

Wirelessly Controlled Smart Outlet

By

Andrew Godsil & Ariel Dubinsky

Senior Project

ELECTRICAL ENGINEERING DEPARTMENT

California Polytechnic State University

San Luis Obispo

2014

TABLE OF CONTENTS

<i>Section</i>	<i>Page</i>
Abstract.....	0
1 Introduction.....	1
1.1 Existing Devices.....	1
1.1 A Friendly User Interface.....	6
2 Requirements and Specifications	7
3 Functional Decomposition	10
4 Design Considerations.....	13
4.1 Microcontroller.....	13
4.2 Communication.....	15
4.2.1 SMS and Twitter.....	15
4.2.2 Email Parsing.....	16
4.2.3 Google Calendar API for Python v.2.....	22
4.2.4 Google Calendar API for Python v.3.....	28
4.3 Wi-Fi Configuration.....	31
4.4 Automatic Reconnection to Wi-Fi.....	36
4.4.1 Testing reconnection to Wi-Fi.....	37
4.5 Hardware.....	41
5 Conclusions.....	47
References.....	49
Appendix A: Senior Project Analysis	54

LIST OF TABLES AND FIGURES

<i>Table</i>	<i>Page</i>
Table 1.1: Existing list of commercially available products and their features alongside proposed Smart Outlet model.....	3
Table 1.2: Extended list of products and their features.....	4
Table 2.1: Wirelessly Controlled Smart Outlet requirements and specifications	77
Table 2.2: Wirelessly Controlled Smart Outlet deliverables.....	9
Table 3.1: Smart Outlet inputs, outputs, and functionality	10
Table 3.2: AC-DC converter inputs, outputs, and functionality	111
Table 3.3: DC-DC converter inputs, outputs, and functionality	122
Table 3.4: Microcontroller inputs, outputs, and functionality.....	122
Table 3.5: Relay inputs, outputs, and functionality.....	122
Table 4.1: Microcontroller specifications	133
Table 4.2: Microcontroller Decision Matrix	14
Table 4.3: Raspberry Pi Specifications	14
Table 4.4: Components needed for Adafruit tutorial	1717
Table 4.5: Adafruit tutorial code version 1.1	1919
Table 4.6: API v.2 code for Smart Outlet operation	2323
Table 4.7: API v.3 code for Smart Outlet operation	2828
<i>Figure</i>	<i>Page</i>
Figure 1.1: Image of Belkin Insight Switch with smartphone application	5
Figure 1.2: Sample Google Calendar interface	66
Figure 3.1: Smart Outlet level 0 block diagram.....	10
Figure 3.2: Smart Outlet level 1 block diagram.....	111
Figure 4.1: Flowchart of proposed Twitter method.....	16
Figure 4.2: Adafruit tutorial setup	1818
Figure 4.3: Sample status report of unread messages in Smart Outlet's inbox.....	19
Figure 4.4: Flowchart of proposed email parsing method	21
Figure 4.5: Flowchart of Google Calendar API method	27
Figure 4.6: General network settings located within the terminal	333
Figure 4.7: List of available Wi-Fi networks.....	333
Figure 4.8: Network specifications within the terminal.....	34
Figure 4.9: Wpa_supplicant file with multiple types of network connection points	35
Figure 4.10: Confirming the link to the network, by looking at the IP address.....	36
Figure 4.11: Identifying hotspot's name.....	37
Figure 4.12: Turning on the hotspot before plugging in the Smart Outlet.....	37
Figure 4.13: Confirmation on Smart Outlet startup that a wireless connection exists.....	38
Figure 4.14: iPhone recognizes the Smart Outlet being connected to it	38
Figure 4.15: List of available networks the Raspberry Pi can detect.....	39
Figure 4.16: Dropping the network and then reestablishing connection, looking at the Raspberry Pi.....	40
Figure 4.17: Test circuit for relay with light bulb.....	41
Figure 4.18: Hardware schematic	422

Figure 4.19: GPIO pins triggering individual relays	433
Figure 4.20: Individual relay output characteristics.....	433
Figure 4.21: Wiring the outlets	44
Figure 4.22: Wiring the charger.....	45
Figure 4.23: Top side of the relay board.....	46
Figure 4.24: Bottom side of the relay board	46
Figure 5.1: The Smart Outlet prototype.....	48

Acknowledgements

Ariel:

I would like to thank my family and friends for their support during my time at Cal Poly, especially my parents and grandparents. I would also like to thank Dr. Braun for his guidance as the project advisor. And finally, I would like to thank my father for his practical guidance and wisdom during the length of the project. His assistance proved invaluable.

Andrew:

This would not have been possible without the tremendous support of my family and friends during the last four years I have had at Cal Poly. I have learned so much, and for that I must also thank the faculty and staff in the Electrical Engineering Department at Cal Poly, especially Dr. Braun. The opportunities everyone has given me are endless.

Abstract

Popularity of home automation devices has increased greatly in recent years due to higher affordability and simplicity through smartphone and tablet connectivity. For the purpose of this experiment, we have developed the Smart Outlet: a stand-alone communication unit, used to connect home outlets to the internet. The Smart Outlet controls lighting and simple appliances throughout the home, remotely, using wireless commands from the web. A Google Calendar interface directly controls the outlets' function by scheduling them with dedicated calendar events. The calendar's interface can even be accessed from multiple different platforms (i.e. phones, computers, tablets) for the convenience of all users. This paper presents the design and implementation of the Smart Outlet together with the calendar application to enable home outlets to receive commands and function on a scheduled basis.

1 Introduction

Home automation is a relatively new area of technology. With the Internet of Things becoming a popular topic within the scientific community, many questions have risen concerning how and which technologies should communicate with each other [1]. In the field of home automation, the focus to integrate electrical devices in the home with each other, utilizes different techniques, but not one correct way [2]. For the purpose of this experiment, we tested different technologies for their viability to expand the existing knowledge in this new field. Our goal introduces another possible solution of centralized control, while discussing its advantages and a possible future direction for home automation.

This paper starts by presenting five representative home automation technologies which exist on the consumer market and describe their main characteristics. We then introduce a new approach, which solves existing models' shortcomings. Next, we illustrate some functions that can contribute to a friendly user interface. Currently, many devices exist in the market, some standard and some proprietary. The diversity constitutes a problem, as these devices are limited to their cost and incompatibility with each other [3]. This poses a dilemma to business owners and homeowners regarding which technology to choose, and the market fragmentation contributes to a slow widespread and evolution of home automation [4].

Rather than proposing yet another method of communicating with outlets, we advocate the possibility of achieving integration and interoperation of existing applications at a high level. We present an approach to interface home outlets with a familiar calendar application that people already use in their everyday lives. This approach favors the integration of many types of schedules and achieves interoperability between the device's schedules and the user's.

1.1 Existing devices

The common American home contains many electronics and home appliances. Every day we switch them on/off one by one. Sometimes we may need to walk upstairs just to turn off a switch. Additionally, when away from home we sometimes wonder: "Did I leave the light on?", "Oh, no. I forgot to turn off the iron!", "Are the kids watching TV during finals week?" With automated plug in modules, we can wirelessly switch our appliances off in seconds with our smartphones.

Automated plug-in modules offer consumers a way to put their devices on a schedule in addition to instant switching. These modules grant us the ability to control the time we would like our devices to turn on thereby eliminating standby power, and resulting in utility savings [5]. Standby power, also known as vampire power or phantom load, wastes energy that many people cannot do anything about and happens to cost consumers, in the US alone, more than \$3 billion a year [6]. It occurs in almost every power supply when plugged in. Just notice a cell-phone charger that stays plugged into a wall all day. It becomes warm after leaving it plugged in because it uses electricity – even at a full charge. Our secondary objective to placing devices on a

schedule is to integrate smart plug in modules into consumers' homes and eliminate vampire power.

Many of these devices already exist on the consumer market, available for purchase at online retailers or directly at home hardware stores, such as The Home Depot. Their similarities lie in their single input-to-output ratio, smart phone compatibility, and controllability through Wi-Fi, 3G, or 4G. However, each of these devices can only control one appliance per device and only function with separate calendar applications. Currently, IFTTT, If This Then That, can offer a solution to integrate the application's calendar to a personal or work calendar, but such application for current devices does not exist [7]. Furthermore, a true graphical way for users to drag and drop ON/OFF statuses of an appliance does not exist.

Another communication protocol worth mentioning is X10, which has become an industry standard for today's home automation products [52]. It works by communicating with plug-in modules via power line, by receiving user inputs with RF receivers. Once a user inputs a command through an X10 controller, a receiver module configured to the signal of the controller turns plug-in modules on or off at the other side of the house. The signal traveling through power line passes through the existing wiring of home in the form of a 12-bit 120kHz signal which does not interfere with the 60Hz AC signal [52]. An X10 system requires a user to have an RF controller, an RF receiver, and separate plug-in modules to operate. The benefits of having an X10 system include inexpensive costs, little time to setup, and easy usability. Some disadvantages are communication distances limited to the confines of the home, and a single system can support only a limited amount of plug-in modules. Currently, smartphone app RF controllers exist to accommodate more devices on a single switch, but standard garage-door style clickers also exist. An example of one of these apps is the X10 Commander by Melloware, available on the iTunes App store for \$9.99 [48].

Table 1.1, below, shows a list of existing devices.

Table 1.1: Existing list of commercially available products and their features alongside proposed Smart Outlet model

Existing Products	Price	Number of Outlets	Max Load	Communication Protocol	Does it have an App?	Create Schedules	Manual Override	Monitors Energy Usage
 [8] Belkin WeMo Insight Switch	\$59.99	1	1.8kW/15A	WPA, WPA2, WEP	WeMo App (iOS6 and Android 4.0 only)	Yes	Yes	Yes
 [9] D-Link Wireless Smart Plug	\$49.99	1	1.8kW/15A	WPA, WPA2, WEP, WPS	Smart Plug App (iOS6 and Android 4.0 only)	Yes	Yes	Yes
 Smart Outlet	\$120	4	2.4kW /20A (total)	WPA, WPA2, WEP	Google Calendar (sync with any platform)	Yes	No	No

Existing Products	Price	Number of Outlets	Max Load	Communication Protocol	Does it have an App?	Create Schedules	Manual Override	Monitor Energy Usage
 [51] X10 AM466 3-Pin Appliance Module	\$21	1	1.8kW/15A	X10 Power Line	It uses any X10 remote controller	Yes, using Home Control Assistant	Yes	No
 [50] X10 SR227 Split Receptacle	\$29	1 X10 Outlet 1 Standard operations	1.8kW/15A	X10 Power Line	It uses any X10 remote controller	Yes, using Home Control Assistant	Yes	No
 [49] X10 TM 751 Plug In RF Receiver	\$22	0	N/A	X10 RF/Power Line	It uses any X10 remote controller	Yes, using Home Control Assistant	Yes	No

□

Table 1.2: Extended list of products and their features

Existing Product (Brand)	Features
On/Off Module 2635-222 (INSTEON) [10]	<ul style="list-style-type: none">• Communication Protocol: INSTEON• Smartphone app included• Device needs connects to central hub• Creates schedules for appliances• Blocks 2 outlet ports• Includes manual override• Max General Purpose Load: 15 Amps
SmartPower Outlet (SmartThings) [11]	<ul style="list-style-type: none">• Communication Protocol: Zigbee• Smartphone app included• Creates schedules for appliances• Control over WiFi/3G/4G• Device connects to central hub• Phone's location can activate device• Blocks 2 outlet ports• Includes manual override• Max General Purpose Load: 12 Amps
Zigbee 3-Series Appliance Module (CentraLite) [12]	<ul style="list-style-type: none">• Communication Protocol: Zigbee• Creates schedules for appliances• Control over WiFi/3G/4G• Device connects to central hub• Load detection detects when appliance is plugged in and already on.• Blocks 2 outlet ports• Includes manual override• Max General Purpose Load: 12 Amps

These devices all consist of a 1-to-1 appliance to outlet configuration. The SmartThings SmartPower Outlet retails at \$55, and the Belkin WeMo Insight Switch retails at \$60. These readily available modules can control just one appliance. Currently, a unit does not with multiple outlet control at an affordable price.



Figure 1.1: Image of Belkin Insight Switch with smartphone application

The advantages of using these devices include monitoring energy usage, and controlling appliances. These devices are intended for plug-and-play use, with no complicated setup required so consumers can install them into their own homes with minimal knowledge of home networking or electrical wiring.

Most of these devices function at any time and any location with Wi-Fi or cell service. They also come with dedicated smartphone apps, and all offer a manual override. With smartphones, users can receive notifications right to their phones. This provides a convenient way to check on the status of your home.

We wanted to come up with a solution to the affordable, multiple outlet problem, so we came up with the Smart Outlet. This plug in module, capable of controlling up to 4 appliances independently, only takes up 1 wall socket. All together, we narrowed a price point down to \$120 for this extended feature. In addition to increasing the possible amount of appliances controlled, we wanted to focus on a calendar integration system, which doesn't use a smartphone app. A smartphone app cleverly manages and monitors the use of an appliance, but it clutters the phone, takes up valuable memory, and doesn't merge with other calendars users may have. The Smart Outlet utilizes Google Calendar to activate your appliances. This makes it easy to merge your Smart Outlet schedule with your existing schedules so your appliances can work with your arrival times, meetings, and others' schedules as well. In addition, this provides a graphical way for users to focus on the ON/OFF status of appliances. Google Calendar allows users to schedule devices months in advance too. The operation for the Smart Outlet does not just limit to smartphones anymore, but functions over a desktop and tablets too. Home automation does not have to limit itself solely to mobile application.

1.2 A friendly user-interface

We wanted to use a web-based calendar service familiar to many people, so to minimize a learning curve. A web based calendar system is a perfect tool for organizing events for home, personal, and business use. It's also a great way to share schedules between peers. Google Calendar, a popular calendar application for consumers and businesses, became the natural choice as our user interface and has the following features [13]:

- Share your calendar with others and have them see scheduled events
- Access your calendar on the go with two-way syncing on your phone or tablet
- Receive text and email notifications prior to a scheduled event
- Invite other people to events
- Sync with your desktop applications
- Check your calendar even when offline
- Free

A sample Google Calendar page, in Figure 1.2, shows what a control interface might look like for the Smart Outlet. In order for it to connect with Google Calendar a user must register the Smart Outlet with a Gmail account. The account is free and easy to set up, and should be unique to the device. Multiple appliances can connect to the Smart Outlet, and based on the illustration below synchronous events can occur. After the event ends, the outlet turns off the corresponding appliance, and waits for the next event to occur.

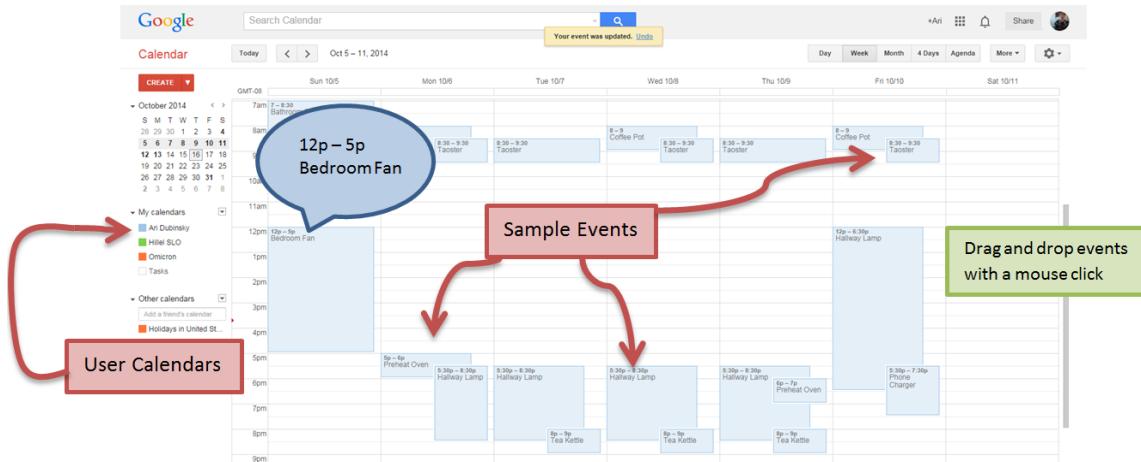


Figure 1.2: Sample Google Calendar interface

This calendar application lets users create and repeat events relatively easily. It allows users to compress and modify times in real time all with drag and drop functionality. Another feature integrates the Smart Outlet calendar with other Google Calendars a user might have access to, in order to see everything in one picture. This helps people integrate their business, home, and Smart Outlet schedules all in one.

