FarmBot RFID Integration

Senior Project

Final Report

June 2017

Written by: Laura Swart
Prepared for: Rory Aronson
Advisor: Jeffrey Gerfen

Table of Contents

Introduction

Summary: To assist the company FarmBot improve their product by adding RFID tracking to the FarmBot robot. RFID tracking will allow the robot to select and pick up tool heads without any user interference.

Client: Rory Aronson, CEO and founder of FarmBot. FarmBot's goal is to take the tediousness and monotony out of home farming and allow people to easily and effortlessly grow food. With FarmBot a user can drag and drop plants in a web app and create customized regimens based on the plant's needs. More information can be found on their website at https://farmbot.io/.

Background: FarmBot is a small farming robot that moves around on tracks and navigates a plot of land by imaging the plot in a cartesian based plane. There are a few essential parts of the system to be familiar with the bot, the tracks, the tools, the servos, and the tool bay. The bot is the part that moves around over the plot using the tracks and performs all of the robot's routines. It does this by picking up the correct tool head which is stored in the tool bay near the edge of the plot, the bot picks up these tools from the tool bay using strong magnets. The bot can select from different tools to accomplish a variety of tasks from planting seeds, watering, weeding, checking soil moisture and more. Some of the current limitations in this iteration of FarmBot is that plants cannot be taller than 3 ft or wider than the planter box since the bot has limited mobility. Additionally, the initial setup is complex for a user with a nontechnical background since many items need to be specified by the user beforehand rather than predetermined by "factory settings". The user must specify exactly where tools are located on

the coordinate grid for the bot to be able to find them and tools must be kept in their designated places.

Purpose: Currently, FarmBot tool heads are organized in a tool bay located at the edge of the plot with each head serving a specific purpose such as a seeder, watering attachment, weeder, etc. The issue is that the user must specify the exact x, y, and z coordinates in the plane which can be a tedious task. That requires the user to calibrate the location and keep track of where each tool head belongs. My project aims to help solve this problem by using RFID technology to allow the bot to keep track of the different tool heads without the user having to keep track of tools individually.

Research: One of the biggest issues with addition of RFID is the mechanical constraints. There are several things to consider when designing for FarmBot. The first being the size limitations for the RFID reader. The reader must be small enough to fit on the bottom face of the robot without interfering with its other tasks. Secondly, the reader and tags would be in the presence of strong magnetic fields so more testing is needed to determine how these will affect the readings. With this in mind, I conducted research on appropriate RFID readers for FarmBot and made several suggestions to my sponsor. The most promising suggestion is readers from sparkfun specifically the ID-20LA and the ID-12LA. The main difference between these two is the size and read range with the former having a slightly larger size and range. Both readers do require being within inches of the tag which advantage for FarmBot so as not to accidentally read tags from other tool heads. Another advantage of these readers is they do not need to be a part of a printed circuit board (PCB) so the average user has the ability to add on this feature to an existing FarmBot product.

Project Goals: The main goal of this project is to add an extra feature and level of autonomy to farmbot by integrating RFID into the robot and tool heads which is achieved by meeting the specific objectives listed below:

- Be able to uniquely identify each tool
- The reader can read the tag without any human interference
- The system can run autonomously after initial setup
- Tools can be placed in any position in the tool bay

Requirements

- The RFID reader and any supporting circuitry must be easily integratable with the current FarmBot robot, and not impede the robot's performance
- The reader needs to be able to communicate with the Arduino
- The end product must have a lifetime of greater than 2 years
- The system must store the tags and be accessible to other parts of the system
- Any exposed parts will need weatherproofing

Specifications

- The system shall have a PCB of size 2x2in
- The system shall have a serial voltage that is greater than +/- 5v
- Serial signal between the RFID reader and Arduino shall travel at least 8 ft in the wire
- The system shall withstand inclement weather
- The system shall not interfere with other operations of the robot

- The reader shall read a tag from a distance of 0 - 1.5 inches
- The PCB shall have a 5v VCC input to the board
- The PCB shall be grounded to the Arduino

Material Costs

Estimated Costs: Predicted cost of project before project completion.

Item	**Quantity**	**Cost**
Raspberry Pi	1	35.00
Arduino Mega	1	49.95
RFID Reader	1	34.95
RFID Tags	1	3.95
Total:		123.85

Bill of Materials: Final cost after completion of project.

Item	**Part Number**	**Quantity**	**Cost**
Raspberry Pi	RASPBERRY PI 3 MODEL B	1	35.00
Arduino Mega	Arduino Mega 2560	1	45.95
RFID Reader	ID-20LA (125 kHz)	1	34.95
RFID Reader	ID-12LA (125 kHz)	1	29.95
RFID Tags	SEN-09417	1	3.95
RS232 Converter Chip	MAX233CPP	2	4.89
Wires	LYSB017NEGTXC-ELECTRNCS	1	7.97
*PCB Boards	N/A	N/A	--
**3D Printing	N/A	N/A	--
Total:			166.67

*PCB board pricing varies based on quantity, number of layers, and manufacturer.

**3D Printing prices reflect estimate given by Digital Fabrication Lab at Cal Poly. Printing cost vary based on cost of materials.

