**Smart Home Energy Management System**

**(SHEMS)**

*Team Members: Blake Wehunt, Andrew Lam, Ryan Mears*

*Faculty Advisor: Dr. Kourosh Sedghisigarchi*

*California State University Northridge*

*November 9, 2018*

# ***Abstract— The Shems project is a complete design with improvements from a previous senior design. Shems consists of four subsystems: the database, implemented with the use of xampp and aws; mobile app, made with android studio ide; shems circuit; and microcontroller, controlled with an arduino nano and esp8266 module. the integration of the 4 subsystems defines the Shems project, which provide users the ability to view power consumption of devices in their home with a user-friendly mobile app.***

# ***Keywords— Smart home, Android App, Arduino, Energy Management***

# Introduction

This semester’s goals were to take the existing design, model, and programing from Spring 2018, and to improve upon these to create a refined project. These can be broken down into five major task categories:

1) **Compact**- The device shall become smaller through the use of commercial off the shelf (COTS) parts and re-design with EDA software to design the first SHEMS pcb.

2) **Optimization**- Lower costs in the hardware and improve efficiency by acquiring different components and incorporating the use of more integrated components, such as an arduino module that includes wifi capability.

3) **Transfer**- Move the database from the local XAMMP server to an Amazon Web Services (AWS) server.

4) **Improve/Create**- The mobile app required a bill summary report to display daily costs from power consumption, background update task, and further modifications for a user-friendly GUI.

5) **Communications**- The microcontroller must be able to send data collected from SHEMS circuit to AWS database and the mobile app must be able to retrieve the same data in order to provide an accurate cost analysis and visual displays for the user..

# Project Description

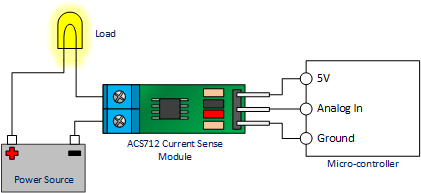
## Overview

The Smart Home Energy Management System (SHEMS) project hardware is constructed in four primary sections. These individual sections each do a specific part, and the data retrieved from the individual sections, processed, and send to a user friendly application to be viewed. We utilized these main components overall, while changing and optimizing parts within these main groups. Three of the four main groups are the Current Sensor, Voltage Sensor, and the Microcontroller. The fourth component enables switching, which gives the user the ability to turn on and off the load device if desired. This is done through a Triac.

The basis of this device is centered around the voltage sensor, and the current sensor. with these two devices, we can measure the phase angle, Vrms, and Irms, which in turn can all be used to calculate the Apparent Power, and the Power Factor of whatever load may be applied to the device. This data is sent to the Microcontroller, where it is calculated based on an an imbedded code. Once this has been calculated, it is sent to the database, where the data can be viewed and the device controlled via a user friendly cell phone application.

## Current Measurement

The current measuring device was not changed from the previous year. we are using a Hall effect Sensor, P.N. ACS712. the implementation of this device can be seen below:

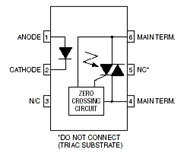


*Figure 1.1 Example of ACS712 in a working circuit*

## Voltage Measurement

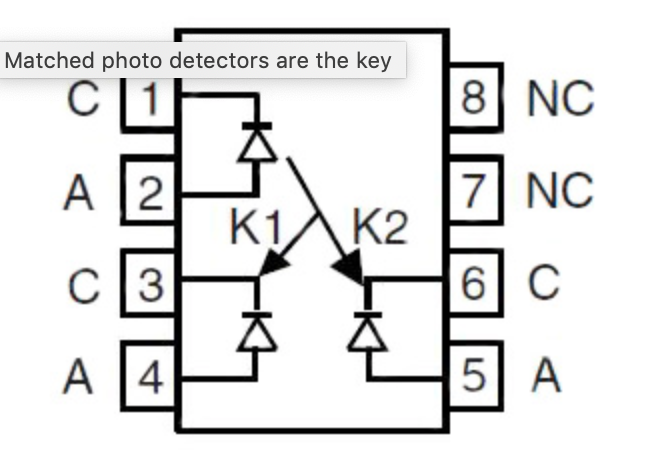
The Voltage sensor went through some back and forth changes this semester. Initially we were using a digital opto-coupler. The use of one of these devices enables the system to isolate, optically, the microcontroller from the rest of the circuit systems and their main line potential, which can potentially have an impact on the linearity of the output of the opto-isolator. The benefit of using this device, is that it scales the voltage down to a significantly smaller range, about 5V.

Fig. 1.2 is a display of the opto-isolator. This shows how the input and outputs are completely isolated from one another, and uses an LED on the input side to trigger a sensor on the output side.



