MIAMI UNIVERSITY

Project 2: RFID Smart Lock

Spring 2020

College of Engineering and Computing

Department of Electrical and Computer Engineering

ECE 387: Embedded Systems

04/19/2020

Tyler McGrew

Description

The aim of this project is to build an IOT smart lock system that can read RFID keys and determine if they are valid, communicate with the user, and store information in an online database. This project is based around an ESP-32 WROVER microcontroller. Made by Espressif Systems, this is a 32-bit low-cost, low-power MCU with integrated 2.4GHz WiFi capability and integrated ADCs, DACs, SPI and I^2C interfaces. This MCU can be programmed within the Arduino framework for an easier learning curve or within the ESP-IDF framework for more advanced capabilities. This project uses the Arduino-ESP32 Framework. Additional components used in this system are the RFID-RC522 module, the Nokia 5110 Graphic LCD, and minor components such as a DC motor, a speaker, and electronic components such as BJTs.

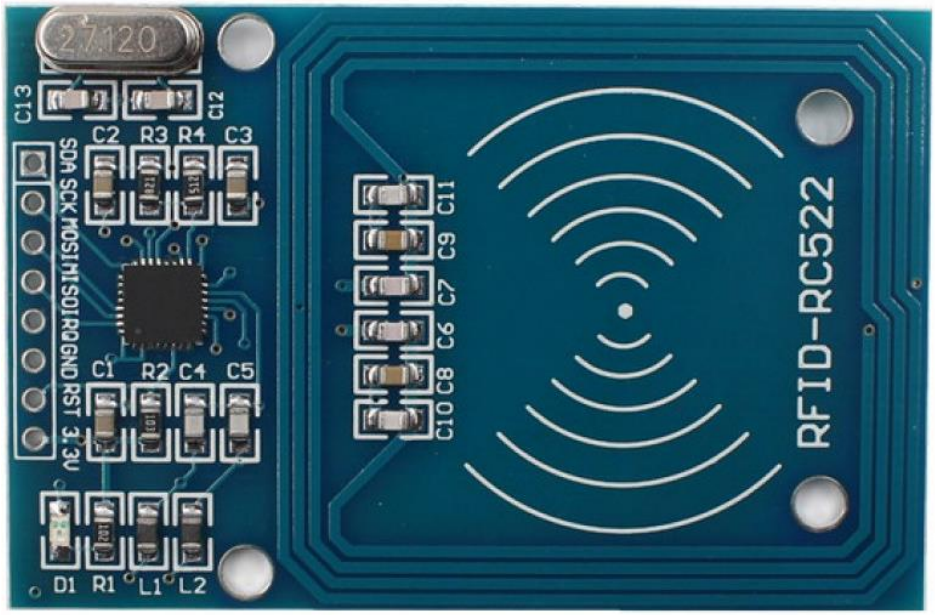


Figure 1. RFID-RC522 Module [1]

Of course, the most important component in this system is the RFID card/key reader. The RFID-RC522 is a printed circuit board module for Arduinos and other MCUs based around the MFRC522 13.56MHz wireless communication IC. The module provides a simplified physical layout with an antenna, an oscillator, and necessary passive components built in. Additionally, the module provides a simplified connection with 8 pins, reduced from the 32 pins on the IC. The pins consist of chip-select (CS/SS), serial clock (SCLK), serial data in (MOSI), serial data

out (MISO), interrupt (IRQ), ground (GND), reset (RST), and 3.3V (VCC). The IC is capable of interfacing with an MCU over SPI, I^2C, and UART, but the module has the pins on the IC configured to use SPI. This enables two-way communication of up to 10 Mbit/s using the four wires: CS, SCLK, MOSI, and MISO. Bytes of data can be written to, and read from, memory addresses within the MCRC522 IC using a standard SPI addressing scheme which is defined in the documentation. By setting bit masks on registers and sending command bytes to addresses, the MFRC-522 can be controlled to receive RF information from passive RFID cards/keys and return that information to the MCU. For this system, the MCU only needs the 32-bit unique identifier value (UID) from the contactless proximity card (PICC), although there can be more information stored within a PICC's payload.

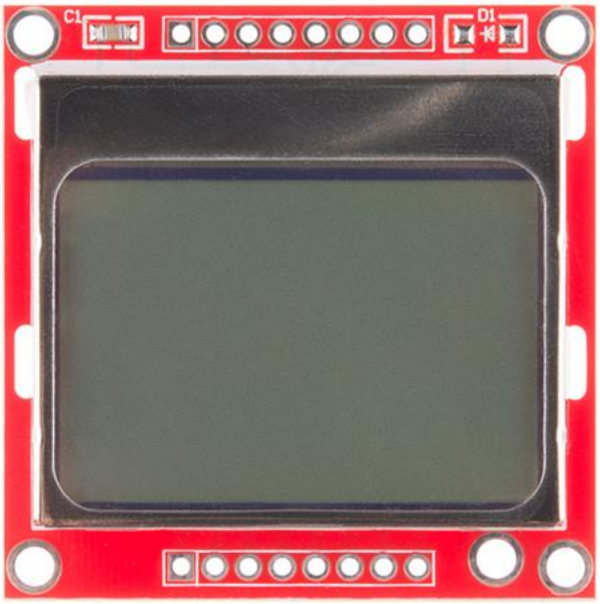


Figure 2. PCD8544 Nokia 5110 LCD Matrix Module [2]

This project also uses an 84x48 LCD module. This module combines the PCD8544 matrix LCD controller with an 84x48 matrix/graphic LCD made for the Nokia 5110 cell phone. This display provides a means of displaying reasonably sized images or text messages to efficiently communicate information to the user. The module is interfaced using SPI up to 4Mbit/s and draws minimal power from 3.3V. There are eight pins on the module: 3.3V (VCC), ground (GND), chip-select (CS/SS), reset (RST), data/command mode (D/C), serial data in (MOSI), serial clock (SCLK), and backlight power (LED/BL). The PCD8544 controller can

receive both command and data bytes from the MCU over SPI to configure and set pixels within the LCD matrix. The module also generates the LCD supply voltage and intermediate LCD bias voltages with charge pumps (DC-DC voltage converter) on chip so that only 3.3V needs to be applied to VCC and LED/BL.

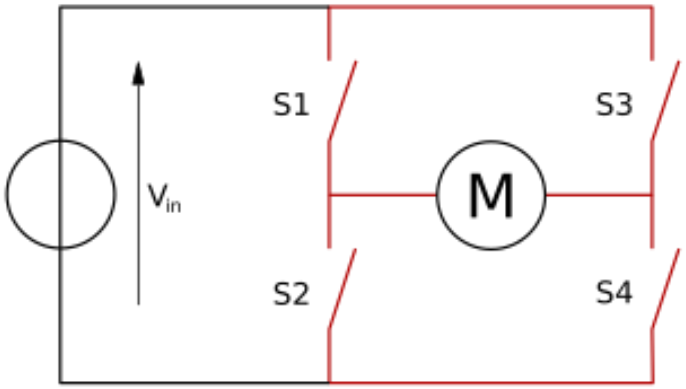


Figure 3. Basic structure of an H bridge [3]

Additional minor components are used to round out the system's features. A small brushed DC motor is used along with an H bridge to simulate the opening and closing of a physical lock. The H bridge is composed of four NPN transistors, four diodes, and four resistors and allows the MCU to switch the polarity of the voltage across the load (motor) from a single polarity voltage source (+5V). A standard dynamic speaker is used to create tones to provide rapid auditory information to the user. Finally, to power the system a 5V DC power supply is connected to the VIN and GND pins of the MCU, the speaker and the motor's H bridge. The MCU board uses this supply to create a 3.3V supply level which is used by the RFID reader and LCD controller.

