INSTALOCK

23/11/2023 - 18/1/2024

Vault Making

Instalock is a project about making a state of the art safe with cutting edge technology. Instalock group is made by extremely skilled top tier professionals which will ensure the result will be of highest quality. We will be following the V-Model to ensure a rigorous and efficient development cycle. Additionally we will be using the MoSCoW method to efficiently set goals.

Table of Contents:

Chapter 1: Introduction:

Planning

Members

Aritra Biswas
Project Lead
Email: 545606@student.saxion.nl
Mobile Phone: +31 6 687656965

Jimmy Tithphit
Hardware Specialist
Email: 518275@student.saxion.nl
Mobile Phone: +31 6 623420691

Ivan Hu
Software Specialist
Email: 549010@student.saxion.nl
Mobile Phone: +31 6 024969696

Trung Nghia Hoang
Software Specialist
Email: 553429@student.saxion.nl
Mobile Phone: +31 6 687655775

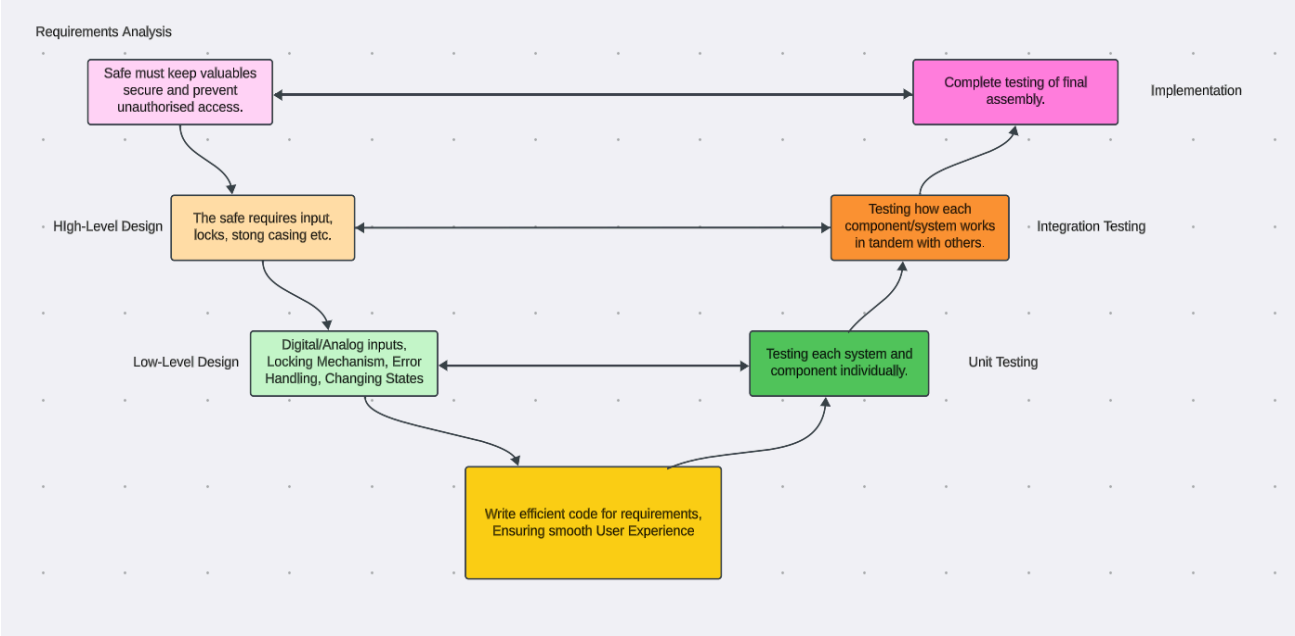
Mathieu Geiller
Hardware Specialist
Email: 553016@student.saxion.nl
Mobile Phone: +31 6 69696969

Work distribution

During the planning phase one of the most important ways to efficiently complete the project is to divide the tasks between team members and give specific roles to each team member. We decide to divide the group into two subgroups. One of the subgroups will be focused on the hardware design of the project while the other subgroup will be working on the software design.

A	B	C	D	E	F
	Mathieu	Jimmy	Nghia	Aritra	Ivan
Altium					
Coding					
Report Writing					
Hardware					
Plan Sketch/Breadboard/TinkerCAD					

V-model:



- **Requirements Analysis**: The safe has to be secure and prevents access from outside without permission
- **Implementation**: The final assembly of all components and final testing
- **High-level Design:** The safe requires components for running, a place for user to insert password and a locking mechanics
- **Integration Testing**: The testing of each components together to make sure everything works with each other without any conflict
- **Low-level Design**: The mechanical requirement for each component and its purpose, such as the Locking Mechanism, how to handle errors or the Digital/Analog Inputs
- **Unit Testing:** After the integration testing, we disassemble them to test them out alone to make sure everything works without defects
- Finally, we put in the code to connect everything with each other to create a safe that ensures the best experience for users.

MoSCoW:

MUST:

- Inputs (both analog and digital) to access safe: Both options are available for the customers to choose via purchasing, 7 Segment Display.
- Locking mechanism: Every safe needs one.
- Strong casing: Hard Case/Shell to give the best protection to the contents inside the safe no matter the pressure from outside (pressure, fire, wind, water).
- maximum input tries (3 or 5): attempts to open the safe.

SHOULD:

- Fail safe system: with a fail safe system is to protect what is inside the safe no matter what happens (for example: if one of the components in the safe is not functioning the other part of the safe can take control of it for a period of time).
- Master key unlock (if required) : basically what this means is you can have more

than one key to unlock the safe (the safe is customizable ex: Bank Safe).

COULD:

- Various outputs: Buzzer for aesthetics, OLED Display

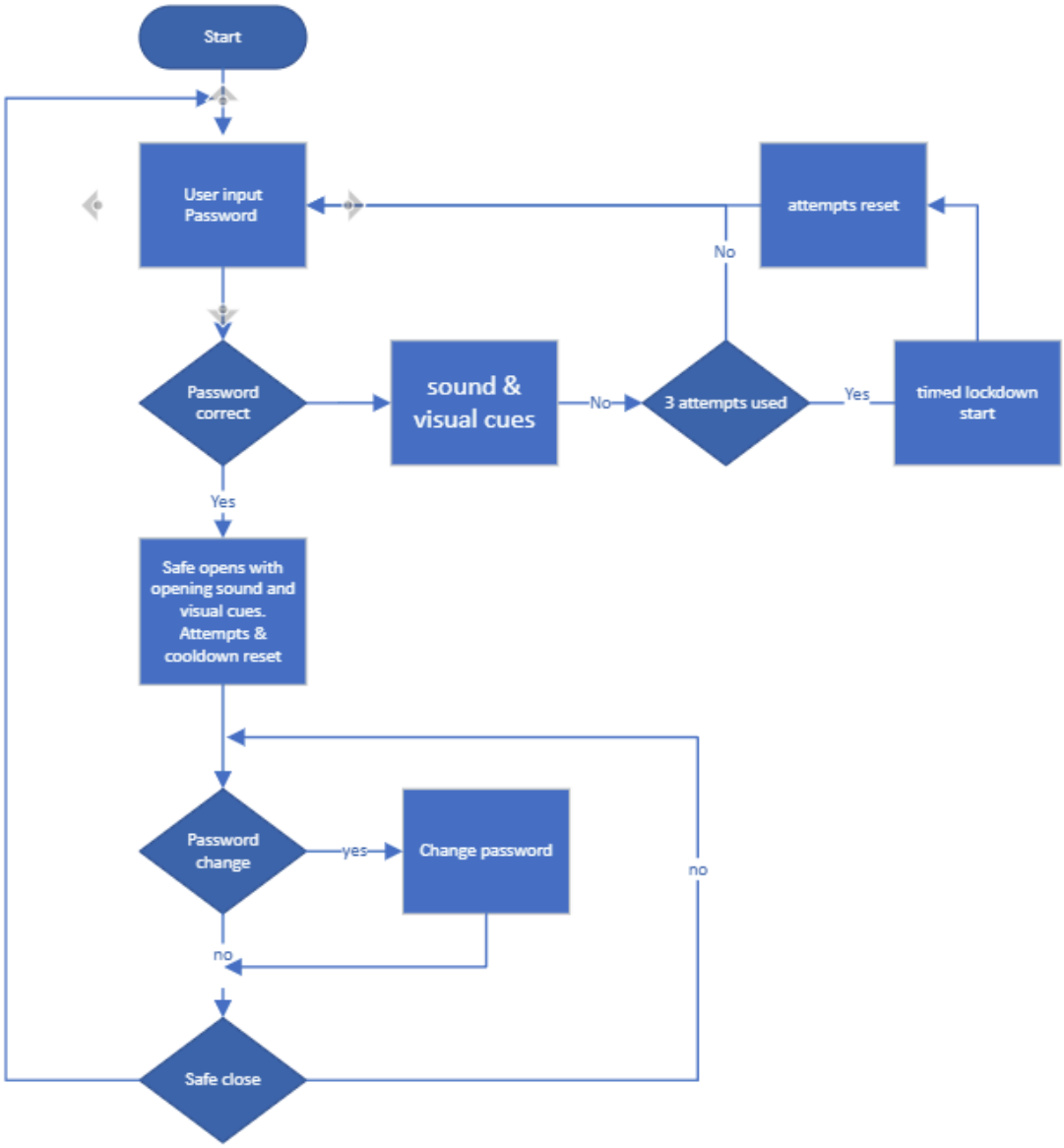
WOULD:

- Other input methods :
  Fingerprint Sensor / Voice recognition / Face ID

Chapter 2: Functional Design

Software:

High level design:

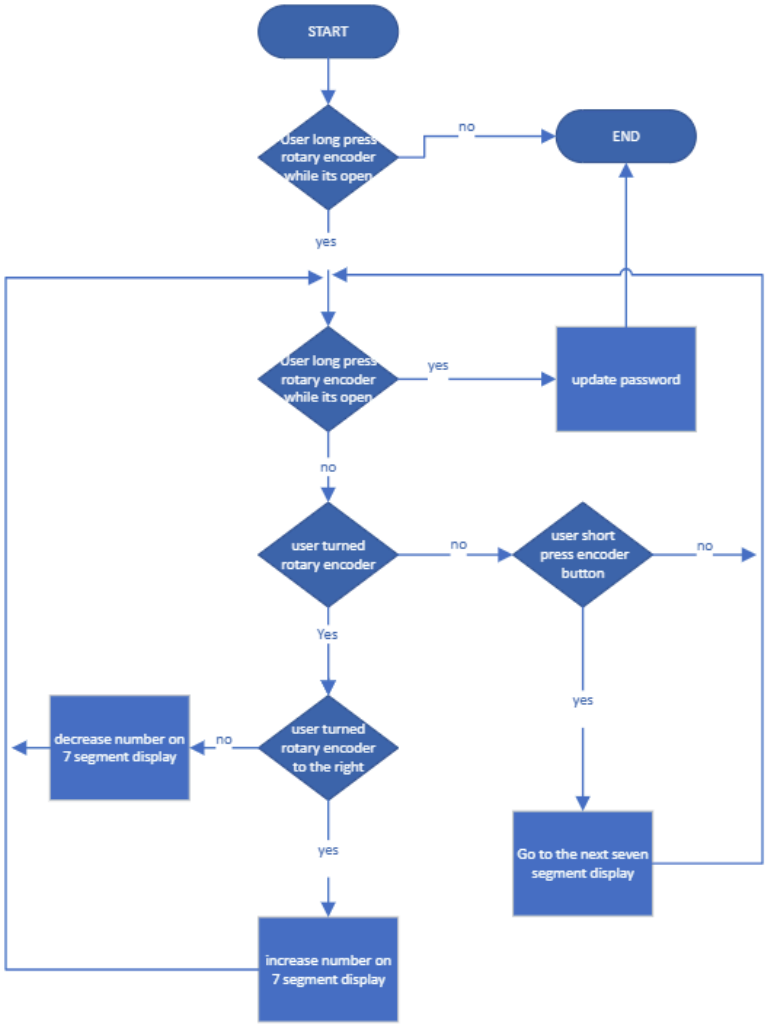


The flowchart illustrates how the process would work out step by step. It vaguely explained the whole process would work without going into the technical detail about how each step would achieve it. The process shows the main process of opening the safe such as password checks and closing the safe. It also shows the additional

features like a customizable password and theft protection with attempts and lockdowns.

Low Level Design Flowchart:

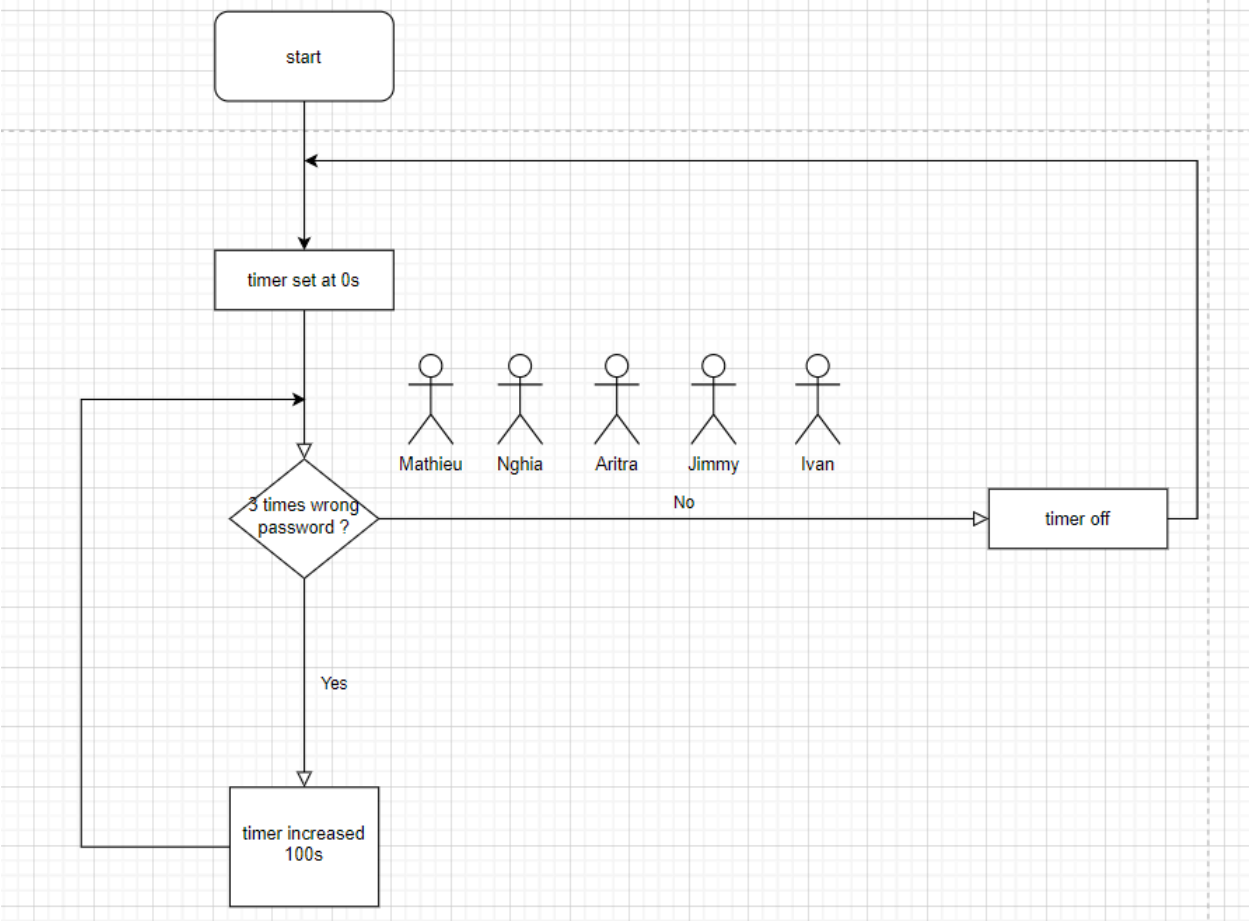
Password change:



This flowchart illustrates how the password change should function for the safe. If the state of the safe is unlocked it checks if there is a long press then the next long press is

used to input the new password in. To choose the password it uses the same function as the normal rotary encoder

