UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Automated Cocktail/Mocktail Maker

Electrical & Computer Engineering

ECE 445 Team 9

Presentation Date:
05/01/2026

Project Introduction

The team, our project, and today's agenda

Team Introduction

Nick Kubiak
Electrical Engineering

Dominic Andrejek
Computer Engineering

Benjamin Kotlarski
Electrical Engineering

TA: Wesley Pang

Today's Agenda

Project Introduction

Problem

Inconsistent home mixing; small liquid volumes are hard to measure accurately.
Overpouring / unexpected alcohol content; risk of overdrinking.
Requires skill or measuring tools; average user lacks jiggers/experience.
Existing automated machines are expensive; poor consumer adoption.
Many products use proprietary consumables; limits customization.
Manual stirring adds variability; inconsistent mixing.
Unclear performance targets; users need predictable results.

Solution

Automated precise dispensing; peristaltic pumps + load-cell feedback for repeatable pours.
Ingredient-level checks; verify cup presence and container levels before dispensing.
Simple UI; push-button selection with LED status indicators for clear user feedback.
Low-cost design; compact enclosure and ESP32 control to target average consumers.
Open consumables; accepts standard bottles/tubes to preserve user freedom.
Automated stirring sequence; linear actuator + timed motor for consistent mixing.

Objective Statement

Design and build an affordable, fully automated cocktail and mocktail mixer that solves inconsistent home drink mixing by accurately dispensing ingredients, detecting cup presence and ingredient levels, completing a standard two-ingredient mix quickly, and avoiding proprietary consumables, thereby preventing overpouring by making repeatable, safe drinks

High Level Requirements

Requirement 1:

Must dispense each ingredient within ±5 grams of target amount

Requirement 2:

Should detect the presence of a cup and verify sufficient ingredient amounts prior to use.

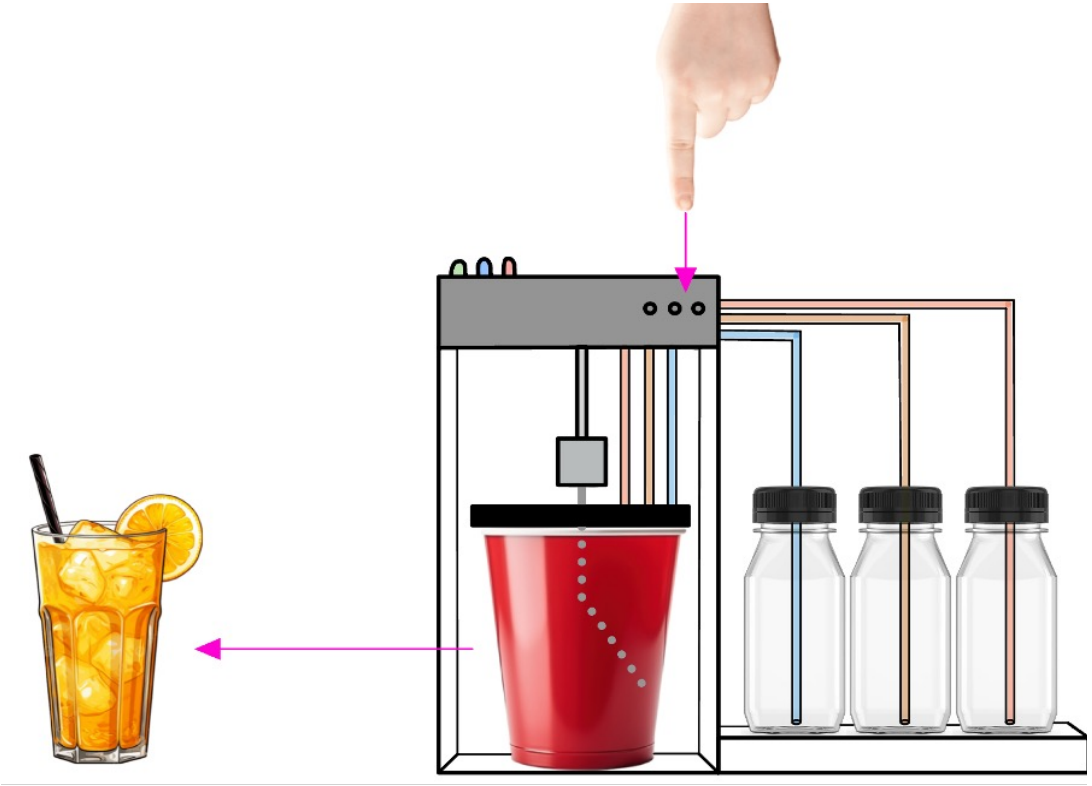
Requirement 3:

Total process should be completed in under 150 seconds

Design Decisions (Hardware Components and Visual)

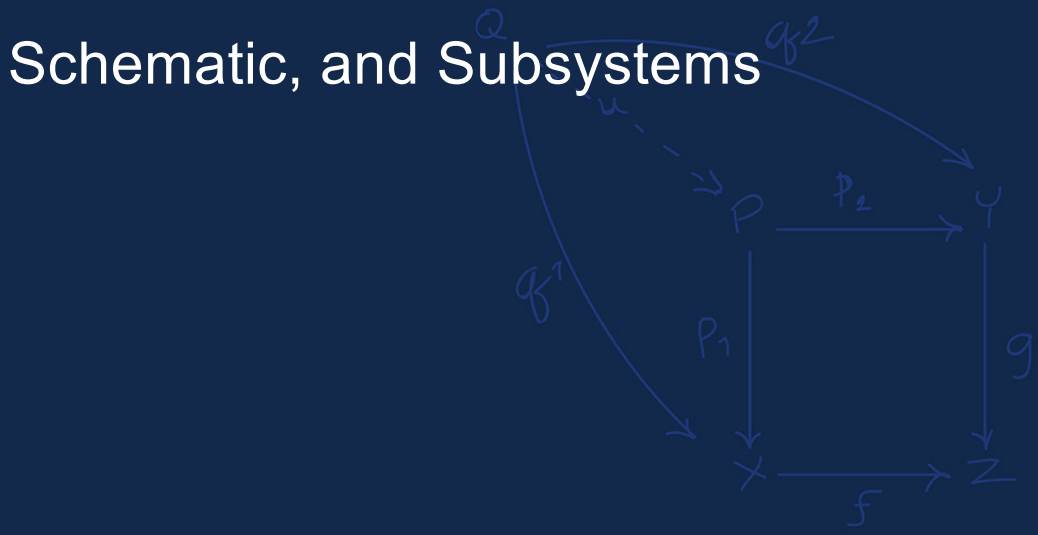
Components:

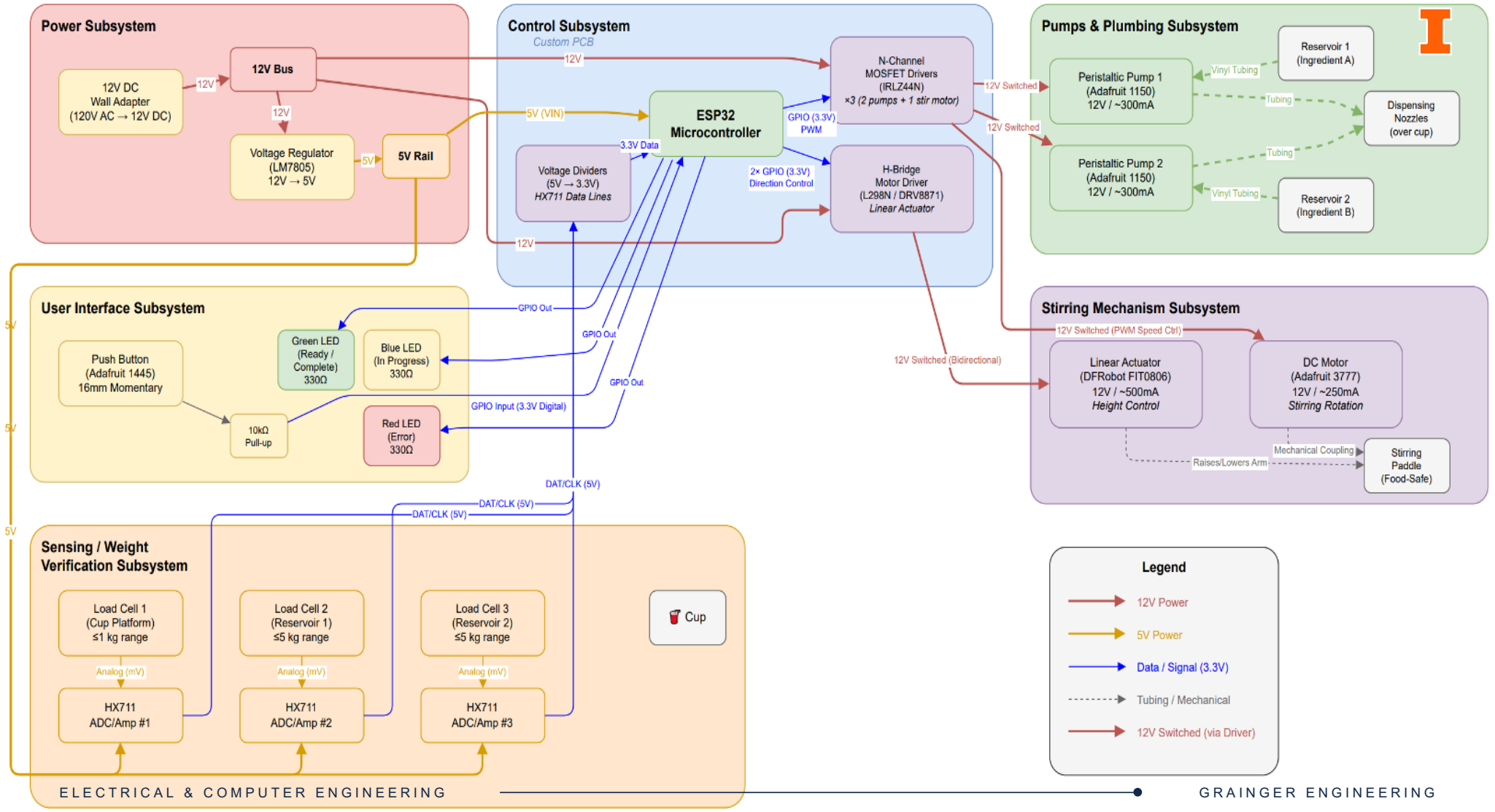
- 3 HX711 Weight Sensors
- 3 Status LEDs
- 2 Self-Priming Pumps
- 1 Start Button
- 1 Power Switch
- 1 Linear Actuator
- 1 Gear Reduction Motor



