TECHNICAL UNIVERSITY OF MOLDOVA
FACULTY OF COMPUTERS, INFORMATICS AND MICROELECTRONICS
DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATICS

Report of laboratory work №1

Theme: User interactions

Fulfilled: st.gr. FAF-202 **Pogorevici Daniel**

Controlled: univ. lecturer **Moraru Dumitru**

Chișinău 2023

THE TASKS OF THE LABORATORY WORK

Task 1: To design an MCU-based application that would change the status of an LED when a button press is detected.

Task 2: To design an MCU-based application that would receive commands from the terminal via the serial interface to set the status of an LED.

- led on for switching on and led off for switching off…, the system must respond with text messages about the order confirmation
- STDIO library for terminal text exchange has to be used

Task 3: To design an MCU-based application that would detect code from a 4x4 keyboard, verify it, and display a message on an LCD.

- for valid code to light a green LED, for invalid code, a red LED
- STDIO library to scan the keyboard and display information on the LCD must be used

Implementation

1. **Task 1**

Description:

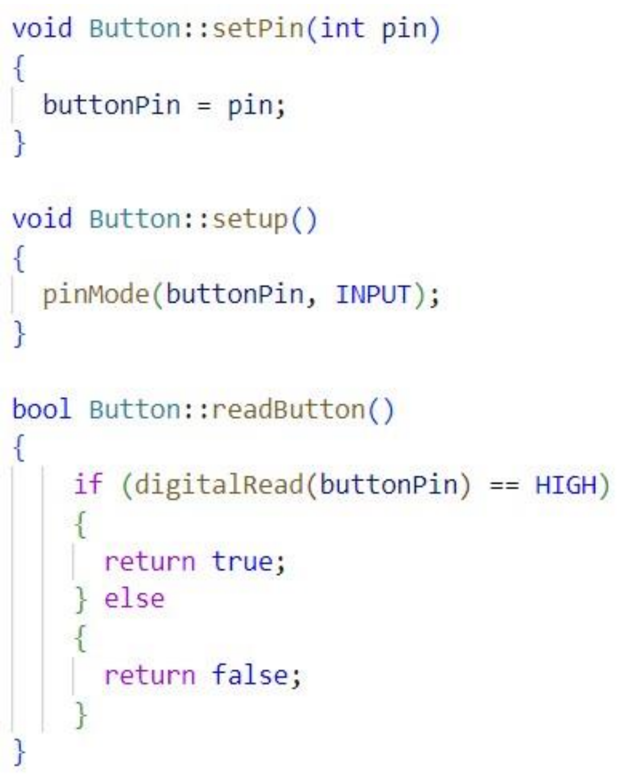
To design an MCU-based application that would change the status of an LED when a button press is detected.

Solution:

For the following task I created 2 packages for the separate components, “Button” and “Led”.

The first component, “Button” (appendix 1) having a few basic functions which are “readButton()” that is used in order to check if the button is pressed or not. In the first case if we press the button, it return the True value, otherwise it returns False. Another functions for the button being the “setPin()” and “setup()” which are used to set the pin of the button and set the pin mode.

Appendix 1

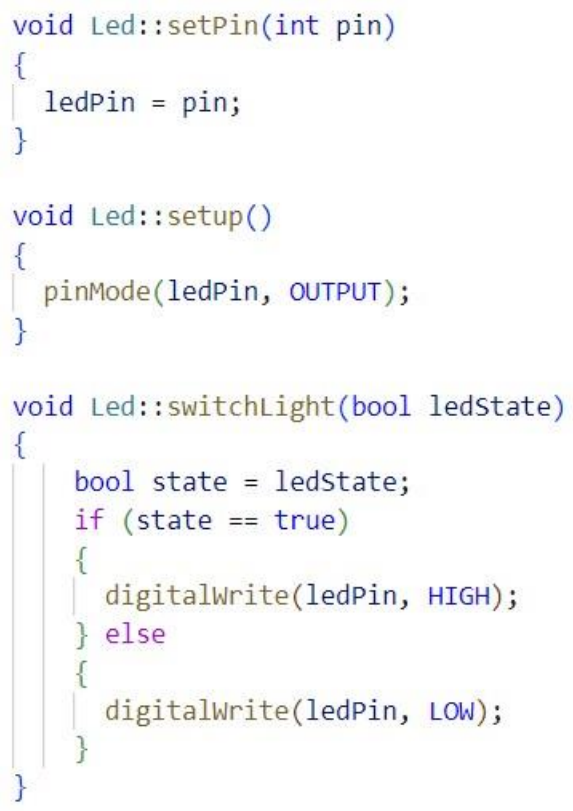


```
void Button::setPin(int pin)
{
  buttonPin = pin;
}

void Button::setup()
{
  pinMode(buttonPin, INPUT);
}

bool Button::readButton()
{
    if (digitalRead(buttonPin) == HIGH)
    {
      return true;
    } else
    {
      return false;
    }
}
```

Fig. 1.1 Button.cpp – button component.

The second component is “Led” (appendix 2) which consists of the same functions “setup()” and “setPin” as for the button, and also the “switchLight” which takes the parameter and turns the led on and off.

Appendix 2

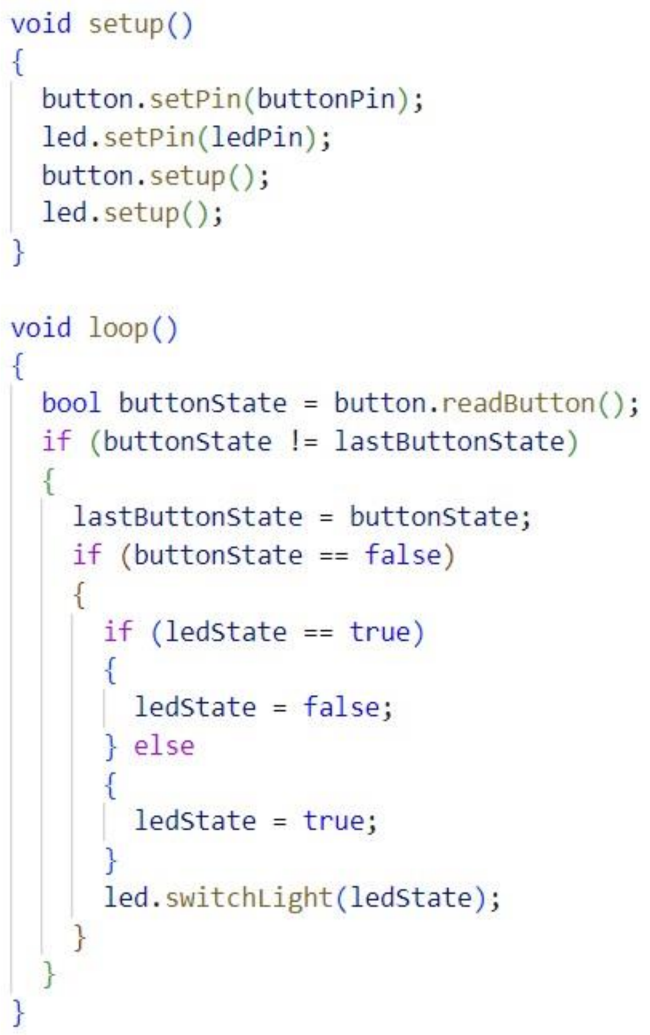


```cpp
void Led::setPin(int pin)
{
  ledPin = pin;
}

void Led::setup()
{
  pinMode(ledPin, OUTPUT);
}

void Led::switchLight(bool ledState)
{
    bool state = ledState;
    if (state == true)
    {
      digitalWrite(ledPin, HIGH);
    } else
    {
      digitalWrite(ledPin, LOW);
    }
}
```