2 Requirements and Specifications

Developing design constraints revolves heavily around market research. Market research done on available power switching outlets gives a good baseline on existing products and their features. Currently, available switching outlets lack a compact design with multiple outlet control. In addition, most homes have a home wireless network already running by some kind of router which can interface household appliances over a web based schedule. A Google Calendar interface provides a user-friendly way to interface calendars and to schedule events. It also has the option to send notifications when an event occurs. Using this calendar system, a homeowner can schedule events to happen in the future with the option of making them daily occurrences [14].

Our Smart Outlet's compact design eliminates clutter on someone's wall space. Existing plug-in modules get in the way of other corded devices, making existing wall sockets a shortage in the home. In order to accommodate the user, our device would only occupy one socket space, while providing multiple independently controlled outputs at the end. In this way, the Smart Outlet acts a switching power strip rather than a single input-to-output terminal.

The ability to distinguish between multiple outlets allows a user to turn on one appliance without activating another one simultaneously. Each outlet has its own ID, and after receiving a wireless signal, relays 120VAC from the grid to the object plugged into it. In addition, the enclosure's nonconductive material prevents shock [15]. Table 2.1, below, provides a full list of marketing requirements and specifications for the Smart Outlet.

Table 2.1: Wirelessly Controlled Smart Outlet requirements and specifications

Engineering Specifications	Justification	Marketing Requirements
Google Calendar acts as a control interface for the Smart Outlet, with scheduling capability [13]	Outputs are placed on a schedule.	1
The cost of the device shouldn't exceed \$200 [8-12]	Prototype should be priced appropriately for the typical consumer.	2
A microcontroller receives 5V between 1.2A and 700mA from a common input. The input power should be divided between three outputs and one microcontroller [17]	This minimizes the amount of plugs which go into the wall, reducing outlet space consumption.	1, 5
The Smart Outlet has three 120VAC independently-controlled outputs for one 120VAC input	This makes the device more advantageous than the current competition	5

Three relays switch power to three independent outputs, with specifications at 120VAC ± 10%, 15A, 60Hz. The turn on voltage for the relays is between 0-5V [17, 18]	GPIO pins on the microcontroller control the relays, allowing a high current to pass by use of a low ‘on’ voltage.	6
Microcontroller memory should be non-volatile, so it remembers Wi-Fi connectivity settings each time it is reset [17]	This allows the system to remember its settings after disconnection or resetting occurs.	1, 7
The device should communicate over WLAN using IEEE 802.11b/g specifications. A sample communication protocol includes WPA2-AES [16]	The device should adapt to existing home wireless networks, so to interface with Google Calendar.	4
Enclose it in a non-metallic enclosure designed to work in accordance with Article 314 of the National Electric Code [15]	Non – conductive material improves durability of the enclosure and the UL listing helps ensures safety	3, 8

Marketing Requirements

1. User-friendly
2. Reasonably Priced
3. Safe
4. Wireless communication
5. Independent multiple output control
6. Can turn on any household appliance
7. Plug-and-play operation
8. Enclosed

Table 2.2: Wirelessly Controlled Smart Outlet deliverables

Deliverable Description	Delivery Date
EE 461 Design Review	4/28/14
EE 461 demo	5/19/14
EE 461 report	5/27/14
EE 462 demo device	10/29/14
ABET Sr. Project Analysis	12/8/14
Sr. Project Expo Poster	12/4/14
EE 462 Report	12/11/14

Table 2.2 shows the submission dates for aspects of this project. We portioned the project into two different classes, EE 461 and EE462, where each class required different deliverables. On the right hand column, notice the presentation date for each deliverable. EE 462 required the final report due on 12/11/14.

3 Functional Decomposition

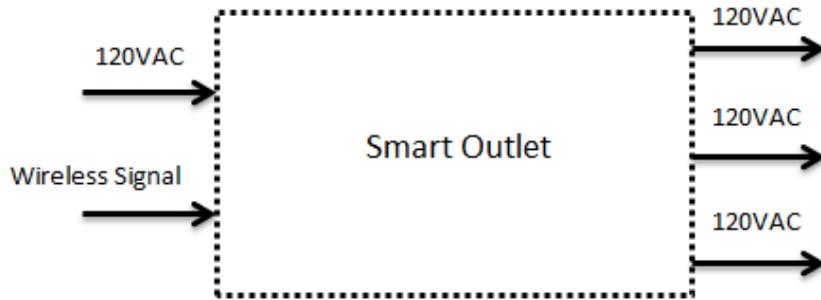


Figure 3.1: Smart Outlet level 0 block diagram

The Smart Outlet acts as a single input, multiple output switch, as shown in Figure 3.1. Each output connects to a common input, where the sum of the total power leaving the Smart Outlet equals the total input power. A wireless signal independently controls each output, which can either operate as an open or closed path. An internal microcontroller device controls each of the loads independently so that one load does not affect the status of another one, but the total power shared across all three outputs cannot exceed 2.4kW [18].

Table 3.1: Smart Outlet inputs, outputs, and functionality

Smart Outlet	
Inputs	1. 120VAC, 60Hz powers the Smart Outlet from a standard American wall outlet 2. Receives a wireless message type via control interface
Outputs	1. 2. 3. Sends 120VAC from the wall outlet when ON and 0V when OFF
Functionality	The Smart Outlet should switch power to an appliance once it receives a wireless message telling it to do so [17]. When it sees the message it responds by sending or blocking power from the outlet with minimal power loss. The total power is either 0W with all three outputs off or 1.8kW with at least one on [20].

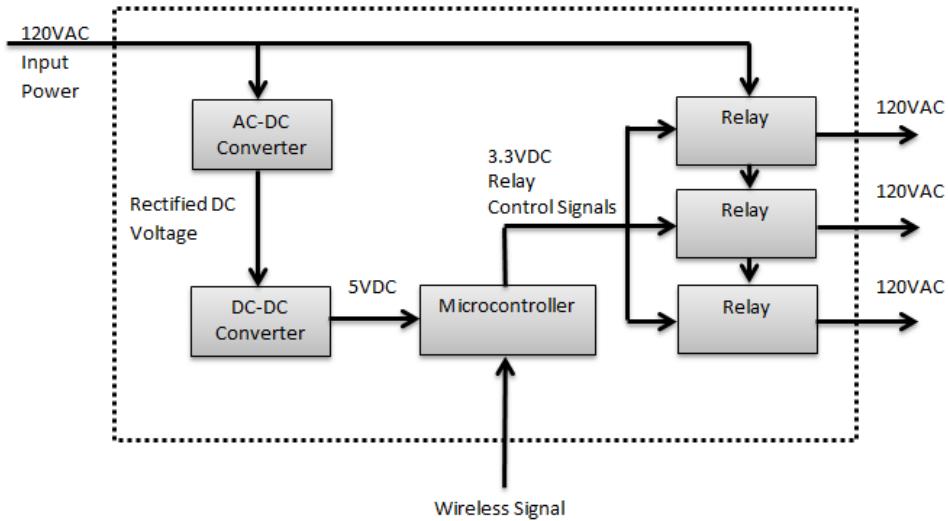


Figure 3.2: Smart Outlet level 1 block diagram

Within the level 1 block diagram, as shown in Figure 3.2, we see a lower level design consisting of black-box-style components. The Smart Outlet acts as a wirelessly controlled switch. The microcontroller receives power from a wall socket, which outputs 120VAC. The AC-DC converter rectifies the signal and the DC-DC converter steps it down to 5VDC, powering on the microcontroller [21]. The microcontroller waits for a communication signal from Google Calendar [22], which designates a specific relay to turn on at a specific time [23]. The microcontroller passes a control voltage of 3.3VDC to the relay, turning it on which allows 120VAC to flow from the mains to the appliance at the load.

Table 3.2: AC-DC converter inputs, outputs, and functionality

AC-DC converter	
Inputs	120VAC
Outputs	120VDC
Functionality	The AC-DC converter takes an AC signal from a wall outlet and outputs a DC signal.

The AC-DC converter and DC-DC converter combine together to form the power supply unit in the Smart Outlet. We utilized the two converters in a cell phone charger, seen in the *Hardware* section of the report, but we split the components of the charger into two tables in order to describe the power supply's functionality. Table 3.2 lists the AC-DC converter's requirements and Table 3.3 lists the DC-DC converter's requirements. The power supply's main function is to convert a 120VAC input signal to a useable 5V/700mA signal to power the microcontroller. The

input voltage must be converted to a DC signal and then stepped down to 5VDC. A cell phone charger neatly and compactly provides this solution.

Table 3.3: DC-DC converter inputs, outputs, and functionality

DC-DC converter	
Inputs	Rectified DC voltage
Outputs	5VDC
Functionality	The DC-DC converter steps down a rectified DC voltage to 5V, powering on the microcontroller.

Table 3.4 contains the microcontroller's basic requirements. The microcontroller needs to connect to the user interface, Google Calendar, and look for events to happen once per minute. A once per minute length provides enough of a response time for appliances to turn on without using too much processing power. When an event occurs, our code tells the microcontroller to send a DC signal from one of its GPIO pins to the relays once a certain event is detected. In this way, the microcontroller serves as the brains for the whole project.

Table 3.4: Microcontroller inputs, outputs, and functionality

Microcontroller	
Inputs	1. 5VDC 2. Google calendar event
Outputs	3.3VDC
Functionality	The microcontroller polls Google Calendar once per minute. When an event occurs it toggles the relay on, until the scheduled even completes.

The relays act as voltage controlled switch with a DC control portion and an AC pass-through portion. Each relay receives a 3.3VDC signal supplied from the microcontroller's GPIO pins which signals the relay to close the path for 120VAC to pass through. 120VAC supplied by the input signal gets passed through the dedicated relay, and turns on a connected appliance. The three different relays connect to the different GPIO pins which designate which relay needs to turn on.

Table 3.5: Relay inputs, outputs, and functionality

Relay	
Inputs	1. 3.3VDC 2. 120VAC
Outputs	120VAC
Functionality	120VAC taken as an input and passed through the relay to the device plugged into it. A 3.3VDC signal turns the relay on and off.

4 Design Considerations

This section presents the design choices and iterations of prototypes the Smart Outlet underwent from conception to most recent model. When considering a possible design, a decision matrix clearly presents the pros and cons. For example; clock speed, power consumption, memory, and cost vary between different products and a decision matrix helps to highlight which product fits the application most suitably. Each module of the Smart Outlet underwent different designs and considerations before landing on the final choice. Refer to Figure A-1 to see initial project plans and Figure A-2 to see the adjusted timeline.

We present different considerations for selecting a microcontroller in this section, and talk about the advantages of one model over the others. We also iterate through the evolutionary steps we took in selecting an appropriate communication protocol. In each communication design section, we reach a stopping point that may not result in fully finished product. We purposefully decide on an advantageous stopping point with that method, in order to move on to the next method. We also discuss our final design choices and features in this section, where we present information for someone to recreate the Smart Outlet's hardware and software if they wish. For a table of project cost estimates, refer to Table A-1 in the Appendix.

4.1 Microcontroller

The following decision matrix, as shown in Table 4.1, helped us decide on the best way to control relays wirelessly. Microcontrollers provide a low power consumption solution in a compact package. Below, Table 4.1 shows a table comparing the product specifications for 3 popular microcontrollers [17] [24] [25].

Table 4.12: Microcontroller specifications

Microcontroller Selection	Price	Processor	Operating Systems	Power Draw	Storage	GPIO Capability	Peripherals	Wireless Comms	Online Support
Raspberry Pi Model B	\$35	700MHz ARM1176JZFS	Raspbian (Recommended), Ubuntu, Android, ArchLinux, FreeBSD, Fedora, RISC OS, others...	150-700 mA @ 5V under varying conditions	SD	8 Pins	2 USB Hosts, 1 Micro-USB Power, 1 10/100 Mbps Ethernet,	\$10 Wi-Fi Dongle	2 nd Largest Community
Arduino UNO Rev. 3	\$26	16MHz ATMega 328	C (Default), C++, Java, ArdOS, Pyxis OS	40-50 mA @ 5V under varying conditions	16kB non-volatile flash memory	14 Digital (6PWM), 6 analog	ISCP header, power jack, USB Type B port	\$90 Wi-Fi Shield	1 st Largest Community
Beaglebone Black	\$45	1Ghz TI Sitara AM3359 ARM Cortex A8	Angstrom (Default), Ubuntu, Android, ArchLinux, Gentoo, Minix, RISC OS, others...	210-460 mA @ 5V under varying conditions	2 GB on-board eMMC, MicroSD	65 Pins	1 USB Host, 1 Mini-USB Client, 1 10/100 Mbps Ethernet	\$10 Wi-Fi Dongle	3 rd Largest Community

Below the specifications, a scale of 1 to 3, within the matrix, signifies the best option (3) to the worst option (1). This ranking system objectively ranks each category in performance based on gathered values and specifications from each of the microcontroller's datasheets. The sums are then totaled to reveal the highest score, which marks the component we chose.

Table 4.2: Microcontroller Decision Matrix

Microcontroller Boards	Raspberry Pi Type B	Arduino UNO Rev. 3	Beaglebone
Price	2	3	1
Processor	2	1	3
Storage	3	1	2
OS	3	1	2
Power Draw	1	3	2
GPIO Capabilities	1	2	3
Peripherals	3	1	2
Wireless Communications	2	1	3
Support	3	2	1
Total	20	15	19

Table 4.3: Raspberry Pi Specifications

- \$35 price tag
- 700MHz ARM1176JZFS processor
- SD ROM storage
- Raspbian (an Ubuntu operating system) with graphical and command line interface
- 750mA@5V power draw
- 8 GPIO pins
- 2 powered USB ports, 1 micro USB power port, 10/100Mbps Ethernet port
- Wireless linkable to the Internet with an extra add-on \$10 Wi-Fi USB dongle
- Large online support community

The Beaglebone has an onboard storage feature which makes installation of Linux operating software easier than the Raspberry Pi. It can also connect to the internet right away, whereas the Raspberry Pi needs proper configuration before it can do this. The Raspberry Pi has a longer initialization process and a steeper learning curve at the beginning, but a larger support community exists for it [26].

The Raspberry Pi is intended for the beginner enthusiast, whereas the Beaglebone is directed to a more serious crowd due to its faster RAM, more GPIO pins, and an overall better graphical user interface. For our purposes, the Raspberry Pi provides enough GPIO pins and the speed differences between the two boards does not matter, because the two boards only differ by a few seconds (when comparing browser initialization, and opening a word processor)[26]. In addition, the Raspberry Pi consumes less power, a useful characteristic, since we want to save power and reduce cost coming from standby mode.

We prioritize a large support community, belonging to the Raspberry Pi rather than the large capabilities of the Beaglebone. Both can act as email clients, and servers, but due to a large support community the Raspberry Pi fits our application better.

4.2 Communication

The heart of this project lies in communicating with the Smart Outlet from a user standpoint. Our objective, to talk to the Smart Outlet by use of an existing interface, may offer the user seamless integration between home and social life in a way that seems natural to the user. Buying into this system forces the user to buy into a social media path predetermined by the creators of Smart Outlet, further increasing user dependency on one media tool but increasing functionality of the tool at the same time. This forms a crosslink between social life and home life in a co-dependent way which blends social habits with household operation. These events can be customized for every user. The program would simply understand which event occurs in a user's life and it would relate that to an event which needs to be executed within the home. For example, the message "coming home" interpreted by the system could mean to turn on a hallway light, start a space heater, and turn on a tea kettle. All of these events customize to each user, in a way that makes a home seem responsive and feel alive. Our primary goal with communications utilizes a popular user interface without the need to create our own. A second objective provides insight to a possible standard of home automation that can include social media to act as a way to control home events. In this paper, we call the wireless interaction between user and home events, peer-to-home communications.

Three possible considerations for peer-to-home communication include SMS messages via Twitter, parsing through email, and using a Google Calendar API. These design choices follow an evolutionary path within the project, where each idea builds on the later to form a final plan. The following sections present benefits and shortcomings of each method.

4.2.1 SMS and Twitter

Twitter's popularity among other social media tools establishes its appeal to the Smart Outlet's application along with its inherent user-friendly interface. Twitter's SMS capability grants the user a familiar input side which can relay information to the Smart Outlet via text messages [27]. Through this peer-to-home method, the Smart Outlet links social commands inputted by the user to events scheduled within the home. This method benefits the user with social commands, such as "bedtime" and "going out" which are quick to type and can switch multiple appliances.

Ideally, every Smart Outlet device gets a separate user account. A user with access to the account would schedule their Smart Outlet to turn on and off by posting specific text commands to the Smart Outlet's account. Twitter would then send an SMS notification, containing the message, to the Raspberry Pi which would interpret and toggle the Smart Outlet's outputs between ON/OFF states. Unfortunately this method requires an additional GSM Shield for the Raspberry Pi, along with a prepaid or monthly cellular plan. The GSM Shield costs approximately \$120 and the

cellular plans add another \$10-\$25 per month [28]. The extra costs within this method make it unrealistic and impractical when compared to other designs. Based on costs alone, this design choice eliminates itself as a too expensive method. The following designs reflect more considerations towards cost.

