Mini-Project:
Digital Door Lock

Jack Gatfield & Joseph Gozum
EE128 Fall 2018

---

https://www.youtube.com/watch?v=VnQdv9X3OQI&feature=youtu.be

Table of Contents

Description

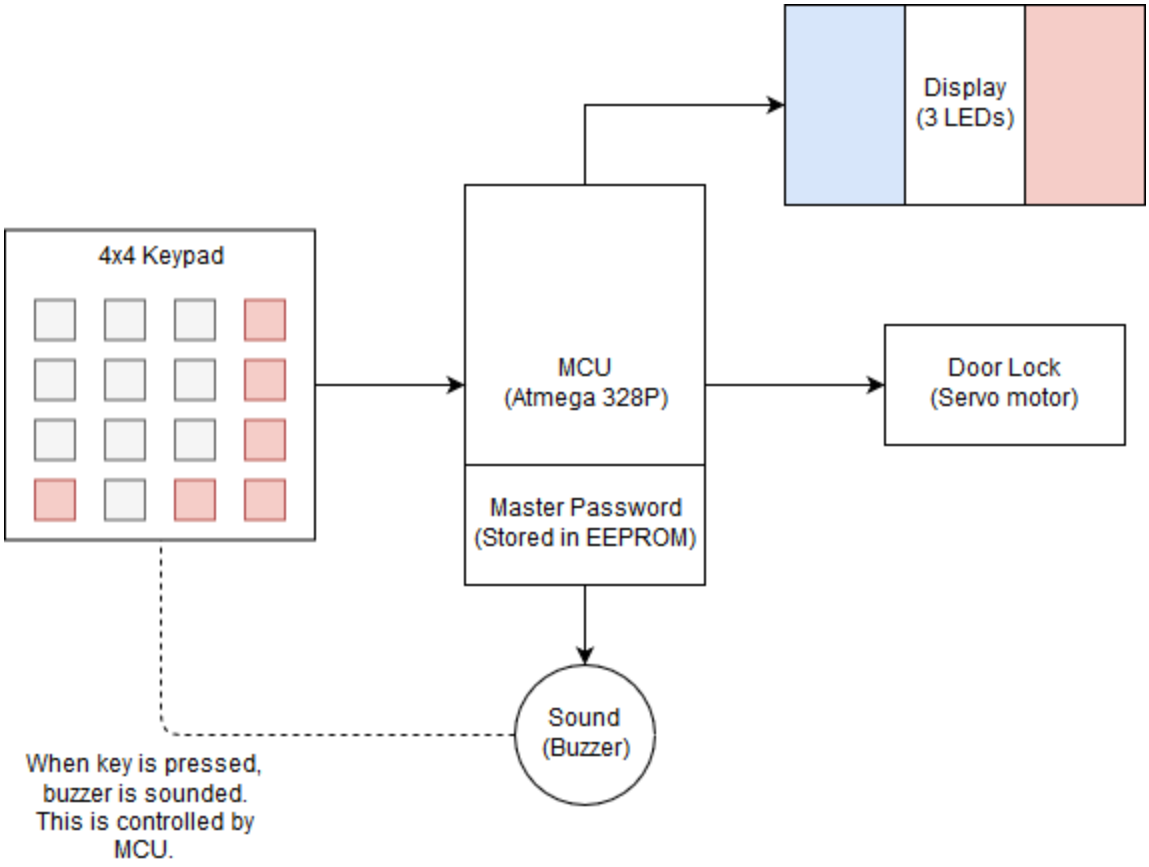
_____Our project is a digital door lock control system with the following capabilities:

Requirements

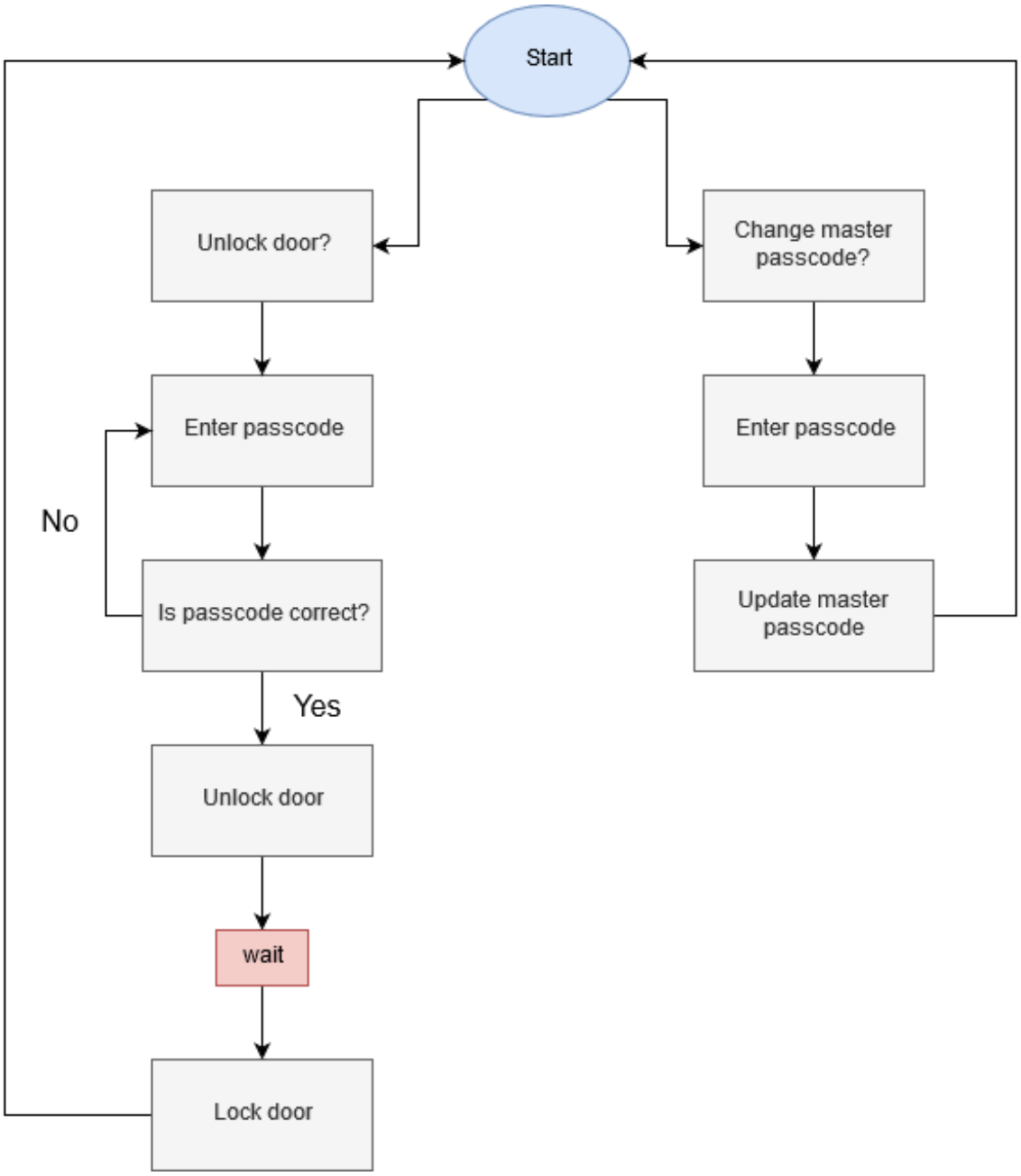
1. Receive input from the user via a 4x4 keypad
2. Unlock “door” (the servo) when the correct password is inputted
    a. Rotate the servo to represent unlocking
3. Be able to update the master password which is stored in EEPROM when ‘#’ entered 4 times
    a. Must enter old password to be able to enter a new password
4. A buzzer that goes off in the following situations:
    a. Receive input from the keypad
    b. Entering new password mode
    c. Incorrect password inputted
    d. Consecutive wrong number reaches the threshold
5. LEDs to represent the different situations:
    a. White LED and Blue LED on when entering a new password
    b. Red LED on when an incorrect password is entered
    c. Blue LED on when waiting for keypad input
    d. White LED on for when the lock is open

System Design

High-level Block Diagram



Logic Flowchart



Implementation Details

Materials

To implement this project we used:

- Arduino Uno
- 4 x 4 keypad
- 3 LEDs
    - Red, White, Blue
- 4 resistors
- Buzzer
- Servo

Uses

The Arduino Uno provided all logic and power functions needed in our project. It was effective for our project but it limited the peripherals we could use due to the limited pin counts. Even though we had limited pins we were able to achieve the functionality we set out to do.

