Vault design

Final report

Company name: Alcatraz

Team:
Catalin Chiprian
Nikolay Georgiev
Victoria Shvets (team leader)
Ali Mobini
Nikolay Todorov

ALCATRAZ
project

Table of contents

Table of contents **1**

Chapter 1 **2**
- User Requirements 2
- System testing procedure 2

Chapter 2 **3**
- High Level Design 3
- Software 3
- Hardware 4
- Integration testing procedure 4

Chapter 3 **17**
- Low Level Software 17
- Software unit testing procedure 21
- Hardware 28
- Choice of electronic components and their functioning explained 28
- Calculation of component values 29
- Technical details / explanation of schematic 30
- Low Level Hardware 30
- Electronic schematic drawing with CAD (Altium) 31
- Explanation of the schematic and relation with the block diagram 31
- Design of the PCB / layout with CAD (Multisim/Ultiboard, or Altium) 32
- Hardware Unit testing procedure 32

Chapter 4 **34**
- Test results 34

Appendix **35**
- Code 35
- Reference list 48
- Disclaimer 49

Chapter 1

User Requirements

Users should be able to securely store their belongings with the system without worrying about loss.

- The safe will have the ability to lock and unlock, keeping the user's belongings secure and out of the hands of unauthorized individuals.
- A screen that will allow the user to view the numbers he is writing.
- The safe will be encrypted with a three-digit combination.
- User input will receive both visual and auditory feedback.
- The user has the option to manually modify the code.

System testing procedure

The system test was conducted to determine whether the complete system (hardware and software) worked correctly according to user requirements. Firstly, for the locking mechanism Servo motor was used. For the lock to operate correctly, it was necessary to include a button for indication when the door was closed. To enter the password itself, rotary encoder and 7 segment displays were needed. During the test, all went according to the expected result: vault door closing, servo motor locks the door, and for safety the user needs to enter the password to open the door. As it was mentioned before, a 7 segment display was used for displaying code. Each of three 7 segment displays were able to print one digit for three-digit combination. To use 0-9 digits and to change the place for one position to another, a rotary encoder and rotary encoder button was integrated. During the test it was found that the rotary encoder can bounce and skip some numbers, it was fixed in the software. For the visual and auditory feedback two LEDs and buzzer were implemented. The tests were flawless: when the user entered the correct passcode, green LED lights up; and when password is incorrect red LED lights up; after three wrong tries, the buzzer and red LED turned on. By pressing an additional button, the code can be changed manually using rotary encoder. The system works according to user requirements and tested under different conditions; for example if power would be cut off after a second try, and then the arduino would be plugged in again user still would only have one try left.

Chapter 2

High Level Design

Software

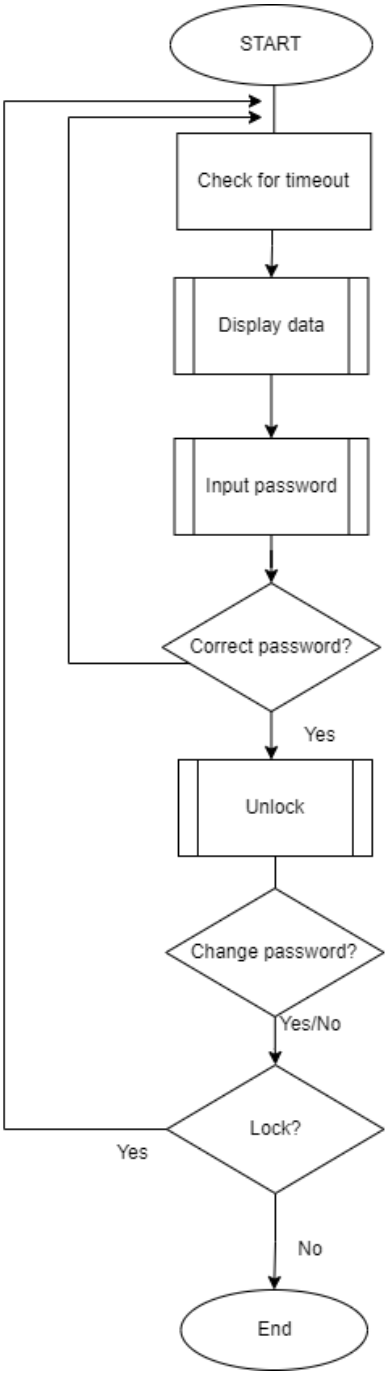

Figure 2.1 high level software flowchart

Hardware

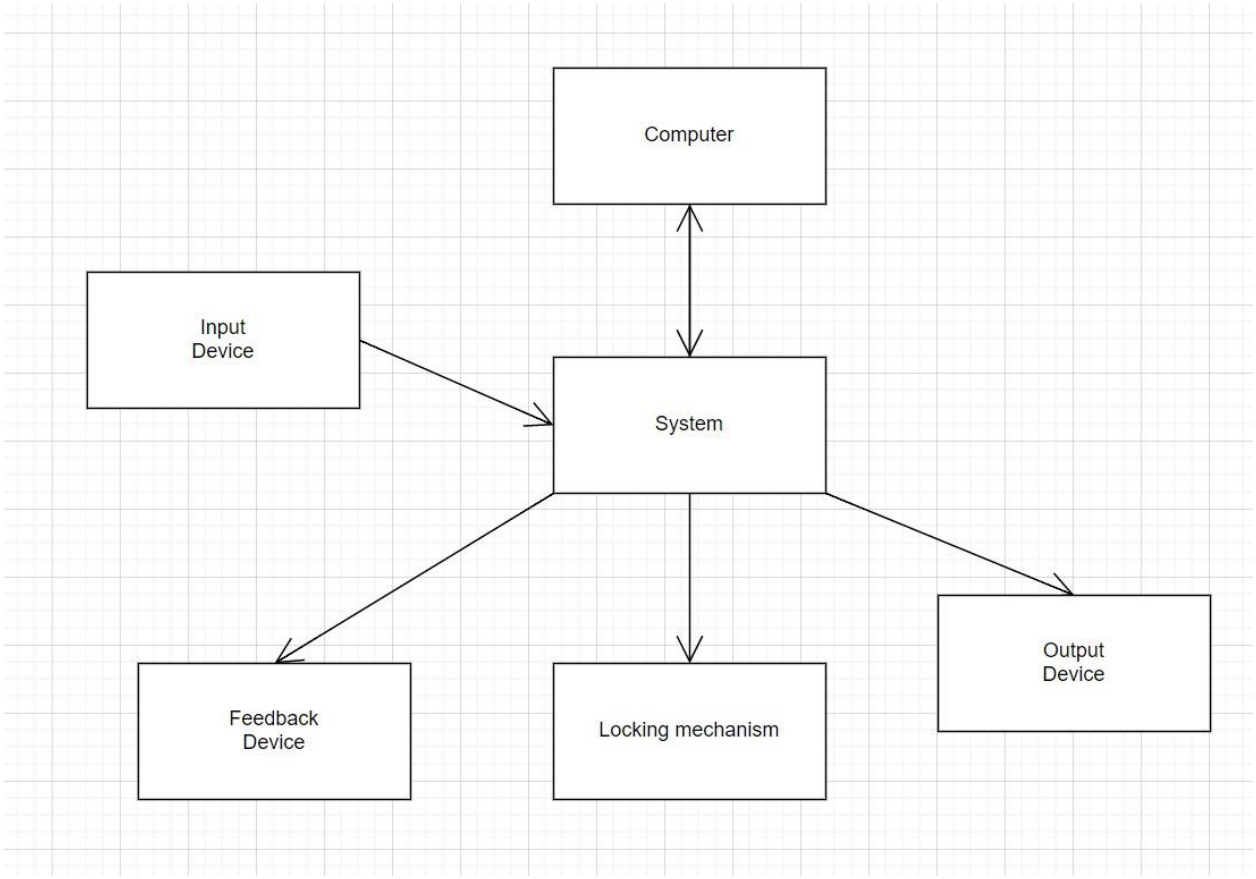

Figure 2.2 high level hardware flowchart

Personal belongings kept inside the safe can be locked and unlocked by the user. The password will be displayed on a device for output. A device will provide feedback to the user, letting him know if the password he entered matches. With this system, the user is assured that their possessions will remain secure and safe from unwanted intruders.

Integration testing procedure

Two 7 segments together. The expected result: to have numbers on both displays.

```
//start
#define DELAY 500
void setup()
{
  DDRB = 0b111111;
  DDRD = 0b1111111;
}
```

```
void loop()
{
  PORTD = 0b1111111;
  PORTB = 0b111001;
  delay(DELAY);
  PORTB = 0b100100;
  PORTD = 0b10111111;
  delay(DELAY);
  PORTB = 0b110000;
  delay(DELAY);
  PORTB = 0b011001;
  delay(DELAY);
  PORTB = 0b010010;
  delay(DELAY);
  PORTB = 0b000010;
  delay(DELAY);
  PORTB = 0b111000;
  PORTD = 0b11111111;
  delay(DELAY);
  PORTB = 0b000000;
  PORTD = 0b10111111;
  delay(DELAY);
  PORTB = 0b010000;
  delay(DELAY)
```

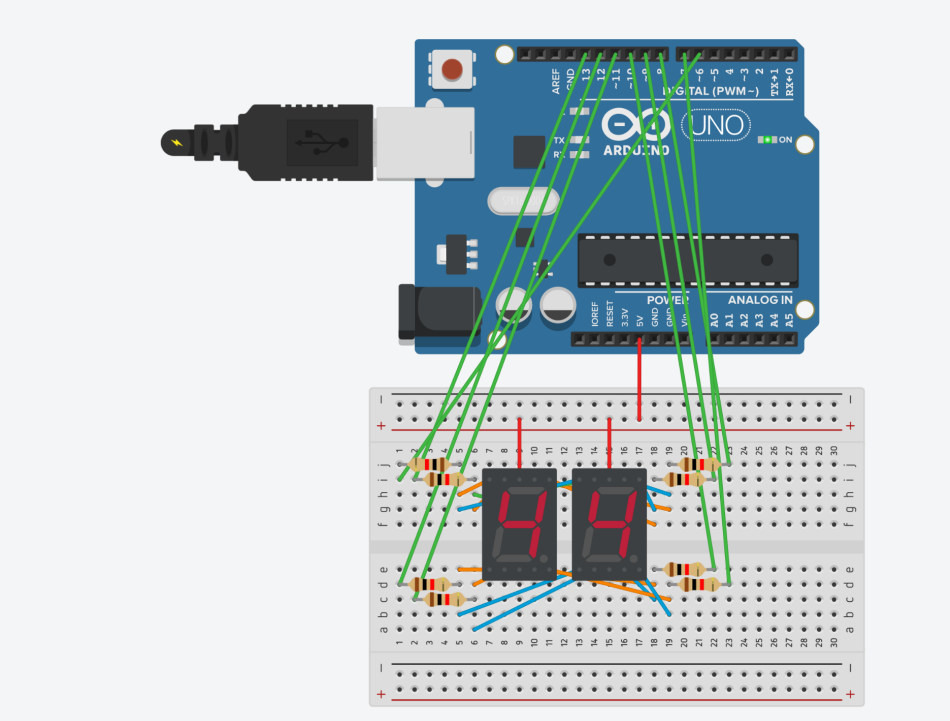

Figure 2.3 Two 7 segment displays connected

Three 7 segments together. The expected result: to have numbers on three displays. Previous code was used for the test.

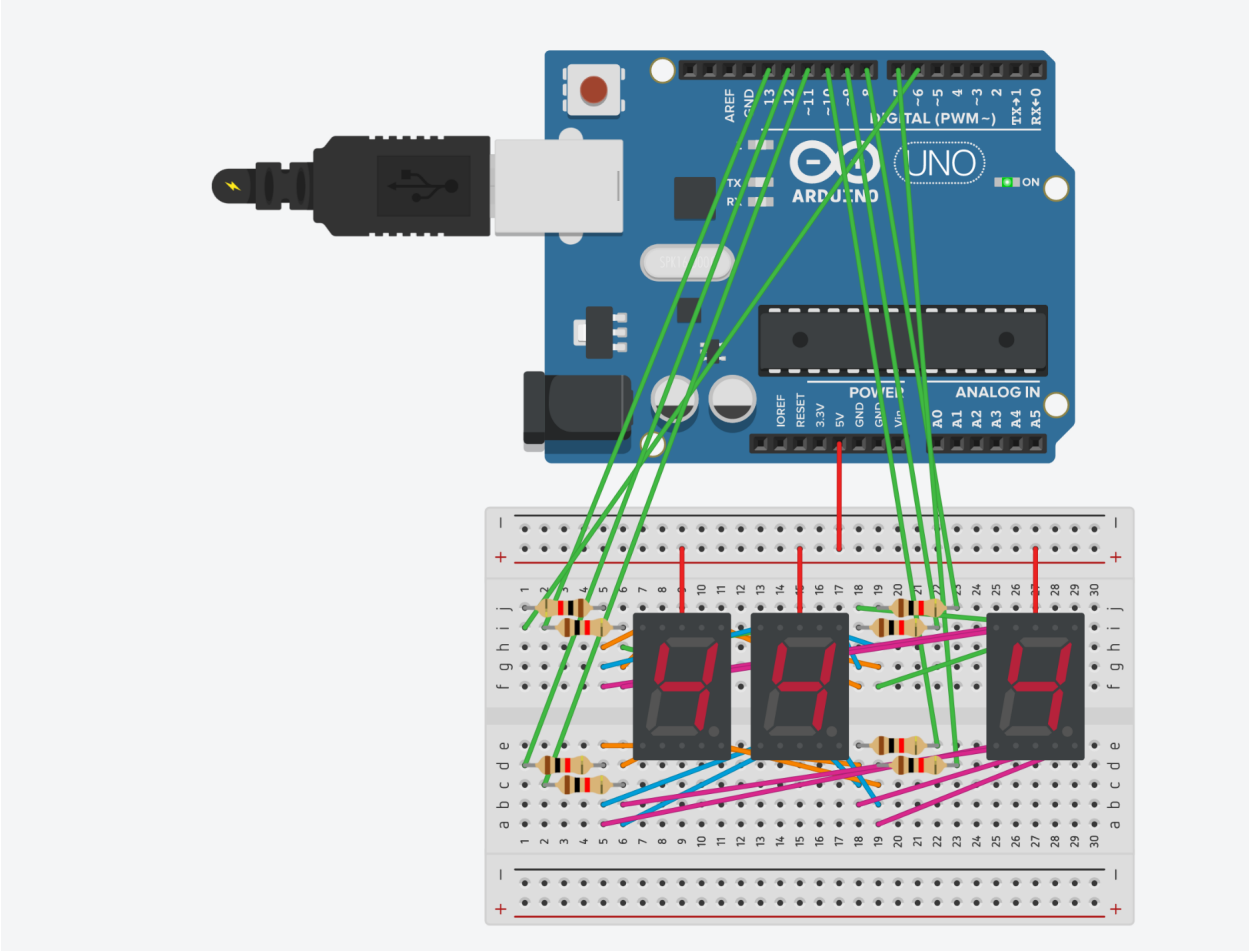

Figure 2.4 Three 7 segment displays connected

Two LEDs. The expected result: each LED lights up one at the time.

```
//start
const int RED = 13;
const int GREEN = 12;


void setup()
```

```
{
  pinMode(RED, OUTPUT);
  pinMode(GREEN, OUTPUT);
}

void loop()
{
  digitalWrite(RED, HIGH);
  delay(1000);
  digitalWrite(RED, LOW);
  delay(1000);
  digitalWrite(GREEN, HIGH);
  delay(1000);
  digitalWrite(GREEN, LOW);
  delay(1000);
}
```

