

# Submetering ECEB

---

By

Xiaolin Wu

Tingyu Wang

Xiaoyi Shen

Final Report for ECE 445, Senior Design, Spring 2020

TA: Evan Widloski

7 May 2020

Project No. 2

## Abstract

This report summarizes the process that our project has been constructed throughout the semester. We proposed to realize the idea about measure the real time power consuming in each room in our ECE building. We aimed to display the information on a webserver for anyone with access to view it at any time and the data will not lost after power down or internet disconnections. Our method involves using ESP32 [1] as local microprocessor for measuring and data uploading; EmonLib [2] for signal processing and power calculation; we also used Amazon Web Services(AWS) for data storage, and for plotting and data display, we used D3.js[3] for chart and AWS SDK for Javascript[4] for data query. Compared to other previous methods, our method cost less but still maintained good accuracy both at high load and low load situations.

# Contents

1	Introduction . . . . .	1
1.1	Problem and Solution Overview . . . . .	1
1.2	High-level Requirements . . . . .	1
2	Design . . . . .	2
2.1	Analog Front-end Design . . . . .	2
2.1.1	Voltage Measurement Circuit . . . . .	3
2.1.2	Current Measurement Circuit . . . . .	3
2.1.3	Emonlib . . . . .	4
2.1.4	Magnitude Calibration . . . . .	5
2.1.5	Phase Calibration . . . . .	7
2.1.6	Apparent Power Calculation . . . . .	8
2.1.7	Real Power Calculation . . . . .	8
2.1.8	PCB Design . . . . .	8
2.2	ESP32 Data Collection and Upload . . . . .	9
2.2.1	ESP32 Local WiFi Connection . . . . .	9
2.2.2	ESP32 Local Data Collection . . . . .	10
2.2.3	ESP32 Local Data Storage . . . . .	10
2.2.4	ESP32 Topics Publish . . . . .	11
2.3	Cloud Data Handling and storage . . . . .	11
2.3.1	AWS Lambda Topics Subscribe . . . . .	11
2.3.2	DynamoDb database . . . . .	11
2.4	Plotting and Data Display . . . . .	11
2.4.1	Web Page Display . . . . .	12
3	Design Verification . . . . .	13
3.1	Analog Front-end Design . . . . .	13
3.1.1	Voltage Measurement Verification . . . . .	13
3.1.2	Current Measurement Verification . . . . .	14
3.1.3	Power Factor Measurement Verification . . . . .	16
3.1.4	Apparent Power measurement Verification . . . . .	17
3.1.5	Real Power measurement Verification . . . . .	18
3.2	ESP32 Data Collection and Upload . . . . .	20

3.3	Cloud Data Handling and storage . . . . .	20
4	Cost . . . . .	21
4.1	Parts . . . . .	21
4.2	Labor . . . . .	22
5	Conclusion . . . . .	23
5.1	Accomplishments . . . . .	23
5.2	Uncertainties . . . . .	23
5.3	Ethical considerations . . . . .	23
5.4	Future work . . . . .	24
	Reference . . . . .	25
	Appendix A Requirement and Verification Table . . . . .	26
	Appendix B Repository for Project Codes . . . . .	28

# 1 Introduction

## 1.1 Problem and Solution Overview

Although our ECE building has the measurement of the total power generated by the solar panels on the roof and has its measure of total power consumption in the building, it still lacks the ability to track the power consumed by each classroom, lab, and office. So our problem to solve in this project is to design a system that is able to measure the power used in individual rooms in ECEB. Therefore, aiming at the need to solve the issue for detection and display of the power consumed in the room, our project includes several local sensing systems that are placed in different rooms as well as a web-server that communicates with all the devices through Wi-Fi and have it displayed on the desired place. From the aspects of customer benefits, our project could help our administrators gain a better understanding about the distribution of the power consumption in this building and thus benefit any of their further decisions for the building. Currently, there are devices that can measure the AC power consumption using Arduino which is a little bit more expensive than our choices: we build our communication systems based on ESP32s and we are using AWS as our cloud server, thus the cost of the whole project will be limited to a satisfactory state. Besides, if an Arduino is involved in the design, we would also acquire an external data transmission module which will make the project bigger and cost even more to build [5]. In addition, due to our projects' capability to measure both 3-phase and single phase electric circuits, it is thoroughly very marketable.

One thing needs to point out at the first stage is, due to the COVID-19 pandemic in United States at present, our school shuts down all the in-person instructions and we are prohibited from entering the lab to perform any test prior to our ordering of PCBs or any other real implementation that should have been come afterwards. Eventually, our professor has approved that we will cut the requirements a bit hence we can continue the essential logistics of our project. Right now, our mission is to measure a relatively milder objective: to have a single AC voltage from a function generator, use RC circuit to add phase in representing the current passing a shunt resistor and still have the power consumption displayed through the web server.

## 1.2 High-level Requirements

- Functions to create a phase difference in representing the actual situation where current and voltage may have a difference in phase.
- Ability to offload data (both real power and apparent power in Watts) for displaying and the data to web server at least 4 times per hour.
- Have enough local storage (up to 24 hours) of power measurement data

## 2 Design

On a larger picture of our design, we will have multiple local devices that can measure the voltage and current generated by the function generator and use the ESP32 as the local microprocessor to calculate the power consumption. After the data is calculated, we will utilize WIFI to communicate with AWS web-server and eventually the data is offloaded to a screen that can access the server and have the data displayed as we desired. The following paragraphs will explain in detail about our design by subsystems.

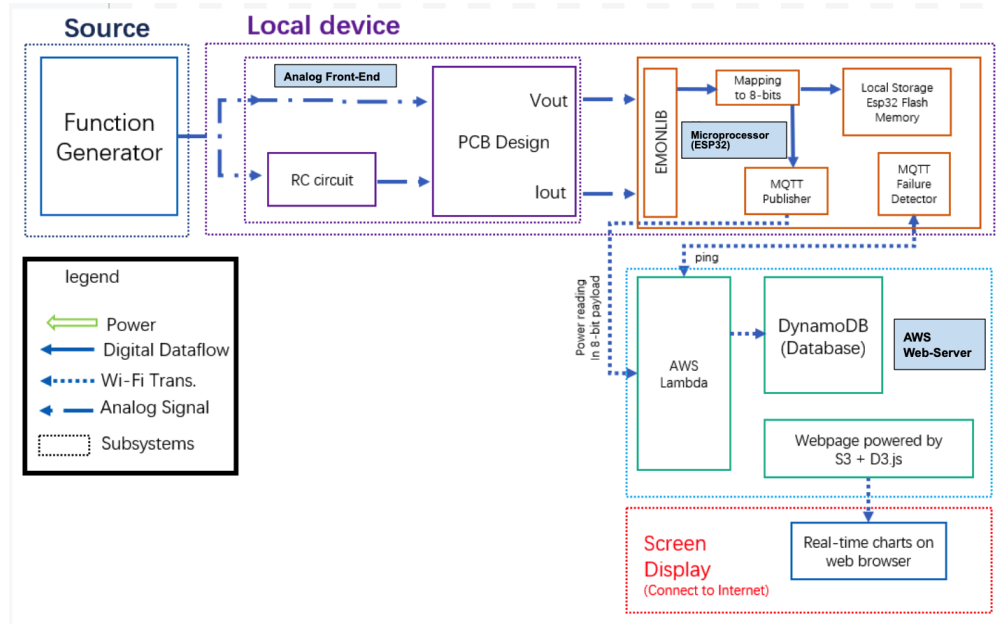


Figure 1: Block Diagram

### 2.1 Analog Front-end Design

Originally, our Analog front-end design was to step down the high voltage and current from room mains in ECE Building into a measurable rang (0.15 volts  $\sim$  3.1 volts) for our microprocessor, ESP32, to read. However, due to the COVID-19 pandemic and some safety issues, the analog front-end design changed from dealing with the high voltage and high current situations to measuring the voltage and current within a smaller range since we cannot derive the AC voltage and current from wall plugs in ECE building. Instead, a single-channel function generator was used to generate an AC voltage source to represent the amount of voltage that the microprocessor measured, and a Resistor-Capacitor circuit was used to generate a waveform with a phase shift and magnitude changes with respect to the waveform directly from the function generator to represent the amount of current that the microprocessor measured. On the microprocessor, the analog voltage signals for both the stepped down and then up-shifted voltage and the up-shifted voltage across the shunt resistor in representing the AC current is passed to the analog input pins. Using EmonLib, a method that will be discussed in detail below, five variables are calculated: real power, apparent power, root mean square voltage, root mean square current, and power factor.

### 2.1.1 Voltage Measurement Circuit

Originally, a voltage transformer with ratio 10:1 was used to step down the voltage ( $\sim 120$  volts) from room mains into a lower range ( $\sim 12$  volts). Currently, we swap the voltage transformer with a function generator and then feed the voltage from function generator directly into our circuit. A single-channel function generator was used to generate an AC voltage source to represent the amount of voltage that the microprocessor measured so our current voltage measurement circuit was to convert the voltage from function generator into the measurable range ( $0.15\text{v} \sim 3.1\text{v}$ ) for ESP32 to read. The voltage from the function generator corresponds to the voltage at the secondary side of the voltage transformer in our original design. The voltage divider circuit was used to lower the voltage further into  $\sim 1.65\text{V}$ . The ADC of the microprocessor, ESP32, starts the linear behavior at  $0.15$  volts and ends at  $3.1$  volts. Therefore, a level shift circuit, consisting of one capacitor and two resistors with one  $3.3$  DC bias, is needed to shift up the whole waveform into this range ( $0.15\text{v} \sim 3.1\text{v}$ ) for ESP32 to read.

