Controlled Hydration Automated Nutrient Optimization System

Elliott D'Amato, Brian Ericson, Brandon James, Adrian Vergara

Dept. of Electrical and Computer Engineering, University of Central Florida, Orlando, Florida, 32816

Abstract - This project assists amateur plant enthusiasts by automatically watering a plant based on user-defined moisture threshold and timer values. We improve on existing products by implementing automatic drainage and solar charging mechanisms in order to reduce the environmental footprint. We also implement monitoring of the sunlight exposure and water supply level. Our web application allows users to configure watering parameters and monitor plant status. Configurations for multiple plants may be saved and retrieved at any time. The application also provides users with notifications regarding plant health.

***Index Terms* – capacitive sensor, float switch, solar panel, Li-ion battery, ReactJS**

I. Introduction

A. Engineering Specifications

The highlighted engineering specifications in Table I are intended to be demonstrated in the demo video. First, every time the ESP32 wakes up from sleep mode, it will measure the sensor values which are transmitted to the web server. The time for this activity should be less than 5 minutes. Second, the system should be able to detect that the water level is low when less than 0.5L of water is in the supply tank. Third, the water pump, which is driven by the ESP32 and 12V regulator, should have a flow rate of 0.1 to 5 mL per second. Lastly, the time to activate the water pump after detecting water in the supply tank and low soil moisture should be less than 1 minute.

TABLE I
ENGINEERING SPECIFICATIONS

Parameter	Specification
Sampling of sensor data frequency	15 - 20 min
Time to observe sensor data in web app after wakeup	< 5 min
Accuracy of water level	< 0.5L
Water Flow Rate	0.1 - 5 mL / s
Supply Water Level Accuracy	>= 90%
Cost of final design	< $75 USD
Battery life	21-27 hours
Time to activate water pump after low soil moisture detection	< 1 min
Temperature	21 C < T < 50 C

II. System Overview

A. Hardware Block Diagram

The C.H.A.N.O.S. (Controlled Hydration Automated Nutrient Optimization System) platform is designed as a modular, solar-powered plant life support system that autonomously monitors environmental conditions and controls irrigation. The hardware architecture, shown in Figure 1, consists of three primary subsystems: Power, PCB Control, and Life Sustain.

The Power subsystem features a dual 18650 2S Li-ion battery pack charged through a BQ24650 solar charge controller with MPPT tuned for an 18V photovoltaic panel. Energy from the pack feeds a LM2577 based 12V boost converter used to drive the water pump and a LM2596 based 3.3V buck regulator that powers the ESP32 microcontroller and sensors. A secondary MP2672A USB charger provides auxiliary charging for backup or indoor use, with an ideal diode isolating the solar and USB inputs to prevent backflow.

The PCB Control subsystem centers around the ESP32-WROOM MCU, which processes sensor data and controls the pump driver through GPIO logic. The PCB also integrates test points, voltage regulators, and communication interfaces for debugging and wireless connectivity to the system's cloud-based web server.

The Life Sustain subsystem includes the capacitive soil moisture sensor, fixed point water level sensor, and light sensor - all custom-designed PCBs interfaced with the MCU. These boards provide continuous data to determine plant hydration needs and optimize irrigation. The MCU activates the pump when soil moisture drops below a set threshold, maintaining ideal conditions while minimizing water usage.

This distributed hardware design with separate boards for power regulation, sensing, and actuation simplifies

development, improves reliability, and allows modular testing of each subsystem under varied environmental conditions.

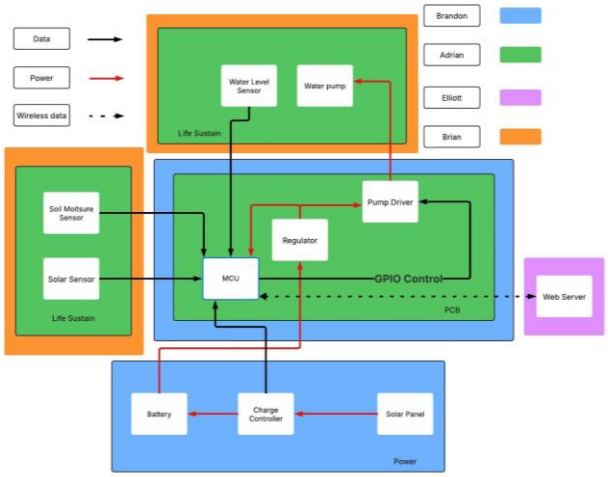

Fig. 1. Hardware Block Diagram.

B. Final Design

The overall encasing for the project was done by using a 3d printer. We can separate the design into 3 sections: 1. The pot, 2. The Tank, and 3. The Component Housing.

Fig. 2. Pot Design.

The pot is made as a single solid structure with two special features that include an inner grove to have the tubing in and a hole at the bottom to drain any excess water that the plant might not absorb. This hole also leads to the upper lid and drainage hole of the water tank for reuse.

