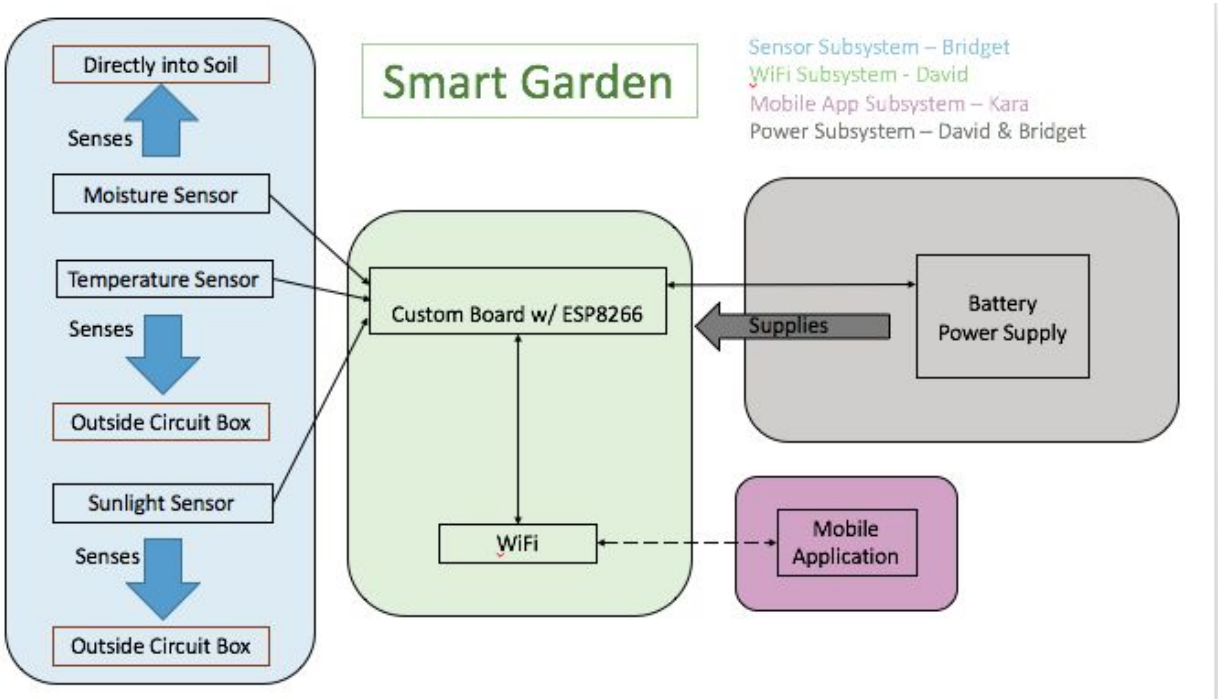
Team GrEEn First Design Review
March 4, 2016

System Theory of Operation:

Many plant owners face the problem of how to care for their plants when they are too busy to constantly monitor these plants. To monitor these plants, moisture, temperature, and sunlight are key factors that need to be detected. Our proposed solution provides an all-in-one device that will have three different sensors to monitor the moisture level, temperature level, and sunlight level of the different types of plants that are growing. The device will be in a 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. This device will stick right into the potted plant. The device will also send these plant owners updates on their plants using Wi-Fi through a mobile application on their phones. The MQTT Protocol facilitates communication between the sensor subsystem and mobile application subsystem through the use of the ESP. Through the mobile application, the plant owners will have an easier way to track how their plants are doing.

System Block Diagram:



Detailed System Requirements:

1. Sensors must accurately measure sunlight, temperature, and moisture levels of plant.
2. Sensor system must connect to the ESP8266 to receive measurement data.

3. ESP8266 via MQTT Protocol must communicate with Mobile Application.
    a. Eventually these must be published as retained messages.
4. ESP8266 must be able to deep sleep and wake up when necessary.
5. Mobile Application must send a push alert when the moisture levels hit critical values.
6. Mobile Application must connect to certain “Garden Names” which will contain unique plants.