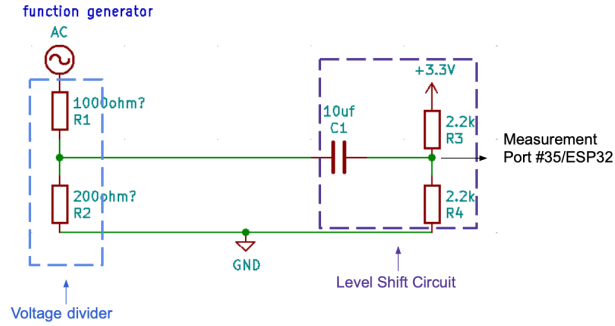


Figure 2: Voltage Measurement Circuit

### 2.1.2 Current Measurement Circuit

Originally, a current transformer with ratio 1:20 was used to step down the current into a lower range ( $\sim 5\text{A}$ ). Currently, a waveform with a phase shift and magnitude response with respect to the voltage waveform directly from the function generator was generated by a Resistor-Capacitor (RC) circuit to represent the current that the microprocessor measured, since we cannot derive the AC current from room mains. The ADC of the microprocessor, ESP32, starts the linear behavior at  $0.15$  volts and ends at  $3.1$  volts. Therefore, a level shift circuit, consisting of one capacitor and two resistors with one  $3.3$  DC bias, is needed to shift up the whole waveform into this range ( $0.15\text{v} \sim 3.1\text{v}$ ) for ESP32 to read.

- RC(Resistor-Capacitor) Circuit

A RC circuit was used to generate an waveform that has different phase and magnitude compared to the waveform directly from the function generator. The voltage at point Vi in Figure 3 corresponded to the voltage across the shunt resistor in our original design and it represented the amount of current that the microprocessor would measure.

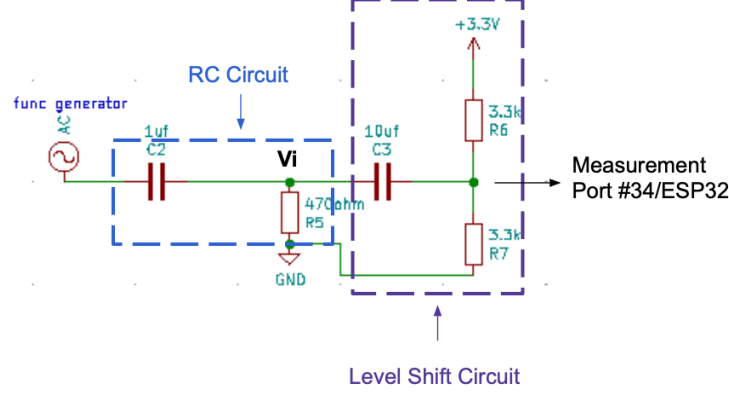


Figure 3: Current Measurement Circuit

The value of the phase shift that the RC circuit generated is determined by the resistance of R5, R6 and R7 in Figure 3 and the capacitance of C2 and C3 in Figure 3. Based on the AC analysis of the circuit in Figure 3, the transfer function is

$$\frac{V_{outc}}{V_{in}} = -\frac{C2 * C3 * R5 * R6 * R7 * w^2}{d} \quad (1)$$

which d is

$$d = d1 + d2$$

$$d1 = R6 + R7 - C2 * C3 * R5 * R6 * R7 * w^2 + C2 * R5 * R6 * w * i$$

$$d2 = C2 * R5 * R7 * w * i + C3 * R5 * R6 * w * i + C3 * R5 * R7 * w * i + C3 * R6 * R7 * w * i$$

As long as the resistance of resistors and the capacitance of capacitors in Figure 5 do not change, the value of the phase shift is a fixed number no matter how the  $V_{peak}$  in function generator is swept.

### 2.1.3 Emonlib

Based on our research online, we found a public library running in the Arduino environment that is used to monitor the AC power consumption of people's houses. [6] We looked into the code and tried to modify it to work on the ESP32 for our task. Ideally, we can use this library to read in analog inputs representing our voltage and current respectively and have the power calculated directly as output. Based on learnt knowledge, the formula for apparent power is

$$Pa = I_{rms} * V_{rms} \quad (2)$$

where  $P_a$  represents the apparent power and  $I_{rms}$  is the root-mean-square of the measured current in the loop and  $V_{rms}$  is the root-mean-square of the total supply voltage in the circuit. Hence, the real power we care about is calculated as



$$I_{rms} = \frac{I_{peak}}{\sqrt{2}} \quad (3)$$

$$V_{rms} = \frac{V_{peak}}{\sqrt{2}} \quad (4)$$

$$PowerFactor = \cos(\theta_v - \theta_i) \quad (5)$$

$$RealPower = I_{rms} * V_{rms} * power\ factor \quad (6)$$

The current and voltage are sampled at a rate of 5588 samples per second [7] which is sufficient to deal with a 60Hz original measure target as we have been told. ESP32 initializes the serial connection at 300 bits per second in order to get rid of the redundant outputs with the same meaning for us to record since our desired output is simply 2 representative samples per minute as required.

In the example using the EmonLib, in the line of calling the functions in EmonLib, the lines of codes are as simple as:

$$emon1.voltage(2, 234.26, 1.7); \quad (7)$$

$$emon1.current(1, 111.1); \quad (8)$$

where ‘emon1’ is the name for the library when imported; 1 and 2 refer to the analog read pin numbers; 234.26 and 111.1 are the calibration constant for voltage and current respectively; and 1.7 is the phase calibration constant. Due to the magnitude transforms that we apply to the voltages such as stepping down circuit in our original plan as well as the up-shift, ESP32 has to know the actual real numbers to give accurate measurement. Furthermore, since different computers have different processing delays, the phase of the read in values may also varies a bit. Thus, to recover the actual data in the field, these constants are necessary and they may vary under different working environment.

#### 2.1.4 Magnitude Calibration

The calibration constants can be categorized to two types, magnitude calibration constant and phase calibration constant since we needs to count for both the magnitude transforms and phase drift as well.

- Voltage Calibration

For the voltage calibration constant [8] , it is derived with the following strategies: Since we want to recover the initial alternating mains voltage, we want to derive a calibration constant that satisfies

$$voltage\ constant = alternating\ mains\ voltage \div alternating\ voltage\ at\ ADC\ input\ pin \quad (9)$$

Thus, in that example, the voltage calibration constant is derived by

$$\text{voltage constant} = 230 \times 11 \div (9 \times 1.20) = 234.26 \quad (10)$$

where 230 is the RMS Voltage of the room mains to be measured, 11 is the ratio the further step-down ratio of the circuit on the PCB after the transformer. The ratio for voltage transformer used in the example is calculated as

$$230 \div (9 \times 1.20) \quad (11)$$

as explained that “the voltage transformer output is nominally 9 V for 230 V input, but this is at full load. When used as a voltage monitor for the EmonTx, it is effectively running unloaded, and the voltage is approximately 20 percent higher (this is called the transformer “regulation” and the value depends on the design of the transformer. 20 percent is typical for this type and size).” [8]

Therefore, the voltage calibration constant for our design should be calculated in the idea of Equation 9 by calculating specifically the real transformer ratio and the step-down ratio on the PCB as well. As a result, we will find a constant that reveals the real voltage per voltage read at the analog pin to recover any processes we perform previously to lower the voltage to the measure-able range of ESP32.

In our project, the voltage is measured at a point after a voltage division on the PCB, so we simply needs a constant to account for this process only. With the calibrated constant, we will accurately have the Vrms value, that is the square root of the peak value from the function generator, printed out as result.

- Current Calibration

For the current calibration constant, since current cannot be wired directly to the analog read pins of the microprocessor, we need a “burden resistor” to change that into a voltage value. Therefore, the current calibration constant is calculated as

$$\text{current constant} = (\text{original current} \div \text{current after CT}) \div \text{shunt resistor} \quad (12)$$

Thus, 111.1 is calculated in the example as

$$\text{current constant} = (100 \div 0.050) \div 18 = 111.1 \quad (13)$$

where the current transformer used is 100A:50 mA and the shunt resistor value is 18 ohms.

In our project, since the shunt resistor we selected is 0.33 ohms, we modify the constant and let the current value matched up with the theoretical computed one, that is, the current flow through a measured voltage across a 0.33 ohm resistor.

To summarize this module, the magnitude calibrations are used mainly to recover the voltage and current transformers ratio, but in our design right now, since no transformers are involved, we simply needs to

calibrate once to make sure that the voltage is the RMS value of the function generator output and current equals the point we measure the voltage across the shunt resistor divided by the shunt resistor value. Later we will see in the verification section, that once the calibration is made, the accuracy is maintained while the function generator gives output at various levels.

