RFID Controlled Door Lock

Brayden Burns
Trinity University, bburns@trinity.edu

Daniel Dahlinger
Trinity University, ddahling@trinity.edu

Brian Guenther
Trinity University, bguenthe@trinity.edu

Trevor Johnson
Trinity University, tjohnso3@trinity.edu

Follow this and additional works at: https://digitalcommons.trinity.edu/engine_mechatronics

Part of the Engineering Commons

Repository Citation
Burns, Brayden; Dahlinger, Daniel; Guenther, Brian; and Johnson, Trevor, "RFID Controlled Door Lock" (2018). Mechatronics Final Projects. 9.
https://digitalcommons.trinity.edu/engine_mechatronics/9
RFID Controlled Door Lock

Group F: Brayden Burns, Daniel Dahlinger, Brian Guenther, and Trevor Johnson

ENGR: 4367-1
Table of Contents:

Design Summary 2
System Details 4
Design Evaluation 8
Partial Parts List 10
Lessons Learned 10
Appendix 10

Design Summary

This device is an automated door lock attachment for an existing deadbolt-locked door. The device takes an exterior RFID input and can both lock and unlock the door. The device also implements an exterior doorbell switch where a guest can turn on a buzzer, and an interior switch for the operator to unlock the door from the inside. The front interface of the device is pictured in Figure 1; the working mechanism to turn the lock is pictured in Figure 2, and the housing for the processing and wirings is pictured in Figure 3. The logic is implemented using a PIC16F88 and Arduino Nano. The code implemented on the PIC is located in Appendix A1.1; the code for the Arduino is in A1.2; and the wiring diagram that ties the entire design together is in A2.

Figure 1. Front of door and interface for operator
Figure 2. Interior door lock apparatus

Figure 3: Housing for processing. Contains all wirings, the solderable breadboard, batteries, the Arduino Nano, and the PIC16F88.
System Details

Referencing Fig. 4 above for a visual aid, the smart door is comprised of several main parts: the RFID reader, the microcontrollers (Arduino Nano and PIC16F88), DC Stepper Motor with the 3D printed attachment, the doorbell switch, the speaker for the doorbell, two LEDs standing as indicators, and a battery power supply.

The RFID reader shown in Fig. 9 scans and reads the two available RFID cards. This information is sent to the Arduino Nano found in Fig. 5. The Arduino then processes both according to whether or not the RFID reader is reading a card and whether or not the card being read is the correct or incorrect card. The Arduino has two digital outputs: one represents a card being read and another represents whether the card is correct or incorrect. Based on the RFID reader reading, the Arduino writes these two outputs as digital high (reading card; correct card) or digital low (not reading card; incorrect card) and both values are sent to the PIC.

The PIC, pictured in Fig. 6, then reads these values. If both outputs from the Arduino are high (reading the correct card), the pic then turns the motor, flashes the green LED, and sounds one buzzer tone. If one output is high and the other is low (reading the incorrect card), then the pic
flashes the red LED and sounds two successive buzzer tones. If both of the outputs from the Arduino are low, then the PIC does nothing and continues to wait for the other two scenarios.

The flowchart shown in Fig. 7 shows how this process works in more of a structural representation. The PIC and the Arduino codes, which are shown in the appendix, are written based on this flowchart. The physical electrical connections of the circuit between all of the components for the system are shown in the schematic which can also be found in the appendix. All of the electronics are stored in a box attached to the inside of the door as seen in Fig. 8. The 3D printed part is attached to the stepper motor using a set screw and the motor attachment combo is attached to the deadbolt as seen in Fig. 10.

