Dorm Door Locking Mechanism

Mason Hoppe

Thomas Orr

Karan Usgaonkar

**Abstract**

Our attachable door lock adds extra security to dorm door locks without damaging or changing the lock or door. After attaching the device right to the deadbolt knob on the inside the room, it will be able to automatically lock the door when the door swings closed. To further increase security, we have added two new methods of unlocking the door. Both an RFID scanner and a keypad outside the door offer new ways to unlock the door without removing the option of using a key. The low-power device runs primarily on wall power but it has a long lasting battery backup in case of power outages or fluctuations. Although some of the methods have changed since our original design, our project ended up meeting all requirements and specifications and has been demonstrated to execute its role consistently.

**Contents**

[**1. Introduction** 5](#_Toc500966390)

[**2 Design** 6](#_Toc500966391)

[2.1 System Power 6](#_Toc500966392)

[2.1.1 Power Circuit Schematic 6](#_Toc500966393)

[2.1.3 Resistor Values 6](#_Toc500966394)

[2.2 Outside Interface 7](#_Toc500966395)

[2.2.1 RFID 7](#_Toc500966396)

[2.2.2 Keypad 8](#_Toc500966397)

[2.3 Lock Mechanism 8](#_Toc500966398)

[2.4 Inside Interface 9](#_Toc500966399)

[2.4.1 Proximity Sensor 9](#_Toc500966400)

[2.4.1 User Interface 9](#_Toc500966401)

[2.5 Microcontroller 10](#_Toc500966402)

[2.5.2 Device Interfaces 10](#_Toc500966403)

[2.5.3 Open / Close Logic 10](#_Toc500966404)

[**3. Design Verification** 12](#_Toc500966405)

[3.1 Power Supply 12](#_Toc500966406)

[3.2 Outside Interface Verification 12](#_Toc500966407)

[3.3 Inside Interface Verification 13](#_Toc500966408)

[3.4 Microcontroller Verification 13](#_Toc500966409)

[**4. Costs** 14](#_Toc500966410)

[4.1 Parts 14](#_Toc500966411)

[4.2 Labor 14](#_Toc500966412)

[**5. Conclusion** 15](#_Toc500966413)

[5.1 Accomplishments 15](#_Toc500966414)

[5.2 Difficulties 15](#_Toc500966415)

[5.3 Ethical considerations 15](#_Toc500966416)

[5.4 Future work 15](#_Toc500966417)

[**Figures and Tables** 16](#_Toc500966418)

[**References** 20](#_Toc500966419)

[**Appendix A** **Requirement and Verification Tables** 21](#_Toc500966420)

[**Appendix B** **Resistances and Capacitances** 23](#_Toc500966421)

[**Appendix C** **ATMega328p Code** 24](#_Toc500966422)

# **1. Introduction**

The dorms on campus currently do everything that they can to prevent theft in each room. The measures that currently exist are icard scanners to enter each building and deadbolt locks on each individual student’s room. It is very easy to enter the building by simply following someone who lives there into the building, bypassing the first layer of security. Secondly, people tend to leave their doors unlocked as a personal preference or because they think that they’ll be back quickly. In either case, this creates an opportunity for dorm theft to occur. [1] In order to counteract this problem, we are creating a dorm door lock attachment which will automatically lock any dorm door that it is attached to.

One problem that naturally arises is when a student leaves their room without their key on hand. In order to prevent this from being an issue, we will be creating this attachment with alternate entry methods in mind. We will be using a pin pad and an RFID scanner to allow the student to unlock the door without needing a key. The system will be wall powered with a battery backup in case of a power outage. This breaks our system up into an external section, an internal section, and a power section. These sections are described further with our block diagram. The final result of the project we fully successful on all fronts of these requirements and also included a few additional improvements that were not a part of our original design.

**2 Design**

Our overall design includes three major sections as shown in Figure 1: the power block, the internal block, and the outside interface. The power section can be broken down into four parts: the wall voltage source, the battery charging circuit, the batteries, and the voltage regulator. Each of these parts are described in detail in system power section. The external section of the circuit includes the RFID Scanner, and a Numerical Pin Pad. Both of these are described in the Outside Interface section of the design. Finally, we have the internal section which is comprised of the User interface, Proximity Sensor, Microcontroller, and Lock Mechanism. These are described in further detail in the Inside Interface section

## **2.1 System Power**

Our system is powered through two sources: A wall input and a 6-battery, 7.2 V, pack. Our first thought on how to implement this was simply to have a trickle charge constantly flowing into the battery when the wall unit is plugged in. However, after looking through more documentation on the subject, we determined that it was better for us to have more safeguards in place for this recharging so we settled on creating a circuit that monitors the batteries as they are charging.

### **2.1.1 Power Circuit Schematic**

This system ideally uses the wall power between 10-12 V. Due to inconsistencies with the wall adapter that we used we wound up using an 18 V wall source and adjusted resistances accordingly. Figure 2 contains the schematic for our power circuit as we implemented it and Appendix 2 contain the values of the resistors and capacitors in Figure 2. Parts of this circuit comes from the datasheet for the DS2715. [2]

### **2.1.3 Resistor Values**

All of these calculations either come from the datasheet for the DS2715 [2] or from interpretation of that same datasheet. The DIV and THM ports work in tandem to determine the temperature of the chip. By placing a 10 kΩ resistor between the DIV and THM ports, R5 from Figure 2,and a 10 kΩ thermistor between the THM port and GroundBattery, the chip knows to expect the THM port to be above a certain fraction of the DIV voltage. At 50° C, the thermistor’s resistance is 3.542 kΩ. According to Equation 1, the voltage at THM must be 0.2615 \* VDIV. It is also important to note that temperature regulation was not a part of our requirements and is an added feature.

The VBatt port is meant to be the voltage across one battery. This is achieved by doing a simple voltage division using the resistor between VBatt and GroundBattery, R7 from Figure 2, to be 100 kΩ. Knowing that there are 6 batteries total, and that one of those is the voltage we want at VBatt, we can use Equation 2 to see that R8 from Figure 2 must be a 500 k resistor.

The DS2715 allows for a fast charging and a top-off charging rate. It regulates these rates via the two sense ports on the DS2715 (SNS+ and SNS-) by applying either 0.107 V or 29 mV across them. This regulates the current flowing across the resistor (R11 in Figure 2) between them. By altering the value of those resistors, we can alter the charging current. Our resistance that we selected according to Appendix 2 Table 18 is 3 . This gives a fast charging charging current of 35 mA and a top-off charging current of 9.66 mA by Equations 3 and 4.

The last important resistor for this circuit is R10 in Figure 2. This resistor is used to determine the length of time before the fast charge and top-off states for the DS2715 complete. This time can be anywhere from 30 minutes to 6 hours according to the datasheet[2]. We selected a 4 hour time out for this and selected R10 via Equation 5.

The current from the battery pack and the wall power are the same when only one of them is inserted into the system. These current draws are given in Table 1. When both the wall and battery power are connected the voltage into the battery is what matters most. In this case, we have those values listed in Table 2. These tables demonstrate that the batteries are can provide a max current (712 mA) to the circuit for an active cycle (0.5 seconds). A negative current here means that current is flowing into the battery. Using 2800 mAh batteries and assuming that 99% of the time the circuit will be in an idle mode, Equation 6 shows that the circuit can run at most for 49.12 hours on battery power alone.

|  |  |
| --- | --- |
| 3542 / (10000 + 3542) = 0.2615 | (1) |
| R7 \* (6-1) = R8 = 500 k | (2) |
| 0.107 V / 3 = 35 mA | (3) |
| 29 mV / 3 = 9.66 mA | (4) |
| R10 = (time(minutes) \* 1000) / 1.5= (240\*1000)/1.5 = 160 k | (5) |
| 2800 / 57 = 49.12 Hours | (6) |

## **2.2 Outside Interface**

The two new input devices are a 96 Series 16 Button Keypad and a 28440 Read/Write RFID Module. Using these two devices we create the possibility of a user opening their lock with a memorized PIN or with a carried RFID tag1. Both devices are mounted on the outside of the door and their wires are wrapped around the door to the PCB’s. So malicious persons aren’t able to spoof any critical signal, only raw input data is sent around the door.