Hardware

ID-20LA

Read range: 180mm

Dimensions: 38x40x7mm

Frequency: 125KHz

ID-12LA

Read range: 120mm

Dimensions: 25x26x9mm

Frequency: 125KHz

MAX233CPP

Description: Converts a 5v TTL/CMOS signal to a +/- 12v RS232 signal and vice versa. Used to boost the signal between the reader and the Arduino to create a more reliable connection.

Inputs:

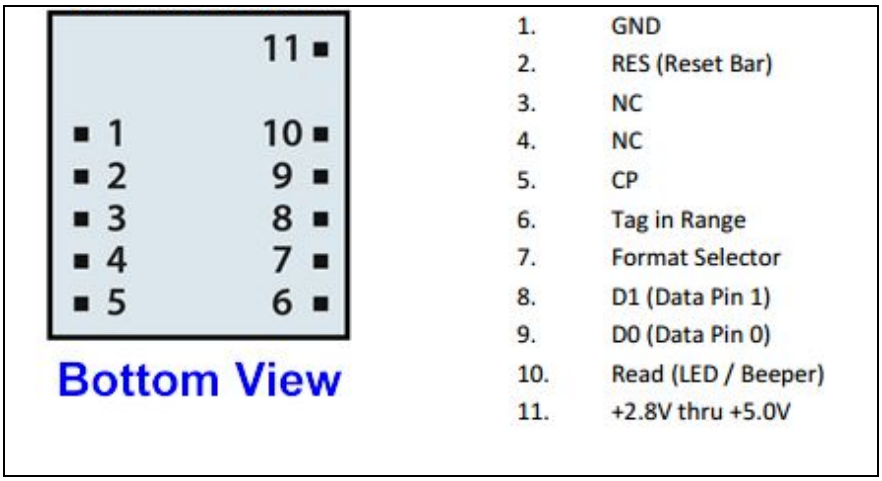
- 5v
- GND
- 9600 Baud signal or RS232

Outputs:

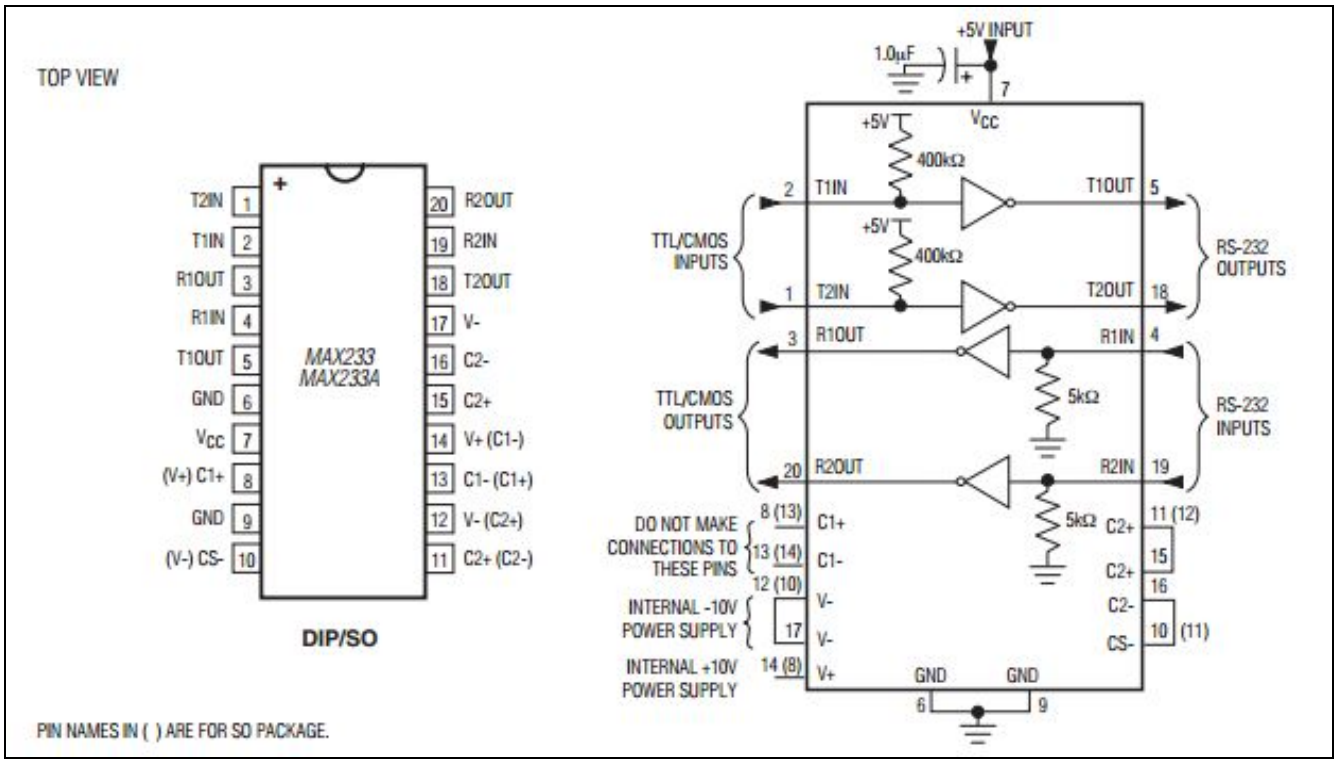
- TTL/CMOS 0-5v for RS232 input
- RS232 for TTL/CMOS input

Pinouts

ID-12LA ID-20LA



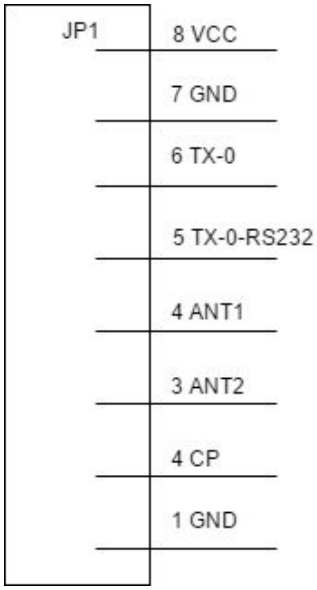
MAX233CPP



RFID USB Reader-v18

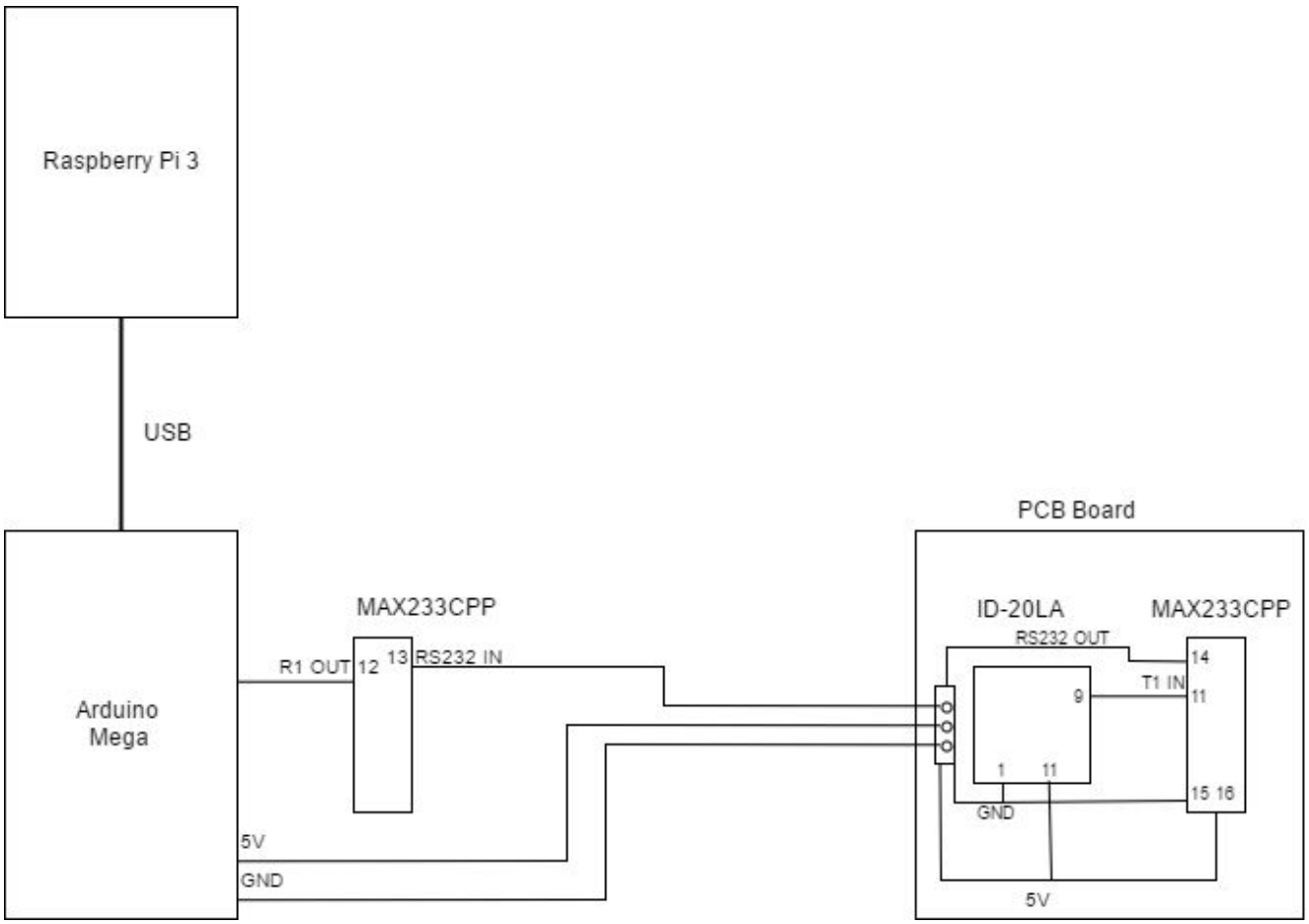
This part was used to interface the reader with the serial monitor on the laptop. This was useful in early stages of design for easy reading of tags and performing testing between the tag and reader.

Note: Used in prototype and demo only



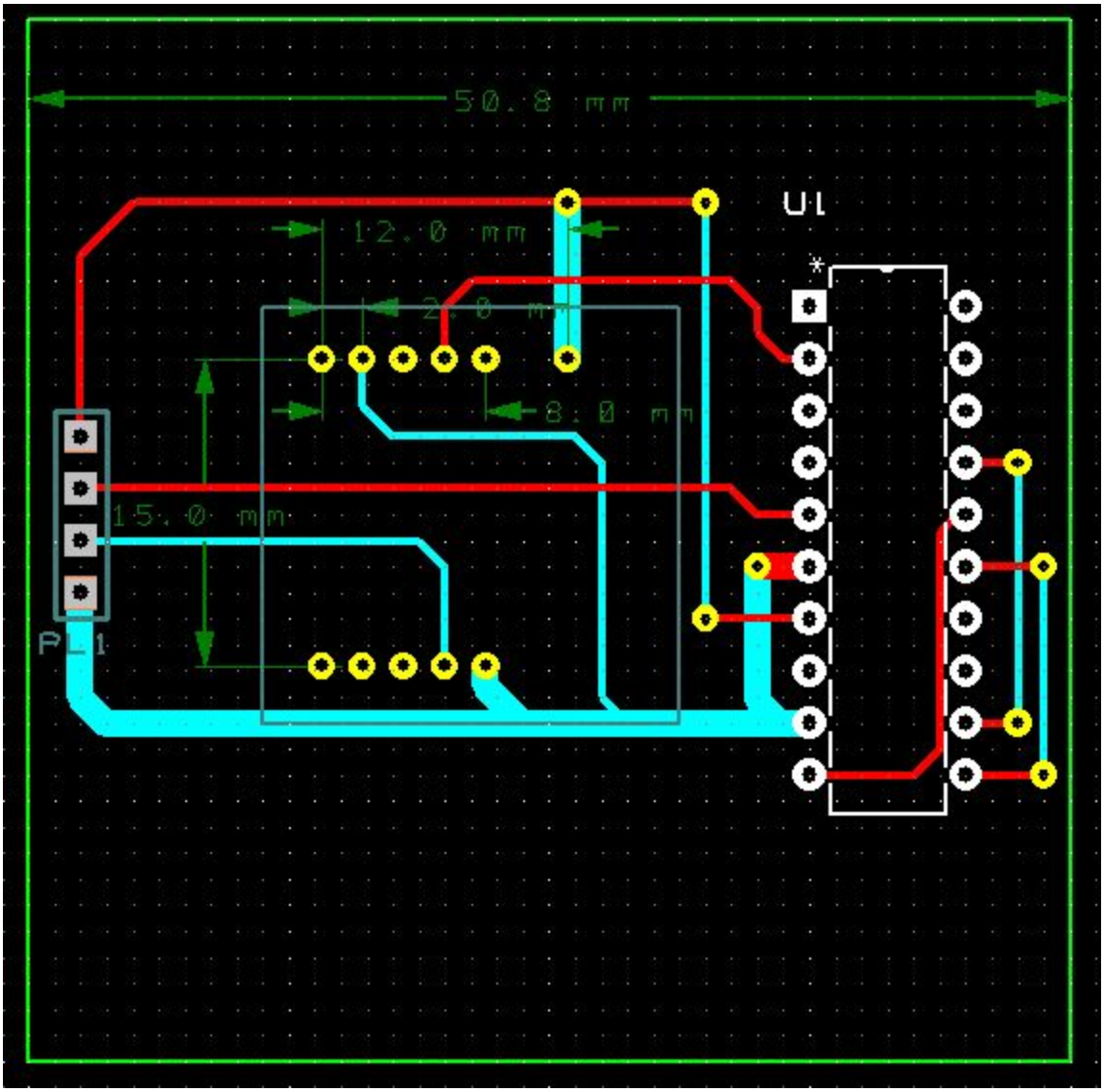
Schematic

The schematic shows the



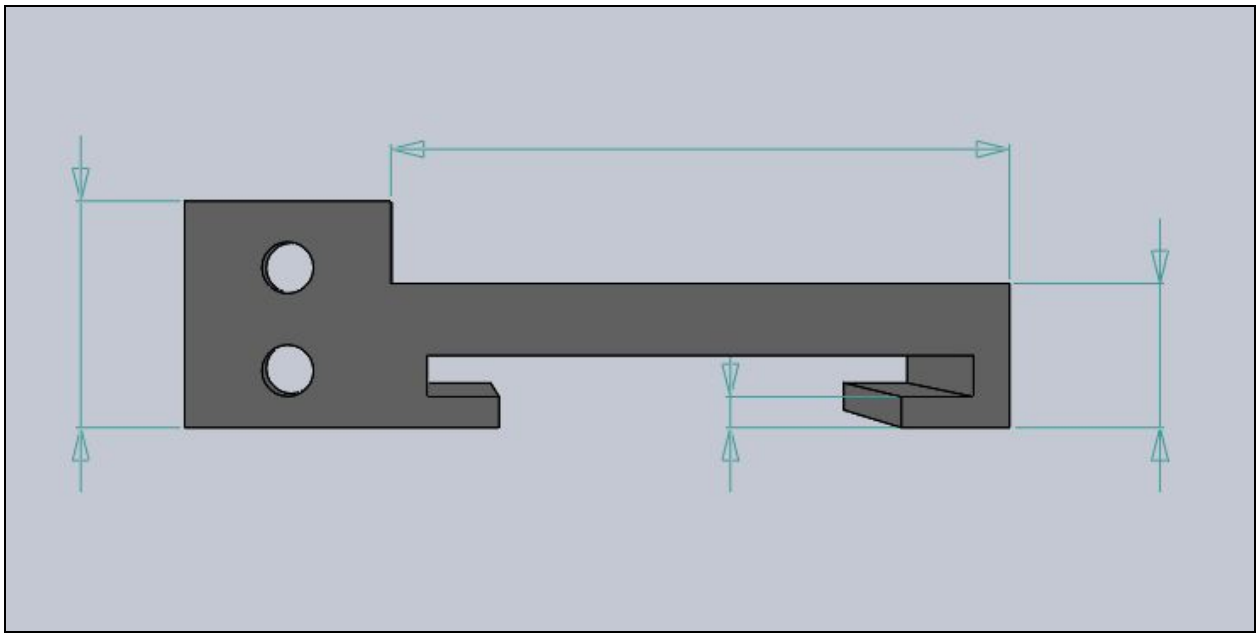
PCB Layout

The main goal of the PCB layout is to package the MAX233 and the RFID reader so that it is easy to use and simple to integrate into FarmBot. The challenging part of the layout was creating the custom part for the reader which has a unique pin configuration and spacing. A connector on the left bring ground, 5v to the board. A ground pour was intentionally not included because the circuit is not large and will work without one.

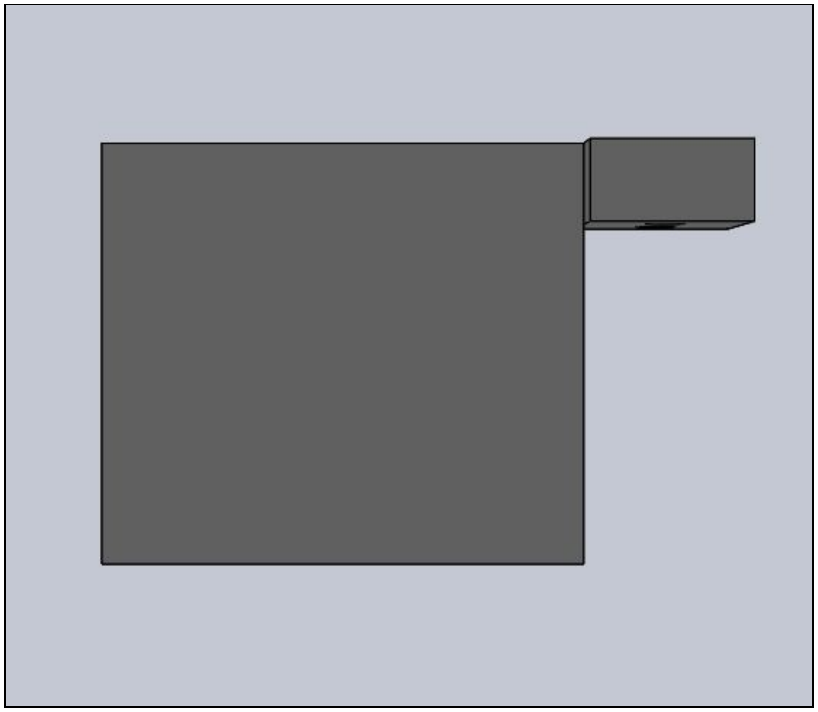


Solidworks Design

The PCB holder is an important part of the design. It is what allows the reader and MAX233 chip on the PCB board be easily attached to the robot by using screws though the two holes on the side. The holder is designed so that the PCB slides in and fits securely while also allowing the reader to face downward to read tags. This holder also protects the board from the elements.