Figure 5. Arduino Nano

Figure 6. PIC16F88
Figure 7. Programming flowchart for the codes of both the PIC and the Aruduino
Figure 8. Electronics Housing Compartment

Figure 9. MFRC-522 RFID Reader
Design Evaluation

The final prototype was successful in exhibiting functional components for each of the six required element categories. The red and green LEDs on the front control panel satisfied the requirement for an output display. The second functional element requirement was an audio output device, which was accomplished with the buzzer. The buzzer acted as a doorbell and as a response to an incorrect RFID reading. The project requirements also included a manual user input, and the Automated Door System included that in the form of the switches, one for the doorbell and one to manually lock/unlock the door. The RFID sensor accomplished the fourth functional element requirement of an automatic sensor, as the system would respond to the sensor picking up an RFID reading without any user action (other than holding up an RFID to the front panel). The fifth requirement was that the project include actuators, mechanisms, and hardware. These were primarily accomplished with the stepper motor and its rig to hold it in contact with the deadbolt switch. The stepper motor successfully met the requirements of an actuator and had requirements of its own that properly utilized mechanisms and hardware. The 3D printed part that connected the stepper motor to the deadbolt was effective and appropriate, and even utilized the use of a set screw, which was vital in preventing the 3D part from slipping from the motor shaft. The attachment used to keep the motor on the deadbolt switch was a clever use of sheet metal, as a
custom-fit bracket was constructed. The design was solid and reliable, as the mechanism allowed the motor to successfully turn the lock every time.

The final functional element requirement was logic, processing, and control, as well as any other element not included in the other categories. The code for the PIC and Arduino accomplished this, and this is demonstrated in its ability to use the oscillator to time specific outputs (LED brightness duration, duration of sound played). While the PIC code used mainly if/else logic to check the switches and input from the Arduino to control the stepper, LED’s, and speaker, the code also used the memory of the PIC to save the last state of the switch and lock. It was important to save the last state of the switch so the PIC would know when the switch was changed, and it was crucial to remember the state of the lock stored in a variable called state, with a HIGH representing locked and a LOW representing unlocked. The implementation of this can be seen in appendix A1.1. This state memory was necessary and important because we had two possible ways of unlocking or locking the door (RFID or switch), and implementing this allowed us to not have to use a limit switch or servo motor, which saved money and time. We made the decision to use the Arduino Nano for reading the RFID reader based off of a number of factors. The Arduino Nano cost $8.29 and a PIC that had serial capabilities and enough I/O would have cost about $2 with shipping and tax. However, it took us about 6 hours to set up the Arduino to read RFID cards, differentiate between them based off of their unique UID code, and send data to the PIC based on a correct or incorrect UID being read. This short time frame was because of the vast amount of resources and the incredibly well updated and written, publicly available library for interfacing the MFRC-522 with Arduino. There was very little to no information online relating to interfacing RFID readers with PIC microcontrollers, and there were certainly no libraries. Additionally, many other resources did not use PICBasic Pro, and instead used assembly or C, which further complicated trying to learn from examples. From seeing other students in the class struggle with serial communication, we estimate that it would have taken about 50 hours to get the PIC and RFID working correctly, and that is if we could even accomplish this based on the lack of resources. While our group really would have enjoyed both the challenge and learning experience of implementing the RFID reader directly to the PIC, we simply didn’t have enough time to do so, especially given our workload from other classes such as fluids lab, fluids, and partial differential equations.

Although all of the functional element requirements were met and exceeded in some cases, we did run into an issue when putting it all together. Because we did not shorten the length of our wires, cramming all of the electrical components into the box on the back was a little bit of a struggle, but it was possible. However, when it was all in the box, the red LED on the front did not operate consistently. We believe this is because of either a poor connection from soldering or a part of the circuit being incorrectly shorted due to the excess of wires.
### Partial Parts List

<table>
<thead>
<tr>
<th>Component</th>
<th>Price</th>
<th>Source</th>
<th>Model Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFID Reader</td>
<td>$8.29</td>
<td>Amazon</td>
<td>MFRC-522</td>
</tr>
<tr>
<td>PIC</td>
<td>Free</td>
<td>Shop</td>
<td>PIC16F88</td>
</tr>
<tr>
<td>Arduino</td>
<td>$7.98</td>
<td>Amazon</td>
<td>Nano</td>
</tr>
<tr>
<td>Stepper Motor</td>
<td>Free</td>
<td>Shop</td>
<td>Sanyo Denki (Unsure of exact part number)</td>
</tr>
<tr>
<td>H-Bridge</td>
<td>Free</td>
<td>Shop</td>
<td>SN75441one</td>
</tr>
</tbody>
</table>