### **2.2.1 RFID**

The 28400 RFID Module works with EM4x50 transponder tags and of its many operations we only use the Read to get the 32 bit serial number from a nearby tag. The serial number is compared to the correct number stored on the microcontroller. If the two numbers match, the tag is allowed to open the door.

Data is sent to and from the module through two wires on the device: Serial In and Serial Out. Using the Serial capabilities of the microcontroller and the SoftwareSerial.h arduino library the microcontroller can send and receive bytes from the RFID module at a rate of 9600 bits per second (bps).

The modules active current is far too high (see Table 5) for our low power goals so we don’t want the RFID to be active all the time. Now the module doesn’t search for a tag until the ‘\*’ button is pressed on the keypad. Pushing the button will trigger the active state and a user can then scan their tag with the device. This reduced our total average current draw by 75%

### **2.2.2 Keypad**

The 96 Series Keypad is a very robust keypad device with 16 keys on its face: 0-9, A-D and \*, #. On the back it has eight pins in total: four row pins (one pin per row) and four column pins (one pin per column); it does not require any external power or ground. Instead, when a button is pressed, the row and column lines associated with that button are mechanically connected and current will flow through them. Figure 3 details what buttons are associated with which pins. We use a simple code interface to interpret the raw data and determine which button (if any) is being pressed at a given time.

To determine which button is pressed all eight pins of the keypad are connected to the microcontroller. Half of them (rows2 ) act as outputs from the MC and the other half half (columns) act as inputs. When checking the device, each row line is raised (given a high 5 V) one at a time. The four column lines are read and whichever one is also high indicates which key is pressed. Using the row/column combo with the figure below shows what combinations of pins correspond to which buttons. It’s worth noting that multiple buttons being pressed can also be accounted for but multiple presses are fairly meaningless in our implementation so those cases are ignored.

Similar to the RFID, the microcontroller has a software interface for the keypad that helps it run smoothly with the rest of the system. As keys are read from the device they are added to a queue of the four most recent button pushes. This queue is compared to the stored PIN and when the queue matches the PIN we know the user has entered the correct PIN. For security the queue is automatically cleared after a correct entry and after a few seconds of no input.

2.2 Note 1: We would prefer our design worked with University icards, but the 28440 was more readily available. Having shown that the project works with *a* RFID tag/reader, it should be fairly straightforward to replace the 28440 with a reader that matches the icard system.

2.2 Note 2: In our case, the four row pins are the outputs from the microcontroller. In practice it can be the four row pings or the four column pins but not any four arbitrary pins.

## **2.3 Lock Mechanism**

The lock mechanism is the physical part that unlocks and locks the door using a servo motor (model 31311S HS-311) fixed to the deadbolt on the inside of the door using a machine shop created link between the deadbolt and the servo motor as pictured in Figure 4. The system takes in a PWM of 2 values: one that locks and the other that unlocks the door. The servo motor is capable of turning 180 degrees and turns based on the duty cycle of the PWM. At 4.8 V, the torque output is 3 kg/cm and the required torque to rotate the deadbolt is ~1 kg/cm. The motor needs a supply of any voltage between 4.8 V and 6 V when a PWM is supplied to move the motor. We chose 5 V as our supply to provide sufficient torque to overcome resistance. The servo thus has 3 ports: one for ground, one for 5 V, and one for the PWM. The PWM is generated by the Atmega328, which must share it’s ground with the power circuit and the ground of the servo to work correctly.

To run at 5 V (2.54 W) we use a PWM with 350 Hz and 5 V peak-to-peak. A 25% duty cycle locks the door and a duty cycle of 60% unlocks the door.

## **2.4 Inside Interface**

The two blocks that make up the Inside interface are the MP201802 Proximity Sensor and a set of three 2-pin buttons and three SPDT switches. We call the switches and buttons the User Interface (UI) because this is how the user controls how the system works. The proximity sensor detects if the door is open or closed and the user interface triggers functions like opening the lock, resetting the devices, and enabling the devices.

### **2.4.1 Proximity Sensor**

The MP201802 Proximity Sensor is a very straightforward device. It consists of the reed sensor that attaches to the door and a small magnet that attaches to the door frame. The reed sensor has two wires, one attaches to power and the other is the output. When the magnet is within half an inch (technically 0.25” - 0.7”) of the sensor, the output is close to 0 V. When the two are far from each other (>0.7”) the output wire has 5 V. Measuring this voltage therefore makes it very easy to tell of the door is open or closed.

### **2.4.1 User Interface**

We wanted to give users a measure of control over the system and that is provided through the buttons and switches. Below are the descriptions of each input.

The **Open/Close** button manually locks or unlocks the door from the inside. We use this button because the servo is arranged right on top of the knob so the knob is unreachable from the inside of the door.

For dorm rooms especially, the residents change regularly. For this reason we thought it was important that users be able to change what PIN and what RFID tags can open the door. In both cases, if nothing valid is entered for a few seconds then the Reset state will turn off and nothing will have changed. (e.g.: no tag, faulty tag, no pin, 3 digit pin).

The **Reset RFID** button puts the system in a state where the next tag that is scanned will become a valid tag to open the door with. This is useful for cases when a student loses their icard or gets a new roommate.

Similarly, the **Reset Keypad** button sets up a state where the next four keys pressed become a valid PIN.

Different users will have different preferences for security and we designed our system with this in mind. We allow users to selectively enable the two new devices so they can use whatever setup is most comfortable for them.

**Enable RFID**, when on, means that an RFID tag is a valid way to open the door.

**Enable Keypad**, when on, means that the keypad is a valid way to open the door.

Finally, the **AND/OR Mode** switch, when on, requires a valid RFID tag AND a valid PIN for the door to open.

If a switch is off (say RFID) then even when scanning the correct RFID tag the door will not open. These switches give us four combinations of inputs: RFID, keypad, keypad and RFID, and none of the above in which case the user feels only a traditional key should unlock the door.

## **2.5 Microcontroller**

The ATMega328p microcontroller takes the data from all of the other devices and logically decides if the lock mechanism should lock or unlock the door. We chose the 328 because of its flexibility and ease of use. The current output of each pin was high enough for our devices, it had 22 I/O lines (we needed 18) and it could be easily programmed / debugged. To program the chip we put it in an Arduino UNO and uploaded our code through an Arduino IDE. When the programming was finished we moved the chip back to our circuit for proper use.

Figure 4 details the inputs to and outputs from the microcontroller. The specifics of each device are explained elsewhere, but their software interpretation and handling is described below. Recall that to correctly read the keypad half the pins are inputs and half are outputs. The RFID lines are a Serial In and a Serial Out (RFID perspective). They UI is the 3 buttons and 3 switches and the proximity sensor is a single wire. The only true output of the microcontroller is the servo PWM described in 2.3.

The software has two components. The first is the device interfaces that help read from each of the four device blocks and the second is the open / close logic that decides to turn the lock or not.

### **2.5.2 Device Interfaces**

The device interfaces help the open/close abstract the devices into a small number of easily managed bits. They also help correctly run each device so operation is smooth and consistent.

The Keypad interface is the logic that actually uses the input method described in 2.3.2. After comparing the queue of recently pushed buttons to the correct PIN, the interface outputs a single bit: Keypad\_Correct. This bit is passed to the open close logic. The RFID interface is very similar: if the read serial number matches the stored number then the RFID\_correct bit becomes high/TRUE.

