AUTOMATED COCKTAIL MIXER
FINAL REPORT

By

Dominic Andrejek
Benjamin Kotlarski
Nick Kubiak

Project Proposal for ECE 445, Senior Design, Spring 2026
TA: Wesley Pang

May 2026
Project No. 9

Abstract

This report details the design, implementation, and verification of an automated cocktail mixer created to solve the inconsistencies and inefficiencies of manual drink preparation. The system uses an ESP32 microcontroller to control self-priming pumps, a motorized stirring mechanism, and weight verification using feedback from load cells under both the cup and the ingredients. Through a modular design accepting any sort of cup or bottle, the machine avoids locking users into proprietary hardware like similar commercially available devices. Testing confirmed that the final prototype satisfied all high-level requirements: cups as light as plastic cups are identified, fluid is dispensed to an accuracy of ±5 grams per ingredient, and the full drink-mixing cycle for a two ingredient drink is completed in well under 150 seconds.

Contents

1. Introduction 1
1.1 Problem 1
1.2 Functionality 1
1.3 Subsystem Overview 2
1.4 High-level requirements 3
2 Design 4
2.1 Design Alternatives 4
2.2 Subsystem Descriptions, and Procedures 4
2.2.1 User Interface 4
2.2.2 [Stirring Mechanism] 5
2.2.3 [Pumps and Plumbing System] 5
2.2.4 [Control Subsystem] 6
2.2.5 [Status and Weight Verification System] 6
2.2.6 [Power System] 7
2.3 Subsystem Software Design 7
2.3.1 User Interface and Startup 9
2.3.2 Weight Sensor Calibration 9
2.3.3 Pump Control Logic 9
2.3.4 Linear Actuator and Gear Reduction Motor Control Logic 10
2.3.5 Status and Fault Handling 10
3. Verification 11
3.1 User Interface 11
3.2 Stirring Mechanism 12
3.3 Pumps and Plumbing 12
3.4 Control 13
3.5 Status and Weight Verification 14
3.6 Power 14
3.7 Full System 15
4. Cost 16
5. Conclusion 17
5.1 Ethics 17

5.1.1 Specific Applicable Engineering Standards........ 18
5.2 Societal, Economic, Environmental, and Global Impact 18
5.2.1 [Societal]........ 18
5.2.2 [Economic]........ 18
5.2.3 [Environmental]........ 18
5.2.4 [Global] 19
5.3 Accomplishments 19
5.4 Uncertainties 19
5.5 Future Work and Alternatives........ 19
6. References 20
7. Appendices 22
7.1 Appendix A (Tables) 22
7.1 Appendix B (Figures) 27

1. Introduction

1.1 Problem

Mixing cocktails and mocktails, whether at home or elsewhere, can sometimes be a difficult process. While even simple, two-ingredient cocktails exist, to have consistency when recreating them there must be precision and accuracy in measuring ingredients, and often these are liquid ingredients in small quantities, which can be easy to mess up. The average person may struggle to consistently mix drinks with the proper ingredient ratios without practice and jiggers or other measuring tools. This could lead to overpouring drinks which have more alcohol in them than expected, which can lead to overdrinking if not careful.

A solution to this issue lies in automating the process of measuring and mixing these drinks. Commercially available automated cocktail machines do exist presently, but the average consumer will almost certainly never adopt them due to high prices. Many of these products also require proprietary bottles or flavor pods. There's a clear gap in the market for an affordable, compact, fully automated drink-mixing solution that doesn't rely on proprietary consumables and gives freedom to the user. Our proposed solution is a fully automated, ESP32-controlled cocktail and mocktail mixing machine that verifies the cup presence, checks weights at the beginning and end of the ingredient transfer process to ensure accuracy, and stirs the final beverage to ensure a consistent drink every time without locking the user into a proprietary ecosystem.

1.2 Functionality

The automated cocktail mixer allows a user to select a specific drink via a simple push-button interface. Upon selection, the machine initiates a systemic verification protocol. It reads an integrated load cell beneath the central cup station to ensure a cup is present and has the capacity to hold the drink. It then polls the load cells beneath the ingredient reservoirs to verify that sufficient liquid is available to complete the recipe.

Once verification passes, the machine activates individual mechanical liquid pumps to draw liquids from standard bottles and deposit them into the user's cup of choice. The microcontroller continuously monitors the cup's load cell at ten samples per second, in order to dispense the precise volume of each ingredient within a narrow margin of error. After all of the ingredients have been dispensed, a linear actuator motor lowers a stirring arm attached to a gear reduction motor into the cup. The gear reduction motor spins for thirty seconds to stir the drink, and the linear actuator motor will retract. The system makes use of an extremely simple and intuitive LED interface in order to communicate the system status: green indicates a completed drink, white indicates a drink is in progress, and red will flag an error (for example, a missing cup or insufficient ingredient weight).

1.3 Subsystem Overview

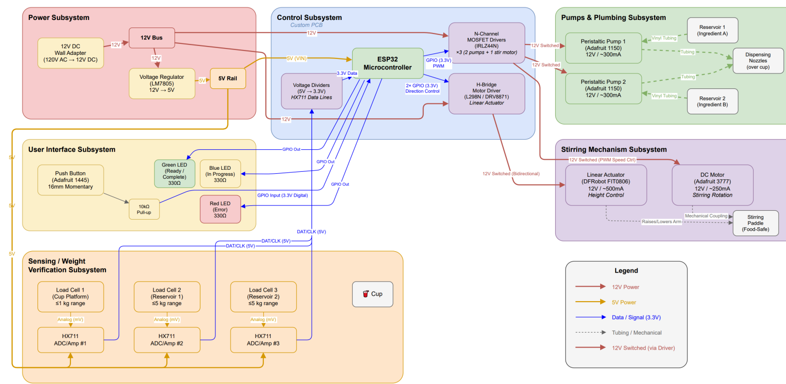

Figure 1: Full System Block Diagram

In order to achieve the full functionality and precise control required, the automated cocktail mixer is split up into six highly integrated subsystems. Figure 1 illustrates the top-level block diagram, detailing the interconnections, power distribution, and data routing between each individual subsystem.

The user interface subsystem serves as the communication bridge between the human operator and the machine. It consists of one pushbutton configured with a hardware debouncing circuit to prevent accidental activations. Visual feedback is provided by three LEDs of varying colors (red, green, and white). The interface is connected by GPIO input and output through the main microcontroller.

The sensing and weight verification subsystem acts as the feedback mechanism for the entire project, comprising of three 5 kg load cells. Each load cell is wired to a dedicated HX711 24-bit ADC and amplifier. These HX711 modules amplify the load cells' output voltages and convert it to a digital signal that can be read by the ESP32 microcontroller, in order to allow for calibration and use in the machine's operation.

The control subsystem is the central processing hub of the project, built around the ESP32-S3-WROOM microcontroller soldered to a custom-designed PCB. The control subsystem constantly and continuously evaluates the system state: processing data from the HX711 modules, interpreting the user's button press to begin the drink-making process, and generating signals to turn the motors and pumps on or off.

The pumps and plumbing subsystem is integral for the physical transport of the ingredients, consisting of two 12V DC self-priming mechanical pumps. The pumps are food-safe, easily cleaned, and operate through high-current DC power which is switched on and off via N-channel MOSFETs located in the control subsystem.

The stirring subsystem handles the stirring process of making the drink, relying on two motors. The first is a 12V DC linear actuator motor which controls the vertical extension and retraction of the stirring paddle, moving it in and out of the user's cup. The actuator is driven by an H-bridge motor driver, allowing for bidirectional current flow and letting the microcontroller control the operation of the linear actuator. A 12V DC gear reduction motor is raised and lowered by the linear actuator, which spins the stirring paddle.

The power subsystem handles and converts the power from our 12V DC wall adapter. The 12V rail feeds the heavy inductive loads (our pumps, motors, and the actuator). To power the logic components, the subsystem has a switching buck converter to step the 12V down to a 5V rail (used for the HX711 ADCs) and a linear regulator to step the 5V down to 3.3V rail which powers the ESP32 microcontroller.

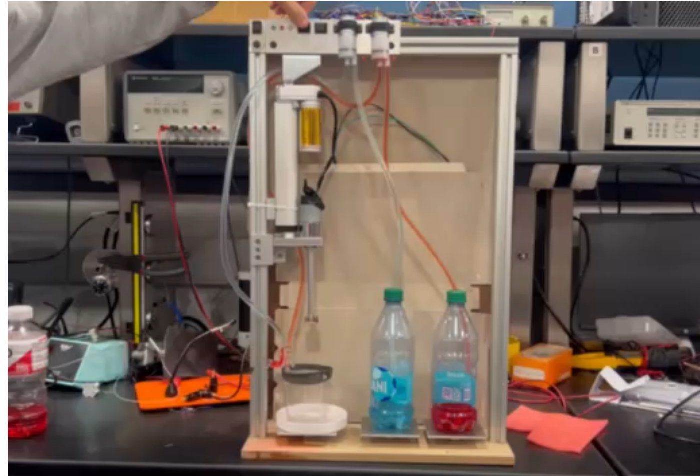

Figure 2: Automated Cocktail Maker Visual

1.4 High-level requirements

To make sure our final prototype solves our problem and proves our design viable, the automated cocktail mixer needs to satisfy the following high-level requirements.