PCB holder front view



PCB holder top view

Testing

To test the integrity, smoothness, and optimally of the system multiple tests were performed to ensure the highest standards Some of the early tests were designed to determine how the system should be designed given the outcome of these tests. The test descriptions cover the purpose of each test and how the test is performed.

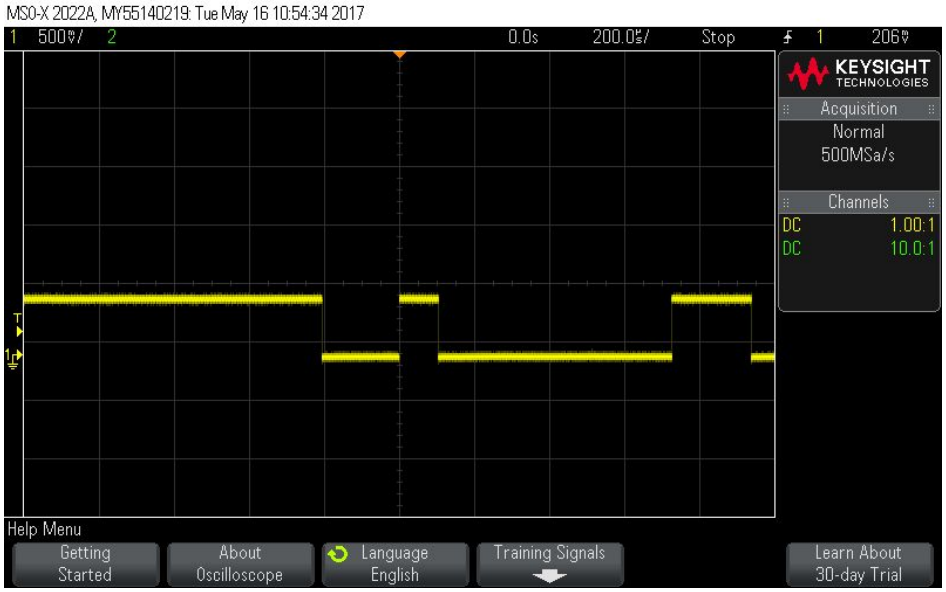
- **Reader to Arduino Communication:** This test is designed to simply ensure that both the Arduino and reader are working properly and a tag is read from the serial port. This will be tested by connecting the reader to the Arduino and connecting the Arduino to the computer through USB. The serial monitor on the laptop should be able to output the tag ID on the screen if it reads a tag.
- **Tag Read Distance:** Determine at what distance each reader can detect a tag for both readers. The tag will start at a distance greater than the maximum read distance for the reader and slowly decrease until the tag present light is on.

