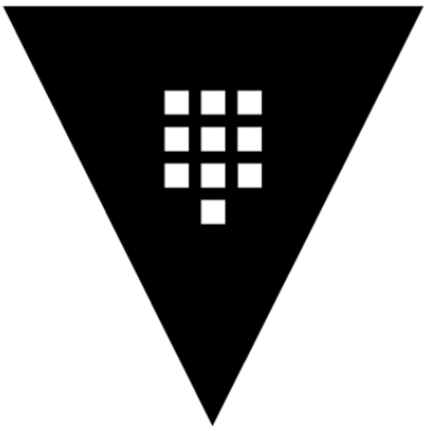
# LOCK ‘EM UP



# FINAL REPORT

# CHAPTER ONE: INTRODUCTION

This project involved the building of a vault system with a locking mechanism using Arduino. Our goal was to build a secure storage system that utilizes the Arduino microcontroller to control access through a locking mechanism. The lock system consists of the following components: the rotary encoder, button, the servo, two LEDs (red and green), buzzer, 7-segment display, shift registers (2), BCD decoder.

For this project, the system is required to be a safe for valuables keeping and so requires some level of security for the users. To achieve this result, some components were assembled in achieving this design and function. The rotary encoder sets the combination to the vault. Additionally, the rotary encoder has a button which confirms the combination after it has been selected. The component is rotated to set the combination and the button is pushed to confirm the combination. The button allows the user to reset the system and change the combination without turning off the device. The servo is another component whose function is to position the opening and closing of the vault's door. The servo rotates at an angle for both its functions. Also, the two LEDs used (green and red) provides feedback to the user. The red LEDs, obviously, indicate a wrong passcode input. The green one, on the other hand, indicates a correct combination. It is also an indication that the vault is opened. Alternatively, the red LED turns off when the vault is opened. In addition, the user requires an additional form of feedback like an auditory signal, and that is what the buzzer providers. The buzzer as an auditory signal is more attention-grabbing than the visual signal, especially in an open or noisy space. In other words, the buzzer serves as a warning signal to the user.

Also, clear visual feedbacks are also provided by the 7-segment display. As the user rotates the rotary encoder to set the combination, the 7-segment display is updated to show the current combination. This means the user can read and verify the combination before confirming it. The shift registers are in control of the segments of the 7-segment display. They send the information to be displayed on the 7-segment display. The BCD decoder is responsible for sending information to the shift registers in decimal form, which is then sent to the 7-segment display.

Furthermore, to verify the system meets the user requirements, a lot of tests were done; individual testing and overall system test (both hardware and software), and this was after we had a clear understanding of the needs and expectation of our users. To ensure the system is working as expected by our users, testing the individual components and their interactions and reactions were as important, because the overall quality of the system was the goal. Firstly, the hardware testing was done. The rotary encoder, for instance, was checked severally to confirm its functionality and observe its interaction with the 7-segment display. The door button and reset were then tested. The door button was connected to the DOORBUTTON pin on

the Arduino board. Using the uploaded code with no errors detected, the LEDs reacted accordingly to the opening and closing movements. When an incorrect combination was input and the door button pushed, the vault door does not open, and the red LED lit up, and the buzzer gave an auditory signal. The Servo was tested by observing the movement of the vault door, that is, the direction or angle when the vault is opened and closed. The buzzer was tested by observing the sound output when the door button is pressed with an incorrect passcode. When the red LED lit up and the buzzer gave a sound, we confirmed functionality. The two LEDs were tested together to know if they can determine the status of the system by their colour output. The 7-segment display was then tested by observing the display output when the rotary encoder is rotated, and the door button is pushed. With the hardware tests, all components were checked and ensure that they were properly connected to the Arduino board by checking that all the wires are securely connected to the correct pins on the board.

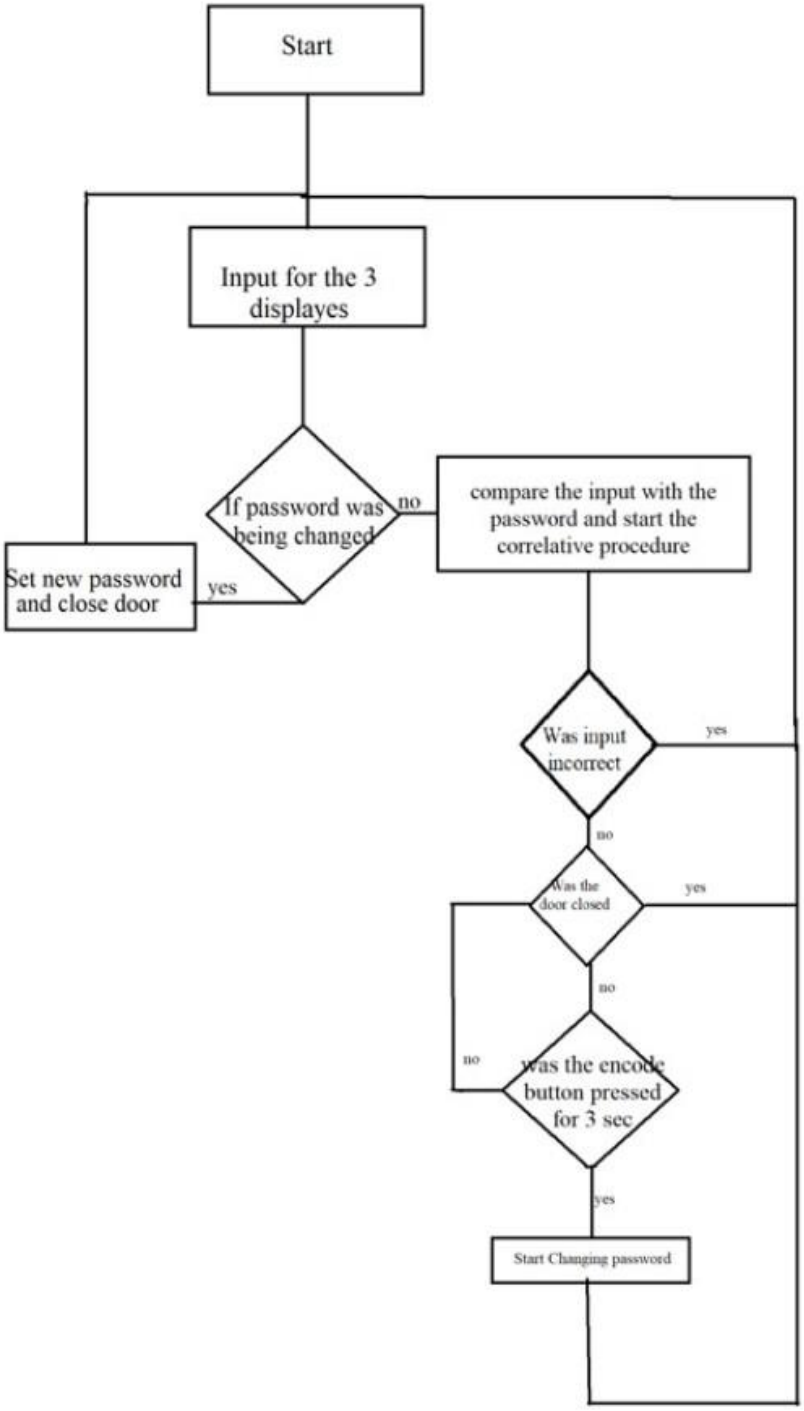
Then, the software test was also done by verifying the code loaded correctly, the code compiles without any errors, it communicates with the board correctly and functioning appropriately, that it was able to control the hardware components as intended. Debugging was done at every stage by identifying where the error is in the code by analysing error messages, reading through the code, and testing different parts of the code to identify the source of the error. At the end, it means the rotary encoder was rotated, and the 7-segment display was updated accordingly. The code can detect the direction of rotation and act accordingly. The code can detect when the buttons are pressed and generate the appropriate visual and auditory outputs. The servo motor acts accordingly by moving correctly during the opening and closing of the vault.

