The George Washington University
School of Engineering and Applied Science
Department of Electrical and Computer Engineering
ECE4925W - Section 10

Final Product Review

The Modern Alexa-Controlled Smart Home

Matt Taylor - Electrical Engineering
Agam Mittal - Electrical Engineering
Marie-Laure Brossay - Computer Engineering

Professor Amir Aslani
Submission Date: 04/10/2019

Abstract

This report outlines the design for a smart plug designed to control power to common electrical appliances around the house and measure the power usage of those appliances. The advantage of using a smart plug is the ability to remotely, by using voice control, turn electrical devices on and off, either by manual control or according to a predefined program. Although such smart plugs do exist in the market today, they are expensive and limited in their functionality. The advantage of the design described here is that when asked, it can tell the user the power usage of the appliance, so that the user may know which device in the home may be leading to a higher electrical bill. This technology utilizes the Amazon Alexa voice service to connect to a central router via a Wi-Fi connection; the router in turn connects to a series of smart plugs via Bluetooth. These plugs may be connected to any standard wall socket. One end of the plug has a socket to connect an appliance, such as a television or a lamp, and the other end contains a connection to the wall outlet. Power data is stored in a cloud database and can be retrieved by the user via a series of commands. We hope that our design will not only help users save time and money but also have a positive impact on the environment with reduced energy usage in the long run.

Introduction

The Modern Alexa-Controlled Smart Home enables users to control appliances by speaking to their Amazon Alexa device, and provides information regarding patterns and performance. Common applications of the "smart home" are as basic as routine "on-off" tasks, such as controlling chargers, lights, televisions, coffee machines, or electronic blinds in the morning when a person wakes up, turning on an electric heater or fan for someone who is physically impaired, or opening/closing an electric gate from a room inside your house.

With a forecasted 165 million smart home units being installed in homes in 2020, it is clear people are looking for competent and effortless systems that can provide control over their technology. The concept has a slew of competitors, including smart plugs from Wemo, TP-Link, and Etekcity; however, the market lacks a design which has "fully integrated" switches. This project seeks to create a network of smart plugs around the home, centered around a central router. Plugs can relay information back to a central router which can inform the user about how their home functions day-to-day. The smart plugs will have Bluetooth capability for communication between themselves and a central router, and the central router will use Wi-Fi to connect to Alexa. Alexa allows the user a voice interface for interacting with their devices.

The global revenue of the smart home automation market is forecasted to reach 20.8 billion U.S dollars in 2020, so it would be financially rewarding to provide a product that appeals to customers. Two major factors that affect this are the size of the product and the product's usability. The project aims to make the system as small as possible and easy to install and use, while packing it with features that exceed those in the systems already on the market. The design requires minimal setup, includes intuitive voice commands, and aims to make life around the home easier for the user, rather than more complicated. The product may be particularly helpful for the elderly, who may rely on a voice assistant to carry out tasks such as turning on a coffee machine or switching off the lights. This also helps younger users who can't easily reach the switches that are built into the appliance, such as on tall lamps.

Security is also a major concern for many users when it comes to smart home applications. In this area, Amazon has shown it is a leader and has demonstrated a commitment to security of user data through Amazon Web Services; AWS remains the top choice for cloud storage among industry professionals. In AWS, all user data can be backed up and is encrypted, ensuring that data will not be lost and service outages will be kept to a minimum within a secure infrastructure.

Overall System Requirements & Specifications

Our project can be broken down into 4 subprojects:

1. The connection between Raspberry Pi and Alexa.
2. The connection between Raspberry Pi and Arduinos.
3. The connection between Arduino and appliances.
4. Power measurement of appliance.

Critical Parameters: In this system, there are several critical parameters. The most important function of this system is the passing of messages from vocal input to the output of a product being turned on or off. The most important aspects of the system have been *italicized* below. These consist of the Alexa being able to take in a vocal cue, which sends a command to an NGROK server. The Raspberry Pi will pull from the NGROK server to send information to the Arduino through Bluetooth. The Arduino will then turn on the plug connected to the item. Then, the Arduino will send back a signal to the Raspberry Pi, letting it know that the item has been turned on. The exact specifications for these aspects are below, along with which requirements are critical to the system. These portions are italicized.

Raspberry Pi <--> Alexa

Table 1. *Requirements & Specifications for Raspberry Pi/Alexa Connection*

Requirements	**Specifications**
Alexa shall recognize the specific commands from user such as "Turn plug 1 on". There shall be a list of these commands, and each relay shall have an associated name which the Alexa can recognize as belonging to it.	Alexa correctly identifying what the user is saying is dependent on the Alexa itself. A few examples of these phrases are: ● "Turn on the living room light" (where in this case living room light is the name of a relay) ● "Turn on the TV" ● "What does my power consumption look like?"
Raspberry Pi must be able to obtain commands from the NGROK web server.	This will be done using a 2.4 GHz 802.11n Wi-Fi connection.
Central controller (RPi) and plugs (Arduino) will be easy to install.	The Raspberry Pi shall have a simple Wi-Fi login, then from there it shall connect to the Arduinos autonomously upon startup.
System will provide feedback to the consumer that it has received their command.	Arduino will send Bluetooth signal back to Raspberry Pi once it receives a command. RPi will communicate with Alexa to tell user that command has been received.
System will be low cost in terms of hardware and software.	Combined cost of controller (Raspberry Pi) and plug (Arduino) will be under $100.

System will be unobtrusive and occupy minimal space.	Central Controller (Raspberry Pi) shall be no larger than a Wi-Fi Router.

Raspberry Pi <--> Arduino

Table 2. *Requirements & Specifications for Raspberry Pi/Arduino Connection*

Requirements	**Specifications**
The Raspberry Pi must be able to connect to every Arduino in the house.	This will be done using Bluetooth Low Energy 5.0 technology, with a range of up to 200m.
Arduino must send a confirmation signal to the Raspberry Pi immediately after it receives a command.	The Arduino code will include a transmitting function that is called immediately as the Arduino receives a command from the Raspberry Pi.
Arduino must send power usage of appliance to Raspberry Pi to be accessed through Alexa.	The Arduino will be collecting information from the current transformer hooked up to the appliance. The Arduino script will contain a module that sends the power information constantly to the Raspberry Pi, which will then store it and provide information to Alexa on command.
The connection must be able to handle a specified amount of Arduino connections.	The Raspberry Pi must be able to process *at least* 3 Arduino subsystems.
Raspberry Pi will ensure that communication between Arduinos is working on an ongoing basis.	Raspberry Pi will send a generic signal out to all Arduinos every 2 minutes. When Arduinos receive these signals, they will send a signal of their own back, each containing unique information about the plug, so that Raspberry Pi knows which Arduinos are plugged in and powered.

Arduino <--> Appliances

Table 3. *Requirements & Specifications for Arduino/Appliances Connection*

Requirements	**Specifications**
System must have a manual on/off function.	A switch will be wired into the Arduino circuit, and can turn on/off the relay switch.
System will turn on appliance when user commands Alexa to do so.	Relay will turn on when Arduino receives the correct Bluetooth signal from the Raspberry Pi.
System must measure power drawn from each plug.	The Arduinos will contain a current transformer which provides current readings back to the Arduino.
Each plug can handle a pull of up to 1800 Watts.	Relay coil must be rated at 15A, 120V.
System must have visual feedback as to whether it is on or off.	LED will be included in circuit and will be red to display off or green to display on.
System must act within a relatively quick time frame.	The system should turn on appliances within 5 seconds of user sending request. This should hold true with a variety of distances and obstacles between the Alexa/Raspberry Pi/Arduinos.
Arduino will be connected to 1 plug, but will have the flexibility to power up to 6 plugs.	The original circuit board will have 1 relay connected on it, but it will have ports which can connect to a breakout board consisting of up to 6 relays.
Plug system will not exceed the size of 3in x 3in x 6in.	A custom PCB will be designed which will compactly hold the Arduino, relay, and current transformer.

Power Measurement

Table 4. *Requirements & Specifications for Power Measurement Module*

Requirements	**Specifications**
Alexa shall tell the user the power draw of a plug when the user asks.	A current transformer will be wrapped around an output wire of the relay. It will be connected into an analog input pin of the Arduino. The Arduino will relay this information to the Raspberry Pi via Bluetooth, which it will relay to the Alexa database for the Alexa device to tell the user.
Appliance controller will measure the power output of the appliance accurate to +/-5%.	The current transformer will be calibrated and tested thoroughly with a multimeter to ensure that it takes the same current measurements as the lab multimeter.
The user will be able to retrieve information from the last 24 hours.	The data sent to the Raspberry Pi will be stored using a MySQL database in AWS RDS, a relational database management system. This information will be available to be pulled from whenever the user asks through Alexa.
The power draw information will be updated every second.	The Arduino will take an average of the analog values it has received from the current transformer over the past 1000ms. It will then send that value via Bluetooth every second to the Raspberry Pi to store in its database.

Approach to Overall Design

Current Design

The objective is to design, develop, and test an Arduino-controlled "smart plug appliance controller" that will communicate with a central Raspberry Pi-driven router. The smart plug will be triggered via voice commands from the user, which are sent to the Raspberry Pi router through an Alexa-compatible device. The router will then communicate with these various Arduino-controlled plugs in the environment, triggering them to turn on or off the appliance/s connected through them. Figure 1 below shows the general theory of the system.

Figure 1. *Context level diagram*

Amazon Alexa was chosen as the primary voice interaction model because of the ubiquity of Amazon's Echo line of products in the household, and its connectivity with other smart-home products which enables further integration with the smart plug. According to Canalys, a U.K.-based market research firm, Amazon Echo led all smart speaker sales in Q3 2018, selling 6.3 million devices and capturing 31.9% market share. It also offers easy integration with Amazon Web Services, a convenient platform for cloud storage of user data.

The central router consists of a Raspberry Pi Model 3B+, which is equipped with built-in 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, and BLE, and a protective casing with ports for the Raspberry Pi. When a user command is sent to the Amazon Alexa servers, that command is sent to the Raspberry Pi via an HTTP tunnel. The user commands and the operations which they trigger are configured with the help of the Alexa Skills Kit, which is a collection of self-service APIs. The Alexa Skills Kit allows developers to create Alexa Skills, which specifies to Alexa the particular commands that it should recognize for the developer's application. The Skill can pull information from the Alexa device (audio input from the user) and pass this through to the Amazon server.

BLE technology was chosen over other alternatives such as ZigBee for two reasons. While it offers shorter range than ZigBee, it is expected that any smart plugs will be placed throughout the home and therefore within range of the router. It has a higher data rate than ZigBee, meaning the transfer of information will be nearly instantaneous, with short burst speeds of 1 Mbit/second. One of the advantages of Bluetooth is its compatibility with many operating systems, including Android, iOS, macOS, and Windows, which allows users to connect using their personal devices; furthermore, it uses less data and power than ZigBee because it sleeps between bursts.

The user's command will be stored in the cloud, and the Raspberry Pi, through the HTTP tunnel, can obtain the information contained in the command with the help of Ngrok. Ngrok is a software that can secure a tunnel to a local host. Running on the Raspberry Pi, it creates a temporary HTTPS address to which the information on the Amazon server can be sent. There is a Python script continuously running on the Raspberry Pi which utilizes a Python

micro-framework called Flask-Ask. This framework, together with Ngrok, can pull the information from the temporary https address that Ngrok had previously set up onto the Raspberry Pi for the Python program to interpret. The reverse process occurs when the Raspberry Pi wants to communicate back to Alexa, with the Python script sending its information through Ngrok onto the https address and the Alexa skill pulling it onto the Amazon server, which the Alexa device then retrieves and reports back to the user. This design can be utilized for two-way communication, where the system can report back to the user with information or status of their request.

Figure 2. *Functional Block Diagram*

When the Raspberry Pi receives its commands from the Alexa device, it will then send these signals, via its built-in Bluetooth module, to the plug/appliance which the user has specified. The plugs each have an Arduino Nano in them with an HC-05 Bluetooth Transceiver module connected to it. The Arduino will be programmed so that whenever it receives a command from the Raspberry Pi, it will send a confirmation message back. This will be done through incorporating specific functions in the Arduino script to recognize when an interpretable signal has been obtained through the transceiver. The Arduino enables power to the appliance via a relay switch. This will be connected via a driver circuit to a digital pin of the Arduino, which can be powered high or low depending on what command it has received from the Raspberry Pi.

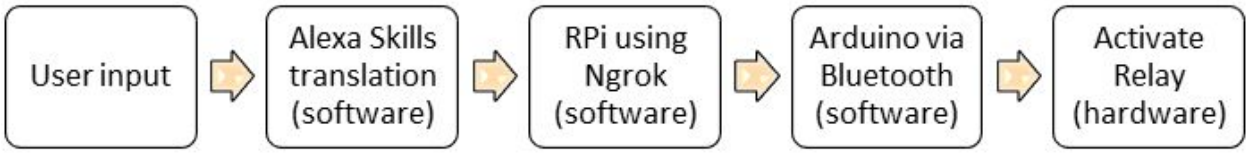

Figure 3. *Level 1 Architecture Block Diagram*

The output wires of the relay switch (to the appliance) will be looped through a current transformer, which measures AC current. The transformer will be connected to an analog input pin of the Arduino, which will then send the information to the Raspberry Pi every second. The

Raspberry Pi will calculate the average of its incoming samples and store the data in order to have several different options for understanding the given data. Once stored and processed, the data will be sent from the Raspberry Pi to the Alexa device when the user commands it to do so. The Raspberry Pi will obtain information in the current moment, an average over the day, and average usage over the entire week.