Fig. 1.2 Led.cpp – led component.

The last part of the first task being the main file (appendix 3) which performs the objects creation and runs the loop to switch the led by clicking the button.

Appendix 3



```
void setup()
{
  button.setPin(buttonPin);
  led.setPin(ledPin);
  button.setup();
  led.setup();
}

void loop()
{
  bool buttonState = button.readButton();
  if (buttonState != lastButtonState)
  {
    lastButtonState = buttonState;
    if (buttonState == false)
    {
      if (ledState == true)
      {
        ledState = false;
      } else
      {
        ledState = true;
      }
      led.switchLight(ledState);
    }
  }
}
```

Fig. 1.3 Main.cpp – main file.

2. Block diagram for the Task 1.

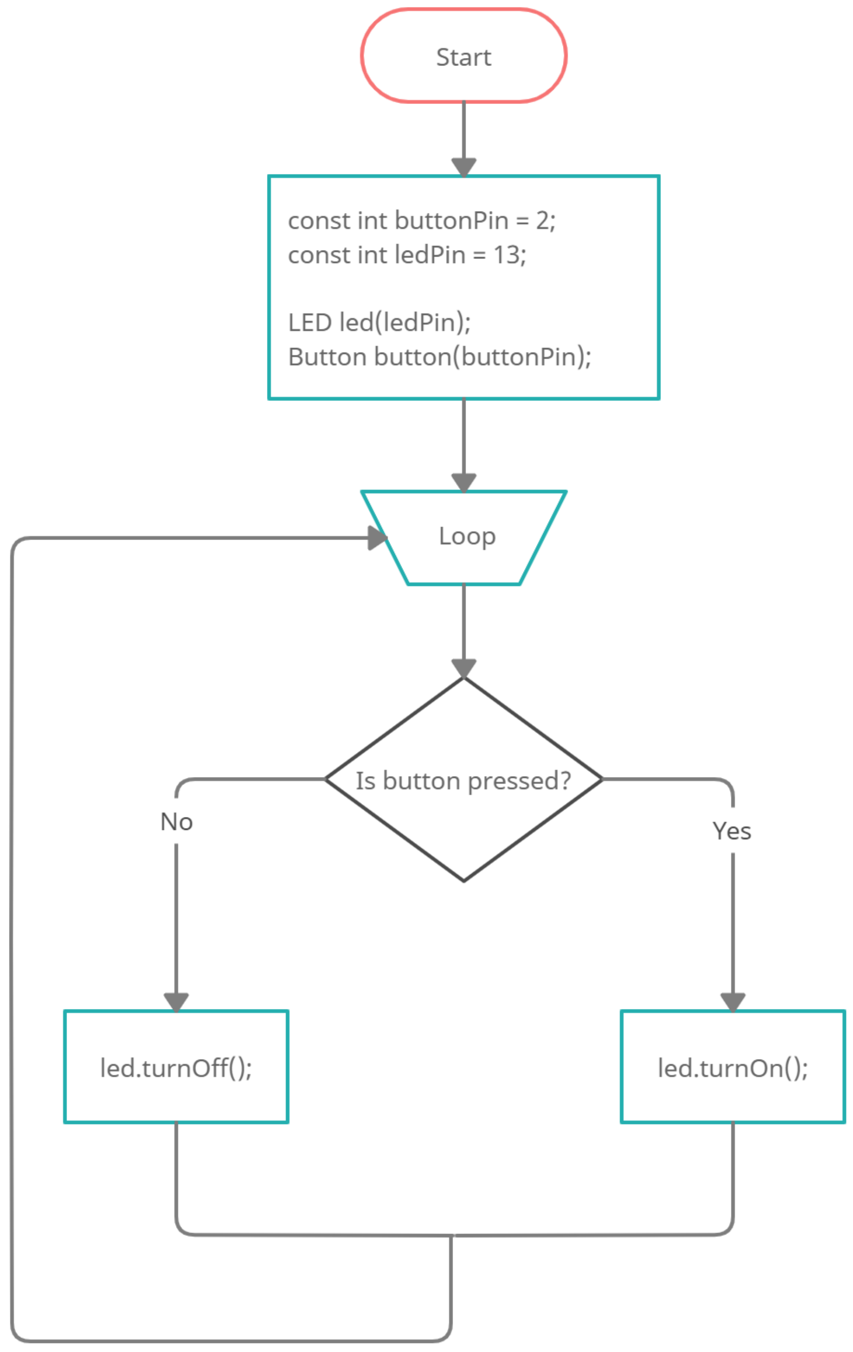


Figure 1.4 – block diagram.

