Final Report
Electrical Brain Surgeons

Joshua Sy

Jose Perrone

Steven Waller

Jean-Pierre Vertil

Andre Bermudez-Perez

April 29, 2016

EE 41440

Table of Contents

Introduction

Problem & Solution

Smart grids are electrical grid networks designed to provide energy in a reliable and efficient way. Smart grid networks are considered the wave of the future due to their reliability, but there are still some challenges that tend to impede their widespread adoption. One of the biggest challenges is determining how to correct the power factor for better energy flow in complex smart grid systems. Often times in complex grid systems it is hard to monitor and thus improve upon the efficiency of the grid. The purpose of the project was to develop a solution to this challenge by creating a device that measures power and power factor at a node in a smart grid network and make this information available via the Internet through the wireless transfer of incoming data. This will enhance the process of pinpointing exactly where improvements in the network need to be made. For example, if the measured power factor is below 1, a capacitive load needs to be added to the system to balance out the effects of the inductive reactance at the node. Vice versa, if the power factor is above 1 an inductive load needs to be added to balance out the effects of the capacitive reactance at the node.

Outcome

While the team was able to successfully demonstrate power measurements and send this information to a broker via wifi - with the use of the ESP8266 Dev Thing as a master device and an MCP39F521 Demo Board for 1 phase- we were unable to get readings with our final board that incorporates 3 MCP39F521 chips (slave device) and a ESP12 (master device that functions similarly to the ESP8266 Dev

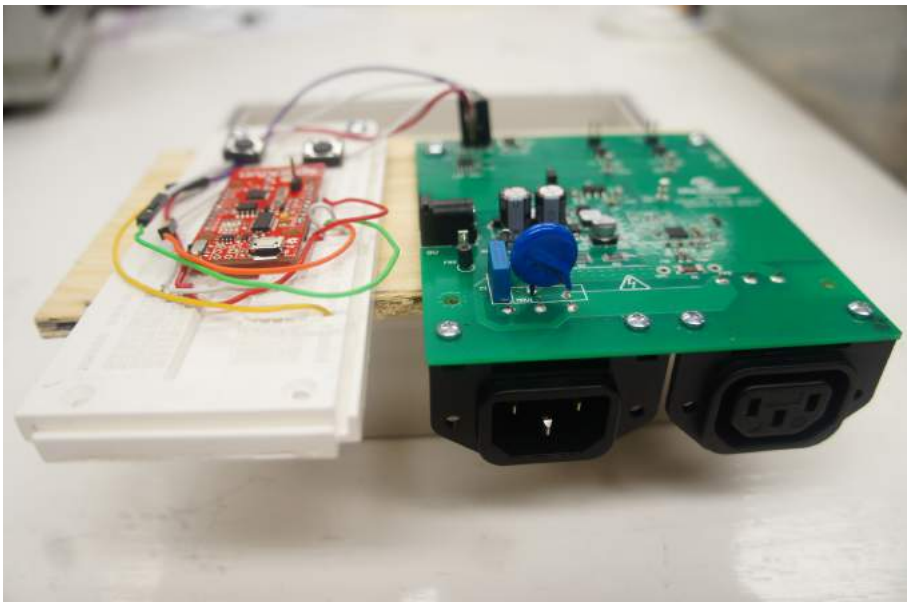

Figure 1 - Hardware of Power Monitor System combining the ESP8266 Dev Thing and the MCP39F521 Demo Board

While we could not get readings from our designed board, it properly distributes voltage and the ESP12 is programmable and performs well if it does not need to interact with the MCP chips.

Possible Sources of Error

We were able to verify that all the MCP chips were powered properly. In order to verify if the I2C communication was taking place properly, we looked at the Clock and Data signal with the use of a logic analyzer. As seen in **Figure 2**, the logic analyzer suggests that there was no proper I2C communication. The pattern seen in the figure is infinitely repetitive.

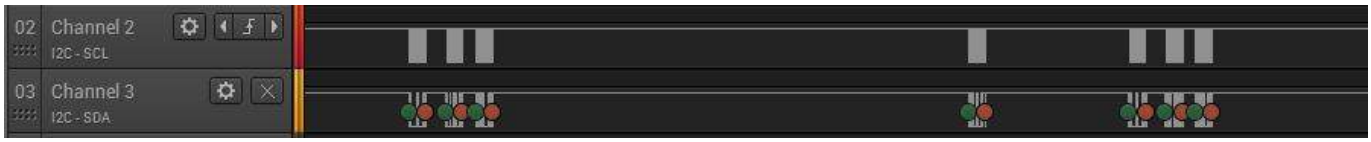

Figure 2- Logic Analyzer Output of non-functioning Board

As were were able to verify the functioning of the code with the MCP39F521 Demo Board, the issue must be at the hardware level and is likely due to a shortening of some wires or pins during the soldering of the board.

Detailed System Requirements

In order to achieve the desired solution, the system must be able to read in voltage, current, and power factor from a node on the grid. Therefore, a printed circuit board with schematics for a voltage reading system, a current reading system, and power factor computing is required along with three MCP chips and the ESP8266. Three MCP's are needed to measure three phase power.

The system behaves as seen in the flowchart displayed in **Figure 3**. In essence, current and voltage values are inputted for 3 different phases, the signals are read, digitalized and published to a broker via WiFi.

Powering

The board is powered with the use of a DC wall wart not exceeding 7V DC. This fed voltage goes through a voltage regulator that outputs 3.3 V to the rest of the board.

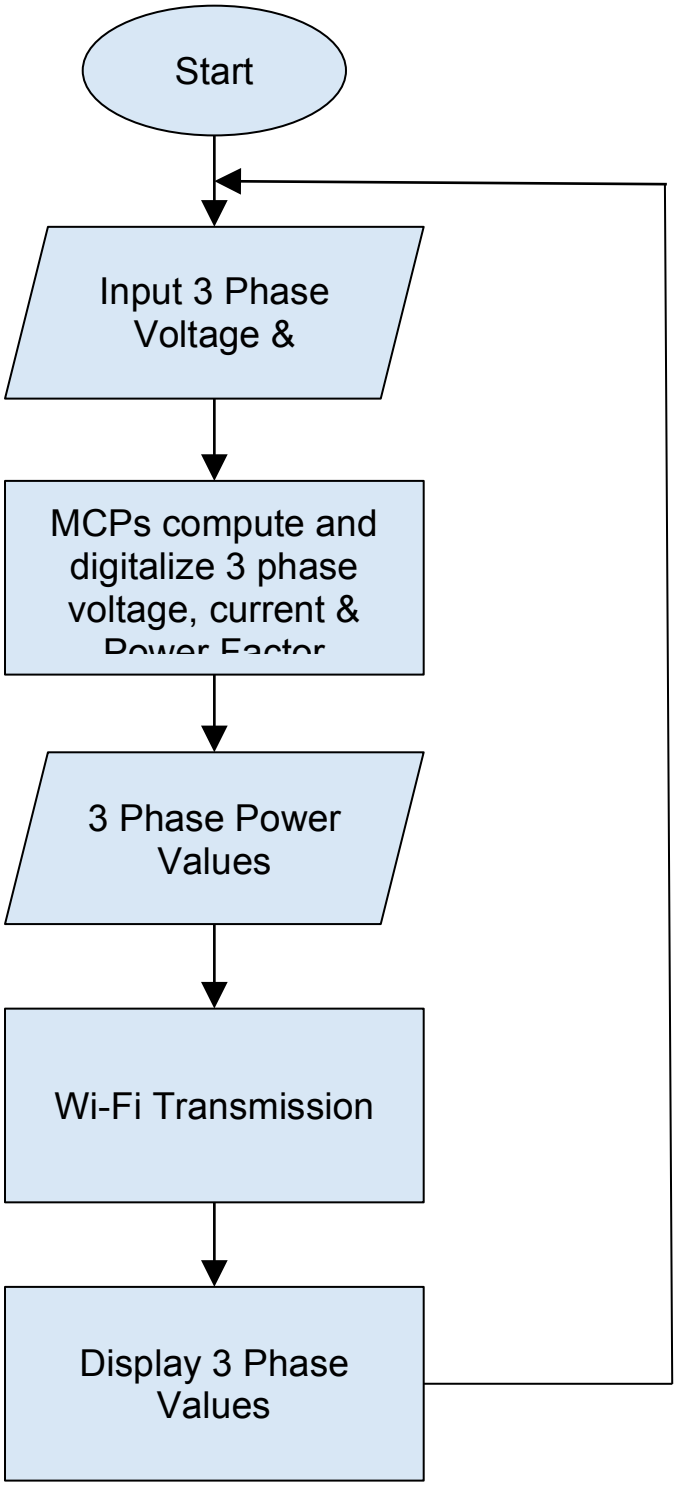

Figure 3- System Flowchart

Voltage Measurement:

Considering the fact that we are dealing with a 3-phase power system, each voltage phase is brought to an individual MCP chip. A common neutral line is also inputted into the MCP which does the measurement.

Current Measurement:

Current is measured through the use of a current sensing transducer. The current transducer generates a voltage which is sent to the MCP for appropriate current measurement.

Power Factor Measurement:

Through inputs of voltage and current as previously described, the MCP chips calculates the power factor for each of the three phases.

Wireless Component:

The current, voltage, and power factor measurements must be transmitted to a server via Wi-Fi.

Detailed Project Description

System Theory of Operation:

The overall system works as following: The system will test any node on a smart grid by measuring 3 phase power factor, voltage, and current at the corresponding node. The values will be continuously read in and transmitted to a server via Wi-Fi to determine inefficiencies on the grid network.

System Block Diagram:

The overall system consists of two subsystems that work together to achieve the proposed solution (See **Figure 4)** . The first subsystem reads voltage, current and power factor. The second subsystem is the Wi-Fi system. The two subsystems communicate to each other via I2C.

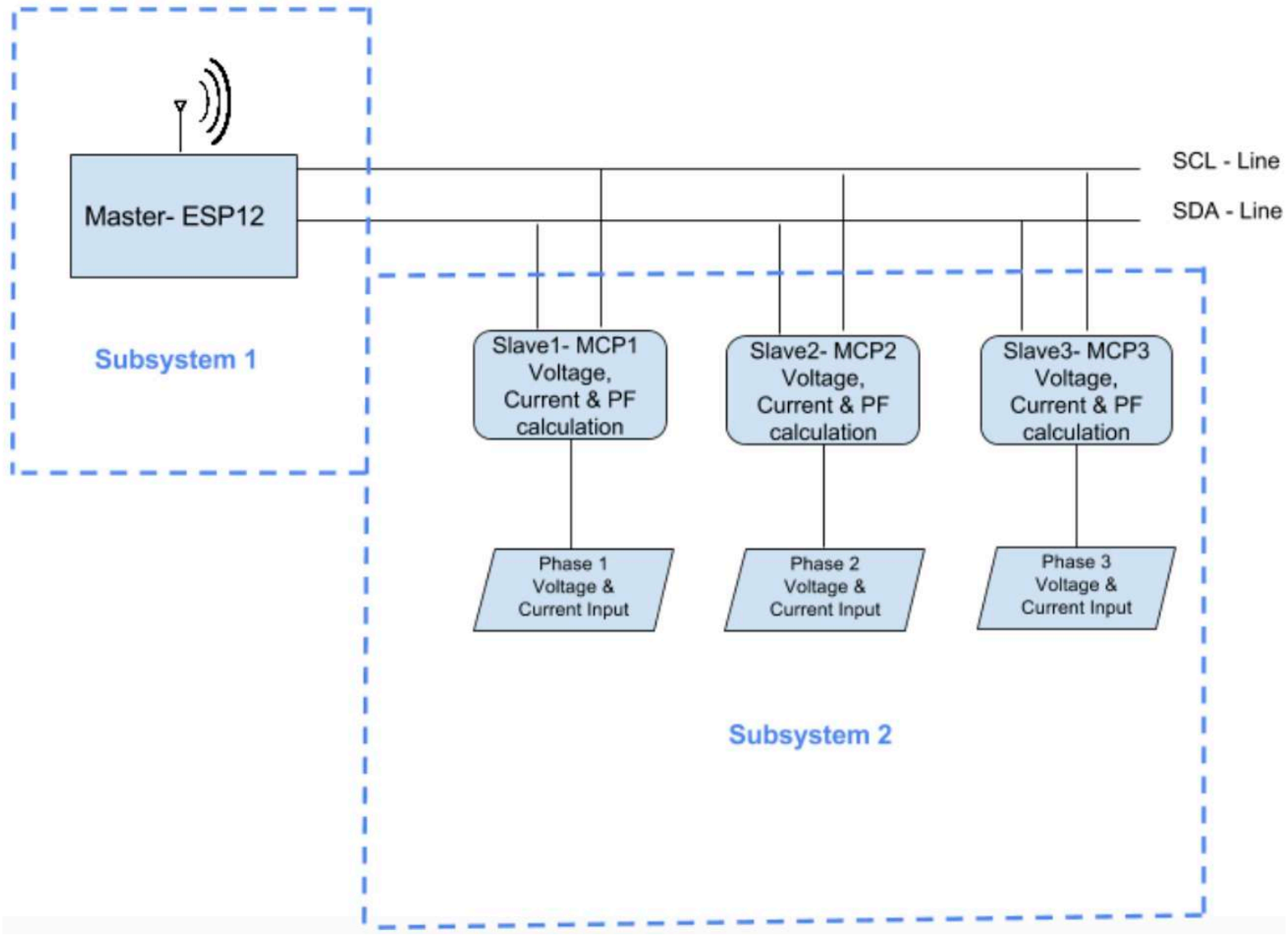

Figure 4- System Block Diagram

The MCP39F521 Power Demonstration Board is a device that measures voltage, current and power factor at a node. The schematics from the MCP device are replicated on the printed circuit board (see **Figure 5 of the Appendix**) and three MCP chips will be placed in the corresponding positions on the board.

Note that the Event and Zero Crossing detector circuits seen in **Figure 5 (Appendix)** were omitted on our design and they are not necessary for our purposes. We also simplified the power input on the board. Instead of transforming the input voltage being measured from AC to DC, we have a wallwart connection to feed the DC components of the board and input the AC input voltage for the sole purpose of being measured (see **Figure 6** in the Appendix**)**.

Subsystem 1 consists of the voltage, current and power factor measuring system. The MCP will effectively measure these three parameters. A current transducer sensor is placed around a wire at the test node. Using the two ends of the current sensor, a voltage will be passed across a burden resistor and through the MCP to measure current. **Figure 7** (Appendix) shows the current measuring schematic. Note that we changed the displayed burden resistor with a 27 Ohm resistor, determined in the following manner: Burden Resistor (ohms) = (Reference Voltage * # of turns in the current sensor) / ($2\sqrt{2}$ * max primary current) = 29.17. The reference voltage is 3.3 volts and there are 3000 turns in the current sensor. We used 27 because it is the closest common resistor value to 29.17. The max

primary current is determined by the current range of the sensor, which is 0-120A. Therefore, the max primary current is 120 A.

An ESP8266 module is used to connect our second subsystem to the MQTT broker, which will subscribe/publish the data. The broker subscribes to three topics: Voltage, Current, and Power Factor. Figure 4 shows a flowchart of the entire system. The two subsystems will communicate using an I2C interface.