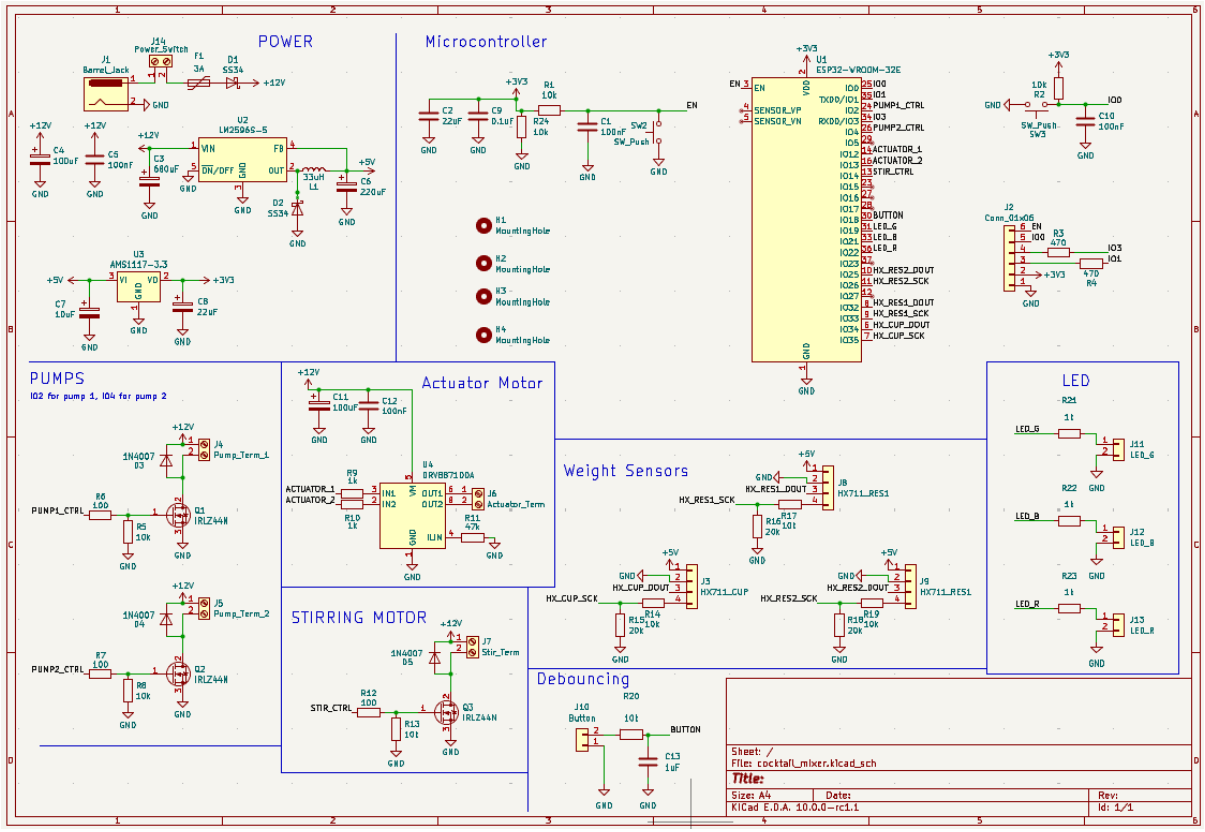
Hardware Design

Block Diagram, Circuit Schematic, and Subsystems



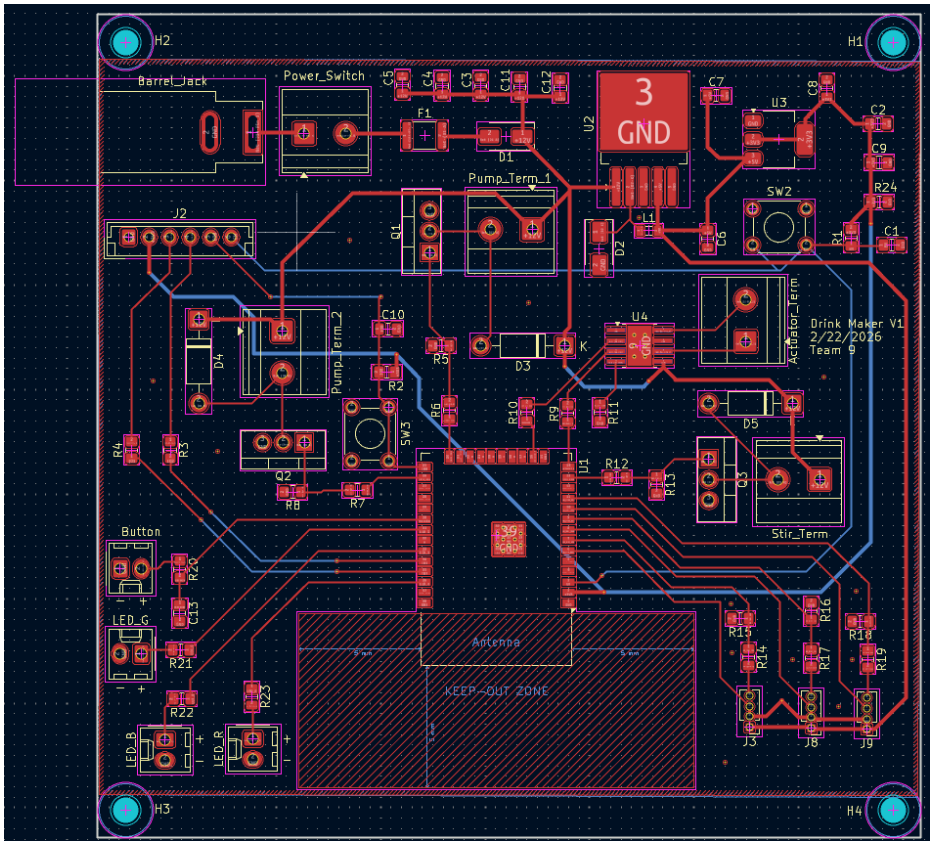

Power Subsystem
12V DC
Wall Adapter
(120V AC → 12V DC)
12V
12V Bus
12V
Voltage Regulator
(LM7805)
12V → 5V
5V
5V Rail
Control Subsystem
Custom PCB
12V
5V (VIN)
ESP32
Microcontroller
3.3V Data
Voltage Dividers
(5V → 3.3V)
HX711 Data Lines
GPIO (3.3V)
PWM
2× GPIO (3.3V)
Direction Control
N-Channel
MOSFET Drivers
(IRLZ44N)
×3 (2 pumps + 1 stir motor)
H-Bridge
Motor Driver
(L298N / DRV8871)
Linear Actuator
12V
Pumps & Plumbing Subsystem
12V Switched
12V Switched
Peristaltic Pump 1
(Adafruit 1150)
12V / ~300mA
Peristaltic Pump 2
(Adafruit 1150)
12V / ~300mA
Vinyl Tubing
Reservoir 1
(Ingredient A)
Tubing
Dispensing
Nozzles
(over cup)
Tubing
Vinyl Tubing
Reservoir 2
(Ingredient B)
User Interface Subsystem
5V
5V
GPIO Out
GPIO Out
GPIO Out
Push Button
(Adafruit 1445)
16mm Momentary
Green LED
(Ready / Complete)
330Ω
Blue LED
(In Progress)
330Ω
GPIO Input (3.3V Digital)
10kΩ
Pull-up
Red LED
(Error)
330Ω
12V Switched (Bidirectional)
Stirring Mechanism Subsystem
12V Switched (PWM Speed Ctrl)
Linear Actuator
(DFRobot FIT0806)
12V / ~500mA
Height Control
DC Motor
(Adafruit 3777)
12V / ~250mA
Stirring Rotation
Mechanical Coupling
Raises/Lowers Arm
Stirring
Paddle
(Food-Safe)
DAT/CLK (5V)
DAT/CLK (5V)
DAT/CLK (5V)
5V
Sensing / Weight
Verification Subsystem
Load Cell 1
(Cup Platform)
≤1 kg range
Load Cell 2
(Reservoir 1)
≤5 kg range
Load Cell 3
(Reservoir 2)
≤5 kg range
Cup
Analog (mV)
Analog (mV)
Analog (mV)
HX711
ADC/Amp #1
HX711
ADC/Amp #2
HX711
ADC/Amp #3
Legend
12V Power
5V Power
Data / Signal (3.3V)
Tubing / Mechanical
12V Switched (via Driver)

