SMART POWER MONITORING NETWORK

Project Report

Team Number: May1725

Adviser: Dr. Nathan Neihart

Client: Commercial Product

Team Members: Joseph Freeland (Co-Lead), Milan Patel (Co-Lead), Adam Cha (Communications Lead), Adam Dau (Webmaster), James Tran (Key Concept Holder), Wei LinLin (Key Concept Holder)

Team Email: may1725@iastate.edu

Contents

1. Introduction

1.1 Project statement

We have created a wireless power sensor which monitors the power usage of different electronic devices that appear around the house. This allows people to compare the power usage of different devices around the house in real time, and can help users improve their power usage. Our device reports the power usage back to the user via a user friendly web application. There is a central hub acting as the middleman between the sensors and the web application. All of this will be connected via the user's wifi network. Our device can monitor power continuously by itself, and will automatically update when the user opens the web application.

1.2 Purpose

Our device's purpose will mostly be decided by how the user would like to implement it. If they want to measure how much energy it takes to run a fan all day, as opposed to turning the A/C up, they could do that. Our power sensor would serve as a way to compare the two to decide which is more cost effective. They may also like to see how much energy leaving a night light on all night for their kids takes up. Or, how much power is being wasted by leaving their laptop charging all night. These are all ways that our power sensors could be used. Generally, we are giving people an opportunity to have more knowledge and control of how users use electrical devices around the house.

1.3 Goals

Our main goal was to be able to build a working power sensor unit that is connected to a user interface. That means that we have a sensor that can measure the power being used by a certain device. Send that data via wifi to a central hub that can then take that data, and send it to a web application that can present that data in a easy to use and understand format. We have completed this goal and have a functioning power monitoring device.

Our next goal was to have multiple devices plugged in and transmitting data at the same time, with the user interface being able to show all the devices in use at once so the user can look at and compare devices at the same time. Also, we would like the necessary equations to put the data in the format we want to be implemented in the power sensors circuitry. We did not have enough time to get multiple prototypes up and running at the same time. After we got our first prototype running we chose to center all of our efforts on making sure that device was functioning without any bugs.

2. Deliverables

There are three main deliverables that we created in order to complete our objective. First, is the power measuring circuit that plugs in-line with the devices it is monitoring to measure the power they are using. This needs to be small and out of the way, and use up very little power itself. Otherwise, the whole point of monitoring the system in order to help curb power consumption is destroyed.

Next, we have a central hub that takes and stores all the data to be used that has been gathered from the sensor. The central hub is currently in the form of a raspberry pi. Here the data is added to a database that the user interface will communicate with in order to get the information needed to present to the user.

Lastly, is the user interface. Again, this is in the form of a web application, and is able to show the power of a device that the user has plugged into our sensors. From this user interface, they are able to name all the devices they are monitoring for easier identification. The user interface also allows for easy scalability of time. This allows a user to easily navigate from minutes to days. Also the power is auto-scaled by the graph on the user interface so you always have a graph that fits to the scale that is currently in use.

Another deliverable that we decided to create is a case that will encapsulate our PCB. We created the case due to safety concerns. We wanted the user to be as isolated from the electrical mains as possible. Our case is 3D printed and held together by screws. The case consists a standard NEMA plug and an extension cord which allows a user to plug in their device without having to worry about taking up both outlets.

3 Design

3.1 Previous work/literature

There are several commercially-available energy monitoring software systems that we looked at before starting the design process. We wanted to make sure that our software and hardware could match or exceed the specifications of the systems on the consumer market.

From a hardware perspective, most systems use high side current sensing measurement. We developed a circuit that measures current and voltage, directly, on the low side of the load. We do this on the low side to avoid the impractical cost that comes with a high common mode differential amplifier needed for high side measurement. Although we use a different method compared to that of most systems, our components are the mostly the same. We are making use of a current sensing resistor, amplifiers, and other passive and active components. One of those amplifiers is digitally-programmable. This digitally-controlled amplifier serves as the basis for our auto-ranging algorithm used for current measurement. This method allows for excellent sensitivity, but the main trade-off comes in the form of power/heat dissipation and excess components which increase the cost. Voltage measurement is achieved through an attenuation structure that has a constant ratio. This is kept constant due to the fact the voltage between the hold and neutral lines is held mainly constant at 120 V. The voltage waveform is measured to cross-reference with the current waveform. When we cross-reference these waveforms, we can calculate a total power consumption accurately; as well as a power factor due to any leading or lagging components in the load. Calculating a power factor helps us achieve the most accurate measurements of power that we will then forward to the user.

On the software side, most systems use a web application or a mobile application to present the user with their energy usage. After considering the advantages of several different approaches, we decided to focus our energy into a web-based application. This removes any OS-dependency, as a web application can be viewed on any device with a browser; opening our application up to many more devices other than just Android or iOS devices. In addition, our design requires a central network server, and the app can easily be hosted on that central hub. An example of a system already created is the TED system[1].

3.2 Proposed System Block diagram

Hardware Block Diagram

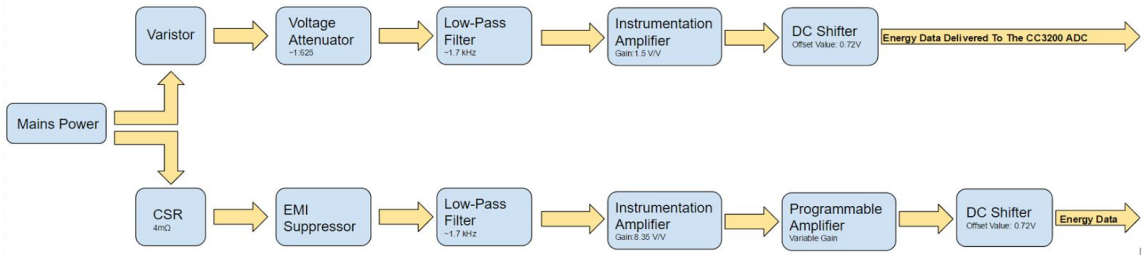


Figure 1: Sensing Circuit Block Diagram

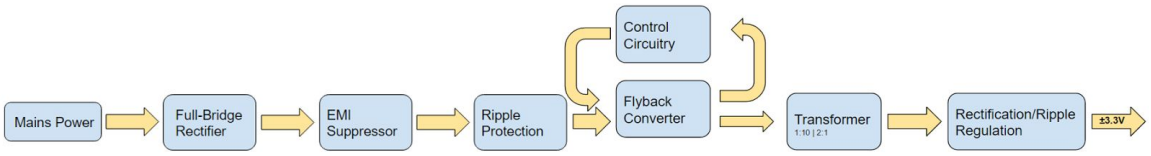


Figure 2: AC-DC Flyback Converter Block Diagram

Central Server (Network) Block Diagram

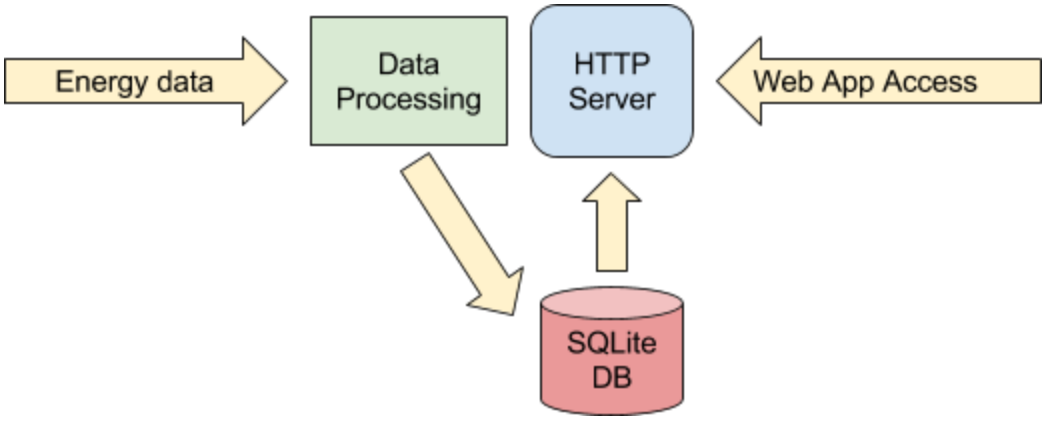


Figure 3: Software Block Diagram

3.3 Assessment of Proposed methods

3.3.1 Hardware Assessment of Proposed methods

Current sensing circuit:

The hardware portion of this project refers to the safe and accurate measurement of current & voltage. We have fuses and surge protection equipment on board to protect the user and their load in the case of a main breaker failure. When it comes to power measurement, we choose to measure on the low side of the load. We do this as to not disturb the load (i.e. avoid brownouts). Once the load has has drawn a certain current, we measure that value of current using our current sensing resistor(CSR). We do this by measuring the voltage across the resistor. Achieving this is possible through Ohm's Law. The voltage drop across our current

sensing resistor is where we begin our measurements. Here, we take the voltage of the CSR and amplify it using a differential amplifier. We then take this amplified signal and run it through a gain-varying amplifier and a DC offset buffer.

We chose to use low-side measurement not only to avoid disturbing the load, but due to product availability and cost. As we alluded to earlier, measuring on the high side entails a high common mode amplifier and, when we throw in all of our other specifications; the field of usable devices becomes restrictive. Therefore, we are chose to move forward measuring on the low side. However, this is not a huge issue. Usually, when one measures on the low side of the load, it will will cause some instability since they are lifting that terminal off of the common node. But, the real worry comes from not being able to sense short circuits.

Since we are dealing with household power, most circuits are self-regulating, in the fact that they will break the current path internally if a current greater than the max rated continuous current. Furthermore, a lot of modern breaker panels consist of GFI switches; these switches will break the circuit path if it sees any current diverting from the path of the load. The current ratings of these single breakers tend to be 15A RMS, which corresponds to our circuit's maximum measurable continuous current. However, some breakers are rated for 20A RMS. To protect our device in those cases, we included our own internal fuse that takes action when a continuous current greater than 15A RMS is detected. However, in our research of common household products that may be used with our power sensing meter; very few products exceeded 15A RMS continuous current. Hence, we do not expect issues in that aspect. Now that the issue of short circuit is mitigated; the differences in measuring on the high side versus the low side become trivial.

The remaining problem in the circuit is to lessen the effect of start-up transients. Since nearly all loads have surge protection components on board, we mainly have to worry about protecting our own device. This can be done by adding a few components like capacitors, diodes, etc.

Now that we have addressed safety issues, we can move on to ensuring compatibility with our microprocessor, the CC3200MOD from Texas Instruments. Looking at the output of the differential amplifier on the current sensing branch, we have a current dependent voltage waveform. We have set up the amplifiers such that a max current will correspond to a voltage amplitude equal to the max input voltage of our microprocessor's ADC, 1.4V. However, for currents less than the max rated continuous current, we implement the microprocessor controlled variable gain amplifier to better make use of the input range of the ADC.

When the microprocessor ADC voltage is under a certain threshold of 1.4V, the microprocessor will implement an auto-range algorithm to find the best gain to measure the waveform. This information is then feedbacked into our power calculations for accurate data logging. For a visual representation of the schematic up to the input of the ADC, refer to Appendix IV. Going forward, all tasks are executed by the software components.

Microcontroller Unit:

When designing a low-power consumption device with network capabilities, the CC3200 microcontroller from Texas Instruments is an optimal solution. However, the nature of our design involves wireless communication, which makes power management challenging. Therefore, a low-power MCU with a wifi shield is needed. The CC3200MOD is proposed for our design because it has all the features we desire in compact footprint. With the ARM Cortex-M4, we can easily handle all the computation, interrupts from the sensor, and power management. The built-in Wifi provides reliable connectivity and an easy interface for the programmer.

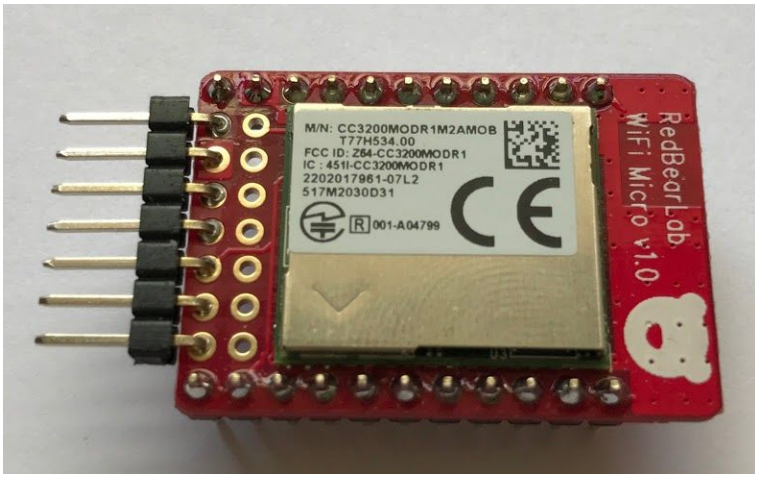


Figure 4: CC3200MOD Wifi Micro

3.3.2 Network Assessment of Proposed methods

We define the 'network' portion of the project to refer to the data processing and receiving code, HTTP server, and database system. As with all web-based applications, there are many available approaches that we could choose.

The data processing code is responsible for receiving (over WiFi) all the data packets from the monitoring stations. The data is sent over a network, so we essentially have two options for data transmission, UDP and TCP. TCP is a connection-oriented protocol that offers several error-handling utilities. UDP is connectionless and offers no guarantee for delivery, but it is the one we will be using. It is well-suited for data streaming and it is simpler and easier to implement.

The HTTP server portion of the system also includes the web application it is hosting. This part of the project is fairly straightforward, as there are a few industry-standard servers available, like Apache. The specific server we choose isn't important, so we will most likely stick with what is installed on our machine. We have chosen to host the server on the user's computer.

For the database system, we are presented with many different options. Eventually, we settled on using SQLite. SQLite is a single-file database that is accessed like a file, directly from the disk. This eliminates the need for a separate database server/connection, making our overall system simpler. It is also a single file, making it easy to backup and transfer between machines. To create a database, we store three variables in a file or separate files: timestamp, the id of the station that recorded it, and the raw value. We use integer type to store timestamp and id, real type to store raw value. The limit range of these values are 12 bits.

3.4 Validation

3.4.1 Network Validation

To verify the network portion of the project, we will have several different testing strategies, one for each portion of bigger component.

To test the UDP server that processes the incoming power measurements, we sent dummy packets to the central hub via a test device, at the specified UDP address. This allowed us to make sure the packets were being sent perfectly, in addition to testing how many

simultaneous packets can be received before overloading occurs. We eventually determined that overloading tests were unnecessary, as the relative rate of data transmission is low.

To test the database, we wrote a simple Python script that injects artificial data into the database. We then wrote another test script that reads the data similar to how the web server will. These tests were very simple and verified that the database format was correct.

3.4.2 Hardware Validation

Power monitoring component

Software Validation: Using Multisim, we were able to simulate the functionality of our circuit to achieve software validation before physically realizing the circuit. Multisim is a very power simulation tool which allows us to perform interactive simulation. In addition, Multisim has a large database of components so that we were able to model the system accurately.

Functionality: The functionality and accuracy of our power measuring device will rely mainly on the performance of the current sensing component. It is safe to assume that because the power comes from the utility company, it has very strict regulations in maintaining 120V RMS at 60Hz. However, we still need to be prepared for small nominal changes. The validating process will have two components:

1) Functionality, where we will be using a commercial ACS712 sensor to compare the results. (The ACS712 was a previous measurement method studied by our team. For more information about the ACS712 method, refer to Appendix II.)
2) Accuracy, where we will be using the Agilent 34401a multimeter to compare the measured values versus expected values.

Accuracy: One way to check the accuracy of the current sensing circuit is by comparing the measured current value with the multimeter 34410A. Verifying accuracy is the biggest challenge of our design. With the multimeter 34410A, we can validate the accuracy in the range from 0A to 10A RMS. Current above 10A will be validated mathematically using component models obtained from lower range measurements. Basically, once we test the circuit in the range from 0A to 10A, we can characterize the resistor properly. At the same time, we will be able to account for the resistance of internal wires of the system, DC offset voltage of the amplifier, shunt resistance degradation, etc.

Set of components to be tested: There are two types of devices that we will test: analog and digital. The reason that we have two sets of components is because their power consumption patterns vary. For most analog devices, they draw current continuously. On the other hand, digital devices draw current at partial duty cycle.

Analog loads:

1) Calibrated Resistive Load (Rated Power Consumption: 10W)
2) Incandescent light bulb (Rated Power Consumption: 60W)
3) Clothes Iron (Rated Power Consumption: 1200W)

Digital Loads:

1) iPhone charger (Rated power consumption 5W)
2) Macbook charger (Rated power consumption 60W)

4 Project Requirements/Specifications

4.1 Functional

4.1.1 Web Application System Requirements

1. The web application shall allow the user to change the period of energy data collection
2. The web application shall show the user energy graphs over a selectable time range
3. The web application shall allow the user to give the monitoring station a user-friendly name
4. The web application shall retrieve its' data from a central database
5. The web application shall read power with at most a 5% error
6. The web application shall fulfill all these requirements in the Chrome browser

4.1.2 Central Data Processing Requirements

1. The data processor shall receive data from the connected monitoring station
2. The data processor shall convert the data from X units into Y units
3. The data processor shall store each data point, along with a timestamp, into the central database
4. The data processor shall be able to receive simultaneous transmissions from at least 10 monitoring stations

4.1.3 Hardware System Requirements

1. The hardware shall not be a source of significant power drain (power consumption less than 5W).
2. The differential circuitry must be able to maintain a bidirectional current with an RMS-amplitude of 15A.
3. The hardware will have a current floor of 100mA.
4. The hardware shall be sensitive to changes in current and voltage measurement to the value of, at least, 100mV/A.
5. The hardware shall not make the load operate in conditions that are harmful to it and/or affect performance of the circuit.
6. The hardware shall be able to provide an open circuit in the event of operation outside absolute maximum ratings (max current is rated at 15A).
7. The hardware shall provide a output with minimal phase shift and no frequency modulation.
8. The hardware shall operate in temperate range from -25℃ to 80℃.

4.1.4 Case Requirements

1. Design shall be created using SolidWorks 2016.
2. Manufactured using 3D printing.
3. Material must be strong enough to protect circuit inside and not break while devices are plugged in and out of the case outlet.
4. Ventilation must be available for cooling of the circuitry.

4.2 Non Functional Requirements

4.2.2 Hardware Non-functional Requirements

The hardware should be able to fit in a package that is non-intrusive to other devices on the electrical outlet. Additionally, we are aiming to be IP22 compatible; as are all modern electrical sockets. This device should also not produce any obtrusive audible noise when it is

in on, or standby mode. The overall goal is to produce a device that is negligibly intrusive to the user.

4.2.2 Web App Non-Functional Requirements

The web app allows the user to view the power consumption of the device in a graph. The web app should be modern and well-designed, with a sensible UI and easy to use controls. Commercially-available designs set a high standard for usability and our application should be no different. We want our web app to be easy to read and use so navigating the application will seem trivial to the user.

4.2.3 Case Non-Functional Requirements

We wanted to create a case that would secure our PCB and CC3200, as well as look professionally appealing for a possible future product.

5 Challenges

Three biggest challenges that we will be facing when implementing the hardware part of the energy measurement device are:

Power Consumption:

Low-power consumption is a common characteristic of the many electronic devices these days. Without a doubt, engineers have taken this challenge seriously. In our project, however, building an accurate energy measurement to measure low power consuming devices is much more challenging. We want to minimize the design so that the overall energy consumption of the device must be lower than 5W. This number is based on the typical power consumption of a phone charger. To achieve this specification, we have to be very careful with passive components like resistors, microcontrollers, and wireless communication modules. The obvious trade-off would be between the functionality and power consumption. Critical tasks like signal processing and data transferring can not be further simplified. Thus, we are pretty much relying on the software algorithm to improve power management.

