Hands Free Drinks Mixer

By

Dawith Ha

Eric Mysliwiec

Matthew Gross

Final Report for ECE 445, Senior Design, Spring 2019

TA: Channing Philbrick

01 May 2019

Project No. 40

Abstract

The purpose of this project is to design, build and test an automatic drink maker. A user interface was utilized to allow the user to select a drink. Once selected, a NodeMCU microcontroller will rotate a cup and place it under the appropriate nozzle. The microcontroller will then trigger the corresponding pump allowing liquid to flow. Aside from the microcontroller, pumps and stepper motor, everything was designed by the team and put on either custom printed circuit boards or perfboards. In the final product implementation everything worked as planned apart from the custom stepper motor driver. It was replaced with a prebuilt chip due to the unreliability of the custom design.

Table of Contents

1. Introduction

1.1 Purpose

Waiting in lines is something that most people cannot avoid on a day to day basis. In fact, people wait in lines so often that Richard Larson, an MIT professor, estimates “some people spend a year or two of their lives waiting in line.” [1] A great way to minimize this waste of time is to decrease the wait time to get a drink at the bar. For bartenders, trying to serve every customer and finding the right tab behind the bars can be challenging. Adding more staff to speed up the process is neither space nor cost effective, especially since bars have limited space behind the counter. Smaller lines and faster processing at bars is necessary to increase their productivity, as well as reduce the amount of stress for both the bartenders and the customers.

Building an automated drink mixer will ensure that there are fewer tasks for bartenders, so the waiting time could be cut significantly. This machine will assist bartenders by serving customers who want preselected drink specials for the day. Having this system in place will cause a significant increase in the amount of drinks served in a given amount of time by reducing the workload for bartenders. In the upcoming chapters, the physical components of the system, as well as its design process and decisions will be discussed to see how it came to fruition.

1.2 Functionality

The proposed solution differs from systems used in bars commercially in that it is smaller in size and is personalized, whilst having the ability to transmit sales data via Wi-Fi. This will assist bartenders when lines get long by functioning as a self-serve device. The increased rate in drinks being made will keep customers happy and streamline sales during more extreme times such as rush hours. The core of the functionality is the automated drink making process, in which the system accepts user input from a simple user interface consisting of an LCD and four buttons. The user will be able to choose which drink to make and place the cup on a disk. The disk will rotate, and the pumps above the disk will dispense liquid as required by the recipe of the chosen drink. Then, the drink will be served by moving the cup to the user.

Additionally, through having an RFID scanner for cards, the system will remove the need for the bartenders to look for the right tab among many cards. Places such as Pour Bros have incorporated RFID cards to track customers’ drinks and allowing the customers to pour their drinks themselves. [2] This project is more advanced in that it combines Wi-Fi transmission with RFID drink tracking. To make the drinks, all that the bartender would need to do is scoop some ice into the cup and put it in the machine. All that the customer needs to do is to place their RFID tag on the scanner and select their drink.

1.3 Subsystem Overview

The project consists of various subsystems to make it work smoothly. The block diagram in Figure 1 contains five main modules: the power circuitry module, motors module, sensors module, logic module and user interface module. The power circuitry ensures that the standard 120 Volt AC wall outlet gets

properly converted to usable voltages – 5V DC for the microcontroller and sensors, 12V DC for the weight sensors and the stepper driver, and 24V DC for the pump drivers. The motors module controls the core of the system. This includes rotating the disk with the stepper motor and pumping/dispensing the liquids. The sensors module is used for the fine tuning of the system. The tasks of measuring the flow of liquids, calibrating the position of the cup, and warning operators of when an ingredient is low are all done with the sensors module. The logic module is the brains of the system, this module uses data in two different ways. Firstly, the microcontroller reads various inputs and displays the menu on the LCD for proper drink selection. Secondly, the refill indication control uses input logic from the weight sensors beneath each liquid tank to identify empty bottles. Finally, the user interface module allows the user to interact with the system. Any request for a drink or information being delivered to the user is done here. Independent from the system, a server or a network-connected computer is necessary to receive the tab log file from the machine in real-time as each drink is made.

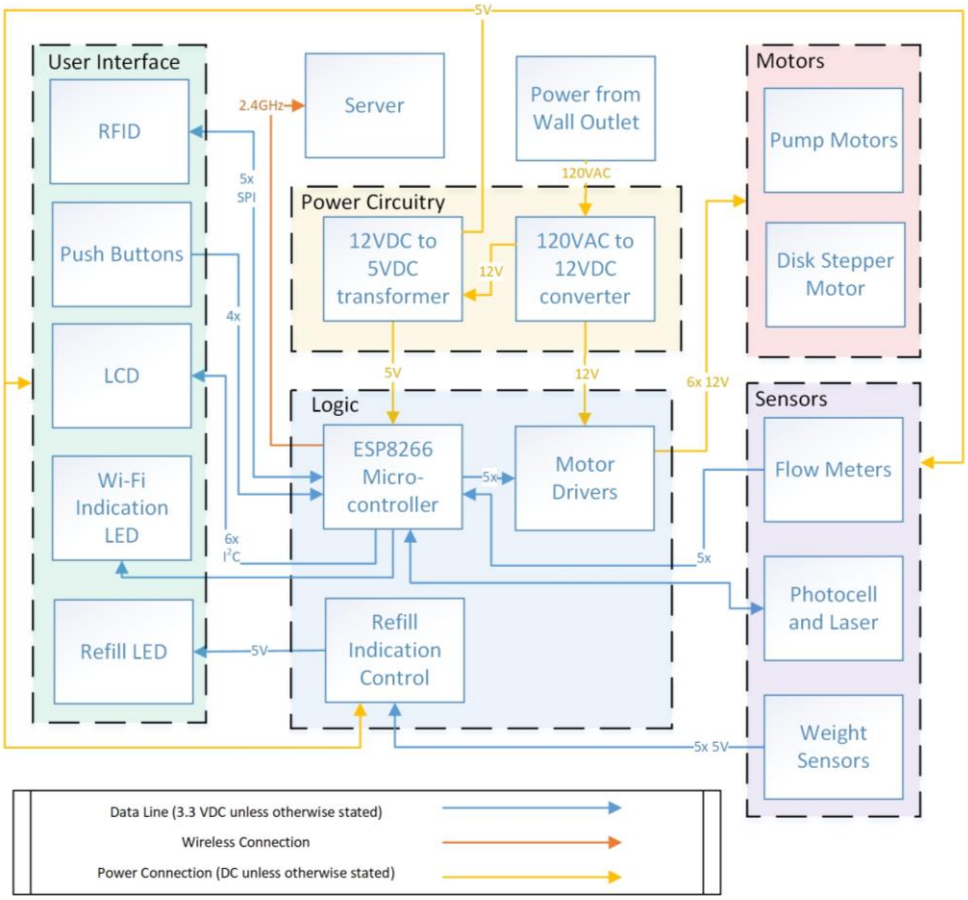

Figure 1. Block Diagram

2 Design

It is important to understand how and why this project works before going into the specifics of each component. The main objective is to use a microcontroller to get fluid from their respective bottles into a cup to serve. Pumps are utilized to accomplish this task by pulling fluid out of a bottle and into tubing when triggered by the controller. However, this is useless unless a cup can be positioned under the pump that is currently active. This is done by using a stepper motor to rotate a disk with an attached cup under the appropriate pump. While a stepper motor allows for accurate movement of the disk to different positions, it does not provide the starting position of the disk. A laser and photocell calibration technique is used to provide this information. A laser is placed under the rotating disk and a photocell is placed directly above the laser. The bottom disk will obstruct the line of sight from the laser to the photocell until a small hole in the disk passes this line of sight. The laser will then shine through the hole and hit the photocell alerting the controller the exact orientation of the disk. From this the stepper motor can place the cup within .9 degrees of any given nozzle. Now that the project can pump fluid and place a cup in the proper position there are still user interface problems to overcome. The first is allowing a user to navigate a menu and select a drink they would like to be made. This is done by placing an LCD screen and four push buttons in a location that is easy to access. Next, it needs to be able to both track users' drinks and verify that the user can get a drink. An RFID reader is employed to require a registered RFID card to be scanned before the project can function. As and added feature, this project also contains an indication LED to alert when a supply is getting low. The specific implementation and design details of each component, including supplying power, are described in the following sections.

2.1 Physical Design

The implementation of our physical design changed slightly from our original design to the design shown in Figure 2. The main frame was built from aluminum railings and attached to a wooden base. The disks were cut from thin aluminum sheets using a CNC router. This allowed for precise cuts required by our sensors and nozzle placement. The bottom disk is attached to the stepper motor and contains a hole for the cup and laser calibration. The top disk contains holes for each nozzle and photocell placement. By using a wooden base we were able to add a section for the containers to rest on top of our weight sensors.

2.2 Power Circuitry

Distributing power to components is essential to the success of this project. Undervoltage will cause the components to stop running, while overvoltage or overcurrent could destroy components or potentially the entire system. This subsystem will also be responsible for successfully isolating noise from travelling between different subsystems through the power lines.

2.2.1 120V AC to 24V DC Transformer

This project will take power in from a standard 120V AC wall outlet. Most components in the project will not be able to handle nor use the 120V supplied by the wall outlet. This transformer will lower the voltage to a usable 24V. Originally a 120V AC to 12V DC transformer was used, however, this was changed to 24V to increase the voltage to the pumps to obtain a higher flow rate.

2.2.2 24V DC to 12V DC Converter

The stepper motor as well as the weight sensor circuitry is built to run off a 12V power supply to provide the necessary potential gradient for precise motor control and weight sensing respectively. A switching voltage regulator is used to step down the voltage instead of a linear regulator to avoid needing a heatsink and fan. [3]

2.2.3 24V DC to 5V DC Converter

Like the 24V DC to 12V DC converter, the project will also require a step down to 5V DC. Many components in this project can only withstand 5V, and most logic circuitry will begin failing outside of the 4.5 to 6.5 Volt range. A switching voltage regulator is used as above to step down the voltage instead of a linear regulator to avoid needing a heatsink and fan. [3]

2.3 Motors

The motors carry out the mechanical tasks of the machine such as moving liquid from the tanks to the cup. Furthermore, it moves the cup such that it accurately rests below dispensers for the duration of the dispensing process for each ingredient.

2.3.1 Pumps

The pumps will require high flow rates for every type of liquid being dispensed to be able to keep up with demand and improve upon the efficiency of a human bartender. Fluids containing alcohol will be less dense than water and thus for testing purposes water will be considered enough for testing speed of the pumps. Peristaltic pumps are used due to their accurate method of pumping fluid. This allows for precise estimations of how much liquid is dispensed.

2.3.2 Disk Stepper Motor

The stepper motor will allow us to position the cup under whatever nozzle is about to be dispensing. It will have a one way control loop communicating with the microcontroller and can be re-centered when necessary. [5] The stepper motor will give 28 N of force at 20 mm from the center of the axle which translates to 5.6 N of force at the center of the cup when placed 100 mm out. [6] Putting the stepper motor in the half step mode will be used to allow for consistent and accurate positioning of the cup as opposed to full, quarter, and eighth steps.

2.4 Sensors

This project's sensors will be purposed to re-calibrate the position of the cup, measure the amount of fluid dispensed, and detect when the weight of an ingredient is below an arbitrary threshold.