The raw data from the User Interface doesn’t need much of an interface except the signals are inverted from the physical active-low to active-high in the software. Only 1 button and the 3 switches are used for the open/close logic. The proximity sensor interface adds an additional signal to detect change. The logic looks at both the current and previous state to tell if, say, the door just closed.

### **2.5.3 Open / Close Logic**

To decide to lock or unlock the door, the logic looks only at RFID\_correct, Keypad\_correct and the 5 inverted UI / 2 proximity sensor bits. These 8 values are each either high or low which makes the logic very easy.

1 Keypad is correct and Keypad is enabled and Mode is OR Open

2 RFID is correct and RFID is enabled and Mode is OR Open

3 Keypad is correct and Keypad is enabled and Mode is And

and RFID is correct and RFID is enabled Open

4 Open/close button pressed and lock is closed Open

5 Open/close button pressed and lock is open Close

6 Door just closed Close

The output of this system is the PWM to the servo. The ATMega has built-in square wave capabilities but the frequencies available didn’t match the servo requirements. Instead we used a digital output and coded our own PWM generator using the internal clock to help time it. The signal details are explained in 2.4.

# **3. Design Verification**

## **3.1 Power Supply**

The power supply design changed dramatically since the design review. As such, many of the requirements and verifications are no longer valid. In this section, we will explain either how each requirement was met and how it was verified. If the requirement was unable to be met due to a design change, we will explain the how each changed section was meant to work, and how we verified that it was working correctly.

Our requirements and verification from our design review for the power supply is listed at Table 10 of Appendix 1. The wall power supply section was created based on the assumption that we would need to create a wall adapter ourselves. As this is not the case, the only need that we have for this section was that we would be able to get at least 10 V from our power supply and that it could support 0.7 A of current being run through it. We verified this by simply measuring the voltage produced by the power supply and using resistors to simulate a load that draws 0.7 A of current. Both tests succeeded, and as such, this component’s requirements were verified.

The next section of of the power requirements was for the battery charger. This requirement has changed only due to the voltage regulator that we were working with. The meaning behind this requirement was to ensure that the wall input had a high enough voltage to charge the batteries. This essentially meant that the wall input had to be greater than the voltage across the batteries. In order to verify this section we will use the meaning behind having the requirement in order to show that the requirement was fulfilled.

We originally planned to use a low dropout voltage regulator. However, after we learned that there were other regulators available to us, we swapped that out. This new regulator required at least 7 V as an input to work correctly. Since we were using 1.2 V batteries, we needed 6 to meet this requirement. Another change here is that the chip that we decided to use requires 1.65\*(number of batteries) V as a wall input in order to be used. This means that the wall input must be at least 10 V. This was confirmed earlier to be the wall input voltage.

The final requirements were for the battery. These requirements have not changed since these were aimed towards the battery being able to power the circuit. The first requirement was that the batteries be able to apply at least 4.8 V to the circuit. Using a 5 V regulator, we regulated the voltage to 4.96 V, which is above 4.8 V. The last requirement involves inserting the batteries backwards. This requirement is verified, but in a fairly unique way. The battery casing is so tight, that the negative side will be unable to touch the positive end of the holder. As such, the battery pack would be considered an open circuit, and this requirement is verified.

## **3.2 Outside Interface Verification**

The two key parameters of the RFID module are detection distance and power consumption. We want the RFID module to be in idle more for as long as possible because the idle current is so low. We wanted the RFID to scan tags that were up to 3 inches away. In reality, our effective distance was much less. Table 11 (R&V) details the process and Table 6 details our testing results, which indicate the tag didn’t consistently (as in 100% scan rate) scan until the tag was about 0.75” from the module. The distance is acceptable for functionality, however, we don’t have as much room for user error as we hoped.

Our keypad verification (Table 12) was just confirmation that the device worked the way we expected. I powered each row with 5 V one at a time and measured from all four columns simultaneously. In the table below, the Row column indicates which row was powered. The other columns are the keypad columns that were powered when the button indicated in the first row was pressed. A “high” signal here means > 4.9 V. The unsurprising results are in Table 7.

## **3.3 Inside Interface Verification**

Initially, the datasheets made us think the proximity sensor worked by sending a pulse whenever the state (read: distance) changed. When we received the parts it was actually the way described in 2.4.1 which was better for us long-term. During our testing (Table 14) we got the results in Table 8.

Because the user interface is just buttons and switches, verifying the unit (Table 13) is pretty straightforward. Our final design did not use edge-detectors, nor did the layout of the buttons have much bearing on where we put them. All the hardware testing was done on a small breadboard circuit with all 6 items spread out. It was easy to see button and switch was working.

## **3.4 Microcontroller Verification**

The microcontroller ended up being complicated software in an easy hardware PCB. Appendix C is the final code loaded on the microcontroller. Initially we had a lot of of the logic in hardware (Table 16 would have been the verification) but that logic was all easily moved into software. Now Table 15 and Table 16 are both verification of different aspects of the microcontroller.

To make testing simple, we unit-tested as much as possible. The code was split up into several “modules” that each could be tested independently but all worked together for the final product. Those are: RFID\_interface, keypad\_interface, ui\_interface, servo\_interface and the open/close\_logic.

Testing each module was a matter hooking up the correct hardware and trying every iteration of inputs to make sure the software could handle anything the user tried. For the keypad and RFID modules we entered different PINs and tags to see if the modules correctly marked the “correct” flags when the inputs were correct. The UI module had to simply read the state of the prox sensor and the 3 buttons and switches. Finally, that data was all sent to the lock/unlock logic described in 2.5.3 which in the final demo locked and unlocked the door correctly in every circumstance we tried.

# **4. Costs**

## **4.1 Parts**

Table 9 includes the costs to purchase all of the parts for this product including bulk costs, and the cost that we paid for the prototype. The lock that we used was only used for testing and demonstration purposes, and is not needed in order to reproduce our design. As such, we are listing it as only part of our actual costs, not the retail or bulk costs. Our total retail cost comes to $141.60, our total bulk cost comes to $112.52, and the total amount we spent on this project was $107.00

## **4.2 Labor**

Using an estimate of $35/hour, Equation 7 is our estimate for the cost to create the electronic sections of our circuit across all three partners and Equation 8 is our estimate to create the mechanical parts of our circuit. We estimated that each group member worked on the project for 10 hours a week for 3/5ths of the semester. For the machine shop, we estimated that it would take 10 hours to complete this project. Overall, this comes to a labor cost of $26075.

|  |  |
| --- | --- |
| 3 \* $35/hour \* 10 hours/week \* 16 weeks/semester \* 0.6 semesters \* 2.5 = $25200 | (7) |

|  |  |
| --- | --- |
| $35/hour \* 10 hours \* 2.5 = $875 | (8) |

# 

# 

# **5. Conclusion**

## **5.1 Accomplishments**

We were ultimately able to fulfill all of the high level requirements that we gave to ourselves. Two additional modes of entry were added to the door, our power efficiency allows the circuit to remain powered for a little over 2 days before losing power and the lock will automatically lock when the door closes. In terms of our requirements, we found that 18 out of 19 requirements have been verified. For the one that was not verified, we have explained the reasons why this is the case in the design verification section.

## **5.2 Difficulties**

A major difficulty that we ran into when combining the power circuit with all of the other components is that the voltage out of the power circuit would drop up to 2 V when current was drawn from it and only the batteries were attached. We discovered that was due to the current drawn being pulled through the sense resistor (R11 in Figure 2), thus reducing the difference between the battery voltage and the circuit ground. We solved this error by changing the overall ground from the wall ground to the battery ground. This made the voltage drop not matter and allowed us to power the circuit in all circumstances.