### Lessons Learned

One of the biggest lessons we learned was to always start things early. Additionally we learned that it is important to test every element working together before moving on to finalization. We had the entire design working wired up on breadboard but we were using the power supplies from the Elenco digital trainer. We thought we could just wire up our 12V power supply no problem after everything was soldered, but when we attempted to do this the stepper motor could not pull enough current from the battery. We should have tested how much current the motor pulled, and how much power it used and then researched batteries that could provide these, but instead we just kind of assumed our 12V supply would be ok, because we had just been using the built in power supplies with no problem. The biggest lesson to be taken away from this is that even the small uncomplicated parts of a mechatronic design deserve a lot of attention from the beginning to the end of the design process. To solve this we ended up connecting 9V batteries in series and burning through them after a few dozen motor turns.

The biggest lesson we learned was that the finalization from breadboard to fully functioning and finalized design takes a significant amount of time and thought. From designing a motor mount and shaft attachment, to soldering and mounting components, this all took much much longer than we expected. However, the longest and most infuriating process was trouble shooting the soldering. We had a few bad connections that had to be soldered, and resoldered, and then still didn’t work consistently. While are design did work, it didn’t work consistently, unless some percussive maintenance was applied to the box containing the electronics.

Our advice to future students is to get your design working as early as possible, and start finalizing it as early as possible because things will go wrong. Additionally, plan everything way ahead of time. From the physical mechanical processes, to the circuit, to the way that everything will attach and be held together, plan everything in detail, in advance.
Appendix
A1 Code

A1.1 PIC16F88 Code

The main purpose of this code is to control the LED lights, stepper motor, and buzzer, based off of input from the switches, and Arduino. Additionally, this code also remembers the state of the lock, so the PIC always knows whether the door should be locked or unlocked.

```javascript
'Initializing Variables/IO

step_a var PORTA.2 'a,b,c,d are all sent to the H-bridge to control
step_b var PORTB.1 'the Stepper Motor
step_c var PORTB.2
step_d var PORTB.7
i var word 'variable used in for loops
OC var PORTB.5 'Open/Close switch on inside of door
OC_LS var PORTA.1 'Last state of Open/Close Switch
Green var PORTA.0 'Green LED for correct RFID
Red var PORTO.3 'Red Led for incorrect RFID
RFID var PORTB.0 'HIGH = Correct RFID, LOW = Incorrect
RFID_R var PORTB.3 'HIGH = RFID scanned, LOW = No RFID scanned
Speaker var PORTB.6
state var word 'Represents state of lock, HIGH = locked,
DoorBell var PORTB.4 'low = unlocked
B0 var byte

'Initializing Last switch state to high, the state of the lock to high, and the
'input from the arduino to low

OC_LS = 1
state = 1
RFID = 0
RFID_R = 0
```
The main loop reads the switches and inputs from the Arduino, and then uses this information to control LED's, the speaker, and calls subroutines to control the stepper motor.

main:

'This if loop just checks to see if the doorbell is being pushed, and if it is then it blinks the green LED, and plays a tone on the speaker

if(Doorbell ==1) then
  freqout Speaker,300,150
  high Green
  pause 100
  low Green
else
  low Speaker
endif

'This if loop checks to see if the state of the switch has changed and if it has, then the next three if loops unlock, lock, or do nothing to the door depending on what state the lock is in. They also updates the last state of the switch and the lock

if(OC != OC_LS) then  'Checking open close switch to see if it has changed
This first if loop should be unnecessary and should be able to be absorbed into the two after it, but when I tried to incorporate it there were many bugs. Note that this loop just updates the last state of the switch and not the lock. This loop is crucial, because if someone locks the door from the outside with the RFID, then the switch must be switched up and then down before it will unlocks the door. This can be hard to visualize but a flow chart is included in the report to further illuminate the issue.

```c
if(OCC==1) then  'If the switch is high
    pause 15    'Pause to debounce
    if(OCC==1) then  'If the switch is still high
        OCC_LS = OCC  'Setting the last state of the switch
    endif
endif

if(OCC == 0) then  'If the the switch is low
    pause 15    'Pause to debounce
    if(OCC==0) then  'If switch is still low
        OCC_LS=OC  'Setting the last state of the switch.
    endif
endif
```