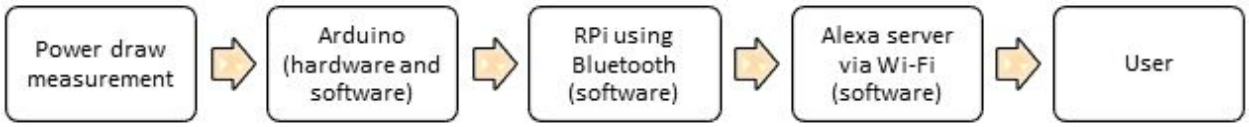

Figure 4. *Level 1 Architecture Block Diagram, user feedback loop*

Evolution of the Current Design

A number of alternate approaches were considered. Per the recommendation of other teams, ZigBee was being considered because of its range and versatility, but it lacks the compatibility of Bluetooth Low Energy. Second, the design outlined above uses Bluetooth for communicating between the Raspberry Pi and the Arduino-based plugs, but an alternate design might use solely Wi-Fi for this communication. However, such a design would present a number of disadvantages. Although Wi-Fi has longer range and faster data transmission rates than Bluetooth, setting up wireless network access on each individual Arduino-plug becomes burdensome for the user, because the wireless network in many homes and businesses is secured and protected. However, with Bluetooth, the proposed design can automatically connect to any Bluetooth devices detected within range, and using the mesh capability of Bluetooth Low Energy, devices can have nearly unlimited range, as they can access “out-of-range” devices through closer devices. With the presence of a central router in our design - a feature that other smart-plugs do not have - there is one-time Wi-Fi set-up. An alternate design was considered, which might utilize smart-plugs that connect to Alexa alone, but again, this would require the Wi-Fi to be set-up more than once, and a central router is used to combat this. For storing data in the cloud, using a SEAS server was considered, but ultimately it was decided that this would introduce an additional size and security disadvantage, and that Amazon Web Services would be easier to integrate with Alexa.

System Functional Diagram

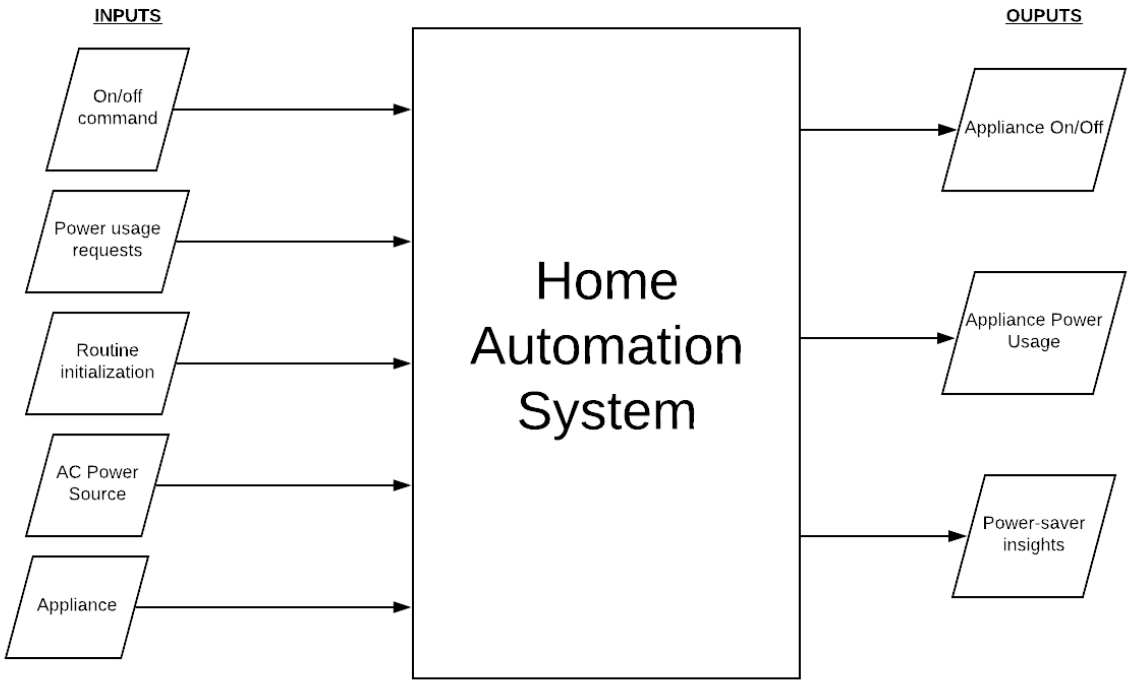

Figure 5. *Level 0 Block Diagram*

Subsystem Functional Diagram

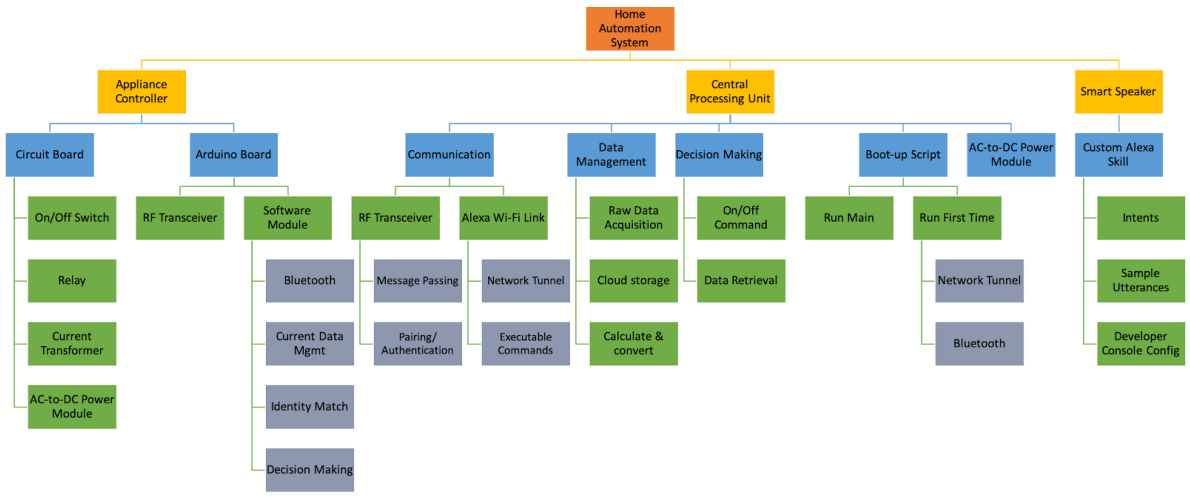

Figure 6. *Subsystem Functional Diagram*

In Figure 6 shown above, each design level is represented by a unique color. Level 0 is orange, Level 1 is yellow, Level 2 is blue, Level 3 is green, and Level 4 is gray.

1. Appliance Controller Requirement and Specifications

The appliance controller shall:

1. Measure the power drawn from each plug.
2. Handle AC pull of up to 1800 Watts from the appliance.
3. Turn on the appliance when the user commands Alexa to do so.
4. Act within a relatively quick time frame.

The appliance controller specifications:

1. The Arduinos will contain a current transformer which provides current readings back to the Arduino.
2. Relay coil must be rated at 15A, 120V.
3. Relay will turn on when Arduino receives the correct Bluetooth signal from the Raspberry Pi.
4. System should turn on appliances within 5 seconds of user sending request. This should hold true with a variety of distances and obstacles between the Alexa/Raspberry Pi/Arduinos.

1.1 Circuit Board Requirement and Specifications

The circuit board shall:

1. Be connected to 1 plug, but will have the flexibility to power up to 6 plugs.
2. Measure the power output of the appliance accurate to +/-0.1W.
3. Be connected to a button which turns the relay switch on or off.

The circuit board specifications:

1. The original circuit board will have 1 relay connected on it, but it will have ports which can connect to a breakout board consisting of up to 6 relays.
2. The current transformer will be calibrated and tested thoroughly with a multimeter to ensure that it takes the same current measurements as the lab multimeter.
3. A switch will be wired into the Arduino circuit, and can turn on/off the relay switch.

1.1.1 On/Off Switch description

We will be using a single-pole single-throw switch consisting of one input contact and one output contact. When ON, the switch will turn the relay ON, and when off, it will disconnect the relay from power, turning the relay (and thus the appliance) OFF. This switch measures 14.8mm x 20.8mm x 19.4 mm at its widest.

1.1.2 Relay description

A JQX-15F(787) miniature heavy-duty DC electromagnetic relay is controlled by 5V logic through a transistor. The voltage rating of the relay is 220VAC/28VDC. This relay is 3.2 x 2.7 x 2.5cm, is black in color, and weighs 25g.

1.1.3 Current Transformer description

We will use a 20A max split core current transformer. Output voltage will be 0~1V. Work frequency 50-1kHz. Accuracy +/- 1%.

1.1.4 AC-to-DC Power Module description

The appliance controller will be powered by an AC to DC converter. The converter will feed off the AC power that the appliance controller will plug into. It will output 5V at a 1A maximum.

1.2 Arduino Board Requirement and Specifications

The Arduino shall:

1. Send a confirmation signal to the Raspberry Pi immediately after it receives a command.
2. Send power usage of appliance to Raspberry Pi to be accessed through Alexa
3. Update the power draw information every second.

Arduino specifications:

1. The Arduino used in the final product shall be an Arduino Nano. For testing, an Arduino Uno will be used.
2. The Arduino code will include a transmitting function that is called immediately as the Arduino receives a command from the Raspberry Pi.
3. The Arduino will be collecting information from the current transformer hooked up to the appliance. The Arduino script will contain a module that sends the power information constantly to the Raspberry Pi, which will then store it and provide information to Alexa on command
4. The Arduino will send a power usage value via Bluetooth every second to the Raspberry Pi to store in its database.

1.2.1 RF Transceiver description

An HC-05 module is a Bluetooth SPP (Serial Port Protocol) module that is designed for transparent wireless serial connection setup.

1.2.2 Software Module description

1.2.2.1 Bluetooth

The Arduino script will include a loop that waits for Bluetooth signals to be received from the Central Processing Unit. It will use the SoftwareSerial library. Once received, it will send a confirmation signal back to the Raspberry Pi. When necessary, it will insert its data into a string to be sent via Bluetooth to the Raspberry Pi.

1.2.2.2 Current data management

The script will include a function such that it reads data from the current transformer on an ongoing basis and stores them in an array. Every 3 seconds, it will compute the average of all the values in the array and store that as a single number, ready to be sent to the central processing unit. The burden resistor for the current transformer can be found by:

$$\textit{Choose } 20A \textit{ as max current}$$
$$\textit{Primary } I_{peak} = 20\ A\ * \sqrt{2} = 28.284\ A$$
$$\textit{Secondary } I_{peak}\ =\ 28.284\ A\ \div 2000\ turns\ =\ 0.01414\ A$$
$$\textit{Ideal burden resistance}\ =\ 2.5\ \div 0.01414\ =\ 176.76\ \Omega$$
$$\rightarrow 2.5 \textit{ is half of the Arduino reference voltage.}$$

And the power information can be found by the calculations in the code:

```
float getPower() {
  float voltage;
  float current;
  float sum = 0;
  long Time = millis();
  int counter = 0;

  while(millis() - Time < 1000){
    voltage = ads.readADC_Differential_0_1() * multiplier;
    Serial.print(voltage);
    current = voltage * FACTOR; //Conductance
    current /= 1000.0;

    sum += sq(current);
    counter = counter + 1;
  }
  current = sqrt(sum/counter); //avg
  float currentRMS = current;
  float power = 120 * currentRMS;

  //return(power);
  return 10;
}
```

Figure 7. *Code to calculate the power*

1.2.2.3 Identity match

The Bluetooth signals sent from the Raspberry Pi will have a specific header to determine which appliance controller the data is meant for. The

Arduino script will include a function that checks the incoming signals and checks the header. If the header does not match its identity, it will ignore the data. If it matches, then the Arduino will store the information for decision making.

1.2.2.4 Decision making

If the Bluetooth module has received a signal and the identity matches, then this information will be passed onto the decision making module. This will be a loop in the Arduino script that either turns the on-board relay on or off.

2. Central Processing Unit Requirement and Specifications

2.1 Communication Requirement and Specifications

2.1.1 RF Transceiver description

2.1.1.1 Message Passing

The Python script will have a loop where it waits for signals to be received from one of its appliance controllers. When received, it will store this information as a string to be passed onto the decision-making module. When information needs to be sent to the appliance controllers, the information will be encapsulated with a header to indicate which controller the information is meant for. It will also have a termination tail at the end of the data to indicate the end of the message. Once this is done, the script will send the message through the Raspberry Pi's built-in Bluetooth module.

2.1.1.2 Pairing/Authentication

The CPU shall make its on-board Bluetooth transceiver available to other devices. It will scan for all available devices and look for those with the address of the HC-05 module that is connected into the Arduino. Once it finds these, it will connect and pair with them. All of this will be done by executing "Bluetoothctl" commands in terminal.