## **5.3 Ethical considerations**

Our largest ethical consideration remains the protection of individuals and their property. Users are trusting us and it is our responsibility to ensure the system cannot be bypassed in any way. We only send raw data through the exposed wires, not any sort of critical signal that can be spoofed. We reset variables and states after a short time. We include a proximity sensor to close the door automatically when it shuts. We give the users options so they aren’t forced to use devices they don’t trust. In all of our testing so far we have never been able to open the door maliciously, but much more testing would be required before this design became a shippable product.

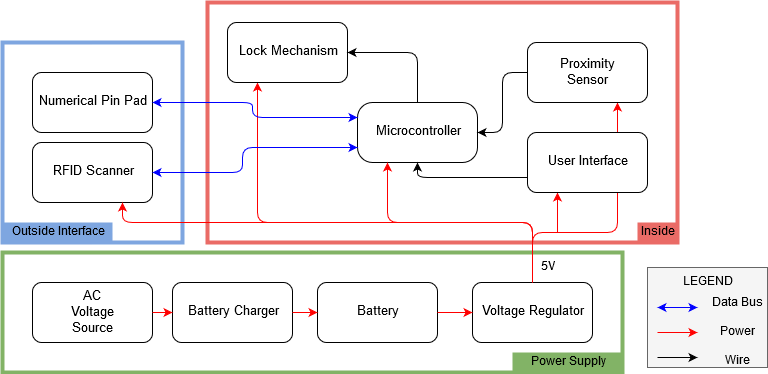
## **5.4 Future work**

Having demonstrated a working prototype of our design, we can consider future changes / additions to the system. First, we would like to simplify our design by combining our two PCB’s into one and putting the buttons/switches directly on the board. This would simplify the wiring inside the box and reduce the size of the entire box to make it easier and less intrusive for users.

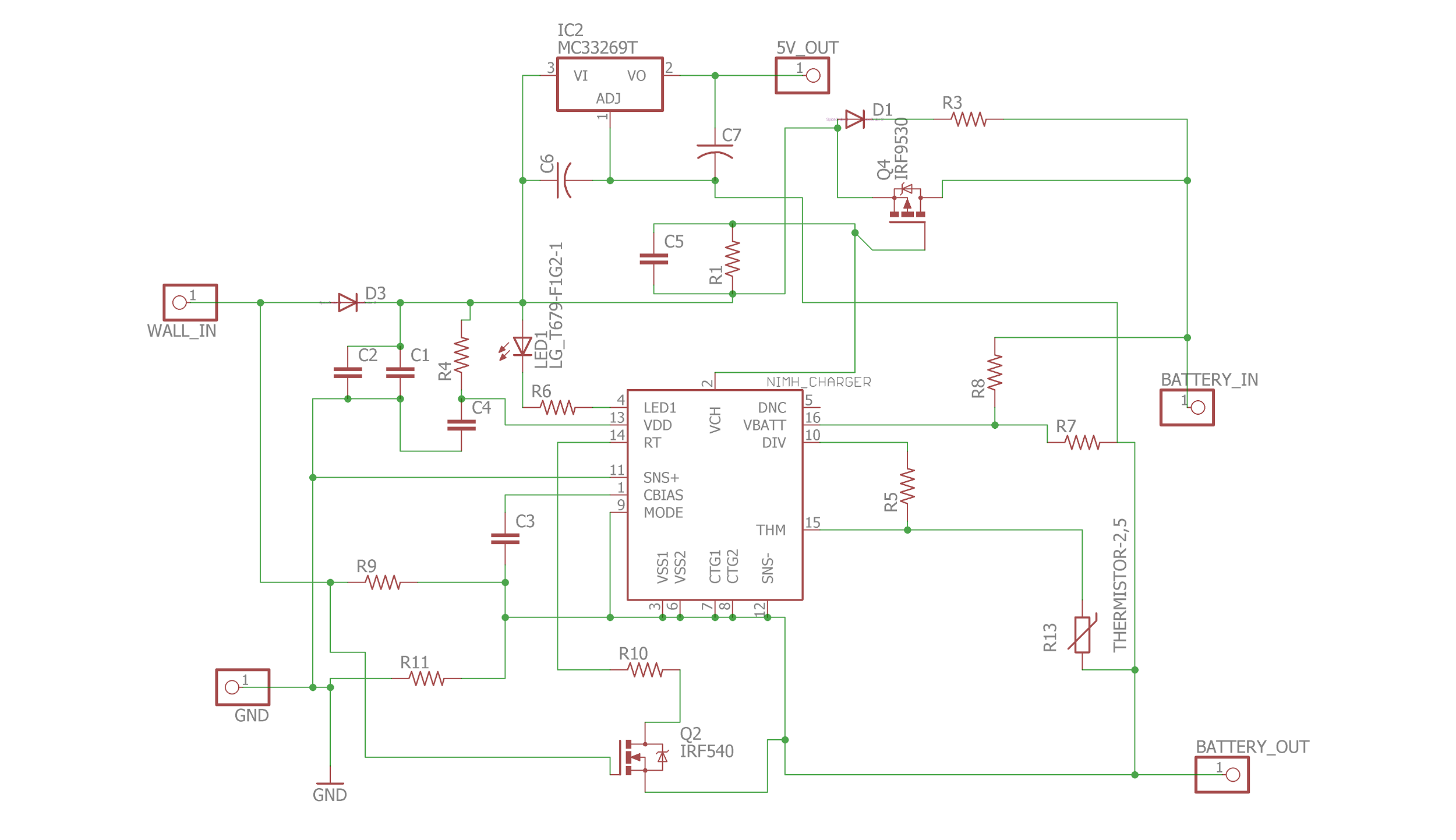
One large change would be physically separating the outside and inside blocks entirely. Using BlueTooth instead of wires connecting the two means easier application on more types of doors. However, then the outside needs a separate power unit (which may run out).

Finally, having demonstrated a keypad and RFID sensor working, it’s easy to imagine additional methods of entry like fingerprints, card swiping. It is worth noting that because our design is supposed to be an addon to a door as opposed to a replacement so one could really only use a device or two in a particular design at a time.