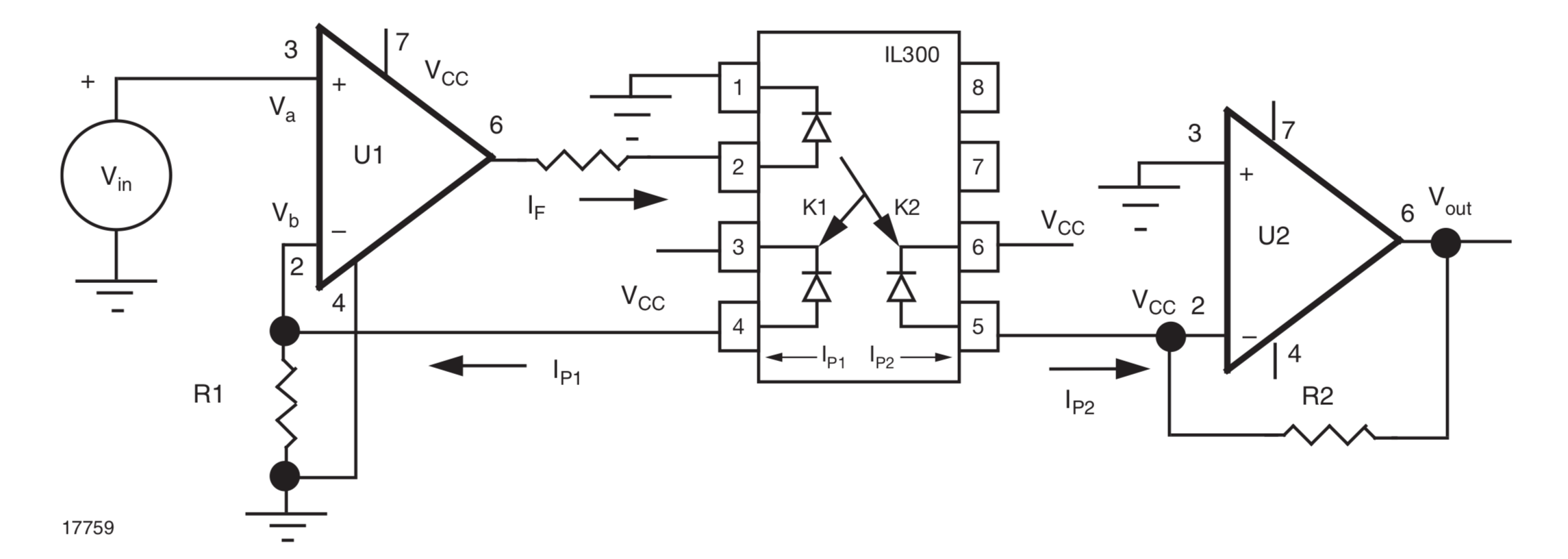
*Figure 1.2 Digital Optocoupler/Optoisolator Internals*

Initially the fear was that this device, though isolating and independent, was not truly linear. The decision was made at this point to change to an analog optocoupler, spefifically the IL300, seen below In Figure 1.3:



*Figure 1.3 Analog Optocoupler/Optoisolator Internals (IL300)*

The IL300 was tested in a stand alone design, shown in Figure 1.4, and this device too, was deemed unsuitable for the needs of the project. Test used two lm741 op-amps.



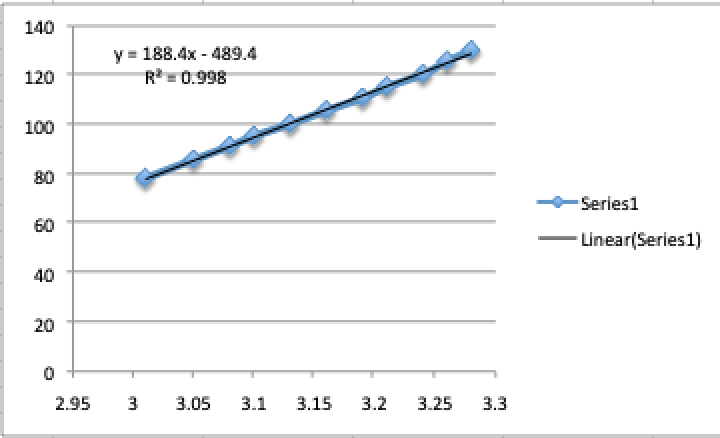
*Figure 1.4 Testing linearity of IL300*

After testing the IL300, as configured in figured 1.4, we discovered issue with using the analog optocoupler. In order to make the IL300 work, we would need three separate power supplies in order to make the system truly independent and isolated, because using a common power for the op-amps, and for the micro controller would lead to interference and distortion. This system would also require the use of a different op-amps, which would not be possible for both budget and time constraints.