2.1.2 Alexa Wi-Fi Link description

2.1.2.1 Network Tunnel

The network tunnel shall be established by Ngrok. This will be downloaded onto the Raspberry Pi, and will be continuously running on the Pi using port 5000. It will have a unique https address that is used as an endpoint in the Alexa Skill. The Python script on the Raspberry Pi will use the library "Flask-Ask" to pull information from the Ngrok tunnel onto its' local server.

2.1.2.2 Executable Commands

The Python script will include functions for each 'intent' that is triggered by the smart speaker. Intents are actions that fulfill user's spoken requests and are programmed into the Alexa skill to correspond to any word/s that the designer chooses. When one of these intents is triggered, it is passed through the network tunnel to the Raspberry Pi. The Python script will have a function for each intent such that when any one of them is initialized, the CPU will act accordingly.

2.2 Data Management Requirement and Specifications

2.2.1 Raw Data Acquisition description

Data about power will be collected from each of the Arduinos from the relays. This data will be collected every second as an average of AC voltage. This will send data frequently in order to receive as accurate as possible an approximation of power draw to the appliance.

2.2.2 Temporary Local Storage description

The power information will be temporarily stored on the Raspberry Pi while it gathers more data and does calculations. Before storing data on the server the data will be simplified, and this process is done on the Raspberry Pi. As the data waits to be processed it will be stored on the Raspberry Pi in order to not have to continually pull from the server. The Pi does all of the simplification calculations.

2.2.3 Calculate and convert description

It is difficult to store data every second so it will be stored in compact values. These values will be hourly, daily, monthly, and yearly. With time, older values will average, such that the oldest values cannot be accessed in small increments but rather as averages.

2.3 Decision Making Requirement and Specifications

The Raspberry Pi can run on several different modes. These are python scripts that run and work with other processes and subsystems.

2.3.1 Manual command description

When the user says something to the smart speaker, this should override any process or mode that is currently happening. Vocal input will therefore be prioritized over modes or machine decisions.

2.3.2 Routine description

The Raspberry Pi can receive commands from the user to carry out individual actions for individual plugs. This is described in 7.2.3.1 above. However, sometimes the user may own multiple plugs connected to different appliances around the house. If the user wants to initialize a "mode" or "routine" such as "night mode" or "bedtime routine" he or she can make one voice command to the smart speaker and the Raspberry Pi will execute all the appropriate actions

according to the pre-defined routine, changing the state of multiple plugs with just one command.

2.3.3 Shutdown description

When the shutdown process is launched, the Raspberry Pi will turn off all of the data collection happening at Arduino nodes. It will also finish uploading all of the data that is temporarily stored to the server.

2.4 Boot-up Script Requirement and Specifications

When the appliance is plugged in or there is a power outage, the boot up script will need to be run. There are two portions of code in the Raspberry Pi. These are the main code, which is effectively constantly running, and the boot up code. The boot up code only runs when the Raspberry Pi gets power after not having it for some time.

2.4.1 Run Main description

The main script will run during normal operation. This is the script that will contain all of the linked functions with intents. It will also contain the code for forwarding information to the Arduino through Bluetooth, and for sending information back to the Alexa. This is the bulk of the Raspberry Pi code.

2.4.2 Run First Time description

The first time that the code runs, it must run differently from the main code. It must initialize several different systems within the Raspberry Pi system. These are included below.

2.4.2.1 Network Tunnel

The NGROK tunnel from the pi to the Alexa needs to be set up every time that the Raspberry Pi is turned on because it does not remain set up while everything is off.

2.4.2.2 Bluetooth

The Bluetooth connection between the Arduino and the pi also needs to be set up every time that a component is turned off to ensure that messages can be passed.

2.5 AC-to-DC Power Module Requirement and Specifications

The Raspberry Pi will be powered by an AC-DC converter that plugs into a wall socket and outputs 5V at 2.5A max rating onto a micro usb cable that will be plugged into the board.

3. Smart Speaker Requirement and Specifications

The Smart Speaker Shall:

1. Take in vocal cues from the user and respond to these accordingly.
2. Send on messages to other appliances in order to have the rest of the sequence respond to the input.

3. Be able to give the user information on the system as a whole and the power usage of given plugs.

The Smart Speaker Specifications:

1. The vocal cues will be a list of phrases that the user can say which can be asking for a switch to be turned on, asking for a switch to be turned off, asking for a specific mode or setting, or asking for power information.
2. The smart speaker will pass a message through Wi-Fi to the Raspberry Pi about whether to turn the switch on or off, or about what information is needed.
3. The smart speaker will receive information from Raspberry Pi which pulls from the server about power usage and data specifications collected from each plug.

3.1 Custom Alexa Skill Requirement and Specifications

The Custom Alexa Skill Shall:

1. The custom Alexa skill shall have intents for each possible sentence request.
2. The custom Alexa skill shall have intents that are linked with Raspberry Pi code.
3. The custom Alexa skill shall cover extensive phrases and be heavily tested for phrases that are missed.

The Custom Alexa Skill Specifications:

1. The intents will cover the possibility of turning a switch on, turning it off, switching modes, and asking for power data.
2. The intents are linked with functions in python code on the Raspberry Pi which execute when the intents are signaled.
3. The intents will be based on at least 5 different variations of each command that a user could say for each possible ask option.

3.1.1 Intents description

An intent is a portion of an Alexa skill that defines what a user could mean. It is possible to have several different user vocal cues that would signify one intent. An example would be “yes” and “yeah”. There will be 4 major intents, which are listed above, and a variety of minor ones. We will run through a variety of possible sentences and include all of the potential “utterances”.

3.1.2 Sample Utterances description

An “utterance” is something that the user says. There are important because it is common for users to say several different things that all desire the same response. It is important to understand variation in how people will express what they want. For this reason, we will put in several options for each intent.

3.1.3 Developer Console Config description

We will be linking different intents with the correct python functions on the Raspberry Pi. The team will also be linking all of the utterances that apply with the correct intents. Finally we will be linking the Raspberry Pi to Ngrok which will be the message passing channel between the Raspberry Pi and the Alexa.

Module Design

1. Appliance Controller

1.1 Circuit Board

The circuit board will consist of the following components on one printed circuit board (PCB). The PCB will have power rails of 5V and 0V, provided by the AC-DC power module and will provide power to all of the following modules on the board. It will also connect other digital outputs, analog inputs, and tx & rx pins to and from the Bluetooth module. An alternative was to forego the PCB and loosely wire each of the modules together. This was not done because the appliance controller is expected to be moved around a lot as it is plugged and unplugged. Therefore, a PCB is used in order to have all of the modules mounted onto the same board.

1.1.1 On/Off Switch

The on/off switch will be connected as a digital input to the Arduino board. When pressed, it will notify the board with a positive voltage signal, indicating that the relay shall be turned on.

1.1.2 Relay

The Arduino will send +5V across the coils of the relay switch. When this happens it will conduct its “Normally open” contacts, thus forming continuity between the “Common” pin and the “Normally open” pin. One of these pins will be connected to the phase of the power coming from the wall socket, and the other pin will be connected to the phase of the appliance controller’s output socket (where the appliance will be plugged in). When continuity occurs, the output socket of the appliance controller will have available power for an appliance to plug into. The relay coil rating will be just big enough to handle every typical household appliance. Anything bigger would take up unnecessary space in the appliance controller.

1.1.3 Current Transformer

One wire will be connected to ground, while the other wire will be connected as an analog input to the Arduino board. The output of the relay will be looped through this before it is soldered onto the contact of the output socket. The current draw from the appliance will be proportional to the voltage induced in the current transformer. This will be sent to the Arduino board. There were no real alternatives on the hardware side here.

1.1.4 AC-to-DC Power Module

The AC-to-DC power module will have an input of 120V from the wall socket power and output 5V with the use of a rectifier circuit. Other alternatives were to use a battery, however this is impractical considering the switch must be

connected to a wall socket in order to power the appliance, so power would always be available to the circuitry within the appliance controller.

1.2 Arduino Board

The Arduino board will connect to the current transformer, on/off switch, and relay switch on the circuit board. It will receive information through the use of Bluetooth and digital/analog inputs regarding turning the relay on and off and making calculations on the current information. The Arduino Nano was chosen because of its small size and simplicity. One alternative was the ESP32, however, this was too large for the appliance controller's size requirements.

1.2.1 RF Transceiver

HC-05 module will be powered up to +5V and its Tx and Rx pins connected to the Rx and Tx ports of the Arduino board. The transceiver will send and receive Bluetooth signals through these pins. The received information will be passed through to the Arduino board for processing and translation. The HC-05 was chosen because of its reliability when working with Arduino boards.

1.2.2 Software Module

An Arduino script will run continuously that governs the following modules. Whenever the switch is plugged in to a wall socket this script will boot up automatically and run itself.

1.2.2.1 Bluetooth

The module will use Serial Read and Serial Write commands in the SoftwareSerial library to send and receive information to and from the Arduino's Tx and Rx pins. Received information will be stored in an array for processing by the other modules. Information that is sent will be in string form.

1.2.2.2 Current Data Management

The design for this module is to store 3 seconds worth of information, compute an average, send it through to the Bluetooth module, and then erase it. This keeps the Arduino's memory free and introduces slightly more computing, which it can handle. One alternative for this design was to calculate and store current data on the Arduino. However, the Arduino nano has less than 1 kilobyte of EEPROM memory, making this impossible.

1.2.2.3 Identity Match

The string of information received by the Arduino will be passed to this module. Through a series of if statements, the first few characters of the string will be compared with the appliance controller's specific identity number. If they match, a 1 will be returned from this module. If not, a 0

will be returned. This method was chosen due to the minimal processing required to get the result.

1.2.2.4 Decision Making

Will traverse the string of information to see what relay is being requested and whether or not it needs to be turned on or off. It will send a digitalWrite signal accordingly, either turning the relay on or off.

2 Central Processing Unit

2.1 Communication

The communication will function by passing information from the Alexa to the Raspberry Pi to the Arduino.

2.1.1 RF Transceiver

The transceiver is mounted on the Raspberry Pi's circuit board, so there was no alternative necessary.

2.1.1.1 Message Passing

Socket commands were chosen to be used in this case because they provide for the simplest sending and receiving of data. The commands "sock.recv" and "sock.send" are called to send or receive data in the form of strings.

2.1.1.2 Pairing/Authentication

"Bluetoothctl" is the main command for configuring Bluetooth devices in Linux, so this was used to initiate pairing and authentication. To pair a device this command is called in Terminal where the appliance controller's Bluetooth module can then be paired by using its' Bluetooth address and a pin code.

2.1.2 Alexa Wi-Fi Link

The Alexa will be connected to the Raspberry Pi through a Wi-Fi link. This is further explained below, but Wi-Fi was chosen so that the Alexa could be far away from the Raspberry Pi and still function.

2.1.2.1 Network Tunnel

This will function through a network tunnel program called NGROK. NGROK will be used to pass messages between the Raspberry Pi and the Alexa. This is instead of other network tunnel softwares which would have been more complicated to implement.

2.1.2.2 Executable Commands

The Alexa will trigger an intent which will execute code on the Raspberry Pi. This is default.

2.2 Data Management

The Raspberry Pi will be saving data to a server in order to report back power usage information to the user. This is important because it can be used for power saving.

2.2.1 Raw Data Acquisition

This data will be acquired at the Arduino level. This will be done through previously described current transformers and sent back through Bluetooth to the Raspberry Pi. The purpose of gathering data is to report it back to the user.

2.2.2 Cloud storage

Rather than storing data locally and consuming storage space, power usage values will be averaged and sent to a cloud database stored in Amazon Web Service's Relational Database Service (RDS) every minute.

2.2.3 Calculate and Convert

Calculating and converting data functions by taking averages of the power over certain periods of time. Rather than storing every single datapoint, users will be able to ask for averages of power usage over the last minute, 30 minutes, 60 minutes, and 24 hours; all these calculations take place within the database itself.

2.3 Decision Making

The Raspberry Pi will make decisions based on predefined settings and modes. This is in order for power saving to be achieved when modes are chosen. This also allows for a more understandable way of running code.

2.3.1 On/off Command

Manual command is when the user requests something specific from the smart speaker, such as "turn on the light" or "turn off the TV". This should happen no matter what setting or mode is currently on, and overrides anything else. This is chosen because a user should have permission to change settings whenever they would like.

2.3.2 Data Retrieval

When it is not sending an on or off command, the Raspberry Pi will be collecting all the power information that arrives at its port from the Arduino within the smart switch over a Bluetooth connection.

2.4 Boot-up Script

The intention for the boot-up script is to set up Bluetooth and Wi-Fi connections and the system as a whole so that it can run effectively. There are some necessary initialization features that need to run, and this should run automatically without user prompting. A boot up script is used as opposed to a method that requires user prompting because the user should have to interact in complicated ways as little as possible.

2.4.1 Run Main

This code will run constantly. The choice to have a main function that loops constantly was made out of convenience. Systems that constantly check a Wi-Fi tunnel are usually built like this.

2.4.2 Run First Time

This is the code that will happen the first time a system runs. It should run no matter how the system last shutdown. This code, as mentioned above, is needed to run some initializations.