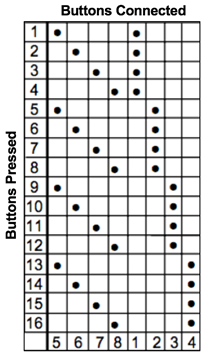
# **Figures and Tables**

**Figure 1: Block Diagram**

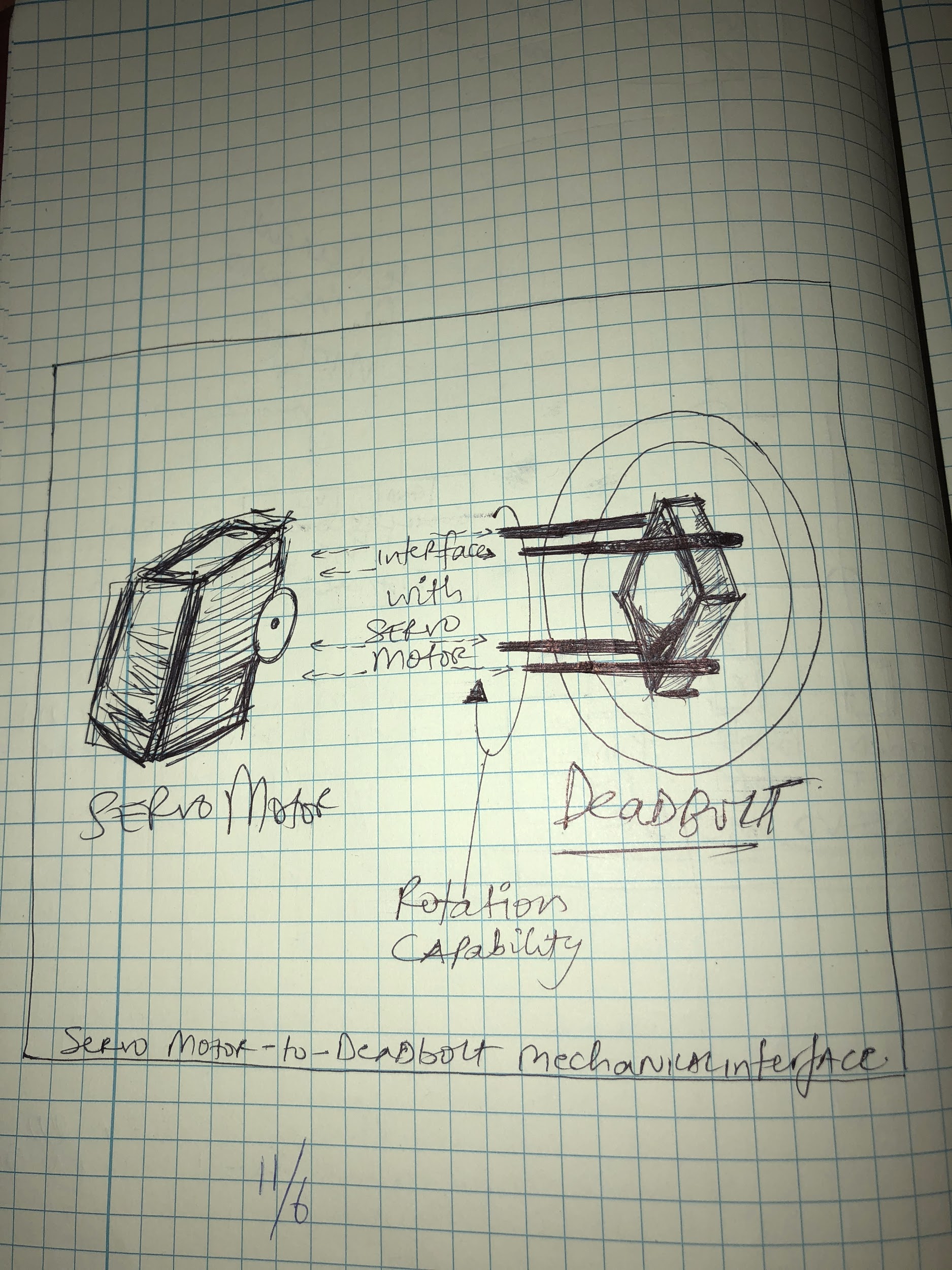
**Figure 2: Power Circuit Schematic**



**Figure 3: Keypad Pin/Button Mappings**



**Figure 4:Servo Motor Deadbolt Interface**



**Table 1: Current Draw from Wall or Battery Alone**

|  |  |  |
| --- | --- | --- |
| **Test Number** | **Circuit Conditions** | **Current Drawn** |
| 1 | Idle | 57 mA |
| 2 | Idle | 56.5 mA |
| 3 | RFID | 280 mA |
| 4 | RFID | 281 mA |
| 5 | Servo | 570 mA |
| 6 | Servo | 575 mA |
| 7 | Both | 740 mA |
| 8 | Both | 744 mA |

**Table 2: Current draw from the batteries when the wall is plugged in.**

|  |  |  |
| --- | --- | --- |
| **Test Number** | **Circuit Conditions** | **Current Drawn** |
| 1 | Idle | -1.7 mA |
| 2 | Idle | -1.7 mA |
| 3 | RFID | 67 mA |
| 4 | RFID | 68 mA |
| 5 | Servo | 300 mA |
| 6 | Servo | 305 mA |
| 7 | Both | 475 mA |
| 8 | Both | 480 mA |

**Table 3: User Interface Descriptions**

|  |  |
| --- | --- |
| Name | Description |
| Open/Close Button | Lock or unlock the lock |
| Reset RFID Button | Change the valid RFID Tag |
| Reset Keypad Button | Change the valid PIN |
| Enable RFID Switch | Enable usage of RFID scanner |
| Enable Keypad Switch | Enable usage of Keypad |
| AND/OR Mode Switch | Require both RFID AND Keypad |

**Table 4: Microcontroller Input / Output Table**

|  |  |  |  |
| --- | --- | --- | --- |
| Input Name | Input Size | Output Name | Output Size |
| RFID | 2 | Servo PWM | 1 |
| Keypad | 4 | Keypad | 4 |
| Proximity Sensor | 1 |  |  |
| User Interface | 6 |  |  |

**Table 5: RFID Current**

|  |  |  |
| --- | --- | --- |
|  | Expected Current (mA) | Actual Current (mA) |
| Idle | 10 | 9.6 |
| Active | 100-200 | 150 |

**Table 6: RFID Scanning Distance**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Distance (in.) | 3.0 | 2.5 | 2.0 | 1.5 | 1.0 | 0.5 | 0.75 |
| Scanned? | no | no | no | no | no | yes | yes |

**Table 7: Keypad Verification**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Row | 1 | 2 | 3 | A | 4 | 5 | 6 | B | 7 | 8 | 9 | C | \* | 0 | # | D |
| 1 | 1 | 2 | 3 | 4 | - | - | - | - | - | - | - | - | - | - | - | - |
| 2 | - | - | - | - | 1 | 2 | 3 | 4 | - | - | - | - | - | - | - | - |
| 3 | - | - | - | - | - | - | - | - | 1 | 2 | 3 | 4 | - | - | - | - |
| 4 | - | - | - | - | - | - | - | - | - | - | - | - | 1 | 2 | 3 | 4 |

**Table 8: Proximity Sensor Testing**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Input (V) | Distance | Outputs (V) | | | |
| 5 | Close | 0.008 | 0.006 | 0.0065 | 0.009 |
| 5 | Far | 5.0 | 4.99 | 5.0 | 5.0 |
| 4 | Close | 0.004 | 0.006 | 0.0055 | 0.0045 |
| 4 | Far | 4.0 | 4.0 | 4.0 | 4.0 |

**Table 9: Parts Costs**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Part Name** | **Part #** | **Manufacturer** | **Retail Cost ($)** | **Bulk Cost ($)** | **Actual Cost ($)** |
| Battery Charging Chip | DS2715 | Maxim Integrated | 8.41 | 3.67 | 16.00 |
| N-Mos | IRF510 | Vishay/Siliconix | 0.82 | 0.25 | Free |
| P-Mos | IRF9Z24 | Vishay/Siliconix | 1.47 | 0.53 | Free |
| Microcontroller | ATMega328p | Atmel | 2.11 | 1.53 | 7.87 |
| Various Resistor / Capacitors / Wires | n/a | Various | 4.00 | 4.00 | Free |
| Servo | 31311S HS-311 | Hitec RCD Inc. | 13.21 | 13.21 | 13.21 |
| Wall Mount | n/a | Various | 10.00 | 10.00 | 10.00 |
| 12V Wall Adapter | SWI5-12-N-P5 | CUI Inc. | 6.90 | 4.83 | 12.60 |
| Proximity Sensor | AS201801 / MP201802 | ZF Electronics | 8.81 | 5.46 | 12.81 |
| 6AA Battery Case | BH-6AA-WIRE | LEDSupply | 0.53 | 0.42 | 1.53 |
| 2800mAh AA Ni-MH Rechargeable Batteries | n/a | EBL | 10.50 | 10.50 | 13.99 |
| 16MHz Crystal Oscillator | ECS-100AX-160 | ECS Inc. | 2.37 | 1.43 | Free |
| Keypad | 96BB2-006-R | Grayhill Inc. | 19.92 | 12.99 | Free |
| RFID Scanner | 28440 | Parallax | 49.99 | 44.99 | Free |
| UNO R3Board and Atmega328 | N/A | Elgoo UNO | 10.90 | 10.90 | 10.90 |
| Deadbolt Lock | 8111309 | Honeywell | n/a | n/a | 12.09 |
| **Total** |  |  | **141.60** | **112.52** | **107.00** |