1. The cocktail maker must dispense each ingredient to within ±5 grams of the target weight/volume specified by the recipe, as measured by the load cell beneath the drink cup.
2. The cocktail maker must detect the presence of a cup and verify that there is sufficient liquid in each ingredient container before beginning the mixing process, halting and turning on the red LED within 3 seconds if there are any issues.
3. The complete drink mixing cycle, from button press until the stirrer is retracted, should be completed in under 150 seconds for a standard two-ingredient recipe.

2 Design

The design of the automated cocktail mixer was driven by our high-level requirements of accuracy, fault detection, and efficiency. In order to achieve these goals we need a modular approach to our design.

2.1 Design Alternatives

One of the issues we encountered during initial design was the HX711 power requirements. While they function best at 5V, an issue arose where we realized that the output line of the HX711 would then also be 5V, which cannot be directly accepted by our ESP32 microcontroller which runs on 3.3V power. Two options were available to us: add more components to the board in the form of two resistors per load cell that would serve as voltage dividers for the HX711 output lines, or simply running the HX711 modules off of the 3.3V rail instead. To prioritize component efficiency and keeping any PCB redesign to a minimum (as at this point the PCB was already mostly finished), we chose to connect the HX711 modules to the 3.3V rail instead of the 5V rail. While doing so, we also made the decision to not remove the existing 5V rail, which was now only used to convert down to 3.3V. This allowed us to keep the existing components we had already ordered on the board and simply have the 5V as an additional intermediate, vestigial step.

2.2 Subsystem Descriptions, and Procedures

2.2.1 User Interface

The user interface utilizes a standard momentary tactile pushbutton. In order to prevent mechanical contact bounce that would trigger multiple extra button presses, a hardware RC debouncing circuit was implemented in lieu of a software-based solution.

The debouncing circuit utilizes a 10 kΩ resistor and a 0.1 µF capacitor. The time constant dictates how quickly the voltage stabilizes after a button press: $\tau = R \times C = 10{,}000\ \Omega \times 0.0000001\ \text{F} = 1$ millisecond. This 1 ms hardware delay smooths any electrical noise before the signal reaches the ESP32, eliminating any false triggers before the signal reaches the microcontroller extremely quickly. Visual feedback is then provided by three LEDs (red, green, and white), each of which is routed through a 330 Ω resistor that maintains a safe current through the LEDs.

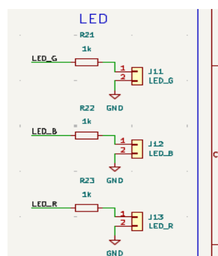

Figure 3: LED Indicator Subsystem Schematic

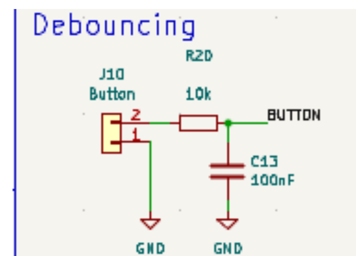

Figure 4: Debouncing Circuit Schematic

2.2.2 [Stirring Mechanism]

The stirring mechanism uses a 12V linear actuator motor for the vertical extension and retraction, along with a 12V DC gear reduction motor that serves to provide rotational torque to mix the drink. Because the linear actuator must extend physically into the beverage and retract back to its default position, bidirectional current control is required, which we achieve via a DRV8871DDA H-bridge motor driver IC.

We chose the DRV8871DDA thanks to its internal current-limiting function, which provides an additional safety net to our design against current spikes. In order to keep the current below 2 A, we used a 33 kΩ resistor such that our current would be limited to $I = \frac{64\ (V/A)}{R_{ILIM}} = \frac{64}{33} = 1.94$ A.

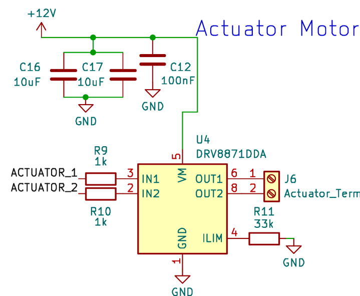

Figure 5: Actuator Motor Schematic

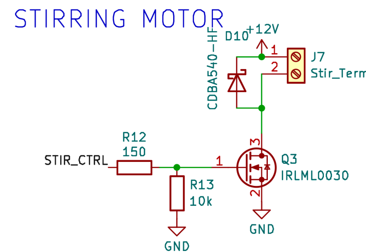

Figure 6: Stirring Motor Schematic

2.2.3 [Pumps and Plumbing System]

The plumbing system makes use of 12V mechanical pumps driven by N-channel MOSFETs which act as low-side switches. A flyback diode is placed in parallel across the pump motor terminals to safely dissipate the inductive voltage spike which occurs when the MOSFET cuts power to the motor coils.

Liquid ingredients are physically transported through ¼ inch inner-diameter, food-safe clear vinyl tubing. The motors are easily cleaned and food-safe, allowing for quick maintenance.

To achieve our required dispensing accuracy, the system relies on the flow rate of the pumps and volume of tubing. By stopping pumping when the amount of ingredient removed from the reservoir hits the requirement and not when the amount the cup increases by hits the requirement, the small amount of liquid left over in the pumps that falls into the cup will ease the cup’s weight to the requirement with great accuracy.

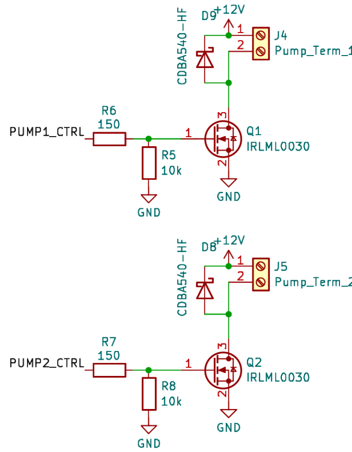

Figure 7: Pump Schematic

2.2.4 [Control Subsystem]

The ESP32-S3 microcontroller serves as the center point of the automated cocktail mixer, coordinating the interactions between every other subsystem. Our 100mm x 100mm PCB was designed to isolate the subsystems, with 12V traces physically distanced from the 3.3V logic traces to minimize electromagnetic interference. 10 kΩ pull-down resistors are also included on many of the data lines to prevent accidental triggering of pins during the ESP32 boot-up sequence. The software is built around a state machine, ensuring that the ESP32 can continuously poll the load cells at 10 Hz without interruption.

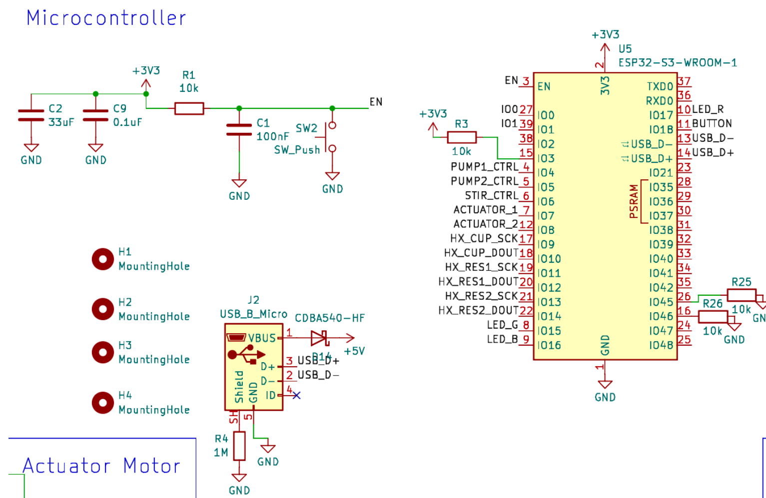

Figure 8: Control Subsystem Schematic

2.2.5 [Status and Weight Verification System]

The precision of the automated cocktail machine's automated poor relies on the three 5 kg load cells mounted underneath the cup and the two ingredient containers. Each load cell is a Wheatstone bridge-style load cell that consists of four strain gauges, producing a differential voltage output upon application of a physical load. Each load cell is wired to an HX711 module,

which is a combination ADC and amplifier. The HX711 module amplifies the signal from the load cells significantly, before converting to a 24-bit serial output that can be read and utilized by the ESP32 microcontroller.

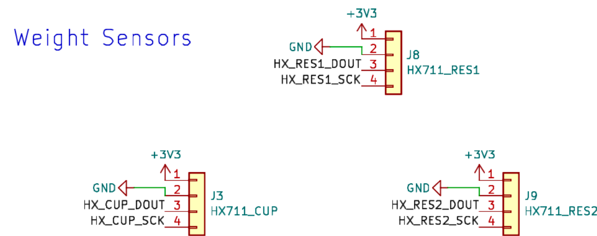

Figure 9: Weight Sensors Schematic

2.2.6 [Power System]

The system is powered by an external 120V AC to 12V DC wall adapter capable of supplying 3A. An external power brick is useful as it keeps any high-voltage AC out of the machine's chassis entirely. The 12V bus is routed directly to the high-current inductive loads of the system: the pumps and the two motors.

