# DIGITIZING THE RESTAURANT

By

Aman Kishore
Jun Woo Seo
Patrick Moore

Final Report for ECE 445, Senior Design, Fall 2020
TA: Sophie Liu

09 December 2020
Project No. 36

## Abstract

This project brings new technology to the restaurant space that has previously only existed by pen and paper. Through a series of tools, some of which are still in development, restauranters can leverage data aggregation and analytics to better manage and promote their business. By utilizing commonplace technologies and designing minimalistic tableside units, any restaurant can adopt this solution at an affordable price. Additionally, this project packages together the new revenue management solutions alongside commonplace tools, such as POS and transaction services. This ensures that this project can be easily adopted by both new and established restaurants. These services also provide an ease of communication between restaurants and diners, which is beneficial to both parties.

# Contents

# 1. Introduction

## 1.1 Problem Statement

The restaurant industry is currently in crisis; the rise of COVID-19 will make managing a business that relies on maintaining a social setting harder than ever this upcoming winter. Recently, restaurants have adopted, forcibly or not, logistic and delivery services that have been gouging them mercilessly. More than ever, restaurants are willing and ready to adopt technology that can lessen the burden these companies pose on them. Additionally, restauranters are eager to adopt new technologies that will assist in revenue management.

## 1.2 Functionality and Benefits

### 1.2.1 Recording Interaction Data

Both diners and restauranters benefit from an accumulation of interaction data. Diners will be able to view previous transactions as well as select from what will be recorded as "popular" restaurants and meals. Restauranters, on the other hand, will be able to manage total income and deduct expenses based on the aggregation of all of their restaurant's interaction data. Additionally, this will act as historical data which will be used by models within the revenue management tools.

### 1.2.2 Reservation and Point-of-Sale Systems

Both reservation and POS services are commonplace within the restaurant industry. Packaging these tools with new technologies provides simplicity for new restauranters who are not trying to invest in a variety of technology providers. Furthermore, the use of these tools is not mandatory as to be inclusive to established restauranters with their own familiar products. What makes our system stand out from others are the customer-facing tools. Utilizing an in-app tool, customers will be able to select their own table and order food from their mobile devices. Additionally, the use of historical data will allow customers to view the modeled wait time on each table. The interactions between servers and the on-table units also provides peace-of-mind to the customer as the app indicates when a table has been sanitized.

### 1.2.3 Table Communication Units

Many people have an adversity against downloading new applications to their personal devices. To accommodate these people, this solution incorporates a design for lightweight table units. Each unit allows the customer to communicate simple wants with the restaurant. The server can then use an unintrusive RFID authentication card to comminate with the system that a request has been completed. Servers may also indicate that tables have been sanitized by tapping their RFID card on the unit.

### 1.2.4 Revenue Management Tools

By leveraging aggreged data, restauranters are provided with various solutions to improve business. One such tool will provide a summary of the day's income, expenses, and revenue. With the use of historical data, restaurants can be provided with a massive amount of modeled data. The most significant models will include projected revenue, occupancy, and ingredient use. These tools will allow for restauranters to better manage rushes, stock, and expenses.

### 1.2.5 Data Storage, User Management, and Deployments

Scalability is the primary objective of a growing application. Through the continued integration and use of these developer tools, user privacy is ensured, customization and tool options for restauranters can continue to grow, and future developers will have the luxury of data recovery, all while maintaining minimal performance and storage costs.

## 2. Design

The intention of our hardware design is to be lightweight: simple with no intensive processes. As seen in Figure 1, this is achieved by breaking the design into 4 simple parts, 3 of which require hardware. By accomplishing this goal, the product is both reliable and affordable. See figure 2 for the physical design of the casing which houses the hardware.