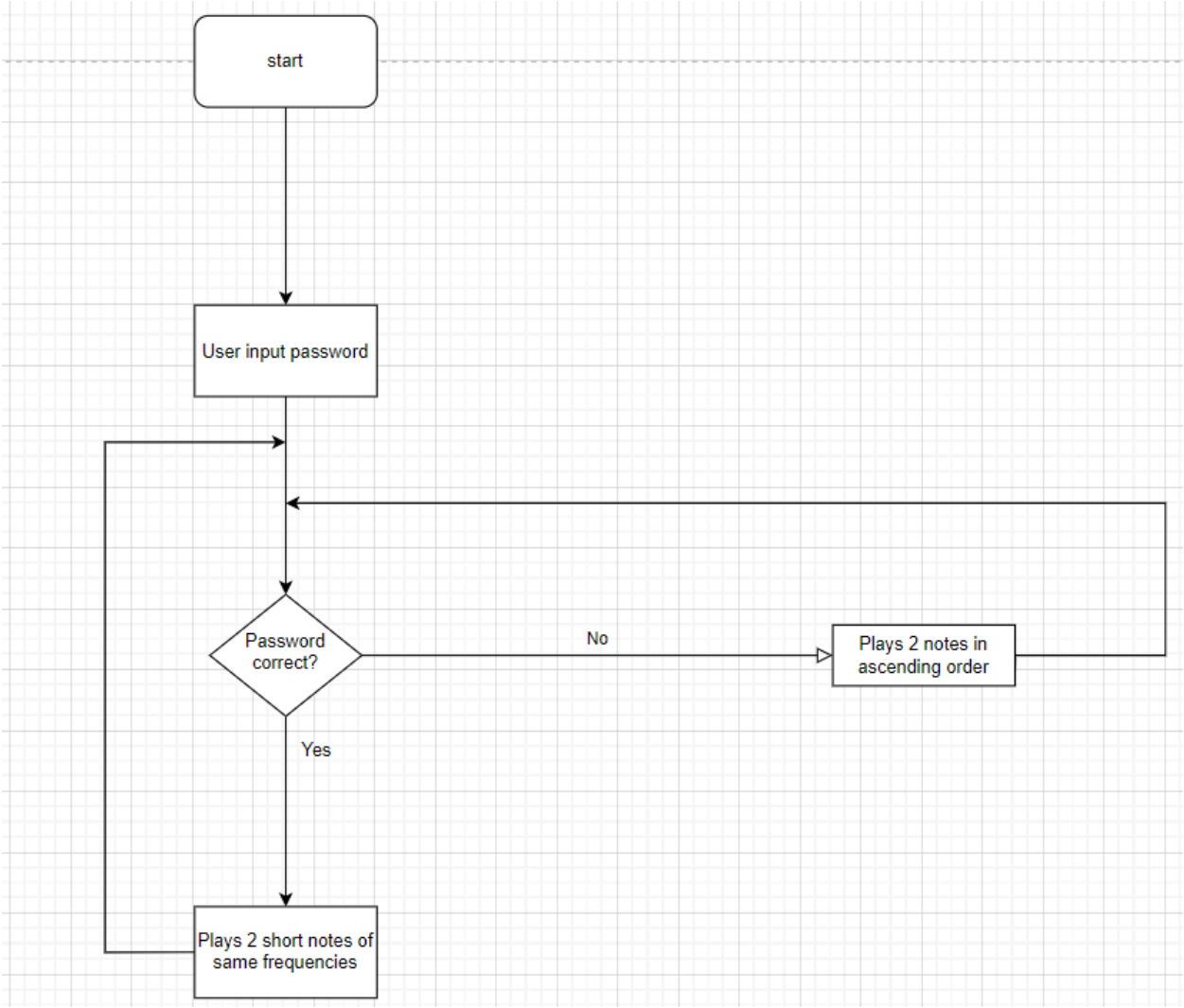
Timer flowchart:



The flowchart checks the condition of the timer. The timer will first be set at 0 second, and it will check if the user has input the wrong password enough time. If the wrong password is typed in 3 times, the timer will increase 100 seconds until the user can input the password again, and it will keep on adding another 100 seconds if the user has their password input wrong again. If the user got their password right, the timer will be reset back to 0 second.

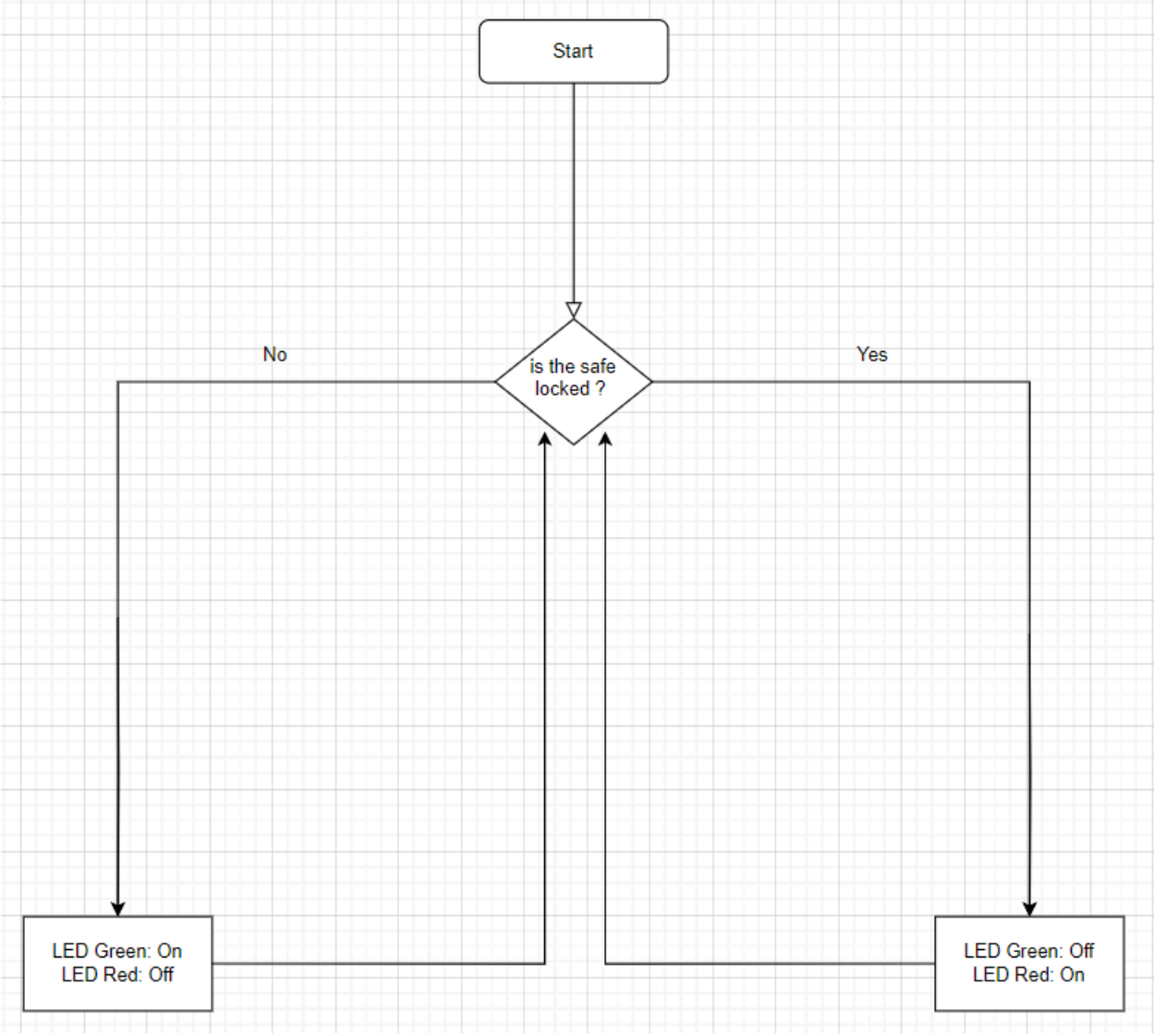
Components Flowcharts:

Buzzer control flowchart:



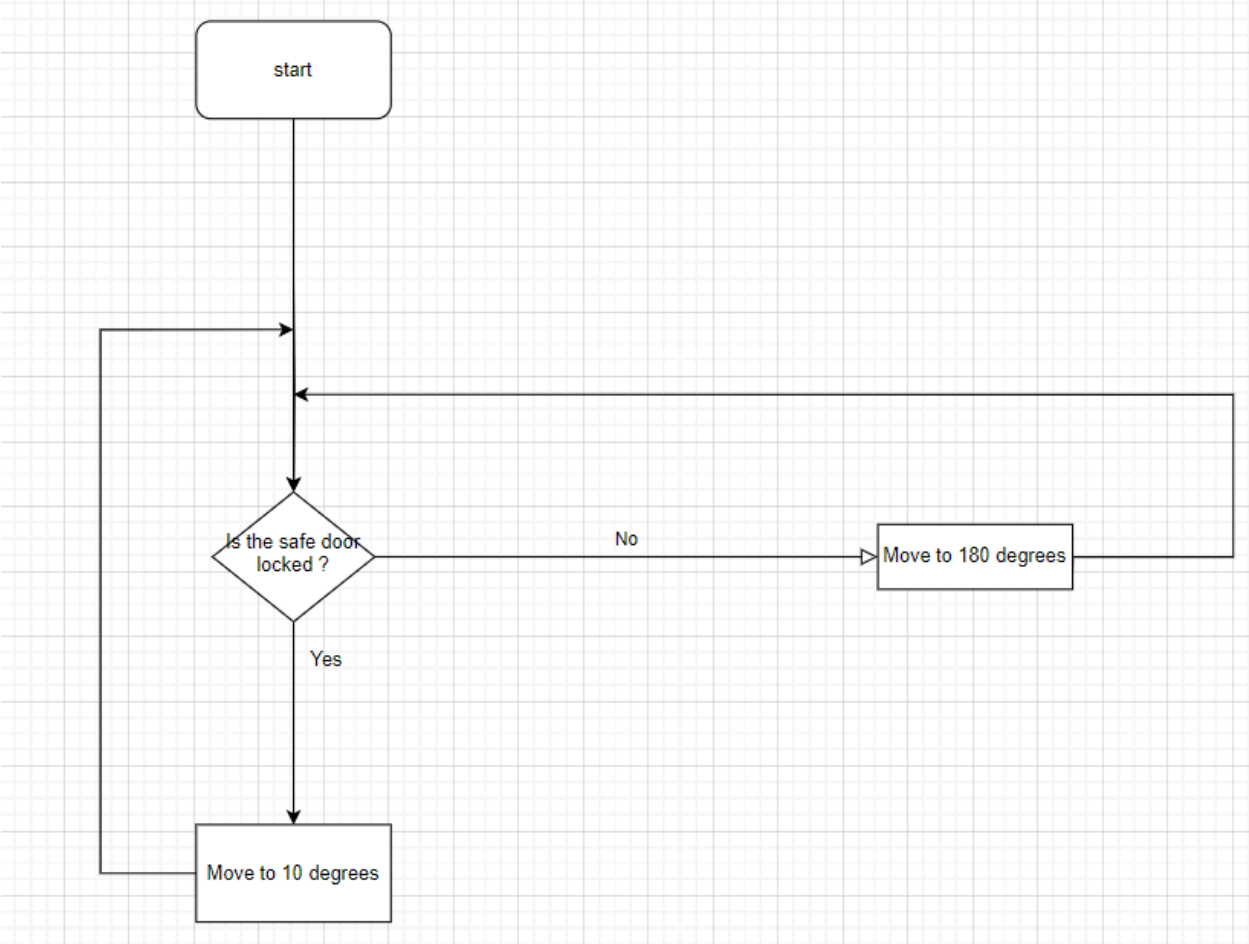
The flowchart describes how the buzzer would function in the safe. It will first be checked after the user has their password input on the safe. If the password is correct, the buzzer will then play 2 short notes of the same frequencies, and notify the user that their password is correct. If the password is incorrect, the buzzer will then plays 2 different notes in ascending order, notify the user that their password is incorrect

LED flowchart:



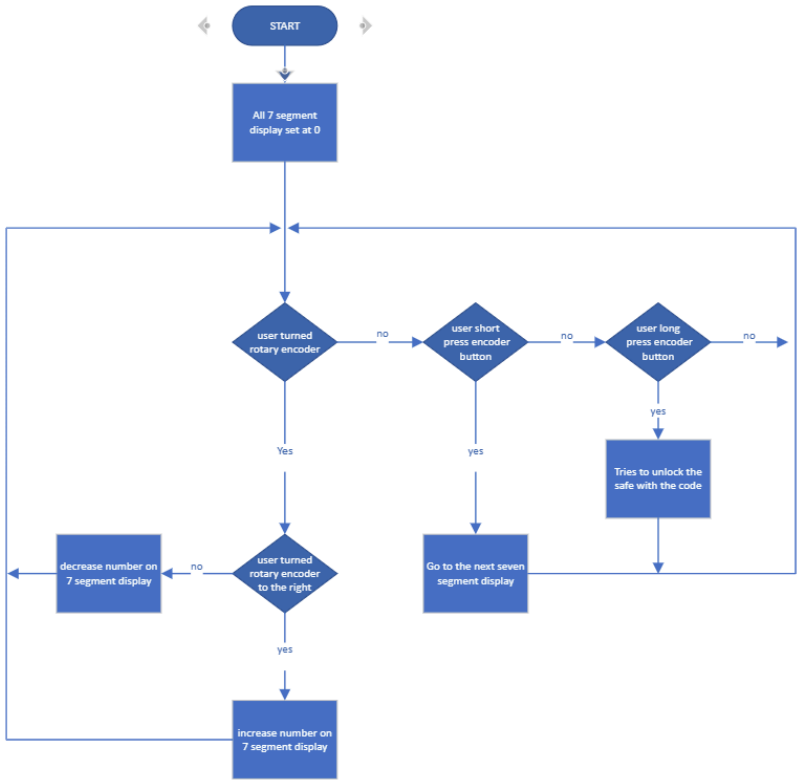
The flowchart illustrates how the 2 LEDs would work in the safe. The process will check if the safe is locked yet. If it is locked, the green LED will be on and the red LED will be off simultaneously until the state of the safe is changed. If it is unlocked, the green LED will be turned off and the red LED will be turned on for the time the safe is left unlocked.

Servo flowchart



The flowchart demonstrates how the Servo would work. It will first check if the safe door is closed yet, if it has, then the Servo will stay at 10 degrees to keep the safe locked. Otherwise, the Servo will move to 180 degrees to open up the safe.

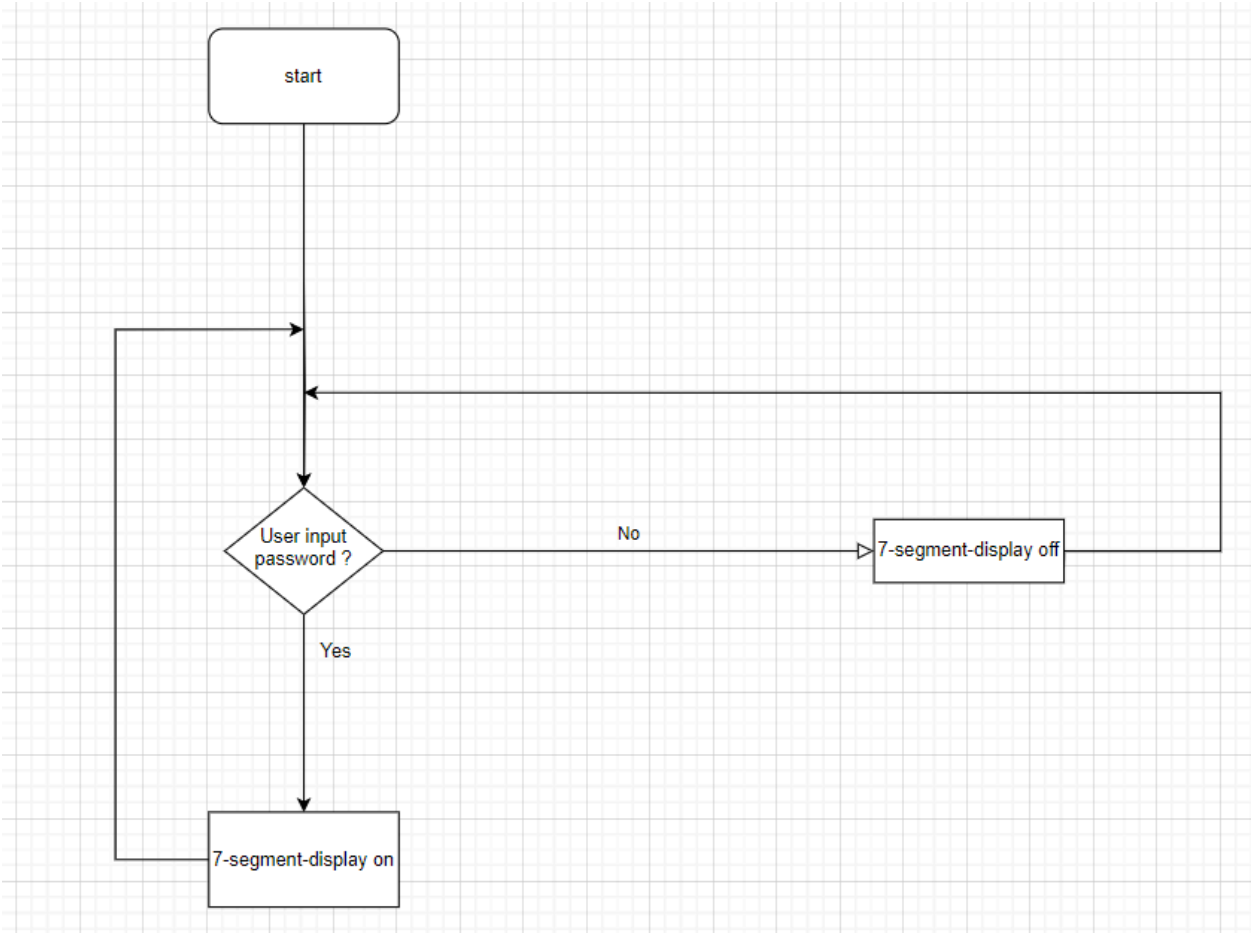
Rotary encoder flowchart:



The flowchart above demonstrates how the rotary encoder would work out in the safe. The 7-segment display will be set at 0 in the beginning, waiting for the user to turn the rotary encoder. Once the user turns the rotary encoder to the left, the number displayed on the 7-segment will increase from 0 to 9 On the other hand, if the user turns the rotary decoder to the right, the number displayed on the 7-segment will then decrease from 9 to 0. If the user does not turn the rotary decoder, it will then check if they have made a short press. If they have, then it will move to the following 7-segment display; if not, the program checks if they have made a long press. The long will try to unlock the safe with

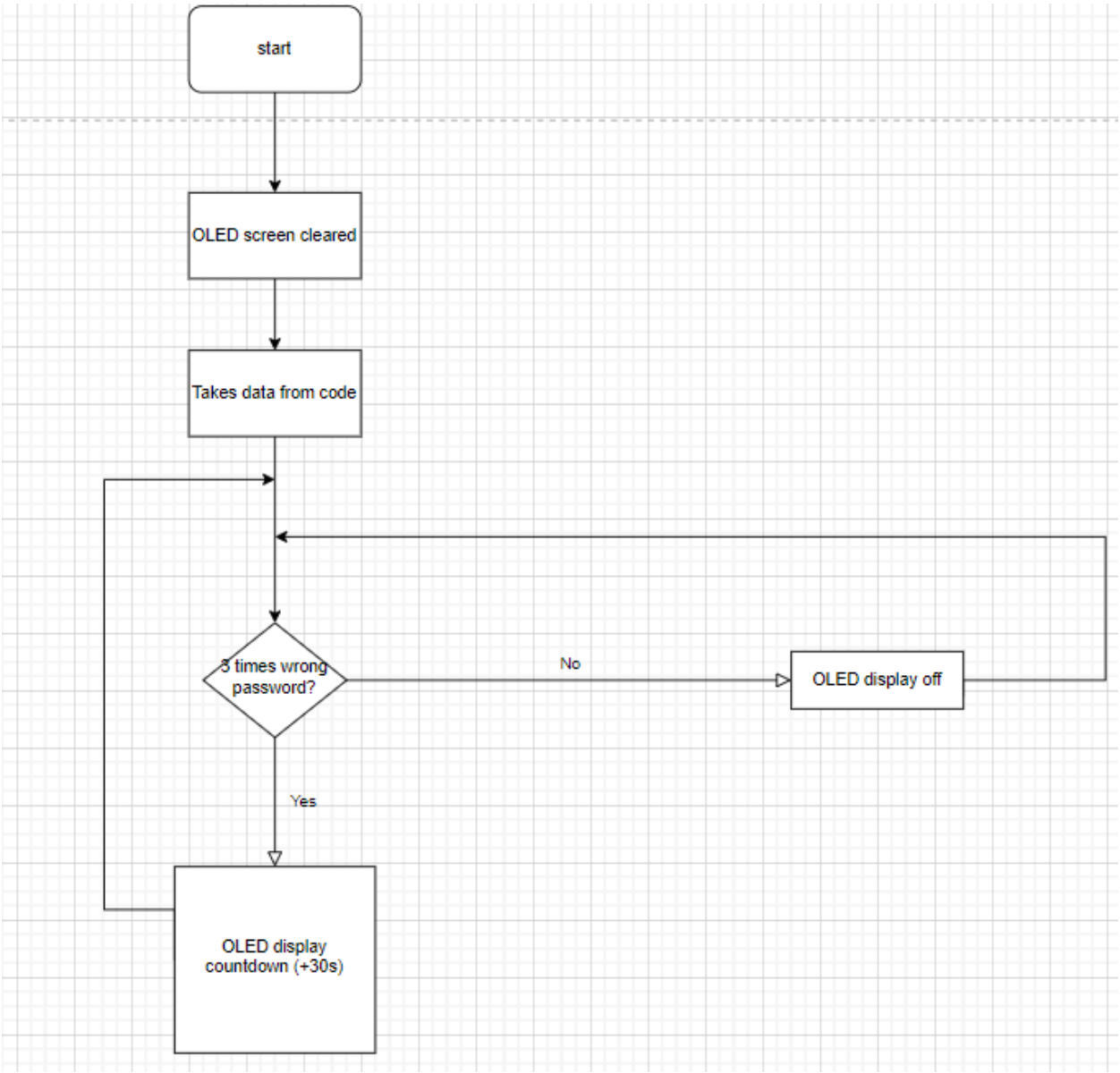
the imputed values. After finishing any action the program will loop itself waiting for another action to be taken..

7-segment display



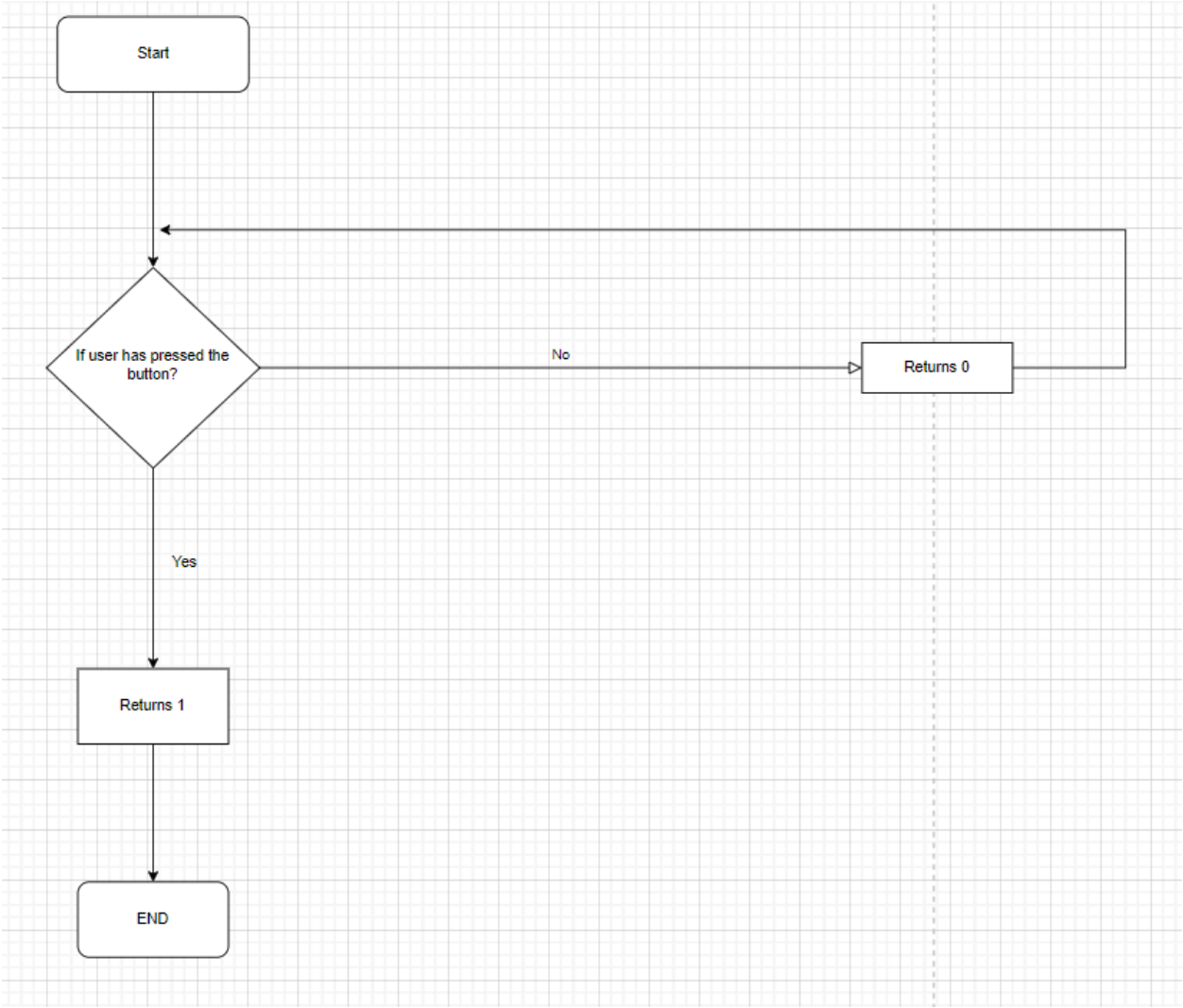
The 7 segment displays work as the place that displays the numbers that the user put in. When the user chooses a number for their passcode and inserts it, that number will be displayed on the 7-segment display. When the user doesn't do anything, the 7-segment display will be off.'

OLED screen flowchart



The OLED screen works as a place to present the countdown times whenever the user has input the wrong code 3 times. When the user got their password wrongly inserted 3 times, the countdown will be displayed on the OLED screen for 100s. After 100 seconds, the user could try again. If they got it wrong 3 times again, the countdown will increase by another 100 seconds until the user got the password correct, then the countdown will be reset to 0 again.

Button flowchart



The button checks if the user has pressed it or not. If the user has not pressed it, the button returns 0 and loop again to check if the user has pressed it again. If the user has pressed it, the button then returns 1 and ends the process.

General:

Functional Design Testing

Integration testing is the procedure of testing multiple components in tandem with each other while making sure no errors occur. This procedure is done after a unit test. This is because once we know about how a component works by itself, it becomes crucial to know if it works with other components.

For this project, it can be grouped into two major tests for integration.

Integration Testing Procedure

1. Input Gathering

a. Gathering Input

The first step after a successful test of the units separately is to see whether an entire input can be gathered from all inputs combined since there are multiple ways of inputting data in our project. An input in our case is a passcode taken from the user.

b. Reading Input

After gathering the input we must make sure that the safe understands the given input and there are no errors or discrepancies in reading and understanding input.

2. Processing

a. Registering Information

The safe must be able to register a given information from the user via any input given to the safe. The inputs such as password, attempts and lockdown timer are saved in the memory.

b. **Software Processing**

After fetching the inputs from memory, the software written should process it and return an output which is going to binary in the case of a safe i.e. locked and unlocked.

3. Output & Result

a. Visible Output

After processing data and coming to a conclusion, the output must be visible to the end user and all relevant data must be displayed to the user as well. In the case of the safe there are visual cues and auditory cues to aid a user in understanding the state of the safe.

b. Registering Output

Alongside displaying output to the user via the means of hardware, the software must also register the outcome of the output and if there are further steps to be taken; such as if wrong attempts fail, a punishment occurs. If not then it returns to the state of input gathering again.

Chapter 3: Technical Design

Software:

Descriptions of Functions

Rotary Encoder Turning:

```
C/C++
#define ENCODER_OPTIMIZE_INTERRUPTS

Encoder Rotary(ROT_A,ROT_B); // Declare the class

int read_encoder() {
 static long RotaryNum;
 static long RotaryTurn;
 RotaryTurn = Rotary.read();
 if (RotaryTurn != RotaryNum) {
  if ((RotaryTurn % 4) == 0) {
   if (RotaryNum < RotaryTurn){
      RotaryNum = RotaryTurn;
    return 1;}
   else{
     RotaryNum = RotaryTurn;
     return 2;}
  }
 }
 return 0;
}
```

Rotary Encoder Turning Function: This function is used to check if the rotary encoder is turned. The function starts by initializing two variables. One of the variables changes when the rotary is turned and the other keeps the previous state. The if statement checks if they are different and returns 1 or 2 depending on the direction turned. The value is checked every 4 times as the rotary encoder changes the value 4 times for each dent. Return 0 is used to reset the return value. To deal with the debouncing we use the “#define ENCODER_OPTIMIZE_INTERRUPTS”

which is an arduino uno optimized interrupts. This eliminates the slight overhard that the more dynamic “attachInterrupt()” creates.

Rotary Encoder Pressing Function:

```C/C++
int rotary_button(){
  static unsigned long time, time2;
  long elapsed_time = 0;
  delay(5);
  if (digitalRead(ROT_BTN) == LOW) {
   time = millis();
   while (!digitalRead(ROT_BTN)) {
   }
   time2 = millis();
   elapsed_time = time2 - time;
  }
  if (elapsed_time > 1500) {
   return 1; // long
  } else if (elapsed_time > 0) {
   return -1; // short
  } else {
   return 0;
  }
}
```

This function does two things. The first is to calculate the elapsed time between the pressing of the rotary encoder button and the release of the button. This is done by starting a millis() clock upon press of a button and while the button is being held, it enters a loop without anything to execute and when the button is released, another millis() clock is started. Thus the elapsed time comes out to be the difference between the two clocks. In this function, if elapsed time is greater than 1500 ms, it returns 1, -1 if more than 0 but less than 1500 ms and 0 in all other cases.

LED Control Function:

```
C/C++
/// Controls state of LEDs that display what state the safe is in
/// Sets value of each LED (high/low) depending on what state is
/// @param state is the state of safe (UNLOCKED/LOCKED)
void led_control(int state){
  if (state == UNLOCKED) {
    digitalWrite(RED_LED, LOW);
    digitalWrite(GREEN_LED, HIGH);
  } else {
    digitalWrite(RED_LED, HIGH);
    digitalWrite(GREEN_LED, LOW);
  }
}
```

This function controls the LEDs that make the user aware if the safe is unlocked or locked. Using the state parameter, the right LED turns on. The logical flow is if the state of the safe is locked, the red LED turns on and the green LED is off, likewise if the state of safe is unlocked, the opposite happens.

Buzzer Control Function:

```C/C++
/// Controls buzzer output based on state (LOCKED/UNLOCKED)
/// Plays two short notes of same freq when state = LOCKED
/// Plays two notes in ascending order when state = UNLOCKED
/// @param state is state of safe
void buzzer_control(int state){
  if (state == UNLOCKED) {
    tone(BUZZER, 500);
    delay(100);
    tone(BUZZER, 600);
    delay(100);
    noTone(BUZZER);
  } else {
    tone(BUZZER, 440);
    delay(100);
    noTone(BUZZER);
    delay(5);
    tone(BUZZER, 440);
    delay(100);
    noTone(BUZZER);
  }
}
```

This function controls the buzzer. Using the tone() builtin function, a sound can be produced on the buzzer. Using the parameter of state, two different sounds emerge from the buzzer. If the state of the safe is unlocked, then 2 short ascending notes are played by the buzzer; if the state is locked then 2 short bursts of notes of the same frequency are played. The noTone() function is used to signify silence. delay() functions are used to extend periods of tone or noTone().

Servo Control Function:

```
C/C++
/// Controls the servo motor
/// Moves to 180 degree when UNLOCKED and 10 when LOCKED
/// @param state is state of safe
void servo_control(int state){
  if (state == UNLOCKED){
    s1.write(90);
  } else {
    s1.write(10);
  }
}
```

This function controls the servo motor. Using the Servo.h library, it becomes easier to control the servo. An initial declaration of the class is made, and an attach() method is written in the setup. Using the write() method, an angle is passed in as a parameter and the servo then turns to the given angle.

Open Close Function:

```C/C++
/// Checks whether safe is open or closed based on reading button state
/// @returns true (1) if safe is CLOSED and false (0) if OPEN

bool open_close_button(){
 const unsigned long time = millis();
 static int oldState = 1;
 static unsigned long oldTime = 0;
 const int newState = digitalRead(BUTTON);


 if (newState != oldState && (time - oldTime) > DEBOUNCE_TIME){
  oldTime = time;
  oldState = newState;


  if (newState == LOW) return true;
 }


 return false;
```

This function simply reads a button press from a button. It avoids any bouncing of the button by using states and a debounce time of 50 ms. The function reads a state from the button and if it is different from the previous state, then a true is returned, else a false.

Check Passcode Function:

```
C/C++
/// Checks if passcode entered is same as correct passcode
/// For loop iterates through two arrays containing user passcode and real
passcode
/// @returns true if passcode correct else false
bool check_passcode(int array1[3], int array2[3]){
  for (int i = 0; i < 3; ++i){
    if (array1[i] != array2[i]) return false;
  }
  return true;
}
```

This function verifies if a passcode entered is correct or incorrect. This is done by using two arrays passed as parameters. A for loop iterates through both arrays and checks each element one by one to verify if they all are the same. If all 3 elements are the same the function returns true else false.

Seven Segment Selection of Display:

```C/C++
/// Selection of seven segment display is made based on decimal point
/// Cycles between 1,2,3 based on rotary encoder button press (short)
void seven_seg_select(int index){

 if (index == 0){
  digitalWrite(DP1, LOW);
  digitalWrite(DP2, HIGH);
  digitalWrite(DP3, HIGH);
 } else if (index == 1) {
  digitalWrite(DP1, HIGH);
  digitalWrite(DP2, LOW);
  digitalWrite(DP3, HIGH);
 } else if (index == 2){
  digitalWrite(DP1, HIGH);
  digitalWrite(DP2, HIGH);
  digitalWrite(DP3, LOW);
 }
}
```

This function operates the visual selection of the seven segment display using the decimal point. Using the index parameter, which is from the loop() functions and it will have only a value of 0, 1 or 2, we can control which decimal point of the 3 seven segment displays will light up indicating that a display is chosen. Here the digitalWrite() function is used to control the 3 decimal points.

