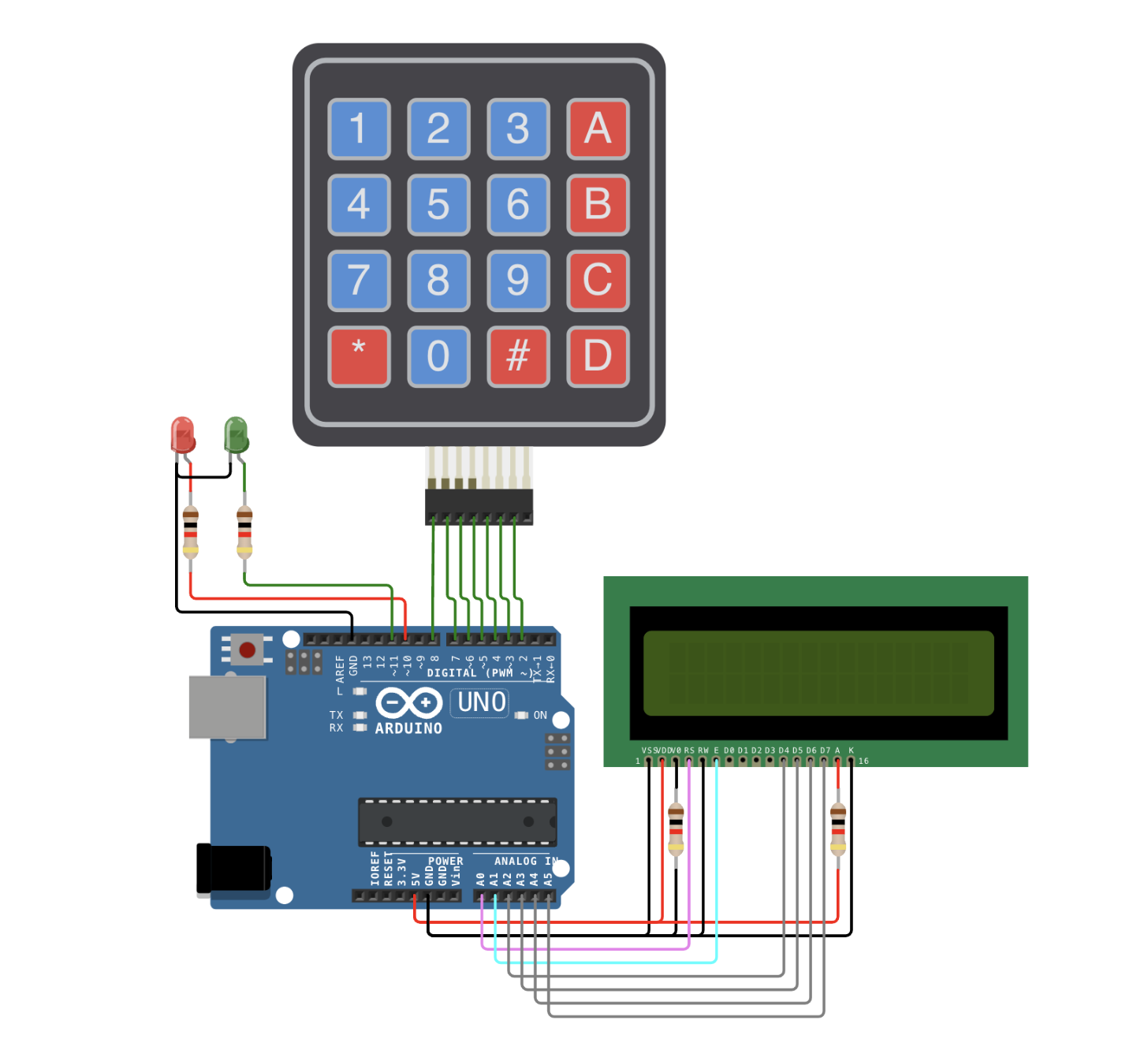
The Figure 1 depicted the UML program flow.