## Unit testing

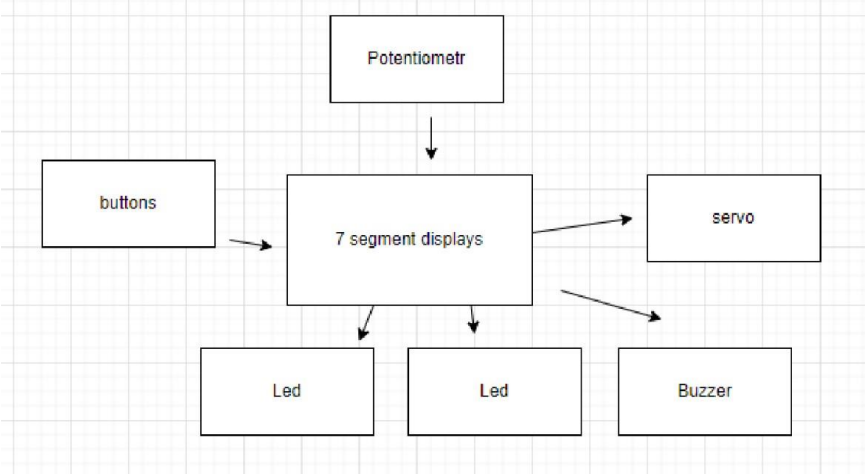
For unit testing, we started by typing every function one by one and simplified ones that were not usable alone and for functions that do not have a visible effect on their own we used series. We tested the code for the shift register, BCD decoder, buzzer, led, servo, independent buttons, clear input button, and the rotary encoder and its button.

# Chapter 2: Functional Design

## Software:



# Hardware:




A block diagram of electronic functionality is a graphical representation of the components and their interactions in an electronic system. It is a powerful tool for simplifying the understanding of complex systems and for aiding in the design, testing, and troubleshooting of electronic circuits.

The block diagram consists of blocks, which represent the various components of the system, and lines, which represent the flow of signals and power. The blocks are connected by lines that show the relationships between the components and the flow of signals.

The input and output signals are represented by arrows on the lines connecting the blocks. The processing elements, such as amplifiers, filters, and control circuits, are also represented by blocks. These blocks are connected to the input and output signals to show how they affect the signals.

In the design above, the buttons including the potentiometer are connected to the 7 segment display as inputs and the LEDs, buzzer, and the servo are the outputs.

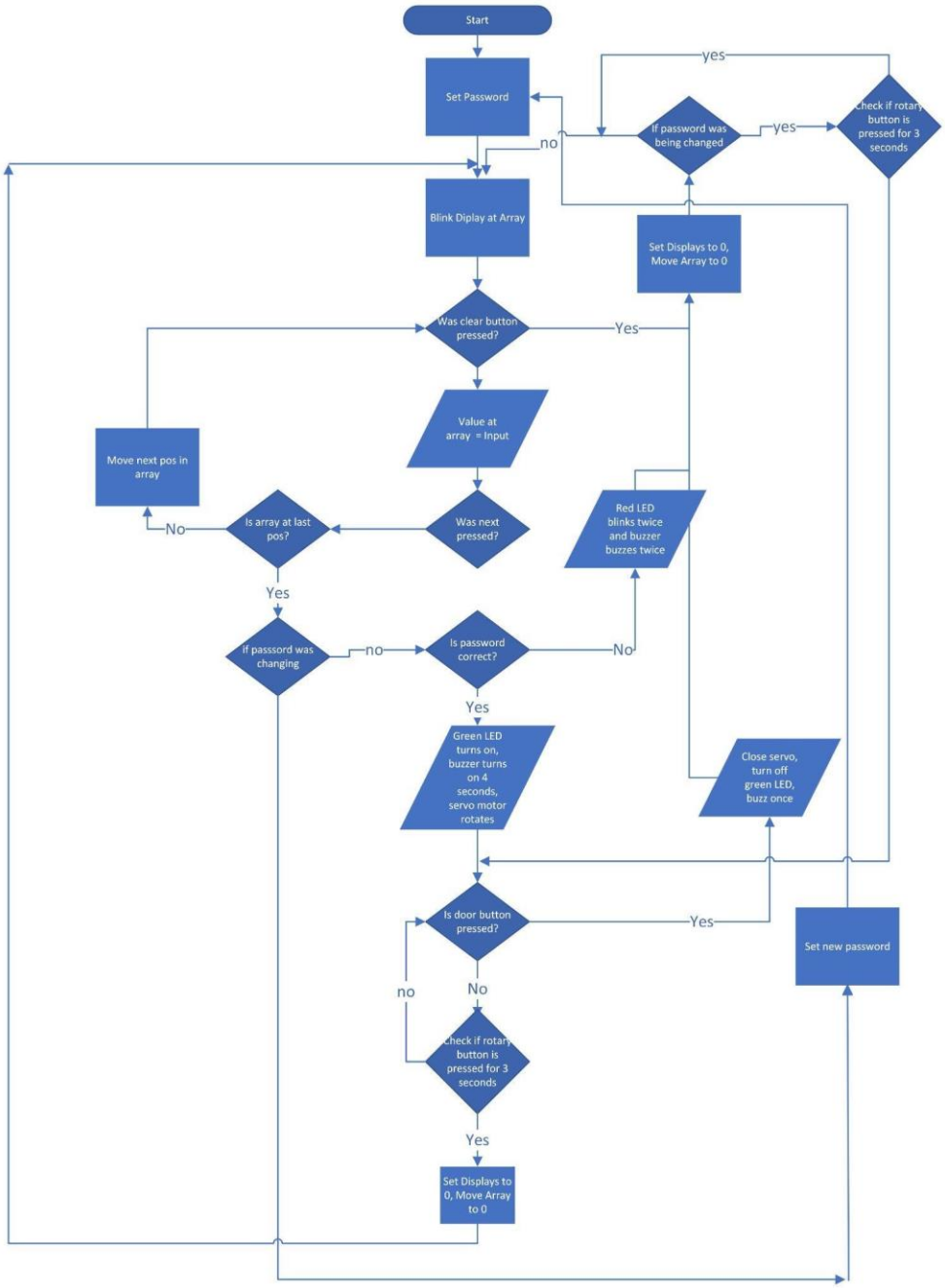
General:

Functional design

For the technical design, we grouped several functions that fall under the same group in the high-level design, testing each one. We test the grouped code for changing the password, the input for the seven-segment displays, and the codes to check the password and that we get the desired response

# Chapter 3: Technical Design

## Software:

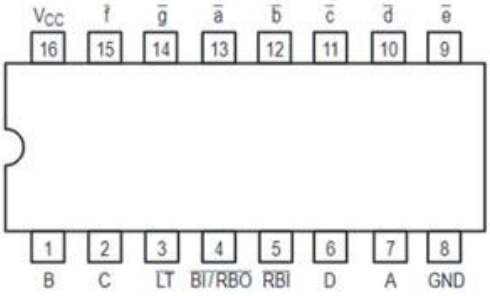


# Hardware:

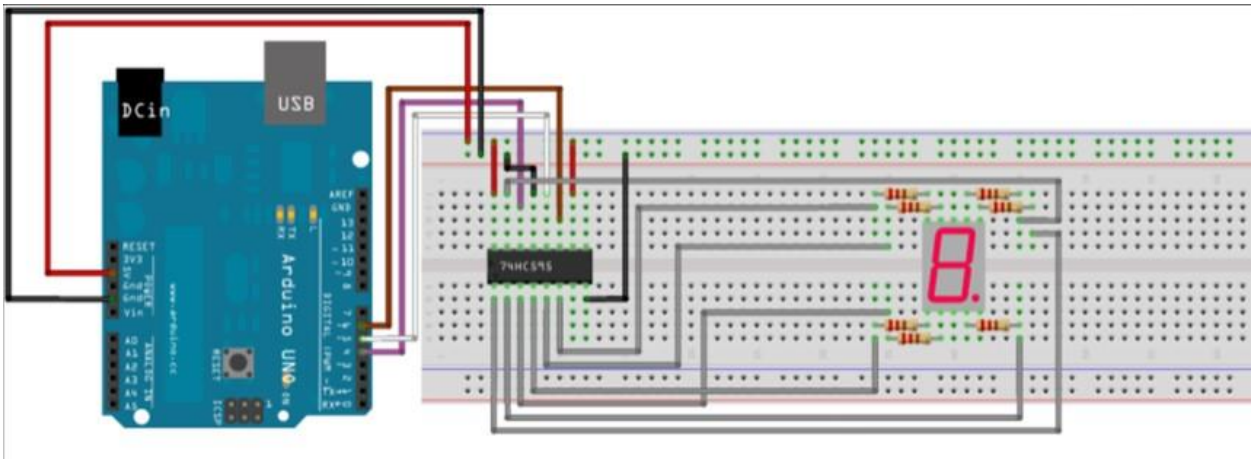
## BCD

BCD 74LS47N is a BCD-to-7-segment decoder/driver IC that can be used to display numerical data on a 7-segment LED display. It converts BCD (binary-coded decimal) input into the appropriate pattern to drive a 7-segment LED display.

CONNECTION DIAGRAM (TOP VIEW)



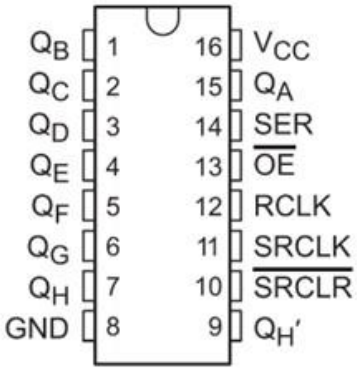

To check if it's working properly, we will connect pins A, B, C, and D of the BCD decoder to the Arduino board. Then, we will connect the output pins of the BCD decoder to the 7-segment display. After connecting everything, we will upload a code to the Arduino, which will display the decimal numbers from 0 to 9 in 4-bit binary format. The BCD decoder will then convert this binary code into the corresponding numerical form.



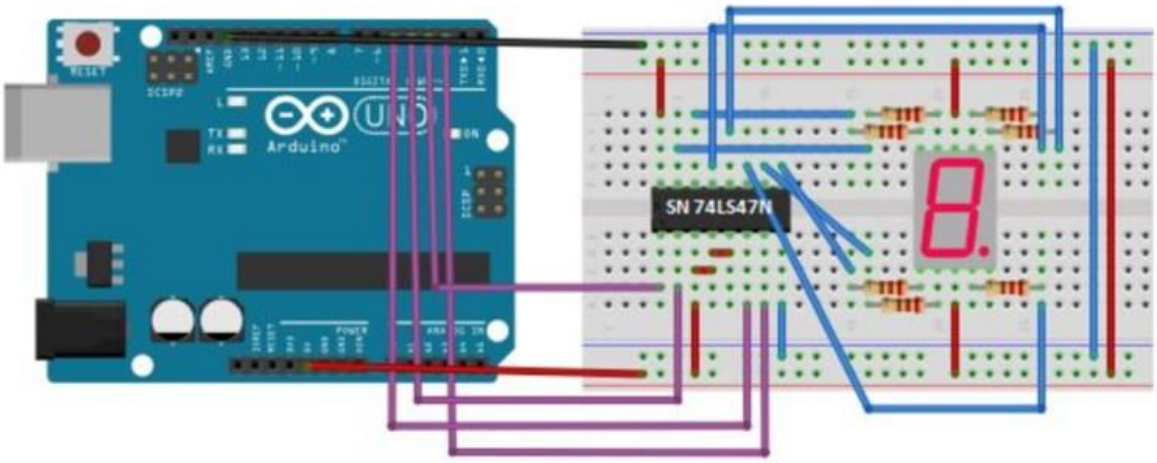

## Shift-register

The SN74HC595N is an 8-bit serial-in, parallel-out shift register. It is used to convert serial data into parallel data. It has an input for serial data (DS), a clock input (SH_CP), and a latch input (ST_CP). When the clock input is pulsed, the data at the DS input is shifted into the register. The parallel outputs (Q0-Q7) can then be used to drive other devices.