A rough estimation of power consumption will give us some ideas to see what we can do to improve the power consumption of the whole device. However, the power consumption of the shunt resistor will stay constant. The plot below shows the power consumption of a 4 mΩ resistor at the minimum current magnitude of 100 mA and maximum current magnitude of 15A RMS. At the max current measurement, the power consumption of the resistor will dominate (about 1W). At small current measurements, the power consumption of the resistor is relatively small (microwatts). The power consumption of the MCU will dominate so we can perform power gating to the reduce power consumption.

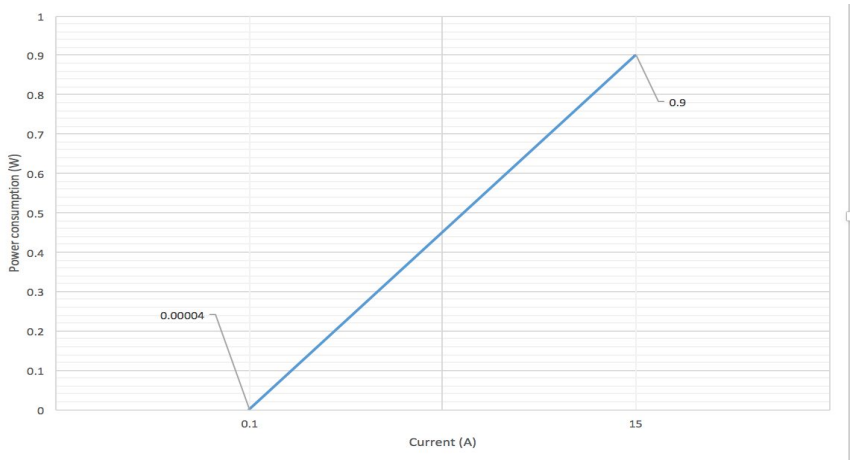


Figure 5: Sensing Resistor Power Consumption

Noise:

When dealing with measuring signals for verification, noise is major issue. Noise mainly comes from components themselves. We started addressing the noise issue by replacing the ACS712 sensor with the resistor current sensing circuit. However, there is still background noise in the system. And, at low current levels, the noise level can be quite noticeable. Our aim is to increase the signal-to-noise ratio. In order to achieve this goal, we first need to account for the amount of noise by performing noise analysis. There are two prominent types of noise in our the current sensing circuit. Broadband noise and flicker noise must be analyzed carefully. However, since we know that most types of noise have a Gaussian distributions; we can mitigate the effect of noise in our system by implementing a averaging filter. However, we must design the filter carefully so that we don't lose any data.

Accuracy:

The energy measurement that our team is designing intends to handle the current ranging from -15A to 15A, RMS. This is a relatively large current range. Our approximate sensitivity is 4V/30A, which is approximately 133 mV/A. With a small sensitivity, the accuracy of the measurement relies pretty much on the Analog-to-Digital converter, in the sense that the ADC must be capable of converting relatively small decimal voltage. If we measure high current values, there is nothing to be concerned about because the output analog signal can be easily read by the microcontroller. However, once we reach the level the of milliamp current, the accuracy reduces dramatically. This is the level where the noise dominates the signals in our circuit. With the programmable gain amplifier, the MCU will have the capability to amplify the output signal to increase the resolution. However, because all the programmable gain amplifiers are digitally controlled; the gain will be discrete instead of the continuous. This constraint will cause a few ranges to have lower resolution.

Phase Shift:

Phase shift must be modeled for our system to have accurate power calculations. If we didn't model it, we could arrive with incorrect power factor calculations; hence, making the user think the device is consuming more/less power than it actually is.

Network Connection:

Some challenges we will be having is the ability to connect to a WiFi network that has a security passcode. For now, we are assuming that there will not be a security passcode, but we will need to incorporate secure networks to be able to please a wider range of users. Some other challenges we may consider including would be the strength of the wifi connection to our CC3200 microcontroller, for one.

Data Handshaking:
An initial challenge was having multiple energy measuring devices running at the same time. The challenge occurs when two or more units send data at the exact same point in time and how the receiver will be able to choose to interpret one, and when that's finished interpret the remaining. Otherwise only one signal could be recognized or there could be an error in the system not allowing the signal to go through. However, we escaped this problem by using a microcontroller with a full network stack. All data transfer was handled using the UDP protocol, so the server receiving the packets will be able to sort through many packets received at the same time.

ADC:
At the end of our hardware circuit, we will have an analog signal. In order to store this data into our database, we will need to convert this signal from analog to digital. We do this through our CC3200 microcontroller. We will be using the datasheet to look at the appropriate channels to use for the ADC; however, some challenges may occur when dealing with high noise or low voltage signals as far as transferring that signal over. Whenever an ADC is used you need to look at quantization errors that happen when putting a continuous signal into discrete steps. In our case, we are using a 12-bit ADC with a 1.46 V scale. This gives each "step" 0.36 mV of range. This is well-within our performance limits.

6 Implementation

6.1 Hardware implementation

The circuit schematic was designed in Multisim. Multisim offers an extensive library of components so it's more convenient to do simulations. In our proposed circuit, we are using a 4 mΩ resistor on the low side of our circuit. This resistor produces a voltage drop across it proportional to the current that the load draws. Through a series of controlled amplifiers, we can vary the gain to detect currents ranging from 100 mA - 15 A. Our circuit also tracks an attenuated line-voltage waveform to cross-reference against the current waveform and calculate a power factor.

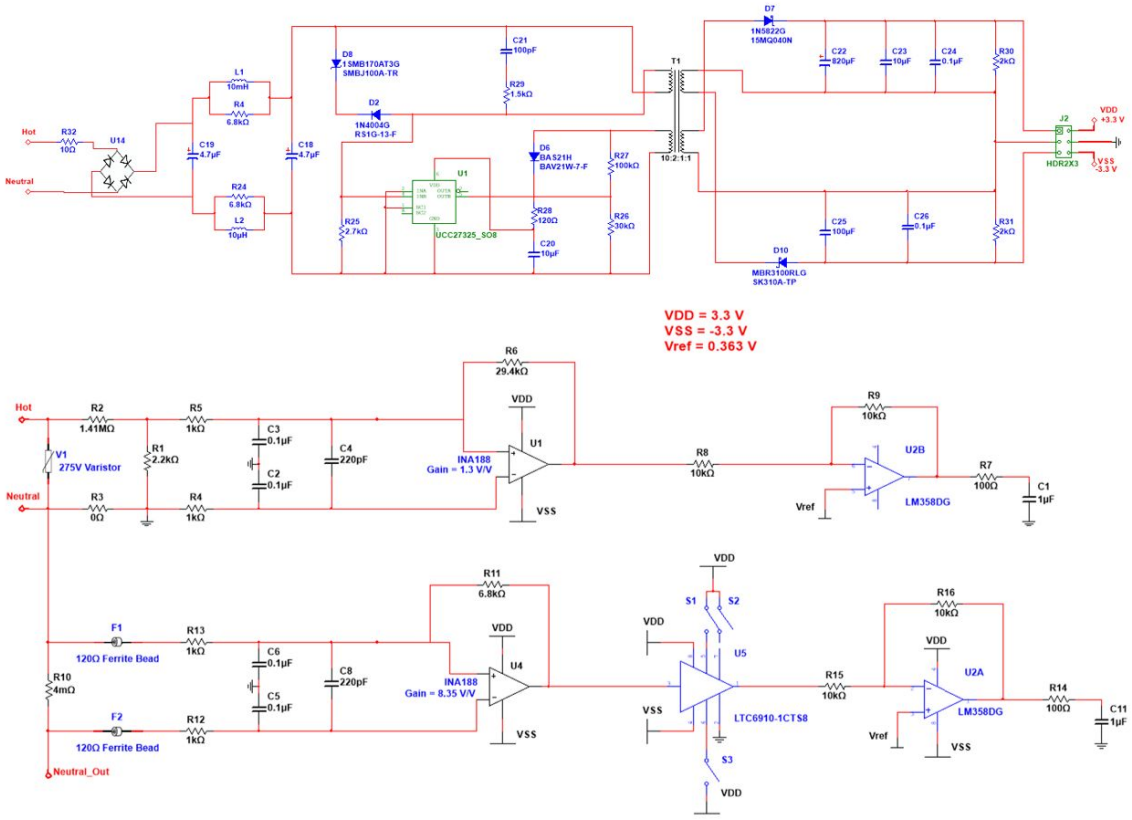


Figure 6: Circuit Design in Multisim

For voltage sensing, we used the INA188 instrumentation amplifier from Texas Instruments. This particular amplifier uses auto-zeroing techniques to achieve low offset voltage. The gain can be controlled by only a single resistor. The three op-amp design offers a rail-to-rail output, which is needed for the data collection since the power line swings between negative and positive voltage.

For DC offset, we used the LM358 amplifier from Texas Instruments. The basic configuration was inverting amplifier with Vref on the noninverting terminal. Vref is connected directly to a voltage divider that has 0.32 V, which shifts the input signal to 0.72 V.

For the autoranging feature on the current measurement, we placed a LTC6910 right between the INA188 and LM358. The gain of LTC6910 can be controlled by three digital input signals. They can be connected directly to CC3200MOD without any extra circuitry. However, there is still a need to suppress the noise coming in from the the power line and the loads. We used two low-pass differential filters with a cut-off frequency of 1.5 kHz; one after the sensing resistor and one after the voltage divider.

The ±3.3V supplies that power the circuit come from using flyback topology courtesy of the TI UCC288910. The reference design is reliable and easy to modify.

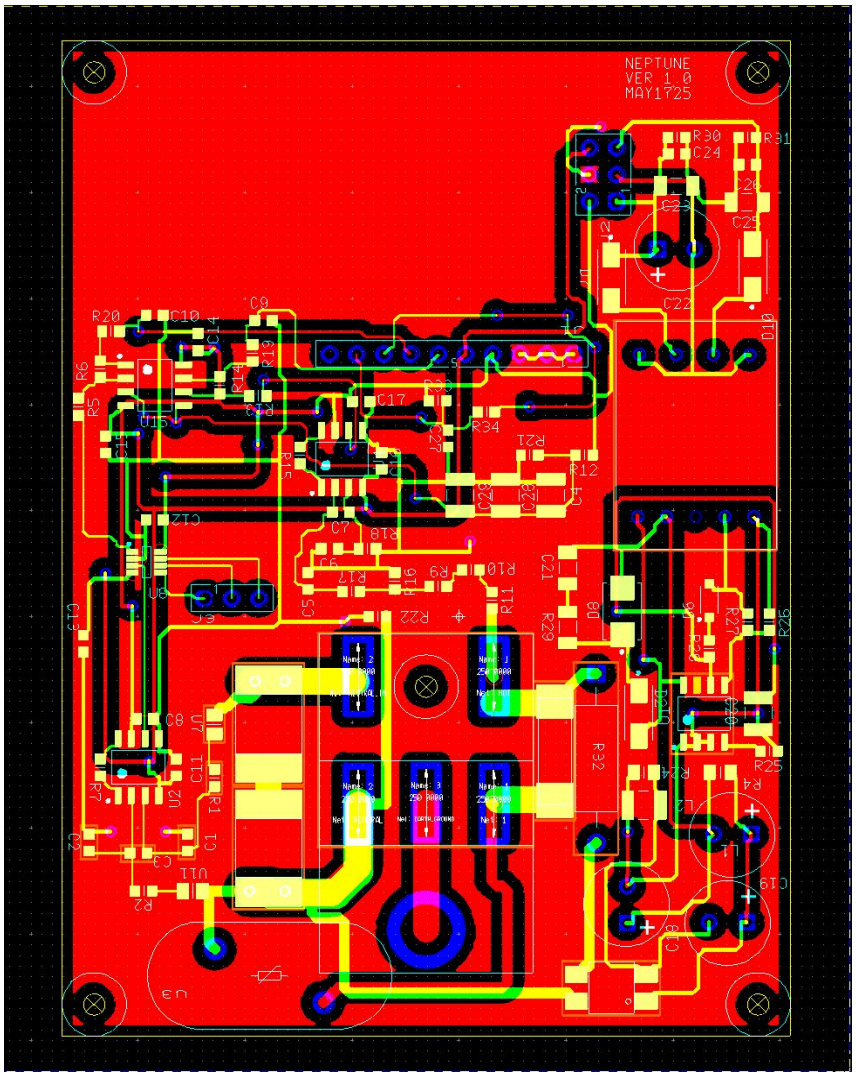


Figure 7: PCB layout in Ultiboard

The layout was designed in Ultiboard. Using the exported netlist from Multisim, we can easily use the auto-routing feature to connect all the traces. For the ground, we created a ground plane on the second layer of the PCB. We performed both DRC and connectivity check before sending out the design to a PCB maker.

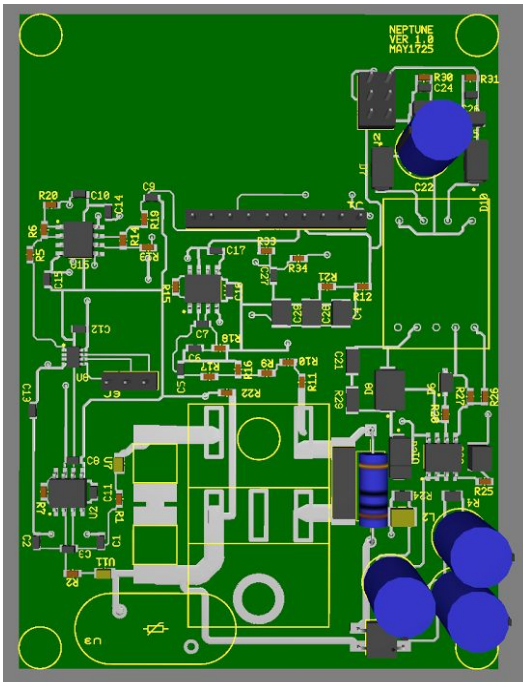


Figure 8: 3D rendered PCB

The 3D PCB was rendered to ensure that geometrical dimensions turn out to be what we intended. The 3D version of components superimposed on top of the PCB, we checked for any overlaying conflicts between components

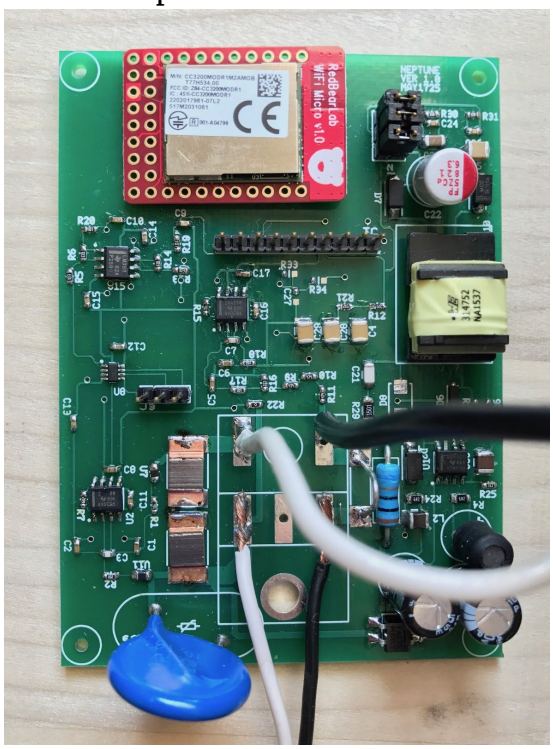


Figure 9: Physically Realized PCB

Once we received the PCB and components, we started soldering and checked connectivity at each joint using Multimeter.

6.2 Software implementation

As you can probably tell, the hardware side of the project is very extensive and took the bulk of the effort in this project. The software was much simpler for a few reasons. First, much of the "calculation" of the power was done in the hardware, so only simple software is needed to finish that section. Secondly, libraries and API's were abundant throughout the whole system. TI provided a sample application that almost completely satisfied our requirements, so only minimal C code was needed to finish it. The web application also made use of third-party libraries (all licenses are OK), so the code for that portion was simple as well. The software is all very self-explanatory, so not much can be written about it. Specific implementation details can be found in the code and comments in section 12.4.

6.2.1 Database Implementation

For the database portion of the software, we followed through with our initial decision to use SQLite. It is a connectionless database, meaning it requires no host server. It is also stored in a single file on the disk drive. Although it's a single file, it allows multiple "connections", thus solving the problem of reading to the database while the data processing software is writing to it, if a collision should ever occur.

We designed our software with a final product in mind. SQLite is a great choice because the database is stored in a single file, meaning it's very easy to create copies and backups. Thus, the user could easily back up years of power data in a single file.

6.2.2 Microcontroller Implementation

On the microcontroller (CC3200), we had to implement the code in a TI-RTOS environment. This software provided by TI gave us almost everything we needed to make our software. The system includes code to communicate on a WiFi network, read from the ADC, and use the GPIO pins. We were able to combine these API's with minimal extra code to make our whole system.

6.2.2.1 Network Connection

On startup, the board connects to the user's WiFi network. All that is needed is the network name and password.

6.2.2.2 ADC Code

To read from the ADC, we simply used the sample code as a baseline. The ADC is extremely simple to use: you simply turn the ADC on and read the values from a register using a built-in function call. Of course, we do this twice. One ADC channel reads the VSENSE voltage to calculate the voltage, and the other ADC channel reads the current reading. Once we have the voltages, we divide by the amplifier gains to get the actual current and voltage readings.

6.2.2.3 RMS Algorithm

Once we have the current and voltage, we calculate the RMS by using a one-step Newton-Rhapson method. Essentially, this algorithm uses a running average to find the

RMS value of a sequence of samples. Once we have the RMS current and voltage, we simply multiply them, along with the power factor, together to get the power.

6.2.2.4 UDP Transmitter

Once the power value is ready (once every second), we simply send the power data in a UDP packet. Also contained in the packet is a device-specific identifier which separates each power station from the rest. Again, TI API's are used to send this packet to a specific address belonging to the central hub.

Again, as you can see, the software on the microcontroller was extremely minimal. We made extensive use of TI API's. More details can be found in the code appendix.