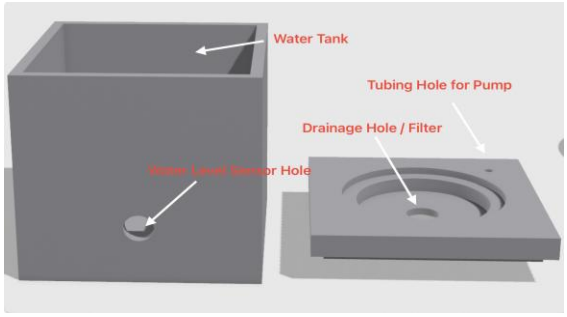

Fig. 3. Tank Design

The tank is made up of two pieces, the lid and the reservoir. The lid has a circular groove at the top to comfortably hold the pot in place. It also contains a deeper circular section for a filter. The reservoir is a 7inch x 7inch container which holds the water. On the side we have a hole that is used to house our water level sensor which will be explained in greater detail later.

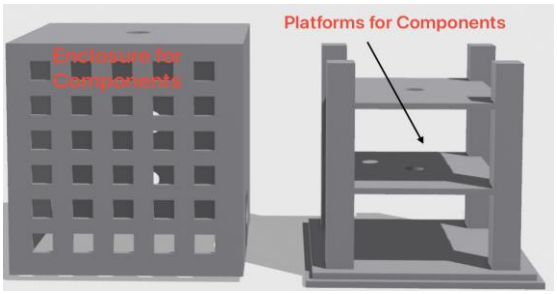

Fig. 4. Component Housing.

The component housing is also made up of 2 sections. This includes the enclosure and platform area for components. One side of the enclosure includes a 5 x 6 square pattern to ensure proper ventilation for our components. In addition to this there are 4 holes, 3 on the side, one on top, to ensure that our components can reach the other sections. The platform area is separated into 3 levels to allow for variety in where we can place our components. The size of the enclosure is about 9inch x 9inch while the platform area is around 9inch by 8.5inch.

Overall, the design of the project was made with modularity in mind to account for situations where things would leak or if something needed to be up and/or downsized. Many of the features mentioned before needed to be tailored to our needs which made 3d printing a versatile option for the project.

III. MCU & PERIPHERAL HARDWARE

C. MCU PCB

The ESP32-WROOM microcontroller module is placed on the edge of the PCB so that no signal traces run underneath the built-in RF antenna. The WROOM also contains built-in oscillators and flash memory. Of the 25 available general purpose IO (GPIO) pins, 3 are connected to LEDs used for debugging purposes and one is used for on-board pump control [5]. The rest, as well as the UART and power pins, are tied to through-holes for soldering to pin headers or wires in order to supply power to or communicate with the other PCBs.

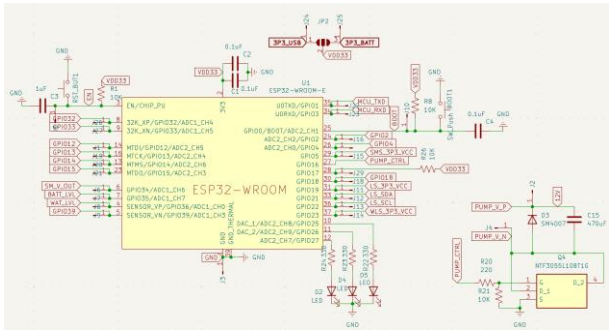

Fig. 5. MCU PCB Schematic.

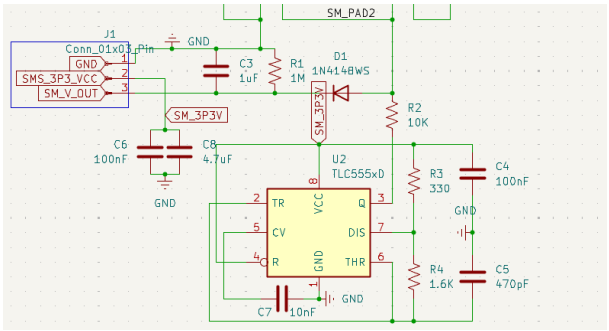

Fig. 6. Soil Moisture Sensor Schematic.

B. Water Pump

The water pump control circuit consists of a mosfet transistor, with the gate connected to an ESP32 GPIO pin, drain connected to the negative terminal of the pump, and source connected to ground. The pump is connected in parallel with a flyback diode and capacitor. The pump's positive terminal is connected to the 12V line.

The ESP32 controls the pump activity via a GPIO pin. When the pin is set high, the mosfet is activated, the negative terminal of the pump to ground, closing the circuit and allowing current through the pump.

C. Soil Moisture Sensor PCB

The soil moisture was adapted from the Analog Capacitive Soil Moisture Sensor by DFRobot. A TLC555 timer IC generates a square wave signal with fixed frequency and duty cycle [4]. The square wave signal is input to an RC circuit. The capacitor is composed of two copper pads that extend across the sensor. As the moisture of the soil increases, the dielectric constant increases, which increases the capacitance. The increased capacitance lowers the voltage that the capacitor can be charged to, because of the fixed period and duty cycle. The signal is passed through a diode which sets the low state voltage near 0V instead of 0.3-0.5V. The signal is finally passed to the output pin which is also connected to a parallel RC circuit, which filters out high frequency noise due to impedance of the capacitor. Decoupling capacitors were used to stabilize the input signal. During the PCB design, lower frequency signals were routed between layers first, if necessary. Stitching vias were used to improve signal integrity and thermal conduction.