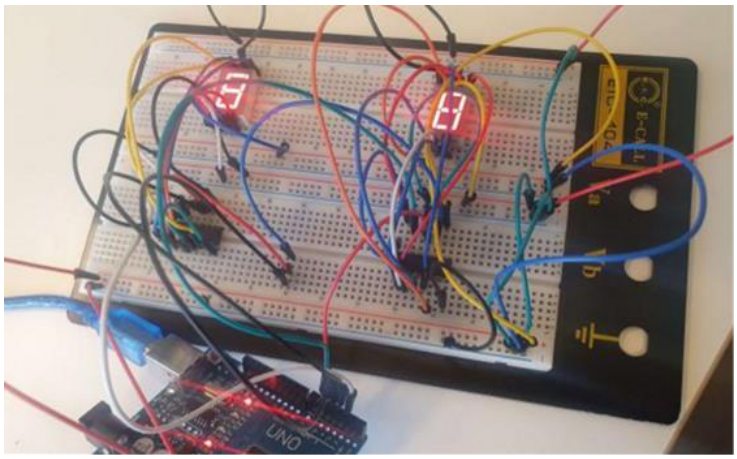
CONNECTION DIAGRAM (TOP VIEW)



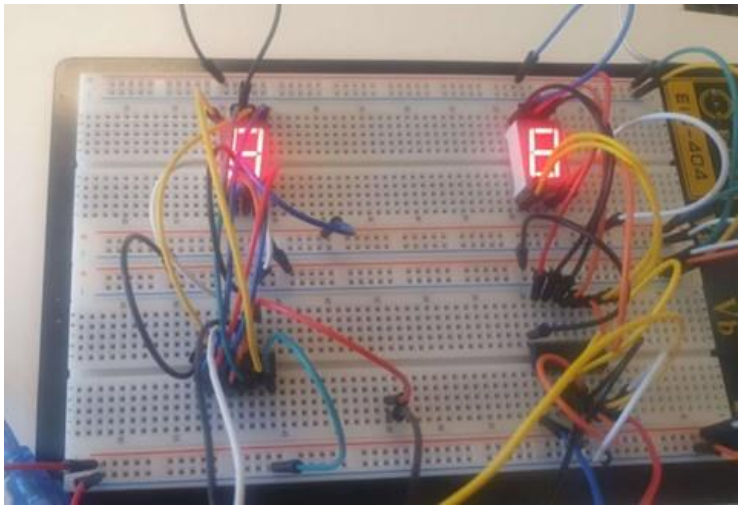

To test the correct function of the SN74HC595N shifter, we will connect the input pins of the shifter to the Arduino board, and then connect the output pins of the shifter to the 7-segment display. Once the physical connections are created, a code will be uploaded to the Arduino board, which will automatically output the decimal numbers 0 to 9 in a serial format. The shifter IC will then convert the serial code into parallel data and send it to the 7-segment display.




Functionality of the BCD decoder in combination with one shift register

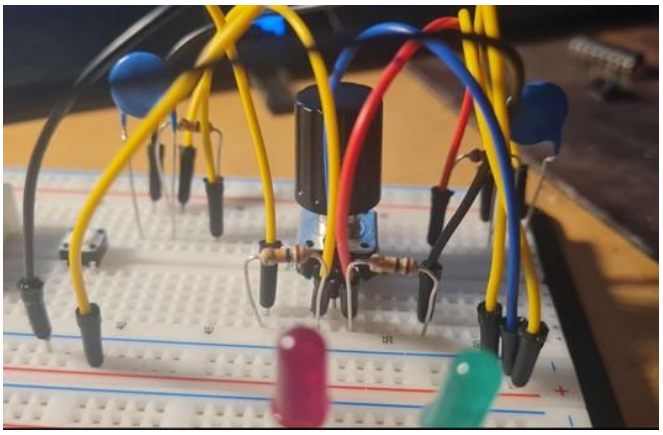


Testing 2 shift registors at the same time

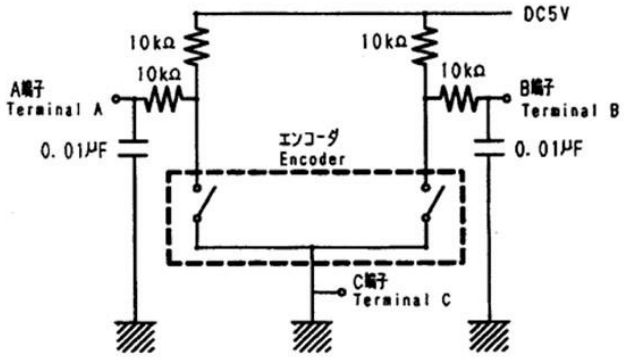


## Rotary Encoder

An experimental evaluation of the EC12E2444 rotary encoder electrical component was conducted. The rotary encoder was connected to an Arduino board and its output pins were connected to an LED display. The Arduino code was written to read the input from the rotary encoder and to show the corresponding output on the LED display. The experiment was performed by rotating the knob of the rotary encoder and observing the changes in the LED display. The results of the experiment showed that the EC12E2444 rotary encoder accurately converted the rotational movement of the knob into digital signals, which were then transmitted to the Arduino board and displayed on the LED screen. This experiment verified the proper functioning of the EC12E2444 rotary encoder component and its compatibility with the Arduino board.