2.4.2.1 Network Tunneling

The chosen tunnel will be an NGROK tunnel. We chose NGROK because some members had experience with it and also because of how easy it was to set up. It was also easy to find resources for it. This tunnel is meant to pass information from the Alexa to the Raspberry Pi. This is initialized in the first run time code.

2.4.2.2 Bluetooth

A Bluetooth connection was chosen between the Raspberry Pi and the Arduino. This is meant to pass information back and forth relating to turning something on or off, or power information. A Bluetooth connection was chosen because this is simple enough and not dependent on Wi-Fi.

2.5 AC-to-DC Power Module

The power module will be purchased from a store and will plug into the wall, converting a 120V AC input to 5V DC output. This will plug into the Raspberry Pi's power port. Its' rating of 2.5A was chosen to handle the Raspberry Pi's power draw, along with a screen if necessary for user interface or testing.

3 Smart Speaker

3.1 Custom Alexa Skill

The Alexa Skill is a software base provided by Amazon intended to be used with the Alexa program in order to produce desired responses from "Alexa". Although there are other smart speaker alternatives, such as the Google Home, the Alexa is the most widely recognized and the most programmable. Amazon's Alexa provides the skill software which allows easier access for developers.

3.1.1 Intents

Intents work through the Alexa Skill software by effectively being a segment of code that can be called by several different "utterances" or vocal cues. Due to our decision to use the Alexa as our smart home base, the use of intents only naturally followed.

3.1.2 Sample Utterances

Sample utterances are a way of defining vocal cues. Amazon gives an easy way to define specifically what these utterances, or vocal cues, can and should be. Again, there was no real other option to using these, given the knowledge that Alexa was the chosen smart speaker.

3.1.3 Developer Console Config

The developer console is the provided system that allows for the developer to connect utterances, intents, and code elsewhere (in this instance, on our Raspberry Pi). This is an easy to use interface that helps users link intents and utterances so that the Alexa knows that a vocal cue means for something specific to happen. Again, this is an Amazon product and therefore a natural progression of our choice.

Design Changes Since FDR

One change the team made to the appliance controller was the addition of an analog-to-digital converter to get a reliable stream of digital data from the current transformer to the Arduino. This made the data easy to process because there was a library attached with the converter module. The team also introduced a second appliance controller which is capable of controlling two appliances. The controller has one microcontroller and bluetooth module, like the initial appliance controller; however it has an extra relay, current transformer, and A-D converter.

Another design change that was made was the removal of local storage of user data on the SD card. Local storage is less secure and also more prone to loss of data, in addition to being more limited in its nature; it is constrained by the size capacity of the SD Card. Instead of using local storage, all data is being stored in an array, averaged over a per-minute span, and sent directly to the AWS database, where the data can again be averaged and sent to the user. This design is not only simpler but also more secure.

Module and System Tests

Ensuring each and every module performs as expected is vital to the success of the project. Any defect in a module will affect the module above it, and in turn affect the overall performance of the system. Thus, the testing description has been broken down in the same way that the subsystem functional diagram was to ensure that every module has the appropriate tests done on it.

1 Appliance Controller

1.1 Circuit Board

Test 1: An appliance which is rated to draw 15A of current will be connected to outlet power through a multimeter. The voltage and current through this appliance will be recorded and it will be disconnected. Then, one port of the relay will be connected to power from an outlet, while the other side will be connected to the appliance through a multimeter. The relay will be turned on by a 5V DC voltage and then the performance will be observed by the multimeter. A **PASS** will occur if the multimeter displays the same voltage and current +/ 10% of the original values. This will be left on for a duration

of 1 hour and the values will be checked every 5 minutes to see any change in output values. A **FAIL** will occur if the output is anything outside the 10% limit.

Test 2: The relay will eventually be enclosed in the switch's 3D printed case, so its temperature must also be tested. At every 5 minute observation, a thermometer will also be held onto the relay to test its temperature. It will **PASS** if the temperature does not exceed 85 degrees celsius, and it will **FAIL** if it does exceed this temperature

1.2 Arduino Board

Test 1: A series of 20 arrays will be sent from the Central Processing Unit to the Arduino board with a spacing of 5 seconds. 15 arrays will contain a header that is destined for this specific appliance controller. The other 5 will be destined for some other controller. Each of the 15 arrays will contain a command to either turn the switch on or off. This will be completely random. The module will **PASS** if it turns the relay on/off when an "on"/"off" command was sent from the Central Processing unit that was destined for the switch. It will **FAIL** if it turns the relay on when it was meant to be turned off, or vice versa, or if it turns on/off in response to an array of information that was not destined for it.

Test 2: Five appliances with a variety of power ratings will be connected through the current transformer and a multimeter. These will be turned on for a duration of 5 minutes each. A test code will be written on the Central Processing Unit to print the data it receives from the Appliance Controller. This module will **PASS** if the Central Processing Unit prints values every 3 seconds that are within 10% of the value displayed on the multimeter. It will **FAIL** if the readings fail to print, or if they vary by more than 10% from the observed value on the multimeter.

2 Central Processing Unit

2.1 Communication

Test 1: This test will verify that the Raspberry Pi is receiving information from the Arduino. A test code will be written on the Arduino that sends 20 arrays of information to the Central Processing Unit. These arrays will be sent every 3 seconds. Thus, 400 arrays will be sent in one minute. There will be test code on the Central Processing Unit that prints any information that it receives from the Arduino. The system will **PASS** if all 20 arrays are printed correctly. It will **FAIL** if it does not print all 20 arrays, or if the information differs to what was originally sent.

Test 2: This test will verify that the Raspberry Pi is sending information to the Arduino. Test code on the Central Processing Unit will pass 20 arrays of information destined to be sent to the Appliance Controller. A test on the Arduino will print whatever information it receives. The module will **PASS** if the 20 arrays are printed on the Arduino. This will

demonstrate that the information passed to the communication module was correctly packaged with a header destined for the Appliance Switch, and sent across the built-in Bluetooth module. It will **FAIL** if not all arrays were printed or if the information differs from what was originally sent.

Test 3: This test will verify that the Ngrok tunnel from the Alexa server to the local Raspberry Pi is set up correctly. The Central Processing Unit will boot up a total of 10 times and execute the command "Ngrok http 5000 -subdomain=sd-amm" in terminal. It will **PASS** if the tunnel successfully sets up, and **FAIL** if there is an error and the tunnel does not set up any of the 10 times the CPU reboots.

Test 4: This test will verify that the correct intent is being executed when the user submits a request on the smart speaker. Test code will be inserted into the Python script so that it prints whenever an intent was triggered by the smart speaker. Each intent will be spoken into the smart speaker, and the terminal on the Central Processing Unit will be observed to see whether or not the intent was triggered on it. The module will **PASS** if each spoken intent was printed on the terminal, and it will **FAIL** if spoken intents were not printed.

2.2 Data Management

Test 1: A test code will be written to test every possible type of data storage and print these results. This will include the results for an average of the past day, week, month, year, etc. Because it will be impossible to have these results, we will send pre-written values to the CPU every 0.3 seconds (10x faster than usual) for 3 days to simulate 30 days = 1 month. Then the team will take the CSV file on which the actual data is listed, and hand calculate the results without the code. A **PASS** would happen if the data is identical on our calculations versus our test, whereas a **FAIL** would mean the data was not the same.

2.3 Decision Making

Test 1: In order to test that the different modes run correctly, we will be testing them separately, and then together to ensure that they all run at the correct times. We will find every possible sequence of modes, including making manual commands before, during, or after routine commands. We will acknowledge all cases and all error checking cases. This test will **PASS** if we test every sequence possible of modes and find that the system successfully processes them. The test will **FAIL** if it misses an edge case, and we will test with unexpected variables to make sure of this.

2.4 Boot-up Script

Test 1: In order to test the boot up script, we will try to make the system fail in all types of ways. We will have it turn off by unplugging it (simulating a power outage), by manually turning it off, and by unplugging relay nodes. The system will **PASS** if when the boot up script runs after these incidents, it runs successfully and all data management

still works. The system will **FAIL** if the system does not boot up correctly after these incidents.

2.5 AC-to-DC Power Module

Test 1: The Power adapter will be plugged into a wall outlet and the output connected to a multimeter. It will **PASS** if the output is 5V, 2.5A +/- 10% and **FAIL** if the output lies outside the 10% limit.

3 Smart Speaker

3.1 Custom Alexa Skill

Test 1: In order to test that all sample utterances successfully link to the correct python functions, we will run through every single sample utterance that we have come up with, and make sure that it links to the correct python code. The test will **PASS** if every utterance correctly links to first the intent and then the python code (and thus executes correctly), and **FAIL** if any one of the utterances does not correctly link, or runs the wrong code.

Applicable Standards

Table 5. *Applicable Standards*

Standard	**Definition of standard**	**How this is applicable**
OSHA 1910.305	Wiring methods, components, and equipment for general use	There is wiring in the node.
OSHA 1910.305(a)(2)	Temporary wiring	Any testing that we do will apply here.
OSHA 1910.305(a)(2)(i)	Temporary electrical power and lighting installations of 600 volts, nominal, or less may be used under certain circumstances	Again, this is relevant for testing.
OSHA 1910.305(a)(2)(i)(C)	For experimental or development work, and during emergencies.	This might also be for testing.
OSHA 1910.305(c)	Switches	We will probably have a physical switch on the Arduino.

OSHA 1910.305(j)(2)	Receptacles, cord connectors, and attachment plugs (caps).	There is wiring in the node.
OSHA 1910.305(j)(2)(i)	All 15- and 20-ampere attachment plugs and connectors shall be constructed so that there are no exposed current-carrying parts except the prongs, blades, or pins. The cover for wire terminations shall be a part that is essential for the operation of an attachment plug or connector (dead-front construction). Attachment plugs shall be installed so that their prongs, blades, or pins are not energized unless inserted into an energized receptacle. No receptacles may be installed so as to require an energized attachment plug as its source of supply.	We will be working with current and measuring it.
OSHA 1910.305(j)(3)	Appliances	
OSHA 1910.305(j)(3)(iii)	Each electric appliance shall be provided with a nameplate giving the identifying name and the rating in volts and amperes, or in volts and watts. If the appliance is to be used on a specific frequency or frequencies, it shall be so marked. Where motor overload protection external to the appliance is required, the appliance shall be so marked.	We will be using electrical appliances that may have this requirement.
OSHA 1910.305(j)(3)(iv)	Marking shall be located so as to be visible or easily accessible after installation.	See above.
OSHA 1910.305(g)	Flexible cords and cables	

OSHA 1910.305(g)(1)	Use of flexible cords and cables.	We may have these.
OSHA 1910.305(g)(2)	Identification, splices, and terminations.	When wiring the circuit this may occur as a consideration.
OSHA 1910.305(g)(2)(ii)	Flexible cords may be used only in continuous lengths without splice or tap. Hard-service cord and junior hard-service cord No. 14 and larger may be repaired if spliced so that the splice retains the insulation, outer sheath properties, and usage characteristics of the cord being spliced.	Again, applicable for the node.
OSHA 1910.305(g)(2)(iii)	Flexible cords and cables shall be connected to devices and fittings so that strain relief is provided that will prevent pull from being directly transmitted to joints or terminal screws.	Again, applicable for the node.
Bluetooth SIG . Bluetooth Core Specification: 5.0. Bluetooth Special Interest Group; Kirkland, WA, USA: 2016.	Bluetooth Core Specification: 5.0.	We will be using Bluetooth to connect the nodes.
IEEE 802.15.4n-2016	IEEE Standard for Low Rate Wireless Networks	We will be using WiFi and Bluetooth.
802.11 - WLAN technology		Echo connects to dual-band Wi-Fi (2.4 GHz / 5 GHz) networks that use the 802.11a / b / g / n standard. Echo does not connect to ad-hoc (or peer-to-peer) networks.

RFC2616 - HTTP standard **9.9 CONNECT**		The information tunnel from Alexa to Raspberry Pi requires this standard. This specification reserves the method name CONNECT for use with a proxy that can dynamically switch to being a tunnel. https://tools.ietf.org/html/rfc2616#section-9.9
IEEE C57.13-2008	IEEE Standard Requirements for Instrument Transformers	We will have a current transformer in our Arduino switch node.
IEEE C37.90-2005	IEEE Standard for Relays and Relay Systems Associated with Electric Power Apparatus	The relay which will be connected to the Arduino should use this standard.

Summary and Conclusions

The final product demonstrates a successful connection loop, beginning with a voice command from the user and culminating in a response back to the user from Alexa. At the end of last semester, the user was able to do a basic on/off command to one switch or one plug. Now, the user is able to control multiple smart plugs, not just one plug. We demonstrate that multiple plugs can be placed in a single smartplug unit, and multiple smartplug units can be placed around the house, and connections can be made in both scenarios. The Raspberry Pi has been configured to be able to distinguish from information arriving over Bluetooth from multiple plugs simultaneously, and storing the information in the appropriate database tables.

Beyond on/off information, the functionality to store and convey the power usage is now fully implemented. If the user verbally asks Alexa for the power usage of his connected appliance, Alexa can now connect to the Raspberry Pi central router, and the router can connect to the Arduino in the smartplug via Bluetooth. Next, the router collects the power usage information, and this information can be stored in a database, converted to an average, and relayed back to the user as an average. The user has the flexibility to ask for usage over the past minute, 30 minutes, 60 minutes, or 24 hours, and added functionality can be implemented to expand the kind of information the user can ask for. Because testing the ability to extract power usage over a long period of time (e.g. multiple days to a week) was not feasible, such functionality was not included.