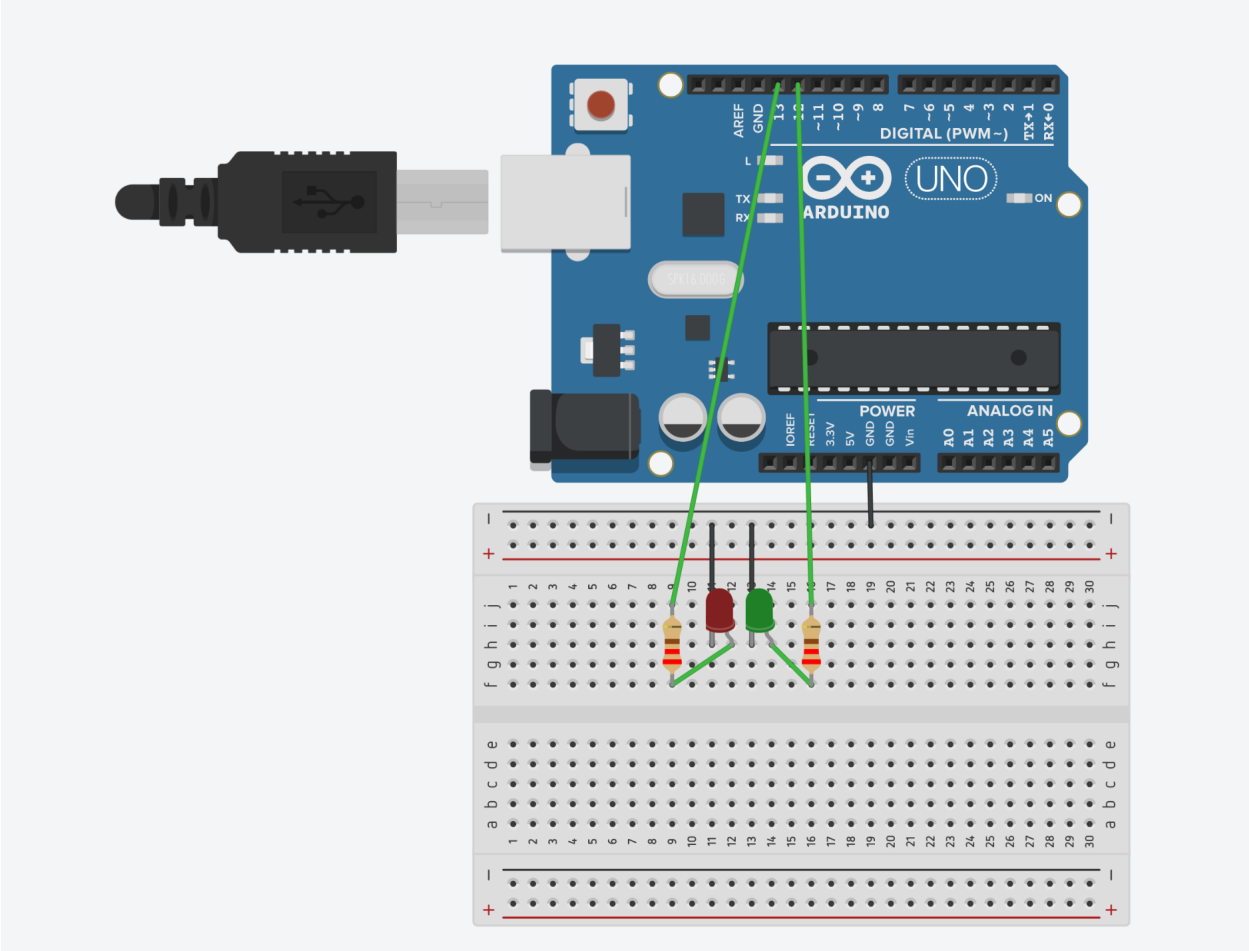

Figure 2.5 Two LEDs connected

Two LEDs and a buzzer. The expected result: each signal turns on one at the time.

```
//start
#define DELAY 1000
const int RED = 13;
const int GREEN = 12;
const int BUZZER = 10;


void setup()
{
  pinMode(RED, OUTPUT);
  pinMode(GREEN, OUTPUT);
  pinMode(BUZZER, OUTPUT);
}

void loop()
{
  digitalWrite(RED, HIGH);
  delay(DELAY);
  digitalWrite(RED, LOW);
  delay(DELAY);
  digitalWrite(BUZZER, HIGH);
  delay(DELAY);
  digitalWrite(BUZZER, LOW);
  delay(DELAY);
  digitalWrite(GREEN, HIGH);
  delay(DELAY);
  digitalWrite(GREEN, LOW);
  delay(DELAY);

}
```

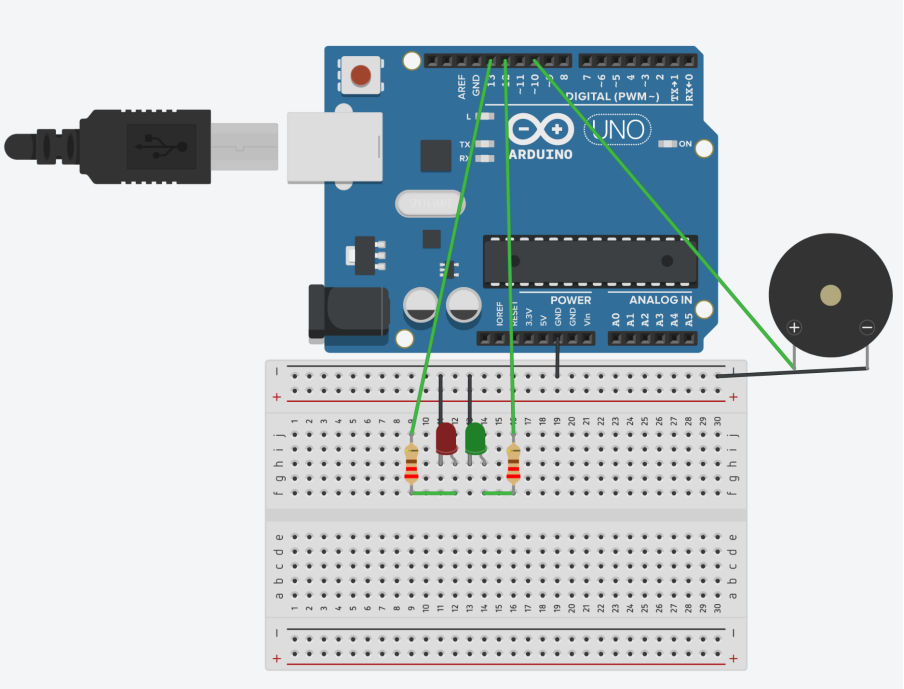

Figure 2.6 Two LEDs and buzzer connected

Servo and a button. The expected result: servo motor goes to position ‘closed’ when button is pressed

```
// start
//Servo goes to position 'open'

#include<Servo.h>

Servo Myservo;
int pos = 0;
void setup()
{
 pinMode(13,INPUT);
 Myservo.attach(9);
}

void loop()
{

 if(digitalRead(13)==LOW){
  Myservo.write(180);
 }
 else

Myservo.write(90);
//if button is pressed, the door is closed, but once it is released, servo is go to initial position

}
```

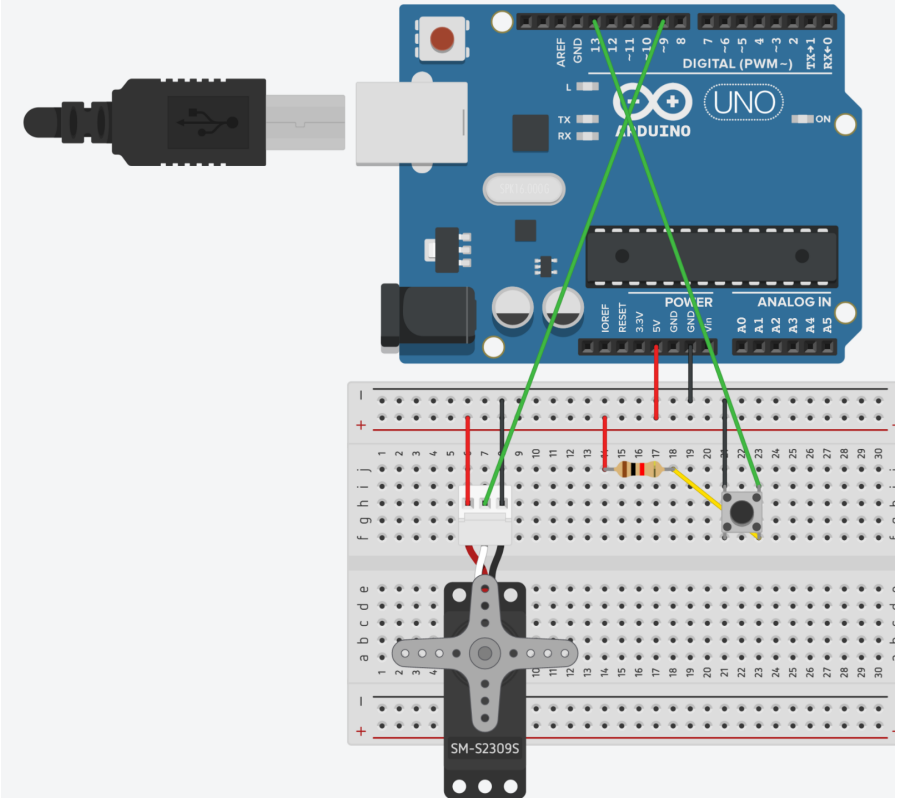

Figure 2.7 Servo motor and button connected

Servo and two buttons. The expected result: servo motor goes to position ‘closed’ when button is pressed and to position ‘open’ when other button is pressed. In the system the second button was implemented as a confirmation password button, and the servo motor only opens when the password is correct.

```
// start

#include<Servo.h>

Servo Myservo;
int pos = 0;
void setup()
{
 pinMode(13,INPUT);
 pinMode(12, INPUT);
 Myservo.attach(9);
}

void loop()
{
 if(digitalRead(13)==LOW){
        Myservo.write(180);
        //if button pressed, the door is closed
 }


 if(digitalRead(12)==LOW){
        Myservo.write(90);
 }
 // if button pressed, the door is open


}
```

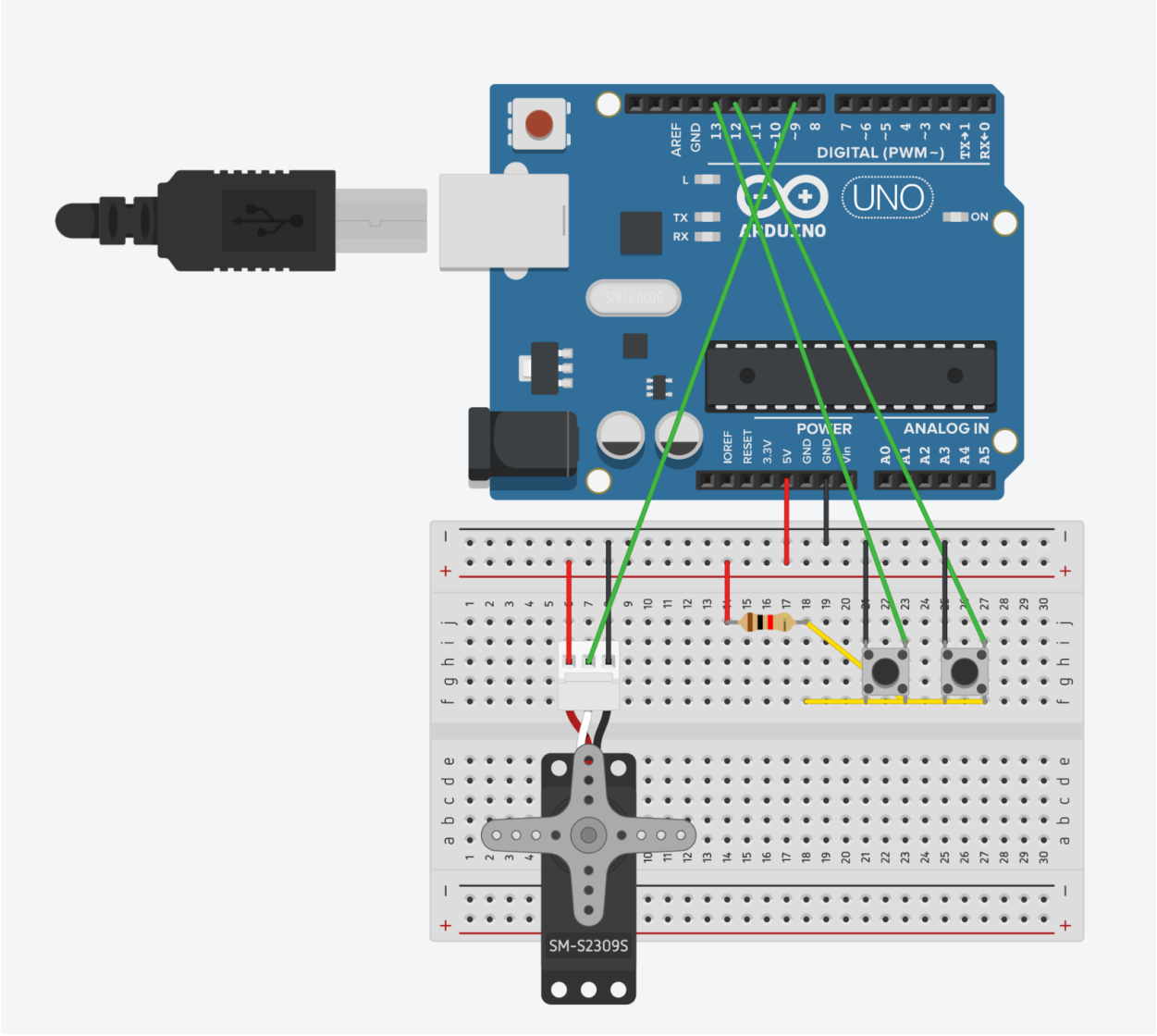

Figure 2.8 Two buttons and servo motor connected

Shift register and 7 segment display. The expected result: 7 segment display lights up.

```
const int latchPin = 10;
const int clockPin = 11;
const int dataPin = 12;
int datArray[10] = {B00000011, B10011111, B00100101, B00001101, B10011001, B01001001,
B01000001, B00011111, B00000001, B00001001};

void setup()
{
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
```

```
  pinMode(dataPin, OUTPUT);
}

void loop()
{
  int num;
  for (num = 0; num < 10; num++){

          digitalWrite(latchPin, LOW);
          shiftOut(dataPin, clockPin, MSBFIRST, datArray[num]);
          digitalWrite(latchPin, HIGH);

          delay(1000);

  }
}
```

