Plant Irrigation and Monitoring System

By
Carlos Toledo
John Burns
Kevin Le

Final Report for ECE 445 Senior Design Fall 2023
TA: Sainath Barbhai

6 December 2023
Project No. 8

Abstract

Our plant irrigation and monitoring system automatically waters a series of plants using a solenoid valve based on user input parameters of scheduled hours and maintains the plant health above its minimum moisture level. Each of our subsystems are tested and they work independently with hardcoded data or theoretical values, but our project ultimately fell short as the valves do not open or close as expected when integrating everything together.

Contents

1. Introduction

Gardening takes a lot of time, skill, and effort. Each plant has its own specific living conditions that can be difficult to keep track of when taking care of many plants. The plant irrigation and monitoring system is designed to maintain an average person's plants' health and alleviate the pressure off those lacking the skills or time to maintain their plants themselves. The system consists of a hose connected to a series of solenoid valves with water running through PVC pipes based on the number of desired plants, where each plant station has two moisture sensors to detect its current health and one to detect whether it is adequately watered. For each plant, the user just needs to do a one time configuration of specific parameters of the plants minimum moisture level and desired scheduling times through the user interface; the system will solely take care of maintaining the plants from here on allowing the user to focus on their busy lives. The entire system consists of the master control subsystem which costs $34.06, as well a user defined number of moisture sensing subsystems needed for each plant, which is $46.73 per plant. Construction materials such as PVC piping and waterproofing equipment will vary based on the area and configuration of the user's garden; per unit prices are provided in the cost section. In the upcoming chapters, this report will be covering the entire design of the system from design to costs and labor. This report will go over each subsystem in detail: UI-Subsystem, Control Subsystem, Moisture Sensing Subsystem, and whether these subsystems work and how we tested them for validity. The UI-Subsystem is where the user can control the specific parameters for each plant through a Web-App and is how data coming from the other subsystems are displayed; we will be going over the WiFi communication as well as the data transmission between the frontend and the microcontroller. The Control Subsystem is the brain of our entire system and is where the communication logic is written to pass on inputs to the UI-Subsystem and the Moisture Sensing Subsystem; we will be going over the logic behind the Read/Write communication protocol. The Moisture Sensing Subsystem is a modular system that can be replicated for each number of plants and is where moisture sensors are located and is where the solenoid valve is to water the plants; we will be going over the logic on how each value is controlled. Upon completion of the project, we learned a lot. We were able to get each of our subsystems individually working but fell a bit short in combining each subsystem and trying to get the system to work as a whole. There are several features that can be integrated in the future such as complete water usage tracking and auto detecting slave devices to name a few. Overall, our project fell a bit short in integrating everything together, but we were successful in many other ways.

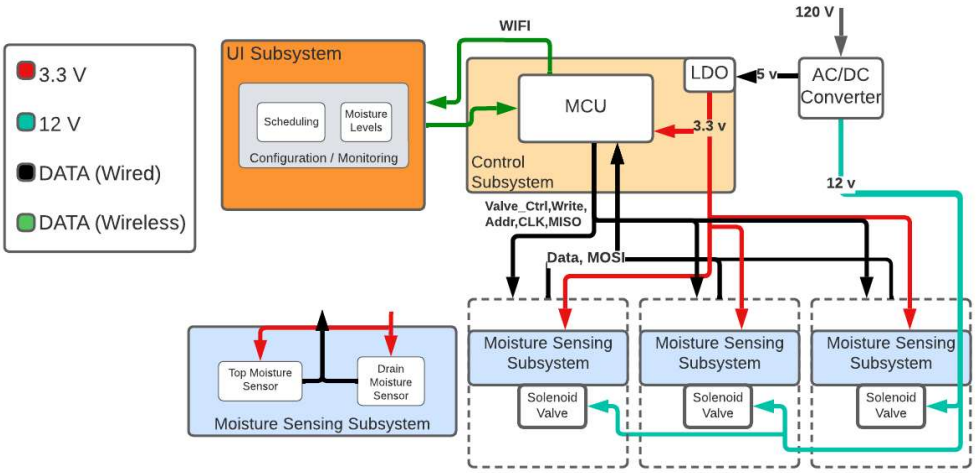

Figure 1: Block Diagram

2 Design

2.1 UI Subsystem

2.1.1 Design Procedure

This subsystem is responsible for displaying the plants health data received from moisture sensors in the Moisture Sensing Subsystem as well as allowing the user to configure specific parameters of the minimum moisture level and scheduled times for each plant. A key part of this subsystem is WiFi communication with the microcontroller in the Control Subsystem to the frontend. The ESP32 microcontroller was specifically chosen for this task as it allows for WiFi and Bluetooth connectivity. It has a built-in WiFi library that lets us connect to WiFi without any hassle. The ESP32 also can act as a station or an access point, which is what we specifically wanted because we planned to store the WiFi credentials on the ESP32 itself instead of on the user local device for security purposes. We were considering other microcontrollers, but quickly settled on the ESP32 for its popularity in WiFi once we decided to use WiFi. For the user interface, we thought that a Web-App would be appropriate because it is easy for the user to see and use without much overhead. Specifically used React.js and CSS for styling as it is a modern javascript framework that allows for easy state management and live updates. Other design approaches include using Angular.js which is another framework, or even a more simple interface using buttons and LED, but a React Web-App is much easier for the user to use and the React library is more straightforward to use compared to Angular. For the communication between the Web-App and the microcontroller, we initially used HTTP requests but realized this was not what we needed as it is a one way communication. We settled on using Websockets which allows for bidirectional communication between the server and the client, which allows us to send data from the user interface to the ESP32 and from the ESP32 back to the user interface.