6.2.3 Web Application

The final piece of software is the web application. The web application is written purely in Javascript and makes extensive use of third-party libraries. As a result, the code we wrote was very small. We made use of a third-party library called Highcharts to implement the charts. We also used a UI library called Semantic UI to make a user interface that is modern and very attractive. As a result, we have only a few hundred lines of boilerplate Javascript. The code for the entire web application can be found in the appendix. A screenshot of the completed website in action can be seen below:

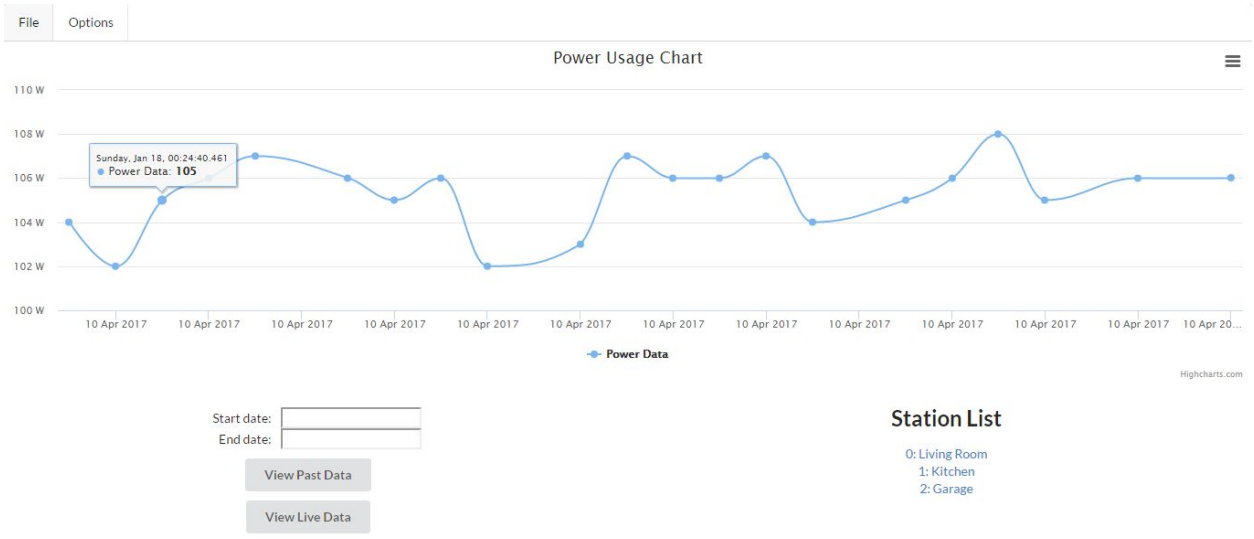


Figure 10: Web Application

6.3 Case implementation

We used SolidWorks 2016 to design our case. We made the case deep enough to house all of the internal wiring from the outlet to our PCB, but not too large to where it would be impractical. Our PCB has wiring which could be dangerous if people were able to come into contact with it, so the case adds a needed protection from our circuit. We designed the case such that it acts like an extension cord for the outlet. We wanted to have an extension cable on the case so both ports of the wall outlet wouldn't be blocked, and the user could plug the device in without much difficulty, but also be able to access the physical case itself very easily. We also created lips around the top and bottom parts of the case to allow the case to

snap together. As well as two screws to be used to secure the two parts together even further. The PCB is mounted to 4 pegs on the bottom part of the case. We also added slots on the sides of the case to allow for ventilation and cooling of the internals of our device.

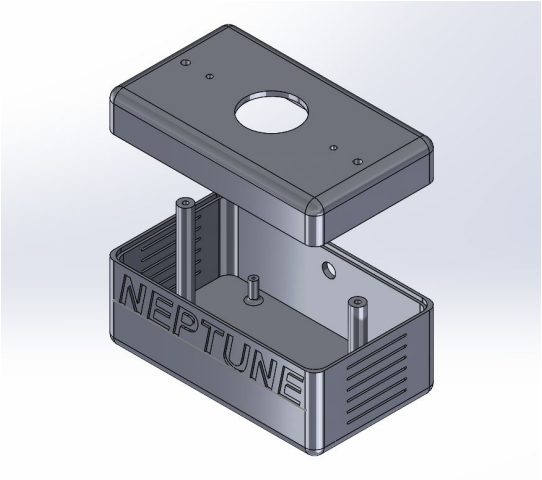


Figure 11: 3D Printed Case

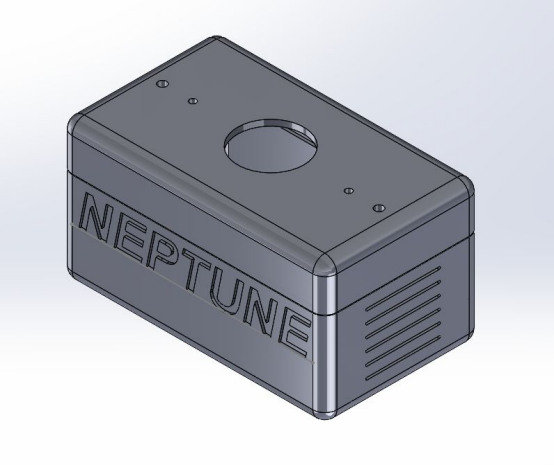


Figure 12: Closed Case

We also have a picture of the completed case with the PCB installed:

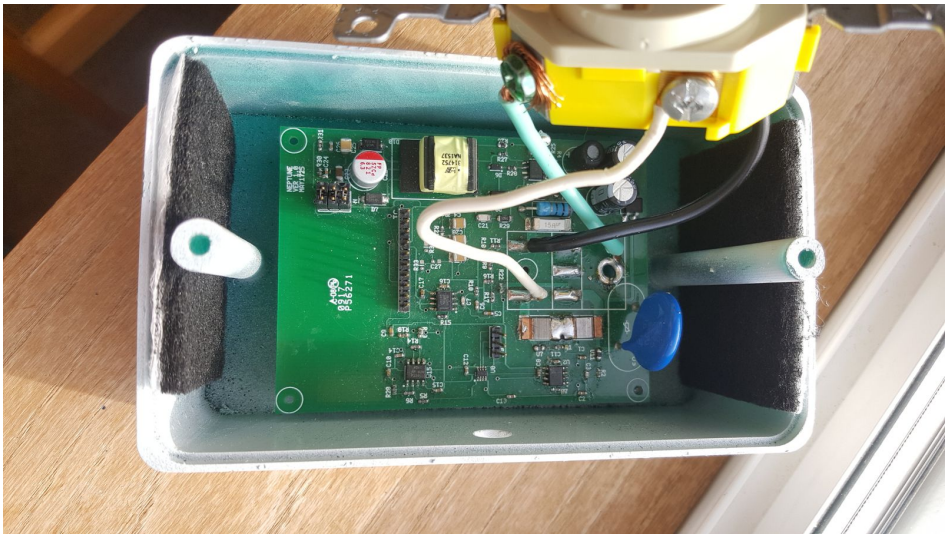


Figure 13: Case with PCB installed

7 Testing and Results

7.1 Hardware

Using a NEMA 5-15 interface, we were able to simulate how our circuit would be connected in the household setting. When we plug in a load, we can collect data off our the circuit's current sensing and voltage sensing pins using the CC3200. All the collected data will be imported into Matlab for signal analysis.

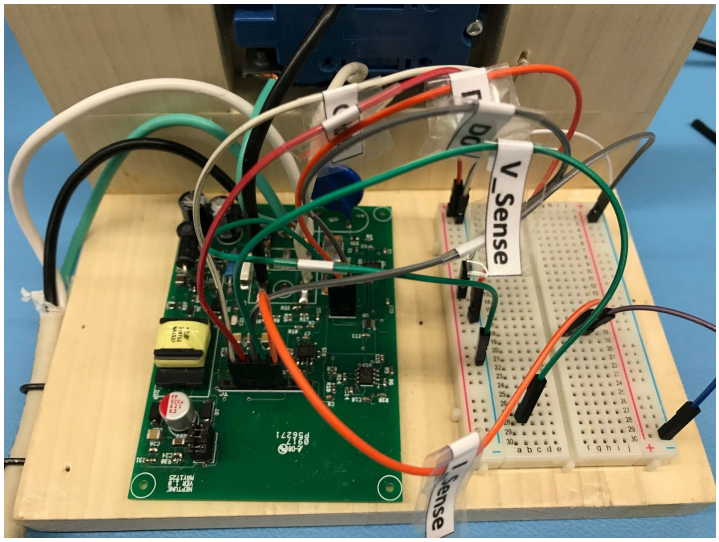


Figure 14: Testing Prototype - Back

Figure 15: Testing Prototype - Front

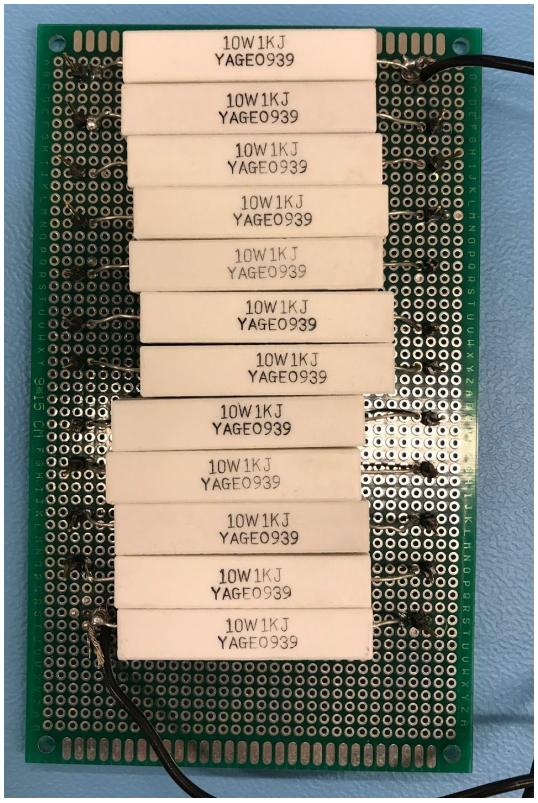


Figure 16: Calibration Load

To test these items, we first place them in series with the ammeter and record the current the device draws. Then we place the same device in-line with our current sensing circuit. Once this is done, we can look at the output voltage waveform of the current sensing branch, Isense, and make the appropriate calculations to see if the amount of current we measured is the same as the amount that is expected as measured by the multimeter. A similar process was done to test the voltage sensing branch, Vsense.

7.1.1 Phase Shift Measurement

ADC phase shift

According to CC3200MOD datasheet, the phase shift from the CC3200's ADC round robin sampling contributes to an additional $4\mu s$

Sensing circuit phase shift

In our system, we found that phase shift will be of little concern. The phase shift between the voltage sensing pin and the current sensing pin is in the range of hundredths of a degree.All together, the maximum time delay can be capped at $7.5\mu s$; which will be barely detectable by our setup.

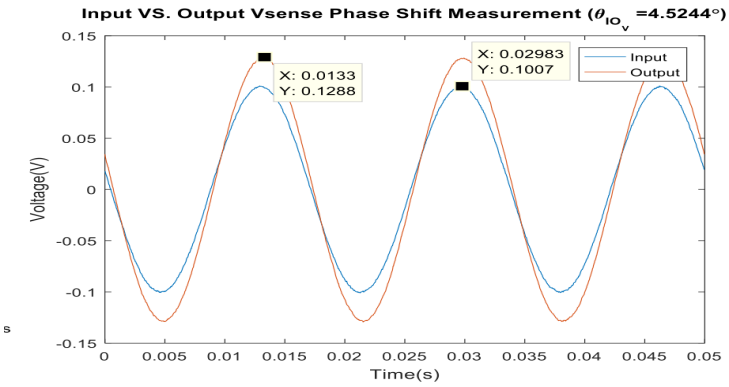


Figure 17: Vsense (voltage) vs. Vin

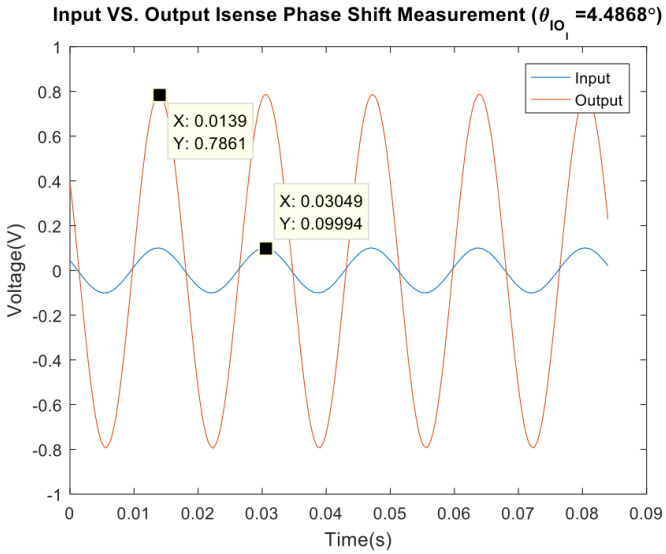


Figure 18: Vsense (current) vs. Vin

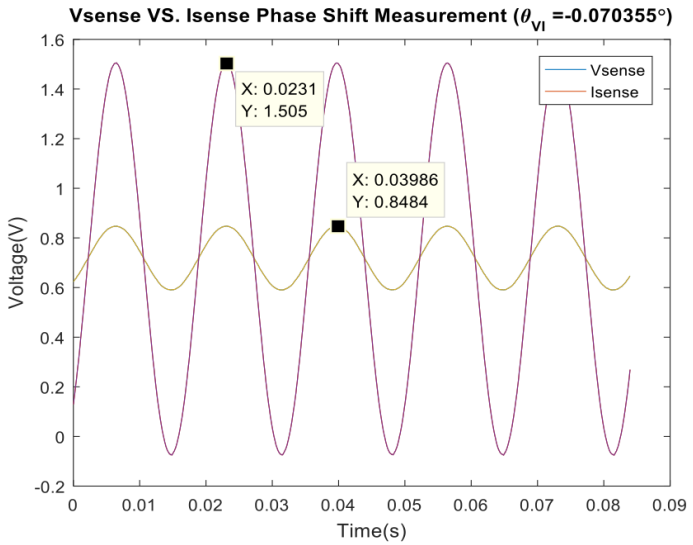


Figure 19: Vsense vs. Isense

7.1.2 Voltage measurement

Resistance	**N/A**
Power Rating	N/A
Expected current	N/A
Sampling Frequency	2.75 kHz
Number of sample	500

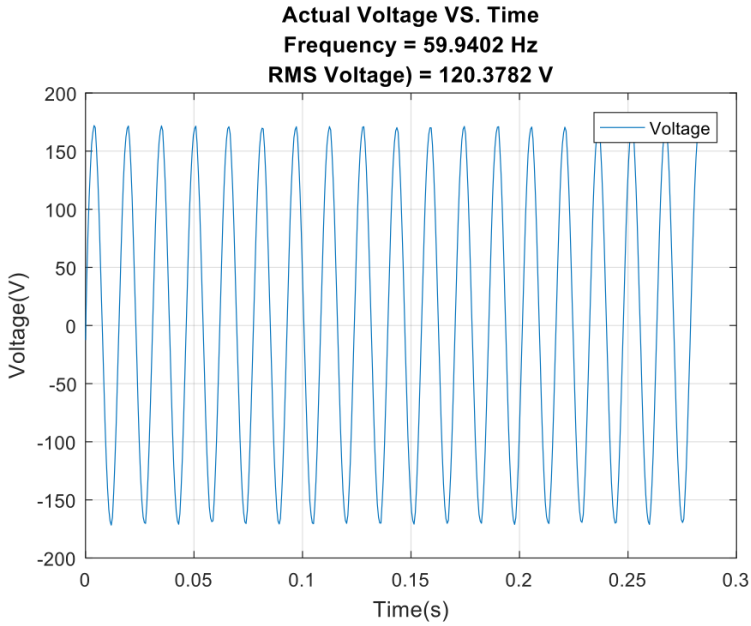


Figure 20: Voltage vs. Time

7.1.3 Current Measurement

Load 1: Calibration load

Resistance	**11.9 kOhms**
Power Rating	10 W
Sampling Frequency	2.7 kHz
Number of sample	500
Expected current	10.8 mA_{rms}

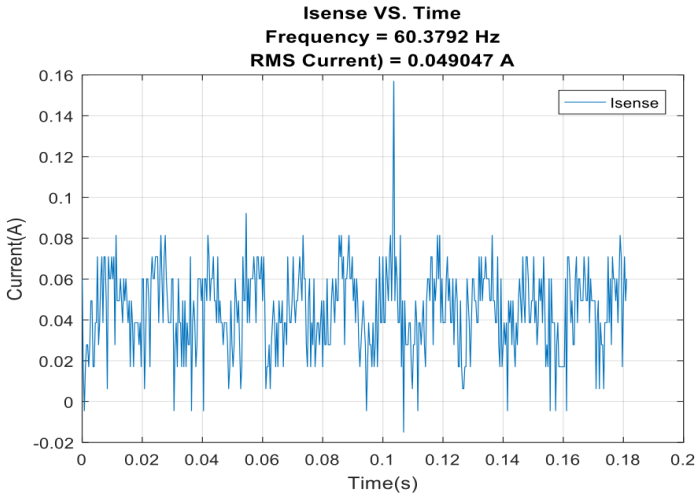


Figure 21: Using gain of 1 V/V

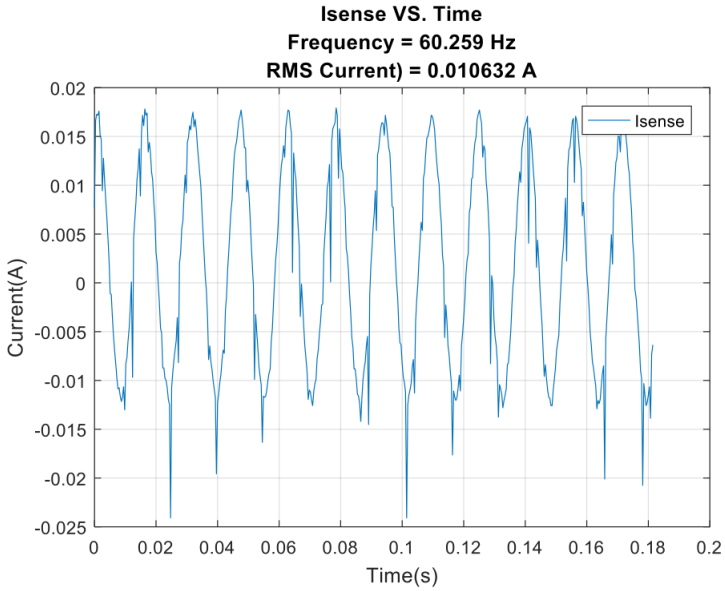


Figure 22: Using gain of 100 V/V

Load 2: Calibration Load

Resistance	**1.98 kOhms**
Power Rating	10 W
Sampling Frequency	2.7 kHz
Number of sample	500
Expected current	60.6 mA_{rms}

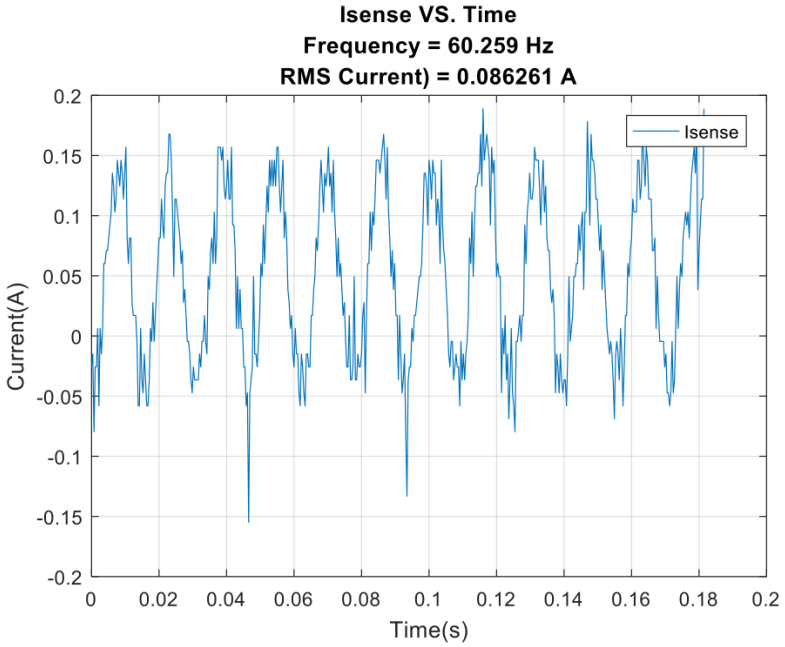


Figure 23: Using gain of 1 V/V

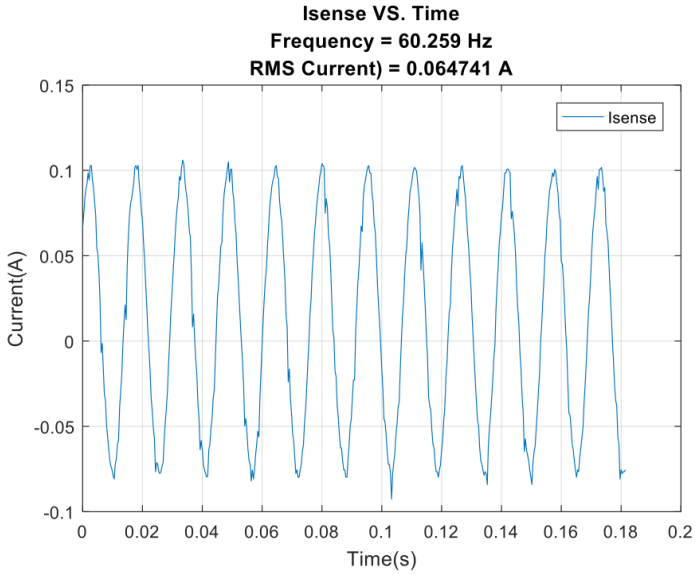


Figure 24: Using gain of 10 V/V

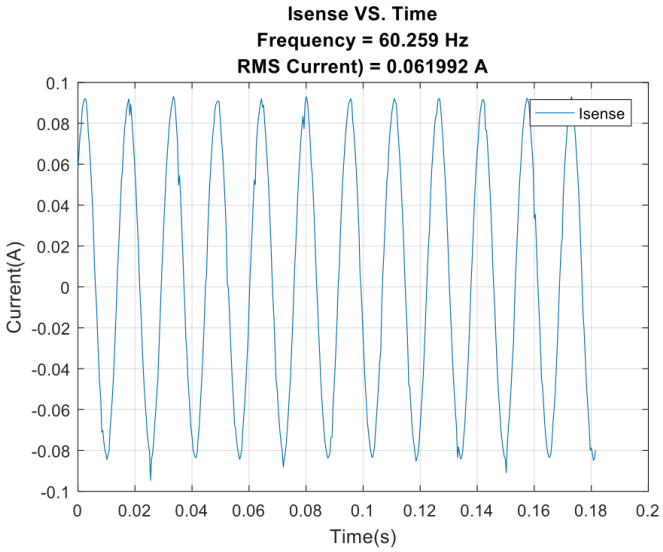


Figure 25: Using gain 100 V/V

Load 3: Calibration Load

Resistance	**1.192 kOhms**
Power Rating	10 W
Sampling Frequency	2.7 kHz
Number of sample	500
Expected current	101 mA_{rms}