Timer OLED Display Function:

```
C/C++
/// Displays a seconds counter on oled display
/// Latch and Latch Clear built into the function
/// @param seconds is for number of seconds remaining to be displayed on the
oled
void timer_oled_display(int seconds){
  while (seconds >= 0){
    display.clearDisplay();

    display.setTextSize(2);
    display.setTextColor(SSD1306_WHITE);
    display.setCursor(0, 0);
    display.print("Locked for");

    display.setTextSize(4);
    display.setCursor(40, 30);
    display.print(seconds);

    display.display();

    delay(1000);
    seconds--;
  }
```

This function controls the OLED displays and displays a timer. Using display.clearDisplay() and display.Display() which act like a latch, a message can be displayed on the OLED display. The message declared here is the number of seconds remaining after 3 incorrect attempts have been made. In this function a while loop is used to count down from the seconds() parameter. A text size is set, text font color is set to white, and the coordinate of where the content will be printed on the oled display is given using respective methods from the Adafruit libraries.

Attempts OLED Display Function:

```
C/C++

/// Displays attempts remaining on display
/// Latch and Latch Clear built into the function
/// @param attempts is for the number of attempts remaining to be displayed on
the oled.
void attempts_oled_display(int attempts){
  display.clearDisplay();

  display.setTextSize(2);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0, 0);
  display.print("Attempts:");

  display.setTextSize(4);
  display.setCursor(40, 30);
  display.print(attempts);

  display.display();}
```

This function controls the OLED displays and displays a timer. Using display.clearDisplay() and display.Display() which act like a latch, a message can be displayed on the OLED display. The message declared here is the number of seconds remaining after 3 incorrect attempts have been made. In this function a while loop is used to count down from the seconds() parameter. A text size is set, text font color is set to white, and the coordinate of where the content will be printed on the oled display is given using respective methods from the Adafruit libraries

EEPROM Function:

```
C/C++
///allow user to store password in EEPROM
///@param address The address where password is store
///@numbers The amount of numbers (password) stored in EEPROM (3 in this case)

void writeArrayIntoEEPROM(int address, byte numbers[])
{
 int addressIndex = address; //Address will be updated in each loop
 for (int i = 0; i < 3; i++)  //A loop runs 3 times to store 3 separate values of
the password
 {
  EEPROM.write(addressIndex, numbers[i]); //Store the value into 1 address
  addressIndex += 1; //Shift to another address to store another value
 }
}
```

- The writeArrayIntoEEPROM function allows users to store their password into the EEPROM.
- There are 2 parameters inside it, the address is where the password will be stored, and the numbers array is the passcode itself.
- There will be a loop that runs 3 times to store all 3 of the passcode value to it, the address will also be updated through each loop to store in separate addresses.

```
C/C++
///function that reads the password stored in EEPROM
///@param address The address where password is store
///@numbers The amount of numbers (password) stored in EEPROM (3 in this case)
void readArrayFromEEPROM(int address, byte numbers[])
{
 int addressIndex = address;  //Address will be updated in each loop
 for (int i = 0; i < 3; i++)  //A loop runs 3 times to reads 3 separate values store
on 3 separate address of the password
 {
  numbers[i] = (EEPROM.read(addressIndex));  //Reads the value of an address
  addressIndex += 1; //Shift to another address to reads another value
 }
}
```

- The readArrayFromEEPROM function allows the computer to read the values that were stored in the EEPROM
- There are 2 parameters inside it, the address is where the password will be stored, and the numbers array is the passcode itself
- There will be a loop that runs 3 times to read all 3 of the passcode value to it, the address will also be updated through each loop to store in separate addresses.

```
C/C++
void setup() {
 Serial.begin(9600);
 const int ADDRESS = 5;  //Original address

 byte numbers[3] = { 3, 8, 1};  //Passcode
 writeArrayIntoEEPROM(ADDRESS, numbers);  //Store the passcode into EEPROM

 byte newNumbers[3];
 readArrayFromEEPROM(ADDRESS, newNumbers);  //Read the passcode in the
addresses

 for (int i = 0; i < 3; i++)
 {
  Serial.println(newNumbers[i]);  //Print out the passcode
 }
}
```

- The setup allows us to print out the value (password) that were stored in the EEPROM
- We declare an address where we will store our first number of the password in, and an array of number for the user to put their passcode in
- We then store the passcode with the writeArrayIntoEEPROM function, and then read it in the readArrayFromEEPROM function, and finally we print them out again.

Shiftout Function:

```
C/C++
int DS_pin = 4;
int STCP_pin = 3;
int SHCP_pin = 2;


const int dec_digits [10] {1, 79, 18, 6, 76, 36, 32, 15, 0, 4};


void sr_display_num(int num1, int num2){
 digitalWrite(STCP_pin, LOW);
 shiftOut(DS_pin, SHCP_pin, LSBFIRST, dec_digits[num2]);
 shiftOut(DS_pin, SHCP_pin, LSBFIRST, dec_digits[num1]);
 digitalWrite(STCP_pin, HIGH);
}


void setup() {
 pinMode(DS_pin, OUTPUT);
 pinMode(STCP_pin, OUTPUT);
 pinMode(SHCP_pin, OUTPUT);
}


void loop() {


 sr_display_num(3, 4);
}
```

- The first lines in the code used to define the pins.
- In the line const in dec_digits, we are taking the numbers from 0 to 9, and converting them to binary, so for example to display 0 on the seven segment display, we would need

to light up A(1), B(1), C(1), D(1), E(1), F(1), G(0), so this gives us 01111110 in binary, then we reverse binary this and get 00000001, giving us the value of 1 for 0.

Conversion table:

Number	Binary	Reverse Binary	Decimal Value
0	0 1 1 1 1 1 1 0	0 0 0 0 0 0 0 1	1
1	0 0 1 1 0 0 0 0	0 1 0 0 1 1 1 1	79
2	0 1 1 0 1 1 0 1	0 0 0 1 0 0 1 0	18
3	0 1 1 1 1 0 0 1	0 0 0 0 0 1 1 0	6
4	0 0 1 1 0 0 1 1	0 1 0 0 1 1 0 0	76
5	0 1 0 1 1 0 1 1	0 0 1 0 0 1 0 0	36
6	0 1 0 1 1 1 1 1	0 0 1 0 0 0 0 0	32
7	0 1 1 1 0 0 0 0	0 0 0 0 1 1 1 1	15
8	0 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0	0
9	0 1 1 1 1 0 1 1	0 0 0 0 0 1 0 0	4

- The conversion table shows how we get the decimal values, this will then be stored inside the array.
- By using shiftout, when a number is chosen, it will first go to the dec_digits[i] array, the position of i, this will then convert to binary and send it serially to the seven segment display.
- Then the sr_display_num, displays the numbers that the user chooses by turning the rotary encoder.

Binary Coded Decimal Function:

```
C/C++
/// Loop that runs through 1, 2, 4 and 8, as inputs on the BCD
/// Perform a bitwise shift to the right
/// Performs an and gate with 1, this helps us see which is high and which is
low
/// @param is decimal number to display
void bcd_display_num(int number){
  for (int i = 0; i < 4; i++){
    digitalWrite(BCD_PINS[i], (number >> i) & 1);
  }
}
```

In this code, we used a for loop to compare each bit, 1, 2, 4 and 8.

For example, if the number was 5 then, it would first do a bitwise right shift (the symbol for the bitwise shift is >>) by 1. So the binary number for 5 would be 0101 and it would then shift it by 0, so it stays the same. Then it would go through a and gate (symbol for and gate is &), so it compares 0101 and 0001, so it would take the last digit, 1 and 1 = 1.

And it goes again until it comes to 3:

0101 >> 1 = 0010 & 1 = 0

0101 >> 2 = 0001 & 1 = 1

0101 >> 3 = 0000 & 1 = 0

So the result of each comparison is 1010, and we would use this to say HIGH LOW HIGH LOW, so both A and C are HIGH and the rest are LOW. This will then translate to the seven segment display and displaying it.

But before having this code, we had a bigger code where we listed all the outputs down:

```
C/C++

void bcd_display_digit(int digit){
 if (digit == 0){
   digitalWrite(BCD_A, LOW); // 1
   digitalWrite(BCD_B, LOW); // 2
   digitalWrite(BCD_C, LOW); // 3
   digitalWrite(BCD_D, LOW); // 4
 } else if (digit == 1){
   digitalWrite(BCD_A, HIGH);
   digitalWrite(BCD_B, LOW);
   digitalWrite(BCD_C, LOW);
   digitalWrite(BCD_D, LOW);
 } else if (digit == 2){
   digitalWrite(BCD_A, LOW);
   digitalWrite(BCD_B, HIGH);
   digitalWrite(BCD_C, LOW);
   digitalWrite(BCD_D, LOW);
 } else if (digit == 3){
   digitalWrite(BCD_A, HIGH);
   digitalWrite(BCD_B, HIGH);
   digitalWrite(BCD_C, LOW);
```

```
    digitalWrite(BCD_D, LOW);
  } else if (digit == 4){
    digitalWrite(BCD_A, LOW);
    digitalWrite(BCD_B, LOW);
    digitalWrite(BCD_C, HIGH);
    digitalWrite(BCD_D, LOW);
  } else if (digit == 5){
    digitalWrite(BCD_A, HIGH);
    digitalWrite(BCD_B, LOW);
    digitalWrite(BCD_C, HIGH);
    digitalWrite(BCD_D, LOW);
  } else if (digit == 6) {
    digitalWrite(BCD_A, LOW);
    digitalWrite(BCD_B, HIGH);
    digitalWrite(BCD_C, HIGH);
    digitalWrite(BCD_D, LOW);
  } else if (digit == 7){
    digitalWrite(BCD_A, HIGH);
    digitalWrite(BCD_B, HIGH);
    digitalWrite(BCD_C, HIGH);
    digitalWrite(BCD_D, LOW);
  } else if (digit == 8){
```

```
    digitalWrite(BCD_A, LOW);
    digitalWrite(BCD_B, LOW);
    digitalWrite(BCD_C, LOW);
    digitalWrite(BCD_D, HIGH);
  } else if (digit == 9){
    digitalWrite(BCD_A, HIGH);
    digitalWrite(BCD_B, LOW);
    digitalWrite(BCD_C, LOW);
    digitalWrite(BCD_D, HIGH);
  }
}
```

After trying a few loops which weren't working we set it to ChatGPT to debug the code. After some back and forth with ChatGPT on code not working, it suggested a completely different alternate code to this. Seeing that code made use of binary algebra in c++ code, we decided to use it since it was efficient and readable.

We asked ChatGPT to reduce this code and it gave us a line for the code that we wrote. We needed a smaller code as it needed to be stored and also for readability wise.

You

How would you simplify this code?

```
void disp1(int num){ //this method entails making 0-9 combinations using the BCD pins
by sending different outputs to turn each segment on or off.
 if(num == 0)//0000
 {
  digitalWrite(a1, LOW);
  digitalWrite(a2, LOW);
  digitalWrite(a3, LOW);
  digitalWrite(a4, LOW);
 }
 if(num == 1)//0001
 {
  digitalWrite(a1, HIGH);
  digitalWrite(a2, LOW);
  digitalWrite(a3, LOW);
  digitalWrite(a4, LOW);
 }
 if(num == 2)//0010
 {
  digitalWrite(a1, LOW);//0
  digitalWrite(a2, HIGH);//1
  digitalWrite(a3, LOW);//0
  digitalWrite(a4, LOW);//0
 }
```

This was the prompt that we used to simplify and fix our code.

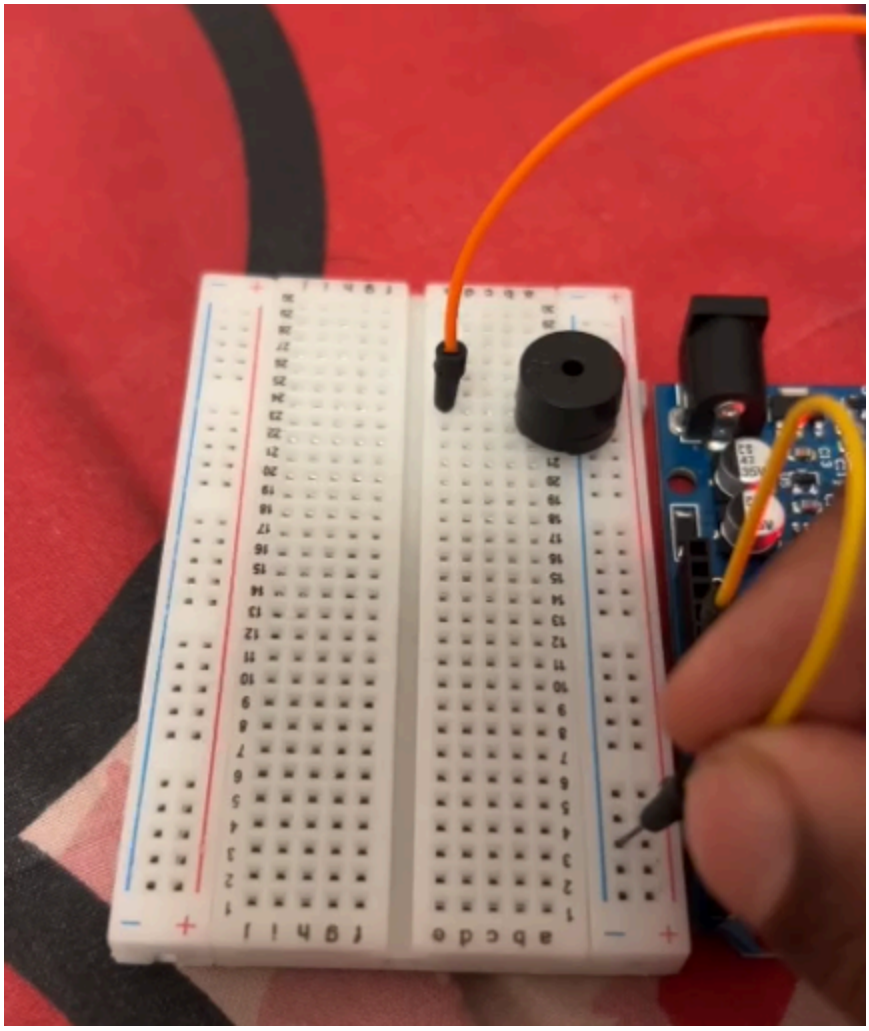
cpp — Copy code

```cpp
// Define pins for display 1 and display 2
int display1Pins[] = {a1, a2, a3, a4};
int display2Pins[] = {b1, b2, b3, b4};

void setDisplay(int pins[], int num) {
  for (int i = 0; i < 4; i++) {
    digitalWrite(pins[i], (num >> i) & 0x01);
  }
}

void disp1(int num) {
  setDisplay(display1Pins, num);
}

void disp2(int num) {
  setDisplay(display2Pins, num);
}
```

Here is the final answer ChatGPT gave us for the prompt.

Hardware:

Buzzer Testing:

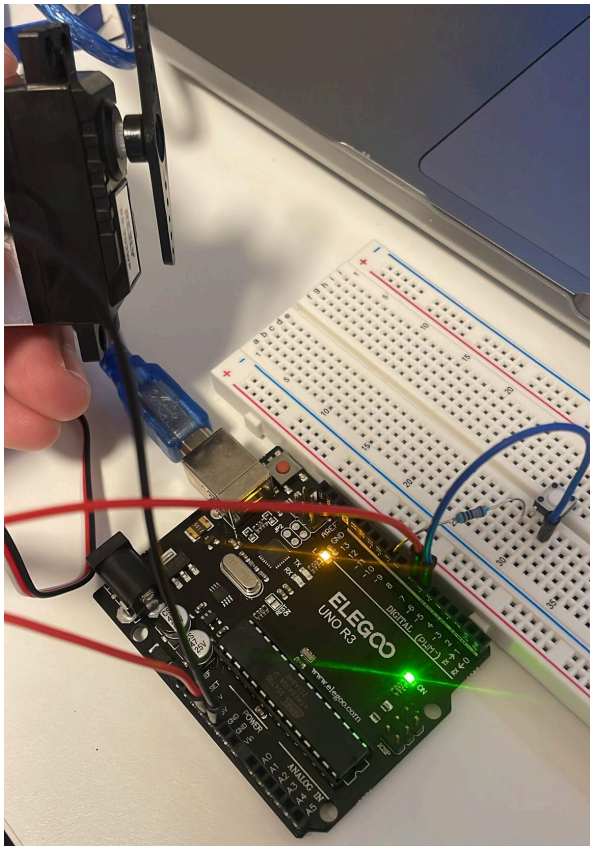


For the buzzer testing, we simply powered it and connected it to the Arduino. Then by running a code, we play a simple buzzing sound that would signify that the safe is closed or open, locked or unlocked, it will play different sounds depending on the situations.