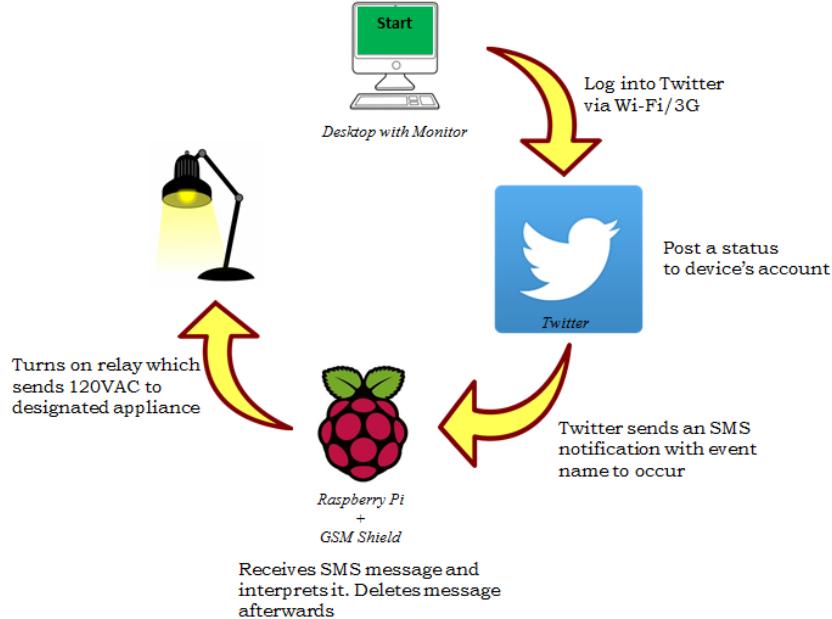


Figure 4.1: Flowchart of proposed Twitter method

Above, Figure 4.1 shows a flow chart explaining the cycle of communication when using the Twitter method. It begins with an inputted command by the user to the device's Twitter account either by directly posting or by hash tagging the device's ID. The Smart Outlet's account would identify this message as relevant and would interpret the command to activate relays. An example of such a command would be “bedtime” and within the software, relays dedicated to turning a lamp off, turning off a cell phone charger, and maybe turning off a space heater would all occur synchronously. The Raspberry Pi would then wait for the next SMS message for the next event.

4.2.2 Email Parsing

This method's objective turns on a relay, when the Raspberry Pi parses through specific text of an unread email. This method relies on a procedure that performs at regular intervals where the Raspberry Pi can receive email messages, read those messages, extract subject lines and intuitively understand what action needs performing, and then activates hardware based on “key phrases”. All of these actions require a Wi-Fi network, and a schedule which operates at periodic intervals. This much more cost friendly method improves on the earlier design because emails are free to send, and there is no additional hardware needed to send and receive emails.

We started by creating an email address for the Raspberry Pi, so it could read incoming messages. We based our code on a script from Adafruit.com, for a project titled “Raspberry Pi Email Notifier Using LEDs” written by Mikey Skylar, which focuses on converting the Raspberry Pi into an email server [29]. This code helped us because we had no experience using the Raspberry Pi and it helped get us to give the Raspberry Pi email client functionality. The code works by informing the user when a new message has appeared in his inbox by having one of the Raspberry Pi’s GPIOs light up an LED on a breadboard, providing a visual notification for a new message.

This tutorial opens the door to our main goal of sending a 3.3V voltage to a control pin of a relay [17]. Once we can send a 3.3V using this tutorial, then receiving emails to turn on a pin stage can be accomplished easily. The following table consists of parts needed to be purchased for the tutorial.

Table 4.4: Components needed for Adafruit tutorial

Quantity	Item	Location	Cost
1	Raspberry Pi Development Board	Amazon.com	\$38.92
1	Adafruit Assembled Pi Cobbler Breakout + Cable	Adafruit.com/products/914	\$7.95
1	Micro USB for power	Already have	--
1	Ethernet Cable	Already have	--
1	4GB SD/Micro SD Memory Card	Adafruit.com/products/102	\$7.95
1	HDMI to DVI cable	Amazon.com	\$5.19
1	USB Keyboard + Mouse	Already have	--

We configured the components in a similar way to the hardware setup of the tutorial (Fig. 4.2). A Secure Shell Client, Putty, is a software program which uses secure shell protocol to connect to a remote computer [30]. In this case, the desktop uses Putty to communicate with the RPi, based on given IP address from the network, known as headless display. Both the Pi and desktop need IP addresses to enable headless communication. For our initial testing purposes we connected the Raspberry Pi to a home network using Ethernet [31]. Later on, we switched to wireless using the Edimax EW-7811Un USB adapter. For an explanation on how to connect Wi-Fi, see the section titled *Connecting to Different Types of Wi-Fi*.

We configured the hardware in the following manner, where we used a ribbon cable connected to a breadboard to interface the LEDs to the RPi’s GPIO pins. The figure below shows how a headless display works to achieve the LED tutorial.

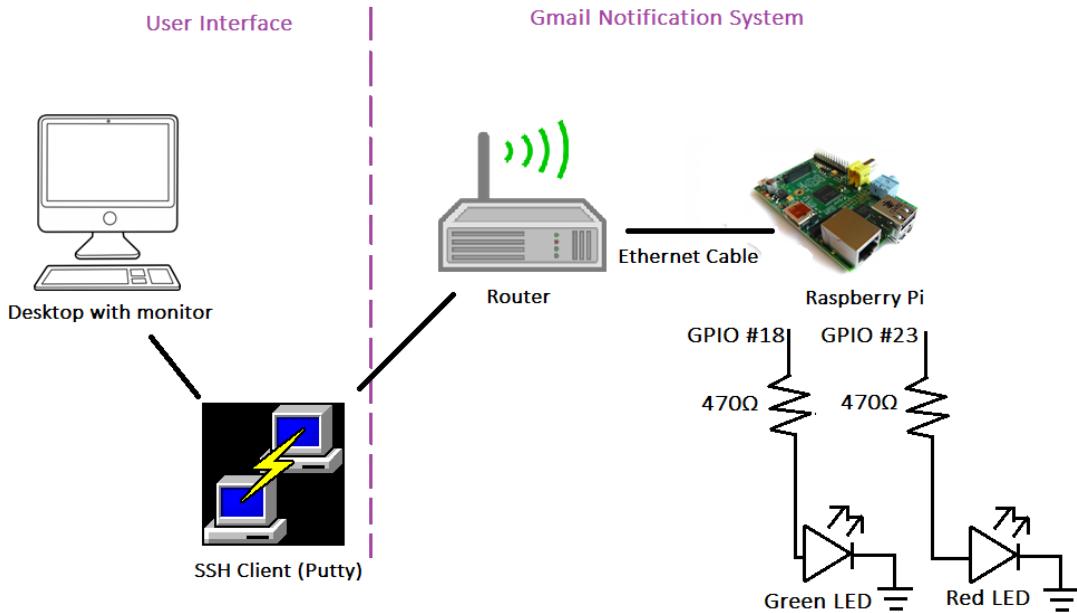


Figure 4.2: Adafruit tutorial setup

This tutorial uses the following steps:

1. Follow tutorial's hardware setup
2. Open up Putty
3. Log into the RPi with IP, username, and password
4. Copy the Python script provided in the tutorial and save it as a .py file within the main directory of the Raspberry Pi. Make sure to include the Smart Outlet's Gmail account in place of existing account
5. Make the file into an application, type chmod +x raspi-gmail.py
6. Run the application, sudo ./raspi-gmail.py

The working code allows a user to read subject lines of all unread messages using the LXTerminal program within the OS. An output message printed to the terminal features numeric markings for each unread message along with a full subject line. The code repeats itself every minute and only checks for unread messages. Our objective with this method uses Google Calendar as a frontend interface for the user, so the user can create events and have an email notification sent to the device's inbox indicating the start of an event. The code polls every minute to check for new unread messages. Once a new unread message is received, the Pi performs text parsing and performs an action based on the 'event title', 'start time', and 'stop

time'. Once the message is understood by the device, a GPIO pin specific to the event title turns on or off, depending on the event, similar to the Twitter method. The code then deletes the message after the stop time, in order to conserve memory.

Figure 4.3 shows a sample output message containing unread messages in the Raspberry Pi's email account, after running the script.

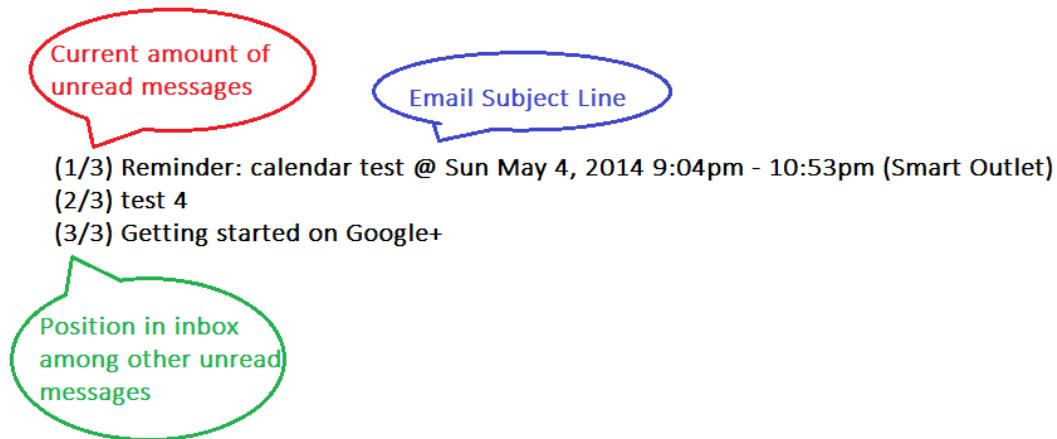


Figure 4.3: Sample status report of unread messages in Smart Outlet's inbox

A similar outputted message prints to the terminal every time the code runs. It contains only unread messages, and gives 3 pieces of important information for each message. First, in parenthesis the relevant time prints next to the total amount in the unread folder. The first number marks which message came first, 1 the newest message and 3 the oldest, in this case. The subject line comes next, and based on the email notification received by Google Calendar, it contains all of the information needed to perform an action. The sequence of key information includes the event name followed by the start time, then end time, finishing with the specific Gmail account. Table 4.5, listed below, contains code modified slightly from its original version [32].

Table 4.5: Adafruit tutorial code version 1.1

```
#This code is borrowed from http://mitchtech.net/connect-raspberry-pi-to-gmail-facebook-twitter-more/
```

```
cat <<!> raspi-gmail.py
#!/usr/bin/env python

import feedparser

DEBUG = 1
```

```

USERNAME = "user@gmail.com"
PASSWORD = "passphrase"

NEWMAIL_OFFSET = 1      # my unread messages never goes to zero, yours might
MAIL_CHECK_FREQ = 10    # check mail every 10 seconds

GPIO.setmode(GPIO.BCM)
GREEN_LED = 18
RED_LED = 23
GPIO.setup(GREEN_LED, GPIO.OUT)
GPIO.setup(RED_LED, GPIO.OUT)

try:
    input=raw_input
except NameError:
    pass

while True:

    response = feedparser.parse("https://" + USERNAME + ":" + PASSWORD +
"@mail.google.com/gmail/feed/atom")      #Create the RSS feed
    unread_count = int(response["feed"]["fullcount"])

    for i in range(0,unread_count):
        print "(" + str((i+1)) + "/" + str(unread_count) + ")" + response['items'][i].title
        #Output amount of unread elements along with subject line on a new line

    if unread_count > NEWMAIL_OFFSET:
        GPIO.output(GREEN_LED, True)
        GPIO.output(RED_LED, False)
        time.sleep(2)           #LED stays green for 2 seconds before resetting
    back to 'no new message' status
        GPIO.output(GREEN_LED, False)
        GPIO.output(RED_LED, True)
    else:
        GPIO.output(GREEN_LED, False)
        GPIO.output(RED_LED, True)

    time.sleep(MAIL_CHECK_FREQ)
    NEWMAIL_OFFSET = unread_count          #If new mail received,
    MAIL_CHECK_FREQ increments

    input("Press any key to continue ")   #Ctrl+Z gets the user out of the loop
!
```

The issue with this method came during the text parsing phase. We need a way to read ‘unread emails’, process and act, then either discard or mark the email as read. Parsing through text poses one issue, but marking messages at the end of an action creates a whole separate problem. The reason for marking messages as unread and read keeps the Pi from parsing through hundreds of unread messages. This would take an enormous amount of time to complete one event. We discontinued going in the text parsing route, because both of these issues require a lot of code writing and processing power. A simpler solution lies in using an existing API client for Google Calendar, which directly links the Pi to events created in Google Calendar [33]. The API method was discovered when browsing Google Data’s site on how to mark messages as read and unread. A more conventional method, using Google Calendar’s API, provides a direct link to the user interface and bypasses parsing through email, which reduces the amount of step to perform an action. Below, a flowchart describes the communication flow of the email parsing method.

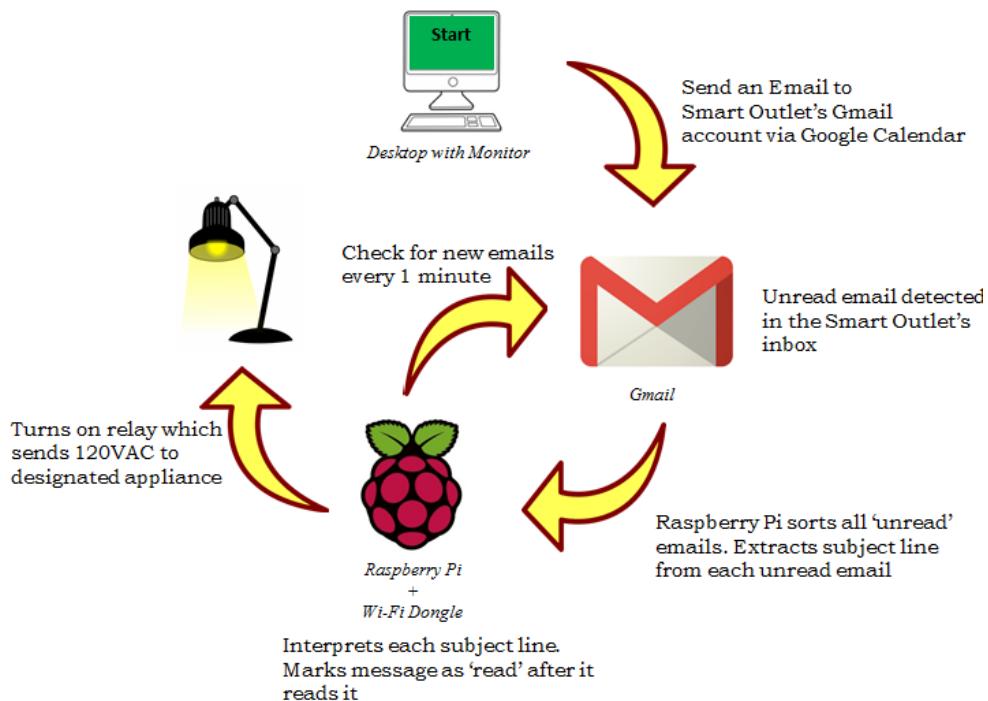


Figure 4.4: Flowchart of proposed email parsing method

Figure 4.4 shows a flowchart example of the email parsing method. A user creates an event in the Raspberry Pi’s Calendar application which sends an email notification to the device’s inbox, upon start time. The message displays a subject line that contains all 3 pieces of important information. The Raspberry Pi then parses word-by-word the message and compares strings to a bank of existing commands the user has specified within the code. The code marks the message

as unread and waits a minute to check the inbox again. Once it reaches the stop time, the Raspberry Pi knows to perform the opposite instruction than what the event titled specified. The code deletes the message from the inbox after.

4.2.3 Google Calendar API for Python v.2

This is a design based on a deprecated version of Google Data API. For an updated code, refer to version 3.

The final design choice was inspired by an open-source watering system which polls Google Calendar to perform an action, such as turning on a sprinkler [34]. This tutorial matches the needs of this project closely, but instead of turning on lawn sprinklers via the web we would like to send electrical power through an outlet by similar means. The tutorial utilizes Google Calendar as a user-interface, which allows users to fully customize times they would like different sprinklers to turn on. Users can modify, create and repeat events all in real time that updates the hardware within 1 minute.

This tutorial gives a good starting point, by providing a foundation to link the microcontroller to Google Calendar. We decided to modify this tutorial for our purposes. We also do not plan on commercializing this product in the near future. Rayshobby.net created the tutorial and can be found by web searching OpenSprinklerPi [34].