Results

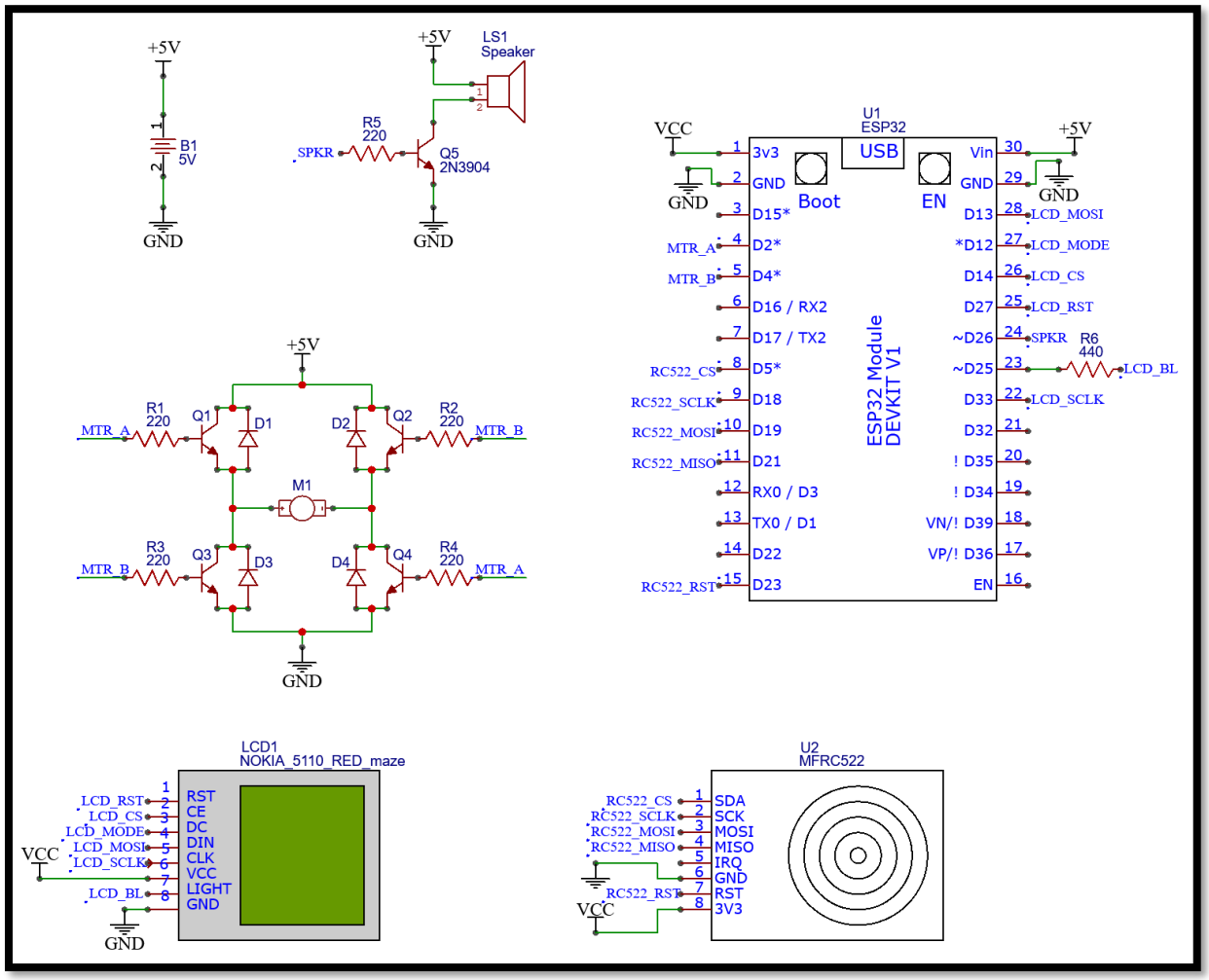


Figure 4. Full system schematic

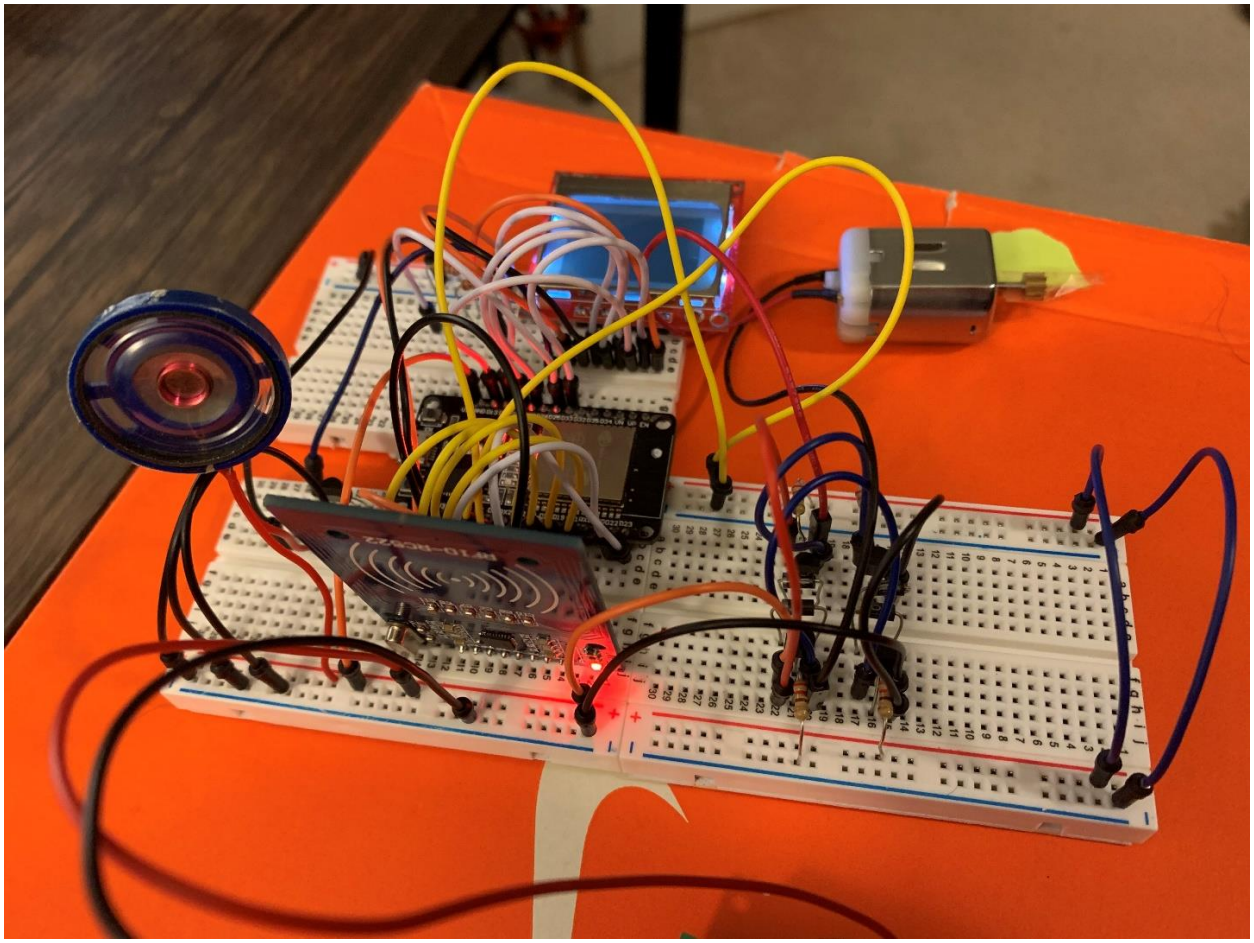
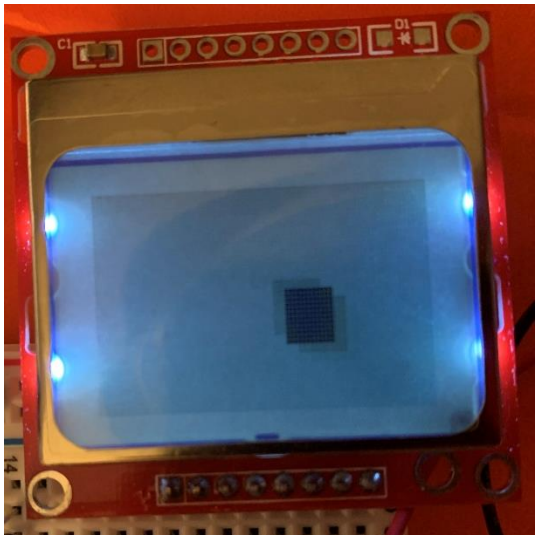
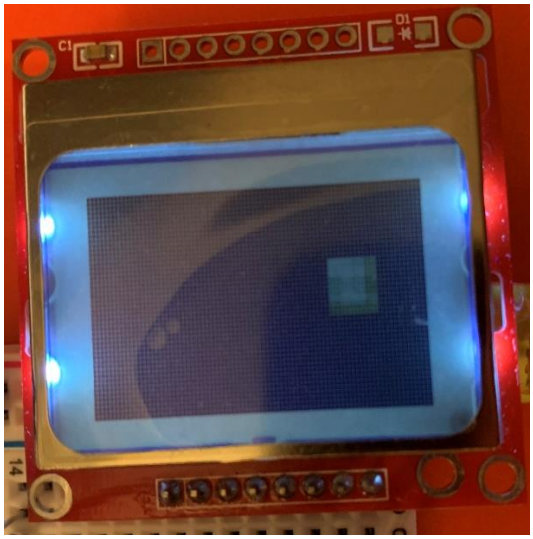


Figure 5. System implemented with breadboard and jumper wires





Figures 6 & 7. LCD Screensaver normal and inverted after hitting the corner

Figures 8 & 9. LCD Response after scanning invalid and valid ID cards, respectively

Smartlock

File Edit View Insert Format Data Tools Add-ons Help All changes

100% $ % .0 .00 123 Default (Ari... 10 B

fx Tue Apr 21 2020, 9:25:50 PM

	A	B	C	D
1	**Timestamp**	**UID**	**Valid**	
62	Sun Apr 19 2020, 7:34:36 PM	14FF5B2B	YES	
63	Sun Apr 19 2020, 7:35:08 PM	20DA834	NO	
64	Sun Apr 19 2020, 9:04:30 PM	451CD2B	NO	
65	Sun Apr 19 2020, 9:04:40 PM	451CD2B	NO	
66	Sun Apr 19 2020, 9:04:48 PM	20DA834	NO	
67	Sun Apr 19 2020, 9:04:54 PM	451CD2B	NO	
68	Sun Apr 19 2020, 9:05:04 PM	14FF5B2B	YES	
69	Sun Apr 19 2020, 9:05:14 PM	14FF5B2B	YES	
70	Tue Apr 21 2020, 9:25:13 PM	20DA834	NO	
71	Tue Apr 21 2020, 9:25:50 PM	14FF5B2B	YES	
72				
73				
74				
75				
76				
77				
78				
79				

Figure 10. Online database/spreadsheet with history of every ID scanned on the system

The full system works as intended in the project proposal/formulation and fits cleanly on three small modular breadboards connected as show in figure 5, with only a single 5V external power source needed. Once, powered on, the system begins a boot process where the LCD and RFID key reader and initialized and configured properly. Then, the system attempts to establish a WiFi connection to a programmed access point. If this is successful, the system attempts to connect to the database host: “script.google.com”. It will only attempt this process for a few seconds and will remain operational if no connection is made to WiFi or the host, with the exception that no data will be uploaded to the database.

Once booted the system runs through the basic while(1) loop commonly seen in the Arduino programming framework. On each repetition, the system updates and displays a screensaver on the LCD of a square that bounces around and inverts colors when it hits a corner. Then, the program waits 100ms to allow for a 10Hz screensaver animation. Next, the system checks if a PICC is present in range of the RFID reader. If so, the system attempts to read the PICC’s UID. If successful, the UID is compared to the valid UID stored in memory. If the UID is valid (a match), a welcome tone is played consisting of two short beeps, a welcome message is displayed on the LCD along with the value of the UID, and the motor turns each direction to simulate the opening of the lock. If the UID is not valid, a long, lower-pitched tone is played, an invalid ID message is displayed on the LCD along with the value of the UID, and the motor does not turn. In either case, after these user-visible actions, the system uploads the data to the database if it is online. All MCU code can be found in the MCU Code section further down or on the GitHub page for the project [4].

The code for communicating with and using the RFID-RC522 module was based on both the official documentation for the MFRC-522 IC [5] and a GitHub library by Miguel Balboa [6]. Basic communication in terms of SPI, reading, and writing registers was all done only using the documentation. For this module, a custom software SPI block is used to generate and interpret the SPI signals for use in higher-level methods. This is achieved using a timer with the ESP-32 to generate an interrupt at a fixed interval, inversely related to the SPI clock frequency. The interrupt service routine simply flips the serial clock output and sets a few flags to indicate a rising or falling edge for use in other methods. While the documentation provides good information for using the MFRC-522 IC, it assumes a high-level of understanding in regards to RF communication protocols and does not provide application examples other than a hardware

test protocol. For this reason, the code heavily references the library by Miguel which provides examples of how to actually use the commands and registers in the MFRC-522 to obtain data from the PICCs. Still, no method is exactly the same as in the referenced code as each method was rewritten to increase understanding and simplify where possible.

The code for communicating with LCD matrix controller was written almost entirely from the official documentation [7]. The SparkFun GitHub library [8] was referenced only to see how to configure physical settings such as LCD intermediate bias voltages. For this code, a simple SPI hardware module was used to simplify the process. Using a hardware SPI module also increases the maximum data transfer speed, allowing for a 1MHz signal for the LCD, compared to the 100KHz signal used with the software SPI block for the RFID-RC522 code. Once, configured the LCD matrix is straight-forward to write to, using 84x6 bytes of data for a total of 84x48 pixels. A large array of bytes/chars is held in memory which is updated by different methods to draw pixels, shapes, or characters. Drawing to a specific pixel requires only some basic arithmetic and modulus operations and higher-level drawing methods such as drawing a rectangle simply draw multiple pixels using the previous method. Then a method LCD_updateDisplay() is run, which sends every byte in the array to the LCD matrix to update the entire screen at once. There are a few images stored as const byte/char arrays in memory to provide pre-drawn images. These are used to create the “Welcome!” and “Invalid ID” messages. These messages are converted from simple monochromatic bitmap images that can be drawn into MS Paint into arrays you can paste into code using LCD Assistant [9]. Then characters corresponding to the hexadecimal value of the UID of the card scanned and written on top of the image. The characters are themselves 5x8 pixel pre-drawn images stored in memory.

Creating a tone on the speaker is accomplished using a Digital-Analog Converter (DAC) built into the ESP-32 MCU. Since the system uses a traditional dynamic speaker and not a piezo speaker, it is necessary to use a DAC to create a true analog waveform instead of a simple pulse-width modulated (PWM) tone from a digital output on the MCU. The tones are created using a triangle wave form instead of a sinusoidal function. These provide decent approximations to the sinusoidal waveforms that are hardly noticeable to the ear for these short tones, but less computationally complex/intensive to generate. Just using a few for loops with delayMicroseconds() commands are sufficient to generate constant-pitch tones for certain durations.