Subsystem 1 (Hardware)

Subsystem 1 must be able to measure voltage, current and power factor at any node on a smart grid network. **Figure 8** (Appendix) shows a schematic of the voltage reading system of the MCP. Subsystem 1 will use the MCP39F521 device to measure the voltage and power factor at a node on a smart grid and read in the voltage from the device with the microcontroller using the I2C interface. The necessary components are the MCP chips, the implementing voltage reading schematic on the printed circuit board, and the proper I2C interface. The I2C interface allows the ESP device, which transmits data over a wireless network, to communicate and receive readings from the MCP chips. Figure 4 shows a flowchart of subsystem 1.

Note that Subsystem the current measurement whose MCP current channel schematic is depicted in **Figure 7** is implemented on the printed circuit board with the difference that the printed board uses a current coil.

The current measuring component will:

- Place a current sensing coil around a load wire at the test node
- Run the current across a burden resistor

Components:

- Current Sensor.
- Resistors for the voltage divider.

Wifi Component

Subsystem 2 (Hardware)

The Wifi subsystem will:

- Collect the data from the measurements on the microcontroller
- Use the ESP8266 device to transmit the data to a web server

Component:

- ESP12

Subsystem 1 & 2 (Software)

While the MCP chips offer calibration options through the Microchip Power Monitor software, they are not programmable per se as they do not require code to change their behavior.

This being said, an arduino code was written to gather the power information read by the MCP chips and communicate this information with the ESP12 before being transmitted via WiFi.

The code that enables the interaction between the two subsystems is generated as seen in the Program Flow Chart displayed in **Figure 9**. For simplification and practicality purposes, the program is broken down into multiple functions. A full listing of the code is available in the Appendix with extensive comments (**in blue**) for guidance.

*In order to use the WiFi functions of the ESP12, the PubSubClient.h and the ESP8266WiFi.h libraries had to be included in the program. In order to enable the I2C communication between the ESP12 and the MCP chips, the Wire.h library had to be included in the library.

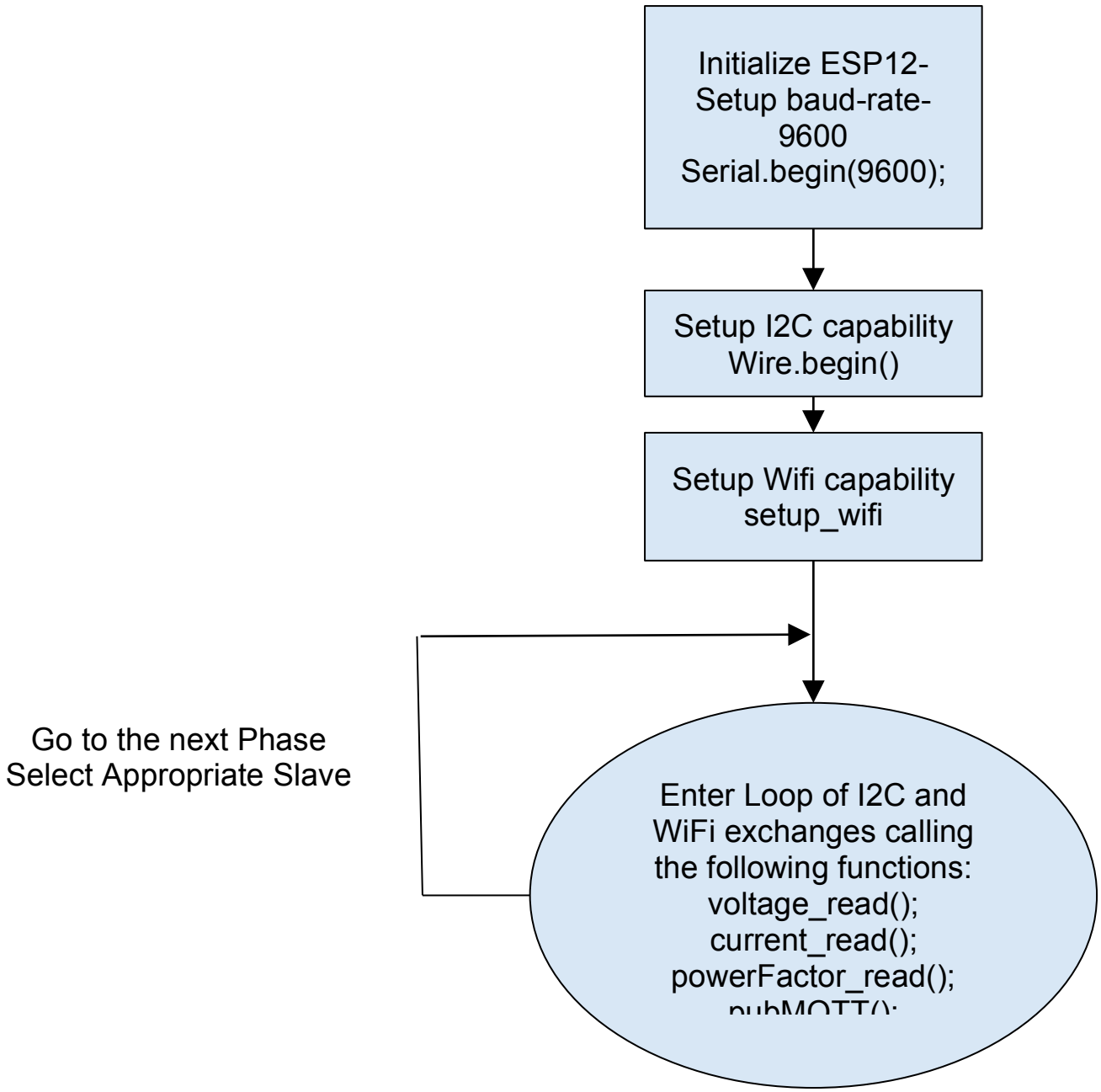

Figure 9- Program Flow Chart

System Integration Testing

Each subsystem was separately tested, then the full system was tested as a whole. In order to test the first subsystem of voltage, current and power factor readings, we communicated with MCP chips via I2C and used a logic analyzer to make sure that the proper bytes were being returned and that the acknowledges were being set appropriately.

In order to make sure that the values returned on the I2C channel were correct, we also tested the first subsystem with the Microchip Power Monitor software that is meant to display the readings in question. We were also able to verify the current and voltage readings by measuring the voltage and current in question with the help of a multimeter.

The second subsystem- wiFi- was tested by sending a known variable with expected updates to the broker. The values were displayed as expected.

Finally, when assembled as one system, we were able to confirm that the readings from the Arduino serial monitor were the same appearing on the online broker as seen in **Figure 10**.