D. Light Sensor PCB

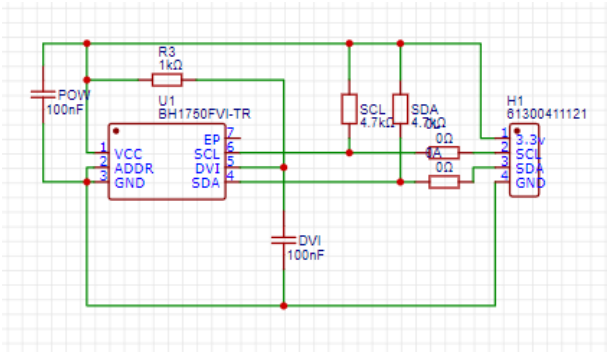

Fig. 7. Light Sensor Schematic.

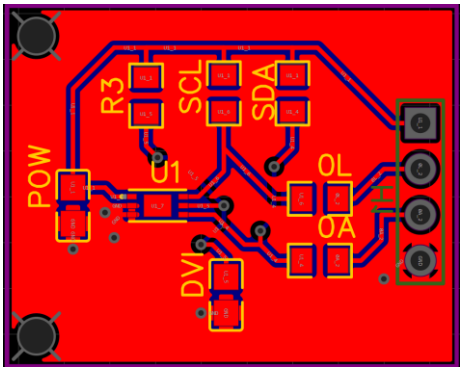

Fig. 8. Light Sensor PCB Design.

The light sensor PCB utilizes the BH1750 digital ambient light sensor, which communicates with the ESP32 microcontroller over the I2C protocol. The BH1750 provides a 16-bit digital output proportional to the ambient light intensity in lux, eliminating the need for analog to digital conversion [1]. Pull up resistors were included on the SDA and SCL lines to ensure stable communication and 0 ohm resistors were included for troubleshooting.

Decoupling capacitors were placed on the sensor's VCC and GND pins to reduce power noise. The BH1750 operates at 3.3V and draws 120uA, making it suitable for our low

power application needs. The sensor, located on the BH1750 chip, is oriented on the PCB to face outward, ensuring that the photodiode is exposed to ambient light without obstruction. Ground pours were added around signal traces to minimize noise coupling.

E. Water Level Sensor & PCB

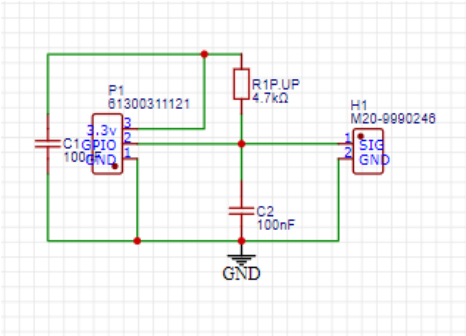

Fig. 9. Water Level Sensor Schematic.

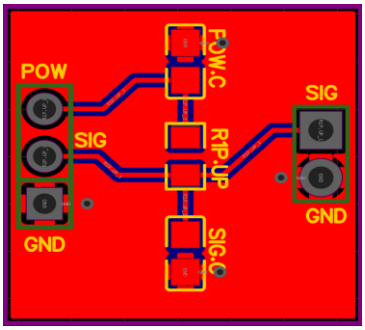

Fig. 10. Water Level Sensor PCB Design.

The water level sensor PCB interfaces with a horizontal float switch to detect the water level in the tank. The switch functions as a simple on/off mechanical sensor that closes when the water reaches a set height [2]. The sensor output is sent to an ESP32 GPIO pin configured as a digital input. A pull up resistor ensures a stable high logic level when the switch is open.

When the water level rises, the switch will float, leaving the circuit open; when the water level is low the switch will be closed. The PCB includes a two-pin terminal block for easy connection to the float switch and a 3 pin terminal for the GPIO connection, ground, and a 3.3V supply to pull up the resistor.

IV. Power System Hardware

A. Overall Power Usage

Based on the hourly runtimes and current draw of each device, the system averages about 0.22 W of power, or 0.22 Wh per hour. Most of that comes from the ESP32 running for five minutes and the 12 V pump running for about a minute and a half, while the sensors only draw brief, low currents each cycle. Runtime was calculated using equation 1, where $E_{battery}$ = 25.2 Wh and P_{avg} = 0.22 W.

$$t = \frac{E_{battery}}{P_{avg}}. \quad (1)$$

Equation 1 gives roughly 115 hours, and after factoring in efficiency losses (≈80–85%), the realistic runtime is around 94–100 hours, or about 4 days per charge. Total daily use stays under 6 Wh, making it ideal for solar operation.

TABLE II
Component Power Specifications

Device	Voltage	Current	Power	Runtime per hour
Pump	12V	250mA	3W	1.5 min
Float switch	3.3V	.1mA (current rating)	.33W (max power)	1 min
Soil Moisture	3.3V	~120 uA	~0.4 mW	1 min
Light Sensor	3.3V	~120uA	~.4mW	1 min
MCU	3.3V	500 mA	1.65 W	5 min

B. 3.3V Regulator PCB

The 3.3 V rail is generated by a buck converter based on the LM2596-3.3. The regulator steps the 2S Li-ion pack (6.0–8.4 V) down to a fixed 3.3 V for the ESP32 and sensors. The design follows the standard topology with an input filter, 27 µH shielded inductor, Schottky freewheel diode, and output filter. The LM2596 switches at 150 kHz and supports up to 3 A continuously when thermally managed and paired with low-ESR capacitors.