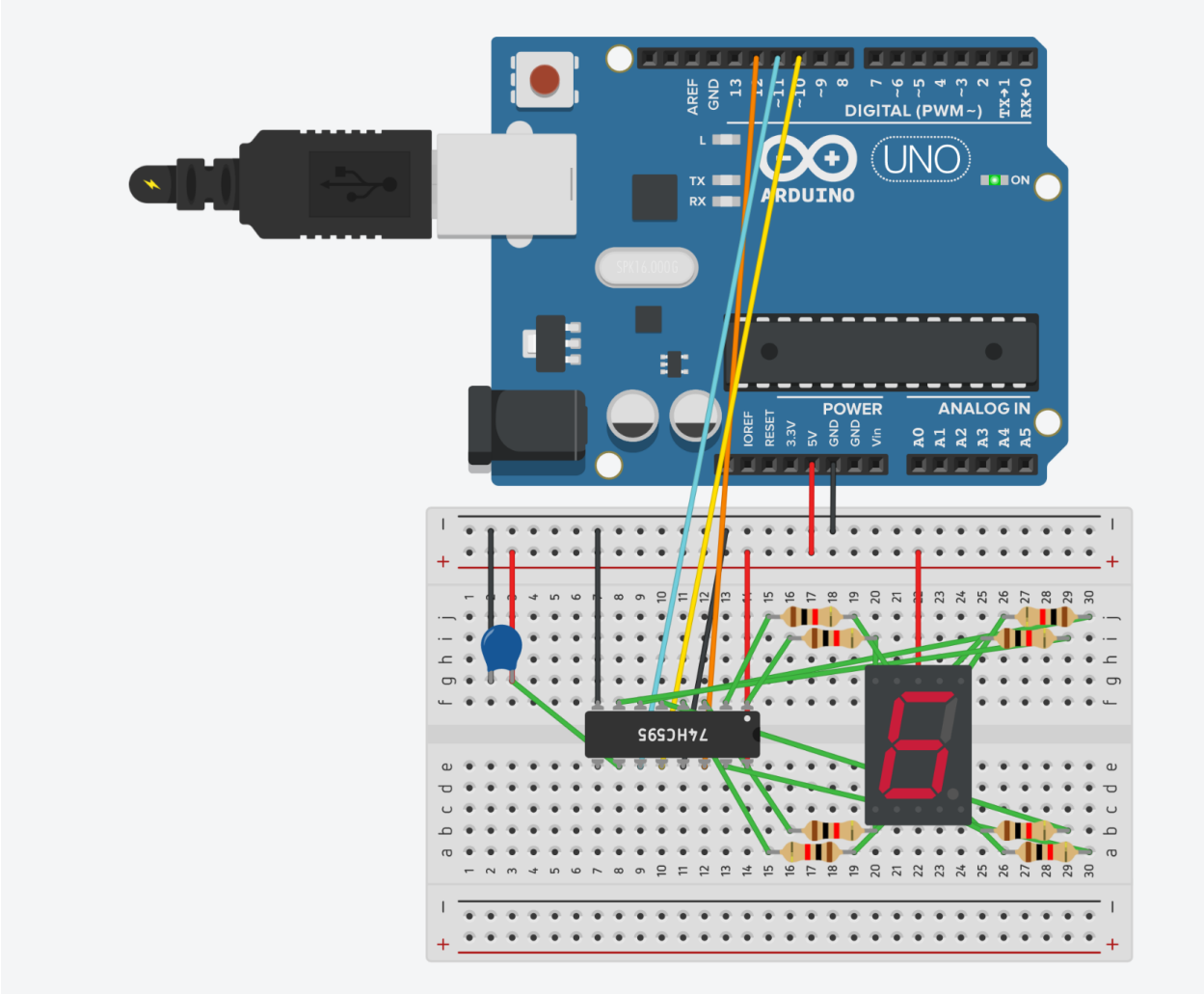

Figure 2.9 Shift register and 7 segment display connected

Shift register and two 7 segment displays. The expected result: to have numbers on two displays. Previous code was used for the test.

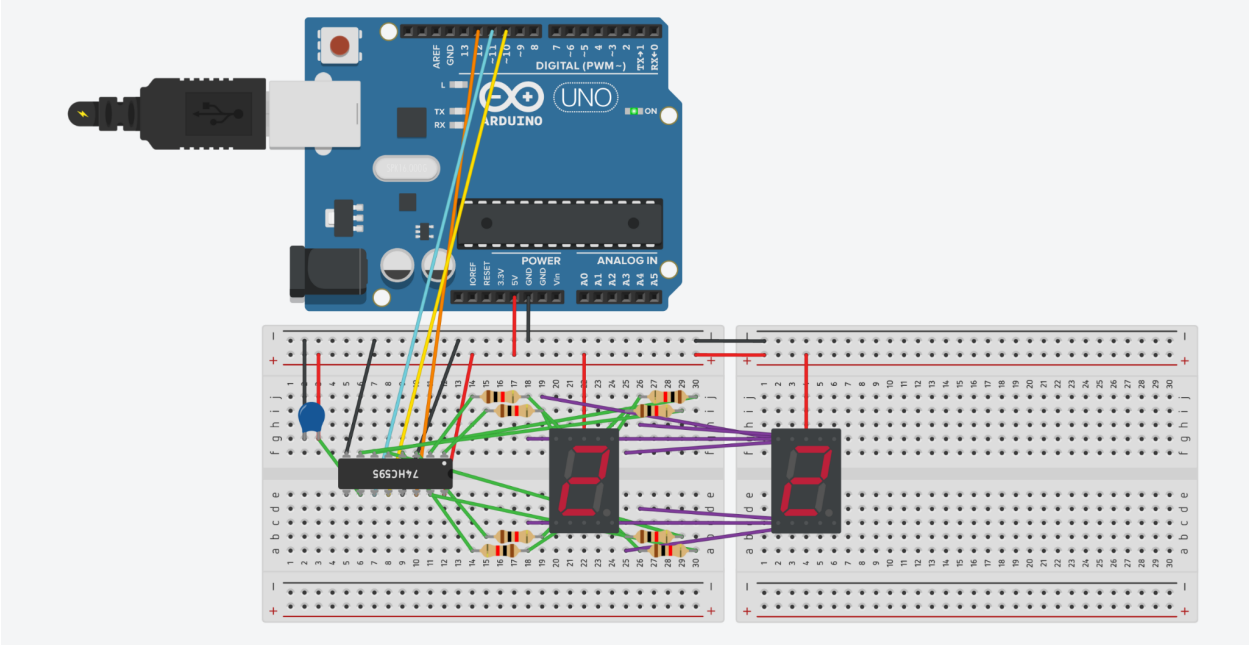

Figure 2.10 Shift register and two 7 segment display connected

Shift register and three 7 segment displays. The expected result: to have numbers on three displays. Previous code was used for the test.

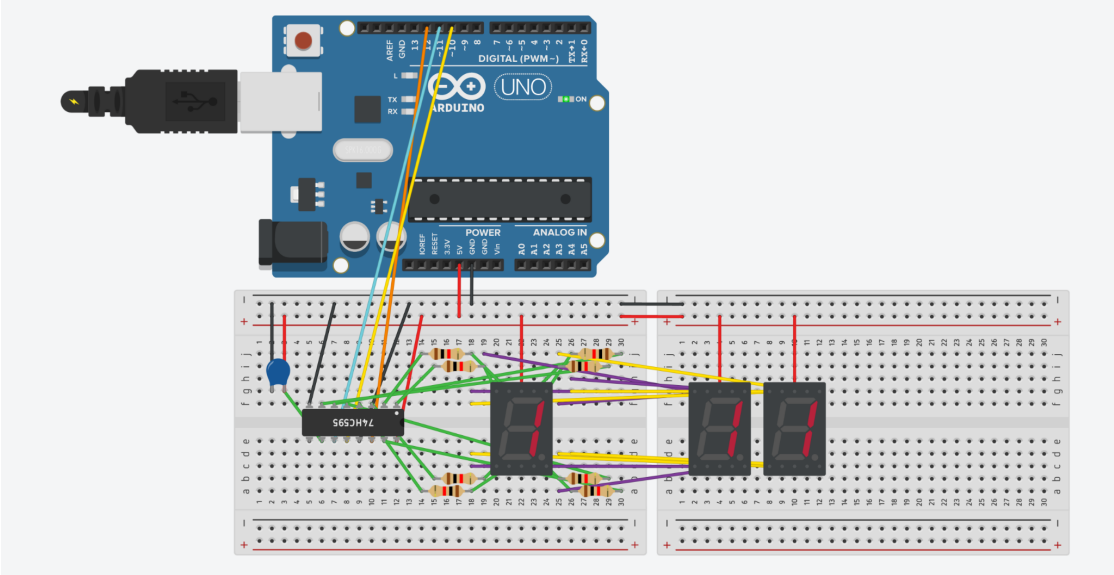

Figure 2.11 Shift register and three 7 segment display connected

Two shift registers and 7 segment display. The expected result: to have numbers on the display. Previous code was used for the test.

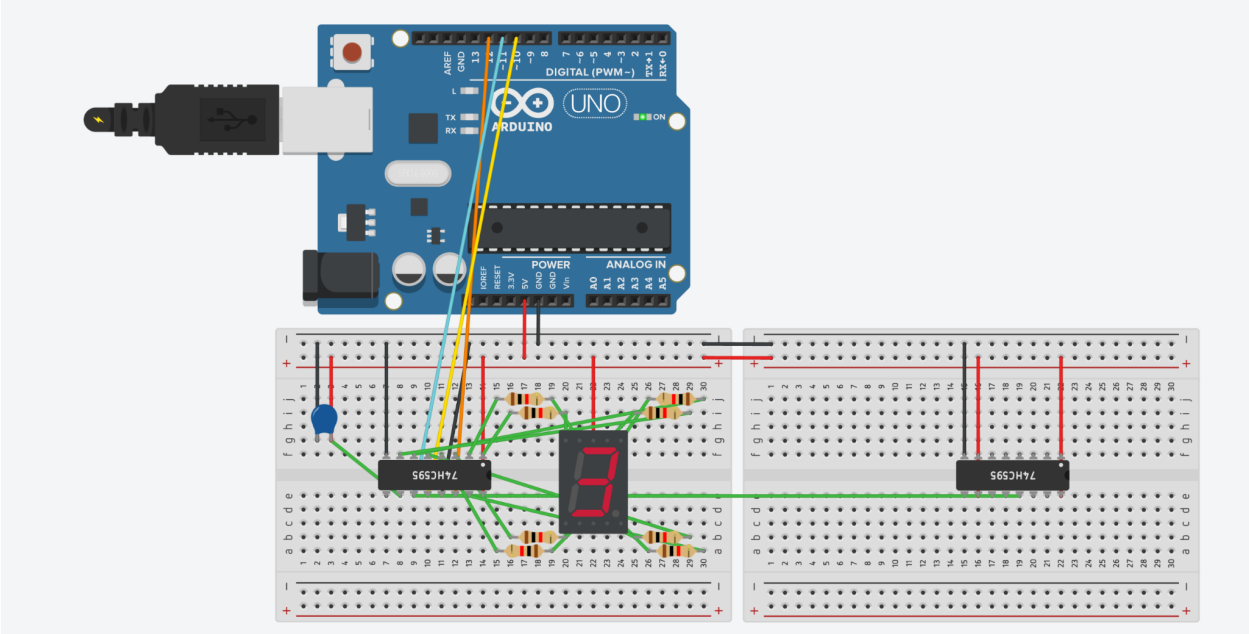

Figure 2.12 Two shift register and 7 segment display connected

Two shift registers and two 7 segment displays. The expected result: to have numbers on two displays. Previous code was used for the test.

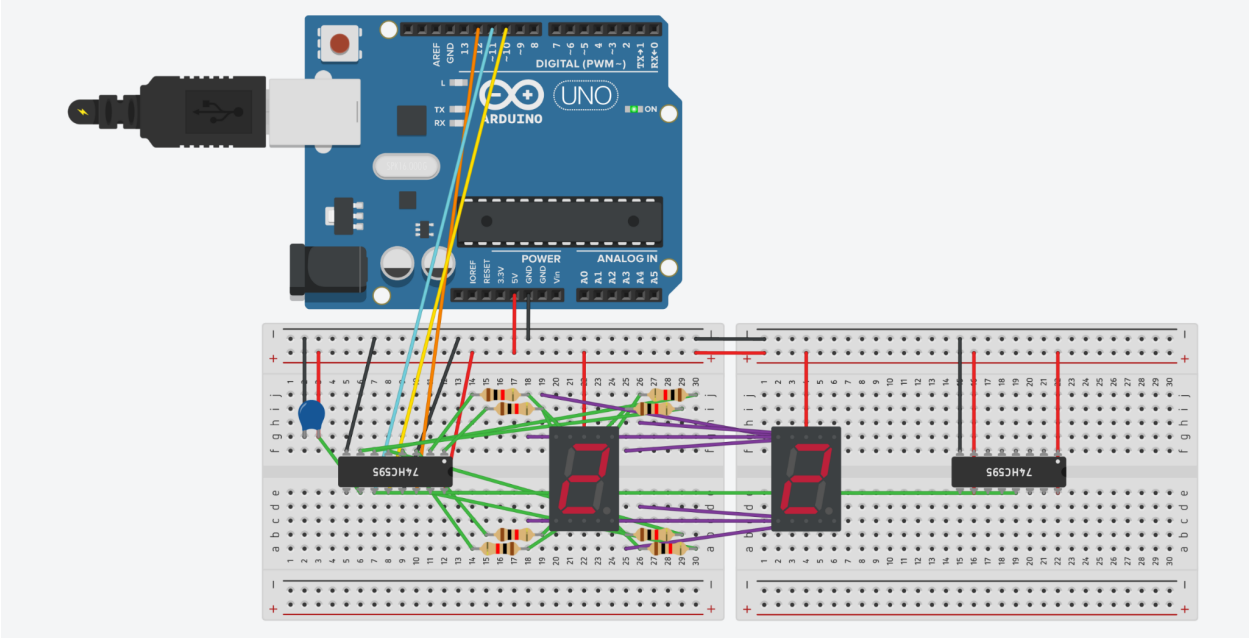

Figure 2.13 Two shift register and two 7 segment display connected

Two shift registers and three 7 segment displays. The expected result: to have numbers on three displays. Previous code was used for the test.