**Figure 1** Program schema.

## 3 Simulated or real assembled electrical schematic diagram:

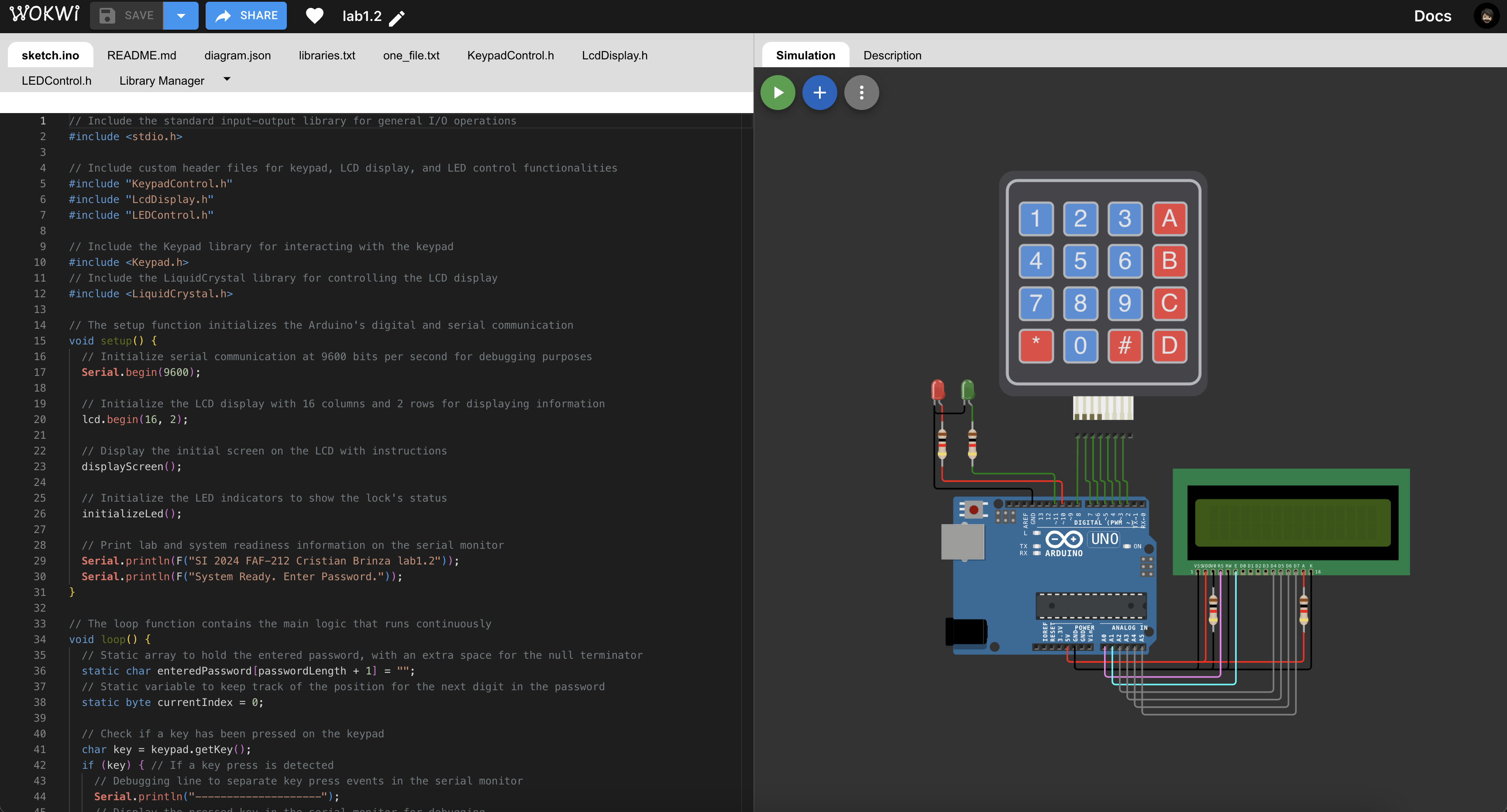
The Figure 2 depicted the phisical board schema.



**Figure 2** Phisical board schema.

## 4 Screenshots of the simulation execution:

The Figure 3 depicted a sceenshot of the WOKWI simulation :



**Figure 3** Screenshot of the simulation.

## CONCLUSION

The lab work was all about making an LED light turn on and off when we enter the corect code, which was really cool to see in action. We learned to write our code in a neat way that makes it easy to read and fix if something goes wrong. Also, figuring out how to make sure the button press didn't mess up because of tiny electrical glitches was a big win. We used an Arduino board, which was super helpful because there's a lot of help out there if we got stuck. This project was a great lesson in how to keep everything organized, especially when working with code

The task provided a practical experience in embedded system programming, emphasizing the importance of CamelCase coding conventions and modular code organization for peripheral device functionalities.