At the worst-case full battery (Vin = 8.4 V, Vout = 3.3V), the duty cycle is 0.39 according to equation 2. With L = 27 µH and fs = 150 kHz, the inductor ripple current is 0.5 according to equation 3.

$$D \approx \frac{V_{out}}{V_{in}}. \quad (2)$$

$$\Delta IL \approx \frac{V_{out}(1-D)}{L \cdot f_s}. \quad (3)$$

At 1 A load, the inductor peak current is ≈ 1.25 A, well below the SRR1210A-270M saturation rating. Using equation 4, the diode's average current at 1 A load is approximately 0.61 A, within the DFLS240L's 2 A rating.

$$(1 - D)I_{out}. \quad (4)$$

PCB/layout: High-di/dt paths (SW-L1-D1-COUT) are kept short and wide; CIN is placed adjacent to VIN/GND; the power ground returns converge to a single star node near the IC to minimize loop area and noise. Test points TP_3V and TP_GND support bring-up and verification.

Thermal analysis was performed with θJA ≈ 50 °C/W and η = 80 % at 8.4⟶3.3 V. Even at 1 A load and 25 °C ambient, the junction remains well below the 150 °C limit. At a warmer 50 °C ambient, the same 1 A case projects Tj ≈ 91 °C, still within safe operation with adequate copper pour.

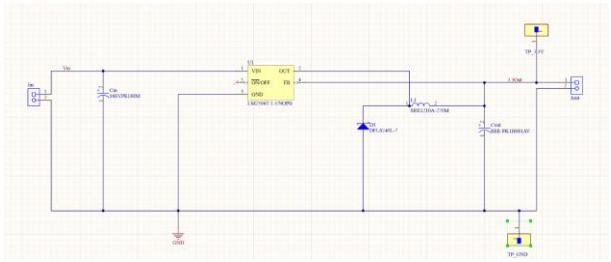

Fig. 11. 3.3V Regulator Schematic.

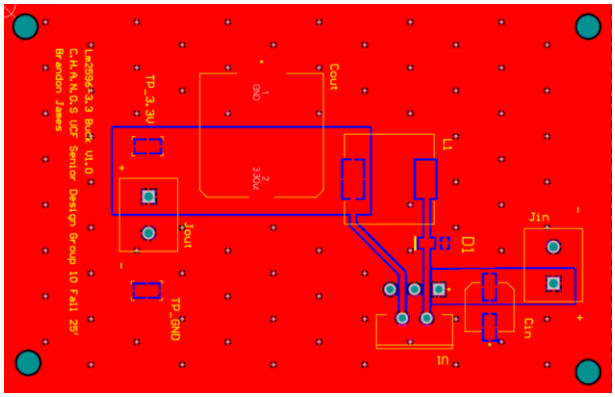

Fig. 12. 3.3V Regulator PCB Design.

C. 12V Regulator PCB

The 12 V rail is generated by a boost converter based on the LM2577-ADJ switching regulator, which steps the 2S Li-ion pack (6.0 V – 8.4 V) up to a regulated 12 V output for the system's water pump. The LM2577 operates at approximately 52 kHz and is capable of sourcing up to 3 A switch current, providing sufficient margin for 12 V at 1 A continuous output.

The circuit employs a non-synchronous boost topology with the following components: input filter capacitor, 100 uH shielded inductor, Schottky diode for recovery, output filter capacitor for ripple suppression, output voltage divider, and series RC compensation network.

At Vin = 8.4 V, the converter duty cycle is 0.30 where Vout = 12V.

$$D \approx 1 - \frac{V_{in}}{V_{out}}. \quad (5)$$

The inductor ripple current is approximated by equation 3. Using values Vin = 8.4V, D = 0.3, L = 100 µH, and 52 kHz, then ΔIL = 0.48A. The equation yields a peak current of ≈ 1.24 A at 1 A load, within the inductor and diode ratings.

High-current paths between the switch node, inductor, diode, and output capacitor were kept short and wide to reduce losses and EMI. The LM2577 TAB is tied to ground and thermally connected to a large copper pour with multiple vias. The feedback and compensation traces were routed separately from the power ground return to minimize noise coupling. Test points TP_12V and TP_GND provide access for debugging and verification.

Thermal analysis was performed with θJA = 50 °C/W and η = 85 % efficiency across a 2S range. Even under full-load conditions, the LM2577 junction temperature remains below 100 °C at 25 °C ambient, and below 125 °C at 50 °C ambient, providing a safe thermal margin.

This design demonstrates reliable 12 V regulation and thermal stability while supplying high transient current to the water pump, completing the upper-voltage subsystem of the C.H.A.N.O.S. power board.

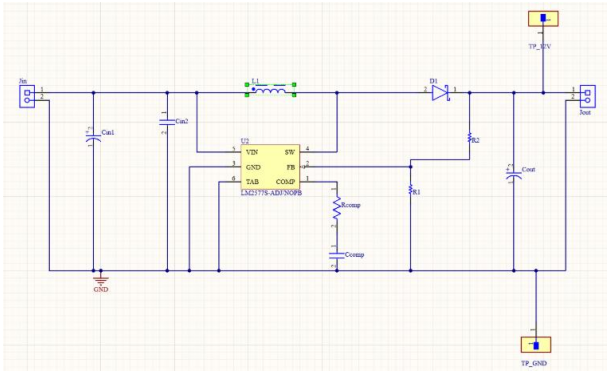

Fig. 13. 12V Regulator Schematic.