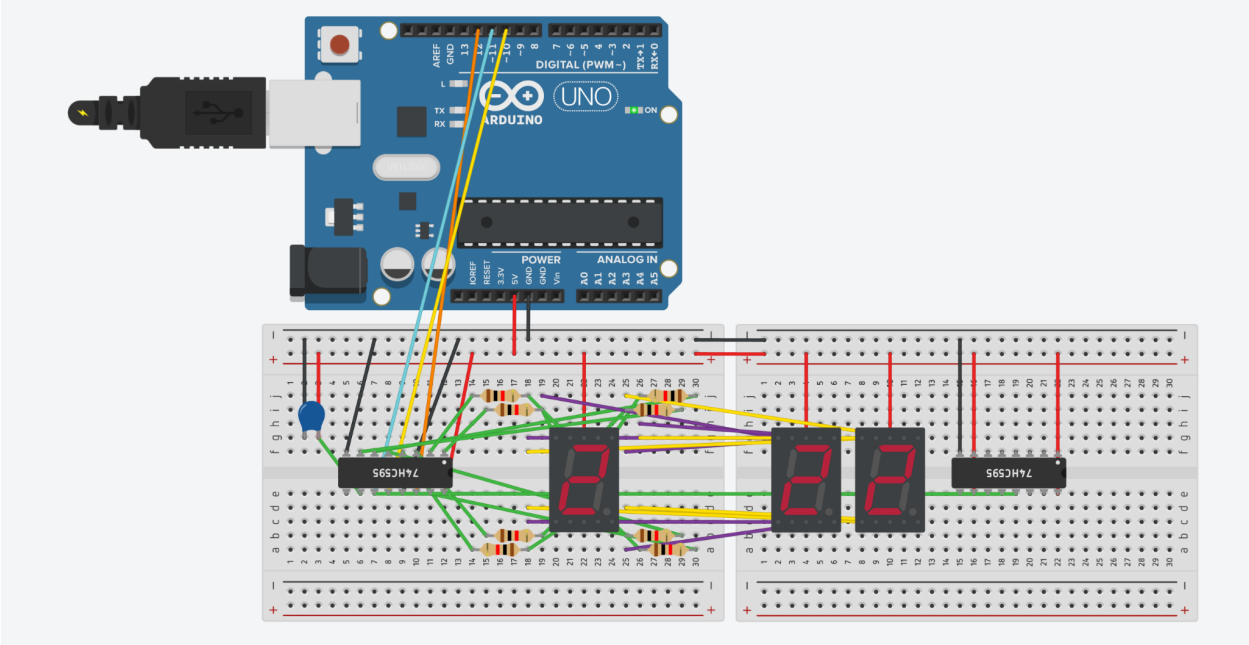

Figure 2.14 Two shift register and three 7 segment display connected

Chapter 3

Low Level Software

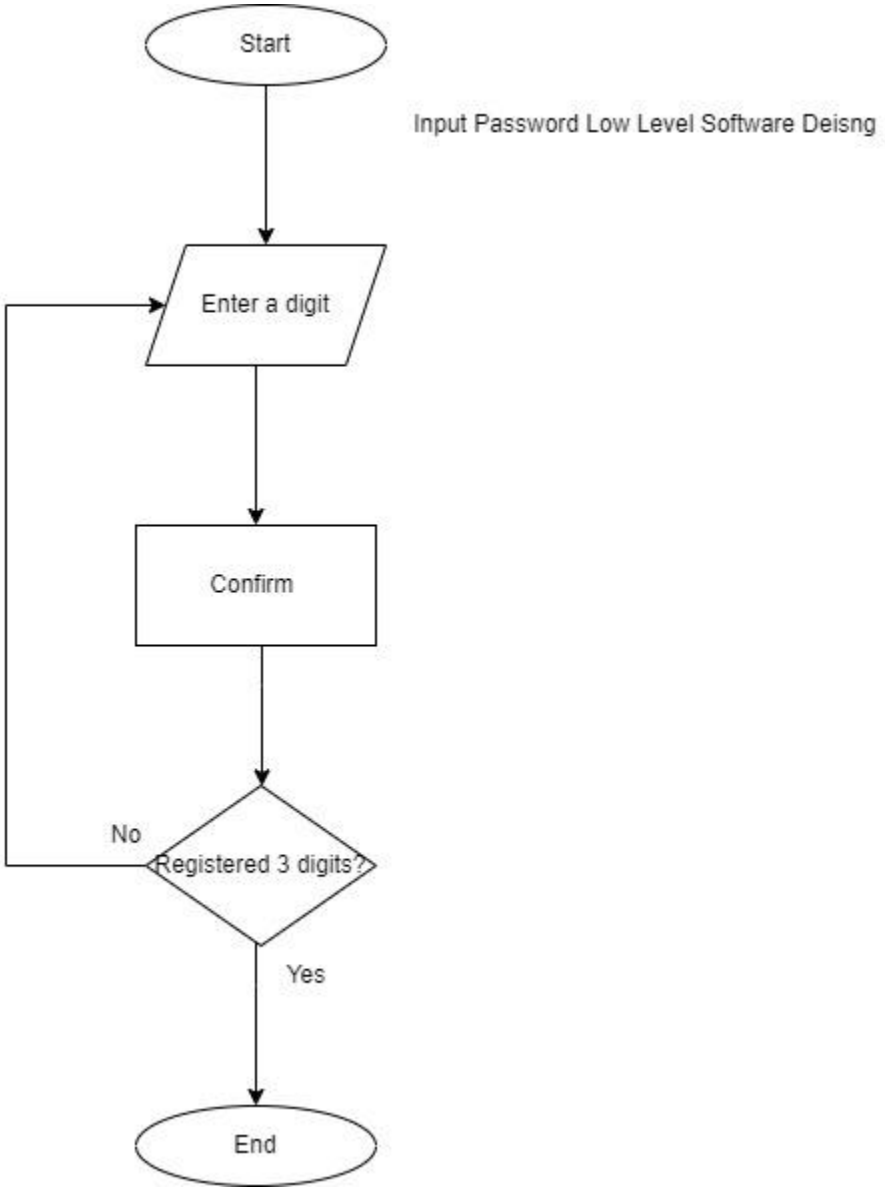

Figure 3.1 low level software pt. 1

This structure of code is given a set of integers, and it is only allowed to accept numbers between 0 and 9.

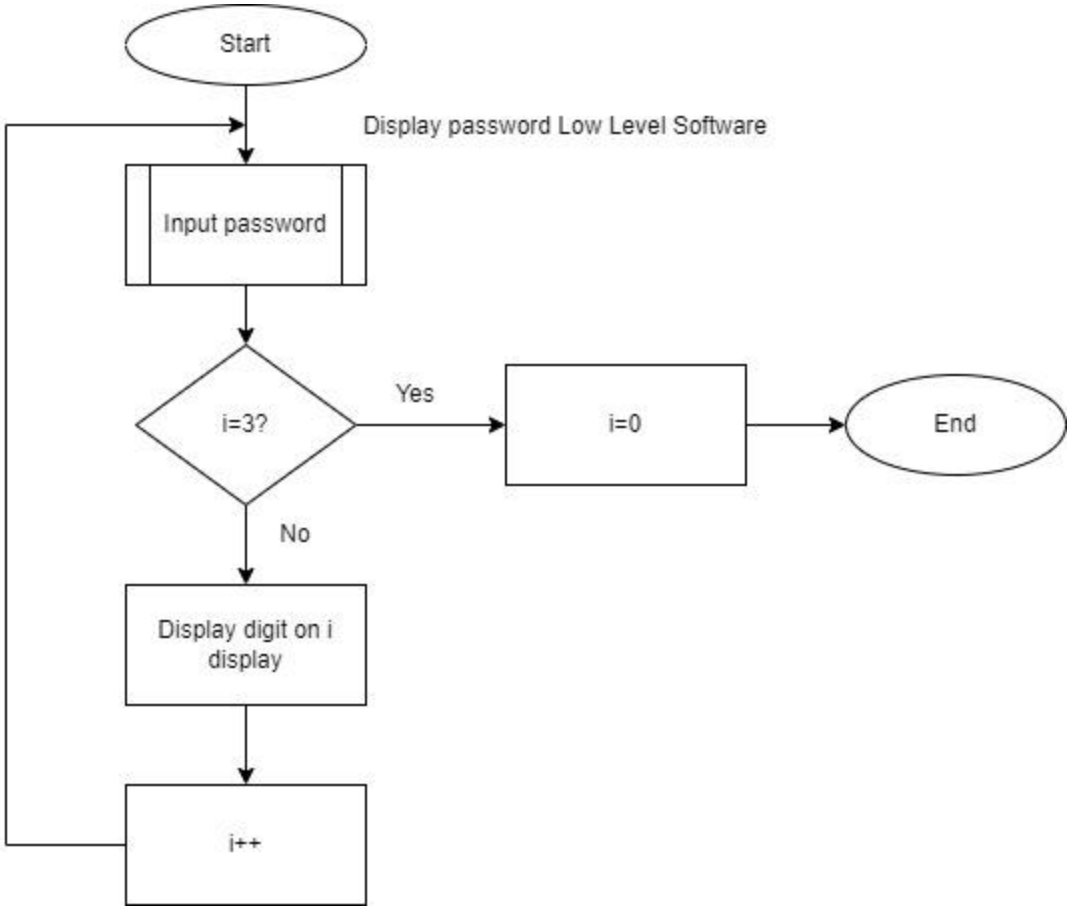

Figure 3.2 low level software pt. 2

On a 7-segment display, the integers that were previously received will be displayed sequentially.

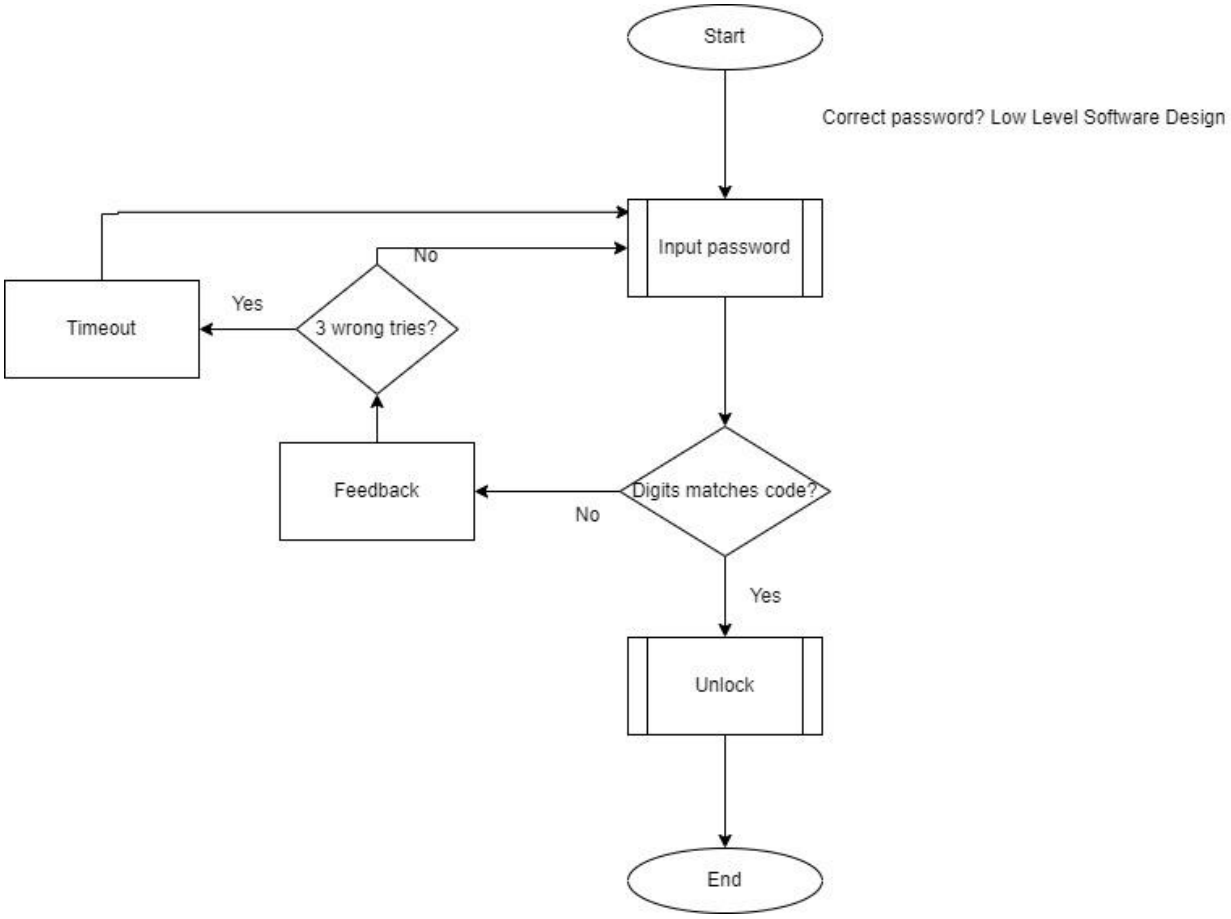

Figure 3.3 low level software pt. 3

This program compares the given set of integers with the password and provides feedback; after three incorrect attempts, a timeout is activated.

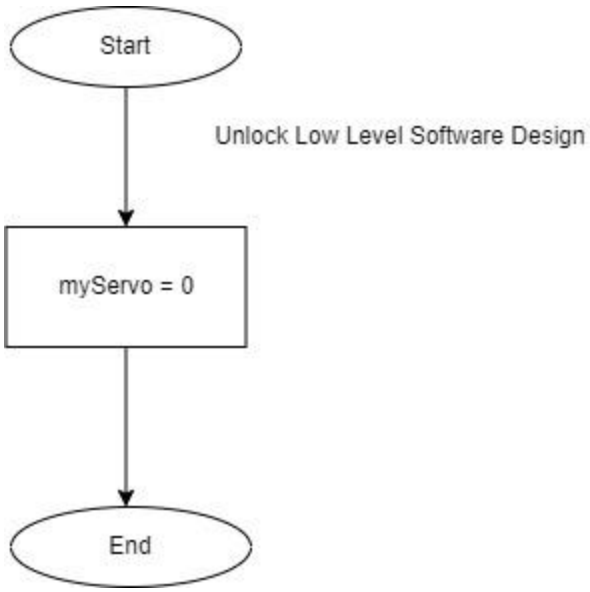

Figure 3.4 low level software pt. 4

This code opens the safe once the password is correct.

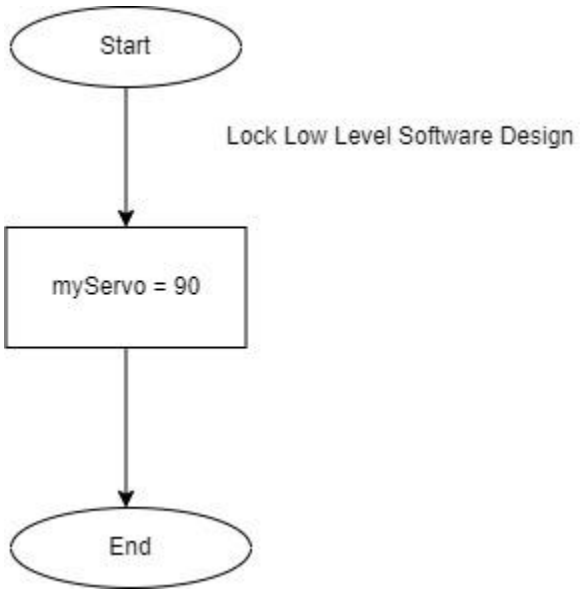

Figure 3.5 low level software pt. 5

After the safe is open it will be closed using this program.