3. **Electrical Scheme**

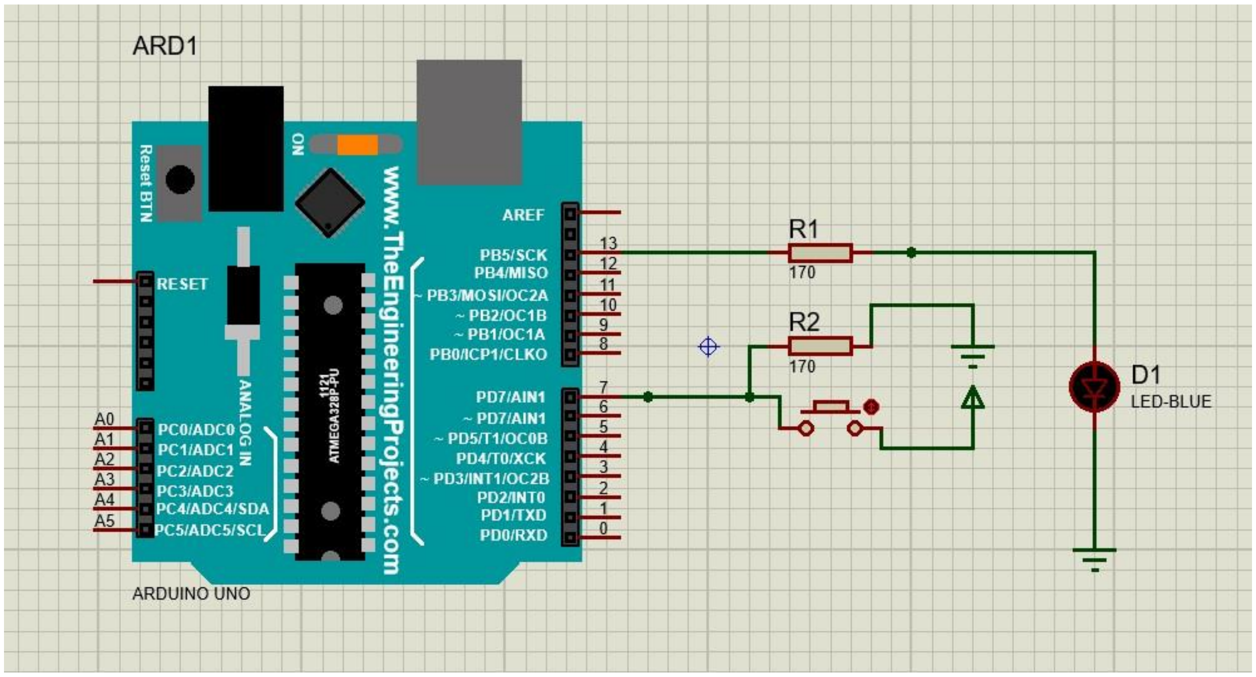


Figure 1.5 – Electrical Scheme.

2. Task 1

Description:

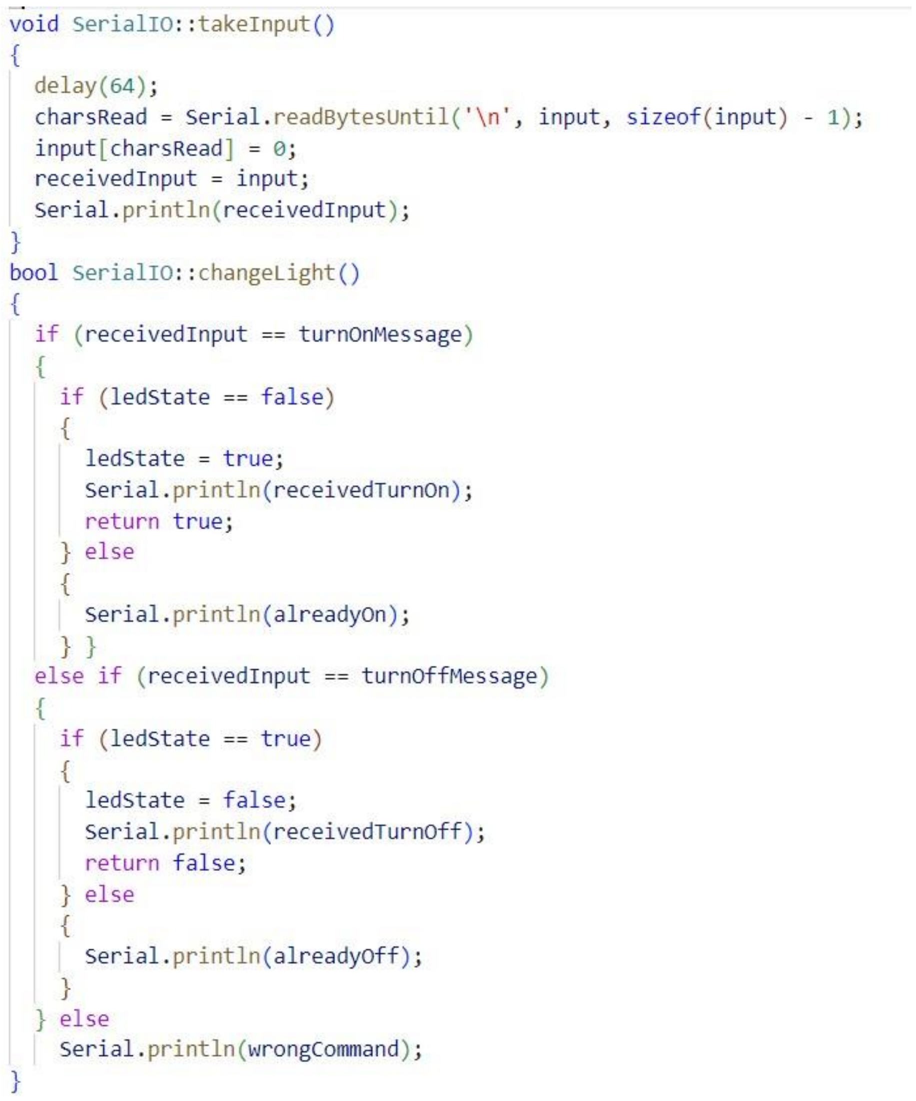
To design an MCU-based application that would receive commands from the terminal via the serial interface to set the status of an LED.

Solution:

For the following task I make usage of 2 components, “Led” and “SerialIO”. The first component being re-used from the previous task.

“SerialIO” (appendix 4) is the new class which I use to handle the serial input, to give commands to the led, and to output messages about the led state. It consists of some functions, “sets()” for setting the interface, “waitInput()” waits for the input from the interface, “takeInput()” takes the input and “changeLight()”.

Appendix 4



```
void SerialIO::takeInput()
{
  delay(64);
  charsRead = Serial.readBytesUntil('\n', input, sizeof(input) - 1);
  input[charsRead] = 0;
  receivedInput = input;
  Serial.println(receivedInput);
}
bool SerialIO::changeLight()
{
  if (receivedInput == turnOnMessage)
  {
    if (ledState == false)
    {
      ledState = true;
      Serial.println(receivedTurnOn);
      return true;
    } else
    {
      Serial.println(alreadyOn);
    } }
  else if (receivedInput == turnOffMessage)
  {
    if (ledState == true)
    {
      ledState = false;
      Serial.println(receivedTurnOff);
      return false;
    } else
    {
      Serial.println(alreadyOff);
    }
  } else
    Serial.println(wrongCommand);
}
```

The last one being the main file “main.cpp” (appendix 5) which creates the objects, and runs the loop to switch the led by the return value of the “changeLight()” function.