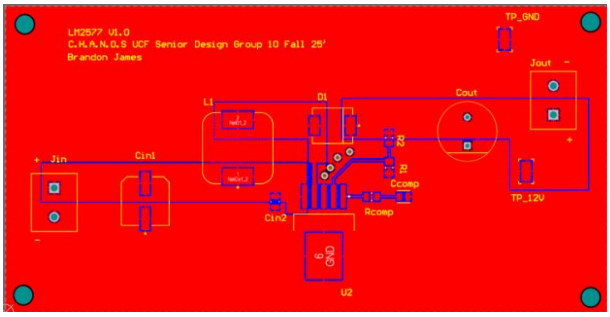

Fig. 14. 12V Regulator PCB Design.

D. Solar Charging PCB

The solar input and Li-ion charging subsystem is built around the BQ24650 synchronous buck charger configured for a 2-series (2S) Li-ion pack. The charger accepts power from an 18 V photovoltaic panel (Voc ≈ 21–22 V) and regulates charge into the battery while tracking the panel's maximum power point (MPPT). The design priorities were: (1) hands-off operation without firmware control, (2) predictable MPPT behavior under fast irradiance changes, and (3) robust protection for a fielded enclosure.

Topology & key functions. The BQ24650 operates as a high-efficiency synchronous buck from panel to battery. It implements:

- MPPT (MPPSET): panel voltage is divided down to the MPPSET pin and compared to an internal reference. We target VMPP ≈ 17.5 V for the chosen panel. Using the standard divider equation (equation 6) with VREF ≈ 1.2 V, choosing Rupper ≈ 270 kΩ and a practical Rlower (~20 kΩ) yields VMPP ≈ 17.5 V.

$$VMPP = VREF \times (1 + \frac{R_{upper}}{R_{lower}}). \qquad (6)$$

Component choices. The power stage uses the EVM-proven FETs and inductor values to minimize switching losses and maintain compatibility with the selected photovoltaic panel. The chosen 18 V panel (nominal Vmpp ≈ 17.5 V, Impp ≈ 0.8–1.0 A) aligns perfectly with the MPPT target of the BQ24650 circuit. This ensures the panel operates near its peak power under standard test conditions and allows consistent charging even with moderate irradiance or partial shading. Input and battery capacitors are low-ESR aluminum electrolytics paralleled with ceramics for high-frequency decoupling, while the sense resistor and ISET2 network are tuned to limit charge current in accordance with the panel's rated output.

The charger's output connects directly to the 2S pack through an ideal diode, which then feeds the two downstream rails (12 V boost for the pump and 3.3 V buck for logic). This architecture allows simultaneous system operation and battery charging without firmware involvement and eliminates I²C dependencies while still providing visual charge status via LEDs and accessible test points for PV, BAT, and GND.

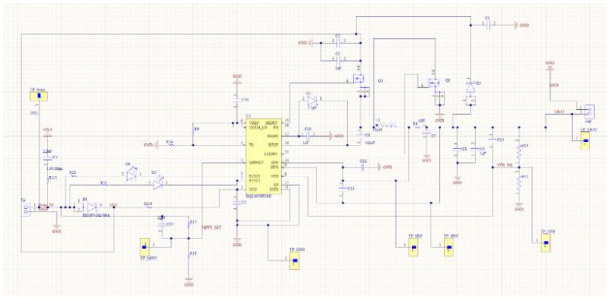

Fig. 15. Solar Charger Schematic.

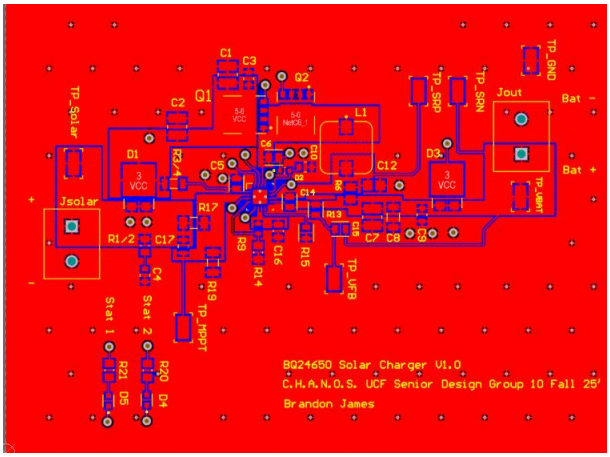

Fig. 16. Solar Charger PCB Design.

E. USB Charging PCB

The USB charging circuit uses the MP2672A single-chip power-path and charging controller to provide a backup charging method for the 2-series (2S) Li-ion battery pack. The circuit is powered from a standard 5 V USB-A input, enabling convenient off-grid or indoor charging when solar energy is unavailable.

The USB 5 V input connects through an EMI filter and bulk capacitor (CIN = 220 µF / 10 V) to the VIN pin. The MP2672A's internal control loop limits input current according to the programmed USB limit resistor (typically 1 A), preventing excessive draw from standard USB ports. The charger operates in a buck configuration, regulating charge current and voltage via an external sense resistor (RSENSE = 100 mΩ) and the ISET resistor network, which sets the fast-charge current to ≈ 1 A and the float voltage to 8.4 V for the 2S pack.

When both solar and USB inputs are present, the system automatically prioritizes the solar charger because of the ideal diode on the BQ24650 output. This diode prevents backflow of current into the solar input when USB power is active, ensuring proper source isolation.