This is where an engineering decision came into play, the options were to go over budget and schedule while doubling overall the physical size of the device, or make a few changes and work with what we had. The decision was made to move forward with our current set up, and make any corrections for error in the current set up, in our code and programing. Moving forward, we decided to keep the digital optocoupler, and work around the possibility of it being not very linear. to do this, we set the system up in the lab, and using a variable AC power supply, we were able to vary our input power, and measure our output powers on the output of the optocoupler. we varied our input voltages in steps of about 5V and graphed the results to find a trend line, and percent error in the system.

|  |  |
| --- | --- |
| Vin(AC RMS) | Vout (DC) |
| 78.2 | 3.01 |
| 85.7 | 3.05 |
| 91.2 | 3.08 |
| 95.3 | 3.1 |
| 100.2 | 3.13 |
| 105.7 | 3.16 |
| 110.8 | 3.19 |
| 115.3 | 3.21 |
| 120.4 | 3.24 |
| 125.7 | 3.26 |
| 130 | 3.28 |

Graphing this data resulted in the following graph.



This graph surprised us in producing a very linear output. This proved to us that the use of the digital optocoupler was sufficient for our needs. We obtained the formula for the best fit line to be

y = 188.4x - 489.4

with an R2 = 0.998. This gives us an error of 0.001.

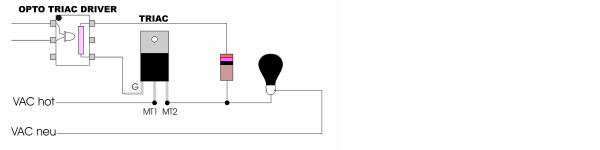
As stated previously, there are two main reasons for using an opto-isolator. To keep the the input and output independent, and to scale down the voltage range. The driving force for scaling down the voltage range is to feed the microcontroller a safe and accurate voltage sensor input. The microcontroller we are using is designed to safely measure a range of 0-5V. Note that this means that voltages cannot be below 0V. A negative voltage will damage the microcontroller, and because of these two restrictions, just measuring the 120V input sine wave would give negative voltages, and voltages far beyond the 5V range of swing.

The positive voltage only input was obtained by the use of a DC offset, bringing the voltage range up 2.5V to give an output sine wave with a final range of 0V-5V. We can then use this value to calculate the phase angle, Irms, Vrms, and the Active and Reactive Power.

This design, using the digital opto-coupler was chosen by the previous team over a step down or a 1:1 transformer because of its simple design, yet not having any hysteresis. It also helps that it can be powered off of the same voltage as the hall effect, thus making things more modular. Another this this device prevents over a transformer, is that it prevents the system from going into common mode, which can be devastating to the measurements.

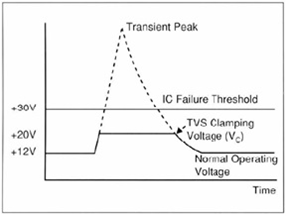
## Control Circuit

The control capability of this project has not changed from the previous team. The SHEMS device has the ability to switch the load(s) it is in line with on or off, depending on the user input via the mobile application. This feature is made possible by the use of a Triac and the photo-isolator. This particular part of the circuit can change the state of conduction. The circuit acts as a normally open switch, waiting on a command from the mobile app via the microcontroller so that it can conduct power to the load. A signal voltage (3V) is sent from the microcontroller to switch the triac to a closed switch state, thus conducting power.



*Figure 1.3 Example of an Optocoupler driven Triac for Solid State Switching*

When a signal voltage is applied from the microcontroller to the triac, the optocoupler will trigger thus allowing current to flow to the load.. An added benefit to using the photo-isolator is that it utilizes a “Zero Crossing” circuit, helping to control an inrush current and inductive kickback voltage when commanding a device to off.



*Figure 1.4 Graph of operation of Transorb during voltage spike*

A transorb was also added across the load as an added safety. This helps to protect the triac, and the photo-isolation circuit.

# Design Approach

## Arduino Nano Overview

The SHEMS unit was implemented with an Arduino Nano microcontroller. The benefits that the come with choosing the Nano include small size, highly compatible programming language, as well as the relatively low cost. The Nano is equipped with the ATmega328 microcontroller with 32KB flash memory, of which 2 KB is used by the bootloader.

The Arduino Nano was selected instead of the NodeMCU ESP8266-12E in part to the Nano’s more affordable pricing ranging from $10 to $20 in addition to the more compact size with a PCB size of dimensions 18mm x 45mm. However, the primary benefit of using the Arduino Nano is that the analog input pins are capable of measuring the voltage and current readings required for power calculations including real power apparent power, and power factor without the need to include external hardware such as an analog to digital converter (ADS1115) as required in past SHEMS designs.