Appendix 5

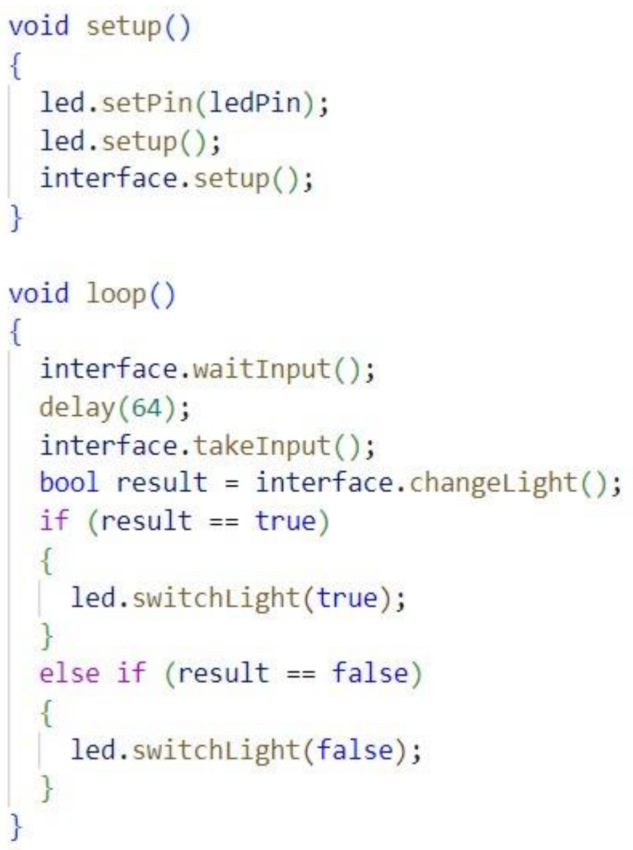
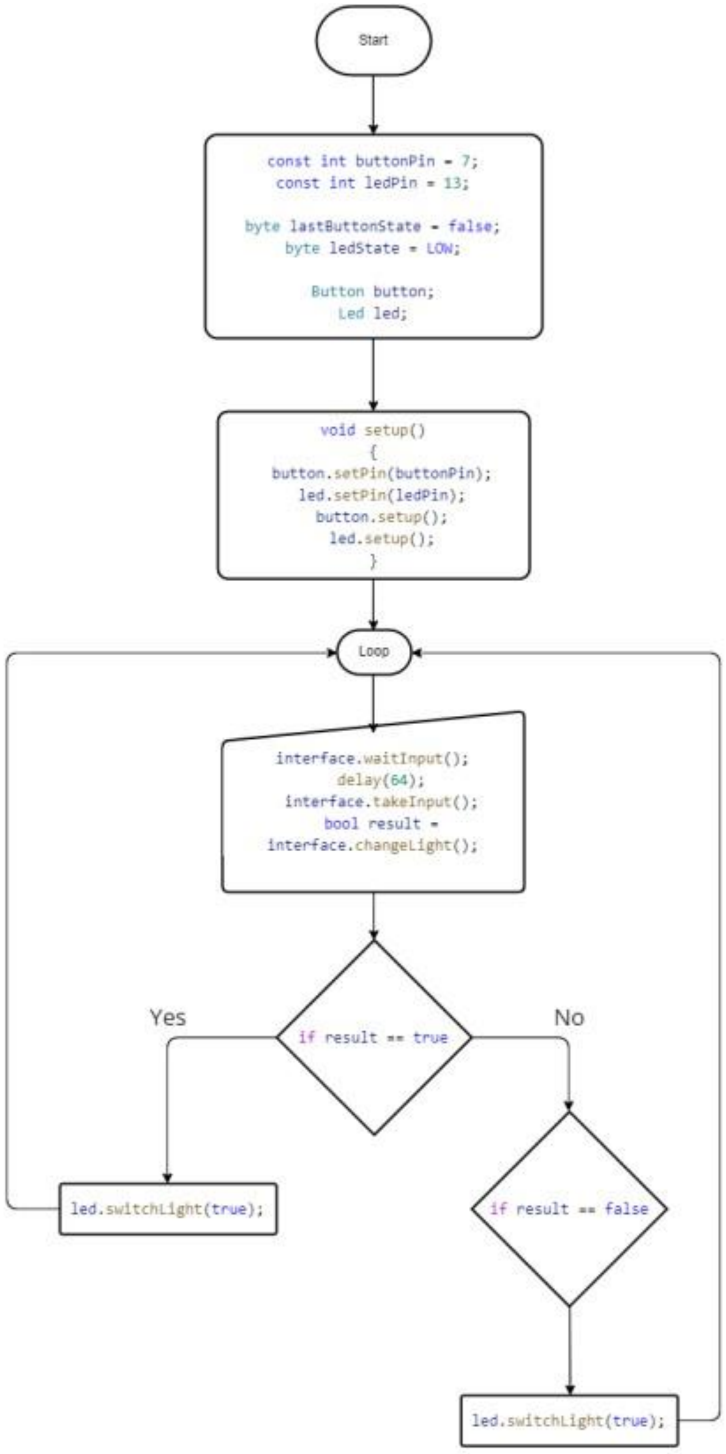

```
void setup()
{
  led.setPin(ledPin);
  led.setup();
  interface.setup();
}

void loop()
{
  interface.waitInput();
  delay(64);
  interface.takeInput();
  bool result = interface.changeLight();
  if (result == true)
  {
    led.switchLight(true);
  }
  else if (result == false)
  {
    led.switchLight(false);
  }
}
```

Fig. 2.1 – main file

2. Block diagram for the Task 2.



```
const int buttonPin = 7;
const int ledPin = 13;

byte lastButtonState = false;
byte ledState = LOW;

Button button;
Led led;
```

```
void setup()
{
button.setPin(buttonPin);
led.setPin(ledPin);
button.setup();
led.setup();
}
```

```
interface.waitInput();
delay(64);
interface.takeInput();
bool result =
interface.changeLight();
```

```
if result == true
```