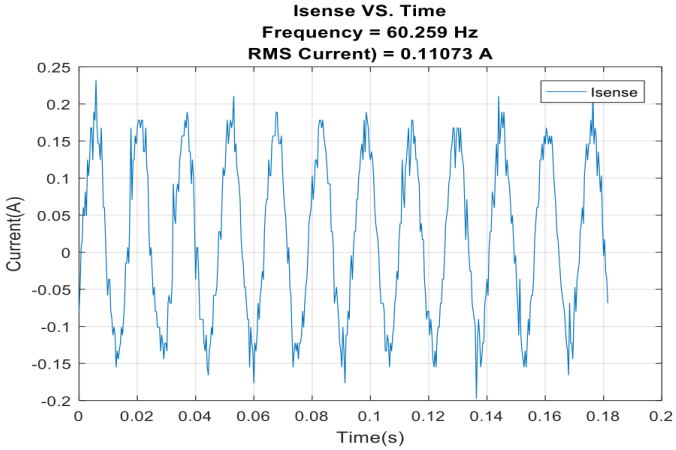


Figure 26: Using gain of 1 V/V

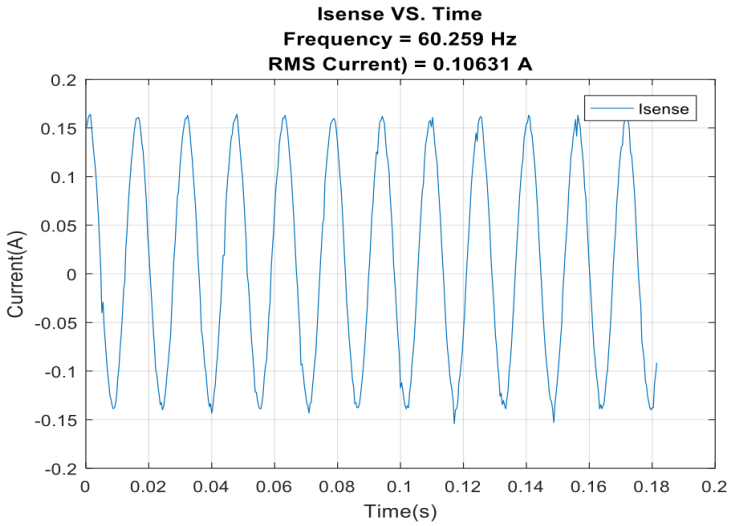


Figure 27: Using gain of 10 V/V

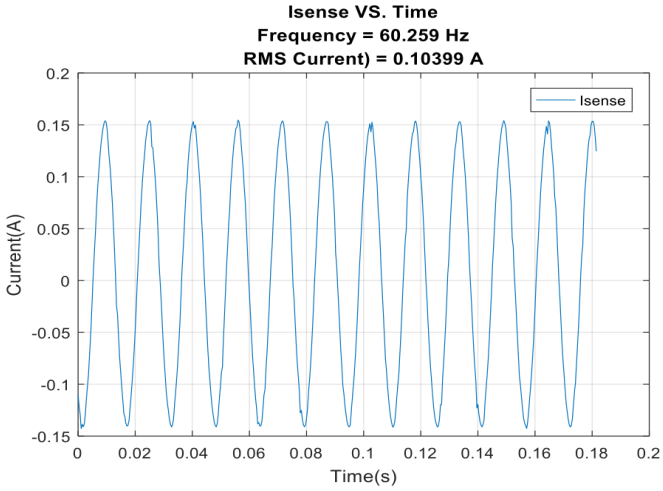


Figure 28: Using gain of 100 V/V

Load 4: LightBulb (60W)

Resistance	**240 Ohms**
Power Rating	60 W
Sampling Frequency	2.7 kHz
Number of sample	500
Expected current	0.5 A_{rms}

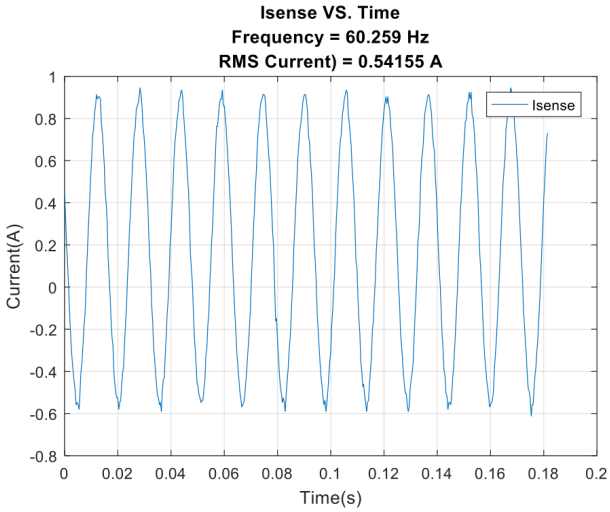


Figure 29: Using gain of 1 V/V

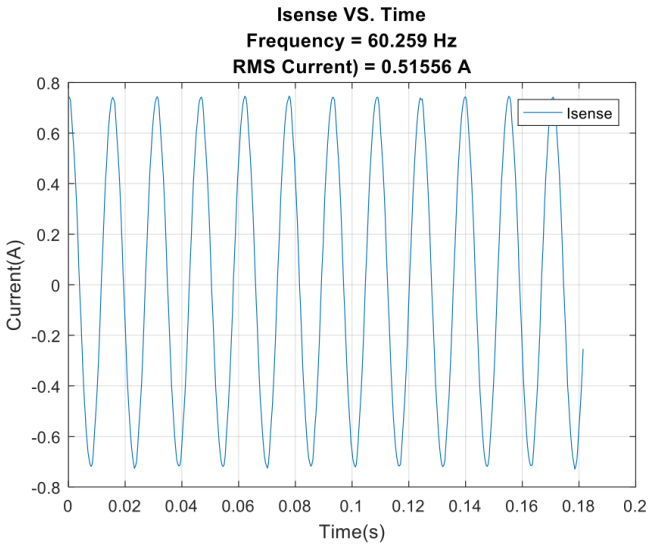


Figure 30: Using gain of 10 V/V

Load 5: Clothes Iron (1200W)

Resistance	**12 Ohms**
Power Rating	1200 W
Sampling Frequency	12.77 kHz
Number of sample	500
Expected current	10 A_{rms}

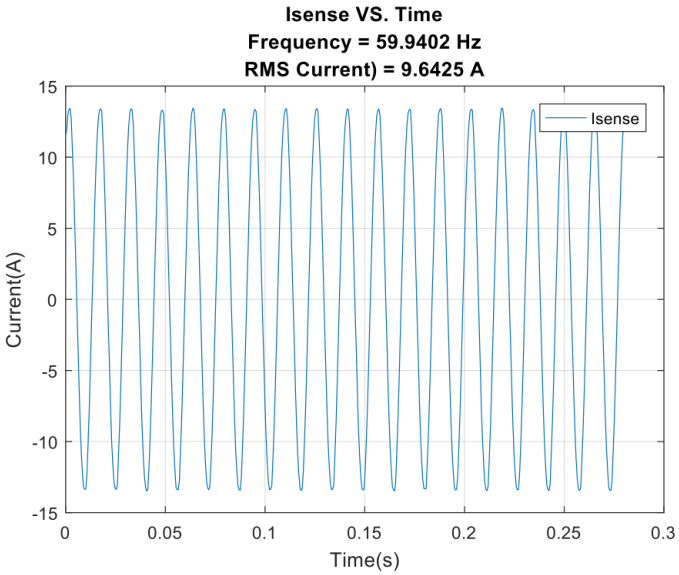


Figure 31: Using gain of 1 V/V

Load 6: iPhone charger (5W)

Resistance	**N/A (digital load)**
Power Rating	5 W
Sampling Frequency	2.7 kHz
Number of sample	500
Expected current	83 mA_{rms}

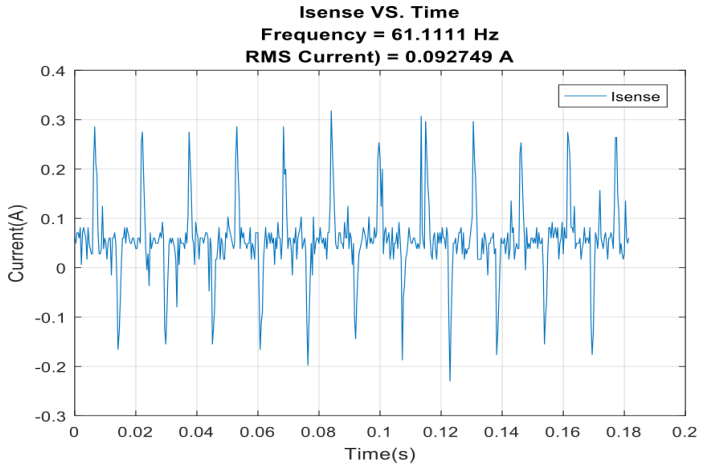


Figure 32: Using gain of 1 V/V

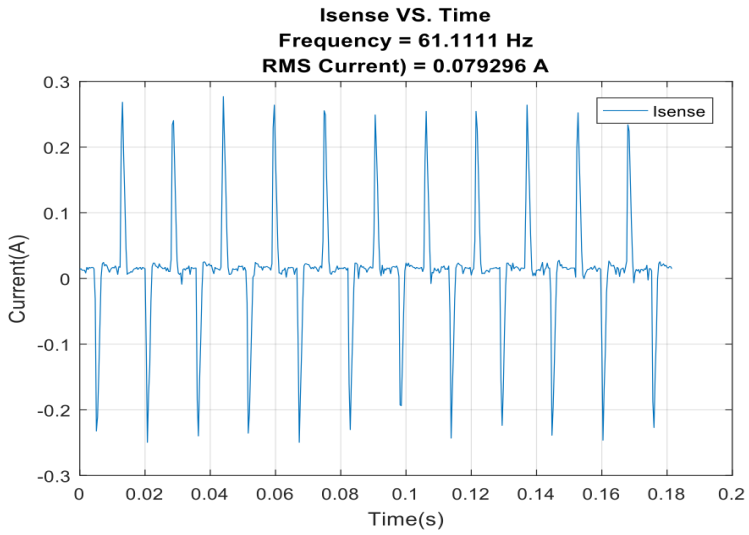


Figure 33: Using gain of 10 V/V

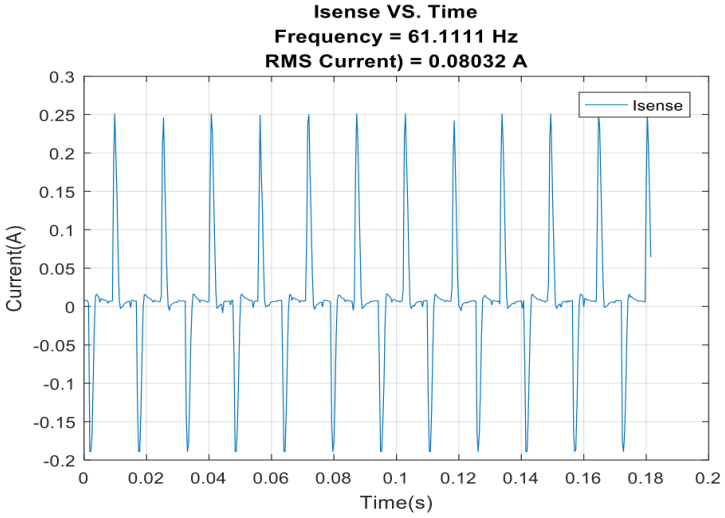


Figure 34: Using gain of 100 V/V

Load 6: Mac charger (60W)

Resistance	N/A (digital load)
Power Rating	60 W
Sampling Frequency	2.7 kHz
Number of sample	500
Expected current	1 A_{rms}

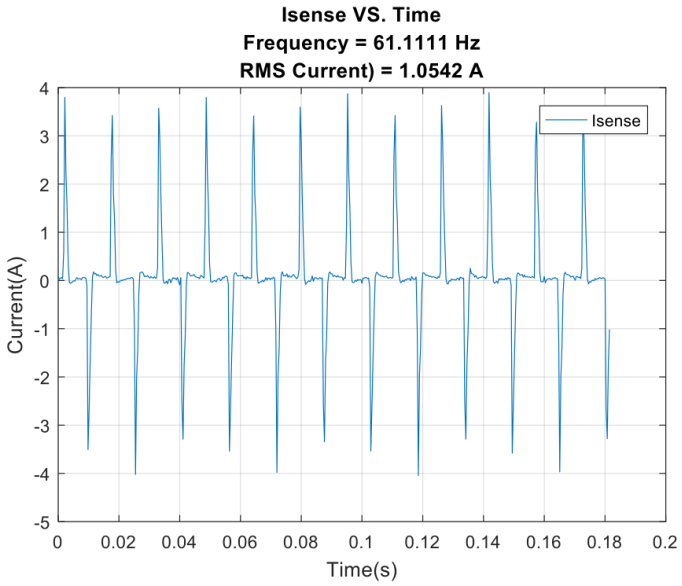


Figure 35: Using gain of 1 V/V

Actual Current	Error at G1	Error at G10	Error at G100
10.8 mA_{RMS}	354%		-2%
60.6 mA_{RMS}	42%	7%	2%
83 mA_{RMS}	12%	-4%	-3%
101 mA_{RMS}	10%	5%	3%
0.5 mA_{RMS}	8%	3%	
1 A_{RMS}	5%		
10 A_{RMS}	-4%		
Actual Voltage	Error		
120.287 V_{RMS}	1%		

Table 1: Error Summary

As we can see, with the optimal gain settings, we can achieve an accuracy of within ∓5%, which was one of our technical requirements. Additionally, we can see that there is no frequency modulation. Although, at times, the waveform may seem distorted, but that is due to the noise becoming more of a factor at low current measurements. This is expected, and, as it can be seen in graphs above, we can still arrive at an accurate result even at this low current range.

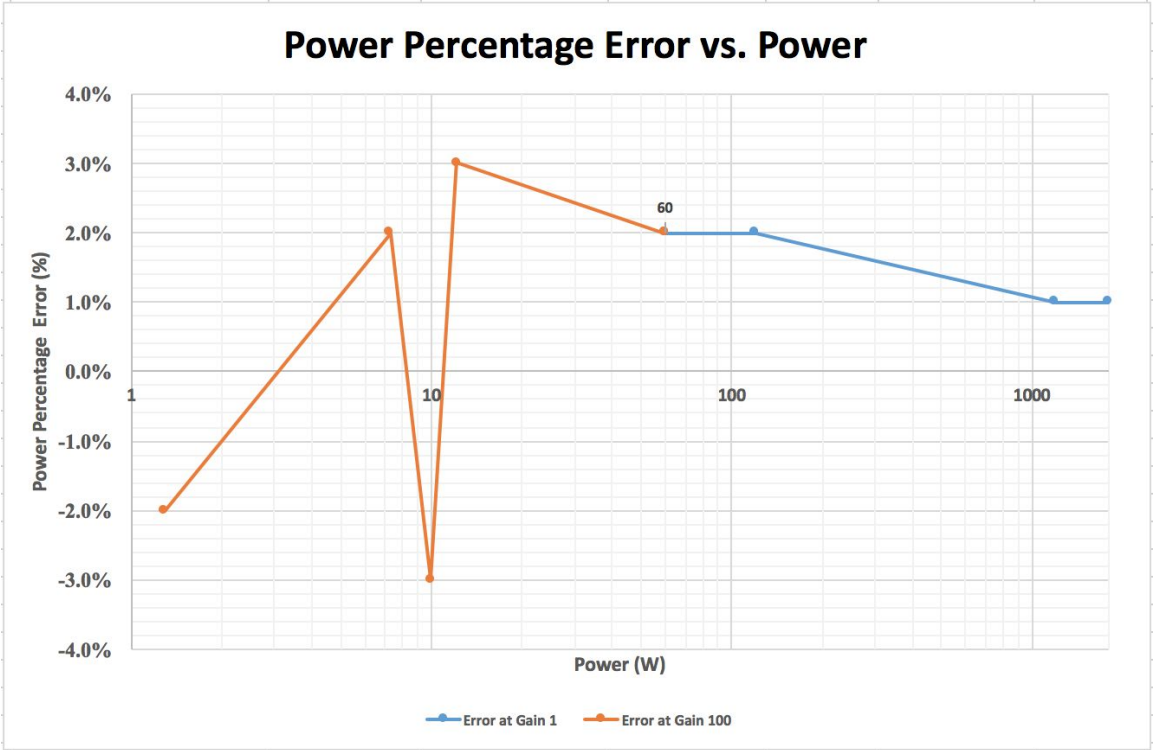


Figure 36: Power Percentage Error

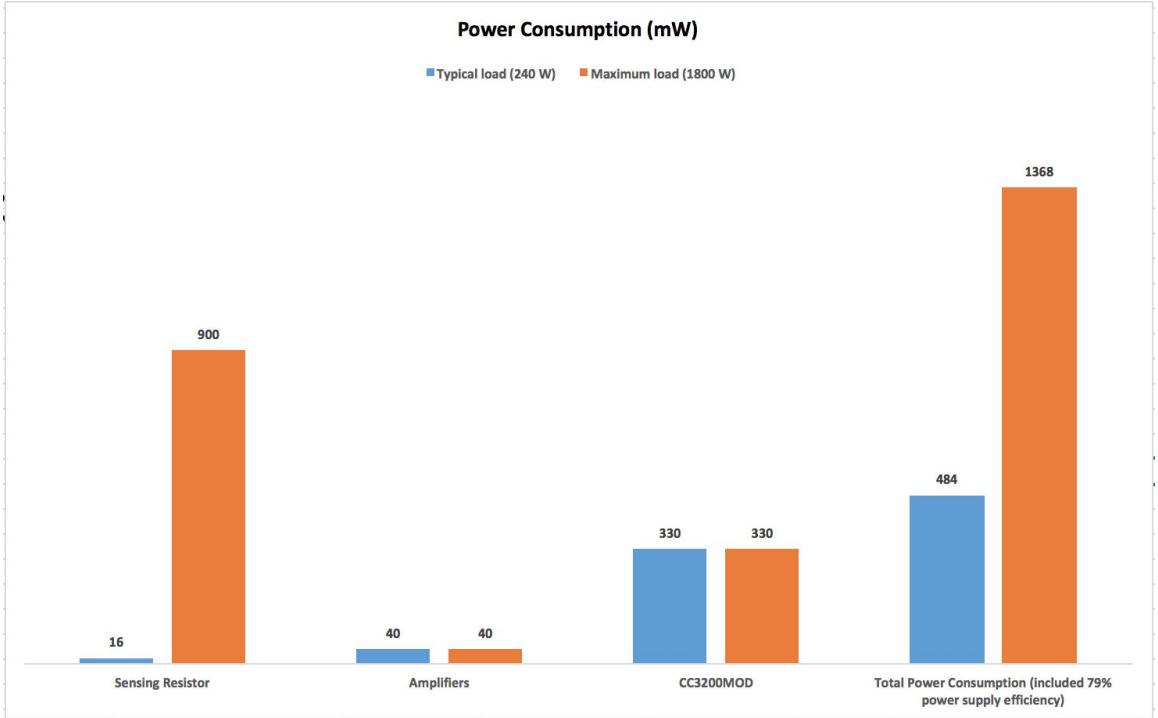


Figure 37: Power Consumption

Using a multimeter, we can measure the power consumption of the power meter itself under different scenarios: typical load and maximum load. The results can be seen in *Figure 5*. For

a typical load, our device only consumes approximately 0.5W. The maximum load tripled the power consumption due to the current sensing resistor power dissipation.