2.1.2 Design Details

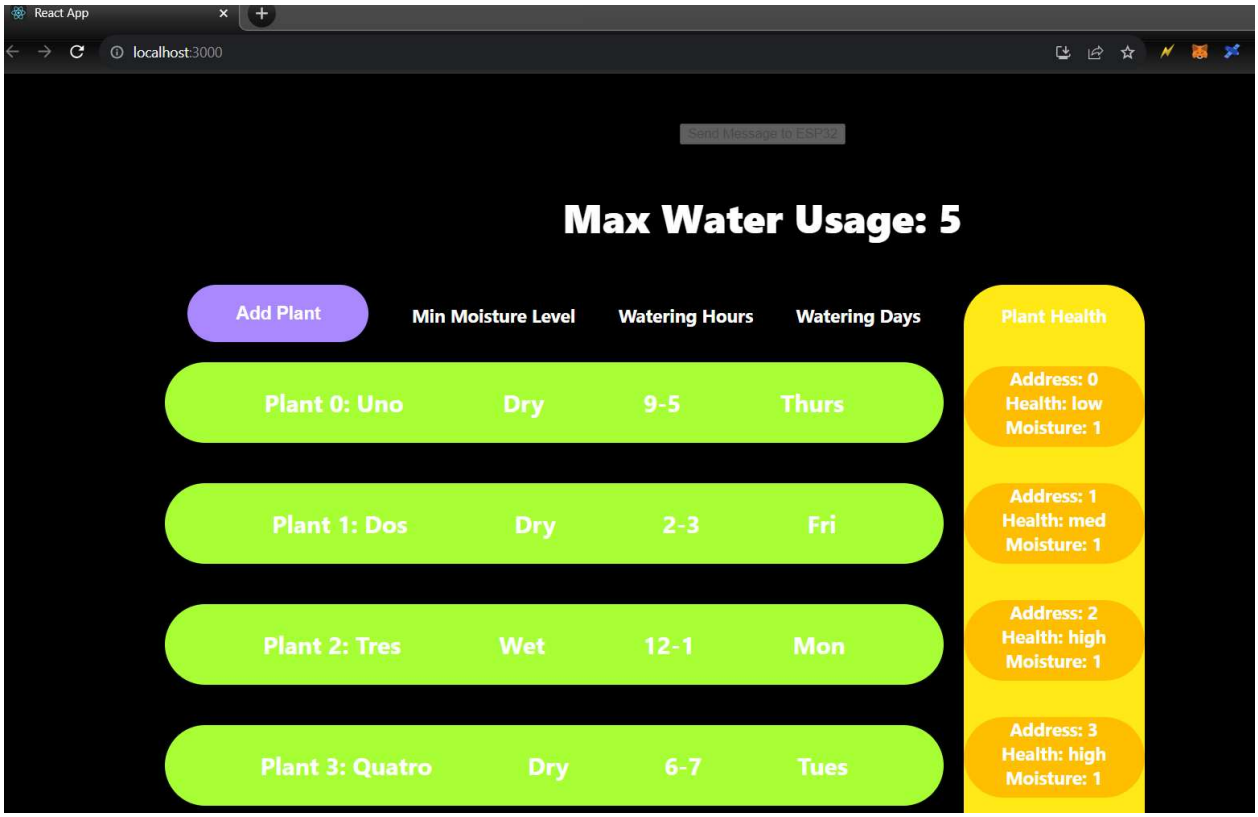

Figure 2: UI Snapshot

Figure 2 is a snapshot of the user interface that the user gets to see and interact with. They can click the "Add Plant" button and then fill in the necessary parameters: plant name, Min Moisture Level, Watering Hours, Watering Days. These parameters are the minimum parameters necessary for the functionality of our system. It allows us to keep track of the minimum moisture level required for the respective plants to survive, and the watering hours and days are present to allow flexibility for which days the user prefers to water their plants. This scheduling takes secondary precedence after local watering state laws. On the right hand side, we decided to display the feedback information on the plants' health which also includes its moisture level and respective slave address. The top displays the max water usage so the user knows how much water they are using and we use this to account for any state watering laws.

2.2 Control Subsystem

2.2.1 Design Procedure

The control subsystem is the center of our decision making process for each plant. It communicates to our UI Subsystem via WiFi to take in parameters and display plant health. The control subsystem can cycle a max of 15 addresses (plants) and is wired in series to our slave subsystems which encodes the state of each plant's two sensors. Using the information from the state of the plant and the restrictions from the user's input on whether watering should occur, it will send out a signal to open or close the valve for each individual plant. The restrictions each user can configure include: allowed days, time, and

moisture level. Taking these into account and use of our SPI communication, configuration between each plant must be saved into local storage. This will help with loss prevention and allow the system to power off and power on without having to re-configure each plant. The ESP 32 is configured to be in both access point and station mode to allow users to connect their system to WiFi by entering a SSID and password. As for powering the whole system, we use a manufactured AC/DC converter from a 120 v outlet, to a 12v, 5v and 3.3v output. We originally had wanted to use our 3.3v output but were required to add a LDO to step down from 5v to 3.3v for 'additional complexity.' The control system pcb is programmed via USB with debounced reset and boot buttons, based on the ESP32 dev board.

2.2.2 Design Details

The control subsystem is fairly simple in terms of physical design, consisting of a 4 bit address output, CLK, data in/out, write and our valve control signal. It accepts JSON input from the UI subsystem that is then saved locally with SPIFFs. This is to prevent data loss with restarting the device. Figure 4 shows a more detailed representation of what components will be on the control system PCB. The voltage supply connection to the ESP32 is wired in accordance with the ESP32 datasheet to operate properly. Each pinout is needed for functionality of the whole system, with SPI communication and status LEDs for the user and DEMO. The USB data in/out connections need TVS diodes. TVS Diodes are used to protect semiconductor components from high-voltage transients. For the power subsystem, since we are using a manufactured AC/DC converter, I am only stepping down from 5v to 3.3v. 12v will be passed through the control system PCB to power the moisture subsystem. Figure 3 shows our decision making code as it stands. It compares the current time and day with watering time/day restrictions and then makes its final decisions based on the moisture content of the soil + drainage.

```
int makeDecision(){//change this later
  if(timeHour < Timethresh || timeHour > Timethresh2){ // outside water time
    return 0;
  }
  int f = 0;
  for(int i = 0 ; i < days ; i++){ // check if current day is in allowed water days
    if(water_days[i] == timeWeekDay){
      f = 1;
      break;
    }
  }
  if (f == 0){ // outside water days
    return 0;
  }
  if (topval < thresh && drainval < drainThresh){ // check top soil value + drain value
    return 1;
  }
  return 0;
}
```