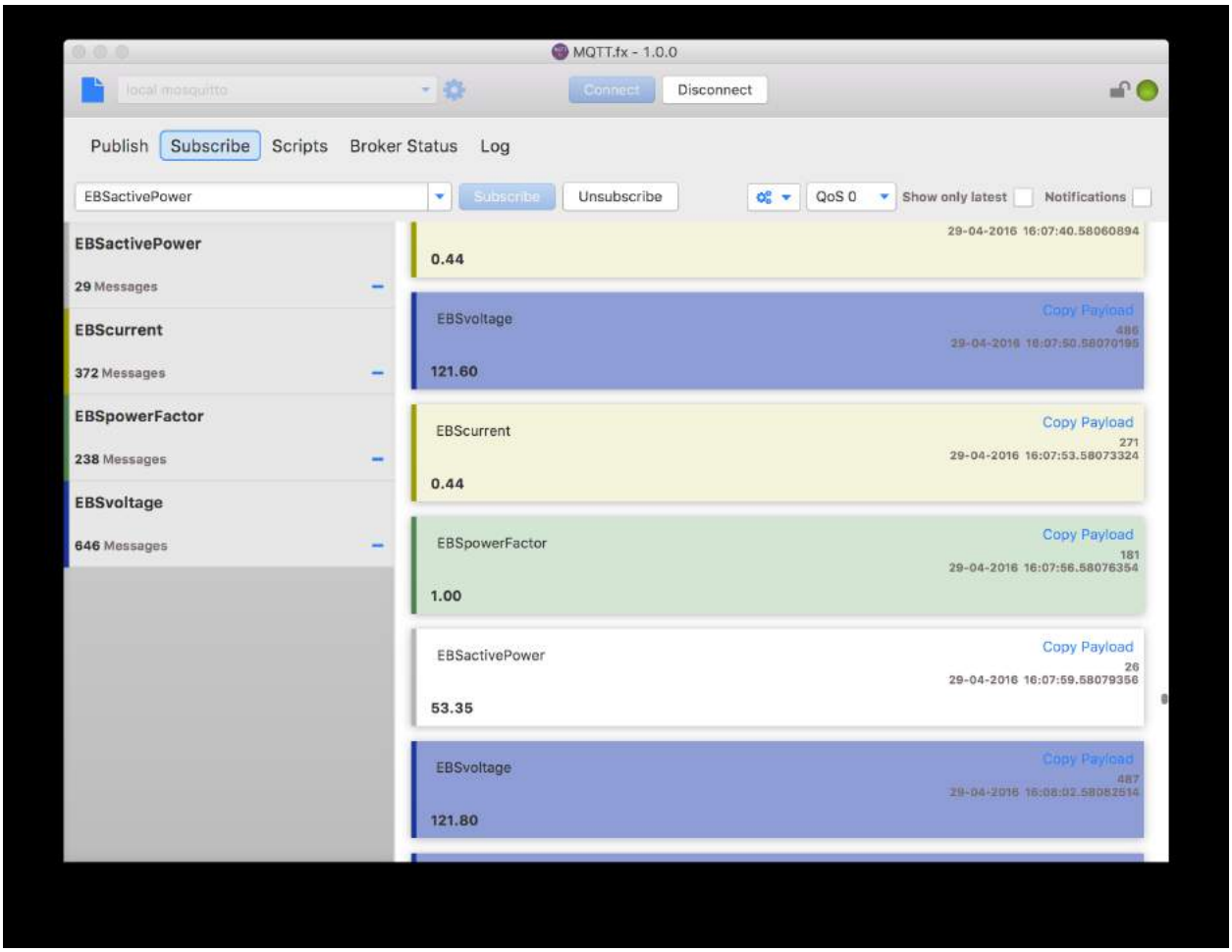

Figure 10- Proper Readings Displayed on the MQTT Broker

Recall that the right readings were only possible on the demo-board as our printed board is not functioning properly. As mentioned earlier, the I2C channels of the printed board were monitored but the readings were not detected. We also made sure that there was not short between Vdd and GND in the different locations where Vdd could be tested.

User Manual/Installation

The final product can be seen in **Figure 11**. It is equipped with a programmer that enables the user to do the following:

- Enter the name and password to a local SSID.
- Modify the time interval at which the power information are being read from the appropriate MCP registers.
- Modify the time interval at which the power information is being sent to the broker.
- Choose which phase to be monitored.
- Choose which power information to be measured and displayed.

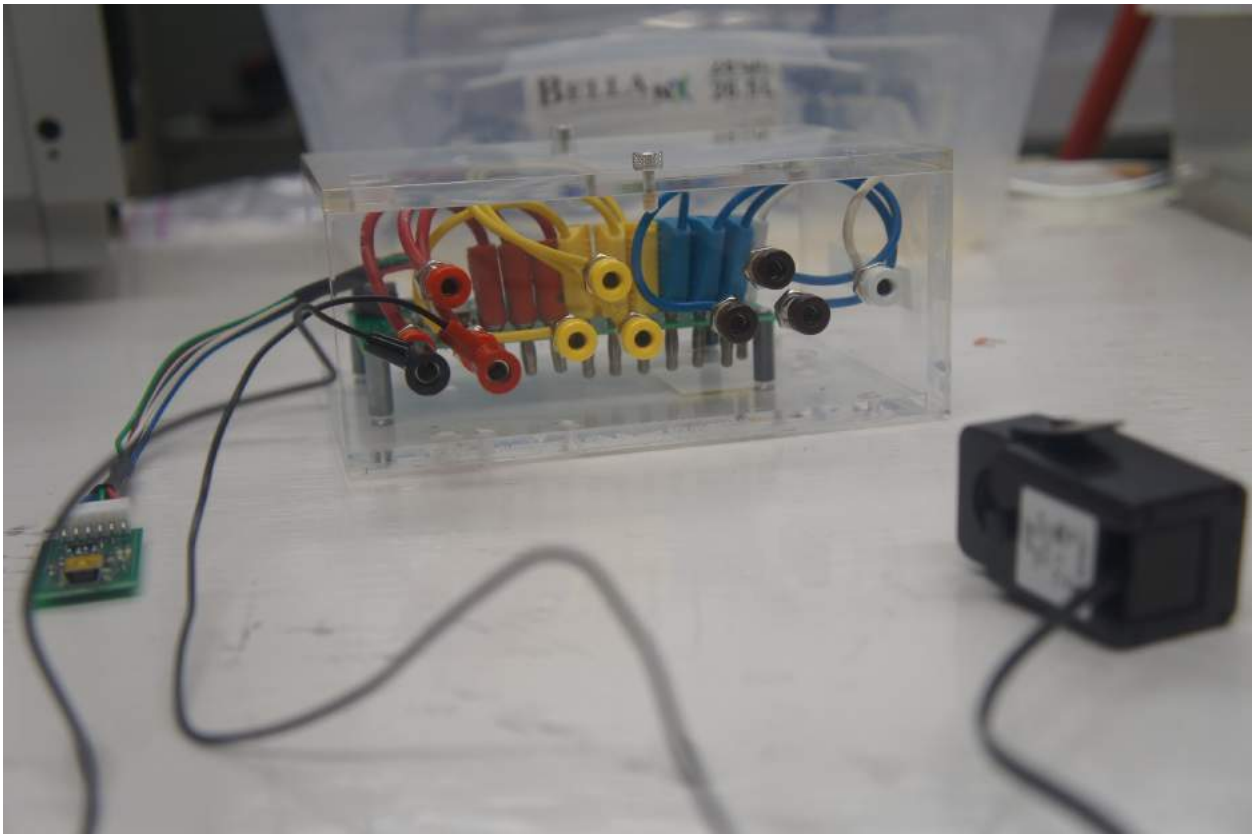

Figure 11 - Final Power Monitor Product

The device is also equipped with 3 current transducers for the current measurements. 1 Transducer is displayed in Figure 11 above. In order to properly connect the device at a particular node, the following banana plug connections must be done:

- Refer to **Figure 11.**
- Note that each phase is represented by a different color. E.g.:
    - Phase A: Red
    - Phase B: Yellow
    - Phase C: Brown
- The top banana plug in each phase is a voltage input.
- The bottom 2 banana plugs in each phase are for the current transducer out.
- The white banana plug is for the neutral line of the 3 phase voltage system.

Also note that the device must be powered by a DC wallwart made visible in **Figure 12**. The power supplied must not exceed 7V DC.

In order to verify that the Wifi reading are being displayed properly, the user is recommended to double check the readings on the broker with those on the serial monitor of Arduino. A simple check for the right output of power is to plug in a light bulb, a purely resistive load, which should have a known power factor of 1. The input voltage will be 120 V if plugged into a wall.