### 2.1.5 Phase Calibration

The need for the phase calibration constant is that firstly, the current and voltage transformers have internal phase errors that may cause error even before the processing in software. The normal error for this situation is: 3.5 degrees at 225V, 4.5 degrees at 240V ,etc. And for the current sensor with a 15 ohms burden resistor is: 3.5 degrees at 3A, 2.5 degrees at 30A and 2 degrees at 60A [8]. Another issue needs to be concerned in the context of phase calibration constant is that in the software, the voltage and current are sampled in that order and conversion takes approximately 100 micro-seconds, thus the voltage is measured earlier than current by about 2 degrees for a 50 Hz system. Thus, derivation of this phase calibration constant is also necessary for the accuracy of our measurement. EmonLib have one line written as

$$phaseShiftedV = lastFilteredV + PHASECAL \times (filteredV - lastFilteredV) \quad (14)$$

where it created a shifted set of sampled data in a sense according to the figure below

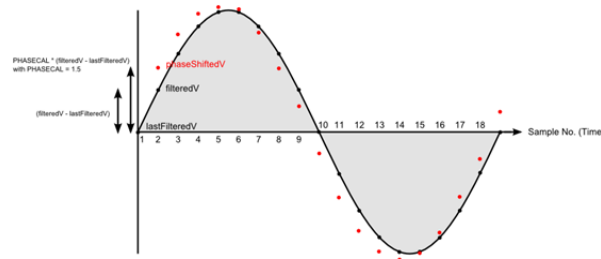


Figure 4: Phase Calibration [9]

The red dot is the shifted data by setting the phase calibration constant to 1.5. The first sample is read from the analogue input, filtered, and becomes 'filteredV'. At the start of the second loop, that first sample is copied to the program variable 'lastFilteredV'. The equation (14) puts the point phaseShiftedV above the filteredV by 1.5 times the vertical distance between this point and the previous point. In a way that a new waveform has been made that is shifted to the left we made it equivalent to a phase shift. Thus, in this manner, 1 would gives no shift in the phase, 0 would move the waveform to the right by one sample period and 2 would move the waveform to the left by one sample period. In reality, after calculating the voltage and current calibration constants, we set the phase calibration in a known test environment to adjust for the voltage and current transformers we selected for correcting the phase error generated by them as well as our own computer for correcting the processing time.

In addition, the reason why we only need phase calibration constant in the scope of voltage is that the phase difference between current and voltage are relative. There's no such thing as absolute phase difference. As a result, we only calibrate the phase so that the original phase difference is achieved so we only needs to modify the phase of the voltage.

### 2.1.6 Apparent Power Calculation

With the voltage and current measurement at hand, we could easily compute the apparent power of the circuit, which is based on the formula

$$P_{apparent} = V_{rms} \times I_{rms} \quad (15)$$

Thus, in calculation Emonlib has the sum of voltage and sum of current over the samples of a full cycle (determined by check zero crossings), take the average of each, and multiply the two averages together to be the apparent power of this cycle.

### 2.1.7 Real Power Calculation

For the real power calculation, Emonlib adopted the idea that "over a fixed number of cycles, the average of the instantaneous power of the sine wave is simply as" [10]

$$p = V \times I \times \cos(\theta) \quad (16)$$

with V representing the RMS voltage, I representing the RMS current and  $\cos(\theta)$  is the cosine of phase difference which gives the power factor of our measurement. Hence, using this idea, EmonLib has the instantaneous power calculated with every voltage and current sampled inputs, take the average in the end and this gives the real power of the measurement cycle.

$$P_{real} = \text{sum}P \div \text{numberOfSamples} \quad (17)$$

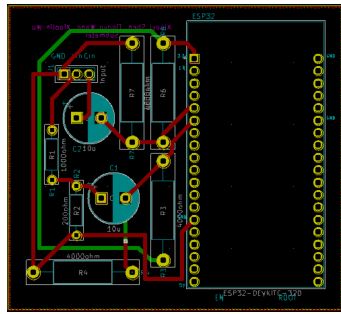
With the apparent power and real power calculated, the power factor can also be easily retrieved as

$$\text{power factor} = \text{real power} \div \text{apparent power} \quad (18)$$

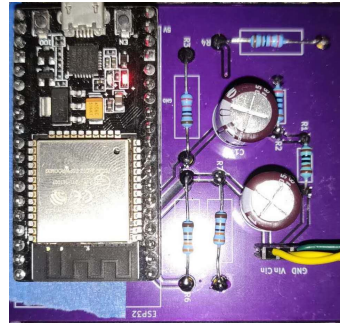
Thus, we have all the five main components we need for data uploading.

### 2.1.8 PCB Design

The voltage measurement circuit and current measurement circuit except the RC circuit are on our PCB Board.



(a) PCB Design



(b) PCB Board

Figure 5: PCB Diagram

## 2.2 ESP32 Data Collection and Upload

After the circuits have stepped down current and voltage to a desired level and Emonlib has processed the readings from the the input Pins, the microprocessor ESP32 will then process the data generated by Emonlib and store it in to the database. The display program will later query data from the database for real time plotting on the web browser.

The detailed code is provided in the appendix and it will not be elaborated in this section.

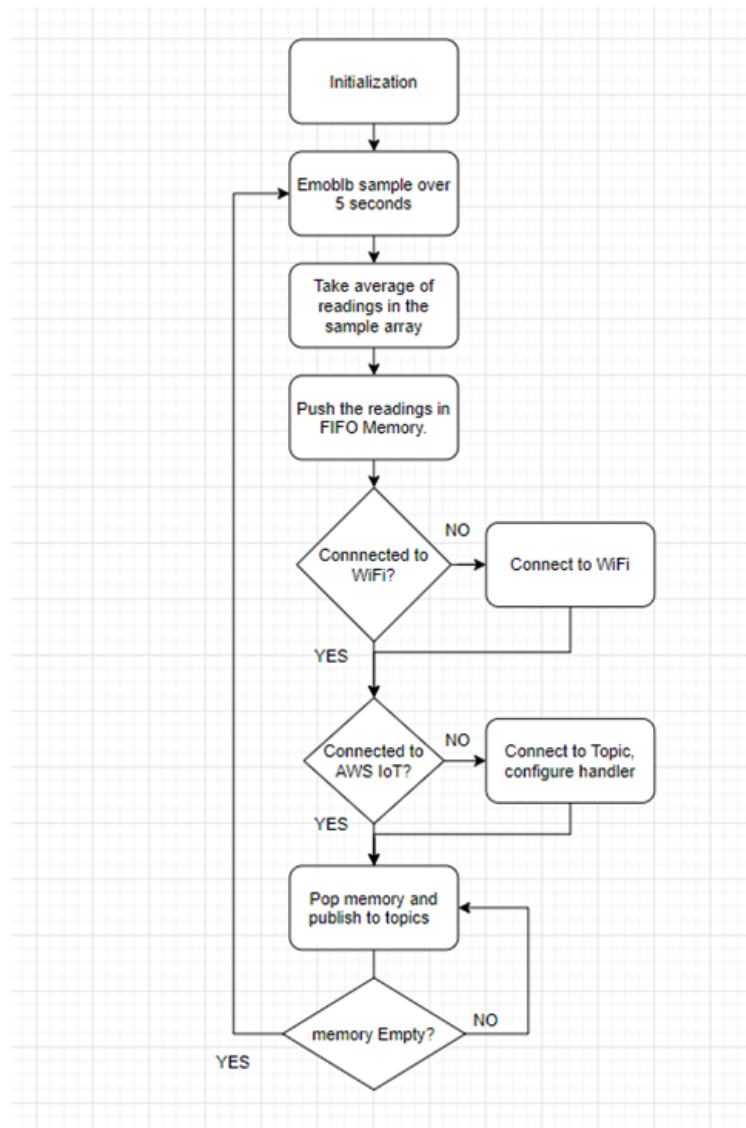


Figure 6: flowchart of the code on ESP32

### 2.2.1 ESP32 Local WiFi Connection

ESP32 needs to be connected to WiFi in order to upload any data to the database. The ability of ESP32 to connect and reconnect to the designated WiFi is critical to the completion of this project. Without the ability to connect to WiFi, ESP32 is unable to make the configuration that uploads the reading to the cloud. And without the ability to reconnect to WiFi, temporary shutdown of the access points or routers will stop

ESP32 from uploading in the future and must be reset manually to restore connection.

In order to avoid this, ESP32 will constantly check WiFi connection before uploading, and will automatically reconnect when disconnected.

In the presumed environment, the ECEB, ESP32 will connect to IllinoisNet Guest. The reason for this is that ESP32's library for WPA2 connection is very limited. IllinoisNet requires a password confirmation which never happened to be success in early experiment we conducted. While IllinoisNet Guest only check the IP address of ESP32, it is the most desirable network we could have with minimum expense. Of course, due to the shutdown of the ECEB during the pandemic, we change the network to our home router, but the idea is useful for broader application.

ESP32 comes with built in library WiFi.h that checks WiFi Connection. We use this to determine if ESP32 is connected. The code that implement WiFi connection and re-connection is in main.ino.