Figure 3: Decision Making Code

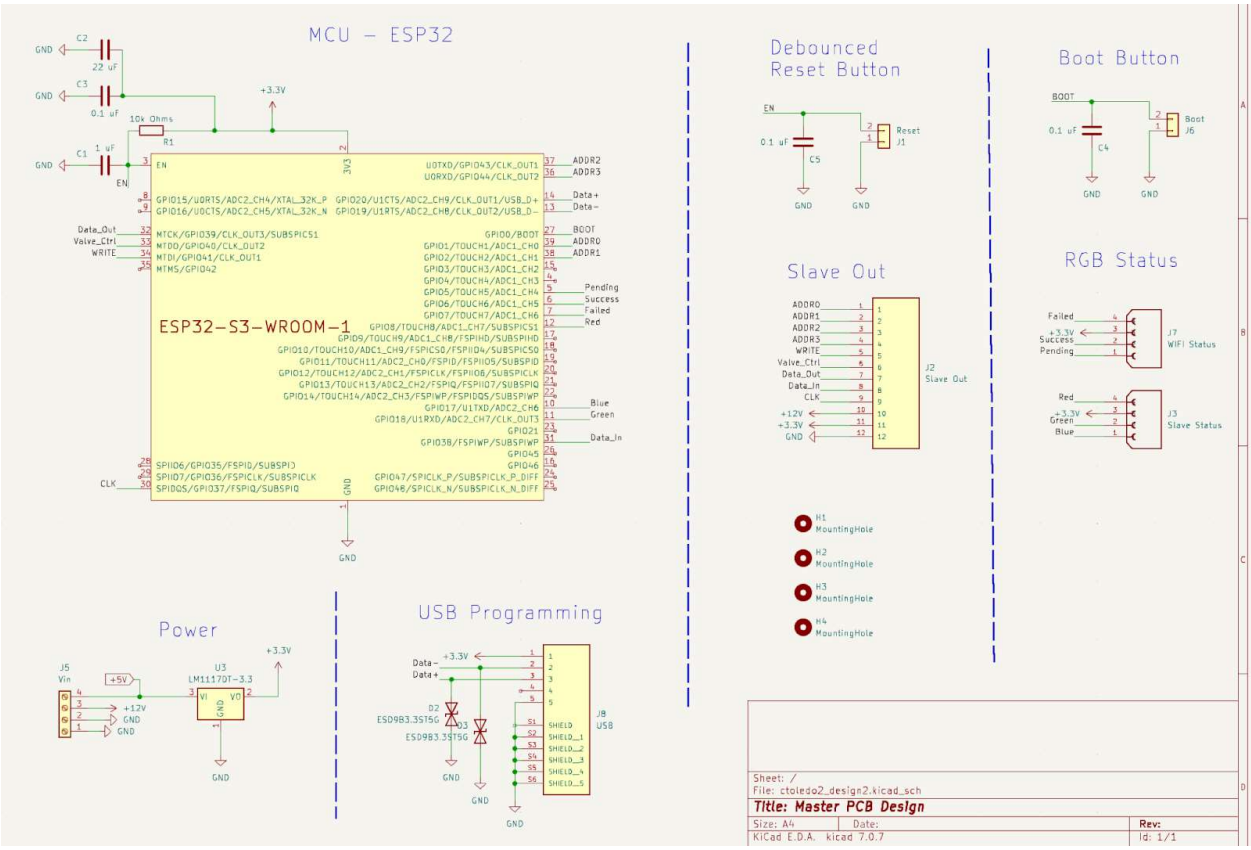

Figure 4: Final Control Subsystem PCB

2.3 Moisture Sensing Subsystem

2.3.1 Design Procedure

This subsystem is responsible for delivering data from the top and drain moisture sensors of a plant to the master control subsystem and for modulating the state of the plant's solenoid valve based on the decision reached by the master control unit. There are many different ways that this communication could be conducted. To achieve efficient scaling in both cost and power, it was decided to avoid using a microcontroller for each moisture sensing subsystems. Thus, custom digital hardware was built on the moisture sensing subsystem to enable the master-slave communication between the control and moisture sensing subsystems using wired communication on data lines coming out of the control subsystem. A wired connection is cheaper and far less complex than wireless communication, and comes with no real downsides as there must be a physical connection of water between plants, presenting a natural route for the wires by binding them along the side of the PVC pipes.

Shown in Figure 5 is a simplified block diagram of the moisture sensing subsystem.High-level functionality is that during a read operation to the slave's hard-coded internal address, the digital logic will produce power to the sensor and pull down the CS Low signal on the ADC when appropriate. The ADC will then communicate the moisture sensor data to the ESP32. During a write operation to the slave's hard-coded internal address, the digital logic will store and output the appropriate valve open

signal which controls a power MOSFET connected in series with the plant's valve, opening or closing it as necessary. Figure 6 shows within the digital logic block on Figure 5 and shows how the CS Low, Solenoid Power, and Sensor Power signals are generated. One can see how adding plants will only involve adding a new address pin when the number of plants is doubled; all the other pins are shared and will not increase in number, allowing for dozens of plants to be hooked up to the same system.