Software unit testing procedure

Servo RS-303WP.

```
//Power to 5V, ground to ground and Signal to port 9.
// Unit testing Servo RS-303WP using existing Servo.h library.
#include <Servo.h>


Servo myservo;

int pos = 0;

void setup() {

  myservo.attach(9);
}


void loop() {
//servo goes from 0 degrees to 180
  for (pos = 0; pos <= 180; pos += 1) {

          // in steps of 1 degree

          myservo.write(pos);
          delay(15);
  }
//servo comes back from 180 to 0 degrees

  for (pos = 180; pos >= 0; pos -= 1) {
          myservo.write(pos);

          delay(15);

  }

}
```

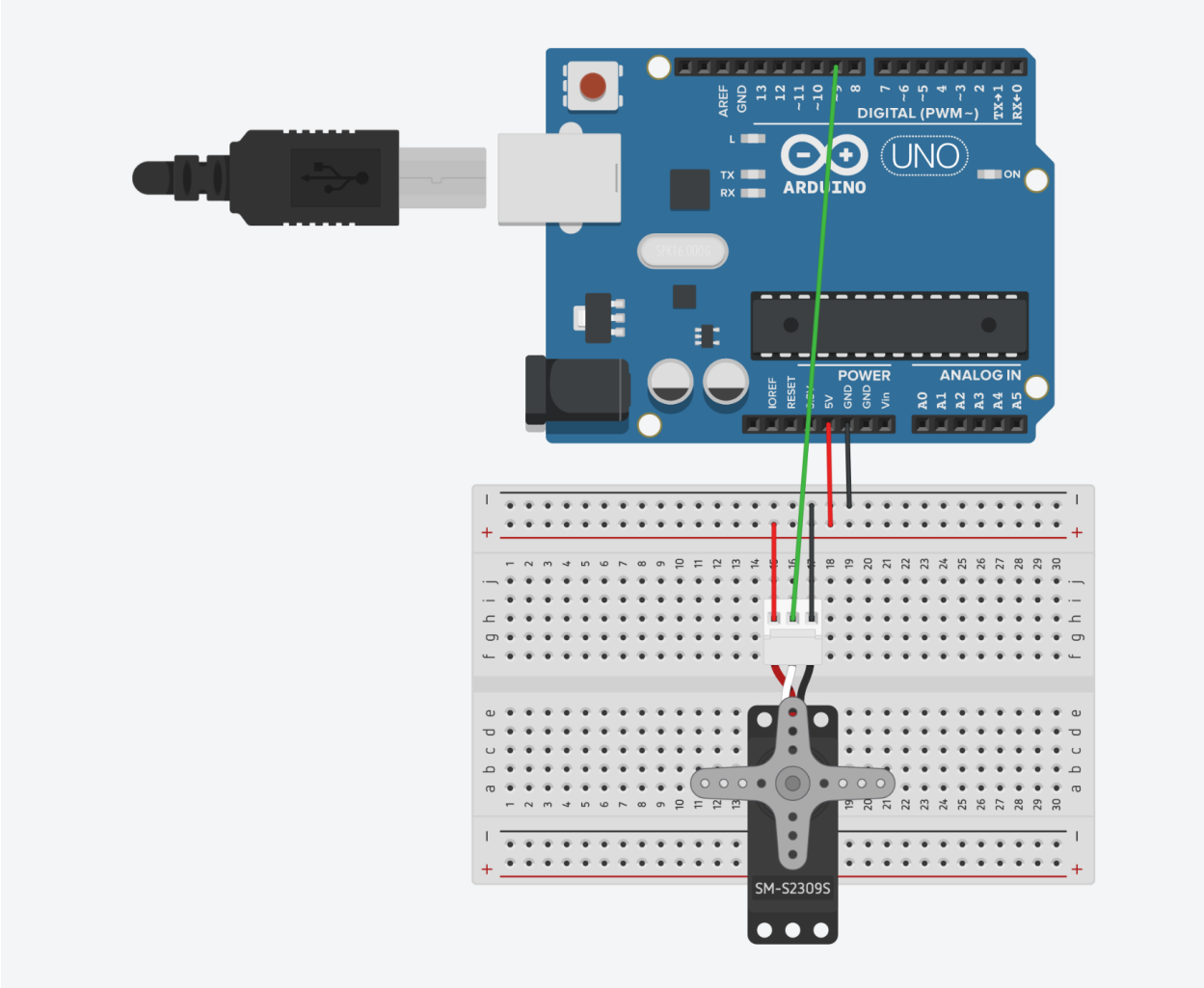

Figure 3.6 servo motor simulation

7 segment display:

```
#define DELAY 500
void setup()
{
  DDRB = 0b111111;
  DDRD = 0b1111111;
}

void loop()
{
  PORTD = 0b1111111;
  PORTB = 0b111001;
  delay(DELAY);
```

```
  PORTB = 0b100100;
  PORTD = 0b10111111;
  delay(DELAY);
  PORTB = 0b110000;
  delay(DELAY);
  PORTB = 0b011001;
  delay(DELAY);
  PORTB = 0b010010;
  delay(DELAY);
  PORTB = 0b000010;
  delay(DELAY);
  PORTB = 0b111000;
  PORTD = 0b11111111;
  delay(DELAY);
  PORTB = 0b000000;
  PORTD = 0b10111111;
  delay(DELAY);
  PORTB = 0b010000;
  delay(DELAY);

}
```

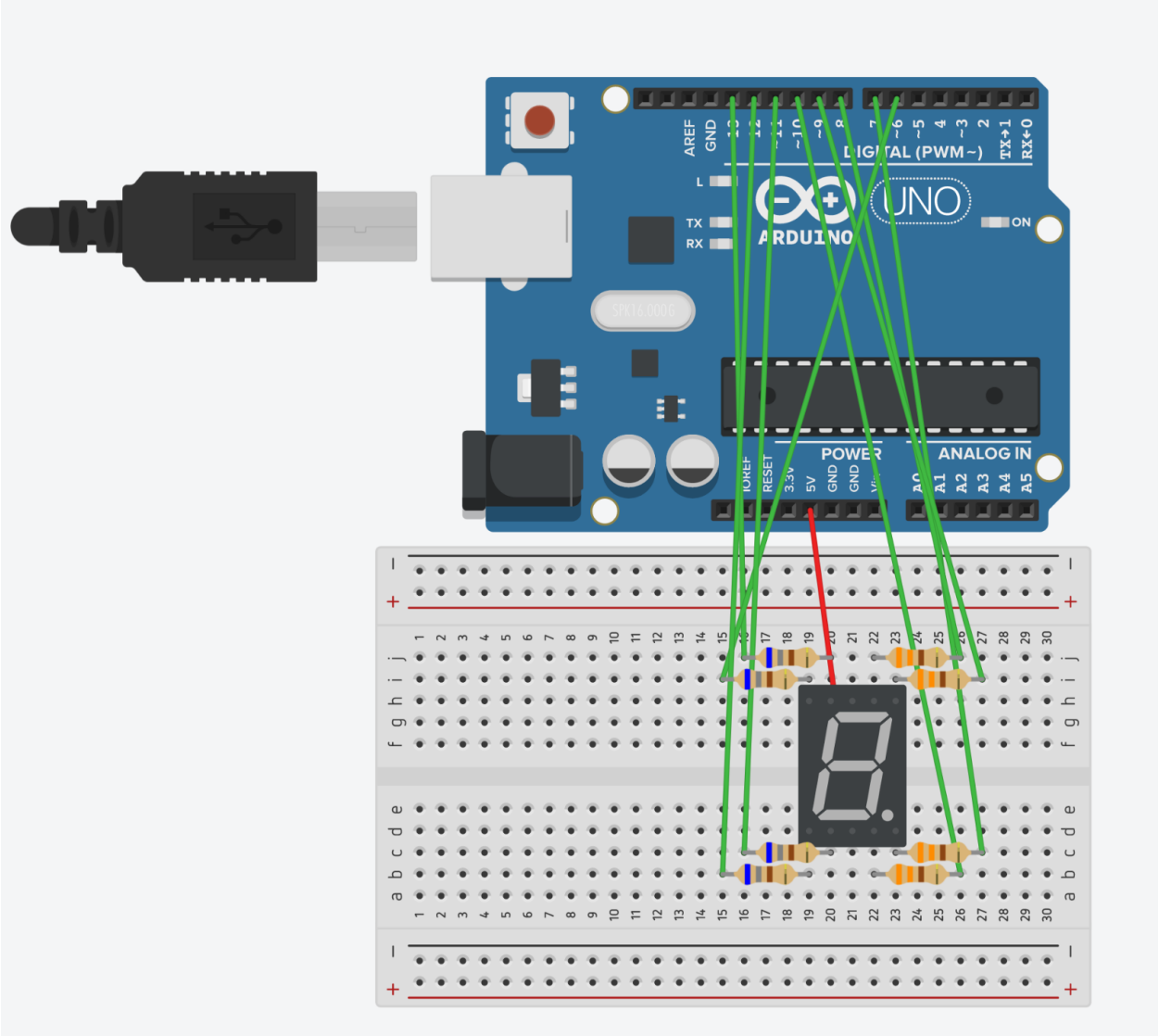

Figure 3.7 7 segment display simulation

Led

```
const int LED = 10;

void setup()
{
 pinMode(LED, OUTPUT);
}

void loop()
{
 digitalWrite(LED, HIGH);
 delay(1000);
 digitalWrite(LED, LOW);
 delay(1000);
```

}

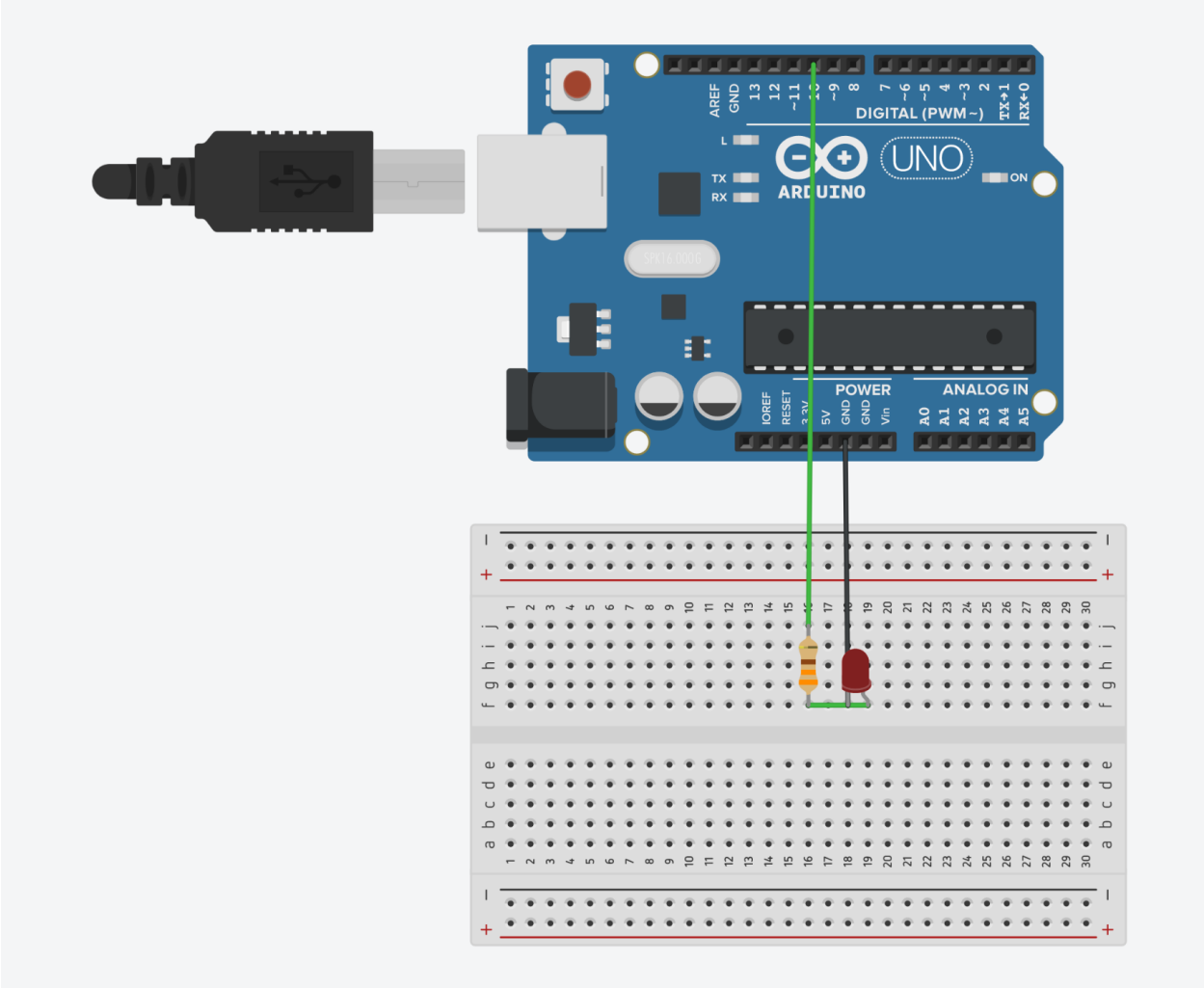

Figure 3.8 LED simulation

Button

```
const int buttonPin = 13;
int currentState;
int pushCounter = 0;
int lastState = HIGH;

void setup() {
  Serial.begin(9600);
  Serial.println("Push Button Counter");
  Serial.println(" ");
  pinMode(buttonPin, INPUT);
}
```

```
void loop() {
  currentState = digitalRead(buttonPin);
  if (currentState != lastState) {
            if (currentState == LOW) {
            pushCounter++;
            Serial.print("Button Press Count = ");
            Serial.println(pushCounter);
            }
  }
  lastState = currentState;
  if (pushCounter >= 10) {
            Serial.println(" ");
            Serial.println("Resettng the Counter...");
            pushCounter = 0;
  }
  delay(50);
}
```