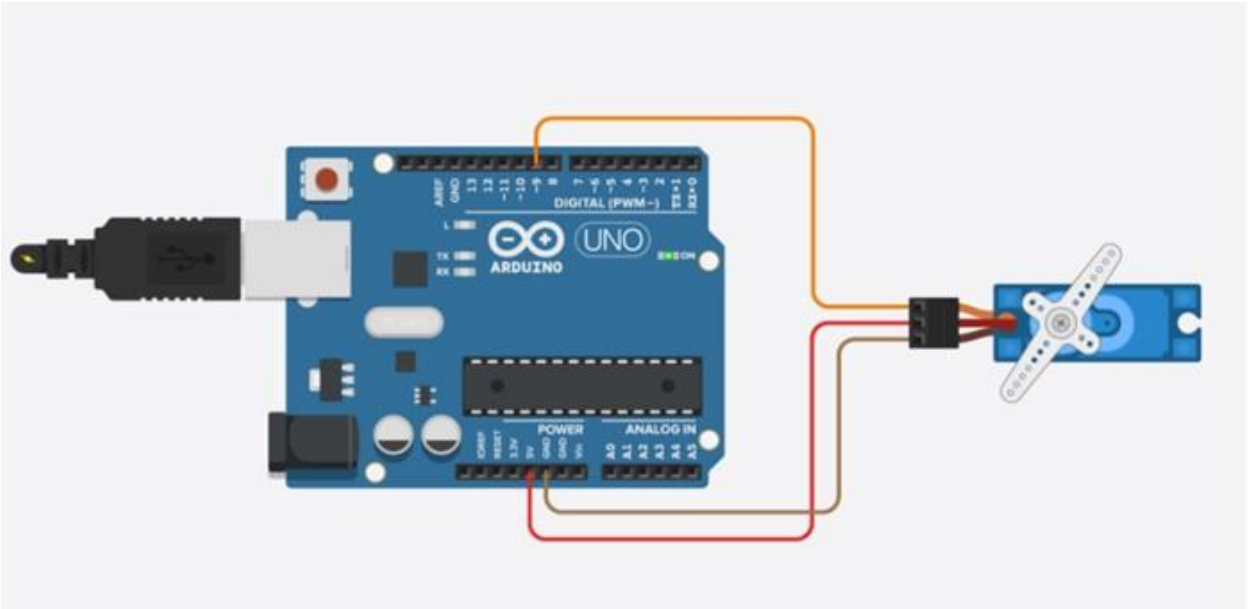
The process of debouncing the rotary encoder is thoroughly detailed within the technical documentation of the component, specifically in the datasheet. This information is critical for ensuring proper operation of the encoder within the overall Printed Circuit Board Assembly. It is important to reference and implement the debouncing techniques outlined in the datasheet to prevent any inaccuracies or malfunctions in the system.




## Servo Motor

A servo motor is a type of electrical motor that is commonly used in robotics and other precision control applications. In the specific case of our vault project, the servo motor is utilized to lock and unlock the vault. It is a closed-loop control system that uses position feedback to maintain the precise angular position of the output shaft. Servo motors are typically composed of three main parts: a motor, a control circuit, and a gear train. The motor provides the rotational power for the servo and can be either a DC motor or a brushless DC motor. The control circuit receives position commands from an external controller, such as Arduino, and uses this information to drive the motor. The gear train converts the high-speed, low-torque output of the motor into a slower, higher-torque output that is used to rotate the servo's output shaft.

Testing the servo motor is an important step in ensuring the proper function of the overall circuit and its ability to control the locking mechanism of the vault.




## Buzzer

A buzzer in a circuit is a device that generates an audible sound when a specific input is provided. The buzzer is connected to the circuit and is activated when the input is received. When the input signal is received, the control circuit sends a current to the buzzer, which causes it to generate an audible sound. In the circuit, the buzzer buzzes when the input from the potentiometer is received.

## Leds

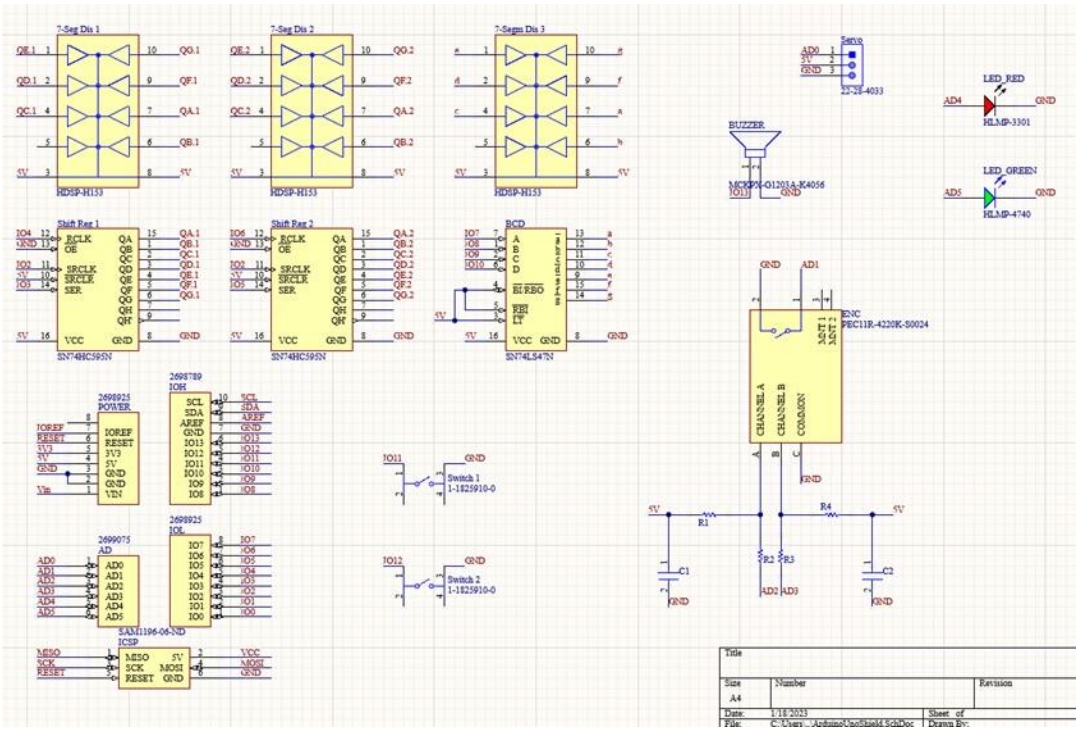
There are two leds in our circuit, a red one and a green one. When the input signal is received, the control circuit sends a current to the LED, which causes it to light up. Depending on the input either one of the leds lights up or blinks.

## Buttons

The buttons are connected to a power source, and to a control circuit. The button is connected to the control circuit through a pair of wires, one for the power supply and one for the input signal. When the button is pressed, it completes the circuit, allowing current to flow and triggering the control circuit to take an action. One of the buttons reset the password while the other one close the lock by sending input signals.

## Altium Design

The integration of multiple components, including two shift registers, one binary-coded decimal (BCD) converter, three 7-segment displays, one rotary encoder, one buzzer, one header for interfacing with a servo motor, two buttons, and two LED, into the Altium software platform, in order to create a schematic diagram. Additionally, the necessary pin connections for interfacing with an Arduino microcontroller are incorporated into the design.