The USB port, input capacitor, and sense resistor are placed near the MP2672A to minimize parasitic resistance. Power and ground planes are continuous under the chip and connected through multiple vias for heat dissipation. The battery connection is shared with the solar board through a common connector labeled BAT+ / BAT–, while the system output (SYS) connects to the main power bus feeding the 12 V and 3.3 V regulators. The layout ensures isolation between the USB and solar paths while maintaining a low impedance connection to the shared battery node.

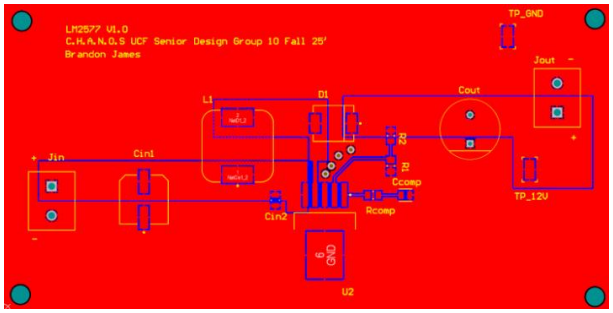

Fig. 17. USB Charger PCB Design.

V. Embedded Software

As illustrated in Figure 18, the ESP32 first enters the IO Init state upon power-up or sleep mode wakeup, where it initializes the required IO and I2C pins and sets the output pins low.

If a valid Wi-Fi SSID and password exist in the ESP32 flash memory, the ESP32 connects to the Wi-Fi network as a station (STA). If no valid info exists in the flash or the Wi-Fi connection is not successful, the ESP32 configures itself as an access point (AP). The ESP32 hosts an asynchronous web server and displays an HTML form at a predefined IP address. The user enters this IP address into a browser and submits their Wi-Fi information on the web page, which the ESP32 will write into flash memory.

After connecting to the Wi-Fi network, the ESP32 will receive a pot UUID from flash memory. If no UUID exists in flash (i.e. initial power-up), the ESP32 will receive a UUID from the web server and write it to the flash. This UUID is used in the URL when making GET/POST requests to the web server.

In the threshold init state, the ESP32 makes a GET request to the web server for sensor threshold values: maximum sunlight, sensor sampling period, soil moisture percentage, and watering duration. If any of the threshold values are not within certain bounds or limits, the ESP32 will make another GET request until all values are valid.

Within the peripheral state, the battery level is measured. If the battery is less than 0.5V, the ESP32 does not check the remaining sensors in order to recharge. Otherwise, the water level and soil moisture are measured. If the water level is not low and the soil moisture is higher than the threshold value, the water pump is activated for a duration defined by the user. The sunlight amount is updated if the lux value is greater than a certain threshold.

Within the sensor data update state, the ESP32 serializes the sensor measurements into a JSON document, which it transmits in a POST request to the web server. The notifications for the web application are handled by the web application backend code, not the embedded ESP32 code. After this state, the ESP32 sets a timer wakeup equal to the sensor sampling period and enters deep sleep mode.

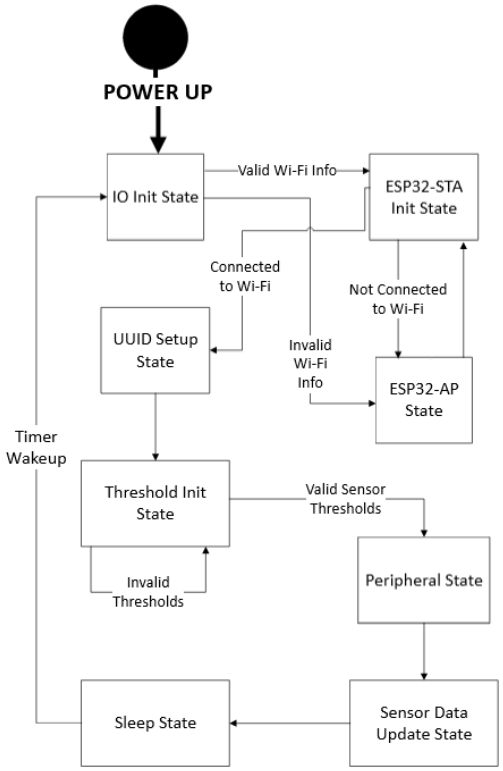

Fig. 18. Embedded Software Block Diagram.

VI. Communication

A. USB-UART PCB

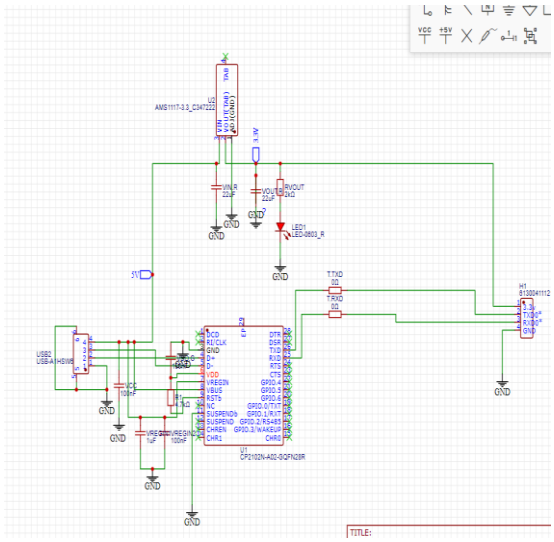

Fig. 19. USB-UART Schematic.