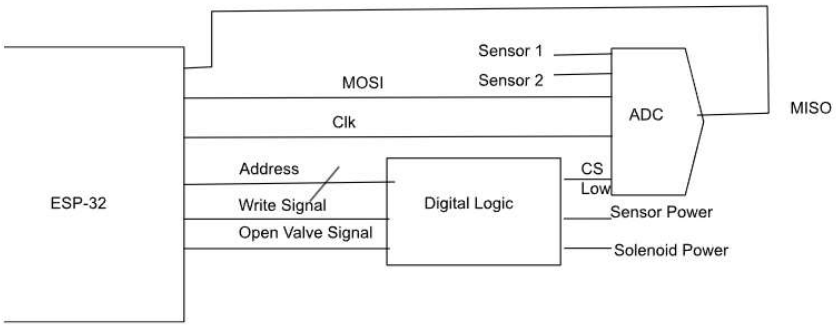

Figure 5: High Level Overview of Slave Hardware + Data Flow

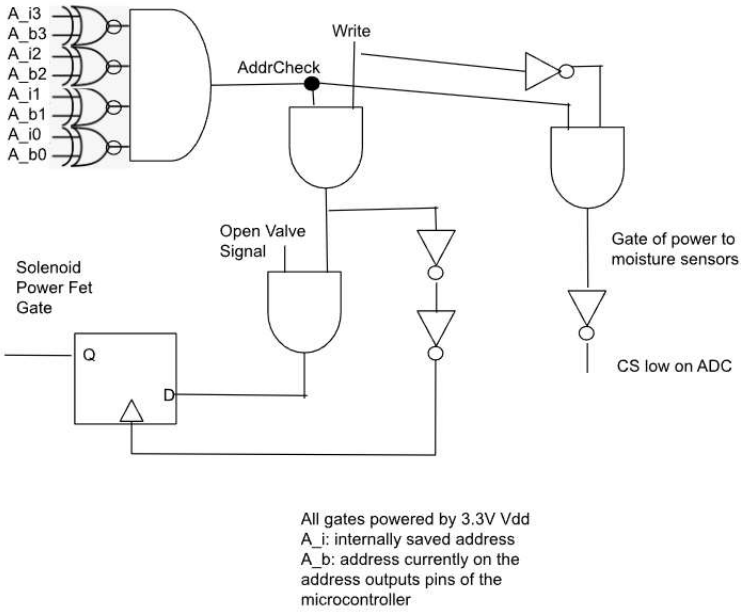

Figure 6: Digital Logic Schematic

2.3.2 Design Details

The moisture sensing slave hardware was built around the idea of being able to scale to high numbers of plants. Having dedicated signals for each slave would have been easy to implement, but would quickly fill up all the pins on a microcontroller. As such, the system operates with shared data lines, and it is the role of 4 address pins to ensure only one slave is communicating at a time. 4 address pins gives 2^4 = 16 possible addresses, allowing for 15 plants to be connected as we reserve one address (1111) to be used

on no slaves for communication protocol purposes (explained below). It is trivial to increase the number of bits here to allow for more plants if desired. Each slave will have a custom arrangement of 4 bits tied to source or ground and these bits are XNOR'd with the respective address bits, giving a 1 if the bits match and a 0 otherwise. These 4 bitwise checks are then AND'd together to give an output which is 1 if the address on the address lines matches the slave's internal address and 0 otherwise. This signal (referred to as AddrCheck from here on) is used at every other stage of the slave hardware- if this signal is low, nothing will happen on the slave.

Writing to the slave is implemented by ANDing the Write signal from the master control unit with AddrCheck and using this as the clock to a flip flop, so it will take in the new Open signal value when this write check signal rises. The output of the flip flop is the gate of a power N-Channel MOSFET positioned between the negative terminal of the solenoid valve and ground. When this signal is 1, the FET will be on and allow for charge to flow, letting a voltage differential develop across the terminals of the valve and causing the valve to open. When the signal is 0, the transistor will be turned off and current will not be able to flow across the valve, so a voltage differential will not develop and the valve will remain closed. A flyback diode is used to avoid component damage when the gate transitions. Collin's Lab Notes on the Adafruit YouTube channel was essential for setting up this solenoid system correctly **[1]**.

Reading is done from two moisture sensors, one at the top soil of the plant and one at the bottom. These sensors give analog voltage outputs and are thus connected to an ADC for communication to the master control unit. The communication protocol is performed by ANDing AddrCheck with Not Write to give a read check signal which is 1 when the slave in question is being read from and 0 otherwise. This signal is used to provide power to the moisture sensors in the same manner as power-gating the solenoid valve, and this signal is also inverted to become Chip Select Low on the ADC. With Chip Select Low pulled low, the ADC is now ready to communicate with the master control unit (explained in further detail below). An example of how to connect this sensor to a microcontroller to control water for a single plant was seen in "Automatic Irrigation System using an Arduino Uno" project by Rajesh on Circuit Digest **[2]**. Although we have a much more complex read operation to perform, with multiple plants, two sensors per plant, and digital rather than analog data communication, it was a very helpful start point to help us see how the project's goal is possible.

Shown in Figure 7 is the final moisture sensing subsystem schematic. The entire subsystem operates on 3.3 V except for the solenoid valve, which will either have a 12 V or 0 V differential across its terminals depending on the state of the signal at the gate of the power MOSFET.