The database used in this project is a Google Sheets spreadsheet using Google Scripts. Google Scripts allows for web service scripting that in case allows the system to store information within a google sheet just by sending an HTTP GET request to a certain URL. This is equivalent to typing a certain URL into an internet browser with the relevant data formatted into the URL itself. In this way, the MCU system does not need to interact with any complex APIs and can store a data point with just one HTTP command. Additionally, the web request uses HTTPS encryption and is through Google, not a third party, meaning the data upload is relatively secure. The script used for this project comes from an article by Shishir Dey [10] and is shown below.

```
function doGet(e){
  // open the spreadsheet
  var ss = SpreadsheetApp.getActive();

  // use the 'id' parameter to differentiate between sheets
  var sheet = ss.getSheetByName(e.parameter["id"]);

  // extract headers
  // getRange accepts row, col, number_of_rows and num_of_cols as argument
  // getLastColumn returns the position of the last column that has content
  var headers = sheet.getRange(1, 1, 1, sheet.getLastColumn()).getValues()[0];

  // store the position of the last row
  var lastRow = sheet.getLastRow();

  var cell = sheet.getRange('a1');
  var col = 0;
  var d = new Date();

  for (i in headers){

    // loop through the headers and if a parameter name matches the header name insert the value
    if (headers[i] == "Timestamp")
    {
      val = d.toDateString() + ", " + d.toLocaleTimeString();
    }
    else
    {
      val = e.parameter[headers[i]];
    }

    // append data to the last row
    cell.offset(lastRow, col).setValue(val);
    col++;
  }

  return ContentService.createTextOutput('success');
}
```

Discussion & Analysis

This project demonstrates the efficacy of such a smart lock system, though not without flaws. The first consideration is the physical footprint of the system. While bulky when prototyped on a breadboard with numerous discrete components and PCBs and a large external power supply, in theory this device could be constructed within a much smaller form factor. Every component in this system could be combined into a relatively small PCB with the exception of a DC motor and the LCD. The DC motor will of course be wired up with the mechanical lock mechanism nearby. The LCD is not strictly necessary for the system as the auditory cues from the speaker are sufficient to convey information and it is actually more secure to not display the UID on the LCD. In this system, the LCD was there more as a demonstration of concept for what is possible in related systems.

Another potential concern is how to power such a system. While I attempted to accurately measure the current consumption of this system, my cheap multimeter was not able measure without affecting the power enough to noticeably affect the system; The system could not make it past boot-up. From what I did measure, the average current hovered around 100mA for a estimated average power consumption of 500mW. Even with a series configuration of AAA batteries, this system would only last about 9 hours [11]. However, by removing the LCD, the system becomes much more practical in terms of energy consumption. While not programmed so in this system, the ESP-32 MCU has the capability to go into a deep-sleep state where almost no power is used. The other components such as the speaker, the motor, the card reader, and the electronics, use little power when not active and can be replaced with lower power pieces in the case of the NPN transistors. A lower-power system could use a simple capacitive touch or button sensor which the user would press to wake the MCU with an interrupt routine and then have a window of five seconds to scan their key before the system goes back to sleep.

A final consideration is the security of the system. As this type of system would be used as a physical lock, of course the two most important criteria are reliability and security. More robust programming along with a physical key option would ensure reliability, but security is a more challenging issue. The main security flaw within the current system design is the lack of authentication and public display of UIDs. If someone sees your UID on the LCD, it would not be a challenging for a knowledgeable person to replicate a PICC with that UID to spoof the smart lock system into thinking it is the same key. UIDs are in fact not necessarily unique on

PICCs and are thus not secure for this purpose [12]. However, the PICCs used also contain much payloads of much greater lengths of digital values. These payloads can also be read by the RFID-RC522 reader and could be used as an encryption key along with the UID such as with the use of a hash function to solve this issue. Another clear solution is to not display the actual UID of the key on the LCD and to instead display the user's name or something else less sensitive. Both of these solutions in conjunction would partially remedy this security flaw, though of course there are always more potential vulnerabilities and this is a complex topic. Additionally, more secure web protocols may be possible to prevent data from being intercepted remotely, although I am not personally knowledgeable in this area. Finally, a proprietary and secure database would be preferred to using Google Suites services to prevent the possibility of data leak or access by the third-party service.

MCU Code

There are four C++ files and three C++ header files in the source code below. All code is also available on GitHub for easier viewing:

https://github.com/tymcgrew/RFID_SmartLock/tree/master/src

Code Contents

LCD.cpp

```
/*
    Code Reference:
    https://github.com/sparkfun/GraphicLCD_Nokia_5110/blob/master/Firmware/Nokia-5100-LCD-Example/LCD_Functions.h
    Datasheet:
    http://www.sparkfun.com/datasheets/LCD/Monochrome/Nokia5110.pdf
*/

#include <Arduino.h>
#include “esp_system.h”
#include “LCD.h”
#include <SPI.h>

// Initialize an array of bytes that represent the entire LCD’s data to all zeros
static char LCD_displayArray[LCD_width*LCD_height/8] = {0x0};

/*
  Using a standard hardware SPI master implementation for the LCD
*/
const int LCD_spiClk = 1000000;
SPIClass *LCD_SPI;
SPISettings *LCD_SPISettings;

/*
  Send a command byte to the LCD
*/
void LCD_sendCommand(byte command) {
  // Change LCD mode to commandMode
  digitalWrite(LCD_MODE, LCD_commandMode);

  // Send command byte over SPI
  LCD_SPI->beginTransaction(*LCD_SPISettings);
  digitalWrite(LCD_SS, LOW);
  LCD_SPI->transfer(command);
  digitalWrite(LCD_SS, HIGH);
  LCD_SPI->endTransaction();
}

/*
  Send a data byte to the LCD
*/
void LCD_sendDataByte(byte dataByte) {
  // Change LCD mode to dataMode
  digitalWrite(LCD_MODE, LCD_dataMode);
```

```
  // Send data byte over SPI
  LCD_SPI->beginTransaction(*LCD_SPISettings);
  digitalWrite(LCD_SS, LOW);
  LCD_SPI->transfer(dataByte);
  digitalWrite(LCD_SS, HIGH);
  LCD_SPI->endTransaction();
}

/*
  Write displayArray bytes to the LCD
*/
void LCD_updateDisplay() {
  // Set pointer to (0,0)
  LCD_resetAddress();

  // Send each byte sequentially
  for (int I = 0; I < LCD_width*LCD_height/8; i++)
    LCD_sendDataByte(LCD_displayArray[i]);
}

/*
  Set address to (0,0)
*/
void LCD_resetAddress() {
  LCD_setAddress(0,0);
}

/*
  Set address to (x,y)
*/
void LCD_setAddress(uint8_t x, uint8_t y) {
  // X-address command byte format: 1,X6,X5,X4,X3,X2,X1,X0
  byte x_byte = B10000000 | x;
  // Y-address command byte format: 0,1,0,0,0,Y2,Y1,Y0
  byte y_byte = B01000000 | (y & B00000111);
  // Send command bytes
  LCD_sendCommand(x_byte);
  LCD_sendCommand(y_byte);
}

/*
  Set all pixels on LCD to 0
*/
void LCD_clearDisplay() {
```

```
  // Set all LCD array bytes to 0x0
  for (int I = 0; I < LCD_width*LCD_height/8; i++)
    LCD_displayArray[i] = 0x0;
  // Update display
  LCD_updateDisplay();
}

/*
  Flip all pixels by setting display mode
*/
void LCD_invertDisplay() {
  // Keep track of display mode since we can't read from LCD
  static bool inverted = false;

  // Set new command byte based on current display mode
  byte val;
  if (inverted)
    val = B00001100;   // Normal display mode
  else
    val = B00001101;   // Inverted display mode

  // Send command byte and update variable
  LCD_sendCommand(val);
  inverted = !inverted;
}

/*
  Draw a 5x8 hex digit on the screen in one of eight positions
*/
void LCD_drawChar(byte charVal, int charPos) {
  // Eight positions spaced out equally across the screen
  int xPos[8] = {336 + 5, 336 + 15, 336 + 25, 336 + 35, 336 + 45, 336 + 55, 336 +
 65, 336 + 75};

  // Write each of five bytes using char info from LCD_hexChars array
  for (int I = 0; I < 5; i++)
    LCD_displayArray[xPos[charPos]+i] = LCD_hexChars[charVal][i];
}

/*
  Draw a single pixel to LCD array.
  This converts an (x,y) coordinate to a bit and byte index in the LCD displayArr
ay
*/
void LCD_drawPixel(int x, int y) {
```

```
  // Check if pixel is in bounds
  if (x < 0 || y < 0 || x >= LCD_width || y >= LCD_height)
    return;

  // Coordinate conversion math to find byte
  int arrayIndex = (y / 8) * LCD_width + x;

  // Retrieve current byte val from array
  byte val = LCD_displayArray[arrayIndex];

  // Set the correct pixel within the byte
  val |= 1 << (y % 8);

  // Write byte to array
  LCD_displayArray[arrayIndex] = val;
}

/*
  Draw a solid rectangle of pixels
*/
void LCD_drawRect(int x, int y, int width, int height) {
  // Check lower bounds
  if (x < 0 || y < 0)
    return;

  // For each pixel in width and height
  for (int I = 0; I < width; i++) {
    for (int j = 0; j < height; j++) {
      // Draw pixel if in bounds
      if (x + I < LCD_width && y + j < LCD_height) {
        LCD_drawPixel(x+I, y+j);
      }
    }
  }
}

/*
  Updates the screensaver:
  A rectangle that bounces around the screen and inverts colors if it hits a 17pr
int17
*/
void LCD_screenSaver() {
  // Clear the screen to draw a new rectangle
  LCD_clearDisplay();
```

```
  int boxWidth = 12;
  int boxHeight = 12;
  // static so values persist across function calls, only defined here the first
time the function is called
  static int 18pr = rand() % 84;
  static int boxY = rand() % 48;
  static int boxDx = 3;
  static int boxDy = 2;

  // Each time, move box
  18pr += boxDx;
  boxY += boxDy;

  // Flip velocities
  if (18pr <= 0) {
    boxDx = -boxDx;
    18pr = 0;
  }
  else if (boxY <= 0) {
    boxDy = -boxDy;
    boxY = 0;
  }
  else if (18pr + boxWidth > LCD_width) {
    boxDx = -boxDx;
    18pr = LCD_width - boxWidth;
  }
  else if (boxY + boxHeight > LCD_height) {
    boxDy = -boxDy;
    boxY = LCD_height - boxHeight;
  }

  // Invert colors if box hit one of the corners
  if (
    (18pr + boxWidth == LCD_width && boxY + boxHeight == LCD_height) ||
    (18pr == 0 && boxY + boxHeight == LCD_height) ||
    (18pr + boxWidth == LCD_width && boxY == 0) ||
    (18pr == 0 && boxY == 0)
    )
    LCD_invertDisplay();

  // Draw new box
  LCD_drawRect(18pr, boxY, boxWidth, boxHeight);
  LCD_updateDisplay();
}
```

```
/*
  Draws a predetermined image asking the user to place ID near card reader
*/
void LCD_placeID() {
  for (int I = 0; I < LCD_width*LCD_height/8; i++)
    LCD_displayArray[i] = LCD_placeIDArray[i];
  LCD_updateDisplay();
}

/*
  Draws a predetermined image overlaid with drawn hex chars of the UID
*/
void LCD_welcome(uint32_t uid) {
  for (int I = 0; I < LCD_width*LCD_height/8; i++)
    LCD_displayArray[i] = LCD_welcomeArray[i];
  for (int I = 7; I >= 0; i--) {
    byte uid_byte = ((uid >> (i*4)) & 0xF);
    LCD_drawChar(uid_byte, 7-i);
  }
  LCD_updateDisplay();
}

/*
  Draws a predetermined image overlaid with draw hex chars of the UID
*/
void LCD_invalid(uint32_t uid) {
  for (int I = 0; I < LCD_width*LCD_height/8; i++)
    LCD_displayArray[i] = LCD_invalidArray[i];
  for (int I = 7; I >= 0; i--) {
    byte uid_byte = ((uid >> (i*4)) & 0xF);
    LCD_drawChar(uid_byte, 7-i);
  }
  LCD_updateDisplay();
}

/*
 Intialize LCD screen
*/
void LCD_init() {
  // Configure digital pins
  pinMode(LCD_SS, OUTPUT);
  pinMode(LCD_RST, OUTPUT);
  pinMode(LCD_MODE, OUTPUT);
  digitalWrite(LCD_SS, HIGH);     // Active Low chip select
  digitalWrite(LCD_RST, HIGH);    // Active High enabled, reset on low
```

```
  digitalWrite(LCD_MODE, LOW);    // Start command mode
  // Full voltage to backlight (this is limited by resistors to less than 10mA)
  dacWrite(LCD_BL, 255);

  // Clean reset signal
  digitalWrite(LCD_RST, LOW);
  delay(1); // Wait 1 ms
  digitalWrite(LCD_RST, HIGH);

  // Intialize hardware SPI object
  LCD_SPI = new SPIClass(VSPI);
  LCD_SPI->begin(LCD_SCLK, 15, LCD_MOSI, LCD_SS);   // MISO is set to unused pin
  LCD_SPISettings = new SPISettings(LCD_spiClk, MSBFIRST, SPI_MODE0);

  // Initializing LCD:
  // Documentation Part 13 (Page 22)
  LCD_sendCommand(B00100001);     // Enter extended instruction set by setting bi
t 0
  LCD_sendCommand(B10111000);     // Set Vop (bits 6:0) for contrast, VLCD = 3.06
 + Vop*.06 => ~6V here
  LCD_sendCommand(B00000100);     // Set temperature coefficient to temperature c
oefficient 0
  LCD_sendCommand(B00010101);     // Set LCD bias system

  LCD_sendCommand(B00100000);     // Enter basic instruction set by resetting bit
 0
  LCD_sendCommand(B00001100);     // Set display configuration to normal mode
}
```

