Garden Management System

4th Year Project Submission

Peter Colls

15th June 2023

Table of Contents

Executive Summary

As part of the 4th year assessment, the scope of the project must include:

- RF wireless technology.
- Digital solutions.
- Micro and analogue electronics.
- Deliver a beneficial, well-designed product with marketing potential.

The Garden management system was selected as it addressed all the above criteria, and the scope could be developed so that it was reasonably unique in the marketplace.

Throughout this document, each of the five components are discussed in detail, therefore, to assist the reader each of the components will be referred as follows:

1. Sensor Unit.
2. Master Unit.
3. Cloud Control Unit.
4. Arduino Cloud.
5. Phone App

Sensor Unit

Data is collected by the sensors, including Sensor Id, soil and air temperature, soil moisture, and battery voltage. This information is sent to the Cloud Control Unit. The system then receives the minimum soil moisture and water valve data from the sensors. To ensure the minimum soil moisture level is maintained, water is supplied to a maximum of eight garden sections through individual garden valves. Multiple sensors may be present in each garden section, but only one water valve is permitted. Watering is only executed when the weather is dry and the moisture level in the garden section drops below the minimal soil moisture level indicated by the Phone App.

Master Unit

The Unit regulating data flow between the Sensor and Phone App is the Master Unit. Its main function is distributing the appropriate amount of water necessary to maintain optimal moisture levels. Additionally, it sends and receives information from the Cloud Control Unit.

The Phone App operates the system, with the central control unit known as the Master Unit playing a crucial role in managing the following functions: System Communications, which involves exchanging data between sensors, collecting information such as sensor ID, soil and air temperature, soil moisture, and battery voltage.

The Phone App receives the garden control Values then transmits the collected information to the Cloud Control Unit, where the Garden Watering Control Information is re-transmitted to the Mater Unit.

On receiving the minimum soil moisture level and watering duration for each Sensor Unit. The system asses the moisture levels and if required the Master Unit will supply water to the garden via eight separate water valves catering to the moisture needs of eight different garden sections. A single garden section can have multiple sensors, but only one water valve.

Cloud Control Unit

The main purpose of this device is to establish a channel for data transmission between the Master Unit and the Phone App. This allows for the transfer of soil sensor readings to the Phone and the exchange of garden watering information to the Master Unit.

Arduino Cloud.

The Arduino IoT Cloud is a platform that allows anyone to create IoT projects, with a user-friendly interface, and an all in one solution for configuration, writing code, uploading and visualization.

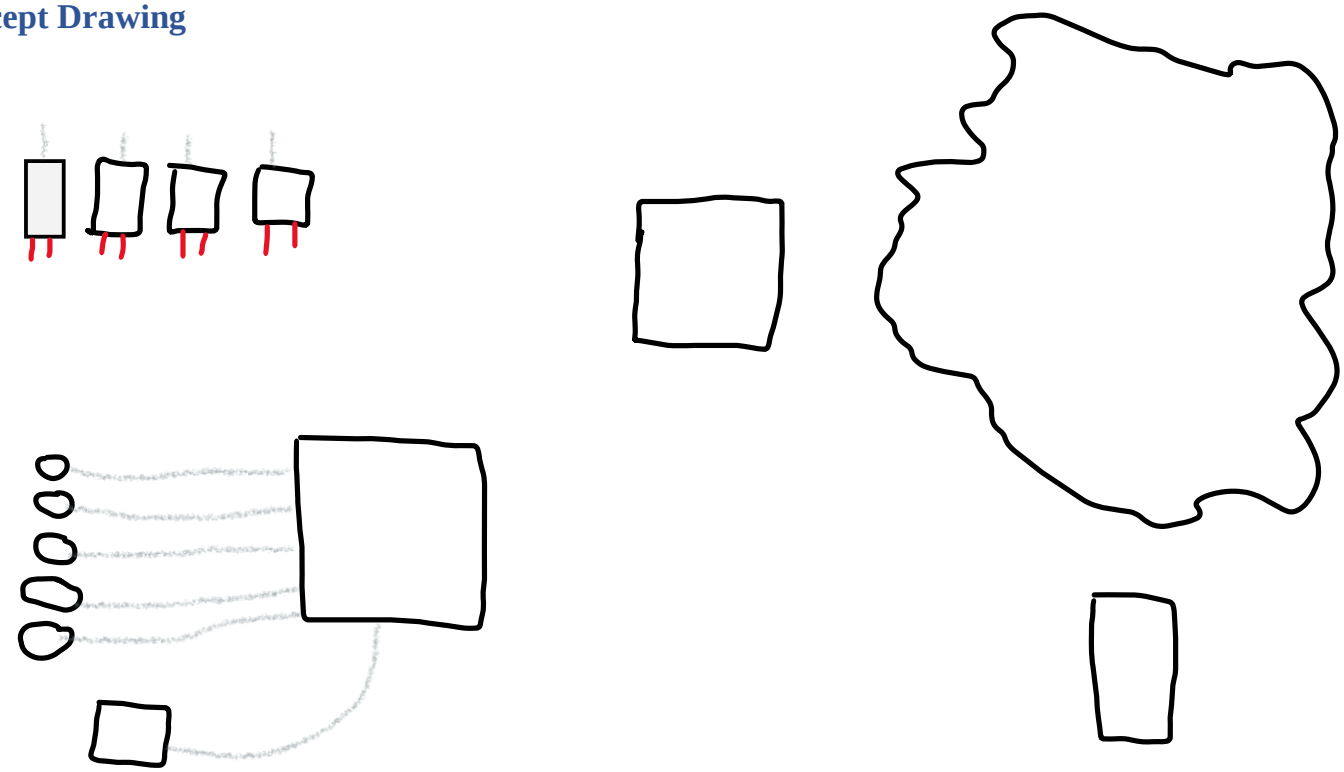
Phone App

You can use our Phone App to water up to eight different sections of your garden using separate water valves. Each section can have multiple sensors, but only one water valve. The App will ensure that each section receives water only, when necessary, to maintain the minimum soil moisture level. If it's raining or the section already has enough moisture, the water valve won't turn on, preventing overwatering.

User preferences can be managed through the phone app. The User can input their Garden watering information such as Unit Number, Valve Number, and Maximum Watering Duration. When modifications are made, the data is automatically sent to the Master Unit.

To be completed by Dianne

Concept Drawing



Acknowledgements

The persons and companies below have been critical to the completion of this project, I offer my sincere thanks to them all.

- John D. Lenk – Book “Simplified design of Micropower and Battery Circuits. Source of information for the Use of the LT1300 power management system.

- Andre LaMothe – University Lecturer, significant help in all arears.

- The following YouTube sites have been invaluable for their advice with the coding of the ESP32 and a number of components.
    - Andreas Spiess
    - Bill at the DroneBot Workshop
    - Rui Santos at Random Nerd Tutorial
    - Lee Wiggins

- Applications Used
    - NI Multisim
    - Ni Ultiboard
    - KiCad
    - Design Spark Mechanical CAD
    - Arduino IDE
    - Visual Source Code
    - Arduino IOT Cloud

- Suppliers used and provided invaluable help on components and techniques.
    - PCBWay
    - JLCPCB – 3D Printing
    - Digi Key
    - Muser
    - Ali Express
    - Jaycar
    - Battery World

Glossary of Terms

Analogue electronics.	Non-Micro, e.g., Capacitors, resistors Diodes Etc
Arduino IOT Cloud	Arduino Cloud site - The Internet of Things (IoT) describes the network of physical objects
ADC Battery Volts	ADC "Analogue Digital Conversion" A function used to convert electrical voltage to a digital number
C++	A software language used for technical applications
Copper pour, vias	Terms for PCB development processes
Deep-sleep function	Putting the Microcontroller into sleep to save power
ESP32 microcontroller	A brand of microcontroller (small computer for controlling devices)
Espressif Systems	A Microcontroller production company
IDE's.	An integrated development environment (IDE) is a software application that helps programmers develop software code
I2C	Pronounced " I square C" the name of a communications protocol
Net Names	Nets are usually given a name that specifically states the purpose of signals on that wire. For example, power nets might be labelled "VCC" or "5V", while serial communication nets might be labelled "RX" or "TX".
NI Multisim, Ni Ultiboad and KiCad	Names of application suppliers for Circuit schematic and PCB development
On-Change" function	Software function - When a Variable content change - take action.
Point-to-point network	Wireless network to connect specific items together
PCB	A printed circuit board
PCBWay	PCB board production company
Panelisation	PCB Panelisation is a technique through which small-size boards of a single design are created and linked in the shape of the array in a single board) to reduce the cost of quantity production runs.
RF wireless technology	RF is a term used to describe Radio Frequency
Microelectronics.	Micro - Microcontroller or IC Chips
Four-layer stackup	The number of layers applied to a PCB board
Surface-mount (SMD)	Surface mount device - very small electronic components
TFT display screen	Small display screen used to display test data
TX and RX	Transmit and Receive data signals

Introduction

The initial objectives for the scope of this project are to encompass:

- RF wireless technology.
- Digital solutions.
- Micro and analogue electronics.
- Deliver a beneficial, well-designed product with marketing potential.

With the above objectives in mind, I researched the following for consideration:

- Home communication system with several transmit / receiver units using several dedicated selectable frequencies.
- Garden Management system that used wireless technology, multiple Sensors to control the watering requirement and summarise the sensor information on a smart phone.
- Doorbell answering system that uses Wi-Fi to send visual and sound data to a smart phone.

My findings are as follows:

- The Home communication system was interesting and technically changeling, but the selection and use of public available frequencies proved to be impractical.

- The Doorbell answering system was rejected due to the amount and wide selection of products currently on the market.

- The Garden management system was selected as it addressed all the above criteria, and the scope could be developed so that it was reasonably unique in the marketplace.

Component Naming Standards

The Garden Management System comprises of 5 primary components:

1. Soil and Environment Sensor Unit/s.
2. Master Communication and Water Control Unit.
3. Phone Information and control system.
4. The Arduino IOT Cloud.
5. The Phone Application.

Throughout this document, each of the above five components are discussed in detail, therefore, to assist the reader each of the components will be referred as follows:

1. Soil and Environment Sensor Unit/s – Sensor Unit.
2. Master Communication and Water Control Unit – Master Unit.
3. Phone Information and control system – Cloud Control Unit.
4. The Arduino IOT Cloud – Arduino Cloud.
5. The Phone Application – Phone App

Project Concepts

Overview

The Garden Management System comprises of three sections:

1. Soil and Environment Sensor Unit/s (Sensor Unit) – The Sensor Units collect the primary information to allow the watering system to maintain the correct moisture levels to promote healthy plant growth. The principal characteristics are:
    a. The independent sensor control units can be placed up to 320 meters (free line of sight) from the Master unit.
    b. A maximum of fifteen sensor units can be located throughout the garden. Each unit reads soil moisture, ground, and air temperature, plus optionally, PH and Nitrogen levels.
    c. The sensor units can be inserted into the ground to a depth of 15cm to obtain consistent environmental readings.
    d. To support battery life, the microcontroller is placed into a four-hour deep sleep cycle. On completion of the Deep Sleep cycle, the microcontroller reads all the sensor information and transfer the data to the Master Unit. Using the deep-sleep function in this manner yields a battery life of one-year.
    e. The Master Unit has been designed to manage the data transactions from a maximum of 30 Soil Sensor Units; however, due to the limited space to display information on the phone, the practical number of soil sensors is restricted to 15.
    f. Each Sensor Unit measures the soil and air temperature, soil moisture, battery voltage on a 4-hour cycle. The primary reason for this time cycle is to maintain a one-year battery life. The design has included a set-up option to allow the user to customise the sensor unit for a preferred number of soil sensor readings per day. This would be at the cost of Battery life.
    g. The sensor units do not communicate directly with the Phone app and cannot astatine the User's requested minimal soil moisture levels. Therefore, the design provides a global minimum soil moisture level. If the Soil moisture reading is less than the global minimum soil moisture level, then the Sensor Unit will only sleep for 30 minutes and read/transmit the soil measurements once more to prevent overwatering.
    h. The design encompasses the ability to collect both PH and Nitrogen soil readings, but this feature will not be available in the first production release.

2. Master Communication and Water Control Unit (Master Unit) – The Master Unit delivers the central control functionality that manages the:

    a. System Communications - Data communications between all the Sensors that collect the Sensor ID number, soil and air temperature, soil moisture and battery voltage.
    b. Updates Phone App - Transmits the received sensor information to the Cloud Control Unit.
    c. Garden Watering Control information - Receives from the phone app. The minimum soil moisture and watering time duration information for each Sensor Unit.
    d. Watering of the Garden - To maintain the minimal soil moisture level. Water is applied to the garden via eight separate water valves servicing the needs of eight garden sections. A single garden section may have multiple sensors but only one water valve. If the soil moisture level is less than the minimum soil moisture value, water is applied to address the minimum moisture level supplied by

the Phone App. Thus, if the weather is rainy, the garden moisture levels will be greater than the minimal soil moisture levels, and watering will not occur.

e. The Master Unit has been designed to manage the data transactions from a maximum of 30 Soil Sensor Units; however, due to the limited space to display information on the phone, the practical number of soil sensors is restricted to 15.

3. Phone Information and control system (Cloud Control Unit) – The Cloud Control Unit establishes a point-to-point network between the Master Unit and Phone Application. It transfers the latest soil sensor data from the Master Unit to the Phone Application. It downloads the minimum soil moisture and control values from the Phone Application to the Master Unit.

Master Communication and Water Control Unit (Master Unit)

The master Unit is designed to control the data flow between the Sensor Unit's and the Phone App, and distribute the correct amount of water to maintain the minimum moisture levels. Then send and receive information to and from the Cloud Control Unit.

Power

Power is provided by a 12v 1.2 ah battery, charged by a 21v, 10W, 22cm X 30cm Solar Panel.
The Master Unit houses the battery charging circuit that maintains a full 12.7v charge. The 12v power rail drives the relays, and water valves, and the 3.3v power rail for the ESP32 and microelectronics.

Manage user preferences.

The phone app allows the User to input the systems Garden watering information. Unit Number, Valve Number and Maximum Watering Duration; and "on change" send the modified data to the Master Unit.

- *Sensor control data*.
  - Minimum moisture level
  - Related Water Valve No
- *Garden Section data*.
  - Water Valve No
  - Maximum watering time

The system can power up to eight water valves, with many Sensor Units that can relate to one water valve. The Master Unit stores this information in permanent memory to use, to control, and minimise the water applied to the garden sections, thereby avoiding overwatering.

Phone Information display and control input system (Cloud Control Unit)

The Cloud Control Unit forms the information conduit between the Master Unit and the Arduino IOT Cloud:

- The Cloud Control Unit uses a dedicated ESP32 microcontroller to connect the Master Unit to the Arduino IOT Cloud. This enables the transmission of information between the Phone App and the Cloud Control Unit. The ESP32 is located within the Master Unit enclosure and uses point-to-point and RS232 communications.
- The Arduino IOT Cloud uses Wi-Fi for data transmission and requires the User's Wi-Fi Login Details.

Design

ESP32 Microcontroller

The ESP32 is a microcontroller with integrated Wi-Fi and dual-mode Bluetooth. includes a built-in antenna power amplifier, low-noise receive amplifier, filters, and power-management modules. ESP32 was created and developed by Espressif Systems, a Shanghai-based Chinese company.

The ESP32 microcontroller has been used within all modules of this project, due to its capacity, flexibility and performance. C++ is used for all software via the Arduino and Visual Studio Code IDE's.

Point to Point Communications

ESP-Now is a communication protocol developed by Espressif Systems. It is designed for efficient and low-power communication between ESP32 devices without the need for a traditional Wi-Fi network infrastructure.

Overview of ESP-Now and its key features:

1. ESP-Now is primarily intended for local communication between ESP8266 and ESP32 devices within a limited range, typically within a single building or small area, using line-of-sight transmission a distance up to 320 meters can be considered reliable. ESP-Now enables direct device-to-device communication without the need for an intermediate access point or router.

2. ESP-Now utilizes the 2.4 GHz ISM (Industrial, Scientific, and Medical) band to establish a point-to-point or point-to-multipoint communication link. It employs the IEEE 802.11 Wi-Fi protocol as its foundation but strips away most of the Wi-Fi stack to reduce overhead and improve efficiency.

3. One of the main advantages of ESP-Now is its ability to operate in low-power modes, making it suitable for battery-powered devices and IoT applications. The protocol allows devices to quickly establish a connection, transmit data, and then return to sleep mode, conserving power and extending battery life.

4. ESP-Now offers a simplified API, making it easy to implement and use for data transmission. It provides a fast and reliable connection with minimal latency and overhead, allowing for efficient transmission of small data packets to a maximum of 256 bytes.

5. ESP-Now provides basic data encryption to ensure secure communication between devices. It utilizes a pre-shared key (PSK) for encrypting and decrypting the data packets, preventing unauthorized access to the transmitted information.

6. ESP-Now is a versatile protocol that finds applications in scenarios such as home automation, sensor networks, wireless sensor nodes, remote control systems, and other IoT projects where simple and low-power communication between ESP devices is required.

Scalability

The Garden Management System is expandable by adding additional Sensor units or water valves. The system can accommodate fifteen Sensor Units and eight water valves. The Sensor Units can be placed anywhere in the garden, and multiple Sensor Units can control a single water valve.

Easy to Use

Ease of use is a primary objective; unlike other garden watering systems, the user is not required to establish a comprehensive watering schedule (What days to water, time of day and how long Etc.) The User enters the minimum moisture levels and maximum watering duration via the Phone App. The Master Unit controls the gardens watering requirement. No more modifying the watering schedule due to seasonal change or rainy days.

Minimum Maintenance

The only maintenance required, is to change the Sensor Unit 3 X AA batteries when indicated by the Phone system.

PCB Board Manufacturer and Assembly

My main goal is to create the PCB boards in the most efficient way possible, with a goal to produce them in large quantities. To achieve this, I am focusing on using best practices in designing the schematic, developing the printed circuit board (PCB), and assembling its components. Here are some essential factors and methods I am considering for these processes:

PCB Board Development:

1. Schematic Design – The schematic demonstrates a well-defined design that accurately represents the intended functionality and structure for the PCB. Components are presented in groups as per their functionality with extensive use of Net Names for power and signal circuits. All components use standard symbols, pinouts, and, where possible, the manufacture footprints as defined by the datasheets. PCB trace specifications are included with the Net Name Specifications, and a comprehensive set of electronic rules has been included.

2. PCB Layout – Attention has been paid to component placement, trace routing, and signal integrity. To reduce signal noise and improve the integrity, I have used a four-layer stackup for the primary controlling circuits for both the Sensor and Master Units. Layers 1 and 4 hold the signal traces, with Layer 2 for the ground plane and Layer 3 for the power planes.

3. PCB Production and Manufacturability - I have used NI Multisim, Ni Ultiboad and KiCad for the schematic and PCB design. Design rule checks (DRC) are in line with PCBWay standards have been used to ensure compliance with fabrication and assembly guidelines.

4. Panelisation – Consideration has been given to PCB Board Panelisation for the two smaller boards, the Master Unit Bottom and the Ten micro–LED PCB's.

5. Thermal Considerations - Thermal management uses Heat sinks, copper pour, vias, and thermal relief pads to dissipate heat effectively.

6. EMI/EMC Design - I have used proven electromagnetic interference (EMI) and electromagnetic compatibility (EMC) design practices by minimising signal traces lengths, good grounding techniques, and capacitor filtration solutions to reduce noise and interference.

Component Assembly:

1. Initial Prototype system – The PCB Boards for the initial prototype have been manufactured by PCBWay and assembled and tested by the author. PCB Boards for future production runs will be manufactured, components assembled and tested by PCBWay.

2. Component Selection – All components are RoHS Compliant and purchased from Digi Key, Muser, Element 14 and Ali Express. Most are surface-mount (SMD) using standard packages 1206, 0806 etc.

3. Bill of Materials (BOM) - A complete and accurate BOM has been included for each System Unit encompassing component schematic and PCB reference numbers, description specifications, manufacturer and supplier part numbers, footprints, costs, and procurement sources. Refer Appendix X

Garden Sensors Unit

- The garden sensor unit is designed to undertake multiple categories of soil and environment readings:
  1. Soil Moisture
  2. Soil Temperature
  3. Air Temperature
  4. Battery voltage
  5. Soil PH (optional)
  6. Soil Nitrogen (optional)

- Provide a long-life battery power supply from three AA batteries.
- Use Wireless data communications.
- Robust construction - A custom-made enclosure is used to house the microcontroller, sensors, and electronics. As most of the unit is buried in the soil, the enclosure must be waterproof and robust.

Master Unit

- The Master Unit manages the amount of water applied to the garden and provides the central control for data transaction between the Sensor Units and the Phone Cloud system:

  1. Received Soil and environment data from all sensor units.
  2. Sends and receives information to the Cloud information Unit.
  3. Manages all watering scenarios.
  4. Power independent with the use of sola battery charging system.
  5. Robust and weather resistant.
  6. Small and easy to mount.

Cloud Control Unit

The Cloud information unit was designed to be as minimalistic as possible. It is housed with the Master Unit and receives the 3.3v power supply; the TX and RX pins are used for the optional RS232 data transmission, and no other IO pins are used. It is designed to use one of two data transmission options to transfer data to and from the Master Unit:

1. Point to Point via ESP-Now. ESP-NOW is a wireless communication protocol defined by Espressif, which enables the direct, quick, and low-power control of smart devices without the need for a router. ESP-NOW can work with Wi-Fi and Bluetooth. It's widely used for inter-processor communications.

2. RS232 is a serial communication standard that uses a short twisted-pair cable to transmit/receive data. As the microcontrollers are on the same PCB board, short traces can connect the RX and TX pins, ensuring reliable data communication.
   The use of RS232 protocol within the design of this project is to allow for a future data transmission option, that requires minimal software space and the non-requirement of large operational libraries; thus, providing increased CPU space to drive the Phone app directly in the future.

Functions and Specifications

Sensor Unit

Enclosure box

- The Sensor Unit Box is specially crafted, and 3D printed using nylon material. It is created to be waterproof as it will be buried to a depth of 150mm. The lid comes in two sections; the first is 120mm long and is fixed permanently while the second is 50mm long and can be removed to change the batteries. Inside the box, are all the necessary components such as sensors, PCB circuit board, microcontroller, and batteries. The sensors and on/off switch are made waterproof by sealing them with silicon.
- Size: 201 x 96 x 35.
- PCB and battery stand 12mm high.
- Custom-made.
- 3D Printed.
- Power source: 3 X AA batteries.
- Material: Nylon.

PCB Board

The PCB Board provides the circuit for the sensors, microcontroller, and the power supply. A 14-pin header is included to allow for a 3.5” TFT display screen to be used for diagnostics.

- Size: 100 x 70
- Layers: 4

Circuit Functions

- Wake-up and Test Switches – The switches are used to wake up the microcontroller from deep sleep and to allow the software to switch on the logic to display the diagnostic information to the display screen.
- Ground Temp – The system uses a DS18B20 temperature sensor with a range of -55 – 125c.
- Soil Moisture – The system uses a STEMMA I2C Multiple Function Soil moisture and Air temperature Sensor.
- Air Temp – Refer above.
- Battery Voltage – Battery voltage is calculated by using an ADC (Analogue to Digital Conversion) circuit with data sent to the microcontroller for processing.

Circuit Design

- 1-year battery life power supply – It has been determined that a 3 X AA battery can last for up to 348 days by utilizing a microprocessor that goes into a deep sleep 4-hour cycle. With a battery capacity of 8,400 mAh, a control circuit is implemented using the LT1300 Buck-Boost Switching Regulator. This circuit ensures a constant 3.3v voltage supply from a battery voltage range of 4.7v to 1.8v with minimal overhead. -Refer Diag - 8
- Ground Temp Sensor - DS18B20 temperature sensor powered by the 3.3v power rail and a single data signal held high by a 4.7k resistor. -Refer Diag - 1
-

- I2C Moisture Sensor - STEMMA Multiple Function Soil moisture and Air temperature Sensor powered by the 3.3 power rail. Data is provided by an I2C transmission protocol to the Microcontroller. -Refer Diag - 11
- Air Temp Sensor- Refer above.
- ADC Battery Volts – The circuit provides two options to acquire the battery voltage: -Refer Diag – 12, 13,14
  1. Via a voltage divider resistor circuit that passes the signal voltage to the built in ADC within the ESP32. This solution, although straightforward, does not yield consistent voltage values.
  2. Using a MCP3204 12 Bit Analog to Digital Converter, with a REF191GS 2.048v voltage reference, together with a source voltage divider to send voltage information to the microcontroller will be efficient. Although this circuit option is complex and costly, the result is a very accurate battery voltage reading.

  Both options have been included with the PCB circuit design.

- Reverse voltage protection – If the User replaces the battery incorrectly. The circuit provides reverse voltage protection to protect the microcontroller and components from damage. -Refer Diag - 15

- TFT 3.5 Screen for Testing – The circuit has all the traces to connect the TFT 3.5 Screen to the microcontroller via a 14-pin header.
- On/Off Switch – A single pole switch connects or disconnects the primary battery voltage supply. -Refer Diag - 12

- Wake-up Button – The microcontroller is programmed to receive an interrupt wake-up signal via a Tact switch. -Refer Diag - 5

- Test Button – The microcontroller is programmed to receive a Test signal via a Tact switch. -Refer Diag - 9

Software

Except for the “Send Sensor data to Host Processor” the software features below follow the circuit descriptions above. Therefore, to avoid duplication, I have included the software line reference numbers for each topic to allow the reader to view the software code for their interest.

Send Sensor data to the host processor – All three microcontrollers (Sensor Unit, Unit and the Phone Information display and control input system) use the Point-to-Point communication feature of the ESP32 known as ESP-Now. Refer to “General Concepts” above.

After the 4-hour deep sleep cycle, the Sensor Unit system will wake up and read the soil moisture, soil and air temperature and current battery voltage. Together with the Sensor Unit number, this data is sent to the Master Unit using ESP-Now.

The Sensor Unit data is sent to the Master Unit for processing and transfer to the Cloud Control Unit.

For detailed Software information regarding the categories below – refer Appendix 1 – Sensor Unit Software

- Send Sensor Unit data to the host processor.
- Battery Volts.
- Ground Temp.
- I2C Moisture.
- Air Temp.
- 4 Hr. – Deep sleep cycle.

- Wake up interrupt function.
- Test software and live display data.

Master Unit (Master Unit)

Enclosure box

The box is a standard Polycarbonate Enclosure with Mounting Flange and clear lid from Jaycar part number HB6223. The enclosure holds the 12v battery, PCB circuit board, two microcontrollers (Master and Cloud Control Units) and water valve relays. The Aerial Socket and on/off switch are waterproofed by sealing with silicon.

The battery charging circuit displays the battery charge status via 10 colored LEDs which can be viewed through the clear lid.

- Size: 170 x 120 x 90.
- Power source:
  1. 1 X 12v 1.2 AH battery.
  2. 1 x 21v 10W 22cm 30m Solar Panel.

PCB Boards

Two PCB Boards, stacked on top of each other and connected via a 10-pin header. Together, they provide the circuits for the two microcontrollers, sola charging, power supply and water relays. A 14-pin header is included to allow for a 3.5" TFT display screen to be used for diagnostics.

- Board 1 size: 87mm X 87mm 4 Layer
- Board 2 size: 110mm X 60mm 2 Layer

Circuit Functions

- Receive soil data from Sensor Units – On a 4-hour cycle the Sensor Unit reads the soil moisture, soil and air temperature and battery voltage, then collate the data for point-to-point transmission to the Master Unit using ESP-Now. On receiving the new Sensor Unit information, the Master unit will undertake two actions:

  1. Implement watering requirements – Via the Sensor reference number, use the soil moisture and control data saved in permanent memory, compare the latest soil moisture reading with the minimum value. If the new value is lower, activate the watering system based on the sensor watering parameters.
  2. Send soil sensor data to the Cloud Control Unit – Send the new soil sensor information to the Cloud Control Unit using ESP-Now point-to-point data transmission.

- Receive soil moisture and control information from the Cloud Control Unit – To configure the soil moisture and control data for each Soil Sensor Unit (up to 15), the User can utilise the Phone App. The information is saved permanently within the app and can be modified as the User needs. The information is referenced by Soil unit number with the values as follows:
  - Minimum moisture value – used to trigger the water valve.
  - Maximum water time – Sets the maximum watering period.
  - Water Valve number – the water valve number associated with this sensor No. If watering is required, this valve is turned on.

If the User modifies any of the above values. The software triggers an “On-Change” function, where the Cloud Control Unit transmits the changed values to the Master Unit. On receiving the data packet, the Master Unit will update the values in permanent memory.

- Battery Voltage – Battery voltage is calculated by using an ADC (Analogue to Digital Conversion) circuit with data sent to the microcontroller for processing.

Circuit Design

- Battery Charger – The battery charging circuit monitors the battery voltage and, as it approaches 12v, then turns on the charge supply to apply 14.4 volts to the battery, slowly reducing to 13.3 volts as the battery becomes fully charged. The source voltage is generated by a 21v 10W 22cm 30m Solar Panel connected to an LM338 voltage regulator. Then monitored by LM3915 dot LED display, as the battery voltage approaches the charge voltage an BC557 PNP transistor will go to reverse bias shutting down the charge voltage and ending the charging cycle. -Refer Diag - 1

- Power supply – The Master Unit uses two voltage power planes, 12v for the relay circuits and 3.3v for the microcontrollers and ICs. The 12v is supplied from the battery, and the 3.3v are provided jointly by the LM29405 5v and MIC29310 3.3v 1A Low Dropout Regulators. -Refer Diag - 2

- Relay Circuit – Controlling the eight relays is a MCP23017 I2C 16 channel extender driving a ULN2003V12 7-Channel Relay and Inductive 12v Load Sink Driver, that turns on / off the OJE-SH-112LM 12v Miniature PCB Relays. -Refer Diag - 5

- On/Off Switch - A simple single pole switch connects or disconnects the 12v battery voltage supply. -Refer Diag - 6

- ADC Battery Volts - The circuit provides two options to acquire the battery voltage:
  1. Using a resistor voltage divider, pass the signal voltage to the built-in ADC within the ESP32. This solution, although straightforward, does not yield consistent voltage values.
  2. An efficient and highly accurate battery voltage value can be obtained by combining an MCP3204 12 Bit Analog to Digital Converter, with a REF191S 2.048v voltage reference and a source voltage divider. Although this circuit option is complex and costly, the result is very beneficial.

  Both options are included within the PCB design.

- Reverse voltage protection – If the User replaces the battery incorrectly. The circuit provides reverse voltage protection to protect the microcontroller and components from damage.

- TFT 3.5 Screen for Testing – The circuit has all the traces to connect the TFT 3.5 Screen to the microcontroller via a 14-pin header. -Refer Diag - 8

- Test Button – The microcontroller is programmed to receive a Test signal via a Tact switch. -Refer Diag - 3

Software

The software features below follow the circuit descriptions above. Therefore, to avoid duplication, I have included the software line reference numbers for each topic to allow the reader to view the software code for their interest.

For detailed Software information regarding the categories below – refer Appendix 15 – Master Unit Software

- Receive soil data from Sensor Unit
- Send Soil data to Phone Cloud System
- Receive water Management data from the Phone system.
- Relay Circuit
- Water Management
- Battery Volts.
- Test software and live display data.

Cloud Control Unit

Enclosure box

- The ESP32 microprocessor that controls the Cloud Control Unit housed within the Master Enclosure.

PCB Board

- Size: included within the Master Unit PCB

Functions

The primary function is to provide a data transmission conduit between the Master Unit and the Cloud Phone App, passing soil sensor readings to the Phone and receiving and sending garden watering information to the Master Unit.

- Receive sensor data from the Sensor Unit.
- Send Sensor data to iPhone Cloud.
- Receive minimum Soil Moisture set level from iPhone.
- Transmit minimum soil moisture to Master Unit.

Circuits

The Cloud Control Unit does not require a typical PCB circuit to operate, as it utilises the ESP32 microcontroller with two TX and RX wired connections and 3.3v power from the Master Unit power supply.

- ESP32 Microcontroller – housed within the Master Unit Enclosure.
- RS232 TX and RX data – Used for future data transmission to/from the Master Unit.
- Power – 3.3v from the Power Supply.

Software

For detailed Software information regarding the categories below – refer Appendix 17 – Cloud Control Unit Software

- Get sensor and battery voltage from Master Unit.
- Connect iPhone cloud and send sensor data.
- Connect to the Master unit and send moisture set data.
- Test software and live display data.

Appendix:

The List of Appendices are as follows:

Appendix 1 - Sensor Unit Software

Soil Sensor System

```
1   //****** Soil_Ground _Modual V03 *******
2   //
3   // ***** Software for the Slave Moduals *******
4   //
5
6   //Library for Moisture Sensor
7   #include "Adafruit_seesaw.h"
8
9   //Library for Ground Temp
10  #include <OneWire.h>
11  #include <DS18B20_INT.h>  // Library for Ground Yemp Probe
12
13  //Library for Screen
14  #include <SPI.h>
15  #include <TFT_eSPI.h>     // ESP32 TFT library
16
17  //Library for ESP-Now
18  #include <esp_now.h>
19  #include <WiFi.h>
20
21  // ESP Now Channel
22  //#define CHANNEL 1
23
24  //Set up Sensor Info
25  Adafruit_seesaw ss;
26  #define ONE_WIRE_BUS  5 // Ground Temp Probe DS18B20 on pin 5
27  OneWire     oneWire(ONE_WIRE_BUS);
28  DS18B20_INT sensor(&oneWire);
29
30  //Set up Screen Info
31  TFT_eSPI tft = TFT_eSPI();      // Invoke custom TFT library
32  uint16_t bg = TFT_BLACK;        // Set foreground and Background Colours
33  uint16_t fg = TFT_WHITE;
34
35  // ******** Set up ESP-Now Info *******
36
37  // Variables for Send data
38  const int Slave_No = 1;  // Slave ESP32 number 1
39  int Air_Temp;
40  int Ground_Temp;
41  float Moisture_Value;
```

```
42      float Battery_Volts;
43
44      // MAC Address of responder/ Master - ESP32 number 1
45      // C4:DD:57:5E:26:54
46      //uint8_t broadcastAddress[] = {0xC4, 0xDD, 0x57, 0xSE, 0x26, 0x54}; // Master MAC Address
47
48      // MAC Address of responder/ Master - ESP32 number 2
49      // C4:DD:57:5E:2C:98
50      //uint8_t broadcastAddress[] = {0xC4, 0xDD, 0x57, 0x5E, 0x2C, 0x98}; // Master MAC Address
51
52      // MAC Address of responder/ Master - ESP32 number 3
53      // C4:DD:57:5E:2A:38
54      uint8_t broadcastAddress[] = {0xC4, 0xDD, 0x57, 0x5E, 0x2A, 0x38}; // Master MAC Address
55
56      // MAC Address of responder/ Master - ESP32 number 4
57      // 84:0D:8E:E6:C2:A0
58      //uint8_t broadcastAddress[] = {0x84, 0x0D, 0x8E, 0xE6, 0xC2, 0xA0}; // Master MAC Address
59
60      // Define a data structure
61      typedef struct struct_message {
62      char a[32];
63      int Send_Slave_No;
64      int Send_Air_Temp;
65      float Send_Moisture_Value;
66      int Send_Ground_Temp;
67      float Send_Battery_Volts;
68      } struct_message;
69
70      // Create a structured object
71      struct_message myData;
72
73      // Peer info
74      esp_now_peer_info_t peerInfo;
75
76      // Callback function called when data is sent
77      void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
78       Serial.print("\r\nLast Packet Send Status:\t");
79       Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery Fail");
80        // Success to Screen
81       // tft.fillRect(5, 200, 420, 30, bg);
82       // tft.setCursor(5, 200);
83       //tft.print(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery Fail");
84
85      }
86
87      void setup() {
```

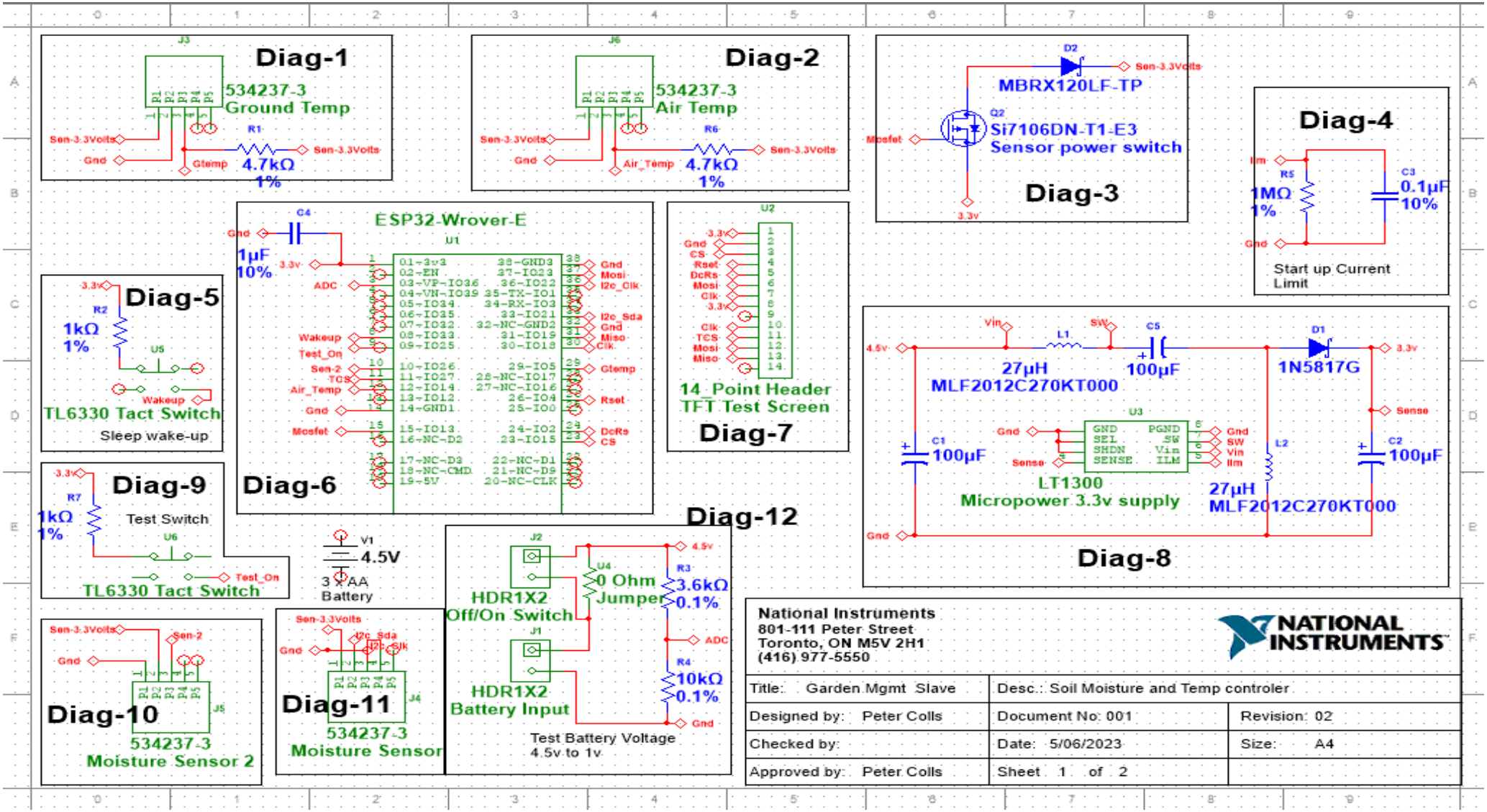
```
88        Serial.begin(115200);
89        Serial.print("DS18B20_INT_LIB_VERSION: ");
90        Serial.println(DS18B20_INT_LIB_VERSION);
91
92        // Set ESP32 as a Wi-Fi Station
93        WiFi.mode(WIFI_STA);
94
95        // Initilize ESP-NOW
96        if (esp_now_init() != ESP_OK) {
97          Serial.println("Error initializing ESP-NOW");
98          return;
99        }
100      // Register the send callback
101       esp_now_register_send_cb(OnDataSent);
102
103       // Register peer (The Master)
104       memcpy(peerInfo.peer_addr, broadcastAddress, 6);
105       peerInfo.channel = 0;
106       peerInfo.encrypt = false;
107
108       // Add peer
109       if (esp_now_add_peer(&peerInfo) != ESP_OK){
110         Serial.println("Failed to add peer");
111         return;
112       }
113
114       //  *** Set up Mosfet to turn on 3.3v circuit ***
115       int Mosfet = 13;  // Mosfet switch pin no.
116       pinMode(Mosfet, OUTPUT);
117       digitalWrite(Mosfet, HIGH); // Turn on Mosfet Switch
118
119
120       // Setup the TFT
121       tft.begin();
122       tft.setRotation(1);
123       tft.setTextColor(fg, bg);
124       tft.fillScreen(bg);
125       tft.setCursor(2, 0);
126       // Set the font colour to be yellow with no background, set to font 7
127       tft.setTextColor(TFT_YELLOW); tft.setTextFont(4);
128
129       if (sensor.begin() == false)
130       {
131         Serial.println("ground Temp not connected!");
132       }
133
```

```
134       // Constant for Dry Sensor
135       const int DryValue = 413;
136       // Constant for Wet Sensor
137       const int WetValue = 1015;
138
139       // Variables for Soil Moisture
140       int SoilMoistureValue;
141       int SoilMoisturePercent;
142
143       Serial.println("seesaw Soil Sensor example!");
144
145       if (!ss.begin(0x36)) {
146         Serial.println("ERROR! seesaw not found");
147         while(1) delay(1);
148       } else {
149         Serial.print("seesaw started! version: ");
150         Serial.println(ss.getVersion(), HEX);
151       }
152     }
153
154     void loop() {
155
156       //Variables for I2C Moisture and Temp Sensor (MS-1, J4)
157       float tempC = ss.getTemp();          // Get the air temp from the I2C sensor
158       uint16_t capread = ss.touchRead(0);  //?????
159       int SoilMoistureValue;
160       int SoilMoisturePercent;
161       const int DryValue = 413;
162       // Constant for Wet Sensor
163      const int WetValue = 1015;
164
165       // **** Battery Volts ****
166       int ADC_Pin = 36;                  // ADC Pin - Battery voltage divided to 1v for ADC
167       float xloop = (99.0 * xloop + analogRead(ADC_Pin)) / 100.0;
168       float yvolt = (xloop * 4095.0 / 9.396)/1000; //4.5v
169
170
171       //Processor Number
172       Serial.print("Procesor No: ");
173       Serial.println(Slave_No);
174
175
176       // ********* Get Soil Moisture and Air Temp From I2C Sensor ( J4)
177       //
178       SoilMoisturePercent = map(capread, DryValue, WetValue, 0, 100);
179
```

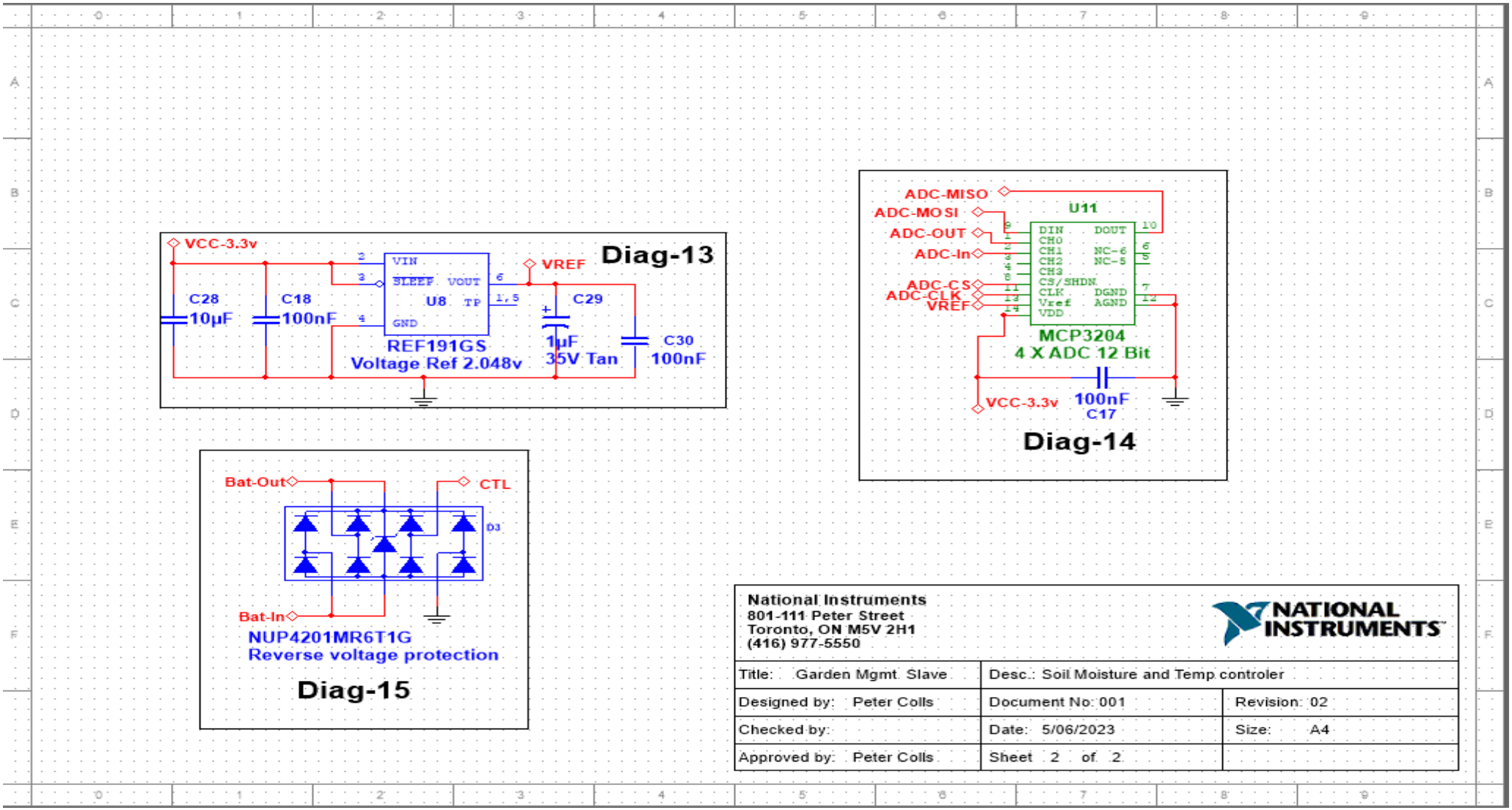
```
180       //Keep Values between 0 and 100%
181       SoilMoisturePercent = constrain (SoilMoisturePercent, 0, 100);
182       Air_Temp = tempC; //Air Temp
183       Serial.print("Air Temperature: "); Serial.print(Air_Temp); Serial.println("*C");
184
185       Moisture_Value = (SoilMoisturePercent);
186       Serial.print("Soil Moisture%: "); Serial.print(Moisture_Value);Serial.println("% ");
187      //
188
189      //Get Ground Temp
190      sensor.requestTemperatures();
191       while (!sensor.isConversionComplete());  // (BLOCKING!!) wait until sensor is ready
192       Serial.print("Ground Temp: ");
193       Ground_Temp = (sensor.getTempC())+3;
194       Serial.println(Ground_Temp); // New line
195
196       // Battery Volts
197       Battery_Volts = yvolt;
198       Serial.print("Battery Volts: ");
199       Serial.println(Battery_Volts);
200
201      // ************ Format structured data for sending to ESP_Now Master *************
202       strcpy(myData.a, "Sending Processor No 1");
203       myData.Send_Slave_No = Slave_No;
204       myData.Send_Air_Temp = Air_Temp;
205       myData.Send_Moisture_Value = 62.3;
206       //myData.Send_Moisture_Value = Moisture_Value;
207       myData.Send_Ground_Temp = 39;
208       //myData.Send_Ground_Temp = Ground_Temp;
209       myData.Send_Battery_Volts = 99.9;
210       //myData.Send_Battery_Volts = Battery_Volts;
211
212       // Send message via ESP-NOW
213       esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &myData, sizeof(myData));
214
215       if (result == ESP_OK) {
216         Serial.println("Sending confirmed");
217         Serial.println(myData.Send_Slave_No);
218         Serial.println(myData.Send_Air_Temp);
219         Serial.println(myData.Send_Moisture_Value);
220         Serial.println(myData.Send_Ground_Temp);
221         Serial.println(myData.Send_Battery_Volts);
222
223       }
224       else {
225          Serial.println("Sending error");
```

```
226         }
227
228         //tft.fillScreen(bg);
229         tft.setCursor(0, 0);
230
231        // Print Slave No
232         tft.fillRect(5, 50, 420, 30, bg);
233         tft.setCursor(5, 50);
234         tft.print("Slave No: ");
235         tft.print(Slave_No);
236
237         // Print Air Temp
238         tft.fillRect(5, 80, 420, 30, bg);
239         tft.setCursor(5, 80);
240         tft.print("Air Temperature:  ");
241         tft.print(Air_Temp);
242
243         // Print Input Moisture
244         tft.fillRect(5, 120, 420, 30, bg);
245         tft.setCursor(5, 120);
246         tft.print("Soil Moisture%: ");
247         tft.print(Moisture_Value);
248         tft.println(" %");
249
250         // Print Ground Temp
251         tft.fillRect(5, 160, 420, 30, bg);
252         tft.setCursor(5, 160);
253         tft.print("Ground Temp: ");
254         tft.println(Ground_Temp);
255
256         // Battery Volts
257         tft.fillRect(5, 180, 420, 30, bg);
258         tft.setCursor(5, 180);
259         tft.print("Battery Volts ");
260         tft.print(Battery_Volts);
261
262
263         delay(3000);
264       }
265       // End Loop
```

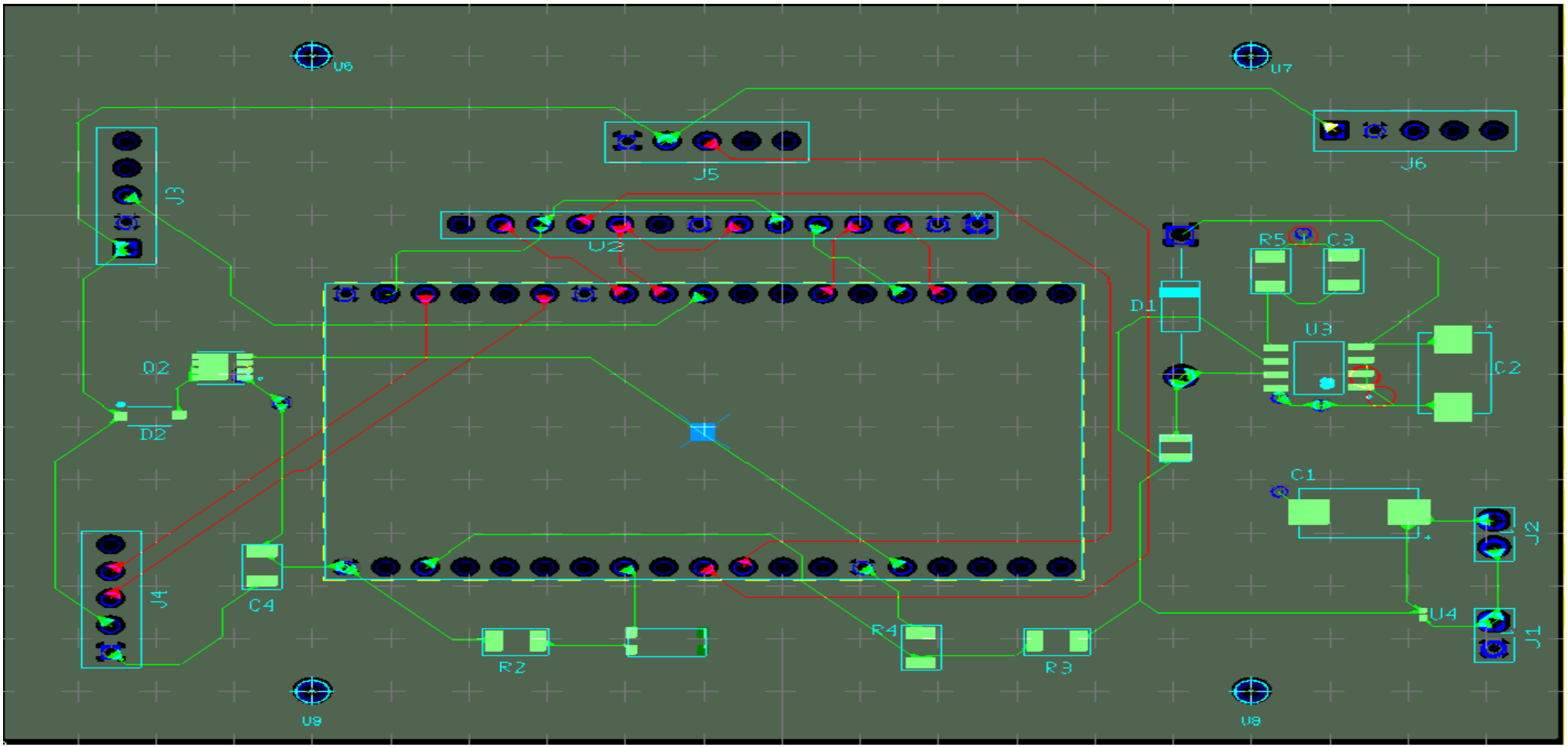
Appendix 2 – Sensor Unit Circuit - 1



Appendix 3 - Sensor Unit Circuit - 2



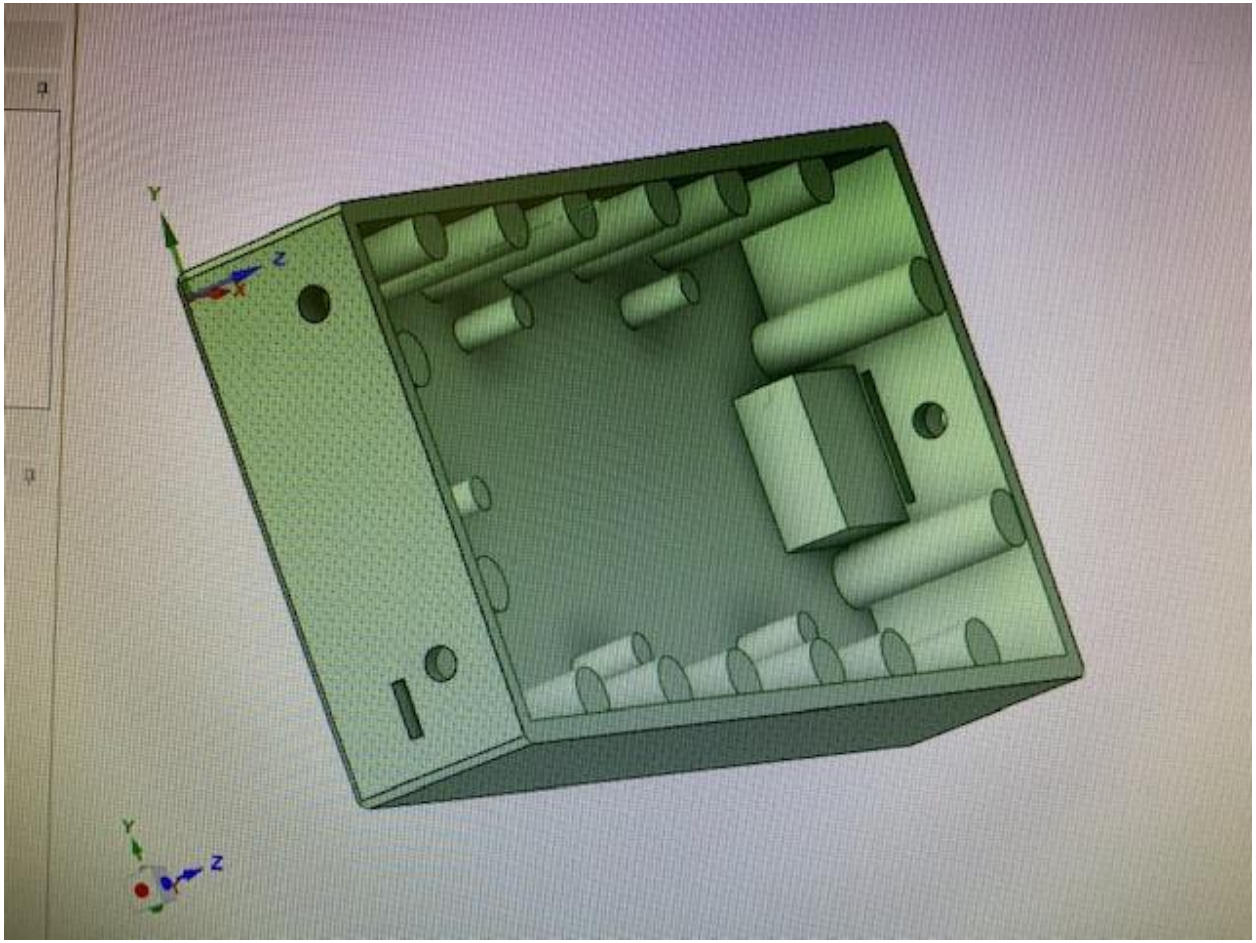
Appendix 4 - Sensor Unit PCB Layout



Appendix 5 – Sensor Unit Bill of Materials (BOM)

Quantity	Description	RefDes	Package	Type
1	VOLTAGE_REFERENCE, REF191GS	U8	Analog Devices\SOIC-N-8(R-8)	
1	CAPACITOR, 10µF	C28	Generic\1206	Capacitor
3	CAPACITOR, 100nF	C17, C18, C30	Generic\1206	Capacitor
1	CAP_ELECTROLIT, 1µF	C29	Generic\1206	TAN 35v
1	ICs, MCP3204	U11	IPC-7351\SOIC-14	
1	PROTECTION_DIODE, NUP4201MR6T1G	D3	ON Semiconductor\TSOP-6-6(CASE 318G-02U)	
1	MicroController, ESP32-Wrover-E	U1	Ultiboard\Rover-E-38-Pins	
1	Def, 14_Point Header	U2	Ultiboard\HEADER1X14	
1	Regulator, LT1300	U3	IPC-7351\SOIC-8	
1	SCHOTTKY_DIODE, 1N5817G	D1	ON Semiconductor\Axial Lead-2(CASE 59-10U)	
1	RESISTOR, 3.6kΩ 0.1%	R3	Generic\1206	Metal Oxide Film
1	RESISTOR, 10kΩ 0.1%	R4	Generic\1206	Metal Oxide Film
3	CAP_ELECTROLIT, 100µF	C1, C2, C5	IPC-SMT-782\7343TAN_CAP	Tantalum
1	RESISTOR, 1MΩ 1%	R5	Generic\1206	
2	HEADERS_TEST, HDR1X2	J1, J2	Generic\HDR1X2	
1	Thermistor, 0 Ohm	U4	Bourns\EIA 0201(CR)	
4	HEADERS_TEST, 534237-3	J3, J4, J5, J6	TE Connectivity\534237-3	
2	RESISTOR, 4.7kΩ 1%	R1, R6	Generic\1206	
1	Mosfet, Si7106DN-T1-E3	Q2	Vishay\PowerPAK-8(1212-8 Single)	
1	Rectifier, MBRX120LF-TP	D2	IPC-7351\SOD-123	
1	CAPACITOR, 0.1µF 10%	C3	Generic\1206	Ceramic
1	CAPACITOR, 1µF 10%	C4	Generic\1206	Ceramic
2	RESISTOR, 1kΩ 1%	R2, R7	Generic\1206	
2	Switch, TL6330 Tact Switch	U5, U6	Ultiboard\SPSD-SMD	
2	INDUCTOR, 27µH	L1, L2	Generic\1206	Coil

Appendix 6 - Sensor Unit Enclosure Housing Box - 1



Appendix 7 - Sensor Unit Enclosure Housing Box - 2

Appendix 8 – Sensor Unit Battery Long Life Calc

Your device will probably run for 8363 hours or around 348 days and 11 hours

Its estimated, average power consumption per hour 0.9 mAh

Software

duration of code execution		sleep time	
2.5	sec	240	sec

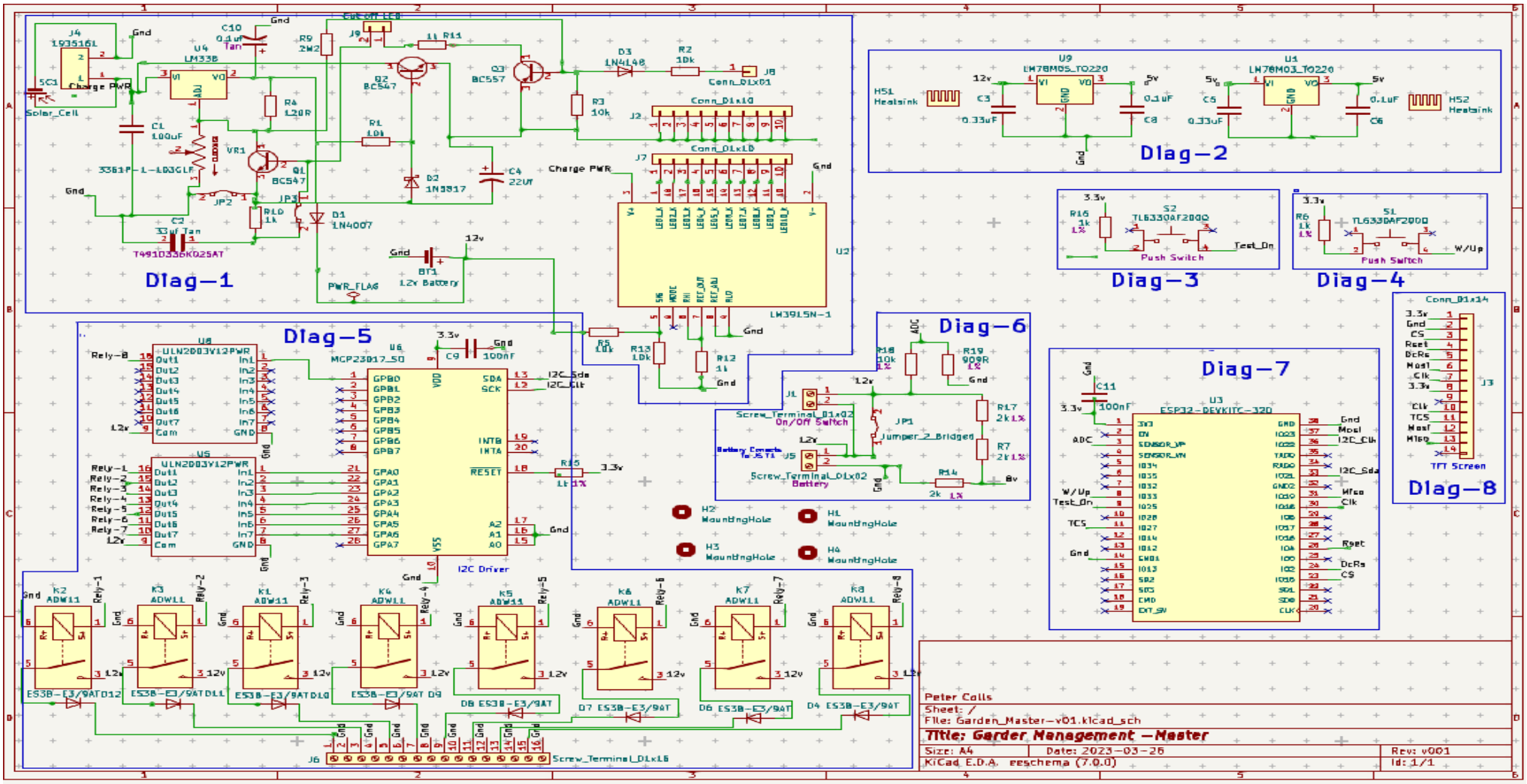
Hardware

consumption during code execution		consumption in sleep mode*	
80	mA	80	µA

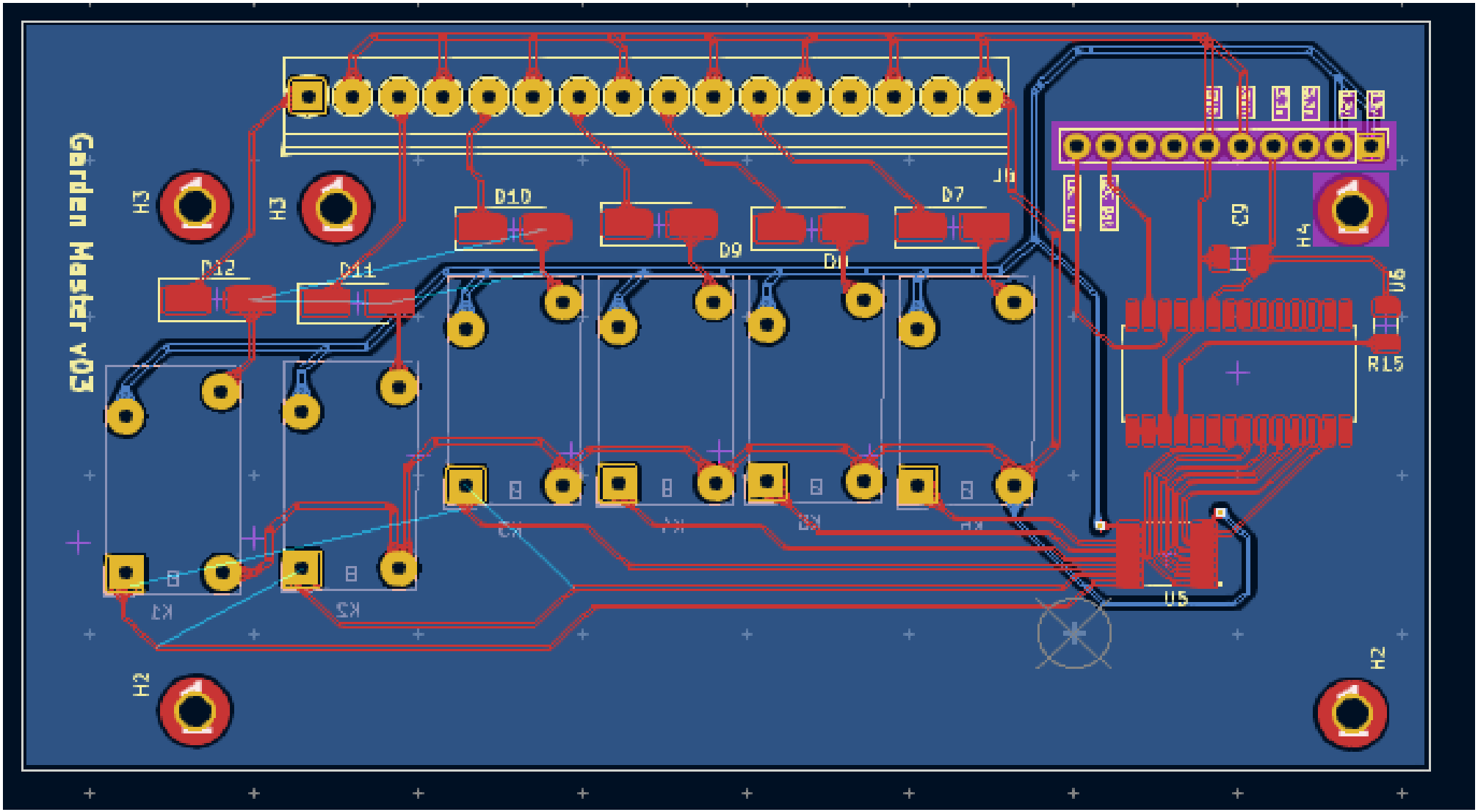
Battery

power of battery		discharge safety	
8400	mAh	10	%

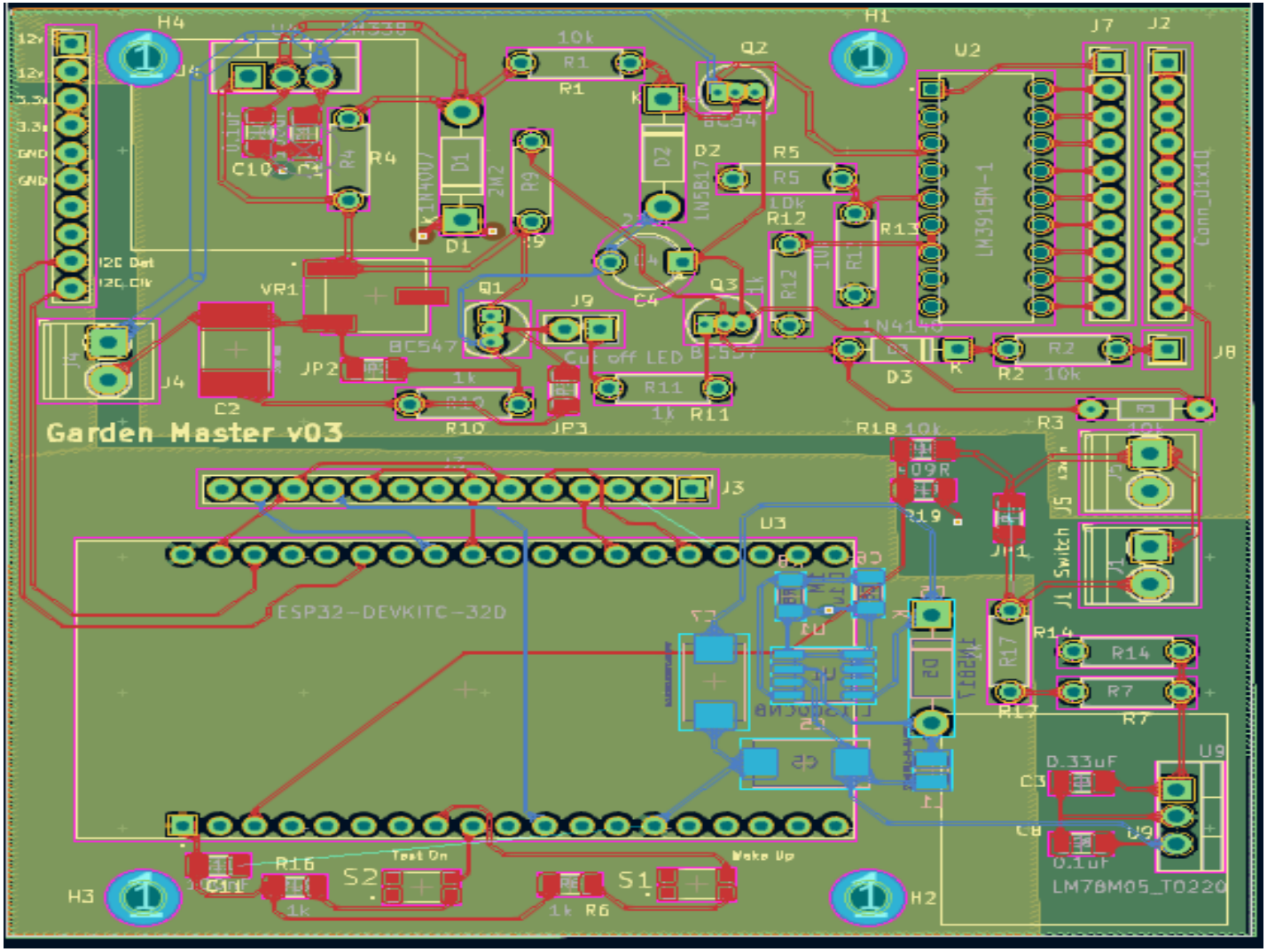
Appendix 9 - Master Unit Circuit



Appendix 10 – Master Unit Bottom PCB



Appendix 11 - Master Unit Top PCB



Appendix 12 – Master Unit Top Bill of Materials (BOM)

Master Unit Top BOM

Loc	References	Value	Footprint	Quantity
1	C6, C8, C10, C11	100nF	C_1206_3216Metric	4
2	C1	100uF	C_1206_3216Metric_Pad1.33x1.80mm_HandSolder	1
3	C2	33uf Tan	CAPPC7343X430N	1
4	C3	0.33uF	C_1206_3216Metric	1
5	C4	22Uf	CP_Radial_Tantal_D5.0mm_P5.00mm	1
6	C5	TPSD107M010R0100	CP_EIA-7343-20_Kemet-V_Pad2.25x2.55mm_HandSolder	1
7	C7	TPSD107M010R0100	CAPPM7343X310N	1
8	R1, R2, R5, R13	10k	R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal	4
9	R7, R14, R17	2k	R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal	3
10	R10, R11, R12	1k	R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal	3
11	R6, R16	1k	R_1206_3216Metric	2
12	R3	10k	R_Axial_DIN0204_L3.6mm_D1.6mm_P7.62mm_Horizontal	1
13	R4	120R	R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal	1
14	R8	1M	R_1206_3216Metric	1
15	R9	2M2	R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal	1
16	R18	10k	R_1206_3216Metric	1
17	R19	909R	R_1206_3216Metric	1
18	L1	1269AS-H-100N=P2	INDC2520X100N	1
19	D2, D5	1N5817	D_DO-41_SOD81_P10.16mm_Horizontal	2
20	D1	1N4007	D_DO-41_SOD81_P10.16mm_Horizontal	1
21	D3	1N4148	D_DO-35_SOD27_P7.62mm_Horizontal	1
22	U1	LT1300CN8	SOIC-8_3.9x4.9mm_P1.27mm	1
23	U2	LM3915N-1	DIP795W25P254L2337H533Q18B	1
24	U3	ESP32-DEVKITC-32D	MODULE_ESP32-DEVKITC-32D	1
25	U4	LM338	TO-220-3_Vertical	1
26	U9	LM78M05_TO220	TO-220-3_Vertical	1
27	H1, H2, H3, H4	Mounting Hole	MountingHole_2.5mm_Pad_TopBottom	4
28	JP1, JP2, JP3	Jumper_2_Bridged	R_1206_3216Metric	3
29	Q1, Q2	BC547	TO-92_Inline	2
30	S1, S2	TL6330AF200Q	SW_TL6330AF200Q	2

31	Q3	BC557	TO-92_Inline	1
32	VR1	3361P-1-103GLF	TRIM_3361P-1-103GLF	1
33	J2, J2, J7	Conn_01x10	PinHeader_1x10_P2.54mm_Vertical	3
34	J1, J5	Screw_Terminal_01x02	TerminalBlock_Phoenix_PT-1,5-2-3.5-H_1x02_P3.50mm_Horizontal	2
35	J3	Conn_01x14	PinHeader_1x14_P2.54mm_Vertical	1
36	J4	1935161	TerminalBlock_Phoenix_PT-1,5-2-3.5-H_1x02_P3.50mm_Horizontal	1
37	J8	Conn_01x01	PinHeader_1x01_P2.54mm_Vertical	1
38	J9	Cut off LED	PinHeader_1x02_P2.54mm_Vertical	1

Appendix 13 – Master Unit Bottom Bill of Materials (BOM)

Master Unit Bottom BOM

LOC	References	Value	Footprint	Quantity
1	C9	100nF	C_1206_3216Metric	1
2	R15	1k	R_1206_3216Metric	1
3	D7, D8, D9, D10, D11, D12	ES3B-E3/9AT	D_SMA_Hand soldering	6
4	U5	ULN2003V12PWR	SOP65P640X120-16N	1
5	U6	MCP23017_SO	SOIC-28W_7.5x17.9mm_P1.27mm	1
6	K1, K2, K3, K4, K5, K6	ADW11	Relay_18.1 x 10.10 PWC _THT	6
7	H2, H2, H3, H3, H4	Mounting Hole	MountingHole_2.5mm_Pad_TopBottom	5
9	J2	Conn_01x10	PinHeader_1x10_P2.54mm_Vertical	1
10	J6	Screw_Terminal_01x16	TerminalBlock_Phoenix_PT-1,5-16-3.5-H_1x16_P3.50mm_Horizontal	1

Appendix 14 – Master Unit Enclosure Layout

The sealed enclosure is waterproof and provides two mounting flanges. the lid is made of a clear Polycarbonate material allowing the User to observe the ten LED lights that represent the battery charge state.

Appendix 15 – Master Unit Software

Receiver Master - Multiple Slaves-v03

```
1   //Receiver Master - Multiple Slaves-v03
2   /*
3     Master Code:
4     - Receives sensor data from all slaves
5     - Sends sensor data to ESP32-2 (Cloud Processor) to update the phone app
6     - Receives moisture set data from a phone app and stored in permanent memory
7     - Get and send battery volt info
8     - Test slave moisture values with saved set data and turn on the water if required
9     - Manage and control Test and data print logic
10
11    - Note: Due to the size and complexity of the Master code, a separate ESP processor is used
12      to interface with Arduino Cloud - refer above
13
14    ***** Technical help from:  ******
15    DroneBot Workshop 2022 https://dronebotworkshop.com
16    Rui Santos - Random Nerd Tutorials - http://RandomNerdTutorials.com
17  */
18
19  // Include required libraries
20  #include <WiFi.h> // WiFi Library
21  #include <esp_now.h>  //ESP-Now Comms Library
22
23  //
24  //Library for Screen
25  #include <SPI.h>
26  #include <TFT_eSPI.h>     // ESP32 TFT library
27
28  //Set up Screen Info
29  TFT_eSPI tft = TFT_eSPI();      // Invoke custom TFT library
30  uint16_t bg = TFT_BLACK;        // Set foreground and Background Colours
31  uint16_t fg = TFT_WHITE;
32
33
34  // Set working Variables
35   float Send_Moisture_Value;
36   int Send_Ground_Temp;
37   int Received_Data = 0;
38
39   // Variable to store if sending data was successful
40  String success;
41
```

```
42
43      // MAC Address of Cloud Processor - ESP32 number 2
44      // C4:DD:57:5E:2C:98
45      uint8_t broadcastAddress[] = {0xC4, 0xDD, 0x57, 0x5E, 0x2C, 0x98}; // Master MAC Address
46
47       // Define Cloud data Send structure
48      typedef struct struct_send_message {
49      char a[32];
50      int Send_Slave_No;
51      int Send_Air_Temp;
52      float Send_Moisture_Value;
53      int Send_Ground_Temp;
54      float Send_Battery_Volts;
55      float Send_Master_Volts;
56      } struct_send_message;
57
58      // Create structured data sent object for Cloud
59      struct_send_message mySendData;
60
61       //Serial.println(" My Send Data Set");
62
63      // Define Slave Receive data structure
64      typedef struct struct_receive_message {
65      char a[32];
66      int Receive_Slave_No;
67      int Receive_Air_Temp;
68      float Receive_Moisture_Value;
69      int Receive_Ground_Temp;
70      float Receive_Battery_Volts;
71      } struct_receive_message;
72
73      // Create structured recevied data object from Slaves
74      struct_receive_message myReceiveData;
75
76      // Set up Peer Info Holder for Registration
77      esp_now_peer_info_t peerInfo;
78
79      // Callback when data is sent
80      void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
81        Serial.print("\r\nLast Packet Send Status:\t");
82        Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery Fail");
83        if (status ==0){
84          success = "Delivery Success :)";
85        }
86        else{
87          success = "Delivery Fail :(";
```

```
 88        }
 89       }
 90
 91
 92       // Callback function
 93       void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len)
 94       {
 95         // Get incoming data
 96         memcpy(&myReceiveData, incomingData, sizeof(myReceiveData));
 97
 98         // Received dats from Slave - Now send the Info to Cloud
 99
100         Received_Data = 1; // Send to Cloud Flag
101
102         // Print to Serial Monitor
103         Serial.print("Len "); Serial.println(len);
104         //Serial.println(struct_message myData);
105         Serial.println(myReceiveData.a);
106
107         Serial.print("Processor No ");
108         Serial.println(myReceiveData.Receive_Slave_No);
109
110         Serial.print("Air Temp ");
111         Serial.println(myReceiveData.Receive_Air_Temp);
112
113         Serial.print("Moisture Value ");
114         Serial.println(myReceiveData.Receive_Moisture_Value);
115
116         Serial.print("Ground Temp ");
117         Serial.println(myReceiveData.Receive_Ground_Temp);
118
119         Serial.print("Battery Volts ");
120         Serial.println(myReceiveData.Receive_Battery_Volts);
121
122          //tft.fillScreen(bg);
123         tft.setCursor(0, 0);
124
125           // Print processor number
126         tft.fillRect(5, 50, 420, 30, bg);
127         tft.setCursor(5, 50);
128         tft.print("Processor number ");
129         tft.print(myReceiveData.Receive_Slave_No);
130
131          // Print Air Temp
132         tft.fillRect(5, 80, 420, 30, bg);
133         tft.setCursor(5, 80);
```

```
134       tft.print("Air Temperature:  ");
135       tft.print(myReceiveData.Receive_Air_Temp);
136
137       // Print Input Moisture
138       tft.fillRect(5, 120, 420, 30, bg);
139       tft.setCursor(5, 120);
140       tft.print("Soil Moisture%: ");
141       tft.print(myReceiveData.Receive_Moisture_Value);
142       tft.println(" %");
143
144       // Print Ground Temp
145       tft.fillRect(5, 160, 420, 30, bg);
146       tft.setCursor(5, 160);
147       tft.print("Ground Temp: ");
148       tft.println(myReceiveData.Receive_Ground_Temp);
149
150       // Battery Volts
151       tft.fillRect(5, 180, 420, 30, bg);
152       tft.setCursor(5, 180);
153       tft.print("Battery Volts ");
154       tft.print(myReceiveData.Receive_Battery_Volts);
155
156        Serial.println(" Received Info");
157
158       delay(1000);
159
160     }
161
162     void setup() {
163
164       // Set up Serial Monitor
165       Serial.begin(115200);
166
167
168       // Setup the TFT
169       tft.begin();
170       tft.setRotation(1);
171       tft.setTextColor(fg, bg);
172       tft.fillScreen(bg);
173       tft.setCursor(2, 0);
174       // Set the font colour to be yellow with no background, set to font 7
175       tft.setTextColor(TFT_YELLOW); tft.setTextFont(4);
176
177
178       // Start ESP32 in Station mode
179       WiFi.mode(WIFI_STA);
```

```
180
181       // Initalize ESP-NOW
182       if (esp_now_init() != 0) {
183         Serial.println("Error initializing ESP-NOW");
184         return;
185       }
186
187       // Register the send callback
188       esp_now_register_send_cb(OnDataSent);
189
190       // Register peer (The Cloud Master)
191       memcpy(peerInfo.peer_addr, broadcastAddress, 6);
192       peerInfo.channel = 0;
193       peerInfo.encrypt = false;
194
195       // Add peer
196          if (esp_now_add_peer(&peerInfo) != ESP_OK){
197         Serial.println("Failed to add peer");
198         return;
199          }
200
201           Serial.println(" Set up Peer");
202
203        // Register Receive callback function
204       esp_now_register_recv_cb(OnDataRecv);
205
206       }
207
208     void loop() {
209
210
211       // **** Battery Volts ****
212       int ADC_Pin = 36;                 // ADC Pin - Battery voltage divided to 1v for ADC
213       float xloop = (99.0 * xloop + analogRead(ADC_Pin)) / 100.0;
214       float yvolt = ((xloop * 4095.0 / 354.5)/10.0); //13.27v
215
216       // Test for New Valid Data to Send
217       if(Received_Data>0){
218
219         // Set values to send
220           strcpy(mySendData.a, "Sending New Slave Info");
221           mySendData.Send_Slave_No = myReceiveData.Receive_Slave_No;
222           mySendData.Send_Air_Temp = myReceiveData.Receive_Air_Temp;
223           mySendData.Send_Moisture_Value = myReceiveData.Receive_Air_Temp;
224           mySendData.Send_Ground_Temp = myReceiveData.Receive_Ground_Temp;
225           mySendData.Send_Battery_Volts = myReceiveData.Receive_Battery_Volts;
```

```
226           mySendData.Send_Master_Volts = yvolt;
227
228         // Send message via ESP-NOW
            esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &mySendData,
229       sizeof(mySendData));
230
231           if (result == ESP_OK) {
232           Serial.println("Sent with success");
233           }
234           else {
235           Serial.println("Error sending the data");
236           }
237       }
238
239       delay(2000);
240
241        }
```

Appendix 16 – Master Unit Calculate ADC Voltage

```
1
2        //
3         // Dfine ADC data
4         //
5         float voltage0 = 0;              // Set inital voltage
6         float voltage1 = 0;
7         float voltage2 = 0;
8         float voltage3 = 0;
9         int Channel = 0;                 // define channel to zero
10        const int adcChipSelectPin = 26;  // set pin 26 as the chip select for the ADC
11
12        //
13       //Init Code for DAC
14       //
15       // Set Constants
16        const int dacChipSelectPin = 25;      // set pin 25 as the chip select for the DAC:
17
18      // Used for reading DT signal- VA = Volts CA = Current
19      const int PinVB = 32;
20      const int PinCB = 39;
21
22      //
23          // Set up values for ADC
24          //
25          // set the ChipSelectPins high initially:
26          pinMode (adcChipSelectPin, OUTPUT);
27          digitalWrite(adcChipSelectPin, HIGH);
28
29
30        SPI.begin();
31        SPI.setBitOrder(MSBFIRST);         // Not strictly needed but just to be sure.
32        SPI.setDataMode(SPI_MODE0);        // Not strictly needed but just to be sure.
33        //Set SPI clock divider to 16, therfore a 1 MhZ signal
34        //                                 due to the maximum frequency of the ADC.
35        SPI.setClockDivider(SPI_CLOCK_DIV16);
36
      // Function to set the DAC, Accepts the Value to be sent and the cannel of the DAC to be
37    used.
38    void setDac(int value, int channel)
39    {
```

```
  byte dacRegister = 0b00110000;                    // Sets default DAC registers B00110000,
40 1st bit choses DAC, A=0 B=1, 2nd Bit bypasses input Buffer, 3rd bit sets output gain to 1x,
   4th bit controls active low shutdown. LSB are insignifigant here.

41 int dacSecondaryByteMask = 0b0000000011111111;       // Isolates the last 8 bits of the 12
   bit value, B0000000011111111.

   byte dacPrimaryByte = (value >> 8) | dacRegister;     //Value is a maximum 12 Bit value, it
42 is shifted to the right by 8 bytes to get the first 4 MSB out of the value for entry into th
   Primary Byte, then ORed with the dacRegister

43 byte dacSecondaryByte = value & dacSecondaryByteMask; // compares the 12 bit value to
   isolate the 8 LSB and reduce it to a single byte.

44  // Sets the MSB in the primaryByte to determine the DAC to be set, DAC A=0, DAC B=1
45  switch (channel)
46  {
47   case 0:
48     dacPrimaryByte &= ~(1 << 7);
49   break;
50   case 1:
51     dacPrimaryByte |= (1 << 7);
52  }
53  noInterrupts(); // disable interupts to prepare to send data to the DAC
54  digitalWrite(dacChipSelectPin,LOW); // take the Chip Select pin low to select the DAC:
55  SPI.transfer(dacPrimaryByte); //  send in the Primary Byte:
56  SPI.transfer(dacSecondaryByte);// send in the Secondary Byte
57  digitalWrite(dacChipSelectPin,HIGH);// take the Chip Select pin high to de-select the DAC:
58  //Serial.println("send dac info");
59  interrupts(); // Enable interupts
60 }
```

Appendix 17 – Cloud Control Unit Software

```
1
2       //
3       // Ardunio_Cloud_v03 (iPhone IOT Display System)
4       //
5       #include <WiFi.h> // WiFi Lib
6       #include <esp_now.h>  //ESP-Now Comms Library
7
8       //Library for Screen
9       #include <SPI.h>
10      #include <TFT_eSPI.h>     // ESP32 TFT library
11
12
13
14
15       // Include required libraries
16
17       // Sketch generated by the Arduino IoT Cloud Thing "Untitled"
18       //https://create.arduino.cc/cloud/things/89368119-2d69-4799-ae86-536e1f9432e5
19
20       //Arduino IoT Cloud Variables description
21
         //The following variables are automatically generated and updated when changes are made to the
22      Thing
23
24       float cloud_Master_Volts;
25       float cloud_Moisture_1;
26       float cloud_Moisture_10;
27       float cloud_Moisture_2;
28       float cloud_Moisture_3;
29       float cloud_Moisture_4;
30       float cloud_Moisture_5;
31       float cloud_Moisture_6;
32       float cloud_Moisture_7;
33       float cloud_Moisture_8;
34       float cloud_Moisture_9;
35       float cloud_Set_Moisture_1;
36       float cloud_Set_Moisture_10;
37       float cloud_Set_Moisture_2;
38       float cloud_Set_Moisture_3;
39       float cloud_Set_Moisture_4;
40       float cloud_Set_Moisture_5;
```

```
41      float cloud_Set_Moisture_6;
42      float cloud_Set_Moisture_7;
43      float cloud_Set_Moisture_8;
44      float cloud_Set_Moisture_9;
45      float cloud_Slave_Volts_1;
46      float cloud_Slave_Volts_10;
47      float cloud_Slave_Volts_2;
48      float cloud_Slave_Volts_3;
49      float cloud_Slave_Volts_4;
50      float cloud_Slave_Volts_5;
51      float cloud_Slave_Volts_6;
52      float cloud_Slave_Volts_7;
53      float cloud_Slave_Volts_8;
54      float cloud_Slave_Volts_9;
55      int cloud_Air_Temp_1;
56      int cloud_Air_Temp_10;
57      int cloud_Air_Temp_2;
58      int cloud_Air_Temp_3;
59      int cloud_Air_Temp_4;
60      int cloud_Air_Temp_5;
61      int cloud_Air_Temp_6;
62      int cloud_Air_Temp_7;
63      int cloud_Air_Temp_8;
64      int cloud_Air_Temp_9;
65      int cloud_Ground_Temp_1;
66      int cloud_Ground_Temp_10;
67      int cloud_Ground_Temp_2;
68      int cloud_Ground_Temp_3;
69      int cloud_Ground_Temp_4;
70      int cloud_Ground_Temp_5;
71      int cloud_Ground_Temp_6;
72      int cloud_Ground_Temp_7;
73      int cloud_Ground_Temp_8;
74      int cloud_Ground_Temp_9;
75      int cloud_Slave_No_1;
76      int cloud_Slave_No_10;
77      int cloud_Slave_No_2;
78      int cloud_Slave_No_3;
79      int cloud_Slave_No_4;
80      int cloud_Slave_No_5;
81      int cloud_Slave_No_6;
82      int cloud_Slave_No_7;
83      int cloud_Slave_No_8;
84      int cloud_Slave_No_9;
85
86      //Variables which are marked as READ/WRITE in the Cloud Thing will also have functions
```

```
87        //which are called when their values are changed from the Dashboard.
88        //These functions are generated with the Thing and added at the end of this sketch.
89
90
91      //#include "thingProperties.h"
92
93      int Slave_No;
94      int Received_Message = 0;
95
96      //Set up Screen Info
97      TFT_eSPI tft = TFT_eSPI();      // Invoke custom TFT library
98      uint16_t bg = TFT_BLACK;        // Set foreground and Background Colours
99      uint16_t fg = TFT_WHITE;
100
101
102       ///
103       // Define data structure
104     typedef struct struct_message {
105     char a[32];
106     int Receive_Slave_No;
107     int Receive_Air_Temp;
108     float Receive_Moisture_Value;
109     int Receive_Ground_Temp;
110     float Receive_Battery_Volts;
111     float Receive_Master_Volts;
112     } struct_message;
113
114     // Create structured data object
115     struct_message myData;
116
117     // Callback function
118     void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len)
119
120     {
121       // Get incoming data
122       memcpy(&myData, incomingData, sizeof(myData));
123
124       //tft.fillScreen(bg);
125       tft.setCursor(0, 0);
126
127      // Print Slave No
128       tft.fillRect(5, 50, 420, 30, bg);
129       tft.setCursor(5, 50);
130       tft.print("Incomming Data: ");
131       //tft.print(Slave_No);
132
```

```
133
134       Slave_No = myData.Receive_Slave_No;
135
136       switch (Slave_No) {
137
138       case 1:
139         //Garden Sensor equals 1
140         cloud_Slave_No_1 = myData.Receive_Slave_No;
141         cloud_Air_Temp_1 = myData.Receive_Air_Temp;
142         cloud_Moisture_1 = myData.Receive_Moisture_Value;
143         cloud_Ground_Temp_1 = myData.Receive_Ground_Temp;
144         cloud_Slave_Volts_1 = myData.Receive_Battery_Volts;
145         cloud_Master_Volts = myData.Receive_Master_Volts;
146         Received_Message = 1; // set received message true
147         break;
148
149       case 2:
150         //Garden Sensor equals 2
151         cloud_Slave_No_2 = myData.Receive_Slave_No;
152         cloud_Air_Temp_2 = myData.Receive_Air_Temp;
153         cloud_Moisture_2 = myData.Receive_Moisture_Value;
154         cloud_Ground_Temp_2 = myData.Receive_Ground_Temp;
155         cloud_Slave_Volts_2 = myData.Receive_Battery_Volts;
156         cloud_Master_Volts = myData.Receive_Master_Volts;
157         Received_Message = 1; // set received message true
158         break;
159
160         case 3:
161         //Garden Sensor equals 3
162         cloud_Slave_No_3 = myData.Receive_Slave_No;
163         cloud_Air_Temp_3 = myData.Receive_Air_Temp;
164         cloud_Moisture_3 = myData.Receive_Moisture_Value;
165         cloud_Ground_Temp_3 = myData.Receive_Ground_Temp;
166         cloud_Slave_Volts_3 = myData.Receive_Battery_Volts;
167         cloud_Master_Volts = myData.Receive_Master_Volts;
168         Received_Message = 1; // set received message true
169         break;
170
171         case 4:
172         //Garden Sensor equals 4
173         cloud_Slave_No_4 = myData.Receive_Slave_No;
174         cloud_Air_Temp_4 = myData.Receive_Air_Temp;
175         cloud_Moisture_4 = myData.Receive_Moisture_Value;
176         cloud_Ground_Temp_4 = myData.Receive_Ground_Temp;
177         cloud_Slave_Volts_4 = myData.Receive_Battery_Volts;
178         cloud_Master_Volts = myData.Receive_Master_Volts;
```

```
179         Received_Message = 1; // set received message true
180         break;
181
182         case 5:
183         //Garden Sensor equals 5
184         cloud_Slave_No_5 = myData.Receive_Slave_No;
185         cloud_Air_Temp_5 = myData.Receive_Air_Temp;
186         cloud_Moisture_5 = myData.Receive_Moisture_Value;
187         cloud_Ground_Temp_5 = myData.Receive_Ground_Temp;
188         cloud_Slave_Volts_5 = myData.Receive_Battery_Volts;
189         cloud_Master_Volts = myData.Receive_Master_Volts;
190         Received_Message = 1; // set received message true
191         break;
192
193         case 6:
194         //Garden Sensor equals 6
195         cloud_Slave_No_6 = myData.Receive_Slave_No;
196         cloud_Air_Temp_6 = myData.Receive_Air_Temp;
197         cloud_Moisture_6 = myData.Receive_Moisture_Value;
198         cloud_Ground_Temp_6 = myData.Receive_Ground_Temp;
199         cloud_Slave_Volts_6 = myData.Receive_Battery_Volts;
200         cloud_Master_Volts = myData.Receive_Master_Volts;
201         Received_Message = 1; // set received message true
202         break;
203
204         case 7:
205         //Garden Sensor equals 7
206         cloud_Slave_No_7 = myData.Receive_Slave_No;
207         cloud_Air_Temp_7 = myData.Receive_Air_Temp;
208         cloud_Moisture_7 = myData.Receive_Moisture_Value;
209         cloud_Ground_Temp_7 = myData.Receive_Ground_Temp;
210         cloud_Slave_Volts_7 = myData.Receive_Battery_Volts;
211         cloud_Master_Volts = myData.Receive_Master_Volts;
212         Received_Message = 1; // set received message true
213         break;
214
215         case 8:
216         //Garden Sensor equals 8
217         cloud_Slave_No_8 = myData.Receive_Slave_No;
218         cloud_Air_Temp_8 = myData.Receive_Air_Temp;
219         cloud_Moisture_8 = myData.Receive_Moisture_Value;
220         cloud_Ground_Temp_8 = myData.Receive_Ground_Temp;
221         cloud_Slave_Volts_8 = myData.Receive_Battery_Volts;
222         cloud_Master_Volts = myData.Receive_Master_Volts;
223         Received_Message = 1; // set received message true
224         break;
```

```
225
226         case 9:
227         //Garden Sensor equals 9
228         cloud_Slave_No_9 = myData.Receive_Slave_No;
229         cloud_Air_Temp_9 = myData.Receive_Air_Temp;
230         cloud_Moisture_9 = myData.Receive_Moisture_Value;
231         cloud_Ground_Temp_9 = myData.Receive_Ground_Temp;
232         cloud_Slave_Volts_9 = myData.Receive_Battery_Volts;
233         cloud_Master_Volts = myData.Receive_Master_Volts;
234         Received_Message = 1; // set received message true
235         break;
236
237         case 10:
238         //Garden Sensor equals 10
239         cloud_Slave_No_10 = myData.Receive_Slave_No;
240         cloud_Air_Temp_10 = myData.Receive_Air_Temp;
241         cloud_Moisture_10 = myData.Receive_Moisture_Value;
242         cloud_Ground_Temp_10 = myData.Receive_Ground_Temp;
243         cloud_Slave_Volts_10 = myData.Receive_Battery_Volts;
244         cloud_Master_Volts = myData.Receive_Master_Volts;
245         Received_Message = 1; // set received message true
246         break;
247
248             }
249
250
251     }
252
253       void setup() {
254       ////
255       // Initialize serial and wait for port to open:
256       Serial.begin(9600);
257       // This delay gives the chance to wait for a Serial Monitor without blocking if none is found
258
259       // Setup the TFT
260       tft.begin();
261       tft.setRotation(1);
262       tft.setTextColor(fg, bg);
263       tft.fillScreen(bg);
264       tft.setCursor(2, 0);
265       // Set the font colour to be yellow with no background, set to font 7
266       tft.setTextColor(TFT_YELLOW); tft.setTextFont(4);
267
268
269       // Print Slave No
270       tft.fillRect(5, 50, 420, 30, bg);
```

```
271       tft.setCursor(5, 50);
272       tft.print("Set Up Data: ");
273       //tft.print(Slave_No);
274
275
276       delay(1500);
277
278       // Defined in thingProperties.h
279       //initProperties();
280
281       // Connect to Arduino IoT Cloud
282       //ArduinoCloud.begin(ArduinoIoTPreferredConnection);
283
284       /*
285          The following function allows you to obtain more information
286          related to the state of network and IoT Cloud connection and errors
287          the higher number the more granular information you'll get.
288          The default is 0 (only errors).
289          Maximum is 4
290       */
291       //setDebugMessageLevel(2);
292       // ArduinoCloud.printDebugInfo();
293      }
294
295      void loop() {
296
297       //if(Received_Message >0) {
298       //ArduinoCloud.update();
299       // Received_Message = 0;
300       //}
301       // Your code here
302
303
304      }
```