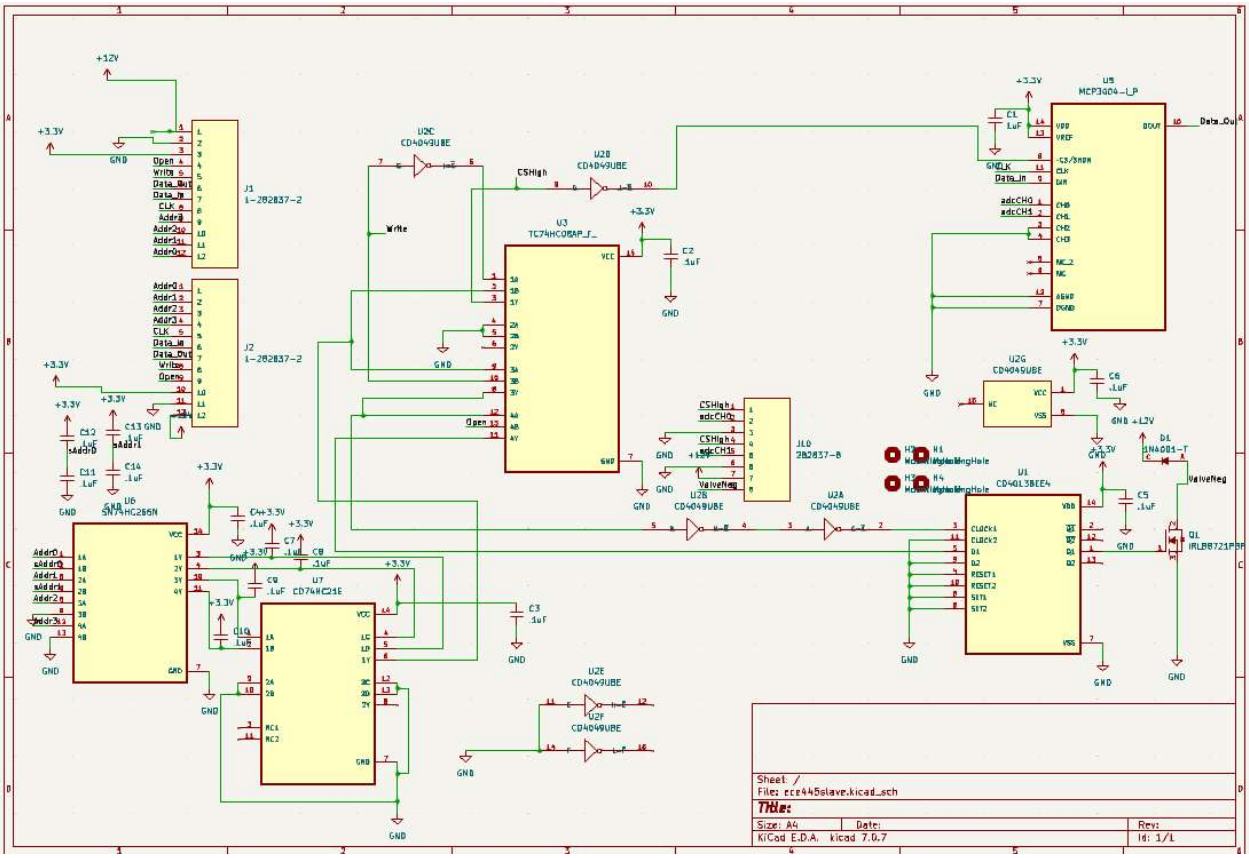

Figure 7: Final Moisture Sensing Subsystem Schematic

After some deliberation, it was decided to not try to force the ESP-32's built in SPI protocol to fit into our usage and instead just manually set the MOSI pin and read the MISO pin depending on the clock cycle. The clock cycling is done by toggling the clock every 10 us and updating a cycle counter if the code just switched to a 0 on the clock as shown in Figure 8.

```
// the loop function runs over and over again forever
void loop() {
  unsigned long currentMicros = micros();
  if (currentMicros - previousMicros >= interval) {
    previousMicros = currentMicros;
    clk = ~clk;
    digitalWrite(clkpin, clk);


    if(clk == 0){
      cycle++;
    }
```

Figure 8: Clock toggling/cycle incrementing on the ESP32.

From there, we used the MCP3004 datasheet page on Figure 9 to work through what each pin needs to be in each clock cycle, writing data on the low edge of the clock and reading during the high edge.

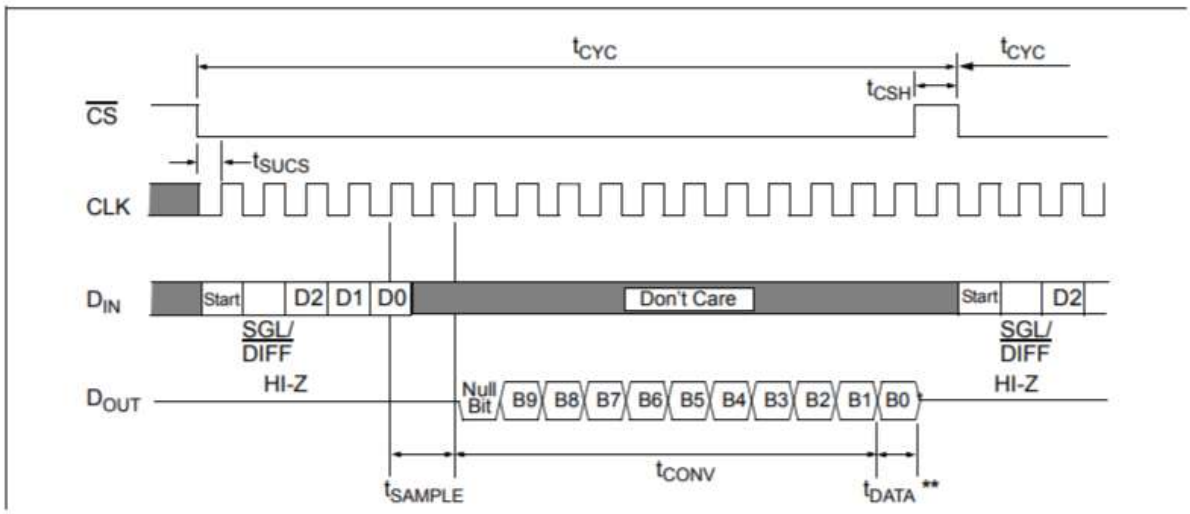

Figure 9: Waveforms for SPI communication with MCP3004 ADC.

Shown in Appendix B is a rough outline of the communication protocol, each entry representing what happens during each clock cycle.

3. Design Verification

Individual subsystems were verified satisfactorily. As shown in the RV tables in Appendix A, the failures came during total system integration. This failure will be discussed more in the conclusion, but all the individual subsystems met the tasks and specifications we set for them.

3.1 UI Subsystem

To verify that the UI-Subsystem works, the user clicks on “Add Plant” and fills in the parameters of the Plant Name, Minimum Moisture Level, Watering Hours, and Watering Days. They then click the “Send msg to ESP32” button and this data is now sent to the ESP32. We can see this exact sent data in the Arduino terminal running the microcontroller code to verify communication works from the frontend to the microcontroller works properly. In the microcontroller code, we have a hardcoded JSON mocking the plant data read from the moisture sensor. We can verify that the communication between the ESP32 and the Web-App works as the right hand column of the Web-App displays the exact same values from the mock data of each plant with its respective address, health, and moisture value. This verifies the functionality of sending data from the microcontroller to the frontend. All this functionality works and is verified.

3.2 Control Subsystem

The verification of the control subsystem consists of testing the WiFi connectivity with User input for WiFi SSID and password, testing the decision making algorithm, and LDO testing.

WiFi Test:

- The User connects to PLANT_SYS WIFI network with password 'pass1234'
- The User enters the WiFi credentials of the desired WIFI network by going to the browser and entering this url: http://192.168.4.1/
    - This will always be accessible to change the WiFi network
- A WiFi Status LED will continuously blink blue until the ESP32 connects to the Internet, then stays green once successful.