# 

# **References**

[1] - University of Illinois Police Department, ‘Daily Crime Log’,

(2017,September 20). [Online] Available at:

https://illinois.edu/blog/files/7512/557491/118800.pdf [Accessed: 21-September-2017]

[2] - *NiMH Battery Pack Charge Controller*, datasheet, Maxim Integrated, 2009. Available at:

https://pdfserv.maximintegrated.com/en/ds/DS2715.pdf

# 

# 

# **Appendix A Requirement and Verification Tables**

**Table 10: Power Supply R&V**

|  |  |  |  |
| --- | --- | --- | --- |
| Component | Requirement(s) | Corresponding Verification Method(s) | Verification  (Y/N) |
| Wall Power Supply | 1] A function generator supplying 220 V AC powering the circuit must be able to output 9 V DC from the terminals of the voltage regulator (minus the LED voltage drop). | 1] Use a function generator to apply the 220 V AC circuit across the power supply, and measure the voltage across the voltage regulator to ensure it is at least 8.3 V DC (0.7 V is the estimated voltage drop from the LED). | Y |
| Battery Charger | 1] Voltage applied across the entire combination of batteries is greater than 6 V when plugged into the wall outlet. | 1] Use a voltmeter to estimate the voltage across the batteries when the battery charger is plugged into the wall. The reading must be 6 V or more. | Y |
| Battery | 1] Should be able to maintain a voltage of at least 4.8 V to the circuitry inside and outside the door  2] Should be able to suppress any errors from a battery inserted backwards. These batteries should have no impact on the system | 1] Use voltmeter when battery is being drained and measure potential difference across ends. Reading must be 4.8 V or more.  2] Insert a battery backwards and use an ammeter to test current flowing from the batteries to the input pin of the voltage regulator. Ensure that the ammeter reading is 0 A in this situation. | Y  Y |

**Table 11: RFID Scanner R&V**

|  |  |  |
| --- | --- | --- |
| Requirement(s) | Corresponding Verification Method(s) | Verification  (Y/N) |
| 1] Scan RFID tags at a maximum range of 2.5-3 inches  2] Go into a low-power mode when not in use to conserve power (less than 0.05 W). Active power should be between 0.5 W and 1.0 W | 1a] Set up an test arduino to read the tag data. The circuit must be able to read 4 bytes of data from the tag.  1b] Start with tags far from the scanner.  1c] Move a tag 3 inches from the scanner. This might scan.  1d] Move the tag 2 inches from the scanner. This and anything closer should definitely scan.  2a] Attach the scanner to an oscilloscope in order to measure power consumption  2b] Record the power consumption when no tags are in range. This is the idle power, compare it to the expected value of < 0.05 W  2c] Scan a tag multiple times and record the power consumption. Verify the active power spikes against the expected 0.5 - 1.0 W | N (Actual: 0.75)  Y |

**Table 12: Numerical Keypad R&V**

|  |  |  |
| --- | --- | --- |
| Requirement(s) | Corresponding Verification Method(s) | Verification  (Y/N) |
| 1] Must be able to output high V signal indicating which button was just pressed.  2] Must be easy to push the buttons | 1a] Hook up the keypad to an oscilloscope or multimeter  1b] Power the row pins with 5 V  1c] Push every button and verify the correct column pins have between 4.5 - 5.5 V  2] Ensure the buttons are easy to press without strain or error | Y  Y |

**Table 13: User Interface R&V**

|  |  |  |
| --- | --- | --- |
| Requirement(s) | Corresponding Verification Method(s) | Verification  (Y/N) |
| 1] Every switch and button should alternate between low (0 - 0.7 V when off or unpressed ) and high (4.5 - 5.5 V when on or pressed)  2] Buttons passing through the edge-detector should create a pulse as long as the clock cycle.  3] Must be intuitive for the user and properly labeled. | 1a] Test each switch and button individually  1b] Start the item in its low state (unpressed or off). Verify the output voltage is between 0 V and 0.7 V.  1c] Change the item to its high state (pressed or on). Verify the output voltage is between 4.5 V and 5.5 V.  2a] Build a button in series with an edge detector (circuit no longer present in final product). Use an oscilloscope to measure the output of the edge detector.  2b] Use a function generator to make a square wave clk signal.  2c] Push the button down (don’t let up). The output should be a single pulse with the same width as the clk signal.  2d] Test with many clk signal widths  3] Show the UI to people not involved in the project and ask what each part does. If the majority answers incorrectly, we need to redesign that part. | Y  Y  Y |

**Table 14: Magnetic Proximity Sensor R&V**

|  |  |  |
| --- | --- | --- |
| Requirement(s) | Corresponding Verification Method(s) | Verification  (Y/N) |
| 1] The sensor outputs a short 0.5 ms pulse every time the two parts separate or meet | 1a] Start with the magnet far from the sensor.  1b] The sensor should output between 4.5 - 5.5 V  1c] Move the magnet close  1d] Each pulse length should be 0.4 - 0.6 ms long | Y |

**Table 15: Controller R&V**

|  |  |  |
| --- | --- | --- |
| Requirement(s) | Corresponding Verification Method(s) | Verification  (Y/N) |
| 1] Chip outputs low signal (0 - 0.8 V) when no or incorrect output is given  2] Chip outputs high signal (4.5 - 5.5 V) when a correct input is given | 1a] Set up a test circuit with saved values for PIN and RFID numbers that are correct  1b] Input incorrect PIN and scan the wrong tag. Output should be low (0 - 0.8 V)  2a] Input correct PIN. Output should be high (4.5 - 5.5 V)  2b] Scan correct tag. Output should be high (4.5 - 5.5 V) | Y  Y |

**Table 16: State Machine R&V**

|  |  |  |
| --- | --- | --- |
| Requirement(s) | Corresponding Verification Method(s) | Verification  (Y/N) |
| 1] RFID and Keypad Inputs output a 2.7 - 5.5 V signal when they the corresponding switches are turned on and between -0.5 - 0.8 V when an incorrect combination is input.  2] The Unlock/Lock state must be between 2.7-5.5 V when the bit is high and -0.5 - 0.8 V when the bit is low.  3] Use proximity sensor to lock the door | 1a] Set up the circuit as shown in Figure 5 of the design review (no longer present in final product).  1b] Apply 5 V inputs in every combination shown in Tables 1 and 2.  1c] Ensure that between 2.7-5.5 V are output for combinations where a 1 appears in Tables 1 and 2 and that between -0.5-0.8 V are output where there is a 0.  2a] Set up a circuit as shown in Figure 6 of the design review (no longer present in final product).  2b] Test every combination of signals in Table 3 of the design review (no longer present in final product). Ensure that the end voltage is 2.7 - 5.5 V where a 1 shows up in the table and between -0.5 - 0.8 V where a 0 shows up on the table.  3a] set the circuit created in 2a to a state where having the proximity sensor input at 0 V puts the state to unlocked, and changing that same input to 5 V would put the state to locked without changing any other inputs.  3b] Ensure that the state changes from -0.5 - 0.8 V output to 2.7 - 5.5 V output. | Y  Y  Y |

**Table 17: Lock Mechanism R&V**

|  |  |  |  |
| --- | --- | --- | --- |
| Component | Requirement(s) | Corresponding Verification Method(s) | Verification  (Y/N) |
| Lock Mechanism, associated gear train, and servo motor | 1] Motor is capable of moving 90 degrees which would need a range of PWM from 575 to 1517 μseconds time period to execute the rotation. Motor must move to the correct position from PWM inputs at a voltage of 5 V.  2] Power is conserved when the motor is not in motion. In our case, the power consumed by the servo motor when not in use is at most 0.0375 W. | 1] Using an oscilloscope, measure the PWM. At Neutral (0 degrees) should be 575 μseconds dc. Locked (90 degrees) should have a 1517 μseconds dc.  2] Use a voltmeter and an ammeter, measure the voltage across the 2 power ports of the servo motor and the current flowing into the servo. | Y  Y |