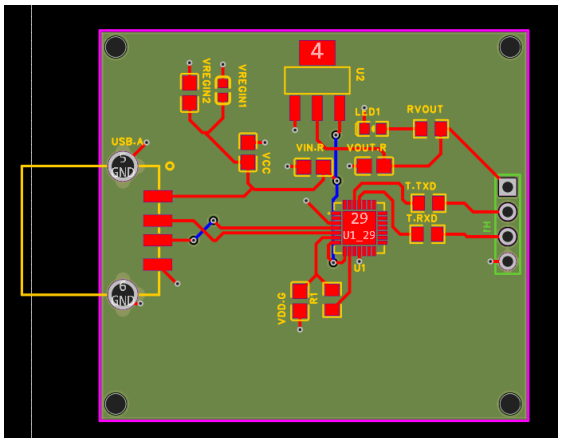

Fig. 20. USB-UART PCB Design.

The USB/UART interface PCB provides serial communication between the ESP32 microcontroller and our computer for programming and debugging purposes. It is based on the CP2102 USB to UART bridge controller, which converts USB data signals to standard UART levels [3].

The module operates from the 5 V USB supply and a voltage regulator brings it down to 3.3 V for logic communication with the ESP32. The CP2102 typically draws around 20 mA. Decoupling capacitors are placed near the power pins to minimize voltage ripple, and short differential D+ and D– traces are maintained for signal integrity.

B. Wireless Communication

The ESP32 WROOM has a built-in RF antenna which supports 2.4G Wi-Fi and Bluetooth. Wi-Fi was selected in order to communicate with the web server over the client's wireless network. HTTP was selected as the application layer protocol due to its ubiquity in modern networking and the simple messaging that is required between the ESP32 and web server. HTTP communication between the ESP32 and web server is programmed in the ESP32 software using a third party library.

VII. Web App Software

A. Server

The server of the web application provides API endpoints to allow for communication of data between the user client and the pot's embedded firmware. The server relies on a traditional approach to web development, where the client and the pot send HTTP requests to the server's API endpoints, and the server responds to those requests by providing or receiving information. The server mainly operates through the use of Express to define API and routing information, and communicates the data received by the API to a database.

For the API implementation, there are two main categories that an endpoint can fall under; those being endpoints for requests from the pot, and endpoints for requests from the user. The pot endpoints mainly take input parameters in the form of the pot's UUID, with the exception of the genUUID endpoint, which the pot uses to register itself with the server, and get its own UUID which it will then store for the purposes of communicating with the server.

The API endpoints for the pot do not require the pot to directly understand any of the underlying data structures stored in the server's database; instead, the pot interacts with all data that is connected to the pot object stored within the database, taking care of connections to other database objects, such as plants, on the server side when data from these objects must be sent to the plant, which simplifies the amount of data that needs to be stored by the pot and makes it easier to update that data on the server and have those updates cascade to the pot.

The user endpoints allow the user to request data from the server. The user endpoints use the user's UUID as a parameter for determining which database objects to access, considering sufficient authentication is provided. From the user's point of view, there are three main objects which are accessible via these API endpoints: users, plants, and pots. These objects are all implemented in the database and passed to the client through HTTP requests made from the client to the user endpoints, in the form of JSON objects and lists of JSON objects, in the case where multiple objects are fetched.

The database defines both the organization and structure of data that is stored within and the relationships between specific objects within the database. The objects that we care about for our database model include users, plants, and pots, as well as other helper objects such as notifications which were created to aid these three main objects. For each object, we assign it a UUID, which serves to allow us to identify objects, as well as connect certain related objects together, such as the pot object holding a reference to a user object UUID to connect a given pot to a specific user.

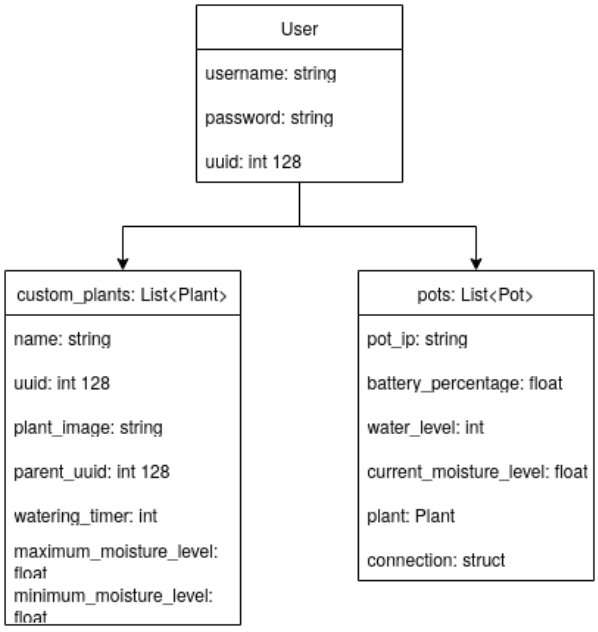

Fig. 21. Database Object Diagram.