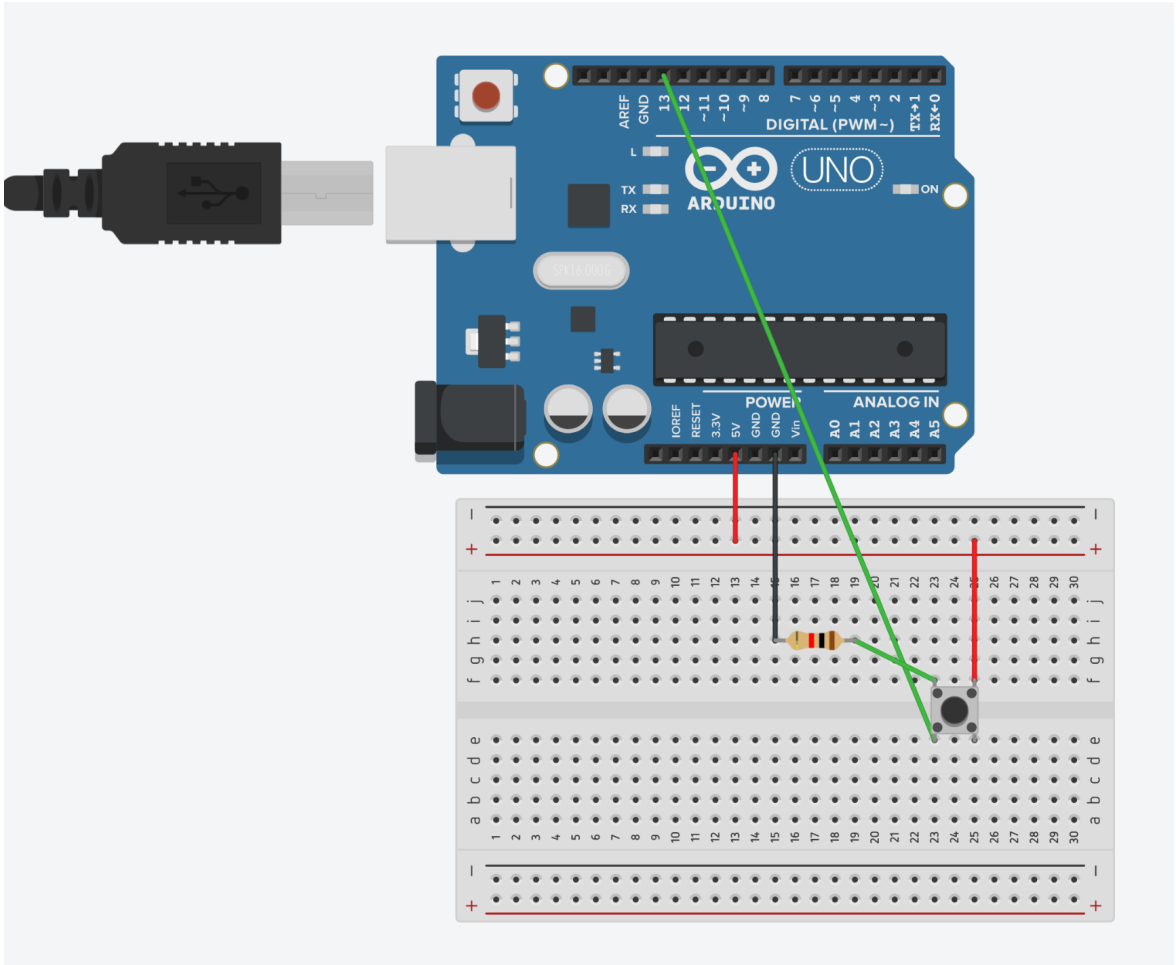

Figure 3.9 button simulation

Buzzer

```
int buzzer=10;
void setup()
{
 pinMode(buzzer, OUTPUT);
}

void loop()
{

 digitalWrite(buzzer, HIGH);
 delay(2000);
 digitalWrite(buzzer, LOW);
 delay(2000);

}
```

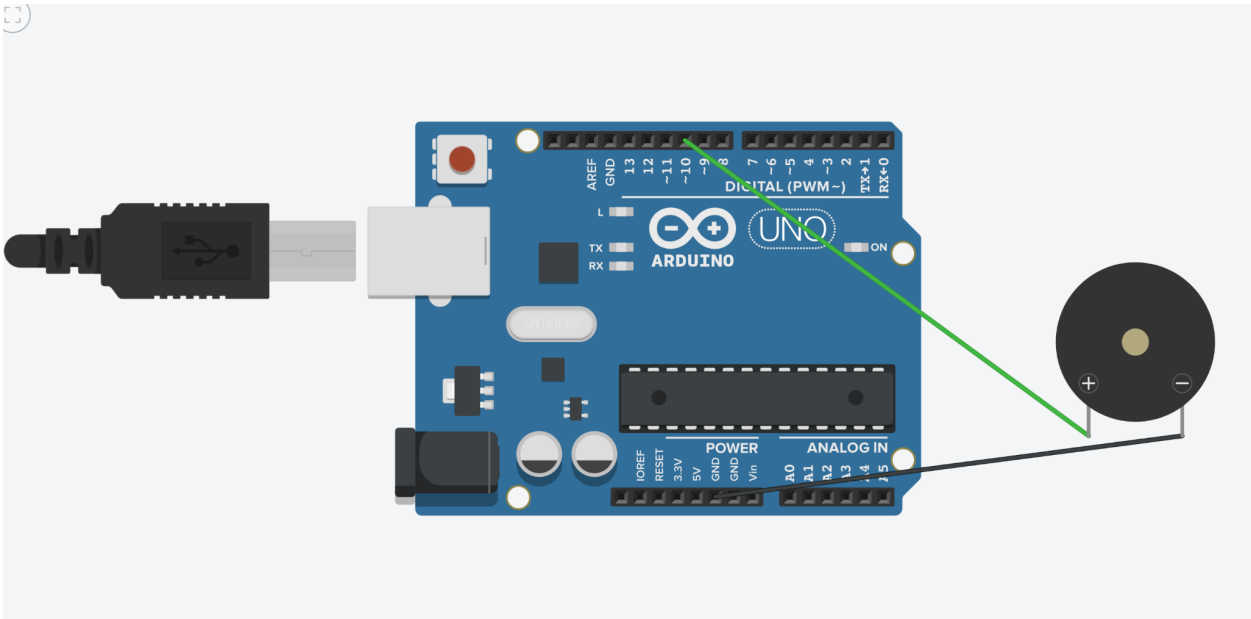

Figure 3.10 buzzer simulation

Hardware

Choice of electronic components and their functioning explained

Starting off with the more basic components we chose to use 3 tactile switches. This was due to the fact that they are simple in terms of how they function, however they provide us with great uses to implement complex functionality in the software. They act as a trigger to activate a piece of code on button press which allows for user interaction. As a type of switch, it was clear that it could have bouncing problems; however, this was an easy fix with the help of software.

LEDs were chosen as the main source of visual user feedback. They are solely used to indicate whether the vault is open or closed. The colors we stopped on were red and green as they are universally used for opposing states of which green usually means the better one.

The buzzer is the only way of auditory feedback for the user. In our solution, the buzzer indicates a wrong sequence of numbers that has been input, thus the vault would not unlock. The duration of the buzz has been set manually using software.

7-Segment display is one of the easiest ways to display a number. Codes usually consist of 3 or more digits. In our case we use 3 of these segments to represent a 3-digit code using the digits from 0 to 9. Another reason to choose these particular displays is that they have a dot LED which we will use as an indicator of the current selected display.

BCD decoder or in our case the SN74LS47N, is a must when using a 7-segment display. It deals with the conversion of a 4-bit binary number to the right signals that will result in the representing number to be displayed on the 7-segment. If the decoder is not used this can be handled through software, however this will take a lot more time and effort, so the wiser choice is to just use one.

The shift registers (SN74HC595N) were picked so that we can pass values to the BCD decoder and the feedback components through them. If we were to do it without them, we would soon realize that we were going to run out of pins to use and still have many inputs to implement.

The rotary encoder was the most convenient way to manipulate the numbers on the displays. It is very intuitive as it represents the turning knob on most real-life vaults. Another reason for the choice of this component was the fact that it has an inbuilt button which means more functionality can be implemented. Naturally both the turning and the button press can cause bouncing during use. We chose to handle the debouncing through software once more because it is way easier to readjust when needed compared to hardware solutions.

The servo motor is the closest resemblance we could think of to a locking mechanism. It can rotate both counterclockwise and clockwise. It is easy to manipulate through software and it doesn’t require a huge amount of voltage or current.

The OLED screen was chosen for the fact that it can display more complex types of feedback to the user. In our case this comes under the form of showing the time remaining of the lockout procedure. It is relatively easy to use and is insanely versatile in what it can offer.

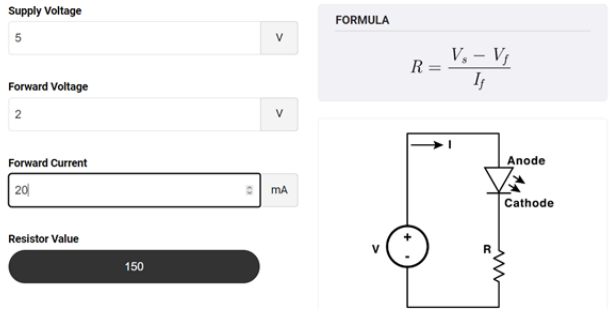

Figure 3.11 resistance calculation [1]

Calculation of component values

Using this formula, we know that we need at least a 150 Ohm resistor to have the red LED light up. Using the same formula but for the green LED the result is 100 Ohm resistor. To be on the safer side we decided to use 220 Ohm resistors for both LEDs.

LED Color	Typical Vf Range
Red	1.8 to 2.1
Amber	2 to 2.2
Orange	1.9 to 2.2
Yellow	1.9 to 2.2
Green	2 to 3.1
Blue	3 to 3.7
White	3 to 3.4

Figure 3.12 typical Vf range [2]

After some research and experimentation, we came to the conclusion that the buttons need to be pulled down by resistors. The most appropriate value for the resistor we found was 10k Ohm as it is widely used, thus can be found easily, and does the job perfectly in our case.

For the rotary encoder we followed the recommended example schematic and settled on 10k Ohm resistors once again.

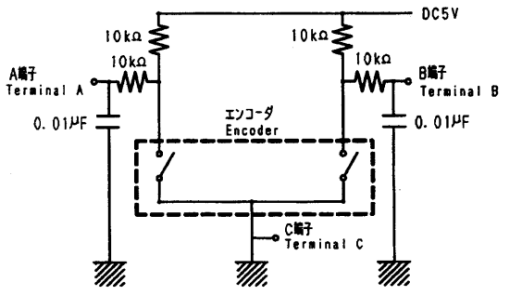

Figure 3.13 rotary encoder documentation [3]

Technical details / explanation of schematic

Low Level Hardware

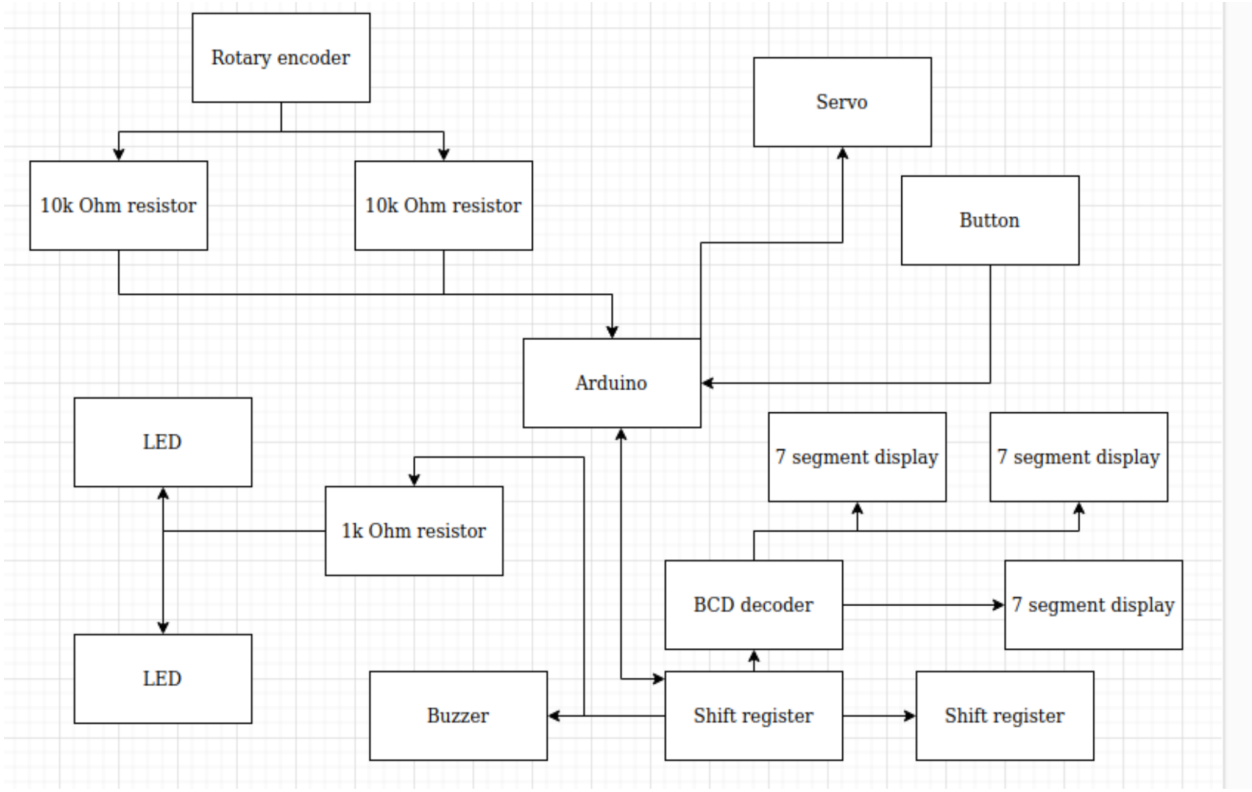

Figure 3.14 low level hardware

Electronic schematic drawing with CAD (Altium)

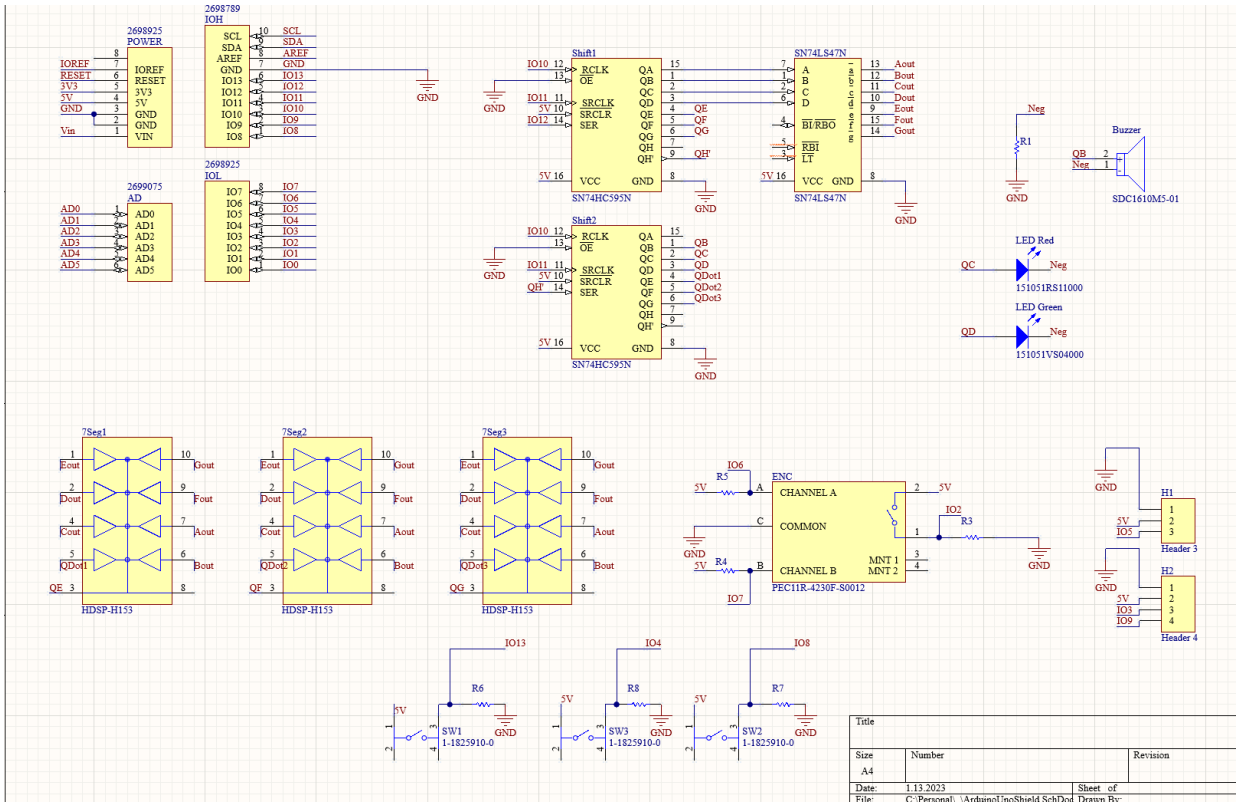

Figure 3.15 schematic

Explanation of the schematic and relation with the block diagram

The rotary encoder is connected to the power supply (The Arduino Uno used in the project) with 2 10k Ohm resistors to create current and another 10k Ohm connecting it to ground. The information from the encoder goes to the Arduino through connections with pins : 2,6,7(IO2;IO6;IO7). Subsequent to the Arduino getting the information it relays it to one of the shift registers which runs it through the BCD decoder. Following the decoding of the information given by the shift register, then it's relayed to the three seven segment display which shows the number currently given by the rotary encoder. The servo motor acts as a closing mechanism and the state that it’s currently in is saved in the memory of the Arduino. The button on the low level design can be seen on the schematic as SW1. On press, if the servo is in an open state, it closes it.