### 2.2.2 ESP32 Local Data Collection

In the design we required at least two uploads per minute. Since ESP32 has more potential in its computational power. We could collect the reading from Emonlib during the one minute period, then refine the data by average them, so that the two data we uploaded will be more representative of the minute.

We noticed that when ESP32 is plugged in with readings, the initial values is usually extremely large or small and takes few seconds to stable and consolidate, we intentionally remove the greatest and least value in the duration, then takes the average of the rest of the data. In this way, the mean will be less likely to be influence by extreme value and is more accurate.

During testing, a collection of around 12 data could be collected during 30 seconds period, this functionality is in function float glitch filtered sum in main.ino.

### 2.2.3 ESP32 Local Data Storage

The primary concern of data uploading is the safety of the data, for instance, under circumstances that the WiFi connection has been broken. The data must be able to be preserved locally and wait for the next success re-connection. Aiming to maintain the synchronous order, we implement an first in first out data structure to stores the data. In an other world, old data will be the first to be be uploaded when WiFi reconnected.

After the data is refined and taken average, the mean data that will be uploaded will be pushed into this structure immediately. And when the WiFi connection is confirmed, the oldest data will be popped and published to the cloud.

This idea is altered from the original idea in the design documents where we wished to write to flash memory and the read. The reason is the reduce the data reading into 8 bits reduced the significant figures. And after intense write and erase, one of the ESP32 chip we have is significantly slower that the other one. Assumed reason is that erasing data from flash memory is slow. So we abort this method and just store it in the heap memory of the code, where erase is not needed. The data structure is located in the library file "queue.h".

### 2.2.4 ESP32 Topics Publish

We implemented AWS MQTT[11] protocols in this project for data uploading. MQTT is frequently used in the industry of IoT due to its lightweight and its publish and subscribe structure suits our needs. ESP32 will first configure the endpoint, the topics and the certificates. Then the publish function will handle the rest by publishing the data we wrapped into Json format to the cloud, where the subscriber, which in our case is the AWS Lambda, will subscribe and grab the data. The code of this functionality is found in main.io.

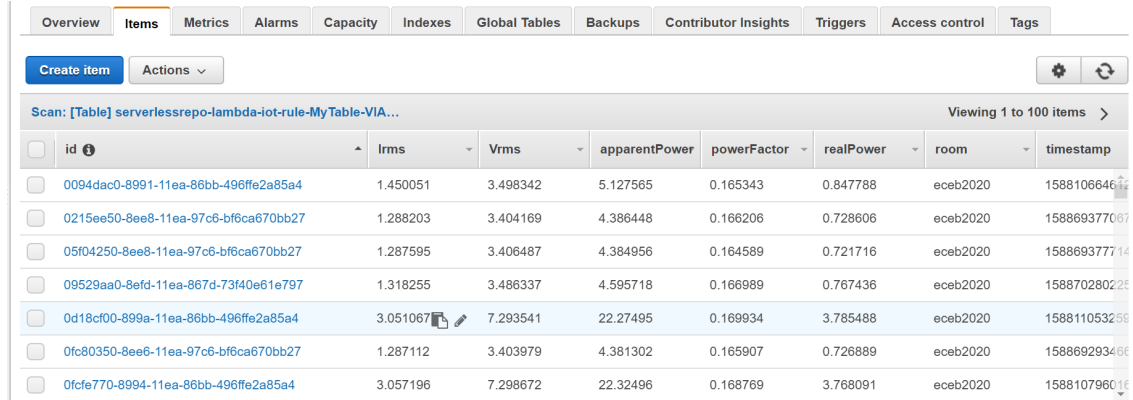
## 2.3 Cloud Data Handling and storage

### 2.3.1 AWS Lambda Topics Subscribe

AWS Lambda[12] is a platform that allow us to run codes on the cloud. We simply subscribe the same topic that we published to in the ESP32. With correct configuration, Lambda will catch the Json package, parsing it and store it into our database. We also add a timestamp during the parsing so that the database could sort the order by the value of timestamp. The code of AWS Lambda could be found in app.js.

### 2.3.2 DynamoDb database

DynamoDB[13] is the database that we used for storing the readings. It is a non-SQL data base that the attributes is set by the format of our input. We set this to be exactly the same as the output of the Emonlib. So we have apparent power, real power,  $V_{rms}$ ,  $I_{rms}$  and power factor. We set the parturition of the database by the attribute room, which is unique to each ESP32 to represent different classroom. And we sort the order of the entry by their timestamp.



id	lrms	Vrms	apparentPower	powerFactor	realPower	room	timestamp
0094dac0-8991-11ea-86bb-496ffe2a85a4	1.450051	3.498342	5.127565	0.165343	0.847788	eceb2020	158810664612
0215ee50-8ee8-11ea-97c6-bf6ca670bb27	1.288203	3.404169	4.386448	0.166206	0.728606	eceb2020	158869377067
05f04250-8ee8-11ea-97c6-bf6ca670bb27	1.287595	3.406487	4.384956	0.164589	0.721716	eceb2020	158869377714
09529aa0-8efd-11ea-867d-73f40e61e797	1.318255	3.486337	4.595718	0.166989	0.767436	eceb2020	158870280225
0d18cf00-899a-11ea-86bb-496ffe2a85a4	3.051067	7.293541	22.27495	0.169934	3.785488	eceb2020	158811053255
0fc80350-8ee8-11ea-97c6-bf6ca670bb27	1.287112	3.403979	4.381302	0.165907	0.726889	eceb2020	158869293466
0fcfe770-8994-11ea-86bb-496ffe2a85a4	3.057196	7.298672	22.32496	0.168769	3.768091	eceb2020	158810796016

Figure 7: Screenshot of DynamoDB

Figure 7 is the a screen shot of DynamoDB, it shows how the item is stored by attributes and sorted in the order of timestamp values.

## 2.4 Plotting and Data Display

### 2.4.1 Web Page Display

The code of Display is located in the folder Final Web, the idea of how it works is that we first generate seed data by grabbing the ten latest entries from the database. Then later upon new entry uploaded into the database, we fetch the new entry and update it on the chart.

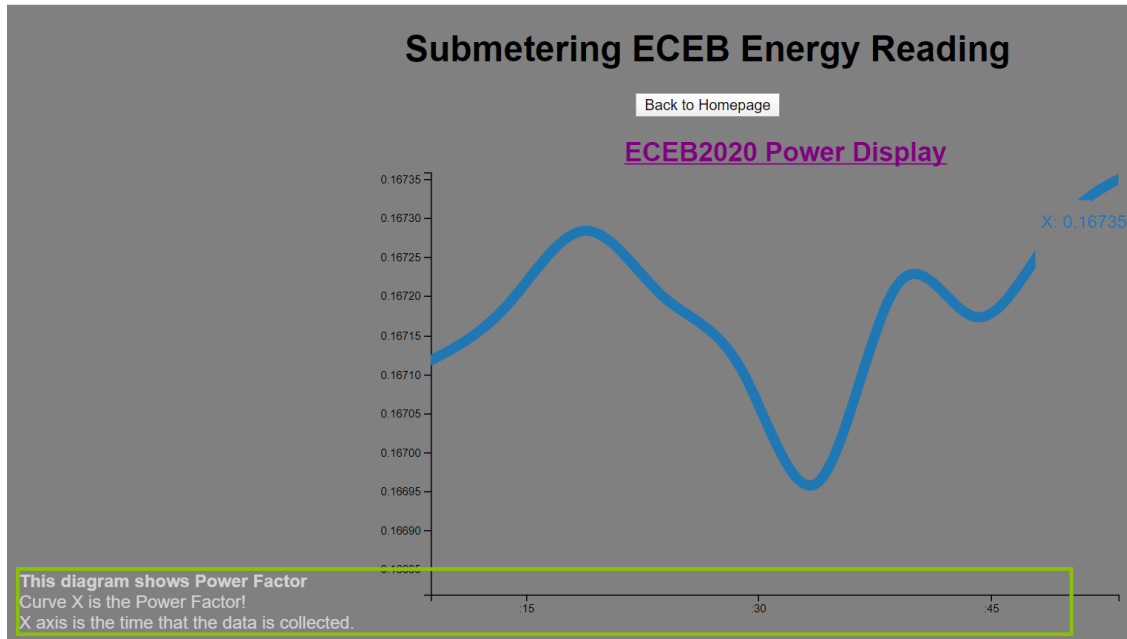


Figure 8: Screenshot of the display

The figure 8 above is the screenshot of the web page. We did not invest too much effort into the artistic view of the page but it certainly served our purpose of demonstrating the feasibility of our ideas.



### 3 Design Verification

#### 3.1 Analog Front-end Design

We tested four sets of data by setting the function generator to 20Vpeak-peak, 16Vpeak-peak, 10Vpeak-peak, and 1Vpeak-peak separately to sweep the voltage and current accordingly and read the root mean square voltage and current, power factor, apparent power and real power from our microprocessor, ESP32. The reason we chose these four sets of data is that these four sets of data almost cover the range of the amount of current (0A ~ 5A) we want to measure, which corresponds to 0A ~ 100A in our original design.

##### 3.1.1 Voltage Measurement Verification

Firstly, we will start from using simulation tools to visualize the wave for voltage that the function generator provides under four occasions. Four labels indicating the peak value in voltage.