- **Read Tag in Presence of Metal:** Determine the effect of metal, if any, on the distance at which a tag is read. Repeat the same experiment for tag read distance, except metal is placed near the reader to simulate metal from the robot.
- **Read Tag in Presence of Magnets:** Determine the effect of magnets, if any, on the distance at which a tag is read. Repeat experiment for tag read distance, except a strong magnet as used for the toolhead is placed near the tag.
- **Serial Signal Distance:** Find if the serial signal outputted by the RFID reader has enough strength to be read over at least 6 feet of wire. This is necessary because Farmbot communicates between the robot and Arduino over 8-10 feet of wire and is expected to be scalable to larger distances.
- **RS232 Signal Conversion:** Check that the signal from the MAX233 chip is indeed converted to RS232. Wire serial signal from reader to MAX233 and view signal on the oscilloscope to verify that it is RS232.
- **RS232 to TTL/CMOS:** Ensure that the signal can be converted back to UART by viewing on the oscilloscope.
- **Tag Cache:** Test that the tag cache can add, lookup, and replace and delete tags. This is done by printing the output of each action, setting the cache to a small number and reading tags until the cache is full and ensuring that the correct tag is replaced.

Results

Reader to Arduino Communication: This test was successful, the serial monitor was able to readout the tag ID.

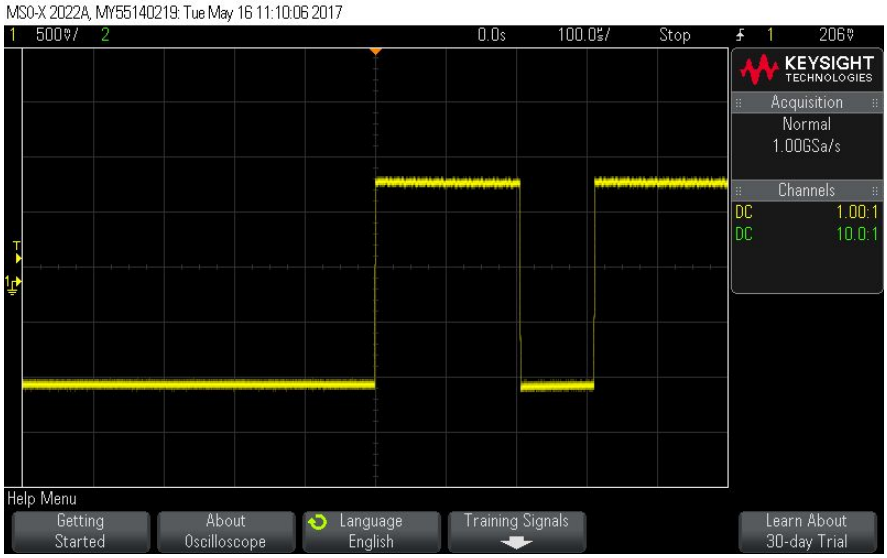
Serial monitor output after reading a tag.



Oscilloscope capture shows the serial output from the RFID reader.

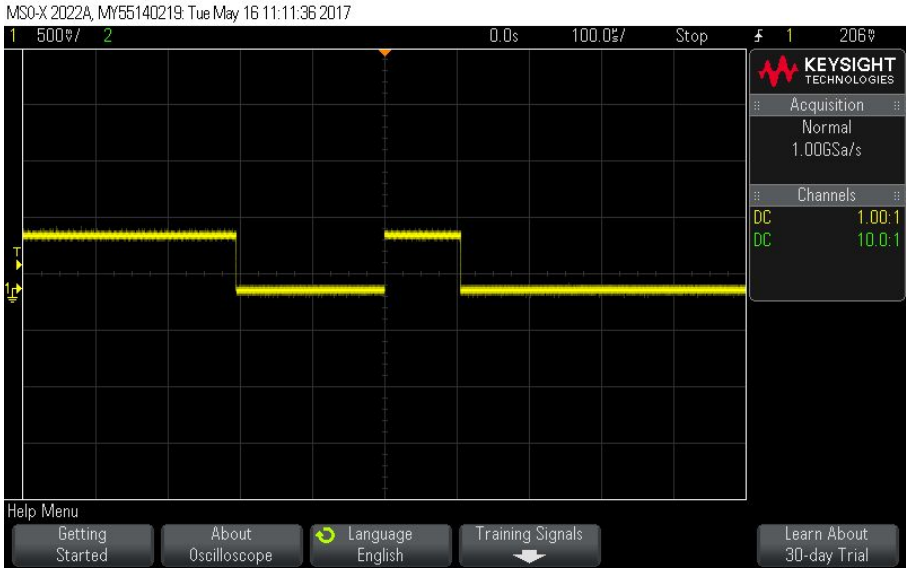
RS232 Signal Conversion: This test was not successful at first when measuring the output, there was a lot of noise. The issue was a pins 10 and 17 were left unconnected in the original configuration which is needed for the chip's internal -10v supply. Connecting these two pins produced a stable output.