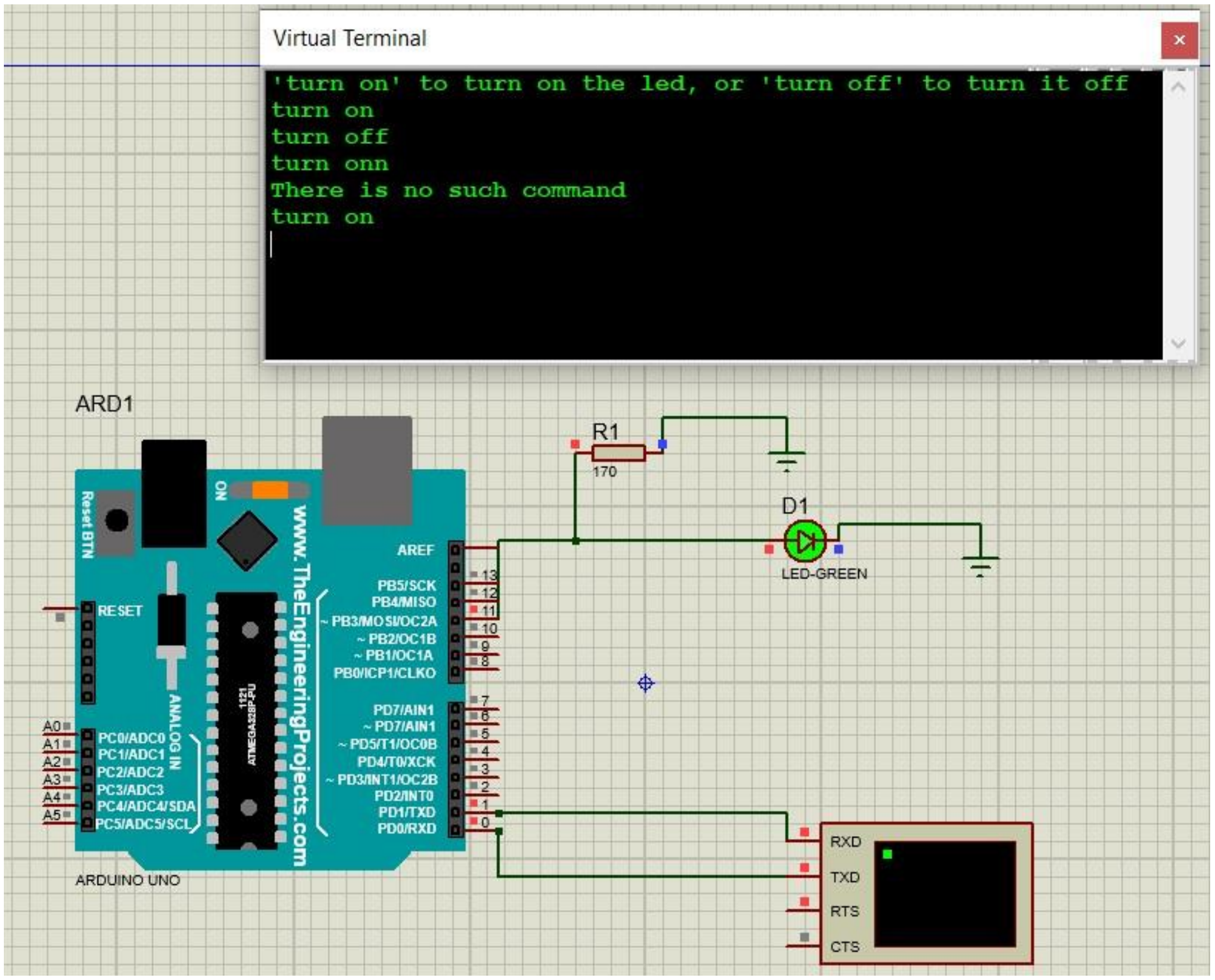
```
led.switchLight(true);
```

```
if result == false
```

```
led.switchLight(true);
```

Fig. 2.2 – block diagram

3. Electrical Scheme



```
'turn on' to turn on the led, or 'turn off' to turn it off
turn on
turn off
turn onn
There is no such command
turn on
```

Fig. 2.3 – electrical scheme.

3. Task 3

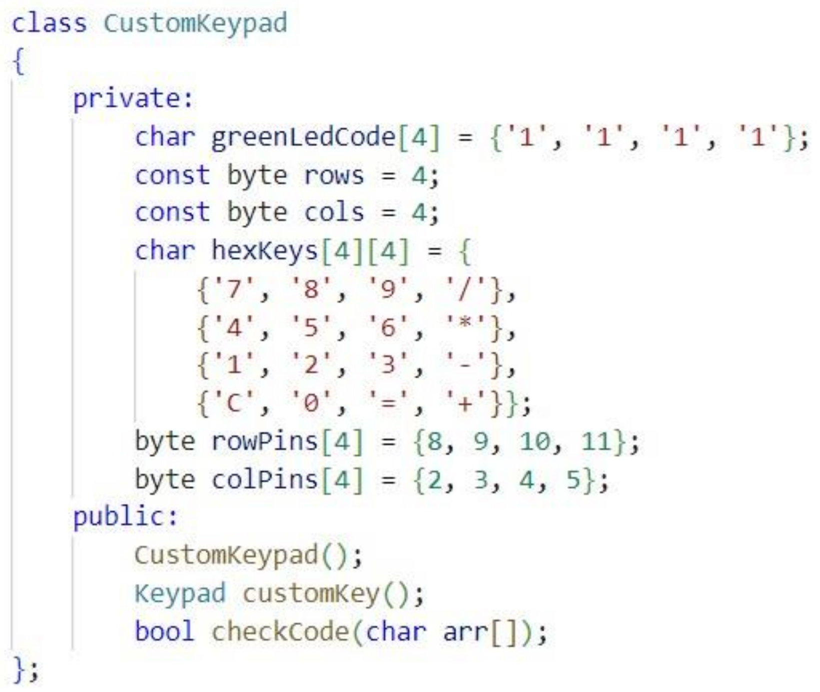
Description:

To design an MCU-based application that would detect code from a 4x4 keyboard, verify it, and display a message on an LCD.

Solution:

For the following task I use the “Led” and “SerialIO” classes from previous tasks. Beside them I am using an external library for keypad and also “customKeypad”(appendix 6) which helps to check the password matching and give commands to the led.

