Spring 2021

Soil Sensor Network

Andrea Wyder
The University of Akron, alw179@uakron.edu

Ross Klonowski
The University of Akron, rak112@uakron.edu

Alexis Alves
The University of Akron, ara87@uakron.edu

Luke Farnsworth
The University of Akron, lmf78@uakron.edu

Soil Sensor Network

Senior Project Final Report

Design Team 7

Alexis Alves

Luke Farnsworth

Ross Klonowski

Andrea Wyder

Faculty Advisor: Dr. Bahrami

23 April 2021

TABLE OF CONTENTS

LIST OF FIGURES

List of Tables

ABSTRACT

Water management during crop irrigation is a problem for the agricultural industry. To help farmers better maintain water usage, a wireless soil sensor network comprised of a sensor pod and wireless communication has been designed and implemented. It was proven that the sensor pod can be installed 6-8 inches below the ground and communicate up to at least a 6km distance back to the gateway. The senor pod shells have a 2 mm thick shell to prevent the pod from shattering when coming into contact with the ground after being released from the planter, as calculated through the force of impact equations. The sensor pod contains a capacitive soil moisture sensor with an accuracy of 90% and a temperature sensor with an accuracy of ±0.2ºC. Lithium-ion batteries with a 2800 mA-H rating were chosen to ensure the sensor pods would be power-efficient in order to last an entire growing season. The sensor data is transmitted wirelessly through LoRaWAN communication using a RN2903 transceiver and a quarter wavelength, 3” monopole antenna. A Sentrius Laird gateway was used to collect and forward sensor pod data to the Senet dashboard. The Senet dashboard then forwarded the data to a web-based application that farmers can reference to check the status of their fields.

Keywords:

- Capacitive soil moisture sensor
- LoRaWAN communication
- RN2903 transceiver
- Senet gateway
- PIC24 microcontroller
- Mobile-friendly application

1 PROBLEM STATEMENT

Owners of large farms need a way to efficiently monitor water consumption and the health of their field. Traditional Wireless Sensor Networks that are aimed at collecting data from the soil use costly equipment. The new Wireless Sensor Network proposed will take advantage of advanced protocols and embedded systems available to aid in the process of growing and maintaining crops with technology that is more accessible to farmers.

1.1 NEED AA, LF, RK, AW

The agriculture industry experiences water shortages in different areas and at different times due to the mobility of water as a resource; the amount of water in a given area one day may not be the same amount of water there the next day. In addition to water scarcity, water cost is also a serious concern for farm owners. Because of this, there is a need for an easier and more reliable method for monitoring water consumption and soil composition of crops for the farming industry. Current methods of manually monitoring water distribution can lead to irrigation systems over watering, which wastes money, or under watering, which creates bad harvests. Many soil sensing systems that help eliminate water inefficiencies are networked together through physical wires, leading to difficult installations. There is a need for an off the shelf, easy-to-install sensor system that can be accomplished by using wireless soil sensors.

1.2 OBJECTIVE AA, RK, AW

The goal of this project is to create a low-cost, power-efficient software-sensor network that acquires soil data from sensors, contained in a single Sensor Pod, to aid in irrigation and crop management. The Sensor Pods can be “planted” with the crops throughout the field for easy installation. Data will be transferred using wireless communication and will then be stored in the

Cloud to be accessed through an app. This will allow farmers to check the history of their crops and forecast when the most water will need to be supplied to their field. The sensor system created will automatically update at predetermined intervals throughout the day to give updates on the status of the crop. These interval times of reading can be altered to conform to a farmer's needs and schedule preferences. These updates will be used to inform the farmer when crops are in need of water.

1.3 Soil Sensor Network Background AA, RK, AW

According to the United States Department of Agriculture, 80 to 90 percent of water in the United States is consumed by the agriculture industry [1]. To compensate for water management inefficiencies, an automated, sensor-based irrigation system is needed. Improvements in irrigation efficiency can go a long way in reducing water consumption globally for the agriculture industry. A scalable solution that includes soil sensors and automated irrigation is needed to achieve these desired changes. This system can be developed with a feedback loop relaying information from data acquisition sensors in a field back to a remote interface that would control the irrigation system, only activating the sprinklers located in water-depleted areas. The system can be tailored for the type of crop in the field in order to ensure accurate water management. The data collection will be stored and trended in the Cloud, which can then be accessed through a software application so farmers can better maintain their crops. The creation of a software-sensor network system is important in reducing agricultural water consumption.

The theory behind the Soil Sensor Network system is to "plant" a Sensor Pod into the ground to collect data useful for crop and water management. The sensors will be stored in a container that can be mounted on farm planting equipment to allow for automated planting. Once the Sensor Pods are planted, they are in position to communicate with a base station through LoRaWAN technology. To save energy, the Sensor Pod will take measurements only when triggered. Once triggered, the sensors will collect data and transmit it to the base station, which will then relay it to the Cloud. The web server in the Cloud will analyze the data over time and create trends that farmers can access from their device to be used to monitor the moisture levels and other valuable properties of their fields.

In order to acquire data from the system, a few sensors will be integrated, one of which will be a capacitive soil moisture sensor. To measure moisture levels in the soil, capacitive soil sensors use the discharge rate of the soil to calculate soil moisture content. Other sensors that will be researched and analyzed for potential use are electrochemical sensors, salinity sensors, and pH level sensors. These three types of sensors have the ability to measure nutrient and pH levels of the soil to tell farmers when and what kind of fertilizer needs to be added to the field. The sensors chosen to be used will be combined into the Sensor Pod. Each Sensor Pod's location can be recorded through a unique identification.

Although current automated irrigation designs have proven to be very effective in the field, they also have limited communication abilities. In general, implementations involve a microprocessor for computation and decision making, sensors for capturing data, and a network to connect the sensors to a base station. The entire network of Wireless Sensor Units (WSU) forms a Wireless Sensor Network (WSN). A variety of communication protocols have been used for the relay of data between sensors and the base station. One of the implementations of an automated irrigation system has relied upon Global System for Mobile Communication (GSM), an older form of telecommunications that is available in remote areas. General Packet Radio

Service (GPRS) is another form of communication that uses 2G and 3G cellular networks, which are also outdated [3].

In recent times, technologies such as LoRa have been suggested to be the backbone of the WSN. LoRaWAN, a low-power wide-area network (LPWAN), has become more popular in agriculture technology communication due to its low cost and long-range capabilities. LoRa technologies operate off of the unlicensed spectrum, allowing the costs of data communication to be relatively cheap compared to the currently used licensed spectrum (i.e.: 4G technologies) [4]. LoRa operates on the 915 MHz frequency in the U.S. with a physical range capability of 10 km in rural areas, making it a preferred method of communication for agricultural sensor networks [3]. Most sensors do not use wireless communication but rather rely on physical wires for power and communication back to a base station. This method restricts the measurement to local areas due to the sensors being physically connected. The WSN has recently been used for communication in agricultural sensory systems; an example of this is the patent by RainBird Corporation, which uses WSN to monitor and control irrigation [6]. This technology allows for a wider range of monitoring that can more accurately be attained by using LoRa as a means of communication to standalone sensors.

Automated soil sensing has previously been implemented in dry, remote locations, such as India, and has shown promising results. Ozawa et. al. patented an underground sensor design in February of 2020 to be used for fertigation. The sensor was attached to a discharge valve that would distribute minerals into the soil through salinization to prevent pollution and oversaturation of salt. The soil electrical conductivity (EC) sensor measured soil content and communicated back to a main controller through a wireless fertigation system [5]. In a paper published at Karpagam Academy of Higher Education, the results from their smart irrigation system yielded a significant water usage reduction. The traditional method they observed had used 174 liters of water per drip hole, whereas their system used just 14 liters per drip hole [3]. With almost 90% water savings, the proposed design by Ozawa et. al. was very effective and showed that under good conditions, the system can save water and help farmers efficiently manage crops.

In addition to a capacitive soil sensor, many other types of sensors can be used to collect data that is useful for crop growth management. Popular sensors include air humidity sensors, water level sensors, and soil moisture sensors. Similarly, the focus of the project will be on capturing soil moisture readings, as well as potential readings on pH levels. The two methods for measuring moisture in soil are soil water content and soil water tension [8]. Soil water content sensors measure the mass ratio of the soil of when it is wet to when it is dry. Soil water tension sensors measure the force it takes the roots to grow through the soil.

The three most commonly used sensors that measure soil water content are neutron probes, time domain transmission (TDT) sensors, and frequency domain reflectors (FDR) [8] [9]. Neutron probes can sample large areas at a time accurately, but are expensive and have strict government regulations [8]. TDT sensors use wave propagation to measure water content, do not require government paperwork, and are relatively cheap and accurate. However, TDT sensors encounter much signal interference and are typically planted permanently in the ground. FDRs, also known as capacitive sensors, use the soil as a capacitor and are cheaper and more accurate than TDT sensors. Disadvantages to this sensor are that it must be calibrated, and it cannot measure as large of areas as neutron probes [8] [9].

The two most commonly used sensors that measure soil water tension are tensiometers and granular matrix sensors. Tensiometers work much like a thermometer, consisting of a glass, water-filled tube and gauge. They are extremely accurate and a cheaper solution than water content sensors. However, tensiometers require large amounts of maintenance, must be reset often, and don't work well below freezing temperatures because of the water-filled tube. Granular matrix sensors are extremely inexpensive and are capable of logging data. However, these sensors are also more inaccurate because they rely heavily on the salt content of the soil [8] [9].

For the project at hand, the sensors are solely used to collect data at specific points in time, which will then be processed through a server. This eliminates the need to log data in the sensor itself as granular matrix sensors do. The desired sensor is a volumetric-based soil moisture content sensor that covers an optimal amount of distance for its cost; a capacitive (FDR) sensor has been chosen to be used rather than a resistive (EC) sensor in order to prevent corrosion. The less distance the sensor can cover, the more sensors will need to be planted. In addition to soil moisture readings, it is also important to measure soil compaction to determine the available capacity for additional volumes of water per square inch of soil [2]. Local farmers have also requested that a pH level reading solution be implemented. Unlike current irrigation sensor technology, Sensor Pods containing all necessary sensors will be buried in the ground and communicate wirelessly to a base station within the WSN. The sensor readings will be cycled so that only one sensor is turned on at a time to maximize power efficiency.

The presentation of acquired sensor data to the user about their land and irrigation system is a critical part of the proposed automated irrigation system. The data must be able to be accessed remotely - in the field and on the farm. It also must be accessible through common technology mediums, such as a smartphone or web application. The most commonly used mobile operating system internationally is Android, dominating the market by supplying services to

82.8% of all smartphone users [7]. A web application will be created for ease of use, as well as its accessibility to many smartphone users. To be accessible anytime, on the farm or not, a database of sensor readings must be stored in the Cloud. Web servers, such as what Amazon hosts, are relatively cheap and can be easily integrated with the proposed web application. Using technology that is already available to many people will help promote the adoption of this new type of WSN for automated soil sensing and irrigation.

The goal of the automated irrigation system, from a farmer's perspective, is to save water and monitor the health of their crops. Whether the system is deployed in the middle of the United States or on the other side of the world in rural China, this versatile soil monitoring system will help farmers save water and money. When the soil sensors have an established connection via LoRa, data transmission will be reliable and precise. The data that the soil sensors acquire will be transmitted via the WSN back to a gateway that can be accessed through any internet connected device so that farmers can know the status of their crops and know when watering is needed.

1.4 Marketing Requirements

Marketing and engineering requirements play a valuable role in designing and constructing new products. The following sections describe how each set of requirements is used. The marketing requirements are used to validate the engineering design specifications laid out which ensure the development of a product that the intended user finds desirable. A list of these requirements is listed below.

1. The Sensor Pods will be compatible with most planters to allow for automated installation of sensors to field.
2. The sensors will accurately measure moisture in the soil.

3. The sensors will be power efficient in order to last an entire growing season.
4. A detailed interface will display sensor data collection history and trended data.
5. Wireless communications with the system will enable convenient access anywhere and anytime.

When creating marketing requirements, it is important to keep the end user in mind. The Soil Sensor Network is a project designed to aid farmers (or farmhands) in irrigation by allowing them to monitor the status of their fields. The requirements listed were chosen to create the foundation for a design that will be easy implement and easy to use, and therefore will save farmers time by automating a process that is currently done by hand on family farms.

2 ENGINEERING ANALYSIS

In the design stage of the Wireless Sensor System, it is important to look closely at all technologies and processes that are available to meet the requirements of marketing and engineering. This may include analyzing cost, materials, electrical components, and technological advantages and disadvantages. A complete engineering analysis will provide a thorough investigation of preexisting technologies and guide future development.

2.1 ELECTRONICS ANALYSIS

With the constraints of energy availability and environmental factors, multiple types of sensors and antennas were analyzed in order to accurately measure and transmit soil data.

2.1.1 Soil Sensor Analysis

The two most commonly used types of soil moisture sensors on the market are of the resistive and capacitive types. A resistive soil moisture sensor is designed with two conductive plates, as seen in Figure 1.

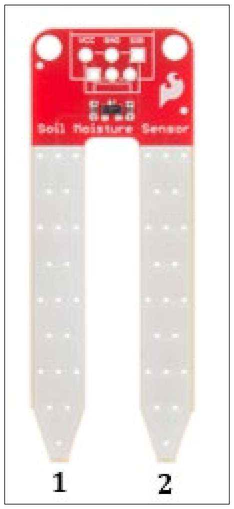


Figure 1: SparkFun Soil Moisture Sensor. Image Retrieved from https://www.sparkfun.com/products/13322.

A voltage is applied to one of the nodes on conducting Plate 1, and a current is sent from Plate 1, through the soil, to Plate 2. The moisture in the soil acts as a conductor. If the soil is extremely wet, most of the current sent from Plate 1 will arrive at Plate 2. However, if the soil is extremely dry, Plate 2 will receive almost no current from Plate 1. Resistive soil moisture sensors are easy to design, which can be as simple as connecting two conducting surfaces to an NPN transistor coupled with a potentiometer.

There are two main problems with this type of design. First, the sensor introduces a DC current into the soil. Second, it has exposed circuitry that is susceptible to corrosion. Introducing a DC current can be harmful to plants because it decreases the capacity for light reflectivity [10]. Light reflectivity plays an important role in photosynthesis by preventing the plant from absorbing too much light. When plants absorb too much light, the light waves break down the plant's chemicals used for photosynthesizing, which causes the plant to die over time. The second problem, exposed circuitry, is a cost expense to the farmer. When components are exposed to the elements, they corrode more quickly, thereby decreasing the lifetime of the sensor. A decreased lifetime means the farmer has to buy replacement sensors more frequently.

To prevent these issues, a capacitive soil moisture design was chosen to be used to detect moisture levels in the soil. The comparison of the resistive and capacitive soil moisture sensors is presented in Table 1.

Table 1: Soil Moisture Sensor Analysis.

Description	MFG Part Number	Advantages	Disadvantages
Resistive soil moisture sensor	SEN0114 101020008	Circuit Simplicity	Introduce DC current Corrode easily
Capacitive soil moisture sensor	101020614 1528-2753-ND	Corrosion resistant Better accuracy	Circuit complexity Calibration complexity

Capacitive soil moisture sensors have fully enclosed circuitry that consists of a timer, discharge capacitor circuit, and linear integrator. An example of such a sensor is shown in Figure 2.

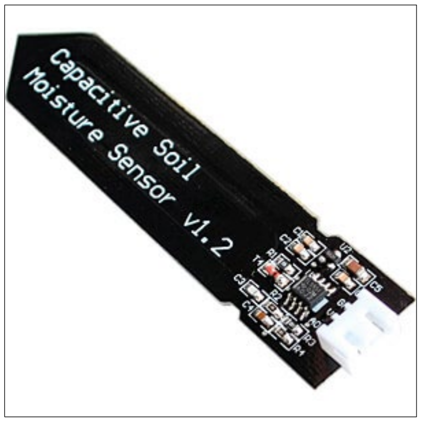


Figure 2: Analog Capacitive Soil Moisture Sensor. Image retrieved from https://www.reichelt.com/de/en/development-boards-soil-moisture-sensor-capacitive-analogue-debo-cap-sens-p223620.html?r=1.

Corrosion is not an issue for capacitive soil moisture sensors, and studies show that the capacitive sensor is more accurate than the resistive sensor, sometimes having a higher accuracy than the data sheet guarantees [11]. Both sensors are relatively cheap to build, however, the capacitive sensor has more complex circuitry compared to that of the resistive sensor. The discharge rate of the capacitor is heavily dependent on volumetric water content and soil properties.

Soil itself is a conductive material, and therefore has its own discharge rate. Every type of soil has a different discharge rate, which affects the discharge rate of the capacitor. Because of this, the capacitor contained in the discharge circuitry must be calibrated to the conductivity of the material in which it will be placed.

2.1.2 Soil Nutrient Analysis AW

Another type of sensor that local farmers requested to have on the Sensor Pod was one to analyze pH or soil nutrient levels. Table 2 lists multiple options that were researched.

Table 2: Chemical Composition of Soil Sensor Analysis.

Description	MFG Part Number	Advantages	Disadvantages
Electrochemical sensors	EC4-1000-H2 EC4-1000-H2S	Measures multiple chemical readings in the soil Fits in Sensor Pod Circuit simplicity	Expensive
pH level sensor	ZPS CIO-000-00064	pH level readings Easy to implement	Too big for Sensor Pod Expensive
Optic sensor reading litmus paper	PIR-02	pH level readings Circuit simplicity	Mechanical design complexity Single use paper

Salinity sensor	N/A	Salinity readings Easy to implement	Too big for Sensor Pod Expensive

The three types of sensors that could track the desired data are electrochemical, pH level, and salinity sensors. It was determined that none of the three types of sensors would be feasible due to their size or cost.

A fourth option considered was to use an optic sensor that could read the color of an LED flashing through a litmus paper. Optic sensor circuitry can be designed using power-efficient components that are low cost and small relative to the size of the Sensor Pod. Although a litmus paper test is not reusable, farmers only test the nutrient levels of their soil once before the growing season (before the crops are planted) and once during the growing season. However, an obstacle that would need to be overcome in order to implement such a sensor is the complexity of the mechanical design.

Litmus paper tests can only read the pH levels of content absorbed. This means that opening a door and placing the litmus paper into the dirt will not work because the litmus paper will not be able to absorb the dirt unless it is saturated. Litmus paper testing would require the pod to have a compartment containing distilled water that could be released and filtered through the dirt to be absorbed by the paper. The mechanical design and power consumption required by such a design render this option unfeasible. After multiple options were researched extensively, it was determined that adding a soil nutrient sensor was outside the scope of the project.

2.1.3 Antenna Analysis AW

To maintain power-efficient transmission, directional, dipole, and monopole antennas were researched. Since the Sensor Pods will be planted beneath the surface of the soil, the antenna chosen must have a strong enough power radiation to be transmitted through both the soil and air to arrive at the gateway. A directional antenna was a reasonable option because its small beamwidth allows it to easily penetrate through the soil without high amounts of signal loss. The polar plot for the power radiation of such an antenna is presented in Figure 3.

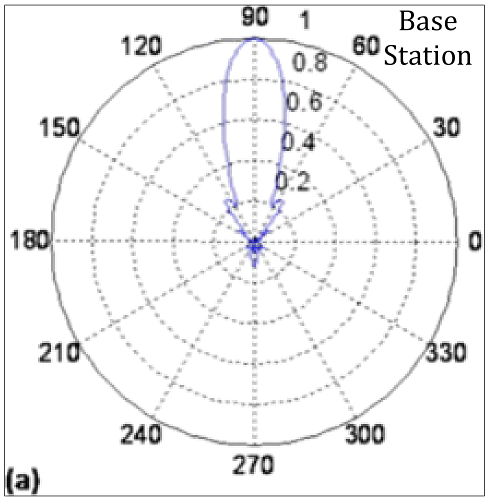


Figure 3: Power Radiation Polar Plot for Directional Antennas. Image Retrieved from https://www.sciencedirect.com/science/article/pii/S2215098616304256

The directional antenna is the most efficient of the three antennas when it comes to power radiation because the beam is capable of being directed towards the gateway. However, directing the antenna would require a motor, which consumes an unacceptable amount of power in comparison to the battery capacity. Since power is a constraint, using a directional antenna is not a viable option.

If it is not possible to direct the antenna, a small beamwidth is not desired because it does not allow room for orientation error. For instance, the pod orientation of how it lands when initially planted is unknown. If a directive antenna is used and the pod falls such that the antenna is sideways, the transmitted data will never reach the gateway because the antenna is not aimed in that direction.

For this reason, an antenna with a larger beamwidth, such as a dipole antenna, is necessary to allow for orientation error. The power radiation polar plot is shown in Figure 4.

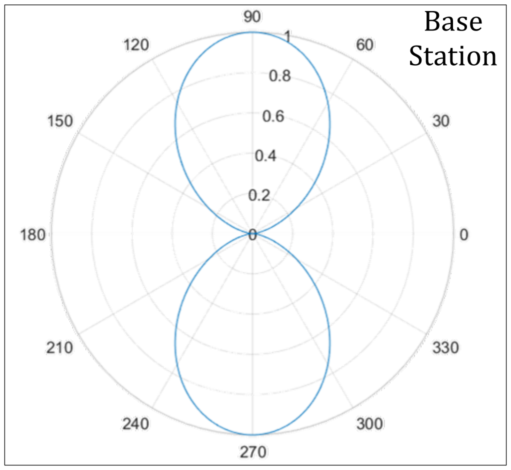


Figure 4: Power Radiation Polar Plot for Dipole Antennas.

Although the dipole antenna is a better option than the directional antenna considering power consumption, it is still not optimal. As can be seen in the power radiation plot, half of the power is radiated toward the base station, and the other half is radiated down below the Sensor Pod toward the dirt, never reaching the base station. This means that at least 50% of the power consumed is wasted.

To prevent wasted power consumption, a monopole antenna will be used. The power radiation diagram of a monopole antenna is shown in Figure 5.

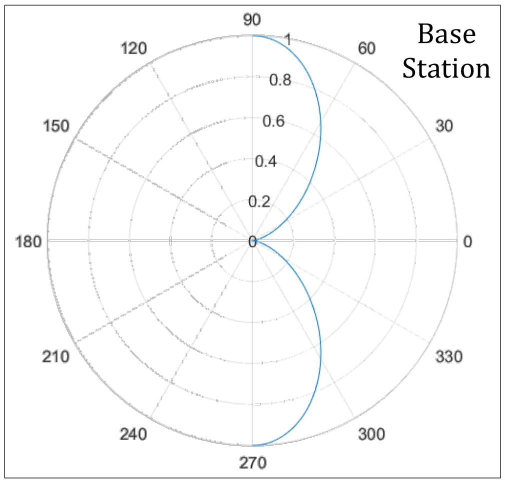


Figure 5: Power Radiation Polar Plot for Monopole Antennas.

A monopole antenna operates much like a dipole antenna, but only consumes half the power. By using a monopole antenna, it is possible to concentrate all the radiated power upwards to the surface of the soil.

Both monopole and dipole antennas have a larger beamwidth which means there is a greater chance of signal scattering. Although signal scattering is a factor that must be taken into consideration, scattering does not weaken the signal to the extent that it cannot reach the gateway. A comparison of the three different types of antennas researched is shown in Table 3.

Table 3: Antenna Analysis.

Description	**Advantages**	**Disadvantages**
Monopole	LoRa processors optimized for monopole antenna use Circuit Simplicity Power efficiency	Larger beamwidth could lead to signal scattering
Dipole	Circuit Simplicity Covers large range of frequencies	Larger beamwidth could lead to signal scattering
Directional Antenna	Small beam width allows for better soil penetration	Complex circuitry

	Stronger, more concentrated signal allows for better transmission over longer distance	Not power efficient

By analyzing the beamwidth, power consumption, and power radiation of each antenna, it was determined that the monopole antenna is the most viable option. A quarter wavelength was chosen in order to further reduce power consumption.

2.2 Circuit Analysis

The Wireless Sensor Pod is powered by a Lithium-Ion battery that is intended to power 3.3V components and last an entire growing season. To accomplish this, a circuit analysis was conducted to verify the battery life and input power on all circuit components.

2.2.1 Battery Analysis

In order for the sensor pod to last the intended duration of time, a battery was chosen that would be able to output a voltage high enough to allow all components to operate despite the charge of the battery.

2.2.1.1 *Battery Research* *AA, LF, AW*

One of the major constraints for circuit design is power consumption. Given that the Sensor Pod has a dimension requirement of 90 x 90 x 100 mm, all batteries used must fit into this size requirement. To determine how much voltage and current are needed to power the components, a list of devices with their power consumptions was created.

Table 4: Battery Power Analysis.

Description	**MFG Part Number**	**Operating Voltage (V)**	**Current Draw**
Soil Moisture Sensor	Built In-House	3.3	100 µA
Microcontroller	PIC24F265	3.3-3.7	Active: dependent on peripherals Deep Sleep: 500 nA
Lora Module (Antenna)	RN2903	3.3-3.7	Transmit: 121 mA Sleep: 1.3 µA
Temperature Sensor	MAX6607IXK+T	1.8-3.6	15 µA max

Every device chosen can operate safely at 3.3 V. Because of this, the base voltage the battery will supply is 3.3 V. A standard battery operates at 3.7 V, so a voltage regulator will be implemented between the battery and microcontroller to ensure safe operation of components and increase the life of the Sensor Pod. The microcontroller and LoRa devices are able to conserve power by entering into different power modes, such as active, idle, or sleep. Active mode means the device is on and actively waiting for commands to transmit and receive data. Idle mode means the device is on but has limited functionality. Sleep mode is when the device turns off the majority of its functionalities and waits for a "wake up" signal.

Equation 1 was used to determine how long the battery will be in use throughout the growing season.

$$\frac{5\ minutes}{use} * \frac{3\ uses}{day} * \frac{153\ days}{growing\ season} * \frac{hour}{60\ minutes} = 38.25\ [Hrs] \qquad (1)$$

It is not required for the sensors to take readings at every instant of every day. Traditionally farmers take soil readings two to three times per day. In the calculation above, worst-case scenario was assumed. For the project at hand, sensor readings will be taken three times a day, and it will take up to five minutes to transmit the data back to the gateway. The sensors will be on this cycle for an entire growing season, which lasts approximately 153 days, or five months. To conserve power, the ideal mode of operation for the microcontroller and LoRa module when the sensors are not collecting data is sleep mode.

When deciding on the size of the battery, the following assumption were made. The current draw for the sensor pod *c* is assumed as 73.5 mA*H; this will cover the average current draw during the hours of operations. As calculated in Equation 1, the total hours of operation h is 38.25 Hrs. The mA-H of the battery *b* required for proper operation of the Sensor Pod is calculated as follows:

$$c * h = b[mAH] = (73.2) * (38.25) \approx 2800[\mathrm{mAH}] \tag{2}$$

As seen through Equation 2, the minimum size for the battery is 2800 mA*Hr. For this reason, a battery with a size of at least 2800 mA*H was chosen. An estimated current draw for the system of approximately 73.5 mA and an estimated time of 38.25 hours based off of the length of an average growing season (5 months) and a run time of 5 minutes 3 times a day, as seen in Equations 1 and 2, the estimated size of battery would be approximately 2800 mA*Hr. The microprocessor requires 3.3 V to run optimally, with a minimum of 2 V and a maximum of 5 V. A buck-boost voltage regulator has been implemented to maintain a battery voltage of at least 3.3 V, and will send a signal to the farmer if the voltage drops below 3.3 V so the Sensor Pod can be retrieved before the battery dies if this situation occurs.

2.2.1.2 *Battery Testing* *LF*

In order to provide increased battery life, the battery is interfaced with a switching regulator, allowing limited voltage and current pull from the battery. The battery depleted after approximately 40 hours when a 100 Ω resistor load was applied with a current draw of approximately 70 mA. This was calculated using Equation 2. Based off the time of depletion, it will take approximately 38 hours before the battery enters its end-of-discharge voltage, 2V, given a maximum allowable battery voltage of 4.2V. The end-of-charge voltage was determined based on the discharge profile graph from the Adafruit website and the maximum voltage is based off of the base battery voltage [22].

2.2.2 Voltage Regulation Analysis LF, AW

In order to preserve battery life through increased efficiency, voltage regulators that could be integrated into the circuit were researched.

2.2.2.1 *Voltage Regulator Research*

Several voltage regulators were analyzed when designing the power stage. These were the linear, the switching, and the Single-Ended Primary-Inductor Converter (SEPIC) regulators.

The linear regulator was first chosen due to its simplicity and virtually noiseless supply. However, this type of regulator dissipates excess voltage as heat, making it power inefficient. Because power efficiency is an integral aspect of the application, this regulator will not be used.

The SEPIC regulator was analyzed next. The design consists of a boost converter followed by an inverting buck-boost regulator. This was quickly disregarded, as it creates circuitry that is too complex and too large to fit within the size constraints.

A switching regulator was found to be one of the best choices for design implementation. The switching regulator is similar to a buck regulator in that it has a high power efficiency, 80-90%, but does not have the downsides of a buck regulator, such as complex circuitry and an inverted output [21]. The reason the switching regulator is able to step up or step down voltage without having the same issues as the buck regulator is due to the fact that the switching regulator does not have a large voltage conversion range. The comparison of the four types of regulators can be seen in Table 5.

Table 5: Voltage Regulator Comparisons.

	Linear Regulator	Switching Regulator	Buck Regulator	SEPIC
Power Efficient	No	Yes	Yes	Yes
Inverted Output	No	No	Yes	No
Complex Circuitry	No	No	Yes	Yes
Noise Created	None	Low	High	High

The switching regulator allows the system to convert high or low voltage to the desired voltage without causing too much noise to be produced. It also requires no complex circuitry for filtering and gives a noninverted voltage output. This will allow use of minimal PCB space because of simpler circuit configuration.

Although the switching regulator was efficient, after more research it was not found to be the most efficient. The switching regulator, as stated previously, has an 80-90% efficiency. This means the circuit could lose up to 20% of its supplied power. More research was conducted and a buck-boost regulator was found to have the highest efficiency, 97%, while not being as complex as the SEPIC regulator.

2.2.2.2 *Voltage Regulator Testing*

During the design phase, the switching boost regulator was tested. The voltage regulator was connected to a power supply so that when the voltage was increased and decreased the input voltage to the circuit after the regulator could be measured at a constant value to prove the regulator was working properly. A few problems were encountered during testing. The first was that the regulator has a minimum current draw in order to activate. The default setting on the power supply is 0 A. Once the current was set to the minimum current of the regulator, the regulator worked for the first twenty minutes and then stopped. After much troubleshooting it was found that the regulator needed a couple resistors in front of one of the pins to act as a voltage divider and prevent the component from being burned out. The circuit was reconfigured and a new regulator was put in place. The reconfigured circuit design worked properly, but it was soon realized that a switching boost regulator only boosts the voltage up to 3.3V and has no effect on the upper-end voltage above 3.3V.

The final design tested was the buck-boost regulator. After constructing the correct circuit design, the regulator entered its boost stage between 2.9-3.2V, then entered its buck stage between 3.4-3.7V. The third stage, pass-through, was entered when the voltage dropped below 2.9V. A PCB of the circuit was created to be used in the implementation phase.

2.2.3 555 Timers **AW**

There are various types of 555 timers that can be used when designing a circuit. The TLC555 timer was chosen because it has an operating voltage of 3.3V and is located in the university stock room. Having the parts in stock eliminates shipping time and minimizes cost.

A timing circuit can be either monostable or astable. A monostable circuit consists of two states; the circuit is stable in the first state and unstable in the second state. The circuit only enters the second state if the trigger input of the timer is externally excited. Once the circuit enters the second state, it produces a single output pulse and then returns to the first state. An astable timing circuit consists of two states of which neither are stable. The circuit therefore oscillates between the two states automatically without receiving an external trigger signal.

The Sensor Pod circuitry is located inside the pod, which is buried in the ground during operation. Because of this, a monostable circuit would be extremely difficult to use since an external signal is required to transfer from state one to state two. An astable circuit can be implemented more easily because it oscillates between the two states and does not require an external excitation. This type of timer produces a square wave at its output that can be used to measure how long the capacitive sensor circuit takes to discharge.

2.3 Communications Analysis

When designing the Wireless Sensor Network (WSN), reliable and power efficient data transmission is extremely important. As will be discussed in the Computer Network section, the Sensor Pods will be communicating via the Long-Range Wide Area Network (LoRaWAN). This will allow sensor readings from the farm to be transmitted long distances to a gateway while maintaining power efficiency. As shown in the communication stack in Figure 6, there are two main sections to LoRaWAN: the MAC layer and physical layer.

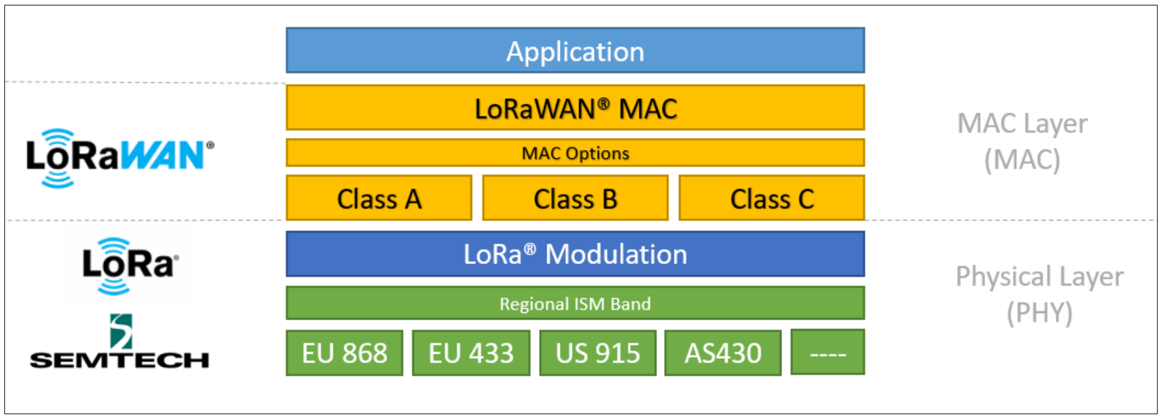


Figure 6: LoRaWAN Communication Stack.

The MAC layer is broken down into the LoRaWAN MAC Protocol and Mac Option class. The physical layer is composed of LoRa Modulation and Regional ISM Band.

2.3.1 LoRa Modulation

Long Range (LoRa) modulation, as will be further explained in the Computer Network section, is a Low-Power Wide-Area Network (LPWAN) protocol developed by Semtech. At the physical layer, LoRa modulation is based on Compressed High-Intensity Radiated Pulse (CHIRP) spread spectrum, and although the modulation is proprietary to Semtech, a base understanding of how the physical layer operates can still be explained. The standard composition of the LoRa's physical layer, as shown in Figure 7, is composed of the preamble (up-chirp), sync (down-chirp), and data modulated.

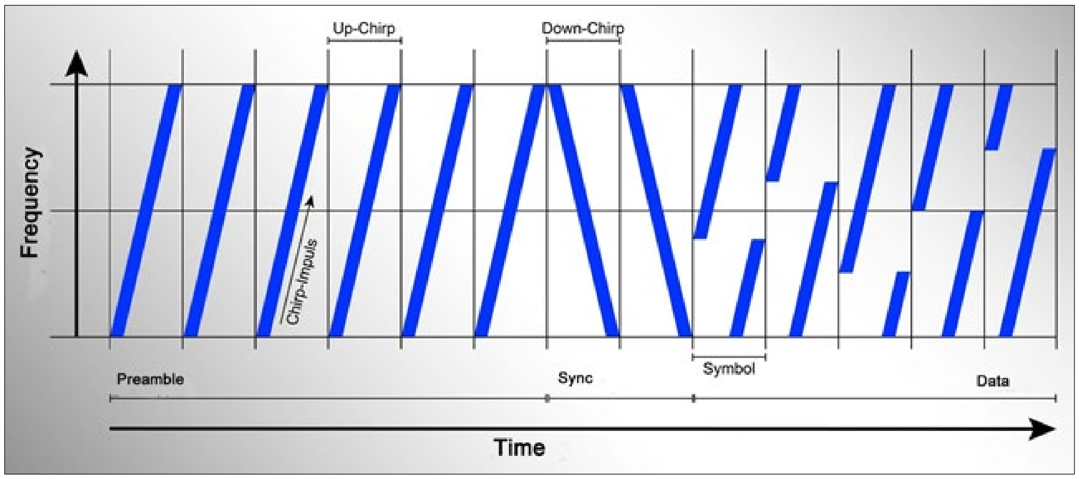


Figure 7: LoRa Communication CHIRP (Modulated Data). Image retrieved from

The preamble is used for detection of LoRa chirps, which is followed by sync for time synchronization. Time synchronization is important so that the receiver is synchronized with the incoming signal so that the data packets can be interpreted properly. Data modulation is implemented to allow for signals to be transmitted over higher frequencies in order to decrease the size of the receiving antenna. The Regional ISM band of the physical layer is used for regional standards. For the current application, US 915 frequency is used. The North American band has 64 up-links at 125 Hz along with 8 up-links at 500 Hz and 8 down-links at 500 Hz.

2.3.2 LoRaWAN MAC Protocol **AA, RK, AW**

LoRaWAN is the MAC layer that operates on top of the LoRa physical layer. LoRAWAN MAC protocol operates similar to ALOHA protocol, although the packet lengths can change for LoRaWAN. The LoRaWAN frame format is composed of the MAC header, MAC Payload, and Message Integrated Code (MIC), as shown in Figure 8.

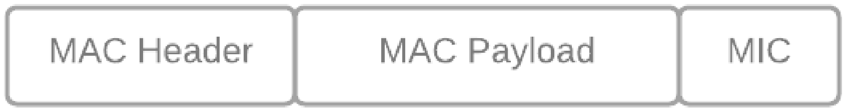


Figure 8: MAC Layer Format.

The MAC Header contains information with regards to the protocol version and message type, and MIC is used for message authentication calculate based on a portion of the MAC Payload and Network Key. The MAC Payload, while holding the actual Data (encrypted by the App Session Key), also contains information with regards to the application layer such as device key, app session key, port, etc. The Network Key is used to validate the MIC and aids in authentication and routing to the correct network.

The LoRaWAN module consumes the most power when transmitting and receiving data. LoRaWAN devices can be classified into three types based on receiving patterns: Class A, Class B, and Class C. The class chosen will dictate how much power is consumed/conserved. The device is able to conserve power by entering into one of the three different modes described in the Circuit Analysis. It is important to choose the correct class because the device is unable to conserve power, by entering sleep mode, while transmitting or receiving data. Figure 9 displays the power consumption of the three device types.

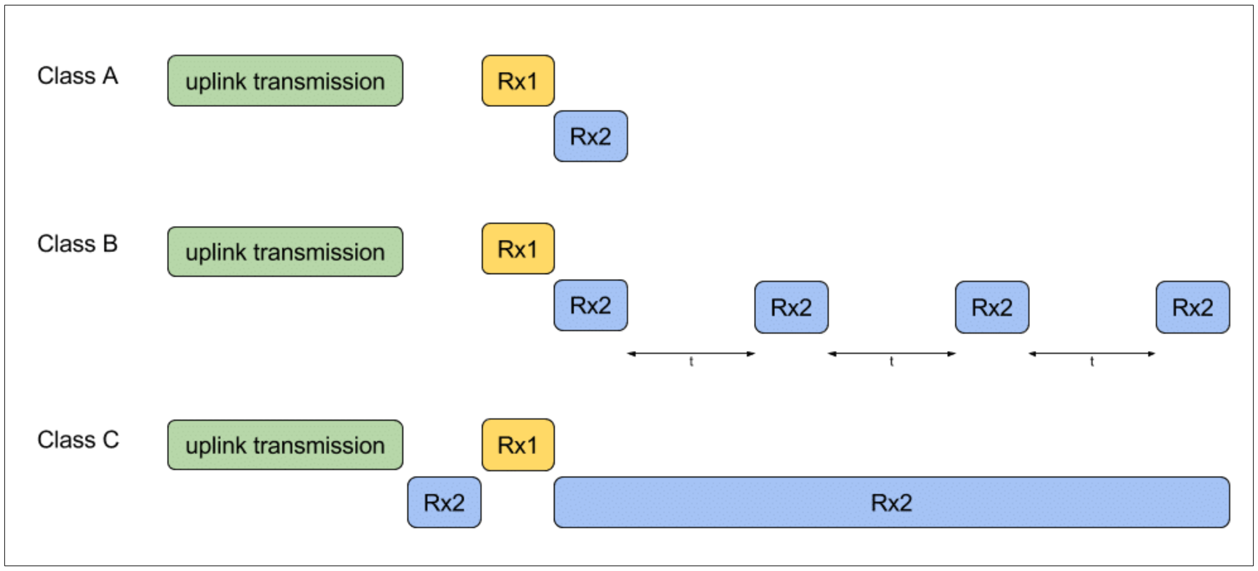


Figure 9: LoRaWAN Class Types. Image retrieved from https://witekio.com/blog/lorawan-a-dedicated-iot-network/

Class A devices first transmit data *Rx1*, which is followed by one or two periods in which it can receive messages *Rx2*. Class B and C expand the amount of time, or the number of periods, for receiving a message. Class B allows for additional time-synchronized receive period, and Class C keeps the receive period continuous while the device is not transmitting.

The use of additional periods for receiving messages in the Class B and C devices are not required for the Soil Pod WSN application. Not only would this increase the power consumption of the devices without adding any benefits, but also would restrict the device from a power-saving sleep mode. Therefore, Class A is the optimal choice to ensure low power consumption while not hindering operations of the device.

2.3.3 Communication Range AA, AW

To ensure effective communication of the LoRa device to the gateway, the range of operations must be taken into consideration. As stated in the Engineering Requirements, the range necessary for a large family farm was 3.39 km. To verify the module could communicate over this range, the average distance for a large family farm was calculated and compared to the maximum possible range of the LoRa device.

The distance required for proper communication on an average-sized farm was first calculated.

$$\sqrt{5.75\ km^2} = [2.4\ km] \tag{3}$$

The assumption for the above calculation is that the farm is square in shape and the average size of a large family farm as stated by the USDA is 1421 acres, or 5.75 km^2, with a perimeter of 2.4 km [18].

The longest distance would be the hypotenuse of the farm:

$$\frac{2.4\ km}{\cos 45} = [3.39\ km] \tag{4}$$

The range needed for communication on such a farm is 3.39 km, assuming a square field. To account for changes in shape and position of the gateway, a twenty percent deviation can be applied.

$$3.39\ km + (3.39\ km *\ 0.2) \approx 4\ [km] \tag{5}$$

The LoRa communication needs to reach a range of 4 km to insure proper operation of the WSN.

When calculating the maximum possible range for the LoRa module some factors needed to be taken into consideration. These factors are the power transmitted, sensitivity of the receiver, and signal attenuation due to the soil. All equations listed below were sourced from *Engineering Electromagnetics Third Edition* [14]. To the calculate the range, the terms in the average power in Equation 6 were rearranged to solve for range *R* as seen in Equation 7.

$$P_{av} = \frac{P_{rad}}{2\pi R^2} \tag{6}$$

$$R = \sqrt{\frac{P_{rad}}{2\pi P_{av}}} \tag{7}$$

This R will be the maximum possible range given that P_{rad} is power radiated in terms of mW and P_{av} is average power. Given that the LoRa transmitter is rated at a maximum of 18 dBm ($P_{t,max}$), the power of P_{rad} can be derived from Equation 8 and solved for in Equation 9 [19].

$$P_{t,max} = 10\, log_{10}(P_{rad}) \tag{8}$$

$$P_{rad} = 10^{\frac{1}{10}P_{t,max}} = 10^{1.8} = 63\ [mW] \tag{9}$$

Next the power received by receiver P_{rec} is determined as shown in Equation 10.

$$P_{rec} = \frac{P_{rad}}{2\pi R^2} * A_e = \frac{P_{rad}}{2\pi R^2} * \left(\frac{0.328}{4\pi} D\right) \tag{10}$$

Here A_e is the effective area, and assuming worst-case scenario of a uniform radiation, the value D is set to 1. Since P_{rec} is in mW and the $sensitivity$ of the LoRa gateway is in dBm, a conversion must be made to create a common unit.

$$sensitivity = 10\, log_{10}(P_{rec}) = 10 log_{10}\left(\frac{P_{rad}}{2\pi R^2}\left[\frac{0.328}{4\pi}\right]\right) \tag{11}$$

$$\frac{P_{rad}}{2\pi R^2}\left[\frac{0.328}{4\pi}\right] = 10^{\frac{sensitivity}{10}} \tag{12}$$

Equation 12 is then rearranged to solve for the distance R as shown in Equation 13.

$$R = \sqrt{\frac{0.328^2 P_{rad}}{8\pi^2\left(10^{\frac{sensitivity}{10}}\right)}} \tag{13}$$

Since the Sensor Pod will be under ground, the signal loss due to soil attenuation needs to be considered. For the calculation performed in Equation 14, the conductivity of the soil *σ* is dependent on the soil properties and frequency. For simplicity, the conductivity of the soil will be approximated to 0.1 [S/m]. The relative permittivity ε_r is approximately five times that of permittivity in free space ε_0. The angular frequency *ω* is dependent on the frequency at which the signal is being transmitted.

$$Soil\ loss = tan\theta_{loss} = \frac{\sigma}{\omega\varepsilon} = \frac{\sigma}{2\pi\varepsilon_r} = \frac{0.1}{2\pi(915x10^6)[5(8.854*10^{-12})]} = 0.39 \qquad (14)$$

Furthermore, the signal loss can also be accounted for by calculating the attenuation constant α, as seen in Equation 15. The relative permeability is considered to be equal to the permeability in free space μ_0.

$$\alpha = \sqrt{\pi f \mu_0 \sigma} = \sqrt{\pi(915 * 10^6)(4\pi * 10^{-7})(0.1)} = 19 \left[\frac{Np}{m}\right] \qquad (15)$$

The attenuation constant can then be used to find the signal power that will reach the surface if buried at a distance *d* of 6 inches, or 0.15 m. The surface power $P_{surface}$ is calculated in Equation 16. Note that the radiated power P_{rad} is considered to be 0 dBm, or 1 mW, at the transmitter's output.

$$P_{surface} = P_{rad}e^{-2\alpha d} = 1e^{-2(19)(0.15)} = 0.0034\ [mW] \qquad (16)$$

The radiated power at the surface, accounting for the loss caused by the soil during propagation, was found to be 0.0034 mW. Now that the power that reaches the surface is known, it can be inserted into Equation 17 to find the distance the signal can travel. The sensitivity as shown on the datasheet is -146 dBm [19].

$$R = \sqrt{\frac{0.328^2 P_{rad}}{8\pi^2\left(10^{\frac{sensitivity}{10}}\right)}} = \sqrt{\frac{0.328^2 * 0.0034}{8\pi^2(10^{-14.6})}} = 108\ [km] \tag{17}$$

Since it is not possible with the known information to account for every loss (i.e.: soil properties, antenna position, exact depth, etc.) the distance will be derated by 50% to account for these losses.

$$R_{losses} = \frac{108}{2} = 54\ [km] \tag{18}$$

Comparing the maximum range with the necessary range stated in the Engineering Requirements, it is evident that the required range falls well within the possible range of LoRa communication.

2.4 COMPUTER NETWORKS ANALYSIS RK

When developing a Wireless Sensor Network (WSN), there are many different communication protocols that can be chosen. The choice should be application specific, taking into account the outdoor environmental conditions, as well as requirements for data transmission. The Sensor Pods will be planted underground and will be placed at a target distance of roughly 3.3 kilometers from its destination (receiver/Gateway). The pods will not be easily serviceable once installed because they are underground. For this reason, interaction with the pods will only take place at the beginning and end of growing seasons. With the requirements of range coverage and environmental factors, a few different communication standards will be considered.

One common method of data transmission in home networks and embedded systems is the use of Wi-Fi. In general, Wi-Fi excels at short-to-medium range communication between tablets, smartphones, and many other embedded devices. Current Wi-Fi Standards support 5

GHz and 2.4 GHz bands in which devices can connect. Although the faster 5 GHz frequency can handle greater bandwidth, it can be easily obstructed, proving reliable only for short distances. The slower 2.4 GHz band will stretch longer distances, up to 300 feet, but is also easily obstructed by structures such as walls or floors in a home. In both cases, this is not even close to the target distance for the SPU to transmit data.

A much longer communication protocol that has been popular since 2015, is the LoRa protocol. LoRa is a highly specialized, long distance routing protocol for embedded devices. Compared to other Low Power Wide Area Networks (LPWAN), the battery life of LoRa-powered devices is three to five times longer [13], proving the power efficiency of the distance routing protocol. The data transmitted is limited to small payloads, which is adequate for the Sensor Pod because only lightweight data, such as sensor readings and time stamps, are going to be transmitted. According to LoRa Alliance, baud rates range from 0.3 kbps to 50 kbps. When choosing LoRa communication, the frequency band at which it operates must be considered. In the U.S, the 902-928 MHz band has been allocated for LoRa usage, which requires U.S. specific modules for operation [14].

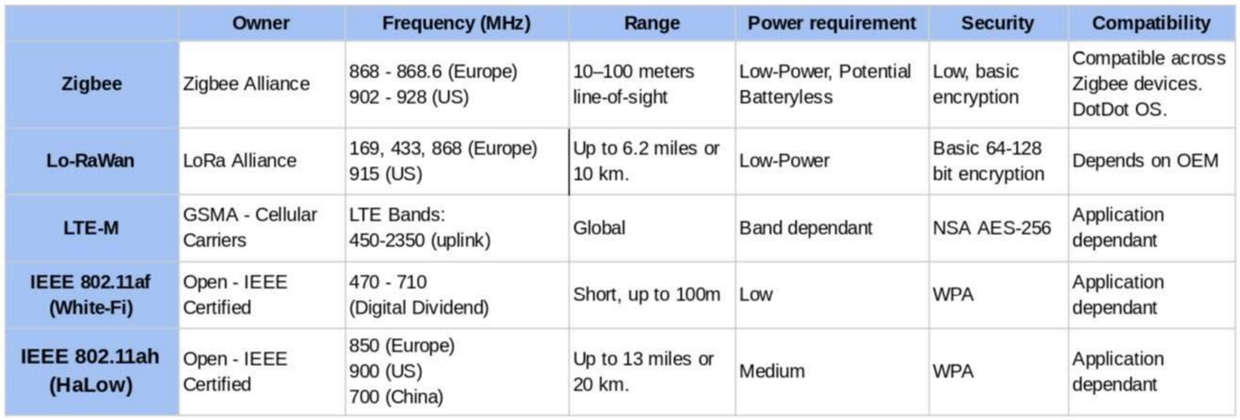
The Narrow Band Internet of Things (NB-IoT) is another long-range communication protocol that is supported by the Third Generation Partnership Project (3GPP) and GSMA [15]. There are a few key similarities between it and LoRa technology. They are both low power networks that target long range distances. One key difference is that it uses the licensed spectrum, which is what the cell networks operate on, as opposed to the unlicensed portion of the spectrum. Using the 3G and 4G networks requires access to cell towers, which costs money to use, and the farms where the WSN is located may not have a strong connection to them.

The final communication protocol that will be analyzed is Zigbee. This is another communication protocol in addition to LoRa, that operates in the "amateur" 902-928 MHz band of frequencies. The advantage of using Zigbee is that it can be integrated into existing Zigbee-powered networks. In contrast, the compatibility of LoRa will depend on the OEM who made the embedded device. For the application of the Wireless Sensor Network, only official Sensor Pods that are of the same type will be integrated into the system, therefore there is no extra care that needs to be taken to ensure compatibility.

When considering the range of communication that Zigbee can support, it is often regarded as the premier smart home network communication. Zigbee usually transmits from 10 meters to a maximum of roughly 100 meters, given a perfect line of sight. This is where Zigbee falls short of the transmission requirements of the Wireless Sensor Network.

Wi-Fi, LoRa, NB-IoT, and Zigbee are some of the most powerful protocols in the industry. Comparisons of these technologies are shown in Table 6.

Table 6: Top Wireless Standards for IoT Devices. Retrieved from IoT EE Times.



	Owner	Frequency (MHz)	Range	Power requirement	Security	Compatibility
Zigbee	Zigbee Alliance	868 - 868.6 (Europe) 902 - 928 (US)	10–100 meters line-of-sight	Low-Power, Potential Batteryless	Low, basic encryption	Compatible across Zigbee devices. DotDot OS.
Lo-RaWan	LoRa Alliance	169, 433, 868 (Europe) 915 (US)	Up to 6.2 miles or 10 km.	Low-Power	Basic 64-128 bit encryption	Depends on OEM
LTE-M	GSMA - Cellular Carriers	LTE Bands: 450-2350 (uplink)	Global	Band dependant	NSA AES-256	Application dependant
IEEE 802.11af (White-Fi)	Open - IEEE Certified	470 - 710 (Digital Dividend)	Short, up to 100m	Low	WPA	Application dependant
IEEE 802.11ah (HaLow)	Open - IEEE Certified	850 (Europe) 900 (US) 700 (China)	Up to 13 miles or 20 km.	Medium	WPA	Application dependant

For its long-distance capabilities, ease of use, and low cost of operation, LoRa will form the WSN communication backbone.

2.5 Embedded Systems Analysis AA, RK

At the core of the Wireless Sensor Pod will be an embedded processor. For the application of a Wireless Sensor Network, a series of low power embedded device is crucial to delivering extended battery life while buried in soil. A few different microprocessors ranging in power, communications, and Input/Output specifications will be analyzed.

First, a common choice for developers is the PIC family of processors. In particular, the 8-bit PIC18F2525 has a few key features that make it suitable for the Wireless Sensor Pod. According to Microchip's datasheet, this microcontroller can accept a wide range of input voltages, from 4.0V down to 2.0V to support low voltage inputs in power-sensitive applications [16]. The downfall of this processor is that it does not have any Universal Asynchronous Receiver Transmitter (UART) ports. The Lora Module and at least one of the soil sensors will require one UART port each.

In the PIC24F "GB" family of Microchip processors exists the PIC24FJ256GB410. This processor is desirable because there are 6 total UART ports, which will make it possible to connect the LoRa Transceiver to the chip. The LoRa transceiver uses 2 UART Ports (1 optional) to communicate with the microprocessor. Another core feature of this microcontroller is that it implements Extreme Low-Power (XLP) operation. The availability of multiple sleep modes is vital to the survival of the Sensor Pod during the growing season. Maximizing the life of the battery means that it will not need to be serviced. When the Sensor Pod is not acquiring or transmitting measurements, it will be placed into a deep sleep mode, optimally consuming around 650 nA when running the Real Time Clock Calendar (RTCC). The RTCC can be run in deep sleep mode to generate an interrupt, which will feed back into the microcontroller to wake it up. For all the reasons listed, the PIC24FJ256GB410 Microcontroller will be the brains of every Sensor Pod.

The main Microcontroller that is responsible for taking measurements, will also be joined alongside a LoRa transceiver module. This module will take care of the physical layer of data communication from the Sensor Pod to the Gateway. A RN2903 transceiver is connected to the microchip through UART and will be used to support antenna communications. Since LoRa is band-limited in different countries, this transceiver has been chosen since it operates within the valid 902 to 928 MHz frequencies.

As was previously described with the main microcontroller, this embedded chip also allows for an ultra-low-power consumption mode called “Deep Sleep”. In its idle state when powered with 3.3 V, it is expected that the transceiver will draw about 2.8 mA. In sleep mode, current consumption decreases by more than a factor of 2000, only consuming 1.3µA. This information is only an approximation, and assumes an ambient temperature of about 25 °C. The transceiver can also assume to be powered from a 3.3V battery, although it can support input voltages as low as 2.1V. The RN2903 Module can be powered from the main microcontroller of the Sensor Pod to reduce wiring complexity, making manufacturing and design easier and cheaper [17].

On-board UART support will be used for communication between the microcontroller and the transceiver. The RN2903 transmits data with a default baud rate of 57 kbps, which should be adequate for the low amount of data that will be transmitted. Another connection that must be made on the transceiver is to the antenna, attached at the RF port.

When choosing embedded components, such as the microcontroller and transceiver module, the necessary ports, power consumptions, and frequencies has to be analyzed. A

PIC24FJ256GB410 microcontroller will be used for its multiple UART ports and a RN2903 will be used as a transceiver for the external LoRa antenna because of the deep sleep mode and compatibility with the necessary frequency range.

3 ENGINEERING REQUIREMENTS AA, LF, RK, AW

The Wireless Sensor System developed should meet strict engineering requirements. These requirements will set certain specifications, for components regarding power management and communications to design a marketable product; it will take advantage of current technologies used in the industry. The following table will guide the design and engineering process of the Wireless Sensor System.

Table 7: Engineering and Marketing Requirements.

Marketing Requirements	Engineering Requirements	Justification
3	The Sensor Pod system will be *low power*, requiring a maximum of 5600mA-H.	The Sensor Pod is battery-powered and must last an entire growing season (3-5 months) since the pod is in the ground and cannot be recharged mid-season. The Sensor Pod is also small, and therefore the battery must be small to be able to fit inside the pod.
3	The Sensor Pod will be designed with *energy efficient circuitry*; when no sensors are in use, the pod's energy management system will allow the electronics to enter sleep mode, automatically "waking up" to take a reading 3 times per day.	
1	The *Sensor Pod dimensions* will be no larger than 90 x 90 x 100 mm (excluding antenna) to allow for easy installation.	Having a Sensor Pod of this size will allow it to be inserted into a mechanism attached to the planter so that the pod can be "planted" with the seed, minimizing installation time.

3, 5	*Wireless communications* of the sensor pod must transmit data a minimum distance of 3.39 km.	<u>Rough Calculation:</u> Avg. distance between sensors: 1 per 25 acres Avg. size of large family farm (USDA): 1421 acres Convert to 1421 acres to km^2: 5.75 km^2 Distance (perimeter/side): sqrt(5.75): 2.4 km Hypotenuse (longest distance): 2.4/cos(45): 3.39 km Communications will have the ability to be transmitted long distances to connect with individual Sensor Pods.
2, 4	The sensor pod will *automatically* measure and transmit the soil moisture level and temperature three times a day.	Farmers do not need to know what the moisture of the soil is at every instant of the day. Typically, measurements are only taken 2-3 times a day according to local farmers.
5	The sensor data will be *stored and trended* to give a visual representation of field analytics.	Displaying and trending field data is a request from local farmers.
2	The soil sensor will be able to measure the moisture level of the soil with a minimum *80% accuracy*.	The average accuracy of a commercial soil moisture sensor is 94-97%. Since the sensors are being designed and built, the sensors may be less than 94% accurate, but must be accurate enough for the farmers to know when and to what extent to irrigate their field.
5	Sensor Pods will accurately *communicate wirelessly through at least 3 inches* of soil.	Communication beneath soil is crucial because the Sensor Pods are located underground.
4, 5	*An application* will contain *fault detection* to determine if communication has been lost with the Sensor Pod and will alert the farmer.	If the Sensor Pods are not communicating with the server, farmers will not know when the field needs to be watered. Therefore, farmers need to know if communication has been lost.

2, 4, 5	*An application will alert the farmer* if excessive soil conditions occur (i.e.: if the soil is exceptionally dry) so immediate action can be taken.	According to local farmers, irrigation systems are set to turn on at a certain time and run for a certain duration. If a field is exceptionally dry, the irrigation system may need to run for longer to supply enough water to the field.
Marketing Requirements 1. The Sensor Pods will be compatible with most planters to allow for automated installation of sensors to field. 2. The sensors will accurately measure moisture in the soil. 3. The sensors will be power efficient to last an entire growing season. 4. A detailed interface will display sensor data collection history and trended data. 5. Wireless communications with the system will enable convenient access anywhere and anytime.		

4 ENGINEERING STANDARDS

The Wireless Sensor System will be built upon common industry standards and specifications. Certain standards may be used for increased security, physical durability, and a robust software infrastructure. The six main engineering standard specifications are listed in Table 8.

Table 8: Engineering Standard Specifications.

	Standard	Use
Data Formats	SQL	Web Server
Programming Languages	C/ C++	Main Microcontroller
	TypeScript	Web Server Frontend
	C#	Web Server Backend
Connector Standards	UART	LoRa Module and Microcontroller
Communications	LoRa	Sensor Pods

The above table mentions specific programming languages, such as C and C++ for the main microcontroller, and LoRa communication for the Sensor Pods. These standards will be followed for each of the components listed.

4.1 DATA FORMAT

A few key data types and data storage conventions will be used to maintain farm data for the Wireless Sensor System. The Database will be structured in the format depicted in Figure 10, broken down into three tables: Farms, Sensor Pods, and Sensor Data.

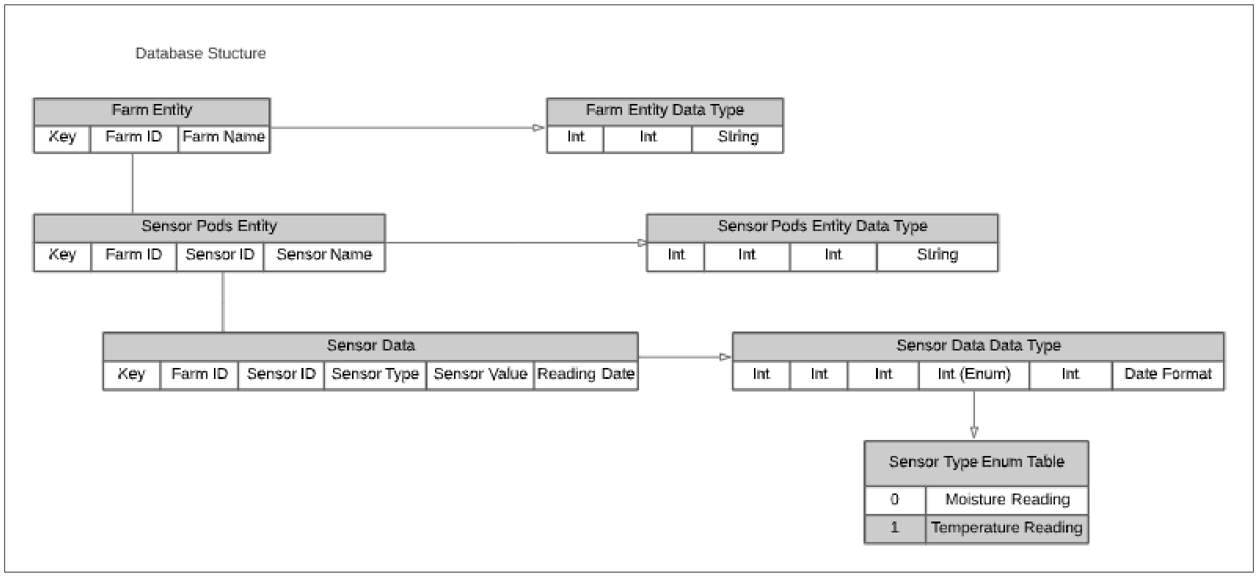


Figure 10: Database Structure.

The Farm tables will contain an entry for each Farm with their Name and a Farm ID. The Farm ID will be the main connection point to organize the collection of Sensor Pods and Sensor Data that belong to a specific farm. The Sensor Pods table will contain a Farm ID for farm assignment, Sensor ID, and a Sensor Name. The Sensor Data table will then hold all data collected and will contain the Farm ID, Sensor ID, Sensor Type, Sensor Value, and Reading Date. These values will ensure that each entry can be traceable to a specific Sensor Pod and Farm.

4.2 Programming Language AA, RK

Best practices and techniques will be used in the development for the software of the Soil Sensor Network. Programming languages like C/C++ will be used for the embedded microcontroller programming. Higher level languages like TypeScript and C# will be used to develop the user interface.

4.3 Communications RK

The Wireless Sensor Pod communications will comply with FCC Radio regulatory Approvals, including under § 15.247, titled "Operation within the bands 902-928 MHz, 2400-2483.5 MHz, and 5725-5850 MHz".

4.4 Connector Standards RK, AW

The RJ45 connector is used to connect the microchip to a PC for programming via a RJ11 cable. In the NEC § 725.144 it states that a cable must not exceed the current rating of its connector. An RJ11 cable has a lower current rating than an RJ45 connector, and therefore meets the new 2017 NEC standard.

Both types of connectors were considered for programming the PIC, however due to the small space requirements of the board and Pod, header pins were placed on the edge of the board instead. The header pins connected to a PICKit 3 programming device that is specially designed for Microchip processors.

When using cables for an outdoor environment application, IP ratings must be taken into consideration. IP67 rating states that the component is water resistant up to 1 meter. Since the Sensor Pod is only buried 6 inches below the soil, IP67 rating will suffice for this application. P2 and P3 connectors will be used for the connections from the main PCB to the supplementary PCBs and exterior sensors.

5 ACCEPTED TECHNICAL DESIGN

The Wireless Sensor System can be broken down into multiple subsystems comprised of hardware and software.

5.1 HARDWARE DESIGN

There are four main components to hardware design: block diagrams, schematics, simulations, and printed circuit boards (PCB). Block diagrams give a basic top-down overview of the major subsystems. Once the subsystems are broken down into basic components, schematics were created. Schematics show the circuit design that can be implemented during the prototype stage. Before prototyping, simulations were created to verify the ideal circuit design and give a general understanding of how the circuit will operate. PCBs were then designed based off the schematics and simulations to create a microcircuit that would meet the dimensions stated in the Engineering Requirements.

5.1.1 Block Diagrams

The following section consists of block diagrams ranging from Level 0 to Level 2 to show the design process used to create the Soil WSN systems and subsystems.

5.1.1.1 *Level 0 Block Diagram* *AA, RK, AW*

The wireless sensor network components can be categorized into three different functional blocks, as seen in Figure 11.

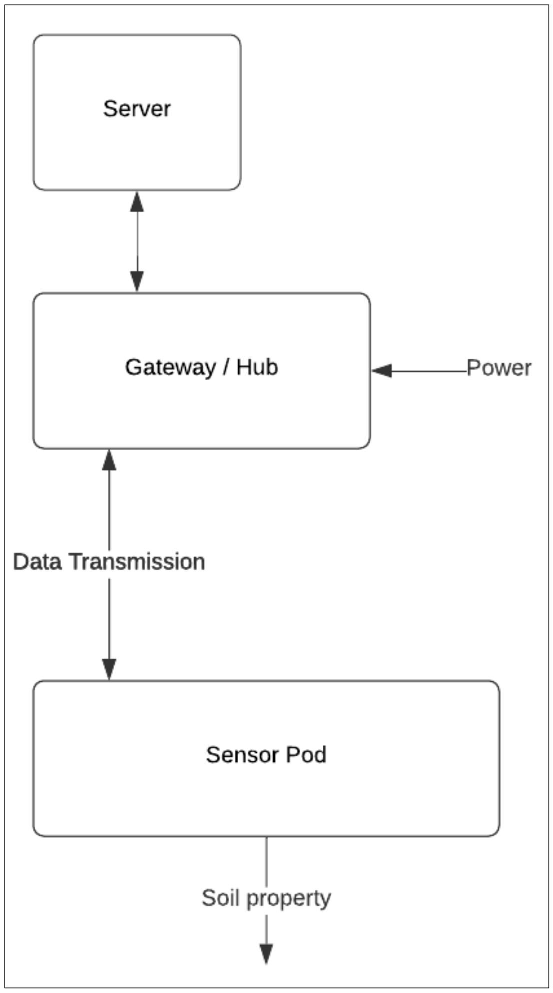


Figure 11: Level 0 Block Diagram of Soil Sensor Network.

The functions of the Sensor Pod, Gateway, and Server are shown in Tables 9-11.

Table 9: Level 0 FR Table: Sensors.

Module	Sensor Pod
Designer	Andrea Wyder, Luke Farnsworth
Inputs	Power Wake-Up Signal (Microchip)

	Soil Properties
Output	Acquired Soil Data Location of Pods
Description	Wireless sensors units acquire measurements from soil to transmit to a gateway.

Table 10: Level 0 FR Table: Gateway/Hub.

Module	Gateway/Hub
Designer	Aléxis Alves, Andrea Wyder
Inputs	Battery Powered Soil Data Location of Sensor Pods
Output	Soil Data communicated via HTTP
Description	The gateway will communicate with Soil Sensors and relay that information to the database.

Table 11: Level 0 FR Table: Server.

Module	Server
Designer	Aléxis Alves, Ross Klonowski
Inputs	Soil data communicated via HTTP from gateway
Output	Soil data communicated via HTTP to software application
Description	The server enables remote use of the automated irrigation system, as well as serves as a database.

5.1.1.2 *Level 1 Block Diagram* ***AA, RK, AW***

The three tiers of Level 0 were broken down further as seen in Figure 12.

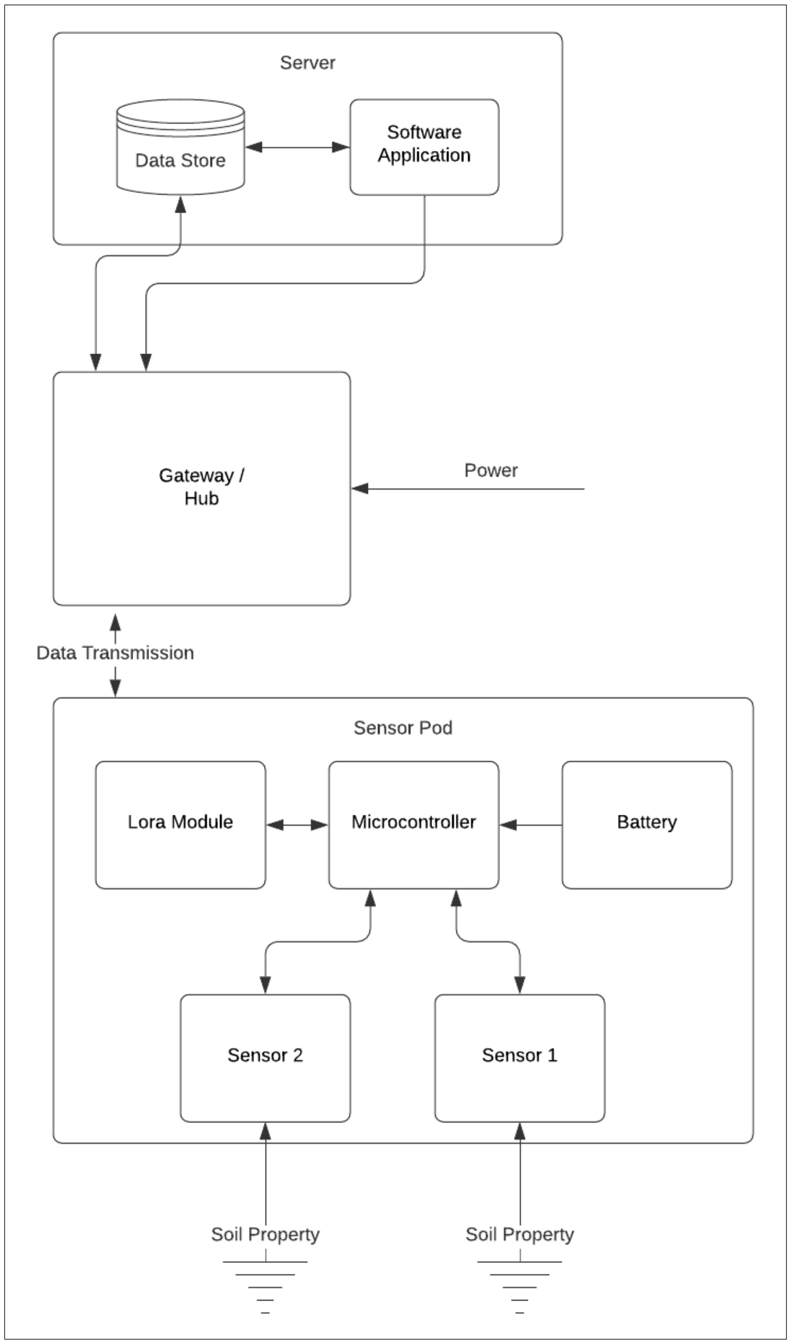


Figure 12: Level 1 Block Diagram of Soil Sensor Network.

The first tier, the Sensor Pod, consists of a Lora module, microcontroller, battery, and two soil property sensors. The functional requirements for the first tier are listed in Tables 12-16.

Table 12: Level 1 FR Table: Battery.

Module	Battery
Designer	Luke Farnsworth, Andrea Wyder
Inputs	None
Output	Power
Description	The battery will power the microprocessor.

Table 13: Level 1 FR Table: Sensor Pod Microprocessor.

Module	Sensor Microcontroller
Designer	Aléxis Alves, Ross Klonowski
Inputs	Battery Power, Sensor Data, LoRa communication
Output	LoRa communication, Sensor Interrupt
Description	Collects sensor readings for data transmission.

Table 14: Level 1 FR Table: Sensor Pod Lora Module.

Module	Sensor Pod Lora Module
Designer	Aléxis Alves, Ross Klonowski
Inputs	Battery Power, Data from Microprocessor
Output	RF Communication
Description	The LoRa module communicates to the gateway.

Table 15: Level 1 FR Table: Sensor 1.

Module	Sensor 1
Designer	Luke Farnsworth, Andrea Wyder
Inputs	Power Wake-Up Signal (Microchip) Soil Properties
Output	Acquired Soil Data Location of Pods
Description	Sensor 1 will collect data from the soil and then transmit the soil properties to the microprocessor.

Table 16: Level 1 FR Table: Sensor 2.

Module	Sensor 2
Designer	Luke Farnsworth, Andrea Wyder
Inputs	Power Wake-Up Signal (Microchip) Soil Properties
Output	Acquired Soil Data Location of Pods
Description	Sensor 2 will collect data from the soil and then transmit the soil properties to the microprocessor.

The second tier, the gateway/hub, consists of a Lora module and microcontroller, and power source. These functional requirements for the second tier are listed in Tables 17 and 18.

Table 17: Level 1 FR Table: Gateway Lora Module.

Module	Gateway Lora Module
Designer	Aléxis Alves, Ross Klonowski
Inputs	Power Source
Output	New sensor readings from Sensor Pod™

Description	Transmits sensor readings to gateway

Table 18: Level 1 FR Table: Gateway Microcontroller.

Module	Gateway Microcontroller
Designer	Aléxis Alves, Ross Klonowski
Inputs	Power, Ethernet, Lora Data communication
Output	Data
Description	The microcontroller further processes data for upload to server

The third tier, the server, consists of data storage and a software application, which are listed in Tables 19-20.

Table 19: Level 1 FR Table: Data Storage.

Module	Data Storage
Designer	Aléxis Alves, Ross Klonowski
Inputs	New Sensor Data
Output	Requested Sensor Data
Description	Persistent storage of Sensor Pod readings.

Table 20: Level 1 FR Table: Software Application.

Module	Software Application
Designer	Aléxis Alves, Ross Klonowski
Inputs	Farm data from Web Server
Output	An Interface for the Automated Irrigation System
Description	A software application will communicate with a web server for use on the farm or offsite.

5.1.1.3 *Level 2 Block Diagram* ***AA, LF, RK, AW***

The Sensor Pod was further broken into a second level to analyze designed sensor components. This can be seen in Figure 13.