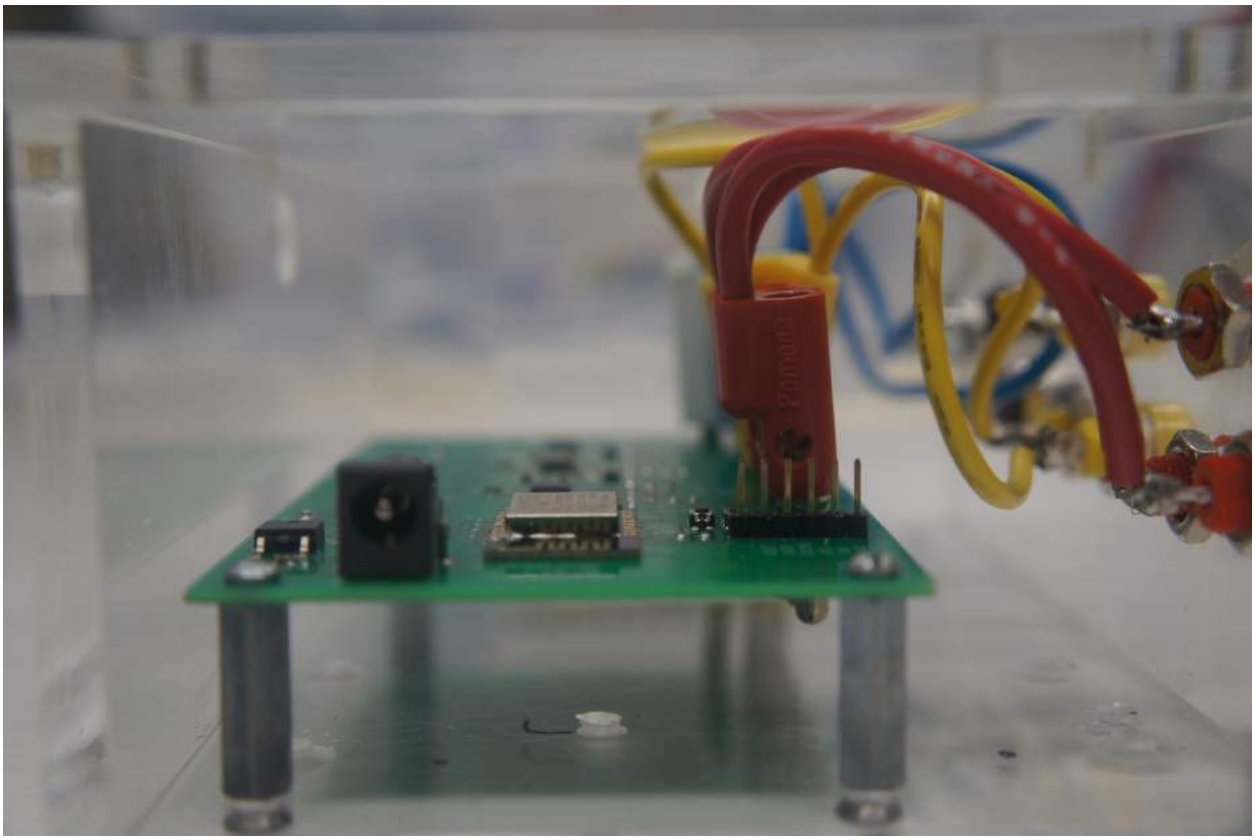

Figure 12- Wall wart connector

To-Market Design Changes

Because only a functional prototype has been created for this project, some changes need to be implemented before the device can be sold commercially. The

package size needs to be reduced, and should be fully closed- as of now the side on which the power in inputted is open.

In terms of the software, the delay times must be adjusted so that the readings from each register is appropriately stored. As of now, sometimes the output return a “0” as some readings are not appropriately stored.

For greater safety, there should also be an on-off switch to power the board. As of now, in order to stop powering the board, the wall wart must be disconnected altogether. Also for safety, the banana plug metal that go into the board should not be exposed to avoid any potential short.

Conclusions

Although the final product was not able to perform as expected, we were able to monitor power measurements (voltage, current and power factor) for single phase loads using the demo boards. This being said, the team has learned much about the programming of microcontrollers, the implementation of I2C communication, the implementation of Wi-Fi communication, the design and assembly of electronic boards and power monitoring.