# **Appendix B Resistances and Capacitances**

**Table 18: Figure 2 Resistor / Capacitor values**

|  |  |  |  |
| --- | --- | --- | --- |
| **Resistor** | **Value** | **Capacitor** | **Value** |
| R1 | 830 Ω | C1 | 1 µF |
| R3 | 220 Ω | C2 | 33 µF |
| R4 | 150 Ω | C3 | 0.1 µF |
| R5 | 10 KΩ | C4 | 0.1 µF |
| R6 | 780 Ω | C5 | 4.7 µF |
| R7 | 100 KΩ | C6 | 0.33 µF |
| R8 | 500 KΩ | C7 | 0.1 µF |
| R9 | 10 KΩ |  |  |
| R10 | 160 KΩ |  |  |
| R11 | 3 Ω |  |  |

# **Appendix C ATMega328p Code**

#include <cQueue.h>

#include <Keypad.h>

#include <SoftwareSerial.h>

#include <time.h>

// TODO

// ############## KEYPAD CONSTANTS #################### //

const byte ROWS = 4;

const byte COLS = 4;

//define the symbols on the buttons of the keypads

byte rowPins[ROWS] = {7,6,5,4};

byte colPins[COLS] = {10,11,12};

char Keys[ROWS][COLS] =

{

{'1','2','3','A'},

{'4','5','6','B'},

{'7','8','9','C'},

{'\*','0','#','D'}

};

Keypad customKeypad = Keypad(makeKeymap(Keys),rowPins,colPins,ROWS,COLS);

Queue\_t qe;

Queue\_t\* q = &qe;

char in[4] = {'X','X','X','X'};

// #################### RFID CONSTANTS ################# //

#define rxPin 9 // Serial input (connects to the RFID's SOUT pin)

#define txPin 8 // Serial output (connects to the RFID's SIN pin)

#define BUFSIZE 12 // Size of receive buffer (in bytes) for incoming data from the RFID R/W Module (this should be adjusted to be larger than the expected response)

// RFID R/W Module Commands

#define RFID\_Read 0x01 // Read data from specified address, valid locations 1 to 33 (5)

#define RFID\_Write 0x02 // Write data to specified address, valid locations 3 to 31 (1)

#define ERR\_OK 0x01 // No errors

// For use with RFID\_ReadLegacy command

#define LEGACY\_StartByte 0x0A

#define LEGACY\_StopByte 0x0D

// set up a new serial port

SoftwareSerial rfidSerial = SoftwareSerial(rxPin, txPin);

// ###################################################### //

char pin[4] = {'1','2','3','4'};

uint8\_t correct\_tag[4] = {0x02, 0xc1, 0xed, 0x86};

// uint8\_t tag\_2[4] = {0x02, 0xc1, 0xec, 0x77};

unsigned long led\_time = 0;

unsigned long time\_k = 0; // Time to reset the keypad being correct

unsigned long time\_r = 0; // Time to reset the rfid being correct

unsigned long time\_q = 0; // Time to reset the queue

int rfid\_correct = 0; // If the correct rfid tag was detected

int key\_correct = 0; // If the correct pin was entered

unsigned long last = 0;

unsigned long micro = 0;

// ############################################################### //

#define s\_both A0 // AND mode button

#define s\_r A1 // RFID on

#define s\_k A2 // keypad on

#define prox\_pin 2

#define SERVO\_PIN 3

// uo kr rr

int button\_pins[3] = {A3, A4, A5};

int switch\_pins[3] = {s\_both, s\_r, s\_k};

byte buttons[3] = {LOW, LOW, LOW};

byte switches[3] = {LOW, LOW, LOW};

byte prox = LOW;

byte oldprox = LOW;

byte unlock\_s = LOW;

// Reset data

byte reset\_rfid = LOW;

unsigned long rr\_time;

byte reset\_keypad = LOW;

byte rk\_count = 0;

unsigned long rk\_time = 0;

byte use\_rfid = LOW;

unsigned long ur\_time = 0;

#define STATUS\_UNLOCKED 0

#define STATUS\_LOCKED 1

byte status\_l = 1;

byte or\_last = 0;

unsigned int P = 2857; // in microseconds

// ############################### MAIN FUNCTIONS ################################ //

void setup()

{

Serial.begin(9600);

setup\_keypad();

setup\_rfid();

setup\_ui();

pinMode(LED\_BUILTIN, OUTPUT);

}

void loop()

{

if(use\_rfid == HIGH)

handle\_rfid();

handle\_keypad();

read\_ui();

check\_unlock();

update\_status();

//while(1)

//servo\_action(1);

}

// ############################## HELPER FUNCTIONS ########################### //

void check\_unlock()

{

// inputs

//rfid\_open : rfid\_correct

//keypad\_open : key\_correct

//rfid\_switch : switches[1]

//keypad\_switch : switches[2]

// output : unlock

//unlock = LOW;

if(switches[0] == LOW) // OR state

{

if( (switches[2] && key\_correct) || (switches[1] && rfid\_correct) ){

unlock();

Serial.println("UNLOCK 1");

}

}else{ // AND state

if( key\_correct && !switches[1] && switches[2])

{

unlock();

Serial.println("UNLOCK 2");

}

if(rfid\_correct && switches[1] && !switches[2])

{

unlock();

Serial.println("UNLOCK 3");

}

if(rfid\_correct && key\_correct && switches[1])

{

unlock();

Serial.println("UNLOCK 4");

}

}

//digitalWrite(LED\_BUILTIN, unlock);

}

void lock()

{

if(status\_l == STATUS\_LOCKED)

return;

status\_l = STATUS\_LOCKED;

key\_correct = 0; // if its unlocked, no need to store this.

rfid\_correct = 0;

digitalWrite(LED\_BUILTIN, HIGH);

delay(125);

digitalWrite(LED\_BUILTIN, LOW);

delay(125);

digitalWrite(LED\_BUILTIN, HIGH);

delay(125);

digitalWrite(LED\_BUILTIN, LOW);

delay(125);

servo\_action(0);

}

void unlock()

{

// check if already unlocked

if(status\_l == STATUS\_UNLOCKED)

return;

status\_l = STATUS\_UNLOCKED;

key\_correct = 0; // if its unlocked, no need to store this.

rfid\_correct = 0;

digitalWrite(LED\_BUILTIN, HIGH);

delay(250);

digitalWrite(LED\_BUILTIN, LOW);

servo\_action(1);

}

// Trigger timeouts of a lot of variables

void update\_status()

{

unsigned long temp = millis();

if(temp - rr\_time > 8000)

reset\_rfid = LOW;

if(temp - rk\_time > 8000)

reset\_keypad = LOW;

if(temp - time\_k > 8000)

key\_correct = 0;

if(temp - time\_r > 8000)

rfid\_correct = 0;

if(temp - time\_q > 4000)

clear\_kq();

if(temp - ur\_time > 9000)

use\_rfid = LOW;

}

void read\_ui()

{

for(int i = 0; i < 3; i++){

buttons[i] = !(digitalRead(button\_pins[i]));

switches[i] = digitalRead(switch\_pins[i]);

}

//buttons[3] = !(digitalRead(button\_pins[3]));

prox = !(digitalRead(prox\_pin));

// HANDLE UI

// (right now the checks are all independent. Can be made to have priority later)

// Reset RFID button pushed

if(reset\_rfid == LOW && buttons[2] == HIGH)