The 4 x 4 keypad allowed the user to input the codes and interact with the system. It was effective for our project but during development, we had lots of hardware bugs. Two different keypads had two different dead zones on them. We tried to fix the issues but we concluded that we could not fix the zones and acquired a new keypad. Once we had a new keypad wired up the hardware bugs went away and we were able to focus on the coding.

The LEDs and buzzer provided visual and auditory information to the user regarding system state and what is happening. The buzzer had multiple modes.

- Chirp for when a keystroke was captured
- Short ring for when the entered passcode was wrong
- 3 chirps for entering password update mode telling the user to input old password
- Long ring for when the entered password was wrong too many consecutive times

This functionality was implemented using function calls and setting PWM pins to different duty cycles and toggling output on and off. The function calls used to implement the buzzer blocked the program and made it slower but it made implementation cleaner and the speed did not affect our system. The compiler also can

be set to use certain optimization techniques, such as inlines, to speed up the code if needed. This speed difference was not observable to the user.
The LEDs being on had different meanings.

- White on means the lock is open
- Red on mean the previously entered password was wrong
- Blue on means the system is ready and waiting for input
- White and Blue on means the system is in the update password branch

Implementing the LEDs had many hardware bugs to fix before proper functionality was achieved. We are not sure where the bugs came from but we were able to work through them and achieve our specifications

System Operation

Our project is a digital door lock. When looking at the system you will see a keypad, a servo motor, white red and blue LEDs, and a buzzer. With the blue led on the system is ready for input from the keypad. The user then can click a key and hear a chirp from the buzzer. This means that the input was taken in. Once enough keystrokes are captured for the passcode length (demo length was 4) the system will take three paths, one for a correct passcode, one for an incorrect passcode, and one for the update passcode code.

For a correct passcode, the system will open the servo and turn on the white led. For an incorrect passcode, the system will turn on the red led and ring the buzzer. The length of this response is dependent on the number of consecutive wrong entered passcodes. For the update passcode path, the system will ring the buzzer a few times and then turn on the white and blue led asking for the old passcode. When the correct old passcode is entered the white and blue led will blink and then the system will take in the user input and make that input the new master passcode. If an incorrect passcode is entered when the system asks for the old one it will break out of the branch and enter the incorrect passcode branch.

Important Snippets of Code

EEPROM Update

```
void updatePasscode()
{
  digitalWrite(WhiteLED,High);                    // Turns on White LED for visual cue
  digitalWrite(BlueLED, High);                    // Turns on Blue LED for visual cue

  EEPROM.update(PASSCODE_FLAG, 1);       // Sets flag to let system now master
passcode exists in EEPROM
  getPasscode();                  // Calls getPasscode() to get new passcode to store in
EEPROM

  for(int i = START_ADDRESS; i <= END_ADDRESS; i++)    // Updates EEPROM with new
passcode
  {
        EEPROM.update(i, passcode[i]);
  }

  digitalWrite(WhiteLED,Low);                     // Turns off White LED
  digitalWrite(BlueLED, Low);                     // Turns off Red LED
}
```

Password Check Function

```
bool isPasscode()
{
  unsigned char matched = 0;                    //Variable to monitor passcode matching
  for(int i = START_ADDRESS; i <= END_ADDRESS; i++)          //Iterates through passcode
  {
        if(EEPROM.read(i) == passcode[i])
        {
        matched++;                          //Saves number of matched characters
        }
        else
        {
        digitalWrite(RedLED,High);          // Sets wrong LED --> Red on
        totalWrong++;                       // Increases total wrong count for alarm system
        WrongPasscodeBuzzer();              // Sounds buzzer
        digitalWrite(RedLED,Low);           // Sets wrong LED --> Red off
        return false;                       // Returns false indicating entered passcode is wrong
        }
  }
```

```
  if(matched == 4)
  {
        totalWrong = 0;                    // Resets total number wrong for alarm system due to correct input
        return true;                       // Returns true if entered passcode matched saved passcode
  }
  return false;                            // Default value
}
```