The Raspberry Pi and Google Calendar form a wireless connection using a Google Calendar Python API [35]. The microcontroller acts as the client which views the calendar events in the form of Google Data API feeds. The API also grants the microcontroller writing capabilities, but it only needs to read events for the purposes of this project.

This program works by first having the user set up a separate Google Calendar account for their Smart Outlet device. The device can have any name, as long as the Calendar sets ‘public’ in its settings. Do this by going to Settings>Calendars>Shared: Edit Settings>Share this calendar>Check “Make this Calendar Public”. This helps the Raspberry Pi identify the calendar application without additional authentication scripts. Once this account is setup for public access, make sure that the correct calendar is enabled within Google Calendar. To simplify setup, disable all other calendars except the one dedicated to the Smart Outlet.

In the future, we would like to give the user the option of setting the calendar as ‘private’, but the current version of code only supports public settings [36]. From the settings menu of Google Calendar, the user inputs the following information to the provided script. Note that inputted information remains static for each user, but varies for every account. So if someone has multiple

Smart Outlets in their home they can all be tuned to one calendar, requiring the same setup information including Calendar ID, and networks settings.

Below, a list of steps shows the user of the Smart Outlet how to configure the device for their application. We understand that different users will have different Google Calendar's and different appliances, which all need to be altered to enable specific operation. We provide a simplified and generic guide with images, to help anyone who is familiar with the Raspberry Pi setup their Smart Outlet.

1. The user needs to edit the code in Table 4.6 to include a unique Calendar ID and the names of each appliance, so to make the Smart Outlet unique to the appropriate application. Access the Calendar ID by going to Settings>Calendars>Shared: Edit Settings>Calendar Details>Calendar Address. Input the appliance names in the STATIONS section of the code. Include any type of spellings the user may call one appliance to reduce error in inputting the wrong event name in the calendar. Also make sure that the GPIO pins declarations match the provided ones under STATIONS. Copy the code from the table below into a text document and save it with the .py extension. Download the latest version of FileZilla, an FTP client, and connect to the Raspberry Pi via its IP address [37]. Log into the Raspberry Pi and obtain its IP address, to ensure network connection. Connect it to Wi-Fi and type the instructions **ifconfig** to the terminal. This section does not explain how to connect to Wi-Fi, so view the section titled *Connecting to Different Types of Wi-Fi* before running the code below. Input the IP address of the Raspberry Pi into FileZilla and transfer the code to the/home/pi directory. The user can see it in the Raspberry Pi by executing an **ls** function [38]. Table 4.6 contains the API version 2 code.

2. Type:

sudo chmod +x code.py sudo ./code.py	#turns the file into an executable #manually runs the code to check if it is working
---	---

Table 4.6: API v.2 code for Smart Outlet operation

```
#This code is borrowed from the OpenSprinklerPi project, created by Rayshobby.net
```

```
#!/usr/bin/python
```

```
import time
```

```

import sys
import string
import datetime
import RPi.GPIO as GPIO
import atexit

try:
    from xml.etree import ElementTree # for Python 2.5 users
except ImportError:
    from elementtree import ElementTree
import gdata.calendar
import gdata.calendar.service

# =====
# !!! MODIFY THE CALENDAR ID AND STATION NAMES BELOW !!!
# =====

# PUBLIC GOOGLE CALENDAR ID
# - the calendar should be set as public
# - calendar id can be found in calendar settings
# - !!!!!!! PLEASE CHANGE THIS TO YOUR OWN CALENDAR ID !!!!!!
CALENDAR_ID = 'STRING@group.calendar.google.com'

# STATION NAMES
# - specify the name : index for each station
# - station index starts from 0
# - station names are case sensitive
# - you can define multiple names for each station

STATIONS = {
    "master" : 0,
    "relay_a" : 1,      # user can map multiple common names
    "relay a" : 1,      # to the same station index
    "relay_b" : 2,
    "relay b" : 2,
    "relay_c" : 3,
    "s05" : 4,          # user can have up to 16 stations, we just use 3
    "s06" : 5,
    "s09" : 8,
    "s10" : 9,
    "s16" : 15
}

```

```

}

# =====

# MAXIMUM NUMBER OF STATIONS
MAX_NSTATIONS = 64

# PIN DEFINITIONS
relay_a = 22
relay_b = 27
relay_c = 4

calendar_service = gdata.calendar.service.CalendarService()
query = gdata.calendar.service.CalendarEventQuery(CALENDAR_ID, 'public', 'full')
query.orderby = 'starttime'
query.singleevents = 'true'
query.sortorder = 'a'
station_bits = [0]*MAX_NSTATIONS

def runSmartOutlet():

    global station_bits
    now = datetime.datetime.utcnow();
    print datetime.datetime.now();
    nextminute = now + datetime.timedelta(minutes=1)

    query.start_min = now.isoformat()
    query.start_max = nextminute.isoformat()

    for key in STATIONS:
        station_bits[STATIONS[key]] = 0;
    try:
        feed = calendar_service.CalendarQuery(query)
        print '[',
        for i, an_event in enumerate(feed.entry):
            if (i!=0):
                print ',',
            try:
                print an_event.title.text,
                station_bits[STATIONS[an_event.title.text]] = 1;
            except:
                print "-> #name not found#",
                #print '%s' % (an_event.title.text)
        print ']'
    except:
        print "#error getting calendar data#"

```

```

def main():
    print('Smart Outlet has started...')

    GPIO.cleanup()
    # setup GPIO pins to interface with relays
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(relay_a, GPIO.OUT)
    GPIO.setup(relay_b, GPIO.OUT)
    GPIO.setup(relay_c, GPIO.OUT)

while True:
    try:

        runSmartOutlet()
        if station_bits[STATIONS["relay_a"]] == 1:
            print "A is on"
            GPIO.output(relay_a, True)
        else:
            GPIO.output(relay_a, False)
        if station_bits[STATIONS["relay_b"]] == 1:
            print "B is on"
            GPIO.output(relay_b, True)
        else:
            GPIO.output(relay_b, False)
        if station_bits[STATIONS["relay_c"]] == 1:
            print "C is on"
            GPIO.output(relay_c, True)
        else:
            GPIO.output(relay_c, False)

    except:
        pass
    time.sleep(60) # check every 60 seconds. This is the polling feature

def progexit():
    global station_bits
    station_bits = [0]*MAX_NSTATIONS
    GPIO.cleanup()

if __name__ == "__main__":
    atexit.register(progexit)
    main()

```

Below, Figure 4.5 illustrates the communication process for the API method.

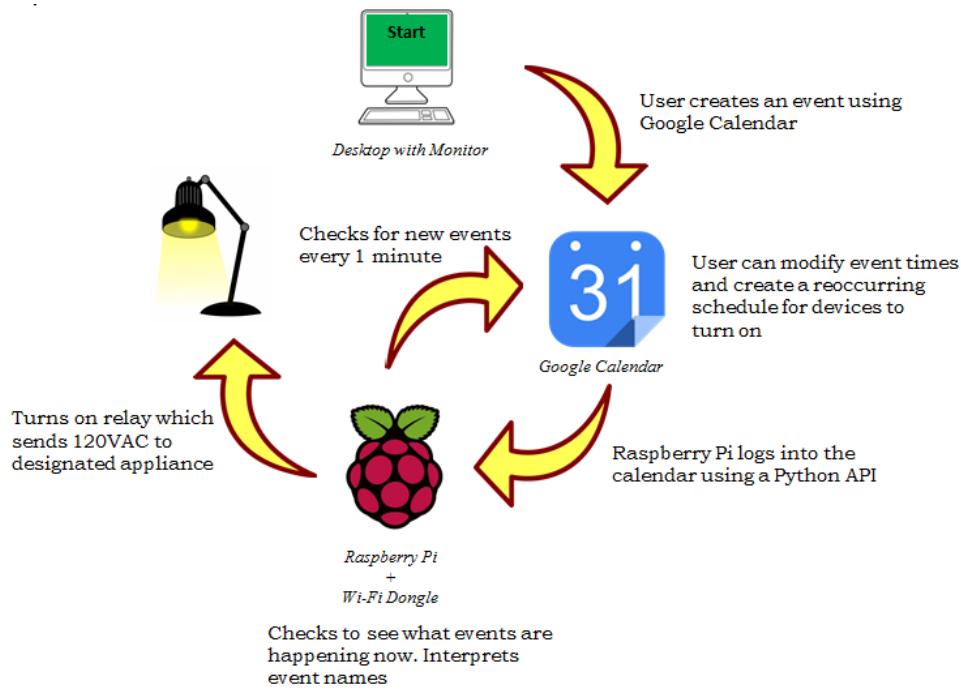


Figure 4.5: Flowchart of Google Calendar API method

The code works by polling the Google Calendar API every minute. Through the API, the Raspberry Pi can access its calendar's public information. See the previous section how to set the calendar to 'public'. Once it logs into the correct calendar, it checks to see what events happen right now. It reads the title of the event and saves it to its memory to compare to a list of commands the user has specified. The event titles match to the correct spellings within the STATIONS array. When the event titles match a GPIO pin signals a relay with 3.3VDC signal supplying constant current of 16mA [17]. The GPIO pins hold on until the event ends. Within the code, a 1 minute polling interval checks for new events every minute.

In order for the code to run on startup without needing log in credentials inputted manually from the user, it needs to run in background applications as a cronjob. A cronjob functions as a safe and special type of file that does not need a user to log into the device to run, but instead runs on startup. This avoids the need for a user to log in to the device, and gives the device its plug-and-play functionality. The code is kept secure from hacking, yet still requires the user ID and password if someone would like to edit it. The code starts running once the Raspberry Pi connects to Wi-Fi. To make the code a cronjob enter the following commands to the input terminal [32].

3. Type:

```
sudo crontab-e  
* * * * * /home/pi/code.py
```

#opens up the editor for cronjob
#The asterisks mark the smallest interval to run
the code once per minute. The directory of the
code follows the asterisks.

4. After this, save the code and exit the editor. Restart the Raspberry Pi and edit some sample events to the calendar application. The code should grab the event titles and switch on a pin. Wire an LED with resistor in series to each pin to check the status of a pin. This can serve as a visual status indicator.
5. After the correct LEDs light up, wire the relays for the next step. Make sure to wire the LEDs in parallel to the DC pins of the relays. The user may find a schematic of how to wire the relays in the *Hardware* section of the report.

The Edimax USB dongle reliably connects the Linux-based Raspberry Pi to Wi-Fi, but it has some issues with connecting over long distances [39]. Make sure to have the device within close proximity to the Wi-Fi source. A weak signal might be the result of a large distance between the Wi-Fi source and the Raspberry Pi. Decrease this distance to strengthen the signal.

4.2.4 Google Calendar API for Python v.3

Starting November 17th, 2014 Google had deprecated the previous version of API, and moved to version 3. This means that the existing code we had for the Smart Outlet is no longer current and can no longer work. Our code is updated for the version 3 API in this section. This also illuminates a weakness of the Smart Outlet in its dependence for Google's services, as Google can change its terms and services which can affect the operation of the Smart Outlet.

We developed new code in a short amount of time, 2 weeks before the project exposition, in order to demonstrate a working product. Table 4.7, below, lists the code.

Table 4.7: API v.3 code for Smart Outlet operation

```
#!/usr/bin/python  
  
import urllib2  
import datetime  
import time
```

```

import json
import RPi.GPIO as GPIO

#PIN DEFINTIONS
relay_a = 25
relay_b = 24
relay_c = 23

print('Smart Outlet has started...')
GPIO.cleanup()
GPIO.setmode(GPIO.BCM) #setup GPIO pins to interface with relays
GPIO.setup(relay_a, GPIO.OUT)
GPIO.setup(relay_b, GPIO.OUT)
GPIO.setup(relay_c, GPIO.OUT)

now = datetime.datetime.utcnow();
nextminute = now + datetime.timedelta(minutes=1)

url =
'https://www.googleapis.com/calendar/v3/calendars/CALENDAR_ID%40group.calendar.google.
com/events?key=public_API_key_for_browser_applications' +\
    '&timeMin=' + now.isoformat() + '-00:00' +\
    '&timeMax=' + nextminute.isoformat() + '-00:00'
#print url

try:
    response = urllib2.urlopen(url).read()#get response from Google server
    #print response

    data = jason.loads(response)          #create JSON object from response string

    activeRealys = "";
    for key in data['items']:
        activeRelays = activeRelays + '-' + key['summary'] + '-'

    #print activeRelays

    if '-relay_a-' in activeRelays:
        print 'A is on'
        GPIO.output(relay_a,True)
    else:
        print 'A is off'
        GPIO.output(relay_a,False)
    if '-relay_b-' in activeRelays:
        print 'B is on'
        GPIO.output(relay_b,True)

```

```

else:
    print 'B is off'
    GPIO.output(relay_b,False)
if '-relay_c' in activeRelays:
    print 'C is on'
    GPIO.output(relay_c,True)
else:
    print 'C is off'
    GPIO.output(relay_c,False)

    print

except:
    print "#error getting calendar data#"
#time.sleep(60) #check every 60 seconds. This is the polling feature

```

This code doesn't exactly poll the API, but instead opens a browser with the Calendar's ID and the API key in the URL.

An API key enables the Raspberry Pi to communicate with Google Calendar using version 3 of the API. Below is a one-time process explaining how to obtain this key. Once the key is fetched, insert it into the proper place within the code in Table 4.7.

1. Create an API key by creating a project in Google's Developer's Console. Then Select APIs & auth>Credentials>Public API access> Create new key>Browser key>Any referrer allowed. The code accesses the calendar's information once every minute by use of polling and with the calendar set to public. The code extracts events for the current time from the calendar in the form of a key 'summary', where the name of the event matches to its specific relay [40]. A GPIO pin turns on, and it continuously supplies 3.3VDC/16mA to the relay until the event ends. Continuous power is sent to an appliance within the one minute waiting period. The process then repeats itself after the one minute mark.
2. Replace the fetched key with the *public_API_key_for_browser_applications* spot within the code.

We removed the polling feature within the code, because a cronjob features a 1 minute repetition cycle. Having a polling feature within the code would double the waiting time, which would lower the response time of the system overall. One minute is the fastest response time the system can achieve based on the execution time limits of the cronjob feature within the Raspberry Pi.

4.3 Wi-Fi Configuration

We wanted to connect the Smart Outlet to common types of Wi-Fi networks within the home. The most common types of protocols configure to use WPA2-PSK, WPA-PSK, and WEP which are part of the IEEE 802.11 wireless networking standard [41]. Note that when using a wireless access point or router the user can send information from one device to another, so anyone within range may also receive it. Protecting data via Wi-Fi follows standard security and encryption schemes that come enabled with the wireless hardware.

This section offers a tutorial on how to connect to Wi-Fi with encryption schemes, so the Smart Outlet can link to a network on startup. Many of these tutorials already appear on the web, but we borrowed their information in order to provide our own tutorial on how to connect to Wi-Fi [42, 43]. Up to this point we could connect to the network by use of Ethernet, but this is a clumsy way to connect to the network, because it limits a cable length's distance between the router and the device. Having a cable on the ground also prohibits safety around the device. We wanted the capability to set our device anywhere within the home, without the need for cables, thereby meeting the 4th requirement.

A list of steps below explains how to setup the Edimax Wi-Fi dongle with the Raspberry Pi.

1. Purchase the **Edimax – Wireless 802.11b/g/n nano USB adapter**. It provides the microcontroller with a small and cheap way to connect to Wi-Fi for plug-and-play applications, because all the necessary drivers come pre-loaded. We purchased ours from Amazon.com for \$10 [39].
2. Add the Wi-Fi adapter to one of the available USB ports on the Raspberry Pi. The only USB devices that the Raspberry Pi needs at this time are the Edimax USB, and a wireless keyboard/mouse dongle.
3. After booting and logging in you want to make sure the Raspberry Pi can find the wireless USB adapter. To look at which peripherals the operating system finds on startup, run the following command within the terminal:

dmesg | more

4. Scroll to the bottom of the page towards the end until similar message appears in the lines:

```
[ 3.339746] usb 1-1.2: new high-speed USB devices number 4 using dwc_otg
[ 3.451892] usb 1-1.2: New USB device found, idVendor=7392, idProduct=7811
[ 3.465994] usb 1-1.2: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[ 3.478450] usb 1-1.2: Product: 802.11n WLAN Adapter
```

This means that the operating system recognized the USB Wi-Fi adapter using one of the built-in drivers (you can return to the terminal by pressing ‘ctrl + z’). Next, we configure the network connection.