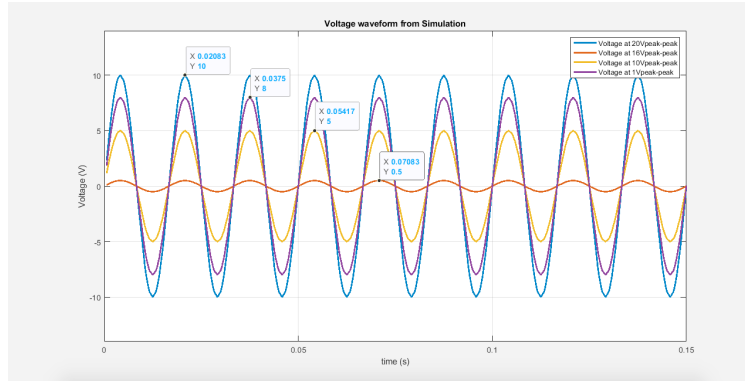


Figure 9: Voltage Waveform from Simulation

Based on the readings from above, we can calculate by hand the root mean square value of the data, and these values on the right of the table are the ideal values we want to measure and print.

Table 1: Vrms based on Simulation graph (Figure 9)

Amplitude of voltage waveform(V)	Vrms(V)
10	7.071
8	5.6568
5	3.535
1	0.3535

For the verification results, below is the collection of the printouts under four occasions. We collected twenty samples for each occasion for checking our accuracy.

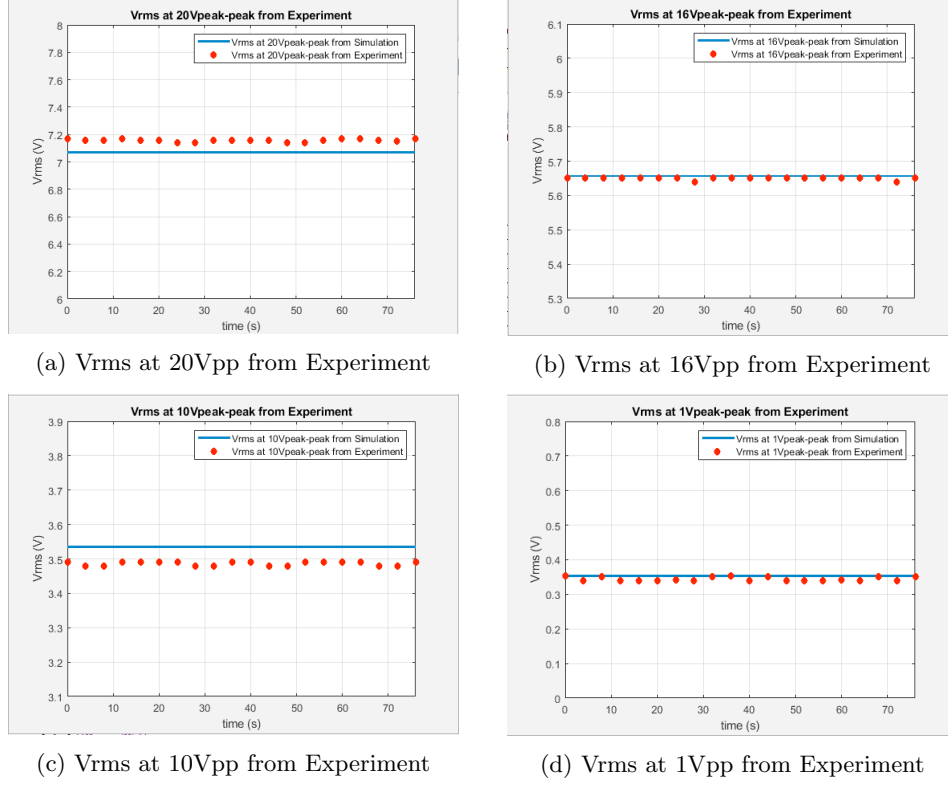


Figure 10: Vrms from Experiment

Apparently, based on the graphs presented, the accuracy of our result of our RMS voltage is pleasing and even though there are mild fluctuations, the result seems working fine from low voltage levels to high voltage levels.

### 3.1.2 Current Measurement Verification

Secondly, aside from the RMS voltage, RMS current is the other key component for us to calculate the power. Therefore, we did a simulation first and check if the values of our print-out console could match up with each other.

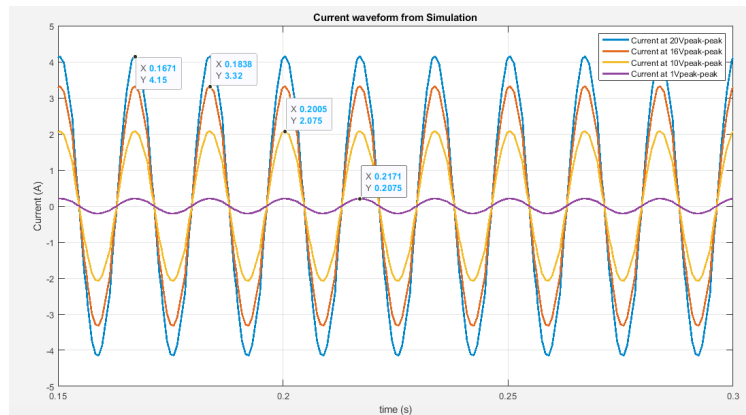


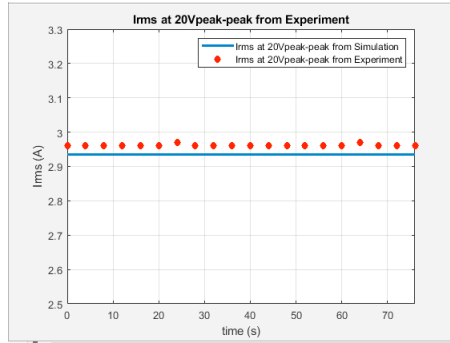
Figure 11: Current waveform from Simulation

Similar to the voltage simulation graph, this picture reveals the current magnitude flowing over the 0.33 ohm resistor under 4 different occasions provided by the function generator. With 4 blocks, the values at peak are labeled clearly.

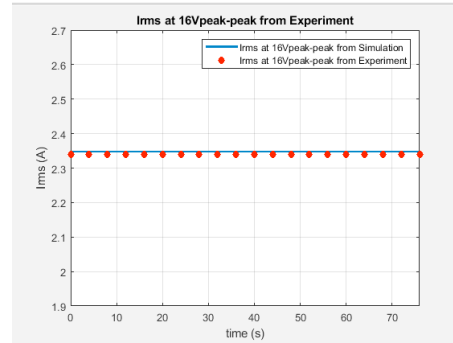
**Table 2: Irms based on Simulation graph (Figure 11)**

Amplitude of Current Waveform (A)	Irms(A)
4.15	2.934
3.32	2.347
2.075	1.467
0.0275	0.14672

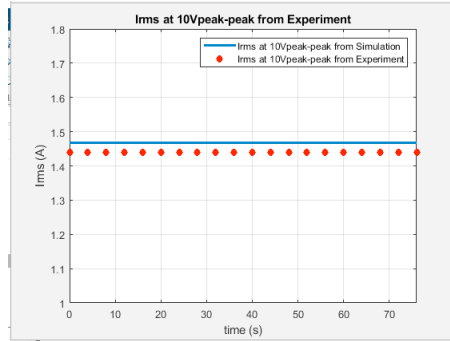
Similar to the voltage, we calculate the RMS current values by hand and these are the values used to check current measurement accuracy in our design. The the graphs below, is our comparison of the 20 collected samples with the calculated ideal values.



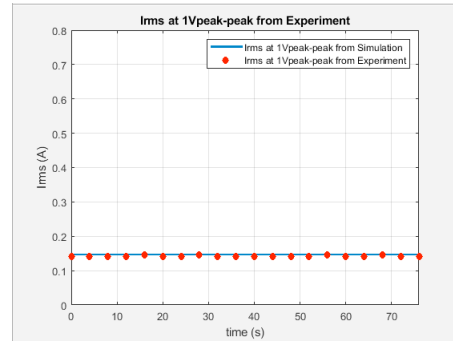
(a) Irms at 20Vpp from Experiment



(b) Irms at 16Vpp from Experiment



(c) Irms at 10Vpp from Experiment



(d) Irms at 1Vpp from Experiment

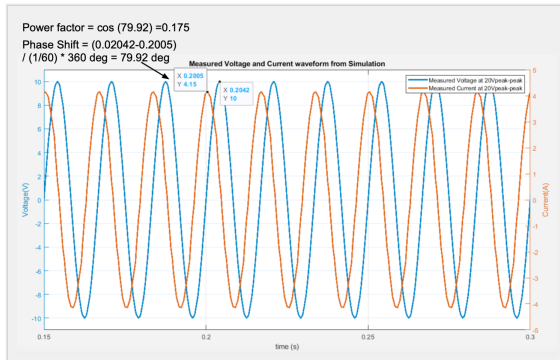
Figure 12: Irms from Experiment

Apparently, based on the graphs presented, the accuracy of our result of RMS current is pleasing and even though there are mild fluctuations, the result seems working fine from low voltage levels to high voltage levels.