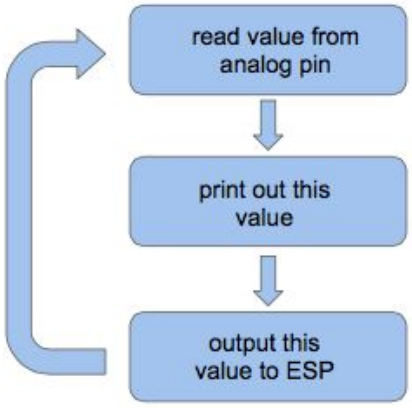
Sensor Subsystem:

System Requirements:

1. Measures sunlight, temperature, and moisture
2. Outputs values to ESP8266 (for test-purposes: outputs to Arduino, displayed on serial monitor)

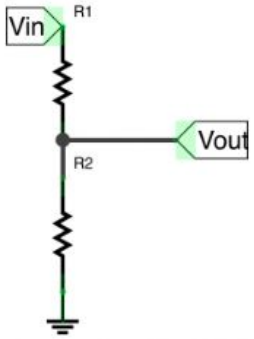
Sensor Flow Chart:

The function of the sensor subsystem is monitoring the sunlight, temperature, and moisture level of the plant and to output values (i.e. when sunlight levels are low) to the ESP8266. The general flow chart of each sensor is:

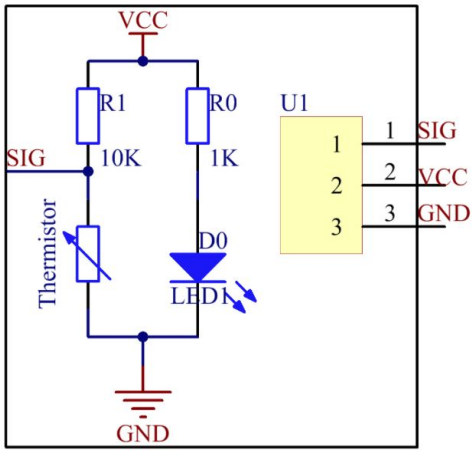


Schematic:

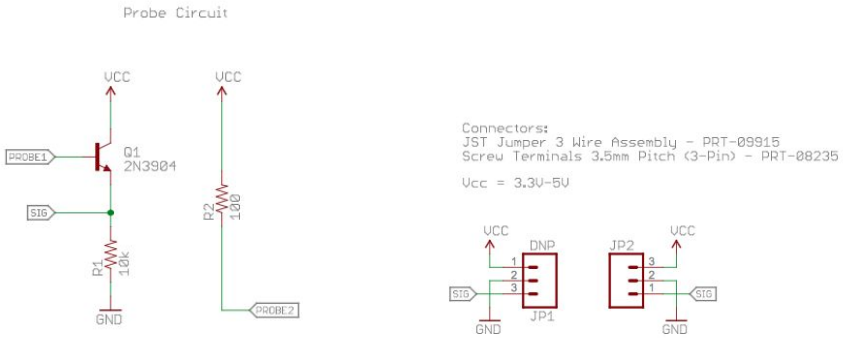
- Sunlight Sensor Schematic: a simple circuit was designed using the voltage divider since the output voltage is directly proportional to the input voltage, with a factor of R1/(R1+R2). The photocell is in place of R1 and R2 is set at a fixed value of 10 kOhms. Vout is used to measure the ambient light level. The value of the photocell resistance varies on the light level. As light increases, a photocell has a decreasing resistance. The schematic of the sunlight sensor is:



- Temperature Sensor Schematic: the temperature sensor is connected to Vin (3.3 V for best precision) and GND is connected to ground. SIG is Vout and connected to the temperature analog pin on the ESP8266 (and on the Arduino for testing purposes). The temperature sensor chosen is the SunFounder NTC Thermistor Sensor Module. The schematic of the temperature sensor is:



- Moisture Sensor Schematic: A SparkFun soil moisture sensor was used to best suit our purposes as it was a more exact way to measure moisture. Vcc of the SparkFun sensor is connected to 5 V and GND is connected to ground. The schematics for the SparkFun soil moisture sensor is as follows:



Software:

The choice of programming language was C as this language is easy to use to program an Arduino for testing purposes as well as the ESP8266. The following code was used for the sensor subsystem:

```
/*
 *
 * Light, Moisture, and Temperature Sensor
 *
 */

 #include <SoftwareSerial.h>

 // pin assignments
 int lightPin = A0;
 int shadeThreshold = 500;
 int lightThreshold = 850;
 int directLightThreshold = 1000;
 int moisturePin = A1;
 int moistThresholdUp = 800;
 int moistThresholdDown = 660;
 int tempPin = A2;
 int tempReading;

 #define aref_voltage 3.3

 // initialize the serial port
 // and declare inputs and outputs
 void setup() {
   pinMode(lightPin, INPUT);
   Serial.begin(9600);
 }

 void loop() {
   //int v = analogRead(sensorPin);
   //Serial.println(v);
   //Read the sensor pin
   //delay(1000);
  int sensorValue = analogRead(lightPin);
  //If light level is low is detected, print sunlight level low
  if (sensorValue < shadeThreshold){
    Serial.println("Sunlight is low!");
  }
  else if (sensorValue < lightThreshold && sensorValue > shadeThreshold){
    Serial.println("Plant is in shade!");
  }
  else if (sensorValue < directLightThreshold && sensorValue > lightThreshold){
    Serial.println("Plant is in sunlight!");
  }
```

```
  else if (sensorValue > directLightThreshold){
    Serial.println("Plant is in direct sunlight!");
  }

    delay(1000);

    int moistureValue = analogRead(moisturePin);
    Serial.println("Water level: ");
    Serial.print(moistureValue);
    Serial.println(" ");
    if (moistureValue <= moistThresholdDown){
      Serial.println("Dry, water plant!");
    }

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

    delay(500);

    tempReading = analogRead(tempPin);

  Serial.print("Temp reading = ");
  Serial.print(tempReading);     // the raw analog reading

  // converting that reading to voltage, which is based off the reference voltage
  float voltage = tempReading * aref_voltage;
  voltage /= 1024.0;

  // print out the voltage
  Serial.print(" - ");
  Serial.print(voltage); Serial.println(" volts");

  // now print out the temperature
  float temperatureC = (voltage - 0.5) * 100 ;  //converting from 10 mv per degree with
                                                 500 mV offset
                                                //to degrees ((volatge - 500mV) times
                                                 100)
  Serial.print(temperatureC); Serial.println(" degrees C");

  // now convert to Fahrenheit
  float temperatureF = (temperatureC * 9.0 / 5.0) + 32.0;
  Serial.print(temperatureF); Serial.println(" degrees F");

  delay(1000);

}
```

Testing:

For testing purposes, an Arduino was used to power the different sensors. The light sensor consisted of a photocell and 10 kOhm resistor connected to Vin and ground as shown above in the schematic. The Vout pin was connected to the Analog Pin A0 and the Vin was connected to 5V on the Arduino. Initially, the different thresholds values where the photocell was lacking sunlight, shaded, in sunlight, and in direct sunlight needed to be found. These values were read in from the analog 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 sunlight. Once these different threshold values were obtained, the Arduino serial monitor printed out what level of sunlight 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 5 V on the Arduino. The GND was connected to ground on the Arduino and the SIG connected to the Analog Pin A1. 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 Vout was connected to the Analog 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 and this temperature in both Celsius and Fahrenheit were displayed.