The LED Red and the LED Green as seen on the low level design along with the buzzer serve as a feedback to the user. If the code entered by the user is deemed correct by the program, the LED Red which by default is ON while the Servo is closed turns OFF and the LED Green turns ON, while the servo rotates to indicate an open state. However if the password entered is wrong the LED Red stays ON and the buzzer creates a sound to indicate a wrong password.

Design of the PCB / layout with CAD (Multisim/Ultiboard, or Altium)

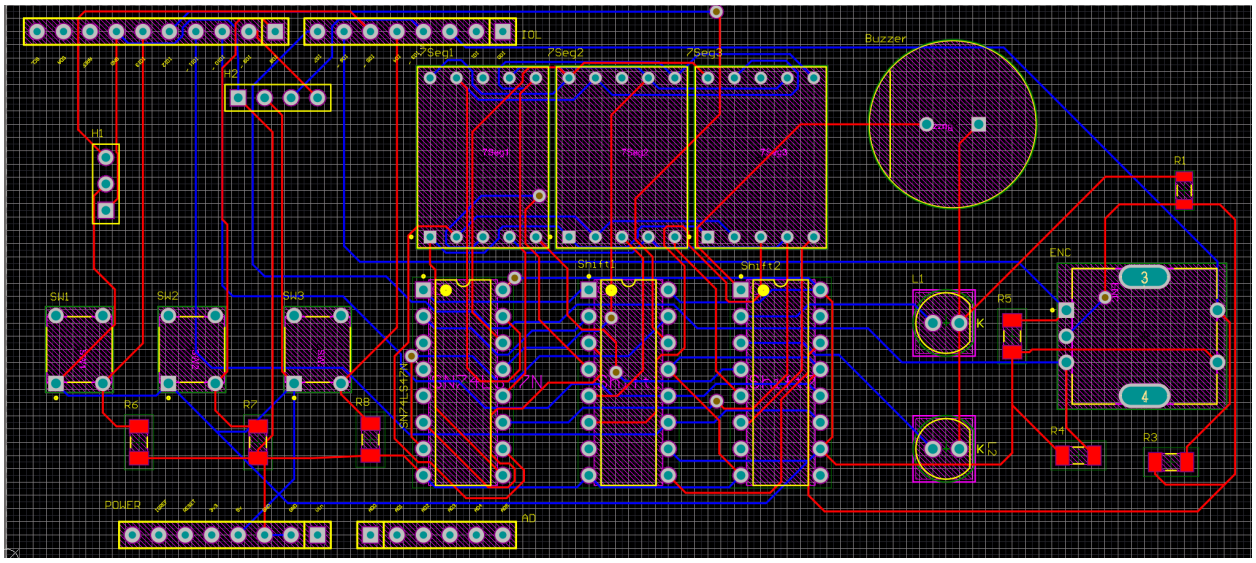

Figure 3.16 PCB layout

Hardware Unit testing procedure

All components were tested before implementation. Test was done using pieces of code in the software unit testing.

Name of component	Simulation	Hardware unit test (Real life component test)	Expected test result
Servo motor	**Fig. 3.6**	**Yes**	**Motor works clockwise and counterclockwise**
7 segment display	**Fig. 3.7**	**Yes**	**Display turns on and shows numbers**
LED	**Fig. 3.8**	**Yes**	**Green and red light on**
Button	**Fig. 3.9**	**Yes**	**Reacting to press**
Buzzer	**Fig. 3.10**	**Yes**	**Make a sound**
Rotary encoder	**No**	**Fig. 3.14**	**Works clockwise and**

			counterclockwise, reactive to button press
Shift register	**No**	**Yes**	**Works as additional arduino inputs**
BCD decoder	**No**	**Yes**	**Pass correct values that lights on a 7 segment display**
LCD module	**No**	**Yes**	**Prints text and numbers**

Tests went as expected and correct responses were received.

Rotary encoder :
Rotary encoder was tested with oscilloscope

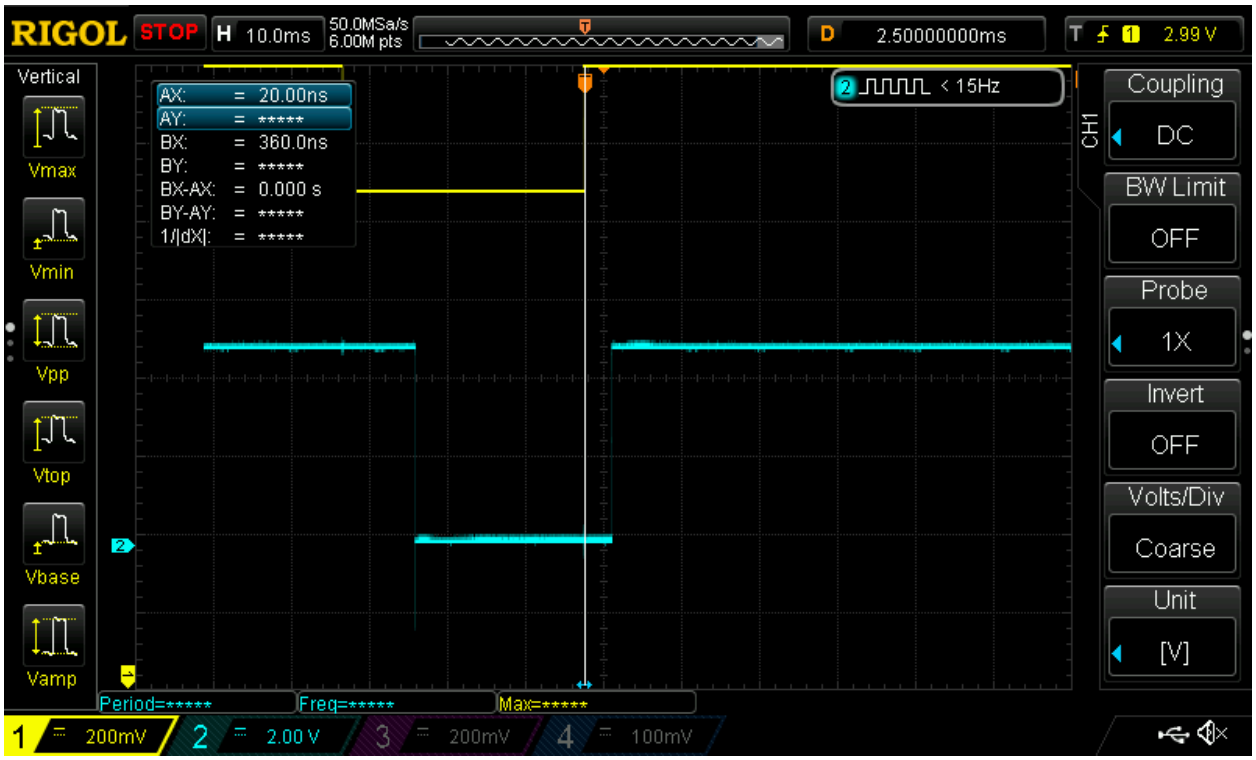

Figure 3.17 rotary encoder

Chapter 4

Test results

During the project numerous tests were held to make sure that system works correctly and according to the user requirements. Unit testing helps to recognise component malfunction if there was one, as well create first software-hardware connections. After seeing that all components work according to their function, further integration tests were conducted. The main point of the integration test was to combine components in a small group and observe their behavior as a group. For example, two or three 7 segment display tests [Fig. 2.10-2.11] showing that numbers can be displayed at the same time on three displays, which is a crucial part as user requirements specifications requested a three-digit password. More expanded explanation of expected results can be found in the second chapter. After the tests were completed, some information about the components were collected. For instance, bouncing of the rotary encoder, as well as bouncing of the integrated button. Sometimes numbers were skipped and when button was pressed, due to bouncing, after 7 segment display #1 input jumped to display #3. These problems were fixed later on in the software. During final system tests it was found that the LCD module and servo share the same address in Arduino analog pins 4 and 5. The solution is to manually change the address of the LCD module in the software. All other tests did not reveal any problems and worked under unusual conditions, e.g. if power was to cut off during time out (time out countdown can be seen on LCD module) when system regains power timer would go with the second it was left on.