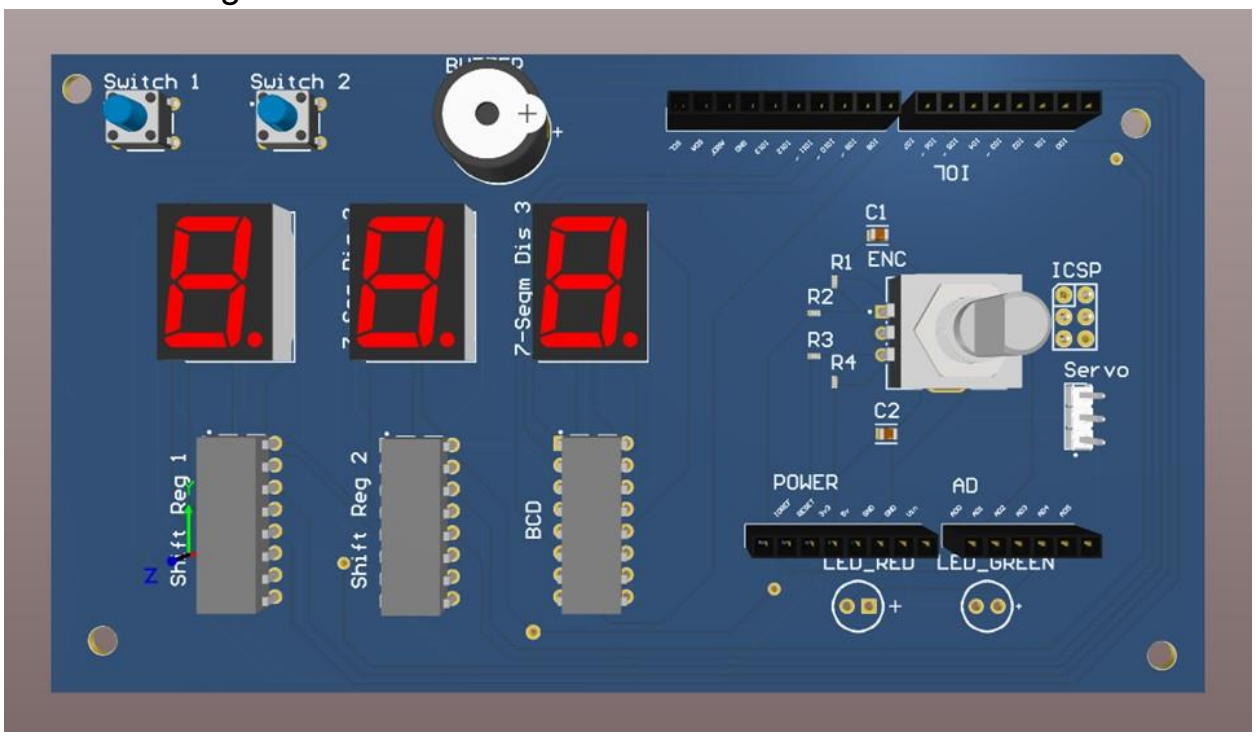

Selection of specific components for implementation:

- Shift register: HDSP-H153
- BCD converter: SN74LS47N
- Switch: 1-1825910-0
- Rotary encoder: PEC11R-4220K-S0024 (due to unavailability of EC12E24244)
- Buzzer: MCKPX-G1203A-K4056 (as the closest alternative)
- Green LED: HLMP-4740
- Red LED: HLMP-3301

- Header for servo motor: 22-28-4033

Note that the component values were chosen based on technical specifications and availability.

The final design of the PCB:




The PCB utilizes two shift registers (SN74HC595N) to expand the number of output pins available to control the three 7-segment displays, which are used to display the code for the vault. A BCD decoder (74LS47N) is used to convert binary coded decimal inputs to the corresponding decimal outputs for the 7-segment displays. The rotary encoder (EC12E2444) is used to provide an input for entering the code to unlock the vault. There is a button for resetting the code on the screen, another button simulates the door opening and closing, a buzzer and LEDs are used to indicate if the code entered is wrong or when the door is open. Overall, this PCB is a control and monitoring circuit for a vault which can be controlled by an Arduino microcontroller and provide visual and audio feedback to the user.

# General:

### Unit testing

For unit testing, we started by typing every function one by one and simplified ones that were not usable alone and for functions that do not have a visible effect on their own we used series. We tested the code for the shift register, BCD decoder, buzzer, led, servo, independent buttons, clear input button, and the rotary encoder and its button.

# Chapter 4: Implementation and testing

# Software:

### 1-Input for the 3 displays

Use an if statement in the void loop to check if the rotary encoder has turned and then call the function “changeNum” based on the direction in which the rotary rotates and that relates to input for 3 displays.

### 2-if the password was being changed

It is a static bool named” changingPass” that gets checked with whether the safe is unlocked. If both are true, then we would change the password and save it to memory. If changingPass is false, then we would check the password using the if statement to check the position of the password.

### 3- compare the input with the password and start the correlative procedure

The function “pass Completed” returns a bool to the static bool “unlocked”. If it is wrong, the red LED and the buzzer is turned on to indicate that the input was incorrect and then reset the “current Pass”  array to  {0,0,0}. ”. If it is right the green LED and the buzzer is turned on to indicate that the input was correct, and the door opens because it returns a bool and if “changingPass” is false then else if statement is executed.

### 4-Check for the door

The function “doorButton” is executed to check whether the door button was pressed and if the door button is pressed rest “pos” to 0 and unlocked to false and reset the “currentPass” array to {0,0,0}.

### 5-Changing the password

Calling the function “changePass” to check the rotary button if it was pressed for 3 sec. If yes, “changingPass” is set to true, and “currentPass” is set to {0,0,0}.

# Hardware:

## Description of the Printed Circuit Board Assembly

The Printed Circuit Board Assembly (PCBA) process for our vault project began by preparing our work area and gathering all necessary components and our PCB. The first step was to use a pick and place machine to place the resistors onto the PCB, then the PCB was placed into an oven for reflow soldering to ensure proper sticking of the resistors. Next, we soldered the remaining components such as the shift registers, BCD decoder, 7-segment display, rotary encoder, buzzer, buttons, LEDs, and headers onto the PCB. Once all the components were soldered in place, we carefully inspected the PCB for any mistakes or errors. Finally, we tested the assembled board to ensure that it functions as intended and met the requirements for controlling and monitoring the vault's locking mechanism.

# Test results:

## Results for unit testing:

The shift register worked as intended as well as the BCD but the way the 6 and 9 are displayed by the BCD was different from what was displayed by the shift registers, so we changed the part of the code relating to these 2 numbers. The buzzer, led, servo, and some buttons worked as intended. But the rotary encoder sometimes jumped two numbers, and 1 button kept being pressed hundreds of times, we then identified the problem in the code and fixed it.

## Results Functional testing:

When running the code there were issues in saving the password, but other than that the rest were fine. The codes to check the password and get the desired response wore working fine, as well as the inputting and changing password functions

## Results of Technical test

There were issues with the power because we used a power bank without a ground. And when changing the password every time it was implanted, the selected display did not change to the next one afterward, and we had to recode the changing password functions. However, other than that, there were no issues.