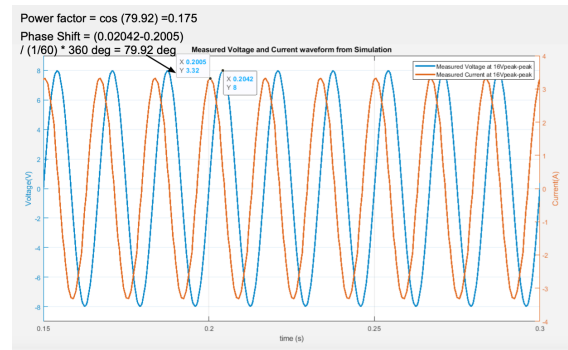
### 3.1.3 Power Factor Measurement Verification

After measuring voltage and current correctly, we could have the correct apparent power measurement. However, since our high-level requirements asked us to measure the real power, which catches more interests, we need to make sure that the real power we calculated satisfies the relation that the real power equals the apparent power times the power factor which is the cosine of the relative phase difference between the voltage and current. Therefore, to illustrate that we are having the real power calculated correctly, we have to compare the actual power factor.

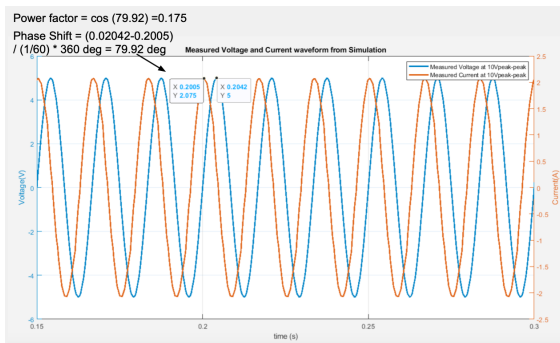
Below are the graphs from simulation results that the phase difference under different function generator supply. Not to our surprise, the phase difference does not change from high voltage to low voltage and we want to see the same result on our measurement as well.



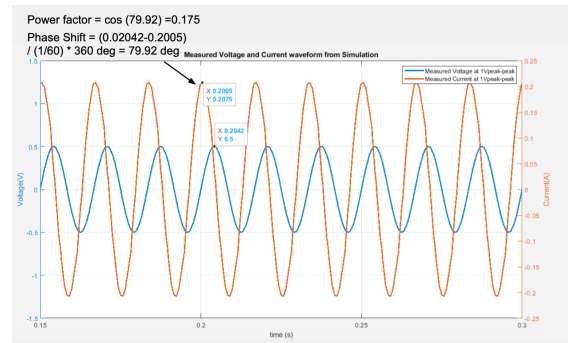
(a) Power Factor at 20Vpp from Simulation



(b) Power Factor at 16Vpp from Simulation



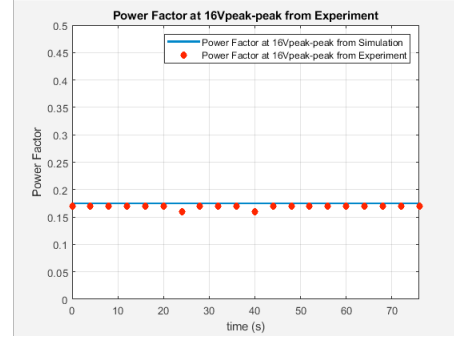
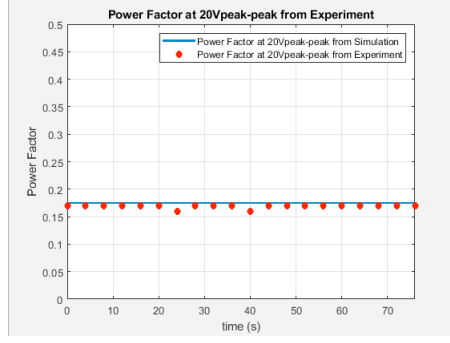
(c) Power Factor at 10Vpp from Simulation



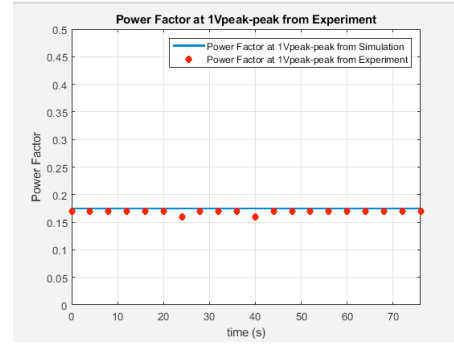
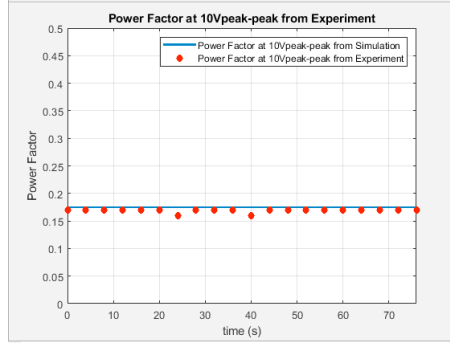
(d) Power Factor at 1Vpp from Simulation

Figure 13: Power Factor from Simulation

As introduced above, the phase difference does not vary too much, leading to a constant power factor in the circuit. Therefore, our goal is to see a fixed power factor with correct value printing out as well.



(a) Power Factor at 20Vpp from Experiment (b) Power Factor at 16Vpp from Experiment



(c) Power Factor at 10Vpp from Experiment (d) Power Factor at 1Vpp from Experiment

Figure 14: Power Factor from Experiment

Obviously, our measurement does not vary too much when voltage supply from the function generator rises or drops. Therefore, the power factor is also considered to be measured correctly.

### 3.1.4 Apparent Power measurement Verification

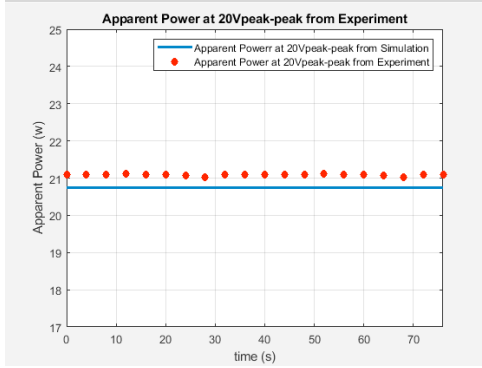
After measuring RMS voltage and current correctly, we could have the correct apparent power measurement. The apparent power is calculated based on the Equation(2). The simulated RMS voltage and current in Table 3, is derived from the Table 1 and Table 2.

Table 3: Apparent Power calculated based on the Table 1 and Table 2

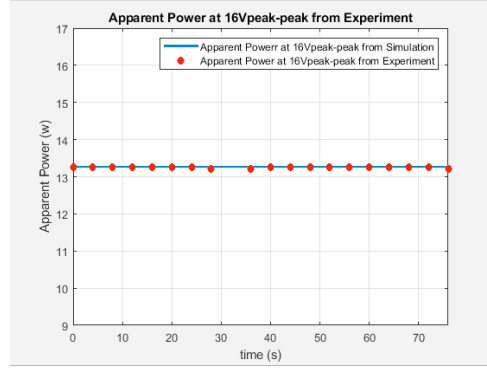
Vpeak-peak in function generator	Vrms(V)	Irms(A)	Apparent Power(W)
20Vpeak-peak	7.071	2.934	<b>20.74</b>
16Vpeak-peak	5.656	2.347	<b>13.27</b>
10Vpeak-peak	3.535	1.467	<b>5.185</b>
1Vpeak-peak	0.3535	0.14672	<b>0.05186</b>

The the graphs below, is our comparison of the 20 collected samples of apparent power with the calculated

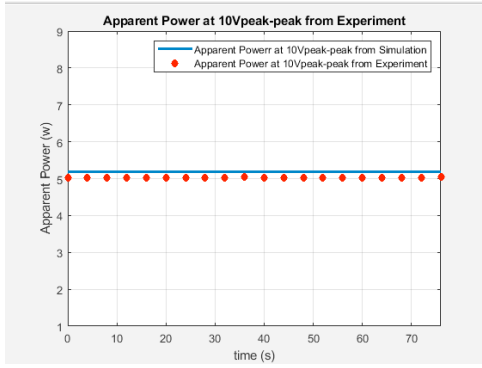
ideal values.



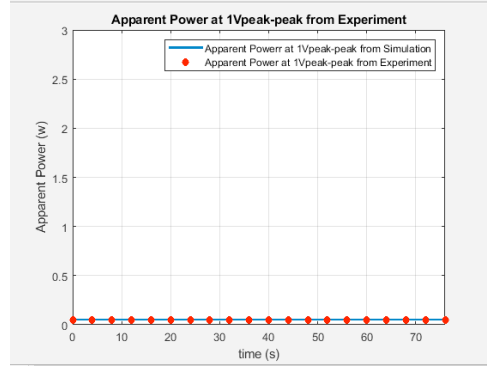
(a) Apparent Power at 20Vpp from Experiment



(b) Apparent Power at 16Vpp from Experiment



(c) Apparent Power at 10Vpp from Experiment



(d) Apparent Power at 1Vpp from Experiment

Figure 15: Apparent Power from Experiment

Apparently, based on the graphs presented, the accuracy of our result of apparent power is pleasing and even though there are mild fluctuations, the result seems working fine from low voltage levels to high voltage levels.