Code for Buzzer Testing:

```
C/C++

#define BUZZER 5


void setup(){

pinMode(BUZZER, OUTPUT);

}


void loop(){

tone(BUZZER, 440);

delay(100);

noTone(BUZZER);

delay(10);

}
```

Button Testing:



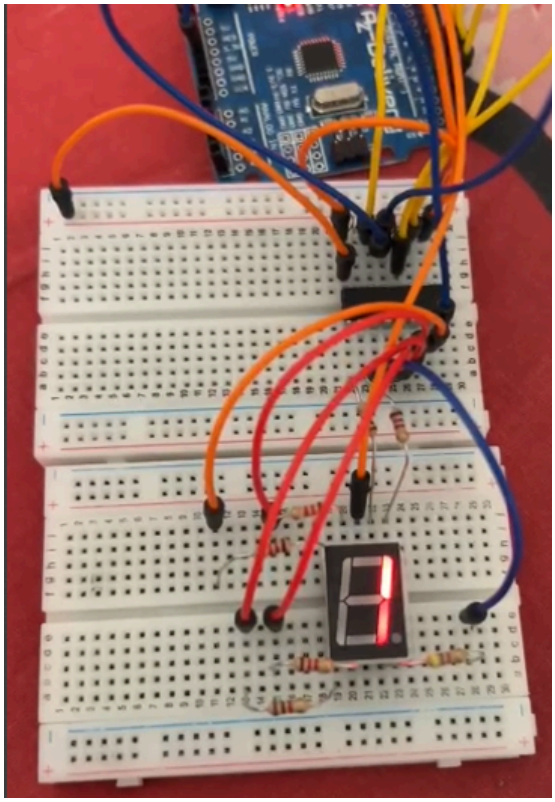
For the button, we run a simple code, and what it does is that when the button is pressed, it displays 1 and if it isn't it displays 0. We could also use an LED, to show when the button would be pressed.

Code for Button Testing:

```
C/C++

#define BUTTON 12

#define DEBOUNCE_TIME 50


bool read_button(){

  const unsigned long time = millis();

  static int oldState = 1;

  static unsigned long oldTime = 0;

  const int newState = digitalRead(BUTTON);


  if (newState != oldState && (time - oldTime) > DEBOUNCE_TIME){

    oldTime = time;

    oldState = newState;


    if (newState == LOW) return true;

  }


  return false;

}

void setup(){

Serial.begin(9600);

pinMode(BUTTON, INPUT_PULLUP);
```

```
}

void loop(){

static int num = 0;

if (read_button())num++;

Serial.println(num);

}
```

Shift Register Testing:



For the shift register, we connected with a 7 segment display. After that we ran a code to test if it was working or not. At first it wasn't working due to wrong connections. After reading the data sheet and understanding more about the shift register, we were able to make the 7 segment display work.

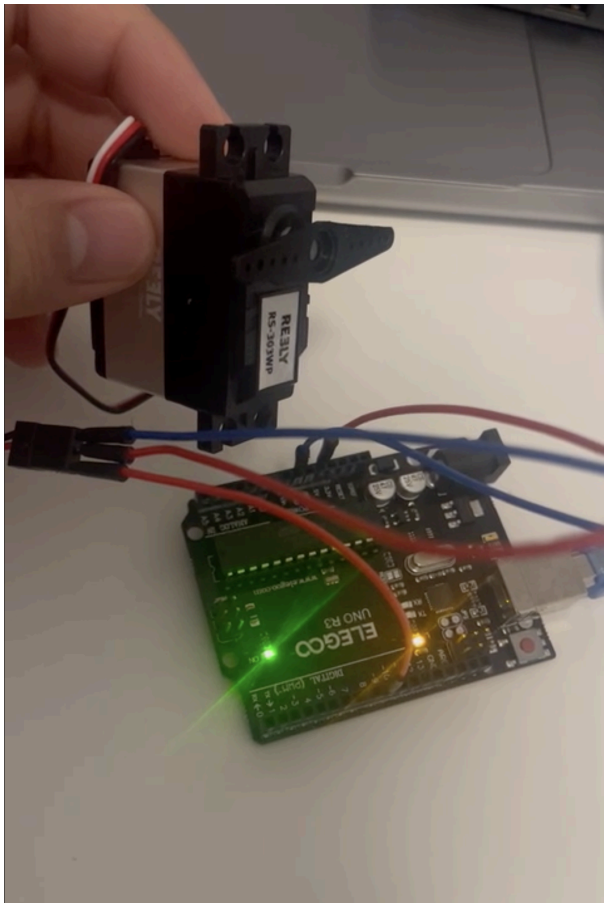
Pin Configuration :

- Pin 1: Output A
- Pin 2: Output B
- Pin 3: Output C
- Pin 4: Output D
- Pin 5: Output E
- Pin 6: Output F
- Pin 7: Output G

- Pin 8: Ground
- Pin 9: QH’
- Pin 10: SRCLR
- Pin 11: SRCLK
- Pin 12: RCLK
- Pin 13: OE
- Pin 14: SER
- Pin 15: QA
- Pin 16: VCC

Code is same as mentioned in Chapter 3 under Shiftout function

Servo Testing:



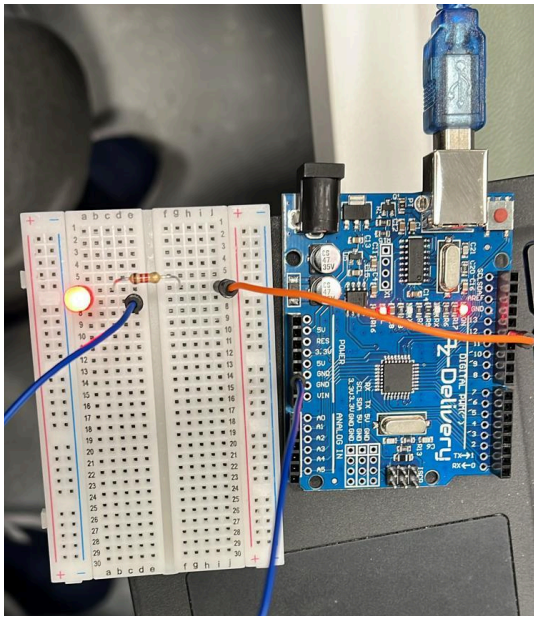
Servo and 5V regulator Testing: For servo testing, Our main concern is " does the servo receive the right amount of voltage". We solve that problem by using a 9v battery and use a 5V regulator to tune it down to 5V.

Code for Servo Testing:

```
C/C++

#include <Servo.h>

#define SERVO 12

Servo s1; // Class declaration
```

```
void setup(){
s1.attach(SERVO);
}
void loop(){
s1.write(180);
delay(1000);
s1.write(0);
delay(1000);
}
```

LED Testing:



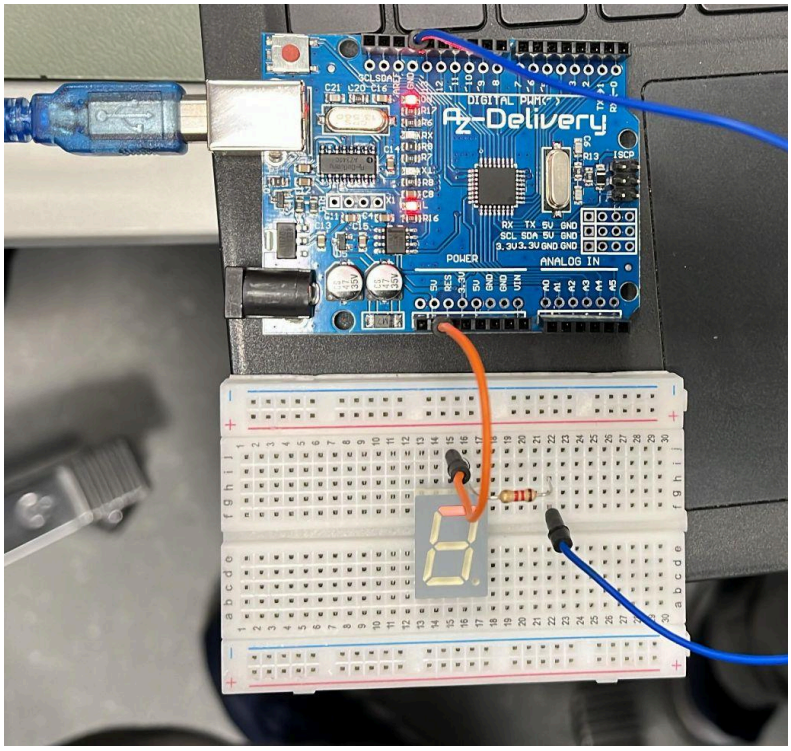
To test the LED, all we did was put either HIGH or LOW in the code to power up to the LED.

Code for LED testing:

```
C/C++

#define LED 13

void setup(){

pinMode(LED, OUTPUT);

}

void loop(){

digitalWrite(LED, HIGH);

delay(1000);

digitalWrite(LED, LOW);
```

```
delay(1000);

}
```

Seven Segment Display Testing:



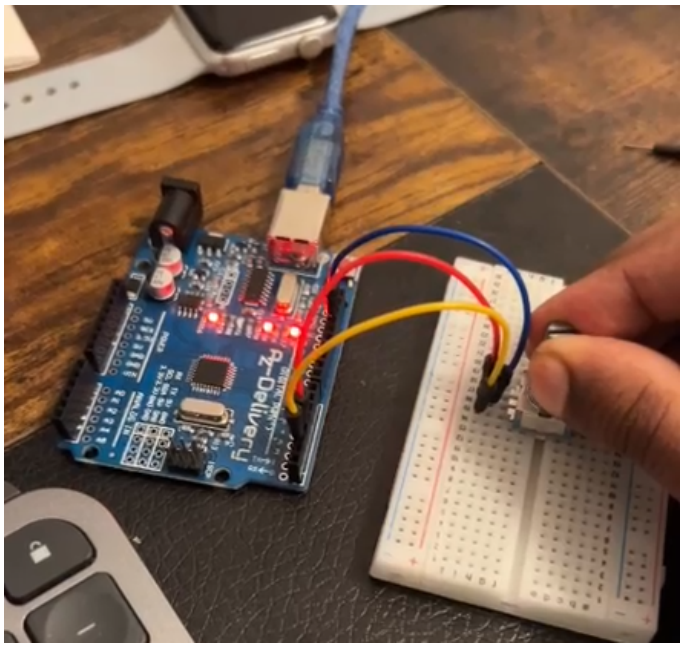
For the seven segment display, we just power each a, b, c, d, e, f, g and the decimal point with one cable to see how it works. Then after testing all seven segment displays, we ran a simple code with HIGH and LOW to display a number of the seven segment displays.

Pin Configuration :

- Segment A
- Segment B
- Decimal Point
- Segment C
- Segment D
- Segment E
- Segment F
- Segment G
- Com (Connected to GND or VCC depends on type of 7 segment display)

No code was used to test the seven segment display, it was simply connected to 5V common and GND.

Rotary Encoder Testing:



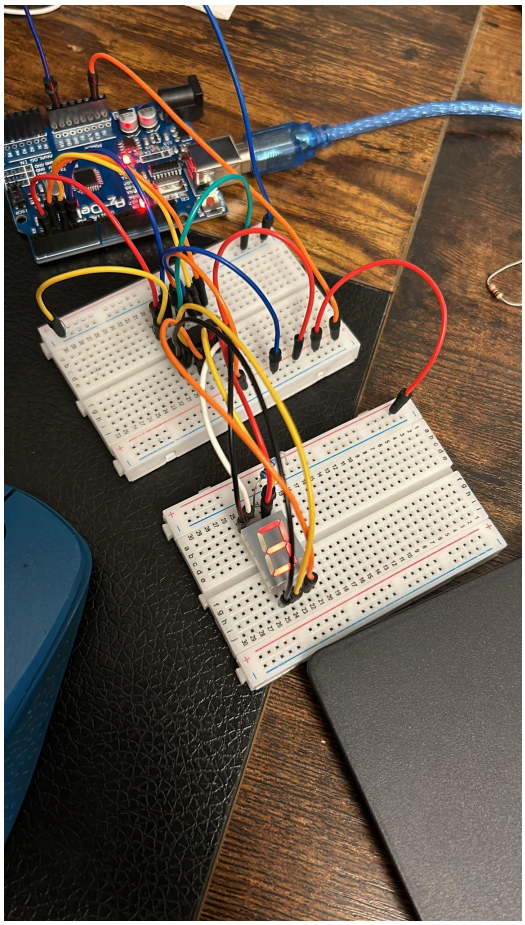
For the rotary encoder, while turning the rotary, it should return a 1. Some of the problems we faced while using the component was that it would go back and forth with the numbers if you just turned normally. We fixed it by adding an interrupt.

Pin Configuration:

- Encoder Pin A
- Encoder Pin B
- NO Push button Switch
- 5V supply
- Ground

Code is the same as software part of Chapter 3 under Rotary encoder turning.

BCD Testing:



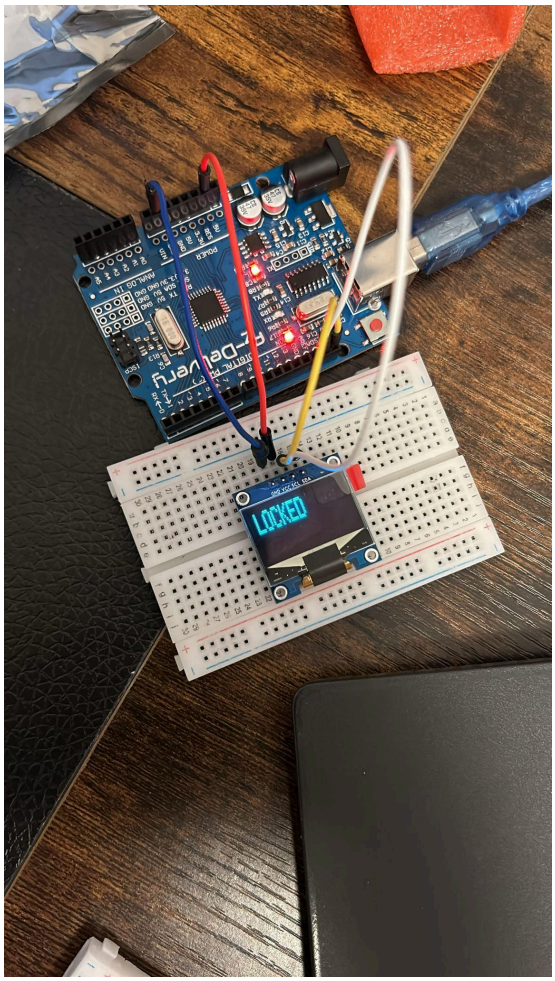
Pin Configuration:

- Pin 1: BCD input B
- Pin 2: BCD input C
- Pin 3: Lamp Test (LT)
- Pin 4: Blanking Input (BI)
- Pin 5: Ripple Blanking Input (RBI)
- Pin 6: BCD input D
- Pin 7: BCD input A
- Pin 8: Ground
- Pin 9: Segment E
- Pin 10: Segment D
- Pin 11: Segment C

- Pin 12: segment B
- Pin 13: Segment A
- Pin 14: Segment G
- Pin 15: Segment F
- Pin 16: VCC (5V)

Code is the same as the BCD function (old) mentioned in Chapter 3 under Binary Coded Decimal Function in software.

OLED Screen Testing:



The OLED screen allows us to print out the countdown if the user has their passcode inserted

wrong 3 times. We clear the screen first, and then have it print out the countdown from 100 seconds when the passcode is wrongly inserted 3 times. When the timer hits 0, the screen erases itself, and the countdown will increase by 100 seconds every time the user got the wrong passcode 3 times.