# Code

```
#include <Servo.h>
#include <EEPROM.h>

// Rotary encoder pins
#define ROTARYA A2
#define ROTARYB A3
#define ROTBUTTON A1

#define SERVOPIN A0
Servo vaultServo;


#define BUZZER 13
#define REDLED A4
#define GREENLED A5
#define DOORBUTTON 11
#define RESETBUTTON 12

#define SHIFTCLK 2

const int SHIFT1[2] = {
  3, // Data
  4 // Latch
};

const int SHIFT2[2] = {
  5, // Data
  6 // Latch
};

#define BCD1 7
#define BCD2 8
#define BCD3 9
#define BCD4 10

const int RomPass1 = 0;
const int RomPass2 = 2;
const int RomPass3 = 4;
const int RomEdited = 6;
bool value;

byte bcdNum[11][4] = {
  { 0, 0, 0, 0 }, // 0
  { 1, 0, 0, 0 }, // 1
```

```
  { 0, 1, 0, 0 }, // 2
  { 1, 1, 0, 0 },  // 3
  { 0, 0, 1, 0 },  // 4
  { 1, 0, 1, 0 },  // 5
  { 0, 1, 1, 0 },  // 6
  { 1, 1, 1, 0 },  // 7
  { 0, 0, 0, 1 },  // 8
  { 1, 0, 0, 1 },  // 9
  { 1, 1, 1, 1 }   // Off
}; int
lastStateClk;


const char common = 'c'; // 7-segment with common cathode
// cases for detecting the binary of the numbers given from 0 to 9
*/ byte binary(int Number) { switch (Number) { case 0:
      return             // if 0 is given then return binary for zero
    B11111100; break;
    case 1:
      return             // if 1 is given then return binary for one
    B01100000; break;
    case 2:
      return             // if 2 is given then return binary for two
    B11011010; break;
    case 3:
      return             // if 3 is given then return binary for three
    B11110010; break;
    case 4:
      return             // if 4 is given then return binary for four
    B01100110; break;
    case 5:
      return             // if 5 is given then return binary for five
    B10110110; break;
    case 6:
      return             // if 6 is given then return binary for six
    B00111110; break;
    case 7:
      return             // if 7 is given then return binary for seven
    B11100000; break;
    case 8:
      return             // if 8 is given then return binary for eight
    B11111110; break;
    case 9:
      return B11100110;  // if 9 is given then return binary for nine
    case 10:
      return B00000000;  // if 10 is given then set a blank screen
  } return
  B00000000;
}
```

```
// Function to write an int into the EEPROM. EEPROM uses bytes too store
data, and since int is 2 bytes we place it in accordingly using bitwise
operations void writeIntIntoEEPROM(int address, int number) {
  byte byte1 = number >> 8;
  byte byte2 = number & 0xFF;
  EEPROM.write(address, byte1);
  EEPROM.write(address + 1, byte2);
}

// Reading from EEPROM and returning a read int. The int is taken from 2
bytes, so it needs to be reconstucted using bitwise operations int
readIntFromEEPROM(int address) {
  return (EEPROM.read(address) << 8) + EEPROM.read(address + 1);
}

// Function used to display a byte number to the 7-segment
void display(byte eightBits, const int shift[]) {
  if (common == 'c') { // If the configuration is common cathode eightBits
= eightBits ^ B11111111; // Change the bits from 0 to 1
  }
  digitalWrite(shift[1], LOW); // Preparing shift register for data1
  shiftOut(shift[0], SHIFTCLK, LSBFIRST, eightBits); // Sending the data1 to
7-segmnet starting from the least significant bit
  digitalWrite(shift[1], HIGH);     // Displaying the data1 on the 7
segment }

// Writing the BCD number through individual pins using the bcdNum
array initialized at the top void writeNum(byte num) {
  digitalWrite(BCD1, bcdNum[num][0]);
  digitalWrite(BCD2, bcdNum[num][1]);
  digitalWrite(BCD3, bcdNum[num][2]);
  digitalWrite(BCD4, bcdNum[num][3]);
}

// Change the numbers in the array using the rotary encoder void
changeNum(int *currentStateClk, int *currentPass, int *pos) {
  // If the data from ROTARYB is different from the current state that means
we are
  // rotating CW so we increment
  if (digitalRead(ROTARYB) != *currentStateClk) {
    // Check if the password is getting higher than 18. We use 18 becuase
even though the
    // rotary encoder is debounced, it still increments and decrements
in values of 2 if (currentPass[*pos] > 18) {
      currentPass[*pos] = 0;
    } else {
      currentPass[*pos]++;
```

```
    }
  // Else the rotary is rotating CCW so we decrease the number
  } else { if
    (currentPass[*pos] < 0) {
      currentPass[*pos] = 18;
    } else {
      currentPass[*pos]--;
    }
  }
}

// Check if the rotary button is pressed to move the pos of the password
void nextButtonCheck(int *pos) {
  int buttonState = digitalRead(ROTBUTTON);
  // Since it is a INPUT_PULLUP it means the button has an input of LOW when
being pressed
  if (buttonState == LOW) {
    // Wait for user to release button
    while (buttonState == LOW) {
      buttonState = digitalRead(ROTBUTTON);
      delay(10);
    }
    *pos = *pos + 1;
  }
}

// Check whether the password is correct and return a bool
bool passCompleted(int *originalPass, int *currentPass) {
  // Check the password
  for (int i = 0; i < 3; i++) {
    // If it is wrong, reset the currentPass and display red led and buzzer
    if (originalPass[i] != currentPass[i] / 2) {
      for (int j = 0; j < 3; j++) {
        currentPass[j] = 0;
      } digitalWrite(BUZZER,
      HIGH);
      digitalWrite(REDLED,
      HIGH); delay(50);
      digitalWrite(BUZZER, LOW);
      delay(50);
      digitalWrite(BUZZER,
      HIGH); delay(150);
      digitalWrite(BUZZER, LOW);
      digitalWrite(REDLED, LOW);
      return false;
    }
```

```
    // If the end is reached and everything matches, return true after
turning on buzzer
    if (i == 2) {
      for (int j = 0; j < 2; j++) {
        digitalWrite(BUZZER, HIGH);
        delay(50);
        digitalWrite(BUZZER, LOW);
        delay(50);
      } return
      true;
    }
  }
}

// Button that resets user input void
resetButton(int *currentPass, int *pos) {
  int resetState = digitalRead(RESETBUTTON);
  if (resetState == LOW) {
    while (resetState == LOW) {
      resetState = digitalRead(RESETBUTTON);
      delay(10);
    }
    // Set the currentPass to 0
    for (int i = 0; i < 3; i++) {