The laboratory work offered a comprehensive experience that extended beyond the mere toggling of an LED. It fostered an appreciation for the intricacies of embedded system design, from the meticulous organization of code to the thoughtful consideration of hardware-software interactions. The skills and insights gained from this project are invaluable, providing a strong foundation for future endeavors in the field of embedded systems.

## BIBLIOGRAPHY

1. EDUCBA: *Introduction to Embedded Systems.* Cursuri electronice online ©2020 [quote 10.02.2024] Available: <https://www.educba.com/what-is-embedded-systems/>
2. ARDUINO: *Arduino UNO.* The official website of Arduino modules, ©2020 [quote 10.02.2024]. Available: <https://www.arduino.cc/>.
3. TUTORIALSPOINT: *Embedded Systems Tutorial*. Comprehensive guide for beginners and professionals to learn Embedded System basics to advanced concepts, including microcontrollers, processors, and real-time operating systems. ©2023 [quote 10.02.2024] Available: <https://www.tutorialspoint.com/embedded_systems/index.htm>
4. GURU99: *Embedded Systems Tutorial: What is, History & Characteristics*. A detailed introduction to embedded systems, covering microcontrollers, microprocessors, and the architecture of embedded systems, as well as their applications and advantages. ©2023 [quote 10.02.2024] Available: <https://www.guru99.com/embedded-systems-tutorial.html>
5. JAVATPOINT: *Embedded Systems Tutorial.* Offers basic and advanced concepts of Embedded System, designed for beginners and professionals. ©2023 [quote 10.02.2024] Available: <https://www.javatpoint.com/embedded-systems-tutorial>
6. DEEPBLUE*: Embedded Systems Tutorials Introduction* | Embedded Systems Online Course. ©2023 [quote 10.02.2024] Available: <https://deepbluembedded.com/embedded-systems-tutorials/>

**APPENDIX 1**

Code of main.ino:

// Include the standard input-output library for general I/O operations

#include <stdio.h>

// Include custom header files for keypad, LCD display, and LED control functionalities

#include "KeypadControl.h"

#include "LcdDisplay.h"

#include "LEDControl.h"

// Include the Keypad library for interacting with the keypad

#include <Keypad.h>

// Include the LiquidCrystal library for controlling the LCD display

#include <LiquidCrystal.h>

// The setup function initializes the Arduino's digital and serial communication

void setup() {

// Initialize serial communication at 9600 bits per second for debugging purposes

Serial.begin(9600);

// Initialize the LCD display with 16 columns and 2 rows for displaying information

lcd.begin(16, 2);

// Display the initial screen on the LCD with instructions

displayScreen();

// Initialize the LED indicators to show the lock's status

initializeLed();

// Print lab and system readiness information on the serial monitor

Serial.println(F("SI 2024 FAF-212 Cristian Brinza lab1.2"));

Serial.println(F("System Ready. Enter Password."));

}

// The loop function contains the main logic that runs continuously

void loop() {

// Static array to hold the entered password, with an extra space for the null terminator

static char enteredPassword[passwordLength + 1] = "";

// Static variable to keep track of the position for the next digit in the password

static byte currentIndex = 0;

// Check if a key has been pressed on the keypad

char key = keypad.getKey();

if (key) { // If a key press is detected

// Debugging line to separate key press events in the serial monitor

Serial.println("--------------------");

// Display the pressed key in the serial monitor for debugging

Serial.print(F("Key Pressed: "));

Serial.println(key);

// Validate if the pressed key is a digit and within the password length

if (currentIndex < passwordLength && isdigit(key)) {

// Store the digit in the password array and increment the position

enteredPassword[currentIndex++] = key;

// Update the LCD display to show an asterisk for each entered digit

lcd.setCursor(9 + currentIndex - 1, 1);

lcd.print("\*");

// Echo the current state of entered password in the serial monitor

Serial.print(F("Entered: "));