To step down the voltage for the logic components, an LM2596 switching buck converter is used to provide a 5V rail, which is then stepped down again by a linear low-dropout regulator to 3.3V for the ESP32. This tiered distribution of power makes sure that the logic components remain sufficiently isolated from the 12V rail that may be affected by the motors and pumps.

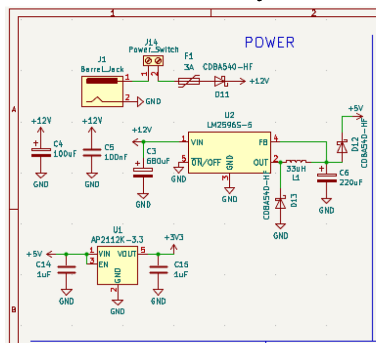

Figure 10: Power Subsystem Schematic

2.3 Subsystem Software Design

The first major design decision we made was using a finite state machine to drive the main logic behind the cocktail maker. This method seemed most logical as the cocktail maker operates in a sequence of steps or states. There is an idle state, precheck fault state, vodka pumping state, mixer pumping state, lower actuator state, stirring state, raise actuator state, success state, and a fail state. In the loop function of the program there is a switch statement which allows the machine to consistently switch between machine states based on the values of different signals. This method seemed easiest to understand in terms of writing the code, reading the code, and future debugging as the only actions that are needed in the current state can execute and are relevant to the next state.

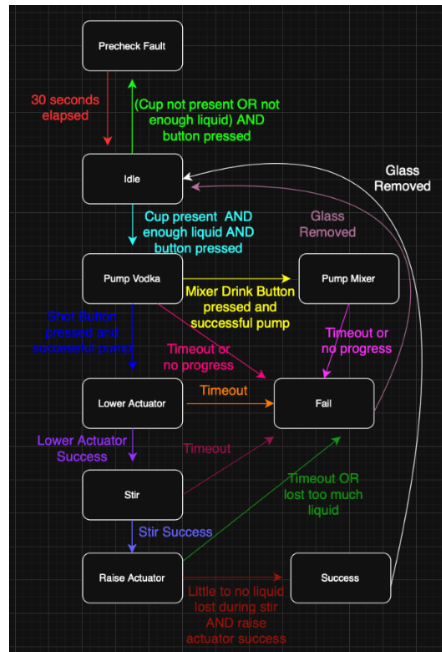

Figure 11: FSM for Control Logic

Another design decision we made that felt was necessary was the use of the ESP32 Preferences library to allow values to be stored in flash memory for future use. There are two main reasons for this. The first is it allows us to store any calibration data easily. Things like the user glass weight, liquid housing container weights, any liquid densities, etc. can all be stored so that they don't constantly need to be updated. Another reason is that in our project we have a strict rule where if power is lost during the drink making process, the drink must be scrapped. To accomplish this we must be able to know what was happening before the most recent loss of power, so upon entering a progress making state, a flag is written to flash memory and then the flag is cleared on a fail or hold state. Then on every power on this flag is checked to see if we previously were in progress of making a drink.

See Figures 13 and 14 From Appendix B

Another major design decision made was making us an interactable terminal screen/serial monitor. The serial monitor had many commands that we could run to make debugging and testing the machine easier. We could read the values of all scales which was very useful when taring and checking for proper amount of ingredients were dispensed. We could tare individual scales from the serial monitor instead of explicitly having to insert it in our code whenever we wanted to change containers and we could save all calibration values right from the serial monitor. All of these things made debugging and verification much easier.

See Figures 15 From Appendix B

The last main design decision made was to keep much of the control logic in helper functions to make the code much easier to read. Functions like setState(), updateLEDs(), handlePumpStep(), etc. contain much of the actual control logic for the system and are used in the finite state machine defined in the loop function of the program which makes reading finite state machine code much easier.

2.3.1 User Interface and Startup

In this aspect of the program, 1 button which is active low is read. The button is for a 1:2 mix drink of alcohol and mixer. We implemented debouncing using the milis() function with some stable state logic so that no false triggers occur and one long press of the button only forces a single drink request. The function itself that runs for drink verification is verifyCanStart() which checks a few things. First wall power is checked, then the weight of the cup sensor is checked against a certain value (-2 grams since cup is tared to 0) to make sure a cup is present, and then it does the same for the housing containers and ensures they have at least 120% of the required drink. If all of these are true, then this function returns true.

See Figure 16 From Appendix B

2.3.2 Weight Sensor Calibration

Since we must use 3 weight sensors for the automated cocktail maker, we need 3 HX711s. These return a different raw value other than grams. To convert them to grams we used a simple linear equation to convert to grams. However, during assembly we accidentally soldered the HX711s backward so the order of offset and raw is reversed from what is expected.

$$grams = \frac{offset - raw}{scale}$$

Here grams are the weight in grams, raw is the HX711 value, offset is the value of the glass/container in the HX711 raw units and scale is the number of raw units per gram. Equation is used in a rawToGrams() function. To also make this system less noisy, a function averageRaw() is used to get the average raw value over several samples. These functions are used together to read the weight of the user glass sensor, vodka sensor, and mixer sensor. There are also some other functions used for calibration like tare functions which are used to find the offset of the user glass, etc. and other functions which may print values read for debugging purposes.

See Figures 17 and 18 From Appendix B

2.3.3 Pump Control Logic

The control of the pump is primarily handled by two functions startPumpStep() and handlePumpStep(). The first of the two functions sets an active pump, reads the starting weight of both the glass and the liquid housing container, notes some values to be used later in progress checks and then turns on the required pump. Then the second function does quite a few things. The main thing it does is consistently check the gain in weight from the user glass and the loss of weight from the liquid housing container and how much these values differ. With the idea behind this check that theoretically the amount of liquid lost from one should be gained in the other with a little bit of error for liquid currently in the tube. The above values and others are then used to ensure no faults occur during pumping. The first two errors which cause an immediate stop are a loss of power or removal of the glass. A third fault is based on the mismatch value as if that value gets too high then something is wrong so stop. Also, if there is a maximum timer for how long the pump can run and if this timer is exceeded there must be some issue. Lastly, there is a consistent progress checker that updates every few seconds to make sure progress is being made

and there isn't some error like the pump is running slow, a kink in tubing, etc. Having all these different fault checks ensures that the transfer process is very robust and will most certainly be accurate as this is the main area where a malfunction could cause issues. Both pumps also stop once the user glass gains a certain amount of liquid or the housing container loses a certain amount of liquid. While theoretically these should be the same there will be some liquid in the tubing and some of that liquid once the pump turns off will end up falling into the user glass and some into the housing container. Once the pump is off and the liquid has settled, a final weight check is run to confirm liquid transfer.

See Figure 19 From Appendix B

2.3.4 Linear Actuator and Gear Reduction Motor Control Logic

After liquid has been transferred the linear actuator motor is driven downward by a timer. Currently this timer is just a computed value for how long it should take to get to go the full 6 inches plus some buffer time. The equation used for this number is

$$time = \frac{distance}{velocity}$$

Currently this number is 12 seconds and once we get our machine, we will test the travel time. The extension/retraction of the linear actuator is handled by two functions each. One called actuatorExtend/Retract() and another called handleActuatorExtending/Retracting(). The latter is responsible for tasking the actuator to move while the handler function calls the previous and does other small checks like for power and a timeout check. Once lowered our timer based stirring mechanism is started called handleStirring(). All this function does is check for power then turn on the gear reduction motor for a set time. Once complete the linear actuator is raised and a final check is done. This final check compares the pre-stirring weight to the post stirring weight to ensure not too much liquid was lost.

See Figure 20 From Appendix B

2.3.5 Status and Fault Handling

This code corresponds physical LEDs to the current machine state to display progress to the user. A function updateLEDs() is updated based on the current machine state. The red LED is on when in precheck fault state or the failed state. The green LED is active when in the success state. The blue LED is on in any of the in-progress states. If in the default state one of the LEDs are on.

See Figure 21 From Appendix B

3. Verification

3.1 User Interface

To test and verify the functionality of our user interface prior to integrating it into our full PCB, a miniature simplified breadboard prototype of the entire system was created. The schematic for this system can be seen in Appendix B, Figure 22 and its purpose was to simulate and verify the interaction between the buttons RC debounced input signal, the LED status indicator subsystem, and miniature simplified system. Altogether, the breadboard test setup included the RC debouncer connected to a push button, an ESP32 DevBoard microcontroller, a switch which simulated the weight sensors input for the cup detection, three different colored LEDs for system status indicators, and an NMOS which drove a small motor and represented the mixing system.