A strong, secure, and simple storage system for user data was introduced, using the Amazon Web Services Relational Database Service. This allows for long-term storage of data if desired, and it maintains backups of data, which is another added advantage of utilizing a large-scale cloud storage service rather than our own local server.

There were no major problems or hurdles in the design process. The only minor hurdles were coding bugs and physical construction challenges in designing a two-plug smart-switch. All in all, this smart plug is close to being ready for use in a home and has the added advantage of requiring little setup. Bluetooth pairing, for as complicated as it can be, is done automatically and requires no added Energy monitoring is as important as ever as environmental concerns begin to surround our daily lives, and as the cost of living continues to rise, cutting back on power usage begins with knowing the power usage for appliances around the home.

References

Biagi, L. (n.d.). The Statistics Portal. Retrieved from https://www.statista.com/page/dec

Bluetooth Communication between Raspberry Pi and Arduino. (n.d.). Retrieved from http://www.uugear.com/portfolio/Bluetooth-communication-between-raspberry-pi-and-Arduino/

How to build an Arduino energy monitor - measuring mains voltage and current. (n.d.). Retrieved from https://openenergymonitor.org/forum-archive/node/58.html

Instructables. (2017, October 14). Raspberry Pi Auto-boot. Retrieved from http://www.instructables.com/id/Raspberry-PI-auto-boot/

Manz, B. (n.d.). Bluetooth 5: Mesh Networking, Greater Range, and Ability to Coexist Offer Potential for IoT and IIoT Applications. Retrieved from https://www.mouser.com/applications/Bluetooth-5-mesh-networking-standard/

Public URLs for exposing your local web server. (n.d.). Retrieved from https://Ngrok.com/

Amazon Moves Back to Top Spot for Q3 2018 Smart Speaker Sales – Canalys. (2018, November 15). Retrieved from https://voicebot.ai/2018/11/15/amazon-moves-back-to-top-spot-for-q3-2018-smart-speaker-sales-canalys/

Appendix 1 - Timeline Estimation and Milestones

ID	Task Mode	Task Name	Duration	Start	Finish	Predecessors
8		Alexa-to-Rpi: Ngrok Network	6 days	Mon 9/17/18	Mon 9/24/18	
31		Sprint 1		Thu 9/27/18		
1		Documentation - CDR Part I	5 days	Sun 10/14/18	Thu 10/18/18	
14		Alexa: Configure Alexa Developer Console	5 days	Mon 10/15/18	Fri 10/19/18	
32		Sprint 2 / CDR Part 1		Thu 10/18/18		
3		Documentation - CDR Part II	3 days	Fri 10/19/18	Tue 10/23/18	1
15		Rpi: Specify intents to link	5 days	Mon 10/22/18	Fri 10/26/18	
19		Rpi: Code intents in python script for indiv. user actions	5 days	Mon 11/5/18	Fri 11/9/18	
33		Sprint 3 / CDR Part 2		Thu 11/8/18		
4		Arduino-to-RPi: Feedback function	7 days	Sun 11/11/18	Sat 11/17/18	
11		Rpi: Write Bluetooth connection script	5 days	Mon 11/12/18	Fri 11/16/18	
18		Alexa-to-Rpi: Test intent invocation	2 days	Mon 11/12/18	Tue 11/13/18	19
7		Documentation - FDR	6 days	Mon 12/3/18	Mon 12/10/18	3,1
2		Arduino-to-RPI: Send power info	5 days	Mon 12/10/18	Fri 12/14/18	
34		Sprint 4 / Final Design Review		Mon 12/10/18		
5		Design a printed circuit board in PCBArtist	10 days	Mon 12/24/18	Fri 1/4/19	
9		Alexa: Compose intents and sample utterances	3 days	Mon 12/24/18	Wed 12/26/18	
6		Process PCB schematic with 4PCB.com	10 days	Mon 1/7/19	Fri 1/18/19	5
12		Rpi: Test bluetooth	3 days	Mon 1/7/19	Wed 1/9/19	
22		Test and configure current	3 days	Mon 1/21/19	Wed 1/23/19	
10		Arduino: Write power usage calculation script	5 days	Thu 1/24/19	Wed 1/30/19	22
13		Rpi: Auto-startup script	5 days	Mon 1/28/19	Fri 2/1/19	
27		Rpi: Shutdown script	5 days	Mon 1/28/19	Fri 2/1/19	
16		Test multi-relay system	3 days	Mon 2/4/19	Wed 2/6/19	
28		Rpi: Specify routines	5 days	Mon 2/4/19	Fri 2/8/19	
30		Rpi: Test emergency shutdown	3 days	Mon 2/4/19	Wed 2/6/19	
17		Replace with Arduino Nano	5 days	Mon 2/18/19	Fri 2/22/19	
20		Test measured current against collected current data	5 days	Mon 2/18/19	Fri 2/22/19	
24		Design Arduino case using Autodesk Inventer	5 days	Mon 2/18/19	Fri 2/22/19	
21		Arduino-to-Rpi: test feedback and data transfer	5 days	Mon 2/25/19	Fri 3/1/19	
29		Rpi: test routines	5 days	Mon 2/25/1	Fri 3/1/19	
35		Product Test Review		Thu 2/28/19		
23		Design Rpi case using Autodesk Inventer	5 days	Mon 3/4/19	Fri 3/8/19	
25		3D print Rpi and Arduino cases	5 days	Mon 3/11/19	Fri 3/15/19	23,24
26		Test case against specifications	2 days	Mon 3/18/19	Tue 3/19/19	25
36		Final Product Review		Thu 4/11/19		
37		Testing vs. Specifications		Thu 4/25/19		

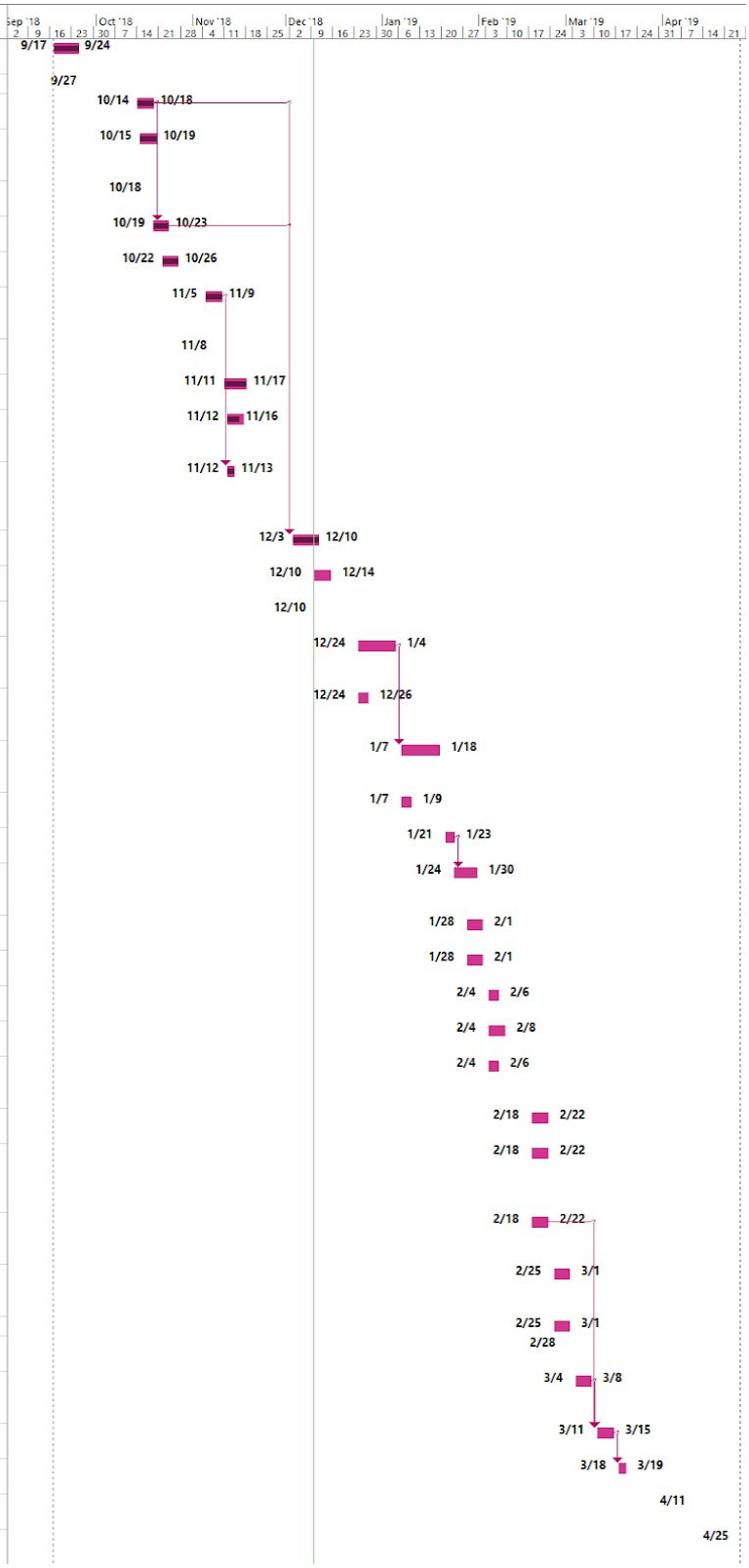

Figure 8. *Timeline estimation*

Appendix 2 - Economic Analysis

In order to estimate the cost for this prototype, the students had to make a number of assumptions. The first assumptions were regarding labor hours. With this project being reliant on software and testing, they assumed that there would be 2 software engineers working a total of 72 combined hours, and 3 test engineers working a total combined 48 hours. They predicted 1 person working as project manager (48 hours), design engineer (24 hours), hardware engineer (24 hours), and technical writer (24 hours). This led to a total cost to contract of $27, 540.

Assuming that parts will be bought in bulk for manufacturing, the students predicted that the final pre-manufacturing prototype would cost $50 for parts and $25 for PCB fabrication, leading to a total price of $78.75 after a pass through fee. This led to a total prototype cost of $27,619.

The students predicted 20 hours for manufacturing at a $45 hourly rate, along with 30 hours of software testing at a $40 hourly rate, leading to a production labor cost of $4,200.

The assumption of selling 2000 parts with a predicted $3 printing & packaging cost per unit led to a total production cost of $195,319. Adding a 40% overhead and 20% profit margin, the total cost came out to be $312,510, which equates to a $156.26 total production cost per unit.

The total cost of the project was predicted as $340,129, leading to a $170.06 total project cost per unit. Adding a wholesale multiplier brought the wholesale price to $204.08 per unit, and a retail multiplier pulled the price up to $306.12 per unit.

Based on the cost estimation, the students would prefer to sell the product as a manufactured good; however, in order to keep the retail price down they would need to sell over 2000 units. If they are predicting to sell this many units, however, then one may argue that more effort must be put into testing and manufacturing to ensure that this is a sustainable product, which may increase the price of the product.

Appendix 3 - Parts List

Table 6. *Bill of Materials*

Subsystem	Part Name	Description	Part #	Cost (+ Tax)	Manufacturer	Source
Smart Speaker	Amazon Echo Dot	Smart Speaker	B01DFKC2SO	$49.99	Amazon	Amazon Echo

Central Processing Unit	Raspberry Pi 3 (B+)	Mini Computer powered by 5V	BCM132 LKD4	$39.95 + $6.15 shipping	Raspberry Pi	Sparkfun - RPi
Central Processing Unit	8gb SD Card	Memory Card used as the hard drive for Raspberry Pi	FBA-RA SPBERR Y-PI-SD	$12.79	CanaKit	Amazon - SD Card
Appliance Controller	Arduino Uno	Single board microcontro ller used in appliance controller	DBAD10 0200	$8.70	Arduino	Amazon - Micro
Appliance Controller	Relay switch module	220VAC, 20A (NO) relay switch and circuit board	JQX-15F/ 005-1Z1	$7.95	Sparkfun	Sparkfun - Relay
Central Processing Unit	120V-5V Converter	Wall power supply with micro usb output	28-1933- 8	$7.95	Pro-Elec	Sparkfun - 5V
Appliance Controller	HC-05 Bluetooth Module (for Arduino)	Wireless Bluetooth serial RF transceiver	B01CKW 4FSI	$7.39	HiLetgo	Amazon - Bluetooth
Appliance Controller	Push-butto n Switch	On/off switch that will close when pushed,	N/A	$0.50	Sparkfun	Sparkfun - Switch (Also available from ECE

		open when resting.				department)
Central Processing Unit	7" Touch Screen Display	Touch screen display which interfaces with Raspberry Pi	49Y1712	$60.00	Raspberry Pi	Element 14
Central Processing Unit	RPi Keyboard and Mouse	Inputs for the Raspberry Pi for keyboard and mouse	B071FZNC17	$13.49	Riitek	Amazon
Central Processing Unit	Adafruit Raspberry Pi B+ / Pi 2 / Pi 3 Case - Smoke Base - w/ Clear Top	Raspberry Pi case to keep the system constrained and protected	N/A	$17.51	Adafruit	Adafruit
Subsystem	***Software***	***Description***	***Version***	***Cost (+ Tax)***	***Author***	***Source***
Central Processing Unit	Ngrok basic plan	Tunnel between Alexa skill, Raspberry Pi script to give commands	2.2.8	$60	Ngrok	Ngrok

Appendix 4 - Qualifications of Key Personnel

Matt Taylor's qualifications for this project come from his school work, employment, and personal experience. His classes include ECE 3410 (Communications Engineering), ECE 3520 (Microprocessors), ECE 2115 (Engineering Electronics), ECE 2210 (Circuits, Systems, Signals), and ECE 2210 (Circuit Theory). He was previously employed as an electronic assembler by BEP Marine, a manufacturing firm for sports boat electrical systems. Taylor has experience in soldering, fabricating PCBs, wiring battery management panels, and calculating wire sizes and relay ratings based on current draw requirements. In high school, he worked on numerous projects that utilize the Arduino Uno - specifically a quadruped, remote control boat, and an electronic drink fountain.