Appendices

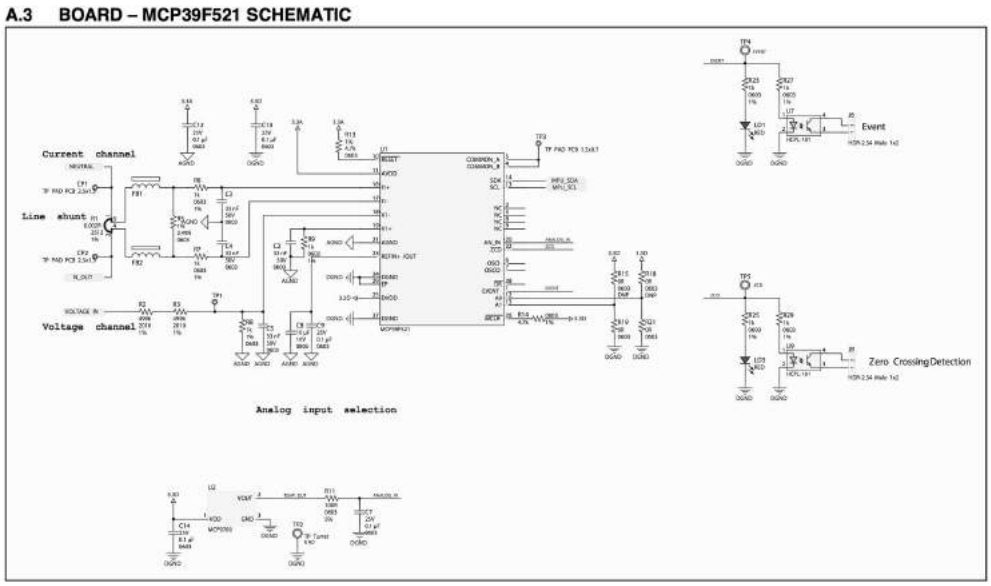

Figure 5- MCP Schematic

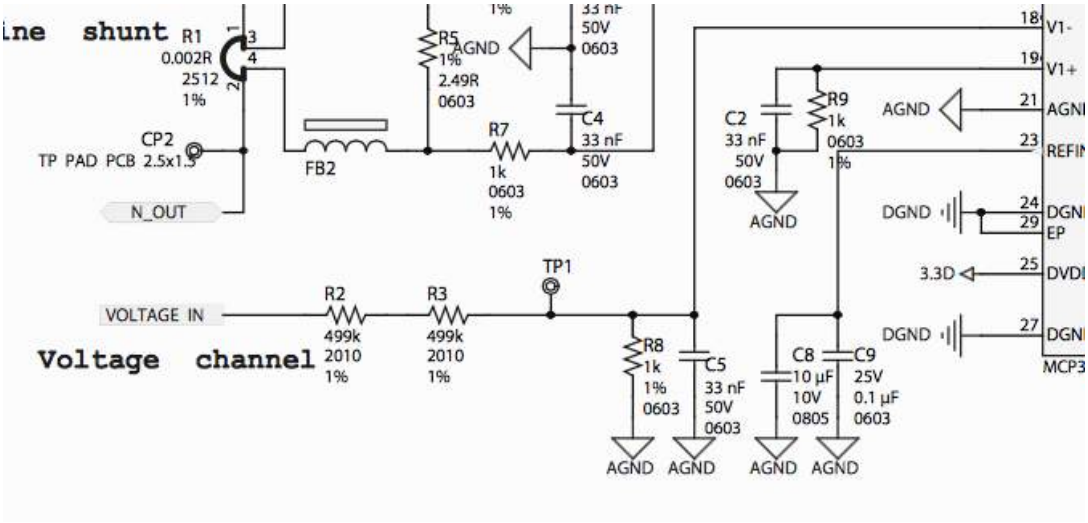

Figure 8- Voltage Channel Schematic

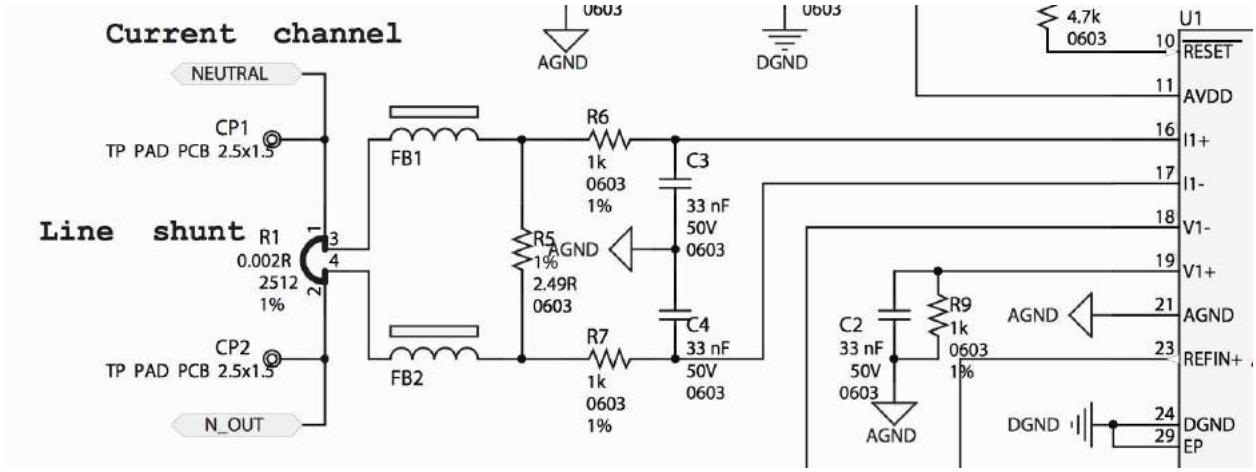

Figure 7- Current Channel Schematic

Hardware Schematics

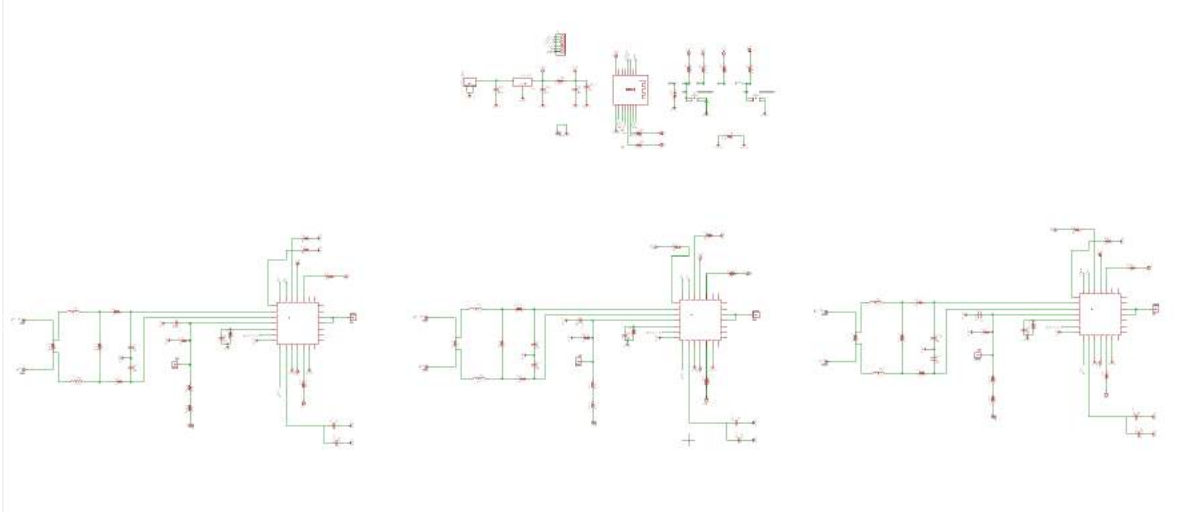

Figure 13- Full Eagle Schematic

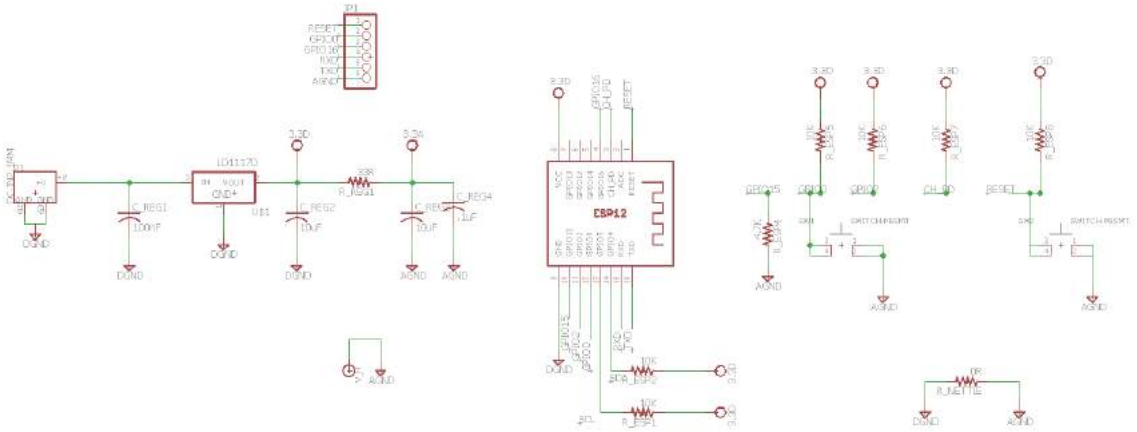

Figure 14- ESP12 Eagle Schematic