Original Circuit Schematic



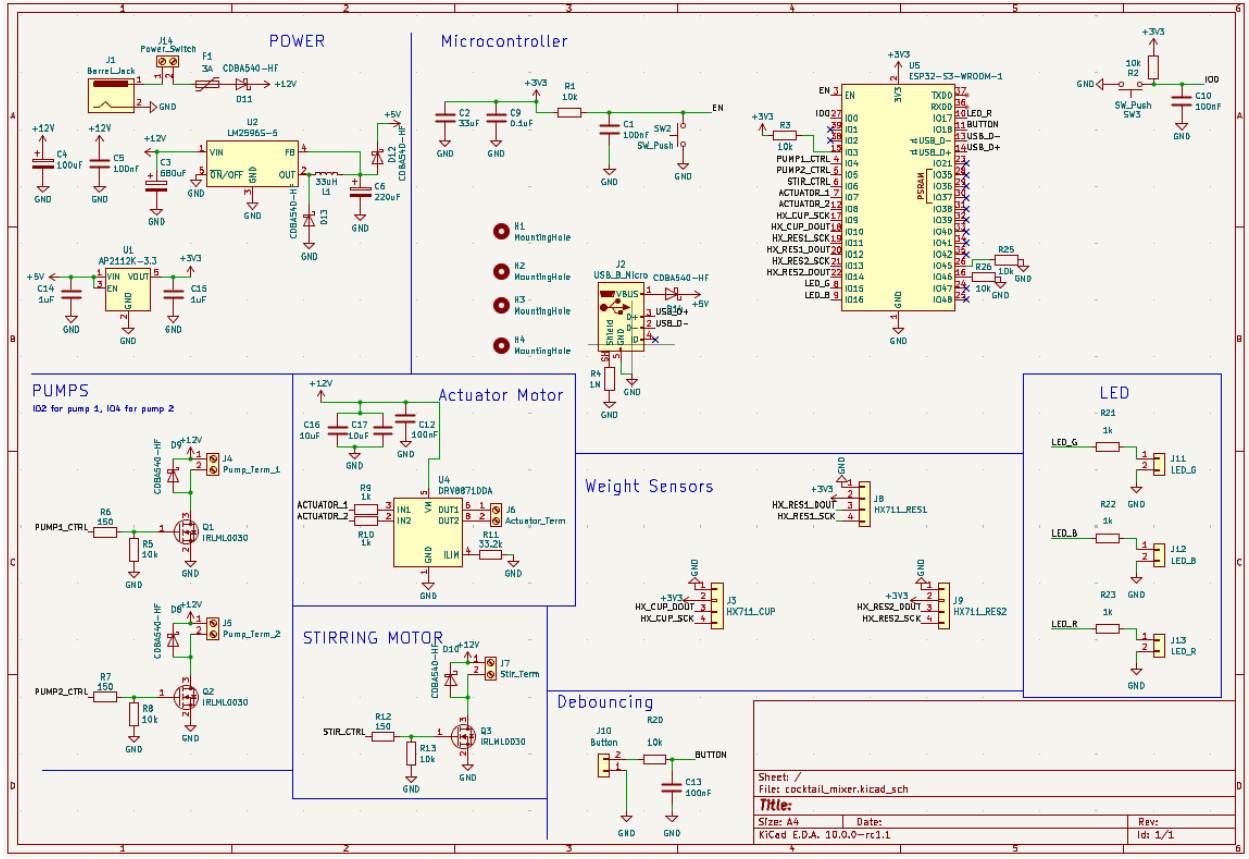
Initial Unoptimized Design

Original PCB



Initial PCB Layout

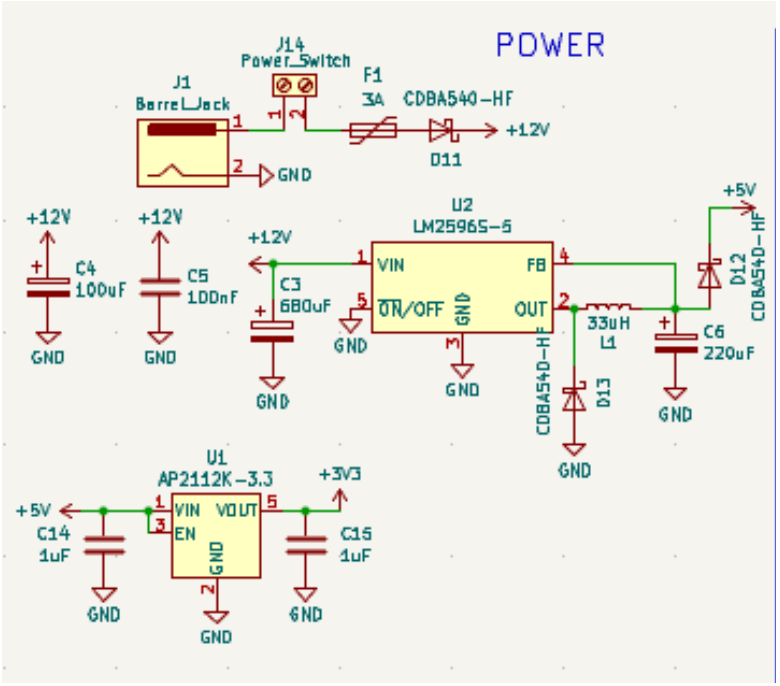
Final Circuit Schematic



All in all, the system is comprised of 8 different subsystems

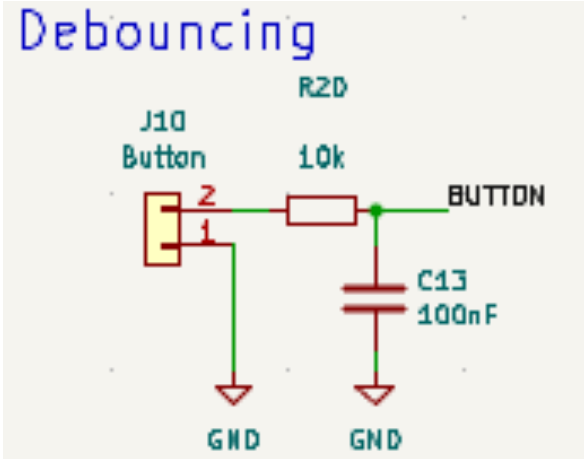
Subsystem - Power

- Converts 120V AC → 12V DC using external adapter
- 12V input protected by switch, fuse, and reverse-polarity diode
- Capacitors filter noise and stabilize voltage rails
- 12V → 5V via efficient buck converter (prebuilt IC)
- 5V → 3.3V via compact linear regulator
- Clean, stable power delivered to microcontroller and sensors



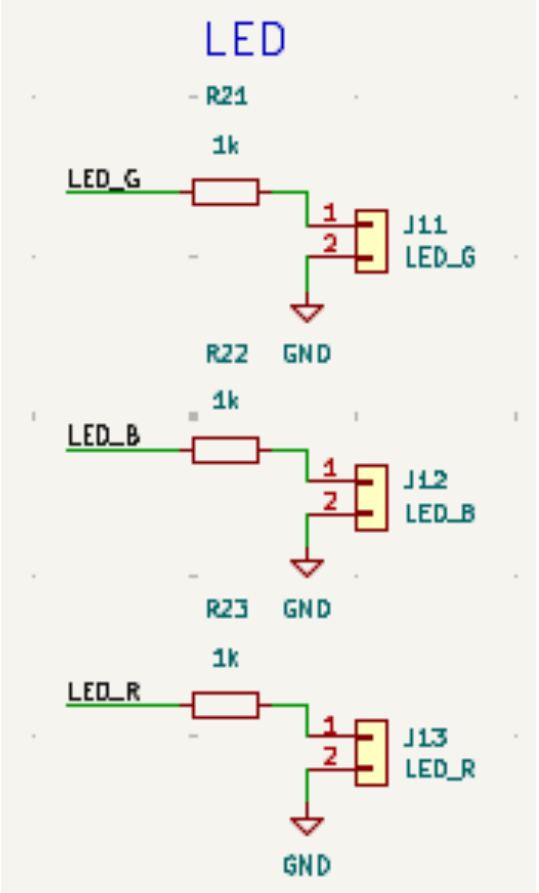
Subsystem – Debouncing Circuit

- Pushbutton provides user input to ESP32
- Mechanical buttons create noise (contact bounce)
- RC filter used to debounce signal
- Capacitor charges/discharges to smooth rapid transitions
- Ensures single, clean input per press
- Component values balance responsiveness and noise filtering



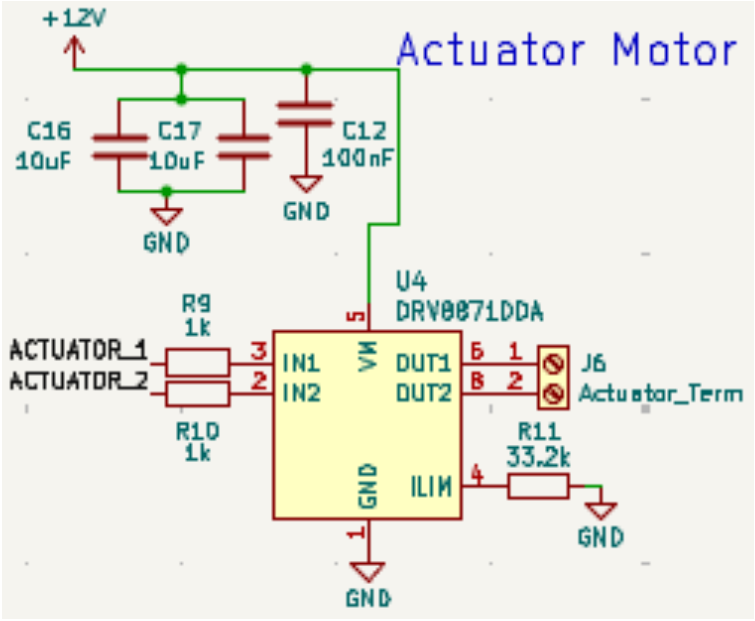
Subsystem – LEDs

- LEDs provide visual feedback of system state
- Controlled directly by ESP32 GPIO output
- Output pin → current-limiting resistor → LED → ground
- Resistor prevents excess current and protects components
- 1000 Ω resistor used for safe operation



