"deMETER" Soil Monitoring System

Ryan Matthews
rcm62@uakron.edu

Rachel Rummer
rar86@uakron.edu

Temilolu Fayomi
tcf12@uakron.edu

Alex Fuller
ajf125@uakron.edu

deMETER Remote Soil Monitor

Final Design Report

Design Team 07

Temilolu Fayomi (TF) – CPE (+ESI)

Alex Fuller (AF) - EE

Ryan Matthews (RM) – EE (HONORS STUDENT)

Rachel Rummer (RR) – CPE (+ESI)

Dr. Hamid Bahrami - FA

4/25/2022

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Demeter is the Greek goddess of the harvest. The deMETER soil probe is designed to aid hobbyist gardeners, small-scale farms, and nurseries to monitor their dynamic soil conditions and maximize their harvest. The probe is a self-powered system that can monitor the moisture and essential nutrients of the soil profile to determine which areas should receive water and fertilizer. This would significantly cut water and fertilizer waste. The solution would include an embedded system with sensors that provides continuous monitoring of an area of soil. This device would connect wirelessly with another embedded system via an app with a streamlined user interface. The user interface will show the current moisture and nutrient levels of the soil being monitored and allow for the user to set a moisture/nutrient level setpoint. If the current level of the soil is at, above, or below the setpoint, it will trigger an alarm to notify the user to address the issue.

I. PROBLEM STATEMENT

A. NEED

With the sudden arrival of COVID-19 and the pandemic, gardening and small-scale agriculture have been on the rise. According to a survey called “Gardening in a COVID-19 World”, it found that 86% of homeowners plan to continue gardening in 2021. 47% said that they will be planting more and expanding their garden spaces next season.

Considering this, many more people are taking time to care for a home garden which requires quite a bit of work. They must ensure that plants are watered well and have enough fertilizer and sunlight. In addition, gardens and farms are not homogenous and have many kinds of plants that require different needs. Overwatering or underwatering plants can be detrimental to their health and decrease the yield of crops they produce. Similarly, too much or not enough fertilizer can also have severe negative consequences on the health of the plant.

B. OBJECTIVE

The objective is to create a device that measures key soil health indicators such as temperature, moisture, and nutrient levels (phosphorus, nitrogen, potassium, etc.) and allows users to view these conditions over time with a phone application. Users will receive alerts if the temperature, amount of moisture, and nutrients are too high or low and the user can provide what the plant needs (if it is more fertilizer, a shadier spot with cooler soil, or more water).

C. BACKGROUND

A wireless network of soil probes can provide gardens, nurseries, and small farms with real-time information on soil health and environmental conditions without the need for lab testing or personnel. These soil probes can provide insight into soil nutrient levels, soil temperature, and moisture. Real-time data on these parameters allows the user to adjust water consumption and nutrient application to better address the soil conditions and improve the health of the crops or plants in the area. Additionally, logging the evolving soil conditions will allow the user to forecast future conditions months or even years ahead. By allowing users to forecast and predict what their crops will need, crops will be healthier, water waste will be reduced, fertilizer runoff due to an excessive application will be reduced, and soil health will be improved. Additionally, users can see financial returns by applying less fertilizer and reducing costs for excessive irrigation.

There are many remote soil monitoring technologies that are being utilized in the agricultural sector, but most soil testing is done in laboratories. Lab tests evaluate both physical and chemical properties of the soil and usually require many samples to accurately describe a geographic location. Remote soil monitoring technologies are still an emerging concept for traditional farming applications and favor large-scale farming applications. The remote soil probes most similar to the team's design include the Soil Scout, HOBOnet Field Monitoring System, and the Teralytic Wireless NPK Probe.

The Soil Scout solution uses wireless all-in-one sensors permanently installed underground that communicate to a base station with patented low-power radio technology. The base station connects to a cellular network to upload sensor data to the cloud (Soil Scout server). The user can access the sensor data through a Soil Scout web application. The Steven's model also uses a cellular network to log data. Accessing soil data for most current solutions requires internet access to use the web applications where the data is displayed. Soil Scout uses a patented radio technology embedded in the soil sensors. This technology utilizes backscattering for low-power radio transmission. The Soil Scout has an option to use a solar panel, but it is for extending the wireless communication range rather than powering the Soil Scout.

The patented HOBOnet Field Monitoring Systems provide a weatherproofed data logging station that is capable of recording temperature, humidity, dew point, soil moisture, water level, and other weather-related parameters. Alarm notifications are available for user-set thresholds and a network of up to 50 probes is supported. HOBOnet uses a cloud bases software for data access and has an operating range of roughly 600 meters at 10ft off the ground. Solar panels charge nickel-metal hydride batteries to power each probe with a 3-5 year life expectancy or lithium batteries with a lifespan of 1 year. HOBOnet devices employ OQPSK (offset quadrature phase-shift keying) modulation and operate at 904-924 MHz. All sensors are removable, and each unit can support multiple sensors at any given time.

Teralytic Wireless NPK Probe is a patented weather-resistant soil probe with one or more sensor arrays. Each array may contain moisture sensors, conductivity sensors, temperature sensors, or several other sensors. These sensors give real-time data that is logged and can be used to approximate ammonium, calcium carbonate, chloride, nitrate, phosphate, potassium, sodium, or sulfate. A solid-state reference electrode is coupled to an ISFET sensor or shared via multiplexing circuits. The data that is recorded is wirelessly transmitted via communication hardware to a cloud. Analytics within the Teralytic software determine the soil conditions and report to the user in real-time using charts. The system has a 10-mile range and can be accessed from mobile devices.

All the devices like the DT07 solution are limited in functionality and hampered by price. The Soil Scout requires permanent installation into the soil substrate which introduces obstacles into the fields and requires more extensive construction when initially installing the system.

Additionally, this application is intended to target moisture content and not nutrient or temperature measurements. To properly protect the probes in the system, they must be buried below the topsoil till layer which typically lies at about 0-8 inches. Unfortunately, the topsoil layer is essential for early seedling growth and this system does not support measurements this shallow. The HOBOnet system is limited in range and in sensing capabilities. Each probe must be separately connected to the transmitter unit and the range of the units is limited to 600 meters. Additionally, the HOBOnet system does not provide soil nutrient sensor data for the user. The Teralytic Wireless sensor contains the widest sensor array options and provides the most data for all soil substrates, however, the sensor requires a cellular data plan for function and can cost upwards of $15,000 for a one-year lease of 10 probes. A single probe one-year lease will cost $1,200.

The DT07 soil monitoring solution is targeted for small-scale nurseries and home gardeners and will cost significantly less than the previously mentioned technologies. The device will be a solar-powered microcontroller physically connected to moisture, temperature, and nutrient sensors that collect data. The data will be read off the analog input pins as a voltage and passed through an A/D converter. The digital data can then be processed into relevant values (temperature, moisture level, nutrient level). The processed data will then be transmitted wirelessly to a cloud database. The user will be able to view the sensor data using an Android phone application. The application will be responsible for requesting the data from the database and displaying in a user-friendly way. The application will also allow the user to create and edit setpoints for the sensor data. Setpoints can be used to send notifications to the user if the sensor data falls below or shoots above the specified setpoint value.

Like existing solutions, the DT07 solution will utilize soil electrical conductivity probes and soil capacitance probes to measure and describe soil nutrient levels, temperature, and moisture levels. These probes utilize exposed stainless-steel electrodes that when energized, will provide real-time feedback on changes in soil conductivity and capacitance. The feedback will directly correlate to soil temperature and moisture levels as different soil conditions provide different soil conductivity and capacitance. The DT07 solution will utilize a photodiode color sensor array rather than Ion-Selective Electrode (ISE) technology or Ion Selective Field Effect Transistors (ISFET). ISEs are fragile and limited to a lab or short-term use and ISFETs are still an emerging technology with common ISFET probes being expensive. The color sensor approach will allow for a cheaper and more solution that can approximate soil nutrient conditions.

Current sensor solutions use long-lasting batteries instead of renewable energy. This is an issue because the sensors will eventually need to be dug up to replace the battery. Finding the sensors may be difficult since they are completely underground. The sensors of the DT07 solution are connected to the above-ground device which can be easily seen and readily portable. The DT07 Solution would have the microcontroller in a waterproof enclosure attached to a shaft and a solar panel. Solar power will provide power to the microcontroller and allow for efficient energy use. The DT07 Solution is self-contained and does not require many separate parts to be put into place for it to work. This feature makes the solution portable and differs from the Soil Scout because the Soil Scout sensors are not easy to find and relocate.

To maintain maximum efficiency the device will be powered by solar panels. When powering the system, the solar power will be run into a rechargeable battery and this battery will act as the main power source for all equipment. To ensure that the device lasts through the nights when there is no sunlight, a fully charged battery should last up to 16 hours without external power. Additionally, the battery life should be at least 6 months so the user will only on rare occasions have to change out the battery.

Since the DT07 is self-powered, power management will be critical to the design. Most power will be consumed when the device is transmitting data. To save power, it will not send data continuously but in time intervals. The microcontroller should switch between different power modes when a specified condition is detected. Such conditions could be a triggered setpoint or time-based. The device must also be able to continuously gather sensor data while in a power-saving mode to keep an accurate reading of the parameter in memory.

D. MARKETING REQUIREMENTS

Requirement Number:	**Marketing Requirements**
1	The deMETER Remote Soil Monitor must be portable and small enough for an average adult to easily carry.
2	The deMETER Remote Soil Monitor must have simple installation and minimal setup for the user.
3	The deMETER Remote Soil Monitor must allow for user defined settings.
4	The deMETER Remote Soil Monitor must sense temperature, moisture and nutrient levels in real time.
5	The deMETER Remote Soil Monitor must sense temperature, moisture and nutrient levels with reasonable accuracy.
6	The deMETER Remote Soil Monitor must transmit the acquired data wirelessly.
7	The deMETER Remote Soil Monitor app must provide a user interface for viewing data.
8	The deMETER Remote Soil Monitor must utilize internal batteries for power and recharge them via solar power.
9	The deMETER Remote Soil Monitor must function properly in a vast range of outdoor conditions.
10	The deMETER Remote Soil Monitor must be scalable from one to multiple sensor nodes

Table 1: Marketing Requirements

II. ENGINEERING ANALYSIS

Microcontroller (RR):

The microcontroller will be responsible for reading and processing sensor data and transmitting the sensor data wirelessly to a cloud database. This subsystem must be power efficient, have enough memory to store the required firmware, have enough analog input pins for the sensors, and have wireless communication capability. To meet these needs, an ESP32 microcontroller will be used for this application.

The most power will be consumed when the device is transmitting data. In this case, the ESP32 will be in active mode and will be draining 240mA of current at 3.3V. When the device is not transmitting data, it will be in deep-sleep mode. In this mode, it will drain $150\mu A$ at 3.3V.

Power Mode	Current consumption	Power consumption
Active (RF working)	240mA	792mW
Deep-sleep	0.150mA	0.15mW

Table 2: Power Modes and Consumption

Since the device only needs to be in active mode when it is transmitting data, it will spend most of the time in deep-sleep mode. The ULP coprocessor is designed to measure voltage while operating in deep-sleep mode. In this mode, it can continue to gather sensor data while limiting the power consumption.

The ESP32 uses a 32-bit instruction set and has 520KB of SRAM. So, the total memory required for the embedded firmware solution must be less than 520KB.

$$(Total\ instructions) * 4\ Bytes\ + (total\ data\ memory\ in\ Bytes) \leq 520{,}000\ Bytes.$$

Estimating the number of assembly instructions for each block of the level-1 flow (page #13) chart yields the following:

$$Total\ instructions = (total\ instructions)_{main} + (total\ instructions)_{interrupt}$$

Block	Estimated number of assembly instructions
1. Initialization	50
2. Read analog inputs	50
3. Process analog input	100
4. Replace last reading	10
Total:	210

Table 3: Main thread estimation

Block	Estimated number of assembly instructions
1. Wake up	10
2. Connect to database server	500
3. Insert records	300
4. Disconnect from database	200
2. Go to sleep	10
Total:	1020

Table 4: Interrupt thread estimation

Using 1200 Bytes as an over-estimation for how much data will be needed:

$$Estimated\ required\ memory = 210Bytes + 1020Bytes + 1200Bytes$$

$$= 2430Bytes$$

So, the estimated required memory fits well within the constraint of 520KB.

The ESP32 supports two sets of ADC channels: ADC1 and ADC2. ADC1 includes 8 channels on GPIO32 – GPIO29. ADC2 includes 10 channels on GPIO0, GPIO2, GPIO4, GPIO12-15, GPIO25 – GPIO27. However, some of the ADC2 pins are used as strapping pins and cannot be used freely. So, to avoid any collision, ADC1 will be used for this application.

Since ADC1 includes 10 channels, this will be more than enough analog input pins for the amount of sensors we will be using.

The ESP32 has an 802.11 b/g/n/e/i Wi-Fi module that operates at a 2.4 GHz frequency with speeds up to 150Mbits/s. Since the ESP32 covers a wide range of 802.11, it is easily compatible with many networks. The ESP32 can be set up to be an access point which means it can connect to any device with Wi-Fi without using a router. In this case, the ESP32 will be set up as a Wi-Fi station where it will need to use a router as an access point and transmit data to the cloud. The ESP32 has an antenna with a maximum transmission power of 20.5 dBm (decibel-milliwatts) which can be adjusted. If needed, the transmission range for an ESP32 can be extended to 10km (32,808.4 ft) with a directional antenna.

Structure (AF):

For the structure of the soil monitor, there is two main components. The first section will be the top, housing all power, battery, sensor control, and solar panels. This section is a 6" cylinder that is 5" tall. The middle has a 3D printed housing for all circuitry and provides just enough room for wires to run smoothly to the sensors and solar panels. This 3D printed housing uses 0.25 wall thickness and was found to be sturdy enough for the demonstration. The housing is shown below.

The second section of the structure will be a stake housing all sensors that is meant to be driven underground. This measures 30" by 2" with custom cutouts for each sensor. Again, using a wall thickness of 0.25", the structure was sturdy enough to withstand the demonstrations and all sensors fit as they should.

<u>**Power**</u> **(AF)**

There are three key components to consider when designing the power structure of the system. The first is providing the correct amount of power to each subsystem. The second is a circuit that will take the power provided by the solar panel and recharge the battery. Finally, the third section will be the circuit that contains the solar panel itself. The first section includes the microcontroller and the soil sensors. Each sensor, the RGB, temperature, and moisture sensor should each require a similar amount of power. To provide accurate power to each component, a simple circuit will be constructed with each load in parallel and resistors regulating the voltage. This will be calculated and implemented with Ohms law. The estimated power requirements can be found in the table below as well:

Component	**Estimated Power**
Microcontroller	500 mA at 3.3 V
RGB Sensor	30 mA at 3.3 V
Moisture Sensor	15 mA at 3.3 V
Temperature Sensor	15 mA at 3.3 V

Table 5: Estimated Power Consumption per Subsystems

Localized voltage regulators will be integrated into the sensor boards so that they can be tailored to the specific sensor they are with. This is intended to reduce variations in inputs to the sensors and avoid needing and additional regulator board for the entire system. The

XC6206P302MR-G is the selected voltage regulator to feed the capacitive sensor and Figure 1 illustrates it's a typical implementation circuit for the voltage regulator.

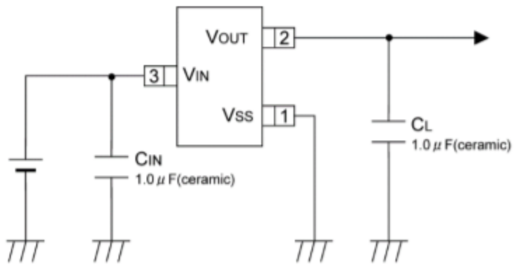


Fig. 1: Typical implementation of the selected voltage regulator

The main power source of the circuit will be a 3.7V, 4400 mAh lithium-ion battery. The exact battery chosen is produced by Vidar and is shown below. This battery was chosen because of its high mAh capacity, the ability to handle up to a 1.5A charging current and being readily available should it need to be replaced.

VIDAR
优能特
Huizhou Unity Technology Co., LTD
Tel: 0752-5800716
Website: www.hzunte.com
Add: 3F Building C,Linhai
Industrial Estate,Shuikou,
Huicheng District,Huizhou

Fig. 2: Vidar Battery

The charging circuit itself has two main goals. This first is to charge the battery accurately and safely without overcharging. The second is to allow for use of the system regardless of if the charging circuit is active or not. Seen in the schematic below labeled Figure 4, the circuit first has the LM317 directly at the input. The adjust pin runs to potentiometer P1 which controls the target voltage across the battery. This is done with simple voltage division with the 330Ω resistor. Next potentiometer P2 controls the max charging current. This potentiometer acts as the shunt resistor across the diode and the transistor. As the voltage across P2 approaches 0.95V, the transistor begins to open and allowing more current to “escape” and lowering the overall charging current. This will protect the battery from overcharge because once the target voltage is reached, the current should be nonexistent. Additionally Figure 4 shows how the output voltage remains constant as the input voltage increases

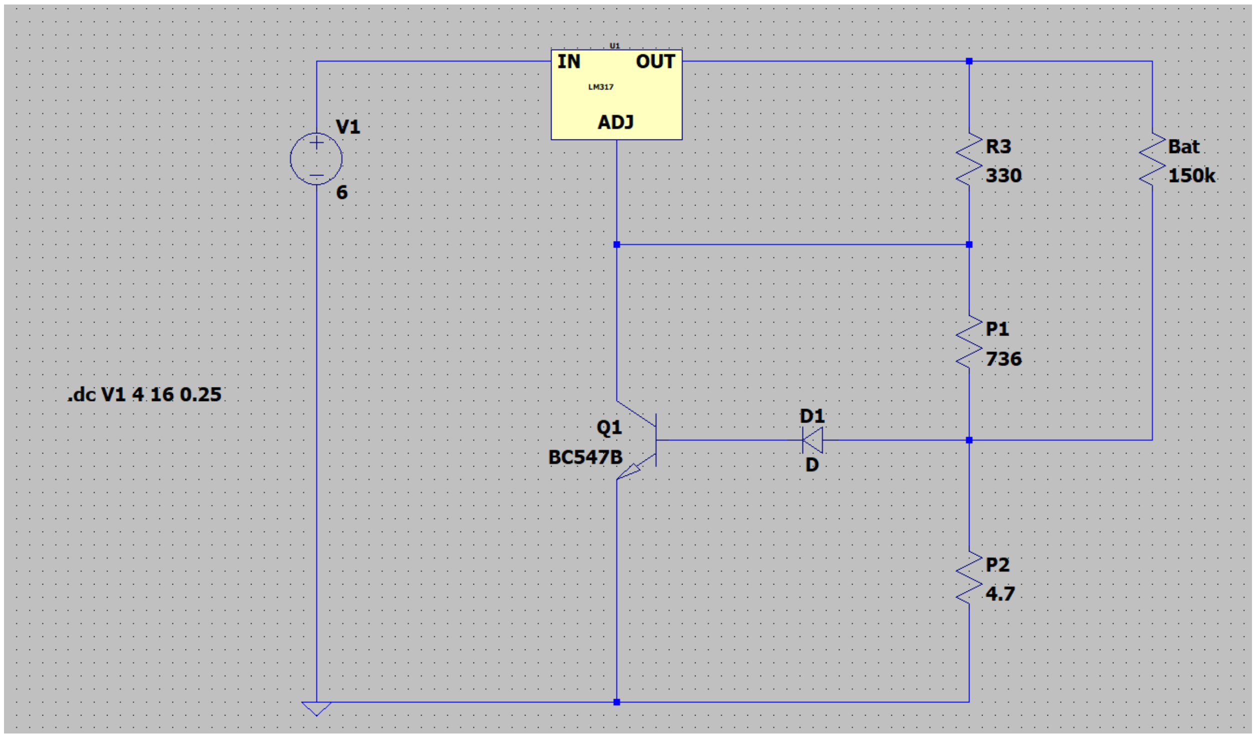


Fig. 3: Charging Circuit Schematic

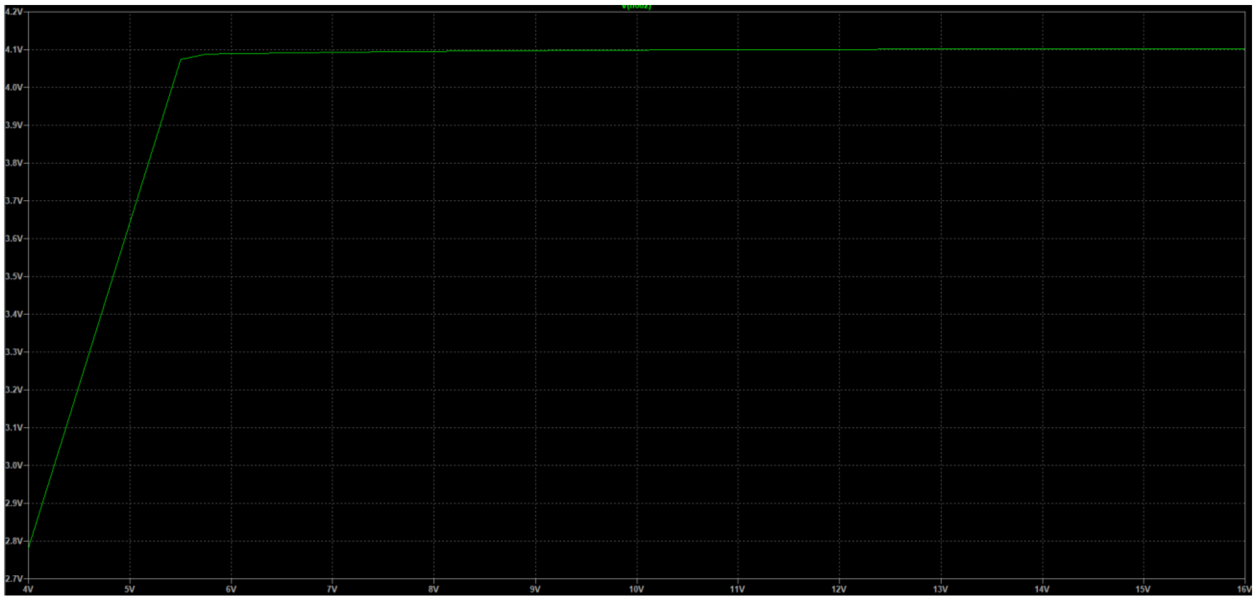


Fig. 4: Charging Circuit Output

This circuit will also allow for the rest of the system to be powered by the battery regardless of the state of the charging circuit. While not pictured in the schematic, the XC6206P302MR-G voltage regulator mentioned above will be placed directly after the battery before any of the other components to keep the voltage constant. Should any of the components require less voltage than is put out by the regulator, simple voltage division will be used to fine tune the required voltage. This was demonstrated in the subsystem presentations and proven to be an effective method of powering the sensors and microcontrollers. Below is a picture of the actual circuit built on a breadboard

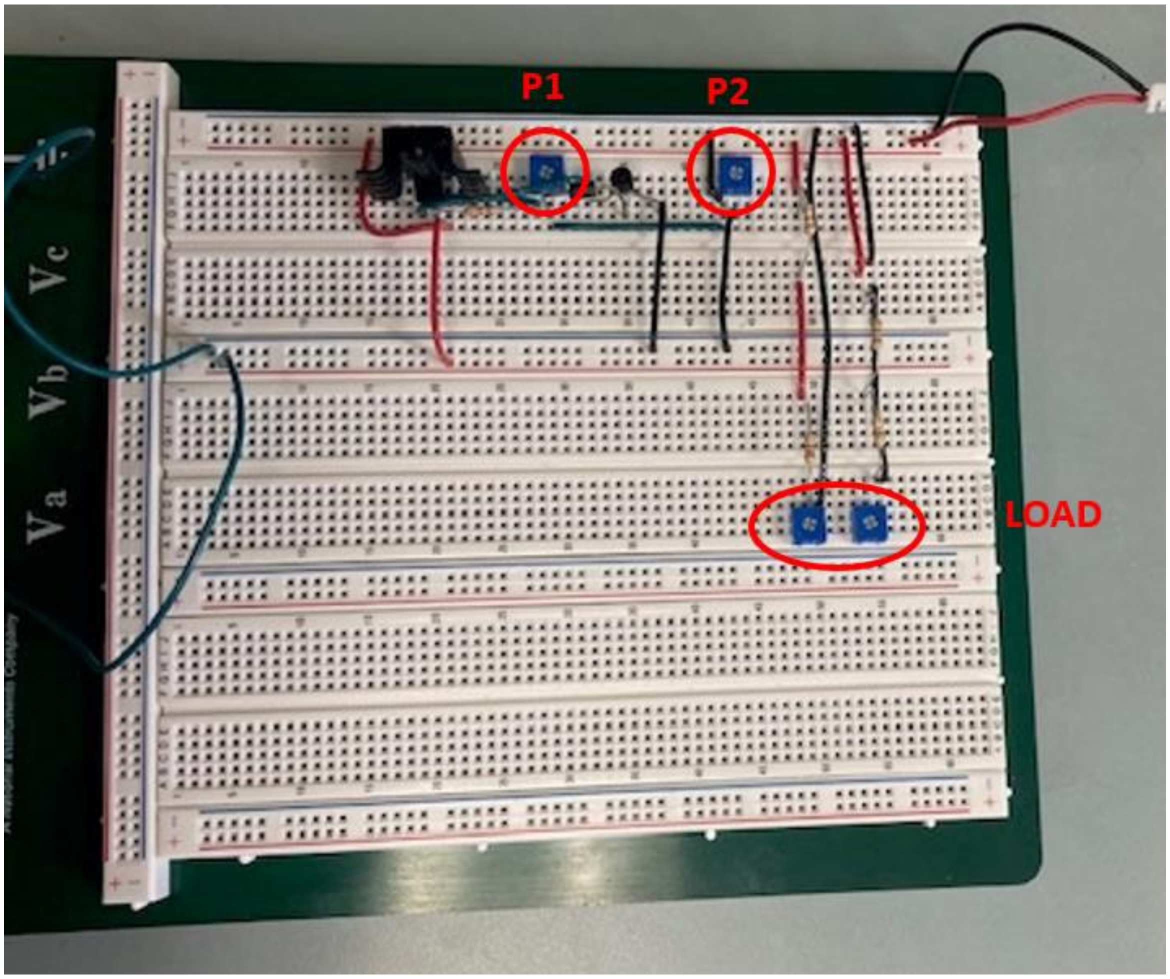


Fig. 5: Physical Charging Circuit

Above shown is the physical charging circuit that was built and demonstrated in the subsystem presentations. To the far left is the LM317, with a black heat sink to protect it from getting too hot. The input voltage is coming from a constant voltage source simulating the solar panel. The power rail and ground rail are below the circuit. Just to the right of the LM317 is potentiometer P1. Next to that is the 1N4007 Diode and the BC547 transistor. Finally, to finish the circuit, potentiometer P2 is positioned right after the output to the battery. The top rail leads to the battery and simulated loads. The red and black wire on the top right of Figure 6 lead to the battery. The input resistances of the moisture sensor circuit and the microcontroller were taken, and two potentiometers were used to create the same resistances. The voltages were then measured across those loads to ensure 3.3 V could be achieved while the circuit was charging and when the constant power supply was turned off. The next step after the breadboard was proven to work was to construct the circuit on a solder board.

During this the schematics shown above were used and carefully followed to create a compact solder board capable of doing everything the breadboard circuit could. This was done over several days and tested thoroughly once complete. Additionally, a ZXCL330E5TA 3.3 V voltage regulator was added at the output of the battery to ensure that as the battery discharges, it does not lower the voltage being distributed to each of the subsystems. This was a challenge as the voltage regulator only needed to drop the voltage from 4.1 volts to 3.3 volts. But this regulator met the requirements and worked perfectly. At the end of the solder board is a rail providing a protected and constant 3.3 volts that is then distributed to the rest of the subsystems.

The final step was to add the solar panel power source and mount it to the frame. Four 6 volt, 1 watt solar panels were used. These four solar panels were split into two groups of two panels in series. These two groups of series panels were then configured in parallel to provide a max charging voltage of 12 volts at 2 watts. This allowed for the circuit to charge even when the sun is getting low to ensure there is enough power for the system. This can be seen below mounted to the top of the housing.

Overall the circuit worked as expected. It provided enough power to the system, allowed for continuous use and charging, and protects against any unforeseen voltage surges or drops. The compact mounting system kept the circuitry in place and protected from the outside. The biggest challenges were learning about the solar panel circuit and applying what has been learned in classes over the past 5 years as well as protecting the subsystems from any voltage changes.

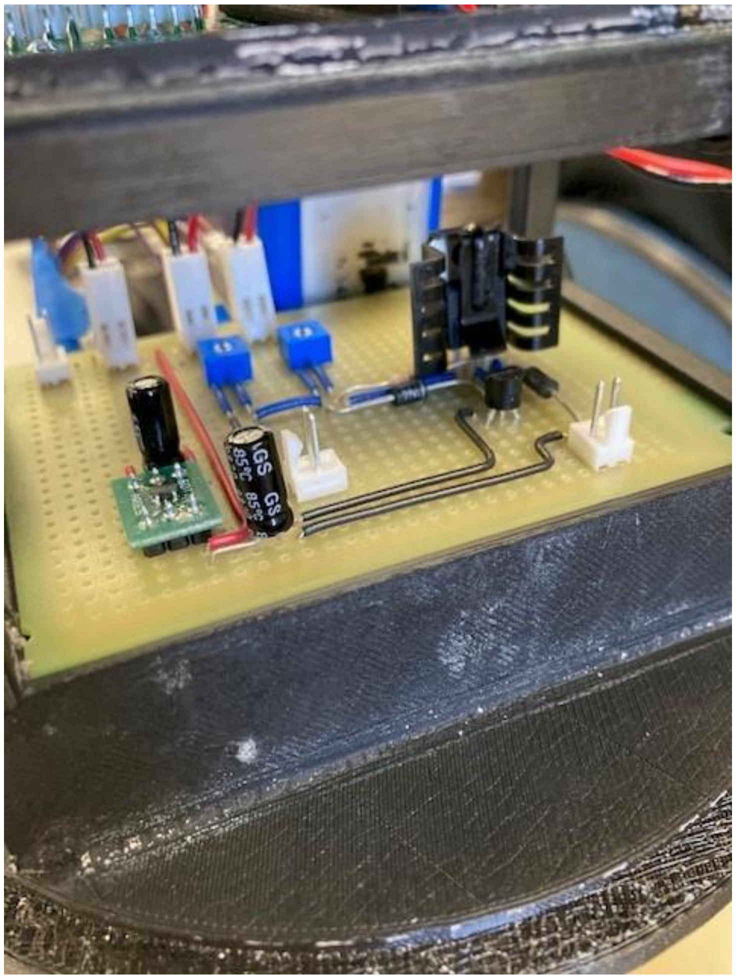


Fig 6: Implemented Charging Circuit

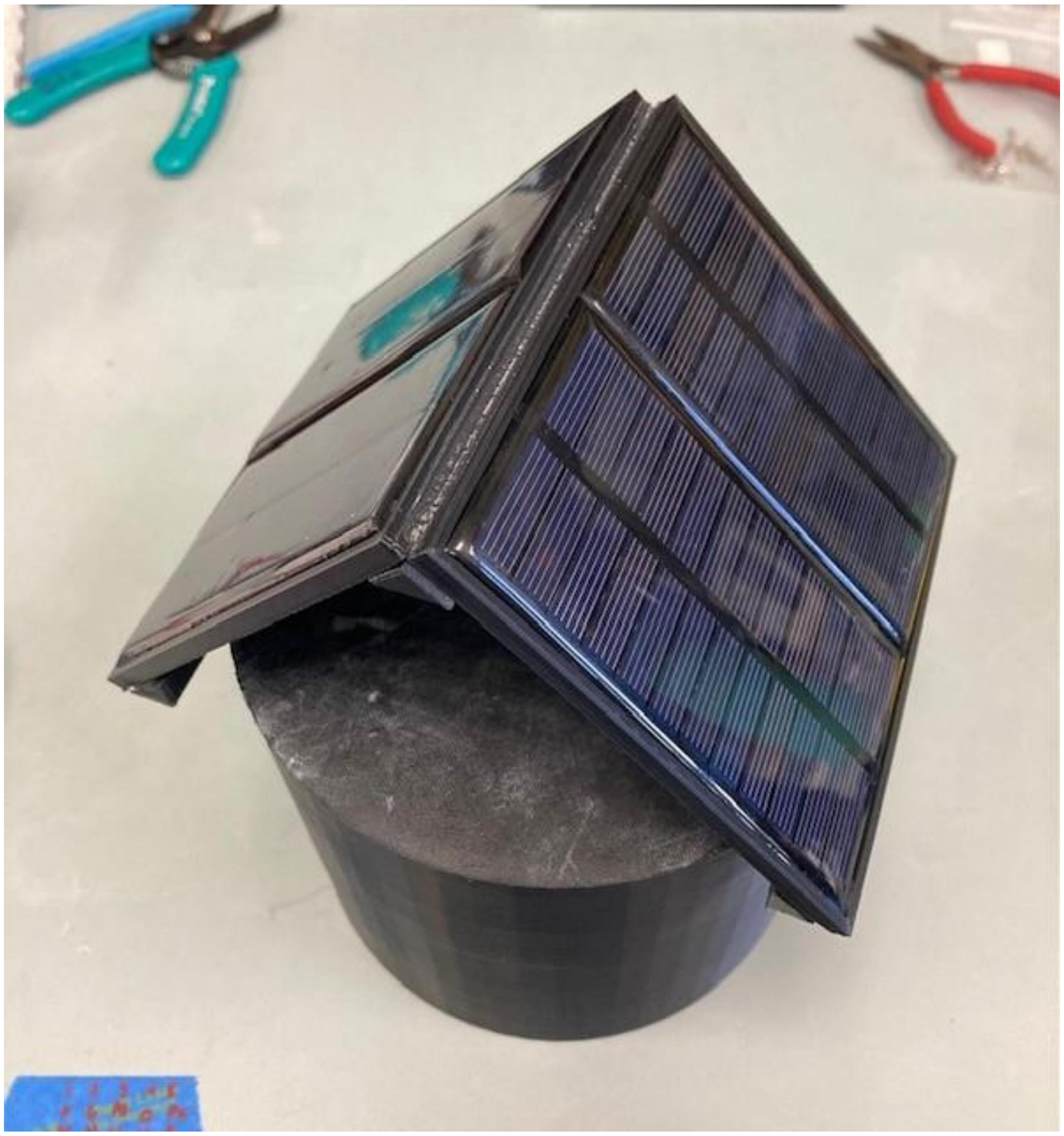


Fig 7: Mounted Solar Panels

Sensors (RM)

There will be four different sensors used on the deMETER soil monitoring probe all of which will be designed based on a 3.3V DC input in order to minimize the number of different voltages and to reduce overall complexity of the system. The first sensor is a soil moisture sensor, the second is a soil temperature sensor, the third is a PH sensor, and the last will be a nutrient sensor. These sensors will be mounted directly on the exterior of the probe body to make direct contact with the soil profile. It is imperative that each sensor be durable and able to survive prolonged exposure to dynamic conditions including the presence of abrasive substrate, acids, deposits of alkali and alkaline earth sulfates and carbonates, and water. The average growing season in the state of Ohio is approximately 170 days, thus the probe and sensors must be able to survive 170 continuous days of exposure to the soil profile.

The mounting locations of the sensors on the probe will be dependent on their function. The soil profile is broken into 4 horizons including the O (0-2”), A (2-10”), B (10-30”), and C (30-48”). For the purpose of moisture measurement, the lower b horizon of the soil profile will be targeted. This places the target depth at approximately 20-30” below the surface thus the sensor is planned to be mounted at 23” below the surface. The presence of nitrogen, phosphorous, potassium and other nutrients will be monitored at both the 7” depth and 15” by the color sensors to target the A and B horizons. The temperature sensor will target root depth temperature and will reside in the bottom half of the B Horizon; thus the sensor will be mounted at a depth of 23”. The PH sensor will be mounted at the same depth of the temperature sensor.

Moisture Sensor

The sensors will be designed in such a way that they utilize the natural relationships between temperature, moisture, chemical compositions, and electrical resistivity and conductivity. For the purpose of the soil moisture sensor design, the relationship between water presence and soil resistance and capacitance will be utilized. The resistance and capacitance of soil varies depending on its structure, texture, temperature, and the presence of moisture. The soil structure and texture are fixed, and temperature variations nearer the frost line are minimized. Therefore, the remaining variable is moisture which can be measured through a resistive or capacitive sensor. Equation 2 below shows the relationship between soil resistance and its resistivity changes with moisture content.

$$R = \rho * \frac{L}{A} \quad [\Omega] \tag{2}$$

R: The resistance seen between the electrodes of the sensor $[\Omega]$

A: The area of the face of the electrode that is in contact with the soil in square meters $[m^2]$

L: The distance between the two electrode faces in meters $[m]$

ρ : The soil resistivity in $[\Omega - m]$

Table 6 shows the varying resistivity values for three types of soil as moisture content changes. IEEE 142 provides more extensive tables of resistivity for common soil types and various conditions. These tables will be used to verify our sensitivity and accuracy of the soil moisture sensor as well as the temperature sensor. It can be clearly seen that under nominal conditions, as moisture increases, the soil resistivity will decrease.

MOISTURE CONTENT (%)	RESISTIVITY $[\Omega - m]$		
	TOP SOIL	SANDY LOAM	RED CLAY
2	***	1850	***
4	***	600	***
6	1350	380	***
8	900	280	***
10	600	220	***

12	360	170	1800
14	250	140	550
16	200	120	200
18	150	100	140
20	120	90	100
22	100	80	90
24	100	70	80

Table 6: *Electrical Engineering Portal* Soil Resistivity table based on IEEE 142

Additionally, Equation 3 demonstrates the relationship between soil capacitance and the moisture content. When two electrodes are applied to the soil, the soil and the electrodes create a capacitor that will increase in capacitance or decrease depending on the moisture. The capacitance of the soil is larger with the presence of water. Thus, as water content increases, the capacitance increase and the capacitive reactance decreases.

$$X_C = \frac{1}{2\pi f C} \qquad (3)$$

X_C: The capacitive reactance seen between the electrodes of the sensor $[\Omega]$

f: The frequency in hertz of the source signal applied to the electrodes [Hz]

C: The capacitance of the capacitor formed by the soil and the electrodes $[F]$

A resistive soil sensor is one way to measure this change in water content. It is a simplistic design that utilizes a single transistor and two resistors. As the soil resistance between the two electrodes decreases, a current can flow into the base of the transistor, the base current will increase, and a higher collector current results through amplification. Thus, the lower the resistance of the sensor electrodes, the higher the base current, the higher the collector current. Figure 8 shows the resistive sensor basic layout and premise.

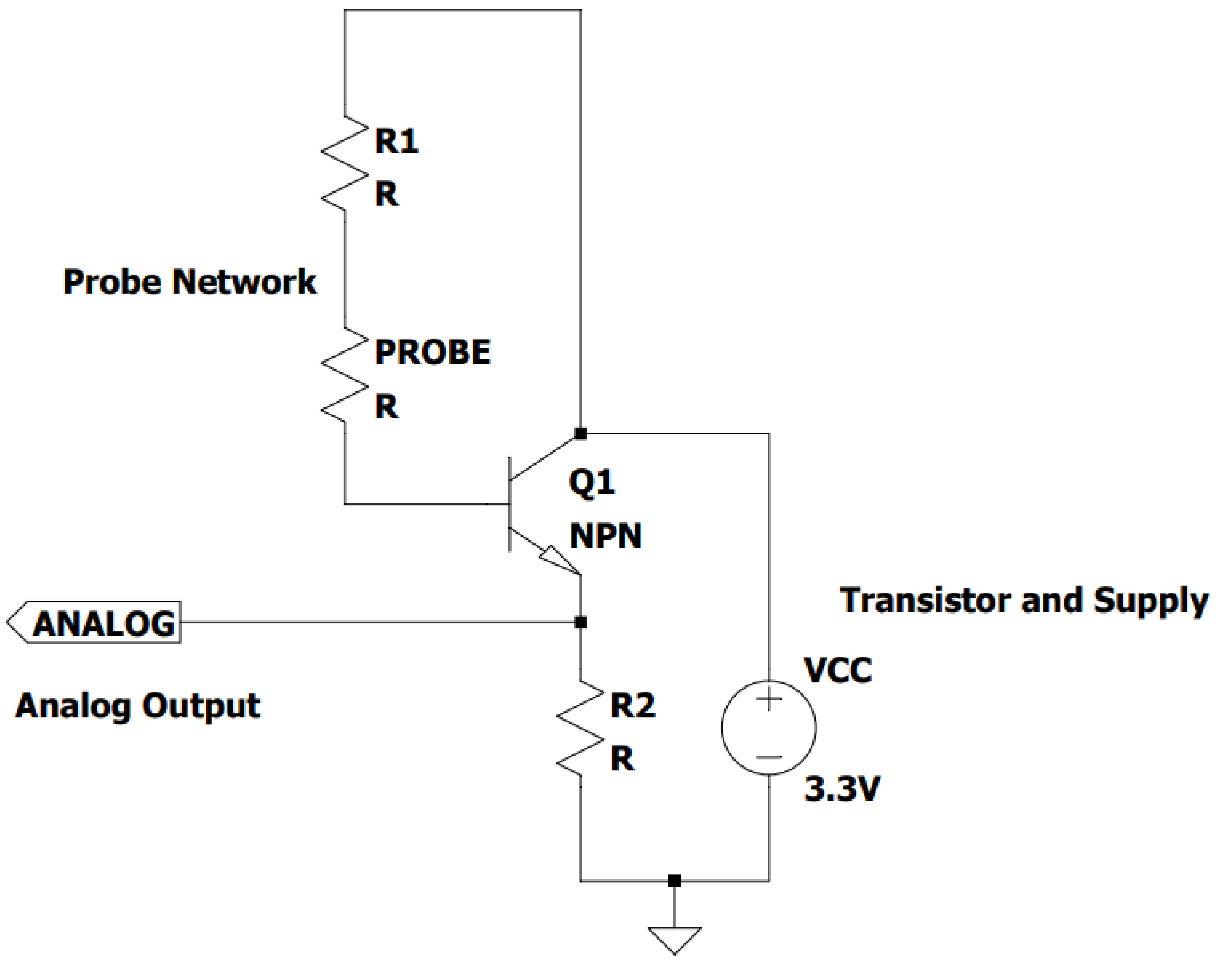


Fig. 8: Inferior resistive soil sensor design using a transistor

It was decided that the resistive sensor would not be utilized in the development of the deMETER soil probe because of significant durability issues. A direct current source causes a form of electrolysis on the exposed electrodes between the copper and the moisture in the soil. Without electroplating the electrodes in a gold coating, deterioration occurs rapidly within a few months of sustained soil exposure.

The capacitive sensor design does not have the electrolysis problem and for that reason, the capacitive sensor was chosen over the resistive sensor. The electrodes do not need to be exposed to form a capacitor in the soil, thus the sensor can remain in the soil indefinitely without corrosion.

In premise, a square wave input can be applied to two copper electrodes in the soil. A capacitor is formed between the two electrodes using the soil as the dielectric in between and as water increases the capacitance of the soil, the analog voltage output will decrease in magnitude. Conversely, as the capacitance of the soil decreases, the output voltage will increase. This relationship is governed by both equation (3) and (4). Equation (4) shows that the capacitance, which the sensor will be detecting, is equal to product of the permittivity of the dielectric and the area of the capacitor plates, divided by the distance between the plates.

$$C = \frac{\varepsilon A}{d} \quad [F] \tag{4}$$

C: Capacitance $[F]$

ε : The permittivity of the dielectric [F/m]

A: Area of the plate overlap $[m^2]$

d: Distance between plates $[m]$

The design of the capacitive sensor utilizes pulse generator, a low pass filter, and a half-wave rectifier. To generate the pulse, a TLC555 timer was placed in an astable multivibrator mode. The astable mode allows for the frequency and duty cycle of the pulse to be adjusted to the application. The TLC555 was selected over a regular NE555 because the maximum output frequency of the traditional NE555 was limited to around 500kHz while the TLC555 allows for up to 2MHz. Equation (5) was taken from the TLC555 specification sheet and shows how to calculate the output frequency based on the selected resistors.

$$f \approx \frac{1.44}{(R_a + 2R_b)(C)} \quad [Hz] \tag{5}$$

f : Output frequency of generated pulse $[Hz]$

R_a and R_b: Resistors $[\Omega]$

C: Capacitor for timer $[F]$

It is important to note that for frequencies higher than 100kHz, propagation delay times and on-state internal resistance introduce error. For this application, precise timing is not necessary but a high frequency output is desired. Thus, the delays and internal resistances were neglected and equation (5) was assumed to be true for all frequencies. Equation (6) and Equation (7) govern the time on and time off for the timer generated waveform and they were used to design the timer schematic for this application.

$$T_{on} = 0.693(R_a + R_b)(C) \quad [s] \tag{6}$$

$$T_{off} = 0.693(R_b)(C) \quad [s] \tag{7}$$

T_{on}: Time on [s]

T_{off}: Time off [s]

R_a and R_b: Resistors $[\Omega]$

C: Capacitor for timer $[F]$

Equation (8) was used to calculate the duty cycle based on the selected resistor values.

$$D = \frac{(R_a + R_b)}{(R_a + 2R_b)}(100) \tag{8}$$

D : Duty cycle [%]

R_a and R_b: Resistors $[\Omega]$

To pick the resistors and capacitor for this circuit, it was necessary to consider which combination gave the highest output frequency, which was closest to a perfect square wave (50% duty cycle), and which combination used the smallest and cheapest components. It was found that selected components below gave satisfactory results to all the considerations.

$$R_a = 330\Omega \qquad R_b = 1600\Omega \qquad C = 470pF$$

$$D = \frac{(R_a+R_b)}{(R_a+2R_b)}(100) = 54.67\%$$

$$f \approx \frac{1.44}{(R_a + 2R_b)(C)} = 867.9[kHz]$$

The low-pass filter serves as the connection point for the physical probe. The probe, comprised of two coated copper plates, creates a capacitor with the soil as the dielectric and this in conjunction with a resistor on the output pin of the TLC555 timer forms the filter. It serves to pass the low-frequency signal and attenuate the high frequency signal. This will pass a triangle output signal from the filter to the half wave rectifier circuit. For simplicity and availability, a 10kΩ resistor was selected to form the filter.

The half-wave rectifier portion of the sensor functions as a peak detector. The triangle wave from the low-pass filter is converted into a dc output signal using a diode, a resistor, and a capacitor. The diode and capacitor serve to produce the DC signal and the resistor was selected to reduce the ripple in the signal. To design the peak detector, a snapshot of the triangle wave output was captured from LTspice and used to calculate the RC time constant necessary to detect the next peak which came to be .0978. A common 1N4148 diode was selected along with a 100nF capacitor and a 1MΩ resistor. The combination of the 100nF capacitor and 1MΩ resistor made for a time constant of 0.1. Refer to Appendices Figure 1 for the detailed calculation of the RC time constant. The output of the peak detector is a DC signal that directly changes with soil capacitance. The completed design is shown in Figure 9.

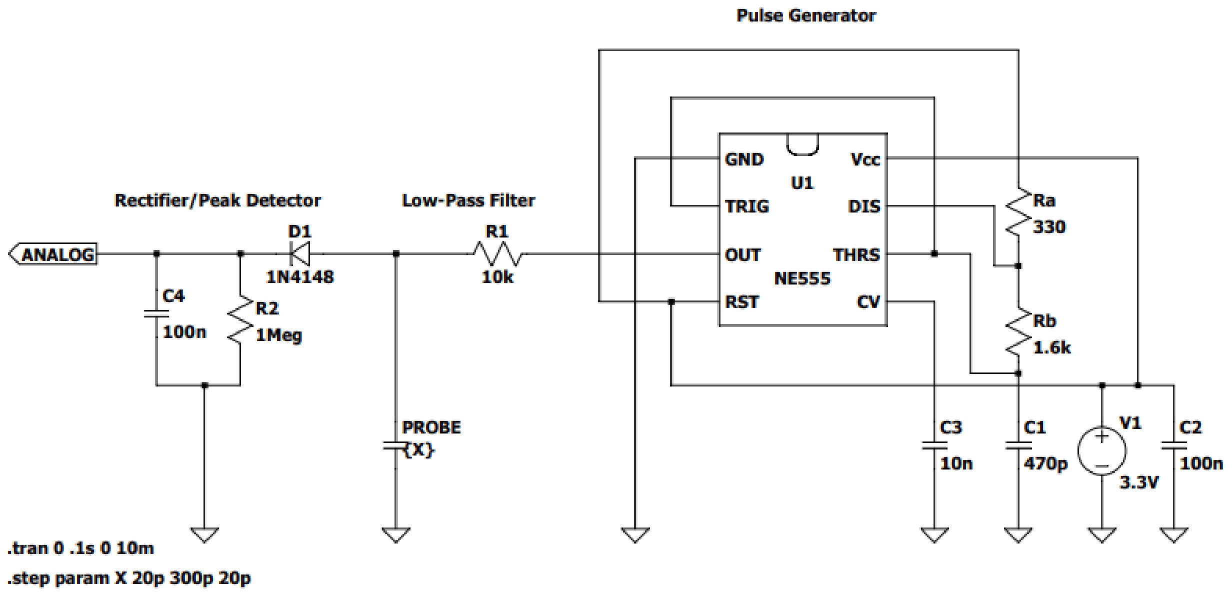


Fig. 9: Completed capacitive soil sensor LTspice model design

Appendices Figure 2 details the capacitive sensor being swept across multiple soil capacitance values and it shows a reduction in output voltage with each increase in soil capacitance. Even a .01V change in the output voltage can be detected but further lab testing will be required in order to correlate specific capacitances and tolerances with output voltages down to plus or minus 2.5%. Additionally, Appendices Figure 3 shows circuit breadboard implementation and Figure 4 shows the functioning multivibrator generating the desired pulses. Appendices Figure 5 shows the Analog Discovery scope view of the breadboarded capacitive soil moisture sensor as it was being tested in the lab with varying soils. It is important to note that in the final design of the deMETER soil monitor, printed circuit boards will be used for the designed circuits and not breadboards. Below Figures 10 and 11 show the completed schematics and the PCB layouts which were done for the final design. These were submitted to the PCB manufacturers for construction. The temperature sensor and the PH sensor design are also on the same schematic as the moisture sensor. Some of the resistors and biasing circuitry were broken into equivalent parallel or series values in order to minimize variation in parts. Some of the resistors and capacitors were changed from the original SPICE schematics due to part availability. No functionality was changed and the performance was equivalent.

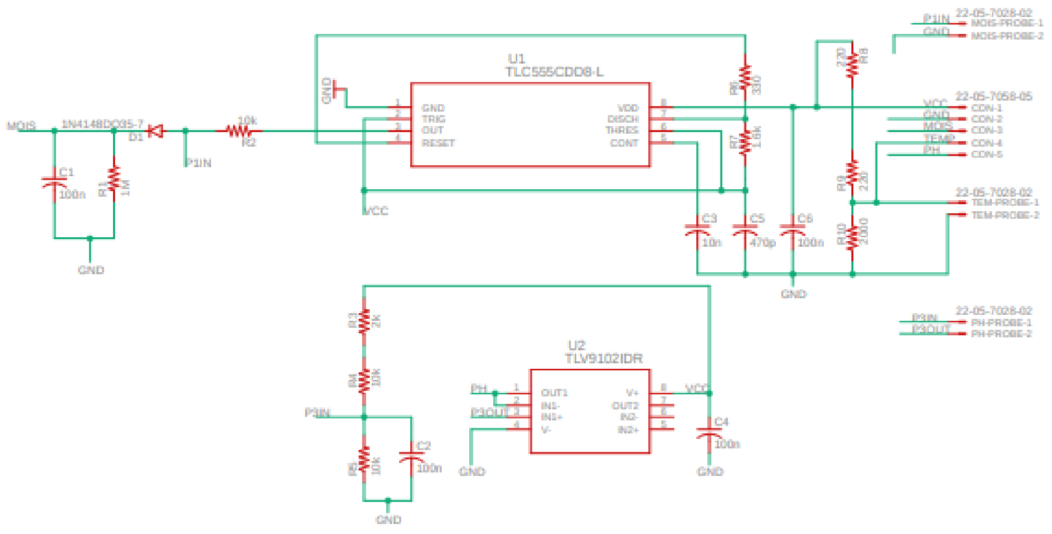


Fig. 10: Moisture Sensor Schematic with Additional Sensors

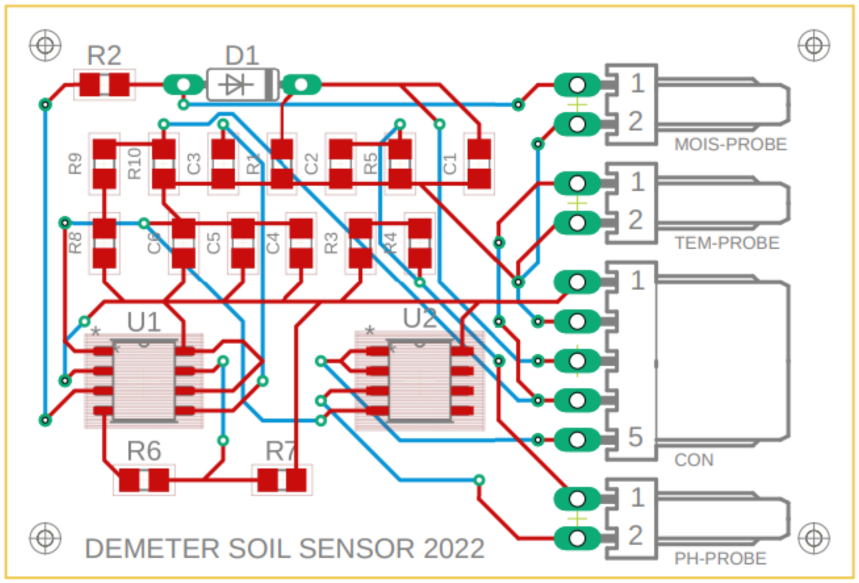


Fig. 11:PCB Layout of Sensor Board Optical Sensor

Temperature Sensor

The temperature sensor design is based on the *Steinhart and Hart* equation which relates resistances of semiconductors to varying temperatures. Equation (9) and (10) are the *Steinhart and Hart* equations used to calculate the resistances.

$$R_T = R_{ref}\left(e^{A+\frac{B}{T}+\frac{C}{T^2}+\frac{D}{T^3}}\right) \quad [\Omega] \qquad (9)$$

$$T_R = \left[A_1 + B_1 \ln\left(\frac{R}{R_{ref}}\right) + C_1\left(\ln\left(\frac{R}{R_{ref}}\right)\right)^2 + D_1\left(\ln\left(\frac{R}{R_{ref}}\right)\right)^3\right]^{-1} \quad [\mathrm{K}] \qquad (10)$$

T: Temperature [K]

T_R: Operating temperature as a function of resistance [K]

R_{ref}: Resistance value at a specified reference temperature [Ω]

R_T: Resistance value at an intermediate temperature [Ω]

$A, B, C, D, A_1, B_1, C_1, D_1$: Semiconductor material constants

The NTCLE100E3681 is a negative temperature coefficient thermistor which has increasing resistance with decreasing temperature and decreasing resistance with an increase in temperature. This specific thermistor was selected because it satisfied the design requirement that the sensor be accurate withing 5% of nominal value. The resistance tolerance of the NTCLE100E3681 is 5% with a B value tolerance of 1.5%. Additionally, this thermistor was readily available while other comparable transistors had extended lead times and limited availability. The Appendices includes the component specification sheets for the thermistor.

The transistor, along with an additional resistor in parallel and another resistor ins series, forms a voltage divider that is dependent on the temperature being sensed by the thermistor. As the resistance decreases with temperature, the parallel combination of the thermistor and resistor will decrease and thus a smaller output voltage. As temperature decreases, the divider will develop a much larger resistance and a larger voltage at the output will be seen. Equation (11) is the voltage divider formula that was used to construct the divider circuit.

$$V_{out} = V_{in}\left(\frac{R_2}{R_1 + R_2}\right) \quad [V] \tag{11}$$

V_{out}: Voltage across output resistance R_2 [V]

V_{in}: Input voltage [V]

R_1: First resistor in series $[\Omega]$

R_2: Second resistor in series or the output resistor $[\Omega]$

Figure 12 is the LTspice completed temperature sensor design which was simulated and is awaiting lab testing in the coming semester. The R_1 and R_2 components were selected based on the manufacturer provided typical application circuit.

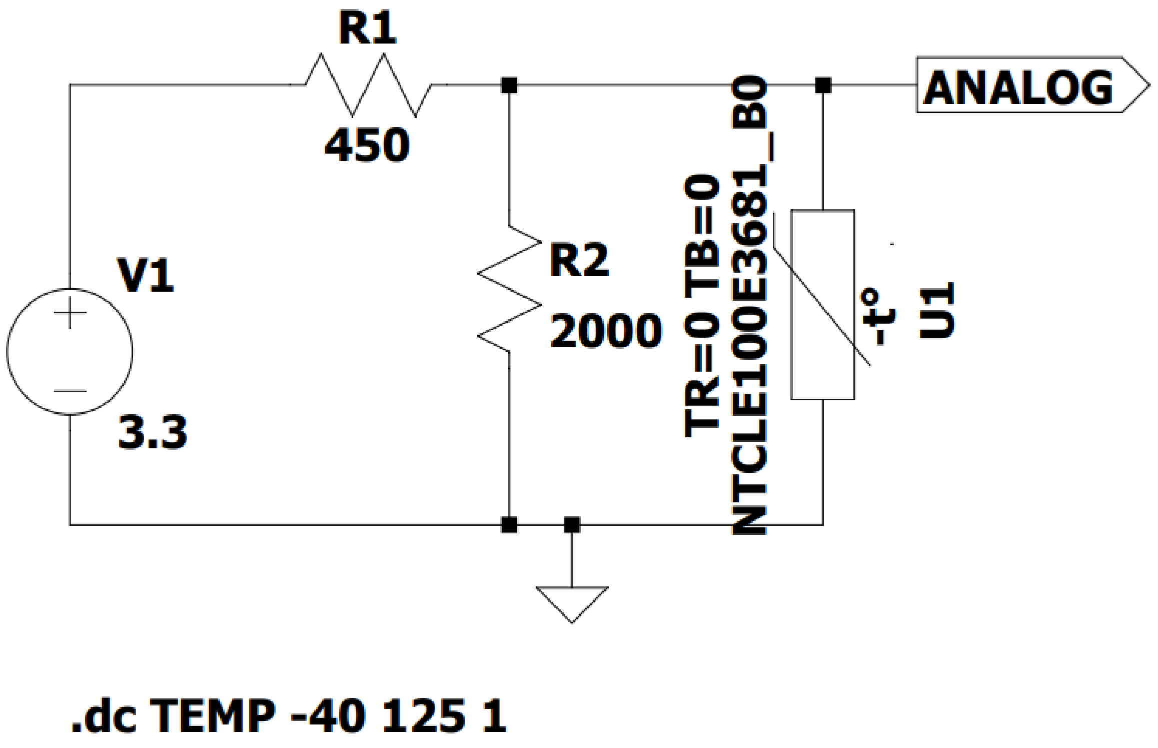


Fig. 12: Completed NTC thermistor-based soil temperature sensor LTspice model

The LTspice simulation was swept across the temperature range of -40°C to 125°C with output voltages plotted for analysis. The plot of this simulation can be found in Appendices Figure 6.

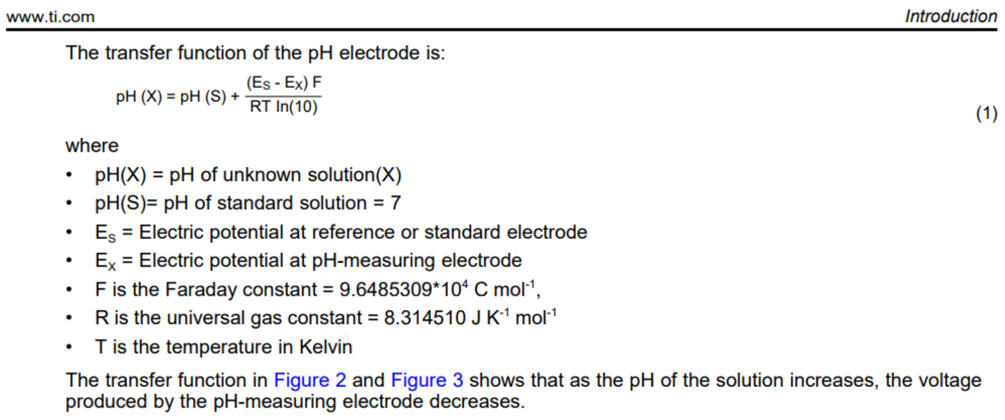
Calibration of the sensor was accomplished on the processing end of the returned data. Based on the output voltage of the circuit, room temperature was determined to be 76°F in the laboratory room with a measured temperature on the circuit of 74°F. This was found to be within 3% accurate and held constant at within 3% accuracy up to 100°F and down to 50°F. Testing was done using a heat gun and ice to simulate dynamic weather conditions. A regular thermometer from a hardware store was used. For more accurate readings, a digital or scientific thermometer should be used but the team did not have access to the more precise instruments. No calibration or compensation was added to the implemented circuit as a result of the accuracy achieved. The sensor was first tuned while implemented on a breadboard and Analog Discovery II unit and then verified on the converted printed circuit board for implementation in the final probe assembly. This sensor did not need to have direct contact with the soil and instead indirectly senses temperature through a plastic membrane. The membrane was on the outside of the probe and exposed to the soil with the thermistor attached to probe and protected by the membrane.

Figure 10 as shown above illustrates the voltage divider for the temperature sensor using resistor R8, R9, R10, and a temperature probe connection which goes out to the NTC thermistor. Figure 11 further shows the PCB layout of the board.

PH Sensor

The PH sensor was a late addition in the design process. The PH probe was originally excluded during the design phase due to the high cost of obtaining an accurate and durable screen-printed electrode. A cheaper but less accurate carbon electrode was sourced through a third-party sensor manufacturer on Amazon which was not durable and limited in accuracy, but able to sense PH without costing hundreds of dollars.

The PH sensor was initially designed using a TLV9102 op-amp without a biasing circuit. The TLV9102 was unable to be acquired due to parts backorders and shortages as a result of the pandemic. A redesigned PH sensor was accomplished using the LMP7721 precision op-amp from Texas Instruments and their recommended PH sensor design. The team's design differs from the TI design in that it uses the onboard regulator from the power subsystem to provide a reliable 3.3V source to the circuit and the biasing of the circuit was set to 1.1V rather than 1.024V. A biasing circuit is necessary as a PH reading will return a positive voltage when more acidic and a negative voltage when more basic. PH levels vary by 59.16mV from 1 to 14 and can swing up to 518.29mV in range at temperature extremes. By biasing the circuit at 1.1V, a negative voltage output can be avoided and the ESP32 can correctly transmit that data. Figure 13 indicates the transfer function used in the design of the circuit to return a PH reading based on the output voltage and reference voltage. The transfer function was taken from the Texas Instruments information page on the LMP7721 applications. The design the team used did not utilize temperature compensation, but this could be incorporated in a further iteration of the design. As a result, the transfer function was simplified further without the use of the temperature or Faraday constant as they were assumed to be non-factors. This assumption led to less accurate results but allowed for simpler implementation. Additionally, the probe is already limited in accuracy and ability so temperature compensation would not provide substantial benefit in a limited design.



www.ti.com Introduction

The transfer function of the pH electrode is:

$$pH(X) = pH(S) + \frac{(E_S - E_X)F}{RT\ln(10)} \quad (1)$$

where

- pH(X) = pH of unknown solution(X)
- pH(S)= pH of standard solution = 7
- E_S = Electric potential at reference or standard electrode
- E_X = Electric potential at pH-measuring electrode
- F is the Faraday constant = $9.6485309*10^4$ C mol^{-1},
- R is the universal gas constant = 8.314510 J K^{-1} mol^{-1}
- T is the temperature in Kelvin

The transfer function in Figure 2 and Figure 3 shows that as the pH of the solution increases, the voltage produced by the pH-measuring electrode decreases.

Fig. 13: TI data sheet showing PH transfer function used to code the PH sensor data return

The LTspice simulated circuit is shown in Figure 14 and the EAGLE CAD final design schematic is shown in Figure 15. Figure 16 shows the EAGLE CAD PCB Layout which was modified to use a protoboard instead of a full printed PCB due to component lead times and availability.

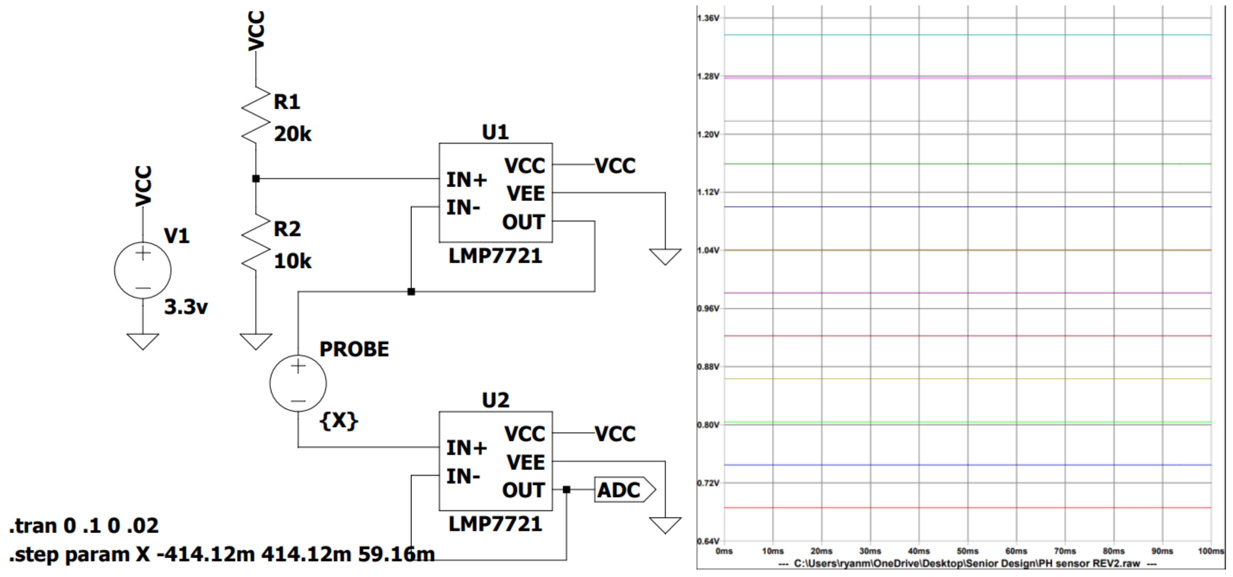


Fig. 14: LTspice Simulation and output of PH sensor circuit

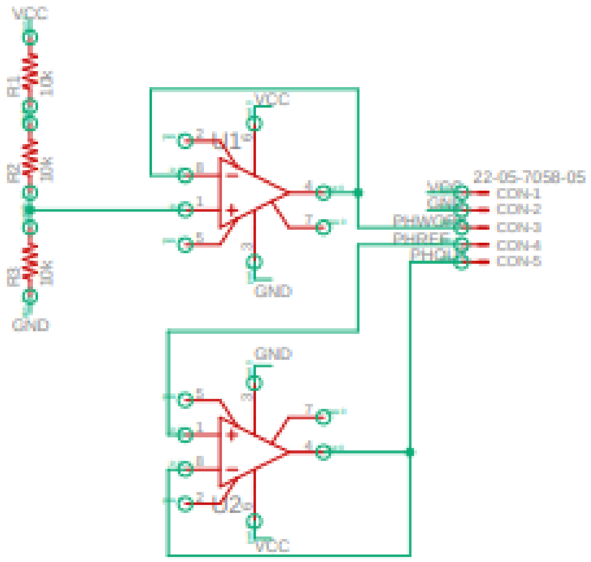


Fig. 15: Final schematic of the PH sensor

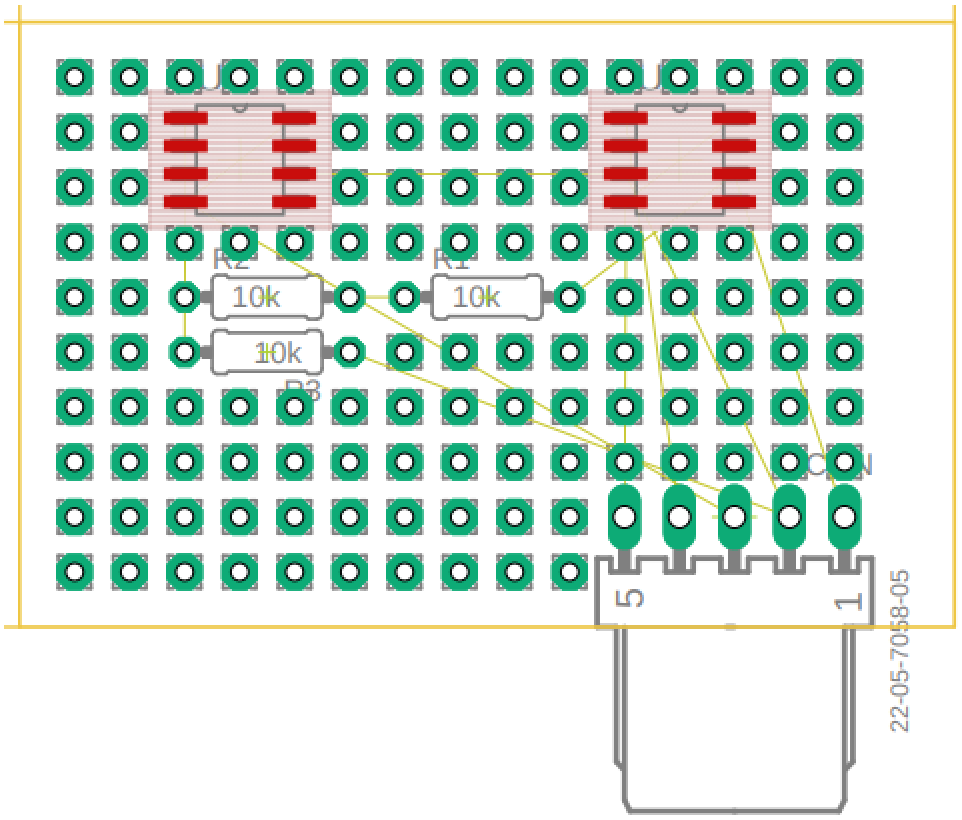


Fig. 16: Final schematic of the PH sensor

The PH sensor was verified in the laboratory using PH adjusted water to simulate different PH environments. Solutions of 9.18, 6.86, and 4.01 were used to test acidic, basic, and neutral solutions. Testing results varied wildly due to the imperfections and inaccuracy in the probe. Measurement took up to 15 minutes to settle into a constant value and the sensor circuit was overly sensitive and bounced wildly when the probe was disturbed. However, after the 15-minute window, 1723mV was returned for the solution of 9.18. The biasing point was found to be 1562mV which was higher than the 1.1V designed biasing point but it was due to component tolerances. Appendix Figure 17 shows the calculations of the 9.18, 6.86 and 4.01 PH results and how accurate they were. It was found that the accuracy of the probe under stable conditions was plus or minus 1 PH.

Nutrient Sensor

Early in the design of the deMETER soil monitor, a designated nitrogen, phosphorous, and potassium (NPK) sensor was desired. However, upon further research and design consideration, the use of ion selective electrodes and ion selective field effect transistors (ISFETs) are neither practical nor affordable. Ion selective electrodes utilize the principle of diffusion where the diffusion from high concentration to lower concentration results in a free energy change. The implementation of ISFETs is a new and emerging technology and relies on the ISFET principle that as concentrations of a selected ion increases or decreases, the current through the transistor will change. Both ISFETs and ISEs can cost between $400 and $1200 and require dilute solutions and concentrations in laboratory settings for accurate results.

The deMETER soil monitor takes a very different approach to soil nutrient sensing. The presence of soil nutrients can alter the coloration of soil media and color has a strong correlation to the presence or lack of nutrients in the soil profile. A Munsell soil color chart can be used to correlate soil characteristics to hundreds of different soil colors. By knowing the approximate color of the soil, nutrient presence, drainage, soil structure, and soil degradation to crops can be determined. To obtain the color of the soil, the deMETER monitor will use a color sensing probe to capture the RGB value of the topsoil and subsoil. This RGB value can then be correlated to a Munsell soil color value. That value will be displayed in the cellphone app along with a nutrient breakdown of that soil sample. Optical sensing requires numerous photodiodes in order to accurately calculate the RGB value of the soil in question. The TCS3200 color light-to-frequency converter made by AMS was selected to accomplish the optical sensing needs. The pre-printed circuit board package that contains 64 photodiodes in an array with a square wave output of 50% duty cycle being directly proportional to the irradiance of the soil. The package output frequency is then converted to an RGB value in the coding. Figure 17, which is from ACOPTEX, shows the schematic of the converter which will be directly connected to the ESP32. This design is direct from the manufacturer as it allows for scaling pins to adjust the frequency input and output and count the pulses.

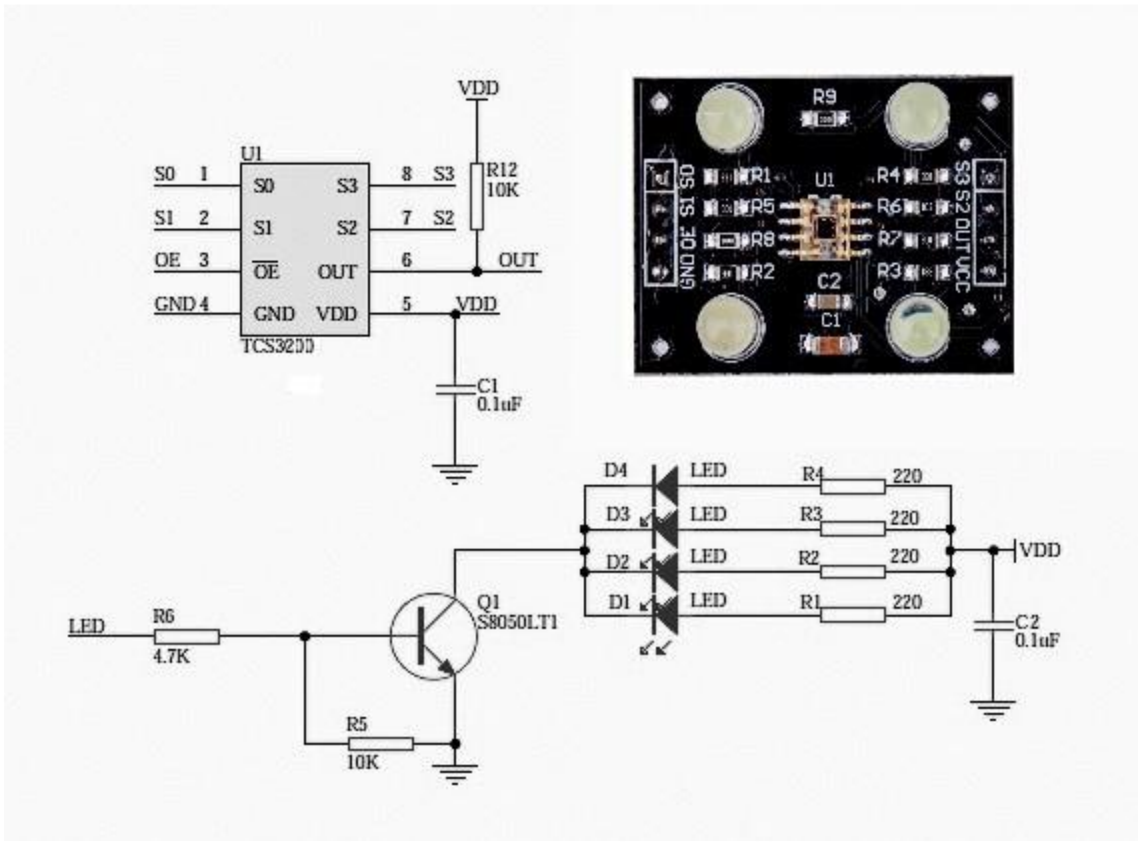


Fig. 17: Color light-to-frequency converter schematic

The photodiodes array will be able to view the soil profile from viewports installed on the side of the deMETER soil probe. The white emitting LEDs will be able to illuminate the substrate while still protected behind the clear window and the array will sense the reflected colors while remaining inside the probe body.

The TCS3200 will need calibration on each Munsell color before implementation on the probe. Four total colors were calibrated for the design this semester each with it's own distinct RGB and characteristics as shown in Table 7. While the optical sensing method does not give exact nutrient levels, it can approximate nutrient presences without the need for laboratory testing as was required by the design stipulations of the project.

RGB Values	Munsell Color Designator	Soil Descriptors
87 70 58	A7.5 YR 3/2	Sufficient Organic Matter, Presence of Goethite, Iron Sulfide, NPK balance Sufficient
159 112 65	A7.5 YR 5/6	Elevated Iron content, sufficient drainage, Phosphorous, sulfur, zinc deficient
198 169 127	A10 YR 7/4	Elevated Gypsum, presence of Jarosite, soil structure impeding drainage
149 115 85	A7.5 YR 5/4	Adequate drainage, nitrogen deficient, insufficient organic matter content

Table 7: Soil colors used in calibration of sensor with descriptors from NRCS and NAPA Master Gardeners

The optical sensor was tested and found to correctly identify each of the 4 calibrated colors when exposed to paint samples from Sherwin Williams with similar RGB values. Tolerances were built into the coding of the sensor such that colors with similar RGB values may still be sensed however will be displayed as one of the calibrated colors that is most similar.

Communications (TF):

The actual implementation ended up using a locally hosted MySQL database on a laptop, where the schema only included the sensor data. Connecting the mobile application to the database required the use of a WAMP software stack, which stands for Windows, Apache, MySQL and PHP. Since HTTP was used to get and send data, MQTT did not end up being used for the actual implementation.

For the communication side, the ESP32 sent the sensor data to the database and the mobile app would retrieve the sensor data. When the ESP32 sent the data to the database, it created an HTTP POST request to the server and once that request was okayed, the data was posted to the respective database. On the other hand, the mobile app had to use a REST API (Representational State Transfer Application Programming Interface) to send an HTTP GET request.

An API allows two software applications to talk to each other. Usually, an application connects to the Internet and then send data to a server. In this case, the mobile application was communicating with the WAMP server with an HTTP GET request for the sensor data contained in the database. REST APIs are generally preferred because it uses less bandwidth, high loads of clients can be managed with help from the HTTP proxy server and cache and allows for flexible formats by serializing data in XML or JSON format. A REST API assumes a direct point-to-point communication between the server and the client.

Ideally, if the project were to be scaled to more than one monitor, it might be worthwhile to consider MQTT (MQ telemetry transport). MQTT will allow the device to save even more power in the case of packet loss and retransmissions. Using MQTT avoids some of the overhead seen in HTTP and allows the use of less bandwidth, less latency and higher throughput. Why MQTT is a better choice than HTTP is because the binary data must be base64_encoded and that would require more resources from the network and the CPU. MQTT is run on top of TCP/IP and is meant more for machine-to-machine(M2M) applications, such as a Fitbit sending data to a smartphone. Since only sensor data would be sent between the device and mobile application, it is ideal to use MQTT as the communication protocol. The connection between the cloud and the microcontroller would be one way, where the microcontroller sends the data, and the cloud would confirm that it received the data. On the other hand, the connection between the mobile application and the cloud would be bidirectional to send data from the cloud to the app and store user credentials and device identification in the cloud. MQTT doesn't rely on the address of devices to determine where to send data, but instead uses a publish/subscribe architecture. MQTT has a broker that manages receiving information and transmitting the information to whichever client needs it. In this case, the broker would be the cloud database and the clients would be the soil monitoring device and the Android application. This protocol allows for a many-to-one architecture in which users can have multiple devices and still receive the information from each corresponding device in the mobile application. MQTT has three different levels of Quality of Service (QoS). Quality of Service determines how the messages are delivered and the reliability of the delivery. MQTT QoS simplifies communication in unreliable networks as the protocol is responsible for handling retransmissions and guarantees the delivery of a message regardless of the reliability of the underlying transport layer.

To calculate approximately how much data being sent between the ESP32 and the cloud with the MQTT protocol, the size of the packets getting sent and which packets are getting sent must be considered. In this case, there is a connect packet getting sent from the ESP32 to the cloud and the cloud sends an acknowledgement packet back. Then the ESP32 will send three packets containing sensor data. The first packet will have the temperature data, the second will have the moisture data and the third will have the nutrient data. In this case, it is assumed that the highest Quality of Service will be used which is QoS 2 in MQTT. It guarantees that each message is received exactly once, but since it requires three packets (PUBREC, PUBREL and PUBCOMP) for each message sent it is the slowest QoS and most reliable. Since the goal is to reduce the amount of needing to retransmit to further save power, it would benefit to use this.

Microcontroller (Client 1)		**Cloud Database (Broker)**	
Packet	**Size**	**Packet**	**Size**
Send Connect to Broker	Max 131,111 bytes	Send CONNACK to Client 1	4 bytes
Send Sensor Data	Max 65,543 bytes for a single packet	Send PUBREC to Client 1	4 bytes
Send PUBREL to broker in response to PUBREC	4 bytes	Send PUBCOMP in response to PUBREL from Client 1	4 bytes

Table 8: Packet Sizes sent between Microcontroller and Cloud

The max data rate between the ESP32 and the cloud will be 150 Mbits/s on 802.11n and the max amount of data being sent between the two is 327,780 bytes which becomes 2.62224 Mbits and that would be a rate of about 0.017 seconds or 17ms.

150Mbits1 s÷2.62224 Mbits=0.017 s150Mbits1 s÷2.62224 Mbits=0.017 s

For sending the sensor data, the max package size won't be needed since most of the bytes are dedicated to the topic name and the message name, which allow UTF-8 encoded strings that can have any length in the range 0 – 65535 bytes. So, the amount of data will be lower than the estimate calculated here and as a result the amount of time will be brought even lower.

Another feature that can be leveraged is the ESP32's A-MPDU (MAC Protocol Data Unit) and A-MSDU (MAC Service Data Unit) aggregation with a 0.4 microsecond guard interval support feature which allows packets to be sent as one single transmission unit. In this case, instead of sending all three packets individually, it would help to use this and cut down time on transmissions, but since the amount of data being sent is already very small, it may not make that much of a difference to use this feature. It would help with reducing the amount of acknowledgement packets since without aggregation it would be 9 packets (36 bytes/288 bits) and with aggregation it would cut it down to only 3 packets (12 bytes/96 bits).

Compared to the calculations before, if aggregation is used, 327,756 bytes which is 2.622048 Mbits and the time would be

$$150 \text{ Mbits} 1 \text{ s} \div 2.622048 \text{ Mbits} = 0.017 \text{ s}$$

which is still the same time as above, so it really makes no difference to use aggregation or not.

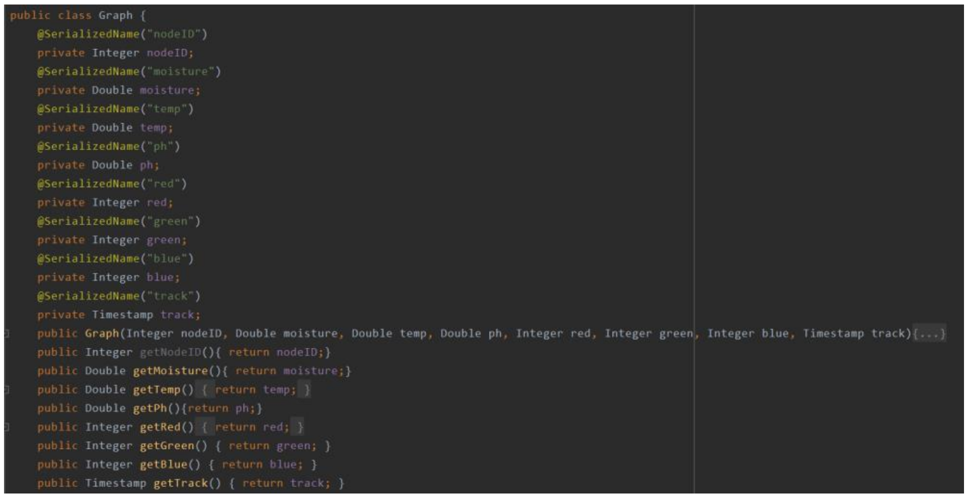
Potentially to communicate between an ESP-32 and an AWS Amplify cloud database, the AWS IoT Core can be used to ensure that MQTT is the mode of transportation. The AWS IoT Core ensures the data from the ESP-32 is sent to the cloud via MQTT and on the side of the app, the AWS IoT Device SDK for Java will be used (accesses MQTT over the WebSocket protocol) to display the data from the ESP-32 on the app. In this case, the AWS Amplify database is the broker that facilitates receiving and sending information. One perk of the AWS IoT Device SDK for Java is that it's built with shadow support which can be accessed using HTTP methods such as GET, UPDATE, and DELETE.

Since the ESP32 and the mobile app could only obtain/post data from the data base if they were on the same Wi-Fi network as the locally hosted database, MQTT and AWS Amplify may be the way to go if the project is scaled to more than one soil monitor.

Mobile Application (TF):

The mobile app will be on Android and developed with Android Studio and Java.

As mentioned in the previous section, the mobile application made use of a REST API so that the app could talk to the server when getting sensor data. To implement the REST API for the app required the use of an Android Studio library called Retrofit which is a type-safe REST client developed by Square. This library downloads JSON data from a web API and then it is parsed into a POJO (Plain Old Java Object) which is used on the Android mobile app side to graph the data in this case.



```java
public class Graph {
    @SerializedName("nodeID")
    private Integer nodeID;
    @SerializedName("moisture")
    private Double moisture;
    @SerializedName("temp")
    private Double temp;
    @SerializedName("ph")
    private Double ph;
    @SerializedName("red")
    private Integer red;
    @SerializedName("green")
    private Integer green;
    @SerializedName("blue")
    private Integer blue;
    @SerializedName("track")
    private Timestamp track;
    public Graph(Integer nodeID, Double moisture, Double temp, Double ph, Integer red, Integer green, Integer blue, Timestamp track){...}
    public Integer getNodeID(){ return nodeID;}
    public Double getMoisture(){ return moisture;}
    public Double getTemp() { return temp; }
    public Double getPh(){return ph;}
    public Integer getRed() { return red; }
    public Integer getGreen() { return green; }
    public Integer getBlue() { return blue; }
    public Timestamp getTrack() { return track; }
```

Figure 18: POJO (Plain Old Java Object) Android Studio Code

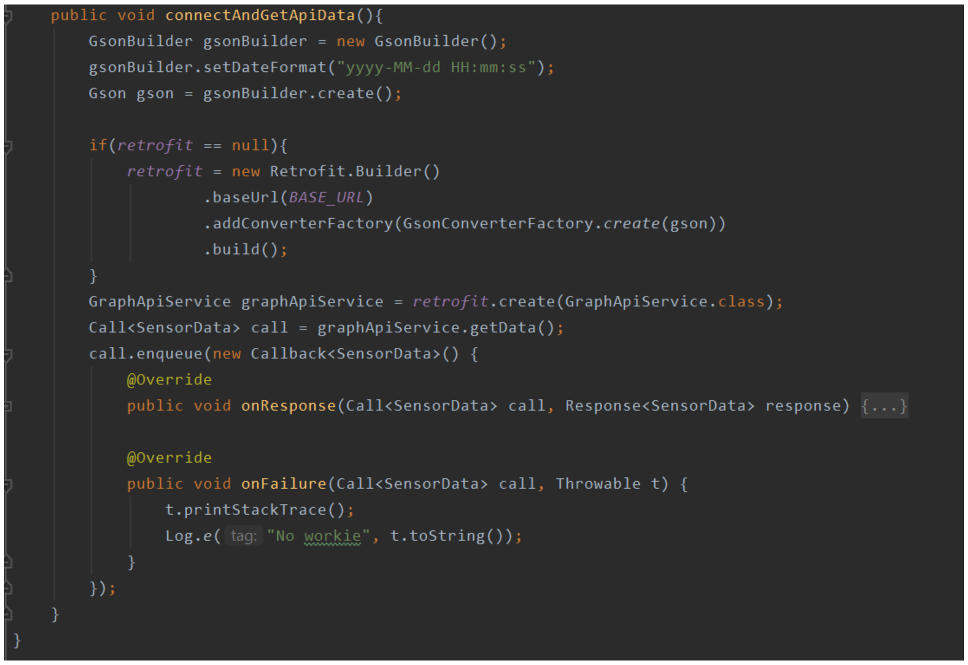

```
    public void connectAndGetApiData(){
        GsonBuilder gsonBuilder = new GsonBuilder();
        gsonBuilder.setDateFormat("yyyy-MM-dd HH:mm:ss");
        Gson gson = gsonBuilder.create();

        if(retrofit == null){
            retrofit = new Retrofit.Builder()
                    .baseUrl(BASE_URL)
                    .addConverterFactory(GsonConverterFactory.create(gson))
                    .build();
        }
        GraphApiService graphApiService = retrofit.create(GraphApiService.class);
        Call<SensorData> call = graphApiService.getData();
        call.enqueue(new Callback<SensorData>() {
            @Override
            public void onResponse(Call<SensorData> call, Response<SensorData> response) {...}

            @Override
            public void onFailure(Call<SensorData> call, Throwable t) {
                t.printStackTrace();
                Log.e( tag: "No workie", t.toString());
            }
        });
    }
}
```

Figure 19: Retrofit Connect to API Function

The database is a relational database with three tables which are a table for the moisture, nutrients and temperature from the device, a table for the user setpoints placed on the moisture, nutrients, and temperature (hi and low) and user authentication/credentials such as password, username, and user ID. The relationships between these three tables can be seen in the entity-relationship diagram shown below.

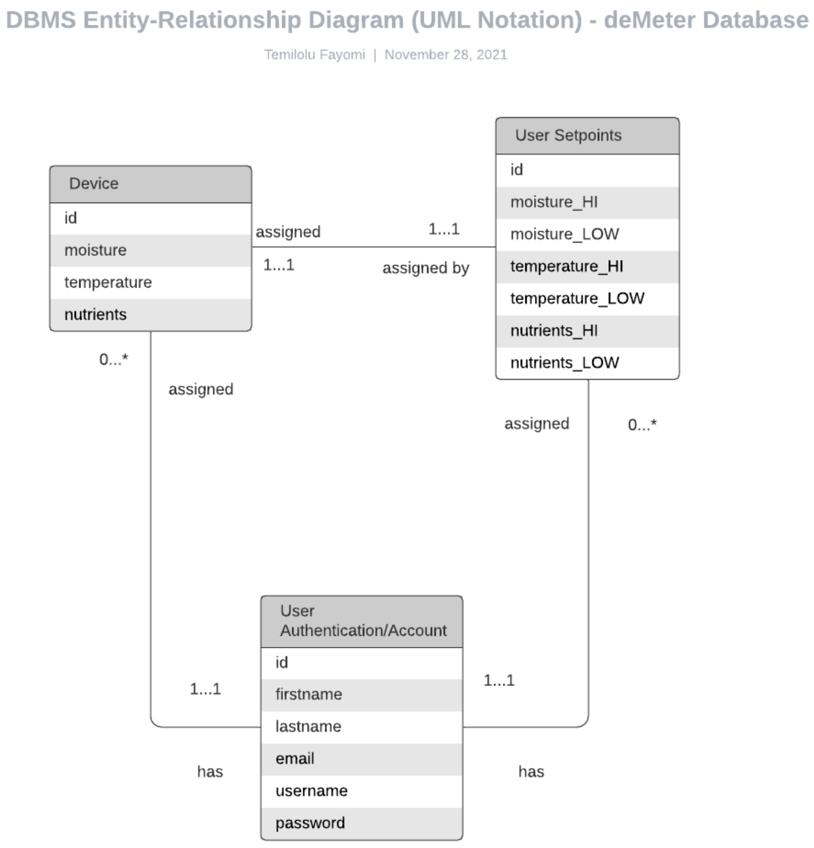


Figure 20: Database Management System Entity Relationship Diagram

Since the device should be easily scalable and users may have more than one device, the relationship between the user auth and device will be one to many. The relationship between user setpoints and device will be one to one, because each device has a specific set of thresholds the users have specified. Since the intended design is to have multiple users, the many to one relationship between user auth and device will be a weak relationship as multiple many to one relationships are needed. On the other hand, the relationship between user accounts and the devices/setpoints is one to many because only one user account will have a multiple number of devices registered and the associating setpoints for each device that is registered.

Once the app is connected to the cloud database, it creates DataStore elements which can be used within the code to either update/insert/add or pull data from the database. Amplify DataStore is an on-device storage engine and allows for the data between the app and the database synchronize in real time and offline, so there is no need for a separate script to update data if the app is connected to the internet and if the app is offline. These datastore are designed after the tables created in Figure 18 and the device Datastore class created in Java is shown in Figures 10, 11 and 13 from the appendices. The code to update the user setpoints data from the app is shown in Figure 12 in the appendices.

There are three different transmissions between the mobile app and the cloud. The mobile app will need to obtain the sensor data from the cloud, the app will send user credentials to the cloud and the user set-points to the cloud. This is the reason for the three different tables. Since IEEE 802.11n will be used to wirelessly transmit, the maximum data rate is 600Mbit/s according to IEEE. Since the only constraint is the user's device and not the microcontroller, the IEEE standard is used instead of the 150Mbit/s.

Mobile App (Client 2)		**Cloud Database (Broker)**	
Packet	**Size**	**Packet**	**Size**
Send Connect to broker	Max 131,111 bytes	Send CONNACK to Client 2	4 bytes
Send SUBSCRIBE to get sensor data from cloud	Max 65,542 bytes	Send SUBACK to Client 2	8 bytes
-	-	Send Disconnect	2 bytes

Table 9: Sensor Data between Mobile App and Cloud

Getting the sensor data from the cloud requires only 327.767 bytes or 2.622136 Mbits. There would need to be 3 subscribe packets and 3 SUBACK packets, since the sensor data will be sent to the cloud in three separate packets as well, with a topic name for each one. The app will subscribe to the topic and receive the data in that manner. The amount of time it would take to send these packets is

$$\frac{600Mbits}{1\ s} \div 2.622136Mbits = 0.00437\ s\ or\ 4.37ms$$

Mobile App (Client 2)		**Cloud Database (Broker)**	
Packet	**Size**	**Packet**	**Size**
Send Connect to Broker	Max 131,111 bytes	Send CONNACK to Client 1	4 bytes
Send User Data	Max 131,076 bytes	Send PUBREC to Client 1	4 bytes
Send PUBREL to broker in response to PUBREC	4 bytes	Send PUBCOMP in response to PUBREL from Client 1	4 bytes
-	-	Send disconnect	2 bytes

Table 10: User Data (username and password) between Mobile App and Cloud

Mobile App (Client 2)		**Cloud Database (Broker)**	
Packet	**Size**	**Packet**	**Size**
Send Connect to Broker	Max 131,111 bytes	Send CONNACK to Client 1	4 bytes
Send User Set-Points	Max 65,543 bytes for a single packet	Send PUBREC to Client 1	4 bytes
Send PUBREL to broker in response to PUBREC	4 bytes	Send PUBCOMP in response to PUBREL from Client 1	4 bytes
-	-	Send disconnect	2 bytes

Table 11: User Set-Points sent from Mobile App to Cloud

Sending the user's inputs (credentials and set points for temperature, moisture and nutrients) is very similar to the ESP32 sending sensor data. The amount of data required for the user credentials is 262,205 bytes or 2.097640 Mbits. The amount of time it would take to send this data would be

$$\frac{600Mbits}{1\ s} \div 2.097640Mbits = 0.0035\ s\ or\ 3.5ms$$

For the user set-points it would be 327,782 bytes or 2.622256 Mbits. It would take

$$\frac{600Mbits}{1\ s} \div 2.622256Mbits = 0.0044\ s\ or\ 4.4ms$$

to finish sending the user set-points to the cloud database. For the user set-points, they will be 16-bit numbers. As mentioned before in the communications section, these are the maximum amount for these packets and the packet sizes will less than this so the times will only improve. While the information on AWS DataStores and the AWS IoT Core SDK are estimates and were not implemented in the final result, if this project is to be continued, the information here can be implemented in further iterations of the soil monitoring system.

III. ENGINEERING REQUIREMENTS SPECIFICATION

Marketing Requirement Number(s):	Engineering Requirement	Justification
4, 5	The device must have at least 3 analog inputs.	To measure a minimum of 3 different soil parameters.
5, 9	The device must be operational within a temp range of -20°F to 120°F.	To ensure that the device will function properly outdoors, all year round.
2, 5, 9	Sensor probes, conductors, and electrodes must be moisture/corrosion resistant.	To ensure that the device will function properly while installed in the soil.
2, 8	Internal battery must be able to reliably recharge from a solar source.	To ensure low maintenance and sustain extended periods without user intervention.
1, 9	The device will be constructed of ABS plastic, aluminum, and copper.	To ensure that the device remains light weight yet durable for installation and use.
3, 5	Temperature sensor must be within 5% accuracy.	To provide reliable data for the user app and setpoints.
3, 5	Moisture sensor must be within 5% accuracy.	To provide reliable data for the user app and setpoints.
3, 5	Nutrient sensor must be within 5% accuracy of nominal RGB value.	To provide reliable data for the user app and setpoints.
4, 6	Must reliably transmit data within a maximum of 3 attempts.	To avoid entering long cycles of retransmission that causes the device to stay in active mode and waste power.
3, 7, 10	The app interface must display the data and allow for user settings.	To allow users to read and interpret the data and set alarms for changing conditions for up to two sensors
6	Must have wireless data transmission range of 50ft-300ft.	To meet applicable range for targeted consumers (home gardeners, small commercial).

Table 12: Engineering Requirements

IV. ENGINEERING STANDARDS SPECIFICATION

Communication

For the link layer, the IEEE WiFi standard 802.11n will be used to wirelessly transmit data from the embedded system to the cloud database. This standard offers faster and more reliable transmission than the other WiFi standards supported by the microcontroller.

For the transport layer, the OASIS standard MQTT will be used. It prioritizes high throughput, low latency, and low bandwidth. It also provides a framework that can be easily scaled up or down.

Data Formats

The ESP32 has a 12-bit analog/digital converter which will be used to bring in sensor data. So, it must be stored in a data type with a size greater than or equal to 12 bits. Negative values are possible real values from the sensor readings, so the data type must also be signed. Taking in these considerations, the data type int16_t has been chosen to hold sensor data. To avoid losing accuracy due to the truncation behavior of integer types, the sensor readings will be stored in tenths of a unit. The sensor data must be formatted into a string using sprintf before being sent to the cloud database. Decimal points will be inserted after the sensor data has been formatted into a string.

Programming Languages

For the firmware, C/C++ will be used. These languages are suited for the job because they offer direct memory control and are fast in terms of execution compared to higher-level languages such as Python.

For the mobile application, Java will be used. The IDE Android Studio supports Java and makes it simple to develop and deploy applications on Android.

For the cloud database, MySQL will be used. The database will be relational database management system which will allow users to access the database at the same time. This type of database will be simple to deploy and be easily accessible.

Data Interpretation

The Munsell Soil Color chart will be used to correlate the RGB value of the soil substrate with the corresponding hue. The hue in the color chart is then equated to dominant soil nutrients presence, soil deficiencies, drainage, organic mater concentrations, and overall soil health.

V. ACCEPTED TECHNICAL DESIGN

A. Hardware Design: Power, Sensors

Level 0 Block Diagram

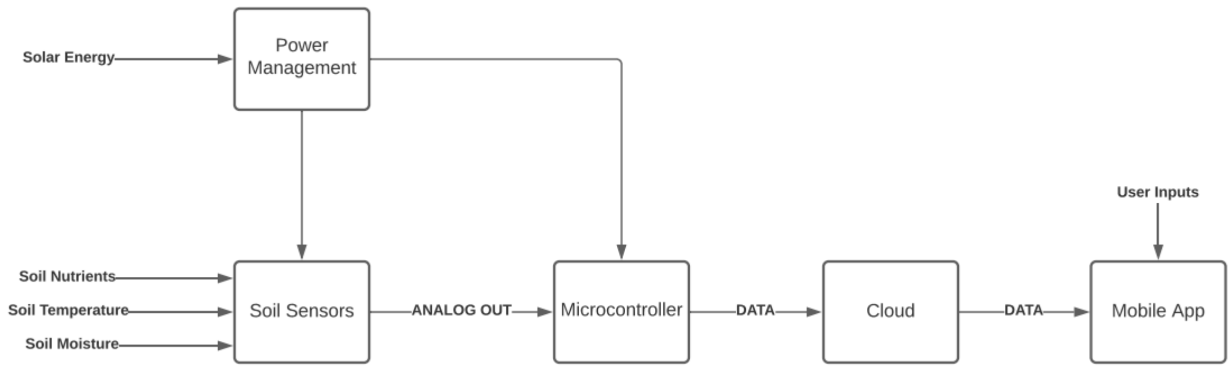


Fig. 21: Level 0 Hardware Block Diagram

Module	Soil Sensors
Designer	• Ryan Matthews
Inputs	• Nitrogen, phosphorous, potassium, salts, and other nutrients • Soil moisture • Soil Temperature • PH • Power
Outputs	• Analog and digital output signals
Functionality	The sensors acquire soil data and relay that information back to the microcontroller for processing and transmission to the cloud and then to the app.

Module	Power Management
Designer	• Alex Fuller
Inputs	• Solar energy
Outputs	• DC power
Functionality	The power management module will use batteries to provide power to all the subsystems in the probe and then recharge those batteries using a solar panel.

Module	Microcontroller
Designer	• Rachel Rummer
Inputs	• DC Power (3.3V) • Sensor signals
Outputs	• Processed data
Functionality	The microcontroller takes the raw data acquired by the circuits via an analog or digital signal and converts it into usable values to be transmitted and ultimately received by the user.

Level 1 Block Diagram

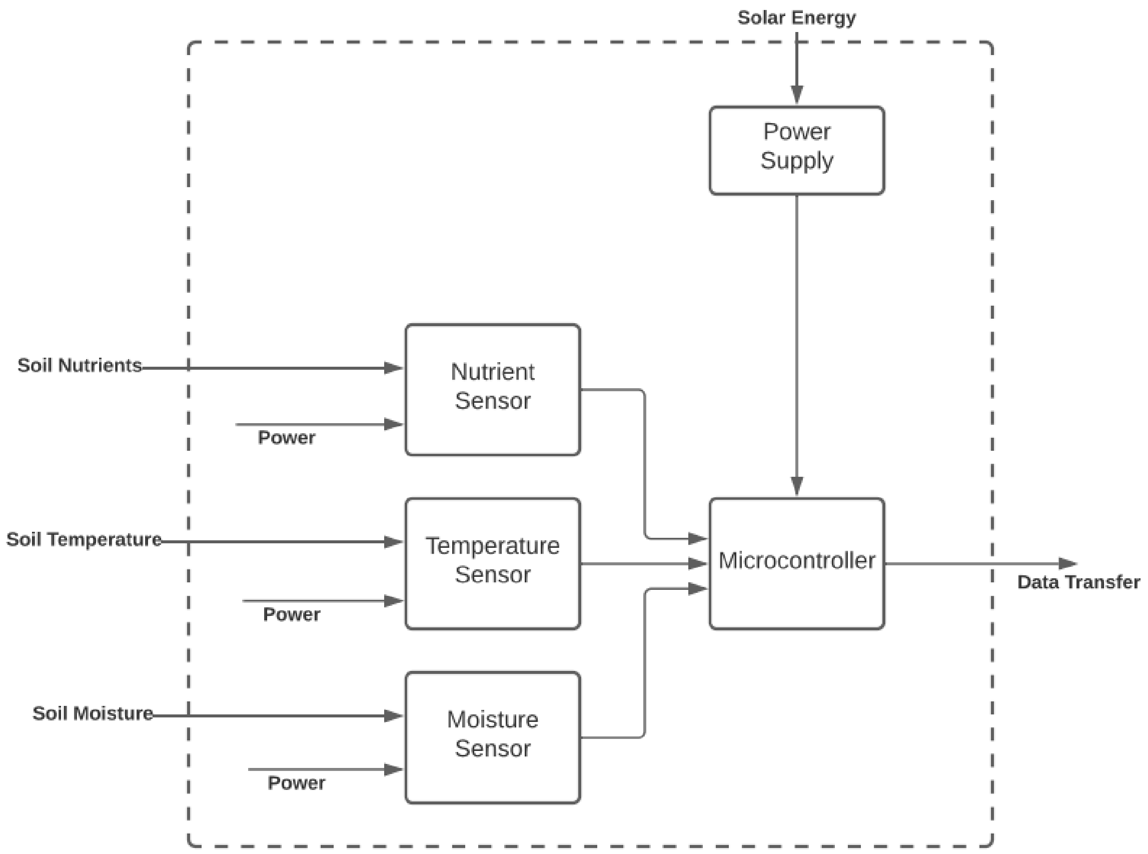


Fig. 22: Level 1 Hardware Block Diagram

Module	Nutrient Sensor
Designer	• Ryan Matthews
Inputs	• Nitrogen, phosphorous, potassium, salts, PH and other nutrients • Power
Outputs	• Digital output signals
Functionality	The sensors acquire soil nutrient data via an optical sensor and relay that information back to the microcontroller for processing and transmission to the cloud and then to the app.

Module	Temperature Sensor
Designer	• Ryan Matthews
Inputs	• Soil temperature • Power
Outputs	• Analog output signals
Functionality	The sensors acquire soil temperature data via thermistors and relay that information back to the microcontroller for processing and transmission to the cloud and then to the app.

Module	Moisture Sensor
Designer	• Ryan Matthews
Inputs	• Soil moisture • Power
Outputs	• Analog output signals
Functionality	The sensors acquire soil temperature data via a capacitive method and relay that information back to the microcontroller for processing and transmission to the cloud and then to the app.

Module	Power Supply
Designer	• Alex Fuller
Inputs	• Solar energy
Outputs	• DC power
Functionality	The power supply module will use batteries to provide power to all the subsystems in the probe and then recharge those batteries using a solar panel. This will also include voltage regulation on the output and conditioning on the incoming.

Module	Microcontroller
Designer	• Rachel Rummer
Inputs	• DC Power (3.3V) • Sensor signals
Outputs	• Processed data
Functionality	The microcontroller takes the raw data acquired by the circuits via analog and digital signals and converts it into usable values to be transmitted and ultimately received by the user.

Level 2 Block Diagram

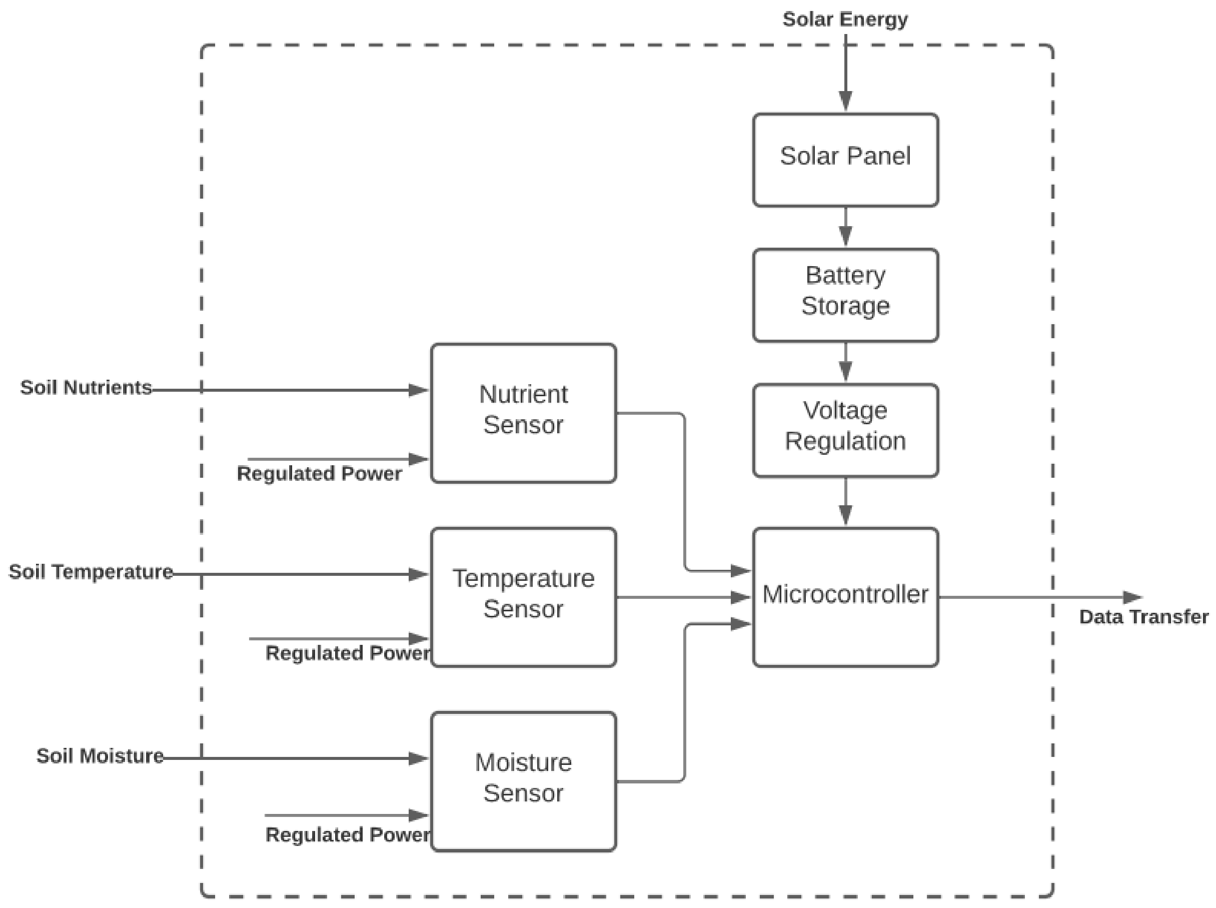


Fig. 23: Level 2 Hardware Block Diagram

Module	Nutrient Sensor
Designer	• Ryan Matthews
Inputs	• Nitrogen, phosphorous, potassium, salts, PH, and other nutrients • Power
Outputs	• Digital output signals
Functionality	The sensors acquire soil nutrient data via an optical color sensor and relay the color code back to the microcontroller for processing and transmission to the cloud and then to the app.

Module	Temperature Sensor
Designer	• Ryan Matthews
Inputs	• Soil temperature • Power
Outputs	• Analog output signals
Functionality	The sensors acquire soil temperature data via thermistors and a voltage divider and relay that analog voltage back to the microcontroller for processing and transmission to the cloud and then to the app.

Module	Moisture Sensor
Designer	• Ryan Matthews
Inputs	• Soil moisture via capacitance • Power
Outputs	• Analog output signals
Functionality	The sensors acquire soil temperature data via a capacitive method and relay the analog voltage output back to the microcontroller for processing and transmission to the cloud and then to the app.

Module	Solar Panel
Designer	• Alex Fuller
Inputs	• Solar energy
Outputs	• DC power
Functionality	The solar panel module will provide renewable energy to charge the batteries, condition the incoming power, and relay that to the battery for storage.

Module	Battery Storage
Designer	• Alex Fuller
Inputs	• Solar panel DC power
Outputs	• DC power
Functionality	The battery storage module will power to the module for extended periods of time without input. It will be recharged by the solar panel and will deliver 3.3V DC to the sensors and the microcontrollers.

Module	Voltage Regulation
Designer	• Alex Fuller • Ryan Matthews
Inputs	• DC battery
Outputs	• Regulated DC
Functionality	The voltage regulation module will be used to ensure a precise and consistent 3.3V on the sensor, is delivered at all times. The incoming power will be regulated and the output passed to the sensors.

Module	Microcontroller
Designer	• Rachel Rummer
Inputs	• DC Power (3.3V) • Analog/Digital sensor signals
Outputs	• Processed data
Functionality	The microcontroller takes the raw data acquired by the circuits via an analog signal and converts it into usable values to be transmitted and ultimately received by the user.

Level 3 Block Diagram

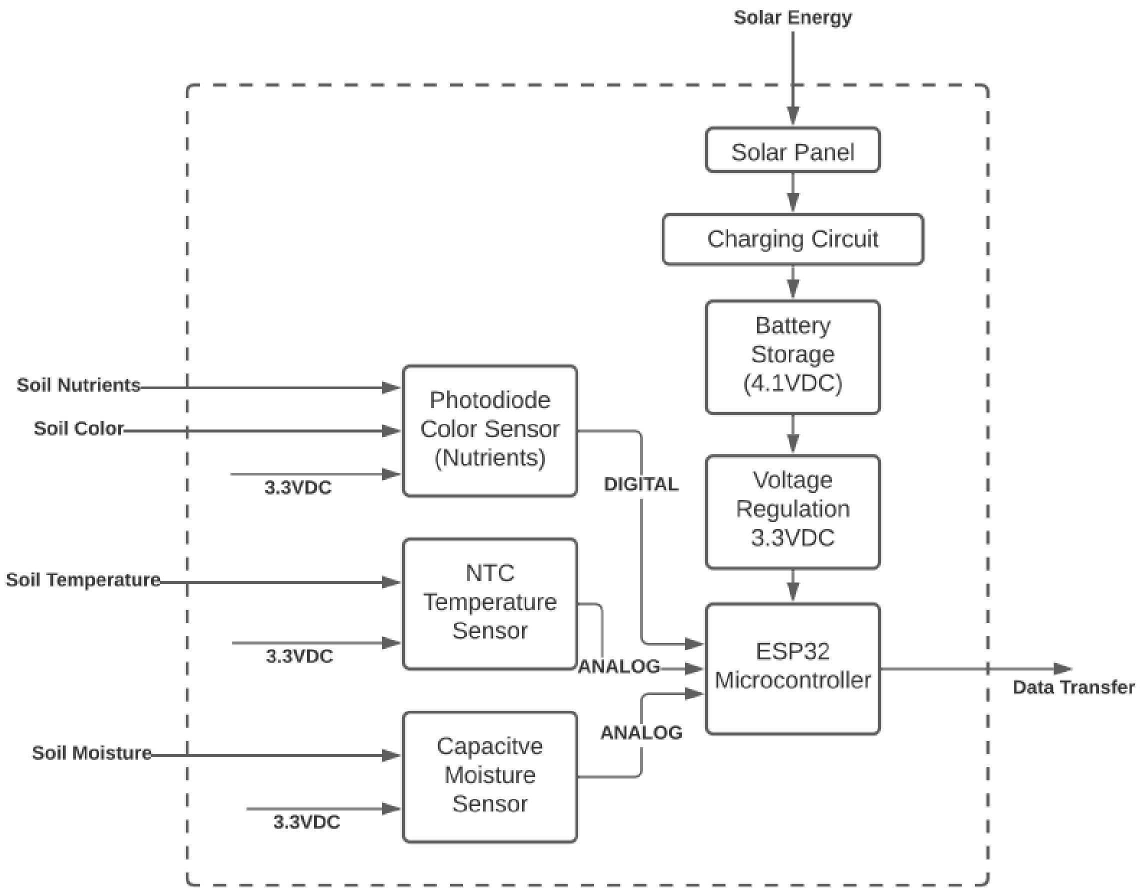


Fig. 24: Level 3 Hardware Block Diagram

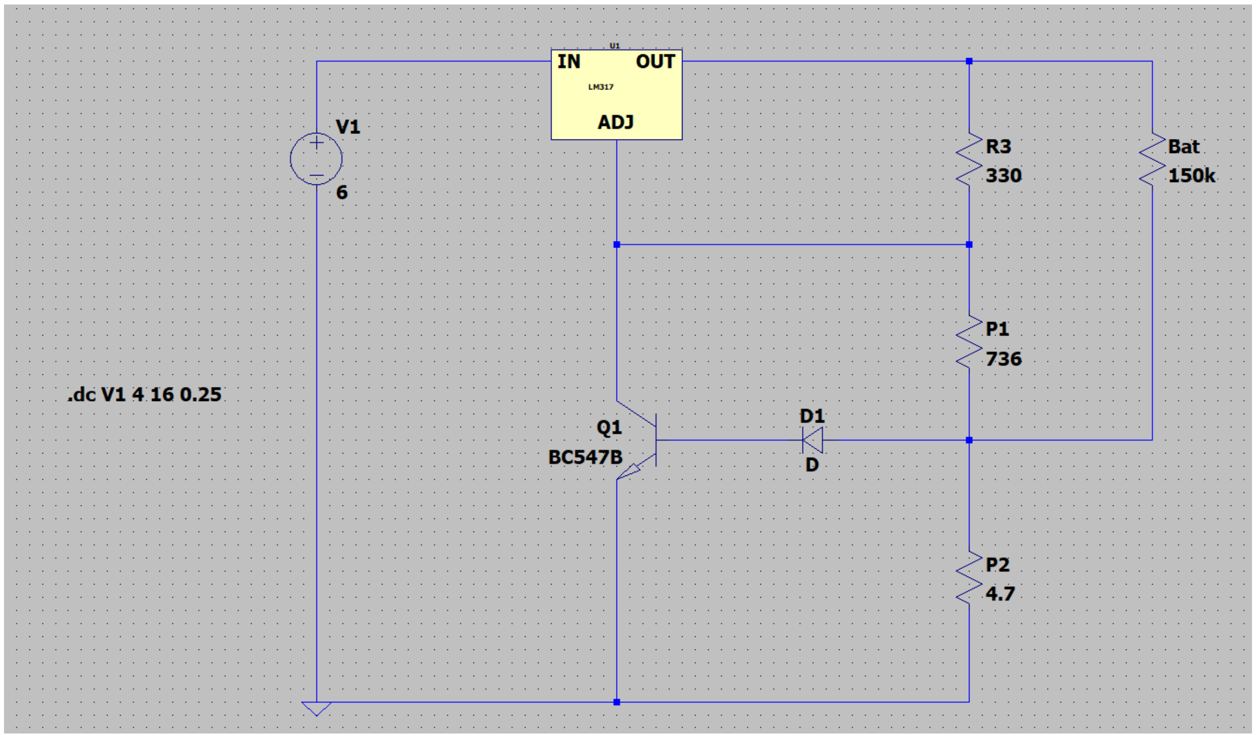


Fig. 25: Charging Circuit Schematic

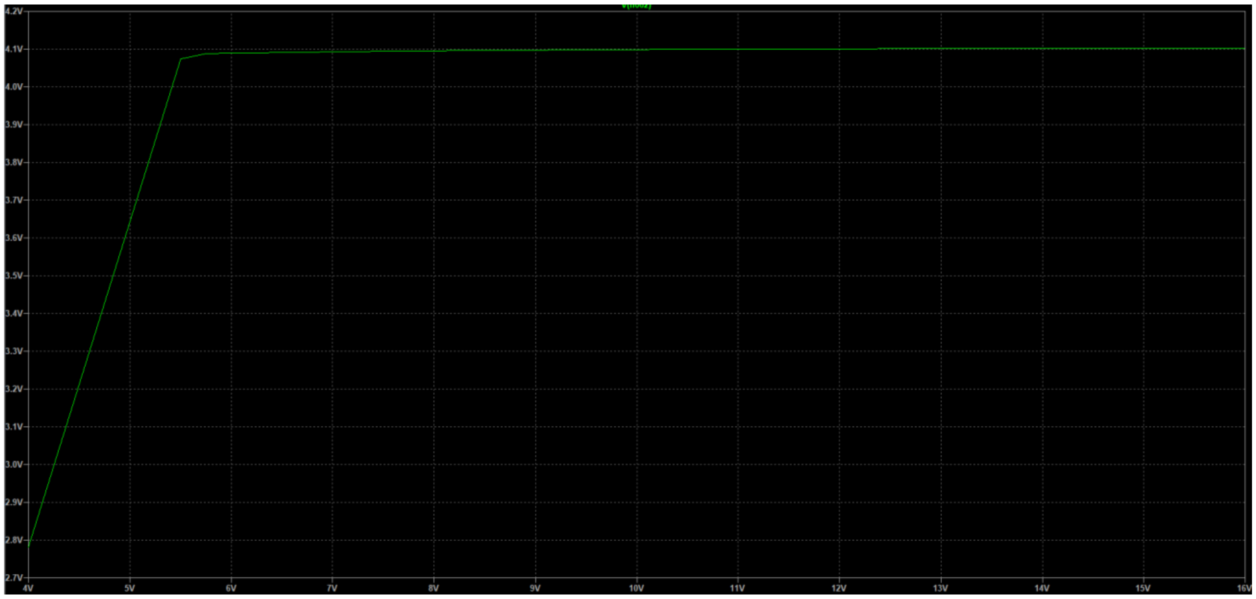


Fig. 26: Charging Circuit Output

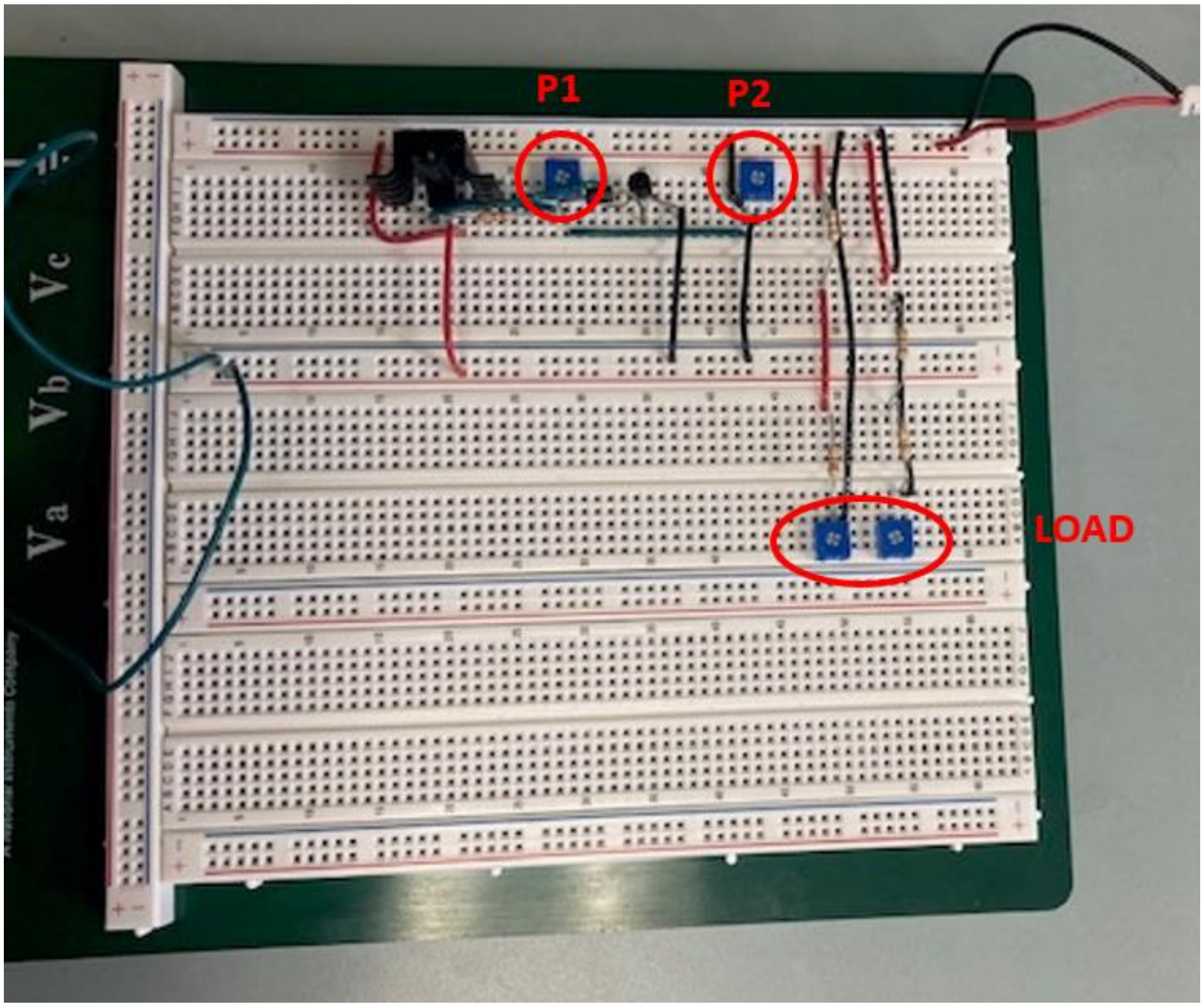


Fig. 27: Physical Charging Circuit

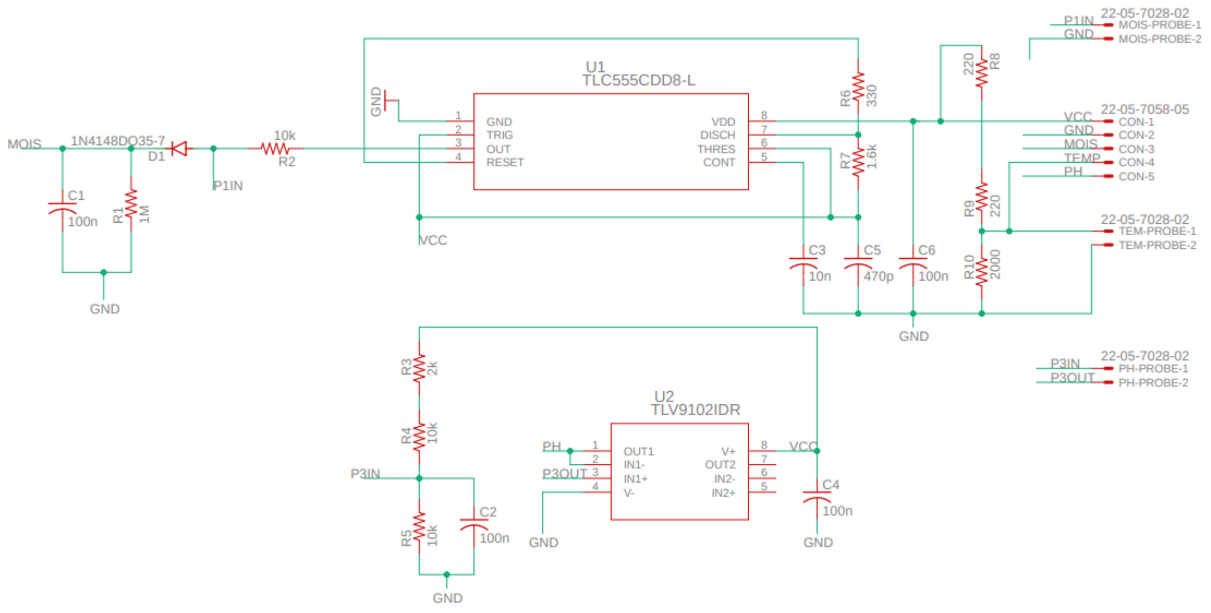


Fig. 28: Final accepted design of Sensor PCB Board Schematic (Includes Moisture and Temperature)

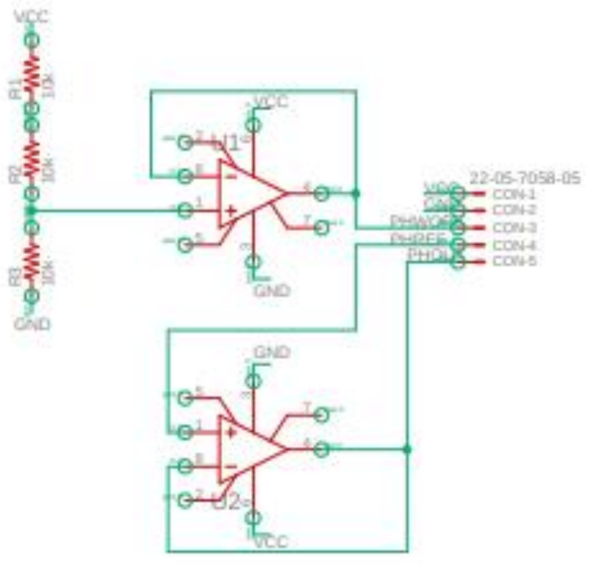


Fig. 29: Final accepted design of PH Sensor PCB Board Schematic

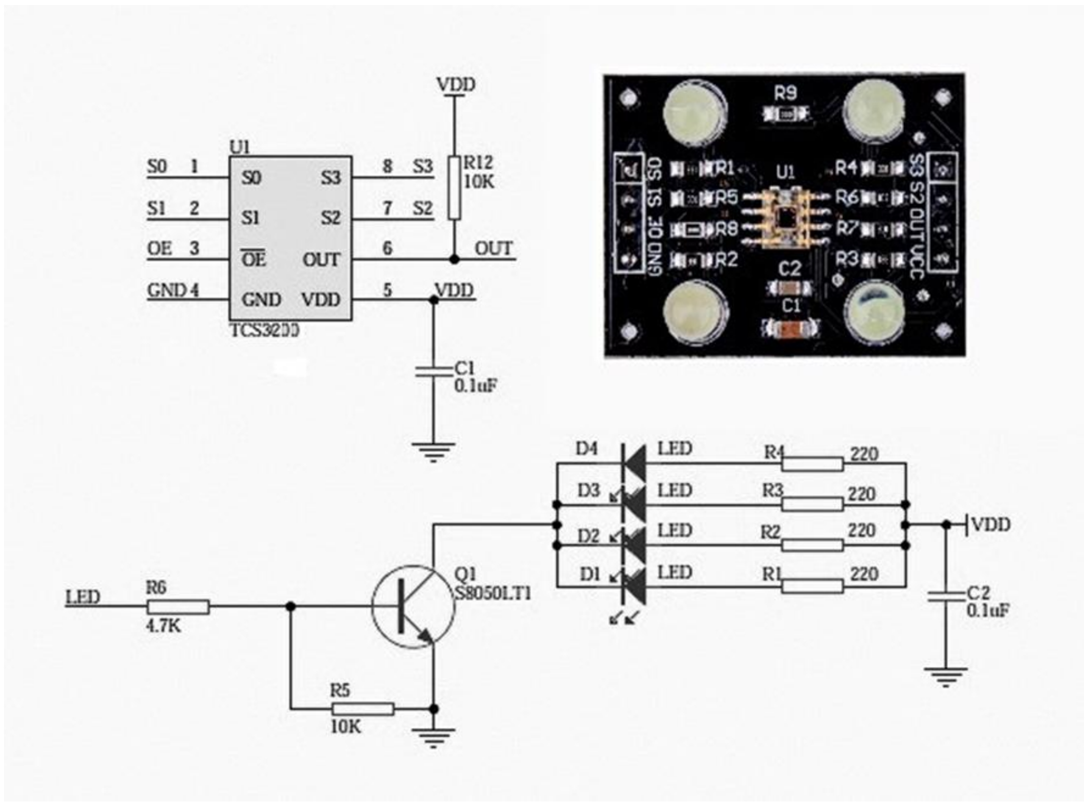


Fig. 30: Color light-to-frequency converter schematic (Courtesy of ACOPTEX)

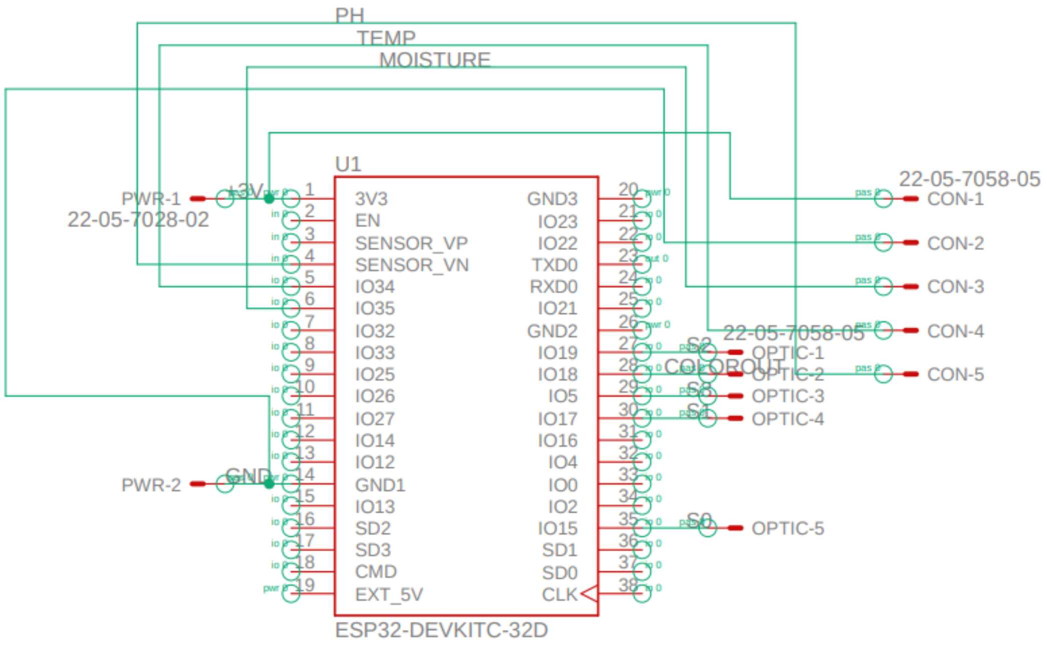


Fig. 31: ESP32 Connection Board Schematic

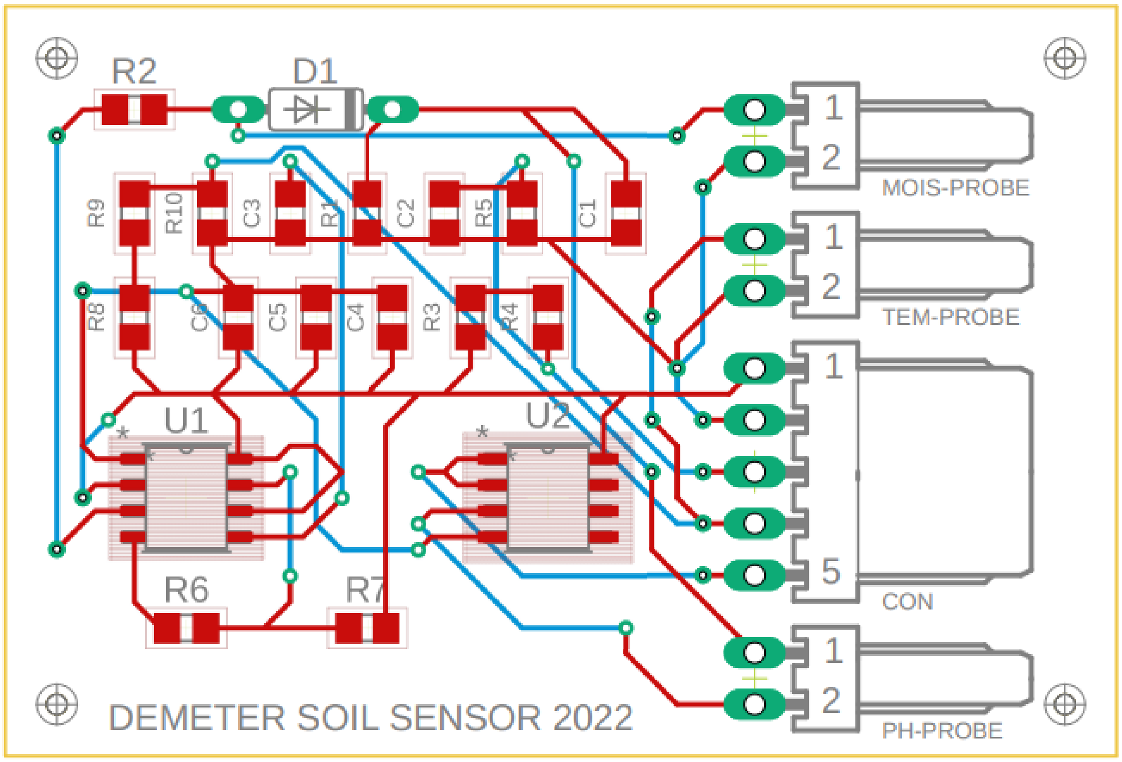


Fig. 32: Accepted Sensor Board PCB

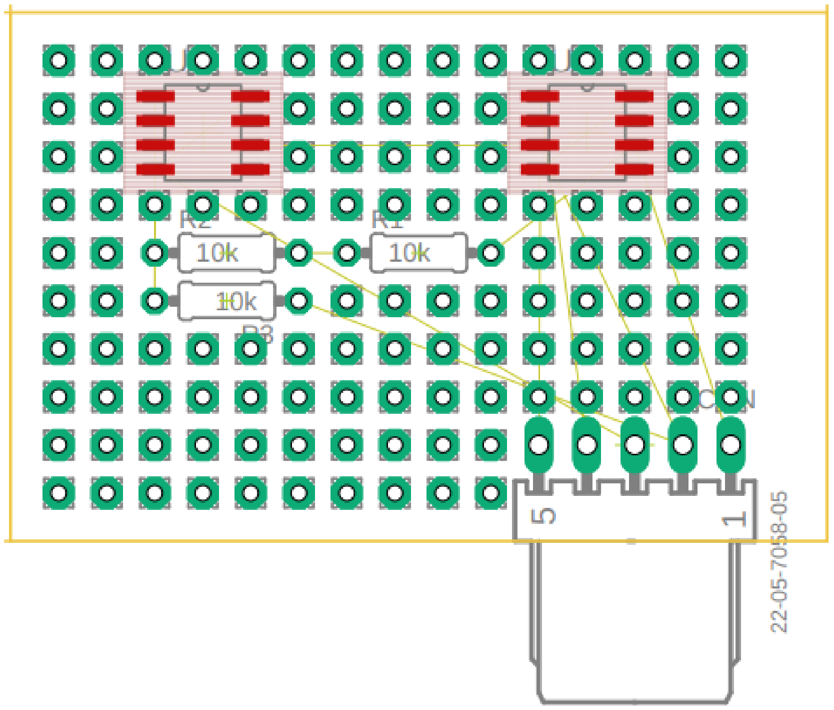


Fig. 33: Accepted PH Protoboard/PCB

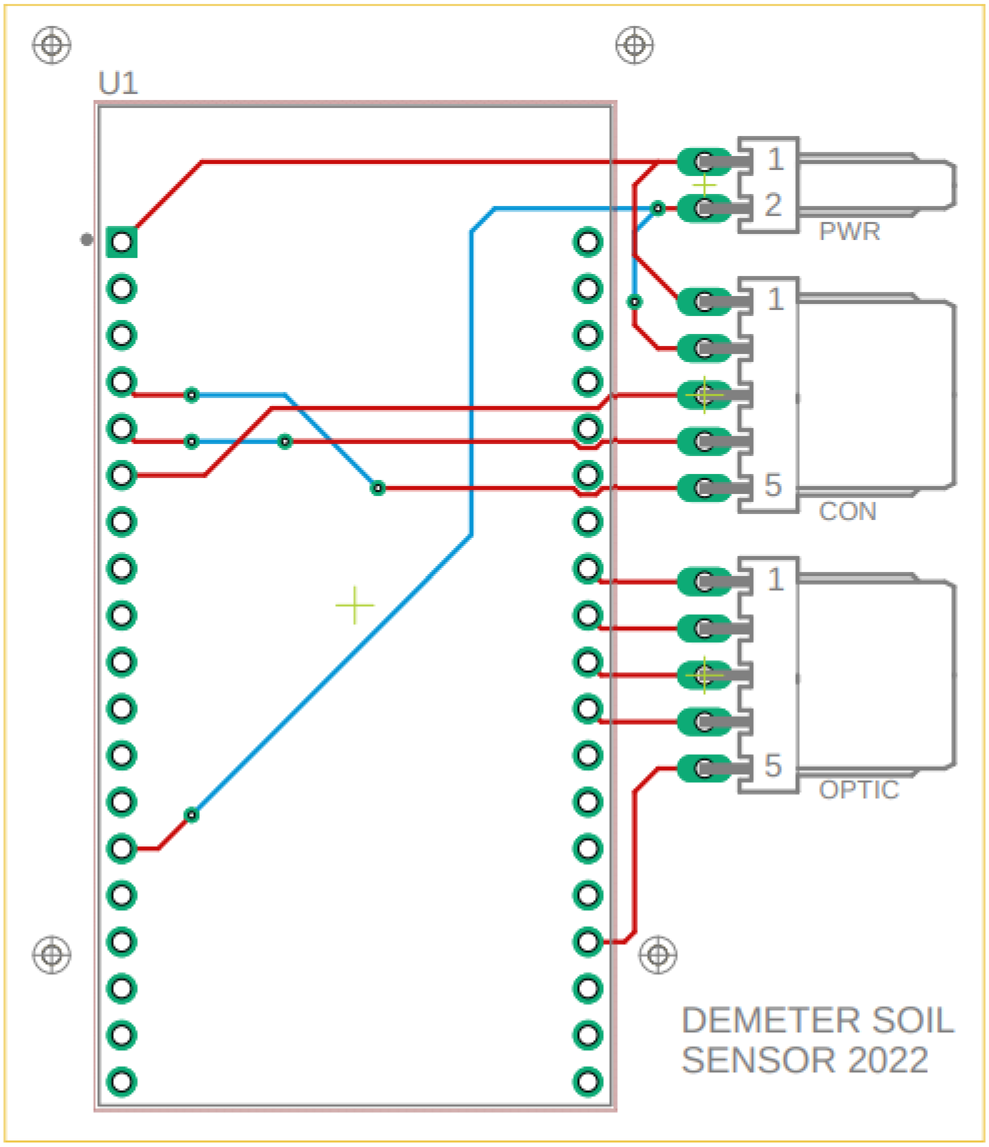


Fig. 34: Accepted ESP32 PCB

Module	Photodiode Color Sensor for Nutrients
Designer	• Ryan Matthews
Inputs	• Soil pigments due to nitrogen, phosphorous, potassium, salts, and other nutrients • Soil pigments due non-nutrients • Power (3.3VDC)
Outputs	• Digital output signals
Functionality	The sensors acquire soil nutrient data via an optical color sensor and creates a unique RGB code for the soil in the form of a digital output which is transmitted to the cloud and then to the app.

Module	NTC Temperature Sensor
Designer	• Ryan Matthews
Inputs	• Soil temperature • Power (3.3VDC)
Outputs	• Analog output signals
Functionality	The sensors acquire soil temperature data via an NTC thermistor and a voltage divider, relays that analog voltage back to the microcontroller for processing and transmission to the cloud and then to the app.

Module	Capacitive Moisture Sensor
Designer	• Ryan Matthews
Inputs	• Soil moisture via capacitance • Power (3.3VDC)
Outputs	• Analog output signals
Functionality	The sensors acquire soil temperature data via a probe capacitor method and generated pulse. A filter and rectifier relay an analog output to the controller for processing and transmission to the cloud and then to the app.

Module	Solar Panel
Designer	• Alex Fuller
Inputs	• Solar energy
Outputs	• DC power
Functionality	The solar panel module will provide renewable energy to charge the batteries, condition the incoming power, and relay that to the battery for storage.

Module	Charging Circuit
Designer	• Alex Fuller
Inputs	• Solar panel DC power
Outputs	• Regulated DC power
Functionality	The charging circuit module will provide regulated power to charge the battery and prevent it from exceeding the recommended voltage and current specifications.

Module	Battery Storage
Designer	• Alex Fuller
Inputs	• Solar panel DC power
Outputs	• DC power
Functionality	The battery storage module will provide power to the module for extended periods of time without input. It will be recharged by the solar panel and will deliver 3.3V DC to the sensors and the microcontrollers.

Module	Voltage Regulation
Designer	• Alex Fuller
Inputs	• DC battery
Outputs	• Regulated DC
Functionality	The voltage regulation module will be used to ensure a precise and consistent 3.3V on the sensor, is delivered at all times. The incoming power will be regulated and the output passed to the sensors.

Module	ESP32 Microcontroller
Designer	• Rachel Rummer
Inputs	• DC Power (3.3V) • Analog/Digital sensor signals
Outputs	• Processed data
Functionality	The microcontroller takes the raw data acquired by the circuits via an analog signal and converts it into usable values to be transmitted and ultimately received by the user.

B. Software Design: Mobile App & Microcontroller

Level 0 Block Diagram

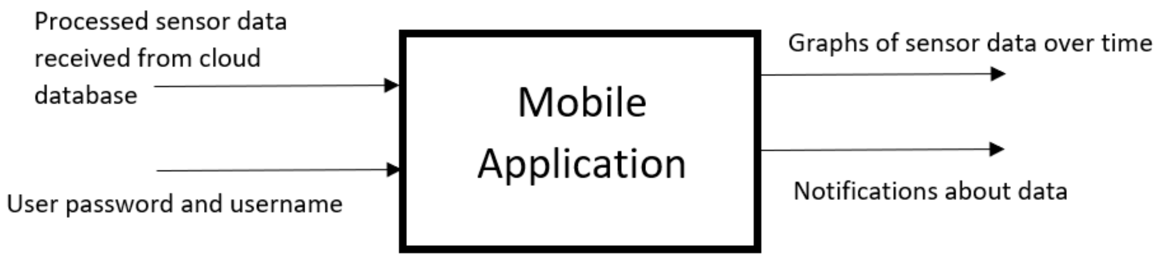

Processed sensor data received from cloud database
User password and username
Mobile Application
Graphs of sensor data over time
Notifications about data

Fig. 35: Level 0 Mobile Block Diagram

Module	Mobile Application
Designer	Temilolu Fayomi
Inputs	Processed sensor data (temperature, moisture, nutrients) stored in cloud database, users' usernames, and passwords
Outputs	Graph of temperature over time and moisture over time, notifications based upon data points outside of user's setpoints
Functionality	The mobile application will let users login to their account to view data over time from the remote sensor device.

Level 1 Flowchart (Mobile app)

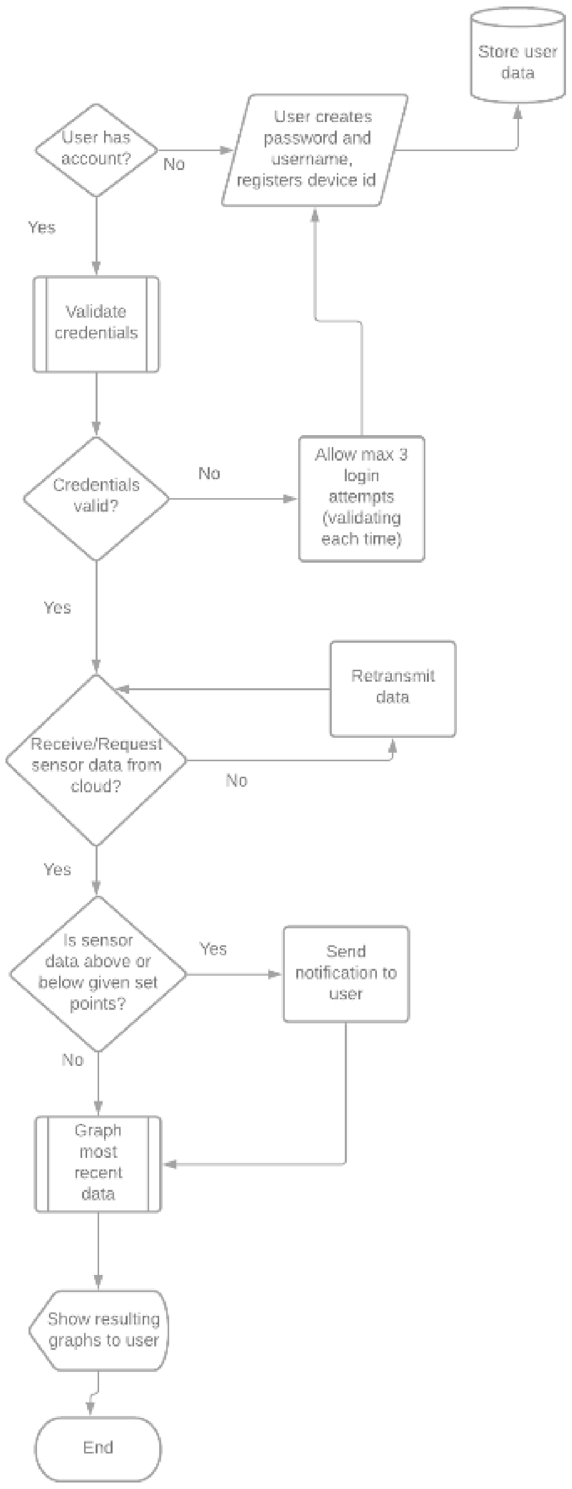

Store user data
User has account?
No
User creates password and username, registers device id
Yes
Validate credentials
Credentials valid?
No
Allow max 3 login attempts (validating each time)
Yes
Retransmit data
Receive/Request sensor data from cloud?
No
Yes
Is sensor data above or below given set points?
Yes
Send notification to user
No
Graph most recent data
Show resulting graphs to user
End

Fig. 36: Level 1 Mobile Flowchart

Module	Microcontroller
Designer	Rachel Rummer
Inputs	• Analog sensor data, power
Outputs	• Processed sensor data
Functionality	Read and process sensor data and send wirelessly to a cloud database.

Level 0 Block Diagram (Microcontroller)

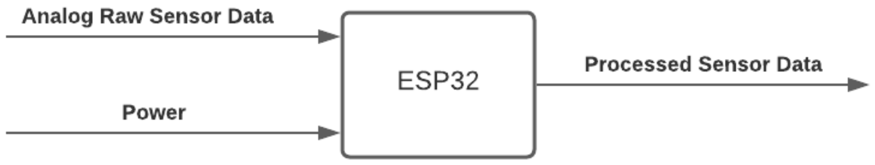
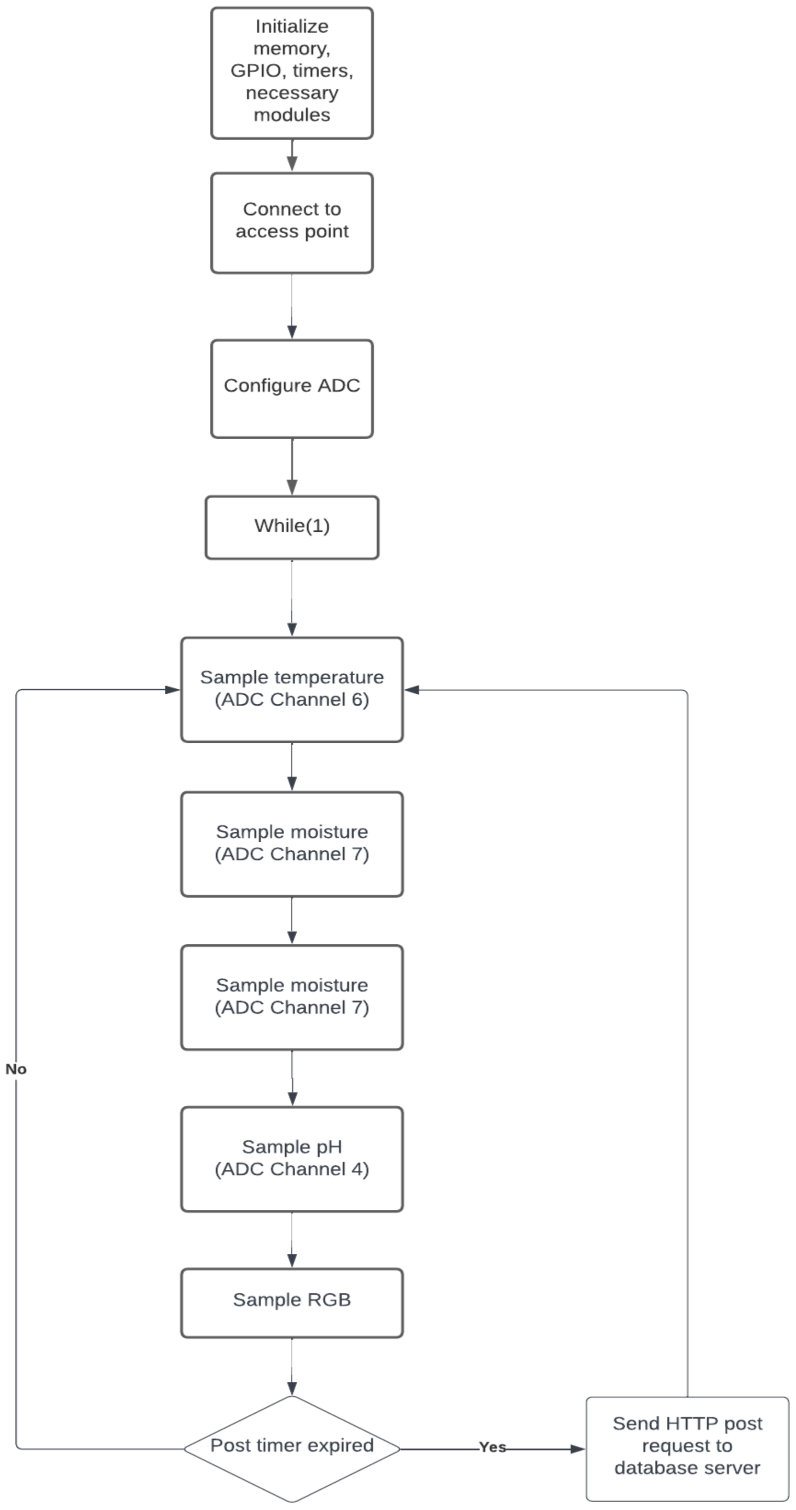


Fig. 37: Level 0 Microcontroller Block Diagram

The level-0 block diagram for the microcontroller subsystem shows the inputs and outputs for the embedded device. Constant power will be supplied by the power management subsystem and the raw sensor data will be supplied by the sensor subsystem. To read the raw sensor data, I2C will be used to interface with the sensors. Sensor data will be read as a 12-bit value from the ADC and stored in 16-bit integers. The data will be processed using fixed-point arithmetic to avoid losing accuracy due to truncation. A time-based interrupt will be implemented to wake up the CPU and send the processed sensor data to the cloud database. For this UART will be used to interface with the ESP32 WiFi module. The flow charts below show the basic flow of data for both the main thread and the interrupt thread.

Level 1 Flow Chart (Microcontroller):



Initialize memory, GPIO, timers, necessary modules
Connect to access point
Configure ADC
While(1)
Sample temperature (ADC Channel 6)
Sample moisture (ADC Channel 7)
Sample moisture (ADC Channel 7)
Sample pH (ADC Channel 4)
Sample RGB
Post timer expired
No
Yes
Send HTTP post request to database server

Fig. 38: Level 1 Microcontroller Flowchart

Level 2 Flow Chart (Microcontroller) ADC readings:

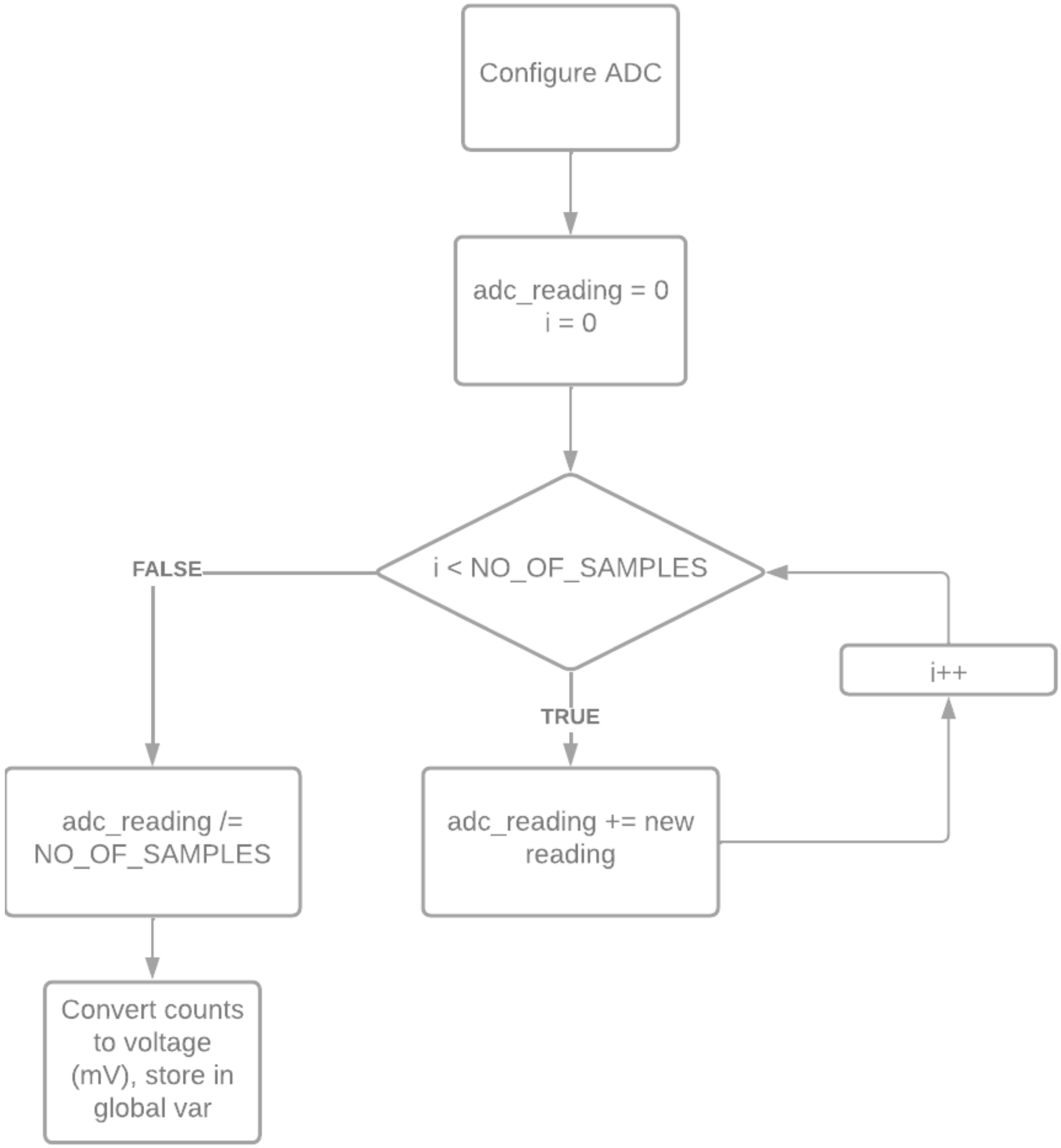

Configure ADC
adc_reading = 0
i = 0
i < NO_OF_SAMPLES
FALSE
TRUE
i++
adc_reading /= NO_OF_SAMPLES
adc_reading += new reading
Convert counts to voltage (mV), store in global var

Fig. 39: Level 2 Microcontroller Flowchart – ADC readings

Level 2 Flow Chart (Microcontroller) RGB readings:

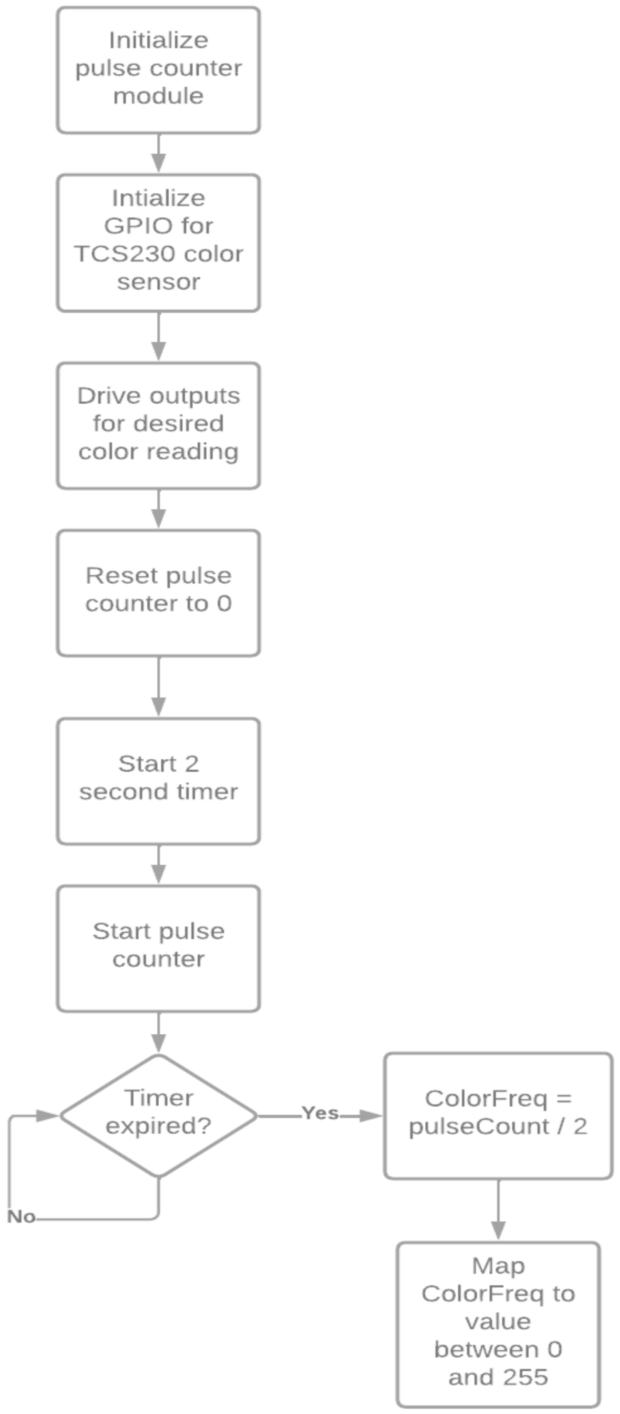

Initialize pulse counter module
Intialize GPIO for TCS230 color sensor
Drive outputs for desired color reading
Reset pulse counter to 0
Start 2 second timer
Start pulse counter
Timer expired?
Yes
No
ColorFreq = pulseCount / 2
Map ColorFreq to value between 0 and 255

Fig. 40: Level 2 Microcontroller Flowchart RGB sampling

VI. MECHANICAL SKETCH

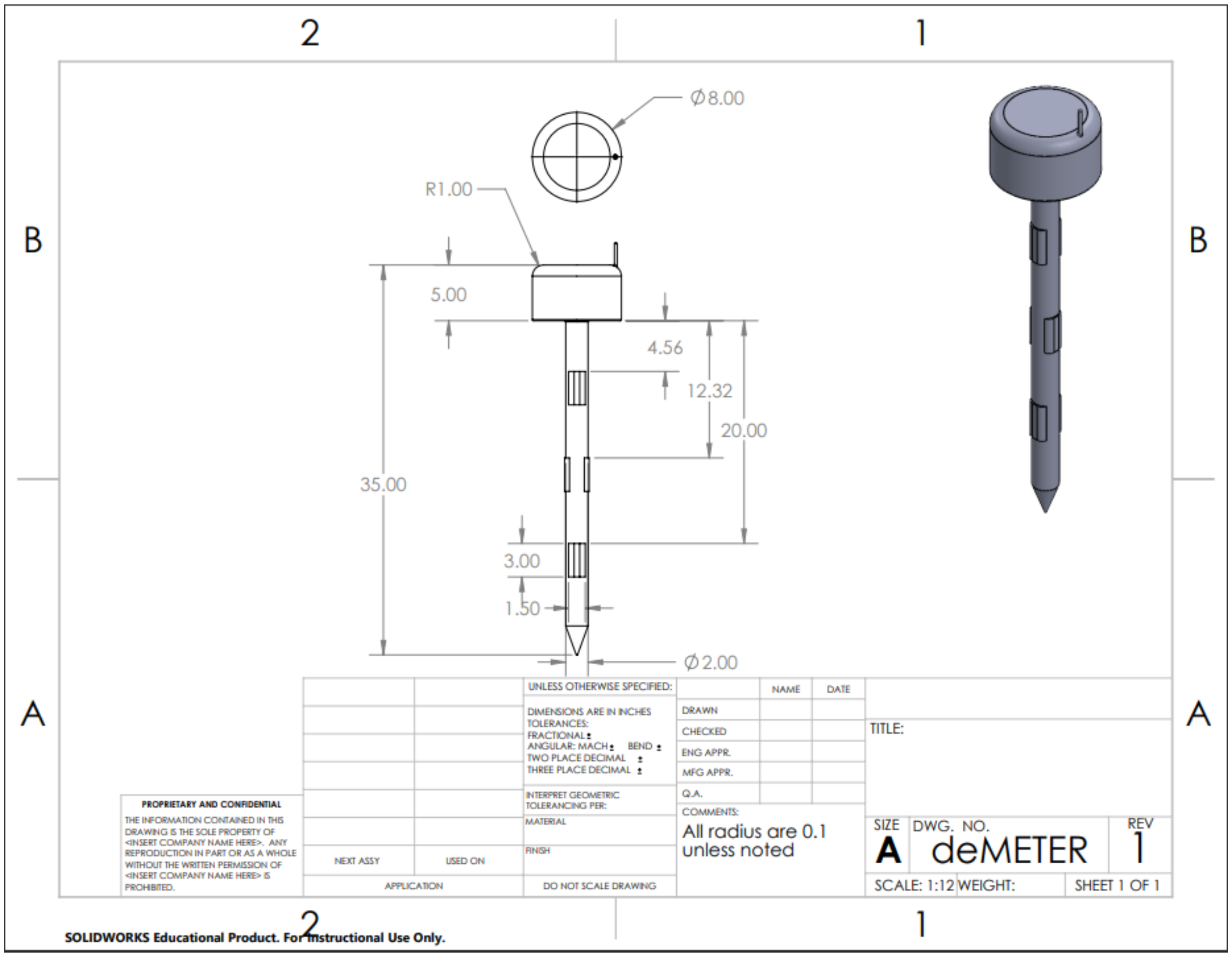

2
1
B
A
Ø8.00
R1.00
5.00
4.56
12.32
20.00
35.00
3.00
1.50
Ø2.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±
INTERPRET GEOMETRIC TOLERANCING PER:
MATERIAL
FINISH
NEXT ASSY
USED ON
APPLICATION
DO NOT SCALE DRAWING
NAME
DATE
DRAWN
CHECKED
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
All radius are 0.1 unless noted
TITLE:
SIZE
A
DWG. NO.
deMETER
REV
1
SCALE: 1:12
WEIGHT:
SHEET 1 OF 1
PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.
SOLIDWORKS Educational Product. For Instructional Use Only.

Fig. 41: Mechanical Sketch

VII. TEAM INFORMATION

Rachel Rummer

- Project Leader
- Computer Engineering

Temilolu Fayomi

- Software Manager
- Computer Engineering

Ryan Matthews

- Engineering Data Manager
- Electrical Engineering

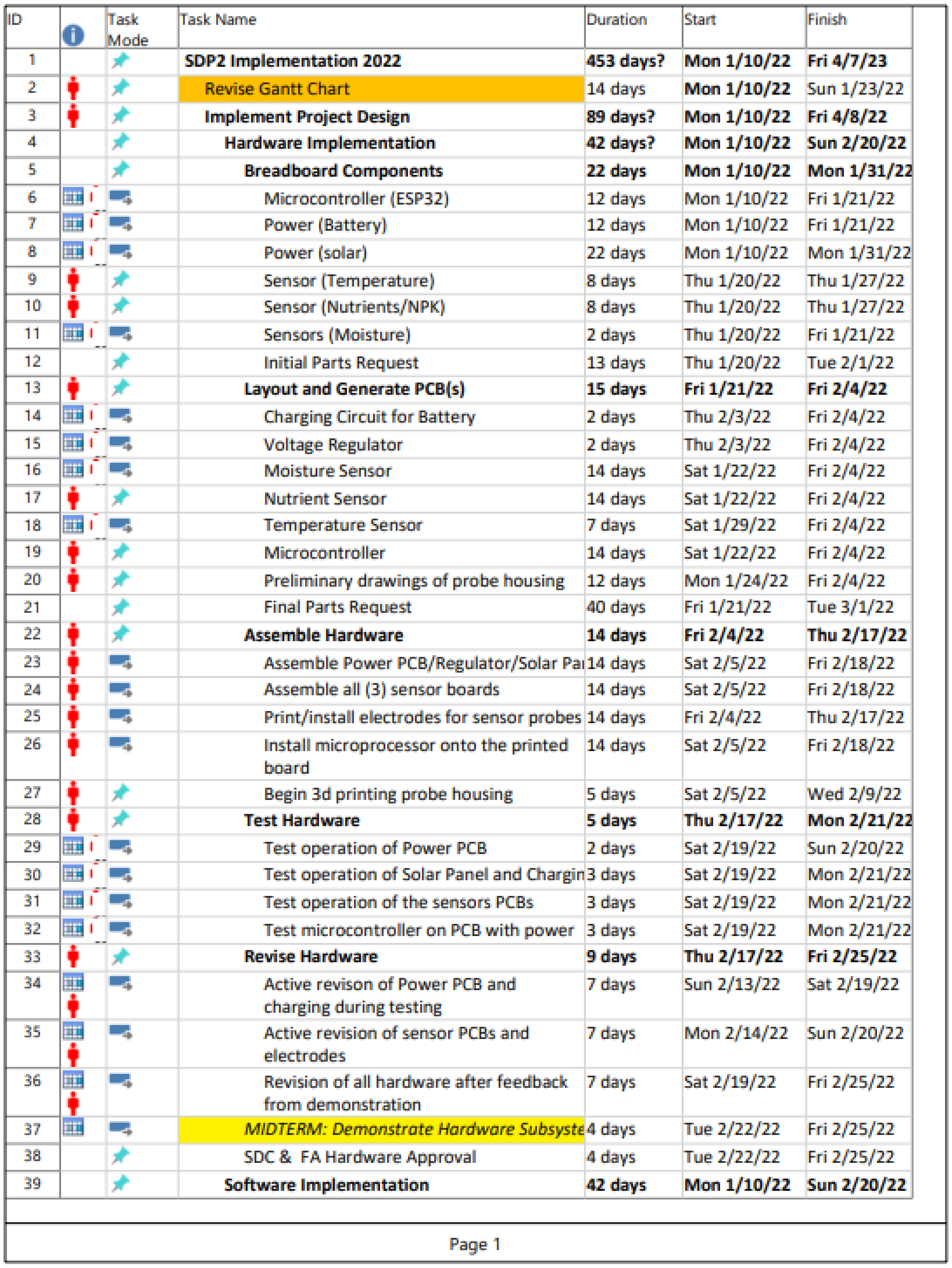
Alex Fuller

- Hardware Manager
- Electrical Engineering

VIII. PARTS LIST

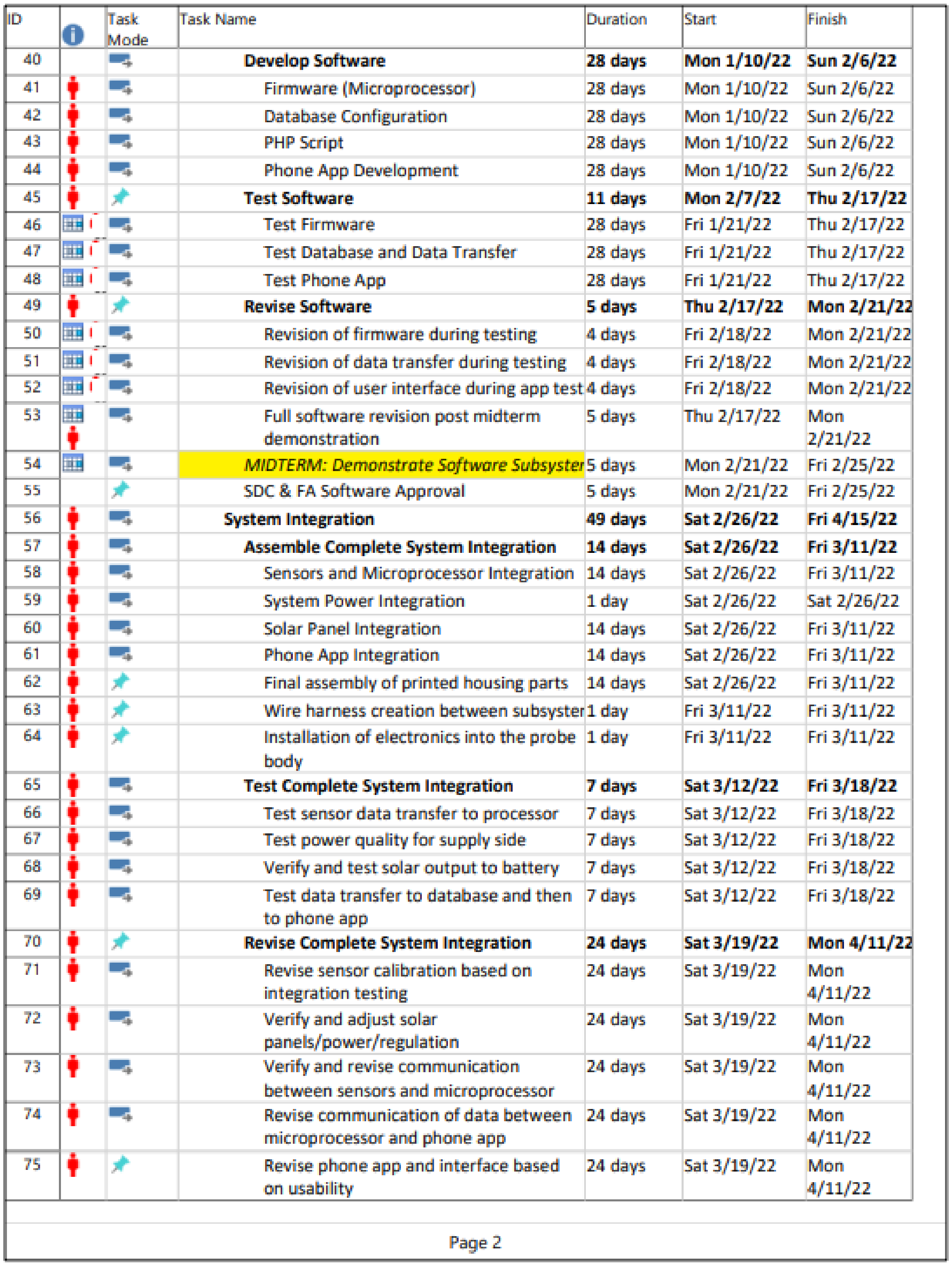
Subsystem	Part Name	Quantity
Moisture Sensor+Temperature Sensor	TLC555CPSR	1
(FOR THE PCB)	C0805C104K5RAC7210	2
	C0805C103J5RAC7800	1
	C0805C471J5GAC7800	1
	RCV08051M00FKEA	1
	CRCW080510K0FKEAC	1
	CRCW08052K00FKEAC	1
	CRCW0805330RFKEAC	1
	CRCW08051K60FKEAHP	1
	CRCW0805220RFKEAC	2
	1N4148 Diode	1
	0022124052 Molex 5 Pin Right Angle	1
	0022124022 Molex 2 Pin Right Angle	2
	Copper Sheet	1
	450Ω Resistor	1
	2kΩ Resistor	1
	NTCLE100E3681 Thermistor	1
PH Sensor	LMP7721MA/NOPB	2
(FOR THE PCB/PROTOBOARD)	PA0001 8pin SMD to DIP	2
	SBBTH1510-1 Protoboard	1
	10kΩ Resistor	3
	0022124052 Molex 5 Pin Right Angle	1
Nutrient Sensor	TCS3200 Color Light-to-Freq Converter	1
Charging Circuit	LM317	1
	330Ω Resistor	1
	BC547 Transistor	1
	1N4007 Diode	1
	1kΩ Potentiometer	1
	50Ω Potentiometer	1
	3.7V Vidar Battery	1
Microcontroller	ESP32	1
(MOUNTED TO PCB)	0022124052 Molex 5 Pin Right Angle	2
	0022124022 Molex 2 Pin Right Angle	1

IX. PROJECT SCHEDULES



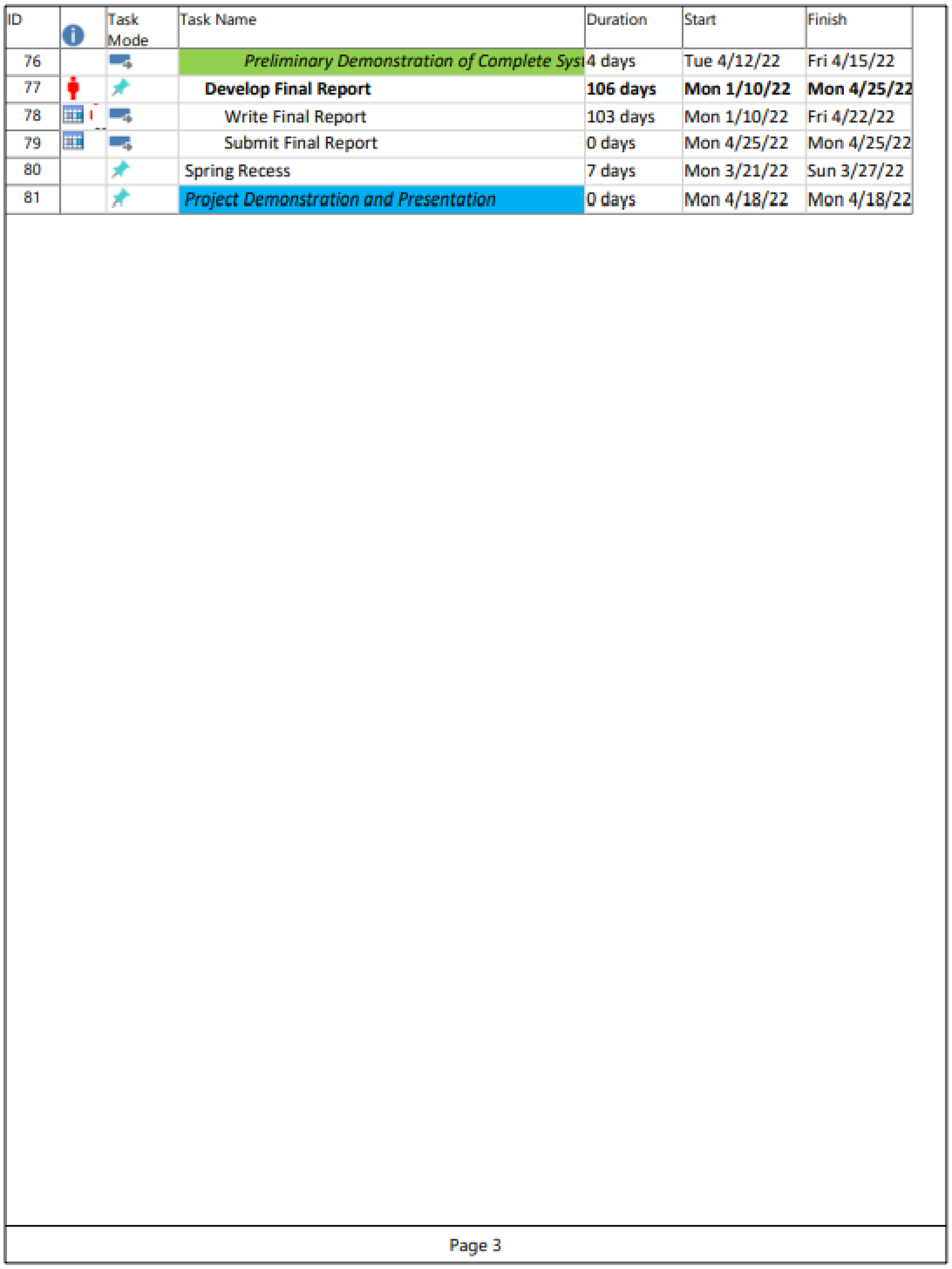
ID	ⓘ	Task Mode	Task Name	Duration	Start	Finish
1		📌	**SDP2 Implementation 2022**	**453 days?**	**Mon 1/10/22**	**Fri 4/7/23**
2	🧍	📌	Revise Gantt Chart	14 days	**Mon 1/10/22**	Sun 1/23/22
3	🧍	📌	**Implement Project Design**	**89 days?**	**Mon 1/10/22**	**Fri 4/8/22**
4		📌	**Hardware Implementation**	**42 days?**	**Mon 1/10/22**	**Sun 2/20/22**
5		📌	**Breadboard Components**	**22 days**	**Mon 1/10/22**	**Mon 1/31/22**
6	📅	➡	Microcontroller (ESP32)	12 days	Mon 1/10/22	Fri 1/21/22
7	📅	➡	Power (Battery)	12 days	Mon 1/10/22	Fri 1/21/22
8	📅	➡	Power (solar)	22 days	Mon 1/10/22	Mon 1/31/22
9	🧍	📌	Sensor (Temperature)	8 days	Thu 1/20/22	Thu 1/27/22
10	🧍	📌	Sensor (Nutrients/NPK)	8 days	Thu 1/20/22	Thu 1/27/22
11	📅	➡	Sensors (Moisture)	2 days	Thu 1/20/22	Fri 1/21/22
12		📌	Initial Parts Request	13 days	Thu 1/20/22	Tue 2/1/22
13	🧍	📌	**Layout and Generate PCB(s)**	**15 days**	**Fri 1/21/22**	**Fri 2/4/22**
14	📅	➡	Charging Circuit for Battery	2 days	Thu 2/3/22	Fri 2/4/22
15	📅	➡	Voltage Regulator	2 days	Thu 2/3/22	Fri 2/4/22
16	📅	➡	Moisture Sensor	14 days	Sat 1/22/22	Fri 2/4/22
17	🧍	📌	Nutrient Sensor	14 days	Sat 1/22/22	Fri 2/4/22
18	📅	➡	Temperature Sensor	7 days	Sat 1/29/22	Fri 2/4/22
19	🧍	📌	Microcontroller	14 days	Sat 1/22/22	Fri 2/4/22
20	🧍	📌	Preliminary drawings of probe housing	12 days	Mon 1/24/22	Fri 2/4/22
21		📌	Final Parts Request	40 days	Fri 1/21/22	Tue 3/1/22
22	🧍	📌	**Assemble Hardware**	**14 days**	**Fri 2/4/22**	**Thu 2/17/22**
23	🧍	➡	Assemble Power PCB/Regulator/Solar Pai	14 days	Sat 2/5/22	Fri 2/18/22
24	🧍	➡	Assemble all (3) sensor boards	14 days	Sat 2/5/22	Fri 2/18/22
25	🧍	➡	Print/install electrodes for sensor probes	14 days	Fri 2/4/22	Thu 2/17/22
26	🧍	➡	Install microprocessor onto the printed board	14 days	Sat 2/5/22	Fri 2/18/22
27	🧍	📌	Begin 3d printing probe housing	5 days	Sat 2/5/22	Wed 2/9/22
28	🧍	📌	**Test Hardware**	**5 days**	**Thu 2/17/22**	**Mon 2/21/22**
29	📅	➡	Test operation of Power PCB	2 days	Sat 2/19/22	Sun 2/20/22
30	📅	➡	Test operation of Solar Panel and Chargin	3 days	Sat 2/19/22	Mon 2/21/22
31	📅	➡	Test operation of the sensors PCBs	3 days	Sat 2/19/22	Mon 2/21/22
32	📅	➡	Test microcontroller on PCB with power	3 days	Sat 2/19/22	Mon 2/21/22
33	🧍	📌	**Revise Hardware**	**9 days**	**Thu 2/17/22**	**Fri 2/25/22**
34	📅 🧍	➡	Active revison of Power PCB and charging during testing	7 days	Sun 2/13/22	Sat 2/19/22
35	📅 🧍	➡	Active revision of sensor PCBs and electrodes	7 days	Mon 2/14/22	Sun 2/20/22
36	📅 🧍	➡	Revision of all hardware after feedback from demonstration	7 days	Sat 2/19/22	Fri 2/25/22
37	📅	➡	*MIDTERM: Demonstrate Hardware Subsyste*	4 days	Tue 2/22/22	Fri 2/25/22
38		📌	SDC & FA Hardware Approval	4 days	Tue 2/22/22	Fri 2/25/22
39		📌	**Software Implementation**	**42 days**	**Mon 1/10/22**	**Sun 2/20/22**

Page 1



ID	ℹ	Task Mode	Task Name	Duration	Start	Finish
40		▬	**Develop Software**	**28 days**	**Mon 1/10/22**	**Sun 2/6/22**
41	🔴	▬	Firmware (Microprocessor)	28 days	Mon 1/10/22	Sun 2/6/22
42	🔴	▬	Database Configuration	28 days	Mon 1/10/22	Sun 2/6/22
43	🔴	▬	PHP Script	28 days	Mon 1/10/22	Sun 2/6/22
44	🔴	▬	Phone App Development	28 days	Mon 1/10/22	Sun 2/6/22
45	🔴	📌	**Test Software**	**11 days**	**Mon 2/7/22**	**Thu 2/17/22**
46	▦	▬	Test Firmware	28 days	Fri 1/21/22	Thu 2/17/22
47	▦	▬	Test Database and Data Transfer	28 days	Fri 1/21/22	Thu 2/17/22
48	▦	▬	Test Phone App	28 days	Fri 1/21/22	Thu 2/17/22
49	🔴	📌	**Revise Software**	**5 days**	**Thu 2/17/22**	**Mon 2/21/22**
50	▦	▬	Revision of firmware during testing	4 days	Fri 2/18/22	Mon 2/21/22
51	▦	▬	Revision of data transfer during testing	4 days	Fri 2/18/22	Mon 2/21/22
52	▦	▬	Revision of user interface during app test	4 days	Fri 2/18/22	Mon 2/21/22
53	▦ 🔴	▬	Full software revision post midterm demonstration	5 days	Thu 2/17/22	Mon 2/21/22
54	▦	▬	*MIDTERM: Demonstrate Software Subsyster*	5 days	Mon 2/21/22	Fri 2/25/22
55		📌	SDC & FA Software Approval	5 days	Mon 2/21/22	Fri 2/25/22
56	🔴	▬	**System Integration**	**49 days**	**Sat 2/26/22**	**Fri 4/15/22**
57	🔴	▬	**Assemble Complete System Integration**	**14 days**	**Sat 2/26/22**	**Fri 3/11/22**
58	🔴	▬	Sensors and Microprocessor Integration	14 days	Sat 2/26/22	Fri 3/11/22
59	🔴	▬	System Power Integration	1 day	Sat 2/26/22	Sat 2/26/22
60	🔴	▬	Solar Panel Integration	14 days	Sat 2/26/22	Fri 3/11/22
61	🔴	▬	Phone App Integration	14 days	Sat 2/26/22	Fri 3/11/22
62	🔴	📌	Final assembly of printed housing parts	14 days	Sat 2/26/22	Fri 3/11/22
63	🔴	📌	Wire harness creation between subsyster	1 day	Fri 3/11/22	Fri 3/11/22
64	🔴	📌	Installation of electronics into the probe body	1 day	Fri 3/11/22	Fri 3/11/22
65	🔴	▬	**Test Complete System Integration**	**7 days**	**Sat 3/12/22**	**Fri 3/18/22**
66	🔴	▬	Test sensor data transfer to processor	7 days	Sat 3/12/22	Fri 3/18/22
67	🔴	▬	Test power quality for supply side	7 days	Sat 3/12/22	Fri 3/18/22
68	🔴	▬	Verify and test solar output to battery	7 days	Sat 3/12/22	Fri 3/18/22
69	🔴	▬	Test data transfer to database and then to phone app	7 days	Sat 3/12/22	Fri 3/18/22
70	🔴	📌	**Revise Complete System Integration**	**24 days**	**Sat 3/19/22**	**Mon 4/11/22**
71	🔴	▬	Revise sensor calibration based on integration testing	24 days	Sat 3/19/22	Mon 4/11/22
72	🔴	▬	Verify and adjust solar panels/power/regulation	24 days	Sat 3/19/22	Mon 4/11/22
73	🔴	▬	Verify and revise communication between sensors and microprocessor	24 days	Sat 3/19/22	Mon 4/11/22
74	🔴	▬	Revise communication of data between microprocessor and phone app	24 days	Sat 3/19/22	Mon 4/11/22
75	🔴	📌	Revise phone app and interface based on usability	24 days	Sat 3/19/22	Mon 4/11/22

Page 2



ID		Task Mode	Task Name	Duration	Start	Finish
76			*Preliminary Demonstration of Complete Sys*	4 days	Tue 4/12/22	Fri 4/15/22
77			**Develop Final Report**	**106 days**	**Mon 1/10/22**	**Mon 4/25/22**
78			Write Final Report	103 days	Mon 1/10/22	Fri 4/22/22
79			Submit Final Report	0 days	Mon 4/25/22	Mon 4/25/22
80			Spring Recess	7 days	Mon 3/21/22	Sun 3/27/22
81			*Project Demonstration and Presentation*	0 days	Mon 4/18/22	Mon 4/18/22

Page 3

X. CONCLUSIONS AND RECOMMENDATIONS

In conclusion, the DeMeter soil monitor project was a success. The microcontroller, power, sensor, and phone application subsystems were all successfully integrated, and a finished prototype was demonstrated. The prototype included several working sensors including temperature, moisture, pH, and RGB. While the accuracy of the PH sensor varied, the temperature, moisture, and RGB all accurately returned data and achieved the design goals. With further filtering, a greater budget, and improved electrodes, all sensors would be able to better sense soil conditions. During the final demonstration, the microcontroller system showed that accurate real time data could be sampled from 3 analog sources (temperature, moisture, pH) and 1 digital source (RGB). This data was posted (HTTP) wirelessly to a nearby laptop using the built-in WiFi module. Upon receiving the HTTP post request, the laptop server inserted the data posted into a MySQL database. The android phone application was successfully retrieving the data from the MySQL database and displaying it on a chart for the user. The power subsystem showed a lithium-ion battery charging circuit with overcharge protection and showed that the battery could accurately power the rest of the system while being charged.

XI. REFERENCES

Artigas, J., Beltran, A., Alonso, J., Bartrolí, J., Jiménez, C., Muñoz, J., Mas, R. and Domínguez, C. “APPLICATION OF ISFET BASED SENSORS TO SOIL ANALYSIS.”, 2001. https://www.ishs.org/ishs-article/562_33

Bhanage, G. “AMSDU vs AMDPU: A Brief Tutorial on WiFi Aggregation Support.”, 2017.

Hocker. “Temperature Compensated Soil Moisture Sensor.”, Onset Computer Corp., PATENT: US5430384A., 1995. https://patentimages.storage.googleapis.com/af/20/f9/6eb2d0b8c44c20/US5430384.pdf

Jameel, Furqan and Hassan, Syed Ali, Eds., “*Wireless-Powered Backscatter Communications for Internet of Things*.” Cham: Springer International Publishing, 2021. https://ebooks.ohiolink.edu/content/0d7e2274-6500-11eb-be71-0a9b31268bf5

Li, P. “Ion-Selective Electrode.” http://www.sfu.ca/chemistry/groups/Li/chem215/selective.PDF

Liu, J. “Automatic Sunlight Tracking Device.”, PATENT: AU2011235479B2., 2011. https://patentimages.storage.googleapis.com/df/99/03/f657ecb7558bef/AU2011235479B2.pdf

Mansergh et al. “Extensible, multimodal sensor fusion platform for remote, proximal terrain sensing.”, Teralytic Inc., PATENT: US10866208B2., 2020. https://patentimages.storage.googleapis.com/ce/bf/71/bfe875aef6e4a0/US10866208.pdf

Mobilian, J. “Axiom shares 2021 gardening insights survey results.”, 2020. https://www.nurserymag.com/article/axiom-releases-2021-gardening-insights-survey/

Petchjatuporn, P., Ngamkham, W., Khaehintung, N., Sirisuk, P. and Kiranon, W. "A Solar-powered Battery Charger with Neural Network Maximum Power Point Tracking Implemented on a Low-Cost PIC-microcontroller," *2005 International Conference on Power Electronics and Drives Systems*, 2005, pp. 507-510, doi: 10.1109/PEDS.2005.1619739. https://ieeexplore-ieee-org.ezproxy.uakron.edu:2443/document/1619739

Poghossian, A., Berndsen, L., & Lüth, H., Schoening, M. “Novel concept for flow-rate and flow-direction determination by means of pH-sensitive ISFETs.”, Proc SPIE. 4560. 19-27. 10.1117/12.443050. 2001. https://www.researchgate.net/figure/Structure-of-the-ISFET-RE-reference-electrode-V-G-gate-voltage-V-DS-drain-source_fig1_234987070

“Solar Photovoltaic System Design Basics.”, Office of Energy Efficiency & Renewable Energy. https://www.energy.gov/eere/solar/solar-photovoltaic-system-design-basics

“Sustainability in Agriculture.”, 2021. https://www.fb.org/land/sustainability

“Testing Your Soil Why and How to Take a Soil-Test Sample.”, 1997. https://www.nrcs.usda.gov/Internet/FSE_DOCUMENTS/nrcs142p2_037208.pdf

“555 Oscillator Tutorial - The Astable Multivibrator.” *Basic Electronics Tutorials*, ElectronicsTutorials, 14 Apr. 2021, https://www.electronics-tutorials.ws/waveforms/555_oscillator.html.

Beaudette, D. E. “Natural Resources Conservation Service.” *Soil Color | NRCS Wisconsin*, USDA-NRCS, https://www.nrcs.usda.gov/wps/portal/nrcs/detail/wi/soils/?cid=NRCSEPRD1370419.

“TCS3200 Colour Sensor Module.” *Elecrow*, https://www.elecrow.com/wiki/index.php?title=TCS3200_Colour_Sensor_Module.

Love, Serena. (2017). Field Methods for the Analysis of Mud Brick Architecture. Journal of Field Archaeology. 42. 1-13. 10.1080/00934690.2017.1345222. https://www.researchgate.net/publication/318350364_Field_Methods_for_the_Analysis_of_Mud_Brick_Architecture/citation/download

Hrisko, J. (2021, April 7). Capacitive soil moisture sensor calibration with Arduino. Maker Portal. Retrieved April 25, 2022, from https://makersportal.com/blog/2020/5/26/capacitive-soil-moisture-calibration-with-arduino

de Caritat, P. (n.d.). Legend for soil colour maps. colours from Munsell™ soil ... Research Gate. Retrieved April 25, 2022, from https://www.researchgate.net/figure/Legend-for-soil-colour-maps-Colours-from-Munsell-Soil-Color-Charts-Munsell-Color_fig18_291831041

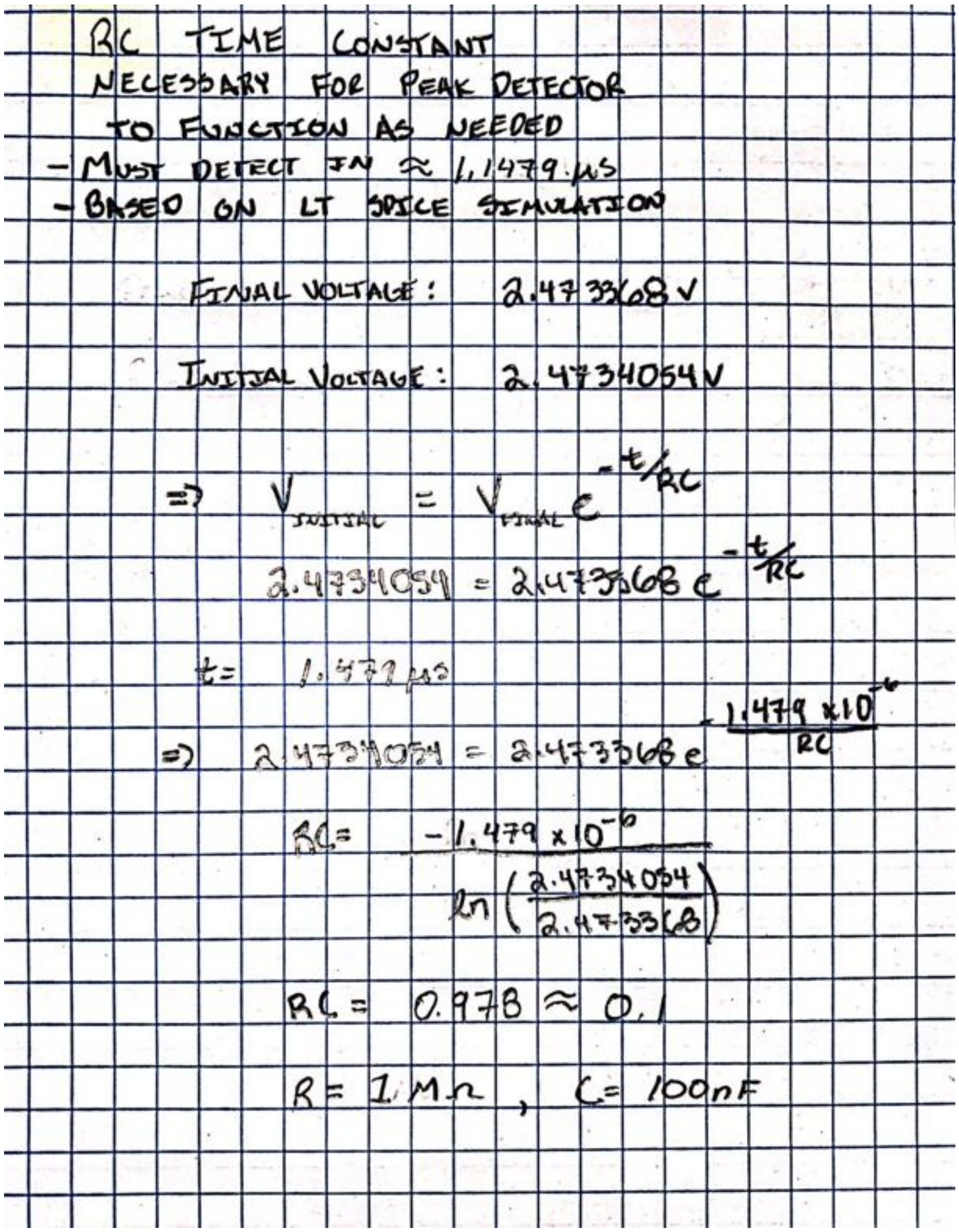
BT, R. (1970, January 1). Munsell to RGB conversion tables. Retrieved April 25, 2022, from http://munsell-to-rgb.blogspot.com/2012/01/munsell-soil-color-space-is-commonly.html

NAPA Master Gardener Column. (n.d.). The Colors of Soil. ANR Blogs. Retrieved April 25, 2022, from https://ucanr.edu/blogs/blogcore/postdetail.cfm?postnum=20112

Texas Instruments. (2013, April). AN-1852 Designing With pH Electrodes. Texas Instruments Application Report. Retrieved April 25, 2022, from https://www.ti.com/lit/an/snoa529a/snoa529a.pdf

XII. APPENDENDICES

Code, Simulations, Calculations, and Testing



RC TIME CONSTANT
NECESSARY FOR PEAK DETECTOR
TO FUNCTION AS NEEDED
- MUST DETECT IN ≈ 1,1479.µs
- BASED ON LT SPICE SIMULATION

FINAL VOLTAGE: 2.473368 V

INITIAL VOLTAGE: 2.4734054 V

$$\Rightarrow V_{INITIAL} = V_{FINAL}\, e^{-t/RC}$$

$$2.4734054 = 2.473368\, e^{-t/RC}$$

$t = 1.479\ \mu s$

$$\Rightarrow 2.4734054 = 2.473368\, e^{-\frac{1.479\times10^{-6}}{RC}}$$

$$RC = \frac{-1.479\times10^{-6}}{\ln\left(\frac{2.4734054}{2.473368}\right)}$$

$$RC = 0.978 \approx 0.1$$

$$R = 1\ M\Omega, \quad C = 100\ nF$$

Fig 1: RC time constant calculation

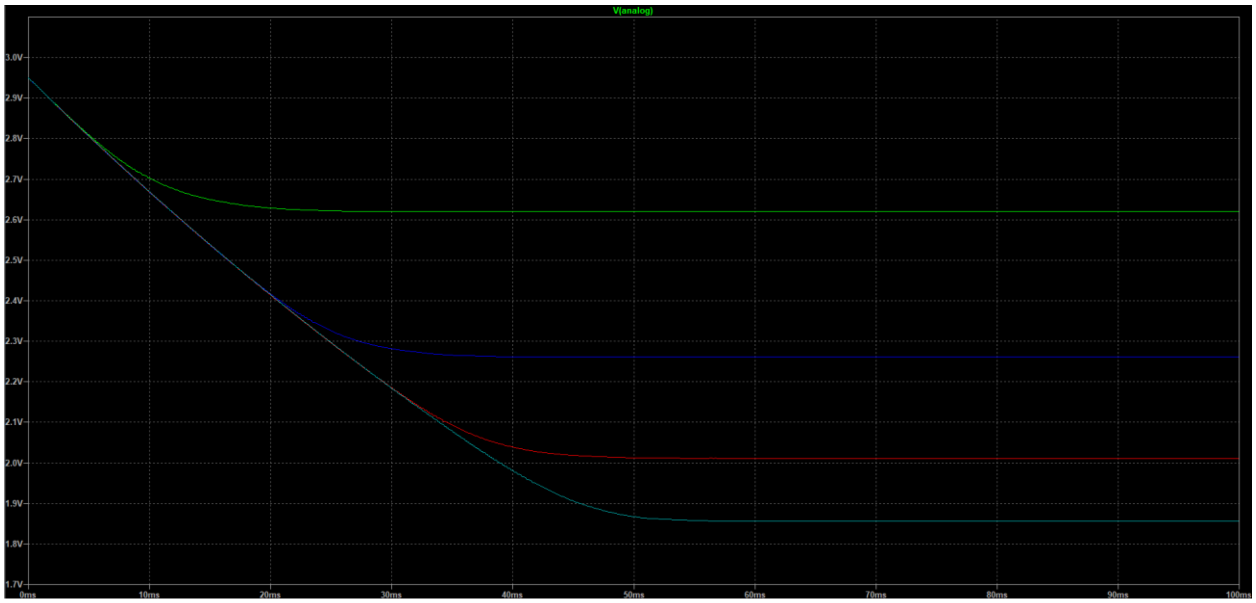


Fig 2: LTspice simulation of varying soil moisture content for sensor design.

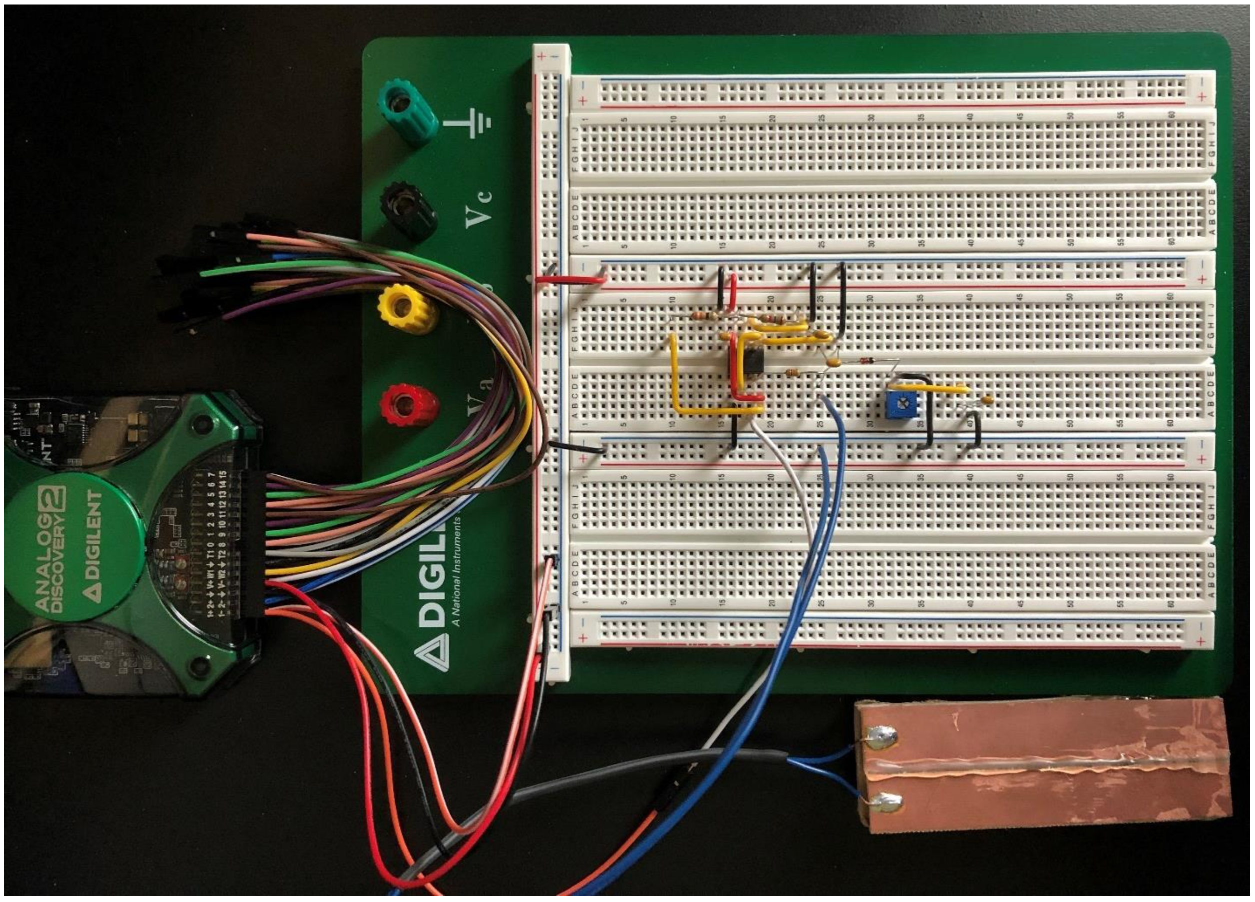
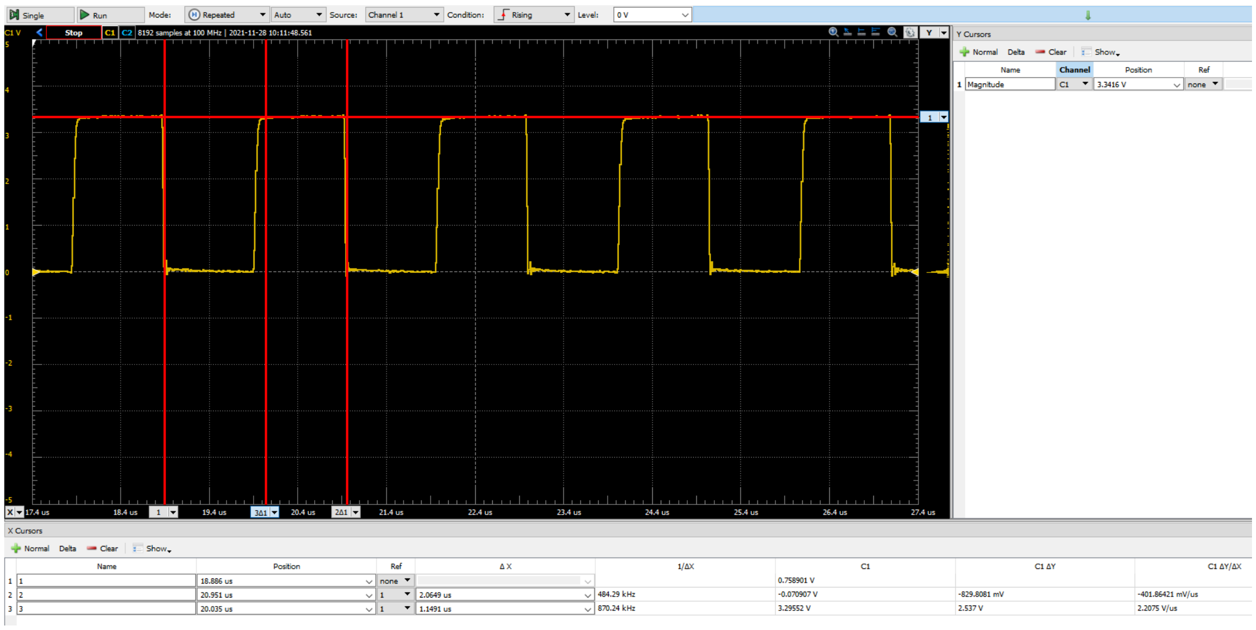


Fig 3: Breadboarded capacitive soil moisture sensor.


Single | Run | Mode: Repeated | Auto | Source: Channel 1 | Condition: Rising | Level: 0 V

Stop | C1 | C2 | 8192 samples at 100 MHz | 2021-11-28 10:11:48.561

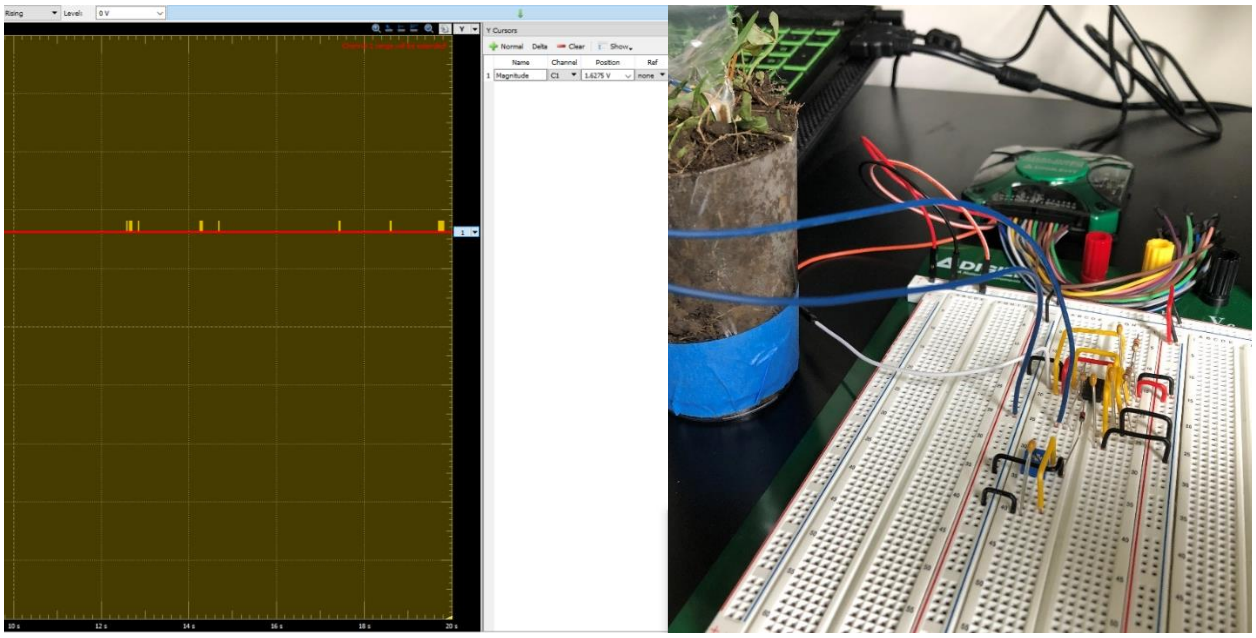
Y Cursors

	Name	Channel	Position	Ref
1	Magnitude	C1	3.3416 V	none

X Cursors

	Name	Position	Ref	Δ X	1/ΔX	C1	C1 ΔY	C1 ΔY/ΔX
1	1	18.886 us	none			0.758901 V		
2	2	20.951 us	1	2.0649 us	484.29 kHz	-0.070907 V	-829.8081 mV	-401.86421 mV/us
3	3	20.035 us	1	1.1491 us	870.24 kHz	3.29552 V	2.537 V	2.2075 V/us

Fig 4: Output of the TLC555 timer acquired from Analog Discovery II.


Rising | Level: 0 V

Y Cursors

	Name	Channel	Position	Ref
1	Magnitude	C1	1.6275 V	none

Fig 5: Analog Discovery II capture of soil sensor detecting moisture content in real soil. 1.6275V is approximately 150pF which is approximately 20% moisture content.

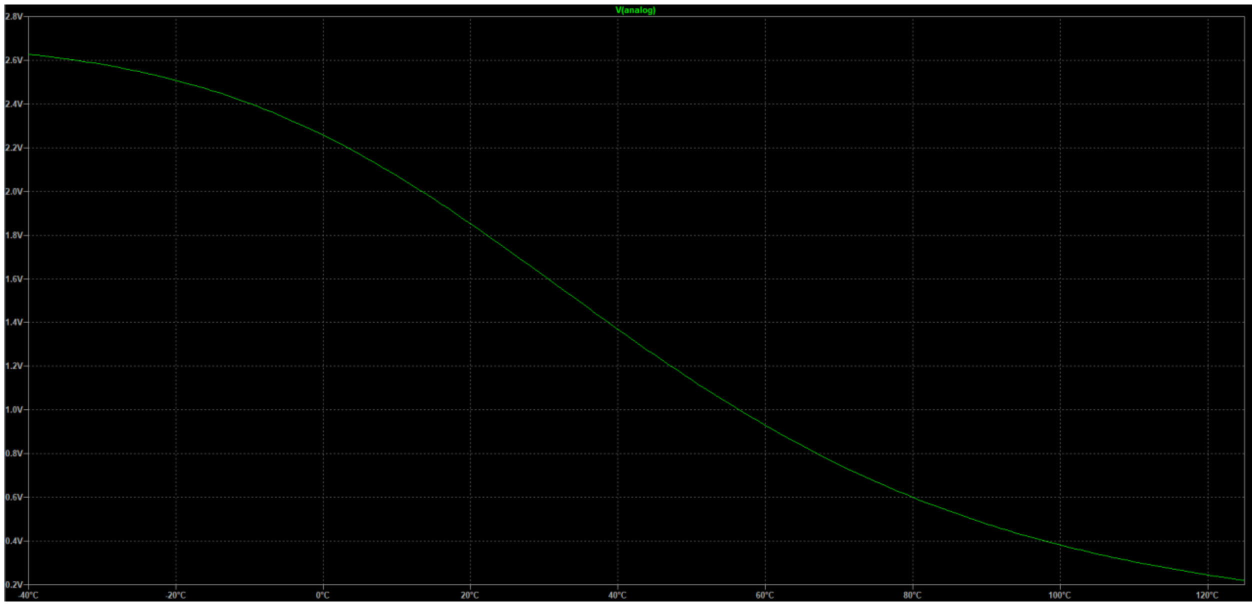
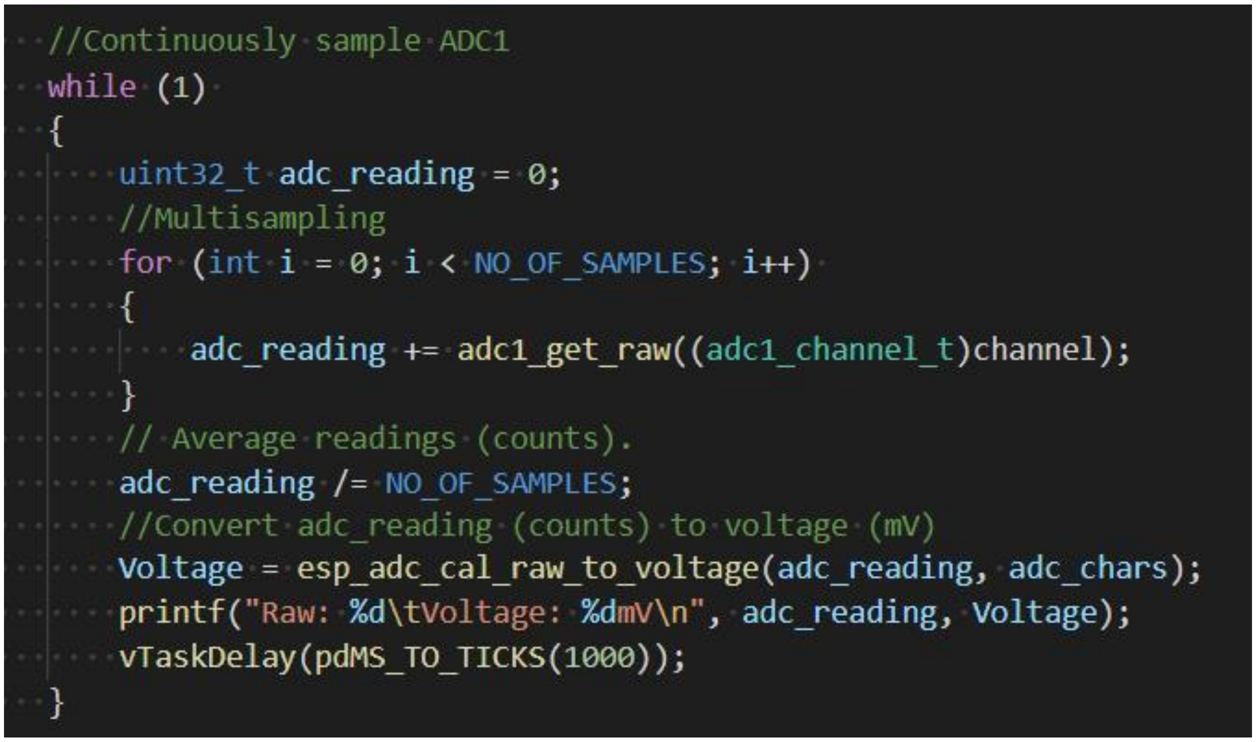
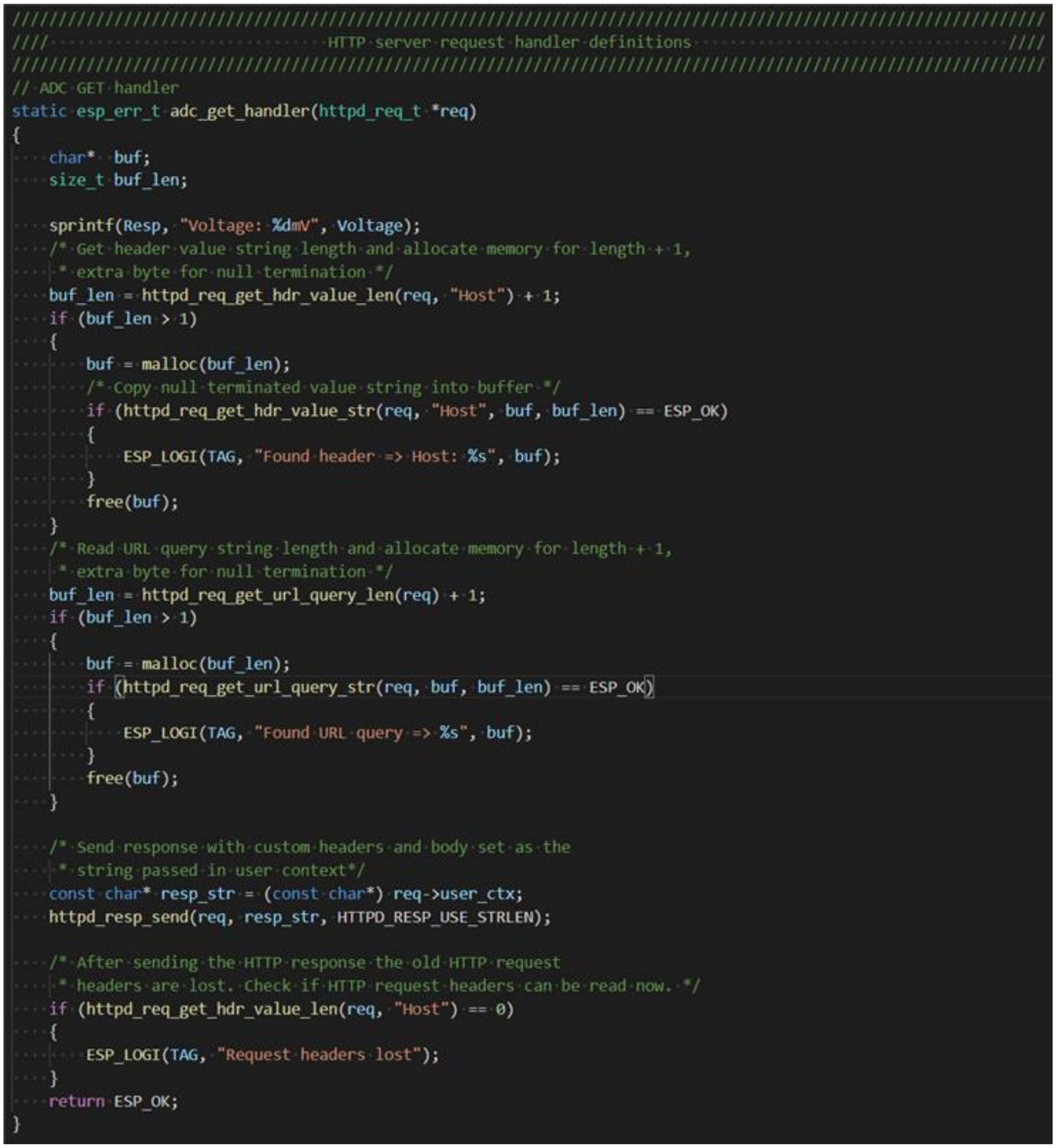


Fig 6: LTspice simulation of varying soil temperature for sensor design.



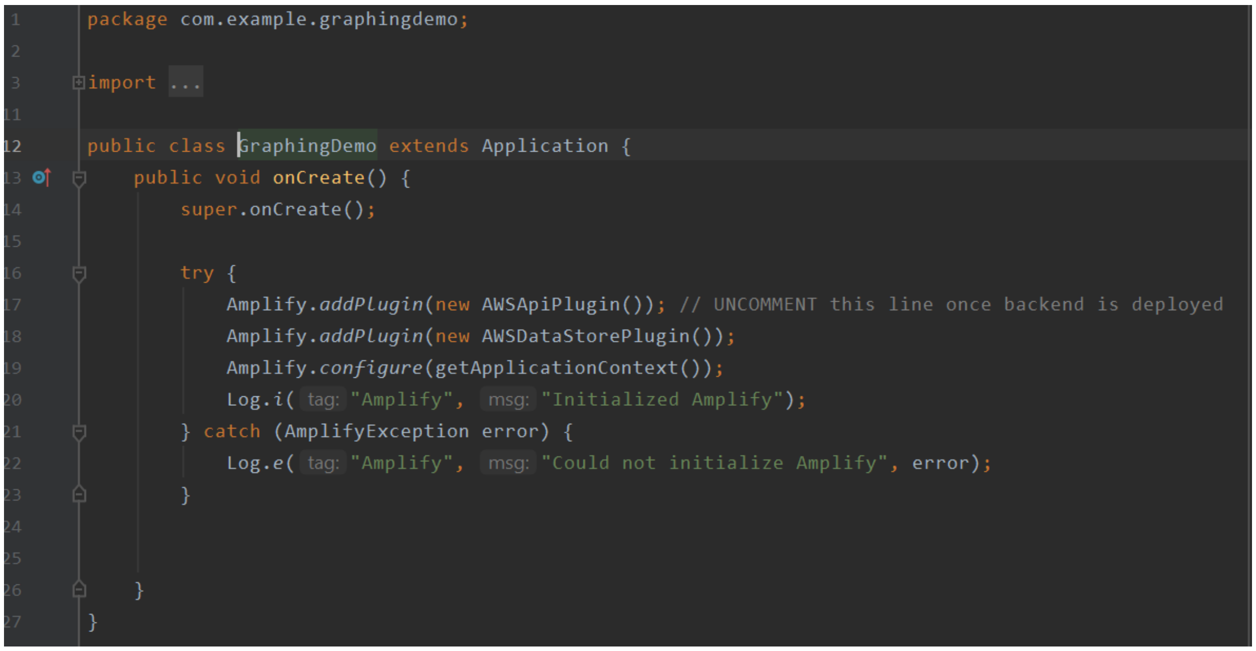
```
//Continuously sample ADC1
while (1)
{
    uint32_t adc_reading = 0;
    //Multisampling
    for (int i = 0; i < NO_OF_SAMPLES; i++)
    {
        adc_reading += adc1_get_raw((adc1_channel_t)channel);
    }
    // Average readings (counts).
    adc_reading /= NO_OF_SAMPLES;
    //Convert adc_reading (counts) to voltage (mV)
    Voltage = esp_adc_cal_raw_to_voltage(adc_reading, adc_chars);
    printf("Raw: %d\tVoltage: %dmV\n", adc_reading, Voltage);
    vTaskDelay(pdMS_TO_TICKS(1000));
}
```

Fig 7: Code for sampling and averaging of ADC



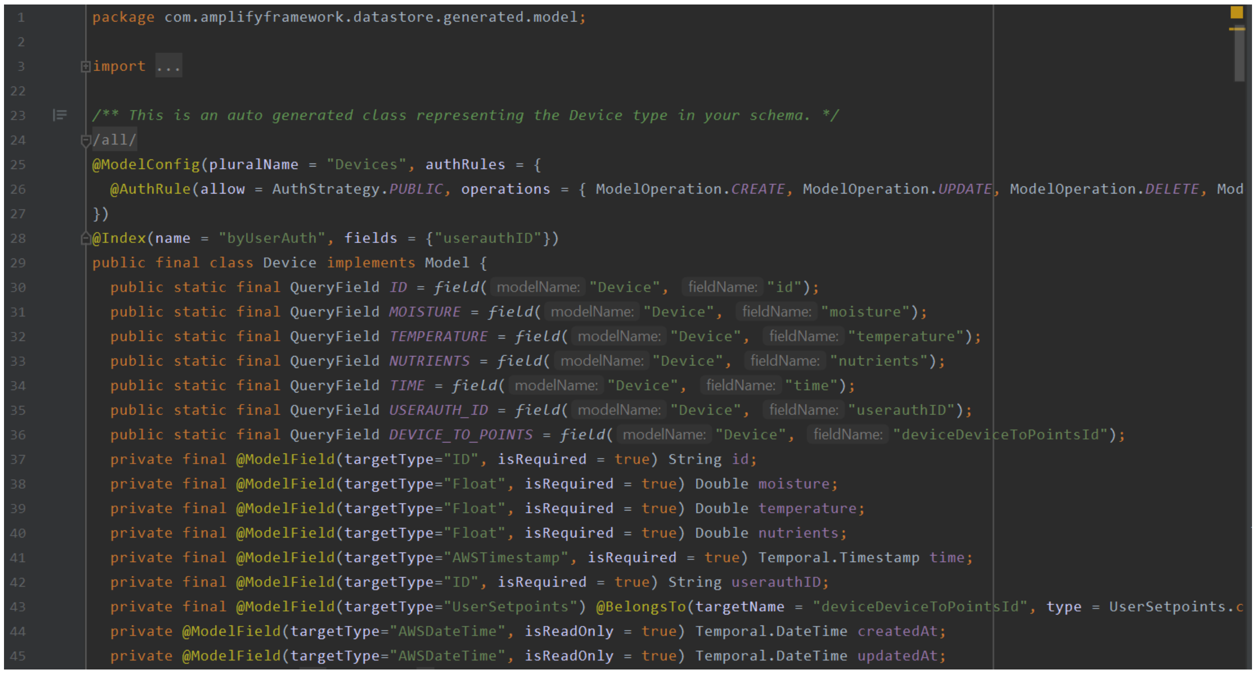
```
///////////////////////////////////////////////////////////////////////////////////////////////
////                          HTTP server request handler definitions                          ////
///////////////////////////////////////////////////////////////////////////////////////////////
// ADC GET handler
static esp_err_t adc_get_handler(httpd_req_t *req)
{
    char*  buf;
    size_t buf_len;

    sprintf(Resp, "Voltage: %dmV", Voltage);
    /* Get header value string length and allocate memory for length + 1,
     * extra byte for null termination */
    buf_len = httpd_req_get_hdr_value_len(req, "Host") + 1;
    if (buf_len > 1)
    {
        buf = malloc(buf_len);
        /* Copy null terminated value string into buffer */
        if (httpd_req_get_hdr_value_str(req, "Host", buf, buf_len) == ESP_OK)
        {
            ESP_LOGI(TAG, "Found header => Host: %s", buf);
        }
        free(buf);
    }
    /* Read URL query string length and allocate memory for length + 1,
     * extra byte for null termination */
    buf_len = httpd_req_get_url_query_len(req) + 1;
    if (buf_len > 1)
    {
        buf = malloc(buf_len);
        if (httpd_req_get_url_query_str(req, buf, buf_len) == ESP_OK)
        {
            ESP_LOGI(TAG, "Found URL query => %s", buf);
        }
        free(buf);
    }

    /* Send response with custom headers and body set as the
     * string passed in user context*/
    const char* resp_str = (const char*) req->user_ctx;
    httpd_resp_send(req, resp_str, HTTPD_RESP_USE_STRLEN);

    /* After sending the HTTP response the old HTTP request
     * headers are lost. Check if HTTP request headers can be read now. */
    if (httpd_req_get_hdr_value_len(req, "Host") == 0)
    {
        ESP_LOGI(TAG, "Request headers lost");
    }
    return ESP_OK;
}
```

Fig 8: HTTP GET handler for ADC reading (used for demo)



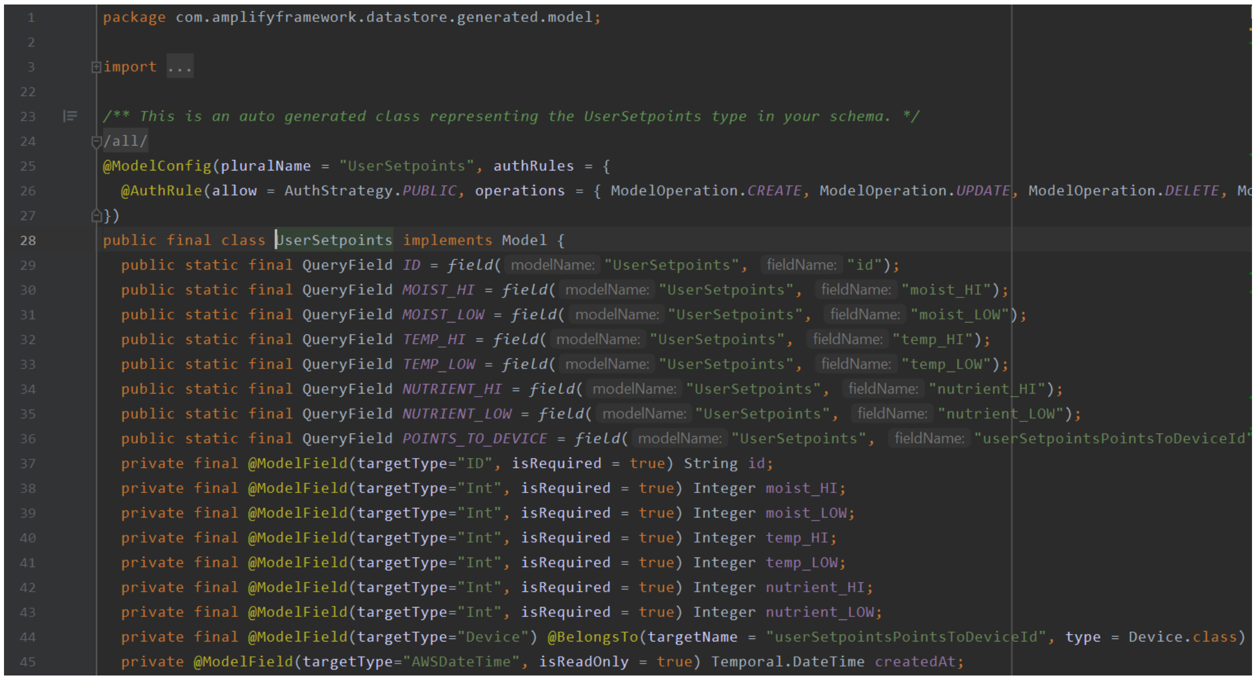
```java
package com.example.graphingdemo;

import ...

public class GraphingDemo extends Application {
    public void onCreate() {
        super.onCreate();

        try {
            Amplify.addPlugin(new AWSApiPlugin()); // UNCOMMENT this line once backend is deployed
            Amplify.addPlugin(new AWSDataStorePlugin());
            Amplify.configure(getApplicationContext());
            Log.i( tag: "Amplify",  msg: "Initialized Amplify");
        } catch (AmplifyException error) {
            Log.e( tag: "Amplify",  msg: "Could not initialize Amplify", error);
        }


    }
}
```

Fig 9: Initialize AWS Amplify in Android App



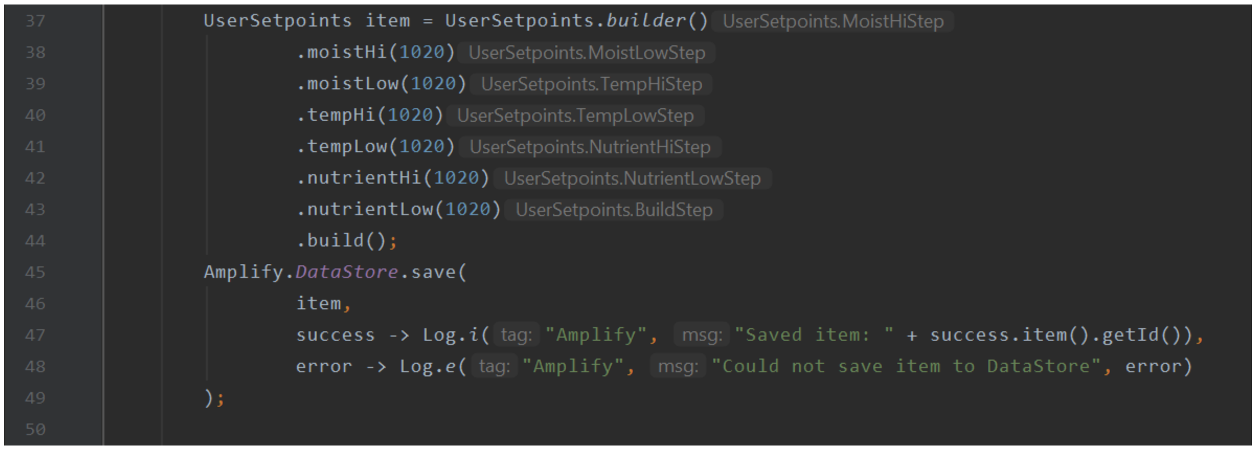
```java
package com.amplifyframework.datastore.generated.model;

import ...

/** This is an auto generated class representing the Device type in your schema. */
/all/
@ModelConfig(pluralName = "Devices", authRules = {
  @AuthRule(allow = AuthStrategy.PUBLIC, operations = { ModelOperation.CREATE, ModelOperation.UPDATE, ModelOperation.DELETE, Mod
})
@Index(name = "byUserAuth", fields = {"userauthID"})
public final class Device implements Model {
  public static final QueryField ID = field( modelName: "Device",  fieldName: "id");
  public static final QueryField MOISTURE = field( modelName: "Device",  fieldName: "moisture");
  public static final QueryField TEMPERATURE = field( modelName: "Device",  fieldName: "temperature");
  public static final QueryField NUTRIENTS = field( modelName: "Device",  fieldName: "nutrients");
  public static final QueryField TIME = field( modelName: "Device",  fieldName: "time");
  public static final QueryField USERAUTH_ID = field( modelName: "Device",  fieldName: "userauthID");
  public static final QueryField DEVICE_TO_POINTS = field( modelName: "Device",  fieldName: "deviceDeviceToPointsId");
  private final @ModelField(targetType="ID", isRequired = true) String id;
  private final @ModelField(targetType="Float", isRequired = true) Double moisture;
  private final @ModelField(targetType="Float", isRequired = true) Double temperature;
  private final @ModelField(targetType="Float", isRequired = true) Double nutrients;
  private final @ModelField(targetType="AWSTimestamp", isRequired = true) Temporal.Timestamp time;
  private final @ModelField(targetType="ID", isRequired = true) String userauthID;
  private final @ModelField(targetType="UserSetpoints") @BelongsTo(targetName = "deviceDeviceToPointsId", type = UserSetpoints.c
  private @ModelField(targetType="AWSDateTime", isReadOnly = true) Temporal.DateTime createdAt;
  private @ModelField(targetType="AWSDateTime", isReadOnly = true) Temporal.DateTime updatedAt;
```

Fig 10: Generated Device Table DataStore Class



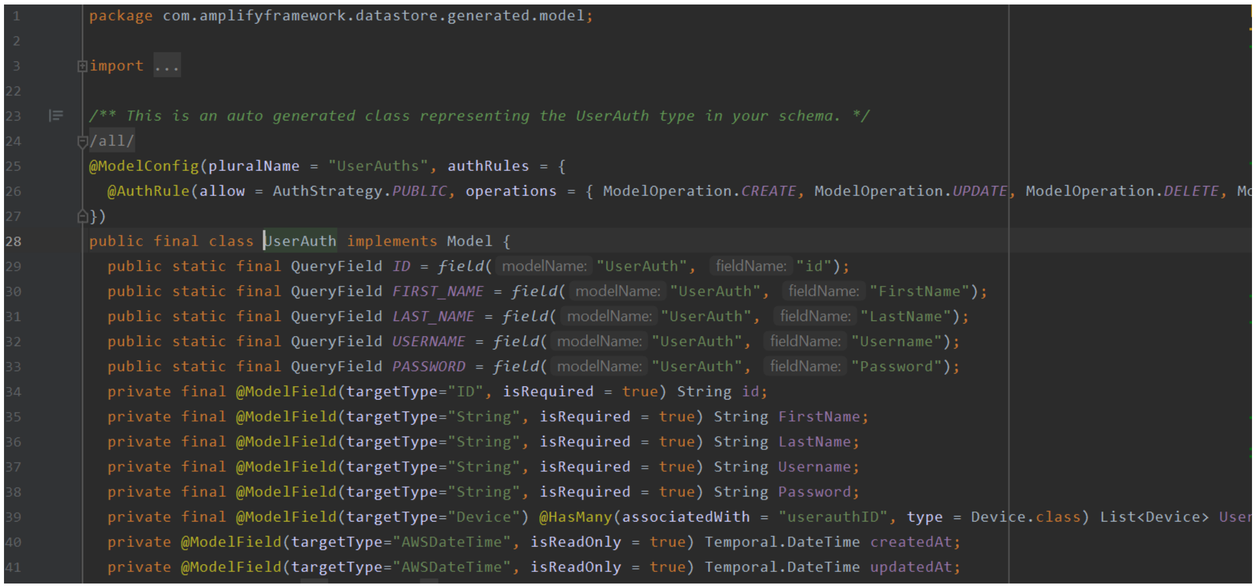
```java
package com.amplifyframework.datastore.generated.model;

import ...

/** This is an auto generated class representing the UserSetpoints type in your schema. */
/all/
@ModelConfig(pluralName = "UserSetpoints", authRules = {
  @AuthRule(allow = AuthStrategy.PUBLIC, operations = { ModelOperation.CREATE, ModelOperation.UPDATE, ModelOperation.DELETE, Mo
})
public final class UserSetpoints implements Model {
  public static final QueryField ID = field( modelName: "UserSetpoints",  fieldName: "id");
  public static final QueryField MOIST_HI = field( modelName: "UserSetpoints",  fieldName: "moist_HI");
  public static final QueryField MOIST_LOW = field( modelName: "UserSetpoints",  fieldName: "moist_LOW");
  public static final QueryField TEMP_HI = field( modelName: "UserSetpoints",  fieldName: "temp_HI");
  public static final QueryField TEMP_LOW = field( modelName: "UserSetpoints",  fieldName: "temp_LOW");
  public static final QueryField NUTRIENT_HI = field( modelName: "UserSetpoints",  fieldName: "nutrient_HI");
  public static final QueryField NUTRIENT_LOW = field( modelName: "UserSetpoints",  fieldName: "nutrient_LOW");
  public static final QueryField POINTS_TO_DEVICE = field( modelName: "UserSetpoints",  fieldName: "userSetpointsPointsToDeviceId"
  private final @ModelField(targetType="ID", isRequired = true) String id;
  private final @ModelField(targetType="Int", isRequired = true) Integer moist_HI;
  private final @ModelField(targetType="Int", isRequired = true) Integer moist_LOW;
  private final @ModelField(targetType="Int", isRequired = true) Integer temp_HI;
  private final @ModelField(targetType="Int", isRequired = true) Integer temp_LOW;
  private final @ModelField(targetType="Int", isRequired = true) Integer nutrient_HI;
  private final @ModelField(targetType="Int", isRequired = true) Integer nutrient_LOW;
  private final @ModelField(targetType="Device") @BelongsTo(targetName = "userSetpointsPointsToDeviceId", type = Device.class)
  private @ModelField(targetType="AWSDateTime", isReadOnly = true) Temporal.DateTime createdAt;
```

Fig 11: Generated UserSetpoints DataStore Element



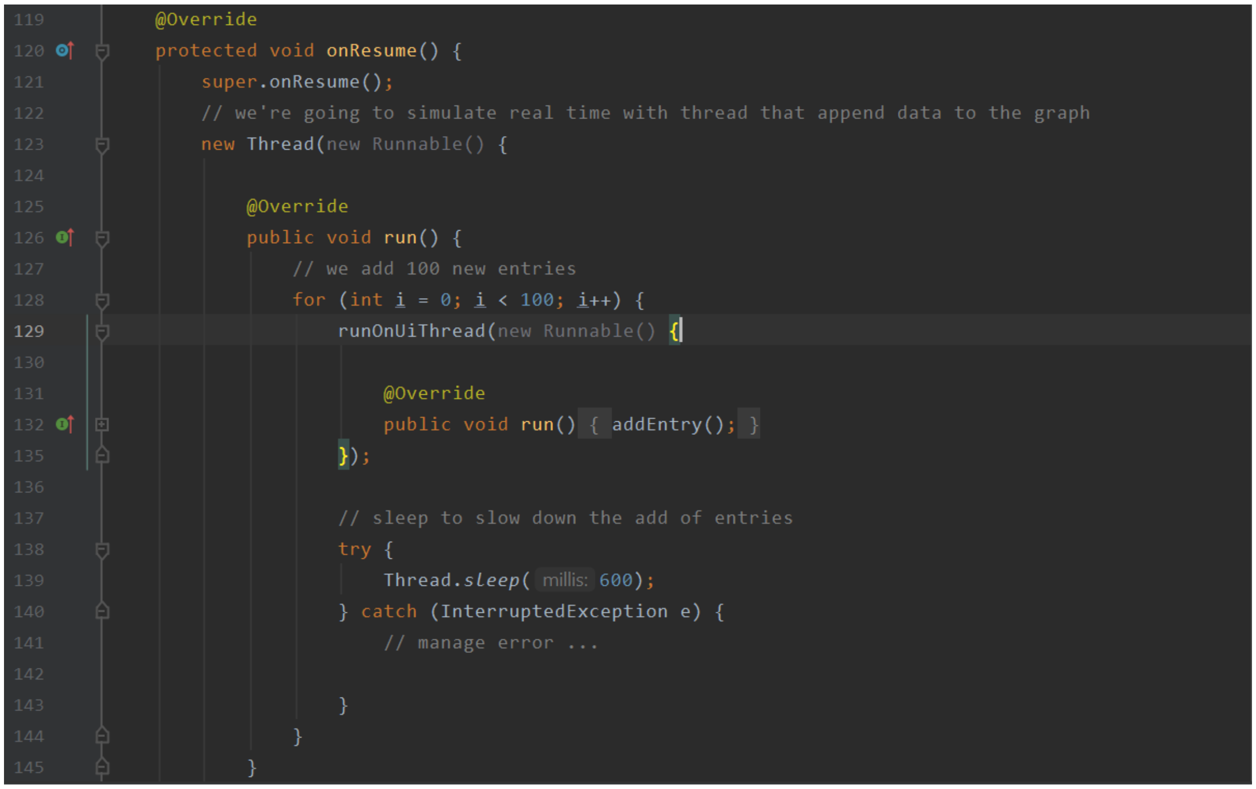
```java
        UserSetpoints item = UserSetpoints.builder() UserSetpoints.MoistHiStep
                .moistHi(1020) UserSetpoints.MoistLowStep
                .moistLow(1020) UserSetpoints.TempHiStep
                .tempHi(1020) UserSetpoints.TempLowStep
                .tempLow(1020) UserSetpoints.NutrientHiStep
                .nutrientHi(1020) UserSetpoints.NutrientLowStep
                .nutrientLow(1020) UserSetpoints.BuildStep
                .build();
        Amplify.DataStore.save(
                item,
                success -> Log.i( tag: "Amplify",  msg: "Saved item: " + success.item().getId()),
                error -> Log.e( tag: "Amplify",  msg: "Could not save item to DataStore", error)
        );
```

Fig 12: Code to Update UserSetpoints DataStore Element with Dummy Variables



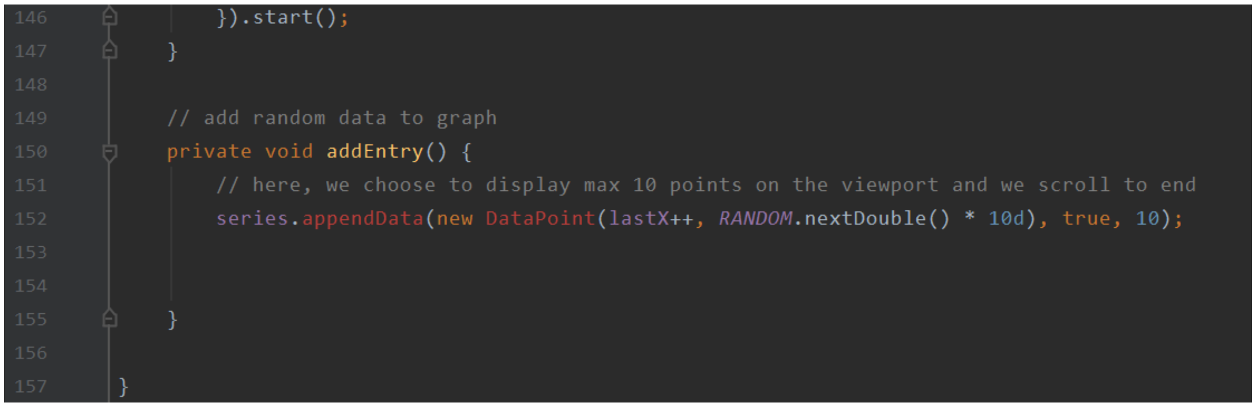
```java
package com.amplifyframework.datastore.generated.model;

import ...

/** This is an auto generated class representing the UserAuth type in your schema. */
/all/
@ModelConfig(pluralName = "UserAuths", authRules = {
  @AuthRule(allow = AuthStrategy.PUBLIC, operations = { ModelOperation.CREATE, ModelOperation.UPDATE, ModelOperation.DELETE, Mo
})
public final class UserAuth implements Model {
  public static final QueryField ID = field( modelName: "UserAuth",  fieldName: "id");
  public static final QueryField FIRST_NAME = field( modelName: "UserAuth",  fieldName: "FirstName");
  public static final QueryField LAST_NAME = field( modelName: "UserAuth",  fieldName: "LastName");
  public static final QueryField USERNAME = field( modelName: "UserAuth",  fieldName: "Username");
  public static final QueryField PASSWORD = field( modelName: "UserAuth",  fieldName: "Password");
  private final @ModelField(targetType="ID", isRequired = true) String id;
  private final @ModelField(targetType="String", isRequired = true) String FirstName;
  private final @ModelField(targetType="String", isRequired = true) String LastName;
  private final @ModelField(targetType="String", isRequired = true) String Username;
  private final @ModelField(targetType="String", isRequired = true) String Password;
  private final @ModelField(targetType="Device") @HasMany(associatedWith = "userauthID", type = Device.class) List<Device> User
  private @ModelField(targetType="AWSDateTime", isReadOnly = true) Temporal.DateTime createdAt;
  private @ModelField(targetType="AWSDateTime", isReadOnly = true) Temporal.DateTime updatedAt;
```

Fig 13: Generated UserAuth DataStore Class



```java
    @Override
    protected void onResume() {
        super.onResume();
        // we're going to simulate real time with thread that append data to the graph
        new Thread(new Runnable() {

            @Override
            public void run() {
                // we add 100 new entries
                for (int i = 0; i < 100; i++) {
                    runOnUiThread(new Runnable() {

                        @Override
                        public void run() { addEntry(); }
                    });

                    // sleep to slow down the add of entries
                    try {
                        Thread.sleep( millis: 600);
                    } catch (InterruptedException e) {
                        // manage error ...

                    }
                }
            }
```

Fig 14: Function to Graph Data Display in Real-Time



```
        }).start();
    }

    // add random data to graph
    private void addEntry() {
        // here, we choose to display max 10 points on the viewport and we scroll to end
        series.appendData(new DataPoint(lastX++, RANDOM.nextDouble() * 10d), true, 10);


    }

}
```

Fig 15: Function that Adds Random Values to Graph

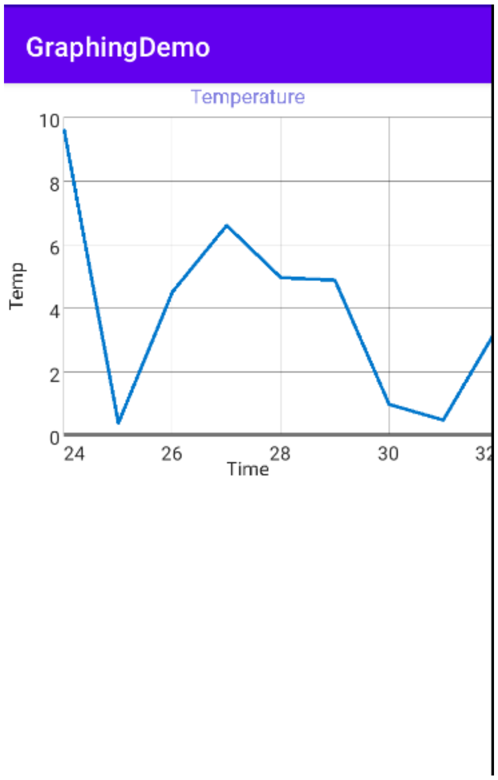
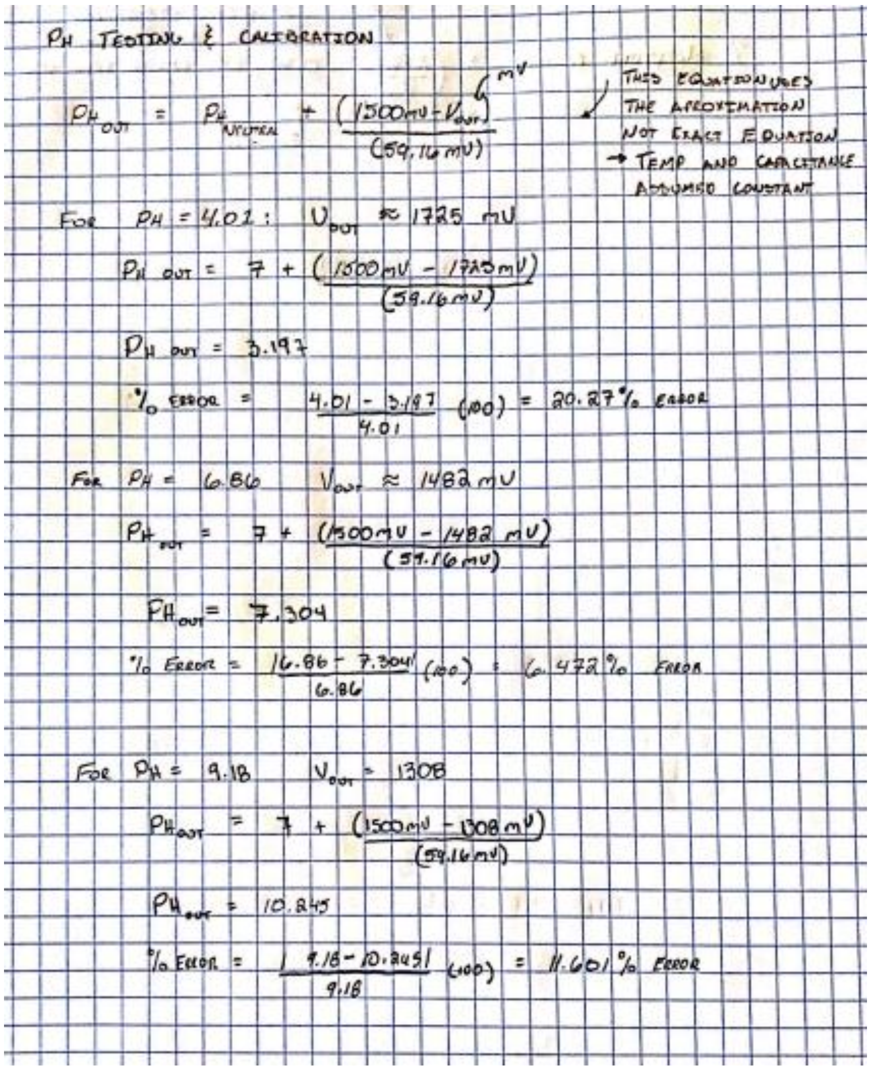


Fig 16: Resulting Graph from Fig 14 & Fig 15



PH TESTING & CALIBRATION

$$PH_{OUT} = PH_{NEUTRAL} + \frac{(1500mV - V_{OUT})}{(59.16mV)}$$

mV

THIS EQUATION USES THE APPROXIMATION NOT EXACT EQUATION

→ TEMP AND CAPACITANCE ASSUMED CONSTANT

FOR $PH = 4.01$: $V_{OUT} \approx 1725\ mV$

$$PH_{OUT} = 7 + \frac{(1500mV - 1725mV)}{(59.16mV)}$$

$$PH_{OUT} = 3.197$$

$$\%\ ERROR = \frac{4.01 - 3.197}{4.01}(100) = 20.27\%\ \text{ERROR}$$

FOR $PH = 6.86$ $V_{OUT} \approx 1482\ mV$

$$PH_{OUT} = 7 + \frac{(1500mV - 1482\ mV)}{(59.16mV)}$$

$$PH_{OUT} = 7.304$$

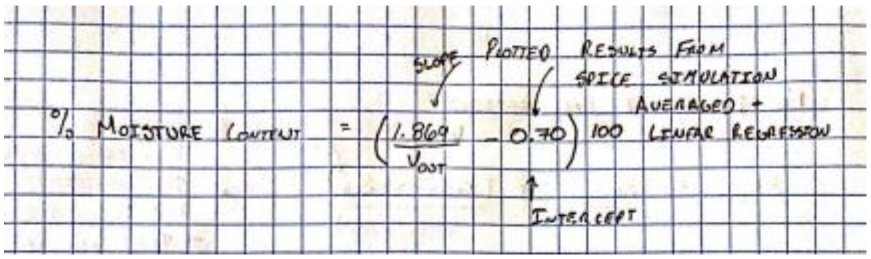
$$\%\ ERROR = \frac{|6.86 - 7.304|}{6.86}(100) = 6.472\%\ \text{ERROR}$$

FOR $PH = 9.18$ $V_{OUT} = 1308$

$$PH_{OUT} = 7 + \frac{(1500mV - 1308mV)}{(59.16mV)}$$

$$PH_{OUT} = 10.245$$

$$\%\ ERROR = \frac{|9.18 - 10.245|}{9.18}(100) = 11.601\%\ \text{ERROR}$$



$$\%\ MOISTURE\ CONTENT = \left(\frac{1.869}{V_{OUT}} - 0.70\right)100$$

SLOPE

INTERCEPT

PLOTTED RESULTS FROM SPICE SIMULATION AVERAGED + LINEAR REGRESSION

Fig 17: Calibration for Moisture and PH Sensors

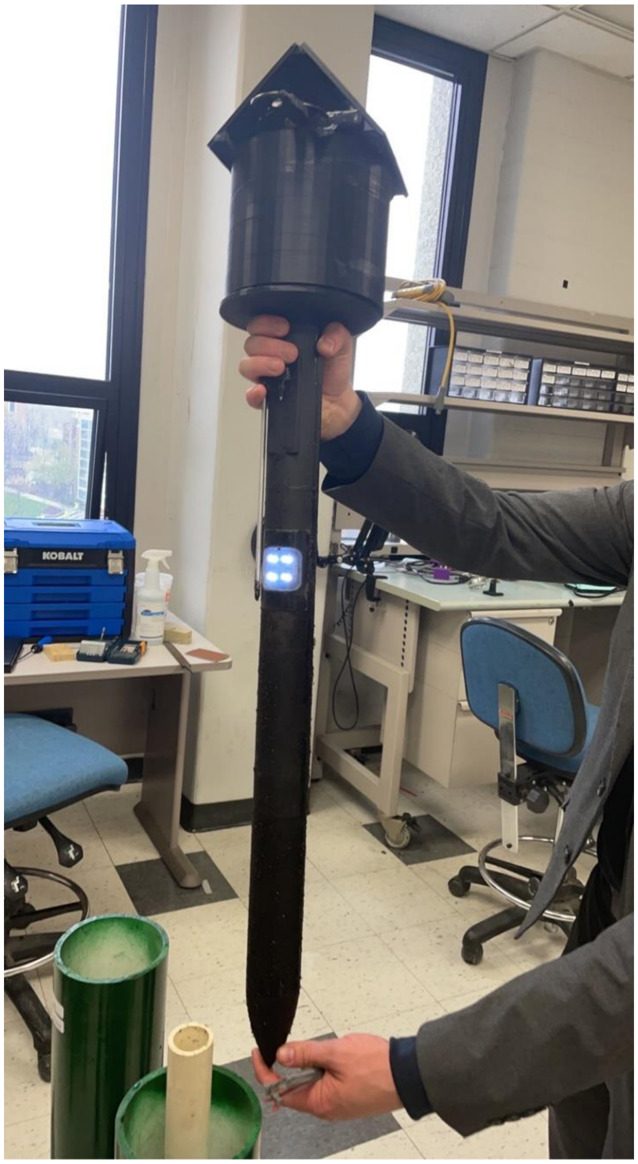


Fig 18: Completed Design With ABS and PLA+ Plastic Housing

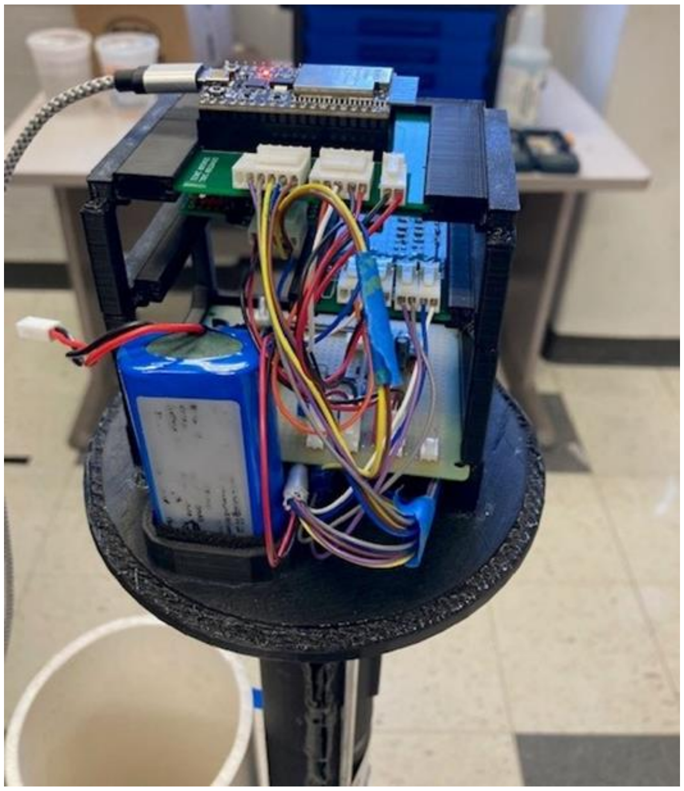


Fig 19: Internal Housing

Datasheets for Components

ESP32 Datasheet:
https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

OASIS MQTT Datasheet:
http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.pdf

ESP32-WROOM-32 Datasheet:
https://cdn.sparkfun.com/assets/learn_tutorials/8/0/4/esp32-wroom-32_datasheet_en.pdf

TCS3200 Datasheets:
https://components101.com/sites/default/files/component_datasheet/TCS3200-Color-Sensor-Module.pdf
https://ams.com/documents/20143/36005/TCS3200_DS000107_3-00.pdf/c53d702f-63aa-eda1-745f-d513aa4f535f
https://www.mouser.com/datasheet/2/321/28302-TCS3200-doc-370245.pdf

NTC Thermistor Datasheet:
https://www.vishay.com/docs/29049/ntcle100.pdf

TLC555 Timer Datasheet:
https://www.ti.com/lit/ds/symlink/tlc555.pdf?ts=1637984776404&ref_url=https%253A%252F%252Fwww.google.com%252F

LMP7721 Op-Amp Datasheet
https://www.ti.com/product/LMP7721