Decision Making:

- Setting restrictions on days and times other than the current day and hour will always decide no water should be done
- Setting allowed watering day and time to present time will allow decision making continue
- Putting the top sensor in a dry location will keep the valve open until the drain sensor is wet, submerging the top sensor in water will do nothing but relay the data to the control subsystem

Linear Voltage Regulator (LDO):

- After connecting our power supply to our Master PCB, we confirmed the LDO has an output of 3.3v from 5v max input. The output was just about 3v.

3.3 Moisture Sensing Subsystem

Accurate transmission of data is essential for this project to succeed. After building the moisture sensing slave hardware on a breadboard, the following tests were performed and results were as desired, using an ESP32 dev board and its 3.3V power supply output:

Write Test:

- Wire the slave to the microcontroller as it will be during normal operation. Hardwire the slave's internal address to 0001.
- Disconnect the Open signal from the microcontroller and connect it directly to Vdd.
- Connect an LED with positive terminal at the slave's solenoid control signal and negative terminal at ground.
- Implement the communication protocol described in section 2.3.2. Set the max address variable to 3. Set the clock period slow enough for the human eye to see (1500 ms).
- Add code to output the current address to a hex display and connect a hex display to the specified pins.
- Boot the code to the ESP32 and run it.
- Observe the LED light up and remain lit.

- Manually connect the Open signal to ground.
- Observe the LED turn off.
- Halt the program and edit the code to turn the Open signal on and off depending on what the current address is. The specific sequence is arbitrary, but keep track of what the Open signal is when address == 1.
- Connect the Open signal back to the microcontroller, and add an LED from it to ground to see the Open signal changing.
- Boot the code and run it again.
- Observe the solenoid control LED remains at whatever value Open was set for address == 1, despite the addresses incrementing and the Open signal changing

Read Test:

- Wire the slave to the microcontroller as it will be during normal operation. Hardwire the slave's internal address to 0001.
- Implement the communication protocol described in section 2.3.2. Set the max address variable to 3. Set the clock frequency to faster than the minimum clock speed required by the ADC, which is 10kHz. We used 50 kHz for this test.
- Add code to save the sensor data from each channel when address == 1 and continuously output it to 20 pins (10 bits per each channel).
- Connect strip LEDs to the output pins to visually confirm the data that the microcontroller receives from the ADC's two channels.
- Use resistor division to manually set a distinct voltage at the input of Channel 0 and Channel 1 on the ADC (two identical resistors in series with Channel 0 connected in between gives Vdd/2 at Channel 2, three identical resistors in series with Channel 1 connected between the final resistor and ground gives Vdd/3 at Channel 1).
- Boot the code to the ESP32 and run it.
- Observe and write down the binary sequence corresponding to each channel on the LEDs.
- Compute what integer each channel's LEDs represent in unsigned binary. Divide this by 2^10 (1024) and verify that the decimal is within a 10% tolerance of the value created with the resistor divider.

All our tests with the communication between master and slave were satisfactory. The slave changed its Open Valve control signal when it should and remained the same when it should during the Write Test. When the clock period was set to 10 us, we saw LED response within 2 s of changing the moisture sensor's state, well within the 2 minutes we gave ourselves in the high level requirements for a system responsive enough for gardening purposes. The Read Test yielded correct results as well for both channels, outputting 0111111111 = 503 / 1024 = 0.4912 when set to Vdd/2, 0101010110 = 342 / 1024 = 0.3340 when set to Vdd/3, and 0100000001 = 257 / 1024 = 0.2510 when set to Vdd/4.

A test of the integration of Write and Read was performed by setting up a simple version of the makeDecision function on the ESP32 to turn a valve on or off based only on the current value of a plant's top level moisture sensor. The system performed exactly as intended, with the LED on one moisture

sensing subsystem's valve control signal turning off when the moisture sensor was placed in water and on when it was in open air, and two moisture sensing subsystems operated entirely independently of each other.

4. Costs

4.1 Parts

The costs of the necessary parts are shown below. Table 1 shows the total parts purchased in the course of designing this project throughout the semester, including development parts and using the actual quantities of materials purchased, using two moisture sensing subsystems. Entries marked with an asterisk designate costs for one moisture sensing subsystem and can be assumed to scale linearly with the amount of plants an individual consumer wishes to take care of.

Table 1 Total Parts Costs

Part	Manufacturer	Retail Cost ($)	Bulk Purchase Cost ($)	Actual Cost ($)
ESP32 Devkit	Espressif Systems	9.00	9.00	0.00
20x First round PCBs	PCBWay	2.00	2.00	Covered by the department
1x RPT-6003 AC/DC Power Converter	MEAN WELL USA	21.46	20.20	21.46
1x 3 prong outlet cord	Qualtek	5.34	3.10	5.34
1x LM1117-5.0	Texas Instruments	1.14	0.52	0.00
1x CONN RCPT HSG 6POS	JST Sales America	0.22	0.10	0.22
1x CONN RCPT HSG 3POS	JST Sales America	0.15	0.06	0.15
1x ESP32-S3-WROOM	Espressif Systems	3.48	3.48	3.48
10x Master Control Subsystem PCB	PCBWay	2.00	2.00	Covered by department
4x TVS Diode	Onsemi	0.24	0.05	0.096
1x 3 Position Terminal Block Plug	Adam Tech	0.98	0.41	0.98
1x 1/2 in. PVC Hose Adapter	Orbit	3.55	3.55	3.55
1x LP-55P Plastic Electronics Enclosure	Polycase	4.97	3.99	0.00
10x Moisture Sensing Subsystem PCB*	PCBWay	2.00	2.00	Covered by department
2x Plastic Water Solenoid Valve*	Adafruit Industries	6.95	6.95	6.95