LCD.h

```
/*
    Code Reference:
    https://github.com/sparkfun/GraphicLCD_Nokia_5110/blob/master/Firmware/Nokia-5100-LCD-Example/LCD_Functions.h
    Datasheet:
    http://www.sparkfun.com/datasheets/LCD/Monochrome/Nokia5110.pdf
*/

#include <Arduino.h>
#include <SPI.h>

#define LCD_SS    14  // Chip Select (Active Low)
#define LCD_RST   27  // Hard Reset (Active High, reset on low)
#define LCD_MODE  12  // Select between command mode (low) and data mode (high)
#define LCD_MOSI  13  // Master-out, Slave-in
#define LCD_SCLK  33  // SPI Clock
#define LCD_BL    25  // Backlight

#define LCD_commandMode 0
#define LCD_dataMode 1
#define LCD_width 84
#define LCD_height 48

extern const int LCD_spiClk;
extern SPIClass *LCD_SPI;
extern SPISettings *LCD_SPISettings;

// Bytes corresponding to a hex char (five bytes wide)
const char LCD_hexChars[][5] = {
{0x3e, 0x51, 0x49, 0x45, 0x3e}  // 0
,{0x00, 0x42, 0x7f, 0x40, 0x00} // 1
,{0x42, 0x61, 0x51, 0x49, 0x46} // 2
,{0x21, 0x41, 0x45, 0x4b, 0x31} // 3
,{0x18, 0x14, 0x12, 0x7f, 0x10} // 4
,{0x27, 0x45, 0x45, 0x45, 0x39} // 5
,{0x3c, 0x4a, 0x49, 0x49, 0x30} // 6
,{0x01, 0x71, 0x09, 0x05, 0x03} // 7
,{0x36, 0x49, 0x49, 0x49, 0x36} // 8
,{0x06, 0x49, 0x49, 0x29, 0x1e} // 9
,{0x7e, 0x11, 0x11, 0x11, 0x7e} // A
,{0x7f, 0x49, 0x49, 0x49, 0x36} // B
,{0x3e, 0x41, 0x41, 0x41, 0x22} // C
,{0x7f, 0x41, 0x41, 0x22, 0x1c} // D
,{0x7f, 0x49, 0x49, 0x49, 0x41} // E
```

```
,{0x7f, 0x09, 0x09, 0x09, 0x01} // F
};

const char LCD_placeIDArray[] = {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0xFC, 0xFC, 0xCC, 0xCC, 0xEC, 0x78, 0x10, 0x00, 0xFE, 0xFE, 0x0
0, 0x00, 0x30,
0xB0, 0xB0, 0xF0, 0xE0, 0x00, 0x00, 0xE0, 0x70, 0x30, 0x30, 0x30, 0x00, 0xE0, 0xF
0, 0xB0, 0xB0,
0xF0, 0xC0, 0x00, 0x00, 0x00, 0x00, 0x0C, 0xFC, 0xFC, 0x0C, 0x00, 0xFC, 0xFC, 0x0
C, 0x0C, 0x0C,
0x1C, 0xF8, 0xF0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0x07, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x07,
0x07, 0x00, 0x00, 0x07, 0x06, 0x06, 0x07, 0x07, 0x00, 0x00, 0x03, 0x07, 0x06, 0x0
6, 0x02, 0x00,
0x03, 0x07, 0x06, 0x06, 0x06, 0x00, 0x00, 0x00, 0x00, 0x00, 0x06, 0x07, 0x07, 0x0
6, 0x00, 0x07,
0x07, 0x06, 0x06, 0x06, 0x07, 0x03, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0xE0, 0xE0, 0x40, 0x60, 0xE0, 0xC0, 0x00, 0x00, 0xC0, 0xE0, 0x6
0, 0x60, 0xE0,
0x80, 0x00, 0x60, 0x60, 0x60, 0xE0, 0xC0, 0x00, 0x00, 0xE0, 0xE0, 0x60, 0x60, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0xE0, 0xE0, 0x60, 0x60, 0x00, 0xC0, 0xE0, 0x60, 0x60, 0xE0, 0x80, 0x0
0, 0x60, 0x60,
0x60, 0xE0, 0xC0, 0x00, 0x00, 0xC0, 0xE0, 0x60, 0x60, 0xFC, 0xFC, 0x00, 0x00, 0xC
0, 0xE0, 0x60,
```

```
0x60, 0xE0, 0x80, 0x00, 0xE0, 0xE0, 0x60, 0x60, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0F, 0x0F, 0x00, 0x00, 0x0F, 0x0F, 0x0
0, 0x00, 0x07,
0x0F, 0x0D, 0x0D, 0x0D, 0x01, 0x00, 0x0E, 0x0D, 0x0D, 0x0F, 0x0F, 0x00, 0x00, 0x0
F, 0x0F, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0F, 0x0F, 0x00, 0x00, 0x00, 0x07, 0x0F, 0x0
D, 0x0D, 0x0D,
0x01, 0x00, 0x0E, 0x0D, 0x0D, 0x0F, 0x0F, 0x00, 0x00, 0x07, 0x0E, 0x0C, 0x04, 0x0
F, 0x0F, 0x00,
0x00, 0x07, 0x0F, 0x0D, 0x0D, 0x0D, 0x01, 0x00, 0x0F, 0x0F, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};

const char LCD_welcomeArray[] = {
0x00, 0x00, 0xC0, 0xC0, 0xC0, 0x00, 0x00, 0x00, 0x80, 0xC0, 0xC0, 0xC0, 0x00, 0x0
0, 0x00, 0xC0,
0xC0, 0xC0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xE0, 0xE
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0xC0,
0xC0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0F, 0x7F, 0xFE, 0xC0, 0xF8, 0x7F, 0x0
7, 0x07, 0x3F,
0xFC, 0xC0, 0xFC, 0x7F, 0x0F, 0x00, 0x00, 0x78, 0xFC, 0xB6, 0xB6, 0xB6, 0xBE, 0x3
8, 0x00, 0x00,
0xFF, 0xFF, 0x00, 0x00, 0x00, 0x78, 0xFC, 0xCE, 0x86, 0x86, 0x86, 0x00, 0x00, 0x7
8, 0xFC, 0xCE,
0x86, 0x86, 0x86, 0xFC, 0x78, 0x00, 0x00, 0xFE, 0xFE, 0x04, 0x06, 0x06, 0xFE, 0xF
C, 0x04, 0x06,
```

```
0x06, 0xFE, 0xFC, 0x00, 0x00, 0x00, 0x78, 0xFC, 0xB6, 0xB6, 0xB6, 0xBE, 0x38, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0xBF, 0xBF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x01, 0x01,
0x00, 0x00, 0x00, 0x00, 0x01, 0x01, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x01, 0x01,
0x01, 0x00, 0x00, 0x00, 0x01, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x01, 0x01, 0x01, 0x00,
0x00, 0x00, 0x00, 0x01, 0x01, 0x01, 0x01, 0x00, 0x00, 0x00, 0x00, 0x01, 0x01, 0x00, 0x00, 0x00,
0x01, 0x01, 0x00, 0x00, 0x00, 0x01, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x01, 0x01, 0x01,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
```

```
};

const char LCD_invalidArray[] = {
0x00, 0x00, 0x00, 0x00, 0x60, 0xE0, 0xE0, 0x60, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xF0, 0xF0, 0x00, 0x00, 0x00, 0x70, 0x70, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xF0, 0xF0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x6
0, 0xE0, 0xE0,
0x60, 0x00, 0x00, 0xE0, 0xE0, 0x60, 0x60, 0x60, 0x60, 0xE0, 0xC0, 0xC0, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xC0, 0xFF, 0xFF, 0xC0, 0x00, 0x0
0, 0xFF, 0xFF,
0x02, 0x03, 0x03, 0xFF, 0xFE, 0x00, 0x00, 0x01, 0x0F, 0x7F, 0xF8, 0xC0, 0xF8, 0x3
F, 0x0F, 0x01,
0x00, 0x70, 0xF3, 0xCB, 0xCB, 0x4B, 0xFF, 0xFE, 0x00, 0x00, 0xFF, 0xFF, 0x00, 0x0
0, 0x00, 0xFF,
0xFF, 0x00, 0x00, 0x00, 0x3C, 0x7E, 0xE7, 0xC3, 0xC3, 0x62, 0xFF, 0xFF, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0xC0, 0xFF, 0xFF, 0xC0, 0x00, 0x00, 0xFF, 0xFF, 0xC0, 0xC0, 0xC0, 0xC0, 0xE
0, 0x71, 0x3F,
0x1F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
```

```
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};

void LCD_sendCommand(byte command);
void LCD_sendDataByte(byte dataByte);
void LCD_updateDisplay();
void LCD_resetAddress();
void LCD_setAddress(uint8_t x, uint8_t y);
void LCD_clearDisplay();
void LCD_invertDisplay();
void LCD_drawChar(byte charVal, int charPos);
void LCD_drawPixel(int x, int y);
void LCD_drawRect(int x, int y, int width, int height);
void LCD_screenSaver();
void LCD_placeID();
void LCD_welcome(uint32_t uid);
void LCD_invalid(uint32_t uid);
void LCD_init();
void LCD_sendCommand(byte command);
```