Appendix

Code

```
#include <Servo.h>
#include <EEPROM.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <SPI.h>
#include <Wire.h>

#define OLED_RESET 4
#define OLED_WIDTH 128
#define OLED_HEIGHT 64

Adafruit_SSD1306 display(OLED_WIDTH, OLED_HEIGHT, &Wire, -1);

// st-cp pin 12
#define LATCH_PIN 10
// sh-cp pin 11
#define CLOCK_PIN 11
// ds pin 14
#define DATA_PIN 12

// rotary output pins 6 , 7
#define ROTERY_OUT_A 6
#define ROTERY_OUT_B 7
#define ROTERY_BUTTON 2

#define SERVO_PIN 5

// buttons
#define PASS_CONFIRM_BUTTON 13
#define LOCK_BUTTON 4
#define CHANGE_PASS_BUTTON 8
// enum to addresses in memory
enum memory
{
  TIMER = 0,
  LOCKSTATE = 2,
  PASSWORD1 = 7,
```

```
  PASSWORD2 = 8,
  PASSWORD3 = 9,
  WRONGTRIES = 10,
  TIMEOUT = 20
};

// OLED
// Connect SCL to A5 and SDA to A4
int wrongTries = 0;
int timer = 60;
int timeOut = 0;

// for GUI
String serialData;
char command;

// var for rotary encoder
int rotaryCounter = 0;
int rotaryCurrentStateA;
int rotaryLastStateA;
String rotaryCurrentDir = "";
int chosenSeg = 0;
int password[3];
// other vars
Servo myServo;
int numbers[3] = {0, 0, 0};

// variables to store state change time
unsigned long lastTimeButtonStateChanged = 0;
unsigned long lockButLastTimeButtonStateChanged = 0;
unsigned long submitButlastTimeButtonStateChanged = 0;
unsigned long changeButLastTimeButtonStateChanged = 0;
unsigned long lastChangeTime = 0;
unsigned long BuzzerDuration = 0;

// var for looping 7seg in general loop function
int i = 0;

// change code that is in memory
void ChangeCode(int newCode[])
{
  EEPROM.update(PASSWORD1, newCode[0]);
  EEPROM.update(PASSWORD2, newCode[1]);
  EEPROM.update(PASSWORD3, newCode[2]);
```

```
  Serial.println("Code changed to - " + (String)newCode[0] + newCode[1] + newCode[2]);
}

byte IsLocked()
{
  return EEPROM.read(LOCKSTATE);
}

void GetPass(int password[])
{
  for (int i = 0; i < 3; i++)
  {
          password[i] = EEPROM.read(i + 7);
  }
}

String GetCurrentPass()
{
  String data;
  data += EEPROM.read(PASSWORD1);
  data += EEPROM.read(PASSWORD2);
  data += EEPROM.read(PASSWORD3);
  return data;
}

void NewCodeToArr(int code[], int number)
{

  for (int i = 2; i >= 0; i--)
  {
          code[i] = number % 10;
          number /= 10;
  }
}

void rememberTimer()
{
  timer = EEPROM.read(TIMER);
}

void rememberWrongTries()
{
  wrongTries = EEPROM.read(WRONGTRIES);
}
```

```
void rememberTimeOut()
{
 timeOut = EEPROM.read(TIMEOUT);
}

// send 16 bit of data to shift register
void sendDataToShift(int data)
{
 digitalWrite(LATCH_PIN, LOW);
 shiftOut(DATA_PIN, CLOCK_PIN, MSBFIRST, data);
 shiftOut(DATA_PIN, CLOCK_PIN, MSBFIRST, data >> 8);
 digitalWrite(LATCH_PIN, HIGH);
}
// Will check if the duration of buzzer not finished it will play the buzzer
int BuzzerOut()
{
 unsigned long duration = millis();
 // play the buzzer until the desire time
 if (BuzzerDuration > duration)
 {
        return 2;
 }
 else
 {
        return 0;
 }
}

// Series of action will be activated in case of timeout
void timeOutFunction()
{
 // This will be used to set a delay
 unsigned long previousTimeOutMillis = 0UL;
 unsigned long timeOutinterval = 1000UL;
 unsigned long currentTimeOutMillis = millis();
 while (timeOut == 1)
 {
        rememberTimer();
        // Set up the display and write the time left
        display.clearDisplay();
        display.setTextColor(WHITE);
        display.setTextSize(2);
        display.setCursor(10, 28);
```

```
        display.println("Timeout:");
        display.setCursor(103, 28);
        display.print(timer);
        // This code will be executed once every second
        unsigned long currentTimeOutMillis = millis();
        if (currentTimeOutMillis - previousTimeOutMillis > timeOutinterval)
        {
        timer--;
        previousTimeOutMillis = currentTimeOutMillis;
        }
        // Save timer to memory
        EEPROM.update(TIMER, timer);
        display.display();
        // Inform user that the timer has ended
        if (timer <= 0)
        {
        display.clearDisplay();
        display.setTextSize(2);
        display.setCursor(15, 18);
        display.print("Timeout");
        display.setCursor(15, 38);
        display.print("expired.");
        display.display();
        int period = 5000;
        unsigned long time_now = 0;
        time_now = millis();
        while (millis() < time_now + period)
        {
        // wait approx. 5 ms
        }
        timer = 60;  // reset timer
        timeOut = 0; // turn off the timeout
        EEPROM.update(TIMEOUT, timeOut);
        EEPROM.update(TIMER, timer); // reset the memory
        }
        // If there is no timeout this will display nothing
}
if (timeOut == 0)
{

        display.clearDisplay();
        display.setTextColor(WHITE);
        display.setTextSize(2);
        display.setCursor(45, 28);
```

```
        display.println(" ");
        display.display();
  }
}

// this function shows the numbers in 7seg , on or off the LEDs and play the buzzer
void showData()
{
  // empity 16bit for shift register
  int myEmpity = 0B0000000000000000;

  // set bits for the 7seg to turn on in order
  int first7Seg = 0B00010000;
  int second7Seg = 0B00100000;
  int third7Seg = 0B01000000;

  // array for 7seg bit on shift register
  int my7Seg[3] = {first7Seg, second7Seg, third7Seg};

  // data on the first shift register 7seg common and 4bit data to BCD
  int firstShift = my7Seg[i] | numbers[i];

  int secondShift = 0B00000000; // data on the second shift registery

  firstShift = firstShift << 8;

  int completeData = myEmpity | firstShift;

  if (IsLocked())
  {
        secondShift = secondShift | 0B00001000; // turn on the red led
  };
  if (!IsLocked())
  {
        secondShift = secondShift | 0B00000100; // turn on the green led
  };
  // turning on the dot on the 7seg
  switch (chosenSeg)
  {
  case 0:
        secondShift = secondShift | 0B01100000;
        break;
  case 1:
        secondShift = secondShift | 0B01010000;
```

```
            break;
  case 2:
            secondShift = secondShift | 0B00110000;
            break;
  }

  secondShift = secondShift | BuzzerOut(); // turn the buzzer on

  completeData = completeData | secondShift; // build complete data

  // push bits to shift register for showing numbers on 7 segments and leds and buzzer
  sendDataToShift(completeData);

  // restart the loop for 7seg
  i += 1;
  if (i == 3)
            i = 0;
}
// lock the valut
void lock()
{
  myServo.write(90);
  EEPROM.update(LOCKSTATE, 1);
  numbers[0] = 0;
  numbers[1] = 0;
  numbers[2] = 0;
}
// series of actions to check and unlock or alarm the vault
void unLock()
{
  GetPass(password);
  if (numbers[0] == password[0] && numbers[1] == password[1] && numbers[2] == password[2])
  {
            wrongTries = 0;
            EEPROM.update(10, wrongTries); // reset the wrong tries
            myServo.write(180);                 // move the servo
            EEPROM.update(LOCKSTATE, 0);   // change the lock state
  }
  else // if the password is not correct
  {
            BuzzerDuration = millis() + 2000; // play the buzzer for next 2 second
            rememberWrongTries();             // remember from memory the wrong tries left
            wrongTries++;
            EEPROM.update(WRONGTRIES, wrongTries); // save the wrong tries in memory
```

```
        // Count the wrong tries, activate timeout and then reset the counter
        if (wrongTries >= 3)
        {
        timeOut = 1;                        // activate timeout
        EEPROM.update(TIMEOUT, timeOut);            // save timeout
        wrongTries = 0;                     // reset wrong tries
        EEPROM.update(WRONGTRIES, wrongTries); // save it in the memory
        }
        if (timeOut == 1)
        {
        // Turn off everything when lockdown is active
        sendDataToShift(0);
        timeOutFunction();
        }
  }
}
// read the data from serial and do the command that comes from serial (admin access)
void gui()
{
  if (Serial.available())
  {
        serialData = Serial.readString();
        command = serialData.charAt(0);

        switch (command)
        {
        case 'L':
        if (!IsLocked())
        {
        lock();
        Serial.println("lock");
        }
        break;
        case 'U':
        if (IsLocked())
        {
        // Change the locked state to false
        unLock();
        Serial.println("unlock");
        }
        break;

        case 'C':
        Serial.println("Current code - " + GetCurrentPass());
```

```
        break;
        case 'N':
        int number = serialData.substring(1).toInt();
        int code[3];
        NewCodeToArr(code, number);
        ChangeCode(code);
        break;
        case 'D':
        String text = serialData.substring(1);
        if (timer == 0)
        {
        Serial.println("Text dispalyed!");
        display.clearDisplay();
        display.setTextColor(WHITE);
        display.setTextSize(1);
        display.setCursor(10, 28);
        display.println(text);
        display.display();
        }
        }
  }
}
// function for the rotary encoder to read data from rotary and change the number on selected 7-seg
void rotery()
{
  // for preventing multi count & debounce
  unsigned long currentChangeTime = millis();

  // Read the current state of ROTERY_OUT_A
  rotaryCurrentStateA = digitalRead(ROTERY_OUT_A);
  // If last and current state of ROTERY_OUT_A are different, then pulse occurred
  // React to only 1 state change to avoid double count
  if (rotaryCurrentStateA != rotaryLastStateA && rotaryCurrentStateA == 1 && currentChangeTime -
lastChangeTime > 5)
  {
        lastChangeTime = currentChangeTime;
        // If the ROTERY_OUT_B state is different than the ROTERY_OUT_A state then
        // the encoder is rotating CCW so decrement
        if (digitalRead(ROTERY_OUT_B) != rotaryCurrentStateA)
        {
        if (rotaryCounter < 9)
        rotaryCounter++;
        else
        rotaryCounter = 0;
```

```
            rotaryCurrentDir = "CCW";
            }
            else
            {
            // Encoder is rotating CW so increment
            if (rotaryCounter > 0)
            rotaryCounter--;
            else
            rotaryCounter = 9;
            rotaryCurrentDir = "ACW";
            }
            // change the number of the 7-seg
            numbers[chosenSeg] = rotaryCounter;
  }
  // Remember last ROTERY_OUT_A state
  rotaryLastStateA = rotaryCurrentStateA;
}

// button of the rotary encoder it will change selected 7-seg
void roteryButton()
{
  byte lastButtonState = LOW;
  unsigned long debounceDuration = 250; // millis

  if (millis() - lastTimeButtonStateChanged > debounceDuration)
  {
            byte buttonState = digitalRead(ROTERY_BUTTON);
            if (buttonState != lastButtonState)
            {
            lastTimeButtonStateChanged = millis();
            lastButtonState = buttonState;
            if (buttonState == HIGH)
            {
            if (chosenSeg < 2)
            chosenSeg++;
            else
            chosenSeg = 0;

            rotaryCounter = numbers[chosenSeg];
            }
            }
  }
}
// password confirm button it will triger the unlock function
```

```
void passSubmitButton()
{

 byte lastPassSubmitButtonState = LOW;
 unsigned long submitButDebounceDuration = 250; // millis

 if (millis() - submitButlastTimeButtonStateChanged > submitButDebounceDuration)
 {
        byte buttonState = digitalRead(PASS_CONFIRM_BUTTON);
        if (buttonState != lastPassSubmitButtonState)
        {
        submitButlastTimeButtonStateChanged = millis();
        lastPassSubmitButtonState = buttonState;
        if (buttonState == HIGH && IsLocked())
        {
        unLock();
        }
        }
 }
}
// fuction for the valut door button it will lock the valut when door of valut is closed by calling the lock
function
void lockButton()
{
 byte lockButLastPassSubmitButtonState = LOW;
 unsigned long lockButDebounceDuration = 200; // millis

 if (millis() - lockButLastTimeButtonStateChanged > lockButDebounceDuration)
 {
        byte buttonState = digitalRead(LOCK_BUTTON);
        if (buttonState != lockButLastPassSubmitButtonState)
        {
        lockButLastTimeButtonStateChanged = millis();
        lockButLastPassSubmitButtonState = buttonState;
        if (buttonState == HIGH)
        {
        if (!IsLocked())
        lock();
        }
        else if (buttonState == LOW)
        {
        // play the buzzer as long as the dore opend and it shouldn't
        BuzzerDuration = millis() + 1;
        }
```

```
        }
  }
}
/*
 * function for manually change password button it will check for vault lock status
 * change vault password to entered password
 */
void passChangeButton()
{
  byte changeButLastPassSubmitButtonState = LOW;
  unsigned long changeButDebounceDuration = 200; // millis

  if (millis() - changeButLastTimeButtonStateChanged > changeButDebounceDuration)
  {
        byte buttonState = digitalRead(CHANGE_PASS_BUTTON);
        if (buttonState != changeButLastPassSubmitButtonState)
        {
        changeButLastTimeButtonStateChanged = millis();
        changeButLastPassSubmitButtonState = buttonState;
        if (buttonState == HIGH && (!IsLocked()))
        {
        ChangeCode(numbers);
        }
        }
  }
}
// function that checks for timeout and if there is it will activated
void timeoutCheck()
{
  rememberTimeOut(); // First of all remember if there is a timeout active
  if (timeOut == 1)
  {
        // Update the Servo to Locked state
        EEPROM.update(LOCKSTATE, 1);
        myServo.write(90);
        // Turn off everything when lockdown is active
        sendDataToShift(0);
        timeOutFunction();
  }
}

void setup()
{
```

```
  // setup OLED screen
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);

  // set shift register pins as output
  pinMode(LATCH_PIN, OUTPUT);
  pinMode(CLOCK_PIN, OUTPUT);
  pinMode(DATA_PIN, OUTPUT);

  // Set encoder pins as inputs
  pinMode(ROTERY_OUT_A, INPUT);
  pinMode(ROTERY_OUT_B, INPUT);
  pinMode(ROTERY_BUTTON, INPUT);

  myServo.attach(SERVO_PIN);

  Serial.begin(9600);

  // read the state of rotery encoder
  rotaryLastStateA = digitalRead(ROTERY_OUT_A);
}

void loop()
{

  timeoutCheck(); // Check for timeOut

  showData(); // Display data show data on 7seg , LEDs and buzzer

  gui(); // read and do the command that comes from gui(serial)

  rotery(); // Input password

  roteryButton(); // Input password change selected 7seg

  passSubmitButton(); // Check input password and lock or alarm

  lockButton(); // Lock the vault and check for unauthorized opening

  passChangeButton(); // Change password manually by user
}
```

Reference list

[1] Figure 3.11. Source:
https://www.digikey.com/en/resources/conversion-calculators/conversion-calculator-led-series-resistor
[2]Figure 3.12 Source:
https://www.digikey.com/en/resources/conversion-calculators/conversion-calculator-led-series-resistor
[3]Figure 3.13 Source:
https://learn-eu-central-1-prod-fleet01-xythos.content.blackboardcdn.com/608d78f160587/36136419?X-Blackboard-S3-Bucket=learn-eu-central-1-prod-fleet01-xythos&X-Blackboard-Expiration=1674345600000&X-Blackboard-Signature=vJHWLZUSzLM5BhOlk7tYijjSOSpw3pIQaePw9fymZds%3D&X-Blackboard-Client-Id=329908&X-Blackboard-S3-Region=eu-central-1&response-cache-control=private%2C%20max-age%3D21600&response-content-disposition=inline%3B%20filename%2A%3DUTF-8%27%27ENCODER_STEC12E08.pdf&response-content-type=application%2Fpdf&X-Amz-Security-Token=IQoJb3JpZ2luX2VjEEIaDGV1LWNlbnRyYWwtMSJHMEUCIBf39oEqOVm5C5jeK5YpPPp8j57qQI7YGPsQqRW6iapBAiEAjrNpmRs1CbPR6n%2B0OqRjZl4GX6i898gScv8XEtsePfEq3wQIq%2F%2F%2F%2F%2F%2F%2F%2F%2F%2F%2FARACGgw2MzU1Njc5MjQxODMiDJeCZhYT%2FJGFuAEMiiqzBFycwcVpxe1krIm6Z%2Fm%2F2okmbv8vA6%2FYsKrNmbnZxd4PniIwrk62%2BxmB8fdzhJGRXWRKJptvrjNXi4Xl%2FXesfYyF%2Fihqo%2BgZEKk0YSlgThkgNZSUSP%2BSMMQbMcdGlOZlOclrQSIFgX9i%2FDyRBMRbJOAYL2JAU9hEPFSyBsfxPu1VXkYwoO%2FhKew7Fo4IZZNWcWZcJXclbL6yEIYyZTBL9OfSahszsayyGRoiUmk8XMlU4hE6Rw3PwmcnKCX2qv0xJe%2BiwwlYXdeOFt6lt8q%2BBSCfqua5AHdC26iCHeWO1eULUkLf9dbnZAp2oMJ1RcWPe2x7cQshfx1FUpG2Gw%2F2vA3SocS0CccWkRZSsujuIaCJ74ZP7Eue1WxzPxCG810ln6sUE60LN2aWnuZVXHdqs5BMvtZTDi5ItYjgq8yRUZ0a2f8zwkUCw7eUP8E665AVj%2BeUd9YTe0Esp1SChGdnGsJcAfwbjUC5oY0OKm4fXKm8U0G9WoEXpwYPigbn4boHJY8VwPTad8F4%2BMOoucyxW073K1V9vdYuDz2J3wQlp%2FsNdBTISy30xBkSkBbeOxdX34oPPSiYnBA0CL6YiNLiVh1jUU2sVqlJjZk2A06hQzcmHHLK1yCEG%2Fsqha2AWjjTgqLL8NCBXtEv2x%2BjHGLT8pnIOp%2Bvtr788zjmggApzn%2BPN5O6DF0KQ7R6Sr25BCzDD7Mrg501oUDOfa%2BEyFUeZp8Y228GCFlr3fP1wTYejDq4%2FEvjMKHNsJ4GOqkBx%2Fug2FcBghGuA62CV3sRGsZeMeUwH6vUrhJueJV%2BmC4D4YnIoaB37dKOsDHRAI%2FwKsyzR%2F8IFPb1Tc6FQyZwTIlARtD8iKW1j8mrabuRFreWhwbR27qrY88yJFbxUgtmpeRAWYUwAADVfwTi1tTtHgsRPdNih%2B2G%2BIOoIrfdEcq6nIqnGCkru92g%2B%2BhJkJkhWED8XcuX%2FEJHDSP5G3v%2BRdQBqhc%2FrgvYhg%3D%3D&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20230121T180000Z&X-Amz-SignedHeaders=host&X-Amz-Expires=21600&X-Amz-Credential=ASIAZH6WM4PLV2UG5VZM%2F20230121%2Feu-central-1%2Fs3%2Faws4_request&X-Amz-Signature=f9b552a61f2b7b26b8a3cdd4fa3f8ea6a86e830f22edbbdbb5d22b7ed7e094a5

Disclaimer

Work during the project:

Nikolay Todorov 530022: project plan documentation; hardware unit testing; PCB schematic; PCB assembly;

Ali Mobini 528592: project plan documentation; hardware unit testing; software; breadboard;

Nikolay Georgiev 528561: project plan documentation; hardware unit testing; high level hardware; PCB schematic; GUI;

Catalin Chiprian 528729: project plan documentation; hardware unit testing; high level software; low level software; LCD module implementation;

Victoria Shvets 539968: project plan documentation; hardware unit testing; unit testing simulation; low level hardware; integration testing and simulation;

Report work:

Nikolay Todorov 530022: Explanation of the schematic and relation with the block diagram:

Ali Mobini 528592: final touch ups and software edit

Nikolay Georgiev 528561: Choice of electronic components and their functioning explained; Calculation of component values;

Catalin Chiprian 528729: user requirements; flowchart functions of High Level Flowchart identifying as comment in the code;

Victoria Shvets 539968: unit, integration and system testing description; testing results; formatting;