Code for OLED testing

```
C/C++

#include < Wire.h>

#include <Adafruit_GFX.h>

#include < Adafruit_SSD1306.h>


#define OLED_RESET (-1)

#define SCREEN_WIDTH 128

#define SCREEN_HEIGHT 64

#define SCREEN_ADDRESS 0x3C


Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);


void setup(){

display.begin(SSD1306_PAGEADDR, SCREEN_ADDRESS);

display.clearDisplay();

}


void loop(){
```

```
display.setTextSize(4);

display.setTextColor(WHITE);

display.setCursor(0,0);


display.println("LOCKED");

display.display();


}
```

Calculation:

- We use 1k for all 7 segment Displays because $R = V / I$

  $R = \frac{5v-2v}{0.003A}$

  $=> R = 1000\ \Omega.$

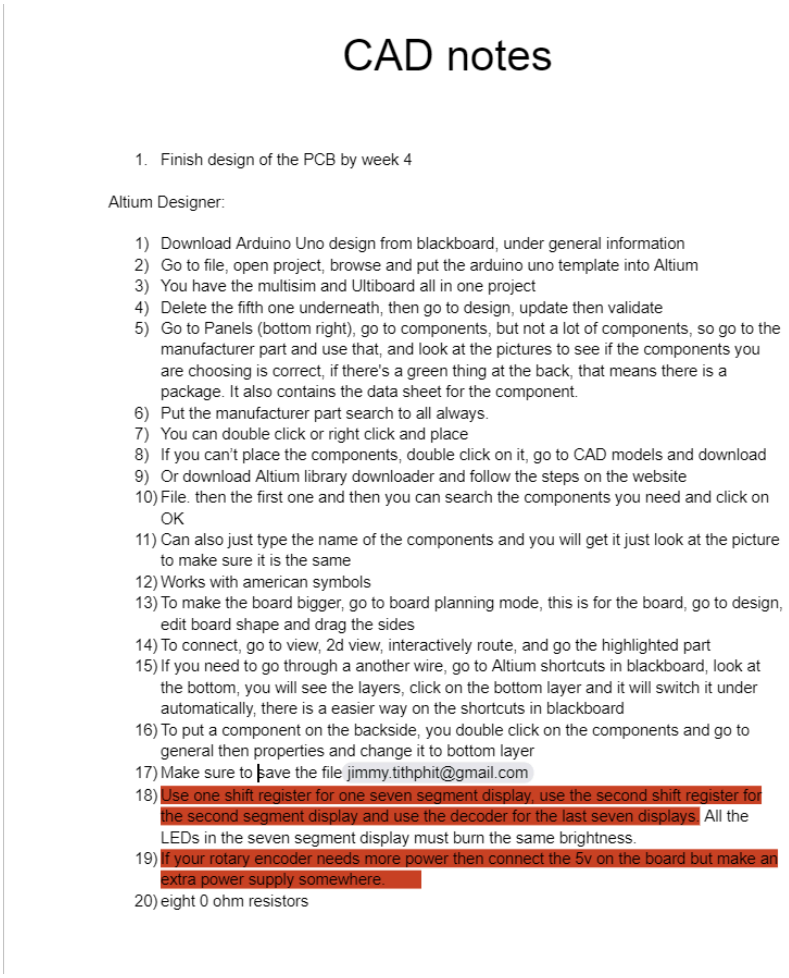
- We use $10k\ Ohms$ and $10k\ \mu F$ for the filter use with the Rotary Encoder (Datasheet)
- We use 220 for the LeDs because $R = V / I$

  $R = \frac{5v}{0.02A}$

  $=> R = 250\ \Omega.$

Altium:

We first started Altium by attending a 2 hour course to learn the basics for the application. This was during week 3, while attending the class, we took down notes about the tools and shortcuts for Altium.



CAD notes

1. Finish design of the PCB by week 4

Altium Designer:

1) Download Arduino Uno design from blackboard, under general information
2) Go to file, open project, browse and put the arduino uno template into Altium
3) You have the multisim and Ultiboard all in one project
4) Delete the fifth one underneath, then go to design, update then validate
5) Go to Panels (bottom right), go to components, but not a lot of components, so go to the manufacturer part and use that, and look at the pictures to see if the components you are choosing is correct, if there's a green thing at the back, that means there is a package. It also contains the data sheet for the component.
6) Put the manufacturer part search to all always.
7) You can double click or right click and place
8) If you can't place the components, double click on it, go to CAD models and download
9) Or download Altium library downloader and follow the steps on the website
10) File. then the first one and then you can search the components you need and click on OK
11) Can also just type the name of the components and you will get it just look at the picture to make sure it is the same
12) Works with american symbols
13) To make the board bigger, go to board planning mode, this is for the board, go to design, edit board shape and drag the sides
14) To connect, go to view, 2d view, interactively route, and go the highlighted part
15) If you need to go through a another wire, go to Altium shortcuts in blackboard, look at the bottom, you will see the layers, click on the bottom layer and it will switch it under automatically, there is a easier way on the shortcuts in blackboard
16) To put a component on the backside, you double click on the components and go to general then properties and change it to bottom layer
17) Make sure to save the file jimmy.tithphit@gmail.com
18) Use one shift register for one seven segment display, use the second shift register for the second segment display and use the decoder for the last seven displays. All the LEDs in the seven segment display must burn the same brightness.
19) If your rotary encoder needs more power then connect the 5v on the board but make an extra power supply somewhere.
20) eight 0 ohm resistors

Screenshot of the notes we had for Altium, including changes and things we were going to add to the Altium design.

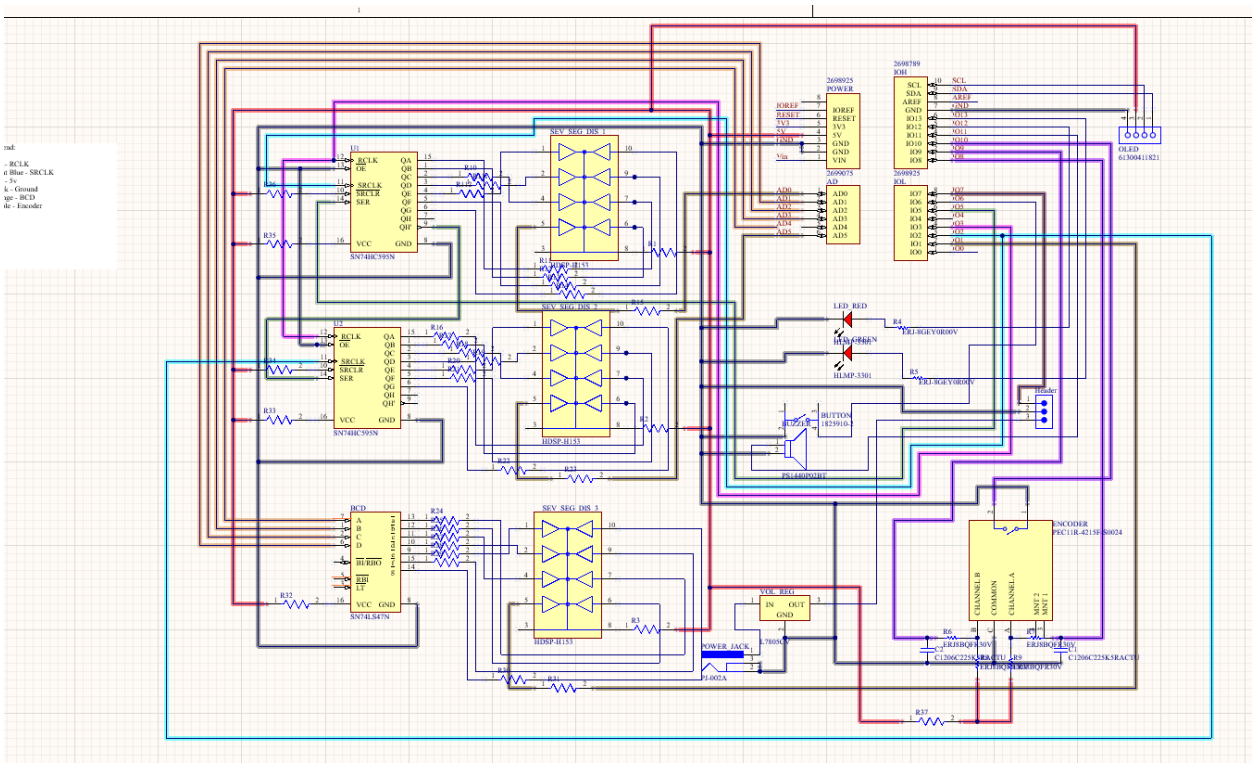
After attending the Altium course, we had a meeting about the components that we will be using and the ones that we will be adding into as special features.

Components:

- 74hc595 - x2
- Sa52 - x3 - Seven segments displays
- Rotary encoder - x1
- 1206 resistor
- Led 5mm - x2
- 7805 - to use to make a extra power supply because your PCB might not be powered with the normal voltage
- Buzzer

We made a list of the components that we were going to use and some little notes for some components.

The design below, is the final design of the of the Altium, as you can see we have two shift registers, one BCD, a rotary encoder, 3 seven segment displays, a buzzer, two LEDs (red and green), some header pins, both female and male for the OLED screen and the servo. And a lot of resistors

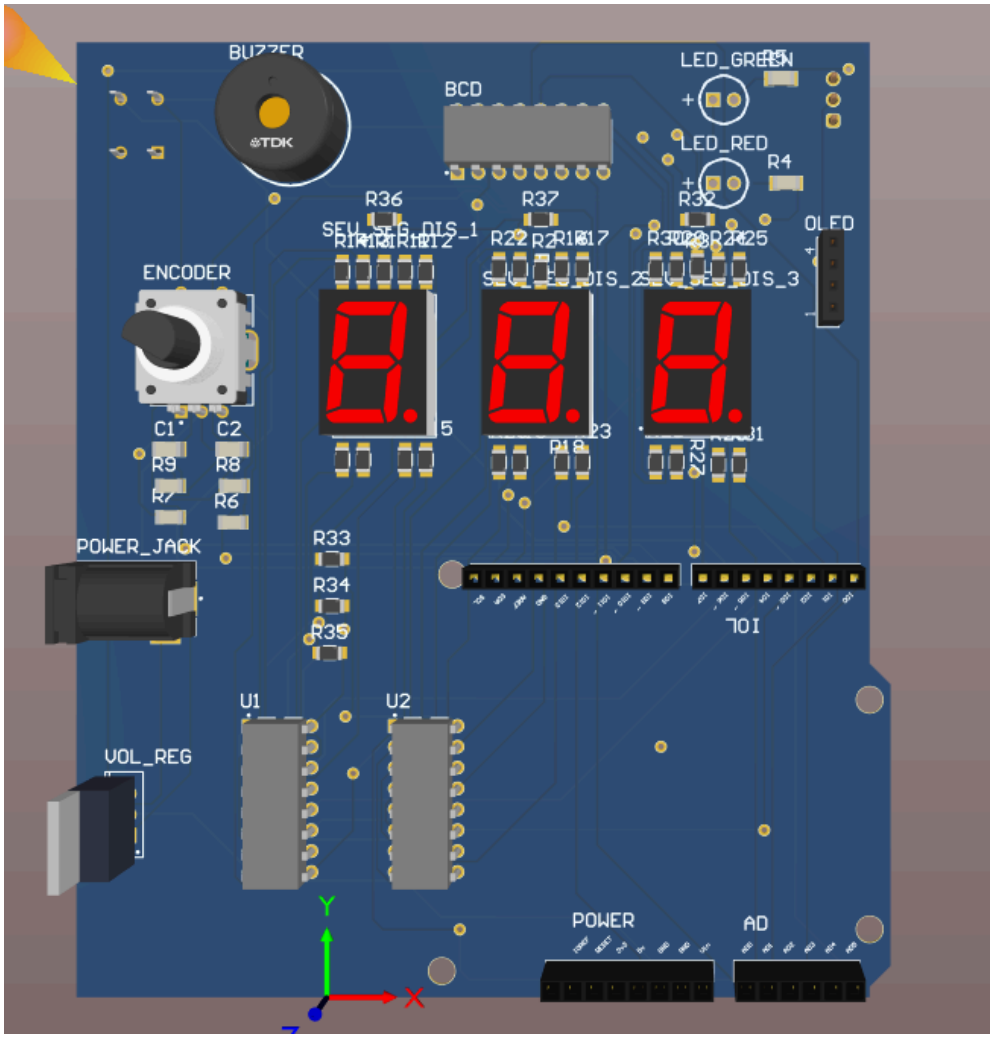


We color coded the lines to not get confused, but there was an easier way where you connect them wirelessly.

The first step was to research all the correct components and download an extra function, a library, to get more components if Altium didn't have it on the components panel. After getting most components, we referred to the Datasheet for every component to figure out how to connect, for example, the seven segment displays to the shift register or how to connect the shift register together.

Then we started connecting everything together, we also added at first 1k ohm resistors for everything but then we changed everything to 0 ohm resistors so we can add any resistors with any value.

For the 3D design on the PCB, we first decided where each component would go:

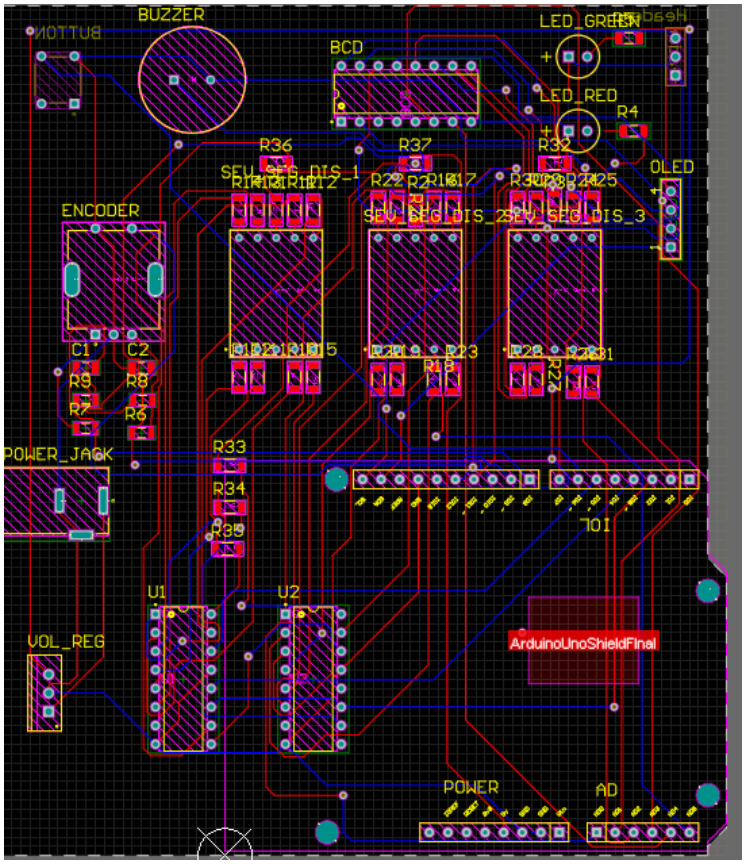


The first step of creating the 3D PCB, is to first put the components down to the correct place, after holding a meeting and discussing where we were going to put the components, we finally started placing them.

After deciding the position of each component, tracing the copper lines was the next step. Midway through the plotting of the copper lines, I had too many vias on my board, I started researching on how to do the lines properly and I discovered the autoroute. Auto route basically routes all components together automatically, all you need to do, is first make sure that all components are placed correctly and after it is done routing, check if the auto route did all the routing correctly, because sometimes it doesn't add a via hole for some connection when it switches between copper top to copper bottom.

After all of the steps were done, we had to first check if there were any errors on the PCB, there were a few (via holes missing) and some cables weren't connected properly. Then we made our gerber file that would be sent out to the factory to make our PCB.

Here is a 2D view of our PCB:



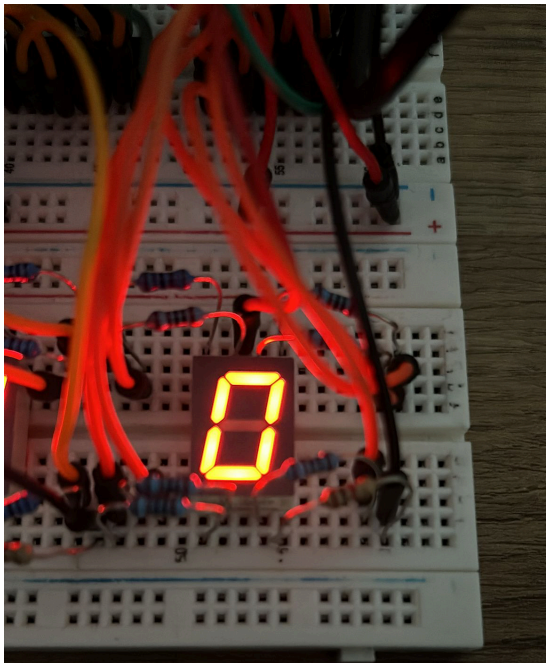
General:

Unit Testing Procedure (Problem Solving):

Seven Segment display:

For the seven segment display at first we used a 220 Ohm resistor on the common anode which made the brightness of each display not the same and some segments were too low to even see.

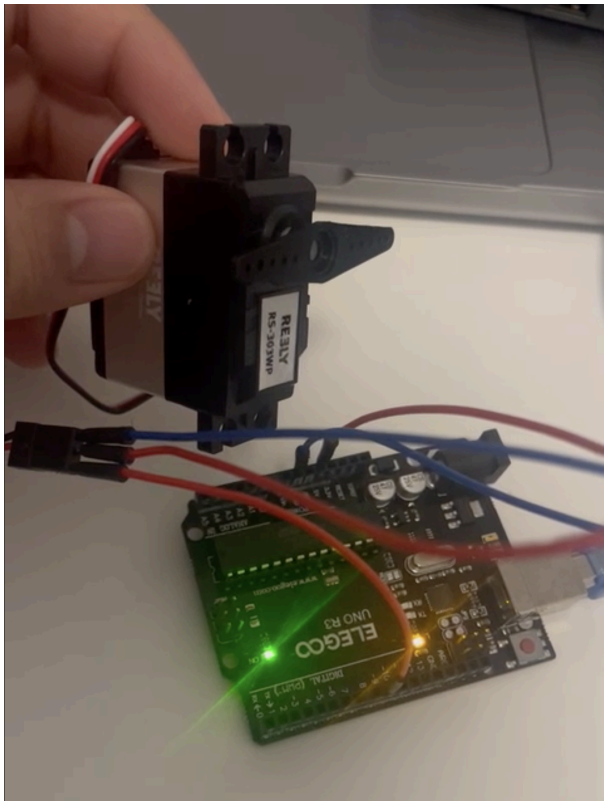
What we did to solve that problem was to change the value of the resistor to 1K Ohm and add it to each segment of the display. This ensures that each segment is getting the same current. As a result all the 7 segment display brightness are all bright and equal.



