Department of Electrical Engineering

Agricultural Sensor Monitoring Network

Degree Project Report

Prepared by:

Obatosin Obat-Olowu	0680475
Jarod Andrade	0877859
Mohamad Taih	1142568

Supervisors:

Dr. Yushi Zhou

April 17th, 2022

Table of Content

List of Figures

List of Tables

List of Equations

1. Overview

The agricultural sensor monitoring network is a wide-range, low-cost monitoring system for conditions that affect crop growth. This design is being created to aid the ongoing worldwide food crisis, specifically in regions the most impacted by this crisis.

For crops to grow effectively, they need moisture, nutrients and proper temperature. This design will detect these conditions so people to more efficiently grow crops and help alleviate the current worldwide food insecurity.

2. Statement of the problem

Due to the ongoing war in Ukraine, a major agricultural producer, as well as climate change and rapidly rising inflation, there is currently a massive worldwide food crisis.

“In just two years, the number of people facing, or at risk of, acute food insecurity increased from 135 million in 53 countries pre-pandemic, to 345 million in 82 countries today.” [1]

This project aims to assist with alleviating this problem by increasing farming yields by enabling farmers to better use the resources limited by recent disasters. The intention of this design is to assist farmers in developing countries that may not be able to utilize more expensive farm monitoring equipment.

For example, crop monitoring in North America is done with satellite imaging and large-scale, expensive monitoring systems, which might not be practical for poorer locations, such as those impacted the most by war and drought. This design aims to create a cheaper, portable, and more convenient alternative for these regions.

The cheaper, more portable design could also be used by amateur gardeners who are interested but do not want to pay for a more expensive and involved system. The system can be used to measure the conditions in a backyard vegetable patch from the user's home, plus some of these measurements can be very difficult to take by hand, especially for amateurs. This could help the average Canadian struggling with the current cost of living crisis by increasing the yield of a backyard vegetable garden.

3. System Overview

The overall design for this system consists of two separate parts, a central hub and a sensor node. Several sensor nodes are to be connected to a single central hub to create a wide range of coverage, such as spreading the nodes out throughout a large field so that they can relay information to a single location, like a barn or farmhouse. The nodes are intended to be used outdoors in fields and gardens, as well as in any remote and isolated place where it can be difficult to regularly travel to manually check conditions. Therefore, this system needed to be rugged and low maintenance to endure prolonged periods of outdoor exposure and isolation, while still being low-cost to ensure it could be used by those struggling with food insecurity.

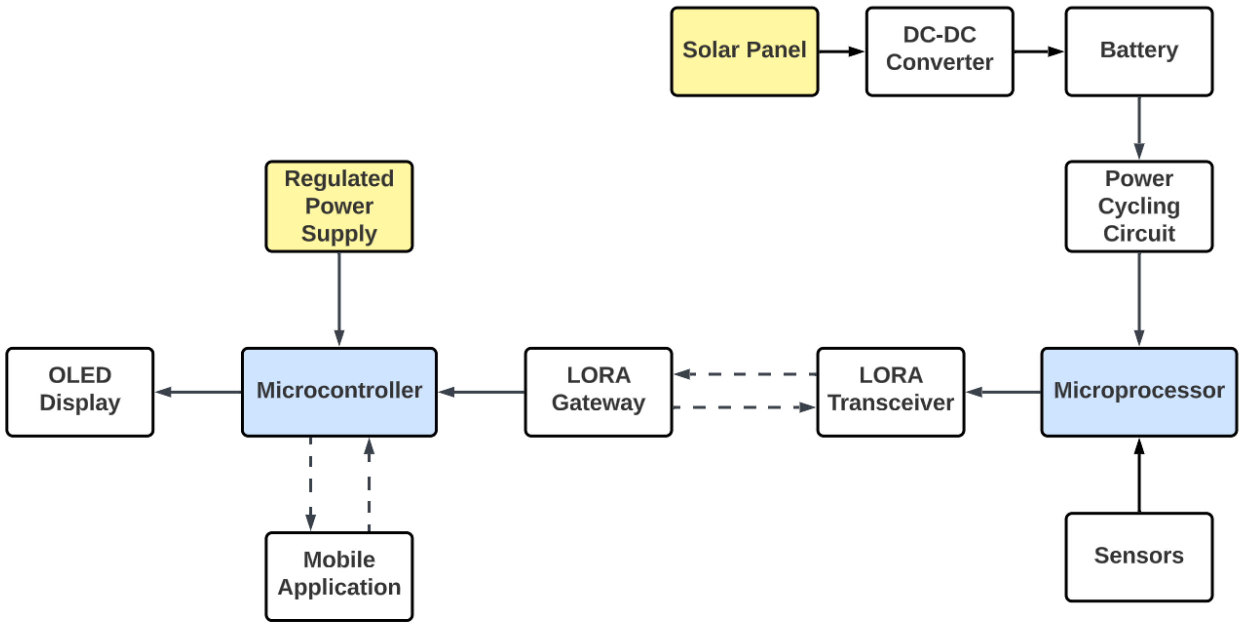


Figure 1: Block Diagram

In order to limit the amount of maintenance, most notably having to change the batteries, the nodes were designed to be as self-contained and low power as possible. This was accomplished by two separate blocks working together, a solar panel to charge an integrated battery and a power cycling circuit designed to cut off current to the microprocessor when it is sitting idle.

A solar panel was decided on as the main power source for the nodes due to their intended long-term outdoor exposure, which provides ample time for solar charging.

The power delivered from the solar panel is controlled by a charging circuit that is responsible for regulating the incoming voltage and current from the solar panel to safely charge the battery and power the rest of the node when the solar panel is active. The battery will then power the node once the solar panel is inactive due to limited sunlight exposure.

Because the sensor node only needs to be active, taking readings and transmitting them for a few seconds at a time and sitting idle the rest of the time, it was decided to shut off the power flowing from the battery to the microprocessor by using a power cycling circuit to minimize the power usage. This results in the design requiring a lower-capacity battery and smaller solar panel, which in turn reduces the overall cost of the nodes.

The microprocessor, when it is turned on by the power cycling circuit, in turn, powers on the attached sensor and LORA transceiver through the 3.3V output pin. The microprocessor then takes readings from the attached sensors over an I2C connection and transmits the collected data using the LORA transceiver.

The base node was designed to be located closer to the barn or farmhouse. It includes a Raspberry Pi 4, an OLED display, and a LoRa transceiver. The measured data from the sensor is received by the central hub, stored, and then displayed on both the OLED display and the Android application. The LoRa gateway utilized is a stripped-down version of a full gateway implemented using an RFM95W transceiver. The OLED display used is a small monochrome 128-by-32 display.

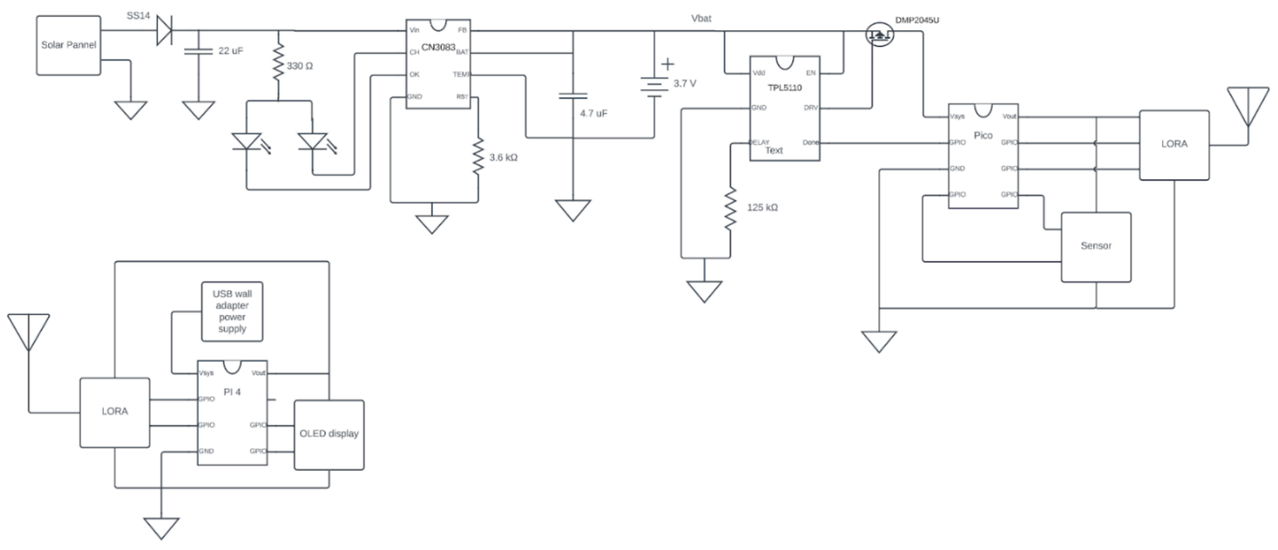


Figure 2: Complete Circuit Diagram

4. Project impacts

This project aims to introduce a cheap farming alternative in the hopes that it encourages self-sustainability as individuals rely less on local markets. It is stated that 75% of Canadians purchase food from one of 5 major chains [2]. Local farming will provide alternative food sources and introduce competition for major chains and markets worldwide, which will encourage prices to decrease. Due to the design being low profile and the nodes being self-sufficient by relying on solar power, the project's design is anticipated to have a minimal negative environmental impact, however, farming and the use of chemical fertilizers are responsible for a substantial amount of global greenhouse gasses and pollutants produced [3]. If farming becomes more accessible, it may create more pollution as farmers will utilize farming aids such as pesticides that can spread pollution to the surrounding environment.

5. Design details for each block

A. Solar panel

Power for the system will be drawn by solar cells, also known as photovoltaic cells (PV). Gathering energy from the sun through PV cells is a portable, renewable solution to powering electrical systems, and coupled with a battery, will keep the system active regardless of sunlight exposure.

The sun emits electromagnetic radiation (EMR) due to its high temperatures that can reach 27 million degrees Fahrenheit [5]. This heat creates a nuclear fusion that splits atoms into their components, which causes protons to collide and generate a large amount of energy that arrives on Earth in the form of photons [6]. Photons come in different wavelengths, most of which is absorbed or reflected by the Earth’s atmosphere before reaching the ground, however, if the wavelength is under the visible spectrum, it will be scattered by the atmosphere before reaching the Earth’s surface [5].

$$E\lambda = \frac{hc}{\lambda}$$

Equation 1: Photon Energy

Equation 1 shows the energy carried by a photon, where h is plank’s constant (6.63*10^(-34) Js), c is the speed of light (3*10^8ms^(-1)) and is the wavelength of the photon.

A solar cell utilizes semiconducting materials that absorb the oncoming photons and convert them to DC power. Silicon is a semiconducting material that is commonly used by solar cells due to its abundance. Silicon is produced by mixing and heating sand with carbon at temperatures nearing 4000 degrees Celsius [6]. This process will create highly purified crystalline silicon which gets cut into thin silicon wafers that are doped with a P-type semiconductor such as

Boron, and an N-type semiconductor such as phosphorus. This doping creates a P-N junction that allows electric flow in the solar cell which can now be used to supply a circuit with solar power [6].

The solar panel selected is a 2.5W 5V, 500mAh Monocrystalline Silicon Solar Panel by Allpowers. This panel can provide a maximum of 0.5 amps of current which is ideal for fast charging of the battery. The dimensions of the panel are 130 by 150mm which must be taken into consideration when designing the protective case.

Solar panels come with an efficiency rating, meaning not all the energy absorbed by the cell is turned to usable power for the circuit.

$$Efficiency(\eta) = \frac{Pm}{Pin}$$

Equation 2: Efficiency of Solar Cell

Equation 2 demonstrates how to find the efficiency of a solar cell, where Pm is the maximum power achieved by the panel and Pin is the radiated power input [3]. The characteristic I-V graph of a solar panel will help illustrate the behaviour of the cell.

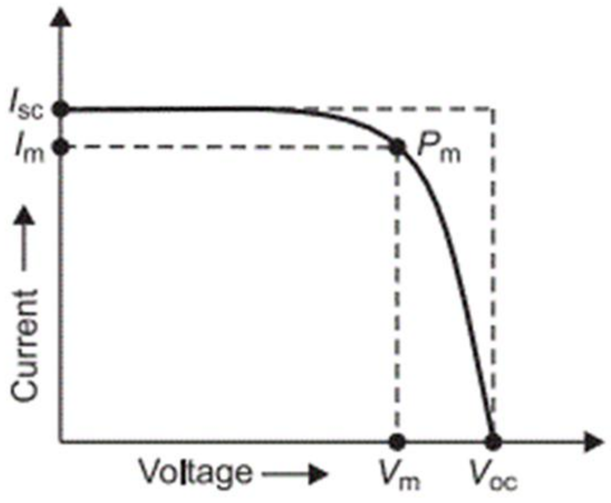


Figure 3: I-V characteristics of Solar Cells [4]

Isc represents the maximum possible current a cell can produce while it has been shorted. Voc is the maximum voltage produced under an open circuit. Since an open circuit gives maximum voltage but no current, and short circuit will generate maximum current but not voltage, power can only be estimated from these values. Im and Vm are the maximum current and voltage at which maximum power occurs and they can be estimated by multiplying Isc or Voc by 0.9 or 0.8 [4].

B. Charging circuit

Since it is unsafe to deliver power from the solar panel directly to the battery, a circuit must be implemented to handle safe delivery and cut-off of voltage and current. There were several circuits explored that could accomplish this task with varying pros and cons.

The circuit considered was a DC-to-DC converter. These converters are also known as switching regulators as they require a switching device to step up (boost) or step down (buck) a supplied DC voltage. It is worth noting that as voltage is stepped down, current will increase and vice versa so voltage will be inversely proportional to current. However, stepping down the voltage is only one of the requirements. There must also be a way to cut off current once the battery has completed its charging cycle to prevent overcharging and damaging the battery. Furthermore, while it is not expected of the solar panel to draw more current than the battery can handle, it would nonetheless be advised to implement a current limiting feature to not supply more current than the battery can handle.

The CN3083 integrated circuit (IC) was chosen for its ability to regulate voltage, account for any changes in input levels to maintain a constant desired output, has automatic recharging and cut-off to not overcharge and damage the battery as well as some other features that will be highlighted.

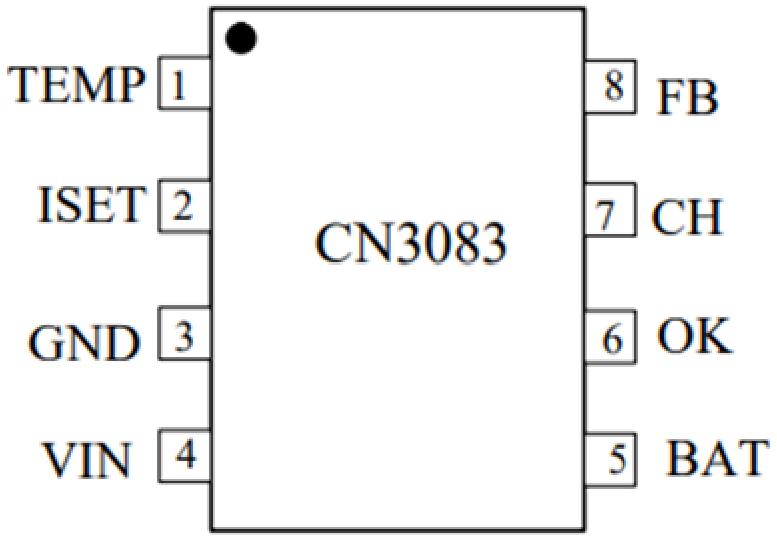


Figure 4: CN3083 pin layout [8]

CN3083 is an 8-pin, surface-mount SOP-8 chip. Pin 1 is the TEMP pin which is responsible for sensing the temperature at which the battery is operating. This is to prevent any overheating and protect the battery from any thermal damage. The battery must come with an NTC thermistor to sense the temperature, otherwise, the pin must be grounded, and the temperature-sensing safety functions will not be in use. Pin 2 is the ISET pin, and it is used in conjunction with a resistor to set the current charging value up to 600mA. Pin 3 connects to the ground and pin 4 to the solar panel. Pin 5 is the BAT pin which connects to the positive terminal of the battery, it can also supply voltage to a different load to not use the battery when sufficient sunlight is present. Pins 6 and 7 are used to display the charging status of the battery and pin 8 is the feedback pin used to ensure constant voltage output [8].

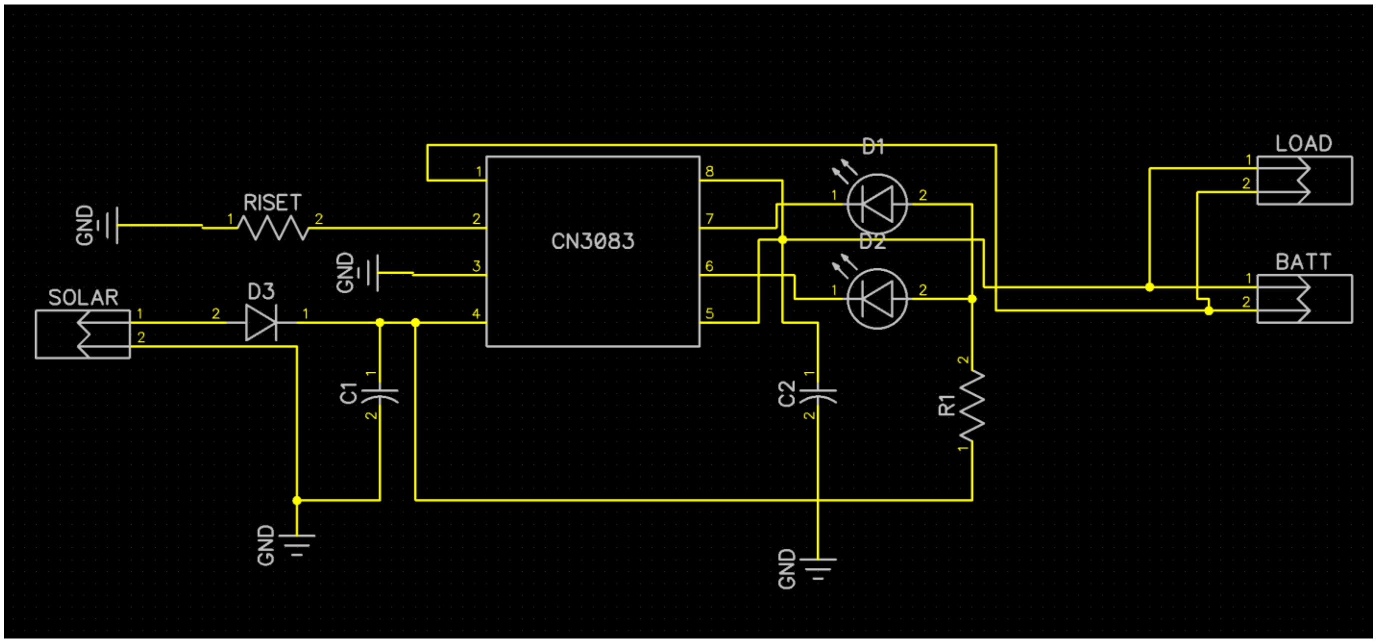


Figure 5: CN3083 Circuit Diagram

Figure 5 shows a circuit diagram for the CN3083 chip in use. A diode D3 is connected between the solar panel and pin 4 of the chip as a safety measure to prevent backflow voltage arriving from the battery to the solar panel when the panel is not operational, saving the panel from any damage that the backflow may cause it. A 22μF bypass capacitor C1 is also utilized to filter out any unwanted AC noise that may be on the DC signal supplied by the solar panel. D1 and D2 are Light emitting diodes (LEDs) used to display when pins 6 or 7 are on. The OK pin 6 is connected to a green LED and once the charge termination is activated, the pin is pulled low by an internal switch and activates the LED, otherwise, the pin is in a high impedance state, preventing its operation. The CH pin 7 works the same way except in reverse where the internal switch is always pulled until the charge termination is active where it remains in a high impedance state. A resistor R1 is connected to the LEDs to prevent them from burning out as they cannot handle any excess voltage past their rated voltage drop of 2 volts.

$$Rvd = Vin - Vf$$

Equation 3: LED Resistor Voltage Drop

$$R = \left(\frac{Rvd}{If}\right)$$

Equation 4: LED Resistor

Rvd is the resistor voltage drop. Vin is the voltage of the supply, that being 5 volts of the solar panel, and Vf is the forward voltage of the LED which is 2 volts. 5-2 will give an Rvd of 3 volts. dividing the Rvd value with the forward current of the LED (10mA) will provide the resistor value at 300 ohms. It is best to go with a slightly higher value in the chance that the solar panel supplies more than its rated voltage. Iset pin 2 is also used with a resistor Rset to impose a maximum charging current value for the battery [8].

$$Ich = \left(\frac{VISET}{RISET}\right) \cdot 900$$

Equation 5: Maximum Current Charging [8]

Most rechargeable batteries can handle a charging current of 500mA, therefore that will be the Ich value used. VISET is the voltage on the Iset pin which is regulated to 2 volts. With these parameters provided the resistor RISET can be found by rearranging the formula to give 3.6kohm. BAT pin 5 is the regulated 4.2 output voltage which will charge the battery. When there is adequate sunlight, the solar panel will power the chip and pin 5 will provide power for the node, however, when the solar panel is not operational, the battery will begin providing the required power. The feedback pin 8 will be connected to pin 5 as well as a stability capacitor C2 to stabilize the feedback loop which will be grounded along with the temperature sensor pin 1 and ground pin 3. A resistor can also be connected to pin 8 to reduce the desired output BAT pin voltage of the chip [8].

$$\text{Vbat} = 4.2 + 3.04 \times 10\text{-}6 \times \text{Rx}$$

Equation 6: Output Voltage Reduction [8]

This chip can only handle a maximum of 6 volts before being damaged, so it is important to only provide an input below that threshold. Furthermore, the operating temperature is from -40° to 85° Celsius, meaning we should not charge the battery at a high current to prevent overheating the chip which will likely be used in warmer climates [8].

C. Battery

There are 2 types of commonly used rechargeable portable batteries, lithium-ion, and lithium-polymer. The key difference between them has to do with safety, as their chemical structures are different. Lithium-ion batteries typically consist of 2 sides, the negative graphite electrode anode, a positive electrode cathode with a separator in between and a nonaqueous liquid electrolyte in which the lithium ions can flow through during discharge [9].

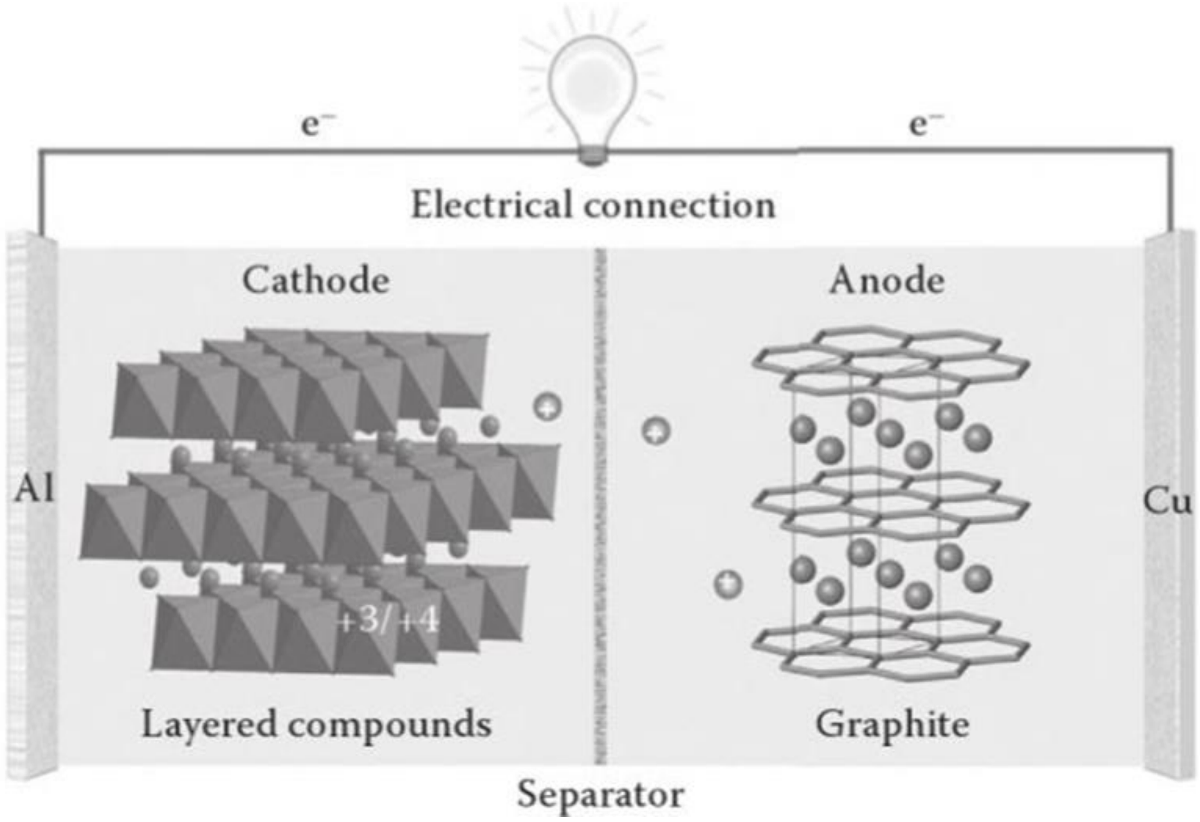


Figure 6: Battery layout [9]

It is this liquid electrolyte that creates a safety concern as any leakage can cause a thermal runaway. A lithium polymer battery was chosen as the electrolyte is generally made of a gel-like substance that is less susceptible to leakage. Other than the chemical composition, these 2 batteries are similar in operation. Lithium polymer rechargeable batteries usually possess a nominal voltage of 3.7V and charge up to a voltage of 4.2 with a typical constant current charge of 0.5C5A. Batteries are measured in C5A where A is the amps and C5 indicates the rate of 5 hours as the cell's capacity is not the same at all rates. That is why it is recommended to charge at half the rated charge current to avoid any unexpected changes in the rate of charge and preserve the battery. Milliamp hour (mAh) is a measurement used by batteries to indicate the maximum capacity of energy it can hold before dissipation is ceased. A capacity of 1200mAh was selected as larger capacities will entail a larger battery body and this capacity was sufficient to power a low-power system while maintaining a small profile. This means that the circuit can drain 1.2A for one hour, however, the circuit used will only drain a few milliamps (20-40mA) for several seconds to transmit data every hour.

D.Power Cycling Circuit

The power cycling circuit is composed of a TPL5110 low power timer and a DMP2045U P-channel MOSFET, both receiving power from the 3.7V battery. The TPL5110 has an integrated MOSFET driver, which is connected to the gate of the DMP2045U.

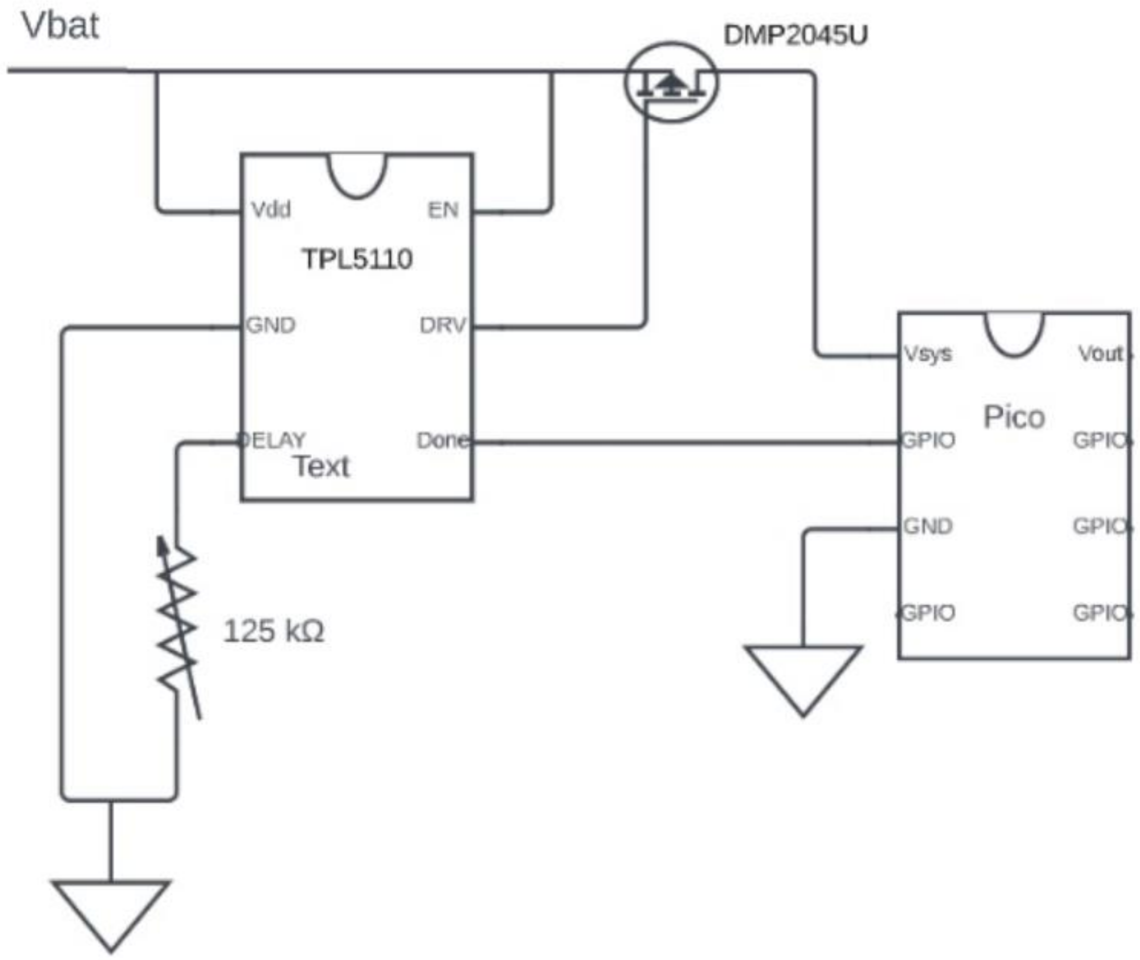


Figure 7: Power Cycling Circuit Diagram

This timer is intended for use in low power, energy-saving systems and has a current draw of only 35 nA, as well as a supply voltage range of 1.8V to 5.5V, the same range as the microprocessor, making it an ideal choice for this circuit.

The TPL5110 timer has a delay of 100 ms to two hours, which can be set by changing the value of the resistor connected from the delay pin to ground. The value of the resistor for the desired time can be determined by using the formula below.

$$R_{EXT} = 100 \bullet \frac{-b + \sqrt{b^2 - 4a(c - 100\,T)}}{2a}$$

Equation 7: Timer resistor calculations

This is the resistance needed to set the desired time (T) where a, b and c are coefficients depending on the range of the time interval. These coefficients are given in the datasheet, as well as several common time and equivalent resistor values. The calculated resistance for a one-hour delay is 124.9 kΩ.

Once the timer has delayed the set amount of time, it outputs a low signal to the gate of the MOSFET. Because this timer outputs a low signal, it is required to use a P channel MOSFET in conjunction with this timer, so that once the low signal is received from the timer, the current can flow from the battery through the MOSFET into the microprocessor. By using a MOSFET specifically selected for low leakage current, the power consumption when the microprocessor is not active is minimal. The microcontroller current draw was measured at the battery voltage of 3.7V. The DMP2045U-7 MOSFET measured a current draw of 0 mA at the battery voltage of 3.7V, and the relevant data sheet gives a current leakage of 10 uA at 8V.

Table 1: Pico Power consumption at 3.7V

Mode	Current (mA)
Standard	28.44
Sleep	2.2
Power Cycling	0.00

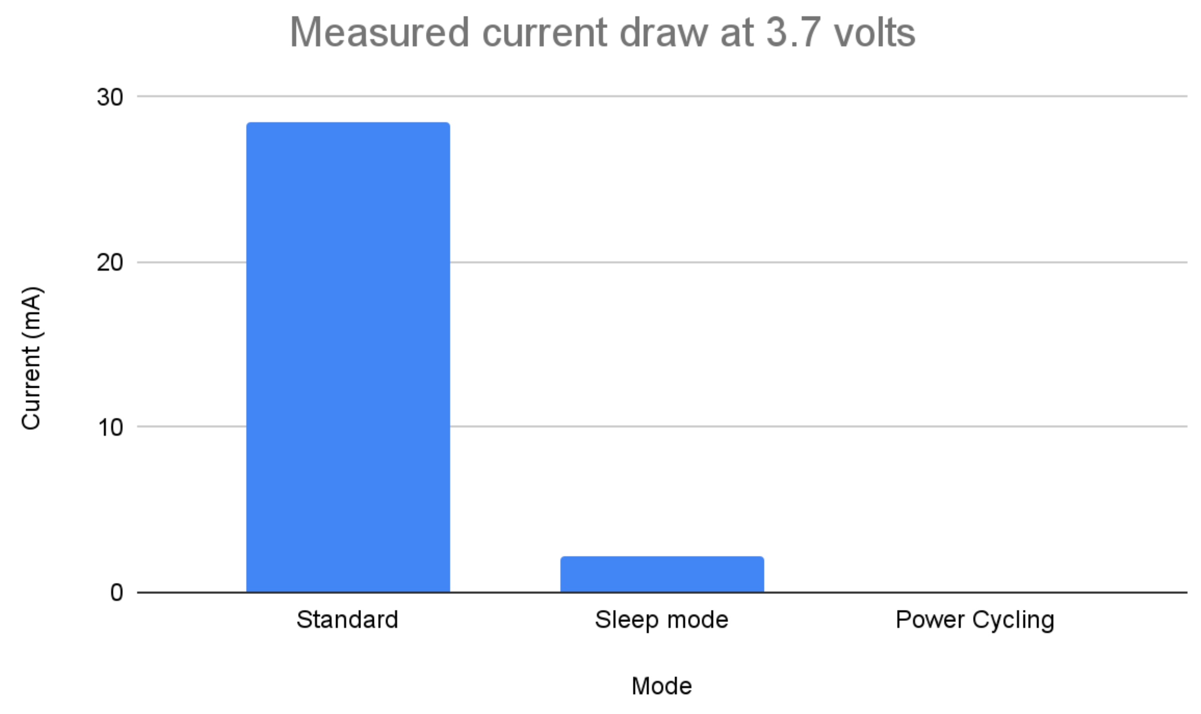


Figure 8: Power Consumption Graph

E. Microprocessor

The microprocessor selected for this project was the raspberry pi pico. This microprocessor excels in several of the most important criteria decided upon for the selection of a microprocessor, such as cost and power consumption. The incredibly low cost of $4 per pico is very relevant to the goal of creating a low-cost system.

The different configurations of GPIO pins on the pico are compatible with multiple types of data transfer, such as I2C, which allowed a wider selection of choices for the attached peripherals. The soil moisture and temperature sensor are connected to the pico via I2C and the LORA is connected via SPI.

An additional benefit of this number of GPIO pins is that several additional sensors or other components can be added if needed later without increasing the complexity of the circuit. It also means that a wide range of sensors can be added to the sensor node to obtain new kinds of information in future updates.

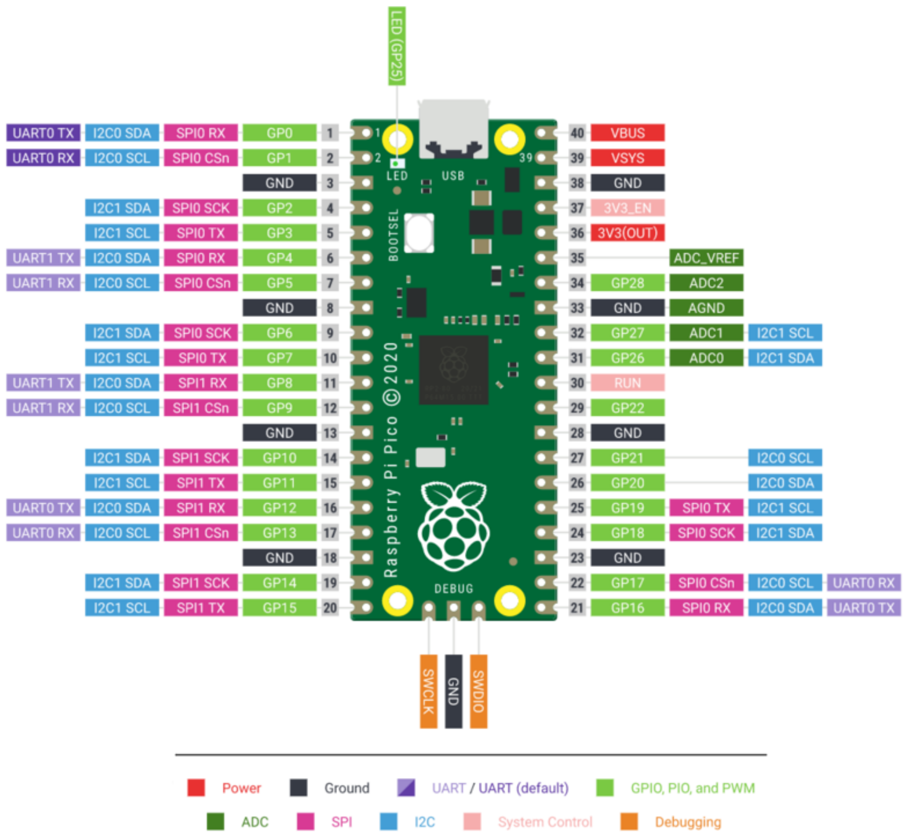


Figure 9: Raspberry Pico Pinouts [16]

The sensor and LORA receive power from the 3.3V output pin, are grounded to the ground pins, and transmit data through the I2C and SPI pins respectively.

The pico itself receives power from the battery connected to the power cycling circuit, connected to Vsys. This is the main system input voltage used to power the pico. An onboard voltage regulator accepts any voltage from 1.8 to 5.5V and outputs 3.3V, the system voltage for the pico. This means that there is no need for an additional voltage regulator between the pico and the battery.

Additionally, the raspberry pi pico is very compatible with the raspberry pi 4 selected as the base, as they are from the same manufacturer.

F. Sensors

The sensor nodes are a crucial part of this design as it enables the collection and transmission of measured data from the agricultural area to the central hub where the data can be further analyzed. The sensor used in this design is an Adafruit STEMMA Soil Sensor. This is a capacitive soil moisture sensor with an onboard temperature sensor. A capacitive soil moisture sensor was selected, as opposed to a resistive sensor, as these nodes are meant to be self-sufficient for extended periods. Capacitive moisture sensors erode much slower than resistive versions, as no exposed metal can be corroded by the soil over time. Resistive soil moisture sensors also don’t work as well in several non ideal conditions, such as loose or rocky soil.

The node antenna and receiver circuit used was the RFM95W Low Power Long Range Transceiver module. This transceiver module was chosen due to its low power consumption, robustness to interference and long range. The major downside is the cost of $20 per module, which is the largest single contributor to the cost of the nodes.

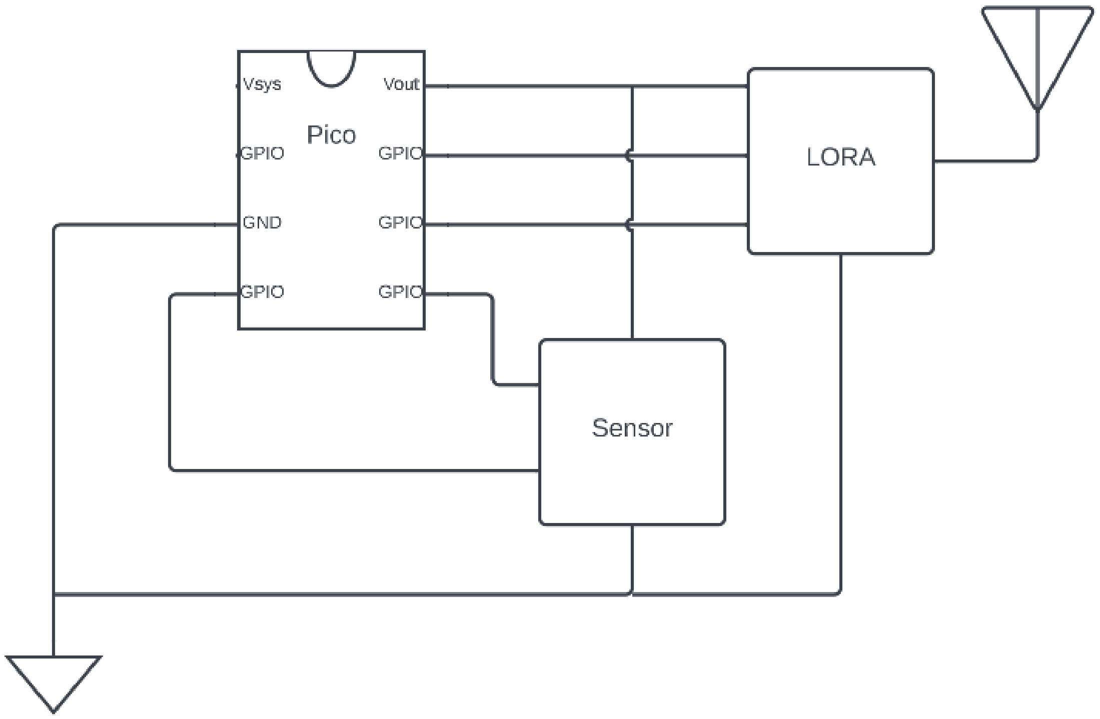


Figure 10: Node and Sensor Layout

The two components are connected to the node raspberry pico. The communication protocol used between the sensor and the microcontroller is an I2C protocol. It is a simple, flexible, and reliable serial communication protocol, whose main advantage is the ability to connect multiple devices to the same bus. The transceiver is connected to the microcontroller using SPI communication protocol. This is a synchronous serial communication protocol commonly used for short-distance communication, its key advantage is its high data transfer rates and full-duplex support.

The output of the sensor values is read from the STEMMA register. The soil sensor has a recommended library which makes it easier to collect the readings from the sensor. The format of the output is a numerical variable.

The RFM95 transceiver is a versatile device that can operate in multiple transmission modes, including LoRa, FSK, and GFSK. However, for this design, only the LoRa transmission mode was utilized. The power rating of the RFM95 during LoRa transmission is measured at 13 dBm which can be adjusted up to 23 dBm during high-power transmission mode. The supply voltage used was a 3.3 V power from the microcontroller. The transmission power impacts the range and reliability of the communication link. It was important to ensure that the power output is set to an appropriate level to optimize the performance of the RFM95 transceiver.

G. The Central Hub and Communication System

At the core of this design lies the central hub, which serves as the primary node responsible for coordinating and managing the various subsystems and peripheral devices. The components of the central hub are as follows, a microprocessor, an OLED display screen and an RFM95W transceiver module. Below is a System block diagram for the communication unit and central hub.

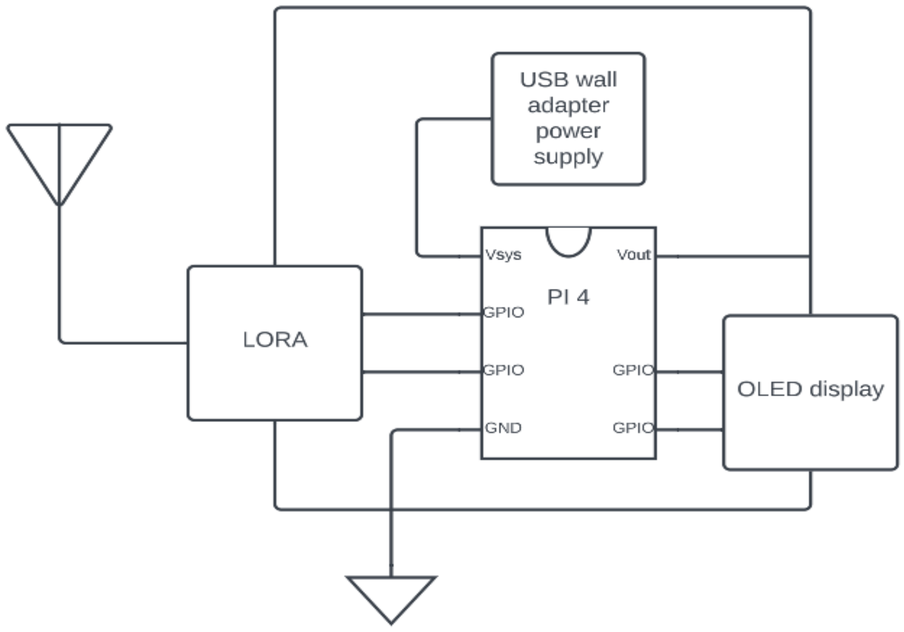


Figure 11: Central Hub Circuit Diagram [5]

The following are the requirements specifications for the communication system.

1. Range: The communication system must be capable of providing a wireless communication range of at least 1 km between the central hub and each sensor node.
2. Power Consumption: The power consumption of the communication system shall not exceed 50 mW to ensure efficient energy usage.
3. Collision Avoidance: The system must have a protocol in place to avoid data collision between the central hub and sensor nodes. Preferably, a scheduler shall be implemented to ensure that data transmissions do not overlap.
4. Channel Selection: Proper channel selection and time blocking shall be implemented to ensure optimal communication performance and minimize interference.
5. Message Acknowledgement: The system must provide message acknowledgement to ensure that data transmissions are successful and to identify and re-transmit lost data packets.

LoRa Transceiver (RFM95 Module)

LoRa, or Long Range, is a type of wireless communication protocol that operates at low frequencies to enable long-range, low-power communication for the Internet of Things (IoT) and other applications. Unlike other wireless communication protocols such as WiFi or Bluetooth, which operate at higher frequencies and have shorter ranges, LoRa uses spread-spectrum modulation and a unique chirp spread spectrum (CSS) modulation technique to enable communication over distances of several kilometres.

LoRa technology is typically used for low-bandwidth applications where longer range and lower power consumption are critical, such as environmental monitoring, asset tracking, and smart city infrastructure. The range of communication for LoRa can vary depending on the conditions, but in ideal conditions, it can be up to several kilometres.

In terms of signal quality, LoRa typically has a signal-to-noise ratio (SNR) of around 10 dB, with a bit error rate (BER) of less than 1%.

The packet structure for LoRa communication involves a preamble, sync word, address byte, payload and cyclic redundancy check (CRC), with the length of each field determined by the specific LoRa protocol being used. There are three types of packet structures used.

Fixed Length Packet Format:

To select a fixed length packet format, set the bit PacketFormat to 0. If the value of PayloadLength is greater than 0, the packet format will be set. The payload length is limited to 2047 bytes and it contains the following fields:

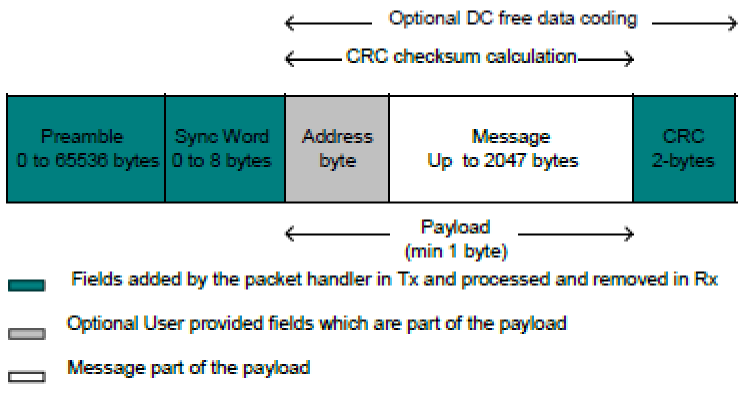

Optional DC free data coding
CRC checksum calculation
Preamble 0 to 65536 bytes
Sync Word 0 to 8 bytes
Address byte
Message Up to 2047 bytes
CRC 2-bytes
Payload (min 1 byte)
Fields added by the packet handler in Tx and processed and removed in Rx
Optional User provided fields which are part of the payload
Message part of the payload

Figure 12: Fixed length packet format [13]

Variable Length Packet Format:

The variable length packet format is chosen if the bit PacketFormat is set to 1. This format is useful for applications where the packet length is unknown beforehand and can vary over time. In such cases, the transmitter must send the length of information along with each packet to ensure the proper operation of the receiver. It contains the following fields:

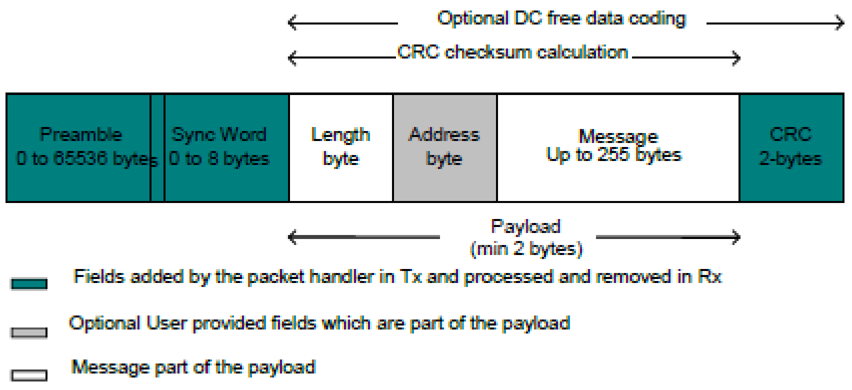

Optional DC free data coding
CRC checksum calculation
Preamble 0 to 65536 bytes
Sync Word 0 to 8 bytes
Length byte
Address byte
Message Up to 255 bytes
CRC 2-bytes
Payload (min 2 bytes)
Fields added by the packet handler in Tx and processed and removed in Rx
Optional User provided fields which are part of the payload
Message part of the payload

Figure 13: Variable length packet format [13]

Unlimited Length Packet Format:

The unlimited length packet format is chosen by setting the bit PacketFormat to 0 and the PayloadLength to 0. With this format, the user can transmit and receive packets of any length, and the PayloadLength register is not utilized for counting the length of the bytes transmitted or received in Tx/Rx modes. It contains the following fields:

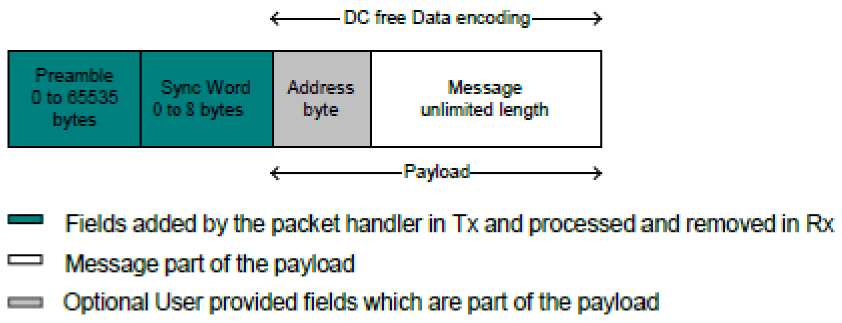

DC free Data encoding
Preamble 0 to 65535 bytes
Sync Word 0 to 8 bytes
Address byte
Message unlimited length
Payload
Fields added by the packet handler in Tx and processed and removed in Rx
Message part of the payload
Optional User provided fields which are part of the payload

Figure 14: Unlimited length packet format [13]

The modulation and demodulation of LoRa signals is a complex process that involves several stages. At first, the modulator takes the digital data and transforms it into a waveform. This waveform is then spread across a wider frequency band using a spreading factor (SF). By increasing the SF, the modulator can create a more spread-out signal that is less susceptible to interference and thus has an improved range.

On the other end of the communication channel, the demodulator uses a combination of matched filtering and correlation techniques to recover the original data from the received signal. This process is crucial in extracting the original data from the modulated signal, which has been spread out over a wider frequency band. By using the matched filtering technique, the demodulator can identify the frequency content of the signal, which is then used to recover the original data.

Overall, the modulation and demodulation of LoRa signals is a critical process in ensuring the reliable transmission of data over long distances with minimal interference. The use of spreading factors and correlation techniques in LoRa technology helps to improve the robustness and range of the communication system. LoRa technology provides a powerful and flexible communication solution for low-power, long-range applications, enabling new possibilities for IoT and other industries.

Communication Protocol

The protocol is based on a request-response mechanism where the sending device requests a channel from the receiving device to transmit data, and the receiving device responds with an acknowledgment (ACK). Once the sending device has received the ACK, it sends the actual data. If the ACK is not received within a specified time, the sending device retries sending the request or data.

Synchronization of the request-response was done using a transmission window. A transmission window refers to a specific time interval within which a wireless communication device or system is allowed to transmit data. During the transmission window, the sensor node would be able to transmit data to the central hub, and the central hub would be able to receive the data.

Once the transmission window closes, the sensor node would stop transmitting data until the next scheduled transmission window opens. This approach can help to reduce the likelihood of data collisions and other issues that can arise from simultaneous transmissions by multiple wireless devices.

In the transmission window approach, each device is assigned a specific time slot within which it can transmit its data. These time slots are carefully scheduled to ensure that no two devices transmit their data at the same time, which helps avoid packet collisions. For example, if there are four devices, each device might be assigned a time slot of 10 milliseconds, starting from 0 milliseconds for device one, 10 milliseconds for device two, 20 milliseconds for device three, and 30 milliseconds for device four. During its assigned time slot, a device can transmit its data without interference from other devices, which helps avoid collisions.

In this communication design, a transmission window of 60 minutes was established to coincide with the time that soil sensors collect environmental data. By implementing time-scheduled requests, data transmissions are organized to avoid packet collisions that can cause network congestion and reduce network performance. This allows each sensor node to transmit data to the central hub without interference from other nodes, thereby ensuring data integrity.

To optimize data transmission efficiency, the data packet format used a combination of variable and unlimited formats. When sensor data and acknowledgment messages are sent to the central hub, a variable format is used to accommodate the varying data size of each sensor node. However, in situations where the fault log data exceeds 2047 bytes, an unlimited data format is utilized to ensure that all data is transmitted. This approach guarantees that data is not lost due to truncation and that all data is transmitted reliably to the central hub. By implementing this communication design, the soil sensor network can operate effectively and provide accurate environmental data for agricultural applications. Additionally, CRC is enabled on all packet modes except the unlimited mode.

Software Configuration and Setup

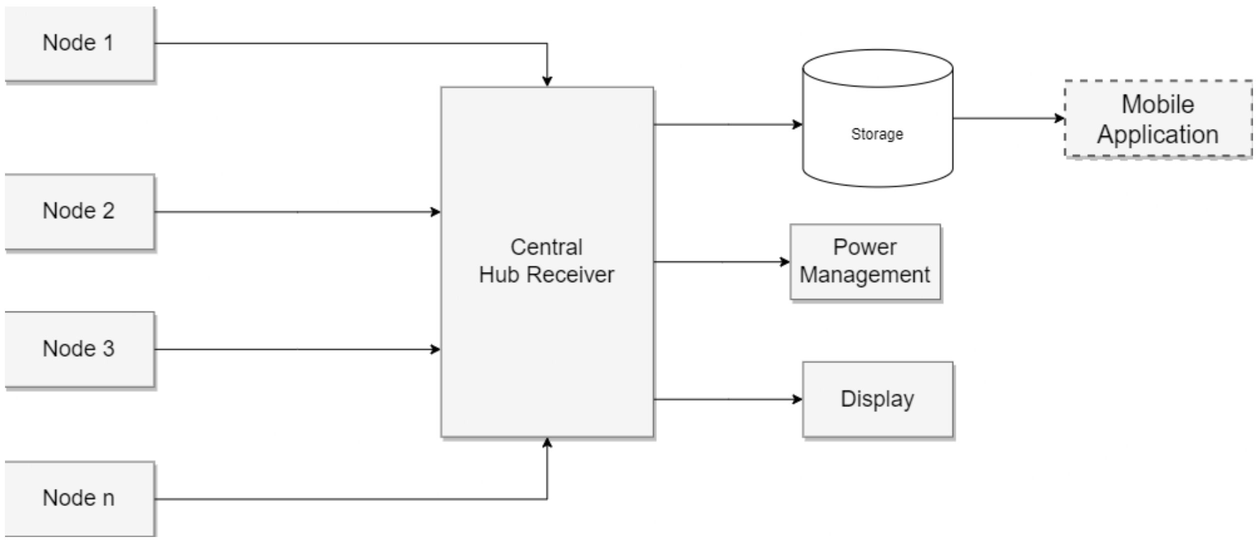


Figure 15: Software Overview Diagram

This Software functionality of the design was implemented as follows

- Node Side
  - Transceiver
  - Collect sensor data
- Central Hub
  - Transceiver
  - Display
  - Server API
  - Storage
- Android Application
  - Recieve API data
  - Display UI

This was implemented using the flowchart below.

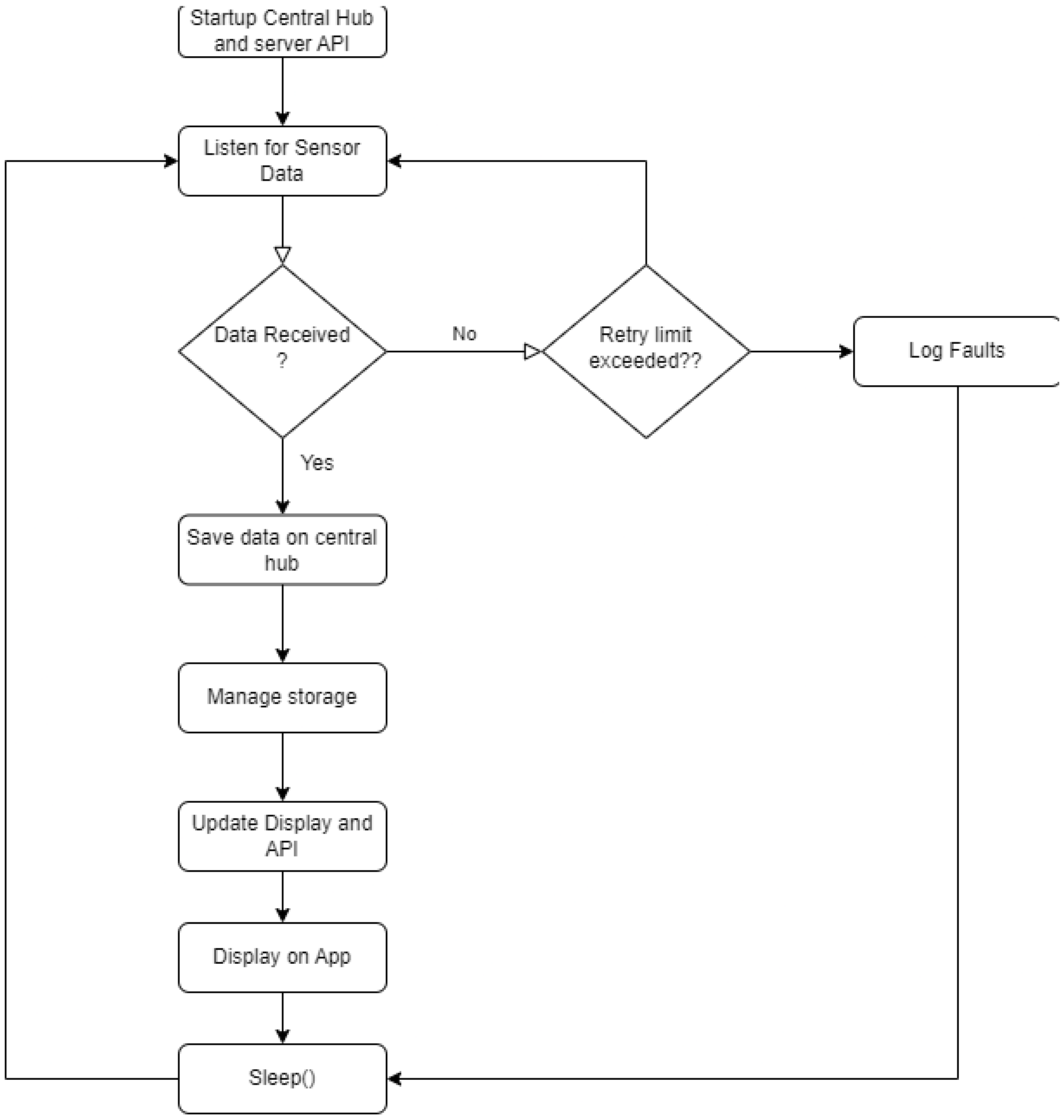


Figure 16: Flowchart for software implementation.

See the Appendix for full implementation.

H. Display and Storage

Storage

Sensor data is stored in a CSV file on the central hub, with new data added to the end of the file. After some time, data that is over 6 months old are deleted to keep the files clean. Additionally, there is a fault log that keeps track of any instances of data or channel failures for troubleshooting purposes. The CSV files can be accessed by the Android app through a Flask API, and the most recent data is sent to an OLED display.

Display

With the use of an I2C protocol, the OLED display can easily communicate with the central hub and display the data received from the sensor node. The attached picture shows a sample display of sensor information.

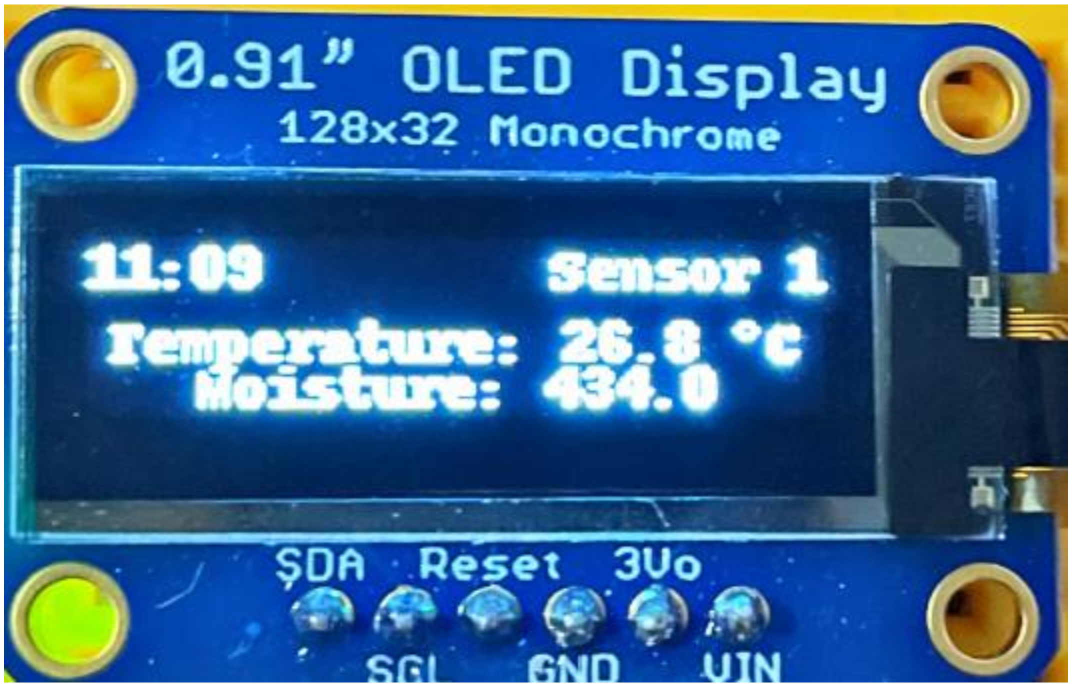


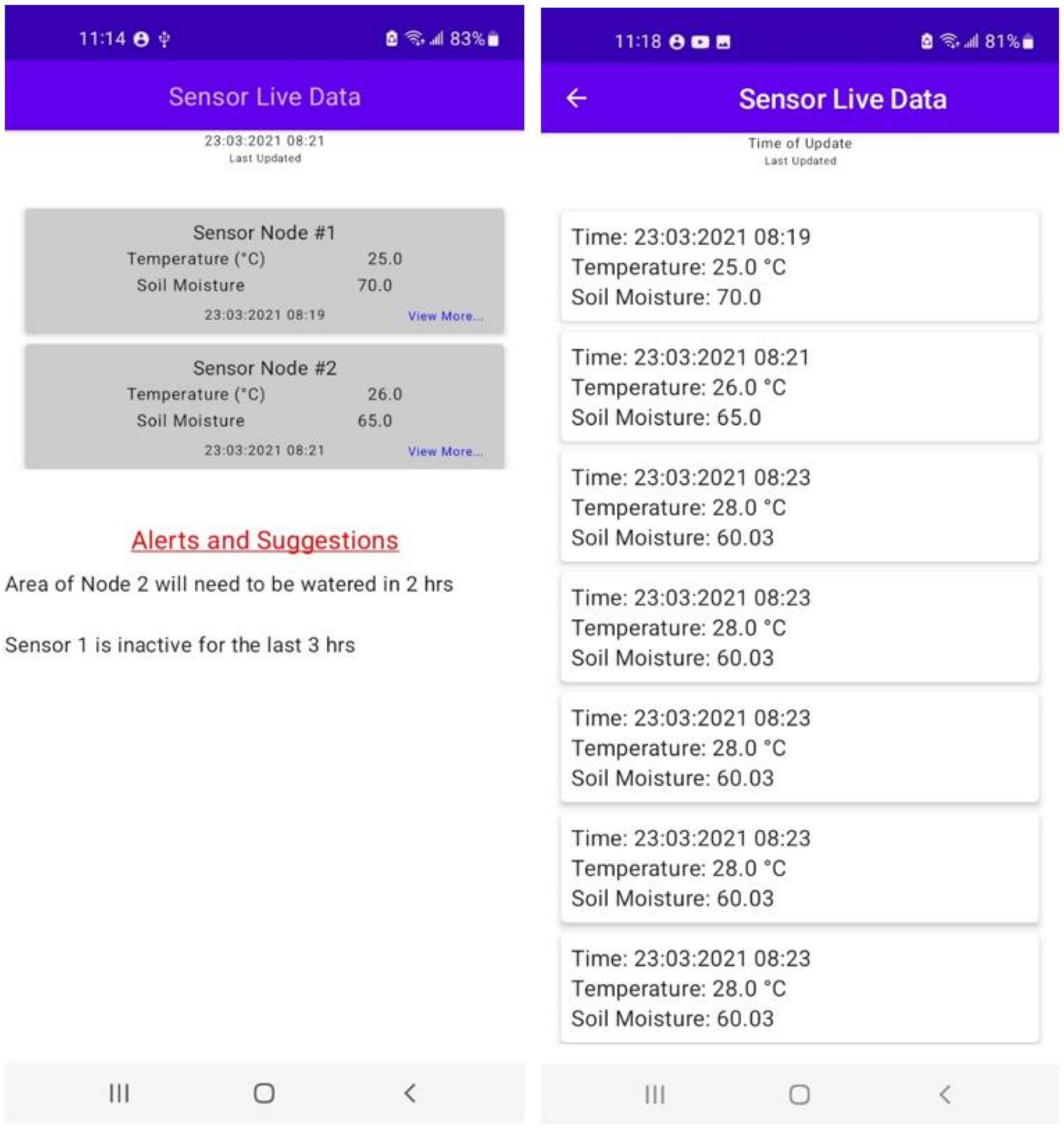
Figure 17: Display Output

Android Application

The Android app implementation is designed to retrieve data from a Flask API that is hosted on the central hub. The app is designed to display two screens. The first screen is a dashboard display that provides an overview of all sensor node's most recent data. The second screen is accessed by clicking on a sensor node's data. This opens the second page that displays 24-hour data from each sensor node.

The app uses RESTful APIs to connect with the Flask API, retrieving sensor data from the Raspberry Pi. The data is displayed in a visually appealing and easy-to-understand dashboard format on the first screen. The second screen displays detailed information on a specific sensor node, providing a graphical representation of the data over the past 24 hours. The app is user-friendly and easy to navigate, providing agricultural users with accurate environmental data that they can use to optimize crop production.

See screenshots of the Android app implementation below



11:14 83%

Sensor Live Data

23:03:2021 08:21
Last Updated

Sensor Node #1

Temperature (°C)	25.0
Soil Moisture	70.0

23:03:2021 08:19 View More...

Sensor Node #2

Temperature (°C)	26.0
Soil Moisture	65.0

23:03:2021 08:21 View More...

Alerts and Suggestions

Area of Node 2 will need to be watered in 2 hrs

Sensor 1 is inactive for the last 3 hrs

11:18 81%

Sensor Live Data

Time of Update
Last Updated

Time: 23:03:2021 08:19
Temperature: 25.0 °C
Soil Moisture: 70.0

Time: 23:03:2021 08:21
Temperature: 26.0 °C
Soil Moisture: 65.0

Time: 23:03:2021 08:23
Temperature: 28.0 °C
Soil Moisture: 60.03

Time: 23:03:2021 08:23
Temperature: 28.0 °C
Soil Moisture: 60.03

Time: 23:03:2021 08:23
Temperature: 28.0 °C
Soil Moisture: 60.03

Time: 23:03:2021 08:23
Temperature: 28.0 °C
Soil Moisture: 60.03

Time: 23:03:2021 08:23
Temperature: 28.0 °C
Soil Moisture: 60.03

Figure 18: Android Application Preview

6. Results and validation

A test was conducted to observe the charging behaviour of the battery and whether the CN3083 chip would cut off charging as intended. A constant voltage supply of 5 volts DC was provided to the chip and the results were tabulated into a graph.

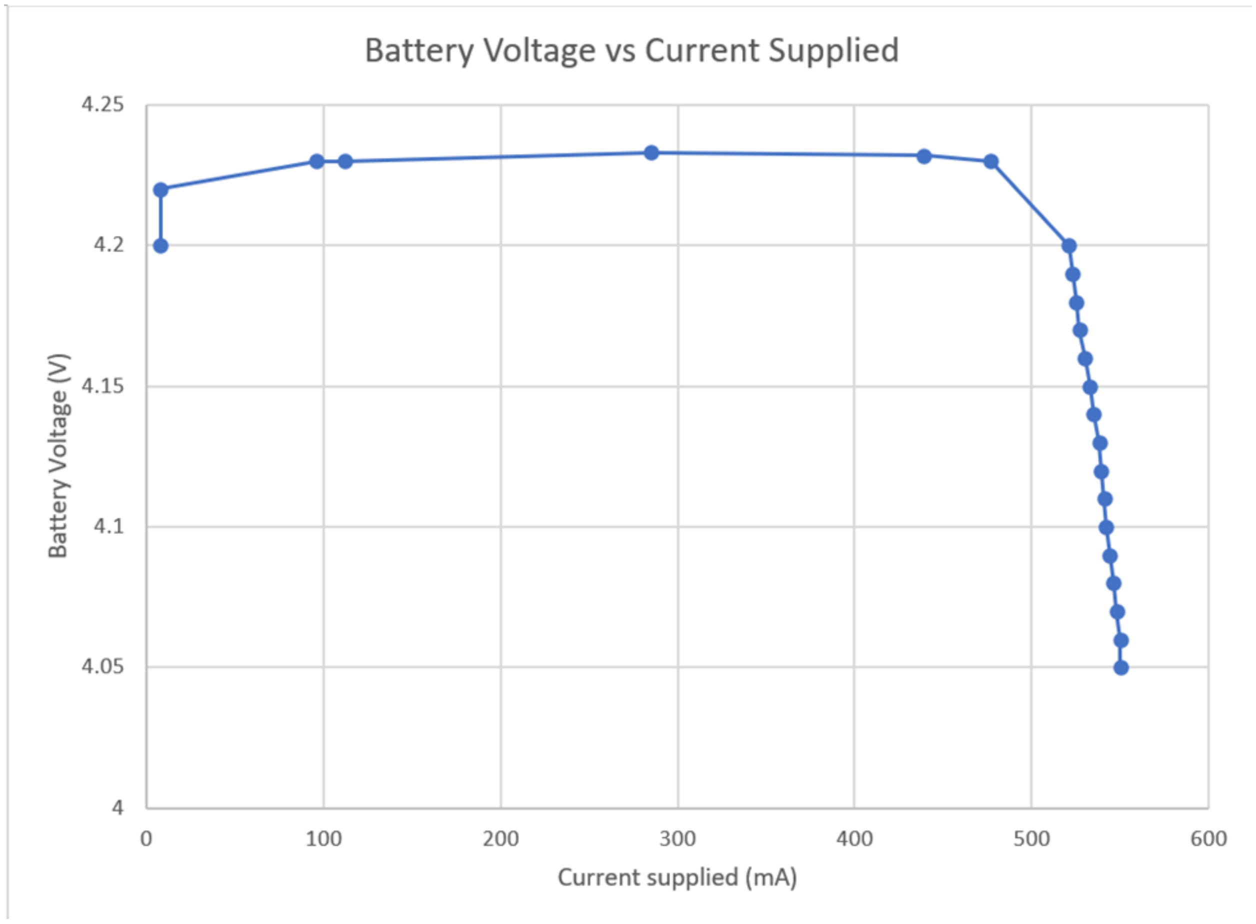
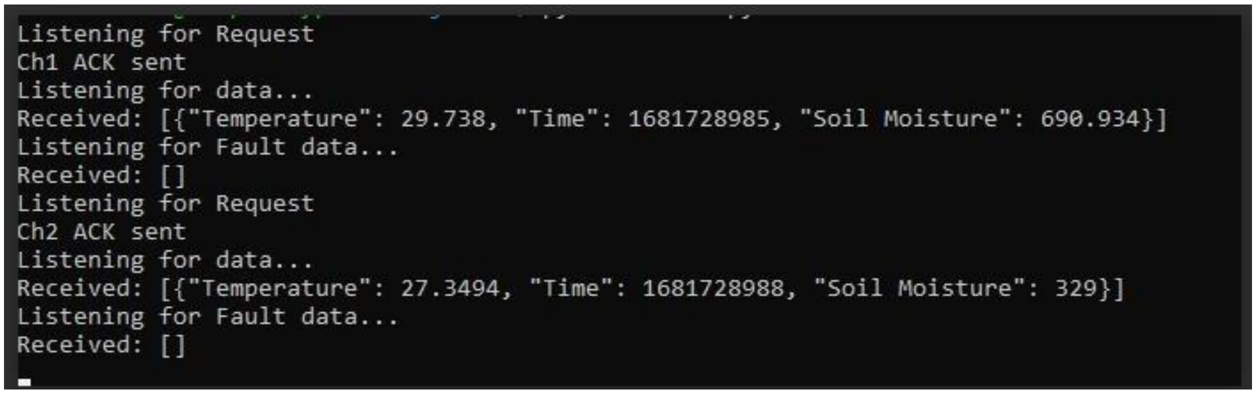


Figure 19: Battery Voltage vs Supplied Current Graph

The graph in Figure X displays the battery voltage in volts and current supplied in milliamps by the CN3083 chip. The chip maintains a steady supply current of approximately 550 to 500 milliamps. As the voltage of the battery increases, the current supplied begins to gradually decrease until the voltage reaches the maximum point of slightly over 4.2V, after which the current decreases drastically until it stabilizes at 0.008amps. That value of current does not further charge the battery as the voltage begins dropping regardless as shown at the 0-amp point. It is also at this point that the red LED turns off and the green LED is enabled by the CN3083 chip which confirms the correct operation of the chip.

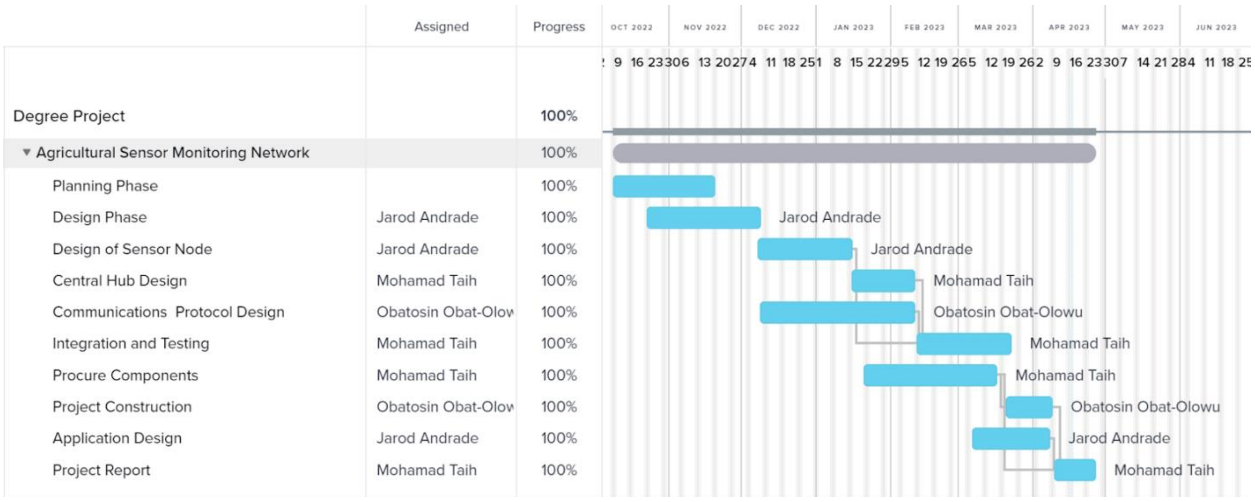
The communication range was tested up to a distance of approximately one kilometre, with a focus on testing signal range and detection time. Within the test range, the signal was received with delays of less than one second. The image below depicts a successful data transmission from the sensor node to the central hub.



```
Listening for Request
Ch1 ACK sent
Listening for data...
Received: [{"Temperature": 29.738, "Time": 1681728985, "Soil Moisture": 690.934}]
Listening for Fault data...
Received: []
Listening for Request
Ch2 ACK sent
Listening for data...
Received: [{"Temperature": 27.3494, "Time": 1681728988, "Soil Moisture": 329}]
Listening for Fault data...
Received: []
```

Figure 20: Successful communication from Node to Hub

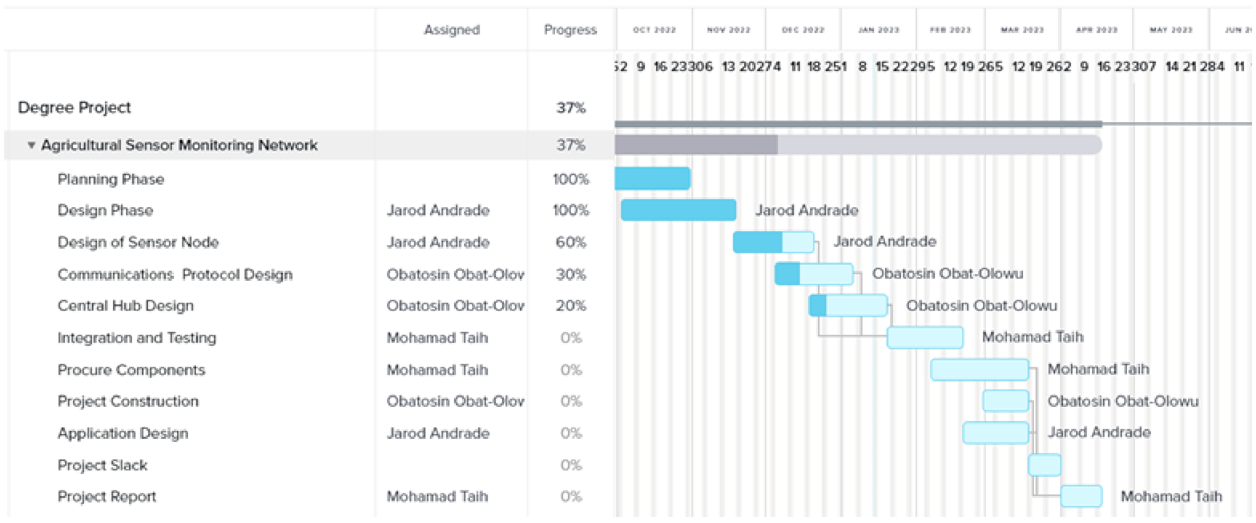
7. Progress and Revisions



	Assigned	Progress
Degree Project		100%
▾ Agricultural Sensor Monitoring Network		100%
Planning Phase		100%
Design Phase	Jarod Andrade	100%
Design of Sensor Node	Jarod Andrade	100%
Central Hub Design	Mohamad Taih	100%
Communications Protocol Design	Obatosin Obat-Olow	100%
Integration and Testing	Mohamad Taih	100%
Procure Components	Mohamad Taih	100%
Project Construction	Obatosin Obat-Olow	100%
Application Design	Jarod Andrade	100%
Project Report	Mohamad Taih	100%

Figure 21: Revised Gantt Chart

The Gantt chart for this project was changed to reflect the new workflow that was devised after working on the project for some time. The workflow for designing the sensor node, central hub and communication protocol was altered to allow for concurrent work on the communication protocol. The timeline for procuring components was moved up, as many of the needed components were decided on earlier than expected. Once construction of the project began, additional purchases had to be made for miscellaneous components, such as adapters. The timeline for integrating the components was decreased, as it went much better than expected.



	Assigned	Progress
Degree Project		37%
▾ Agricultural Sensor Monitoring Network		37%
Planning Phase		100%
Design Phase	Jarod Andrade	100%
Design of Sensor Node	Jarod Andrade	60%
Communications Protocol Design	Obatosin Obat-Olov	30%
Central Hub Design	Obatosin Obat-Olov	20%
Integration and Testing	Mohamad Taih	0%
Procure Components	Mohamad Taih	0%
Project Construction	Obatosin Obat-Olov	0%
Application Design	Jarod Andrade	0%
Project Slack		0%
Project Report	Mohamad Taih	0%

Figure 22: Unrevised Original Gantt Chart

The overall design of the sensor node underwent one significant revision, which involved the addition of a voltage regulator for the battery and a new control circuit to cycle power to the Arduino.

An issue encountered with the power cycling circuit was that when connected to the battery at full charge, the power cycling circuit would not always receive the done signal and turn off properly. A stopgap solution was to send the done signal multiple times until the power to the pico was cut off. After lab analysis, it was theorized that this issue was due to the voltage difference between the 4.2V fully charged battery and the 3.3V output of the pico microcontroller. This issue is planned to be resolved by decreasing the voltage coming to the timer from the battery, through the use of a diode.

.

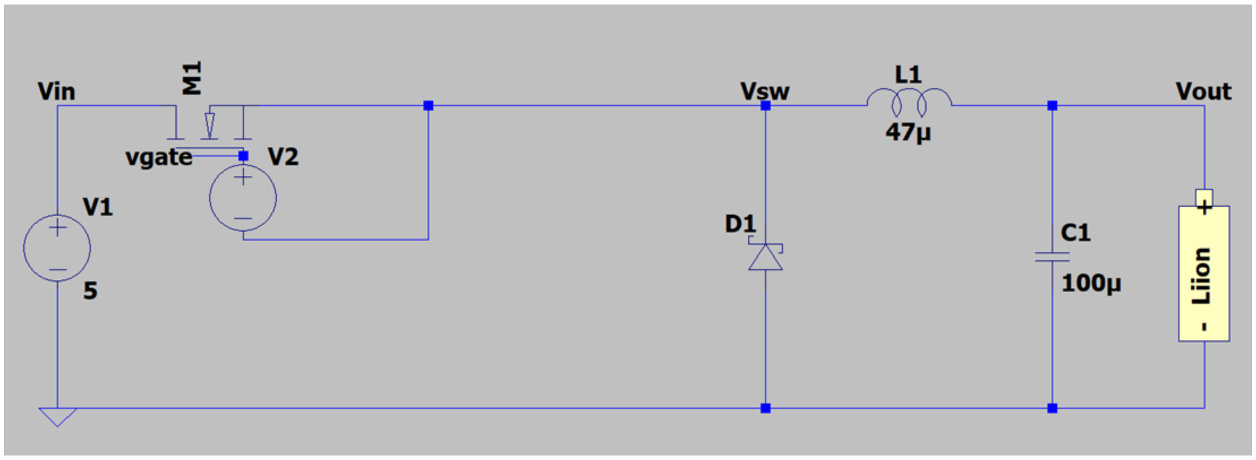


Figure 23: Linear voltage regulator diagram

Figure **7** shows a basic buck converter circuit to step down an input of 5 volts to an acceptable battery charging range of 3.7 to 4.2 volts. Component M1 is a MOSFET that will behave as a switching device when a pulse is supplied to the gate terminal. When the switch is closed, the inductor L1 will charge and store energy as current flows through the inductor and to the battery load and capacitor C1 after which it will return to the negative terminal of the DC supply V1.

When the switch is open, the inductor will behave as the power supply as it transfers the energy it has stored to the load and capacitor. The power will now loop back through the diode (D1) until the switch returns to a closed state. The purpose of the capacitor is to filter any AC component at the output to maintain a steady DC supply to the load. This constant switching will ensure that only a certain amount of voltage is supplied to the load. The voltage will rise to a certain point, once that point is exceeded, the switch will open, and the inductor will keep supplying the voltage to the load until the switch is closed. One problem with this circuit is the fact that it is a linear regulator, meaning it does not compensate for a change in input voltage. This is not ideal as the solar panel is expected to supply a varying range of voltages depending on sunlight exposure, providing too much voltage could damage the battery and not enough voltage will not initiate charging. Another issue is supplying a pulse supply to the gate as there will only be one source of DC power at the node. This means that another circuit is required to convert DC to a pulse through the gate to drive the MOSFET.

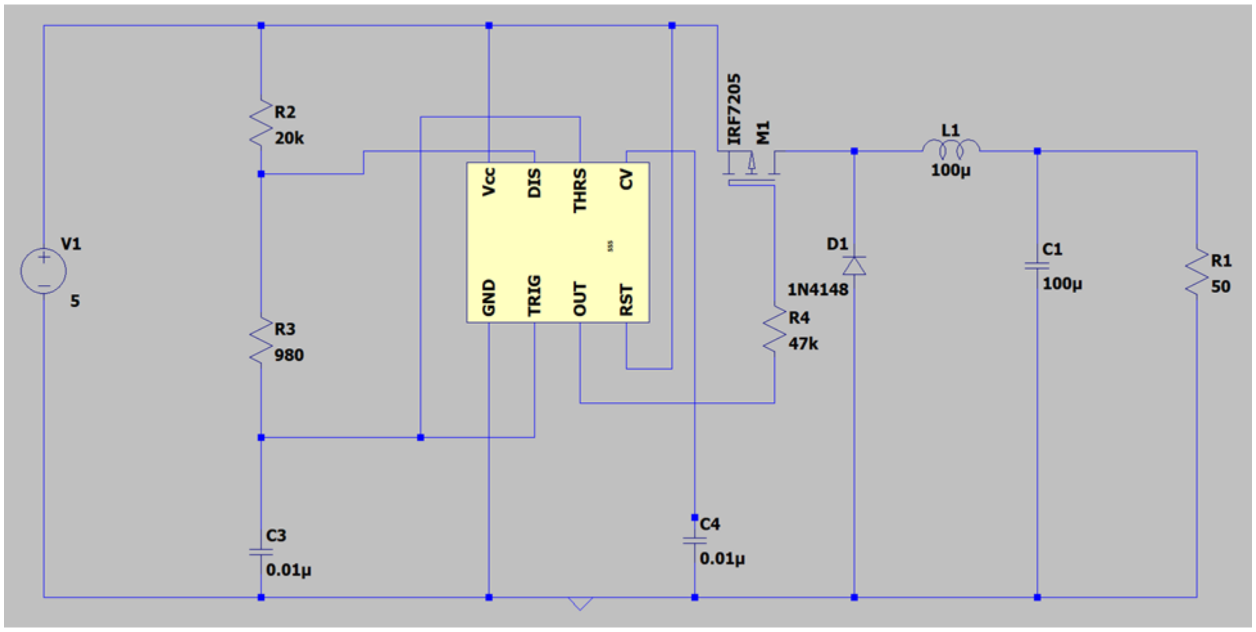


Figure 24: Linear regulator with pulsing timer

An attempt to simulate a pulse using a 555 timer was successful as the timer cuts off and on supply to the gate, however, the circuit requires many more components which will translate to a larger overall size and an involved troubleshooting process if one component fails. The switching will also happen at a slower rate as the speed is limited by the timers' capabilities. This circuit is also still a linear regulator and is missing safety features such as voltage cut-off when charging is complete. To provide compensation for input voltage change, a comparator amplifier should be used to compare the input with the output, however, this will contribute to a larger circuit and higher cost. Ultimately, a battery charging IC was decided upon due to the reduction of components needed, which will lead to a reduction in size and greater portability, as well as a cost reduction and added simplicity which makes troubleshooting potential issues easier.

8. Economic and profitability analysis

The price of the individual central hub and sensor nodes are broken down below. The price for the PCB and protective case used in the sensor node are conservative estimates.

Table 22: Cost per Node

Component	Quantity	Unit Price	Price
Raspberry Pi Pico	x1	$4	$4
STEMMA Soil Sensor	x1	$11	$11
TPL5110 Timer	x1	$1.95	$1.95
DMP2045U P-channel MOSFET	x1	$0.69	$0.69
RFM95W LORA Radio Transceiver	x1	$29.32	$29.32
Spring Antenna	x1	$0.95	$0.95
CN3083 battery charger chip	x1	$1.33	$1.33
2.5W 5V/500mAh Solar Panel	x1	$9	$9
3.7V 250mAh Li-Polymer Battery	x1	$9.52	$9.52
SS14 Diode	x1	$0.58	$0.58
LEDS	x2	$0.50	$1
Resistors	x3	$0.10	$0.30
Capacitors	x2	$0.15	$0.30
PCB	x1	$2	$2
Protective Case	x1	$3	$3
Total	**x19**	**$74.94**	

A sensor node that performs similar functions, collecting data about soil moisture and temperature, is available on Digi key, although the T0005986 KIWI agricultural sensor also detects air humidity. However, this agricultural sensor retails for $244 per node. This is more than three times the price of our current sensor node design. This demonstrates the success of making our design low-cost, as even after allowing margins for manufacturing and profit, our sensor node is still significantly cheaper. This should significantly aid the goal of helping those most impacted by food insecurity. Additionally, given the price of the KIWI system, an air humidity sensor could be added to the sensor nodes while still keeping them significantly cheaper than the competitor's version.

Table 33: Cost per Central Hub

Cost per Central Hub		
Component	**Quantity**	**Price**
Raspberry Pi 4B	x1	$48
RFM95W LORA Radio Transceiver	x1	$29.32
Spring Antenna	x1	$0.95
Monochrome 0.91" 128x32 I2C OLED Display	x1	$12.50
Raspberry Pi 4 Power Supply Cable	x1	$8
Total	**x5**	**$98.77**

The price of the central hub is larger than that of the nodes, mainly due to the need for a microcontroller, which is significantly more expensive than the microprocessor used in the nodes. The larger price is offset by the fact that only one central hub is required.

9. Future Plans

More sensors, such as a soil nitrogen sensor or an air humidity sensor, can be added with reasonable ease given the nature of the pico microcontroller, the only concern is the price of the sensors themselves.

The communication protocol design is continuously evolving to meet the growing demands of our users. In the future, various new features will be introduced to enhance the functionality and security of the protocol. These features include:

- Asynchronous Communication: To provide users with the ability to communicate with their devices at any time, the protocol will incorporate asynchronous communication that enables data transmission beyond predefined timeframes, recognizing the importance of flexibility and expediency in communication.
- Security Key: A security key feature will be implemented to prevent hijacking by unauthorized users, ensuring that only authorized personnel have access to the protocol. This feature acknowledges that security is of paramount concern to users.

- Android App Sensor Set Point Adjustment: An Android app will be introduced to allow users to remotely adjust their sensor set points, providing convenience and accessibility while making operations more efficient.

Due to time constraints, a prototype board was utilized in the final design. Prototype boards possess a smaller thickness than conventional breadboards. The thickness of prototype boards is usually 1mm while breadboards are around 8 to 9mm. This will allow for a smaller overall node to maintain a portable frame. A printed circuit board (PCB) has been designed to further save space and easily mass-produce nodes without needing to construct prototype boards for each.

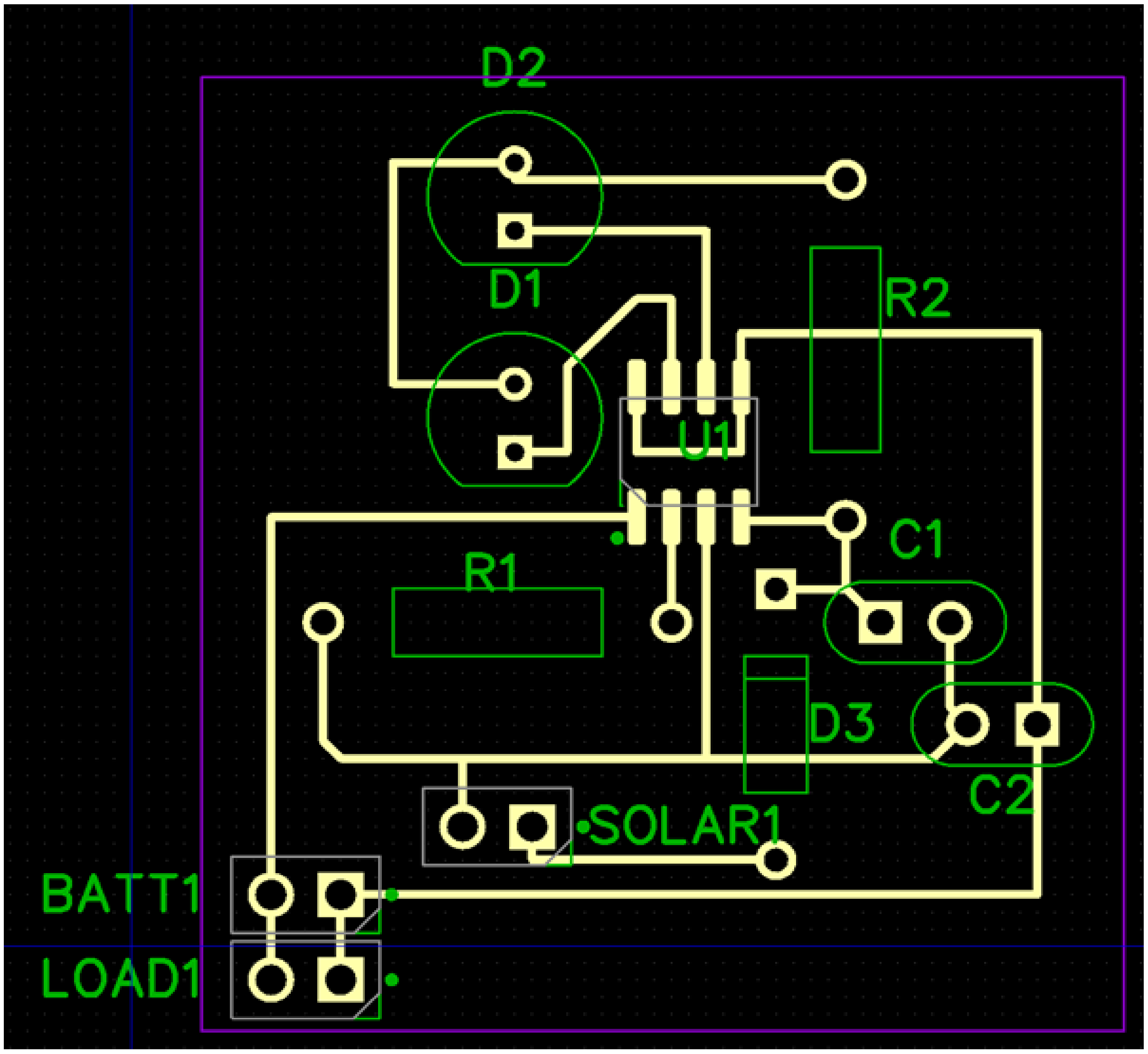


Figure 25: PCB layout

Furthermore, surface mount devices (SMD) can be used instead of through-hole components to further reduce the body of the PCB.

A rudimentary case was designed to protect the node components from harm in the field. The case will be 159x149mm due to the size of the solar panel selected. The case will have an open top where the solar panel will rest and act as a cover for the internals. To further protect the solar panel, the case must be mounted higher up to avoid accidental damage. It will be attached to a steel tube that is a few inches in length. This tube will protect the sensor wires that will be routed through from the bottom of the case to the tube attached and finally to the soil. Since heat rises, opening slits should be placed near the top of the case to prevent overheating, however, this will expose the internals to outside elements such as rain. Also, water may seep through the solar panel to the internals if it is not properly attached, so further revisions must be made to protect the internals from water damage. Also, the case can be made smaller if a smaller solar panel is used as well as PCB and SMD integration for the internals.

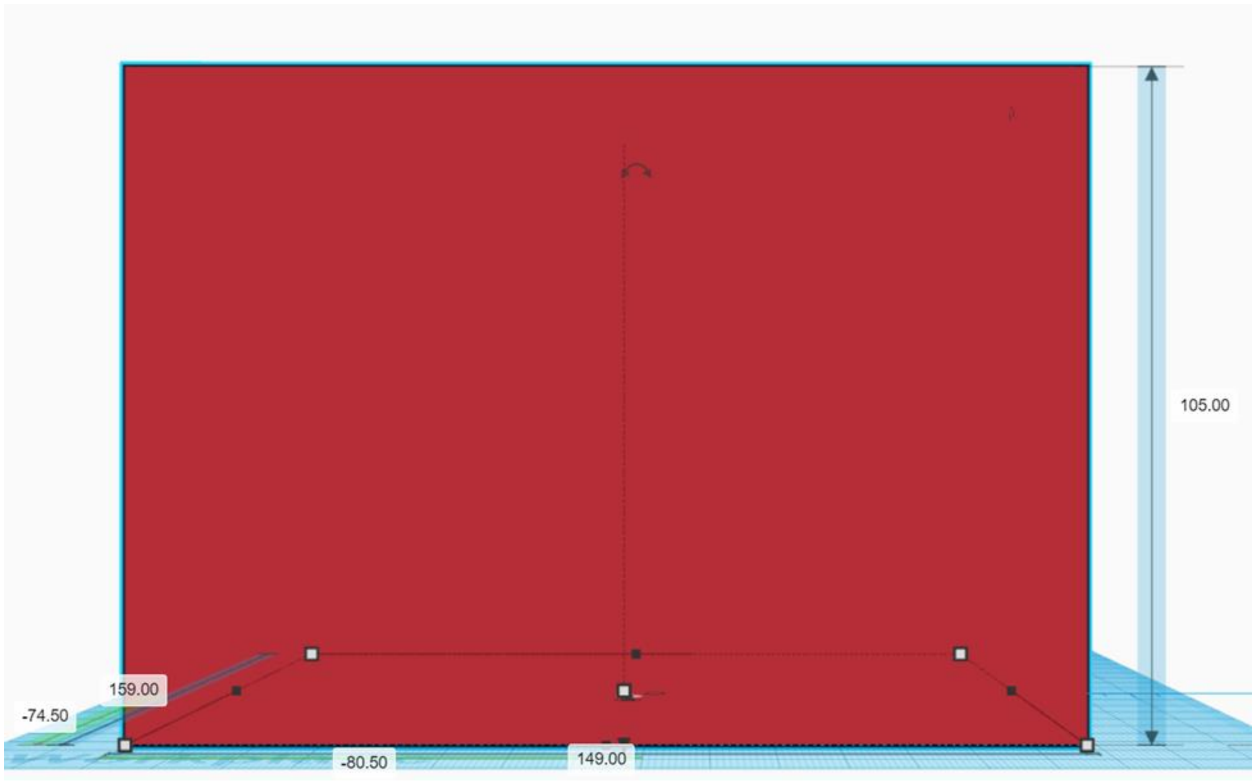


Figure 26: Protective Case Front view

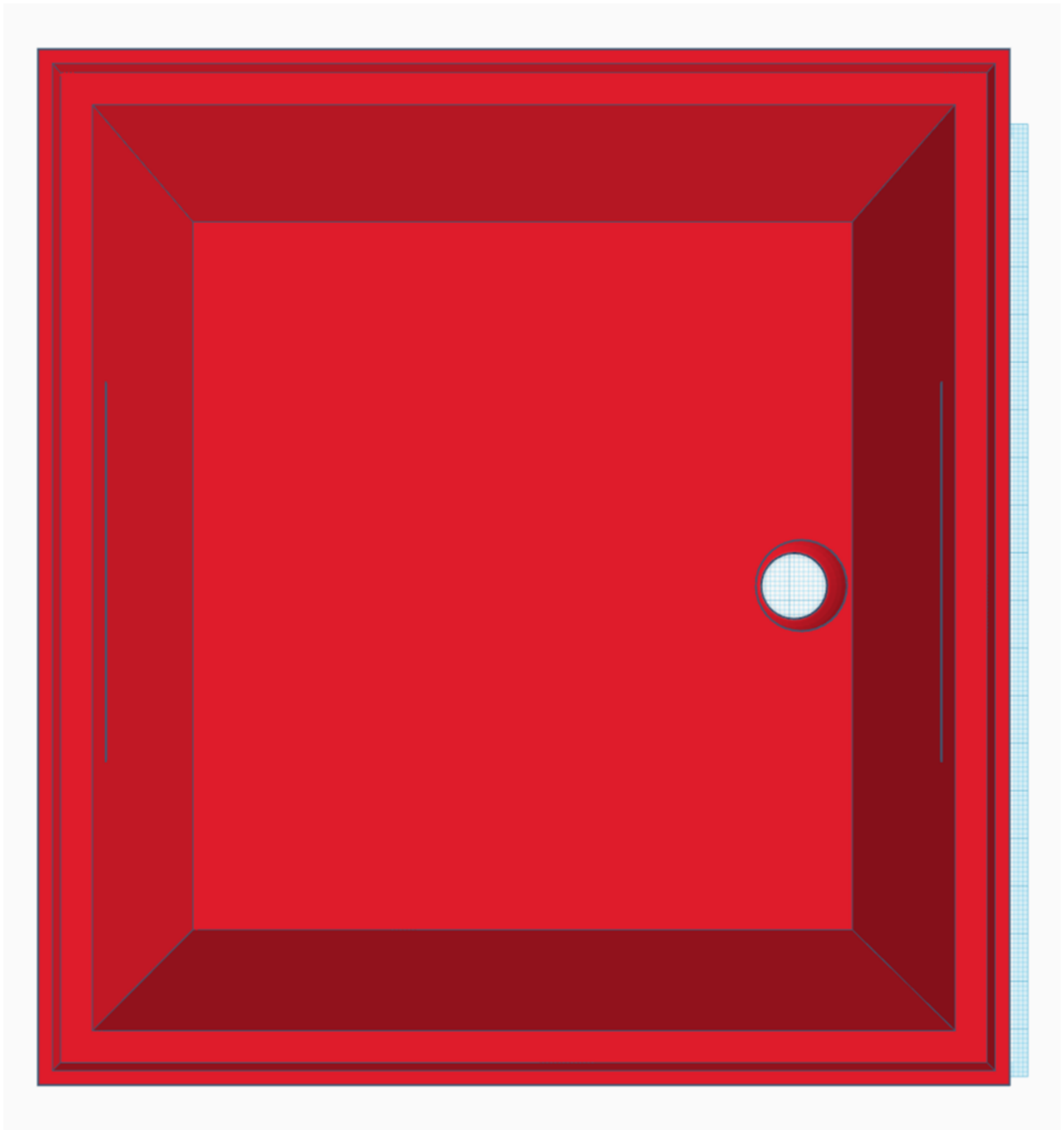


Figure 27: Protective Case Top view

10. References

[1] “Global Food Crisis: World Food Programme,” UN World Food Programme. [Online]. Available:https://www.wfp.org/emergencies/global-food-crisis#:~:text=345%20million%20people%20are%20facing,hunger%20crisis%20of%20unprecedented%20proportions.

[2] B. B. · T. C. Press ·, “High food prices amid easing inflation has consumers questioning pricing power at big grocery chains | CBC News,” CBC, Apr. 12, 2023. https://www.cbc.ca/news/business/competition-grocery-1.6807721

[3] S. Payraudeau and H. M. G. van der Werf, “Environmental impact assessment for a farming region: a review of methods,” Agriculture, Ecosystems & Environment, vol. 107, no. 1, pp. 1–19, May 2005, doi: 10.1016/j.agee.2004.12.012.

[4] “Basic Characteristics and Characterization of Solar Cells,” in Materials Concepts for Solar Cells, 2nd ed.WORLD SCIENTIFIC (EUROPE), 2018, pp. 3–43.

[5] “A Photon’s Million-Year Journey from the Center of the Sun,” Futurism. https://futurism.com/photons-million-year-journey-center-sun

[6] J. L. Gray, “The Physics of the Solar Cell,” in Handbook of Photovoltaic Science and Engineering, A. Luque and S. Hegedus, Eds., Chichester, UK: John Wiley & Sons, Ltd, 2011, pp. 82–129. doi: 10.1002/9780470974704.ch3.

[7] “It’s Elemental - The Element Silicon.” https://education.jlab.org/itselemental/ele014.html

[8] “Lithium-Ion Battery Charger for Solar-Powered Systems, CN3083, Rev. 1.1. [Online]. Available; https://pdf1.alldatasheet.com/datasheet-pdf/view/1179262/CONSONANCE/CN3083.html”

[9] X. Yuan, H. Liu, and J. Zhang, Lithium-Ion Batteries: Advanced Materials and Technologies. in Green Chemistry and Chemical Engineering. CRC Press, 2016. [Online]. Available: https://books.google.ca/books?id=FqfMBQAAQBAJ

[10] Y. Zhang, X. Ni, W. Rhee, and Z. Wang, “A 1.8MW 2MB/S chirp-UWB transceiver with burst-mode transmission and slope-based detection,” 2016 IEEE International Symposium on Radio-Frequency Integration Technology (RFIT), 2016.

[11] R. Dutta, A. B. Kokkeler, R. v. Zee, and M. J. Bentum, “Performance of chirped-FSK and chirped-PSK in the presence of partial-band interference,” 2011 18th IEEE Symposium on Communications and Vehicular Technology in the Benelux (SCVT), 2011.

[12] Low Power Long Range Transceiver Module Model No. RFM95W/96W/98W, Version 2.0, HopeRF

[13] R. V. Şenyuva, "Comparison of LoRa-Based Modulations," 2022 30th Signal Processing and Communications Applications Conference (SIU), Safranbolu, Turkey, 2022, pp. 1-4, doi: 10.1109/SIU55565.2022.9864731.

[14] T. Elshabrawy and J. Robert, "The Impact of ISM Interference on LoRa BER Performance," 2018 IEEE Global Conference on Internet of Things (GCIoT), Alexandria, Egypt, 2018, pp. 1-5, doi: 10.1109/GCIoT.2018.8620142.

[15] L. Y. Astutik and R. Yusuf, "Performance Study of LoRa IoT Technology," 2022 12th International Conference on System Engineering and Technology (ICSET), Bandung, Indonesia, 2022, pp. 31-35, doi: 10.1109/ICSET57543.2022.10011110.

[16] “Raspberry pi pico datasheet,” https://www.raspberrypi.com/, 02-Mar-2023. [Online]. Available: https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf.

[17] “TPL5110 data sheet,” TPL5110 data sheet, product information and support | TI.com, Jan-2015. Online]. Available: https://www.ti.com/product/TPL5110.

[18] “DMP2045U data sheet,” https://www.diodes.com, Jul-2021. [Online]. Available: https://www.diodes.com/assets/Datasheets/DMP2045U.pdf.

11. Appendix

Sensor Node Code

```
import board, busio, digitalio
import adafruit_rfm9x
import time, json
from adafruit_seesaw.seesaw import Seesaw

channel_ack = False
data_ack = False
sens_data = 0
fault_log = []

###Initialize pins and objects
def setup():
    global ss, rfm9x, done
    #Soil sensor pins
    i2c = busio.I2C(scl=board.GP1, sda=board.GP0)  # uses board.SCL and 
board.SDA
    #RFM95 pins
    spi = busio.SPI(board.GP18, MOSI=board.GP19, MISO=board.GP16)
    cs =digitalio.DigitalInOut(board.GP17)
    reset = digitalio.DigitalInOut(board.GP20)
    g0 = digitalio.DigitalInOut(board.GP7)

    ###Initialize objects
    ss = Seesaw(i2c, addr=0x36)
    rfm9x = adafruit_rfm9x.RFM9x(spi, cs, reset, 915.0)
    #Initialize rfm9x parameters
    rfm9x.enable_crc = True #Enable CRC Checking
    rfm9x.node = 1    # Node address
    rfm9x.destination = 0    #Destination address

    #this is the timer off (done) signal code 1/2
    done = digitalio.DigitalInOut(board.GP13)
    done.direction = Direction.OUTPUT
    done.value = False

#Read moisture and temperature levels
def read_sens():
    global sens_data
    moist = ss.moisture_read()
    temp = ss.get_temp()
    data = [ {'Time': int(time.time()), 'Temperature': temp, 'Soil Moisture': 
moist} ]
    sens_data = bytearray(json.dumps(data).encode('utf-8'))
    #return sens_data
```

```
#Channel request
def Channel_req():
    global channel_ack
    channel_ack = False
    rfm9x.send('1 Channel request'.encode('utf-8'))     #send request to Hub
    channel_ack_byte = rfm9x.receive(timeout=0.5)      #listen for approval
message
    if channel_ack_byte is not None:
        channel_ack = True      #channel_ack_byte.decode('utf-8')
        #print('Received: {0}'.format(channel_ack))
    else:
        channel_ack = False
        print('Channel Request Failure')
    #return channel_ack

#Send data to central hub
def tx_data(channel_ack, sens_data):
    global data_ack
    data_ack = False
    if channel_ack:
        rfm9x.send(sens_data)       #send data
        #Listen for ACK
        ACK_data_byte = rfm9x.receive(timeout=1.0)
        if ACK_data_byte is not None:
            ACK_data = ACK_data_byte.decode('utf-8')
            print('Received: {0}'.format(ACK_data))
            data_ack = True
        else:
            print('Data ACK failure')
            data_ack = False
    #return data_ack

# Fault Handling
def Fault_log(channel_ack, data_ack, sens_data):
    global fault_log
    if not channel_ack or not data_ack:
        log = json.loads(sens_data.decode('utf-8'))
        fault_log.extend(log)
        print(fault_log)
    else:
        byte = bytearray(json.dumps(fault_log).encode('utf-8'))
        rfm9x.send(byte)       #send the fault log
        print("Fault log sent")
        #Listen for ACK
        Fault_ACK_byte = rfm9x.receive(timeout=5.0)
        if Fault_ACK_byte is not None:
            Fault_ACK = Fault_ACK_byte.decode('utf-8')
            print('Fault ACK Received : {0}'.format(Fault_ACK))
            fault_log = []
            byte = []
        else:
            print('Fault log Transmission FAILURE')

#RFM9x Sleep function
def RFM9x_sleep(sleep_time):
    rfm9x.sleep()
    time.sleep(sleep_time)
    rfm9x.sleep()


# Main code to run indefinetely
def main():
    global sens_data, channel_ack, data_ack
    while True:
        read_sens() # Read sensor values
        Channel_req() # Request central hub channel
        if channel_ack:
            # Transmit sensor data to central hub
            tx_data(channel_ack, sens_data)
            if not data_ack: # retry data transmission on ACK Failure
                for retry in range(1,3):
                    tx_data(channel_ack, sens_data)
                    if data_ack:
                        break  # exit the loop if data_ack is True
        else: # retry channel request on ACK Failure
            for retry in range(1,4):
                Channel_req()
                if channel_ack:
                    tx_data(channel_ack, sens_data)
                    if not data_ack: # retry data transmission on ACK Failure
                        for retry in range(1,4):
                            tx_data(channel_ack, sens_data)
                            if data_ack:
                                break  # exit the loop if data_ack is True
                    break  # exit the loop if channel_ack is True

                time.sleep(5)    # delay 5s before next channel request
retry
        #add to fault log
        Fault_log(channel_ack, data_ack, sens_data)
        #sleep till next read
        RFM9x_sleep(15) #30 seconds
        done.value = True

setup()
main()
```

Central Hub Code

Main hub code (main.py)

```
import board, busio, digitalio
import adafruit_rfm9x
from PIL import Image, ImageDraw, ImageFont
import adafruit_ssd1306
import time, json
import threading

from store import data_store, Fault_log, data_clean, dashboardData, nodeData
from server import create_App, check_values


#Define Transmission Flags
channel_sel = False
data_ack = False


## ************* Initialization Function ************* ##
def  setup():
    global rfm9x, oled, draw, font, image, heading, sensor1, sensor2
    #RFM95 Initialization
    spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
    cs =digitalio.DigitalInOut(board.CE1)
    reset = digitalio.DigitalInOut(board.D25)
    g0 = digitalio.DigitalInOut(board.D24)
    rfm9x = adafruit_rfm9x.RFM9x(spi, cs, reset, 915.0) # rfm9x object module
    rfm9x.enable_crc = True #Enable CRC Checking
    rfm9x.node = 0   # Central hub address

    #Display initialization
    WIDTH = 128
    HEIGHT = 32
    BORDER = 5
    # Define the Pins of the OLED
    oled_reset = digitalio.DigitalInOut(board.D4)
    i2c = board.I2C()  # uses board.SCL and board.SDA
    # Initialize oled object
    oled = adafruit_ssd1306.SSD1306_I2C(WIDTH, HEIGHT, i2c, addr=0x3c,
reset=oled_reset)
    # Create blank image for drawing.
    image = Image.new("1", (oled.width, oled.height))
    # Get drawing object to draw on image.
    draw = ImageDraw.Draw(image)
    # Load default font.
    font = ImageFont.load_default()

    # Initialize the app server with the last saved sensor data
    check_values()
    server_run_thread = threading.Thread(target=create_App)
    server_run_thread.daemon = True
    server_run_thread.start()

## ************* Transceiver Functions ************* ##
# Channel selection
def Channel_sel():
    global ch_sel
    ch_sel = False
    print('Listening for Request')
    channel_req_byte = rfm9x.receive(timeout=20.0)  #listen for node request
    if channel_req_byte is not None:
         channel_req = channel_req_byte.decode('utf-8')
         if int(channel_req[0]) == 1:
            rfm9x.destination = 1
            rfm9x.send('Ch1 ACK'.encode('utf-8'))
            print('Ch1 ACK sent')
            ch_sel = True
         elif int(channel_req[0]) == 2:
            rfm9x.destination = 2
            rfm9x.send('Ch2 ACK'.encode('utf-8'))
            print('Ch2 ACK sent')
            ch_sel = True
         else:
            rfm9x.destination = 255
            print("sent to all")
            ch_sel = False
    else:
         print('Channel Select Failed')
         ch_sel = False
# Receive packet from sensor node
def rx_data():
    global data_ack, temp, moist
    data_ack = False
    moist = 0
    temp = 0
    print('Listening for data...')
    sens_data_byte = rfm9x.receive(timeout=5.0)
    if sens_data_byte is not None:
        sens_data = sens_data_byte.decode('utf-8')
        print('Received: {0}'.format(sens_data))
        data_list = json.loads(sens_data)
        data_dict = data_list[0]
        temp = data_dict["Temperature"]
        moist = data_dict["Soil Moisture"]
        if moist or temp != 0:
            rfm9x.send("Data ACK".encode('utf-8')) # send ACK
            data_ack = True
            return sens_data
    else:
        print('Data NOT recieved')
        data_ack = False
        return 0
# Listen for any fault data
def RxFault():
    print('Listening for Fault data...')
    fault_data_byte = rfm9x.receive(timeout=5.0)
    if fault_data_byte is not None:
        rfm9x.send("Fault ACK".encode('utf-8')) # send ACK
        fault_data = fault_data_byte.decode('utf-8')
        print('Received: {0}'.format(fault_data))
        return fault_data
    else:
        print('No Fault Data recieved')
# Sleep mode for rfm9x
def Rfm9x_sleep(sleep_time):
    rfm9x.sleep()
    time.sleep(sleep_time)
    rfm9x.sleep()
# Main Channel Selection and receiver code with error retrials
def Channel_Selection():
    global ch_sel, data_ack, temp, moist, rfm9x, node1, node2
    Channel_sel()   #Listen for node requests
    if ch_sel:
        data = rx_data() # Receive sensor data
        if not data_ack:
            data = rx_data()
            if data_ack:
                fault = RxFault() # Listen for previous fault data
                #update csv files
                if rfm9x.destination == 1:
                    data_store(data, 'node1.csv')
                    if len(fault) != 0:
                        data_store(fault,'node1.csv')
                    node1 = True
                elif rfm9x.destination == 2:
                    data_store(data, 'node2.csv')
                    if len(fault) != 0:
                        data_store(fault,'node2.csv')
                    node2 = True
        else:
            fault = RxFault() # Listen for previous fault data
            #update csv files
            if rfm9x.destination == 1:
                data_store(data, 'node1.csv')
                if len(fault) != 0:
                    data_store(fault,'node1.csv')
                node1 = True
            elif rfm9x.destination == 2:
                data_store(data, 'node2.csv')
                if len(fault) != 0:
                    data_store(fault,'node2.csv')
                node2 = True
    else:   # Retry channel Selection
        Channel_sel()
        if ch_sel:
            data = rx_data() # Receive sensor data
            if not data_ack:
                data = rx_data()
                if data_ack:
                    fault = RxFault() # Listen for previous fault data
                    #update csv files
                if rfm9x.destination == 1:
                    data_store(data, 'node1.csv')
                    if len(fault) != 0:
                        data_store(fault,'node1.csv')
                    node1 = True
                elif rfm9x.destination == 2:
                    data_store(data, 'node2.csv')
                    if len(fault) != 0:
                        data_store(fault,'node2.csv')
                    node2 = True
            else:
                fault = RxFault() # Listen for previous fault data
                #update csv files
                if rfm9x.destination == 1:
                    data_store(data, 'node1.csv')
                    if len(fault) != 0:
                        data_store(fault,'node1.csv')
                    node1 = True
                elif rfm9x.destination == 2:
                    data_store(data, 'node2.csv')
                    if len(fault) != 0:
                        data_store(fault,'node2.csv')
                    node2 = True
## End of Transceiver Functions

## ************* Display Function ************* ##
def Display(page, temp, moist):
    global oled, draw, font, image
    # Clear display and set background
    oled.fill(0)
    oled.show()
    # Draw the time
    current_time = time.strftime("%H:%M")
    draw.text((0, 0), current_time, font=font, fill=255)
    # Draw sensor
    sens = "Sensor " +str(page)
    (font_width, font_height) = font.getsize(sens)
    draw.text((oled.width - font_width, 0), sens, font=font, fill=255)
    # Draw the temperature
    text = "Temperature: {:.1f} °C".format(temp)
    (font_width, font_height) = font.getsize(text)
    draw.text(((oled.width - font_width) // 2, ((font_height + 2 )* 2) // 2),
text, font=font, fill=255)
    # Draw Soil Moisture
    text = "Moisture: {:.1f}".format(moist)
    (font_width, font_height) = font.getsize(text)
    draw.text(((oled.width - font_width) // 2, (oled.height - font_height)),
text, font=font, fill=255)

    # Display image
    oled.image(image)
    oled.show()
## End of Display Function

## ************* App Server Functions ************* ##

def main():
    global ch_sel, data_ack, temp, moist, rfm9x, node1, node2
    while True:
        Channel_Selection()
        if ch_sel and data_ack: #data received correctly
            Channel_Selection() #repeat for a second node
            if not(ch_sel and data_ack):
                Fault_log(rfm9x.destination, ch_sel, data_ack)
        else:
            Fault_log(rfm9x.destination, ch_sel, data_ack)

        #=====================#
        #update serverside code
        check_values()
        #=====================#

        # Display the most recent values on the oled screen
        header = dashboardData()
        sensor_1 = header[0]
        sensor_2 = header[1]

        temp1 = float(sensor_1['Temperature'])
        moist1 = float(sensor_1['Soil Moisture'])
        temp2 = float(sensor_2['Temperature'])
        moist2 = float(sensor_2['Soil Moisture'])
        display_counter = 0
        page = 1
        while display_counter <10:
            if page == 1:
                Display(page, temp1, moist1)
                page = 2
            else:
                Display(page, temp2, moist2)
                page = 1
            time.sleep(5) # Swap pages every 5 seconds
            display_counter += 1

## Run the program
setup()
main()
```

Storage code (store.py)

```
import csv, json, time

time_threshold = 15780000       #6 months to retain data

def data_store(sens_data, filename):
    field_names = ['Time', 'Temperature', 'Soil Moisture']
    load_data = json.loads(sens_data)

    with open(filename, mode='a', newline='') as csv_file:
        writer = csv.DictWriter(csv_file, fieldnames=field_names)
        for row in load_data:
            writer.writerow(row)

def data_clean(filename, time_threshold):
    clean_flag = False
    while not clean_flag:
        #Data delete after
        with open(filename, 'r') as file:
            reader = csv.reader(file)
            data = list(reader)

        oldest_timestamp = int(data[1][0])

        if time.time() - oldest_timestamp > time_threshold:
            del data[1]
            #field_names = ['Time', 'Temperature', 'Soil Moisture']

            with open(filename, 'w') as file:
                # headerwrite = csv.DictWriter(file, field_names)
                # headerwrite.writeheader()

                writer = csv.writer(file)
                writer.writerows(data)
        else:
            clean_flag = True

def nodeData(filename):
    with open(filename, 'r') as file:
        # Create a CSV reader object
        reader = csv.DictReader(file)
        entries = []

        # Iterate through the rows in reverse order, up to 24 rows
        for row in reversed(list(reader)):
            if len(entries) >= 24:
                break
            # Convert the row to a dictionary and add it to the entries list
            entries.append(dict(row))
    return entries

def dashboardData():
    heading = []
    # Loop through the two CSV files
    for file_id, filename in enumerate(['node1.csv', 'node2.csv'], 1):

        # Open the CSV file for reading
        with open(filename, 'r') as file:
            # Create a CSV reader object
            reader = csv.reader(file)

            # Get the last row in the CSV file
            last_row = None
            for row in reader:
                last_row = row

            # Create a dictionary from the last row and add the file id
            entry = {'Sensor_id': file_id, 'Time': last_row[0],
'Temperature': last_row[1], 'Soil Moisture': last_row[2]}

            # Add the dictionary to the data list
            heading.append(entry)
    return heading

def Fault_log(node_num, ch_sel, data_ack):
    print('Fault Logging ...')
    if not(data_ack and ch_sel) is True:
        # Open the CSV file for writing
        with open('fault_log.csv', mode='a', newline='') as csv_file:
            # Create a CSV writer object
            writer = csv.writer(csv_file)
            current_time = int(time.time())

            # Write the time and fault type to the CSV file
            writer.writerow([node_num, current_time, 'Fault: Channel 
Selection' if not ch_sel else 'Fault: Data Not Received'])
```

Server-side code (server.py)

```
# Import the Flask and jsonify classes from the flask module
from flask import Flask, jsonify
from apscheduler.schedulers.background import BackgroundScheduler
from store import dashboardData, nodeData


def check_values():
    global heading, sensor1, sensor2
    heading = dashboardData()
    sensor1 = nodeData('node1.csv')
    sensor2 = nodeData('node2.csv')

def create_App():
    global heading, sensor1, sensor2
    app = Flask(__name__)

    @app.route('/dashboard')
    def get_dashboard_data():
        # Create a List containing the Node # and recent data
        data = heading
        return jsonify(data=data)

    @app.route('/sensor1')
    def get_sensor1_data():
        data = sensor1
        # Return a JSON response with the data list
        return jsonify(data=data)

    @app.route('/sensor2')
    def get_sensor2_data():
        data = sensor2
        # Return a JSON response with the data list
        return jsonify(data=data)

    # Start the APScheduler to run check_values() every minute
    scheduler = BackgroundScheduler()
    scheduler.add_job(check_values, 'interval', minutes=1)
    scheduler.start()

    app.run(host='0.0.0.0', port=5000)
```