2.4.1 Flow Meters

The flow meters will increase accuracy in dispensing of the liquids. Attached to the tubing, it will collect the flow data in real-time and send it to the microcontroller to calculate how much fluid is being dispensed. The standard liquid volume of a full red solo cup is 12 fl.oz (355 mL) [7] so the dispensed volume must be accurate for all volumes less than 355 mL. This component will not be used due to the inconsistencies in readings, discussed in more detail in Chapter 4. Using time instead of the flow meters is much more accurate.

2.4.2 Photocell and Laser

The photocell and laser will allow us to calibrate the stepper motor positioning of the cup using the microcontroller. This is necessary to do at startup as well as during use to avoid error in the stepper motor positioning stemming from inevitable drift. It was proposed that a hall effect sensor would provide a better calibration. However, this will not be used because the laser system has been tested and verified to be extremely accurate and provides the project with a more appealing look.

2.4.3 Weight Sensors

The weight sensors will be used to detect when an ingredient is getting low. It will then notify the user by lighting up an LED. To know the proper weight at which the system notifies the user, custom bottles of uniform weight made from empty 2L bottles will be used, which weigh 45 g empty and 2042 g full. [8] [9] We alternatively could have used load cells which are essentially the same thing but instead of directly measuring the weight, they measure the strain on a metal bar. The weight of an item placed on the bar correlates directly with the strain on the bar, and it is ultimately more accurate for measuring items of different weight. For this specific project however we only needed to know if we are above or below one specific weight, so the load cell was unnecessary.

2.5 Logic

The logic unit serves as a central processing unit for all the data collected through the sensor, giving orders to specific modules or devices as needed. It will get the RFID input and store it in a log of tabs, as well as sending the data to the server or an email inbox. It will control the selection of drinks through push buttons, as well as displaying different drinks to the LCD. Furthermore, it will make sure that the cups rotate the right amount of steps to correctly position beneath the required nozzle for the drink recipe.

2.5.1 ESP8266 Microcontroller

The microcontroller, powered by 5V DC, will run the written program to interface all data and make decisions accordingly. ESP8266 NodeMCU microcontroller fits the needs for this project, since it includes WiFi capabilities along with data serialization, along with a few GPIOS for sending signals at a low cost.

The ESP8266 NodeMCU contains flash memory to store and run programs. For this project, the program can be divided into seven parts below, as seen in Figure 3:

1. Establish connections with all sensors and modules, and initialize all signals and data structures.
2. Wait for customer interaction through an RFID sensor, and show selection menu on the LCD screen upon successfully scanning a user's RFID tag.
3. Interact with user choices through button signals: navigating through the menu, and selecting or cancelling order.
4. Updating the data structure with proper key-pair value of user ID and number of drinks bought.
5. Sending rotation signals to the step motor, and liquid dispensing signal to the peristaltic pump to make the mix according to the recipe stored in the program.

6. Reset and align the disk to its zero position, indicating that the drink is done. Return to step 2.

7. Upon holding the cancel button for 3 seconds, initiate the shutdown sequence; send the log file to an email or server, and power off the motor. Indicate on LCD that the machine is ready to be unplugged.

2.5.2 Motor Drivers

To prevent damaging the microcontroller, a separate system of transistors is needed to control the pumps and stepper motors due to their large power consumption. The microcontroller will send a 3.3 V signal to the driver to turn on a Bipolar Junction Transistor. The BJT will then allow the 24V DC from the power supply to drive the motors. The motor circuitry will also be protected using diodes to handle any sudden spikes in voltage occurring throughout the motor operation cycle.

2.5.3 Refill Indication Control

This is a custom built PCB utilizing a voltage divider and the large gain of an N-channel MOSFET. This drives an LED when the weight drops to near 10% of the maximum weight of fluid in a tank. [7][11][9]

2.6 User Interface

The user interface provides a means for the user to interact with the system. Through the RFID sensor, the user can let the system know who is buying the drink, and append the relevant information on their tab. Through the buttons, the user can select drinks, and cancel the order during the selection stage. The LCD will display relevant information as the user progresses through the order placement process. Furthermore, the refill LED will light up when a liquid tank runs low.

The design includes four buttons and an LCD screen, to minimize the logic.

2.6.1 RFID

The RFID sensor correctly reads the unique value stored in an RFID tag. Furthermore, it communicates with the ESP8266 microcontroller and sends the data over to be processed, through SPI connection.

2.6.2 Push Buttons

Push buttons, powered by the 5V DC converter, correctly send the push signals to the ESP8266 microcontroller upon being pressed and released. There are four buttons: left, right, select and exit/shutdown. Left and right buttons help the user select, and select and back button enables selection and cancellation of drink making.

2.6.3 LCD

The LCD, powered by 3.3 VDC from the voltage regulator on the microcontroller, display the value formed by the Logic module through getting the input from the microcontroller. The LCD displays drink names during selection, and the ID upon scanning the RFID.

2.7 Server

The server receives the log of bar tabs upon the shutdown sequence from the microcontroller. Through the Wi-Fi module, the microcontroller successfully sends a text file of entries, which consist of key-value pair from user ID to number of drinks purchased. Since having a server is outside the scope of this class and project, data has been simply received through email transmission using SMTP.

2.8 Supporting Tables and Figures

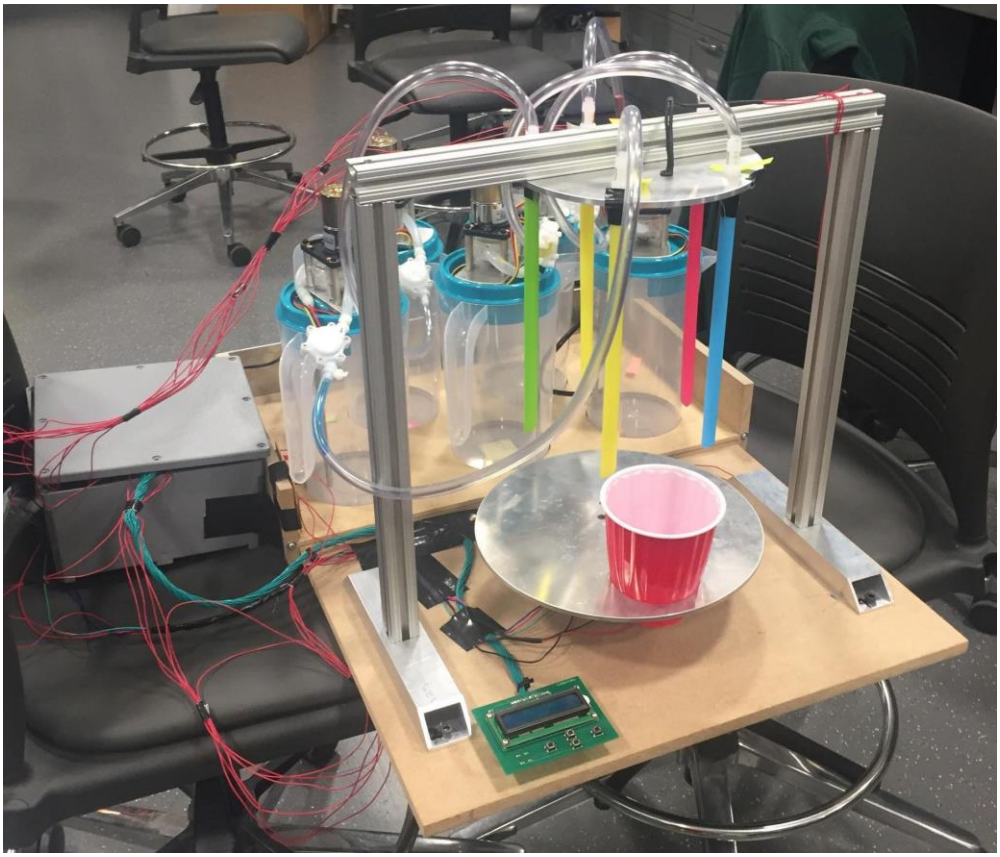

Figure 2. Realized Physical Design

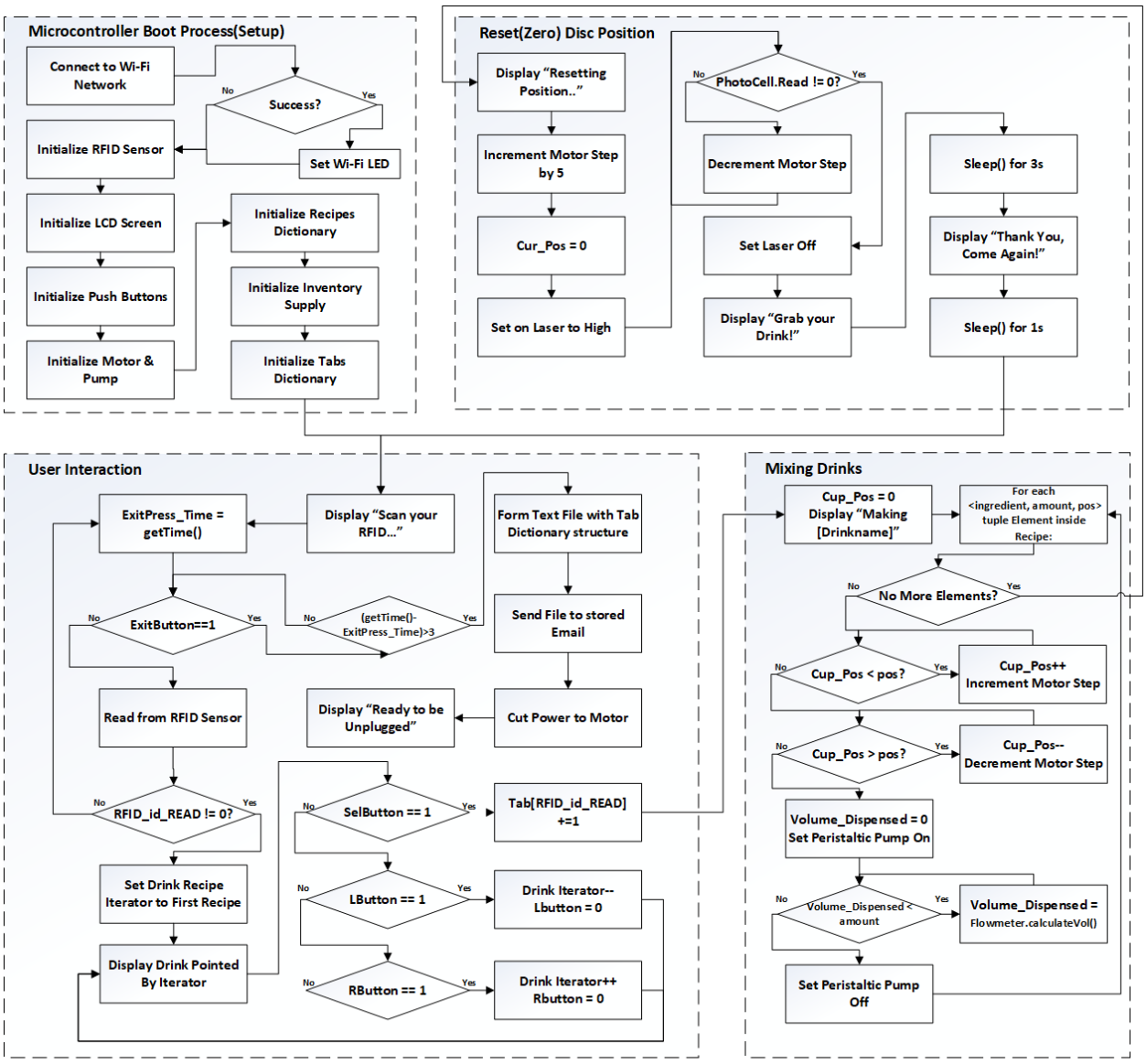

Figure 3. Software Diagram

3 Cost and Schedule

3.1 Cost Analysis

The overall cost of development includes both the time each project member put into working on the prototype as well as the price of all the parts used for the prototype. To calculate the labor cost, it is assumed that each of the three members is making around 40 dollars an hour, working 10 hours a week and will do this for 16 weeks. The total labor cost is calculated as shown in Equation 1.

$$\boldsymbol{Total\ Labor\ Cost} = \mathbf{3}\ (\boldsymbol{members}) * \frac{\$40}{hr} * \frac{10\ hr}{wk} * \mathbf{16}\ (\boldsymbol{weeks}) * \mathbf{2.5} = \$\mathbf{48,000} \quad (1)$$

With a labor cost of $48,000 and a prototype cost as shown in Table 1 of $270.17, the total cost of the project will be about $48,270.

Part	Cost (prototype)	Cost (bulk ~100)
RFID reader (Amazon)	$6.99	$5.59
Push Button x4 (DigiKey)	$.96	$0.80
Mini LCD (DigiKey)	$10.25	$8.10
LED x5	$0.75	$0.60
DC to DC Converter(Amazon) x2	$6.99	$6.99
AC to DC Transformer(Amazon)	$11.99	$11.99
High Flow Rate pumps x5 (Amazon)	$129.00	$60.75
Stepper motor (Amazon)	$13.99	$11.19
NodeMCU (Amazon)	$8.39	$2.00
MCP23017 Expander chip (DigiKey)	$1.24	$0.94
Shields (PCBWay)	$8.00	$0.60
Refill Control Circuit (PCBWay)	$4.00	$0.10
Photocell (DigiKey)	$0.95	$0.90
Weight sensor x5 (DigiKey)	$60.72	$36.70
Laser diode (DigiKey)	$5.95	$4.76
Total	**$270.17**	**$151.94**

Table 1. Cost Analysis Table

3.2 Schedule

Week	Eric: **Flow Meters, Photocell/ Laser Sensor, & Refill Indication System**	Matthew **Motor Driver & Motors, Power Circuit, Breakout Logic**	Dave **ESP, LCD, RFID, pushbuttons**
2/25	1. Schematic 2. Simulations 3. Datasheets 4. Order Parts 5. Build Prototypes of Breakout Logic, Power Circuit	1. Schematics 2. Simulations 3. Datasheets 4. Order Parts 5. Build Prototypes of Breakout Logic, Power Circuit	1. Plan out entire overview of system logic, including pin assignments 2. Plan out recipes and positions of ingredients, create data structures 3. Find libraries and tutorials for all sensors and devices 4. Order Parts
3/4	1. Finalize physical design with Machine Shop 2. Build Prototypes 3. Design, get approved and order PCBs	1. Test Prototype Circuits 2. Adjust schematics as necessary	1. Prototype all logic on the microcontroller: a. connect to Wi-Fi network with proper credentials b. connect MCP23107 expansion chip c. connect and verify RFID, LCD, Buttons d. verify signals sent to stepper motor, flowmeters, peristaltic pumps, pinhole laser and Wi-Fi indicator LED e. connect and set thresholds for pinhole photocell reading
3/11	1. Test and prototype all circuits	1. Test Motor Driver 2. Test Power Circuit	1. Test Wi-Fi file transmission through email
3/18	SPRING BREAK	SPRING BREAK	SPRING BREAK
3/25	1. Integrate Sensors with breakout logic	1. Integrate Power Circuit with motor driver, breakout logic	1. Integrate physical container and disc system with microcontroller 2. Integrate the PCB together with all the parts 3. Test make a drink

4/1	Debug and Pray	Debug and Pray	Debug and Pray
4/8	Debug and Pray	Debug and Pray	Debug and Pray
4/15	Mock Demo	Mock Demo	Mock Demo
4/22	Real Demo and Mock Presentation Prepare Final Report	Real Demo and Mock Presentation Prepare Final Report	Real Demo and Mock Presentation Prepare Final Report

Table 2. Proposed Schedule

4. Requirements & Verification

4.1 Power Circuitry

4.1.1 120 V AC to 24 V DC Transformer

It was decided that a voltage tolerance of 20% is tolerable because the DC to DC converters are variable and thus can be powered by anything over their highest voltage necessary, which for this project is 12V DC. The only other system relying upon our 24V DC supply directly is the pump motors. These motors originally would operate at 12V DC but were running too slow to meet the two minute time frame for making a full mixed drink. Now these motors can be driven with around 19V which translates to just over 20% tolerance. The transformer was tested by plugging the unit into a standard US wall outlet and using an oscilloscope to measure the voltage output. This method also provided a way to visually determine the voltage ripples as the power supply rectified the sine wave into a DC signal. The results of these tests provided a maximum voltage ripple of 0.3 Volts which can be seen in Figure 4. The voltage across the terminals of our power source was 24.3V DC which was measured with a Digital Multimeter.

4.1.2 DC to DC Converters

The DC to DC converters were given a strict and a relaxed tolerance respectively. The 5V DC source powers sensitive logic circuitry. The logic circuitry would only operate if consistently fed between 4.5 and 6.5V so from a high level point of view, the requirements for this source are that the average voltage output give or take the voltage ripple must always be between 4.5 and 6.5 V. This was tested using a Digital Multimeter capable of taking samples over time as well as functions to average the values while also give the maximum and minimum deviation for the duration of the test. The converters were connected to the circuitry in their final PCB and measured with the DMM to give the measurements recorded. The results of these tests were that the 5V DC converter gave 5.023 with less than one millivolt of deviation which met the requirements and can be seen in Figure 5. The 12V DC converter had significantly greater allowed tolerance because the stepper motor could be driven on a wide range of voltages. The voltage was measured to be 12.03V DC with a basic DMM and no visible change in voltage was measured over the range of a 30 second test. Given this information, the ripple voltage for the 12V DC source must logically be less than 10 mV which meets the requirements laid out.

4.2 Motors

4.2.1 Pumps

The rate at which the pumps should flow was determined by setting a goal for how long a customer should wait for their drink and dividing the total volume that would have to be filled in order for an order to be completed to find the rate in milliliters per minute as in Equation 2. The pumps would thus need to be able to flow at any rate greater than the calculated rate.

$$(Volume\ /\ Time\ )\ =\ 375\ mL\ /\ 1.5\ minutes\ =\ 250\ mL\ per\ min \quad (2)$$

The pumps were tested by first filling the pump and all connected tubing with water. Then the pump was supplied with 24V DC for 30 seconds and the tubing was aimed into an empty measuring cup so as to measure the fluid that was output. Multiplying this value by 2 would give the correct rate of flow per minute. The results of this test are pictured in Figure 6 that at 24V DC our pumps would output 780 mL of fluid in one minute, which met our design specifications.

4.2.2 Disk Stepper Motor

It was determined that the rim of the cup must be .25 inches past the stream of liquid from the nozzle. The top rim of the solo cup is four inches in diameter and the calibration allows for .5 inch of error in each direction. This means, given a worse case scenario, the stepper must place the solo cup within 1 inch of the nozzle. Code was made to test this by first calibrating, then going to a given position. Every time the stepper was finished there was an unmeasurable offset from the nozzle. This proved that not only is the calibration more than enough but also the stepper motor would be able to consistently position the cup in the exact position needed.

4.3 Sensors

4.3.1 Flow Meters

The flow meters were originally set to fill a solo cup filled with ice, which cut the estimated volume to fill to complete an order at 178mL. This meant that the flowmeters would have to collectively measure 178mL, and using the arbitrary standard that % error should be around 5% then that gave the value of 10 mL either way for tolerance. This was tested by flowing water through our peristaltic pumps, through the flowmeter, and then outputting the water into a measuring cup. The flowmeter gives a pulse train output so the amount of pulses was counted with the microcontroller and compared against the amount of fluid measure by the measuring cup. The results of this test are shown in Figure 7 and show that the flowmeters begin with a relatively inconsistent measurement and then stabilizes around 200 mL. For this project design which is intended to dispense an accurate shot of 1.5 oz (44 mL) then this was unacceptable. The measurement of time by the microcontroller was enough and more accurate than using the flow meters due to the inherent consistent flow rate of the peristaltic pumps used in our design.

4.3.2 Photocell and Laser

A simple Arduino program was written to print the data received at the analog to digital pin. A photocell from the 3.3V power supply was connected to the analog-to-digital pin and a resistor was connected from this pin to ground. Finally, a laser was shined onto the photocell, giving the output shown in Table 3. The zeros represent no laser and the 1023 represents the laser being shined. This test not only proved

this system would work but also showed that the system is not sensitive to nearby lighting and a pinhole is not needed for the photocell. Each loop had a delay of a millisecond, which was precise enough for our calibration system waves of 15 milliseconds to catch.

4.3.3 Weight Sensors

When a bottle is at about 9 oz it will need to be refilled to make another drink. 270 grams of water equates to about 9 oz of water. Different amounts of water were put in a cup and placed on a weight sensor. The resistance of the weight sensor was then measured. Figure 8 shown at the end of Chapter 4 shows the data collected during this test. It is obvious that 270 grams is distinguishable from 195 grams and 345 grams.

4.4 Logic

4.4.1 ESP8266 Microcontroller

The microcontroller was able to communicate with each of its submodular devices. MFRC-522 RFID sensor was able to read different unique RFID cards and their UID's within 5ms time, as shown in Figure 9. The code can be referenced from Appendix B.

The stepper motor received pulses with width of 20ms as shown in Appendix B, being much more precise and accurate than the required 50ms. Flowmeter signals were read through the GPIO's as shown in Table 4. Furthermore, the pumps operated with intended signals as shown in the code function pump(). The laser values were read through GPIO's as shown in Table 3. The code function calibrate() shows how the laser was turned on and off. WiFi connection success was indicated through setting the right GPIO values high and low, as indicated by wifiProcess(). Furthermore, drink order data was transmitted over SMTP within 91ms, as indicated in sendEmail() function, with times shown in Table 5.

4.4.2 Motor Drivers

The motor drivers for both the pumps and the stepper motor were built and tested to be able to handle 0.8 Amperes of current flow at 12V. After testing the flow rate for the pumps and increasing the voltage supplied to the pumps then it would follow that the pumps would be using more current, however when tested the pumps only used as much current as was shown in Figure 10. The motor drivers were tested by connecting the pump motor to a 24V DC source and placing the driver between the motor and the negative terminal of the 24V DC source. The final part was to turn the driver on and measure the current through the pump.

The result is that the designed motor drivers were more than enough to drive their respective loads. This is due to the inherently massive amount of gain in the TIP100 transistors, totaling in at a gain of 2500 which will allow 2.5 Amperes through when supplied with 1 mA. Since the motors only used around 350 mA then the pump drivers functioned flawlessly, and no additional heat dissipation hardware was necessary for operation.

4.4.3 Refill Indication Control

The refill indication control requirement was determined by using the spec sheet of the used LED. It was rated to operate in a comfortable range of 30mA to 70mA and only available voltage available was 12V. This was tested by altering the weight on the weight sensor and determining the voltage drop across the

LED. The MOSFET required a voltage less than 2 to trigger the LED and the measured voltage drop across the LED was 1.698V.

4.5 Supporting Tables and Figures

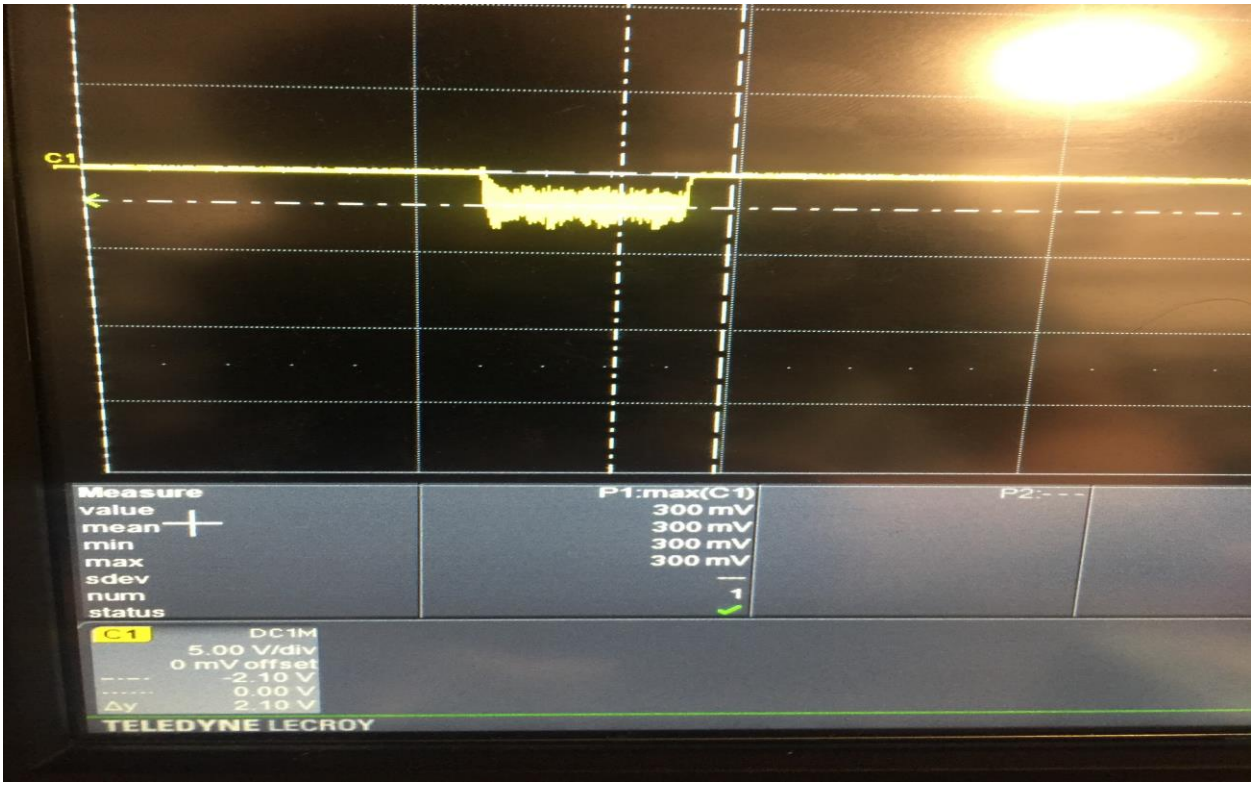

Figure 4. Power Supply Ripple Study

Figure 5. Recorded 5V DC converter DMM output

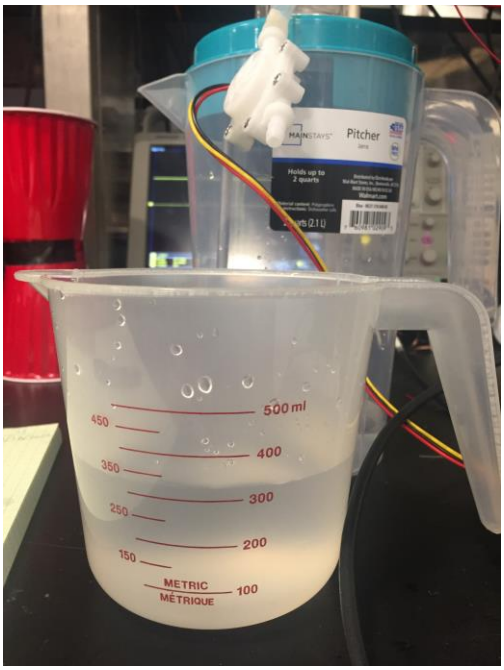

Figure 6. Pump output measurement

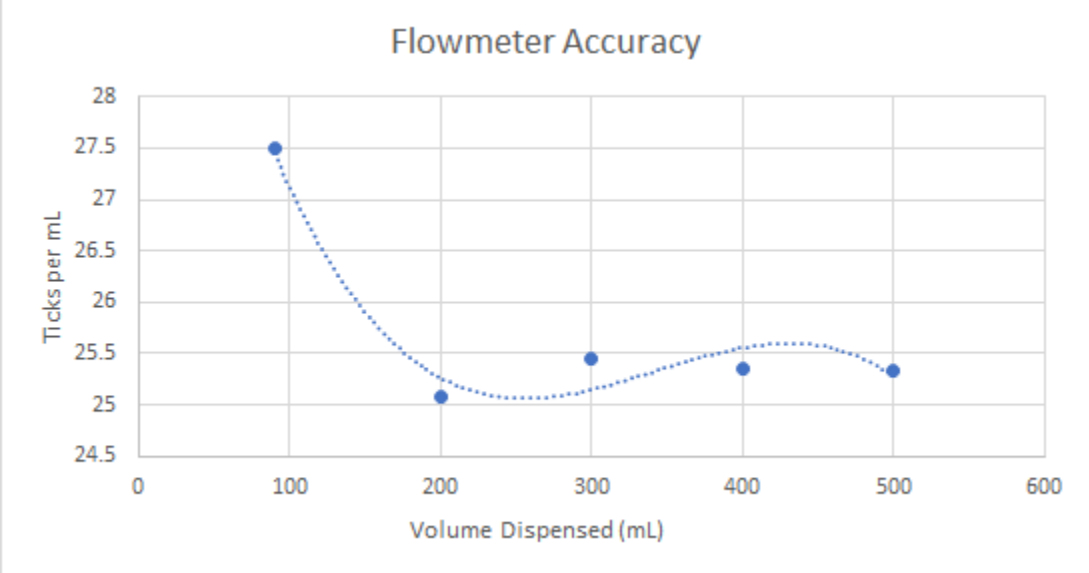

Figure 7. Flow Sensor

Loop #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Value	0	0	1023	1023	1023	1023	1023	1023	1023	1023	1023	1023	0	0	0

Table 3. Photocell/Laser System Output

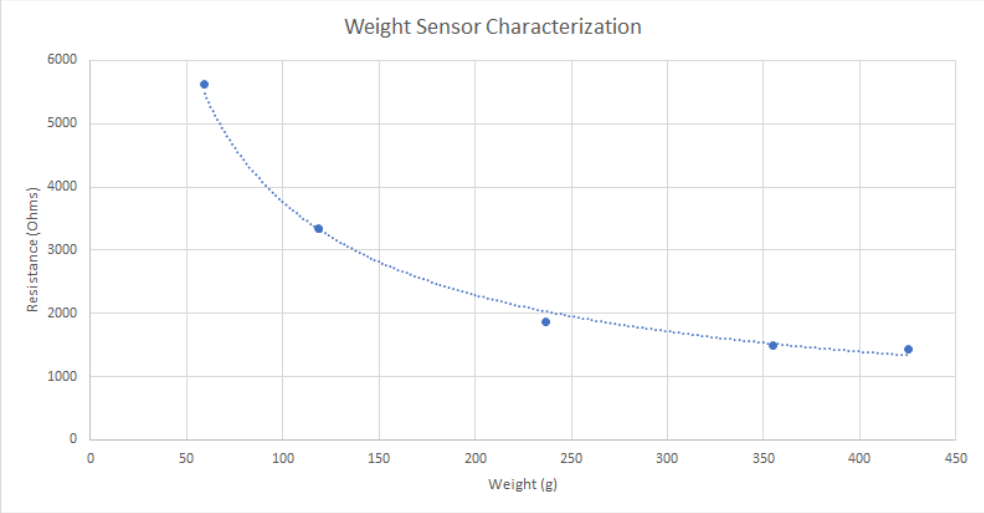

Figure 8. Weight Sensor Characterization Graph

Time (s)	Volume Dispensed (mL)	Ticks Measured
5.00	90	2474
11.10	200	5014
16.66	300	7635
22.22	400	10140
27.77	500	12666

Table 4. Flowmeter Collected Data

```
This code scans the MIFARE Classsic NUID.
Using the following key: FF FF FF FF FF FF

**Card Detected:**
Time(ms) Spent: 4
PICC type: MIFARE 1KB
The NUID tag is:
In hex:  D9 4D F5 5D
In dec:  217 77 245 93

**Card Detected:**
Time(ms) Spent: 3
PICC type: MIFARE 1KB
The NUID tag is:
In hex:  8B 65 B9 15
In dec:  139 101 185 21

**Card Detected:**
Time(ms) Spent: 4
PICC type: MIFARE 1KB
The NUID tag is:
In hex:  E9 39 F5 5D
In dec:  233 57 245 93
```

Figure 9. RFID Reading Test

Figure 10. Motor Driver Current Test

Time Spent(ms)	Order Data
2317	BB5B530D
2317	BB5B530D

Table 5. SMTP Data Transfer Time

5. Conclusion

5.1 Accomplishments

In the end the project was a great success with every subsystem working flawlessly. All modules were able to communicate to the microcontroller and behave with proper signals, with each signal amplified and cut to different voltage and amperage levels as intended within tolerance ranges. These values were in accordance with the requirements set in the tables in Appendix A. The project also solved the original problem, which was to produce a drink within a single process that streamline and minimize the effort involved in making a drink. The test runs were able to produce a mixed drink with each user selection.

This project contained many difficult tasks to properly function. The hardest was getting the custom-built stepper motor driver to work. After many burnt out components and altered designs, the custom-built stepper motor was a success. Despite this success, it was opted to use a premade stepper drive because of its increased accuracy and reliability.

5.2 Uncertainties

The project has several areas of uncertainty spanning from the seemingly random LCD changes to the noisy stepper motor fluctuations. These can only be being caused by the noise from the pump motors running because these unsatisfactory results only show up when the pumps are running. A possible solution to this issue would be to further isolate the power systems by rectifying the power signals for the pumps separately from the rest of the system. This would provide galvanic isolation which would in theory eliminate the issue. The stepper motor fluctuation issue is currently patched by rewiring the direction control to the enable pin, so the motor only moves one direction but turns off when the pumps are in use.

5.3 Ethical considerations

There are two main issues that the project could potentially involve. The first is the potential to over-serve customers by having an automated drink system put into place. This is a conflict with IEEE Code of Ethics 7.8.1 which states that engineers must hold in high value the “health, and welfare of the public.” [14] To solve this, a drink limiter logic will be implemented, where everyone’s account will hold the number of drinks bought. After a predetermined limit is reached, the customer will need approval to place his/her order. The second issue is the potential to serve customers who are underage. Allowing

this with the project, especially on a college campus, would be a clear conflict with the pledge to “to strive to comply with ethical design” [14] In order to prevent this, bars would have a system to register an RFID only to customers who provide proper identification. This project does not deal with batteries or any other potential safety issues and uses no voltages above a normal household outlet. Given that the outlet is following IEEE standards, it should have its own circuit protection. In addition to this, and to protect the people and technology in use within and around this product, power source was chosen carefully -- that contains short-circuit protection, over-voltage shutdown, and an internal fuse to ensure the protection of the community’s health. Another possible safety issue is a faulty stepper motor. If by some Act of God or through some mechanical problem the stepper motor malfunctioned, it could spin out of control and send parts of this project and drinks flying. To solve this problem, an exit sequence is implemented to manually force a shutdown on motors sub-system upon hitting exit button over 3 seconds in the main menu.

5.4 Future work

If this project were to continue there are many things that could be improved or added. The first is making a product the looks nicer and has permanent wiring. Next, it could incorporate Alexa to be able to make drinks based on mood, time of year, or even voice recognition. The final alteration that could be made would be to allow users to custom make recipes via the user interface rather than reprogramming the microcontroller.

Appendix A Requirement and Verification Table

Requirement	Verification	Verification status (Y or N)
1. *Power supply must supply 24VDC +/- 20% and 2A +/- .25A from a 120V AC source.*	1. *To verify:* a. *Use a multimeter to measure the voltage supplied by the converter.* b. *Then use an oscilloscope to verify stability.*	Y
2. *DC to DC converter must supply 5V +/- 10% with at least 0.5 Amperes.[4]*	2. *To verify:* a. *Use a multimeter to measure voltage across transformer* b. *Use an oscilloscope to verify stability.* c. *Then use a 10 Ohm resistor and multimeter to verify amperage output.*	Y
3. *Pumps must provide at least 250 ml of fluid per minute.*	3. *To verify:* a. *Turn on the pump until water is being dispensed.* b. *Then turn the pump off, wait one minute and measure how much fluid was dispensed.*	Y
4. *Stepper motor must be able to consistently align a given nozzle within 1 inch from the center of the solo cup. This translates to being within around +/- 11.46 degrees of the target.*	4. *With the disk connected to the stepper motor and the stepper motor connected to a cutoff switch:* a. *Engage cutoff switch.* b. *Mark how far the disk continues to rotate after cutoff switch has been pressed.*	Y
5. *Flow meters must be able to calculate a total dispensed volume of 178ml +/- 10 ml.*	5. *To verify complete the following 3 times:* a. *Run a premeasured amount of fluid through the meter.* b. *Compare the the meter value to the actual value.*	Y
6. *Photocell must be able to detect the laser going through a 1/8" radius hole upon light contact in under a second to limit the disk*	6. *Write a simple program that does the following:* a. *Read the photocell data.* b. *Print this data along with time stamps.*	Y

from rotating too far past the calibration point.	c. *Shine a laser at the photocell and see how quickly it detects this change.*	
7. *Weight Sensors must be able to differentiate 270 g from 270 g + 75 g.*	7. *To verify:* a. *Measure 270 g on a precise scale and compare value to the sensor's value.* b. *Then measure another 75 g and add it to the sensor.*	Y
8. *Microcontroller must be able to send and receive data to/from RFID sensor through SPI connection.*	8. *After initializing the RFID module, read the value through mfrc522.PICC_ReadCardSerial() and print values inside mfrc522.uid.uidByte buffer onto terminal. Exact steps in section 2.7.1.*	Y
9. *Microcontroller must be able to send pulses of 50 +- 5 ms(to account for programming constructs), of 3.3V onto the stepper motor driver through digital I/O pin.*	9. *Connect the motor driver circuit to pin 3, and run the following sketch code:* a. *inside setup(), call pinmode(3, OUTPUT);* b. *inside loop(), call digitalWrite(3,HIGH);* c. *call digitalWrite(3,LOW);* d. *delay(50); to let the motor catch the pulse* e. *visually confirm that motor rotated half a step.*	Y
10. *Microcontroller must be able to send 3.3V signal to any selected flow meter & peristaltic pump driver, and read the output pulse from the flow meter through the digital I/O pin.*	10. *Connect the flow meter driver input to pin 3, flowmeter output to pin 2 to be read, and peristaltic pump driver input to pin 4 of ESP8266. Pump pipe must be fully immersed in any liquid. Then run the following sketch code:* a. *inside setup(), call pinmode(3, OUTPUT);* b. *call pinmode(2, INPUT); to control the flowmeter.* c. *call pinmode(4, OUTPUT); to control the pump.* d. *inside loop(), call digitalWrite(3,HIGH); and digitalWrite(4,HIGH); to power the pump and flowmeter*	Y

	e. *serial.print(digitalRead(2)); to see the flowmeter value.* f. *visually confirm that motor, and flowmeter are operational.*	
11. *Microcontroller must be able to set the pinhole laser high through digital I/O, thereby sending 3.3V, and read the photocell value between zero and non-zero values between on/off*	11. *Connect the laser to pin 3 and photocell to pin 4, and run the following:* a. *inside setup(), call pinmode(3,OUTPUT);* b. *pinmode(4,INPUT);* c. *inside loop(), call serial.print(digitalRead(4)) and verify the read value is 0.* d. *call digitalWrite(3,High); and verify the laser shooting.* e. *serial.print(digitalRead(4)) and verify that the read value is non-zero.* f. *call digitalWrite(3,Low);* g. *visually confirm that the laser turned off.*	Y
12. *Microcontroller must be able to set the LED through digital I/O pin to indicate successful WiFi connection.*	12. *Connect LED to pin 3, and run the following sketch:* a. *#include "ESP8266WiFi.h" and set const char* ssid and const char * password to right values* b. *in setup(), call Wifi.begin(ssid,password); pinmode(3,OUTPUT);* c. *ifi(WiFi.status() == WL_CONNECTED) digitalWrite(3,High);* d. *Visually verify that the LED was lit.*	Y
13. *Microcontroller must be able to send a file smaller than 1MB in size in 91 ms, on a 802.11 a/b/g/n network (802.11b network has speeds up to 11 Mbps, whereas others go up to 600 Mbps. Across different networks of a/b/g/n, the speed should average higher). Due to speed of additional*	13. *Call the following sketch code:* a. *get time before sending email through time = millis()* b. *Send email through gsender->Send(to, message);* c. *get time afterwards, and subtract time to see if it meets requirements by*	Y

programming constructs, the upper bound for acceptable speed will be 1s.	*running Serial.print(millis() - time). Thoroughly discussed in section 2.8*	
14. *Pump drivers must provide 12VDC +/- 1V to all motors.*	14. *To verify:* a. *Connect voltmeter in parallel with motor to measure the voltage across it* b. *Apply power to the 12V supply for the motor driver* c. *Apply a 3.3V Logic signal to the input of the logical control for the motor driver* d. *Report the voltage from the voltmeter*	Y
15. *Pump drivers must provide at least 0.8 +/- 0.05 A to drive the high flow pump motors. [10]*	15. *To verify:* a. *with motor to measure the current through it* b. *Apply power to the 12V supply for the motor driver* c. *Apply a 3.3V Logic signal to the input of the logical control for the motor driver* d. *Report the current from the ammeter*	Y
16. *Refill indication control must drive an LED at 12V +/- 2V with 50 mA +/- 20 mA.*	16. *To verify:* a. *Place weight matching 10% of the maximum weight of the tank on a powered weight sensor* b. *Measure the resistance across the weight sensor* c. *Match the resistance of the weight sensor with one potentiometer in a voltage divider made up of two potentiometers* d. *Adjust the other potentiometer until 50mA is found to be passing through the LED*	Y
17. *RFID module must be able to read at least 3 different RFID tags through SPI, each within 0~3 seconds(clock speed on ESP8266 is 80 MHz, and on SPI, half. With slowest rates of 8bit/writes, a 32-*	17. *Run the following:* a. *time = millis()* b. *read in the RFID card through mfrc522.PICC_ReadCardSerial()*	Y

*byte string should take 32 * 40MHz = 0.8ms. Adding time for other programming constructs such as loops, should be less than 1s)[12].*	c. *check time difference by serial.print(millis() - time);* d. *Do it three times for different cards, and see if the content in the buffer differs. Repeat, and see if the content matches that of earlier iteration.*	
18. *LCD module must display the value set by the logic module(ESP8266) through I2C; since 2 rows of 16 characters are transferred at most, 32 bytes of data are transferred; these need to be transferred within 50ms(since clock speed of ESP8266 is 80MHz, and 32*80MHz = 0.4ms, but accounting for other programming constructs, upper bounded to 50ms) .*	18. *run the following:* a. *#include <LiquidCrystal.h>* b. *set button constants for different digital i/o inputs, as following: RS=D2, EN=D3, d4=D5, d5=D6, d6=D7, d7=D8;* c. *in setup(), set row, col by lcd.begin(16,2)* d. *time = millis();* e. *print message by lcd.print(time-millis());* f. *visually confirm message on lcd, and see time spent*	Y

Appendix B System Code

```
/*
   Automated Drink Mixer
   Dave Ha, Eric Mysliwiec, Matt Gross
*/

/* MFRC522: RFID */
#include <SPI.h>
#include <MFRC522.h>

/* MCP23017: 2->16 Expansion Chipset */
#include <Wire.h>
#include "Adafruit_MCP23017.h"

/* WIFI Connection*/
#include <ESP8266WiFi.h>
#include "Gsender.h"

/* LCD */
```

```
#include <LiquidTWI2.h>

/******** Configurations and pin definitions **********/

// MFRC522 RFID
constexpr uint8_t RST_PIN = 3;     // GPIO# 5->3; configurable
constexpr uint8_t SS_PIN  = 2;      // GPIO# 4->2; configurable
MFRC522 rfid(SS_PIN, RST_PIN);     // Instance of the class
MFRC522::MIFARE_Key key;

// MCP23017 Expansion
Adafruit_MCP23017 mcp0;
Adafruit_MCP23017 mcp1;

// WiFi Config
const char* ssid = "--------";              // WIFI network name
const char* password = "---------";         // WIFI network password
uint8_t connection_state = 0;               // Connected to WIFI or not
uint16_t reconnect_interval = 5000;         // If not connected wait time
to try again
uint8_t WIFI_LED_PIN = 8;                   // Physical pin 1 on MCP1

// LCD Config
LiquidTWI2 lcd (0x20);

// Buttons Config (MCP0)
enum ButtonPress {
  LEFT,
  RIGHT,
  SELECT,
  BACK,
  UNDEFINED
};
uint8_t BUTTON_LEFT_PIN   =  7;
uint8_t BUTTON_RIGHT_PIN  =  6;
uint8_t BUTTON_BACK_PIN   =  5;
uint8_t BUTTON_SELECT_PIN =  4;

// Inventory and Recipes
char * INVENTORY[] = {"Red", "Green", "Blue", "Yellow", "Water"}; // drink
ingredients/supplies
int POSITIONS[] = {0, 78, 158, 240, 318}; // each position is number of
clockwise ticks, each of 0.9 degrees
```

```
int NUM_RECIPES = 3;
int NUM_INGREDIENTS = 5;
char * RECIPE_NAME[] = {" Purple", " Green", " Water"}; // change
NUM_RECIPES if adding more
int RECIPE[3][5] = {
  {5000, 0, 5000, 0, 5000},  // Purple
  {0, 0, 5000, 5000, 5000},  // Green
  {0, 0, 0, 0, 10000}  // Water
};
volatile int CURRENT_DRINK_IDX = 0;

// Stepmotor, cup positioning
volatile int CUP_POSITION = 0;

// MOTOR/FLOWMETER PINS
uint8_t STEPPER_SIGNAL_PIN =     9;
uint8_t STEPPER_DIRECTION_PIN = 10;
uint8_t FLOWMETER_5_PIN =       11;
uint8_t FLOWMETER_4_PIN =       12;
uint8_t FLOWMETER_3_PIN =       13;
uint8_t FLOWMETER_2_PIN =       14;
uint8_t FLOWMETER_1_PIN =       15;
uint8_t PUMP_1_SIGNAL_PIN =      0; // connected to PUMP_1_GND on PCB
uint8_t PUMP_2_SIGNAL_PIN =      1; //               PUMP_2_GND
uint8_t PUMP_3_SIGNAL_PIN =      2; //               PUMP_3_GND ...
uint8_t PUMP_4_SIGNAL_PIN =      3;
uint8_t PUMP_5_SIGNAL_PIN =      4;

uint8_t LASER_SIGNAL_PIN =       6;
uint8_t PHOTOCELL_PIN =          7;

void setup() {
  Serial.begin(115200);
  Serial.println(F("..System Booting........"));

  // RFID Module Initialization
  SPI.begin(); // Init SPI bus
  rfid.PCD_Init(); // Init MFRC522
  for (byte i = 0; i < 6; i++) {
    key.keyByte[i] = 0xFF;
  }
  Serial.println(F("...RFID Module Initialized."));
  Serial.print(F("Using the following key:"));
```

```
printHex(key.keyByte, MFRC522::MF_KEY_SIZE);
Serial.println();


// MCP23017 Expansion Module Initialization
mcp0.begin(0);
mcp1.begin(1);
mcp0.pinMode(BUTTON_LEFT_PIN, INPUT);
mcp0.pinMode(BUTTON_RIGHT_PIN, INPUT);
mcp0.pinMode(BUTTON_BACK_PIN, INPUT);
mcp0.pinMode(BUTTON_SELECT_PIN, INPUT);
mcp1.pinMode(WIFI_LED_PIN, OUTPUT);
mcp1.pinMode(STEPPER_SIGNAL_PIN, OUTPUT);
mcp1.pinMode(STEPPER_DIRECTION_PIN, OUTPUT);
mcp1.pinMode(FLOWMETER_1_PIN, INPUT);
mcp1.pinMode(FLOWMETER_2_PIN, INPUT);
mcp1.pinMode(FLOWMETER_3_PIN, INPUT);
mcp1.pinMode(FLOWMETER_4_PIN, INPUT);
mcp1.pinMode(FLOWMETER_5_PIN, INPUT);
mcp1.pinMode(PUMP_1_SIGNAL_PIN, OUTPUT);
mcp1.pinMode(PUMP_2_SIGNAL_PIN, OUTPUT);
mcp1.pinMode(PUMP_3_SIGNAL_PIN, OUTPUT);
mcp1.pinMode(PUMP_4_SIGNAL_PIN, OUTPUT);
mcp1.pinMode(PUMP_5_SIGNAL_PIN, OUTPUT);
mcp1.pinMode(LASER_SIGNAL_PIN, OUTPUT);
mcp1.pinMode(PHOTOCELL_PIN, INPUT);
Serial.println(F("...MCP23017 Expansion Module Initialized."));
Serial.println();

// Enable WIFI and Light LED
//wifiProcess();

// LCD Initialization
lcd.setMCPType (LTI_TYPE_MCP23017);
lcd.begin (16, 2); // dimension
lcd.clear();
displayWelcomeScreen();


// Keep all Pumps Low (although they should be by default)
mcp1.digitalWrite(PUMP_1_SIGNAL_PIN, LOW);
mcp1.digitalWrite(PUMP_2_SIGNAL_PIN, LOW);
mcp1.digitalWrite(PUMP_3_SIGNAL_PIN, LOW);
```

```
  mcp1.digitalWrite(PUMP_4_SIGNAL_PIN, LOW);
  mcp1.digitalWrite(PUMP_5_SIGNAL_PIN, LOW);

  mcp1.digitalWrite(STEPPER_DIRECTION_PIN, LOW);  // ENABLES STEPPER
  recalibrate();
  while(1){
    delay(50);
    if(mcp0.digitalRead(BUTTON_SELECT_PIN) == HIGH) break;
  }
  fillPipes();

  recalibrate();

  delay(100);
}

void loop() {
  //manualMode(); // must have everything else commented out within loop()

  // STEP 1: SCAN RFID

  if ( ! rfid.PICC_IsNewCardPresent()) return; // Look for new cards
  if ( ! rfid.PICC_ReadCardSerial()) return;   // Verify if the NUID has
been read

  Serial.println();
  Serial.println(F("**Card Detected:**"));
  Serial.print(F("NUID tag #:"));
  String UID = byteToString(rfid.uid.uidByte, rfid.uid.size);
  Serial.println(UID);
  rfid.PICC_HaltA();
  rfid.PCD_StopCrypto1();
  Serial.println("Select your drink: (*display first recipe name)");
  lcd.setCursor(0, 0);
  lcd.print(" Select Drink:  ");
  CURRENT_DRINK_IDX = 0;
  displayDrink(CURRENT_DRINK_IDX);

  // STEP 2: SHOW MENU, HAVE THE USER SELECT W/ BUTTONPRESS
  delay(1000);
  while (1) {
    delay(100);
    // LEFT BUTTON PRESSED
```

```
    if (mcp0.digitalRead(BUTTON_LEFT_PIN) == HIGH) {
      Serial.println("Left Pressed : Show prev recipe");
      Serial.println(CURRENT_DRINK_IDX);
      CURRENT_DRINK_IDX = (CURRENT_DRINK_IDX == 0)? NUM_RECIPES-1 : --
CURRENT_DRINK_IDX;
      displayDrink(CURRENT_DRINK_IDX);
      continue;
    }
    // RIGHT BUTTON PRESSED
    if (mcp0.digitalRead(BUTTON_RIGHT_PIN) == HIGH) {
      Serial.println("Right Pressed : Show next recipe");
      Serial.println(CURRENT_DRINK_IDX);
      CURRENT_DRINK_IDX = (++CURRENT_DRINK_IDX) % NUM_RECIPES;
      displayDrink(CURRENT_DRINK_IDX);
      continue;
    }
    // SELECT BUTTON PRESSED
    if (mcp0.digitalRead(BUTTON_SELECT_PIN) == HIGH) {
      Serial.println("Select Pressed : Make selected recipe");
      break;
    }
    // BACK BUTTON PRESSED
    if (mcp0.digitalRead(BUTTON_BACK_PIN) == HIGH) {
      //manualMode();

      Serial.println("Back Pressed : Order canceled, back to main menu");
      displayWelcomeScreen();
      lcd.setCursor(0, 0);
      lcd.print("Order Cancelled!");
      CURRENT_DRINK_IDX = 0;
      return;

    }
    delay(100);
  }


  // STEP 3: MAKE DRINKS
  Serial.println("Making Selected Drink...");
  /*
  lcd.setMCPType (LTI_TYPE_MCP23017);
  lcd.begin (16, 2); // dimension
  lcd.clear();
```

```
  */
  lcd.setCursor(0,0);
  lcd.print("  Making Drink  ");
  displayDrink(CURRENT_DRINK_IDX);
  delay(100);
  makeDrink(CURRENT_DRINK_IDX);

  /*
  lcd.setMCPType (LTI_TYPE_MCP23017);
  lcd.begin (16, 2); // dimension
  lcd.clear();
  */
  lcd.setCursor(0,0);
  lcd.print("Drink Finished! ");
  displayDrink(CURRENT_DRINK_IDX);
  delay(1000);

  // STEP 4: RESET POSITION
  lcd.setCursor(0,1);
  lcd.print("  calibrating...");
  delay(1000);
  recalibrate();


  // STEP 5: SEND EMAIL
  //sendEmail(UID);


  // STEP 6: RESET SCREEN
  /*
  lcd.setMCPType (LTI_TYPE_MCP23017);
  lcd.begin (16, 2); // dimension
  lcd.clear();
  */
  displayWelcomeScreen();


}

// HELPER FUNCTIONS BELOW*********************************************

void manualMode(){
  // MANUAL MODE: choose pumps and pour
```

```
  delay(1000);
  while (1) {
    delay(100);
    // LEFT BUTTON PRESSED
    if (mcp0.digitalRead(BUTTON_LEFT_PIN) == HIGH) {
      Serial.println("Left Pressed : Show prev position");
      Serial.println(CURRENT_DRINK_IDX);
      CURRENT_DRINK_IDX = (CURRENT_DRINK_IDX == 0)? NUM_RECIPES-1 : --
CURRENT_DRINK_IDX;
      moveCupTo(POSITIONS[CURRENT_DRINK_IDX]);
      continue;
    }
    // RIGHT BUTTON PRESSED
    if (mcp0.digitalRead(BUTTON_RIGHT_PIN) == HIGH) {
      Serial.println("Right Pressed : Show next recipe");
      Serial.println(CURRENT_DRINK_IDX);
      CURRENT_DRINK_IDX = (++CURRENT_DRINK_IDX) % NUM_RECIPES;
      moveCupTo(POSITIONS[CURRENT_DRINK_IDX]);
      continue;
    }
    // SELECT BUTTON PRESSED
    if (mcp0.digitalRead(BUTTON_SELECT_PIN) == HIGH) {
      Serial.println("Select Pressed : Pour selected pump");

      // turn pump on
      mcp1.digitalWrite(STEPPER_DIRECTION_PIN, HIGH); // disable stepper
      delay(100);
      mcp1.digitalWrite(PUMP_1_SIGNAL_PIN + CURRENT_DRINK_IDX, HIGH);
      while(mcp0.digitalRead(BUTTON_SELECT_PIN) == HIGH){delay(20);}
      // turn pump off
      mcp1.digitalWrite(PUMP_1_SIGNAL_PIN + CURRENT_DRINK_IDX, LOW);
    }
    // BACK BUTTON PRESSED
    if (mcp0.digitalRead(BUTTON_BACK_PIN) == HIGH) {
      lcd.setCursor(0,1);
      lcd.print("  calibrating...");
      delay(100);
      recalibrate();
      return;
    }
    delay(100);
  }
}
```

```
void fillPipes(){
  moveCupTo(POSITIONS[0]);
  pour(0,3000);
  moveCupTo(POSITIONS[1]);
  pour(1,3000);
  moveCupTo(POSITIONS[2]);
  pour(2,3000);
  moveCupTo(POSITIONS[3]);
  pour(3,3000);
  moveCupTo(POSITIONS[4]);
  pour(4,3000);
}

void pour(int pump_idx, int volume){
  mcp1.digitalWrite(STEPPER_DIRECTION_PIN, HIGH); // disable stepper
  mcp1.digitalWrite(PUMP_1_SIGNAL_PIN + pump_idx, HIGH);

  delay(volume);
  mcp1.digitalWrite(PUMP_1_SIGNAL_PIN + pump_idx, LOW);
  delay(1000);

  mcp1.digitalWrite(STEPPER_DIRECTION_PIN, LOW); // re-enable stepper
}

// STEPPER MOTOR HELPER FUNCTIONS(MAKING DRINKS)
void moveCupTo(int target_pos){
  //mcp1.digitalWrite(STEPPER_DIRECTION_PIN, LOW); // enable stepper
  delay(500);
  if(CUP_POSITION == target_pos) return;

  int distance = target_pos - CUP_POSITION; // clockwise distance
  if(distance < 0) distance += 400;

  delay(500);
  for(int i=0;i<distance;++i){
    mcp1.digitalWrite(STEPPER_SIGNAL_PIN, HIGH);
    delay(20);
    mcp1.digitalWrite(STEPPER_SIGNAL_PIN, LOW);
    delay(20);
  }
  CUP_POSITION = target_pos;
}
```

```
void makeDrink(int CURRENT_DRINK_IDX){
  for(int i=0;i<NUM_INGREDIENTS;++i){
    if(RECIPE[CURRENT_DRINK_IDX][i] == 0) continue; // skip if 0 amount is
to be poured
    delay(100);
    moveCupTo(POSITIONS[i]);
    // note: pump pin #'s are increments of each other; pump_1 = 0, pump_2
= 1, pump_3 = 2, etc
    delay(100);
    pour(i, RECIPE[CURRENT_DRINK_IDX][i]); // pump idx(0~4),amount)
    delay(100);
  }
}


void recalibrate(){
  // turn laser on
  mcp1.digitalWrite(LASER_SIGNAL_PIN, HIGH);

  // move until laser value 0
  while(mcp1.digitalRead(PHOTOCELL_PIN) == 1){
    mcp1.digitalWrite(STEPPER_SIGNAL_PIN, HIGH);
    delay(30);
    mcp1.digitalWrite(STEPPER_SIGNAL_PIN, LOW);
    delay(30);
  }
  mcp1.digitalWrite(LASER_SIGNAL_PIN, LOW);

  // turn the cup to front
  for(int i=0;i<200;++i){
    mcp1.digitalWrite(STEPPER_SIGNAL_PIN, HIGH);
    delay(30);
    mcp1.digitalWrite(STEPPER_SIGNAL_PIN, LOW);
    delay(30);
  }
  CUP_POSITION = 200;
}

// LCD HELPER FUNCTIONS

void displaySetupScreen(){
```

```
  lcd.setCursor(0, 0);
  lcd.print(" Setting Up..   ");
  lcd.setCursor(0, 1);
  lcd.print("                ");
}

void displayWelcomeScreen() {
  lcd.setCursor(0, 0);
  lcd.print("    Welcome!    ");
  lcd.setCursor(0, 1);
  lcd.print(" Scan your RFID ");
  Serial.println(" Welcome! Please Scan your RFID ");
}

void displayDrink(int idx){ // display drink on the bottom row of the lcd
  if(idx >= NUM_RECIPES) Serial.println(F("drink idx invalid"));
  lcd.setCursor(0, 1);
  lcd.print("                ");
  lcd.setCursor(0, 1);
  lcd.print(RECIPE_NAME[idx]);
}


// WIFI HELPER FUNCTIONS *********************************
uint8_t WiFiConnect(const char* nSSID = nullptr, const char* nPassword =
nullptr)
{
  static uint16_t attempt = 0;
  Serial.print("Connecting to ");
  if (nSSID) {
    WiFi.begin(nSSID, nPassword);
    Serial.println(nSSID);
  } else {
    WiFi.begin(ssid, password);
    Serial.println(ssid);
  }

  uint8_t i = 0;
  while (WiFi.status() != WL_CONNECTED && i++ < 50)
  {
    delay(200);
    Serial.print(".");
  }
```

```
  ++attempt;
  Serial.println("");
  if (i == 51) {
    Serial.print("Connection: TIMEOUT on attempt: ");
    Serial.println(attempt);
    if (attempt % 2 == 0)
      Serial.println("Check if access point available or SSID and
Password\r\n");
    return false;
  }
  Serial.println("Connection: ESTABLISHED");
  Serial.print("Got IP address: ");
  Serial.println(WiFi.localIP());
  return true;
}

void Awaits()
{
  uint32_t ts = millis();
  while (!connection_state)
  {
    delay(50);
    if (millis() > (ts + reconnect_interval) && !connection_state) {
      connection_state = WiFiConnect();
      ts = millis();
    }
  }
}

void wifiProcess(){
  Serial.print(F("Connecting to "));
  WiFi.begin(ssid, password);
  Serial.println(ssid);
  uint8_t i = 0;

  while(WiFi.status()!= WL_CONNECTED && i++ < 50)
  {
    delay(200);
    Serial.print(F("."));
  }
  if(WiFi.status() == WL_CONNECTED){      // if connected to WIFI
    mcp1.digitalWrite(WIFI_LED_PIN,HIGH);
    Serial.println(F("...WiFi connection: ESTABLISHED"));
```

```
    Serial.print(F("Got IP address: "));
    Serial.println(WiFi.localIP());
  }else{
    Serial.println(F("ERROR: WiFi could not connect, check ssid and
password."));
  }
  Serial.println();
}

void sendEmail(String UID){
  Serial.println(F("Sending Order to the Server..."));
  Gsender *gsender = Gsender::Instance();    // Getting pointer to class
instance
  if(gsender->Subject(UID)->Send("handsfreemixer445@gmail.com",
RECIPE_NAME[CURRENT_DRINK_IDX])) {
    Serial.println("Message sent.");
  } else {
    Serial.print("Error sending message: ");
    //Serial.println(gsender->getError());
  }
  Serial.println();
}

// RFID HELPER FUNCTIONS *********************************************

String byteToString(byte *buffer, byte bufferSize) {
  String UID;
  for (byte i = 0; i < bufferSize; i++) {
    UID += String(buffer[i] < 0x10 ? "0" : "");
    UID += String(buffer[i], HEX);
  }
  UID.toUpperCase();
  return UID;
}

/**
   Helper routine to dump a byte array as hex values to Serial.
*/
void printHex(byte *buffer, byte bufferSize) {
  for (byte i = 0; i < bufferSize; i++) {
    Serial.print(buffer[i] < 0x10 ? " 0" : " ");
    Serial.print(buffer[i], HEX);
  }
```

```
}

/**
   Helper routine to dump a byte array as dec values to Serial.
*/
void printDec(byte *buffer, byte bufferSize) {
  for (byte i = 0; i < bufferSize; i++) {
    Serial.print(buffer[i] < 0x10 ? " 0" : " ");
    Serial.print(buffer[i], DEC);
  }
}
```