5. We use the latest version of Rasbian (updated on September 2014) that comes with all the correct packages pre-installed, so the system should understand all of the following commands, but in case it doesn’t run this command to install them:

sudo apt-get install wpasupplicant wireless-tools

6. Configure the general network settings in “/etc/network/interfaces”, and set the Wi-Fi details in the “/etc/wpa_supplicant/wpa_supplicant.conf” file. First edit the interfaces file:

sudo nano /etc/network/interfaces

7. To configure your wireless network the user might want to modify the file such that it looks like the following Figure 4.6:

```

GNU nano 2.2.6

auto lo

iface lo inet loopback

auto eth0
iface eth0 inet dhcp

allow-hotplug wlan0
auto wlan0

iface wlan0 inet manual
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
iface default inet dhcp

```

Figure 4.6: General network settings located within the terminal

Address any differences as they appear. Press ‘ctrl + x’ to exit nano (press ‘Y’ then ENTER when prompted).

- To get a list of currently available wireless network names, input the following command:

sudo iwlist wlan0 scan|grep ESSID

```

pi@raspberrypi ~ $ sudo iwlist wlan0 scan|grep ESSID
ESSID:"Ultra hyphy machine"
ESSID:"MacNowWiFi"
ESSID:"SeaDog2"
ESSID:"MyCharterWiFi74-2G"
ESSID:"Boxbuilders"
ESSID:"I'M A DRAGON!"
ESSID:"NETGEAR71"
ESSID:"MyCharterWiFi9b-2G"
ESSID:"Steal This"
ESSID:"HP8D9CB9"
ESSID:"HouseOfFlow"
pi@raspberrypi ~ $

```

Figure 4.7: List of available Wi-Fi networks

- Pick a network the user has access to and add the network authentication information in the “wpa_supplicant ” file. For a WPA2-PSK 2.4GHz network protocol, modify the file to read as Figure 4.8:

sudo nano /etc/wpa_supplicant/wpa_supplicant.conf

```

GNU nano 2.2.6

ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1

network={
    ssid="Network_Name"
    proto=WPA2
    key_mgmt=WPA-PSK
    scan_ssid=1
    pairwise=CCMP TKIP
    group=CCMP TKIP
    psk="Password"
}

```

Figure 4.8: Network specifications within the terminal

We provided a list of parameters within the “wpa_supplicant.conf” file and their definitions [44]:

- **ssid:** Wi-Fi name
- **psk:** Wi-Fi password
- **proto:** A list of accepted protocols. Your choice of RSN or WPA. RSN is WPA2 and WPA is WPA1 (also WPA2 can be used as an alias for RSN)
- **key_mgmt:** A list of accepted authenticated key management protocols. Your choice of WPA-PSK or WPA-EAP
- **scan_ssid:** Your choice of either 0 or 1. 0 is default value which doesn’t scan this SSID with specific Probe Request frames. 1 scans the SSID specific Probe Request frame (use this to find Access Points that do not broadcast SSID or use multiple SSID’s; this adds latency to scanning, so enable this only when needed, such as with a mobile hotspot.)
- **pairwise:** A list of accepted pairwise (unicast) ciphers for WPA. Your choice of CCMP or TKIP. If not set, this defaults to CCMP TKIP
- **group:** A list of accepted group (broadcast/multicast) ciphers for WPA. Your choice of CCMP or TKIP. If not set, this defaults to CCMP TKIP

To take the Smart Outlet from place to place and to store the SSIDs and passwords of more than one network, just include however many `network={...}` sections in the “wpa_supplicant.conf” file as the user would like. The network at the top of the file has the highest priority, where subsequent networks below have priority based on their position within the file. The script runs through the Wi-Fi options in a sequential order until connection establishes. If you would like to change the priority of any network then add the following line to the bottom of each network specification:

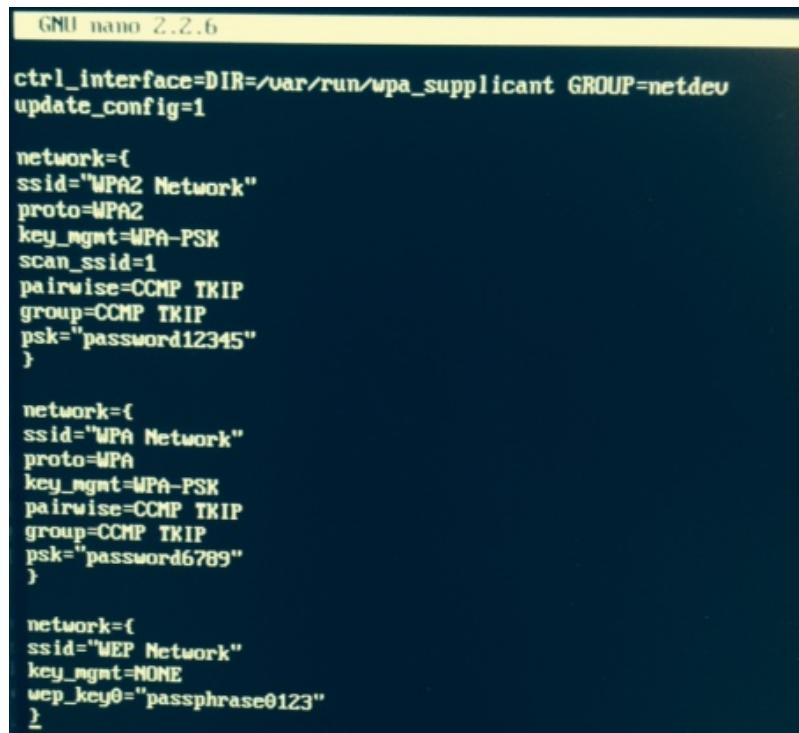
Priority=0 #connects to this network first. Only need to input a single integer for this parameter

Priority=1 #second most important network to connect to

Priority=2 # third

Etc. #and so on...

A sample file looks like Figure 4.9:



```
GNU nano 2.2.6

ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1

network={
    ssid="WPA2 Network"
    proto=WPA2
    key_mgmt=WPA-PSK
    scan_ssid=1
    pairwise=CCMP TKIP
    group=CCMP TKIP
    psk="password12345"
}

network={
    ssid="WPA Network"
    proto=WPA
    key_mgmt=WPA-PSK
    pairwise=CCMP TKIP
    group=CCMP TKIP
    psk="password6789"
}

network={
    ssid="WEP Network"
    key_mgmt=NONE
    wep_key0="passphrase0123"
}
```

Figure 4.9: wpa_supplicant file with multiple types of network connection points

10. Press ‘ctrl + x’ to exit the nano. Save the file by pressing ‘Y’ and then ‘ENTER’. Finally reboot the Raspberry Pi.

sudo reboot

11. After reloading the network, check the status of the wireless connection by typing in the **ifconfig** command. The result should show a valid IP address under the ‘inet addr’, meaning that it connects to the network. The user can check to see which network the wireless adapte uses, by typing in the **iwconfig** command.

```

pi@raspberrypi ~ $ ifconfig
eth0      Link encap:Ethernet HWaddr b8:27:eb:8a:a6:41
          UP BROADCAST MULTICAST MTU:1500 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

lo       Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          UP LOOPBACK RUNNING MTU:65536 Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1104 (1.0 Kib) TX bytes:1104 (1.0 Kib)

wlan0     Link encap:Ethernet HWaddr 80:1f:02:da:4b:d7
          inet addr:172.20.10.3 Bcast:172.20.10.15 Mask:255.255.255.24
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:39 errors:0 dropped:240 overruns:0 frame:0
          TX packets:26 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:10622 (10.3 Kib) TX bytes:3540 (3.4 Kib)
pi@raspberrypi ~ $

```

Figure 4.10: Confirming the link to the network, by looking at the IP address

Check under the wlan0 section to see a valid IP address. Figure 4.10 says 172.20.10.3.

4.4 Automatic Reconnection to Wi-Fi

What happens to the network when a power outage occurs? The router turns off and remains off until power comes back online, usually for an unknown amount of time. This means a drop of network signal. So far, the Smart Outlet can connect to a home network (WPA, WPA2, WEP) on start up, but it cannot reconnect to a Wi-Fi signal without needing to reset the whole system. This means a user has to plug the device back into the wall to restart the whole initialization process, which is only useful once the network comes back online. An easy solution exists for this inconvenient problem.

We borrowed a way for the Raspberry Pi to scan for its previously connected networks, once a signal has been dropped [45]. This method runs through a background application, which automatically reconnects the Raspberry Pi to Wi-Fi when the signal drops.

Type the following steps into the terminal:

Type:

1. sudo mv -i
 /etc/ifplugged/action.d/ifupdown
 /etc/ifplugged/action.d/ifupdown.original

#This replaces one version of ifupdown with another. Ifupdown contains no script, but the script copies to the ifupdown.original file. Ifupdown is an automatic network configuration tool which can only manage Ethernet connections. This step moves unplugged/replugged Ethernet cable issues to ifupdown.original.

2. sudo cp
`/etc/wpa_supplicant/ifupdown.sh`
`./ifupdown`

#This copies a standardized tool for bringing up and down the network to the empty file, ifupdown. It works by restarting the wpa_supplicant daemon when the network is lost, by replacing the default ifupdown script with wpa_supplicant's version.
[\[http://www.denzilferreira.com/rpi-chronicles/\]](http://www.denzilferreira.com/rpi-chronicles/)
3. sudo reboot

#This does a software reset of the system. User can now drop a signal, and the Raspberry Pi attempts to automatically reconnect to it.

4.4.1 Testing reconnection to Wi-Fi

We ran a procedure to test how the Raspberry Pi would pick up a dropped signal. For testing purposes we used a mobile Wi-Fi hotspot supplied by an iPhone 5s using Verizon as a carrier. First, we identified the network name for the hotspot under Settings > General > About (Fig. 4.11). We then began the test by turning on our network before start up (Fig. 4.12).

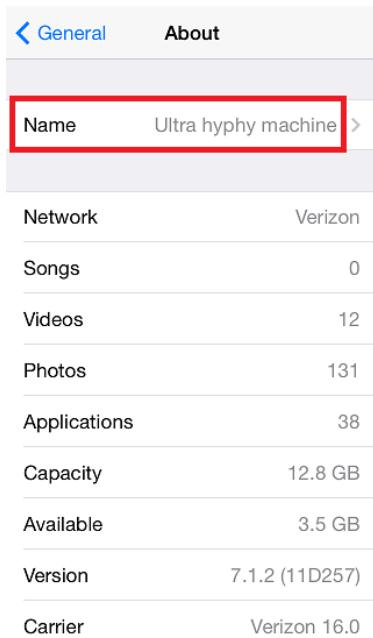


Figure 4.11: Identifying hotspot's name



Figure 4.12: Turning on the hotspot before plugging in the Smart Outlet

Next, we plugged in the Smart Outlet and it ran its initialization process. On startup it immediately detected a Wi-Fi network (Fig. 4.13). We confirmed this by looking on our iPhone, that a link establishes with another device (Fig. 4.14). Both the Raspberry Pi and iPhone confirm a made connection.

```
[ ok ] Setting up console font and keymap...
[ ok ] Setting up X socket directories... /tmp/.X11-unix
INIT: Entering runlevel: 2
[info] Using makefile-style concurrent boot
[ ok ] Network Interface Plugging Daemon.../etc/init.d/networking
[ ok ] Starting enhanced syslogd: rsyslogd.
[ ok ] Starting periodic command scheduler: cron.
[ ok ] Starting system message bus: dbus.
Starting dphys-swapfile swapfile setup ...
want /var/swap=100MByte, checking existing: k
done.
[ ok ] Starting NTP server: ntpd.
[ ok ] Starting OpenBSD Secure Shell server: sshd.
[ ok ] Starting Network connection manager: wpa_supplicant.
My IP address is 172.20.10.3

Raspbian GNU/Linux 7 raspberrypi tty1

raspberrypi login: [REDACTED]
Password: [REDACTED]
Last login: Wed Nov  5 14:51:50 PST 2014 on ttys000
Linux raspberrypi 3.10.25+ #622 PREEMPT Fri Jan 10 14:46:20 PST 2014

The programs included with the Debian GNU/Linux
the exact distribution terms for each program &
individual files in /usr/share/doc/*/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY
permitted by applicable law.
pi@raspberrypi ~ $
```

Figure 4.13: Confirmation on Smart Outlet startup that a wireless connection exists

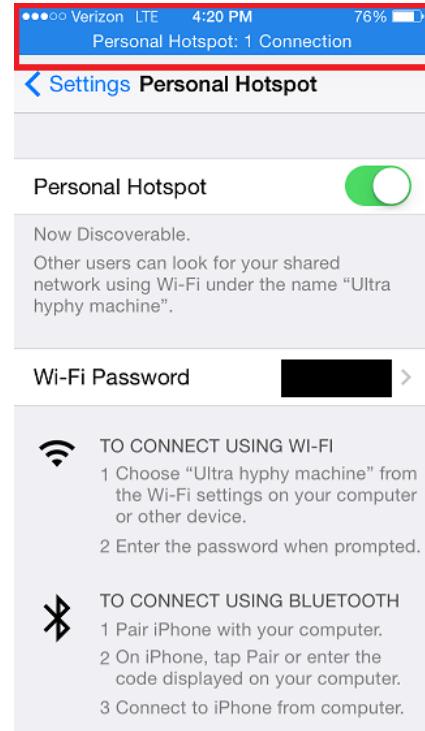


Figure 4.14: iPhone recognizes the Smart Outlet connected to it

We would like to see what networks the Raspberry Pi can also connect to, so we ran the following command:

```
sudo iwlist wlan0 scan|grep ESSID
```

This gave us a list of all 2.4GHz, IEEE 802.11, Wi-Fi networks in range, and can be used by the Raspberry Pi (Fig. 19). The signals which concern us follow the IEEE 802.11bgn protocol, not the IEEE 802.11b protocols, because these are Ad-Hoc networks unsuitable for our purposes, such as those given off by printers. We focus on the WPA2 networks, which follow the IEEE 802.11bgn protocol. Notice our device's name in the red rectangle.

```

RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

lo      Link encap:Local Loopback
        inet addr:127.0.0.1 Mask:255.0.0.0
        UP LOOPBACK RUNNING MTU:65536 Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0
        TX packets:0 errors:0 dropped:0 overruns:0
        collisions:0 txqueuelen:0
        RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

wlan0    Link encap:Ethernet HWaddr 80:1f:02:da:4b:0
        inet addr:172.20.10.3 Bcast:172.20.10.15
        UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
        RX packets:134 errors:0 dropped:12 overruns:0
        TX packets:102 errors:0 dropped:0 overruns:0
        collisions:0 txqueuelen:1000
        RX bytes:21414 (20.9 KiB) TX bytes:12980 (12.6 KiB)

pi@raspberrypi ~ $ sudo iwlist wlan0 scan|grep ESSID
ESSID:"Ultra hyphy machine"
ESSID:"Macnowifi"
ESSID:"SeaDog2"
ESSID:"MyCharterWiFi74-2G"
ESSID:"Boxbuilders"
ESSID:"I'M A DRAGON!"
ESSID:"NETGEAR71"
ESSID:"MyCharterWiFi9b-2G"
ESSID:"Steal This"
ESSID:"HPBD9CB9"
ESSID:"HouseOfflow"

pi@raspberrypi ~ $

```

Figure 4.15: List of available networks the Raspberry Pi can detect

Next, we wanted to simulate a power outage, by dropping our hotspot signal. We disconnected our phone's Wi-Fi, and saw the blue rectangle on the iPhone disappear, showing a disconnected link to the hotspot. We also typed in **iwconfig** into the terminal to check on the status on the Raspberry Pi's end. We confirmed that both devices disconnected from each other. Notice the upper red rectangle (Fig. 4.16). Next we waited a minute and powered back on our hotspot. We noticed that it took about two minutes for reconnection to occur, as seen by the lower red rectangle (Fig. 4.16). This shows that the hotspot's SSID is now used as the source of Wi-Fi.

```

pi@raspberrypi ~ $ iwconfig
wlan0      IEEE 802.11gn ESSID:"Ultra hyphy machine" Nickname:"<WIFI@REALTEK>"
Mode:Managed Frequency:2.437 GHz Access Point: FA:27:93:03:7B:6F
Bit Rate:72.2 Mb/s Sensitivity:0/0
Retry:off RTS thr:off Fragment thr:off
Power Management:off
Link Quality=6/100 Signal level=-7/100 Noise level=0/100
Rx invalid nwid:0 Rx invalid crypt:0 Rx invalid frag:0
Tx excessive retries:0 Invalid misc:0 Missed beacon:0

lo        no wireless extensions.

eth0      no wireless extensions.

pi@raspberrypi ~ $ iwconfig
wlan0      unassociated Nickname:"<WIFI@REALTEK>"
Mode:Managed Frequency=2.437 GHz Access Point: Not-Associated
Sensitivity:0/0
Retry:off RTS thr:off Fragment thr:off
Power Management:off
Link Quality:0 Signal level:0 Noise level:0
Rx invalid nwid:0 Rx invalid crypt:0 Rx invalid frag:0
Tx excessive retries:0 Invalid misc:0 Missed beacon:0

lo        no wireless extensions.

eth0      no wireless extensions.

pi@raspberrypi ~ $ iwconfig
wlan0      IEEE 802.11gn ESSID:"Ultra hyphy machine" Nickname:"<WIFI@REALTEK>"
Mode:Managed Frequency:2.437 GHz Access Point: FA:27:93:03:7B:6F
Bit Rate:72.2 Mb/s Sensitivity:0/0
Retry:off RTS thr:off Fragment thr:off
Power Management:off
Link Quality=100/100 Signal level=94/100 Noise level=0/100
Rx invalid nwid:0 Rx invalid crypt:0 Rx invalid frag:0
Tx excessive retries:0 Invalid misc:0 Missed beacon:0

lo        no wireless extensions.

eth0      no wireless extensions.

pi@raspberrypi ~ $

```

Figure 4.16: Dropping the network and then reestablishing connection, looking at the Raspberry Pi

This procedure confirmed our automatic reconnect application, simulated on other WPA2 networks as well. This simulation can be done for shorter and longer lengths of a pseudo power outage, but each test should give the same results as a real power outage would give. Running this on a different WPA2 network, titled ‘Steal This’ gave similar results, where a network reestablished after about 2 minutes or so of turning back on. Our device can now handle a real power outage.