Agam Mittal's qualifications for this project come from his school work and programming experience he derived from prior internships. He was previously employed by Merck & Co. as a data scientist, working to build a customer service pipeline in Amazon Web Services. He is comfortable working with Java, C, and Python after past internships with the National Library of Medicine at NIH and prior to that at the Naval Research Lab. As an electrical engineering major, Mittal has taken classes such as ECE 3520 (Microprocessors), ECE 3130 (Digital Electronics Design) and ECE 3125 (Analog Electronics Design). In high school, he worked extensively with Bluetooth and interfacing with environmental sensors for a senior design project involving an autonomous rover.

Marie-Laure Brossay's qualifications for this project come primarily from programming experience. She is a computer science minor and has taken classes such as CSCI 3411 (Operating Systems), CSCI 2311 (Software Engineering), ECE 3515 (Computer Organization), ECE 3525 (Embedded Systems) and various other programming classes. Brossay has experience programming in Java, Python, and C. She has done a lot of operating system and systems work, as well as work in embedded systems.

Appendix 5 - Intellectual Contributions

Agam Mittal was focused on working with user data, which in this project refers to the power usage and time information that is received by the central router by each plug that is measuring the power usage. He decided to use a cloud database on Amazon Web Services Relational Database Service (RDS) as opposed to a local database, and he created a simple MySQL database. He then resolved connectivity issues between the Raspberry Pi local machine and the remote database, and organized the database to work with multiple switches and multiple plugs. Next, he wrote code to extract data from the database upon a user request and computer averages of user data over certain periods of time. Agam also helped Matt with code to compute power usage, and assisted Marie with debugging and testing of all the code. On the project

management side, Agam made the Gantt chart as well as the subsystem and system functional diagrams.

Marie Brossay was focused on optimizing and programming the central Raspberry Pi node. She was in charge of making sure that the Raspberry Pi interfaced with all three other core parts of the project, meaning she focused on interfacing with the Alexa, the database, and the Arduino/plug units. Most of her contribution was the code base which the Raspberry Pi is constantly running, which is written in the Python language. She also worked with Amazon's Developer service to create the intents and the options for the Alexa Skill. Marie's main role was to ensure that the code from the other two members worked seamlessly through the entire design path, as well as debugging the problems that came from linking separate work together. She was also in charge of ensuring that the boot up of the raspberry pi is instantaneous.

Matt Taylor was focused on designing and implementing the hardware and software for the appliance controller, along with the programming required on the central Raspberry pi's bluetooth port. His work consisted of rapid prototyping which required building prototype boards, 3d printing materials, and writing and debugging code. Matt's goal was to get the appliance controllers working autonomously and worry-free with the rest of the system. His job required thorough testing of each piece of hardware to ensure that it was safe for the user.

Appendix 6 - Teaming Arrangements

The success of this project depends on both the functionality of the hardware and the comprehensive communication between them, so the members decided that only putting one person in charge of each piece of hardware was insufficient. Instead, they assigned a member to oversee two of the three modules, along with having a larger focus on one of the sections. Because the Central Processing Unit module is the largest and most all encompassing of the module, this one is worked on by all team members. All members should be able to help with all modules, but will have focuses and modules that they are primarily in charge of.

- Matt Taylor will mainly focus on the appliance controller module. He will also be jointly responsible for the central processing unit module along with the other team members. He will focus on the plug hardware, including the relay control of the appliance, current measurement and calculation of the power flow to the appliance, and Bluetooth capabilities of the Arduino.
- Agam Mittal will focus on the central processing unit module. He will also be helping Matt with the appliance controller module. Because he will be working on mostly the largest module, he will have the module which receives input from both other team members as well. He will be focusing on relaying power data back from the appliance to the Raspberry Pi, managing stored data on the Raspberry Pi, and developing the Alexa Skill with Marie. He will also help Matt with some of the hardware components.
- Marie-Laure Brossay will focus on the smart speaker module and helping Agam with the Central Processing Unit module. She will be focusing on data processing within the

NGROK server, establishing the NGROK connection with the Alexa system, and establishing the Bluetooth Arduino connection.

Appendix 7 - User Manual

Product description

Welcome to your Alexa-controlled Smart Plug! This device will allow you to control common household appliances with a simple voice command, giving you the ability to turn on and off objects such as your televisions, coffee machines, lamps, or any other appliance that connects to power via a type A or type B power plug.

This device also lets you track the power usage of such objects via a series of simple voice commands. Remember to say "Ask HOME to" before each command:

***Table** 7. Voice commands*

Task	**Ask "HOME"**
Turn on plug one	*"turn plug one on"* *"turn one on"* *"switch plug one on"* *"turn switch one on"*
Turn off plug one	*"turn plug one off"* *"turn one off"* *"switch plug one off"* *"turn switch one off"*
Turn on plug two	*"turn plug two on"* *"turn two on"* *"switch plug two on"* *"turn switch two on"*
Turn off plug two	*"turn plug two off"* *"turn two off"* *"switch plug two off"* *"turn switch two off"*
Turn on plug three	*"turn plug three on"* *"turn three on"* *"switch plug three on"* *"turn switch three on"*
Turn off plug three	*"turn plug three off"* *"turn three off"* *"switch plug three off"*

	“turn switch three off”
Gets the most recent (last minute average) power usage from plug one	*“the most recent power for plug one”*
Gets the most recent (last minute average) power usage from plug two	*“the most recent power for plug two”*
Gets the most recent (last minute average) power usage from plug three	*“the most recent power for plug three”*
Gets the average power usage from plug one over the last half hour (30 min)	*“for the average power usage for plug one in the last half hour”*
Gets the average power usage from plug two over the last half hour (30 min)	*“for the average power usage for plug two in the last half hour”*
Gets the average power usage from plug three over the last half hour (30 min)	*“for the average power usage for plug three in the last half hour”*
Gets the average power usage from plug one over the last hour (60 min)	*“for plug ones last hour usage”*
Gets the average power usage from plug two over the last half hour (60 min)	*“for plug twos last hour usage”*
Gets the average power usage from plug three over the last half hour (60 min)	*“for plug threes last hour usage”*
Gets the average power usage from plug one over the last day (24 hours)	*“for plug ones last day usage”*
Gets the average power usage from plug two over the last day (24 hours)	*“for plug twos last day usage”*
Gets the average power usage from plug three over the last day (24 hours)	*“for plug threes last day usage”*

First-time Setup

Smartplug

1. Next, plug the smart-plug into the wall using the three-pronged type B plug located on the exterior of the black box. The Arduino inside of the plug should automatically power on and pair.
2. Using the socket located on the other side of black box, plug your appliance into the smart plug.

Central Router

3. First, take a look at the Raspberry Pi, and plug it into the wall in a safe central location in your home. Avoid placing it in corners of your home or far away (more than 100m) from the smart plugs.
4. If your Raspberry Pi does not already have one, insert a microSD card into the USB card reader on the Raspberry Pi.
5. Power on the Raspberry Pi.
    a. The Raspberry Pi should automatically login and the pre-configured GW Wi-Fi. No keyboard or mouse, or user set-up, is necessary.
6. Ensure that the smart plug is powered on before the central router to ensure successful pairing.

Alexa

7. To use voice commands, the user will need any Alexa-enabled device such as the Amazon Echo Dot or Amazon Echo.
8. Download the Alexa app and sign in. The Alexa app is available on
    a. Fire OS 3.0 or higher.
    b. Android 4.4 or higher.
    c. iOS 8.0 or higher.
9. To download the Alexa app, go to the app store on your mobile device and search for "Alexa app." Then select and download the app.
10. Turn on the Alexa-enabled device.
    a. Plug the included power adapter into Amazon Echo (1st Generation) and then into a power outlet. The light ring on Amazon Echo (1st Generation) turns blue, and then orange. When the light turns orange, Amazon Echo (1st Generation) greets you.
11. Following the instructions in the Alexa app, connect Amazon Echo to a Wi-Fi network.
12. You can now talk to Alexa.