Appendic C Schematics & Designs

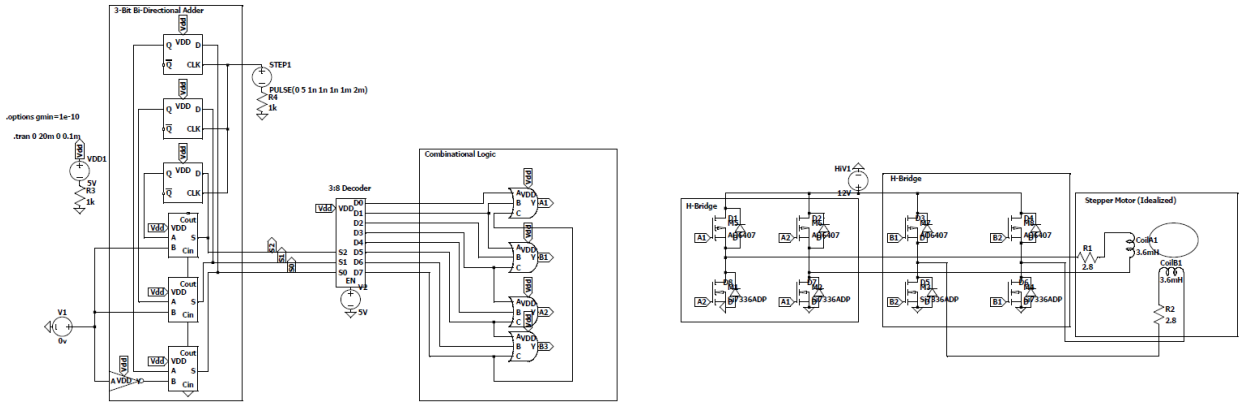

Figure 11. Stepper Motor Driver Schematics

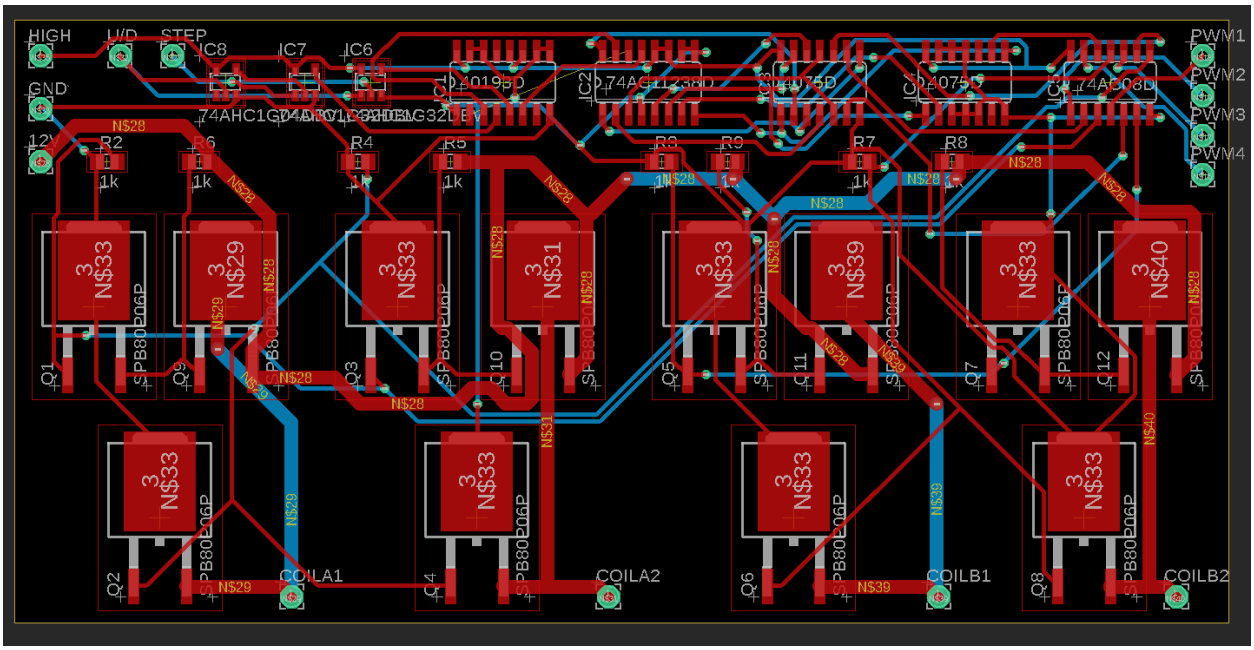

Figure 12. Stepper Motor Driver Printed Circuit Board Design

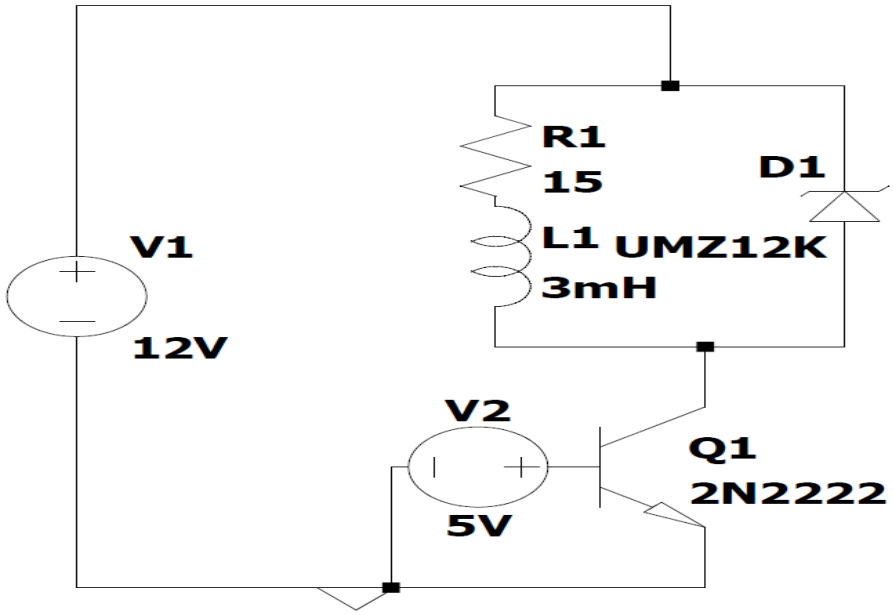

Figure 13. Pump Motor Driver Idealized Schematics

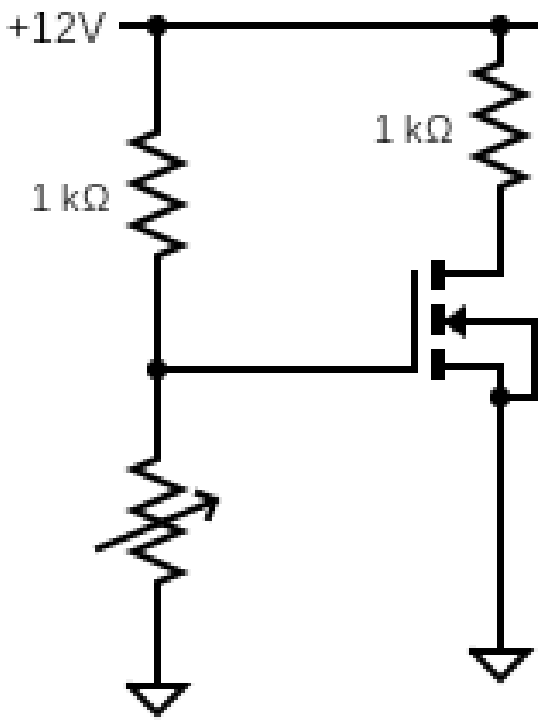

Figure 14. Refill Indication Control Schematic

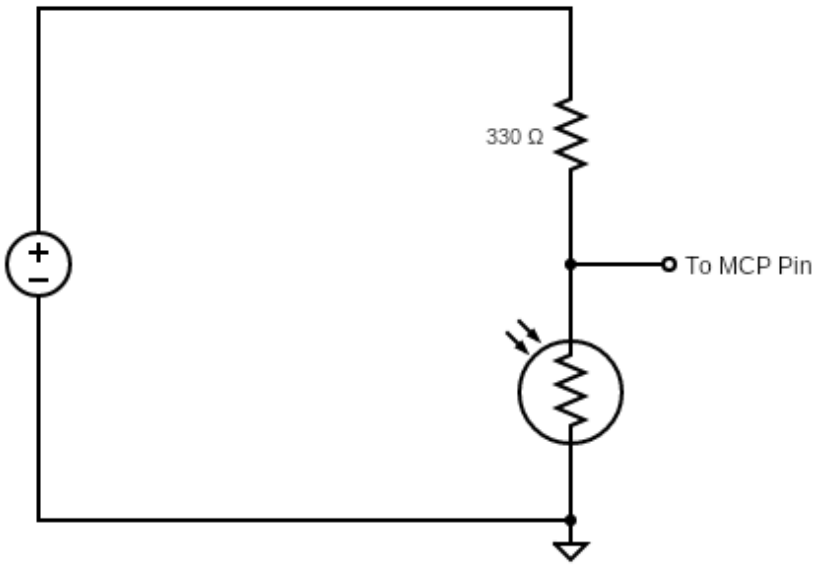

Figure 15. Photocell Sensor Schematic

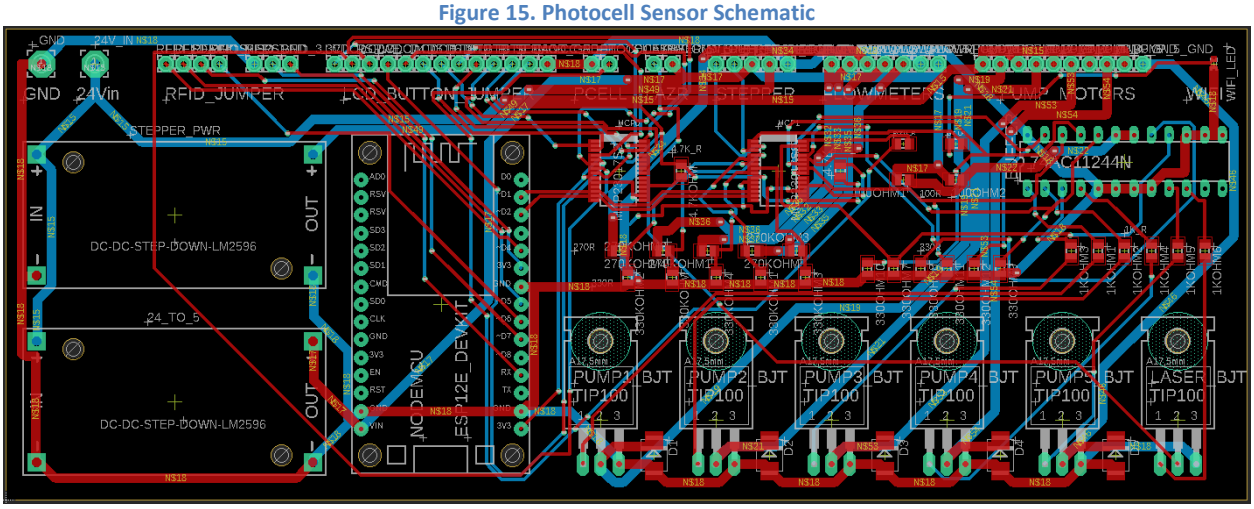

Figure 16. Main Board PCB

References

[1] A. Swanson, “What really drives you crazy about waiting in line (it actually isn't the wait at all),” The Washington Post, 27-Nov-2015. [Online]. Available: https://www.washingtonpost.com/news/wonk/wp/2015/11/27/what-you-hate-about-waiting-in-line-isnt-the-wait-at-all/?utm_term=.d60c8abdd1ef. [Accessed: 22-Feb-2019].

[2] "Pour Bros. Craft Taproom", Pour Bros. Craft Taproom, 2019. [Online]. Available: https://www.pourbrostaproom.com/. [Accessed: 22- Feb- 2019].

[3] “A beginner's guide to switching regulators,” Dimension Engineering. [Online]. Available: https://www.dimensionengineering.com/info/switching-regulators. [Accessed: 22-Feb-2019].

[4]“3W 5V 0.6A DIP-16 Isolated DC-DC Regulated Converter 12 Volt In,” How It Works: Xbox Kinect. [Online]. Available: https://www.jameco.com/z/SLC03A-05-MEAN-WELL-3W-5V-0-6A-DIP-16-Isolated-DC-DC-Regulated-Converter-12-Volt-In_2261815.html. [Accessed: 22-Feb-2019].

[5] B. H., Phil, J. A., and Anthony, “Dart Solo P16R 16 oz. Red Plastic Cup - 1000/Case,” WebstaurantStore, 12-Jun-2018. [Online]. Available: https://www.webstaurantstore.com/dart-solo-p16r-16-oz-red-plastic-cup-case/760P16R.html. [Accessed: 22-Feb-2019].

[6] Adafruit Industries, “Stepper motor - NEMA-17 size - 200 steps/rev, 12V 350mA,” adafruit industries blog RSS. [Online]. Available: https://www.adafruit.com/product/324. [Accessed: 22-Feb-2019].

[7] DNews, “Did You Know the Solo Cup is also a Measuring Cup (for Booze)?,” Seeker, 13-Jun-2012. [Online]. Available: https://www.seeker.com/did-you-know-the-solo-cup-is-also-a-measuring-cup-for-booze-1765829437.html. [Accessed: 08-Feb-2019].

[8] “DID YOU KNOW?,” About PET | PETRA: Information on the Use, Benefits & Safety of PET Plastic., 2016. [Online]. Available: http://www.petresin.org/news_didyouknow.asp. [Accessed: 08-Feb-2019].

[9]“Questions from andrew, a student,” Math Central Quandaries and Queries, Oct-2017. [Online]. Available: http://mathcentral.uregina.ca/QQ/database/QQ.09.06/s/andrew1.html. [Accessed: 08-Feb-2019].

[10] “INTLLAB DIY Peristaltic Pump Dosing Pump 12V DC, High Flowrate for Aquarium Lab Analytical, 170~460 mL/min,” Amazon. [Online]. Available: https://www.amazon.com/dp/B07MTBV5RR/ref=sspa_dk_detail_2?th=1. [Accessed: 22-Feb-2019].

[11] “FSR 400 Series Data Sheet,” Adafruit CDN Shop. [Online]. Available: https://cdn-shop.adafruit.com/datasheets/FSR400Series_PD.pdf. [Accessed: 07-Feb-2019].

[12] “EEVblog Electronics Community Forum,” OVP & OCP - Page 1. [Online]. Available: https://www.eevblog.com/forum/microcontrollers/esp8266-native-spi-hardware-driver/. [Accessed: 22-Feb-2019].

[13] “Load Testing an ESP8266,” arunoda.me. [Online]. Available: https://arunoda.me/blog/load-testing-an-esp8266. [Accessed: 22-Feb-2019].

[14] Ieee.org, "IEEE IEEE Code of Ethics", 2016. [Online]. Available: http://www.ieee.org/about/corporate/governance/p7-8.html. [Accessed: 7- Feb- 2019].