Appendix 6

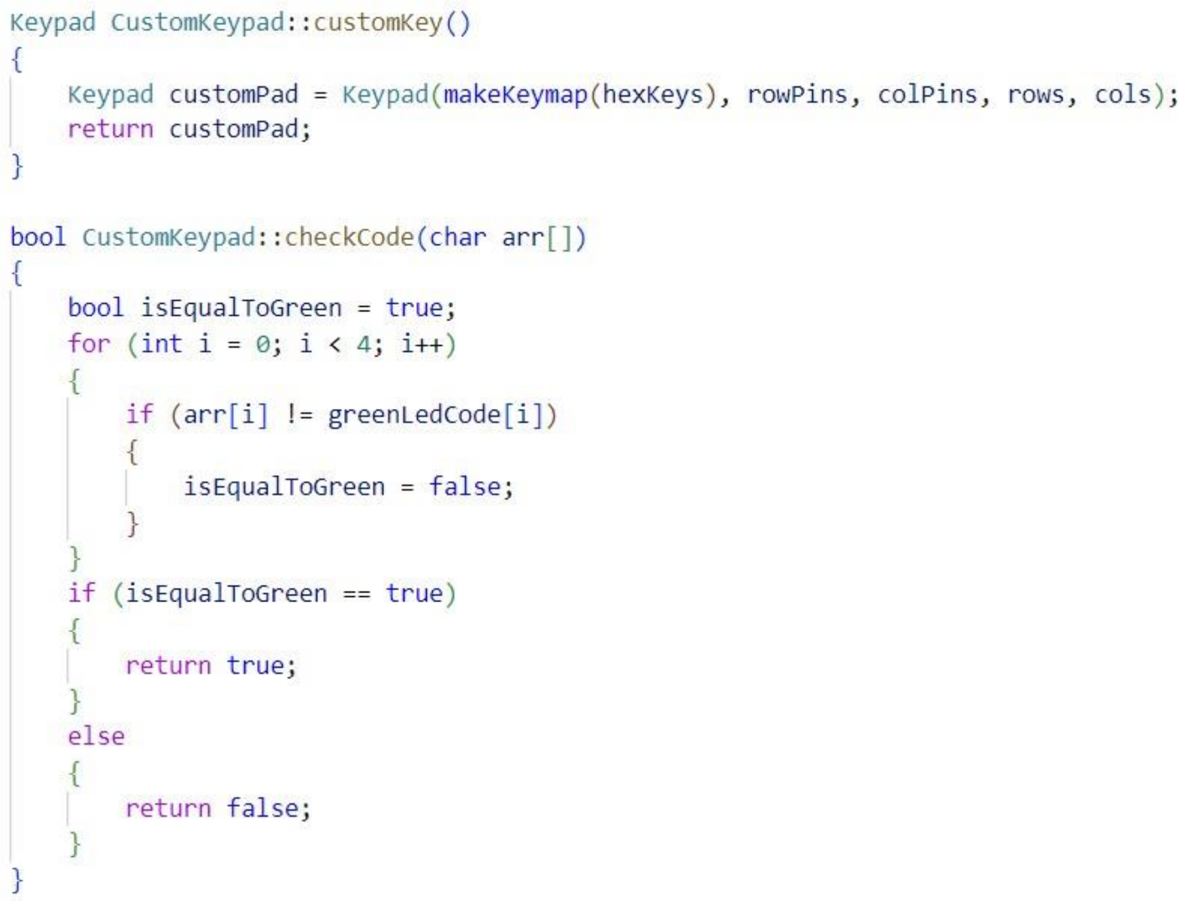


```
class CustomKeypad
{
    private:
        char greenLedCode[4] = {'1', '1', '1', '1'};
        const byte rows = 4;
        const byte cols = 4;
        char hexKeys[4][4] = {
            {'7', '8', '9', '/'},
            {'4', '5', '6', '*'},
            {'1', '2', '3', '-'},
            {'C', '0', '=', '+'}};
        byte rowPins[4] = {8, 9, 10, 11};
        byte colPins[4] = {2, 3, 4, 5};
    public:
        CustomKeypad();
        Keypad customKey();
        bool checkCode(char arr[]);
};
```

Fig. 3.1 CustomKeypad.h – custom keypad component.

Also I am having functions (appendix 7), “customKey()” which initializes the keyboard from library “Keypad.h” and “checkCode()” which is used to check the input array of numbers. It checks if the input numbers are equal to the secret code, and returns true if yes, otherwise false.