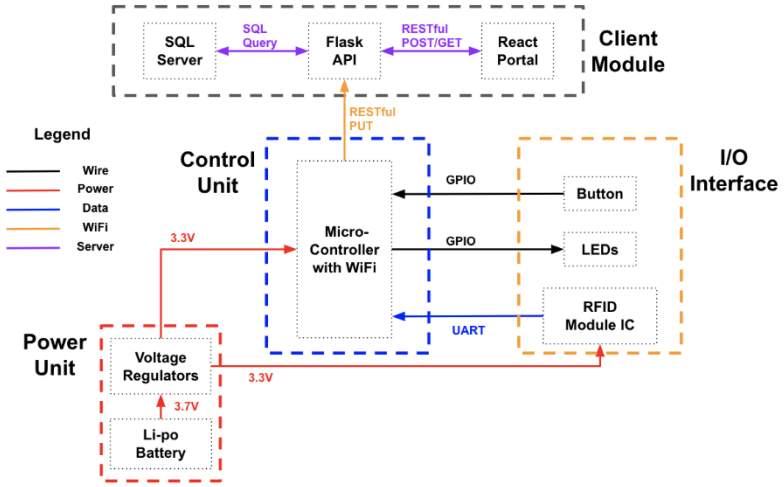


Figure 1 Block Diagram

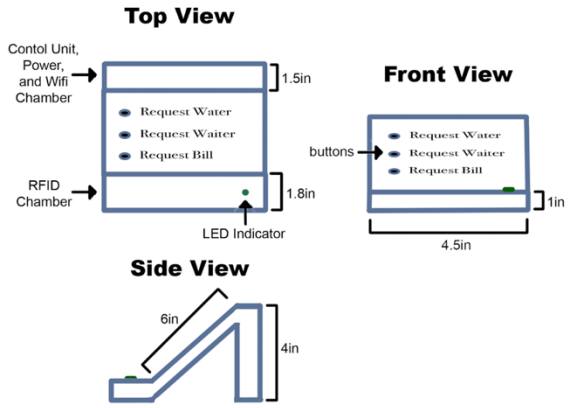


Figure 2 Physical Design

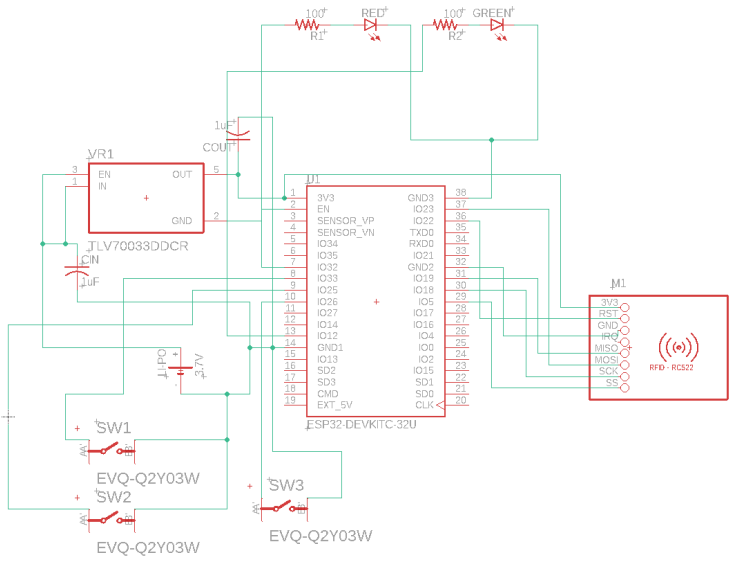


Figure 3 Circuit Schematic

## 2.1 Power Module

A power supply is required to keep the RFID and microcontroller functioning remotely from a host computer. An AC battery and two voltage regulators are used to provide constant power to both the RFID module and microcontroller. Both regulators output 3.3V: the required voltage for both modules.

### 2.1.1 Li-ion battery

The Li-ion battery supplies power to the system. The lithium polymer battery provides a constant voltage of 3.7V with a capacity of 1600mAh. According to their respective datasheets, the microcontroller requires 80mA[1] and the RFID requires 30mA.[2] Since the total current consumption for RFID and microcontroller is 110mA (see equation 1), the total capacity of the battery needs to be at least 1320mAh to run for 12 hours. Thus, the battery is more than sufficient to supply power for 12 hours. Should a restaurant be open for three meal services (breakfast, lunch, and dinner), the restauranter shall be instructed that the battery must be replaced halfway through the day to ensure no loss of power. Else, 12 hours of power is sufficient for 2 meal services.

$$I_{RFID} + I_{MCU} = I_{total} \quad (1)$$

### 2.1.2 Voltage regulators

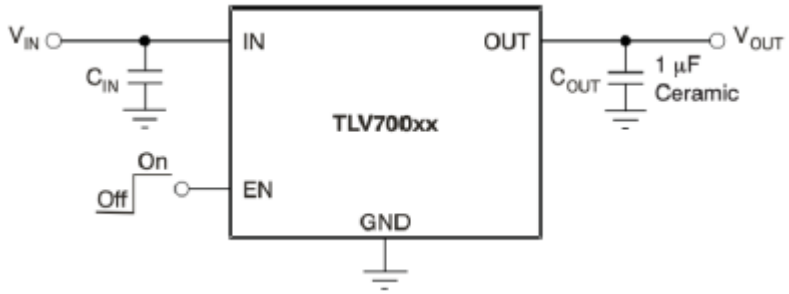


Figure 4 Typical Application Circuit for Voltage Regulator

To output 3.3V and 110mA to the RFID and microcontroller modules constantly, TLV70033 (the voltage regulator) requires an input voltage of 3.7V. The lithium polymer battery supplies this required voltage to both regulators in parallel. TLV70033 is a low dropout liner regulator which also requires the capacitors $C_{in}$ and $C_{out}$ (see figure 3). According to TLV70033's datasheet, the effective output capacitance is generally 1uF or 0.1uF for the stable device. (See figure 3 for full layout.) The voltage dropout of TLV70033 at 100mA is 85mV[3] and it is satisfied the condition (see equation 2).

$$V_{in} - V_{DO} \geq V_{OUT}(\pm 10\%) \quad (2)$$

## 2.2 I/O Interface

Every unit contains an RFID reader which acts as an unintrusive means to indicate when a table has been sanitized or served. The RFID reader acts as an authentication device which is meant to indicate that a server, who possess an RFID card, has fulfilled a customer's request. Reading a proper RFID card sends a signal to the micro controller, indicating the presence of a card. The microcontroller then clears the customer's request from the SQL server or, in the case of no active request, will indicate the table is sanitized.

### 2.2.1 RFID Reader

The RFID reader, RC522, is contained within the on-table unit. RC522 can recognize the unique RFID cards that are in the possession of wait-staff and, when read, will indicate the table has been sanitized or served. When RC522 recognizes a tag, it triggers an LED indicator (which is placed on the circuit) and sends a signal to the microcontroller. The microcontroller then modifies the SQL server appropriately.

### 2.2.2 RFID Tag

Each staff member will be equipped with an RFID tag (card). The tag acts as an unobtrusive and swift way to clear notifications from the on-table unit. The RFID reader only accepts registered cards' unique identification number.

## 2.3 POS/Customer Interface

The Customer Interface's primary use is to communicate a customer's needs with the restaurant's staff. This is accomplished through a series of labeled buttons (see Figure 2) which,

when pressed, causes the control unit to send a signal which populates a data-server with the appropriate data. This data-server is monitored by the staff via web-portal. This is the primary means of communication for the customer to send requests to the staff. Requests made through both the app and the table units are processed by the backend and stored within the SQL database.

### 2.3.1 Client Module

The Client Module (i.e., The Restaurant Module) specifically includes the SQL database, the server which processes incoming data, and the server which provides a user interface. The Flask API acts as the bridge between the database and customer or staff. The customer's request, which are communicated through either the app or the table unit, are processed by the Flask API. This process includes timestamping the request and attaching an ID which corresponds with the table making said request. The restaurant, through use of a web-app, constantly fetches information from the API. An authentication process within the API ensures the integrity of the SQL database before returning data to authorized accounts. When a table status changes, the app recognizes this and sends a notification to the restaurant's app. Staff may modify the status of any table through use of their RFID card or application access.

### 2.3.2 POS

Point-of-Sale systems are commonplace in the restaurant industry and one of the most important tools a restaurant will invest in; a revenue management product would be incomplete without some sort of POS functionality. The current implementation of the POS allows for customers to order and purchase their own food through the application, which will then send the order to the staff. Currently, the tool is a work in progress as the payment methods has yet to be implemented.

### 2.3.3 Analytics

Each restaurant relies on historical customer data to best serve the needs of the customer. By utilizing cron jobs and stored interaction data, a script frequently analyzes the data collected by the application. This can give the restaurant insight on how to better serve their customers by revealing details like average time spent at a table, favorite meals, loyal customers, etc. The GCP cron job toolkit allowed for a series of scripts to continuously pull data, update the analytics, and then update the front-end application. This, in turn, would give the restaurant up to date information for all their analytics.

### 2.3.4 Expanding Toolkits

The data is stored in a series SQL tables which are hosted on GCP. The frontend application uses React: a very common framework built for state and user management. The Flask backend contains all API and authentication processes. Both the frontend and backend have various deployments hosted by Netlify and Heroku respectively. Both Netlify and Heroku have impressive Continuous Integration (CI) toolkits which bolster a developer's ability to troubleshoot once properly initialized. By leveraging CI tools, the code and database have become modularized. This allows for developers to release updates after thorough testing and quality assurance. In turn, restaurants will be sure that improvements will be frequent, and all

testing practices will have no harmful impact. By establishing CI tools early in development, the path to swift functional growth has been paved.

## 2.4 Control Unit

The Control Unit is the bridge between the I/O Interface and Client Module. It will interpret signals from the RFID reader and buttons and send data to Flask API through the Wi-Fi.

### 2.4.1 Microcontroller

The ESP32-WROOM-32 acts as the microcontroller for the table unit. This microcontroller was chosen because of its Wi-Fi integrated circuit. The Wi-Fi IC allows for the microcontroller to send data to the Flask API hosted on the Heroku. There are a series of API commands in the flash storage of the microcontroller which will be called when certain conditions are met. For example, when the button is pressed, the microcontroller receives the data via GPIO. Then, data is sent back to LEDs via GPIO, one of the three LEDs turns on (green), and an API call to change the unit's table status is made.

# 3. Design Verification

## 3.1 Power Module

### 3.1.1 Li-ion battery

To measure the output voltage of battery, a multimeter was placed across the positive and negative terminals. The reading on the multimeter is 3.89V and it is satisfied the requirement as the Table 1 in Appendix A.


Figure 5 The Output Voltage of Battery

### 3.1.2 Voltage regulators

To measure the output voltage of battery, a multimeter was placed across the positive and negative terminals. The reading on the multimeter is 3.28V and it is satisfied the requirement as the Table 2 in Appendix A.


Figure 6 The Output Voltage Regulator

## 3.2 I/O Interface

### 3.2.1 RFID Reader

To measure the frequency of RFID, an oscilloscope was placed across the clock and the ground. The reading on the oscilloscope is 13.78MHz and it is satisfied the requirement as the Table 3 in Appendix A.

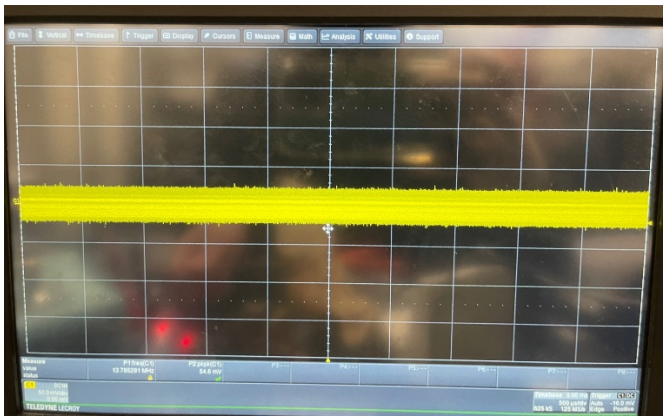

Figure 7 The frequency of RFID when it receives the data from RFID card

### 3.2.1 RFID Tag

As shown in Figure 7 the reading on the oscilloscope is 13.78MHz which satisfied the requirement as the Table 4 in Appendix A.

## 3.3 POS/Customer Interface

### 3.3.1 Client Module

The client module is centered around latency and how quickly it can interact with the dataset. The image below shows that the latency is consistently under 1.5 seconds (see figure 8). Since the refresh rate for the application is 0.4Hz, the longest possible delay before changes become appear on the app is 4.9 seconds (see equation 3) which satisfies requirement 2 in Table 5 in Appendix A. Requirements 1, 3, and 4 was proven during the demonstration; without accomplishing such requirements would result in application failure. Hence, by its functionality, those requirements are proven to be fulfilled.