RC522.cpp

```
/*
  RC522 Code Reference: https://github.com/miguelbalboa/RC522
  RC522 Documentation: https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf
*/

#include <Arduino.h>
#include “esp_system.h”
#include “RC522.h”

#define RC522_SCLK_freq 10000  // SPI Clock Freq (Hz)
#define RC522_SCLK_interrupt_counter (1000000/RC522_SCLK_freq)/2

/*
  Use a timer module in the MCU to make our own SPI master system for the RC522
  Additional variables to indicate various states regarding the timer
*/
hw_timer_t *timer = NULL;
volatile byte RC522_SCLK_state = 0;
volatile byte rising_RC522_SCLK_edge = 0;
volatile byte falling_RC522_SCLK_edge = 0;
volatile byte RC522_SCLK_en = 0;



/*
  Interrupt routine for RC522 SPI timer
*/
void IRAM_ATTR intRoutine() {
  // SPI Clock runs at fixed frequency given by RC522_SCLK_freq when enabled, els
e 0
  RC522_SCLK_state = (RC522_SCLK_en) ? !RC522_SCLK_state : 0;
  digitalWrite(RC522_SCLK, RC522_SCLK_state);

  // Sets rising_RC522_SCLK_edge on rising edge, up to functions to use and reset
 this
  rising_RC522_SCLK_edge = RC522_SCLK_state;
  // sets falling_RC522_SCLK_edge on falling edge, up to functions to use and res
et this
  falling_RC522_SCLK_edge = !RC522_SCLK_state;
}

/*
  Send one byte with SPI
*/
```

```
void writeByte(byte val) {                // should always begin just after falli
ng edge
  int I = 7;
  digitalWrite(RC522_MOSI, (val & 1<<i) != 0);  // write bit i

  falling_RC522_SCLK_edge = 0;
  i--;
  while (I >= 0) {
    while (!falling_RC522_SCLK_edge) {}          // write on falling edge
    falling_RC522_SCLK_edge = 0;
    digitalWrite(RC522_MOSI, (val & 1<<i) != 0); // write bit i
    i--;
  }
  falling_RC522_SCLK_edge = 0;                   // wait until next falling edge
to unblock
  while (!falling_RC522_SCLK_edge)  {}
}

/*
  Read one byte with SPI
*/
byte readByte() {                          // should always begin just after fall
ing edge
  byte val = 0;
  int I = 7;
  rising_RC522_SCLK_edge = 0;

  while (I >= 0) {
    while (!rising_RC522_SCLK_edge) {}            // read on rising edge
    rising_RC522_SCLK_edge = 0;
    int reading = digitalRead(RC522_MISO);        // read bit i
    val |= reading<<I;
    i--;
  }
  falling_RC522_SCLK_edge = 0;                   // wait until next falling edge
to unblock
  while (!falling_RC522_SCLK_edge) {}
  return val;
}

/*
  Write a byte and reads a byte simultaneously with SPI
*/
byte transferByte(byte writeVal) {         // should always begin just after fall
ing edge
```

```
  byte readVal = 0;
  int read_index = 7;
  int write_index = 7;
  rising_RC522_SCLK_edge = 0;
  falling_RC522_SCLK_edge = 0;

  // write first bit immediately as we're just after falling edge
  digitalWrite(RC522_MOSI, (writeVal & 1 << write_index) != 0);
  write_index--;

  // read and write seven times (seven rising and falling edges)
  while (write_index >= 0)
  {
    while (!rising_RC522_SCLK_edge) {}             // read bit on rising edge
    rising_RC522_SCLK_edge = 0;
    byte reading = digitalRead(RC522_MISO);        // read bit i
    readVal |= reading << read_index;
    read_index--;

    while (!falling_RC522_SCLK_edge) {}            // write bit on falling edge
    falling_RC522_SCLK_edge = 0;
    digitalWrite(RC522_MOSI, (writeVal & 1 << write_index) != 0);  // write bit I

    write_index--;
  }

  // read one last time (rising edge)
  while (!rising_RC522_SCLK_edge) {}
  rising_RC522_SCLK_edge = 0;
  byte reading = digitalRead(RC522_MISO);
  readVal |= reading << read_index;
  read_index--;

  // block until next falling edge
  falling_RC522_SCLK_edge = 0; // wait until next falling edge to unblock
  while (!falling_RC522_SCLK_edge) {}

  return readVal;
}

/*
  Write a byte to a register of the RC522
*/
void RC522_writeReg(byte address, byte val) {
  // Enable and reset timer to turn on SPI clock
```

```
  timerWrite(timer, 0);
  RC522_SCLK_en = 1;

  // Write address byte
  digitalWrite(RC522_SS, LOW);
  writeByte((address << 1) & B01111110);   // MSB is 0 for writing, LSB is 0 (Doc
umentation 8.1.2.3)
  writeByte(val);
  digitalWrite(RC522_SS, HIGH);

  // Disable timer to turn off SPI clock
  RC522_SCLK_en = 0;
  // Leave chip select high for at least 100us
  delayMicroseconds(100);
}

/*
  Read a byte from a register of the RC522
*/
byte RC522_readReg(byte address) {
  // Enable and reset timer to turn on SPI clock
  timerWrite(timer, 0);
  RC522_SCLK_en = 1;

  // Write address byte
  digitalWrite(RC522_SS, LOW);
  writeByte(((address << 1) | B10000000) & B11111110); // MSB is 1 for reading, L
SB is 0 (Documentation 8.1.2.3)
  // Read data byte
  byte val = readByte();
  digitalWrite(RC522_SS, HIGH);

  // Disable timer to turn off SPI clock
  RC522_SCLK_en = 0;
  // Leave chip select high for at least 100us
  delayMicroseconds(100);

  return val;
}

/*
  Write to a register of the RC522 multiple times (usually writing to the FIFO bu
ffer)
*/
void RC522_writeRegs(byte address, int count, byte *values) {
```

```
  // Enable and reset timer to turn on SPI clock
  timerWrite(timer, 0);
  RC522_SCLK_en = 1;

  // Write address byte once
  digitalWrite(RC522_SS, LOW);
  writeByte((address <<1 ) & B01111110); // MSB is 0 for writing, LSB is 0 (Docum
entation 8.1.2.3)
  // Write data bytes sequentially (don't need to send address anymore)
  for (int I = 0; I < count; i++) {
    writeByte(values[i]);
  }
  digitalWrite(RC522_SS, HIGH);

  // Disable timer to turn off SPI clock
  RC522_SCLK_en = 0;
  // Leave chip select high for at least 100us
  delayMicroseconds(100);
}

/*
  Read a register of the RC522 multiple times (usually reading from the FIFO buff
er)
*/
void RC522_readRegs(byte address, int count, byte *values, byte rxAlign) {
  // Enable and reset timer to turn on SPI clock
  timerWrite(timer, 0);
  RC522_SCLK_en = 1;

  // Write address byte once
  digitalWrite(RC522_SS, LOW);
  count--;
  writeByte(((address << 1) | B10000000) & B11111110); // Send address, MSB is 1
for reading, LSB is 0 (Documentation 8.1.2.3)
  // Write address byte n-1 more times while reading returned byte each time
  for (int I = 0; I < count; i++) {
    // rxAlign means we only care about certain digits on the first byte
    if (I == 0 && rxAlign) {
      byte mask = (B11111111 << rxAlign) & B11111111;
      byte val = transferByte(((address << 1) | B10000000) & B11111110);
      values[0] = (values[0] & ~mask) | (val & mask);
    }
    else
      values[i] = transferByte(((address << 1) | B10000000) & B11111110);
  }
```

```
  // Read final (nth) byte. Don't care what to send on last byte
  values[count] = readByte();
  digitalWrite(RC522_SS, HIGH);
  // Disable timer to turn off SPI clock
  RC522_SCLK_en = 0;
  // Leave chip select high for at least 100us
  delayMicroseconds(100);
}

/*
  Performs a soft reset of the RC522 module (Documentation 9.3.1.2 & 10.3.1.10)
*/
void RC522_softReset() {
  RC522_writeReg(CommandReg, RC522_SoftReset);
  do
    delay(10);
  while (RC522_readReg(CommandReg) & (1 << 4)); // Loop until bit 4 is reset
}

/*
  Intialize RC522 module.
  This is referenced from the linked github library as there
  is no recommended 32print32aver32on sequence within the documentation
*/
void RC522_init() {
  // Configure digital pints
  pinMode(RC522_RST, OUTPUT);
  pinMode(RC522_IRQ, INPUT);
  pinMode(RC522_MISO, INPUT);
  pinMode(RC522_MOSI, OUTPUT);
  pinMode(RC522_SCLK, OUTPUT);
  pinMode(RC522_SS, OUTPUT);
  // RFID Reader
  digitalWrite(RC522_RST, HIGH);    // Active High enabled, reset on low
  digitalWrite(RC522_MOSI, LOW);
  digitalWrite(RC522_SCLK, LOW);
  digitalWrite(RC522_SS, HIGH);     // Active Low Chip Select

  // Intialize custom software SPI timer
  timer = timerBegin(0, 80, true);
  timerAttachInterrupt(timer, &intRoutine, true);
  timerAlarmWrite(timer, RC522_SCLK_interrupt_counter, true);
  timerAlarmEnable(timer);

  // Reset timer states
```

```
  RC522_SCLK_state = 0;
  rising_RC522_SCLK_edge = 0;
  falling_RC522_SCLK_edge = 0;
  RC522_SCLK_en = 0;

  // Hard reset of RC522
  digitalWrite(RC522_RST, LOW);
  delay(1);
  digitalWrite(RC522_RST, HIGH);
  delay(50);

  // Reset baud rates
  RC522_writeReg(TxModeReg, 0);
  RC522_writeReg(RxModeReg, 0);

  // Reset ModWidthReg
  RC522_writeReg(ModWidthReg, 0x26); // Default value

  // Create timeout timer for communicating with device
  RC522_writeReg(TmodeReg, B10000000); // Set Tauto bit to start timer automatica
lly after all transmiRC522_Ssions
  RC522_writeReg(TprescalerReg, B10101001); // 169 for a timer 33print33av of 40k
Hz, period=25us (Documentation 8.5)
  RC522_writeReg(TreloadReg_hi, B00000011);
  RC522_writeReg(TreloadReg_lo, B11101000); // 00000011 11101000 => 1000decimal,
for a timeout period of 25ms

  // RF Settings
  RC522_writeReg(TxASKReg, B01000000); // Set bit 7 to force a 100% ASK modulatio
n independent of the ModGsPReg register setting
  RC522_writeReg(ModeReg, B00111101); // Setting lower two bits to 01 for 0x6363
preset value for CRC coporceRC522_Ssor for CalcCRC command

  // Turn antenna on
  byte val = RC522_readReg(TxControlReg);
  RC522_writeReg(TxControlReg, val | B00000011); // Setting Tx2RFEn and Tx1RFEn
}

/*
  Check if a card is in range of the RC522 antenna.
  This is referenced from the linked github library as there
  is no recommended sequence within the documentation.
*/
bool RC522_isNewCardPresent() {
  byte bufferATQA[2];                   // buffer to store answer to request
```

```
  byte bufferSize = sizeof(bufferATQA);

  // Reset baud rates
  RC522_writeReg(TxModeReg, 0x0);
  RC522_writeReg(RxModeReg, 0x0);
  // Reset ModWidthReg
  RC522_writeReg(ModWidthReg, 0x26);

  StatusCode result = PICC_RequestA(bufferATQA, &bufferSize);
  return (result == STATUS_OK || result == STATUS_COLLISION);
}

/*
  Send a PICC_RequestA.
  This is referenced from the linked github library as there
  is no recommended sequence within the documentation.
*/
StatusCode PICC_RequestA(byte *bufferATQA, byte *bufferSize) {
  return PICC_REQA_or_WUPA(PICC_CMD_REQA, bufferATQA, bufferSize);
}

/*
  Configure and send a PICC_RequestA or WUPA.
  This is referenced from the linked github library as there
  is no recommended sequence within the documentation.
*/
StatusCode PICC_REQA_or_WUPA(byte command, byte *bufferATQA, byte *bufferSize) {
  byte validBits;
  StatusCode status;

  if (bufferATQA == nullptr || *bufferSize < 2) {
    return STATUS_NO_ROOM;
  }
  // All received bits will be cleared after a collision (Documentation 9.3.1.15)
  byte tmpReg = RC522_readReg(CollReg);
  tmpReg = tmpReg & B01111111;
  RC522_writeReg(CollReg, tmpReg);

  // For REQA and WUPA we need the short frame format - transmit only 7 bits of t
he last (and only) byte. TxLastBits = BitFramingReg[2..0]
  validBits = 7;
  status = RC522_TransceiveData(&command, 1, bufferATQA, bufferSize, &validBits);

  if (status != STATUS_OK)
    return status;
```

```
  if (*bufferSize != 2 || validBits != 0)  // ATQA must be exactly 16 bits.
    Return STATUS_ERROR;
  return STATUS_OK;
}

/*
  Configure and send a signal to the RFID card.
  This is referenced from the linked github library as there
  is no recommended sequence within the documentation.
*/
StatusCode RC522_TransceiveData(
                          byte *sendData,
                          // Pointer to the data to transfer to the FIFO.
                          Byte sendLen,     // Number of bytes to transfer to the
 FIFO.
                          Byte *backData,   // nullptr or pointer to buffer if da
ta should be read back after executing the command.
                          Byte *backLen,    // In: Max number of bytes to write t
o *backData. Out: The number of bytes returned.
                          Byte *validBits,  // In/Out: The number of valid bits i
n the last byte. 0 for 8 valid bits. Default nullptr.
                          Byte rxAlign,     // In: Defines the bit position in ba
ckData[0] for the fiRC522_RST bit received. Default 0.
                          Bool checkCRC     // In: True => The last two bytes of
the response is aRC522_Ssumed to be a CRC_A that must be validated.
                 ) {
  byte waitRC522_IRQ = 0x30; // RxRC522_IRQ and IdleRC522_IRQ
  return RC522_CommunicateWithPICC(RC522_Transceive, waitRC522_IRQ, sendData, 35p
rint, backData, backLen, validBits, rxAlign, checkCRC);
}

/*
  Communicate directly with the RFID card. This is used by many higher level func
tions.
  This is referenced from the linked github library as there
  is no recommended sequence within the documentation.
*/
StatusCode RC522_CommunicateWithPICC(
                            byte command,    // The command to execute. One of th
e PCD_Command enums.
                            Byte waitRC522_IRQ,    // The bits in the ComRC522_IR
QReg register that signals succeRC522_Ssful completion of the command.
                            Byte *sendData,  // Pointer to the data to transfer t
o the FIFO.
```

```
                            Byte sendLen,    // Number of bytes to transfer to th
e FIFO.
                            Byte *backData,  // nullptr or pointer to buffer if d
ata should be read back after executing the command.
                            Byte *backLen,   // In: Max number of bytes to write
to *backData. Out: The number of bytes returned.
                            Byte *validBits, // In/Out: The number of valid bits
in the last byte. 0 for 8 valid bits.
                            Byte rxAlign,    // In: Defines the bit position in b
ackData[0] for the fiRC522_RST bit received. Default 0.
                            Bool checkCRC    // In: True => The last two bytes of
 the response is aRC522_Ssumed to be a CRC_A that must be validated.
                   ) {
  // Prepare values for BitFraminReg
  byte txLastBits = validBits? *validBits : 0;
  byte bitFraming = (rxAlign << 4) + txLastBits; // RxAlign = BitFramingReg[6..4]
. TxLastBits = BitFramingReg[2..0]

  RC522_writeReg(CommandReg, RC522_Idle);           // Stop any current commands
  RC522_writeReg(ComIrqReg, B01111111);            // Clear all seven interrupt r
equest bits
  RC522_writeReg(FIFOLevelReg, B10000000);         // Set FlushBuffer to initiali
ze FIFO
  RC522_writeRegs(FIFODataReg, sendLen, sendData); // Write sendData to FIFO buff
er
  RC522_writeReg(BitFramingReg, bitFraming);       // Adjust bit framing
  RC522_writeReg(CommandReg, command);             // Send command
  if (command == RC522_Transceive) {               // Set StartSend bit to start
the transmiRC522_Ssion of data
    byte val = RC522_readReg(BitFramingReg);
    val = val | B10000000;
    RC522_writeReg(BitFramingReg, val);
  }

  // Wait for command to complete. In init, Tauto flag is set in TmodeReg which a
utomatically starts when the PCD stops transmitting.
  Long startTime = millis();
  while(1) {
    byte val = RC522_readReg(ComIrqReg);
    if (val & waitRC522_IRQ)       // One of the interrupts that signal succeRC52
2_SS has been set.
      Break;
    if (val & B00000001);    // Timer interrupt - nothing received in 25ms
      return STATUS_TIMEOUT;
    if (millis() - startTime > 100)
```

```
      return STATUS_TIMEOUT; // Nothing happened after 100ms, comms might not be
working
  }

  // Check for errors
  byte errorRegVal = RC522_readReg(ErrorReg);
  if (errorRegVal & B00010011) // Checking for BufferOvfl, ParityErr, or Protocol
Err
    return STATUS_ERROR;

  byte _validBits = 0;

  // Collect return data if needed
  if (backData && backLen) {
    byte n = RC522_readReg(FIFOLevelReg);
    if (n > *backLen)
      return STATUS_NO_ROOM;
    *backLen = n;
    RC522_readRegs(FIFODataReg, n, backData, rxAlign);  // Get return data from F
IFO buffer
    _validBits = RC522_readReg(ControlReg) & B00000111; // Find how many valid bi
ts in last byte, B000 means all are valid
    if (validBits)                                      // If not all bits are val
id, update how many are valid for caller function
      *validBits = _validBits;
  }

  // Check for collision
  if (errorRegVal & B00001000)
    return STATUS_COLLISION;

  // Validate CRC_A if needed
  if (backData && backLen && checkCRC) {
    // In this case a MIFARE ClaRC522_Ssic NAK is not OK.
    If (*backLen == 1 && _validBits == 4)
      return STATUS_MIFARE_NACK;
    // We need at least the CRC_A value and all 8 bits of the last byte must be r
eceived.
    If (*backLen < 2 || _validBits != 0)
      return STATUS_CRC_WRONG;
    // Verify CRC_A - do our own calculation and store the control in controlBuff
er.
    Byte controlBuffer[2];
    StatusCode status = RC522_CalculateCRC(&backData[0], *backLen - 2, &controlBu
ffer[0]);
```

```
    if (status != STATUS_OK) {
      return status;
    }
    if ((backData[*backLen - 2] != controlBuffer[0]) || (backData[*backLen - 1] !
= controlBuffer[1])) {
      return STATUS_CRC_WRONG;
    }
  }

  return STATUS_OK;
}

/*
  Obtain the 32-bit unique-ID of the RFID card.
  This is referenced from the linked github library as there
  is no recommended sequence within the documentation.
*/
bool PICC_getUID(byte *uid_bytes) {
  byte validBits = 0;
  bool uidComplete;
  bool useCascadeTag;
  byte cascadeLevel = 1;
  StatusCode result;
  byte count;
  byte checkBit;
  byte index;
  byte uidIndex;              // The fiRC522_RST index in uid_bytes that is used i
n the current Cascade Level
  int currentLevelKnownBits; // The number of known UID bits in the current Casca
de Level
  byte buffer[9];             // The SELECT/ANTICOLLISION commands uses a 7 byte s
tandard frame + 2 bytes CRC_A
  byte bufferUsed;            // The number of bytes used in the buffer, ie the nu
mber of bytes to transfer to the FIFO
  byte rxAlign;               // Used in BitFramingReg. Defines the bit position f
or the fiRC522_RST bit received
  byte txLastBits;            // Used in BitFramingReg. The number of valid bits i
n the last transmitted byte
  byte *responseBuffer;
  byte responseLength;

  if (validBits > 80)
    return false;

  // Reset ValuesAfterColl so all received bits will be cleared after a collision
```

```
  byte val = RC522_readReg(CollReg);
  val = val & B01111111;
  RC522_writeReg(CollReg, val);

  // Repeat Cascade Level loop until there is a complete UID
  uidComplete = false;
  while (!uidComplete) {
    // Set the Cascade Level in the SEL byte, find out if we need to use the Casc
ade Tag in byte 2.
    Switch (cascadeLevel) {
      case 1:
        buffer[0] = PICC_CMD_SEL_CL1;
        uidIndex = 0;
        useCascadeTag = validBits && 0; // When we know that the UID has more tha
n 4 bytes
        break;

      case 2:
        buffer[0] = PICC_CMD_SEL_CL2;
        uidIndex = 3;
        useCascadeTag = validBits && 0; // When we know that the UID has more tha
n 7 bytes
        break;

      case 3:
        buffer[0] = PICC_CMD_SEL_CL3;
        uidIndex = 6;
        useCascadeTag = false;            // Never used in CL3.
        Break;

      default:
        return false;
        break;
    }

    // How many UID bits are known in this Cascade Level?
    currentLevelKnownBits = validBits - (8 * uidIndex);
    if (currentLevelKnownBits < 0)
      currentLevelKnownBits = 0;

    // Copy the known bits from uid->uidByte[] to buffer[]
    index = 2; // destination index in buffer[]
    if (useCascadeTag)
      buffer[index++] = PICC_CMD_CT;
```

```
    byte bytesToCopy = currentLevelKnownBits / 8 + (currentLevelKnownBits % 8 ? 1
 : 0); // The number of bytes needed to represent the known bits for this level.
    If (bytesToCopy) {
      byte maxBytes = useCascadeTag ? 3 : 4; // Max 4 bytes in each Cascade Level
. Only 3 left if we use the Cascade Tag
      if (bytesToCopy > maxBytes) {
        bytesToCopy = maxBytes;
      }
      for (count = 0; count < bytesToCopy; count++) {
        buffer[index++] = uid_bytes[uidIndex + count];
      }
    }

    // Now that the data has been copied we need to include the 8 bits in CT in c
urrentLevelKnownBits
    if (useCascadeTag)
      currentLevelKnownBits += 8;

    // Repeat anti collision loop until we can transmit all UID bits + BCC and re
ceive a SAK - max 32 iterations.
    While (1) {
      // Find out how many bits and bytes to send and receive.
      If (currentLevelKnownBits >= 32) { // All UID bits in this Cascade Level ar
e known. This is a SELECT.
        //Serial.print(F("SELECT: currentLevelKnownBits=")); Serial.println(curre
ntLevelKnownBits, DEC);
        buffer[1] = 0x70; // NVB - Number of Valid Bits: Seven whole bytes
        // Calculate BCC - Block Check Character
        buffer[6] = buffer[2] ^ buffer[3] ^ buffer[4] ^ buffer[5];
        // Calculate CRC_A
        result = RC522_CalculateCRC(buffer, 7, &buffer[7]);
        if (result != STATUS_OK)
          return false;
        txLastBits    = 0; // 0 => All 8 bits are valid.
        bufferUsed    = 9;
        // Store response in the last 3 bytes of buffer (BCC and CRC_A - not need
ed after tx)
        responseBuffer  = &buffer[6];
        responseLength  = 3;
      }
      else { // This is an ANTICOLLISION.
        //Serial.print(F("ANTICOLLISION: currentLevelKnownBits=")); Serial.printl
n(currentLevelKnownBits, DEC);
        txLastBits    = currentLevelKnownBits % 8;
```

```
        count     = currentLevelKnownBits / 8;  // Number of whole bytes in the U
ID part.
        Index     = 2 + count;          // Number of whole bytes: SEL + NVB + UID
s
        buffer[1]   = (index << 4) + txLastBits;  // NVB - Number of Valid Bits
        bufferUsed    = index + (txLastBits ? 1 : 0);
        // Store response in the unused part of buffer
        responseBuffer  = &buffer[index];
        responseLength  = sizeof(buffer) - index;
      }

      // Set bit adjustments
      rxAlign = txLastBits;                     // Having a separate variable is
overkill. But it makes the next line easier to read.
      RC522_writeReg(BitFramingReg, (rxAlign << 4) + txLastBits); // RxAlign = Bi
tFramingReg[6..4]. TxLastBits = BitFramingReg[2..0]

      // Transmit the buffer and receive the response.
      Result = RC522_TransceiveData(buffer, bufferUsed, responseBuffer, &response
Length, &txLastBits, rxAlign);
      if (result == STATUS_COLLISION) { // More than one PICC in the field => col
lision.
        Byte valueOfCollReg = RC522_readReg(CollReg); // CollReg[7..0] bits are:
ValuesAfterColl reserved CollPosNotValid CollPos[4:0]
        if (valueOfCollReg & 0x20) { // CollPosNotValid
          return false; // Without a valid collision position we cannot continue
        }
        byte collisionPos = valueOfCollReg & 0x1F; // Values 0-
31, 0 means bit 32.
        If (collisionPos == 0) {
          collisionPos = 32;
        }
        if (collisionPos <= currentLevelKnownBits) { // No progreRC522_SS - shoul
d not happen
          return false;
        }
        // Choose the PICC with the bit set.
        currentLevelKnownBits = collisionPos;
        count     = currentLevelKnownBits % 8; // The bit to modify
        checkBit    = (currentLevelKnownBits - 1) % 8;
        index     = 1 + (currentLevelKnownBits / 8) + (count ? 1 : 0); // FiRC522
_RST byte is index 0.
        Buffer[index] |= (1 << checkBit);
      }
      else if (result != STATUS_OK) {
```

```
        return false;
      }
      else { // STATUS_OK
        uid_bytes[0] = buffer[2];
        uid_bytes[1] = buffer[3];
        uid_bytes[2] = buffer[4];
        uid_bytes[3] = buffer[5];
        return true;
      }
    }
  }
  return false;
}

/*
  Determine the CRC (Cyclic redundancy check).
  This is referenced from the linked github library as there
  is no recommended sequence within the documentation.
*/
StatusCode RC522_CalculateCRC(byte *data, byte length, byte *result) {
  RC522_writeReg(CommandReg, RC522_Idle);      // Stop current command
  RC522_writeReg(DivIrqReg, B00000100);       // Reset CRCRC522_IRQ interrupt req
uest bit
  RC522_writeReg(FIFOLevelReg, B10000000);    // Set FlushBuffer bit to clear FIF
O buffer
  RC522_writeRegs(FIFODataReg, length, data); // Send data to FIFO buffer
  RC522_writeReg(CommandReg, RC522_CalcCRC);   // Begin CRC calculation

  // Wait for CRC calculation to finish
  long startTime = millis();
  while(1) {
    byte val = RC522_readReg(DivIrqReg);
    // If CalcCRC command is active and all data is proceRC522_Ssed (Calculation
complete) (Documentation 9.3.1.6)
    if (val & B00000100) {
      RC522_writeReg(CommandReg, RC522_Idle);   // Stop calculating CRC for anyth
ing new in the FIFO buffer
      result[0] = RC522_readReg(CRCResultReg_lo);
      result[1] = RC522_readReg(CRCResultReg_hi);
      return STATUS_OK;
    }
    if (millis() - startTime > 100)
      return STATUS_TIMEOUT;
  }
}
```