7.2 Software Testing

Software testing was less quantitative than the hardware testing. We tested the software in four main areas: data receiving, database, web server, and web application. The software portion of the project didn't have hard specifications like the hardware did, with the exception of the ADC linearity and the accuracy of the RMS. The user portions of the software will either work or not work. Since the tests weren't necessarily quantitative, I will describe the general procedure that we used to test each component.

7.2.1 Data Receiving

The data receiving portion of the software is responsible for reading the incoming data packets that contain the power data. It was very easy to test; we sent perfectly formed UDP packets from a Python test script. These packets contained mock power data. After the data receiver read the packets and parsed out the data, we verified that the parsed data was the same as the sent data. All tests passed.

7.2.2 Database

Testing the database was a little trickier since it's not necessarily testable. Once the schema was created, we used some test SQL statements to insert data and make sure the formatting was correct. All the fields in the database were correct and the data able to be inserted and read correctly.

7.2.3 Web Server

The web server is a RESTful web service that serves data from the database. When the web app makes a HTTP query, the web server reads the requested data from the database and sends it back in a JSON format. We tested each part of this system individually.

First, we made sure that the web server was making the correct HTTP requests by checking the format in Chrome Developer Console. The requests were all well-formed.

Next, we tested each of our SQL statements by using an SQL editor and using our original schema. All SQL statements worked correctly and gave us the desired data.

Finally, we tested the returned JSON structure by using an online JSON validator; it made sure that our JSON was well-formed and able to be parsed by all third-party JSON libraries. All tests passed with flying colors.

7.2.4 CC3200 Code

We tested two aspects of the CC3200: ADC linearity and RMS accuracy.

To test the ADC linearity, we used a lab-grade power supply to gradually step-up the input voltage to the ADC by a value of 25 mV. We then recorded the digital value read by the ADC and compared it to the actual input voltage, which was measured by a lab multimeter. By using many data points, we were able to verify that the ADC is in fact extremely linear. We also stepped-down our voltage and recorded the same measurements to test for any hysteresis in our ADC. We found that there was no measurable difference and therefore no

hysteresis present:

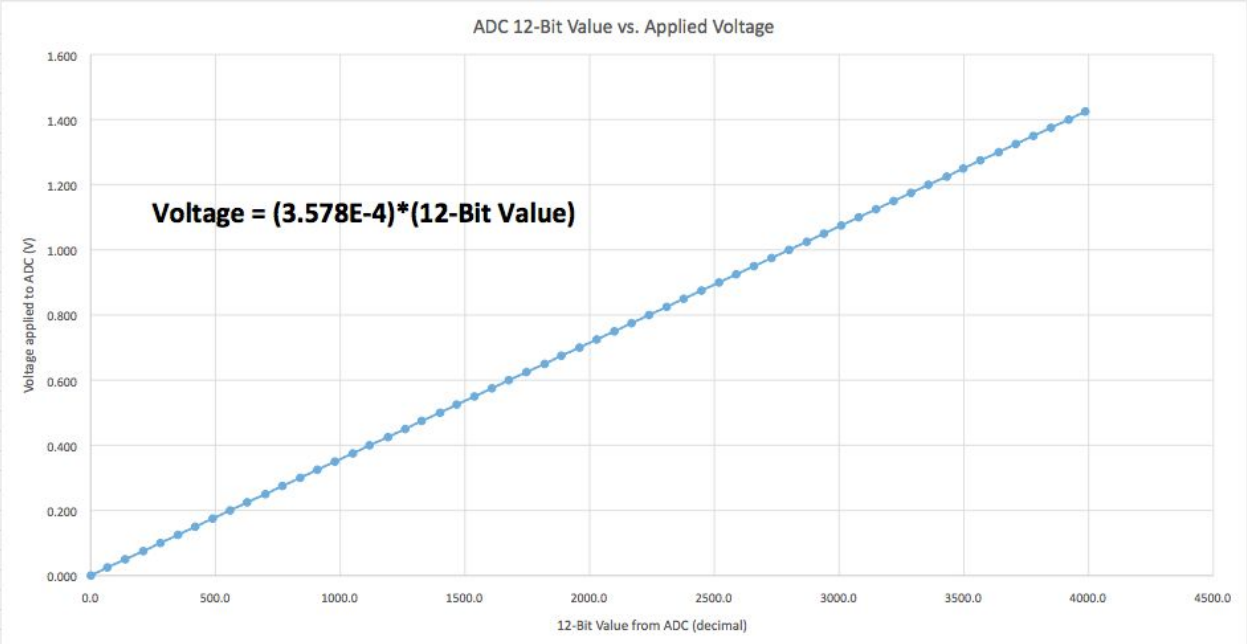


Figure 38: ADC linearity

To test the RMS algorithm, we used a similar approach. We used an AC signal generator to create a 60 Hz sine wave of a known amplitude. Since we know the RMS amplitude of the input signal, we can read the CC3200's calculated RMS value and compare it to the actual value. We knew the actual RMS value from a lab multimeter. Here are the test results:

Actual RMS Value (V)	CC3200 RMS Value (V)	Error (%)
0.036	0.0363	0.83
0.143	0.1429	0.07
0.251	0.2502	0.32
0.322	0.3218	0.06
0.358	0.3573	0.20

Table 2: RMS percentage error

We were very pleased with the accuracy of our RMS calculation. It certainly helped keep us within our overall accuracy requirements.

7.3 Case Testing

The first revision of our case broke from trying to cut down the fastening poles using a bandsaw. We also had some dimensional mistakes on the first revision that were revised on the second revision. The material is strong enough however to withstand devices being plugged in and out of the case.

8 Cost Summary

Here is the Bill of Materials for our project. The total estimated cost of creating our device was $105.72

Quantity	Description	RefDes	Package	Price	Total Cost
1	RESISTOR, 2.2kΩ	R16	IPC-7351\Chip-R0603	$ 0.10	$ 0.10
1	RESISTOR, 29.4kΩ	R15	IPC-7351\Chip-R0603	$ 0.12	$ 0.12
1	RESISTOR, 6.8kΩ	R7	IPC-7351\Chip-R0603	$ 0.10	$ 0.10
1	RESISTOR, 41.2kΩ	R12	IPC-7351\Chip-R0603	$ 0.12	$ 0.12
1	RESISTOR, 5.1kΩ	R21	IPC-7351\Chip-R0603	$ 0.10	$ 0.10
1	Varistor, S20K275	U3	Ultiboard\VAR_S20K230	$ 0.95	$ 0.95
1	RESISTOR, 0Ω	R22	IPC-7351\Chip-R0603	$ 0.10	$ 0.10
1	OPAMP, LTC6910-1CTS8	U8	IPC-7351\TSOT23-8	$ 2.79	$ 2.79
1	OPAMP, LM358DG	U15	ON Semiconductor\SOIC-8-8(CASE 751-07AJ)	$ 0.47	$ 0.47
1	CAPACITOR, 0.22µF	C27	IPC-7351\Chip-C0603	$ 0.10	$ 0.10
1	RESISTOR, 200kΩ	R33	IPC-7351\Chip-R0603	$ 0.10	$ 0.10
1	RESISTOR, 1kΩ	R34	IPC-7351\Chip-R0603	$ 0.10	$ 0.10
1	INDUCTOR, 10mH	L1	IPC-2221A/2222\CAPPR350-800X1150	$ 0.29	$ 0.29
1	INDUCTOR, 10µH	L2	IPC-7351\Chip-L1210	$ 0.26	$ 0.26
1	MOSFET DRIVER, UCC27325 SO8	U1	IPC-7351\SOIC-D-8	$ 1.59	$ 1.59
1	RESISTOR, 2.7kΩ	R25	IPC-7351\Chip-R0603	$ 0.10	$ 0.10
1	CAPACITOR, 10µF	C20	IPC-7351\Chip-C1210	$ 0.86	$ 0.86
1	RESISTOR, 30kΩ	R26	IPC-7351\Chip-R0603	$ 0.10	$ 0.10
1	RESISTOR, 100kΩ	R27	IPC-7351\Chip-R0603	$ 0.10	$ 0.10
1	RESISTOR, 120Ω	R28	IPC-7351\Chip-R0603	$ 0.10	$ 0.10
1	SWITCHING DIODE, BAS21H	D6	NXP Semiconductors\SOD-123-2(SOD123F)	$ 0.19	$ 0.19
1	DIODE, 1N4004G	D2	ON Semiconductor\SMA-2(CASE 403D-02F)	$ 0.47	$ 0.47
1	RESISTOR, 1.5kΩ	R29	IPC-7351\Chip-R1206	$ 0.10	$ 0.10
1	CAPACITOR, 100pF	C21	IPC-7351\Chip-C1206	$ 0.31	$ 0.31
1	PROTECTION DIODE, 1SMB170AT3	D8	ON Semiconductor\SMB-2(CASE 403A-03G)	$ 0.46	$ 0.46
1	SCHOTTKY DIODE, 1N5822G	D7	ON Semiconductor\SMA-2(CASE 403D-02F)	$ 0.68	$ 0.68
1	CAP ELECTROLIT, 820µF	C22	IPC-2221A/2222\CAPPR350-800X1150	$ 0.67	$ 0.67
1	CAPACITOR, 10µF	C23	IPC-7351\Chip-C1206	$ 0.25	$ 0.25
1	CAPACITOR, 100µF	C25	IPC-7351\Chip-C1206	$ 0.74	$ 0.74
1	SCHOTTKY DIODE, MBR3100RLG	D10	ON Semiconductor\SMA-2(CASE 403D-02F)	$ 0.60	$ 0.60
1	RESISTOR, 10Ω	R32	IPC-2221A/2222\RES1600-1000X400	$ 0.72	$ 0.72
1	Transformer, XMFR314752	U13	Ultiboard\XMFR314752	$ 7.63	$ 7.63
1	GENERIC, HDR2X3	J2	Generic\HDR2X3	$ 0.25	$ 0.25
1	MB6S	U14	Ultiboard\MB6S	$ 0.38	$ 0.38
1	GENERIC, HDR1X3	J3	Generic\HDR1X3	$ 0.25	$ 0.25
1	GENERIC, HDR1X10	J1	Generic\HDR1X10	$ 0.50	$ 0.50
1	CC3200MOD	U11	MCU	$ 10.63	$ 10.63
2	CAPACITOR, 220pF	C3, C7	IPC-7351\Chip-C0603	$ 0.11	$ 0.22
2	Instrumentation, INA188	U2, U4	IPC-7351\SOIC-8	$ 2.08	$ 4.16
2	RESISTOR, 2mΩ	R3, R8	Ultiboard\RES3921	$ 2.21	$ 4.42
2	RESISTOR, 100Ω	R19, R20	IPC-7351\Chip-R0603	$ 0.10	$ 0.20
2	CAPACITOR, 1µF	C9, C10	IPC-7351\Chip-C0603	$ 0.19	$ 0.38
2	BLM21BD121SN1, BLM21BD121SN1	D7, U11	IPC-7351\Chip-L0805	$ 0.15	$ 0.30
2	CAP ELECTROLIT, 4.7µF	C18, C19	IPC-2221A/2222\CAPPR350-800X1150	$ 0.48	$ 0.96
2	RESISTOR, 6.8kΩ	R4, R24	IPC-7351\Chip-R0805	$ 0.10	$ 0.20
2	CAPACITOR, 0.1µF	C24, C26	IPC-7351\Chip-C0603	$ 0.10	$ 0.20
2	RESISTOR, 2kΩ	R30, R31	IPC-7351\Chip-R0603	$ 0.10	$ 0.20
3	RESISTOR, 470kΩ	R9, R10, R11	IPC-7351\Chip-R0603	$ 0.10	$ 0.30
3	CAPACITOR, 100µF	C4, C28, C29	IPC-7351\Chip-C1210	$ 0.74	$ 2.22
4	CAPACITOR, 0.1µF	C1, C2, C5, C6	IPC-7351\Chip-C0603	$ 0.10	$ 0.40
4	RESISTOR, 1kΩ	R1, R2, R17, R18	IPC-7351\Chip-R0603	$ 0.10	$ 0.40
4	RESISTOR, 10kΩ	R5, R6, R13, R14	IPC-7351\Chip-R0603	$ 0.10	$ 0.40
8	CAPACITOR, 0.1µF	C8, C11, C12, C13, C14, C15, C16, C17	IPC-7351\Chip-C0603	$ 0.10	$ 0.80

Figure 39: Bill of Materials

PCB	$12.00
CC3200MOD	$10.63
Passive Components	$24.08
Active Components	$9.01
Case and Miscs	$50.00
Total	$105.72
Total Man Hours	650

Figure 40: Cost Summary

9. Conclusions

In conclusion, there were two main goals of our project and several steps to achieve these goals. The first goal was to build a working power sensor that measures and transfers power consumption data of a certain device. The second goal was to deal with data from multiple devices simultaneously and to display them properly on the user's interface. We were not able to fully complete this goal due to time and material constraints. Still we are happy with the result we got and feel good knowing that will the adequate time we may have implemented this feature without to much trouble. To achieve these goals, we used a Hall effect transimpedance amplifier to get a proportional output voltage and current which can be converted to digital data, and calculated to power through our microprocessor. Then we built a network portion of the project which includes the data processing and receiving code, HTTP server and database system. At last, we added functions on the web application to allow users to monitor the power of their device.

10. Timeline

See Appendix IV.

10.1 First Semester

The first semester will be spent mostly in the conception, research, and prototype phases as seen above. Once the conception was complete we split into two groups: hardware and software, with three members on each team. At this point, each team researched their respective components to achieve our goal. For example, the hardware team looked at ways of power measurements and conceived a circuit that should theoretically output a voltage that is inputted into the microprocessor. From there the software team will write algorithms to compute average power and transmit the data. However, the software team is reaching libraries and tinkering with development kits to better understand how to process and transmit data all the way back to the user. Once the research is complete, we order sample parts to test their behavior. If as expected, or close enough, we will develop a prototype by the end of the semester. That being said, the prototype will still be in testing phases until the following semester.

10.2 Second Semester

The second semester will involve hardware refinement and software prototyping. We should have a working sensor prototype that can send messages to a central hub. We will be testing the reception of the UDP server and the storage of the data into the database. Next, we will refine the prototype of the web application to include all of the user functionality we want. We will do UI testing and make sure that the aesthetics of the site are up to par. In addition, we will finalize the PCB design of our hardware section, which includes the power sensors and the microcontroller.

We will finish the semester with documentation and preparing for our final presentation. Documentation will include a complete explanation of our final approach and implementation. Preparing for our final presentation will include comparing our solution to commercial systems and also rehearsing for the oral presentation.

11. References

1. "The Energy Detective Electricity Monitor." *The Energy Detective Electricity Monitor*. TED, n.d. Web. 12 Oct. 2016. <http://www.theenergydetective.com/>.
2. "CC3200." *CC3200 | SimpleLink CC3x | Wireless MCUs | Description & Parametrics*. Texas Instruments, n.d. Web. 20 Nov. 2016. <http://www.ti.com/product/CC3200>.
3. Develco Products. (n.d.). *Smart Plug.* Retrieved from http://www.develcoproducts.com/media/1459/smart-plug-datasheet.pdf
4. Maxim Integrated. (2002). *Linear Charger for Single-Cell Li+ Battery - MAX1898.* Retrieved from https://datasheets.maximintegrated.com/en/ds/MAX1898.pdf
5. ROHM Semiconductor. (2015). *PSR400ITQFJ2L00 Datasheet.* Retrieved from http://www.mouser.com/ds/2/348/psr-951869.pdf
6. Stencel, L. (2014, Mar 4). Effective Surge And Lightning Strike Protection For AC And DC Power Line Applications. Retrieved from http://powerelectronics.com/circuit-protection-ics/effective-surge-and-lightning-strike-protection-ac-and-dc-power-line-applicat?page=2
7. Texas Instruments. (2000, September 27). *Programmable Gain Difference Amplifier - INA145.* Retrieved from http://www.ti.com/lit/ds/sbos120/sbos120.pdf
8. Texas Instruments. (2000, September 27). *Programmable Gain Difference Amplifier - INA146.* Retrieved from http://www.ti.com/lit/ds/symlink/ina146.pdf
9. Texas Instruments. (2014, December 5). *CC3200MOD SimpleLink Wi-Fi and Internet-of-Things Module Solution, a Single-Chip Wireless MCU.* Retrieved from http://www.ti.com/lit/ds/symlink/cc3200mod.pdf
10. Yarborough, B. (2012, Jan 6). Components and Methods for Current Measurement. Retrieved from http://powerelectronics.com/power-electronics-systems/components-and-methods-current-measurement

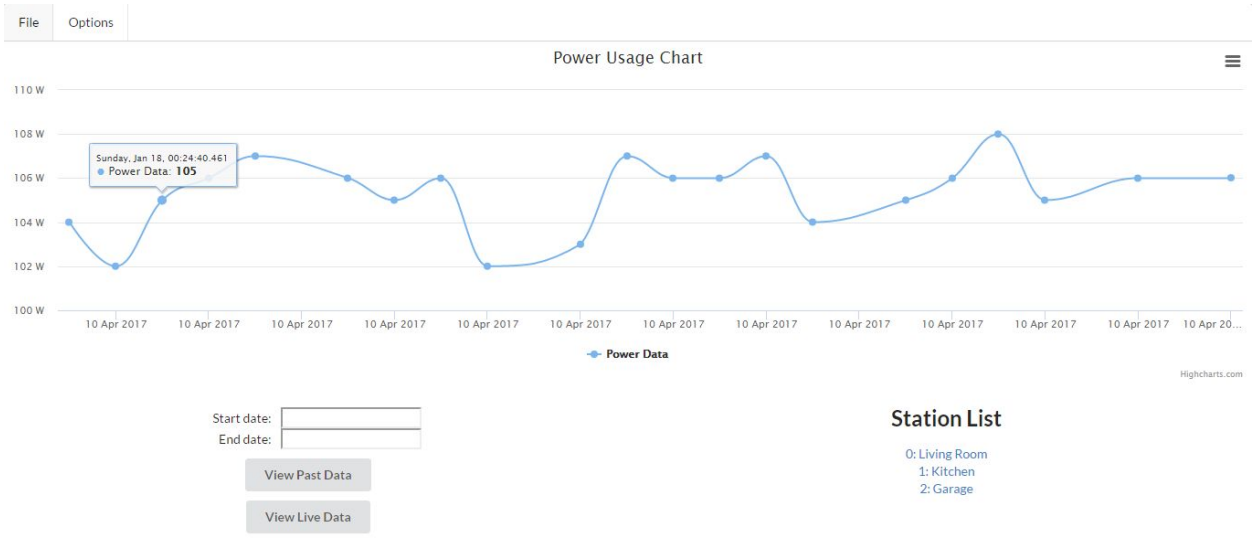
12. Appendices

12.1 Appendix I:

Project Neptune is very easy to use in your home. This textual operation manual should guide you through the basic setup.