One buzzer function

```
void WrongPasscodeBuzzer()
{
  if(totalWrong >= 3) { // Checks if the total number of wrong passcodes is 3 or more If greater a special alarm goes off
        analogWrite(BuzzerPin, BuzzerTone);     // Turns buzzzer on
        delay(3000);                            // Waits 3 seconds (Demo time length) with buzzer on
        analogWrite(BuzzerPin, 0);              // Turns buzzer off
        totalWrong = 0;                         // Resets total number of wrong passcodes for alarm system
  }
  else{
        analogWrite(BuzzerPin, BuzzerTone);     //Turns buzzer on
        delay(500);                             // Waits .5 seconds with buzzer on
        analogWrite(BuzzerPin, 0);              //Turns buzzer off
  }
}
```

Testing and Evaluation

Video Demo - https://www.youtube.com/watch?v=VnQdv9X3OQI&feature=youtu.be

Test Benches

We did incremental testing. First, we tested if we were indeed getting keypad input. We checked this with print statements to the Arduino serial terminal. Then we tested if we could populate an array with keypad input characters. We checked this with loops and print statements to the Arduino serial terminal. Then we tested if our buzzer and LEDs were working to the specifications. We tested by having different situations that require different led and buzzer operations and checked output. Then we tested if we could control the servo motor. We did this by sending different positions to the motor and seeing if it did the correct movement.

Once we knew the subsystems were working independently we put them together and tested if the overall system was working. We did this by having a hardcoded passcode in memory and testing if the system responded to getting the correct or incorrect passcode properly and if it controlled the motor correctly. When the system was working with a hard-coded master passcode we implemented a way to update the memory with a user-defined passcode. We then tested this system the same way as we tested the system with a set master passcode.

Evaluation:

When our testing showed our system was working properly we went to evaluate our project.

We tried different combinations of the passcode to make sure it worked for all possible values excluding the update passcode combination (####). We also checked that passcode order mattered and that there was no obvious security hole.

We then check that if powering off and on the system opened a security hole. A power cycling did not break the security, the passcode was still set and the system needed a correct code before opening the servo.

Once we observed that the passcode and servo system worked for all cases we check that the buzzer and LEDs were following the system state and providing the correct system information to the user. We tested all cases and once again the systems worked as intended.

Conclusion:

Through our testing and evaluation of the project we saw and proved that our system was working as intended and meet our desired functionality. With edge cases, unusual use, and normal use of the system it functioned correctly and stayed secured.

Discussions

Challenges:

1. We were experiencing problems with certain keys not working on several different keypads.
2. Code to verify user-input from keypad matched master password did not always work
3. Limitations on amount of add-ons due to lack of pins, and the LCD screen that could work with the pin count was not working fully then eventually fell apart

Solutions: (Correspond to challenges)

1. Received replacement keypad from IEEE-UCR as well as changing the code we used to receive key input. We got help by using the official Arduino library for keypads.
2. Re-did the code for this portion
3. We did not have enough pins for our LCD screen. The LCD that could operate with the number of pins was damaged and did not function properly and the other LCD screen required too many pins so we simply used LEDs to show the operating mode of the digital door lock.
4. Demo problem solutions:
    a. Before changing the passcode you now need to input the old passcode
    b. In setup() we were also resetting the master passcode to some random combination instead of remembering the old master passcode

Improvements since demo:

1. Before changing the passcode you now need to input the old password
2. Setup() no longer resets the passcode and instead, the system checks EEPROM to see if a passcode is saved otherwise it asks for a passcode to use

Roles & Responsibilities of Group Members

Joseph Gozum:

- Implemented code for the following:
    - Controlling the servo (which acts as our door lock)
    - Receiving keypad input
    - Updating EEPROM memory that contains master password
    - The logic to change the master password
    - The logic to check if the inputted password matches the master

Jack Gatfield:

- Implemented code for the following:
    - Buzzer that gives user-feedback in the following situations:
        - Entering new master password
        - Incorrect password entered
        - When a key is pressed on the keypad
        - When the number of consecutive wrong password attempts is greater than a threshold
    - LEDs to tell the operating mode the door lock is in
        - Ready for input
        - Wrong password
        - Lock open
        - Waiting for the current password before allowing to update the password
    - Security logic for changing the password
- Commented the code and improved readability and variable naming

Conclusion

We set out to make a door lock that fulfilled the previously stated specifications. We had hardware and software problems along the way but we were able to work through them and create fixes to allow us to carry on with developing the project. At the end of the development period, we created a project that fulfilled all our specifications and were able to improve it with the feedback we received in the demo. The project meets all specifications and was an overall success.