B. Client

The client of the web application uses a React framework to display data received from a specific pot for its user as well as allowing the user to update static plant data that is used by the pot. The features of the client include:

- Providing a process for registering and authenticating users
- Allowing users to modify or remove their own information
- Registering pots to users
- Adding plants to users
- Tying a plant with specific information to a given pot
- Displaying pot information
- Displaying warning notifications in the event that the pot returns data that signals that user intervention is required

In order for the user to access the full features of the client, they must first register an account through the client. Successful registration will result in a user account being created with a UUID in the server database. The user can then use their credentials to log into this account and access data associated with it.

Of the data associated with the user account, the most important would be the pot data. Following initialization of the pot, the device will register itself with the server. Once the pot is registered with the server, it will direct the client to a pot registration page, with the pot's UUID as a parameter. The client can then use this page to register the user account with the pot object that is already connected to the physical device. Once connected, the pot will appear on the user's homepage but will still require further setup.

Additional setup includes connecting a plant to the pot, which is accomplished through allowing the user to open a more detailed view of the pot, which then allows the user to modify the pot's plant. Here, the user selects either a preexisting plant with information already available or creates a new plant with information that they populate. This new plant will be tied to the user account and will not be visible globally as with preexisting plants.

Once fully connected, the user will be able to view data collected by the pot, for all pots that they may have connected to their account. The user will be able to modify the pot by connecting it to new plant data, which will update the pot's embedded firmware with the new data as well, and they will be able to view notifications at the top of their homepage when a pot requires user intervention.

In terms of design of the client, the focus is on creating a design that follows our UI guidelines. These guidelines define the UI and UX rules that promote an accessible and adaptable UI, and focus on:

- Adaptability of the interface to different screen sizes
- Content-centric view that promotes displaying content in a way that promotes the use of scrolling to see more content instead of clicking on new tabs
- Readability of text in the application through the use of a small, high-contrast color palette
- Accessibility through the use of Alt Text and ease of navigation

This is accomplished through the use of React to define and create components that allow us to implement the required feature set of our web application, while adhering to our UI guidelines.

Fig. 22. Mock Web App User Interface.

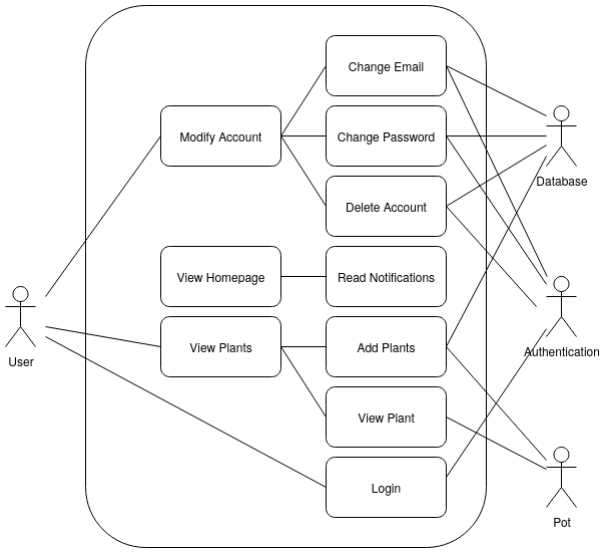

Fig. 23. Use Case Diagram for Web Server Client.

ACKNOWLEDGEMENTS

We would like to extend our heartfelt thanks to our professors Dr. Chung Yong Chan, Dr. Weeks, and Dr. Aravinda Kar for their continuous guidance and insightful feedback throughout the research and development of this project.. Their mentorship played a vital role in the success of the C.H.A.N.O.S. project.

We are also deeply grateful to our reviewers Dr. George Atia, Dr. Jonguok Choi, and Mr. Mark Maddox for dedicating their valuable time to evaluate our project. - Brandon J., Adrian V, Brian E., and Elliot D'Amato

I would also like to thank my wife for the support throughout this project, balancing a full time job and finishing this degree has been quite a journey. - Brandon James

I'm just happy I'm done. But a big thanks to my mom and my sister for helping me tremendously on this journey. I will spend the rest of my life in an effort to pay you back. - Adrian Vergara

I thank my mother (Jenifer), brother (Jon), uncles (Randy and Tony), and grandparents (Sharon and John) for all of their support. - Brian.

I would like to thank my parents for supporting me all these years; my mom, who showed me what it means to be an engineer, and my dad, who showed me how to work hard to achieve my goals. I would also like to thank my grandmother, who was also there for me when I needed her and helped me to truly understand what it means to have someone rooting for you. Rest in peace, grandma. - Elliott D'Amato

REFERENCES

[1] Adafruit. *Adafruit BH1750 Ambient Light Sensor*. Adafruit Learning System, https://learn.adafruit.com/adafruit-bh1750-ambient-light-sensor/overview.

[2] Ximimark. *Water Level Sensor Horizontal Float Switch for Arduino*. Amazon, https://www.amazon.com/Ximimark-Sensor-Horizontal-Switch-Arduino/dp/B0B7WPBPBC/.

[3] Silicon Laboratories. *CP2102/9 Single-Chip USB-to-UART Bridge Data Sheet*. Rev. 1.8, 17 Jan. 2017, www.silabs.com/documents/public/data-sheets/CP2102-9.pdf.

[4] "TLC555 LinCMOSTM Technology Timer." *Texas Instruments*, 1 Nov. 2023, www.ti.com/lit/ds/symlink/tlc555.pdf.

[5] Espressif. *ESP32 Series Datasheet.* Ver. 4.9. Espressif, https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.