A drawback to using the Nano is the lack of integrated WiFi capabilities on the device. To rectify this problem, it is necessary to include an external WiFi module in order to transmit the power calculations to a network database. To this end, the ESP8266-01 is the WiFi module used to transmit the data to the database.

## Power Calculations and Code

The entire purpose of this project comes down to the power calculation ability of the device. This is done by measuring the applicable analog inputs of voltage and current being drawn by an appliance, light bulb, or other type of household electrical load, and perform the necessary calculations accurately, and in real time. This feature is achieved by having the microcontroller receive the stepped down voltage value from the optocoupler voltage sensor and the stepped down current value from hall effect. These two values are input to the microcontroller. These are then processed through an 8 channel 10 bit Analog to Digital converter. This A to D converter is responsible for converting the input voltage to values from 0V to 5V, to integer values between 0 and 1023. This is accomplished at a rate of 10,000 times per second.

After the analog input signal is converted to a digital signal, it is send to the microcontroller for data samples to be taken of the current and voltage values. This sample is taken once every 2 occurrences, which is a full period, due to the positive only values (0V-5, the bottom of the sine wave is clipped at 0). The actual number of samples taken will be later used to calculate the average and RMS values of the current and voltage. the real and apparent power can also be calculated at this point.

This semester, we did not change how the microcontroller calculates the average and RMS current and values, nor the power factor and apparent and real power values. We were able to implement these calculations and formulas into the changes we made in both the hardware and the software.

Since we have stepped down the voltage and current values from the actual values, we needed to calibrate the system so that we would end up with an accurate value after calculations were made by the system. This calibration required the use of an AVR microprocessor. In order to be sure that the correct values are being interpreted from the received input values in comparison with the voltage value that the Nano is using as a reverence to Vcc, it is required that the system be calibrated.

The AVR chip selected for the SHEMS project is the ATmega328P. This was selected because of the compatibility with the Arduino Nano that the project utilizes. The selected AVR has a specific calibration constant of 1098595L. On top of the reference voltage calibration being required, calibration constants for , , and are also required. This is required so that the internal analog current and voltage values can be scaled to the correct phase and magnitude. The equation used achieve the correct calibration constants can be considered as:

After using this equation several repeatedly, the values for the calibrations constants for current, voltage, and phase were found to be:

The obtained values of and are converted into ratios and . When these values are multiplied to the RMS value of the voltage and current, the result would provide the correctly scaled values of the respective two. The formulas for these values can be written as:

where

where

One important thing to consider is that the can vary depending on the sample which the controller has referenced.

With these values calculated, the RMS values can now be obtained. It is important to note that a digital low pass filter has been coded in as a result of the 2.5 volt offset and half bridge rectifier waveform output. This is implemented to extract the voltage offset. This filter is written in such a way that the digital voltage values can be taken at the start and end of the period, when the bit values equal 0V.

Once the signal is centered, the digital instantaneous voltage and current signals will be converted to Vrms and Irms. From these values, the power factor of the load, as well as the apparent power being consumed can be obtained. The real power can also be determined by use of the instantaneous signal values of the voltage and current. This value is obtained using the summation of the instantaneous power at each sample taken, in conjunction with the below formula:

, where .

The power factor and the apparent power can be calculated by first squaring each of the respective waveforms that is needed. We can then average the square of the two waveforms and divide them by the number of samples that were taken to achieve the summation. Finally, the ratio of the voltage or current, respectively is multiplied by this square root value, seen below:

Since we have both the voltage and current digital signals in an RMS form, we can calculate the apparent power:

and the power factor:

Once the values of the apparent power, real power and power factor have been obtained, they are stored and sent to the ESP8266 WiFi module to that they can be transmitted to the database and the mobile application.

How the power factor is calculated is by getting the output sine wave curves for the voltage and the current. These two waveforms will have a slight delay (θ) between them. To obtain the value θ, we apply the following equation:

θ =

The value of ‘60’ in the denominator is the frequency of the input voltage. In our case this is 60Hz because we are on a 60Hz power grid and our input voltage is at 60Hz. Once we obtain a value for θ, we take the cosine of it and this will give us the Power Factor.

This was tested in the lab, using a load bank, variable AC power supply, and the ability to change the load to have a leading or lagging power factor, we ran the system at 0, full lagging, and full leading. We found this to have a lagging delta T of 1.2ms, and a leading delta T of 1.6ms. These respective delta T’s gave us a leading PF of 0.82 and a lagging PF of 0.899. One thing to note is that a leading PF results in a high-pass filter, and a lagging PF will result in a low-pass filter.