Serial.print(enteredPassword);

Serial.println();

// Check if the password is fully entered and validate it

if (currentIndex == passwordLength) {

enteredPassword[currentIndex] = '\0'; // Null-terminate the password string

// Compare the entered password with the correct password

if (strcmp(enteredPassword, correctPassword) == 0) {

unlockDoor(); // Unlock the door if the password matches

} else {

incorrectCode(); // Show error and prompt again if the password is incorrect

}

// Clear the entered password and reset the index for a new attempt

for (int i = 0; i < passwordLength; i++) {

enteredPassword[i] = ' ';

}

currentIndex = 0;

displayScreen(); // Refresh the LCD display for the next password entry

}

} else if (key == '\*' && currentIndex > 0) { // Implement backspace functionality

currentIndex--; // Decrement the current index to "erase" the last digit

lcd.setCursor(9 + currentIndex, 1); // Adjust the LCD cursor for the correction

lcd.print(" "); // Replace the last asterisk with a space to show the digit removal

Serial.println(F("Backspace")); // Indicate a backspace operation in the serial monitor

}

}

}

// Function to display the initial LCD screen for password entry

void displayScreen() {

lcd.clear(); // Clear any previous text from the LCD

lcd.setCursor(0, 0); // Set the cursor to the beginning of the first line

lcd.print("Enter Password:"); // Prompt the user to enter the password

lcd.setCursor(9, 1); // Set the cursor to the start of the second line for password input

lcd.print("\_\_\_\_"); // Display underscores as placeholders for a 4-digit password

// Activate the red LED to indicate the system is locked and awaiting correct password

digitalWrite(redPinLock, HIGH); // Set the red LED pin to HIGH (LED on)

digitalWrite(greenPinUnlock, LOW); // Ensure the green LED is off by setting its pin to LOW

// The digitalWrite function changes the voltage of a specified pin. Setting a pin to HIGH (usually 5V) turns the LED on, while LOW (0V) turns it off.

}

// Function to handle the unlocking process once the correct password is entered

void unlockDoor() {

lcd.clear(); // Clear the LCD to display the access granted message

lcd.print("Access Granted"); // Inform the user that access has been granted

// Switch the LED statuses to indicate the door is now unlocked

digitalWrite(redPinLock, LOW); // Turn off the red LED

digitalWrite(greenPinUnlock, HIGH); // Turn on the green LED to indicate unlocked status

// Print a message on the serial monitor to indicate the door is unlocked

Serial.println("-----------------------------------");

Serial.println(F("Door Unlocked! Enjoy your access."));

Serial.println("-----------------------------------");

delay(5000); // Keep the door unlocked for 5 seconds

// After 5 seconds, re-lock the door and prompt for the password again

digitalWrite(redPinLock, HIGH); // Reactivate the red LED to show re-locking

digitalWrite(greenPinUnlock, LOW); // Turn off the green LED

lcd.clear(); // Clear the LCD for the next password entry

displayScreen(); // Display the password entry prompt again

Serial.println(F("Door re-locked. Please enter password again."));

}

// Function to handle incorrect password attempts

void incorrectCode() {

lcd.clear(); // Clear the LCD to show the access denied message

lcd.print("Access Denied"); // Inform the user that the entered password is incorrect

// Keep the red LED on to indicate the door remains locked

digitalWrite(redPinLock, HIGH); // Ensure the red LED remains activated

digitalWrite(greenPinUnlock, LOW); // Ensure the green LED remains off

// Print a message on the serial monitor indicating the incorrect password attempt

Serial.println("------------------------------");

Serial.println("Incorrect Password! Try again.");

Serial.println("------------------------------");

delay(3000); // Pause to allow the user time to read the message before trying again

}

Code of KeypadControl.h:

// Include the standard input-output library for general I/O operations

#include <stdio.h>

// Prevent multiple inclusions of this header file

#ifndef KEYPADCONTROL\_H

#define KEYPADCONTROL\_H

// Include the Keypad library for keypad interfacing

#include <Keypad.h>

// Define Arduino pins connected to the keypad rows

#define ROW\_PIN\_1 8

#define ROW\_PIN\_2 7

#define ROW\_PIN\_3 6

#define ROW\_PIN\_4 5

// Define Arduino pins connected to the keypad columns

#define COL\_PIN\_1 4

#define COL\_PIN\_2 3

#define COL\_PIN\_3 2

#define COL\_PIN\_4 1

// Define the number of rows and columns on the keypad

const byte numRows = 4; // The keypad has 4 rows

const byte numCols = 4; // The keypad has 4 columns

// Define the correct password and calculate its length

const char correctPassword[] = "4567"; // Set the correct password

const byte passwordLength = sizeof(correctPassword) - 1; // Calculate the length of the correct password excluding the null terminator

// Define the layout of the keypad

// Each character represents a button on the keypad

char keys[numRows][numCols] = {

{'1', '2', '3', 'A'}, // First row of buttons

{'4', '5', '6', 'B'}, // Second row

{'7', '8', '9', 'C'}, // Third row

{'\*', '0', '#', 'D'} // Fourth row

};

// Specify the Arduino pins connected to the keypad rows

byte rowPins[numRows] = {ROW\_PIN\_1, ROW\_PIN\_2, ROW\_PIN\_3, ROW\_PIN\_4}; // Connect keypad rows to these pins on Arduino

// Specify the Arduino pins connected to the keypad columns

byte colPins[numCols] = {COL\_PIN\_1, COL\_PIN\_2, COL\_PIN\_3, COL\_PIN\_4}; // Connect keypad columns to these pins on Arduino

// Initialize the keypad by passing the keymap, row pins, column pins, and the number of rows and columns

// This creates a Keypad object that can be used to detect key presses

Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, numRows, numCols);

// End of the condition to prevent multiple inclusions

#endif

Code of LcdDisplay.h:

// Include the standard input-output library for general I/O operations

#include <stdio.h>

// Use preprocessor directive to ensure the content of this file is included only once

#ifndef LCDDISPLAY\_H

#define LCDDISPLAY\_H

// Include the LiquidCrystal library to control LCD displays

#include <LiquidCrystal.h>

// Define constants for the LCD's pin connections to the Arduino

#define LCD\_RS A0 // Define RS (Register Select) pin connected to Arduino's analog pin A0

#define LCD\_EN A1 // Define EN (Enable) pin connected to Arduino's analog pin A1

#define LCD\_D4 A2 // Define D4 pin connected to Arduino's analog pin A2

#define LCD\_D5 A3 // Define D5 pin connected to Arduino's analog pin A3

#define LCD\_D6 A4 // Define D6 pin connected to Arduino's analog pin A4

#define LCD\_D7 A5 // Define D7 pin connected to Arduino's analog pin A5

// Initialize an instance of the LiquidCrystal class with the defined pins

LiquidCrystal lcd(LCD\_RS, LCD\_EN, LCD\_D4, LCD\_D5, LCD\_D6, LCD\_D7);

// This setup configures the LCD to use a 4-bit communication mode with the specified pins.

// Using analog pins for digital communication is a common practice in Arduino projects to

// increase the number of available digital pins, especially on boards with limited GPIO pins.

#endif // End of the preprocessor directive to prevent multiple inclusions of this file

Code of LEDControl.h:

// Include the standard input-output library for general I/O operations

#include <stdio.h>

// Prevent multiple inclusions of this header file using preprocessor directives

#ifndef LEDCONTROL\_H

#define LEDCONTROL\_H

// Define macro constants for the LED pins connected to the Arduino

// These constants improve code readability and maintainability

#define redPinLock 10 // Assign the pin number 10 to the red LED used for indicating the lock status

#define greenPinUnlock 11 // Assign the pin number 11 to the green LED used for indicating the unlock status

// Function to initialize the LEDs used for lock status indication

void initializeLed() {

// Set the pin mode of the red LED to OUTPUT, allowing it to be turned on or off

pinMode(redPinLock, OUTPUT);

// Similarly, set the pin mode of the green LED to OUTPUT

pinMode(greenPinUnlock, OUTPUT);

// The pinMode function configures the specified digital I/O pin as either INPUT or OUTPUT.

// Here, configuring the pins as OUTPUT enables the Arduino to control the LEDs by sending voltage signals,

// turning them on (HIGH) or off (LOW) as needed to indicate the system's lock or unlock status.

}

// End of the conditional inclusion directive

#endif