This if loop checks to see if the switch is high and the door unlocked, and if both of these are true then it locks the door and updates the state.

```c
if (OCC == 1) and (state == 0) then    'Checking to see if switch is high
    pause 15    'Pausing to make sure it is still high
    if (OCC==1)and(state ==0) then    'If high then rotate clockwise
        for i = 1 to 24
            OCCD Clockwise
        next i
        OCC_LS = OC  'Setting last state to high
        state = 1  'Setting door state to locked
    endif
endif
```

Table A1.1.1 Explanation of first if loop as seen above, this logic is necessary because we opted to not use limit switches or a servo motor to track position/lock state and instead did using code.

<table>
<thead>
<tr>
<th>Action</th>
<th>State of Switch</th>
<th>State of Lock</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Initialization</td>
<td>1</td>
<td>1</td>
<td>Program sets both high</td>
</tr>
<tr>
<td>Flip Switch</td>
<td>0</td>
<td>0</td>
<td>Lock and switch state are synced</td>
</tr>
<tr>
<td>Flip Switch</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Scan Correct RFID</td>
<td>1</td>
<td>0</td>
<td>Scanning RFID throws off the lock and switch status so they are no longer synced</td>
</tr>
<tr>
<td>Flip Switch</td>
<td>0</td>
<td>0</td>
<td>Flipping switch changes switch state but not lock</td>
</tr>
<tr>
<td>Flip Switch</td>
<td>1</td>
<td>1</td>
<td>Flipping switch again gets them back into sync and locks the door</td>
</tr>
</tbody>
</table>
if (OC == 0) and (state == 1) then   'Checking to see if switch is low and the door unlocked
    pause 15  'Pausing to make sure it is still low
    if (OC == 0) and (state == 1) then  'If low then rotate counter clockwise
        for i = 1 to 24
            gosub Counter_Clockwise
        next i
        OC_LS = OC  'Updating last to state to low
        state = 0  'Setting door state to unlocked
    endif
endif
endif

'Below are two main if loops that each scan to see if an RFID has been scanned
'and if the state of the door is locked or unlocked. This should have been
'able to be done in one overarching loop instead of two separate ones, but
'once again the PIC was having issues with that implementation.

if(RFID_R == 1) and (state == 0) then   'Checking to see if RFID has been
    scanned and the door is unlocked

if(RFID == 1) then   'Checking to see if correct RFID
    high Green
    pause 300   'Turn on green LED
    freqout Speaker,300,150   'Play correct tone
    pause 1000
    for i = 1 to 24   'Rotate Clockwise
        gosub Clockwise
    next i
    state = 1   'Update door state to locked
    low Green   'Turn off green LED
    RFID_R = 0   'Setting the RFID back to 0
    goto main   'The PIC was having trouble leaving this loop
    'so this and the above RFID reset were added
endif
if(RFID ==0) then 'Checking to see if incorrect RFID

   high Red 'Turn on red LED
   pause 300
   freqout Speaker,300,100 'Play incorrect tone
   pause 1000 'Pause to keep light on
   freqout Speaker,300,100 'Turn off red LED
   low Red
endif
endif

if(RFID_R == 1) and (state == 1) then 'Checking to see if RFID has been scanned and the door is locked

   if(RFID == 1) then 'Checking to see if correct RFID

      high Green 'Turn on green LED
      pause 300
      freqout Speaker,300,150 'Play correct tone
      for i = 1 to 24 'Rotate Clockwise
         gosub Counter_Clockwise
      next i
      state = 0