WiFi Application Subsystem

System Requirements:

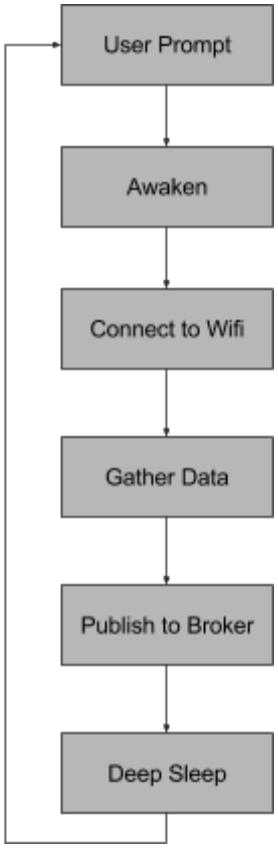
1. Publish information from sensor readings to MQTT broker
2. The ability to go into deep sleep and then awaken and publish necessary data.

WiFi Flow Chart:

The ESP8266 module is the bridge between the information and the user. It must be able to wake from a deep sleep, when required. After waking, the ESP must establish a WiFi connection. Once the connection is made, the device can take the sensor readings and the publish them to a specific topic that is match to the topic subscribed by the mobile app.

The two interfaces that the ESP will have are a WiFi connection to the mobile app and a physical connection to the three different sensors. Between the those two interfaces the ESP will be responsible for ensuring the proper data is available for the user. Arduino offers a simple addon for the ESP module and made the selection in programming language straightforward.

WiFi Flow Chart



There are a number of things outlined in the ESP8266 Thing Dev schematic. All of them do not pertain to this system. One critical part of the ESP and its scope in this project is the voltage regulator. The voltage regulator on the ESP keeps all I/O pins around 3.3V. In doing this, some of the sensors that will be used must be shifted from 5V. (Schematic seen here)

The code for the ESP is straightforward. After initializing, the module connects to the internet. After a connection is established it attempts to connect to the MQTT server, and continues to try and connect. Upon a successful connection to the MQTT broker, the data is then published.

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#define wifi_ssid "ND-guest"
#define wifi_password "YOURPASSWORD"
```

```
#define mqtt_server "senior-mqtt.esc.nd.edu"
#define mqtt_user ""
#define mqtt_password ""
#define topic1 "t1dj"
#define topic2 "t2"
WiFiClient espClient;
PubSubClient client(espClient);
void setup() {
  int finish = 0;
const int  sleepTimeS = 30;
    Serial.begin(9600);
    setup_wifi();
    client.setServer(mqtt_server, 1883);
    /*while (PostToPhant() != 1)
  {
    delay(100);
  }*/
  //digitalWrite(LED_PIN, LOW);
  // deepSleep time is defined in microseconds. Multiply
  // seconds by 1e6
  /*ESP.deepSleep(sleepTimeS * 1000000);
  Serial.print("Delay ended");*/
}

void setup_wifi() {
    delay(10);
    // We start by connecting to a WiFi network
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(wifi_ssid);
  //  WiFi.begin(wifi_ssid, wifi_password);
    WiFi.begin(wifi_ssid);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("");
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
```

```
    Serial.println(WiFi.localIP());
}
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 {
            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" try again in 5 seconds");
            // Wait 5 seconds before retrying
            delay(5000);
        }
    }
}
//NOTE: if a user/password is used for MQTT connection use:
//if(client.connect("TestMQTT", mqtt_user, mqtt_password)) {
void pubMQTT(String topic,float 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 finish = 0;
long lastMsg = 0;
float t1 = 75.5;
float t2 = 50.5;
void loop() {
  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)
        t1 = t1>90 ? 0 : ++t1;
        t2 = t2>90 ? 0 : ++t2;
        //Publish Values to MQTT broker
       // pubMQTT(topic1,t1,true);
        // pubMQTT(topic2,t2,);