4.5 Hardware

Solid State Relay

A solid-state relay (SSR) acts as an electrically operated switch controlled with a small signal from the Raspberry Pi. The relay controls the flow of electricity from mains to the load, making it a key component in the Smart Outlet. The three most important characteristics for the SSR in the Smart Outlet are input voltage, output voltage, and maximum load current. In order to function, the relay must have an input within the range of the Raspberry Pi's output, and it must handle a load voltage of 120VAC from mains. Finally, the relay's maximum load current must comply with the 20A maximum current any household device can produce [23].

The solid-state relay made it into the final design after being selected over a mechanical relay. An SSR's lack of moving parts gives it a longer lifetime than that of a mechanical relay. A solid-state relay also operates silently; a major selling point considering our targeted consumer base consists of homeowners that do not want to hear the Smart Outlet while it operates. Finally, a solid-state relay's robustness makes it much less sensitive than a mechanical relay to environmental factors such as humidity, external magnetic fields, and vibrations.

The M-0AC5AH relay satisfies all of the requirements of the application [18]. Its input voltage can accept our microcontrollers 3.3V output, it can pass the voltage from mains, and it can handle up to 25A; more than enough for our Smart Outlet to operate with any household appliance.

The test circuit used in Figure 4.17, below, shows how the relay works with the microcontroller.

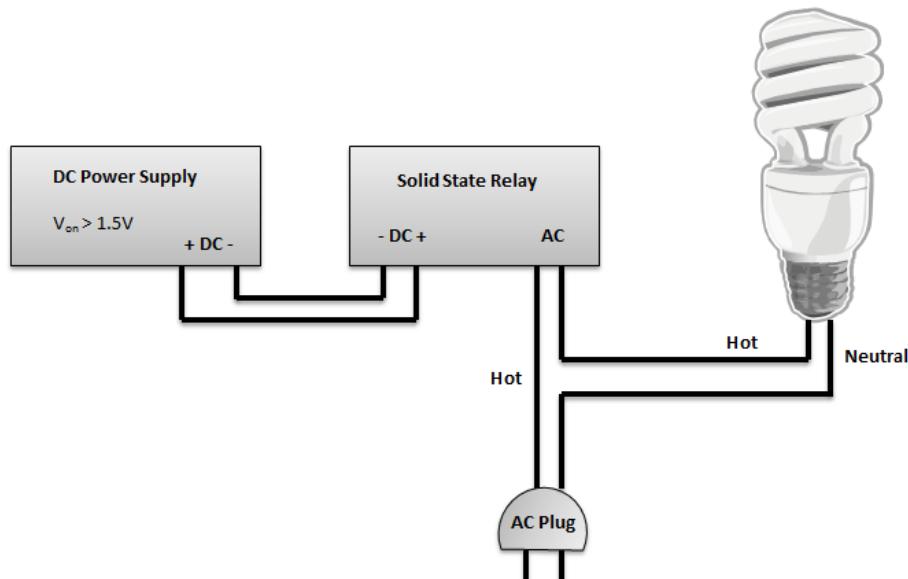


Figure 4.17: Test circuit for relay with light bulb

The minimum V_{on} is 1.5V, which we measured by incrementing the power supply up in 0.1V steps. This works in our current setup because the GPIO pins on the microcontroller can supply 3.3VDC.

Buck Converter

After designing the full-wave rectifier and buck converter to power the Raspberry Pi, we decided to use a Blackberry phone charger instead. Though our designs passed the simulation phase, time constraints did not allow us to wait for the actual hardware to ship. Instead, we found a Blackberry power supply that met our 5V, 700mA spec which gave us a way to power the Smart Outlet right away. Going forward with this project, we may choose to implement the power supply designs to make the Smart Outlet more compact.

Connecting the Raspberry Pi to Multiple Outlets

We wanted to deliver power to the microcontroller and the relays using the same AC source. Following the topology described below, we delivered 5V/700mA to the microcontroller via a Blackberry charger and 120VAC to each relay [47]. The disadvantage to this topology is that current is limited to each outlet, by the demand of each active appliance. If there too many heavy-usage appliances operate together, like those with heating coils or induction motors, the appliances cannot receive the necessary power they need to operate. Figure 4.18, below, shows the schematic used to wire up the hardware.

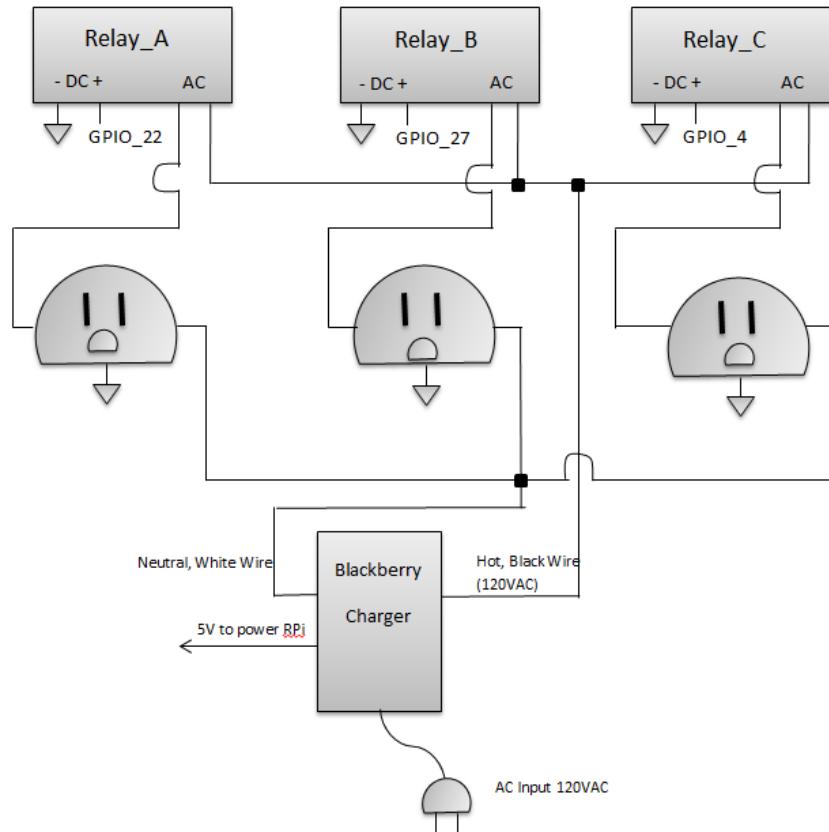


Figure 4.18: Hardware schematic

We ran a simulation of the circuit to test in LT Spice. We used voltage controlled switches for the relays and a pulse governing their operation, as seen in Figure 4.19. The pulse acts as if the GPIO pins supply the DC portion of the relays with voltage to turn them on. A 120VAC sine wave passes through the input and at the output we see an identical sine wave at the load of each switch, as shown in Figure 4.20. This model confirms our plan to use relays to deliver power to the outlets.

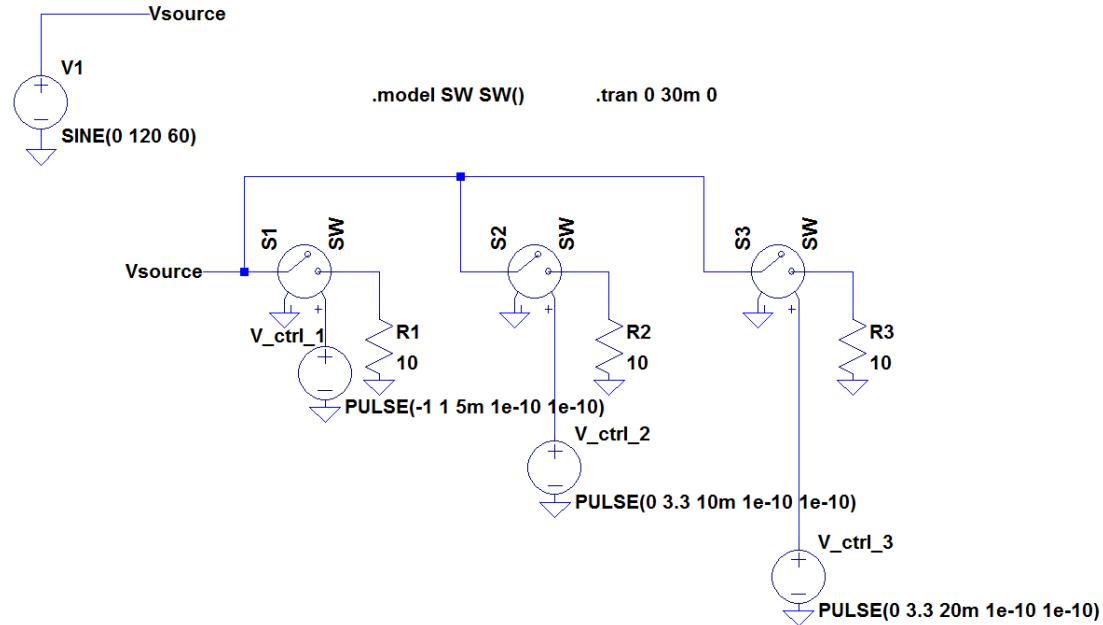


Figure 4.19: GPIO pins triggering individual relays

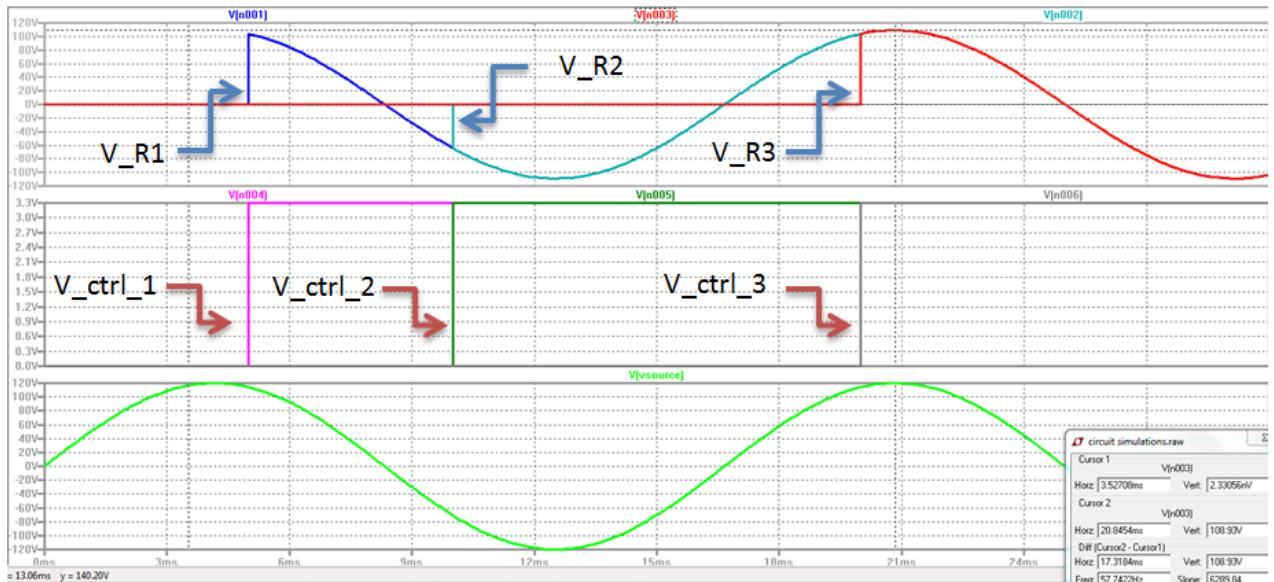


Figure 4.20: Individual relay output characteristics

We used two outlet sockets to model our three different outputs (Outlet_A, Outlet_B, Outlet_C). Each relay independently controls a single outlet, and each has a single output except for A, which has 2 dedicated output ports for that relay. Figure 4.21 shows how we wired the outlets to achieve individual control.

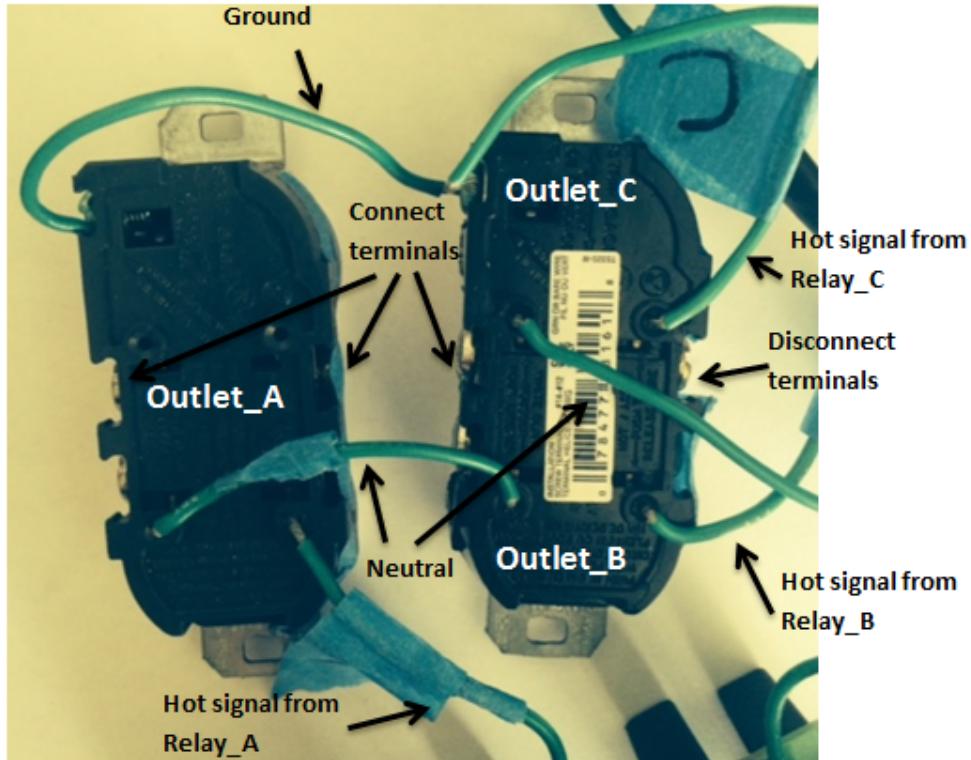


Figure 4.21: Wiring the outlets

The Blackberry charger can deliver 5V of power to the microcontroller. We wired it in such a way that the same power used to power the microcontroller can deliver power from the same source, as shown in Figure 4.22. In this way, we minimized the number of 3-pronged plugs required for this device.

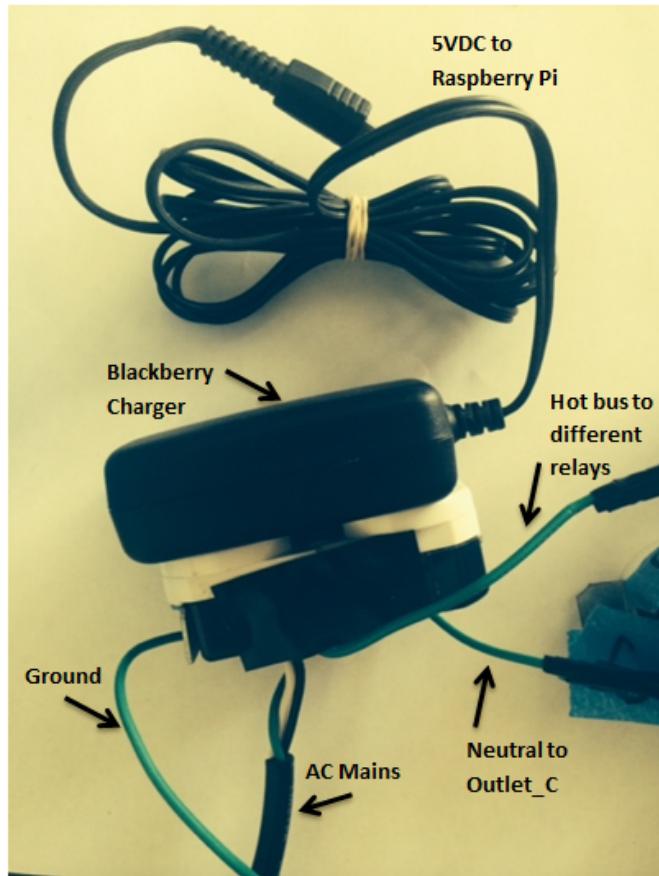


Figure 4.22: Wiring the charger

The relay board design accommodates three different relays and a peripheral for the microcontroller. The board attaches to the bottom of the Raspberry Pi where the relays face away from the board. Figure 4.23 shows the top side of the board while the Figure 4.24 shows the bottom.