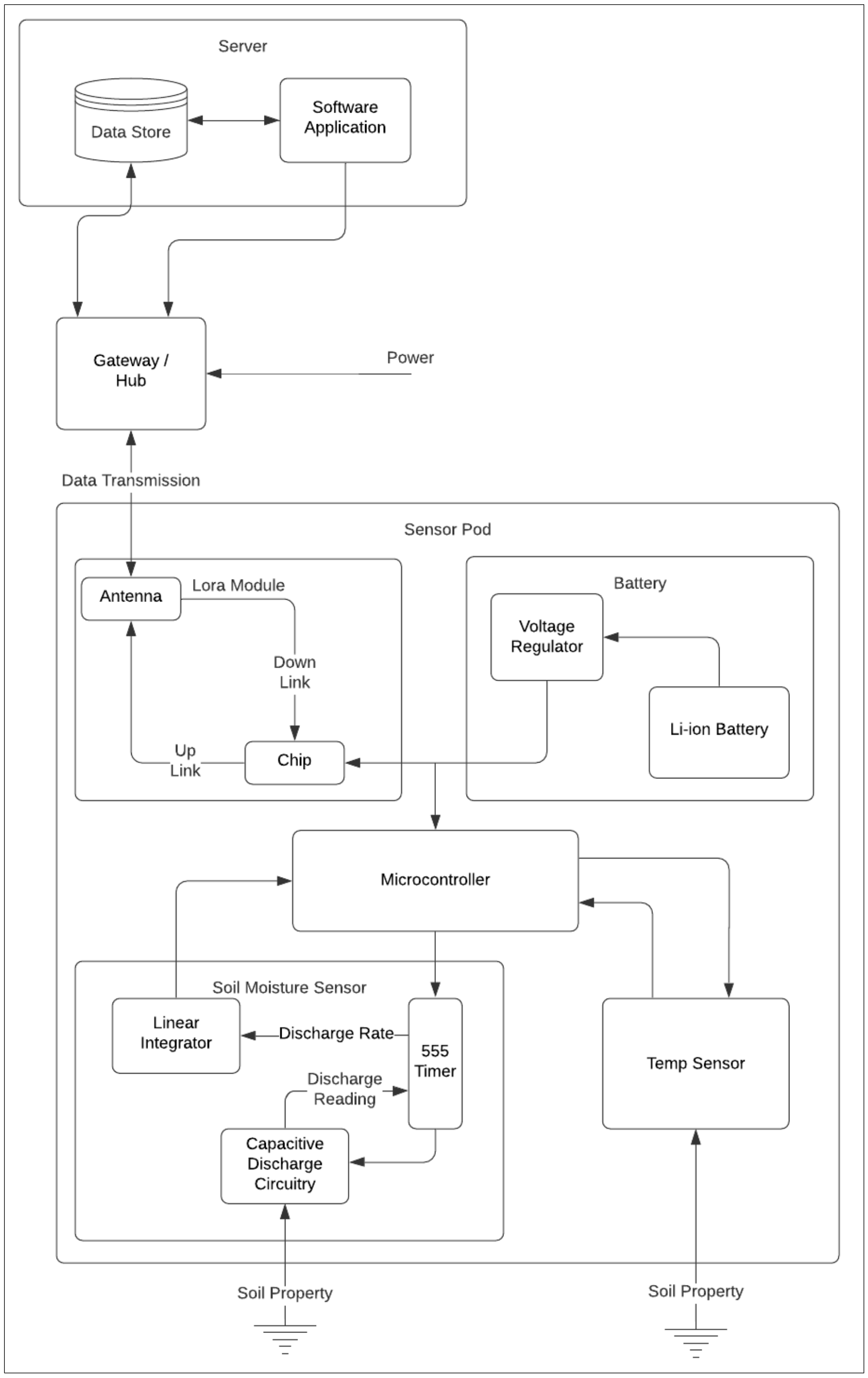


Figure 13: Level 2 Block Diagram for Soil Sensor Network.

The LoRa Module can be further broken down into the antenna and transceiver module, as explained in Tables 21-22.

Table 21: Level 2 FR Table: Antenna.

Module	Antenna
Designer	Andrea Wyder
Inputs	Data Signal
Output	Data Signal
Description	The Antenna will act as a means for the Sensor Pod and Gateway to wirelessly send and receive data.

Table 22: Level 2 FR Table: Transceiver Module.

Module	Transceiver Module
Designer	Aléxis Alves, Ross Klonowski
Inputs	Power UART Data Communication from Main Processor Received Signal from LoRa Antenna
Output	Data Signal
Description	The transceiver module will implement the LoRaWAN Communication protocol at the physical layer to enable long distance communication between Sensor Pod and Gateway.

Within the battery exists a voltage regulator and Lithium-Ion battery. The voltage regulator maintains how much voltage is distributed to the components. The functionality of the regulator is described in Table 23.

Table 23: Level 2 FR Table: Voltage Regulator.

Module	Voltage Regulator
Designer	Luke Farnsworth, Andrea Wyder
Input	Voltage
Output	Voltage
Description	The regulator will regulate the voltage that is introduced to the system.

The soil moisture sensor is comprised of a linear integrator, capacitive discharge circuit, and a 555 timer. The voltage discharge is fed through the timer to create a discharge rate, which is then connected to a linear integrator that converts the discharge rate to a ramp function so it can be used as an analog input signal to the microcontroller. Further details for these components are found in Tables 24-26.

Table 24: Level 2 FR Table: Linear Integrator.

Module	Linear Integrator
Designer	Luke Farnsworth, Andrea Wyder
Input	Voltage Discharge Rate
Output	Analog Voltage Signal as Ramp Function
Description	Since the amplitude of the voltage increases linearly over time, the linear integrator will convert the discharge rate to a ramp function and will send the function back as an analog signal.

Table 25: Level 2 FR Table: Capacitive Discharge Circuitry.

Module	Capacitive Discharge Circuitry
Designer	Luke Farnsworth, Andrea Wyder
Input	Voltage
Output	Voltage
Description	Discharge rate will determine the moisture of the soil.

Table 26: Level 2 FR Table: NE555 Timer.

Module	NE555 Timer
Designer	Luke Farnsworth, Andrea Wyder
Input	Voltage
Output	Voltage Discharge Rate
Description	An NE555 timer will be used to measure the time it takes for the capacitor to discharge.

Another tool on the sensor pod is the temperature sensor as seen in Table 27. This will be an off-the-shelf sensor that meets all necessary power requirements.

Table 27: Level 2 FR Table: Temperature Sensor.

Module	Temperature Sensor
Designer	Luke Farnsworth
Inputs	Voltage Temperature
Output	Analog Voltage Signal
Description	The temperature sensor will read the temperature of the soil.

5.1.1.4 *Level 3 Block Diagram* *AA, AW*

Examining the Level 2 Block Diagram more closely through circuits and simulation, it was determined that the linear integrator circuit in the Sensor Pod was unnecessary. A third iteration of the Hardware Block diagram can be seen in Figure 14.

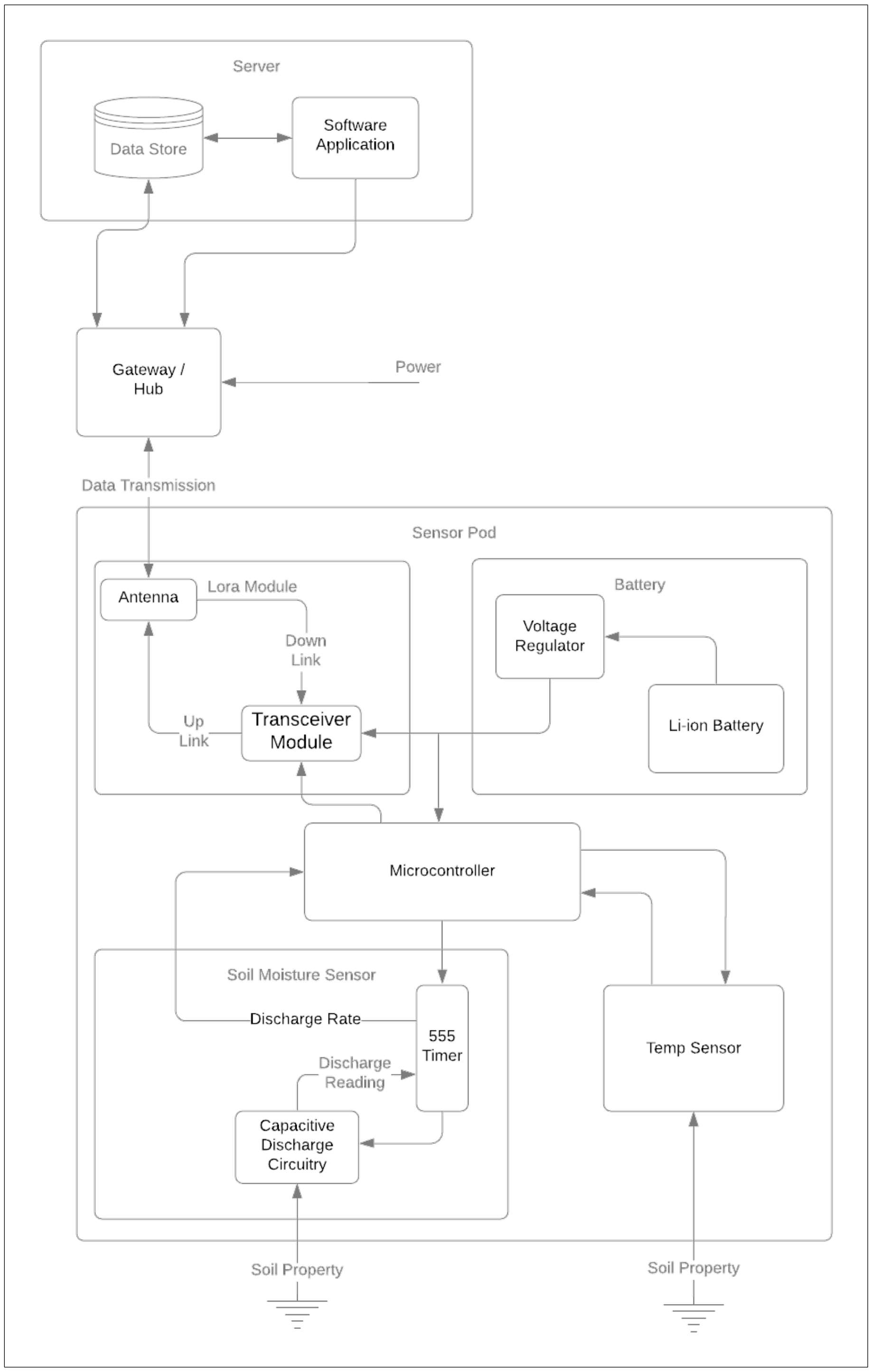


Figure 14: Level 3 Block Diagram for Soil Sensor Network.

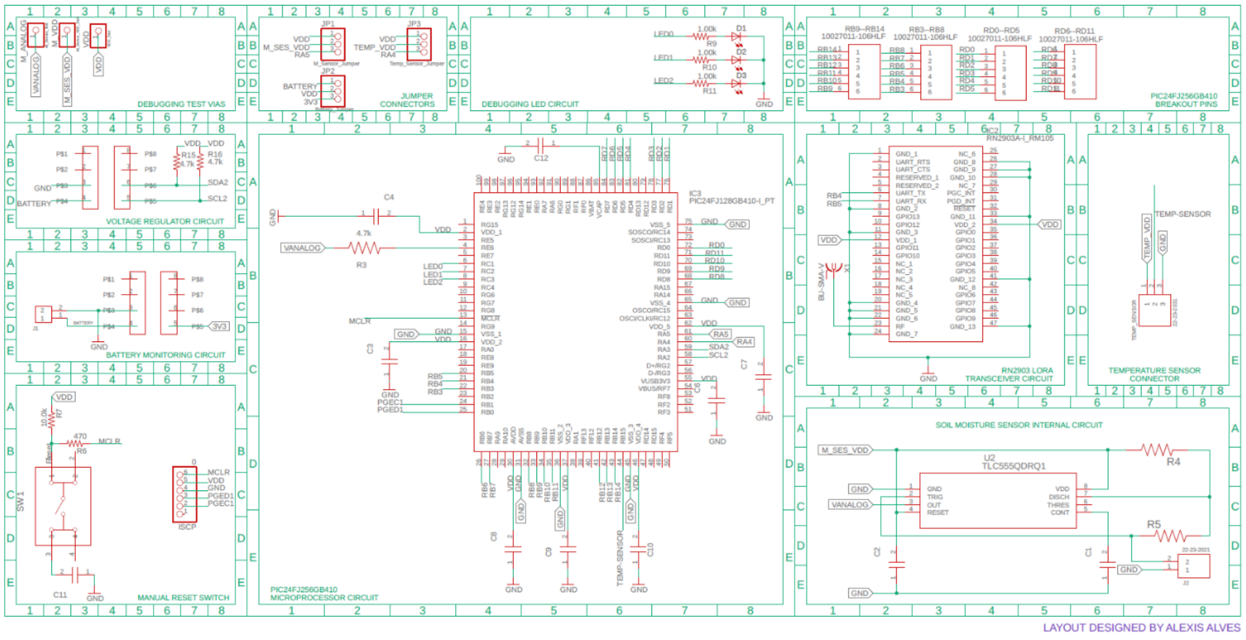
No new functional requirement tables are shown for this diagram because they are the same as the tables for the Level 2 Block Diagram. The only change is that the discharge rate exiting the 555 Timer will go straight into the analog input of the microcontroller rather into a linear circuit.

5.1.2 Schematics

Once the hardware block diagrams were completed, schematics were created for each subdivision of the level 3 block diagram to gain a better understanding of how to connect all of the components together.

5.1.2.1 *Circuit Overview* ***AW***

The hardware of the Soil Sensor Network is broken down into eleven different components, as seen in Figure 15.



DEBUGGING TEST VIAS
JUMPER CONNECTORS
DEBUGGING LED CIRCUIT
PIC24FJ256GB410 BREAKOUT PINS
VOLTAGE REGULATOR CIRCUIT
BATTERY MONITORING CIRCUIT
MANUAL RESET SWITCH
PIC24FJ256GB410 MICROPROCESSOR CIRCUIT
RN2903 LORA TRANSCEIVER CIRCUIT
TEMPERATURE SENSOR CONNECTOR
SOIL MOISTURE SENSOR INTERNAL CIRCUIT
LAYOUT DESIGNED BY ALEXIS ALVES

Figure 15: EagleCAD Soil Sensor Network Circuit Overview.

The circuit is powered by two 3.7V, 2800 mA-H batteries designed to last the duration of a growing season. The batteries are connected to a battery PCB that is separate from the main board, and the connector for the battery can be seen in the JUMPER CONNECTORS block. Since all components operate at 3.3V rather than 3.7 V, a voltage regulator was implemented to ensure the voltage supplied to the circuit remained at a constant 3.3V. The voltage regulator was placed on a separate breakout board as well and can be seen in the VOLTAGE REGULATOR CIRCUIT block.

From the voltage regulator, the power is supplied to microcontroller, as seen in the PIC24FJ256GB410 MICROPROCESSOR CIRCUIT block. The microcontroller collects data from the sensors and transfers the information to the LoRa module, which is seen in the RN2903 LORA TRANCEIVER MODULE CIRCUIT. The two sensors connected to the microcontroller are a soil moisture sensor and a temperature sensor. The soil moisture sensor has both internal and external circuitry. The internal circuitry can be seen in the SOIL MOISTURE SENSOR CIRCUIT block. The external circuitry attaches to the connector at the right side of the block. The temperature sensor is on an external board, but the connector can be seen in the TEMPERATURE SENSOR CONNECTOR block. The sensor data, once transferred to the LoRa transceiver, is transmitted to the gateway through an antenna, also located in the LORA TRANCEIVER MODULE CIRCUIT block, which in turn is transmitted and stored in the database by a network server. The blocks pertaining to the RESET SWITCH, DEBUGGING TEST VIAS, DEBUGGING LED CIRCUIT, and PIC24FJ256GB410 BREAKOUT PINS are additional circuits integrated for testing and debugging.

5.1.2.2 *Battery Monitor* *AW*

The battery monitor is beneficial to the Sensor Pod circuit design in order to monitor voltage, current, and temperature of the battery to give an accurate estimation of when the battery will fully deplete so that the farmer will know if the pod will need to be retrieved in order to recharge the battery.

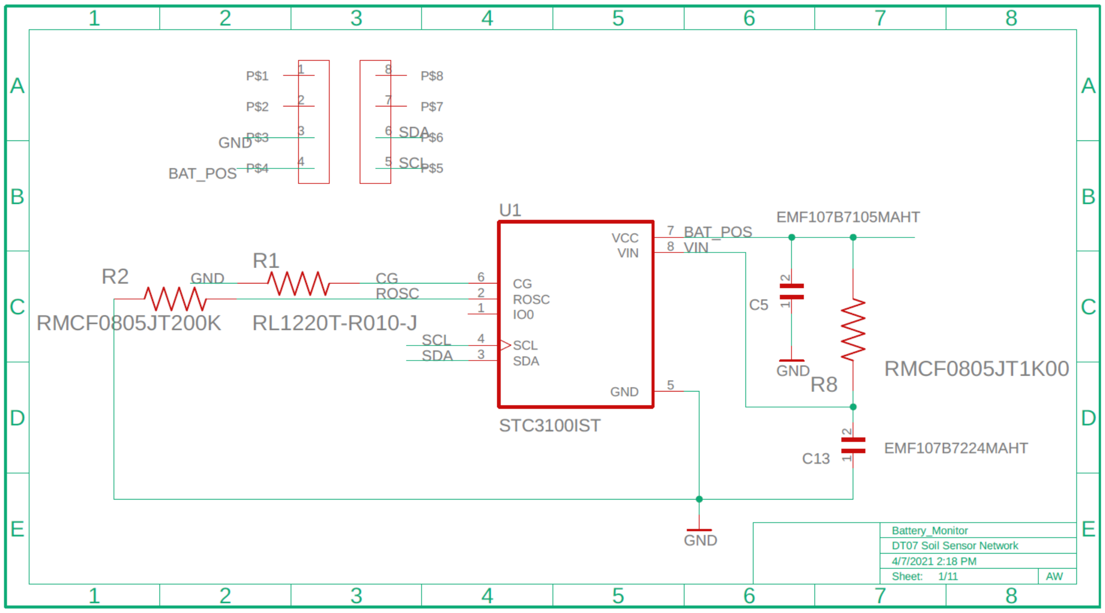


Figure 16: EagleCAD STC3100IST Battery Monitor.

Following the recommended component values on the STC3100IST datasheet, a 200kΩ resistor R2 was connected to the oscillator input pin ROSC to shift the voltage rails from GND and V_{DD} to V+ and V- with a bias resistance so that the monitoring unit could properly read and interpret the voltage coming from the battery. A 10mΩ resistor R1 was connected to the gas gauge current sense pin CG to monitor the current draw of the circuit. A 1uF capacitor C5 was connected to the V_{CC} pin to aid in noise reduction. A 1kΩ resistor R8 was coupled with a 220uF capacitor C13 and connected to V_{CC}, V_{IN}, and GND in order to filter the voltage being inputted to the main power pin. The SDL an SDA pins are read/write pins that are used to control the shift registers of the I2C interface.

The STC3100IST has 32 RAM registers in order to store information concerning battery life, discharge rates, charging cycles, etc. so that the farmer knows if the battery needs charged or replaced. The battery monitor maintains its power-on stage down to 2V. Upon each new battery connected, the monitor assigns it a new ID in order to keep track of which battery is connected and what that specific battery's life cycle has been.

5.1.2.3 *Voltage Regulator*

As stated previously, the voltage regulator chosen was required to have a supply voltage of around 3.7V and output a voltage of 3.3V. The two main design iterations are discussed in detail in the following section.

5.1.2.3.1 Voltage Regulator Phase 1 LF, AW

To meet these specifications, the XC9140A331MR-G boost switching regulator was chosen to be used in the first design iteration.

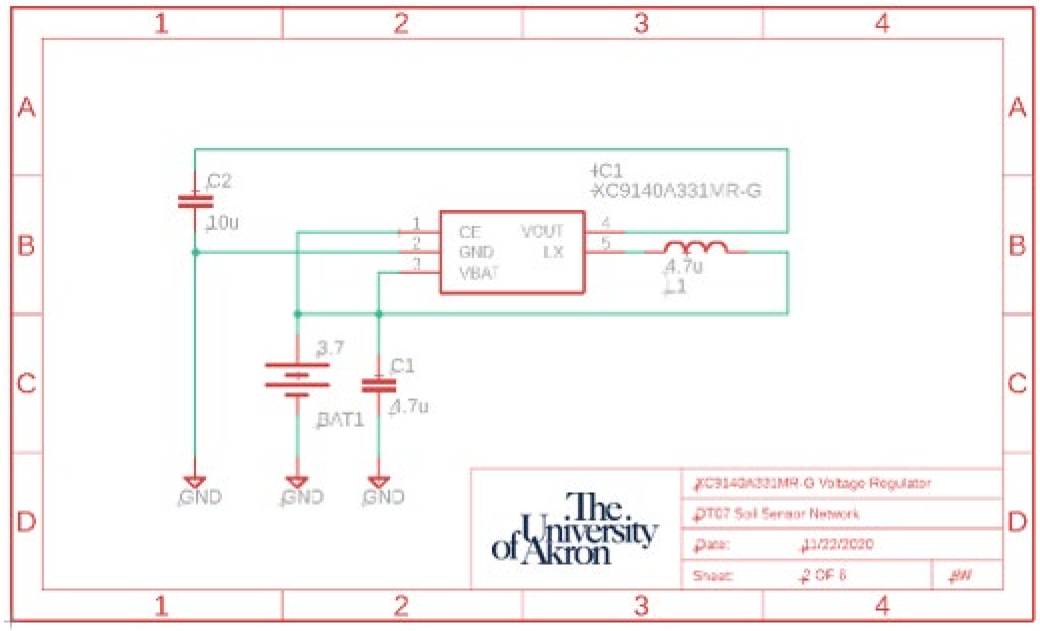
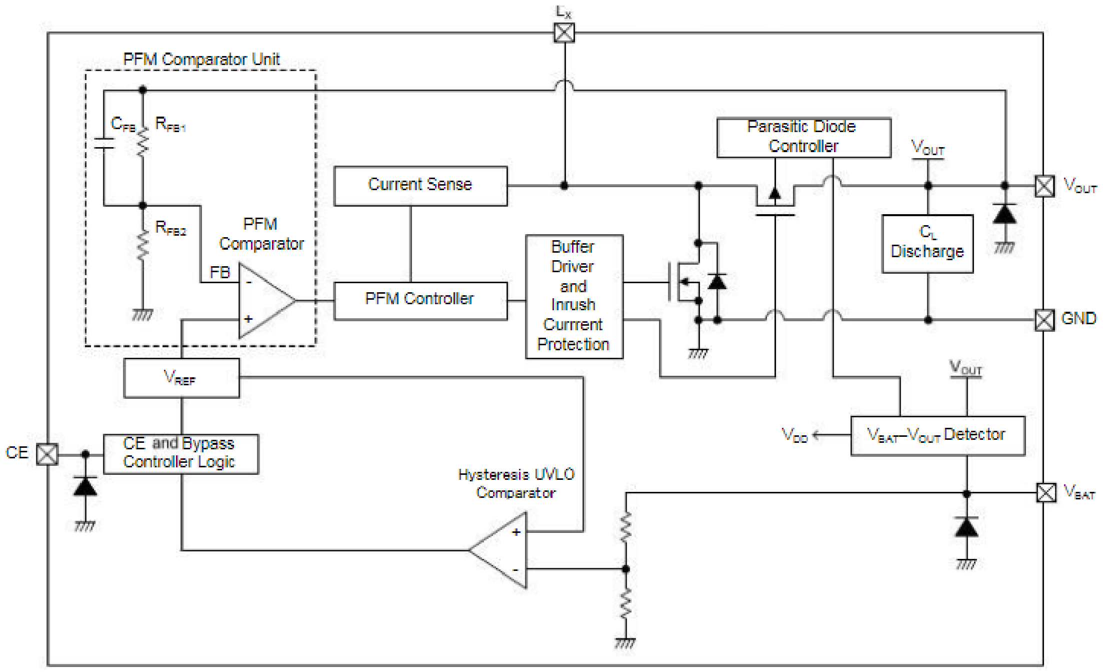


Figure 17: EagleCAD XC9140A331MR-G Boost Voltage Regulator.

The voltage regulator pin 2 *CE* is powered by the battery. An LC circuit comprised of *L1* and *C1* is attached to pin 3 *Lx* to aid in switching, and a capacitor *C2* filters out any unwanted noise between the output pin *Vout* and the ground pin *GND* before returning to ground.



L_X
PFM Comparator Unit
C_{FB}
R_{FB1}
R_{FB2}
PFM Comparator
FB
Current Sense
PFM Controller
Buffer Driver and Inrush Current Protection
Parasitic Diode Controller
V_{OUT}
C_L Discharge
GND
V_{REF}
CE
CE and Bypass Controller Logic
Hysteresis UVLO Comparator
V_{DD}
V_{BAT}–V_{OUT} Detector
V_{BAT}

Figure 18: Internal Circuitry for Voltage Regulator XC9140A331MR-G. Image retrieved from https://www.digikey.com/htmldatasheets/production/1228326/0/0/1/xc9140-series.html.

The importance of using a 4.7 µH inductor for the *Lx* pin is for inrush current protection. Lx is the input switch, and when inrush current is introduced to the input switch melting/blow-up can be caused. The output of the circuit is connected to the microcontroller to provide a 3.3V input.

5.1.2.3.2 Voltage Regulator Phase 2 AW

Although the boost regulator proved to be successful, after testing and reevaluating the purpose of the regulator for the project, it was determined that the boost was not the most optimal solution. The batteries chosen could fully charge up to 4.2V. Since the voltage threshold on a few of the components was 3.7V, if only a boost regulator were to be used those components had the potential of being burned out. As the name implies, a boost regulator boosts the voltage when it falls below a certain threshold (i.e.: 3.3V); it has no effect on the circuit when the voltage is above that threshold.

The other two types of regulators researched were linear and buck-boost. A linear regulator is an easy solution – it does not require complex circuitry, extensive reading into a datasheet, or additional programming. However, one of the main engineering requirements is low power such that the pods will last an entire growing season. A linear regulator requires the same amount of current to operate at both higher and lower voltages; the additional power supplied to the circuit is dissipated as heat. The purpose of a regulator in this project is to keep the power consumption at a minimum, and although a linear regulator will maintain the correct voltage being supplied to the circuit, the operation of a linear regulator is extremely inefficient, consuming more power than what would be consumed if the regulator didn't exist in the circuit. Therefore, this is not a viable solution for the project.

The third type of regulator is a buck-boost regulator. A buck-boost regulator sets a target voltage given the pin configuration. If the supplied voltage is larger than the target voltage, the regulator steps the voltage down to the target voltage. Similarly, if the supplied voltage is smaller than the target voltage, the regulator steps the voltage up to the target voltage. This design is power efficient and optimizes battery life. The design chosen was a STBB1-APUR buck-boost regulator, as seen in Figure 19.

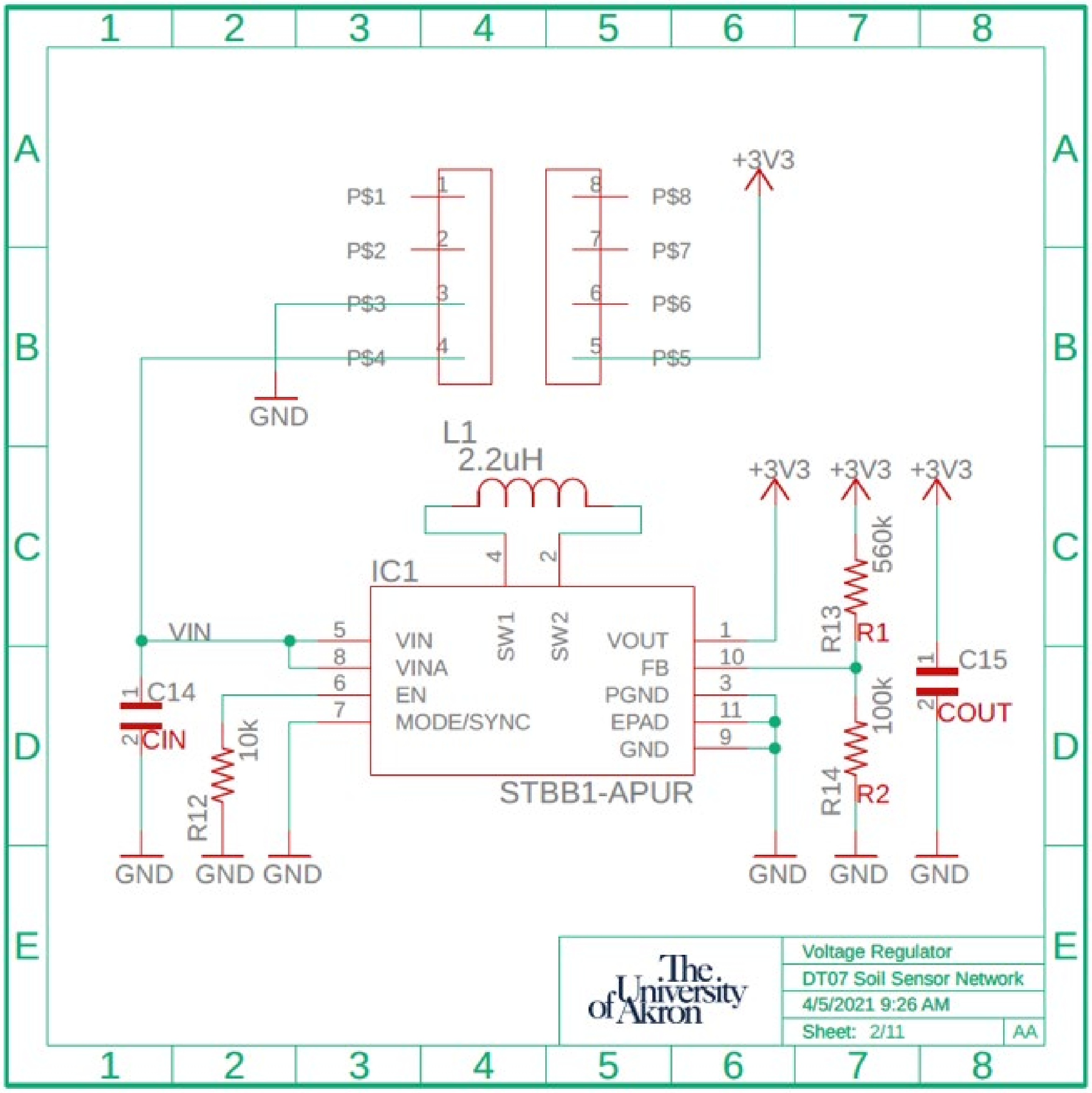
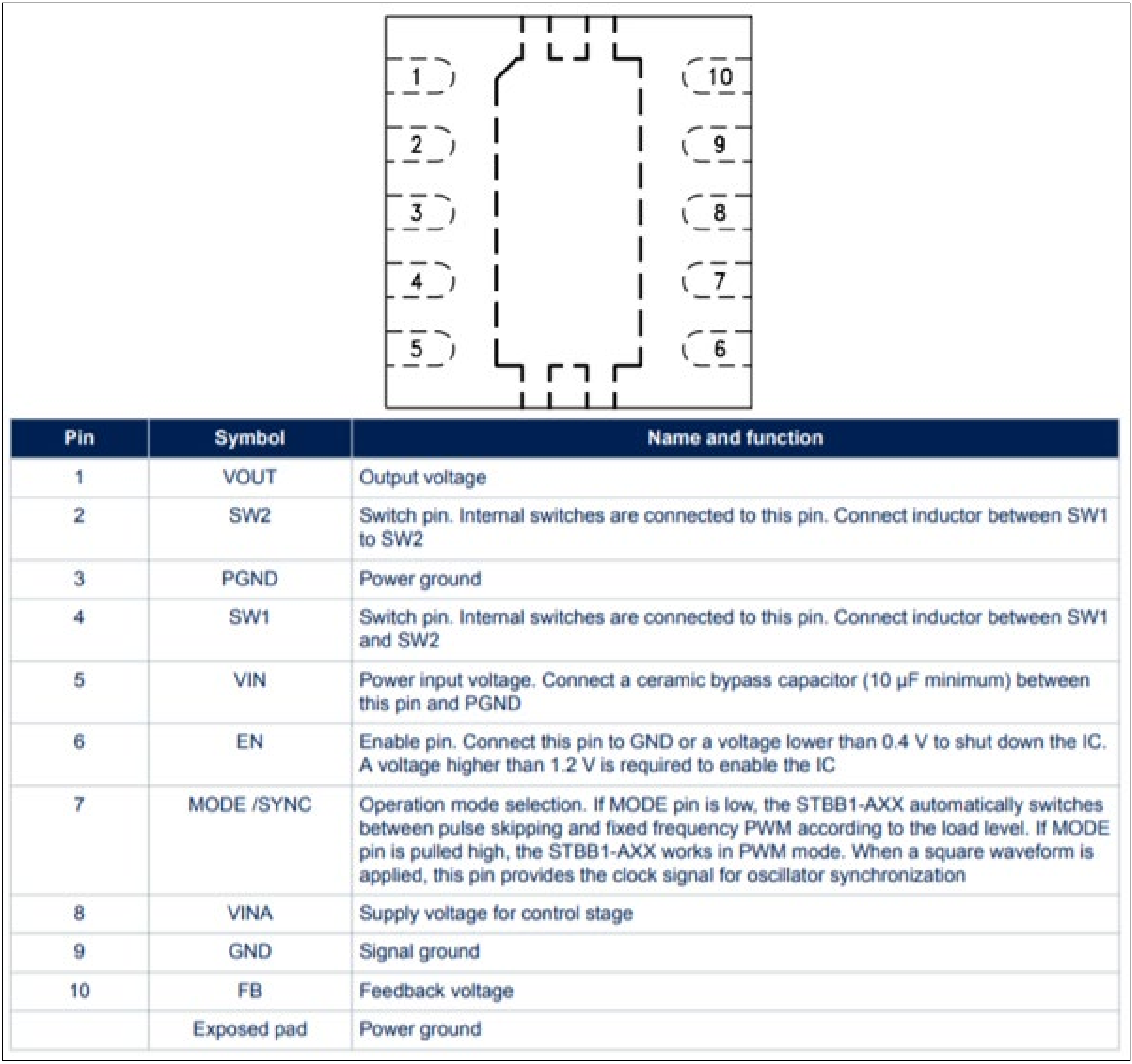


Figure 19: EagleCAD STBB1-APUR Voltage Regulator.

The orientation as specified from the datasheet is shown in Figure 20.



Pin	Symbol	Name and function
1	VOUT	Output voltage
2	SW2	Switch pin. Internal switches are connected to this pin. Connect inductor between SW1 to SW2
3	PGND	Power ground
4	SW1	Switch pin. Internal switches are connected to this pin. Connect inductor between SW1 and SW2
5	VIN	Power input voltage. Connect a ceramic bypass capacitor (10 µF minimum) between this pin and PGND
6	EN	Enable pin. Connect this pin to GND or a voltage lower than 0.4 V to shut down the IC. A voltage higher than 1.2 V is required to enable the IC
7	MODE /SYNC	Operation mode selection. If MODE pin is low, the STBB1-AXX automatically switches between pulse skipping and fixed frequency PWM according to the load level. If MODE pin is pulled high, the STBB1-AXX works in PWM mode. When a square waveform is applied, this pin provides the clock signal for oscillator synchronization
8	VINA	Supply voltage for control stage
9	GND	Signal ground
10	FB	Feedback voltage
	Exposed pad	Power ground

Figure 20: Buck-Boost Voltage Regulator STBB1-APUR Pinout. Image retrieved from https://www.st.com/content/ccc/resource/technical/document/datasheet/20/a6/10/e0/63/85/43/c1/DM00037824.pdf/files/DM00037824.pdf/jcr:content/translations/en.DM00037824.pdf.

The two configuration pins that dictate how the regulator operates are pins 6 and 7. Pin 6 is the enable pin which communicates to the regulator when to turn off and on. The enable pin was pulled high (V_{DD}) so that the boost mode would be activated once the regulator reached at least a 1.2V input. Pin 7 is the mode select (sync) which communicates to the regulator how often to oscillate between frequencies. The mode select pin was originally pulled low so that the regulator would operate based off the load, however in reality, the regulator got stuck between the skipping and fixed frequency pulses and did not operate as intended. The circuit was modified on the PCB and the regulator then performed as expected.

5.1.2.4 *Microcontroller* ***AA, RK, AW***

From the voltage regulator, power is supplied to the microcontroller through the Master Clear (MCLR) pin. The voltage divider *R3* and *R4* coupled with capacitor *C11* are used for resetting and programming the microprocessor. The capacitors *C6-C10* and *C12-C14* are decoupling capacitors that are used to suppress high frequency noise. The microcontroller is programmed with a MPLAB In-Circuit Debugger/Programmer via the RJ45 connector coming in to pins 24 and 25.

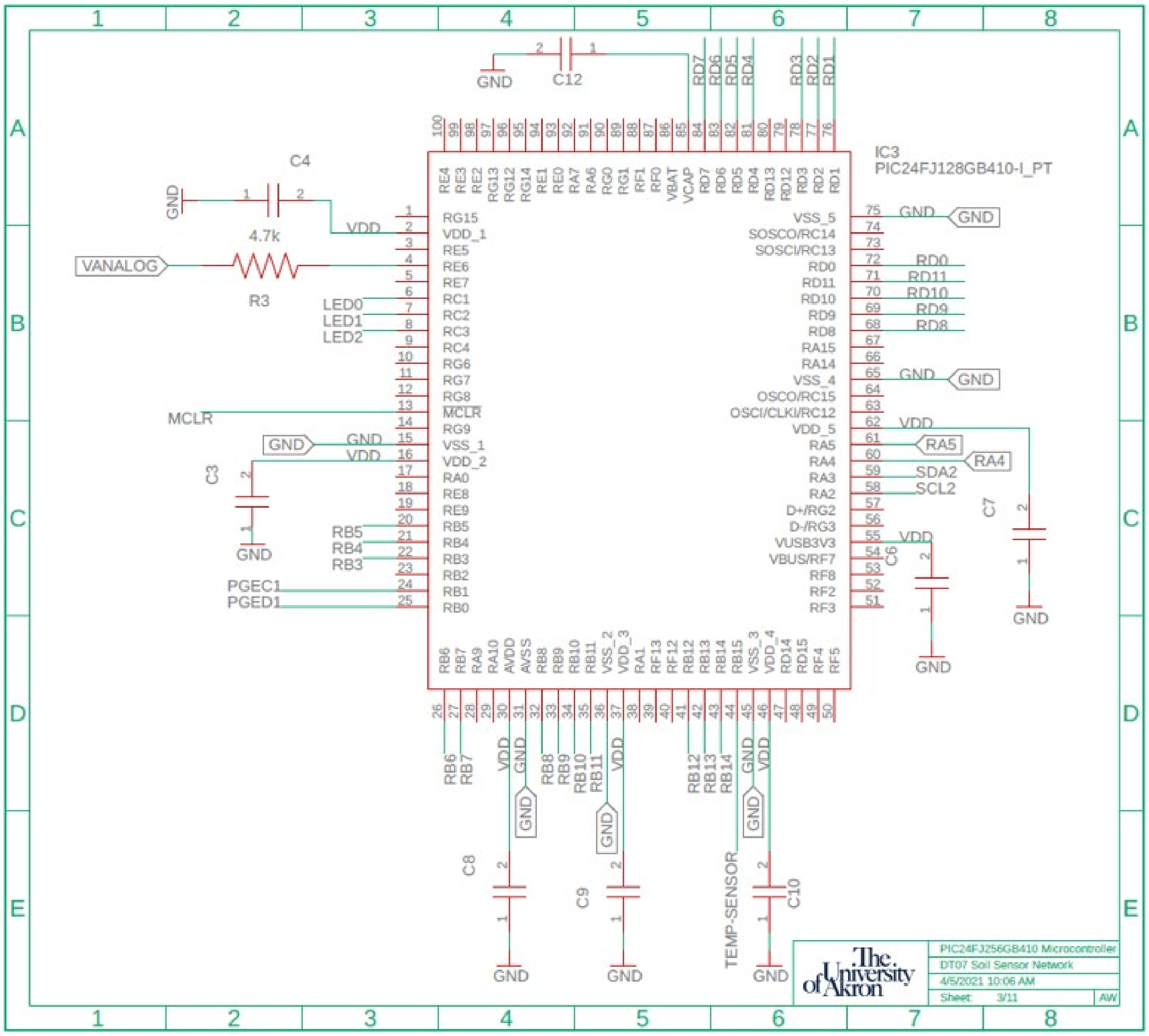


Figure 21: EagleCAD PIC24FJ128GB410 Microcontroller.

The microcontroller supplies power to all other components of the circuit. The analog input pins 41 and 44 read in data from the temperature sensor and soil moisture sensor.

The collected data is then sent to the LoRa module via pins 78 and 83.

5.1.2.5 *LoRa Module* ***AA, RK, AW***

The LoRa module has three main connections, which are for data, power, and signal transmission, as seen in Figure 22.

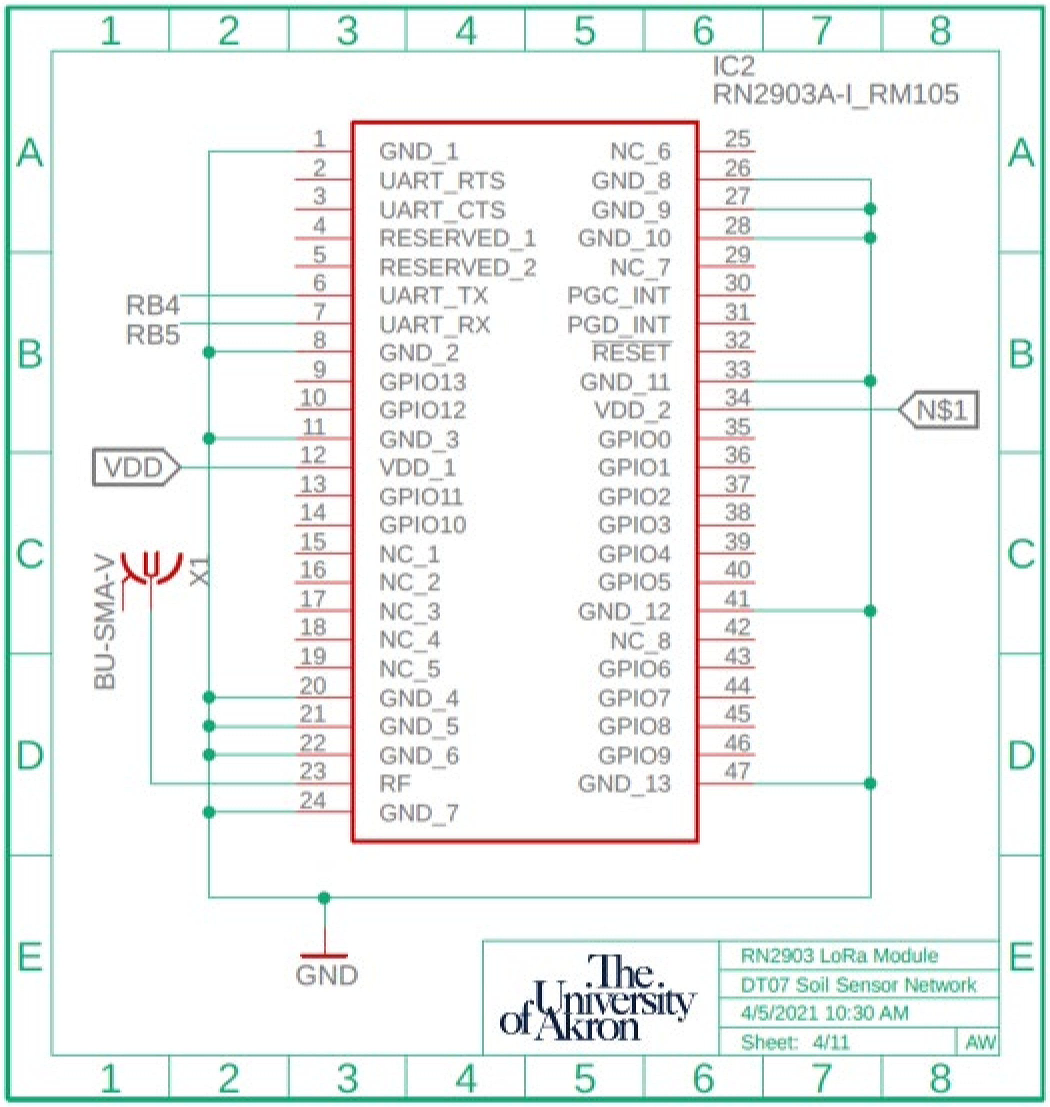


Figure 22: EagleCAD RN2903 LoRa Transceiver.

Like the other components, this module is supplied with 3.3V. For data communication with the microprocess, one UART port is used to transmit serial data. The third connection that is made is at the RF port of the LoRa transceiver, which connects to a 3”, quarter-wavelength monopole antenna.

5.1.2.6 *Soil Moisture Sensor* *AW*

As stated previously, a capacitive soil moisture sensor was designed and constructed. Many capacitive soil moisture sensor circuits were analyzed to aid in design.

5.1.2.6.1 Moisture Sensor Phase 1 AW

Capacitive soil moisture sensors can be broken down into three basic components: a timer, a capacitive discharge circuit, and a linear integrator. The schematic for a simple timer-capacitive discharge circuit can be seen in Figure 23.

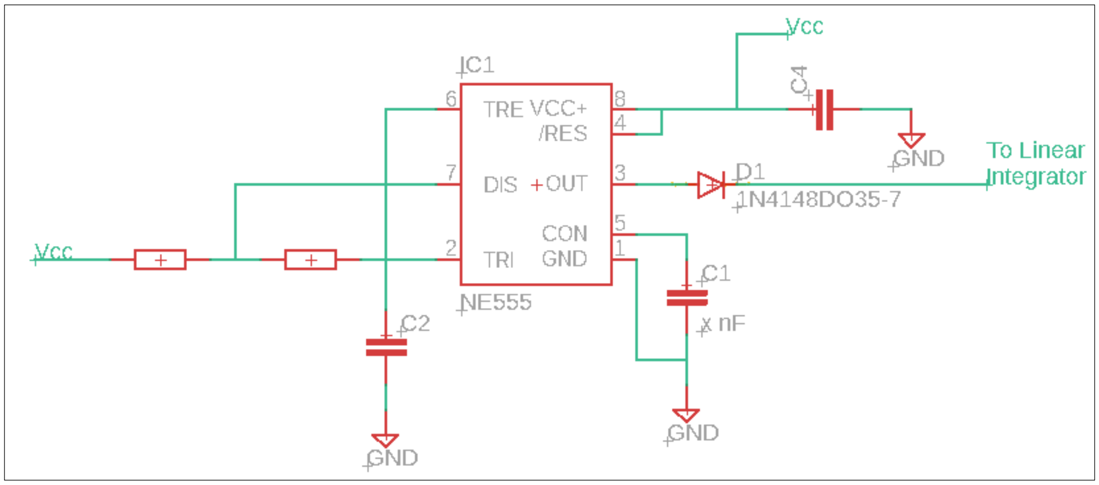


Figure 23: EagleCAD Capacitive Soil Moisture Sensor: Iteration I.

The timer is coupled with the discharge circuit to measure discharge rate over time. The capacitors *C2* and *C4* are placed in parallel to aid in noise reduction. Two resistors acting as a voltage divider are placed after the input voltage to ensure the correct power is being inputted to the timer. The diode *D1* is put in place as a current buffer so that no current will come back through the output of the timer.

The discharge circuit in its simplest form is a capacitor *C1* on the order of nano Farads. The timer clocks how long the capacitor takes to discharge. The discharge rate is then fed into a linear integrator. There are a few different methods that exists for creating linear integrator circuits; the most common method is using an op-amp connected to resistors and capacitors. Since the amplitude of the voltage increases linearly with time, the integrator converts the linear function to a ramp function which can then be read by the microcontroller as an analog input.

5.1.2.6.2 Moisture Sensor Phase 2 AW

To further the soil moisture sensor design, calculations were completed to determine the values for each component. The NE555 timer used previously was replaced with a TLC555 timer in order to keep consistent with the voltage values of the other components of the sensor pod. All other components (microcontroller, LoRa module, etc.) operate at a minimum of 3.3V. When examining the datasheet for the NE555 timer, it was found that the timer had a minimum operating voltage of 5V. Because of this, the NE555 timer was replaced with a TLC555 timer. The two timers are essentially the same, but the TLC555 timer has the ability to operate at 3.3V.

To determine the resistor values acting as a voltage divider for the timer, the TLC555 datasheet was referenced [20]. The timer is used as a DC oscillator in the application of a capacitive soil moisture sensor, and as explained in the Electronic Analysis section, is created using an astable circuit. The astable circuit located in the datasheet can be seen in Figure 24.

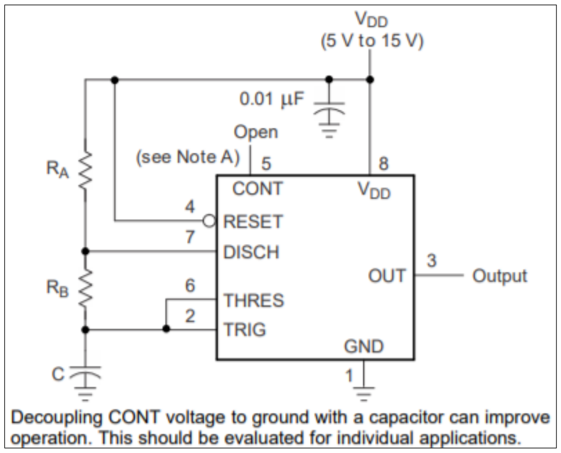

V_{DD}
(5 V to 15 V)
0.01 µF
Open
(see Note A)
5
8
R_A
CONT
V_{DD}
4
RESET
7
DISCH
3
OUT
Output
R_B
6
THRES
2
TRIG
GND
C
1
Decoupling CONT voltage to ground with a capacitor can improve operation. This should be evaluated for individual applications.

Figure 24: TLC555 Timer Datasheet: Astable Circuit. Image retrieved from https://www.ti.com/lit/ds/symlink/tlc555.pdf?HQS=TI-null-null-digikeymode-df-pf-null-wwe&ts=1603553851426.

Reading further into the datasheet as to what signal each input requires, the voltage being supplied to the discharge pin is typically around one-third of the voltage supplied to the input V_{CC}. The threshold input monitors the voltage across the timing capacitor and determines when the circuit should oscillate from state one to state two [20]. Since R_A and R_B act as a voltage divider, and the input does not have a maximum voltage threshold, Equation 19 was used to find a ratio of the values of R_A and R_B.

$$I\left(\frac{1}{3}R_A + \frac{2}{3}R_B\right) = Vcc = IR \rightarrow \frac{1}{3}R_A + \frac{2}{3}R_B = R \tag{19}$$

By using Equation 19, it was noted that the ratio of the resistors was more important than the size of the resistors. For 555 timers, R_A and R_B are on the order of kilohms. The standard resistors that satisfy the 1:2 ratio are 150kΩ and 330kΩ resistors. After simulating the circuit, it was found that this ratio was not accurate for the circuit at hand, and the 330kΩ resistor had to be exchanged for a 3300kΩ (or 3.3MΩ).

The capacitor *C* is used to eliminate noise in most cases, but for the soil sensor application, *C* is a representation of the capacitive discharge circuit. For the sensor pod prototype, the timer will measure the discharge rate of a parallel plate capacitor. The capacitance needed for *C* is calculated using Equation 20.

$$C = \frac{\varepsilon S}{d} = \frac{(5\varepsilon_0)S}{d} = \frac{(5*8.854*10^{-12})(0.05*0.02)}{0.005} = 8.854\ [pF] \approx 10\ [pF] \tag{20}$$

It is recommended by the TLC555 timer datasheet that the control voltage input be connected to at least a 10nF capacitor if it is not being used. For this application, a 100nF capacitor was used to connect the control input to ground. The second iteration of the soil sensor schematic is located in Figure 25.

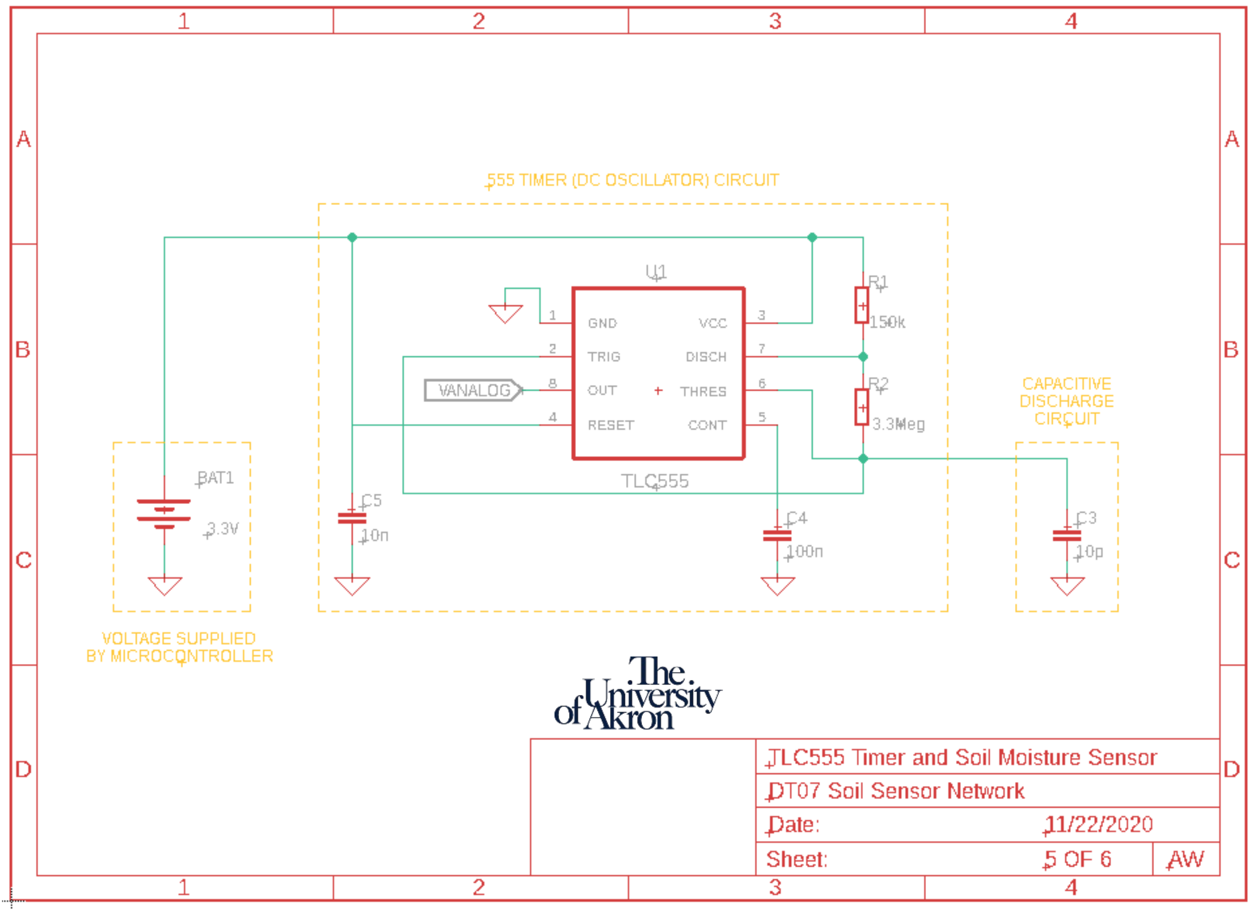


Figure 25: EagleCAD Soil Moisture Sensor: Iteration II.

The linear integrator circuit was omitted on the second iteration of schematics. Although voltage is one method that can be used to measure discharge rate and be sent through a linear integrator circuit to be inputted back to the microprocessor, it may not be the most straightforward method of measuring the capacitor. After taking a closer look at the linear integration circuit, it was determined that more than an integrating op-amp is needed to convert the information to a signal that can be read by the microcontroller. This method contains complex circuitry and is very difficult to implement, so measuring voltage to determine the discharge rate of the capacitor is not a good method for the Sensor Pod application.

A second method to measure discharge rate is to measure capacitance and relate the capacitance value to frequency. Since the 555 timer acts as an oscillator, where part of the time the circuit is in state one and the other part of the time it is in stage two, the output of the signal is a square wave. Although many of the capacitive soil moisture sensors on the market use the linear integration method, the frequency method is desirable because the timer sends a square wave signal that can easily be read by the microcontroller and converted to frequency because as seen in Equation 21, frequency directly corresponds with time.

$$f = 1/t \tag{21}$$

As stated previously, the discharge capacitance increases linearly with time. Using this logic, the oscillations of the signal should increase and decrease depending on how quickly the soil sensor capacitor discharges.

5.1.2.6.3 Moisture Sensor Phase 3 AW

The third design iteration of the soil moisture sensor timing circuit was due to a design change of the timer itself. For testing purposes in iteration two, a pin through TLC555 timer was used. Once the design was transferred to a PCB, it was necessary to transfer the pin through component to a surface mount component to optimize board space; the surface mount component was a third the size of a pin through component. However, when the component was transferred, it was found in the datasheet that the circuit design had a slight change, as seen in Figure 26.

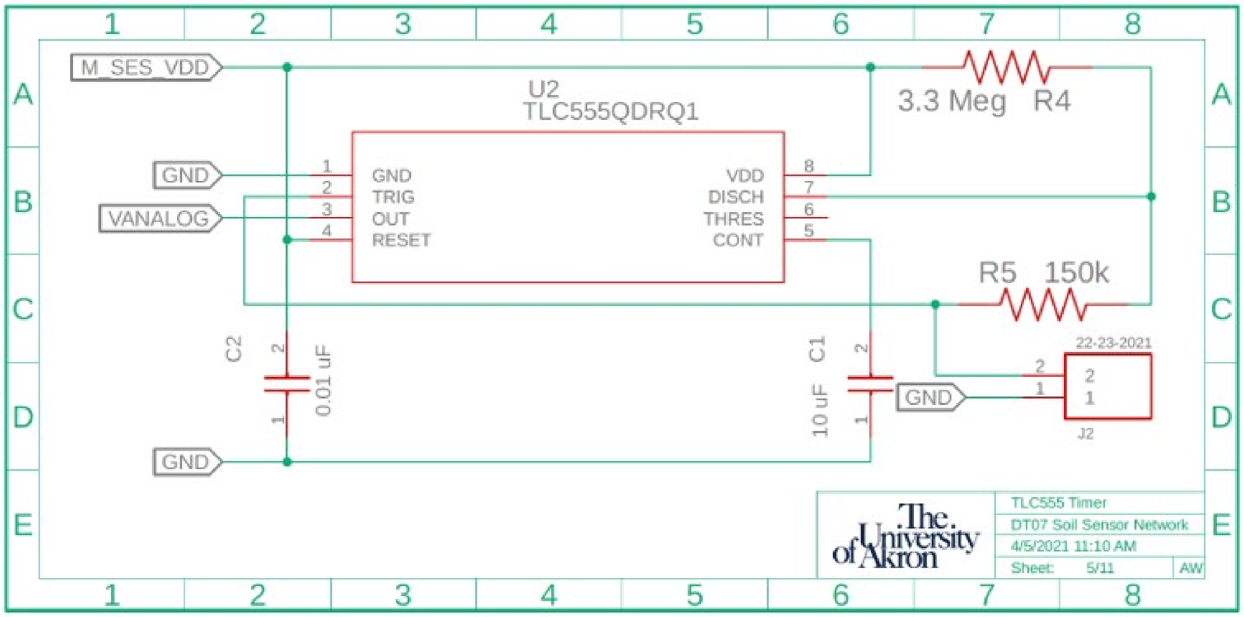


Figure 26: Soil Moisture Sensor Internal Circuit: Iteration III.

Once the design was changed to match the configuration on the datasheet, the circuit performed as expected.

5.1.2.7 *Temperature Sensor* *AW*

The MAX6607IUK+T temperature sensor was implemented into the design in addition to the moisture sensor. The typical application circuit for the sensor package consists of 2 noise reducing capacitors. The analog output from the temperature sensor will be fed into pin 41 of the microcontroller in order to be read by the microprocessor. This temperature sensor produces an output voltage that is proportional to absolute temperature and relates the two via Equation 22.

$$T(\text{°C}) = \frac{V_{out} - 500mV}{10mV/\text{°C}} \tag{22}$$

This gives an analog output that is able to be read by the microcontroller. The schematic for this type of circuit design can be seen in Figure 27.

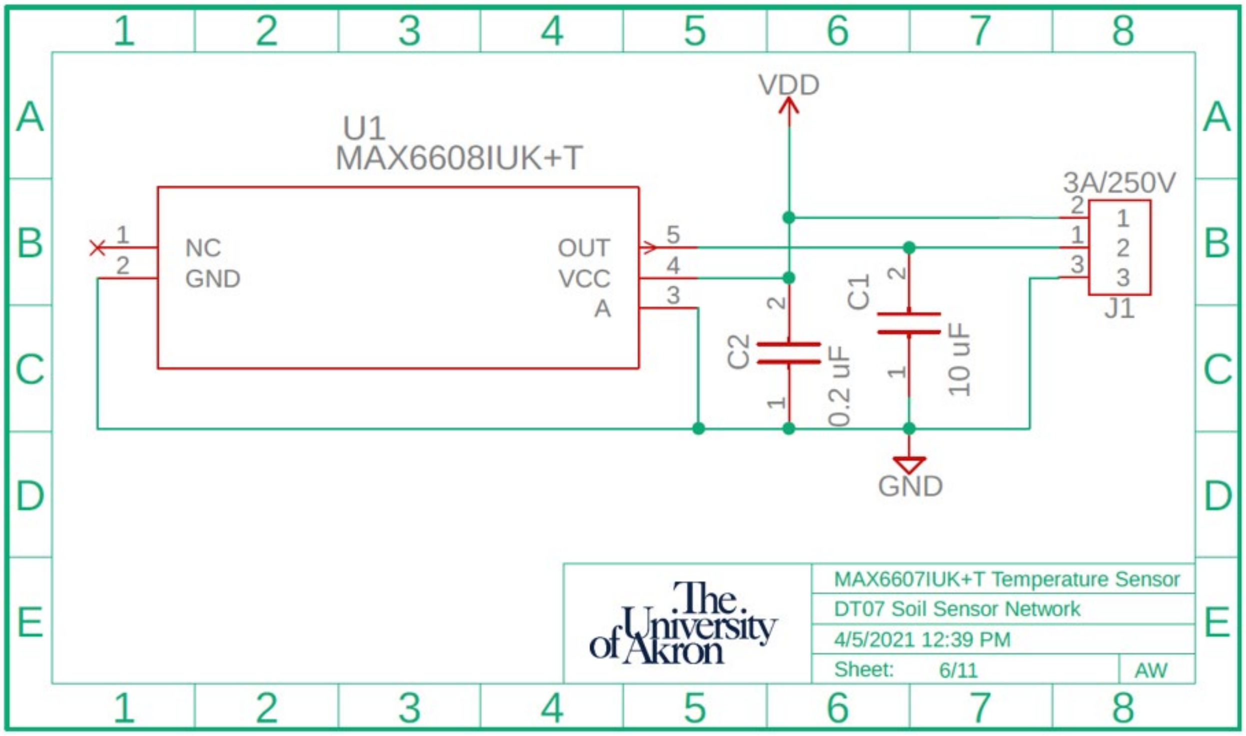


Figure 27: EagleCAD MAX6607IUK+T Temperature Sensor.

5.1.2.8 *Connectors* *AW*

Connector boards were created as an intermediate step to connect the external sensors to the main board. The external sensors, temperature and moisture, contained one side of a magnetic connector. The other side of the magnetic connector was mounted to the connector board, which would eventually be attached to the inside face of the pod.

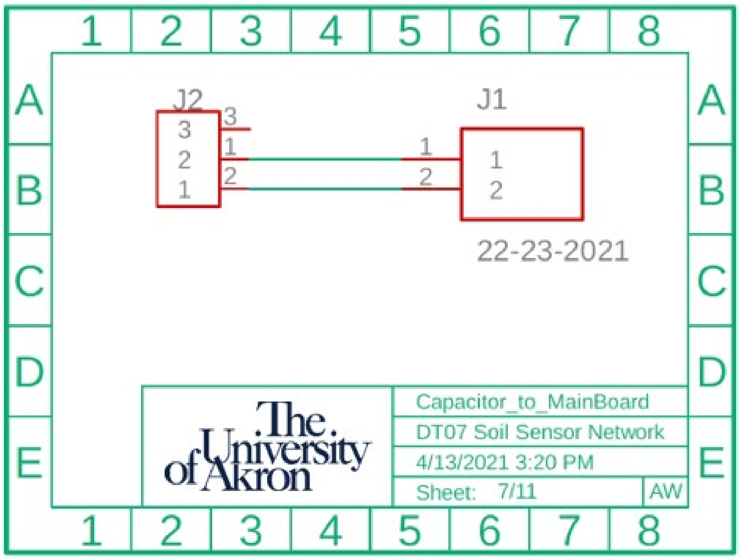


Figure 28: EagleCAD Moisture Sensor to Main Connector Board.

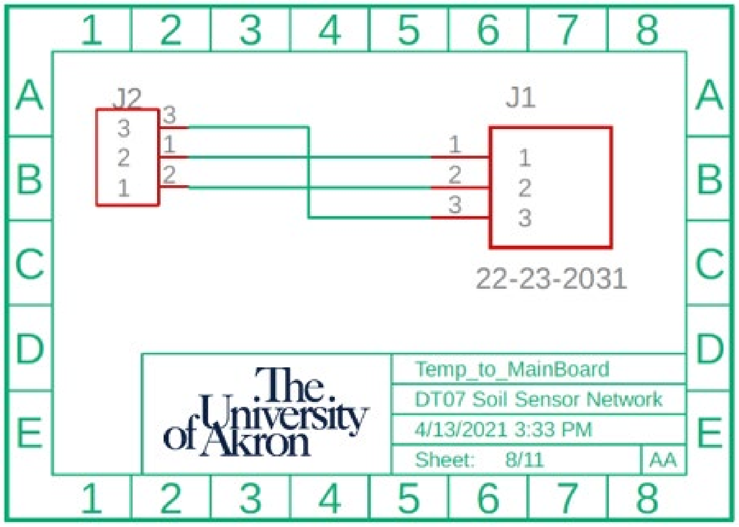


Figure 29: EagleCAD Temperature Sensor to Main Connector Boad.

On the back side of the connector board there is a WM4200-ND 2-pin connector for the soil moisture sensor and a WM4201-ND 3-pin connector for the temperature sensor. These male connectors are attached to a male connector on the main board through a wire containing a WM2011-ND (2-pin) or WM2012-ND (3-pin) female connector on either end. Connector boards were chosen to be used rather than hard-wired connections from the external pod to the main board because if the wired connection were to break, it would be difficult and time consuming to unsolder and resolder the connections. In the case of quick disconnects, if a wire or connection point is faulty, it can easily be exchanged for another connector.

5.1.2.9 *Debugging Circuitry* *AA, AW*

The debugging circuitry for the main board consists of LEDs, a reset switch, and additional pinouts for the PIC microcontroller and LoRa module.

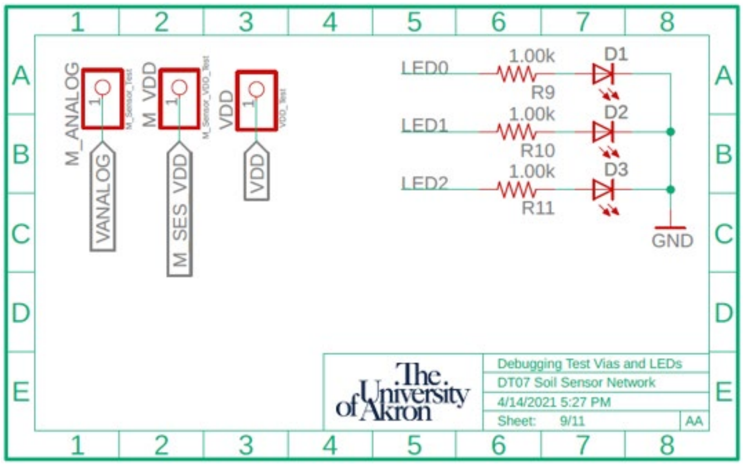


Figure 30: EagleCAD Debugging Test Vias and LEDs.

The three LEDs in the circuit are connected to the PIC in order to determine whether or not the PIC and LoRa module are properly entering and exiting sleep mode. Another debugging method used on the main PCB were test vias. Test vias are a larger size via located in the board such that a probe can fit through the hole in the case of testing for circuit continuity or voltage. The first test via was located after the analog pin on the soil moisture timing circuit and before the trace arrived at the PIC pin. The second test via was located between the supplied voltage to the timer circuit and where the voltage was inputted to the timer circuit at V_{DD}. These two test vias were chosen to give access to test if the subsystem was working properly in the case the signal was not being transferred to display a correct reading on the Senet server. It is important to monitor this subsystem as opposed to other subsystems because the majority of hardware subsystems were placed on a breakout board that connected to the main PCB whereas in this case, the subsystem was on the PCB and therefore harder to troubleshoot if no test vias existed. This subsystem was proven to work during the design phase on a PCB, which is the reason the circuit was placed directly on the main PCB.

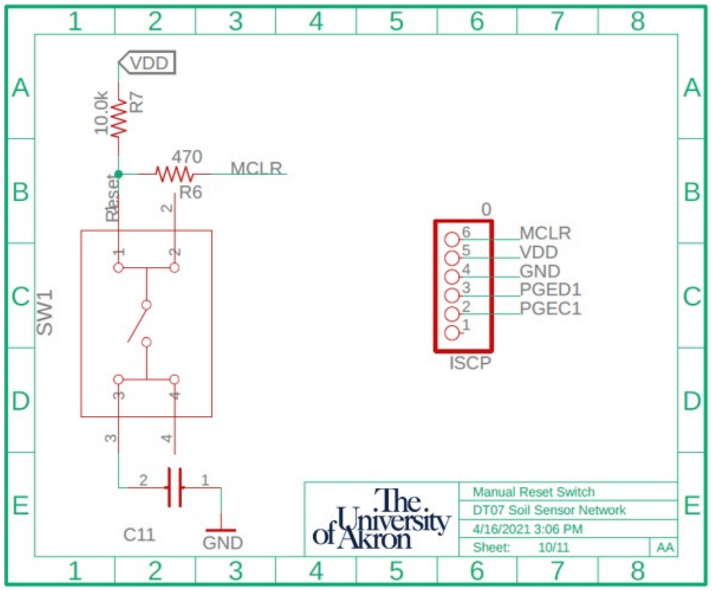


Figure 31: EagleCAD Manual Reset Pushbutton.

The Master Clear (MCLR) pin on the PIC24 serves two specific purposes: resetting the device and for device programming and debugging. Having the capability to reset the device with a single switch or button allows for faster debugging by reducing the need to manually remove the power source for reset. The MCLR also must be properly configured for the device to be programmed and debugged properly. Although this is not needed for production due to the prototype nature of this project, the MCLR pin had to be configured for testing and development.

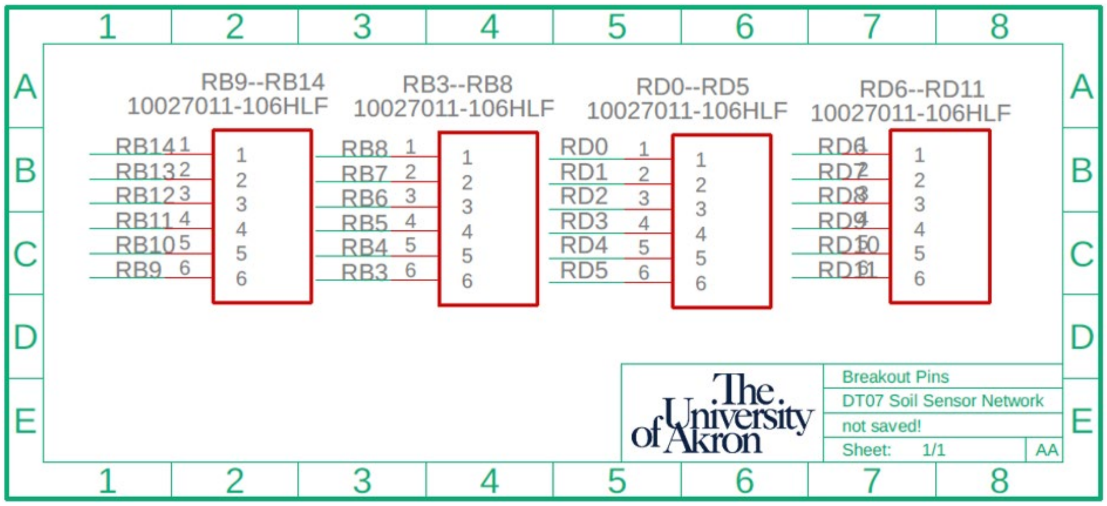


Figure 32: PIC24FJ256GB410 Microcontroller and RN2903 LoRa Module Breakout Pins.

To create a more modular PCB, breakout pins were connected to GPIO pins on both the PIC24FJ256GB410 microcontroller and RN2903 LoRa module. Pins RB0 to RB8 were used for testing and debugging the program. The remaining pins were reserved for testing and developing additional features.

5.1.3 Simulations AW

Before building a prototype, it is important to verify the designed circuit design will function as theorized before physically implementing it. The most efficient way to do this is through simulation. Both the soil moisture sensor and temperature sensor were designed in LTSpice and simulated. It was not possible for the voltage regulator to be simulated due to the limited amount of information the manufacturer released on the device.

5.1.3.1 *Soil Moisture Sensor* *AW*

The same circuit as in Figure 20 was constructed in LTSpice and a simulation was run to verify the calculations were correct and that the circuit would work properly once built. The circuit can be seen in Figure 24.

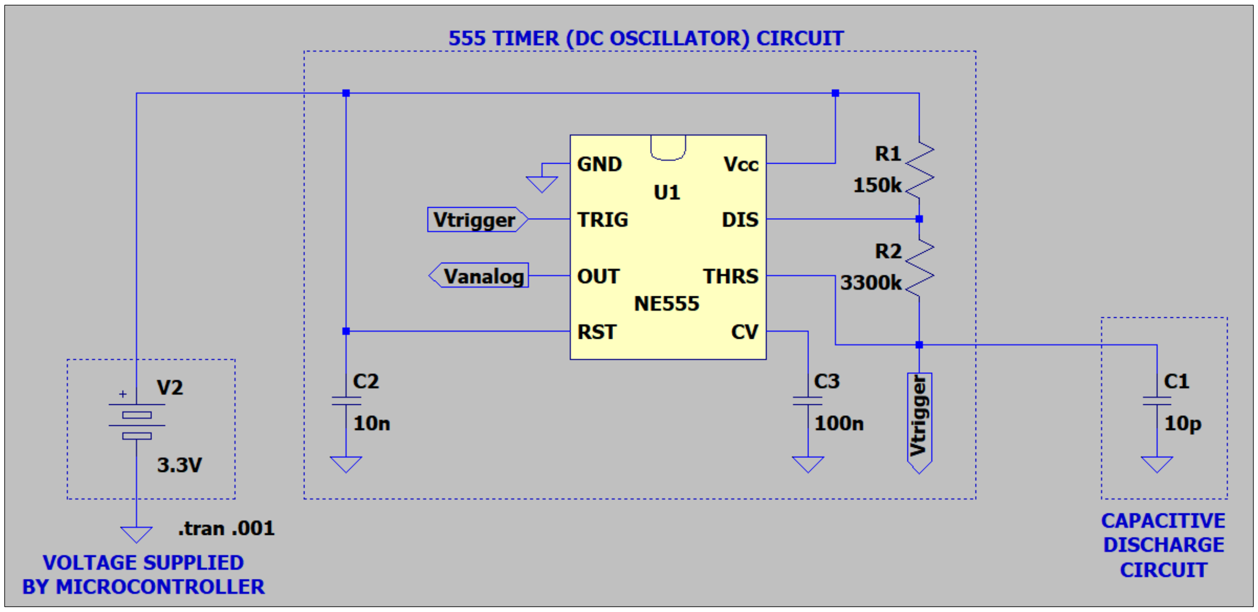


Figure 33: LTSpice Capacitive Soil Sensor Circuit.

Although a TLC555 timer is used in both the Eagle CAD schematic and implemented into the actual circuit, an NE555 timer was used for simulation because it was the only timer in the LTSpice directory. There are very few differences between the TLC555 an NE555 timer; the main difference is that the NE555 timer controls the 'on' state by altering frequency whereas the TLC555 timer controls the 'on' state by altering duty cycle. The timer for a soil sensor is used as an oscillator, as mentioned previously, and is only concerned with the change in time (ratios of time) given a capacitive discharge. For this reason, the way the timer measures time is irrelevant for the simulation.

To simulate the capacitive discharge rate of the soil moisture sensor, *C1* was set to 10 pF for the first simulation, and then set to 20 pF for the second simulation. As seen in Equation 20, the capacitance value directly corresponds to the permittivity of the soil. Soil with a greater amount of moisture has a higher permittivity. The permittivity causes the capacitance to be higher or lower. Since it is difficult to simulate the effect of the permittivity of moisture on the soil, the capacitor value was varied, which essentially 'varied' the permittivity. The results from the first simulation with *C1* equal to 10 pF is seen in Figure 25.

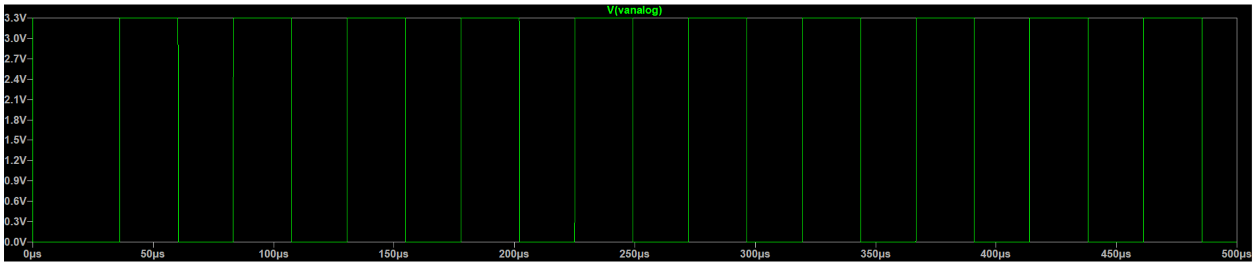


Figure 34: LTSpice: Soil Moisture Sensor Simulation C1=10pF.

The capacitor *C1* discharged in approximately 40 µs, resulting in a 25 kHz frequency.

The soil moisture sensor was then set to 20 pF and the simulation was re-run. The results from the second simulation can be seen in Figure 26.

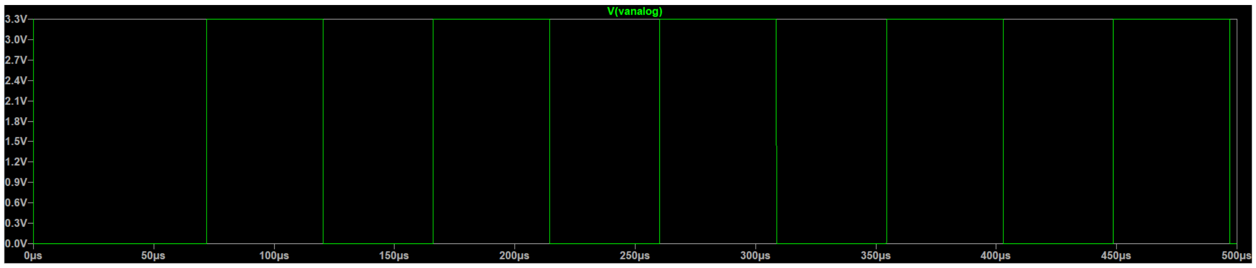


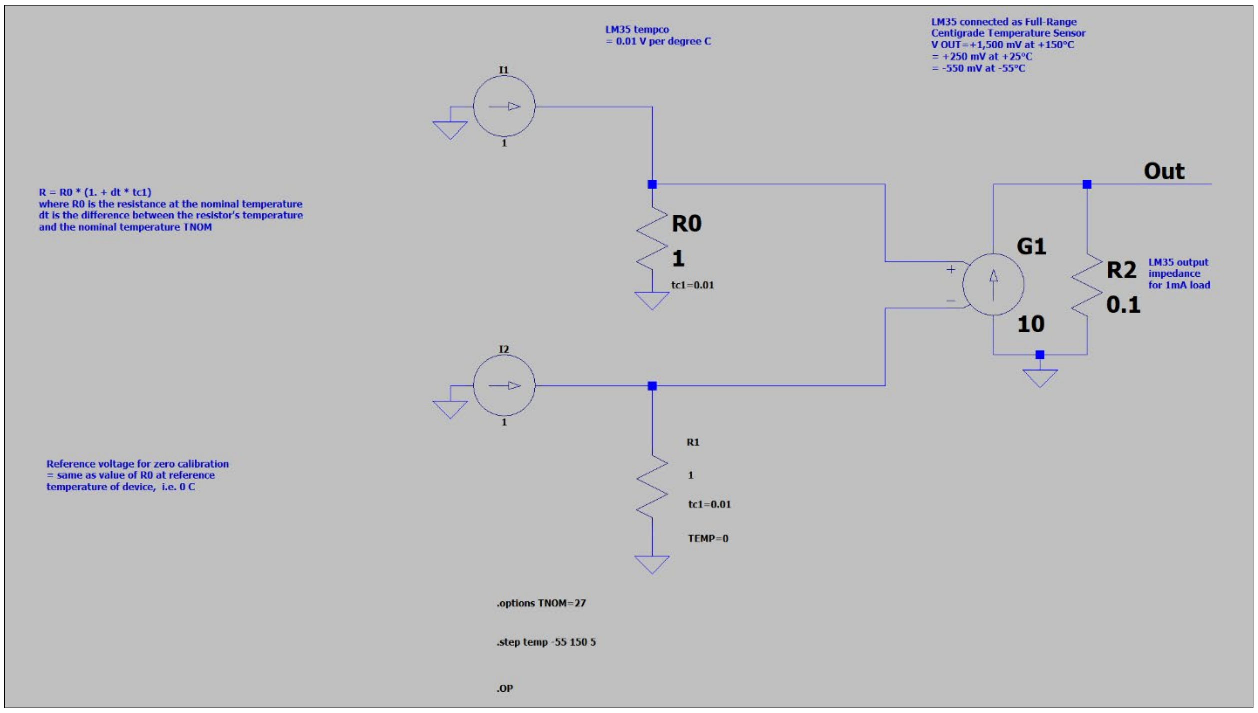
Figure 35: LTSpice: Soil Moisture Sensor Simulation C1=20pF.

As expected, the discharge rate doubled to 80 pF, resulting in a 12.5 kHz frequency. It is important to maintain a frequency in the kHz range to stay within the limit of operation for the TLC555 timer (2 MHz) while also optimizing the sampling rate.

5.1.3.2 *Temperature Sensor* *LF*

The circuit shown in Figure 27 was used to simulate a similar temperature sensor to the MAX6607. The temperature sensor being simulated is the LM35, which has a similar port set up and similar outputs. While the LM35 has 3 ports and the MAX6607 has 4 ports, the MAX6607 has a pseudo-3 port network because two of the ports are grounded together.

The LM35 also has a minor change with its transfer function. The MAX6607 has a 500mV offset at the 0°C mark in order to allow measurement of negative temperatures, whereas the LM35 does not have the offset at the expense of accuracy at higher temperatures. With this in mind, both give similar linear responses, making the LM35 a viable substitute for the MAX6607 in the simulation.



LM35 tempco
= 0.01 V per degree C

LM35 connected as Full-Range
Centigrade Temperature Sensor
V OUT=+1,500 mV at +150°C
= +250 mV at +25°C
= -550 mV at -55°C

R = R0 * (1. + dt * tc1)
where R0 is the resistance at the nominal temperature
dt is the difference between the resistor's temperature
and the nominal temperature TNOM

LM35 output
impedance
for 1mA load

Reference voltage for zero calibration
= same as value of R0 at reference
temperature of device, i.e. 0 C

```
.options TNOM=27

.step temp -55 150 5

.OP
```

Figure 36:LTSpice: Temperature Sensor Simulation Circuit.

As seen in Figure 28, as temperature increases, the output voltage increases at a linear rate.

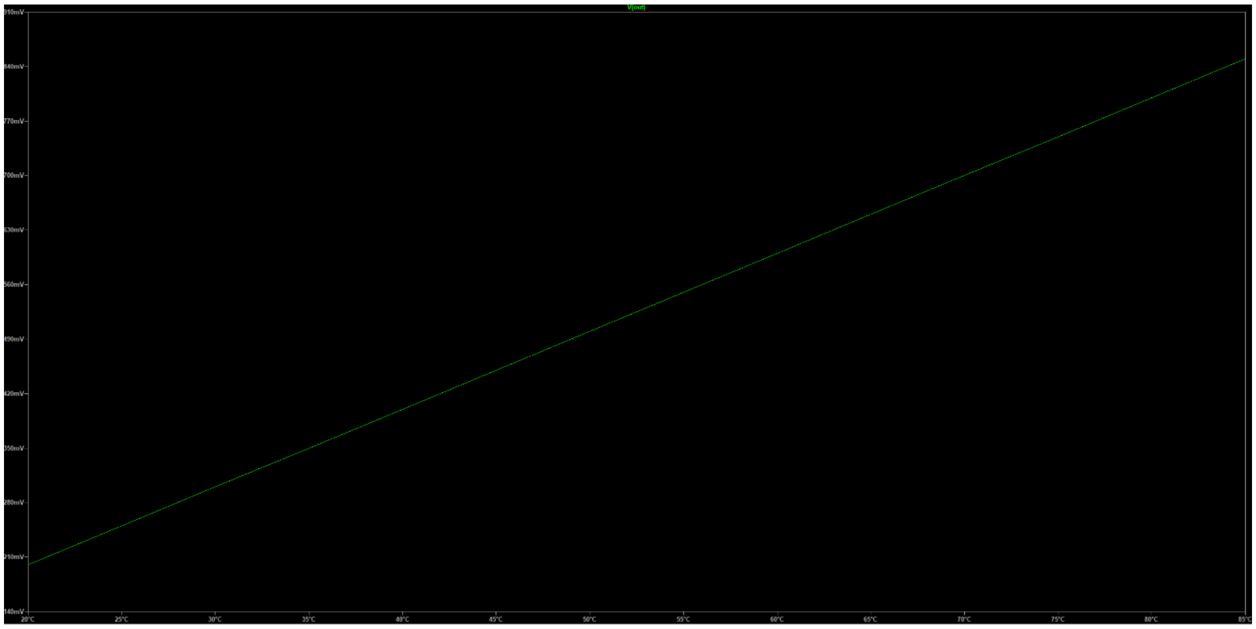


Figure 37: Temperature Sensor Simulation Diagram.

Referring back to Equation 22, the simulation graph verifies that the temperature sensor is responding as intended. Since the simulation runs properly for the LM35, it can be assumed it will also run properly for the MAX6607 temperature sensor.

5.1.4 PCB Designs AA, AW

Each of the schematics were taken and configured into a PCB. The voltage regulator and battery monitoring boards were placed onto breakout boards to aid in troubleshooting. The soil moisture sensor and temperature sensor were placed on separate boards as well since they are external to the device. All individual boards are connected to the main board.

5.1.4.1 *Main Board PCB*

The three main components on the main board are the PIC microprocessor, LoRa transceiver module, monopole antenna, and internal soil moisture sensor components. As seen in Figure 38, the microprocessor is located at the top center, the LoRa module at the bottom right, the antenna in the center of the board, and the internal soil moisture sensor components at the bottom center.

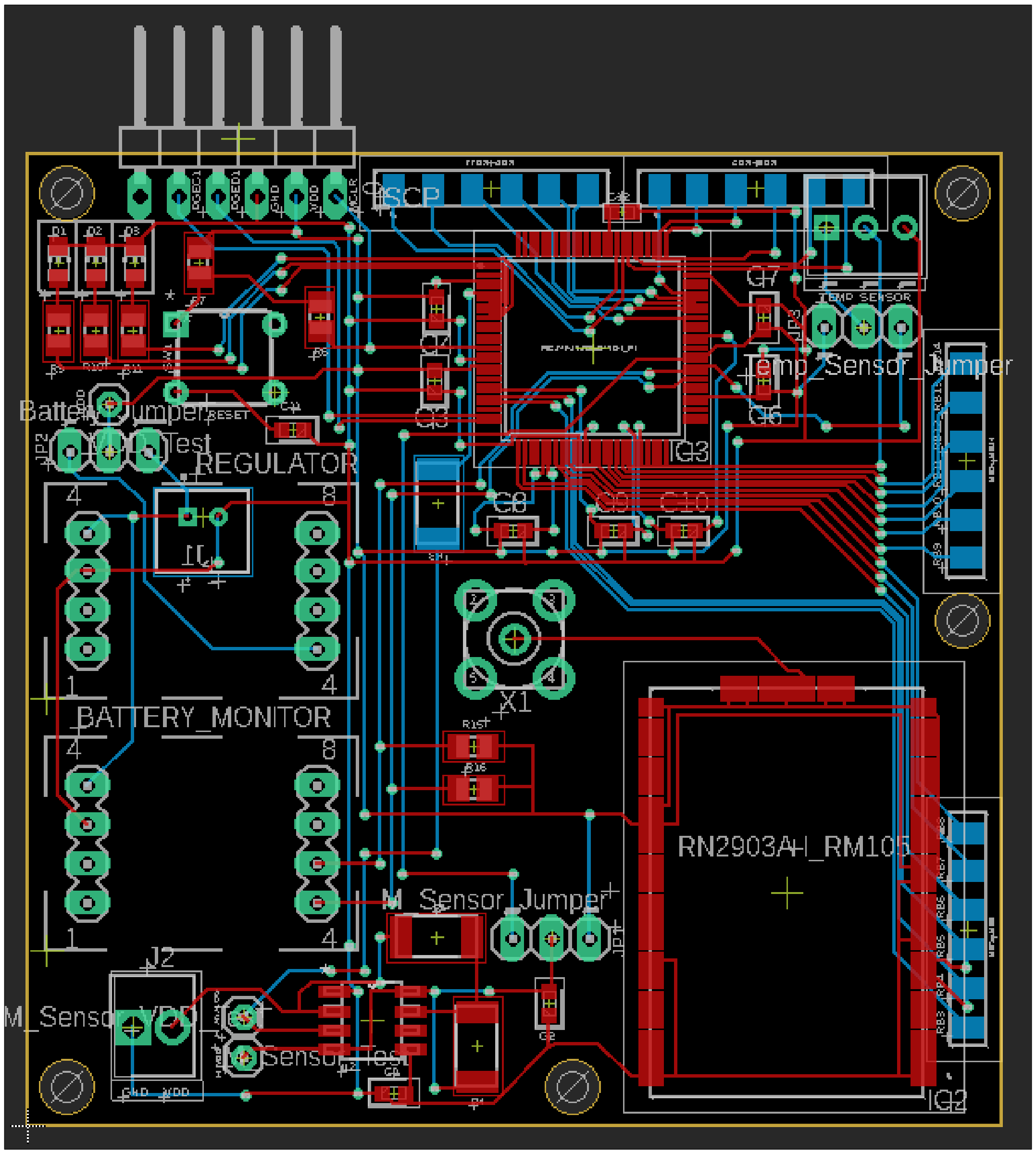


Figure 38: Main PCB Design.

The antenna was placed at the center of the board so that if it was necessary that it extruded through the top of the pod, it would be aligned with the tip of the pyramid and maintain symmetry. All other components were fitted around it. The designs of the voltage regulator and battery monitor, at the point of ordering the PCBs, had not been tested and proven to work; they were placed on separate breakout boards so that they could be switched in and out for testing until a working prototype was created. The soil moisture sensor circuit was proven to work the previous semester, so the circuit was directly placed onto the board.

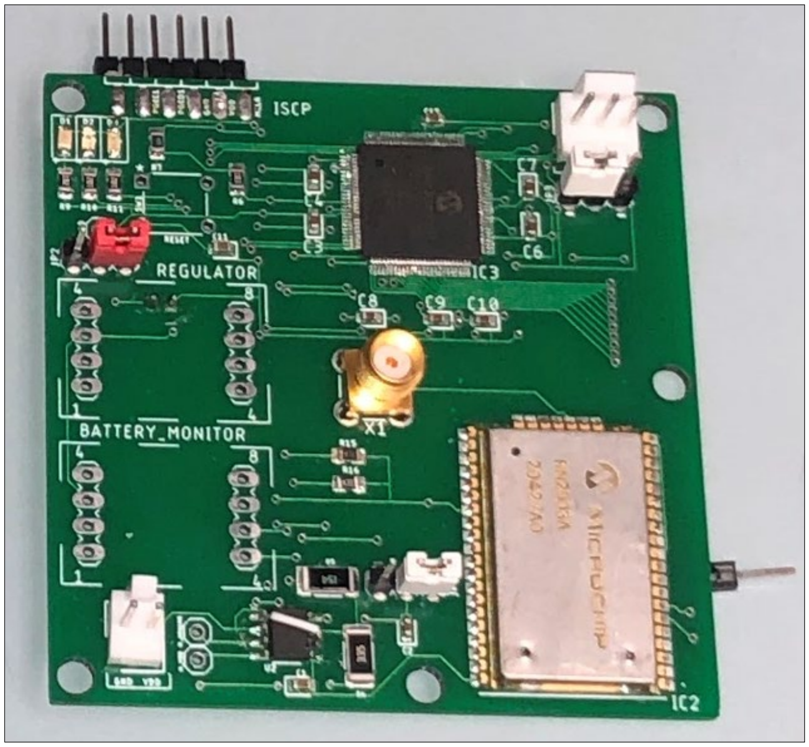


Figure 39: Main PCB.

Many design enhancements were included to help with testing and troubleshooting the circuit. At the top left of the board, a reset button with three LEDs next to it can be seen. The reset button enables the Master Clear (MCLR) pin on the PIC to be manually reset without resetting the code. The three LEDs are attached to PIC in order to verify the microcontroller is entering and exiting sleep mode properly. In addition to this, spare PIC pins are connected to breakouts across the top and left side of the board in case additional components need tested that were not part of the original design. The breakout pins will be removed once the prototype is complete. Test point vias are next to each individual circuit (microprocessor, transceiver, moisture sensor) that allow the power, ground, and signal traces going to each component to be probed for troubleshooting purposes.

5.1.4.2 *Power Management System*

The power management system is broken down into three main sections. The first system is the buck-boost voltage regulator. As stated previously, a buck-boost regulator was used in order to maintain 3.3V despite whether the battery is charged above or below the desired voltage. The PCB layout is shown in Figures 40 and 41 .

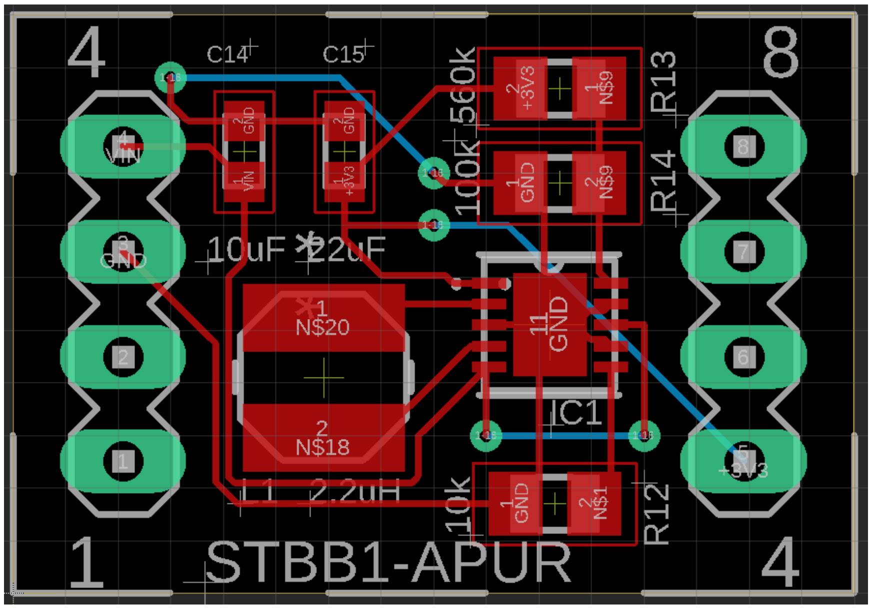


Figure 40: Buck-Boost Voltage Regulator PCB Design.

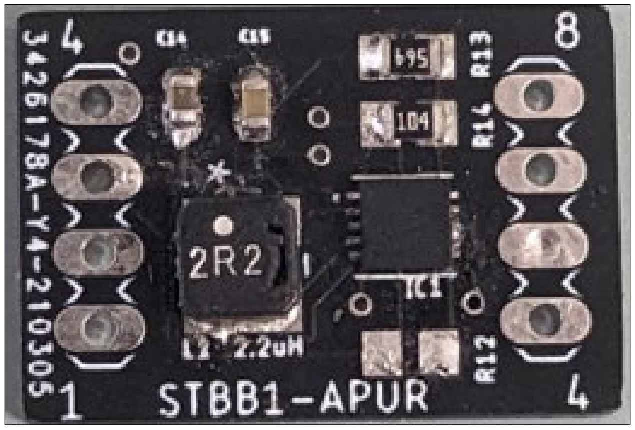


Figure 41: STBB1-APUR Voltage Regulator PCB.

The headers on either side are not only utilized for power and signal connections, but also for stabilizing the board onto the main board.

The second section of the system is the battery monitoring PCB.

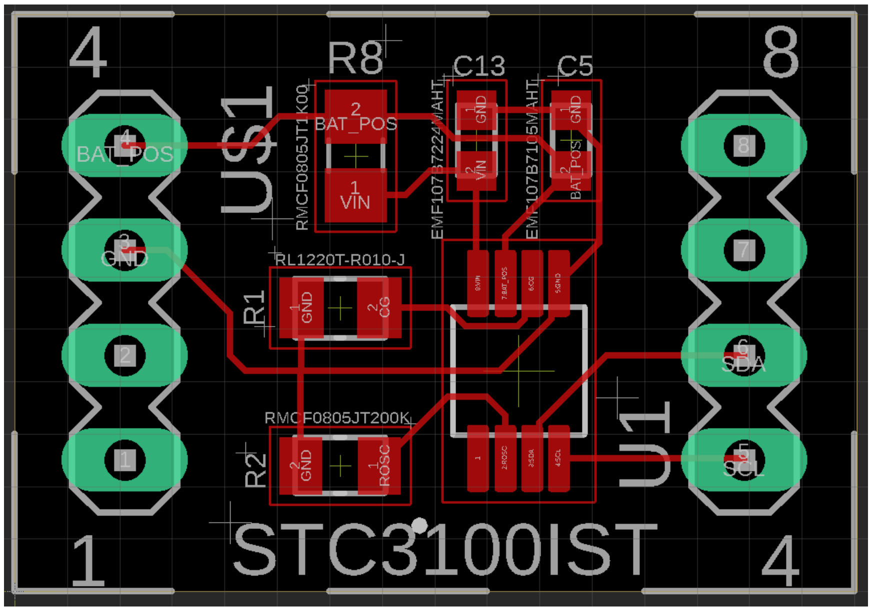


Figure 42: Battery Monitoring PCB Design.

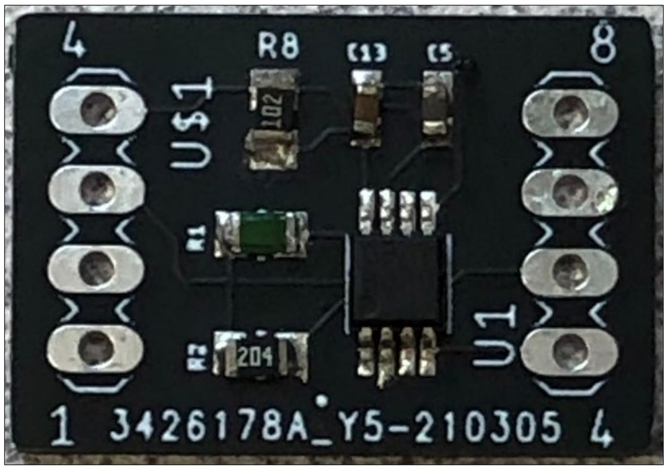


Figure 43: Battery Monitoring PCB.

The battery monitor is connected to the battery before the voltage regulator and collects data such as voltage, current, and battery depletion. The microcontroller is capable of monitoring system voltage, but this is after the voltage has gone through the regulator. Since a buck-boost regulator is used, this means the regulator will stabilize the supplied voltage to 3.3V until the battery is almost fully depleted. A monitoring circuit is useful because it will monitor the actual battery voltage, not the regulated voltage. The monitor can then send a signal to the microcontroller which can then transmit a signal to alert the farmer if the battery is low and the Sensor Pod is in need of retrieval.

The third system involved in power management is the battery pack PCB.

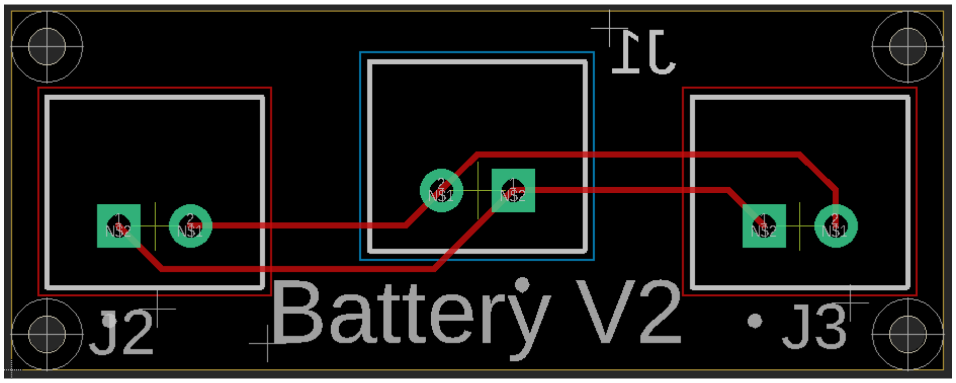


Figure 44: Battery Pack PCB Design.

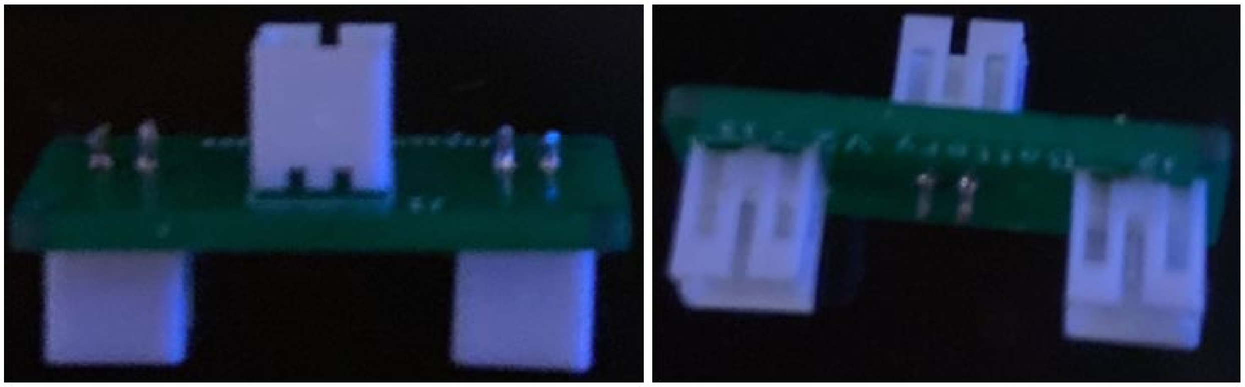


Figure 45: Battery Pack PCB.

As will be discussed in later sections, all designs created must keep the end user in mind. When a farmer needs to recharge the Sensor Pod batteries, it is most convenient if they can both be recharged at once. The battery pack PCB connects two batteries in parallel on the under-side of the board to a circuit that has a third connector on the top-side of the board. The top connector is connected to a wire going to the main PCB; when the batteries need recharged, the top connector can be disconnected from the main PCB and connected to the battery charger. Charging the pack from the top connector will allow both batteries to charge at the same time.

5.1.4.3 *External Sensors*

The external sensors on the pod are a soil moisture and temperature sensor. The soil moisture sensor, as depicted in Figure 46, is a multi-level parallel plate capacitor with power going to one lead and ground going to the other. The middle pogo pin on the magnetic connector is unused. The other half of the circuitry containing the timer is on the Main PCB, internal to the Sensor Pod.

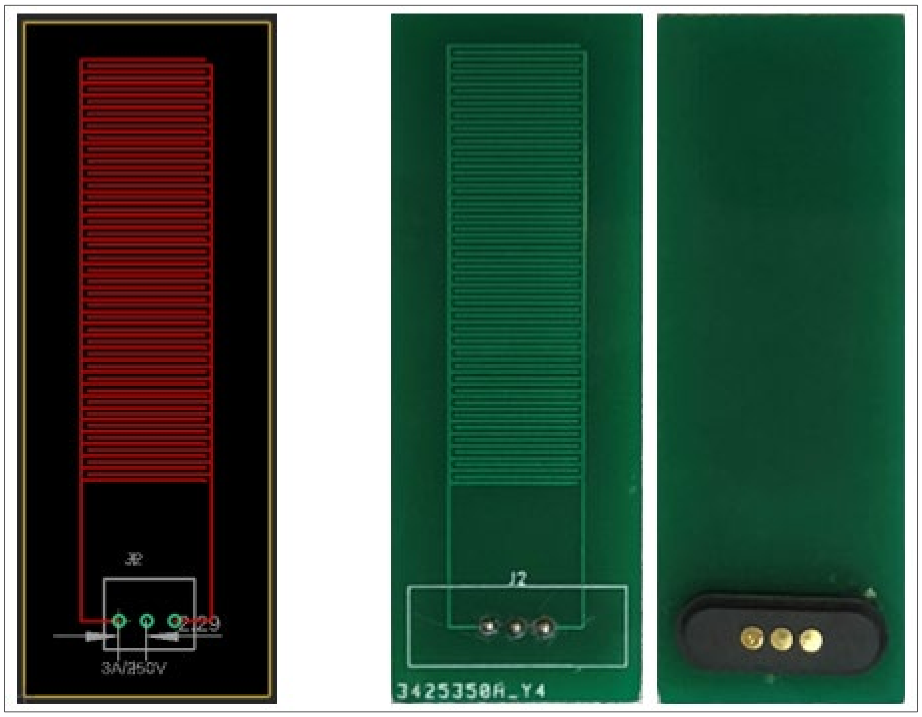


Figure 46: Soil Moisture Sensor PCB.

The temperature sensor circuit is completely external. One capacitor is connected to V_{CC} and the other is connected to the output. Both capacitors are used for noise reduction.

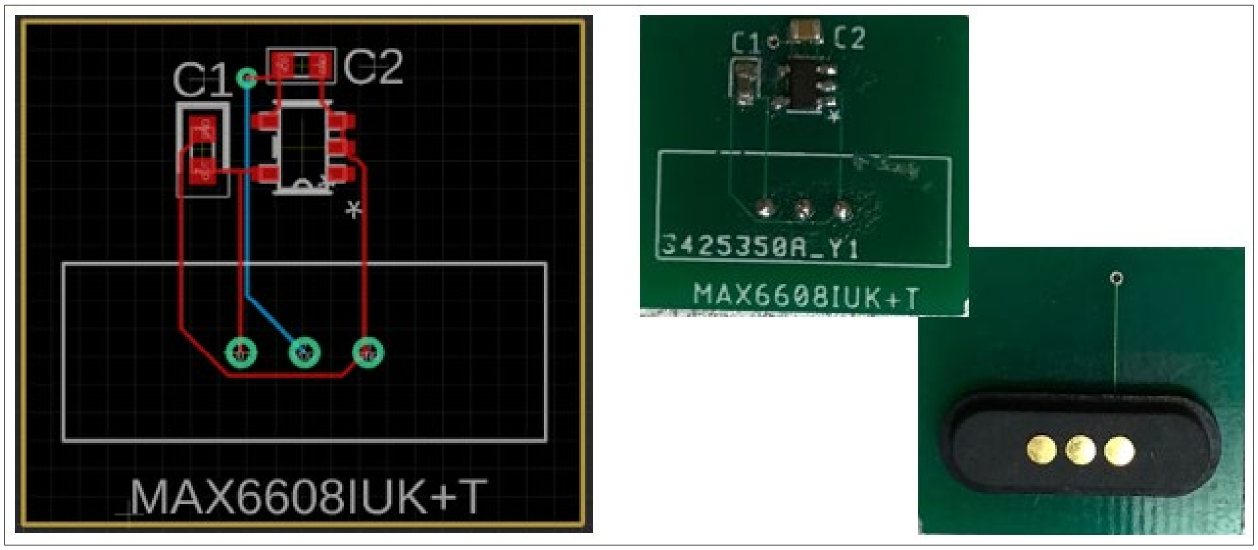


Figure 47: Temperature Sensor PCB.

For future design implementation, it should be noted that each eternal sensor should have a clip quick disconnect clip on the top of it so that the sensor is more firmly secured to the pod.

5.1.4.4 *Connectors*

Both external sensors have an intermediary connector PCB that is mounted to the inside of the Sensor Pod walls. An example of the intermediary connector for the temperature sensor is shown in Figures 48 and 49. The only difference between the temperature sensor intermediary connector and moisture sensor intermediary connector is that the temperature sensor PCB has a 3-pin male connector whereas the moisture sensor PCB has only a 2-pin male connector.

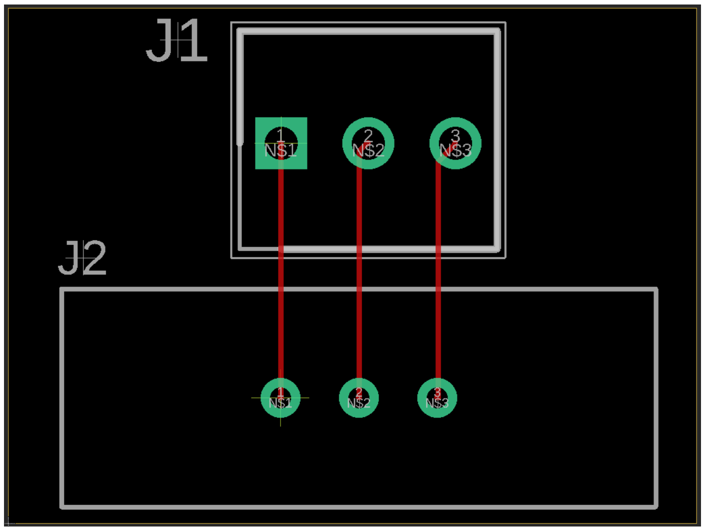


Figure 48: External Sensor to Main Board PCB Design.

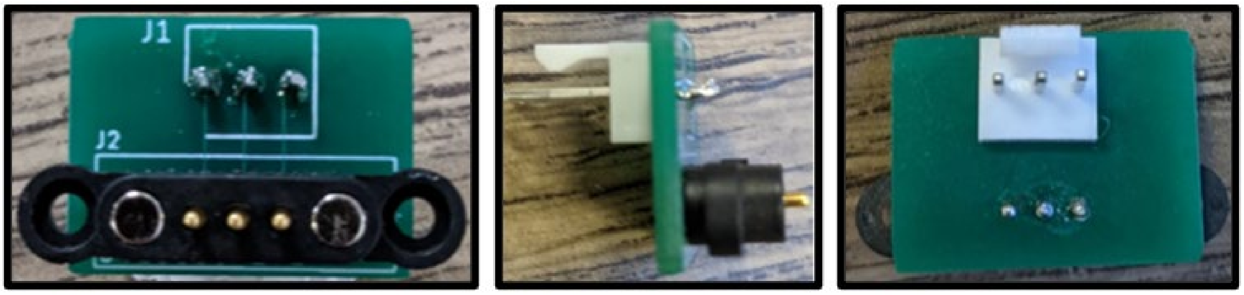


Figure 49: External Sensor to Main Board PCBs.

The black magnetic connector is secured to the inside wall of the pod and connects to the magnetic connector of the external sensor. The white connector attaches to a wire that is routed to the main PCB.

5.2 SOFTWARE DESIGN

The two main software components of the Wireless Soil Sensor Network are the web application and the embedded software. The web application will act as an interface for the user to visualize and maintain their farm data. The embedded software will be the main control program for Sensor Pod operation. The design of each component has a different focus, such as a power-efficient embedded software and a user- and mobile-friendly web application. These ideas must be kept in mind to meet Engineering Requirements.

5.2.1 Embedded Firmware AA, RK

The flowchart in Figure 50 depicts the general series of events that the embedded components of the Sensor Pod will adhere to. Note that the creation of the flowchart begins at Level 1 because Level 0 is a hardware representation of the embedded system, which is encompassed into the Level 0 Block Diagram.

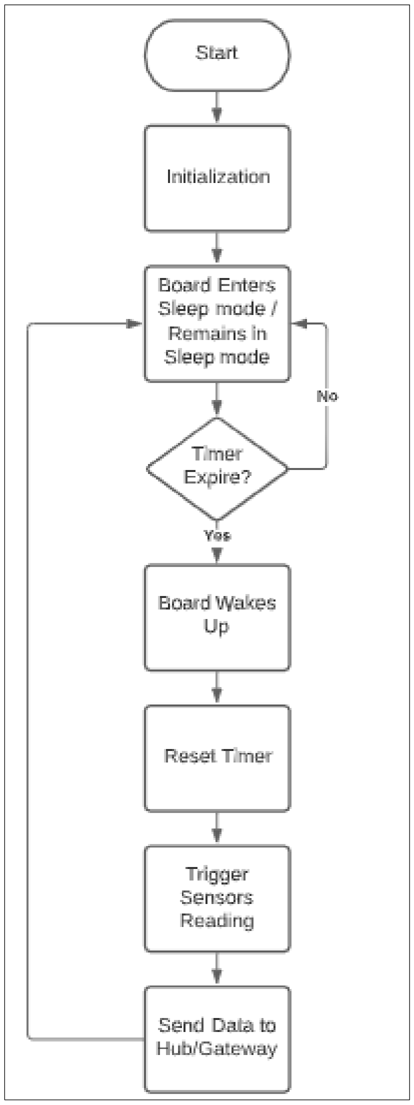


Figure 50: Level 1 Embedded Flowchart.

The flow of control for the embedded system follows a strict set of events. First, the system initializes, which will configure the system for data acquisition and transmission. To save power, it will enter a sleep mode and continuously check for a timer expiration. If the timer has expired, the system will reset the timer, “wake up”, and trigger the sensors to take readings. The sensor readings can then be transmitted to the gateway for data storage. The system will then “go to sleep” and wait until the timer expires and the process will begin again.

5.2.1.1 Trigger Sensor Reading AA, RK

Triggering a sensor reading will follow four steps to acquire soil data. When a timer has expired, the sensors equipped on the Sensor Pod can be activated and take a measurement. The microcontroller will then process the acquired data, and package it to be sent over LoRaWAN protocol to the Gateway.

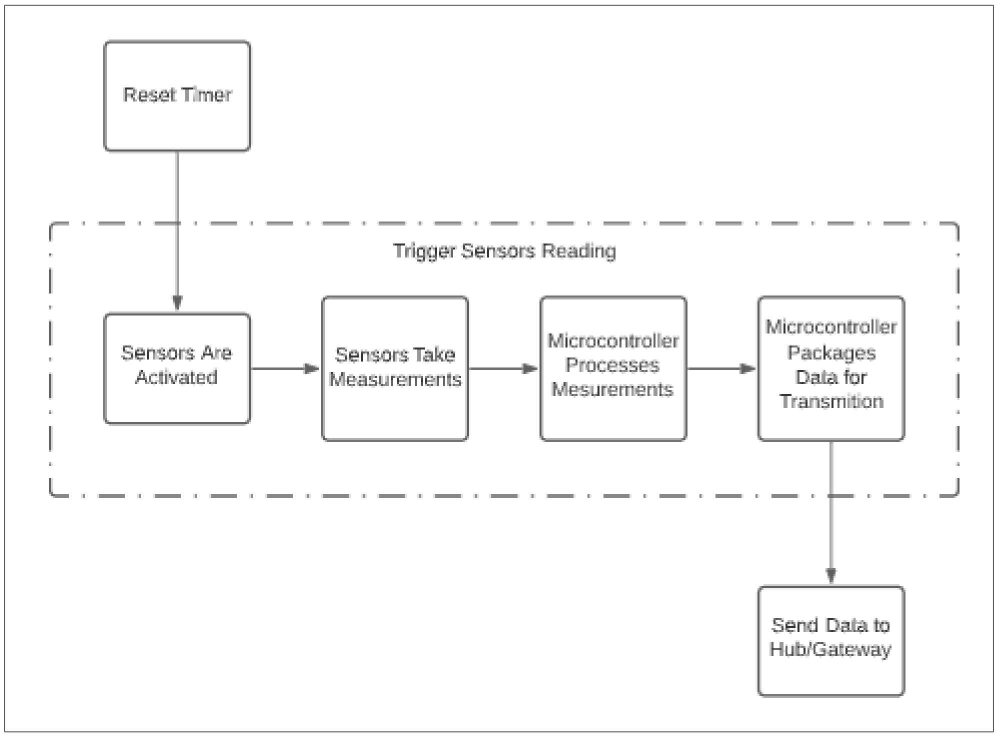

Reset Timer
Trigger Sensors Reading
Sensors Are Activated
Sensors Take Measurements
Microcontroller Processes Mesurements
Microcontroller Packages Data for Transmition
Send Data to Hub/Gateway

Figure 51: Level 2 Embedded Flowchart: Trigger Sensor Readings.

5.2.1.2 Gateway/Hub Data Communication AA, RK

To efficiently manage power and transmit data to the gateway, a strict set of steps should be followed. When a reading is triggered, the reading will be processed and put in a form that the main microcontroller can send to the LoRa transceiver. Once the transceiver is "woken-up", it will receive the payload, and transmit data to the Gateway. When data transmission is complete, the LoRa module can go back into its sleep mode and wait for the process to be completed next cycle.

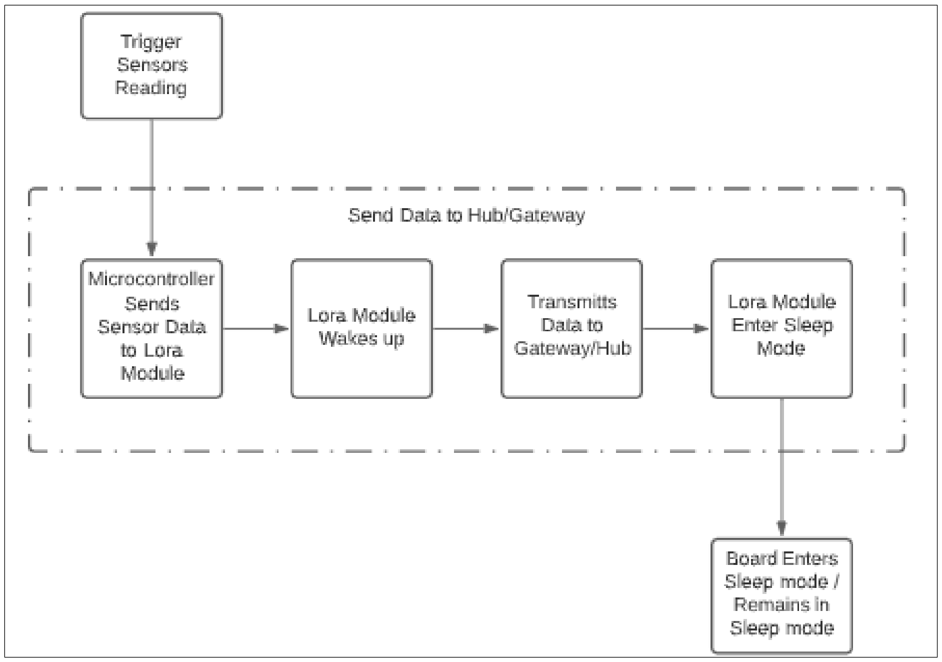

Trigger Sensors Reading
Send Data to Hub/Gateway
Microcontroller Sends Sensor Data to Lora Module
Lora Module Wakes up
Transmitts Data to Gateway/Hub
Lora Module Enter Sleep Mode
Board Enters Sleep mode / Remains in Sleep mode

Figure 52: Level 2 Embedded Flowchart: Send Data to Gateway/Hub.

5.2.2 Software Block Diagrams **AA**

The following software block diagrams depict the overall architecture at each design level.

5.2.2.1 *Level 1 Block Diagram* ***AA, RK, AW***

Further expanding the Level 3 Block Diagram seen in Figure 52, the data storage, server, and gateway were broken down into additional components.

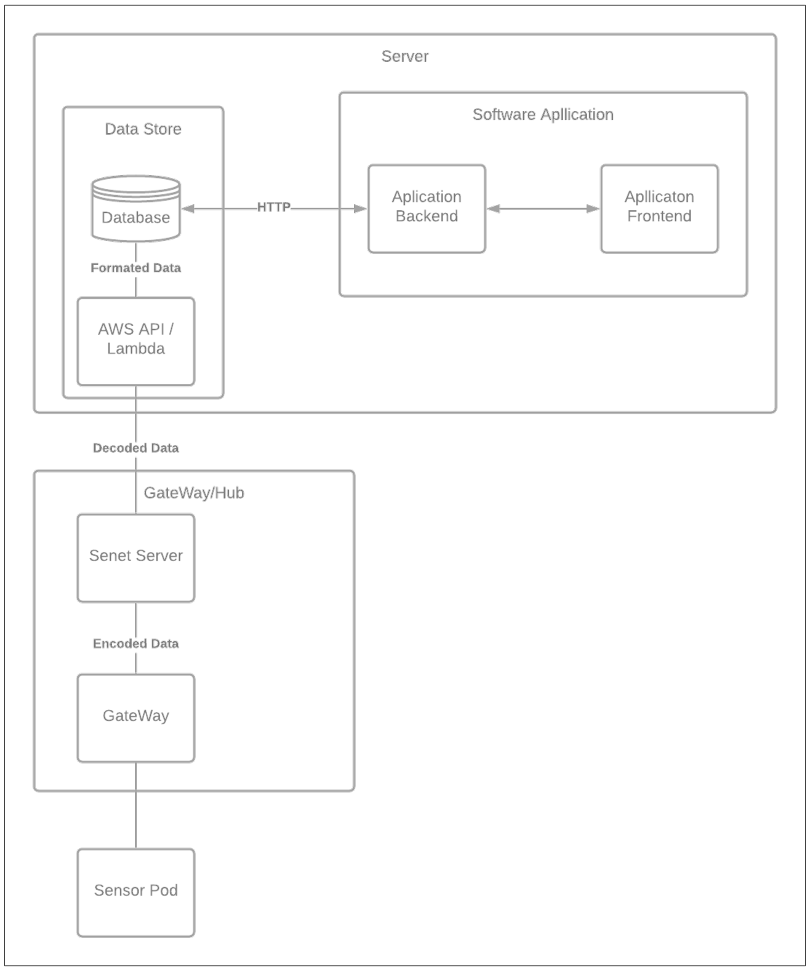


Figure 53: Level 1 Software Block Diagram.

Sensor Pod data is sent first to the Gateway, and then encoded to be sent to the Senet server.

Table 28: Level 1 Software Block Diagram: FR Table: Gateway.

Module	Gateway
Designer	Ross Klonowski
Inputs	LoRa radio messages
Output	Encoded data
Description	Gateway collects data over LoRa frequencies and forwards data to the Senet server

Table 29: Level 1 Software Block Diagram: FR Table: Senet Server.

Module	Senet Server
Designer	Ross Klonowski
Inputs	Encoded data from the Gateway
Output	Decoded data sent to AWS API/Lambda via HTTP
Description	A hub for sensor data to be stored and forwarded for use in the application

Data Store is broken down into the database and AWS/API/Lambda blocks.

Table 30: Level 1 Software Block Diagram: FR Table: AWS API/Lambda.

Module	AWS API/Lambda
Designer	Aléxis Alves
Inputs	Raw Senet data
Output	Formatted data sent to database via HTTP
Description	Interface between database and Senet server; models data to match format of database

Table 31: Level 1 Software Block Diagram: FR Table: Database.

Module	Database
Designer	Aléxis Alves
Inputs	Formatted data from AWS API
Output	Formatted data sent to application backend via HTTP
Description	

The Server is comprised of a Frontend and Backend of the application, as seen in Tables 32-33.

Table 32: Level 1 Block Diagram: FR Table: Backend of the Application.

Module	Backend of the Application
Designer	Aléxis Alves
Inputs	Formatted data sent from database via HTTP
Output	API call sent to the frontend of the application
Description	Handles all operations regarding database interface modeling and logic calls.

Table 33: Level 1 Block Diagram: FR Table: Frontend of the Application:

Module	Frontend of the Application
Designer	Aléxis Alves
Inputs	API call sent from the backend of the application
Output	Data displayed to user
Description	User interface for Wireless Sensor System interaction; handles displaying data

5.2.2.2 *Level 2 Software Block Diagram*

The software application frontend and backend of the software block diagram was further broken, as seen in Figure 54.

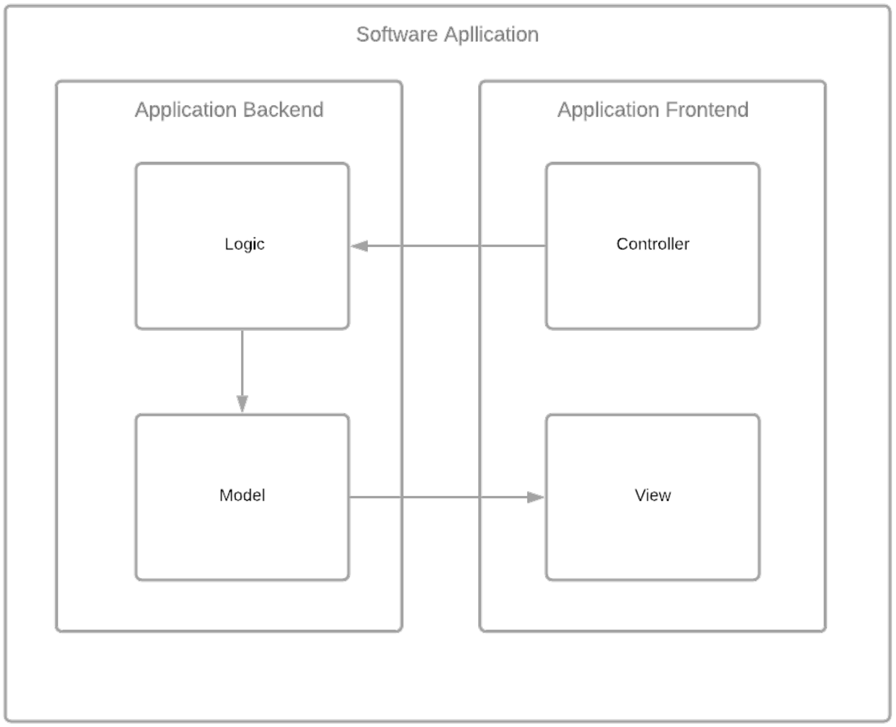


Figure 54: Level 2 Software Block Diagram.

The backend of the application was further broken down into Logic and Model blocks, as seen in Tables 34-35.

Table 34: Level 1 Software Block Diagram: FR Table: Logic.

Module	Logic
Designer	Aléxis Alves
Inputs	API Calls
Output	Results
Description	This handles all logic operation in the backend such as acquiring data, calculation, and validation.

Table 35: Level 1 Software Block Diagram: FR Table: Model.

Module	Model
Designer	Aléxis Alves
Inputs	Results
Output	Modeled Results
Description	The modeling operation is used to format any results from requested from the logic operation.

The frontend of the application was further broken down into Controller and View, as seen in Tables 36-37.

Table 36: Level 2 Software Block Diagram: FR Table: Controller.

Module	Controller
Designer	Aléxis Alves
Inputs	User interaction
Output	API call
Description	The controller operation reacts to user interaction and makes the respective API to the backend.

Table 37: Level 2 Software Block Diagram: FR Table: View.

Module	View
Designer	Aléxis Alves
Inputs	Modeled Result
Output	Application display
Description	This View operation is used to render the display page with modeled results from the backend.

5.2.3 LoRa Communication Setup RK

There is a slew of commands that must be sent to the transceiver to enable the transceiver for communication to the Gateway. Commands follow a general format consisting of about four fields. The first field can be "mac" for commands regarding the mac protocol or "radio" for commands regarding the physical layer of communication. The next field is either a "set" or "get" for if a setting should be read from or written to. The last two commands will be the field desired, and then the parameter, if the command is to change a setting. For example, to set the device's address, the following command can be sent: *"mac set devaddr 123456789"* In this case, the mac layer's device address will be set to the included parameter. This general format should be used for the other commands that are needed. To configure the RN2903 to work with the application in Senet, the Network Access Key, Application EUI, and Application Key must also be set.

Another important command that must be sent is to turn on Cyclic Redundancy Check (CRC) which is used for error correction when the packet is sent to the destination. To further improve communication reliably, the Adaptive Data Rate (ADR) will be turned on which adds information to the messages about the device, which then allows the destination to optimize the data rate of transmission. The last and most important step is to join the network. Using the preconfigured keys and parameters, the *"mac join abp"* will make the mote attempt to join the network. Activation By Personalization (ABP) is one of two methods that allow a device to be added to a network. This means that information is exchanged between the node and network server before addition to the network.

5.2.4 Data Flow AA

The flowchart in Figure 55 shows the flow of data from the Gateway to the web application.

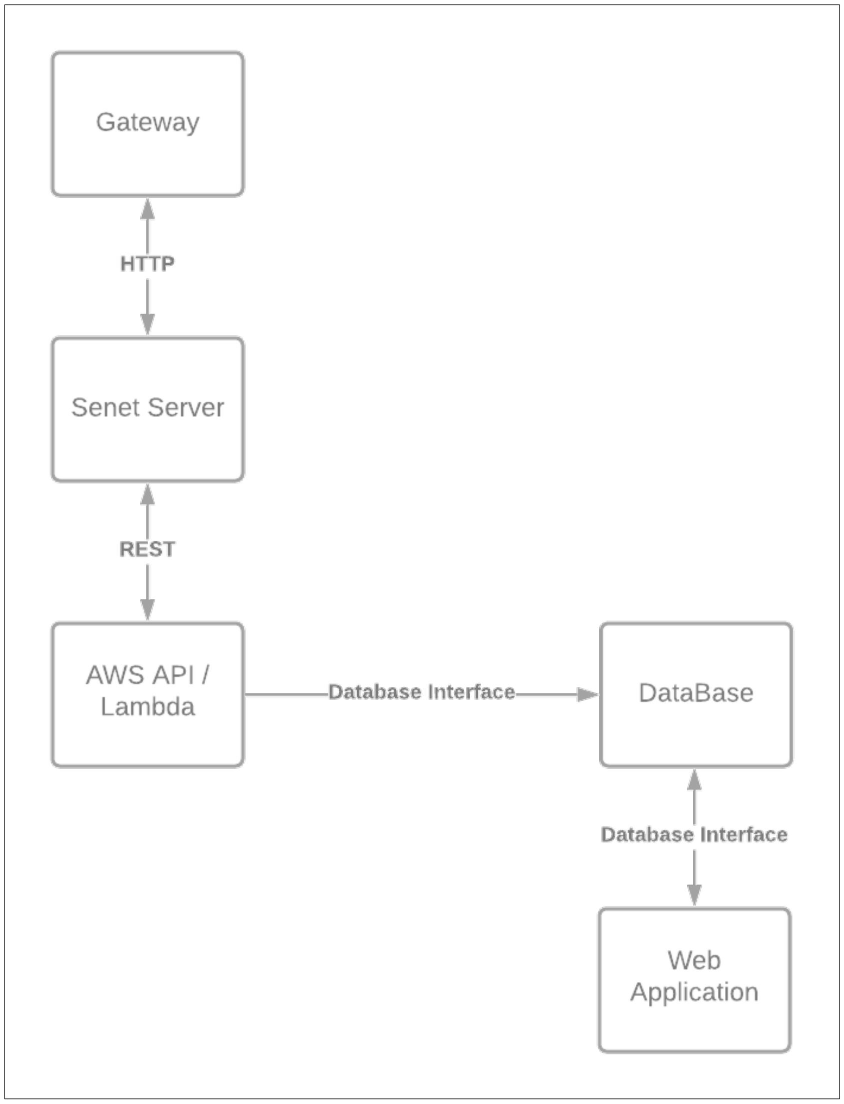


Figure 55: Data Flow Flowchart.

Tables 38-41 give a brief overview that explains how the data is moved from each component from the gateway to the web application.

Table 38: Data Flow: FR Table: Gateway.

Module	Gateway
Designer	Ross Klonowski
Input	LoRaWAN Communication
Output	HTTP Call
Description	The gateway receives information form the sensor pods via LoRaWAN communication. It then forwards the information via HTTP calls to the Senet server.

Table 39: Data Flow: FR Table: Senet Server.

Module	Senet Server
Designer	Ross Klonowski
Input	HTTP Call
Output	REST API Call
Description	The Senet Server receives information from the gateway by mean of HTTP calls. Once the information is received the server decodes the information and stores it locally. The server than utilizes a link forwarded to offload information to the database by REST API calls to an API endpoint.

Table 40: Data Flow: FR Table: Amazon Web Server (AWS).

Module	Amazon Web Services (AWS) API / Lambda
Designer	Aléxis Alves
Input	REST API Call
Output	Database Interface
Description	The AWS API is used as an endpoint for REST API calls. Once the REST call is received it is then routed by a Lambda function. The lambda function which is written in node.js parses the call for information and converts the payload from HEX to String before and sends it to the database using the database interface.

Table 41: Data Flow: FR Table: Web Application.

Module	Web Application
Designer	Aléxis Alves
Input	Database Interface
Output	Database Interface
Description	Once the data is in the database the web application can interface with it using the database interface. The web application backend can then use the database interface to change, get, delete, and create data.

5.2.5 Gateway / Senet Server RK

The Gateway is a key component that collects and forwards all the data that is transmitted from the Sensor Pods. For the star topology network that has been created for this Wireless Sensor System, the Gateway chosen should reliably capture data from all Sensor Pods, even if they are communicating at the same time. For this application, a Gateway from Laird has been chosen which is shown in Figure 56.

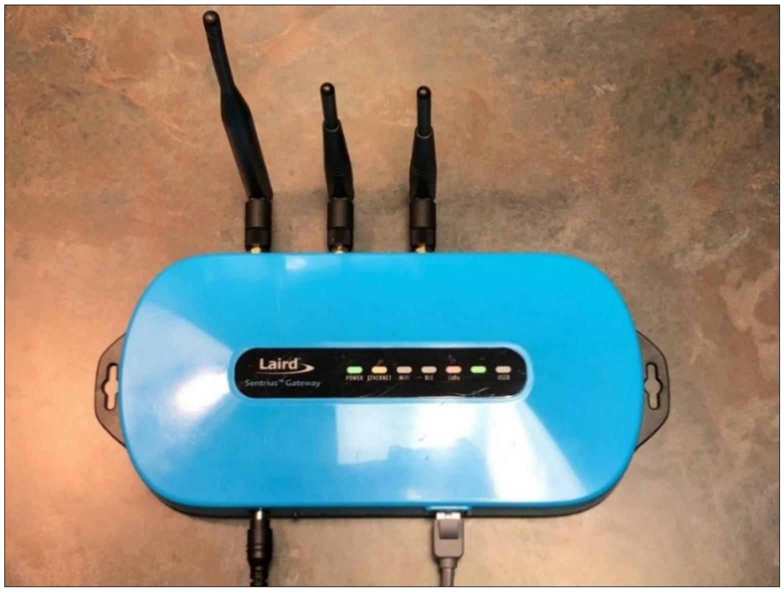


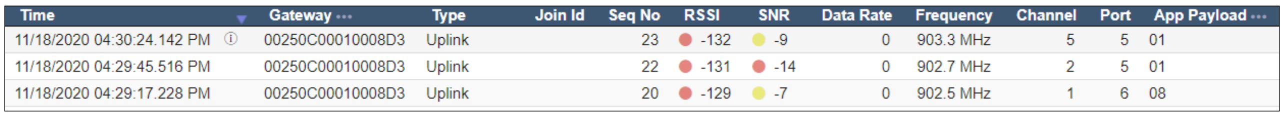
Figure 56: RG191 Senet Laird Gateway.

The Gateway can listen to the United States LoRa frequencies, which are designated as the 902 – 928 MHz band. The Gateway also conveniently features wired LAN and Wireless connectivity for data to be forwarded to the internet/ IoT Hub.

When transmitted data is received by the Gateway, it is then forwarded to a network server. For this application, the Senet network server was chosen to be used. Senet is an IoT Hub where the devices, such as Sensor Pods and Gateway, can be registered. The Gateway is registered with an external omnidirectional antenna, proper location, and network connection, which is wired-ethernet. Next, the Sensor Pods that will communicate with the Gateway should be registered. In the registration process, a device ID should be inputted into Senet that can be obtained from the device. Senet also generates a couple different fields including application EUI, Network Access Key, and Application Session Key which should be inputted on the Sensor Pod. Once all devices are registered properly, Senet can begin receiving messages from the Gateway and the Pods in which it is connected.

Senet stores the data and provides access to the metadata associated with it, such as the timestamp, any acknowledgements, packet length, sequence number, and much more. This information can be seen in Table 42.

Table 42: Senet Uplink Data.



Time	Gateway	Type	Join Id	Seq No	RSSI	SNR	Data Rate	Frequency	Channel	Port	App Payload
11/18/2020 04:30:24.142 PM	00250C00010008D3	Uplink		23	🔴 -132	🟡 -9	0	903.3 MHz	5	5	01
11/18/2020 04:29:45.516 PM	00250C00010008D3	Uplink		22	🔴 -131	🔴 -14	0	902.7 MHz	2	5	01
11/18/2020 04:29:17.228 PM	00250C00010008D3	Uplink		20	🔴 -129	🟡 -7	0	902.5 MHz	1	6	08

With the added metadata, the Senet provides information to troubleshoot issues with the devices, such as if communication strength was unusually low. There are many graphs that display useful information about past received data, such as if signal strength is decreasing or if packets have stopped going through which could indicate a node failure. The graphs also show Ethernet errors and Gateway Utilization. One of the most important features of the Senet server is its ability to store and forward the data. Details regarding this will be discussed in § 5.2.6. All in all, the Senet Hub is a tool to view and debug the status of the system's communications.

5.2.6 API & Lambda Function AA

The Sensor Pod data that is received and decoded by the Senet server is stored on the Senet platform. This platform/database has a limited storage size and provides a limited application interface for detailed retrieval of data. This is not an ideal method since the data stored must be cleansed so that applications can interpret it and make trends. The solution to this issue is to forward the data from the Senet server to an external database that allows for more control and long-term storage of data. To accomplish this solution, a link forwarder on the Senet server is used to forward data. The link forwarder uses a RESTful API to generate a POST call to forward the data to an API endpoint.

The API endpoint interface will be used to route RESTful calls to a Lambda function. The Lambda function is a serverless program written in node.js. It is used here to modify the data to match the database models before sending it to the database.

The API endpoint will be implemented with AWS API. As shown in Figure 36, once the API call reaches the AWS API endpoint, it is validated for authorization and request structure issues. If the call does not contain a valid authorization key, the API will terminate and return a

403 code. If there is an issue with the call structure, a 4xx/5xx code is returned. If both criteria are met the call is finally routed to the Lambda Function.

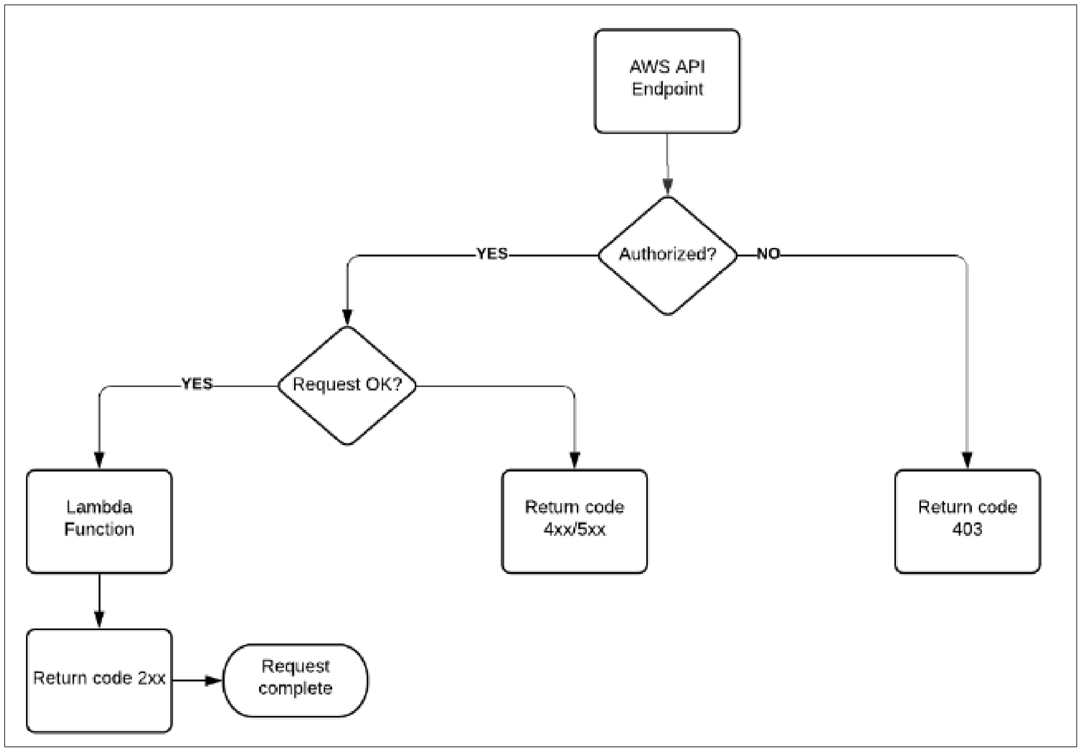


Figure 57: API & Lambda Function Flowchart.

Once the Lambda Function receives the API call it will process the call through a function as described in the pseudo code in Figure 58.

```
dynamo = new AWS.DynamoDB.DocumentClient();
API_Route_Handler(call)
  try {
        data = call.body;
        Item = convertToDatabaseItemFormat(data);
        Item.data.convertHEXToString();
        body = dynamo.put(Item);
        break;
      }
  } catch (err) {
    statusCode = '400';
    body = err.message;
  } return {
    statusCode = 200,
     };
};
```

Figure 58: Lambda Function Pseudo Code.

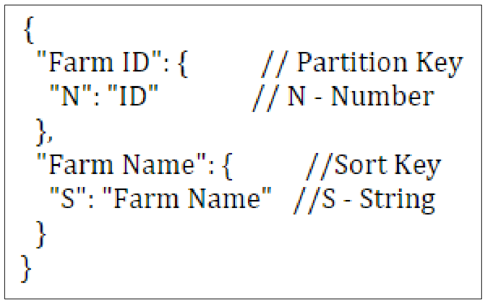
The function first parses the API call body, converting it to match the database item format model. The actual data portion is then converted from data type HEX to a String. Once the information is converted, a DynamoDB API call is made to place the item into the database table. If no issue occurs with this process, the data is successfully placed in the database and the Lambda Function will notify the API Endpoint to return a success code of 200 back to Senet. Otherwise, the error message along with an error code of 400 is returned.

5.2.7 *Database* **AA**

The database that is being used is AWS DynamoDB, a type of database based upon NoSQL. This type of structure is more flexible and scalable to work with since its structure can be modified. DynamoDB uses database tables to store data, which are ultimately held in database Items. These Items are composed of a Partition key and an optional Sort key which will be discussed when further explaining the tables in the database. For the Soil Sensor database, three tables will be used for managing data: Farm Table, Sensor Data Table, and Sensor Pod Table.

5.2.7.1 *Farm Table* ***AA***

The Farm Table is used to sperate information for different farms. The information contained in the Farm Table will be a Sensor Pod list as well as the sensor data from the pods. The Farm Table baseline, as shown in Figure 38, will require items to be composed of a Partition Key (Farm ID) and a Sort Key (Farm Name).



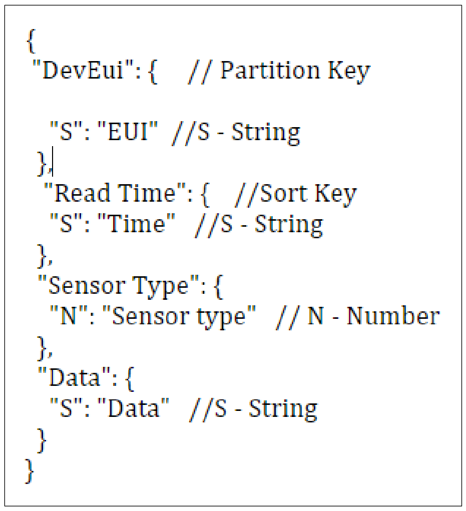
```
{
 "Farm ID": {        // Partition Key
   "N": "ID"         // N - Number
 },
 "Farm Name": {       //Sort Key
   "S": "Farm Name"  //S - String
 }
}
```

Figure 59: Farm Table Baseline.

The Farm ID will be used to maintain a unique ID for each farm, while the Farm Name will be used for sorting against a user set Name for their Farm. The Farm ID will also be used as a parent ID for other tables, which will allow for querying of data from other tables that are connected to specific Farm.

5.2.7.2 *Sensor Data Table* ***AA***

The Sensor Data Table holds data from all pods regardless of the farm in which it belongs. The baseline for this table, as shown in Figure 60, used Partition Key DevEUI to denote which Sensor Pod the data originated from along with a Sort Key Read Time to allow sorting to be used for data querying.



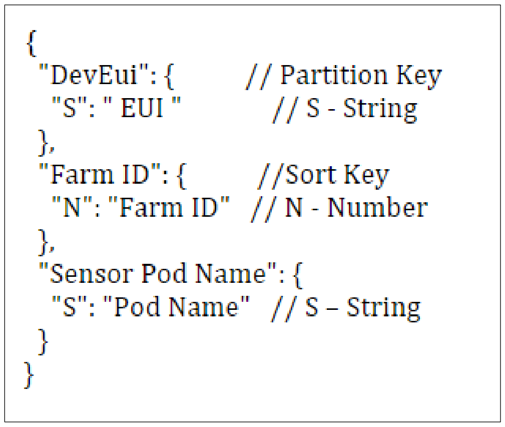
```
{
 "DevEui": {    // Partition Key

   "S": "EUI"  //S - String
  },
  "Read Time": {   //Sort Key
   "S": "Time"   //S - String
  },
  "Sensor Type": {
   "N": "Sensor type"   // N - Number
  },
  "Data": {
   "S": "Data"   //S - String
  }
}
```

Figure 60: Sensor Data Table Baseline.

The table has additional fields that hold the data and the data type. The DevEUI will be used as child link information in the Sensor Pods Table, which will specify which farm the pod and the data belong to for data modeling and populating the web application display.

5.2.7.3 *Sensor Pod Table* ***AA***

The Sensor Pod Table is used as a list of all Sensor Pods, holding information regarding Sensor Pod information as well as which Farm the pod is a part of. The baseline for the Sensor Pod Table is shown in Figure 61.



```
{
 "DevEui": {       // Partition Key
  "S": " EUI "       // S - String
 },
 "Farm ID": {       //Sort Key
  "N": "Farm ID"   // N - Number
 },
 "Sensor Pod Name": {
  "S": "Pod Name"   // S – String
 }
}
```

Figure 61: Sensor Pod Table.

Similar to the Sensor Data Table, the Partition Key is Dev Eui while the Sort Key is Farm ID. Additionally, there is an optional Sensor Pod Name for user-specified naming. The Dev EUI is used for identification of the Pod but also acts as a Parent link for Sensor Data. The Farm ID is used as Child link to the Farm Table for designating which farm the Pod belongs. The Sensor Pod list acts as a way to couple the Farm Table and Sensor Data Table as well as keeps a track of all Pods that are active.

5.2.7.4 *Interfacing* ***AA***

The DynamoDB Database uses a handful of interfaces to access the information stored in it. For a DynamoDB application, two main methods can be utilized for interfacing with the database: AWS.DynamoDB.DocumentClient and Amazon.DynamoDBv2. These two methods use the same underlying DynamoDB API with variations dependent on programming language. As explained in the Data Flow section, the database interface is accessed from both AWS API /

Lambda and the Web Application. When accessing from the AWS API / Lambda, AWS.DynamoDB.documentClient() is used for interfacing. Since AWS API / Lambda and DynamoDB are both integrated into AWS services, no authentication is needed to access the database. The AWS.DynamoDB.DocumentedClient() is a Class in node.js that uses the DocumentedClient DynamoDB API to make calls to the database for reading and writing data.

When Accessing the database from the web application backend using Amazon.DynamoDBv2, authentication is needed. The web application backend settings hold a set of AWS Security Keys that are needed for authentication when accessing the database. The Amazon.DynamoDBv2 is a DynamoDB software development kit that allows for .NET Core to make DynamoDB API calls to the database for reading and writing data. Both these methods are used to access the database information in a secure and efficient way.

5.2.8 Web Application AA

The web application is designed based on a Model, View, Controller (MVC) design. The MVC design, as the name suggests, breaks all web application operation into three groups: model, view, and controller. The model operation is all logic involving modeling data, view is all operations involving displaying information to the user, and controller is all operations involving user interaction with the web application. The standard flow of this design separates the web application into a frontend and backend. The user interacts with the frontend controller, which then makes a request to the backend logic. The backend receives the request and handles all logic with modeling and fulfilling the request before returning it to the frontend view. The frontend view will then display the requested information.

The backend application runs on a dedicated server and handles all model and other operation logic. In the case of the Soil Moisture Sensor Application, this will be all logic involved with connecting to the database and modeling the information from the database. The purpose of offloading this kind of logic to the backend is to reduce the amount of computation that the browser end (frontend) needs to perform. This is favorable due to the limited number of computational resources on the browser; the server end (backend) is more suitable for these computations due to the greater number of resources available.

The application frontend runs on the user's browser and handles all webpage display rendering (view) and all user requests (controller) to the backend logic. By separating responsibilities, the user experience will be more fluid and responsive. It also allows for a layer of security to be implemented by placing restrictions on request from the frontend by means of security tokens.

The application frontend and backend need a method to communicate between each other for transfer of request and replies. This is accomplished by implementing an API on the backend that will route, process, and model all requests from the frontend. This is an asynchronous method, where the request makes a promise API call to the backend. The frontend will then wait until the request is fulfilled before utilizing the data from the request; while waiting for this promise, the frontend will process and compute other information that is not dependent upon the requested data.

5.2.8.1 *Frontend Web Application* *AA*

To give users the ability to use and access the Wireless Sensor System, an easy-to-use interface will be created. The general flow of user interaction is shown in Figure 62. When the user starts the application, they will be taken to homepage. The homepage will greet the user and request the desired Farm ID. Once a specific ID is given, the user will be taken to the desired Farm Overview page. There will also be a side menu to allow the user to navigate to other sections of the application at any time once a farm has been selected. The application will be spilt into three main pages consisting of a Homepage, Farm Overview, and Sensor Pod List. The pages are further explained in § 5.2.8.1.1 Web Pages.

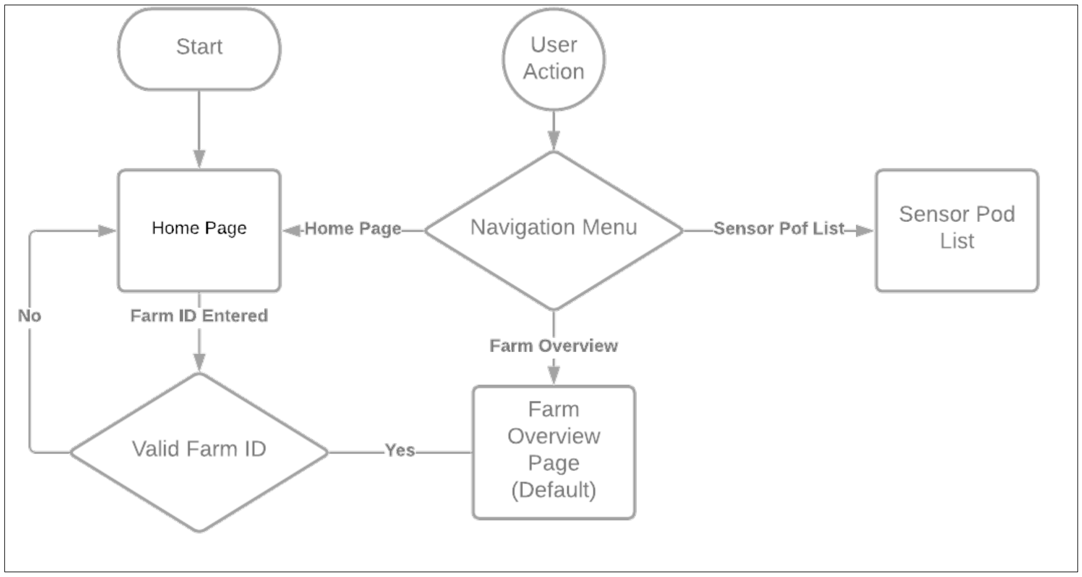


Figure 62: Level 3 Web Application View and Controller Flowchart.

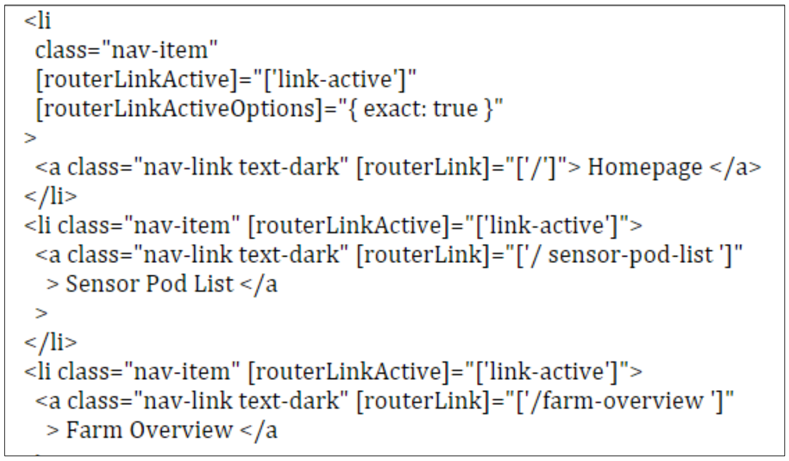
5.2.8.1.1 Web Pages (View)

The three main pages of the web application – Homepage, Farm Overview, and Sensor Pod List – each serve a specific purpose. Each page makes an API call to the backend for required data needed to display prior to rendering the page.

The Homepage is used as an entry point for the application; it greets the user and determines the farm in which the user wishes to view and interact. The page makes a call to the backend to validate Farm ID before calling for the page to route to the Farm Overview page of the desired valid Farm ID. The Farm Overview page is used a means to display all important information from the Soil Sensor System belonging to a desired Farm. The page also displays all data trending for the farm as well a small list of all Sensor Nodes associated with the Farm. The Sensor Pod List page is used as detailed list of all Sensor Pods associated with the current farm. This list will provide all information available related to Sensor Pod. All data displayed on these pages is requested from the backend prior to the page rendering.

5.2.8.1.2 Interaction (Controller)

As stated earlier the frontend application only handles view and controller operations. In the figure all transition are handles by API calls to the backend based on the user's interaction. There are two main types of API calls to the backend: data call and logic call. The data call requests data from the database and will expect a data returned in a model that fits the current page. Logic call requests logic operation such as validation, calculation, and other logic operation too heavy for the frontend to handle, the request return results of the logic operation. A good example of a logic call to the backend is a request to navigate to another page as shown in Figure 63 and 64.



```
<li
  class="nav-item"
  [routerLinkActive]="['link-active']"
  [routerLinkActiveOptions]="{ exact: true }"
>
  <a class="nav-link text-dark" [routerLink]="['/']"> Homepage </a>
</li>
<li class="nav-item" [routerLinkActive]="['link-active']">
  <a class="nav-link text-dark" [routerLink]="['/ sensor-pod-list ']"
    > Sensor Pod List </a
  >
</li>
<li class="nav-item" [routerLinkActive]="['link-active']">
  <a class="nav-link text-dark" [routerLink]="['/farm-overview ']"
    > Farm Overview </a
```

Figure 63: Frontend Navigation Calls.

```
RouterModule.forRoot([
    { path: '', component: HomepageComponent, pathMatch: 'full' },
    { path: 'sensor-pod-list, component: SensorPodListComponent },
    { path: 'farm-overview', component: FarmOverviewComponent}
  ]),
```

Figure 64: Frontend Routing Table.

5.2.8.2 *Backend Web Application* *AA*

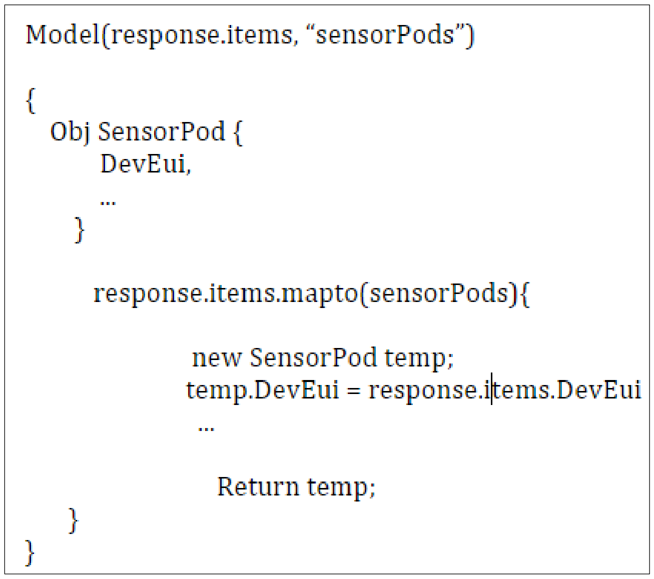
The backend application, as explained earlier, is designed to handle all logic including gathering and modeling data from the database, then returning the data to the frontend to be displayed. The following example will demonstrate how these calls are handled for data call and modeling. Most logic and data calls are written in the same formats; a sample of the code will be shown below, and the full code can be referenced in the Appendix.

When the backend receives a data API call, it is routed to the proper API Controller. An example of the API Controller for DynamoDB is shown in Figure 65. The API call contains the information for the operation to which the call refers. In the example, the API call is as such: (api/DynamoDB/getitems(parameter)).

```
[ApiController]
  [Route("api/DynamoDb")]
  public DynamoDbController()
  {
  [Route("getitems")]
    GetItem(table , parametar)
    {
      request(table , item)
      response =  await dynamoDbClient.ScanAsync(request);

      if(response)
        Return Model(response.items, “sensorPods”)
      Else
        Return emptyItem;
   }
```

Figure 65: Backend API Controller Pseudo Code.

As shown, the API will make a call to the DynamoDB database to collect all data related to the request. This data is then modeled based on the type of information that is being requested. An example for this case is shown in Figure 66. The data returned is mapped to a model that is appropriate for the frontend application.



```
Model(response.items, “sensorPods”)

{
   Obj SensorPod {
         DevEui,
         ...
      }

        response.items.mapto(sensorPods){

                    new SensorPod temp;
                    temp.DevEui = response.items.DevEui
                     ...

                       Return temp;
      }
}
```

Figure 66: Backend Modeling Pseudo Code.

The information that is returned to the front is the data that was requested which is also modeled to meet the frontend view format. Logic call are similar to this without the need for modeling of the information; instead, the request result is returned.

5.3 PROTOTYPES: DESIGN VERIFICATION

To show the final design has the ability to be implemented next semester, prototypes were designed and implement to test proof of concept.

5.3.1 Voltage Regulator **AA, LF, AW**

The circuit from Figure 17 was used to construct a test circuit for the XC9140A331MR-G voltage regulator. The circuit consisted of two capacitors and one inductor from UA stock with ratings of 10 µC, 4.7 µC, and 4.7 µH, as seen in Figure 67. The circuit was first powered using a voltage source in order to show that the system would regulate any voltage to approximately

3.3V. It was quickly found that this would not work due to the fact that voltage sources do not supply a high enough current to trigger the switching function of the regulator.

The circuit was then powered using a battery. The battery voltage was measured to be 3.9V as the input for the circuit. The output voltage was approximately 3.2V, showing the voltage regulator working as intended. This was not quite the 3.3V expected, but it is an acceptable result. There are a few possible reasons why the result was not what was expected. The first is the breadboard itself. There is a possibility that some of the wires shorted to other rails, causing voltages to change and be inaccurate. The second is the result of a no-load circuit. The output voltage was measured without respect to a load of any kind. This could change the results of the voltage that is being outputted. The final is a connection issue with the XC9140A331MR-G and the adapter. A faulty connection to the board or an inconsistent connection could cause the output to not be what was desired.

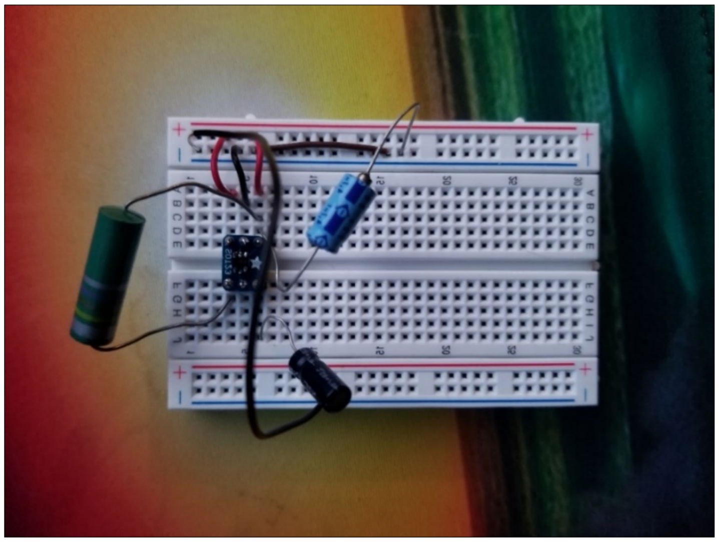


Figure 67: XC9140A331MR-G Voltage Regulator Prototype Circuit.

After additional research it was found that the XC9140A331MR-G was not the most optimal regulator to be used, as it only had an 80-90% efficiency, being the most likely reason the voltage stabilized at 3.2V rather than 3.3V. A STBB1-APUR buck-boost regulator was breadboarded, which boosted the lower-end voltage up to 3.3V and bucked the upper-end voltage down to 3.3V. This type of regulator has a 97% efficiency.

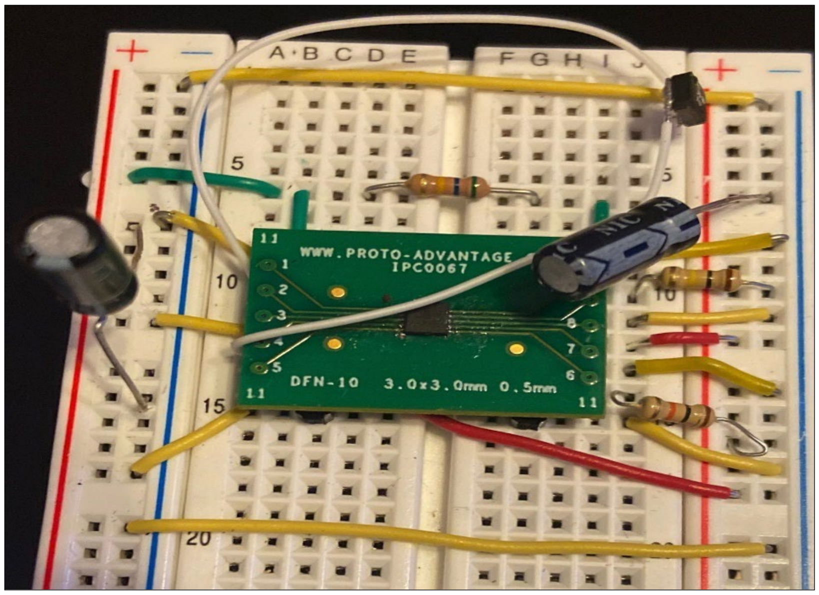
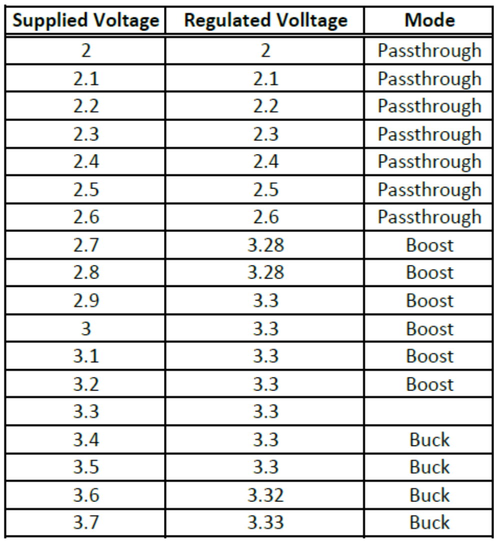


Figure 68: STBB1-APUR Buck-Boost Voltage Regulator Prototype Circuit.

The STBB1-APUR regulator was never fully tested before the implementation stage due to time constraints, but when implemented onto a PCB, the circuit performed as expected.



Supplied Voltage	Regulated Volltage	Mode
2	2	Passthrough
2.1	2.1	Passthrough
2.2	2.2	Passthrough
2.3	2.3	Passthrough
2.4	2.4	Passthrough
2.5	2.5	Passthrough
2.6	2.6	Passthrough
2.7	3.28	Boost
2.8	3.28	Boost
2.9	3.3	Boost
3	3.3	Boost
3.1	3.3	Boost
3.2	3.3	Boost
3.3	3.3	
3.4	3.3	Buck
3.5	3.3	Buck
3.6	3.32	Buck
3.7	3.33	Buck

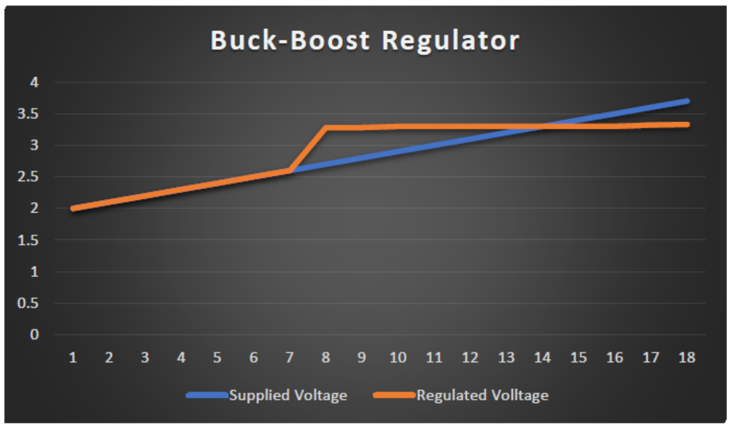


Figure 69: Data Collection from STBB1-APUR Buck-Boost Voltage Regulator.

Data from the PCB regulator was trended and plotted. The regulator operated in passthrough mode until reaching 2.7V. It then transferred to boost mode, increasing the voltage to 3.3V. Once exceeding 3.3V, the regulator entered buck mode in which it decreased the voltage to 3.3V. Another way to verify the regulator was operating as expected is to look at the value when the supply voltage was at 3.3V. For the previous regulator, the regulated value when 3.3V was supplied fluctuated quite a bit due to only being 80-90% accurate. In the case of the buck-boost regulator, the voltage remained constant at a 3.3V supplied voltage.

5.3.2 Soil Moisture Sensor AW

A parallel plate capacitive soil moisture sensor was constructed using two copper pennies with a plastic dielectric. The capacitor was placed in two types of soil and measured time constants when water was added incrementally. The data points were plotted to determine if a direct relationship between frequency and capacitance existed. This capacitor was designed to test proof-of-concept for the PCB capacitor that will be implemented next semester.

5.3.2.1 *Setup and Procedure*

The circuit from Figure 25 was constructed using a breadboard, Diligent Analog 2 Discovery, and UA stock components. Two types of soil were collected and tested. The first was the Earth & Wood Super Soil, which has a similar dielectric constant to fertilized soil in a field. The second was a more porous (sandy) soil, chosen to be compared to the fertilized soil to observe the difference in rate of saturation. Both soil types were baked at 150ºC for two hours to extract the moisture from the soil. Once cooled, 3 oz of soil from each type was placed into 5oz containers to be tested. The capacitive soil moisture sensor setup can be seen in Figure 70.

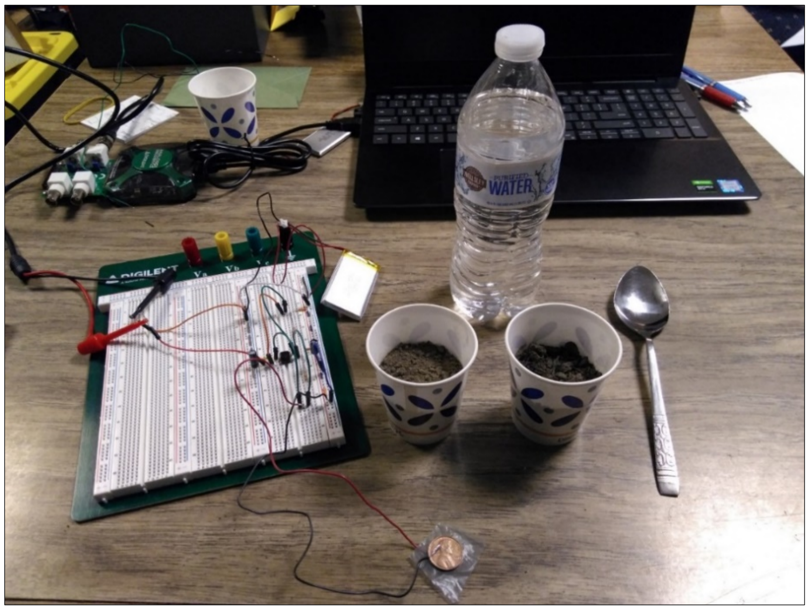


Figure 70: Capacitive Soil Moisture Sensor Prototype Setup.

Capacitive soil moisture sensors must be calibrated to the environment. To verify the range of the sensor, values were taken for when the sensor was in air (minimal moisture) and when the sensor was in water (maximum moisture). The sensor was then placed in each container, as seen in Figure 71, and collected readings of the soil when water was added incrementally by 1 Tbsp from 0 Tbsp to 5 Tbsp.

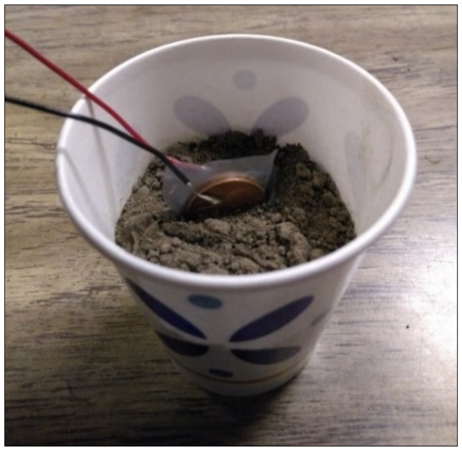


Figure 71: Moisture Sensor in Soil.

5.3.2.2 *Results*

The upper and lower calibration limits were first found to be used as control values by measuring the time it took to discharge in air, Figure 30, and the time it took to discharge in water, Figure 72. The time it takes a capacitor to discharge is known as the time constant.

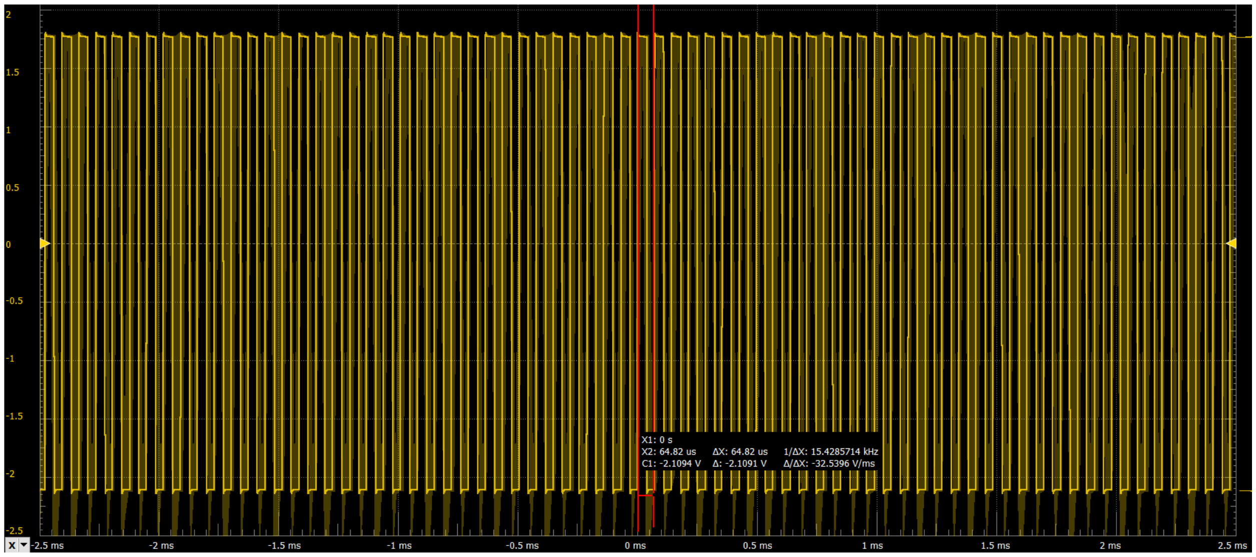


Figure 72: Soil Moisture Sensor Control Lower Limit: Air.

The smallest time constant possible was found to be 64.82 μs, equivalent to a 15.43 kHz frequency.

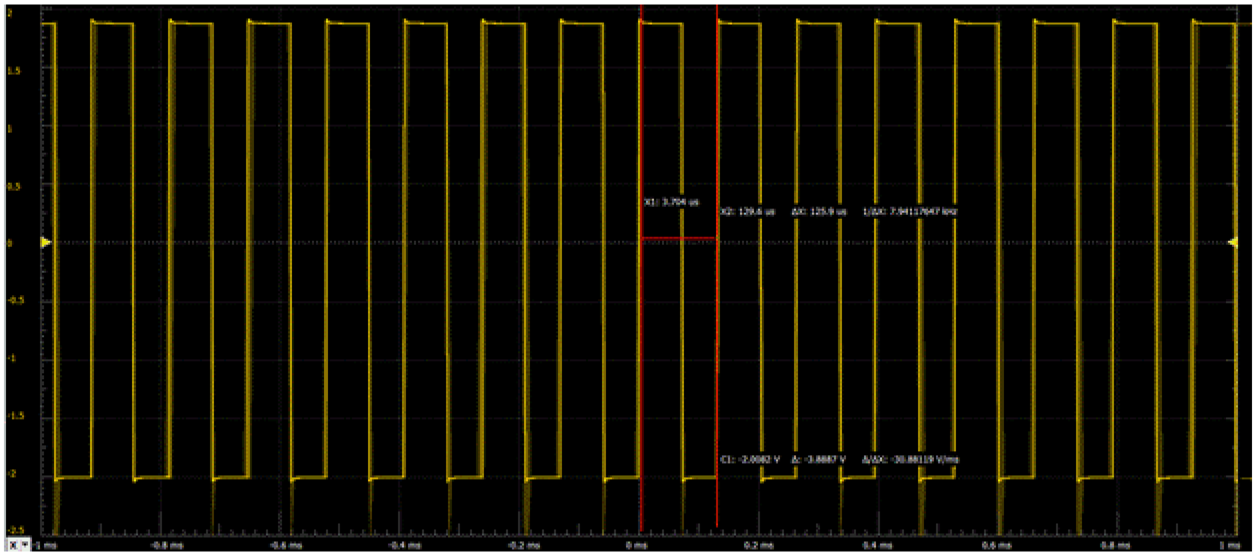


Figure 73: Capacitive Soil Moisture Sensor Control Upper Limit: Water.

The largest time constant possible was found to be 124.7 µs, equivalent to a 7.94 kHz frequency. Once the upper and lower limits of the discharge rate were defined, the time constant for dry Super Soil was measured. The results can be seen in Figure 74.

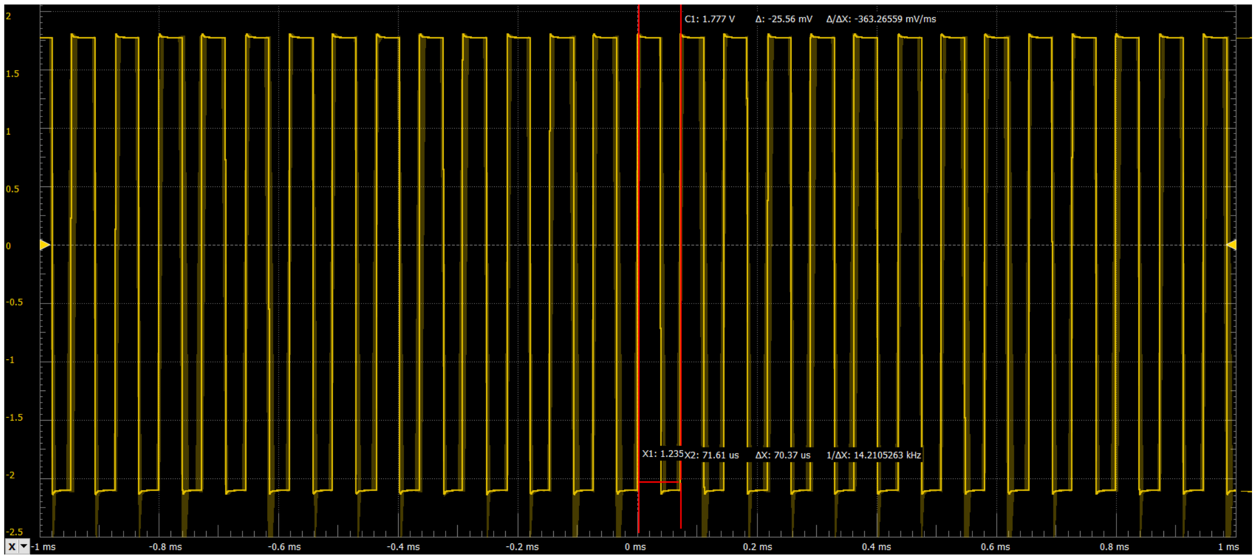


Figure 74: Container 1: Super Soil Dry.

The time constant for dry, fertilized soil was measured to be 70.37 µs, equivalent to 14.21 kHz. One Tbsp of water was then added to the container, and the waveform was captured as seen in Figure 75.

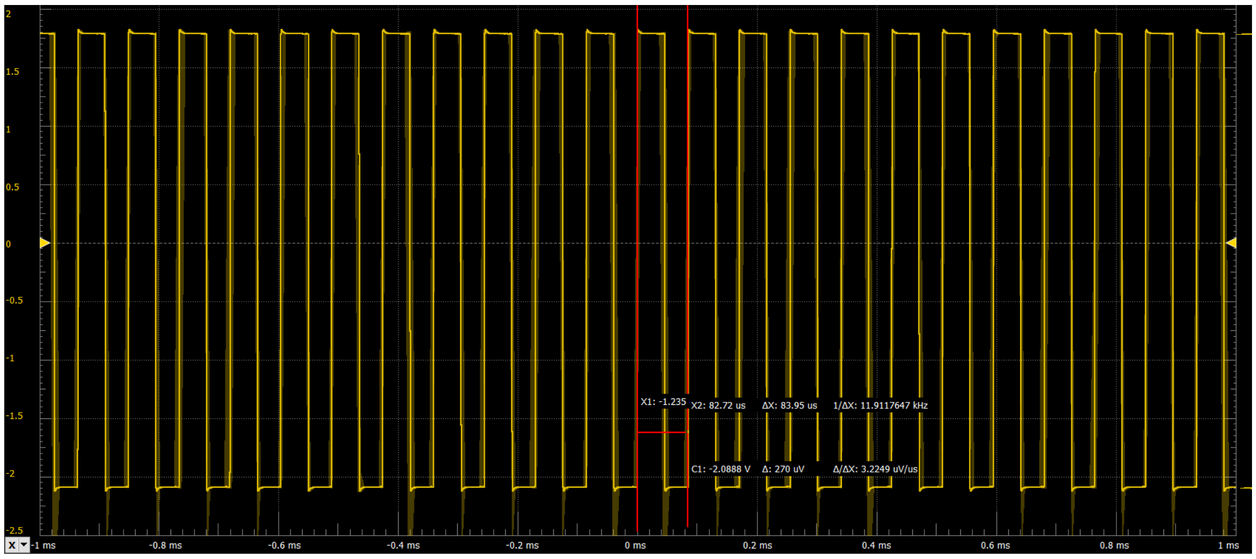


Figure 75: Container 1: Super Soil with 1 Tbsp Water Added.

The time constant changed to 83.95 µs, equivalent to 11.91 kHz.

The slow increase in time constant was to be expected. When the capacitor is inserted into the soil, the soil essentially becomes part of the dielectric between the parallel plates of the capacitor. When the dielectric decreases, the capacitance increases, as proven in Equation 20. A larger capacitance directly corresponds to a longer time to discharge, meaning a larger time constant. The amount of water in the soil was then increased incrementally to 2 Tbsp, 3 Tbsp, 4 Tbsp, and 5 Tbsp, and data was recorded for each increment.

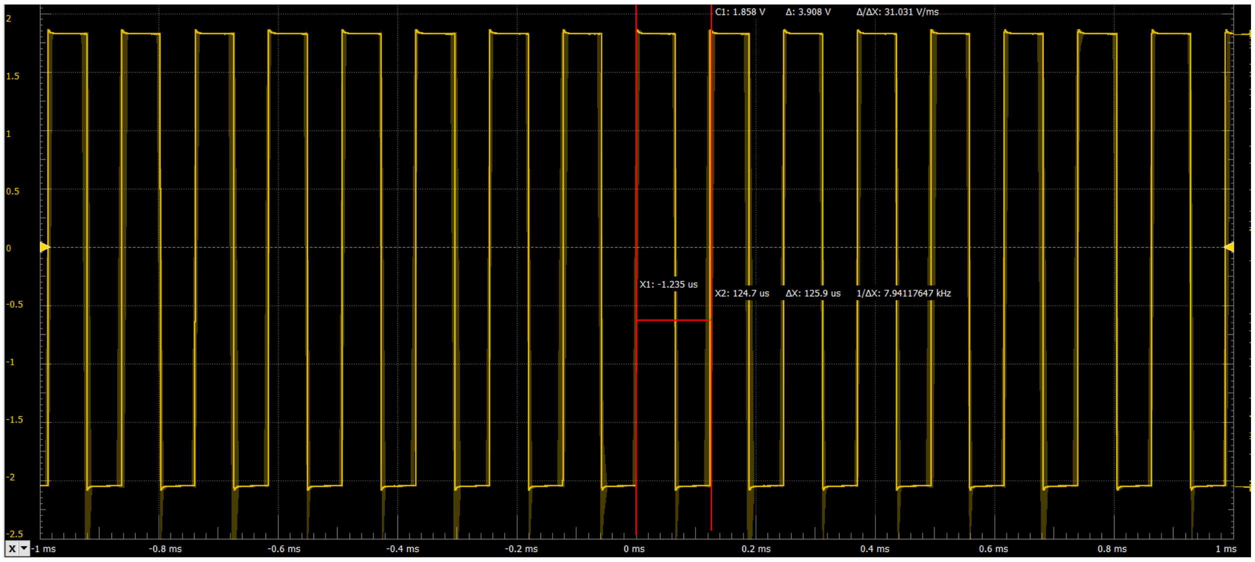


Figure 76: Container 1: Super Soil with 5 Tbsp Water Added (Saturation).

When a fifth tablespoon of water was added, the soil reached saturation and could not absorb any more water. The soil saturated at a time constant of 124.7 µs, or at 7.94 kHz. The results can be seen in Table 43.

Once saturation was reached for the Super Soil, the capacitive sensor was placed in the porous (sandy) soil and process began again.

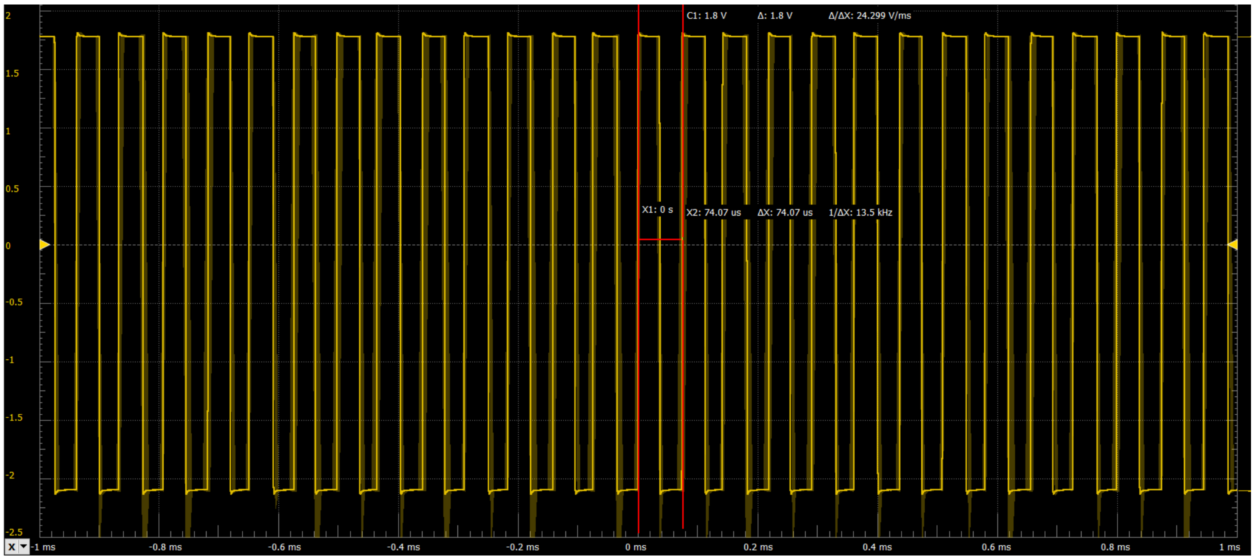


Figure 77: Container 2: Sandy Soil Dry.

The time constant of sandy soil was measured to be 74.07 μs, equivalent to 13.5 kHz.

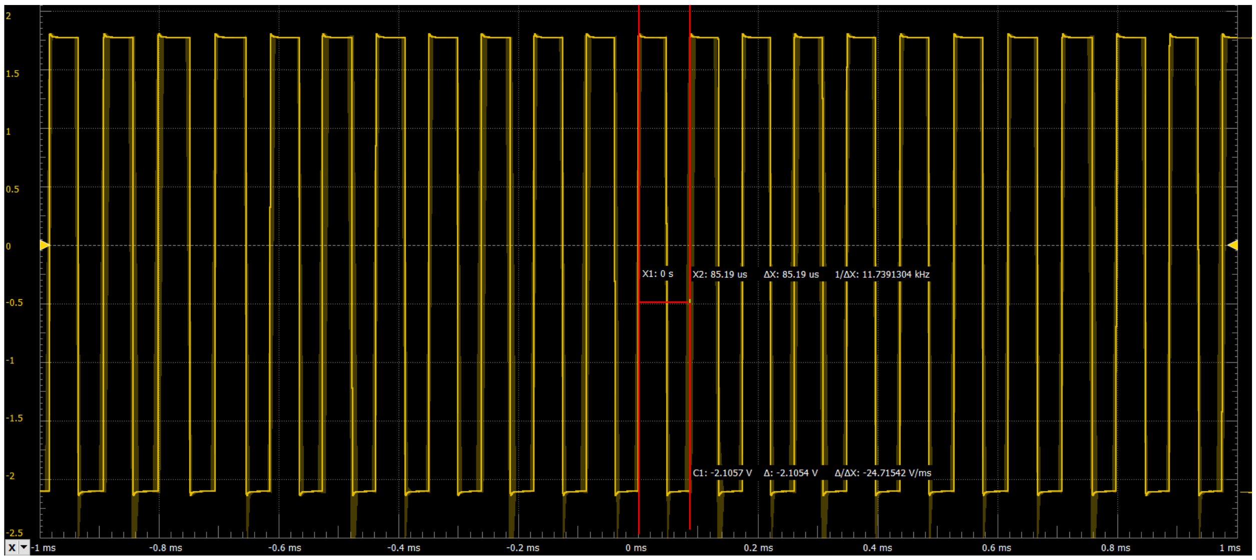


Figure 78: Container 2: Sandy Soil with 1 Tbsp Water Added.

The time constant of sandy soil with 1 Tbsp of water added was found to be 85.19 μs, or 11.73 kHz. The same procedure followed for the Super Soil was also followed for the sandy soil, increasing water content incrementally by 1 Tbsp and recording data for each increment.

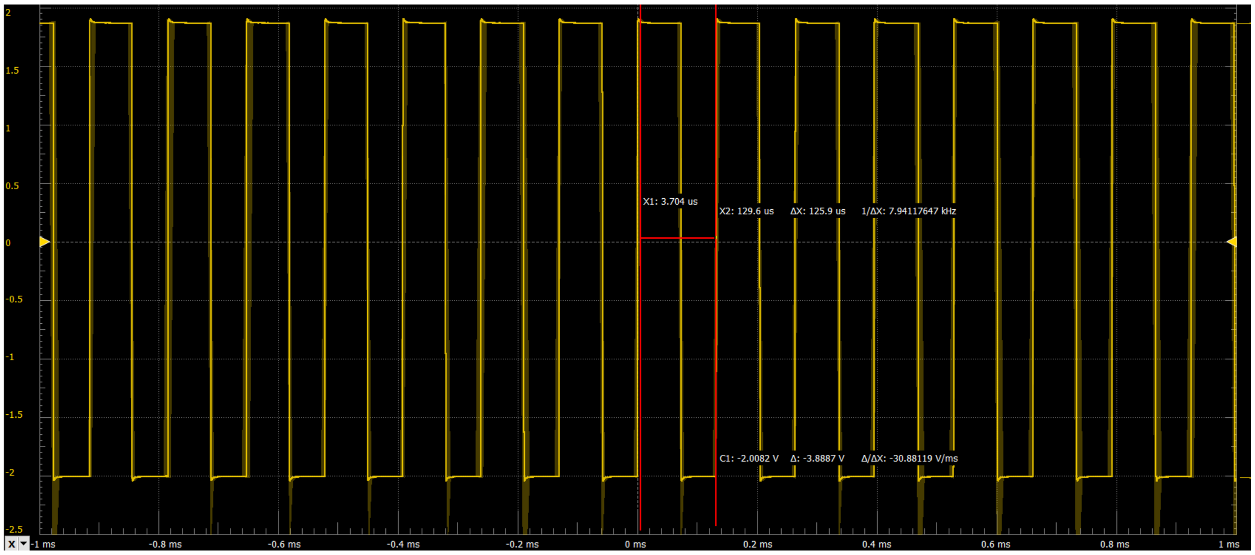


Figure 79: Sandy Soil with 4 Tbsp Water Added (Saturation).

Once 4 Tbsp of water were added to the sandy soil, the soil reached saturation. The results can be seen and compared to the Super Soil in Table 43.

Table 43: Soil Moisture Measurements with Incremental Increase of Water.

Water Added (Tbsp)	Super Soil		Sandy Soil	
	Time Constant (µsec)	Frequency (kHz)	Time Constant (µsec)	Frequency (kHz)
0	70.37	14.2	74.07	13.5
1	83.95	11.9	85.19	11.7
2	97.53	10.3	97.53	10.3
3	101.2	9.8	115.4	8.66
4	122.2	8.2	125.9	7.9
5	124.7	7.9	125.9	7.9

As can be seen from the results, the sandy soil started with a higher time constant and saturated more quickly than the Super Soil. This is to be expected because the sandy soil is more porous than the Super Soil; more porous soil contains more aggregate and therefore cannot absorb water as easily as an organic soil. The frequency data was plotted for both types of soil and can be seen in Figure 80.

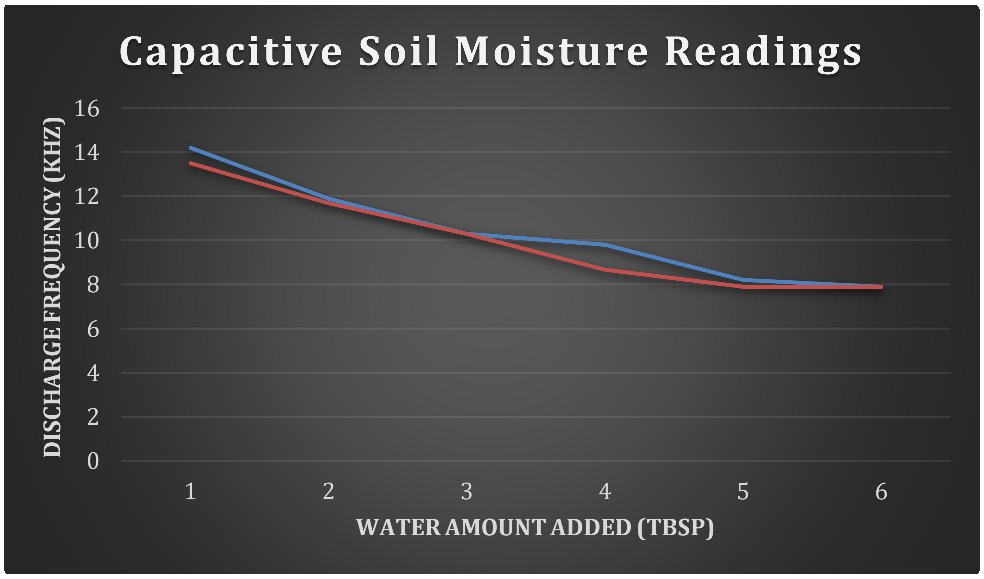


Figure 80: Soil Moisture Measurements Frequency vs. Additional Water Graph.

The frequency of the soil has a relatively linear decrease until saturation is reached in both the case of the Super Soil and sandy soil. The graph verifies that there is a direct correlation between frequency and amount of water added. The given data also verifies that the microcontroller will be able to easily read and interpret soil moisture data accurately by converting the time constant to frequency.

5.3.3 Temperature Sensor LF

Due to extenuating circumstances, the MAX6607IUK-T was not available for use in the first prototype demonstration. Instead, the TC1047A onboard temperature sensor was used, shown in Figure 81.

Figure 81: TC1047A Temperature Sensor.

It has a similar pin layout and operation voltage to the MAX6607. Both sensors also give the same output, as seen in Equation 22. This allows for either part to be interchangeable for testing purposes. As seen in Figure 82, the MAX is a 4-pin integrated circuit while the TC, shown in Figure 81, is only a 3 pin.

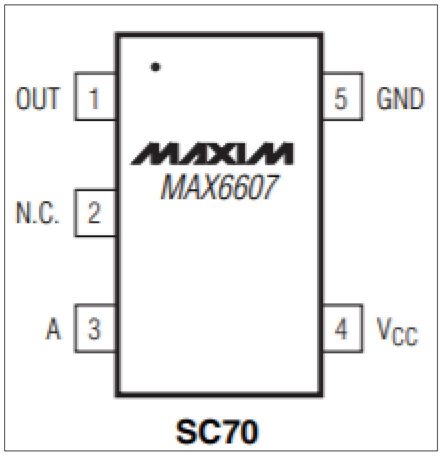


Figure 82: TC1047A Temperature Sensor Pinout.

The reason these can both be seen as 3 pin layouts is because the A and GND are both grounded, giving a pseudo-3 pin layout. Both integrated circuits will also give an analog voltage output that will be converted using the onboard ADC.

5.3.4 Microcontroller Data Collection AA

To demonstrate data collection with the Microcontroller, the Explorer 16/32 development board with a PIC24FJ128GA010 PIM was used along with a capacitive moisture sensor. This is a similar PIM that will be used in the final design of the Wireless Sensor Pod. The demonstration portrays how to read the analog signal from the sensors. Two sensors were used in the program, including the built-in temperature sensor on the development board and a capacitive soil moisture sensor that was connected externally via Input/Output pins. Both sensors output an analog voltage signal that can be captured by the microcontroller and displayed on the Explorer 16/32 LCD display. Applying heat to the temperature sensor on the board will indicate a change in the reading displayed on the screen, which can be seen in Figure 83.

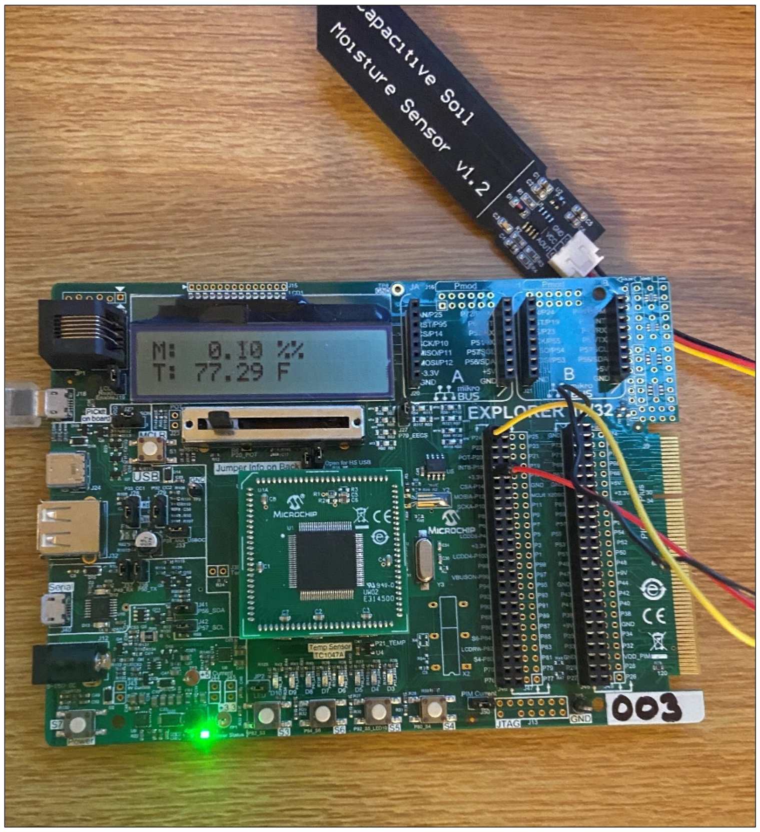
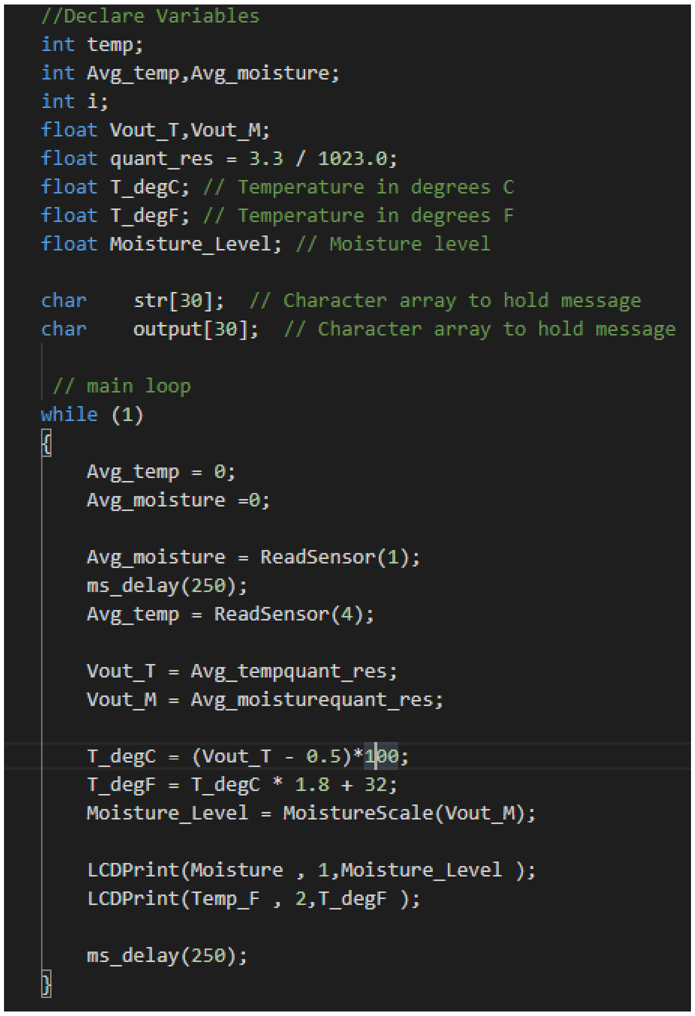


Figure 83: Explorer 16/32 Development Board Demo.

The main component of the demonstration is creating an effective program to read the sensors. The pseudocode in Figure 61 represents the setup and control of the microprocessor. There is a simple while loop which runs periodically and can be controlled with a delay. For each pass in the loop, both the analog ports will be accessed and then the value which they return will be processed and cleansed so that the LCD screen can properly display the temperature and moisture levels. The code in Figure 84 shows the process of data acquisition.



```c
//Declare Variables
int temp;
int Avg_temp,Avg_moisture;
int i;
float Vout_T,Vout_M;
float quant_res = 3.3 / 1023.0;
float T_degC; // Temperature in degrees C
float T_degF; // Temperature in degrees F
float Moisture_Level; // Moisture level

char    str[30];  // Character array to hold message
char    output[30];  // Character array to hold message

 // main loop
while (1)
{
    Avg_temp = 0;
    Avg_moisture =0;

    Avg_moisture = ReadSensor(1);
    ms_delay(250);
    Avg_temp = ReadSensor(4);

    Vout_T = Avg_tempquant_res;
    Vout_M = Avg_moisturequant_res;

    T_degC = (Vout_T - 0.5)*100;
    T_degF = T_degC * 1.8 + 32;
    Moisture_Level = MoistureScale(Vout_M);

    LCDPrint(Moisture , 1,Moisture_Level );
    LCDPrint(Temp_F , 2,T_degF );

    ms_delay(250);
}
```

Figure 84: Main Program.

In the final design of the project, the LCD will be replaced with the LoRa communication subroutine, which will transmit the cleansed data to Gateway, allowing the application to be viewed on the Web Application.

To enable analog to digital conversion of the sensors, a separate file called “ADC.c” was included. This file provides logic for reading the analog channel. A code snippet for the A/D conversion is shown in Figure 85. The ReadADC function is a routine for using the A/D converter and returning the conversion result, which is the acquired temperature or moisture level to be stored in the “temp” variable. Then, each reading is added to the sum in the loop to eventually generate the average value over the course of 32 readings.

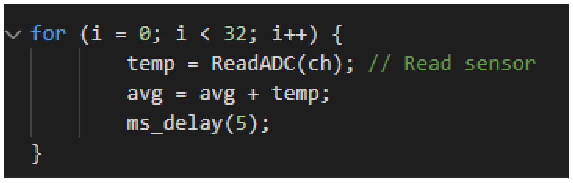

```
for (i = 0; i < 32; i++) {
        temp = ReadADC(ch); // Read sensor
        avg = avg + temp;
        ms_delay(5);
}
```

Figure 85: Analog Sampling.

5.3.5 Lora Module Communication RK

To demonstrate Microchip processor to LoRa module communications, a couple development motes were configured for testing. Development motes are small-sized, prefabricated units that use similar hardware to what will be used in the final design. As seen in Figure 86, the motes feature an LED screen for debugging and information about the current state of the mote. There is also a micro-USB port on the mote to send serial commands to the Lora Modules directly, as the microchip processor simply forwards all UART communication to the RN2903 LoRa transceiver.

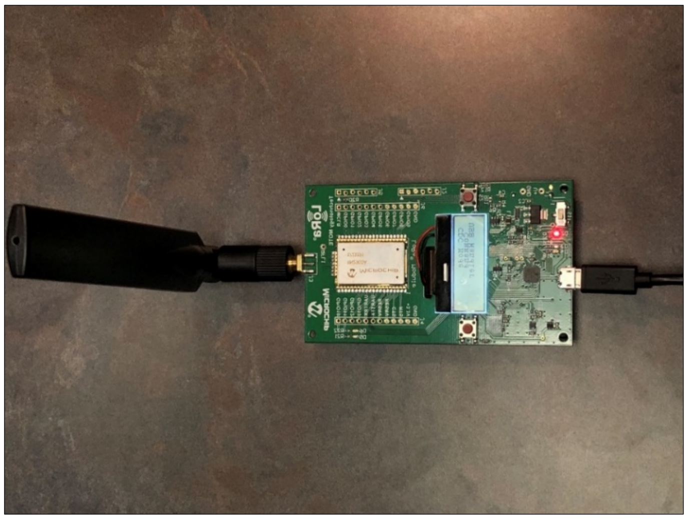
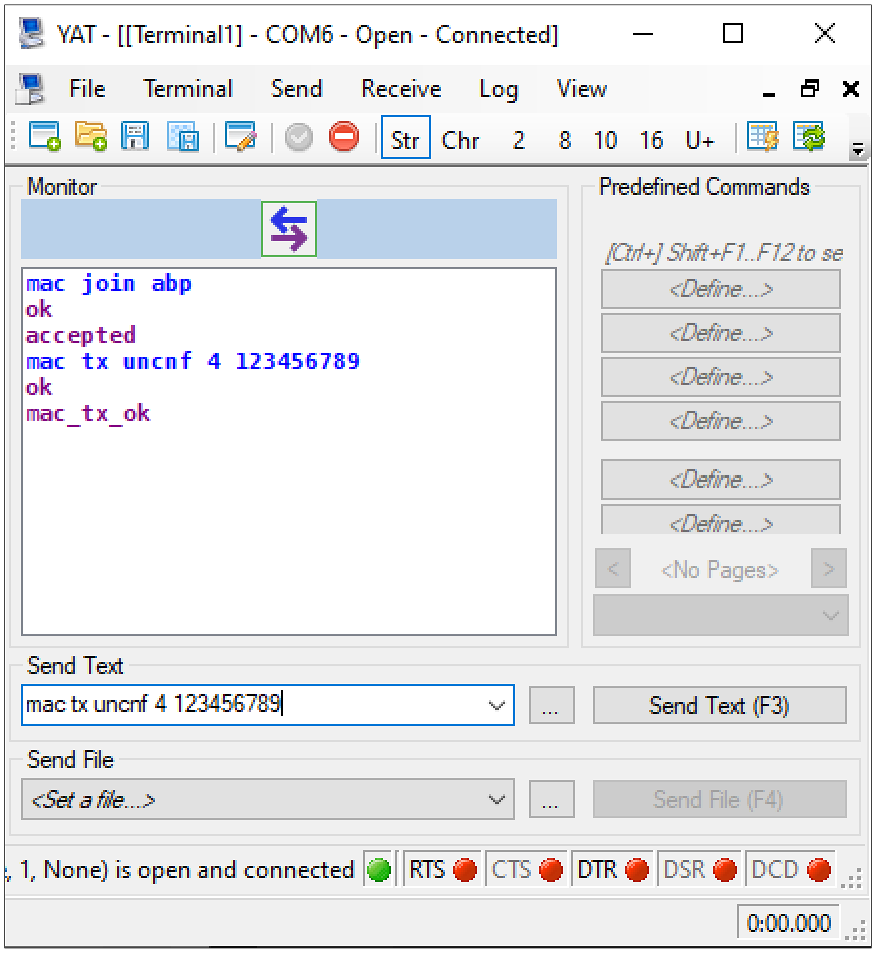


Figure 86: RN2903 LoRa Module with 6" Monopole Antenna.

As mentioned in detail in § 5.2.2, there are a set of parameters on the LoRa transceiver that must be configured before sending data to the Gateway and the Senet Portal. The Mote can be accessed via serial communication with any PC. An application called Yet Another Terminal (YAT) can be used to access the USB Serial Bus. The terminal can communicate with the LoRa chip with a baud rate of 57600 Bps, and Serial Settings of 8/N/1 which means 8 data bits, no parity, and 1 stop bit. Once updating all of the necessary parameters, a "mac join abp" command can be sent which will respond back with "ok" meaning the command was in proper format and was sent, and "accepted" if the join procedure to the network was successful. The command format and response can be seen in Figure 87.



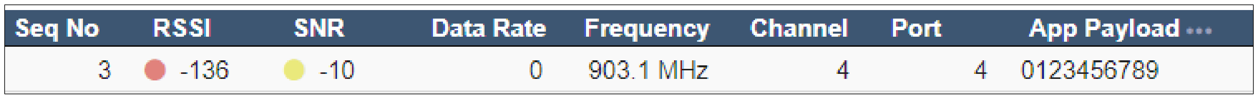
```
mac join abp
ok
accepted
mac tx uncnf 4 123456789
ok
mac_tx_ok
```

Send Text: mac tx uncnf 4 123456789

, 1, None) is open and connected

Figure 87: Screenshot of YAT Terminal for Serial Communication.

After joining the network successfully, messages can be sent to the Gateway and Forwarder. Figure 88 depicts a transmit command being sent. This command contains a keyword “mac” which means the mac OS layer will be accessed to send a message. It is followed by “tx uncnf”, which means a message will be transmitted of unconfirmed type. An unconfirmed message means it does not require an acknowledgement from the Server.



Seq No	RSSI	SNR	Data Rate	Frequency	Channel	Port	App Payload •••
3	● -136	● -10	0	903.1 MHz	4	4	0123456789

Figure 88: Senet Screenshot of Successfully Delivered LoRa Message.

Finally, the last two parameters are the port number and the test payload. Once the command is sent, an “ok” message is received indicating the syntax was correct, and “mac_tx_ok” meaning uplink transmission is successful. In Figure 65, a few pieces of the metadata of that message can be seen as well as the payload which is a hex value. From this demonstration we can see that the Gateway and Forwarder side of the system has been proven and is ready for integration with real Wireless Sensor Pods.

5.3.6 LoRaWAN Propagation Models AW

To verify LoRaWAN communication would work to send signals over a distance of 3.3 km, propagation models were created to theorize how the signal would respond over different distances and weather conditions.

5.3.6.1 *Pathloss Over Distance*

LoRaWAN transmits signals over an unlicensed frequency band, ranging from 433 MHz to 923 MHz, depending on the country in which the signal is transmitted. To best determine how the signals are affected, three frequencies of LoraWAN (433 MHz, 750 MHz, and 923 MHz) were plotted against a distance ranging from 0 to 10 km. Path losses that were considered were those of free space and soil attenuation. The loss due to the soil was calculated previously in § 2.1.3. The path loss components were then subtracted from the total radiated power, defined as 18 mW in § 2.1.3, to determine the signal power that will arrive at the receiver. The received power was then converted to dBm and plotted against distance using MATLAB.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%free space path loss
c = physconst('lightspeed');
freq = [433 750 923].*1e6; %standard frequencies for LoRaWAN comms
R0 = (0:10000);
apathloss = fspl(R0,c./freq);

% path loss over distance given optimal LoRaWAN transmission
Pt_max = 18; %in dBm
Prad = 10^(Pt_max/10); %in mW
PathLoss = 10.^(apathloss./10); %in mW
soilloss = 3.4; %in mW

Prec = (Prad - apathloss.*10^-3 - soilloss); %convert apathloss to mW

Prec_dBm = 10*log10(Prec); %convert mW to dBm

loglog(R0, Prec_dBm);

%plot pathloss over distance for given frequencies
grid on;
legend('Range: 433 MHz', 'Range: 750 MHz', 'Range:923 MHz', ...
'location', 'northeast')
xlabel('Distance (m)');
ylabel('Received Power (dBm)')
title('LoRaWAN Signal Path Loss')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

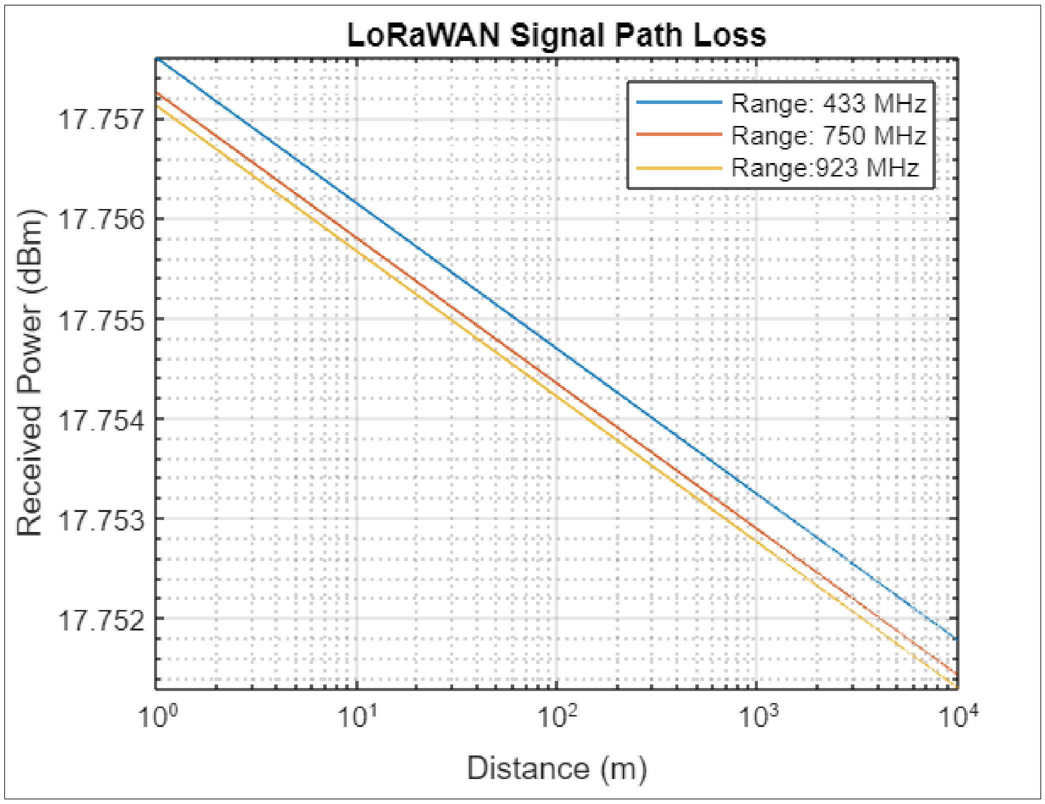


Figure 89: LoRaWAN Signal Path Loss.

Taking all path losses into consideration, it was theorized that only a 0.05 dBm drop would occur between 0 and 10 km for all frequencies of LoRaWAN communication.

5.3.6.2 *Rain Attenuation*

Another important characteristic that should be considered is if LoRa communication is affected by inclement weather. The frequency was swept from 433 MHz to 915 MHz, and rain attenuation was plotted against it in MATLAB.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%propagation loss due to rain
R0 = 1e3;                       % 1 km range
rainrate = [1, 4, 16, 50];      % rain rate in mm/h
el = 0;                         % 0 degree elevation
tau = 0;                        % horizontal polarization
freq = (433:915).'*1e9;

for m = 1:numel(rainrate)
```

```
    rainloss(:,m) = rainpl(R0,freq,rainrate(m),el,tau)';
end

loglog(freq/1e6,rainloss);
grid on;

% subplot(3,1,2)
ylim([0 20]);
legend('Light rain','Moderate rain','Heavy rain','Extreme rain', ...
    'Location','SouthEast');
xlabel('Frequency (GHz)');
ylabel('Rain Attenuation (dB/km)')
title('LoRaWAN Rain Attenuation for Horizontal Polarization');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

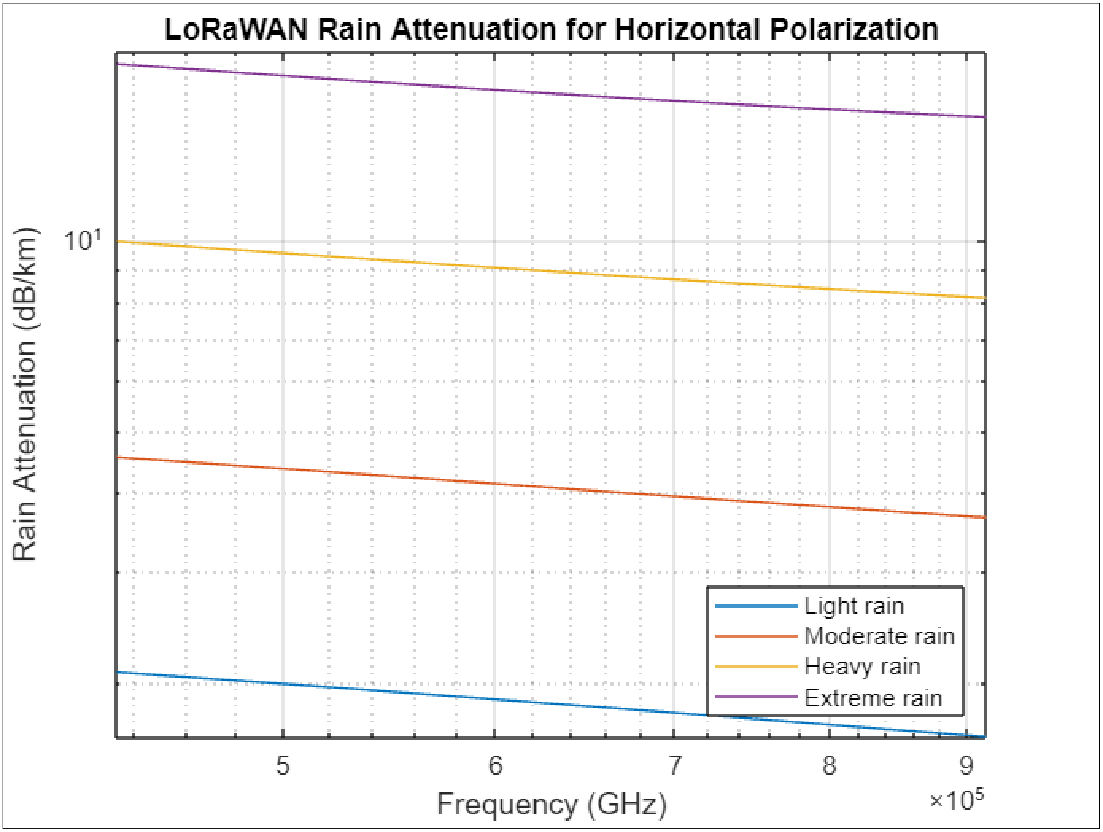


Figure 90: LoRaWAN Rain Attenuation.

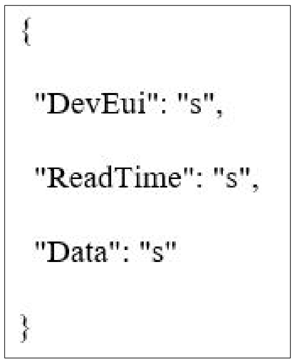
Four different rain rates were plotted to determine if rain had an effect on signal propagation. Rain droplets are known to cause interference in signals with low frequencies because the geometry of the droplet can reflect and scatter the signal [12]. As seen from the

MATLAB simulation, LoRa operates on a high enough frequency band that rain has minimal effect on signal transmission.

5.3.7 Database AA

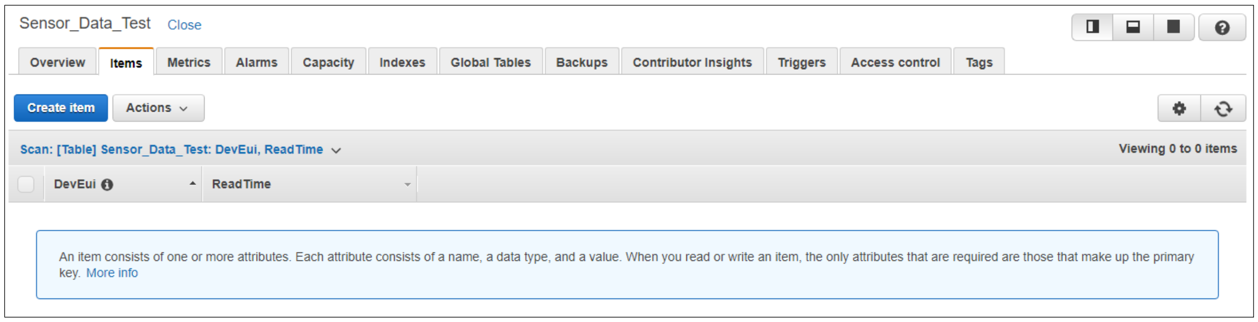
To demonstrate a proof of concept for the DynamoDB database, a test database table, AWS API, and Lambda function were created. The concept was tested populating a test table using API call to AWS API, similar to the way Senet interfaces with AWS API and Lambda Function. The test database table, AWS API, and Lambda function was created and setup as follows.

The test table Sensor_Data_Test table, as shown in Figures 91 and 92, was created following the baseline standard for Sensor Data table explained in The Database section.



```
{
  "DevEui": "s",
  "ReadTime": "s",
  "Data": "s"
}
```

Figure 91: Sensor_Data_Test_Baseline.



Scan: [Table] Sensor_Data_Test: DevEui, ReadTime

Viewing 0 to 0 items

DevEui	ReadTime

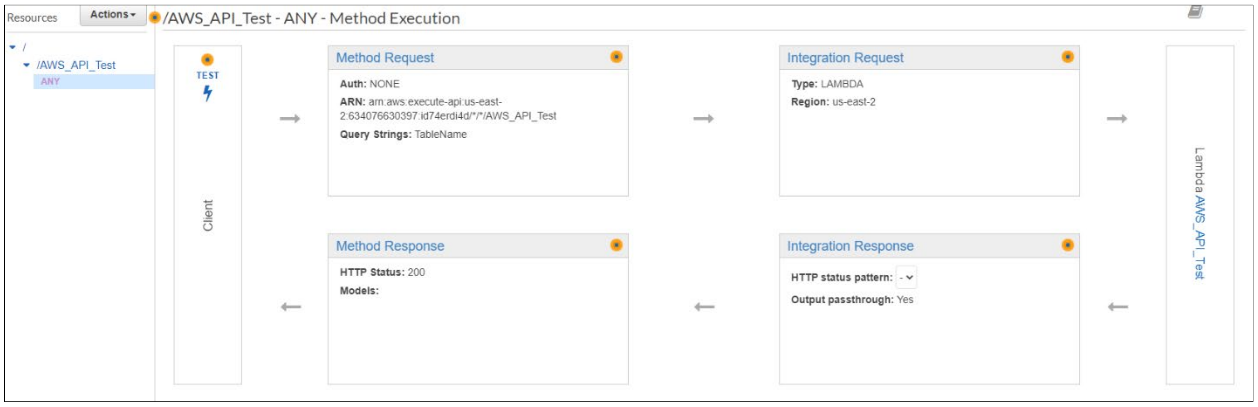
An item consists of one or more attributes. Each attribute consists of a name, a data type, and a value. When you read or write an item, the only attributes that are required are those that make up the primary key. More info

Figure 92: DynamoDB Table.

The test AWS API AWS_API_Test was created as shown in Figure 93. This API was set to no require any antiunification or authorization to allow for test of just the database interface. The following setting were defined.

Auth: NONE
ARN: arn:aws:execute-api:us-east-2:634076630397:id74erdi4d/*/*/AWS_API_Test
Query Strings: TableName
Type: LAMBDA
Region: us-east-2
HTTP Status: 200
Output passthrough: Yes

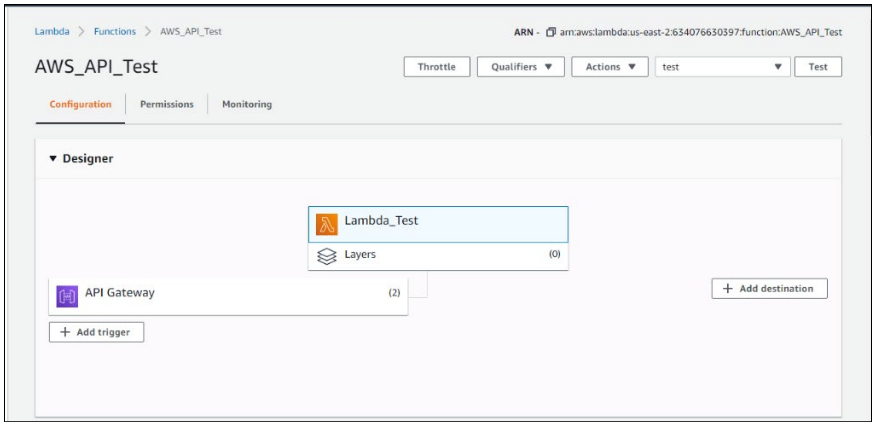
The information received from AWS API is then sent to a Lambda function.



Auth: NONE
ARN: arn:aws:execute-api:us-east-2:634076630397:id74erdi4d/*/*/AWS_API_Test
Query Strings: TableName
Type: LAMBDA
Region: us-east-2
HTTP Status: 200
Models:
HTTP status pattern: -
Output passthrough: Yes

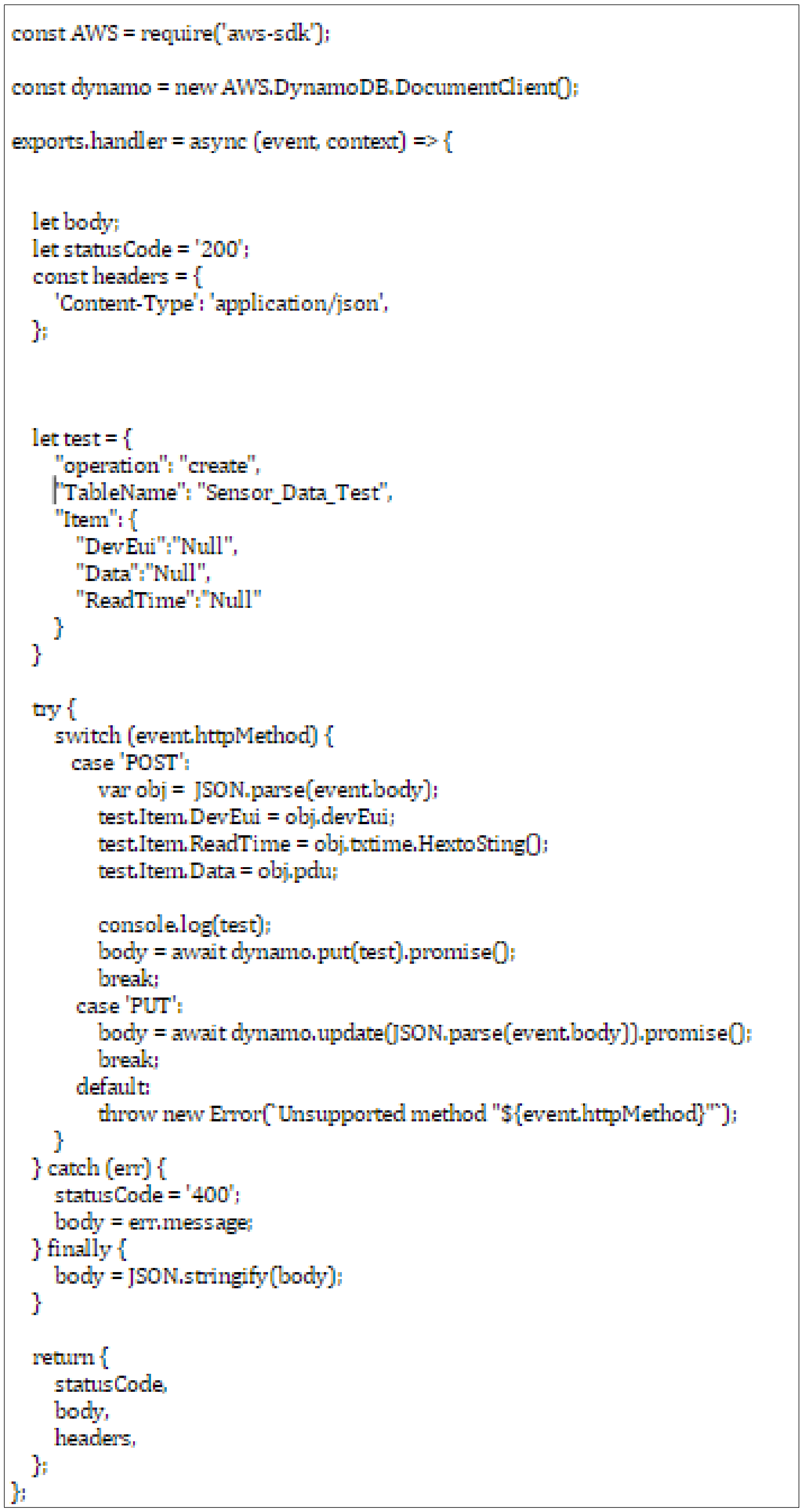
Figure 93: AWS API Gateway.

The Test Lambda Function AWS_API_Test was also created and set as the target location of the AWS API AWS_API_Test. The test lambda function is shown in Figure 94. The code for this test function is also depicted in Figure 95.



ARN - arn:aws:lambda:us-east-2:634076630397:function:AWS_API_Test

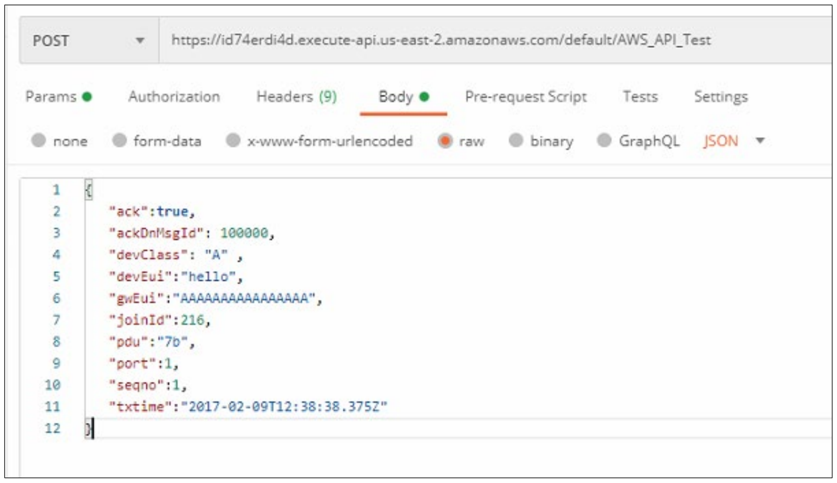
Figure 94: Lambda Function Design Flow.



```
const AWS = require('aws-sdk');

const dynamo = new AWS.DynamoDB.DocumentClient();

exports.handler = async (event, context) => {


  let body;
  let statusCode = '200';
  const headers = {
    'Content-Type': 'application/json',
  };



  let test = {
    "operation": "create",
    "TableName": "Sensor_Data_Test",
    "Item": {
      "DevEui":"Null",
      "Data":"Null",
      "ReadTime":"Null"
    }
  }

  try {
    switch (event.httpMethod) {
      case 'POST':
        var obj =  JSON.parse(event.body);
        test.Item.DevEui = obj.devEui;
        test.Item.ReadTime = obj.txtime.HextoSting();
        test.Item.Data = obj.pdu;

        console.log(test);
        body = await dynamo.put(test).promise();
        break;
      case 'PUT':
        body = await dynamo.update(JSON.parse(event.body)).promise();
        break;
      default:
        throw new Error(`Unsupported method "${event.httpMethod}"`);
    }
  } catch (err) {
    statusCode = '400';
    body = err.message;
  } finally {
    body = JSON.stringify(body);
  }

  return {
    statusCode,
    body,
    headers,
  };
};
```

Figure 95: Lambda Function Pseudo Code.

Once the database table, AWS API, and Lambda function is created the concept can be tested by making Postman API calls to the AWS API to create an item entry. An example of this API call is shown in Figure 96.

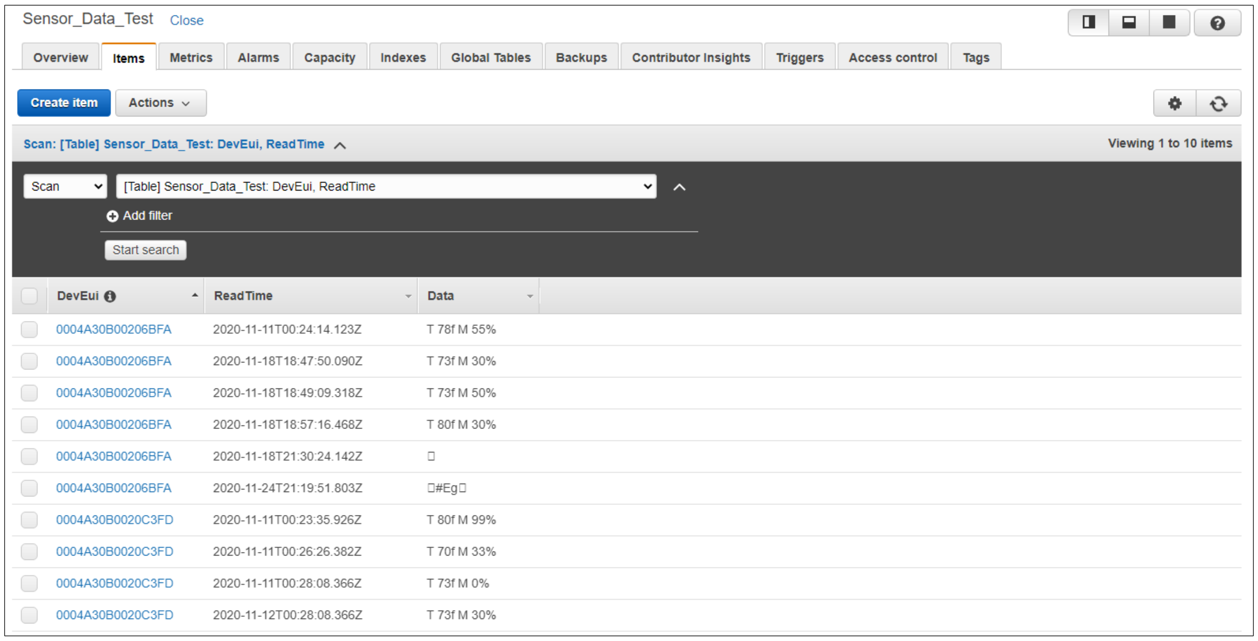


https://id74erdi4d.execute-api.us-east-2.amazonaws.com/default/AWS_API_Test

```
{
    "ack":true,
    "ackDnMsgId": 100000,
    "devClass": "A" ,
    "devEui":"hello",
    "gwEui":"AAAAAAAAAAAAAAAA",
    "joinId":216,
    "pdu":"7b",
    "port":1,
    "seqno":1,
    "txtime":"2017-02-09T12:38:38.375Z"
}
```

Figure 96: Postman API Call.

A handful of POST calls were tested to the AWS API similar to the one above. The results of these call are shown in the populated database table in Figure 97.



Scan: [Table] Sensor_Data_Test: DevEui, ReadTime

Viewing 1 to 10 items

[Table] Sensor_Data_Test: DevEui, ReadTime

DevEui	ReadTime	Data
0004A30B00206BFA	2020-11-11T00:24:14.123Z	T 78f M 55%
0004A30B00206BFA	2020-11-18T18:47:50.090Z	T 73f M 30%
0004A30B00206BFA	2020-11-18T18:49:09.318Z	T 73f M 50%
0004A30B00206BFA	2020-11-18T18:57:16.468Z	T 80f M 30%
0004A30B00206BFA	2020-11-18T21:30:24.142Z	□
0004A30B00206BFA	2020-11-24T21:19:51.803Z	□#Eg□
0004A30B0020C3FD	2020-11-11T00:23:35.926Z	T 80f M 99%
0004A30B0020C3FD	2020-11-11T00:26:26.382Z	T 70f M 33%
0004A30B0020C3FD	2020-11-11T00:28:08.366Z	T 73f M 0%
0004A30B0020C3FD	2020-11-12T00:28:08.366Z	T 73f M 30%

Figure 97: DynamoDB Table.

5.4 Prototypes: Implementation

Once the Design Verification Phase was complete, PCBs were ordered, and the software was integrated with the finished hardware prototypes for complete system implementation.

5.4.1 Power Management RK

Since the Sensor Pod is designed to stay buried beneath the ground for an entire growing season, reducing the energy required of the Pod is one of the most important concerns of the project. The original calculations for power requirements suggested that two batteries would be needed for a total of 5600mA-H, but it is important to know the actual power draw in real-world use. Reducing the energy pulled from the battery required looking at all components of design, including the power draw of the program, the voltage regulator chosen (if used), as well as what type of current each component will draw in active and sleep modes. In this section, the Sensor Pod's energy requirements and results from a real-world application of a battery test will be shared.

5.4.1.1 *Voltage Regulator*

After further review, it was found that all components in the Sensor Pod circuit had a 3V-3.7V threshold. The battery chosen is 100% full at 3.7V but can be charged up to 4.2V. Initially when designing the circuit, a voltage regulator was considered so that it could regulate the upper-end voltage down to 3.3V and the lower-end voltage up to 3.3V. The lithium-ion battery chosen may charge to 4.2V, but because of the chemical reactions that occur internally, the battery will never supply more than 3.7V to the circuit. When the lithium-ion battery is depleted, it's voltage will be around 3.2V. This means the battery should never deplete past the 3.2V threshold. Because of these datasheet findings, it was determined that a voltage regulator was unnecessary for Sensor Pod application. All power monitoring for the system can be accomplished through the PIC24FJ256GB410 microcontroller along with the RN2903 LoRa module.

5.4.1.2 *Battery Testing*

Once the final PCB design and program were completed, a proper analysis of battery consumption and current draw could be performed. The Fluke Handheld multimeters provided in the Lab did not have low enough current ranges to analyze the Sensor Pods. Instead, a Keithley Digital Multimeter with a 6-and-a-half-digit resolution of accuracy was used. This device can acquire current measurements down to 10pA, which is necessary since the PIC uses around 650nA in sleep mode. The test setup can be seen picture below.

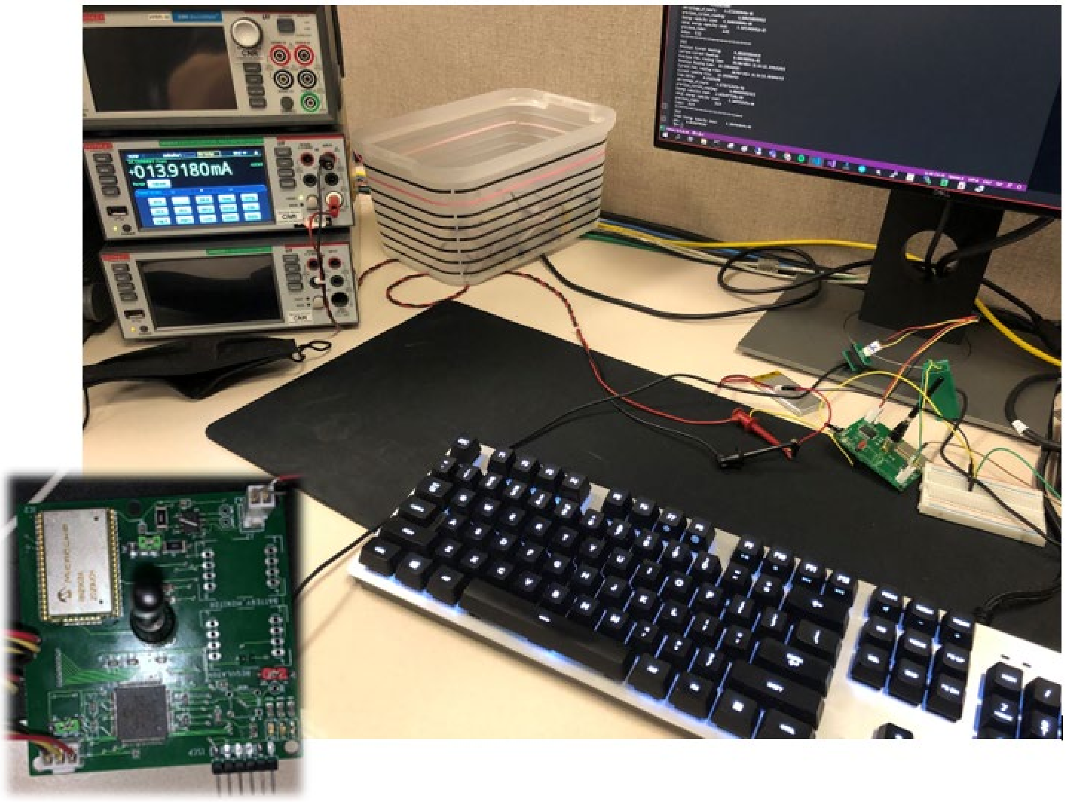
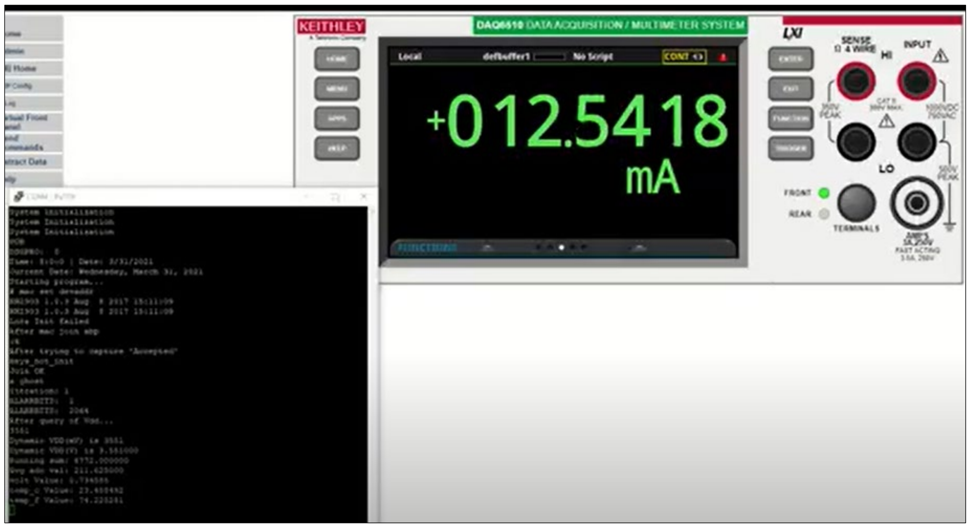


Figure 98: Battery Testing Setup (Sponsored by Keithley).

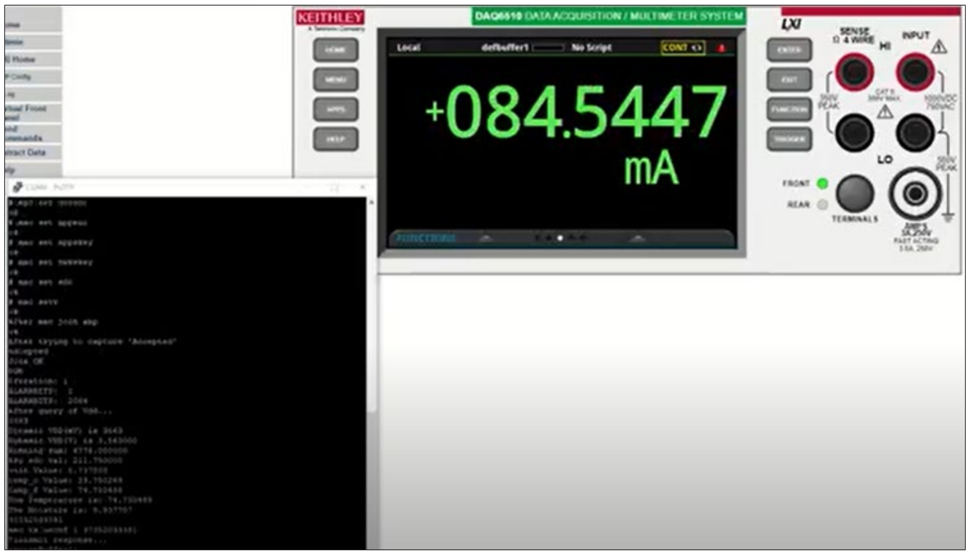
The Sensor Pod is connected in series to the Digital Multimeter, setup as an Ammeter, along with a single 2800mA-H battery. The test involved letting the test program run, which had sensors connected, as well as the antenna for transmission to simulate real conditions. The special test program was designed to connect to the LoRa network, acquire sensor measurements, transmit them, and go to sleep. The RN2903 LoRa transceiver sleep mode was activated, along with the PIC's deep sleep mode. The program executed this process every 10 seconds so that many sleep mode cycles could be observed on the Multimeter. Figures 99-101 show output from the Pod captured with a UART to USB adapter, contrasted with the Virtual Front Panel of the Multimeter.



```
System Initialization
System Initialization
System Initialization
POR
DSGPR0:  0
Time: 0:0:0 | Date: 3/31/2021
Current Date: Wednesday, March 31, 2021
Starting program...
# mac set devaddr
RN2903 1.0.3 Aug  8 2017 15:11:09
RN2903 1.0.3 Aug  8 2017 15:11:09
Lora Init failed
After mac join abp
ok
After trying to capture "Accepted"
keys_not_init
Join OK
a ghost
Iteration: 1
ALARMBITS:  1
ALARMBITS:  2064
After query of Vdd...
3561
Dynamic VDD(mV) is 3561
Dynamic VDD(V) is 3.561000
Running sum: 6772.000000
Avg adc val: 211.625000
Volt Value: 0.734586
temp_c Value: 23.458492
temp_f Value: 74.225281
```

Figure 99: PIC24FJ256GB410 Active Mode.

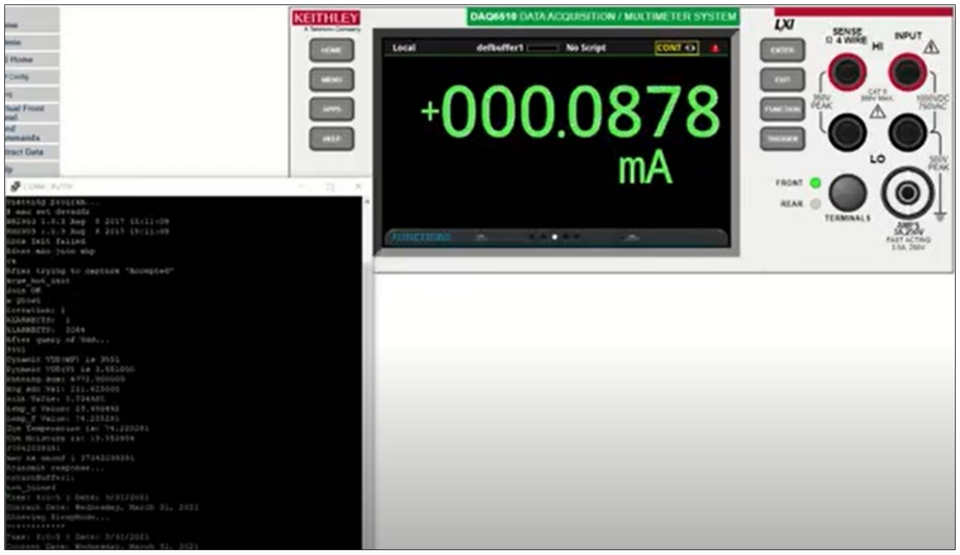
In the above figure, the Ammeter is reading ~12mA of current draw while taking current measurements in the active mode of the program which is taking sensor measurements and initializing the LoRa subsystem.



```
[illegible]
```

Figure 100: PIC24FJ256GB410 Active Transmit Mode.

In Figure 100, the screenshot was acquired in the very short duration of the transmit process in which the LoRa transceiver uses the antenna to transmit to the Gateway. This point in the program is the most power-intensive and is the part that should be kept to a minimum to preserve battery life.



```
[illegible]
```

Figure 101: PIC24FJ256GB410 Sleep Mode.

After taking measurements and transmitting, the circuit pulls an instantaneous current draw of about 88uA which is the lowest sleep state that the Sensor Pod will be in.

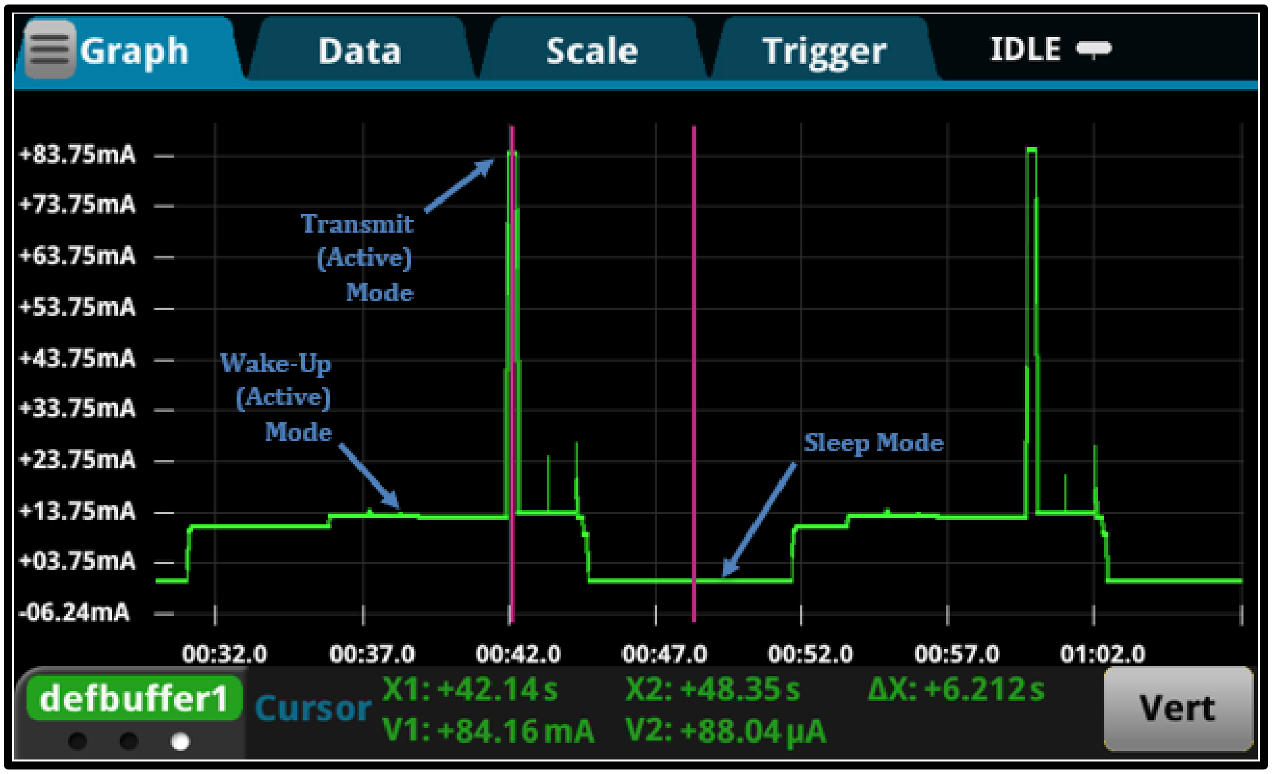


Figure 102: Current Draw of Sensor Pod in Different PIC Modes.

Since sleep mode in the test program was designed to only last 10 seconds, the multimeter could capture multiple cycles in a relatively short amount of time. In Figure 102, two cycles were captured in which the Sensor Pod, including the PIC and LoRa transceiver, were awakened from the sleep state to take measurements, transmit, and go back to sleep.

After capturing a few cycles, the buffer data could be extracted for further analysis. A program was created to analyze one period of sleep and one period in the active state. To determine the amount of charge depleted from the battery in the mentioned periods, the amount of time between each reading can be multiplied by the current reading. The equation is as follows.

$$Charge\ Depleted\ in\ One\ Wakeup\ Cycle\ (mAh) = 1000 * \sum_{n=wakeup}^{sleep} \left(\frac{time\ (s)}{60*60}\right) Current\ Measurements\ (A) \qquad (33)$$

After integrating through each period, the charge depleted from the battery could be calculated and extrapolated to determine what the battery consumption would be like during a full growing season. For a 153-day growing season, and transmitting 3 times a day, the Sensor Pod would draw 26.19mA-H in its active state, and 323.14mA-H in its lowest power state. All-in-all, that would be about 347.33mA-H, or depleted charge from the 2800mA-H battery used. This calculation satisfied the engineering requirement that the Pod would be a low power device requiring less than 5600mA-H.

To meet the engineering and marketing requirements for that the Sensor Pod be low power, code was added to the main program to trigger certain sleep modes and processes that would reduce power consumption.

```
while(1) {
    reset_analysis();
    ms_delay(100);
    reset_analysis();
    ms_delay(100);
    reset_analysis();
    DSCONbits.RELEASE = 0;

    // LED Check
    LED_0_SetHigh();
    ms_delay(300);
    LED_1_SetHigh();
    ms_delay(300);
    LED_2_SetHigh();

    char buf[255];
    sprintf(buf, "Iteration: %d ", iteration++);
    iteration++;
    Console_Write(buf);

    sprintf(buf,"ALARRBITS:  %d", RTCCON1Hbits.ALMRPT);
    Console_Write(buf);
    sprintf(buf,"ALARRBITS:  %d", ALMTIMEH);
    Console_Write(buf);


    // Get VDD
    char vddBuffer[4] = "";
    RN2903_Query_Command("sys get vdd", vddBuffer, 0);
    Console_Write("After query of Vdd...");
    Console_Write(vddBuffer);

    char newBuf[10];
    int vdd = atoi(vddBuffer);
    sprintf(newBuf, "Dynamic VDD(mV) is %d", vdd);
    Console_Write(newBuf);
    float vddVolts = vdd / 1000.00;
    sprintf(newBuf, "Dynamic VDD(V) is %f", vddVolts);
    Console_Write(newBuf);

    t = Read_ADC(vddVolts);
    m = Read_Moisture_Sensor();

    sprintf(message, "The Temperature is: %f", t);
    Console_Write(message);
    sprintf(message, "The Moisture is: %f", m);
    Console_Write(message);

    encodeMessage(message, vddVolts, t, m);
    Console_Write(message);

    int portNumber = 1;
    transmit("uncnf", portNumber, message);
    GetTime();

    Console_Write("Entering SleepMode...");
    Console_Write("************");
    LED_0_SetLow();
    LED_1_SetLow();
    LED_2_SetLow();

    RN2903_Write("sys sleep 9000");
    GetTime();
    ms_delay(50);
    EnterDeepSleep();
  }
```

The code above is the main part of the program that executes just after connecting to the LoRa network. Variables for data are allocated, and the function calls which operate on them are called near the end. At the very end, the sys sleep 9000 command is sent to the LoRa transceiver, and the EnterDeepSleep() function is called which puts the PIC in deep sleep until scheduled to wake up.

The following is a sample of code which enables power savings for the sensor Pod. After executing assembly instructions that directly interact with the sleep mode registers, the Sleep() system function can be called to finally put the PIC in deep sleep mode.

```
void EnterDeepSleep() {
 ms_delay(100);
 asm volatile("MOV #0x8000, W2"); // sequence to set the DSEN bit, must do it twice
 asm volatile("MOV W2, DSCON");
 asm volatile("MOV W2, DSCON");
 Sleep();
}
```

Additionally, every time the Sensor Pod wakes up, an analysis is run to determin why it woke up. This is important to know to make sure that the Pod is entering its deep sleep state of operation and reducing energy consumption. Below is code that determines the reason for wakeup.

```
void reset_analysis() {

    // ------------------------ Reset Analysis -------------------------------
    //Status from reset (1:POR, 2:Sleep, 3:DeepSleep, 4:Watchdog, 5:Ext, 6:SwReset)
    //Device leading to reset (1:INT0, 2:Watchdog, 3:RTCC, 4:MCLR, 5:POR)
    int reset_status;
    int reset_device;

    if(RCONbits.DPSLP == 1){

        char buf[255] = "";
        RN2903_Read(buf);

        reset_status=3;
        RCONbits.DPSLP = 0;
        DSCONbits.RELEASE = 0;
        if(DSWAKEbits.DSWDT) reset_device=2;
        else if(DSWAKEbits.DSINT0) reset_device=1;
        else if(DSWAKEbits.DSMCLR) reset_device=4;
        else if(DSWAKEbits.DSRTCC) reset_device=7;
        else reset_device=0;
    }
    else if(RCONbits.SLEEP){
        reset_status=2;
        RCONbits.SLEEP = 0;
        DSCONbits.RELEASE = 0;
    }
    else if(RCONbits.WDTO){
        reset_status=4;
        RCONbits.WDTO = 0;
        DSCONbits.RELEASE = 0;
    }
    else if(RCONbits.EXTR){
        reset_status=5;
        RCONbits.EXTR = 0;
        DSCONbits.RELEASE = 0;
    }
    else if(RCONbits.SWR){
        reset_status=6;
        RCONbits.SWR = 0;
        DSCONbits.RELEASE = 0;
    }
    else if(RCONbits.POR){
        reset_status=1;
        RCONbits.POR = 0;
        DSGPR0 = 0;
        DSGPR1 = 0;
    }
    else{ //Unknown state
        reset_status=0;
        DSGPR0 = 0;
        DSGPR1 = 0;
    }

    ms_delay(500);

    //Status from reset (1:POR, 2:Sleep, 3:DeepSleep, 4:Watchdog, 5:Ext, 6:SwReset, 7:RTCC)
    switch (reset_status){
        case 7:
            Console_Write("RTCC Alarm");
        case 6:
            Console_Write("SwReset");
            break;
        case 5:
            Console_Write("HwReset");
            break;
        case 4:
            Console_Write("Watchdog");
            break;
        case 3:
            Console_Write("DeepSleep");
            break;
        case 2:
            Console_Write("Sleep");
            break;
        case 1:
            Console_Write("POR");
            break;
        default:
            Console_Write("a ghost");
    }
}
```

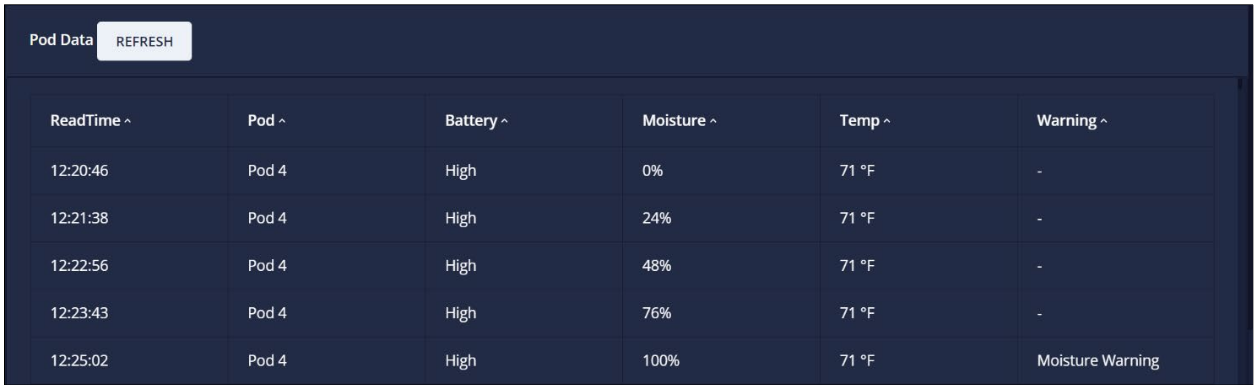
5.4.2 Soil Moisture Sensor AA, RK, AW

Reading soil moisture required the use of a separate PCB and connector. A capacitive sensor is used, which is fed into a dedicated circuit on the PCB. At the PIC, the input capture port is used to read frequency from the dedicated circuit which can be used to infer what type of moisture the sensor is being subjected to. To get this method of input capture working, a high and low moisture reading had to be taken to set thresholds so that a spectrum of soil moistures that the program will detect could be created. Below is the code that is responsible for capturing frequency, which is essentially reading rising and falling edges on an input square wave.

```
double Read_Moisture_Sensor() {
    float TimerFreq = 250000.0;
    bool bufferStatus;
    uint16_t data1,data2;
    char buf[255] = "";
    float avg = 0;
    int i = 0;
    int nb_samples = 15;
    for (i = 0; i < nb_samples; i++) {
        IC5_Start();
        bufferStatus = IC5_IsCaptureBufferEmpty();
        if(!bufferStatus) {
            data1 = IC5_CaptureDataRead();
            data2 = IC5_CaptureDataRead();
        }

        uint16_t period;
        double freq;
        period = data2 - data1; // determine signal period
        freq = TimerFreq / period; // calculate frequency
        ms_delay(200);
        avg = avg + freq;
    }
    IC5_Stop();
    avg = avg / nb_samples;
    return (avg);
}
```

Essentially, the function above samples the period between two rising edges. The frequency can be determined from the inverse of the period, and the calculation is then averaged over 15 readings to provide and accurate estimate of the frequency. The value returned to the main program is the frequency, but this value is converted into a normalized value, from 0 100, when sent to the database/Web Server.

Two tests were performed on the soil moisture sensor to verify the circuit was working properly. The first test was purely monitoring the sensor as moisture levels of the soil increased. A soil moisture sensor was placed in a container with 10 oz. dry dirt. Like the prototype test, water was added in 2.5 oz increments from 0 oz. (dry) to 10 oz. (saturated). The data obtained from the experiment is displayed on the webpage in Figure 103.



ReadTime	Pod	Battery	Moisture	Temp	Warning
12:20:46	Pod 4	High	0%	71 °F	-
12:21:38	Pod 4	High	24%	71 °F	-
12:22:56	Pod 4	High	48%	71 °F	-
12:23:43	Pod 4	High	76%	71 °F	-
12:25:02	Pod 4	High	100%	71 °F	Moisture Warning

Figure 103: Soil Moisture Sensor Readings.

The sensor data was exported into an excel spreadsheet and trended, as seen in Figure 104.

ReadTime	Moisture Percentage (%)	Water Percentage (%)
12:20:46	0	0
12:21:38	24	25
12:22:56	48	50
12:23:43	76	75
12:25:02	100	100

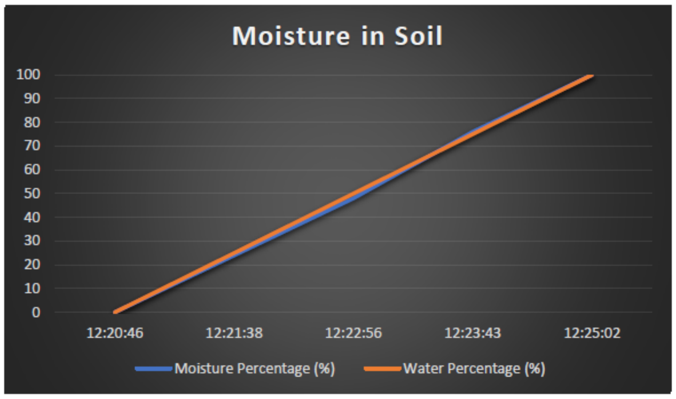
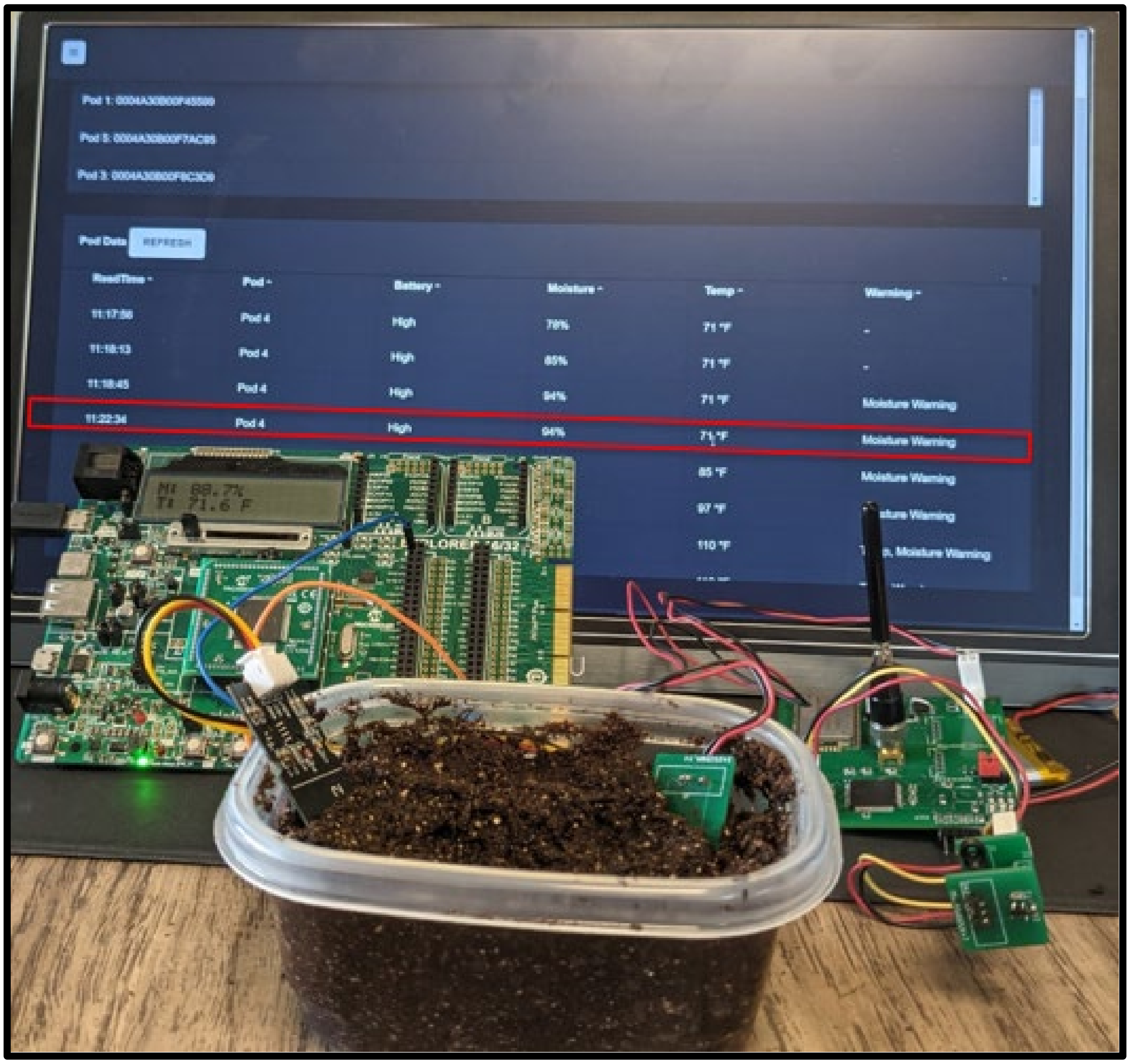


Figure 104: Trended Soil Moisture Sensor Data.

As the moisture levels in the soil increased, the percentage populated from the senor increased proportionally, proving that the sensor was functional.

To further verify the moisture sensor was accurate, the PCB sensor was placed in soil with a 97% accurate off-the-shelf sensor from AdaFruit. The PCB sensor was connected to the Sensor Pod, transmitting data to the gateway to be displayed on the Soil Sensor Network Application, and the AdaFruit sensor was connected to a development board with an LCD.



ReadTime	Pod	Battery	Moisture	Temp	Warning
11:17:56	Pod 4	High	79%	71 °F	-
11:18:13	Pod 4	High	85%	71 °F	-
11:18:45	Pod 4	High	94%	71 °F	Moisture Warning
11:22:34	Pod 4	High	94%	71 °F	Moisture Warning
				85 °F	Moisture Warning
				97 °F	[illegible]
				110 °F	[illegible] Moisture Warning

Figure 105: PCB vs. Adafruit Soil Moisture Sensor.

A picture of the results was taken once the moisture sensor stabilized. Sensor Pod 4 stabilized at a 94% moisture reading. The Adafruit moisture sensor stabilized at a 88.7% soil moisture reading. According to the datasheet, the Adafruit sensor is 97% accurate. Assuming worst-case scenario, the Adafruit sensor reading at 97%, the accuracy of the PCB soil moisture sensor can be calculated using the following equation.

$$PCB\ accuracy = 0.97 - \frac{94-88.7}{94} = 0.91 = 91\% \tag{34}$$

The soil moisture PCB was at least 91% accurate, taking into consideration worst-case scenario. This fulfilled the Engineering Requirement that the soil moisture sensor must read with at least an 80% accuracy.

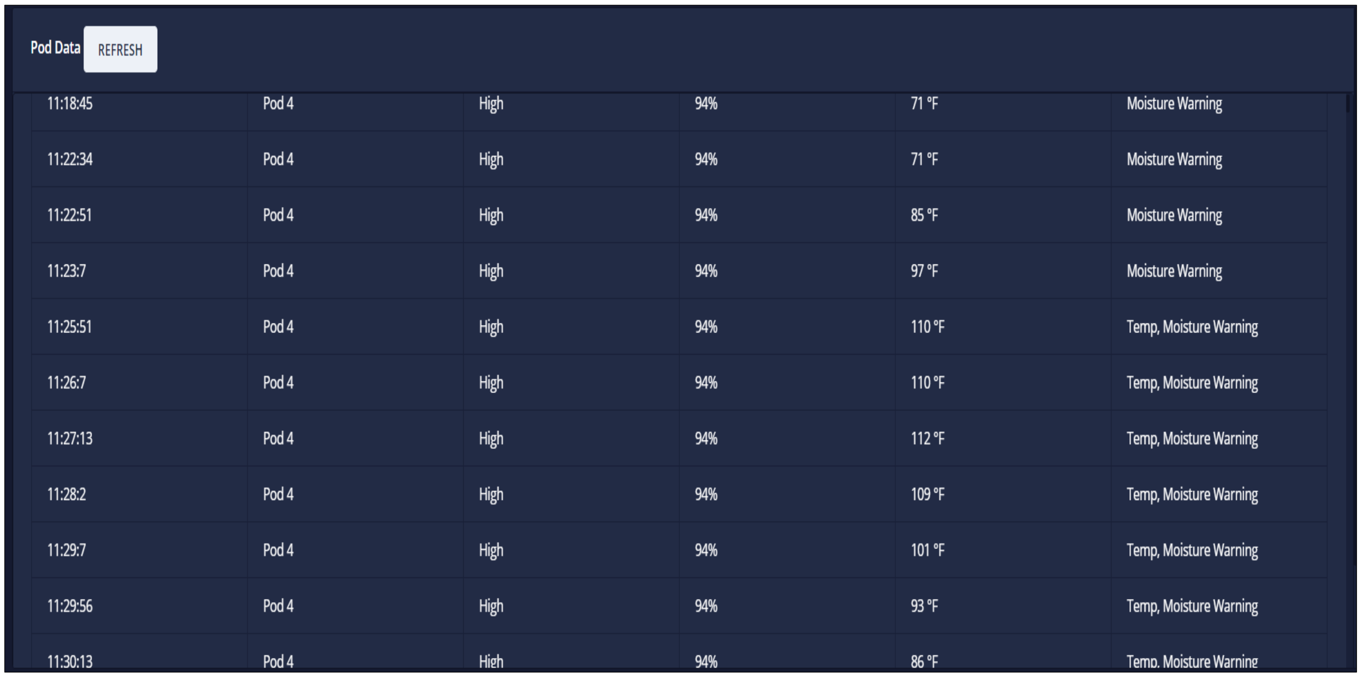
5.4.3 Temperature Sensor RK, AW

The temperature sensor in its final design was attached to the side of the Sensor Pod housing, and successfully responded to ambient temperatures in air and soil. To reach a complete design, a dedicated PCB had to first be created that would integrate with the MAX6607 temperature sensor. The magnetic quick disconnect component was then implemented to allow it to connect to the pod. From the connector to the pod was a 3-wire connection for VDD, ground, and the analog voltage line. This connection led to the main PCB that would be fed into an analog voltage input port of the PIC where the program could convert the given voltage to Fahrenheit degrees. Below is the part of code that handles the analog-to-digital conversion.

```
double Read_ADC(float vdd) {
    float quant_res = vdd / 1023.0;
    double ADJ = 0.0; // adjustment due to voltage drop

    int digital_value = 0;
    double running_sum = 0.0;
    double average_adc_val = 0.0;
    ADC1_CHANNEL channel = channel_AN15;

    int nb_samples = 32;
    int i = 0;
    for (i = 0; i < nb_samples; i++) {
        ADC1_Enable();
        ADC1_ChannelSelect(channel);
        ADC1_SoftwareTriggerEnable();
        ms_delay(5);
        ADC1_SoftwareTriggerDisable();
        while (!ADC1_IsConversionComplete(channel));
        digital_value = ADC1_ConversionResultGet(channel);
        running_sum = running_sum + digital_value;
        ADC1_Disable();
    }

    average_adc_val = running_sum / nb_samples; // divide to get the average
    double volt = average_adc_val * quant_res;
    double temp_c = (volt - .5 + ADJ) * 100;
    double temp_f = (temp_c * 1.8) + 32;

    return(temp_f);
} // ReadADC
```

The temperature is obtained by sampling the analog voltage input port 32 times and taking an average. This sampling also considers the supply voltage of the temperature sensor by querying the voltage of the RN2903 since the quantization depends on the supply voltage. This is what the VDD parameter is for in the Read_ADC function. Before the final temperature is returned to the main program, it is converted from the quantized value to a Fahrenheit value, which is placed in the message sent to the LoRa server. This information is then displayed on the website for the farmer to monitor.

The Sensor Pod was powered on and set aside until the temperature stabilized. Heat was applied to the temperature sensor and the data was collected. The final temperature measurements were transmitted and stored in the database to be displayed in the Web Application, which displayed a trend of temperature readings over time.


Pod Data REFRESH

11:18:45	Pod 4	High	94%	71 °F	Moisture Warning
11:22:34	Pod 4	High	94%	71 °F	Moisture Warning
11:22:51	Pod 4	High	94%	85 °F	Moisture Warning
11:23:7	Pod 4	High	94%	97 °F	Moisture Warning
11:25:51	Pod 4	High	94%	110 °F	Temp, Moisture Warning
11:26:7	Pod 4	High	94%	110 °F	Temp, Moisture Warning
11:27:13	Pod 4	High	94%	112 °F	Temp, Moisture Warning
11:28:2	Pod 4	High	94%	109 °F	Temp, Moisture Warning
11:29:7	Pod 4	High	94%	101 °F	Temp, Moisture Warning
11:29:56	Pod 4	High	94%	93 °F	Temp, Moisture Warning
11:30:13	Pod 4	High	94%	86 °F	Temp, Moisture Warning

Figure 106: Temperature Sensor Readings.

As heat was applied, the temperature slowly increased, and a temperature warning appeared on the website to inform the farmer that excessive heat conditions (greater than 100ºF) have occurred. The other excessive conditions monitored were soil moisture and battery life. These warnings satisfied the Engineering Requirement that an application will alert the farmer if excessive soil conditions occur (i.e.: if the soil is exceptionally dry) so immediate action can be taken. Once heat was no longer applied, the values of the temperature sensor began to decrease to the initial point of stability.

To verify the temperature sensor was working properly, an off-the-shelf temperature sensor was connected to the explorer board and heat was applied to both the PCB and store-bought sensor. The results can be seen in Figure 105 of the Soil Moisture Sensor Setup previously explained. When the temperature sensor was at the stabilized value, the PCB temperature sensor read 71ºF and the off-the-shelf read 71.6ºF.

5.4.4 Embedded Firmware RK

After prototyping each subsystem such as reading temperature, enabling sleep mode, and communicating with the LoRa transceiver, it was time to integrate it all for a final revision that could be flashed to all of the Sensor Pods. Microchip's dedicated PIC programming IDE, MPLAB X, helped provide the tools that enable features on the PIC. For example, the Real Time Clock Calendar that is used to wake up the Pod in deep sleep can be configured within the editor and will program the registers with proper values. Each Pod ran very similar programs, which helped make development easier because the only parts that were unique were the LoRa transceivers on each Pod.

As an overview, the Sensor Pod upon being first power on would initialize all of the communication and interfacing components, and then connect to the pre-configured LoRa network. Below is the code that is the beginning of the program which calls functions like SYSTEM_Initialize() to setup the peripherals.

```
SYSTEM_Initialize();
int val = DSGPR0;
Console_Write("System Initialization");
DSCONbits.RELEASE = 0;

ms_delay(100);
reset_analysis();

char buf[255];
sprintf(buf,"DSGPR0:  %d", DSGPR0);
Console_Write(buf);
DSGPR0 = val + 1;

GetTime();
ms_delay(50);

// LED Check
ms_delay(300);
LED_1_SetHigh();
ms_delay(300);
LED_2_SetHigh();
Console_Write("Starting program...");
```

```
ms_delay(100);

ms_delay(100);
GetParams();
```

Once connected, the Pod is ready to take measurements, encode the data, and prepare it to send to the Gateway so that the farmer could see it on the Web Application. Once these events were completed, the Pod goes into deep sleep for many hours, as determined by the RTCC, and waits until it needs to wake up and take soil measurements in the day.

5.4.4.1 *Microcontroller Data Collection* ***AA, RK***

The microcontroller and sensors combine to form the Sensor Pod that serves to collect moisture and temperature readings of the soil. As mentioned in § 5.4.2 and § 5.4.3, each sensor is associated with its own specialized code to read frequency and analog voltages. In the main program, the two functions ReadADC() and Read_Moisture_Sensor() were called. Both return a value which the program can use later to encode and send the message to the gateway via the transceiver. Below is a snippet of code which collects the readings from both sensors, and outputs to the debugging UART connection on the PCB.

```
    t = Read_ADC(vddVolts);
    m = Read_Moisture_Sensor();

    sprintf(message, "The Temperature is: %f", t);
    Console_Write(message);
    sprintf(message, "The Moisture is: %f", m);
    Console_Write(message);
```

Creating the data collecting function streamlines the main program and made maintaining and debugging the project much easier.

5.4.4.2 *Support Functions* ***RK***

There were many parts of the Sensor Pod firmware which helped enable acquiring sensor data, performing sleep mode analysis, debugging the LoRa transceiver, and completing other various functions. One of the most useful functions of the programming was the implementation of the UART ports. One UART port was used for debug purposes, and one was used to send commands to the LoRa transceiver.

The Console_Write() function is used to generate a UART signal which can be read by a UART to USB device, which was used heavily in development and testing of the program. The code can be seen as follows.

```
void Console_Write(char *text) {
    int i;
    for (i = 0; text[i] != '\0'; i++) {
        UART2_Write(text[i]);
    }
    UART2_Write(0x0D); // Carriage Return
    UART2_Write(0x0A); // Line feed
}
```

Messaages are written character by character until the null terminator is reached. At the end of the message, whether it be transmission from the PIC or RN2903 module, the message is always ended with a carriage return and new line character to signify the end of the sequence.

The UART communication that is used to interface with the LoRa module was very similar to the UART debugger, but a read sequence was implemented to see statuses and responses from the LoRa module.

```
void RN2903_Write(char *text) {

    int i;
    for (i = 0; text[i] != '\0'; i++) {
        UART1_Write(text[i]);
    }
    UART1_Write(0x0D); // Carrage Return
    UART1_Write(0x0A); // Line feed
}

void RN2903_Read(char *output) {

    memset(output, 0, 255);
    int BUF_LEN = 255;
    unsigned int i = 0;

    while (1) {
        char ch;
        ch = UART1_Read();
        if (ch != '\n') { // ignore lf's
            if (ch != '\r') {
                if (i < BUF_LEN - 2) {
                    output[i++] = ch;
                    output[i] = '\0';
                }
            }
        } else {
            break;
        }
    }
}
```

The RN2903_Read() function reads what the module responds with after sending a command. This allows the program to verify that commands are being sent to the LoRa module wih correct syntax. For example, after a command is sent, the transceiver responds with an "ok" message when syntax is correct, and perhaps a value when GET command is issued.

Often used with the UART read and write commands, is a delay function. UART communications takes time to encode and put the bit sequence on the communication bus, which is why delays are needed. If a message is written, but right after the Sensor Pod goes to sleep and no delay is used, then the message will not be written. Below is the ms_delay() function that is specifically designed to work with our chosen PIC and it's timing.

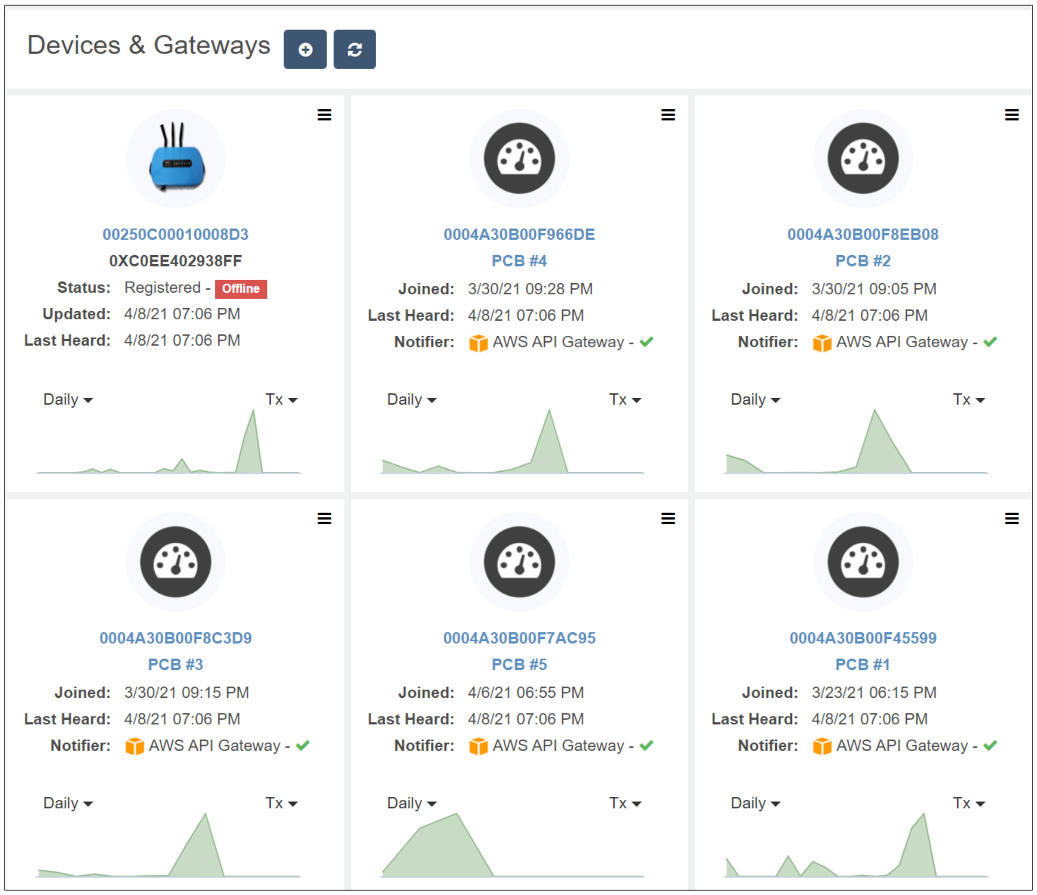
```
void ms_delay(int ms) {
    T2CON = 0x8030; // Timer 2 on, TCKPS<1,0> = 11 this 1:256
    TMR2 = 0;
    while (TMR2 < ms * 62.5);
}
```

5.4.5 Communication

Enabling robust and reliable communication links between the Sensor Pods was paramount when trying to satisfy range requirements. To enable the communication between the Gateway and Sensor Pod, the transceiver had to execute the correct sequence of commands, and the Senet Hub had to be configured to receive those messages. This section will provide detail as to how the Pods were connected to the network, and how Soil data was transmitted.

5.4.5.1 *LoRa Module Communication* *RK*

Each sensor pod was designed to support two-way communication to the Gateway, and Web Application. For each pod to be recognized, the unique pods were registered in the Senet Portal. Figure 107 shows each unique Sensor Pod, and Gateway all registered in the Senet Portal.



00250C00010008D3
0XC0EE402938FF
Status: Registered - Offline
Updated: 4/8/21 07:06 PM
Last Heard: 4/8/21 07:06 PM

0004A30B00F966DE
PCB #4
Joined: 3/30/21 09:28 PM
Last Heard: 4/8/21 07:06 PM
Notifier: AWS API Gateway - ✓

0004A30B00F8EB08
PCB #2
Joined: 3/30/21 09:05 PM
Last Heard: 4/8/21 07:06 PM
Notifier: AWS API Gateway - ✓

0004A30B00F8C3D9
PCB #3
Joined: 3/30/21 09:15 PM
Last Heard: 4/8/21 07:06 PM
Notifier: AWS API Gateway - ✓

0004A30B00F7AC95
PCB #5
Joined: 4/6/21 06:55 PM
Last Heard: 4/8/21 07:06 PM
Notifier: AWS API Gateway - ✓

0004A30B00F45599
PCB #1
Joined: 3/23/21 06:15 PM
Last Heard: 4/8/21 07:06 PM
Notifier: AWS API Gateway - ✓

Figure 107: Senet Portal Device EUI.

The Device EUI for each pod was also recorded in the Web Application to separate the data streams and visualize them correctly to the farmer.

Once the setup was complete, the main program had to execute the Initialization and join command, to join the LoRa network and begin transmitting data. The transmission power was kept at its default setting on the RN2903 transceiver of 20dBm to maximize the range and soil penetration of the signal. Below is the code that can be modified to program each of the 5 prototypes, with different Lora parameters.

```
 int targetBoard = 2;

  struct SensorPod {
    int debug;
    char deviceAddress[50];
    char deviceEUI[50];
    char appEUI[100];
    char appSKey[100];
    char nwkSKey[100];
  };

  switch(targetBoard) {

    case 1 :
      status = LORA_Initialize(1, "12026C69", "0004A30B00F45599", "00250C0000010001", "C76C781
A49558017E8D785702160CA1D", "9EA3BCBB73DFF869F320498F36C44320");
    break;

    case 2 :
      status = LORA_Initialize(1, "1202723C", "0004A30B00F8EB08", "00250C0000010001", "EBCDFF8
A57C65088BB466555AE23B12A", "53940E52788C64BC497D2D589A4DCAF2");
    break;

    case 3 :
      status = LORA_Initialize(1, "1202723E", "0004A30B00F8C3D9", "00250C0000010001", "204FE12
CA6E8664FAA63E3670EA872D9", "DC8D2C19DE2867AE9816780926E9E78F");
    break;

    case 4 :
      status = LORA_Initialize(1, "1202723F", "0004A30B00F966DE", "00250C0000010001", "E8667F8
5946627D0609206FED76ED793", "231DE0A84D17C2A24792E23C4A2872CC");
    break;

    default:
        Console_Write("No Pod Selected");
  }
```

```
if (status == 0) {
    Console_Write("Lora Init failed");
}

ms_delay(1000);
int connectStatus = ConnectABP();
```

The LORA_Initialize() function was designed to send commands with special parameters, to the RN2903 module.

```
int LORA_Initialize(int debug, char* device_address, char *device_eui, char *application_eui,
char *application_s_key, char *application_nwk_s_key) {

    char return_buf[255];
    char command[255];

    sprintf(command, "mac set devaddr %s", device_address);
    Console_Write("# mac set devaddr");
    RN2903_Query_Command(command, return_buf, 1);
    if (strcmp(return_buf, "ok") != 0) {
        Console_Write(return_buf);
        return 0;
    }

    command[0] = '\0';
    sprintf(command, "mac set deveui %s", device_eui);
    Console_Write("# mac set deveui");
    RN2903_Query_Command(command, return_buf, 1);
    if (strcmp(return_buf, "ok") != 0) {
        Console_Write(return_buf);
        return 0;
    }

    command[0] = '\0';
    sprintf(command, "mac set appeui %s", application_eui);
    Console_Write("# mac set appeui");
    RN2903_Query_Command(command, return_buf, 1);
    if (strcmp(return_buf, "ok") != 0) {
        Console_Write(return_buf);
        return 0;
    }

    command[0] = '\0';
    sprintf(command, "mac set appskey %s", application_s_key);
    Console_Write("# mac set appskey");
    RN2903_Query_Command(command, return_buf, 1);
    if (strcmp(return_buf, "ok") != 0) {
        Console_Write(return_buf);
        return 0;
    }
    command[0] = '\0';
    sprintf(command, "mac set nwkskey %s", application_nwk_s_key);
    Console_Write("# mac set nwkskey");
    RN2903_Query_Command(command, return_buf, 1);
    if (strcmp(return_buf, "ok") != 0) {
        Console_Write(return_buf);
        return 0;
    }

    Console_Write("# mac set adr");
    RN2903_Query_Command("mac set adr on", return_buf, 1);
    if (strcmp(return_buf, "ok") != 0) {
        Console_Write(return_buf);
        return 0;
    }
    Console_Write("# mac save");
    RN2903_Query_Command("mac set adr on", return_buf, 1);
    if (strcmp(return_buf, "ok") != 0) {
        Console_Write(return_buf);
        return 0;
    }

    return 1;
}
```

Each time the Pod wakes up from its sleep cycle, the above code was executed and, upon successful completion, would join the network and send data to the Gateway.

Transmitting data to the Gateway is made simple with a transmit command. The most complicated part was encoding the data to be short enough since LoRa has such a low data rate. During testing, the data rate is reduced even lower because the rate of transmission is so high. Soil Temperature, Soil Moisture Level, and Battery Status all had to be communicated in one transmission. Since the message had to be sent in hex, the first two parameters mentioned were directly converted to a string, space separated. The string value was then directly converted to hex, and was appended with a decimal 1, 2, or 3 to reduce the payload length.

```
void encodeMessage(char *fullMessage, float battery, double tempReading, double moistureReadin
g) {

    char payload[100];

    fullMessage[0] = '\0';
    payload[0] = '\0';
    int battery_status = -1;

    if (battery > 3400) {
        battery_status = 3;
    } else if (battery_status > 3200) {
        battery_status = 2;
    } else {
        battery_status = 1;
    }

    if (tempReading > 200) {
        tempReading = 120;
    }

    moistureReading = 99;

    sprintf(payload, "%2.0f %2.0f", tempReading, moistureReading);
    string2hexString(payload, fullMessage);
    // add battery status
    char snum[1];
    sprintf(snum, "%d", battery_status);
    strncat(fullMessage, snum, 1); // append 1 character to full message
}
```

In normal use, the payload should be large enough to accommodate a string that is twice as large as mentioned, but for testing it was necessary to make the modifications to allow for a much faster transmission rate. In normal use, the Pod would communicate 3 times a day, but during testing it would transmit 6 times a minute for quick feedback. The code below was implemented to encode the data into hex, which is the desired data format for transmission to the Gateway.

```
void string2hexString(char* input, char* output){
    int loop;
    int i;

    i=0;
    loop=0;

    while(input[loop] != '\0')
    {
        sprintf((char*)(output+i),"%02X", input[loop]);
        loop+=1;
        i+=2;
    }
    //insert NULL at the end of the output string
    output[i++] = '\0';
}
```

Once the data is properly encoded, the transmit function was used, which takes in the port number and encoded data as parameters to send to the Gateway.

```
void transmit(char* type, int port_num, char *data) {

    char returnBuffer1[255] = "";
    char returnBuffer2[255] = "";
    char command[255];

    if (strcmp(type, "uncnf") == 0){
        sprintf(command, "mac tx uncnf %d %s", port_num, data);
    }
    Console_Write(command);
    RN2903_Write(command);
    processResponse(returnBuffer1, returnBuffer2);
}
```

5.4.5.2 *Distance Testing* — ***AA, RK, AW***

One of the engineering requirements in need of verification was that the Sensor Pod had to successfully transmit signals up to 3.3 km. The Gateway and Sensor Pods were transported to Richville Drive in Massillon where there is an abundance of farmland. A 3.3km distance was mapped out where communication would be close to direct Line of Sight (LOS), testing as if the pod was buried in an actual field. The Senet Gateway was setup on the side of the road, as seen in Figure 108.

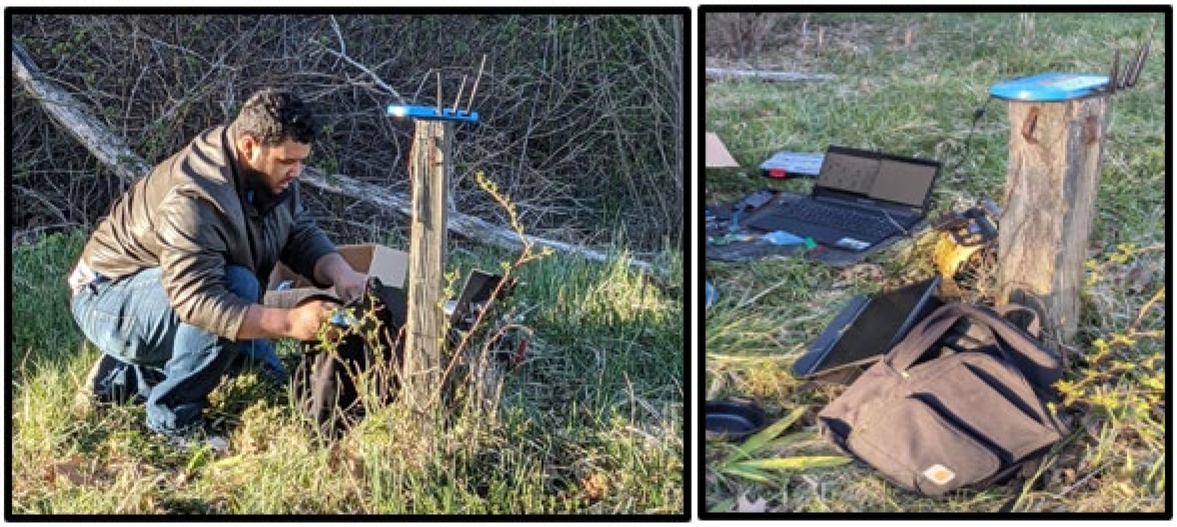


Figure 108: 3.3km Distance Testing Communication Setup.

The Sensor Pod was “planted” six inches into a bucket of dirt to verify that it could communicate through at least 3 inches of soil. Communication through 3 inches of soil was the Engineering requirement that needed to be fulfilled, but realistically for a root or tube application, the roots would be approximately 6 inches beneath the soil so it was planted 6 inches deep rather than 3 inches deep to verify the Soil Sensor Network would work for these applications.

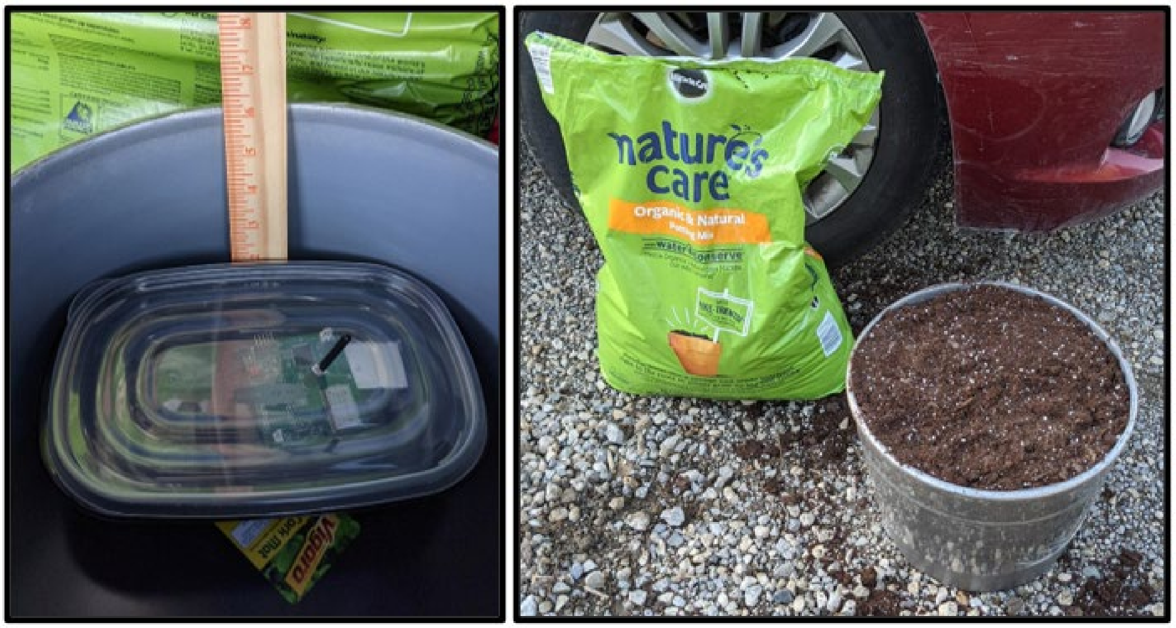
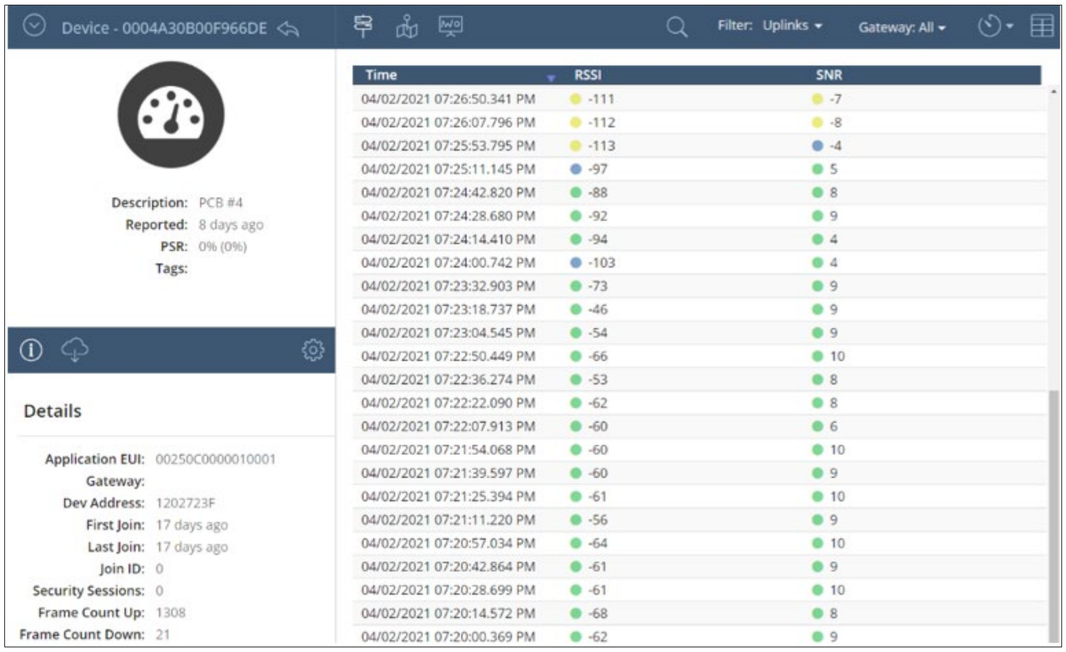


Figure 109: Distance Testing Sensor Pod Setup.

Before the Sensor Pod was buried, it was powered on and connected to the Senet Gateway. Once in the soil, the Senet server was observed to verify the Pod was still communicating to the Gateway at short distances. The data collected is shown in Figure 110. Received Signal Strength Indicator (RSSI) and the Signal to Noise Ratio (SNR) were monitored. The first few signals seen in the figure were fair signal strength because the Sensor Pod was too close to the Gateway. Antennas have a *blind cone* in which the distance is too short for the transmitter to send the signal to the receiver due to the geometry of the antenna’s radiation pattern [12]. The blind cone will not be a concern to farmers because it is highly unlikely that a pod will be planted directly next to the gateway in a practical application. In the case of a dropped packet, the Sensor Pod will attempt to retransmit the packet. When the antenna was moved away from the Gateway, both the RSSI and SNR were of very good signal strength.



Description:	PCB #4
Reported:	8 days ago
PSR:	0% (0%)
Tags:	

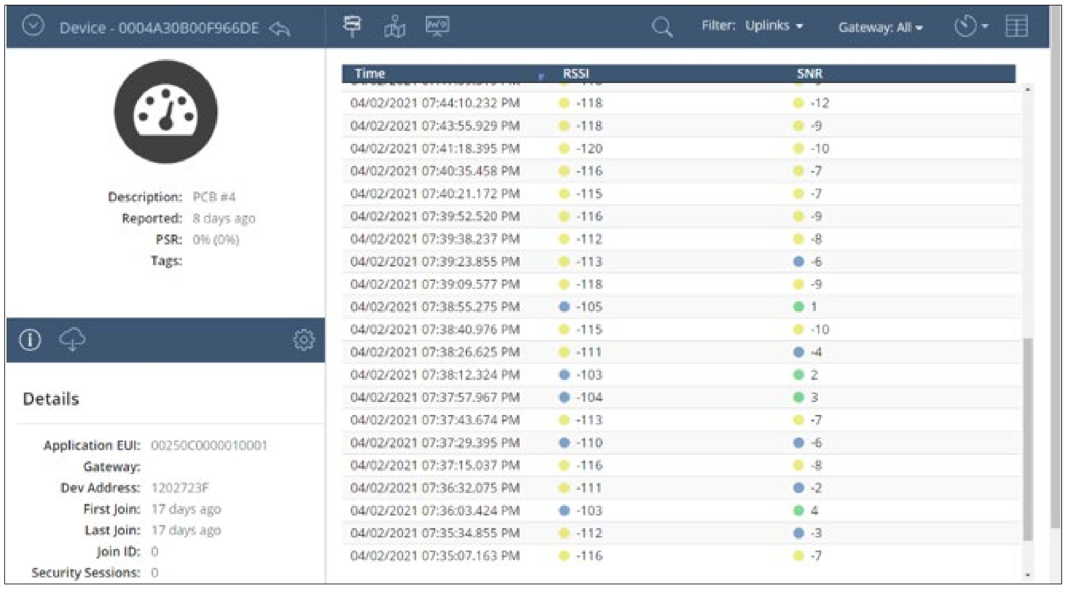
Details

Application EUI:	00250C0000010001
Gateway:	
Dev Address:	1202723F
First Join:	17 days ago
Last Join:	17 days ago
Join ID:	0
Security Sessions:	0
Frame Count Up:	1308
Frame Count Down:	21

Time	RSSI	SNR
04/02/2021 07:26:50.341 PM	🟡 -111	🟡 -7
04/02/2021 07:26:07.796 PM	🟡 -112	🟡 -8
04/02/2021 07:25:53.795 PM	🟡 -113	🔵 -4
04/02/2021 07:25:11.145 PM	🔵 -97	🟢 5
04/02/2021 07:24:42.820 PM	🟢 -88	🟢 8
04/02/2021 07:24:28.680 PM	🟢 -92	🟢 9
04/02/2021 07:24:14.410 PM	🟢 -94	🟢 4
04/02/2021 07:24:00.742 PM	🔵 -103	🟢 4
04/02/2021 07:23:32.903 PM	🟢 -73	🟢 9
04/02/2021 07:23:18.737 PM	🟢 -46	🟢 9
04/02/2021 07:23:04.545 PM	🟢 -54	🟢 9
04/02/2021 07:22:50.449 PM	🟢 -66	🟢 10
04/02/2021 07:22:36.274 PM	🟢 -53	🟢 8
04/02/2021 07:22:22.090 PM	🟢 -62	🟢 8
04/02/2021 07:22:07.913 PM	🟢 -60	🟢 6
04/02/2021 07:21:54.068 PM	🟢 -60	🟢 10
04/02/2021 07:21:39.597 PM	🟢 -60	🟢 9
04/02/2021 07:21:25.394 PM	🟢 -61	🟢 10
04/02/2021 07:21:11.220 PM	🟢 -56	🟢 9
04/02/2021 07:20:57.034 PM	🟢 -64	🟢 10
04/02/2021 07:20:42.864 PM	🟢 -61	🟢 9
04/02/2021 07:20:28.699 PM	🟢 -61	🟢 10
04/02/2021 07:20:14.572 PM	🟢 -68	🟢 8
04/02/2021 07:20:00.369 PM	🟢 -62	🟢 9

Figure 110: Senet Data from Distance Testing: Initial Position.

The Sensor Pod was driven down the road and measurements were recorded at 3.3km. The RSSI and SNR both went from very good to fair. When the Sensor Pod was removed from the vehicle and placed outside, it can be seen that the SNR value went from fair to good and very good.



Device - 0004A30B00F966DE

Filter: Uplinks

Gateway: All

Description: PCB #4
Reported: 8 days ago
PSR: 0% (0%)
Tags:

Details

Application EUI: 00250C0000010001
Gateway:
Dev Address: 1202723F
First Join: 17 days ago
Last Join: 17 days ago
Join ID: 0
Security Sessions: 0

Time	RSSI	SNR
04/02/2021 07:44:10.232 PM	-118	-12
04/02/2021 07:43:55.929 PM	-118	-9
04/02/2021 07:41:18.395 PM	-120	-10
04/02/2021 07:40:35.458 PM	-116	-7
04/02/2021 07:40:21.172 PM	-115	-7
04/02/2021 07:39:52.520 PM	-116	-9
04/02/2021 07:39:38.237 PM	-112	-8
04/02/2021 07:39:23.855 PM	-113	-6
04/02/2021 07:39:09.577 PM	-118	-9
04/02/2021 07:38:55.275 PM	-105	1
04/02/2021 07:38:40.976 PM	-115	-10
04/02/2021 07:38:26.625 PM	-111	-4
04/02/2021 07:38:12.324 PM	-103	2
04/02/2021 07:37:57.967 PM	-104	3
04/02/2021 07:37:43.674 PM	-113	-7
04/02/2021 07:37:29.395 PM	-110	-6
04/02/2021 07:37:15.037 PM	-116	-8
04/02/2021 07:36:32.075 PM	-111	-2
04/02/2021 07:36:03.424 PM	-103	4
04/02/2021 07:35:34.855 PM	-112	-3
04/02/2021 07:35:07.163 PM	-116	-7

Figure 111: Senet Data from Distance Testing: 3.3 km.

Because the Gateway was receiving a decent signal strength at the 3.3km distance, the Sensor Pod was driven further down the road and measurements were taken again at 6km. A visual representation of 6km is seen in Figure 112, and geographical representation of where the gateway and Sensor Pod were located is seen in Figure 113.

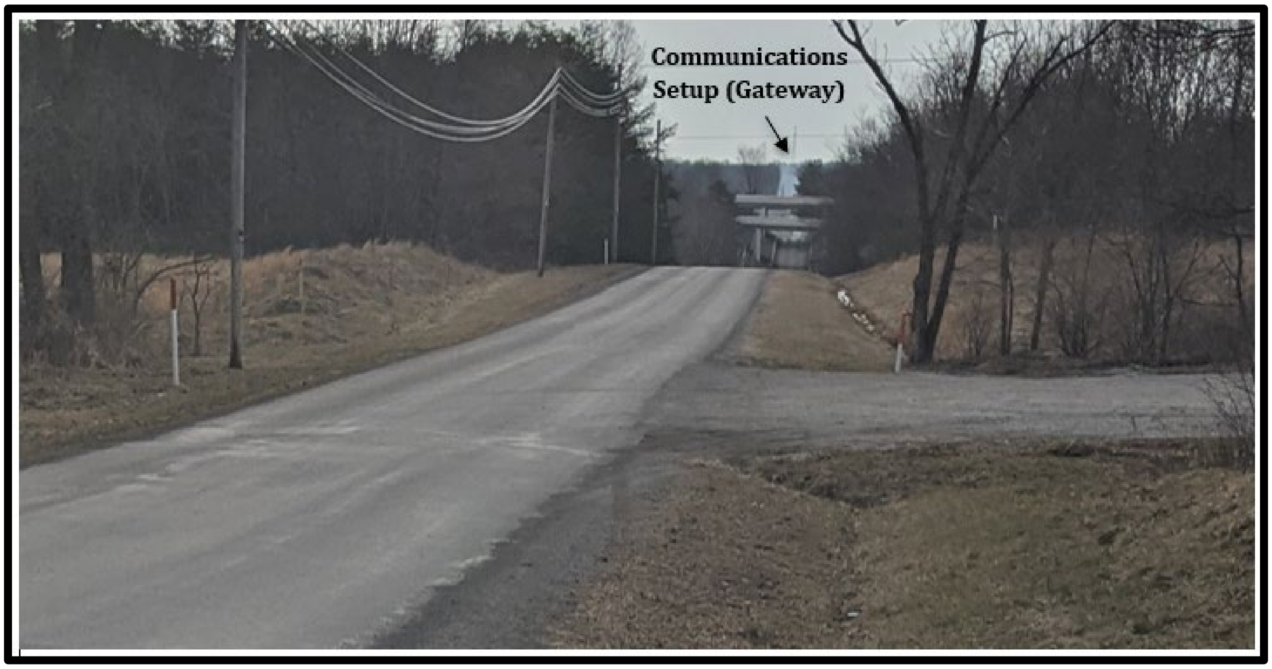


Figure 112: Distance Test: 6 km.

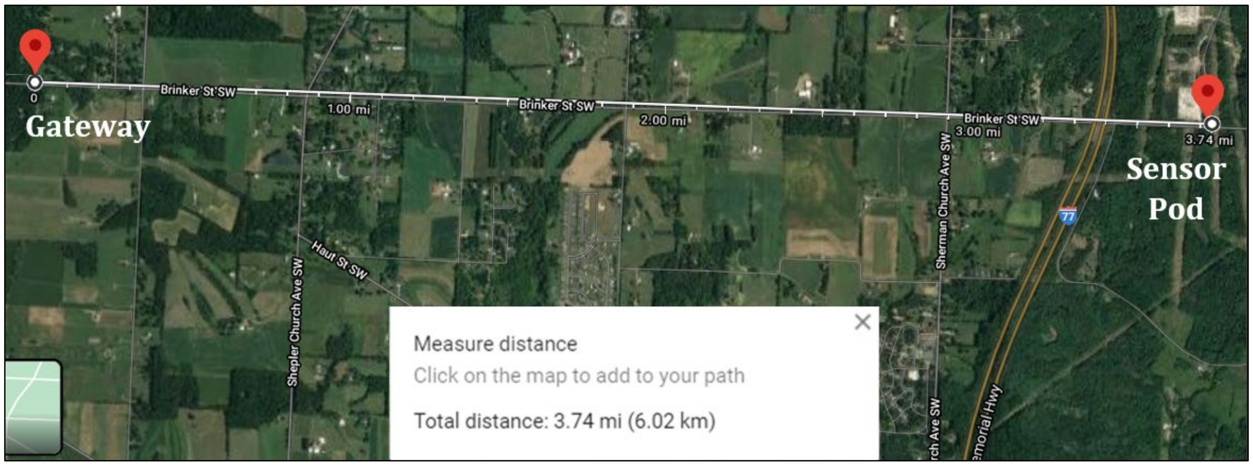
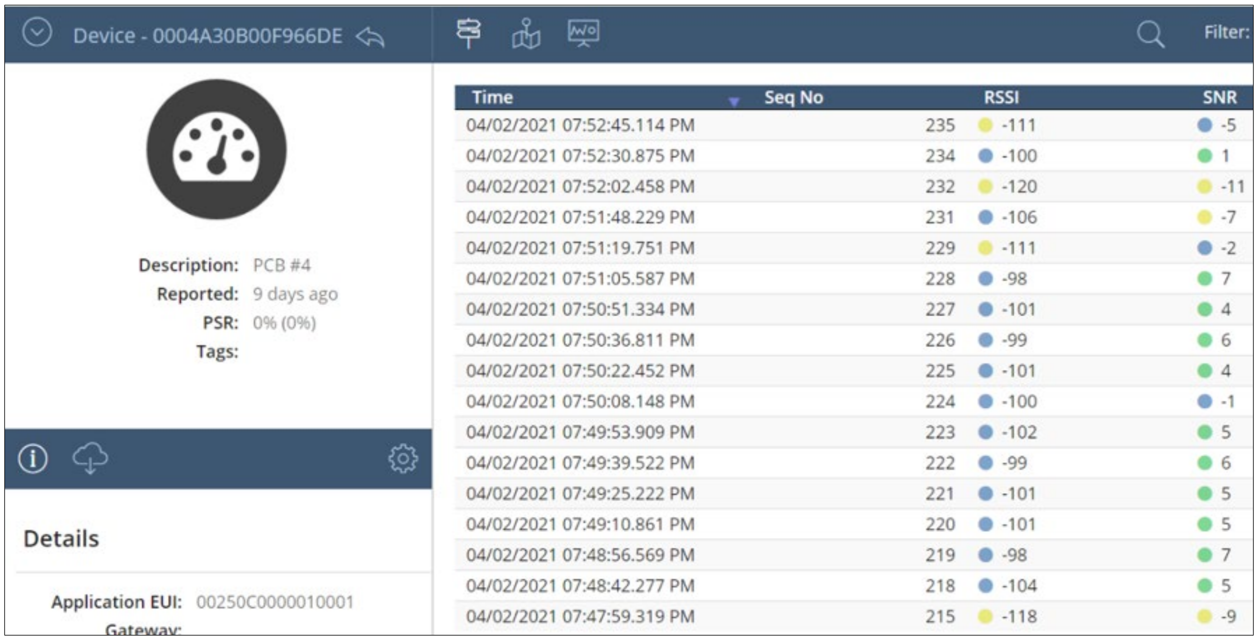


Figure 113: Satellite View of Gateway and Sensor Pod Locations.

As shown in Figure 114, the Senet server was receiving packets frequently without dropping many signals at a 6km LOS distance. Packet numbers are defined in the Seq No column. A number missing in the sequence corresponds to a dropped packet.



Description: PCB #4
Reported: 9 days ago
PSR: 0% (0%)
Tags:

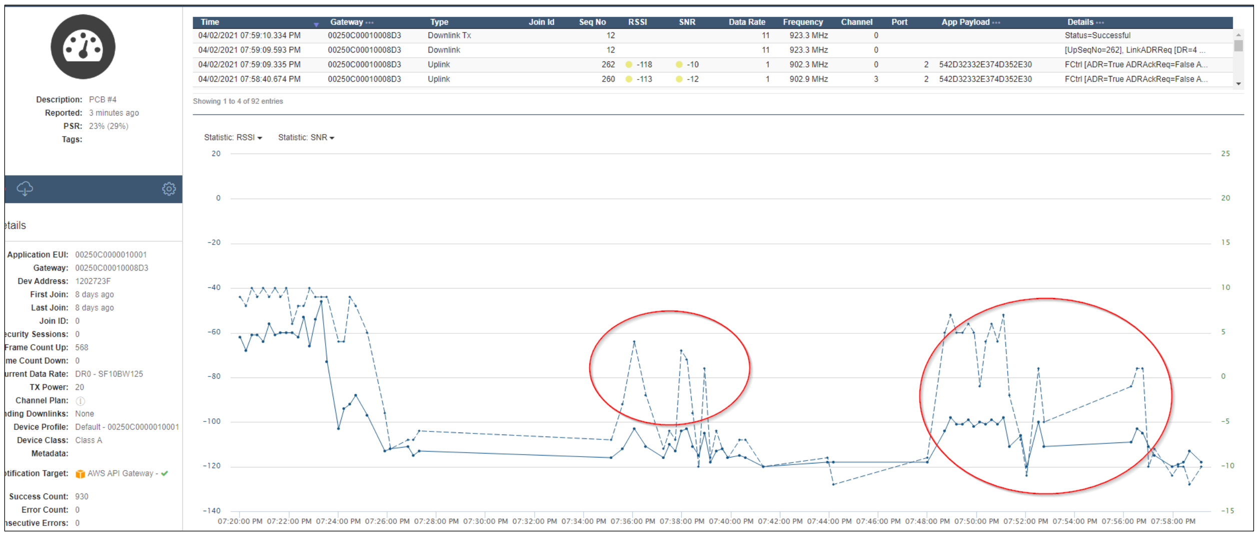
Application EUI: 00250C0000010001
Gateway:

Time	Seq No	RSSI	SNR
04/02/2021 07:52:45.114 PM	235	● -111	● -5
04/02/2021 07:52:30.875 PM	234	● -100	● 1
04/02/2021 07:52:02.458 PM	232	● -120	● -11
04/02/2021 07:51:48.229 PM	231	● -106	● -7
04/02/2021 07:51:19.751 PM	229	● -111	● -2
04/02/2021 07:51:05.587 PM	228	● -98	● 7
04/02/2021 07:50:51.334 PM	227	● -101	● 4
04/02/2021 07:50:36.811 PM	226	● -99	● 6
04/02/2021 07:50:22.452 PM	225	● -101	● 4
04/02/2021 07:50:08.148 PM	224	● -100	● -1
04/02/2021 07:49:53.909 PM	223	● -102	● 5
04/02/2021 07:49:39.522 PM	222	● -99	● 6
04/02/2021 07:49:25.222 PM	221	● -101	● 5
04/02/2021 07:49:10.861 PM	220	● -101	● 5
04/02/2021 07:48:56.569 PM	219	● -98	● 7
04/02/2021 07:48:42.277 PM	218	● -104	● 5
04/02/2021 07:47:59.319 PM	215	● -118	● -9

Figure 114: Senet Data from Distance Testing: 6 km.

There were only three dropped packets from a distance of 6km. The SNR was very good for the majority of the packets sent. At a 6km distance with the Sensor Pod buried 6 inches beneath the soil, the packet was transmitted at an average rate of once per minute. Dropped packets and slower transmission times are to be expected due to the propagation path loss and attenuation of the soil. The RSSI rate can be calculated by adding the SNR to external noise interference, which is generated by other signals transmitted on the same frequency. In the case of the test performed, both the gateway and Sensor Pod were across the street from power stations when these readings were taken, producing additional unwanted noise that most likely caused the RSSI to decreasing more quickly than the SNR.

To better understand the communication path taken while the Sensor Pod was driven down the road, the RSSI and SNR were plotted in the Senet Server.



Time	Gateway ···	Type	Join Id	Seq No	RSSI	SNR	Data Rate	Frequency	Channel	Port	App Payload ···	Details ···
04/02/2021 07:59:10.334 PM	00250C00010008D3	Downlink Tx		12			11	923.3 MHz	0			Status=Successful
04/02/2021 07:59:09.593 PM	00250C00010008D3	Downlink		12			11	923.3 MHz	0			[UpSeqNo=262], LinkADRReq [DR=4 ...
04/02/2021 07:59:09.335 PM	00250C00010008D3	Uplink		262	● -118	● -10	1	902.3 MHz	0	2	542D32332E374D352E30	FCtrl [ADR=True ADRAckReq=False A...
04/02/2021 07:58:40.674 PM	00250C00010008D3	Uplink		260	● -113	● -12	1	902.9 MHz	3	2	542D32332E374D352E30	FCtrl [ADR=True ADRAckReq=False A...

Showing 1 to 4 of 92 entries

Description:	PCB #4
Reported:	3 minutes ago
PSR:	23% (29%)
Tags:	

tails

Application EUI:	00250C0000010001
Gateway:	00250C00010008D3
Dev Address:	1202723F
First Join:	8 days ago
Last Join:	8 days ago
Join ID:	0
curity Sessions:	0
Frame Count Up:	568
me Count Down:	0
urrent Data Rate:	DR0 - SF10BW125
TX Power:	20
Channel Plan:	
nding Downlinks:	None
Device Profile:	Default - 00250C0000010001
Device Class:	Class A
Metadata:	
tification Target:	AWS API Gateway - ✓
Success Count:	930
Error Count:	0
nsecutive Errors:	0

Figure 115: Senet Data from Distance Testing: RSSI and SNR.

There are three strong signal strengths plotted on the graph, two of which are indicated with red circles. The first set of signals on the far left of the graph represents the signals transmitted by the Gateway during initial setup. The second set of signals, indicated by the red circle in the middle of the graph, are the packets transmitted at the 3.3km distance. The last set of signals, indicated by the circle on the right, are the packets transmitted at the 6km distance. There are quite a few dropped packets between these three main locations because the terrain had many hills; the Sensor Pod lost signal when in major valleys where there was not only no LOS, but also no way for a signal to escape the valley and arrive at the gateway. In addition to this, the Sensor Pod was in a bucket of dirt inside a moving vehicle. A moving antenna is more likely to induce a greater amount of unwanted electromagnetic interference into the air, which causes signal scattering and interrupts the communication path to the receiver.

5.4.5.3 *UA Propagation Model* ***AW***

For the Senior Design Day demonstration, it was desired to setup a distance test on the university property to show proof of concept. In a field, there is a direct Line of Sight (LOS) that exists between the Sensor Pod and the Gateway, disregarding the soil and electrical box. Because additional interferences exist on a college campus that do not exist in an open field (i.e.: buildings, signal interference, ground reflections du to concrete, etc.), it would not be possible for the signal to propagate 6 km without being received with large amounts of interference.

To determine how far the signal was capable of travelling, a MATLAB simulation was completed using Site Viewer. A map of the university campus was exported as a *.osm* file from Google Earth and referenced in the MATLAB code. The longitude and latitude coordinates, as well as antenna height, were then defined for the transmitter (Sensor Pod) and receiver (Gateway). In addition to this, the transmitter was set to a 915 MHz frequency. The model used a ray tracing method to plot the propagated signal from the Sensor Pod to the Gateway, given the geographical coordinates. The model also took into consideration building and ground reflections to determine the strength of the signal. Buildings and ground were assumed to be perfect reflectors.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

viewer = siteviewer("Buildings","map.osm");

% Create receiver sites around UA campus.
tx = txsite('Latitude',41.075253, ...
            'Longitude', -81.50733, ...
            'AntennaHeight',0.5, ...
            'TransmitterFrequency',915e6);

% Create transmitter site on 5th floor of ASEC.
rx = rxsite('Latitude',41.076520, ...
            'Longitude',-81.513290, ...
            'AntennaHeight', 22);


% Compute signal strength using ray tracing propagation model and
default single-reflection analysis.
pm = propagationModel("raytracing-image-method");
ssOneReflection = sigstrength(rx,tx,pm)

% Compute signal strength with analysis up to two reflections, where
total received power is the cumulative power of all propagation paths
pm.MaxNumReflections = 2;
ssTwoReflections = sigstrength(rx,tx,pm)

% Observe effect of material by replacing default concrete material
with perfect reflector.
pm.BuildingsMaterial = 'perfect-reflector';
ssPerfect = sigstrength(rx,tx,pm)

% Plot propagation paths.
raytrace(tx, rx, pm)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

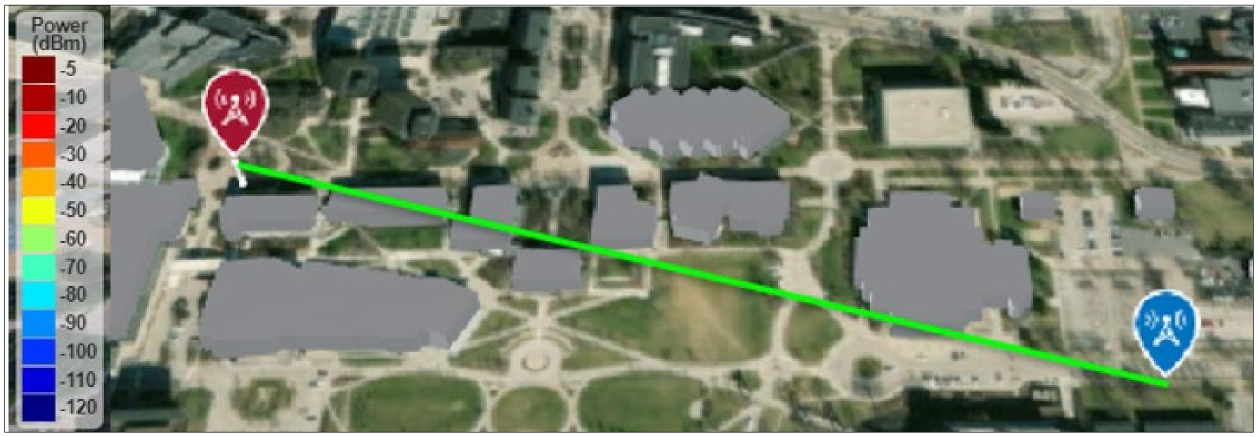
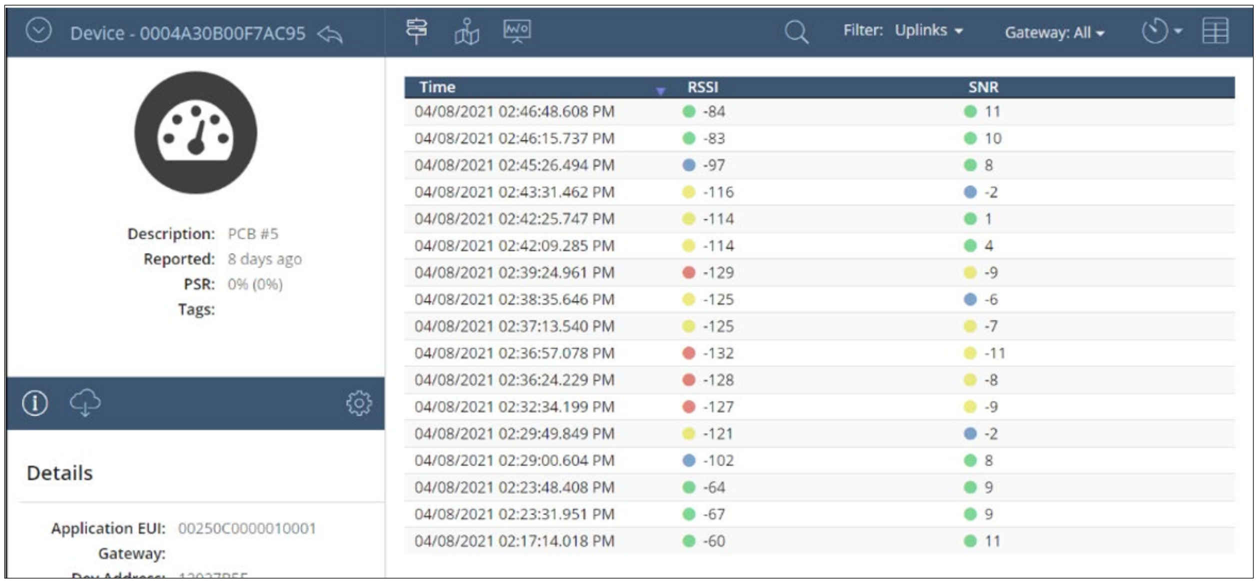


Figure 116: Propagation Model on the University of Akron Campus: 500m Distance.

The Gateway (red) was stationed on the 5th floor of ASEC. The sensor pod was buried three inches beneath the soil in the container used for the previous distance testing and placed in the grassy area beside the Natatorium across campus. The propagation model showed, in theory, the

Gateway should receive a strong signal. On demonstration day, the pod was taken to the determined location and signals were transmitted back to the Gateway.


Device - 0004A30B00F7AC95

Filter: Uplinks Gateway: All

Description: PCB #5
Reported: 8 days ago
PSR: 0% (0%)
Tags:

Details

Application EUI: 00250C0000010001
Gateway:

Time	RSSI	SNR
04/08/2021 02:46:48.608 PM	🟢 -84	🟢 11
04/08/2021 02:46:15.737 PM	🟢 -83	🟢 10
04/08/2021 02:45:26.494 PM	🔵 -97	🟢 8
04/08/2021 02:43:31.462 PM	🟡 -116	🔵 -2
04/08/2021 02:42:25.747 PM	🟡 -114	🟢 1
04/08/2021 02:42:09.285 PM	🟡 -114	🟢 4
04/08/2021 02:39:24.961 PM	🔴 -129	🟡 -9
04/08/2021 02:38:35.646 PM	🟡 -125	🔵 -6
04/08/2021 02:37:13.540 PM	🟡 -125	🟡 -7
04/08/2021 02:36:57.078 PM	🔴 -132	🟡 -11
04/08/2021 02:36:24.229 PM	🔴 -128	🟡 -8
04/08/2021 02:32:34.199 PM	🔴 -127	🟡 -9
04/08/2021 02:29:49.849 PM	🟡 -121	🔵 -2
04/08/2021 02:29:00.604 PM	🔵 -102	🟢 8
04/08/2021 02:23:48.408 PM	🟢 -64	🟢 9
04/08/2021 02:23:31.951 PM	🟢 -67	🟢 9
04/08/2021 02:17:14.018 PM	🟢 -60	🟢 11

Figure 117: Senet Data from Distance Testing: RSSI and SNR.

As displayed in Figure 117, signals can be received clearly from a distance of 500m on campus beneath three inches of soil, verifying the Engineering Requirement that the Sensor Pod will transmit data through 3 inches of soil.

5.4.6 Web Application **AA, RK**

The implementation of the web application followed the design stated in § 5.2.8. The primary purpose of the web application was to allow farmers an easy and functional way to view Sensor Pod data and receive alerts. The website was broken into three main pages as shown in Figure 118, each serving their own primary functions.

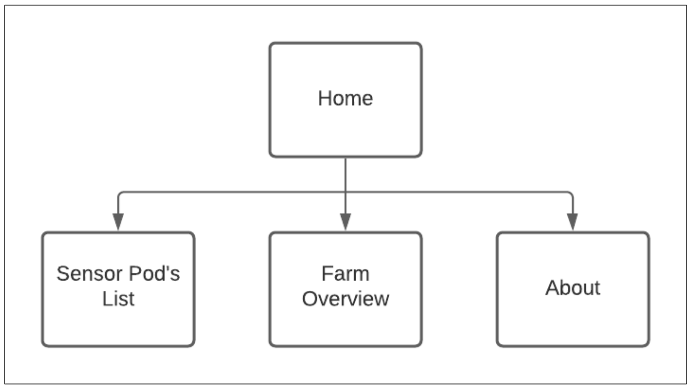
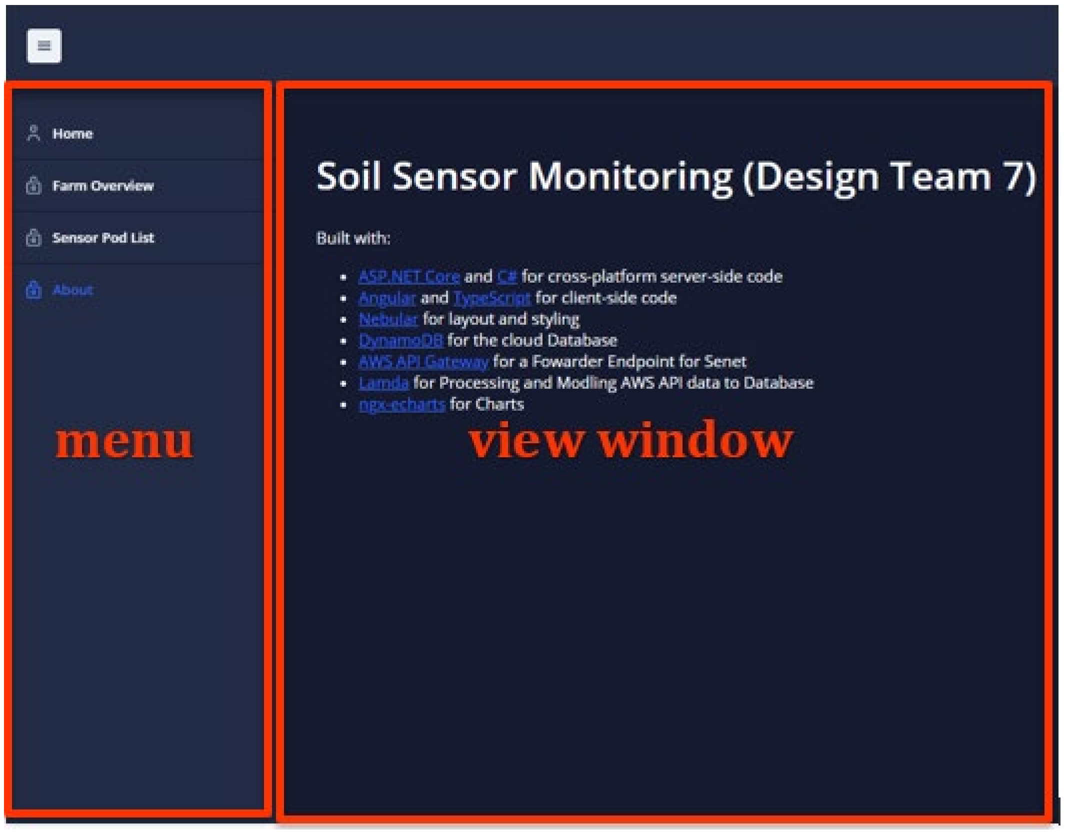


Figure 118: Soil Sensor Network Web Application: Site Map.

5.4.6.1 *Website Interface*

Since the website is implemented as a single page application, the layout is broken down into view window and a side menu as shown in Figure 120. All pages are displayed in the view window, and only the information inside the view windows is updated when switching pages allowing for faster page loading. The side menu is used for navigation between pages.



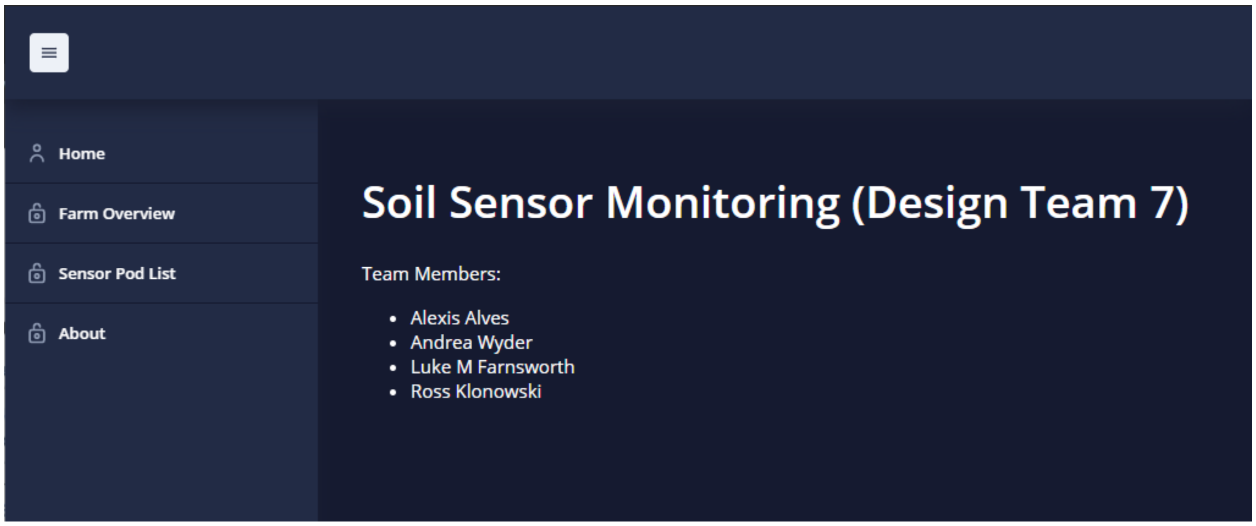
Soil Sensor Monitoring (Design Team 7)

Built with:

- ASP.NET Core and C# for cross-platform server-side code
- Angular and TypeScript for client-side code
- Nebular for layout and styling
- DynamoDB for the cloud Database
- AWS API Gateway for a Fowarder Endpoint for Senet
- Lamda for Processing and Modling AWS API data to Database
- ngx-echarts for Charts

Figure 119: Soil Sensor Network Web Application: Main Interface.

The Home page used as means to welcome the farmer to the application and ask for a valid Farm Id to ensure the farmer access only their farms' information. The page is shown below in Figure 121, the side menu is limited to only the Home and About page until a valid Farm ID. During the demonstration, this page was substituted with the interface seen in Figure 122 to introduce the design team, and all pages were available in the menu.



Soil Sensor Monitoring (Design Team 7)

Team Members:

- Alexis Alves
- Andrea Wyder
- Luke M Farnsworth
- Ross Klonowski

Figure 120: Soil Sensor Network Web Application: Home Page.

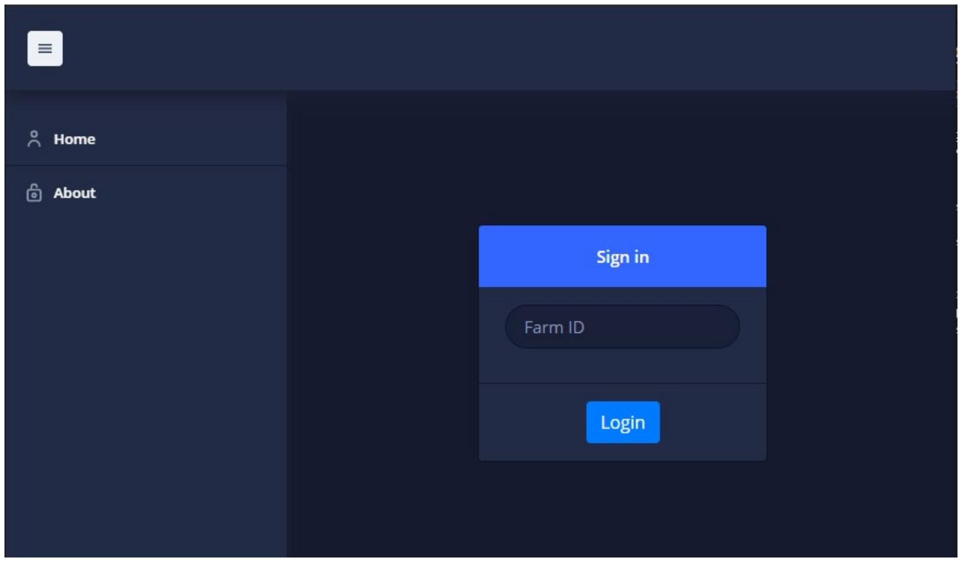
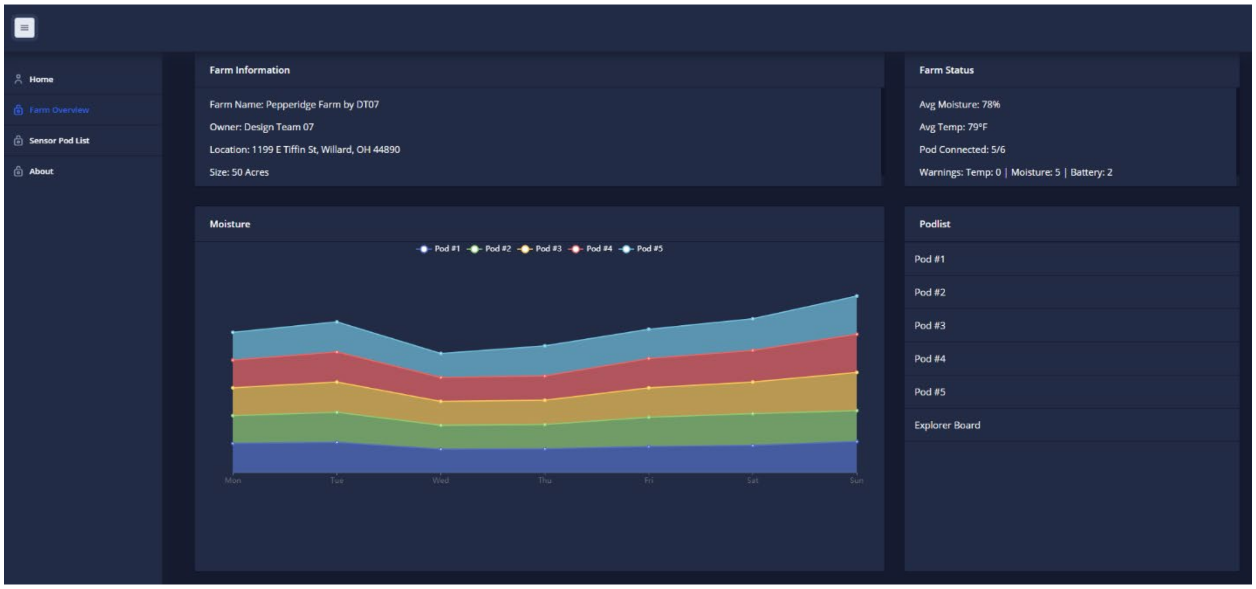


Figure 121: Soil Sensor Network Web Application: Farm ID Sign In.

Once a valid Farm Id is given, the menu bar is updated, and the user is redirected to the Farm Overview page. The Farm Overview page allows the farmer to view a quick overview of farm information as seen in Figure 122.



Farm Information

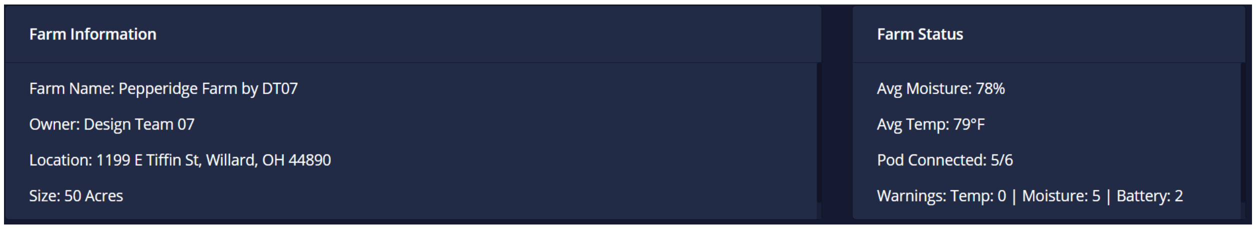
Farm Name: Pepperidge Farm by DT07
Owner: Design Team 07
Location: 1199 E Tiffin St, Willard, OH 44890
Size: 50 Acres

Farm Status

Avg Moisture: 78%
Avg Temp: 79°F
Pod Connected: 5/6
Warnings: Temp: 0 | Moisture: 5 | Battery: 2

Figure 122: Soil Sensor Network Web Application: Farm Overview.

The main functionality of this page are the moisture graph, pod list, and farm information including any alerts about sensor pods. At the top of the page the farmer can see Farm Information and Farm Status with information about their farm.



Farm Information

Farm Name: Pepperidge Farm by DT07
Owner: Design Team 07
Location: 1199 E Tiffin St, Willard, OH 44890
Size: 50 Acres

Farm Status

Avg Moisture: 78%
Avg Temp: 79°F
Pod Connected: 5/6
Warnings: Temp: 0 | Moisture: 5 | Battery: 2

Figure 123: Soil Sensor Network Application: Farm Information and Farm Status.

The moisture graph allows the farmer look at the weekly trend of the moisture percentage for the farm. And while hovering over a day they can see what the value for the highest percentage pods.

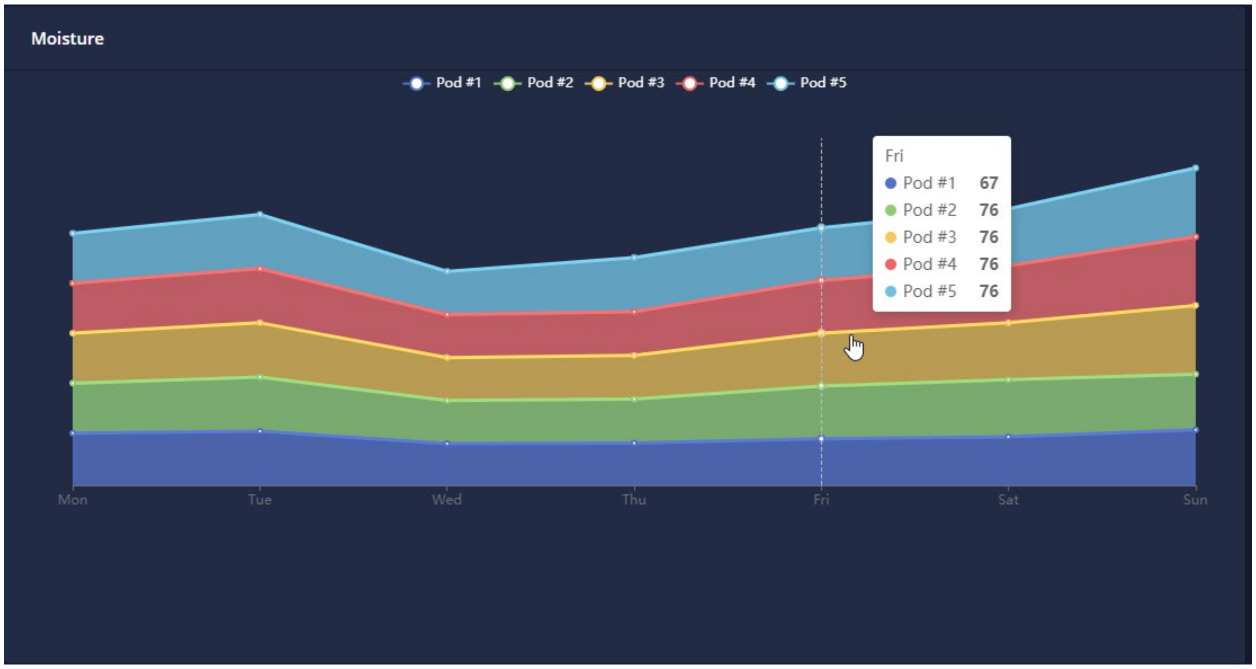
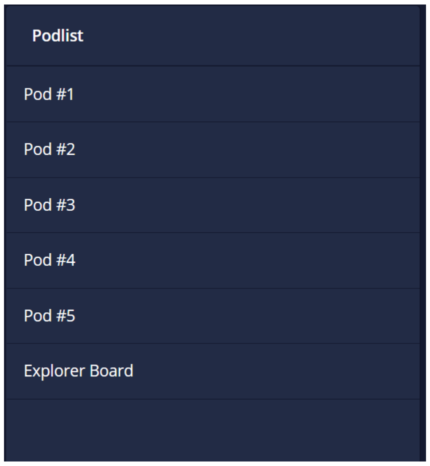


Figure 124: Soil Sensor Network Web Application: Trended Data

The Pod List, as seen in Figure 125, provides a list of pods currently assigned to the farm; this list only shows names as more detailed information is shown in the Sensor Pod List page.



Podlist

Pod #1

Pod #2

Pod #3

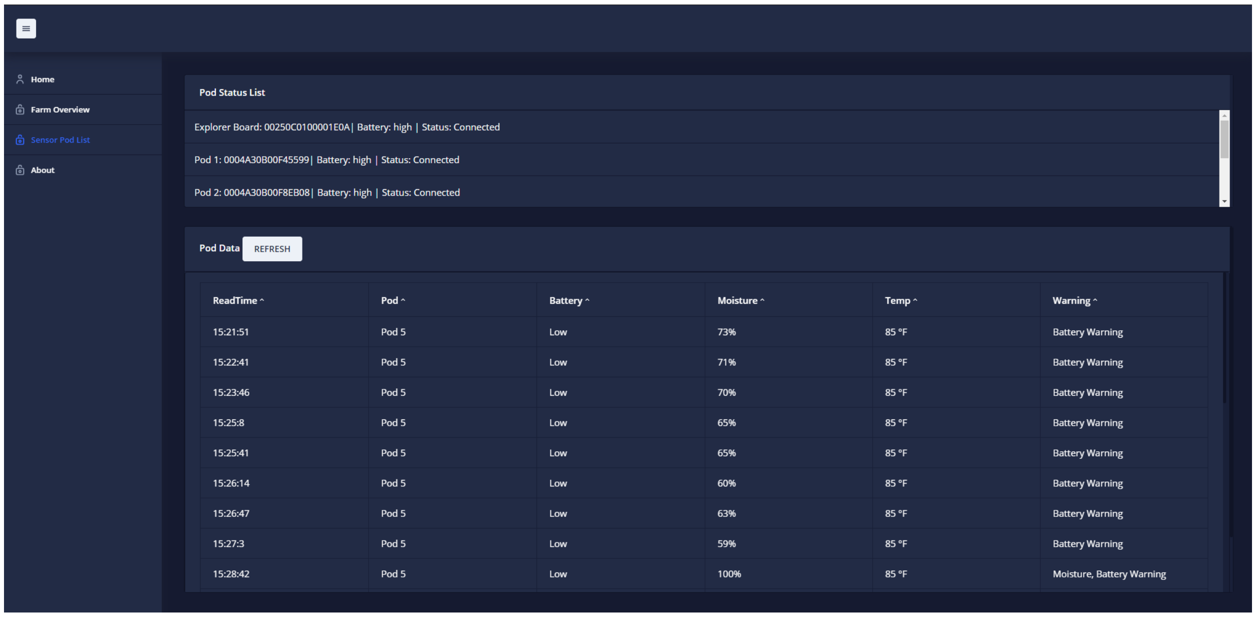
Pod #4

Pod #5

Explorer Board

Figure 125: Soil Sensor Network Web Application: Pod List.

The Pod List Page lets farmers get a more detailed information from their Sensor Pods as shown in Figure 126. The page has two main components: Pod Status List and Pod Data.



Pod Status List

Explorer Board: 00250C0100001E0A| Battery: high | Status: Connected

Pod 1: 0004A30B00F45599| Battery: high | Status: Connected

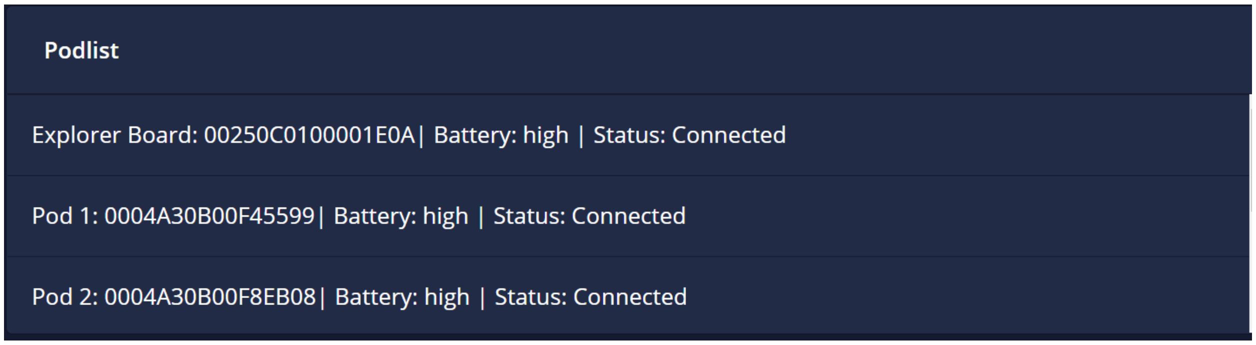
Pod 2: 0004A30B00F8EB08| Battery: high | Status: Connected

Pod Data

ReadTime ^	Pod ^	Battery ^	Moisture ^	Temp ^	Warning ^
15:21:51	Pod 5	Low	73%	85 °F	Battery Warning
15:22:41	Pod 5	Low	71%	85 °F	Battery Warning
15:23:46	Pod 5	Low	70%	85 °F	Battery Warning
15:25:8	Pod 5	Low	65%	85 °F	Battery Warning
15:25:41	Pod 5	Low	65%	85 °F	Battery Warning
15:26:14	Pod 5	Low	60%	85 °F	Battery Warning
15:26:47	Pod 5	Low	63%	85 °F	Battery Warning
15:27:3	Pod 5	Low	59%	85 °F	Battery Warning
15:28:42	Pod 5	Low	100%	85 °F	Moisture, Battery Warning

Figure 126: Soil Sensor Network Web Application: Pod Status List and Pod Data.

The Pod Status List allows for the farmer to see the status of all pods that are currently assigned to their farm. This list, unlike the one on the Farm Overview page, gives information about the Sensor Pods device EUI, battery status, and its connection status as shown in Figure 127.


Podlist

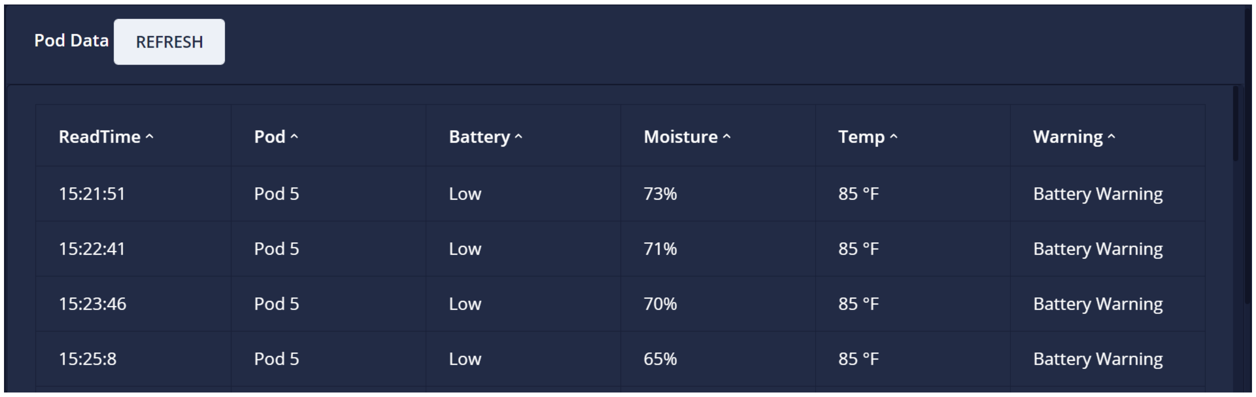
Explorer Board: 00250C0100001E0A| Battery: high | Status: Connected

Pod 1: 0004A30B00F45599| Battery: high | Status: Connected

Pod 2: 0004A30B00F8EB08| Battery: high | Status: Connected

Figure 127: Soil Sensor Network Web Application: Pod Status List.

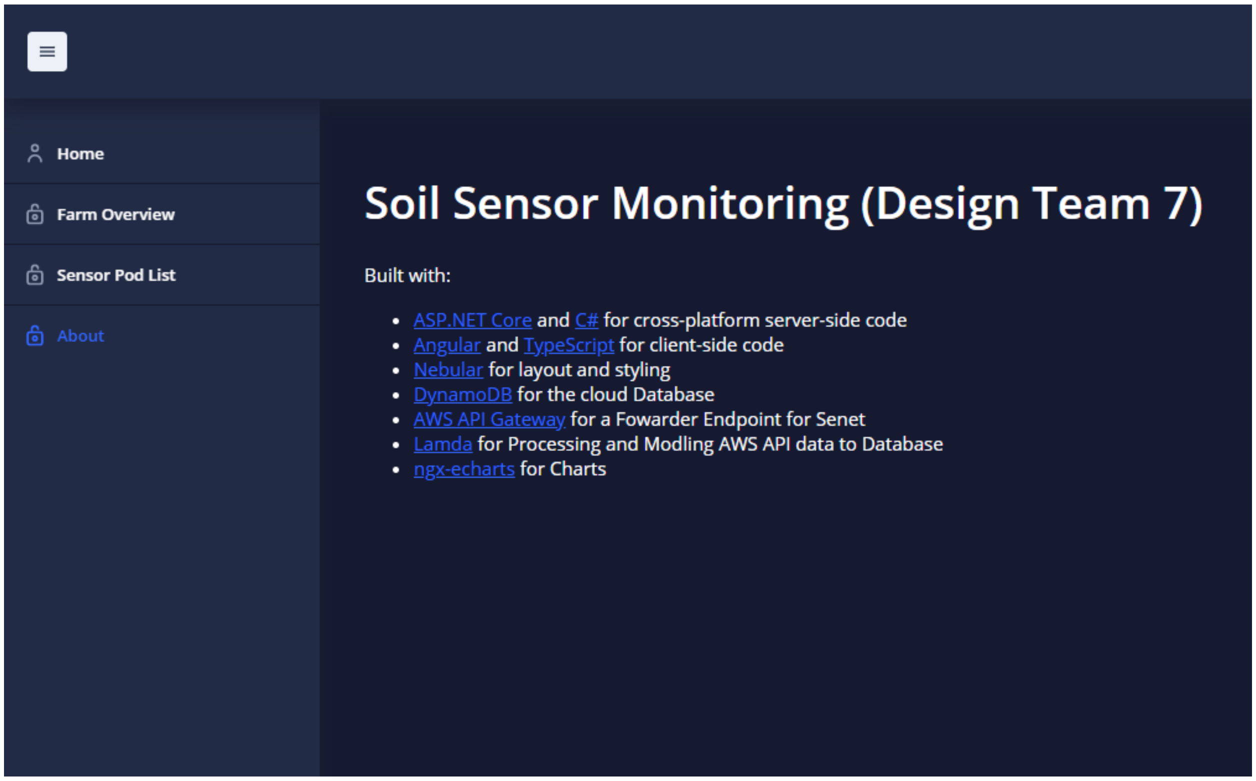
The Pod Data components gives the farmer more detailed information about their Sensor Pods by listing all data collected. Each entry on the list shows the status and the data collected form a Sensor Pod along with the time it was taken as seen in Figure 128. Each entry also shows if any warning was triggered by the information such as temperature and moisture warning.



ReadTime ^	Pod ^	Battery ^	Moisture ^	Temp ^	Warning ^
15:21:51	Pod 5	Low	73%	85 °F	Battery Warning
15:22:41	Pod 5	Low	71%	85 °F	Battery Warning
15:23:46	Pod 5	Low	70%	85 °F	Battery Warning
15:25:8	Pod 5	Low	65%	85 °F	Battery Warning

Figure 128: Soil Sensor Network Web Application: Pod Data.

The About page shows a list of frameworks, database, and language used for the development of the web application.



Soil Sensor Monitoring (Design Team 7)

Built with:

- ASP.NET Core and C# for cross-platform server-side code
- Angular and TypeScript for client-side code
- Nebular for layout and styling
- DynamoDB for the cloud Database
- AWS API Gateway for a Fowarder Endpoint for Senet
- Lamda for Processing and Modling AWS API data to Database
- ngx-echarts for Charts

Figure 129: Soil Sensor Network Web Application: About Page.

5.4.6.2 *Frontend*

The Web Application as explained in § 5.2.8 is based on a MVC scheme. The frontend of the application handles displaying and receiving information to the farmer for the backend to process. Each page of the application is broken into three main components: typescript, html, and sass. The typescript component handles all typescript logic, while the html and sass components determine the how the page is displayed. These components are linked together in the typescript @Componets section.

```
@Component({
  selector: 'app-sensorPodList',
  templateUrl: './sensorPodList.component.html',
  styleUrls: ['./sensorPodList.component.scss']
})
```

Since the web application has many components that are modular in nature, some of the code is reused. For this reason, only some of the code is shown in this section, the full source code can be found in the Appendix.

The following are the four main code schemes for displaying information to the farmer. To keep information as a standard theme, <nb-card> were used on all schemes to encapsulate the information in a consistent way. The main way of displaying information is using basic html <p> tags. This displays information of the web application in standard paragraph form. The information is pulled from the page typescript components by using Angular expressions {{information }}.

```
<nb-card size="tiny">
        <nb-card-header>Farm Information</nb-card-header>
        <nb-card-body>
          <p>Farm Name: {{this.farmInformations.Name}}</p>
          <p>Owner: {{this.farmInformations.Owner}}</p>
          <p>Location: {{this.farmInformations.Location}}</p>
          <p>Size: {{this.farmInformations.Size}}</p>
        </nb-card-body>
      </nb-card>
    </div>
```

The list scheme uses the same method to pull information from the typescript components, though it incorporates a loop components “ngFor” to iteratively add information from a list to be displayed.

```
<nb-card size="large">
        <nb-card-header>Podlist</nb-card-header>
        <nb-list>
          <nb-list-item *ngFor="let device of this.device">
            {{ device }}
          </nb-list-item>
        </nb-list>
```

The last two schemes leverage external and internal classes. The graph uses ngx-echart class; this class is an external chart tool that formats information into custom graphs. The ngx-echart class was used to aid in the visual representation of the moisture trends. This allows farmer to better understand information about their farms. The function calls and structure of this scheme are seen below.

```
 <nb-card size="large">
        <nb-card-header>Moisture</nb-card-header>
        <div echarts [options]="options" theme="macarons" class="chart"></div>
 </nb-card>
```

The Grid data scheme uses the internal ngTreeGrid class to model the data into a grid list format. This class was used to keep the aesthetic and flow of the website similar throughout. The grid was an effective way to display large amounts of data in one location. The use of the grid also allows for information to be refreshed local only to the grid card to improve performance. Other functionality, such as the ability to sort by different tabs, allows the farmer to search through their pod data effectively.

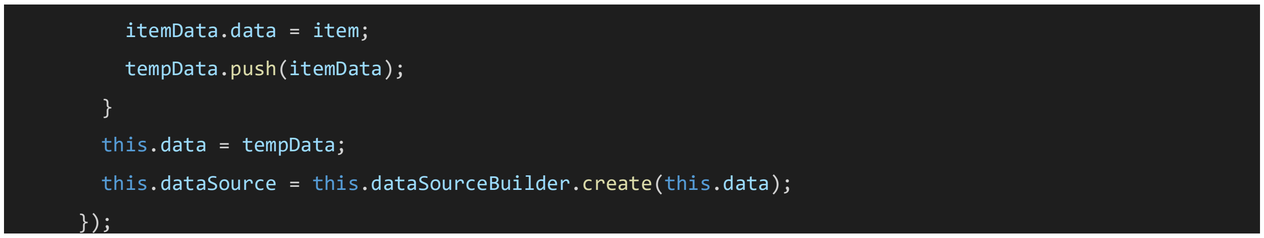
```html
<nb-card size="giant">
  <nb-card-header>
    Pod Data
    <button (click)="onReload()" nbButton>Refresh</button>
  </nb-card-header>
  <div overflow-y=scroll>
    <nb-card>
      <nb-card-body>
        <table [nbTreeGrid]="data"
               nbSort (sort)="changeSort($event)"
               equalColumnsWidth>
          <tr nbTreeGridHeaderRow *nbTreeGridHeaderRowDef="allColumns"></tr>
          <tr nbTreeGridRow *nbTreeGridRowDef="let row; columns: allColumns"></tr>
          <ng-container [nbTreeGridColumnDef]="customColumn">
            <th nbTreeGridHeaderCell [nbSortHeader]="getDirection(customColumn)"
                *nbTreeGridHeaderCellDef>
              {{customColumn}}
            </th>
            <td nbTreeGridCell *nbTreeGridCellDef="let row">
              <nb-tree-grid-row-toggle [expanded]="row.expanded"
                                       *ngIf="row.data.kind === 'dir'">
              </nb-tree-grid-row-toggle>
              {{row.data.ReadTime}}
            </td>
          </ng-container>
          <ng-container *ngFor="let column of defaultColumns"
                        [nbTreeGridColumnDef]="column">
            <th nbTreeGridHeaderCell [nbSortHeader]="getDirection(column)"
                *nbTreeGridHeaderCellDef>
              {{column}}
            </th>
            <td nbTreeGridCell *nbTreeGridCellDef="let row">
              {{row.data[column]}}
            </td>
          </ng-container>
        </table>
      </nb-card-body>
    </nb-card>
  </div>
```

The frontend handles all interaction from the farmer in the typescript components of each page. Since most pages are structured in a similar format the Pod List page will be used as an example. Functions are defined in the typescript component of a page such as onReload( ) as seen below can be linked to interaction in the html tags as such <button (click)="onReload()" nbButton>. For this onReload function when the farmer selects to refresh the Pod Data list the function triggers a call to update the information on the list.

```
onReload(){
  this.getPodData();
 }
 changeSort(sortRequest: NbSortRequest): void {

}
```

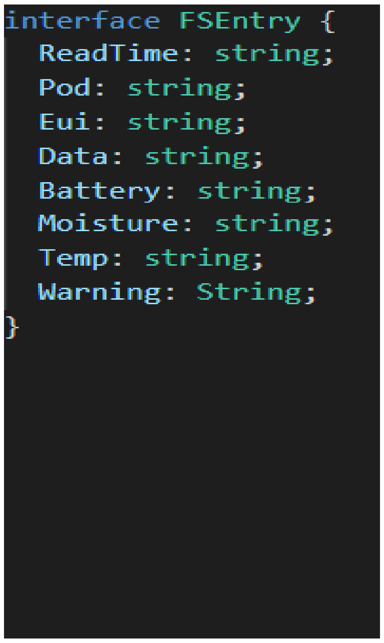
The data used on the frontend is sourced from the backend via API calls as explained in § 5.2.8.2. The API call structure lets the backend know what information is needed and for which farm. The example below the call is “'https://localhost:44385/DynamoDB/PodData/id?=’ + this.farmId “ the call is requesting information from DynamoDB for the Farm with the user’s Farm ID. Once the call is returned the data is mapped to its appropriate model and displayed by the frontend.

```
getPodData()
{
  this._http.get('https://localhost:44385/DynamoDB/PodData/id?=’   + this.farmId  ).subscribe(

    result => {
      let tempdata: any =  result;
      let tempData: TreeNode<FSEntry>[] = [];
      for (let element of tempdata) {
        let itemData = {} as TreeNode<FSEntry>;
        let item = {} as FSEntry;

        item.Pod = element.name;
        item.Temp = element.temp_Sensor_Value + '%';
        item.Eui = element.devEui + ' °F';
        item.ReadTime = this.formatTime(element.time);
        item.Moisture = element.moisture_Sensor_Value;
        item.Battery = element.bat_Value
        item.Warning = this.checkWarning(element.temp_Sensor_Value,
                                        element.moisture_Sensor_Value,
                                        element.bat_Value)
```

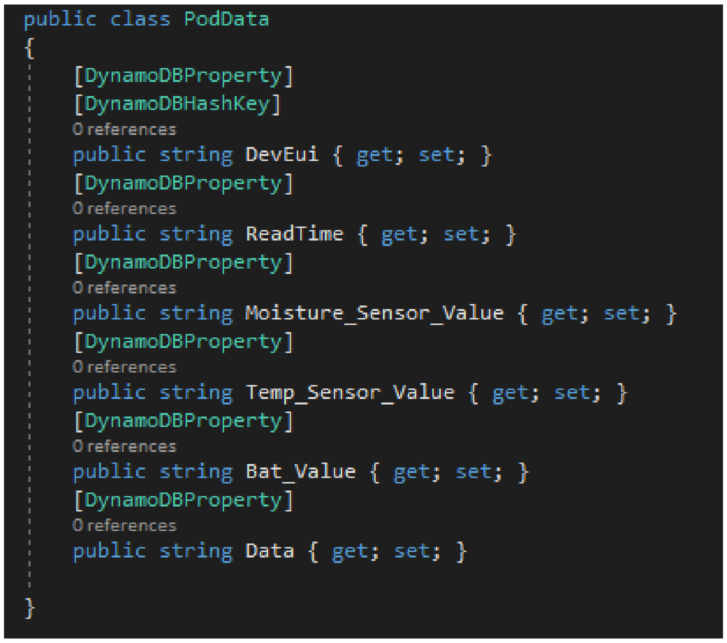


```
          itemData.data = item;
          tempData.push(itemData);
        }
        this.data = tempData;
        this.dataSource = this.dataSourceBuilder.create(this.data);
      });
```

The data that is returned to the frontend is in a format that is defined by its data model. The data models are defined similar in both frontend and backend as seen below. This allows the website to be added to over time, since the data models can be changed without having to rewrite the entire call structure.



```
interface FSEntry {
  ReadTime: string;
  Pod: string;
  Eui: string;
  Data: string;
  Battery: string;
  Moisture: string;
  Temp: string;
  Warning: String;
}
```



```
public class PodData
{
    [DynamoDBProperty]
    [DynamoDBHashKey]
    0 references
    public string DevEui { get; set; }
    [DynamoDBProperty]
    0 references
    public string ReadTime { get; set; }
    [DynamoDBProperty]
    0 references
    public string Moisture_Sensor_Value { get; set; }
    [DynamoDBProperty]
    0 references
    public string Temp_Sensor_Value { get; set; }
    [DynamoDBProperty]
    0 references
    public string Bat_Value { get; set; }
    [DynamoDBProperty]
    0 references
    public string Data { get; set; }

}
```

5.4.6.3 *Backend*

The backend of the web application handles the logic and data retrieval from the database. When an API call is made to from the frontend, it's routed to the appropriate call function. For the Pod data API call, the backend routes the call to the [HttpGet] [Route("PodData")]. The function uses the information passed from the frontend to retrieve the requested data from the DynamoDB database.

```
 [HttpGet]
       [Route("PodData")]
       public async Task<List<PodData>> GetPodData([FromQuery] string id,string paginationToken = "")
       {
           try
           {
               return await _podDataContext.GetaAll(id);
           }
           catch (Exception ex)
           {
               throw new Exception($"Amazon error in GetUser table operation! Error: {ex}");
           }
       }
```

Once the function gets the API call with the required information, it waits for the information to be retrieved form the backend DynamoDB interface. The DynamoDB interface makes an API call using the information to the DynamoDB database in AWS. The API call carries conditions so that only the information that is being requested is returned.

```
public async Task<List<T>> GetaAllPods()
       {
           var scanConditions = new List<ScanCondition>()
           {
               new ScanCondition("DevEui", ScanOperator.NotEqual, "1"),
           };
           var searchResults = base.ScanAsync<T>(scanConditions, null);
           return await searchResults.GetNextSetAsync();
       }
```

The function call is a template-based function, which lets the function know what model to use for the information. The interface for the function can be seen below. The value T is replaced with the appropriate class which represents the data model to be used. The class that represents the data model can be seen at the end of § 5.4.6.3.

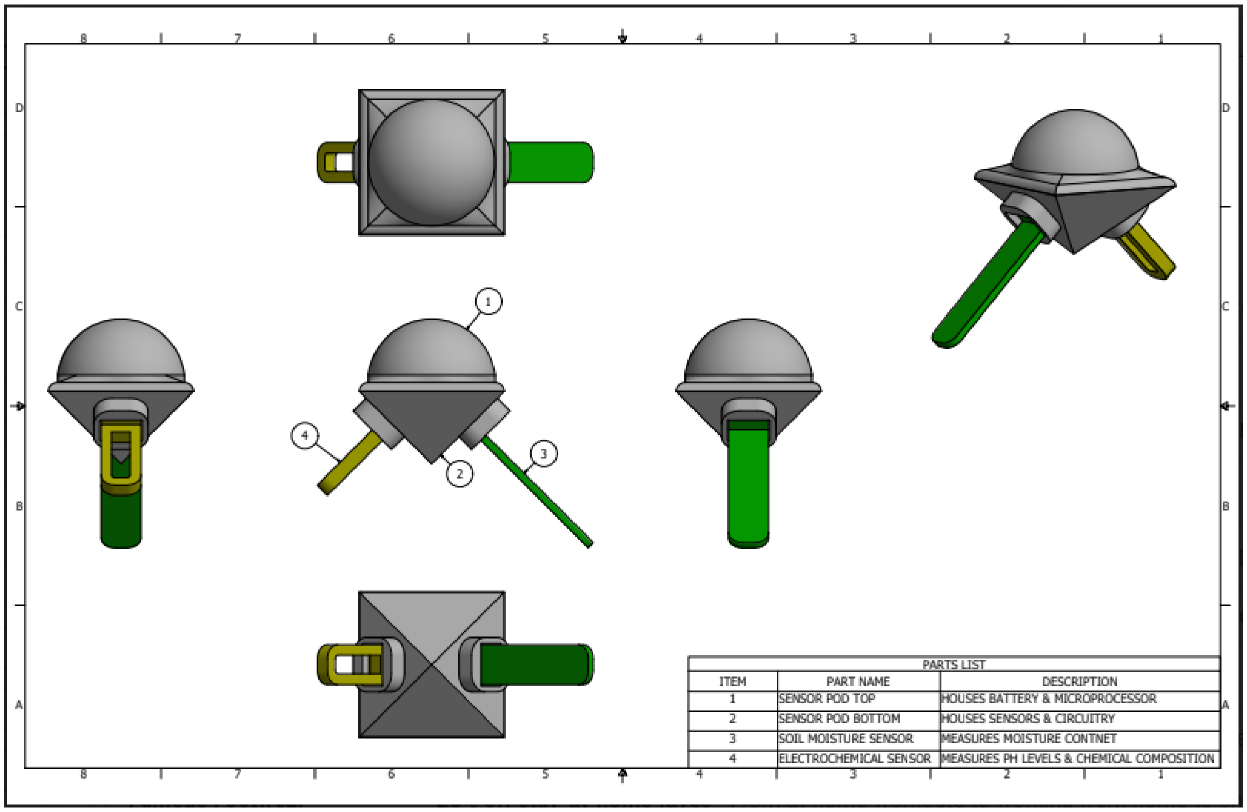
```
namespace SoilSensor.Data.Interface
{
    public interface IDynamoDBContext<T> : IDisposable where T : class
    {
        Task<T> GetByIdAsync(string id , string hash);
        Task SaveAsync(T item);
        Task DeleteByIdAsync(T item);
        string GetTable();
        Task<List<T>> GetaAll(string id);
        Task<List<T>> GetaAllPods(string id);
    }
}
```

6 Mechanical Sketch AW

As stated previously in the Engineering Requirements, the dimensions of the pod should be no larger than 90 x 90 x 100 mm, which will allow the pod to fit into the average planter.

6.1 First Design Iteration

In the first design iteration, the dimensions of the pod itself was minimized, excluding external sensors as seen in Figure 130.



PARTS LIST		
ITEM	PART NAME	DESCRIPTION
1	SENSOR POD TOP	HOUSES BATTERY & MICROPROCESSOR
2	SENSOR POD BOTTOM	HOUSES SENSORS & CIRCUITRY
3	SOIL MOISTURE SENSOR	MEASURES MOISTURE CONTNET
4	ELECTROCHEMICAL SENSOR	MEASURES PH LEVELS & CHEMICAL COMPOSITION

Figure 130: Phase 1 Mechanical Sketch of Sensor Pod.

The capacitive soil moisture sensor upon which the design will be based has dimensions of

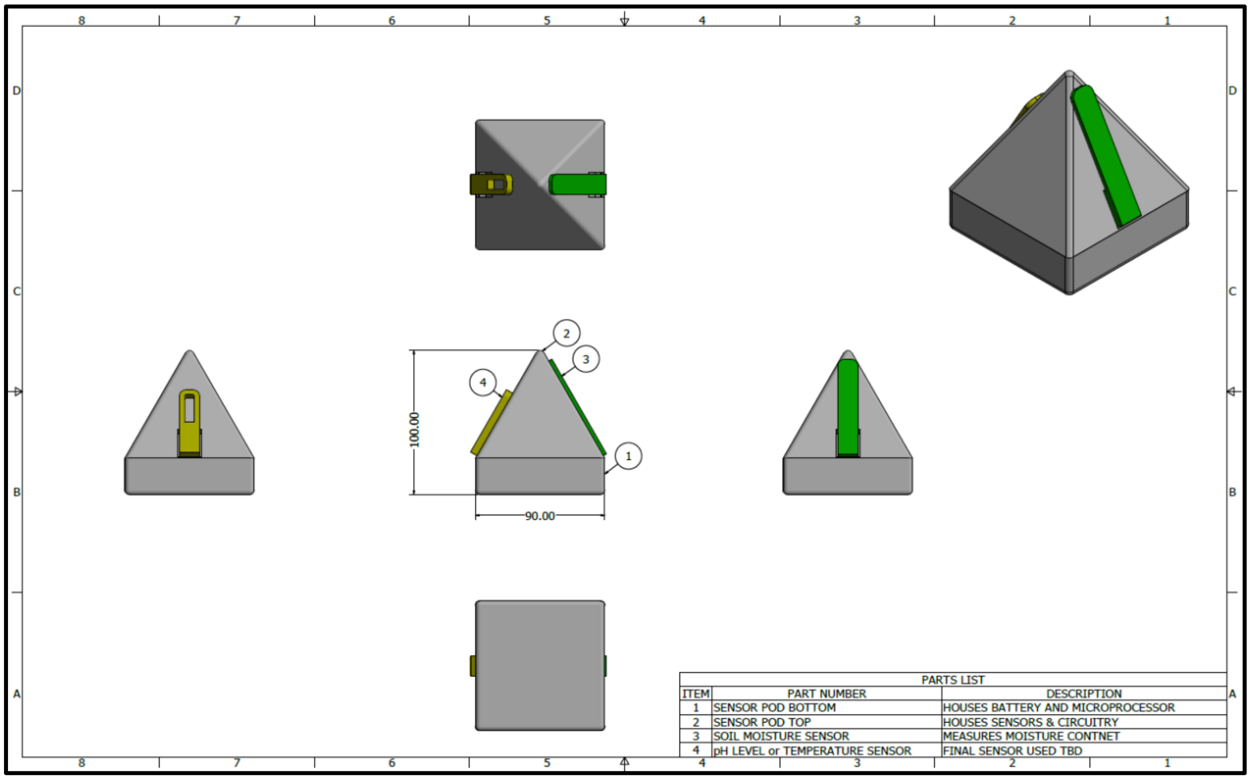
76.22 x 14 x 7 mm. The other sensor shown represents an electrochemical sensor. Both sensors must make contact with the soil in order to obtain readings. The minimum size of each bottom side of the pod can be is dependent on the size of the sensor connector. The top of the pod is a dome because it is an optimal shape for a cylindrical battery. In the square perimeter of the dome will be housed the battery and microcontroller.

6.2 Second Design Iteration

After taking a closer look at the first design iteration, it appeared that configuring the sensors on the pod in this manner made them more susceptible to damage. Whether it be from storing, planting, or using, the sensors themselves could be broken by a force exerted on them, or worse, the connector could be broken, damaging the seal, and allowing water to get into the pod and ruining the components.

Another problem with the first design iteration is that the pod was so small that if a second battery were needed, it would not fit inside the pod. At the time of creating this design, the current and voltage draw for the components was unknown. Now that the current and voltage draw of the components and transmissions are known, adding a second battery is necessary for the design.

To prevent external components from being damaged and to allow space for a second battery, a second design iteration was completed and can be seen in Figure 131.



PARTS LIST

ITEM	PART NUMBER	DESCRIPTION
1	SENSOR POD BOTTOM	HOUSES BATTERY AND MICROPROCESSOR
2	SENSOR POD TOP	HOUSES SENSORS & CIRCUITRY
3	SOIL MOISTURE SENSOR	MEASURES MOISTURE CONTNET
4	pH LEVEL or TEMPERATURE SENSOR	FINAL SENSOR USED TBD

Figure 131: Phase 2 Mechanical Sketch of Sensor Pod.

The physical sizes of the electronics are listed in Table 44.

Table 44: Electronic Component Dimensions.

	MFG	**Length (mm)**	**Width (mm)**	**Height (mm)**
Soil Moisture Sensor	101020614	76.22	14	7
Battery	623360	61	36	5.7
PCB	TBD	TBD	TBD	0.78
Microcontroller	PIC24FJ256GB410	10	10	1.1
LoRaWAN Module	RN2903	26.67	17.78	3.34
Antenna	TBD	82	-	-

The batteries will be located on the bottom of the pod to make the pod bottom-heavy so that the orientation of the pod when it lands will be approximately the same every time. The minimum height for the bottom of the pod was calculated in Equation 23.

$$h_{min} = 2h_{battery} + h_{PCB} + h_{LoRaWAN} = 2(5.7) + 0.78 + 3.34 = 15.52\ [mm] \tag{23}$$

Given that the minimum height of the bottom is 15.52 mm, the top of the pod can be no larger than 84.48 mm.

While it is necessary to protect external circuitry, it is also important to maximize the area inside the housing to allow for additional component design. The soil moisture sensor is 76.22 mm long, and is resting atop a triangular plane that is angled toward the center of the pod, depicted in Figure 132.

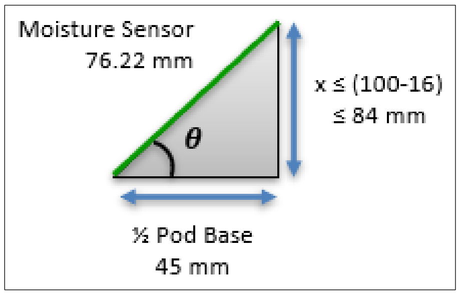


Figure 132: Top Housing Area and Height Requirements.

To allow for a symmetric design, the angle of the sensor is limited to:

$$\theta = cos^{-1}\left[\frac{45}{76.22}\right] = 53.815° \tag{24}$$

By using the angle found in Equation 24, the minimum height of the top of the pod is calculated as:

$$x = 45\tan(\theta) = 61.5\ [mm] \tag{25}$$

Therefore, the height for the top of the pod must be between 61.5 mm and 84 mm.

The optimal volume of the pod would be to have minimum distance for the top housing of the pod and maximum distance for the bottom housing of the pod because the area of a square is greater than the area of a triangle. However, if the top height is set to a minimum of 61.5 mm, the pod would not completely protect the sensor. By examining the orthogonal view and the right-side view of Figure 131, it can be seen that the top of the pod comes to a point. If the height were chosen such that the top of the sensor touched the top of the pod, the sensor would be hanging over the sides of the pod. For this reason, an additional 16.5 mm was added to the height of the top housing, resulting in a minimum height of 78 mm. To satisfy the 100 mm height requirement, the height for the bottom housing became 22 mm.

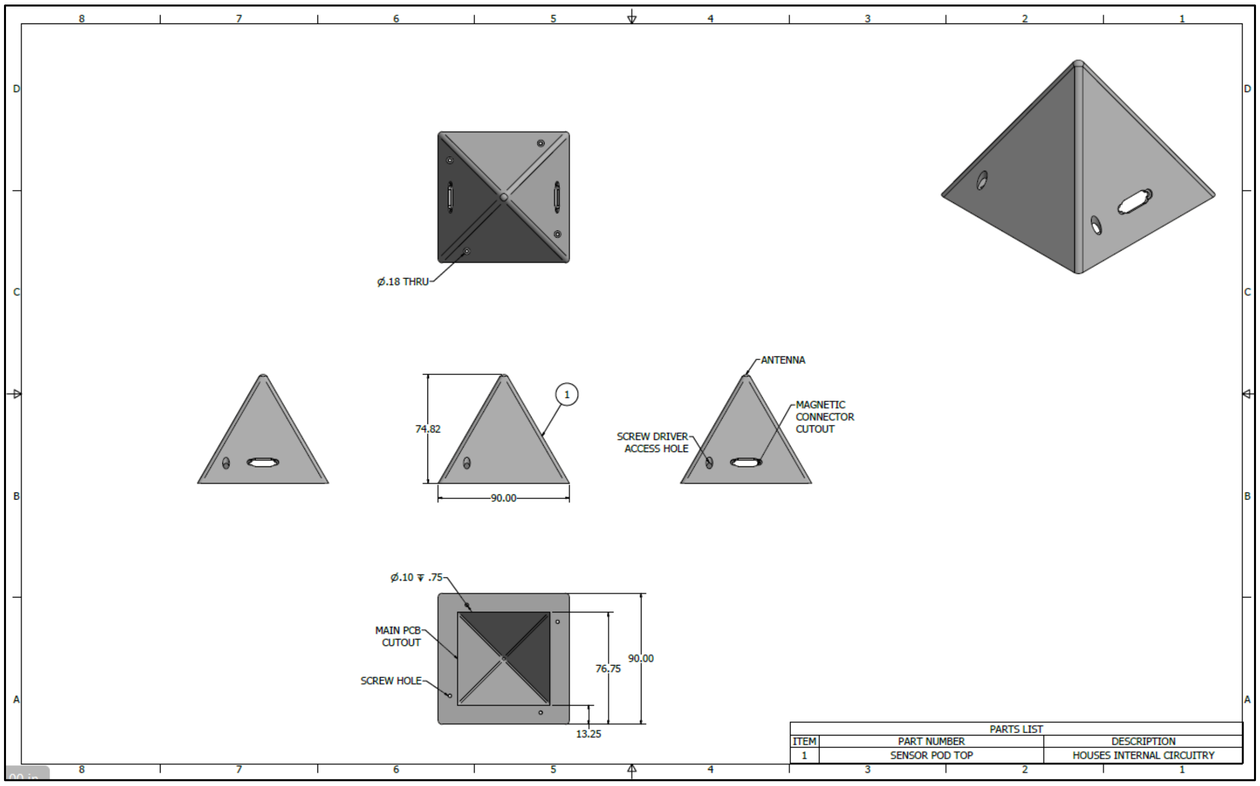
It has not yet been determined if the antenna will be enclosed in the housing or if it will be externally attached to the top of the pod. Given the frequency range of 915 MHz, the length if the antenna is calculated as follows:

$$\lambda_{1/4} = \left(\frac{1}{4}\right)\frac{c}{f} = \left(\frac{1}{4}\right)\frac{3*10^{8}}{915*10^{6}} = 82\ mm \qquad (26)$$

Assuming a quarter wavelength antenna to optimize the power consumption in relation to range, the antenna chosen will be 82 mm in length. Therefore, the antenna will fit inside the housing, and the placement will depend on the arrangement of circuitry.

6.3 Third Design Iteration

The Sensor Pod from the second design iteration was further analyzed and detailed to match specific components purchased for the project. A breakdown of the third iteration designs can be seen in Figures 133-137.



PARTS LIST		
ITEM	PART NUMBER	DESCRIPTION
1	SENSOR POD TOP	HOUSES INTERNAL CIRCUITRY

Figure 133: Phase 3 Mechanical Sketch: Top Housing.

The top of the Sensor Pod was revised to have more accurate cutouts. The first cutout, found on the side, was reconfigured to match the exact dimensions of the magnetic pogo-pin connectors used for quick disconnect of sensors. The holes on the bottom of the top housing were created as a means of connecting the top housing to the base through the use of screws. The round cutouts on all four side faces of the top housing are the size of a terminal screwdriver so that the screws can be inserted through the pod and secured to the base. Plugs can be inserted into these holes in order to make the Sensor Pod water tight. A hole was also drilled into the top of the shell in order to allow room for the antenna.

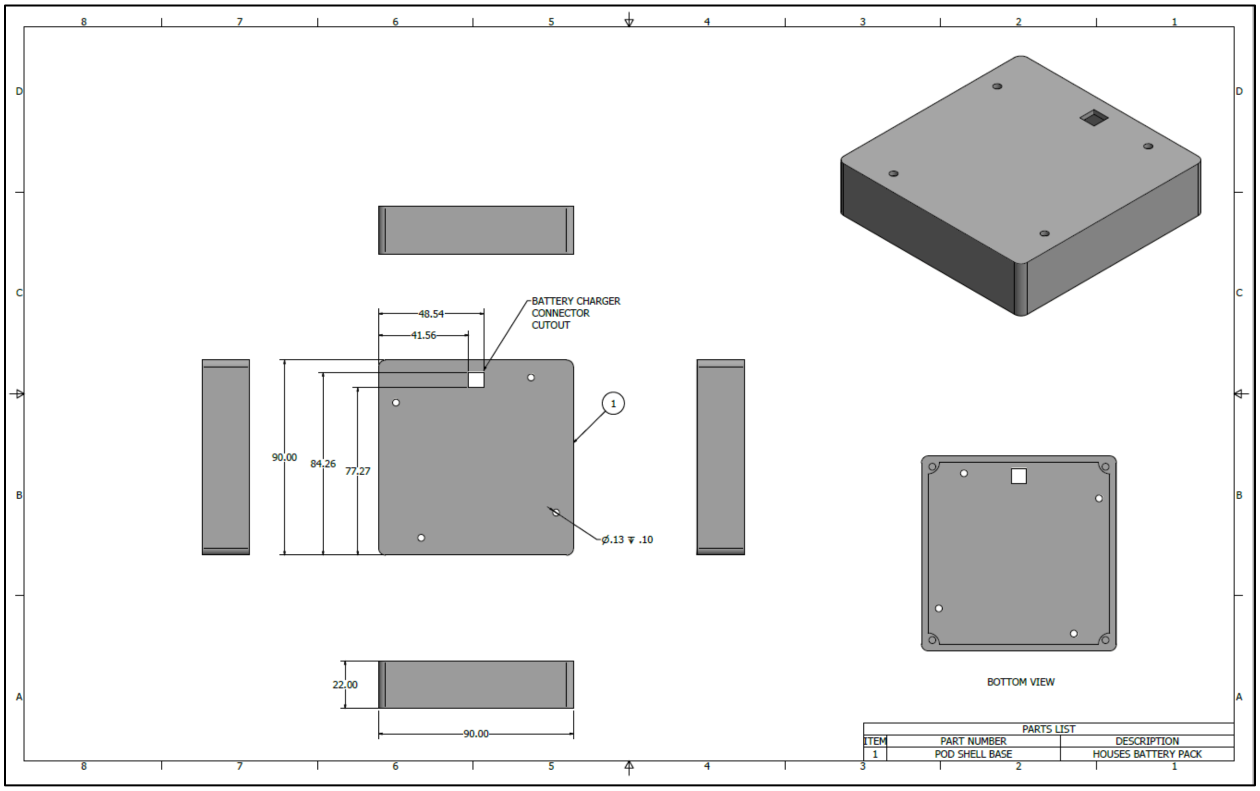
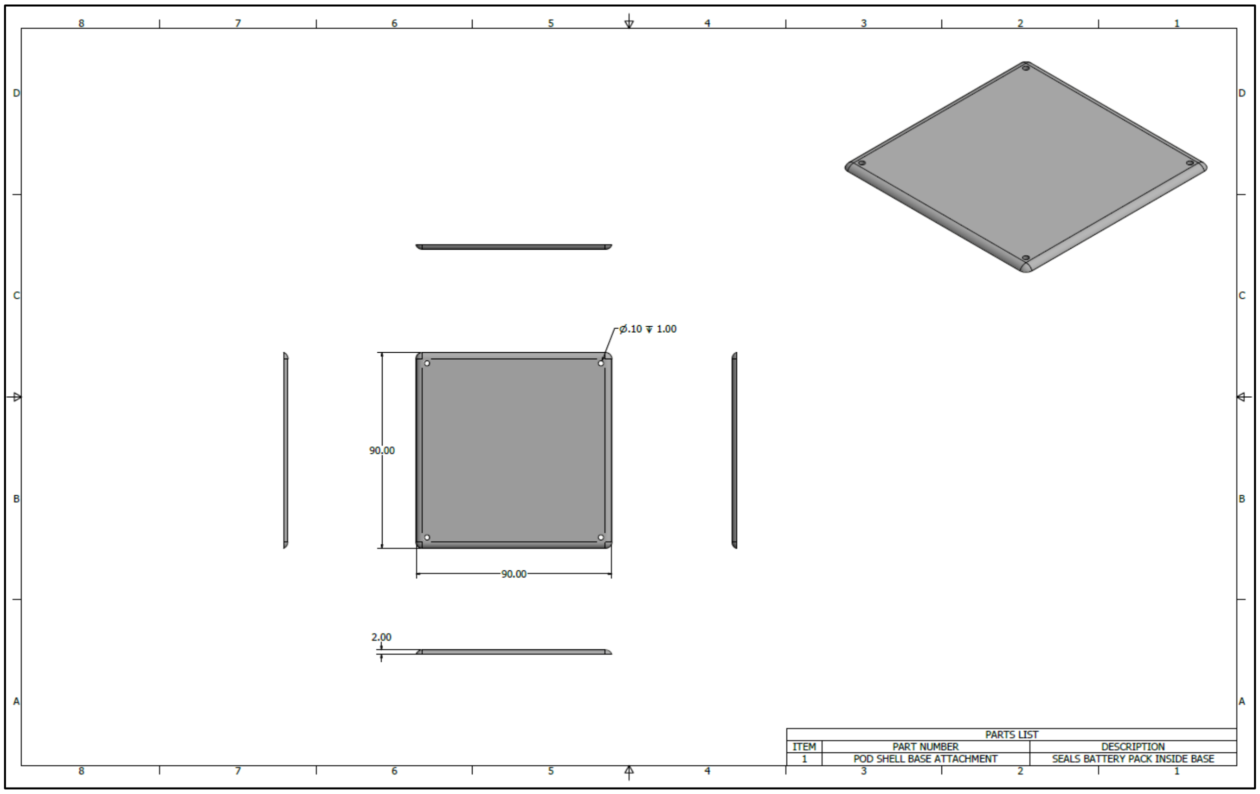


Figure 134: Phase 3 Mechanical Sketch: Base.

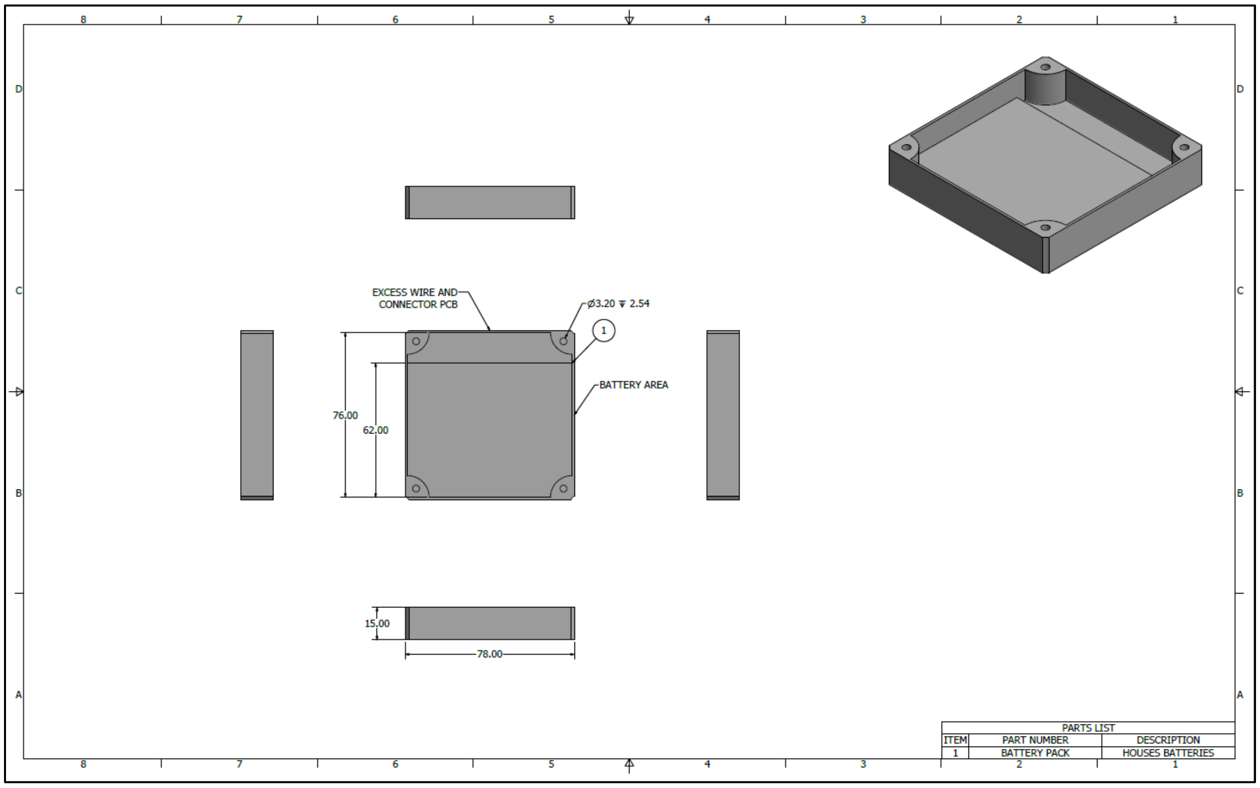
Screw holes were added to the top of the Sensor Pod base to connect to the top housing. In addition to this, a square cutout was added in order to have an access point for the batteries to connect to the main PCB. The battery connector cutout aligns with the connector cutout on the battery pack, as will be seen in Figure 137. The Sensor pod must be water-tight; however, the farmer should also be able to easily access the batteries to recharge them. Because of this, it was determined that the best way to seal the base while still giving the farmer easy access would be to use four screws on the bottom of the base. The base lid is seen in Figure 135.



PARTS LIST		
ITEM	PART NUMBER	DESCRIPTION
1	POD SHELL BASE ATTACHMENT	SEALS BATTERY PACK INSIDE BASE

Figure 135: Phase 3 Mechanical Sketch: Base Attachment.

The primary purpose of the base attachment is to secure the battery pack while granting easy access to the farmer.

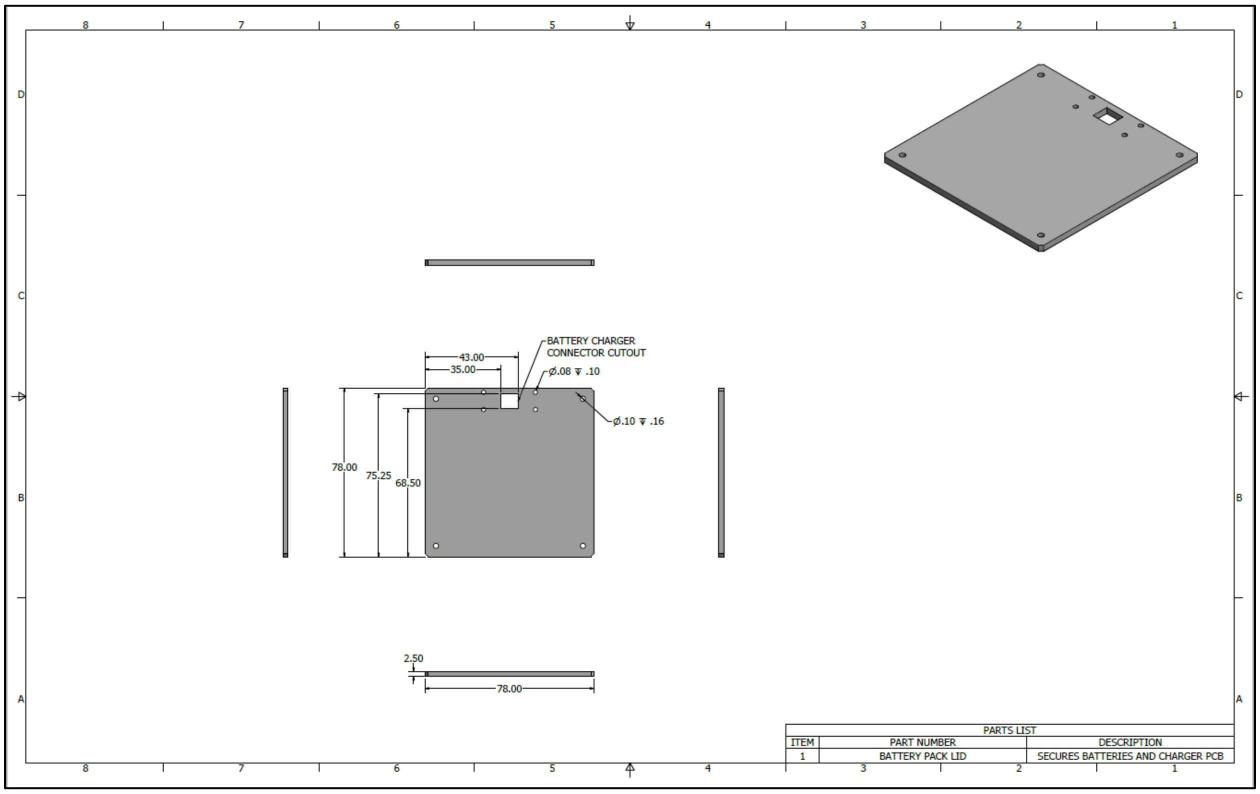


PARTS LIST		
ITEM	PART NUMBER	DESCRIPTION
1	BATTERY PACK	HOUSES BATTERIES

Figure 136: Phase 3 Mechanical Sketch: Battery Pack.

Each Sensor Pod requires two batteries to operate for an entire growing season. As stated previously, the batteries are located at the bottom of the pod to make the Sensor Pod bottom-heavy so it will fall correctly out of the planter during automatic installation. The batteries are connected in parallel to satisfy the Engineering Requirement of a supplied current time of 5600 mA-H. The area in which the batteries set is the exact dimensions of two batteries; no extra room was left to prevent the batteries from moving around and possibly coming disconnected. The area where the wires set is deeper in dimension to allow the wires to have the correct bend radius so they are less likely to break.

In the case that a battery needs recharged, the farmer should not have to go through the hassle of opening the Sensor Pod circuitry and disconnecting the battery from the main PCB itself. In doing so, the farmer not only will struggle to open the top of the pod, but also has a greater chance of breaking the internal circuit. To avoid such problems, a battery pack, as shown in Figure 137, is located inside of the base. The two batteries are connected in parallel to the battery PCB, which has a quick-disconnect connector on the top half of the board that runs back to the main PCB. The connector PCB is mounted to the lid of the battery pack and aligns with the wireway of the pack.



PARTS LIST		
ITEM	PART NUMBER	DESCRIPTION
1	BATTERY PACK LID	SECURES BATTERIES AND CHARGER PCB

Figure 137: Phase 3 Mechanical Sketch: Battery Pack Lid.

When it is time for the famer to recharge the batteries, he must simply remove the four screws on the base attachment, remove the battery pack, and then connect the battery pack to the charger. It is not even necessary to remove the batteries from the pack; connecting the charger to the external connector will charge both batteries at once since the batteries are connected in parallel. After the batteries have been fully charged, the farmer can reinsert the battery pack into the pod and secure the base attachment to the base by inserting the four screws. The process is a simple recharge without ever having to interact with the Sensor Pod circuitry.

6.4 Pod Shell and Force of Impact

Using basic Dynamics concepts, the material and thickness of the pod shell were determined. The average farm planter travels approximately 5 kph [23]. Through observation it was determined that the largest distance the sensor pod would fall is 1.5 m, corresponding to falling from a potato planter. The force of impact was determined by Equations 27-29, corresponding to velocity, kinetic energy, and impact energy respectively.

$$v = \sqrt{2gh} = \sqrt{2(1.5)(9.81)} = 5.425\ m/s \tag{27}$$

Once the velocity of the pod was determined, the value was inserted into the Equation 28 to determine the kinetic energy of the pod. The mass of the pod was estimated to be 0.1 kg, based off components and pod shell material.

$$T_A = \frac{1}{2}mv^2 = \frac{1}{2}(0.1)(5.425)^2 = 1.5\ J \tag{28}$$

The impact energy equation was used to take into consideration the geometry and energy of the pod shell.

$$U = \frac{\sigma^2 AL}{2E} \tag{29}$$

σ is the minimum yield strength of the material, A is the area of the pod shell, L is the length of the shell. E corresponds to the material energy, as calculated in Equation 30.

$$E = \frac{stress}{strain} = \frac{2.3*10^6}{0.015} = 1.5*10^9 \tag{30}$$

The pod shell prototypes will be 3D printed. The two types of materials the University of Akron Electrical Engineering 3D printers can print is ABS plastic and PLA plastic. By examining the mechanical composition of each plastic, it was seen that ABS plastic had a higher yield strength, and therefore was more durable than PLA plastic [24]. Through this analysis, it was determined that ABS plastic would be used for the sensor pod shell prototype.

The length of the pod shell was divided into two components: the base of the pod (square) and the top housing of the pod (triangle). Although the total height of the pod is 100mm, this does not take into consideration the length of the hypotenuse in Figure132. The total length was determined to be the length of the base plus the length of the top housing, which is half the length of the base because the top housing is a 45º right triangle.

Rearranging Equation 30 to solve for area, the minimum thickness of the pod shell can be determined.

$$A = \frac{2UE}{\sigma^2 L} = \frac{2(1.5)(1.5*10^9)}{(29.6*10^6)^2\left(0.05+\frac{1}{2}0.05\right)} = 684.8 * 10^{-6}\ m \qquad (31)$$

The impact energy U was defined to be equivalent to the kinetic energy found in Equation 28, because if the impact energy was less than the kinetic energy, the pod shell would not be able to withstand the force of impact, which would result in the pod shattering upon impact. Note that Equation 31 is the area of the entire pod shell, it does not say what the thickness is of an individual side.

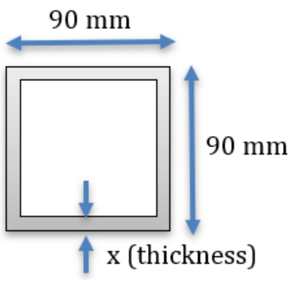


Figure 138: Pod Shell Geometry.

To find the thickness of an individual side, as referenced in Figure 76, the geometry of the shell base was used to construct Equation 32.

$$A = (2x)^2 \quad \rightarrow \quad x = \frac{\sqrt{A}}{2^2} = \frac{\sqrt{68.5}}{4} = 2.069 * 10^{-3}\ m \tag{32}$$

The minimum thickness the ABS plastic pod shell can be without shattering upon impact is 2 mm thick.

6.5 Sensor Pod Prototypes

Once the design iterations were complete, the 3D models were printed using ABS plastic. The prototypes for Pod 1 are shown in Figures 139-141.

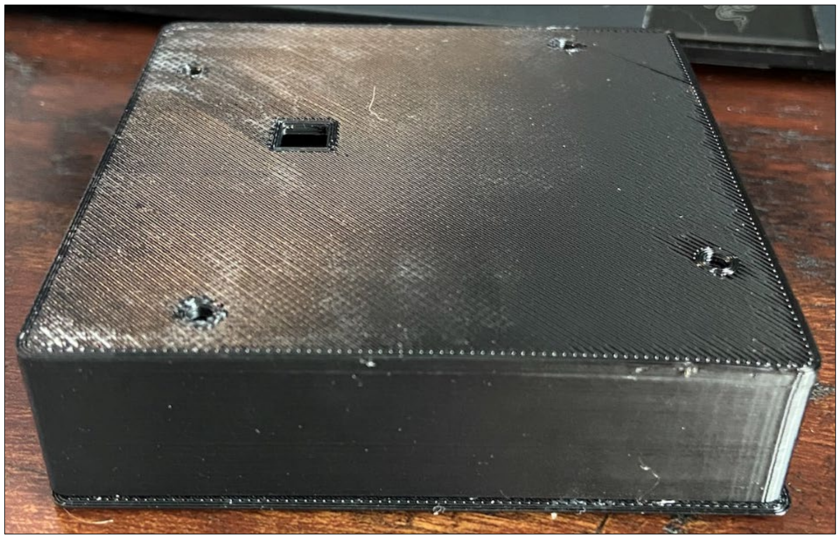


Figure 139: 3D Prototype: Pod Base.

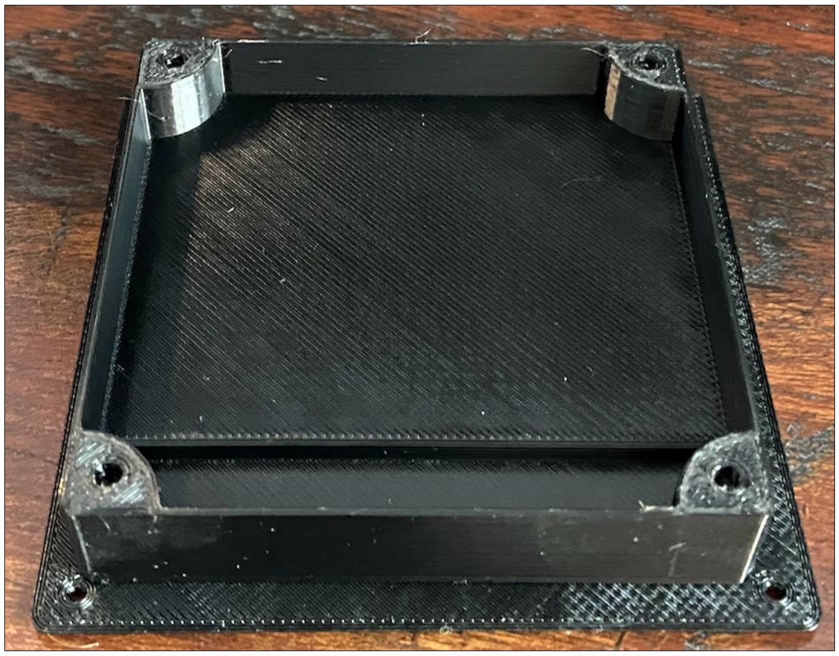


Figure 140: 3D Prototype: Battery Pack.

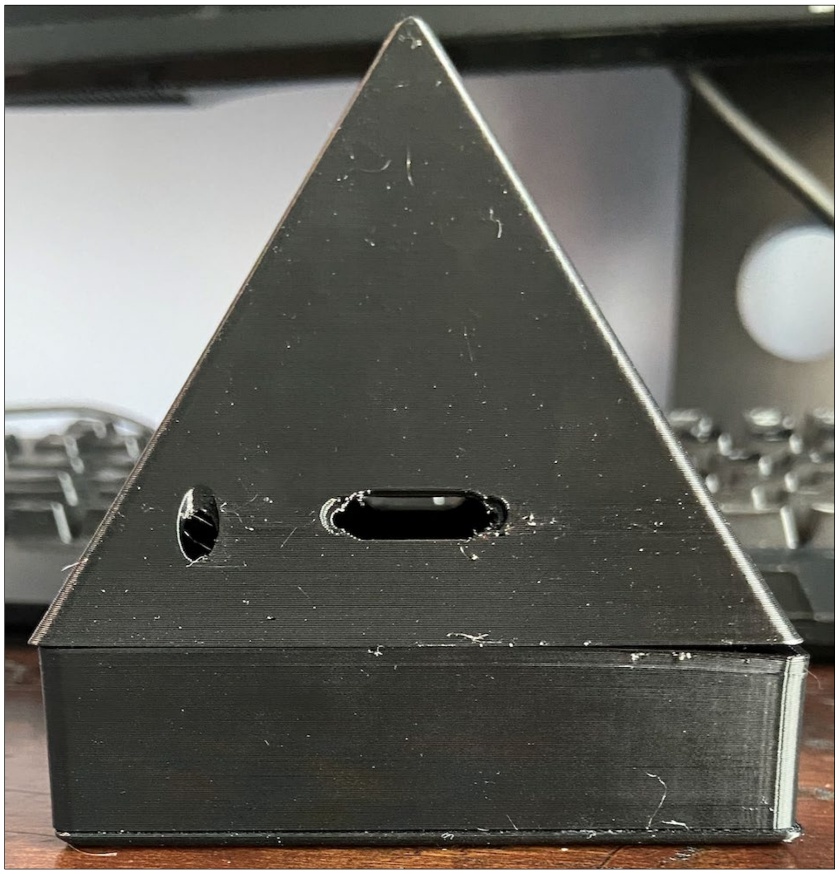


Figure 141: 3D Prototype: Top Housing w/ Base.

A second set of prototypes for Pod 2 was created. The fully assembled Sensor Pod can be seen in Figure 142.

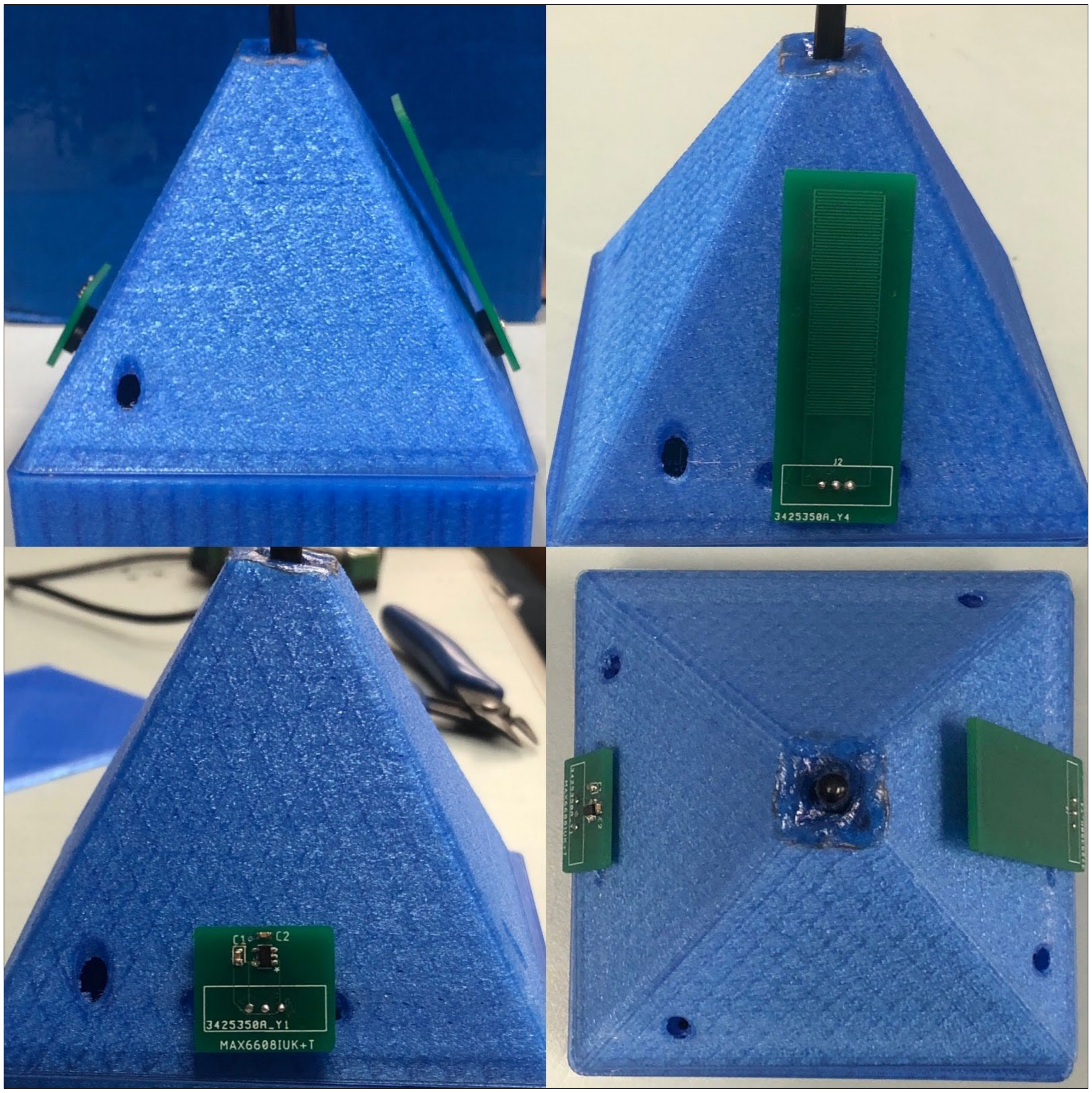


Figure 142: Fully Assembled Sensor Pod.

The battery pack of Pod 2 is located in Figure 143. The batteries attach to the connector PCB shown.

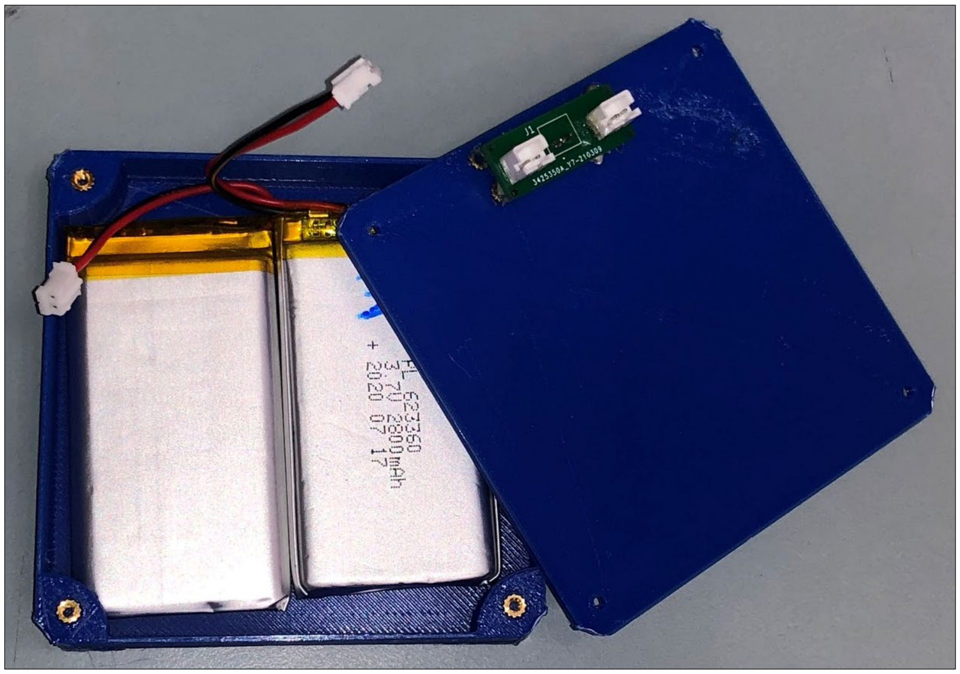


Figure 143: Assembled Battery Pack.

A third protype of the same design was created. The three Sensor Pods are seen in Figure 144.

The middle pod displays the PCB, quick disconnect, and sensor designs.

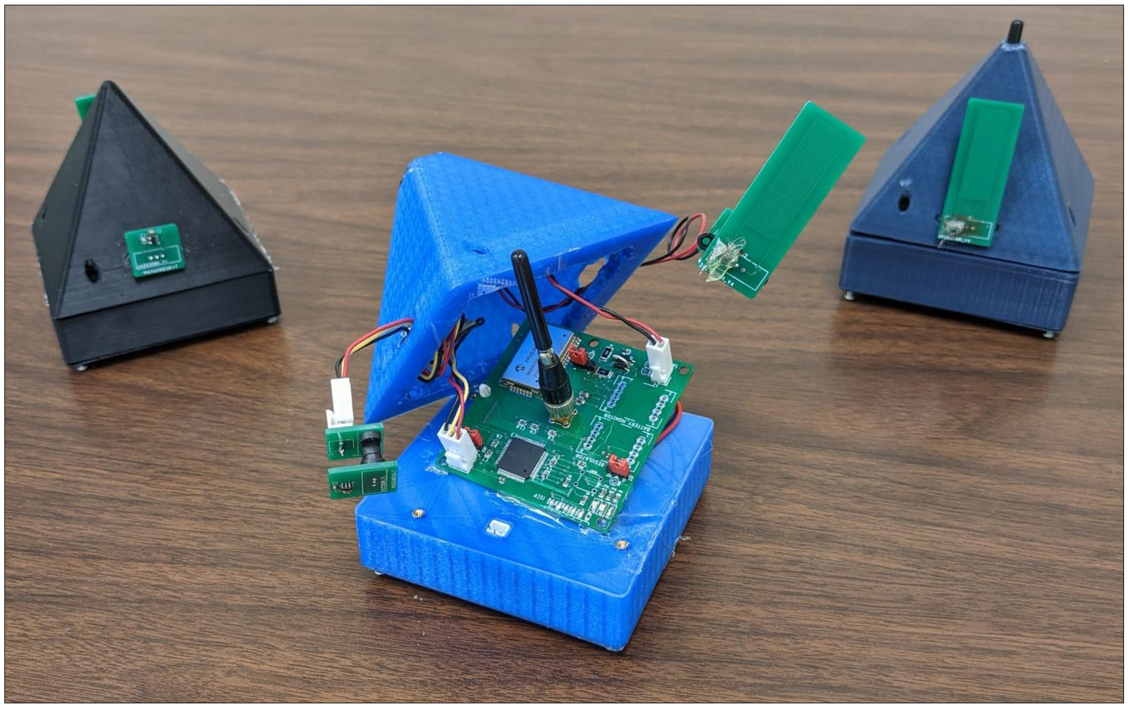


Figure 144: Sensor Pod Prototypes.

7 FUTURE IMPLEMENTATION

Apart from fully programming and utilizing the battery monitoring system, the processes for installation and retrieval were also analyzed and designed for future implementation.

7.1 AUTOMATED INSTALLATION AW

Wired soil sensors currently exist in farm fields. It takes many hours to manually install all of the sensors into the soil in all the fields a farmer owns. To reduce the labor as well as the time it takes for install, the Sensor Pods can be attached to the inside of a planter for automatic installation. The Sensor Pods can be placed into a tube, similar to a PVC pipe, with 4 slits cut out for brackets to hold the sensor pods in place within the tube. A half-view of the setup is seen in Figure 145.

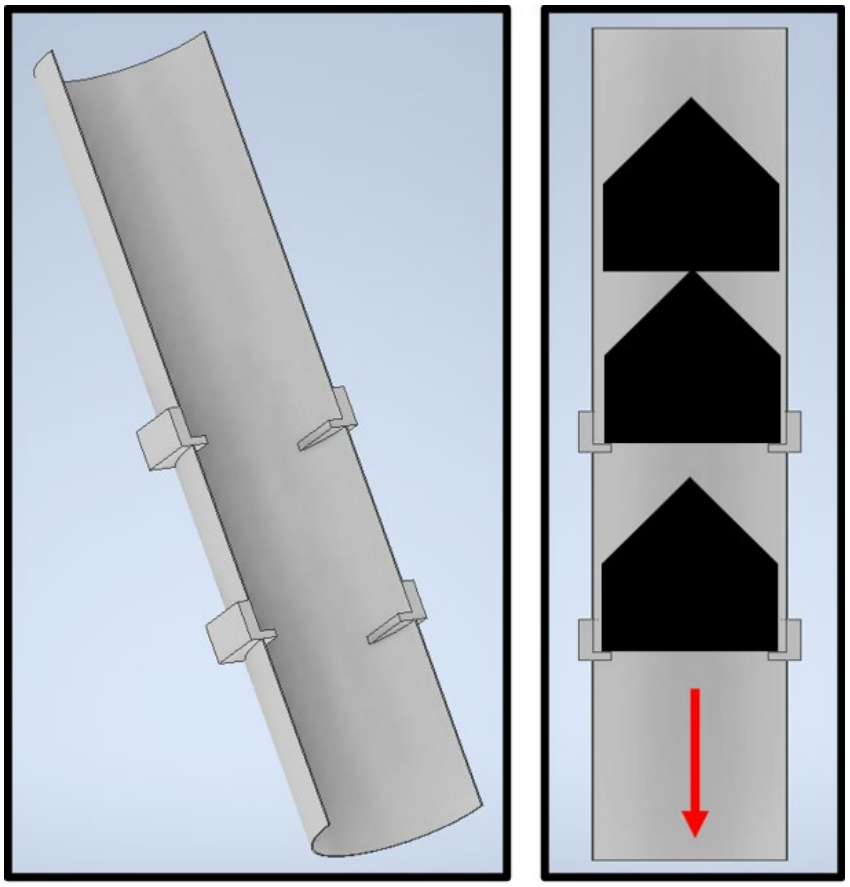


Figure 145: Automated Installation Contraption.

The bottom two brackets are in parallel to hold the sensor about to be planted into place. The top two brackets are inserted to hold the remaining pods in place, separating them from the bottom sensor. Each bracket will be connected to a linear cylinder, and each bracket set will be connected to an electric solenoid. The two solenoids will work independently from one another. The automated installation process works as follows.

Both solenoids start in a closed state, as seen in the figure. The controls of the installation contraption will be connected to the odometer of the planter. Once the planter travels x-number of meters, the lower solenoid is activated, opening the bottom two cylinders, and releasing the bottom pod. Once the pod is released, the bottom cylinders close. The upper solenoid is then activated, opening the top set of brackets to allow the Sensor Pods to shift down to the bottom brackets. The upper solenoid is then deactivated, closing the top set of cylinders and brackets. The planter travels x-number of meters, and the process begins again until every Sensor Pod is installed. The Sensor Pods are released and buried with the seeds in order to ensure the pods will be planted where the roots of the seeds will grow.

7.2 Retrieval Process LF, AW

A few different types of retrieval processes were analyzed but never implemented. The first is to use the LoRa transceiver circuitry. On the instance the that the Sensor Pod voltage goes below a certain threshold and it is determined that the pod will not last and entire growing season, the Sensor pod will alert the farmer that the battery is low and give the farmer the option to begin the Retrieval Process. The farmer would have a retrieval device that contains a second LoRa transceiver that could be taken into the field. Once in the field, the farmer will start the Retrieval Process and drive the LoRa receiving unit to the Sensor Pod that has the almost dead battery. The module in the Sensor Pod would transmit a constant signal for 5 minutes so that the farmer can locate and retrieve it. The only problem using this method is that it would be an extra cost to the farmer since the average person does not have a LoRa receiving device laying around their house.

A more cost-effective approach for the farmer would be to use the LoRa module with an AM radio. LoRa communicates on the 615 MHz US frequency band. The AM radio frequency range is between 550-1720 kHz. In order to reduce the LoRa frequency to the frequency that can be received by an AM radio, a class AB power de-amplifier circuit would be added to the main PCB. A power amplifier will alter the output voltage and current from the LoRa module to match the desired input voltage and current to the AM radio. The power outputted from the module is 18mW. Once the desired receive frequency is determined, the power inputted to the radio can be calculated. When this value is calculated, the power from the module can be divided by the power from the radio to determine the circuit gain of the amplifier.

A disadvantage to the above two proposed processes is that LoRa transmission consumes a significant amount of power, as explained in § 5.4.1.2. The retrieval process is activated because the battery is almost depleted. Using the LoRa module for transmission, and allowing it to transmit consistently for 5 minutes, will consume a lot of power. The farmer would have to retrieve the pod while the pod had enough battery left for the transmission to be possible because once the battery is depleted, the pod would no longer be able to transmit a signal to be retrieved.

The last, and most practical, process that was analyzed was to use a GPS tracker. The manufactured tracking device can be placed on the bottom side of the main PCB and connected to the farmer's phone through an app. An example of a practical GPS tracker that can be used is the Nano Hornet GPS Tracker seen in Figure 146.

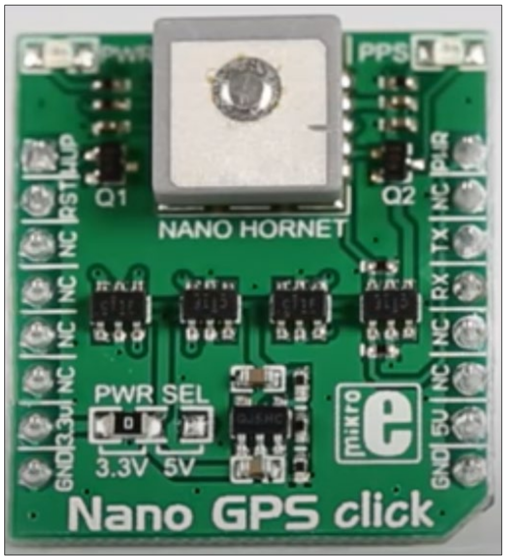


Figure 146: MN5D10HS Nano Hornet GPS Tracker. Image retrieved from https://trackimo.com/micro-gps-tracking-chips/.

The OriginGPS tracking device has a low transmission power and high reading accuracy. It also has dimensions of 10x10x3.8mm, which would easily fit onto the main PCB. The device would be fully integrated into the circuit so that it can be activated when the battery retrieval process begins.

8 DESIGN TEAM INFORMATION RK, AW

The Soil Sensor Network team consists of two electrical and two computer engineers. Below is a list of the members on the team.

Aléxis Alves, Computer Engineering. ESI (Y)

Luke Farnsworth, Electrical Engineering. ESI (N)

Ross Klonowski, Computer Engineering. ESI (Y)

Andrea Wyder, Electrical Engineering. ESI (N)

Ross Klonowski is the project manager, Luke Farnsworth is the hardware manager, Aléxis Alves is the software manager, and Andrea Wyder is the archivist.

9 Parts List AW

9.1 Schematics Parts List

Once the compiled set of schematics was created in EagleCAD, a Bill of Materials (BOM) was exported and formatted, as seen in Table 45.

Table 45: Design Schematics Parts List.

Refdes	Part Num.	Storage	Description
Battery			
BAT1	623360	3.7V, 2800mA-H	lithium-ion battery
Voltage Regulator			
IC1	XC9140A331MR-G	3.3V	step-down regulator
C1	UA stock	4.7uF	capacitor
C2	UA stock	10uF	capacitor
L1	UA stock	4.4uH	inductor
Microcontroller			
C6-C13	UA stock	0.1uF	capacitor (ceramic if possible)
C14	UA stock	10uF	capacitor (tantalum/ceramic if possible)
R3	UA stock	10kΩ	resistor
R4	UA stock	100Ω	resistor
X1	UA stock	RJ-11	PCB connector
Temperature Sensor			
IC4	MAX6607IXK+T	3.3V	temperature sensor
C15	UA stock	1nF	capacitor
C16	UA stock	0.2uF	capacitor
Soil Moisture Sensor			
R1	UA stock	150kΩ	resistor
R2	UA stock	3.3MΩ	resistor
C3	UA stock	10pF	capacitor
C4	UA stock	10nF	capacitor
C5	UA stock	10nF	capacitor
-	UA stock	20pF	capacitor
U1	UA stock	TLC555	timer

LoRa Module			
IC2	RN2903	3.3V	LoRa Module
ANT1	RSRA6982700SSM	3.3V	Antenna
-	1825910-6	-	Reset Switch

The parts were chosen based off the Engineering Analysis and Accepted Technical Design sections. The RefDes column indicates the referenced part in the schematics section, Part Num. indicates the manufacturer part number, Storage indicates the value of the component, and Description indicates the type of device.

The next semester the Sensor Pods were reconfigured for the implementation stage, and the new Parts List for the PCBs and their components is shown in Table 46.

Table 46: Implementation Schematics Parts List.

Refdes	Frame	Part Num.	Description	Case on PCB
SENSOR POD SHELL				
N/A	Pod Shell	310889508	#2-56 x 1/4 in. Phillips Machine Screws	N/A
N/A	PCBs	G01K	PCB Foot Mounts	N/A
MAIN PCB COMPONENTS				
C1	555 Timer	TMK107BBJ106MA-T	10 uF capacitor	0603 (1608)
C2	555 Timer	TMK063B7103KP-F	0.01uF capacitor	0603 (1608)
C3	PIC24	TMK107BJ104KAHT	0.1 uF capacitor	0603 (1608)
C4	PIC24	TMK107BJ104KAHT	0.1 uF capacitor	0603 (1608)
C6	PIC24	TMK107BJ104KAHT	0.1 uF capacitor	0603 (1608)
C7	PIC24	TMK107BJ104KAHT	0.1 uF capacitor	0603 (1608)
C8	PIC24	TMK107BJ104KAHT	0.1 uF capacitor	0603 (1608)
C9	PIC24	TMK107BJ104KAHT	0.1 uF capacitor	0603 (1608)
C10	PIC24	TMK107BJ104KAHT	0.1 uF capacitor	0603 (1608)
C11	Reset	TMK107BBJ106MA-T	10 uF capacitor	0603 (1608)
C12	PIC24	TMK107BBJ106MA-T	10 uF capacitor	0603 (1608)
R3	PIC24	RMCF2010JT3M30	4.7k ohm resistor	2010
R4	555 Timer	RMCF2010JT3M30	3.3M ohm resistor	2010
R5	555 Timer	RMCF0805JT150KTR-ND	150k ohm resistor	2010
R6	Reset	RMCF0805JT10K0	10k ohm resistor	8050
R7	Reset	RMCF0805JT470R	470 ohm resistor	8050
R9	LED	RMCF0805JT1K00	1k ohm resistor	8050
R10	LED	RMCF0805JT1K00	1k ohm resistor	8050
R11	LED	RMCF0805JT1K00	1k ohm resistor	8050

R15	Battery Monitor	RMCF0805JT4K70	4.7k ohm resistor	8050
R16	Battery Monitor	RMCF0805JT4K70	4.7k ohm resistor	8050
D1	LED	AP2012EC	LED Lights Red	CHIP-LED0805
D2	LED	17-21SYGC/S530-E2/TR8	LED Lights Green	CHIP-LED0805
D3	LED	APT2012NW	LED Lights Orange	CHIP-LED0805
JP1	Jumpers	N/A	3 PIN Header	N/A
JP2	Jumpers	N/A	3 PIN Header	N/A
JP3	Jumpers	N/A	3 PIN Header	N/A
J1	Regulator	B02B-PASK(LF)(SN)	2 PIN Header	N/A
J2	555 Timer	CONN02SMLPCB/*FLPCB	2-pin connector male/female	N/A
J3	Temp Sensor	CONN03SMLPCB/*FLPCB	3-pin connector male/female	N/A
IC2	RN2903	RN2903A-I/RM103-ND	LoRa Module	N/A
IC3	PIC24	PIC24FJ256GB410-I/PT	Microprocessor	N/A
U1	555 Timer	TLC555QDRQ1	TLC555 timer	N/A
SW1	Reset	1825910-6	Reset switch	N/A
VOLTAGE REGULATOR PCB COMPONENTS				
C14	Regulator	TMK107BBJ106MA-T	10 uF capacitor	0603 (1608)
C15	Regulator	AMK107BBJ226MA-T	22 uF capacitor	0603 (1608)
R12	Regulator	RMCF0805JT10K0	10k ohm resistor	8050
R13	Regulator	RMCF0805JT560K	560k ohm resistor	8050
R14	Regulator	RMCF0805JT100K	100k ohm resistor	8050
L1	Regulator	VLCF4020T-2R2N1R7	2.2 uH shielded inductor	N/A
IC1	Regulator	STBB1-APUR	2.0 V - 5.5 V IC buck boost regulator	DFN10 (3 x 3 mm)
TEMPERATURE SENSOR PCB COMPONENTS				
C1	Temp Sensor	TMK107BBJ106MA-T	10 uF capacitor	0603 (1608)
C2	Temp Sensor	0805B204K500CT	0.2uF capacitor	0603 (1608)
U1	Temp Sensor	MAX6608IUK+T	temperature sensor	0805 (2012)
BATTERY MONITORING PCB COMPONENTS				
C5	Battery Monitor	EMF107B7105MAHT	1 uF capacitor	0603 (1608)
C13	Battery Monitor	EMF107B7224MAHT	0.22 uF capacitor	0603 (1608)
R1	Battery Monitor	RL1220T-R010-J	10m ohm resistor	8050
R2	Battery Monitor	RMCF0805JT200K	200k ohm resistor	8050
R3	Battery Monitor	RMCF0805JT1K00	1k ohm resistor	8050
U1	Battery Monitor	STC3100IST	IC Battery Monitoring	8-TSSOP
CONNECTOR PCB COMPONENTS				
J2	Moisture Sensor	CONN02SMLPCB/*FLPCB	2-pin connector male/female	N/A
J3	Temp Sensor	CONN03SMLPCB/*FLPCB	3-pin connector male/female	N/A

The parts were chosen based off the Accepted Technical Design as well as the size constraints of the PCB in order to have it fit into the Sensor Pod.

9.2 MATERIALS BUDGET LIST AW

Once the schematics were created and a BOM was exported, a Materials Budget List was created to keep track of what has been purchased. The information on the BOM was inserted in columns 3, 4, and 5 of the Material Budget List, as seen in Table 47.

Table 47: Material Budget List Fall Semester.

Order Form	Qty	Ref des	Part Num.	Description	Suggested Vendor	Vendor Part Num.	Cost	Total Cost
RESEARCH & DEVELOPMENT								
1	1	-	101020614	Soil Moisture Sensor	Digikey	101020614	$6.07	$6.07
1	2	(IC3)	DM164139	Development Boards	Digikey	DM164139-ND	$71.39	$142.78
2	1	-	PIC24FJ256GB410	Development Board Accessory: Plug-In Module (PIM)	Microchip	MA240038	$25.00	$25.00
3	1	-	1110748	Development Board Accessory: Socket Adapter	Digikey	A800AR-ND	$11.44	$11.44
POD INTERIOR								
1	8	IC4	MAX6607IXK+T	Temperature sensor	Digikey	MAX6607IXK+T	$1.70	$13.60
1	8	BAT1	623360	Battery (Out of Stock on Amazon)	Amazon	623360	$7.47	$59.76
1	2	-	ADA1904	Battery Charger	Amazon	ADA1904	$9.33	$18.66
SOIL MOISTURE SENSOR								
3	4	R1	-	150K ohm resistor	UA stock	-	$ -	$ -
3	4	R2	-	3.3M ohm resistor	UA stock	-	$ -	$ -
3	4	C3	-	10pF capacitor	UA stock	-	$ -	$ -
3	4	C4	-	10nF capacitor	UA stock	-	$ -	$ -
3	4	C5	-	10nF capacitor	UA stock	-	$ -	$ -
3	4	-	-	20pF capacitor	UA stock	-	$ -	$ -
3	4	U1	-	TLC555 timer	UA stock	-	$ -	$ -
VOLTAGE REGULATOR CIRCUIT								
4	4	C1	-	4.7 uF capacitor	UA stock	-	$ -	$ -
4	4	C2	-	10 uF capacitor	UA stock	-	$ -	$ -

4	4	L1	-	4.4 uH inductor	UA stock	-	$ -	$ -
4	4	IC1	XC9140A331MR-G	Step-down regulator	Digikey	893-1180-1-ND - Cut Tape (CT)	$1.03	$4.12
POD EXTERIOR								
1	2	ANT1	RSRA6982700SSM	Antenna	Arcantenna	RSRA698/2700SSM	$15.00	$30.00
PROGRAMMABLE DEVICES								
LoRa MODULE								
1	2	IC2	RN2903	LoRa Module	Digikey	RN2903A-I/RM103-ND	$12.80	$25.60
4	4	-	1825910-6	Reset switch	Digikey	450-1650-ND	$0.10	$0.40
MICROCONTROLLER								
4	32	C6-C13	-	0.1 uF capacitor (ceramic if possible)	UA stock	-	$ -	$ -
4	4	C14	-	10 uF capacitor (tantalum if possible)	UA stock	-	$ -	$ -
4	4	R3	-	10k ohm resistor	UA stock	-	$ -	$ -
4	4	R4	-	100 ohm resistor	UA stock	-	$ -	$ -
4	4	X1	-	RJ-11 PCB connector	UA stock	-	$ -	$ -
						Total Spent		**$337.43**
						Development Costs		**$185.29**
						Budget Spent		**$152.14**
						Budget		**$600**
						Budget Remaining		**$447.86**

Multiple Parts Request Forms (Order Forms) make up the Material Budget List. The first column of Table 46 displays on which form each part was ordered, and column 2 displays how many were ordered.

For the final design, five sensor pods were constructed. To avoid backordered parts, five of every part was ordered in the design phase so that each pod would have all of its components beginning the implementation phase. Once the difficulty of soldering microcircuitry was experienced, extra micro components were ordered in case of damage during construction and testing. A few components exist on the Material Budget List that do not exist on the BOM because they are development components that were used this design phase but were not implemented in the final design. The final design budget can be found in Table 48.

Table 48: Material Budget List Spring Semester.

Order Form	Qty.	Refdes	Part Num.	Description	Suggested Vendor	Vendor Part Num.	Cost	Total Cost
VOLTAGE REGULATOR PCB								
6	4	IC1	MIC59150YME	Voltage Regulator	Digikey	MIC59150YME	$1.88	$7.52
8	1	IC1	MAX763ACSA+	3.3V Step-Down Voltage Regulator	Digikey	MAX763ACSA+-ND	$4.67	$4.67
BUCK-BOOST								
8	7	IC1	STBB1-APUR	2.0 V - 5.5 V IC buck boost regulator	Digikey	497-11971-2-ND	$2.48	$17.36
8	10	L1	VLCF4020T-2R2N1R7	2.2 uH shielded inductor	Mouser	810-VLCF4020T2R2N1R7	$0.47	$4.70
8	5	C14	TMK107BBJ106MA-T	10 uF capacitor	Mouser	963-TMK107BBJ106MA-T	$0.47	$2.35
8	10	C15	AMK107BBJ226MA-T	22 uF capacitor	Mouser	963-AMK107BBJ226MA-T	$0.35	$3.50
8	10	R12	RMCF0805JT10K0	10k ohm resistor	Digikey	RMCF0805JT10K0CT-ND	$0.20	$2.00
8	10	R13	RMCF0805JT100K	100k ohm resistor	Digikey	RMCF0805JT100KTR-ND	$0.02	$0.20
8	10	R14	RMCF0805JT560K	560k ohm resistor	Digikey	RMCF0805JT560KTR-ND	$2.00	$20.00
MAIN PCB								
7	2	-	R2-56X1/4	Sensor Pod Screws	Mouser	608-R2-56X1/4	$0.57	$1.14
7	100	-	M20X100C	Sensor Pod Screw Connectors	Mouser	761-M20X100C	$0.18	$18.00
7	1	-	G01K	PCB foot mounts	Amazon	G01K	$9.99	$9.99
10	1	-	310889508	#2-56 x 1/4 in. Phillips Machine Screws	Home Depot	9002676	$8.42	$8.42
MICROCONTROLLER								
6	8	IC3	PIC24FJ256GB410-I/PT	Microprocessor	Microchip	PIC24FJ256GB410-I/PT	$5.83	$46.64
5	50	C3-C12	TWK107B7104MVHT	0.1 uF capacitor	Mouser	963-TWK107B7104MVHT	$0.31	$15.50
5	6	C2	TMK063B7103KP-F	0.01 uF capacitor	Mouser	963-TMK063B7103KP-F	$0.10	$0.60
5	6	C1	TMK107BBJ106MA-T	10 uF capacitor	Mouser	963-TMK107BBJ106MA-T	$0.51	$3.06

5	10	J2, J3	B3B-ZR(LF)(SN)	3-pin connector	Digikey	455-1658-ND	$0.18	$1.80
5	6	J1	B02B-PASK(LF)(SN)	2-pin connector	Digikey	455-1817-ND	$0.25	$1.50
5	6	X2	SMA CONNECTOR	SMA connector	Mouser	471-SMACONNECTOR	$1.08	$6.48
6	50	C3-C12	TMK107BJ104KAHT	0.1 uF capacitor	Digikey	587-3472-2-ND	$0.04	$2.00
6	10	C2	C0603C103J3GACTU	0.01 uF capacitor	Digikey	399-7838-2-ND	$0.25	$2.50
6	10	C1	TMK107BBJ106MA-T	10 uF capacitor	Digikey	587-6023-2-ND	$0.30	$3.00
10	10	C2	C0603C103J3GACTU	0.01 uF (10,000 pF) capacitor	Mouser	80-C0603C103J3G	$0.17	$1.70
10	5	C1	TMK107BBJ106MA-T	10 uF capacitor	Mouser	963-TMK107BBJ106MA-T	$0.35	$1.75
10	2	IC3	PIC24FJ256GB410-I/PT	Microprocessor	Microchip	PIC24FJ256GB410-I/PT	$5.83	$11.66
10	5	U2	TLC555QDRQ1	TLC555 timer	Digikey	296-22999-2-ND	$0.35	$1.75
EXTERNAL SENSOR CONNECTORS								
6	8	J2	CONN02SMLPCB	2-pin connector male	UA Stock	WM4200-ND	-	-
6	8	J2	CONN02SFLPCB	2-pin connector female	UA Stock	WM2011-ND	-	-
6	8	J3	CONN03SMLPCB	3-pin connector male	UA Stock	WM4201-ND	-	-
6	8	J3	CONN03SFLPCB	3-pin connector female	UA Stock	WM2012-ND	-	-
LORA MODULE								
6	4	IC2	RN2903	LoRa Module	Digikey	RN2903A-I/RM103-ND	$12.80	$51.20
INTERNAL MOISTURE SENSOR								
7	8	C2	TMK107BJ104KAHT	0.1 uF capacitor	Digikey	587-3472-2-ND	$0.07	$0.56
7	8	U1	TLC555QDRQ1	TLC555 timer	Digikey	296-22999-2-ND	$0.83	$6.64
7	7	R1	CR2010-JW-101ELF	100 ohm resistor	Digikey	CR2010-JW-101ELFTR-ND	$0.14	$0.98
7	7	R2	RMCF2010JT10K0TR-ND	10k ohm resistor	Digikey	RMCF2010JT10K0CT-ND	$0.20	$1.40
7	7	R3	RMCF2010JT4K70	4.7k ohm resistor	Digikey	RMCF2010JT4K70TR-ND	$0.20	$1.40
7	7	R4	RMCF2010JT3M30	3.3M ohm resistor	Digikey	RMCF2010JT3M30TR-ND	$0.20	$1.40
7	7	R5	RMCF2010JT150KTR-ND	150k ohm resistor	Digikey	RMCF2010JT150K	$0.20	$1.40
10	5	R4	RMCF2010JT3M30	3.3M ohm resistor	Digikey	RMCF2010JT3M30TR-ND	$0.02	$0.10
10	10	R5	RMCF2010JT150KTR-ND	150k ohm resistor	Digikey	RMCF2010JT150K	$0.20	$2.00
10	20	R9-R11	RMCF0805JT1K00	1k ohm resistor	Digikey	RMCF0805JT1K00TR-ND	$0.02	$0.40
10	10	R6	RMCF0805JT10K0	10k ohm resistor	Digikey	RMCF0805JT10K0CT-ND	$0.02	$0.20
10	10	R7	RMCF0805JT470R	470 ohm resistor	Digikey	RMCF0805JT470RTR-ND	$0.02	$0.20

DEBUGGING								
7	1	-	1825910-6	Reset switch	Digikey	450-1650-ND	$0.10	$0.10
10	10	D1	AP2012EC	LED Lights Red	Mouser	604-AP2012EC	$0.13	$1.30
10	10	D2	17-21SYGC/S530-E2/TR8	LED Lights Green	Mouser	638-1721SYGCS530E2	$0.16	$1.60
10	10	D3	APT2012NW	LED Lights Orange	Mouser	604-APT2012NW	$0.19	$1.90
ANTENNA								
6	3	ANT1	RSRA6982700SSM	Antenna	Arcantenna	RSRA698/2700SSM	$15.00	$45.00
TEMPERATURE SENSOR PCB								
5	6	U1	MAX6608IUK+T	temperature sensor	Digikey	MAX6608IUK+TR-ND	$1.49	$8.94
5	6	C1	TMK063BJ102KP-F	1 nF capacitor	Mouser	963-TMK063BJ102KP-F	$0.10	$0.60
5	6	C2	TWK107B7104MVHT	0.1 uF capacitor	Mouser	963-TWK107B7104MVHT	$0.31	$1.86
6	5	U1	MAX6608IUK+T	temperature sensor	Digikey	MAX6608IUK+TR-ND	$1.49	$7.45
6	10	C1	C0603C102K3RACTU	1 nF capacitor	Digikey	399-7834-2-ND	$0.07	$0.70
6	10	C2	TMK107BJ104KAHT	0.1 uF capacitor	Digikey	587-3472-2-ND	$0.07	$0.70
7	7	C2	0805B204K500CT	0.2uF capacitor	Mouser	791-0805B204K500CT	$0.45	$3.15
TEMPERATURE SENSOR CONNECTOR PCB								
6	5	J2	CONN03SMLPCB	3-pin connector male	UA Stock	WM4201-ND	-	-
6	5	J2	CONN03SFLPCB	3-pin connector female	UA Stock	WM2012-ND	-	-
MOISTURE SENSOR CONNECTOR PCB								
6	5	J3	CONN02SMLPCB	2-pin connector male	UA Stock	WM4200-ND	-	-
6	5	J3	CONN02SFLPCB	2-pin connector female	UA Stock	WM2011-ND	-	-
BATTERY PACK PCB								
6	4	-	623360	3.7V 2800mA-Hr Battery	Amazon	623360	$10.99	$43.96
BATTERY MONITORING PCB								
10	7	U1	STC3100IST	IC Battery Monitoring	Newark	55T3932	$2.19	$15.33
10	10	R1 (Rcg Datasheet)	RL1220T-R010-J	10m ohm resistor	Digikey	RL12T.010JTR-ND	$0.02	$0.20
10	10	R15, R16	RMCF0805JT4K70	4.7k ohm resistor	Digikey	RMCF0805JT4K70TR-ND	$0.18	$1.80
10	10	R2 (Rosc Datasheet)	RMCF0805JT200K	200k ohm resistor	Digikey	RMCF0805JT200KTR-ND	$0.02	$0.20
10	10	C5 (C1 Datasheet)	EMF107B7105MAHT	1 uF capacitor	Mouser	963-EMF107B7105MAHT	$0.02	$0.20
10	10	C13 (C2 Datasheet)	EMF107B7224MAHT	0.22 uF capacitor	Mouser	963-EMF107B7224MAHT	$0.02	$0.20
10	10	R3 (R1 Datasheet)	RMCF0805JT1K00	1k ohm resistor	Digikey	RMCF0805JT1K00TR-ND	$0.20	$2.00

PCB DESIGNS								
6	1	set of 5	-	Soil Moisture Capacitor	JLCPCB	-	$2.00	$2.00
6	1	set of 5	-	Capacitor to Main Board	JLCPCB	-	$4.00	$4.00
6	1	set of 5	-	Temperature Sensor	JLCPCB	-	$4.00	$4.00
6	1	set of 5	-	Temperature to Main Board	JLCPCB	-	$4.00	$4.00
6	1	set of 5	-	Main Board Rev 1	JLCPCB	-	$4.00	$4.00
10	1	-	-	Main Board PCB w/ stencil	JLCPBC	-	$10.50	$10.50
10	1	-	-	Battery Pack PCB	JLCPCB	-	$4.00	$4.00

Total Spent	**$440.70**
Budget Spent	**$152.14**
Budget	**$600.00**
Budget Remaining	**$7.16**

Each design team was given a $600 budget to be used for prototypes and final designs. To keep track of the budget, the Total Cost column of the Materials Budget List was added up and subtracted from the total budget. The components in blue (development components) were subtracted from the total cost. Since these components can be reused for future design teams, the university absorbs the costs and it is not considered part of the $600 design team budget. The components in red (experimental components) were used for testing but never implemented in the final design. Two tables were created to separate design costs (Fall semester) and implementation costs (Spring semester). At the end of Spring semester, the Soil Sensor Network Design Team's remaining budget is $7.16.

9.3 Cost Comparison Analysis AW

The Soil Sensor Network is geared towards family farms. Most likely commercial farms can afford commercial-grade equipment to monitor soil properties, however, family farms cannot afford this luxury. Table 49 contains a cost analysis of the Soil Sensor Network, including pod costs and upfront costs.

Table 49: Sensor Pod Network Cost.

Qty.	Refdes	Part Num.	Description	Vendor	Vendor Part Num.	Cost	Total Cost
UP-FRONT COSTS							
1	-	450-00107-K1	LAIRD Gateway (LoRaWAN)	Digikey	776-450-00107-K1-ND	$375.16	$375.16
1	-	ADA1904	Battery Charger	Amazon	ADA1904	$9.33	$9.33
BATTERY MANAGEMENT							
1	-	623360	3.7V 2800 mA-Hr Battery	Amazon	623360	$7.47	$7.47
PROGRAMMABLE DEVICES							
1	IC2	RN2903	LoRa Module	Digikey	RN2903A-I/RM103-ND	$12.80	$12.80
1	IC3	PIC24FJ256GB410-I/PT	Microprocessor	Microchip	PIC24FJ256GB410-I/PT	$5.83	$5.83
MAIN PCB DESIGN							
7	C3-C9	TMK107BJ104KAHT	0.1 uF Capacitor	Digikey	587-3472-2-ND	$0.04	$0.25
1	C5	C0603C103J3GACTU	0.01 uF Capacitor	Digikey	399-7838-2-ND	$0.04	$0.04
1	C14	TMK107BBJ106MA-T	10 uF Capacitor	Digikey	587-6023-2-ND	$0.30	$0.30
1	J2	22232021	2-pin Connector Male	Digikey	WM4200-ND	$0.16	$0.16
1	J2	22012027	2-pin Connector Female	Digikey	WM2011-ND	$0.11	$0.11
1	J3	22232031	3-pin Connector Male	Digikey	WM4201-ND	$0.24	$0.24
1	J3	22012037	3-pin Connector Female	Digikey	WM2012-ND	$0.17	$0.17
1	X2	SMA CONNECTOR	SMA Connector	Mouser	471-SMACONNECTOR	$1.08	$1.08
1	U2	TLC555DR	Timer	Digikey	296-6501-2-ND - Tape & Reel (TR)	$0.36	$0.36
TEMPERATURE SENSOR PCB / CONNECTOR PCB							
1	IC4	MAX6608IUK+T	Temperature Sensor	Digikey	MAX6608IUK+TR-ND	$1.49	$1.49
1	C15	C0603C102K3RACTU	10 uF Capacitor	Digikey	399-7834-2-ND	$0.07	$0.07
1	C16	0805B204K500CT	0.2 uF Capacitor	Digikey	587-3472-2-ND	$0.07	$0.07
1	J3	22232031	3-pin Connector Male	Digikey	WM4201-ND	$0.24	$0.24
1	J3	22012037	3-pin Connector Female	Digikey	WM2012-ND	$0.17	$0.17
SOIL MOISTURE SENSOR / CONNECTOR PCB							
1	J2	22232021	2-pin connector Male	Digikey	WM4200-ND	$0.16	$0.16
1	J2	CONN02SFLPCB	2-pin connector Female	Digikey	WM2011-ND	$0.1	$0.11
BATTERY PACK PCB							
4	J6-J9	B2B-PH-K-S(LF)(SN)	2-pin Connector Female	Digikey	455-1704-ND	$0.16	$0.64
ANTENNA							
1	ANT1	RSRA6982700SSM	Antenna	Arcantenna	RSRA698/2700SSM	$15.00	$15.00
PCB DESIGNS							
1	-	-	Soil Moisture Capacitor	JLCPCB	-	$0.50	$0.50
1	-	-	Capacitor to Main Board	JLCPCB	-	$1.00	$1.00
1	-	-	Temperature Sensor	JLCPCB	-	$1.00	$1.00
1	-	-	Temperature to Main Board	JLCPCB	-	$1.00	$1.00
1	-	-	Main Board	JLCPCB	-	$1.00	$1.00
SENSOR POD DESIGN CONNECTORS							
1	-	R2-56X1/4	Connector Screw Male	Mouser	608-R2-56X1/4	$0.52	$0.52
12	-	M20X100C	Connector Screw Female	Mouser	761-M20X100C	$0.24	$2.88

TOTALS	
$426.41	Initial Setup
$54.65	Price per pod
$38.26	Industry price per pod

Upfront costs are defined as costs that farmers will only have to pay once. These include the gateway and battery charger. The gateway can support up to roughly 1000 pods because of its low bandwidth. Given that the average size of a large family farm is 5.7 km^2 and that there is one Sensor Pod every tenth of a kilometer, a large family farm on average would need 57 pods to monitor their field. This means one gateway is more than capable of managing the traffic of all the Sensor Pods.

The price to build one pod is roughly $54. If this design were being sold in the market, it is reasonable to make the assumption that buying supplies in bulk, and possibly working with vendors, would decrease the cost of the material by at least 30 percent. The cost of the pod would then become roughly $38 to manufacture. If manufacturing is done overseas, the cost of manufacturing the pod would decrease significantly more. By manufacturing in bulk and selling for $50 a pod, the company would make a substantial profit.

To gain a better understanding of how cost effective the proposed approach is to farmers, the average cost of commercial-grade soil monitoring equipment currently on the market is shown in Table 50.

Table 50: Market Costs for Sensor Networks.

Qty.	Part Num.	Description	Vendor	Vendor Part Num.	Cost	Total Cost
EQUIPMENT COSTS						
1	L200	Moisture, Humidity, Temperature	EarthScout	L200	$2,299.00	$2,299.00
1	TERRA 2x Sensor	Moisture, Temperature	TERRA	-	$1,899.00	$1,899.00
1	KT006	Moisture, Temperature	Semtek	KT006	$1,250.00	$1,250.00

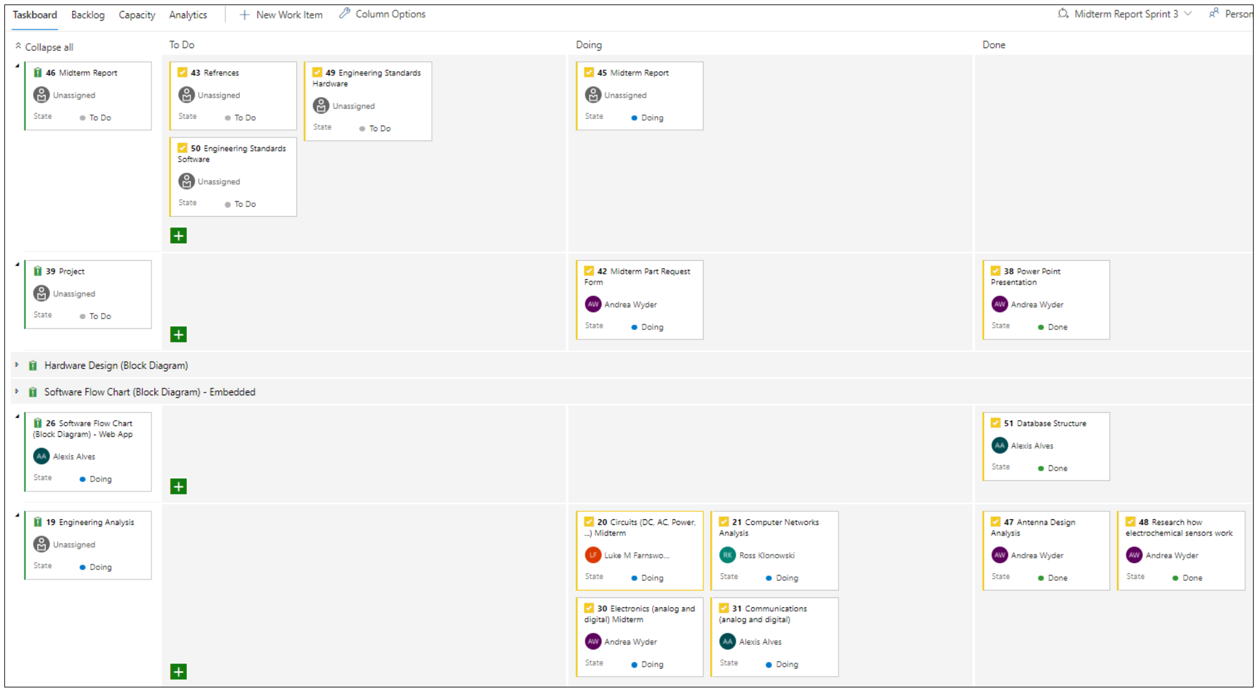
As seen above, equipment with the same sensor types of sensors costs over $1000. In addition to this, companies that sell this equipment only give farmers the option to rent out the communications system. Most wireless communications systems use licensed frequency bands, which means the user has to pay data rates to be granted access to the band since frequency is a limited resource. The proposed solution uses LoRa communication which operates on an unlicensed frequency band and does not require monthly or annual payments, further reducing operational costs.

10 Project Schedules AA, RK, AW

The workload for the Soil Sensor Network project was distributed in two different ways. A Gantt Chart was used for a top-level work distribution to remind the team when sections were due and which team (electrical or computer) was responsible for which section. Azure DevOps was used on a daily basis and updated weekly to give a visual representation of current progress.

10.1 Azure DevOps Sprint Board

Azure DevOps is a project management board, broken into sprints, that displays the progress a team is making per week. As seen in Figure 147, the work was first split up into main categories.



	To Do	Doing	Done
46 Midterm Report; Unassigned; State ● To Do	43 Refrences; Unassigned; State ● To Do 49 Engineering Standards Hardware; Unassigned; State ● To Do 50 Engineering Standards Software; Unassigned; State ● To Do	45 Midterm Report; Unassigned; State ● Doing	
39 Project; Unassigned; State ● To Do		42 Midterm Part Request Form; Andrea Wyder; State ● Doing	38 Power Point Presentation; Andrea Wyder; State ● Done
Hardware Design (Block Diagram)			
Software Flow Chart (Block Diagram) - Embedded			
26 Software Flow Chart (Block Diagram) - Web App; Alexis Alves; State ● Doing			51 Database Structure; Alexis Alves; State ● Done
19 Engineering Analysis; Unassigned; State ● Doing		20 Circuits (DC, AC, Power, ...) Midterm; Luke M Farnswo...; State ● Doing 21 Computer Networks Analysis; Ross Klonowski; State ● Doing 30 Electronics (analog and digital) Midterm; Andrea Wyder; State ● Doing 31 Communications (analog and digital); Alexis Alves; State ● Doing	47 Antenna Design Analysis; Andrea Wyder; State ● Done 48 Research how electrochemical sensors work; Andrea Wyder; State ● Done

Figure 147: Azure DevOps Board Sprint 3.

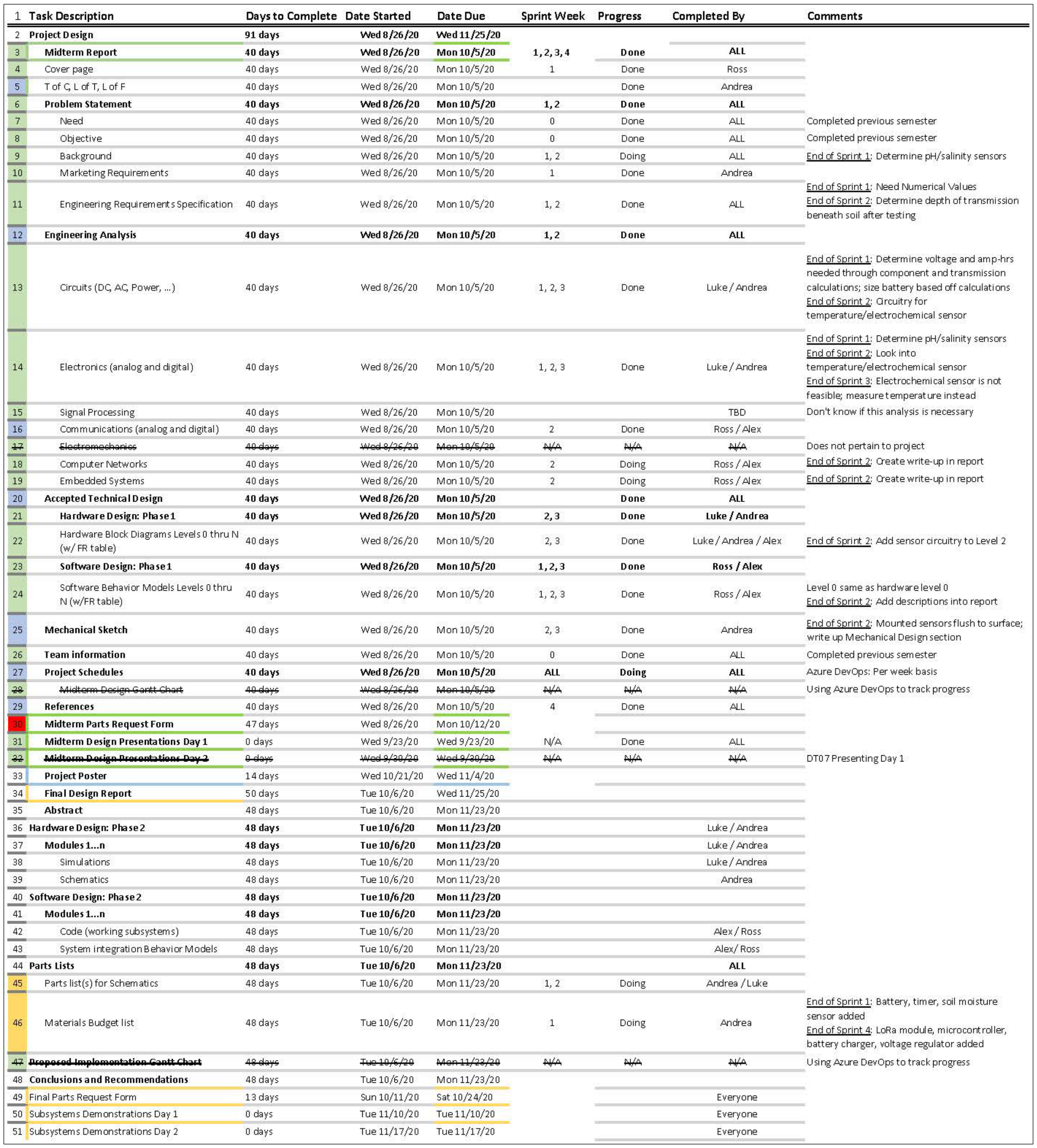
For Sprint 3, a few of these categories included the Midterm Report, Project, and Engineering Analysis. These were then broken down into subcategories and assigned to individual members. For instance, the Engineering Analysis was further broken into Circuit Analysis, Computer

Network Analysis, Electronics Analysis, Communications Analysis, Antenna Design, and Electrochemical Sensor Design. Each subcategory can be organized into three columns: To Do, Doing, and Done. The goal for each sprint is to complete each subcategory and move it to the Done column before the end of the week. Weekly sprints were chosen due to rapidly approaching deadlines. As with Agile Development, any subcategory that does not get completed by the end of the sprint can be reevaluated and carried over to the next sprint if needed.

It can be argued that DevOps is a better alternative than Microsoft Projects for multiple reasons. The software is free and accessible for a small project team of up to five users. It also offers an interactive user interface where teams can physically move tasks, issues, and improvements from one state to another to visualize the progress of their project. The reports for these sprints can then be exported to track progress over the duration of the project. Azure Dev-Ops was not relied upon as heavily during the last semester of the project.

10.2 Design Gantt Chart

A Gantt Chart is used to keep track of project milestones throughout the semester. The chart is populated with every task to be completed. After each sprint, the Sprint Week and Progress columns are updated; the sprint displays the week the task was worked on and the Progress displays the status of the task. If the task was started but not completed, the Progress column is assigned as "Doing" and a comment is made to tell what part of the task was completed during that sprint. An example of the Design Gantt Chart can be seen in Figure 148.

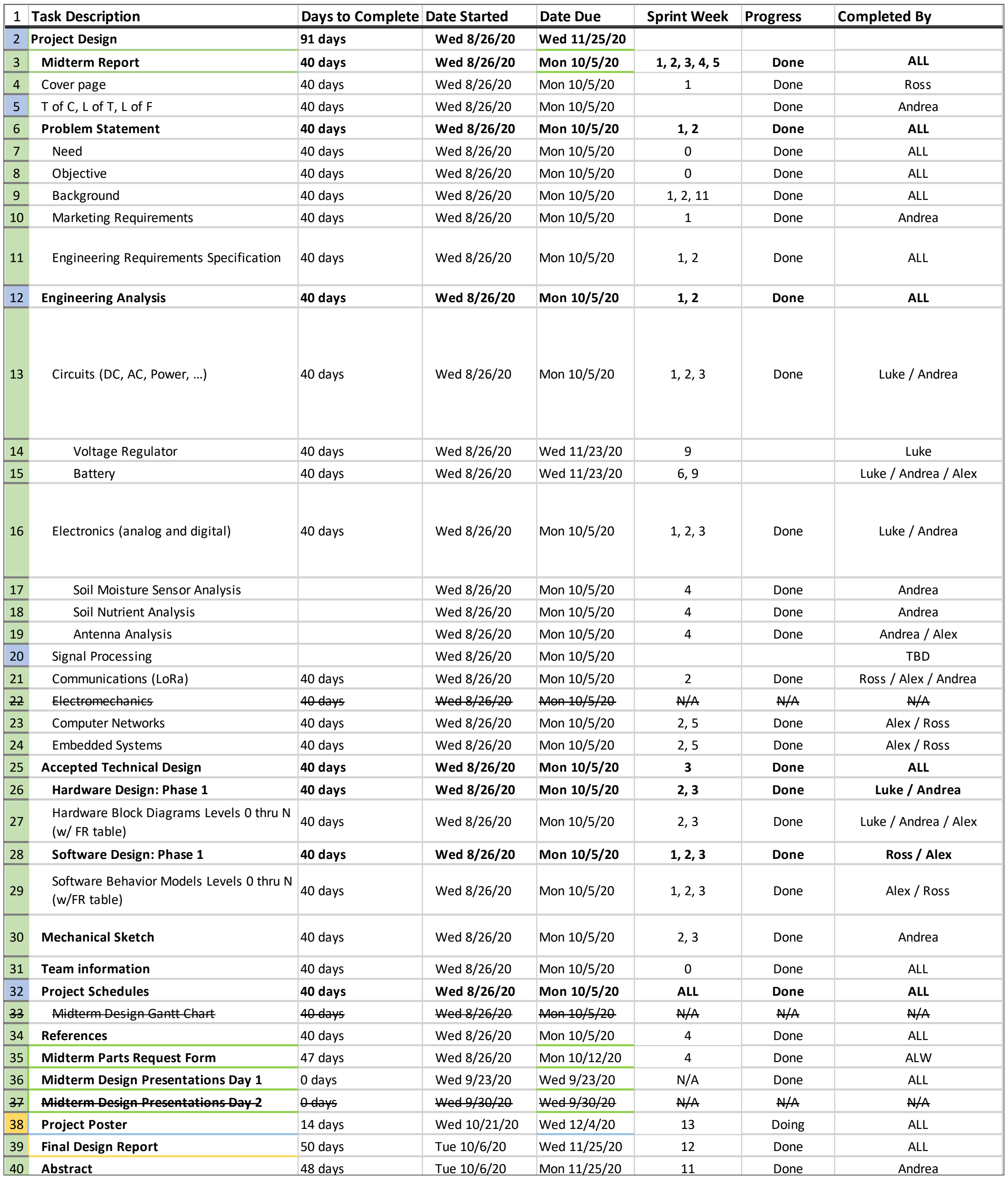


1	Task Description	Days to Complete	Date Started	Date Due	Sprint Week	Progress	Completed By	Comments
2	**Project Design**	**91 days**	**Wed 8/26/20**	**Wed 11/25/20**				
3	**Midterm Report**	**40 days**	**Wed 8/26/20**	**Mon 10/5/20**	**1, 2, 3, 4**	**Done**	**ALL**	
4	Cover page	40 days	Wed 8/26/20	Mon 10/5/20	1	Done	Ross	
5	T of C, L of T, L of F	40 days	Wed 8/26/20	Mon 10/5/20		Done	Andrea	
6	**Problem Statement**	**40 days**	**Wed 8/26/20**	**Mon 10/5/20**	**1, 2**	**Done**	**ALL**	
7	Need	40 days	Wed 8/26/20	Mon 10/5/20	0	Done	ALL	Completed previous semester
8	Objective	40 days	Wed 8/26/20	Mon 10/5/20	0	Done	ALL	Completed previous semester
9	Background	40 days	Wed 8/26/20	Mon 10/5/20	1, 2	Doing	ALL	End of Sprint 1: Determine pH/salinity sensors
10	Marketing Requirements	40 days	Wed 8/26/20	Mon 10/5/20	1	Done	Andrea	
11	Engineering Requirements Specification	40 days	Wed 8/26/20	Mon 10/5/20	1, 2	Done	ALL	End of Sprint 1: Need Numerical Values End of Sprint 2: Determine depth of transmission beneath soil after testing
12	**Engineering Analysis**	**40 days**	**Wed 8/26/20**	**Mon 10/5/20**	**1, 2**	**Done**	**ALL**	
13	Circuits (DC, AC, Power, ...)	40 days	Wed 8/26/20	Mon 10/5/20	1, 2, 3	Done	Luke / Andrea	End of Sprint 1: Determine voltage and amp-hrs needed through component and transmission calculations; size battery based off calculations End of Sprint 2: Circuitry for temperature/electrochemical sensor
14	Electronics (analog and digital)	40 days	Wed 8/26/20	Mon 10/5/20	1, 2, 3	Done	Luke / Andrea	End of Sprint 1: Determine pH/salinity sensors End of Sprint 2: Look into temperature/electrochemical sensor End of Sprint 3: Electrochemical sensor is not feasible; measure temperature instead
15	Signal Processing	40 days	Wed 8/26/20	Mon 10/5/20			TBD	Don't know if this analysis is necessary
16	Communications (analog and digital)	40 days	Wed 8/26/20	Mon 10/5/20	2	Done	Ross / Alex	
~~17~~	~~Electromechanics~~	~~40 days~~	~~Wed 8/26/20~~	~~Mon 10/5/20~~	~~N/A~~	~~N/A~~	~~N/A~~	Does not pertain to project
18	Computer Networks	40 days	Wed 8/26/20	Mon 10/5/20	2	Doing	Ross / Alex	End of Sprint 2: Create write-up in report
19	Embedded Systems	40 days	Wed 8/26/20	Mon 10/5/20	2	Doing	Ross / Alex	End of Sprint 2: Create write-up in report
20	**Accepted Technical Design**	**40 days**	**Wed 8/26/20**	**Mon 10/5/20**		**Done**	**ALL**	
21	**Hardware Design: Phase 1**	**40 days**	**Wed 8/26/20**	**Mon 10/5/20**	**2, 3**	**Done**	**Luke / Andrea**	
22	Hardware Block Diagrams Levels 0 thru N (w/ FR table)	40 days	Wed 8/26/20	Mon 10/5/20	2, 3	Done	Luke / Andrea / Alex	End of Sprint 2: Add sensor circuitry to Level 2
23	**Software Design: Phase 1**	**40 days**	**Wed 8/26/20**	**Mon 10/5/20**	**1, 2, 3**	**Done**	**Ross / Alex**	
24	Software Behavior Models Levels 0 thru N (w/FR table)	40 days	Wed 8/26/20	Mon 10/5/20	1, 2, 3	Done	Ross / Alex	Level 0 same as hardware level 0 End of Sprint 2: Add descriptions into report
25	**Mechanical Sketch**	40 days	Wed 8/26/20	Mon 10/5/20	2, 3	Done	Andrea	End of Sprint 2: Mounted sensors flush to surface; write up Mechanical Design section
26	**Team information**	40 days	Wed 8/26/20	Mon 10/5/20	0	Done	ALL	Completed previous semester
27	**Project Schedules**	**40 days**	**Wed 8/26/20**	**Mon 10/5/20**	**ALL**	**Doing**	**ALL**	Azure DevOps: Per week basis
~~28~~	~~Midterm Design Gantt Chart~~	~~40 days~~	~~Wed 8/26/20~~	~~Mon 10/5/20~~	~~N/A~~	~~N/A~~	~~N/A~~	Using Azure DevOps to track progress
29	**References**	40 days	Wed 8/26/20	Mon 10/5/20	4	Done	ALL	
30	**Midterm Parts Request Form**	47 days	Wed 8/26/20	Mon 10/12/20				
31	**Midterm Design Presentations Day 1**	0 days	Wed 9/23/20	Wed 9/23/20	N/A	Done	ALL	
~~32~~	~~**Midterm Design Presentations Day 2**~~	~~0 days~~	~~Wed 9/30/20~~	~~Wed 9/30/20~~	~~N/A~~	~~N/A~~	~~N/A~~	DT07 Presenting Day 1
33	**Project Poster**	14 days	Wed 10/21/20	Wed 11/4/20				
34	**Final Design Report**	50 days	Tue 10/6/20	Wed 11/25/20				
35	**Abstract**	48 days	Tue 10/6/20	Mon 11/23/20				
36	**Hardware Design: Phase 2**	**48 days**	**Tue 10/6/20**	**Mon 11/23/20**			Luke / Andrea	
37	**Modules 1...n**	**48 days**	**Tue 10/6/20**	**Mon 11/23/20**			Luke / Andrea	
38	Simulations	48 days	Tue 10/6/20	Mon 11/23/20			Luke / Andrea	
39	Schematics	48 days	Tue 10/6/20	Mon 11/23/20			Andrea	
40	**Software Design: Phase 2**	**48 days**	**Tue 10/6/20**	**Mon 11/23/20**				
41	**Modules 1...n**	**48 days**	**Tue 10/6/20**	**Mon 11/23/20**				
42	Code (working subsystems)	48 days	Tue 10/6/20	Mon 11/23/20			Alex / Ross	
43	System integration Behavior Models	48 days	Tue 10/6/20	Mon 11/23/20			Alex/ Ross	
44	**Parts Lists**	**48 days**	**Tue 10/6/20**	**Mon 11/23/20**			**ALL**	
45	Parts list(s) for Schematics	48 days	Tue 10/6/20	Mon 11/23/20	1, 2	Doing	Andrea / Luke	
46	Materials Budget list	48 days	Tue 10/6/20	Mon 11/23/20	1	Doing	Andrea	End of Sprint 1: Battery, timer, soil moisture sensor added End of Sprint 4: LoRa module, microcontroller, battery charger, voltage regulator added
~~47~~	~~**Proposed Implementation Gantt Chart**~~	~~48 days~~	~~Tue 10/6/20~~	~~Mon 11/23/20~~	~~N/A~~	~~N/A~~	~~N/A~~	Using Azure DevOps to track progress
48	**Conclusions and Recommendations**	48 days	Tue 10/6/20	Mon 11/23/20				
49	Final Parts Request Form	13 days	Sun 10/11/20	Sat 10/24/20			Everyone	
50	Subsystems Demonstrations Day 1	0 days	Tue 11/10/20	Tue 11/10/20			Everyone	
51	Subsystems Demonstrations Day 2	0 days	Tue 11/17/20	Tue 11/17/20			Everyone	

Figure 148: Design Gantt Chart.

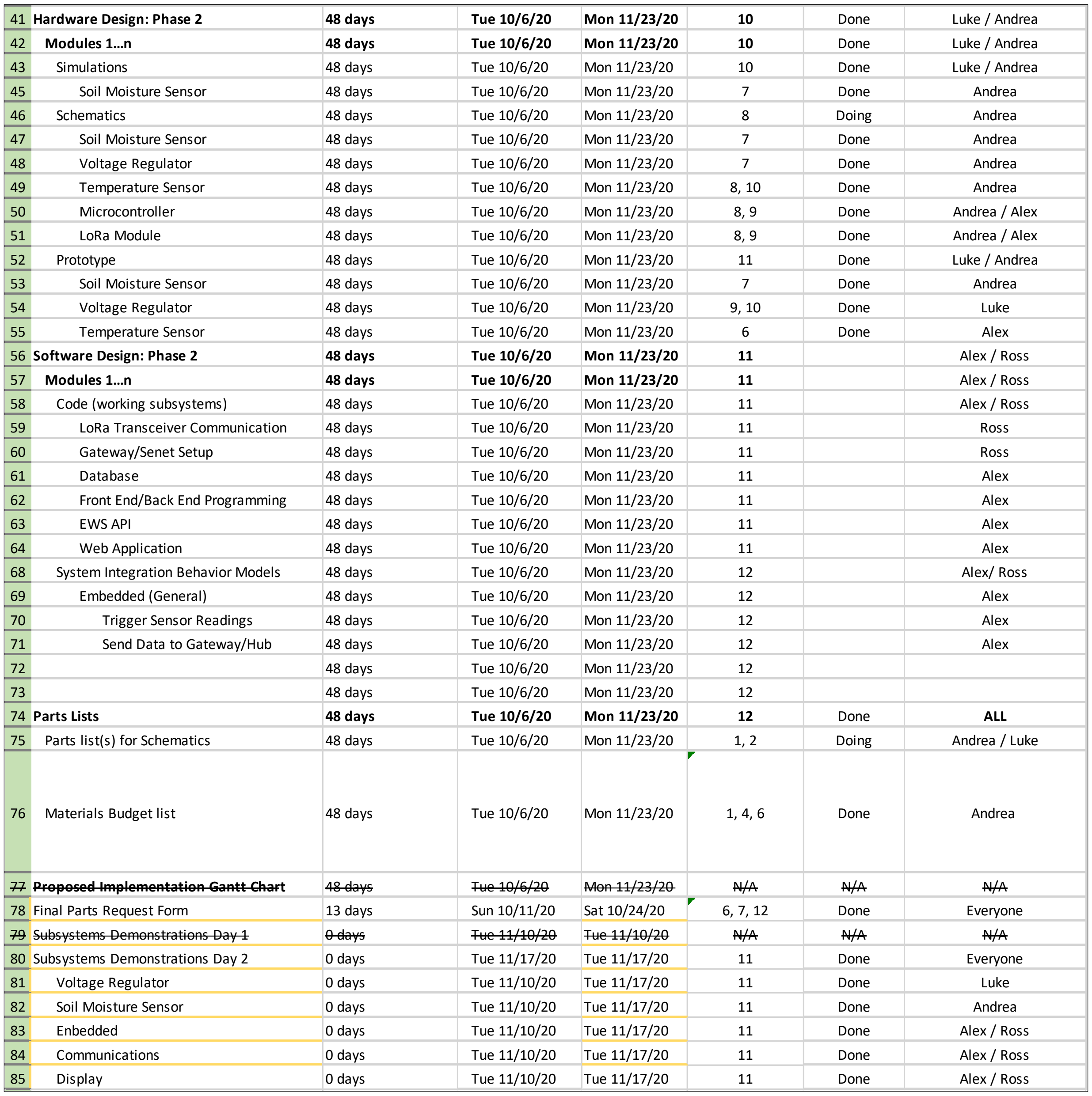
10.3 Implementation Gantt Chart

The Implementation Gantt Chart followed the same format at the Design Gantt Chart, which can be seen in Figures 149 and 150.



1	Task Description	Days to Complete	Date Started	Date Due	Sprint Week	Progress	Completed By
2	**Project Design**	**91 days**	**Wed 8/26/20**	**Wed 11/25/20**			
3	**Midterm Report**	**40 days**	**Wed 8/26/20**	**Mon 10/5/20**	**1, 2, 3, 4, 5**	**Done**	**ALL**
4	Cover page	40 days	Wed 8/26/20	Mon 10/5/20	1	Done	Ross
5	T of C, L of T, L of F	40 days	Wed 8/26/20	Mon 10/5/20		Done	Andrea
6	**Problem Statement**	**40 days**	**Wed 8/26/20**	**Mon 10/5/20**	**1, 2**	**Done**	**ALL**
7	Need	40 days	Wed 8/26/20	Mon 10/5/20	0	Done	ALL
8	Objective	40 days	Wed 8/26/20	Mon 10/5/20	0	Done	ALL
9	Background	40 days	Wed 8/26/20	Mon 10/5/20	1, 2, 11	Done	ALL
10	Marketing Requirements	40 days	Wed 8/26/20	Mon 10/5/20	1	Done	Andrea
11	Engineering Requirements Specification	40 days	Wed 8/26/20	Mon 10/5/20	1, 2	Done	ALL
12	**Engineering Analysis**	**40 days**	**Wed 8/26/20**	**Mon 10/5/20**	**1, 2**	**Done**	**ALL**
13	Circuits (DC, AC, Power, …)	40 days	Wed 8/26/20	Mon 10/5/20	1, 2, 3	Done	Luke / Andrea
14	Voltage Regulator	40 days	Wed 8/26/20	Wed 11/23/20	9		Luke
15	Battery	40 days	Wed 8/26/20	Wed 11/23/20	6, 9		Luke / Andrea / Alex
16	Electronics (analog and digital)	40 days	Wed 8/26/20	Mon 10/5/20	1, 2, 3	Done	Luke / Andrea
17	Soil Moisture Sensor Analysis		Wed 8/26/20	Mon 10/5/20	4	Done	Andrea
18	Soil Nutrient Analysis		Wed 8/26/20	Mon 10/5/20	4	Done	Andrea
19	Antenna Analysis		Wed 8/26/20	Mon 10/5/20	4	Done	Andrea / Alex
20	Signal Processing		Wed 8/26/20	Mon 10/5/20			TBD
21	Communications (LoRa)	40 days	Wed 8/26/20	Mon 10/5/20	2	Done	Ross / Alex / Andrea
~~22~~	~~Electromechanics~~	~~40 days~~	~~Wed 8/26/20~~	~~Mon 10/5/20~~	~~N/A~~	~~N/A~~	~~N/A~~
23	Computer Networks	40 days	Wed 8/26/20	Mon 10/5/20	2, 5	Done	Alex / Ross
24	Embedded Systems	40 days	Wed 8/26/20	Mon 10/5/20	2, 5	Done	Alex / Ross
25	**Accepted Technical Design**	**40 days**	**Wed 8/26/20**	**Mon 10/5/20**	**3**	**Done**	**ALL**
26	**Hardware Design: Phase 1**	**40 days**	**Wed 8/26/20**	**Mon 10/5/20**	**2, 3**	**Done**	**Luke / Andrea**
27	Hardware Block Diagrams Levels 0 thru N (w/ FR table)	40 days	Wed 8/26/20	Mon 10/5/20	2, 3	Done	Luke / Andrea / Alex
28	**Software Design: Phase 1**	**40 days**	**Wed 8/26/20**	**Mon 10/5/20**	**1, 2, 3**	**Done**	**Ross / Alex**
29	Software Behavior Models Levels 0 thru N (w/FR table)	40 days	Wed 8/26/20	Mon 10/5/20	1, 2, 3	Done	Alex / Ross
30	**Mechanical Sketch**	40 days	Wed 8/26/20	Mon 10/5/20	2, 3	Done	Andrea
31	**Team information**	40 days	Wed 8/26/20	Mon 10/5/20	0	Done	ALL
32	**Project Schedules**	**40 days**	**Wed 8/26/20**	**Mon 10/5/20**	**ALL**	**Done**	**ALL**
~~33~~	~~Midterm Design Gantt Chart~~	~~40 days~~	~~Wed 8/26/20~~	~~Mon 10/5/20~~	~~N/A~~	~~N/A~~	~~N/A~~
34	**References**	40 days	Wed 8/26/20	Mon 10/5/20	4	Done	ALL
35	**Midterm Parts Request Form**	47 days	Wed 8/26/20	Mon 10/12/20	4	Done	ALW
36	**Midterm Design Presentations Day 1**	0 days	Wed 9/23/20	Wed 9/23/20	N/A	Done	ALL
~~37~~	~~**Midterm Design Presentations Day 2**~~	~~0 days~~	~~Wed 9/30/20~~	~~Wed 9/30/20~~	~~N/A~~	~~N/A~~	~~N/A~~
38	**Project Poster**	14 days	Wed 10/21/20	Wed 12/4/20	13	Doing	ALL
39	**Final Design Report**	50 days	Tue 10/6/20	Wed 11/25/20	12	Done	ALL
40	**Abstract**	48 days	Tue 10/6/20	Mon 11/25/20	11	Done	Andrea

Figure 149: Implementation Gantt Chart (1).

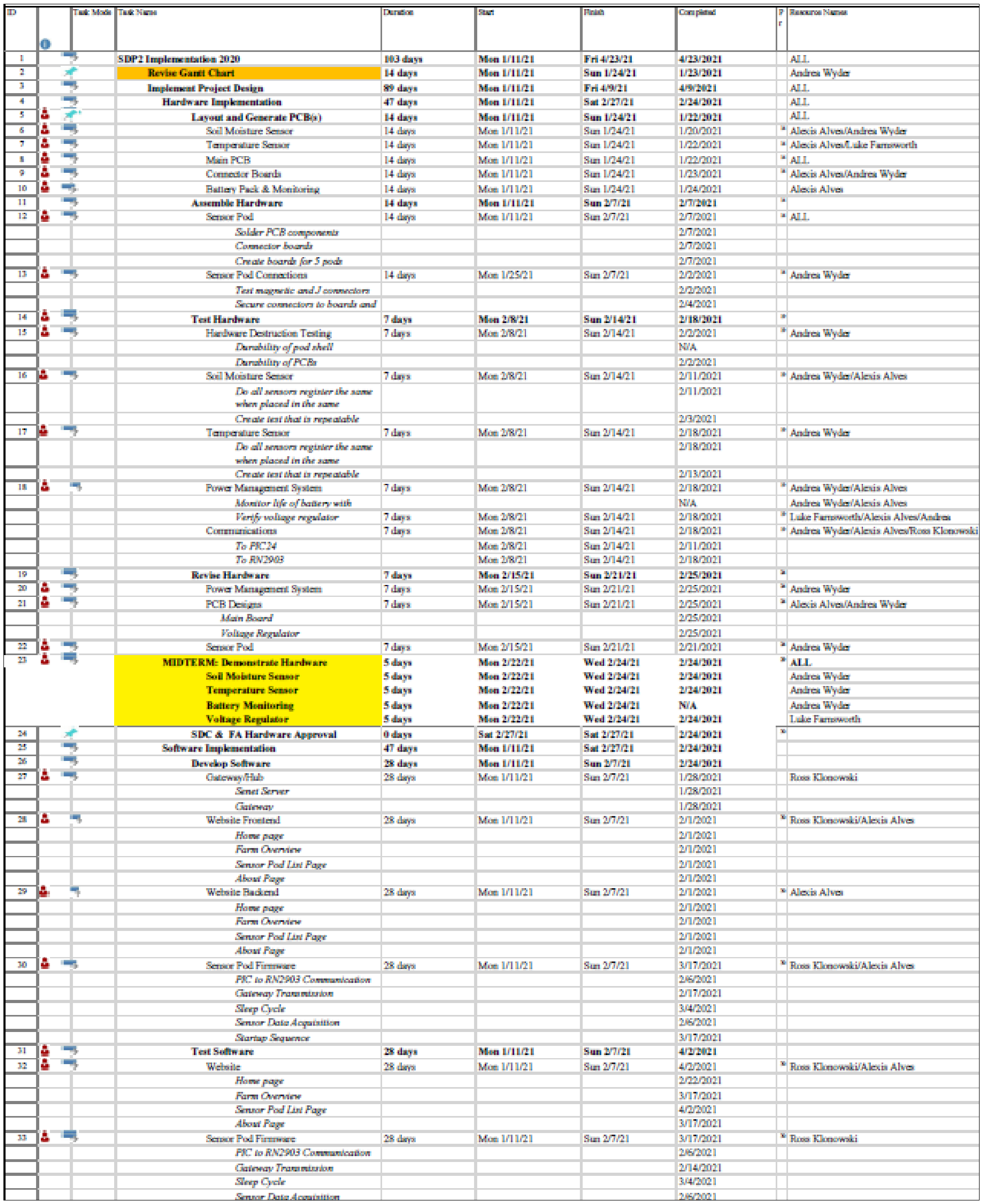


41	**Hardware Design: Phase 2**	**48 days**	**Tue 10/6/20**	**Mon 11/23/20**	**10**	Done	Luke / Andrea
42	**Modules 1…n**	**48 days**	**Tue 10/6/20**	**Mon 11/23/20**	**10**	Done	Luke / Andrea
43	Simulations	48 days	Tue 10/6/20	Mon 11/23/20	10	Done	Luke / Andrea
45	Soil Moisture Sensor	48 days	Tue 10/6/20	Mon 11/23/20	7	Done	Andrea
46	Schematics	48 days	Tue 10/6/20	Mon 11/23/20	8	Doing	Andrea
47	Soil Moisture Sensor	48 days	Tue 10/6/20	Mon 11/23/20	7	Done	Andrea
48	Voltage Regulator	48 days	Tue 10/6/20	Mon 11/23/20	7	Done	Andrea
49	Temperature Sensor	48 days	Tue 10/6/20	Mon 11/23/20	8, 10	Done	Andrea
50	Microcontroller	48 days	Tue 10/6/20	Mon 11/23/20	8, 9	Done	Andrea / Alex
51	LoRa Module	48 days	Tue 10/6/20	Mon 11/23/20	8, 9	Done	Andrea / Alex
52	Prototype	48 days	Tue 10/6/20	Mon 11/23/20	11	Done	Luke / Andrea
53	Soil Moisture Sensor	48 days	Tue 10/6/20	Mon 11/23/20	7	Done	Andrea
54	Voltage Regulator	48 days	Tue 10/6/20	Mon 11/23/20	9, 10	Done	Luke
55	Temperature Sensor	48 days	Tue 10/6/20	Mon 11/23/20	6	Done	Alex
56	**Software Design: Phase 2**	**48 days**	**Tue 10/6/20**	**Mon 11/23/20**	**11**		Alex / Ross
57	**Modules 1…n**	**48 days**	**Tue 10/6/20**	**Mon 11/23/20**	**11**		Alex / Ross
58	Code (working subsystems)	48 days	Tue 10/6/20	Mon 11/23/20	11		Alex / Ross
59	LoRa Transceiver Communication	48 days	Tue 10/6/20	Mon 11/23/20	11		Ross
60	Gateway/Senet Setup	48 days	Tue 10/6/20	Mon 11/23/20	11		Ross
61	Database	48 days	Tue 10/6/20	Mon 11/23/20	11		Alex
62	Front End/Back End Programming	48 days	Tue 10/6/20	Mon 11/23/20	11		Alex
63	EWS API	48 days	Tue 10/6/20	Mon 11/23/20	11		Alex
64	Web Application	48 days	Tue 10/6/20	Mon 11/23/20	11		Alex
68	System Integration Behavior Models	48 days	Tue 10/6/20	Mon 11/23/20	12		Alex/ Ross
69	Embedded (General)	48 days	Tue 10/6/20	Mon 11/23/20	12		Alex
70	Trigger Sensor Readings	48 days	Tue 10/6/20	Mon 11/23/20	12		Alex
71	Send Data to Gateway/Hub	48 days	Tue 10/6/20	Mon 11/23/20	12		Alex
72		48 days	Tue 10/6/20	Mon 11/23/20	12		
73		48 days	Tue 10/6/20	Mon 11/23/20	12		
74	**Parts Lists**	**48 days**	**Tue 10/6/20**	**Mon 11/23/20**	**12**	Done	**ALL**
75	Parts list(s) for Schematics	48 days	Tue 10/6/20	Mon 11/23/20	1, 2	Doing	Andrea / Luke
76	Materials Budget list	48 days	Tue 10/6/20	Mon 11/23/20	1, 4, 6	Done	Andrea
~~77~~	~~**Proposed Implementation Gantt Chart**~~	~~48 days~~	~~Tue 10/6/20~~	~~Mon 11/23/20~~	~~N/A~~	~~N/A~~	~~N/A~~
78	Final Parts Request Form	13 days	Sun 10/11/20	Sat 10/24/20	6, 7, 12	Done	Everyone
~~79~~	~~Subsystems Demonstrations Day 1~~	~~0 days~~	~~Tue 11/10/20~~	~~Tue 11/10/20~~	~~N/A~~	~~N/A~~	~~N/A~~
80	Subsystems Demonstrations Day 2	0 days	Tue 11/17/20	Tue 11/17/20	11	Done	Everyone
81	Voltage Regulator	0 days	Tue 11/10/20	Tue 11/17/20	11	Done	Luke
82	Soil Moisture Sensor	0 days	Tue 11/10/20	Tue 11/17/20	11	Done	Andrea
83	Enbedded	0 days	Tue 11/10/20	Tue 11/17/20	11	Done	Alex / Ross
84	Communications	0 days	Tue 11/10/20	Tue 11/17/20	11	Done	Alex / Ross
85	Display	0 days	Tue 11/10/20	Tue 11/17/20	11	Done	Alex / Ross

Figure 150: Implementation Gantt Chart (2).

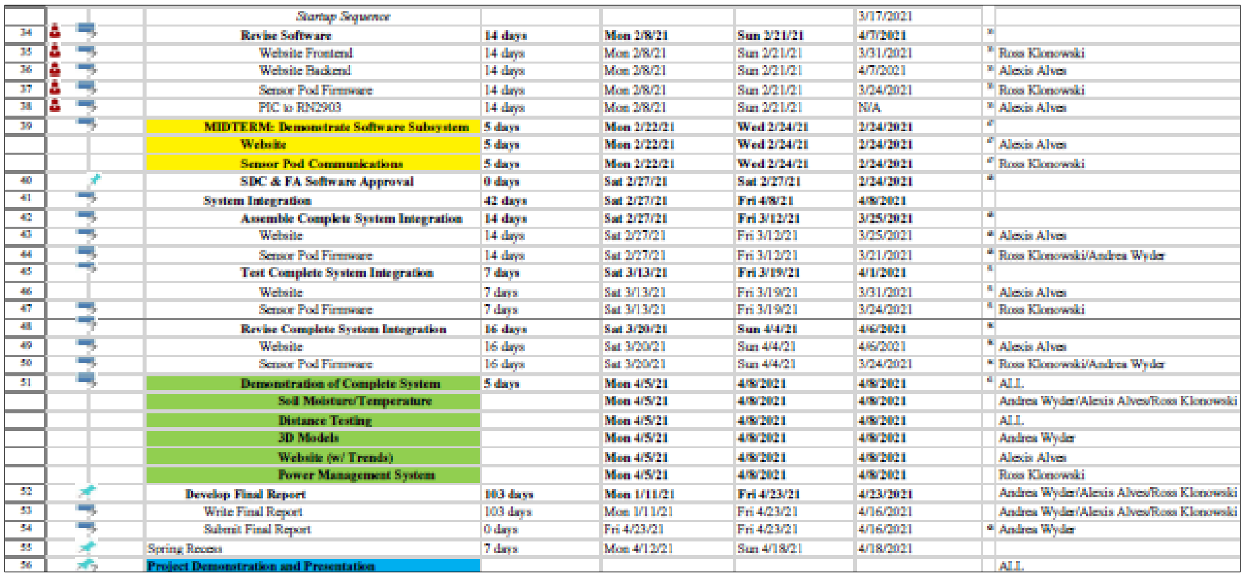
10.4 Actual Gant Chart

The Actual Gantt Chart for the final semester implementation followed the same format as the previous two Gantt Charts. This Gantt Chart provided a more detailed description of how and when each subsystem would be designed and integrated, as seen in Figures 151 and 152.



ID		Task Mode	Task Name	Duration	Start	Finish	Completed	P r	Resource Names
1			**SDP2 Implementation 2020**	**103 days**	**Mon 1/11/21**	**Fri 4/23/21**	**4/23/2021**		ALL
2			**Revise Gantt Chart**	**14 days**	**Mon 1/11/21**	**Sun 1/24/21**	**1/23/2021**		Andrea Wyder
3			**Implement Project Design**	**89 days**	**Mon 1/11/21**	**Fri 4/9/21**	**4/9/2021**		ALL
4			**Hardware Implementation**	**47 days**	**Mon 1/11/21**	**Sat 2/27/21**	**2/24/2021**		ALL
5			**Layout and Generate PCB(s)**	**14 days**	**Mon 1/11/21**	**Sun 1/24/21**	**1/22/2021**		ALL
6			Soil Moisture Sensor	14 days	Mon 1/11/21	Sun 1/24/21	1/20/2021		Alexis Alves/Andrea Wyder
7			Temperature Sensor	14 days	Mon 1/11/21	Sun 1/24/21	1/22/2021		Alexis Alves/Luke Farnsworth
8			Main PCB	14 days	Mon 1/11/21	Sun 1/24/21	1/22/2021		ALL
9			Connector Boards	14 days	Mon 1/11/21	Sun 1/24/21	1/23/2021		Alexis Alves/Andrea Wyder
10			Battery Pack & Monitoring	14 days	Mon 1/11/21	Sun 1/24/21	1/24/2021		Alexis Alves
11			**Assemble Hardware**	**14 days**	**Mon 1/11/21**	**Sun 2/7/21**	**2/7/2021**		
12			Sensor Pod	14 days	Mon 1/11/21	Sun 2/7/21	2/7/2021		ALL
			Solder PCB components				2/7/2021		
			Connector boards				2/7/2021		
			Create boards for 5 pods				2/7/2021		
13			Sensor Pod Connections	14 days	Mon 1/25/21	Sun 2/7/21	2/2/2021		Andrea Wyder
			Test magnetic and J connectors				2/2/2021		
			Secure connectors to boards and				2/4/2021		
14			**Test Hardware**	**7 days**	**Mon 2/8/21**	**Sun 2/14/21**	**2/18/2021**		
15			Hardware Destruction Testing	7 days	Mon 2/8/21	Sun 2/14/21	2/2/2021		Andrea Wyder
			Durability of pod shell				N/A		
			Durability of PCBs				2/2/2021		
16			Soil Moisture Sensor	7 days	Mon 2/8/21	Sun 2/14/21	2/11/2021		Andrea Wyder/Alexis Alves
			Do all sensors register the same when placed in the same				2/11/2021		
			Create test that is repeatable				2/3/2021		
17			Temperature Sensor	7 days	Mon 2/8/21	Sun 2/14/21	2/18/2021		Andrea Wyder
			Do all sensors register the same when placed in the same				2/18/2021		
			Create test that is repeatable				2/13/2021		
18			Power Management System	7 days	Mon 2/8/21	Sun 2/14/21	2/18/2021		Andrea Wyder/Alexis Alves
			Monitor life of battery with				N/A		Andrea Wyder/Alexis Alves
			Verify voltage regulator	7 days	Mon 2/8/21	Sun 2/14/21	2/18/2021		Luke Farnsworth/Alexis Alves/Andrea
			Communications	7 days	Mon 2/8/21	Sun 2/14/21	2/18/2021		Andrea Wyder/Alexis Alves/Ross Klonowski
			To PIC24		Mon 2/8/21	Sun 2/14/21	2/11/2021		
			To RN2903		Mon 2/8/21	Sun 2/14/21	2/18/2021		
19			**Revise Hardware**	**7 days**	**Mon 2/15/21**	**Sun 2/21/21**	**2/25/2021**		
20			Power Management System	7 days	Mon 2/15/21	Sun 2/21/21	2/25/2021		Andrea Wyder
21			PCB Designs	7 days	Mon 2/15/21	Sun 2/21/21	2/25/2021		Alexis Alves/Andrea Wyder
			Main Board				2/25/2021		
			Voltage Regulator				2/25/2021		
22			Sensor Pod	7 days	Mon 2/15/21	Sun 2/21/21	2/21/2021		Andrea Wyder
23			**MIDTERM: Demonstrate Hardware**	**5 days**	**Mon 2/22/21**	**Wed 2/24/21**	**2/24/2021**		**ALL**
			Soil Moisture Sensor	**5 days**	**Mon 2/22/21**	**Wed 2/24/21**	**2/24/2021**		Andrea Wyder
			Temperature Sensor	**5 days**	**Mon 2/22/21**	**Wed 2/24/21**	**2/24/2021**		Andrea Wyder
			Battery Monitoring	**5 days**	**Mon 2/22/21**	**Wed 2/24/21**	**N/A**		Andrea Wyder
			Voltage Regulator	**5 days**	**Mon 2/22/21**	**Wed 2/24/21**	**2/24/2021**		Luke Farnsworth
24			**SDC & FA Hardware Approval**	**0 days**	**Sat 2/27/21**	**Sat 2/27/21**	**2/24/2021**		
25			**Software Implementation**	**47 days**	**Mon 1/11/21**	**Sat 2/27/21**	**2/24/2021**		
26			**Develop Software**	**28 days**	**Mon 1/11/21**	**Sun 2/7/21**	**2/24/2021**		
27			Gateway/Hub	28 days	Mon 1/11/21	Sun 2/7/21	1/28/2021		Ross Klonowski
			Senet Server				1/28/2021		
			Gateway				1/28/2021		
28			Website Frontend	28 days	Mon 1/11/21	Sun 2/7/21	2/1/2021		Ross Klonowski/Alexis Alves
			Home page				2/1/2021		
			Farm Overview				2/1/2021		
			Sensor Pod List Page				2/1/2021		
			About Page				2/1/2021		
29			Website Backend	28 days	Mon 1/11/21	Sun 2/7/21	2/1/2021		Alexis Alves
			Home page				2/1/2021		
			Farm Overview				2/1/2021		
			Sensor Pod List Page				2/1/2021		
			About Page				2/1/2021		
30			Sensor Pod Firmware	28 days	Mon 1/11/21	Sun 2/7/21	3/17/2021		Ross Klonowski/Alexis Alves
			PIC to RN2903 Communication				2/6/2021		
			Gateway Transmission				2/17/2021		
			Sleep Cycle				3/4/2021		
			Sensor Data Acquisition				2/6/2021		
			Startup Sequence				3/17/2021		
31			**Test Software**	**28 days**	**Mon 1/11/21**	**Sun 2/7/21**	**4/2/2021**		
32			Website	28 days	Mon 1/11/21	Sun 2/7/21	4/2/2021		Ross Klonowski/Alexis Alves
			Home page				2/22/2021		
			Farm Overview				3/17/2021		
			Sensor Pod List Page				4/2/2021		
			About Page				3/17/2021		
33			Sensor Pod Firmware	28 days	Mon 1/11/21	Sun 2/7/21	3/17/2021		Ross Klonowski
			PIC to RN2903 Communication				2/6/2021		
			Gateway Transmission				2/14/2021		
			Sleep Cycle				3/4/2021		
			Sensor Data Acquisition				2/6/2021		

Figure 151: Actual Gantt Chart (1).



	Startup Sequence				3/17/2021	
34	**Revise Software**	**14 days**	**Mon 2/8/21**	**Sun 2/21/21**	**4/7/2021**	
35	Website Frontend	14 days	Mon 2/8/21	Sun 2/21/21	3/31/2021	Ross Klonowski
36	Website Backend	14 days	Mon 2/8/21	Sun 2/21/21	4/7/2021	Alexis Alves
37	Sensor Pod Firmware	14 days	Mon 2/8/21	Sun 2/21/21	3/24/2021	Ross Klonowski
38	PIC to RN2903	14 days	Mon 2/8/21	Sun 2/21/21	N/A	Alexis Alves
39	**MIDTERM: Demonstrate Software Subsystem**	**5 days**	**Mon 2/22/21**	**Wed 2/24/21**	**2/24/2021**	
	Website	**5 days**	**Mon 2/22/21**	**Wed 2/24/21**	**2/24/2021**	Alexis Alves
	Sensor Pod Communications	**5 days**	**Mon 2/22/21**	**Wed 2/24/21**	**2/24/2021**	Ross Klonowski
40	**SDC & FA Software Approval**	**0 days**	**Sat 2/27/21**	**Sat 2/27/21**	**2/24/2021**	
41	**System Integration**	**42 days**	**Sat 2/27/21**	**Fri 4/8/21**	**4/8/2021**	
42	**Assemble Complete System Integration**	**14 days**	**Sat 2/27/21**	**Fri 3/12/21**	**3/25/2021**	
43	Website	14 days	Sat 2/27/21	Fri 3/12/21	3/25/2021	Alexis Alves
44	Sensor Pod Firmware	14 days	Sat 2/27/21	Fri 3/12/21	3/21/2021	Ross Klonowski/Andrea Wyder
45	**Test Complete System Integration**	**7 days**	**Sat 3/13/21**	**Fri 3/19/21**	**4/1/2021**	
46	Website	7 days	Sat 3/13/21	Fri 3/19/21	3/31/2021	Alexis Alves
47	Sensor Pod Firmware	7 days	Sat 3/13/21	Fri 3/19/21	3/24/2021	Ross Klonowski
48	**Revise Complete System Integration**	**16 days**	**Sat 3/20/21**	**Sun 4/4/21**	**4/6/2021**	
49	Website	16 days	Sat 3/20/21	Sun 4/4/21	4/6/2021	Alexis Alves
50	Sensor Pod Firmware	16 days	Sat 3/20/21	Sun 4/4/21	3/24/2021	Ross Klonowski/Andrea Wyder
51	**Demonstration of Complete System**	**5 days**	**Mon 4/5/21**	**4/8/2021**	**4/8/2021**	ALL
	Soil Moisture/Temperature		**Mon 4/5/21**	**4/8/2021**	**4/8/2021**	Andrea Wyder/Alexis Alves/Ross Klonowski
	Distance Testing		**Mon 4/5/21**	**4/8/2021**	**4/8/2021**	ALL
	3D Models		**Mon 4/5/21**	**4/8/2021**	**4/8/2021**	Andrea Wyder
	Website (w/ Trends)		**Mon 4/5/21**	**4/8/2021**	**4/8/2021**	Alexis Alves
	Power Management System		**Mon 4/5/21**	**4/8/2021**	**4/8/2021**	Ross Klonowski
52	**Develop Final Report**	**103 days**	**Mon 1/11/21**	**Fri 4/23/21**	**4/23/2021**	Andrea Wyder/Alexis Alves/Ross Klonowski
53	Write Final Report	103 days	Mon 1/11/21	Fri 4/23/21	4/16/2021	Andrea Wyder/Alexis Alves/Ross Klonowski
54	Submit Final Report	0 days	Fri 4/23/21	Fri 4/23/21	4/16/2021	Andrea Wyder
55	Spring Recess	7 days	Mon 4/12/21	Sun 4/18/21	4/18/2021	
56	**Project Demonstration and Presentation**					ALL

Figure 152: Actual Gantt Chart (2).

11 Conclusions and Recommendations

Monitoring water management for irrigation systems is an unresolved issue the farming industry has struggled with for a long time. The proposed solution is to create an affordable Sensor Pod WSN that consists of a unit that can be “planted” with the crops and wirelessly transmit data through LoRaWAN communication. A capacitive soil moisture sensor was designed by using a timer circuit and multi-parallel plate PCB capacitor. The data collected was sent to the microprocessor through an analog input, which was then packaged by the LoRa module and transmitted to the Gateway through a quarter wavelength monopole antenna. The Gateway sends the information to the Database where it can be analyzed, trended, and stored, and then be displayed visually through the web application so that farmers can have better water management.

For the development phase, the group committed to the Engineering and Marketing Requirements listed in this paper. Development was planned by using a Gantt Chart and Azure DevOps platform to ensure the Wireless Sensor System is completed in a timely manner. The overall design of the Sensor Pod was focused on power efficiency to increase battery life to ensure the Sensor Pod would last the entire growing season. To verify that the pods could last the required length of time, a power consumption analysis was performed. The results showed that the pods would last longer than the expected duration taking sleep, active, and transmit modes into consideration.

To complete the soil moisture sensor, capacitors were chosen to satisfy the required 80% accuracy. For the mechanical design, a force calculation was completed to determine the thickness of the walls of the Sensor Pod, and models of the Pods were 3D printed. To progress the software development, a prototype subsystem for gathering sensor data from the two sensors was constructed. Furthering the web application, an Alpha version for the backend web server along with a DynamoDB Cloud database for the data storage was designed. An Alpha version of the front end allowed farmers to interface with the data collected and view trend. The interface also monitored soil and battery conditions to provide alerts when the conditions exceeded threshold limits as well as Sensor Pod status. To support and verify the Communications Analysis with regards to range, a distance test was performed to verify the stated range and planting depth were possible within the conditions defined in the analysis.

Moving forward, a few design changes should be implemented if the Soil Senor Network design is to be manufactured and sold. On the main PCB, the battery system discussed should be incorporated so that accurate power dissipation readings can be monitored. Also on the main PCB, a GPS circuit should be implemented for the retrieval process so that the farmer can easily find the Sensor Pod before the battery depletes. Another electrical design change is to add a power switch to the bottom of the Sensor Pod so the farmer can turn the pod off at the end of each growing season without having to disconnect the battery. On the software side, a QR detection code should be implemented so that when the farmer logs into his farm through the mobile application, the Sensor Pod can be scanned and automatically listed as one of the pods on the farm without a manual entry. The last design addition is to give the farmers the option to purchase an “automatic planting” kit to decrease the time it takes the farmer to install the pods in the field.

With well-defined engineering and marketing requirements that were created at the beginning of the project, the Sensor Pod was continuously developed to be a low-cost and accessible alternative to what farmers have available to them on the market today. Considering all implemented and proposed designs for the Wireless Sensor Pod, the device becomes a marketable product for irrigation management, regardless of the size of the farm.

12 Acknowledgements

The authors would like to thank Dr. Nathan Ida for technical assistance and expertise in antenna communications, signal propagation, and sensor design, Dr. Kye-Shin Lee for technical assistance in capacitive soil moisture sensor design, Dr. Hamid Bahrami for academic advisement, Professor Gregory Lewis for project oversight and guidance, Mr. Erick Rinaldo for parts orders, Mr. Max Fightmaster for PCB assembly guidance and 3D model construction, and Jacob Lenart for 3D modeling assistance.

13 WORKS CITED

[1] Hrozencik, Aaron. “Irrigation & Water Use.” *USDA ERS - Irrigation & Water Use*, 23 Sept. 2019, www.ers.usda.gov/topics/farm-practices-management/irrigation-water-use/#definitions.

[2] “Smart Agriculture Sensors: Helping Small Farmers and Positively Impacting Global Issues, Too.” *Mouser Electronics - Electronic Components Distributor*, www.mouser.com/applications/smart-agriculture-sensors/.

[3] Manimaran, P., and Yasar Arfath. *An Intelligent Smart Irrigation System Using WSN and GPRS Module*. *An Intelligent Smart Irrigation System Using WSN and GPRS Module*.

[4] Kumar, Yugal, and Divyansh Thakur. *Applicability of Wireless Sensor Networks in Precision Agriculture: A Review*. *Applicability of Wireless Sensor Networks in Precision Agriculture: A Review*.

[5] Tokitsu, Hiroshi, et al. *Fertigation System, Fertigation Control Server, Salts Accumulation Determination Method, and Soil EC Sensor*. 18 Feb. 2020.

[6] Ersavas, Bulut F, et al. *METHODS AND SYSTEMS FOR IRRIGATION CONTROL*. 26 Jan. 2016.

[7] Masruroh, Siti Ummi, et al. *Performance Evaluation of Instant Messenger in Android Operating System and IPhone Operating System*. *Performance Evaluation of Instant Messenger in Android Operating System and IPhone Operating System*.

[8] Peters, Troy, and Kefyalew Desta. *Practical Use of Soil Moisture Sensors and Their Data for Irrigation Scheduling*. 2013, *Practical Use of Soil Moisture Sensors and Their Data for Irrigation Scheduling*.

[9] Sample, David J, and James S Owen. Understanding Soil Moisture Sensors: A Fact Sheet for Irrigation Professionals in Virginia. Understanding Soil Moisture Sensors: A Fact Sheet for Irrigation Professionals in Virginia.

[10] Kasalica, B., et al. "Effect of a High DC Electric Field on Plant Leaves Reflectivity." *Taylor & Francis Online*, 24 Feb. 2007, www.tandfonline.com/doi/abs/10.1080/00207230108711337.

[11] Adla, Soham, et al. "Laboratory Calibration and Performance Evaluation of Low-Cost Capacitive and Very Low-Cost Resistive Soil Moisture Sensors." *US National Library of Medicine National Institutes of Health*, 8 Jan. 2020, www.ncbi.nlm.nih.gov/pmc/articles/PMC7014303/.

[12] Ida, Nathan. *Engineering Electromagnetics*. Springer, 2015.

[13] Mohan, Vivek. "10 Things About LoRaWAN & NB-IoT." *Inside Out*, blog.semtech.com/title-10-things-about-lorawan-nb-iot.

[14] "LoRaWAN What Is It?" LoRa Alliance, Nov. 2015.

[15] Valerio, Pablo. "Top Wireless Standards for IoT Devices." *IoT Times*, 14 Nov. 2019, iot.eetimes.com/top-wireless-standards-for-iot-devices/.

[16] "PIC18F2525/2620/4525/4620 Data Sheet." *Microchip*, May 2008, ww1.microchip.com/downloads/en/devicedoc/39626e.pdf.

[17] "16-Bit Flash Microcontrollers with Dual Partition Flash Memory, XLP, LCD, Cryptographic Engine and USB On-The-Go." *Microchip*, Nov. 2019, ww1.microchip.com/downloads/en/DeviceDoc/PIC24FJ256GA412-GB412-Family-Data-Sheet-DS30010089E.pdf.

[18] Mary Dunckel, Michigan State University Extension. "Small, Medium, Large – Does Farm Size Really Matter?" *MSU Extension*, 2 Oct. 2018, www.canr.msu.edu/news/small_medium_large_does_farm_size_really_matter.

[19] "Low-Power Long Range LoRa® Technology Transceiver Module." *Microchip*, Jan. 2018, ww1.microchip.com/downloads/en/DeviceDoc/50002390E.pdf.

[20] "TLC555 LinCMOS™ Timer." *Texas Instruments*, www.ti.com/lit/ds/symlink/tlc555.pdf?HQS=TI-null-null-digikeymode-df-pf-null-wwe&ts=1603553851426.

[21] "Linear and Switching Voltage Regulators – An Introduction." *PREDICTABLE DESIGNS*, 23 Nov. 2020, predictabledesigns.com/linear-and-switching-voltage-regulators-introduction/.

[22] "Ada, Lady. "Li-Ion & LiPoly Batteries." *Adafruit Learning System*, learn.adafruit.com/li-ion-and-lipoly-batteries/voltages.

[23] "Acrylonitrile Butadiene Styrene (ABS) and Its Features." *Acrylonitrile Butadiene Styrene (ABS Plastic): Uses, Properties & Structure*, omnexus.specialchem.com/selection-guide/acrylonitrile-butadiene-styrene-abs-plastic.

[24] Jim Patrico, Progressive Farmer Senior Editor. "Planter Speeds: How Fast Is Too Fast? - DTN." *AgFax*, 13 Mar. 2014, agfax.com/2014/03/13/planter-speeds-fast-fast-dtn/.

14 APPENDIX

14.1 FRONTEND MODELS

14.1.1 Farm Overview

14.1.1.1 *Type Script Component:*

```
import { HttpClient } from '@angular/common/http';
import { Component, Input, OnInit } from '@angular/core';
import { DataBaseCRUDInterfaceService } from '../services/DataBaseCRUDInterface.service';

@Component({
  selector: 'app-farmOverview',
  templateUrl: './farmOverview.component.html',
  styleUrls: ['./farmOverview.component.scss']
})
export class FarmOverviewComponent implements OnInit {

  @Input() farmId: string;

  farmInformations: FarmInfo;
  farmStatus: FarmStatus;
  farmGraph: Graph;
  options:any;

  podList: any;

  constructor(private _http: HttpClient, _Data: DataBaseCRUDInterfaceService) {
    this.getFarmInfo();
    this.getFarmStatus();
    this.getFarmGraphData();
    this.options = {
      tooltip: {
        trigger: 'axis',
        axisPointer: {
          type: 'line',
          label: {
            backgroundColor: '#6a7985'
          }
        }
      },
      legend: {
        data: this.farmGraph.top5Pods,
        textStyle: {
```

```
        color: "#ffffff"
      }
    },
    grid: {
      left: '3%',
      right: '4%',
      bottom: '3%',
      containLabel: true
    },
    xAxis: [
      {
        type: 'category',
        boundaryGap: false,
        data: ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
      }
    ],
    yAxis: [
      {
        type: 'value',
        show: false
      }
    ],
    series: [
      {
        name: this.farmGraph.top5Pods[0],
        type: 'line',
        stack: 'counts',
        areaStyle: { normal: {} },
        data: this.farmGraph.podsData[0]
      },
      {
        name: this.farmGraph.top5Pods[1],
        type: 'line',
        stack: 'counts',
        areaStyle: { normal: {} },
        data: this.farmGraph.podsData[1]
      },
      {
        name: this.farmGraph.top5Pods[2],
        type: 'line',
        stack: 'counts',
        areaStyle: { normal: {} },
        data: this.farmGraph.podsData[2]
      },
      {
        name: this.farmGraph.top5Pods[3],
        type: 'line',
        stack: 'counts',
        areaStyle: { normal: {} },
        data: this.farmGraph.podsData[3]
      },
      {
        name: this.farmGraph.top5Pods[4],
        type: 'line',
        stack: 'counts',
        areaStyle: { normal: {} },
        data: this.farmGraph.podsData[4]
      }

    ]
  };
}

ngOnInit() {
}

getFarmInfo() {
  this._http.get('https://localhost:44385/DynamoDB/Farm?id='
                  + this.farmId +
                  '&hash=1').subscribe(
    result => {
      let data: any = result;
      let itemData = {} as FarmInfo;

      for (let element of data) {
        if (element.devEui != 1) {
          this.podList.push(element.name)
        }
        else {
          itemData.Name = element.name;
          itemData.Location = element.location;
          itemData.Owner = element.owner;
          itemData.Size = element.size;
          this.farmInformations = itemData;
        }
      }

    }
  );
}
getFarmStatus() {
  this._http.get('https://localhost:44385/DynamoDB/Status/Farm?id=' + this.farmId + '&hash=1').subscribe
(
    result => {
      let data: any = result;
      let itemData = {} as FarmStatus;

      for (let element of data) {
        if (element.devEui != 1) {
          this.podList.push(element.name)
        }
        else {
          itemData.avgTemp = element.avgTemp;
          itemData.avgMoisture = element.avgMoisture;
          itemData.pods = element.pods;
          itemData.tWarning = element.tWarning;
          itemData.mWarning = element.mWarning;
          itemData.bWarning = element.bWarning;
          this.farmStatus = itemData;
        }
      }

    }
  );
}
getFarmGraphData(){
  this._http.get('https://localhost:44385/DynamoDB/Status/Farm?id=' + this.farmId + '&hash=1').subscribe
(
    result => {
      let data: any = result;
      let graphData = {} as Graph;
      graphData.top5Pods =  data.pods;
      graphData.podsData =  data.podData;
      }
  );
}

}
interface FarmInfo {
  Name: string;
  Location: string;
  Owner: string;
  Size: string;
}
```

```
interface FarmStatus {
  avgTemp: number;
  avgMoisture: number;
  pods: number;
  tWarning: number;
  mWarning: number;
  bWarning: number;
}

interface Graph {
  top5Pods: string[];
  podsData: any;
}
```

14.1.1.2 *HTML Component*

```
<div class="container-fluid">
  <div class="row h-10">
    <div class="col-md-8">
      <nb-card size="tiny">
        <nb-card-header>Farm Information</nb-card-header>
        <nb-card-body>
          <p>Farm Name: {{this.farmInformations.Name}}</p>
          <p>Owner: {{this.farmInformations.Owner}}</p>
          <p>Location: {{this.farmInformations.Location}}</p>
          <p>Size: {{this.farmInformations.Size}}</p>
        </nb-card-body>
      </nb-card>
    </div>
    <div class="col-md-4">
      <nb-card size="tiny">
        <nb-card-header>Farm Status</nb-card-header>
        <nb-card-body>
          <p>Avg Moisture: {{this.farmStatus.avgTemp}}</p>
          <p>Avg Temp: {{this.farmStatus.avgMoisture}}</p>
          <p>Pod Connected:{{this.farmStatus.pods}}</p>
          <p>Warnings:
             Temp: {{this.farmStatus.tWarning}}
           | Moisture: {{this.farmStatus.mWarning}}
           | Battery: {{this.farmStatus.bWarning}}</p>
        </nb-card-body>
      </nb-card>
    </div>
  </div>
  <div class="row flex-row">
```

```
    <div class="col-md-8" height="17rms">
      <nb-card size="large">
        <nb-card-header>Moisture</nb-card-header>
        <div echarts [options]="options" theme="macarons" class="chart"></div>
      </nb-card>
    </div>
    <div class="col-md-4">
      <nb-card size="large">
        <nb-card-header>Podlist</nb-card-header>
        <nb-list>
          <nb-list-item *ngFor="let device of this.podList">
            {{ device }}
          </nb-list-item>
        </nb-list>
      </nb-card>
    </div>
  </div>
</div>
```

14.1.2 Pod List

14.1.2.1 *Type Script Component*

```
import { HttpClient } from '@angular/common/http';
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';
import { NbSortDirection, NbSortRequest, NbTreeGridDataSource, NbTreeGridDataSourceBuilder } from '@nebula
r/theme';
import { DataBaseCRUDInterfaceService } from '../services/DataBaseCRUDInterface.service';

@Component({
  selector: 'app-sensorPodList',
  templateUrl: './sensorPodList.component.html',
  styleUrls: ['./sensorPodList.component.scss']
})
export class SensorPodListComponent implements OnInit {
  customColumn = 'ReadTime';
  testing: string;
  defaultColumns = ['Pod','Battery','Moisture','Temp','Warning' ];
  allColumns = [ this.customColumn, ...this.defaultColumns ];

  dataSource: NbTreeGridDataSource<FSEntry>;

  sortColumn: string = 'ReadTime';
  sortDirection: NbSortDirection = NbSortDirection.DESCENDING;
  list: string[];
  data: TreeNode<FSEntry>[];
  distinctArray: any;

  podStatusList: PodStatus[];

  tmpMax = 100;
  tmpMin = 0;
  moistureMax = 90;
  moistureMin = 30;


  private _dataSourceBuilder: NbTreeGridDataSourceBuilder<FSEntry>;



  constructor(private router:Router, private route:ActivatedRoute , private dataSourceBuilder: NbTreeGridD
ataSourceBuilder<FSEntry>,private _http: HttpClient, _Data: DataBaseCRUDInterfaceService) {
    this._dataSourceBuilder = dataSourceBuilder;
    this.getPodData();
   }

  ngOnInit() {

  }

  onReload(){
    this.getPodData();
   }
   changeSort(sortRequest: NbSortRequest): void {

  }

  getDirection(column: string): NbSortDirection {
    return NbSortDirection.DESCENDING;
  }
  getPodData()
  {
    this._http.get('https://localhost:44385/DynamoDB/PodData'  + this.farmId)

  .subscribe(

      result => {
        let tempdata: any =  result;
        let tempData: TreeNode<FSEntry>[] = [];
        for (let element of tempdata) {
          let itemData = {} as TreeNode<FSEntry>;
          let item = {} as FSEntry;

          item.Pod = element.name;
          item.Temp = element.temp_Sensor_Value + '%';
          item.Eui = element.devEui + ' °F';
          item.ReadTime = this.formatTime(element.time);
          item.Moisture = element.moisture_Sensor_Value;
          item.Battery = element.bat_Value
          item.Warning = this.checkWarning(element.temp_Sensor_Value,
                                           element.moisture_Sensor_Value,
                                           element.bat_Value)
          itemData.data = item;
          tempData.push(itemData);
        }
        this.data = tempData;
        this.dataSource = this.dataSourceBuilder.create(this.data);
      });
    this._http.get('https://localhost:44385/DynamoDB/Farm/AllPods').subscribe(
      result => {
        let data: any =  result;
        let itemData = {} as PodStatus;

        for (let element of data) {
          itemData.Name = element.name;
          itemData.DeviceEui = element.devEui;
          itemData.statusBattery = element.battery;
          itemData.statusConnection = element.connection;
          this.podStatusList.push(itemData);
        }
      });
  }

  formatTime(time: string)
  {
    var tmpDate: Date = new Date(time)
    var localtime: string = "" + tmpDate.getHours() + ":" + tmpDate.getMinutes() + ":" + tmpDate.getSecond
s();
    return localtime;
  }

  checkWarning(tmp: number,moisture: number,bat: string){
    var warningmsg: string = '';
    var list: string[]= [];
    if(tmp > this.tmpMax || tmp < this.tmpMin)
```

```
      list.push("Temp");
    if(moisture > this.moistureMax || moisture < this.moistureMin)
      list.push("Moisture");
    if(bat == 'Low' )
      list.push("Battery");
    if(list.length == 0)
    {
      warningmsg = "-"
    }
    else
    {
      list.forEach(element => {
        warningmsg += element;
        if(element != list[list.length-1])
        warningmsg += ', '
      });
      warningmsg += " Warning";
    }
    return warningmsg;
  }

}

interface TreeNode<T> {
  data: T;
  children?: TreeNode<T>[];
  expanded?: boolean;
}

interface PodStatus {
  Name: string;
  DeviceEui: string;
  statusBattery: string;
  statusConnection: string;
}

interface FSEntry {
  ReadTime: string;
  Pod: string;
  Eui: string;
  Data: string;
  Battery: string;
  Moisture: string;
  Temp: string;
  Warning: String;
```

```
}
```

14.1.2.2 *HTML Component:*

```
<nb-card size="tiny">
  <nb-card-header>Pod Status List</nb-card-header>
  <nb-list>
    <nb-list-item *ngFor="let device of this.podStatusList">
      " {{ device.Name }}: {{ device.DeviceEui }}
      | Battery: {{ device.statusBattery }}
      | Status: {{ device.statusConnection }}"
    </nb-list-item>
  </nb-list>
</nb-card>
<nb-card size="giant">
  <nb-card-header>
    Pod Data
    <button (click)="onReload()" nbButton>Refresh</button>
  </nb-card-header>

  <div overflow-y=scroll>
    <nb-card>
      <nb-card-body>
        <table [nbTreeGrid]="data" nbSort (sort)="changeSort($event)" equalColumnsWidth>

          <tr nbTreeGridHeaderRow *nbTreeGridHeaderRowDef="allColumns"></tr>
          <tr nbTreeGridRow *nbTreeGridRowDef="let row; columns: allColumns"></tr>

          <ng-container [nbTreeGridColumnDef]="customColumn">
            <th nbTreeGridHeaderCell [nbSortHeader]="getDirection(customColumn)" *nbTreeGridHeaderCellDef>
              {{customColumn}}
            </th>

            <td nbTreeGridCell *nbTreeGridCellDef="let row">

              <nb-tree-grid-row-toggle [expanded]="row.expanded" *ngIf="row.data.kind === 'dir'">
              </nb-tree-grid-row-toggle>

              {{row.data.ReadTime}}

            </td>
          </ng-container>

          <ng-container *ngFor="let column of defaultColumns" [nbTreeGridColumnDef]="column">
            <th nbTreeGridHeaderCell [nbSortHeader]="getDirection(column)" *nbTreeGridHeaderCellDef>
              {{column}}
            </th>

            <td nbTreeGridCell *nbTreeGridCellDef="let row">
              {{row.data[column]}}
            </td>
          </ng-container>

        </table>

      </nb-card-body>
    </nb-card>
  </div>
```

14.1.3 Home

14.1.3.1 *Type Script Component*

```
import { HttpClient } from '@angular/common/http';
import { Component, Output } from '@angular/core';
import { Router } from '@angular/router';
import { delay } from 'rxjs/operators';
import { DataBaseCRUDInterfaceService } from '../services/DataBaseCRUDInterface.service';

@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.scss']
})
export class HomeComponent {
  @Output() farmId: string;
  @Output() validFarmID: boolean;

  status = "primary"
  constructor(private _http: HttpClient, _Data: DataBaseCRUDInterfaceService, router: Router) {
  }
  getFarmId() {
    this._http.get('https://localhost:44385/dynamodb/Farm?id='
      + this.farmId +
      '&hash=1').subscribe(
        async result => {
          let data: any = result;
          if (result == null)
            status = "error"
          else {
            status = "Success";
            this.farmId = result;
            this.validFarmID = true;
            await delay(5);
            this.router.navigateByUrl('/farm-overview')
          }
        }
      );
  }
}
```

14.1.3.2 *HTML Component*

```
<!-- Presentation Only -->
<!-- <h1>Soil Sensor Monitoring (Design Team 7)</h1>
<p>Team Members:</p>
<ul>
  <li>Alexis Alves</li>
  <li>Andrea Wyder</li>
  <li>Luke M Farnsworth</li>
  <li>Ross Klonowski</li>
</ul> -->
<!-- <nb-card  >
  <nb-card-header>
  </nb-card-header>
  <nb-card-body>

  </nb-card-body>
</nb-card> -->
<div class="container">
  <nb-card status={{this.status}} size="tiny" display="flex" align-items="center" justify-
content="center">
    <nb-card-header text-align="center">Sign in</nb-card-header>
    <nb-card-body class="items-col">
      <div class="align-content-center">
        <input type="text" nbInput fullWidth shape="round" placeholder="Farm ID">
      </div>
    </nb-card-body>

    <nb-card-footer>
      <div class="col text-center">
        <button class="btn btn-primary" (click)="getFarmId()">Login</button>
      </div>
    </nb-card-footer>
  </nb-card>
</div>
```

14.1.4 About

14.1.4.1 *Type Script Component*

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-about',
  templateUrl: './about.component.html',
  styleUrls: ['./about.component.scss']
})
export class AboutComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}
```

14.1.4.2 *HTML Component*

```
<h1>Soil Sensor Monitoring (Design Team 7)</h1>
<p>Built with:</p>
<ul>
  <li><a href='https://get.asp.net/'>ASP.NET Core</a> and <a
      href='https://msdn.microsoft.com/en-us/library/67ef8sbd.aspx'>C#</a> for cross-platform server-
side code</li>
  <li><a href='https://angular.io/'>Angular</a> and <a href='http://www.typescriptlang.org/'>TypeScript</a
> for
    client-side code</li>
  <li><a href='https://akveo.github.io/'>Nebular</a> for layout and styling</li>
  <li><a href='https://aws.amazon.com/DynamoDB/'>DynamoDB</a> for the cloud Database</li>
  <li><a href='https://aws.amazon.com/api-
gateway/'>AWS API Gateway</a> for a Fowarder Endpoint for Senet</li>
  <li><a href='https://aws.amazon.com/lambda/'>Lamda</a> for Processing and Modling AWS API data to Databa
se</li>
  <li><a href='https://xieziyu.github.io/ngx-echarts/#/basic/basic-usage'>ngx-echarts</a> for Charts</li>
</ul>
```

14.2 Backend Models

14.2.1 Domain Models

14.2.1.1 *Farm Table Data*

```
namespace SoilSensor.Models.DomainModels
{
    [DynamoDBTable("Farm_Test_1")]
    public class Farm
    {
        [DynamoDBProperty]
        [DynamoDBHashKey]
        public string FarmId { get; set; }
        [DynamoDBRangeKey]
        public string DevEui { get; set; }
        // Property only assigned to DevEui - FarmInformation
        [DynamoDBProperty]
        public string Name { get; set; }
        [DynamoDBProperty]
        public string Location { get; set; }
        [DynamoDBProperty]
        public string Owner { get; set; }
        [DynamoDBProperty]
        public string Size { get; set; }
        // Property assigned all DevEui for pods
        [DynamoDBProperty]
        public string PodName { get; set; }
        [DynamoDBProperty]
        public string Status { get; set; }
    }
}
```

14.2.2 Sensor Pod Data

```
namespace SoilSensor.Models.DomainModels
{
    [DynamoDBTable("Sensor_Data_Test")]
    public class PodData
    {
        [DynamoDBProperty]
        [DynamoDBHashKey]
        public string DevEui { get; set; }
        [DynamoDBProperty]
        public string ReadTime { get; set; }
        [DynamoDBProperty]
        public string Moisture_Sensor_Value { get; set; }
        [DynamoDBProperty]
        public string Temp_Sensor_Value { get; set; }
        [DynamoDBProperty]
        public string Bat_Value { get; set; }
        [DynamoDBProperty]
        public string Data { get; set; }

    }
}
```

14.2.3 Controller

14.2.3.1 *API Router*

```
namespace SoilSensorWeb.Controllers
{
     [ApiController]
    [Route("[controller]")]
    public class DynamoDBController : ControllerBase
    {
        private IDynamoDBContext<Farm> _farmContext;
        private IDynamoDBContext<PodData> _podDataContext;


        public DynamoDBController(IDynamoDBContext<Farm> farmContext, IDynamoDBContext<PodData> podDataCon
text)
        {
            _farmContext = farmContext;
            _podDataContext = podDataContext;
        }

        [HttpGet]
        [Route("Farm")]
        public async Task<Farm> GetUserAsync([FromQuery] string id, [FromQuery] string hash)
        {
            try
            {
                return await _farmContext.GetByIdAsync(id,hash);
            }
            catch (Exception ex)
            {
                throw new Exception($"Amazon error in GetUser table operation! Error: {ex}");
            }
        }

        [HttpGet]
        [Route("Farm/All")]
        public async Task<List<Farm>> All(string id,string paginationToken = "")
        {
            try
            {
                return await _farmContext.GetaAll(id);
            }
            catch (Exception ex)
            {
                throw new Exception($"Amazon error in GetUser table operation! Error: {ex}");
            }
        }

        [HttpGet]
        [Route("Farm/AllPods")]
        public async Task<List<Farm>> AllPods(string paginationToken = "")
        {
            try
            {
                return await _farmContext.GetaAllPods();
            }
            catch (Exception ex)
            {
                throw new Exception($"Amazon error in GetUser table operation! Error: {ex}");
            }
        }


        [HttpGet]
        [Route("PodData")]
        public async Task<List<PodData>> GetPodData([FromQuery] string id,string paginationToken = "")
        {
            try
            {
                return await _podDataContext.GetaAll(id);
            }
            catch (Exception ex)
            {
                throw new Exception($"Amazon error in GetUser table operation! Error: {ex}");
            }
        }
    }
}
```

14.2.3.2 *API Controller Class*

```
namespace SoilSensor.Data.Controllers
{
    public class DynamoDBContext<T> : DynamoDBContext, IDynamoDBContext<T>
    where T : class
    {
        private DynamoDBOperationConfig _config;
        private string _tableName;

        public DynamoDBContext(IAmazonDynamoDB client, string tableName)
            : base(client)
        {
            _tableName = tableName;
            _config = new DynamoDBOperationConfig()
            {
                OverrideTableName = tableName
            };
        }

        public async Task<T> GetByIdAsync(string id, string devEui)
        {
            return await base.LoadAsync<T>(id, devEui, _config);
        }

        public async Task SaveAsync(T item)
        {
            await base.SaveAsync(item, _config);
        }

        public async Task DeleteByIdAsync(T item)
        {
            await base.DeleteAsync(item, _config);
        }

        public string GetTable()
        {

            return _tableName;
        }

        public async Task<List<T>> GetaAll(string id)
        {
            var scanConditions = new List<ScanCondition>() {
                new ScanCondition("FarmId", ScanOperator.Equal,id)
            };
            var searchResults = base.ScanAsync<T>(scanConditions, null);
            return await searchResults.GetNextSetAsync();
        }
        public async Task<List<T>> GetaAllPods()
        {
            var scanConditions = new List<ScanCondition>()
            {
                new ScanCondition("DevEui", ScanOperator.NotEqual, "1"),
            };
            var searchResults = base.ScanAsync<T>(scanConditions, null);
            return await searchResults.GetNextSetAsync();
        }
    }

}
```

14.2.3.3 *API Controller Data Type*

```
namespace SoilSensor.Data
{
    public class DynamoDBOptions
    {
        public string Farm { get; set; }
        public string PodData { get; set; }
    }
}
```

14.2.3.4 *API Controller Interface*

```
namespace SoilSensor.Data.Interface
{
    public interface IDynamoDBContext<T> : IDisposable where T : class
    {
        Task<T> GetByIdAsync(string id , string hash);
        Task SaveAsync(T item);
        Task DeleteByIdAsync(T item);
        string GetTable();
        Task<List<T>> GetaAll(string id);
        Task<List<T>> GetaAllPods(string id);
    }
}
```

14.2.4 Startup Configuration

```
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = new ConfigurationBuilder()
                .SetBasePath(Directory.GetCurrentDirectory())
                .AddJsonFile("appsettings.json")
                .Build();
        }

        public IConfigurationRoot Configuration { get; set; }

        public void ConfigureServices(IServiceCollection services)
        {
            services.AddControllersWithViews();
            services.AddSpaStaticFiles(configuration =>
            {
                configuration.RootPath = "ClientApp/dist";
            });
            Environment.SetEnvironmentVariable("AWS_ACCESS_KEY_ID", Configuration["AWS:AccessKey"]);
            Environment.SetEnvironmentVariable("AWS_SECRET_ACCESS_KEY", Configuration["AWS:SecretKey"]);
            Environment.SetEnvironmentVariable("AWS_REGION", Configuration["AWS:Region"]);
            var awsOptions = Configuration.GetAWSOptions();
            services.AddDefaultAWSOptions(awsOptions);
            var client = awsOptions.CreateServiceClient<IAmazonDynamoDB>();
            var DynamoDBOptions = new DynamoDBOptions();
            ConfigurationBinder.Bind(Configuration.GetSection("DynamoDBTables"), DynamoDBOptions);
            services.AddScoped<IDynamoDBContext<Farm>>(provider => new DynamoDBContext<Farm>(client, Dynam
oDBOptions.Farm));
            services.AddScoped<IDynamoDBContext<PodData>>(provider => new DynamoDBContext<PodData>(client,
 DynamoDBOptions.PodData));

        }

        // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                app.UseExceptionHandler("/Error");
                app.UseHsts();
            }

            app.UseHttpsRedirection();
            app.UseStaticFiles();
            if (!env.IsDevelopment())
            {
                app.UseSpaStaticFiles();
            }

            app.UseRouting();

            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllerRoute(
                    name: "default",
                    pattern: "{controller}/{action=Index}/{id?}");
            });

            app.UseSpa(spa =>
            {
                spa.Options.SourcePath = "ClientApp";
                spa.UseProxyToSpaDevelopmentServer("http://localhost:4200");
            });
        }
    }
}
```