4x SEN-13322 Moisture Sensor*	Sparkfun Electronics	6.50	6.50	6.50
2x MCP3004-I/P ADC*	Microchip Technology	2.97	2.26	2.97
2x 1N4001-T Diode*	Diodes Incorporated	0.20	0.04	0.20
2x SN74HC266N XNOR*	Texas Instruments	0.67	0.29	0.67
2x CD4013BE Flip Flop*	Texas Instruments	0.70	0.30	0.70
2x CD4049UBE Inverter*	Texas Instruments	0.69	0.29	0.69
2x CD74HC21E 4 Input AND*	Texas Instruments	0.73	0.31	0.73
2x TC74HC08APF 2 Input AND*	Toshiba Semiconductor and Storage	0.45	0.14	0.45
2x IRLB8721PBF Power MOSFET*	Infineon Technologies	1.05	0.46	1.05
5x 12 position connector*	TE Connectivity AMP Connectors	4.67	3.04	4.67
2x 8 position connector*	TE Connectivity AMP Connectors	2.62	1.71	2.62
2x WC-21F Outdoor Enclosure*	Polycase	14.15	9.25	0.00
8x LR1F30K Resistor*	TE Connectivity Passive Product	0.14	0.02	0.00
12x C317C104K5R5TA 0.1uF Capacitor*	KEMET	0.43	0.13	0.00
2x 1uF Capacitor*	Murata Electronics	0.74	0.22	0.74
2x 1/2 in. PVC Schedule 40 S x S x S Tee*	Charlotte Pipe	0.64	0.55	0.64
2x 1/2 in. PVC Schedule 40 Male MPT x S Adapter*	Charlotte Pipe	0.75	0.75	0.75
2x 1/2 in. x 3/4 in. F-Adapter Fitting*	DURA	1.27	1.27	1.27
10 feet 1/8" Heat Shrink Tubing	Electriduct	1.00 per foot	1.00 per foot	0.00
1 can Liquid Electrical Tape Spray	Performix	7.49 per can	7.49 per can	7.49 per can
1 can All-Purpose Fast Setting Clear Cement	Oatey	3.98 per can	3.98 per can	3.98 per can
10 feet ½ in PVC pipe	Charlotte Pipe	0.42 per foot	0.42 per foot	0.42 per foot
70 feet 22 AWG wire	Adafruit Industries	0.11 per foot	0.11 per foot	0.00

Total		369.41	255.00	142.02

The above table all pertains to the actual costs used for the development of one master control subsystem and two moisture sensing subsystems. We also report below, using bulk costs and half the quantities shown for the asterisk-marked moisture sensing subsystem parts, the estimated costs of units in mass production.

Costs of hardware per moisture sensing subsystem in bulk production: $46.73

Costs of master control subsystem in bulk production: $34.06

The costs of the waterproofing materials and PVC equipment are variable depending on the dimensions and construction of the consumer's gardens.

4.2 Labor

With an estimation of $48.00 an hour wage for 3 group members, each having given approximately 150 hours of work in the course of the semester, and a multiplier of 2.5x, we estimate a labor cost for the three of us at $54,000. Additionally, with an approximation of 2 hours of fabrication work at $21.96 per hour, we estimate $43.92 of fabrication labor costs.

5. Conclusion

When finishing and demonstrating our design, completing subsystem requirements and combining our individual subsystems yielded an unfinished product. First, we ran into issues when creating our UI subsystem and integrating it with our control subsystem. Creating and testing the frontend is one thing, but gathering data, saving data, and sending to the frontend gave us more room for errors in our data transfer pipeline. This includes data loss between cycles due to timing out from our websocket implementation for sending data to the users. The ESP32 will reset itself mid address cycle and restart the address cycling process. More devastating issues arose when we disconnected our system for water proofing. We had failed to create a system for wiring the slave and master board which caused the wrong wires to be put in the wrong places prior to demo. Spending more time on software testing and better allocation of decision making on project details would have significantly improved our process for designing and building our project.

5.1 Accomplishments

Individually, we were able to complete a UI subsystem that accepts formatted data from the ESP32 board, with updating data to the user. The moisture subsystem is able to output 2 channels of data from the moisture sensors and be received by the ESP32. The control system is able to cycle through addresses based on the number of plants inputted by the user. When budgeting for the project, we stayed really close to our limit by only slightly going over $150 by a few dollars.

5.2 Ethical considerations

Safety Standards: Following the IEEE guideline on the safety of our project, it is designed to be easy to use and compatible. It is safe to use and will not have the ability to cause property damage (IEEE Code of Ethics 7.8.9) **[3]**. Any high voltage components will be properly sealed to protect against the elements, to ensure user safety and prevent property damage. Informing the user how to use the device properly will be crucial to ensure safety and functionality.

Intellectual Property and Attribution: When developing unique Wi-Fi technology, it's important to credit others' work appropriately and ensure that our project respects existing patents and intellectual property rights. Properly citing and respecting the work of others helps maintain ethical standards (ACM Code 1.5). **[4]**

Privacy Concerns: Ensuring the privacy of users' data and information transmitted over the Wi-Fi network is crucial. Following privacy standards of ACM Code 1.6 **[5]**, each user has a right to privacy and privacy standards will be followed to ensure information accessible via the internet is protected. This will be done by safeguarding any of the user's personal information by following industry level privacy practices in our code.

5.3 Future work

Overall, a lot can be done to improve our design. Some future work we have considered was a complete water usage tracking system, auto detecting slave devices and an improvement for our wiring. In addition, we want to add features like failure detection, to notify the user when a valve or moisture sensor is not functioning the way it is supposed to be, i.e. the valve is stuck open or closed. PSI measurement to ensure the minimum PSI is available to open and close the valves. A better wiring system would be needed as well, to make connecting slaves to the master streamlined.

References

[1] "Why Your Solenoid Valve Needs a Diode - Collin's Lab Notes." Adafruit Industries. 2021. https://www.youtube.com/watch?v=-PYasR6Z0KQ

[2] "Automatic Irrigation System using an Arduino Uno." Rajesh, Circuit Digest. 2021. https://circuitdigest.com/microcontroller-projects/automatic-irrigation-system-using-arduino-uno

[3] "IEEE Code of Ethics." IEEE (Institute of Electrical and Electronics Engineers). https://ieee.org/about/corporate/governance/p7-8.html

[4] "ACM Code of Ethics and Professional Conduct." ACM (Association for Computing Machinery). 2018. https://www.acm.org/code-of-ethics

[5] Random Nerd Tutorials. "ESP32 Flash Memory - Save Permanent Data." Random Nerd Tutorials, 2 Mar. 2021, https://randomnerdtutorials.com/esp32-flash-memory/

[6] Random Nerd Tutorials. "ESP32 Async Web Server – Control Outputs." Random Nerd Tutorials, 23 Oct. 2020, https://randomnerdtutorials.com/esp32-async-web-server-espasyncwebserver-library/