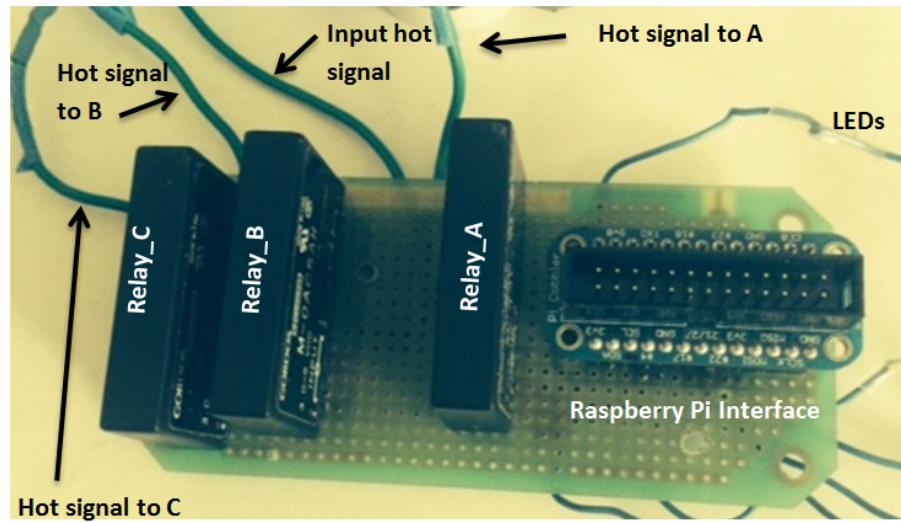


Figure 4.23: Top side of the relay board

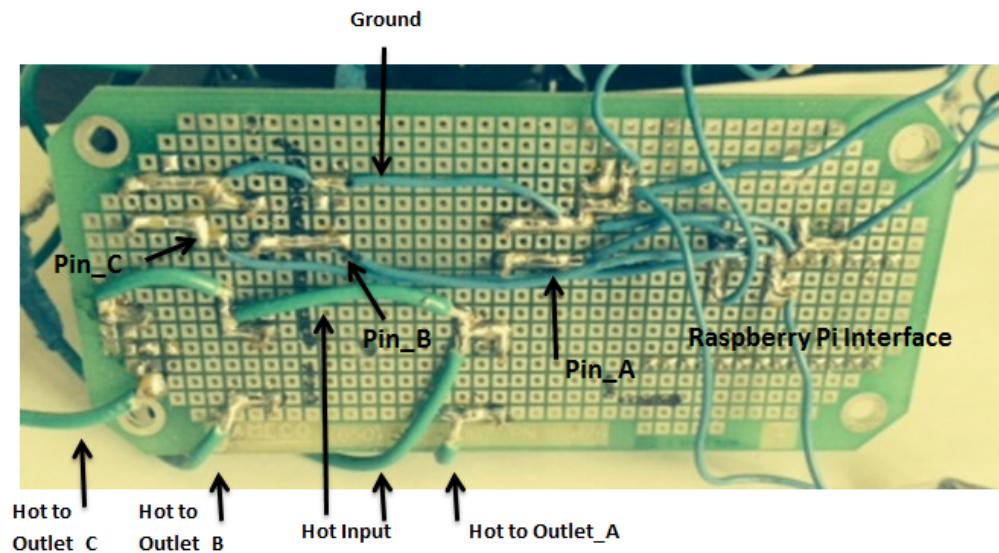


Figure 4.24: Bottom side of the relay board

5 Conclusions

Home automation is a growing field which benefits both businesses and commercial residents alike. Fortunately, we live in a time of great technological speed where this emerging field has quickly caught the attention of the tech industry. In order to make a name for themselves in this new market, several companies each have their own way of establishing peer-to-home connections. Major players in this field such as Belkin, D-Link, INSTEON each provide their own communication protocols of connecting people to their homes. As with all new emerging fields, a vacuum of power has manifested drawing companies to compete against one another until a superior service or product comes to fruition. Ultimately, the consumers control the market and decide on the best approach. Now the market remains fragmented to equally superior companies providing their proprietary take on home automation.

For now, no regulation on the Internet of Things exists, and a standard method of peer-to-home or home-to-home communications has not established itself. It becomes the duty of modern day engineers to create the best possible service for consumers. A standardized method which connects all homes under a communal network of synchronous sprinkler systems, door locks, and heating systems can all benefit safer and energy efficient communities. People have predicted that a standard way to connect to homes, office buildings, and other structures, lurks right around the corner. Consumers might just decide on way to communicate with their homes in a specialized way, just separate protocols exist for communicating in different aspects of life. SMS over phones, emails for business, and Twitter accounts for social media illustrate separate services that many people choose to subscribe to, but serve as the most convenient choice for communicating in that aspect of life. Home automation is still searching for that kind of portal of communication, but this new field has not had as much time for consumers to experiment with. A consumer driven choice filters out the weak ideas from the strong ones and provides a way that offers flexibility, freedom, and control. Rather than entering the market competition amongst other companies, our goal with the Smart Outlet provides a different communication solution, intuitive for users to learn. Our project aims at providing a different approach using a multi-platform calendar interface and existing Wi-Fi networks, which might lead to standardized way of controlling homes.

The Smart Outlet benefits users by providing them with multiple outlets per single unit. It also uses a graphical user interface that many people already use, which allows them to immediately feel familiar with the device. A calendar interface provides scheduling options as well as up to the minute operation. Multiple platforms such as tablets, phones, and desktops enable convenient access and modification to events. The Smart Outlet can even save on electricity costs too, by turning off power to charging devices. The Smart Outlet provides home security as well, as lamps can remotely turn on and off at unpredictable times. While these are great advantages for someone to own a Smart Outlet, the real reason behind this project was for convenience purposes. The inspiration for the Smart Outlet came during one winter night when a college student came home to a freezing dormitory room. A space heater would anticipate the student's arrival so he could come to a pre-warmed room without needing to run the space heater all day. This inspired the Smart Outlet, and with it came the idea of an automated power strip.

A prototype of the Smart Outlet, shown in Figure 5.1, gives an example of the availability of home automation at a consumer level, but by no means is this version final and ready for market. While the Smart Outlet performs to the specifications in the *Requirements and Specifications* section, it merely provides a proof-of-concept of an automated power strip and a different take on interfacing software with hardware. It is able to provide up to 20A of distributed power and it can operate using WPA2, WPA, and WEP Wi-Fi networks. Google Calendar provides an excellent user interface with a wait time of maximum 1 minute updates for events. Status LEDs indicate when individual relays are on. The module operates with 3 independently controlled outlets and 1 slave outlet. An ABS plastic box measuring 8"x4"x4" encloses the whole system. A single AC cord powers the entire device.



Figure 5.1: The Smart Outlet prototype

Future design considerations for the Smart Outlet include a manual override and an energy monitoring system in place. These features already exist in the current competition, so they are not completely novel features to add. A manual override provides the user with an option to physically control the Smart Outlet instead of relying on the web. An energy monitoring system predicts the user's monthly energy bill and notifies them of how their events in place can affect their monthly electricity bill. With these additional features in place, the Smart Outlet can catch up to the leading competition. Cost remains an issue, as this device needs heavy reduction in component pricing in order to compete with existing models. By specializing components and reducing the features in several components, such as with the phone charger and Raspberry Pi, we can reduce costs to make the pricing more competitive.

References

- [1] Wang, Walter. "Home Automation Dominates Internet of Things Conversation - CleanTechies." *CleanTechies*. N.p., 13 July 2014. Web. 07 Dec. 2014.
<<http://cleantechies.com/2014/07/13/home-automation-dominates-internet-of-things-conversation/>>.
- [2] Moorhead, Patrick. "The Problem With Home Automation's Internet Of Things (IoT)." *Forbes*. Forbes Magazine, 26 Sept. 2013. Web. 07 Dec. 2014.
<<http://www.forbes.com/sites/patrickmoorhead/2013/09/26/the-problem-with-home-automations-iot/>>.
- [3] Gray, Patrick. "Five Apps That Will Help Automate Your Home." *TechRepublic*. N.p., 4 Aug. 2014. Web. 07 Dec. 2014. <<http://www.techrepublic.com/blog/five-apps/five-apps-that-will-help-automate-your-home/>>.
- [4] Nunes, Renato Jorge Caleira. "Home Automation - A Step Towards Better Energy Management." *IST – Technical University of Lisbon* (n.d.): 1-7. Web. 7 Dec. 2014.
<<http://www.icrepq.com/pdfs/CALEIRA416.PDF>>.
- [5] "Phantom Load." - *Waverly Light and Power*. Waverly Light and Power, 2014. Web. 07 Dec. 2014. <<http://waverlylp.com/residential/energy-efficiency-opportunities/phantom-load.aspx>>.
- [6] Pierce, Mark. "Watch Out For Vampire Appliances." *Cornell Chronicle* (2002): 1-3. 12 Sept. 2002. Web. 7 Dec. 2014. <<http://www.human.cornell.edu/dea/outreach/upload/watch-out-for-vampires.pdf>>.
- [7] "WEMO + IFTTT." *Belkin Wemo*. Belkin, 2014. Web. 07 Dec. 2014.
<<http://www.belkin.com/us/wemo/ifttt/>>.
- [8] "WeMo® Insight Switch." *Belkin Wemo*. Belkin, 4 Sept. 2014. Web. 07 Dec. 2014.
<<http://www.belkin.com/us/p/P-F7C029/>>.
- [9] "Wi-Fi Smart Plug DSP-W215." *D-link*. D-Link US, 31 May 2014. Web. 07 Dec. 2014.
<<http://us.dlink.com/products/connected-home/wi-fi-smart-plug/>>.
- [10] INSTEON. *INSTEON - On/Off Module*. N.p., Web. 20 Mar. 2014.
<<http://www.insteon.com/2635-222-on-off-module.html>>.

- [11] "Wi-Fi SMART PLUG (DSP-W215)." *Wi-Fi SMART PLUG (DSP-W215)* (n.d.): n. pag. *Dlink.com*. D-Link, 28 Feb. 2013. Web. 28 Sept. 2014.
- [12] "Securifi SZ-ESW01." *Wiki.securifi.com*. Ed. Lars. Securifi Wiki, 11 June 2014. Web. 28 Sept. 2014. <http://wiki.securifi.com/index.php?title=Securifi_SZ-ESW01>.
- [13] "Welcome to Google Calendar." *Support.google.com*. Google, 1 Dec. 2014. Web. 07 Dec. 2014. <<https://support.google.com/calendar/answer/2465776?hl=en>>.
- [14] "Create a Repeating Event." *Support.google.com*. Google, 1 Dec. 2014. Web. 07 Dec. 2014. <<https://support.google.com/calendar/answer/37115?hl=en>>.
- [15] Miller, Charles R. "Box Fill Calculations: Article 314: Boxes; Conduit Bodies; Fittings and Manholes." *Box Fill Calculations: Article 314: Boxes; Conduit Bodies; Fittings and Manholes*. Electrical Contractor, July 2004. Web. 07 Dec. 2014. <<http://www.ecmag.com/section/codes-standards/box-fill-calculations-article-314-boxes-conduit-bodies-fittings-and-manholes>>.
- [16] Beal, Vangie. "802.11 IEEE Wireless LAN Standards." *Webopedia.com*. Webopedia, 20 Aug. 2014. Web. 07 Dec. 2014. <http://www.webopedia.com/TERM/8/802_11.html>.
- [17] "Raspberry Pi FAQs." *Raspberrypi.org*. Raspberry Pi, n.d. Web. 7 Dec. 2014. <<http://www.raspberrypi.org/help/faqs>>.
- [18] "2.75-8VDC Solid State Relay." *Jameco.com*. Jameco Electronics, 2009. Web. 07 Dec. 2014. <http://www.jameco.com/webapp/wcs/stores/servlet/Product_10001_10001_2179462_1>.
- [19] Poole, Ian. "MIMO Formats - SISO, SIMO, MISO, MU-MIMO." *Radio-electronics.com*. Radio-Electronics.com, n.d. Web. 07 Dec. 2014. <<http://www.radio-electronics.com/info/antennas/mimo/formats-siso-simo-miso-mimo.php>>.
- [20] Deziel/ Demand Media, Chris. "How Many Outlets Can Be Placed on a 20 Amp Household Circuit?" *Home Guides*. SF Gate, 6 Dec. 2014. Web. 07 Dec. 2014. <<http://homeguides.sfgate.com/many-outlets-can-placed-20-amp-household-circuit-82633.html>>.
- [21] Ada, Lady. "Transformer-based AC/DC Converters." *Adafruit.com*. Adafruit, 7 Dec. 2014. Web. 07 Dec. 2014. <<https://learn.adafruit.com/power-supplies/transformer-based-ac-slash-dc-converters>>.

[22] "Notifications." *Support.google.com*. Google, 2014. Web. 07 Dec. 2014. <<https://support.google.com/calendar/answer/37242?hl=en>>.

[23] "Solid-state Relays." *Omron.com*. Omron, 1 Oct. 2012. Web. 07 Dec. 2014. <<http://www.ia.omron.com/support/guide/18/overview.html>>.

[24] "Arduino - ArduinoBoardUno." *Arduino.com*. Arduino, 2014. Web. 04 Dec. 2014. <<http://arduino.cc/en/Main/arduinoBoardUno>>.

[25] "BeagleBone Black." *Beagleboard.org*. Jkridner.wordpress.com, 23 Nov. 2014. Web. 07 Dec. 2014. <<http://beagleboard.org/BLACK>>.

[26] Leonard, Michael. "How to Choose the Right Platform: Raspberry Pi or BeagleBone Black?" *MAKE.com*. Makezine, 25 Feb. 2014. Web. 07 Dec. 2014. <<http://makezine.com/magazine/how-to-choose-the-right-platform-raspberry-pi-or-beaglebone-black>>.

[27] "Getting Started with Twitter via Your Mobile Phone." *Twitter Help Center*. Twitter, 2014. Web. 07 Dec. 2014. <<https://support.twitter.com/articles/14589-getting-started-with-twitter-via-your-mobile-phone>>.

[28] Cooking Hacks. "GPRS GSM SIM900 Module for Arduino and Raspberry Pi." *Amazon.com*. Amazon, 2014. Web. 07 Dec. 2014. <<http://www.amazon.com/GPRS-SIM900-Module-Arduino-Raspberry/dp/B00F2HDJBI>>.

[29] Sklar, Mikey. "Raspberry Pi E-mail Notifier Using LEDs." *Adafruit.com*. Adafruit, 7 Dec. 2014. Web. 07 Dec. 2014. <<https://learn.adafruit.com/raspberry-pi-e-mail-notifier-using-leds/overview>>.

[30] "PuTTY Download Page." *Putty.org*. N.p., 6 Nov. 2014. Web. 07 Dec. 2014. <<http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>>

[31] Zagros Robotics. "Configuring the Raspberry Pi Ethernet Port for Remote Control." *Instructables.com*. N.p., 27 May 2013. Web. 07 Dec. 2014. <<http://www.instructables.com/id/Configuring-the-Raspberry-Pi-ethernet-port-for-rem/>>.

[32] Michael. "Easily Connect Raspberry Pi to Gmail, Facebook, Twitter & More!" *Mitchtech.net*. MitchTech, 23 Aug. 2013. Web. 07 Dec. 2014. <<http://mitchtech.net/connect-raspberry-pi-to-gmail-facebook-twitter-more/>>.

[33] "Google Calendar API." *Developers.google.com*. Google, 17 Nov. 2014. Web. 07 Dec. 2014. <<https://developers.google.com/google-apps/calendar/>>.

[34] "OpenSprinkler Pi (OSPi) An Open-Source Sprinkler / Irrigation Extension Board for RPi." *Rayshobby.net*. N.p., 13 Apr. 2013. Web. 07 Dec. 2014. <http://rayshobby.net/?page_id=5816>.