While observing the waveforms of the current and the voltage, as they are taken from the current sensor and the voltage sensor, it was notice that the current output needed to be filtered, as the unclean output signal had several spikes, that could in turn lead to inaccurate measurements and calculations.

## ESP 8266-01S0 WIFI Shield Pros/Cons Overview

The ESP8266 is an inexpensive WiFi module that will meet the requirements needed for most projects that include the need to connect to the internet. The module comes equipped with a range of capabilities such as 2 pins used for GPIO (General Purpose Input Output) and support various functions such as I2C, SPI, and UART. While this module boasts high compatibilities with programming IDEs (Integrated Development Environment), including the Arduino IDE, the official documentation of the ESP8266 is only available in Mandarin, making it hard to access for primarily English speakers. However, the ESP8266 has since gathered a large community of users who have then published a large quantity of third-party documentation on the ESP8266. One consideration to keep in mind when implementing the ESP8266 is that the input voltage should not exceed 3.3 volts for risk of damaging the module. This requirement necessitates the use of either a voltage divider to reduce the incoming voltage to the module from the standard 5V input or alternatively include the use of an additional power supply. For this project, the benefits of implementing the ESP8266 far exceed the drawbacks that the module possesses.

## Pin Connection

The connection of the various components to the microcontroller was crucial in this project. In order for the arduino nano to send or receive data to and from the micro controller, certain connects were to be made. The CHPD and VCC ports were powered with 3.3 V which was reduced by the voltage divider, from the 5V line voltage from the power supply. The RST, GP0, GP2 pins on the microcontroller were not used; the GND port was connected directly to ground. The TXD is directly connected to the nano’s GPIO port, (pin 10). Finally the ESP’s RXD port is directly connected to the arduino nano’s GPIO port (pin 11) with a resistor in line.

## Programming the ESP8266-01S Shield

After configuring and updating the firmware onto the ESP8266-01, the Arduino Integrated Development Environment (IDE) is used to upload programming to the device. A program that is written using the Arduino IDE is called a sketch. An Arduino program sketch consists of two functions: the setup() function, and the loop() function. The setup function is called after the program is first powered up or when a reset to the microcontroller has occurred. The function is used to initialize variables, set modes for the necessary pins, as well as to include the libraries that are required. This function will only be called once after which all included commands have been executed will it then call the loop function. The loop function has the property of continuously executing the commands it contains until the board is either powered down or is reset.

The code used to program the ESP8266 first requires the inclusion of the Software Serial library which is needed for the Arduino Nano and ESP8266 to communicate with each other. The baud rate is the rate at which the two devices communicate; the baud rate for both devices are set at a rate of 9600. In the setup function, the ESP8266 is reset and is set to client mode. The modes are set using a type of Hayes command set known as AT commands which consist of a parsed string read one character at a time. The AT commands follow a strict set of rules requiring the use of a carriage symbol at its end in addition to being issued serially. Once the AT command has been received successfully, the ESP8266 will respond with a serial message of an affirmative ‘OK’ or a negative ‘not OK’. If an affirmative response is not received, the program should then reset and either try again or cease communications.

In the loop function, the ESP8266 is issued more AT commands that will instruct the module to connect with the available network using the credentials provided to it. The Host, User, Password, and Database Name are passed through communications in order to properly connect to the database that will be storing the given calculations.

# Testing

## SHEMS Circuit Subsystem

The circuit was tested both at the individual system component level, as well an an entire, completed unit. This system was especially helpful when it came to the implementation of new hardware, such as the changes made in the power supply, the voltage sensor, and the the new analog optocoupler. Once all of the individual parts were tested and working, we could test the system as a whole. This required the use of a load. For testing the power factor and calculations, we used a standard household light bulb. This was chosen because a light bulb is very close to a perfect load, in the sense of the testers knew the power it consumed because of its wattage rating. This enabled us to be able to monitor what the system was doing, as well as it made it possible for us to do the math behind it and compare and calibrate the system. Once we finished testing with the lightbulb, we could change the load to a variable speed box fan, that let us change the load, thus changing the apparent and real power, as well as the power factor. This data was backed up by tests run in the power lab, where we had a variable load bank, and could change the system between leading and lagging.

## Microcontroller Subsystem

The microcontroller which is the Arduino Nano was used with correlation with the ESP8266 module. The purpose was to test the connection between the SHEMS circuit and the AWS server with the database. In order to send the power calculations to the MySql database through the use of a webpage containing HTML, PHP, and MySql commands. The connection hasn’t been fully integrated and is still a work in progress.