{

reset\_rfid = HIGH;

rr\_time = millis();

use\_rfid = HIGH; // should automatically turn on the rfid also

ur\_time = rr\_time;

}

// Reset keypad button pushed

if(reset\_keypad == LOW && buttons[1] == HIGH)

{

reset\_keypad = HIGH;

rk\_count = 4;

rk\_time = millis();

}

// Door closes: prox sensor was off is now on

if(oldprox == LOW && prox == HIGH)

lock();

if(oldprox == HIGH && prox == LOW)

unlock();

oldprox = prox;

// User override: lock/unlock button

if(buttons[0] == HIGH && or\_last == LOW)

{

if(status\_l == STATUS\_LOCKED)

unlock();

else if(status\_l == STATUS\_UNLOCKED)

lock();

}

or\_last = buttons[0];

print\_ui(); // DEBUG

}

// For debugging

void print\_ui()

{

Serial.print(" ");

for(int i = 0; i < 4; i++)

Serial.print(buttons[i]);

Serial.print(" ");

for(int i = 0; i < 3; i++)

Serial.print(switches[i]);

Serial.print(" ");

Serial.print(prox);

Serial.print(" RR:");

Serial.print(reset\_rfid);

Serial.print(" k:");

Serial.print(key\_correct);

Serial.print(" r:");

Serial.print(rfid\_correct);

Serial.print(" s:");

Serial.print(status\_l);

Serial.print(" ");

}

void update\_time()

{

micro = millis();

Serial.println(micro - last);

last = micro;

}

void servo\_action( int action)

{

//if action = 0, servo is commanded to lock

// if action = 1, servo is commanded to unlock

float DC = 0; //setting duty cycle

float High\_time = 0; // time for which we supply 5V part of the square wave

float Low\_time = 0; // time for which we supply 0V part of the square wave

if (action)

DC = 0.60; // unlock position

else

DC = 0.25; // lock position

High\_time = P\*DC;

Low\_time = P - High\_time;

for(int i = 0; i < 450; i++) // 450

{

digitalWrite(SERVO\_PIN, HIGH);

delayMicroseconds((int)High\_time);

digitalWrite(SERVO\_PIN, LOW); //SERVO\_PIN

delayMicroseconds((int)Low\_time); // 1429

}

digitalWrite(3, LOW); // after servo is done moving, we shut off the PWM

}

// ######################################### SETUP CODE ################################## //

void setup\_keypad()

{

q\_init(q, sizeof(char), 4, FIFO, true);

clear\_kq();

Serial.println("Keypad Ready");

}

void setup\_rfid()

{

// define pin modes

pinMode(rxPin, INPUT);

pinMode(txPin, OUTPUT);

// setup Arduino Serial Monitor

//Serial.begin(9600);

while (!Serial); // wait until ready

// set the baud rate for the SoftwareSerial port

rfidSerial.begin(9600);

Serial.println("RFID Ready");

Serial.flush(); // wait for all bytes to be transmitted to the Serial Monitor

}

void setup\_ui()

{

for(int i = 0; i < 3; i++){

pinMode(button\_pins[i], INPUT);

pinMode(switch\_pins[i], INPUT);

}

}

// ######################################## KEYPAD CODE ################################### //

void handle\_keypad()

{

char key = customKeypad.getKey();

if(key == '\*')

{

use\_rfid = HIGH;

ur\_time = millis();

}

else if(key!=NO\_KEY)

{

q\_push(q, (void\*)& key);

time\_q = millis();

read\_queue(in, q);

//print\_kq();

if(reset\_keypad == HIGH)

{

if(rk\_count > 0)

rk\_count--;

if(rk\_count == 0)

{

reset\_keypad = LOW;

clear\_kq();

for(int i = 0; i < 4; i++)

pin[i] = in[i];

}

}

else{

check\_keypad(in);

}

}

print\_kq();

}

void print\_kq()

{

Serial.print("\nRK:");

Serial.print(reset\_keypad);

read\_queue(in, q);

Serial.print(" Q: ");

Serial.print(in[0]);

Serial.print(in[1]);

Serial.print(in[2]);

Serial.print(in[3]);

Serial.print(" P: ");

Serial.print(pin[0]);

Serial.print(pin[1]);

Serial.print(pin[2]);

Serial.print(pin[3]);

}

void check\_keypad(char\* buf)

{

int i = 0;

for(i = 0; i < 4; i++)

{

if(buf[i] != pin[i])

break;

}

// Correct PIN was entered

if(i == 4)

{

time\_k = millis();

key\_correct = HIGH;

char a = 'X';

q\_push(q, (void\*)&a);

}

}

void read\_queue(char\* arr, Queue\_t\* q)

{

q\_pop( q, (void\*)&arr[0]);

q\_pop( q, (void\*)&arr[1]);

q\_pop( q, (void\*)&arr[2]);

q\_pop( q, (void\*)&arr[3]);

q\_push(q, (void\*)&arr[0]);

q\_push(q, (void\*)&arr[1]);

q\_push(q, (void\*)&arr[2]);

q\_push(q, (void\*)&arr[3]);

}

void clear\_kq()

{

char a = 'X';

q\_push(q, (void\*)&a);

q\_push(q, (void\*)&a);

q\_push(q, (void\*)&a);

q\_push(q, (void\*)&a);

}

// ####################################### RFID CODE ################################## //

// From: Parallax RFID Read/Write: Basic Demonstration

// Author: Joe Grand [www.grandideastudio.com]

void handle\_rfid()

{

char idx;

uint8\_t rfidData[BUFSIZE]; // Buffer for incoming data

//while (rfidRead(rfidData, ADDR\_Serial) != 0);

if(rfidRead(rfidData, ADDR\_Serial) == 0)

{

PrintHex(rfidData, 4); // The rfidData string should now contain the tag's serial number, so display it on the Serial Monitor

//Serial.print("\n");

checkRFID(rfidData);

Serial.flush();

}

}

void checkRFID(uint8\_t\* data)

{

// is it time to reset?

if(reset\_rfid == HIGH)

{

// reset

for(int i = 0; i < 4; i++)

correct\_tag[i] = data[i];

reset\_rfid = LOW;

}

else

{

// regular handle

int i = 0;

for(i = 0; i < 4; i++)

{

if(data[i] != correct\_tag[i])

break;

}

// If read value is the correct one

if(i == 4){

rfid\_correct = HIGH;

time\_r = millis();

use\_rfid = LOW;

data[0] = 0;

}

}

}

char rfidRead(uint8\_t \*data, uint8\_t address)

{

char offset; // offset into buffer

rfidFlush(); // empties any buffered incoming serial data

rfidSerial.print("!RW"); // header

rfidSerial.write(RFID\_Read); // command

rfidSerial.write(address); // address

while (rfidSerial.available() != 5) // wait until data is received from RFID module

{

//\*

update\_status();

handle\_keypad();

read\_ui();

check\_unlock();

update\_status(); //\*/

}

if (rfidSerial.read() == ERR\_OK) // if our status byte is OK

{

for (offset = 0; offset < 4; offset++){

data[offset] = rfidSerial.read(); // get the remaining data

}

return 0; // return good

}

return -1; // return error

}

void rfidFlush() // Empties any buffered incoming serial data

{

while (rfidSerial.available() > 0)

rfidSerial.read();

}

void PrintHex(uint8\_t \*data, uint8\_t length) // prints 8-bit data in hex

{

char tmp[length\*2+1];

byte first ;

int j=0;

for (uint8\_t i=0; i<length; i++)

{

first = (data[i] >> 4) | 48;

if (first > 57) tmp[j] = first + (byte)39;

else tmp[j] = first ;

j++;

first = (data[i] & 0x0F) | 48;

if (first > 57) tmp[j] = first + (byte)39;

else tmp[j] = first;

j++;

}

tmp[length\*2] = 0;

Serial.print(tmp);

}