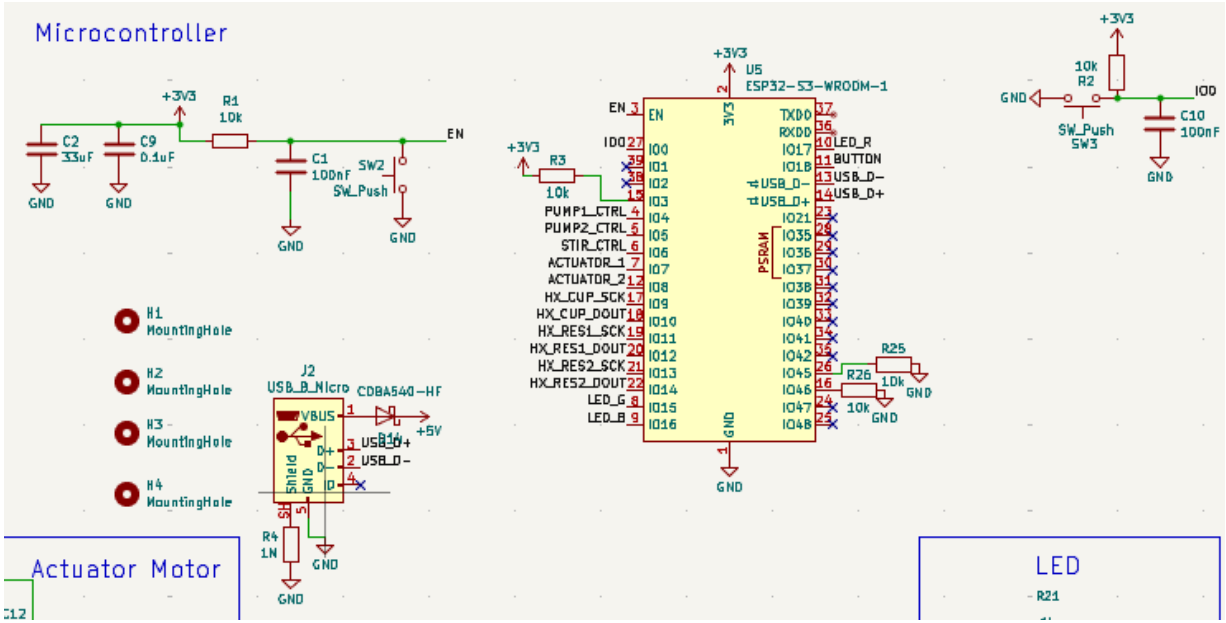
Subsystem – Linear Actuator Motor

- Uses linear actuator motor to move the rotary stirring motor up and down.
- DRV8871DDA H-bridge driver IC allows for bidirectional control, to extend and retract stirring mechanism.
- Powered by 12V rail, 10uF and 100nF capacitors serve to filter electrical noise.
- Logic inputs from microcontroller determine movement direction.
- 33.2kΩ current limiting resistor protects driver and motor.



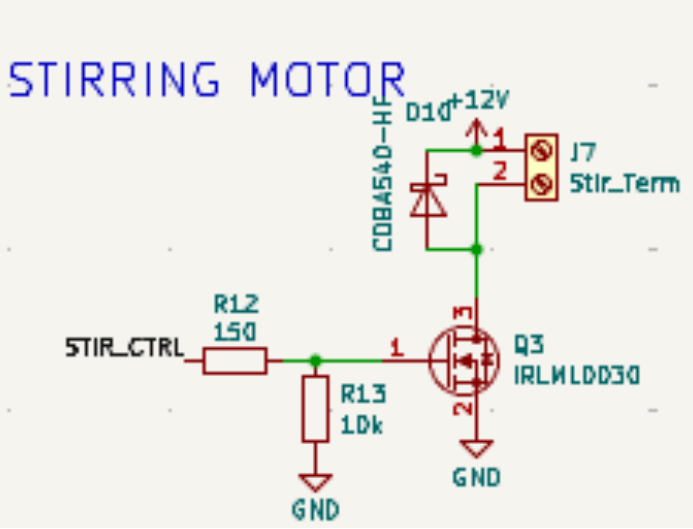
Subsystem – Microcontroller

- Centered on ESP32-S3-WROOM-1 microcontroller module.
- Powered by regulated 3.3V supply rail.
- Interfaces with all inputs (buttons and weight sensors) and outputs (LEDs and motor drivers)
- Micro-USB used for programming and debugging.



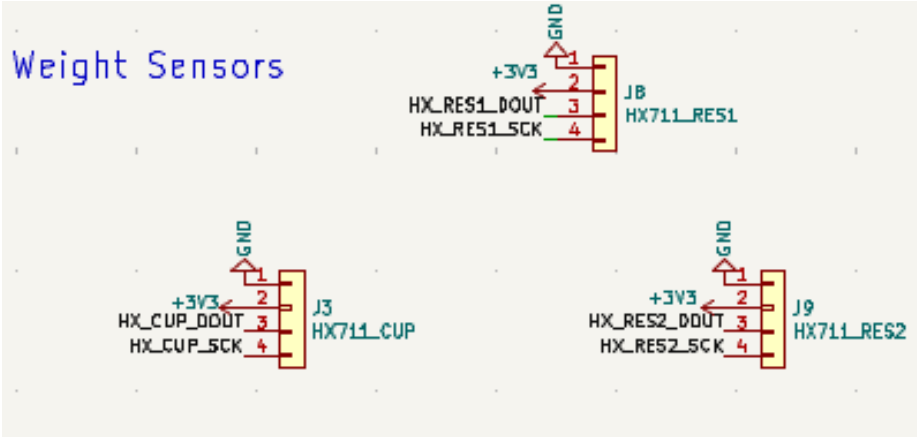
Subsystem – Gear Reduction Motor (Stirring Motor)

- Drives 12V DC gear reduction motor used for mixing ingredients.
- Controlled by output pin from microcontroller.
- Uses an N-channel MOSFET to help handle the motor's 12V power demands.
- Includes flyback diode to protect the circuit from voltage spikes generated by the motor.



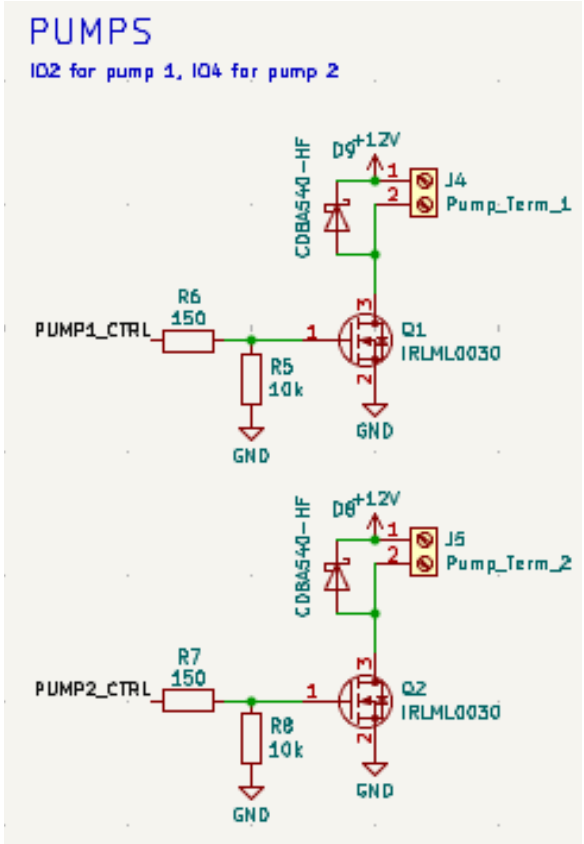
Subsystem – Weight Sensors

- Pins to interface with weight sensors.
- Connects to three HX711 ADC/amplifier modules that process and output analog weight data from load cells.
- Operates on 3.3V rail as opposed to 5V rail, in order to match the microcontroller power.