        return;
    }
}
```

Testing:

In order to test the ESP, Arduino and MQTT.fx were used. Arduino served as the programming platform that was used connect to the internet, generate test data, and then publish it to the MQTT broker. MQTT.fx was used in order to test that published data could be subscribed to and matched to the values being generated by the test code.

Mobile Application Subsystem

System Requirements:

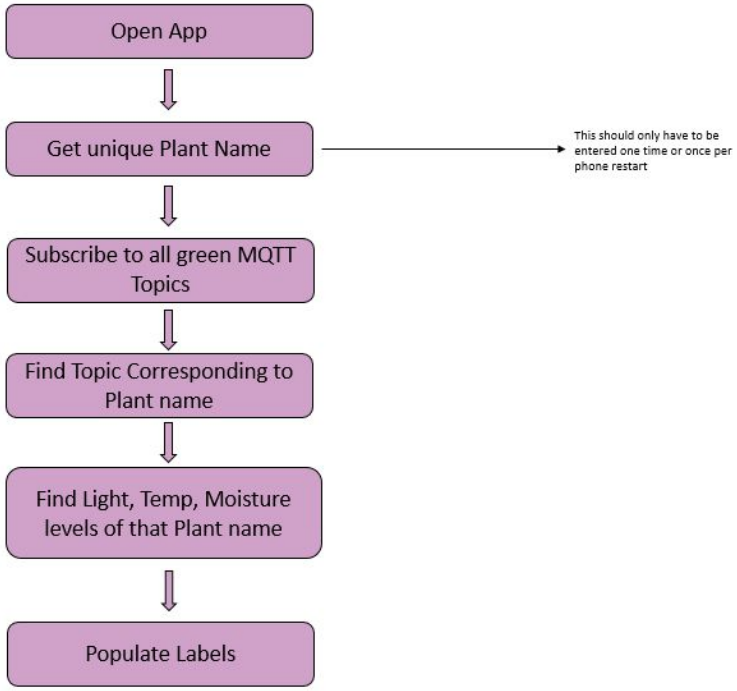
1. Get information via MQTT Protocol from the sensors
2. Notify user at critical values
3. Connect to specific gardens and plants
    i. The garden information will not have to be entered every time the app is open.

App Flow Chart: The flow of information in the Mobile App can be found in the diagram below. When the application is opened for the first time the user will enter the unique garden name. After this the user can add specific plant names to start building their Smart Garden. Once the Garden Name has been entered it should not have to be re-entered unless the iPhone memory is damaged in someway. The user will then click “Enter your Smart Garden” this will bring them to a new view that shows the information collected by the sensors. The app will also send a push 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 the Garden Name and Plant Name topics will be found and the values for light, temperature, and moisture will populate the labels. During the initial setup of the hardware, in addition to inputting the

WiFi network name and password you will also input a “Garden Name” and “Plant Name” to associate with that specific Smart Garden Device. This step has to be done after choosing a “Garden Name” through the app or else there is a potential for duplicate “Garden Names”. Currently because of permissions associated with our developer account the Alert feature is not working. We are also unable to save the Garden Name.

App Flow Chart



Software: The code for the app can be found in a .zip folder on box.nd.edu at this link.

Testing: The Xcode simulator was used to test the functionality of the app. To test the MQTT components in the app MQTT.fx was used to publish and subscribe to topics to ensure the app was correctly publishing and subscribing.

Outer Casing:

The outer casing will be made of laser cut pieces of clear plastic attached together to form a water resistant box. The moisture sensor will be exposed and the prongs of the sensor will be used to keep the Smart Garden electronics attached to the plant by stic.

Power System Requirements:

Since an Arduino was used for testing purposes, the power system requirements will be more finalized by the next design review. 5 V is needed for the sunlight and moisture sensor to be powered and 3.3 V for the temperature sensor. In our final design, the Smart Garden will use 4 AA batteries to power the device.