Several tests were performed to verify that the design worked from there. When the system first powered on, the green LED turned on to indicate that it was ready and powered. If the button was then pressed while the switch representing the sensors indicated no cup, then the system indicated an error by turning off the green LED and turning on the red LED. This state lasted for only a few seconds, but during it all inputs are blocked. After this delay the system returned to the ready state and turned off the red LED and on the green LED. That confirmed that the system could prevent operation when no cup was present and clearly inform users that there was some sort of issue.

If the switch indicated that there was a cup present though, then pressing the button made the system enter its mixing state. During this state, the green LED turned off, and the blue LED turned on. The motor was then activated in a timed mixing sequence where it would run for a few seconds, turn off briefly, and then run again before stopping completely. After this sequence was completed, the blue LED would turn back off, and the green LED would turn back on to indicate the system was ready for use again. This test thus confirmed for us that not only did the UI correctly communicate with itself and the control, but it also confirmed that the control subsystem could easily communicate with the motor subsystem.

Beyond just the functional testing of this design, the button debouncing circuit was also verified more in depth using an oscilloscope. Using a single sweep the voltage signal from the button was observed when pressed, and the recorded signal (seen below in Figure 12) showed a clear and clean transient line rather than some bouncing signal which is typical of mechanical switches. By seeing that result in the oscilloscope, it confirmed the RC debouncing circuit was successfully filtering out the buttons signal and making sure that each actual button press produced a single clear signal to the microcontroller.

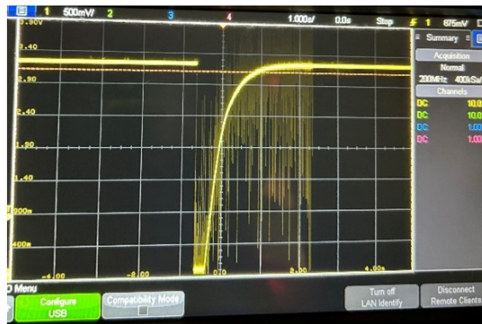

Figure 12: Oscilloscope View of Button Signal

Overall, this breadboard prototype thus successfully verified that the LED indicator subsystem, button debouncing subsystem, pseudo weight sensor input, and motor subsystem all interacted. Once implemented into the actual PCB, we saw that this held true too, and that all the requirements laid out in Table 1 of Appendix A were met.

3.2 Stirring Mechanism

When going about the verification process of the Stirring Mechanism, the requirements listed in Appendix A Table 2 can be summarized as follows: the linear actuator must give enough vertical clearance so the paddles about the users glass, the mixing motor must spin between 50-100 RPM, the process should take no longer than 45 seconds, and the stirring material should be food safe and fit in a normal cup. Luckily all these requirements are easy to verify even prior to running any sort of physical test with them. The vertical clearance was determined by the linear actuator purchased, so we just had to ensure that it was placed at a height of six inches above the cup, plus the clearance required for the actual stirring rod. Similarly, by purchasing an appropriate gear reduction motor rather than a different type, we were able to always maintain a set speed of 85 RPM whenever it was on, clearly staying in the desired range. The final physical component was the small stirring rod attached to the gear reduction motor, and by having this made from stainless steel, we could confidently say that both the size and material requirement of it was met. This only left the time constraint of 45 seconds to be tested when the full system was constructed, but even that was controlled primarily by our codes logic. Nevertheless, after connecting all components of our system and counting the amount of time that it took for the whole stirring process to take place, we found that everything not only functioned, but we remained within the time constraint at 44 seconds.

3.3 Pumps and Plumbing

While running tests of the pump and plumbing subsystem, the primary requirements as seen in Table 3 of Appendix A related to the pump's performance and functionality. Thus, the first thing we as a group had to do while testing our pumps was to ensure they had both a flow rate of at least 100 mL/min and that they were self-priming. However, the first pumps acquired by our group were spare ones that the machine shop had given us, so we needed to see if they were even functional. To do so, we originally connected the pumps one by one to the previously described UI breadboard in place of the motor, however neither pump ever fully activated. Instead, a small clicking noise was heard, so we knew then that a different set of pumps had to be

purchased. Once we did so, we were able to confirm that both functioned properly and move back to confirming the previously stated performance requirements.

The first and easiest part to test was to ensure the pumps are self-priming. This meant that the pumps didn't need to be submerged and could instead begin pumping liquid even if there was air between the pump and itself, and was easily seen functioning once the pumps were turned on. The second performance requirement that played a major part in meeting our high-level time requirement was that each pump must deliver a flow rate of at least 100 mL/min, and when timed and measured we found that each pump was going far beyond that and pumping at around 1150 mL/min.

With the performance of the pumps being confirmed, we then had to verify that the plumbing system was functionally user friendly. This meant that the tubing had to be easily removable for cleaning, or in other words, that it can easily be removed within 30 seconds such that it doesn't require any other components. Since our project isolated all of the electronic components from the plumbing system, and the tubing of the plumbing system was exposed from the front of the project, these requirements were also all verified.

3.4 Control

All of these tests were first done as simulations. Before testing them with the actual weight sensors and those values, we first hardcoded the values in the program and ensured proper functionality. After this worked, then we ran tests with the calibrated weight sensors. The main scenario we wanted to test was if the startup check could work as expected. So, the first way we did this was to manually set the cup weight value to over -2 grams, the ingredients values to over 120% of the amount needed for a drink, a button was pressed, and the power good signal was also set to high. We then had a print statement that would run if the program made it to the next state as expected. The print statement did show up in the terminal meaning we reached the next state (pump vodka state). Then we tried every other combination of not having all these conditions met and saw what happened by putting a print statement in the precheck fault state. Again, we noticed this print statement worked every time, thus the startup process functioned as expected.

Another test we did was removing the cup during many different steps of the drink making process. We would force the program to be in a state where the blue LED should be on (drink making process) and then after a certain amount of time running, we would force the weight cup sensor to be zero. Then we would print things like the red LED value to see if the fail state was entered. On every drink making state (pump vodka, pump mixer, lower actuator, stir, raise actuator) the red LED value was set to high proving the fail state was entered.

We tested the pump progress failure also. To do this all we had to do was keep the cup weight reading constant. Once a certain interval had passed, we were able to read the red LED value. Since progress is expected to be made during the time of pumping, since none was made the fault was thrown. There were many other test we ran of this sort where we would simulate certain scenarios and see if the expected output occurred. In all test we ran, the expected output

was observed whether that was by print statements we inserted or observing expected signal value.

Lastly, every time we read new values from the weight sensors while our liquids were pumping, our program would print all values it read. Then we would time how long the pumps were on. Once we had how long we were pumping, we counted the number of times the values were printed to the serial monitor and made sure then when dividing this number by the amount of time the pumps were on that we got a number greater than 10 since that would tell us we were polling the weight sensors at least 10 times per second.

3.5 Status and Weight Verification

Specific requirements for the Status and Weight Verification subsystem can be found more in depth in Appendix A Table 5. Overall, these requirements ensured that the load cells accurately measured values in grams and that they could handle the required higher weights associated with filled containers of liquid. First, the task of the sensors handling higher weights was easily completed by purchasing sensors with the appropriate ratings. From there the primary focus when verifying this system was to ensure that accurate measurements were taken in grams.

To do so, we had taken two third party scales that both accurately returned the same known weight of an item. We then used this known weight to figure out what the conversion factor was from the raw code into grams by taring the scale, weighing the item, and then dividing the known and measured values by each other. Once that value was calculated and implemented into the code, the program was ran several times subsequently with various other known weights to ensure that it was accurately reading in grams each time. We also tested that the system accurately functioned regardless of viscosity of the liquid being pumped through it by ensuing that the weight sensors values were constantly being checked, and that even when we swapped with a different beverage the system relied on the measured values.

3.6 Power

To verify and test the functionality of the power subsystem tests were done on the mostly assembled PCB. Since the barrel jack for the AC to 12V DC wall adapter was neither soldered nor available, the 12V input had to instead be supplied directly from a regulated bench power supply. This connection was made such that it bypassed both the barrel jack and power switch portion of the circuit but still allowed to verify the voltage regulation and power distribution through the circuit by being connected via the 12V rail and a random ground.

As it was just stated, the point of this test was to verify the function of the power subsystem by confirming that it produced the three necessary voltages (12V, 5V, and 3.3V) and distributed them across the circuit accordingly. So once the supply voltage was connected, a multimeter was used to measure the three different voltages at multiple locations on the board for each voltage rail. The reason we did this was to ensure that all the voltages were not only being generated correctly, but that they were also being evenly distributed throughout the circuit. In the table below are the results:

Expected Voltages	12.0 V	5.0 V	3.3 V
Measured Voltages	12.007 V	4.766 V	3.294 V

Table 9: Power Subsystem Measurements

From these results we can see that the measured 12V rail voltage of 12.007V and the measure 3.3V rail voltage of 3.294 are both within a very tiny tolerance of the exact expected values. This confirms that the input power is being distributed across the board correctly and confirms that the voltage regulator is functioning correctly and supplying the correct voltage to the microcontroller and sensors.