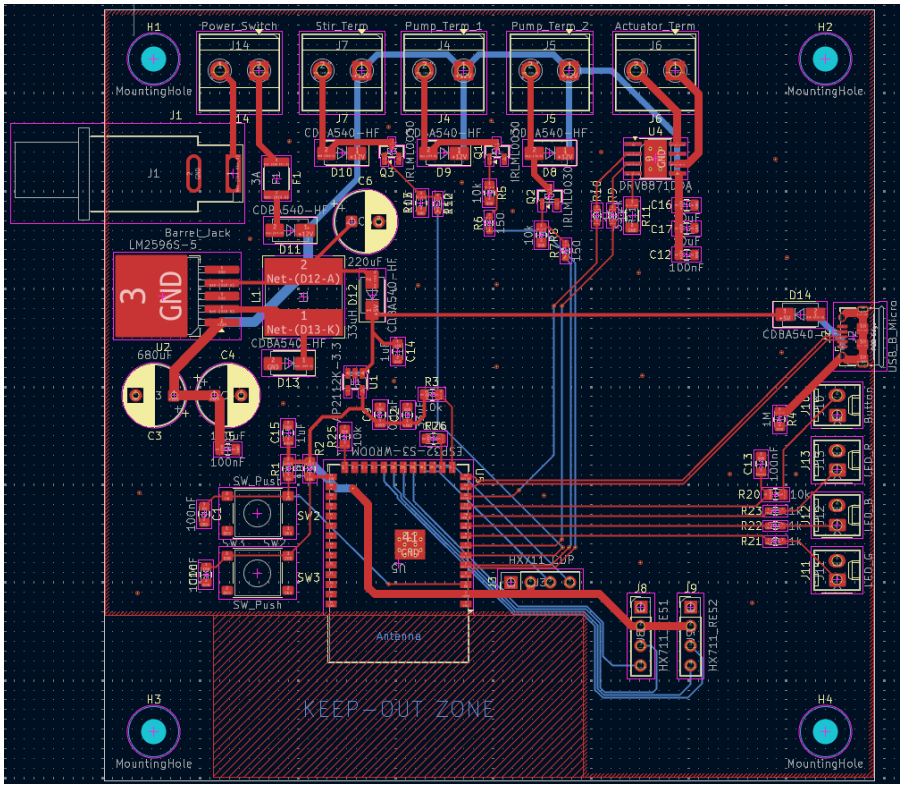


Subsystem – Pumps

- Identical to stirring motor subcircuits.
- Controls two 12V pumps responsible for dispensing liquid ingredients.
- Operated by microcontroller logic signals.
- Driven by N-channel MOSFET.
- Flyback diodes across terminals suppress kickback when pumps turn off.



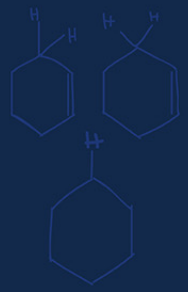
Final PCB

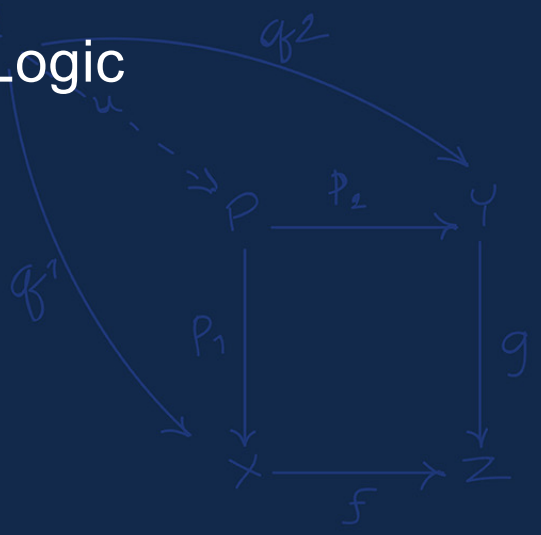


Finalized and Fully Functioning PCB Design/Layout

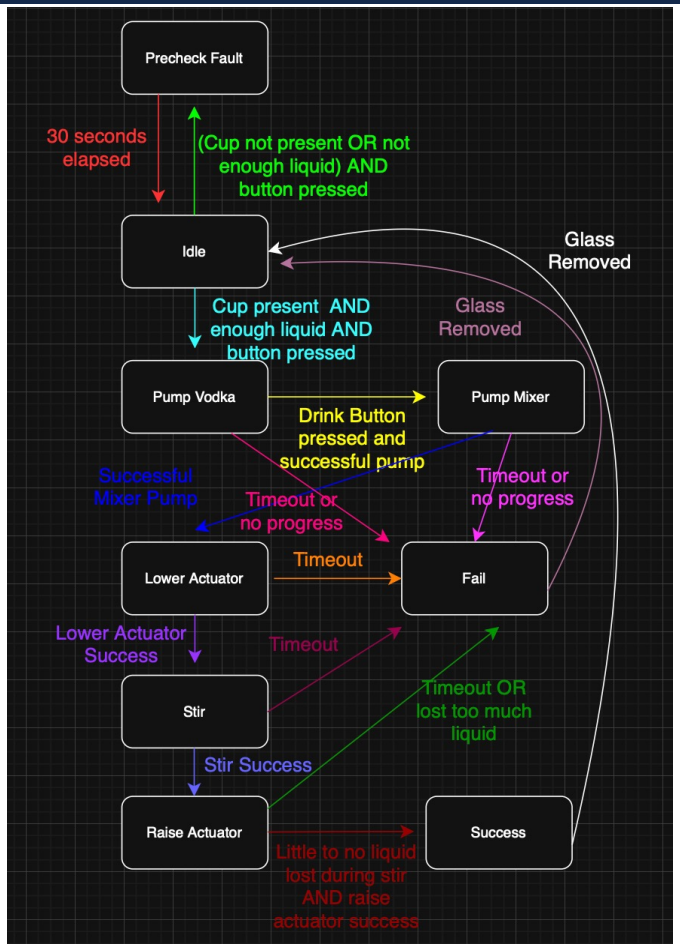
Software Design

FSM, States Breakdown, and Code Logic





FSM


Precheck Fault
30 seconds elapsed
(Cup not present OR not enough liquid) AND button pressed
Idle
Glass Removed
Cup present AND enough liquid AND button pressed
Glass Removed
Pump Vodka
Pump Mixer
Drink Button pressed and successful pump
Successful Mixer Pump
Timeout or no progress
Timeout or no progress
Lower Actuator
Timeout
Fail
Lower Actuator Success
Timeout
Stir
Timeout OR lost too much liquid
Stir Success
Raise Actuator
Success
Little to no liquid lost during stir AND raise actuator success

Idle State & Code Logic

- Automatically in this state on startup
- Checks 3 conditions to see if they are met
- If all met move on to next state in sequence
- If any is NOT met move to a fault state for 30 seconds
- No LED on

```
case STATE_IDLE: {
  //Turn off all
  allOutputsOff();
  //If no power do nothing else
  if (!powerGood()) {
    break;
  }
  //Check button press
  if (mixedBtnStable == LOW) {
    delay(20);
    if (digitalRead(PIN_BTN_MIXED) == LOW) {
      //Start making drink process
      startDrink(DRINK_MIXED);
      //Prevent single long press from being multiple presses
      while (digitalRead(PIN_BTN_MIXED) == LOW) { delay(5); }
    }
  //Same for a shot
  } else if (vodkaBtnStable == LOW) {
    delay(20);
    if (digitalRead(PIN_BTN_VODKA) == LOW) {
      startDrink(DRINK_VODKA_SHOT);
      while (digitalRead(PIN_BTN_VODKA) == LOW) { delay(5); }
    }
  }
  break;
}
```

Precheck Fault State & Code Logic

- Enters state from Idle state
- Enters on any of the 3 conditions from Idle state not being met
- Red LED on for 30 seconds then turns off and goes back to Idle state

```
//If fail pre-drink check
case STATE_PRECHECK_FAULT: {
  allOutputsOff();
  if ((millis() - stateStartMs) >= PRECHECK_FAULT_MS) {
    setState(STATE_IDLE);
  }
  break;
}
```