Appendix A Requirement and Verification Table

Table 2 System Requirements and Verifications

Requirements	Verification	Verification (Yes or No)

• Can be launched successfully locally on a Web-App with intended UI-Subsystem.
  ○ UI-Subsystem loads up and user can click on buttons
  ○ Once user submits parameters, the data will be saved
	• A webpage with the intended arguments and buttons to add plants and configuration settings should pop up, ensuring frontend works properly
	• To ensure saved parameters, the system should only open the valve on scheduled time, the valve should not be opened outside of the scheduled time/date.
		Yes

• Can take in user parameters and communicate them to the control unit. ○ Exact same parameters show up in the ESP32 terminal	• To ensure saved configuration settings functionality, user clicks on “add new plant” button then fills in the four parameters: plant name, water usage, watering hours, and the minimum moisture level • User clicks “Send Message to esp32” button • To ensure proper functionality, the solenoid should open the valve of the specific plant at the specific watering hours and whenever	Yes

	the plant dips below its minimum top level moisture level. Confirm that this is true as no valves should be open outside these times. ● A plant's watering is stopped when the drain sensor becomes very wet. For demo purposes, manually place the drain sensor into the glass of water to stop watering.	
● Can receive plant moisture and water usage information from the master control unit and display it in a legible manner to the user. ○ The expected parameters show up on UI-Subsystem	● User clicks on "add new plant" button then fills in the four parameters: plant name, minimum moisture level, watering hours, and the watering days ● User attaches moisture sensor to the top of the soil of the plant (for demo purposes, leave sensors in open air) ● On the UI-Subsystem, a reading of the total water usage,plant address, plant health, and plant moisture information is displayed. Confirm that these values show up and the parameter does not show up as empty.	No. Communication of data from the moisture sensing subsystem to the master control subsystem was verified. Communication from the master control subsystem to the UI subsystem was verified. Integration of the software to allow moisture data from the sensors to the UI subsystem ran into compilation errors and was lost due to a lack of time.
● When the master pcb is in the WIFI_CONFIG mode upon first startup, the master will act as an access point	● Turn on system, press reset, grab your phone and search for open WIFI networks ● Connect to the Access	Yes

for the user to input WIFI credentials and attempt to connect to the WIFI. ○ AP Name: PLANT_SYS ● BLUE LED = WIFI_CONFIG	Point (AP) named PLANT_SYS with password: config23 ● Follow URL: url_placeholder on your phone to interact with the system ● A prompt will appear to turn on and off an onboard Blue LED to verify connection. There will also be two textboxes to enter the name of your WIFI network and WIFI password. ● Press enter to save password	
● Wifi credentials are saved for the main system to connect to WIFI ○ PLANT_SYS = DNE ○ Green LED = WIFI Connected	● Confirm that your network connection to PLANT_SYS no longer exists ○ a Green LED will be turned on to confirm a successful WIFI connection	Yes
● The system can perform serial communication between the moisture sensing subsystems and control subsystem. Each moisture sensing subsystem operates independently and is not influenced by other plants. ○ Switching between dry and wet environments	● Use system in normal operation with moisture sensors in plant soil. (for demo purposes, sensors will be either in open air or a glass of water). ● Select one moisture sensing subsystem, use the UI subsystem to set the minimum moisture level to the mid-range value. ● Alternate placing both moisture sensors for the plant in question between a pot of	Yes

on one plant opens and closes that plant's valve while not affecting the valve of any other plants.	completely dry soil (open air for demo purposes) and a container of water. ● This specific unit should have its valve open when in the dry soil and closed while in the water; the other units should not be affected by this switching.	
● The moisture sensing subsystem can communicate data to the master control unit within +/- 10% accuracy ○ Data which appears on the UI subsystem for the status of each plant is within +/- 10% of a voltmeter reading of the plant's sensor's output	● Use system in normal operation with moisture sensors in open air. ● Use a voltmeter with negative probe at a plant's top level moisture sensor's ground pin and positive on the sensor's output pin. ● Read the voltmeter's measurement. ● Compare with the value on the UI subsystem for the plant in question and ensure the value is within +/- 10%.	No. All steps were completed except for being able to send data to the UI subsystem. Moisture sensor -> master control microcontroller accuracy was verified to +/- 2% accuracy.

Appendix B Custom Read/Write Communication Protocol

- Start at a new address, assume write was just set to 0 and address pins are set to 1111 (invalid address on all the slaves). Set MOSI pin to 0.
- ~Wait for MOSI signal to propagate~
- Set address pins to the current address number. This, combined with a low Write signal, will pull Chip Select Low down on the desired slave
- ~Wait for address logic to propagate~
- Set MOSI pin to 1 (Start Bit for SPI communication with ADC)
- Set MOSI pin to 1 (Select single-ended mode for ADC read)
- Set MOSI pin to 0 (D2, MSB of channel selection logic on the ADC)
- Set MOSI pin to 0 (D1)
- Set MOSI pin to 0 (D0, LSB of channel selection logic on the ADC. Channel 0 is selected for read).
- ~Wait 3 cycles for ADC sampling~
- Read and store MISO pin (B9, MSB of ADC data)
- Read and store MISO pin (B8)
- .
- .
- .
- Read and store MISO pin (B0, LSB of ADC data)
- Set address pins to 1111 (invalid address for all slaves; this pulls CS Low high, which is necessary before performing another read with the other moisture sensor on the plant)
- ~Wait to allow address pins to propagate~
- Set address pins back to the current address
- ~Wait to allow address pins to propagate~
- Set MOSI pin to 1 (Start Bit for SPI communication with ADC)
- <Repeat the SPI protocol again, this time with the LSB of the channel selection address set to 1 to read from Channel 1>
- ~Wait to allow for computation of open/close valve decision~
- Set Open Valve pin to the decision reached on whether the plant should receive water (1) or not (0)
- ~Wait for Open signal to propagate~
- Set Write pin to 1, Set MOSI pin to 0
- ~Wait for Write signal propagation and storage of decision in the plant's flip flop~
- Set Write pin to 0, increment address value (wrap back to 0 if max address was reached), set address pins to 1111, reset cycle counter

This repeats constantly on the ESP32, with breaks after getting through the entire address set to communicate with the UI subsystem.