Output before connecting pins 10 and 17.



Stable RS232 output.

RS232 to TTL/CMOS: After the serial signal is transported across several feet of wire from the reader to the Arduino it needs to be translated back to the correct voltage for the Arduino to read. The image below show the output of the MAX233CPP chip with a RS232 input.



Output from MAX233CPP with RS232 input.

Tag Read Distance in inches

Reader	**No Interference**	**Metal**	**Magnet**
ID20-LA	2.5	1	2.5
ID12-LA	1.25	.25	1.25

It is an established fact that metal interferes with the RFID reader's ability to read a tag, what was surprising is that magnets did not appear to have any effect on the reader's ability to read a tag.

Serial Signal Distance: After trying to read the serial signal through 6 feet of wire it was determined that the serial output from the reader alone would not be strong enough or reliable across a distance greater than 6 feet. This is due to the fact that the maximum voltage from the reader is .7v as seen in the data sheet.

Tag Cache: Testing the cache revealed that if the cache needed to replace a tag more times than the size of the cache then a new cache is created, meaning that all learned tags would

have to be relearned. After fixing this issue tags could be read, saved, and replaced without issue.

Code

The main purpose of the code is to read the tag ID in the form of serial data. The code reads the tag one byte at a time and rejects tags that are not exactly twelve characters. The reader has the option to either read the tags multiple times while in range of the tag or only once. This can be changed by including the “resetReader()” function which rereads the tag each time it is reset and can be reread at a set interval. The advantage of this is being able to correct a misread tag quicker. The second function of the code is to store the tags in a cache. The cache’s job is to compare tags, add new ones, and replace old tags when the cache is full using first in first out replacement.

Two versions of the code were drafted one for the demonstration which includes both tag reading and caching and the other separates the functions. Each part is meant to accomplish one task and be easy to interface with.

Future Work

Now that this project has reached its completion it is important to remember that this project is not complete just because it has met the stated goals. This project will still need to undergo revisions in order to improve and maximize its potential. Some suggestions to ensure the quality and longevity of this project are to test the system fully integrated into FarmBot, to incorporate the Raspberry Pi by writing code for the Pi to enable it to read in the tag IDs from the Arduino,

and writing a method to allow the Pi to execute instructions based on the tags read. Hopefully in the next iteration of FarmBot RFID integration RFID tracking will be available to the public.

Appendix

Appendix A: Code

Demo Code:

```
#include <string.h>
//#include <string>
#define CACHESIZE 3
#define TAGSIZE 13
int resetPin = 13;
int fifo;
char tagCache[CACHESIZE][TAGSIZE];
void clearTag(char str[]);
void resetReader();
void createCache();
boolean compareTag(char a[], int i);

void setup(){
  Serial.begin(9600);
  Serial.println("Begin");
  pinMode(resetPin, OUTPUT);
  digitalWrite(resetPin, HIGH);
  createCache();
}

void createCache() {
   int i;
   Serial.print("Creating cache size ");
   Serial.println(CACHESIZE);
   for (i = 0; i < CACHESIZE; i++) {
      tagCache[i][0] = '\0'; //initialize all spaces to '\0'
   }
   fifo = 0;
}

void loop(){

  char tagString[TAGSIZE];
  int index = 0;
  boolean reading = false;
```

```
  while(Serial.available()){

    int readByte = Serial.read(); //read next available byte

    if(readByte == 2) reading = true; //begining of tag
    if(readByte == 3) reading = false; //end of tag

    if(reading && readByte != 2 && readByte != 10 && readByte != 13){
      //store the tag
      tagString[index] = readByte;
      index ++;
    }
  }
  tagString[13] = '\0';
  checkTag(tagString); //Check if it is a match
  clearTag(tagString); //Clear the char of all value
  delay(150);
  //resetReader(); //reset the RFID reader
}

void printCache() {
  int i;
  Serial.println("Printing cache: ");
  for (i = 0; i < CACHESIZE; i++) {
     Serial.println((char *)tagCache[i]);
  }
}

void addToCache(char tag[]) {
   int i;
   char temp[TAGSIZE];
   for (i = 0; i < CACHESIZE; i++) {
      if (tagCache[i][0] == '\0') {
         strncpy(tagCache[i],tag, 13); //if space is empty add tag here
         Serial.print("Adding tag to cache: ");
         Serial.println(tag);
      //   return 1; //cache not full
         return;
      }
   }
   //cache is full
   //use FIFO replacement
   Serial.print("Cache is full, replacing tag: ");
   tagCache[fifo][12] = '\0';
   strncpy(temp, tagCache[fifo], 13);
   Serial.print(temp);
   Serial.print(" with tag: ");
```

```
   Serial.print(tag);
   Serial.print(" in location: ");
   Serial.println(fifo);
   strncpy(tagCache[fifo],tag, 13);
   fifo++;
   if (fifo >= CACHESIZE) {
      fifo = 0;
   }
}

void compareToCache(char tag[]) {
   int i;
   for (i = 0; i < CACHESIZE; i++) {
      if (compareTag(tag, i)) {
        //tag found
        Serial.print("Tag found: ");
        Serial.println(tag);
        return;
      }
   }
   addToCache(tag);
}

void checkTag(char tag[]){
//Check the read tag against known tags

  if(strlen(tag) == 0){
    return; //empty
  }
  if(strlen(tag) < 12) {
    Serial.print("Incomplete tag read ");
    Serial.println(tag);
    return;
  }
  compareToCache(tag);
//  printCache();
}

void lightLED(int pin){
  Serial.println(pin);

  digitalWrite(pin, HIGH);
  delay(250);
  digitalWrite(pin, LOW);
}

void resetReader(){
  digitalWrite(resetPin, LOW);
```

```
  digitalWrite(resetPin, HIGH);
  delay(150);
}

void clearTag(char str[]){
//clear the char array by filling with 0
  for(int i = 0; i < strlen(str); i++){
    str[i] = 0;
  }
}

boolean compareTag(char toComp[], int index){
//compare two values to see if same,

  if(strlen(toComp) == 0) return false; //empty

  for(int i = 0; i < 12; i++){
    if(toComp[i] != tagCache[index][i])
       return false;
  }

  return true; //no mismatches
}
```