Pump Vodka State & Code Logic

- Enters state from Idle state
- Starts pumping vodka from housing containers into user glass
- Stops after a shot is poured (45g +/- 5g)
- Multiple checks for proper progress, timeout errors, and correct amount of liquid transfer and goes to fail state if any check fails
- White LED on

```
//Dispense vodka
case STATE_PUMP_VODKA: {
  //Helper function that if true means successful pumping
  //TEMP REMOVED
  if (handlePumpStep()) {
    //Next State logic
    if (selectedDrink == DRINK_VODKA_SHOT) {
      cupWeightBeforeStirG = readCupGrams();
      setState(STATE_ACTUATOR_EXTENDING);
    } else {
      startPumpStep(false, MIXED_MIXER_TARGET_G);
      setState(STATE_PUMP_MIXER);
    }
  }
  break;
}
```

Pump Mixer State & Code Logic

- Enters state from Pump Vodka state
- Starts pumping Mixer from housing containers into user glass
- Stops after correct ratio is poured (currently 2:1 mixer:alcohol +/- 5g)
- Multiple checks for proper progress, timeout errors, and correct amount of liquid transfer and goes to fail state if any check fails
- White LED on

```
//Dispense mixer
case STATE_PUMP_MIXER: {
  if (handlePumpStep()) {
    cupWeightBeforeStirG = readCupGrams();
    setState(STATE_ACTUATOR_EXTENDING);
  }
  break;
}
```

Lower Actuator State & Code Logic

- Enters state from Pump Mixer state
- Lowers linear actuator attached to a gear reduction motor to allow cocktail to be stirred
- Checks against a timer to see if taking too long and goes to fail state if it is
- White LED on

```
//Lower actuator
case STATE_ACTUATOR_EXTENDING: {
  //Helper function for actuator
  if (handleActuatorExtending()) {
    setState(STATE_STIRRING);
  }
  break;
}
```

Stir State & Code Logic

- Enters state from Lower Actuator state
- Uses a gear reduction motor and a paddle to stir cocktail at a fixed rpm
- Stirs for a set basis from a timer set in the code
- White LED on

```
//Stir GRM
case STATE_STIRRING: {
  //Helper function for GRM
  if (handleStirring()) {
    setState(STATE_ACTUATOR_RETRACTING);
  }
  break;
}
```

Raise Actuator State & Code Logic

- Enters state from Stir state
- Raises the linear actuator to move the gear reduction motor out of the way
- Checks against a timer to see if taking too long and goes to fail state if it is
- White LED on

```
//Retract actuator
case STATE_ACTUATOR_RETRACTING: {
  //Raise cup and calculate liquid loss
  if (handleActuatorRetracting()) {
    float cupNow = readCupGrams();
    float lost = cupWeightBeforeStirG - cupNow;

    //Print for debugging
    Serial.printf("Post-stir check: before=%.2f g, after=%.2f g, lost=%.2f g\n",
                  cupWeightBeforeStirG, cupNow, lost);

    //Check if lost too much
    if (cupWeightBeforeStirG > 0.0f &&
        lost > (SPILL_FAULT_PERCENT * cupWeightBeforeStirG)) {
      enterFailHold("Spill fault > 5% after stirring");
    } else {
      enterSuccessHold();
    }
  }
  break;
}
```

Success State & Code Logic

- Enters state from Raise Actuator state
- Turns on the green LED to signify a successful drink is made
- Green LED turns off when the user glass is removed bringing the machine back to the Idle state

```
//Successful drink
case STATE_SUCCESS_HOLD: {
  allOutputsOff();
  if (cupRemoved()) {
    setState(STATE_IDLE);
  }
  break;
}
```

Fail State & Code Logic

- Enters state from any in progress state (white LED on)
- Enters this state if any check is failed from any of the in progress states
- Turns on the red LED to signify a bad drink is made
- Red LED turns off when the user glass is removed bringing the machine back to the Idle state

```
//If we fail during making a drink
case STATE_FAIL_HOLD: {
  allOutputsOff();
  if (cupRemoved()) {
    setState(STATE_IDLE);
  }
  break;
}
```

Preferences Library Use

- Imported Library
- Allows us to store data in flash/non-volatile memory
- Stores things like tare values and the in progress flag

```
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);
}
```

```
//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.");
}
```

HX711 & Weight Sensor Code

- First function converts the raw value read from weight sensor to grams
- Offset – raw because soldered the HX711 boards in reverse
- Second function user to tare the cup scale

```
//Convert raw scale to grams.
//HX711 are wired so the raw reading drops as weight increases
//so the subtraction order is intentionally inverted.
float rawToGrams(long raw, long offset, float scale) {
  if (scale == 0.0f) return 0.0f;
  return (offset - raw) / scale;
}
```

```
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);
}
```

Terminal Interface

- Interface used for debugging and verification
- Read would read the values of all scales
- Tare commands would allow you to tare scales
- Savecal was used to save and tares or calibration data

```
//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;
  }
```

Verification

Design, Subsystem, and Full System Tests

Verification – User Interface Design

Purpose Verify UI interactions (button debouncing, LED status, “cup detection,” microcontroller control, and motor driver) on a simplified breadboard before PCB integration.

Test setup

•*Components*: RC debouncer + pushbutton; MicroBlaze (ESP32 stand-in); cup-detect switch; 3 LEDs (green/red/blue); NMOS-driven motor (mixing proxy).

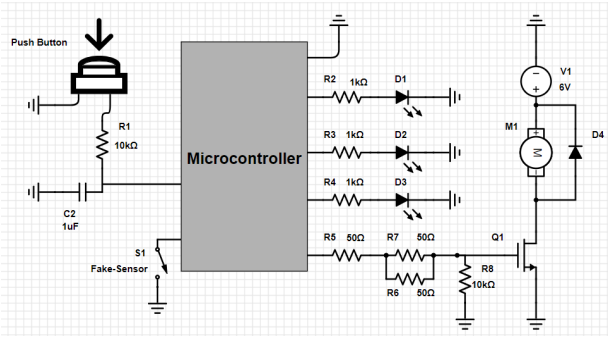
Debounce verification

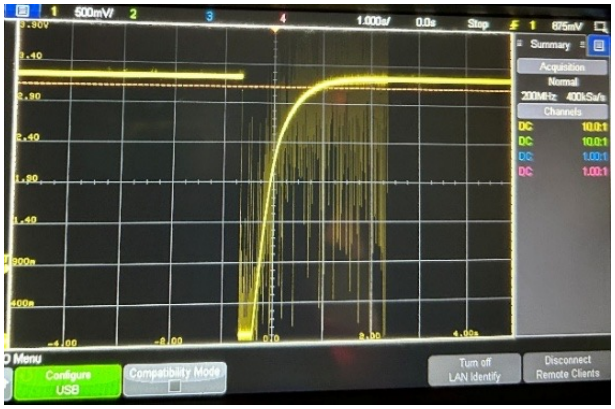
•**Method:** Oscilloscope single-sweep of button signal.

•**Result:** Clean transient with no bouncing — RC debouncer produces a single clean pulse to the microcontroller.

Conclusion and next steps

•Breadboard confirmed UI logic, LED feedback, cup-detection blocking, and motor control sequencing.





Verification – Power Subsystem

Test setup

- *Board state:* Mostly assembled PCB; barrel jack and power switch not yet installed.
- *Input method:* 12 V supplied from a regulated bench supply, bypassing barrel jack and switch; common ground used.
- *Purpose:* Verify regulators produce and distribute 12 V, 5 V, and 3.3 V across the board.