1. Unpack your Neptune power monitors and central hub.
2. Power on your central hub and plug it into your home router.
3. Verify that the central hub powers on correctly.
4. Set up a power monitor station
    a. Plug the station into your wall outlet
    b. Plug the device to be monitored into the Neptune station
    c. You are ready to go!

Now that the units are all plugged in and powered on, you can view the data by using your laptop to go the IP address of the central hub. The default IP address is 192.168.100.100, on port 80. If you enter the IP address in your web browser, you should see the following screen (without the data):



If you've plugged in your station correctly, you should see live data appearing on your screen! This means that everything is working ok.

Here are a few tips for using the web application:

1. To see data for a specific station, click on it under the "Station List" menu.
2. To see the live data, click "View Live Data". This will automatically switch to live mode.
3. Hovering over a dot on the graph will show you the exact power value.
4. Clicking the three-lined icon on the upper-right hand corner of the graph will allow you to download the graph as an image.
5. You can view past data by using the "Start Date" and "End Date" pickers and clicking "View Past Data".

12.2 Appendix II

The critical part of our design is current sensing. Throughout out the project, we have tried two different methods. One is using the Hall-effect sensor. This worked quite well for the high current level but not for the current below 1 Amperes. This results in low resolution and larger error. In addition, Hall-effect sensor is sensitive to magnetic field. CC3200MOD is integrated in the final design. So the chance the wifi signal affects the current sensing is very high. The other method is using 1 mOhm resistor. This is relatively low resistor value. Our ultimate goal is to reduce power consumption of the overall device. The challenge that we faced when using 1 mOhm resistor is noise. The SNR is relatively low and unsuitable for the our design requirement. Our solution was to increase the resistor value to 4 mOhms. This effectively increased the power consumption. However, we get higher SNR and feel more confident about the accuracy.

a. Hall-effect sensor (ACS712 sensor)

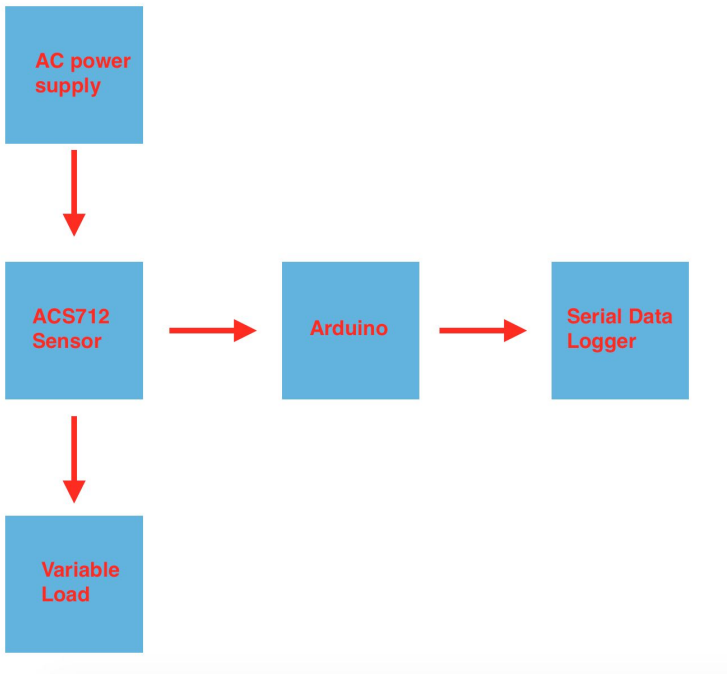


Figure 4.3-1: Testing ACS712 sensor flow chart

Testing the ACS712 sensor is done by simply connecting the sensor in series with the power supply and load. We modified the AC plug socket so that we can switch between different loads for different measurements. The output pin of the ACS712 will be connected directly to the analog pin of the Arduino microcontroller. Arduino is used because our CC3200 is not fully functional at this stage of the design. In addition, Arduino interface provide serial monitor, which makes it easier to obtain raw data. During the testing phase, we connect different appliances (ranging from high to low power consumption device) to the modified AC plug socket. Then we record the data through serial port at 250 kBaud rate.

Comments:

- From this set of the data, we learned that most resistive loads such as: heaters, incandescent bulbs, and water boilers draw sinusoidal current. On the other hand, devices like phone charger draws current at different duty cycle. This result fundamentally changed how we sample data. With sinusoidal current, low sampling rate can be applied to calculate the current because the current varies in a predictable way. However, if the current changes randomly, a higher sampling rate is needed in order to capture all the data points.
- Secondly, we found that ACS712 is capable measuring high current level without any issue. However, once the current level drop below 1A, ACS712 seems to perform poorly due to the noise level. This is something that we can't improve much because it is a common characteristic of the Hall-effect sensor.
- The failure of the ACS712 is that it is sensitive to magnetic field. We thought about shielding the sensor but considering its performance, ACS712 might not be the best current method.
- The main target of the design is to account for low-power consumption appliances. However, with the test results from the ACS712, we have to defer to the shunt resistor method to achieve better a better current measurement range.

b. Current sensing using shunt resistor (1 mOhms resistor)

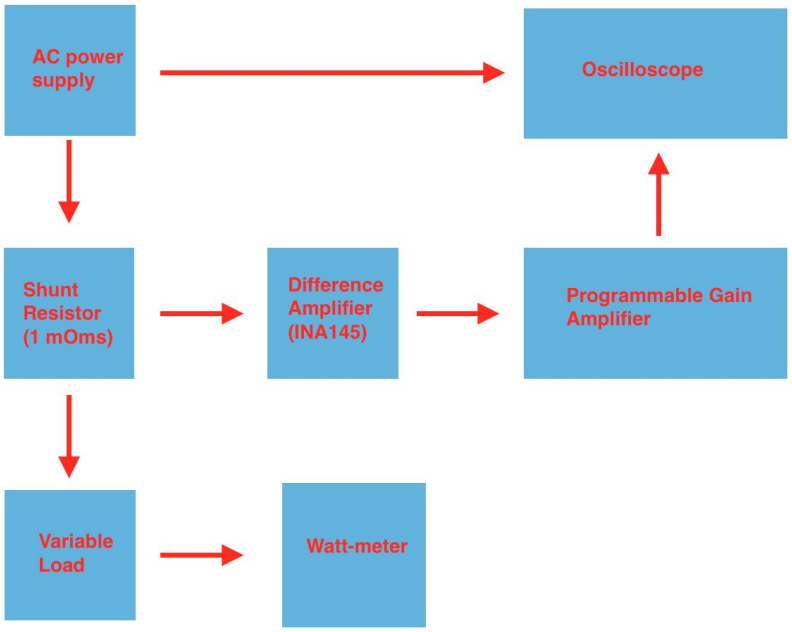


Figure 4.3-2: Current sensing using shunt resistor flowchart

Testing current sensing using the shunt resistor is a bit more complicated than the previous hall-effect sensor. First, we connect the series shunt resistor in series with AC power supply and load. Then, we connect both input terminals of the difference amplifier to the shunt resistor. Because the voltage drops across the resistor is relatively small (~microvolts), it is very insufficient to look at the voltage using the oscilloscope. The next step would be

amplifying the voltage difference using the programmable gain amplifier. The output voltage is now large enough to be observed using the oscilloscope. In addition, we will observe the AC voltage. In most cases, the voltage provided by power company doesn't vary much. However, during the testing stage, we want to make sure everything is accurate as much as possible. Once we obtain the voltages from voltage and current sensing, we can mathematically analyze the data to revert back to the actual values. The oscilloscope that we use in the lab has this math capability so we can calculate the instantaneous power directly. The final step of testing would be comparing the measured power with the value from from Wattmeter. The testing cycle will repeat for each load.

Comments:

- Using shunt resistors, we were able to measure current as low as 100 mA. This is something that we couldn't achieve using the ACS712. Additionally, the noise level at the output is significantly reduced. We are confident that our actual circuit will perform equivalently to the simulation because Multisim has very accurate amplifier models.
- As alluded to before, we decided on using shunt resistors to measure current. With the programmable gain, we can divide the current range into multiple ranges. By doing this, better resolution can be achieved

12.3 Appendix III

We learned a lot during the course of this project.

Hardware

1. We learned how to utilize different tools like Multisim, Ultiboard, and Matlab to aid our design. PCB design is a very complicated process. We started out with a blank sheet of schematic and built off from there. Simulations are helpful but the actual circuit might behave different, which makes debugging process much more challenging.
2. Debugging or measuring circuit involving high AC voltage is always hazardous. We were very cautious and took everything slowly to ensure safety.
3. Time Management is key in our design process. We always planned ahead and kept track of the arrival date for each components. This ensures we're always on track and find alternative solutions if delay occurs.

Software

Things learned during software development:

1. If we could start over, we would have used a microcontroller with a better development environment. TI provides good tools but they are not as good as other industry tools like Arduino and Netduino
2. In terms of web development, finding a library is ALWAYS easier than writing one yourself. There is such an abundance of libraries that making your own UI elements especially is usually not necessary.
3. Databases can slow down your round-trip data retrieval time significantly, and should be optimized.