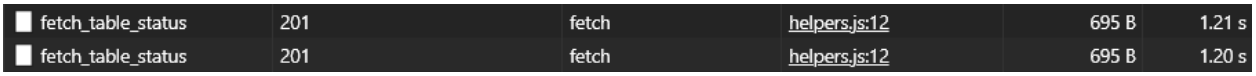


Figure 8 The latency of fetching data from the database

$$Delay_{Max} = \frac{1}{frequency} + (2 * T_{function}) \quad (3)$$

## 3.4 Control Unit

### 3.4.1 Microcontroller

The microcontroller is verified by the datasheet and the rest of the requirements and verifications are met which we demonstrated during the demo. They can be referenced on Table 6 in Appendix A.

# 4. Costs

## 4.1 Parts

**Table 1 Parts Costs**

| Part | Manufacturer | Retail Cost ($) | Bulk Purchase Cost ($) | Actual Cost ($) |
|---|---|---|---|---|
| LED | Rohm | $0.49 | $1.47 | $1.47 |
| Button | WURTH ELEKTRONIK | $0.50 | | $0.50 |
| RFID Module | HiLetgo | $5.49 | | $5.49 |
| Microcontroller | HiLetgo | $10.95 | | $10.95 |
| Voltage Regulator | Texas Instruments | $0.35 | | $0.35 |
| 3D Printer Filament (for Printed Casing) | Snapmaker | $8.00 | | $8.00 |
| Li-po 3.7V 1600mah | Snapmaker | $7.69 | | $7.69 |
| **Total** | | **$33.47** | | **$34.45** |

## 4.2 Labor

On average an ECE Major from UIUC made $88,000 as a starting salary. This roughly translates to $42.31 per hour.

The main aspects of the project are as follows:
SQL Database Creation: 8 hours (Aman)
Flask API: 8 hours (Patrick)
Making the React Application: 40 hours (Aman & Patrick)
Designing the PCB: 10 hours (Junwoo & Aman)
Building the RFID: 5 hours (Junwoo)
Building the WIFI Module: 5 hours (Junwoo)
Wiring the Circuit: 4 hours (Junwoo & Patrick)
3D Design/Print Shell: 3 hours (Patrick)

The total amount of labor has been around 83 hours which (using the equation ($/hour) x 2.5 x hours to complete = TOTAL) roughly equates to about $8,779.33 cost of labor for creating this project.

# 5. Conclusion

## 5.1 Accomplishments

We were able to make a proof of concept that shows a fully functional application that can be used by the restaurant industry in order to optimize the operation of a dining room.

## 5.2 Uncertainties

It is uncertain how robust the RFID solution will be and if that will make it into our final project.

## 5.3 Ethical considerations

There are several safety hazards that we will need to handle when it comes to this project. In accordance with the IEEE Code of Ethics[4] I.1: "to hold paramount the safety, health, and welfare of the public, to strive to comply with ethical design and sustainable development practices." The first safety hazard that we face is that all of the power needs to be properly grounded. If there is an exposed wire, there is a chance of electrical. Additionally, we will need to make sure that our project has some level of waterproofing. If a customer accidentally spills water on our system then it could cause some electrical issues, not to mention that it will be extremely unsafe for the consumer. Some safety considerations when designing this project are as follows. We need to ensure that when we are constructing each module that we are cautious of common electrical standards.

We will adhere to the IEEE Code of Ethics[4] I.1: "to hold paramount the safety, health, and welfare of the public, to strive to comply with ethical design and sustainable development practices." In order to ensure that our project is ethically sound we will need to ensure that all of the user data that is inputted in our POS will be encrypted and not stored locally. Keeping consumer safety is paramount to safety in this digital age when personal information can be stolen and used for gain. We need to ensure that the technology is understandable and discloses exactly what information it needs from the user. Ethically, it is important that while we design this system, we ensure that how it works is easy to understand by both the customer and the server. That entails having clear descriptions as to what information the POS system requires and what information will be stored. This includes any payment information, especially if the customer uses their credit card to pay. Additionally, these practices allow for these services to adhere to the CCPA This information must be kept secure and confidential in order for our design to fully adhere to the IEEE code of ethics. Additionally, these practices allow for these services to adhere to the California Consumer Protection Act.

## 5.4 Future work

In the future we plan to continue working with our sponsor to bring this product to market and hopefully produce a phenomenal project.

## References


[1] "esp32-wroom-32_datasheet_en.pdf." *espressif.com*. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-wroom- 32_datasheet_en.pdf.