      'Pause for light to stay on
      low Green 'Turn off green LED
      RFID_R = 0 'Added to make sure PIC leaves loop
goto main
   endif
endif

if(RFID ==0) then 'Checking to see if incorrect RFID

   high Red 'Turn on red LED
   pause 300
   freqout Speaker,300,100 'Play incorrect tone
   pause 1000 'Pause to keep light on
   freqout Speaker,300,100 'Turn off red LED
   low Red
endif
endif
goto main     'Added to make sure PIC could exit loop
end

'The following subroutines are used to rotate the stepper motor

'Subroutine to turn stepper clockwise
Clockwise:
gosub step4
pause 5
gosub step3
pause 5
gosub step2
pause 5
gosub step1
pause 5
return

'Subroutine to turn stepper counter-clockwise
Counter_Clockwise:
gosub step1
pause 5
gosub step2
pause 5
gosub step3
pause 5
gosub step4
pause 5
return
Subroutines step1, step2, step3, step4 all excite different parts of stepper motor and when called in different orders cause the shaft to rotate.

step1:
    high step_a
    low step_b
    high step_c
    low step_d
    return

step2:
    high step_a
    low step_b
    low step_c
    high step_d
    return

step3:
    low step_a
    high step_b
    low step_c
    high step_d
    return

step4:
    low step_a
    high step_b
    high step_c
    low step_d
    return

stopmotor:
    low step_a
    low step_b
    low step_c
    low step_d
    return
A1.2 Arduino Code

The purpose of this code is to control the Arduino Nano in reading an MFRC-522 RFID reader, and then send data to the PIC when an incorrect or correct code was read. This data was sent through two bits from two digital pins.

```c
#include <SPI.h>
#include <MFRC522.h>

MFRC522 rfid(SS_PIN, RST_PIN); // Instance of the class

byte muidPICC[5];
int A; // Value representing RFID Card being read, used for debugging and testing
int B; // Value representing Correct RFID Card being read, used for debugging and testing

void setup()
{
    Serial.begin(9600); // Used for Debugging and Testing
    SPI.begin(); // Init SPI bus

    // Further Initializing RFID
    rfid.PCD_Init();

    // Initializing Output Pins used to control PIC
    pinMode(2, OUTPUT);
    pinMode(3, OUTPUT);

    // Further Setting Up RFID based on library
    for (byte i = 0; i < 6; i++)
    {
        key.keyByte[i] = 0xFF;
    }
    muidPICC[0] = 56;
    muidPICC[1] = 65;
    muidPICC[2] = 149;
    muidPICC[3] = 166;
}
```
void loop()
{
    // Look for new cards, this loop continually looks for card until one is scan
    if ( ! rfid.PICC_IsNewCardPresent())
    {
        return;
    }

    // Verify if the NUID has been read this loop reads the card until the code has been read.
    if ( ! rfid.PICC_ReadCardSerial())
    {
        return;
    }

    // If an incorrect RFID is scanned, it writes one pin high and one low.
    // Then it turns them both off after a short delay.
    // It also sets one variable high and one low
    // for debugging.
    // Notice that this if loop checks every single value in the array, so the ID has to be exactly correct.
    if ( rfid.uid.uidByte[0] != nuidPICC[0] ||
    {
        A = -1;
        B = 0;
        digitalWrite(2, HIGH);
        digitalWrite(3, LOW);
        delay(100);
        digitalWrite(2, LOW);
        digitalWrite(3, LOW);
    }

    // If the correct RFID is scanned, then it writes two pins high.
    // Then after a short delay it turns them both low.
    // It also sets two variables high for debugging.
    if ( rfid.uid.uidByte[0] == nuidPICC[0] &&
    {
        A = -1;
        B = 1;
        digitalWrite(2, HIGH);
        digitalWrite(3, HIGH);
        delay(100);
        digitalWrite(2, LOW);
        digitalWrite(3, LOW);
    }

    // Using the serial monitor to print debugging variables for testing
    // The system before connecting it to the PIC.
    Serial.println("A = ");
    Serial.println(A);
    Serial.println("B = ");
    Serial.println(B);
}