12.4 Appendix IV (code)

12.4.1 CC3200 Code (main.c file, only file needed)

```
//*****************************************************************************
//
//  Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/
//
//
//  Redistribution and use in source and binary forms, with or without
//  modification, are permitted provided that the following conditions
//  are met:
//
//  Redistributions of source code must retain the above copyright
//  notice, this list of conditions and the following disclaimer.
//
//  Redistributions in binary form must reproduce the above copyright
//  notice, this list of conditions and the following disclaimer in the
//  documentation and/or other materials provided with the
//  distribution.
//
//  Neither the name of Texas Instruments Incorporated nor the names of
//  its contributors may be used to endorse or promote products derived
//  from this software without specific prior written permission.
//
//  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
//  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
//  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
//  A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
//  OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
//  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
//  LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
//  DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
//  THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
//  (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
//  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
//
//*****************************************************************************


//*****************************************************************************
//
```

```
// Application Name     -   Getting started with WLAN STATION
// Application Overview -   This is a sample application demonstrating how to
//                          start CC3200 in WLAN-Station mode and connect to a
//                          Wi-Fi access-point. The application connects to an
//                          access-point and ping the gateway. It also checks
//                          for an internet connectivity by pinging "www.ti.com"
// Application Details  -
// http://processors.wiki.ti.com/index.php/CC32xx_Getting_Started_with_WLAN_Station
// or
// doc\examples\CC32xx_Getting_Started_with_WLAN_Station.pdf
//
//*****************************************************************************


//*****************************************************************************
//
//! \addtogroup getting_started_sta
//! @{
//
//*****************************************************************************

// Standard includes
#include <stdlib.h>
#include <string.h>

// Simplelink includes
#include "simplelink.h"

//Driverlib includes
#include "hw_types.h"
#include "hw_ints.h"
#include "rom.h"
#include "rom_map.h"
#include "interrupt.h"
#include "prcm.h"
#include "utils.h"

#include "network_if.h"


//Free_rtos/ti-rtos includes
```

```
#include "osi.h"

//Common interface includes
#include "gpio_if.h"
#ifndef NOTERM
#include "uart_if.h"
#endif
#include "common.h"
#include "pinmux.h"

#include "utils.h"
#include "hw_memmap.h"
#include "hw_common_reg.h"
#include "hw_types.h"
#include "hw_adc.h"
#include "hw_ints.h"
#include "hw_gprcm.h"
#include "rom.h"
#include "rom_map.h"
#include "interrupt.h"
#include "prcm.h"
#include "uart.h"
#include "pinmux.h"
#include "pin.h"
#include "adc.h"




#define APPLICATION_NAME        "WLAN STATION"
#define APPLICATION_VERSION     "1.1.1"

#define HOST_NAME               "www.ti.com"

#define TIME2013                3565987200u      /* 113 years + 28 days(leap) */
#define YEAR2013                2013
#define SEC_IN_MIN              60
#define SEC_IN_HOUR             3600
#define SEC_IN_DAY              86400
```

```
//
// Values for below macros shall be modified for setting the 'Ping' properties
//
#define PING_INTERVAL       1000    /* In msecs */
#define PING_TIMEOUT        3000    /* In msecs */
#define PING_PKT_SIZE       20      /* In bytes */
#define NO_OF_ATTEMPTS      3

#define OSI_STACK_SIZE      2048

const char g_acDigits[] = "0123456789";

struct
{
    unsigned long ulDestinationIP;
    int iSockID;
    unsigned long ulElapsedSec;
    short isGeneralVar;
    unsigned long ulGeneralVar;
    unsigned long ulGeneralVar1;
    char acTimeStore[30];
    char *pcCCPtr;
    unsigned short uisCCLen;
}g_sAppData;

SlSockAddr_t sAddr;
SlSockAddrIn_t sLocalAddr;

// Application specific status/error codes
typedef enum{
    // Choosing -0x7D0 to avoid overlap w/ host-driver's error codes
    LAN_CONNECTION_FAILED = -0x7D0,
    INTERNET_CONNECTION_FAILED = LAN_CONNECTION_FAILED - 1,
    DEVICE_NOT_IN_STATION_MODE = INTERNET_CONNECTION_FAILED - 1,

    STATUS_CODE_MAX = -0xBB8
}e_AppStatusCodes;


//*****************************************************************************
```

```
//                  GLOBAL VARIABLES -- Start
//*****************************************************************************
unsigned long  g_ulStatus = 0;//SimpleLink Status
unsigned long  g_ulPingPacketsRecv = 0; //Number of Ping Packets received
unsigned long  g_ulGatewayIP = 0; //Network Gateway IP address
unsigned char  g_ucConnectionSSID[SSID_LEN_MAX+1]; //Connection SSID
unsigned char  g_ucConnectionBSSID[BSSID_LEN_MAX]; //Connection BSSID

#if defined(gcc)
extern void (* const g_pfnVectors[])(void);
#endif
#if defined(ewarm)
extern uVectorEntry __vector_table;
#endif
//*****************************************************************************
//                  GLOBAL VARIABLES -- End
//*****************************************************************************




//*****************************************************************************
//                      LOCAL FUNCTION PROTOTYPES
//*****************************************************************************
static long WlanConnect();
void WlanStationMode( void *pvParameters );
static long CheckLanConnection();
static long CheckInternetConnection();
static void InitializeAppVariables();
static long ConfigureSimpleLinkToDefaultState();


#ifdef USE_FREERTOS
//*****************************************************************************
// FreeRTOS User Hook Functions enabled in FreeRTOSConfig.h
//*****************************************************************************

//*****************************************************************************
//
//! \brief Application defined hook (or callback) function - assert
//!
//! \param[in]  pcFile - Pointer to the File Name
```

```
//! \param[in]  ulLine - Line Number
//!
//! \return none
//!
//*****************************************************************************
void
vAssertCalled( const char *pcFile, unsigned long ulLine )
{
    //Handle Assert here
    while(1)
    {
    }
}

//*****************************************************************************
//
//! \brief Application defined idle task hook
//!
//! \param  none
//!
//! \return none
//!
//*****************************************************************************
void
vApplicationIdleHook( void)
{
    //Handle Idle Hook for Profiling, Power Management etc
}

//*****************************************************************************
//
//! \brief Application defined malloc failed hook
//!
//! \param  none
//!
//! \return none
//!
//*****************************************************************************
void vApplicationMallocFailedHook()
{
    //Handle Memory Allocation Errors
```

```
    while(1)
    {
    }
}

//*****************************************************************************
//
//! \brief Application defined stack overflow hook
//!
//! \param  none
//!
//! \return none
//!
//*****************************************************************************
void vApplicationStackOverflowHook( OsiTaskHandle *pxTask,
                                    signed char *pcTaskName)
{
    //Handle FreeRTOS Stack Overflow
    while(1)
    {
    }
}
#endif //USE_FREERTOS



//*****************************************************************************
// SimpleLink Asynchronous Event Handlers -- Start
//*****************************************************************************


//*****************************************************************************
//
//! \brief The Function Handles WLAN Events
//!
//! \param[in]  pWlanEvent - Pointer to WLAN Event Info
//!
//! \return None
//!
//*****************************************************************************
void SimpleLinkWlanEventHandler(SlWlanEvent_t *pWlanEvent)
```

```
{
    switch(pWlanEvent->Event)
    {
        case SL_WLAN_CONNECT_EVENT:
        {
            SET_STATUS_BIT(g_ulStatus, STATUS_BIT_CONNECTION);

            //
            // Information about the connected AP (like name, MAC etc) will be
            // available in 'slWlanConnectAsyncResponse_t'-Applications
            // can use it if required
            //
            //  slWlanConnectAsyncResponse_t *pEventData = NULL;
            // pEventData = &pWlanEvent->EventData.STAandP2PModeWlanConnected;
            //

            // Copy new connection SSID and BSSID to global parameters
            memcpy(g_ucConnectionSSID,pWlanEvent->EventData.
                   STAandP2PModeWlanConnected.ssid_name,
                   pWlanEvent->EventData.STAandP2PModeWlanConnected.ssid_len);
            memcpy(g_ucConnectionBSSID,
                   pWlanEvent->EventData.STAandP2PModeWlanConnected.bssid,
                   SL_BSSID_LENGTH);

            UART_PRINT("[WLAN EVENT] STA Connected to the AP: %s ,"
                       "BSSID: %x:%x:%x:%x:%x:%x\n\r",
                     g_ucConnectionSSID,g_ucConnectionBSSID[0],
                     g_ucConnectionBSSID[1],g_ucConnectionBSSID[2],
                     g_ucConnectionBSSID[3],g_ucConnectionBSSID[4],
                     g_ucConnectionBSSID[5]);
        }
        break;

        case SL_WLAN_DISCONNECT_EVENT:
        {
            slWlanConnectAsyncResponse_t*  pEventData = NULL;

            CLR_STATUS_BIT(g_ulStatus, STATUS_BIT_CONNECTION);
            CLR_STATUS_BIT(g_ulStatus, STATUS_BIT_IP_AQUIRED);

            pEventData = &pWlanEvent->EventData.STAandP2PModeDisconnected;
```

```
            // If the user has initiated 'Disconnect' request,
            //'reason_code' is SL_WLAN_DISCONNECT_USER_INITIATED_DISCONNECTION
            if(SL_WLAN_DISCONNECT_USER_INITIATED_DISCONNECTION == pEventData->reason_code)
            {
                UART_PRINT("[WLAN EVENT]Device disconnected from the AP: %s,"
                "BSSID: %x:%x:%x:%x:%x:%x on application's request \n\r",
                           g_ucConnectionSSID,g_ucConnectionBSSID[0],
                           g_ucConnectionBSSID[1],g_ucConnectionBSSID[2],
                           g_ucConnectionBSSID[3],g_ucConnectionBSSID[4],
                           g_ucConnectionBSSID[5]);
            }
            else
            {
                UART_PRINT("[WLAN ERROR]Device disconnected from the AP AP: %s,"
                "BSSID: %x:%x:%x:%x:%x:%x on an ERROR..!! \n\r",
                           g_ucConnectionSSID,g_ucConnectionBSSID[0],
                           g_ucConnectionBSSID[1],g_ucConnectionBSSID[2],
                           g_ucConnectionBSSID[3],g_ucConnectionBSSID[4],
                           g_ucConnectionBSSID[5]);
            }
            memset(g_ucConnectionSSID,0,sizeof(g_ucConnectionSSID));
            memset(g_ucConnectionBSSID,0,sizeof(g_ucConnectionBSSID));
        }
        break;

        default:
        {
            UART_PRINT("[WLAN EVENT] Unexpected event [0x%x]\n\r",
                       pWlanEvent->Event);
        }
        break;
    }
}

//*****************************************************************************
//
//! \brief This function handles network events such as IP acquisition, IP
//!           leased, IP released etc.
//!
//! \param[in]  pNetAppEvent - Pointer to NetApp Event Info
```

```
//!
//! \return None
//!
//*****************************************************************************
void SimpleLinkNetAppEventHandler(SlNetAppEvent_t *pNetAppEvent)
{
    switch(pNetAppEvent->Event)
    {
        case SL_NETAPP_IPV4_IPACQUIRED_EVENT:
        {
            SlIpV4AcquiredAsync_t *pEventData = NULL;

            SET_STATUS_BIT(g_ulStatus, STATUS_BIT_IP_AQUIRED);

            //Ip Acquired Event Data
            pEventData = &pNetAppEvent->EventData.ipAcquiredV4;

            //Gateway IP address
            g_ulGatewayIP = pEventData->gateway;

            UART_PRINT("[NETAPP EVENT] IP Acquired: IP=%d.%d.%d.%d , "
            "Gateway=%d.%d.%d.%d\n\r",
            SL_IPV4_BYTE(pNetAppEvent->EventData.ipAcquiredV4.ip,3),
            SL_IPV4_BYTE(pNetAppEvent->EventData.ipAcquiredV4.ip,2),
            SL_IPV4_BYTE(pNetAppEvent->EventData.ipAcquiredV4.ip,1),
            SL_IPV4_BYTE(pNetAppEvent->EventData.ipAcquiredV4.ip,0),
            SL_IPV4_BYTE(pNetAppEvent->EventData.ipAcquiredV4.gateway,3),
            SL_IPV4_BYTE(pNetAppEvent->EventData.ipAcquiredV4.gateway,2),
            SL_IPV4_BYTE(pNetAppEvent->EventData.ipAcquiredV4.gateway,1),
            SL_IPV4_BYTE(pNetAppEvent->EventData.ipAcquiredV4.gateway,0));
        }
        break;

        default:
        {
            UART_PRINT("[NETAPP EVENT] Unexpected event [0x%x] \n\r",
                       pNetAppEvent->Event);
        }
        break;
    }
}
```

```
//*****************************************************************************
//
//! \brief This function handles HTTP server events
//!
//! \param[in]  pServerEvent - Contains the relevant event information
//! \param[in]    pServerResponse - Should be filled by the user with the
//!                                     relevant response information
//!
//! \return None
//!
//*****************************************************************************
void SimpleLinkHttpServerCallback(SlHttpServerEvent_t *pHttpEvent,
                                  SlHttpServerResponse_t *pHttpResponse)
{
    // Unused in this application
}

//*****************************************************************************
//
//! \brief This function handles General Events
//!
//! \param[in]     pDevEvent - Pointer to General Event Info
//!
//! \return None
//!
//*****************************************************************************
void SimpleLinkGeneralEventHandler(SlDeviceEvent_t *pDevEvent)
{
    //
    // Most of the general errors are not FATAL are are to be handled
    // appropriately by the application
    //
    UART_PRINT("[GENERAL EVENT] - ID=[%d] Sender=[%d]\n\n",
               pDevEvent->EventData.deviceEvent.status,
               pDevEvent->EventData.deviceEvent.sender);
}


//*****************************************************************************
```

```
//
//! This function handles socket events indication
//!
//! \param[in]      pSock - Pointer to Socket Event Info
//!
//! \return None
//!
//*****************************************************************************
void SimpleLinkSockEventHandler(SlSockEvent_t *pSock)
{
    //
    // This application doesn't work w/ socket - Events are not expected
    //
    switch( pSock->Event )
    {
        case SL_SOCKET_TX_FAILED_EVENT:
            switch( pSock->socketAsyncEvent.SockTxFailData.status)
            {
                case SL_ECLOSE:
                    UART_PRINT("[SOCK ERROR] - close socket (%d) operation "
                               "failed to transmit all queued packets\n\n",
                                   pSock->socketAsyncEvent.SockTxFailData.sd);
                    break;
                default:
                    UART_PRINT("[SOCK ERROR] - TX FAILED  :  socket %d , reason "
                               "(%d) \n\n",
                               pSock->socketAsyncEvent.SockTxFailData.sd,
pSock->socketAsyncEvent.SockTxFailData.status);
                  break;
            }
            break;

        default:
              UART_PRINT("[SOCK EVENT] - Unexpected Event [%x0x]\n\n",pSock->Event);
          break;
    }

}


//*****************************************************************************
```

```
//
//! \brief This function handles ping report events
//!
//! \param[in]     pPingReport - Ping report statistics
//!
//! \return None
//!
//*****************************************************************************
static void SimpleLinkPingReport(SlPingReport_t *pPingReport)
{
    SET_STATUS_BIT(g_ulStatus, STATUS_BIT_PING_DONE);
    g_ulPingPacketsRecv = pPingReport->PacketsReceived;
}

//*****************************************************************************
// SimpleLink Asynchronous Event Handlers -- End
//*****************************************************************************



//*****************************************************************************
//
//! \brief This function initializes the application variables
//!
//! \param    None
//!
//! \return None
//!
//*****************************************************************************
static void InitializeAppVariables()
{
    g_ulStatus = 0;
    g_ulPingPacketsRecv = 0;
    g_ulGatewayIP = 0;
    memset(g_ucConnectionSSID,0,sizeof(g_ucConnectionSSID));
    memset(g_ucConnectionBSSID,0,sizeof(g_ucConnectionBSSID));
}


//*****************************************************************************
//! \brief This function puts the device in its default state. It:
```

```
//!           - Set the mode to STATION
//!           - Configures connection policy to Auto and AutoSmartConfig
//!           - Deletes all the stored profiles
//!           - Enables DHCP
//!           - Disables Scan policy
//!           - Sets Tx power to maximum
//!           - Sets power policy to normal
//!           - Unregister mDNS services
//!           - Remove all filters
//!
//! \param   none
//! \return  On success, zero is returned. On error, negative is returned
//*****************************************************************************

static long ConfigureSimpleLinkToDefaultState()
{
    SlVersionFull   ver = {0};
    _WlanRxFilterOperationCommandBuff_t  RxFilterIdMask = {0};

    unsigned char ucVal = 1;
    unsigned char ucConfigOpt = 0;
    unsigned char ucConfigLen = 0;
    unsigned char ucPower = 0;

    long lRetVal = -1;
    long lMode = -1;

    lMode = sl_Start(0, 0, 0);
    ASSERT_ON_ERROR(lMode);

    // If the device is not in station-mode, try configuring it in station-mode
    if (ROLE_STA != lMode)
    {
        if (ROLE_AP == lMode)
        {
            // If the device is in AP mode, we need to wait for this event
            // before doing anything
            while(!IS_IP_ACQUIRED(g_ulStatus))
            {
#ifndef SL_PLATFORM_MULTI_THREADED
              _SlNonOsMainLoopTask();
```

```
#endif
            }
        }

        // Switch to STA role and restart
        lRetVal = sl_WlanSetMode(ROLE_STA);
        ASSERT_ON_ERROR(lRetVal);

        lRetVal = sl_Stop(0xFF);
        ASSERT_ON_ERROR(lRetVal);

        lRetVal = sl_Start(0, 0, 0);
        ASSERT_ON_ERROR(lRetVal);

        // Check if the device is in station again
        if (ROLE_STA != lRetVal)
        {
            // We don't want to proceed if the device is not coming up in STA-mode
            ASSERT_ON_ERROR(DEVICE_NOT_IN_STATION_MODE);
        }
    }

    // Get the device's version-information
    ucConfigOpt = SL_DEVICE_GENERAL_VERSION;
    ucConfigLen = sizeof(ver);
    lRetVal = sl_DevGet(SL_DEVICE_GENERAL_CONFIGURATION, &ucConfigOpt,
                                &ucConfigLen, (unsigned char *)(&ver));
    ASSERT_ON_ERROR(lRetVal);

    UART_PRINT("Host Driver Version: %s\n\r",SL_DRIVER_VERSION);
    UART_PRINT("Build Version %d.%d.%d.%d.31.%d.%d.%d.%d.%d.%d.%d.%d\n\r",
    ver.NwpVersion[0],ver.NwpVersion[1],ver.NwpVersion[2],ver.NwpVersion[3],
    ver.ChipFwAndPhyVersion.FwVersion[0],ver.ChipFwAndPhyVersion.FwVersion[1],
    ver.ChipFwAndPhyVersion.FwVersion[2],ver.ChipFwAndPhyVersion.FwVersion[3],
    ver.ChipFwAndPhyVersion.PhyVersion[0],ver.ChipFwAndPhyVersion.PhyVersion[1],
    ver.ChipFwAndPhyVersion.PhyVersion[2],ver.ChipFwAndPhyVersion.PhyVersion[3]);

    // Set connection policy to Auto + SmartConfig
    //      (Device's default connection policy)
    lRetVal = sl_WlanPolicySet(SL_POLICY_CONNECTION,
                                SL_CONNECTION_POLICY(1, 0, 0, 0, 1), NULL, 0);
```

```
    ASSERT_ON_ERROR(lRetVal);

    // Remove all profiles
    lRetVal = sl_WlanProfileDel(0xFF);
    ASSERT_ON_ERROR(lRetVal);



    //
    // Device in station-mode. Disconnect previous connection if any
    // The function returns 0 if 'Disconnected done', negative number if already
    // disconnected Wait for 'disconnection' event if 0 is returned, Ignore
    // other return-codes
    //
    lRetVal = sl_WlanDisconnect();
    if(0 == lRetVal)
    {
        // Wait
        while(IS_CONNECTED(g_ulStatus))
        {
#ifndef SL_PLATFORM_MULTI_THREADED
              _SlNonOsMainLoopTask();
#endif
        }
    }

    // Enable DHCP client
    lRetVal = sl_NetCfgSet(SL_IPV4_STA_P2P_CL_DHCP_ENABLE,1,1,&ucVal);
    ASSERT_ON_ERROR(lRetVal);

    // Disable scan
    ucConfigOpt = SL_SCAN_POLICY(0);
    lRetVal = sl_WlanPolicySet(SL_POLICY_SCAN , ucConfigOpt, NULL, 0);
    ASSERT_ON_ERROR(lRetVal);

    // Set Tx power level for station mode
    // Number between 0-15, as dB offset from max power - 0 will set max power
    ucPower = 0;
    lRetVal = sl_WlanSet(SL_WLAN_CFG_GENERAL_PARAM_ID,
            WLAN_GENERAL_PARAM_OPT_STA_TX_POWER, 1, (unsigned char *)&ucPower);
    ASSERT_ON_ERROR(lRetVal);
```

```
    // Set PM policy to normal
    lRetVal = sl_WlanPolicySet(SL_POLICY_PM , SL_NORMAL_POLICY, NULL, 0);
    ASSERT_ON_ERROR(lRetVal);

    // Unregister mDNS services
    lRetVal = sl_NetAppMDNSUnRegisterService(0, 0);
    ASSERT_ON_ERROR(lRetVal);

    // Remove  all 64 filters (8*8)
    memset(RxFilterIdMask.FilterIdMask, 0xFF, 8);
    lRetVal = sl_WlanRxFilterSet(SL_REMOVE_RX_FILTER, (_u8 *)&RxFilterIdMask,
                       sizeof(_WlanRxFilterOperationCommandBuff_t));
    ASSERT_ON_ERROR(lRetVal);

    lRetVal = sl_Stop(SL_STOP_TIMEOUT);
    ASSERT_ON_ERROR(lRetVal);

    InitializeAppVariables();

    return lRetVal; // Success
}

//*****************************************************************************
//! \brief This function checks the LAN connection by pinging the AP's gateway
//!
//! \param  None
//!
//! \return 0 on success, negative error-code on error
//!
//*****************************************************************************
static long CheckLanConnection()
{
    SlPingStartCommand_t pingParams = {0};
    SlPingReport_t pingReport = {0};

    long lRetVal = -1;

    CLR_STATUS_BIT(g_ulStatus, STATUS_BIT_PING_DONE);
    g_ulPingPacketsRecv = 0;
```

```
    // Set the ping parameters
    pingParams.PingIntervalTime = PING_INTERVAL;
    pingParams.PingSize = PING_PKT_SIZE;
    pingParams.PingRequestTimeout = PING_TIMEOUT;
    pingParams.TotalNumberOfAttempts = NO_OF_ATTEMPTS;
    pingParams.Flags = 0;
    pingParams.Ip = g_ulGatewayIP;

    // Check for LAN connection
    lRetVal = sl_NetAppPingStart((SlPingStartCommand_t*)&pingParams, SL_AF_INET,
                           (SlPingReport_t*)&pingReport, SimpleLinkPingReport);
    ASSERT_ON_ERROR(lRetVal);

    // Wait for NetApp Event
    while(!IS_PING_DONE(g_ulStatus))
    {
#ifndef SL_PLATFORM_MULTI_THREADED
        _SlNonOsMainLoopTask();
#endif
    }

    if(0 == g_ulPingPacketsRecv)
    {
        //Problem with LAN connection
        ASSERT_ON_ERROR(LAN_CONNECTION_FAILED);
    }

    // LAN connection is successful
    return SUCCESS;
}




//*****************************************************************************
//
//! \brief Connecting to a WLAN Accesspoint
//!
//!  This function connects to the required AP (SSID_NAME) with Security
//!  parameters specified in te form of macros at the top of this file
//!
```

```
//! \param  None
//!
//! \return  None
//!
//! \warning    If the WLAN connection fails or we don't aquire an IP
//!            address, It will be stuck in this function forever.
//
//*****************************************************************************
static long WlanConnect()
{
    SlSecParams_t secParams = {0};
    long lRetVal = 0;

    secParams.Key = (signed char*)SECURITY_KEY;
    secParams.KeyLen = strlen(SECURITY_KEY);
    secParams.Type = SECURITY_TYPE;

    lRetVal = sl_WlanConnect((signed char*)SSID_NAME, strlen(SSID_NAME), 0, &secParams, 0);
    ASSERT_ON_ERROR(lRetVal);

    // Wait for WLAN Event
    while((!IS_CONNECTED(g_ulStatus)) || (!IS_IP_ACQUIRED(g_ulStatus)))
    {
        // Toggle LEDs to Indicate Connection Progress
        GPIO_IF_LedOff(MCU_IP_ALLOC_IND);
        MAP_UtilsDelay(800000);
        GPIO_IF_LedOn(MCU_IP_ALLOC_IND);
        MAP_UtilsDelay(800000);
    }

    return SUCCESS;

}



int counter = 0;
//*****************************************************************************
//
//! \brief Send a UDP packet to the central HUB
//!
```

```
//!
//! \param  usPort: The port to send to
//! \param  sample: The power in Watts
//!
//! \return  None
//
//*****************************************************************************
int BsdUdpClient(unsigned short usPort, int sample)
{
    short           sTestBufLen;
    SlSockAddrIn_t  sAddr;
    int             iAddrSize;
    int             iSockID;
    int             iStatus;

    sTestBufLen  = 12;
    char g_cBsdBuf[sTestBufLen];

    //filling the UDP server socket address
    sAddr.sin_family = SL_AF_INET;
    sAddr.sin_port = sl_Htons((unsigned short)usPort);
    sAddr.sin_addr.s_addr = sl_Htonl((unsigned int) 0xC0A80180/*0xC0A8001E*/);

    iAddrSize = sizeof(SlSockAddrIn_t);

    // creating a UDP socket
    iSockID = sl_Socket(SL_AF_INET,SL_SOCK_DGRAM, 0);
    if( iSockID < 0 )
    {
        // error
        //ASSERT_ON_ERROR(SOCKET_CREATE_ERROR);
    }

       // Populate the packet
       g_cBsdBuf[0] = ID;
       g_cBsdBuf[1] = 0;
       g_cBsdBuf[2] = 0;
       g_cBsdBuf[3] = 0;
       g_cBsdBuf[4] = 0;
       g_cBsdBuf[5] = 0;
       g_cBsdBuf[6] = 0;
```

```
        g_cBsdBuf[7] = 0;
        memcpy(g_cBsdBuf + 8, &sample, 4);

        if(counter > 20)
        {
                counter = 0;
        }

        // sending packet
        iStatus = sl_SendTo(iSockID, g_cBsdBuf, sTestBufLen, 0,
                                                (SlSockAddr_t *)&sAddr, iAddrSize);
        if( iStatus <= 0 )
        {
                // error
                sl_Close(iSockID);
                //ASSERT_ON_ERROR(SEND_ERROR);
        }

    UART_PRINT("Sent %u packets successfully\n\r",1);

    //closing the socket after sending 1000 packets
    sl_Close(iSockID);

    return SUCCESS;
}

//*****************************************************************************
//
//! \brief Perform an RMS calculation
//!
//!
//! \param  val: Value of sample
//! \param  hint: Hint used for algorithm
//!
//! \return  None
//
//*****************************************************************************
static double one_step_newton_raphson_sqrt(double val, double hint)
{
        double probe;
        if (hint <= 0) return val /2;
```

```
        probe = val / hint;
        return (probe+hint) /2;
}

// Variables used for the RMS algorithm
static double      acc_load_current1 = 0.0;           // accumulator = (I1*I1 + I2*I2 + ... +
In*In)
static double      rms_current1 = 1.0;
float       adc_value1, inst_current1;
double      tmp_rms_current1;


static double      acc_load_current2 = 0.0;           // accumulator = (I1*I1 + I2*I2 + ... +
In*In)
static double      rms_current2 = 1.0;
float       adc_value2, inst_current2;
double      tmp_rms_current2;



// Now the <rms_current> is the APPROXIMATE RMS current

//*****************************************************************************
//
//! \brief Start simplelink, connect to the ap and run the ping test
//!
//! This function starts the simplelink, connect to the ap and start the ping
//! test on the default gateway for the ap
//!
//! \param[in]  pvParameters - Pointer to the list of parameters that
//!             can bepassed to the task while creating it
//!
//! \return  None
//
//*****************************************************************************
void WlanStationMode( void *pvParameters )
{

    long lRetVal = -1;
    InitializeAppVariables();

    //
    // Following function configure the device to default state by cleaning
```

```
// the persistent settings stored in NVMEM (viz. connection profiles &
// policies, power policy etc)
//
// Applications may choose to skip this step if the developer is sure
// that the device is in its default state at start of applicaton
//
// Note that all profiles and persistent settings that were done on the
// device will be lost
//
lRetVal = ConfigureSimpleLinkToDefaultState();
if(lRetVal < 0)
{
    if (DEVICE_NOT_IN_STATION_MODE == lRetVal)
    {
        UART_PRINT("Failed to configure the device in its default state\n\r");
    }

    LOOP_FOREVER();
}

UART_PRINT("Device is configured in default state \n\r");

//
// Assumption is that the device is configured in station mode already
// and it is in its default state
//
lRetVal = sl_Start(0, 0, 0);
if (lRetVal < 0 || ROLE_STA != lRetVal)
{
    UART_PRINT("Failed to start the device \n\r");
    LOOP_FOREVER();
}

UART_PRINT("Device started as STATION \n\r");

//
//Connecting to WLAN AP
//
lRetVal = WlanConnect();
if(lRetVal < 0)
{
```

```
    UART_PRINT("Failed to establish connection w/ an AP \n\r");
    LOOP_FOREVER();
}

UART_PRINT("Connection established w/ AP and IP is aquired \n\r");

//Set the pin type as ADC for required pin
   PinTypeADC(PIN_58, 0xFF);
   PinTypeADC(PIN_59, 0xFF);

   //Enable the ADC channel ADCChannel
   UART_PRINT("Line 873\n\r");
   ADCChannelEnable(ADC_BASE, ADC_CH_1);
   ADCChannelEnable(ADC_BASE, ADC_CH_2);

   //Optionally configure internal timer for time stamping
   UART_PRINT("Line 877\n\r");
   ADCTimerConfig(ADC_BASE, 2^17);
   UART_PRINT("Line 879\n\r");
   ADCTimerEnable(ADC_BASE);

   //Enable the ADC module
   UART_PRINT("Line 883\n\r");
   ADCEnable(ADC_BASE);

   //Read out the ADC samples using following code
   UART_PRINT("Line 887\n\r");
   int oldTime = 0;
   while(true)
   {
          int i;
          GPIO_IF_LedOn(MCU_RED_LED_GPIO);
          for(i = 0; i < 1000; i++)
          {
                 if( ADCFIFOLvlGet(ADC_BASE, ADC_CH_1) )
                 {
                        // Read the ADC samples from the registers
                        unsigned long ulSample1 = ADCFIFORead(ADC_BASE, ADC_CH_1);
                        unsigned long ulSample2= ADCFIFORead(ADC_BASE, ADC_CH_2);
                        int sample1 = (ulSample1 >> 2) & 0xFFF;
                        int sample2 = (ulSample2 >> 2) & 0xFFF;
```

```
                            int timestamp = (ulSample1 >> 14) & 0x1FFFF;
                            float voltage1 = 0.0003578 * sample1;
                            float voltage2 = 0.0003578 * sample2;

                            // Account for the DC offset
                            voltage1 -= 0.762;
                            voltage2 -= 0.75;

                            // Calculate the real instantanous value from the ADC reading
                            inst_current1 = (voltage1) / (1.85 / (642));    // 10bit ADC,
Voltage ref. 2.5V, so formula is: x=(adc/1024)*2.5V
                            inst_current2 = (voltage2) / (8.35 * 5 * 0.004);

                            double RMS_SIZE = 64;
                            // Update the RMS value with the new instananous value:
                            // Substract 1 sample from the accumulator (sample size is 512,
so divide accumulator by 512 and substract it from the accumulator)
                            acc_load_current1 -= (acc_load_current1 / RMS_SIZE);
                            inst_current1 *= inst_current1;          // square the
instantanous current
                            acc_load_current1 += inst_current1;      // Add it to the
accumulator

                            acc_load_current2 -= (acc_load_current2 / RMS_SIZE);
                            inst_current2 *= inst_current2;          // square the
instantanous current
                            acc_load_current2 += inst_current2;      // Add it to the
accumulator

                            tmp_rms_current1 = (acc_load_current1 / RMS_SIZE);
                            rms_current1 = one_step_newton_raphson_sqrt(tmp_rms_current1,
rms_current1);

                            tmp_rms_current2 = (acc_load_current2 / RMS_SIZE);
                            rms_current2 = one_step_newton_raphson_sqrt(tmp_rms_current2,
rms_current2);         // Polish RMS value

                            osi_Sleep(1); // sleeps for a MS (1KHz sampling rate)

                            oldTime = timestamp;
                        }


                }
```

```
                // Calculate power
                double power = rms_current1 * rms_current2;

                // Trigger LED's
                GPIO_IF_LedOff(MCU_RED_LED_GPIO);
                GPIO_IF_LedOn(MCU_GREEN_LED_GPIO);

                // Print and send power
                UART_PRINT("%f , %f, %f\n\r", rms_current1, rms_current2, power);
                BsdUdpClient(12000, (int) power);
                GPIO_IF_LedOff(MCU_GREEN_LED_GPIO);


        }


    LOOP_FOREVER();

}


void PrintData( void *pvParameters )
{
        while(true)
        {
                UART_PRINT("%f | %f\r\n", rms_current1, rms_current2);
                osi_Sleep(500);
        }
}
//*****************************************************************************
//
//! Application startup display on UART
//!
//! \param  none
//!
//! \return none
//!
//*****************************************************************************
static void
DisplayBanner(char * AppName)
```

```
{

    UART_PRINT("\n\n\n\r");
    UART_PRINT("\t\t ************************************************\n\r");
    UART_PRINT("\t\t           CC3200 %s Application       \n\r", AppName);
    UART_PRINT("\t\t ************************************************\n\r");
    UART_PRINT("\n\n\n\r");
}
//*****************************************************************************
//
//! \brief  Board Initialization & Configuration
//!
//! \param  None
//!
//! \return None
//
//*****************************************************************************
static void
BoardInit(void)
{
// In case of TI-RTOS vector table is initialize by OS itself
#ifndef USE_TIRTOS
    //
    // Set vector table base
    //
#if defined(ccs) || defined(gcc)
    MAP_IntVTableBaseSet((unsigned long)&g_pfnVectors[0]);
#endif
#if defined(ewarm)
    MAP_IntVTableBaseSet((unsigned long)&__vector_table);
#endif
#endif //USE_TIRTOS

    //
    // Enable Processor
    //
    MAP_IntMasterEnable();
    MAP_IntEnable(FAULT_SYSTICK);

    PRCMCC3200MCUInit();
}
```

```
//*****************************************************************************
//                            MAIN FUNCTION
//*****************************************************************************
void main()
{
    long lRetVal = -1;

    //
    // Board Initialization
    //
    BoardInit();

    //
    // configure the GPIO pins for LEDs,UART
    //
    PinMuxConfig();

    //
    // Configure the UART
    //
#ifndef NOTERM
    InitTerm();
#endif  //NOTERM

    //
    // Display Application Banner
    //
    DisplayBanner(APPLICATION_NAME);

    //
    // Configure all 3 LEDs
    //
    GPIO_IF_LedConfigure(LED1|LED2|LED3);

    // switch off all LEDs
    GPIO_IF_LedOff(MCU_ALL_LED_IND);

    //
    // Start the SimpleLink Host
```

```
    //
    lRetVal = VStartSimpleLinkSpawnTask(SPAWN_TASK_PRIORITY);
    if(lRetVal < 0)
    {
        ERR_PRINT(lRetVal);
        LOOP_FOREVER();
    }

    //
    // Start the WlanStationMode task
    //
    lRetVal = osi_TaskCreate( WlanStationMode, \
                              (const signed char*)"Wlan Station Task", \
                              OSI_STACK_SIZE, NULL, 1, NULL );
    if(lRetVal < 0)
    {
        ERR_PRINT(lRetVal);
        LOOP_FOREVER();
    }

    /*lRetVal = osi_TaskCreate( PrintData, \
                                  (const signed char*)"Print Data Task", \
                                  OSI_STACK_SIZE, NULL, 1, NULL );
       if(lRetVal < 0)
       {
              ERR_PRINT(lRetVal);
              LOOP_FOREVER();
       }*/

    //
    // Start the task scheduler
    //
    osi_start();
  }

//*****************************************************************************
//
// Close the Doxygen group.
//! @}
//
//*****************************************************************************
```