Unfortunately, the 5V rail sees a voltage of 4.766V which is slightly lower than the expected 5V value and a bigger difference than the previous two measurements. However, upon calculating the percent difference we found that it is 4.68% difference and thus falls within the acceptable range for most components. This voltage drop might be because of voltage drop across other components, trace resistances, or even the operation of the buck converter under certain load conditions. Whatever the cause, moving forward this will be further investigated during the full system testing when the system is powered by through the barrel jack and when more subsystems get connected and draw current.

Knowing that this test mostly verified the subsystems' ability to convert the 12V input into usable 5V and 3.3V outputs and distribute them accordingly, we can confirm its readiness for the next steps. This would include installing the barrel jack and plugging in all the external components the machine shop currently has (mainly the power switch here) and doing more tests to ensure the full power input/output portion is functioning correctly. If voltages drop further or instability occurs, then additional filtering capacitors or regulator adjustments may be implemented as a contingency plan. Future tests might also include checking that voltage rails can handle the full system load, check voltage stability, and checking any noise to make sure sensitive electronics can still reliably operate.

3.7 Full System

Once all individual subsystems and components were tested, we put everything together with no constraints to allow the machine to function as expected. We ran multiple attempts with ideal and non-ideal conditions and in every scenario the machine either functioned or didn't function as expected (and threw the expected error message). Additionally, the whole process was completed in a much shorter amount of time (54 seconds) than we originally allotted (150 seconds). Once we knew that the whole system functioned correctly on a high level, we began testing the required portions. We saw in every test that we ran that the actual weight and the expected weight were always within our 5g threshold as you can see below in Table 10.

	Start Weight	End Weight	Amount Withdrawn	Expected Portion	Percent Difference
Alcohol	434	386	48	45	6.7%
Mixer	512	423	89	90	1.1%

Table 10: Final Performance Measurements

4. Cost

In our team contract we spell out that each team member is required to work on the project for at least 8 hours per week. Through the first couple weeks it is apparent this is an underestimate of the actual time spent per person on the project for the week and the real number is probably around 12 hours per person per week. Some weeks it will be less and on deadline weeks it will be more, but for a weekly average 12 is pretty accurate this far. Also, as the project gets toward its completion, we imagine the work will ramp up a bit so an extra 30 hours total will be added per partner. We will say these 12 hour weeks started the week the initial project proposal was due so (These 12 hour weeks will be done for 12 weeks total).

$$hours\ per\ partner = 12\frac{hours}{week} * 12\ weeks + 30\ hours = 174\ hours\ per\ partner$$

$$total\ group\ hours = 3\ partners * 184\frac{hours}{partner} = 522\ total\ group\ hours$$

With each of our expected hourly rates after graduation being around $40/hour we can use this to get a total cost of our own labor. See table 8 in Appendix A for the schedule used throughtout the semester.

$$group\ labor\ cost = 522\ group\ hours * 40\frac{dollars}{hour} = \$20{,}880\ in\ group\ labor$$

Aside from our own labor, we also talked to the machine shop about having them do some of the assembling for us and creating the rigid body of our cocktail maker. For these numbers we will assume it will take 2 machine shop workers 3 days to complete this aspect of our project. Assuming they make about $35/hour we can get a total cost for machine shop labor.

$$machine\ shop\ labor\ cost = 2\ workers * 3\frac{days}{workers} * 8\frac{hours}{day} * 35\frac{dollars}{hour} = \$1{,}680$$

From this we can get our total cost of labor by summing up the group and machine shop labor costs

$$total\ labor\ cost = \$20{,}880 + \$1{,}680 = \$22{,}560$$

In Table 7 from Appendix A the full parts list and the cost breakdown associated with each component can be found. The total cost yielded from these calculations was $223.04. Combining this with the previously calculated total labor costs, we find that the final total cost of production of this product ended up being $22,783.04.

$$total\ cost = \$22{,}560 + \$223.04 = \$22{,}783.04$$

5. Conclusion

5.1 Ethics

The first lens to look through when evaluating this design is an ethical one. The IEEE and ACM Codes of Ethics state that engineers should prioritize public safety and welfare, be honest about a systems limitation, avoid any deceptive practices, and have systems designed to minimize the risk of harm. Keeping all this in mind when reviewing the automated cocktail/mocktail makers design, there are two primary ethical responsibilities which stand out the most here.

Our first, and most obvious, ethical consideration is the responsible use of the product as it relates to its alcohol pouring abilities. While this system can dispense and mix alcoholic drinks, it should not promote any sort of unsafe or excessive consumption practices or misrepresent a drinks strength. To combat this, what our system will do is use weight sensors to measure out ingredients and effectively limit the amount of each liquid added to a cup within some defined tolerance. What this does is prevent any sort of accidental overpouring and can easily be tracked by documenting dispensing tolerances or accuracy. This sort of documentation, when all said and done, also provides users with a clear understanding of the system's limits so that they understand the measurements are approximate and should be treated as such. Also, with this device being capable of dispensing alcoholic beverages, we must consider the fact that there is a chance of misuse. Although we might not be able to eliminate intentional abuse of the system, or determine who may use it post purchase, it is our responsibility to make sure our design will minimize risk through quantity limits and controlled dispensing logic, and that we make it clear that this is intended strictly for adult use. While we acknowledge the fact that the user ultimately determines how the system is used, on the designer side it will not bypass any sort of legal safeguards regarding alcohol consumption and may require age verification measures in the future.

The second main ethical responsibility to consider is that of honest performance reporting. When testing and validating the dispensing accuracy of our machine, it is crucial we do so repeatedly and experimentally to report proper measured errors rather than ideal or theoretical values. By doing this we are letting users know that there is a chance their drinks do not reach a level of commercial certification accuracy, but that if anything it is quite close. Together, all of this aligns with the IEEE principle of truthful representation of system capabilities and limitations, as it makes it clear to users that the result will consistently be within some specific range of values. With the incorporation of the weight sensors and control software too, it also will follow fail-safe principles that are consistent with IEEE and ACM ethical guidelines. On the topic of ACM principles, since our system does not collect, store, or transmit any sort of personal data, and thus no behavioral tracking occurs besides the immediate functional operation, it easily remains in compliance regarding privacy and responsible data stewardship. Besides this, we will all follow good engineering practices by documenting our design decisions, testing procedures, and any failures to avoid misleading users about the system's reliability.

5.1.1 Specific Applicable Engineering Standards

- IEEE Code of Ethics
- IEEE 1012 – Verification and Validation
- IEEE 12207 – Software Lifecycle Process
- ACM Code of Ethics
- ACM 1.2: “Avoid Harm”
- ACM 1.6: “Respect Privacy”
- UL 62368-1 – Safety of Electronic Equipment
- FDA Food Contact Material Regulations (21 CFR 174-178)

5.2 Societal, Economic, Environmental, and Global Impact

5.2.1 [Societal]

Overall, this project contributes societally to public welfare by promoting consistent and responsible beverage dispensing, all while reducing human error commonly associated with manual alcohol measurements. Thus, in a societal context, our engineering solution creates an affordable automated drink mixer capable of improving consistency and accessibility in beverage preparation, while ideally helping to reduce measurement errors commonly present with manually pouring. On top of that, this is done without relying on some sort of proprietary consumable. At the same time, however, this sort of automated alcohol dispensing system could be misused or abused if the right safeguards aren’t in place. To reduce this risk, our design makes sure it requires user input per drink, enforces specific measured quantities, and prevents any continuous or uncontrolled dispensing or liquids. By integrating these safeguards, along with transparency in the systems abilities, this results in a design which aligns perfectly with engineering responsibilities prioritizing safety, reliability, and public welfare.

5.2.2 [Economic]

Looking at this solution from an economic perspective, we found that many of the existing products that function like our design are quite expensive and rely on using their branded consumables or pods. Our approach, emphasizes low costs. The machine itself comes with standard containers and no requirements for some proprietary additional components. Instead, once a consumer has the machine, they can select and purchase whichever liquids they want, from whatever provider they want supporting user accessibility and flexibility.

5.2.3 [Environmental]

Compared to most existing drink mixing machines, our engineering solution is more environmentally conscious. Most existing designs use disposable pods or cartridges, while our system is designed to reuse some standard plastic containers. On top of that, the system will run at a low power level to ensure it doesn’t use too much energy while plugged in to effectively minimize its carbon footprint. Lastly, if a user would like to dispose of the product at the end of its lifetime, all the materials on it should be safe to recycle.

5.2.4 [Global]

In theory, it can be argued that the underlying technology behind our design could in fact have a larger global impact. Since this is ultimately a low cost and closed loop fluid dispensing system, it could potentially have broader applications in both mixology education and even small-scale automation which goes beyond just beverages. Ultimately, with a transparent and well documented design process then it becomes easy for a user to understand the method by which these drinks are made and allows others to find easier ways to adapt the design for other contexts.

5.3 Accomplishments

The automated cocktail mixer successfully realized its main goal of creating and mixing cocktails through a fluid dispensing system. The final prototype successfully verifies cup presence, continuously verifies ingredient weight and transfer, and consistently hits our dispensing target masses. In addition to this, our custom PCB proved to be robust, managing high current loads from our motors and pumps without any operational issues to the logic lines. The system completed the drink making cycles in about 60 seconds, which is significantly under our initial requirement of 150 seconds.

5.4 Uncertainties

While the system met the operational requirements initially set out, testing revealed a critical flaw. When fluid is left in the bottles while tubing is fully primed, a hydrostatic pressure differential is created. Because the pumps don't provide 100% complete pinching of the tubing when the motor is at rest, there's an unintended siphon effect that causes the ingredient to drip into the cup slowly, eventually completely filling and potentially even overfilling the cup and drip tray.

This is a significant design flaw for the machine's idle state. After being left idle for several days with full ingredient bottles, this drip effect managed to overflow a 200 mL cup and resulted in a spill.

5.5 Future Work and Alternatives

To address the unintended siphoning effect and idle leaking observed during testing, one alternative to our current system for future iterations should be the integration of one-way valves of some kind along the fluid routing, which would seal the tubing when the pumps are inactive, eliminating unintended drip leakage.

Another consideration would be expansion of the current user interface. Aside from just adding more buttons, a touch screen interface or even a local web server through the ESP32's native Wi-Fi capabilities would allow for user self-programming through the device's screen or through a phone or computer browser. This would allow users to input their own drink mixes or access a database of existing mixes, plus allow for many more options to be stored on the device. We would also integrate a third pump and ingredient, and maybe a modular system allowing users to add more than 3 ingredients.

6. References

[1] “16mm Panel Mount Momentary Pushbutton - Red,” *Jameco.com*, 2026. https://www.jameco.com/z/1445-Adafruit-Industries-16mm-Panel-Mount-Momentary-Pushbutton-Red_2505367.html?srsltid=AfmBOor2LvGh0oYWIiMRRyUJhcb-uN6TiAP50JI4_iOm_EOFffk_liO8 (accessed Feb. 12, 2026).
[2] “Product Overview Dfrobot fit0806.,” *Mouser.com*, 2026. https://www.mouser.com/pdfDocs/Productoverview-DFRobot-FIT080x (accessed Feb. 12, 2026).
[3] “Metal DC Geared Motor w/Encoder -12V 83RPM 45Kg.cm SKU: FIT0185.” Accessed: Feb. 26, 2026. [Online]. Available: https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/2280/FIT0185_Web.pdf (accessed Feb. 12, 2026).
[4] “Welcome To Zscaler Directory Authentication,” *Homedepot.com*, 2026. https://www.homedepot.com/p/Everbilt-1-6-in-I-D-x-1-4-in-O-D-x-10-ft-Clear-Vinyl-Tubing-T10006003/304185157 (accessed Feb. 12, 2026).
[5] Adafruit Industries, “Peristaltic Liquid Pump with Silicone Tubing - 12V DC Power,” *Adafruit.com*, 2026. https://www.adafruit.com/product/1150#description (accessed Feb. 12, 2026).
[6] “ESP32-S3 Datasheet,” *Espressif.com*, 2025.https://documentation.espressif.com/esp32-s3_datasheet_en.html
[7] “Documentation | KiCad,” *Kicad.org*, 2024. https://docs.kicad.org
[8] Adafruit Industries, “Strain Gauge Load Cell - 4 Wires - 5Kg,” *www.adafruit.com*. https://www.adafruit.com/product/4541 (accessed Feb. 25, 2026).
[9] “PARALLEL BEAM LOAD CELL.” Available: https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/TAL220M4M5Update.pdf
[10] Avia Semiconductor, “AVIA SEMICONDUCTOR 24-Bit Analog-to-Digital Converter (ADC) for Weigh Scales DESCRIPTION,” 2009. Available: https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/hx711_english.pdf
[11] IEEE, “IEEE Code of Ethics | IEEE,” Ieee.org, 2020. https://www.ieee.org/about/corporate/governance/p7-8
[12] Association for Computing Machinery, “ACM Code of Ethics and Professional Conduct,” *Association for Computing Machinery*, Jun. 22, 2018. https://www.acm.org/code-of-ethics
[13] C. for F. S. and A. Nutrition, “Packaging & Food Contact Substances (FCS),” *FDA*, Apr. 07, 2020. https://www.fda.gov/food/food-ingredients-packaging/packaging-food-contact-substances-fcs
[14] UMW “LM2596 /LM2596HV Series 3A Step-Down Voltage Regulator,” report, 1996. [Online]. Available: https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/2577/UMW%20LM2596_HV.pdf

[15] “Pin Assignments (Top View) (Top View),” 2017. Available:
https://www.diodes.com/assets/Datasheets/AP2112.pdf
[16] “DRV8871 3.6-A Brushed DC Motor Driver With Internal Current Sense (PWM Control).” Accessed: Feb. 26, 2026. [Online]. Available:
https://www.ti.com/lit/ds/symlink/drv8871.pdf?HQS=dis-dk-null-digikeymode-dsf-pf-null-wwe&ts=1772080563172&ref_url=https%253A%252F%252Fwww.ti.com%252Fgeneral%252Fdocs%252Fsuppproductinfo.tsp%253FdistId%253D10%2526gotoUrl%253Dhttps%253A%252F%252Fwww.ti.com%252Flit%252Fgpn%252Fdrv8871
[17] Qw-Jb021, “SMD Schottky Barrier Rectifiers Mechanical data CDBA540-HF Thru. CDBA5200-HF Maximum Ratings and Electrical Characteristics.” Accessed: Feb. 26, 2026. [Online]. Available:
https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/506/CDBA540-HF-CDBA5200-HF_RevC.pdf
[18] “282837-2 | DigiKey Electronics,” *DigiKey Electronics*, 2026.
https://www.digikey.com/en/products/detail/te-connectivity-amp-connectors/282837-2/2187973?gclsrc=aw.ds&gad_source=1&gad_campaignid=20470400566&gbraid=0AAAADrbLljRaE7alnZU3wfdQisXOeGjW&gclid=CjwKCAiA2PrMBhA4EiwAwpHyCw2AutLYl4O6dTYM0ikrU1hEjJiRNzUhcHsorWemm8vW9IvOvETOhxoC3aQQAvD_BwE (accessed Feb. 26, 2026).
[19] Online catalog switchcraft, inc. 5555 N. Elston Ave. (n.d.).
https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/7139/137_.25_20in,phone_20jacks.pdf

7. Appendices

7.1 Appendix A (Tables)

Table 1: User Interface Requirements and Verification

Requirements:	Verification:
• When plugged in, the user presses a button, and an LED status indicator will light up notifying the user of success or failure.	• Ensures that the user glass is in position by reading an appropriate value from the load cell under the user glass. • Checks the values of load cells for each ingredient needed for the cocktail to make sure there is enough for the requested drink.
• The red LED lights up if any issue occurs before or during the drink making process. • The blue LED lights up when the cocktail maker is in the process of making a drink.	• Check if the two steps in the first requirement fail/succeed in producing the wanted values. • Check if the values from the liquid housing containers or the user's glass are changing at the expected rates when dispensing the liquids. • Make sure the linear actuator goes to its work and home position properly.
• The green LED lights up when a drink has been made. • The green LED turns off after the user glass has been removed from a successful drink.	• Check all status bits for successful values • Check if the linear actuator is in its home position. • Check if the blue LED is currently on to turn on green. • Check if green LED is currently on and load cell under user glass returns no weight to turn off green.

Table 2: Stirring Mechanism Requirements and Verification

Requirements:	Verification:
• The linear actuator must give enough vertical clearance (at least 50 mm) to move the stirring	• Buy a linear actuator rated for the required length (must be exactly what is needed since linear actuators only go to home and work positions).

paddle from above to into the user glass.	• Measure travel distance to ensure linear actuator performs as expected.
• The gear reduction motor must rotate the stirring paddle at 50-100 RPM to mix the drink with spilling during mixing process.	• Buy a gear reduction motor rated for 50-100 RPMs. • Test independently that motor functions at correct RPM • Test for max RPM that can be used without spilling. • Test for needed time for thoroughly mixed drink.
• The stirring process must take no more than 45 seconds.	• Make sure the raising and lowering of the linear actuator does not take too long for product bought. • Ensure that the gear reduction motor we buy has enough RPM to mix the drink thoroughly in allotted time. • Give extra buffer time for the transmission of signals. • Time whole process to make sure standard met.
• The stirring paddle must be made of food-safe material and small enough to fit in a standard cup.	• Check food safe materials list and buy a paddle made of those materials. • Take measurements of glass circumference and make sure paddle doesn't make contact when rotating around the user glass.

Table 3: Pumps and Plumbing System Requirements and Verification

Requirements:	Verification:
• Each pump must deliver a flow rate of 100 mL/min. • The pumps must be self-priming.	• Check the ratings for the pumps to make sure they achieve at least 100 mL/min and are self-priming. • Test the pumps independently to make sure they work as required.
• Tubing must be easily removable for cleaning.	• Test to make sure tubing can be removed and installed within 30 seconds. • Ensure that to remove tubing no other component besides the tubing needs to be touched. • Have specific holes/notches for tubing so they "snap" into place when installing them.

Table 4: Control Subsystem Requirements and Verification

Requirements:	Verification:
• The ESP32 must poll button input and read all weight HX711 channels at a rate of at least 10 samples per second.	• Configure the HX711 modules in the 80 SPS mode. In the main loop, read all three channels sequentially and print timestamps over serial. Confirm each channel produces ≥ 10 new readings per second.

• The PCB must include an H-bridge driver with control inputs from two ESP32 GPIO pins for the linear actuator.	• With the actuator connected, set ACTUATOR_1 high and ACTUATOR_2 low, confirming the actuator extends. Reverse the signals and confirm the actuator retracts.
• If the user glass is ever removed during an in progress state then the machine stops making a drink, turns on the red LED.	• Test in every state that when removing the user glass the pumps/motors signal goes low and stops and the red LED signal turns on.

Table 5: Status and Weight Verification Subsystem Requirements and Verification

Requirements:	**Verification:**
• The cup platform load cell must support about 1 kg to account for both a cup and up to 350 mL of liquid.	• Test the weight of the user glass and a little over 350 mL of liquid on multiple scales to ensure weight does not go over 1 kg and cross check this with read load cell value.
• The ingredient container load cells should have 5 kg capacity to support larger ingredient quantities.	• Test the weight of the liquid housing containers full of liquid on multiple scales to ensure weight does not go over51 kg and cross check this with read load cell value.
• The cup detection threshold should be set at -2 g, and the system should have the capability to distinguish between an empty platform and the -2 g threshold.	• Program the threshold value to -2g on the user glass load cell. • Make sure the sensor is tared when the cup is present so when the cup is on the scale it reads 0g. • Test items around cup weight on load cell to guarantee accuracy.
• Each ingredient container load cell should be able to verify that the container contains at least 120% of the required ingredient weight.	• Program the required amount of liquid in each housing container to be 120% of a single cocktail requirement. • Test liquids across multiple scales to make sure load cell gets an accurate reading.
• During dispensing, the microcontroller must read the cup load cell and the ingredient load cells to stop the pump within 5 g of the target weight.	• Program the pumps to stop slightly early to allow certain liquid not currently in glass to fall from tubing. • Measure weight of actual stop time versus after extra liquid falls into glass to get exact stop weight.

Table 6: Power Subsystem Requirements and Verification

Requirements:	**Verification:**
• The 12V DC wall adapter must supply consistent voltage to support the simultaneous	• Use a multimeter to measure the voltage the wall adapter converts to and make sure we read 12V.

operation of one pump, the linear actuator, the rotary motor, and overhead for the rest of the system.	• Measure each component individually to make sure it is getting the proper current required.
• The 5V voltage regulator must supply 5V to power at least three HX711 amplifier boards and provide power to the ESP32 microcontroller.	• With a multimeter measure the voltage from the voltage regulator to ensure 5V is provided to the 3 HX711s and the ESP32 microcontroller.

Table 7: Parts List

Description	Manufacturer	Quantity	Price Per	Link
12V 6” 220 lbs Linear Actuator	RVMARINEPAT	1	$39.99	Link
12V 85 RPM Gear Reduction Motor	Greartisan	1	$14.99	Link
ESP32-S3-WROOM *	Hosyond	1	$18.99	Link
Micro USB Connector *	Molex	1	$0.93	Link
AC/DC 12V Wall Mount Adapter	Tri-Mag LLC	1	$12.19	Link
3 Amp Fuse	Littelfuse Inc.	1	$2.37	Link
Red, Green, Blue LEDs	Gebildet	1	$9.99	Link
Drink Selection Buttons *	C&K	2	$2.84	Link
Push Buttons*	Same Sky	2	$0.10	Link
Power Switch *	E-Switch	1	$0.57	Link
Screw Terminals *	Amphenol Anytek	5	$0.64	Link
5 kg Load Cell	Adafruit Industries	3	$3.95	Link
Self-Priming Pumps *	BRINGSMART	2	$18.97	Link
¼” Clear Tubing (9.8 ft)	Yesallwas	1	$5.99	Link
Voltage Regulator (LM2596S-5)	UMW	1	$3.32	Link
Voltage Regulator (AP2112K-3.3TRG1) *	Diodes Incorporated	1	$0.22	Link
Barrel Jack	Switchcraft Inc.	1	$11.39	Link
H-Bridge Motor Driver (DRV8871DDA)	Texas Instrument	1	$2.73	Link
HX711 Amplifier	Soldered Electronics	3	$5.45	Link
PCB Board (100mm x 100mm) *	SchmalzTech LLC	1	$6.49	Link
33uH Inductor	Bourns Inc.	1	$0.67	Link
680uF Capacitor (THT)	Panasonic Electronic Components	1	$0.82	Link
220uF Capacitor (THT) *	Nichicon	1	$0.44	Link
100uF Capacitor (THT) *	Nichicon	1	$0.60	Link
Capacitor – 0.1uF/50V (0805) *	KYOCERA AVX	6	$0.35	Link
Capacitor – 33uF/10V (0805) *	TDK Corporation	1	$0.87	Link
Capacitor – 1uF/25V (0805) *	Murata Electronics	2	$0.16	Link

Capacitor – 10uF/50V (0805) *	Murata Electronics	2	$1.19	Link
Diode – CDBA540-HF (DO214AC) *	Comchip Technology	7	$0.45	Link
MOSFET – IRLML0030TRPBF (SOT23) *	UMW	3	$0.23	Link
Resistor - 10kΩ 5%(1/8W) (0805) *	YAGEO	10	$0.10	Link
Resistor - 150Ω (0603) *	Stackpole Electronics Inc	3	$0.10	Link
Resistor - 1kΩ (0603) *	YAGEO	6	$0.10	Link
Resistor - 1MΩ / 1% / (1/8W) (0805) *	YAGEO	1	$0.12	Link
Resistor - 33KΩ 1%(1/8W) (0805) *	YAGEO	1	$0.10	Link
Wire to Board Header *	Molex	5	$0.13	Link
Total			$223.04	

* Implies part will be obtained from machine shop

Table 8: Schedule

Week	**Tasks**	**Person**
February 23rd – March 1st	Finish Design Document Finish PCB Design Find Parts to Order Order Parts **FIRST ROUND PCB ORDER**	Everyone Nick/Ben Dominic Everyone
March 2nd – March 8th	Design Review Start Programming with Dev Board PCB Design Updates Work on Breadboard Demo (Finish UI and Power System Breadboard Design) **SECOND ROUND PCB ORDER**	Everyone Dominic Nick/Ben Everyone
March 9th – March 15th	Work on Breadboard Demo (Finish Pump System Breadboard Design) Make sure all parts are up to spec Continue Dev Board Programming Breadboard Demo PCB Design Updates Give Machine Shop all parts **TEAMWORK EVALUATION DUE** **THIRD ROUND PCB ORDER**	Everyone Everyone Dominic Everyone Nick/Ben Everyone
March 16th – March 22nd	NONE REQUIRED	Everyone
March 23rd – March 29th	Continue Programming (UI and Status and Weight Verification Subsystems Finished) PCB Design Updates	Dominic Nick/Ben

	FINAL ROUND PCB ORDER	
March 30th – April 5th	Work on PCB Assembly Continue Programming Start Part Integration **INDIVIDUAL PROGRESS REPORT DUE**	Nick/Ben Dominic Everyone
April 6th – April 12th	Continue Programming (Stirring Mechanism and Pumps and Plumbing Subsystem Finished and Cocktail Program Finished) Final PCB Assembly Finished **TEAM CONTRACT ASSESSMENT DUE**	Dominic Nick/Ben
April 13th – April 19th	Full Assembly and Part Integration Debug	Everyone Everyone
April 20th – April 26th	Debug Work on Final Report **MOCK DEMO**	Everyone Everyone
April 27th – May 3rd	Debug Work on Final Report **FINAL DEMO** **FINAL PRESENTATION**	Everyone Everyone
May 4th – May 17h	Work on Final Report **FINAL PAPER DUE** **LAB NOTEBOOK DUE**	Everyone

7.1 Appendix B (Figures)

```
//Write calibration values to memory
void saveCalibrationToPrefs() {
  prefs.putFloat("cupScale", cupScale);
  prefs.putFloat("vodScale", vodkaScale);
  prefs.putFloat("mixScale", mixerScale);

  prefs.putLong("cupOff", cupOffset);
  prefs.putLong("vodOff", vodkaOffset);
  prefs.putLong("mixOff", mixerOffset);

  Serial.println("Calibration saved.");
}
```