[35] "Google Calendar API V2 Developer's Guide: Python." *Developers.google.com*. Google, 28 Nov. 2014. Web. 07 Dec. 2014. <https://developers.google.com/google-apps/calendar/v2/developers_guide_python>.

[36] "Opensprinkler/OpenSprinkler Pi/software/demos/gcalendar/." *GitHub.com*. Rayshobby Shop, 13 Mar. 2013. Web. 07 Dec. 2014. <<https://github.com/rayshobby/opensprinkler/tree/master/OpenSprinkler%20Pi/software/demos/gcalendar>>.

[37] "Client Download." *Filezilla- The Free FTP Solution*. N.p., n.d. Web. 07 Dec. 2014. <<https://filezilla-project.org/download.php?type=client>>.

[38] "Linux Commands." *Raspberrypi.org*. Raspberry Pi Documentation, n.d. Web. 07 Dec. 2014. <<http://www.raspberrypi.org/documentation/linux/usage/commands.md>>.

[39] "150Mbps Wireless IEEE802.11b/g/n Nano USB Adapter EW-7811Un." *Edimax.com*. EDIMAX Technology, 13 May 2014. Web. 07 Dec. 2014. <http://www.edimax.com/edimax/merchandise/merchandise_detail/data/edimax/global/wireless_adapters_n150/ew-7811un>.

[40] "Google Calendar API Version 3.0 Migration Guide." *Developers.google.com*. Google, 19 Nov. 2014. Web. 07 Dec. 2014. <<https://developers.google.com/google-apps/calendar/migration>>.

[41] Malinen, Jouni. "Linux WPA Supplicant (IEEE 802.1X, WPA, WPA2, RSN, IEEE 802.11i)." *W1.fi*. N.p., 12 Jan. 2013. Web. 07 Dec. 2014. <http://w1.fi/wpa_supplicant/>.

[42] Sims, Gary. "How to Set up WiFi on a Raspberry Pi - Make Tech Easier." *Maketecheasier.com*. Make Tech Easier, 27 Jan. 2014. Web. 07 Dec. 2014. <<http://www.maketecheasier.com/setup-wifi-on-raspberry-pi/>>.

[43] "How-To: Add WiFi to the Raspberry Pi | Raspberry Pi HQ." *Raspberrypihq.com*. Raspberry Pi HQ, 5 July 2014. Web. 07 Dec. 2014. <<http://raspberrypihq.com/how-to-add-wifi-to-the-raspberry-pi/>>.

[44] "Example Wpa_supplicant Configuration File." *W1.fi*. N.p., n.d. Web. 07 Dec. 2014. <http://w1.fi/cgit/hostap/plain/wpa_supplicant/wpa_supplicant.conf>.

[45] Ncdownpat. "How to Automatically Reconnect WiFi?" *Stackexchange.com*. Stack Exchange, 9 Mar. 2013. Web. 07 Dec. 2014. <<http://raspberrypi.stackexchange.com/questions/4120/how-to-automatically-reconnect-wifi>>.

[46] BlackBerry. "BlackBerry Folding Blade Micro-USB Charger for BlackBerry 8900, Storm 9530, Tour 9630, and Torch 9800 - Black." *Amazon.com*. Techno Discounters USA, n.d. Web. 07 Dec. 2014. <<http://www.amazon.com/BlackBerry-Folding-Blade-Micro-USB-Charger/dp/B001LRPN0S>>.

[47] R. Ford and C. Coulston, *Design for Electrical and Computer Engineers*, McGraw-Hill, 2007, p. 37

[48] "X10 Commander." *Melloware.com*. Melloware, 2012. Web. 12 Dec. 2014. <<http://melloware.com/x10commander>>.

[49] "X10 TM751 Compact X10 Plug-In RF Receiver Base." *Smarthome.com*. X10 Powerhouse, 06 May 2014. Web. 12 Dec. 2014. <<http://www.smarthome.com/x10-tm751-compact-x10-plug-in-rf-receiver-base.html>>.

[50] "X10 SR227 or PA011 X10 Split Receptacle." *Smarthome.com*. N.p., 31 Mar. 2014. Web. 12 Dec. 2014. <<http://www.smarthome.com/x10-sr227-or-pa011-x10-split-receptacle.html>>.

[51] "X10 AM466 or PAM02 X10 3-Pin Appliance Module." *Smarthome.com*. X10 Pro, 13 July 2014. Web. 12 Dec. 2014. <<http://www.smarthome.com/x10-am466-or-pam02-x10-3-pin-appliance-module.html>>.

[52] "What Is X10 Home Automation, How Does X10 Work, and What Are the Limitations of X10?" *Cleverhome.com*. Cleverhome, 12 Dec. 2014. Web. 12 Dec. 2014. <<http://www.cleverhome.com.au/X10-home-automation.shtml>>.

Appendix A: Senior Project Analysis

Functional Requirements

The Smart Outlet wirelessly connects to Google Calendar and uses scheduled events to deliver power to simple appliances. The Smart Outlet acts like a switch for the outlet and it relays power from mains to power up to three simple appliances connected to it. In addition, Google Calendar provides a user friendly, universal platform for customers to access from their phone, tablet, or computer. The calendar's simplicity allows for easy communication with the Smart Outlet as long as the user and the outlet establish a network connection. Refer to Chapter 2 for a full list of specifications.

Primary Constraints

The Smart Outlet must have network access in order for Google Calendar to interact with it. A Raspberry Pi must be configured to check Google Calendar and communicate with the scheduled events. The solid-state relays must accept 120VAC from mains and a 3.3V control signal from the microcontroller. In order to communicate with the Smart Outlet the user must have access to Google Calendar from a smart phone, tablet, or computer. Finally, the user needs a Gmail account setup for the Smart Outlet to access the calendar.

Economic

Human Capital: The Smart Outlet requires laborers at manufacturing facilities to produce the parts necessary for completion. The project also required us, as designers, in order to complete.

Financial Capital: Components purchased for the Smart Outlet from parts manufacturers require compensation through Cal Poly. The future of the Smart Outlet aims to save power, thus saving customers money on utility bills.

Manufactured or Real Capital: The Smart Outlet contains several electrical components including a microcontroller, solid state relays, resistors, and LEDs.

Natural Capital: The Smart Outlet contains individual components which use up natural resources, including copper and silicon.

Project Costs:

The project costs the sum of its components in addition to utilities. Table A-1 breaks down the various costs of the project as well as the estimated labor cost.

Using the equation from Ford & Coulston, Ch.10 [47] allows us to estimate costs:

$$Cost = \frac{OptimisticCost + 4 * RealisticCost + PessimisticCost}{6}$$

Table A-1: Cost Estimation

Labor:		Price
Research and Manufacturing		\$2700
Utilities:		
Electricity (\$2/month for 9 months)		\$18
Parts Cost (Quantity):		
Raspberry Pi Model B (1)		\$35
Raspberry Pi Power Supply (1)		\$2
SD Memory Card (1)		\$8
PCB Breakout Headers + Ribbon Cable (1)		\$8
Solid State Relay (3)		\$24
Prototype Board (1)		\$6
Outlets (4)		\$8
LEDs (3)		\$0.50
Power Cord (1)		\$4
Wi-Fi Adapter (1)		\$10
Enclosure (1)		\$6
Hardware (Screws, nuts, wire, standoffs, solder)		\$10
Total		\$2839.50

Timing:

Completion time for the project took 3 quarters (Winter 2014, Spring 2014, Fall 2015) in addition to summer. Figure A-1 shows the initial project development time while Figure A-2 displays the actual timing of the project plan.

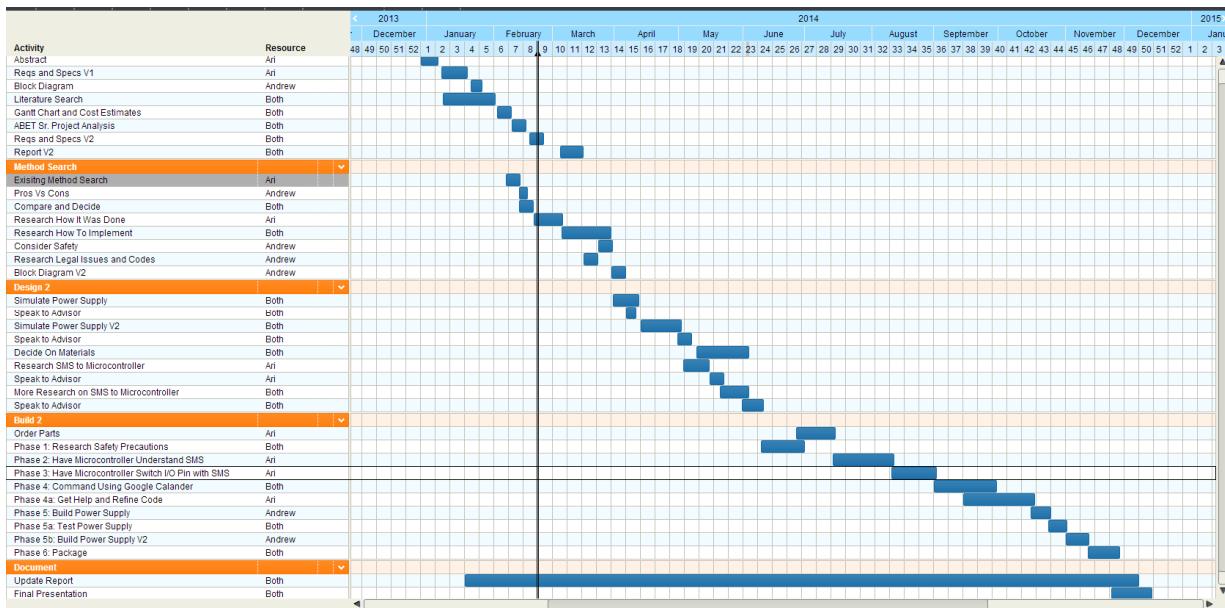


Figure A-1: Initial project plan Gantt Chart

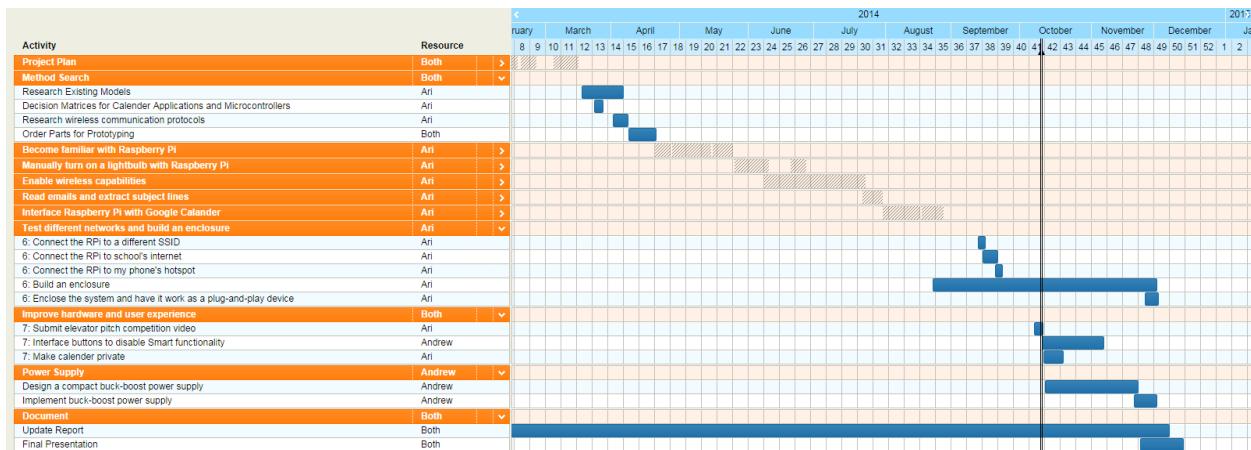


Figure A-2: Adjusted project plan Gantt Chart

If manufactured on a commercial basis

Assuming we sell 5 Smart Outlets per year, the cost per unit of \$120 plus a manufacturing price and a mark-up of \$50 makes the Smart Outlet retail for \$170. Therefore we profit by \$250 per year excluding labor costs. The consumer should not pay anything in addition to this, since the cost includes all the software and hardware. From out of the box, the user only needs 30 minutes to implement the device and set it up for wireless communication. Future designs of the Smart Outlet call for buttons that manually override each individual outlet's status, enabling the consumer to use the Smart Outlet in the event of a compromised Wi-Fi connection.

Environmental Impacts [47]

The Smart Outlet has two major environmental impacts. The first comes from the materials used in the components of the outlet. Silicon, copper, and ceramic materials require energy to mine and even more energy to be transported to various locations and manufacturing plants across the globe. This results in air pollution, particularly from the ships used to transport the raw goods. The second impact comes from the consumer, and has a smaller environmental impact. The energy used from the grid while the Smart Outlet remains on and any pollution created by the appliance at the load directly affects the environment after device manufacture. The future of the Smart Outlet aims to increase its efficiency, thus saving power and having a more positive impact on the environment.

Manufacturability [47]

Manufacturing small components involves soldering the components in an organized and compact fashion, as well as drilling holes for mounting hardware. As production increases, a shift from manual labor to automated technology will occur. While expensive, this method has a

higher yield while keeping reliable results. Each smart outlet requires testing after assembly ensuring proper function.

Sustainability

Maintaining the device depends on how often the user operates it and the kind of appliance hooked up to the outlet. The SD card on the Raspberry Pi may need an upgrade if the device is on continuously over an extended period of time. Overall the Smart Outlet has a very long lifespan as the most important piece, the microcontroller, consumes very little power and stays cool even when operated over long periods of time.

To maintain the system, a constant flow of electricity must power the microcontroller from the wall outlet. In addition, the system keeps heat to a minimum and its overall efficiency doesn't fall below 90% for sending current to appliances. This device saves electricity for the user by turning off appliances when unused. A future model includes a manual override, so the user can interact with the Smart Outlet by pressing buttons on the device, instead of only using wireless commands. This upgrade does not include much more processing power but more component costs.

Ethical

The Smart Outlet should disclose to the public that it has the potential to deliver an electric shock if handled improperly [IEEE Code of Ethics #1]. It connects electronics to a wall outlet driven by a power supply designed by undergraduate students; therefore a warning must appear on the exterior of the Smart Outlet. In addition, some already-made components have to be purchased from manufacturers; such as a DC-to-DC power supply and a microcontroller unit. Credit has been given to the developers of these devices [IEEE Code of Ethics #2]. This project only reports claims realistic to the data; meaning that only empirical data should describe what the Smart Outlet can achieve [IEEE Code of Ethics #3]. We reject bribery of all forms [IEEE Code of Ethics #4]. We improve the understanding of home automation technology by researching a different way to communicate to microcontrollers and documenting the consequences [IEEE Code of Ethics #5]. One important ethical rule, The Golden Rule, makes these criteria possible.

The Golden Rule says one should treat others how oneself would like to be treated. The Smart Outlet benefits others in a way that grants them full control over their appliances when they cannot physically reach them. We would like to have the Smart Outlet in our homes and, therefore, believe that more people would like this product as well.

Health and Safety [47]

This product should not injure anyone so a surge protector feature makes sure that current does not travel back into the outlet or outside the enclosure of the device. Since it deals with high voltages, two layers of insulation wrapped around all live wires within the device eliminate the risk of shock [16]. This complies with safety regulation UL1449.

Social and Political [47]

Table A-2: Direct and Indirect Stakeholders of the Smart Outlet

Direct Stakeholders	Indirect Stakeholders
<ul style="list-style-type: none">• Parts manufacturers• Parts developers• Our advisor• Andrew Godsil• Ari Dubinsky• FCC• Other safety regulators• Homeowners	<ul style="list-style-type: none">• Electric Company• Patent Approvers• Fabricators/ Manufactures• Delivery Services• Cal Poly Facilities• Classmates• EE Department faculty• Late-Night Restaurants• Families and Friends of device users• YouTube• Research Tools

By offering convenience and some means of saving power, all stakeholders in the Smart Outlet benefit. Notable stakeholders include homeowners, as the Smart Outlet has the potential to save them money on utility bills while also providing its service. Part manufacturers benefit financially the more the Smart Outlet sells. Most of our stakeholders come from the California community, but manufacturers and delivery services are international and require the greatest amount of capital to access their resources. International parts manufacturers may not have the ethical standards on par with domestic manufacturers. Should Smart Outlet production begin to grow, we would have to consider the ramifications of buying parts from foreign companies and possibly start buying from domestic companies.

Development

Developing this project required research in design techniques, wireless communication, power management, and Python programming and has taught us invaluable skills in these areas. Becoming accustom to the features and functionality of the Raspberry Pi required various tutorials. The Reference section above details our literature search. The future of the Smart Outlet involves more efficient designs that save consumers power and buttons that toggle the individual outlet statuses, ensuring the outlets integrity should the connected network go down.