      currentPass[i] = 0;
      *pos = 0;
    }
  }
}

// Check if the door button is pressed
void doorButton(int *currentPass, int *pos, bool *unlocked) {
  int doorState = digitalRead(DOORBUTTON);
  if (doorState == LOW) {
    // Wait fopr user to release button
    while (doorState == LOW) {
      doorState = digitalRead(DOORBUTTON);
      delay(10);
    }
    for (int i = 0; i < 3; i++) {
      currentPass[i] = 0;
    }
    // Lock after resetting
    *pos = 0;
    *unlocked = false;
  }
}
```

```
// Change the password after a set delay
void changePass(int *currentPass, int changeState, bool *pressed, int
*pressedTime, int *releaseTime, bool *changingPass, const int period) {
  if (changeState == LOW && !*pressed) {
    // Record time that button was pressed and mark that the button was
    pressed
    *pressedTime = millis();
    *pressed = true;
  } else if (changeState == HIGH && *pressed) {
    // release button and set release time
    *pressed = false;
    *releaseTime = millis();
    // calculate the difference of the times and check if they are more
than the set delay if ((*releaseTime - *pressedTime) > period) {
*changingPass = true;
    }
  }
  // Reset currentPass after changing the saved password
  for (int i = 0; i < 3; i++) {
    currentPass[i] = 0;
  } } void

setup() {

  // Set all the pins
  pinMode(ROTARYA, INPUT);
  pinMode(ROTARYB, INPUT);
  pinMode(ROTBUTTON, INPUT_PULLUP);
  vaultServo.attach(SERVOPIN);

  pinMode(SHIFT1[0], OUTPUT);
  pinMode(SHIFT1[1], OUTPUT);
  pinMode(SHIFTCLK, OUTPUT);
  pinMode(SHIFT2[0], OUTPUT);
  pinMode(SHIFT2[1], OUTPUT);

  pinMode(BCD1, OUTPUT);
  pinMode(BCD2, OUTPUT);
  pinMode(BCD3, OUTPUT);
  pinMode(BCD4, OUTPUT);

  pinMode(GREENLED, OUTPUT);
  pinMode(REDLED, OUTPUT); pinMode(BUZZER,
  OUTPUT); pinMode(DOORBUTTON,
  INPUT_PULLUP); pinMode(RESETBUTTON,
  INPUT_PULLUP);
```

```
  // Set bcd display to be empty
  digitalWrite(BCD1, 1);
  digitalWrite(BCD2, 1);
  digitalWrite(BCD3, 1);
  digitalWrite(BCD4, 1);

  lastStateClk = digitalRead(ROTARYA); value
  = (bool)readIntFromEEPROM(RomEdited);
  // If RomEdited has no value, assign the password 123
  if (!value) {
    writeIntIntoEEPROM(RomPass1, 1);
    writeIntIntoEEPROM(RomPass2, 2);
    writeIntIntoEEPROM(RomPass3, 3);
    writeIntIntoEEPROM(RomEdited, 1);
  }
}

void loop() {
  static bool unlocked = false;
  static int pressedTime; static
  bool pressed = false; static int
  releasedTime; static const int
  period = 3000; static int
  currentStateClk; static int
  blinkTime = millis(); static const
  int blinkPeriod = 250; static bool
  changingPass = false;

  static int originalPass[3] = { 1, 2, 3 };
  static int currentPass[3] = { 0, 0, 0 };
  static int pos = 0;

  // Runs if vault is lcoked if
  (!unlocked) { // Turn green
  led off
  digitalWrite(GREENLED, LOW);

    // Read from the EEPROM and set is as original
    pass originalPass[0] =
    readIntFromEEPROM(RomPass1); originalPass[1] =
    readIntFromEEPROM(RomPass2); originalPass[2] =
    readIntFromEEPROM(RomPass3); // Read the current
    state of the clock currentStateClk =
    digitalRead(ROTARYA); vaultServo.write(180);

    // If last and current state of CLK are different, then pulse occurred
```

```
    // React to only 1 state change to avoid double count
    if (currentStateClk != lastStateClk) {
      changeNum(&currentStateClk, currentPass, &pos);
    }
    // Remember last CLK state
    lastStateClk = currentStateClk;
    nextButtonCheck(&pos);
    // At end of array submit the password
    if (pos > 2) {
      pos = 0;
      unlocked = passCompleted(originalPass, currentPass);
    } resetButton(currentPass,
    &pos);

  }

  // If unlocked and not changing the passwrod
  else if (unlocked && !changingPass) {
    int changeState =
    digitalRead(ROTBUTTON);

    // Unlock the vault and turn on the green
    led vaultServo.write(90);
    digitalWrite(GREENLED, HIGH);

    // Check for door button and the password change button
doorButton(currentPass, &pos, &unlocked); changePass(currentPass,
changeState, &pressed, &pressedTime, &releasedTime, &changingPass, period);
  }

  // If unlocked and changing the password

  else if (unlocked && changingPass) {

  currentStateClk = digitalRead(ROTARYA);

    // If last and current state of CLK are different, then pulse occurred
    // React to only 1 state change to avoid double count
    if (currentStateClk != lastStateClk) {
      changeNum(&currentStateClk, currentPass, &pos);
    }
    lastStateClk = currentStateClk;
    // Check for next and reset buttons being pressed
    resetButton(currentPass, &pos);
    nextButtonCheck(&pos);
    // At end of password replace the originalPass with current pass. We use
currentPass[i] / 2 because
    // the password increments by 2, and we divide by 2 to compensate
```

```
    if (pos > 2) {
      pos = 0; for (int i = 0; i <
      3; i++) {
        originalPass[i] = currentPass[i] /
        2;
      }
      // Stop chaning the pass and lock the bault once password is
      changed changingPass = false; unlocked = false; // Reset
      currentPass
      for (int j = 0; j < 3; j++) {
        currentPass[j] = 0;
      }
      // Add the new pass to EEPROM
      writeIntIntoEEPROM(RomPass1, originalPass[0]);
      writeIntIntoEEPROM(RomPass2, originalPass[1]);
      writeIntIntoEEPROM(RomPass3, originalPass[2]);
      writeIntIntoEEPROM(RomEdited, 1);
    }
    // This blinks the green led to signify the changing of the password
    if ((millis() % blinkPeriod) > (blinkPeriod / 2)) {
      digitalWrite(GREENLED, LOW);
    } else if ((millis() % blinkPeriod) < (blinkPeriod / 2)) {
      digitalWrite(GREENLED, HIGH);
    } resetButton(currentPass,
    &pos);
  }

  // Display 2 numbers on the shift, and one on the bcd. We devide by 2
to compensate for the 2 inputs that one rotation // that the rotary
encoder gives us display(binary((currentPass[0]) / 2), SHIFT1);
display(binary(currentPass[1] / 2), SHIFT2); writeNum(currentPass[2] /
2);
}
```