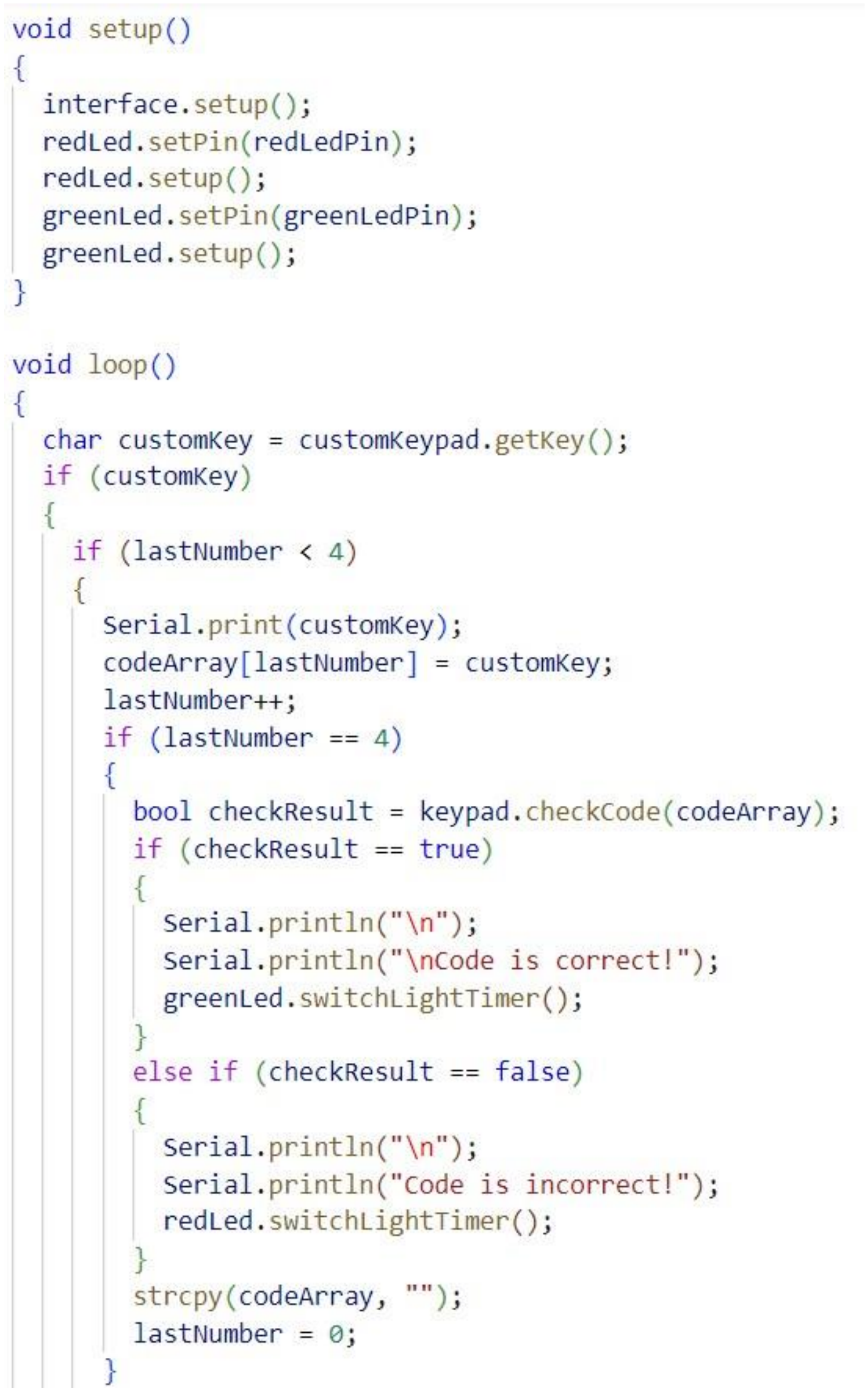
Appendix 7



```
Keypad CustomKeypad::customKey()
{
    Keypad customPad = Keypad(makeKeymap(hexKeys), rowPins, colPins, rows, cols);
    return customPad;
}

bool CustomKeypad::checkCode(char arr[])
{
    bool isEqualToGreen = true;
    for (int i = 0; i < 4; i++)
    {
        if (arr[i] != greenLedCode[i])
        {
            isEqualToGreen = false;
        }
    }
    if (isEqualToGreen == true)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

Fig. 3.2 CustomKeypad.cpp – custom keypad component.

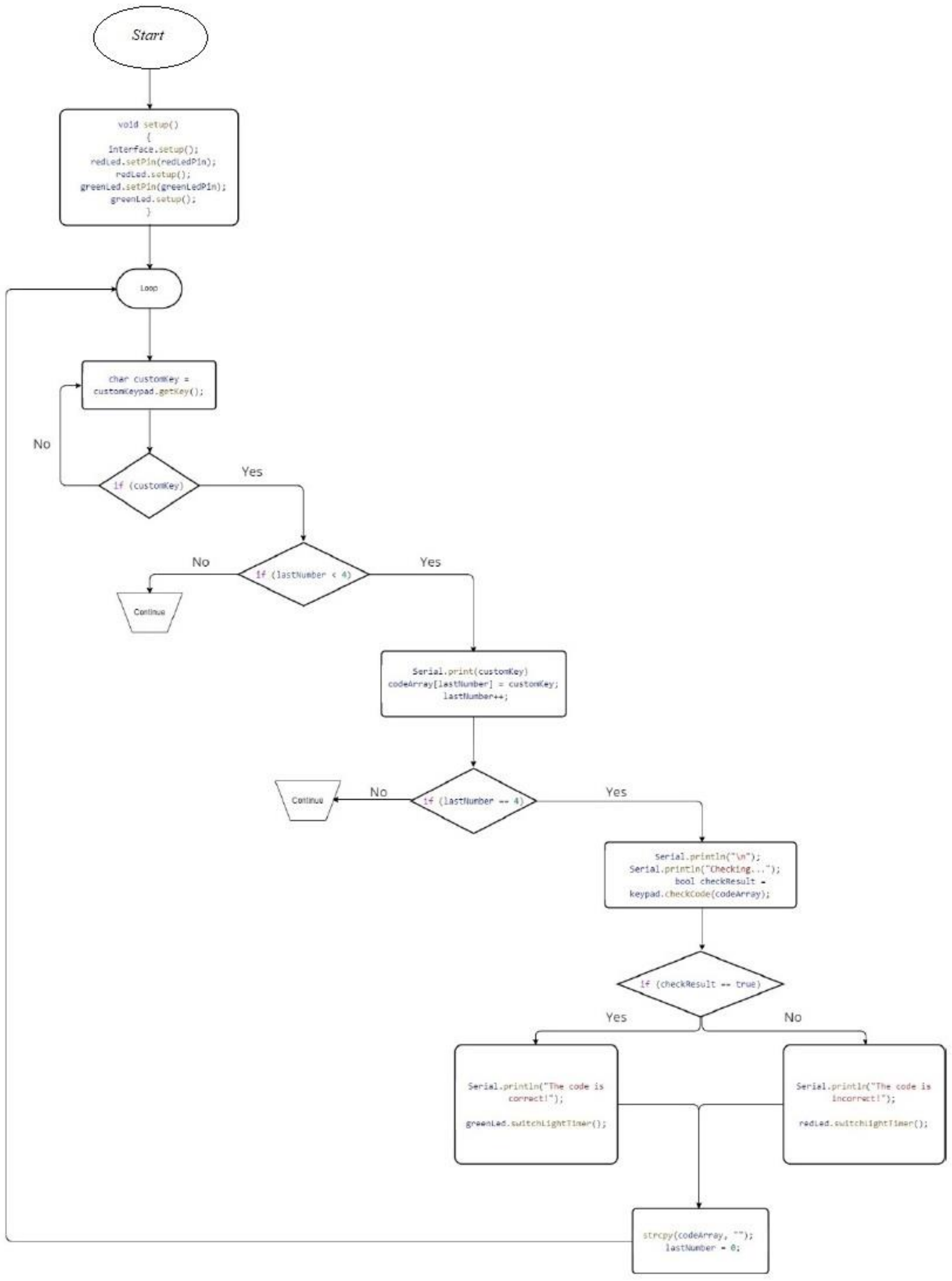
The last one being the main file (appendix 8), which creates the objects and runs the loop where the keyboard waits for the input, takes an array of for numbers and checks the code. If the code is correct, it lights the green led, otherwise the red one.



```
void setup()
{
  interface.setup();
  redLed.setPin(redLedPin);
  redLed.setup();
  greenLed.setPin(greenLedPin);
  greenLed.setup();
}

void loop()
{
  char customKey = customKeypad.getKey();
  if (customKey)
  {
    if (lastNumber < 4)
    {
      Serial.print(customKey);
      codeArray[lastNumber] = customKey;
      lastNumber++;
      if (lastNumber == 4)
      {
        bool checkResult = keypad.checkCode(codeArray);
        if (checkResult == true)
        {
          Serial.println("\n");
          Serial.println("\nCode is correct!");
          greenLed.switchLightTimer();
        }
        else if (checkResult == false)
        {
          Serial.println("\n");
          Serial.println("Code is incorrect!");
          redLed.switchLightTimer();
        }
        strcpy(codeArray, "");
        lastNumber = 0;
      }
```

Fig. 3.3 Main.cpp – main file.

2. Block diagram for the Task 3.