Servo:

For the servo, at first we were using the power through the 5V Pin. During the testing we realized that the 5V we're using to power the servo and also other components are not enough to run the servo smoothly due to the servo needing 5V for itself.

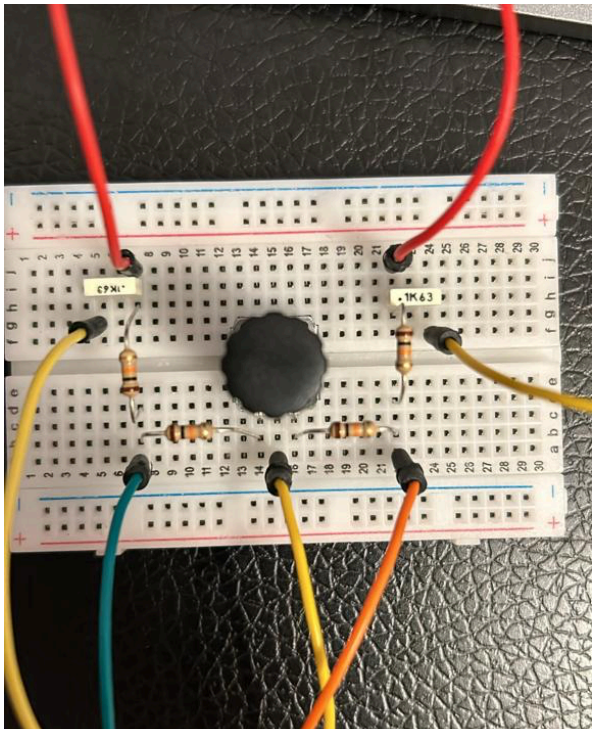
So to solve that problem we used Vin which inputs 12V from the source but we also use a 5V regulator to cap the voltage transmitting to the servo at 5V so it won't get too much voltage than it can handle. As a result the servo has enough voltage it needs to run smoothly.



Rotary Encoder:

For the Rotary Encoder, we connect it normally but during the testing we notice that the rotary outputs too much noise. The rotary encoder also gives no bounce when turned backwards.

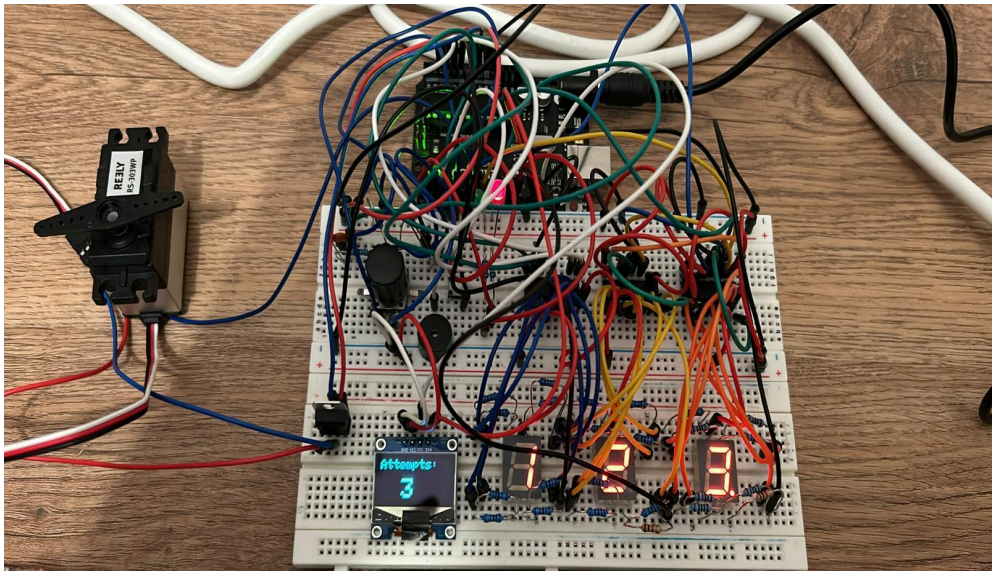
The solution for this problem is to use a filter to reduce the noise from the rotary. As a result the rotary is working smoothly.



Chapter 4: Implementation and Testing

Software:

We did our implementation testing by slowly adding more components to the breadboard. We started with the 7 segment display and rotary encoder to ensure that the turning rotary encoder function was working perfectly. Following up we added and tested one by one the LED, buzzer, oled and servo. However while testing the whole system we encountered the problem where the whole safe would freeze. To bugfix the problem we added a temporary variable that would display in the arduino serial monitor every time it loops. We were able to identify that while the whole system was frozen the program was still looping until we input a password. However that didn't give the solution to the problem. To do further testing we displayed a number for every single function. Running the whole system again we were able to see that the program was consistently freezing during our oled functions. After checking that our oled function was working as intended we checked the physical oled and arduino finding out that the 5 volt regulator was overheating causing the oled function to output values that would freeze the system.



Relation to High Level Design

In our high level design, there were three major testing procedures; input processing and output.

Input

- **Password Input(opening safe & password change)**

  We were able to achieve the password input from our high level design with two functions for the rotary encoder. The first function is used to check if the rotary encoder was turned and which direction it is turned. The second function is used to change which digit to change in the password and to input the password.

- **Closing input**

  To achieve the safe closing input from the high level design we wrote a function that checks if the button used to close the safe was pressed.

Processing

- **Attempts & lockdown**

  While designing for the password input to open the safe we also added to the design an anti theft function which gave attempts to the user and locked down if too many attempts are wrong. To achieve this we used the Password input and used a function to check if it was right or wrong. In case of the latter the safe would remove the attempts and store it in the EEPROM so losing power won't reset the attempts. After the attempts reach 0 we start a lockdown timer which increments by 30 seconds each time it goes into lockdown mode. The timer is also stored in the EEPROM so losing power doesn't reset it.

- **Looping back**

  In our high level design we decided that a safe should keep working normally without having to reset it. This is done by resetting all variables that tell that the safe is open and reset the attempts and lockdown timer.

Output

- **Visual cues**

  We had more visual cues added to the safe than intended from our initial planning of the high level design. While having the visual cues to attempts, closing and opening the safe we also added a visual display using the oled to show the lockdown timer.

- **Sound cues**

  Like the visual cues we had added more sound cues than we intended with our high level design. We had Sound cues to both wrong attempts and opening the safe but we also added a Sound cue whenever the password change got enacted.

Hardware:

To fix the components on the PCBA, there are 2 types of components:

- **Surface Mounted Devices**

  Some components would be resistors or capacitors, these components will be pasted onto the board using a special paste and then baked in an oven.

- **Through Holes components**

  For the through components, we have to solder it manually with a solder machine. Components such as the seven segment display or the button or even the rotary encoder have to be soldered with the solder machine.

Then we would align the pins underneath the PCBA with the holes on the Arduino to connect them together. After connecting it, we need to then connect the power jack to the PCBA to power the servo.

One of the problems we encountered is the copper wires that are connected to the Analogue pins, the A4 and A5. These pins are the same as the OLED screen, so we wouldn't be able to make the PCBA work.

Test Results:

Unit Testing*

All units are tested with their required software counterpart. Requirements for input and output for each component are met.

Seven Segment Display – All segments light up equally bright now since the problem of using a single resistor for all segments is not there. Each segment now gets a separate 1k Ohm resistor.

Rotary Encoder – A hardware filter using four 10k Ohm resistors and two 0.1uF capacitors is used to filter the signal. There is no more bouncing. The button also works as well.

Servo - The servo gets its separate 5V from the Vin port on the Arduino via a 12V DC source limited through a 5V regulator LM7805. This way there is enough voltage to power the Servo and also not too much such that the servo burns out.

Buzzer - Works as intended and output is given when it is required. A faint tone is heard when the Arduino is plugged in but that can be attributed simply to the buzzer receiving some voltage from the Arduino.

OLED - Displays the required output of remaining attempts and also displays countdown timer as intended.

Shift Register - Since 2 shift registers are used, they are daisy chained so that only 3 pins from the arduino are used to control two seven segment displays in Integration testing. However, in unit testing one shift register was used to control output of one seven segment display.

BCD - 4 pins on the arduino feed in data to the BCD which then displays the number on the seven segment display.

Push Button - Is able to read press from the user, and the hardware along with the code is able to make the push button a functional component to register input.

*See Chapter 3 for all relevant pictures under unit testing documentation

---

Integration Testing

Based on the high level design of the project and the flowcharts listed in Chapter 2, our high level tests of the project began.

- In the **input stage** of our integration testing , we ensured that the three methods of input; the turning of the rotary encoder, the button press of the rotary encoder and the push button were in working order and were able to control output along with the input that was registered.

  In the software part, functions of reading encoder, button press and push button reading were in perfect working order eliminating bouncing of each component input.

- The **processing** part of integration testing was done on TinkerCAD before the hardware breadboard and the outcome of integration tests like daisy chaining shift registers were a success. Other integration tests like working of the Servo, LED and buzzer in tandem were also a success.

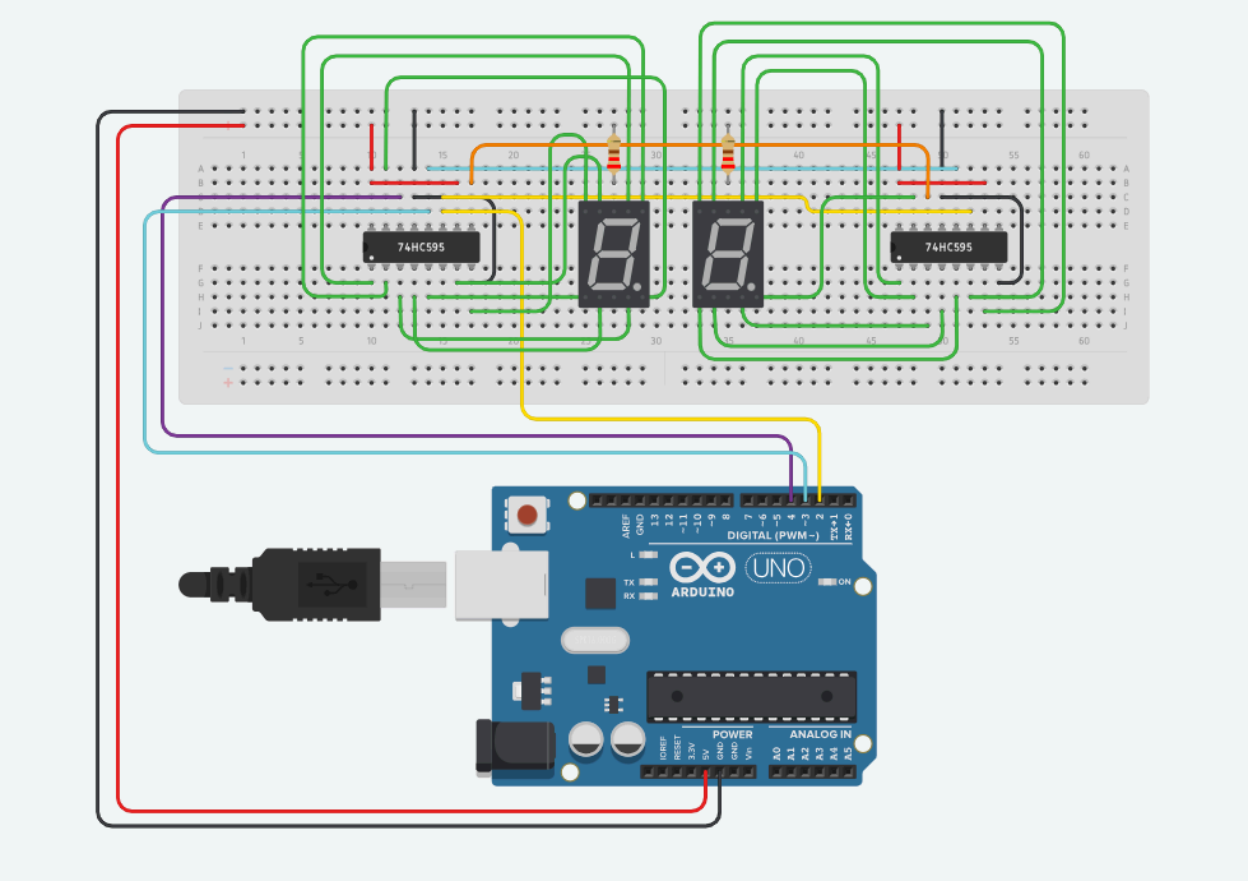
  In the software part of processing the data, the functions could successfully create 3 attempts and a pattern of locking itself for 30 seconds increment for every 3 wrong

attempts made. The code was working perfectly. EEPROM storage also ensured to begin countdown again once power was reconnected after outage.

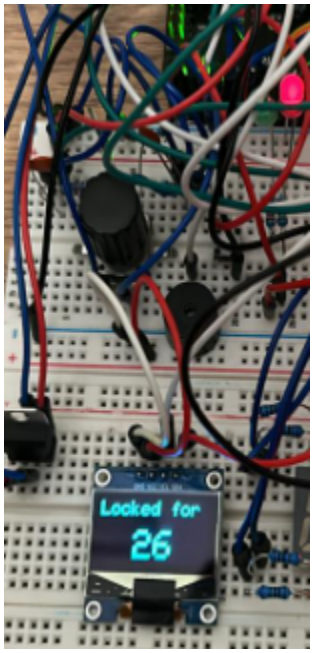
- The **output** part of integration testing such as visual cues from LED both red and green, different tones from the buzzer on right and wrong input, OLED display displaying different states of the safe and the required outputs from the shift registers and BCD connected with the seven segment displays.

  The software ensured smooth output control, with servo turning, LED green turning on and red turning off, buzzer playing a tone that the safe could be controlled in two states of locked and unlocked as stated in the high level design.

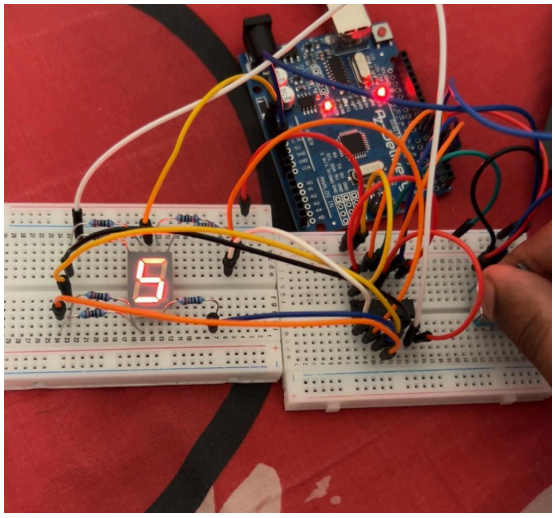
Some integration tests are given below



PROCESSING - When two shift registers are daisy chained and how to control each one of them.



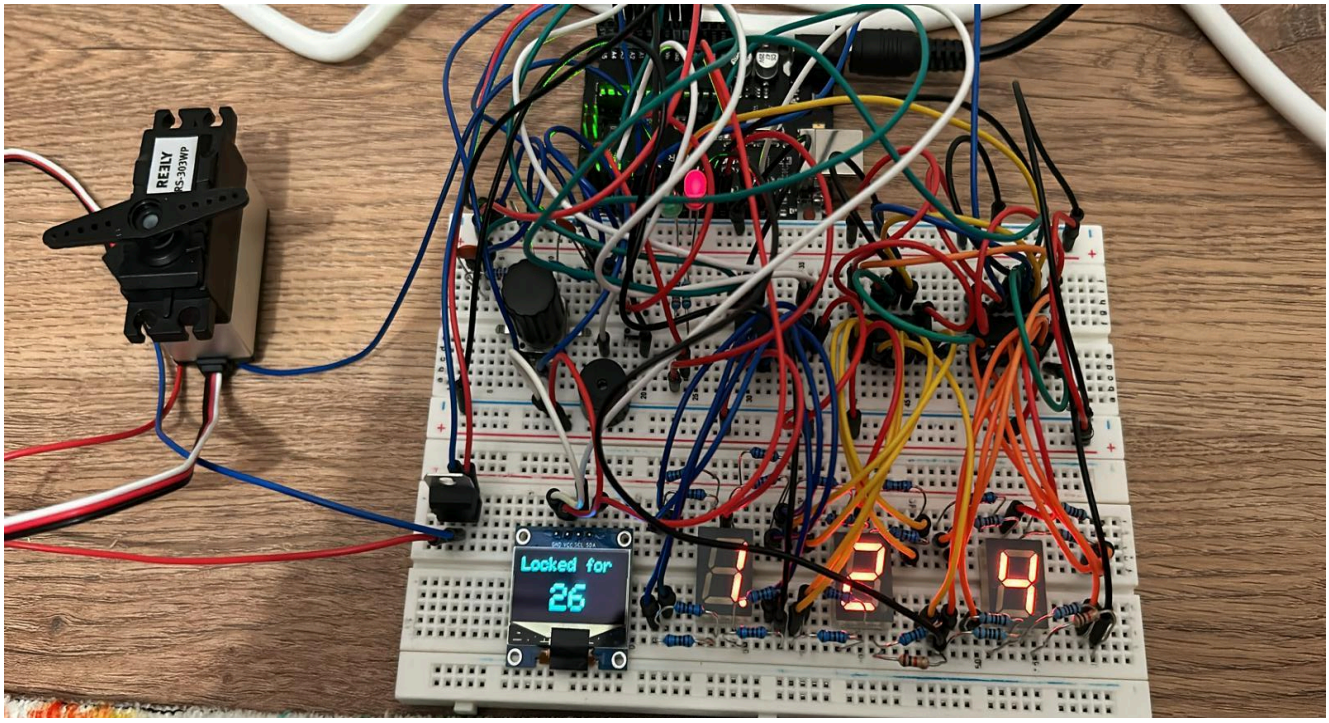
PROCESSING - The process of being in lockdown after 3 attempts.