Figure 13: Code for Saving Calibration Data

```
bool wasInProgress = prefs.getBool("inProg", false);
if (wasInProgress) {
  Serial.println("Recovered from power loss during active drink. Bad Drink");
  setState(STATE_FAIL_HOLD);
  actuatorRetract();
  delay(12000);
  stopActuator();
  persistInProgressFlag(false);
  //setState(STATE_FAIL_HOLD);
} else {
  setState(STATE_IDLE);
}
```

Figure 14: Code for Preferences Library in Progress Flag

```
//Manual interface for debugging and calibration
void handleSerialCommands() {
  if (!Serial.available()) return;

  String cmd = Serial.readStringUntil('\n');
  cmd.trim();
  cmd.toLowerCase();

  if (cmd == "help") {
    Serial.println("Commands:");
    Serial.println("read");
    Serial.println("tare cup");
    Serial.println("tare vodka");
    Serial.println("tare mixer");
    Serial.println("tare all");
    Serial.println("setcal cup <value>");
    Serial.println("setcal vodka <value>");
    Serial.println("setcal mixer <value>");
    Serial.println("cal cup <known grams>");
    Serial.println("cal vodka <known grams>");
    Serial.println("cal mixer <known grams>");
    Serial.println("savecal");
    return;
  }

  if (cmd == "read") {
    printScaleReadings();
    return;
  }

  if (cmd == "tare all") {
    tareAllScales();
    return;
  }
```

Figure 15: Code for Serial Monitor Screen

```
//Drink Checks
bool verifyCanStart(DrinkType drink) {
  if (!powerGood()) {
    Serial.println("Precheck fail: wall power not good");
    return false;
  }
  if (!cupPresent()) {
    Serial.println("Precheck fail: cup not present");
    return false;
  }
  //Needed amount for a drink
  float vodkaNeeded = 0.0f;
  float mixerNeeded = 0.0f;
  if (drink == DRINK_VODKA_SHOT) {
    vodkaNeeded = VODKA_SHOT_TARGET_G;
  } else if (drink == DRINK_MIXED) {
    vodkaNeeded = MIXED_VODKA_TARGET_G;
    mixerNeeded = MIXED_MIXER_TARGET_G;
  }
  //Read current values
  float vodkaAvail = readVodkaGrams();
  float mixerAvail = readMixerGrams();
  //Check if we have enough of ingredients
  if (vodkaAvail < vodkaNeeded * LIQUID_MARGIN_FACTOR) {
    Serial.printf("Precheck fail: not enough vodka. Need %.2f g, have %.2f g\n", vodkaNeeded * LIQUID_MARGIN_FACTOR, vodkaAvail);
    return false;
  }
  if (mixerNeeded > 0.0f && mixerAvail < mixerNeeded * LIQUID_MARGIN_FACTOR) {
    Serial.printf("Precheck fail: not enough mixer. Need %.2f g, have %.2f g\n", mixerNeeded * LIQUID_MARGIN_FACTOR, mixerAvail);
    return false;
  }
  return true;
}
```

Figure 16: Code for verifyCanStart()

```
//Read cup load cell
float readCupGrams() {
  float raw = averageRaw(hxCup, SCALE_SAMPLES);
  if (isnan(raw)) return 0.0f;
  return rawToGrams((long)raw, cupOffset, cupScale);
}
```

Figure 17: Code for readCupGrams()

```
void calibrateCupScale(float knownWeightG) {
  if (!hxCup.is_ready() || knownWeightG <= 0.0f) {
    Serial.println("Cup calibration failed: HX711 not ready or invalid weight.");
    return;
  }

  long raw = (long)averageRaw(hxCup, 10);
  cupScale = (cupOffset - raw) / knownWeightG;
  Serial.printf("Cup calibration set: known=%.2f g raw=%ld scale=%.6f counts/g\n",
                knownWeightG, raw, cupScale);
}
```

Figure 18: Code for Calibrating Cup Scale

```
if (!cupPresent()) {
  enterFailHold("Cup removed during pumping");
  return false;
}
//Issue mismatch too large
if (mismatch > MISMATCH_FAULT_G) {
  enterFailHold("Pump mismatch fault");
  return false;
}
//Issue took too long
if ((millis() - stateStartMs) > PUMP_TIMEOUT_MS) {
  enterFailHold("Pump timeout fault");
  return false;
}
//Issue if pump isn't pumping enough
if ((millis() - lastProgressCheckMs) >= PROGRESS_CHECK_MS) {
  float deltaSinceLast = cupNow - lastProgressCupG;
  if (deltaSinceLast < MIN_PROGRESS_G) {
    enterFailHold("No meaningful pump progress detected");
    return false;
  }
  lastProgressCupG = cupNow;
  lastProgressCheckMs = millis();
}
//Stop pumping on either of these conditions (need to fine tune)
if (cupGain >= (activeTargetG - STOP_TOLERANCE_G) || sourceLoss >= (activeTargetG - STOP_TOLERANCE_G)) {
  digitalWrite(PIN_PUMP_VODKA, LOW);
  digitalWrite(PIN_PUMP_MIXER, LOW);
  //Allow drip to fall
  delay(300);
  //measure final cup and how much was transferred
  float finalCup = readCupGrams();
  float delivered = finalCup - activeCupStartG;
  Serial.printf("Pump complete. delivered=%.2f g target=%.2f g\n", delivered, activeTargetG);
  //Make sure delivered amount is in tolerance range
  if (fabs(delivered - activeTargetG) > (STOP_TOLERANCE_G + 5.0f)) {
    enterFailHold("Delivered amount too far from target");
    return false;
  }
  return true;
}
//If still here something went wrong or still pumping
return false;
}
```

Figure 19: Code for handlePumpStep() (Trivial Lines and Declarations omitted)

```
//Raises actuator
bool handleActuatorRetracting() {
  if (!powerGood()) {
    enterFailHold("Power-good lost during actuator retract");
    return false;
  }
  //Force actuator back up
  actuatorRetract();
  unsigned long elapsed = millis() - stateStartMs;
  //Normal timed completion
  if (elapsed >= ACTUATOR_RETRACT_MS) {
    stopActuator();
    Serial.println("Actuator retract complete (timer-based).");
    return true;
  }
  //Safety timeout
  if (elapsed > ACTUATOR_TIMEOUT_MS) {
    stopActuator();
    enterFailHold("Actuator retract timeout");
    return false;
  }
  //Retract still happening
  return false;
}
```

Figure 20: Code for handleActuatorRetracting()

```
void updateLEDs() {
  //Assume LEDs off
  bool red = false;
  bool green = false;
  bool blue = false;

  switch (state) {
    //Red LED on
    case STATE_PRECHECK_FAULT:
    case STATE_FAIL_HOLD:
      red = true;
      break;

    //Green LED on
    case STATE_SUCCESS_HOLD:
      green = true;
      break;

    //Blue LED on
    case STATE_PUMP_VODKA:
    case STATE_PUMP_MIXER:
    case STATE_ACTUATOR_EXTENDING:
    case STATE_STIRRING:
    case STATE_ACTUATOR_RETRACTING:
      blue = true;
      break;

    default:
      break;
  }
  //Write values to LED pins
  digitalWrite(PIN_LED_RED, red ? HIGH : LOW);
  digitalWrite(PIN_LED_GREEN, green ? HIGH : LOW);
  digitalWrite(PIN_LED_BLUE, blue ? HIGH : LOW);
}
```

Figure 21: Code for updateLEDs()

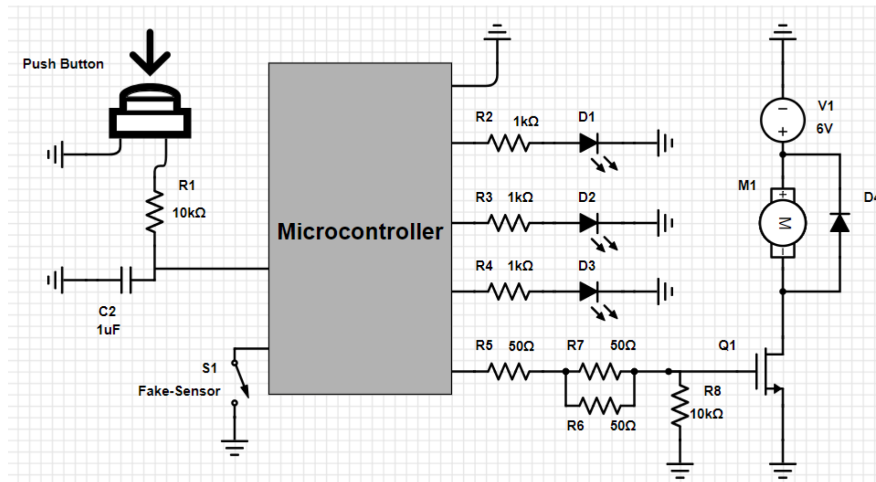

Figure 22: Initial Breadboard Testing Circuit