## Mobile Application Subsystem

The mobile app was created through the Android IDE, developed by Google and JetBrains, the features implemented by previous members were a login page, power factor graphs, on/off device switch, and statistical data display. The newly tested features from current members were a summary report, auto-update retrieval of data task, and improved user-friendly graphical user interface (GUI).

1. *Bill Summary Report:* The implementation of a cost analysis summary for users to view in real time was to provide another feature for the mobile app. The bill summary retrieved current power in watts and electric rate from database then was calculated to display device name with is current energy usage and charge. Originally a timestamp would accompany with the data retrieval at a set interval, however complications with conversions and updating new timestamp lead for a different approach. The algorithm below describes the new approach taken:

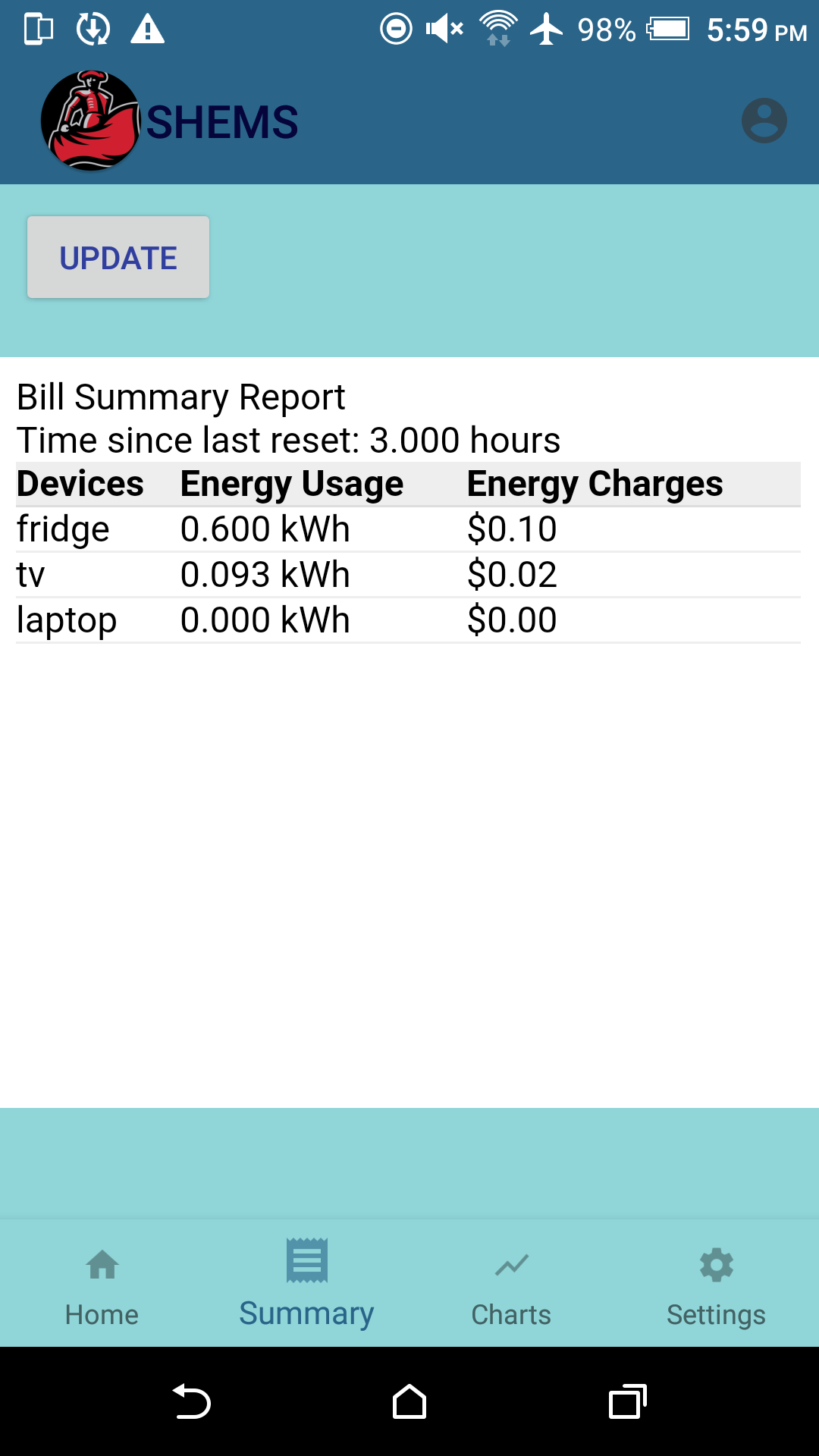
a)*\*(Sample-1)+) / Sample*

b) *\* ) / 60*

c)  */ ) \**

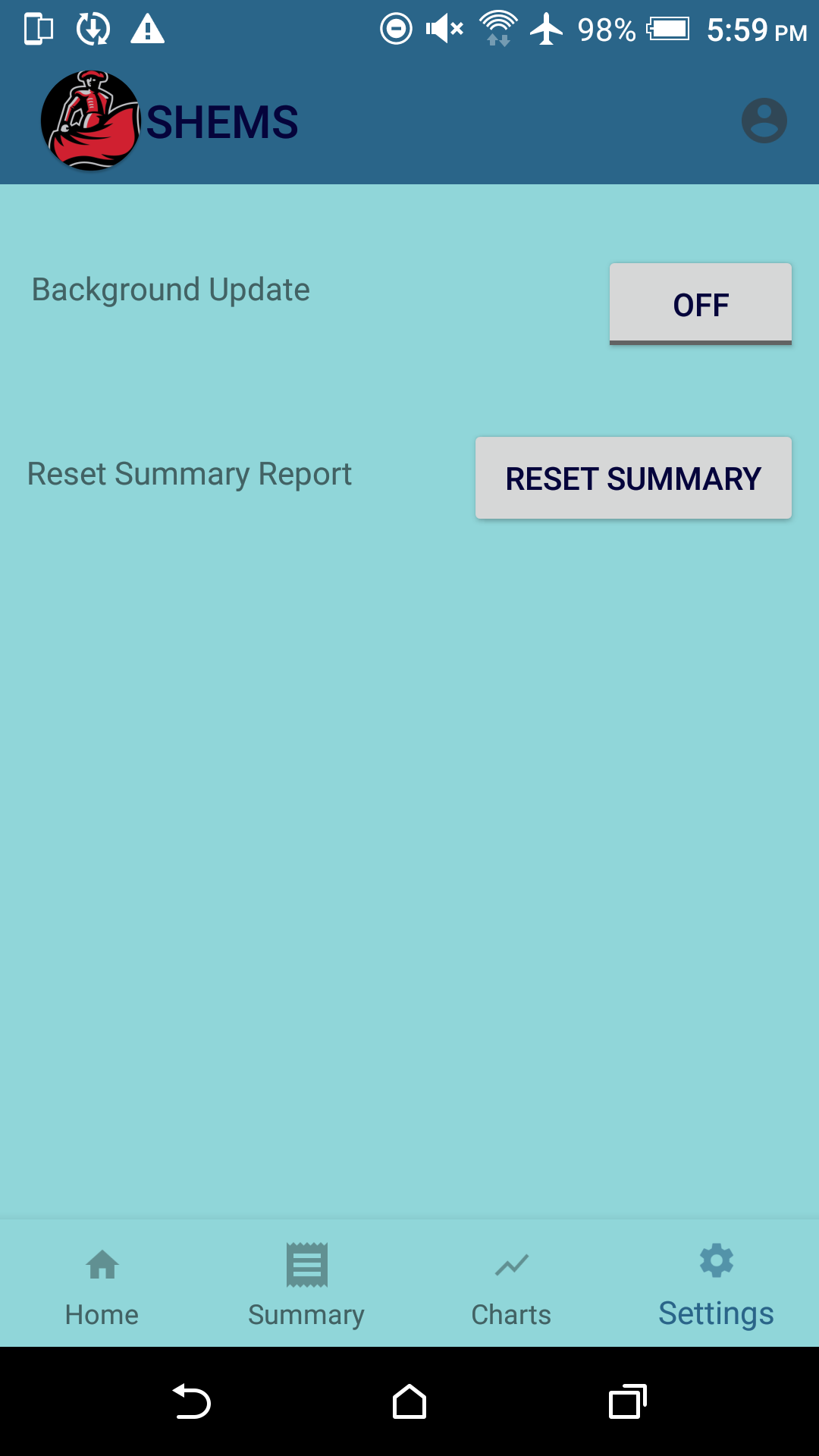
d)  *\* electric\_rate*

The algorithm proven to have worked based off testing with 200 W, 31 W, 0 W of three different devices which were sample every 10 minutes until it reached an hour of time. Figure 2.1 shows the energy usage and energy charges from 3 devices after 3 hours has passed of power consumption. Overall the bill summary is quite functional, it display the data in a string of data converted into a html webpage image.