### 3.1.5 Real Power measurement Verification

After measuring RMS voltage, RMS current and power factor correctly, we could have the correct real power measurement. The real power is calculated based on the Equation(6). The simulated RMS voltage, RMS current and power factor in Table 4, is derived from the Table 1, Table 2 and Figure 13.

Table 4: Real Power calculated based on the data from Table 1, Table 2 and Figure 13

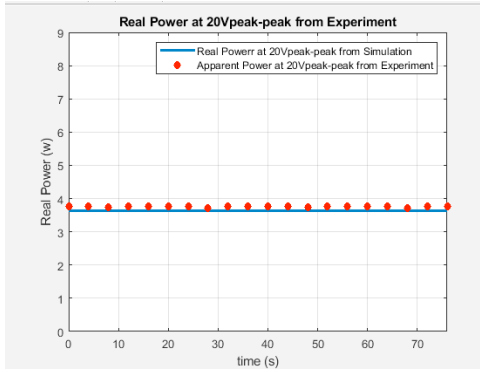
Vpeak-peak in function generator	Vrms(V)	Irms(A)	Power Factor	Real Power(W)
20Vpeak-peak	7.071	2.934	0.175	<b>3.63</b>

Continued on next page

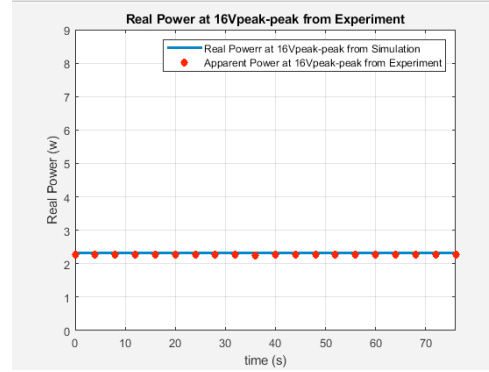
Table 4 – continued from previous page

Vpeak-peak in function generator	Vrms(V)	Irms(A)	Power Factor	Real Power(W)
16Vpeak-peak	5.656	2.347	0.175	<b>2.32</b>
10Vpeak-peak	3.535	1.467	0.175	<b>0.635</b>
1Vpeak-peak	0.3535	0.14672	0.175	<b>0.00907</b>

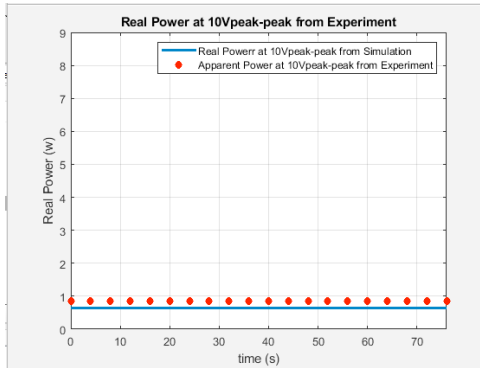
The the graphs below, is our comparison of the 20 collected samples of real power with the calculated ideal values.



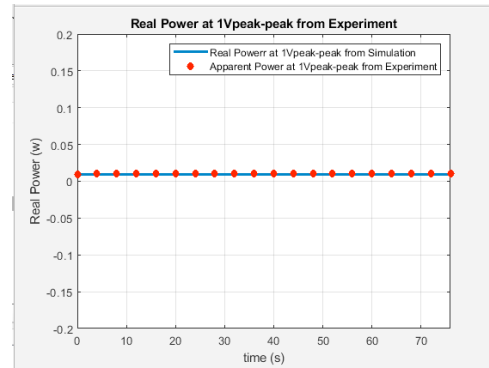
(a) Real Power at 20Vpp from Experiment



(b) Real Power at 16Vpp from Experiment



(c) Real Power at 10Vpp from Experiment



(d) Real Power at 1Vpp from Experiment

Figure 16: Real Power from Experiment

Apparently, based on the graphs presented, the accuracy of our result of real power is pleasing and even though there are mild fluctuations, the result seems working fine from low voltage levels to high voltage levels.

### 3.2 ESP32 Data Collection and Upload

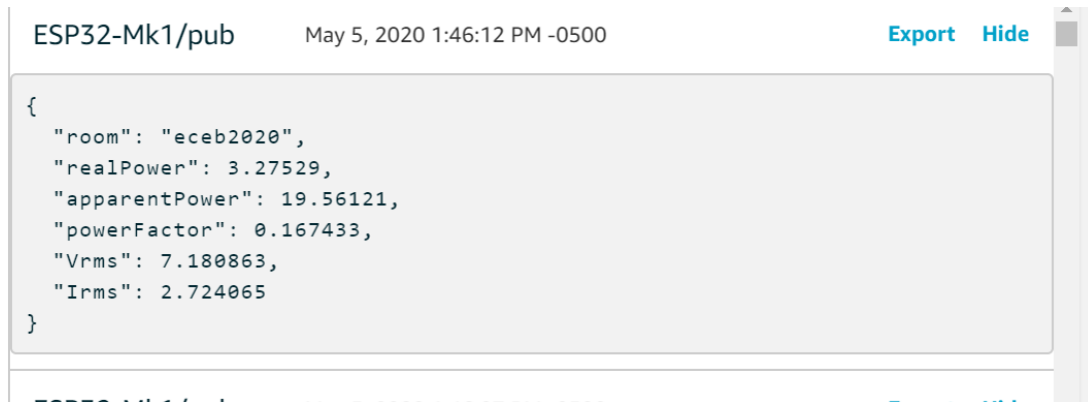


Figure 17: Screenshot of IoT Topics

The figure above is a screenshot from AWS IoT console. After manually subscribing to the topic that ESP32 is publishing to and acquired this Json package. This demonstrated that the ESP32 successfully connected to WiFi and complete the IoT configuration as planned. And the the data structure that is responsible for storing data did not lose the significant bits during the storage.

### 3.3 Cloud Data Handling and storage

Scan: [Table] serverlessrepo-lambda-iot-rule-MyTable-VIA... < Viewing 1401 to 1500 items >							
	Irms	Vrms	apparentPower	powerFactor	realPower	room	timestamp
0-8ee8-11ea-97c6-bf6ca670bb27	1.261463	3.334884	4.206834	0.165856	0.697728	eceb2020	1588694055580
0-8ee4-11ea-97c6-bf6ca670bb27	0.017198	0.015862	0.000273	0.167908	0.000079	eceb2020	1588692340029
0-8992-11ea-86bb-496fe2a85a4	3.058571	7.30537	22.35501	0.169024	3.778763	eceb2020	1588107369975
0-8f00-11ea-b4aa-fb4489116bff	2.724065	7.180863	19.56121	0.167433	3.27529	eceb2020	1588704372337
0-8f05-11ea-b4aa-fb4489116bff	2.724149	7.180422	19.56067	0.167344	3.273471	eceb2020	1588706523842
0-8eea-11ea-97c6-bf6ca670bb27	1.279443	3.382998	4.325497	0.166089	0.718415	eceb2020	1588694928101
0-8f00-11ea-b4aa-fb4489116bff	2.724316	7.181159	19.5632	0.167243	3.27189	eceb2020	1588704377437

Figure 18: Screenshot of Dynamodb

The figure 18 is a screenshot of DynamoDB. The selected entry exactly matched the package in Figure 17, and the timestamp matched figure 17 as well. This demonstrate the success of storing ESP32 local data into the database.



## 4 Cost

### 4.1 Parts

Table 5: Parts Costs

Part	Manufacturer	Retail Cost (\$)	Bulk Purchase Cost (\$)	Actual Cost (\$)
UCY2G100MPD	Digi-Key	0.82	0.62	3.28
UMA1H010MCD2TP	Digi-Key	0.24	0.16	0.48
MBA02040C4700FRP00	Digi-Key	0.24	0.2	0.72
BC4060CT-ND	Digi-Key	0.29	0.24	1.16
BC3742CT-ND	Digi-Key	0.4	0.3	1.6
RNF14FTD1K00CT-ND	Digi-Key	0.1	0.07	0.3
PPC200ZCT-ND	Digi-Key	0.29	0.24	0.87
2.54mm Pitch Female PCB Header Connector	Amazon	6.15	6.15	6.15
PCB Board	OSH.Park	33.33	33.33	33.33
AWS cloud cost	Amazon Web Service	1.54		1.54
<b>Total</b>		<b>43.4</b>		<b>43.4</b>

Figure 19 below is the AWS Billing Cost graph, which shows the last month, moth-to-date, and month-end forecasted costs. The cost is really negligible at the scale of only one device activated. A rise in scale could leads to rise in cost, however, at this stage the cost of use cloud server is significantly lower than setting up a physical server.