INPUT & OUTPUT - The input given by the rotary encoder makes the output of the shift register and hence the seven segment display change.

System Testing

The results were a perfectly working safe with the features working as intended. The rotary encoder was working perfectly as every turn the 7 display number would change. The button on the rotary would change which 7 segment display it was changing. The LED was changing properly depending if the safe was open or not. The buzzer was making sounds whenever the safe was opened, the password was incorrect and when the user inputs a new password. The oled also worked perfectly as it would change displaying the right number of attempts and lockdown timer.



Bibliography:

Ideas, Mario's. "How to Use 74HC595 Shift Registers to Control Mulitple 7 Segment Displays." *YouTube*, YouTube, 3 Aug. 2020, www.youtube.com/watch?v=QI1IJLB42G8&t=517s.

Academy, Programming Electronics. "3 Ways to Power an Arduino Board - Do You Know Them?" *YouTube*, YouTube, 10 Sept. 2021, www.youtube.com/watch?v=c03UuefFB3A&t=102s.

Buddies, Science. "How to Power an Arduino Project (Lesson #19)." *YouTube*, YouTube, 16 Aug. 2023, www.youtube.com/watch?v=I7MrL5Q7zvY&t=533s.

Avvan, Hamza. "BCD to 7 Segment Display Arduino." *YouTube*, YouTube, 18 Apr. 2020, www.youtube.com/watch?v=gCQC1Nhs0HA.

ubcengphysprojectlab. "Arduino 04: Attaching and Testing a Servo Motor." *YouTube*, YouTube, 21 Dec. 2015, www.youtube.com/watch?v=a96WIpDho64&t=2s.

Ed. "Arduino Store Array into EEPROM - the Robotics Back." *End*, 28 Sept. 2021, roboticsbackend.com/arduino-store-array-into-eeprom/.

Stoffregen, Paul. "Encoder Library." *PJRC*, www.pjrc.com/teensy/td_libs_Encoder.html. Accessed 20 Jan. 2024.

"7 Segment Display." *Components101*, components101.com/displays/7-segment-display-pinout-working-datasheet. Accessed 21 Jan. 2024.

Instruments, Texas. *BCD-to-Seven-Segment Decoders/Drivers Datasheet - Texas Instruments India*, www.ti.com/lit/ds/symlink/sn5447a.pdf. Accessed 21 Jan. 2024.

Instruments, Texas. *SNX4HC595 8-Bit Shift Registers with 3-State Output Registers ... - Analog*, www.ti.com/lit/ds/symlink/sn54hc595-sp.pdf. Accessed 21 Jan. 2024.

apine, alps. “Rotary Encoder Datasheet.” *Products/Technology*, tech.alpsalpine.com/e/products/detail/EC1110120201/#ancFig5. Accessed 21 Jan. 2024.

“Arduino OLED I2C Tutorial : 0.96" 128 x 32 for Beginners.” *YouTube*, YouTube, 14 Oct. 2018, www.youtube.com/watch?v=_KD7skmusTQ.

King Bright USA. *SA52-11ewa 13.2 Mm (0.52 Inch) Single Digit Numeric Display*, www.kingbrightusa.com/images/catalog/spec/SA52-11EWA.pdf. Accessed 21 Jan. 2024.

Appendix

```C/C++
#include <Arduino.h>

#include <Servo.h>

#include <EEPROM.h>

#include <Wire.h>

#include <Adafruit_GFX.h>

#include <Adafruit_SSD1306.h>

#include <EEPROM.h>

#define ENCODER_OPTIMIZE_INTERRUPTS

#include <Encoder.h>


// Pins

#define RED_LED 1  // OP

#define GREEN_LED 13 // OP

#define BUZZER 3   // OP

#define SERVO 8  // ATTACH

#define ROT_A 5   //INP

#define ROT_B 10   //INP

#define ROT_BTN 7  //INP_PL

#define SR_DATA 6 // OP

#define SR_LAT 2   // OP

#define SR_CLK 9  // OP
```

```
#define BCD_A 4  // OP
#define BCD_B 17  // OP
#define BCD_C 16  // OP
#define BCD_D 15  // OP
#define BUTTON 0 // INP_PL
#define DP1 14   // OP
#define DP2 11   // OP
#define DP3 12  // OP
// All other consts
#define DEBOUNCE_TIME 60
#define LOCKED 100
#define UNLOCKED 101
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64

#define OLED_RESET (-1)
#define SCREEN_ADDRESS 0x3C

const int DEC_DIGITS[10] = {1, 79, 18, 6, 76, 36, 96, 15, 0, 12}; // 32 4
const int BCD_PINS[4] = {BCD_A, BCD_B, BCD_C, BCD_D};

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
Servo s1;
```

```
Encoder Rotary(ROT_A,ROT_B);


//High level flowchart: Visual clue
/// Controls state of LEDs that display what state the safe is in
/// Sets value of each LED (high/low) depending on what state is
/// @param state is the state of safe (UNLOCKED/LOCKED)
void led_control(int state){
  if (state == UNLOCKED) {
    digitalWrite(RED_LED, LOW);
    digitalWrite(GREEN_LED, HIGH);
  } else {
    digitalWrite(RED_LED, HIGH);
    digitalWrite(GREEN_LED, LOW);}
}


void writeArrayIntoEEPROM(int address, byte num1, byte num2, byte num3)
{
 int numbers[3] = {num1, num2, num3};
 int addressIndex = address; //Address will be updated in each loop
 for (int i = 0; i < 3; i++)  //A loop runs 3 times to store 3 seperate values of the
password
 {
  EEPROM.write(addressIndex, numbers[i]); //Store the value into 1 address
```

```
  addressIndex += 1; //Shift to another address to store another value
 }
}

//High level flowchart: Sound
/// Controls buzzer output based on state (LOCKED/UNLOCKED)
/// Plays two short notes of same freq when state = LOCKED
/// Plays two notes in ascending order when state = UNLOCKED
/// @param state is state of safe
void buzzer_control(int state){
  if (state == UNLOCKED) {
    tone(BUZZER, 500);
    delay(100);
    tone(BUZZER, 600);
    delay(100);
    noTone(BUZZER);
  } else {
    tone(BUZZER, 440);
    delay(100);
    noTone(BUZZER);
    delay(5);
    tone(BUZZER, 440);
    delay(100);
```

```
    noTone(BUZZER);
  }
}

/// Reads rotary encoder adjusted for debouncing
/// @returns 0 if no rotation, 1 if anti-clockwise and -1 is clockwise
/// Do at 9600 baud
int read_encoder() {
 static long RotaryNum;
 static long RotaryTurn;
 RotaryTurn = Rotary.read();
 if (RotaryTurn != RotaryNum) {
  //Serial.print(RotaryTurn);
  //Serial.print(RotaryNum);
  if ((RotaryTurn % 4) == 0) {
   if (RotaryNum < RotaryTurn){
      RotaryNum = RotaryTurn;
    return 1;}
   else{
     RotaryNum = RotaryTurn;
     return 2;}
  }
 }
```

```
 return 0;
}
//High level flowchart: Safe closing/opening
/// Controls the servo motor
/// Moves to 180 degree when UNLOCKED and 10 when LOCKED
/// @param state is state of safe
void servo_control(int state){
  if (state == UNLOCKED){
    s1.write(90);
  } else {
    s1.write(10);
  }
}


/// @returns for how long the rotary encoder button was pressed
/// Compares by the difference between two clocks starting at button press and
button release
int rotary_button(){
  static unsigned long time, time2;
  long elapsed_time = 0;
  delay(5);
  if (digitalRead(ROT_BTN) == LOW) {
   time = millis();
```

```
   while (!digitalRead(ROT_BTN)) {
   }
   time2 = millis();
   elapsed_time = time2 - time;
  }
  if (elapsed_time > 1500) {
   return 1; // long
  } else if (elapsed_time > 0) {
   return -1; // short
  } else {
   return 0;
  }
}
//High level flowchart: User closing safe
/// Checks whether safe is open or closed based on reading button state
/// @returns true (1) if safe is CLOSED and false (0) if OPEN
bool open_close_button(){
 const unsigned long time = millis();
 static int oldState = 1;
 static unsigned long oldTime = 0;
 const int newState = digitalRead(BUTTON);

 if (newState != oldState && (time - oldTime) > DEBOUNCE_TIME){
```

```
  oldTime = time;
  oldState = newState;

  if (newState == LOW) return true;
 }

 return false;
}
//High level flowchart: password check
/// Checks if passcode entered is same as correct passcode
/// For loop iterates through two arrays containing user passcode and real passcode
/// @returns true if passcode correct else false
bool check_passcode(const int array1[3], int array2[3]){
  for (int i = 0; i < 3; ++i){
    if (array1[i] != array2[i]) return false;
  }
  return true;
}

/// Selection of seven segment display is made based on decimal point
/// Cycles between 1,2,3 based on rotary encoder button press (short)
void seven_seg_select(int index){
```

```
 if (index == 0){
  digitalWrite(DP1, LOW);
  digitalWrite(DP2, HIGH);
  digitalWrite(DP3, HIGH);
 } else if (index == 1) {
  digitalWrite(DP1, HIGH);
  digitalWrite(DP2, LOW);
  digitalWrite(DP3, HIGH);
 } else if (index == 2){
  digitalWrite(DP1, HIGH);
  digitalWrite(DP2, HIGH);
  digitalWrite(DP3, LOW);
 }
}
//High level flowchart: Lockdown
/// Displays a seconds counter on oled display
/// Latch and Latch Clear built into the function
/// @param seconds is for number of seconds remaining to be displayed on the
oled
void timer_oled_display(int seconds){
  while (seconds >= 0){
    display.clearDisplay();
```

```
    display.setTextSize(2);
    display.setTextColor(SSD1306_WHITE);
    display.setCursor(0, 0);
    display.print("Locked for");

    display.setTextSize(4);
    display.setCursor(40, 30);
    display.print(seconds);

    display.display();

    delay(1000);
    seconds--;
  }
}
//High level flowchart: attempts visuals
/// Displays attempts remaining on display
/// Latch and Latch Clear built into the function
/// @param attempts is for number of attempt remaining to be displayed on the
oled.
void attempts_oled_display(int attempts){
  display.clearDisplay();
```

```
  display.setTextSize(2);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0, 0);
  display.print("Attempts:");

  display.setTextSize(4);
  display.setCursor(40, 30);
  display.print(attempts);

  display.display();
}

/// Loop that runs through 1, 2, 4 and 8, as inputs on the BCD
/// Perform a bitwise shift to the right
/// Performs an and gate with 1, this helps us see which is high and which is low
/// @param is decimal number to display
void bcd_display_num(int number){
  for (int i = 0; i < 4; i++){
    digitalWrite(BCD_PINS[i], (number >> i) & 1);
  }
}
```

```
/// Displays the number on seven segment display using shift register
/// Using shiftOut choses number based on DEC_DIGITS array
/// @param number1 is the first number to display
/// @param number2 is the second number to display
void sr_display_num(int number1, int number2){
  digitalWrite(SR_LAT, LOW);
  shiftOut(SR_DATA, SR_CLK, LSBFIRST, DEC_DIGITS[number1]);
  shiftOut(SR_DATA, SR_CLK, LSBFIRST, DEC_DIGITS[number2]);
  digitalWrite(SR_LAT, HIGH);
}

void displayDigit(int bcdNum, int srNum1, int srNum2){
 bcd_display_num(bcdNum);
 sr_display_num(srNum1, srNum2);
}

void setup() {
  const int lock_address = 20;
  s1.attach(SERVO);
  s1.write(0);

  display.display();
```

```
delay(2000);
display.clearDisplay();


attempts_oled_display(3);


pinMode(RED_LED, OUTPUT);
pinMode(GREEN_LED, OUTPUT);
pinMode(BUZZER, OUTPUT);
pinMode(ROT_A, INPUT);
pinMode(ROT_B, INPUT);
pinMode(ROT_BTN, INPUT_PULLUP);
pinMode(DP1, OUTPUT);
pinMode(DP2, OUTPUT);
pinMode(DP3, OUTPUT);
pinMode(SR_CLK, OUTPUT);
pinMode(SR_DATA, OUTPUT);
pinMode(SR_LAT, OUTPUT);
pinMode(BCD_A, OUTPUT);
pinMode(BCD_B, OUTPUT);
pinMode(BCD_C, OUTPUT);
pinMode(BCD_D, OUTPUT);
const int ADDRESS = 5;
byte num1;
```

```
  byte num2;
  byte num3; //Passcode
   //Read the passcode in the addresses

}

void loop() {
 static int ssd = 0, stateOfSafe = LOCKED;
 static int cooldown = EEPROM.read(0);
 static int userPscd[3] = {0,0,0};
 static int userAttempts = 3;
 const int PASSCODE[3] = {EEPROM.read(5),EEPROM.read(6),EEPROM.read(7)};
 if (userAttempts != 3 && userAttempts != EEPROM.read(20)){
  userAttempts = EEPROM.read(20);
  attempts_oled_display(EEPROM.read(20));
 }
 else if (userAttempts == 3 && userAttempts != EEPROM.read(20)){
  userAttempts = EEPROM.read(20);
  attempts_oled_display(EEPROM.read(20));

 }
 led_control(stateOfSafe);
 servo_control(stateOfSafe);
```

```
 if (ssd > 2) ssd = 0;
 int rotarybut = rotary_button();
 if (rotarybut == -1){
  ssd++;
 }
 else if (rotarybut == 1 && stateOfSafe == LOCKED){
//High level flowchart:  password check
  if (check_passcode(PASSCODE, userPscd)){
   stateOfSafe = UNLOCKED;
//High level flowchart: safe open
   servo_control(stateOfSafe);
//High level flowchart: Visual clue
   led_control(stateOfSafe);
//High level flowchart: opening sound
   buzzer_control(stateOfSafe);
//High level flowchart: reset cooldown and attempts
   EEPROM.write(0, 30);
   EEPROM.write(20,3);
   attempts_oled_display(EEPROM.read(20));
  } else {
//High level flowchart: attempts
   userAttempts--;
   EEPROM.write(20, userAttempts);
```

```
   buzzer_control(stateOfSafe);
   attempts_oled_display(EEPROM.read(20));
 }}
//High level flowchart: password change
 else if (rotarybut == 1 && stateOfSafe == UNLOCKED){
  Serial.println(userPscd[0]);
  Serial.println(userPscd[1]);
  Serial.println(userPscd[2]);
  writeArrayIntoEEPROM(5, (byte) userPscd[0], userPscd[1], userPscd[2]);
  buzzer_control(stateOfSafe);
 }



 int rotaryanswer = read_encoder();
 if (rotaryanswer == 1) {
  *(userPscd + ssd)-=1;
  if (*(userPscd + ssd) < 0) *(userPscd + ssd) = 9;
 }
 else if (rotaryanswer == 2) {*(userPscd + ssd)+=1;}
  if (*(userPscd + ssd) >9) *(userPscd + ssd) =0;
//High level flowchart: safe closed
 if (open_close_button() && stateOfSafe == UNLOCKED){
  stateOfSafe = LOCKED;
```

```
 servo_control(stateOfSafe);
 led_control(stateOfSafe);
 attempts_oled_display(EEPROM.read(20));
}

displayDigit(userPscd[0], userPscd[1], userPscd[2]);
seven_seg_select(ssd);
//High level flowchart: lockdown
if (userAttempts == 0){
 timer_oled_display(cooldown);
 cooldown+=30;
 EEPROM.update(0, cooldown);
 userAttempts = 3;
 EEPROM.write(20, userAttempts);
 attempts_oled_display(EEPROM.read(20));
}
}
```

THE END