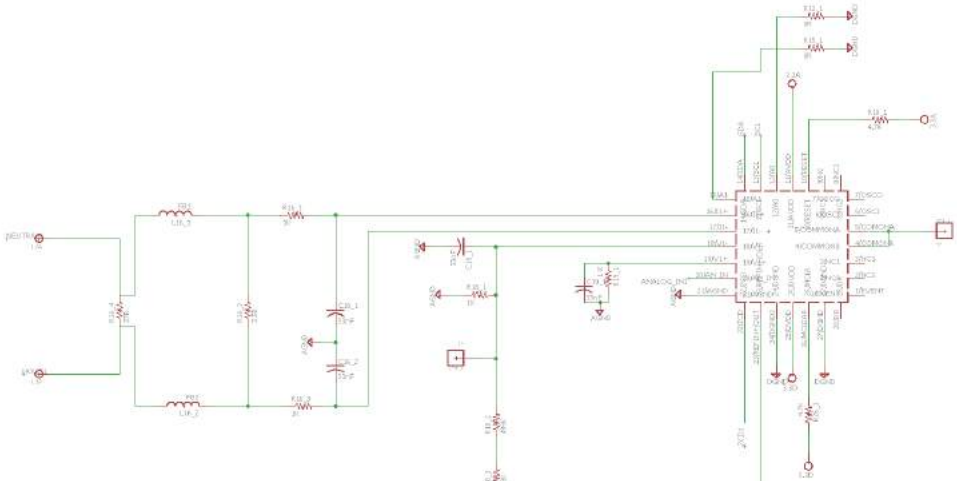

Figure 6- MCP Slave 1 Eagle Schematic

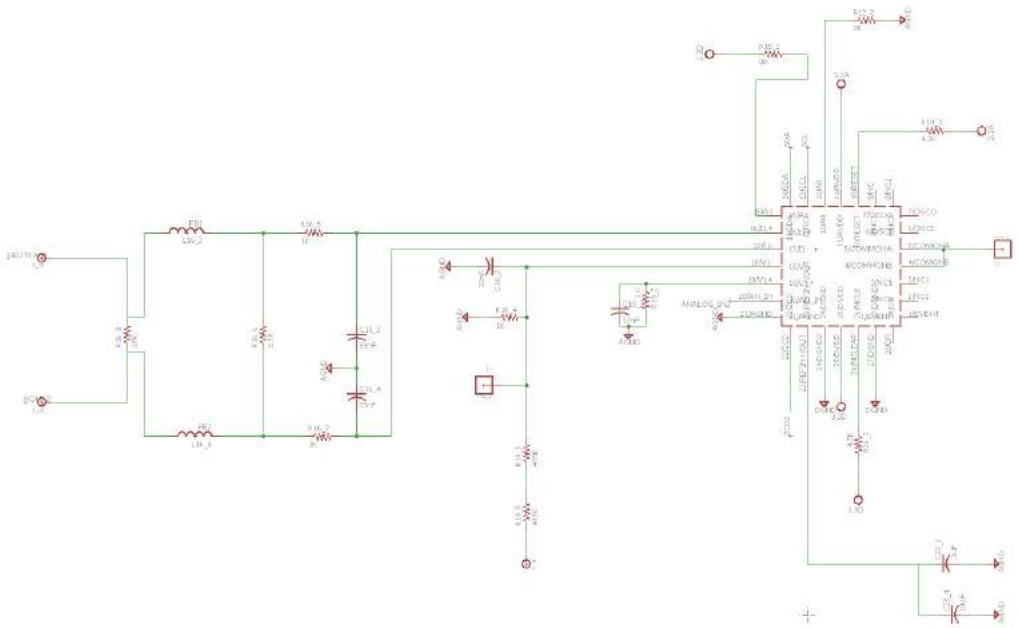

Figure 15- MCP Slave 2 Eagle Schematic

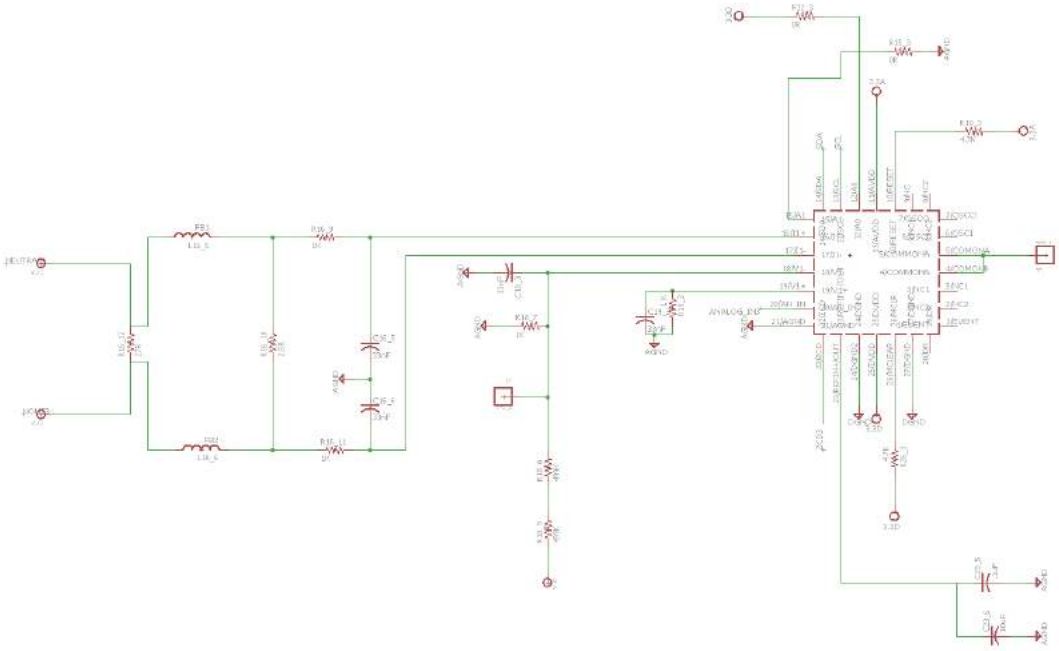

Figure 16- MCP Slave 3 Eagle Schematic

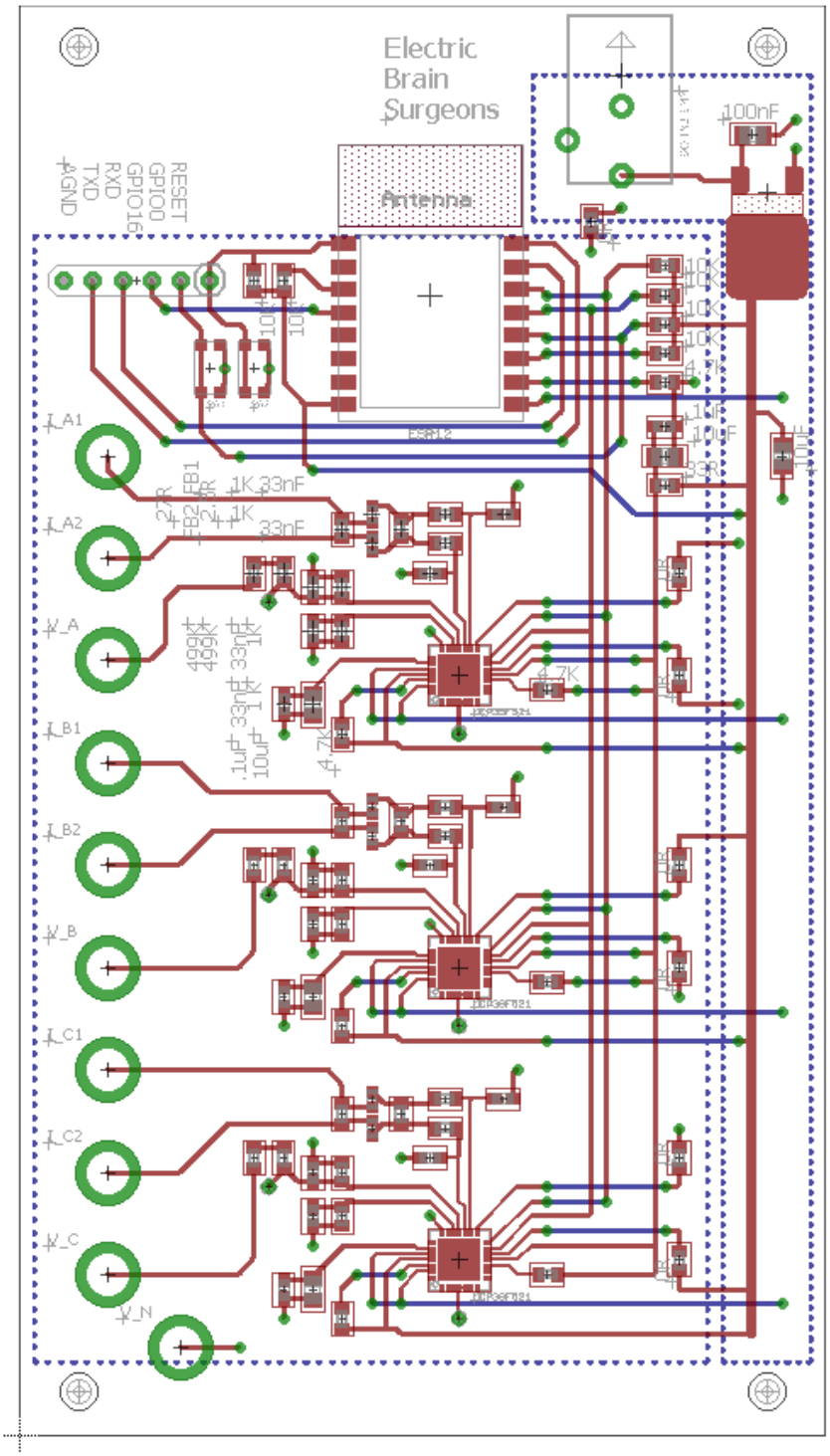

Figure 17- Full Eagle Board

Code Full Listing with Extensive Comments

Begin by including the appropriate libraries and global variables.