RC522.h

```
/*
  RC522 Code Reference: https://github.com/miguelbalboa/RC522
  RC522 Documentation: https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf
*/

#include <Arduino.h>

#define RC522_RST  23  // Hard Reset (Active High)
#define RC522_IRQ  22  // Interrupt Request
#define RC522_MISO 21  // Master-in, Slave-out
#define RC522_MOSI 19  // Master-out, Slave-in
#define RC522_SCLK 18  // SPI Clock
#define RC522_SS   5   // Chip Select (Active Low)




/*
  Registers
  Documentation 9.3
*/
// Page 0: Command and Status
const byte CommandReg    = 0x1;
const byte ComlEnReg     = 0x2;
const byte DivlEnReg     = 0x3;
const byte ComIrqReg     = 0x4;
const byte DivIrqReg     = 0x5;
const byte ErrorReg      = 0x6;
const byte Status1Reg    = 0x7;
const byte Status2Reg    = 0x8;
const byte FIFODataReg   = 0x9;
const byte FIFOLevelReg  = 0xA;
const byte WaterLevelReg = 0xB;
const byte ControlReg    = 0xC;
const byte BitFramingReg = 0xD;
const byte CollReg       = 0xE;
// Page 1: Command
const byte ModeReg        = 0x11;
const byte TxModeReg      = 0x12;
const byte RxModeReg      = 0x13;
const byte TxControlReg   = 0x14;
const byte TxASKReg       = 0x15;
const byte TxSelReg       = 0x16;
```

```
const byte RxSelReg       = 0x17;
const byte RxThresholdReg = 0x18;
const byte DemodReg       = 0x19;
const byte MfTxReg        = 0x1C;
const byte MfRxReg        = 0x1D;
const byte SerialSpeedReg = 0x1F;
// Page 2: Configuration
const byte CRCResultReg_hi   = 0x21;
const byte CRCResultReg_lo   = 0x22;
const byte ModWidthReg       = 0x24;
const byte RFCfgReg          = 0x26;
const byte GsNReg            = 0x27;
const byte CWGsPReg          = 0x28;
const byte ModGsPReg         = 0x29;
const byte TmodeReg          = 0x2A;
const byte TprescalerReg     = 0x2B;
const byte TreloadReg_hi     = 0x2C;
const byte TreloadReg_lo     = 0x2D;
const byte TcounterValReg_hi = 0x2E;
const byte TcounterValReg_lo = 0x2F;
// Page 3: Test register:
const byte TestSel1Reg     = 0X31;
const byte TestSel2Reg     = 0X32;
const byte TestPinEnReg    = 0X33;
const byte TestPinValueReg = 0X34;
const byte TestBusReg      = 0X35;
const byte AutoTestReg     = 0X36;
const byte VersionReg      = 0X37;
const byte AnalogTestReg   = 0X38;
const byte TestDAC1Reg     = 0X39;
const byte TestDAC2Reg     = 0X3A;
const byte TestADCReg      = 0X3B;


/*
  Commands
  Documentation 10.3
*/
const byte RC522_Idle             = 0x0;
const byte RC522_Mem              = 0x1;
const byte RC522_GenerateRandomID = 0x2;
const byte RC522_CalcCRC          = 0x3;
const byte RC522_Transmit         = 0x4;
const byte RC522_NoCmdChange      = 0x7;
const byte RC522_Receive          = 0x8;
```

```
const byte RC522_Transceive       = 0xC;
const byte RC522_MFAuthent        = 0xE;
const byte RC522_SoftReset        = 0xF;


/*
  Return Status Codes
  Referenced from linked github library
*/
enum StatusCode : byte {
    STATUS_OK             ,       // Success
    STATUS_ERROR          ,       // Error in communication
    STATUS_COLLISION      ,       // Collission detected
    STATUS_TIMEOUT        ,       // Timeout in communication
    STATUS_NO_ROOM        ,       // A buffer is not big enough
    STATUS_INTERNAL_ERROR ,       // Internal error in the code
    STATUS_INVALID        ,       // Invalid argument
    STATUS_CRC_WRONG      ,       // The CRC_A does not match
    STATUS_MIFARE_NACK    = 0xff  // A MIFARE PICC responded with NAK.
};


/*
  Commands sent to the PICC.
  Referenced from linked github library
*/
enum PICC_Command : byte {
    // The commands used by the PCD to manage communication with several PICCs (I
SO 14443-3, Type A, section 6.4)
    PICC_CMD_REQA    = 0x26,    // REQuest command, Type A. Invites PICCs in stat
e IDLE to go to READY and prepare for anticollision or selection. 7 bit frame.
    PICC_CMD_WUPA    = 0x52,    // Wake-
UP command, Type A. Invites PICCs in state IDLE and HALT to go to READY(*) and pr
epare for anticollision or selection. 7 bit frame.
    PICC_CMD_CT      = 0x88,    // Cascade Tag. Not really a command, but used du
ring anti collision.
    PICC_CMD_SEL_CL1 = 0x93,    // Anti collision/Select, Cascade Level 1
    PICC_CMD_SEL_CL2 = 0x95,    // Anti collision/Select, Cascade Level 2
    PICC_CMD_SEL_CL3 = 0x97,    // Anti collision/Select, Cascade Level 3
    PICC_CMD_HLTA    = 0x50,    // HaLT command, Type A. Instructs an ACTIVE PICC
 to go to state HALT.
    PICC_CMD_RATS    = 0xE0,    // Request command for Answer To Reset.
};

void IRAM_ATTR intRoutine();
```

```
void writeByte(byte val);
byte readByte();
byte transferByte(byte writeVal);
void RC522_writeReg(byte address, byte val);
byte RC522_readReg(byte address);
void RC522_writeRegs(byte address, int count, byte *values);
void RC522_readRegs(byte address, int count, byte *values, byte rxAlign);
void RC522_softReset();
void RC522_init();
bool RC522_isNewCardPresent();
bool PICC_getUID(byte *uid);
StatusCode RC522_CalculateCRC(byte *data, byte length, byte *result);
StatusCode RC522_TransceiveData(byte *sendData, byte sendLen, byte *backData, byt
e *backLen, byte *validBits = nullptr, byte rxAlign = 0, bool checkCRC = false);
StatusCode RC522_CommunicateWithPICC(byte command, byte waitIRq, byte *sendData,
byte sendLen, byte *backData = nullptr, byte *backLen = nullptr, byte *validBits
= nullptr, byte rxAlign = 0, bool checkCRC = false);
StatusCode PICC_RequestA(byte *bufferATQA, byte *bufferSize);
StatusCode PICC_REQA_or_WUPA(byte command, byte *bufferATQA, byte *bufferSize);
```