*Figure 2.1 Bill Summary app page*

1. *Background Update Task:* The purpose of the background update task was to move away from manually pressing a button to retrieve data on mobile app. As shown in Figure 2.1, the button labeled “Update” was the only way to retrieve data from the MySql database which is inconvenient and would not be able to keep track of power consumption on a set period of time. The button was kept for debugging only and will be replaced with a new option which is a background service that would pull data at certain intervals over time. The current interval being every 10 minutes which was implemented through the use of a Alarm Receiver class and Broadcast Receiver class. The two classes function the same way as an alarm clock, waits for a certain time then run a code when known time is reached. In Figure 2.2, the background update is placed in the settings for users to turn off if needed to conserve battery usage as background services can caused drain of device battery if constantly being used to run a service or program. The reset summary option was added for users as well if users wanted only daily summary reports and not a cumulative summary report.



*Figure 2.2 Settings page*

1. *Improved GUI:* The user interface has been revamped to provide seamlessly navigation through the entire app and is features without the clunkiness from the previous version. The previous version display each button feature in a newly created activity page which cover the whole screen, with the only option to navigate back to previous page is with the back button on phone device. The improved GUI was implemented with the use of the Fragments Class, which placed multiple activity pages in a multi-pane fragment container which then are called fragments, reducing the need to click the back button. As shown in Figure 2.3 as well as the previous two figures, the only change occurs is the center where fragment container overrides with new fragment when a button on the navigation bar at the bottom is pressed. Another improvement is the storage of values being saved to view after closing the mobile application at a later time. This was able to be achieved through the use of SharedPreferences class, which stored key data into the phone device memory and retrieve it back when the application start again, therefore eliminating the need to start at default value of 0 or null.



*Figure 2.3 Main Screen app page*

*D. Database Subsystem*

The XAMMP server, an open source web server developed by Apache Friends, was locally run from a PC laptop. The slow server load handling speed based off the PC specifications and database information being inaccessible when machine was turned off or being only available if user’s device was on same network would become problematic. Thus, the transfer of a local server to a cloud computing platform where server requests handling and database usage being accessible all the time throughout the internet would need to be implemented.

1. *AWS Server:* The Amazon Web Services was tested with the transfer of the MySql/MariaDB database. The AWS had multiple cloud services that had compatibility with the existing XAMMP server and phpMyAdmin administration tool used to build the database. The newly public AWS server was created through Amazon EC2, virtual servers on the cloud instance service, which had to be manually configured into a LAMP(Linux, Apache, MySql, Php) server whereas XAMMP created the server automatically. The Amazon documentation on how to create the LAMP server was straightforward and easy to follow [1]. The configuration settings needed to be setup to allow HTTP requests to grab data from database and security rules to open ports on the public IP address. Once the AWS server was configured, the transfer of files from the XAMMP was needed which was transfer through SSH, a cryptographic network protocol which can remotely send files and login to the AWS server. The SHEMS project would now be able to handle an increased of users load on the server and the server can be accessible anywhere where Amazon Web Services is available.

# Conclusion

In conclusion, the hardware of this project initially was ready to be implemented to a PCB design and final product when we finished out last semester. As we began the current semester however, plans changed and several changes were desired, such as a change in power supply, and a power in the voltage sensor circuit. After several designs and ideas, we did find a smaller, more efficient power supply, but the voltage sensor ended up being left, after it was discovered that the current system was very accurate, and was cost, time, and space efficient.

The communication between the Arduino Nano microprocessor and database was left incomplete following the end of the last semester. Currently, the Nano can communicate with a private database including the ability to both write and read data from the said database as well as record the timestamp at which the database was accessed. The values stored in the database can then be updated in real time and displayed in a user-friendly manner.

The mobile application was improved over previous work now that users are able to see the costs over time their device are consuming and navigation has become more user friendly. The background service does need to improve such as take into account if the app is closed or device has shutdown or crashes. The database may have moved successfully to a public server than a local server however the network security has now become an importance, safeguarding the user's privacy and information when using SHEMS.

1. References

[1]Docs.aws.amazon.com. (2018). *Tutorial: Install a LAMP Web Server on Amazon Linux 2 - Amazon Elastic Compute Cloud*. [online] Available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-lamp-amazon-linux-2.html [Accessed 25 Aug. 2018].

[2]W3schools.com. (2018). *PHP Select Data From MySQL*. [online] Available at: https://www.w3schools.com/php/php\_mysql\_select.asp [Accessed 15 Oct. 2018].

[3]Android Developers. (2018). *Schedule repeating alarms | Android Developers*. [online] Available at: https://developer.android.com/training/scheduling/alarms [Accessed 10 Sep. 2018].

[4] *Preparation of Papers for IEEE Transactions and Journals*, 2007.

[5]Android Developers. (2018). *Creating a Fragment | Android Developers*. [online] Available at: https://developer.android.com/training/basics/fragments/creating [Accessed 2 May. 2018].

[6] Vishay IL300 Data. (2017) https://www.vishay.com/docs/83708/appnote50.pdf [Accessed 28 September. 2018]