#include <PubSubClient.h> **(This library is needed for the WiFi capability)**

#include <ESP8266WiFi.h> **(This library is also need for the WiFi capability)**

#include <Wire.h> **(This library is needed for I2C communication)**

#define ADDRESS1 0x74 **(Address of first slave device)**

#define ADDRESS2 0x75 **(Address of second slave device)**

#define ADDRESS3 0x76 **(Address of third slave device)**

#define wifi_ssid "ND-guest"

#define mqtt_server "senior-mqtt.esc.nd.edu"

WiFiClient espClient; **(This function establishes the ESP12 as a client)**

PubSubClient client(espClient);

//---

void InsertProtocol(byte registerAddress, byte checksum) **(This function is a protocol that needs to be sent to the MCP devices to indicate that the ESP12 master device will read from them)**

{

 Wire.beginTransmission(ADDRESS);

 Wire.write(0XA5); //165 **(Header Byte)**

 Wire.write(0x08); //8 **(Number of Bytes in Frame)**

 Wire.write(0x41); //65 **(Command -Set Address Pointer)**

 Wire.write(0x00); //0 **(Address High)**

 Wire.write(registerAddress); **(Address Low)**

 Wire.write(0x4E); //78 **(Command- Register Read, N bytes)**

 Wire.write(0x20); //32 **(Number of Bytes to Read)**

 Wire.write(checksum); **(Checksum)**

 Wire.endTransmission();

```
}

//-------------------------------------------------------------------------------------------------------------------
```

double MultiplicationCombine(unsigned int x_high, unsigned int x_low) **(This function concatenates 2 bytes into 1 double value)**

```
{
 int combined;
 combined = x_high;
 combined = combined*256;
 combined |= x_low;
 return combined;
}

//-------------------------------------------------------------------------------------------------------------------
```

float voltage_read(byte slaveAddress) **(This function reads voltage from a particular slave MCP- note that the MCP address is one of its inputs. The current and power factor functions are similar and will thus have fewer comments)**

```
{
 InsertProtocol(slaveAddress, 0x06, 0x62);
```

(to input the right protocol, indicate the appropriate slave address, register address and checksum).

```
 delay(1);
```

(Now store each byte being read into an array initially populated by 0s).

```
 byte c[35] = {0};
  Wire.beginTransmission(ADDRESS);
  Wire.requestFrom(ADDRESS,35);
    for (int i=1; i<36; i++)
    {
     if (Wire.available())
     {
       c[i] = Wire.read();
```

```
      }
    }
    /*  for (int j=1; j<35; j++)
    {
      Serial.println(c[j], HEX); // print byte  (For debugging purposes)
    } */
    byte holder4 = c[4]; (Note that the fourth and third bytes read store the actual value wanted. The first returned by is an acknowledge and the second is the number of bytes in the frame. Refer to the MCP39F521 spec sheet. )
    byte holder3 = c[3];
    double voltageRaw = MultiplicationCombine(holder4, holder3); (This function converts the bytes read into decimal values)
    double voltage = voltageRaw/10; (The decimal value read is a factor of 10 greater than the actual value- Refer to the MCP39F521 spec sheet).
    Serial.print("Voltage = ");
    Serial.print(voltage);
    Serial.print(" v");
    Serial.println(" ");
    Wire.endTransmission();
    return voltage;
}

//-----------------------------------------------------------------------------------------------------------------------

(This function reads current from a particular slave MCP- note that the MCP address is one of its inputs)

float current_read(byte slaveAddress)
{
  InsertProtocol(slaveAddress, 0x0E, 0x6A);
  delay(1);
```

```
byte d[35] = {0};
 Wire.beginTransmission(ADDRESS);



 Wire.requestFrom(ADDRESS,35);
   for (int i=1; i<36; i++)
   {
    if (Wire.available())
    {
      d[i] = Wire.read();
    }
   }
   /* for (int j=1; j<35; j++)
   {
    Serial.println(d[j], HEX); // print bytes (for debugging purposes)


   }  */
   byte holder4 = d[4];
   byte holder3 = d[3];
   double currentRaw = MultiplicationCombine(holder4, holder3);
   float current = currentRaw/10000; (The decimal value read is 10000 times greater than the
actual value- Refer to the MCP39F521 spec sheet).



    Serial.print("Current = ");
   Serial.print(current);
   Serial.print(" A");
   Serial.println(" ");
   Wire.endTransmission();
```

```
    return current;
}

//-----------------------------------------------------------------------------------------------------------------------
```

(This function reads power factor from a particular slave MCP- note that the MCP address is one of its inputs)

```
float powerFactor_read(byte slaveAddress)
{
 InsertProtocol(slaveAddress, 0x0C, 0x68);
 delay(1);
 byte c[35] = {0};
  Wire.beginTransmission(ADDRESS);
   Wire.requestFrom(ADDRESS,35);
    for (int i=1; i<36; i++)
    {
     if (Wire.available())
     {
       c[i] = Wire.read();
     }
    }
   /*  for (int j=1; j<35; j++)
    {
     Serial.println(c[j], HEX); // print bytes (for debugging purposes)
          }          */
   byte holder4 = c[4];
   byte holder3 = c[3];
   double powerFactorRaw = MultiplicationCombine(holder4, holder3);
   double powerFactor = 0;
```

```
    if (powerFactorRaw <= 32767)
    {
     powerFactor = powerFactorRaw/32767; (Convertion scheme as required by the MCP39F521 spec sheet).
    }
    else if (powerFactorRaw > 32767)
    {
     powerFactor = (powerFactorRaw - 65535)/32767;
    }
    Serial.print("Power Factor = ");
    Serial.print(powerFactor);
    Serial.println(" ");
    Wire.endTransmission();
    return powerFactor;

}

//-------------------------------------------------------------------------------------------------------------------
```

This function sets up the WiFi capability

```
 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 setup() { **(This function initializes the ESP12, sets up the baud rate, sets up the I2C and WiFi capabilities).**

```
 // put your setup code here, to run once:
 Serial.begin(9600);
 Wire.begin();
 delay(15);
 setup_wifi();
 client.setServer(mqtt_server, 1883);
}
//--------------------------------------------------------------------------------------------------------------------
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);
}
long lastMsg = 0;
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;

    (Publish Values to MQTT broker)
     // Phase 1
    pubMQTT("Phase ", 1);
```

```
double voltage = voltage_read(ADDRESS1);
delay(500);
double current = current_read(ADDRESS1);
delay(500);
double powerFactor = powerFactor_read(ADDRESS1);
delay(500);
pubMQTT(topic1,voltage);
pubMQTT(topic2, current);
pubMQTT(topic3, actPower);

// Phase 2
pubMQTT("Phase ", 2);
voltage = voltage_read(ADDRESS2);
delay(500);
current = current_read(ADDRESS2);
delay(500);
actPower = powerFactor_read(ADDRESS2);
delay(500);
pubMQTT(topic1,voltage);
pubMQTT(topic2, current);
pubMQTT(topic3, actPower);

// Phase 3
pubMQTT("Phase ", 3);
voltage = voltage_read(ADDRESS3);
delay(500);
current = current_read(ADDRESS3);
delay(500);
```

```
    actPower = powerFactor_read(ADDRESS3);
    delay(500);
    pubMQTT(topic1,voltage);
    pubMQTT(topic2, current);
    pubMQTT(topic3, actPower);
  }
}
```

Relevant Spec Sheets:

- Spec Sheet for the MCP39F521: http://bit.ly/24qIhC7
- User's Guide for the MCP39F521: http://bit.ly/1WwpySU
- Spec Sheet for the ESP12: http://bit.ly/23ffuhf

Most Relevant Source: The brains of the Electric Brain System team.