Upload.cpp

```
/*
  Functions for connecting to WiFi and uploading info to google doc
  HTTP Example: https://github.com/espressif/arduino-esp32/blob/master/libraries/HTTPClient/examples/BasicHttpClient/BasicHttpClient.ino
*/

#include "Upload.h"

/*
  Certificate for HTTPS
*/
const char* root_ca  =\
"-----BEGIN CERTIFICATE-----\n"\
"MIIEkjCCA3qgAwIBAgIQCgFBQgAAAVOFc2oLheynCDANBgkqhkiG9w0BAQsFADA/\n"\
"MSQwIgYDVQQKExtEaWdpdGFsIFNpZ25hdHVyZSBUcnVzdCBDby4xFzAVBgNVBAMT\n"\
"DkRTVCBSb290IENBIFgzMB4XDTE2MDMxNzE2NDA0NloXDTIxMDMxNzE2NDA0Nlow\n"\
"SjELMAkGA1UEBhMCVVMxFjAUBgNVBAoTDUxldCdzIEVuY3J5cHQxIzAhBgNVBAMT\n"\
"GkxldCdzIEVuY3J5cHQgQXV0aG9yaXR5IFgzMIIBIjANBgkqhkiG9w0BAQEFAAOC\n"\
"AQ8AMIIBCgKCAQEAnNMM8FrlLke3cl03g7NoYzDq1zUmGSXhvb418XCSL7e4S0EF\n"\
"q6meNQhY7LeqxGiHC6PjdeTm86dicbp5gWAf15Gan/PQeGdxyGkOlZHP/uaZ6WA8\n"\
"SMx+yk13EiSdRxta67nsHjcAHJyse6cF6s5K671B5TaYucv9bTyWaN8jKkKQDIZ0\n"\
"Z8h/pZq4UmEUEz9l6YKHy9v6Dlb2honzhT+Xhq+w3Brvaw2VFn3EK6BlspkENnWA\n"\
"a6xK8xuQSXgvopZPKiAlKQTGdMDQMc2PMTiVFrqoM7hD8bEfwzB/onkxEz0tNvjj\n"\
"/Pizark5McWvxI0NHWQWM6r6hCm21AvA2H3DkwIDAQABo4IBfTCCAXkwEgYDVR0T\n"\
"AQH/BagwBgEB/wIBADAOBgNVHQ8Baf8EBAMCAYYwfwYIKwYBBQUHAQEEczBxMDIG\n"\
"CCsGAQUFBzABhiZodHRwOi8vaXNyZy50cnVzdGlkLm9jc3AuaWRlbnRydXN0LmNv\n"\
"bTA7BggrBgEFBQcwAoYvaHR0cDovL2FwcHMuaWRlbnRydXN0LmNvbS9yb290cy9k\n"\
"c3Ryb290Y2F4My5wN2MwHwYDVR0jBBgwFoAUxKexpHsscfrb4UuQdf/EFWCFiRAw\n"\
"VAYDVR0gBE0wSzAIBgZngQwBAgEwPwYLKwYBBAGC3xMBAQEwMDAuBggrBgEFBQcC\n"\
"ARYiaHR0cDovL2Nwcy5yb290LxgxLmxldHNlbmNyeXB0Lm9yZzA8BgNVHR8ENTAz\n"\
"MDGgL6AthitodHRwOi8vY3JsLmlkZW50cnVzdC5jb20vRFNUUk9PVENBWDNDUkwu\n"\
"Y3JsMB0GA1UdDgQWBBSoSmpjBH3duubRObemRWXv86jsoTANBgkqhkiG9w0BAQsF\n"\
"AAOCAQEA3TPXEfNjWDjdGBX7CVW+dla5cEilaUcne8IkCJLxWh9Keik3JHRRHGJo\n"\
"uM2VcGfl96S8TihRzZvoroed6ti6WqEBmtzw3Wodatg+VyOeph4Eypr/1wXKtx8/\n"\
"wApIvJSwtmVi4MFU5aMqrSDE6ea73Mj2tcMyo5jMd6jmeWUHK8so/joWUoHOUgwu\n"\
"X4Po1Qyz+3dszkDqMp4fklxBwXRsW10KXzPMTZ+sOPAveyxindmjkW8lGy+QsRlG\n"\
"PfZ+G6Z6h7mjem0Y+iWlkYcV4PIWL1iwBi8saCbGS5jN2p8M+X+Q7UNKEkROb3N6\n"\
"Koqkqm57TH2H3eDJAkSnh6/DNFu0Qg==\n"\
"-----END CERTIFICATE-----\n";

/*
  Two sets of SSIDs and passwords for use in two locations
*/
```

```
const char* ssid = "****";
const char* password = "****";

/*
  More network info
*/
const char *GscriptID = "AKfycbzYPSzUWES48WMIpE4-Hn_WeyyYr9WwoTXwO7ySPs8gAfrP-
UNJ";
const int httpsPort = 443;   // default HTTPS port
const char* host = "script.google.com";
// Format URL
String url = String("/macros/s/") + GscriptID + "/exec?";
WiFiClientSecure client;

/*
  Attempt to establish wifi connection
*/
bool WiFi_init() {
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  // Try for 5 seconds to connect to first network
  long startTime = millis();
  while ((WiFi.status() != WL_CONNECTED) && ((millis() - startTime) < 5000)) {
    Serial.print(".");
    delay(500);
  }

  // Try for 5 seconds to connect to second network if not connected to the first
  if (WiFi.status() != WL_CONNECTED) {
    WiFi.begin(ssid2, password2);
    long startTime = millis();
    while ((WiFi.status() != WL_CONNECTED) && ((millis() - startTime) < 5000)) {
      Serial.print(".");
      delay(500);
    }
  }

  // Return false if neither connected
  if (WiFi.status() != WL_CONNECTED) {
    return false;
  }


  Serial.println("\nWiFi connected");
```

```
  Serial.println(WiFi.localIP());

  // Connect to host
  Serial.print(String("Connecting to "));
  Serial.println(host);
  bool WiFiFlag = false;
  for (int I = 0 ; I < 5; i++) {
    int retval = client.connect(host, httpsPort);
    if (retval == 1)  {
      WiFiFlag = true;
      break;
    }
    else {
      Serial.println("Connection failed. Retrying...");
    }
  }
  Serial.println("Connection Status: " + String(client.connected()));
  Serial.flush();

  // If couldn't connect to host (no internet or google down), return false
  if (!WiFiFlag) {
    Serial.print("Could not connect to server: ");
    Serial.println(host);
    Serial.println("Exiting...");
    Serial.flush();
    return false;
  }

  // Everything is online
  return true;
}

/*
  Post data through HTTPS
*/
void postData(String UID_string, String is_valid) {
  HTTPClient http;

  // Format URL with data
  String urlFinal = String("https://") + host + url + "id=" + "Data" + "&UID=" +
UID_string + "&Valid=" + String(is_valid);
  Serial.println(urlFinal);
  Serial.println("Making a request");

  // Send HTTP get request
```

```
  http.begin(urlFinal, root_ca);
  int httpCode = http.GET();
  http.end();

  // Debug payload
  Serial.println(“: done”+httpCode);
}
```