Appendix 8 - Board Fabrication Details

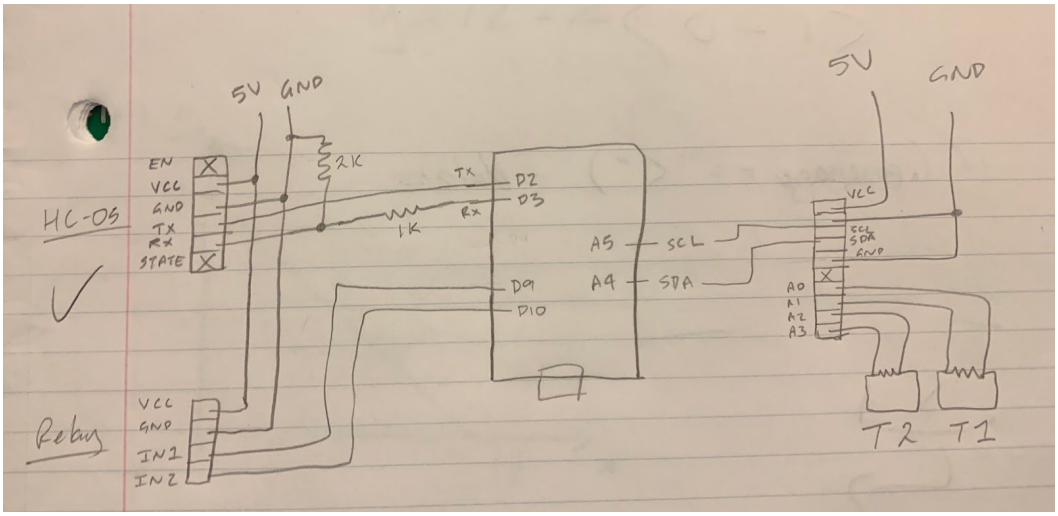

Figure 9. *Circuit Board*

Appendix 9 - Additional Figures and Images

Figure 10. *CPU Setup for Testing*

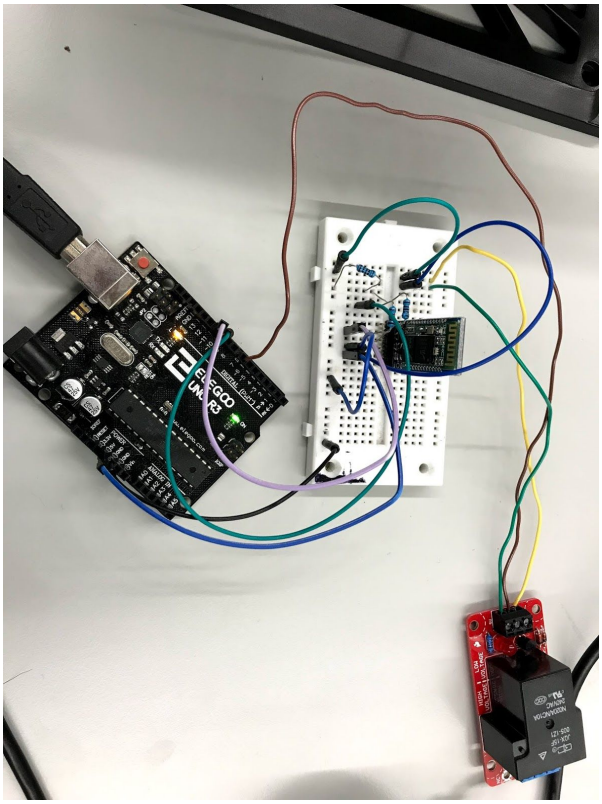

Figure 11. *Appliance Controller Setup for Testing*

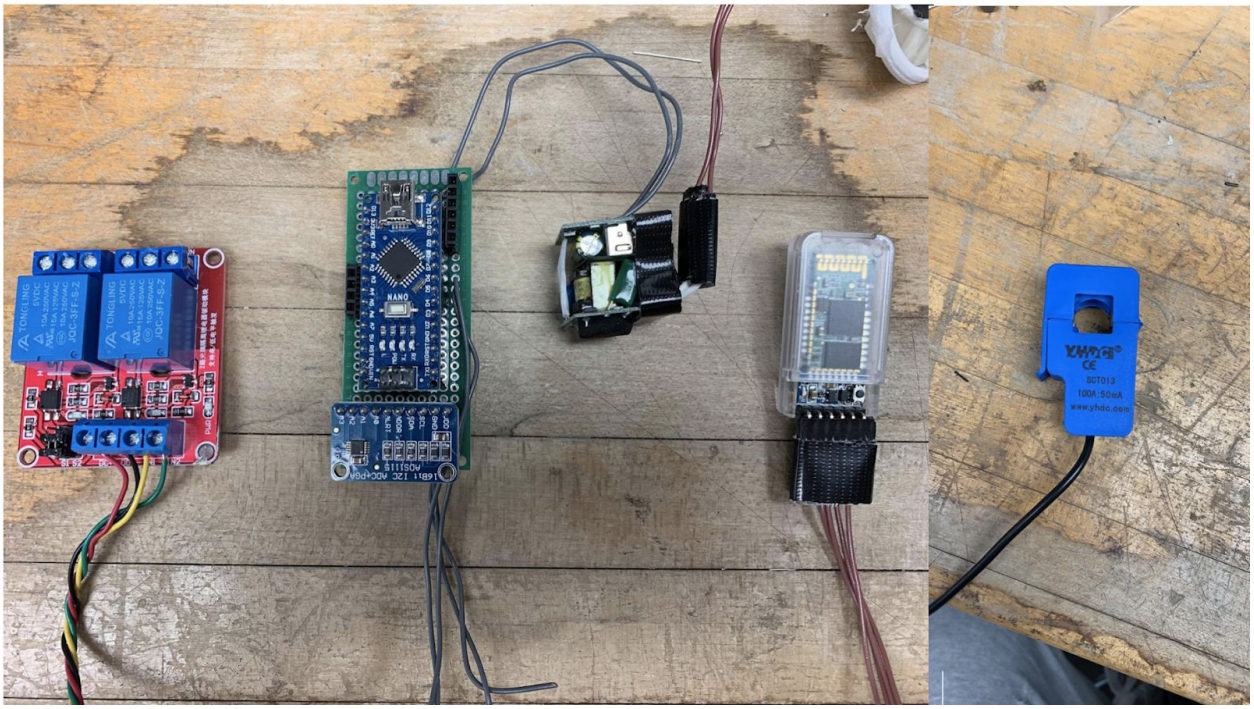

Figure 12. *Switch Internal Components*

From right to left: Current transformer, HC-05 bluetooth module, AC-DC 5V converter, circuit board with Arduino Nano and ADS1115 analog to digital converter, dual relay board

Figure 13. *Appliance controller - Final smartplug prototype with casing*

Appliance 10 - All Software Codes

Raspberry Pi Code

```
import serial
import numpy as np
import time
from flask import Flask
from flask_ask import Ask, statement, question, session
from datetime import datetime
from threading import Thread
```

```
import queue
import MySQLdb
import random

cnx= {'host': 'ammdb.clhkk5qyeyu2.us-east-1.rds.amazonaws.com',
  'username': 'amm',
  'password': 'Aslani123!',
  'db': 'sdamm'}

db = MySQLdb.connect(cnx['host'],cnx['username'],cnx['password'], cnx['db'])

cur = db.cursor()

cur.execute("DELETE FROM plug1")
cur.execute("DELETE FROM plug2")
cur.execute("DELETE FROM plug3")

db.commit()

q = queue.Queue()
q.put("0.0")

ser1 = serial.Serial('/dev/rfcomm0', 9600)
ser2 = serial.Serial('/dev/rfcomm1', 9600)


ino1 = [0,0,0,0,0,0,0,0,0,0,0,0]
ino2 = [0,0,0,0,0,0,0,0,0,0,0,0]
ino3 = [0,0,0,0,0,0,0,0,0,0,0,0]

app = Flask(__name__)
ask = Ask(app, "/switch")

print('------ ', datetime.now(), ' Program started')   #Will print when program initially runs

print("Waiting for data...")

# Sends the switch number, then a '-', then a '1' for high or a '0' for low
def switch1_on():    #Turns on switch 1
    print('switch 1 has turned on')
    ser1.write('<1-1>'.encode('utf-8'))

def switch1_off():   #Turns off switch 1
    print('switch 1 has turned off')
    ser1.write('<1-0>'.encode('utf-8'))
```

```
def switch2_on():    #Turns on switch 2
    print('switch 2 has turned on')
    ser2.write('<2-1>'.encode('utf-8'))

def switch2_off():   #Turns off switch 2
    print('switch 2 has turned off')
    ser2.write('<2-0>'.encode('utf-8'))

def switch3_on():    #Turns on switch 3
    print('switch 3 has turned on')
    ser2.write('<3-1>'.encode('utf-8'))

def switch3_off():   #Turns off switch 3
    print('switch 3 has turned off')
    ser2.write('<3-0>'.encode('utf-8'))


# Added intents for all 3 switches
@ask.intent("OnFromLaunchIntentOne")
def turn_on1_from_launch():
    switch1_on()          #function called to turn off switch
    print(datetime.now(), " Switch turned on") #Tells user what has happened at current date/time
    on_text = "Okay, I've turned it on"
    return statement(on_text)

@ask.intent("OffFromLaunchIntentOne")
def turn_off1_from_launch():
    switch1_off()          #function called to turn off switch
    print(datetime.now(), " Switch turned off") #Tells user what has happened at current date/time
    off_text = "Okay, I've turned it off"
    return statement(off_text)

@ask.intent("OnFromLaunchIntentTwo")
def turn_on2_from_launch():
    switch2_on()          #function called to turn off switch
    print(datetime.now(), " Switch turned on") #Tells user what has happened at current date/time
    on_text = "Okay, I've turned it on"
    return statement(on_text)

@ask.intent("OffFromLaunchIntentTwo")
def turn_off2_from_launch():
    switch2_off()          #function called to turn off switch
    print(datetime.now(), " Switch turned off") #Tells user what has happened at current date/time
    off_text = "Okay, I've turned it off"
```

```
    return statement(off_text)

@ask.intent("OnFromLaunchIntentThree")
def turn_on3_from_launch():
    switch3_on()          #function called to turn off switch
    print(datetime.now(), " Switch turned on") #Tells user what has happened at current date/time
    on_text = "Okay, I've turned it on"
    return statement(on_text)

@ask.intent("OffFromLaunchIntentThree")
def turn_off3_from_launch():
    switch3_off()          #function called to turn off switch
    print(datetime.now(), " Switch turned off") #Tells user what has happened at current date/time
    off_text = "Okay, I've turned it off"
    return statement(off_text)



#intents for all types of power
@ask.intent("AskMostRecentOne")
def mostRecentMinute1():
    cur.execute("SELECT AVG(a.power_data) FROM (SELECT power_data FROM plug1
ORDER BY collection_time DESC LIMIT 1) a")
    temp = cur.fetchone()
    recent_text = "Your plug one most recent power usage was " + str(temp) + " watts"
    print(recent_text)
    return statement(recent_text)

@ask.intent("AskMostRecentTwo")
def mostRecentMinute2():
    cur.execute("SELECT AVG(a.power_data) FROM (SELECT power_data FROM plug2
ORDER BY collection_time DESC LIMIT 1) a")
    temp = cur.fetchone()
    recent_text = "Your plug two most recent power usage was " + str(temp) + " watts"
    print(recent_text)
    return statement(recent_text)

@ask.intent("AskMostRecentThree")
def mostRecentMinute3():
    cur.execute("SELECT AVG(a.power_data) FROM (SELECT power_data FROM plug3
ORDER BY collection_time DESC LIMIT 1) a")
    temp = cur.fetchone()
    recent_text = "Your plug three most recent power usage was " + str(temp) + " watts"
    print(recent_text)
    return statement(recent_text)
```

```
@ask.intent("AskThirtyAverageOne")
def mostRecentHalfHour1():
    cur.execute("SELECT AVG(a.power_data) FROM (SELECT power_data FROM plug1
ORDER BY collection_time DESC LIMIT 30) a")
    temp = cur.fetchone()
    recent_text = "In the last half hour plug one used " + str(temp) + " watts"
    print(recent_text)
    return statement(recent_text)

@ask.intent("AskThirtyAverageTwo")
def mostRecentHalfHour2():
    cur.execute("SELECT AVG(a.power_data) FROM (SELECT power_data FROM plug2
ORDER BY collection_time DESC LIMIT 30) a")
    temp = cur.fetchone()
    recent_text = "In the last half hour plug two used " + str(temp) + " watts"
    print(recent_text)
    return statement(recent_text)

@ask.intent("AskThirtyAverageThree")
def mostRecentHalfHour3():
    cur.execute("SELECT AVG(a.power_data) FROM (SELECT power_data FROM plug3
ORDER BY collection_time DESC LIMIT 30) a")
    temp = cur.fetchone()
    recent_text = "In the last half hour plug three used " + str(temp) + " watts"
    print(recent_text)
    return statement(recent_text)

@ask.intent("AskHourAverageOne")
def mostRecentHour1():
    cur.execute("SELECT AVG(a.power_data) FROM (SELECT power_data FROM plug1
ORDER BY collection_time DESC LIMIT 60) a")
    temp = cur.fetchone()
    recent_text = "In the last hour plug one used " + str(temp) + " watts"
    print(recent_text)
    return statement(recent_text)

@ask.intent("AskHourAverageTwo")
def mostRecentHour2():
    cur.execute("SELECT AVG(a.power_data) FROM (SELECT power_data FROM plug2
ORDER BY collection_time DESC LIMIT 60) a")
    temp = cur.fetchone()
    recent_text = "In the last hour plug two used " + str(temp) + " watts"
    print(recent_text)
    return statement(recent_text)
```

```
@ask.intent("AskHourAverageThree")
def mostRecentHour3():
    cur.execute("SELECT AVG(a.power_data) FROM (SELECT power_data FROM plug3
ORDER BY collection_time DESC LIMIT 60) a")
    temp = cur.fetchone()
    recent_text = "In the last hour plug three used " + str(temp) + " watts"
    print(recent_text)
    return statement(recent_text)

@ask.intent("AskDayAverageOne")
def mostRecentDay1():
    cur.execute("SELECT AVG(a.power_data) FROM (SELECT power_data FROM plug1
ORDER BY collection_time DESC LIMIT 1440) a")
    temp = cur.fetchone()
    recent_text = "In the last day plug one used " + str(temp) + " watts"
    print(recent_text)
    return statement(recent_text)

@ask.intent("AskDayAverageTwo")
def mostRecentDay2():
    cur.execute("SELECT AVG(a.power_data) FROM (SELECT power_data FROM plug2
ORDER BY collection_time DESC LIMIT 1440) a")
    temp = cur.fetchone()
    recent_text = "In the last day plug two used " + str(temp) + " watts"
    print(recent_text)
    return statement(recent_text)

@ask.intent("AskDayAverageThree")
def mostRecentDay3():
    cur.execute("SELECT AVG(a.power_data) FROM (SELECT power_data FROM plug3
ORDER BY collection_time DESC LIMIT 1440) a")
    temp = cur.fetchone()
    recent_text = "In the last day plug three used " + str(temp) + " watts"
    print(recent_text)
    return statement(recent_text)




def saveReading(temp, q):
    power = str(temp)
    q.get()
    q.put(power)
    newReading = time.strftime("%Y-%m-%d %H:%M:%S") + \
```

```
            ',' + power + '\n'
    print('Saving new reading: ' + newReading)
    with open('/home/pi/Desktop/powerReadings.csv', 'ab') as file:
        file.write(newReading.encode('utf-8'))

def readFrom(q):
    start = False
    temp = []
    ser1.flushInput()
    ser2.flushInput()
    while (True):
        ser1.flushInput()
        ser2.flushInput()

        for i in range(12):
            #send signal ready for 1
            ser1.write('<1-2>'.encode('utf-8'))

            #Read one byte at a time
            while(True):
                if(ser1.inWaiting() > 0):
                    character = ser1.read()
                    asciiOrd = ord(character)
                    #If it is a start sequence and we have already started,
                    #start over.
                    if (asciiOrd == 60 and start == True):
                        temp = []
                    #If it is a start sequence and we have not started,
                    #start now
                    elif (asciiOrd == 60 and start == False):
                        start = True

                    #If it is not a start or a stop, and we have started,
                    #simply append.
                    elif (asciiOrd != 60 and asciiOrd !=62 and start == True):
                        temp.append(character.decode('ascii'))
                    #If it is an end character, and we have started then we are done.
                    elif (asciiOrd == 62 and start == True):

                        #If there is something there, and it is a proper float
                        if len(temp) > 0:
                            try:
                                converted = float(''.join(temp))
                                ino1[i] = converted;
                                power = str(converted)
```

```
                saveReading(converted, q)
                #Acknowledge receipt of data
                ser1.write('<5>'.encode('utf-8'))
            except Exception as e:
                print(e)
        start = False
        temp = []
        break

#send signal ready for 2
ser2.write('<2-2>'.encode('utf-8')) #to be updated

while(True):
    if(ser2.inWaiting() > 0):
        character = ser2.read()
        asciiOrd = ord(character)
        #If it is a start sequence and we have already started,
        #start over.
        if (asciiOrd == 60 and start == True):
            temp = []
        #If it is a start sequence and we have not started,
        #start now
        elif (asciiOrd == 60 and start == False):
            start = True

        #If it is not a start or a stop, and we have started,
        #simply append.
        elif (asciiOrd != 60 and asciiOrd !=62 and start == True):
            temp.append(character.decode('ascii'))
        #If it is an end character, and we have started then we are done.
        elif (asciiOrd == 62 and start == True):

            #If there is something there, and it is a proper float
            if len(temp) > 0:
                try:
                    converted = float(''.join(temp))
                    ino2[i] = converted
                    power = str(converted)
                    saveReading(converted, q)
                    #Acknowledge receipt of data
                    ser2.write('<5>'.encode('utf-8'))
                except Exception as e:
                    print(e)
            start = False
            temp = []
```

```
            break


    #send signal ready for 3
    ser2.write('<3-2>'.encode('utf-8')) #to be updated

    while(True):
        if(ser2.inWaiting() > 0):
            character = ser2.read()
            asciiOrd = ord(character)
            #If it is a start sequence and we have already started,
            #start over.
            if (asciiOrd == 60 and start == True):
                temp = []
            #If it is a start sequence and we have not started,
            #start now
            elif (asciiOrd == 60 and start == False):
                start = True

            #If it is not a start or a stop, and we have started,
            #simply append.
            elif (asciiOrd != 60 and asciiOrd !=62 and start == True):
                temp.append(character.decode('ascii'))
            #If it is an end character, and we have started then we are done.
            elif (asciiOrd == 62 and start == True):

                #If there is something there, and it is a proper float
                if len(temp) > 0:
                    try:
                        converted = float(''.join(temp))
                        ino3[i] = converted
                        power = str(converted)
                        saveReading(converted, q)
                        #Acknowledge receipt of data
                        ser2.write('<5>'.encode('utf-8'))
                    except Exception as e:
                        print(e)
                start = False
                temp = []
                break


print(ino1)
switch1 = sum(ino1)/len(ino1)
```

```
        print(switch1)
        print(ino2)
        switch2 = float(sum(ino2)/len(ino2))
        print(switch2)
        print(ino3)
        switch3 = float(sum(ino3)/len(ino3))
        print(switch3)


        now = datetime.now()
        formatted_date = now.strftime('%Y-%m-%d %H:%M:%S')

        cur.execute("INSERT INTO plug1 (collection_time,power_data) VALUES (%s,%s)",
(formatted_date, switch1))
        cur.execute("INSERT INTO plug2 (collection_time,power_data) VALUES (%s,%s)",
(formatted_date, switch2))
        cur.execute("INSERT INTO plug3 (collection_time,power_data) VALUES (%s,%s)",
(formatted_date, switch3))

        db.commit()


if __name__ == '__main__':
    t1 = Thread(target = readFrom, args = (q,))
    t1.start()
    app.run(debug=False)
```