Current month-to-date balance for May 2020

**\$0.26**

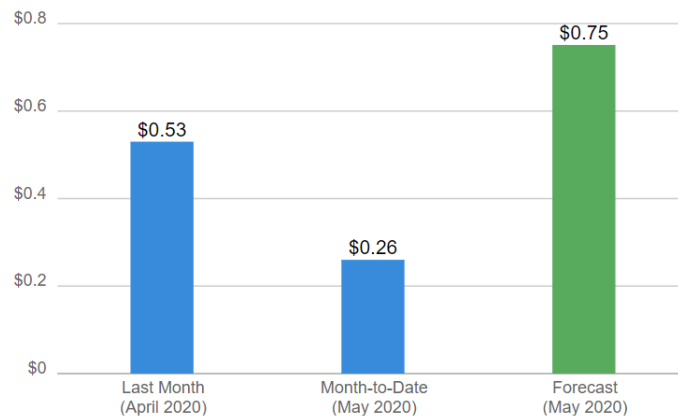


Figure 19: AWS cloud cost during the semester

## 4.2 Labor

For each partner of the project, the labor cost can be calculated as:

$$30\text{dollars/hr} \times 8\text{hrs/week} \times 11\text{weeks} \times 2.5 = 6600\text{dollars}$$

This should be our labor value building this project.

## 5 Conclusion

### 5.1 Accomplishments

First of all, it is our honor to be a student here at UIUC and to have this course serve as the end-chapter assignment of our undergraduate studies. We feel delighted and grateful for the support from our professor and TA for assisting us throughout the semester, especially when the pandemic forces many groups to wrap up on their implementation. The actual device has been constructed, even it is just a prototype and does not have fancy encapsulation covers, it functions well. We look forward to extend our project into our original design, a product that can be placed in the walls of the ECEB and have wires that flow high voltage and current around them. In summary, we achieved all our high-level requirements for the home-scope AC power measurement device and the accuracy is less than 5 percent. We appreciate the time working with group members as well as the course stuff and the workshops such as the Kidcad assignment and soldering assignment definitely added significant part to our skill-lists and we believe it will be very helpful after our graduation.

### 5.2 Uncertainties

The uncertainties for our project is minimal, as we tested through most of the situations (from high voltage source to low voltage source) that mimic the real one. However, we do still have curiosity about how our design would behave if we are using an inductor for the "phase creation" process. Our simulation result looks as fine but since the PCB ordering has been very limited this semester, we were not able to put inductors in the place of capacitors to see if our design could work as well.

### 5.3 Ethical considerations

Based on the 10 ethics mentioned on the IEEE code of ethics [14], we have a thorough consideration of our project and believe that our project will not violate these items.

- 1. to hold paramount the safety, health, and welfare of the public, to strive to comply with ethical design and sustainable development practices, and to disclose promptly factors that might endanger the public or the environment;
  - Our project will be enclosed in a plastic case in the wall and no smell nor harmful stuff will leak to the public.
- 2. to avoid real or perceived conflicts of interest whenever possible, and to disclose them to affected parties when they do exist;
  - Our project will not cause any conflicts to any forms of interests: it's simply a small device that helps monitoring the power consumption at our own place.
- 3. to be honest and realistic in stating claims or estimates based on available data;
  - Our project serves primarily to report the measurement from our local devices. Thus, even though tiny measurement errors may occur, there would be no dishonesty involved.
- 4. to reject bribery in all its forms;

- Our project is automated from front-end to back-end and so since no man-force is involved in this measurement, no bribery is possible.
- 5. to improve the understanding by individuals and society of the capabilities and societal implications of conventional and emerging technologies, including intelligent systems;
  - Our project will eventually work as a public presentation in the building for any public curiosities of any form.
- 6. to maintain and improve our technical competence and to undertake technological tasks for others only if qualified by training or experience, or after full disclosure of pertinent limitations;
  - Our project will be implemented in the lab first and then to carry out as real designs to measure the power in walls, so it should be a fully-tested project.
- 7. to seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of others;
  - We will do credit for any work we utilized and are welcome for any advice.
- 8. to treat fairly all persons and to not engage in acts of discrimination based on race, religion, gender, disability, age, national origin, sexual orientation, gender identity, or gender expression;
  - Our project has nothing to do with discrimination.
- 9. to avoid injuring others, their property, reputation, or employment by false or malicious action;
  - Our project is automated from front-end to back-end. Therefore, there is no chance for people to get injured on our project.
- 10. to assist colleagues and co-workers in their professional development and to support them in following this code of ethics.
  - Our project is built on our own thoughts and the previous works of many others and hopefully our design could be useful for any followers in this field.

## 5.4 Future work

Certainly, for the future work of our project, we expect that there should not be too much other than the two transformers to add in order to extend our design to our original plans. With the functionality to work with inductive circuits tested as well as the transformers working along with room mains well, our device should be able to measure the power consumption on each phase of the target room. By simply adding those functions together and upload to the server, the initial pitch from the professor can be realized. In addition, if we can expand our budget, we could also try to add an external ADC to our microprocessor or even change to another microprocessor that has more bits for analog signal readings. This way, the accuracy of our project could also be enhanced and thus be applied to even more occasions in real life.

## References

- [1] User-Twitter, “Introduction to esp32.” [Online]. Available: <http://www.iotsharing.com/2017/05/introduction-to-esp32.html>
- [2] “openenergymonitor/emonlib.” [Online]. Available: <https://github.com/openenergymonitor/EmonLib>
- [3] “Data-driven documents.” [Online]. Available: <https://d3js.org/>
- [4] “Aws sdk for javascript documentation.” [Online]. Available: <https://docs.aws.amazon.com/sdk-for-javascript/index.html>
- [5] “Iot based energy meter reading using arduino.” [Online]. Available: <https://www.innovatorsguru.com/iot-based-energy-meter-reading-using-arduino/>
- [6] A. the A. InnovatorsGuru, “Ac power measurement using arduino.” [Online]. Available: <https://www.innovatorsguru.com/ac-power-measurement-using-arduino/>
- [7] R. Wall, “Sampling rate of emonlib.” [Online]. Available: <https://community.openenergymonitor.org/t/sampling-rate-of-emonlib/4383>
- [8] “Installation and calibration.” [Online]. Available: <https://learn.openenergymonitor.org/electricity-monitoring/ctac/ct-and-ac-power-adaptor-installation-and-calibration-theory>
- [9] “Explanation of the phase correction algorithm.” [Online]. Available: <https://learn.openenergymonitor.org/electricity-monitoring/ctac/explanation-of-the-phase-correction-algorithm?redirected=true>
- [10] E. tutorials, “Electrical power in ac circuits and reactive power.” [Online]. Available: <https://www.electronics-tutorials.ws/accircuits/power-in-ac-circuits.html>
- [11] “Mqtt.” [Online]. Available: <https://docs.aws.amazon.com/iot/latest/developerguide/mqtt.html>
- [12] “Aws lambda.” [Online]. Available: <https://aws.amazon.com/lambda/>
- [13] “Aws dynamodb.” [Online]. Available: <https://aws.amazon.com/dynamodb/>
- [14] “Ieee code of ethics.” [Online]. Available: [Available:https://www.ieee.org/about/corporate/governance/p7-8.html](https://www.ieee.org/about/corporate/governance/p7-8.html)

## Appendix A Requirement and Verification Table

**Table 6: System Requirements and Verifications**

Requirement	Verification	Verification status (Y or N)
<p>1. Analog Front-end design</p> <p>(a) The voltage that the analog input pins read in ESP32 should be between 0.15 ~ 3.1V</p> <p>(b) The value of RMS voltage and current, power factor, apparent power and real power which read from the microprocessor (ESP32) should be measured correctly</p>	<p>1. Analog Front-end design</p> <p>(a) 1.Set the function generator to 20vpeak-to-peak, 16vpeak-to-peak, 10vpeak-to-peak, 1vpeak-to-peak separately.</p> <p>2.Connect the function generator to the analog front-end circuit.</p> <p>3.Connect the probes of the oscilloscope with the test points in the circuit and check if the waveform on the oscilloscope perfectly shifts up into the range (0.15 3.1V).</p> <p>(b) 1.Simulate the value of RMS voltage and current, power factor, apparent power and real power on Hspice</p> <p>2.Set the function generator to 20vpeak-to-peak, 16vpeak-to-peak, 10vpeak-to-peak, 1vpeak-to-peak separately.</p> <p>3.Connect the function generator to the analog front-end circuit</p> <p>4.Check if the value of RMS voltage and current, power factor, apparent power and real power read from the microprocessor (ESP32) match the calculated ideal value from simulation.</p>	Y
Continued on next page		

**Table 6 – continued from previous page**

Requirement	Verification	Verification status (Y or N)
2. Requirement (a) ESP32 is able to offload data to the server at least 4 times per hour (b) ESP32 has enough local storage (c) The data is accurate within 5% error	2. Verification (a) ESP32 is able to collect 24 readings per minute, and could upload at least 2 readings per minute. (b) ESP32 has enough local storage for 24 hours. (c) The significant figures are all preserved and the validity of Emoblib is confirmed	Y
3. Display on high resolution screen with Google Chrome browser	3. Worked on Chrome Browser perfectly with fixed aspect ration, but not tested on any high resolution screen comparable to the ones in ECEB lobby.	Partially

## Appendix B Repository for Project Codes

<https://github.com/Xiaoyis2/SubmeterECEB-Final>