Upload.h

```
/*
  Functions for connecting to WiFi and uploading info to google doc
  HTTP Example: https://github.com/espressif/arduino-esp32/blob/master/libraries/HTTPClient/examples/BasicHttpClient/BasicHttpClient.ino
*/

#include “Wifi.h”
#include <HTTPClient.h>

bool WiFi_init();
void postData(String UID_string, String is_valid);
```

Main.cpp

```
/*
  Tyler McGrew
  RC522 RFID reader implementation on ESP-32 with LCD display and peripherals
  https://github.com/tymcgrew/RFID_SmartLock
*/

#include <Arduino.h>
#include “esp_system.h”
#include “RC522.h”
#include “LCD.h”
#include <SPI.h>
#include “Upload.h”

/*
  Defining a few pins
*/
#define speaker 26
#define motorA 2
#define motorB 4

/*
  Is the WiFi connected?
*/
bool online = false;


/*
  Plays two short, high-pitched beeps on the speaker
*/
void welcomeTone() {
  // Triangle wave form /\/\/\/\/\/\ at a fairly high frequency, not sure the exa
ct pitch
  for (int repeats = 0; repeats < 500; repeats++) {
    for (int I = 0; I < 256; i+=20) {
      dacWrite(speaker, i);
    }
    for (int I = 254; I >= 1; i-=20) {
      dacWrite(speaker, i);
    }
  }

  // Wait a bit
  delay(50);
```

```
  // Second beep, same as the first
  for (int repeats = 0; repeats < 500; repeats++) {
    for (int I = 0; I < 256; i+=20) {
      dacWrite(speaker, i);
    }
    for (int I = 254; I >= 1; i-=20) {
      dacWrite(speaker, i);
    }
  }
}

/*
  Plays a longer, low-pitches tone on the speaker
*/
void invalidTone() {
  // Triangle wave form /\/\/\/\/\/\ at a lower frequency, not sure the exact pit
ch
  for (int repeats = 0; repeats < 200; repeats++) {
    for (int I = 0; I < 256; i+=1) {
      dacWrite(speaker, i);
      delayMicroseconds(10);
    }
  }
}

/*
  Turns a DC motor (H-
Bridge configuration) back and forth to simulate opening and closing a lock
*/
void openLock() {
  digitalWrite(motorA, HIGH);
  digitalWrite(motorB, LOW);
  delay(2000);

  digitalWrite(motorA, LOW);
  digitalWrite(motorB, LOW);
  delay(200);

  digitalWrite(motorA, LOW);
  digitalWrite(motorB, HIGH);
  delay(2000);

  digitalWrite(motorA, LOW);
  digitalWrite(motorB, LOW);
}
```

```
void setup() {
  Serial.begin(9600);

  // Intialize motor outputs
  pinMode(motorA, OUTPUT);
  pinMode(motorB, OUTPUT);
  digitalWrite(motorA, LOW);
  digitalWrite(motorB, LOW);

  delay(1000);

  // Intiazlize LCD
  LCD_init();
  // Initialize RFID Reader
  RC522_init();
  // Check if WiFi is available and connect if so
  online = WiFi_init();
}

void loop() {
  // Default screen: screen-saver aat 10Hz
  LCD_screenSaver();
  delay(100);

  // Check for card
  if (!RC522_isNewCardPresent())
    return;

  // Get card's UID
  byte uid_bytes[4];
  if (!PICC_getUID(uid_bytes))
    return;

  // Convert UID to int
  uint32_t uid = 0;
  for (int I = 0; I < 4; i++)
    uid += uid_bytes[i] << ((3-i) * 8);
  Serial.println(uid, HEX);

  // Convert to Hex String
  char hex_string[20];
  54print(hex_string, "%X", uid);

  // Check if UID is authorized
```

```
  if (uid == 0x14FF5B2B) {
    // Put the welcome screen on the LCD
    LCD_welcome(uid);
    // Play welcome tone
    welcomeTone();
    // Turn lock motor
    openLock();
    // Post data to google doc if system is online
    if (online)
      postData(String(hex_string), “YES”);
    else
      delay(2000);
  }
  else {
    // Put the invalid ID screen on the LCD
    LCD_invalid(uid);
    // Play the invalid tone
    invalidTone();
    // Post data to google doc if system is online
    if (online)
      postData(String(hex_string), “NO”);
    else
      delay(2000);
  }
}
```

References

[1]: https://www.makerfabs.com/rc522-rfid-reader-with-cards-kit-13.56mhz

[2]: https://www.sparkfun.com/products/10168

[3]: https://en.wikipedia.org/wiki/H_bridge

[4]: https://github.com/tymcgrew/RFID_SmartLock/tree/master/src

[5]: https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf

[6]: https://github.com/miguelbalboa/rfid/tree/master/src

[7]: https://www.sparkfun.com/datasheets/LCD/Monochrome/Nokia5110.pdf

[8]: https://github.com/sparkfun/GraphicLCD_Nokia_5110

[9]: http://en.radzio.dxp.pl/bitmap_converter/

[10]: https://medium.com/@shishir_dey/upload-data-to-google-sheet-with-an-esp32-and-some-scripting-2d8b0ccbc833

[11]: https://data.energizer.com/pdfs/max-eu-aaa.pdf

[12]: http://www.smartstripe.com/wp-content/uploads/2012/10/mifare-UIDs1.pdf