Read Tag:

```
#include <string.h>
//#include <string>
#define TAGSIZE 13
int resetPin = 13;

void clearTag(char str[]);
void resetReader();

void setup(){
  Serial.begin(9600);
  Serial.println("Begin");
  pinMode(resetPin, OUTPUT);
  digitalWrite(resetPin, HIGH);
}

void loop(){

  char tagString[TAGSIZE];
  int index = 0;
  boolean reading = false;

  while(Serial.available()){
```

```
    int readByte = Serial.read(); //read next available byte

    if(readByte == 2) reading = true; //begining of tag
    if(readByte == 3) reading = false; //end of tag

    if(reading && readByte != 2 && readByte != 10 && readByte != 13){
      //store the tag
      tagString[index] = readByte;
      index ++;
    }
  }
  tagString[13] = '\0';
  return tagString;
  //resetReader(); //reset the RFID reader
}

void resetReader(){
  digitalWrite(resetPin, LOW);
  digitalWrite(resetPin, HIGH);
  delay(150);
}
```

Tag Cache:

```
#include <string.h>
//#include <string>
#define CACHESIZE 3
#define TAGSIZE 13
int resetPin = 13;
int fifo;
char tagCache[CACHESIZE][TAGSIZE];

void clearTag(char str[]);
void resetReader();
void createCache();
boolean compareTag(char a[], int i);

void setup(){
  Serial.begin(9600);
  pinMode(resetPin, OUTPUT);
  digitalWrite(resetPin, HIGH);
  createCache();
}

void createCache() {
   int i;
   for (i = 0; i < CACHESIZE; i++) {
```

```
      tagCache[i][0] = '\0'; //initialize all spaces to '\0'
   }
   fifo = 0;
}

void printCache() {
  int i;
  Serial.println("Printing cache: ");
  for (i = 0; i < CACHESIZE; i++) {
     Serial.println((char *)tagCache[i]);
  }
}

void addToCache(char tag[]) {
   int i;
   char temp[TAGSIZE];
   for (i = 0; i < CACHESIZE; i++) {
      if (tagCache[i][0] == '\0') {
         strncpy(tagCache[i],tag, 13); //if space is empty add tag here
         return;
      }
   }
   //cache is full
   //use FIFO replacement
   tagCache[fifo][12] = '\0';
   strncpy(temp, tagCache[fifo], 13);
   strncpy(tagCache[fifo],tag, 13);
   fifo++;
   if (fifo >= CACHESIZE) {
      fifo = 0;
   }
}

void compareToCache(char tag[]) {
   int i;
   for (i = 0; i < CACHESIZE; i++) {
      if (compareTag(tag, i)) {
        //tag found
        Serial.print(tag);
        return;
      }
   }
   addToCache(tag);
}

void checkTag(char tag[]){
//Check the read tag against known tags
```

```
  if(strlen(tag) == 0){
    return; //empty
  }
  if(strlen(tag) < 12) {
    return; //tag too short
  }
  compareToCache(tag);
//  printCache();
}

void resetReader(){
  digitalWrite(resetPin, LOW);
  digitalWrite(resetPin, HIGH);
  delay(150);
}

void clearTag(char str[]){
//clear the char array by filling with 0
  for(int i = 0; i < strlen(str); i++){
    str[i] = 0;
  }
}

boolean compareTag(char toComp[], int index){
//compare two values to see if same,

  if(strlen(toComp) == 0) return false; //empty

  for(int i = 0; i < 12; i++){
    if(toComp[i] != tagCache[index][i])
       return false;
  }

  return true; //no mismatches
}
```

Appendix B: Links to Datasheets

ID12-LA/ID20-LA:

https://cdn.sparkfun.com/datasheets/Sensors/ID/ID-2LA,%20ID-12LA,%20ID-20LA2013-4-10.pdf

MAX233CPP:

http://www.mouser.com/ds/2/256/MAX220-MAX249-67423.pdf

ATmega:

http://www.atmel.com/Images/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf

Arduino Mega:

https://www.arduino.cc/en/Main/ArduinoBoardMega2560