[2] "MFRC522 Standard performance MIFARE and NTAG frontend" *nxp.com*. [Online]. Available: https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf

[3] "TLV700 200-mA, Low-IQ" *ti.com*. [Online]. Available: https://www.ti.com/lit/ds/symlink/tlv700.pdf?HQS=TI-null-null-digikeymode-df-pf-null-wwe&ts=1607570782518

[4] "IEEE Code of Ethics." IEEE, 1974, Available: www.ieee.org/about/corporate/governance/p7-8.html.

## Appendix A Requirement and Verification Table

**Table 1 Li-ion battery Requirements and Verifications**

| Requirement | Verification | Verification status (Y or N) |
|---|---|---|
| 1. Provides 3.7V +/- 5% | 1. Connect battery to a multimeter and ensure that the output voltage is 3.7V +/- 5% | Y |
| 2. The battery can store at least 1560mAH of charge | 2. Discharge the battery by connecting it to a positive terminal for 12 hours at 110mA with oscilloscope and multimeter | Y |

**Table 2 Voltage Regulator Requirements and Verifications**

| Requirement | Verification | Verification status (Y or N) |
|---|---|---|
| 1. Provides 3.3V +/- 10% from a 3.7v li-po battery | 1. Connect the output voltage of regulator to a multimeter and ensure 3.3V +/- 5% | Y |
| 2. Maximum current of voltage regulator is 200mA | 2. Connect the output current of regulator to a multimeter and ensure below 200mA | Y |

**Table 3 RFID Reader Requirements and Verifications**

| Requirement | Verification | Verification status (Y or N) |
|---|---|---|
| 1. Determine the reader should be able to receive signals in the 13.56MHz range as that is the requirement for the high frequency tags we will be designing. | 1. Using a frequency generator determine if the reader is able to pick up frequencies in the range of 13.56MHz | Y |
| 2. The RFID reader should only work with the RFID tag used by the restaurants and no other RFID enabled electronics | 2. Verify that only the RFID reader only accepts one RFID tag (with the correct information) by testing multiple RFID tags (e.g., school id) | Y |

| | | |
|---|---|---|
| 3. Once the RFID reader scans and RFID tag, the SQL database should be updated to toggle the table's state (available or occupied) | 3. Verify that the SQL database is updated correctly and that the proper table state is stored. | Y |
| 4. The RFID reader needs 3.3V +/- 10% | 4. Connect the input voltage of RFID reader to a multimeter and ensure that the voltage is 3.3V +/- 5% | Y |

**Table 4 RFID Tag Requirements and Verifications**

| Requirement | Verification | Verification status (Y or N) |
|---|---|---|
| 1. Each RFID Tag has a unique ID. | 1. Check UID by running the code in Arduino IDE. | Y |
| 2. RFID Tag should communicate to the SQL database and update the table's status every time it is scanned. | 2. Verify that the database entry corresponding to the table is modified each time the RFID is swiped | Y |

**Table 5 Client Module Requirements and Verifications**

| Requirement | Verification | Verification status (Y or N) |
|---|---|---|
| 1. The web application properly fetches data from the SQL tables | 1. Check SQL tables to ensure that the application shows ALL data from a single table (and that the table is different for every account). | Y |
| 2. The web app properly refreshes to display new/current data within 15 seconds when updating SQL tables. | 2. Add new data via API call and ensure that the new data is viewable via the platform within 15 seconds of updating the SQL tables. | Y |
| 3. Ensure that the web app is responsive on desktop and mobile. | 3. Use Chrome's dev tools to test multiple devices. | Y |
| 4. Be able to update status and tables via the web platform. | 4. Check the SQL tables to ensure the web platform has made the proper changes. | Y |

**Table 6 Microcontroller Requirements and Verifications**

| Requirement | Verification | Verification status (Y or N) |
|---|---|---|
| 1. Ensure SPI flash holds all the proper API commands | 1. Reading the contents of flash will prove this to be true or false | Y |
| 2. Ensure all sensors produce signals as expected | 2. Use oscilloscope to verify the sensors are not flawed | Y |
| 3. Ensure circuit recognizes all signals | 3. Use multimeter to verify that our controller attempts to send data to the Flask | Y |
| 4. The microcontroller needs 3.3V +/- 10% | 4. Connect the input voltage of RFID to a multimeter and ensure that the voltage is 3.3V +/- 10% | Y |