Key findings:

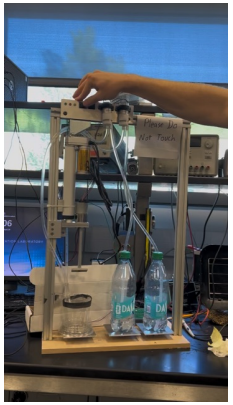
- 12 V and 3.3 V are within tight tolerance and indicate the input distribution and 3.3 V regulator are functioning correctly.
- 5 V reads 4.766 V (≈ 4.68% low) — outside ideal but generally acceptable; likely causes include trace/component voltage drop or buck converter behavior under the current test conditions.

Rail	Expected	Measured	Percent Difference
12 V	12.0 V	12.007 V	0.06%
5 V	5.0 V	4.766 V	4.48%
3.3 V	3.3 V	3.294 V	0.18%

Verification – Full System

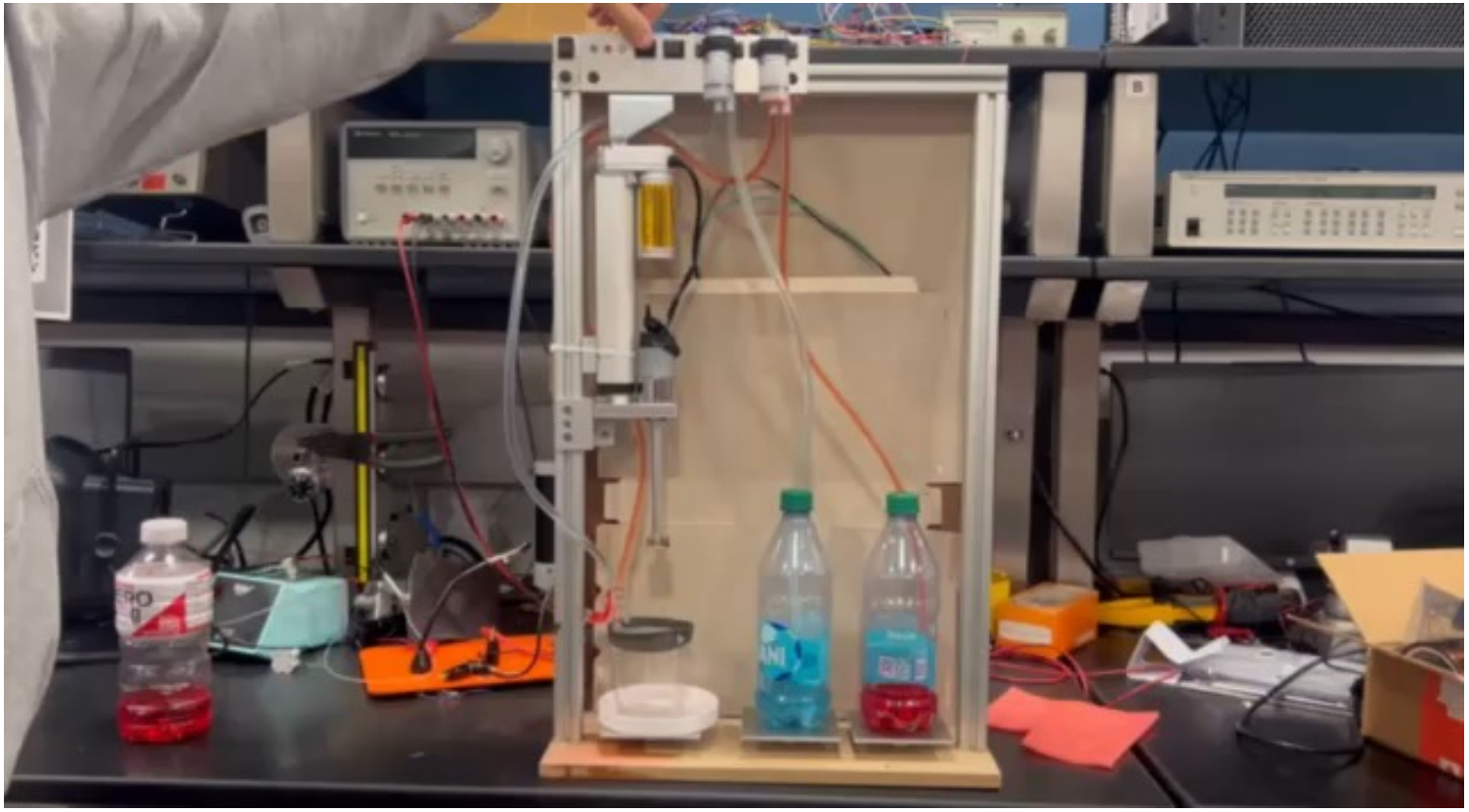
Generally, the full system tests proved to be successful

✓All subsystems functioned individually
✓All subsystems interfaced correctly
✓The whole process successfully completed
✓Time and safety goals were met
✓Portion constraints were met



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

Final Functional Circuit



Conclusion

Requirement Status, Things Learned, and Future Steps

Conclusion

Must dispense each ingredient within ±5 grams of target amount	Max differentiation was 3 grams
Should detect the presence of a cup and verify sufficient ingredient amounts prior to use	Sensed ingredient amounts and container presence
Total process should be completed in under 150 seconds	Total process took about 54 seconds

Lessons Learned

Decisions made early on in the design process will carry on throughput the entirety of the project

It is very important to organize your PCB in a smart way (high voltages near each other, etc)

It is very important to independently test subsystems and states before integrating them

Conclusion

Future Steps

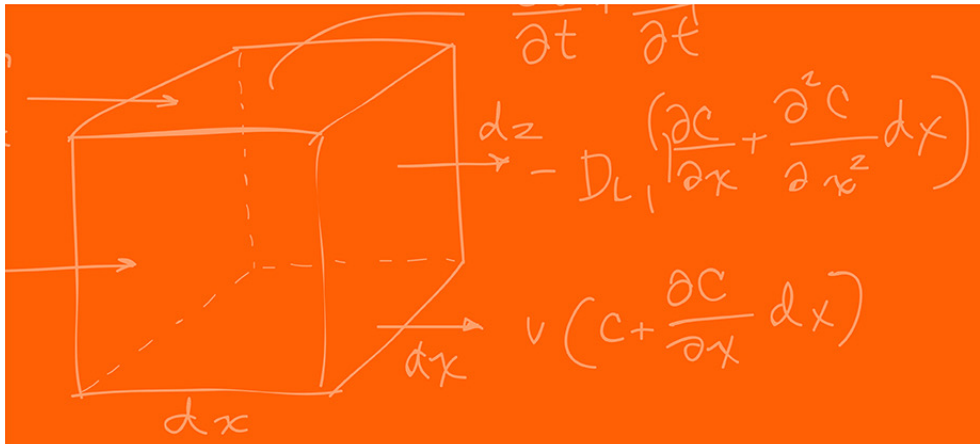
Optimization	Reanalyze the design, and find places to save space, energy, and money
Increased Customization	Expand the UI from a button and light system to an interactive and adjustable screen
Expansion	Add onto the physical design so it can create more complex mixtures moving forward
External Analysis	Run a market analysis to better position our design against what already exists

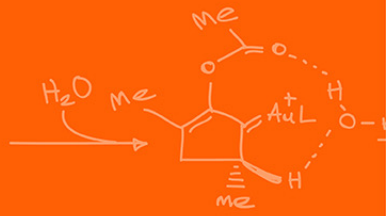
Thank You

Questions?

Team 9

ECE 445





The Grainger College of Engineering

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN