# Smart Garden

team grEEn
Bridget Harrington, Kara Vitale, & David Wieczorek

# 1. Introduction

According to several recent psychological studies, potted plants make you feel better and, in particular, help lower blood pressure and raise job satisfaction. Many plant owners face the problem of how to care for their plants when they are too busy to constantly monitor them. Monitoring these plants require a specific detection of moisture, temperature, and light. Different plants require different thresholds of these three key plant health factors. Most plants need to be watered consistently to ensure they are not being over or under watered. Also, the plant cannot be exposed to a climate that is too hot or cold. Plants require light in order for photosynthesis to occur; however, monitoring each of these characteristics can prove to be a difficult task.

Our proposed solution provides an all-in-one device that will have three different sensors to monitor the moisture, temperature, and light levels of different types of plants. The device will be in a water-proof clear container with the exception of the moisture sensor so that all of the electronic parts will not be damaged when the plant is watered. The clear container allows the light sensor to accurately take readings of the plant's current light threshold. The moisture sensor will stick right into the soil of the potted plant. The device will also send plant owners updates on their plants using Wi-Fi through a mobile application on their phones. MQTT Protocol facilitates communication between the Smart Garden and mobile application through the use of the ESP8266 12 (ESP).

The ESP is the hub of the device. It facilitates the signals that are gathered from the ADC, as well as push that information for the user to see. The ESP gathers the information through I2C and sends the data using MQTT protocol. MQTT is the bridge between the device and the user. Using a publish and subscribe method, information is exchanged from the ESP to the mobile application.

The sensor system is designed to monitor the three critical factors of a plant. The sensor system consists of a temperature sensor, moisture sensor, and light sensor. The temperature sensor consisting of a thermistor, 10K resistor, and 1K resistor measured resistivity inversely to

temperature allowing for the device to accurately monitor temperature. The moisture sensor used was a SparkFun moisture sensor that measured resistivity between two metal strips at a set distance. This resistivity correlated to how much water touched the moisture sensor. This moisture sensor provided an accurate way to monitor moisture of a plant. The light sensor system consisted of a photocell and 10K resistor. This sensor system also relied on resistivity to perform accurate readouts of light levels. As light increases, the photocell has a decreasing resistance. This light sensor system accurately monitored the light levels of the plant. These sensor values were analog signal inputs to the ADS1015 which outputs digital signals to the ESP8366 to read these sensor values.

The mobile application is an integral part of the Smart Garden because it will receive the most interaction with the user. The mobile app provides a means for the user to interpret what the hardware is thinking. Its main feature is subscribing to the Smart Garden devices and reading their published messages. Using the information in these messages they populate an easy to ready view of all your plant vitals as well as sends notifications when the garden needs to be watered. Without the app the hardware and user would have no clear way of interacting with each other.

The final Smart Garden design was very close to meeting our expectations. Unfortunately there was a hardware issue with our daughter board (sensor board), as a result the ESP could not find the address of the ADS1015 and therefore, could not read the three different sensor values. However, our overall project design was successful as when using breakout boards to replace the daughter board, our project correctly worked as planned. The mobile application accurately showed the threshold value of each sensor. The mobile application is user friendly and allows the user to track how the plant is doing and receives notifications whenever the plant needed to be watered. The Smart Garden was able to use the MQTT protocol and correctly subscribe and publish the information needed to the mobile application. The sensor system used parts that offered a level of accuracy that was necessary for the device to operate. While the sensors didn't provide extreme accuracy, the sensor system was designed with budget in mind as we found the cost of the device to take precedence over the extreme accuracy. A plant user doesn't need to know the exact temperature, light, and moisture level of the plant, but rather needs to know if the plant needs to be watered or not, or if the plant is in sunlight or not. These ranges can be greater and thus the need for extreme accuracy was not worth the price that other parts could

provide the sensor system. Thus the design of our sensor system met our expectations and was accurate as needed to be.

# 2. Detailed System Requirements

- Sensors must accurately measure light, temperature, and moisture levels of plant.
- Sensor values must be properly converted from analog signals to a digital signals.
- The power supply must supply 3.3 V to the sensor system
- Powered by a single cell LiPo rechargeable battery
- Device must be WiFi enabled
- ESP8266 must connect to ADC device to receive data.
- ESP8266, via MQTT Protocol, must communicate with iOS Mobile Application.
- These must be published as retained messages.
- ESP8266 must be able to deep sleep and wake up when necessary.
- Mobile Application must send an alert to user when the moisture levels hit critical values.

To ensure the smart garden was a well functioning product, there were a number of requirements the system had to meet (outlined above). The sensors were an important part of the smart garden. Each sensor was responsible for gathering critical information that would be available for the user. There were three different sensors that were used in the smart garden. The three sensors used were a temperature, moisture, and light sensor. The temperature sensor consisted of a thermistor and two resistors and measured resistivity inversely to the temperature. The moisture sensor measured the resistance between two pieces of metal that were at a set distance. The light sensor consisted of a photocell and resistor. As light increases, the photocell has a decreasing resistance. Each of these sensors were incorporated on the daughter board (sensor board). The ADS1015 part on the daughter board was used to read each of these three sensor analog signals and outputted a digital signal to the ESP.

With all of our hardware, the smart garden was determined to require a 3.3 voltage, overall. In order to regulate 3.3V across the device, a lithium polymer battery, coupled with a

voltage regulating circuit were needed. The battery supplied efficient power to our device, and the regulator ensured that the battery had an output of 3.3V.

After power had been delivered to the system, communication between the user and the device was necessary. For the scope of our project the ESP8266 12 (ESP) was found to be a sufficient answer. The ESP would function as the microcontroller for the device, as well as the bridge of communication between the sensors and the user. In order to make the communication possible, an information broker is used to push and receive data. MQTT protocol was decided as the best method for communication between the two. Proper communication involved using the ESP to publish data to specific topics, and an app, subscribed to the same topic, was able to access the information.

The Mobile Application had to be capable of subscribing to the ESP's MQTT topics and displaying the information published to those topics. It had to allow for users to add new ESP devices and select which of those devices to monitor. It also had to notify users when the plant needed to be watered. Most importantly the Application had to be intuitive and easy to use for the user.

Because the device will often be away from an outlet, and even outside, battery life is a major focus of the project. One way of keeping a high battery life, involves putting the ESP into a low state of power usage. This low state of power is called deep sleep. With deep sleep enabled the device practically shuts down, only waiting for a wake up signal to resume normal functioning. Built into the ESP device is a WiFi antenna, which is how the device communicates between the ESP and the user.

# 3. Detailed Project Description

## 3.1 System Theory of Operation

The Smart Garden is WiFi enabled, so the user can be notified of a plant's moisture, temperature, and light levels via a mobile application. The mobile application will also notify the user when moisture levels are low and the plant needs to be watered. The temperature sensor has

three different thresholds: below room temperature, room temperature, and above room temperature. The moisture sensor has two thresholds (but could be programmed more specifically in the future based on plant needs): dry and watered. The light sensor has four thresholds: sunlight low, in shade, in sunlight, and in direct sunlight. The mobile application allows the user to see the most recent threshold from the three sensors monitored and updates these thresholds every 4 to 5 hours. The user receives notifications whenever the plant needs to be watered as this is the most vital factor to be monitored for plants. The device is powered by a single cell LiPo battery and includes a charging circuit that allows this battery to be charged via USB as well as a voltage regulator that keeps the output of the battery at 3.3 V. The device has two casing options. For smaller plants, the device can be velcroed to the side of the pot in a waterproof case and the moisture sensor overhung the top of the pot and stuck directly into the soil of the plant. For larger potted plants or outdoor gardens, the device is in a waterproof case with a decorative snail with the moisture sensor coming out of the bottom of the device so it stuck directly into the ground or soil of larger potted plants.

## 3.2 System Block Diagram

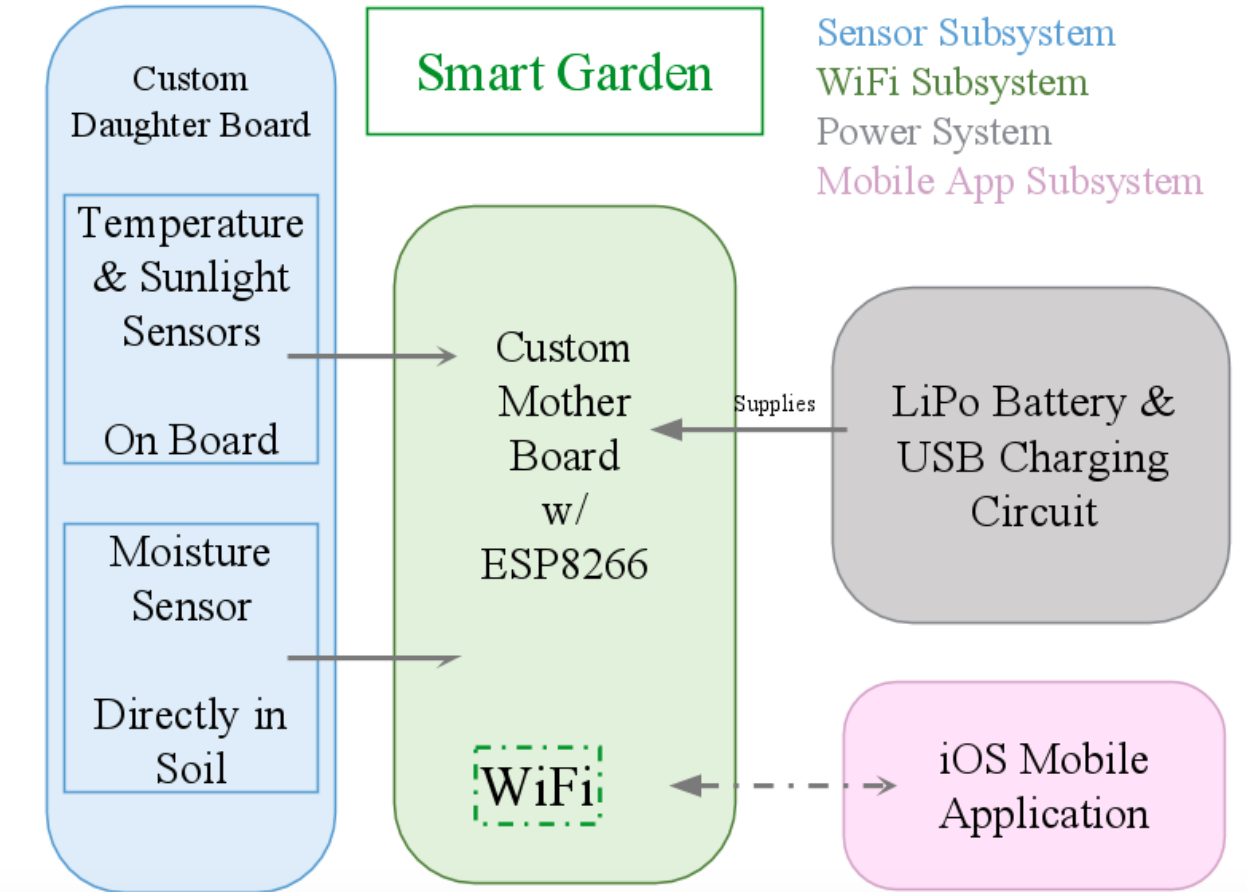



*Figure 1: System Block Diagram*

## 3.3 Detailed Operation of Daughter board (Sensor board) Subsystem

### 3.3.1 Daughter board Requirements:

- Sensors must accurately measure light, temperature, and moisture levels of plant.
- Sensor values must be properly converted from analog signals to a digital signals.

### 3.3.2 Daughter board Flow Chart:

The function of the sensor subsystem is to monitor the light, temperature, and moisture level of the plant and to output values (i.e. when light levels are low) to the ADS1015. The temperature, light, and moisture sensor system is incorporated within the daughterboard (see schematic in Appendices). The moisture sensor is placed into the soil of the potted plant and is

wired to the daughter board. The temperature and light sensors are embedded into the daughter board. The general flow chart of each sensor is:

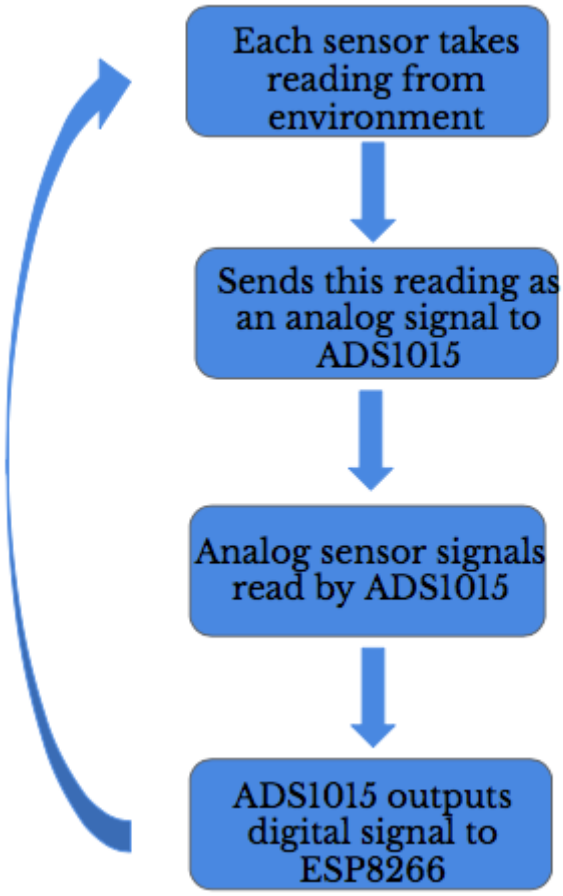
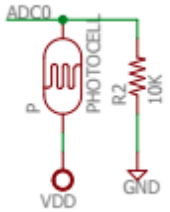
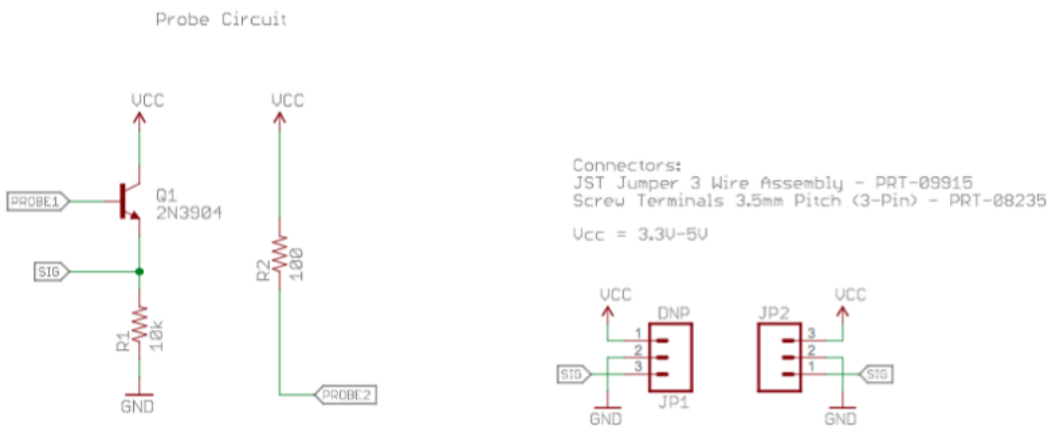



*Figure 2: Sensor Flowchart*

### 3.3.3 Daughter board Schematics:

Light Sensor Schematic: a simple circuit was designed using the voltage divider rule. The output voltage is directly proportional to the input voltage, which is 3.3 V. The output voltage has a factor of R1/(R1+R2). The photocell is in place of R1, and R2 is set at a fixed value of 10 kOhms. The photocell is then connected to the ADC signal pin (ADC0), which is used to measure the ambient light level. The value of the photocells' resistance varies with the light level.






As the photocell is exposed to more light, the resistance decreases. The schematic of the light sensor is:

*Figure 3: Light Sensor Schematic*

Temperature Sensor Schematic: the temperature sensor is connected to VDD (3.3 V for best precision) and ground. ADC1 is connected to our ADC signal pin and will be outputting the temperature value. A thermistor, 10K resistor, and 1K resistor are connected as seen below:

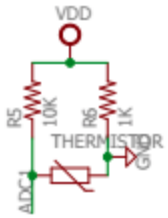



*Figure 4: Temperature Sensor Schematic*

Moisture Sensor Schematic: A SparkFun soil moisture sensor was used because it was the best suited for our purposes as it was a more exact way to measure moisture. Vcc of the SparkFun sensor is connected to 3.3 V and GND is connected to ground. The schematics for the SparkFun soil moisture sensor are as follows:

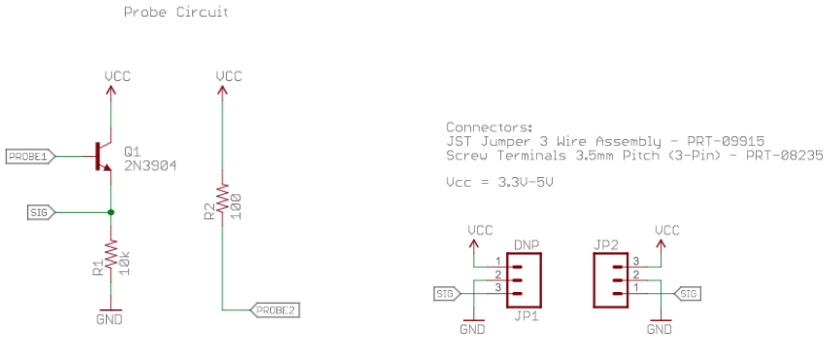



*Figure 5: Moisture Sensor Schematic*

### 3.3.4 Daughter board Software:

The complete software can be found in the Appendix. In the code, each of the sensors were coded as the corresponding pin that each was connected to on the ADC due to the pin-selectable addresses on the ADC. The different thresholds of each sensor was hard coded based on values on the serial monitor in Arduino found during testing.

### 3.3.5 Daughter board Testing:

For testing purposes, a breakout board of the ESP8266 DEV thing on a breadboard was used to power the different sensors. The sensor values were read through a breakout board of the ADS1015. The light sensor consisted of a photocell and 10 kOhm resistor connected to a VDD of 3.3 V and ground on the DEV thing breakout board as shown above in the schematic. The ADC0 pin was connected to the A1 pin on the ADS1015 breakout board. The DEV thing was programmed using Arduino with the code found above in the software section.
Initially, the different thresholds values where the photocell was lacking light, shaded, in light, and in direct light needed to be found and calibrated. These values were read in from the ADS pin as a hand was placed over the photocell to shade it or place it in complete darkness. A light was shone on the photocell for direct light. Once these different threshold values were obtained, the Arduino serial monitor printed out what level of light the photocell was detecting and was very accurate.

For the moisture sensor, a SparkFun soil moisture sensor was used and connected the Vcc on the moisture sensor to the VDD on the DEV thing breakout board. The GND was connected

to ground on the DEV thing and the SIG connected to the ADS pin A0. When the sensor was stuck into dry dirt and then watered, the threshold values for being dry and watered were found and implemented into the code. When the dirt was watered, a message was displayed that the plant was watered on the serial monitor and if the dirt was too dry, a message on the serial monitor displayed that the plant was too dry. Since our own thresholds were found prior to testing the overall code, this moisture sensor proved to be highly accurate.
The temperature sensor was connected to 3.3 V on the Arduino, as this allowed for the temperature sensor to be more precise and less noisy. The ADC1 was connected to the ADS pin A2 and GND is connected to ground. The sensor was then tested by placing two fingers to warm and increase the temperature and used an ice pack to test the lower temperatures. The raw temperature value and this temperature in both Celsius and Fahrenheit were displayed on the Arduino serial monitor. The next step was testing our daughter board with the mother board, which is outlined in section 4.

## 3.4 Detailed Operation of Mother board (WiFi and Power) Subsystem

### 3.4.1 Mother board Requirements:

- Device must be WiFi enabled
- ESP8266 must connect to ADC device to receive data.
- ESP8266 via MQTT Protocol must communicate with iOS Mobile Application.
- These must be published as retained messages.
- ESP8266 must be able to deep sleep and wake up when necessary.
- The power supply must supply 3.3 V to the sensor system
- Powered by a single cell LiPo rechargeable battery

### 3.4.2 Mother board Schematics/ Software:

The WiFi subsystem functions as a bridge between the hardware and the mobile app. Based on its availability and the Smart Garden's requirements, the ESP8266 12 (ESP) was determined to be the correct choice. The ESP has the function of a microcontroller, while also having integrated WiFi capabilities. Figure 6 shows the schematic for the ESP device. Aside from the powering and grounding of the device, there were a number of connection that were made

for the ESP to function properly within the Smart Garden. RESET and GPIO16 were hardwired together in order to use one of the ESP's built in functions, called deepsleep. Deepsleep enables the ESP to put itself into a state of low voltage. The low voltage increases the life of the battery dramatically. Along with deepsleep capabilities, the ESP was the main microcontroller for the Smart Garden. The Smart Garden did not need to be programmed more than once, and there was no other function for integrating a FTDI, or USB interface; therefore, an external USB-UART programmer was chosen.

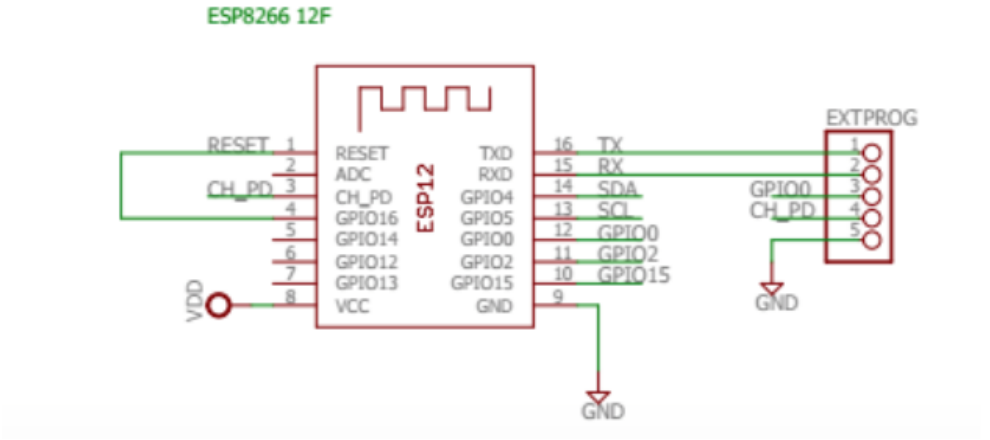



Figure 6:

ESP8266 12 Schematic

The UART utilizes a specific set of pins for programming, specifically: TX and RX. TX and RX transmit and receive instructions. There are three other pins that determine the ESP's mode of operation: GPIO15, GPIO0, and GPIO2. In order to get the ESP into the UART configuration GPIO0 and GPIO2 are pulled high, while GPIO15 was pulled low. Although stated that GPIO0 should be low, it is only during programming that it was necessary, and the external programmer has the capability of pulling GPIO0 low.

The final step for programming the ESP externally involves the ability to reset the device when GPIO0 was pulled low. This can be done either through RESET or CH_PD. Because RESET was used in the ESP's deepsleep functionality, CH_PD was chosen to be used when resetting the device during programming. There were also a number of pins that had to be pulled high in order for the ESP to work properly. The complete list of pins can be seen in Figure 7 below.


*Figure 7: Pull-up Pull-down Configuration*

The power that supplied the ESP was a single cell LiPo battery. The hardware necessary to charge this LiPo battery is shown below in figure 8. There is a port to plug in a USB to charge the battery and a molex to attach the single cell LiPo battery to power the boards. The MCP73831 is a Li-Polymer charge management controller. A data sheet for the MCP73831 can be found in the Appendix.

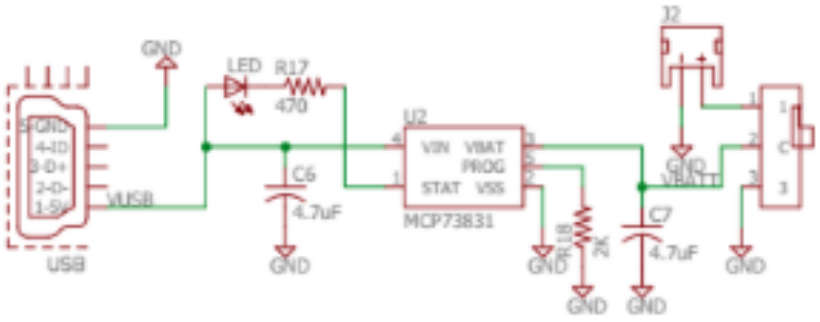



*Figure 8: Battery Charging Circuit*

The power of the battery needed to be regulated at a constant 3.3 V to power the ESP and daughter board containing the sensor system. The mother board consisted of a voltage boost converter shown below in figure 9 that allowed for this design to be possible. The TPS61200 outputs a 300 mA current at 3.3 V. A data sheet for this part can be found in the Appendix.

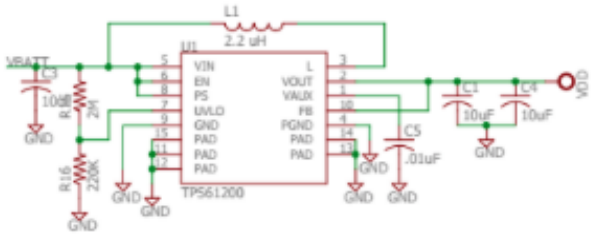



*Figure 9: Voltage Boost Converter Circuit*

The final piece of hardware that the ESP needs to be connected with was the analog to digital converter (ADC). The ESP was responsible for controlling the ADC and then properly distributing the data that was fed to it. The communication between the two integrated circuits was done through I2C. The I2C protocol, shown in Figure 10, uses only two lines to communicate. SDA transmits the data between the two microcontrollers, and SCL is the clock that synchronizes the use of the SDA line. The Smart Garden utilizes the ESP as the master and the ADC as the slave. Since the ESP is the hub of the device, it controls the clock and when the communication occurs. On the ESP there are built in SDA and SCL pins; therefore connecting the two was a straightforward process.

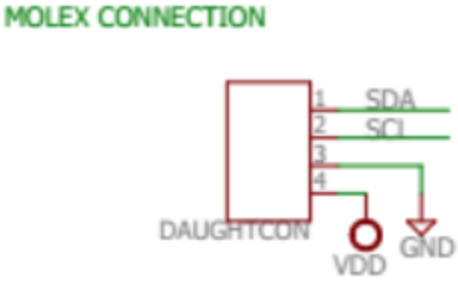



*Figure 10: I2C connection between Mother and Daughter Board*

Once the ESP was properly wired, it also needed to be properly programmed to perform the tasks needed by the Smart Garden. An entire listing of the program can be found in the

appendix, figure ??. The main breakdown of the ESP requirements involved: Initialization, establishing WiFi connection, connecting to MQTT broker, gathering values from ADC, and then publishing those values to the MQTT server. Figure 11 shows simple flowchart of the software for the ESP. Establishing a WiFi connection was done one of two ways. For a first time connection, the ESP makes itself an access point. As an access point the user is able to choose the WiFi connection they wish. After the ESP is connected it will store the connection, and reuse it every time it awakens from deepsleep. If at any time the ESP comes out of deepsleep and cannot establish the connection given to it, it will make itself an access point and repeat the initialization process.

After a WiFi connection has been established, the ESP will then attempt to connect to MQTT. If no connection can be found, the ESP will pause and then attempt to reconnect. It will continue to do this until a connection is established. One connected to the MQTT broker, the ESP can then gather the data that has been converted by the ADC. These digital signals are put in to comparative statements and simplified for for the user. Once the values are correctly sorted, they are published to the MQTT broker, for the users' app to subscribe to.

Finally, after completing the publishing process, the ESP will go into deepsleep for a predetermined amount of time. After the designated time, it will awaken and repeat the process to update the user with the new information of their plant. All coding of the ESP was completed using Arduino. Arduino was chosen because of its' compatibility for testing each of the breakout boards, as well as its integration into the chosen parts for the ESP. Using Arduino IDE, made it simple to transition from the testing phase of the Smart Garden to a completed project.

### 3.4.3 Mother board Subsystem Flowchart:

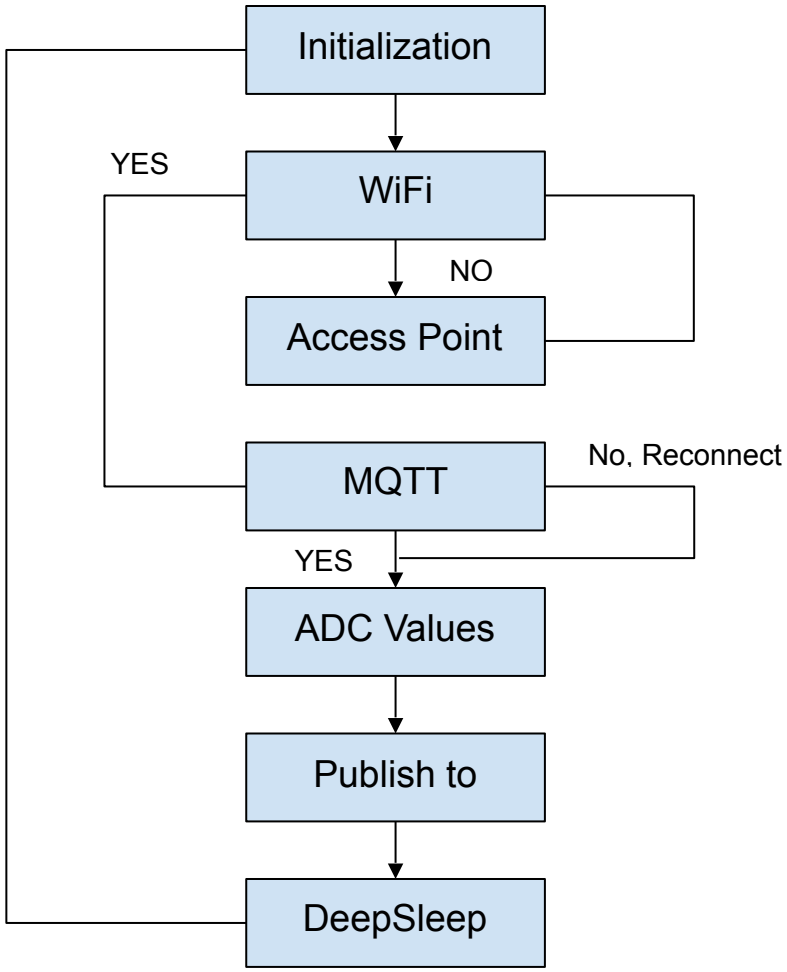



*Figure 11: ESP Software Flowchart*

As noted above, MQTT protocol was decided as the correct method of communicating between the Smart Garden and the mobile application. Using a publish and subscribe method, MQTT was capable of performing the tasks necessary for the scope of the project. Using unique topics, data can be published to the broker, and anything that is subscribed to the same topic will be able to view the data.

### 3.4.4 Testing of Mother board Subsystem

Testing began with just an ESP breakout board called, ESP8266 Dev Thing. This board included simple programming through a built in USB, and all of the functionality needed in the Smart Garden. The first test involved connecting WiFi. the SSID and password were hard-coded

into the device. Directly choosing the WiFi address allowed different testing to be done without having to attempt connection everytime. After connecting to WiFi, the functionality of the ESP was tested. This included: deepsleep, creating an access point, and posting a unique address to a test page. Testing the deepsleep functionality was similar to the code that was used in the final software; however, the access point and posting would be adapted in later testing.

After testing the Dev thing alone, it was time to incorporate it with MQTT protocol. In this test, data was generated within the code to make testing MQTTs' integration easier. Unique addresses were created and those topics were published onto the MQTT server. Using a separate program the topics were subscribed to in order to validate the publications. With MQTT and WiFi capabilities working properly, an Arduino ADS1015 breakout board was added into the testing. The breakout board test the data acquisition of the sensors, and relayed that information to the ESP as a digital signal. Both breakout boards were used to make sure the ESP could read in data and then continue to publish that data to MQTT properly.

Once the breakout boards were functioning properly, and the Smart Garden prototype had come in, testing resumed to integrate the two Smart Garden boards into the system. First, the mother board was subjected to the same testing of the Dev Thing that had been used earlier. Because of the programming language type chosen, testing the mother board was fairly simple. The correct connections between the board and the ADC breakout board were made, and the same code could be run. The ADS required a declaration of the SDA and SCL wires, which was done in the initialization phase of the code.

After the mother board was tested properly for the same functionality as the Dev Thing, the daughter board was swapped in for the ADC breakout board and the same process was completed.

## 3.5 Detailed Operation of Mobile App Subsystem

### 3.5.1 Mobile Application Requirements

- Mobile Application must send an alert to user when the moisture levels hit critical values.
- ESP8266, via MQTT Protocol, must communicate with iOS Mobile Application.
- These must be published as retained messages.

### 3.5.2 Mobile Application Software

The Mobile Application's main function was to subscribe to messages published by the ESP. It also had to monitor a plant's water situation and notify the user if the plant needs to be watered. In order to meet the requirements there had to be a way for iOS to subscribe to MQTT topics. There is no built in way to do this in Xcode so the first step was to find a library. After filtering through a lot of open source libraries, CocoaMQTT was selected because it was the most robust. Unfortunately, there was not a lot of documentation available so example code was used to try to decipher how to use the library. Once that was figured out, connecting the App and the ESP was just a matter of syncing the topic names.

After the most essential function was completed, additional features were added. An add a plant view was added so that more than one plant could be tracked. This introduced the problem of how to distinguish between ESP devices. We decided to hard code a unique value into each ESP that would be used in the topic name and would be included in the packaging of "shipped" Smart Garden units. On the app side this number is entered in the add the plant page and associated with a "Plant Name". The "Plant Name" is essentially a nickname for your plant so that when you are searching through your "Smart Garden" you can easily identify your different plants. Finally, a home page of sorts was added. This is the central hub of the App. From here you can access the "Add a Plant" or scroll through the picker wheel and select the plant you want to check on.

### 3.5.3 Mobile Application Flowchart

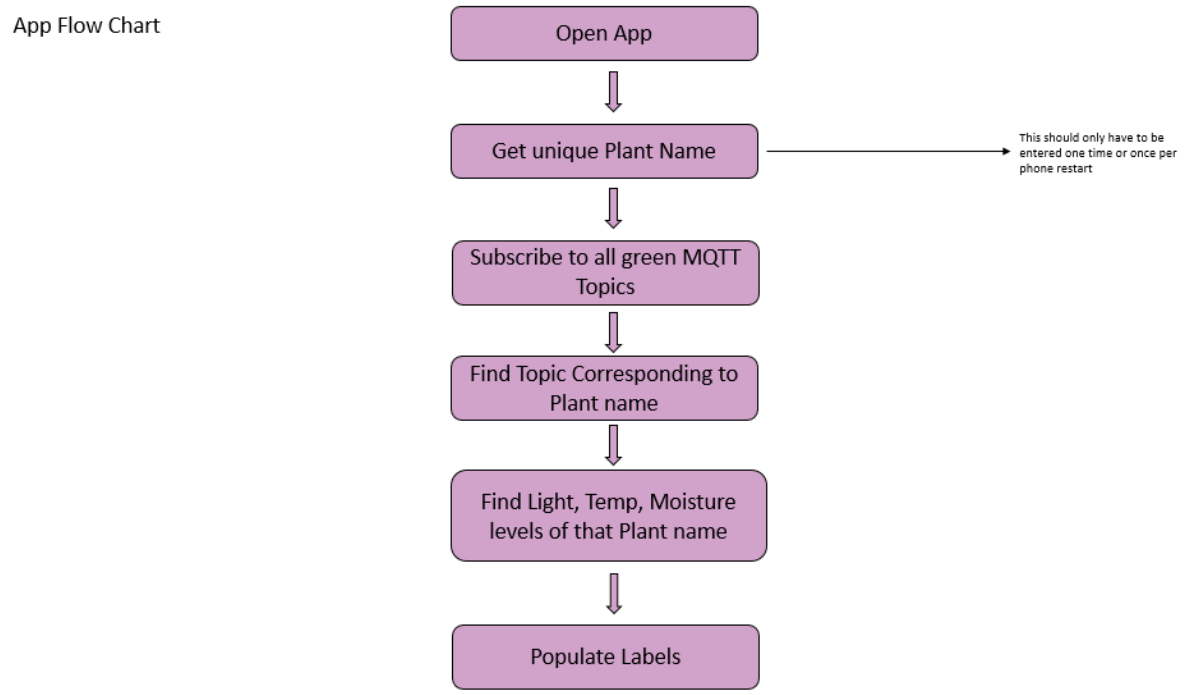



*Figure 12: Mobile Application Flow Chart*

When the application is opened for the first time the user will enter the Garden and proceed to the "Add a Plant" feature. From this page the user can add specific plant names associated to specific Smart Garden devices. The user will then select the plant they would like to check on and click "Press to Enter" this will bring them to a new view that shows the information collected by the sensors. Additionally the app will also send an alert when the moisture level hits a critical level.

What is going on behind the scenes is the app will subscribe to all grEEn topics ( grEEn/ + ) and from there unique ESP identifier will be added to the topic string and the values for light, temperature, and moisture for that particular device will populate the labels.

### 3.5.4 Mobile Application Testing

Testing the Mobile App's MQTT aspect was as simple as using MQTT.fx to publish messages and checking if the app was receiving those same messages. The difficult part was the lack of documentation for the MQTT library. Code from example apps was combed through in an effort to try to figure out how the library worked. Subscribing was not as straightforward as

expect but it was eventually figured out. Extensive testing came while testing the notification system. The alert code was put in many different spots of the code to see which worked best. Many different spots were tested before ultimately deciding on putting it within the MQTT subscribe function.

## 3.6 Interfaces

### 3.6.1 MQTT

MQTT functions on a publish and subscribing method. Publish and subscribe are control packets that are built into MQTT software. These commands allow for communication from one machine to another. One device is able to publish data using a specific topic, and the other device is able to access that information by subscribing to the same topic. MQTT also allows for publishing clients to subscribe to their own publications, which is useful during testing. The most important part of ensuring that the communication is possible, is making sure that the topics are the same between the two devices. MQTT protocol was found to be the correct method of communication because the data packets that are being sent are very small, and MQTT excels in the transfer of small amounts of data; therefore, for the scope of the Smart Garden, MQTT was sufficient.

### 3.6.2 I2C

The ADS1015 used an I2C interface with the ESP8266. This inter-integrated communication (I2C) interface required SDA, SCL, VDD, and GND to be connected between the ESP and ADC. The I2C bus consists of these two signals: SCL and SDA, where SCL is the clock signal and SDA is the data signal. This communication protocol allows multiple slave chips, but in this case, we only had one slave chip: ADC, to communicate with a master chip (ESP). The ADC (analog digital converter) read the three analog sensor values and outputs digital signals to the ESP using this I2C interface. Data is transferred through the I2C-compatible serial interface. The SCL and SDA signals are both digital signals that are read by the ESP which is the reason the ADC is necessary for the Smart Garden to function. The analog signals from each of the sensors must be converted to digital signals in order for the ESP to read these values. The ADDR input of the I2C interface is the I2C slave address select which is necessary to know when

programming the ESP. There are a number of different wiring configurations that determine the address the ADC will be using. The Smart Garden used the default address on the breakout board, which involved wiring ADDR to ground.

# 4. System Integrated Testing

### 4.1 Integrated Subsystem Testing

In order to integrate the Smart Garden's subsystems the software and hardware components needed to be merged. Once the mother board and daughter board were soldered and connected to each other via the four pin molex, the ESP was programmed with the code found in the appendix. To test if the ESP was programmed correctly, the logic analyzer was used to make sure the ADS1015 on the daughter board was correctly reading the three analog signal values.

Unfortunately, it was during this step of testing the daughter board, we realized that the ADC part on our daughter board was missing hardware as the address was not being read by the ESP and therefore, the ESP was not correctly reading the digital signal from the ADC nor was the ADC correctly reading the three sensor analog signals. Through testing, we realized the ADDR pin on the ADC wasn't grounded, which caused the address on the ADC to be floating. By pulling the ADDR low, the address was no longer floating. Unfortunately this wasn't enough to fix our hardware issue with our daughter board. However, using the breakout boards, the address was able to be found and the ESP was able to be programmed to receive the digital value outputted from the ADC. Due to time constraints, we focused on making sure our overall project worked and spent as much time as possible troubleshooting the hardware issue on the daughter board.

When we couldn't fix the hardware issue, we went back to using the ADS1015 breakout board part and breadboarding the other parts on the daughter board to test that our mother board was working. Using Arduino to program the ESP and pulling up the serial monitor showed the values were correctly being read from the sensors to the ADC to the ESP. After our software worked with our hardware, the next step was testing our device with the mobile application.

To test the mobile application, first MQTT.fx was used to publish messages to the app's topics. Once the hardware was finished and the topic names were synced, the app was tested with these messages published by the ESP. It worked seamlessly and not a lot of testing had to be done for this interface.

### 4.2 Meets Design Requirements

After integrated subsystem testing, the Smart Garden met all of the design requirements as discussed in section 2. The three sensors each were designed to accurately measure light, temperature, and moisture levels of plant. For our prototype, these values were loosely calibrated as different ranges. These values could be made more accurate in the future. The temperature value of the plant could be designed to accurately read the current temperature of the plant in either Celsius or Fahrenheit. However, due to the plant needing to just be in room temperature during the day, this accuracy was not needed for our project to be successful. The light values were also loosely calibrated into different ranges: sunlight was low, plant is shaded, plant is in light, and plant is in direct light. These values were hard coded and programmed into the ESP. Once again, these values did not need to be extremely accurate as the user really only needed to know whether or not the plant was in light. If the plant was shaded or if sunlight was low, the user would know to move the plant into an area with more light. The moisture values were also hard coded into two ranges, whether the plant was watered or not. For the design requirements of our project, these ranges were enough to meet the requirements. However, if given more time and The sensor system was designed with budget in mind instead of precision. The parts in the sensor system were fairly inexpensive allowing for a cost-friendly product.

These sensor values needed to be properly converted from analog signals to digital signals. This was successfully done using an ADS1015 to read the analog sensor signals and output the digital SDA and SCL signals to the ESP8266. The I2C interface allowed the ESP microcontroller to communicate with the ADS1015 to receive this data.

The device was powered by a single LiPo rechargeable battery with a charging circuit that allowed the battery to be recharged. The battery circuit also included a voltage regulator that outputted a consistent power supply of 3.3 V to power the contents on the device, including the ESP and the sensor system. Due to power-saving reasons, the ESP8266 used deepsleep and was

also able to wake up every 4 to 5 hours to perform sensor readings so the user could view these values on the mobile application.

The device was also WiFi enabled which allowed the device to connect to the mobile application that provided a user-interface with the Smart Garden. Using MQTT protocol, the ESP8266 accurately communicated with this iOS mobile application. The data was published as retained messages. This design allowed the user to see the values the ESP read from the sensors on the mobile application on the user's phone. The retained messages allowed the user to see the latest message because they only update when the next message comes in. The mobile application sent an alert via a notification on the user's phone when the moisture levels hit critical values, meaning the plant needed to be watered. While they work, the notifications lack some reliability because of the limitations of using Xcode without having a developer license. This robustness is what will be improved with the addition of an iOS Developer before we go to market.

## 5. User Manual/ Installation Manual

### 5.1 How to Install Smart Garden Application

To install the application you must have Xcode downloaded on your computer and your iPhone connected to via USB. Download the zip file of the Xcode project and open up the SmartGarden project. Once the project is open you will see the "run" button in the top left. This can be seen in Figure 13 below.

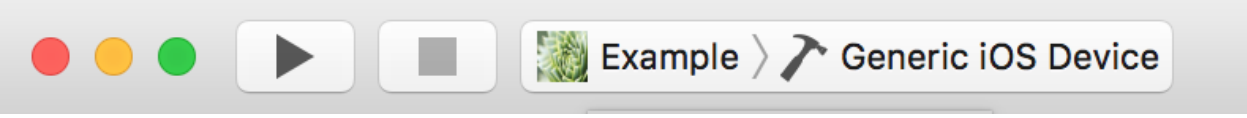



*Figure 13: Run code Xcode Button*

To run the SmartGarden app click the "Generic iOS Device" button and select your phone from the drop down menu. Follow any prompts they give you. The app should build and begin to run automatically on your phone. Eventually, once this goes to market it installing the application would be as simple as searching "Smart Garden" on the App Store.

### 5.2 How to Setup the Smart Garden

To set-up the Smart Garden, the user must take the velcro strip and adhesive strip, that comes with the casing, and stick the velcro strip to the side of the pot of the plant that requires monitoring. Then, the user must open the casing with the Smart Garden device and plug a USB micro into the device to charge the battery. The light on the mother board next to the USB will light up when charging. The user can unplug the Smart Garden from the USB and switch the device to ON. The battery should last for about six months.Velcro the casing with the Smart Garden to the side of the potted plant while placing the moisture sensor into the dirt of the plant. The device is good to go!

The next step is setting up the mobile application on your phone. Once the mobile application is downloaded onto your phone you will first be prompted to allow notifications. This is necessary so that you can receive alerts when the plants need to be watered. The next step is to "Enter" the app, on the second screen in the top right hand corner you will see an "Add a Plant" button. This will take you to a form where you will enter your factory ID for your specific Smart Garden device as well as a Plant Name, this can be anything but it basically a nickname for that device. Once you hit submit you are ready to go. To access your plant's information you select it from the picker wheel and press the "Press to Enter" button.

### 5.3 Working Smart Garden

The first way to see if the Smart Garden is working is to begin charging the device. A red LED should begin emitting, and signify that the battery is charging. The best way to test if the Smart Garden is working is to begin using the product and compare the information being given to data that is available. For example, compare the reading of the temperature, from the device, to the reading a thermostat gives. If the device reads that the the plant is in room temperature, a reading of approximately 72 degrees should be seen from the thermostat. A properly working device will give simplified measurements for water, light, and moisture. All three of these readings should be comparable to inspecting the plant that is being monitored. Finally, if the switch of the device is toggled, the Smart Garden should turn on and off.

## 5.4 Troubleshooting

### 5.4.1 Incorrect Readouts on App

If the Smart Garden has any incorrect readouts (i.e. “dry, water plant” after the user just watered the plant), first restart your device. If this doesn’t work make sure all the sensors are in the correct place. For example, if the mobile application is showing that the plant is dry but the plant’s soil is moist, make sure the moisture sensor is fully in the soil. Another potential situation is, if your plant is in light but reporting it is in the shade, check to make sure no leaves are covering the Smart Garden. If you are still having trouble contact Smart Garden customer support.

### 5.4.2 Disconnected Hardware

If the Smart Garden won’t connect to the internet try turning the device off. Take your cell phone or laptop and connect to the access point “AutoConnectAP”. After the access point connects, an interactive window should pop up asking you to select a WiFi ID and enter the password. This should solve the problem. If you are still having trouble contact Smart Garden customer support.

# 6. To-Market Design Changes

Before bringing the Smart Garden to market, more features should be added. Given more time and a larger budget we hope to add an automatic watering system. This would make the plant completely autonomous. We also hope to reduce the size of the device to make it even more discreet while in the plant. By decreasing the size the Smart Garden we will be able to reside in a wider variety of plants making it more marketable.

The Mobile Application needs to be improved before being released on the app store. The app as we have it now works for our purposes but would need to be much more robust for mass use. We also hope to include the database feature, allowing users to select which specific plant is in their Smart Garden so that the notifications are more specific to plant type. If we had an automatic watering system we would also want to include manual start and shut off for that system in the app. With the addition of the automatic watering system we would also add security so that a

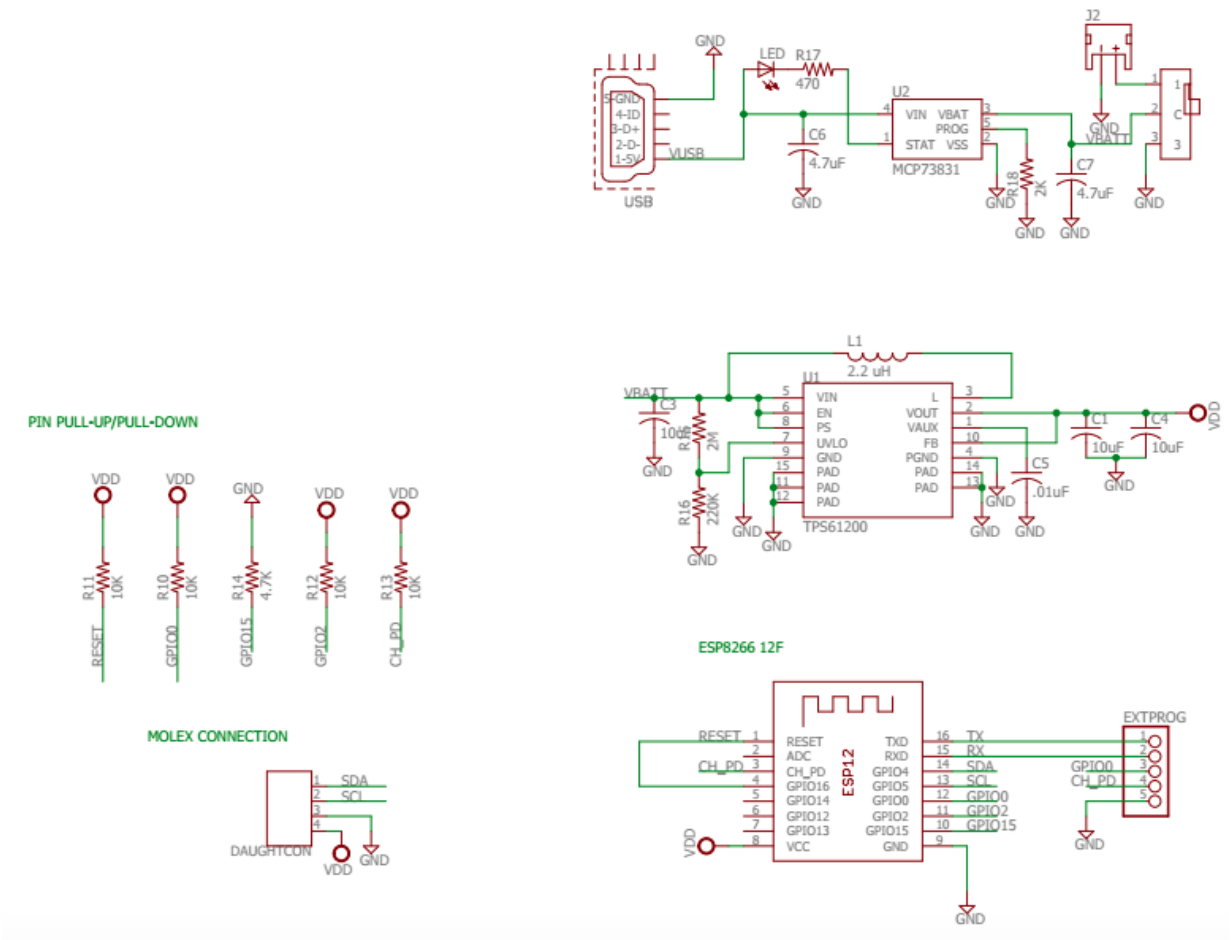
hacker couldn't also control the system. Finally we would want to have an identical app available for Android.

# 7. Conclusion

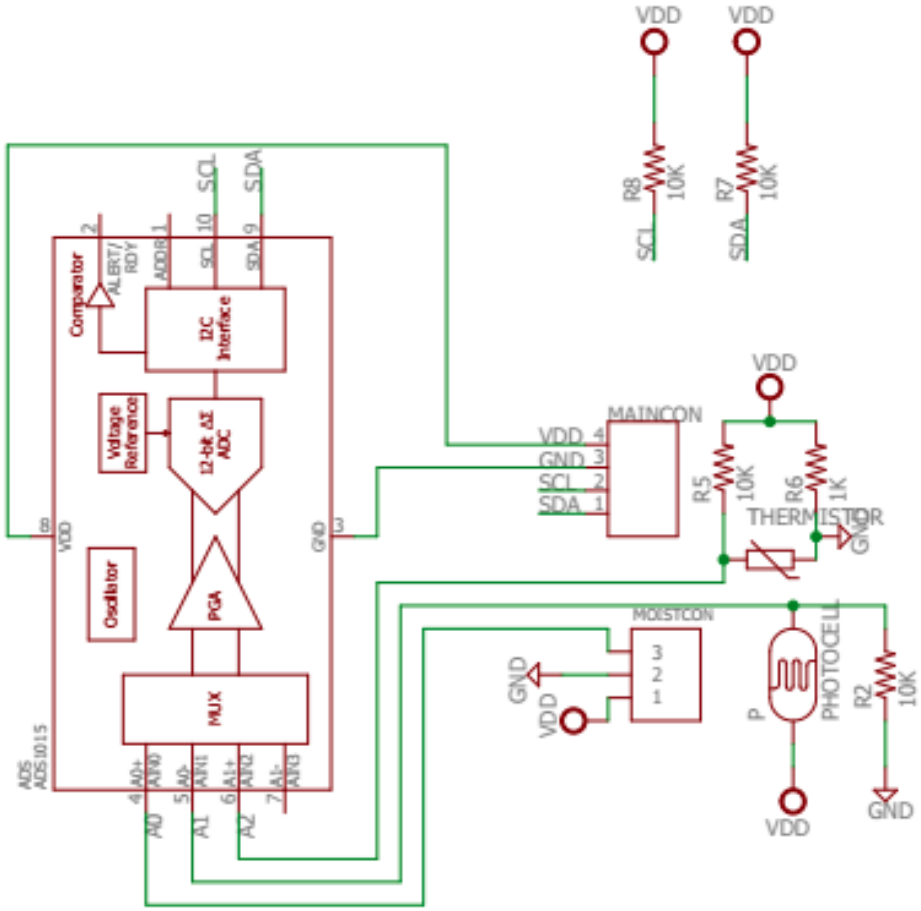
Overall, the Smart Garden is almost completely a success. If given more time and the daughter board hardware issue had been fixed, the project would have been a complete success. However, using a breakout board with each of the components on the daughter board with an ADS1015 breakout board did garner the success we were hoping for. As this was a prototype and the app was able to correctly display the correct sensor readouts: i.e. "Plant is watered" when plant had been watered; "Plant is in room temperature" as the plant was in room temperature consistently and this changed to "Plant is above room temperature, it's hot" when the thermistor was warmed using two fingers; and "Plant is in sunlight" when the plant was in a lit room. The sunlight sensor had four different thresholds: in sunlight, in shade, in direct light, and light is low (when there isn't any light). Our final project accurately showed each correct readout for each of these light thresholds. The moisture sensor had two thresholds and also accurately displayed the correct readouts on the mobile application. The temperature sensor had three thresholds: above room temperature, room temperature, and below room temperature. The basic requirements for the Smart Garden were completed, except for the hardware issue with the daughter board. However, for a prototype, the Smart Garden displayed effective functionality as well as provided a design to build off of for future projects.

# Appendices

## Complete Hardware Schematics




*Mother Board Complete Schematic*




*Daughter Board Complete Schematic*

## Complete Software Listing

**A zip file for the Mobile Application can be found on our website at Mobile Application Online .**

**Code Listing for Hardware:**

```
#include <Adafruit_ADS1015.h>
#include<Wire.h>
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <DNSServer.h>
#include <ESP8266WebServer.h>
#include <WiFiManager.h>

// unique plant id is 1028
#define topic1 "grEEn/0220/moisture"
#define topic2 "grEEn/0220/light"
#define topic3 "grEEn/0220/temp"
#define mqtt_server "senior-mqtt.esc.nd.edu"

Adafruit_ADS1015 ads1015;
WiFiClient espClient;

PubSubClient client(espClient);

void setup() {
  int finish = 0;
  const int  sleepTimeS = 3;


    Serial.begin(9600);
    WiFiManager wifiManager;
    client.setServer(mqtt_server, 1883);
    Wire.begin(5,4);
    ads1015.begin();
```

```
 wifiManager.autoConnect("AutoConnectAP");
 Serial.println("connected...yey :)");
}

void reconnect() {
    // Loop until we're reconnected
    while (!client.connected()) {
     Serial.print("Attempting MQTT connection...");
         //Attempt to connect
        if (client.connect("TestMQTT")) { //* See //NOTE below
           Serial.println("connected");
        } else {
           delay(500);
        }
    }
}
//NOTE: if a user/password is used for MQTT connection use:
//if(client.connect("TestMQTT", mqtt_user, mqtt_password)) {

void pubMQTT(String topic,String topic_val){
    Serial.print("Newest topic " + topic + " value:");
    Serial.println(String(topic_val).c_str());
    client.publish(topic.c_str(), String(topic_val).c_str(),
true);
}
//Variables used in loop()
//int moisturePin = adc0; //need to change based on board
int moistThresholdUp = 600; //need to calibrate
int moistThresholdDown = 500; //need to calibrate
//int lightsensorPin = adc1; //change based on board
int shadeThreshold = 1000; //need to calibrate
int lightThreshold = 1050; //need to calibrate
int directLightThreshold = 1080; //need to calibrate
//int tempPin = adc2; //change based on board
int lowTemp = 570;
int highTemp = 605;
double REF_VOLTAGE = 3.3;
const int  sleepTimeS = 3;
int finish = 0;
```

```
long lastMsg = 0;
float t1 = 10.0;
float t2 = 75.5;
float t3 = 50.5;
int16_t adc0, adc1, adc2;
String tempvar;
String moisturevar;
String lightvar;
void loop() {

 moisturevar = "Plant is watered!";
 lightvar = "Plant is in direct light!";
 tempvar = "Plant is in room temperature!";

  adc0 = ads1015.readADC_SingleEnded(0);
  adc1 = ads1015.readADC_SingleEnded(1);
  adc2 = ads1015.readADC_SingleEnded(2);

  Serial.print("Moisture: "); Serial.println(adc0);
  Serial.print("Light: "); Serial.println(adc1);
  Serial.print("Temp: "); Serial.println(adc2);

  Serial.println(" ");

  delay(1000);

if (adc0 <= moistThresholdDown){
      Serial.println("Dry, water plant!");
      moisturevar = "Dry, water plant";
    }

    else if (adc0 >= moistThresholdUp){
      Serial.println("Plant is watered!");
      moisturevar = "Plant is watered!";
    }

    delay(1000);

    //check light values
```

```
    //If light level is low is detected, print light level low
  if (adc1 < shadeThreshold){
    Serial.println("light is low!");
    lightvar = "light is low!";
  }
  else if (adc1 < lightThreshold && adc1 > shadeThreshold){
    Serial.println("Plant is in shade!");
    lightvar = "Plant is in shade!";
  }
  else if (adc1 < directLightThreshold && adc1 > lightThreshold)
{
    Serial.println("Plant is in light!");
    lightvar = "Plant is in light!";
  }
  else if (adc1 > directLightThreshold){
    Serial.println("Plant is in direct light!");
    lightvar = "Plant is in direct light!";
  }
    delay(1000);

//temperature comparisons
 if (adc2 > highTemp){
    Serial.println("Plant is less than room temperature. It's
cold!");
    tempvar = "Plant is less than room temperature. It's cold!";
  }
  else if (adc2 < lowTemp && adc2 > highTemp){
    Serial.println("Plant is in room temperature!");
    tempvar = "Plant is in room temperature!";
  }
  else if (adc2 < lowTemp){
    Serial.println("Plant is higher than room temperature. It's
hot!");
    tempvar = "Plant is higher than room temperature. It's
hot!";
  }

  delay(1000);
```

```
 if (!client.connected()) {
    reconnect();
  }
  client.loop();
  //2 seconds minimum between Read Sensors and Publish
  long now = millis();
  if (now - lastMsg > 2000) {
      lastMsg = now;
      //Read Sensors (simulate by increasing the values, range:
0-90)
      //Publish Values to MQTT broker
       pubMQTT(topic1,moisturevar);
       pubMQTT(topic2,lightvar);
       pubMQTT(topic3,tempvar); // Tempurature needed still...
    ESP.deepSleep(sleepTimeS * 1000000);
    Serial.print("Delay ended");

      return;
  }
}
```

## Relevant Data Sheets

ADS1015: http://www.ti.com/lit/ds/symlink/ads1015.pdf

MCP73831: http://ww1.microchip.com/downloads/en/DeviceDoc/20001984g.pdf

TPS61200: http://www.ti.com/lit/ds/symlink/tps61200.pdf