```
void setup()
{
Interface.setup();
redLed.setPin(redLedPin);
redLed.setup();
greenLed.setPin(greenLedPin);
greenLed.setup();
}
```

```
char customKey =
customKeypad.getKey();
```

```
if (customKey)
```

```
if (lastNumber < 4)
```

```
Serial.print(customKey)
codeArray[lastNumber] = customKey;
lastNumber++;
```

```
if (lastNumber == 4)
```

```
Serial.println("\n");
Serial.println("Checking...");
bool checkResult =
keypad.checkCode(codeArray);
```

```
if (checkResult == true)
```

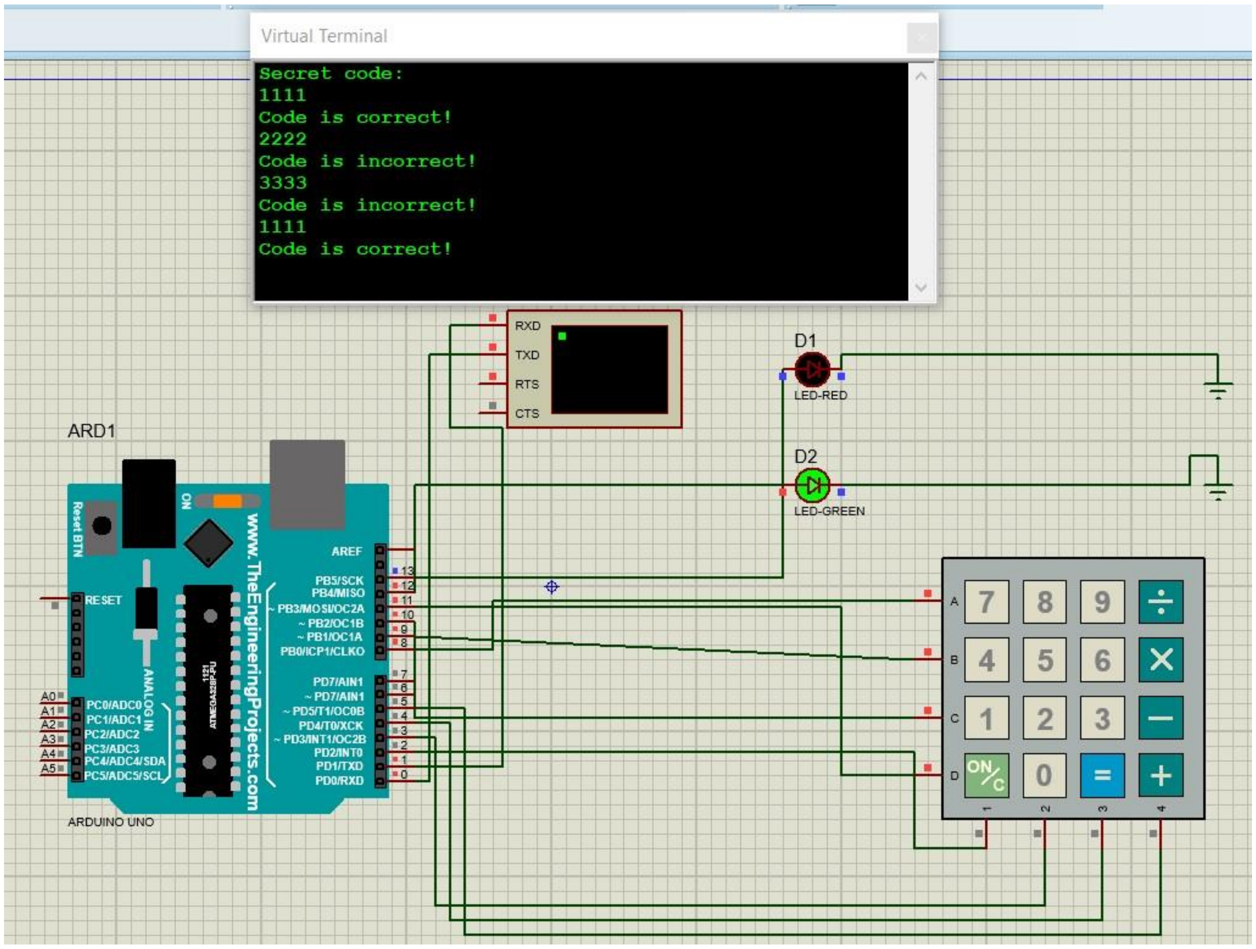
```
Serial.println("The code is
correct!");

greenLed.switchLightTimer();
```

```
Serial.println("The code is
incorrect!");

redLed.switchLightTimer();
```

```
strcpy(codeArray, "");
lastNumber = 0;
```

Fig. 3.4 – block diagram.

3. Electrical Scheme



```
Secret code:
1111
Code is correct!
2222
Code is incorrect!
3333
Code is incorrect!
1111
Code is correct!
```

Fig. 3.5 – electrical scheme.

Conclusions

In conclusion, this laboratory work involved designing MCU-based applications for different tasks. The first task involved designing an application to change the status of an LED when a button is pressed, while the second task focused on designing an application that can receive commands from a terminal via the serial interface to control the LED. The third task required the design of an application that can detect code from a 4x4 keyboard, verify it, and display a message on an LCD, with different LEDs indicating whether the code entered is valid or not.

In addition to providing practical experience in designing MCU-based applications, this laboratory work also allowed for the exploration of different aspects of embedded systems. The first task demonstrated the ability to control an LED with a button press, a simple yet essential task in many embedded systems. The second task further expanded on this concept, introducing the idea of serial communication between a terminal and the MCU to control the LED. This allowed for more flexibility and ease of use, especially in situations where physical interaction with the MCU is not possible. The third task introduced the use of a 4x4 keyboard for user input and an LCD for displaying information, highlighting the versatility of embedded systems in accepting input from a variety of sources and providing output in different forms. The use of different LEDs to indicate the validity of the code entered added an extra layer of feedback, making the system more user-friendly. Throughout this laboratory work, the importance of using libraries to simplify the programming process was emphasized. The use of the STDIO library for text exchange with the terminal, scanning the keyboard, and displaying information on the LCD greatly simplified the code and reduced the development time. In conclusion, this laboratory work provided valuable experience in designing MCU-based applications and demonstrated the versatility and practicality of embedded systems in various applications.