12.4.2 Web Server Code (Python)

```
#!flask/bin/python

from flask import Flask, jsonify, request

import random

import sqlite3

import time


app = Flask(__name__)


@app.route('/api/powerdata/', methods=['GET'])

def getPowerData():


    # Get the initial time

    startTime = int(round(time.time() * 1000))

    # Get the various variables from the request URL

    dates = request.args.to_dict()

    start = int(dates['start'])

    end = int(dates['end'])

    station = int(dates['station'])

    maxPoints = 50 # This is just hardcoded in the server

    statement = ''


    # If start = end = 0, we want the most recent point

    if start == 0 and end == 0:

        statement = ("SELECT * FROM may1725 WHERE timestamp=(SELECT MAX(timestamp) FROM
may1725) and stationID=%d" % (station))

    elif start == -1 and end == -1: # Get all points for a certain station

        statement = ("SELECT * FROM may1725 WHERE stationID=%d" % (station))

        # Get stuff from the database!

        conn = sqlite3.connect('may1725.db')
```

```
        c = conn.cursor()
        cursor = c.execute(statement)
        rows = cursor.fetchall()

        # Decimate the array if it's too big
        if len(rows) < maxPoints:
            # Iterate through each row, creating an appropriate JSON object
            points = []
            for row in rows:
                points.append({'timestamp': row[0], 'power': row[2], 'station_id' : row[1]})
            return jsonify(points)
        else:
            # Iterate through each row, creating an appropriate JSON object
            points = []
            i = 0
            for row in rows:
                points.append({'timestamp': row[0], 'power': row[2], 'station_id' : row[1]})
            return jsonify(points[-maxPoints:])

    else:
        statement = ("SELECT * FROM may1725 WHERE stationID=%d AND timestamp BETWEEN %d AND
%d" % (station, start, end))

    # Get stuff from the database!
    conn = sqlite3.connect('may1725.db')
    c = conn.cursor()
    cursor = c.execute(statement)
    rows = cursor.fetchall()

    # Iterate through each row, creating an appropriate JSON object
    points = []
    for row in rows:
```

```
        points.append({'timestamp': row[0], 'power': row[2], 'station_id' : row[1]})


        # Limit the number of points returned
    if len(points) > maxPoints:
        stepSize = len(points) // maxPoints # Gets the step size of the slice
        points = points[0:len(points) - 1:stepSize]


    print("Total time: %d\n" % (int(round(time.time() * 1000)) - startTime))
    return jsonify(points)


# Handle a 404 Error
@app.errorhandler(404)
def not_found(error):
    return jsonify({'error': 'Not found'})


if __name__ == '__main__':
    app.run(debug=True)
```

12.4.3 Web Application Code

The code for the web application, given in a single Javascript file:

```
// Global variables
var highChart;
var liveActive = true;



$('.ui.dropdown')
  .dropdown()
;


// Plot an array of JSON power data items (ie, from the server directly)
function plotData(items)
```

```
{
	//clearChart();
	var series = highChart.series[0],
		 shift = series.data.length > 20; // shift if the series is
						// longer than 20

	for(var i = 0; i < items.length; i++)
	{
		var item = items[i];
		var date = new Date(item['timestamp']*1000);
		if(highChart.series[0].data.length == 0)
		{
			highChart.series[0].addPoint([item['timestamp'], item['power']], true,
shift);
		}
		else if(highChart.series[0].data[highChart.series[0].data.length - 1].x !=
item['timestamp'])
		{
			highChart.series[0].addPoint([item['timestamp'], item['power']], true,
shift);
		}
	}
}

// Plot an array of data items
function plotAllData(items)
{
	// Make sure to clear the chart first!
	clearChart();
	for(var i = 0; i < items.length; i++)
	{
		var item = items[i];
		var date = new Date(item['timestamp']*1000);
```

```
                highChart.series[0].addPoint([item['timestamp'], item['power']], true);
        }
}



/*******************************
* Class PowerDataRequest
*******************************/
PowerDataRequest = function(callback ) {
    this._callback = callback;
    this._loadData();
    highChart.hideLoading();
};


PowerDataRequest.prototype._loadData = function() {
    $.ajax({
        url: "http://localhost:5000/api/powerdata/?start=0&end=10&station=0",
        //url: "http://localhost:5000/api/powerdata/?start=1484476930&end=1484477110",
        success: this._onDataReceived.bind( this ),
        cache: true,
        dataType: 'json',
                error: function (jqXHR, exception) {
                        console.log(jqXHR);
                        // Your error handling logic here..
                }
    });
};


// Callback for JSON packet received
PowerDataRequest.prototype._onDataReceived = function( rawData ) {
    var highchartsData = this._transformDataForHighCharts( rawData );
```

```
    this._callback( highchartsData );
};

// Callback for AJAX requests
function onDataReceived( rawData ) {
    plotData(rawData);
}

// Clears all data from the chart
function clearChart()
{
       highChart.series[0].setData([]);
}

// Transforms the JSON data into a format readable by Highcharts
PowerDataRequest.prototype._transformDataForHighCharts = function( rawData ) {

    var data = [];
    var i;

    for( i = 0; i < rawData.length; i++)
       {
              // Do a simple packet parse
        data.push([
            ( new Date(rawData[i]['timestamp'] * 1000) ).getTime(),
            parseFloat(rawData[i]['power'])
        ]);
    }

    return data;
};
```

```
// Called when data is ready to be plotted
var onDataReady = function( highchartsData ) {
    highChart.addSeries({
        color: "green",
        name: "Senior Design",
        data: highchartsData
    });

    highChart.hideLoading();
};

/**
 * Request data from the server, add it to the graph and set a timeout
 * to request again
 */
function requestData()
{
    $.ajax({
        url: 'http://localhost:5000/api/powerdata/?start=0&end=0&station=0',
        success: function(point) {

                        // add the point
                        if(liveActive == true)
                        {
                                plotData(point);
            }

            // call it again after one second
            setTimeout(requestData, 1000);
        },
        cache: false
    });
```

```
}


function viewPowerClicked()

{

	// Find the start and endtimes in UNIX time

	startTime = Math.round(startDatePicker.getDate().getTime() / 1000);

	endTime = Math.round(endDatePicker.getDate().getTime() / 1000);



	// Create the AJAX request

	$.ajax({

	 url: "http://localhost:5000/api/powerdata/?start=" + startTime + "&end=" + endTime +
"&station=" + 0,

	 success: plotAllData,

	 cache: true,

	 dataType: 'json',

		error: function (jqXHR, exception) {

			console.log(jqXHR);

			// Your error handling logic here..

		}

    });

}



function viewLiveClicked()

{

	if(!liveActive)

	{

		clearChart();

		liveActive = true;

	}

}



function setupChart()
```

```
{
        // Sets up the config for the chart
        var highChartsConfig =
        {
                chart: {
             renderTo: 'container',
             defaultSeriesType: 'spline',
             events: {
                 load: requestData
             }
         },
                tooltip: {
                        dateTimeLabelFormats: {hour:'%H:%M %p'}
                },
                title: { text: 'Power Usage Chart' },
                series: [{
             name: 'Power Data',
             data: []
         }],
                plotOptions: { line: { marker: { enabled: false } } },
                xAxis: {
                        type: 'datetime',
                        labels: {
                                formatter: function() {
                                  return Highcharts.dateFormat('%d %b %Y', this.value * 1000); //
milliseconds not seconds
                                },
                          }
                },
                yAxis: {
                        title: 'Watts' ,
                        labels: {
```

```
                            formatter: function () {
                        return this.axis.defaultLabelFormatter.call(this) + ' W';
                    }
                        },
                        min: 0,
                        max: 125
                }
        };

        Highcharts.setOptions({
         global: {
             useUTC: false
         }
    });

        highChart = new Highcharts.Chart(highChartsConfig );
        //highChart.showLoading( 'Loading data...' );
        //new PowerDataRequest( onDataReady );
}



function getStations()
{
    $.ajax({
         url: "http://localhost:5000/api/stations",
         success: onStationsReceived.bind( this ),
         cache: true,
         dataType: 'json',
                error: function (jqXHR, exception) {
                        console.log(jqXHR);
                        // Your error handling logic here..
                }
```

```
    });
}

// Callback for a station being clicked
function stationClicked(id)
{
        // Set up the dates
        liveActive = false;
        var start = new Date();
        var end = new Date();
        start.setMonth(end.getMonth() - 1);
        startDatePicker.setDate(start);
        endDatePicker.setDate(end);

        startTime = Math.round(start.getTime() / 1000);
        endTime = Math.round(end.getTime() / 1000);

        // make AJAX request
        console.log('Station clicked:' + id);
        $.ajax({
         url: "http://localhost:5000/api/powerdata/?start=" + startTime + "&end=" + endTime +
"&station=" + id,
         success: plotAllData,
         cache: true,
         dataType: 'json',
                error: function (jqXHR, exception) {
                        console.log(jqXHR);
                        // Your error handling logic here..
                }
    });
}
```

```
// Callback for STATIONS AJAX request

var onStationsReceived = function( rawData )

{

	// Add a heading

	document.getElementById('mySidenav').innerHTML += "<h2 class=\"ui header\">Station
List</h2>";



    for( i = 0; i < rawData.length; i++)

	{

		// Add each station

		document.getElementById('mySidenav').innerHTML += '<a href=\"#\"
onclick=\"stationClicked(' + rawData[i]['id'] + ')\">' + rawData[i]['id'] + ": " +
rawData[i]['name'] + '</a><br>';

    }

};



// Variables for the pickers

var startDatePicker;

var endDatePicker;



// set up the stations menu!

$(function()

{

	getStations();

	setupChart();



	// Set up the date pickers

	startDatePicker = new Pikaday({

		field: document.getElementById('startdatepicker'),

		position: 'top left'

	});



	endDatePicker = new Pikaday({
```

```
                field: document.getElementById('enddatepicker'),
                position: 'top left'
        });
});
```

And the corresponding HTML file for the main page:

```
<!DOCTYPE html>
<!-- saved from url=(0044)http://kenedict.com/networks/worldcup14/vis/ , thanks Andre!-->
<html><head><meta http-equiv="content-type" content="text/html; charset=UTF8">
  <title>Project Neptune</title>

  <script type="text/javascript" src="js/jquery-3.1.1.min.js"></script>
  <script type="text/javascript" src="js/semantic.min.js"></script>
  <script type="text/javascript" src="js/goldenlayout.min.js"></script>
  <script type="text/javascript" src="js/Chart.bundle.min.js"></script>
  <script type="text/javascript" src="js/highcharts.js"></script>
  <script src="http://code.highcharts.com/modules/exporting.js"></script>
  <script src="http://code.highcharts.com/modules/offline-exporting.js"></script>
  <script type="text/javascript" src="js/pikaday.js"></script>


  <!--<link type="text/css" rel="stylesheet" href="css/vis.min.css">-->
  <link type="text/css" rel="stylesheet" href="css/semantic.min.css">
  <link type="text/css" rel="stylesheet" href="css/goldenlayout-base.css">
  <link type="text/css" rel="stylesheet" href="css/goldenlayout-light-theme.css">
  <link type="text/css" rel="stylesheet" href="css/neptune.css">
  <link type="text/css" rel="stylesheet" href="css/pikaday.css">

</head>

<body>
```

```
<div id="main">


    <!-- Top dropdown menu -->
    <div class="ui top attached menu">
      <div class="ui dropdown icon item">File
        <div class="menu">
          <div class="item">Save...</div>
        </div>
      </div>
      <div class="ui dropdown icon item">Options
        <div class="menu">
          <div class="item">View Stations...</div>
          <div class="item">Preferences...</div>
        </div>
      </div>
    </div>


       <div class="ui center aligned grid">
              <div id="container" class="sixteen wide column" style="width:100%;
height:400px;"></div>


              <div class="eight wide column">
                     <label>Start date: </label><input type="text" id="startdatepicker"><br>
                     <label>End date: </label><input type="text" id="enddatepicker"><br>
                     <button class="ui button" onclick="viewPowerClicked()"
style="margin-top:10px;">View Past Data</button><br>
                     <button class="ui button" onclick="viewLiveClicked()"
style="margin-top:10px;">View Live Data</button>
              </div>


              <div class="eight wide column">
                     <div id="mySidenav"/>
              </div>
```

```
        </div>
</div>
</body>
<script type="text/javascript" src="js/main.js"></script>
```

12.5 Project Timeline