Switch 1 Code

```
#include <SoftwareSerial.h>
#include <Wire.h>
#include <Adafruit_ADS1015.h>

Adafruit_ADS1115 ads; //(0x48);

SoftwareSerial btSerial(14, 16); // RX, TX

const int senderPin = 7; //Tells us we are sending over Serial
const int receiverPin = 8; //Tells us we are receiving confirmation over Serial
const int sendSwitch = 2;

int count = 0;
int flag = 0;
int current_flag = 0;
char data[5] = "";
```

```
const float FACTOR = 16.5; //CT Calibration factors
const float multiplier = 0.0625F;

float degreesC;
unsigned long timer = 0;
int sendStatus = 0;

int buttonState;             // the current reading from the input pin
int lastButtonState = LOW;   // the previous reading from the input pin
unsigned long lastDebounceTime = 0;  // the last time the output pin was toggled
unsigned long debounceDelay = 50;    // the debounce time; increase if the output flickers

void setup() {
  //Pin Modes
  pinMode(senderPin, OUTPUT);
  pinMode(receiverPin, OUTPUT);
  pinMode(sendSwitch, INPUT_PULLUP);
  pinMode(7, OUTPUT);
  pinMode(6, INPUT_PULLUP);

  ads.setGain(GAIN_TWO); // +/- 2.048V 1 bit = 0.0624mV
  ads.begin();

  //Setup and flush the serials to begin
  btSerial.begin(9600);
  Serial.begin(9600);
  btSerial.flush();
  Serial.flush();
  delay(50);
  digitalWrite(7, HIGH);
  delay(50);
  digitalWrite(7, LOW);
}

void printMeasure(String prefix, float value, String postfix)
{
  Serial.print(prefix);
  Serial.print(value, 8); // 3 decimals??
  Serial.print(postfix);
}

void loop() {
  //I want to send the current, I want the Raspberry PI to grab it, process it
  //and send back a message. I don't want to continue spamming the raspberry pi, so
```

```
  //I will only send a signal every 1 seconds.

  //Non-blocking every 10 seconds.
  if ((timer == 0 || millis() >= timer) && current_flag == 1){
   float power = getPower();
   printMeasure("Power: ", power, "W \n");
   //Send the current power usage. Didn't use readline to avoid blocking problem
   String sendPower = "<" + String(power, 3) + ">";

   //Convert to byte array
   char charArray[sendPower.length() + 1];
   sendPower.toCharArray(charArray, sendPower.length()+1);

   btSerial.write(charArray);
   current_flag = 0;

   //Reset the timer for another 1 seconds.
   timer = millis() + 1000;

  }
 //Debouncing
  int reading = digitalRead(6);
  if (reading != lastButtonState) {
   // reset the debouncing timer
   lastDebounceTime = millis();
  }
  if ((millis() - lastDebounceTime) > debounceDelay) {
   if (reading != buttonState) {
    buttonState = reading;
    if (buttonState == LOW) {
     if(flag){
      digitalWrite(7, LOW);
      flag = 0;
     }
     else{
      digitalWrite(7, HIGH);
      flag = 1;
     }
    }
   }
  }
  lastButtonState = reading;

 //Bluetooth receive
  while (btSerial.available()) {
```

```
    char rpiMessage = btSerial.read();
    if(rpiMessage == '<'){
      count = 0;
      for (int x = 0; x < sizeof(data) / sizeof(data[0]); x++)
      {
        data[x] = 0;
      }
      data[count] = rpiMessage;
    }
    else if(rpiMessage == '>'){
      data[count] = rpiMessage;
      if(data[1] == '1'){
        if(data[3] == '1'){
          digitalWrite(7, HIGH);
          flag = 1;
        }
        else if(data[3] == '0'){
          digitalWrite(7, LOW);
          flag = 0;
        }
        else if(data[3] == '2'){
          current_flag = 1;
        }
      }
    }
    else{
      data[count] = rpiMessage;
    }
    count ++;
  }

}

float getPower() {
  float voltage;
  float current;
  float sum = 0;
  long Time = millis();
  int counter = 0;

  while(millis() - Time < 1000){
    voltage = ads.readADC_Differential_0_1() * multiplier;
    current = voltage * FACTOR; //Conductance
    current /= 1000.0;
```

```
   sum += sq(current);
   counter = counter + 1;
  }
  current = sqrt(sum/counter); //avg
  float currentRMS = current;
  float power = 120 * currentRMS;
  return(power);
}
```

Switch 2 Code

```
#include <SoftwareSerial.h>
#include <Wire.h>
#include <Adafruit_ADS1015.h>

Adafruit_ADS1115 ads; //(0x48);

SoftwareSerial btSerial(2, 3); // RX, TX

const int senderPin = 7; //Tells us we are sending over Serial
const int receiverPin = 8; //Tells us we are receiving confirmation over Serial
const int sendSwitch = 2;

int count = 0;
int flag2 = 0;
int flag3 = 0;
int current_flag_2 = 0;
int current_flag_3 = 0;
char data[5] = {};

const float FACTOR = 17.3; //CT Calibration factors
const float FACTOR2 = 16.5;
const float multiplier = 0.0625F;

float degreesC;
unsigned long timer2 = 0;
unsigned long timer3 = 0;

int buttonState;             // the current reading from the input pin
int buttonState1;             // the current reading from the input pin
int lastButtonState = LOW;   // the previous reading from the input pin
int lastButtonState1 = LOW;   // the previous reading from the input pin
unsigned long lastDebounceTime = 0;  // the last time the output pin was toggled
unsigned long lastDebounceTime1 = 0;  // the last time the output pin was toggled
unsigned long debounceDelay = 50;    // the debounce time; increase if the output flickers
```

```
void setup() {
  //Pin Modes
  pinMode(senderPin, OUTPUT);
  pinMode(receiverPin, OUTPUT);
  pinMode(sendSwitch, INPUT_PULLUP);
  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);
  pinMode(6, INPUT_PULLUP);
  pinMode(7, INPUT_PULLUP);

  ads.setGain(GAIN_TWO); // +/- 2.048V 1 bit = 0.0624mV
  ads.begin();

  //Setup and flush the serials to begin
  btSerial.begin(9600);
  Serial.begin(9600);
  btSerial.flush();
  Serial.flush();
  delay(50);
  digitalWrite(8, HIGH);
  digitalWrite(9, HIGH);
  delay(50);
  digitalWrite(8, LOW);
  digitalWrite(9, LOW);
}

void loop() {
  if ((timer2 == 0 || millis() >= timer2) && current_flag_2 == 1){
    float power2 = getPower2();
    String sendPower2 = "<" + String(power2, 3) + ">";
    char charArray2[sendPower2.length() + 1]; //Convert to byte array
    sendPower2.toCharArray(charArray2, sendPower2.length()+1);
    btSerial.write(charArray2);
    current_flag_2 = 0;
    //Reset the timer for another 1 seconds.
    timer2 = millis() + 1000;

  }
  if ((timer3 == 0 || millis() >= timer3) && current_flag_3 == 1){
    float power3 = getPower3();
    String sendPower3 = "<" + String(power3, 3) + ">";
    char charArray3[sendPower3.length() + 1]; //Convert to byte array
    sendPower3.toCharArray(charArray3, sendPower3.length()+1);
    btSerial.write(charArray3);
```

```
    current_flag_3 = 0;
    //Reset the timer for another 1 seconds.
    timer3 = millis() + 1000;

  }

//Debouncing
  int reading = digitalRead(6);
  if (reading != lastButtonState) {
    // reset the debouncing timer
    lastDebounceTime = millis();
  }
  if ((millis() - lastDebounceTime) > debounceDelay) {
    if (reading != buttonState) {
      buttonState = reading;

      // only toggle the LED if the new button state is HIGH
      if (buttonState == LOW) {
        if(flag2){
          digitalWrite(8, LOW);
          flag2 = 0;
        }
        else{
          digitalWrite(8, HIGH);
          flag2 = 1;
        }
      }
    }
  }
  int reading1 = digitalRead(7);
  if (reading1 != lastButtonState1) {
    // reset the debouncing timer
    lastDebounceTime1 = millis();
  }
  if ((millis() - lastDebounceTime1) > debounceDelay) {
    if (reading1 != buttonState1) {
      buttonState1 = reading1;
      if (buttonState1 == LOW) {
        if(flag3){
          digitalWrite(9, LOW);
          flag3 = 0;
        }
        else{
          digitalWrite(9, HIGH);
          flag3 = 1;
```

```
      }
     }
    }
   }
  lastButtonState = reading;
  lastButtonState1 = reading1;

//Bluetooth receive
  while (btSerial.available()) {
    char rpiMessage = btSerial.read();
    Serial.print(rpiMessage);
    if(rpiMessage == '<'){
      count = 0;
      for (int x = 0; x < sizeof(data) / sizeof(data[0]); x++)
      {
        data[x] = 0;
      }
      data[count] = rpiMessage;
    }
    else if(rpiMessage == '>'){
      data[count] = rpiMessage;
      if(data[1] == '2'){
        if(data[3] == '1'){
          digitalWrite(8, HIGH);
          flag2 = 1;
        }
        else if(data[3] == '0'){
          digitalWrite(8, LOW);
          flag2 = 0;
        }
        else if(data[3] == '2'){
          current_flag_2 = 1;
        }
      }
      if(data[1] == '3'){
        if(data[3] == '1'){
          digitalWrite(9, HIGH);
          flag3 = 1;
        }
        else if(data[3] == '0'){
          digitalWrite(9, LOW);
          flag3 = 0;
        }
        else if(data[3] == '2'){
          current_flag_3 = 1;
```

```
      }
     }

   }
   else{
    data[count] = rpiMessage;
   }
   count ++;
  }
}

float getPower2(){
 float voltage;
 float current;
 float sum = 0;
 long Time = millis();
 int counter = 0;

 while(millis() - Time < 1000){
  voltage = ads.readADC_Differential_0_1() * multiplier;
  current = voltage * FACTOR2; //Conductance
  current /= 1000.0;

  sum += sq(current);
  counter = counter + 1;
 }
 current = sqrt(sum/counter); //avg
 float currentRMS = current;
 float power = 120 * currentRMS;
 return(power);
}

float getPower3() {
 float voltage;
 float current;
 float sum = 0;
 long Time = millis();
 int counter = 0;

 while(millis() - Time < 1000){
  voltage = ads.readADC_Differential_2_3() * multiplier;
  current = voltage * FACTOR; //Conductance
  current /= 1000.0;

  sum += sq(current);
```

```
    counter = counter + 1;
  }
  current = sqrt(sum/counter); //avg
  float currentRMS = current;
  float power = 120 * currentRMS;
  return(power);
}
```